Leveraging Structure and Context for Language-Adjacent Representation Learning

Nikita Srivatsan

CMU-LTI-24-004

April 4th, 2024

Language Technologies Institute
School of Computer Science
Carnegie Mellon University
5000 Forbes Ave., Pittsburgh, PA 15123
`www.lti.cs.cmu.edu`

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Language and Information Technologies.

Dedicated to SOPHIE (1986-2021)

Abstract

When learning representations from large corpora of language data, the overwhelming strategy is to interpret that data as a collection of IID samples to be modeled in isolation from one another. While this approach is in some ways beneficial as it allows for efficient training via SGD and doesn't rely on metadata that may not always be present, it does come with limitations. Taking advantage of more complex structural links between individual datapoints can let information flow within our corpora, making learned representations more context-sensitive, and allowing for heavier parameter sharing to more easily generalize to examples from unseen class types. In this work, we will apply this idea to a variety of settings — largely those that lie at the boundary between language and other modalities — for which much of the existing prior work has not explicitly made use of observable structure within the data, and also show how we can add useful inductive bias to our models through lower-level modeling choices. In order to retain interpretability and control we will do this both using probabilistic variational learning frameworks, and also non-variational approaches such as checklist models and retrieval guided generation.

This dissertation is organized into three parts which apply this broad theme to various specific applications. First we'll examine the task of learning disentangled representations of style and structure in digital fonts, and then apply similar modeling ideas to the task of analyzing handwriting styles of scribal hands of Linear B. In the next part we'll investigate ways to learn contextualized representations for temporally ordered data where predictions for one datapoint may influence nearby predictions, such as piano fingering estimation and discursive topic modeling on social media. Finally we'll put forward new approaches for settings where the input signal is itself multimodal, such as the task of writing descriptive captions for images and music.

Acknowledgments

This thesis would not have been close to possible for me to complete alone, and over the years I spent working on it I have relied heavily on the collaboration, guidance, support, and friendship of a great many people. It would be challenging if not impossible for me to sufficiently and exhaustively thank everyone who has helped me throughout this process in text here, but I will nonetheless attempt to. Some of you continue to be supportive presences in my life, some of you were simply there when I needed you, but for all I am eternally grateful.

First and foremost I would like to thank my advisor Taylor Berg-Kirkpatrick for being such a fantastic mentor. Thank you for being a source of inspiration to me, for helping me find and maintain my footing in such an unstable and constantly evolving field, and encouraging me to pursue above all else research that I truly felt excited to work on. Thank you to the rest of my committee, Yonatan Bisk, Matt Gormley, and Julian McAuley for your intellectual contributions to this work, and for challenging me to put forward the best version of this thesis that I could. Thank you also to Jason Eisner, Suchi Saria, and Yair Amir for giving me opportunities in research as a mere undergrad, and encouraging me to continue pursuing it as a grad student.

I must also acknowledge the immense contributions both to the text of this document and the conceptual and empirical work that it represents from all of my co-authors and collaborators: Zachary Wojtowicz, Jon Barron, Dan Klein, Jason Vega, Christina Skelton, Laurel Selvig, Si Wu, Sofia Samaniego, Omar Florez, Danlu Chen, Zachary Novack, Ke Chen, and Shlomo Dubnov.

Thank you to all my other labmates both current and former for being such a joy to work alongside, and for your always valuable advice and mentorship. In particular: Daniel Spokoyny, Yasaman Jafari, Maria Ryskina, Volkan Cirik, Harsh Jhamtani, Jack Wieting, Tim Viera, Dingquan Wang. I am so lucky to be able to call some of you not simply my coworkers but close friends. You have been by my side during late nights that turned into early mornings, and through uncountable adventures over these many years. The thought of your company was consistently the reason I left my apartment each day: Nikolai Vogler, Kartik Goyal, and Niloofar Mireshghallah. I would also like to thank the many wonderful friends who I met outside the walls of this lab through the broader CMU and UCSD communities, who have continually been such a positive presence in my life: Sam Maskell, Arnav Kumar, James Route, and Michael B. James.

My life as an academic would not be sustainable without the love, friendship, and support of the many vibrant people in my life who have for reasons I still struggle to understand shown me affection and community outside that bubble, in particular: Nat, Be, Dylan, and Messi. Thank you for helping me grow into the person I am, and for giving me permission to always be myself.

To all the medical professionals who assisted me through the seemingly insurmountable task of transitioning as a grad student, I would not be alive today without you: Erin Walsh, Karen McCowen, Mike Hsieh, Justine Lee, Abie Mendelsohn, and Abigail Weissman.

Thank you finally to my family for their emotional, financial, and moral support, without whom I would never have had the opportunities presented to me in this life. To my Amma and Appa, aunts and uncles, cousins, grandparents, and above all my steadfast sibling Ashwath, thank you for being there for me.

Contents

Chapter 1

Introduction

1.1 Representation Learning

Representation learning is a broad sub-field within machine learning that is primarily concerned with learning dense, and ideally low-dimensional embeddings of more complex and potentially structured data. These embeddings ideally retain and expose key high-level features necessary for performing downstream tasks, while discarding lower-level noise. Typically this is done in an unsupervised or semi-supervised way, with the aim being that the representations learned are task-agnostic and therefore applicable to a variety of more specific downstream applications, perhaps being additionally fine-tuned at a later supervised training stage. Historically this has spanned a variety of methods from dimensionality reduction techniques like PCA (Jolliffe and Cadima, 2016) to topic models like LDA (Ng and Jordan, 2003). The advent of dense pretrained word embeddings (Mikolov et al., 2013) also paved the way for RNNs and in turn neural language models more broadly. In parallel, autoencoders parameterized by CNNs saw massive popularity for vision tasks (Vincent et al., 2008). These would eventually give way to contrastive methods like SimCLR (Chen et al., 2020) and CLIP (Radford et al., 2021). While these methods will prove useful for our purposes, our interest lies less in designing new losses, and more on the models themselves and their parameterization. On that note, a large amount of work in the space of representation learning has focused on probabilistic graphical modeling. Most notable among these is the Variational Autoencoder (Kingma and Welling, 2014), which established the basic paradigm of treating a hidden embedding as a latent variable, with the encoder and decoder networks serving as parameterizations for a posterior and conditional likelihood distribution respectively. Diffusion models — an approach that extends this framework in a more iterative direction — have seen unprecedented success across a variety of domains ranging from image generation to music synthesis (Ho et al., 2020; Strudel et al., 2022; Hawthorne et al., 2022).

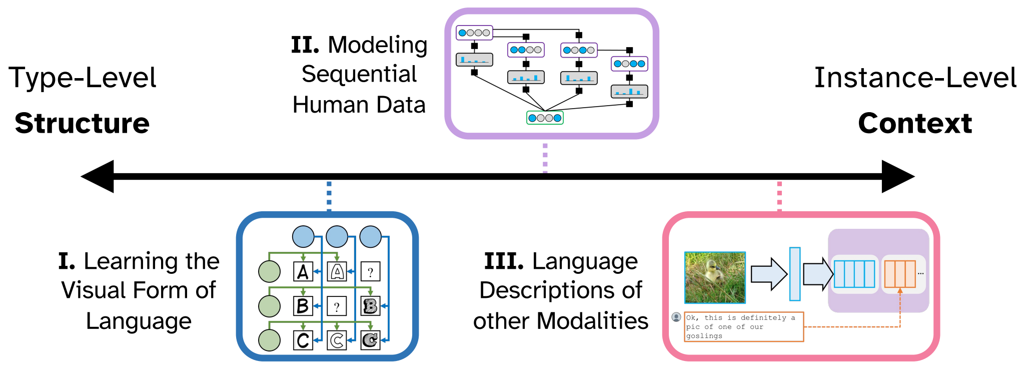

Type-Level
Structure
II. Modeling Sequential Human Data
Instance-Level
Context
I. Learning the Visual Form of Language
A
B
C
?
III. Language Descriptions of other Modalities
Ok, this is definitely a pic of one of our goslings

Figure 1.1: Overview of the major parts of this dissertation, organized roughly coarse-to-fine from type-level tasks on the left to instance-level on the right.

For the purposes of this dissertation, we are primarily interested in answering the question "To what extent can we learn better representations not through different loss functions, but by adding structure to our model design?" Following the general guiding philosophy of the prior work mentioned above, we will employ well-defined probabilistic graphical models parameterized by dense neural architectures for many of the tasks we consider. The remainder will make use of retrieval guided generation and checklist models that still retain some controllability over black box systems. This will allow us to in a sense achieve the best of both worlds, combining the expressive power of bespoke deep learning architectures with the smoothness and interpretability of probabilistic distributions over explicit latent variables.

1.2 Structure and Context

When learning representations from large corpora of language data specifically, the overwhelming strategy is to interpret that data as a collection of IID samples to be modeled in isolation from one another. While this approach is in some ways beneficial as it allows for efficient training via SGD and doesn't rely on metadata that may not always be present, it does come with limitations. Frequently we actually do observe relations between individual datapoints that can better inform the properties we might want to capture. For instance, authors of the same language may have social, geographic, or even pedagogical relationships to one another that influence how they write certain characters (Skelton, 2008). Words themselves share structural properties with cognates in other related languages (Hall and Klein, 2010, 2011). In a more modern context, alt-text descriptions of images posted online may need to be phrased differently depending on the context of why an image was posted in the first place. In fact at a higher level, the goal of modeling context has arguably been one of the principal driving forces of progress in the field of NLP over the past several years. The shift from type-level to contextualized word embeddings saw massive improvements across a variety of tasks (Peters et al., 2018). Recent work with masked language models has also shown benefits to including a next sentence prediction task in pretraining, as this encourages the model to learn how individual sentences fit together into larger documents (Devlin et al., 2019). The current paradigm of large-scale LLM pretraining has seen increasing attention towards strategies such as State Space Models that allow for longer and longer contexts (Gu and Dao, 2023). However even in this setting, these long individual sequences are still treated as IID compared to each other. By instead taking advantage of more complex structural links, we can allow information to flow within our corpora and allow our models to be more context-sensitive, and make use of heavier parameter sharing to more easily generalize to new examples.

We can in a way think of this line of work as adding inductive bias to more general purpose strategies based on additional context in which we find that data "in the wild". For example, topic modeling tends to view text samples as independent draws from some background distribution, but many types of text data exist as parts of larger conversations with complex graphs connecting them. Social media posts will often reference others, and we can learn more robust topics by considering them jointly, making us less likely to overlook information that might not be explicitly repeated for the sake of avoiding redundancy. Sometimes those dependencies are explicitly observed — we might see that some comment is a direct response to another — or sometimes the reply chain may be more implicit, and the fact that two posts are engaging in the same broader discourse is something we need to infer. In a way this mirrors recent advances in learning word embeddings by considering token level context, and not just learning interchangeable type level representations.

With these observations in mind, this thesis will focus on an underlying strategy of representation learning that asks the question, "How can we learn representations that decompose over the various properties we are interested in?" and answers it with the technique of leveraging structure and context within our corpora. To more directly state the overall argument of this dissertation:

Structure and context play an important role in much of our data, both at a high and low level respectively, and by injecting these dependencies into the design of our models we can learn representations that are more faithful to the underlying generative process.

By "structure" we are specifically taking inspiration from prior work on matrix factorization and recommender systems, where the data can be factored at the type level along a few different axes, and we can model it with embeddings that decompose along those axes (Freeman and Tenenbaum, 1997; Tenenbaum and Freeman, 2000). However unlike traditional matrix completion tasks, the data here is complex, with cells representing entire words, comments, or images of glyphs, and not just numbers. By "context" we mean systems that pay attention to metadata that may not typically be used in traditional task setups. In between these are domains containing token-level dependencies between adjacent or similar datapoints. The work we present here will fall varyingly along a spectrum between these rough endpoints, as illustrated in Figure 1.1.

1.3 Model Design

Beyond simply adjusting the input/output setup of our tasks though, we will also look into how we can add useful inductive bias to our models through lower-level modeling choices. For example, while a model that can learn a fully independent representation for all datapoints in a corpus might technically have higher capacity, by constraining those embeddings in specific ways we can more strongly pressure it to capture certain properties we care about. We may for instance choose to have those embeddings be composed of a collection of sub-embeddings which are shared by instances belonging to the same class, which in turn encourages them to reflect features common to all members of that class, and disregard those that are not. Furthermore, we can use the network architecture to add more pressure to what types of information these sub-embeddings contain. Looking forward to an example from image generation, we may use one set of embeddings to define an initial activation map as the input to a deconvolutional network, and another to define the filters which will be used to unpool it (Chapter 2). This creates an implicit pressure for these embeddings to learn properties related to visual structure and style respectively.

In terms of specific modeling choices, there exists a tradeoff between capacity and controllability that we will need to balance. On one hand modern neural networks offer powerful, expressively parameterized distributions, but we also desire hooks for control and interpretability within our systems. There are many ways that we will address this within the following chapters. Some of our work will center on what are at their core, well-defined probabilistic models that are in turn parameterized with high capacity neural architectures (Chapters 2, 3, and 5). These give us interpretable variables and more observable information flow than black box approaches. We also examine retrieve and edit, checklist models, and other non-variational strategies that still give us some low-level insight into the computation graph (Chapters 4, 6, and 8).

1.4 Multimodality

While our work on leveraging structure and context does not exclusively fit under Multimodal NLP in a formal sense, it does nonetheless heavily intersect with that field, which is of course a broad area of study with many key pillars to it. One of those shares a lineage with robotics, specifically work on embodiment, grounding, and reinforcement learning (Driess et al., 2023). While we do employ reinforcement learning in some cases, we will do so outside this more typical context of constructing autonomous agents. Transcription, captioning, and VQA constitute another broad category that is more relevant to us (Berg-Kirkpatrick et al., 2013; Antol et al., 2015). Recent work has also seen renewed interest in contrastive pretraining and joint representation learning (Radford et al., 2021; Andrew et al., 2013) with the aim of learning either a shared or aligned embedding space for semantically similar datapoints across different modalities, something which we will make use of. Also relevant to the work presented here is the modern boom in generative AI, diffusion, and text-to-vision/audio/video research (Ho et al., 2020). We are however more concerned here with the much less studied inverse of this setup.

The historical trajectory of the field of multimodal NLP would be challenging to cover exhaustively, but the work presented here will primarily fall under the umbrellas of representation learning and captioning. We use the colloquial term "language-adjacent" to describe our more narrow area of focus. By this we mean aspects of language that do not exist in the text modality, such as typography, data of alternate modalities that appear in contexts adjacent to language, such as images on social media, and even non-linguistic sequential human data that nonetheless benefit from modeling approaches originally developed for language processing tasks, such as music.

1.5 Thesis Contributions

In this work, we will examine these high-level concepts in the context of a variety of more specific domains for which much of the existing prior work has not explicitly made use of observable structure within the corpus. While the concepts of structured output generation and contextualized representation learning are relatively well studied, here we will focus on a few case studies, largely those that lie at the boundary between language and other domains, where existing general purpose methods have not yet considered the ways in which representations can inform one another. These case studies will break down into three broad categories of application, which we order for readability in a coarse-to-fine manner (see Figure 1.1 for a visualization). Part I will focus on the domain of typography, and will present research that improves over prior work via structured inference networks and architectures that add inductive bias. Part II will focus on two case studies of temporally ordered data, specifically social media discourse and piano music, which we will approach with both variational and non-variational message passing systems. Finally, Part III will focus on captioning, both for images and music. We will show how conditioning on signal not just from the non-language modality but also on relevant nearby text can drastically improve the contextual relevance and accuracy of generations. The detailed outline of the contributions of this thesis is presented below:

Part I: Learning the Visual Form of Language with Type Level Structure

- Chapter 2 - In this chapter, we will put forward an approach to representation learning for Latin glyphs that aims to disentangle stylistic and structural visual information. We achieve this by partially sharing embeddings across datapoints and making use of known connections between them. More specifically for this task, we factor glyph representations into two embeddings — one corresponding to character which is shared by all glyphs of that same symbol type, and one corresponding to style which is shared by all glyphs within the same font. By modeling glyphs with a pairwise combination of two embeddings shared by different groups instead of a single independently learned embedding, we impose an inductive bias that more directly encourages representations to capture properties related to group membership. This work dramatically increased the fluency of font reconstructions for Latin characters while also doing so in a way that allowed for the creation of entirely new fonts by interpolating between existing ones. It was presented at **EMNLP 2019** (Srivatsan et al., 2019).
- Chapter 3 - These next two chapters directly extend the work of the previous by applying it to two new datasets, each with their own unique challenges. We start by recasting the character embedding as a latent variable instead of a directly learned model parameter, and evaluate this method on Google Fonts, a much larger (and also sparser) dataset representing multiple writing systems. This allows us to generalize to characters that were unseen during training time by simply performing inference over a few reference examples without having to retrain the full system. This work enabled font reconstruction to be scaled up to much larger and sparser character sets and be able to operate on non-Latin writing systems. It was presented at **EMNLP 2021** (Srivatsan et al., 2021b).
- Chapter 4 - Next we investigate a dataset not of digital fonts, but of hand written inscriptions of Linear B, a Mycenean Greek writing system. Our goal here is to learn representations of the stylistic idiosyncracies of particular scribal hands. To do this, we migrate to a non-probabilistic model setup that uses a pair of discriminator networks to encourage extracting meaningful features from the comparatively smaller dataset. This work for the first time automated previously human approaches to identifying patterns in scribal hands' writing styles, and was able to implicitly learn to group them by findplace despite not having been trained on that. It was presented at the **IWCP** at **ICDAR 2021** (Srivatsan et al., 2021a).

Part II: Modeling Sequential Human Data with Temporal Dependencies

- Chapter 5 - In this chapter, we'll examine the task of topic modeling for threads posted on Reddit, a popular online forum that covers a wide variety of topics of discussion, roughly organized into "subreddits". Traditional approaches to topic modeling tend to focus on documents as independent from one another, with some background distribution on topic balance that is randomly sampled for each document. We however want to leverage the fact that threads on Reddit tend to follow a roughly hierarchical structure where discussions may be broadly organized around a particular topic, but individual comments also vary wildly in their word distributions and might delve into increasingly niche subjects deeper into the conversation. For this we propose a topic model that factors into two separate representations — a thread-level distribution that captures higher level topical information present throughout the thread and is shared by all comments within it, and a comment level distribution that captures more stylistic language properties and finer grained subtopics. Importantly, our comment level representations are not treated as conditionally independent variables. Instead, we institute a log-bilinear factor that explicitly tracks how topical information from parent comments influence that of their children. This work created the first branching distributed topic model, which learns disentangled representations of style and semantic topic. It was presented at **EMNLP 2018** (Srivatsan et al., 2018).

- Chapter 6 - In the next chapter, we put forward an approach to piano fingering prediction. This model uses a checklist to track where the player's fingers are at a given point in time based on previous predictions, and uses this to inform which choices are most physically feasible at the current moment. This minimizes the occurrence of patterns common in previous method's outputs which are accurate on average per note, but may frequently contain disfluencies that would be challenging to play in ways not reflected by an overall accuracy score. In order to perform more meaningful evaluation on this task, we also put forward several new metrics that explicitly track the playability of an output sequence, and use reinforcement learning to optimize these directly at train time. This work significantly improved the actual human playability of automatically generated fingerings, exposed weaknesses in existing evaluation metrics, and proposed new ones with strategies for directly optimizing them. It was presented at **ISMIR 2022** (Srivatsan and Berg-Kirkpatrick, 2022).

Part III: Contextually Relevant Text Descriptions of other Modalities

- Chapter 7 - In this chapter, we'll look into the use of multimodal sources of control for text generation, specifically in the domain of alt-text captioning for images on Twitter. Image captioning has so far been treated as a strictly image-to-text task, but that framework ignores the fact that when describing an image for accessiblity purposes, the description itself depends on the context in which the image was posted. We find that by allowing our model to condition on both visual features from the image and also text features from the accompanying tweet, we can produce descriptions that are more contextually relevant, and also more accurate at transcribing text. This work created a viable model for automatic alt-text generation that improved on known issues in previous systems, opening the door for significantly expanded accessibility in social media. It was presented at **ICLR 2024** (Srivatsan et al., 2024b).

- Chapter 8 - The final chapter takes this idea and extends it to the domain of music captioning. Here we make use of a similarly multimodal prefix, although this time the text source is not an accompanying tweet, but rather a candidate description retrieved from our training set. This allows the model to use a neighbor song's gold description as a starting point, taking pressure off of the decoder, and exploiting the inherent similarity of descriptions of songs in similar genres. This work improved over existing captioning systems by directly leveraging the unexpected strength of a previously overlooked naive baseline, potentially allowing for far more detailed automatic closed captioning. It was presented at the **IJCAI 2024** Special Track on AI, the Arts, and Creativity (Srivatsan et al., 2024a).

Part I

Learning the Visual Form of Language with Type Level Structure

Chapter 2

A Deep Factorization of Style and Structure in Fonts

2.1 Introduction

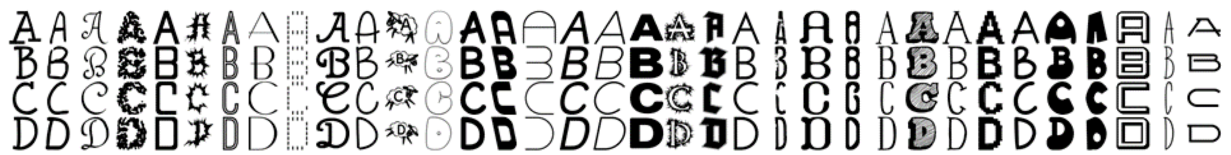


Figure 2.1: Example fonts from the Capitals64 dataset. The task of font reconstruction involves generating missing glyphs from partially observed novel fonts.

We'll begin by describing our approach to disentangled representation learning for the specific domain of digital fonts. This is an application where the matrix factorization framework that we alluded to in the introduction will hopefully be quite tangible. This chapter (and the subsequent one) will also emphasize the use of expressive decoders that will use asymmetrical architectures to add inductive bias to the model, one of the primary directions in which we'll be improving prior work over the course of this thesis. We'll now say more about the domain itself and in doing so motivate the use of this approach.

One of the most visible attributes of digital language data is its typography. A font makes use of unique stylistic features in a visually consistent manner across a broad set of characters while preserving the structure of each underlying form in order to be human readable – as shown in Figure 2.1. Modeling these stylistic attributes and how they compose with underlying character structure could aid typographic analysis and even allow for automatic generation of novel fonts. Further, the variability of these stylistic features presents a challenge for optical character recognition systems, which typically presume a library of known fonts. In the case of historical document recognition, for example, this problem is more pronounced due to the wide range of lost, ancestral fonts present in such data (Berg-Kirkpatrick et al., 2013; Berg-Kirkpatrick and Klein, 2014). Models that capture this wide stylistic variation of glyph images may eventually be useful for improving optical character recognition on unknown fonts.

In this work we present a probabilistic latent variable model capable of disentangling stylistic features of fonts from the underlying structure of each character. Our model represents the style of each font as a vector-valued latent variable, and parameterizes the structure of each character as a learned embedding. Critically, each style latent variable is shared by all characters within a font, while character embeddings are shared by characters of the same type across all fonts. Thus, our approach is related to a long line of literature on using tensor factorization as a method for disentangling style and content (Freeman and Tenenbaum, 1997; Tenenbaum and Freeman, 2000; Vasilescu and Terzopoulos, 2002; Tang et al., 2013) and to recent deep tensor factorization techniques (Xue et al., 2017).

Inspired by neural methods' ability to disentangle loosely coupled phenomena in other domains, including both language and vision (Hu et al., 2017; Yang et al., 2017; Gatys et al., 2016; Zhu et al., 2017), we parameterize the distribution that combines style and structure in order to generate glyph images as a transpose convolutional neural decoder (Dumoulin and Visin, 2016). Further, the decoder is fed character embeddings early on in the process, while the font latent variables directly parameterize the convolution filters. This architecture biases the model to capture the asymmetric process by which structure and style combine to produce an observed glyph.

We evaluate our learned representations on the task of *font reconstruction*. After being trained on a set of observed fonts, the system reconstructs missing glyphs in a set of previously unseen fonts, conditioned on a small observed subset of glyph images. Under our generative model, font reconstruction can be performed via posterior inference. Since the posterior is intractable, we demonstrate how a variational inference procedure can be used to perform both learning and accurate font reconstruction. In experiments, we find that our proposed latent variable model is able to substantially outperform both a strong nearest-neighbors baseline as well as a state-of-the-art discriminative system on a standard dataset for font reconstruction. Further, in qualitative analysis, we demonstrate how the learned latent space can be used to interpolate between fonts, hinting at the practicality of more creative applications.

2.2 Related Work

Discussion of computational style and content separation in fonts dates at least as far back as the writings of Hofstadter (1983, 1995). Some prior work has tackled this problem through the use of bilinear factorization models (Freeman and Tenenbaum, 1997; Tenenbaum and Freeman, 2000), while others have used discriminative neural models (Zhang et al., 2018, 2020) and adversarial training techniques (Azadi et al., 2018). In contrast, we propose a deep probabilistic approach that combines aspects of both these lines of past work. Further, while some prior approaches to modeling fonts rely on stroke or topological representations of observed glyphs (Campbell and Kautz, 2014; Phan et al., 2015; Suveeranont and Igarashi, 2010), ours directly models pixel values in rasterized glyph representations and allows us to more easily generalize to fonts with variable glyph topologies.

Finally, while we focus our evaluation on font reconstruction, our approach has an important relationship with style transfer – a framing made explicit by Zhang et al. (2018, 2020) – as the goal of our analysis is to learn a smooth manifold of font styles that allows for stylistic inference given a small sample of glyphs. However, many other style transfer tasks in the language domain (Shen et al., 2017) suffer from ambiguity surrounding the underlying division between style and semantic content. By contrast, in this setting the distinction is clearly defined, with content (i.e. the character) observed as a categorical label denoting the coarse overall shape of a glyph, and style (i.e. the font) explaining lower-level visual features such as boldness, texture, and serifs. The modeling approach taken here might inform work on more complex domains where the division is less clear.

2.3 Font Reconstruction

We can view a collection of fonts as a matrix, X, where each column corresponds to a particular character type, and each row corresponds to a specific font. Each entry in the matrix, x_{ij}, is an image of the glyph for character i in the style of a font j, which we observe as a 64×64 grayscale image as shown in Figure 2.1. In a real world setting, the equivalent matrix would naturally have missing entries wherever the encoding for a character type in a font is undefined. In general, not all fonts contain renderings of all possible character types; many will only support one particular language or alphabet and leave out uncommon symbols. Further, for many commercial applications, only the small subset of characters that appears in a specific advertisement or promotional message will have been designed by the artist – the majority of glyphs are missing. As a result, we may wish to have models that can infer these missing glyphs, a task referred to as font reconstruction.

Following recent prior work (Azadi et al., 2018), we define the task setup as follows: During training we have access to a large collection of observed fonts for the complete character set. At test time we are required to predict the missing glyph images in a collection of *previously unseen* fonts with the same character set. Each test font will contain observable glyph images for a small randomized subset of the character set. Based on the style of this subset, the model must

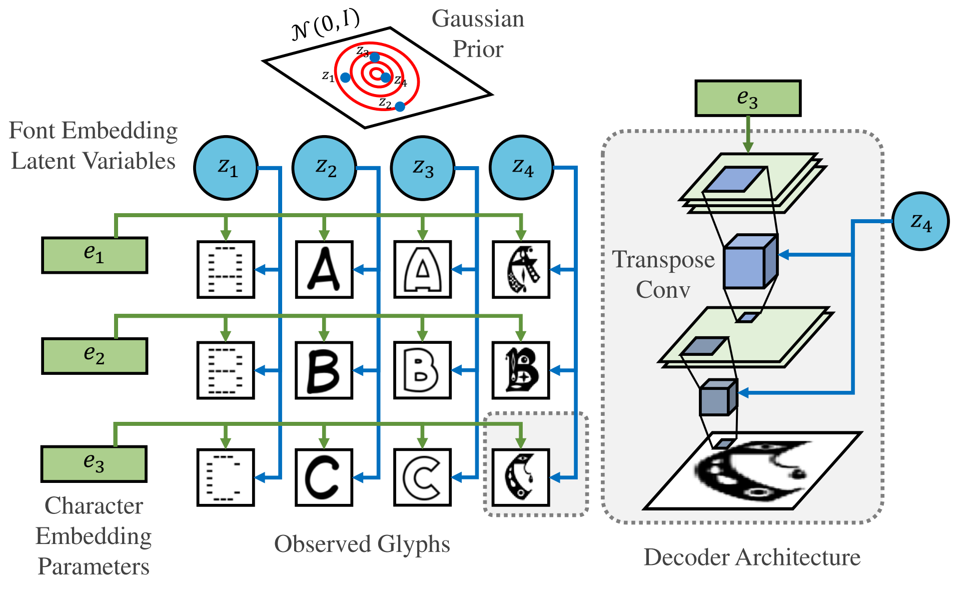


Figure 2.2: Depiction of the generative process of our model. Each observed glyph image is generated conditioned on the latent variable of the corresponding font and the embedding parameter of the corresponding character type. For a more detailed description of the decoder architecture and hyperparameters, see Section 2.6.4.

reconstruct glyphs for the rest of the character set.

Font reconstruction can be thought of as a form of matrix completion; given various observations in both a particular row and column, we wish to reconstruct the element at their intersection. Alternatively we can view it as a few-shot style transfer task, in that we want to apply the characteristic attributes of a new font (e.g. serifs, italicization, drop-shadow) to a letter using a small number of examples to infer those attributes. Past work on font reconstruction has focused on discriminative techniques. For example Azadi et al. (2018) used an adversarial network to directly predict held out glyphs conditioned on observed glyphs. By contrast, we propose a generative approach using a deep latent variable model. Under our approach fonts are generated based on an unobserved style embedding, which we can perform inference over given any number of observations.

2.4 Model

Figure 2.2 depicts our model's generative process. Given a collection of images of glyphs consisting of I character types across J fonts, our model hypothesizes a separation of character-specific structural attributes and font-specific stylistic attributes into two different representations. Since all characters are observed in at least one font, each character type is represented as an embedding vector which is part of the model's parameterization. In contrast, only a subset of fonts is observed during training and our model will be expected to generalize to reconstructing unseen fonts at test time. Thus, our representation of each font is treated as a vector-valued latent variable rather than a deterministic embedding.

More specifically, for each font in the collection, a font embedding variable, $z_j \in \mathbb{R}^k$, is sampled from a fixed multivariate Gaussian prior, $p(z_j) = \mathcal{N}(0, I_k)$. Next, each glyph image, x_{ij}, is generated independently, conditioned on the corresponding font variable, z_j, and a character-specific parameter vector, $e_i \in \mathbb{R}^k$, which we refer to as a character embedding. Thus, glyphs of the same character type share a character embedding, while glyphs of the same font share a font variable. A corpus of $I * J$ glyphs is modeled with only I character embeddings and J font variables, as seen in the left half of Figure 2.2. This modeling approach can be thought of as a form of deep matrix factorization, where the content at any given cell is purely a function of the vector representations of the corresponding row and column. We denote the full corpus of glyphs as a matrix $X = ((x_{11}, ..., x_{1J}), ..., (x_{I1}, ...x_{IJ}))$ and denote the corresponding character embeddings as $E = (e_1, ..., e_I)$ and font variables as $Z = (z_1, ..., z_J)$.

Under our model, the probability distribution over each image, conditioned on z_j and e_i, is parameterized by a neural network, described in the next section and depicted in Figure 2.2. We denote this decoder distribution as $p(x_{ij}|z_j; e_i, \phi)$, and let ϕ represent parameters, shared by all glyphs, that govern how font variables and character embeddings combine to produce glyph images. In contrast, the character embedding parameters, e_i, which feed into the decoder, are only shared by glyphs of the same character type. The font variables, z_j, are unobserved during training and will be inferred at test time. The joint probability under our model is given by:

$$p(X, Z; E, \phi) = \prod_{i,j} p(x_{ij}|z_j; e_i, \phi)p(z_j)$$

2.4.1 Decoder Architecture

One way to encourage the model to learn disentangled representations of style and content is by choosing an architecture that introduces helpful inductive bias. For this domain, we can think of the character type as specifying the overall shape of the image, and the font style as influencing the finer details; we formulate our decoder with this difference in mind. We hypothesize that a glyph can be modeled in terms of a low-resolution character representation to which a complex operator specific to that font has been applied.

The success of transpose convolutional layers at generating realistic images suggests a natural way to apply this intuition. A transpose convolution[1] is a convolution performed on an undeci-

[1] sometimes erroneously referred to as a "deconvolution"

mated input (i.e. with zeros inserted in between pixels in alternating rows and columns), resulting in an upscaled output. Transpose convolutional architectures generally start with a low resolution input which is passed through several such layers, iteratively increasing the resolution and decreasing the number of channels until the final output dimensions are reached. We note that the asymmetry between the coarse input and the convolutional filters closely aligns with the desired inductive biases, and therefore use this framework as a starting point for our architecture.

Broadly speaking, our architecture represents the underlying shape that defines the specific character type (but not the font) as coarse-grained information that therefore enters the transpose convolutional process early on. In contrast, the stylistic content that specifies attributes of the specific font (such as serifs, drop shadow, texture) is represented as finer-grained information that enters into the decoder at a later stage, by parameterizing filters, as shown in the right half of Figure 2.2. Specifically we form our decoder as follows: first the character embedding is projected to a low resolution matrix with a large number of channels. Following that, we apply several transpose convolutional layers which increase the resolution, and reduce the number of channels. Critically, the convolutional filter at each step is not a learned parameter of the model, but rather the output of a small multilayer perceptron whose input is the font latent variable z. Between these transpose convolutions, we insert vanilla convolutional layers to fine-tune following the increase in resolution.

Overall, the decoder consists of four blocks, where each block contains a transpose convolution, which upscales the previous layer and reduces the number of channels by a factor of two, followed by two convolutional layers. Each (transpose) convolution is followed by an instance norm and a ReLU activation. The convolution filters all have a kernel size of 5×5. The character embedding is reshaped via a two-layer MLP into a $8 \times 8 \times 256$ tensor before being fed into the decoder. The final 64×64 dimensional output layer is treated as a grid of parameters which defines the output distribution on pixels. We describe the specifics of this distribution in the next section.

2.4.2 Projected Loss

The conventional approach for computing loss on image observations is to use an independent output distribution, typically a Gaussian, on each pixel's intensity. However, deliberate analysis of the statistics of natural images has shown that images are not well-described in terms of statistically independent pixels, but are instead better modeled in terms of edges (Field, 1987; Huang and Mumford, 1999). It has also been demonstrated that images of text have similar statistical distributions as natural images (Melmer et al., 2013). Following this insight, as our reconstruction loss we use a heavy-tailed (leptokurtotic) distribution placed on a transformed representation of the image, similar to the approach of Barron (2019). Modeling the statistics of font glyphs in this fashion results in sharper samples, while modeling independent pixels with a Gaussian distribution results in blurry, oversmoothed results.

More specifically, we adopt one of the strategies employed in Barron (2019), and transform image observations using the orthonormal variant of the 2-Dimensional Discrete Cosine Transform (2-D DCT-II) (Ahmed et al., 1974), which we denote as $f : \mathbb{R}^{64\times 64} \rightarrow \mathbb{R}^{64\times 64}$ for our 64×64 dimensional image observations. We transform both the observed glyph image and the corresponding grid or parameters produced by our decoder before computing the observation's likelihood.

This procedure projects observed images onto a grid of orthogonal bases comprised of shifted and scaled 2-dimensional cosine functions. Because the DCT-II is orthonormal, this transformation is volume-preserving, and so likelihoods computed in the projected space correspond to valid measurements in the original pixel domain.

Note that because the DCT-II is simply a rotation in our vector space, imposing a normal distribution in this transformed space should have little effect (ignoring the scaling induced by the diagonal of the covariance matrix of the Gaussian distribution) as Euclidean distance is preserved under rotations. For this reason we impose a heavy-tailed distribution in this transformed space, specifically a Cauchy distribution. This gives the following probability density function

$$g(x; \hat{x}, \gamma) = \frac{1}{\pi\gamma\left(1 + \left(\frac{f(x)-f(\hat{x})}{\gamma}\right)^2\right)}$$

where x is an observed glyph, $\hat{x}$ is the location parameter grid output by our decoder, and γ is a hyperparameter which we set to $\gamma = 0.001$.

The Cauchy distribution accurately captures the heavy-tailed structure of the edges in natural images. Intuitively, it models the fact that images tend to be mostly smooth, with a small amount of non-smooth variation in the form of edges. Computing this heavy-tailed loss over the frequency decomposition provided by the DCT-II instead of the raw pixel values encourages the decoder to generate sharper images without needing either an adversarial discriminator or a vectorized representation of the characters during training. Note that while our training loss is computed in DCT-II space, at test time we treat the raw grid of parameter outputs $\hat{x}$ as the glyph reconstruction.

2.5 Learning and Inference

Note that in our training setting, the font variables Z are completely unobserved, and we must induce their manifold with learning. As our model is generative, we wish to optimize the marginal

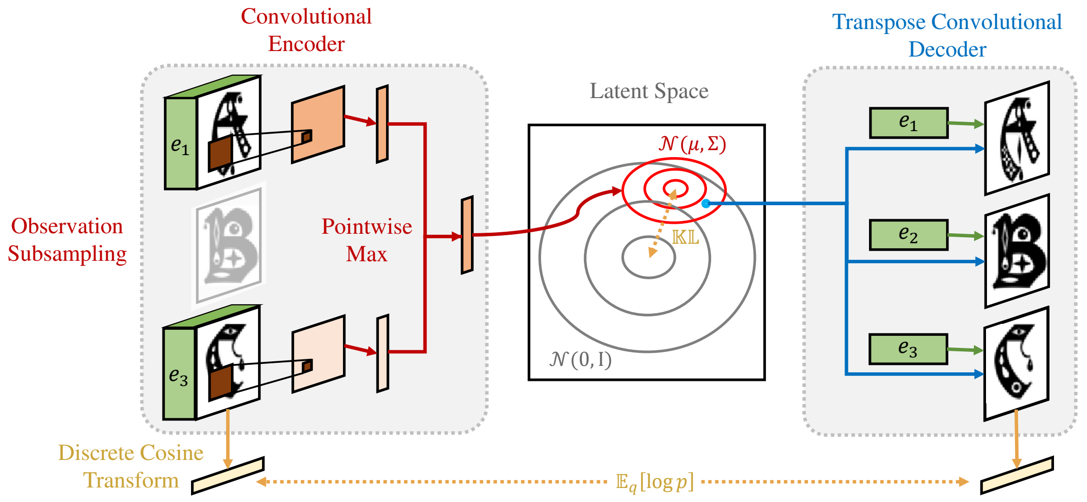


Figure 2.3: Depiction of the computation graph of the amortized variational lower bound (for simplicity, only one font is shown). The encoder approximates the generative model's true posterior over the font style latent variables given the observations. It remains insensitive to the number of observations by pooling high-level features across glyphs. For a more specific description of the encoder architecture details, see Section 2.6.4.

probability of just the observed X with respect to the model parameters E and ϕ:

$$p(X; E, \phi) = \int_Z p(X, Z; E, \phi) dZ$$

However, the integral over Z is computationally intractable, given that the complex relationship between Z and X does not permit a closed form solution. Related latent variable models such as Variational Autoencoders (VAE) (Kingma and Welling, 2014) with intractable marginals have successfully performed learning by optimizing a variational lower bound on the log marginal likelihood. This surrogate objective, called the evidence lower bound (ELBO), introduces a variational approximation, $q(Z|X) = \prod_j q(z_i|x_{1j}, \ldots, x_{Ij})$ to the model's true posterior, $p(Z|X)$. Our model's ELBO is as follows:

$$\begin{aligned}\text{ELBO} = \sum_j & \mathbb{E}_q[\log p(x_{1j}, \ldots, x_{Ij}|z_j)] \\ & - \mathbb{KL}(q(z_j|x_{1j}, \ldots, x_{Ij})||p(z_j))\end{aligned}$$

where the approximation q is parameterized via a neural encoder network. This lower bound can be optimized by stochastic gradient ascent if q is a Gaussian, via the reparameterization trick described in (Kingma and Welling, 2014; Rezende et al., 2014) to sample the expectation under q while still permitting backpropagation.

Practically speaking, a key property which we desire is the ability to perform consistent inference over z given a variable number of observed glyphs in a font. We address this in two ways: through the architecture of our encoder, and through a special masking process in training; both of which are shown in Figure 2.3.

2.5.1 Posterior Approximation

Observation Subsampling: To get reconstructions from only partially observed fonts at test time, the encoder network must be able to infer z_j from any subset of $(x_{1j}, \ldots, x_{Ij})$. One approach for achieving robustness to the number of observations is through the training procedure. Specifically when computing the approximate posterior for a particular font in our training corpus, we mask out a randomly selected subset of the characters before passing them to the encoder. This incentivizes the encoder to produce reasonable estimates without becoming too reliant on the features extracted from any one particular character, which more closely matches the setup at test time.

Encoder Architecture: Another way to encourage this robustness is through inductive bias in the encoder architecture. Specifically we use a convolutional neural network which takes in a batch of characters from a single font, concatenated with their respective character type embedding. Following the final convolutional layer, we perform an elementwise max operation across the batch, reducing to a single vector representation for the entire font which we pass through further fully-connected layers to obtain the output parameters of q as shown in Figure 2.3. By including this accumulation across the elements of the batch, we combine the features obtained from each character in a manner that is largely invariant to the total number and types of characters observed. This provides an inductive bias that encourages the model to extract similar features from each character type, which should therefore represent stylistic as opposed to structural properties.

Overall, the encoder over each glyph consists of three blocks, where each block consists of a convolution followed by a max pool with a stride of two, an instance norm (Ulyanov et al., 2016b), and a ReLU. The activations are then pooled across the characters via an elementwise max into a single vector, which is then passed through four fully-connected layers, before predicting the parameters of the Gaussian approximate posterior.

Reconstruction via Inference: At test time, we pass an observed subset of a new font to our encoder in order to estimate the posterior over z_j, and take the mean of that distribution as the inferred font representation. We then pass this encoding to the decoder along with the full set of character embeddings E in order to produce reconstructions of every glyph in the font.

2.6 Experiments

We now provide an overview of the specifics of the dataset and training procedure, and describe our experimental setup and baselines.

2.6.1 Data

We compare our model against baseline systems at font reconstruction on the Capitals64 dataset (Azadi et al., 2018), which contains the 26 capital letters of the English alphabet as grayscale 64×64 pixel images across $10,682$ fonts. These are broken down into training, dev, and test splits of 7649, 1473, and 1560 fonts respectively.

Upon manual inspection of the dataset, it is apparent that several fonts have an almost visually indistinguishable nearest neighbor, making the reconstruction task trivial using a naive algorithm (or an overfit model with high capacity) for those particular font archetypes. Because these datapoints are less informative with respect to a model's ability to generalize to previously *unseen* styles, we additionally evaluate on a second test set designed to avoid this redundancy. Specifically, we choose the 10% of test fonts that have maximal L_2 distance from their closest equivalent in the training set, which we call "Test Hard".

2.6.2 Baselines

As stated previously, many fonts fall into visually similar archetypes. Based on this property, we use a nearest neighbors algorithm for our first baseline. Given a partially observed font at test time, this approach simply "reconstructs" by searching the training set for the font with the lowest L_2 distance over the observed characters, and copy its glyphs verbatim for the missing characters.

For our second comparison, we use the GlyphNet model from Azadi et al. (2018). This approach is based on a generative adversarial network, which uses discriminators to encourage the model to generate outputs that are difficult to distinguish from those in the training set. We test from the publicly available epoch 400 checkpoint, with modifications to the evaluation script to match the setup described above.

We also perform an ablation using fully-connected instead of convolutional layers. For more architecture details see Section 2.6.4.

2.6.3 Training Details

We train our model to maximize the expected log likelihood using the Adam optimization algorithm (Kingma and Ba, 2015) with a step size of 10^{-5} (default settings otherwise), and perform early stopping based on the approximate log likelihood on a hard subset of dev selected by the process described earlier. To encourage robustness in the encoder, we randomly drop out glyphs during training with a probability of 0.7 (rejecting samples where all characters in a font are dropped). All experiments are run with a dimensionality of 32 for the character embeddings and font latent variables. Our implementation[2] is built in PyTorch (Paszke et al., 2017) version `1.1.0`.

2.6.4 Network Architecture

We now provide further details on the specific architectures used to parameterize our model and inference network. The following abbreviations are used to represent various components:

- F_i : fully-connected layer with i hidden units
- R : ReLU activation
- M : batch max pool
- S : 2×2 spatial max pool
- C_i : convolution with i filters of 5×5, 2 pixel zero-padding, stride of 1, dilation of 1
- I : instance normalization
- $\mathrm{T}_{i,j,k}$: transpose convolution with i filters of 5×5, 2 pixel zero-padding, stride of j, k pixel output padding, dilation of 1, where kernel and bias are the output of an MLP (described below)
- H : reshape to $26 \times 256 \times 8 \times 8$

Our fully-connected encoder is:
F_{128}-R-F_{128}-R-F_{128}-R-F_{1024}-R-M-F_{128}-R-F_{128}-
R-F_{128}-R-F_{64}

Our convolutional encoder is:
C_{64}-S-I-R-C_{128}-S-I-R-C_{256}-S-I-R-F_{1024}-M-R-
F_{128}-R-F_{128}-R-F_{128}-R-F_{64}

Our fully-connected decoder is:
F_{128}-R-F_{128}-R-F_{128}-R-F_{128}-R-F_{128}-R-F_{4096}

Our transpose convolutional decoder is:

[2] `https://bitbucket.org/NikitaSrivatsan/DeepFactorizationFontsEMNLP19`

	Test Full				Test Hard			
Observations	1	2	4	8	1	2	4	8
NN	483.13	424.49	386.81	363.97	880.22	814.67	761.29	735.18
GlyphNet	669.36	533.97	455.23	416.65	935.01	813.50	718.02	653.57
Ours (FC)	353.63	316.47	293.67	281.89	**596.57**	556.21	527.50	513.25
Ours (Conv)	**352.07**	**300.46**	**271.03**	**254.92**	615.87	**556.03**	**511.05**	**489.58**

Table 2.1: L_2 reconstruction per glyph by number of observed characters. "Full" includes the entire test set while "Hard" is measured only over the 10% of test fonts with the highest L_2 distance from the closest font in train.

F_{128}-R-F_{16384}-R-H-$\mathrm{T}_{256,2,1}$-R-C_{256}-I-R-C_{256}-I-
R-$\mathrm{T}_{128,2,1}$-R-C_{128}-I-R-C_{128}-I-R-$\mathrm{T}_{64,2,1}$-I-R-C_{64}-
I-R-C_{64}-I-R-$\mathrm{T}_{32,1,0}$-I-R-C_{32}-I-R-C_{1}

MLP to compute a transpose convolutional parameter of size j is:
F_{128}-R-F_{j}

2.7 Results

We now present quantitative results from our experiments in both automated and human annotated metrics, and offer qualitative analysis of reconstructions and the learned font manifold.

2.7.1 Quantitative Evaluation

Automatic Evaluation: We show font reconstruction results for our system against nearest neighbors and GlyphNet in Table 2.1. Each model is given a random subsample of glyphs from each test font (we measure at 1, 2, 4, and 8 observed characters), with their character labels. We measure the average L_2 distance between the image reconstructions for the unobserved characters and the ground truth, after scaling intensities to $[0, 1]$.

Our system achieves the best performance for both the overall and hard subset of test for all numbers of observed glyphs. Nearest neighbors provides a strong baseline on the full test set, even outperforming GlyphNet. However it performs much worse on the hard subset. This makes sense as we expect nearest neighbors to do extremely well on any test fonts that have a close equivalent in train, but suffer in fidelity on less traditional styles. GlyphNet similarly performs worse on test hard, which could reflect the *missing modes* problem of GANs failing to capture the full diversity of the data distribution (Che et al., 2016; Tolstikhin et al., 2017). The fully-connected ablation is

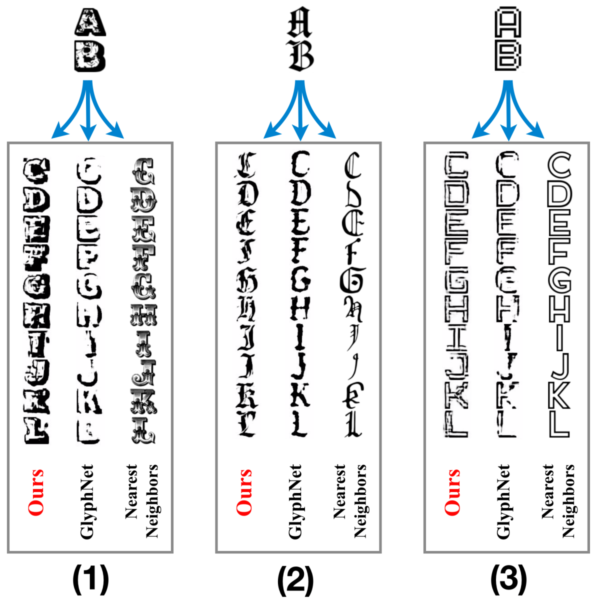


Figure 2.4: Reconstructions of partially observed fonts in the hard subset from our model, GlyphNet, and nearest neighbors. Given images of glyphs for ‘A’ and ‘B’ in each font, we visualize reconstructions of the remaining characters. Fonts are chosen such that the L_2 loss of our model on these examples closely matches the average loss over the full evaluation set.

also competitive, although we see that the convolutional architecture is better able to infer style from larger numbers of observations. On the hard test set, the fully-connected network even outperforms the convolutional system when only one observation is present, perhaps indicating that its lower-capacity architecture better generalizes from very limited data.

Human Evaluation: To measure how consistent these perceptual differences are, we also perform a human evaluation of our model's reconstructions against GlyphNet using Amazon Mechanical Turk (AMT). In our setup, turkers were asked to compare the output of our model against the output of GlyphNet for a single font given one observed character, which they were also shown. Turkers selected a ranking based on which reconstructed font best matched the style of the observed character, and a separate ranking based on which was more realistic. On the full test set (1560 fonts, with 5 turkers assigned to each font), humans preferred our system over GlyphNet 81.3% and 81.8% of the time for style and realism respectively. We found that on average 76% of turkers shown a given pair of reconstructions selected the same ranking as each other on both criteria, suggesting high annotator agreement.

2.7.2 Qualitative Analysis

In order to fully understand the comparative behavior of these systems, we also qualitatively compare the reconstruction output of these systems to analyze their various failure modes, showing

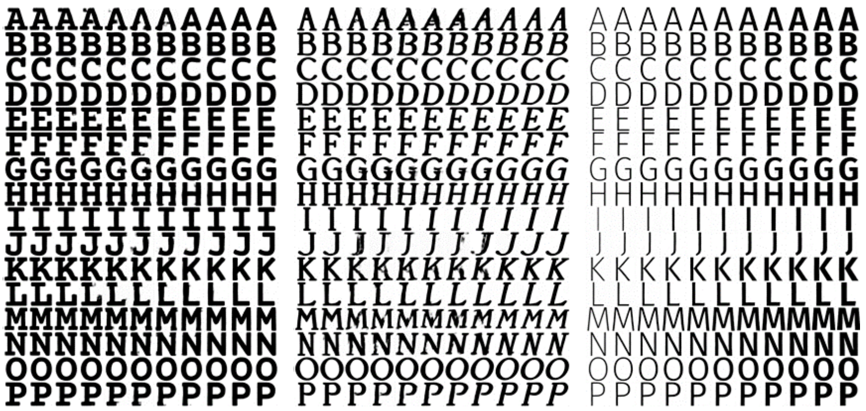


Figure 2.5: Interpolation between font variants from the same font family, showing smoothness of the latent manifold. Linear combinations of the embedded fonts correspond to outputs that lie intuitively "in between".

examples in Figure 2.4. We generally find that our approach tends to produce reconstructions that, while occasionally blurry at the edges, are generally faithful at reproducing the principal stylistic features of the font. For example, we see that for font (1) in Figure 2.4, we match not only the overall shape of the letters, but also the drop shadow and to an extent the texture within the lettering, while GlyphNet does not produce fully enclosed letters or match the texture. The output of nearest neighbors, while well-formed, does not respect the style of the font as closely as it fails to find a font in training that matches these stylistic properties. In font (2) the systems all produce a form of gothic lettering, but the output of GlyphNet is again lacking in certain details, and nearest neighbors makes subtle but noticeable changes to the shape of the letters. In the final example (3) we even see that our system appears to attempt to replicate the pixelated outline, while nearest neighbors ignores this subtlety. GlyphNet is in this case somewhat inconsistent, doing reasonably well on some letters, but much worse on others. Overall, nearest neighbors will necessarily output well-formed glyphs, but with lower fidelity to the style, particularly on more unique fonts. While GlyphNet does pick up on some subtle features, our model tends to produce the most coherent output on harder fonts.

2.7.3 Analysis of Learned Manifold

Since our model attempts to learn a smooth manifold over the latent style, we can also perform interpolation between the inferred font representations, something which is not directly possible using either of the baselines. In this analysis, we take two fonts from the same font family, which differ along one property, and pass them through our encoder to obtain the latent font variable for each. We then interpolate between these values, passing the result at various steps into our decoder to produce new fonts that exists in between the observations. In Figure 2.5 we see how our model can apply serifs, italicization, and boldness gradually while leaving the font unchanged in other respects. This demonstrates that our manifold is smooth and interpretable, not just at the

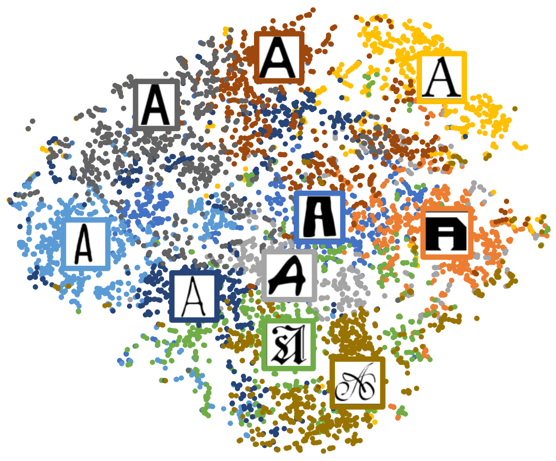


Figure 2.6: t-SNE projection of latent font variables inferred for the full training set, colored by k-means clustering with $k = 10$. The glyph for the letter "A" for each centroid is shown in overlay.

points corresponding to those in our dataset. This could be leveraged to modify existing fonts with respect to a particular attribute to generate novel fonts efficiently.

Beyond looking at the quality of reconstructions, we also wish to analyze properties of the latent space learned by our model. To do this, we use our encoder to infer latent font variables z for each of the fonts in our training data, and use a t-SNE projection (Maaten and Hinton, 2008) to plot them in 2D, shown in Figure 2.6. Since the broader, long-distance groupings may not be preserved by this transformation, we perform a k-means clustering (Lloyd, 1982) with $k = 10$ on the high-dimensional representations to visualize these high-level groupings. Additionally, we display a sample glyph for the centroid for each cluster. We see that while the clusters are not completely separated, the centroids generally correspond to common font styles, and are largely adjacent to or overlapping with those with similar stylistic properties; for example script, handwritten, and gothic styles are very close together.

To analyze how well our latent embeddings correspond to human defined notions of font style, we also run our system on the Google Fonts [3] dataset which despite containing fewer fonts and less diversity in style, lists metadata including numerical weight and category (e.g. *serif*, *handwriting*, *monospace*). In Figure 2.7 we show t-SNE projections of latent font variables from our model trained on Google Fonts, colored accordingly. We see that the embeddings do generally cluster by weight as well as category suggesting that our model is learning latent information consistent with how humans perceive font style.

[3] https://github.com/google/fonts

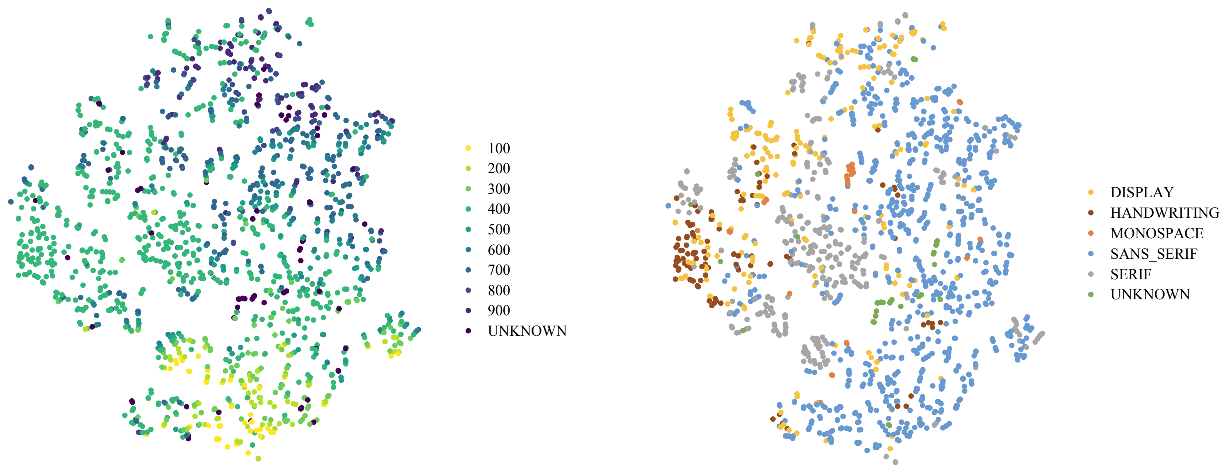


Figure 2.7: t-SNE projection of latent font variables inferred on Google Fonts, colored by weight and category.

2.8 Conclusion

This chapter presented a latent variable model of glyphs which learns disentangled representations of the structural properties of underlying characters from stylistic features of the font. We evaluated our model on the task of font reconstruction and showed that it outperformed both a strong nearest neighbors baseline and prior work based on GANs especially for fonts highly dissimilar to any instance in the training set. In future work, it may be worth extending this model to learn a latent manifold on content as well as style, which could allow for reconstruction of previously unseen character types, or generalization to other domains where the notion of content is higher dimensional.

Acknowledgments

This project is funded in part by the NSF under grant 1618044 and by the NEH under grant HAA-256044-17. Special thanks to Si Wu for help constructing the Google Fonts figures.

Chapter 3

Scalable Font Reconstruction with Dual Latent Manifolds

3.1 Introduction

In the previous chapter we described an approach to factored representation learning and reconstruction for completion of digital fonts. However, while that work was specifically successful in the context of Latin uppercase characters, the number of character types that a font may be expected to support is extremely large, with `Unicode 13.0.0` including as many as 143,859 character types (Unicode). As a result, graphic designers often create glyphs only for a subset of these characters, which tends to be determined by their own cultural context. This can create an accessibility gap for users seeking to create or read digital content in languages with less widespread orthographies, due to the relative lack of available options. Figure 3.1 shows that within the Google Fonts library (Google) there is a long tail of fonts with a large proportion of missing glyphs. While this domain presents what is arguably a more realistic use-case for our methodology, it does in turn present many additional challenges which we will need to develop strategies to solve.

To reiterate the task of font reconstruction, our overall goal is for a model, given a small set of example glyphs from an incomplete font, to generate glyphs for the remaining characters in a consistent style. While some approaches such as the one presented in the previous chapter do use a variational framework (Srivatsan et al., 2019; Lopes et al., 2019), the key limitation which we will now attempt to solve is that generally their models only treat the font style as a latent variable, and not the character shape. Such methods can therefore only handle a small, fixed, and an a priori known set of character types. Other work such as that by Zhang et al. (2018); Gao et al. (2019) use discriminative models which dynamically compute both embeddings, allowing them to generalize to unseen characters. However these networks typically require a pre-specified

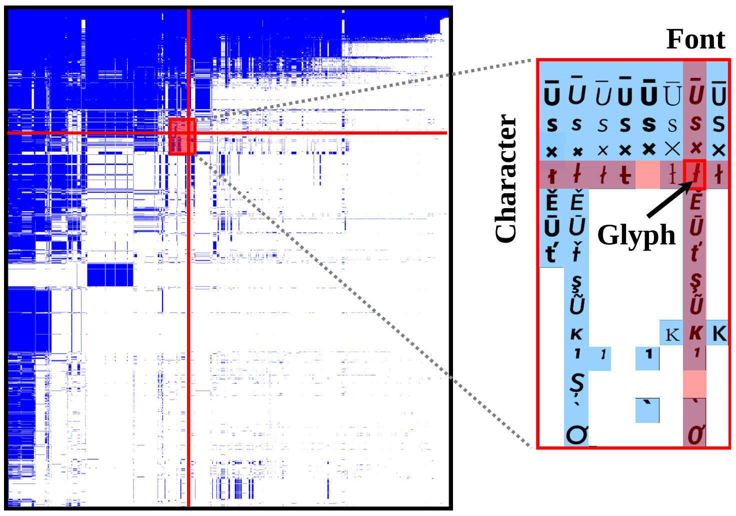


Figure 3.1: Supported glyphs in Google Fonts organized by character type and font. A blue pixel indicates that column's font includes that row's character. Our proposed model allows font reconstruction over this large, sparse character set.

number of observations as input, and their lack of a probabilistic prior can lead to learning a brittle manifold on datasets with a large number of infrequently observed characters.

By contrast, our method learns two smooth manifolds over character shape and font style in order to better share parameters across structurally similar characters, letting it scale to a larger set and more effectively generalize to characters never seen during training. Our model treats font reconstruction as a matrix factorization problem, where we view our corpus as a matrix with rows corresponding to character type, and columns corresponding to fonts. Each row and column is assigned a latent variable that determines its structure or style respectively. A decoder network consisting of transposed convolutional layers parameterizes the model's distribution on each cell in that matrix, *i.e.* an image of a glyph, conditioned on the corresponding row and column embeddings. This approach can be thought of as a generalization of Srivatsan et al. (2019), who used a similar factorization framework, but with only one manifold over font style.

In addition to model structure, the loss function is also important in font reconstruction as pixel independent losses like L_2 tend to produce blurry output, reflecting an averaged expectation instead of something realistic. Some have used generative adversarial networks (GANs) to mitigate this (Azadi et al., 2018), but these can suffer from missing modes and collapse issues. We instead introduce a novel adaptive loss to font reconstruction that operates on a wavelet image representation, while still permitting a well formed likelihood.

Specifically, in this chapter we make the following contributions: (1) Propose the "Dual Manifold" model which treats both style and structure representations as latent variables (2) Propose a new adaptive loss function for synthesizing glyphs, and demonstrate its improvements over more common losses (3) Put forward two datasets that emphasize few-shot reconstruction, and are preprocessed to remove near-duplicate fonts resulting in more challenging train/test splits.

We evaluate on the task of few-shot font reconstruction, reporting the structural similarity (SSIM) – a popular metric for image synthesis better correlated with human judgement than L_2 (Snell et al., 2017) – between reconstructions and a gold reference. These experiments are further split into *known* characters, which the model observed in at least one font at train time, and *unknown* characters, which can be thought of as a few-shot task. In addition we also perform human evaluation using Amazon Mechanical Turk. Our approach outperforms various baselines including nearest neighbor, the single manifold approach we build on (Srivatsan et al., 2019), and the previously mentioned discriminative model (Zhang et al., 2018).

3.2 Related Work

A variety of style transfer work has focused specifically on font style, and therefore, font reconstruction. Some approaches have sought to model the style aspect as a transformation on an underlying topological or stroke-based representation which must be learned for each character (Campbell and Kautz, 2014; Phan et al., 2015; Suveeranont and Igarashi, 2010). However this requires characters to have consistent topologies across fonts. Other work has learned a font skeleton manifold using Gaussian Process Latent Variable Models (Lian et al., 2018). One of the more philosophically similar approaches to ours is the bilinear factorization model of Freeman and Tenenbaum (1997); Tenenbaum and Freeman (2000) which also learns vector representations for each font and character type, albeit in a non-probabilistic and linear manner. Some more recent research has treated font reconstruction as a discriminative task, using modern neural architectures and techniques from the style transfer literature (Zhang et al., 2018, 2020; Azadi et al., 2018; Gao et al., 2019). Furthermore the concept of learning manifolds for Chinese characters based on shared structure has also been studied (Cao et al., 2018), albeit with different downstream goals. Lopes et al. (2019) used VAEs which do not observe font alignment across glyphs, but condition on the character type (this work also primarily focuses on generating vector instead of pixel representations). Finally, more general-purpose style transfer methods for images are well explored (Gatys et al., 2015; Yang et al., 2019; Johnson et al., 2016; Wang and Gupta, 2016; Kazemi et al., 2019; Chen et al., 2017; Ulyanov et al., 2016a), although these largely lack inductive biases specially suited to typography.

3.3 Dual Manifold Model

Srivatsan et al. (2019) is the most similar prior work, as it also builds from a matrix factorization framework, and learns a latent manifold over font embeddings. Our model generalizes theirs by

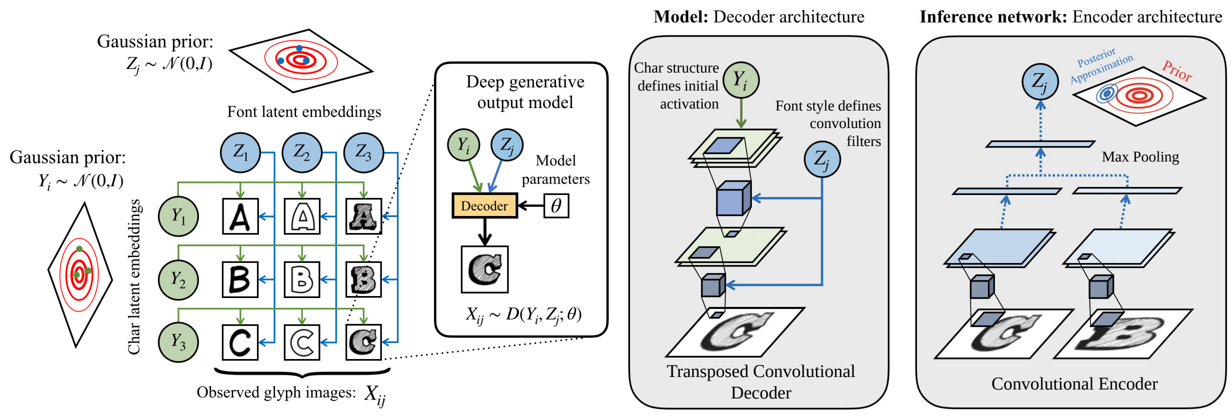

Gaussian prior:
$Z_j \sim \mathcal{N}(0,I)$
Font latent embeddings
Gaussian prior:
$Y_i \sim \mathcal{N}(0,I)$
Char latent embeddings
Z_1 Z_2 Z_3
Y_1 Y_2 Y_3
Observed glyph images: X_{ij}
Deep generative output model
Y_i Z_j
Model parameters
Decoder
θ
$X_{ij} \sim D(Y_i, Z_j; \theta)$
Model: Decoder architecture
Char structure defines initial activation
Font style defines convolution filters
Transposed Convolutional Decoder
Inference network: Encoder architecture
Posterior Approximation
Prior
Max Pooling
Convolutional Encoder

Figure 3.2: Overview of our generative factorization model, and important architecture details. Glyphs in the same row share a latent variable Y_i representing character shape, and those in the same column share Z_j representing font style. These variables are inferred by a network that takes in an entire row or column. Our decoder combines these representations to output a distribution on the glyph image.

learning a second manifold over character shape, letting us massively scale up the number of characters that can be modeled. In Section 3.4 we also describe our novel loss.

Figure 3.2 depicts our model's generative process. For a corpus consisting of J fonts, each defined over up to I character types, we characterize each particular glyph image as a combination of properties relating to the style of that particular font and to the shape of that character. Our model effectively factorizes the data by assigning a vector representation to every row and column which correspond to character and font respectively. Therefore, our approach works by leveraging the fact that all glyphs of the same character type (*i.e.* an entire row in our data) share the same underlying structural shape, and all glyphs within the same font (*i.e.* an entire column) share the same stylistic properties. By forming separate representations over each of these two axes of variation, we can reconstruct missing glyphs in our data by separately inferring the relevant row and column variables, and then pushing new combinations of those inferred variables through our generative process. This can be thought of as a form of matrix completion, where unobserved entries correspond to characters not supported by particular fonts.

Given a corpus X consisting of I characters across J fonts, we assign to each observed glyph X_{ij} a pair of latent variables which model the properties of that glyph's character type and font style. Specifically we define these as $Y_i \in \mathbb{R}^k$ and $Z_j \in \mathbb{R}^k$, which we draw from a standard Gaussian prior $\mathcal{N}(0, I_k)$, with Y modeling the shape of the character (e.g. a q or <), and Z modeling the properties of the font (e.g. *Times New Roman* or *Roboto Light Italic*). Given a particular Y_i and Z_j, we combine them via a neural decoder to obtain a distribution $p(X_{ij}|Y_i, Z_j; \theta)$ which scores the corresponding glyph image X_{ij}. This yields the following likelihood function:

$$p(X, Y, Z; \theta) = \prod_{I} p(Y_i) \prod_{J} p(Z_j) \prod_{I,J} p(X_{ij}|Y_i, Z_j; \theta)$$

Both Y and Z are unobserved, and we must therefore infer both to train our model and produce reconstructions at test time. Note that by contrast, Srivatsan et al. (2019) represents characters as fixed parameters, and must only perform inference over font representations. We use a pair of encoder networks to perform amortized inference, as depicted in Figure 3.3.

3.3.1 Decoder Architecture

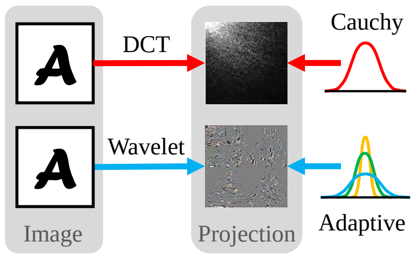
The basic structure of our decoder is largely identical to the popular U-Net architecture (Ronneberger et al., 2015) which has seen much success on image generation tasks with its coarse-to-fine layout of transposed convolutional layers. However, we make a few key modifications (depicted in Figure 3.2) in order to imbue our decoder with stronger inductive bias for this particular task. Following Srivatsan et al. (2019), instead of directly parameterizing the transposed convolutional layers that appear within each block of the network, we allow the weights of each layer to be the output of an MLP that takes as its input the font variable Z_j. This is effectively a form of HyperNetwork (Ha et al., 2016), a framework in which one network is used to produce the weights of another. In this way, the parameters of the transposed convolutional layers are dynamically chosen based on the font variable. By contrast, Y_i is the input fed in at the top of the decoder, to which these filters are applied. The purpose of this asymmetry is to encourage Z_j to learn properties relating to finer stylistic information, while Y_i learns more spatial information about the characters. In another manner of speaking, Y_i should learn "what" to write, and Z_j should learn "how" to write it.

3.4 Adaptive Wavelet Loss

Our decoder architecture outputs a grid of values, but an important decision is what distribution (and therefore loss) these should finally parameterize to score actual pixels. Traditional approaches using variational autoencoders have modeled each pixel as an independent Normal distribution, which results in the model minimizing the L_2 loss between its output and gold. This however leads to oversmoothed images, as it treats adjacent pixels as independent despite their strong correlations (Bell and Sejnowski, 1997), and fails to account for the heavy-tailed distribution of oriented edges in natural images (Field, 1987). As a result L_2 penalizes the model for generating images that are realistic but slightly transposed or otherwise not perfectly aligned with gold, which encourages models to produce fuzzy edges in order to be closer on average. GANs are often employed to force sharper output (Azadi et al., 2018; Gao et al., 2019), but following recent work we instead use a projected loss for a similar effect.

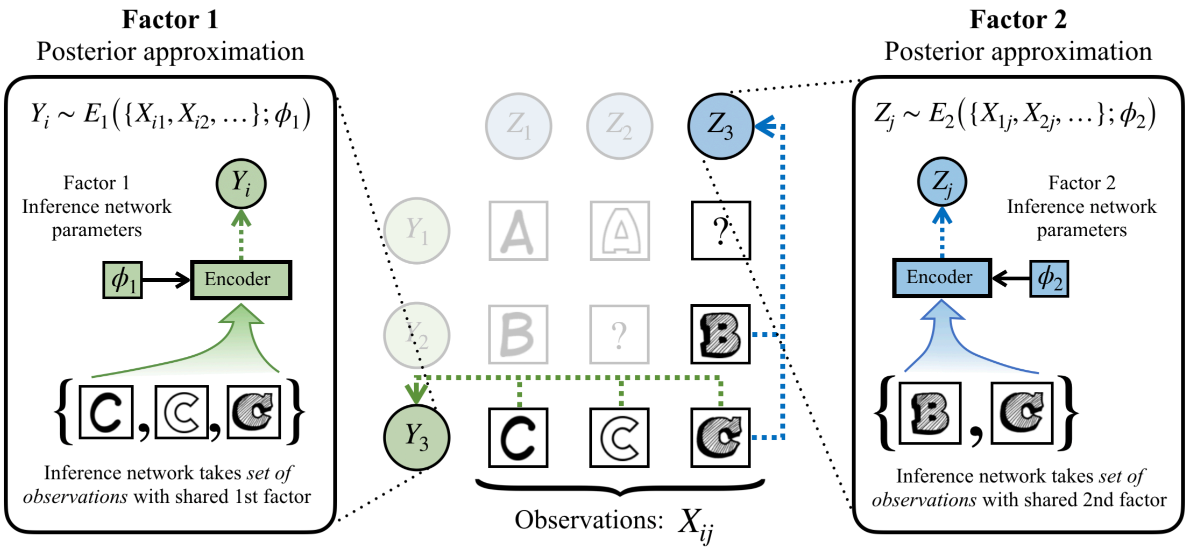
At a high level, our approach will first project images to a feature space, and let the model's output parameterize a distribution on this projection. If that projection is invertible and volume-preserving, this is equivalent to directly parameterizing a distribution on pixels, but allows for more expressivity (Rezende and Mohamed, 2015; Dinh et al., 2014, 2016). Ideally, such a loss requires a distribution expressive enough to capture the variable frequency characteristics of natural images, and a representation of the image that explicitly reasons about spatially-localized edges.

A good example of this technique is that of Srivatsan et al. (2019), which modeled images by placing a Cauchy distribution on a 2D Discrete Cosine Transform (DCT) representation of glyphs. Though this is an improvement over the default choice of placing a Normal distribution on individual pixels as it both decorrelates pixels and is tolerant of outliers, this approach is limited in its expressiveness and its ability to model spatially localized edges: Cauchy distributions are excessively heavy-tailed and so have difficulty modeling inliers, and since DCT is a global representation it does not allow the model to reason about *where* image gradient content is located.



We extend this approach in two ways (as depicted above): (1) by using a wavelet image representation instead of DCT, and (2) by using a distribution with an adaptive shape instead of a Cauchy.

Representation We opt for a wavelet representation, as unlike DCT it jointly encodes the frequency *and spatial location* of an image feature. As might be expected, an image representation in which location is directly encoded is helpful in our task; a stroke has a fundamentally different meaning at the top of a character than at the bottom. Barron (2019) quantitatively demonstrated the advantages of specifically the Cohen-Daubechies-Feauveau (CDF) 9/7 wavelet decomposition (Cohen et al., 1992) for training likelihood-based models of natural images. Based on their findings that CDF 9/7 in front of an adaptive loss achieves better performance than DCT in front of a Cauchy (the setup of Srivatsan et al. (2019)), we expect similar performance benefits in the context of our own model, and our ablations in Table 3.1 (Right) support this belief empirically.



Factor 1
Posterior approximation
$Y_i \sim E_1(\{X_{i1}, X_{i2}, \ldots\}; \phi_1)$
Factor 1
Inference network
parameters
Y_i
ϕ_1
Encoder
Inference network takes *set of observations* with shared 1st factor
Z_1
Z_2
Z_3
Y_1
Y_2
Y_3
?
?
Observations: X_{ij}
Factor 2
Posterior approximation
$Z_j \sim E_2(\{X_{1j}, X_{2j}, \ldots\}; \phi_2)$
Z_j
Factor 2
Inference network
parameters
Encoder
ϕ_2
Inference network takes *set of observations* with shared 2nd factor

Figure 3.3: Overview of the inference procedure. First the character encoder infers a representation of structure over each row, and then the font encoder infers a representation of style conditioned on a (perhaps partially observed) column and the character embeddings.

Distribution An adaptive distribution lets the model select between using leptokurtotic (Cauchy-like) distributions that are well suited to the high-frequency image edges found at the finer levels of the wavelet decomposition, or more platykurtic (Normal-like) distributions that are better suited to low-frequency DC-like average image intensities found at the coarsest levels of the wavelet decomposition. Specifically, we use the probability distribution of Barron (2019):

$$f(x|\mu,\sigma,\alpha) = \frac{\exp\left(-\frac{|\alpha-2|}{\alpha}\left(\left(\frac{(x-\mu)^2}{\sigma^2|\alpha-2|}+1\right)^{\alpha/2}-1\right)\right)}{\sigma Z(\alpha)}$$

where $Z(\alpha)$ is the distribution's partition function, and α determines the distribution's shape. As $\alpha \to 0$ the distribution approaches a Cauchy distribution, as $\alpha \to 2$ the distribution approaches a Normal distribution.

Taken together, these yield a conditional likelihood function parameterized by the decoder of our variational model, which we now describe. Given an image $X_{i,j}$, we first project it using the CDF 9/7 wavelet decomposition – which we denote as $\psi(X_{i,j})$. Because this decomposition is a biorthogonal volume-preserving transformation, it can be applied before the likelihood computation. It further serves as a whitening transformation, avoiding the need to learn a covariance matrix for $X_{i,j}$.

Our decoder outputs a grid of parameters $\hat{X}_{i,j}$, the projection of which serves as the mean μ of our adaptive distribution for scoring $\psi(X_{i,j})$. For the other distribution parameters σ and α, rather than using fixed settings we construct a set of latent variables for both: we allow each wavelet coefficient to have its own vector of latent shape parameter $\boldsymbol{\ell}^\alpha$ and scale parameter $\boldsymbol{\ell}^\sigma$, where the non-latent shape and scale are parameterized as scaled and shifted sigmoids and softplus of those latent values:

$$\alpha_k = \frac{2}{1+\exp\left(\ell_k^\alpha\right)}, \sigma_k = \frac{\log\left(1+\exp\left(\ell_k^\alpha\right)\right)}{\log(2)} + \epsilon$$

We initialize $\boldsymbol{\ell}^\alpha = \boldsymbol{\ell}^\sigma = \vec{0}$, thereby initializing $\boldsymbol{\alpha} = \boldsymbol{\sigma} = \vec{1}$. These latent variables $(\boldsymbol{\ell}^\alpha, \boldsymbol{\ell}^\sigma)$ are optimized during training using gradient descent along with all other model parameters θ, which allows the model to adapt the shape and scale of each wavelet coefficient's distribution during training. Overall, this yields the following likelihood function:

$$p(X_{i,j}|Y_i, Z_j; \theta) = \prod_k f(\psi(X_{i,j})_k | \, \psi(\hat{X}_{i,j})_k, \sigma_k, \alpha_k)$$

3.5 Learning and Inference

We now describe our approach to training this model. This process mirrors that of previous variational work, although since we are learning a dual manifold, our model will require two separate inference networks. The projected loss we add (Section 3.4) will not fundamentally affect the learning process, but does change how the reconstruction term is computed.

As our model is generative, we wish to maximize the log likelihood of the training data with respect to the model parameters, which requires summing out the unobserved variables Y and Z. However, this integral is intractable and does not permit a closed form solution. We therefore resort to optimizing a variational approximation, a strategy which has seen success in similar settings (Kingma and Welling, 2014; Srivatsan et al., 2019). Rather than directly optimize the likelihood (which we cannot compute the gradient of), we maximize a lower bound on it known as the Evidence Lower Bound (ELBO). We compute the ELBO via a function $q(Y, Z|X) = q(Y|X) * q(Z|Y, X)$ which approximates the posterior $p(Z, Y|X)$ of the distribution defined by our decoder network.

$$\text{ELBO} = \mathbb{E}_q[\log p(X|Z, Y)] - \mathbb{KL}(q(Z, Y|X)||p(Z)p(Y))$$

We define $q(Y|X)$ and $q(Z|Y, X)$ via a pair of encoder networks which operate over one row or column of the matrix respectively. An encoder passes each glyph in that row or column through a series of convolutional layers, and then max pools the output features across all glyphs, ensuring it can handle a variable number of observations (See Figure 3.3). Note that the method of pooling (*e.g.* min, max, avg), as well as the order in which to infer Y and Z are important choice points that allow for different inductive biases. The pooled feature representation is then passed through an MLP which outputs parameters μ and Σ to define a Gaussian posterior over Y_i or Z_j. Given these, we compute approximate gradients on the ELBO via the reparameterization trick described by Kingma and Welling (2014).

3.6 Experiments

We evaluate on the task of font reconstruction, in which given a small random subset of glyphs from a held out font, models must reconstruct the remaining ones. We separately report performance on known (*i.e.* observed at least once during training) and unknown character types. During training, we mask out a randomly chosen 20% of character *types* to serve as unknowns. At test time, models observe examples of previously masked characters to infer their representations for reconstruction. This can be thought of as a few-shot task, where models must generate glyphs for character types they did not observe at train time based on limited test-time examples.

3.6.1 Datasets

Capitals64, the dataset used by Azadi et al. (2018) and Srivatsan et al. (2019), only contains the 26 English capital letters, with no missing characters, meaning it does not require learning a manifold over character shape. We instead evaluate on the following datasets to best demonstrate our method's ability to scale to settings with a large number of character types and a high degree of sparsity.

Google Fonts Google Fonts is a dataset of 991 font families, which is publicly available[1]. Most fonts in the dataset support standard Latin characters, but many also support special symbols, and characters found in Greek, Cyrillic, Tamil, and several other orthographies. A visualization of this is shown in Figure 3.1. We restrict our work to the 2000 most frequently supported character types for simplicity. After removing near duplicates (described below) we are left with 2017 fonts in total, split into train, dev, and test in a $3:1:1$ ratio. The data was split by font family rather than individual fonts, to ensure that there are no fonts in train with a "sibling" in test.

Chinese Simplified We scraped a list of the most common 2000 Chinese simplified characters from the internet as well as a dataset that labels each character's radical. Together, we compile a new dataset that consists of the most common 2000 Chinese characters along with their radicals for further analysis on the character embeddings. For each Chinese character, we scraped over 1524 fonts, split similarly to Google Fonts. The total font number shrinks down to 623 after removing near duplicates, which we now discuss.

[1]https://github.com/google/fonts

Removing near duplicates One major issue with font corpora is that most fonts belong to a small handful of modes, within which there is little stylistic diversity. To ensure that our metrics best measure generalization to novel fonts unlike those seen in train, we preprocess out fonts that are extremely similar to others in the data. We first perform agglomerative clustering, and then retain only the centroid of each cluster. The number of clusters is determined by cutting the dendrogram at a height which eliminates most fonts that are to a human largely indistinguishable from their nearest neighbor.

3.6.2 Baselines

We compare our model – which we refer to as Dual Manifold – to two baselines and various ablations. Our primary baseline is EMD (Zhang et al., 2018), a discriminative encoder-decoder model that does not share embeddings across "rows" and "columns", but rather computes style and content representations for each glyph given a set of provided examples, and then passes them to a generator which constructs the final image. This model is useful for comparison as it has a similar computation graph and also learns separate embeddings for font and character shape, but computes its loss directly in pixel space, and lacks a probabilistic prior.

We also compare to a naive nearest neighbor (NN) model, which reconstructs fonts at test time by finding the font in train with the closest L_2 distance over the observed characters, and outputs that neighbor's corresponding glyphs for the missing characters. If the neighbor does not support all missing characters, we pull the remaining from the 2nd nearest neighbor, and so on. It should be noted that NN cannot reconstruct any character that is not present in train.

Similarly to EMD, the first of our ablations, denoted -KL, does not explicitly model the character and font embeddings as random variables. This effectively removes the KL divergence from the loss function, resulting in a non-probabilistic autoencoder. The next, denoted -Dual, is an ablation which treats the character representations as parameters of the model, rather than latent variables which must be inferred. This is essentially the model of Srivatsan et al. (2019) with our architecture*. We also ablate our adaptive wavelet loss against the DCT + Cauchy loss used by Srivatsan et al. (2019), denoted with -Adapt. Finally, we compare performing Max or Min pooling over elements of a row/column within the encoder network.

3.6.3 Training Details

We perform stochastic gradient descent using Adam (Kingma and Ba, 2015), with a step size of 10^{-4}. Batches contain 10 fonts, each with only 20 random characters observable to encourage robustness to the number of inputs. However at test time, the model can infer the character representations Y based on the the entire training set. We find best results when both character and style representations are $k = 256$ dimensional. See Section 3.6.4 for a full description of architecture. Our model trains on one NVIDIA 2080ti GPU in roughly a week, and is implemented in PyTorch (Paszke et al., 2017) version `1.3.1`

3.6.4 Architecture Details

The architectures of our encoder and decoder are largely identical to that of U-Net (Ronneberger et al., 2015) with key differences described here. We find significantly improved results by inserting Instance Normalization layers (Ulyanov et al., 2016b) after convolution layers in our decoder. We also replace the max pool layers within the encoder with blur pool layers (Zhang, 2019). As stated previously, we max pool the output of the encoders across character types or fonts, and then pass the flattened pooled representation through a fully connected layer to obtain the approximate posterior parameters μ and Σ. A similar fully connected layer projects the character representation Y_i to the appropriate size before being passed to the decoder. As noted earlier, the parameters of the last two transposed convolutional layers in the decoder are dynamically output by MLPs which take as input the font representation Z_j. These consist of a 256 dimensional fully connected layer, a ReLU, and then a second fully connected layer to produce the relevant parameter.

We now provide further details on the specific layer sizes used in our model and inference network. The following abbreviations are used to represent various components:

- F_i : fully connected layer with i hidden units
- R : ReLU activation
- S : sigmoid activation
- M : batch max pool
- B : 2×2 spatial blur pool (Zhang, 2019)
- C_i : convolutional layer with i filters of 3×3, 1 pixel zero-padding, stride of 1
- I : instance normalization
- T_i : transpose convolution with i filters of 2×2, stride of 2
- D_i : transpose convolution with i filters of 2×2, stride of 2, where kernel and bias are the output of an MLP (described below)
- H : reshape to $-1 \times 256 \times 8 \times 8$

Our encoder is:

$C_{64} - R - C_{64} - R - C_{64} - R - B - C_{128} - R - C_{128} - R - B - C_{256} - R - C_{256} - R - B - C_{512} - R - C_{512} - R - B - M - F_{512}$

Observations	1	8	16	32	1	8	16	32
	Google Fonts: Known Char				Google Fonts: Unknown Char			
NN	0.755	0.816	0.830	**0.839**	-	-	-	-
EMD	0.706	0.702	0.539	0.597	0.698	0.695	0.534	0.595
Dual Manifold	**0.799**	**0.828**	**0.833**	0.834	**0.801**	**0.826**	**0.829**	**0.830**
	Chinese Simplified: Known Char				Chinese Simplified: Unknown Char			
NN	**0.428**	**0.488**	**0.495**	**0.499**	-	-	-	-
EMD	0.278	0.271	0.291	0.288	0.270	0.266	0.283	0.280
Dual Manifold	0.392	0.405	0.407	0.407	**0.375**	**0.387**	**0.390**	**0.390**

Observations	1	8	16	32
+Dual, +KL, +Adapt, Min	0.7728	0.8041	0.8083	0.8088
Srivatsan et al. (2019)*	0.713	0.702	0.701	0.698
-Dual, +KL, +Adapt, Max	0.704	0.703	0.701	0.703
+Dual, -KL, +Adapt, Max	0.785	0.817	0.821	0.823
+Dual, +KL, -Adapt, Max	0.795	0.823	0.828	0.829
+Dual, +KL, +Adapt, Max	**0.799**	**0.828**	**0.833**	**0.834**

Table 3.1: (Left) SSIM per glyph by number of observed characters for both the Google Fonts and Chinese Simplified datasets. (Right) Ablations of our model, showing SSIM results on known characters for Google Fonts.

Our decoder is:

$F_{1024\times8\times8} - T_{1024} - C_{512} - I - R - C_{512} - I - R - T_{512} - C_{256} - I - R - C_{256} - I - R - D_{256} - C_{128} - I - R - C_{128} - I - R - D_{128} - C_{64} - I - R - C_{64} - I - R - C_1 - S$

MLP to compute transpose convolutional parameter of size j is:

$F_{256} - R - F_j$

3.6.5 Metrics

We measure average SSIM per glyph (Azadi et al., 2018; Gao et al., 2019), having scaled pixel intensities to $[0, 1]$. While the details of SSIM are beyond the scope of this work, it can be thought of as a feature-based metric that does not factor over individual pixels, but rather looks at the matches between higher level features regarding the structure of the image. SSIM is widely used in image processing tasks since it measures structural similarity instead of raw pixel distance, and has been shown to better correlate with human judgement than L_2 (Snell et al., 2017).

Evaluating models using L_2 can reward unrealistic reconstructions that split the difference between many hypotheses as opposed to picking just one (part of the reason we avoid training our model on such a loss). Over the course of individual training runs, we found it was almost counter correlated with human judgement, with the lowest distance early in training while output was blurry, becoming larger as the model converged. We do however include these numbers in Section 3.8, as they nonetheless support our findings.

We also perform human evaluation using Amazon Mechanical Turk. For each font in our test set, 5 turkers were shown 8 example glyphs, and a sample of reconstructions for the remaining characters by Dual Manifold and EMD. Turkers were also shown examples of each character in a neutral style. They were asked to select which if either reconstruction was better, and briefly justify their reasoning.

	Google Fonts: Known Char				Google Fonts: Unknown Char			
Observations	1	8	16	32	1	8	16	32
NN	405.15	258.11	227.91	207.04	-	-	-	-
EMD	371.06	367.18	658.85	512.26	378.08	375.19	667.66	511.69
Dual Manifold	**275.56**	**202.58**	**193.34**	**189.94**	**276.47**	**212.76**	**205.34**	**202.91**
	Chinese Simplified: Known Char				Chinese Simplified: Unknown Char			
NN	1086.58	908.58	883.18	872.52	-	-	-	-
EMD	1013.80	1019.97	1288.32	1287.85	1303.96	1020.89	1303.92	1303.48
Dual Manifold	**916.41**	**879.03**	**873.48**	**868.41**	**917.91**	**883.60**	**878.77**	**875.03**

Table 3.2: L_2 per glyph by number of observed characters for both the Google Fonts and Chinese Simplified datasets.

3.7 Results

3.7.1 Quantitative Evaluation

We list SSIM results in Table 3.1 for various numbers of observed characters. Note that NN is not capable of reconstructing character types not observed at training time. On Google Fonts, our model performs best overall; however on Chinese Simplified, we see NN winning on known characters, as well as a marked drop in SSIM overall. This could be due to the increased challenge in generating Chinese characters given the relatively higher number of strokes, leading SSIM to prefer the realism of NN, or because fonts in this dataset generally contain most characters, unlike Google Fonts which is much sparser. Observing more characters taperingly increases similarity, which matches our intuition that this allows for a better understanding of stylistic properties. Performance drops when evaluating on characters not observed in training. This makes sense as models may have less support in their manifolds for structural forms they were not trained on, but the drop is small enough to suggest our model is able to infer meaningful representations for novel character types at test time.

We see also that EMD has significant issues at 16 and 32 observed characters (it's worth noting that EMD must be separately trained for each number of observations). Qualitatively, we find certain fonts for which EMD emits the same output for every character in that font. We suspect this indicates overfitting leading to broken style representations for some novel fonts when given more observations than its default of only 10.

Within our ablations, we find that using a dual latent manifold, as opposed to treating character embeddings as model parameters, is responsible for the majority of our gain in SSIM over prior work. The next largest difference comes from using either MIN pooling within the autoencoder

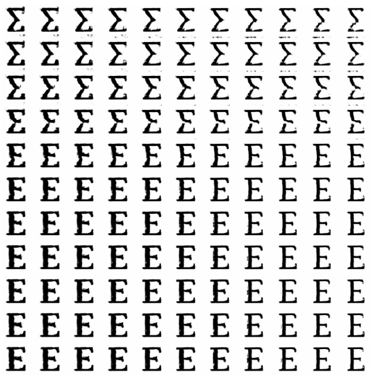


Figure 3.4: Generated glyphs for interpolating between both character type (horizontal axis) and font style (vertical axis) simultaneously.

or Max pooling. We also see more of a drop in performance from removing the KL divergence, than we do from replacing our adaptive wavelet loss with the DCT + Cauchy loss.

3.8 L_2 Results

In Table 3.2 we show results on Google Fonts and Chinese simplified for our model and baselines in terms of L_2. Rankings are generally the same, and see that our approach performs best by this metric as well as SSIM. We do however note that in places the L_2 numbers and SSIM numbers are not well correlated, and attribute this to L_2's propensity for rewarding blurry output that minimizes expected distance over sharp output that may have slightly misaligned edges.

3.8.1 Human Evaluation

In our AMT experiments, we found that for **48.2%** of known character reconstructions, turkers preferred our model's output, with 42.0% preferring EMD, and 9.8% finding both equal. For unknown character reconstructions, **50.5%** preferred ours, vs 38.7% for EMD, and 10.9% finding no difference. A majority of turkers agreed 86.3% of the time in the case of known characters, and 83.2% for unknown.

3.9 Analysis

3.9.1 Qualitative Inspection

In Figure 3.4 we show output from our model interpolating between a bold font and a light one, as well as a capital E and a Σ simultaneously. This demonstrates the smoothness of our manifolds

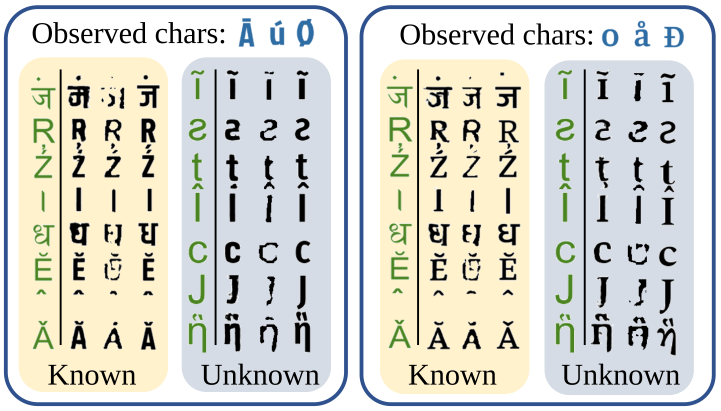


Figure 3.5: Reconstructions of two fonts from our model, EMD, and NN — shown in black in that order — for both known and unknown character types. Green characters show the expected shape in a neutral font, and blue characters are a sample of those observed by the models for either font.

and also suggests how they might offer support for font and character types not seen during training. Figure 3.5 shows examples of reconstructions by models on two fonts for a variety of both known and unknown characters. Our approach is more coherent and faithful than EMD, and NN is realistic but often stylistically incorrect.

3.9.2 Chinese Radicals

Figure 3.6 shows t-SNE projections (Maaten and Hinton, 2008; Van Der Maaten, 2014) of learned character embeddings colored by their radical, a sub-component of Chinese characters. Radicals like 丨, 丿, and 火 — which either share forms with others or can occur in different structures — don't cluster together, while unique radicals like 心, 刂, and 攵 do.

3.10 Broader Impacts

As our work can be used to augment or even replace the labor of human artists, it is worth discussing its potential broader impacts. The most obvious positive is that this technique can add value to font designers, by minimizing the overhead required to design a font that supports widespread internationalization. Our model's ability to interpolate stylistic properties can also make it easy to automatically generate completely novel fonts that are roughly similar to existing ones.

This also benefits speakers of languages that rely on less common glyphs, as it broadens their font selection. It can make it easier for them to both produce and consume digital content, allowing for better accessibility for demographics that currently have fewer options for orthographies

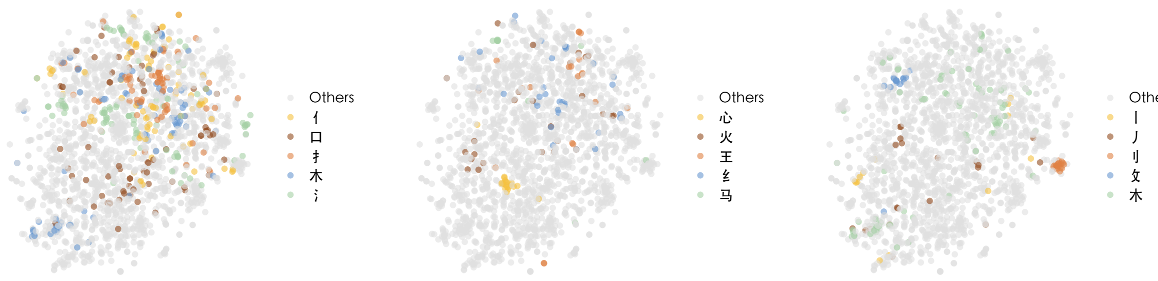


Figure 3.6: t-SNE plot of Chinese character embeddings from our model for the top 5 radicals (left), and randomly chosen groups of 5 (middle, right).

they are most familiar with.

One potential negative impact is on the business of some font artists who cater to niche audiences that have less common glyph needs. Our model could potentially be used to replace such workers, and if so could also lead to less coherent renderings for uncommon orthographies if those who are not fluent in such scripts simply employ our system without a thorough understanding of the types of errors it may make.

3.11 Conclusion

In this chapter we introduced a generative model for typography capable of reconstructing characters in a novel font, of a novel shape, or both, and demonstrated its improvements over previous approaches on two datasets containing large numbers of characters. We analyzed the results qualitatively, and inspected learned manifolds for smoothness.

In future work, this methodology has potential value not just to fonts, but to any domain which can also be factored over independent axes of variation, such as handwriting by different authors. One could also incorporate this model into more complex downstream tasks such as OCR. That being said, these domains also feature complex interactions between physically adjacent glyphs (our model treats different characters within a font as conditionally independent), so some further innovation would likely still be required.

There are also extensions to the model itself that might be worth exploring in future work, for instance operating on a stroke-based representation in order to perform reconstruction in the original TTF space instead of raw pixel space as we do here. This would also likely assist with smoothness of edges and reduce the incidence of "corroded" output glyphs.

Acknowledgements

This project is funded in part by the NSF under grants 1618044 and 1936155, and by the NEH under grant HAA256044-17.

Chapter 4

Neural Representation Learning for Scribal Hands of Linear B

4.1 Introduction

In the previous chapter, we showed how our matrix factorization approach can be extended to handle reconstruction for large, sparse datasets of modern digital fonts. In this chapter, we will demonstrate a similar method applied to what is in a way the opposite setting: a small dataset of handwritten characters in an ancient writing system.

As mentioned previously, neural methods have seen much success in extracting high level information from visual representations of language. Many tasks from OCR, to handwriting recognition, to font manifold learning have benefited greatly from relying on architectures imported from computer vision to learn complex features in an automated, end-to-end trainable manner. However much of the emphasis of this field of work has been on well-supported modern languages, with significantly less attention paid towards low-resource, and in particular, historical scripts. While self-supervised learning has seen success in these large scale settings, in this work we demonstrate its utility in a low data regime for an applied analysis task. Specifically, we propose a novel neural framework for script analysis and present results on scribal hand representation learning for Linear B.

Linear B is a writing system that was used on Crete and the Greek mainland ca. 1400-1200 BCE. It was used to write Mycenaean, the earliest dialect of Greek, and was written on leaf or page-shaped clay tablets for accounting purposes. Linear B is a syllabary, with approximately 88 syllabic signs as well as a number of ideograms, which are used to indicate the commodity represented by adjacent numerals. The sites which have produced the most material to date are Knossos, Pylos, Thebes, and Mycenae (see Palmer (2008) for more detail) but for the purposes of this chapter, we focus specifically on Knossos and Pylos from which we collect a dataset of images of specific instances of glyphs written by 74 different scribal hands. Many signs exhibit slight variations depending on which scribe wrote them. Therefore, being able to uncover patterns in how different scribes may write the same character can inform us about the scribes' potential connections with one another, and to an extent even the ways in which the writing system evolved over time. This is a process that has previously been performed by hand (Skelton, 2008), but to which we believe neural methods can provide new insights.

Rather than building representations of each scribe's writing style based on the presence of sign variations as determined by a human annotator, we propose a novel neural model that disentangles features of glyphs relating to sign shape and scribal idiosyncrasies directly from the raw images. Our model accomplishes this by modeling each image using two separate real-valued vector embeddings — we share one of these embeddings across glyphs written by the same scribe, and the other across glyphs depicting the same sign, thereby encouraging them to capture relevant patterns. These embeddings are learned via three networks. The first is a decoder that attempts to reconstruct images of glyphs from their corresponding embeddings using a series of transpose convolutional layers. We penalize the reconstructed output based on its mean squared error (MSE) vs. the original. In addition, we also use two discriminator networks, which attempt to predict whether a given image-embedding pair actually correspond to one another or not, both for scribe and sign embeddings respectively. These losses are backpropagated into those original vectors to encourage them to capture distinguishing properties that we care about.

This model has many potential downstream applications, as paleography has long been an integral part of the study of the Linear B texts (Palaima, 2011). For a handful of examples from prior work, Bennett's study of the Pylos tablets (Bennett Jr, 1947) revealed the existence of a number of different scribal hands, all identified on the basis of handwriting, since scribes did not sign their work. This research was carried out even before the decipherment of Linear B. Later, Driessen (2000) used the archaic sign forms on Linear B material from the Room of the Chariot Tablets at Knossos, in conjunction with archaeological evidence, to argue that these tablets pre-dated the remainder of the Linear B tablets from Knossos. Skelton (2008) and later Skelton et al. (2016) carried out a phylogenetic analysis of Linear B based on discrete paleographical characteristics of the sign forms, and were able to use this analysis to help date the remainder of the Linear B tablets from Knossos, whose relative and absolute dates had already been in question. Our hope is that leveraging the large capacity of neural architectures can eventually lead to further strides in these areas.

Additionally, qualitative assessments of Linear B paleography have long been used to date tablets whose history or findplaces are uncertain. This has been vital in the case of the Knossos tablets, which were excavated before archaeological dating methods were well-developed (Firth et al., 2016), and the tablets from the Pylos megaron, which seem to date to an earlier time period than the rest of the palace (Skelton, 2011). While qualitative techniques have been fruitful, a key motivating question in our work is to what extent we can use quantitative methods to improve on these results.

Going back further, Linear B was adapted from the earlier writing system Linear A, which remains undeciphered. There is a hundred-year gap between the last attestation of Linear A and the first attestation of Linear B. With a more thorough understanding of the evolution of these two writing systems, can we reconstruct the sign forms of the earliest Linear B, and elucidate the circumstances behind its invention? With this in mind, our approach explicitly incorporates a decoder network capable of reconstructing glyphs from learned vector representations, that we hope can eventually be useful for such investigation.

In total, three writing systems related to Linear B remain undeciphered (Cretan Hieroglypic, Linear A, and Cypro-Minoan), and a fourth (the Cypriot Syllabary) was used to write an unknown language. One motivating question is whether a better understanding of the evolution of sign forms would help contribute additional context that could make a decipherment possible. In particular, scholars are not yet in agreement over the sign list for Cypro-Minoan, and how many distinct languages and writing systems it may represent (Ferrara, 2012). Understanding variation in sign forms is foundational for this work (Palaima, 2011), and it makes sense to test our methods on a related known writing system (i.e. Linear B) before attempting a writing system where so much has yet to be established.

Overall these are just some of the applications that we hope the steps presented here can be directed towards.

4.2 Prior Work

One of our primary points of comparison is the work of Skelton (2008), who manually defined various types of systematic stylistic variations in the way signs were written by different scribal hands, and used these features to analyze similarity between scribes via phylogenetic systematics. In a sense this work can serve as our human baseline, giving insight into what we might expect reasonable output from our system to look like, and helping us gauge performance in an unsupervised domain without a clear notion of ground truth.

While neural methods have not previously been used in analyzing this particular writing system, there is established literature on applying them to typography more broadly. For example, Srivatsan et al. (2019) used a similar shared embedding structure to learn a manifold over font styles and perform font reconstruction. There has also been extensive work on writer identification from handwritten samples (Bulacu and Schomaker, 2007; Siddiqi and Vincent, 2010; Zhang et al., 2016; Djeddi et al., 2018), as well as writer retrieval (Christlein et al., 2019). How-

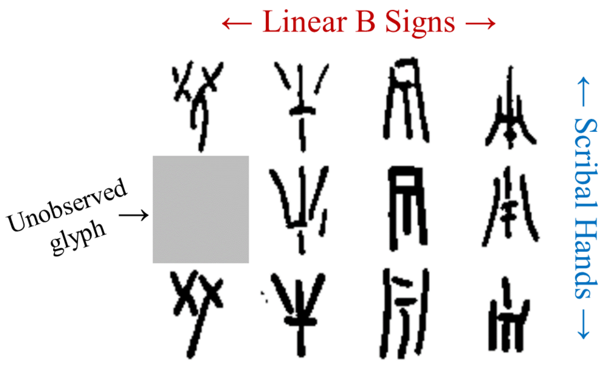


Figure 4.1: Binarized examples from our collected dataset. Note the stylistic differences particularly in the left and right most signs, and that we do not have an observation for the left most sign in the middle scribe's hand.

ever these approaches rely on supervised discriminative training, in contrast to ours which is not explicitly trained on predicting authorship, but rather learning features.

4.3 Methods

In this section we describe the layout of our model and its training procedure. At a high level, our goal is to train it on our dataset of images of glyphs, and in doing so learn vector embeddings for each scribe that encode properties of their stylistic idiosyncrasies — for example which particular variations of each sign they preferred to use — while ignoring more surface level information such as the overall visual silhouettes of the signs we happen to observe written by them. To do this, we break the roles of modeling character shape and scribal stylistic tendencies into separate model parameters, as we now describe.

Our model, depicted in Figure 4.2, assigns each sign and each scribal hand a vector embedding to capture the structural and stylistic properties of glyphs in our data. These embeddings are shared by glyphs with the same sign or scribe, effectively factorizing our dataset into a set of vectors, one for each row and column. The loss is broken into two parts, a reconstruction loss and a discriminative loss. The reconstruction loss is computed by feeding a pair of sign and scribe embeddings to a decoder, which then outputs a reconstruction of the original glyph that those embeddings correspond to. A simple MSE is computed between this and the gold image. The discriminative cross-entropy loss is computed by the discriminator, a binary classifier which given an image of a glyph and a scribe embedding must predict whether or not the two match (i.e. did that scribe write that glyph). We train this on equally balanced positive and negative pairs. A similar discriminator is applied to the sign embeddings. In our experiments, we ablate these three losses and examine the effects on performance.

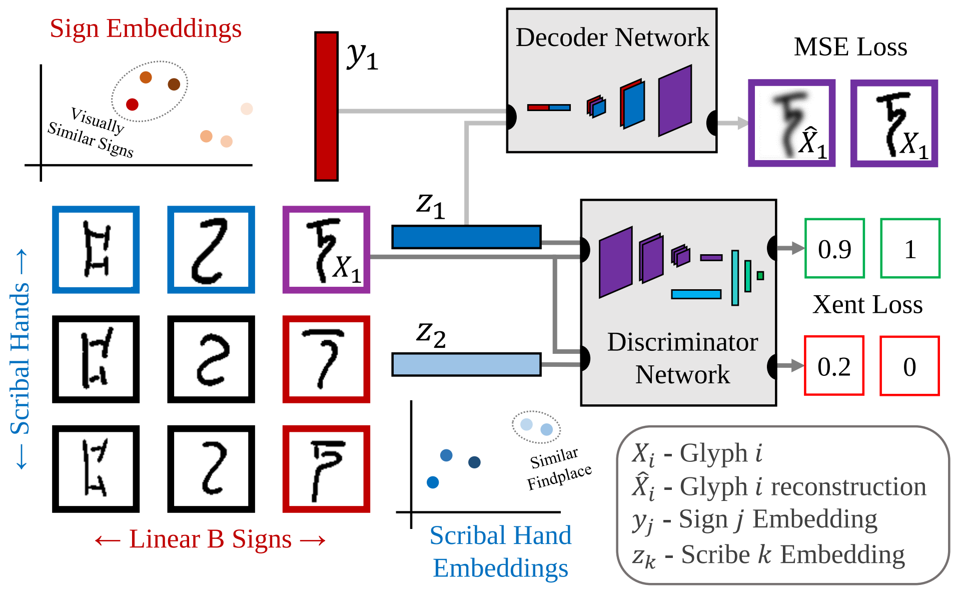


Figure 4.2: Diagram of the model — each row is assigned a scribe embedding, and each column is assigned a sign embedding, which feed into a decoder and discriminator to collectively compute the loss function.

More formally, suppose we have a corpus of images of glyphs X, where $X_i \in [0,1]^{64\times 64}$ depicts one of J characters and is authored by one of K scribes. We define a set of vectors Y, where each $Y_j \in \mathbb{R}^d$ represents features of sign j. Similarly we define a set of vectors Z, where each $Z_k \in \mathbb{R}^d$ represents features of scribe k. Thus, a given X_i depicting sign j and authored by scribe k is represented at a high level in terms of its corresponding Y_j and Z_k, each of which is shared by other images corresponding to the same character or scribal hand respectively.

For a given X_i, the decoder network takes in the corresponding pair of vectors Y_j and Z_k, and then passes them through a series of transpose convolutional layers to produce a reconstructed image $\hat{X}_i$. This reconstruction incurs a loss on the model based on its MSE compared to the original image.

Next, the scribal hand discriminator with parameters ϕ passes X_i through several convolutional layers to obtain a feature embedding which is then concatenated with the corresponding Z_k and passes through a series of fully connected layers, ultimately outputting a predicted $p(Z_k|X_i;\phi)$ which scores the likelihood of the scribal hand represented by Z_k being the true author of X_i. Similarly, a sign discriminator with parameters ψ predicts $p(Y_j|X_i;\psi)$, i.e. the likelihood of X_i depicting the sign represented by Y_j. These discriminators are shown both one true match and one false match for an X_i at every training step, giving us a basic cross-entropy loss.

These components all together yield the following loss function for our model:

$$\begin{aligned} L(X_i, Y_j, Z_k) =& \|X_i - \hat{X}_i\|_2 \\ &+\lambda_1 * (\log p(Y_j|X_i;\phi) + \log(1 - p(Y_{j'}|X_i;\phi))) \\ &+\lambda_2 * (\log p(Z_k|X_i;\psi) + \log(1 - p(Z_{k'}|X_i;\psi))) \end{aligned}$$

where j' and k' are randomly selected such that $j \neq j'$ and $k \neq k'$.

The format of this model lends itself out of the box to various downstream tasks. For example, when a new clay tablet is discovered, it may be useful to determine if the author was a known scribal hand or someone completely new. The discriminator of our model can be used for exactly such as a purpose, as it can be directly queried for the likelihood of a new glyph matching each existing scribe. Even if this method is not exact enough to provide a definitive answer, we can at least use the predictions to get a rough sense of who the most similar known scribes are, despite never having seen that tablet during training.

Also, while our model treats the scribe and sign embeddings as parameters to be directly learned, one could easily imagine an extension in which embeddings are inferred from the images using an encoder network. Taking this further, we could even treat those embeddings as latent variables by adding a probabilistic prior, such as in Variational Autoencoders (Kingma and Welling, 2014). This would let us compute embeddings without having trained on that scribe's writing, taking the idea from the previous paragraph one step further, and letting us perform reconstructions of what we expect a full set of signs by that new scribe to look like.

4.3.1 Architecture

We now describe the architecture of our model in more detail. Let F_i represent a fully connected layer of size i, T represent a 2x2 transpose convolutional layer, C be a 3x3 convolution, I be an instance normalization layer (Ulyanov et al., 2016b), R be a ReLU, M be a 2x2 blur pool (Zhang, 2019). The decoder network is then $F_{1024*4*4} \rightarrow 4 \times (T \rightarrow 2 \times (C \rightarrow I \rightarrow R))$, where each convolutional layer C reduces the number of filters by half. The discriminator's encoder network is then $C \rightarrow 4 \times (3 \times (C \rightarrow R) \rightarrow M) \rightarrow F_{16}$, where each convolutional layer C increases the number of filters by double, except for the first which increases it to 64. Having encoded the input image, the discriminator concatenates the output of its encoder with the input scribe/sign embedding, and then passes the result through 7 fully connected layers to obtain the final output probability.

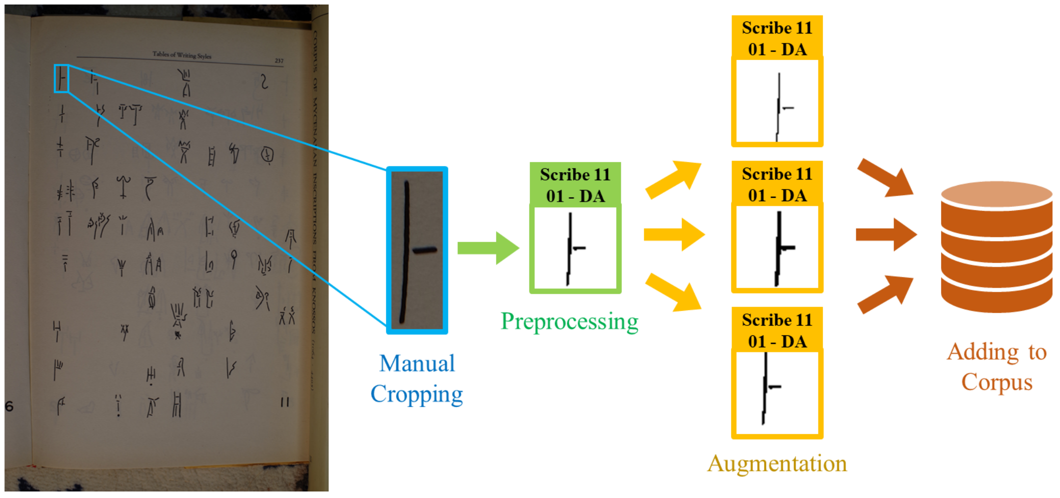


Figure 4.3: Visualization of the data pipeline. Glyphs are (1) manually cropped from photos of pages and labeled by sign and scribe, (2) preprocessed to $64x64$ binary images, and (3) augmented to produce multiple variants, before being added to the overall corpus.

4.4 Data

4.4.1 Collection

There has been previous work on transcribing the tablets of Knossos by Olivier (1965) and Pylos by Palaima (1988). These authors present their transcriptions in form of a rough table for each hand containing every example of each sign written by them. While these are an important contribution and certainly parseable by a human reader, they are not of a format readily acceptable by computer vision methods. Since our model considers the atomic datapoint to be an individual glyph, we must first crop out every glyph from each table associated with a scribal hand. This is a nontrivial task as the existing tables are not strictly grid aligned, and many glyphs overlap with one another, which requires us to manually paint out the encroaching neighbors for many images. Ultimately we were able to convert the existing transcriptions into a digital corpus of images of individual glyphs, labeled both by scribal hand as well as sign type. This process was carried out by two graduate students under the guidance of a domain expert. Before being fed into our model, images were also binarized via Otsu's thresholding (Otsu, 1979).

The digitization of this dataset into a standardized, well-formatted corpus is of much potential scientific value. While this process was laborious, it opens the doors not just to our own work, but also to any other future research hoping to apply machine learning methods to this data. Of course had this been a high-resource language such as English, we would have been able to make use of the multitude of available OCR toolkits to automate this preprocessing step. As such, one possible direction for future work is to train a system for glyph detection and segmentation, for

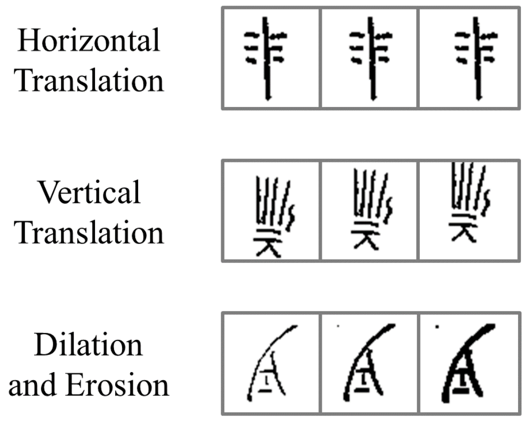


Figure 4.4: Examples of the three types of data augmentation we perform. Each original image is augmented in all possible ways, resulting in an effective 27x increase in the size of our corpus.

which the corpus we have already created can serve as a source of supervision. This would be useful in more efficiently cropping similarly transcribed data, whether in Linear B or other related languages.

4.4.2 Augmentation

Due to the limited size of this corpus, there is a serious risk of models overfitting to spurious artifacts in the images rather than properties of the higher level structure of the glyphs. To mitigate this, we perform a data augmentation procedure to artificially inflate the number of glyphs by generating multiple variants of each original image that differ in ways we wish for our model to ignore. These are designed to mimic forms of variation that naturally occur in our data — for example, glyphs may not always be perfectly centered within the image due to human error during cropping, and strokes may appear thicker or thinner in some images based on the physical size of the sign being drawn relative to the stylus' width, or even as a result of inconsistencies in binarization during preprocessing. Following these observations, the specific types of augmentation we use include translation by 5 pixels in either direction horizontally and/or vertically, as well as eroding or dilating with a 2x2 pixel wide kernel (examples shown in Figure 4.4). All together we produce 27 variations of each image. Doing this theoretically encourages our model to learn features that are invariant to these types of visual differences, and therefore more likely to capture the coarser stylistic properties we wish to study.

4.4.3 Statistics

Our dataset represents 88 Linear B sign types, written by 74 scribal hands (of these 48 are from Knossos and 26 are from Pylos). For each scribal hand, we observe an average of 56 glyph images, some which may represent the same sign. Conversely each sign appears an average of 47 times across all scribal hands. This gives us a total of $4,134$ images in our corpus, which after performing data augmentation increases to $111,618$.

4.5 Experiments

Having trained our model, we need to be able to quantitatively evaluate the extent to which it has learned features reflective of the sorts of stylistic patterns we care about, something which is nontrivial given the lack of any supervised ground truth success criteria. There are two metrics we use to accomplish this. First, given that we expect scribes of the same findplace to share stylistic commonalities, we train a findplace classifier based on our scribal hand embeddings to probe whether our model's embeddings contain information that is informative for this downstream task. Secondly, since we have human generated scribal hand representations in the form of Skelton's manual features (Skelton, 2008), we can measure the correlation between them and our model's learned manifold to see how well our embeddings align with human judgements of relevant features. We now describe these evaluation setups in more detail.

4.5.1 Findplace Classification

The first way we evaluate our model is by probing the learned embeddings to see how predictive they are of findplace, a metadata attribute not observed during training. Since our goal is to capture salient information about broad similarities between scribal hands, one coarse measure of our success at this is to inspect whether our model learns similar embeddings for scribes of the same findplace, as they would likely share similar stylistic properties. One reason for this underlying belief is that such scribes would have been contemporaries of each other, as can be infered from the fact that existing Linear B tablets were written "days, weeks, or at most months" before the destruction of the palaces (Palaima, 2011). For any scribes working together at the same place and time, we observe similar writing styles, likely as a result of the way in which they would have been trained. The number of scribes attested at any one palatial site is quite low, which would seem to rule out a sort of formal scribal school as attested in contemporary Near Eastern sources, and instead suggests that scribal training took place as more of a sort of apprenticeship, or as a profession passed from parents to children, as is attested for various sorts of craftsmen in the Linear B archives (Palaima, 2011). Thus, we expect that scribes found at the same findplace would exhibit similar writing styles, and therefore measuring how predictive our model's embeddings are of findplace serves as a reasonable proxy for how well they encode stylistic patterns.

To this end, we train a supervised logistic regression classifier (one per model) to predict the corresponding findplaces from the learned embeddings, and evaluate its performance on a held out set. In principal, the more faithfully a model's embeddings capture distinguishing properties of the scribal hands, the more accurately the classifier can predict a scribe's findplace based solely on those neural features. We perform $k = 5$ fold cross validation, and report average F_1 score over the 5 train/test splits. Our final numbers reflect the best score over 15 random initializations of the classifier parameters for each split.

Findplaces for the Knossos tablets were described in Melena and Firth (2019), and for Pylos in Palaima (1988). However, there are some findplaces from which we only have one or two scribal hands in our corpus. As supervised classification is not practically possible in such cases, we exclude any findplaces corresponding to 2 or fewer scribal hands from our probing experiments. This leaves us with 63 scribal hands, labeled by 8 different findplaces to evaluate on.

4.5.2 QVEC

While findplace prediction is a useful downstream metric, we also wish to measure how well the neural features our model learns correlate with those extracted by expert humans. In order to do this, we use `QVEC` (Tsvetkov et al., 2015) — a metric originally developed for comparing neural word vectors with manual linguistic features — to score the similarity between our learned scribal hand embeddings and the manual features of Skelton (2008). At a high level, `QVEC` works by finding the optimal one-to-many alignment between the dimensions of a neural embedding space and the dimensions of a manual feature representation, and then for each of the aligned neural and manual dimensions, summing their Pearson correlation coefficient with respect to the vocabulary (or in our case the scribal hands). Therefore, this metric provides an intuitive way for us to quantitatively evaluate the extent to which our learned neural features correlate with those extracted by human experts. As a side note, while the original `QVEC` algorithm assumes a one-to-many alignment between neural dimensions and manual features, we implement it as a many-to-one, owing to the fact that in our setting we have significantly more manual features than dimensions in our learned embeddings, implying that each neural dimension is potentially responsible for capturing information about multiple manual features rather than vice versa as is the case in the original setting of Tsvetkov et al. (2015).

4.5.3 Baselines

Our primary baseline is a naive autoencoder, that assumes a single separate embedding for every image in the dataset, therefore not sharing embeddings across signs or scribal hands. Its encoder

Model	**Findplace Classifier F_1**	`QVEC` **Similarity to Manual**
Most Common	0.154	-
Autoencoder	0.234	54.1
-Recon, +Scribe, +Sign	0.198	53.2
+Recon, -Scribe, -Sign	0.234	51.1
+Recon, +Scribe, -Sign	0.206	52.9
+Recon, +Scribe, +Sign	**0.292**	**57.0**

Table 4.1: (Left) Best cross validated F_1 achieved by logistic regression findplace classifier on embeddings produced by ablations of our model, compared to an autoencoder, and naive most common label baseline. (Right) `QVEC` (Tsvetkov et al., 2015) score between models' embeddings and manual features of Skelton (2008).

layout is the same as that of our discriminator, and its decoder is identical to that of our main model so as to reduce differences due to architecture capacity. In order to obtain an embedding for a scribal hand using this autoencoder, we simply average the embeddings of all glyphs produced by that scribe. We report several ablations of our model, indicating the presence of the reconstruction loss, scribal hand discriminator loss, and sign discriminator loss by +Recon, +Scribe, and +Sign respectively. Finally, we provide the F_1 score for a naive model that always predicts the most common class to indicate the lowest possible performance.

4.5.4 Training Details

We use sign and scribe embeddings of size 16 each for all experiments. Our model is trained with the Adam (Kingma and Ba, 2015) optimization algorithm at a learning rate of 10^{-4} and a batch size of 25. It is implemented in Pytorch (Paszke et al., 2017) version `1.8.1` and trains on a single NVIDIA 2080ti in roughly one day. The classifier used for evaluation is trained via SGD at a learning rate of 10^{-3} and a batch size of 15.

4.6 Results

Table 4.1 shows results of a findplace classifier trained on our system's output vs. that of an autoencoder, with the score for a most common baseline as a lower bound. We see that our full model with all three losses achieves the highest score, with reduced performance as they are successively removed. The worst score comes from the model trained on purely discriminative losses with no decoder, suggesting that the reconstruction loss provides important bias to prevent

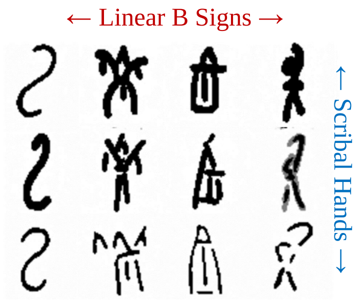


Figure 4.5: Example reconstructions from our model for various scribal hands. The model manages to capture subtle stylistic differences, but we see a failure mode in the middle row right column where the output appears as a superposition of two images.

overfitting given the limited dataset size. This also makes sense given the strong performance by the vanilla autoencoder.

For reference, Skelton (2008) provides manual feature representations for 20 of the scribal hands in our corpus (17 of which are from the 6 findplaces shared by at least one other scribe in that set). If we perform a similar probing experiment with the hand crafted features for those 17 scribes, we get an F_1 score of 0.37. While this result is useful as an indicator of where the upper bound on this task may be, it is very important to note that this experiment is only possible on a small subset of the scribes we evaluate our model on and is therefore not directly comparable to the results in Table 4.1. A more direct comparison to the neural features of our model would require extracting manual features for the remaining 54 scribal hands, which is beyond the scope of this work.

Table 4.1 also shows the `QVEC` similarity between each of the models' learned embeddings and the manual features of Skelton (2008). Once again, our system ranks highest, with the autoencoder also performing more strongly than our ablations. Interestingly, we see here that our discriminative losses appear more important than the reconstructive loss, suggesting that by encouraging the model to focus on features that are highly distinguishing between scribes or signs we may end up recovering more of the same information as humans would.

Figure 4.5 shows example reconstructions from our decoder. These reconstructions are based solely on the scribe and sign representations for the corresponding row and column. We see that the model is able to realistically reconstruct the glyphs, showing that it is very much capable of learning shared representations for sign shape and scribal hand styles, which can then be recombined to render back the original images. This would imply that our embedding space is able to adequately fit the corpus, and perhaps requires some additional inductive bias in order to encourage those features to be more high level.

There are also additional downstream tasks we have not yet evaluated on that may be worth attempting with our system. The most obvious is that of phylogeny reconstruction. Now that we have trained scribal hand embeddings, it remains to be seen how similar a phylogenetic reconstruction based on these features would resemble that found by Skelton (2008). This may give us a better idea as to whether the embeddings we have computed are informative as to the broader hierarchical patterns in the evolution of Linear B as a writing system.

4.7 Conclusion

In this chapter we introduced a novel approach to learning representations of scribal hands in Linear B using a new neural model which we described. To that end, we digitized a corpus of glyphs, creating a more standardized dataset more readily amenable to machine learning methods going forwards. We benchmarked the performance of this approach compared to simple baselines and ablations by quantitatively measuring how predictive the learned embeddings were of held out metadata regarding their findplaces. In addition, we directly evaluated the similarity of our scribal hand representations to those from existing manual features. We believe this work leaves many open avenues for future research, both with regard to the model design itself, as well as to further tasks it can be applied towards. For example, one of the key applications of manual features in the past has been phylogenetic analysis, something which our neural features could potentially augment for further improvements. This type of downstream work can further be used in service of dating Linear B tablets, understanding the patterns in the writing system's evolution, building OCR systems, or even analyzing other related but as of yet undeciphered scripts.

Part II

Modeling Sequential Human Data with Temporal Dependencies

Chapter 5

Modeling Online Discourse with Coupled Distributed Topics

5.1 Introduction

In the previous part of this thesis, we described how making use of implicit links in typographic data can allow us to build factorized representations and perform style transfer and font reconstruction. Now we will refocus on a new domain, where the information flow between datapoints will take place at the level of local context, and we will therefore need to perform inference in a more hierarchical manner. Specifically we'll be transitioning from the domain of typography to that of social media threads. On the surface this may appear quite different, but we find that our general paradigm of using deep variational learning to extract asymmetrical embeddings based on structural associations is in fact still quite applicable.

Topic models have become one of the most common unsupervised methods for uncovering latent semantic information in natural language data, and have found a wide variety of applications across the sciences. However, many common models - such as Latent Dirichlet Allocation (Ng and Jordan, 2003) - make an explicit exchangeability assumption that treats documents as independent samples from a generative prior, thereby ignoring important aspects of text corpora which are generated by non-ergodic, interconnected social systems. While the direct application of such models to datasets such as transcripts of The French Revolution (Barron et al., 2017) and discussions on Twitter (Zhao et al., 2011) have yielded sensible topics and exciting insights, their exclusion of document-to-document interactions imposes limitations on the scope of their applicability and the analyses they support. For instance, on many social media platforms, comments are short (the average Reddit comment is 10 words long), making them difficult to treat as full documents, yet they do cohere as a collection, suggesting that contextual relationships should be considered. Moreover, analysis of social data is often principally concerned with understanding

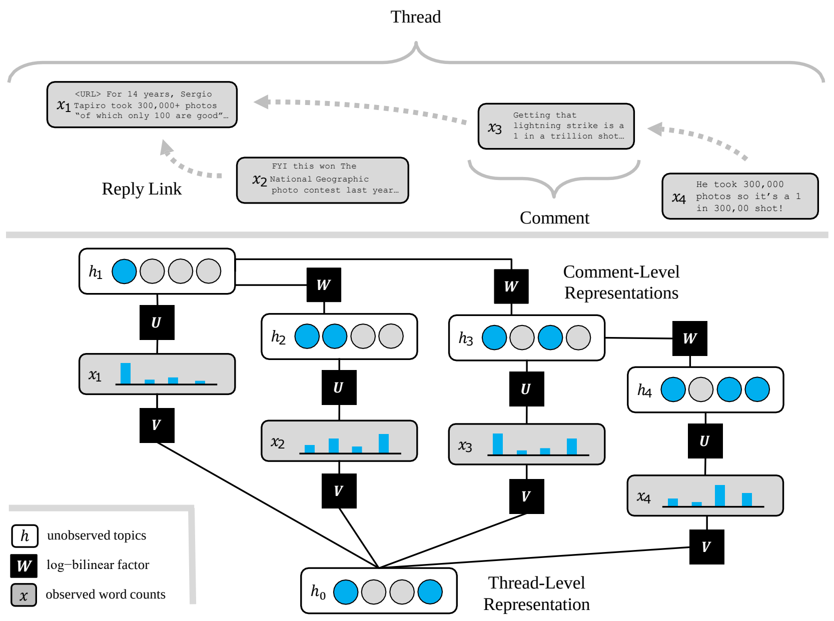

Thread
x_1 <URL> For 14 years, Sergio Tapiro took 300,000+ photos "of which only 100 are good"…
x_2 FYI this won The National Geographic photo contest last year…
x_3 Getting that lightning strike is a 1 in a trillion shot…
x_4 He took 300,000 photos so it's a 1 in 300,00 shot!
Reply Link
Comment
Comment-Level Representations
Thread-Level Representation
h_1
h_2
h_3
h_4
h_0
x_1
x_2
x_3
x_4
W
U
V
unobserved topics
log−bilinear factor
observed word counts

Figure 5.1: DDTM factor graph for an example thread. Each comment is modeled as an observed bag-of-words x with topics represented by a latent binary vector h. Log-bilinear factors connect the latent and observed variables of each comment, and the latent variables of parent-child comment pairs along observed reply links. Biases are omitted for clarity.

relationships between documents (such as question-asking and -answering), so a model able to capture such features is of direct scientific relevance.

To address these issues, we propose a design that models representations of comments jointly along observed reply links. Specifically, we attach a vector of latent binary variables to each comment in a collection of social data, which in turn connect to each other according to the observed reply-link structure of the dataset. The inferred representations can provide information about the rhetorical moves and linguistic elements that characterize an evolving discourse. An added benefit is that while previous work such as Sequential LDA (Du et al., 2012) has focused on modeling a linear progression, the model we present applies to a more general class of acyclic graphs such as tree-structured comment threads ubiquitous on the web.

Online data can be massive, which presents a scalability issue for traditional methods. Our approach uses latent binary variables similar to a Restricted Boltzmann Machine (RBM); related models such as Replicated Softmax (RS) (Salakhutdinov and Hinton, 2009) have previously seen success in capturing latent properties of language, and found substantial speedups over previous methods due to their GPU amenable training procedure. RS was also shown to deal well with documents of significantly different length, another key characteristic of online data. While RBMs permit exact inference, the additional coupling potentials present in our model make inference intractable. However, the choice of bilinear potentials and latent features admits a mean-field inference procedure which takes the form of a series of dense matrix multiplications followed by nonlinearities, which is particularly amenable to GPU computation and lets us scale efficiently to large data.

Our model outperforms LDA and RS baselines on perplexity and downstream tasks including metadata prediction and document retrieval when evaluated on a new dataset mined from Reddit. We also qualitatively analyze the learned topics and discuss the social phenomena uncovered.

5.2 Model

We now present an overview of our model. Specifically, it will take the probabilistic form of an undirected graphical model whose architecture mirrors the tree structure of the threads in our data.

5.2.1 Motivating Dataset

We evaluate on a corpus mined from Reddit, an internet forum which ranks as the fourth most trafficked site in the US (Alexa, 2018) and sees millions of daily comments (Reddit, 2015). Discourse on Reddit follows a branching pattern, shown in Figure 5.1. The largest unit of discourse is a thread, beginning with a link to external content or a natural language prompt, posted to a relevant subreddit based on its subject matter. Users comment in response to the original post (OP), or to any other comment. The result is a structure which splits at many points into more specific or tangential discussions that while locally coherent may differ substantially from each other. The data reflect features of the underlying memory and network structure of the generating process; comments are serially correlated and highly cross-referential. We treat individual comments as “documents” under the standard topic modeling paradigm, but use observed reply structure to induce a tree of documents for every thread.

5.2.2 Description of Discursive Distributed Topic Model

We now introduce the Discursive Distributed Topic Model (DDTM) (illustrated in Figure 5.1). For each comment in the thread, DDTM assigns a latent vector of binary random variables (or *bits*) that collectively form a distributed embedding of the topical content of that comment; for instance, one bit might represent sarcastic language while another might track usage of specific acronyms - a given comment could have any combination of those features. These representations are tied to those of parent and child comments via coupling potentials (see Section 5.2.3), which allow them to learn discursive properties by inducing a deep undirected network over the thread. In order to encourage the model to use these *comment-level* representations to learn discursive and stylistic patterns as opposed to simply topics of discussion, we incorporate a single additional latent vector for the entire thread that interacts with each comment, explaining word choices that are mainly topical rather than discursive or stylistic. As we demonstrate in our experiments (see Section 5.6) the *thread-level* embedding learns distributions more reminiscent of what a traditional topic model would uncover, while the comment-level embeddings model styles of speaking and mannerisms that do not directly indicate specific subjects of conversation. The joint probability is defined in terms of an energy function that scores latent embeddings and observed word counts across the tree of comments within a thread using log-bilinear potentials, and is globally normalized over all word count and embedding combinations.

5.2.3 Probability Model

More formally, consider a thread containing N comments each of size D_n with a vocabulary of size K. As depicted in Figure 5.1, each comment is viewed as a bag-of-words, densely connected via a log-bilinear potential to a latent embedding of size F. Let each comment be represented as as an integer vector $x_n \in \mathbb{Z}^K$ where x_{nk} is number of times word k was observed in comment n, and let $h_n = \{0,1\}^F$ be the topic embedding for each comment, and let $h_0 = \{0,1\}^F$ be the embedding for the entire thread. To model topic transitions, we score the embeddings of parent-child pairs with a separate coupling potential as shown in Figure 5.1 (comments with no parents or children receive additional start/stop biases respectively). Let replies be represented with sets R, P_N, and C_N where $(n,m) \in R$ and $n \in P_m$ and $m \in C_n$ if comment m is a reply to comment n. DDTM assigns probability to a specific configuration of x, h with an energy function scored by the emission (π_e) and coupling (π_c) potentials.

$$\begin{aligned} E(x,h;\theta) &= \underbrace{\sum_{n=1}^{N} \pi_\text{e}(h,x,n)}_{\text{Emission Potentials}} + \underbrace{\sum_{(n,m)\in R} \pi_\text{c}(h,n,m)}_{\text{Coupling Potentials}} \\ \pi_\text{e}(h,x,n) &= h_n^\intercal U x_n + x_n^\intercal a + D_n h_n^\intercal b \\ &\quad + h_0^\intercal V x_n + D_n h_0^\intercal c \\ \pi_\text{c}(h,n,m) &= h_n^\intercal W h_m \end{aligned} \tag{5.1}$$

Note that the bias on embeddings is scaled by the number of words in the comment, which controls for their highly variable length. The joint probability is computed by exponentiating the energy and dividing by a normalizing constant.

$$
\begin{aligned}
p(x, h; \theta) &= \frac{\exp(E(x, h; \theta))}{Z(\theta)} \\
Z(\theta) &= \sum_{x', h'} \exp(E(x', h'; \theta))
\end{aligned}
\tag{5.2}
$$

This architecture encourages the model to learn discursive maneuvers via the coupling potentials while separating within-thread variance and across-thread variance through the comment-level and thread-level embeddings respectively. The coupling of latent variables makes factored inference impossible, meaning that even the exact computation of the partition function is no longer tractable. This necessitates approximating the gradients for learning which we will now address.

5.3 Learning and Inference

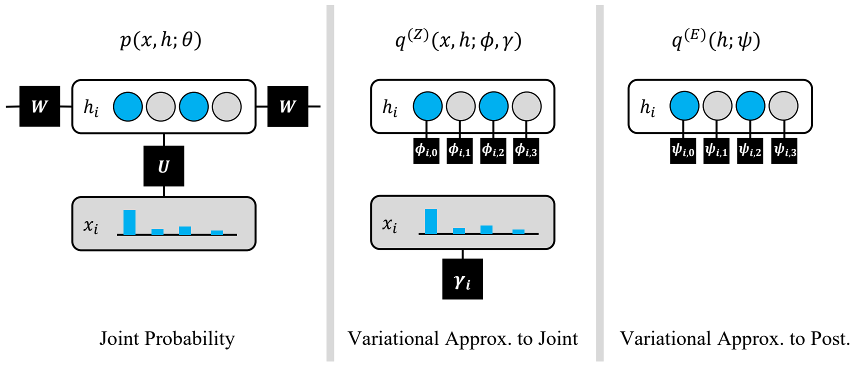

$p(x, h; \theta)$
W
h_i
W
U
x_i
Joint Probability
$q^{(Z)}(x, h; \phi, \gamma)$
h_i
$\phi_{i,0}$ $\phi_{i,1}$ $\phi_{i,2}$ $\phi_{i,3}$
x_i
γ_i
Variational Approx. to Joint
$q^{(E)}(h; \psi)$
h_i
$\psi_{i,0}$ $\psi_{i,1}$ $\psi_{i,2}$ $\psi_{i,3}$
Variational Approx. to Post.

Figure 5.2: Factor graph of full joint compared to mean-field approximations to joint and posterior.

Inference in this model class in intractable, so as has been done in previous work on topic modeling (Ng and Jordan, 2003) we rely on variational methods to approximate the gradients needed during training as well as the posteriors over the topic bit vectors. Specifically, we will need the gradients of the normalizer and the sum of the energy function over the hidden variables

$$
E(x; \theta) = \log \sum_{h} \exp(E(x, h; \theta)) \tag{5.3}
$$

which we refer to as the *marginal energy*. Following the approach described for undirected models by Eisner (2011), we approximate these quantities and their gradients with respect to the model parameters θ as we will now describe (thread-level embeddings are omitted in this section for clarity).

5.3.1 Normalizer Approximation

We aim to train our model to maximize the marginal likelihood of the observed comment word counts, conditioned on the reply links. To do this we must compute the gradient of the normalizer $Z(\theta)$. However, this quantity is computationally intractable, as it contains a summation over all exponential choices for every word in the thread. Therefore, we must approximate $Z(\theta)$. Observe that under Jensen's Inequality, we can form the following lower bound on the normalizer using an approximate joint distribution $q^{(\mathrm{Z})}$.

$$\begin{aligned} \log Z(\theta) &= \log \sum_{x,h} \exp(E(x,h;\theta)) \\ &\geq \mathbb{E}_{q^{(\mathrm{Z})}}[E(x,h;\theta)] - \mathbb{E}_{q^{(\mathrm{Z})}}[\log q^{(\mathrm{Z})}(x,h;\phi,\gamma)] \end{aligned} \tag{5.4}$$

We now define $q^{(\mathrm{Z})}$ as depicted in Figure 5.2 as a mean-field approximation that treats all variables as independent. We parameterize $q^{(\mathrm{Z})}$ with $\phi_{nf} \in [0,1]$, independent Bernoulli parameters representing the probability of h_{nf} being equal to 1, and γ_{nk} replicated softmaxes representing the probability of a word in x_n taking the value k. Note that all words in x_n are modeled as samples from this single distribution. The approximation then factors as follows:

$$\begin{aligned} q^{(\mathrm{Z})}(x,h;\phi,\gamma) &= q^{(\mathrm{Z})}(x;\gamma) \cdot q^{(\mathrm{Z})}(h;\phi) \\ q^{(\mathrm{Z})}(x;\gamma) &= \prod_{n=1}^{N} \prod_{k=1}^{K} (\gamma_{nk})^{x_{nk}} \\ q^{(\mathrm{Z})}(h;\phi) &= \prod_{n=1}^{N} \prod_{f=1}^{F} \left((\phi_{nf})^{h_{nf}} (1-\phi_{nf})^{(1-h_{nf})} \right) \end{aligned} \tag{5.5}$$

We optimize the parameters of $q^{(\mathrm{Z})}$ to maximize its variational lower bound, via iterative mean-field updates, which allow us to perform coordinate ascent over the parameters of $q^{(\mathrm{Z})}$. Maximizing the lower bound with respect to particular ϕ_{nf} and γ_{nk} while holding all other parameters frozen, yields the following mean-field update equations (biases omitted for clarity):

$$\begin{aligned} \phi_{n\cdot} &= \sigma\left(U\gamma_n + \sum_{m \in C_n} W\phi_m + \phi_{P_n}^{\intercal} W \right) \\ \gamma_{n\cdot} &= \sigma\left(\phi_n^{\intercal} U \right) \end{aligned} \tag{5.6}$$

We iterate over the parameters of $q^{(\mathrm{Z})}$ in an "upward-downward" manner; first updating ϕ for all comments with no children, then all comments whose children have been updated, and so on up to the root of the thread. Then we perform the same updates in reverse order. After updating all ϕ, we then update γ simultaneously (the components of γ are independent conditioned on ϕ). We iterate these upward-downward passes until convergence.

5.3.2 Marginal Energy Approximation

We can now approximate the normalizer, but still need the marginal data likelihood in order to take gradient steps on it and train our model. In order to recover the marginal likelihood, we must next approximate the marginal energy $E(x;\theta)$ as it too is intractable. This is due to the coupling potentials, which make the topics across comments dependent even when conditioned on the word counts. To do this, we form an additional variational approximation (see Figure 5.2) to the marginal energy, which we optimize similarly.

$$\begin{aligned} E(x;\theta) &= \log \sum_h \exp(E(x,h;\theta)) \\ &\geq \mathbb{E}_{q^{(\mathrm{E})}}[E(x,h;\theta)] - \mathbb{E}_{q^{(\mathrm{E})}}[\log q^{(\mathrm{E})}(h;\psi)] \end{aligned} \tag{5.7}$$

Since $q^{(\mathrm{E})}(h;\psi)$ need only model the hidden units h, we can parameterize it in the same manner as $q^{(\mathrm{Z})}(h;\phi)$. Note that while these distributions factor similarly, they do not share parameters, although we find that in practice, initializing $\phi \leftarrow \psi$ improves our approximation. We optimize the lower bound on $E(x;\theta)$ via a similar coordinate ascent strategy, where the mean-field updates take the following form (biases omitted for clarity):

$$\psi_{n\cdot} = \sigma\left(Uh_n + \sum_{m \in C_n} W\psi_m + \psi_{P_n}^{\intercal} W\right) \tag{5.8}$$

We can use $q^{(\mathrm{E})}$ to perform inference at test time in our model, as its parameters ψ directly correspond to the expected values of the hidden topic embeddings under our approximation.

5.3.3 Learning via Gradient Ascent

We train the parameters of our true model $p(x,h;\theta)$ via stochastic updates wherein we optimize both approximations on a single datum (i.e. thread) to compute the approximate gradient of its log-likelihood, and take a single gradient step on the model parameters (repeating on all training instances until convergence). That gradient is given by the difference in feature expectations under the approximations (entropy terms from the lower bounds are dropped as they do not depend on θ).

$$\begin{aligned}\nabla \log p(x;\theta) \approx\ & \mathbb{E}_{q^{(\mathrm{E})}(h;\psi)}\left[\nabla E(x,h;\theta)\right] \\ & - \mathbb{E}_{q^{(\mathrm{Z})}(x',h;\psi)}\left[\nabla E(x',h;\theta)\right]\end{aligned} \tag{5.9}$$

In summary, we use two separate mean-field approximations to compute lower bounds on the marginal energy $E(x,h;\theta)$, and its normalizer $Z(\theta)$, which lets us approximate the marginal likelihood $p(x;\theta)$. Note that as our estimate on the marginal likelihood is the difference between two lower bounds, it is not a lower bound itself, although in practice it works well for training.

5.3.4 Scalability and GPU Implementation

Given the magnitude of our dataset, it is essential to be able to train efficiently at scale. Many commonly used topic models such as LDA (Ng and Jordan, 2003) have difficulty scaling, particularly if trained via MCMC methods. Improvements have been shown from online training (Hoffman et al., 2010), but extending such techniques to model comment-to-comment connections and leverage GPU compute is nontrivial.

In contrast, our proposed model and mean-field procedure can be scaled efficiently to large data because they are amenable to GPU implementation. Specifically, the described inference procedure can be viewed as the output of a neural network. This is because DDTM is globally normalized with edges parameterized as log-bilinear weights, which results in the mean-field updates taking the form of matrix operations followed by nonlinearities. Therefore, a single iteration of mean-field is equivalent to a forward pass through a recursive neural network, whose architecture is defined by the tree structure of the thread. Multiple iterations are equivalent to feeding the output of the network back into itself in a recurrent manner, and optimizing for T iterations is achieved by unrolling the network over T timesteps. This property makes DDTM highly amenable to efficient training on a GPU, and allowed us to scale experiments to a dataset of over 13M total Reddit comments.

5.4 Experimental Setup

5.4.1 Data

We mined a corpus of Reddit threads pulled through the platform's API. Focusing on the twenty most popular subreddits (gifs, todayilearned, CFB, funny, aww, AskReddit, BlackPeopleTwitter, videos, pics, politics, The_Donald, soccer, leagueoflegends, nba, nfl, worldnews, movies, mildlyinteresting, news, gaming) over a one month period yielded $200,000$ threads consisting of $13,276,455$ comments total. The data was preprocessed by removing special characters, replacing URLs with a domain-specific token, stemming English words using a Snowball English Stemmer (Porter, 2001), removing stopwords, and truncating the vocabulary to only include the top $10,000$ most common words. OPs are modeled as a comment at the root of each thread to which all top-level comments respond. This dataset will be made available for public use after publication.

5.4.2 Baselines and Comparisons

We compare to baselines of Replicated Softmax (RS) (Salakhutdinov and Hinton, 2009) and Latent Dirichlet Allocation (LDA) (Ng and Jordan, 2003). RS is a distributed topic model similar to our own, albeit without any coupling potentials. LDA is a locally normalized topic model which defines topics as non-overlapping distributions over words. To ensure that DDTM does not gain an unfair advantage purely by having a larger embedding space, we divide the dimensions equally between comment- and thread-level. Unless specified 64 bits/topics were used. We experiment with RS and LDA treating either comments or full threads as documents.

5.4.3 Training and Initialization

SGD was performed using the Adam optimizer (Kingma and Ba, 2015). When running inference, we found convergence was reached in an average of 2 iterations of updates. Using a single NVIDIA Titan X (Pascal) card, we were able to train our model to convergence on the training set of 10M comments in less than 30 hours. It is worth noting that we found DDTM to be fairly sensitive to initialization. We found best results from Gaussian noise, with comment-level emissions at variance of 0.01, thread-level emissions at 0.0001, and transitions at 0. We initialized all biases to 0 except for the bias on word counts, which we set to the unigram log-probabilities from the train set.

5.5 Results

5.5.1 Evaluating Perplexity

We compare models by perplexity on a held-out test set, a standard evaluation for generative and latent variables models.

	Perplexity (nats)			
Bits	32	64	96	128
RS (thr)	2240	2234	2233	2257
RS (cmt)	1675	1894	2245	2518
DDTM (-cpl)	2027	1704	1766	1953
DDTM	**1624**	**1590**	**1719**	**713**

Table 5.1: Perplexity of DDTM with and without coupling potentials (-cpl) vs. baselines trained at comment (cmt) or thread (thr) level across various numbers of topics and bits. For reference, a unigram model achieves 2644.

Setup: Due to the use of mean-field approximations for both the marginal energy and normalizer we lose any guarantees regarding the accuracy of our likelihood estimate (both approximations are lower bounds, and therefore their difference is neither a strict lower bound nor guaranteed to be unbiased). To evaluate perplexity in a more principled way, we use Annealed Importance Sampling (AIS) to estimate the ratio between our model's normalizer and the tractable normalizer of a base model from which we can draw true independent samples as described by Salakhutdinov and Murray (2008). Note that since the marginal energy is intractable in our model, unlike a standard RBM, we must sample the joint - and not the marginal - intermediate distributions. This yields an unbiased estimate of the normalizer. The marginal energy must still be approximated via a lower bound, but given that AIS is unbiased and empirically low in variance, we can treat the overall estimate as a lower bound on likelihood for evaluation. Using 2000 intermediate distributions, and averaging over 20 runs, we evaluated per-word perplexity over a set of 50 unseen threads. Results are shown in Table 5.1.

Results: DDTM achieves the lowest perplexity at all dimensionalities. Note our ablation with the coupling potentials removed (-cpl), increases perplexity noticeably, indicating that modeling replies helps beyond simply modeling threads and comments jointly, particularly at larger embeddings. For reference, a unigram model achieves 2644. We find that LDA's approximate perplexity is even worse, likely due to slackness in its lower bound.

5.5.2 Upvote Regression

To measure how well embeddings capture comment-level characteristics, we feed them into a linear regression model that predicts the number of upvotes the comment received. Upvotes provide a loose human-annotated measure of likability. We expect that context matters in determining how well received a comment is; the same comment posted in response to different parents may receive a very different number of upvotes. Hence, we expect comment-level embeddings to be

Task	**Upvote Regr.** (MSE)	**Deletion Pred.** (% acc.)
LDA (thr)	1.952	68.35
LDA (cmt)	2.047	59.26
RS (thr)	2.024	69.92
RS (cmt)	2.007	66.45
DDTM	**1.933**	**70.39**

Table 5.2: Performance of DDTM vs. Replicated Softmax (RS) and Latent Dirichlet Allocation (LDA) at predicting upvotes and child deletion.

more informative for this task when connected via our model's coupling potentials.

Setup: We trained a standard linear regressor for each model. The regressor was trained using ordinary least squares on the entire training set of comments using the model's computed topic embeddings as input, and the number of upvotes on the comment as the output to predict. As a preprocessing step, we took the log of the absolute number of votes before training. We compared models by mean squared error (MSE) on our test set. Results are shown in Table 5.2.

Results: DDTM achieves lowest MSE. To assess statistical significance, we performed a 500 sample bootstrap of our training set. The standard errors of these replications are small, and a two-sample t-test rejects the null hypothesis that DDTM has an average MSE equal to that of the next best method ($p < .001$). Note that our model outperforms both comment- and thread-level embeddings, suggesting that modeling these jointly, and modeling the effect of neighboring representations in the comment graph, more accurately learns information relevant to a comment's social impact.

5.5.3 Deletion Prediction

Comments that are excessively provocative or in violation of site rules are often deleted, either by the author or a moderator. We can measure whether DDTM captures discursive interactions that lead to such intervention by training a logistic classifier that predicts whether any of a given comment's children have been deleted.

Setup: For each model, a logistic regression classifier was trained stochastically with the Adam optimizer on the entire training set of comments using the model's computed topic embeddings as input, and a binary label for whether the comment had any deleted children as the output to predict. We compared models by accuracy on our test set. Results are shown in Table 5.2.

Results: DDTM gets the highest accuracy. Interestingly, thread-level models do better than comment-level ones, which suggests that certain topics or even subreddits may correlate with

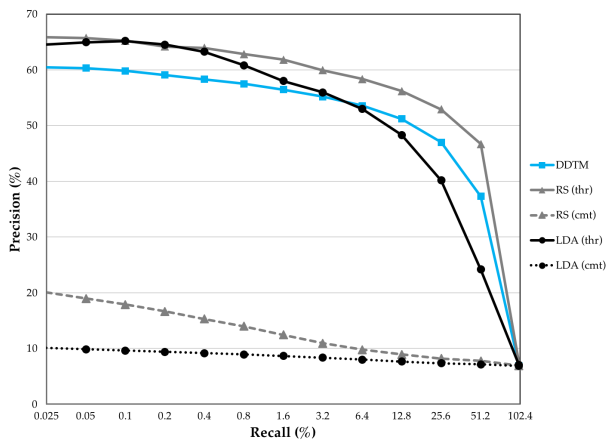


Figure 5.3: Precision vs. recall for document retrieval based on subreddit comparing various models for 1000 randomly selected held-out query comments.

comments being deleted. This makes sense given that subreddits vary in severity of moderation. DDTM's performance also demonstrates that modeling comment-to-comment interaction patterns is helpful in predicting when a comment will spawn a deleted future response, which strongly matches our intuition.

5.5.4 Document Retrieval

Finally, while DDTM is not designed to better capture topical structure, we evaluate the extent to which it can still capture this information by performing document retrieval, a standard evaluation, for which we treat the subreddit to which a thread was posted as a label for relevance. Note that every comment within the same thread belongs to the same subreddit, which gives thread-level models an inherent advantage at this task. We include this task purely for the purpose of demonstrating that by capturing discursive patterns, DDTM does not lose the ability to model thread-level topics as well.

Setup: Given a query comment from our held-out test set, we rank the training set by the Dice similarity of the hidden embeddings computed by the model. We consider a retrieved comment relevant to the query if they both originate from the same subreddit, which loosely categorizes the semantic content. Tuning the number of documents we return allows us to form precision recall curves, which we show in Figure 5.3.

Results: DDTM outperforms both comment-level baselines and is competitive with thread-level models, even beating LDA at high levels of recall. This indicates that despite using half of its dimensions to model comment-to-comment interactions DDTM can still do almost as good a job of modeling thread-level semantics as a model using its entire capacity to do so. The gap between comment-level RS and LDA is also consistent with LDA's known issues dealing with

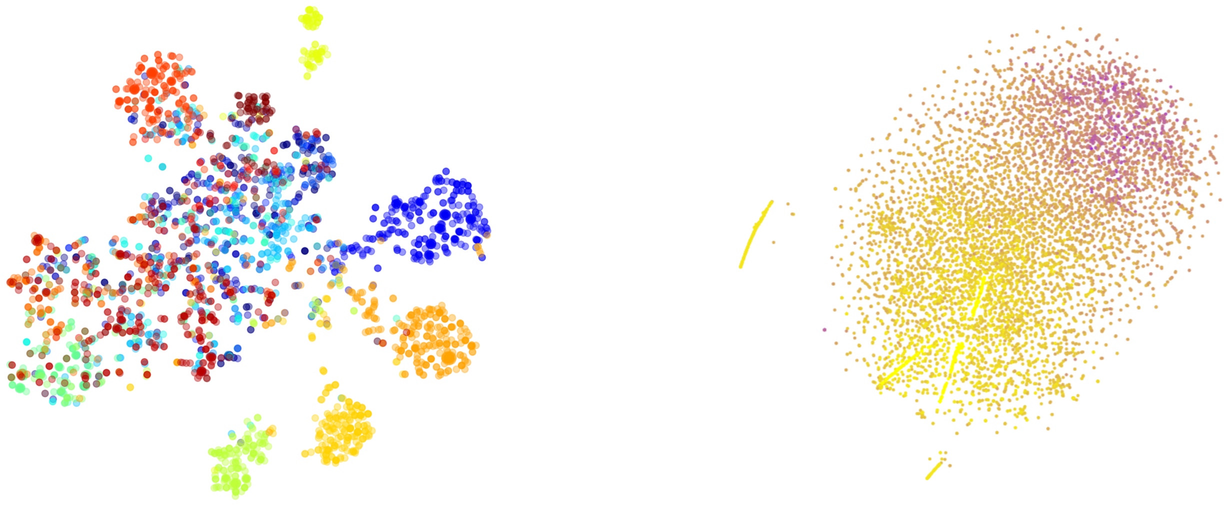


Figure 5.4: (Right) t-SNE visualization of a random sample of DDTM thread-level embeddings colored by subreddit (not observed in training) (Left) t-SNE visualization of a random sample of DDTM comment-level embeddings colored by log of comment length (darker is longer).

sparse data (Sridhar, 2015), and lends credence to our theory that distributed topic representations are better suited to such domains.

5.6 Qualitative Analysis of Topics

We now offer qualitative analysis of the topic embeddings learned by our model. Note that since we use distributed embeddings, our bits are more akin to filters than complete distributions over words, and we typically observe as many as half of them active for a single comment. In a sense, we have an exponential number of topics, whose parameterization simply factors over the bits. Therefore, it can be difficult to interpret them as one would interpret topics learned by a model such as LDA. Furthermore, we find that in practice this effect is correlated with the topic embedding size; the more bits our model has, the less sparse and consequently less individually meaningful the bits become. Therefore for this analysis, we specifically focus on DDTM trained with 64 bits total.

5.6.1 Bits in Isolation

Directly inspecting the emission parameters, reveals that the comment-level and thread-level halves of our embeddings capture substantially different aspects of the data (shown in Table 5.3) akin to vertical, within-thread, and horizontal, across-thread sources of variance respectively. The comment-level topic bits tend to reflect styles of speaking, lingo, and memes that are not

Bit #	Associated Word Stems by Emission Weight (Higher Score → Lower Score)
Comment-Level	
Bit 1	`faq tldrs pms 165 til keyword questions feedback chat pm`
2	`irl riamverysmart legend omfg riski aboard favr madman skillset tunnel`
3	`lotta brah ouch spici oof bummer buildup viewership hd uncanni`
4	`funniest mah tfw teleport fav hoo plz bah whyd dumbest`
5	`handsom hipster texan hottest whore norwegian shittier scandinavian jealousi douch`
Thread-Level	
Bit 1	`btc gameplay tutori cyclist dev currenc kitti bitcoin rpg crypto`
2	`url_youtu url_leagueoflegends url_businessinsider url_twitter url_redd url_snopes`
3	`comey pede macron pg13 maga globalist ucf committe cuck distributor`
4	`maduro venezuelan ballot puerto catalonia rican quak skateboard venezuela quebec`
5	`nra scotus opioid cheney nevada metallica marijuana vermont colorado xanax`

Table 5.3: Words with the highest emission weight for various comment-level and thread-level bits.

unique to a particular subject of discourse or even subreddit. For example, comment-level Bit 2 captures many words typical of taunting Reddit comments; replying with "/r/iamverysmart" (a subreddit dedicated to mocking people who make grandiose claims about their intellect) is a common way of jokingly implying that the author of the parent comment takes themselves too seriously — and thus corresponds to a certain kind of rhetorical move. Further, it is grouped with other words that indicate related rhetorical moves; calling a user "risky" or a "madman" is a common means of suggesting that they are engaging in a pointless act of rebellion. They also cluster at the coarsest level by length (see Figure 5.4 left) which we find to correlate with writing style.

By contrast, the thread-level bits are more indicative of specific topics of discussion, and unsurprisingly they cluster by subreddit (see Figure 5.4 right). For example, thread-level Bit 3 captures lexicon used almost exclusively by alt-right Donald Trump supporters as well as the names of various political figures. Bit 4 highlights words related to civil unrest in Spanish speaking parts of the world.

5.6.2 Bits in Combination

While these distributions over words (particularly for comment-level bits) can seem vague, when multiple bits are active, their effects compound to produce much more specific topics. One can think of the bits as defining soft filters over the space of words, that when stacked together carve out patterns not apparent in any of them individually. We now analyze a few sample topic embeddings. To do this, we perform inference as described on a held-out thread, and pass the comment-level topic embedding for a single sampled comment through our emission matrix and inspect the words with the highest corresponding weight (shown in Table 5.4). In generative terminology, these can be thought of as reconstructions of comments.

Sample #	Associated Word Stems by Emission Weight (Higher Score → Lower Score)
Comment-Level	
Sample 1	`grade grader math age 5th 9th 10th till mayb 7th`
2	`repost damn dope bamboozl shitload imagin cutest sad legendari awhil`
3	`heh dawg hmm spooki buddi aye m8 aww fam woah`
4	`hug merci bless tfw prayer pleas dear bear banana satan`
5	`chuckl cutest funniest yall bummer oooh mustv coolest ok oop`
6	`cutest heard coolest funniest havent seen ive craziest stupidest weirdest`
7	`reev keanu christoph murphi walken vincent chris til wick roger`
8	`moron douchebag stupid dipshit snitch jackass dickhead idioci hypocrit riddanc`
9	`technic actual realiz happen escal werent citat practic memo cba`
10	`reddit shill question background user subreddit answer relev discord guild`

Table 5.4: Words with the highest emission weight for sample held-out comment reconstructions.

These topic embeddings capture more specific conversational and rhetorical moves. For example, Sample 6 displays supportive and interested reactionary language, which one might expect to see used in response to a post or comment linking to media or describing something intriguing. This is of note given that one of the primary aims of including coupling potentials was to encourage DDTM to learn "topics" that correspond to responses and interactive behavior, something existing methods are largely not designed for. By contrast, Sample 9 captures a variety of hostile language and insults, which unlike those discussed previously do not denote membership in a particular online community. As patterns of toxic and hateful behavior on Reddit are more well-studied (Chandrasekharan et al., 2017), it could be useful to have a tool to analyze precipitous contexts and parent comments, something which we hope systems based on coupling of comment embeddings have the capacity to provide. Sample 10 is of particular interest as it consists largely of Reddit terminology. Conversations about the meta of the site can manifest for example in users accusing each other of being "shills" (i.e. accounts paid to astroturf on behalf of external interests) or requesting/responding to "guilding", a feature which lets users purchase premium access for each other often in response to a particularly well made comment.

5.7 Related Work

Many topic models such as LDA (Ng and Jordan, 2003) treat documents as independent mixtures, yet this approach fails to model how comments interact with one another throughout a larger discourse if such connections exist in the data. Other work has considered modeling hierarchy in topics (Griffiths et al., 2004). These models form hierarchical representations of topics themselves, but still treat documents as independent. While this approach can succeed in learning topics of various granularities, it does not explicitly track how topics interact in the context of a nested conversation.

Some approaches such as Pairwise-Link-LDA and Link-PSLA-LDA (Nallapati et al., 2008) attempt to model interactions among documents in an arbitrary graph, albeit with important drawbacks. The former models every possible pairwise link between comments, and the latter models links as a bipartite graph, limiting its ability to scale to large tree-structured threads. Similar work on Topic-Link LDA (Liu et al., 2009) models link probabilities conditioned on both topic similarity and an authorship model, yet this approach is poorly suited to high volume, semi-anonymous online domains. Other studies have leveraged reply-structures on Reddit in the context of predicting persuasion (Hidey and McKeown, 2018), but DDTM differs in its generative, unsupervised approach.

DDTM's emission potentials are similar to those of Replicated Softmax (Salakhutdinov and Hinton, 2009), an undirected model based on a Restricted Boltzmann Machine. Unlike LDA-style models, RS does not assign a topic to each word, but instead builds a distributed representation. In this setting, a single word can be likely under two different topics, both of which are present, and lend probability mass to that word. LDA-style models by contrast would require the topics to compete for the word.

5.8 Conclusion

In this chapter we introduced a novel way to learn topic interactions in observed discourse trees, and describe GPU-amenable learning techniques to train on large-scale data mined from Reddit. We demonstrated improvements over previous models on perplexity and downstream tasks, and offered qualitative analysis of learned discursive patterns. The dichotomy between the two levels of embeddings hints at applications in style-transfer.

Chapter 6

Checklist Models for Improved Output Fluency in Piano Fingering Prediction

6.1 Introduction

Previously we presented an approach that learned contextualized topic representations of comments in a hierarchical, temporally ordered thread. In this chapter, we extend that idea to the domain of music. Here, instead of using a variational strategy, we will shift to a checklist approach that passes messages forward in time in order to enforce fluencies in an output label space.

While sheet music is often very specific as to *what* notes a musician must play, it can also be vague about *how* to play them. Composers generally write notation that focuses on describing the desired musical output, but leave the specifics of how to achieve this with their instrument up to the performer, as these details are outside the scope of their role in the creative process or in some cases beyond their expertise. The piano is an instrument which requires an extensive amount of decision-making on the performer's part. In order to perform a piece of music, a pianist must either consciously or intuitively select which finger to use to play every note in the song. Notes may overlap in duration or even have simultaneous onsets. Since any key on the piano can potentially be played by any finger, there are an exponential number of ways one could perform any given piece; of these however, the majority would be uncomfortable, difficult to play at full tempo, or even physically impossible (Parncutt et al., 1997). The challenge of deciding the most ergonomic fingering to use can be nontrivial for less experienced pianists, who would not yet have the ability to quickly map common musical patterns to conventional fingering strategies, and might produce more consistent and accurate performances if provided with them (Sloboda et al., 1998).

However recent work has shown that the use of machine learning techniques may be able to assist in this regard (Nakamura et al., 2020; Moryossef et al., 2019; Zhao et al., 2022; Barbara et al., 2021). Automatic piano fingering prediction is the task of inferring finger assignments for each note in a symbolic representation of a piano song, given knowledge of which hand is meant to play each note. Human players generally prefer fingerings that balance physical comfort and efficiency (Sloboda et al., 1998), yet these criteria are difficult to quantify and highly subjective (Clarke et al., 1997). Decisions are simultaneously constrained by the current placement of the musician's fingers, and by how that decision may in turn affect possible options for the notes that follow. And while the spatial location of the notes on the keyboard is arguably the most salient factor, timing is also crucial; certain fingerings might be easy for a set of consecutive notes, but impossible if the same notes are in a chord.

Prior approaches to this task have largely treated it as a sequence tagging problem, where the input at each step is simply the pitch of the note and the output is a softmax over indices corresponding to each finger, and have employed methods analogous to those used for part of speech tagging such as LSTMs and HMMs. However, these techniques are limited in their capacity to model output dependencies, and also disregard the amount of elapsed time between notes which can greatly affect a prediction's performability.

We propose an autoregressive approach, where prior predictions are fed back into the network in order to encourage the model to produce fingerings that are not just accurate on average, but also locally *fluent* and therefore playable. We do this using a checklist which indicates which fingers have either recently been or are currently in use, and where they are on the keyboard relative to the note at the current timestep. By directly exposing this information, the model can more easily learn to restrict its predictions to fingers that are actually free, and also make decisions more directly influenced by the hand's physical placement. We also experiment with an approach that only feeds back in the most recent prediction, as well as an autoregressive tagger which maintains a neural representation of prior predictions using an encoder network.

Furthermore, the evaluation metrics used in prior work do not always correlate with what a pianist might intuitively perceive as "good" predictions. In fact there are many types of desirable (or undesirable) patterns that may emerge in a model's output, which a simplistic metric such as labeling precision may not actually reflect. We demonstrate that it is possible for two models with nearly identical labeling precision to differ dramatically in the frequency of specific types of output subsequences that are physically challenging to play. Therefore, we evaluate on a battery of metrics that collectively convey a more holistic and interpretable overview of fingering quality.

Since these metrics are not always correlated with the labeling precision that a cross entropy loss optimizes, we also investigate incorporating them explicitly into our loss function at train time. While these scores are non-differentiable, we show that reinforcement learning techniques—which have previously seen success in sequence generation tasks outside of music—can successfully optimize them. This, in conjunction with our checklist approach, ensures that the model not only has direct access to the information it would need to avoid undesirable output patterns, but is also encouraged to do so.

Finally, we conduct human evaluations and qualitative analysis which confirm that our approach improves predicted fingerings in ways consistent with our modeling choices, and also suggest directions for future work.

In summary our contributions are as follows: (1) We provide a comparison of various input representations for LSTM models (2) We put forward checklist based approaches which directly incorporate information from previous decisions (3) We introduce several additional metrics which track fluency of output, and demonstrate how to train directly on them using reinforcement learning.

6.2 Related Work

While constraint-based models for automatic piano fingering have long existed (Parncutt et al., 1997), the standardization as a machine learning task which we follow was formalized by Nakamura et al. (2020), which introduced the PIano fingerinG (PIG) dataset, put forward LSTM and HMM baselines (following prior work using HMMs (Nakamura et al., 2014; Yonebayashi et al., 2007)), and has since been followed up on by others. Moryossef et al. (2019) demonstrated the value of pretraining on even a noisy automatically annotated dataset, although they focused on basic LSTM models and evaluated exclusively on labeling precision. Zhao et al. (2022) put forward the idea of representing inputs based on relative difference in pitch rather than absolute pitch, and also showed improvements from the use of a constrained transition matrix, albeit one that was built off of prior knowledge of the task. Barbara et al. (2021) recast fingering as an information retrieval problem, although they focused on the Czerny corpus instead of the PIG dataset.

Our work also draws from research into implicitly learning output structure by feeding prior decisions into the computation of future predictions. There are classic examples of graphical models that condition predictions both on inputs at the current timestep and outputs at the previous (McCallum et al., 2000). Recent work on explicit checklists in neural settings, such as by Kiddon et al. (2016) found success using a structured agenda that tracked ingredient usage in conditional cooking recipe generation. Our checklists are however more transient; outputs can be removed from the checklist if enough time has passed. We also track not just the presence of outputs, but information about how they were used.

There is also much work on the use of REINFORCE (Williams, 2004) for sequence generation tasks. While Ranzato et al. (2016) performed several tasks including summarization, translation, and image captioning, but required a critic model for stabilization, Rennie et al. (2017) developed

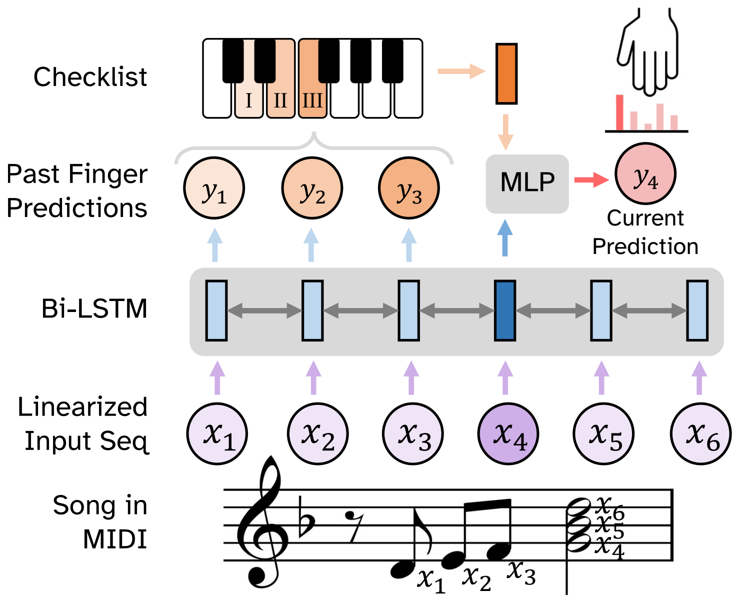


Figure 6.1: Visualization of the checklist model. Notes are embedded and then passed to a Bi-LSTM which outputs contextualized embeddings at each timestep. A checklist encodes where the fingers that have recently been used are located based on prior predictions. These are both fed into an MLP which predicts the next finger.

a self-critical method for captioning that did not require this additional network. Paulus et al. (2017) applied a similar technique to abstractive news summarization. Reinforcement learning has also been applied in a musical setting (Jaques et al., 2016), including the REINFORCE algorithm specifically (Yu et al., 2017; Guimaraes et al., 2017), but most of these have focused on music generation rather than downstream prediction of performance-oriented attributes such as fingerings.

6.3 Model

We now describe the basic layout of the models we compare, and detail the specific variations between them. First, we will formalize the basic model contract and explain the basic architecture components that most of them share.

At a high level, each model takes in a MIDI-derived representation of either the left or right hand of a complete piano song, and outputs a predicted finger for each note. More formally, we are given an input sequence of notes $\mathbf{x} = \{x_1, ..., x_N\}$, where each $x_i = (p_i, t_i^-, t_i^+)$ is a tuple consisting of a pitch $p_i \in \{1, ..., 88\}$ as well as a corresponding onset time $t_i^- \in \mathbb{R}_{\geq 0}$ and offset time $t_i^+ > t_i^-$. From $\mathbf{x}$ we must predict a corresponding sequence $\mathbf{y} = \{y_1, ..., y_N\}$ where each $y_i \in \{1, 2, 3, 4, 5\}$ is an index representing the finger used to play x_i from thumb to pinky.

To this end, each predictive model defines a distribution $p(\mathbf{y}|\mathbf{x}; \theta)$, generally parameterized by an LSTM variant. Under our non-checklist simple Bi-LSTM baseline, the individual y_i are

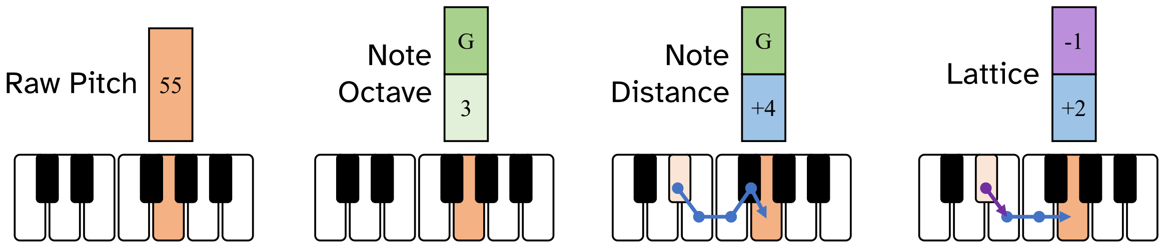


Figure 6.2: Examples of the input representations we compare. Note that the labels on the vectors are *indices* corresponding to sub-embeddings shared across notes.

conditionally independent given $\mathbf{x}$: $p(\mathbf{y}|\mathbf{x};\theta) = \prod_{i=1}^{N} p(y_i|\mathbf{x};\theta)$. However for our autoregressive models, the likelihood factorizes according to chain rule: $p(\mathbf{y}|\mathbf{x};\theta) = \prod_{i=1}^{N} p(y_i|\mathbf{y}_{<\mathbf{i}}, \mathbf{x};\theta)$

As shown in Figure 6.1, the notes are first embedded, and then passed to a Bi-LSTM which outputs a contextualized representation. This contextualized representation is optionally concatenated with a d dimensional vector representation of the checklist $\mathbf{c} \in \mathbb{R}^{N \times d}$ before being passed to an MLP which outputs a softmax distribution over finger indices. We can think of the checklist as a function of the past and current notes as well as the past finger predictions $c_i(\mathbf{x}_{\leq i}, \hat{\mathbf{y}}_{<i})$. The model is trained to optimize the cross entropy loss between this distribution and the true labels.

6.3.1 Input Representations

We now discuss four different strategies to encode the song into an input sequence of vector embeddings (illustrated in Figure 6.2). These each expose different information to the model which can significantly affect performance.

Raw Pitch: This approach simply represents each note by its pitch p_i, assigning a learnable embedding to each value. This was done by most previous neural implementations (Nakamura et al., 2020; Moryossef et al., 2019), and requires learning 88 vector embeddings.

Note $\oplus$ Octave: Given the fact that a piano keyboard's layout is identical across octaves, we might believe that more important than the exact key being played is the named note itself (e.g. knowing that the next note is a `G#` might be more useful in predicting which finger to use than knowing that it is the MIDI pitch 68). To this end we also evaluate the use of an input representation that consists of an embedding corresponding to the named note concatenated with an embedding corresponding to the octave.

Note $\oplus$ Relative Distance: If finger choice is primarily a function not of the note's pitch but of its location on the keyboard relative to the other notes in the song, then directly exposing that information saves the model from having to infer it. For this representation, we concatenate an embedding for the named note with one corresponding to the change in pitch $p_i - p_{i-1}$ (see Figure 6.2). Since large distances between notes require physically lifting the hand from the keys, an act which eliminates any sequential constraints, we cap step sizes at 15 semitones. Zhao et al. (2022) used a similar approach without the named note embedding.

Lattice: We finally evaluate a "lattice" representation based on the one introduced by Nakamura et al. (2020) (note that this was only used in that work for HMMs and not neural models). As shown in Figure 6.2, we can think of this as a two-dimensional encoding of relative position, where the first dimension corresponds to the number of white keys between notes (i.e. the *horizontal* distance), and the second dimension indicates if we have moved from a white key to a black key or vice versa (i.e. the *vertical* distance). These dimensions are each embedded with corresponding vector embeddings, with a horizontal cutoff of 9 steps. This is the default representation where not otherwise specified.

6.3.2 Checklist Formulation

To encourage the model to produce fluent predictions, rather than simply choosing the most likely finger at each timestep independently, we also consider a series of extensions that feed recent predictions into the MLP alongside the contextualized embeddings produced by the LSTM, in the form of a vector embedding $c_i(\mathbf{x}_{\leq i}, \hat{\mathbf{y}}_{<i}) \in \mathbb{R}^d$.

Autoregressive Tagger: We start with a baseline autoregressive variant of our simple Bi-LSTM model in which the forward half of the Bi-LSTM acts as an encoder over prior predictions as well as notes. In this setup, we do away with the Bi-LSTM over notes $\mathbf{x}$, and instead replace it with a forward LSTM $f(\mathbf{x}, \hat{\mathbf{y}})$, and a backward LSTM $b(\mathbf{x})$. At each timestep i, the MLP is fed the concatenated output of both LSTMs $f_i(\mathbf{x}_{<i}, \mathbf{y}_{<i}) \oplus b_i(\mathbf{x}_{\geq i})$.

Prev Finger Embedding: As a precursor to a full checklist, we also experiment with a neural variant of a Maximum Entropy Markov Model (MEMM) (McCallum et al., 2000). Under this setup, we map the predicted label of the previous timestep $\hat{y}_{i-1}$ to a corresponding vector embedding, and concatenate this with the contextualized embedding of x_i produced by the Bi-LSTM, before passing the result to the MLP. This approach only takes into account the most recently used finger and ignores timing, but also requires the fewest additional model parameters.

Binary Checklist: This checklist embedding is a 5 dimensional vector, where each dimension corresponds to a finger. The ith dimension is set to 1 if that finger has recently been predicted for a note of a higher pitch than the current, a -1 if it has recently been predicted for a lower note, or a 0 otherwise. We empirically find that "recent" is best defined as within 100 milliseconds.

Distance Checklist: This representation is a concatenation of 5 vector embeddings, one for each finger. These finger embeddings use the lattice input representation strategy, where the first half corresponds to the horizontal distance between the note that that finger was recently predicted for and the current one, and the second half corresponding to the vertical distance. If a finger has not recently been used, its spot in the checklist will be filled by a learnable "dummy" embedding.

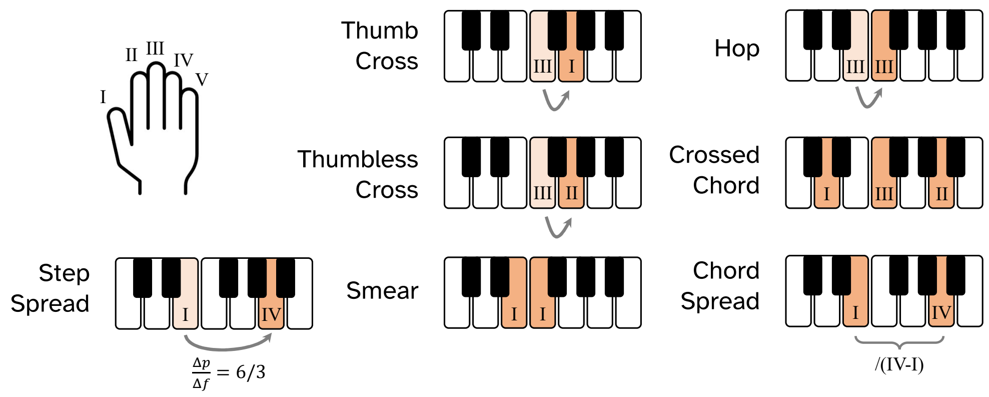


Figure 6.3: Fingering patterns tracked by our fluency metrics. Arrows and shading indicate sequential notes.

6.4 Learning and Evaluation

Next we will detail the additional metrics that we introduce, and show how by using reinforcement learning we can train our model to optimize some of them directly.

6.4.1 Fluency Metrics

We now describe the metrics we use to measure model performance. While research into pianists' fingering strategies shows that decisions are mostly motivated by minimizing ergonomically unfavorable patterns (Sloboda et al., 1998; Clarke et al., 1997), prior work has primarily evaluated on per-note labeling precision averaged over annotators, referred to as M_{gen} (albeit with some different strategies for handling disagreements between annotators (Nakamura et al., 2020)). However, by simply checking if each prediction agrees with any annotator's label for that note in isolation, we miss whether the model's predictions are coherent with one another. For instance, a model may output two adjacent fingers which each agree with an annotator's label, but not the same one, creating a pattern that is harder to play than if it had solely aligned with one of them, despite these having the same M_{gen} score. Also, a model may output a sequence that agrees with none of the annotators but is at least playable, and yet receives the same score as a physically impossible sequence. This is especially an issue for songs labeled by fewer annotators.

To address this, we expand the scope of our evaluation to include metrics that also measure how *fluent* model predictions are within themselves. To this end, we track statistics on the frequency of several types of output pattern that would increase the physical difficulty for a human performer, but may not be reflected in the raw labeling precision score. This allows us to more holistically compare models in ways that are both attentive to sustained agreement with annotators, and also expose the specific ways in which their predictions are more or less playable. Visual examples of these are provided in Figure 6.3 for clarity.

4-gram: Anchored 4-gram precision is the proportion of subsequences of four consecutive notes that were all predicted correctly with respect to a single annotator. This metric measures output coherence, but is still directly tied to the gold labels. Where M_{gen} scores plateau as they approach the inter-annotator agreement of 71.4, 4-gram precision has more headroom and better stratifies models.

Thumb/Thumbless Crosses and Crossed Chords: Thumb crosses are where a note lower in pitch than the previous is played with a "higher" finger or vice versa, one of those two fingers being a thumb. These shift the hand up or down the keyboard without fully lifting off of the keys. We also track crosses where neither finger is a thumb, which is much less common due to their difficulty. We refer to cases where two fingers are crossed and the notes overlap in time as crossed chords. These are only performed under very specific circumstances, as most chords are not physically playable with crossed fingers.

Hop: These are cases where one finger is used to play two different consecutive notes. While not intuitively difficult, these do not allow the player to rest their hand in a fixed spot, and make distances harder to accurately judge, limiting the maximum tempo of a piece, similar to how hunt-and-peck typing is generally slower.

Smear: In a smear, more than one note within a chord is played by a single finger. These are generally utilized when two adjacent keys are both in the chord, but because of the placement of the other notes it is hard to use two separate fingers. While these are extremely rare in our gold labels, naive baselines tend to produce them quite frequently.

Step/Chord Spread: Step spread tracks how far the player must stretch their fingers apart while playing. Specifically, it is the average number of semitones per finger separating any two adjacent but not overlapping notes. For example, if a song contains an `E` followed by the `B` a fifth above it, the step spread would be 7 for a model that predicts playing them with fingers 1 and 2 respectively, but only a 3.5 for a model that predicts 1 and 3. We measure similar cases where the notes do overlap in time as chord spread. A large average chord spread is more challenging since it requires stretching the hand over a larger distance.

6.4.2 Loss Functions

Having defined our model as above, we can train it on the cross entropy loss between the predicted distribution $p(\mathbf{y}|\mathbf{x};\theta)$ and the gold labels by simply backpropagating to obtain parameter gradients and taking gradient descent updates on our training set. Autoregressive models are teacher forced at train time (i.e. conditioned on gold labels from prior timesteps in lieu of the model's own previous predictions), and we decode with beam search at test time.

However, while training on cross entropy can yield strong performance on labeling precision, we find that this is not consistently correlated with other non-differentiable metrics which we may also wish to optimize with respect to. We therefore also investigate using a supplemental loss function based on the REINFORCE algorithm (Williams, 2004) which measures the frequency of undesirable fingering patterns in predicted outputs. While REINFORCE is traditionally associated with environment navigation, it has seen success in sequence generation tasks as well (Ranzato et al., 2016; Rennie et al., 2017; Paulus et al., 2017).

Under this framework, the loss function is formulated as an expected reward, which we approximate by sampling a sequence $\tilde{\mathbf{y}}$ from our output distribution (using ancestral sampling) and weighting its reward by its likelihood.

$$L = \frac{(\bar{r} - r(\mathbf{x}, \tilde{\mathbf{y}}))}{N} \sum_{i=1}^{N} \log p(\tilde{y}_i | \tilde{\mathbf{y}}_{<i}, \mathbf{x}; \theta)$$

Where $\bar{r}$ is the rolling average reward over the past 50 examples, serving as our reward baseline. In order to ensure stronger signal during training, we also calculate the REINFORCE loss over 10 note chunks at a time, rather than over an entire song as this would make the attribution of reward to specific decisions less direct. In our experiments, we set the reward to be a function of the number of hops and smears (see Section 6.4.1):

$$r(\mathbf{x}, \mathbf{y}) = \exp\big(-\#\{i : x_i \neq x_{i+1}, y_i = y_{i+1}\} - \#\{i : y_i = y_{i+1}, t_i^+ > t_{i+1}^-\}\big)$$

Because training solely on the REINFORCE objective can lead the model towards degenerate solutions that trivially minimize undesirable patterns while compromising predictive accuracy, we take an approach similar to Paulus et al. (2017) and Wu et al. (2016) where both loss functions are summed into a mixed training objective. This provides a reasonable tradeoff between the predictive signal of cross entropy, and the pressure towards fluent output given by REINFORCE. Mixed objective runs are warm-started on just the cross entropy loss to avoid degenerate solutions.

6.5 Experimental Setup

6.5.1 Implementation Details

We use 2 layer LSTMs and 2 layer MLPs with a hidden size of 1024, and a dropout of 0.2 in both of these networks. We use d=256 for our input embedding size. Beam search is performed to decode autoregressive models at test time using a beam size of 10. Models are trained using the Adam optimizer (Kingma and Ba, 2015) with a learning rate of $1\text{e}{-4}$. Our code is implemented in PyTorch `1.8.1` (Paszke et al., 2017) and trains on an NVIDIA 2080ti GPU in roughly 12 hours.

6.5.2 Data

We train and evaluate on the PIano fingerinG dataset (PIG) (Nakamura et al., 2020) which contains 150 piano songs written by 24 composers. Each song has up to 6 fingerings produced by human pianists, yielding 309 annotated songs in total. Because of the imbalance in the dataset's original splits, we use alternate splits created by Moryossef et al. (2019) which increase the relative size of the train and validation sets. As a baseline we compare to a reimplementation of the LSTM model of Nakamura et al. (2020) (which was also reimplemented by Moryossef et al. (2019) in their experiments), albeit with the same architecture of our other systems. We also show results from the third order HMM implementation of Nakamura et al. (2020), although the more similar neural models are our primary point of direct comparison.

We employ similar preprocessing as prior work. Rather than modeling left and right hands separately, we reflect the pitches of all left hand parts and reverse the corresponding finger labels, thereby constructing a second "right hand part" which we treat as an independent song. This prevents overfitting by halving the label space. To handle chords, i.e. multiple notes with identical onsets, we simply arpeggiate them from lowest to highest pitch. Otherwise, notes are ordered by onset. We do however retain the original timing information for constructing accurate checklists.

6.6 Results

6.6.1 Input Representation Ablation

We start by investigating the effects of input representations on a simple Bi-LSTM baseline model that outputs independent prediction probabilities at each timestep. Table 6.1 shows our results across all metrics. Overall the lattice representation does best, followed by note $\oplus$ octave. This matches our hypothesis that learning a separate embedding for each note on the keyboard leads to overparameterization, and that modular representations are more effective. We also see that relative distance based embeddings lead to fewer hops and smears specifically, which makes sense given that it allows the model to more easily observe when notes are or are not repeated, and therefore whether repeating the same finger would make sense.

6.6.2 Checklist Models

We see in Table 6.2 that the checklist models tend to do best overall, with Binary Checklist using REINFORCE getting the highest 4-gram score among LSTMs. All the autoregressive systems significantly outperform the Bi-LSTM baselines, but with different tradeoffs in terms of the output patterns we measure. REINFORCE minimizes the number of hops and smears compared to just cross entropy, and in doing so can sometimes boost other metrics as well.

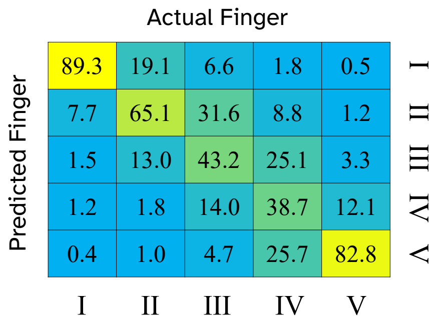
Model	Thumb Cross (#)	Thumbless Cross (#)	Crossed Chord (#)	Hop (#)	Smear (#)	Step Spread (s/f)	Chord Spread (s/f)	4-gram (Acc)	M_{gen} (Acc)
Gold	331	58	28	155	7	2.78	2.64	100	100
Bi-LSTM (MIDI) (Nakamura et al., 2020)†	228	84	47	485	148	2.63	2.56	66.7	27.8
Bi-LSTM (Note ⊕ Octave)	**257**	90	**36**	394	135	**2.69**	2.61	67.0	28.3
Bi-LSTM (Note ⊕ Rel Dist)	218	87	52	390	101	2.65	2.59	69.1	32.2
Bi-LSTM (Lattice)	217	**43**	40	**334**	**74**	2.66	**2.62**	**69.6**	**33.1**

Table 6.1: Results on PIG test set for Bi-LSTM model with different input representation schemes. The first row represents the value of each metric under the gold fingerings produced by human annotators. Values in bold are the closest to the gold labels' score for that metric. †This model is based specifically on the neural approach used in Nakamura et al. (2020), but is our own implementation of it. It was similarly reimplemented by Moryossef et al. (2019) in their work. Units for most metrics are simple frequency counts, except for spreads which are in semitones per finger, and 4-gram and M_{gen} which are accuracies.

Note that M_{gen} fails to fully reflect what are at times substantial differences in model output made apparent from the other metrics. For instance, the Prev Finger Embedding produces fewer hops, smears, and crossed chords than the Autoregressive tagger, likely indicating that it would be far easier for a human to perform despite having worse M_{gen}. This is especially apparent when comparing the checklist models to the Bi-LSTM baselines. Also while the HMM does strongly on M_{gen} and 4-gram, we only see its deficits when we look at other metrics which reveal subtle issues such as an especially high finger spread and number of smears, and low utilization of thumb crosses.

6.6.3 Label Confusion

This figure shows the confusion matrix for predictions by the Binary Checklist model with REINFORCE. The distribution of misclassifications is fairly different for each finger; adjacent fingers are more likely to be confused than ones that are further apart, reflecting a degree of interchangeability in how pianists use them. We also see that thumb and pinky predictions are most accurate, perhaps because being at the ends of the hand makes them less versatile.



6.6.4 Human Evaluation and Conclusion

Since the ultimate success criteria of our task is to produce fingerings that are well-suited to human performance, we also present results from a small scale round of human evaluation. Specifically, we recruited a college professor of piano with a doctorate in piano performance (who was not

Model	Thumb Cross (#)	Thumbless Cross (#)	Crossed Chord (#)	Hop (#)	Smear (#)	Step Spread (s/f)	Chord Spread (s/f)	M_{gen} (Acc)	4-gram (Acc)
Gold	331	58	28	155	7	2.78	2.64	100	100
Bi-LSTM (MIDI) (Nakamura et al., 2020)†	228	84	47	485	148	2.63	2.56	66.7	27.8
HMM-3 (Nakamura et al., 2020)‡	220	32	31	196	84	2.84	2.72	**70.2**	**39.5**
Autoregressive Tagger	278	88	49	**168**	58	2.74	2.67	68.3	36.7
(+REINFORCE)	274	66	49	59	18	2.70	2.63	68.7	36.5
Prev Finger Embedding	271	**56**	40	127	30	**2.78**	2.67	68.1	35.5
(+REINFORCE)	283	53	54	53	**12**	2.81	2.76	68.4	36.5
Binary Checklist	227	72	**26**	332	74	2.67	2.58	69.3	35.8
(+REINFORCE)	261	49	**26**	217	42	2.69	**2.64**	69.5	37.1
Distance Checklist	274	72	37	195	37	2.80	2.69	68.8	36.6
(+REINFORCE)	**284**	78	41	134	32	2.75	2.68	69.0	36.1

Table 6.2: Results on PIG test set for various autoregressive setups, shown with and without using REINFORCE to minimize hops and smears (see Section 6.4.2). ‡ A third order HMM implementation by Nakamura et al. (2020) is also included – while it ranks similarly as in prior work against LSTMs by M_{gen}, it falls behind on other metrics.

involved with the creation of the original dataset) to assess the outputs of two models—the Bi-LSTM (MIDI) (Nakamura et al., 2020) and Binary Checklist with REINFORCE—according to three main criteria: physical comfort, mechanical efficiency, and ease of learnability. The pianist was asked to score the models' outputs with respect to each of these categories on a scale of 1-3 for all 10 songs in the validation set (composed by Brahms, Mendelssohn, and Rachmaninoff).

We show the models' average ratings per category in Table 6.3. The strong contrast suggests that the checklist model is not just predicting gold label fingers more frequently (shown by M_{gen}) but also that it is producing more locally coherent outputs in a way that substantially improves performability. Furthermore, the correlation of human judgements with our proposed metrics indicates that they are meaningful measurements of output quality.

The pianist also provided qualitative observations which we summarize. He noted that both systems performed best on simpler, repetitive passages such as those found in Mendelssohn. Our model also reportedly produced fewer difficult stretches, which we expect as it is directly aware of interval distances in its input representation. Another observed issue was that the Bi-LSTM's fingerings did not consider tempo, often containing sections that would be impossible to play fast enough, especially for Rachmaninoff pieces. We suspect our model's advantage is from the checklist's dependence on absolute timing, not just the ordering of notes. The pianist also reported that our model more frequently used the same fingering for repeated notes and chords, whereas the Bi-LSTM would often change fingerings without reason, which can also likely be explained by our conditioning on recent predictions instead of decoding them independently. Quantitatively, we found that REINFORCE was able to reduce the number of hops and smears, which was confirmed by the pianist to have a significant impact on playability, emphasizing the importance of integrating these into the training loop instead of purely relying on a traditional

Model	Physical Comfort	Mechanical Efficiency	Ease of Learning
Bi-LSTM (MIDI) (Nakamura et al., 2020)	1.6	1.4	1.6
Binary Checklist (+REINFORCE)	**2.8**	**2.5**	**2.9**

Table 6.3: Human evaluation ratings across various criteria.

likelihood loss.

Finally, he noted that the models particularly struggled on successive chords (as opposed to stepwise passages). Chord transitions require fingerings that are not just comfortable in isolation but also connect together. This requires the model to reason about *voices* within the same hand, something which may require moving beyond a framework that treats songs as linear sequences of individual notes, and is more sensitive to polyphony.

Part III

Contextually Relevant Text Descriptions of other Modalities

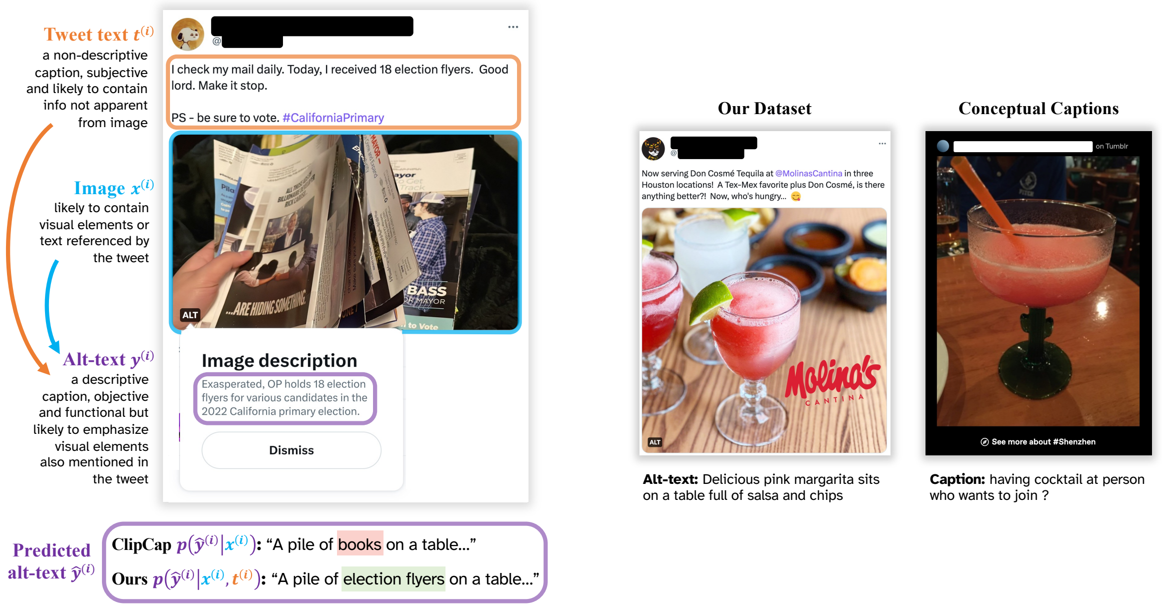
Chapter 7

Alt-Text with Context: Improving Accessibility for Images on Twitter

7.1 Introduction

Having discussed cases of learning contextualized representations of text and music, we will shift now to captioning tasks. In this setting, we are not directly leveraging connections between datapoints, but rather zooming in to the instance level to examine additional metadata that prior work has not considered utilizing. Specifically, this means that we will have to form multimodal representations of tweets that can then be passed to a unimodal language model decoder.

An increasingly important aspect of the social media user experience centers around the sharing and discussion of visual content. However, for users who are blind or have low vision (BLV), this type of content is often inaccessible. One solution to this problem is alt-text, an HTML attribute associated with digital images, which is intended to provide a literal natural language description of an image's content, and can be rendered by a screen reader or braille display. Despite being infrequently included, it is incredibly important from an accessibility standpoint, as it allows users to perceive the content of the image and thus have a better online experience. Some social media websites, notably Twitter, have recently given users the option to write their own alt-text for images that they upload (an example is shown in Figure 7.1). This approach however offloads the responsibility of making the site accessible onto the users, who frequently either for convenience or lack of awareness simply do not choose to do so (Gleason et al., 2019). We find that as many as 98% of images uploaded to Twitter even after the widespread feature rollout do not have alt-text, to say nothing of the quality of those that do. When a screen reader encounters such an image, it will simply say "Image", leaving the user with no meaningful information about what the image is actually of. Even when images on Twitter do have accompanying user-written alt-text, the quality is inconsistent as not all users are well informed regarding best practices.


Tweet text $t^{(i)}$
a non-descriptive caption, subjective and likely to contain info not apparent from image
Image $x^{(i)}$
likely to contain visual elements or text referenced by the tweet
Alt-text $y^{(i)}$
a descriptive caption, objective and functional but likely to emphasize visual elements also mentioned in the tweet
Our Dataset
Conceptual Captions
Predicted alt-text $\hat{y}^{(i)}$
ALT
Dismiss
See more about #Shenzhen

[redacted] @[redacted]

I check my mail daily. Today, I received 18 election flyers. Good lord. Make it stop.

PS - be sure to vote. #CaliforniaPrimary

Image description

Exasperated, OP holds 18 election flyers for various candidates in the 2022 California primary election.

ClipCap $p(\hat{y}^{(i)}|x^{(i)})$: "A pile of books on a table..."

Ours $p(\hat{y}^{(i)}|x^{(i)}, t^{(i)})$: "A pile of election flyers on a table..."

[redacted] @

Now serving Don Cosmé Tequila at @MolinasCantina in three Houston locations! A Tex-Mex favorite plus Don Cosmé, is there anything better?! Now, who's hungry... 😋

[redacted] on Tumblr

Alt-text: Delicious pink margarita sits on a table full of salsa and chips

Caption: having cocktail at person who wants to join ?

Figure 7.1: **Left:** An image that requires textual context to write accurate alt-text for. Without conditioning on the tweet text, the election flyers are indistinguishable from books to a traditional captioning system. **Right:** Two similar images from our dataset and Conceptual Captions with their gold labels. The alt-text for the first image is literally descriptive while the second is more colloquial.

Note that while there is a long line of existing research on the broader task of captioning images more generally, alt-text generation for social media is a special case of this task, which in turn comes with its own challenges. Well written alt-text is generally more explicitly descriptive than a high level caption and may emphasize specific details in the image based on context (Kreiss et al., 2022a). See Figure 7.1 for an example. Furthermore, the distribution of image types on Twitter differs substantially from those found in traditional captioning datasets, and may contain digital artwork, promotional graphics, or screenshots containing text. An additional challenge is that Twitter users are not well trained in the practice of writing alt-text, and therefore native "gold" examples can vary substantially in their level of quality. To support continued research in this area, we collect and release a first-of-its-kind dataset of image and alt-text pairs scraped from Twitter.

Our central motivating idea is that despite the challenges associated with this domain, social media images come with additional contextual information that if properly leveraged can significantly improve the quality of automatically generated alt-text. Images posted to social media often require some amount of background knowledge to adequately caption. For example the left image in Figure 7.1 may easily be mistaken for a stack of books or magazines. However, we do in fact have a source of text that might provide context not present in the visual information contained in the image itself, namely the text associated with the actual post which more specifically identifies the pamphlets as election flyers. We therefore see an opportunity to augment existing captioning systems with textual conditioning in order to allow them to achieve better performance on this challenging task.

One model that is easily modifiable to accept multimodal inputs is ClipCap (Mokady et al., 2021). ClipCap operates using a prefix method, where an image encoder — in this case CLIP (Radford et al., 2021) — produces a vector embedding of the image which is then projected to a sequence of embeddings that occupy the same dimensionality as word embeddings. These (despite not actually being the embedding of any specific language sequence) can then be taken as inputs to a language model decoder — in this case GPT-2 (Radford et al., 2019) — which then autoregressively completes the sequence to produce the caption.

Our approach (shown in Figure 7.2) supplements the text-like image embedding sequence with an embedding of the tweet text, in the hopes that these two information sources will provide meaningfully non-overlapping signal to the decoder, in order to produce more accurate captions. This reduces the burden on the image encoder to be able to distinguish very similar object types, and allows the decoder access to details that may be very difficult to glean from the visual features.

In summary our contributions are as follows: (1) We collect a novel benchmark dataset for the task of alt-text prediction on social media including both images and surrounding textual context (2) We introduce a novel alt-text generation method that allows for multimodal conditioning with a unimodal language model (3) We perform a quantitative analysis of this approach on our collected dataset, outperforming prior work on captioning (ClipCap, Mokady et al. (2021)) and vision-and-language pretraining (BLIP-2, Li et al. (2023)) by more than 2x on BLEU@4. We also conduct human evaluation which substantiates these findings.

7.2 Related Work

Image captioning in a generic sense is a relatively well studied task. Since the introduction of the COCO benchmark (Lin et al., 2014) a variety of systems have been released, and state-of-the-art quantitative performance has risen across several metrics (Vinyals et al., 2015; Fang et al., 2015; Bernardi et al., 2016). Alt-text generation is relatively understudied by comparison, and there are few established datasets for it.

Kreiss et al. (2022b) collected one such dataset of images with descriptive alt-text and contextual captions from Wikipedia. This corpus was smaller than the one we put forward here at 97k images, and also represents a fundamentally different domain as Wikipedia contains different image types than social media, more frequently has gold alt-text, and has much more carefully curated text captions. They do however demonstrate that conditioning on textual context can improve the quality of generated alt-text, suggesting the value of a similar approach for our setting.

Examining alt-text in the context of social media posts has been studied as well, albeit to an even more limited extent. Twitter A11y (Gleason et al., 2020) is a browser extension based on a pipeline system that primarily works through scraping matching alt-text from alternate sources, only generating it abstractively through a separate caption generator if other methods fail.

Facebook developed and published a system they refer to as Automatic Alt-Text (AAT) (Wu et al., 2017). The approach described does not generate text abstractively, and instead essentially outputs a list of tags indicating various objects detected within the image. While this is arguably more informative than nothing, it's worth emphasizing that this format does not strictly speaking adhere to alt-text best practices. The system has since been refined to include additional details regarding position, count, and size (Meta, 2021). Facebook's AAT has been critically examined, notably by Hanley et al. (2021) who focused particularly on how it describes personal identity categories. They summarize a variety of limitations BLV users experience with it including a lack of detail, a reliance on tags as opposed to fluid natural language captions, and an inability to reason about the contextual salience of various parts of the image.

Chiarella et al. (2020) outlined steps to increase the accessibility of scientific images posted to Twitter. Alt-Textify (Belle. et al., 2022) proposed a method specifically designed for generating alt-text on SVG-based data visualizations and charts, and Chintalapati et al. (2022) put forward a dataset and analysis of visualizations within HCI publications. Li et al. (2020) introduced a method for generating alt-text specifically for mobile UI elements.

One prominent recent example of a more general-purpose system for aligning text with images is CLIP (Radford et al., 2021). While CLIP is not exactly a captioning system, it can be used to score text/image pairs as a way of reranking or selecting captions from a candidate list. Elements of CLIP's architecture have however found success within the pipelines of broader captioning models. One example is ClipCap (Mokady et al., 2021), which uses a mapping network to project a CLIP image embedding into word embedding space and then feeds the result as a prefix to a language model which outputs the caption. We inherit this basic architectural framework, but extend it to multimodal prefixes which also contain textual context.

While there are other existing datasets for image captioning, they do differ from the one put forward in this chapter in a few key ways. One of the most similar examples is Conceptual Captions (Sharma et al., 2018), a dataset of images with short and descriptive text captions. These captions are produced by processing the alt-text descriptions found within HTML to simplify and improve fluency. However, the distribution of images is fundamentally different from that found on social media, and the captions are not generally descriptive enough for accessibility purposes (as a result their captions average only 10 tokens, whereas ours average 40). An example showing this contrast is provided in Figure 7.1. Furthermore this dataset does not include

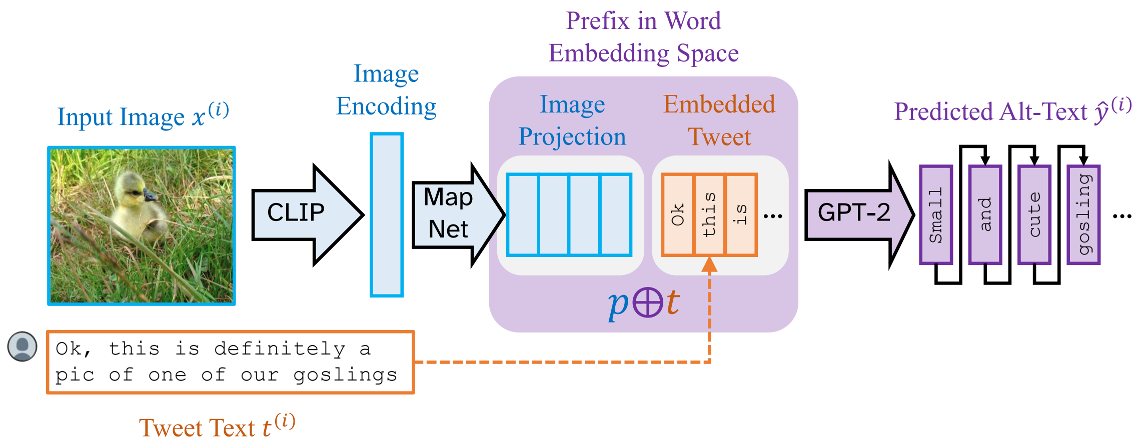


Figure 7.2: Overview of the alt-text model. An image is encoded via CLIP to obtain an embedding of visual features. This gets projected via a mapping network into word embedding space, where it is then concatenated with an embedded representation of the text from the corresponding tweet. This prefix is passed to a finetuned GPT-2 which autoregressively generates the alt-text caption.

surrounding textual context, which we demonstrate to be extremely informative for improved alt-text generation.

More recently the emerging subfield of large language models (LLMs) has expanded into exploring multimodal systems capable of conditioning on images interleaved with text. Systems including Flamingo (Alayrac et al., 2022) and GPT-4 (OpenAI, 2023) have gained popularity, although their closed nature makes them difficult to study. Further, there is strong reason to prefer federated systems that are small and controllable by end users rather than through paid opaque APIs that are subject to change without notice and make scientific reproducibility impossible. Open source alternatives such as OpenFlamingo (Awadalla et al., 2023) and BLIP (Li et al., 2022, 2023) have also emerged. These models are capable of captioning images based on few if any in-context examples, despite not being specifically trained on this task. It should be emphasized that these models are however not only incomparable due to their massive and unequal training sets, but are arguably inefficiently overparameterized in their capacity to model a wide variety of tasks beyond captioning. The system we propose instead is trainable on reasonable compute hardware, does not require esoteric prompting strategies, and as we will demonstrate still outperforms BLIP-2 on alt-text generation. Certain large pretrained language models have also been shown to exhibit disability bias (Herold et al., 2022) which is undesirable in general and especially in this context.

7.3 Model

At a high level, our model's primary innovation is being able to condition on not just visual information from the image, but also on textual content from the tweet itself. This is in contrast to prior work which captions images in isolation from the text alongside which they appear. We will now describe the multimodal conditioning setup more formally, and over the rest of this section describe our model's architecture for parameterizing the relevant distributions. Suppose that we are given a dataset $X = \{x^{(1)}, ..., x^{(N)}\}$ of images and corresponding tweet text $T = \{t^{(1)}, ..., t^{(N)}\}$[1] as well as gold alt-text descriptions $Y = \{y^{(1)}, ..., y^{(N)}\}$. Our model defines a conditional distribution $p(y|x, t; \theta)$ over the captions given the tweet and image. Having optimized this distribution with respect to θ over our dataset at train time, at test time our task is to predict $\hat{y} = \arg\max_y p(y|x, t)$. Note that prior work can be thought of as a special case where the tweet t is not considered.

7.3.1 Multimodal Conditioning

Our model's decoder must output text in order to generate the alt-text caption. A natural choice for this is an autoregressive transformer-based language model – we specifically opt for GPT-2 (Radford et al., 2019). Conveniently, such architectures' masked self-attention mechanisms make them inherently well suited to conditioning on text. In order to generate alt-text conditioned on the tweet, we can simply feed the tweet to the language model as a prompt and train the model to autoregressively complete it with the corresponding alt-text. The more challenging step is allowing the language model to condition on visual information from the image as well. However, we would also like to avoid fundamentally modifying the decoder's architecture and still be able to leverage unimodal pretraining. In order to solve this, what we require is a way to project image features into text space (or more specifically word embedding space) so that we can directly incorporate them into the tweet prefix. Fortunately, this is something that prior work has addressed. ClipCap (Mokady et al., 2021) learns a mapping network on top of a pretrained CLIP image encoder that projects the visual vector encoding into a sequence of embeddings that while not directly corresponding to specific words, are still in word embedding space and therefore acceptable by a pretrained language model.

It's worth emphasizing that the crux of this method relies on "translating" image features into a format which the language model can receive as input as if it were a text prefix. An observable artifact of this is that the mapping network often places its outputs near embeddings corresponding to words that are relevant to the contents of the image (Mokady et al., 2021). The prefix can thus be "de-embedded" back into text as a rough means of inspecting what information about the image it encodes. Crucially, this also suggests that further context not present in the image itself could be provided to the decoder in the form of additional text added to the prefix. In this way, we can effectively create a prefix reflecting multimodal information that nonetheless exists in a

[1]Note $t^{(i)} = t^{(j)}$ if images i, j are from the same tweet.

unimodal space, and can be input to a pretrained unimodal language model.

See Figure 7.2 for an overview of our model's pipeline. Drawing inspiration from ClipCap's architecture (see Mokady et al. (2021) for details), we pass an image x through a pretrained CLIP image encoder, and project the output to word embedding space using a mapping network. This is parameterized by a multi-layer perceptron that takes in a 512 dimensional vector CLIP embedding and outputs a k long sequence of 768 dimensional vectors (the same size GPT-2 expects for each token). These embeddings constitute a prefix p which is the correct shape to be passed to GPT-2.

Having obtained p, a sequence of token embedding-esque vectors, the procedure for combining it with the tweet text to produce a prefix containing both visual and textual information is fairly straightforward. We simply concatenate that projection with the embedded tweet text t (note that they are both sequences in word embedding space) to obtain the complete prefix $p \oplus t$. We can condition on this, treating it as something analogous to a multimodal prompt to our language model, which then autoregressively outputs the alt-text caption $\hat{y}$. Our decoder can therefore condition both on the tweet text, which is the same modality as its output, and the image itself which is not.

To train the model, we simply teacher force and optimize the cross entropy loss between the predicted logits and the gold reference. We backpropagate into the weights of both the mapping network and GPT-2, but leave CLIP frozen both for speed and performance reasons, following Mokady et al. (2021). Further implementation details are provided in Section 7.5.2.

7.3.2 Decoding

After training, there are important choice points for how we decode captions at test time that can significantly impact performance. While ClipCap originally used greedy decoding, we instead opt for beam search with a beam size of 5. Furthermore, as with many language models we noticed that ours occasionally repeated words or short phrases towards the end of captions. We solve this with the trigram blocking strategy described by Paulus et al. (2017). For fair comparison we use this same decoding method for ClipCap in our experiments.

7.3.3 Reranking

We also experiment with reranking the candidates from beam search by their similarity to the tweet text, as the highest likelihood caption might not necessarily score best under our nondifferentiable evaluation metrics. While the tweet text is not itself a descriptive caption, it may reference specific things in the image that are especially relevant, and therefore high quality alt-text will likely have a significant degree of ngram overlap with it. While the model does get to condition on the tweet text through the prefix, we may wish to ensure that that information does ultimately appear in the final output. In order to steer our generations in this direction, we choose a new best caption among the top 5 candidates returned by beam search based on their ROUGE-L similarity to the tweet text t. We also experimented with reranking based on BLEU and CLIP text embedding similarity but found that neither of these performed as well as ROUGE-L.

7.4 Dataset Collection

One contribution of this chapter is the collection and release of a large-scale benchmark dataset for training and evaluating alt-text generation systems. We now describe how we collect a publicly available dataset of tweets containing images and corresponding user-written alt-text descriptions.

These tweets were scraped via the public Twitter APIv2 from a 21 day period in the summer of 2022. We focused on English tweets containing still images with user-written alt-text. One important consideration was the removal of duplicates. We noticed that many accounts — especially bots and promotional accounts — would post the same image multiple times, and others (e.g. those tracking extreme weather events or stock prices) often post extremely similar images with only a few words differing between their corresponding alt-text captions. We addressed this by deduplicating based on textual and visual matches.

7.4.1 Filtering

Reply tweets were included, but retweets were not. We filtered based on tweets containing still images, as alt-text for GIFs tends to be of very low quality and requires different modeling approaches. We also restricted our search to only include tweets which Twitter's language identification model categorized as English, thereby also ensuring that they all contained at least some amount of tweet text. Additional preprocessing included stripping all leading phrases such as "Image of" or "Photo of" (as these are generally considered redundant for alt-text purposes), removing any examples with alt-text that were identical to the text of the tweet itself or simply contained placeholder text such as "Image", and removing any examples with alt-text containing URLs, user handles, and hashtags. We also only included alt-text that was at least four space-separated tokens long, as text shorter than this is rarely sufficiently descriptive.

7.4.2 Deduplication

We only retained the oldest of any images that contained identical alt-text. This largely eliminated any exact duplicates posted multiple times by spammy users. Next we identified clusters of visual matches based on their downscaled pixel overlap. To do this, we resized images and center cropped to a max size of 32×32 pixels, computed all pairwise pixel diffs within our data, and agglomeratively grouped them into clusters using a < 100 differing pixels threshold to the nearest match as a cutoff for cluster membership. Within each cluster, we only retained the oldest tweet. It's also worth mentioning that while we did consider further deduplication based on SSIM (Snell et al., 2017) similarity and ngram similarity of the alt-text itself, these approaches proved computationally infeasible.

7.4.3 Named Entities

We also took steps to reduce the incidence of names in our data. While many users caption images using the real names of the people in them, predicting a person's name from an image would implicitly require learning to perform facial recognition, something well beyond the scope of this work. Furthermore, from a user privacy standpoint, we would not want our model to be able to output the names of real people, regardless of whether they actually appear in that image. We used the existing NER system of Mishra et al. (2020); Eskander et al. (2020) to identify named entities in our alt-text, and replaced all instances of the `Person` type with the string "person". While this substitutes no additional visual description of the person, and can lead to grammatically cumbersome transformations, we leave further improvements to future work.

7.4.4 Statistics and Quality

This yielded a dataset of 371,270 images. We split into train (330,449), val (20,561), and test (20,260) sets based on tweet ID (i.e. images from the same tweet are assigned to the same split) to prevent leakage of tweet text. The corresponding alt-text captions were on average 144 characters long, which translates to 40 tokens. The tweet text was similarly 152 characters long on average, or 46 tokens. We also hard crop both the alt-text and tweet text at 150 tokens max length. While the raw data cannot be distributed directly in order to protect users' right to be forgotten, we release the dehydrated tweet IDs and media URLs [2].

In order to quantitatively assess the quality of the user-written alt-text descriptions, we randomly sampled 200 tweets from the validation set, and manually inspected them to determine if they were fluent, and descriptive of the image. We found that of the 68.0% of images that still loaded and passed our preprocessing filters (described above), 94.1% were fluently written, and 86.0% contained some form of literal description of the visual contents. Of those that didn't, the majority simply did not provide enough detail or treated the field as a second caption that provided context for what was being depicted, but not a visual description.

[2] `https://github.com/NikitaSrivatsan/AltTextPublicICLR`

7.5 Experiments

Having described our method, and our dataset collection procedure, we now conduct experiments to assess how well existing models perform on this new task, and by how much our multimodal system improves over those approaches. This section provides an overview of our experimental setup, including details of our baselines and ablations, and the metrics we evaluate on.

7.5.1 Baselines

We examine a few key baselines and ablations in our experiments in addition to our full model. The first of these is a **Nearest Neighbor** system, which returns the verbatim alt-text caption associated with the most visually similar image from the training set. We determine visual similarity based on the dot product of the pretrained CLIP features of the images (the same visual encoding used by our full models). The second baseline always returns the corresponding tweet text as the predicted alt-text, which we refer to as **Copy Tweet Text**. This tells us to what extent the tweet text is already a reasonable alt-text caption without transformation. Following our motivation for conditioning on the tweet text, we expect that this system will occasionally return something informative albeit likely not well formed, and certainly redundant from the perspective of a BLV user. This can also be thought of as a simpler variant of our text only system, described below.

Our first neural baseline is **ClipCap**, described previously, using its publicly available implementation. ClipCap is already pretrained on a large captioning dataset, so we evaluate it both out of the box, and finetuned on our own data. For fair comparison, we also use the same updated decoding strategy for our ClipCap experiments. We also compare to pretrained **BLIP-2** (Li et al., 2023) using their OPT (Zhang et al., 2022) 2.7B checkpoint.

Viewing ClipCap as a variant of our model that only includes image features in the prefix begs the question of whether we could similarly include the tweet text, but not the projected image prefix. We therefore also consider a baseline that includes the tweet text t in the prefix, but not the projected image prefix p (referred to as **Ours (Text Only)**). This is effectively a translation model between the tweet text and alt-text; it can transform the style and filter out irrelevant details, but cannot inject new information, and thus serves as an indicator of the performance achievable based only on surrounding context without observing the image itself. We also consider an ablation of our system that substitutes the retrieved nearest neighbor for a randomly chosen tweet, which we call **Ours (Rand Text)**. This lets us measure how robust our system is to errors in retrieval and how well it can generalize to cases where a similar neighbor does not exist in train.

7.5.2 Implementation Details

We train our models with a batch size of 100 and an initial learning rate of $1e-4$ using the Adam optimization algorithm (Kingma and Ba, 2015). Our prefixes are of size $k = 10$. This allows training to fit on a single A6000 GPU. Our implementation is written in PyTorch (Paszke et al., 2017) and inherits some code from the ClipCap (Mokady et al., 2021) repository. Experiments ran in roughly 6-18 hours depending on the model, and when early stopping happens (we perform early stopping based on model likelihood on the held-out validation set).

7.5.3 Metrics

Most image captioning work has evaluated using quantitative metrics that roughly measure the string similarity between a model's generated caption and a reference gold caption. Since we have user-written alt-text for each image in our dataset, we simply treat these as our gold references to compare to. Specifically, we measure performance on BLEU@4 (Papineni et al., 2002), METEOR (Denkowski and Lavie, 2014), ROUGE-L (Lin and Och, 2004), and CIDEr (Vedantam et al., 2015). It's worth noting however that because the "gold" alt-text captions on Twitter are written by untrained users, they can be noisy, inconsistent in form and specificity, and occasionally do not even describe the image contents (Gleason et al., 2019). As a result, the absolute scores may seem relatively low compared to numbers typically reported on other datasets, although based on qualitative inspection and human assessments they do nonetheless meaningfully correlate with output quality.

7.5.4 Human Evaluation Setup

Due to this variability, as well as the broader problems with contextless evaluation for this task (Kreiss et al., 2022a), we also conduct human evaluation using Amazon Mechanical Turk. For this experiment, we subsampled 954 examples from our test set (originally 1000 but some tweets had been deleted over time), and asked a group of sighted human annotators to compare the outputs of our multimodal system with two variants of ClipCap for those images (all systems using beam search with no repeats).

For each example, turkers were shown the image, the original tweet the image was from (including its text and any other images it contained), and the two candidate alt-text captions. They were never shown the original user-written alt-text. After providing them with a detailed explanation of alt-text and the list of desirable criteria according to Twitter's accessibility guide, we asked two questions: first which candidate alt-text was more fluent, and second which had a better literal description of the image. Models were anonymized and their order was shuffled.

Model	Decoding	BLEU@4	METEOR	ROUGE-L	CIDEr
Naive Baselines					
Nearest Neighbor	-	0.563	1.827	5.074	1.671
Copy Tweet Text	-	0.230	2.066	4.801	0.453
Neural Baselines					
BLIP-2 (Frozen)	BS	0.111	1.449	6.744	1.381
ClipCap (Frozen)	BS (NR)	0.372	1.400	6.690	0.830
ClipCap (Finetuned)	BS (NR)	0.681	2.685	8.620	1.557
ClipCap (From Scratch)	BS (NR)	0.696	2.643	8.483	1.588
Tweet Text Conditioning					
Ours (Rand Text)	BS (NR)	0.546	2.547	7.906	1.171
Ours (Text Only)	BS (NR)	0.693	2.459	7.498	1.738
Ours (Text + Image)	BS (NR)	**1.826**	**3.067**	**8.652**	**7.661**
Decoding Ablation					
Ours (Text + Image)	Greedy	0.587	2.247	7.166	1.450
Ours (Text + Image)	Greedy (NR)	0.469	2.168	7.774	1.520
Ours (Text + Image)	BS	0.400	2.311	6.275	1.159
Ours (Text + Image)	BS (NR)	1.826	3.067	8.652	**7.661**
ClipCap (From Scratch)	BS (NR) + RR	0.691	2.899	8.760	1.379
Ours (Text + Image)	BS (NR) + RR	**1.897**	**3.257**	**8.818**	6.791

Table 7.1: Quantitative test results across various metrics (all numbers x100). Our method and ClipCap are decoded using beam search with no repeats. **BS** indicates Beam Search, **NR** indicates no repeats, and **RR** indicates ROUGE-L reranking.

In order to ensure high quality labels, we performed a qualification round on our reviewers. Considering only turkers who had completed over $10,000$ HITs, we showed them three example images where the two candidate alt-text captions were the gold user-written reference and a manually written, intentionally clearly lower quality caption. Only reviewers who successfully picked the real user-written caption for both fluency and descriptiveness for all three examples were allowed to participate in the final larger scale comparisons between our model and frozen ClipCap, and our model and ClipCap from scratch.

7.6 Results

We now present results on our dataset, both using automatic metrics comparing model output to user-written reference alt-text, and using human evaluation. We also perform qualitative evaluation to analyze our output more subjectively.

7.6.1 Automatic Metrics

In Table 7.1 we show reconstruction performance on our test set across a variety of metrics. Copying tweet text achieves the lowest performance on all metrics except for METEOR, which

Model	Fluency	Descriptiveness
ClipCap (Frozen)	29.8	26.2
Ours (Text + Image)	**61.4**	**66.9**
Equal Quality	8.8	6.9

Model	Fluency	Descriptiveness
ClipCap (From Scratch)	34.2	37.7
Ours (Text + Image)	**37.4**	**44.4**
Equal Quality	28.5	17.9

Table 7.2: **Left:** Results from human evaluation, showing annotator preference for our model vs. frozen ClipCap in terms of fluency and descriptiveness. **Right:** Results showing annotator preference for our model vs. finetuned ClipCap in terms of fluency and descriptiveness.

is perhaps expected given that tweets are generally not descriptions of the images they accompany. While users will occasionally recycle their tweets into the alt-text field, such examples are filtered out by our preprocessing. Nearest neighbor achieves slightly better performance. This makes sense as there are certain styles of image, for instance selfies, that may appear frequently and have relatively similar captions, although this approach likely makes frequent mistakes when it comes to finer details.

Of the neural methods, we see that frozen ClipCap scores lowest, almost certainly due to domain mismatch in both the distribution of images and text. Finetuning on our dataset does lead to some improvement, bringing ClipCap ahead of the naive baselines on most metrics. Perhaps supporting the hypothesis that our Twitter dataset is fundamentally mismatched from Conceptual Captions, we find that a ClipCap trained from scratch on our dataset performs about on par with the finetuned one depending on the metric. We see similar performance to frozen ClipCap from BLIP-2, which despite its extensive pretraining on data not available to other systems, fails to produce alt-text with sufficient detail and accuracy for this dataset.

However we see by far the strongest results from our approach which combines the mapped image prefix with the text of the tweet as input to a decoder, achieving more than 2x on BLEU and 4x on CIDEr. This is also ahead of the ablation which disregards the image entirely and only uses the tweet text in the prefix, demonstrating that these two sources of information *stack* with one another. The competitive performance of the text-only baseline demonstrates the strong amount of necessary context present in the original post, while the fact that it outperforms merely copying the tweet text emphasizes that alt-text is nonetheless functionally different in many cases from just another caption. Randomizing the retrieved tweet does hurt our model's performance as well. We find qualitatively that under this setup our model is sometimes still able to produce a reasonable caption presumably based solely on the signal from the image features, although it will occasionally insert spurious details from the random tweet, so it's not completely robust to irrelevant input tweet captions.

7.6.2 Decoding Ablation

Table 7.1 also shows an ablation of various decoding strategies for our multimodal model. Greedy decoding does poorly, and trigram blocking helps on some metrics but hurts on others. Beam search does not do better than greedy if allowed to repeat trigrams, but the specific combination of beam search without repeats does extremely well. Reranking the top 5 candidates from beam search with ROUGE-L similarity to the tweet text offers minor improvements on most metrics, but hurts performance on CIDEr. We also observe similar rankings of these methods for ClipCap.

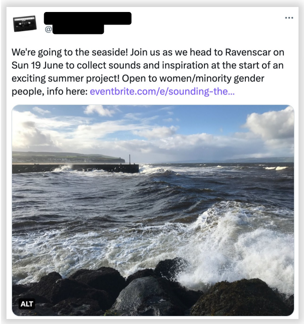
7.6.3 Human Evaluation

Note that while the automatic metrics are informative, the absolute scores are below what these same systems achieve on datasets such as Conceptual Captions (Sharma et al., 2018). We suspect this is largely due to the noisiness of gold references. It's worth emphasizing that Twitter users are not explicitly trained in how to write alt-text, and as a result the level of quality is extremely variable (Gleason et al., 2019). Furthermore, some users write alt-text that functions more as a "second caption" than as a literal description of the image's contents. This means that for an observably large proportion of our data (although the exact amount is difficult to quantify), the gold reference is either undesirable to match or idiosyncratic enough that measuring a candidate caption's similarity to it is not truly reflective of how well that caption describes the image. Going through the results manually, we often find instances where the model's output is actually more appropriate as alt-text than what the user wrote, or where they are both functional but phrased differently enough that there is little string overlap, leading to low scores that do not necessarily reflect output quality. We thus conclude that these automated metrics, while informative, are not sufficient for assessing model performance.

We therefore also perform two rounds of human evaluation as described above, with results shown in Table 7.2. On the left we observe that in roughly two thirds of images the human annotators judged our model's output to be both more fluent and descriptive than that of the frozen ClipCap baseline, and annotators generally agreed on rankings. To the right we see similar rankings albeit with a tighter margin when we compare our system against the ClipCap trained from scratch on our data. This matches the findings of our automated metrics, and supports our hypothesis that conditioning on context from the tweet yields higher quality descriptions.

7.6.4 Qualitative Inspection

In order to analyze what specific types of differences lead to these improvements and also what kinds of errors the model still makes, we now describe some observations from qualitative in-

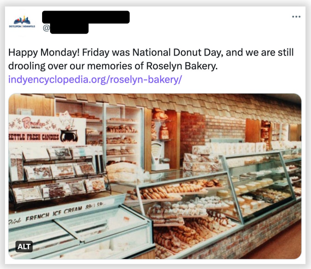

[redacted]
@[redacted]

We're going to the seaside! Join us as we head to Ravenscar on Sun 19 June to collect sounds and inspiration at the start of an exciting summer project! Open to women/minority gender people, info here: eventbrite.com/e/sounding-the...

Gold: A photo of the coast with waves crashing onto rocks and a pier in the background.

Ours: A view of the sea from the top of Ravenscar. The waves are crashing against the rocks and the sky is blue with white clouds.

ClipCap: stormy seas return after western christian holiday

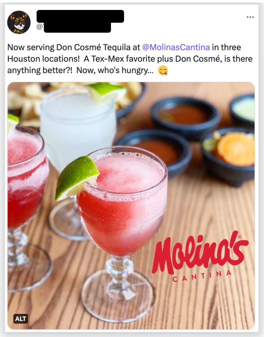

[redacted]
@[redacted]

Happy Monday! Friday was National Donut Day, and we are still drooling over our memories of Roselyn Bakery. indyencyclopedia.org/roselyn-bakery/

Gold: Color photo of the interior of a Roselyn Bakery store. Image credit: Indy Star

Ours: A bakery with a rack of donuts on the counter.

ClipCap: a bakery in the neighborhood

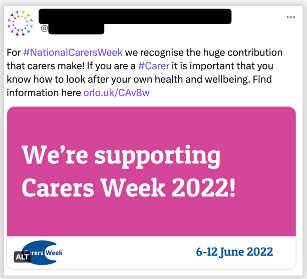

[redacted]
@[redacted]

Now serving Don Cosmé Tequila at @MolinasCantina in three Houston locations! A Tex-Mex favorite plus Don Cosmé, is there anything better?! Now, who's hungry... 😋

Gold: Delicious pink margarita sits on a table full of salsa and chips

Ours: Two margaritas on a table.

ClipCap: cocktail is the perfect drink for a party!


[redacted]
@[redacted]

For #NationalCarersWeek we recognise the huge contribution that carers make! If you are a #Carer it is important that you know how to look after your own health and wellbeing. Find information here orlo.uk/CAv8w

Gold: Pink backgrou nd with the words We're supporting Carers Week 2022. The Carers Week logo and dates 6-12 June are at the bottom of the image

Ours: National Carers Week 2022. 6-12 June.

ClipCap: we are growing exponentially as a brand

Figure 7.3: Selected tweets with the user-written alt-text alongside our prediction and ClipCap's. We see that by conditioning on the tweet text, our model is able to focus on relevant details in the images, reference named places, and provide better transcription despite not being trained on OCR.

spection. Figure 7.3 shows some example tweets from our dataset, accompanied by the actual user-written alt-text as well as our model and a frozen pretrained ClipCap's predictions. We generally see that our system is able to focus on contextually relevant details, such as the donuts in the bakery. It's also able to leverage text in the tweet to more accurately transcribe digital text in graphics. However it does at times still struggle with counting, and sometimes omits thorough visual details. ClipCap provides reasonable albeit simplistic captions for more natural images, but struggles with images that fall farther outside the Conceptual Captions distribution. It is also unable to incorporate information from the tweet itself, or use that to influence which parts of the image to focus on.

7.7 Limitations

This work still has several limitations which are important to acknowledge. The dataset we collected and trained our model on is variable in quality as the gold references are user-written, and this necessarily introduces a gap between the training objective and actual alt-text best practices. It could also be improved with more sophisticated identification and removal of bots, and using entity linking instead of just named entity recognition to provide more finegrained descriptions of people in lieu of their names. The model itself, while it frequently produces reasonable captions, still at times makes mistakes regarding specific details, is imprecise at transcribing text found in images as it does not contain an OCR system, and occasionally outputs the tweet text verbatim which despite at times serving as a reasonably descriptive caption is nonetheless redundant. Since our system conditions on an accompanying social media post, it is not easily adaptable to settings where images appear without any textual context, or where the specific section of text meant to correspond to the image is ambiguous such as in a long form web post or news article. We also rely on having a pretrained image encoder which means that the visual features extracted cannot be informed by the context, they instead must feed into the language model as independent information signals. Finally, it's worth emphasizing that our human evaluation is performed using sighted reviewers, and we therefore cannot make any claims about the downstream efficacy of our method without a larger scale study using BLV users which was unfortunately beyond the scope of this work.

7.8 Ethics Statement

Given these limitations of our approach we find it critical to emphasize that simply applying this method as-is to producing alt-text captions in an actual social media production setting would be premature and likely harm the user experience for BLV users. Accessibility technology must demonstrably be of more utility than other alternatives in order to justifiably be adopted, and while we hope that our work is a step in that direction, we have not yet conclusively determined that that bar has been passed. The research we perform here has the potential to be co-opted by social media platforms as a cheaper alternative to hiring manual alt-text writers, or having users write their own alt-text as is Twitter's current approach, which would arguably worsen the online experience for BLV users whose accessibility needs are already to a large extent unmet in this regard. Furthermore as with any open-ended text generation system trained on social media, our model does have the capacity to produce toxic language, regurgitate sensitive user information, and produce insensitive and harmful captions for personal photos.

7.9 Conclusion

In this work we presented an approach for alt-text generation that conditions both on image features and the raw text of the corresponding social media post, and collected a new dataset for this task from Twitter. Despite limitations discussed above, we believe this work is an important step forward and has potential for various downstream applications. While the most obvious use case is to simply use the model to produce alt-text for images that do not have any, we could also see this approach being used in concert with human annotators, perhaps providing a starting point for captions. Additionally, since our model can form a conditional likelihood on the caption, we could use it to flag poorly written alt-text and notify users to reread the guidelines and rephrase their description. In summary, we believe this is an important task and hope that our work encourages future research that leverages the surrounding social media context in which these images appear.

Chapter 8

Retrieval Guided Music Captioning via Multimodal Prefixes

8.1 Introduction

In this final chapter of Part III we will take the approach considered previously and extend it to the domain of music. Here we no longer have an equivalent of tweet text, but will instead seek to form our multimodal prefixes by way of retrieving a neighboring example from the train set that can function as a starting point for our captioner.

Captioning is an important multimodal task that aims to generate natural language descriptions for data of other modalities. While this task has been well-studied for various domains including images, speech, and video, one domain that is less thoroughly researched by comparison is music. Thorough musical captions span several different aspects of the song – a good description needs to be able to identify tempo, instrumentation, genre, and/or emotional character, while rendering this information in concise natural language.

Being able to produce these captions automatically is important for a variety of reasons. Musical descriptions are helpful for curation and promotional purposes, and also play a meaningful role in *accessibility*. For example, much audiovisual content including streaming media, movies, and television is made accessible to deaf and hard-of-hearing users through the use of closed captions. Beyond simply transcribing speech, these captions must also convey other forms of audio such as background noises and music. Music of course has many attributes that would be apparent to a hearing user, all of which contribute to the emotional content of the scene. Yet even manually written captions frequently fall short in their descriptiveness (Kim, 2020; Davidson, 2022) despite their importance in conveying meaningful information to the viewer (Aleksandrowicz, 2020; Revuelta et al., 2020). Musical descriptions are also useful for categorization, search, and many other applications where the audio is not renderable or is computationally intractable to directly reason over. Further, there is also growing interest in using text representations in *creative musical applications* – for instance, in systems that can produce and edit music in collaboration with humans using natural language as an intermediary modality (Zhang et al., 2023). There is therefore reason to want systems that can at large scale produce higher quality captions for music.

Prior research on music captioning as a task has predominantly explored either retrieval or generative methodologies (He et al., 2022a; Manco et al., 2021). Retrieval-based models are inherently tethered to the distribution in their train set. On the other hand, generative models require large training sets and cannot as easily take advantage of near matches to datapoints previously seen. They also require extensive pretraining and are less adaptable to frequently changing corpora. Therefore, an approach that combines these ideas to mitigate their respective downsides is desirable.

Unfortunately, building such a system requires large scale and freely available captioning datasets for popular contemporary music, which are currently underutilized in prior work on this task. For example, **Mus**Caps (Manco et al., 2021), a recent step forward in this domain, uses a private dataset of production music with just 6035 audio-caption pairs. In order to both evaluate our method robustly and also encourage future work in this direction, we instead use a public dataset of ~250k song-caption pairs scraped from the royalty-free music website Audiostock, spanning a wide variety of genres and moods [1]. We also perform further evaluation on Agostinelli et al. (2023)'s smaller dataset of 5.5k pairs.

This larger scale enables the modeling strategy that we put forward here, which we call RGMC (**R**etrieval **G**uided **M**usic **C**aptioner). RGMC takes inspiration from prior work on image captioning and contrastive pretraining. Namely, ClipCap (Mokady et al., 2021) previously introduced a strategy in which an image is embedded via a pretrained CLIP encoder network (Radford et al., 2021) and then projected to a vector sequence in word embedding space in order to be conditioned on by a language model decoder, specifically GPT-2 (Radford et al., 2019). We similarly use a pretrained audio encoder from CLAP (Wu et al., 2023) to build a music feature prefix for our songs. We then further augment this encoding with a retrieved candidate caption from our training set, allowing our decoder to condition on signal from both and either combine them or back off to one if the other is uninformative. By allowing the model to incorporate both audio features and text examples through a multimodal prefix, RGMC can generate captions that are both well-formed and stylistically similar to those in the dataset while also being adaptable to the specific details of each song.

We evaluate this model across several automatic captioning metrics, and demonstrate large improvements over a strong nearest neighbor baseline as well as prior neural work. We also conduct a round of human evaluation which we similarly lead in both measures of fluency and

[1]https://github.com/LAION-AI/audio-dataset/blob/main/laion-audio-630k/README.md

descriptiveness. Additionally, we conduct various ablations which test the brittleness of our approach and measure the relative contributions of the audio and text components.

In summary, this chapter makes the following contributions: we (1) Put forward a novel method for captioning music clips based on a retrieval guided generative pipeline (2) Demonstrate improvements in generated caption quality across two datasets in terms of both automatic metrics and human eval.

8.2 Related Work

Audio captioning more broadly has been relatively well studied, with a variety of published datasets and models for that task (Mei et al., 2022). However, noticeably less work has gone into studying music captioning specifically, and there are relatively few open datasets and models built with music-to-text applications in mind. At the same time, music captioning as a task differs substantially from general audio captioning. The language of musical description is focused and rich, and therefore musical captions can contain complex detail and analytical insights about tempo, harmony, and timbre.

Music retrieval systems do exist both for captioning and other multimodal tasks. He et al. (2022b) and He et al. (2022a) take advantage of a retrieval strategy to augment a text decoder, although their method conditions on song lyrics, which many songs (especially those used as background music) do not have. McKee et al. (2023) developed a model that retrieves music potentially relevant to a provided video, although they do not tackle captioning. There is also much work on using retrieval-augmentation generation in the context of language models, although this is largely focused on unimodal settings (Lewis et al., 2020; Borgeaud et al., 2021; Khandelwal et al., 2019).

One of the most directly relevant works to our own is **Mus**Caps (Manco et al., 2021) a generative system for music captioning. Despite achieving strong performance and being one of the most prominent models for this task, they use a relatively small scale and private dataset, and do not incorporate retrieval into their pipeline. By scaling up to a larger corpus, we can by contrast take advantage of retrieval strategies. Other related work on generative music captioning has focused on classical music (Kuang et al., 2022), although this limits dataset size and contemporary applicability. There is also recent work on using LLMs for data augmentation for this task by generating pseudo-captions from a taglist (Doh et al., 2023). While this method operates under an incomparable task setup as it assumes tags are available and does not have a way of conditioning on audio, we do compare to their supervised music-to-text baseline. There has been some similar work on building interactive QA systems capable of understanding audio, speech, and music as well (Deng et al., 2023; Gong et al., 2023b,a).

Prior work has also studied contrastive text-audio representation learning for pretraining purposes. Audio and text representations from CLAP (Wu et al., 2023) have performed well on a variety of downstream tasks, such as audio classification, music genre classification, and sound event detection. MuLan (Huang et al., 2022) is another recent work in this space, and has even been used for pseudo-labeling of music for training diffusion models (Huang et al., 2023). CALL (Manco et al., 2022) has also seen similar success. This underscores the versatility of jointly learned embeddings and the potential they hold for music captioning.

Our architecture also takes inspiration from image captioning work that similarly takes advantage of contrastively pretrained multimodal encoders. CLIP (Radford et al., 2021) has seen tremendous success across several vision tasks. Its image encoder was utilized by ClipCap (Mokady et al., 2021) which put forward the idea of projecting its visual feature embeddings into word embedding space in order to be conditioned on by a text decoder. Srivatsan et al. (2024b) further augmented this projection with relevant text in order to create prefixes representing multimodal features for the downstream task of alt-text generation.

There does also exist a recent body of work on the adjacent task of playlist captioning, which aims to assign a description to not just one song but an entire set of them (Choi et al., 2016; Gabbolini et al., 2022; Kim et al., 2023; Doh et al., 2021). This requires models to infer similarities across songs instead of prioritizing individual specificity.

Of course beyond the technical component, this task necessarily has meaningful considerations from an accessibility standpoint. Foley and Ferri (2012) write on the delicateness of assistive vs accessible technologies to either improve or diminish access for people with disabilities in a broad sense. Aleksandrowicz (2020) conducted a recent study on the way that deaf and hard-of-hearing people perceive the emotions of film music from captions, motivating the importance of this domain. Revuelta et al. (2020) highlighted the difficulties in transmitting sufficient emotional information through traditional captioning alone, with Lucía et al. (2020) proposing vibrotactile captioning as an alternative.

8.3 Model

At a high level, RGMC works by building a prefix in word embedding space and then feeding that as input to a language model decoder which autoregressively continues the sequence to generate the predicted caption. See Figure 8.1 for a visualization. This prefix consists of two halves: the candidate caption $\tilde{y}$ retrieved by nearest neighbor, and an audio embedding of the song x that has been projected to a sequence of vectors in word embedding space (but does not exactly map to any particular token sequence) which we refer to as $e(x)$. Our aim is that $e(x)$ will capture the specific details of the audio, and $\tilde{y}$ will provide a stylistic template or perhaps include reference

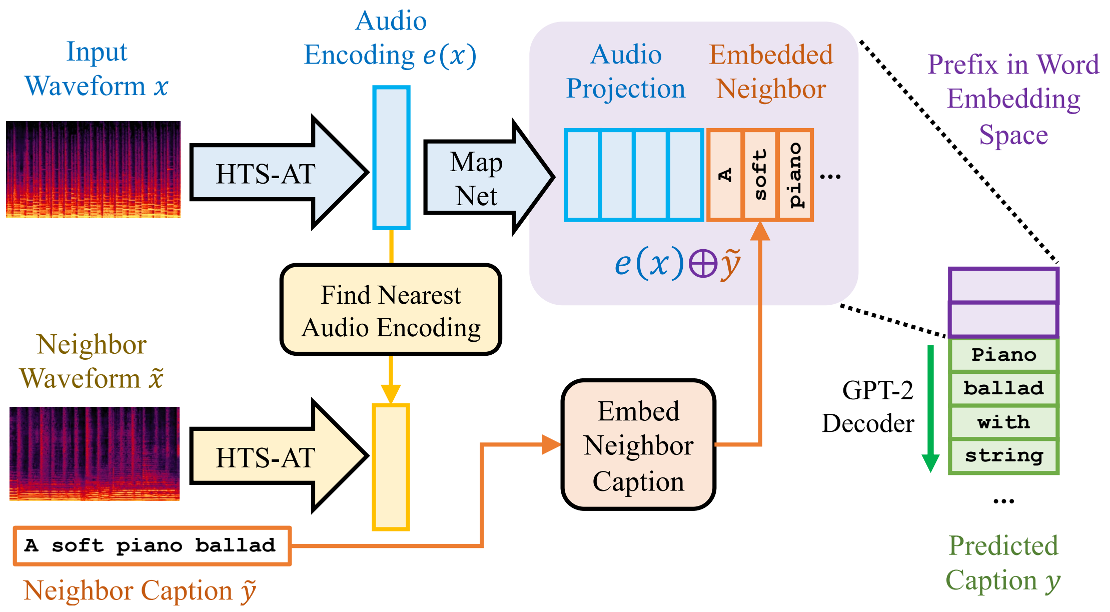


Figure 8.1: Visualization of RGMC, our music captioning model. We first encode the input song using an HTS-AT audio encoder, and project it through a mapping network to a sequence of word embedding sized vectors. We separately retrieve the song in train with the most similar audio encoding and postpend its caption to the audio prefix. This gets passed to GPT-2 which autoregressively outputs the predicted caption.

to uncommon features not picked up on by the audio encoder, and that by combining these two sources of information our decoder can produce a more accurate caption.

More formally, our modeling goal is to formulate and optimize a distribution $p(y|x, \tilde{y})$ over the caption y conditioned on both the audio of that particular song x as well as a corresponding candidate caption $\tilde{y}$ with respect to our model parameters. We represent x as a 10 second waveform cropped from the middle of the full song with a sample rate of 48 kHz such that $x \in \mathbb{R}^{480\text{k}}$. We will optimize our model using a dataset $D = \{(x_1, y_1), ..., (x_N, y_N)\}$ consisting of N songs along with corresponding text captions. In this case, our set of candidates is simply our training set excluding that example if it is contained, or $\tilde{D} = D - \{(x, y)\}$. This prevents our model from learning a degenerate solution where it simply copies the nearest candidate and ignores the audio features.

Over the rest of this section we will walk through the three basic steps of our pipeline, namely the encoding and projection of audio features into a conditionable prefix, the retrieval of a nearest neighbor candidate caption, and finally the decoder itself.

8.3.1 Audio Conditioning

In order for our decoder to be able to condition on audio information from the original waveform, we need to be able to map a feature representation of it to a manifold that our language model can reason over. This process is shown in blue in the upper left portion of Figure 8.1. Formally, we desire an encoder function $f : \mathbb{R}^{480\text{k}} \rightarrow \mathbb{R}^{512}$ which maps these waveforms to a music feature space.

For the purposes of captioning audio data, a cross-modal representation that aligns language and audio features is extremely desirable. Therefore, we specifically parameterize $f(x)$ via the audio encoder from CLAP (Wu et al., 2023), which is itself an HTS-AT (Chen et al., 2022) network followed by an MLP projection layer. While we only directly use the audio encoder, CLAP is explicitly trained to maximize the dot product of corresponding audio and text embeddings (while also incorporating erroneous pairs as negative samples) which makes it an inherently well-suited pretrained model for our purposes.

Having obtained the audio encoding $f(x)$, we use a mapping network to project it to word embedding space. This takes the form of a function $g : \mathbb{R}^{512} \rightarrow \mathbb{R}^{10\times d}$ where $d = 768$ is the token dimensionality that our language model expects. We parameterize g with a simple two layer MLP. Together then we can define our prefix $e(x) = g(f(x))$, a text-like sequence that contains audio features from the song.

8.3.2 Retrieve and Edit

Given however the relatively shallow diversity of song captions, it becomes useful to allow RGMC to condition on a candidate caption suggested by a retrieval system (depicted to the lower left in yellow in Figure 8.1). This allows the decoder to learn simple edits instead of having to construct the entire caption from scratch. It also biases it towards learning a copy mechanism which it can back off to. We can choose a neighbor $\tilde{x}$ for any song based on the dot product of the audio embeddings $f(x)$. Supposing that for every song in our candidate set we have a corresponding caption $\tilde{y}$, this can be concatenated together with the audio prefix we already have to obtain a multimodal prefix $e(x) \oplus \tilde{y}$.

Of course, the addition of a search system within our pipeline complicates training. Not only is the nearest neighbor retrieval non-differentiable as it involves the discrete selection of a single candidate, it is also computationally expensive to have to execute for each batch during training as the encoder (and therefore the embeddings being compared) will change after each gradient step. In order to circumvent this, we simply do not backprop through the candidate selection process, and instead assume that it is a fixed observation that the model conditions its caption on. That being said, in order to keep the candidate embeddings relatively up to date over the course of training, we recompute them at the beginning of each epoch based on the current parameters of the CLAP encoder. The encoder therefore does not receive explicit learning signal from retrieval, but our hope is that since it does receive loss from captioning it will nonetheless create feature representations that would place songs with similar captions next to each other in its embedding space.

8.3.3 Decoding

Having computed the multimodal prefix, our GPT-2 (Radford et al., 2019) decoder simply conditions on it as a textual prompt and autoregressively outputs the predicted caption, as shown in green to the right of Figure 8.1. Putting together our notation from earlier, this yields our desired distribution over the caption:

$$\begin{aligned}\hat{y} &= \arg\max_{y} \log p(y|x, \tilde{y}) \\ &= \arg\max_{y} \log p(y|e(x) \oplus \tilde{y}) \\ &= \arg\max_{y} \log p(y|g(f(x)) \oplus \tilde{y})\end{aligned}$$

At train time, we perform teacher forcing and backprop the cross entropy loss between the logits and the gold reference tokens through the network to obtain gradients for GPT-2, the mapping network, and HTS-AT. At test time, we employ beam search with a beam size of 5, and also perform trigram blocking as described by Paulus et al. (2017).

8.4 Implementation Details

We rely on HTS-AT's official pretrained music checkpoint which is trained on a composite collection of music datasets contained within the LAION-Audio-Dataset[2]. For our audio encoder, we use the projected 512-dimensional contrastively trained embedding, which is itself the output of a 1024-dimensional CLAP encoding passed through an MLP layer.

For the GPT-2 (Radford et al., 2019) decoder, we use the publicly available `gpt2' checkpoint from HuggingFace.

We use the public implementation of **Mus**Caps[3] which we train from scratch on the Audiostock data due to the unavailability of their dataset and checkpoints, performing preprocessing and tokenization as they describe.

[2]https://github.com/LAION-AI/audio-dataset
[3]https://github.com/ilaria-manco/muscaps

We train our own models with a batch size of 32 and the Adam optimizer (Kingma and Ba, 2015) with a learning rate of $1e-5$. Our implementation is written in PyTorch (Paszke et al., 2017) and runs on a single NVIDIA A6000 GPU in roughly 24-36 hours, and inherits some code from the ClipCap (Mokady et al., 2021) repository. We perform early stopping based on our loss on the held out validation set.

8.5 Dataset

CLAP introduced a 250k dataset (Wu et al., 2023; Chen* et al., 2024) of music tracks and associated metadata scraped from Audiostock[4], which we use for our experiments. These tracks are accompanied by both a short text caption and a long text caption, as well as three tag lists, one describing purpose, one describing impression, and one general. For our purposes we only consider the short text caption, as the long text is inconsistent in format and quality, and the tag lists are not natural language. These tracks have an original sampling rate varying from $32-48\,\mathrm{kHz}$, which we resample to a fixed $48\,\mathrm{kHz}$. The total duration of the raw dataset is $11\,305$ hours. Since the audio input is processed by CLAP which resamples all audio inputs to $48\,\mathrm{kHz}$ to extract the embeddings, our proposed music caption model can support audio inputs with arbitrary sample rates. Following **Mus**Caps (Manco et al., 2021), we standardize the formatting of the captions by casting them to lowercase and removing all punctuation.

One issue with Audiostock as a source of music/caption pairs is the prevalence of duplicate and near-duplicate captions. We frequently found instances of songs with almost identical captions, for example with the only difference being a version number at the end. We therefore needed to remove these redundant examples, both to prevent train/test leakage and also to keep the model from overfitting to those captions. Following Lee et al. (2022) we computed the Jaccard similarity between the set of 5-grams within all pairs of captions and marked any that scored above 0.8 as duplicates. We agglomeratively clustered duplicates based on this threshold, and for each cluster retained only the datapoint with the smallest ID number. This ultimately left us with $200\,170$ total datapoints, split into $184\,242$ train, 7925 val, and 8003 test.

In addition to Audiostock, we also perform some experiments on **Music**Caps (Agostinelli et al., 2023), a much smaller scale dataset of 5.5k music-text pairs (not to be confused with our baseline system **Mus**Caps). **Music**Caps specifies a roughly even train/eval split. We additionally split off 487 from the train set to use as dev for hyperparameter tuning and early stopping. It's worth noting that while the captions in **Music**Caps are lengthy and descriptive, they also reflect a high degree of annotator subjectivity (Lee et al., 2023).

[4]https://audiostock.net/

8.6 Experiments

Having described our model, we can now perform captioning experiments to evaluate our method. In this section we will go over the setup of these experiments including the baselines we will compare against, and the metrics we will use to evaluate our output against the gold references.

8.6.1 Baselines

We compare RGMC against a variety of baselines. These include naive retrieval methods, generative prior work, and ablations of the modalities represented in our prefix.

Random Caption The first of these is a Random Caption system which just returns a randomly selected caption from the train set. This is effectively a lower bound for this task, and measures to some extent the uniformity of our data.

Nearest Neighbor We also try a Nearest Neighbor system, identical to the one inside our model's pipeline. It encodes every song in train via a frozen HTS-AT, and when given a song at test time retrieves the one with the highest dot product against its embedding, and returns its corresponding caption verbatim.

MusCaps Our primary neural baseline is **Mus**Caps. This model differs from ours in a few key ways, most notably in that it is not retrieval guided and uses an LSTM decoder as opposed to a transformer LM. The encoder and decoders of our systems are also pretrained on different data. We retrain **Mus**Caps on the Audiostock dataset using their publicly available implementation.

LP-MusicCaps Supervised LP-MusicCaps developed a pseudo-captioning system as well as an audio-to-text music captioning model. They listed numbers for their supervised captioning system trained and evaluated exclusively on the **Music**Caps dataset, which we compare to here.

LTU Gong et al. (2023b) and Gong et al. (2023a) are QA systems that both have the ability to perform music captioning. Their performance on the **Music**Caps dataset was recorded by Deng et al. (2023), which we show here.

RGMC (Audio Only) The first of our baselines forgoes the nearest neighbor candidate retrieval, and instead simply feeds the projected audio prefix $e(x)$ directly into the decoder. This lets us measure the relative predictive importance of the audio features in our prefix, and indicates the level of quality achievable in the absence of an on hand candidate set to perform retrieval over.

Model	Descriptiveness	Fluency
MusCaps	29.0	20.4
RGMC	**63.4**	**49.5**
Equal Quality	7.5	30.1

Table 8.1: Results from human evaluation, showing annotator preference for RGMC vs. MusCaps by fluency and descriptiveness.

RGMC (Text Only) We also examine a version that masks out the audio encoding, and only conditions on the candidate. This measures the redundance between the information provided by the neighbor caption and the audio features themselves. Note that this model is still able to indirectly condition on the audio insofar as the retrieved candidate describes it (and it being chosen based on those audio features).

8.6.2 Metrics

Previous work on music captioning (as well as other forms of captioning) evaluate using standard string similarity metrics that compare the ngram overlap of the model's generated caption with a gold reference. Since we have the user written captions for songs from the uploader, we can treat these as gold and measure these same scores. Specifically, we measure performance on BLEU@4 (Papineni et al., 2002), METEOR (Denkowski and Lavie, 2014), ROUGE-L (Lin and Och, 2004), and CIDEr (Vedantam et al., 2015) using a public fork of the MS COCO (Lin et al., 2014) evaluation repository[5]. We also use two neural evaluation metrics that measure the alignment of our generated captions under a pretrained contrastive model, specifically CLAP (Wu et al., 2023) using the default checkpoint, following a similar approach to Chen* et al. (2024). Specifically we measure the average cosine similarity between CLAP's embedding for the music and our predicted text (which we call CLAP A-T), as well as the similarity between its embedding of our predicted text and the gold caption (which we call CLAP T-T). The former is an especially insightful metric as unlike all the others, it is not measuring the similarity between our output and the subjective human written reference but rather directly measuring the similarity between that output and the original song. We also include the similarity between the gold caption and the audio as a rough proxy for human captioning performance on the former. We find that these metrics qualitatively line up with human judgement of quality for this dataset, and are mostly comparable in range to those reported for existing systems on other datasets for this task.

[5]https://github.com/LuoweiZhou/coco-caption

Model	CIDEr	BLEU@4	METEOR	ROUGE-L	CLAP A-T	CLAP T-T
Gold Caption	1000	100	100	100	22.01	100
Random Caption	2.81	0.11	1.39	3.26	10.86	29.50
Nearest Neighbor	57.41	**5.93**	8.69	16.79	21.62	44.22
MusCaps	26.98	2.05	6.62	14.22	24.52	42.47
RGMC (Audio Only)	41.63	3.08	8.56	16.78	**27.68**	45.61
RGMC (Text Only)	47.04	3.65	7.72	15.15	24.69	43.63
RGMC	**59.69**	4.96	**9.83**	**19.15**	27.00	**46.36**

Table 8.2: Results from RGMC on Audiostock as compared to Nearest Neighbor and **Mus**Caps. We additionally provide an ablation of our system that ignores the candidate caption and only conditions on audio, and similarly one that only conditions on the candidate without the audio. Scores are also shown for returning the gold and a randomly selected caption from train. CLAP A-T indicates the CLAP score between audio and predicted text, and CLAP T-T indicates the score between the predicted and gold text.

8.7 Results

We now discuss the results of our experiments, including automatic captioning metrics, an ablation of both our multimodal prefix and our retrieval method, human evaluation, and a qualitative inspection of our generated descriptions.

8.7.1 Human Evaluation

Despite the usefulness and scalability of automatic metrics, the true test of output quality is human judgement. We therefore perform a survey of human annotators comparing the quality of captions generated by RGMC against our neural baseline **Mus**Caps. We randomly selected 31 songs from the Audiostock test set and presented them to a group of three hearing human annotators via an online survey. For each song the annotators were provided a 10 second clip from the middle of the track (i.e. the same input fed to the model's encoder) as well as two candidate captions for that song, one from each model. The order of the models' captions was randomly shuffled and they were not labeled. Annotators were asked which of the two captions was the more accurate description of the provided song, and also which of the two was more fluent in terms of grammar and phrasing. They could answer that the two were too similar to distinguish, but discouraged from doing so frequently.

We report the proportional votes for each model across all annotators in Figure 8.1. We can see that the annotators had a strong preference for our system with respect to descriptiveness, and infrequently selected that the two models were indistinguishably similar. In terms of fluency

Candidate	Copy %	CIDEr	BLEU@4	METEOR	ROUGE-L
Random	3.97	35.66	2.70	7.72	15.32
NN	12.66	59.69	4.96	9.83	19.15
Gold	9.76	231.28	22.04	20.25	36.42

Table 8.3: Comparison of various methods for selecting the candidate caption, including random, nearest neighbor search (default), and returning the gold caption itself. We show scores on the same metrics as well as the Copy %, or the frequency of our decoder returning an identical string to the candidate.

our model was still preferred by plurality although by a much smaller margin, with annotators reporting that the two were similarly fluent almost a third of the time. This could be explained by the high variability in the fluency of the gold descriptions themselves. We find annotators are in unanimous agreement 35.5% of the time on descriptiveness and 38.7% of the time on fluency (chance would be 11.1%).

8.7.2 Qualitative Inspection

In Table 8.4 we show some examples of RGMC's predicted captions as compared to **Mus**Caps and the gold reference. We generally observe that we do slightly better at identifying musical genre and repeat adjectives less frequently than the baseline. We can also see from looking through the gold examples that they tend to vary in which aspect of the music they primarily focus on describing (e.g. instrumentation, mood) and also in the fluency of their style, with some reading closer to a list of tags than a natural language description. This variation in captioning style does of course manifest in the output of the models themselves.

8.7.3 Automatic Metrics

Table 8.2 shows results from our system alongside our baselines and ablations for Audiostock. The full RGMC model gets the highest performance on most metrics. Nearest neighbor is a fairly competitive baseline, perhaps to some extent due to dense genre clusters of similar songs present in our data. It's also worth emphasizing that BLEU is a precision driven metric as opposed to for example ROUGE which is based on recall. The fact that nearest neighbor does better on the former but not latter could mean that while it can find similar songs with captions that do not contain incorrect information, it may be less able to produce idiosyncratic details that the gold may mention. This is also supported by its relatively poor performance on CLAP A-T. **Mus**Caps does quite poorly by comparison, even underperforming the audio only ablation of our system. This could be due to its differences in architecture and pretraining. Our ablations also show that a

Song ID	Model	Music Caption
119693	Gold	`sad and lonely violin and piano sound`
	MusCaps	`a magnificent and moving orchestra that feels the beginning of the story`
	RGMC	`a sad ballad of piano and strings`
1702	Gold	`cute and nimble fantasy pop`
	MusCaps	`a heartwarming and cute reggae song`
	RGMC	`a little comical dark fantasy song`
128382	Gold	`light and catch a kind of electric jingle`
	MusCaps	`a song that makes you feel the beginning`
	RGMC	`danceable beat electro pop`
73107	Gold	`japanese style hip hop with slow tempo`
	MusCaps	`a refreshing and bright song with a refreshing acoustic guitar`
	RGMC	`easy to use japanese style bgm with koto and shakuhachi`
58045	Gold	`strings dissonance jingle horror`
	MusCaps	`a magnificent and magnificent song that feels the universe`
	RGMC	`horror bgm with a sense of tension`
1349464	Gold	`slightly sad music box with ambient sound`
	MusCaps	`music box and environmental sounds that match the rain scene`
	RGMC	`a music box that fits the sad scene with ambient sound`

Table 8.4: Comparison of RGMC's output captions against **Mus**Caps and the original gold reference for various songs in the Audiostock dataset. We generally see that our predictions are more accurate to the genre and other details, and also less likely to repeat adjectives and produce other grammatical mistakes.

meaningful amount of our system's performance comes from the retrieval component, although it is not sufficiently informative by itself. Some songs will naturally not have a close equivalent in the train set, and the retrieval method itself is imperfect and the caption it retrieves may not fully describe the audio content of the original piece. We also note that many of our systems achieve a higher CLAP A-T score than the gold; this is not entirely unexpected as the gold captions are not necessarily the only correct output for a given song, and there may easily be other captions that are more similar under CLAP's embedding space.

Table 8.5 shows results of our system and various baselines on the **Music**Caps dataset. This dataset has been evaluated on by some prior work which makes it a useful point of comparison. However at the same time, it is substantially smaller, which makes it far less ideal for assessing the performance of a retrieval guided approach such as ours, which benefits substantially from a large scale and diverse candidate pool. Nonetheless, we see that our system achieves the best performance on all metrics compared to the retrained **Mus**Caps as well as Doh et al. (2023) and Gong et al. (2023b,a). It's likely that on a small dataset like this, our model is better able to learn to produce fluent output by way of its retrieval mechanism. We also investigate the effects of transfer learning. Specifically we take RGMC trained on Audiostock and then further finetune it on the **Mus**Caps train set, and find that this yields a mild boost in performance.

Model	CIDEr	BLEU@4	METEOR	ROUGE-L	CLAP A-T	CLAP T-T
Gold Caption	1000	100	100	100	34.02	100
Random Caption	3.30	3.21	8.91	17.47	10.64	33.19
Nearest Neighbor	8.07	4.49	11.27	19.77	30.97	54.96
MusCaps	1.0	2.1	10.3	19.6	18.79	38.95
LP-MusicCaps Supervised	-	4.79	-	19.22	-	-
LTU	-	-	7.6	8.5	-	-
LTU-AS	-	-	6.0	6.3	-	-
RGMC	8.76	5.90	**11.85**	20.82	30.34	56.06
RGMC (Transfer)	**11.25**	**5.92**	11.83	**21.25**	**32.38**	**57.06**

Table 8.5: Results from RGMC on the **Music**Caps dataset as compared to Nearest Neighbor and **Mus**Caps, as well as reported results from the LP-MusicCaps supervised baseline, and LTU. Scores are also shown for returning the gold and a randomly selected caption from train. CLAP A-T indicates the CLAP score between audio and predicted text, and CLAP T-T indicates the score between the predicted and gold text. RGMC (Transfer) indicates our model pretrained on Audiostock and then finetuned on **Music**Caps. All other models except LTU are only trained on **Music**Caps, and we only include results from the literature that use the same test set and metric implementations.

8.7.4 Retrieval Method Ablation

In order to measure the downstream effect of the specific retrieved candidate we performed an ablation which replaces the nearest neighbor with a more or less informative alternative. We can use these as a way to artificially strengthen or weaken the retrieval aspect of our pipeline, and examine how much of an effect this has on downstream performance. This tells us how reliant the decoder is on the relevance of the candidate, how robust it is to retrieval mistakes, and how well it can take advantage of in-domain close matches at test time.

Random Candidate First we replace the candidate with a randomly chosen one from the train set. This lets us see how the model responds when the retrieval system does not recover anything relevant to the particular example.

Gold Candidate Second we try an oracle that replaces the nearest neighbor with the gold caption. This measures how well the system can take advantage of a "perfect" retrieval that returns the most possibly relevant caption.

We also list the frequency at which the model's generated caption exactly matches the retrieved candidate. This lets us measure to what extent the decoder is simply regurgitating the retrieved candidate as opposed to synthesizing it with the audio signal and producing something novel.

Results on Audiostock are shown in Table 8.3. We see that random does worst, followed by nearest neighbor, and then the gold oracle. Furthermore, we find that gold and especially random tend to copy the candidate exactly less frequently. This could indicate that when retrieval fails, the model is able to recognize that and back off to only conditioning on the audio, and avoid being distracted by the confounding signal from the "neighbor" caption. Similarly when it produces something desirable, the model is able to take advantage of that and enjoy a large boost in performance, despite not exactly duplicating it more frequently than it would otherwise. This shows that the quality of search has a large effect on downstream captioning, and there may be headroom left here.

8.8 Ethics Statement

The work presented here has the potential to affect broader impacts that we believe are worth discussing. If simply rolled out in its current state as a substitute for human captioning, it may lead to an overall worse experience for those who rely on those captions for accessibility purposes. Given the dataset that the model is trained on, it may also exhibit biases towards genres that are overrepresented within it, and may for example not generalize well to other styles. It is also likely to mirror potentially problematic descriptions that users write, as we do observe usage of Eurocentric phrases like "oriental" and "exotic" within the captions in the dataset. The model also has the potential to misgender vocalists. The copyright implications of training on the datasets we consider are at the moment relatively ambiguous.

8.9 Conclusion

In this chapter we put forward RGMC, a novel method for music captioning that combines retrieval and generative strategies. We trained and evaluated this system on a large scale dataset of music-caption pairs scraped from Audiostock, and demonstrated considerable improvements over prior approaches both in terms of performance on quantitative automatic metrics, and also human evaluation. This model could be useful for curation purposes, accessibility, and has the potential to play a strong role in human-in-the-loop systems for music creation and editing. There is however still significant headroom left on this task, and many avenues for future work exist that may continue to close it. The results from our ablation indicate that we may see further gains from improving the retrieval method, or perhaps even allowing the decoder to condition on a top-k list as opposed to simply the closest individual caption. The datasets also contain other sources of information that we have not yet utilized. For example, jointly training a predictive head on our encoder over the tag lists, or even using the short captions to pretrain learning generation of the longer ones may be fruitful. It may also be worth explicitly biasing the model towards generating interpretable prefixes, as one of the advantages of our approach is that it implicitly learns pseudo-textual audio embeddings. Finally, in order to make strong claims about the usefulness of our system from an accessibility perspective we would need to conduct an evaluation using deaf or hard-of-hearing annotators, which was unfortunately beyond the scope of this work.

Chapter 9

Conclusion

In this dissertation, we presented several new approaches for modeling complex intra-corpus data dependencies at varying levels of granularity for multiple language-adjacent tasks. We applied these in turn to a variety of tasks that largely lie at the intersection between language and other modalities. In all settings, we demonstrated that current approaches are leaving headroom on the table by not fully utilizing known relations between datapoints.

9.1 Recap of Contributions

The specific contributions are as follows:

- Part I presented an approach to modeling glyphs via a factored representation strategy that significantly outperforms prior approaches that model glyphs independently. We put forward variants of this model that operate using both variational and adversarial frameworks, and also demonstrated how careful architecture design can instill useful inductive bias.
- Part II tackled two different tasks where the data has a temporal ordering that requires modeling structure in the output domain, specifically discursive topic modeling for social media and piano fingering prediction. In both cases we approach this with some form of message passing, which takes the form of mean field updates in the former, and a neural checklist in the latter.
- Part III introduced an approach to captioning that makes use of a multimodal input signal, incorporating features not just from the image or song, but also from a relevant text source. We in both cases combined those features and fed them into a language model decoder by projecting them from the alternate modality into word embedding space. We demonstrate that this is a viable and generalizable strategy for building language-like pseudo-prefixes, which can leverage unimodal pretraining in a multimodal context.

9.2 Limitations and Broader Impacts

While the work presented here represents large strides forward, it still has existing limitations that must be noted. See the corresponding chapters for more detail.

All of our work on typography for example operates on pixel representations of images of glyphs, albeit with projections to extract more structural features. As a result, while we can synthesize images of glyphs, we cannot synthesize actual specifications for fonts in their native vector format, which does limit the deployability of such a system. Even if this issue was overcome, this work would still come with potential negative consequences. For example, it may incentivize designers to use automatically generated glyphs for writing systems they are not familiar with, which could potentially result in fonts that are disproportionately less readable for readers of uncommon languages. It's also possible that the ability to generate fonts automatically may make it harder for human font designers to find work, and potentially even lead to an overall reduction in the creation of visually novel fonts. Nonetheless, it is our hope that such technology can be ultimately beneficial if used not as a replacement for human designers but as a tool to fill in gaps in existing font specifications.

Our work on Reddit topic modeling is somewhat limited by the fact that it requires observed dependencies between comments in a thread. While this is reasonable for a forum like Reddit, it is less so for example on Twitter, where conversations may be more implicit, with users posting about a shared topic independent of any observable reply structure, or using alternate types of replies like quote tweets that would need to be modeled differently. There are also ethical considerations with training large scale models on social media data in the first place, which despite being public may not necessarily have been posted with the understanding that it could be used for such a purpose. We do believe however that such work is ultimately useful for analysis, recommendation, discovery of harmful content, and does not carry nearly the same safety risks as autoregressive approaches to such domains.

Our work on piano fingering prediction is inherently limited by the fact that it must represent music monophonically in order to model it as a sequence. The model makes no distinction between left and right hand parts, and does not have the ability to reason about their interactions with one another. We also rely on hand designed metrics for measuring fluency that despite correlating with human judgement may not be sufficient for measuring all the ways in which a generated fingering is or is not easily playable. That being said, our work still represents a meaningful step forward, and carries the validation of human expert judgement.

Finally the work on alt-text and music captioning while potentially useful from an accessibility standpoint may also cause harm to those dependent on language descriptions of images and audio if deployed prematurely. It is necessary to say that the point at which automatically generated captions are as good as human written ones has not yet arrived, and until then they will always be an inferior substitute. This system was developed with relatively minimal input from potential users of this technology, and therefore needs more rigorous evaluation. However we do believe that if deployed client-side in a way similar to existing apps for object detection, this technology could meaningfully assist a population whose online accessibility needs are currently unmet.

9.3 Future Directions

There are many possible avenues for future research based on the work presented here, some which could potentially address the limitations described above.

There is an obvious extension to our typography work that would replace a pixel-space decoder/encoder architecture with one that operates over a vector representation of the images. While the wavelet (and to a lesser extent DCT) projection alleviates the classical issues of VAEs outputting blurry shapes, we could receive harder guarantees of well formed glyphs by moving to a native representation. This would of course require an alternate model architecture. Autoregressive models today are far more powerful than those that existed at the time this research was initially conducted, and it remains an open question whether for example a pretrained LLM could be used here. We also consider rows and columns in our approach to be independent of one another, but this assumption is at times faulty. Scribal hands for instance correlate based on findplace, and while our model was able to uncover this, we might form better representations if we explicitly model those groupings, adding dependencies to the embeddings themselves. Fonts and character types also have implicit connections; we would expect two fonts of the same family to strongly correlate, as well as characters within the same writing system or linguistically related ones.

One potential followup to the work on social media modeling would be to extend it to consider multimodal replies. Images and GIFs are increasingly being used on both Reddit and other platforms interspersed with text, and the ability to model these and the way they relate to the language around them is important. One could imagine a fusion of the branching DDTM model with our alt-text model for one that can reason about images and text in a combined, discursive manner. Thinking in this direction also forces us to confront the fact that DDTM as it exists treats comments as bags of words, which while traditional for topic models of the time is an inherently limiting approach.

9.4 Closing Thoughts

Thinking more broadly, the findings presented here do lend insight to the overall heading of the field at the current moment. General purpose LLMs and so called "foundation models" dominate the academic and especially industrial landscapes. However our work showcases the usefulness of bespoke solutions that deliberately trade task-agnosticism and promptability for performance gained through design choices inspired by the underlying goal itself and the ways in which existing systems fail. Also, the overwhelming paradigm at the moment is to train large scale unimodal sequential systems on vast amounts of text without explicitly reasoning about the connections within corpora. We argue here that models that have the ability to branch, and to model images, audio, and video not just as prompts to a text system, but in an interleaved, first-order, and generative manner are the true path to reasoning about online data in the same way that it is actually created.

This doesn't however mean that there is nothing useful that we can gain from these large pretrained foundation models. On the contrary, we find that rather than being opposing strategies these can in fact be quite synergistic. Many if not all of the systems presented here were fairly modular, especially those that were built on a PGM framework. By this we mean that while our broad model designs may call for certain distributions or functions, they do not require them to be parameterized in any specific way. For example, the decoder network contained within our typography model in Part I merely requires any conditional distribution over an image given two input vectors. And while previously the dominant architecture had been a transpose convolutional network, we could easily reparameterize that distribution with a modern diffusion system (conveniently also a probabilistic approach). In other words, an off-the-shelf baseline network that does not explicitly reason about structure can still be leveraged within our broader framework to learn representations that do.

This also allows us to take further advantage of pretraining. Many of the tasks we investigated here were relatively data-sparse. Therefore it makes sense to try and finetune existing systems on them, rather than trying to learn things like fluent language generation from scratch with the little data we do have. For example, our alt-text generation model in Chapter 7 makes use of a pretrained language model decoder, which we can finetune to accept image features as input, but don't need to teach how to output fluent sentences. This approach also makes it easy to update such a system as more powerful language models are developed; we can simply swap out the various modules for newer ones as they are developed. While there is presently much anxiety within the field about LLMs replacing all manner of task-specific models, we argue here that their capacity can simply be built on top of in ways that give them stronger inductive biases and controllability. Rather than attempting to compete directly with them using "old school" methods and less data, we might get the best of both worlds by building structured systems in ways that have points of attachment for pretrained networks. When we can do this, it allows us to effectively lower bound our performance with whatever out-of-the-box capabilities they might already have.

On a more philosophical note, when it comes to what tasks we choose to pursue, we believe that there is now and will continue to always be space for work that is not just technically impressive but also beneficial to society. While it is shocking and even inspiring how rapidly text-to-anything systems have progressed in recent years, there are arguably more important use cases for multimodal technology than simply as a replacement for the labor of human creatives. Methods that reason about data produced by humans have real world applications, as shown here, that go beyond the unestablished usefulness of systems that can reproduce it but do not do so in a way that lends greater insight or assistance to the generative process.

In summary, we present here approaches for a variety of NLP tasks that improve over previous systems by way of incorporating additional signal in the form of structure and context within our corpora, both at the type and token level. This allows us to learn more sophisticated contextualized representations both for generative and labeling purposes. In an era where large-scale multimodal systems are increasingly becoming not only desirable but also achievable, we believe that this is an ever important framing to maintain as the field continues to rapidly move forward.

Bibliography

Andrea Agostinelli, Timo I. Denk, Zalán Borsos, Jesse Engel, Mauro Verzetti, Antoine Caillon, Qingqing Huang, Aren Jansen, Adam Roberts, Marco Tagliasacchi, Matthew Sharifi, Neil Zeghidour, and Christian Havnø Frank. 2023. Musiclm: Generating music from text. *ArXiv*, abs/2301.11325. 8.1, 8.5

Nasir Ahmed, T_ Natarajan, and Kamisetty R Rao. 1974. Discrete cosine transform. *IEEE Transactions on Computers*. 2.4.2

Jean-Baptiste Alayrac, Jeff Donahue, Pauline Luc, Antoine Miech, Iain Barr, Yana Hasson, Karel Lenc, Arthur Mensch, Katie Millican, Malcolm Reynolds, Roman Ring, Eliza Rutherford, Serkan Cabi, Tengda Han, Zhitao Gong, Sina Samangooei, Marianne Monteiro, Jacob Menick, Sebastian Borgeaud, Andrew Brock, Aida Nematzadeh, Sahand Sharifzadeh, Mikolaj Binkowski, Ricardo Barreira, Oriol Vinyals, Andrew Zisserman, and Karen Simonyan. 2022. Flamingo: a visual language model for few-shot learning. 7.2

Paweł Aleksandrowicz. 2020. Can subtitles for the deaf and hard-of-hearing convey the emotions of film music? a reception study. *Perspectives*, 28(1):58–72. 8.1, 8.2

Alexa. 2018. Reddit.com traffic, demographics and competitors. `https://www.alexa.com/siteinfo/reddit.com`. Accessed: 2018-02-22. 5.2.1

Galen Andrew, Raman Arora, Jeff Bilmes, and Karen Livescu. 2013. Deep canonical correlation analysis. In *International conference on machine learning*, pages 1247–1255. PMLR. 1.4

Stanislaw Antol, Aishwarya Agrawal, Jiasen Lu, Margaret Mitchell, Dhruv Batra, C Lawrence Zitnick, and Devi Parikh. 2015. Vqa: Visual question answering. In *Proceedings of the IEEE international conference on computer vision*, pages 2425–2433. 1.4

Anas Awadalla, Irena Gao, Joshua Gardner, Jack Hessel, Yusuf Hanafy, Wanrong Zhu, Kalyani Marathe, Yonatan Bitton, Samir Gadre, Jenia Jitsev, Simon Kornblith, Pang Wei Koh, Gabriel Ilharco, Mitchell Wortsman, and Ludwig Schmidt. 2023. Openflamingo. 7.2

Samaneh Azadi, Matthew Fisher, Vladimir G Kim, Zhaowen Wang, Eli Shechtman, and Trevor Darrell. 2018. Multi-content GAN for few-shot font style transfer. *CVPR*. 2.2, 2.3, 2.6.1, 2.6.2, 3.1, 3.2, 3.4, 3.6.1, 3.6.5

Diamond Goldin Barbara, Ugenio, B. Justin, and Adgerow. 2021. Expected reciprocal rank for evaluating musical fingering advice. *Sound and Music Computing Conference*. 6.1, 6.2

Alexander TJ Barron, Jenny Huang, Rebecca L Spang, and Simon DeDeo. 2017. Individuals, institutions, and innovation in the debates of the french revolution. *arXiv preprint arXiv:1710.06867*. 5.1

Jonathan T. Barron. 2019. A general and adaptive robust loss function. *CVPR*. 2.4.2, 3.4

Anthony J Bell and Terrence J Sejnowski. 1997. Edges are the'independent components' of natural scenes. *NeurIPS*. 3.4

Aspen Belle., Vanessa Goh., Akshay Kumar., Richard Pranjatno., Pui Man Yip., Umayangani Wickramaratne., and Humphrey O. Obie. 2022. Alt-texify: A pipeline to generate alt-text from svg visualizations. In *Proceedings of the 17th International Conference on Evaluation of Novel Approaches to Software Engineering - ENASE,*, pages 275–281. INSTICC, SciTePress. 7.2

Emmett Leslie Bennett Jr. 1947. *The Minoan linear script from Pylos*. University of Cincinnati. 4.1

Taylor Berg-Kirkpatrick, Greg Durrett, and Dan Klein. 2013. Unsupervised transcription of historical documents. *ACL*. 1.4, 2.1

Taylor Berg-Kirkpatrick and Dan Klein. 2014. Improved typesetting models for historical OCR. *ACL*. 2.1

Raffaella Bernardi, Ruket Cakici, Desmond Elliott, Aykut Erdem, Erkut Erdem, Nazli Ikizler-Cinbis, Frank Keller, Adrian Muscat, and Barbara Plank. 2016. Automatic description generation from images: A survey of models, datasets, and evaluation measures. *Journal of Artificial Intelligence Research*, 55:409–442. 7.2

Sebastian Borgeaud, Arthur Mensch, Jordan Hoffmann, Trevor Cai, Eliza Rutherford, Katie Millican, George van den Driessche, Jean-Baptiste Lespiau, Bogdan Damoc, Aidan Clark, Diego de Las Casas, Aurelia Guy, Jacob Menick, Roman Ring, T. W. Hennigan, Saffron Huang, Lorenzo Maggiore, Chris Jones, Albin Cassirer, Andy Brock, Michela Paganini, Geoffrey Irving, Oriol Vinyals, Simon Osindero, Karen Simonyan, Jack W. Rae, Erich Elsen, and L. Sifre. 2021. Improving language models by retrieving from trillions of tokens. In *International Conference on Machine Learning*. 8.2

Marius Bulacu and Lambert Schomaker. 2007. Text-independent writer identification and verification using textural and allographic features. *IEEE transactions on pattern analysis and machine intelligence*, 29(4):701–717. 4.2

Neill DF Campbell and Jan Kautz. 2014. Learning a manifold of fonts. *ACM TOG*. 2.2, 3.2

Shaosheng Cao, Wei Lu, Jun Zhou, and Xiaolong Li. 2018. cw2vec: Learning chinese word embeddings with stroke n-gram information. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32. 3.2

Eshwar Chandrasekharan, Umashanthi Pavalanathan, Anirudh Srinivasan, Adam Glynn, Jacob Eisenstein, and Eric Gilbert. 2017. You can't stay here: The efficacy of reddit's 2015 ban examined through hate speech. *Proceedings of the ACM on Human-Computer Interaction*, 1(CSCW):31. 5.6.2

Tong Che, Yanran Li, Athul Paul Jacob, Yoshua Bengio, and Wenjie Li. 2016. Mode regularized generative adversarial networks. *arXiv preprint arXiv:1612.02136*. 2.7.1

Dongdong Chen, Lu Yuan, Jing Liao, Nenghai Yu, and Gang Hua. 2017. Stylebank: An explicit representation for neural image style transfer. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1897–1906. 3.2

Ke Chen, Xingjian Du, Bilei Zhu, Zejun Ma, Taylor Berg-Kirkpatrick, and Shlomo Dubnov. 2022. Hts-at: A hierarchical token-semantic audio transformer for sound classification and detection. pages 646–650. IEEE. 8.3.1

Ke Chen*, Yusong Wu*, Haohe Liu*, Marianna Nezhurina, Taylor Berg-Kirkpatrick, and Shlomo Dubnov. 2024. Musicldm: Enhancing novelty in text-to-music generation using beat-synchronous mixup strategies. In *IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP*. 8.5, 8.6.2

Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. 2020. A simple framework for contrastive learning of visual representations. In *International conference on machine learning*, pages 1597–1607. PMLR. 1.1

Domenico Chiarella, Justin Yarbrough, and Christopher A-L Jackson. 2020. Using alt text to make science twitter more accessible for people with visual impairments. *Nature Communications*, 11. 7.2

Sanjana Shivani Chintalapati, Jonathan Bragg, and Lucy Lu Wang. 2022. A dataset of alt texts from hci publications: Analyses and uses towards producing more descriptive alt texts of data visualizations in scientific papers. *Proceedings of the 24th International ACM SIGACCESS Conference on Computers and Accessibility*. 7.2

Keunwoo Choi, György Fazekas, and Mark B. Sandler. 2016. Towards music captioning: Generating music playlist descriptions. *ArXiv*, abs/1608.04868. 8.2

Vincent Christlein, Anguelos Nicolaou, Mathias Seuret, Dominique Stutzmann, and Andreas Maier. 2019. Icdar 2019 competition on image retrieval for historical handwritten documents.

In *2019 International Conference on Document Analysis and Recognition (ICDAR)*, pages 1505–1509. IEEE. 4.2

Eric Clarke, Richard Parncutt, Matti Raekallio, and John A. Sloboda. 1997. Talking fingers: An interview study of pianists' views on fingering. *Musicae Scientiae*, 1:107 – 87. 6.1, 6.4.1

Albert Cohen, Ingrid Daubechies, and J-C Feauveau. 1992. Biorthogonal bases of compactly supported wavelets. *Communications on pure and applied mathematics*, 45. 3.4

Michael Davidson. 2022. 7. a captioned life. In *Distressing Language*, pages 157–182. New York University Press. 8.1

Zihao Deng, Yi Ma, Yudong Liu, Rongchen Guo, Ge Zhang, Wenhu Chen, Wenhao Huang, and Emmanouil Benetos. 2023. Musilingo: Bridging music and text with pre-trained language models for music captioning and query response. *ArXiv*, abs/2309.08730. 8.2, 8.6.1

Michael Denkowski and Alon Lavie. 2014. Meteor universal: Language specific translation evaluation for any target language. In *Proceedings of the ninth workshop on statistical machine translation*, pages 376–380. 7.5.3, 8.6.2

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics. 1.2

Laurent Dinh, David Krueger, and Yoshua Bengio. 2014. Nice: Non-linear independent components estimation. *arXiv preprint arXiv:1410.8516*. 3.4

Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. 2016. Density estimation using real nvp. *arXiv preprint arXiv:1605.08803*. 3.4

Chawki Djeddi, Somaya Al-Maadeed, Imran Siddiqi, Gattal Abdeljalil, Sheng He, and Younes Akbari. 2018. Icfhr 2018 competition on multi-script writer identification. In *2018 16th International Conference on Frontiers in Handwriting Recognition (ICFHR)*, pages 506–510. IEEE. 4.2

Seungheon Doh, Keunwoo Choi, Jongpil Lee, and Juhan Nam. 2023. Lp-musiccaps: Llm-based pseudo music captioning. *ArXiv*, abs/2307.16372. 8.2, 8.7.3

Seungheon Doh, Junwon Lee, and Juhan Nam. 2021. Music playlist title generation: A machine-translation approach. In *Proceedings of the 2nd Workshop on NLP for Music and Spoken Audio (NLP4MusA)*, pages 27–31, Online. Association for Computational Linguistics. 8.2

Danny Driess, Fei Xia, Mehdi S. M. Sajjadi, Corey Lynch, Aakanksha Chowdhery, Brian Ichter, Ayzaan Wahid, Jonathan Tompson, Quan Vuong, Tianhe Yu, Wenlong Huang, Yevgen Chebotar, Pierre Sermanet, Daniel Duckworth, Sergey Levine, Vincent Vanhoucke, Karol Hausman, Marc Toussaint, Klaus Greff, Andy Zeng, Igor Mordatch, and Pete Florence. 2023. Palm-e: An embodied multimodal language model. In *arXiv preprint arXiv:2303.03378*. 1.4

Jan Driessen. 2000. *The scribes of the Room of the Chariot Tablets at Knossos: interdisciplinary approach to the study of a Linear B deposit*. 4.1

Lan Du, Wray Buntine, Huidong Jin, and Changyou Chen. 2012. Sequential latent dirichlet allocation. *Knowledge and information systems*, 31(3):475–503. 5.1

Vincent Dumoulin and Francesco Visin. 2016. A guide to convolution arithmetic for deep learning. *arXiv preprint arXiv:1603.07285*. 2.1

Jason Eisner. 2011. High-level explanation of variational inference. `https://www.cs.jhu.edu/~jason/tutorials/variational.html`. Accessed: 2018-02-22. 5.3

Ramy Eskander, Peter Martigny, and Shubhanshu Mishra. 2020. Multilingual named entity recognition in tweets using wikidata. In *The fourth annual WeCNLP (West Coast NLP) Summit (WeCNLP), virtual*. Zenodo. 7.4.3

Hao Fang, Saurabh Gupta, Forrest Iandola, Rupesh K Srivastava, Li Deng, Piotr Dollár, Jianfeng Gao, Xiaodong He, Margaret Mitchell, John C Platt, et al. 2015. From captions to visual concepts and back. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1473–1482. 7.2

Silvia Ferrara. 2012. *Cypro-Minoan Inscriptions: Volume 1: Analysis*, volume 1. Oxford University Press. 4.1

David J. Field. 1987. Relations between the statistics of natural images and the response properties of cortical cells. *JOSA A*. 2.4.2, 3.4

Richard J Firth, Christina Skelton, et al. 2016. A study of the scribal hands of knossos based on phylogenetic methods and find-place analysis. pages 159–188. 4.1

Alan Foley and Beth A Ferri. 2012. Technology for people, not disabilities: ensuring access and inclusion. *Journal of Research in Special Educational Needs*, 12(4):192–200. 8.2

William T Freeman and Joshua B Tenenbaum. 1997. Learning bilinear models for two-factor problems in vision. In *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 554–560. IEEE. 1.2, 2.1, 2.2, 3.2

Giovanni Gabbolini, Romain Hennequin, and Elena Epure. 2022. Data-efficient playlist captioning with musical and linguistic knowledge. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 11401–11415, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics. 8.2

Yue Gao, Yuan Guo, Zhouhui Lian, Yingmin Tang, and Jianguo Xiao. 2019. Artistic glyph image synthesis via one-stage few-shot learning. *ACM Transactions on Graphics (TOG)*, 38(6):1–12. 3.1, 3.2, 3.4, 3.6.5

Leon A Gatys, Alexander S Ecker, and Matthias Bethge. 2015. A neural algorithm of artistic style. *arXiv preprint arXiv:1508.06576*. 3.2

Leon A Gatys, Alexander S Ecker, and Matthias Bethge. 2016. Image style transfer using convolutional neural networks. *CVPR*. 2.1

Cole Gleason, Patrick Carrington, Cameron Cassidy, Meredith Ringel Morris, Kris M. Kitani, and Jeffrey P. Bigham. 2019. "it's almost like they're trying to hide it": How user-provided image descriptions have failed to make twitter accessible. *The World Wide Web Conference*. 7.1, 7.5.3, 7.6.3

Cole Gleason, Amy Pavel, Emma McCamey, Christina Low, Patrick Carrington, Kris M. Kitani, and Jeffrey P. Bigham. 2020. Twitter a11y: A browser extension to make twitter images accessible. *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*. 7.2

Yuan Gong, Alexander H Liu, Hongyin Luo, Leonid Karlinsky, and James Glass. 2023a. Joint audio and speech understanding. In *2023 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*. 8.2, 8.6.1, 8.7.3

Yuan Gong, Hongyin Luo, Alexander H Liu, Leonid Karlinsky, and James Glass. 2023b. Listen, think, and understand. *arXiv preprint arXiv:2305.10790*. 8.2, 8.6.1, 8.7.3

Google. Google fonts. `http://fonts.google.com`. Accessed: 2020-6-4. 3.1

Thomas L Griffiths, Michael I Jordan, Joshua B Tenenbaum, and David M Blei. 2004. Hierarchical topic models and the nested chinese restaurant process. In *Advances in neural information processing systems*, pages 17–24. 5.7

Albert Gu and Tri Dao. 2023. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*. 1.2

Gabriel Lima Guimaraes, Benjamín Sánchez-Lengeling, Pedro Luis Cunha Farias, and Alán Aspuru-Guzik. 2017. Objective-reinforced generative adversarial networks (organ) for sequence generation models. *ArXiv*, abs/1705.10843. 6.2

David Ha, Andrew Dai, and Quoc V Le. 2016. Hypernetworks. *arXiv preprint arXiv:1609.09106*. 3.3.1

David Hall and Dan Klein. 2010. Finding cognate groups using phylogenies. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 1030–1039, Uppsala, Sweden. Association for Computational Linguistics. 1.2

David Hall and Dan Klein. 2011. Large-scale cognate recovery. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 344–354, Edinburgh, Scotland, UK. Association for Computational Linguistics. 1.2

Margot Hanley, Solon Barocas, Karen Levy, Shiri Azenkot, and Helen Nissenbaum. 2021. Computer vision and conflicting values: Describing people with automated alt text. In *Proceedings of the 2021 AAAI/ACM Conference on AI, Ethics, and Society*, AIES '21, page 543–554, New York, NY, USA. Association for Computing Machinery. 7.2

Curtis Hawthorne, Ian Simon, Adam Roberts, Neil Zeghidour, Josh Gardner, Ethan Manilow, and Jesse Engel. 2022. Multi-instrument music synthesis with spectrogram diffusion. *arXiv preprint arXiv:2206.05408*. 1.1

Zihao He, Weituo Hao, Weiyi Lu, Changyou Chen, Kristina Lerman, and Xuchen Song. 2022a. Alcap: Alignment-augmented music captioner. 8.1, 8.2

Zihao He, Weituo Hao, and Xuchen Song. 2022b. Recap: Retrieval augmented music captioner. *ArXiv*, abs/2212.10901. 8.2

Brienna Herold, James Waller, and Raja Kushalnagar. 2022. Applying the stereotype content model to assess disability bias in popular pre-trained NLP models underlying AI-based assistive technologies. In *Ninth Workshop on Speech and Language Processing for Assistive Technologies (SLPAT-2022)*, pages 58–65, Dublin, Ireland. Association for Computational Linguistics. 7.2

Christopher Hidey and Kathleen McKeown. 2018. Persuasive influence detection: The role of argument sequencing. In *Association for the Advancement of Artificial Intelligence*. 5.7

Jonathan Ho, Ajay Jain, and Pieter Abbeel. 2020. Denoising diffusion probabilistic models. *Advances in Neural Information Processing Systems*, 33:6840–6851. 1.1, 1.4

M. Hoffman, D. Blei, and F. Bach. 2010. Online learning for latent dirichlet allocation. *Neural Information Processing Systems*. 5.3.4

Douglas R Hofstadter. 1983. Metamagical themas. *Scientific American*, 248(5):16–E18. 2.2

Douglas R Hofstadter. 1995. *Fluid concepts and creative analogies: Computer models of the fundamental mechanisms of thought.* Basic books. 2.2

Zhiting Hu, Zichao Yang, Xiaodan Liang, Ruslan Salakhutdinov, and Eric P Xing. 2017. Controllable text generation. *arXiv preprint arXiv:1703.00955*, 7. 2.1

Jinggang Huang and David Mumford. 1999. Statistics of natural images and models. *CVPR*. 2.4.2

Qingqing Huang, Aren Jansen, Joonseok Lee, Ravi Ganti, Judith Yue Li, and Daniel P. W. Ellis.

2022. Mulan: A joint embedding of music audio and natural language. In *International Society for Music Information Retrieval Conference*. 8.2

Qingqing Huang, Daniel S. Park, Tao Wang, Timo I. Denk, Andy Ly, Nanxin Chen, Zhengdong Zhang, Zhishuai Zhang, Jiahui Yu, Christian Havnø Frank, Jesse Engel, Quoc V. Le, William Chan, and Weixiang Han. 2023. Noise2music: Text-conditioned music generation with diffusion models. *ArXiv*, abs/2302.03917. 8.2

Natasha Jaques, Shixiang Gu, Richard E. Turner, and Douglas Eck. 2016. Generating music by fine-tuning recurrent neural networks with reinforcement learning. In *Deep Reinforcement Learning Workshop, NIPS*. 6.2

Justin Johnson, Alexandre Alahi, and Li Fei-Fei. 2016. Perceptual losses for real-time style transfer and super-resolution. In *European conference on computer vision*, pages 694–711. Springer. 3.2

Ian T. Jolliffe and Jorge Cadima. 2016. Principal component analysis: a review and recent developments. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 374. 1.1

Hadi Kazemi, Seyed Mehdi Iranmanesh, and Nasser Nasrabadi. 2019. Style and content disentanglement in generative adversarial networks. In *2019 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 848–856. IEEE. 3.2

Urvashi Khandelwal, Omer Levy, Dan Jurafsky, Luke Zettlemoyer, and Mike Lewis. 2019. Generalization through memorization: Nearest neighbor language models. *ArXiv*, abs/1911.00172. 8.2

Chloé Kiddon, Luke Zettlemoyer, and Yejin Choi. 2016. Globally coherent text generation with neural checklist models. In *EMNLP*. 6.2

Christine Sun Kim. 2020. [closer captions]. `https://www.youtube.com/watch?v=tfe479qL8hg`. 8.1

Haven Kim, Seungheon Doh, Junwon Lee, and Juhan Nam. 2023. Music playlist title generation using artist information. *ArXiv*, abs/2301.08145. 8.2

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. *3rd International Conference for Learning Representations*. 2.6.3, 3.6.3, 4.5.4, 5.4.3, 6.5.1, 7.5.2, 8.4

Diederik P Kingma and Max Welling. 2014. Auto-encoding variational bayes. *ICLR*. 1.1, 2.5, 3.5, 4.3

Elisa Kreiss, Cynthia Bennett, Shayan Hooshmand, Eric Zelikman, Meredith Ringel Morris, and Christopher Potts. 2022a. Context matters for image descriptions for accessibility: Challenges for referenceless evaluation metrics. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 4685–4697, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics. 7.1, 7.5.4

Elisa Kreiss, Fei Fang, Noah Goodman, and Christopher Potts. 2022b. Concadia: Towards image-based text generation with a purpose. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 4667–4684, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics. 7.2

Zhihuan Kuang, Shi Zong, Jianbing Zhang, Jiajun Chen, and Hongfu Liu. 2022. Music-to-text synaesthesia: Generating descriptive text from music recordings. *ArXiv*, abs/2210.00434. 8.2

Katherine Lee, Daphne Ippolito, Andrew Nystrom, Chiyuan Zhang, Douglas Eck, Chris Callison-Burch, and Nicholas Carlini. 2022. Deduplicating training data makes language models better. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 8424–8445, Dublin, Ireland. Association for Computational Linguistics. 8.5

Minhee Lee, Seungheon Doh, and Dasaem Jeong. 2023. Annotator subjectivity in the musiccaps dataset. In *HCMIR@ISMIR*. 8.5

Patrick Lewis, Ethan Perez, Aleksandara Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Kuttler, Mike Lewis, Wen tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. *ArXiv*, abs/2005.11401. 8.2

Junnan Li, Dongxu Li, Silvio Savarese, and Steven Hoi. 2023. Blip-2: Bootstrapping language-image pre-training with frozen image encoders and large language models. *arXiv preprint arXiv:2301.12597*. 7.1, 7.2, 7.5.1

Junnan Li, Dongxu Li, Caiming Xiong, and Steven Hoi. 2022. Blip: Bootstrapping language-image pre-training for unified vision-language understanding and generation. In *International Conference on Machine Learning*, pages 12888–12900. PMLR. 7.2

Yang Li, Gang Li, Luheng He, Jingjie Zheng, Hong Li, and Zhiwei Guan. 2020. Widget captioning: Generating natural language description for mobile user interface elements. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 5495–5510, Online. Association for Computational Linguistics. 7.2

Zhouhui Lian, Bo Zhao, Xudong Chen, and Jianguo Xiao. 2018. Easyfont: a style learning-based system to easily build your large-scale handwriting fonts. *ACM Transactions on Graphics (TOG)*, 38(1):1–18. 3.2

Chin-Yew Lin and Franz Josef Och. 2004. Automatic evaluation of machine translation quality using longest common subsequence and skip-bigram statistics. In *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics (ACL-04)*, pages 605–612. 7.5.3, 8.6.2

Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. 2014. Microsoft coco: Common objects in context. In *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13*, pages 740–755. Springer. 7.2, 8.6.2

Yan Liu, Alexandru Niculescu-Mizil, and Wojciech Gryc. 2009. Topic-link lda: joint models of topic and author community. In *proceedings of the 26th annual international conference on machine learning*, pages 665–672. ACM. 5.7

Stuart Lloyd. 1982. Least squares quantization in pcm. *IEEE Transactions on Information Theory*. 2.7.3

Raphael Gontijo Lopes, David Ha, Douglas Eck, and Jonathon Shlens. 2019. A learned representation for scalable vector graphics. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 7930–7939. 3.1, 3.2

María J Lucía, Pablo Revuelta, Álvaro García, Belén Ruiz, Ricardo Vergaz, Víctor Cerdán, and Tomás Ortiz. 2020. Vibrotactile captioning of musical effects in audio-visual media as an alternative for deaf and hard of hearing people: an eeg study. *IEEE Access*, 8:190873–190881. 8.2

Laurens van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605. 2.7.3, 3.9.2

Ilaria Manco, Emmanouil Benetos, Elio Quinton, and György Fazekas. 2021. Muscaps: Generating captions for music audio. *2021 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. 8.1, 8.2, 8.5

Ilaria Manco, Emmanouil Benetos, Elio Quinton, and György Fazekas. 2022. Contrastive audio-language learning for music. In *International Society for Music Information Retrieval Conference*. 8.2

Andrew McCallum, Dayne Freitag, and Fernando C Pereira. 2000. Maximum entropy markov models for information extraction and segmentation. In *ICML*. 6.2, 6.3.2

Daniel McKee, Justin Salamon, Josef Sivic, and Bryan Russell. 2023. Language-guided music recommendation for video via prompt analogies. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14784–14793. 8.2

Xinhao Mei, Xubo Liu, MarkD . Plumbley, and Wenwu Wang. 2022. Automated audio captioning: an overview of recent progress and new challenges. *EURASIP Journal on Audio, Speech,* *and Music Processing*, 2022:1–18. 8.2

José L Melena and Richard J Firth. 2019. *The Knossos Tablets*. INSTAP Academic Press (Institute for Aegean Prehistory). 4.5.1

Tamara Melmer, Seyed Ali Amirshahi, Michael Koch, Joachim Denzler, and Christoph Redies. 2013. From regular text to artistic writing and artworks: Fourier statistics of images with low and high aesthetic appeal. *Frontiers in Human Neuroscience*. 2.4.2

Meta. 2021. [link]. 7.2

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*. 1.1

Shubhanshu Mishra, Sijun He, and Luca Belli. 2020. Assessing demographic bias in named entity recognition. In *Proceedings of the AKBC Workshop on Bias in Automatic Knowledge Graph Construction, 2020*. arXiv. 7.4.3

Ron Mokady, Amir Hertz, and Amit H Bermano. 2021. Clipcap: Clip prefix for image captioning. *arXiv preprint arXiv:2111.09734*. 7.1, 7.2, 7.3.1, 7.5.2, 8.1, 8.2, 8.4

Amit Moryossef, Yanai Elazar, and Yoav Goldberg. 2019. At your fingertips: Automatic piano fingering detection. 6.1, 6.2, 6.3.1, 6.5.2, 6.1

Eita Nakamura, Nobutaka Ono, and Shigeki Sagayama. 2014. Merged-output hmm for piano fingering of both hands. In *ISMIR*, pages 531–536. 6.2

Eita Nakamura, Yasuyuki Saito, and Kazuyoshi Yoshii. 2020. Statistical learning and estimation of piano fingering. *Information Sciences*, 517:68–85. 6.1, 6.2, 6.3.1, 6.4.1, 6.5.2, **??**, 6.1, **??**, **??**, 6.2, 6.6.4, **??**

Ramesh M Nallapati, Amr Ahmed, Eric P Xing, and William W Cohen. 2008. Joint latent topic models for text and citations. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 542–550. ACM. 5.7

D. Blei A. Ng and M. Jordan. 2003. Latent dirichlet allocation. *Journal of Machine Learning Research*, 3:993–1022. 1.1, 5.1, 5.3, 5.3.4, 5.4.2, 5.7

Jean-Pierre Olivier. 1965. Les scribes de cnossos: essai de classement des archives d'un palais mycénien. 4.4.1

OpenAI. 2023. Gpt-4 technical report. 7.2

Nobuyuki Otsu. 1979. A threshold selection method from gray-level histograms. *IEEE transactions on systems, man, and cybernetics*, 9(1):62–66. 4.4.1

Thomas Palaima. 1988. The scribes of pylos. incunabula graeca. 4.4.1, 4.5.1

Thomas G Palaima. 2011. Scribes, scribal hands and palaeography. pages 33–136. Peeters

Louvain-la-Neuve. 4.1, 4.5.1

Ruth Palmer. 2008. How to begin? an introduction to linear b conventions and resources. pages 25–68. Peeters Louvain-la-Neuve. 4.1

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, pages 311–318. 7.5.3, 8.6.2

Richard Parncutt, John A. Sloboda, Eric Clarke, Matti Raekallio, and Peter Desain. 1997. An ergonomic model of keyboard fingering for melodic fragments. *Music Perception*, 14:341–382. 6.1, 6.2

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in PyTorch. In *NIPS Autodiff Workshop*. 2.6.3, 3.6.3, 4.5.4, 6.5.1, 7.5.2, 8.4

Romain Paulus, Caiming Xiong, and Richard Socher. 2017. A deep reinforced model for abstractive summarization. *arXiv preprint arXiv:1705.04304*. 6.2, 6.4.2, 7.3.2, 8.3.3

Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 2227–2237, New Orleans, Louisiana. Association for Computational Linguistics. 1.2

Huy Quoc Phan, Hongbo Fu, and Antoni B Chan. 2015. Flexyfont: Learning transferring rules for flexible typeface synthesis. In *Computer Graphics Forum*, volume 34, pages 245–256. Wiley Online Library. 2.2, 3.2

Martin Porter. 2001. Snowball: A language for stem-ming algorithms. `http://snowball.tartarus.org/texts`. Accessed: 2018-02-22. 5.4.1

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. 2021. Learning transferable visual models from natural language supervision. In *International Conference on Machine Learning*, pages 8748–8763. PMLR. 1.1, 1.4, 7.1, 7.2, 8.1, 8.2

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9. 7.1, 7.3.1, 8.1, 8.3.3, 8.4

Marc'Aurelio Ranzato, Sumit Chopra, Michael Auli, and Wojciech Zaremba. 2016. Sequence level training with recurrent neural networks. *CoRR*, abs/1511.06732. 6.2, 6.4.2

Reddit. 2015. Reddit in 2015. `https://redditblog.com/2015/12/31/reddit-in-2015/`.

Accessed: 2018-02-22. 5.2.1

Steven J. Rennie, Etienne Marcheret, Youssef Mroueh, Jerret Ross, and Vaibhava Goel. 2017. Self-critical sequence training for image captioning. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1179–1195. 6.2, 6.4.2

Pablo Revuelta, Tomás Ortiz, María J Lucía, Belén Ruiz, and José Manuel Sánchez-Pena. 2020. Limitations of standard accessible captioning of sounds and music for deaf and hard of hearing people: An eeg study. *Frontiers in integrative neuroscience*, 14:1. 8.1, 8.2

Danilo Rezende and Shakir Mohamed. 2015. Variational inference with normalizing flows. In *International Conference on Machine Learning*, pages 1530–1538. PMLR. 3.4

Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. 2014. Stochastic backpropagation and approximate inference in deep generative models. *ICML*. 2.5

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. 2015. U-Net: Convolutional networks for biomedical image segmentation. *MICCAI*. 3.3.1, 3.6.4

Ruslan Salakhutdinov and Geoffrey Hinton. 2009. Replicated softmax: An undirected topic model. *Advances in Neural Information Processing Systems*, 22:1607–1614. 5.1, 5.4.2, 5.7

Ruslan Salakhutdinov and Iain Murray. 2008. On the quantitative analysis of deep belief networks. *Proceedings of the 25th International Conference on Machine Learning*. 5.5.1

Piyush Sharma, Nan Ding, Sebastian Goodman, and Radu Soricut. 2018. Conceptual captions: A cleaned, hypernymed, image alt-text dataset for automatic image captioning. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2556–2565, Melbourne, Australia. Association for Computational Linguistics. 7.2, 7.6.3

Tianxiao Shen, Tao Lei, Regina Barzilay, and Tommi Jaakkola. 2017. Style transfer from non-parallel text by cross-alignment. *NIPS*. 2.2

Imran Siddiqi and Nicole Vincent. 2010. Text independent writer recognition using redundant writing patterns with contour-based orientation and curvature features. *Pattern Recognition*, 43(11):3853–3865. 4.2

Christina Skelton. 2008. Methods of using phylogenetic systematics to reconstruct the history of the linear b script. *Archaeometry*, 50(1):158–176. 1.2, 4.1, 4.2, 4.5, 4.5.2, 4.1, 4.6, 4.6

Christina Skelton. 2011. A look at early mycenaean textile administration in the pylos megaron tablets. *Kadmos*, 50(1):101–121. 4.1

Christina Skelton, Richard J Firth, et al. 2016. A study of the scribal hands of knossos based on phylogenetic methods and find-place analysis. part iii: Dating the knossos tablets using phylogenetic methods. pages 215–228. 4.1

John Sloboda, Eric Clarke, Richard Parncutt, and Matti Raekallio. 1998. Determinants of finger choice in piano sight-reading. *Journal of Experimental Psychology: Human Perception and Performance*, 24:185–203. 6.1, 6.4.1

Jake Snell, Karl Ridgeway, Renjie Liao, Brett D Roads, Michael C Mozer, and Richard S Zemel. 2017. Learning to generate images with perceptual similarity metrics. In *2017 IEEE International Conference on Image Processing (ICIP)*, pages 4277–4281. IEEE. 3.1, 3.6.5, 7.4.2

Vivek Kumar Rangarajan Sridhar. 2015. Unsupervised topic modeling for short texts using distributed representations of words. In *Proceedings of the 1st Workshop on Vector Space Modeling for Natural Language Processing*, pages 192–200. 5.5.4

Nikita Srivatsan, Jonathan T. Barron, Dan Klein, and Taylor Berg-Kirkpatrick. 2019. A deep factorization of style and structure in fonts. *EMNLP*. 1.5, 3.1, 3.3, 3.3, 3.3.1, 3.4, 3.4, 3.5, 3.6.1, 3.6.2, **??**, 4.2

Nikita Srivatsan and Taylor Berg-Kirkpatrick. 2022. Checklist models for improved output fluency in piano fingering prediction. In *International Society for Music Information Retrieval Conference*. 1.5

Nikita Srivatsan, Ke Chen, Shlomo Dubnov, and Taylor Berg-Kirkpatrick. 2024a. Retrieval guided music captioning via multimodal prefixes. *IJCAI Special Track on AI, the Arts, and Creativity*. 1.5

Nikita Srivatsan, Sofia Samaniego, Omar Florez, and Taylor Berg-Kirkpatrick. 2024b. Alt-text with context: Improving accessibility for images on twitter. *ICLR*. 1.5, 8.2

Nikita Srivatsan, Jason Vega, Christina Skelton, and Taylor Berg-Kirkpatrick. 2021a. Neural representation learning for scribal hands of linear b. In *Document Analysis and Recognition–ICDAR 2021 Workshops: Lausanne, Switzerland, September 5–10, 2021, Proceedings, Part II 16*, pages 325–338. Springer. 1.5

Nikita Srivatsan, Zachary Wojtowicz, and Taylor Berg-Kirkpatrick. 2018. Modeling online discourse with coupled distributed topics. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 4673–4682, Brussels, Belgium. Association for Computational Linguistics. 1.5

Nikita Srivatsan, Si Wu, Jonathan Barron, and Taylor Berg-Kirkpatrick. 2021b. Scalable font reconstruction with dual latent manifolds. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 3060–3072, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics. 1.5

Robin Strudel, Corentin Tallec, Florent Altché, Yilun Du, Yaroslav Ganin, Arthur Mensch, Will

Grathwohl, Nikolay Savinov, Sander Dieleman, Laurent Sifre, et al. 2022. Self-conditioned embedding diffusion for text generation. *arXiv preprint arXiv:2211.04236*. 1.1

Rapee Suveeranont and Takeo Igarashi. 2010. Example-based automatic font generation. In *International Symposium on Smart Graphics*, pages 127–138. Springer. 2.2, 3.2

Yichuan Tang, Ruslan Salakhutdinov, and Geoffrey Hinton. 2013. Tensor analyzers. In *International conference on machine learning*, pages 163–171. 2.1

Joshua B Tenenbaum and William T Freeman. 2000. Separating style and content with bilinear models. *Neural computation*, 12(6):1247–1283. 1.2, 2.1, 2.2, 3.2

Ilya O Tolstikhin, Sylvain Gelly, Olivier Bousquet, Carl-Johann Simon-Gabriel, and Bernhard Schölkopf. 2017. Adagan: Boosting generative models. In *Advances in Neural Information Processing Systems*, pages 5424–5433. 2.7.1

Yulia Tsvetkov, Manaal Faruqui, Wang Ling, Guillaume Lample, and Chris Dyer. 2015. Evaluation of word vector representations by subspace alignment. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 2049–2054. 4.5.2, 4.1

Dmitry Ulyanov, Vadim Lebedev, Andrea Vedaldi, and Victor S Lempitsky. 2016a. Texture networks: Feed-forward synthesis of textures and stylized images. In *ICML*, volume 1, page 4. 3.2

Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. 2016b. Instance normalization: The missing ingredient for fast stylization. *arXiv preprint arXiv:1607.08022*. 2.5.1, 3.6.4, 4.3.1

Unicode. Unicode® 13.0.0. `http://unicode.org/versions/Unicode13.0.0/`. Accessed: 2020-5-27. 3.1

Laurens Van Der Maaten. 2014. Accelerating t-sne using tree-based algorithms. *The Journal of Machine Learning Research*, 15(1):3221–3245. 3.9.2

M Alex O Vasilescu and Demetri Terzopoulos. 2002. Multilinear analysis of image ensembles: Tensorfaces. In *European Conference on Computer Vision*, pages 447–460. Springer. 2.1

Ramakrishna Vedantam, C Lawrence Zitnick, and Devi Parikh. 2015. Cider: Consensus-based image description evaluation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4566–4575. 7.5.3, 8.6.2

Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. 2008. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pages 1096–1103. 1.1

Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. 2015. Show and tell: A neural image caption generator. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3156–3164. 7.2

Xiaolong Wang and Abhinav Gupta. 2016. Generative image modeling using style and structure adversarial networks. In *European conference on computer vision*, pages 318–335. Springer. 3.2

Ronald J. Williams. 2004. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256. 6.2, 6.4.2

Shaomei Wu, Jeffrey Wieland, Omid Farivar, and Julie Schiller. 2017. Automatic alt-text: Computer-generated image descriptions for blind users on a social network service. *Proceedings of the 2017 ACM Conference on Computer Supported Cooperative Work and Social Computing*. 7.2

Yonghui Wu, Mike Schuster, Z. Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Lukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason R. Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Gregory S. Corrado, Macduff Hughes, and Jeffrey Dean. 2016. Google's neural machine translation system: Bridging the gap between human and machine translation. *ArXiv*, abs/1609.08144. 6.4.2

Yusong Wu, Ke Chen, Tianyu Zhang, Yuchen Hui, Taylor Berg-Kirkpatrick, and Shlomo Dubnov. 2023. Large-scale contrastive language-audio pretraining with feature fusion and keyword-to-caption augmentation. pages 1–5. IEEE. 8.1, 8.2, 8.3.1, 8.5, 8.6.2

Hong-Jian Xue, Xinyu Dai, Jianbing Zhang, Shujian Huang, and Jiajun Chen. 2017. Deep matrix factorization models for recommender systems. *IJCAI*. 2.1

Shuai Yang, Jiaying Liu, Wenjing Wang, and Zongming Guo. 2019. Tet-gan: Text effects transfer via stylization and destylization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 1238–1245. 3.2

Zichao Yang, Zhiting Hu, Ruslan Salakhutdinov, and Taylor Berg-Kirkpatrick. 2017. Improved variational autoencoders for text modeling using dilated convolutions. *ICML*. 2.1

Yuichiro Yonebayashi, H. Kameoka, and Shigeki Sagayama. 2007. Automatic decision of piano fingering based on a hidden markov models. In *IJCAI*. 6.2

Lantao Yu, Weinan Zhang, Jun Wang, and Yong Yu. 2017. Seqgan: Sequence generative adversarial nets with policy gradient. In *AAAI*. 6.2

Richard Zhang. 2019. Making convolutional networks shift-invariant again. In *International Conference on Machine Learning*, pages 7324–7334. PMLR. 3.6.4, 4.3.1

Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, et al. 2022. Opt: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*. 7.5.1

Xu-Yao Zhang, Guo-Sen Xie, Cheng-Lin Liu, and Yoshua Bengio. 2016. End-to-end online writer identification with recurrent neural network. *IEEE transactions on human-machine systems*, 47(2):285–292. 4.2

Yexun Zhang, Ya Zhang, and Wenbin Cai. 2018. Separating style and content for generalized style transfer. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8447–8455. 2.2, 3.1, 3.2, 3.6.2

Yexun Zhang, Ya Zhang, and Wenbin Cai. 2020. A unified framework for generalizable style transfer: Style and content separation. *IEEE Transactions on Image Processing*, 29:4085–4098. 2.2, 3.2

Yixiao Zhang, Akira Maezawa, Gus G. Xia, Kazuhiko Yamamoto, and Simon Dixon. 2023. Loop copilot: Conducting ai ensembles for music generation and iterative editing. *ArXiv*, abs/2310.12404. 8.1

Haoyue Zhao, Xin Guan, and Qiang Li. 2022. Estimation of playable piano fingering by pitch-difference fingering match model. *EURASIP Journal on Audio, Speech, and Music Processing*, 2022:1–13. 6.1, 6.2, 6.3.1

Wayne Xin Zhao, Jing Jiang, Jianshu Weng, Jing He, Ee-Peng Lim, Hongfei Yan, and Xiaoming Li. 2011. Comparing twitter and traditional media using topic models. In *European Conference on Information Retrieval*, pages 338–349. Springer. 5.1

Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. 2017. Unpaired image-to-image translation using cycle-consistent adversarial networks. *ICCV*. 2.1