Université de Montréal

Structured Prediction and Generative Modeling using Neural Networks

par Kyle Kastner

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

Mémoire présenté à la Faculté des arts et des sciences
en vue de l'obtention du grade de Maître ès sciences (M.Sc.)
en informatique

Août, 2016

Résumé

Cette thèse traite de l'usage des Réseaux de Neurones pour modélisation de données séquentielles. La façon dont l'information a été ordonnée et structurée est cruciale pour la plupart des données. Les mots qui composent ce paragraphe en constituent un exemple. D'autres données de ce type incluent les données audio, visuelles et génomiques. La Prédiction Structurée est l'un des domaines traitant de la modélisation de ces données. Nous allons aussi présenter la Modélisation Générative, qui consiste à générer des points similaires aux données sur lesquelles le modèle a été entraîné.

Dans le chapitre 1, nous utiliserons des données clients afin d'expliquer les concepts et les outils de l'Apprentissage Automatique, incluant les algorithmes standards d'apprentissage ainsi que les choix de fonction de coût et de procédure d'optimisation. Nous donnerons ensuite les composantes fondamentales d'un Réseau de Neurones. Enfin, nous introduirons des concepts plus complexes tels que le partage de paramètres, les Réseaux Convolutionnels et les Réseaux Récurrents. Le reste du document, nous décrirons de plusieurs types de Réseaux de Neurones qui seront à la fois utiles pour la prédiction et la génération et leur application à des jeux de données audio, d'écriture manuelle et d'images

Le chapitre 2.2 présentera le Réseau Neuronal Récurrent Variationnel (VRNN pour *variational recurrent neural network*). Le VRNN a été développé dans le but de générer des échantillons semblables aux exemples de la base d'apprentissage. Nous présenterons des modèles entraînées de manière non-supervisée afin de générer du texte manuscrites, des effets sonores et de la parole. Non seulement ces modèles prouvent leur capacité à apprendre les caractéristiques de chaque type de données mais établissent aussi un standard en terme de performance.

Dans le chapitre 3 sera présenté ReNet, un modèle récemment développé. ReNet utilise les sorties structurées d'un Réseau Neuronal Récurrent pour classifier des objets. Ce modèle atteint des performances compétitives sur plusieurs tâches de reconnaissance d'images, tout en utilisant une architecture conçue dès le départ pour de la Prédiction Structurée. Dans ce cas-ci, les résultats du modèle sont utilisés simplement pour de la classification mais des travaux suivants (non-inclus ici) ont utilisé ce modèle pour de la Prédiction Structurée.

Enfin, au Chapitre 4 nous présentons les résultats récents non-publiés en génération acoustique. Dans un premier temps, nous fournissons les concepts musicaux et représentations numériques fondamentaux à la compréhension de notre approche et introduisons ensuite une base de référence et de nouveaux résultats de recherche

avec notre modèle, RNN-MADE. Ensuite, nous introduirons le concept de synthèse vocale brute et discuterons de notre recherche en génération. Dans notre dernier Chapitre, nous présenterons enfin un résumé des résultats et proposerons de nouvelles pistes de recherche.

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

Summary

In this thesis we utilize neural networks to effectively model data with sequential structure. There are many forms of data for which both the order and the structure of the information is ncredibly important. The words in this paragraph are one example of this type of data. Other examples include audio, images, and genomes. The work to effectively model this type of ordered data falls within the field of structured prediction. We also present generative models, which attempt to generate data that appears similar to the data which the model was trained on.

In Chapter 1, we provide an introduction to data and machine learning. First, we motivate the need for machine learning by describing an expert system built on a customer database. This leads to a discussion of common algorithms, losses, and optimization choices in machine learning. We then progress to describe the basic building blocks of neural networks. Finally, we add complexity to the models, discussing parameter sharing and convolutional and recurrent layers. In the remainder of the document, we discuss several types of neural networks which find common use in both prediction and generative modeling and present examples of their use with audio, handwriting, and images datasets. In Chapter 2.2, we introduce a variational recurrent neural network (VRNN). Our VRNN is developed with to generate new sequential samples that resemble the dataset that is was trained on. We present models that learned in an unsupervised manner how to generate handwriting, sound effects, and human speech setting benchmarks in performance.

Chapter 3 shows a recently developed model called ReNet. In ReNet, intermediate structured outputs from recurrent neural networks are used for object classification. This model shows competitive performance on a number of image recognition tasks, while using an architecture designed to handle structured prediction. In this case, the final model output is only used for simple classification, but follow-up work has expanded to full structured prediction.

Lastly, in Chapter 4 we present recent unpublished experiments in sequential audio generation. First we provide background in musical concepts and digital representation which are fundamental to understanding our approach and then introduce a baseline and new research results using our model, RNN-MADE. Next we introduce the concept of raw speech synthesis and discuss our investigation into generation. In our final chapter, we present a brief summary of results and postulate future research directions.

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

Contents

List of Figures

List of Tables

List of Abbreviations

AE	Auto-Encoder
CNN	Convolutional Neural Network
GD	Gradient Descent
GRU	Gated Recurrent Unit
GMM	Gaussian Mixture Model
HMM	Hidden Markov Model
KLD	Kullback-Liebler Divergence
LSTM	Long-Short Term Memory
MLE	Maximum Likelihood Estimation
MLP	Multi-Layer Perceptron
MSE	Mean Squired Error
NADE	Neural Autoregressive Density Estimator
NLL	Negative Log-Likelihood
PMF	Probability Mass Function
PDF	Probability Density Function
RNN	Recurrent Neural Network
SGD	Stochastic Gradient Descent
VAE	Variational Auto-Encoder
XE	Cross Entropy

Acknowledgments

I am extremely grateful for an amazing set of family, friends, colleagues, and advisers without whom my work would not be possible.

Johanna Hansen, thank you for your continued love, support, and partnership. I would not be able to do any of this without you. I am blessed with an incredible immediate and extended family whose continued love, support, and strength has helped me every step of the way. Thanks to my mother, Sandra; father, Victor; and sister, Courtney.

Thank you to Dr. Roland Memisevic, Dr. Aaron Courville, and Dr. Yoshua Bengio for serving as outstanding advisers. Your direction, research mentorship, and guidance has allowed me to grow as a researcher and follow the path I am on today.

To my many friends, co-authors, co-workers, and colleagues (alphabetically):
Adam Roberts, Adriana Romero, Alex Lamb, Alexandre Abraham, Amjad Almahairi, Anna Huang, Arnaud Bergeron, Barron Bichon, Bart van Merriënboer, Ben Poole, Brandon Sammons, Çaglar Gülçehre, Casper Kaae Sønderby, César Laurent, Chris Smith, Christof Angermueller, Cinjon Resnick, Colin Raffel, Colin Vaz, Curtis "Fjord" Hawthorne, Daniel Trejo, Daniel Jiwoong Im, David Ha, David Krueger, David Warde-Farley, Dmitriy Serdyuk, Dzmitry Bahdanau, Elvis Dohmatob, Eduardo Gonzalez, Eugene Belilovsky, Fabian Pedregosa, Faruk Ahmed, Felix Hill, Francesco Visin, Fred Bertsch, Frédéric Bastien, Guillaume Alain, Harm de Vries, Ishaan Gulrajani, Ishmael Belghazi, Iulian Vlad Serban, Jacob Limmer, Jan Chorowski, Jesse Engel, João Felipe Santos, Jörg Bornschein, José Sotelo, Junyoung Chung, Kelvin Xu, Kratarth Goel, Kyunghyun Cho, Laurent Dinh, Li Yao, Luke Vilnis, Marcelo Grave, Mark Pillion, Mary Otto, Mathieu Germain, Matthew Coursen, Mehdi Mirza, Michael Eickenberg, Michael Talbert, Mohammad Pezeshski, Myriam Côté, Natasha Jaques, Nicolas Ballas, Negar Rostamzadeh, Orhan Firat, Pascal Lamblin, Patrick Sammons, Pierre-Luc Carrier, Philemon Brakel, Rick Parker, Sam Stavinoha, Sebastian Öberg, Shawn Tan, Simon Receveur, Sina Honari, Søren Sønderby, Soroush Mehri, Stanislas Lauly, Sungjin Ahn, Tim Cooijmans, Vincent Dumoulin, and Yann Dauphin; Thank you for everything you do.

Thank you to Mohammad Pezeshski for showing me how to use this thesis template by example and to Nicolas Chapados for providing the thesis template. Thank you

to Laurent Dinh for translating the summary portion of this thesis. Thank you also to Céline Begin for all her patience during the thesis submission process and for her organizational help during my studies.

In addition, thanks to the rest of my friends and coworkers in MILA, my previous workplaces, the scikit-learn team, and the internet at large. I am sure I missed some people I shouldn't have forgotten. Thanks to all of you.

To all my wonderful teachers Matt Popnoe, Rex Ewert, Richard Roper, Butch Crudginton, Dr. Donald Olson, Dr. Heather Galloway, Dr. Michael Casey, Dr. Stan McClellan, Dr. William Stapleton, Dr. Jim Lansford, and Dr. Sos Agaian. Thank you for so much education and inspiration.

To my friends and mentors from SwRI (some now elsewhere), I wouldn't be here without you. AF, SB, DC, DH, BM, AH, GM, BR, JJ, MG, RC, KB, DVR, DF, RH, KD, JS, TS, BB, AE, CZ, HO, BN, JP, JN along with many others.

I also want to express an extra thanks to my internship mentors: Gaël Varoquaux and Olivier Grisel at INRIA Parietal, Bhuvana Ramabhadran and Ewout van den Berg at IBM Research, and Douglas Eck and Mike Schuster at Google Brain.

To my friends, family, and mentors who are no longer with us, RIP and thank you for everything.

Finally, the work reported in this thesis would not have been possible without the financial support from: Ubisoft, Samsung, IBM, NSERC, Calcul Quebec, Compute Canada, the Canada Research Chairs and CIFAR.

1 Introduction

1.1 Overview

Machine learning is an important part of modern computer science, with applications across many industries including commerce, finance, logistics, agriculture, and education. There are many detailed references for deeply understanding machine learning such as Bishop (2006), Hastie et al. (2001), Murphy (2012), and Goodfellow et al. (2016). In this introductory chapter, we simply strive to give an overview of the core concepts and terminology necessary for understanding the basics of the later chapters. In this work we will cover some recent advances in the subfields of machine learning known as structured prediction and generative modeling. Structured prediction refers to using machine learning to predict structured elements such as vectors or sets, rather than single values. Generative modeling means that we desire to use our techniques to generate new outputs similar to the inputs. The combination of structured prediction can be used to generate things like audio, handwriting, images, and text. We approach these problems using *deep learning* techniques with *neural networks*. Structured prediction and generative modeling will be covered in more detail later, but first we will provide background on machine learning in general. Machine learning can be partly defined by exploring one of its component terms: *learning*.

What is learning?

One place to begin in our quest to understand *learning* is the dictionary definition. The Myriam-Webster dictionary defines learning as: *the activity or process of gaining knowledge or skill by studying, practicing, being taught, or experiencing something* (myr, 2016). The most important aspect of this definition is that studying, teaching, practicing, or experiencing all involve taking these actions over some information. For our purposes the information that we study is known as *data*. We

expect that the ability to make good *decisions* will improve with algorithmic study (through a learning algorithm) of a particular type data. In machine learning we attempt to make good decisions about data, by using learning algorithms to study the data in question.

What is data?

Data is a catch-all term used to describe any kind of information about an object, process, or concept. In the context of machine learning data is the description of what we want to learn about, for example: the purchase history of a customer, the pixels of an image, the text of a sentence, some recordings from election speeches, or any other domain-dependent information that could be useful Shannon (1948). A typical way of representing data in machine learning is as a collection of attributes, or *features*. These features describe some characteristics of the thing being measured. Let's envision a dataset from a department store as an example. The store manager wishes to learn more about his customers by investigating his historical records about their shopping habits. In the context of machine learning, each customer is a *sample* and the customer's shopping records are known as features. Features may include details about the customer such as the last item purchased, the total of their last bill, and the number of years they have been a customer. In this simple set of data, we can represent each customer with three numerical features: *item_id*, *prev_bill*, *n_years*.

Table 1.1 – Example customer data

	item_id	prev_bill	n_years
Customer A	11764	1000.00	0.6
Customer B	9718	243.16	5.12
Customer C	42	156.19	3.3

Table 1.1 shows a common format for data in a dataset. In this representation, the vertical dimension of the array is the *sample axis*, while the horizontal axis is the *feature axis*. In general, the feature vectors for each sample can be stacked to make a data matrix (often denoted simply as x), and later fed to any number of learning algorithms. There are many other ways to represent data which can

capture additional structure in data such as images, text, graphs, audio, or video that certain algorithms can use for better learning. Before discussing learning algorithms in detail, we must first describe what we wish to learn in the first place: *decisions*.

What are decisions?

In the preceding paragraphs, we introduced *decisions* as the basis which the act of studying, teaching, practicing, or experiencing was meant to improve. However, it is not necessary to have learning algorithms in order to make a decision. Let us construct an example using the department store data which was introduced in Table 1.1.

Table 1.2 – Extended customer data

	item_id	prev_bill	n_years	good_bad
Customer A	11764	1000.00	0.6	1
Customer B	9718	243.16	5.12	0
Customer C	42	156.19	3.3	?

Suppose in addition to the information in Table 1.1, we were also given additional information about each customer, namely whether each customer was "good" (0) or "bad" (1) for the business. We could add this information to the dataset, resulting in a new table, Table 1.2.

Notice in particular that Customer C has unknown information in the "good_bad" column, but it is believable that we might want to *predict* or *infer* that attribute for Customer C. One way to do this might be to find the employee who created the "good_bad" attributes for the other customers, and ask them to encode their logic for making their decision on "good" or "bad" into a procedure, or *function*.

This function (shortened to $f()$) should take the data we have now (x) for each customer, and try to output the value for "good_bad" (sometimes called the *labels* or *targets* typically denoted y), such that for existing customers with known values for "good_bad", $y_c = f(x_c)$ where $_c$ is shorthand to reference the features (x_c) and labels (y_c) for each customer. For Customer C, we have no true value for "good_bad", so we can only judge how well the function $f()$ is working by seeing if it

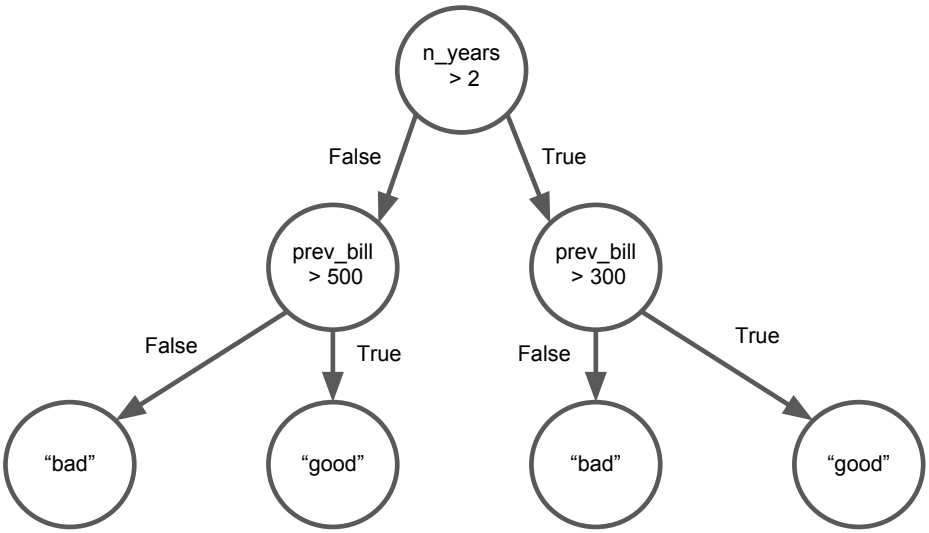


Figure 1.1 – Example decision process for deciding "good" or "bad" customers

is correct on Customers A and B. However if the function is working well, it is now possible to *predict* the "good_bad" attribute for Customer C or *any other customer* with the same feature information. If customers are added to the dataset later, and have all their features input this function could then predict the "good_bad" attribute and add it to the database. The logic for such a decision might be simple, as in Figure 1.1.

Motivations for machine learning

The logic for predicting "good" or "bad" customers is an example of something known as an expert system (Russel and Norvig, 2003). Variants of these expert systems, using a number of different algorithms, have been employed to make machine decisions since the dawn of computing. The solution shown in Figure 1.1 shows an algorithm called a decision tree with explicitly set rules. Decision trees in particular show up in a variety of contexts, and those employing and coding handcrafted decision trees are often unaware that they have already implemented a direct precursor to machine learning by using *machine decisions* indirectly based on data (after being filtered through the mind of one or many employees). The details of decision trees will not be covered in detail here, but Hastie et al. (2001)

is an excellent reference for this algorithm and various extensions.

The primary difference between this hypothetical system and a machine learning approach would be having the *machine* learn the decision thresholds directly, rather than requiring the programmer or another expert to specify the rule manually. Many large codebases in open source and enterprise have files of "magic code", filled with complex if-else statements and unknown or undocumented constants for making choices at each branch. These are real-world examples of "expert systems", and in many cases machine learning techniques can greatly improve performance. Machine learning models can often discover new or unknown relationships in the data which are not obvious to the human expert.

From a machine learning perspective, expert systems are not considered a learning system. The programmer(s) have input the entire logic explicitly into the decision process, and it is not clear how these rules were derived without explanations and expertise. Indeed, it is believable that the rules that work for Customers A and B might not work in every circumstance. More customer data and *labels*, more *features*, or both may be necessary to make rules which are more general. As information for customers is gathered, the rules which exist may be worse than a potential set of new rules, especially as more *features* and *labels* are obtained.

One way to continuously improve this system would be to dedicate one employee fully to creating and testing new rules. As customers and features are added to the data matrix, this employee could potentially write better rules. Eventually the number of customers and features might overwhelm a single employee, and a team would needed to do this job. Soon, a whole section of the company might be entirely focused on writing new rules for this predictive system, an expensive proposition (Brooks, 1995).

What if we could instead write higher level routines, so that the *machine* itself could *learn* basic rules from data? This could scale to much larger problems, without incurring a larger overhead beyond increased computer time. This is the basic premise (and promise) of machine learning.

1.2 Types of Learning

Using the descriptions from the previous section, we have loosely defined learning as the development of a function $f()$ which takes in data, and outputs desired information. This function must have some internal values which can be learned or modified, typically called *parameters* and denoted by θ (theta). In the previous example, the parameters of θ would be the thresholds for each "if" statement, but usually what parameters *do* is dependent on the algorithm contained inside $f()$. This means the function $f()$ is actually a function of both the inputs *and* the learned parameters θ, so it can also be written $f_\theta()$ or $f(a, \theta)$. Most machine learning references don't directly denote θ and instead leave it as a known assumption. There are also approaches which do not have parameters. Approaches without parameters are called (*non-parametric*) functions. For the moment, we will limit our discussion to *parametric* approaches. Machine learning attempts to solve many tasks, but generally we can categorize learning into two broad types: *supervised* learning and *unsupervised* learning.

Supervised learning is focused on learning a mapping between data and label. The label can be a numeric value, as in our department store example. The data can also be any number of human created or curated values such as pixels or words. We continue the previous notation, and denote general supervised tasks as attempting to learn $y_i = f(x_i)$ for each sample i with "supervision" values in the dataset. This function can then be used to label new data samples without a known label. Supervised learning is also called *predictive modeling* in some references.

Unsupervised learning, in contrast to supervised learning, is explicitly focused on learning more suitable features, sometimes called *representations*, without specific labels on the data. Unsupervised tasks often attempt to learn forms of compression, such as $\hat{x}_i = f(x_i)$. Specifically $\hat{x}_i$ is a reconstruction of x_i after being passed through an information bottleneck, so that the function must learn to *compress* information. This compression is important since the alternative is the trivial solution of $x_i = x_i$, which should be avoided. Learning to reconstruct a part of some data, given its surrounding information (sometimes called context) is another way to learn good representations for many types of data. Unsupervised learning is also called *generative modeling* in many settings, and specific approaches explicitly attempt to learn generative distributions of the data. We will cover more on these

approaches in later chapters.

Subgroups of Supervised and Unsupervised Learning

In addition to the broad categorizations of unsupervised and supervised learning, there are sub-groupings such as *classification*, *regression*, and *density estimation*. Specifically, classification and regression are different types of supervised learning, while the basic forms of density estimation are unsupervised tasks. Classification is used to denote tasks which involve categorizing data points into one of K possible buckets, or *classes*. Regression, in the context of machine learning, is a common task which is popular in the statistics literature. Regression algorithms attempt to predict a real value for each input x_i. Density estimation is a technique for fitting probability densities to data, to determine underlying structure. Clustering algorithms are generally categorized as a subset of density estimation. Bishop (2006) and Hastie et al. (2001) are both excellent references for listings of different techniques in these groups and subgroups of machine learning.

In the example case from the previous section, we wanted a function to predict the "good_bad" attribute for each customer, which is one of two values in the set {0, 1}. Because this task requires learning a function to map the data x to a human labeled attribute y, which is one of K possibilities (in this case $K = 2$), we broadly file this task under the heading of *supervised classification*.

1.3 A Formal Definition of Learning

One common mathematical framework for machine learning is to define the problem of discovering the optimal parameters (θ^*) as an optimization problem. The fundamental problem then becomes *minimizing* risk, where risk is the integration of some *loss function* between the true values y and some function $f(x, \theta)$, where the integration is with respect to the joint probability distribution of x and y. This framework was defined in Vapnik (1991), and serves as clear reading material covering much of the basis for machine learning as optimization.

$$R(\theta) = \int L(y, f(x, \theta)) dp(x, y) \tag{1.1}$$

Noting that the joint probability $p(x, y)$ can be rewritten using Bayes rule, we see that $p(x, y) = p(x|y)p(y) = p(y|x)p(x)$. This joint probability is unknown for most problems of interest, since we usually do not know the underlying data distributions $p(x)$ or $p(y)$. Our only available source of information is in the training set, yet we must approximate this integral somehow. Replacing the problem of risk minimization with *empirical risk minimization*, or minimizing the summed loss over the training set (sometimes called the empirical data distribution), is our best approximation to the integral in Equation 1.1.

$$ER(\theta) = \frac{1}{l}\sum_{i=1}^{l} L(y_i, f(x_i, \theta)) \tag{1.2}$$

Thus, the desired set of parameters θ^* is the set of θ that minimize the empirical risk $ER(\theta)$.

$$\theta^* = argmin_y \sum_{i=1}^{l} L(y_i, f(x_i, \theta)) \tag{1.3}$$

These general notions form much of the intuition behind machine learning as optimization, and can be extended in interesting ways to improve results on structured problems (Bahdanau et al., 2015). It is important to note that the parameters θ^* which are optimal on one data subset may not *generalize* to new subsets. Indeed much of the "art" in machine learning is understanding and minimizing differences in performance between the set used for loss minimization (the so called training set) and the application domain. This can partially be done through careful composition of losses, algorithms, regularization, and optimization techniques.

1.4 Losses

After discussing the empirical risk minimization framework, it is necessary to define actual functions to use for $L(y, f(x, y))$. In many cases we use the function $f(x, y)$ to model conditional probability distributions $p_\theta(y|x)$. This directly leads to costs which have a probabilistic interpretation when taking the maximum likelihood estimator (MLE) over whatever output distribution is assumed for $p_\theta(y|x)$ (Goodfellow et al., 2016).

In many common cases, the cost can be derived as the negative log-likelihood between the data and the current model distribution. Negative log-likelihood is also commonly known as cross-entropy, and the two terms are commonly interchanged in machine learning literature. In the standard problem setting data doesn't change during learning, so reducing the cross-entropy amounts to improving the likelihood of $p_\theta(y|x)$ with respect to the data. This is also equivalent to minimization of the Kullback-Liebler divergence (KLD) between the empirical data distribution and the model distribution, as demonstrated in Nowak (2009). Thus optimizing cross-entropy corresponds to reducing the gap between the empirical distribution and the *model*. The θ^* corresponding to the lowest cross-entropy is therefore a best estimate for the parameters $p^*_\theta(y|x)$, under the distribution assumptions of the cost. We will re-derive several common costs using this general principle.

1.4.1 Relationship between KLD and MLE

Beginning with two distributions $p(x)$ (the empirical distribution) and $q(x)$ (the model distribution), we first state the KLD (Nowak, 2009).

$$D_{KL}(p(x)||q(x)) = \int p(x) \log p(x) dx - \int p(x) \log q(x) dx \tag{1.4}$$

$$= \int p(x) \log \frac{p(x)}{q(x)} dx \tag{1.5}$$

$$= \mathbb{E}_{x\sim p}[\log \frac{p(x)}{q(x)}] \tag{1.6}$$

$$= \mathbb{E}_{x\sim p}[\log p(x) - \log q(x)] \tag{1.7}$$

Next, we state the basics of maximum likelihood estimation for set of l data

points.

$$\theta* = \arg\max_{\theta} \prod_{i=1}^{l} q(x_i) \tag{1.8}$$

$$= \arg\max_{\theta} \sum_{i=1}^{l} \log q(x_i) \tag{1.9}$$

$$= \arg\min_{\theta} \frac{1}{l} \sum_{i=1}^{l} -\log q(x_i) \tag{1.10}$$

$$\sim= \arg\min_{\theta} \mathbb{E}_{l\to\infty, x\sim p}[-\log q(x)] \tag{1.11}$$

The last approximate equality occurs due to the "law of large numbers", as l goes to ∞. Note that both the KLD and MLE have a term related to the expectation of the negative log probability of the model distribution ($q(x)$). The KLD between these two probability distributions is always positive, and only 0 if they exactly match. Going back to the original formulation for $D_{KL}(p(x)||q(x))$ we can also reformulate the terms in another way.

$$D_{KL}(p(x)||q(x)) = \int p(x) \log p(x) dx - \int p(x) \log q(x) dx \tag{1.12}$$

$$= -H(p(x)) + XE(p(x), q(x)) \tag{1.13}$$

$$= C + XE(p(x), q(x)) \tag{1.14}$$

Here Equation 1.12 shows that the KLD is a combination of the negative entropy ($-H(p(x))$ of the empirical data distribution, and the cross-entropy $XE(p(x), q(x))$ between the empirical data and our model distribution. Because we are *optimizing* only q(x), we can effectively ignore the entropy term (later named C and combined with other constant terms), as it will not change. This means minimizing the KLD between $p(x)$ and $q(x)$ is equivalent to minimizing the cross-entropy between $p(x)$ and $q(x)$. $p(x)$ is never changing, so minimizing $-\log q(x)$ (also called the negative log-likelihood) *also* minimizes cross-entropy.

In each case, some shorthand will be denoted for the last line of the loss so the final formulation closer resembles the implementation in code. In the classification case (cross-entropies) this involves y_t the true labels, and y_p the predicted labels.

1.4.2 Bernoulli Cross-entropy

Bernoulli cross-entropy is named as such because the labels are assumed to be from a Bernoulli distribution (Buja et al., 2005), which is often used in two-class classification problems. The Bernoulli distribution represents a random variable which has two possible values: 0 or 1, with occurrence probability p and $1 - p$, respectively.

$$p = \begin{cases} p, & \text{if } p = 1 \\ 1 - p, & \text{if } p = 0 \end{cases} \tag{1.15}$$

$$\begin{aligned} D_{KL}(p(x)||q(x)) &= \int p(x) \log p(x) - \int p(x) \log q(x) && (1.16) \\ &= \sum p(x) \log p(x) - \sum p(x) \log q(x) && (1.17) \\ &= C - \sum_{c \in 0,1} p(x) \log q(x) && (1.18) \\ &= - \sum_{c \in 0,1} p(x) \log q(x) + C && (1.19) \\ &\sim= - \sum_{c \in 0,1} p(x) \log q(x) && (1.20) \\ &\sim= -p(x) \log q(x) - (1 - p(x)) \log(1 - q(x)) && (1.21) \\ &\sim= \sum_{i=0}^{l} -y_{t[i]} \log y_p(x_i) - (1 - y_{t[i]}) \log(1 - y_p(x_i)) && (1.22) \end{aligned}$$

The integrals of the KLD become sums because of the discrete nature of the distribution. The final derivation comes due to the definition of the Bernoulli probability mass function (PMF) Equation 1.15. Minimizing the final loss term (C can be ignored because it is constant with respect to q(x)) will match the model predictions $y_p = f(x)$. The function $f(x)$ is a floating point prediction between 0 and 1. This will, after optimizing parameters θ, attempt to match the labeled dataset y_t, which has values of either 0 or 1.

1.4.3 Categorical Cross-entropy

The categorical (also known as multinoulli) distribution is a K class generalization of the two class Bernoulli (Hastie et al., 2001). Generally the prediction is a vector of probabilities for each class, so the target y_t is a class in *one-hot* representation as a vector of length K where K is the number of classes. For example (with 5 classes) 0 becomes $[1, 0, 0, 0, 0]$, 1 becomes $[0, 1, 0, 0, 0]$, and so on. $v[c]$ will generally refer to indexing into this vector v, assuming indices start from 0.

$$p = \begin{cases} p[c], & \text{if } c = k \\ 0, & \text{if } c \neq k \end{cases} \tag{1.23}$$

$$\begin{aligned} D_{KL}(p(x)||q(x)) &= \sum p(x) \log p(x) - \sum p(x) \log q(x) && (1.24)\\ &= C - \sum_{c \in 0 \ldots K} p(x) \log q(x) && (1.25)\\ &= - \sum_{c \in 0 \ldots K} p(x) \log q(x) + C && (1.26)\\ &\sim= - \sum_{c \in 0 \ldots K} p(x) \log q(x) && (1.27)\\ &\sim= - \sum_{c \in 0 \ldots K} p(x)[c] \log q(x)[c] && (1.28)\\ &\sim= - \sum_{i=0}^{l} \sum_{c \in 0 \ldots K} y_{t[i]}[c] \log y_p(x_i)[c] && (1.29) \end{aligned}$$

1.4.4 Gaussian Negative Log-Likelihood

Another approach is to make a Gaussian assumption, using the probability distribution shown in Equation 1.30.

$$p = \frac{1}{(2\pi\sigma)^{\frac{1}{2}}} \exp(-\frac{(x-\mu)^2}{2\sigma^2}) \tag{1.30}$$

We track the parameters of $p(x)$ as μ_1 and σ_1, and the parameters of $q(x)$ as μ_2 and σ_2 (Bayer, 2011).

$$D_{KL}(p(x)||q(x)) = \int p(x)\log p(x)dx - \int p(x)\log q(x)dx \tag{1.31}$$

$$= \int p(x)\log \frac{p(x)}{q(x)}dx \tag{1.32}$$

$$= \int p(x)\log \frac{\frac{1}{(2\pi\sigma_1^2)^{\frac{1}{2}}}\exp(-\frac{(x-\mu_1)^2}{2\sigma_1^2})}{\frac{1}{(2\pi\sigma_2^2)^{\frac{1}{2}}}\exp(-\frac{(x-\mu_2)^2}{2\sigma_2^2})}dx \tag{1.33}$$

$$= \int p(x)\log \frac{1}{(2\pi\sigma_1^2)^{\frac{1}{2}}}\exp(-\frac{(x-\mu_1)^2}{2\sigma_1^2})dx \tag{1.34}$$

$$- \int p(x)\log \frac{1}{(2\pi\sigma_2^2)^{\frac{1}{2}}}\exp(-\frac{(x-\mu_2)^2}{2\sigma_2^2})dx \tag{1.35}$$

$$= -H_g(x) - \int p(x)\log \frac{1}{(2\pi\sigma_2^2)^{\frac{1}{2}}}\exp(-\frac{(x-\mu_2)^2}{2\sigma_2^2}) \tag{1.36}$$

$$= -H_g(x) + \frac{1}{2}\log(2\pi\sigma_2^2) - \int p(x)\log\exp(-\frac{(x-\mu_2)^2}{2\sigma_2^2})dx \tag{1.37}$$

$$= -H_g(x) + \frac{1}{2}\log(2\pi\sigma_2^2) + \int p(x)\frac{(x-\mu_2)^2}{2\sigma_2^2}dx \tag{1.38}$$

$$= -H_g(x) + \frac{1}{2}\log(2\pi\sigma_2^2) + \frac{1}{2\sigma_2^2}(\int p(x)x^2dx - \int p(x)2x\mu_2dx$$

$$+ \int p(x)\mu_2^2dx) \tag{1.39}$$

$$= -H_g(x) + \frac{1}{2}\log(2\pi\sigma_2^2) + \frac{1}{2\sigma_2^2}(E[x^2] - 2E[x]\mu_2 + \mu_2^2) \tag{1.40}$$

$$= -H_g(x) + \frac{1}{2}\log(2\pi\sigma_2^2) + \frac{1}{2\sigma_2^2}(E[x] - \mu_2)^2 \tag{1.41}$$

$$= -H_g(x) + \frac{1}{2}\log(2\pi) + \frac{1}{2}\log(\sigma_2^2) + \frac{1}{2\sigma_2^2}\frac{1}{l}\sum_{i=0}^{l}(x_i - \mu_2)^2 \tag{1.42}$$

$$= C + \log(\sigma_2) + \frac{1}{2\sigma_2^2}\frac{1}{l}\sum_{i=0}^{l}(x_i - \mu_2)^2 \tag{1.43}$$

$$= \log(\sigma_2) + \frac{1}{2\sigma_2^2}\frac{1}{l}\sum_{i=0}^{l}(x_i - \mu_2)^2 \tag{1.44}$$

$$= \sum_{i=0}^{l}\log(\sigma_2(x_i)) + \frac{1}{2(\sigma_2(x_i))^2}(x_i - \mu_2(x_i))^2 \tag{1.45}$$

Here we see that a model with Gaussian negative log-likelihood would need to output two things, $\mu_2(x_i)$ and $\sigma_2(x_i)$ (Nowak, 2009; Bishop, 2006). In practice it is often necessary to restrict the minimum variance allowed. This is because optimizing this cost can improve arbitrarily by shrinking $\sigma_2(x_i)$ toward 0 if $\mu_2(x_i)$ lies directly on the data point.

1.4.5 Mean Squared Error

Given the previous derivation, it is easy to see that if σ_2 is assumed constant, the final result is the well known equation for mean squared error where $\mu_2(x_i)$ is the prediction of the model (Hastie et al., 2001). We typically set $\sigma_2 = 1$ for simplicity, but any constant value can be used and then factored out.

$$\min D_{KL}(p(x)||q(x)) = \min_{\theta} \frac{1}{l} \sum_{i=0}^{l} (x_i - \mu_2(x_i))^2 \tag{1.46}$$

1.5 A Basic Model

The method that we use to calculate the predictions $f()$ (or $q(x)$ in the previous notation) is often called a *model*. One of the simplest models is the *linear model* which is a simple linear transformation of the input (Strang, 2006). Defining the number of features, m, in each sample, x_i, with a total of n samples, results in x (the dataset) having shape (n, m). Introducing parameters W, size (m, d) and b, size (d), the model is as shown in Equation 1.47. The dimension d is the number of outputs needed in the linear model and depends on the cost chosen to couple with the model.

$$q(x) = xW + b \tag{1.47}$$

In practice, the equation is often implemented as in Equation 1.47 but typically written in formulae as Equation 1.48

$$q(x) = Wx + b \tag{1.48}$$

Using this simple model $q()$, our goal is to minimize a chosen loss with respect to the parameters W and b. The formula $q(x) = Wx + b$ gives the predicted value of the model. The theoretical properties of linear models are well explored (Hastie et al., 2001), and form an entire field of study in machine learning, so we only briefly introduce the basics of linear models here.

1.5.1 Gradient Based Optimization

Though some models can be minimized in closed form for certain losses including Equation 1.48 as demonstrated in Hastie et al. (2001), this is not generally true. We may wish to take another approach to optimization, in order to handle cases where closed form solutions are not possible. *Gradient based* optimization methods are a simple and effective way to optimize for loss, (Bottou, 1998). To gain understanding of gradient based methods, we must first introduce the idea of *convexity*. Convexity is a general name for functions which have only one place where the derivative is 0 (also known as the *critical point*). Optimization for machine learning generally operates on the assumption of minimization, although it is possible to convert convex functions to concave ones with a simple sign change.

Imagine we wish to find the minimum of a function which follows the curve $y = x^2$ shown in Figure 1.2. If we continuously move down the "hill" (down the gradient with respect to x, Figures 1.3) we will eventually reach the minimum of the function. Choosing how far to move each time the gradient is calculated is a choice that must be made. We generally call this *step size*, or *learning rate* in many gradient based optimizers. The learning rate is the first of many possible *hyperparameters*, which is a general terminology for settings that cannot be optimized, and must be set beforehand by the algorithm designer or programmer. The choosing of hyperparameters is absolutely critical to the performance of many models, and comprises folk knowledge in different fields of machine learning.

If we instead start on the right as in Figure 1.4, the gradient points the opposite direction, and we still move toward the global minimum. The theory behind gradient based optimization for convex (and non-convex) models is a huge field of study (Bottou, 1998), but here we present a simple example explaining the intuition behind gradient based optimization. Additionally, it is easy to see that this type of model can also work for non-convex functions, given the right settings of

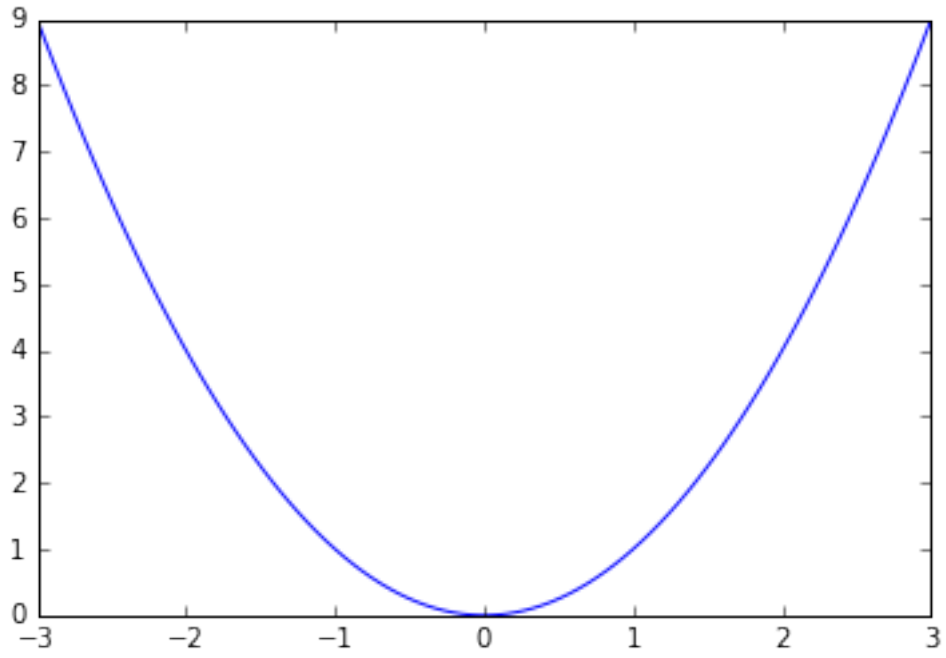


Figure 1.2 – Function $y = x^2$

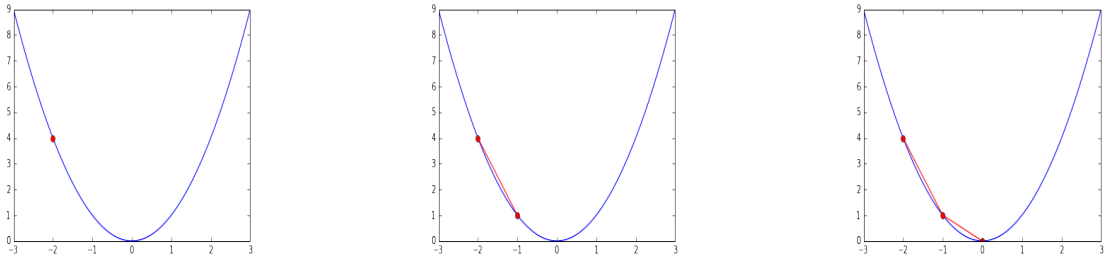


Figure 1.3 – Moving down the gradient to a minimum

the optimizer.

Figure 1.5 shows a *non-convex* problem, with a minima near $x = -1$. It easy to see that with the right step size we would still be able to find the minimum of the function, by stepping directly over the problem region. However, without prior knowledge of where the problem region lies it would be quite difficult to purposefully choose settings that avoid issues. Indeed, there are many different techniques used when bridging the gap from convex linear models to non-convex models such as neural networks (LeCun et al., 1998).

The general problem of avoiding local minima (or more likely, saddle points

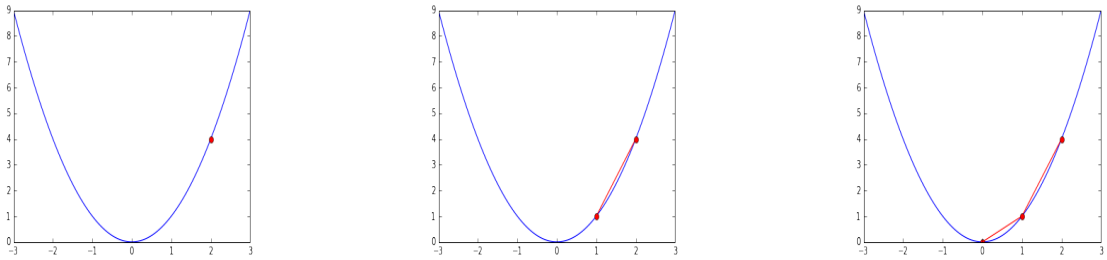


Figure 1.4 – Descending from the other side

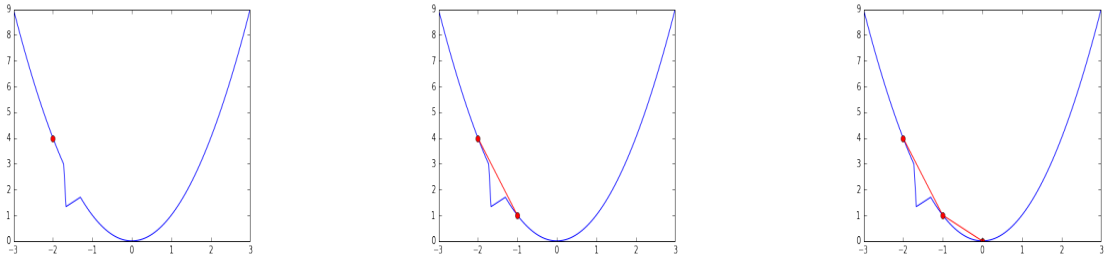


Figure 1.5 – Descending non-convex function, by "skipping" the problem region

as in Choromanska et al. (2014) and Dauphin et al. (2014)) in optimization is a problem of current interest as many models are *not* guaranteed to be convex with respect to all parameters. It is also important to note that we generally optimize $L(y, f(x, \theta))$ with respect to parameters θ. The general intuition of gradient descent as a movement along a surface can be quite useful (Goodfellow, 2015).

The process of proceeding from a starting point, or initialization, to some minimum of a function by changing parameters θ, whether by gradient descent or some other means, is broadly called *learning*. *Optimization* and *learning* are often used interchangeably in existing literature as well as in this thesis, but there exist methods of learning which are not based on optimization explicitly (Mitchell, 1998).

Deriving the *gradients* of the cost with respect to the parameters θ in a linear model (θ being the combined elements of W and b) is a straightforward application of calculus and there are many resources for such derivations such as Bishop (2006) and Hastie et al. (2001). Computational toolkits for gradient based machine learning such as Bastien et al. (2012) and Abadi et al. (2015) often use methods to automatically calculate the gradients of many common functions either symbolically or numerically, and are a crucial piece of implementing more complex models such as neural networks.

An additional note is that optimization generally becomes more difficult as the number of parameters grows, also known as the "curse of dimensionality" in Bellman (1957). There exist models with thousands or millions of parameters and optimizing such models can be quite difficult especially in non-convex settings. Finding clever ways to reduce the number of necessary parameters for a model can often greatly improve performance on a given task, and we will discuss some techniques for *parameter sharing* in later sections.

1.5.2 Stochastic, Minibatch, and Batch Gradient Descent

In the framework of gradient based optimization, we have additional choices on how to actually implement the optimization algorithm. Theoretical guarantees on convergence speed in the case of *convex* functions (Bottou, 1998) often lean toward taking a single example x_i, calculating the gradient of the cost $L(x_i)$ with respect to parameters θ, then performing an update $\theta_t = \theta_{t-1} - \eta \text{grad}_\theta(L(x_i))$, with *learning rate* $\eta > 0$ and previous parameter settings θ_{t-1}. This parameter update rule is commonly known as stochastic gradient descent, or SGD (Bottou, 1998).

By calculating the average cost with respect to several samples (sometimes called a *minibatch*), and then calculating the updates to θ, we can smooth out issues related to badly modeling single samples, while the dataset as a whole is well modeled. In the extreme case, we can calculate the loss over the entire dataset (so called *batch gradient descent*), and then update parameters, θ.

In general the choice of which type of gradient descent (stochastic, minibatch, or batch) to use is yet another *hyperparameter*. Many practitioners choose to use minibatch gradient descent due to the ability to increase or decrease the number of samples in the minibatch, allowing flexibility when modeling new datasets, while also controlling practical issues such as memory usage and data access times. There also exist more advanced gradient based optimizers and adaptations such as discussed in Kingma and Ba (2015), Dauphin et al. (2014), Sutskever et al. (2013), Bengio et al. (2013), Zeiler (2012), Hinton (2012), and Duchi et al. (2011).

1.5.3 Regularization

In addition to a cost assumption, it is common to add constraints on the *type* of solution desired, in the form of an additional *regularizer*, RG. In the framework

of empirical risk minimization, this results in a modified formulation as seen in Equation 1.49.

$$ER(\theta) = \frac{1}{l}\sum_{i=1}^{l} L(y_i, f(x_i, \theta)) + RG(\theta) \tag{1.49}$$

The case of weight norm penalties is discussed in Vapnik (1991), and is a common addition to many models to reduce numerical issues and improve training stability. In the specific case of linear models the addition of weight norm constraints results in the well known LASSO ($L1$), Ridge ($L2$), and ElasticNet estimators ($L1$ and $L2$) as demonstrated in Bishop (2006). The relative weighting between the loss, L, and the regularizer, RG, is another important hyperparameter. It is typically chosen by trial and error on a subset of the full dataset.

1.6 Basic Learning Algorithms

1.6.1 Linear Regression

Combining the linear model $q(x) = Wx + b$ with the mean squared error loss results in the following formulation, where y_i is the paired y coordinate for the sample x_i.

$$L(x) = \frac{1}{l}\sum_{i=0}^{l}(y_i - q(x_i))^2 \tag{1.50}$$

$$L(x) = \frac{1}{l}\sum_{i=0}^{l}(y_i - (Wx_i + b))^2 \tag{1.51}$$

1.6.2 Logistic Regression

Bernoulli cross-entropy with $q(x) = Wx + b$, is a potentially simple model for classification, but there is one major problem. The default output of $q(x)$ is *unbounded*, but for the Bernoulli cross-entropy cost to work, the output of $q(x)$ must be bounded between 0 and 1. One way to bound the output is to introduce a *non-linear activation* to squash the outputs into the correct range. The *sigmoid*

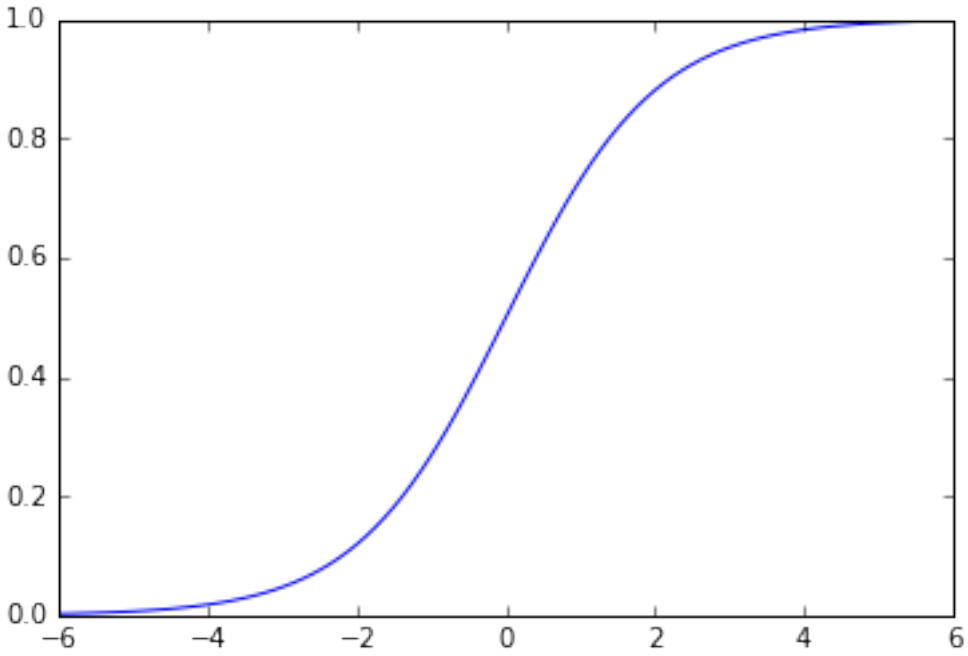


Figure 1.6 – Sigmoid function $sigm(x) = \frac{1}{1+\exp^{-x}}$

(also known as the logistic function), shown in Figure 1.6, is the ideal function to enforce the bounds.

Denoting the sigmoid function as $sigm(x)$, we modify the model $q(x) = sigm(Wx+b)$. Plugging into the Bernoulli cross-entropy, we see the classic formulation for logistic regression. Logistic regression is very closely related to the *perceptron* which forms the foundations of modern neural networks (Rosenblatt, 1958).

$$L(x) = \frac{1}{l}\sum_{i=0}^{l} y_i \log q(x_i) \tag{1.52}$$

$$L(x) = \frac{1}{l}\sum_{i=0}^{l} y_i \log sigm(Wx_i + b) \tag{1.53}$$

1.6.3 Multinomial Logistic Regression

Using the familiar linear model $q(x) = Wx+b$ with the categorical cross-entropy loss has similar problems as two-class logistic regression. Once again, it is necessary to bound the outputs between 0 and 1 in order for the loss function to work. In

the multiclass case, the *softmax* activation function, denoted by $s(x)$ and shown in Equation 1.54, is the best choice (Bishop, 2006).

$$s(x) = \frac{\exp^{x_j}}{\sum_{i=0}^{k} \exp^{x_k}} \; for \; j = 0 \cdots k \tag{1.54}$$

We can define the full loss for multinomial logistic regression using the correct class from the target data, c, combined with $q(x) = s(Wx + b)$.

$$L(x) = \frac{1}{l} \sum_{i=0}^{l} y_i[c] \log q(x_i)[c] \tag{1.55}$$

$$L(x) = \frac{1}{l} \sum_{i=0}^{l} y_i[c] \log s(Wx_i + b)[c] \tag{1.56}$$

1.7 Neural Networks

Next, we might consider ways to add *more* parameters to these kinds of models. One way is to create more input features, m, so that the matrix W (shape (m, d)) becomes larger. Next, we might try having two matrices, W_1 and W_2, of size (m, d_1) and (d_1, d), so that the model becomes $q(x) = W_2 W_1 x + b$. Unfortunately this cannot increase the number of parameters (sometimes called capacity), because the composition of linear transforms can still be represented by a single transform (Ogus, 2007). When holding the feature size constant with a required output size dependent on the cost and targets, there *seems* to be no way to increase the number of parameters.

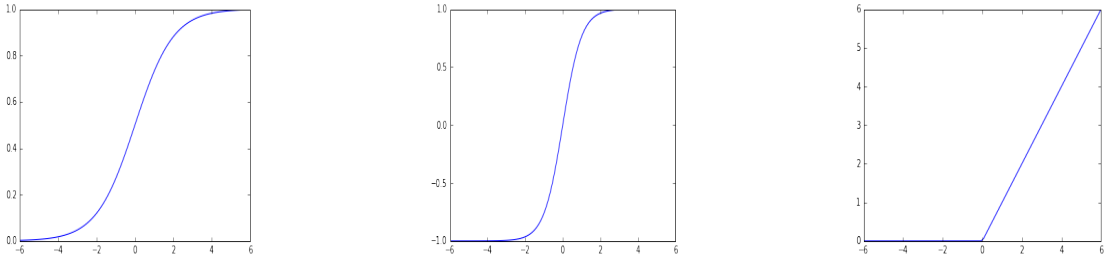


Figure 1.7 – Graphs demonstrating nonlinear functions from left to right: Sigmoid ($\frac{1}{1+\exp -x}$), tanh ($\frac{1-\exp^{-2x}}{1+\exp^{-2x}}$), and rectified linear ($0 \, if \, x < 0 \, else \, x$)

One way to prevent the linear transformations from combining is to introduce a *non-linearity*, such as a sigmoid, hyperbolic tangent, or rectified linear functions. Examples of each can be seen in Figure 1.7. The resulting model may look something like $q(x) = W_1 sigm(W_2 x) + b$, or for two functions $f()$ and $g()$, $q(x) = f(g(x))$. To calculate the gradients with respect to the parameters associated with $f()$ and $g()$, the standard chain rule of calculus can be used (Finney et al., 2003; Rumelhart et al., 1986). This stacking of multiple parameterized non-linear functions (called *layers*) with a loss function forms a basic *neural network* sometimes called a multilayer perceptron, or MLP is shown in Rumelhart et al. (1986). These linear transforms with a non-linearity applied afterward are commonly called feedforward layers. The loose assumption of feedforward layers in the MLP is roughly "everything to everything", or that the value at $x_{i[n]}$ depends on all $x_{i[\neq n]}$, meaning the function captures $f(x_{i[0]}...x_{i[n]})$ for all feature indices n.

1.8 Networks Which Exploit Structure

The models we have talked about so far share parameters over dataset samples, roughly based on the assumption that dataset examples are drawn from some underlying data distribution or have some common information which is shared over examples. There are additional assumptions we can make about the structure of *each* example, which often allows for a further reduction in the number of parameters. *Parameter sharing* is one of the most powerful tools in the arsenal of practitioners, and finding new ways to share parameters on a dataset can often result in much better solutions than previous attempts.

1.8.1 Convolutional Networks

The basic motivation for convolutional neural networks, first seen in Fukushima (1980), centers around the concept of shared statistics. Namely, we believe that in some types of data (such as images) there are statistics which are shared over the data sample. This is commonly exploited in the form of cutting images into smaller "patches", then processing patches, rather than whole images Coates et al. (2012). One common assumption is that images are made from a hierarchy of shape types,

starting from edges and blobs which are then further combined into more complicated shapes. If we wanted to craft an edge detector, it would be a bit strange to choose a number of parameters that is the same size as the image. This is because edge detection is fundamentally *local*. Rather we can use something much smaller such as a 3 by 3 filter which is repeatedly dotted and shifted over the whole image Sobel and Feldman (1968). This repeated "dot and shift" procedure is also known as *convolution*, and is a common operation in image processing. Convolution with handcrafted filters has formed the core of signal and image processing for many years, but by using the right layers in a neural network, we can create *learned filters* which start randomly and are trained by gradient descent. These filters can be optimized toward a specific goal, combined with the right cost and task such as image classification (Krizhevsky et al., 2012a), audio modeling (Dieleman and Schrauwen, 2014), or text classification [cite convnet sentiment]. In addition we can stack these layers on top of each other with the goal of learning a set of hierarchical filters such that for images it first detects edges, then groups of edges, and eventually detectors for complex objects Yosinski et al. (2015). A similar intuition of first matching small features, then hierarchical groupings of features as the network gets deeper, also holds for other types of inputs. The details of convolutional networks can be found in many places such as Fukushima (1980), LeCun et al. (1997), Krizhevsky et al. (2012a), Goodfellow et al. (2016), Sermanet et al. (2014), Simonyan and Zisserman (2015), Szegedy et al. (2014). Sharing weights over dimensions of data because of similar statistics provides an important way to improve the performance of neural networks. We use this technique heavily in modern research, and the papers included in Chapters 2.2 and 3 are no exception. One key piece of convolutional networks is the assumption of independence. In a given layer, each local filter application is assumed to be *independent* of the other filter applications. As such the filters at each layer can be calculated in parallel, allowing for efficient computation on modern computers and parallel processing devices. We can roughly say that convolutional networks model the assumption $f(x_{i[j-\frac{k}{2}]}...x_{i[j+\frac{k}{2}]})$ where k is the convolutional filter size, and j is the current filter position. Convolutional models capture the full relationship between all features $f(x_{i[0]}...x_{i[n]})$ only after stacking many convolutional layers, effectively partitioning the space over depth so that nearby interactions have highly similar paths through the network while more distant features have more disjoint paths (Dinh, 2016). If

the sample x_i is multi-dimensional such as an image, the independence assumption may also have multiple dimensions depending on the size of the filter in each dimension. In the extreme case of filter size equal to sample size, a convolutional layer is equivalent to a feedforward layer with the same nonlinearity.

1.8.2 Recurrent Neural Networks

Recurrent neural networks are another way of sharing parameters, as seen in Jordan (1986) and Elman (1990). Unlike the previous example of convolutional neural networks, recurrent neural networks make no independence assumptions within the same layer. Instead, it factorizes the joint model $f(x_{i[0]}...x_{i[n]})$ as a sequence of conditional functions $f(x_{i[0]})$, $f(x_{i[1]}|x_{i[0]})$, $f(x_{i[2]}|x_{i[1]}, x_{i[0]})$, $f(x_{i[3]}|x_{i[2]}, x_{i[1]}, x_{i[0]})$ ... up to $f(x_{i[n]}|x_{i[n-1]}, ..., x_{i[0]})$. In standard training (combined with the correct cost) these conditional functions form probability distributions, so that we effectively factorize the joint probability distribution $p(x_{i[0]}...x_{i[n]})$ as a product of conditional distributions $p(x_{i[0]}) * p(x_{i[1]}|x_{i[0]})...p(x_{i[n]}|x_{i[n-1]}...x_{i[0]})$ (Larochelle and Murray, 2011a). The general idea of recurrent neural networks is that the output for a specific position k can be thought of as a function of the current input $x_i[k]$ and some previous information (often called the *hidden state*) h_{k-1} which attempts to compress all previous history $x_{i[0...k-1]}$. This results in a function $h_k = f(r(x_{i[k]}) + h_{k-1})$, with input feature model $r()$ and hidden state h. By starting this process recursively, and allowing the hidden state to be *shared* over all steps k in $0...n$, this results in the factorization described above. In addition, the function $f()$ and hidden state processing can have memory, error correction, or a host of different useful computational blocks. Details for specific recurrent neural networks will be found in subsequent chapters, but a general overview of the concept and introductions to specific architectures can be found in Hochreiter and Schmidhuber (1997), Cho et al. (2014), Goodfellow et al. (2016), Graves et al. (2013a) Jozefowicz et al. (2015).

1.9 Structured Prediction

If the chosen model (and associated cost) is more advanced, we may use it to predict multiple things about a single sample. Some examples include classifying each pixel in an image (such as road, car, tree) as in Eigen and Fergus (2015) or notes in a piece of music as in Eck and Schmidhuber (2002) and Boulanger-Lewandowski et al. (2012a). Standard prediction problems (such as classification) can be thought of as learning a mapping function $f(y_i|\mathbf{x_i})$, where x_i is a feature vector of some kind, and y_i is a single target for that mapping. A simple problem dealing with structured prediction might involve prediction of multiple targets for each sample, $f(y_{i[0]}...y_{i[l]}|\mathbf{x_i})$. Many other problems have further knowledge which can be mined or described about the relationship between targets. This allows various breakdowns of the cost function into related subproblems that may (or may not) share parameters. These types of problems are common in natural language processing and speech recognition. Much study in modern research is focused on better ways of handling so called *structured output* tasks, and incorporating prior knowledge in the output structure can provide massive improvements on a given task. There are also ways to learn one to many mappings by outputting the parameters for mixture densities Bishop (1994), but we leave this for further exploration in subsequent chapters. In the next chapter we discuss a method for structured prediction of handwriting and audio sequences, called a variational recurrent neural network, or VRNN.

2 A Recurrent Latent Variable Model For Sequential Data

2.1 Prologue to the Article

A Recurrent Latent Variable Model for Sequential Data. Junyoung Chung, Kyle Kastner, Laurent Dinh, Kratarth Goel, Aaron Courville, Yoshua Bengio.

Published in Advances in Neural Information Processing Systems 28, NIPS 2015.

Personal Contribution. The idea for this form of model with recurrent latent variables was conceived by Yoshua and Junyoung with involvement from Laurent, Aaron, and I in a brainstorming session during the MILA speech synthesis group meeting. Junyoung developed a prototype that showed the idea working and we soon started running experiments on various datasets. Laurent formalized the mathematical side of the model, with Junyoung reviewing and input from Yoshua and Aaron. Later, Kratarth joined Junyoung and I in running additional experiments. I handled dataset issues around the handwriting dataset and assisted Junyoung with speech experiments. Laurent developed the visualization for the change in latent states over the sequence. We all assisted in writing the final paper. I provided content in the background, datasets, and analysis sections of the paper. My research role centered around assisting in debugging the model and code, organizing test datasets, running experiments, and analyzing generated results.

Affiliations

Junyoung Chung, MILA, Département d'Informatique et de Recherche Opérationnelle, Université de Montréal

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

Laurent Dinh, MILA, Département d'Informatique et de Recherche Opérationnelle, Université de Montréal

Kratarth Goel, MILA, Département d'Informatique et de Recherche Opérationnelle, Université de Montréal

Aaron Courville, MILA, Département d'Informatique et de Recherche Opérationnelle, Université de Montréal

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

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

2.2 Abstract

In this paper, we explore the inclusion of latent random variables into the hidden state of a recurrent neural network (RNN) by combining the elements of the variational autoencoder. We argue that through the use of high-level latent random variables, the *variational RNN* (VRNN) [1] can model the kind of variability observed in highly structured sequential data such as natural speech. We empirically evaluate the proposed model against other related sequential models on four speech datasets and one handwriting dataset. Our results show the important roles that latent random variables can play in the RNN dynamics.

2.3 Introduction

Learning generative models of sequences is a long-standing machine learning challenge and historically the domain of dynamic Bayesian networks (DBNs) such as hidden Markov models (HMMs) and Kalman filters. The dominance of DBN-based approaches has been recently overturned by a resurgence of interest in recurrent neural network (RNN) based approaches. An RNN is a special type of neural network that is able to handle both variable-length input and output. By training

1. Code is available at http://www.github.com/jych/nips2015_vrnn

an RNN to predict the next output in a sequence, given all previous outputs, it can be used to model joint probability distribution over sequences.

Both RNNs and DBNs consist of two parts: (1) a transition function that determines the evolution of the internal hidden state, and (2) a mapping from the state to the output. There are, however, a few important differences between RNNs and DBNs.

DBNs have typically been limited either to relatively simple state transition structures (e.g., linear models in the case of the Kalman filter) or to relatively simple internal state structure (e.g., the HMM state space consists of a single set of mutually exclusive states). RNNs, on the other hand, typically possess both a richly distributed internal state representation and flexible non-linear transition functions. These differences give RNNs extra expressive power in comparison to DBNs. This expressive power and the ability to train via error backpropagation are the key reasons why RNNs have gained popularity as generative models for highly structured sequential data.

In this paper, we focus on another important difference between DBNs and RNNs. While the hidden state in DBNs is expressed in terms of random variables, the internal transition structure of the standard RNN is entirely deterministic. The only source of randomness or variability in the RNN is found in the conditional output probability model. We suggest that this can be an inappropriate way to model the kind of variability observed in highly structured data, such as natural speech, which is characterized by strong and complex dependencies among the output variables at different timesteps. We argue, as have others (Boulanger-Lewandowski et al., 2012b; Bayer and Osendorfer, 2014), that these complex dependencies cannot be modelled efficiently by the output probability models used in standard RNNs, which include either a simple unimodal distribution or a mixture of unimodal distributions.

We propose the use of high-level latent random variables to model the variability observed in the data. In the context of standard neural network models for non-sequential data, the variational autoencoder (VAE) (Kingma and Welling, 2014; Rezende et al., 2014) offers an interesting combination of highly flexible non-linear mapping between the latent random state and the observed output and effective approximate inference. In this paper, we propose to extend the VAE into a recurrent framework for modelling high-dimensional sequences. The VAE can model

complex multimodal distributions, which will help when the underlying true data distribution consists of multimodal conditional distributions. We call this model a *variational RNN* (VRNN).

A natural question to ask is: how do we encode observed variability with latent random variables? The answer to this question depends on the nature of the data itself. In this work, we are mainly interested in highly structured data that often arises in AI applications. By highly structured, we mean that the data is characterized by two properties. Firstly, there is a relatively high signal-to-noise ratio, meaning that the vast majority of the variability observed in the data is due to the signal itself and cannot reasonably be considered as noise. Secondly, there exists a complex relationship between the underlying factors of variation and the observed data. For example, in speech, the vocal qualities of the speaker have a strong but complicated influence on the audio waveform, affecting the waveform in a consistent manner across frames.

With these considerations in mind, we suggest that our model variability should induce *temporal dependencies across timesteps*. Thus, like DBN models such as HMMs and Kalman filters, we model the dependencies between the latent random variables across timesteps. While we are not the first to propose integrating random variables into the RNN hidden state (Boulanger-Lewandowski et al., 2012c; Bayer and Osendorfer, 2014; Fabius et al., 2014; Gregor et al., 2015), we believe we are the first to integrate the dependencies between the latent random variables at neighboring timesteps.

We evaluate the proposed VRNN model against other RNN-based models – including a VRNN model without introducing temporal dependencies between the latent random variables – on two challenging sequential data types: natural speech and handwriting. We demonstrate that for the speech modelling tasks, the VRNN-based models significantly outperform the RNN-based models and the VRNN model that does not integrate temporal dependencies between latent random variables.

2.4 Background

2.4.1 Sequence modelling with Recurrent Neural Networks

An RNN can take as input a variable-length sequence $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_T)$ by recursively processing each symbol while maintaining its internal hidden state $\mathbf{h}$. At each timestep t, the RNN reads the symbol $\mathbf{x}_t \in \mathbb{R}^d$ and updates its hidden state $\mathbf{h}_t \in \mathbb{R}^p$ by:

$$\mathbf{h}_t = f_\theta\left(\mathbf{x}_t, \mathbf{h}_{t-1}\right), \tag{2.1}$$

where f is a deterministic non-linear transition function, and θ is the parameter set of f. The transition function f can be implemented with gated activation functions such as long short-term memory (LSTM, Hochreiter and Schmidhuber, 1997) or gated recurrent unit (GRU, Cho et al., 2014). RNNs model sequences by parameterizing a factorization of the joint sequence probability distribution as a product of conditional probabilities such that:

$$\begin{aligned} p(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_T) &= \prod_{t=1}^{T} p(\mathbf{x}_t \mid \mathbf{x}_{<t}), \\ p(\mathbf{x}_t \mid \mathbf{x}_{<t}) &= g_\tau(\mathbf{h}_{t-1}), \end{aligned} \tag{2.2}$$

where g is a function that maps the RNN hidden state $\mathbf{h}_{t-1}$ to a probability distribution over possible outputs, and τ is the parameter set of g.

One of the main factors that determines the representational power of an RNN is the output function g in Eq. (2.2). With a deterministic transition function f, the choice of g effectively defines the family of joint probability distributions $p(\mathbf{x}_1, \ldots, \mathbf{x}_T)$ that can be expressed by the RNN.

We can express the output function g in Eq. (2.2) as being composed of two parts. The first part φ_τ is a function that returns the parameter set ϕ_t given the hidden state $\mathbf{h}_{t-1}$, i.e., $\phi_t = \varphi_\tau(\mathbf{h}_{t-1})$, while the second part of g returns the density of $\mathbf{x}_t$, i.e., $p_{\phi_t}(\mathbf{x}_t \mid \mathbf{x}_{<t})$.

When modelling high-dimensional and real-valued sequences, a reasonable choice of an observation model is a Gaussian mixture model (GMM) as used in (Graves et al., 2013a). For GMM, φ_τ returns a set of mixture coefficients α_t, means $\boldsymbol{\mu}_{\cdot,t}$

and covariances $\Sigma_{\cdot,t}$ of the corresponding mixture components. The probability of $\mathbf{x}_t$ under the mixture distribution is:

$$p_{\boldsymbol{\alpha}_t, \boldsymbol{\mu}_{\cdot,t}, \Sigma_{\cdot,t}}(\mathbf{x}_t \mid \mathbf{x}_{<t}) = \sum_j \alpha_{j,t} \mathcal{N}\left(\mathbf{x}_t; \boldsymbol{\mu}_{j,t}, \Sigma_{j,t}\right).$$

With the notable exception of (Graves et al., 2013a), there has been little work investigating the structured output density model for RNNs with real-valued sequences.

There is potentially a significant issue in the way the RNN models output variability. Given a deterministic transition function, the only source of variability is in the conditional output probability density. This can present problems when modelling sequences that are at once highly variable and highly structured (i.e., with a high signal-to-noise ratio). To effectively model these types of sequences, the RNN must be capable of mapping very small variations in $\mathbf{x}_t$ (i.e., the only source of randomness) to potentially very large variations in the hidden state $\mathbf{h}_t$. Limiting the capacity of the network, as must be done to guard against overfitting, will force a compromise between the generation of a clean signal and encoding sufficient input variability to capture the high-level variability both within a single observed sequence and across data examples.

The need for highly structured output functions in an RNN has been previously noted. Boulanger-Lewandowski et al. (2012a) extensively tested NADE and RBM-based output densities for modelling sequences of binary vector representations of music. Bayer and Osendorfer (2014) introduced a sequence of independent latent variables corresponding to the states of the RNN. Their model, called *STORN*, first generates a sequence of samples $\mathbf{z} = (\mathbf{z}_1, \ldots, \mathbf{z}_T)$ from the sequence of independent latent random variables. At each timestep, the transition function f from Eq. (2.1) computes the next hidden state $\mathbf{h}_t$ based on the previous state $\mathbf{h}_{t-1}$, the previous output $\mathbf{x}_{t-1}$ and the sampled latent random variables $\mathbf{z}_t$. They proposed to train this model based on the VAE principle (see Sec. 2.4.2). Similarly, Pachitariu and Sahani (2012) earlier proposed both a sequence of independent latent random variables and a stochastic hidden state for the RNN.

These approaches are closely related to the approach proposed in this paper. However, there is a major difference in how the prior distribution over the latent random variable is modelled. Unlike the aforementioned approaches, our approach

makes the prior distribution of the latent random variable at timestep t dependent on all the preceding inputs via the RNN hidden state $\mathbf{h}_{t-1}$ (see Eq. (2.5)). The introduction of temporal structure into the prior distribution is expected to improve the representational power of the model, which we empirically observe in the experiments (See Table 2.1). However, it is important to note that any approach based on having stochastic latent state is orthogonal to having a structured output function, and that these two can be used together to form a single model.

2.4.2 Variational Autoencoder

For non-sequential data, VAEs (Kingma and Welling, 2014; Rezende et al., 2014) have recently been shown to be an effective modelling paradigm to recover complex multimodal distributions over the data space. A VAE introduces a set of latent random variables $\mathbf{z}$, designed to capture the variations in the observed variables $\mathbf{x}$. As an example of a directed graphical model, the joint distribution is defined as:

$$p(\mathbf{x}, \mathbf{z}) = p(\mathbf{x} \mid \mathbf{z})p(\mathbf{z}). \tag{2.3}$$

The prior over the latent random variables, $p(\mathbf{z})$, is generally chosen to be a simple Gaussian distribution and the conditional $p(\mathbf{x} \mid \mathbf{z})$ is an arbitrary observation model whose parameters are computed by a parametric function of $\mathbf{z}$. Importantly, the VAE typically parameterizes $p(\mathbf{x} \mid \mathbf{z})$ with a highly flexible function approximator such as a neural network. While latent random variable models of the form given in Eq. (2.3) are not uncommon, endowing the conditional $p(\mathbf{x} \mid \mathbf{z})$ as a potentially highly non-linear mapping from $\mathbf{z}$ to $\mathbf{x}$ is a rather unique feature of the VAE.

However, introducing a highly non-linear mapping from $\mathbf{z}$ to $\mathbf{x}$ results in intractable inference of the posterior $p(\mathbf{z} \mid \mathbf{x})$. Instead, the VAE uses a variational approximation $q(\mathbf{z} \mid \mathbf{x})$ of the posterior that enables the use of the lower bound:

$$\log p(\mathbf{x}) \geq -\mathrm{KL}(q(\mathbf{z} \mid \mathbf{x}) \| p(\mathbf{z})) + \mathbb{E}_{q(\mathbf{z}|\mathbf{x})} \left[\log p(\mathbf{x} \mid \mathbf{z})\right], \tag{2.4}$$

where $\mathrm{KL}(Q\|P)$ is Kullback-Leibler divergence between two distributions Q and P.

In (Kingma and Welling, 2014), the approximate posterior $q(\mathbf{z} \mid \mathbf{x})$ is a Gaussian

$\mathcal{N}(\boldsymbol{\mu}, \text{diag}(\boldsymbol{\sigma}^2))$ whose mean $\boldsymbol{\mu}$ and variance $\boldsymbol{\sigma}^2$ are the output of a highly non-linear function of $\mathbf{x}$, once again typically a neural network.

The generative model $p(\mathbf{x} \mid \mathbf{z})$ and inference model $q(\mathbf{z} \mid \mathbf{x})$ are then trained jointly by maximizing the variational lower bound with respect to their parameters, where the integral with respect to $q(\mathbf{z} \mid \mathbf{x})$ is approximated stochastically. The gradient of this estimate can have a low variance estimate, by reparametrizing $\mathbf{z} = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\epsilon}$ and rewriting:

$$\mathbb{E}_{q(\mathbf{z}|\mathbf{x})} \left[\log p(\mathbf{x} \mid \mathbf{z})\right] = \mathbb{E}_{p(\boldsymbol{\epsilon})} \left[\log p(\mathbf{x} \mid \mathbf{z} = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\epsilon})\right],$$

where $\boldsymbol{\epsilon}$ is a vector of standard Gaussian variables. The inference model can then be trained through standard backpropagation technique for stochastic gradient descent.

2.5 Variational Recurrent Neural Network

In this section, we introduce a recurrent version of the VAE for the purpose of modelling sequences. Drawing inspiration from simpler dynamic Bayesian networks (DBNs) such as HMMs and Kalman filters, the proposed *variational recurrent neural network* (VRNN) explicitly models the dependencies between latent random variables across subsequent timesteps. However, unlike these simpler DBN models, the VRNN retains the flexibility to model highly non-linear dynamics.

Generation The VRNN contains a VAE at every timestep. However, these VAEs are conditioned on the state variable $\mathbf{h}_{t-1}$ of an RNN. This addition will help the VAE to take into account the temporal structure of the sequential data. Unlike a standard VAE, the prior on the latent random variable is no longer a standard Gaussian distribution, but follows the distribution:

$$\mathbf{z}_t \sim \mathcal{N}(\boldsymbol{\mu}_{0,t}, \text{diag}(\boldsymbol{\sigma}_{0,t}^2)) \text{ , where } [\boldsymbol{\mu}_{0,t}, \boldsymbol{\sigma}_{0,t}] = \varphi_\tau^{\text{prior}}(\mathbf{h}_{t-1}), \tag{2.5}$$

where $\boldsymbol{\mu}_{0,t}$ and $\boldsymbol{\sigma}_{0,t}$ denote the parameters of the conditional prior distribution. Moreover, the generating distribution will not only be conditioned on $\mathbf{z}_t$ but also

on $\mathbf{h}_{t-1}$ such that:

$$\mathbf{x}_t \mid \mathbf{z}_t \sim \mathcal{N}(\boldsymbol{\mu}_{x,t}, \text{diag}(\boldsymbol{\sigma}_{x,t}^2)) \text{ , where } [\boldsymbol{\mu}_{x,t}, \boldsymbol{\sigma}_{x,t}] = \varphi_\tau^{\text{dec}}(\varphi_\tau^{\mathbf{z}}(\mathbf{z}_t), \mathbf{h}_{t-1}), \tag{2.6}$$

where $\boldsymbol{\mu}_{x,t}$ and $\boldsymbol{\sigma}_{x,t}$ denote the parameters of the generating distribution, $\varphi_\tau^{\text{prior}}$ and $\varphi_\tau^{\text{dec}}$ can be any highly flexible function such as neural networks. $\varphi_\tau^{\mathbf{x}}$ and $\varphi_\tau^{\mathbf{z}}$ can also be neural networks, which extract features from $\mathbf{x}_t$ and $\mathbf{z}_t$, respectively. We found that these feature extractors are crucial for learning complex sequences. The RNN updates its hidden state using the recurrence equation:

$$\mathbf{h}_t = f_\theta \left(\varphi_\tau^{\mathbf{x}}(\mathbf{x}_t), \varphi_\tau^{\mathbf{z}}(\mathbf{z}_t), \mathbf{h}_{t-1}\right), \tag{2.7}$$

where f was originally the transition function from Eq. (2.1). From Eq. (2.7), we find that $\mathbf{h}_t$ is a function of $\mathbf{x}_{\leq t}$ and $\mathbf{z}_{\leq t}$. Therefore, Eq. (2.5) and Eq. (2.6) define the distributions $p(\mathbf{z}_t \mid \mathbf{x}_{<t}, \mathbf{z}_{<t})$ and $p(\mathbf{x}_t \mid \mathbf{z}_{\leq t}, \mathbf{x}_{<t})$, respectively. The parameterization of the generative model results in and – was motivated by – the factorization:

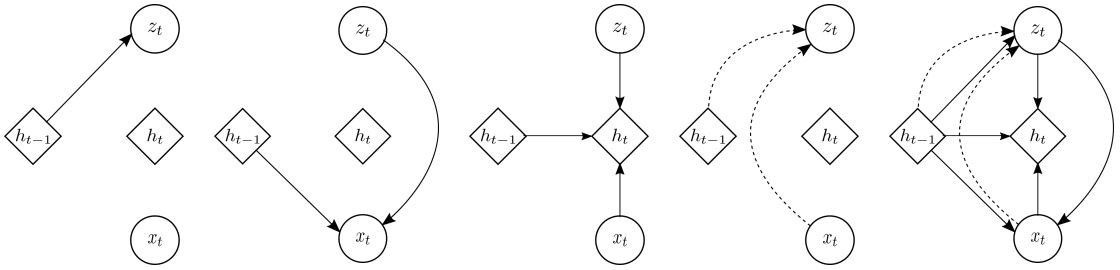
$$p(\mathbf{x}_{\leq T}, \mathbf{z}_{\leq T}) = \prod_{t=1}^{T} p(\mathbf{x}_t \mid \mathbf{z}_{\leq t}, \mathbf{x}_{<t}) p(\mathbf{z}_t \mid \mathbf{x}_{<t}, \mathbf{z}_{<t}). \tag{2.8}$$

Inference In a similar fashion, the approximate posterior will not only be a function of $\mathbf{x}_t$ but also of $\mathbf{h}_{t-1}$ following the equation:

$$\mathbf{z}_t \mid \mathbf{x}_t \sim \mathcal{N}(\boldsymbol{\mu}_{z,t}, \text{diag}(\boldsymbol{\sigma}_{z,t}^2)) \text{ , where } [\boldsymbol{\mu}_{z,t}, \boldsymbol{\sigma}_{z,t}] = \varphi_\tau^{\text{enc}}(\varphi_\tau^{\mathbf{x}}(\mathbf{x}_t), \mathbf{h}_{t-1}), \tag{2.9}$$

similarly $\boldsymbol{\mu}_{z,t}$ and $\boldsymbol{\sigma}_{z,t}$ denote the parameters of the approximate posterior. We note that the encoding of the approximate posterior and the decoding for generation are tied through the RNN hidden state $\mathbf{h}_{t-1}$. We also observe that this conditioning on $\mathbf{h}_{t-1}$ results in the factorization:

$$q(\mathbf{z}_{\leq T} \mid \mathbf{x}_{\leq T}) = \prod_{t=1}^{T} q(\mathbf{z}_t \mid \mathbf{x}_{\leq t}, \mathbf{z}_{<t}). \tag{2.10}$$



(a) Prior (b) Generation (c) Recurrence (d) Inference (e) Overall

Figure 2.1 – Graphical illustrations of each operation of the VRNN: (a) computing the conditional prior using Eq. (2.5); (b) generating function using Eq. (2.6); (c) updating the RNN hidden state using Eq. (2.7); (d) inference of the approximate posterior using Eq. (2.9); (e) overall computational paths of the VRNN.

Learning The objective function becomes a timestep-wise variational lower bound using Eq. (2.8) and Eq. (2.10):

$$\mathbb{E}_{q(\mathbf{z}_{\leq T}|\mathbf{x}_{\leq T})}\left[\sum_{t=1}^{T}\left(-\mathrm{KL}(q(\mathbf{z}_t \mid \mathbf{x}_{\leq t}, \mathbf{z}_{<t})\|p(\mathbf{z}_t \mid \mathbf{x}_{<t}, \mathbf{z}_{<t})) + \log p(\mathbf{x}_t \mid \mathbf{z}_{\leq t}, \mathbf{x}_{<t})\right)\right]. \tag{2.11}$$

As in the standard VAE, we learn the generative and inference models jointly by maximizing the variational lower bound with respect to their parameters. The schematic view of the VRNN is shown in Fig. 2.1, operations (a)–(d) correspond to Eqs. (2.5)–(2.7), (2.9), respectively. The VRNN applies the operation (a) when computing the conditional prior (see Eq. (2.5)). If the variant of the VRNN (VRNN-I) does not apply the operation (a), then the prior becomes independent across timesteps. STORN (Bayer and Osendorfer, 2014) can be considered as an instance of the VRNN-I model family. In fact, STORN puts further restrictions on the dependency structure of the approximate inference model. We include this version of the model (VRNN-I) in our experimental evaluation in order to directly study the impact of including the temporal dependency structure in the prior (i.e., conditional prior) over the latent random variables.

2.6 Experiment Settings

We evaluate the proposed VRNN model on two tasks: (1) modelling natural speech directly from the raw audio waveforms; (2) modelling handwriting generation.

Speech modelling We train the models to directly model raw audio signals, represented as a sequence of 200-dimensional frames. Each frame corresponds to the real-valued amplitudes of 200 consecutive raw acoustic samples. Note that this is unlike the conventional approach for modelling speech, often used in speech synthesis where models are expressed over representations such as spectral features (see, e.g., Tokuda et al., 2013; Bertrand et al., 2008; Lee et al., 2009).

We evaluate the models on the following four speech datasets:

1. **Blizzard**: This text-to-speech dataset made available by the Blizzard Challenge 2013 contains 300 hours of English, spoken by a single female speaker (King and Karaiskos, 2013).
2. **TIMIT**: This widely used dataset for benchmarking speech recognition systems contains 6,300 English sentences, read by 630 speakers.
3. **Onomatopoeia**[2]: This is a set of 6,738 non-linguistic human-made sounds such as coughing, screaming, laughing and shouting, recorded from 51 voice actors.
4. **Accent**: This dataset contains English paragraphs read by 2,046 different native and non-native English speakers (Weinberger, 2015).

For the Blizzard and Accent datasets, we process the data so that each sample duration is $0.5s$ (the sampling frequency used is 16kHz). Except for the TIMIT dataset, the rest of the datasets do not have predefined train/test splits. We shuffle and divide the data into train/validation/test splits using a ratio of 0.9/0.05/0.05.

Handwriting generation We let each model learn a sequence of (x, y) coordinates together with binary indicators of pen-up/pen-down, using the IAM-OnDB dataset, which consists of 13,040 handwritten lines written by 500 writers Liwicki and Bunke (2005). We preprocess and split the dataset as done in (Graves et al., 2013a).

2. This dataset has been provided by Ubisoft.

Table 2.1 – Average log-likelihood on the test (or validation) set of each task.

Models	Speech modelling				Handwriting
	Blizzard	TIMIT	Onomatopoeia	Accent	IAM-OnDB
RNN-Gauss	3539	-1900	-984	-1293	1016
RNN-GMM	7413	26643	18865	3453	1358
VRNN-I-Gauss	$\geq$ 8933	$\geq$ 28340	$\geq$ 19053	$\geq$ 3843	$\geq$ 1332
	$\approx$ 9188	$\approx$ 29639	$\approx$ 19638	$\approx$ 4180	$\approx$ 1353
VRNN-Gauss	$\geq$ 9223	$\geq$ 28805	$\geq$ 20721	$\geq$ 3952	$\geq$ 1337
	$\approx$ **9516**	$\approx$ **30235**	$\approx$ **21332**	$\approx$ 4223	$\approx$ 1354
VRNN-GMM	$\geq$ 9107	$\geq$ 28982	$\geq$ 20849	$\geq$ 4140	$\geq$ 1384
	$\approx$ 9392	$\approx$ 29604	$\approx$ 21219	$\approx$ **4319**	$\approx$ **1384**

Preprocessing and training The only preprocessing used in our experiments is normalizing each sequence using the global mean and standard deviation computed from the entire training set. We train each model with stochastic gradient descent on the negative log-likelihood using the Adam optimizer Kingma and Welling (2015), with a learning rate of 0.001 for TIMIT and Accent and 0.0003 for the rest. We use a minibatch size of 128 for Blizzard and Accent and 64 for the rest. The final model was chosen with early-stopping based on the validation performance.

Models We compare the VRNN models with the standard RNN models using two different output functions: a simple Gaussian distribution (Gauss) and a Gaussian mixture model (GMM). For each dataset, we conduct an additional set of experiments for a VRNN model without the conditional prior (VRNN-I).

We fix each model to have a single recurrent hidden layer with 2000 LSTM units (in the case of Blizzard, 4000 and for IAM-OnDB, 1200). All of φ_τ shown in Eqs. (2.5)–(2.7), (2.9) have four hidden layers using rectified linear units (Nair and Hinton, 2010) (for IAM-OnDB, we use a single hidden layer). The standard RNN models only have $\varphi_\tau^{\mathbf{x}}$ and $\varphi_\tau^{\text{dec}}$, while the VRNN models also have $\varphi_\tau^{\mathbf{z}}$, $\varphi_\tau^{\text{enc}}$ and $\varphi_\tau^{\text{prior}}$. For the standard RNN models, $\varphi_\tau^{\mathbf{x}}$ is the feature extractor, and $\varphi_\tau^{\text{dec}}$ is the generating function. For the RNN-GMM and VRNN models, we match the total number of parameters of the deep neural networks (DNNs), $\varphi_\tau^{\mathbf{x},\mathbf{z},\text{enc},\text{dec},\text{prior}}$, as close to the RNN-Gauss model having 600 hidden units for every layer that belongs to either $\varphi_\tau^{\mathbf{x}}$ or $\varphi_\tau^{\text{dec}}$ (we consider 800 hidden units in the case of Blizzard). Note that

we use 20 mixture components for models using a GMM as the output function.

For qualitative analysis of speech generation, we train larger models to generate audio sequences. We stack three recurrent hidden layers, each layer contains 3000 LSTM units. Again for the RNN-GMM and VRNN models, we match the total number of parameters of the DNNs to be equal to the RNN-Gauss model having 3200 hidden units for each layer that belongs to either $\varphi_\tau^{\mathbf{x}}$ or $\varphi_\tau^{\text{dec}}$.

2.7 Results and Analysis

We report the average log-likelihood of test examples assigned by each model in Table 2.1. For RNN-Gauss and RNN-GMM, we report the exact log-likelihood, while in the case of VRNNs, we report the variational lower bound (given with $\geq$ sign, see Eq. (2.4)) and approximated marginal log-likelihood (given with $\approx$ sign) based on importance sampling using 40 samples as in (Rezende et al., 2014). In general, higher numbers are better. Our results show that the VRNN models have higher log-likelihood, which support our claim that latent random variables are helpful when modelling complex sequences. The VRNN models perform well even with a unimodal output function (VRNN-Gauss), which is not the case for the standard RNN models.

Latent space analysis In Fig. 2.2, we show an analysis of the latent random variables. We let a VRNN model read some unseen examples and observe the transitions in the latent space. We compute $\delta_t = \sum_j (\boldsymbol{\mu}_{z,t}^j - \boldsymbol{\mu}_{z,t-1}^j)^2$ at every timestep and plot the results on the top row of Fig. 2.2. The middle row shows the KL divergence computed between the approximate posterior and the conditional prior. When there is a transition in the waveform, the KL divergence tends to grow (white is high), and we can clearly observe a peak in δ_t that can affect the RNN dynamics to change modality.

Speech generation We generate waveforms with $2.0s$ duration from the models that were trained on Blizzard. From Fig. 2.3, we can clearly see that the waveforms from the VRNN-Gauss are much less noisy and have less spurious peaks than those from the RNN-GMM. We suggest that the large amount of noise apparent in the

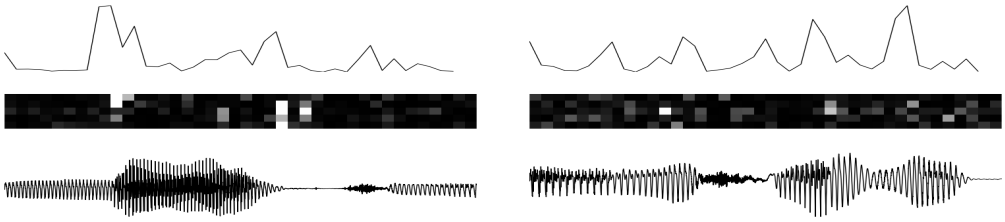


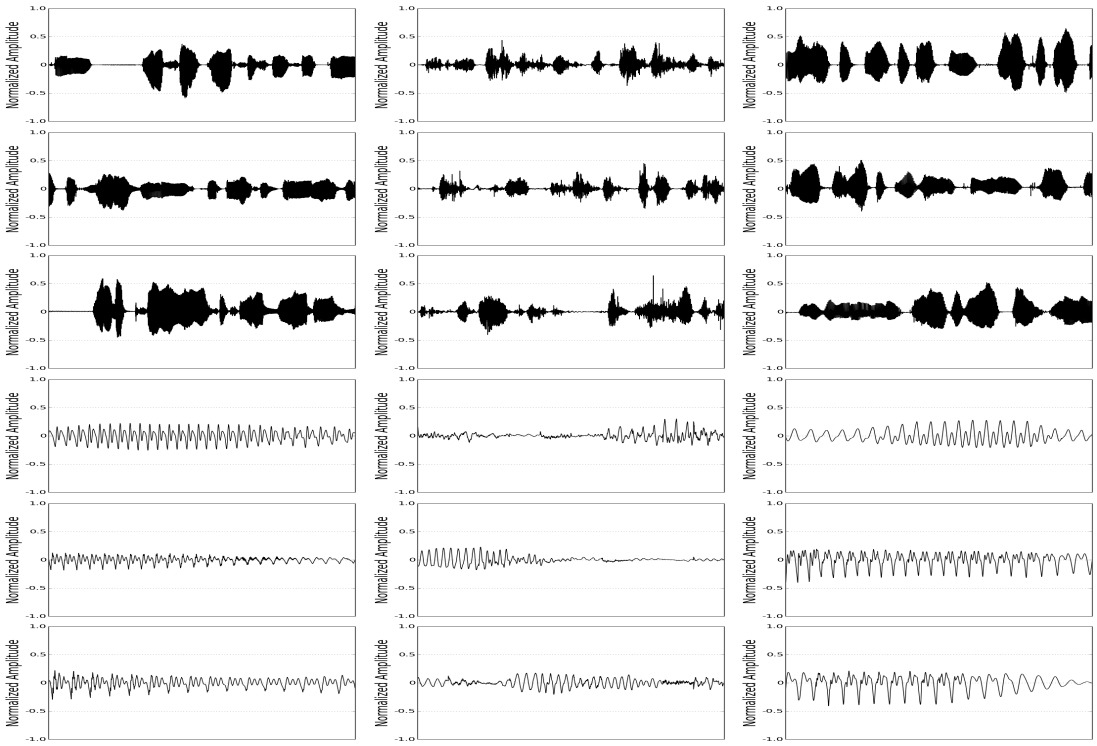
Figure 2.2 – The top row represents the difference δ_t between $\boldsymbol{\mu}_{z,t}$ and $\boldsymbol{\mu}_{z,t-1}$. The middle row shows the dominant KL divergence values in temporal order. The bottom row shows the input waveforms.

waveforms from the RNN-GMM model is a consequence of the compromise these models must make between representing a clean signal consistent with the training data and encoding sufficient input variability to capture the variations across data examples. The latent random variable models can avoid this compromise by adding variability in the latent space, which can always be mapped to a point close to a relatively clean sample.

Handwriting generation Visual inspection of the generated handwriting (as shown in Fig. 2.4) from the trained models reveals that the VRNN model is able to generate more diverse writing style while maintaining consistency within samples.

2.8 Conclusion

We propose a novel model that can address sequence modelling problems by incorporating latent random variables into a recurrent neural network (RNN). Our experiments focus on unconditional natural speech generation as well as handwriting generation. We show that the introduction of latent random variables can provide significant improvements in modelling highly structured sequences such as natural speech sequences. We empirically show that the inclusion of randomness into high-level latent space can enable the VRNN to model natural speech sequences with a simple Gaussian distribution as the output function. However, the standard RNN model using the same output function fails to generate reasonable samples. An RNN-based model using more powerful output function such as a GMM can

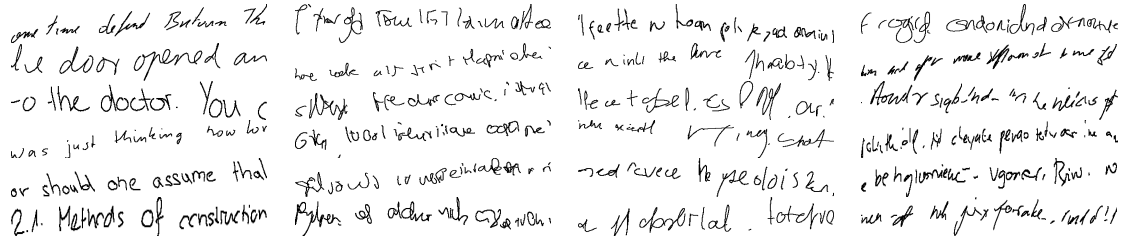


(a) Ground Truth (b) RNN-GMM (c) VRNN-Gauss

Figure 2.3 – Examples from the training set and generated samples from RNN-GMM and VRNN-Gauss. Top three rows show the global waveforms while the bottom three rows show more zoomed-in waveforms. Samples from (b) RNN-GMM contain high-frequency noise, and samples from (c) VRNN-Gauss have less noise. We exclude RNN-Gauss, because the samples are almost close to pure noise.

generate much better samples, but they contain a large amount of high-frequency noise compared to the samples generated by the VRNN-based models.

We also show the importance of temporal conditioning of the latent random variables by reporting higher log-likelihood numbers on modelling natural speech sequences. In handwriting generation, the VRNN model is able to model the diversity across examples while maintaining consistent writing style over the course of generation.



(a) Ground Truth (b) RNN-Gauss (c) RNN-GMM (d) VRNN-GMM

Figure 2.4 – Handwriting samples: (a) training examples and unconditionally generated handwriting from (b) RNN-Gauss, (c) RNN-GMM and (d) VRNN-GMM. The VRNN-GMM retains the writing style from beginning to end while RNN-Gauss and RNN-GMM tend to change the writing style during the generation process. This is possibly because the sequential latent random variables can guide the model to generate each sample with a consistent writing style.

Acknowledgments

The authors would like to thank the developers of Theano (Bastien et al., 2012). Also, the authors thank Kyunghyun Cho, Kelvin Xu and Sungjin Ahn for insightful comments and discussion. We acknowledge the support of the following agencies for research funding and computing support: Ubisoft, Nuance Foundation, NSERC, Calcul Québec, Compute Canada, the Canada Research Chairs and CIFAR.

3 ReNet: A Recurrent Neural Network Based Alternative to Convolutional Networks

3.1 Prologue to the Article

ReNet: A Recurrent Neural Network Based Alternative to Convolutional Networks. Francesco Visin, Kyle Kastner, Kyunghyun Cho, Matteo Matteucci, Aaron Courville, and Yoshua Bengio.

Published in Deep Learning Workshop, International Conference on Machine Learning (ICML) 2015

Personal Contribution. The core idea of using recurrent neural networks in a way to replace convolutional networks was developed by Kyunghyun Cho. A prototype to the model shown in the following paper resulted after some discussion between he and I. Kyunghyun wrote the initial implementation in Theano (Bergstra et al., 2010a; Bastien et al., 2012). After some time and tweaks by Kyunghyun, the model was officially working. After I had run several preliminary experiments, Francesco came onto the project. Francesco took ownership of the implementation, finding and fixing issues and making improvements to the architecture. Francesco spearheaded the research to its successful conclusion including many experiment runs. I continued executing experiments and developed a data pipeline for the Street View House Numbers experiment. Kyunghyun, Francesco, Aaron, and I all discussed hyperparameters, model tweaks, and experiment ideas. I contributed pieces (primarily in the data and experimental sections) of the paper along with rewrites and fixes to other sections written by Francesco, Kyunghyun, Aaron, and Yoshua. Matteo provided editing and review.

My contribution to this effort is summarized by my assistance with the initial concept, experimentation including data pipeline development, and content generation and review in the paper.

Affiliations

Francesco Visin, AIRLab, Dipartimento di Elettronica, Informazione e Bioingeg-

neria, Politecnico di Milano

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

Kyunghyun Cho, MILA, Département d'Informatique et de Recherche Opérationnelle, Université de Montréal

Matteo Matteucci, AIRLab, Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano

Aaron Courville: MILA, Département d'Informatique et de Recherche Opérationnelle, Université de Montréal

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

Funding We acknowledge the support of the following organizations for research funding and computing support: NSERC, Samsung, Calcul Quebec, Compute Canada, the Canada Research Chairs and CIFAR. Francesco Visin was funded by the AI*IA Young Researchers Mobility Grant and the Politecnico di Milano PHD School International Mobility Grant.

3.2 Abstract

In this paper, we propose a deep neural network architecture for object recognition based on recurrent neural networks. The proposed network, called ReNet, replaces the ubiquitous convolution+pooling layer of the deep convolutional neural network with four recurrent neural networks that sweep horizontally and vertically in both directions across the image. We evaluate the proposed ReNet on three widely-used benchmark datasets; MNIST, CIFAR-10 and SVHN. The result suggests that ReNet is a viable alternative to the deep convolutional neural network, and that further investigation is needed.

3.3 Introduction

Convolutional neural networks (CNN, Fukushima, 1980; LeCun et al., 1989) have become the method of choice for object recognition (see, e.g., Krizhevsky et al., 2012a). They have proved to be successful at a variety of benchmark problems including, but not limited to, handwritten digit recognition (see, e.g., Ciresan et al., 2012), natural image classification (see, e.g., Lin et al., 2014; Simonyan and Zisserman, 2015; Szegedy et al., 2014), house number recognition (see, e.g., Goodfellow et al., 2014), traffic sign recognition (see, e.g., Ciresan et al., 2012), as well as for speech recognition (see, e.g., Abdel-Hamid et al., 2012; Sainath et al., 2013; Tóth, 2014). Furthermore, image representations from CNNs trained to recognize objects on a large set of more than one million images (Simonyan and Zisserman, 2015; Szegedy et al., 2014) have been found to be extremely helpful in performing other computer vision tasks such as image caption generation (see, e.g., Vinyals et al., 2014; Xu et al., 2015), video description generation (see, e.g., Yao et al., 2015) and object localization/detection (see, e.g., Sermanet et al., 2014).

While the CNN has been especially successful in computer vision, recurrent neural networks (RNN) have become the method of choice for modeling sequential data, such as text and sound. Natural language processing (NLP) applications include language modeling (see, e.g., Mikolov, 2012), and machine translation (Sutskever et al., 2014; Cho et al., 2014; Bahdanau et al., 2014). Other popular areas of application include offline handwriting recognition/generation (Graves and Schmidhuber, 2009; Graves et al., 2008; Graves, 2013) and speech recognition (Chorowski et al., 2014; Graves and Jaitly, 2014). RNNs have also been used together with CNNs in speech recognition (Sainath et al., 2015). The recent revival of RNNs has largely been due to advances in learning algorithms (Pascanu et al., 2013; Martens and Sutskever, 2011) and model architectures (Pascanu et al., 2014; Hochreiter and Schmidhuber, 1997; Cho et al., 2014).

The architecture proposed here is related and inspired by this earlier work, but our model relies on purely uni-dimensional RNNs coupled in a novel way, rather than on a multi-dimensional RNN. The basic idea behind the proposed ReNet architecture is to replace each convolutional layer (with convolution+pooling making up a layer) in the CNN with four RNNs that sweep over lower-layer features in different directions: (1) bottom to top, (2) top to bottom, (3) left to right and (4)

right to left. The recurrent layer ensures that each feature activation in its output is an activation at the specific location *with respect to the whole image*, in contrast to the usual convolution+pooling layer which only has a local context window. The lowest layer of the model sweeps over the input image, with subsequent layers operating on extracted representations from the layer below, forming a hierarchical representation of the input.

Graves and Schmidhuber (2009) have demonstrated an RNN-based object recognition system for offline Arabic handwriting recognition. The main difference between ReNet and the model of Graves and Schmidhuber (2009) is that we use the usual sequence RNN, instead of the multidimensional RNN. We make the latter two parts of a single layer, usually (horizontal) RNNs or one (horizontal) bidirectional RNN, work on the hidden states computed by the first two (vertical) RNNs, or one (vertical) bidirectional RNN. This allows us to use a plain RNN, instead of the more complex multidimensional RNN, while making each output activation of the layer be computed with respect to the whole input image.

One important consequence of the proposed approach compared to the multidimensional RNN is that the number of RNNs at each layer scales now linearly with respect to the number of dimensions d of the input image ($2d$). A multidimensional RNN, on the other hand, requires the exponential number of RNNs at each layer (2^d). Furthermore, the proposed variant is more easily parallelizable, as each RNN is dependent only along a horizontal or vertical sequence of patches. This architectural distinction results in our model being much more amenable to distributed computing than that of Graves and Schmidhuber (2009).

In this work, we test the proposed ReNet on several widely used object recognition benchmarks, namely MNIST (LeCun et al., 1999), CIFAR-10 (Krizhevsky and Hinton, 2009) and SVHN (Netzer et al., 2011). Our experiments reveal that the model performs comparably to convolutional neural networks on all these datasets, suggesting the potential of RNNs as a competitive alternative to CNNs for image related tasks.

3.4 Model Description

Let us denote by $X = \{x_{i,j}\}$ the input image or the feature map from the layer below, where $X \in \mathbb{R}^{w \times h \times c}$ with w, h and c the width, height and number of channels, or the feature dimensionality, respectively. Given a receptive field (or patch) size of $w_p \times h_p$, we split the input image X into a set of $I \times J$ (non-overlapping) patches $P = \{p_{i,j}\}$, where $I = \frac{w}{w_p}$, $J = \frac{h}{h_p}$ and $p_{i,j} \in \mathbb{R}^{w_p \times h_p \times c}$ is the (i,j)-th patch of the input image. The first index i is the horizontal index and the other index j is the vertical index.

First, we sweep the image vertically with two RNNs, with one RNN working in a bottom-up direction and the other working in a top-down direction. Each RNN takes as an input one (flattened) patch at a time and updates its hidden state, working *along each column* j of the split input image X.

$$v^F_{i,j} = f_{\text{VFWD}}(z^F_{i,j-1}, p_{i,j}), \text{ for } j = 1, \cdots, J \tag{3.1}$$

$$v^R_{i,j} = f_{\text{VREV}}(z^R_{i,j+1}, p_{i,j}), \text{ for } j = J, \cdots, 1 \tag{3.2}$$

Note that f_{VFWD} and f_{VREV} return the activation of the recurrent hidden state, and may be implemented either as a simple tanh layer, as a gated recurrent layer (Cho et al., 2014) or as a long short-term memory layer (Hochreiter and Schmidhuber, 1997).

After this vertical, bidirectional sweep, we concatenate the intermediate hidden states $v^F_{i,j}$ and $v^R_{i,j}$ at each location (i,j) to get a composite feature map $V = \{v_{i,j}\}^{j=1,\ldots,J}_{i=1,\ldots,I}$, where $v_{i,j} \in \mathbb{R}^{2d}$ and d is the number of recurrent units. Each $v_{i,j}$ is now the activation of a feature detector at the location (i,j) with respect to all the patches in the j-th column of the original input ($p_{i,j}$ for all i).

Next we sweep over the obtained feature map V horizontally with two RNNs (f_{HFWD} and f_{HREV}). In a similar manner as the vertical sweep, these RNNs work along each row of V resulting in the output feature map $H = \{h_{i,j}\}$, where $h_{i,j} \in \mathbb{R}^{2d}$. Now, each vector $h_{i,j}$ represents the features of the original image patch $p_{i,j}$ *in the context of the whole image.*

Let us denote by ϕ the function from the input image map of X to the output feature map H (see Fig. 3.1 for a graphical illustration.) Clearly, we can stack

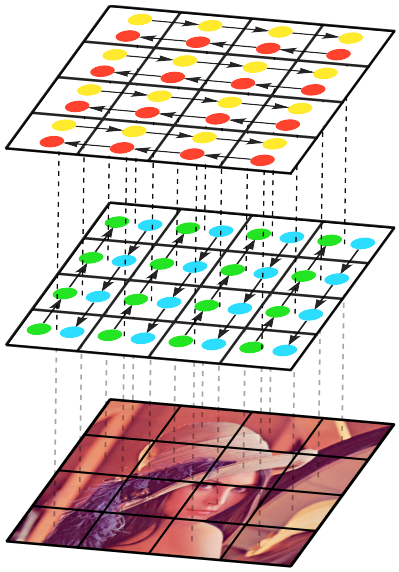


Figure 3.1 – A one-layer ReNet

multiple ϕ's to make the proposed ReNet deeper and capture increasingly complex features of the input image. After any number of recurrent layers are applied to an input image, the activation at the last recurrent layer may be flattened and fed into a differentiable classifier. In our experiments we used several fully-connected layers followed by a softmax classifier (as shown in Fig. 3.2).

The deep ReNet is a smooth, continuous function, and the parameters (those from the RNNs as well as from the fully-connected layers) can be estimated by the stochastic gradient descent algorithm with the gradient computed by backpropagation algorithm (see, e.g., Rumelhart et al., 1986) to maximize the log-likelihood.

3.5 Differences between LeNet and ReNet

There are many similarities and differences between the proposed ReNet and a convolutional neural network. In this section we use LeNet to refer to the canonical convolutional neural network as shown by LeCun et al. (1989). Here we highlight a few key points of comparison between ReNet and LeNet.

At each layer, both networks apply the same set of filters to patches of the input image or of the feature map from the layer below. ReNet, however, prop-

agates information through lateral connections that span across the whole image, while LeNet exploits local information only. The lateral connections should help extract a more compact feature representation of the input image at each layer, which can be accomplished by the lateral connections removing/resolving redundant features at different locations of the image. This should allow ReNet resolve small displacements of features across multiple consecutive patches.

LeNet max-pools the activations of each filter over a small region to achieve local translation invariance. In contrast, the proposed ReNet does not use any pooling due to the existence of learned lateral connections. The lateral connection in ReNet can emulate the local competition among features induced by the max-pooling in LeNet. This does not mean that it is not possible to use max-pooling in ReNet. The use of max-pooling in the ReNet could be helpful in reducing the dimensionality of the feature map, resulting in lower computational cost.

Max-pooling as used in LeNet may prove problematic when building a convolutional autoencoder whose decoder is an inverse [1] of LeNet, as the max operator is not invertible. The proposed ReNet is end-to-end smooth and differentiable, making it more suited to be used as a decoder in the autoencoder or any of its probabilistic variants (see, e.g., Kingma and Welling, 2014).

In some sense, each layer of the ReNet can be considered as a variant of a usual convolution+pooling layer, where pooling is replaced with lateral connections, and convolution is done without any overlap. Similarly, Springenberg et al. (2014) recently proposed a variant of a usual LeNet which does not use any pooling. They used convolution with a larger stride to compensate for the lack of dimensionality reduction by pooling at each layer. However, this approach still differs from the proposed ReNet in the sense that each feature activation at a layer is only with respect to a subset of the input image rather than the whole input image.

The main disadvantage of ReNet is that it is not easily parallelizable, due to the sequential nature of the recurrent neural network (RNN). LeNet, on the other hand, is highly parallelizable due to the independence of computing activations at each layer. The introduction of sequential, lateral connections, however, may result in more efficient parametrization, requiring a smaller number of parameters with overall fewer computations, although this needs to be further explored. We note that this limitation on parallelization applies only to model parallelism, and any

1. All the forward arrows from the input to the output in the original LeNet are reversed.

technique for data parallelism may be used for both the proposed ReNet and the LeNet.

3.6 Experiments

3.6.1 Datasets

We evaluated the proposed ReNet on three widely-used benchmark datasets; MNIST, CIFAR-10 and the Street View Housing Numbers (SVHN). In this section we describe each dataset in detail.

MNIST The MNIST dataset (LeCun et al., 1999) consists of 70,000 handwritten digits from 0 to 9, centered on a 28×28 square canvas. Each pixel represents the grayscale in the range of $[0, 255]$. [2] We split the dataset into 50,000 training samples, 10,000 validation samples and 10,000 test samples, following the standard split.

CIFAR-10 The CIFAR-10 dataset (Krizhevsky and Hinton, 2009) is a curated subset of the 80 million tiny images dataset, originally released by Torralba et al. (2008). CIFAR-10 contains 60,000 images each of which belongs to one of ten categories; airplane, automobile, bird, cat, deer, dog, frog, horse, ship and truck. Each image is 32 pixels wide and 32 pixels high with 3 color channels (red, green and blue.) Following the standard procedure, we split the dataset into 40,000 training, 10,000 validation and 10,000 test samples. We applied zero-phase component analysis (ZCA) and normalized each pixel to have zero-mean and unit-variance across the training samples, as suggested by Krizhevsky and Hinton (2009).

Street View House Numbers The Street View House Numbers (SVHN) dataset (Netzer et al., 2011) consists of cropped images representing house numbers captured by Google StreetView vehicles as a part of the Google Maps mapping process. These images consist of digits 0 through 9 with values in the range of [0, 255] in each of 3 red-green-blue color channels. Each image is 32 pixels wide and 32 pixels high giving a sample dimensionality (32, 32, 3). The number of samples we

2. We scaled each pixel by dividing it with 255.

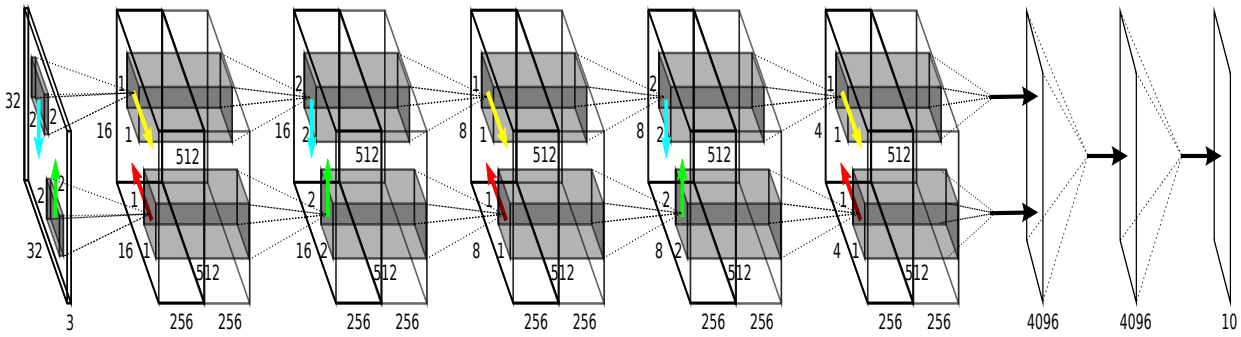


Figure 3.2 – The ReNet network used for SVHN classification

used for training, valid, and test sets is 543,949, 60,439, and 26,032 respectively. We normalized each pixel to have zero-mean and unit-variance across the training samples.

Data Augmentation

It has been known that augmenting training data often leads to better generalization (see, e.g., Krizhevsky et al., 2012a). We decided to employ two primary data augmentations in the following experiments: *flipping* and *shifting*.

For flipping, we either flipped each sample horizontally with 25% chance, flipped it vertically with 25% chance, or left it unchanged. This allows lets the model observe "mirror images" of the original image during training. In the case of shifting, we either shifted the image by 2 pixels to the left (25% chance), 2 pixels to the right (25% chance) or left it as it was. After this first processing, we further either shifted it by 2 pixels to the top (25% chance), 2 pixels to the bottom (25% chance) or left it as it was. This two-step procedure makes the model more robust to slight shifting of an object in the image. The shifting was done without padding the borders of the image, preserving the original size but dropping the pixels which are shifted out of the input while shifting in zeros.

The choice of whether to apply these augmentation procedures on each dataset was chosen on a per-case basis in order to maximize validation performance.

3.6.2 Model Architectures

Gated Recurrent Units Gated recurrent units (GRU, Cho et al., 2014) and long short-term memory units (LSTM, Hochreiter and Schmidhuber, 1997) have

been successful in many applications using recurrent neural networks (see, e.g., Cho et al., 2014; Sutskever et al., 2014; Xu et al., 2015). To show that the ReNet model performs well independently of the specific implementation of the recurrent units, we decided to use the GRU on MNIST and CIFAR-10, with LSTM units on SVHN.

The hidden state of the GRU at time t is computed by

$$h_t = (1 - u_t) \odot h_{t-1} + u_t \odot \tilde{h}_t,$$

where

$$\tilde{h}_t = \tanh\left(W x_t + U(r_t \odot h_{t-1}) + b\right)$$

and

$$[u_t; r_t] = \sigma\left(W_g x_t + U_g h_{t-1} + b_g\right).$$

For more details on the LSTM unit, as well as for an in-depth comparison among different recurrent units, we refer the reader to (Chung et al., 2015).

General Architecture The principal parameters that define the architecture of the proposed ReNet are the number of ReNet layers (N_{RE}), their corresponding receptive field sizes ($w_p \times h_p$) and feature dimensionality (d_{RE}), the number of fully-connected layers (N_{FC}) and their corresponding numbers (d_{FC}) and types (f_{FC}) of hidden units.

In this introductory work, we did not focus on extensive hyperparameter search to find the optimal validation set performance. We chose instead to focus the experiments on a small set of hyperparameters, with the only aim to show the potential of the proposed model. Refer to Table 3.1 for a summary of the settings that performed best on the validation set of the studied datasets and to Fig. 3.2 for a graphical illustration of the model we selected for SVHN.

3.6.3 Training

To train the networks we used a recently proposed adaptive learning rate algorithm, called Adam (Kingma and Ba, 2014). In order to reduce overfitting we

	MNIST	CIFAR-10	SVHN
N_{RE}	2	3	3
$w_p \times h_p$	$[2 \times 2]$–$[2 \times 2]$	$[2 \times 2]$–$[2 \times 2]$–$[2 \times 2]$	$[2 \times 2]$–$[2 \times 2]$–$[2 \times 2]$
d_{RE}	256–256	320–320–320	256–256–256
N_{FC}	2	1	2
d_{FC}	4096–4096	4096	4096–4096
f_{FC}	$\max(0, x)$	$\max(0, x)$	$\max(0, x)$
Flipping	no	yes	no
Shifting	yes	yes	yes

Table 3.1 – Model architectures used in the experiments. Each row shows respectively the number of ReNet layers, the size of the patches, the number of neurons of each ReNet layer, the number of fully connected layers, the number of neurons of the fully connected layers, their activation function and the data augmentation procedure employed.

applied dropout (Srivastava et al., 2014) after each layer, including both the proposed ReNet layer (after the horizontal and vertical sweeps) and the fully-connected layers. The input was also corrupted by masking out each variable with probability 0.2. Finally, each optimization run was early stopped based on validation error.

Note that unlike many previous works, we did not retrain the model (selected based on the validation performance) using both the training and validation samples. This experiment design choice is consistent with our declared goal to show a proof of concept rather than stressing absolute performance. There are many potential areas of exploration for future work.

3.7 Results and Analysis

In Table 3.2, we present the results on three datasets, along with previously reported results.

It is clear that the proposed ReNet performs comparably to deep convolutional neural networks which are the *de facto* standard for object recognition. This suggests that the proposed ReNet is a viable alternative to convolutional neural networks (CNN), even on tasks where CNNs have historically dominated. However, it is important to notice that the proposed ReNet does not outperform state-of-the-art convolutional neural networks on any of the three benchmark datasets, which

Test Error	Model
0.28%	(Wan et al., 2013)⋆
0.31%	(Graham, 2014b)⋆
0.35%	(Ciresan et al., 2010)
0.39%	(Mairal et al., 2014)⋆
0.39%	(Lee et al., 2014)⋆
0.4%	(Simard et al., 2003)⋆
0.44%	(Graham, 2014a)⋆
0.45%	(Goodfellow et al., 2013)⋆
0.45%	**ReNet**
0.47%	(Lin et al., 2014)⋆
0.52%	(Azzopardi and Petkov, 2013)

(a) MNIST

Test Error	Model
4.5%	(Graham, 2014a)⋆
6.28%	(Graham, 2014b)⋆
8.8%	(Lin et al., 2014)⋆
9.35%	(Goodfellow et al., 2013)⋆
9.39%	(Springenberg and Riedmiller, 2013)⋆
9.5%	(Snoek et al., 2012)⋆
11%	(Krizhevsky et al., 2012a)⋆
11.10%	(Wan et al., 2013)⋆
12.35%	**ReNet**
15.13%	(Zeiler and Fergus, 2013)⋆
15.6%	(Hinton et al., 2012)⋆

(b) CIFAR-10

Test Error	Model
1.92%	(Lee et al., 2014)⋆
2.23%	(Wan et al., 2013)⋆
2.35%	(Lin et al., 2014)⋆
2.38%	**ReNet**
2.47%	(Goodfellow et al., 2013)⋆
2.8%	(Zeiler and Fergus, 2013)⋆

(c) SVHN

Table 3.2 – Generalization errors obtained by the proposed ReNet along with those reported by previous works on each of the three datasets. ⋆ denotes a convolutional neural network. We only list the results reported by a single model, i.e., no ensembling of multiple models. In the case of SVHN, we report results from models trained on the Format 2 (cropped digit) dataset only.

calls for more research in the future.

3.8 Discussion

Choice of Recurrent Units Note that the proposed architecture is independent of the chosen recurrent units. We observed in preliminary experiments that gated recurrent units, either the GRU or the LSTM, outperform a usual sigmoidal unit (affine transformation followed by an element-wise sigmoid function.) This indirectly confirms that the model utilizes long-term dependencies across an input image, and the gated recurrent units help capture these dependencies.

Analysis of the Trained ReNet In this paper, we evaluated the proposed ReNet only quantitatively. However, the accuracies on the test sets do not reveal

what kind of image structures the ReNet has captured in order to perform object recognition. Due to the large differences between ReNet and LeNet discussed in Sec. 3.5, we expect that the internal behavior of ReNet will differ from that of LeNet significantly. Further investigation along the line of (Zeiler and Fergus, 2014) will be needed, as well exploring ensembles which combine RNNs and CNNs for bagged prediction.

Computationally Efficient Implementation As discussed in Sec. 3.5, the proposed ReNet is less parallelizable due to the sequential nature of the recurrent neural network (RNN). Although this sequential nature cannot be addressed directly, our construction of ReNet allows the forward and backward RNNs to be run independently from each other, which allows for parallel computation. Furthermore, we can use many parallelization tricks widely used for training convolutional neural networks such as parallelizing fully-connected layers (Krizhevsky, 2014), having separate sets of kernels/features in different processors (Krizhevsky et al., 2012a) and exploiting data parallelism.

Acknowledgments

The authors would like to thank the developers of Theano (Bergstra et al., 2010b; Bastien et al., 2012). We acknowledge the support of the following organizations for research funding and computing support: NSERC, Samsung, Calcul Québec, Compute Canada, the Canada Research Chairs and CIFAR. F.V. was funded by the AI*IA Young Researchers Mobility Grant and the Politecnico di Milano PHD School International Mobility Grant.

4 Experiments in Audio Sequence Generation

4.1 Introduction

Recurrent neural networks have proven adept at modeling sequences of data, excelling in sequential applications such as machine translation, chemical prediction, and dialogue modeling. Given recent successes with this *symbolic* data, especially in the text domain, it is natural to translate these techniques to similar fields. This chapter documents our significant effort in audio sequence modeling. We experiment both on symbolic representations such as computer music and on recorded audio such as human speech. Sections of this work were performed in consultation with collaborators at MILA, IBM Research, and Google Brain though all of it remains unpublished before this thesis.

4.2 Basic Musical Concepts

In this section we provide a light introduction to the concepts needed to understand music as it relates to our modeling research. We attempt to avoid discussion of minutiae unrelated to the task of understanding music using statical models. We also avoid discussion of eastern music here, as many affectations and assumptions made in music modeling are only appropriate for pieces in the western music tradition. The application of many techniques in this thesis to eastern music should be straightforward, but core assumptions (such as the 12 tone scale which will be discussed below), may need to be reconsidered due to base differences in music composition. For a more complete introduction to music theory, one should consult Feynman Liang's thesis (Liang, 2016). Computer music specifically has many additional concepts related to the mechanics of actually producing musical sounds

which depend on their representation, which is not discussed here. Within this chapter, we refer to individual musical samples as scores, pieces, or examples.

4.2.1 Notes

Notes are the events used to make music. Combined with durations, these specify the sequence of events that actually create a piece of music. Unlike "raw" sounds, we simplify musical notes to symbolic representations which signify action on an instrument such as pressing a key on a piano, hitting a drum, or strumming a guitar. These actions may produce *complex* sounds in a recording, but in symbolic representations they are a single, simple event. We also count *not* playing anything (the rest) as another type of note.

In western music there is a common assumption that symbols represent one of 12 notes: C, C sharp (also called D flat), D, E flat (also called D sharp), E, F, F sharp (also called G flat), G, A flat (also called G sharp), A, B flat (also called A sharp), or B. Each "step" up this scale indicates a frequency multiplication of $\sqrt[12]{2}$. This means that taking 12 steps upward will result in arriving back at a note with the same name, but with double the frequency. Steps downward indicate a division by $\sqrt[12]{2}$, meaning 12 steps downward would arrive at a note of with the same name, but with half the frequency. This system of exact relationships between notes is known as "equal temperament". Incidentally, equal temperament seems to have been developed independently in the 1600s in both the East and the West by Simon Stevin and Zho Zaiyu, respectively (Cho, Cho).

4.2.2 Octaves

As notes are repeated by halving or doubling frequency, it is convenient to group the notes in between into so called octaves. Higher octaves represent higher frequency sounds and lower octaves indicate lower frequency sounds. For example A4 would indicate the note A, played in the 4th octave which matches to an approximate base frequency of 440 Hz. A3 would indicate the note A played in the 3rd octave, with a base frequency of 220 Hz. This cyclic relationship forms the core of how notes interact with one another. The ways in which different sounds clash or unite can often be related to their frequency content. Many instruments and voices have an octave range in which they operate (such as A2-C4), which is

further related to its common function within music. All of these subtleties, along with the style when notes are played by different instruments (called *timbre*), are taken into consideration by expert composers. Many of the intricacies of live performance are lost when translating to symbolic notation, but one of the long term goals of computer music research is to more faithfully understand the nuances of human performance, and how it differs from the written page.

4.2.3 Instrumentation

The type of instrument used to play the notes on the page can have a massive impact on how the music sounds. Each instrument has its own unique sound, and many composers use instrumentation to create specific moods and feelings. Accurately modeling the sound and response of an instrument using computers is an open research problem, but is not discussed further here. In this work we rely on existing synthesizers to turn symbolic notes into accurate acoustics.

4.2.4 Tempo and Duration

Tempo indicates how fast or slow a piece of music should be played, as an integer number of beats per minute, or BPM. It is usually accompanied by a time signature, a notation that indicates how long (measured in *beats*) each musician should count before moving to the next short section of music, called a measure. These two indicators also indirectly hint at style and flair for a given piece. There are a number of common note durations, such as the sixteenth note, eighth note, quarter note, half note, and whole note. These names indicate that there are respectively 16, 8, 4, 2, or 1 note in a measure in $\frac{4}{4}$ time. This implicitly gives relative timing, which combined with the tempo BPM allows musicians to stay in sync while playing different parts. Time signatures are given as fractions, where the bottom value indicates the unit which represents one beat, and the top value indicates the number of beats in a measure. Common tempos range from 100 to 140 BPM, and common time signatures include $\frac{4}{4}$, $\frac{3}{4}$, and $\frac{6}{8}$. In addition to these "simple" durations, there are a host of accents which slightly lengthen or shorten the expected duration of a note.

4.2.5 Key

Analyzing the collection of pitches which make up a musical piece often reveals additional structure based on the notes used during arrangement. A *key* is a set of notes which commonly occur together in many different pieces of music. Sad or dramatic music is often written using minor keys, and happy music often uses major keys. Discussion of different types of note groups, or *scales*, and how they are used to imply key is far beyond the scope of this thesis, but it is important to understand that using subtle variations in notes can give vastly different feelings to the music. Indeed, the style of many composers is centered around their preferences for certain keys and scales. Coupled with different instrumentation, small changes in key can make the same written piece unrecognizable to the listener. This an *extremely* coarse view of key, but the note center and scale type (such as D minor or E major) can give musicians a lot of information about the general feel and style of a piece.

When attempting to generate music, one can reduce the variability of key through pre-processing. This is often achieved by transposing or pitch centering each piece to a common key of C (generally preserving major or minor flavor). Musically this makes sense, as composers often change the key of a piece depending on instrumentation or a singer's natural range. In addition when synthesizing our generated music, we generally fix the instrument used to play the piece, in order to more clearly hear what modeling errors exist without covering them using varied instruments. Despite this, key and which instruments play the notes are generally the most recognizable aspects of a piece for the average music listener.

4.2.6 Polyphony

The concept of polyphony is of crucial importance to music modeling. Notably, in a musical piece there are usually multiple instruments playing different parts which interact with one another to form the overall musical score. This presents a difficulty in modeling, as this is a *structured prediction* problem. We can avoid the complexity of polyphony by picking only specific instruments with single note sequences, or by choosing *monophonic* pieces, which were only written with one note played at a time. Many instruments such as piano and guitar have the ability to play multiple simultaneous notes in and of themselves, which means even con-

centrating on individual instruments may not avoid the problem of modeling the relationships between notes. Modeling of polyphonic music is a key research area, but even monophonic modeling is far from solved.

4.2.7 Genre

All of the above musical features combine to form the style or *genre* of the music. Genres such as classical, pop, country, rock, and jazz, are often used to define what music listeners enjoy, but the boundaries between these classes of music are nebulous at best. Music lovers argue constantly over details of genre and how different music should be classified, but it is clear that different styles of music have vastly different norms for different musical concepts. The landscape of "genre classes" is constantly shifting and changing, and classifying any music into a particular genre is largely a personal choice. In fact music composers often strive to create new and interesting music which blends, melds, or twists common tropes of a given genre in order to create new listening experiences. As such, this research will largely avoid dealing with genre directly, although understanding genre and how to adapt generated content or suggestions to user preferences will be crucial to production, music modeling, and generation.

4.3 Musical Formats

4.3.1 Sheet Music

The gold standard for representing music in an interchangeable format is *sheet music*, with the largest corpus of information available. The term "sheet music" covers a huge swath of different styles of notation and writing that have changed through the course of history. Unfortunately most algorithms need to convert the sheet music meant for human musicians to read into a more computer friendly format.

4.3.2 ABC Notation

ABC notation (Usergroup, Usergroup) (or ABC for short) is a text based representation for music, which is highly variable and can capture many of the musical parts of a song. The format itself is almost directly human readable, and shares many similarities with sheet music. Within the ABC standard there are many different features, and different software may add or remove custom features depending on implementation. Highlights of ABC include dedicated symbols for repetition, common note movements, tempo, and ability to place metadata such as piece name, lyrical content, and authorship. Being text based, it is also directly amendable to typical text based approaches for neural networks and statistical modeling.

4.3.3 MusicXML

MusicXML (Good, 2001) is a specification for music that uses the common XML interchange format. This representation uses specific object types which represent common music concepts such as tempos and notes. With MusicXML, it is possible to encode and decode musical scores losslessly between different programs and hardware. There are a number of packages which work with this format in the Python programming language such as music21 (Cuthbert and Ariza, 2010).

4.3.4 Musical Instrument Digital Interface

Musical Instrument Digital Interface (MIDI) (Association, 2008) is a flexible, open ended event sequence format that has been commonly used for musical equipment, lights, and video since the early 1980s. The simplest modes of MIDI utilize only an event channel, event class (generally pitch), and an associated duration for the event class for music representation. There are also a wide variety of modifications to MIDI made by various hardware and software vendors for real-time processing or more advanced messages. In general, MIDI is the default method for sharing symbolic music with computers, though the other formats listed here are also popular. The MIDI format seems deceptively simple, but has a wide array of modifications, customizations, and improvements depending on which parser and platform are used for processing the MIDI data. Given a robust MIDI encoding and

decoding library (such as pretty_midi (Raffel and Ellis, 2014) or music21 (Cuthbert and Ariza, 2010), most musical tasks become straightforward.

4.3.5 Piano Roll

Piano roll is close to sheet music in many ways, but it is simplified through quantization so that it is more useful for modelling. By representing individual notes as repeated events based on the quantization granularity, music can be simplified in memory for algorithmic processing. As an example, quantization at the 16th note boundary would mean 16th notes get one entry in an array, 8th notes get two, quarter notes get four, and so on. This quantization removes the ability to model very fine grained performance timing and also adds a layer of complexity in deciding what quantization level is sufficient to represent a certain song, but the tradeoff is added convenience when working in computer programs. This tradeoff is more appealing by the ease in which piano roll can be converted to and from the more representative MIDI. The simplicity of piano roll makes it the defacto choice for many existing music models.

4.3.6 Custom

In addition to these common formats, there exist a huge number of custom formats for music for both digital distribution and printed reference. These formats vary based on instrument, intended usage, and user preference but generally these formats also have tools to convert to and from a more generic representation such as MIDI or MusicXML.

4.4 Music Generation

4.4.1 Prior Work

Scientists and musicians alike have long been interested in learning what makes music sound appealing and how to generate it algorithmically. Cope (1991) utilized expert systems and rule based composition to produce new works in the styles of

several classical composers. His work made a huge impact on the field of computational creativity, and set a high water mark for subsequent approaches. Eck and Schmidhuber (2002) used LSTM networks to learn accompaniment for blues music, and was a direct precursor to many of the approaches and methods used in this thesis. Boulanger-Lewandowski et al. (2012b) used the combination of an RNN and a restricted Boltzmann machine (RBM) to model polyphonic music on several datasets which have become standard benchmarks for many RNN approaches. In Goel and Vohra (2014), Goel extends this work to LSTM networks and deep belief networks (DBN) with improved qualitative and quantitative results. Sturm et al. (2015) models monophonic ABC format music using LSTM language models, and has even begun to perform these new pieces with live musicians. Liang (2016) uses LSTM networks for both generation and harmonization as well as user studies with a musical "Turing test". Zimmerman (2016), Johnson (2015), and Walder (2016) use a combination of musical preprocessing and RNNs to create compelling generative networks for polyphonic music. Colombo et al. (2016) uses a very similar approach to the one taken here to predict monophonic melodies.

Hadjeres et al. (2016), Pachet et al. (2013), and Pachet et al. (2011) represent a set of alternative approaches to music generation which are published primarily by the Sony Computer Science Laboratory in Paris. Considering style, harmonization, and other affectations as a constraint, they use techniques such as constraint programming, Markov Chain Monte Carlo (MCMC), and belief propagation to sample from these constrained models for generation and harmonization. Their results are impressive, and have recently been used to create an album of "AI pop" including songs such as "Daddy's Car" and "Mr. Shadow". In addition, these techniques have also been used for lyrical style transfer in Barbieri et al. (2012).

4.4.2 Markov chains for ABC Notation

Given the success of recurrent neural networks for modeling sequences (Gers, 2001), language (Józefowicz et al., 2016) and ABC (Sturm et al., 2015) notation, it is reasonable to wonder about other baselines for ABC tasks. The strength of simple Markov models when compared to RNNs for other text (Brown et al., 1992; Chelba et al., 2014) means that Markov chains for ABC notation should be an excellent baseline by which to judge other generative models. In this section we

provide details for generating music with markov chains using ABC notation.

$$p(x_{1...t}) = p(x_t|x_{t-1}) * p(x_{t-1}|x_{t-2}) \cdots p(x_1) \tag{4.1}$$

Markov chains model the joint distribution of a sequence $x_{1...t}$, given some conditional independence assumptions, namely the probability of step t depends only on step $t-1$ through $t-n$, where n represents the *order* of the Markov chain. As an example, an order 1 Markov chain would model the joint probability of a sequence $x_{1...t}$ as in Equation 4.1.

This means that similar to RNNs and CNNs, Markov chains also attempt to model the joint probability distribution of a sequence, but with strong assumptions of conditional independence and relationships within the data. The baseline models used in this thesis predict the next timestep by sampling from the probability distribution defined by normalized counts.

In particular, the simplicity of learning from normalized counts means that Markov chains train extremely quickly, as compared to more complicated models which generally take much longer for equivalent dataset sizes. The downside is that this model has a number of strong assumptions which may be violated by the actual relationships in the data. In addition it is unclear how to extend this model to the polyphonic case without heavy manipulation or preprocessing of the dataset.

One additional issue, common to Markov chain generation but also occurring in RNN generation, is *plagiarism* (Papadopoulos et al., 2016). Specifically, we consider a sequence generation plagiaristic if for some subsequence length l, the generated sequence $g_{1:l}$ occurs in its entirety as a subsequence of the training set. To counter this propensity for plagiarism, we can add a softmax function, allowing for adjustment of the softmax temperature which will increase the likelihood of moving to new areas of the probability space at the expense of possibly making transitions which are relatively less likely under the non-temperature adjusted model. One simple, though potentially slow, way to prevent plagiarism is to check the generations against the training corpus using brute force search. There are also more complex methods, such as Papadopoulos et al. (2016), which make guarantees against plagiarism in generation.

```
%
X:0003
T:The Lads of the Village.
M:2/4
L:1/8
Q:1/4=104
I: :: ::
%% D ^F G A B c d e ^f g a
Z:Jack Campin * www.campin.me.uk * 2009
K:G
G2g>d|ecBG|a>cBG|FA`FD|G2 gd|ecBG|A>cBG|(D/G/``F/A/) G2:|
g>fgd|ecBG|g>fge|a>gfd|g>fgd|ecBG|A>cBG|(D/G/)(F/A/) G2:|
```

Figure 4.1 – Example song from Aird's Airs in ABC format

Data

This model was trained on Aird's Airs, a collection of adaptations by James Aird, in turn compiled and digitized by Jack Campin (Sturm et al., 2015). This dataset consists of 1200 tunes over a variety of subgenres including jigs, waltzes, and marches. All songs are presented in ABC format to the Markov chain during training.

Results

An example piece from the dataset in ABC format can be seen in Figure 4.1, along with a sample generation in Figure 4.2. The generated ABC texts are synthesized using the abc2midi tool. abc2midi is an extremely robust converter, and automatically handles rejection of invalid files as well as correcting a number of common exceptions. This process of automatically curating means that the resulting midi files (when output is possible) are remarkably musical. We find Markov chains trained on ABC notation to be a strong baseline for monophonic music generation.

```
%
M:2/4
Q:1/2=90
I: : || :: (1) || (1) || (2) :: (1) :|
M:C
L:1/8
I: ||
Z:Jack Campin * www.campin.me.uk * 2009
K:G
(3d>ef|g2B>B B2Tc2|d2G>G G2
d2   =f>gaf |gabg  e4  |d4-  d4 ::
e4    |A<F E>D {D}E4  |!
AF   A2d| cdc   e2 dB|TA3 G E2 :|
```

Figure 4.2 – Example Markov chain generation (temperature 1000, Markov order 6) in ABC format

4.4.3 Density Estimation for Polyphonic Music using RNN-MADE

Fundamentally, the problem of monophonic music prediction can be seen as building a model of $p(n_1...n_T) = \prod_{t=1}^{T} p(n_t|n_{r\forall r\notin t})$, with note sequence n and timestep t, where each n_t in n refers to the pitch, and its duration. This prediction may also include extra contextual information, such as chords or genre information c, resulting in $p(n_1...n_T) = \prod_{t=1}^{T} p(n_t|n_{r\forall r\notin t}, c)$. One further simplification is to make the generative process directed, resulting in $p(n_1...n_T) = \prod_{t=1}^{T} p(n_t|n_{i<t})$. This formulation quite naturally fits with the probability factorization modeled by RNNs.

Extending this concept to the polyphonic case, we wish to model a collection of notes. For simplicity in description, we will assume there are always 4 notes per timestep and adopt a subscript set l, d, u, h to indicate the relative position of the note, from lowest pitch to highest pitch. With this notation in mind we now wish to model the sequence shown in Equation 4.2.

Making a similar assumption as in the monophonic case, we apply an *ordering over notes*, as well as a temporal ordering. This note ordering is arbitrary, but allows us to construct a model which has directed connections over time and pitch. For example, the ordering used in this work was lowest pitch to highest pitch, but different tasks or data may indicate a preferred ordering, based on the relative importance of each note within a timestep. The resulting model then becomes as

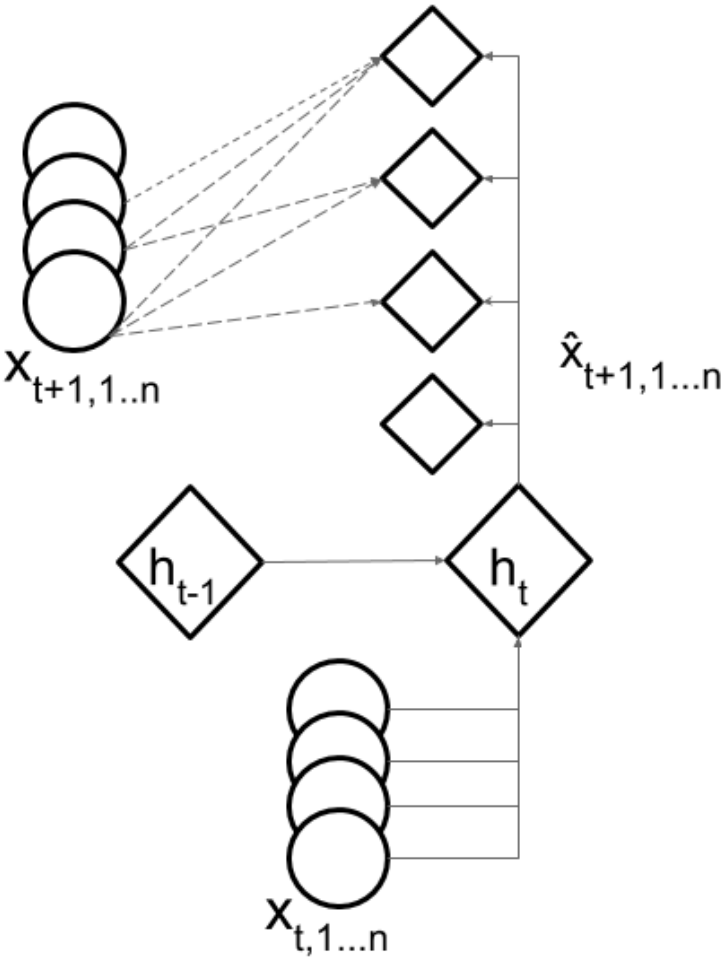


Figure 4.3 – Model diagram for one step in RNN-MADE

shown in Equation 4.3. Because each note n is in fact a pitch and accompanying duration, we assume that pitch and duration are independent at the current step, but conditional on all pitches and durations beforehand in time and note order. This can be seen in Figure 4.3.

$$p(n_{l1}, n_{d1}, n_{u1}, n_{h1}, n_{l2}...n_{hT}) = \prod_{t=1}^{T} \prod_{q \in l,d,u,h} p(n_{qt} | n_{wr \forall w,r \notin q,r}). \tag{4.2}$$

$$p(n_{l1}, n_{d1}, n_{u1}, n_{h1}, n_{l2}...n_{hT}) = \prod_{t=1}^{T} \prod_{q \in l,d,u,h} p(n_{qt} | n_{m<q,i<t}) \tag{4.3}$$

We can also use additional methods to improve training speed. Similar to the

use of teacher forcing to improve training stability of RNNs (Goodfellow et al., 2016), we can use the groundtruth values for notes and durations in the previous context window. In doing this, the vertical sequential dependency at each timestep is broken and each step of the vertical can be computed in parallel. This can be efficiently accomplished by careful masking of the target values, resulting in a structured polyphonic model which takes nearly the same amount of computation time as a monophonic model of the same length.

These simplifications combine to create a model which is straightforward to train and generate from, similar to the approaches proposed by NADE (Larochelle and Murray, 2011b), MADE (Germain et al., 2015), RNN-NADE (Boulanger-Lewandowski et al., 2012b), MDRNN (Graves et al., 2008), ReNet (Visin et al., 2015), pixelRNN (van den Oord et al., 2016), pixelCNN (van den Oord et al., 2016) and many other models for structured prediction. In this spirit, we refer to this model as *RNN-MADE*.

Data

The primary dataset used for this work is the Bach chorale corpus provided as part of the music21 Python package (Cuthbert and Ariza, 2010). This corpus consists of 426 compositions attributed to Johann Sebastian Bach, with typically 4 individual voices per piece. Removing a number of pieces with numbers of voices other than 4, and a few edge cases during parsing, our dataset consists of a final total of 357 pieces. Splitting this data into 90% training, 10% validation split for the purposes of early stopping (Bengio, 2013), we are left with a training set of 321 pieces, and 36 held out for model validation.

Unlike the piano roll representation described previously, we choose to use a composite event representation. The key benefit of this representation is that events are chosen through a small number of decisions, which means it is equally easy to choose a short duration event as a long one. Piano roll has the key downside that generating an event N times longer than the quantization resolution requires N consecutive predictions of the same note. For example, for a song quantized to 8th note resolution, predicting a quarter note (one of the most common notes in classical music) would require 8 consecutive, identical predictions. This is a difficult problem with sequential models, especially when sampling from distributions at every timestep.

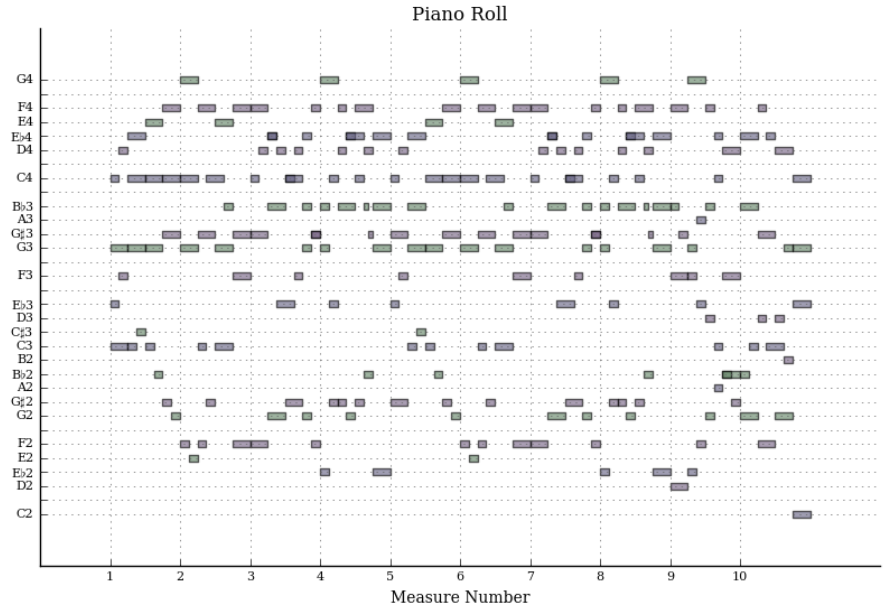


Figure 4.4 – Piano roll plot, training data

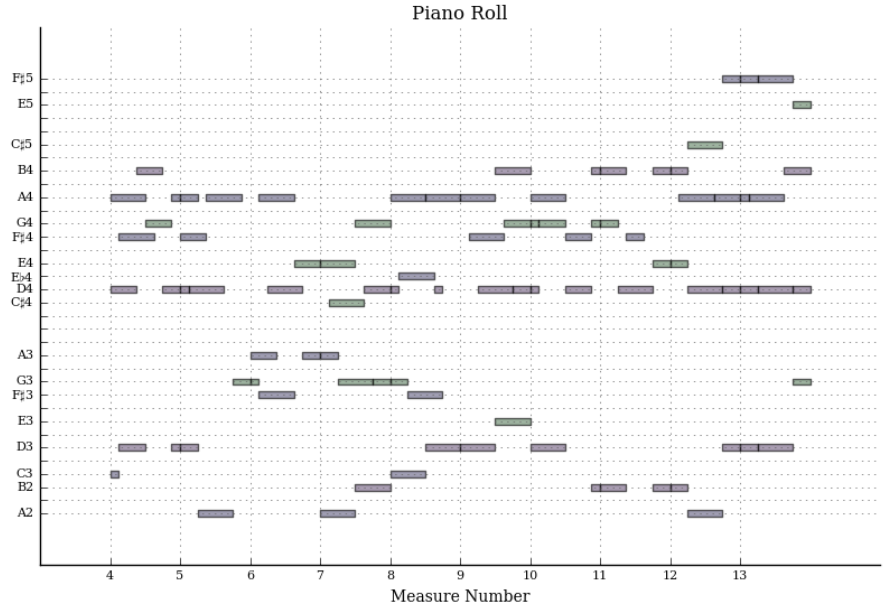


Figure 4.5 – Piano roll plot, generated data

An event based representation, by contrast, greatly reduces the sequence length in time, minimizing excess computation, reducing gradient vanishing or exploding problems with RNN based models, and avoiding decisions during times when the only "action" is continuing what previously occurred. This may not be appropriate for all problems, but for music this choice is natural and corresponds closely with the sheet music and MIDI representations ubiquitous in these tasks. We refer to this note and duration representation as "event sequence representation".

For an example of the potential advantages of this representation, see Figure 4.4. For example, the Piano roll notation of this data at 16th note quantization would require 160 timesteps, while the equivalent event representation would only require 77. The advantage of event representation grows as the length of the sequences increases, and most pieces are longer than the 10 measures plotted in Figure 4.4. The differences between the two representations also grows more pronounced when the average note duration is longer than two quantization steps. Figure 4.5, with its rapidly shifting parts represents something of a worst case for the event sequence representation and it *still* shows some advantage over piano roll. We will however continue to use the equivalent piano roll (or score notation) for plotting, keeping in mind the representation in memory is different than the plot itself. Colombo et al. (2016), which was developed independently and in parallel with our work, also describes a monophonic version of event representation.

Results

A modest, informal user study indicates that while this network appears to capture local correlations and generates interesting musical snippets, the overall pieces are clearly not comparable to the works of Bach. This is no surprise, as directed generative models often struggle with modeling long-range correlations without additional information or architecture modifications. Incorporating more global context using multi-scale methods (Chung et al., 2016) should help with this issue, but utilizing forward context while still having a valid generative model is an area of open research.

4.5 Audio Synthesis

We cover two broad approaches in this section for audio synthesis. One approach is to learn or set a fixed set of basis functions, or atoms, and then use the same techniques as discussed in the previous section for symbolic sequences. Another approach is to use a recurrent neural network to parameterize a probability density function such as in MDN of Bishop (1994) and VRNN which was first published in Chung et al. (2015) and discussed in Chapter 2.2. Because this density is dynamic over time, sampling from this dynamic density should allow us to generate new audio, or analyze the likelihood of a sequence under our learned density model.

For our experiments, we focus primarily on speech modeling due to the ubiquity of open data, and the existence of prior art to judge our own results. In general, working closer to raw timeseries should allow for generalizable insights to many timeseries problem. However, higher level representations, such as those presented in Section 4.7.2, may provide computational advantages or a better representation for learning.

4.6 Raw Audio Concepts

Taking a continuous signal, such as a sound wave traveling through a medium, and recording it in a computer readable format is no easy feat. As seen in Figure 4.6, signals have two primary components: *time*, and *amplitude*.

Fundamentally, the problem at hand is to record a signal that is continuous in time and amplitude, and store an approximation which is discrete in both. The mathematics and hardware behind this are beyond the scope of this thesis, but for algorithmic processing we *do* care about the granularity of discretization in time, called the *sampling rate*, and the discretization in amplitude, sometimes called *quantization* or just amplitude.

For many audio experiments, a sampling rate of 16,000 samples per second (16 kilohertz, or 16 kHz) is sufficient. In some cases 8 kHz is even enough to test ideas, and we further reduce complexity by representing amplitudes with 16 bits (65536 levels) or even 8 bits (256 levels). Both of these simplifications can help modeling, at the expense of input and output quality. Coarse discretization in time and

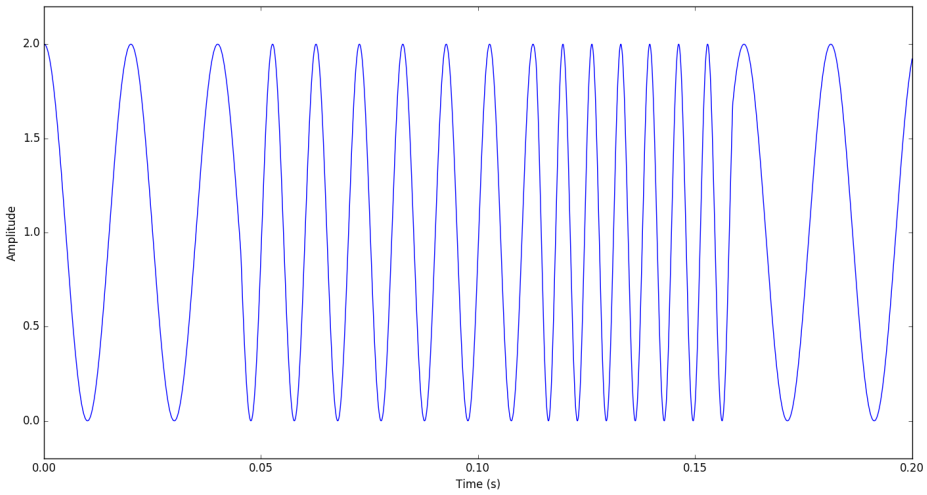


Figure 4.6 – Example waveform

amplitude can make the signal unrecognizable compared to the original, but finely discretized signals will increase computation time and degrade model performance.

Additionally for many audio pipelines a detailed grasp of digital signal processing (DSP) is required. The scope of DSP knowledge necessary to understand modern audio processing is a field in itself, but motivated readers can refer to the classic textbook by Smith (1997) for an introduction, or Oppenheim et al. (1999) for a detailed description of many DSP techniques. The knowledge necessary for understanding this text largely revolves around linear transforms and basis projection. More detailed discussion of this topic can be found in Ahmed and Rao (1975) and Smith (2016).

4.7 Speech Synthesis

4.7.1 Prior Work

Since the very beginning of wireless communications, speech synthesis has played an important role in applied research and development. Secure communications in World War II using SIGSALY (Boone et al., 2000) proved that speech synthesis

techniques could be combined with cryptography in order to securely and reliably transmit information over the Atlantic Ocean. Given tools such as the vocoder, engineers quickly began to use these techniques to compress speech, allowing more users to share the same channels as worldwide communications took the world by storm.

The advances necessary for better machine learning and generative modeling are closely related to compression. Classic systems such as those introduced in Hunt and Black (1996) for concatenative synthesis became ubiquitous in radio, internet, and cellular communications. Improvements in speech synthesis such as Agiomyrgiannakis (2015) have continued to improve voice quality for millions of users around the globe. In stark contrast to image recognition (Krizhevsky et al., 2012b), image generation (van den Oord et al., 2016), machine translation (Bahdanau et al., 2014), and speech recognition (Dahl et al., 2010) (Graves et al., 2013b) (Collobert et al., 2016), there has only very recently been major progress in using modern neural network methods for speech synthesis. Promising work with LSTMs by Zen et al. (2013) has incorporated RNNs in place of HMMs. A recent breakthrough by van den Oord et al. (2016) has largely eliminated the need for complex processing on the raw waveform data, though it still requires a standard text front end in order to build the correct context features to condition the acoustic generator. Generative approaches on raw waveforms have issues with deployment in productions systems, due to the amount of sequential dependencies involved in generated raw audio from a directed generative model. In general, speech synthesis with neural networks has had significant advancement in recent months, and these research directions have greatly influenced our future work.

4.7.2 Unconditional Concatentive Speech Synthesis

Given the historical preference for concatenative methods in speech synthesis, it is straightforward to consider using an RNN to select indices into large vocabulary of potential units. These units could be learned globally, selected based on utterance level statistics, or even crafted per user. When formulated in this way, the model itself becomes identical to an RNN language model such as presented in Mikolov et al. (2011) and Józefowicz et al. (2016). When using an RNN, most of the research effort manifests in finding the best units to feed into RNN model itself

and handling the signal processing necessary for clean reconstruction.

Data

Atom discovery, or *dictionary learning*, is a well established field with a variety of applicable algorithms for finding representative elements for representing a given dataset. In this work we chose to use extremely basic techniques for atom discovery such as K-means (Lloyd, 1982) and K-medians (Arora et al., 1998). The long-term goal is to replace this ad-hoc atom discovery procedure with a pretrained neural network, or a fully differentiable module learned as part of a larger end-to-end neural synthesis system.

As described in Section 4.5, a huge number of potential transforms are available from the signal processing literature. We tried the following representations in these initial experiments:

— Raw waveform snippets
— Complex valued short time Fourier transform (STFT)
— Magnitude and phase converted STFT
— Nonstationary constant Q transform (NS-CQT)
— Magnitude only STFT (spectrogram) with phase recovery
— Discrete cosine transform (DCT) compressed magnitude STFT
— Haar wavelet filterbanks
— Linear predictive coefficients (LPC)
— Line spectral frequencies (LSF)
— Minumum harmonic sinusoids (cite)
— Vocoder representations from STRAIGHT and WORLD vocoders
— Discrete cosine transform (DCT) compressed versions of the above
— Discretized versions of the above

The trials above covered a subsection of common representations for speech synthesis, though there are a still more to try. Overall the best representation in these experiments with regards to audio quality after selection and synthesis seemed to be linear predictive coefficients (LPC), though further investigation should be done. Changes in the model may allow transforms which previously seemed insufficient, and combinations of feature representations are also quite common in the concatenative synthesis literature. Indeed the model in van den Oord et al. (2016) has excellent synthesis quality from raw data using a modified CNN architecture,

and using parts of that model as a fixed feature representation may be useful for building a concatenative system.

In our approach, atoms are found using the K-medians algorithm on the chosen representation. Rather than detail the K-medians algorithm here, we encourage motivated readers to see Arora et al. (1998) for details. The key point for using K-medians over the more standard K-means, is that K-medians results in always using *true data* when selecting the cluster center. This better aligns with handcrafted approaches to concatenative synthesis, where subsets of a larger dataset are hand chosen by engineers and put into a dictionary for use in the unit selection process.

The dataset used for these experiments, Sandsmark (2010), was a set of utterances from a single speaker, saying one of seven possible words. The goal of this initial experiment was to prove that it is possible to do concatenative synthesis with RNNs, then scale up to larger datasets such as used in VRNN and even combine with the VRNN model itself.

Results

To date we are unable to outperform the quality of VRNN using signal processing based intermediate representations. Experimental results are seen in Figure 4.8 compared to the training data Figure 4.7. We see that our vocabulary selections are too varied, and the resulting reconstruction processing cannot correct the boundary effects. We still believe there exist good intermediate representations for use in speech synthesis with neural networks, but as the recent results in van den Oord et al. (2016) show, extreme data and compute time can allow raw waveform training and generation to excel on this task. The small size of our dataset seems to be a limiting factor, and we plan to try large scale training in future work despite the lack of small scale results.

4.8 Future Work

Symbolic modeling has a number of interesting research directions. Many papers, such as Papadopoulos et al. (2016), Pachet et al. (2013), and Pachet et al. (2011) use techniques from constraint programming, Markov-chain Monte Carlo,

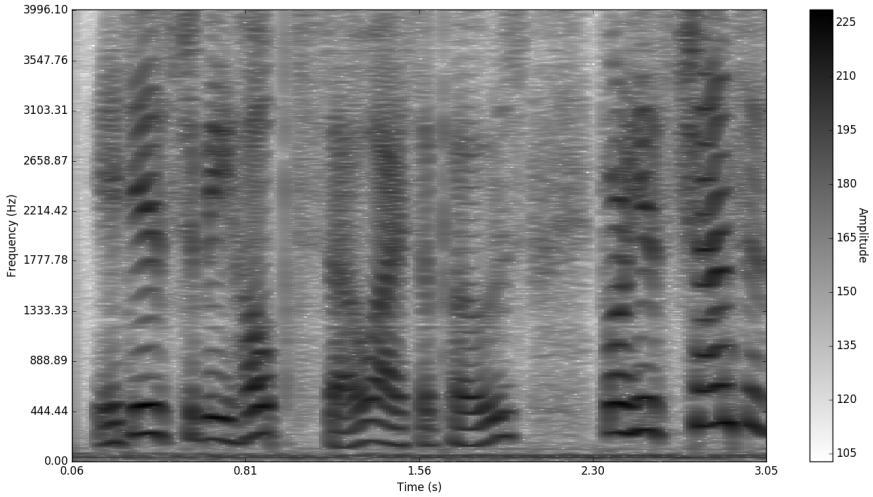


Figure 4.7 – Spectrogram of example utterance from the training set

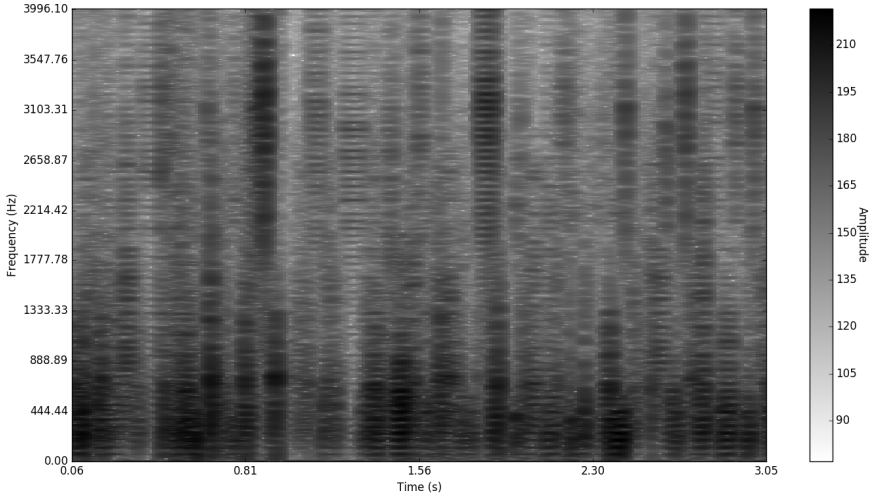


Figure 4.8 – Spectrogram of example unconditional generation

and belief propagation to great effect for symbolic music synthesis under stylistic, non-plagiaristic, and harmonic constraints. Incorporating portions of these ideas into neural network based generation frameworks for symbolic sequences is a clear future direction for our work on symbolic sequences. Beam search is a key component, and generalizing sequence search during conditional decoding for polyphonic sequences is another future research direction we wish to explore. Utilizing new directions in neural machine translation such as wordpiece modeling (Wu et al., 2016) would also allow our models to work at the level of phrases and motifs. Combining the event sequence representation models with additional masking based on voice, as seen in (Goodfellow et al., 2013), should further improve performance, generalize music with a variable number of voices, and is an immediate next step for this work.

The first step for audio tasks is to add conditioning based on textual features, in order to create a neural text-to-speech (TTS) system. In speech modeling, a key feature of signal processing based vocoders is a multi-stage approach which builds simple models of the base sound, then additional models of the error, or residual, of the simple model Agiomyrgiannakis (2015). This also appears in the residual stacks used by van den Oord et al. (2016), and a host of other models for classification and generation (He et al., 2015). Building these iterative refinement steps into generative models seems important for high quality results. Further investigation of how, when, and why these explicit residual connections are needed is an important future direction in neural network research. Accomplishing text-to-speech synthesis directly from characters without complicated text preprocessing frontends is another key step toward fully end-to-end TTS, and a logical next step for our speech synthesis research.

5 Conclusion

This thesis opens with a generic overview of machine learning and its relationship to more typical computer algorithm design. After introducing some core concepts necessary for the later chapters, we give an overview of two recently published papers which focus on structured prediction and generative modeling for sequences and images. We also provide an overview of recent experiments in symbolic music and speech generation, as well as a summary of recent work by other researchers. Summarizing a few of the key points from this thesis:

— Learning from data can create powerful custom solutions for a variety of problems in business, science, engineering, and art.

— Many common losses can be derived from simple distributional assumptions and the Kullback-Liebler divergence.

— Neural networks are particularly flexible tools, and allow a number of different ways to encode prior knowledge in the structure of the network.

— Neural networks are powerful but quality of the input data, structure of the preprocessing steps, and settings of training hyperparameters are all critical to the final performance of a model.

— Generative modeling and structured prediction are related tasks, and share many techniques. Advances in one domain are often easily applied to the other, even across different types of data.

— Structured prediction and generative modeling often improve when domain knowledge is added to the network structure using components such as convolutional and recurrent processing steps.

— Latent variables allow flexibility in interpretation as well as another potential way to add prior knowledge or constraints into the model structure.

— Modeling joint distributions as a product of conditional distributions can create a directed graphical model structure which is easy to sample from and also efficient during training.

— The flexibility of neural networks allow us to apply different tools from the

larger neural network toolkit on a wide variety of problems, but there are important design choices related to the data, prior knowledge, and desired outcomes which must be taken into consideration when designing new architectures.

Generative modeling of real world data using neural networks is beginning to show incredible results, and in the three months from the inception of this thesis to completion a host of benchmarks have been overturned or made obsolete, and tasks projected to be accomplished on multi-month and multi-year timelines have been shown working on today's hardware. These advances will enable a new wave of creativity by hobbyists, artists, and scientists alike. We look forward to what the next months and years bring.

References

(2016). *Myriam-Webster's Learner's Dictionary*. Myriam-Webster. Learning.

Abadi, M., A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

Abdel-Hamid, O., A. Mohamed, H. Jiang, and G. Penn (2012). Applying convolutional neural networks concepts to hybrid nn-hmm model for speech recognition. In *Acoustics, Speech and Signal Processing (ICASSP), 2012 IEEE International Conference on*.

Agiomyrgiannakis, Y. (2015). Vocaine the vocoder and applications in speech synthesis. In *ICASSP*.

Ahmed, N. U. and K. R. Rao (1975). *Orthogonal Transforms for Digital Signal Processing*. Secaucus, NJ, USA: Springer-Verlag New York, Inc.

Arora, S., P. Raghavan, and S. Rao (1998). Approximation schemes for euclidean k-medians and related problems. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, STOC '98, New York, NY, USA, pp. 106–113. ACM.

Association, M. M. (1999/2008). *Complete MIDI 1.0 Detailed Specification*.

Azzopardi, G. and N. Petkov (2013). Trainable COSFIRE filters for keypoint detection and pattern recognition. *IEEE Trans. Pattern Anal. Mach. Intell. 35*(2), 490–503.

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

Bahdanau, D., D. Serdyuk, P. Brakel, N. R. Ke, J. Chorowski, A. C. Courville, and Y. Bengio (2015). Task loss estimation for sequence prediction. *CoRR abs/1511.06456*.

Barbieri, G., F. Pachet, P. Roy, and M. D. Esposti (2012). Markov constraints for generating lyrics with style. In *Proceedings of the 20th European Conference on Artificial Intelligence*, ECAI'12, Amsterdam, The Netherlands, The Netherlands, pp. 115–120. IOS Press.

Bastien, F., P. Lamblin, R. Pascanu, J. Bergstra, I. Goodfellow, A. Bergeron, N. Bouchard, D. Warde-Farley, and Y. Bengio (2012). Theano: new features and speed improvements. Submited to the Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop.

Bastien, F., P. Lamblin, R. Pascanu, J. Bergstra, I. J. Goodfellow, A. Bergeron, N. Bouchard, and Y. Bengio (2012). Theano: new features and speed improvements. Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop.

Bayer, J. (2011). Statistics StackExchange http://stats.stackexchange.com/questions/7440/kl-divergence-between-two-univariate-gaussians.

Bayer, J. and C. Osendorfer (2014). Learning stochastic recurrent networks. *arXiv preprint arXiv:1411.7610*.

Bellman, R. (1957). *Dynamic Programming* (1 ed.). Princeton, NJ, USA: Princeton University Press.

Bengio, Y. (2013). Practical recommendations for gradient-based training of deep architectures. In K.-R. Müller, G. Montavon, and G. B. Orr (Eds.), *Neural Networks: Tricks of the Trade*. Springer.

Bengio, Y., N. Boulanger-Lewandowski, and R. Pascanu (2013). Advances in optimizing recurrent networks. In *ICASSP*, pp. 8624–8628. IEEE.

Bergstra, J., O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-Farley, and Y. Bengio (2010a). Theano: a CPU and GPU math expression compiler. In *Proc. SciPy*.

Bergstra, J., O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-Farley, and Y. Bengio (2010b). Theano: a CPU and GPU math expression compiler. In *Proceedings of the Python for Scientific Computing Conference (SciPy)*.

Bertrand, A., K. Demuynck, V. Stouten, and H. V. Hamme (2008). Unsupervised learning of auditory filter banks using non-negative matrix factorisation. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 4713–4716. IEEE.

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

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

Boone, J. V., R. R. Peterson, and U. States. (2000). *The start of the digital revolution, SIGSALY [microform] : secure digital voice communications in World War II*. Center for Cryptologic History, National Security Agency [Fort George G. Meade, Md.

Bottou, L. (1998). On-line learning in neural networks. Chapter On-line Learning and Stochastic Approximations, pp. 9–42. New York, NY, USA: Cambridge University Press.

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

Boulanger-Lewandowski, N., Y. Bengio, and P. Vincent (2012b). Modeling temporal dependencies in high-dimensional sequences: Application to polyphonic music generation and transcription. In *Proceedings of the Twenty-nine International Conference on Machine Learning (ICML'12)*. ACM.

Boulanger-Lewandowski, N., Y. Bengio, and P. Vincent (2012c). Modeling temporal dependencies in high-dimensional sequences: Application to polyphonic music generation and transcription. In *ICML'2012*.

Brooks, Jr., F. P. (1995). *The Mythical Man-month (Anniversary Ed.)*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.

Brown, P. F., V. J. D. Pietra, P. V. DeSouza, J. C. Lai, and R. L. Mercer (1992). Class-based n-gram models of natural language. *Computational Linguistics 18*, 467–479.

Buja, A., W. Stuetzle, and Y. Shen (2005). Loss functions for binary class probability estimation and classification: Structure and applications, manuscript, available at www-stat.wharton.upenn.edu/ buja.

Chelba, C., T. Mikolov, M. Schuster, Q. Ge, T. Brants, P. Koehn, and T. Robinson (2014). One billion word benchmark for measuring progress in statistical language modeling. In *INTERSPEECH 2014, 15th Annual Conference of the International Speech Communication Association, Singapore, September 14-18, 2014*, pp. 2635–2639.

Cho, G. J.

Cho, K., B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio (2014). Learning phrase representations using rnn encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1724–1734.

Choromanska, A., M. Henaff, M. Mathieu, G. B. Arous, and Y. LeCun (2014). The loss surface of multilayer networks.

Chorowski, J., D. Bahdanau, K. Cho, and Y. Bengio (2014). End-to-end continuous speech recognition using attention-based recurrent nn: First results. arXiv:1412.1602.

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

Chung, J., C. Gulcehre, K. Cho, and Y. Bengio (2015). Gated feedback recurrent neural networks. In *International Conference on Machine Learning*.

Chung, J., K. Kastner, L. Dinh, K. Goel, A. Courville, and Y. Bengio (2015, June). A recurrent latent variable model for sequential data. *ArXiv e-prints abs/1506.02216*.

Ciresan, D., U. Meier, J. Masci, and J. Schmidhuber (2012). Multi-column deep neural network for traffic sign classification. *Neural Networks 32*, 333–338.

Ciresan, D., U. Meier, and J. Schmidhuber (2012). Multi-column deep neural networks for image classification. Technical report, arXiv:1202.2745.

Ciresan, D. C., U. Meier, L. M. Gambardella, and J. Schmidhuber (2010). Deep big simple neural nets excel on handwritten digit recognition. *arXiv abs/1003.0358*.

Coates, A., A. Karpathy, and A. Ng (2012). Emergence of object-selective features in unsupervised feature learning. In *NIPS'2012*.

Collobert, R., C. Puhrsch, and G. Synnaeve (2016). Wav2letter: an end-to-end convnet-based speech recognition system. *CoRR abs/1609.03193*.

Colombo, F., S. P. Muscinelli, A. Seeholzer, J. Brea, and W. Gerstner (2016). Algorithmic composition of melodies with deep recurrent neural networks. *CoRR abs/1606.07251*.

Cope, D. (1991). *Computers and Musical Style*. Madison, WI, USA: A-R Editions, Inc.

Cuthbert, M. S. and C. Ariza (2010, August 9-13). Music21: A toolkit for computer-aided musicology and symbolic music data. In *Proceedings of the 11th International Society for Music Information Retrieval Conference*, Utrecht, The Netherlands, pp. 637–642. http://ismir2010.ismir.net/proceedings/ismir2010-108.pdf.

Dahl, G. E., M. Ranzato, A. Mohamed, and G. E. Hinton (2010). Phone recognition with the mean-covariance restricted Boltzmann machine. In *Advances in Neural Information Processing Systems (NIPS)*.

Dauphin, Y., R. Pascanu, C. Gulcehre, K. Cho, S. Ganguli, and Y. Bengio (2014). Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. In *NIPS'2014*.

Dieleman, S. and B. Schrauwen (2014, May). End-to-end learning for music audio. In *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*, pp. 6964–6968.

Dinh, L. (2016). Personal communication.

Duchi, J., E. Hazan, and Y. Singer (2011, July). Adaptive subgradient methods for online learning and stochastic optimization. *J. Mach. Learn. Res. 12*, 2121–2159.

Eck, D. and J. Schmidhuber (2002). Learning the long-term structure of the blues. In J. Dorronsoro (Ed.), *Artificial Neural Networks – ICANN 2002 (Proceedings)*, Berlin, pp. 284–289. Springer.

Eigen, D. and R. Fergus (2015). Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture. In *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*, pp. 2650–2658.

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

Fabius, O., J. R. van Amersfoort, and D. P. Kingma (2014). Variational recurrent auto-encoders. *arXiv preprint arXiv:1412.6581*.

Finney, R. L., M. D. Weir, and F. R. Giordano (2003). *Thomas' Calculus* (10 ed.). Addison Wesley.

Fukushima, K. (1980). Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics 36*, 193–202.

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

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

Goel, K. and R. Vohra (2014). Learning temporal dependencies in data using a DBN-BLSTM. *CoRR abs/1412.6093*.

Good, M. (2001). MusicXML for Notation and Analysis. In W. B. Hewlett and E. S. Field (Eds.), *The virtual score: representation, retrieval, restoration*, Volume 12 of *Computing in Musicology*, pp. 113–124. Massachusetts: The MIT Press.

Goodfellow, I. (2015). Deep Learning Summer School 2015.

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

Goodfellow, I., D. Warde-Farley, M. Mirza, A. Courville, and Y. Bengio (2013). Maxout networks. In *Proceedings of The 30th International Conference on Machine Learning*, pp. 1319–1327.

Goodfellow, I. J., Y. Bulatov, J. Ibarz, S. Arnoud, and V. Shet (2014). Multi-digit number recognition from Street View imagery using deep convolutional neural networks. In *International Conference on Learning Representations*.

Goodfellow, I. J., Y. Bulatov, J. Ibarz, S. Arnoud, and V. D. Shet (2013). Multi-digit number recognition from street view imagery using deep convolutional neural networks. *CoRR abs/1312.6082*.

Graham, B. (2014a). Fractional max-pooling. *arXiv abs/1412.6071*.

Graham, B. (2014b). Spatially-sparse convolutional neural networks. *arXiv abs/1409.6070*.

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

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

Graves, A., M. Liwicki, H. Bunke, J. Schmidhuber, and S. Fernández (2008). Unconstrained on-line handwriting recognition with recurrent neural networks. In J. Platt, D. Koller, Y. Singer, and S. Roweis (Eds.), *NIPS'2007*, pp. 577–584.

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

Graves, A., A.-r. Mohamed, and G. Hinton (2013b). Speech recognition with deep recurrent neural networks. In *ICASSP'2013*, pp. 6645–6649.

Graves, A. and J. Schmidhuber (2009). Offline handwriting recognition with multi-dimensional recurrent neural networks. In D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou (Eds.), *NIPS'2008*, pp. 545–552.

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

Hadjeres, G., J. Sakellariou, and P. F. (2016, September). Style imitation and chord invention in polyphonic music with exponential families. Technical report, arXiv:1609.05152. http://arxiv.org/abs/1609.05152.

Hastie, T., R. Tibshirani, and J. Friedman (2001). *The Elements of Statistical Learning*. Springer Series in Statistics. New York, NY, USA: Springer New York Inc.

He, K., X. Zhang, S. Ren, and J. Sun (2015). Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*.

Hinton, G. (2012). Neural networks for machine learning coursera video lectures - geoffrey hinton.

Hinton, G. E., N. Srivastava, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov (2012). Improving neural networks by preventing co-adaptation of feature detectors. *arXiv abs/1207.0580*.

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

Hunt, A. J. and A. W. Black (1996). Unit selection in a concatenative speech synthesis system using a large speech database. In *Proceedings of the Acoustics, Speech, and Signal Processing, 1996. On Conference Proceedings., 1996 IEEE*

International Conference - Volume 01, ICASSP '96, Washington, DC, USA, pp. 373–376. IEEE Computer Society.

Johnson, D. (2015). Composing music with recurrent neural networks.

Jordan, M. I. (1986). Serial order: a parallel distributed processing approach. Technical Report 8604, ICS (Institute for Cognitive Science, University of California).

Józefowicz, R., O. Vinyals, M. Schuster, N. Shazeer, and Y. Wu (2016). Exploring the limits of language modeling. *CoRR abs/1602.02410*.

Jozefowicz, R., W. Zaremba, and I. Sutskever (2015). An empirical exploration of recurrent network architectures. In *Proceedings of The 32nd International Conference on Machine Learning*, pp. 2342–2350.

King, S. and V. Karaiskos (2013). The blizzard challenge 2013. In *The Ninth annual Blizzard Challenge*.

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

Kingma, D. P. and J. Ba (2015). Adam: A method for stochastic optimization.

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

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

Krizhevsky, A. (2014). One weird trick for parallelizing convolutional neural networks. *arXiv preprint arXiv:1404.5997*.

Krizhevsky, A. and G. Hinton (2009). Learning multiple layers of features from tiny images. Technical report, University of Toronto.

Krizhevsky, A., I. Sutskever, and G. Hinton (2012a). ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25 (NIPS'2012)*.

Krizhevsky, A., I. Sutskever, and G. E. Hinton (2012b). Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, pp. 1097–1105.

Larochelle, H. and I. Murray (2011a). The Neural Autoregressive Distribution Estimator. In *AISTATS'2011*.

Larochelle, H. and I. Murray (2011b). The Neural Autoregressive Distribution Estimator. In *Proc. AISTATS'2011*.

LeCun, Y., B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel (1989). Backpropagation applied to handwritten zip code recognition. *Neural Computation 1*(4), 541–551.

LeCun, Y., L. Bottou, and Y. Bengio (1997, Apr). Reading checks with multilayer graph transformer networks. In *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP'97)*, Volume 1, pp. 151–154.

LeCun, Y., L. Bottou, G. B. Orr, and K.-R. Müller (1998). Efficient BackProp. In G. B. Orr and K.-R. Müller (Eds.), *Neural Networks: Tricks of the Trade*, pp. 9–50. Springer.

LeCun, Y., P. Haffner, L. Bottou, and Y. Bengio (1999). Object recognition with gradient-based learning. In *Shape, Contour and Grouping in Computer Vision*, pp. 319–345. Springer.

Lee, C., S. Xie, P. Gallagher, Z. Zhang, and Z. Tu (2014). Deeply-supervised nets. *arXiv abs/1409.5185*.

Lee, H., P. Pham, Y. Largman, and A. Y. Ng (2009). Unsupervised feature learning for audio classification using convolutional deep belief networks. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 1096–1104.

Liang, F. (2016). Bachbot: Automatic composition in the style of bach chorales.

Lin, M., Q. Chen, and S. Yan (2014, April). Network in network. In *Proceedings of the Second International Conference on Learning Representations (ICLR 2014)*.

Liwicki, M. and H. Bunke (2005). Iam-ondb-an on-line english sentence database acquired from handwritten text on a whiteboard. In *Proceedings of Eighth International Conference on Document Analysis and Recognition*, pp. 956–961. IEEE.

Lloyd, S. P. (1982). Least squares quantization in PCM. *IEEE Transactions on Information Theory 28*(2), 129–137.

Mairal, J., P. Koniusz, Z. Harchaoui, and C. Schmid (2014). Convolutional kernel networks. In *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pp. 2627–2635.

Martens, J. and I. Sutskever (2011). Learning recurrent neural networks with Hessian-free optimization. In *Proc. ICML'2011*. ACM.

Mikolov, T. (2012). *Statistical Language Models based on Neural Networks*. Ph. D. thesis, Brno University of Technology.

Mikolov, T., S. Kombrink, L. Burget, J. Cernocky, and S. Khudanpur (2011). Extensions of recurrent neural network language model. In *Proc. 2011 IEEE international conference on acoustics, speech and signal processing (ICASSP 2011)*.

Mitchell, M. (1998). *An Introduction to Genetic Algorithms*. Cambridge, MA, USA: MIT Press.

Murphy, K. P. (2012). *Machine Learning: A Probabilistic Perspective*. The MIT Press.

Nair, V. and G. E. Hinton (2010). Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML)*, pp. 807–814.

Netzer, Y., T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng (2011). Reading digits in natural images with unsupervised feature learning. Deep Learning and Unsupervised Feature Learning Workshop, NIPS.

Nowak, R. (2009). University Lecture http://nowak.ece.wisc.edu/SLT09/lecture13.pdf.

Ogus, A. (2007). University Lecture https://math.berkeley.edu/õgus/$Math_5 4 - 07/Lectures/notes8.pdf$.

Oppenheim, A. V., R. W. Schafer, and J. R. Buck (1999). *Discrete-time Signal Processing (2Nd Ed.)*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc.

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

Pachet, F., P. Roy, and F. Ghedini (2013, September). Creativity through style manipulation: the flow machines project. In *2013 Marconi Institute for Creativity Conference (MIC 2013)*, Volume 80, Bologna (Italy).

Pachitariu, M. and M. Sahani (2012). Learning visual motion in recurrent neural networks. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 1322–1330.

Papadopoulos, A., F. Pachet, and P. Roy (2016). *Generating Non-plagiaristic Markov Sequences with Max Order Sampling*, pp. 85–103. Cham: Springer International Publishing.

Pascanu, R., C. Gulcehre, K. Cho, and Y. Bengio (2014). How to construct deep recurrent neural networks. In *ICLR*.

Pascanu, R., T. Mikolov, and Y. Bengio (2013). On the difficulty of training recurrent neural networks. In *ICML'2013*.

Raffel, C. and D. Ellis (2014). Intuitive analysis, creation and manipulation of midi data with pretty_midi. In *Late Breaking and Demo Papers, the 15th International Society for Music Information Retrieval Conference*.

Rezende, D. J., S. Mohamed, and D. Wierstra (2014). Stochastic backpropagation and approximate inference in deep generative models. In *Proceedings of The 31st International Conference on Machine Learning (ICML)*, pp. 1278–1286.

Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review 65*, 386–408.

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

Russel, S. J. and P. Norvig (2003). *Artificial Intelligence: a Modern Approach*. Prentice Hall.

Sainath, T. N., A.-r. Mohamed, B. Kingsbury, and B. Ramabhadran (2013). Deep convolutional neural networks for lvcsr. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pp. 8614–8618. IEEE.

Sainath, T. N., O. Vinyals, A. Senior, and H. Sak (2015). Convolutional, long short-term memory, fully connected deep neural networks. In *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on*.

Sandsmark, H. (2010). Hmm speech recognition. Training data generated for the report by Sandsmark titled 'Isolated-word speech recognition using hidden Markov models'.

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

Shannon, C. E. (1948). A mathematical theory of communication. *Bell System Technical Journal 27*(3), 379–423.

Simard, P. Y., D. Steinkraus, and J. C. Platt (2003). Best practices for convolutional neural networks applied to visual document analysis. In *7th International Conference on Document Analysis and Recognition (ICDAR 2003), 2-Volume Set, 3-6 August 2003, Edinburgh, Scotland, UK*, pp. 958–962.

Simonyan, K. and A. Zisserman (2015). Very deep convolutional networks for large-scale image recognition. In *ICLR*.

Smith, J. O. (2016). *Spectral Audio Signal Processing*. http://-ccrma.stanford.edu/~jos/sasp/. online book, 2011 edition.

Smith, S. W. (1997). *The Scientist and Engineer's Guide to Digital Signal Processing*. San Diego, CA, USA: California Technical Publishing.

Snoek, J., H. Larochelle, and R. P. Adams (2012). Practical bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems*

2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States., pp. 2960–2968.

Sobel, I. and G. Feldman (1968). A 3x3 isotropic gradient operator for image processing. Never published but presented at a talk at the Stanford Artificial Project.

Springenberg, J. T., A. Dosovitskiy, T. Brox, and M. Riedmiller (2014). Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806*.

Springenberg, J. T. and M. A. Riedmiller (2013). Improving deep neural networks with probabilistic maxout units. *arXiv abs/1312.6116*.

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

Strang, G. (2006). *Linear algebra and its applications*. Thomson, Brooks/Cole.

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

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

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

Szegedy, C., W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich (2014). Going deeper with convolutions. *arXiv preprint arXiv:1409.4842*.

Tokuda, K., Y. Nankaku, T. Toda, H. Zen, J. Yamagishi, and K. Oura (2013). Speech synthesis based on hidden markov models. *Proceedings of the IEEE 101*(5), 1234–1252.

Torralba, A., R. Fergus, and W. T. Freeman (2008). 80 million tiny images: A large dataset for non-parametric object and scene recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence 30*(11), 1958–1970.

Tóth, L. (2014). Combining time-and frequency-domain convolution in convolutional neural network-based phone recognition. In *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*, pp. 190–194. IEEE.

Usergroup, A.

van den Oord, A., S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu (2016). Wavenet: A generative model for raw audio.

van den Oord, A., N. Kalchbrenner, and K. Kavukcuoglu (2016). Pixel recurrent neural networks. In *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, pp. 1747–1756.

van den Oord, A., N. Kalchbrenner, O. Vinyals, L. Espeholt, A. Graves, and K. Kavukcuoglu (2016). Conditional image generation with pixelcnn decoders. *CoRR abs/1606.05328*.

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

Vinyals, O., A. Toshev, S. Bengio, and D. Erhan (2014). Show and tell: a neural image caption generator. arXiv 1411.4555.

Visin, F., K. Kastner, K. Cho, M. Matteucci, A. Courville, and Y. Bengio (2015, May). ReNet: A recurrent neural network based alternative to convolutional networks. *ArXiv e-prints abs/1505.00393*.

Walder, C. (2016). Modelling symbolic music: Beyond the piano roll. *CoRR abs/1606.01368*.

Wan, L., M. D. Zeiler, S. Zhang, Y. LeCun, and R. Fergus (2013). Regularization of neural networks using dropconnect. In *Proceedings of the 30th International*

Conference on Machine Learning, ICML 2013, Atlanta, GA, USA, 16-21 June 2013, pp. 1058–1066.

Weinberger, S. (2015). The speech accent archieve. http://accent.gmu.edu/.

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

Xu, K., J. L. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhutdinov, R. S. Zemel, and Y. Bengio (2015). Show, attend and tell: Neural image caption generation with visual attention. arXiv:1502.03044.

Yao, L., A. Torabi, K. Cho, N. Ballas, C. Pal, H. Larochelle, and A. Courville (2015). Video description generation incorporating spatio-temporal features and a soft-attention mechanism. *arXiv:1502.08029*.

Yosinski, J., J. Clune, A. Nguyen, T. Fuchs, and H. Lipson (2015). Understanding neural networks through deep visualization. In *Deep Learning Workshop, International Conference on Machine Learning (ICML)*.

Zeiler, M. D. (2012). ADADELTA: an adaptive learning rate method. *CoRR abs/1212.5701*.

Zeiler, M. D. and R. Fergus (2013). Stochastic pooling for regularization of deep convolutional neural networks. *arXiv abs/1301.3557*.

Zeiler, M. D. and R. Fergus (2014). Visualizing and understanding convolutional networks. In *ECCV'14*.

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

Zimmerman, Y. (2016). A dual classification approach to music language modeling.