Université de Montréal

Speech Synthesis Using Recurrent Neural Networks

par José Manuel Rodríguez Sotelo

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

Décembre, 2016

© José Manuel Rodríguez Sotelo, 2016.

Résumé

Les réseaux neuronaux récurrents sont des outils efficaces pour modeler les données à structure séquentielle. Dans ce mémoire, nous décrivons comment les utiliser pour la synthèse vocale.

Nous commençons avec une introduction à l'apprentissage automatique et aux réseaux neuronaux dans le chapitre 1.

Dans le chapitre 2, nous développons un gradient algorithmique stochastique automatique ayant pour effet de réduire le poids des recherches extensives hyper-paramétrées pour l'optimisateur. L'algorithme proposé exploite un estimateur de courbure du coût de la fonction de moindre variance, et utilise celui-ci pour obtenir un taux d'apprentissage adaptatif qui soit automatiquement calibré pour chaque paramètre.

Dans le chapitre 3, nous proposons un modèle innovateur pour la génération audio inconditionnelle, basée sur la génération d'un seul échantillon audio à la fois. Nous montrons que notre modèle, qui prend avantage de la combination de modules sans mémoire (notamment les perceptrons autorégressifs à plusieurs couches et les réseaux de neurones récurrents dans une structure hiérarchique), est capable de capturer les sources de variation sous-jacentes dans les séquences temporelles, et ce, sur de très longs laps de temps, sur trois ensembles de données de nature différente. Les résultats de l'évaluation humaine à l'écoute des échantillons générés semblent indiquer que notre modèle est préféré à d'autres modèles de compétiteurs. Nous montrons aussi comment chaque composante du modèle contribue à ces performances.

Dans le chapitre 4, nous présentons un modèle d'encodeur-décodeur focalisé sur la synthèse vocale. Notre modèle apprend à produire les caractéristiques acoustiques à partir d'une séquence de phonèmes ou de lettres. L'encodeur se constitue d'un réseau neuronal récurrent bidirectionnel acceptant des entrées sous forme de texte ou de phonèmes. Le décodeur se constitue, pour sa part, d'un réseau neuronal récurrent avec attention produisant les caractéristiques acoustiques. Par ailleurs, nous adaptons ce modèle, afin qu'il puisse réaliser la synthèse vocale de plusieurs individus, et nous la testons en anglais et en espagnol.

Finalement, nous effectuons une réflection sur les résultats obtenus dans ce mémoire, afin de proposer de nouvelles pistes de recherche.

Mots clés: réseaux de neurones, apprentissage automatique, apprentissage de représentations profondes, apprentissage de représentations, synthèse vocale,

traitement du signal, optimisation

Summary

Recurrent neural networks are useful tools to model data with sequential structure. In this work, we describe how to use them for speech synthesis.

We start with an introduction to machine learning and neural networks in Chapter 1.

In Chapter 2, we develop an automatic stochastic gradient algorithm which reduces the burden of extensive hyper-parameter search for the optimizer. Our proposed algorithm exploits a lower variance estimator of curvature of the cost function and uses it to obtain an automatically tuned adaptive learning rate for each parameter.

In Chapter 3, we propose a novel model for unconditional audio generation based on generating one audio sample at a time. We show that our model, which profits from combining memory-less modules, namely autoregressive multilayer perceptrons, and stateful recurrent neural networks in a hierarchical structure is able to capture underlying sources of variation in the temporal sequences over very long time spans, on three datasets of different nature. Human evaluation on the generated samples indicate that our model is preferred over competing models. We also show how each component of the model contributes to the exhibited performance.

In Chapter 4, we present Char2Wav, an end-to-end model for speech synthesis. Char2Wav has two components: a **reader** and a **neural vocoder**. The *reader* is an encoder-decoder model with attention. The encoder is a bidirectional recurrent neural network (RNN) that accepts text or phonemes as inputs, while the decoder is a recurrent neural network with attention that produces vocoder acoustic features. *Neural vocoder* refers to a conditional extension of SampleRNN which generates raw waveform samples from intermediate representations. We show results in English and Spanish. Unlike traditional models for speech synthesis, Char2Wav learns to produce audio directly from text.

Finally, we reflect on the results obtained in this work and propose future directions of research in the area.

Keywords: neural networks, machine learning, deep learning, representation learning, speech synthesis, signal processing, optimization

Contents

List of Figures

List of Tables

List of Abbreviations

AE	Auto-Encoder
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 Squared Error
NLL	Negative Log-Likelihood
RNN	Recurrent Neural Network
SGD	Stochastic Gradient Descent
SP	Signal Processing
SS	Speech Synthesis
VAE	Variational Auto-Encoder

1 Introduction

To conclude what does *learning* really mean would definitely make your favorite philosopher happy. Our concept of learning contains different processes that make the definition unclear. Some of the possible definitions of learning are: *acquire knowledge of something by experience or by study*, *add something the memory*, or *modify the behavior based on experience.*

The ability to learn is the single most important quality that makes us thrive in our environment. It is the ability that allows us to adapt our behavior to respond to threats and to get skills that allow our survival. Learning is such an important part of our lives that without it we could not communicate, we could not make complex tools, we could not transfer our knowledge to new generations, etc. In a few words, if we were not able to learn new things, we would be extinct.

Learning is so rooted in our brains that we do not even need to think about it for it to happen. Usually, when you learn a new skill, you do not think about *how to learn* it. In part because of this, we do not have a clear understanding of how learning works inside our brains. Hence, it is challenging to provide this amazing skill to the devices that we create. By giving machines the ability to learn, we hope that, eventually, they will be able to solve the problems that we have not found the solution of.

This is a thesis about machine learning. In particular, we tackle the problem of applying modern machine learning techniques to speech synthesis (SS). Speech synthesis, also known as Text-to-Speech (TTS), consists on finding the mapping from text to audio signal. In the traditional approach, developing a new SS system requires specialized linguistic knowledge and handcrafted features. In this work, we develop a new algorithm that uses deep learning to make this process much simpler. In summary, this thesis proposes a new solution to an old problem: speech synthesis.

1.1 Machine Learning

Machine learning tries to provide the *ability to learn* to computers. In machine learning, we usually do not know how to describe the solution to a problem. Instead, we give our models the ability to learn the solution to the problem by themselves. The motivation for this is simple: If we are able to teach the computers how to solve problems by themselves, they will be able to solve unsolved problems.

The idea that computers can learn to solve problems by themselves is old. As **?** describe, during the first years of the field of Artificial Intelligence (1952 - 1969), it was believed that in a few years computers would become better than humans to perform complex tasks. However, this optimism was over because of the limit in computational resources of the time. Even more, confidence in the field plummeted. People started doubting that computers would be able to learn how to solve difficult problems. Regardless of the limitations in the computational power of the time, these years were fruitful in ideas about learning. Ideas from other fields, like psychology, were adopted. Therefore, some of the techniques developed in the field come from models to understand human or animal behavior.

Recently, there has been an explosion in the computational power and the data available. These two factors have transformed the field.

1.2 Generative Models

Every adult human has an incredible amount of information about the world that we live in. We understand that the world has 3 dimensions. We understand that we interact with objects that continuously move. We know how to navigate our world, how to communicate with other humans. We learn from historic recordings what happened in our past. We have created tools, like the Internet, that allows us to learn about the present.

Most of this huge amount of information is easily accessible in the Internet or in the physical World. However, it is no trivial to develop models and algorithms that can analyze and understand this treasure trove of data. Generative models are one of the most promising approaches towards this goal (**?**).

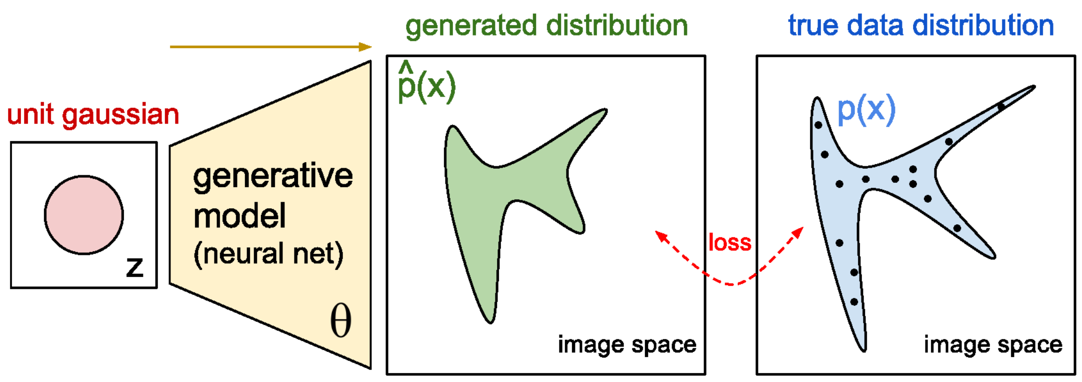


Figure 1.1 – Generative models. Figure from **?**.

The idea behind generative models is that if we are able to artificially generate data that looks like a phenomenon, then we understand how it works. Furthermore, it is important to note that the neural networks we use as generative models should have a number of parameters significantly smaller than the amount of data we train them on. In this way, the models are forced to discover and efficiently internalize the essence of the data in order to generate examples that look like it.

Formally speaking, if we have $\{x_1, \ldots, x_N\}$ where $x_i \sim p(x)$ for any $i \in \{1, \ldots, N\}$ we want to model the true data distribution $p(x)$. We do this with a *deterministic* neural network that takes as input samples from a simple distribution (for example the Gaussian distribution) and maps them to samples in the original space. This neural network implicitly defines a distribution $\hat{p}_\theta(x)$. We try to find parameters θ such that $\hat{p}_\theta(x)$ is close to $p(x)$. This usually means that we try to minimize the Kullback–Leibler (KL) divergence between the two distributions. A visualization of this idea is shown in Figure 1.1.

The most common approaches to tackle this problem are Generative Adversarial Networks (GANs) (**?**), Variational Autoencoders (VAEs) (**?**), autoregressive models such as the PixelRNN (**?**).

1.3 Optimization

An important part of training neural networks is the optimization of an objective function. We usually try to minimize a **cost function**. In particular, we try to find the parameters θ of a neural network that minimize a cost function $J(\theta)$. The function $J(\theta)$ usually includes a measure of performance in the desired task as well as additional regularization terms.

$$J(\theta) = \mathbb{E}_{(x,y)\sim\hat{p}_{data}} L(f(x;\theta), y) \tag{1.1}$$

where L is a per-example cost function, $f(x;\theta)$ is the predicted output when the input is x, and $\hat{p}_{data}$ the empirical distribution.

In contrast with the field of optimization, in machine learning the goal in and of itself is not to minimize the cost function. Instead, the goal is to have a good performance on unobserved examples. That is, our goal is to optimize:

$$J^*(\theta) = \mathbb{E}_{(x,y)\sim p_{data}} L(f(x;\theta), y) \tag{1.2}$$

where p_{data} is the *data generating* distribution. Since we do not know p_{data} in most real world applications, we work with a training set of examples. This makes machine learning different.

Overfitting is a measure of the difference between the goals of optimization and machine learning. Since we work with only a training set of examples, if we use high capacity models and powerful optimization algorithms we can simply memorize the training set. Because of this, machine learning makes us of **regularization techniques** like early stopping. Furthermore, the most effective modern optimization algorithms are based on gradient descent. Since many useful loss functions do not have useful derivatives, we often rely on *surrogate* loss functions which act as a proxy.

In Chapter 2, we develop an automatic stochastic gradient algorithm which reduces the burden of extensive hyper-parameter search for the optimizer. Our proposed algorithm exploits a lower variance estimator of curvature of the cost function and uses it to obtain an automatically tuned adaptive learning rate for each parameter.

1.4 Neural Networks

Deep learning is a subset of Machine Learning that focuses on Neural Networks. An artificial neuron is a simple model of how a neuron in the brain works. In the brain, neurons respond to electric impulses sent by neighboring neurons. In the same way, a neural network is a model that connects many single neurons. Furthermore, deep learning got its name from the fact that we usually organize the neurons in layers and we transform the input layer-by-layer. This is equivalent to composing together many different functions. In particular, we model neural networks using the following equations:

$$h^0 = x \tag{1.3}$$

$$h^i = g_i(w_i^T h^{i-1} + b_i) \quad \forall i \in 1, ..., N \tag{1.4}$$

where x is the input, N is the number of layers, and g_i is the activation function of the i-th layer. This model is called a feed-forward neural network or deep neural network (DNN). Where N is the depth of the network.

1.5 Recurrent Neural Networks

Recurrent neural networks (RNN) have been proved to be a powerful tool to model sequences like text (**?**), handwriting (**?**) and more. In the traditional generative model, a RNN is trained to model a sequence by predicting one step at a time and predicting what comes next. The RNN is trained to use past information to estimate the parameters of a distribution of the next step of the sequence. At testing time, the next step is sampled from this distribution and is taken as truth for the following computations as in Figure 1.2

$$P(y) = P(y_1) \prod_{t=2}^{T} P(y_t | y_1, ..., y_{t-1}) \tag{1.5}$$

$$= P(y_1) \prod_{t=2}^{T} P(y_t | h_t) \tag{1.6}$$

$$h_t^i = g(W_{hh} h_{t-1}^i + W_{ih} h_t^{i-1}) \tag{1.7}$$

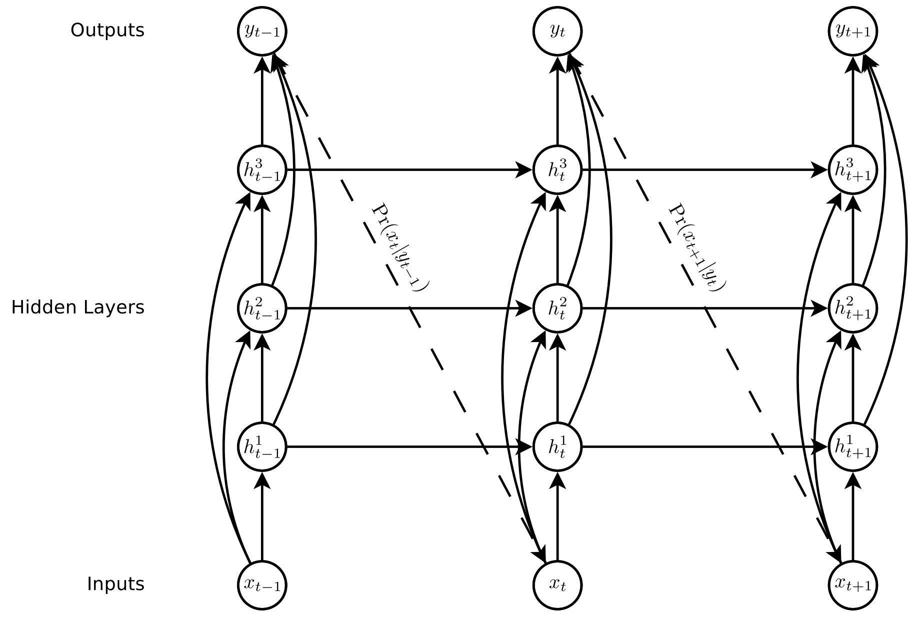


Figure 1.2 – Deep recurrent neural network. The dashed lines represent sampling from a distribution. Figure from **?**.

And we train the network to minimize the negative loglikelihood $\mathcal{L}(y)$.

$$\mathcal{L}(y) = -\sum_{t=1}^{T} \log P(y_t|h_t) \tag{1.8}$$

Furthermore, we can condition the sequence generation with additional information X. In general, X can be anything. For instance, X in image captioning X is an image. In speech recognition X is an audio sequence. In handwriting synthesis and speech synthesis X is a text sequence. Encoder-decoder models (**??**) were developed to tackle problems where X is a sequence. Therefore, these kinds of models are well-suited for speech synthesis.

A weakness of encoder-decoder models is that they encode all the information about X in a fixed size vector. **?** solve this issue using an attention mechanism. Since then, attention mechanisms have been widely adopted in the deep and have recently shown great performance on a variety of tasks including handwriting synthesis (**?**), machine translation (**?**), image caption generation (**?**) and speech recognition (**??**).

1.6 Speech synthesis

Audio generation is a challenging task at the core of many problems of interest, such as text-to-speech synthesis, music synthesis and voice conversion. The particular difficulty of audio generation is that there is often a very large discrepancy between the dimensionality of the the raw audio signal and that of the effective semantic-level signal. Consider the task of speech synthesis, where we are typically interested in generating utterances corresponding to full sentences. Even at a relatively low sample rate of 16kHz, on average we will have 6,000 samples per word generated.[1]

Speech Synthesis consists on the mapping from text to audio signal. It has two main goals, **intelligibility** and **naturalness**. Traditionally, speech synthesis has been solved by dividing the problem in two stages. The first stage, known as the **frontend**, transforms the text into linguistic features. These linguistic features usually include phone, syllable, word, phrase and utterance-level features (**?**) (e.g. phone identities, syllable stress, the number of syllables in a word, and position of the current syllable in a phrase) with additional frame position and phone duration features (**??**). These features are obtained by forced alignment at the training stage. The second stage, known as the **backend**, takes as input the linguistic features generated by the frontend and produces the corresponding sound.

There are two popular approaches to tackle the backend problem: concatenative synthesis and parametric synthesis. In concatenative synthesis, the audio is cut into pieces that are labeled with their phoneme or other linguistic information. At generation time, we find appropriate speech units in the database. Finally, we use signal processing techniques to paste the chunks together. In parametric speech synthesis (**?**), we extract vocoder parameters from speech signals. Then, we train a generative model that will output these parameters given the desired linguistic features. A maximum likelihood parameter estimation algorithm is used to generate the parameters. Finally, the waveform is constructed using the vocoder. The conventional approach to statistical parametric speech synthesis uses tree-clustered context-dependent hidden Markov models (HMMs) to model the probability den-

1. Statistics based on the average speaking rate of a set of TED talk speakers `http://sixminutes.dlugan.com/speaking-rate/`

sities of vocoder parameters. For a more detailed review of traditional models of speech synthesis, we recommend (**?**).

Recently, there have been advances by using neural networks to model vocoder parameters. **?** propose using (feed-forward) Deep Neural Networks to model the acoustic parameters of the vocoder. **?** present a lightweight model for mobile devices using Recurrent Neural Networks (RNN). **??** present a comprehensive review of the progress made by using neural networks for acoustic modeling.

Traditionally, the high-dimensionality of raw audio signal is dealt with by first compressing it into spectral or hand-engineered features and defining the generative model over these features. However, when the generated signal is eventually decompressed into audio waveforms, the sample quality is often degraded and requires extensive domain-expert corrective measures. This results in complicated signal processing pipelines that are to adapt to new tasks or domains. Furthermore, there have been a few recent attempts to take away the vocoder and model the raw waveform directly (**???**).

In Chapter 3, we propose a novel model for unconditional audio generation based on generating one audio sample at a time: SampleRNN. We show that our model, which profits from combining memory-less modules, namely autoregressive multilayer perceptrons, and stateful recurrent neural networks in a hierarchical structure is able to capture underlying sources of variation in the temporal sequences over very long time spans, on three datasets of different nature. Human evaluation on the generated samples indicate that our model is preferred over competing models. We also show how each component of the model contributes to the exhibited performance.

Furthermore, in Chapter 4, we present Char2Wav, an end-to-end model for speech synthesis. Char2Wav has two components: a **reader** and a **neural vocoder**. The *reader* is an encoder-decoder model with attention. The encoder is a bidirectional recurrent neural network (RNN) that accepts text or phonemes as inputs, while the decoder is a recurrent neural network with attention that produces vocoder acoustic features. *Neural vocoder* refers to a conditional extension of SampleRNN which generates raw waveform samples from intermediate representations.

2 Adasecant

A Robust Adaptive Stochastic Gradient Method for Deep Learning. Caglar Gulcehre*, Jose Sotelo*, Marcin Moczulski, Yoshua Bengio. [1]

Personal Contribution. My main contribution to the project was the ablation study. I joined this project after the algorithm had already been theoretically developed. The algorithm was developed by Caglar and Marcin supervised by Yoshua. They presented a preliminary v ersion of this paper in the NIPS Workshop in Optimization, 2015. I joined the team to update the paper with the comments received in the workshop. For this, I coded the handwriting synthesis experiment and performed the ablation study and the comparison with Adasecant. Furthermore, I rewrote most of the Results section. Finally, I did minor edits of the other sections. This chapter was accepted in the IJCNN, 2017.

Affiliations

Caglar Gulcehre*, MILA, Département d'Informatique et de Recherche Opérationnelle, Université de Montréal

Jose Sotelo*, MILA, Département d'Informatique et de Recherche Opérationnelle, Université de Montréal

Marcin Moczulski, University of Oxford

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

Funding We thank the computational resources provided by Compute Canada and Calcul Québec. This work has been partially supported by NSERC, CIFAR, and Canada Research Chairs, Project TIN2013-41751, grant 2014-SGR-221.

1. * denotes equal contribution.

2.1 Abstract

Stochastic gradient algorithms are the main focus of large-scale optimization problems and led to important successes in the recent advancement of the deep learning algorithms. The convergence of SGD depends on the careful choice of learning rate and the amount of the noise in stochastic estimates of the gradients. In this paper, we propose an adaptive learning rate algorithm, which utilizes stochastic curvature information of the loss function for automatically tuning the learning rates. The information about the element-wise curvature of the loss function is estimated from the local statistics of the stochastic first order gradients. We further propose a new variance reduction technique to speed up the convergence. In our experiments with deep neural networks, we obtained better performance compared to the popular stochastic gradient algorithms.[2]

2.2 Introduction

We develop an automatic stochastic gradient algorithm which reduces the burden of extensive hyper-parameter search for the optimizer. Our proposed algorithm exploits a lower variance estimator of curvature of the cost function and uses it to obtain an automatically tuned adaptive learning rate for each parameter.

In deep learning and numerical optimization literature, several papers suggest using a diagonal approximation of the Hessian (second derivative matrix of the cost function with respect to parameters), in order to estimate optimal learning rates for stochastic gradient descent over high dimensional parameter spaces **???**. A fundamental advantage of using such approximation is that inverting such approximation can be a trivial and cheap operation. However generally, for neural networks, the inverse of the diagonal Hessian is usually a bad approximation of the diagonal of the inverse of Hessian. For example, obtaining a diagonal approximation of the Hessian are the Gauss-Newton matrix **?** or by finite differences **?**. Such estimations may however be very sensitive to the noise coming from the Monte-Carlo estimates of the gradients. **?** suggested a reliable way to estimate the local

2. This paper is an extension/update of our previous paper **?**.

curvature in the stochastic setting by keeping track of the variance and average of the gradients.

We propose a different approach: instead of using a diagonal estimate of the Hessian, to estimate curvature along the direction of the gradient and we apply a new variance reduction technique to compute it reliably. By using root mean square statistics, the variance of gradients are reduced adaptively with a simple transformation. We keep track of the estimation of curvature using a technique similar to that proposed by **?**, which uses the variability of the expected loss. Standard adaptive learning rate algorithms only scale the gradients, but regular Newton-like second order methods, can perform more complicate transformations, e.g. rotating the gradient vector. Newton and quasi-newton methods can also be invariant to affine transformations in the parameter space. The **AdaSecant** algorithm is basically a stochastic rank-1 quasi-Newton method. But in comparison with other adaptive learning algorithms, instead of just scaling the gradient of each parameter, AdaSecant can also perform an affine transformation on them.

2.3 Directional Secant Approximation

Directional Newton is a method proposed for solving equations with multiple variables**?**. The advantage of the directional Newton method compared to Newton's method is that, it does not require a matrix inversion and still maintains a quadratic rate of convergence.

In this paper, we develop a second-order directional Newton method for nonlinear optimization. Step-size $\mathbf{t}^k$ of update Δ^k for step k can be written as if it was a diagonal matrix:

$$\Delta^k = -\mathbf{t}^k \odot \nabla_{\boldsymbol{\theta}} \mathrm{f}(\boldsymbol{\theta}^k), \tag{2.1}$$

$$= -\operatorname{diag}(\mathbf{t}^k)\nabla_{\boldsymbol{\theta}} \mathrm{f}(\boldsymbol{\theta}^k), \tag{2.2}$$

$$= -\operatorname{diag}(\mathbf{d}^k)(\operatorname{diag}(\mathbf{H}\mathbf{d}^k))^{-1}\nabla_{\boldsymbol{\theta}} \mathrm{f}(\boldsymbol{\theta}^k). \tag{2.3}$$

where $\boldsymbol{\theta}^k$ is the parameter vector at update k, f is the objective function and $\mathbf{d}^k$ is a unit vector of direction that the optimization algorithm should follow. Denoting by $\mathbf{h}_i = \nabla_{\boldsymbol{\theta}} \frac{\partial \mathrm{f}(\boldsymbol{\theta}^k)}{\partial \theta_i}$ the i^{th} row of the Hessian matrix $\mathbf{H}$ and by $\nabla_{\boldsymbol{\theta}_i} f(\boldsymbol{\theta}^k)$ the i^{th} element of the gradient vector at update k, a reformulation of Equation 2.1 for each diagonal element of the step-size $\mathrm{diag}(\mathbf{t}^k)$ is:

$$\begin{aligned} \Delta_i^k &= -t_i^k \nabla_{\boldsymbol{\theta}_i} \mathrm{f}(\boldsymbol{\theta}^k), && (2.4)\\ &= -d_i^k \frac{\nabla_{\boldsymbol{\theta}_i} \mathrm{f}(\boldsymbol{\theta}^k)}{\mathbf{h}_i^k \mathbf{d}^k}. && (2.5) \end{aligned}$$

so effectively

$$t_i^k = \frac{d_i^k}{\mathbf{h}_i^k \mathbf{d}^k}. \tag{2.6}$$

We can approximate the per-parameter learning rate t_i^k following **?**:

$$\begin{aligned} t_i^k &= \frac{d_i^k}{\mathbf{h}_i^k \mathbf{d}^k}, && (2.7)\\ &= \lim_{|\Delta_i^k| \to 0} \frac{\Delta_i^k}{\nabla_{\boldsymbol{\theta}_i} \mathrm{f}(\boldsymbol{\theta}^k + \Delta^k) - \nabla_{\boldsymbol{\theta}_i} \mathrm{f}(\boldsymbol{\theta}^k)}, \text{for every } i. && (2.8) \end{aligned}$$

Please note that alternatively one might use the R-op to compute the Hessian-vector product for the denominator in Equation 2.7 (**?**).

To choose a good direction $\mathbf{d}^k$ in the stochastic setting, we use block-normalized gradient vector that the parameters of each layer is considered as a block and for each weight matrix $\mathbf{W}_k^i$ and bias vector $\mathbf{b}_k^i$ for $\boldsymbol{\theta} = \{\mathbf{W}_k^i, \mathbf{b}_k^i\}_{i=1 \cdots k}$ at each layer i and update k, $\mathbf{d}_k = \left[\mathbf{d}^k_{\mathbf{W}_k^0} \mathbf{d}^k_{\mathbf{b}_k^0} \cdots \mathbf{d}^k_{\mathbf{b}_k^l}\right]$ for a neural network with l layers.

The update step is defined as $\Delta_i^k = t_i^k d_i^k$. The per-parameter learning rate t_i^k can be estimated with the finite difference approximation,

$$t_i^k \approx \frac{\Delta_i^k}{\nabla_{\boldsymbol{\theta}_i} \mathrm{f}(\boldsymbol{\theta}^k + \Delta^k) - \nabla_{\boldsymbol{\theta}_i} \mathrm{f}(\boldsymbol{\theta}^k)}, \tag{2.9}$$

since, in the vicinity of the quadratic local minima,

$$\nabla_{\boldsymbol{\theta}} \mathrm{f}(\boldsymbol{\theta}^k + \Delta^k) - \nabla_{\boldsymbol{\theta}} \mathrm{f}(\boldsymbol{\theta}^k) \approx \mathbf{H}^k \Delta^k, \tag{2.10}$$

We can therefore recover $\mathbf{t}^k$ as

$$\mathbf{t}^k = \mathrm{diag}(\Delta^k)(\mathrm{diag}(\mathbf{H}^k \Delta^k))^{-1}. \tag{2.11}$$

The directional secant method basically scales the gradient of each parameter with the curvature along the direction of the gradient vector and it is numerically stable.

2.4 Relationship to the Diagonal Approximation to the Hessian

Our secant approximation of the gradients are also very closely tied to diagonal approximation of the Hessian matrix. Considering that i^{th} diagonal entry of the Hessian matrix can be denoted as, $\mathbf{H}_{ii} = \frac{\partial^2 \mathrm{f}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}_i^2}$. By using the finite differences, it is possible to approximate this with as in Equation 2.12,ıı

$$\mathbf{H}_{ii} = \lim_{|\Delta| \to 0} \frac{\nabla_{\boldsymbol{\theta}_i} \mathrm{f}(\boldsymbol{\theta} + \Delta) - \nabla_{\boldsymbol{\theta}_i} \mathrm{f}(\boldsymbol{\theta})}{\Delta_i}, \tag{2.12}$$

Assuming that the diagonal of the Hessian is denoted with $\mathbf{A}$ matrix, we can see the equivalence:

$$\mathbf{A} \approx \mathrm{diag}(\nabla_{\boldsymbol{\theta}} \mathrm{f}(\boldsymbol{\theta} + \Delta) - \nabla_{\boldsymbol{\theta}} \mathrm{f}(\boldsymbol{\theta}))\, \mathrm{diag}(\Delta)^{-1}. \tag{2.13}$$

The Equation 2.13 can be easily computed in a stochastic setting from the consecutive minibatches.

2.5 Variance Reduction for Robust Stochastic Gradient Descent

Variance reduction techniques for stochastic gradient estimators have been well-studied in the machine learning literature. Both **?** and **?** proposed new ways of dealing with this problem. In this paper, we proposed a new variance reduction technique for stochastic gradient descent that relies only on basic statistics related to the gradient. Let g_i refer to the i^{th} element of the gradient vector $\mathbf{g}$ with respect to the parameters $\boldsymbol{\theta}$ and $\mathrm{E}[\cdot]$ be an expectation taken over minibatches and different trajectories of parameters.

We propose to apply the following transformation to reduce the variance of the stochastic gradients:

$$\tilde{g_i} = \frac{g_i + \gamma_i \mathrm{E}[g_i]}{1+\gamma_i}, \tag{2.14}$$

where γ_i is strictly a positive real number. Let us note that:

$$\mathrm{E}[\tilde{g_i}] = \mathrm{E}[g_i] \text{ and } \mathrm{Var}(\tilde{g_i}) = \frac{1}{(1+\gamma_i)^2}\mathrm{Var}(g_i). \tag{2.15}$$

The variance is reduced by a factor of $(1+\gamma_i)^2$ compared to $\mathrm{Var}(g_i)$.

In practice we do not have access to $\mathrm{E}[g_i]$, therefore a biased estimator $\overline{g_i}$ based on past values of g_i will be used instead. We can rewrite the $\tilde{g_i}$ as:

$$\tilde{g_i} = \frac{1}{1+\gamma_i}g_i + (1 - \frac{1}{1+\gamma_i})\mathrm{E}[g_i], \tag{2.16}$$

After substitution $\beta_i = \frac{1}{1+\gamma_i}$, we will have:

$$\tilde{g_i} = \beta_i g_i + (1-\beta_i)\mathrm{E}[g_i]. \tag{2.17}$$

By adapting γ_i or β_i, it is possible to control the influence of high variance, unbiased g_i and low variance, biased $\overline{g_i}$ on $\tilde{g_i}$. Denoting by $\mathbf{g}'$ the stochastic gradient obtained on the next minibatch, the γ_i that well balances those two influences is the one that keeps the $\tilde{g_i}$ as close as possible to the true gradient $\mathrm{E}[g_i']$ with g_i' being the only sample of $\mathrm{E}[g_i']$ available. We try to find a regularized β_i, in order to obtain a smoother estimate of it and this yields us a more stable estimate of β_i. λ is the regularization coefficient for β.

$$\arg\min_{\beta_i} \mathrm{E}[||\tilde{g_i} - g_i'||_2^2] + \lambda(\beta_i)^2. \tag{2.18}$$

It can be shown that this a convex problem in β_i with a closed-form solution (details in appendix) and we can obtain the γ_i from it:

$$\gamma_i = \frac{\mathrm{E}[(g_i - g_i')(g_i - \mathrm{E}[g_i])]}{\mathrm{E}[(g_i - \mathrm{E}[g_i])(g^{i\prime} - \mathrm{E}[g_i]))] + \lambda}, \tag{2.19}$$

As a result, to estimate γ for each dimension, we keep track of a estimation of $\frac{\mathrm{E}[(g_i-g_i')(g_i-\mathrm{E}[g_i])]}{\mathrm{E}[(g_i-\mathrm{E}[g_i])(g_i'-\mathrm{E}[g_i]))]+\lambda}$ during training. The necessary and sufficient condition here, for the variance reduction is to keep γ positive, to achieve a positive estimate of γ we used the root mean square statistics for the expectations.

2.6 Blockwise Gradient Normalization

It is very well-known that the repeated application of the non-linearities can cause the gradients to vanish **??**. Thus, in order to tackle this problem, we normalize the gradients coming into each block-layer to have norm 1. Assuming the normalized gradient can be denoted with $\tilde{\mathbf{g}}$, it can be computed as, $\tilde{\mathbf{g}} = \frac{\mathbf{g}}{||\mathrm{E}[\mathbf{g}]||_2}$. We estimate, $\mathrm{E}[\mathbf{g}]$ via moving averages.

Blockwise gradient normalization of the gradient adds noise to the gradients, but in practice we did not observe any negative impact of it. We conjecture that this is due to the angle between the stochastic gradient and the block-normalized gradient still being less than 90 degrees.

2.7 Adaptive Step-size in Stochastic Case

In the stochastic gradient case, the step-size of the directional secant can be computed by using an expectation over the minibatches:

$$\mathrm{E}_k[t_i] = \mathrm{E}_k[\frac{\Delta_i^k}{\nabla_{\boldsymbol{\theta}_i}\mathrm{f}(\boldsymbol{\theta}^k + \Delta^k) - \nabla_{\boldsymbol{\theta}_i}\mathrm{f}(\boldsymbol{\theta}^k)}]. \tag{2.20}$$

The $E_k[\cdot]$ that is used to compute the secant update, is taken over the minibatches at the past values of the parameters.

Computing the expectation in Equation2.20 was numerically unstable in stochastic setting. We decided to use a more stable second order Taylor approximation of Equation 2.20 around $(\sqrt{\mathrm{E}_k[(\alpha_i^k)^2]}, \sqrt{\mathrm{E}_k[(\Delta_i^k)^2]})$, with $\alpha_i^k = \nabla_{\boldsymbol{\theta}_i}\mathrm{f}(\boldsymbol{\theta}^k + \Delta^k) - \nabla_{\boldsymbol{\theta}_i}\mathrm{f}(\boldsymbol{\theta}^k)$. Assuming $\sqrt{\mathrm{E}_k[(\alpha_i^k)^2]} \approx \mathrm{E}_k[\alpha_i^k]$ and $\sqrt{\mathrm{E}_k[(\Delta_i^k)^2]} \approx \mathrm{E}_k[\Delta_i^k]$ we obtain always non-negative approximation of $\mathrm{E}_k[t_i]$:

$$\mathrm{E}_k[t_i] \approx \frac{\sqrt{\mathrm{E}_k[(\Delta_i^k)^2]}}{\sqrt{\mathrm{E}_k[(\alpha_i^k)^2]}} - \frac{\mathrm{Cov}(\alpha_i^k, \Delta_i^k)}{\mathrm{E}_k[(\alpha_i^k)^2]}. \tag{2.21}$$

In our experiments, we used a simpler approximation, which in practice worked as well as formulations in Equation2.21:

$$\mathrm{E}_k[t_i] \approx \frac{\sqrt{\mathrm{E}_k[(\Delta_i^k)^2]}}{\sqrt{\mathrm{E}_k[(\alpha_i^k)^2]}} - \frac{\mathrm{E}_k[\alpha_i^k \Delta_i^k]}{\mathrm{E}_k[(\alpha_i^k)^2]}. \tag{2.22}$$

2.8 Algorithmic Details

2.8.1 Approximate Variability

To compute the moving averages as also adopted by **?**, we used an algorithm to dynamically decide the time constant based on the step size being taken. As a result algorithm that we used will give bigger weights to the updates that have large step-size and smaller weights to the updates that have smaller step-size.

By assuming that $\bar{\Delta}_i[k] \approx \mathrm{E}[\Delta_i]_k$, the moving average update rule for $\bar{\Delta}_i[k]$ can be written as,

$$\bar{\Delta}_i^2[k] = (1 \ - \ \tau_i^{-1}[k])\bar{\Delta}_i^2[k-1] + \tau_i^{-1}[k](t_i^k \tilde{\mathbf{g}}_i^k), \tag{2.23}$$

and,

$$\bar{\Delta}_i[k] = \sqrt{\bar{\Delta}_i^2[k]}. \tag{2.24}$$

This rule for each update assigns a different weight to each element of the gradient vector . At each iteration a scalar multiplication with τ_i^{-1} is performed and τ_i is adapted using the following equation:

$$\tau_i[k] = (1 \ - \ \frac{\mathrm{E}[\Delta_i]_{k-1}^2}{\mathrm{E}[(\Delta_i)^2]_{k-1}})\tau_i[k-1] + 1 \ . \tag{2.25}$$

2.8.2 Outlier Gradient Detection

Our algorithm is very similar to **?**, but instead of incrementing $\tau_i[t+1]$ when an outlier is detected, the time-constant is reset to 2.2. Note that when $\tau_i[t+1] \approx 2$, this assigns approximately the same amount of weight to the current and the average of previous observations. This mechanism made learning more stable, because without it outlier gradients saturate τ_i to a large value.

2.8.3 Variance Reduction

The correction parameters γ_i (Equation2.19) allows for a fine-grained variance reduction for each parameter independently. The noise in the stochastic gradient methods can have advantages both in terms of generalization and optimization. It introduces an exploration and exploitation trade-off, which can be controlled by upper bounding the values of γ_i with a value ρ_i, so that thresholded $\gamma_i' = \min(\rho_i, \gamma_i)$.

We block-wise normalized the gradients of each weight matrix and bias vectors in $\mathbf{g}$ to compute the $\tilde{\mathbf{g}}$ as described in Section 2.3. That makes AdaSecant scale-invariant, thus more robust to the scale of the inputs and the number of the layers of the network. We observed empirically that it was easier to train very deep neural networks with block normalized gradient descent. In our experiments, we fixed λ to $1e-5$.

2.9 Improving Convergence

Classical convergence results for SGD are based on the conditions:

$$\sum_i (\eta^{(i)})^2 < \infty \text{ and } \sum_i \eta^{(i)} = \infty \tag{2.26}$$

such that the learning rate $\eta^{(i)}$ should decrease **?**. Due to the noise in the estimation of adaptive step-sizes for AdaSecant, the convergence would not be guaranteed. To ensure it, we developed a new variant of Adagrad **?** with thresholding, such that each scaling factor is lower bounded by 1. Assuming a_i^k is the accumulated norm of all past gradients for i^{th} parameter at update k, it is thresholded from below ensuring that the algorithm will converge:

$$a_i^k = \sqrt{\sum_{j=0}^{k} (g_i^j)^2}, \tag{2.27}$$

and

$$\rho_i^k = \text{maximum}(1, a_i^k), \tag{2.28}$$

giving

$$\Delta_i^k = \frac{1}{\rho_i} \eta_i^k \tilde{\mathbf{g}}_i^k. \tag{2.29}$$

In the initial stages of training, accumulated norm of the per-parameter gradients can be less than 1. If the accumulated per-parameter norm of a gradient is less than 1, Adagrad will augment the learning-rate determined by AdaSecant for that update, i.e. $\frac{\eta_i^k}{\rho_i^k} > \eta_i^k$ where $\eta_i^k = \mathrm{E}_k[t_i^k]$ is the per-parameter learning rate determined by AdaSecant. This behavior tends to create unstabilities during the training with AdaSecant. Our modification of the Adagrad algorithm is to ensure that, it will reduce the learning rate determined by the AdaSecant algorithm at each update, i.e. $\frac{\eta_i^k}{\rho_i^k} \leq \eta_i^k$ and the learning rate will be bounded. At the beginning of the training, parameters of a neural network can get 0-valued gradients, e.g. in the existence of dropout and ReLU units. However this phenomena can cause the per-parameter learning rate scaled by Adagrad to be unbounded.

In Algorithm 1, we provide a simple pseudo-code of the AdaSecant algorithm.

2.10 Experiments

We have run experiments on character-level PTB with GRU units, on MNIST with Maxout Networks **?** and on handwriting synthesis using the IAM-OnDB dataset **?**. We compare AdaSecant with popular stochastic gradient learning algorithms: Adagrad, RMSProp **?**, Adadelta **?**, Adam **?** and SGD+momentum (with linearly decaying learning rate). AdaSecant performs as well or better as carefully tuned algorithms for all these different tasks.

Algorithm 1: AdaSecant: minibatch-AdaSecant for adaptive learning rates with variance reduction

```
repeat
    draw $n$ samples, compute the gradients $\mathbf{g}^{(j)}$ where $\mathbf{g}^{(j)} \in \mathcal{R}^n$ for each
    minibatch $j$, $\mathbf{g}^{(j)}$ is computed as, $\frac{1}{n}\sum_{k=1}^{n} \nabla_{\boldsymbol{\theta}}^{(k)} \mathrm{f}(\boldsymbol{\theta})$
    estimate $\mathrm{E}[\mathbf{g}]$ via moving averages.
    block-wise normalize gradients of each weight matrix and bias vector
    for parameter $i \in \{1, \ldots, n\}$ do
        compute the correction term by using, $\gamma_i^k = \frac{\mathrm{E}[(g_i - g_i')(g_i - \mathrm{E}[g_i])]_k}{\mathrm{E}[(g_i - \mathrm{E}[g_i])(g_i' - \mathrm{E}[g_i]))]_k}$
        compute corrected gradients $\tilde{g}_i = \frac{g_i + \gamma_\mathrm{i} \mathrm{E}[g_i]}{1 + \gamma_\mathrm{i}}$
        if $|g_i^{(j)} - \mathrm{E}[g_i]| > 2\sqrt{\mathrm{E}[(g_i)^2] - (\mathrm{E}[g_i])^2}$ or $\left|\alpha_i^{(j)} - \mathrm{E}[\alpha_i]\right| >$
        $2\sqrt{\mathrm{E}[(\alpha_i)^2] - (\mathrm{E}[\alpha_i])^2}$ then
            reset the memory size for outliers $\tau_i \leftarrow 2.2$
        end
        update moving averages according to Equation 2.23
        estimate learning rate $\eta_i^{(j)} \leftarrow \frac{\sqrt{\mathrm{E}_k[(\Delta_i^{(k)})^2]}}{\sqrt{\mathrm{E}_k[(\alpha_i^k)^2]}} - \frac{\mathrm{E}_k[\alpha_i^k \Delta_i^k]}{\mathrm{E}_k[(\alpha_i^k)^2]}$
        update memory size as in Equation 2.25
        update parameter $\theta_i^j \leftarrow \theta_i^{j-1} - \eta_i^{(j)} \cdot \tilde{g}_i^{(j)}$
    end
until stopping criterion is met;
```

2.10.1 Ablation Study

In this section, we decompose the different parts of the algorithm to measure the effect they have in the performance. For this comparison, we trained a model to learn handwriting synthesis on IAM-OnDB dataset. Our model follows closely the architecture introduced in **?** with two modifications. First, we use one recurrent layer of size 400 instead of three. Second, we use GRU **?** units instead of LSTM **?** units. Also, we use a different symbol for each of the 87 different characters in the dataset. The code for this experiment is available online.[3]

We tested different configurations that included taking away the use of Variance Reduction (VR), Adagrad (AG), Block Normalization (BN), and Outlier De-

3. https://github.com/sotelo/scribe

Model	Train Log-Loss	Valid Log-Loss
Adam with 3e-4 learning rate	-1.827	-1.743
Adam with 1e-4 learning rate	-1.780	-1.713
Adam with 5e-4 learning rate	-1.892	-1.773
AdaSecant	**-1.881**	-1.744
AdaSecant, no VR	-1.876	-1.743
AdaSecant, no AG	-1.867	-1.738
AdaSecant, no BN	-1.857	-1.784
AdaSecant, no OD	-1.780	-1.726
AdaSecant, no VR, no AG	-1.848	-1.744
AdaSecant, no VR, no BN	-1.844	-1.777
AdaSecant, no VR, no OD	-1.479	-1.442
AdaSecant, no AG, no BN	-1.878	**-1.786**
AdaSecant, no AG, no OD	-1.723	-1.674
AdaSecant, no BN, no OD	-1.814	-1.764
AdaSecant, no AG, no BN, no OD	-1.611	-1.573
AdaSecant, no VR, no BN, no OD	-1.531	-1.491
AdaSecant, no VR, no AG, no OD	unstable	unstable
AdaSecant, no VR, no AG, no BN	-1.862	1.75

Table 2.1 – Summary of results for the handwriting experiment. We report the best validation log-loss that we found for each model using early stopping. We also report the corresponding train log-loss. In all cases, the log-loss is computed per data point.

tection (OD). Also, we compared against ADAM **?** with different learning rates in Figure 2.1. There, we observe that adasecant performs as well as Adam with a carefully tuned learning rate.

In Figure 2.2, we disable each of the four components of the algorithm. We find that BN provides a small, but constant advantage in performance. OD is also important for the algorithm. Disabling OD makes training more noisy and unstable and gives worse results. Disabling VR also makes training unstable. AG has the least effect in the performance of the algorithm. Furthermore, disabling more than one component makes training even more unstable in the majority of scenarios. A summary of the results is available in Table 2.1. In all cases, we use early stopping on the validation log-loss. Furthermore, we present the train log-loss corresponding to the best validation loss as well. Let us note that the log-loss is computed per data point.

2.10.2 PTB Character-level LM

We have run experiments with GRU-RNN**?** on PTB dataset for character-level language modeling over the subset defined in **?**. On this task, we use 400 GRU units with minibatch size of 20. We train the model over the sequences of length 150. For AdaSecant, we have not run any hyperparmeter search, but for Adam we run a hyperparameter search for the learning rate and gradient clipping. The learning rates are sampled from log-uniform distribution between $1e-1$ and $6e-5$. Gradient clipping threshold is sampled uniformly between 1.2 to 20. We have evaluated 20 different pairs of randomly-sampled learning rates and gradient clipping thresholds. The rest of the hyper-parameters are fixed to their default values. We use the model with the best validation error for Adam. For AdaSecant algorithm, we fix all the hyperparameters to their default values. The learning curves for the both algorithms are shown in Figure 2.3.

2.10.3 MNIST with Maxout Networks

The results are summarized in Figure 2.4 and we show that AdaSecant converges as fast or faster than other techniques, including the use of hand-tuned global learning rate and momentum for SGD, RMSprop, and Adagrad. In our experiments with AdaSecant algorithm, adaptive momentum term γ_i^k was clipped at 1.8. In 2-layer Maxout network experiments for SGD-momentum experiments, we used the best hyper-parameters reported by **?**, for RMSProp and Adagrad, we crossvalidated learning rate for 15 different learning rates sampled uniformly from the log-space. We crossvalidated 30 different pairs of momentum and learning rate for SGD+momentum, for RMSProp and Adagrad, we crossvalidated 15 different learning rates sampled them from log-space uniformly for deep maxout experiments.

2.11 Conclusion

We described a new stochastic gradient algorithm with adaptive learning rates that is fairly insensitive to the tuning of the hyper-parameters and doesn't require tuning of learning rates. Furthermore, the variance reduction technique we proposed improves the convergence when the stochastic gradients have high variance. Our algorithm performs as well or better than other popular, carefully-tuned stochastic gradient algorithms. We also present a comprehensive ablation study where we show the effects and importance of each of the elements of our algorithm. As future work, we should try to find theoretical convergence properties of the algorithm to understand it better analytically.

2.12 Appendix

2.12.1 Derivation of Equation 2.18

$$\frac{\partial \mathrm{E}[(\beta_i g_i + (1-\beta_i)\mathrm{E}[g_i] - g_i')^2]}{\partial \beta_i} + \lambda \beta_i^2 = 0$$

$$\mathrm{E}[(\beta_i g_i + (1-\beta_i)\mathrm{E}[g_i] - g_i') \frac{\partial(\beta_i g_i + (1-\beta_i)\mathrm{E}[g_i] - g_i')}{\partial \beta_i}] + \lambda \beta_i = 0$$

$$\mathrm{E}[(\beta_i g_i + (1-\beta_i)\mathrm{E}[g_i] - g'_i)(g_i - \mathrm{E}[g_i])] + \lambda\beta_i = 0$$

$$\begin{aligned} \mathrm{E}[(\beta_i g_i(g_i - \mathrm{E}[g_i]) + (1-\beta_i)\mathrm{E}[g_i](g_i - \mathrm{E}[g_i]) \\ - g'_i(g_i - \mathrm{E}[g_i])] + \lambda\beta_i && = 0 \end{aligned}$$

$$\begin{aligned} \beta_i &= \frac{\mathrm{E}[(g_i - \mathrm{E}[g_i])(g'_i - \mathrm{E}[g_i])]}{\mathrm{E}[(g_i - \mathrm{E}[g_i])(g_i - \mathrm{E}[g_i])] + \lambda} \\ &= \frac{\mathrm{E}[(g_i - \mathrm{E}[g_i])(g'_i - \mathrm{E}[g_i])]}{\mathrm{Var}(g_i) + \lambda} \end{aligned}$$

2.12.2 Further Experimental Details

In Figure 2.5, we analyzed the effect of using different minibatch sizes for AdaSecant and compared its convergence with Adadelta in wall-clock time. For minibatch size 100 AdaSecant was able to reach the almost same training negative log-likelihood as Adadelta after the same amount of time, but its convergence took much longer. With minibatches of size 500 AdaSecant was able to converge faster in wallclock time to a better local minima.

2.12.3 More decomposition experiments

We have run experiments with the different combinations of the components of the algorithm. We show those results on handwriting synthesis with IAM-OnDB dataset. The results can be observed from Figure 2.6, Figure 2.7, Figure 2.8, and Figure 2.9 deactivating the components leads to a more unstable training curve in the majority of scenarios.

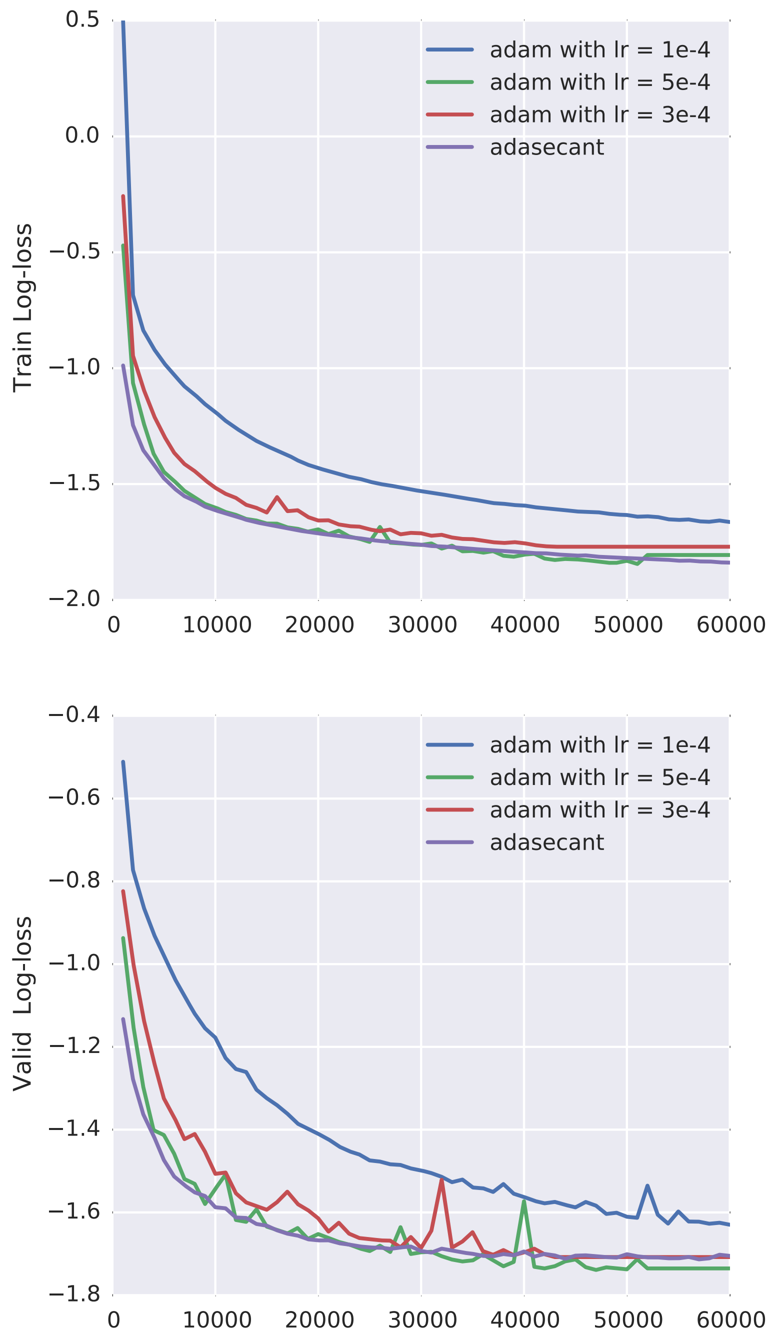
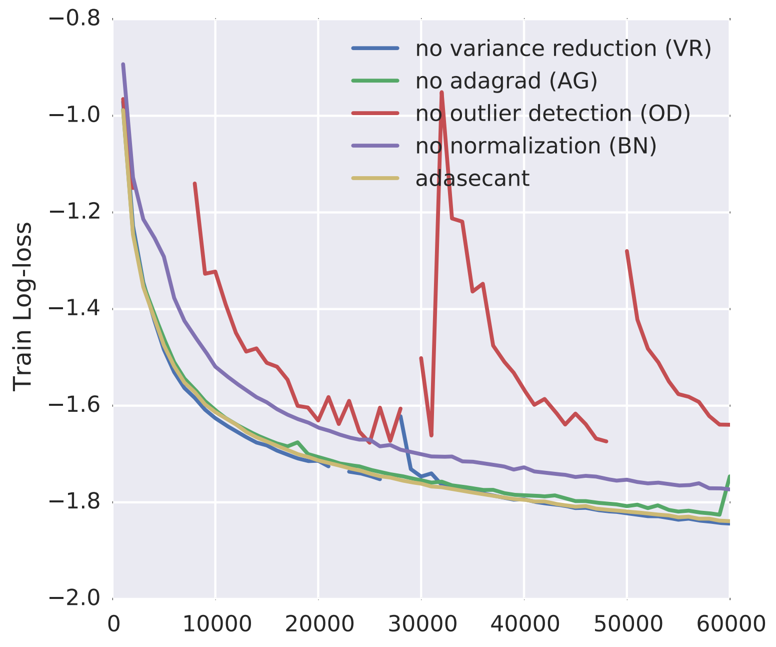


Figure 2.1 – Baseline comparison against Adam. AdaSecant performs as well as Adam with a carefully tuned learning rate.



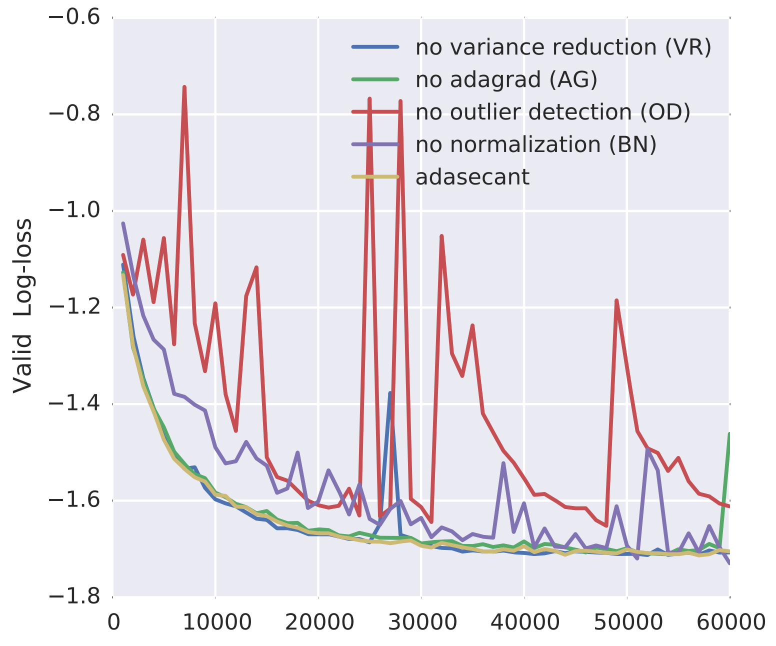


Figure 2.2 – Deactivating one component at a time. BN provides a small but constant advantage in performance. OD is important for the algorithm. Deactivating it makes training more noisy and unstable and gives worse results. Deactivating VR also makes training unstable.

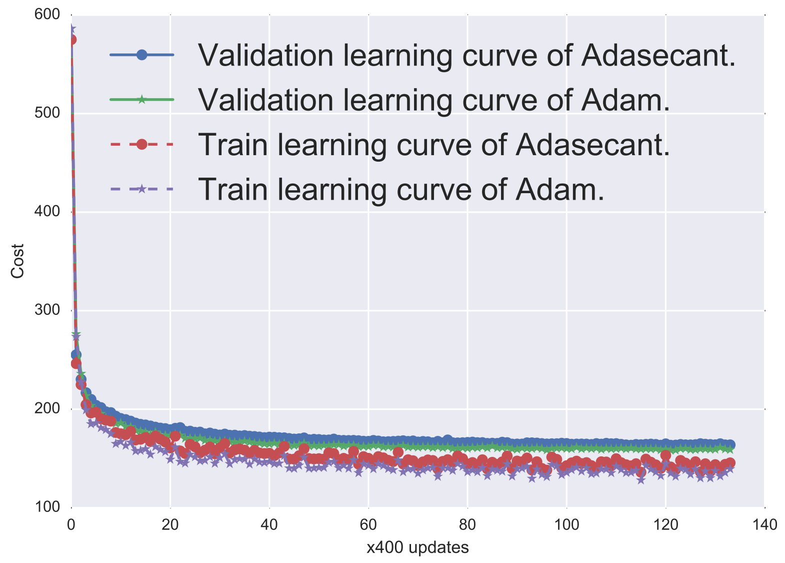


Figure 2.3 – Learning curves for the very well-tuned Adam vs AdaSecant algorithm without any hyperparameter tuning. AdaSecant performs very close to the very well-tuned Adam on PTB character-level language modeling task. This shows us the robustness of the algorithm to its hyperparameters.

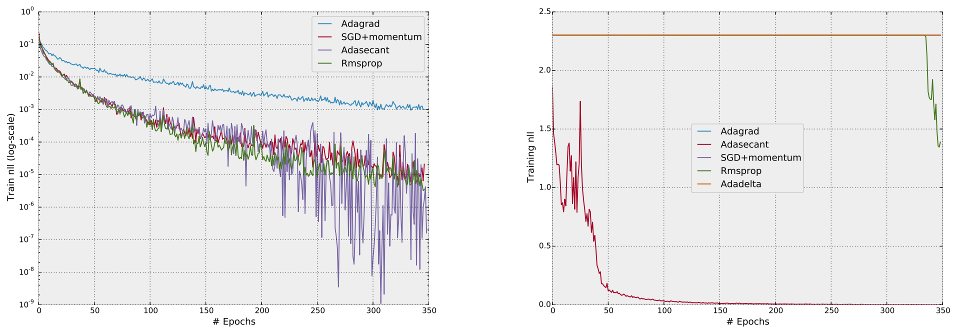


Figure 2.4 – Comparison of different stochastic gradient algorithms on MNIST with Maxout Networks. Both a) and b) are trained with dropout and maximum column norm constraint regularization on the weights. Networks are initialized with weights sampled from a Gaussian distribution with 0 mean and standard deviation of 0.05. In both experiments, the proposed algorithm, Adasecant, seems to be converging faster and arrives to a better minima in training set. We trained both networks for 350 epochs over the training set.

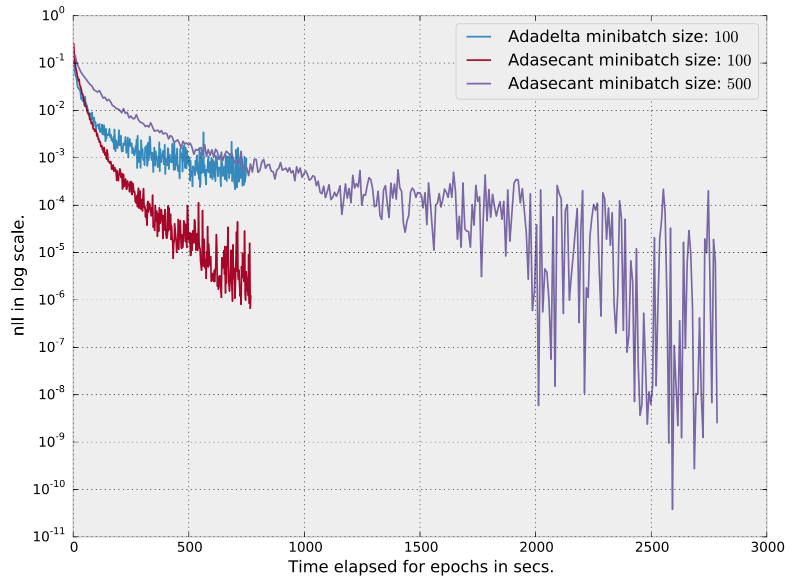
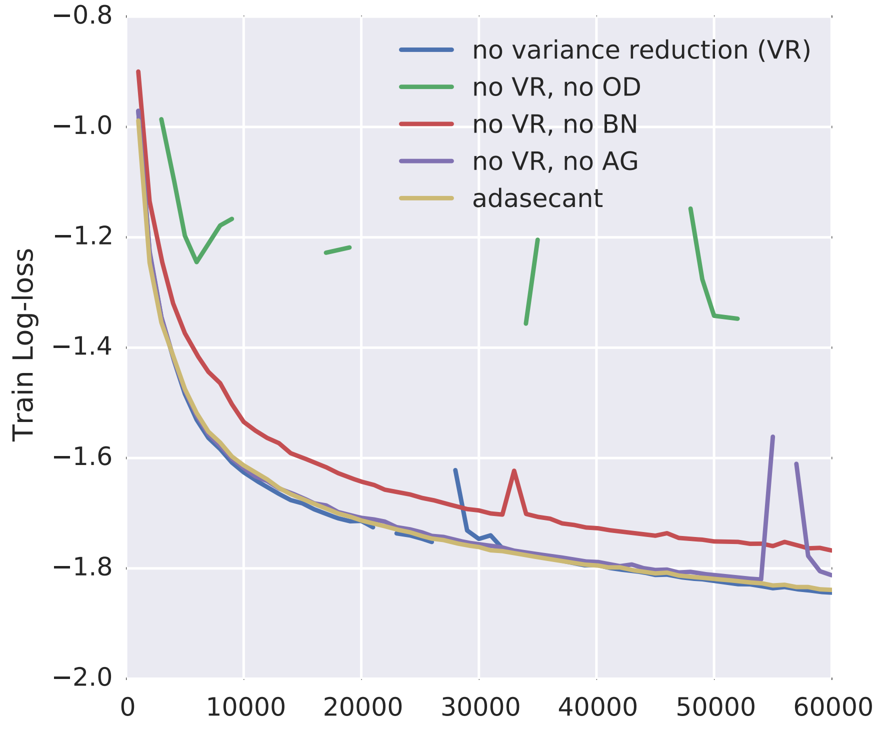


Figure 2.5 – In this plot, we compared AdaSecant trained by using minibatch size of 100 and 500 with adadelta using minibatches of size 100. We performed these experiments on MNIST with 2-layer maxout MLP using dropout.



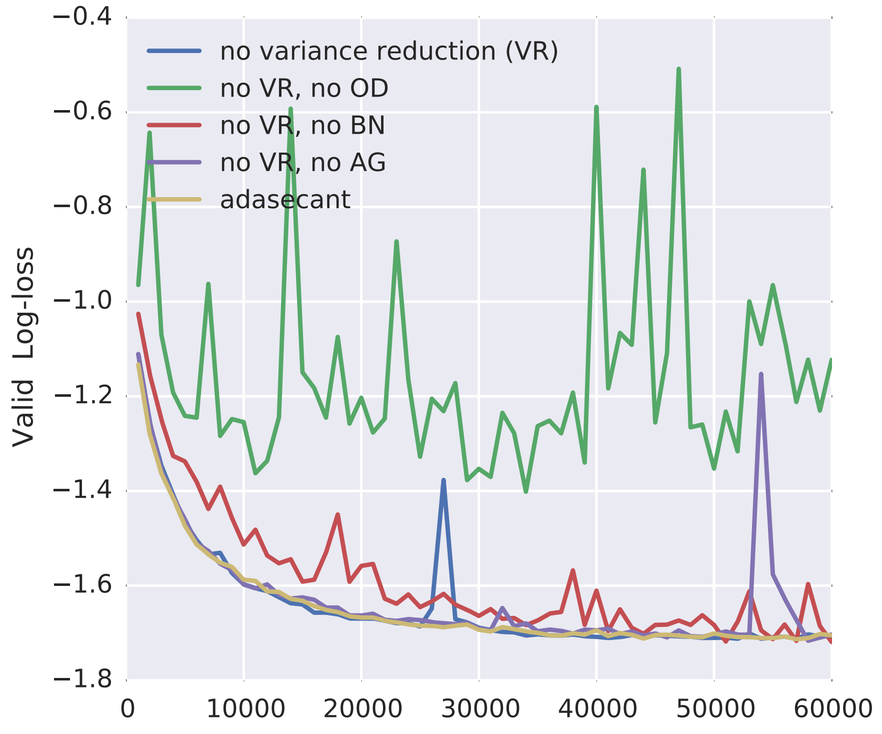
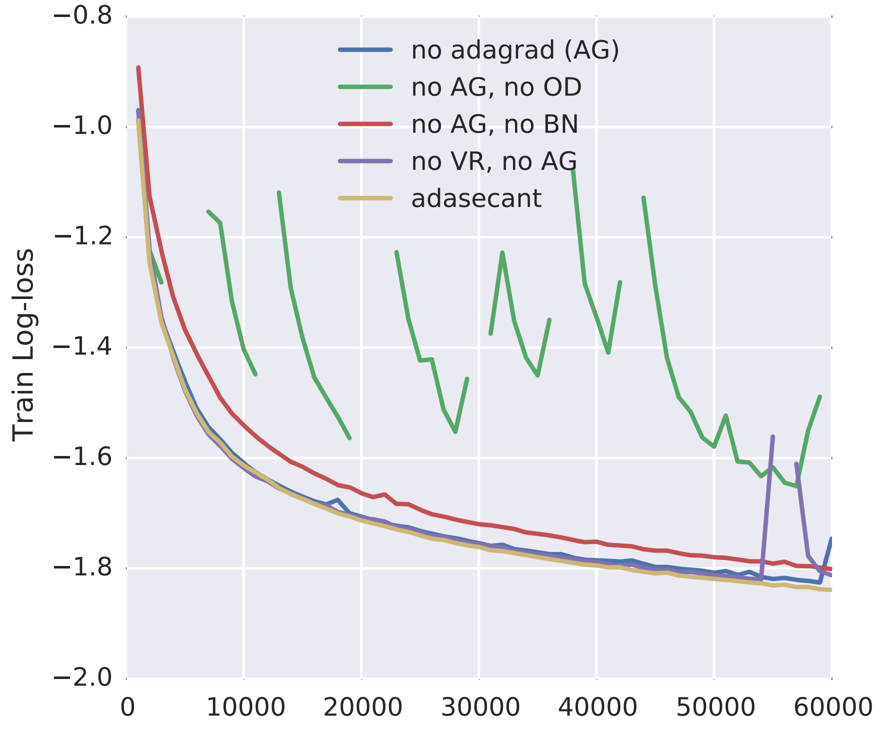


Figure 2.6 – No variance reduction comparison.



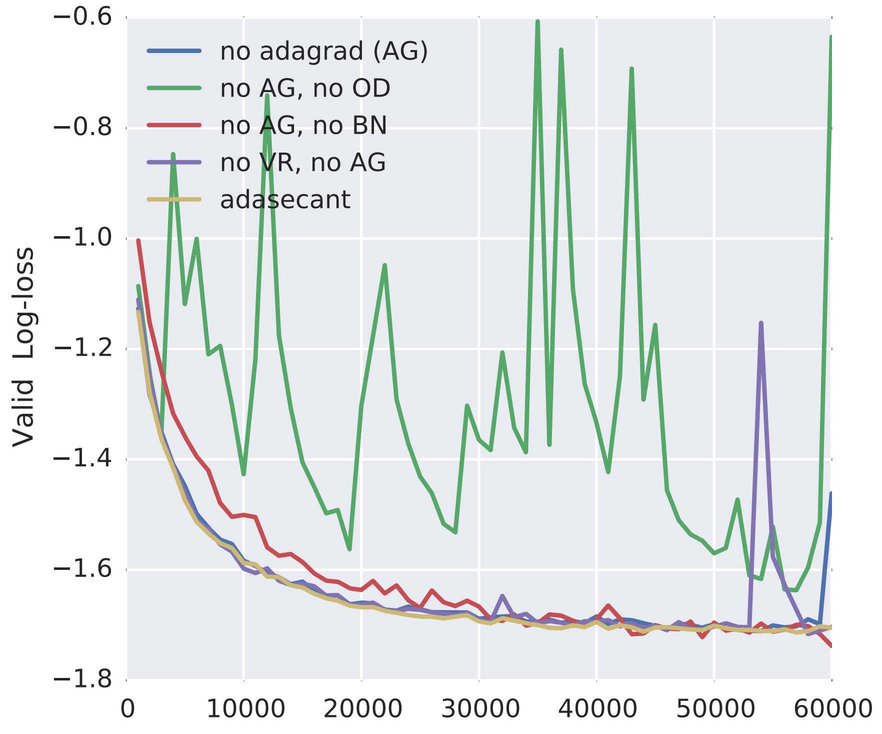
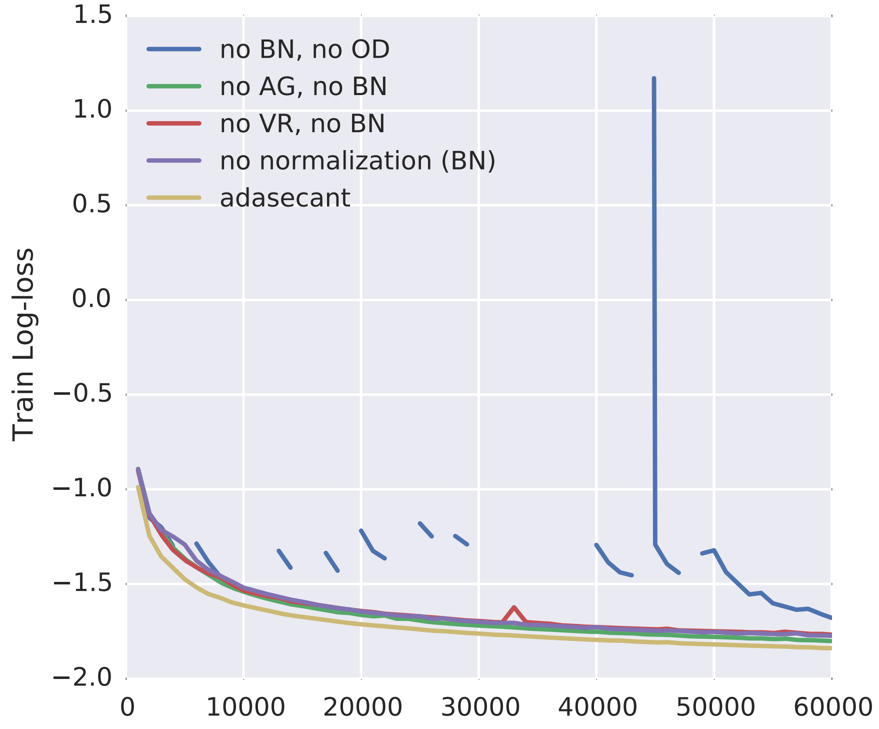


Figure 2.7 – No Adagrad comparison.



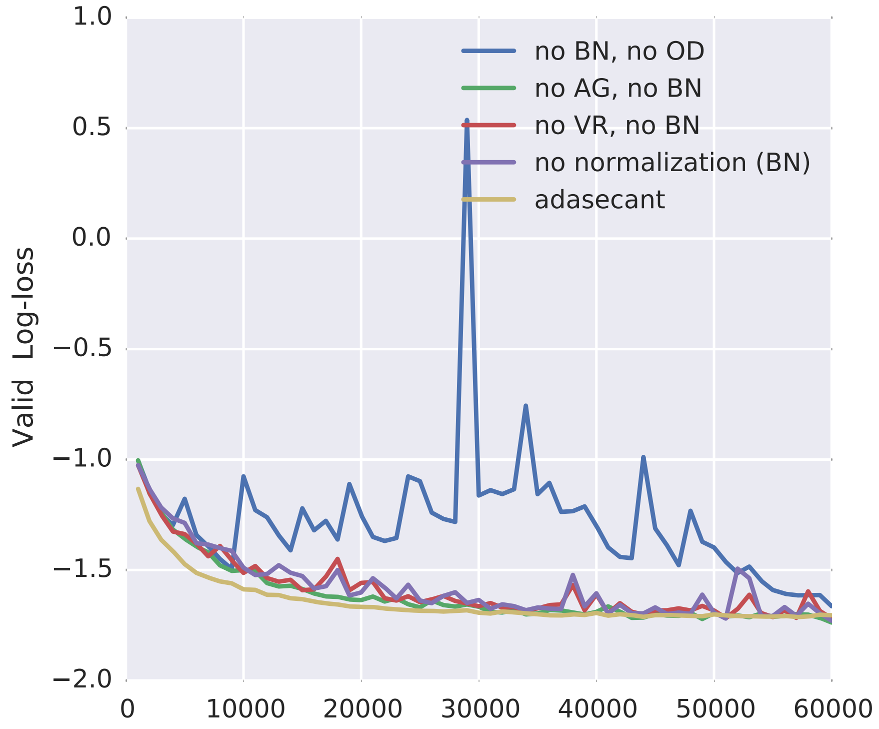
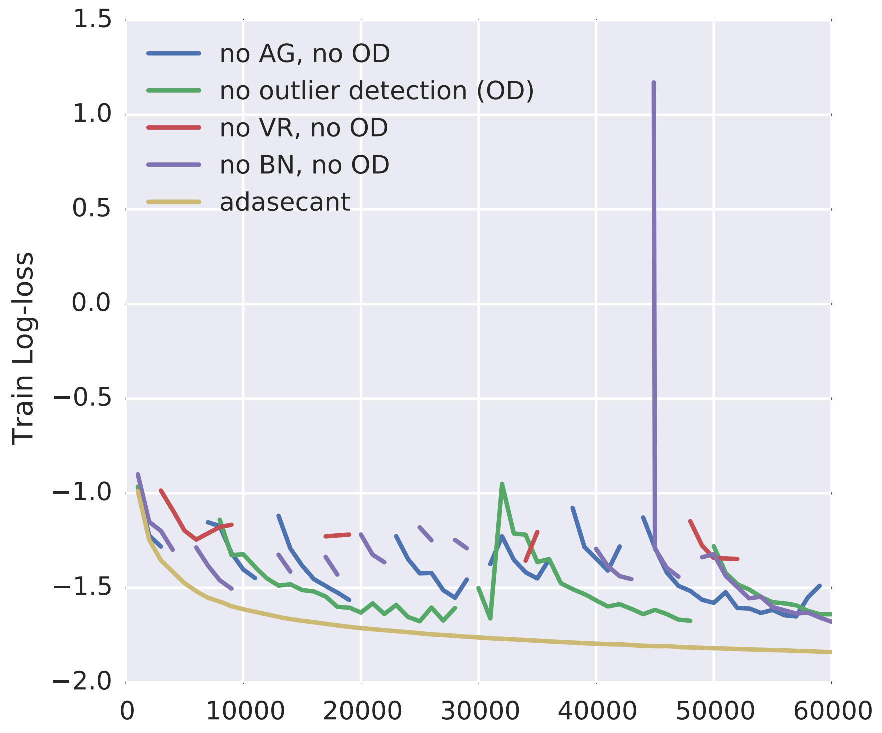


Figure 2.8 – No block normalization comparison.



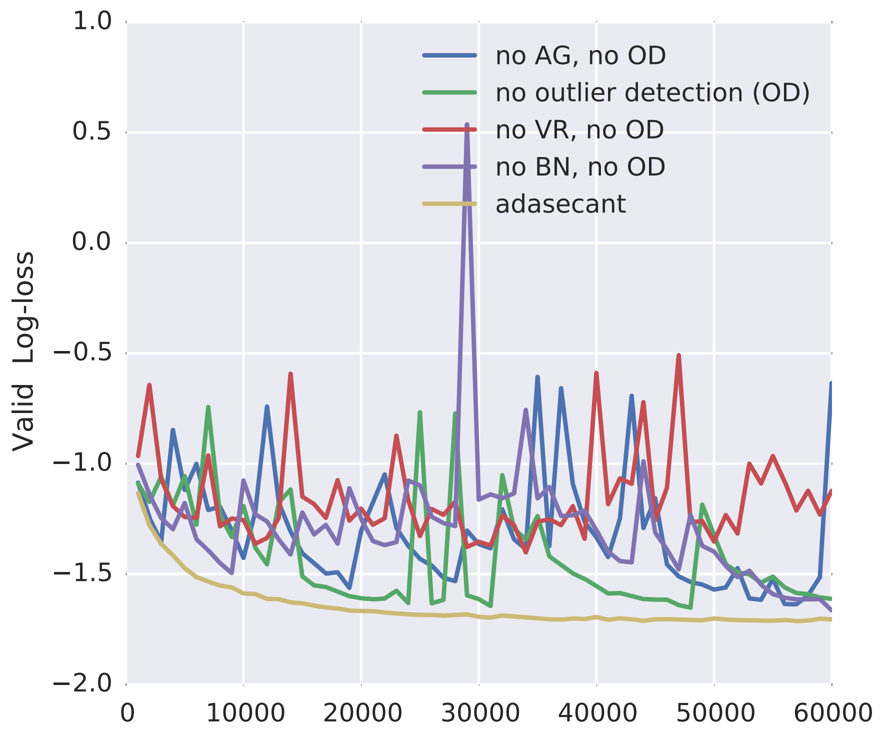


Figure 2.9 – No outlier detection comparison.

3 SampleRNN

SampleRNN: An Unconditional End-to-End Neural Audio Generation Model. Soroush Mehri, Kundan Kumar, Ishaan Gulrajani, Rithesh Kumar, Shubham Jain, Jose Sotelo, Aaron Courville, Yoshua Bengio.

Personal Contribution. My main contribution to the project was helping with the analysis of the results. Also, I wrote the code for a conditional version of the SampleRNN model but unfortunately because of the deadline this work was not included in the paper. The model was initially by Ishaan. After, in the Speech Synthesis group, Soroush, Aaron, Yoshua and I discussed this model extensively. Soroush and Kundan rewrote the code and performed and extensive hyper-parameter search. I ran a few experiments reproducing Ishaan's first results. Later, Soroush wrote most of the paper. I helped analyzing the results and producing some of the figures in the paper as well as with minor editing of the other sections. This chapter was accepted in the ICLR, 2017.

Affiliations

Soroush Mehri, MILA, Département d'Informatique et de Recherche Opérationnelle, Université de Montréal

Kundan Kumar, Computer Science and Engineering, IIT Kanpur

Ishaan Gulrajani, MILA, Département d'Informatique et de Recherche Opérationnelle, Université de Montréal

Rithesh Kumar, Department of Computer Science and Engineering, Sri Sivasubramaniya Nadar College Of Engineering

Shubham Jain, Department of Computer Science and Engineering, IIT Kanpur

Jose Sotelo, MILA, Département d'Informatique et de Recherche Opérationnelle, Université de Montréal

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

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: NSERC, Calcul Québec, Compute Canada, the Canada Research Chairs and CIFAR. This work was a collaboration with Ubisoft.

3.1 Abstract

In this paper we propose a novel model for unconditional audio generation based on generating one audio sample at a time. We show that our model, which profits from combining memory-less modules, namely autoregressive multilayer perceptrons, and stateful recurrent neural networks in a hierarchical structure is able to capture underlying sources of variations in the temporal sequences over very long time spans, on three datasets of different nature. Human evaluation on the generated samples indicate that our model is preferred over competing models. We also show how each component of the model contributes to the exhibited performance.

3.2 Introduction

Audio generation is a challenging task at the core of many problems of interest, such as text-to-speech synthesis, music synthesis and voice conversion. The particular difficulty of audio generation is that there is often a very large discrepancy between the dimensionality of the the raw audio signal and that of the effective semantic-level signal. Consider the task of speech synthesis, where we are typically interested in generating utterances corresponding to full sentences. Even at a relatively low sample rate of 16kHz, on average we will have 6,000 samples per word generated. [1]

Traditionally, the high-dimensionality of raw audio signal is dealt with by first compressing it into spectral or hand-engineered features and defining the generative model over these features. However, when the generated signal is eventually decompressed into audio waveforms, the sample quality is often degraded and requires extensive domain-expert corrective measures. This results in complicated

1. Statistics based on the average speaking rate of a set of TED talk speakers http://sixminutes.dlugan.com/speaking-rate/

signal processing pipelines that are to adapt to new tasks or domains. Here we propose a step in the direction of replacing these handcrafted systems.

In this work, we investigate the use of recurrent neural networks (RNNs) to model the dependencies in audio data. However, a known problem of these models is that they do not scale well at such a high temporal resolution as is found when generating acoustic signals one sample at a time, e.g., 16000 times per second. This is one of the reasons that **?** profits from other neural modules such as one presented by **?** to show extremely good performance.

In this paper, an end-to-end unconditional audio synthesis model for raw waveforms is presented while keeping all the computations tractable.[2] Since our model has different modules operating at different clock-rates (which is in contrast to WaveNet), we have the flexibility in allocating the amount of computational resources in modeling different levels of abstraction. In particular, we can potentially allocate very limited resource to the module responsible for sample-level alignments operating at the clock-rate equivalent to sample-rate of the audio, while allocating more resources in modeling dependencies which vary very slowly in audio, for example, the identity of phonemes being spoken. This advantage makes our model arbitrarily flexible in handling sequential dependencies at multiple levels of abstraction.

Our contribution is threefold:

1. We present a novel method that utilizes RNNs at different scales to model longer term dependencies in audio waveforms while training on short sequences which results in memory efficiency during training.
2. We extensively explore and compare variants of models achieving the above effect.
3. We study and empirically evaluate the impact of different components of our model on three audio datasets. Human evaluation also has been conducted to test these generative models.

2. Code `https://github.com/soroushmehr/sampleRNN_ICLR2017` and samples `https://soundcloud.com/samplernn/sets`

3.3 SampleRNN Model

In this paper we propose SampleRNN (shown in Fig. 3.1), a density model for audio waveforms. SampleRNN models the probability of a sequence of waveform samples $X = \{x_1, x_2, \ldots, x_T\}$ (a random variable over input data sequences) as the product of the probabilities of each sample conditioned on all previous samples:

$$p(X) = \prod_{i=0}^{T-1} p(x_{i+1}|x_1, \ldots, x_i) \tag{3.1}$$

RNNs are commonly used to model sequential data which can be formulated as:

$$h_t = \mathcal{H}(h_{t-1}, x_{i=t}) \tag{3.2}$$

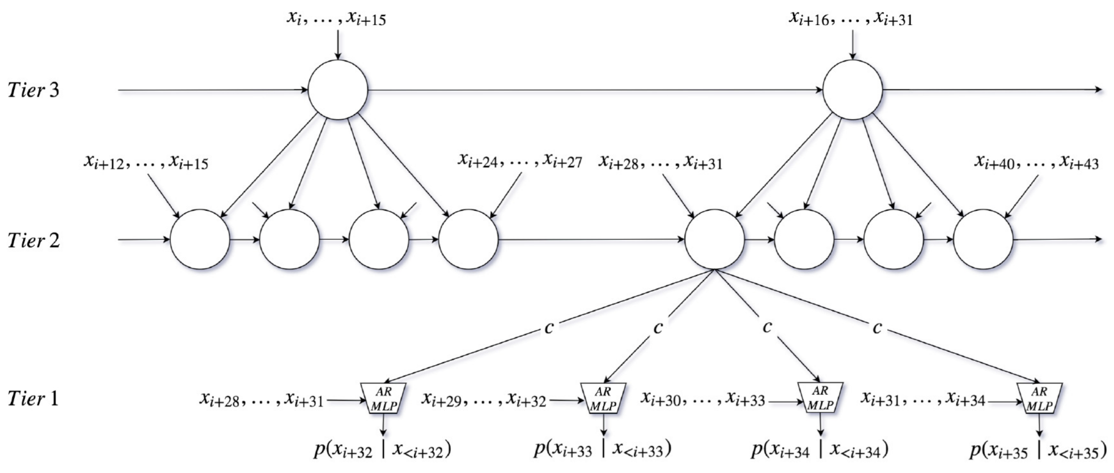
$$p(x_{i+1}|x_1, \ldots, x_i) = Softmax(MLP(h_t)) \tag{3.3}$$

with $\mathcal{H}$ being one of the known memory cells, Gated Recurrent Units (GRUs) (?), Long Short Term Memory Units (LSTMs) (?), or their deep variations (Section 3.4). However, raw audio signals are challenging to model because they contain structure at very different scales: correlations exist between neighboring samples as well as between ones thousands of samples apart.

SampleRNN helps to address this challenge by using a hierarchy of modules, each operating at a different temporal resolution. The lowest module processes individual samples, and each higher module operates on an increasingly longer timescale and a lower temporal resolution. Each module conditions the module below it, with the lowest module outputting sample-level predictions. The entire hierarchy is trained jointly end-to-end by backpropagation.

3.3.1 Frame-level Modules

Rather than operating on individual samples, the higher-level modules in SampleRNN operate on *non-overlapping frames* of $FS^{(k)}$ ("Frame Size") samples at the k^{th} level up in the hierarchy at a time (frames denoted by $f^{(k)}$). Each frame-level module is a deep RNN which summarizes the history of its inputs into a conditioning vector for the next module downward.



$x_i, \dots, x_{i+15}$
$x_{i+16}, \dots, x_{i+31}$
Tier 3
Tier 2
Tier 1
$x_{i+12}, \dots, x_{i+15}$
$x_{i+24}, \dots, x_{i+27}$
$x_{i+28}, \dots, x_{i+31}$
$x_{i+40}, \dots, x_{i+43}$
c
AR MLP
$x_{i+28}, \dots, x_{i+31}$
$x_{i+29}, \dots, x_{i+32}$
$x_{i+30}, \dots, x_{i+33}$
$x_{i+31}, \dots, x_{i+34}$
$p(x_{i+32} \mid x_{<i+32})$
$p(x_{i+33} \mid x_{<i+33})$
$p(x_{i+34} \mid x_{<i+34})$
$p(x_{i+35} \mid x_{<i+35})$

Figure 3.1 – Snapshot of the unrolled model at timestep i with $K = 3$ tiers. As a simplification only one RNN and up-sampling ratio $r = 4$ is used for all tiers.

The variable number of frames we condition upon up to timestep $t - 1$ is expressed by a fixed length hidden state or memory $h_t^{(k)}$ where t is related to clock rate at that tier. The RNN makes a memory update at timestep t as a function of the previous memory $h_{t-1}^{(k)}$ and an input $inp_t^{(k)}$. This input for top tier $k = K$ is simply the input frame. For intermediate tiers $(1 < k < K)$ this input is a linear combination of conditioning vector from higher tier and current input frame. See Eqs. 3.4–3.5.

Because different modules operate at different temporal resolutions, we need to upsample each vector c at the output of a module into a series of $r^{(k)}$ vectors (where $r^{(k)}$ is the ratio between the temporal resolutions of the modules) before feeding it into the input of the next module downward (Eq. 3.6). We do this with a set of $r^{(k)}$ separate linear projections.

Here we are formalizing the frame-level module in tier k. Note that following equations are exclusive to tier k and timestep t for that specific tier. To increase the readability, unless necessary superscript (k) is not shown for t, $inp^{(k)}$, $W_x^{(k)}$,

$h^{(k)}$, $\mathcal{H}^{(k)}$, $W_j^{(k)}$, and $r^{(k)}$.

$$inp_t = \begin{cases} W_x f_t^{(k)} + c_t^{(k+1)}; & 1 < k < K \\ f_t^{(k=K)}; & k = K \end{cases} \tag{3.4}$$

$$h_t = \mathcal{H}(h_{t-1}, inp_t) \tag{3.5}$$

$$c_{(t-1)*r+j}^{(k)} = W_j h_t; \qquad 1 \leq j \leq r \tag{3.6}$$

Our approach of upsampling with $r^{(k)}$ linear projections is exactly equivalent to upsampling by adding zeros and then applying a linear convolution. This is sometimes called “perforated” upsampling in the context of convolutional neural networks (CNNs). It was first demonstrated to work well in **?** and is a fairly common upsampling technique.

3.3.2 Sample-level Module

The lowest module (tier $k = 1$; Eqs. 3.7–3.9) in the SampleRNN hierarchy outputs a distribution over a sample x_{i+1}, conditioned on the $FS^{(1)}$ *preceding samples* as well as a vector $c_i^{(k=2)}$ from the next higher module which encodes information about the sequence prior to that frame. As $FS^{(1)}$ is usually a small value and correlations in nearby samples are easy to model by a simple memoryless module, we implement it with a multilayer perceptron (MLP) rather than RNN which slightly speeds up the training. Assuming e_i represents x_i after passing through embedding layer (section 3.3.2), conditional distribution in Eq. 3.1 can be achieved by following and for further clarity two consecutive sample-level frames are shown. In addition, W_x in Eq. 3.8 is simply used to linearly combine a frame and conditioning vector from above.

$$f_{i-1}^{(1)} = flatten([e_{i-FS^{(1)}}, \ldots, e_{i-1}]) \tag{3.7}$$

$$f_i^{(1)} = flatten([e_{i-FS^{(1)}+1}, \ldots, e_i])$$

$$inp_i^{(1)} = W_x^{(1)} f_i^{(1)} + c_i^{(2)} \tag{3.8}$$

$$p(x_{i+1}|x_1, \ldots, x_i) = Softmax(MLP(inp_i^{(1)})) \tag{3.9}$$

We use a Softmax because we found that better results were obtained by discretizing the audio signals (also see **?**) and outputting a Multinoulli distribution rather than using a Gaussian or Gaussian mixture to represent the conditional density of the original real-valued signal. When processing an audio sequence, the MLP is convolved over the sequence, processing each window of $FS^{(1)}$ samples and predicting the next sample. At generation time, the MLP is run repeatedly to generate one sample at a time. Table 3.1 shows a considerable gap between the baseline model RNN and this model, suggesting that the proposed hierarchically structured architecture of SampleRNN makes a big difference.

Output Quantization

The sample-level module models its output as a q-way discrete distribution over possible quantized values of x_i (that is, the output layer of the MLP is a q-way Softmax).

To demonstrate the importance of a discrete output distribution, we apply the same architecture on real-valued data by replacing the q-way Softmax with a Gaussian Mixture Models (GMM) output distribution. Table 3.2 shows that our model outperforms an RNN baseline even when both models use real-valued outputs. However, samples from the real-valued model are almost indistinguishable from random noise.

In this work we use linear quantization with $q = 256$, corresponding to a per-sample bit depth of 8. Unintuitively, we realized that even linearly decreasing the bit depth (resolution of each audio sample) from 16 to 8 can ease the optimization procedure while generated samples still have reasonable quality and are artifact-free.

In addition, early on we noticed that the model can achieve better performance and generation quality when we *embed the quantized input values* before passing them through the sample-level MLP (see Table 3.4). The embedding steps maps each of the q discrete values to a real-valued vector embedding. However, real-valued raw samples are still used as input to the higher modules.

Conditionally Independent Sample Outputs

To demonstrate the importance of a sample-level autoregressive module, we try replacing it with "Multi-Softmax" (see Table 3.4), where the prediction of each sample x_i depends only on the conditioning vector c from Eq. 3.9. In this configuration, the model outputs an entire *frame* of $FS^{(1)}$ samples at a time, modeling all samples in a frame as conditionally independent of each other. We find that this Multi-Softmax model (which lacks a sample-level autoregressive module) scores significantly worse in terms of log-likelihood and fails to generate convincing samples. This suggests that modeling the joint distribution of the acoustic samples inside each frame is very important in order to obtain good acoustic generation. We found this to be true even when the frame size is reduced, with best results always with a frame size of 1, i.e., generating only one acoustic sample at a time.

3.3.3 Truncated BPTT

Training recurrent neural networks on long sequences can be very computationally expensive. **?** avoid this problem by using a stack of dilated convolutions instead of any recurrent connections. However, when they can be trained efficiently, recurrent networks have been shown to be very powerful and expressive sequence models. We enable efficient training of our recurrent model using *truncated backpropagation through time*, splitting each sequence into short subsequences and propagating gradients only to the beginning of each subsequence. We experiment with different subsequence lengths and demonstrate that we are able to train our networks, which model very long-term dependencies, despite backpropagating through relatively short subsequences.

Table 3.3 shows that by increasing the subsequence length, performance substantially increases alongside with train-time memory usage and convergence time. Yet it is noteworthy that our best models have been trained on subsequences of length 512, which corresponds to 32 milliseconds, a small fraction of the length of a single a phoneme of human speech while generated samples exhibit longer word-like structures.

Despite the aforementioned fact, this generative model can mimic the existing long-term structure of the data which results in more natural and coherent samples that is preferred by human listeners. (More on this in Sections 3.4.2–3.4.3.) This is due to the fast updates from TBPTT and specialized frame-level modules (Section 3.3.1) with top tiers designed to model a lower resolution of signal while leaving the process of filling the details to lower tiers.

3.4 Experiments and Results

In this section we are introducing three datasets which have been chosen to evaluate the proposed architecture for modeling raw acoustic sequences. The description of each dataset and their preprocessing is as follows:

Blizzard which is a dataset presented by **?** for speech synthesis task, contains 315 hours of a single female voice actor in English; however, for our experiments we are using only 20.5 hours. The training/validation/test split is 86%-7%-7%.

Onomatopoeia[3], a relatively small dataset with 6,738 sequences adding up to 3.5 hours, is human vocal sounds like grunting, screaming, panting, heavy breathing, and coughing. Diversity of sound type and the fact that these sounds were recorded from 51 actors and many categories makes it a challenging task. To add to that, this data is extremely unbalanced. The training/validation/test split is 92%-4%-4%.

Music dataset is the collection of all 32 Beethoven's piano sonatas publicly available on `https://archive.org/` amounting to 10 hours of non-vocal audio. The training/validation/test split is 88%-6%-6%.

See Fig. 3.2 for a visual demonstration of examples from datasets and generated samples. For all the datasets we are using a 16 kHz sample rate and 16 bit depth. For the Blizzard and Music datasets, preprocessing simply amounts to chunking the long audio files into 8 seconds long sequences on which we will perform truncated backpropagation through time. Each sequence in the Onomatopoeia dataset is few seconds long, ranging from 1 to 11 seconds. To train the models on this dataset, zero-padding has been applied to make all the sequences in a mini-batch have the same length and corresponding cost values (for the predictions over the added 0s) would be ignored when computing the gradients.

We particularly explored two gated variants of RNNs—GRUs and LSTMs. For the case of LSTMs, the forget gate bias is initialized with a large positive value of 3, as recommended by **?** and **?**, which has been shown to be beneficial for learning long-term dependencies.

As for models that take real-valued input, e.g. the RNN-GMM and SampleRNN-GMM (with 4 components), normalization is applied per audio sample with the

3. Courtesy of Ubisoft

Table 3.1 – Test NLL in bits for three presented datasets.

Model	Blizzard	Onomatopoeia	Music
RNN (Eq. 3.2)	1.434	2.034	1.410
WaveNet (re-impl.)	1.480	2.285	1.464
SampleRNN (2-tier)	1.392	2.026	**1.076**
SampleRNN (3-tier)	**1.387**	**1.990**	1.159

Table 3.2 – Average NLL on Blizzard test set for real-valued models.

Model	Average Test NLL
RNN-GMM	-2.415
SampleRNN-GMM (2-tier)	**-2.782**

global mean and standard deviation obtained from the train split. For most of our experiments where the model demands discrete input, binning was applied per audio sample.

All the models have been trained with teacher forcing and stochastic gradient decent (mini-batch size 128) to minimize the Negative Log-Likelihood (NLL) in bits per dimension (per audio sample). Gradients were hard-clipped to remain in [-1, 1] range. Update rules from the Adam optimizer (**?**) ($\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 1e{-}8$) with an initial learning rate of 0.001 was used to adjust the parameters. For training each model, random search over hyper-parameter values (**?**) was conducted. The initial RNN state of all the RNN-based models was always learnable. Weight Normalization (**?**) has been used for all the linear layers in the model (except for the embedding layer) to accelerate the training procedure. Size of the embedding layer was 256 and initialized by standard normal distribution. Orthogonal weight matrices used for hidden-to-hidden connections and other weight matrices initialized similar to **?**. In final model, we found GRU to work best (slightly better than LSTM). 1024 was the the number of hidden units for all GRUs (1 layer per tier for 3-tier and 3 layer for 2-tier model) and MLPs (3 fully connected layers with ReLU activation with output dimension being 1024 for first two layers and 256 for the final layer before softmax). Also $FS^{(1)} = FS^{(2)} = 2$ and $FS^{(3)} = 8$ were found to result in lowest NLL.

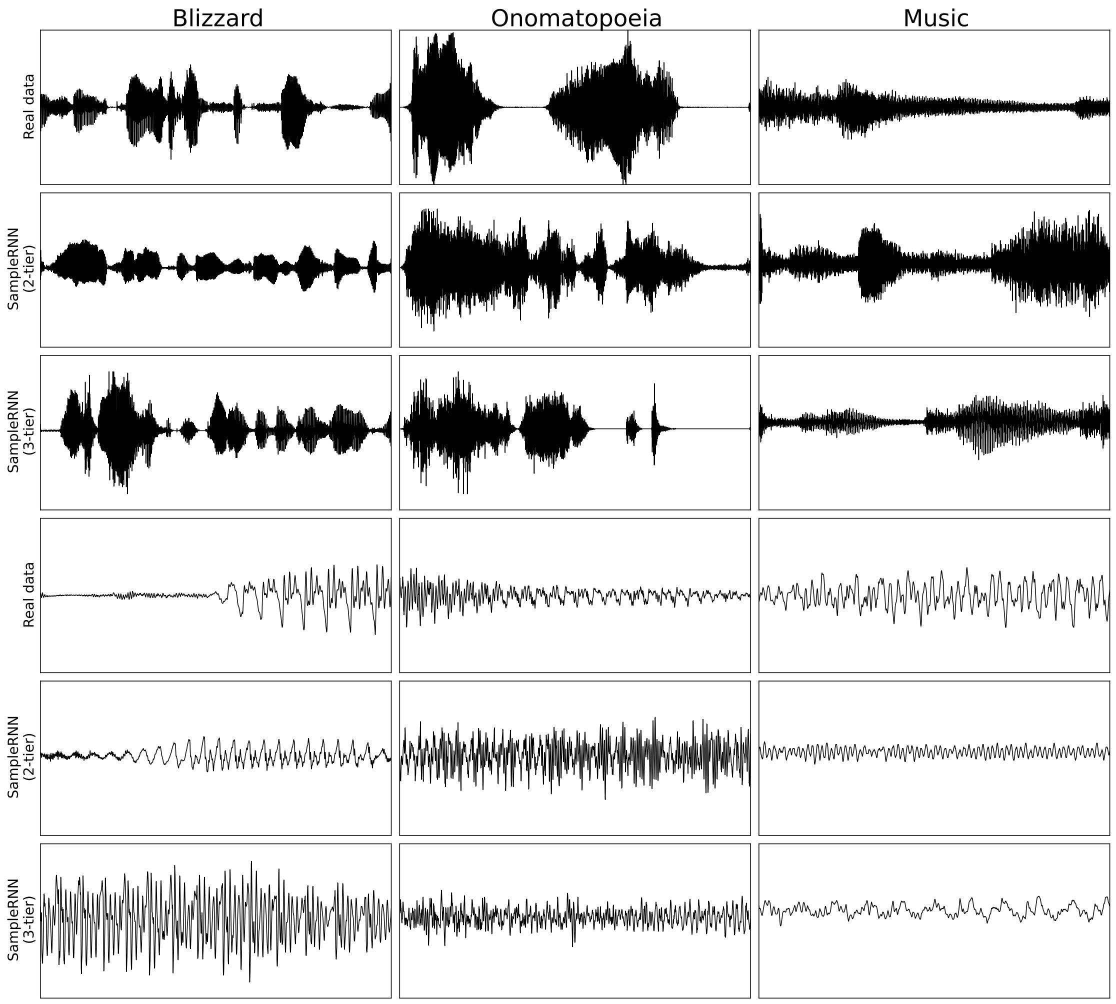


Figure 3.2 – Examples from the datasets compared to samples from our models. In the first 3 rows, 2 seconds of audio are shown. In the bottom 3 rows, 100 milliseconds of audio are shown. Rows 1 and 4 are ground truth from which one can see how the datasets look different and have complex structure in low resolution which the frame-level component of the SampleRNN is designed to capture. Samples also to some extent mimic the same global structure. At the same time, zoomed-in samples of our model shows that it can perfectly resemble the high resolution structure present in the data as well.

Table 3.3 – Effect of subsequence length on NLL (bits per audio sample) computed on the Blizzard validation set.

Subsequence Length	32	64	128	256	512
NLL Validation	1.575	1.468	1.412	1.391	1.364

Table 3.4 – Test (validation) set NLL (bits per audio sample) for Blizzard. Variants of SampleRNN are provided to compare the contribution of each component in performance.

Model	**NLL Test (Validation)**
SampleRNN (2-tier)	1.392 (1.369)
Without Embedding	1.566 (1.539)
Multi-Softmax	1.685 (1.656)

3.4.1 WaveNet Re-implementation

We implemented the WaveNet architecture as described in **?**. Ideally, we would have liked to replicate their model exactly but owing to missing details of architecture and hyperparameters, as well as limited compute power at our disposal, we made our own design choices so that the model would fit on a single GPU while having a receptive field of around 250 milliseconds and a reasonable number of updates per unit time. Although our model is very similar to WaveNet, the design choices, e.g. number of convolution filters in each dilated convolution layer, length of target sequence to train on simultaneously (one can train with a single target with all samples in the receptive field as input or with target sequence length of size T with input of size receptive field + T - 1), batch-size, etc. might make our implementation different from what the authors have done in the original WaveNet model. Hence, we note here that although we did our best at exactly reproducing their results, there would very likely be different choice of hyper-parameters between our implementation and the one of the authors.

For our WaveNet implementation, we have used 4 dilated convolution blocks each having 10 dilated convolution layers with dilation 1, 2, 4, 8 up to 512. Hence, our network has a receptive field of 4092 acoustic samples i.e. the parameters of multinomial distribution of sample at time step t, $p(x_i) = f_\theta(x_{i-1}, x_{i-2}, \dots x_{i-4092})$ where θ is model parameters. We train on target sequence length of 1600 and use batch size of 8. Each dilated convolution filter has size 2 and the number of output channels is 64 for each dilated convolutional layer (128 filters in total due to gated non-linearity). We trained this model using Adam optimizer with a fixed global learning rate of 0.001 for Blizzard dataset and 0.0001 for Onomatopoeia and Music datasets. We trained these models for about one week on a GeForce GTX TITAN X. We dropped the learning rate in the Blizzard experiment to 0.0001 after around

3 days of training.

3.4.2 Human Evaluation

Apart from reporting NLL, we conducted AB preference tests for random samples from four models trained on the Blizzard dataset. For unconditional generation of speech which at best sounds like mumbling, this type of test is the one which is more suited. Competing models were the RNN, SampleRNN (2-tier), SampleRNN (3-tier), and our implementation of WaveNet. The rest of the models were excluded as the quality of samples were definitely lower and also to keep the number of pair comparison tests manageable. We will release the samples that have been used in this test too.

All the samples were set to have the same volume. Every user is then shown a set of twenty pairs of samples with one random pair at a time. Each pair had samples from two different models. The human evaluator is asked to listen to the samples and had the option of choosing between the two model or choosing not to prefer any of them. Hence, we have a quantification of preference between every pair of models. We used the online tool made publicly available by **?**.

Results in Fig. 3.3 clearly points out that SampleRNN (3-tier) is a winner by a huge margin in terms of preference by human raters, then SampleRNN (2-tier) and afterward two other models, which matches with the performance comparison in Table 3.1.

The same evaluation was conducted for Music dataset except for an additional filtering process of samples. Specific to only this dataset, we observed that a batch of generated samples from competing models (this time restricted to RNN, SampleRNN (2-tier), and SampleRNN (3-tier)) were either music-like or random noise. For all these models we only considered random samples that were not random noise. Fig. 3.4 is dedicated to result of human evaluation on Music dataset.

3.4.3 Quantifying Information Retention

For the last experiment we are interested in measuring the memory span of the model. We trained our model, SampleRNN (3-tier), with best hyper-parameters on a dataset of 2 speakers reading audio books, one male and one female, respec-

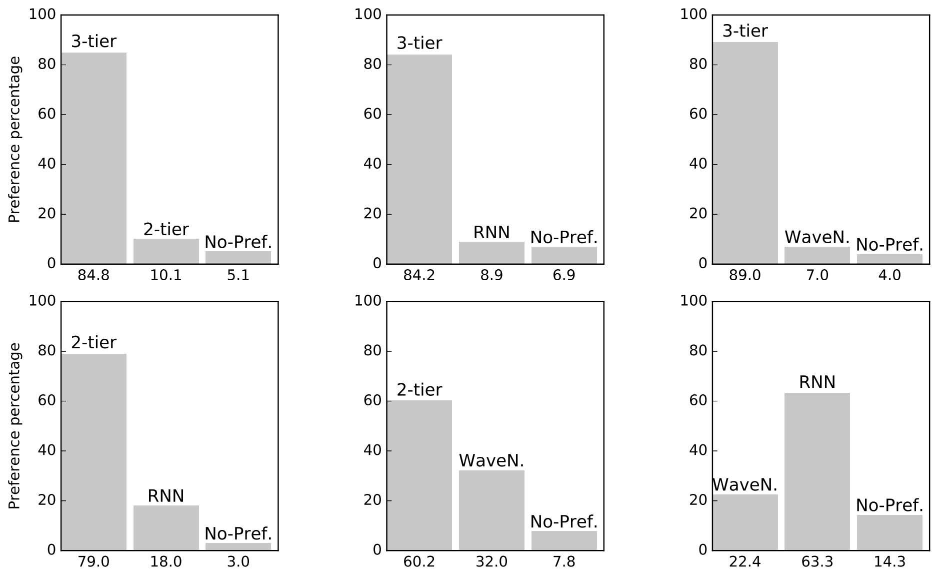


Figure 3.3 – Pairwise comparison of 4 best models based on the votes from listeners conducted on samples generated from models trained on Blizzard dataset.

tively, with mean fundamental frequency of 125.3 and 201.8Hz. Each speaker has roughly 10 hours of audio in the dataset that has been preprocessed similar to Blizzard. We observed that it learned to stay consistent generating samples from the same speaker without having any knowledge about the speaker ID or any other conditioning information. This effect is more apparent here in comparison to the unbalanced Onomatopoeia that sometimes mixes two different categories of sounds.

Another experiment was conducted to test the effect of memory and study the effective memory horizon. We inject 1 second of silence in the middle of sampling procedure in order to see if it will remember to generate from the same speaker or not. Initially when sampling we let the model generate 2 seconds of audio as it normally do. From 2 to 3 seconds instead of feeding back the generated sample at that timestep a silent token (zero amplitude) would be fed. From 3 to 5 seconds again we sample normally; feeding back the generated token.

We did classification based on mean fundamental frequency of speakers for the first and last 2 seconds. In 83% of samples SampleRNN generated from the same person in two separate segments. This is in contrast to a model with fixed past window like WaveNet where injecting 16000 silent tokens (3.3 times the receptive

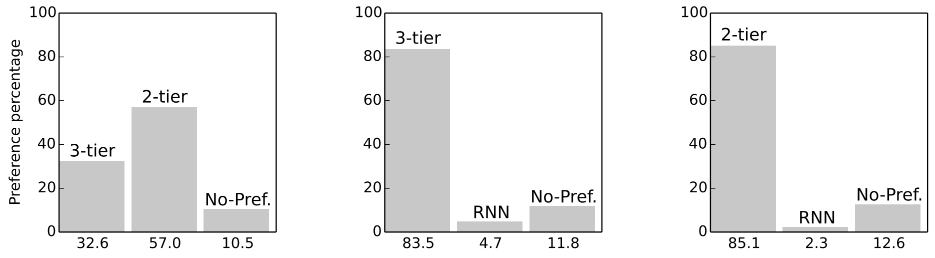


Figure 3.4 – Pairwise comparison of 3 best models based on the votes from listeners conducted on samples generated from models trained on Music dataset.

field size) is equivalent to generating from scratch which has 50% chance (assuming each 2-second segment is coherent and not a mixed sound of two speakers).

3.5 Related Work

Our work is related to earlier work on auto-regressive multi-layer neural networks, starting with **?**, then NADE (**?**) and more recently PixelRNN (**?**). Similar to how they tractably model joint distribution over units of the data (e.g. words in sentences, pixels in images, etc.) through an auto-regressive decomposition, we transform the joint distribution of acoustic samples using Eq. 3.1.

The idea of having part of the model running at different clock rates is related to multi-scale RNNs (**?????**).

? also attempt to model raw audio waveforms which is in contrast to traditional approaches which use spectral features as in **?**, **?**, and **?**.

Our work is closely related to WaveNet (**?**), which is why we have made the above comparisons, and makes it interesting to compare the effect of adding higher-level RNN stages working at a low resolution. Similar to this work, our models generate one acoustic sample at a time conditioned on all previously generated samples. We also share the preprocessing step of quantizing the acoustics into bins. Unlike this model, we have different modules in our models running at different clock-rates. In contrast to WaveNets, we mitigate the problem of long-term dependency with hierarchical structure and using stateful RNNs, i.e. we will always propagate hidden states to the next training sequence although the gradient of the loss will not take into account the samples in previous training sequence.

3.6 Discussion and Conclusion

We propose a novel model that can address unconditional audio generation in the raw acoustic domain, which typically has been done until recently with hand-crafted features. We are able to show that a hierarchy of time scales and frequent updates will help to overcome the problem of modeling extremely high-resolution temporal data. That allows us, for this particular application, to learn the data manifold directly from audio samples. We show that this model can generalize well and generate samples on three datasets that are different in nature. We also show that the samples generated by this model are preferred by human raters.

Success in this application, with a general-purpose solution as proposed here, opens up room for more improvement when specific domain knowledge is applied. This method, however, proposed with audio generation application in mind, can easily be adapted to other tasks that require learning the representation of sequential data with high temporal resolution and long-range complex structure.

Acknowledgments

The authors would like to thank João Felipe Santos and Kyle Kastner for insightful comments and discussion. We would like to thank the **?**[4] and MILA staff. We acknowledge the support of the following agencies for research funding and computing support: NSERC, Calcul Québec, Compute Canada, the Canada Research Chairs and CIFAR. Jose Sotelo also thanks the Consejo Nacional de Ciencia y Tecnología (CONACyT) as well as the Secretaría de Educación Pública (SEP) for their support. This work was a collaboration with Ubisoft.

3.7 Appendix A

3.7.1 A model variant: SampleRNN-WaveNet Hybrid

SampleRNN-WaveNet model has two modules operating at two different clock-rate. The slower clock-rate module (frame-level module) sees one frame (each of which has size *FS*) at a time while the faster clock-rate component(sample-level

4. http://deeplearning.net/software/theano/

component) sees one acoustic sample at a time i.e. the ratio of clock-rates for these two modules would be the size of a single frame. Number of sequential steps for frame-level component would be *FS* times lower. We repeat the output of each step of frame-level component *FS* times so that number of time-steps for output of both the components match. The output of both these modules are concatenated for every time-step which is further operated by non-linearities for every time-step independently before generating the final output.

In our experiments, we kept size of a single frame (*FS*) to be 128. We tried two variants of this model: 1. fully convolutional WaveNet and 2. RNN-WaveNet. In fully convolutional WaveNet, both modules described above are implemented using dilated convolutions as described in original WaveNet model. In RNN-WaveNet, we use high capacity RNN in the frame-level module to model the dependency between frames. The sample-level WaveNet in RNN-WaveNet has receptive field of size 509 samples from the past.

Although these models are designed with the intention of combining the two models to harness their best features, preliminary experiments show that this variant is not meeting our expectations at the moment which directs us to a possible future work.

4 Speech synthesis

Char2Wav: End-to-End Speech Synthesis. Jose Sotelo, Soroush Mehri, Kundan Kumar, João Felipe Santos, Kyle Kastner, Aaron Courville, Yoshua Bengio.

Personal Contribution. I was the main contributor to this project. The idea of using a recurrent neural network with attention for doing speech synthesis was probably first developed by Alex Graves. Implementing this idea was one of the main goals of the speech synthesis group and was discussed many times by its members. I was able to implement this idea supervised by Aaron and Yoshua. Furthermore, the reader was integrated with the neural vocoder.

A shorter version of this chapter was accepted in the ICLR Workshop, 2017.

Affiliations

Jose Sotelo, MILA, Département d'Informatique et de Recherche Opérationnelle, Université de Montréal

Soroush Mehri, MILA, Département d'Informatique et de Recherche Opérationnelle, Université de Montréal

Kundan Kumar, Computer Science and Engineering, IIT Kanpur

João Felipe Santos, INRS-EMT

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

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

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: NSERC, Calcul Québec, Compute Canada, the Canada Research Chairs and CIFAR. João Felipe Santos and Jose Sotelo thank the Fonds de Recherche du Québec - Nature et Technologies (FQRNT) for their support. Jose Sotelo also thanks the Consejo Nacional de Ciencia y Tecnología

(CONACyT) as well as the Secretaría de Educación Pública (SEP) for their support.

4.1 Abstract

We present Char2Wav, an end-to-end model for speech synthesis. Char2Wav has two components: a **reader** and a **neural vocoder**. The *reader* is an encoder-decoder model with attention. The encoder is a bidirectional recurrent neural network (RNN) that accepts text or phonemes as inputs, while the decoder is a recurrent neural network with attention that produces vocoder acoustic features. *Neural vocoder* refers to a conditional extension of SampleRNN which generates raw waveform samples from intermediate representations. We show results in English and Spanish. Unlike traditional models for speech synthesis, Char2Wav learns to produce audio directly from text. This work is a proof of concept showing that speech synthesis is made possible directly from characters.

4.2 Introduction

In speech synthesis the main task consists in mapping text to an audio signal. There are two primary goals in speech synthesis: **intelligibility** and **naturalness**. *Intelligibility* refers to the clarity of the synthesized audio, or in other words, how well can the original message be extracted by a listener. *Naturalness* refers to the information that is not directly captured by intelligibility, such as overall ease of listening, global stylistic consistency, regional or language level nuances among others.

With traditional speech synthesis approaches, this task has been accomplished by dividing the problem into two stages. The first stage, known as the **frontend**, transforms the text into linguistic features. These linguistic features usually include phoneme, syllable, word, phrase and utterance-level features (**???**). The second stage, known as the **backend**, takes the linguistic features generated by the frontend as input and produces the corresponding sound. WaveNet (**?**) is a high quality approach to a "neural backend". For a more detailed review of traditional models for speech synthesis, we recommend consulting **?**.

Defining good linguistic features is often time-consuming and language specific. Our procedure eliminates the need for these linguistic features, hence taking away a major bottleneck in creating synthesizers for new languages. We use a powerful model to learn this information directly from the data.

The main contribution of this work is that we show that raw speech synthesis directly from the sequence of characters is possible. We take advantage of a clever audio representation used in traditional parametric speech synthesis to initialize our model. Our approach requires no linguistic features.

4.3 Char2Wav

In traditional parametric speech synthesis, one starts with a text, originally encoded as a sequence of characters, which is then converted to a sequence of phonemes (and possibly other prosodic information to encode emphasis, stress, and intonation). This sequence of phonemes is then converted into a sequence of acoustic features for a vocoder that encodes how each short frame (normally in the range of 10 to 50 milliseconds) should sound. The vocoder synthesis process finally converts that sequence into a digital audio signal. On all those different levels, the sequences have very different timescales and lengths making direct end-to-end hard.

In Char2Wav, we leverage vocoder features as an intermediate representation and split the training procedure into two stages. First, we pretrain the *reader* to map from a sequence of characters to the vocoder representation. Then, we train a *neural vocoder* to map from the vocoder representation generated by the *reader* to audio samples while keeping the *reader* parameters fixed. Finally, we fine-tune both models end-to-end.

While the approach presented here is specific to speech signals, it can be applied to other domains with huge gaps between the timescales or sequence lengths of the inputs and outputs to the model are present. For example, in a high-resolution image synthesis model, one could imagine training a reader to go from an image caption to a representation that encodes which objects are in the scene and their layout, and the other part of the model could convert that representation to a sequence of pixels.

4.3.1 Attention-based Recurrent Sequence Generator

We adopt the notation of **?**. An attention-based recurrent sequence generator (ARSG) is a recurrent neural network that generates a sequence $Y = (y_1, \ldots, y_T)$ conditioned on an input sequence X. X is preprocessed by an encoder that outputs a sequence $h = (h_1, \ldots, h_L)$. In this work, the output Y is a sequence of acoustic features and X is the text, formatted as a character or phoneme sequence. Furthermore, the encoder is a bidirectional recurrent network.

At the i-th step the ARSG focuses on h and generates y_i:

$$\alpha_i = Attend(s_{i-1}, \alpha_{i-1}, h) \tag{4.1}$$

$$g_i = \sum_{j=1}^{L} \alpha_{i,j} h_j \tag{4.2}$$

$$y_i \sim Generate(s_{i-1}, g_i) \tag{4.3}$$

$$s_i = RNN(s_{i-1}, g_i, y_i) \tag{4.4}$$

where s_{i-1} is the $(i-1)$-th state of the **generator** recurrent neural network and $\alpha_i \in \mathcal{R}^L$ are the attention weights or alignment.

4.3.2 Reader

We use the location-based attention mechanism developed by **?**. We have $\alpha_i = Attend(s_{i-1}, \alpha_{i-1})$ and given a length L conditioning sequence h, we have:

$$\phi(i,l) = \sum_{k=1}^{K} \rho_i^k \exp(-\beta_i^k (\kappa_i^k - l)^2) \tag{4.5}$$

$$\alpha_i = \sum_{l=1}^{L} \phi(i,l) \tag{4.6}$$

where κ_i, β_i, and ρ_i represent the location, width and importance of the window respectively and K corresponds to the number of Gaussians to use in the attention mechanism.

Finally, to complete the specification of the *Attend* function we have:

$$(\hat{\rho}_i, \hat{\beta}_i, \hat{\kappa}_i) = W s_{i-1} \tag{4.7}$$

$$\tilde{\rho}_i = \exp(\hat{\rho}_i) \tag{4.8}$$

$$\beta_i = \exp(\hat{\beta}_i) \tag{4.9}$$

$$\kappa_i = \kappa_{i-1} + \exp(\hat{\kappa}_i) \tag{4.10}$$

We made two changes to the attention mechanism that makes the conditioning attention training more stable. First, we normalize the window mixture (ρ_i) with:

$$\rho_i^k = \frac{\tilde{\rho}_i^k}{\sum_{k=1}^{K} \tilde{\rho}_i^k} \tag{4.11}$$

Second, we add a constant $\mathbf{v}$ to Equation 4.10:

$$\kappa_i = \kappa_{i-1} + \mathbf{v} \exp(\hat{\kappa}_i) \tag{4.12}$$

$\mathbf{v}$ helps control the speed at which the model sees the text. This is important in a context of speech synthesis since the length of the sequences are of different orders of magnitude. In our experiments, $\mathbf{v}$ ranged from 0.02 to 0.08, derived from the average length between text and vocoder features over the training set.

For all the models that we trained, we embedded the text or phoneme sequence into a vector of size 128. After that, the encoder was a 1-layer bidirectional RNN using Gated Recurrent Units (GRU) (**?**) with 128 hidden units in each direction. Finally, the generator was a 3-layer GRU RNN with 1024 hidden units.

4.3.3 Neural Vocoder

Speech generation using a vocoder is limited by the reconstruction quality of that specific vocoder. To enable high quality output, we replace the vocoder with a learned parametric neural module. We use SampleRNN (**?**) as an enhanced function approximator for this purpose.

SampleRNN has recently been proposed to model extremely long-term dependencies in sequential data such as audio signals. The hierarchical structure in SampleRNN is designed to capture dynamics of a sequence at different time scales.

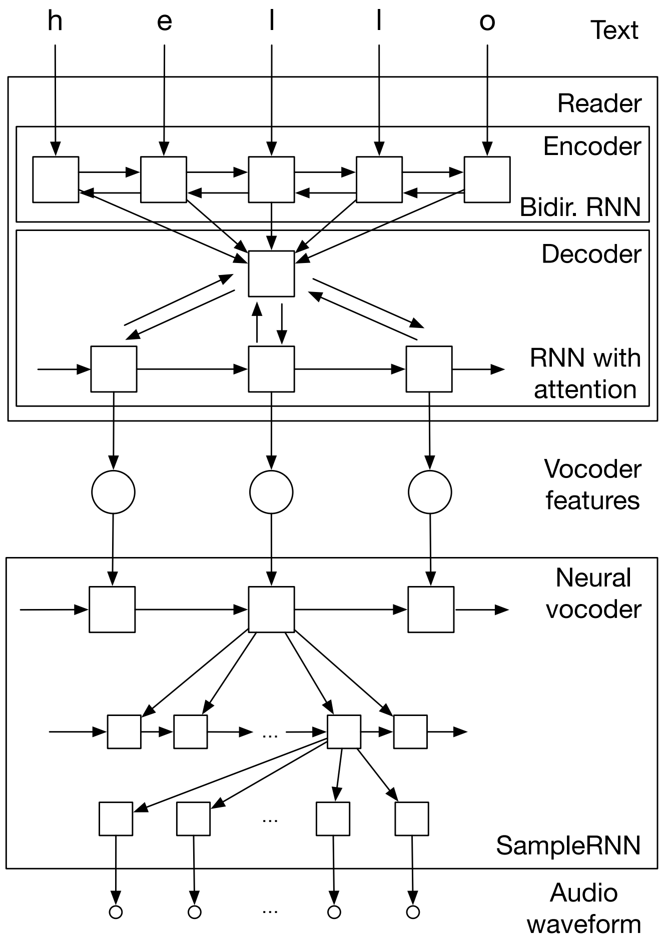


Figure 4.1 – Char2Wav: An end-to-end speech synthesis model.

This is necessary to capture long range correlations between distant audio timesteps (e.g. word-level correlations in speech signals) as well as nearby audio timesteps.

We use a conditional version of the same model with $K = 3$ tiers to learn the mapping from a sequence of vocoder features to the corresponding audio samples. Each vocoder feature frame v_t is added as an extra input for computation of the corresponding state in the top tier that is combined linearly with the input frame to that tier $f_t^{(k=3)}$:

$$inp_t^{(k=3)} = W_f f_t^{(k=3)} + W_v v_t \tag{4.13}$$

Following the notation in the original work, k represents the tier number and $inp^{(k=3)}$ is the input to the tier three. Hence, this module, which is the one operating at the slowest time-scale, has information about past audio samples along with the vocoder frame from the reader.

Each tier is a single layer GRU-RNN. The dimensions of the GRU and multilayer perceptrons are both 1024. The frame sizes for each tier are $FS^{(1)} = FS^{(2)} = 10$ and $FS^{(3)} = 80$. The outputs of this module are discrete values in 8-bit mu-law encoding, rather than 8-bit linear PCM.

4.4 Related Work

4.4.1 Speech Synthesis

There have been several advances made from using neural networks for parametric speech synthesis. **?** suggest using (feed-forward) Deep Neural Networks (DNN) to model the acoustic parameters of the vocoder. **?** present a lightweight model for mobile devices using RNNs. **??** present a comprehensive review of the progress made by using neural networks for acoustic modeling. Furthermore, there have been a few recent attempts to take away the vocoder and model the raw waveform directly (**????**).

4.4.2 Attention Models

Attention based models have been previously used in machine translation (**??**), speech recognition (**??**), and computer vision **?** among other applications. Our work has been heavily influenced by the work of Alex Graves (**??**). In a guest lecture Graves presented a speech synthesis model using an attention mechanism, an extension of his previous work on handwriting generation. Unfortunately, the speech extension was never published, so we cannot directly compare our approach to his work. However, his results were a key inspiration to us, and we hope that this work can be useful as a starting point for further developments in end-to-end speech synthesis.

4.5 Training Details

First, we pretrained the *reader* using WORLD vocoder features (**??**) as targets. We tried using mean squared error (MSE) as well as a Gaussian mixture model (GMM) likelihood as the cost function. We found that GMM achieved similar quality results but took longer to train. Therefore, we used MSE in all our experiments.

Since we use MSE, our reader is a deterministic function of the text and the speaker id. Equivalently, this corresponds to selecting the mean in a Gaussian model with fixed variance. This property of the reader allowed us to get vocoder frame predictions for all text examples in our training set which we stored and used to pre-train the sampleRNN module. This procedure allowed us to make the overall training procedure more efficient. Finally, we fine-tuned the whole model end-to-end. Our code is available online.[1]

We found this method to be extremely robust, with multiple experiments in different languages using nearly all the same hyperparameters, other than the attention step v. However, we did not perform a comprehensive hyperparameter search and an exhaustive search over various settings may further improve results.

4.5.1 Training the Neural Vocoder

Simply training the *neural vocoder* on full-length sequences is difficult in practice due to the fact that unfolding the SampleRNN module to the length of long sequences, with thousands of time steps, is inefficient. Hence, we have used the same training procedure detailed in the original work, i.e., stateful RNNs trained with truncated backpropagation through time on shorter subsequences.

We benefit from training this module at progressively more complex stages, in a process that resembles curriculum learning (**?**). We start with subsequences of 400 audio samples (corresponding to 5 vocoder feature frames), using vocoder features generated by the *reader* as inputs. Gaussian noise is then added to these conditioning features, with the hope that the module learns a contractive mapping to compensate for some of the errors from the *reader* and better generalize to unseen data. Additive Gaussian noise with variances of 0.3, 0.6, and 1.0 were progressively used, with the variance being increased every time learning was stalled. For the

1. http://github.com/sotelo/parrot

next stage, we increase the subsequence length from 400 to 2000, and finally to 4000 audio samples, while keeping the noise level fixed at 1. Doing so allows the network to take into account the dynamics necessary for longer audio contexts. This module did not have access to any other conditioning information, e.g. speaker ID.

4.6 Results

First, we provide samples from our model.[2]. In Figures 4.2, 4.3 and 4.4 we show samples generated by our model and their corresponding alignments to the conditioning information.

4.6.1 Listening Tests

In order to evaluate the outputs of our model, we performed a listening test consisting of two tasks. First, participants were asked to listen to pairs of samples from the same sentence generated by Char2Wav and reader with vocoder output. Twenty pairs of sentences were presented in randomized order, and participants were required to choose which sentence of each pair they considered more natural. They could also indicate they had no preference between both sentences in a pair. In this task, participants were allowed to listen to the stimuli as many times as they wished. Twenty two native Mexican Spanish speakers completed this task. The results for this task are presented in Figure 4.5. As we can see, speech generated by Char2Wav is still considered less natural than speech generated by the reader with vocoder output approach; however, in 38% of the cases, listeners either preferred the output of Char2Wav (21%) or had no preference (17%) between it and the output of the reader with vocoder output.

The second task aimed at measuring speech intelligibility. Three lists of ten sentences each, containing stimuli generated by either Char2Wav, reader with vocoder output, or natural speech processed by the WORLD vocoder, were presented at a random order. After listening to a sentence a single time, participants were asked to transcribe it to the best of their knowledge. Afterwards, we stripped both the

2. http://josesotelo.com/speechsynthesis

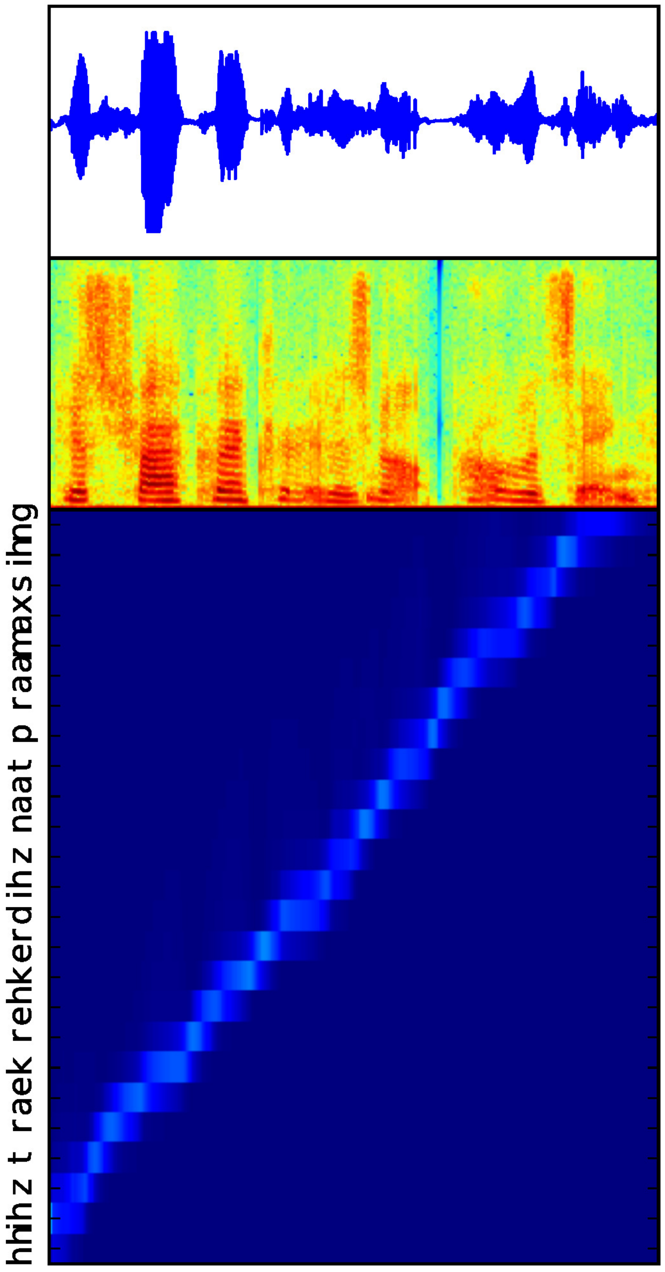


Figure 4.2 – Sample from the model conditioned on English phonemes. The model was trained on the VCTK dataset.

original transcriptions and the participants' transcriptions of diacritics and punctuation, then computed the average word-error rate (WER) for each model. This task was completed by 19 native Mexican Spanish speakers. Results for this task are presented as boxplots in Figure 4.6. Note that although the a word-error rate

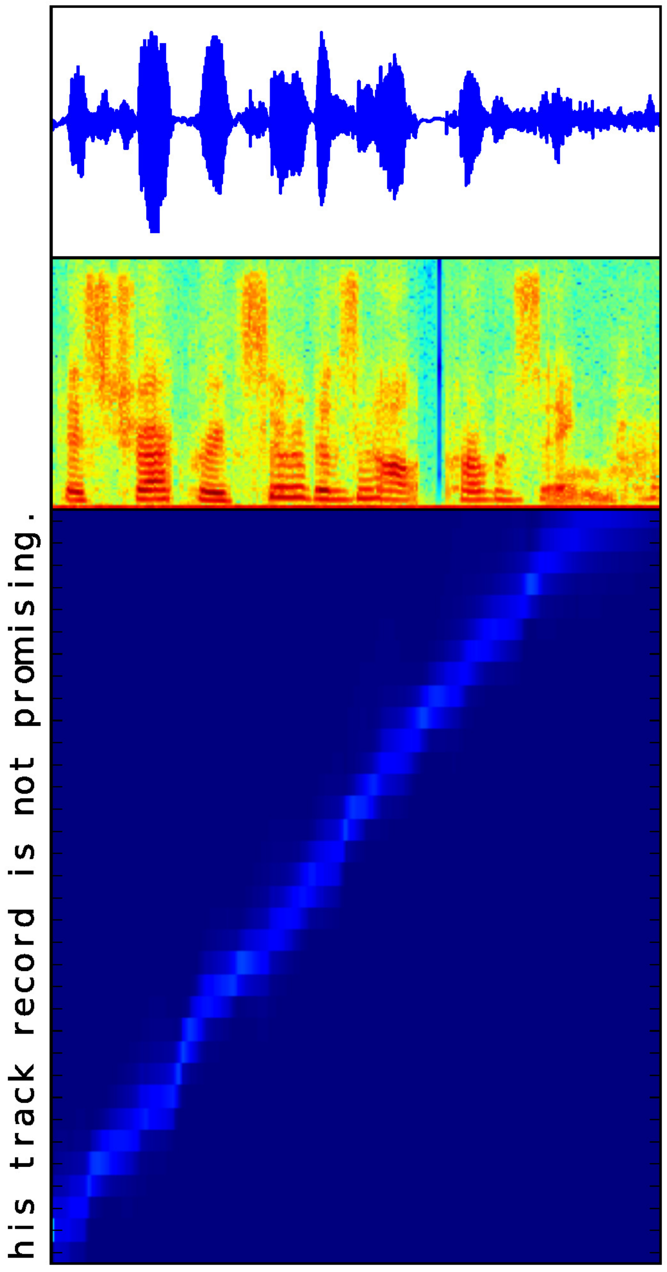


Figure 4.3 – Sample from the model conditioned on English text. The model was trained on the VCTK dataset.

of 10% seems high for natural vocoded speech, such results are typical for a task of this kind since some sentences are long, there is no context for the sentences and participants can only listen to them once. Both models had lower intelligibility than vocoded speech, with larger variability as well. We can observe that,

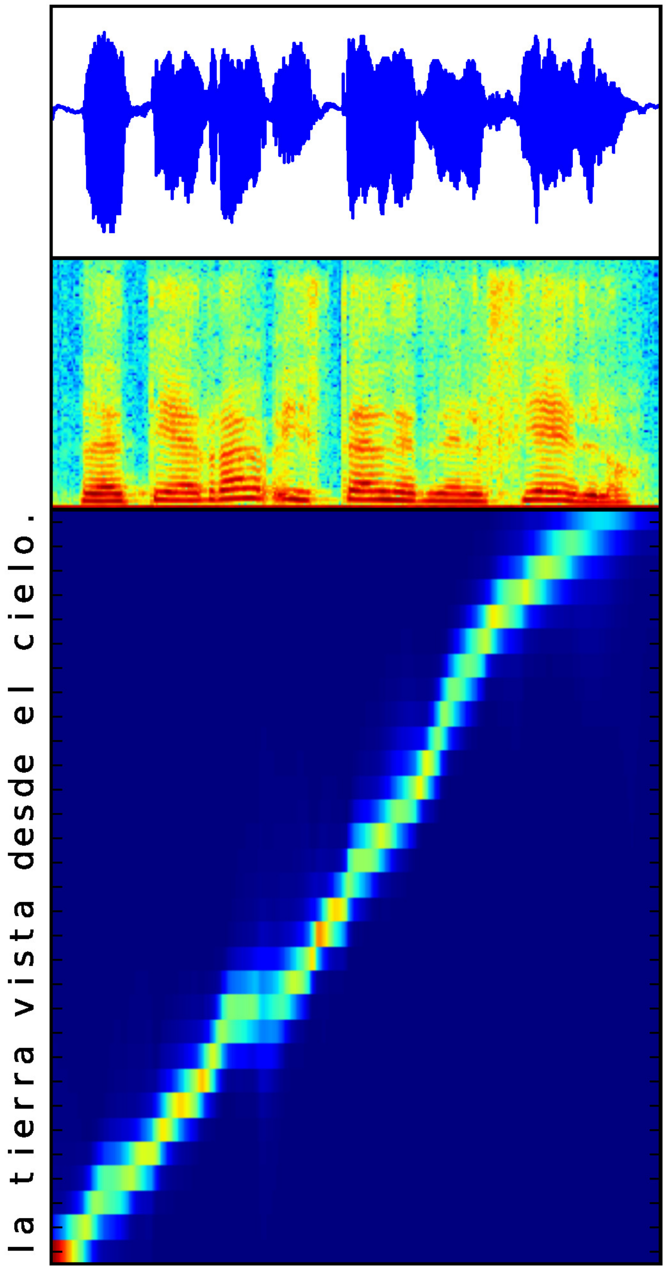


Figure 4.4 – Sample from the model conditioned on Spanish text. The model was trained on the DIMEX-100 dataset (**?**).

for example, some consonant-vowel bounds are not clearly defined in speech generated by our models. The task is hard even for vocoded and natural speech, and the added cognitive load due to the unnaturalness of the synthesized speech might have contributed to the higher error rates.

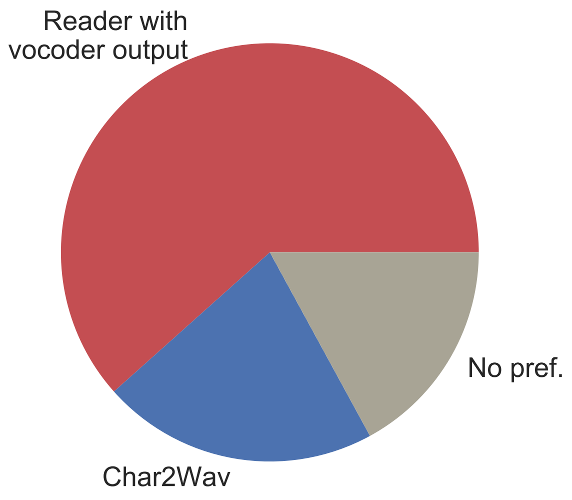


Figure 4.5 – Results of the preference test. Area on the graph corresponds to the proportion of participants that preferred the listed model.

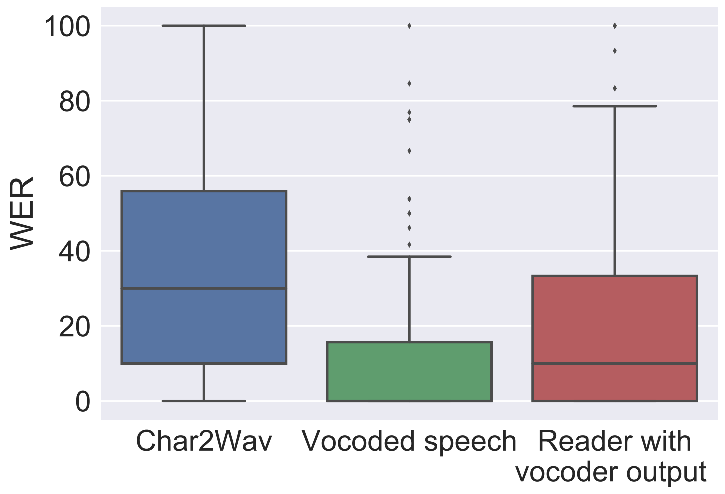


Figure 4.6 – Results of the intelligibility test. Word error rates are reported for the outputs of Char2Wav, original speech processed by the WORLD vocoder, and reader with vocoder output.

4.7 Conclusions

We show that end-to-end speech synthesis is feasible and present Char2Wav, a model that learns to produce speech signals directly from a sequence of phonemes or characters. The model is able to generate speech consistent with a given speaker identity. Furthermore, we showed that it is possible to interpolate between two speakers in the embedding space and generate voices with qualities not present in the original data.

Our approach is stronger for languages where spelling and pronunciation are closely related. For English text, where many character combinations have multiple valid pronunciations, we observe occasional failures in the generated samples. Further work is necessary to determine useful approaches for mapping text to speech in languages with less standard pronunciation rules. We expect that a stronger encoder will be able to ameliorate this issue. In particular, the text encoder developed by **?** appears to be a good compromise between using characters or words. In addition, the training sets we used are smaller than the datasets used for speech recognition. Increasing the size of the data would probably improve the results considerably.

5 Conclusion

In this chapter we reflect on the results obtained in this work and propose future directions of research in the area.

In Chapter 2, we developed an automatic stochastic gradient algorithm which reduces the burden of extensive hyper-parameter search for the optimizer. Our proposed algorithm exploits a lower variance estimator of curvature of the cost function and uses it to automatically tune adaptive learning rates for each parameter.

In Chapter 3, we proposed a novel model for unconditional audio generation based on generating one audio sample at a time. We showed that our model, which profits from combining memory-less modules, namely autoregressive multi-layer perceptrons, and stateful recurrent neural networks in a hierarchical structure is able to capture underlying sources of variation in the temporal sequences over very long time spans, on three datasets of different nature. Human evaluation on the generated samples indicate that our model is preferred over competing models. We also showed how each component of the model contributes to the exhibited performance.

In Chapter 4, we presented Char2Wav, an end-to-end model for speech synthesis. Char2Wav has two components: a **reader** and a **neural vocoder**. The *reader* is an encoder-decoder model with attention. The encoder is a bidirectional recurrent neural network (RNN) that accepts text or phonemes as inputs, while the decoder is a recurrent neural network with attention that produces vocoder acoustic features. *Neural vocoder* refers to a conditional extension of SampleRNN which generates raw waveform samples from intermediate representations. We showed results in English and Spanish.

Furthermore, we recognize that end-to-end speech synthesis is an area that needs further development. There are still two directions that we need to explore further that will improve our results. First, the approach we described in this work is weak for languages where writing and pronunciation are not closely related. We expect that a better encoder will help. In particular, the text encoder developed by **?** might ameliorate this issue. In addition, the training sets we used are smaller than the datasets used for speech recognition. Increasing the size of the data would probably improve this issue as well. Second, we need to improve the quality of the *Neural Vocoder* presented in Chapter 4. Similar approaches have showed state of the art results in speech synthesis. Therefore, we believe that we can achieve even better results by experimenting further with this part of the model.