

000
001
002
003
004
005
006
007
008
009
010
011
012
013
014
015
016
017
018
019
020
021
022
023
024
025
026
027
028
029
030
031
032
033
034
035
036
037
038
039
040
041
042
043
044
045
046
047
048
049
050
051
052
053

THE COOPER UNION
ALBERT NERKEN SCHOOL OF ENGINEERING

A Partitioned Autoencoder
for Audio De-Noising

by
Ethan Lusterman

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Engineering

September 2016

Professor Sam Keene, Advisor

054 THE COOPER UNION FOR THE
055
056 ADVANCEMENT OF SCIENCE AND ART
057
058
059
060

061 ALBERT NERKEN SCHOOL OF ENGINEERING
062
063
064
065
066
067
068
069

070 This thesis was prepared under the direction of the Can-
071 didate's Thesis Advisor and has received approval. It was
072 submitted to the Dean of the School of Engineering and
073 the full Faculty, and was approved as partial fulfillment of
074 the requirements for the degree of Master of Engineering.
075
076
077
078
079
080
081
082
083
084
085
086

087 _____
088 Dean, School of Engineering Date
089
090
091
092
093
094

095 _____
096 Prof. Sam Keene, Thesis Advisor Date
097
098
099
100
101
102
103
104
105
106
107

Acknowledgements

This thesis would not be possible without the guidance and support from my advisor, Dr. Sam Keene. He has mentored me since I was an undergraduate, and I am grateful for him helping this project come to life. I also want to thank Christopher Curro, my informal second advisor who helped me to think outside the box and for whom the overall system architecture is named after.

I would like to thank Kate Thorsen for pushing me past my potential and encouraging me to stay positive despite the frustrations of research. Lastly, I would like to thank my friends and family for their support. This thesis would not have been possible without all their support.

Abstract

Traditional audio denoising systems are often linear time-invariant (LTI) and often require access to clean data to properly train to remove noise. Since clean audio is often unavailable, we build on a partitioned denoising autoencoder for denoising audio signals when clean examples are unavailable for training. In addition, the nonlinearity of a neural network architecture provides additional gains over standard linear models. We compare existing semi-supervised denoising systems as well as canonical supervised denoising autoencoders. We show that for moderate levels of noise, our autoencoder can outperform existing schemes.

216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269

Contents

1	Introduction	1
2	Background	3
2.1	Machine Learning	3
2.1.1	Regression	4
2.1.2	Overfitting and Curse of Dimensionality	4
2.1.3	Loss functions and Regularization	4
2.1.4	Gradient Stuff?	4
2.2	Neural Networks	4
2.2.1	Dense Layer	6
2.2.2	Denoising Autoencoder	6
2.2.3	Convolutional Layer	7
2.2.4	Network Training	7
2.2.5	Regularization	7
2.2.6	Choice of Activation Function	7
2.2.7	Minibatch Training	9
2.2.8	Batch Normalization	10
2.3	Signals and Systems	10
2.3.1	Signals	10
2.3.2	Convolution	11
2.3.3	Frequency Transforms	12
2.3.4	Windowing and Perfect Reconstruction	13
2.3.5	Window Size and Frequency v. Time Resolution Tradeoff	14
2.3.6	Noise and Signal-to-Noise Ratio	14
2.3.7	Magnitude and Phase Spectrum	14
3	Signal Model and Data	15
3.1	Network Input and Output	15
3.2	Signal and Noise Choices	17
3.3	Other Network Parameters	17

270	4 De-noising Architectures	18
271		
272	4.1 Supervised Autoencoder	18
273		
274	4.2 Partitioned Autoencoder	20
275	4.2.1 Phase Reconstruction	20
276		
277	4.3 Curro Autoencoder	20
278		
279		
280	5 Results	21
281		
282	5.1 Supervised Autoencoder	21
283	5.1.1 Batch Normalized Input	21
284	5.1.2 Non-Batch Normalized Input	21
285		
286	5.2 Partitioned Autoencoder	21
287		
288	5.3 Partitioned Curro Autoencoder	21
289		
290	5.4 Comparison of Loss Convergence	21
291		
292	5.5 Comparison of Mean Squared Error Convergence	21
293		
294	6 Conclusions and Future Work	22
295		
296	6.1 Conclusions	22
297		
298	6.2 Future Work	23
299	6.2.1 Models	23
300	6.2.2 Data	24
301		
302		
303		
304		
305		
306		
307		
308		
309		
310		
311		
312		
313		
314		
315		
316		
317		
318		
319		
320		
321		
322		
323		

324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377

List of Figures

1	Example neural network	5
2	Modified Rectified Linear Unit Activation	19
3	Loss at various SNRs for Supervised Single-Layer Autoencoder with Batch Normalization at the Input	21
4	MSE at various SNRs for Supervised Single-Layer Autoencoder with Batch Normalization at the Input	22
5	Loss at various SNRs for Supervised Single-Layer Autoencoder without Batch Normalization at the Input	22
6	MSE at various SNRs for Supervised Single-Layer Autoencoder with Batch Normalization at the Input	23
7	Loss at various SNRs for Single-Layer Partitioned Autoencoder [1]	23
8	MSE at various SNRs for Single-Layer Partitioned Autoencoder [1]	24
9	Loss at various SNRs for Single-Layer Curro Autoencoder	24
10	MSE at various SNRs for Single-Layer Curro Autoencoder	25
11	Loss Comparison of Various Networks at -6 dB	25
12	Loss Comparison of Various Networks at -3 dB	26
13	Loss Comparison of Various Networks at 0 dB	26
14	Loss Comparison of Various Networks at 3 dB	27
15	Loss Comparison of Various Networks at 6 dB	27
16	MSE Comparison of Networks at -6 dB	28
17	MSE Comparison of Networks at -3 dB	28
18	MSE Comparison of Networks at 0 dB	29
19	MSE Comparison of Networks at 3 dB	29
20	MSE Comparison of Networks at 6 dB	30

378	Table of Nomenclature
379	
380	
381	
382	
383	
384	
385	
386	
387	
388	
389	
390	
391	
392	
393	
394	
395	
396	
397	
398	
399	
400	
401	
402	
403	
404	
405	
406	
407	
408	
409	
410	
411	
412	
413	
414	
415	
416	
417	
418	
419	
420	
421	
422	
423	
424	
425	
426	
427	
428	
429	
430	
431	

1 Introduction

Advances in smartphone technology have led to smaller devices with more powerful audio hardware, allowing for common consumers to make higher quality recordings. However, recorded speech and music are subject to noisy conditions, often hampering intelligibility and listenability. The goal of denoising audio recordings is to improve intelligibility and perceived quality. A variety of applications of audio denoising exist, including listening to a recording of a band or an artist’s live performance in a noisy crowd, or listening to a recorded conversation or speech under noisy conditions.

A common technique for denoising involves the use of autoencoder neural networks. [2] Advances in parallel graphics processing units (GPU) and in machine learning algorithms have allowed for training deeper networks faster, utilizing more hidden layers with more neurons.

Prior work in denoising audio has involved the use of noise-free training data. Since common consumers do not often have access to clean audio, we seek to denoise without the use of clean audio. Other work has touched on such a semi-supervised scenario but was used more as a preprocessing step to a classification algorithm than as time-domain denoising. [1]

In this thesis, we compare several neural network architectures and problem scenarios, ranging from data input types, level of noise, depth of network, training objectives, and more. In Chapter 2, we present background information on machine learning, neural networks, and signal processing as well as prior work in audio denoising. In Chapter 3, we detail the problem formally as well as introduce our signal model and sourced data. In Chapter 4, we detail all considered network architectures. In Chapter 5, we compare results

486 from different data inputs, levels of noise, network architectures, and training
487
488 objectives and discuss methods of evaluation. Finally, we make conclusions
489
490 and recommendations for future work in Chapter 6.
491

2 Background

2.1 Machine Learning

Machine learning involves the use of computer algorithms to make decisions based on training data. Generally, this falls into categorizing input data (classification) or determining a mathematical function to determine a continuous output given an input (regression). Popular classification examples include recognizing handwritten digits (MNIST) as well as determining whether an image contains a cat or a dog. (REF) An example of a regression problem is determining the temperature given a set of input features (humidity, latitude, longitude, date, etc.).

Problems where training data contain input data vectors as well as the correct output vectors (targets) are known as supervised learning problems. Training a model to denoise audio where noise was introduced to the clean audio would be a supervised learning problem. On the other hand, training a model to denoise audio where the underlying clean signal is not known is an unsupervised learning problem. Different loss (objective) functions and neural network architectures can be exploited to accomplish denoising without the clean data.

For the purposes of this thesis, we use machine learning to determine an underlying nonlinear function that removes noise from time slices of audio (i.e. regression). These slices can then be pieced back together through overlap-add resynthesis. To clarify, this is a general linear model that maps an input noisy audio vector $y[n] = x[n] + N[n]$ to $\tilde{x}[n]$, a target denoised audio vector, where $x[n]$ is the underlying clean signal and $N[n]$ is the additive background noise.

2.1.1 Regression

A classical regression technique is linear regression, where one or more independent variables x_i are used to determine a scalar dependent variable y . The case of a single independent variable x is known as simple linear regression. More formally, for k independent variables, we would like to determine a weight vector \mathbf{w} and bias vector \mathbf{b} :

$$y_i = w_1 x_{i1} + \dots + w_k x_{ik} + b_i, \quad i = 1 \dots, n \quad (1)$$

$$\mathbf{y} = \mathbf{x}^T \mathbf{w} + \mathbf{b} \quad (2)$$

where the rows of \mathbf{x}^T are the example input observations and \mathbf{y} and \mathbf{b} are column vectors.

By extension, the case of linearly estimating a vector output giving a vector input is known as a generalized linear model. A canonical example would be estimating a sine wave $x[n]$ over some number of N samples given noisy samples $y[n] = x[n] + N[n]$.

2.1.2 Overfitting and Curse of Dimensionality

2.1.3 Loss functions and Regularization

2.1.4 Gradient Stuff?

2.2 Neural Networks

In this thesis, we deal only with feed-forward neural networks, which are essentially directed acyclic graphs (DAG) for computation. In other words, information only moves through the network in one direction. An example neural network is shown in Figure 1.

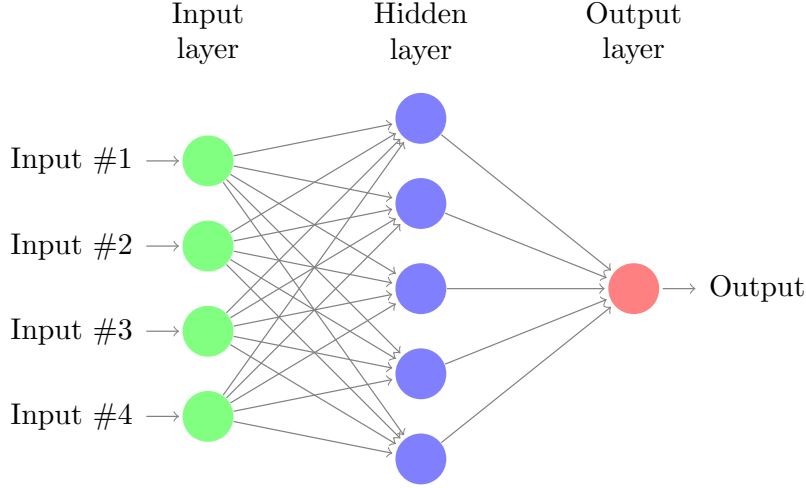


Figure 1: An example neural network. There are 4 input variables, 1 hidden layer with 5 neurons, and 1 output variable.

The connections in a neural network can be represented by linear combinations of the input variables with learned weights \mathbf{w} . [3] Unlike standard linear models however, neural networks apply a nonlinear activation $f(\cdot)$ at the output of each neuron. The circle nodes in a neural network diagram can be thought of as the sum of the linear combinations of the connection edges and the application of the bias and activation function. Therefore, a hidden neuron z_j in a network with N input variable nodes, M hidden nodes, and K output nodes takes on the value

$$z_j = f(a_j) \quad (3)$$

where the activation a_j is given by

$$a_j = \sum_{i=1}^N w_{ji}^{(1)} x_i + w_{j0}^{(1)} \quad (4)$$

The connection values w_{ji} are referred to as weights, and the scalars w_{j0} are referred to as biases. Note that the superscripted numbers refer to the Then, the output y_k is given by

$$y_k = g(a_k) \quad (5)$$

where the output activation a_k is given by

$$a_k = \sum_{j=1}^M w_{kj}^{(2)} z_j + w_{k0}^{(2)} \quad (6)$$

We are free to choose activation functions, which we will discuss later. However, note that at the output, the function $g(\cdot)$ is often an identity for regression problems and a sigmoid $\sigma(\cdot)$ for classification problems.

Often, the weights and biases are grouped into a weight vector \mathbf{w} . In other words, similar to the linear models described earlier, a neural network is a nonlinear function of input variables $\{x_i\}$ to output variables $\{y_k\}$ where the parameters of the function are learned via training techniques.

2.2.1 Dense Layer

Described in the previous section, we refer to a dense layer as a fully connected neural network, in which no interconnections between neurons are missing at each layer. Dense layers can be prone to overfitting. However, as we mention later, overfitting is not an immediate concern for the purposes of this thesis.

2.2.2 Denoising Autoencoder

An autoencoder is normally an abstraction of neural networks in which an encode function $\mathbf{Z} = f(\mathbf{X})$ and a decode function $\hat{\mathbf{X}} = g(\mathbf{Z})$ are learned to

learn a lower-dimensional representation of some input \mathbf{X} . [1] A denoising autoencoder is a supervised process whereby the clean input is first corrupted by some stochastic process $\mathbf{Y} = u(\mathbf{X})$. In other words, the neural network input would be a noisy input Y , and the network would try to learn weights such that the network output $\hat{\mathbf{X}}$ approximates the clean input \mathbf{X} . Another way to frame it is that your network is learning the inverse function of the noise process $u(\mathbf{x})$.

2.2.3 Convolutional Layer

2.2.4 Network Training

In order to train a neural network, we must update the weights such that we minimize a loss function, often some kind of sum-of-square error function. [3]

2.2.5 Regularization

2.2.6 Choice of Activation Function

The most common activation functions used are the logistic sigmoid function and the hyperbolic tangent. [2] The logistic sigmoid function is given by

$$g(x) = \frac{1}{1 + \exp(-x)} \quad (7)$$

Note that the sigmoid function has an output on the range $(0, 1)$. The hyperbolic tangent function (\tanh) is given by

$$g(x) = \frac{\sinh x}{\cosh x} \quad (8)$$

$$= \frac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)} \quad (9)$$

$$= \frac{1 - \exp(-2x)}{1 + \exp(-2x)} \quad (10)$$

Note that the hyperbolic tangent function has an output on the range $(-1, 1)$.

Generally, our choice of nonlinearity should be chosen such that the expected range of desired output matches the nonlinearity's. In the case of audio denoising, different activations can be chosen depending on the input format. For example, time-domain audio frames are often processed with a digital floating point representation on the range of $[-1, 1]$. In such a case, the hyperbolic tangent might be appropriate. On the other hand, if we were working with magnitude spectra of an audio signal, we would use a linearity with an output range of $[0, \infty]$.

Recently, a more popular activation function which has in use is the rectified linear unit (ReLU). [4] The ReLU is defined by the following:

$$g(x) = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases} \quad (11)$$

In other words, $g(x) = \max(0, x)$. This function satisfies the range of output we expect for magnitude spectra. In terms of gradient calculations, the zero derivative for negative input values of x can cause nodes to not be activated, potentially leading to gaps in information at the output and slower training time. To combat this, variations of the ReLU are used which have small, non-zero gradients for negative input values. For example, leaky ReLU's are defined by

$$g(x) = \begin{cases} 0.01x & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases} \quad (12)$$

Advantages of ReLU's include better gradient propagation as well as fast computation and sparse representation. Some disadvantages include non-differentiability at $x = 0$. Also, depending on use case, sparse representation might not be desired.

2.2.7 Minibatch Training

Historically, neural networks were trained one example at a time (online) or in a batch (all examples at once). [5] For the online approach, the network weights are updated after gradients are calculated and backpropagated for each training example. On the other hand, the batch approach accumulates average gradients for all examples and then updates the network weights. The batch approach might approximate the true gradients better than the online approach, but the online approach tends to have faster training time and convergence. [5] This is because with an online approach, the network is less likely to get stuck in a local minimum.

Minibatch training has become more popular recently. Serving as a midway point between the two approaches, minibatch training exposes the network to a small number of examples and then accumulates gradients and updates the network weights. The trend toward minibatch training comes at a time where parallel computing resources are easily accessible.

2.2.8 Batch Normalization

Batch normalization is a technique that helps to speed up training time and convergence. Batch normalization accumulates learned statistics of the network to help achieve loss convergence more quickly. More formally,

2.3 Signals and Systems

Domain knowledge of discrete audio signals and systems better informs our decisions for an audio denoising system, so some background information on signals and systems as it pertains to this thesis is detailed below.

2.3.1 Signals

We deal exclusively with discrete-time audio signals in this thesis. A discrete-time audio signal $x[n]$ is represented as a sequence of numbers (samples), where each integer-valued slot n in the sequence corresponds to a unit of time based on the sampling frequency f_s . This comes from sampling the continuous-time audio signal $x_c(t)$:

$$x[n] = x_c(nT) \quad (13)$$

where $T = 1/f_s$. For example, a 1-second speech signal sampled at 8kHz has 8000 samples. Furthermore, digital signals also have discrete valued sample amplitudes. For the purposes of this thesis, the bit depths of computers we use for analysis are high enough to allow for perfect reconstruction between continuous-time signals and digital signals.

We also assume signals collected have been properly sampled according to the Nyquist-Shannon sampling theorem, which states that a discrete-time signal

must be sampled at at least twice the highest frequency present in the signal to prevent aliasing of different frequencies. For example, speech signals generally have information up to 8kHz, so many speech signals are sampled at 16kHz. Music is more complex in that signals often span up to about 20kHz, so CD quality recordings are often sampled at 44.1kHz or higher. For this thesis, we use recordings sampled at 44.1kHz or lower.

2.3.2 Convolution

The discrete-time convolution operation takes two sequences $x[n]$ and $h[n]$ and outputs a third sequence $y[n] = x[n] * h[n]$:

$$y[n] = \sum_{k=-\infty}^{\infty} x[k]h[n-k] \quad (14)$$

Convolution is commutative, so $x[n] * h[n] = h[n] * x[n]$ holds true.

A linear, time-invariant (LTI) system is characterized by its impulse response $h[n]$, which allows us to determine samples $y[n]$ when $x[n]$ is subject to $h[n]$. For the purposes of this thesis, our underlying clean signal $x[n]$ might be subject to the conditions of an acoustic environment $h[n]$ and crowd noise $N[n]$:

$$y[n] = h[n] * x[n] + N[n] \quad (15)$$

In this scenario, our system would attempt to recover $h[n] * x[n]$ and possibly even $x[n]$ if the acoustic environment were deemed “noisy enough” due to echo and reverberation.

One of our proposed systems also incorporates convolutional neural networks (CNN) which use convolutions between frames of samples instead of simple linear combinations (discussed later).

2.3.3 Frequency Transforms

In some of our proposed systems, we use a frequency transformed version of the input signal as a preprocessing step to the system input. While no new information is gained from transforming the input, networks often respond better to determining the value of the magnitude of varying frequencies at a time slice instead of the individual time samples.

The frequency transform we use in this thesis is the discrete-time Fourier transform (DTFT). A sequence of N discrete-time samples is transformed into another sequence of N samples where each index then corresponds to a frequency bin. The DTFT $X[k]$ of a signal $x[n]$ is given by the following:

$$X[k] = \sum_{n=0}^{N-1} x[n] W_N^{kn} \quad (16)$$

where the twiddle factor W_N is given by $W_N = e^{-j(2\pi/N)}$. Then the reconstruction of $x[n]$ from $X[k]$ is given by:

$$x[n] = \frac{1}{N} \sum_{k=0}^{N-1} X[k] W_N^{-kn} \quad (17)$$

In this thesis, we also exploit the main duality between the time and frequency domain using the convolution theorem, which states that convolution in time is equivalent to multiplication in frequency and vice versa:

$$\mathcal{F}\{h[n] * x[n]\} = H[k]X[k] \quad (18)$$

$$\mathcal{F}^{-1}\{H[k] * X[k]\} = h[n]x[n] \quad (19)$$

This allows us to effectively treat our network as a non-linear filter that can denoise small time/frequency slices of our noisy signal, which can then be pieced back together using overlap-add resynthesis. We detail this in the next section.

2.3.4 Windowing and Perfect Reconstruction

To window a signal is to multiply a window function $w[n]$ by the frame, i.e. $w[n]x[n]$ over the frame length N . Because we are training a network to denoise small segments of a larger audio signal, we window the signal segments. This accommodates the finite-length requirement of the DTFT and helps to prevent spectral leakage. [DSPBOOK]

Also, to be able to properly reconstruct our signal, we use a window function and corresponding overlapping frame percentage to accomplish perfect reconstruction. The corresponding overlapping frame percentage is set such that the window sums to a constant for all time. For example, a rectangular window $w[n] = 1$ over an interval of length N has an overlap of 0% to sum to a constant 1 for all time. Another popular window is the Hanning window, defined over an interval N by the following:

$$w[n] = \frac{1}{2} \left(1 - \cos \left(\frac{2\pi n}{N-1} \right) \right) \quad (20)$$

For the Hann window, the perfect reconstruction overlap is a frame length of $N = 50\%$.

2.3.5 Window Size and Frequency v. Time Resolution Tradeoff

We must consider window size as a hyperparameter to our system. In general, shorter windows give rise to better time resolution at the cost of frequency resolution. On the other hand, longer windows give rise to better frequency resolution at the cost of time resolution. To illustrate, consider FIGURE.

2.3.6 Noise and Signal-to-Noise Ratio

Since we are trying to denoise audio signals, we must discuss how we measure noise. One of the most common measures of degradation of signal quality from additive noise is signal-to-noise ratio (SNR), defined as the ratio of signal variance to noise variance. [DSP] For the signal $y[n] = x[n] + N[n]$, where $x[n]$ is the signal of interest and $N[n]$ is the additive noise, the SNR is defined as

$$SNR = \frac{\sigma_x^2}{\sigma_n^2} \quad (21)$$

where σ^2 refers to the variance of the signal in question over some time interval. For the purposes of this thesis, we achieve desired a desired SNR for a simulation by scaling the noise to match the variance to the signal, then scaling the noise or the signal to achieve the desired SNR.

2.3.7 Magnitude and Phase Spectrum

3 Signal Model and Data

3.1 Network Input and Output

To simulate an audio denoising scheme, we define the following inputs and outputs. We take a known clean signal $x[n]$ which we subject to additive noise $N[n]$ using a specified SNR, resulting in the following noisy signal $y[n]$:

$$y[n] = x[n] + N[n] \quad (22)$$

To achieve a particular average SNR per simulation, we take the average signal energy for each minibatch of size B to determine a multiplicative scale factor k on the noise signal $N[n]$. For example, for additive white Gaussian noise (AWGN), we sample from the zero-mean, unit variance normal distribution (“randn” in Python) and determine our scale factor k as σ using the specified SNR in decibels:

$$\sigma_n^2 = \frac{1}{SNR_{lin}} \frac{1}{BN} \sum_{b=0}^{B-1} \sum_{n=0}^{N-1} x_b^2[n] \quad (23)$$

where SNR_{lin} is given by

$$SNR_{lin} = 10^{\frac{SNR_{db}}{10}} \quad (24)$$

In supervised scenarios, we allow the network to train with access to the ground truth $x[n]$. On the other hand, in semi-supervised scenarios, we only allow the network to train with access to a “soft label” indicating if the signal is (1) noise-only or (2) noise and possibly signal. [DanStowell] However, in both supervised and semi-supervised scenarios, our neural network input is one of the following:

1. Frames of $y[n]$
2. Frames of $\|Y[k]\|$
3. Magnitude spectrogram frames of $Y[k]$
4. Complex spectrogram frames of $Y[k]$

We choose the frame length L , time-domain window $w[n]$, and frame overlap percentage p as hyperparameters. Generally, we use 1024-sample frames at 16 kHz with a Hanning window with 50% overlap unless otherwise specified. In addition, for frequency frames, we use an FFT length the same length as our frame for a total of $L/2$ frequency bins. Note that our choice of frame length and sampling rate allows us to balance time and frequency resolution. With the given frame length and sampling rate, we achieve a frequency resolution of 15.625 Hz/bin by the following:

$$\frac{f_s/2 \text{ Hz}}{N/2 \text{ bins}} = \frac{f_s}{N} \quad (25)$$

$$= 15.625 \text{ Hz/bin} \quad (26)$$

Similary, our time resolution is given by

$$\frac{N}{f_s} = 64 \text{ msec} \quad (27)$$

Since we want to evaluate the level of denoising in the time domain, we recombine the network outputs with the noisy phase components of the spectrum if necessary to obtain an estimate $\hat{x}[n]$. We then compare $\hat{x}[n]$ to $x[n]$, in general using the mean squared error (MSE). For example, when our network outputs frames of $\|\hat{X}[k]\|$, we take the inverse Fast Fourier transform (IFFT) using the

noisy phase $\angle Y[k]$ and use overlap-add to recombine the frames. (Anything to add about phase denoising and failures here?)

3.2 Signal and Noise Choices

Our choice of signals include the following:

1. Sine waves with multiple frequencies and random amplitudes and phases
2. Clean speech signals
3. Studio music recordings
4. Live concert recordings

Similarly, our choice of noise signals include the following:

1. Additive white Gaussian noise (AWGN)
2. Restaurant noise

As mentioned above, we can use the average energy per minibatch to specify a given SNR for an experiment. We take several combinations of clean and noise signals and compare across multiple SNRs.

3.3 Other Network Parameters

Since our networks involve one or more neural network layers, we show some results compared to choices of nonlinearity, number of layers (depth), and number of nodes in each layer (width). Generally, we use an identity at the network output and either the rectified linear unit (ReLU), a modified ReLU (mReLU), leaky rectify, hyperbolic tangent (tanh), or an exponential linear unit (elu).

4 De-noising Architectures

In the following sections, we detail all considered shallow network architectures. Note that these network architectures can easily be extended to deep networks by adding corresponding encode and decode layers before and after the latent representation, respectively. These networks can be trained using the various inputs detailed in Chapter 3. However, for the purposes of presenting first results, we consider single FFT frames only to compare networks.

4.1 Supervised Autoencoder

We adopt the shallow supervised autoencoder from [2]. Used for supervised denoising, we adopt the relative network size as well as their modified nonlinear activation function. The network structure is a single hidden layer, dense neural network. In other words, we can represent our network output $\hat{X}_i[k]$ for various overlapping frames $i = 1, \dots, N$ by the following:

$$\hat{X}_i[k] = f_1(\mathbf{W}^{(1)}\mathbf{h}_i^{(0)} + \mathbf{b}^{(1)}) \quad (28)$$

$$\mathbf{h}_i^{(0)} = f_0(\mathbf{W}^{(0)}Y_i[k] + \mathbf{b}^{(0)}) \quad (29)$$

This network is trained to estimate the various layer weight matrices $\mathbf{W}^{(l)}$ and layer bias vectors $\mathbf{b}^{(l)}$.

Since we are estimating a magnitude spectrogram for values in the interval $[0, \infty)$, we use a nonlinear activation function whose support is on the same interval. A natural choice is the rectified linear unit (ReLU). However, as detailed in [2], the ReLU is subject to a 0-derivative for negative values. The modified ReLU used in [2], which we denote as mReLU, is given by the following:

$$f(x) = \begin{cases} x & \text{if } x \geq \epsilon \\ \frac{-\epsilon}{x-1-\epsilon} & \text{if } x < \epsilon \end{cases} \quad (30)$$

The choice of ϵ used in [2] is 10^{-5} . This modified ReLU allows for nodes to escape zero state since the derivative is always positive. An example plot of the nonlinearity is given in 2.

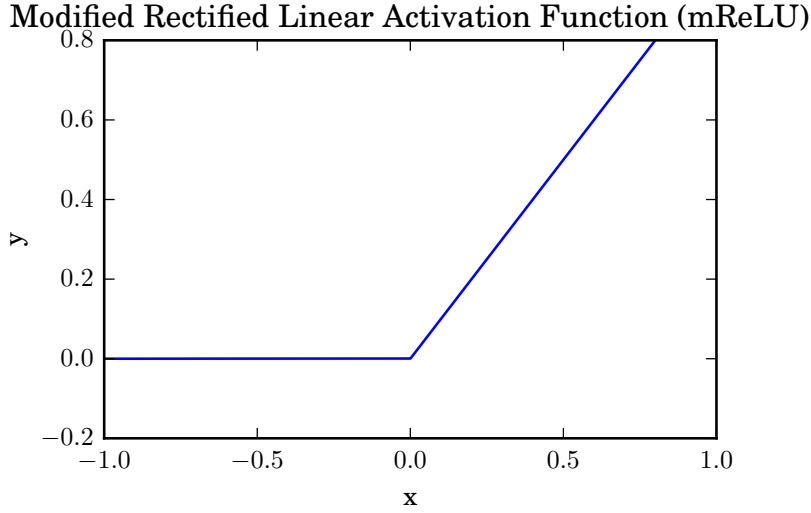


Figure 2: Modified Rectified Linear Unit Activation Function Plot

Since this network is supervised, we allow the training access to the original magnitude spectra $X[k]$. The loss function for training this network is defined as the mean squared error (MSE) between the network output and the clean spectra $X[k]$:

$$l(\mathbf{X}, \hat{\mathbf{X}}) = \|\mathbf{X} - \hat{\mathbf{X}}\|^2 \quad (31)$$

For our simulations, we also apply batch normalization at the input to help train more quickly and efficiently.

1458 4.2 Partitioned Autoencoder

1459

1460

1461

1462

1463

1464

1465

1466

1467

1468

1469

1470

1471

1472

1473

1474

1475

1476

1477

1478

1479

1480

1481

1482

1483

1484

1485

1486

1487

1488

1489

1490

1491

1492

1493

1494

1495

1496

1497

1498

1499

1500

1501

1502

1503

1504

1505

1506

1507

1508

1509

1510

1511

Adopted from [1], the partitioned autoencoder is a variation of a traditional autoencoder in which we do not know the noise corruption process or the underlying clean signals directly. This model more closely models a practical scenario.

1470 4.2.1 Phase Reconstruction

1473 4.3 Curro Autoencoder

5 Results

5.1 Supervised Autoencoder

For the following results, we show

5.1.1 Batch Normalized Input

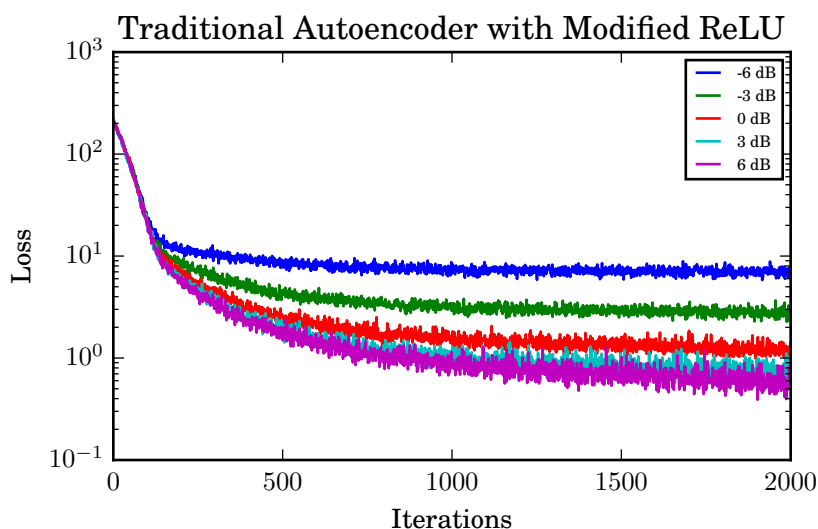


Figure 3: Loss at various SNRs for Supervised Single-Layer Autoencoder with Batch Normalization at the Input

5.1.2 Non-Batch Normalized Input

5.2 Partitioned Autoencoder

5.3 Partitioned Curro Autoencoder

5.4 Comparison of Loss Convergence

5.5 Comparison of Mean Squared Error Convergence

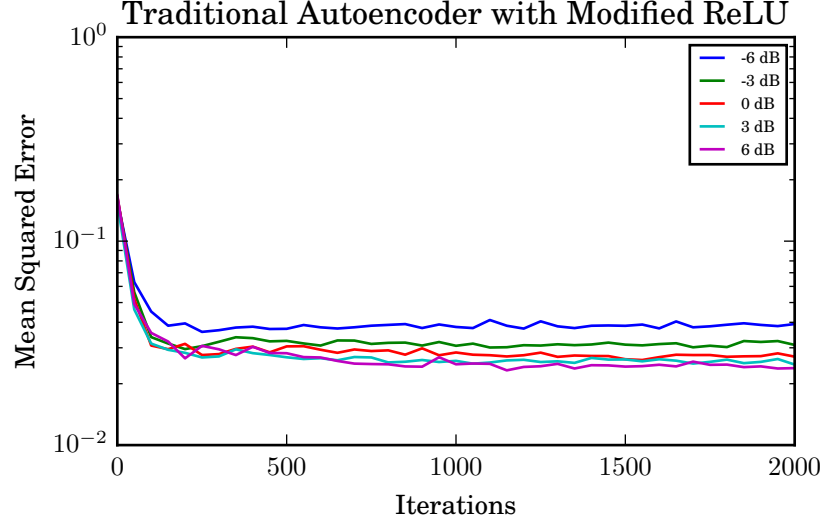


Figure 4: MSE at various SNRs for Supervised Single-Layer Autoencoder with Batch Normalization at the Input

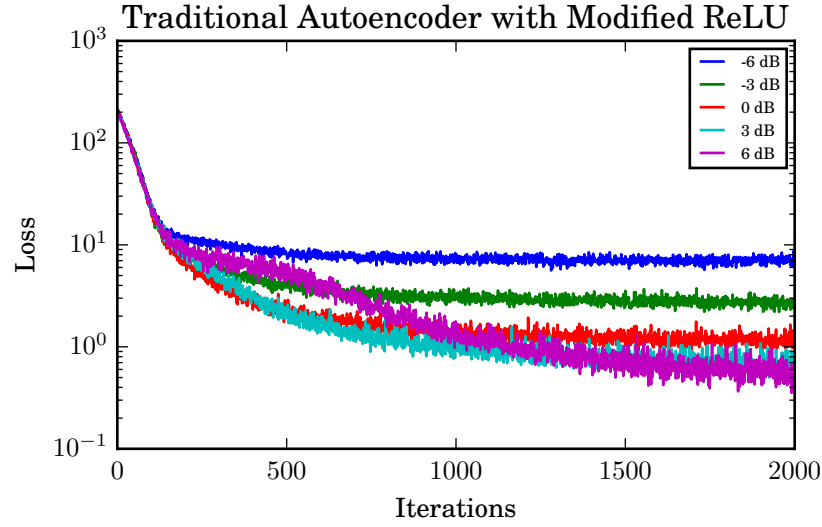


Figure 5: Loss at various SNRs for Supervised Single-Layer Autoencoder without Batch Normalization at the Input

6 Conclusions and Future Work

6.1 Conclusions

While more work is needed, deep partitioned neural network architectures using time and frequency data seem promising in long-term solutions for denoising speech and music signals.

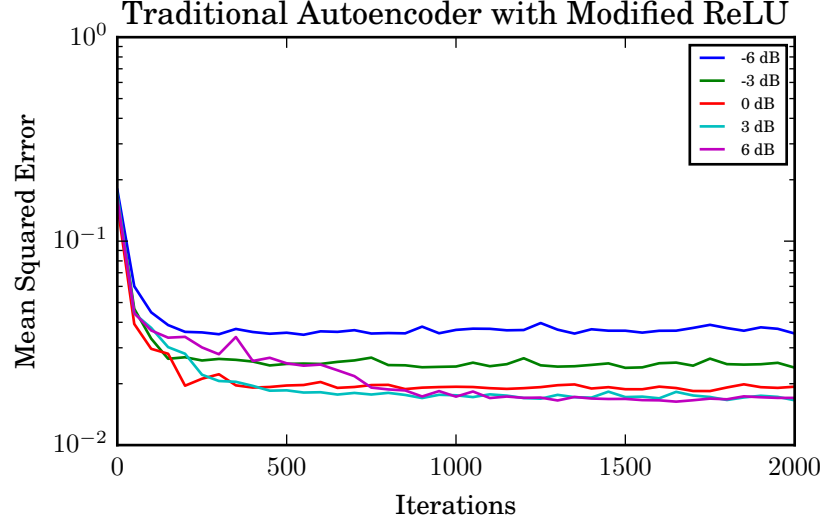


Figure 6: MSE at various SNRs for Supervised Single-Layer Autoencoder with Batch Normalization at the Input

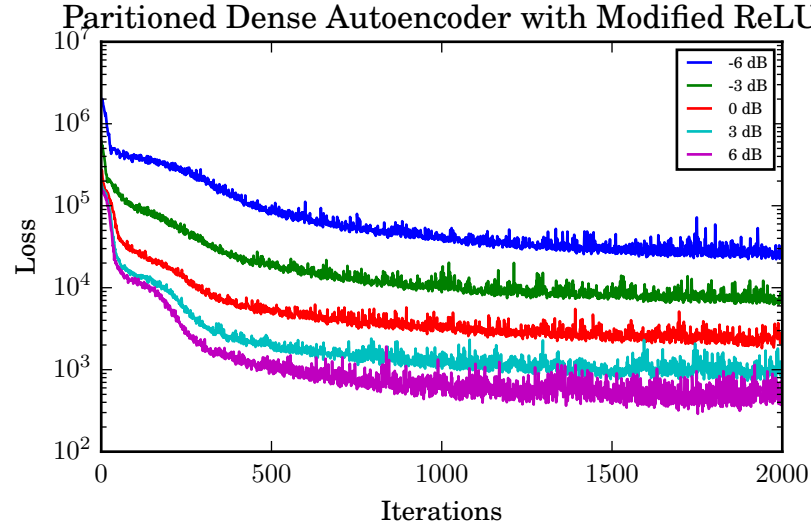


Figure 7: Loss at various SNRs for Single-Layer Partitioned Autoencoder [1]

6.2 Future Work

6.2.1 Models

Make network deeper. Consider gradual partitioning instead of hard.

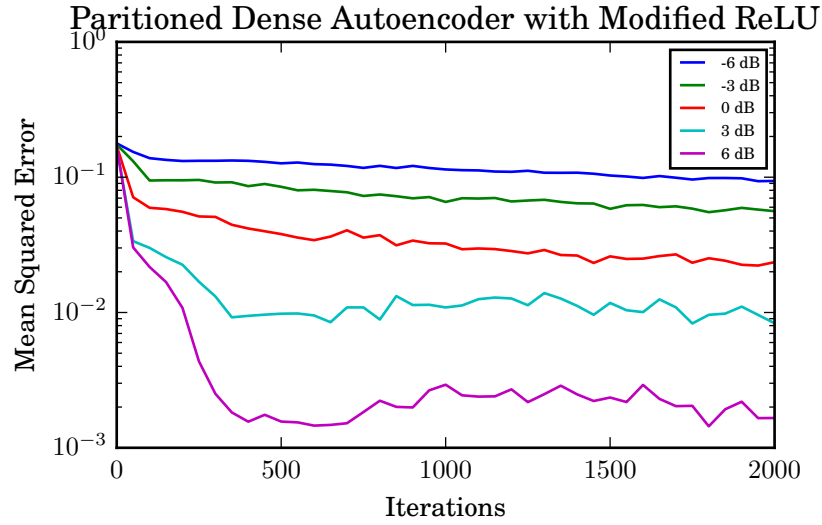


Figure 8: MSE at various SNRs for Single-Layer Partitioned Autoencoder [1]

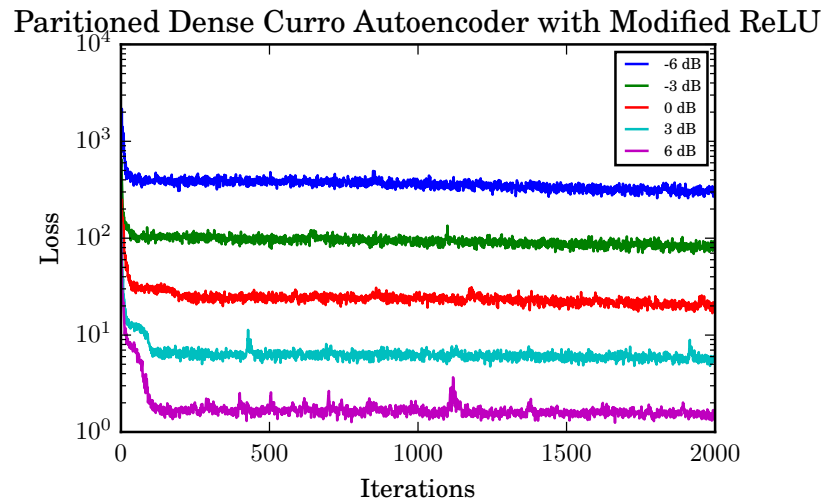


Figure 9: Loss at various SNRs for Single-Layer Curro Autoencoder

6.2.2 Data

Get more data. Consider different noise levels and types of signals.

Partitioned Dense Curro Autoencoder with Modified ReLU

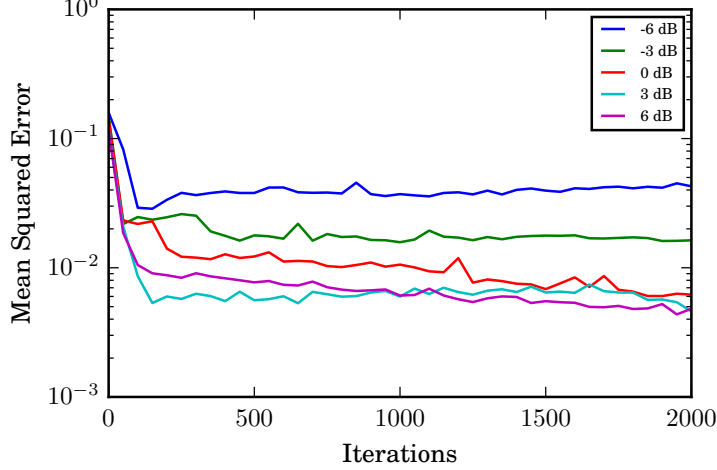


Figure 10: MSE at various SNRs for Single-Layer Curro Autoencoder

Comparison of loss of various networks at -6 dB

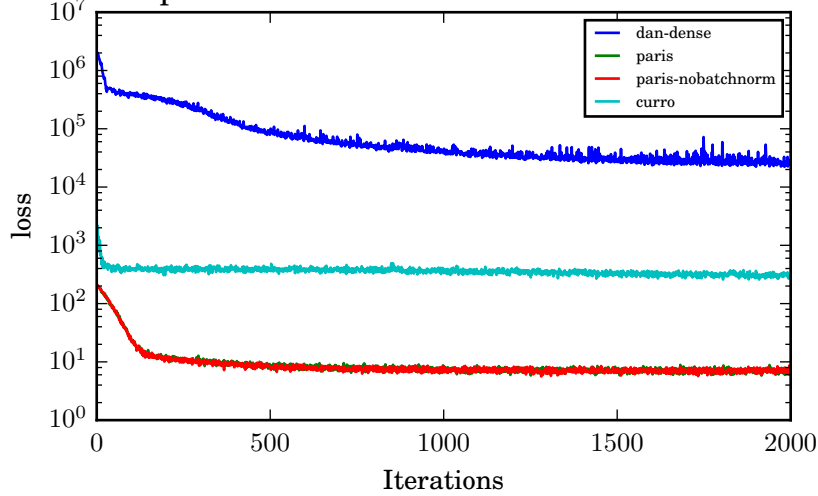


Figure 11: Loss Comparison of Various Networks at -6 dB

References

- [1] D. Stowell and R. E. Turner, “Denoising without access to clean data using a partitioned autoencoder,” *CoRR*, vol. abs/1509.05982, 2015. [Online]. Available: <http://arxiv.org/abs/1509.05982>
- [2] D. Liu, P. Smaragdis, and M. Kim, “Experiments on deep learning for speech denoising.” in *INTERSPEECH*, 2014, pp. 2685–2689.
- [3] C. M. Bishop, *Pattern recognition and machine learning*. springer, 2006.

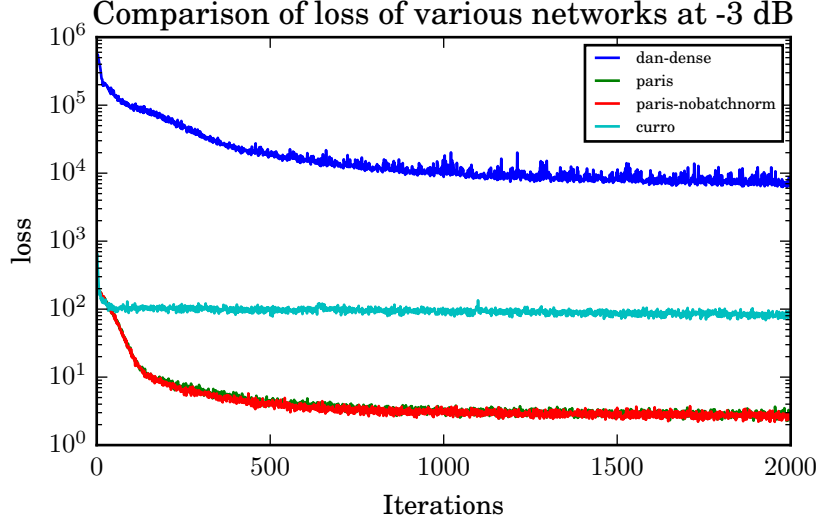


Figure 12: Loss Comparison of Various Networks at -3 dB

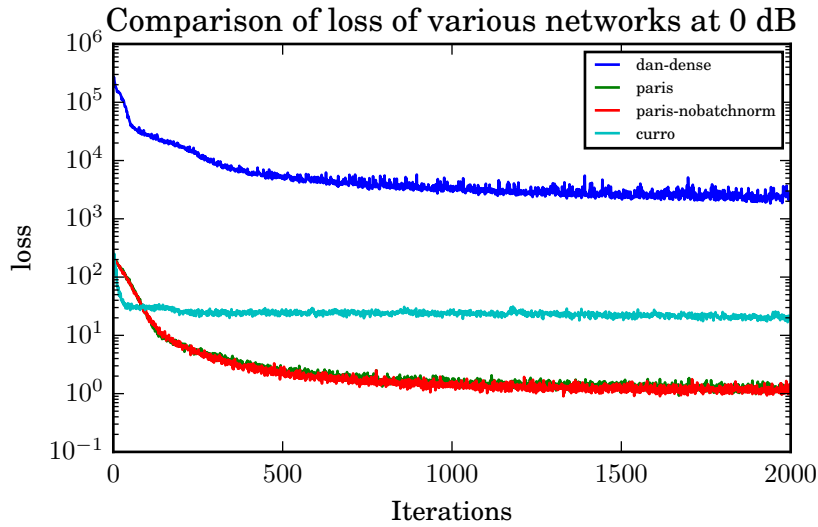


Figure 13: Loss Comparison of Various Networks at 0 dB

- [4] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks." in *Aistats*, vol. 15, no. 106, 2011, p. 275.
- [5] D. R. Wilson and T. R. Martinez, "The general inefficiency of batch training for gradient descent learning," *Neural Networks*, vol. 16, no. 10, pp. 1429–1451, 2003.
- [6] P. Baldi and Z. Lu, "Complex-valued autoencoders," *Neural Networks*, vol. 33, pp. 136–147, 2012.

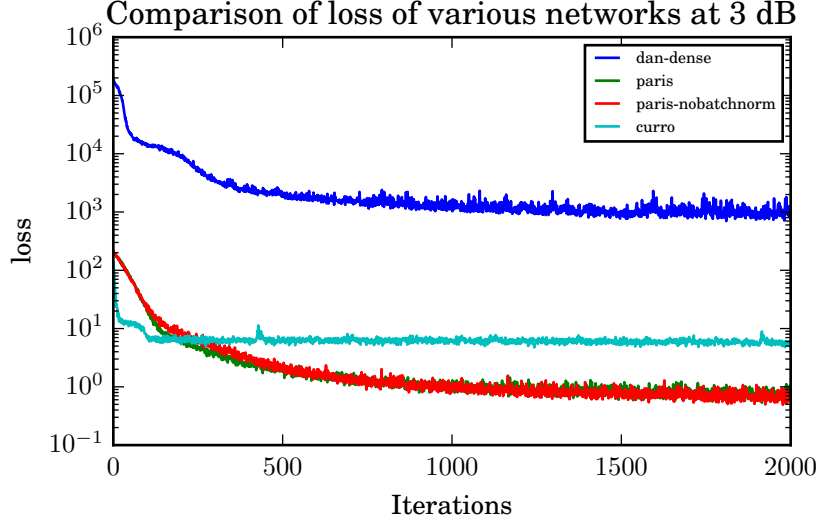


Figure 14: Loss Comparison of Various Networks at 3 dB

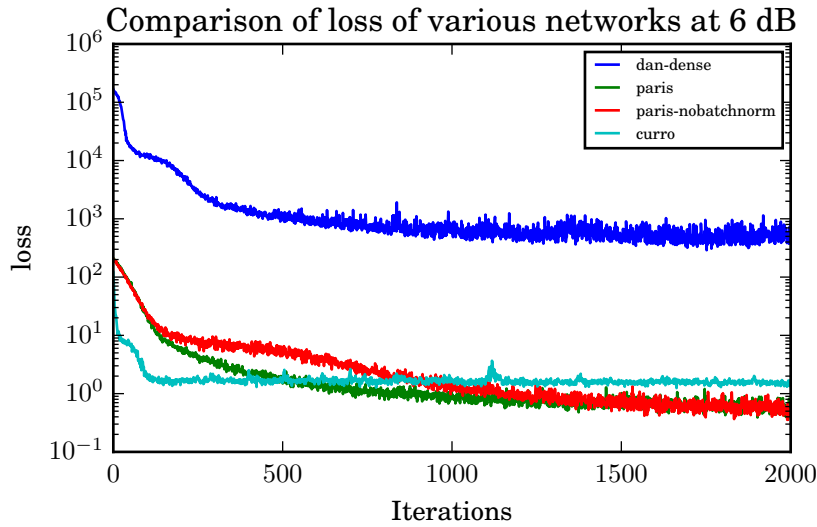


Figure 15: Loss Comparison of Various Networks at 6 dB

- [7] Y. Xu, J. Du, L.-R. Dai, and C.-H. Lee, "An experimental study on speech enhancement based on deep neural networks," *IEEE Signal Processing Letters*, vol. 21, no. 1, pp. 65–68, 2014.
- [8] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol, "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion," *Journal of Machine Learning Research*, vol. 11, no. Dec, pp. 3371–3408, 2010.
- [9] T. Ishii, H. Komiyama, T. Shinozaki, Y. Horiuchi, and S. Kuroiwa, "Re-

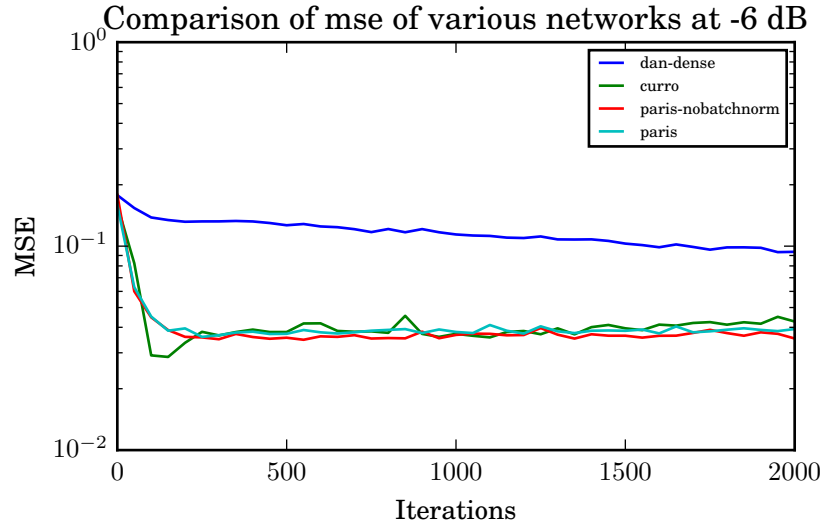


Figure 16: MSE Comparison of Networks at -6 dB

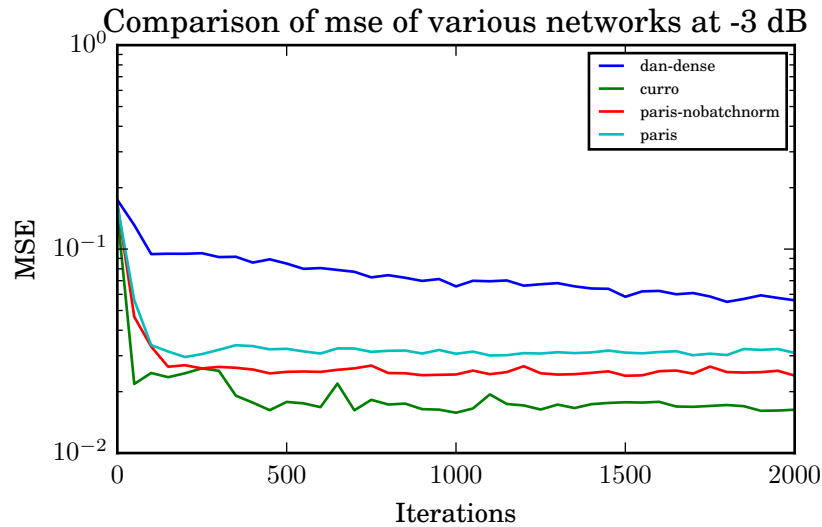


Figure 17: MSE Comparison of Networks at -3 dB

verberant speech recognition based on denoising autoencoder.” in *INTER-SPEECH*, 2013, pp. 3512–3516.

- [10] V. O. Alan, W. S. Ronald, and R. John, “Discrete-time signal processing,” *New Jersey, Printice Hall Inc*, 1989.
- [11] B. Gold, N. Morgan, and D. Ellis, “Speech and audio signal processing: processing and perception of speech and music,” 2011.
- [12] M. Kayser and V. Zhong, “Denoising convolutional autoencoders for noisy speech recognition.”

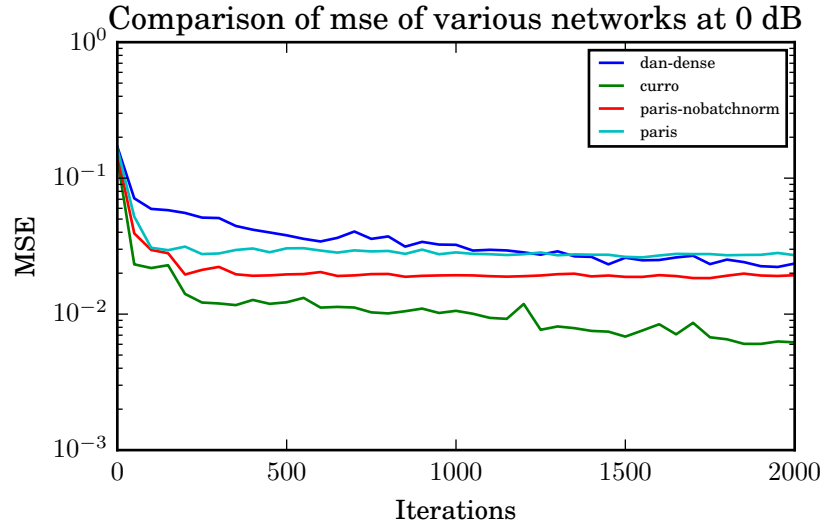


Figure 18: MSE Comparison of Networks at 0 dB

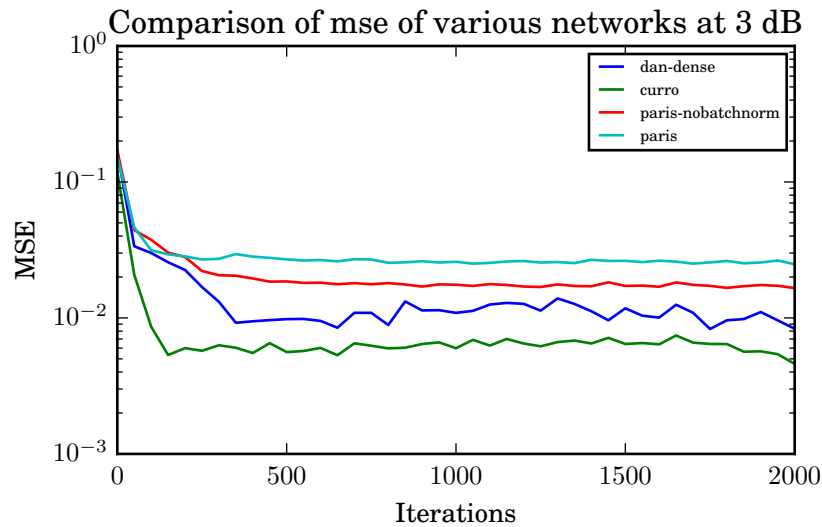


Figure 19: MSE Comparison of Networks at 3 dB

- [13] U. Zölzer, *Digital audio signal processing*. John Wiley & Sons, 2008.
- [14] G. E. Hinton and R. R. Salakhutdinov, “Reducing the dimensionality of data with neural networks,” *Science*, vol. 313, no. 5786, pp. 504–507, 2006.
- [15] A. Rad and T. Virtanen, “Phase spectrum prediction of audio signals,” in *International Symposium on Communications Control and Signal Processing (ISCCSP)*, 2012, pp. 1–5.

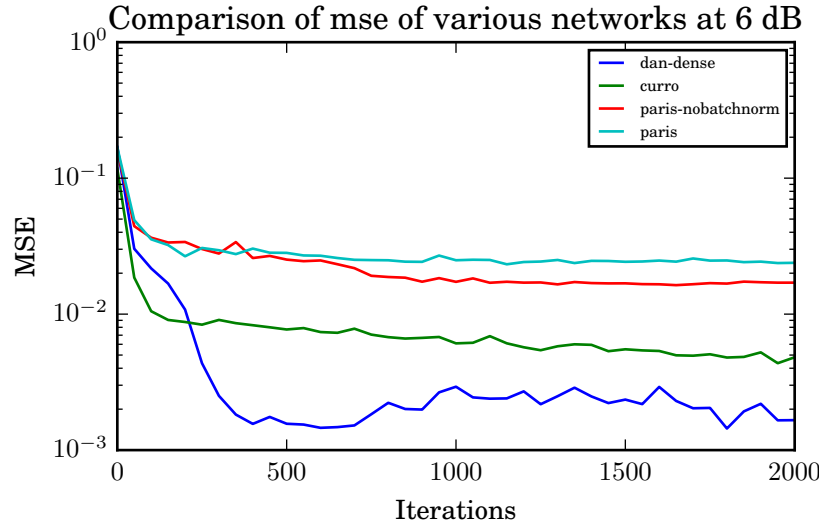


Figure 20: MSE Comparison of Networks at 6 dB

- [16] Theano Development Team, “Theano: A Python framework for fast computation of mathematical expressions,” *arXiv e-prints*, vol. abs/1605.02688, May 2016. [Online]. Available: <http://arxiv.org/abs/1605.02688>
- [17] S. Dieleman, J. Schlüter, C. Raffel, E. Olson, S. K. Sønderby, D. Nouri, D. Maturana, M. Thoma, E. Battenberg, J. Kelly, J. D. Fauw, M. Heilman, diogo149, B. McFee, H. Weideman, takacsg84, peterderivaz, Jon, instagibbs, D. K. Rasul, CongLiu, Britefury, and J. Degraeve, “Lasagne: First release.” Aug. 2015. [Online]. Available: <http://dx.doi.org/10.5281/zenodo.27878>
- [18] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” *arXiv preprint arXiv:1502.03167*, 2015.
- [19] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol, “Extracting and composing robust features with denoising autoencoders,” in *Proceedings of the Twenty-fifth International Conference on Machine Learning (ICML’08)*, W. W. Cohen, A. McCallum, and S. T. Roweis, Eds. ACM, 2008, pp. 1096–1103.
- [20] W. W. Cohen, A. McCallum, and S. T. Roweis, Eds., *Proceedings of the Twenty-fifth International Conference on Machine Learning (ICML’08)*. ACM, 2008.

2052 [21] (2010) Denoising autoencoders (da). [Online]. Available:
2053 <http://deeplearning.net/dA.html>
2054
2055 [22] S. Sonoda and N. Murata, “Decoding stacked denoising au-
2056 toencoders,” *CoRR*, vol. abs/1605.02832, 2016. [Online]. Available:
2057 <http://arxiv.org/abs/1605.02832>
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105