Master's thesis

2019

Mira Lilleholt Vik

NTNU
Norwegian University of
Science and Technology
Faculty of Information Technology and Electrical
Engineering
Department of Mathematical Sciences

Master's thesis

Mira Lilleholt Vik

Speech Enhancement with a Generative Adversarial Network

June 2019

NTNU
Norwegian University of
Science and Technology

NTNU
Norwegian University of
Science and Technology

NTNU
Norwegian University of
Science and Technology

Speech Enhancement with a Generative Adversarial Network

Mira Lilleholt Vik

Applied Physics and Mathematics
Submission date: June 2019
Supervisor: Gunnar Taraldsen
Co-supervisor: Femke Gelderblom

Norwegian University of Science and Technology
Department of Mathematical Sciences

Preface

This thesis concludes my studies at the Norwegian University of Science and Technology, in the field of Applied Mathematics. The work was carried out during the spring semester of 2019.

I would like to thank my co-supervisor Femke Gelderblom for her valuable insights in the field of Speech Enhancement and for having clear answers to all of my questions. I will also thank my supervisor, Gunnar Taraldsen, for our weekly meetings. I am grateful for the opportunity to combine deep learning and speech enhancement. It has been exciting to listen to the gradual improvements of the enhanced audio signals during the development of the program.

Lastly, I want to thank my fellow students and friends for making my years at NTNU such a great experience.

Trondheim, June 2018
Mira Lilleholt Vik

Abstract

Who has not been in a phone call distorted by background noises like traffic or wind? An algorithm able to denoise a distorted speech signal is of interest in many everyday situations. We have implemented a state-of-the-art deep learning algorithm for speech enhancement, a conditional generative adversarial net inspired by Pascual et al. (2017). The algorithm learns a mapping from noisy to clean speech through a two-player game between a generator and a discriminator. This approach is interesting because of two things: it enhances end-to-end and constructs the loss function in an untraditional way. It is hard to capture the quality and intelligibility of a noisy signal with a traditional loss function. Here, the loss function is learned based on competition between the generator and discriminator; the discriminator learns a loss for the generator's enhancement to be accurate.

Initially, the aim of a generative adversarial network (GAN) was to learn to generate samples from a training distribution. The generator receives latent random noise as input and maps to the wanted distribution. The latent noise makes the output of the algorithm stochastic. In the speech enhancement setting, noisy speech is used as a conditional variable in both generator and discriminator - the goal to learn an accurate mapping from noisy to clean speech. If the mapping is accurate, it is not of importance whether or not it is stochastic. Inspired by similar approaches in the image-to-image setting, we have compared the enhancement results for a conditional generative adversarial net with and without latent noise.

The algorithm was trained with speech signals from 220 different speakers from a Norwegian speech database and 99 different noise signals from two noise corpora with environmental noise recordings. The training files were constructed at speech-to-noise ratios $0, 10$ and 15 dB. The test set contains unseen speech and noise signals, combined at the same ratios of SNR, in addition to the unseen ratio 5 dB. Assessment of the performance of the generative adversarial network was evaluated objectively by use of the ITU-T standard Perceptual Evaluation of Speech Quality (PESQ) and the Short-Time Objective Intelligibility (STOI). There have also been some subjective reviews on the enhanced files from the student.

The proposed setup without latent noise perform comparable to the original setup with latent noise, but the scores obtained in terms of PESQ and STOI are slightly lower on average. Both implementations achieve improvements in PESQ similar to other implementations that are using a GAN framework for speech enhancement. The STOI scores decline a little after enhancement, but that might be partly because the STOI scores of the noisy test files were high to begin with. In general, the enhanced speech signals have a reduced noise level. Some of the enhanced signals have high-frequency artifacts and a degree of speech distortion.

The training progress is unstable. Early stopping could have been implemented to ensure that the final model is the best one of the different versions developed during training. Pascual et al. (2019) seems to have found solutions to both unstable gradients and high-frequency artifacts, but this article was not published before late April and was un-

fortunately discovered too late to be included in this work.

Sammendrag

Hvem har ikke vært i en samtale forvrengt av bakgrunnslyd som trafikk eller vind? En algoritme som kan forbedre et støyete talesignal er av interesse i mange hverdagslige situasjoner. Vi har implementert en deep learning algoritme for taleforbedring, et betinget generativt adversarielt nettverk inspirert av Pascual et al. (2017). Algoritmen lærer en transformasjon fra støyete til renere tale gjennom et topersonsspill mellom en generator og en diskriminator. Denne tilnærmingen er interessant på grunn av to ting: den forbedrer i tidsdomenet og konstruerer tapsfunksjonen på en utradisjonell måte. Det er vanskelig å fange både kvaliteten og forståeligheten til et støyende talesignal med en tradisjonell tapsfunksjon. Her læres tapsfunksjonen basert på konkurranse mellom generatoren og diskriminatoren; diskriminatoren lærer et tap for at generatorens forbedring skal være nøyaktig.

I utgangspunktet var målet med et generativt adversarielt nettverk (GAN) å lære og generere fra en treningsfordeling. Generatoren mottar latent tilfeldig støy som input og lærer en transformasjon til ønsket fordeling. Den latente støyen gjør generert output av algoritmen stokastisk. I taleforbedringssituasjonen brukes støyete tale som en betinget variabel i både generator og diskriminator - målet å lære en god transformasjon fra støyete til ren tale. Hvis transformasjonen er god, er det ikke viktig om outputet er stokastisk eller ikke. Inspirert av lignende tilnærminger i bilde-til-bilde-settingen, har vi sammenlignet forbedringsresultatene for nettverk med og uten latent støy.

Algoritmen ble trent med talesignaler fra 220 forskjellige talere fra en norsk taledatabase og 99 forskjellige lydsignaler fra to støydatabaser med naturlige støyopptak. Treningsfilene ble konstruert ved tale-til-støy-forhold på $0, 10$ og 15 dB.

Testsettet inneholder opptak fra 2 talere med 5 unike setninger hver. Det støyete testsettet ble konstruert ved å kombinere talesignaler med støysignaler, ved det usette talestøyforholdet 5 dB i tillegg til forholdene $0, 10$ og 15 dB. Støysignalene i testsettet er plukket ut for å være realistiske når det gjelder hva man møter i virkeligheten. Metodens ytelse ble evaluert objektivt ved bruk av ITU-T standarden ”Perceptual Evaluation of Speech Quality” (PESQ) og ”Short-Time Objective Intelligibility” (STOI). Det har også vært noen subjektive vurderinger på de forbedrede filene fra studenten.

Det foreslåtte oppsettet uten latent støy forbedrer sammenlignbart med det opprinnelige oppsettet med latent støy, men resultatene oppnådd i form av PESQ og STOI er noe lavere i gjennomsnitt. Begge implementeringene oppnår forbedringer i PESQ som kan sammenliknes med andre implementeringer som bruker et GAN-rammeverk for taleforbedring. STOI-poengene avtar etter forbedring, men det kan være delvis fordi input STOI-poengsummene til de støyete testfilene var høye. Generelt har de forbedrede talesignalene et redusert støynivå, men noen ganger på bekostning av høyfrekvente artefakter og litt taleforvrengning.

Treningsfremgangen er ustabil. ”Early stopping” kunne ha blitt implementert for å sikre at den endelige modellen er den beste av de forskjellige versjonene som ble utviklet under trening. Pascual et al. (2019) ser ut til å ha funnet løsninger på både ustabile gradienter og høyfrekvente artefakter, men denne artikkelen ble ikke publisert før slutten av april, og ble dessverre oppdaget for sent for å bli inkludert i dette arbeidet.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Background

The goal of speech enhancement is to improve the quality and/or intelligibility of a speech signal. Speech enhancement has been a field of research for several decades. Traditional approaches include spectral restoration, filtering techniques, and model-based methods (Benesty et al., 2008). Recent approaches view the problem as a supervised learning problem, and the progress in the field has accelerated after the breakthroughs of deep learning.

Tamura and Waibel (1988) applied feedforward multilayer perceptrons (MLPs) on speech enhancement already in the '80s. A range of deep neural nets have later been applied: convolutional neural nets (CNNs) (Hui et al., 2015; Fu et al., 2016; Park and Lee, 2016), recurrent neural nets (RNNs) (Weninger et al., 2015a,b; Erdogan et al., 2015), and generative adversarial networks (GANs) (Pascual et al., 2017; Michelsanti and Tan, 2017). The methods have been used in different manners. MLPs are trained supervised to find a mapping from the input to target functions. CNNs use shared weights, which lead to lower computational costs and local invariance, which is known to be an effective method for grid-like topologies. RNN's allow feedback connections and are known to be good for modeling of time series. GAN's are trained based on a two-player game between a generator and a discriminator.

A majority of the current systems are based on the short-time Fourier transform. The enhancement is done in the time-frequency (T-F) domain, and do often involve an assumption of the phase being of less importance; the noisy phase is often reapplied after enhancement of the spectral magnitude. That was questioned by Paliwal et al. (2011), which showed that the quality of the enhanced speech could be further improved by including enhancement of the phase spectrum. Recent interest in end-to-end methods is motivated by avoiding that assumption.

"What are meaningful optimization criteria for speech enhancement and how can they be mathematically formulated?" (Benesty et al., 2005, p. 62). Fu et al. (2018) pointed out that there has been a mismatch between the training's loss function and the evaluation criterion used on the enhanced speech. Speech enhancement algorithms use loss functions

not necessarily are justified mathematically from the goal of improved intelligibility and quality. The GAN approach, with a competition based loss function, is interesting in that aspect.

1.2 Motivation

It is not hard to imagine fields where noise reduction is of interest. In the field of telecommunication, noise reduction is important to improve the quality of conversations in noisy environments. The field of hearing aids and cochlear implants is another example. Hearing-impaired listeners do often have a greater problem with listening in noisy environments compared to normal-hearing listeners. Enhancement of the speech signal was found to substantially improve the intelligibility of listeners with cochlear implants in noisy environments (Yang and Fu, 2005). Speech enhancement is also used as a preprocessing step to make speech-to-text algorithms more robust to noisy environments. That is useful for smart assistants (Audio Software Engineering and Siri Speech Team, 2018), speaker recognition (Ortega-Garcia and Gonzalez-Rodriguez, 1996) and speech recognition (Hinton et al., 2012).

1.3 Approach

We will implement a generative adversarial network based on Pascual et al. (2017)'s speech enhancement GAN (SEGAN). In the speech enhancement GAN, a generator learns to map from latent noise and noisy speech to enhanced speech. The discriminator receives input pairs consisting of corresponding noisy and clean speech or noisy and enhanced speech and tries to label each pair correctly as being generated or real. The discriminator's loss function is based on the aim of making the correct guess, while the generator's loss function is based on the aim of convincing the discriminator that the noisy-enhanced pair is real and not generated. Through gradient-based training, the discriminator's loss is teaching the generator to do an accurate enhancement. Inspired by the Pix2Pix framework (Isola et al., 2016), who omitted the latent noise earlier used in GANs, we will compare the SEGAN inspired implementation with an alternative implementation without latent noise.

The performance of the GAN's is assessed through evaluation of the enhanced test set, in terms of the standard ITU-T method Perceptual Evaluation of Speech Quality (PESQ) and the intelligibility measure STOI (Short-Time Objective Intelligibility). In addition, minor informal listening tests are performed. The training and test data are constructed by use of the Norwegian speech database NB Tale (Nasjonalbiblioteket, 2016) and the noise corpora by Hu (2014) and Thiemann et al. (2013). The training data have speech-to-noise ratios $0, 10$ and 15 dB, while the test data, in addition, have the unseen ratio 5 dB.

The rest of the thesis is organized as follows. Chapter 2 introduces theory that has been relevant for the project. Some basic concepts from speech processing are defined, before necessary theory from machine learning, and deep learning is presented. Chapter 3 contains a description of the datasets, implementation, and experiment details. The results are in Chapter 4, while Chapter 5 contains the discussion. Lastly, Chapter 6 concludes the thesis with some final remarks.

Chapter 2

Basic Theory

The applied problem, speech enhancement, is from the field of speech processing. Essential concepts like speech quality, intelligibility, and signal-to-noise ratio will be defined. Generative adversarial networks are from the deep learning field. Deep learning theory will be presented, starting from the more basic building blocks before advancing to the generative adversarial network. Lastly, the setup of the speech enhancement GAN used as inspiration will be presented.

2.1 Speech enhancement

Speech enhancement aims to improve the intelligibility and/or perceptual quality of a speech signal. Different types of noise can corrupt a speech signal; noise is a term used for any unwanted signal that interferes with the signal of interest. The noise can be divided into four subcategories: additive noise coming from other sound sources, interfering signals from other speakers, reverberation caused by the reflection of speech on the surfaces nearby and echo resulting from the coupling between loudspeakers and microphones (Keintz et al., 2007, p. 844). Here, we will only consider additive noise. The speech enhancement/noise reduction problem can then be formulated as the goal of recovering a clean speech signal $\mathbf{x} = x(n)$ from the noisy signal $\tilde{\mathbf{x}} = \tilde{x}(n)$, where

$$\tilde{x}(n) = x(n) + v(n) \tag{2.1}$$

and $v(n)$ is the unwanted additive noise. We will focus on monaural speech enhancement, which is enhancement of speech recordings with only one microphone. More information regarding the location of the different sound sources is available when the recordings are done with more than one microphone.

2.1.1 Speech-to-noise ratio

A widely used measure of signal intensity relative to noise intensity is the signal-to-noise ratio (SNR). Throughout the thesis, the signal will be clean speech signal. Let P represent the power of a signal, A the root-mean-square amplitude and N the length. The SNR is defined as

$$\text{SNR} = \frac{P_{\mathbf{x}}}{P_{\mathbf{v}}} = \left(\frac{A_{\mathbf{x}}}{A_{\mathbf{v}}}\right)^2 = \frac{\left(\sqrt{\frac{1}{N}\sum_{i=1}^{N} x_i^2}\right)^2}{\left(\sqrt{\frac{1}{N}\sum_{i=1}^{N} v_i^2}\right)^2}. \tag{2.2}$$

SNR is usually measured in decibels (dB). The difference in decibels between two sound sources with power P_1 and P_2 is defined as

$$10 \log \frac{P_2}{P_1}, \tag{2.3}$$

where the logarithm is taken with base 10. The reference level used to give the sound level in absolute value is $20\,\mu\text{Pa}$. To find the P_2's sound level in dB one set P_1 equal the reference level. By combining Equation (2.2) with Equation (2.3), SNR can be expressed as

$$\text{SNR}_{\text{dB}} = 20 \log \frac{A_{\mathbf{x}}}{A_{\mathbf{v}}}, \tag{2.4}$$

in decibels. It is often of interest to decide the wanted level of SNR. Given a noise signal and a clean signal, a noisy signal with the wanted level of SNR_{dB} can be constructed as

$$\tilde{x}(n) = x(n) + \alpha v(n), \tag{2.5}$$

where the factor α is given by

$$\alpha = \frac{A_{\mathbf{x}}}{A_{\mathbf{v}} 10^{\text{SNR}_{\text{dB}}/20}}. \tag{2.6}$$

In the following, all SNR levels will be in decibels.

2.1.2 Speech quality and intelligibility

Speech quality is the overall impression of the quality of a speech signal. The perceived quality depends on factors like intelligibility, naturalness, loudness, and listening effort. Speech intelligibility is defined as ”the amount of speech understood from the signal alone” (Keintz et al., 2007, p. 223). It can be measured objectively as the fraction of words that listeners can perceive correctly. Speech intelligibility is affected by the quality of the speech signal, noise, and reverberation due to reflections in the surroundings. An example of a speech signal with high intelligibility, but low quality, is ”robot speech”: machine generated speech that sounds artificial or strange.

2.1.3 Evaluation measures

Quality

The mean opinion score (MOS) is a subjective quality assessment method where listeners rate the audio clip from quality 1: ”Bad” to quality 5: ”Excellent” (ITU, 1996), as specified in Table 2.1. An objective method with high correlation with listeners’ subjective rating is the ITU-T standard Perceptual Evaluation of Speech Quality (PESQ; Rix et al., 2001). The score range is $[0.5, 4.5]$, where a higher score corresponds to better speech quality.

Speech quality	Rating
Excellent	5
Good	4
Fair	3
Poor	2
Bad	1

Table 2.1: Grades in the MOS scale.

Intelligibility

A common objective measure for intelligibility is the Short-Time Objective Intelligibility (STOI; Taal et al., 2011). It is based on correlation between temporal envelopes of clean and noisy speech in short time segments. The range is normally between 0 and 1 and can be interpreted as an estimator for the percentage of words correctly perceived. STOI was shown to have high correlation with speech intelligibility by Taal et al. (2011), though Gelderblom et al. (2017) found that that one should not rely solely on STOI when predicting intelligibility.

2.2 Machine Learning Basics

Machine learning algorithms are algorithms that learn from data. A concise definition of learning in this context is: ”A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E” (Mitchell, 1997, p. 2).

One differs between the classes supervised and unsupervised learning. We say that the learning is supervised when the data is given in input-output pairs (x_i, y_i), for $i = 1, \cdots, N$ such that each input value x_i has a corresponding response variable y_i. Correspondingly is unsupervised training data with only measurements x_i, $i = 1, \cdots, N$ and no response variables. We will not go further into unsupervised learning here.

2.2.1 Supervised learning

The aim of supervised learning was described by (James et al., 2014, p. 26) as:

”We wish to fit a model that relates the response to the predictors, with the aim of accurately predicting the response for future observations (prediction) or better understanding the relationship between the response and the predictors (inference).”

Let $\mathbf{X}$ be the input data of dimension $(N \times p)$, where N is the number of observations, and p is the number of features. The corresponding response variable $\mathbf{Y}$ is of dimension $(N \times 1)$. We assume that there is some relationship between the input $\mathbf{X}$ and output $\mathbf{Y}$,

$$\mathbf{Y} = f(\mathbf{X}) + \boldsymbol{\epsilon}, \tag{2.7}$$

where $\boldsymbol{\epsilon} \in (N \times 1)$ is the random error term, representing the irreducible error. We seek to find an estimate of the function f such that we can predict the response variable for new observations x_0 and/or understand relations between the features and the response variable.

In our case, $\mathbf{X}$ is of dimension $N \times 1$, and represents a noisy speech signal of length N. The input-output pair is noisy speech and clean speech $(\tilde{\mathbf{x}}, \mathbf{x})$, and the aim is to find an accurate mapping from $\tilde{\mathbf{x}}$ to $\mathbf{x}$, i.e. find a function f such that $\mathbf{x} = f(\tilde{\mathbf{x}})$.

The Machine Learning approach is to first choose which algorithm or model to estimate f with, and thereafter train the model to fit the data set $(\mathbf{X}, \mathbf{Y})$. In many cases, the model chosen includes a hypothesis regarding the relation between input and output variables.

2.2.2 Model assessment

Let $\hat{\mathbf{Y}}$ be the predictions or estimates made by the algorithm on input $\mathbf{X}$. The performance of the model is evaluated by computing the distance between $\mathbf{Y}$ and $\hat{\mathbf{Y}}$. It is measured with a loss function $L(\hat{\mathbf{Y}}, \mathbf{Y})$. A common example is the mean squared error (MSE), given by $\mathrm{E}(Y - \hat{Y})^2$.

The validation set approach

It is normal to split the data set into three separate parts: a training set, a validation set, and a test set. The training set is used to train the algorithm. The algorithm's performance increases in general when it is exposed to more data, so most of the data should be in this group. A smaller part is put in the validation set. The validation set can be used during training to measure the generalization ability of the model while it is adapted. The test set should not be involved before the model is made. It is important that the test data are previously unseen by the net in order to get a realistic estimate of the test error.

The training MSE used to fit the model is computed by

$$\text{MSE} = \frac{1}{n} \sum_{i}^{n} (\hat{y}_i - y_i)^2, \tag{2.8}$$

where n is the number of observations.

One distinguishes between training loss and test/validation loss. The training loss is only a measure of how well the algorithm performs on the data the model is built upon. The test error, on the other hand, measures how well the model performs on unseen data.

This is often called the model's ability to generalize. The aim is to design an algorithm that performs well on new data, therefore is the test error of highest interest.

The Bias-Variance Trade-Off

Theoretically, the expected test MSE for a new observation can be decomposed into the sum of the components bias, variance, and irreducible error. Let (x_0, y_0) represent a new observation. The expected test MSE is then

$$\mathrm{E}(y_0 - \hat{f}(x_0))^2 = \mathrm{Var}(\hat{f}(x_0)) + \mathrm{Bias}(\hat{f}(x_0))^2 + \mathrm{Var}(\epsilon), \tag{2.9}$$

following the notation from James et al. (2014). The bias represents the expected distance between y_0 and predicted value $\hat{f}(x_0)$, $\mathrm{E}(\hat{f}(x_0) - y_0)$. The variance is a measure of how much $\hat{f}$ would change if it was estimated with a different training set. The variance due to the random error is irreducible. In order to minimize the expected test MSE it is thus necessary to simultaneously minimize bias and variance.

Generally does an increased complexity in the function estimation lead to higher accuracy, that is less bias. However, it also increases the risk of adapting to random patterns or noise in the data set, and do therefore not necessarily lead to better predictions on unseen data. An increase in model complexity may therefore correspond to an increase in variance. This trade-off is called the bias-variance trade-off. James et al. (2014) has illustrated the correspondence between flexibility and test MSE in Figure 2.1. In the left image, one can see data simulated from a distribution f, shown by the black line. Three estimates of f are shown. The orange line with the linear fit has the least flexibility and is a poor fit of the data. The blue line is a bit more flexible and gets the fit that is closest to the true distribution of f. The green curve is the most flexible curve and is following the data points even more closely. In the right image, the corresponding training MSE (grey curve) and test MSE (red curve) are displayed. The dashed line represents the irreducible error. One can observe that the training error is monotonically decreasing with increase in flexibility. However, the lowest training MSE does not correspond to the lowest test MSE, which has a U-shaped curve instead. That is a typical situation, the test error increases when the model gets too flexible and starts adapting to random noise in the data. This phenomenon is called overfitting.

Regularization

The different techniques used to avoid overfitting are called regularization techniques. "Regularization is any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error" (Mitchell, 1997, p. 117). We will go further into a couple of regularization techniques used in Deep Learning.

2.3 Deep Learning

Deep learning is a subfield within machine learning. While machine learning algorithms have a predefined hypothesis space where they can search for function estimates $\hat{f}$, do deep learning algorithms learn features from data by learning successive layers of increasingly

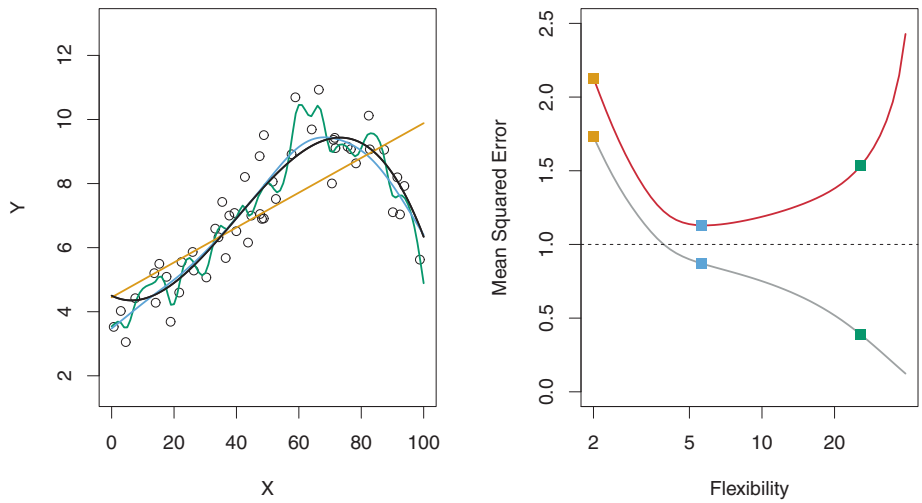


Figure 2.1: Left: data simulated from a distribution f (black line). A linear estimate (orange curve) and two smoothing splines (blue and green curves). Right: The corresponding test (red curve) and training (grey curve) MSEs. The dashed line represents the minimum possible test MSE, or the irreducible error. This is as seen in James et al. (2014)

meaningful representations. These layered representations are usually learned via neural networks. An advantage with neural networks is that no prior assumptions regarding the function shape are needed. Neural networks can approximate any function, according to Leshno et al. (1993).

Neural networks have proven capable of solving tasks that earlier were considered too complex for machine learning algorithms. Examples are near-human level image classification, speech recognition, improved machine translation, and improved ad targeting. One drawback with the neural nets is that they are often treated as a black-box - one does not know specifically how they will react to previously unseen input.

We will start by exploring the simplest form of a deep neural net, the deep feedforward neural network. Essential concepts like activation functions and back-propagation will be introduced in this section. Thereafter, convolutional neural networks will be explained, before we continue with generative adversarial neural networks. Both feedforward and convolutional neural networks can be used as building blocks in generative adversarial networks.

2.3.1 Deep Feedforward Neural Networks

The neural net is a supervised model that want to find a mapping from input x to output y. During the training procedure, the network is given both input and target (x, y), s.t. the network can measure its performance through a loss function, and update itself by use of a gradient-based strategy. One must specify the net architecture, the loss function, the

optimization algorithm, and additional features like regularization techniques.

Nodes and network architecture

A net is feedforward if the information flow is from input to output, without having any connections going the opposite way. Deep feedforward neural networks are commonly called multilayer perceptrons (MLPs). An illustration of a general MLP is shown in Figure 2.2. A neural network consists of nodes, separated in different layers. The first layer is called the input layer. It receives the input data as input. Behind the input layer are successive hidden layers before the last layer, which is called the output layer. In each layer, one must specify the number of nodes, that is the width of the layer, ($n^{(i)}$). In the input layer, there is usually a natural number due to the dimensions of the input data. Similarly, in the output layers, the width depends on the dimensions of the wanted output data. The depth of the model is the number of hidden layers + 1.

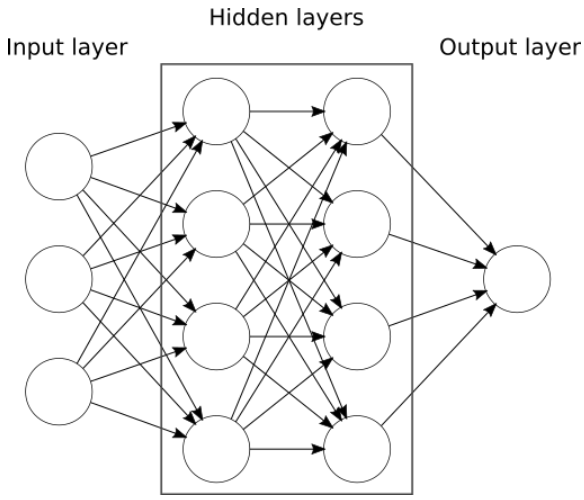


Figure 2.2: An example of a fully connected MLP with two hidden layers.

The nodes in all layers except the input layer receives an affine transformation of the previous layer as input. Let $\mathbf{W}$ represent the weight matrix, which is of dimension $m^{(i-1)} \times m^{(i)}$, such that there is a weight for each arrow between layer $i-1$ and layer i. Let $\mathbf{b}$ be the bias term. A net is called dense if it is fully connected, which means that all nodes in layer $i-1$ are connected to all nodes in layer i. The output from a layer is determined by a non-linear function g called an activation function. The different layers are computed successively: given the input values x, the first hidden layer is given by

$$\mathbf{h}^{(1)} = g^{(1)}(\mathbf{W}^{(1)T}\mathbf{x} + \mathbf{b}^{(1)}), \tag{2.10}$$

then the second layer is given by

$$\mathbf{h}^{(2)} = g^{(2)}(\mathbf{W}^{(2)T}\mathbf{h}^{(1)} + \mathbf{b}^{(2)}), \tag{2.11}$$

and further,

$$\mathbf{h}^{(i)} = g^{(i)}(\mathbf{W}^{(i)T}\mathbf{h}^{(i-1)} + \mathbf{b}^{(i)}), \tag{2.12}$$

for $i = 3, \cdots l$, where l is the output layer.

2.3.2 Activation functions

Each node has an activation function that is applied to the input from the previous layer. The activation functions' derivatives are important because the network is learning through gradient based training. The training stagnates when the gradient is zero or close to zero. There are many different choices of activation functions. We will consider the sigmoid function and the rectified linear unit (ReLU; Jarrett et al. 2009a; Nair and Hinton 2010; Glorot et al. 2011), and some of ReLU's generalizations.

Sigmoid units

Earlier, the sigmoid-function was the default choice in the hidden layers. It is defined by

$$g(x) = \frac{1}{1 + e^{-x}}, \tag{2.13}$$

and the range of the function is $(0, 1)$. One can observe in Figure 2.3 that it is saturating when x moves away from zero. Due to this, it has lost its role as the default choice in hidden layers. However, it is still a popular choice as activation function in the output layer. For example, when modeling a Bernoulli probability, a probability p for an event A and the probability $1 - p$ for the event A^c, the sigmoid function is a common choice as it has range $(0, 1)$.

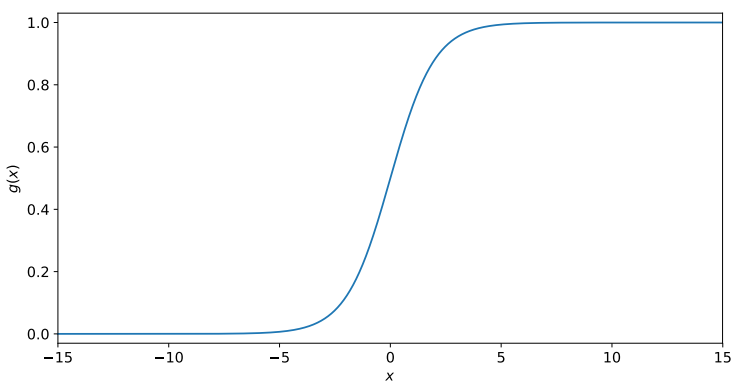


Figure 2.3: The activation function sigmoid is displayed for $x \in [-15, 15]$

Rectified Linear Units

The most popular hidden unit is the Rectified Linear Unit (ReLU), defined by

$$g(x) = \max(0, x). \tag{2.14}$$

It is differentiable in all points except in $x = 0$. The non-convexity of the activation function is not a problem in this setting because the gradient-based search for minimums usually does not actually arrive at a local minimum. Instead, it is satisfied with reducing the cost function's value significantly. It is therefore of greater importance that the derivative is 1 for all x greater than zero. In Figure 2.4 one can see that the derivative is zero on the left side of $x = 0$ and 1 on the right side of $x = 0$. Compared to the sigmoid function, that is a great improvement. While the sigmoid function saturates on both sides of $x = 0$, do ReLU have a derivative equal to 1 on the whole $\mathbb{R}_+$. For points smaller than $x = 0$, there is no effect from the training as the gradient is zero.

It is therefore important to initialize the net with a smart choice of start values such that most of the net has the possibility of being improved. It is common to initialize the constant term b with a small positive value, like 0.1 (Goodfellow et al., 2016, p. 187).

Jarrett et al. (2009a) compared different architectural choices and found that using a rectifying non-linearity was very important for the performance of a recognition system. The use of ReLU in hidden layers is one of the main reasons for the recent improvements in Deep Learning (Goodfellow et al., 2016, p. 219).

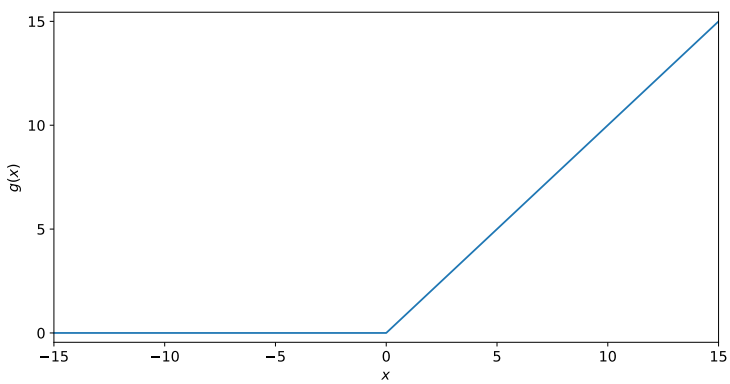


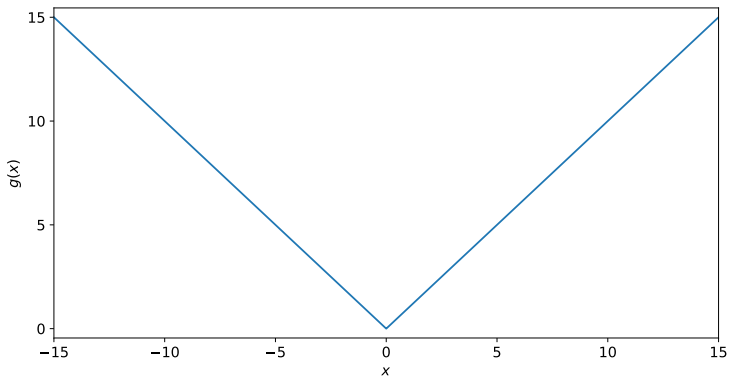
Figure 2.4: The activation function ReLU is displayed for $x \in [-15, 15]$.

Generalizations of Rectified Linear Units

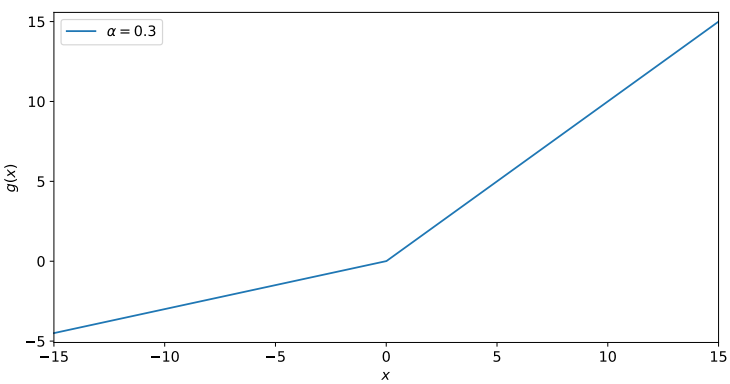
There exist several generalizations of ReLU. Most of these perform comparably, but occasionally better (Goodfellow et al., 2016, p. 187). Three generalizations are based on using a non-zero slope α for $x < 0$:

$$g(x) = \max(0, x) + \alpha \min(0, x). \tag{2.15}$$

Absolute value rectification fixes α to -1, and obtains $g(x) = |x|$. Jarrett et al. (2009b) used it for object recognition from images, where it is natural to want features that are invariant under a reversal of the input. A **leaky ReLU** (Maas, 2013) fixes α to a small positive value. Absolute value rectification and LeakyReLU with $\alpha = 0.3$ is plotted in Figure 2.5a and 2.5b. Another variant is parametric ReLU (**PReLU**; He et al., 2015), where α is treated as a learnable parameter.



(a) Absolute value rectification.



(b) The activation function LeakyReLU.

Figure 2.5: Two generalizations of ReLU is displayed for $x \in [-15, 15]$.

2.3.3 Training the net

The objective function used is typically written as an average of the loss function over the training set,

$$J(\boldsymbol{\theta}) = \mathrm{E}_{(x,y)\sim\hat{p}_{\text{data}}} L(f(\mathbf{x};\boldsymbol{\theta}), y), \tag{2.16}$$

where L is the loss function per sample, and the expectation is taken over the empirical distribution (Probability $1/n$ of drawing each sample x_i.) In order to train the DNN, the weights $\boldsymbol{\theta}$ are optimized such that the value of the loss function $J(\boldsymbol{\theta})$ is minimized. There exist several different optimization algorithms, which are based on stochastic gradient descent.

Gradient descent

In contrast to regular optimization do the algorithm not necessarily halt at a local minimum - instead it halts at a convergence criterion based on early stopping, that has reduced the objective function sufficiently. Early stopping is preventing the model from overfitting by storing a copy of the weights every time the validation set error reaches a new minimum. When the algorithm terminates are these weights returned instead of the latest obtained weights. This is inspired by the concepts discussed in Section 2.2.2

A gradient descent algorithm takes a step in a direction of descent. Let k be the index of the current step, α the step size f the objective function. Steepest descent is then given by

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha\nabla_{\boldsymbol{\theta}} f(\mathbf{x}|\boldsymbol{\theta})). \tag{2.17}$$

The gradient of the objective function $J(\theta)$ can often be expressed as a sum over the observations in the training set,

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \frac{1}{n}\sum_{i=1}^{n} L(f(\mathbf{x}^{(i)};\boldsymbol{\theta}), \mathbf{y}^{(i)}) \tag{2.18}$$

DNNs are often trained with very large data sets. Computing the gradient involves all the training data and is computationally expensive for large data sets. A common solution is to use stochastic gradient descent (SGD). Instead of computing the gradient based on all observations, it is computed on a mini batch of samples drawn randomly from the data set. That is, draw m' random samples from the training set, and estimate the gradient using only these observations,

$$g = \frac{1}{m'}\sum_{i=1}^{m'} L(f(\mathbf{x}^{(i)};\boldsymbol{\theta}), \mathbf{y}^{(i)}). \tag{2.19}$$

Back-propagation

The term back-propagation refers to the algorithm for numerical computation of the gradient of the total loss, J, with respect to the parameter values, $\boldsymbol{\theta}$, $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = L(\mathbf{y}, \hat{\mathbf{y}})$. It is computed by moving from the output layer and towards the input layer. The algorithm is based on the chain rule from calculus and was first introduced by Rumelhart et al. (1986).

We need to compute $\frac{\partial}{\partial \boldsymbol{\theta}_i} J(\boldsymbol{\theta})$ for $i = 1, \cdots N$, where $N = \sum_{i=1}^{l} (m^{(i-1)} + 1) m^{(i)}$ is the number of free weights. Because of the layer-based structure of the net does this include a chain of derivatives.

We are looking at one single input pair (x, y). $L(y, \hat{y})$ = $L(y, g^{(l)}(W^{(l)} h^{(l-1)} + b^{(l)})$, where $h^{(l-1)}$ is the input from the second outermost layer, which again is a function of the layer before and so on, as described in Section 2.3.1. To find the partial derivative of L with respect to a weight $W_{i,j}$ in layer $i = l$, one use the chain rule to first differentiate with respect to $g(W_{i,j})$, and then with respect to $W_{i,j}$:

$$\frac{\partial L(y, \hat{y})}{\partial W_{i,j}^{(l)}} = \frac{\partial L}{\partial g^{(l)}} \frac{\partial g^{(l)}}{\partial W_{i,j}^{(l)}}. \tag{2.20}$$

In order to compute the derivatives of the weights in previous layers, one must ”back-propagate” further inwards. This is a potentially computationally expensive algorithm, but through reuse of already computed derivatives can the computational cost be highly reduced.

The Adam optimizer

There are several different optimizers used in DNNs. Schaul et al. (2014) compared a range of different optimization algorithms and found that the algorithms with adaptive learning rates were more robust to hyperparameter tuning, though no single best algorithm was found. One common choice is Adam, which is an algorithm published by Kingma and Ba (2015). The pseudo code is presented in Algorithm 1, as seen in their article:

Like expressed in Algorithm 1, the aim is to minimize the value of an objective function $f(\theta)$ with respect to the parameters θ. We denote the gradient at time step t g_t. The parameters are updated along exponential moving averages of first and second order, where the exponential decay is due to the hyperparameters $\beta_1, \beta_2 \in [0, 1)$. Because the moments are initialized as a vector of zeros are the following moment estimates biased towards zero, especially in the first time steps and when the decay rate is small. There is therefore included a bias correction of both moments.

The algorithm uses a SGD approach with an adaptive learning rate and a momentum term. Momentum terms are inspired by mass times velocity from physics. It prevents the algorithm from moving in too different directions at successive steps, due to a mean of the previous steps being part of the gradient estimate. The adaptive step size is given explicitly by $\Delta_t = \alpha \cdot \hat{m}_t / (\sqrt{\hat{v}_t} + \epsilon)$. For convergence properties, check out the analysis by Kingma and Ba (2015) and Reddi et al. (2018).

The RMSProp optimizer

Another common choice is the RMSProp optimizer, an unpublished optimizer proposed by Hinton (2012) in a university course. The algorithm is regiven from Goodfellow et al. (2016) in Algorithm 2. An exponentially decaying average is used to discard history from past iterations, such that the method converges rapidly if it finds a convex bowl.

Algorithm 1 Adam, an adaptive stochastic optimization algorithm. With subscript t we mean a a parameter's value at time step t, while superscript t means exponentiating to the power t.

Require: α : Stepsize (Suggested default value: 0.001)
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for moment estimates (Suggested default values: 0.9 and 0.999, respectively)
Require: $f(\theta)$: Objective function with parameters θ
Require: θ_0 : Initial parameter vector
Require: ϵ : Small constant used for numerical stability (Suggested default:10^{-8})
$m_0 \leftarrow 0$ //Initialize 1st moment vector
$v_0 \leftarrow 0$ //Initialize 2nd moment vector
$t \leftarrow 0$ //Initialize time step
while θ_t not converged **do**
$t \leftarrow t + 1$
$g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ //Get gradients w.r.t. stochastic objective at timestep t
$m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ t //Update biased first moment estimate
$v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ //Update biased second raw moment estimate
$\hat{m}_t \leftarrow m_t/(1 - \beta_1^t)$ //Compute bias-corrected first moment estimate
$\hat{v}_t \leftarrow v_t/(1 - \beta_2^t)$ //Compute bias-corrected second raw moment estimate
$\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \hat{m}_t/(\sqrt{\hat{v}_t} + \epsilon)$ //Update parameters
end while
return θ_t //Resulting parameters

Algorithm 2 The RMSProp algorithm

Require: ϵ : Global learning rate
Require: ρ : Decay rates for moment estimates
Require: $f(\theta)$: Objective function with parameters θ
Require: θ_0 : Initial parameter vector
Require: δ : Small constant used for numerical stability (Suggested default:10^{-6})
$r \leftarrow 0$ //Initialize accumulation variable
while stopping criterion not met **do**
Sample a minibatch of m inputs from the training set $\{x^{(1)}, \cdots, x^{(m)}\}$, with corresponding targets $y^{(i)}$
$g \leftarrow \frac{1}{m} \nabla_\theta \sum_i L(f(x^{(i)}; \theta), y^{(i)})$ //Get gradients w.r.t. stochastic objective
$r \leftarrow \rho r + (1 - \rho) g \cdot g$ //Accumulate squared gradient
$\Delta\theta \leftarrow -\frac{\epsilon}{\sqrt{\delta + r}} \cdot g$
$\theta \leftarrow \theta + \Delta\theta$
end while
return θ //Resulting parameters

Batch Normalization

Ioffe and Szegedy (2015) introduced Batch Normalization to optimize network training. The Batch Normalization transform is presented in Algorithm 3. Batch Normalization is applied at one mini-batch at a time (this is how the training is organized, same as back-propagation), before the non-linearity (Section 2.3.2) is applied. The mean and variance of the mini-batch is calculated before the samples are normalized to have mean 0 and variance 1. A constant ϵ is added to the mini-batch variance to ensure numerical stability. Thereafter, the variables are scaled and shifted with the learned parameters γ and β. These parameters make the identity perform possible, if $\gamma = \sqrt{\text{Var}[x^{(k)}]}$ and $\beta^{(k)} = \text{E}[x^{(k)}]$. Thereby is the representational power of the network not limited by use of Batch Normalization.

Algorithm 3 Batch Normalizing Transform, applied to activation x over a mini-batch. Parameters to be learned: γ, β

Require: Values of x over a mini-batch $\mathcal{B} = \{x_{1,\ldots,m}\}$

$\mu_\mathcal{B} \leftarrow \frac{1}{m}\sum_{i=1} mx_i$ // mini-batch mean

$\sigma^2_\mathcal{B} \leftarrow \frac{1}{m}\sum_{i=1} m(x_i - \mu_\mathcal{B})^2$ // mini-batch variance

$\hat{x}_i \leftarrow \frac{x_i - \mu_\mathcal{B}}{\sqrt{\sigma^2_\mathcal{B} + \epsilon}}$ // normalize

$y_i \leftarrow \gamma \hat{x}_i + \beta$ // scale and shift

return $y_i \equiv \text{BN}_{\gamma,\beta}(x_i)$

Batch Normalization makes networks train faster, less sensitive to the initial weights and allows for higher learning rates. Furthermore is Batch Normalization working as a regulizer of the model. There exist other normalization variants like reference normalization and virtual batch norm (Salimans et al., 2016).

2.3.4 Convolutional Neural Networks

Convolutional neural networks (CNNs; Lecun 1989) were among the first neural networks to succeed. CNNs have been used by AT&T to read checks since the 1990s (Lecun et al., 1998). One of the reasons that the CNN was able to succeed earlier than the DNN is its lower computational requirements. While a DNN has layers that often are fully connected, which requires $O(m^2)$ parameters for a layer with width m, do a CNN use parameter sharing and local connections. This can dramatically reduce the number of parameters needed. CNNs are good at capturing local patterns in data and is therefore well suited for grid-like topologies such as images and time series. Other attractive features are sparse interactions, parameter sharing, and equivariant representations. The present application areas are object detection, object tracking, natural language processing, and speech recognition.

There exist a large number of different CNN architectures. We will go through the four typical layers in a CNN: a convolutional layer, a pooling or sub-sampling layer, a non-linear layer, and at last fully connected layers.

The convolutional layer

A convolutional layer applies a filter at its input. The filter's values correspond to the DNN's weights. A convolutional layer learns its weights in order to extract important features from the previous layer, thereby is the output of the applied filter often called a feature map.

In an MLP, layer i receives a weighted combination of layer $i-1$, where the weights in the weight matrix $\mathbf{W}$ are free. If the layers are fully connected, this involves $(m+1) \times m$ parameters. Revisiting Figure 2.2, observe that all connections have its own arrow. In a CNN, the weight matrix is substituted with a filter or kernel K, which often is of dimension $k << m$. The kernel is applied at the layer i by taking the dot products between size $k \times k$ areas of layer i and the kernel. The filter is applied with a specified stride between each receptive area.

Using a kernel with a dimension less than the layer width is equivalent to forcing the weight matrix $\mathbf{W}$ to use the same parameters at different positions. This is called parameter sharing, as the same kernel/weights are applied at several places, instead of learning individual weights for each connection $i \rightarrow i+1$. The interactions are sparse if the kernel is much smaller than the layer width. The sparse interactions can potentially dramatically reduce the number of needed operations. If the number of connections each node can have is limited to k, is there a decrease from $\mathcal{O}(m \times m)$ to $\mathcal{O}(k \times m)$ in run time (Goodfellow et al., 2016, p. 326).

The size of the feature map is controlled by the depth, stride, and zero-padding. Depth is the number of filters used. The stride is the number of elements between each frame taken from the input. It can be specified in all dimensions of the input. In the example in Figure 2.6 is a kernel of dimension 2×2 applied at an input tensor of dimension 4×4, with a stride of 2 elements in both directions. The colored areas mark which elements of the input that contribute to each element in the output. The grey frame results in the grey output frame when the kernel is applied, and similarly for the other colors.

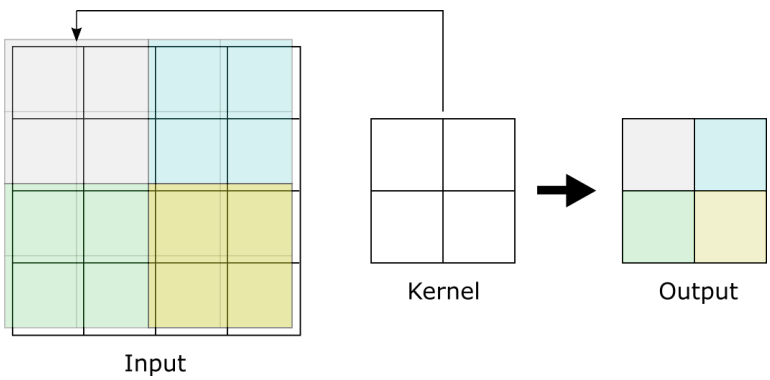


Figure 2.6: An illustrative example of convolution between an input of size 4×4 and a kernel of size 2×2 with stride 2. The kernel applied to the grey input area results in the grey output area, and so on for the other colors.

The concept zero-padding describes different ways of treating the elements close to

the input tensor's border. There are three common types of zero-padding. One option is to use no zero-padding, commonly called `valid` convolution. Only the frames where the kernel is fully contained in the input-matrix are used. In this way are all output elements a function of the same number of input elements. Observe that the size of the output shrinks: A kernel with dimensions $k \times k$ applied on a $m \times m$ matrix with stride 1 results in an output that is $m-k+1 \times m-k+1$. Another option is to use just enough zero-padding to output the same dimension as the input. This is called `same` convolution. Note that input elements near the border affect fewer output elements than elements further away from the border, which can make the border elements underrepresented in the model. That is the motivation behind the third type of zero-padding: `full` convolution. In this variant, there is added enough zeros around the border such that also the border elements influence the same number of output cells as the other input elements. Though, the output elements near the border are a function of fewer elements than output pixels further away from the border. That may make it difficult to find a kernel that performs well on all parts of the image. According to Goodfellow et al. (2016, p. 340) is the optimal amount of zero padding usually between `valid` and `same` convolution.

The non-linear layer

Introducing non-linear layers is what gives neural networks the stacking ability. If all the layers consisted of linear operations, the whole neural network would be a linear transformation. The non-linearities increase the representational abilities of a neural net. In the non-linear layer, there is used activation functions like the ones described in Section 2.3.2.

The pooling layer

A pooling layer summarizes the previous layer by use of a summary statistic like taking the average or choosing the maximum value. The pooling operation is specified and not learned, which means that the pooling layer is not adding any parameters to the network. The pooling operation makes the neural network invariant to small translations in the input. If there are small changes in the location of an input, the pooled feature map will still have the feature in the same location. The filter width is usually small (2 or 3 elements) to avoid losing too much details in the downsampling.

The fully connected layers

After layers of convolution, pooling and non-linearities, are the fully connected layers used to form a non-linear combination of the learned features.

2.3.5 Generative Adversarial Networks (GANs)

Generative Adversarial Networks (GANs) were first introduced by Goodfellow et al. (2014). The general situation is a wish of sampling from a complex, high-dimensional training distribution - but there is no direct way of doing this. A solution is to sample from a simple distribution, like random noise, and learn a transformation to the training distribution.

GANs are based upon a simple but elegant idea: A generator G learns to generate samples from a distribution X by trying to convince its opponent, the discriminator D that the sample is real and not generated. This is a two-player game where G is trying to fool D, while D is trying to classify samples correctly as being ”real” or ”fake”. An illustration is shown in Figure 2.7. Further, we will go deeper into the model and its training procedure, and review the theoretical convergence results presented by Goodfellow et al. (2014).

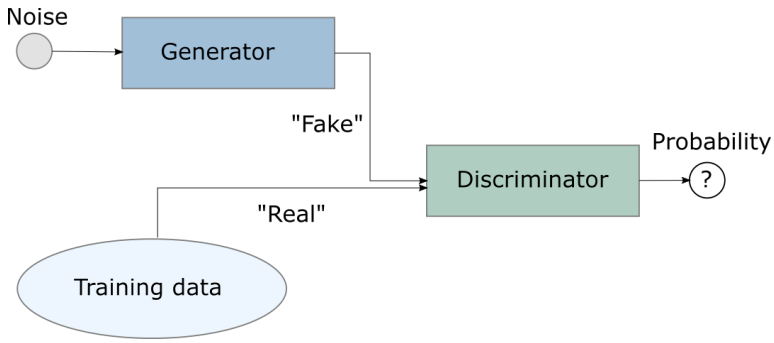


Figure 2.7: An overview of the GAN setup. The discriminator receives either a generated sample or a sample from the training data as input and outputs a probability of the sample being from the training data.

Let $\mathbf{z} \in \mathbf{Z}$ be a noise variable with prior distribution $p_{\mathbf{z}}$. Let $\mathbf{x}_{\text{data}} \in \mathbf{X}$ be the observed training data, from a distribution p_{data}. The generator $G(\mathbf{x}; \theta_g)$ performs a mapping from $\mathbf{Z}$ to $\mathbf{X}$, $G : \mathbf{z} \to \hat{\mathbf{x}} = g(\mathbf{z})$. The discriminator $D(\mathbf{x}; \theta_d)$ maps from the space $\mathbf{X}$ to a probability for the received input being from $\mathbf{x}_{\text{data}}$, $D : \mathbf{x} \to d(\mathbf{x}) \in (0, 1)$. We train D to maximize the probability of assigning the correct label to both training samples and generated samples. In the original article were both G and D represented by a MLP, with parameters θ_g and θ_d.

Let Y be an indicator variable representing whether x is from p_{data} or p_g, i.e.

$$Y = \begin{cases} 1, & \text{if } \mathbf{x} \sim p_{\text{data}}, \\ 0, & \text{if } \mathbf{x} \sim p_g. \end{cases}$$

Every guess performed by the discriminator can be viewed as a bernoulli trial with probability $p = d(\mathbf{x})$ of type $\mathbf{x}_{\text{data}}$ and probability $1 - d(\mathbf{x})$ of a generated sample. The density function for a bernoulli trial is given by

$$f(y) = p^y (1 - p)^{1-y}. \tag{2.21}$$

Thereby is the log-likelihood function given by

$$\log(L(p; y)) = \log(f(y; p)) = y \log(p) + (1 - y) \log(1 - p), \tag{2.22}$$

where the first part of Equation (2.22) has support for $Y = 1$ and the second part has support for $Y = 0$. This results in the following expected value of the log-likelihood function:

$$E[\log(L(p;y))] = E_{\mathbf{x}\sim p_{\text{data}}(\mathbf{x})}\big[\log D(\mathbf{x})\big] + E_{\mathbf{z}\sim p_{\mathbf{z}}(\mathbf{z})}\big[\log(1 - D(G(\mathbf{z})))\big]. \tag{2.23}$$

The training objective for the discriminator is to maximize Equation (2.23). It can be interpreted as maximizing the log-likelihood for the conditional probability $P(Y = y|\mathbf{x})$. Simultaneously, the generator is trained to minimize

$$\log(1 - D(g(\mathbf{z}))), \tag{2.24}$$

in other words fool the discriminator into predicting a value close to one. The loss functions of D and G are illustrated by looking at $\log(D(\mathbf{x}))$ and $\log(1 - D(\mathbf{x}))$ in Figure 2.8. Observe that $\log(1 - D(\mathbf{x}))$ is minimized when $D(\mathbf{x}) \to 1$ and maximized when $D(\mathbf{x}) \to 0$. Thereby are the loss functions of D and G dragging in each their direction for the input $\hat{\mathbf{x}} = G(\mathbf{z})$. When $\mathbf{x} \sim p_{data}$ do the discriminator want to predict a value close to 1, i.e. maximize the objective function.

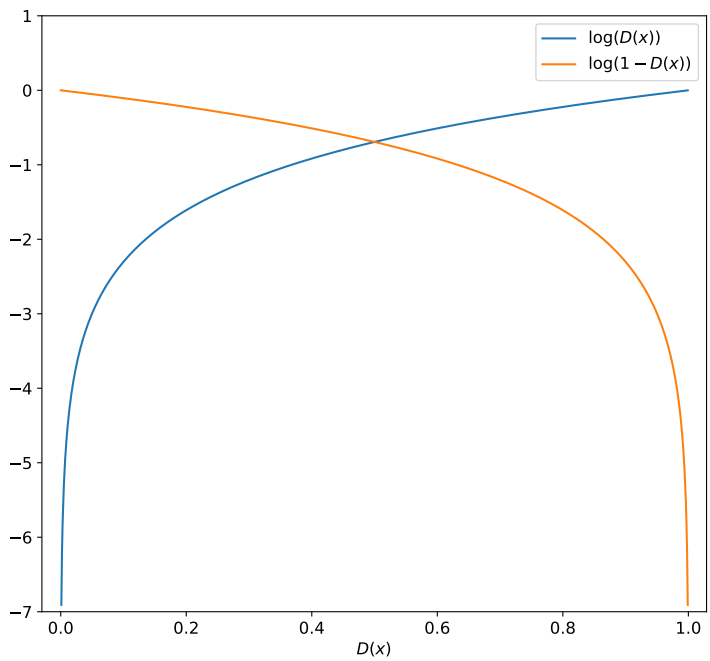


Figure 2.8: A comparison of $\log(D(\mathbf{x}))$ and $\log(1 - D(\mathbf{x}))$ for $D(\mathbf{x}) \in (0, 1)$.

This results in the following minmax two-player game with value function $V(G, D)$

from Equation (2.23):

$$\begin{aligned}&\min_G \max_D V(G,D) = \\ &\min_G \max_D E_{\mathbf{x}\sim p_{\text{data}}(\mathbf{x})}\big[\log D(\mathbf{x})\big] + E_{\mathbf{z}\sim p_{\mathbf{z}}(\mathbf{z})}\big[\log(1-D(G(\mathbf{z})))\big].\end{aligned} \tag{2.25}$$

The training procedure, as presented by Goodfellow et al. (2014) is in Algorithm 4. At each iteration are first the discriminator's weights θ_d updated while the weights of the generator are held fixed. The updates are found by ascending in the direction of the stochastic gradient of the loss (2.23). After that are the generator's weights θ_g updated while the discriminator's weights are held fixed, by descending in the direction of the loss in Equation (2.24). The theoretical justifications for the presented algorithm will be examined. (Rather than training G to minimize Equation (2.24) is G trained to maximize $\log D(G(z))$. That leads towards the same goal, as visualized in Figure 2.8, but provides stronger gradients early in the training phase.)

Algorithm 4 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

for number of training iterations **do**

  for k steps **do**

    Sample minibatch of m noise samples $\{\mathbf{z}^{(1)}, \mathbf{z}^{(2)}, \cdots, z^{(m)}\}$ from noise prior $p_g(\mathbf{z})$.

    Sample minibatch of m examples $\{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \cdots, \mathbf{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\mathbf{x})$.

    Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \Big[\log D(\mathbf{x}^{(i)}) + \log(1 - D(G(\mathbf{z}^{(i)})))\Big].$$

  end for

  Sample minibatch of m noise samples from $\{\mathbf{z}^{(1)}, \mathbf{z}^{(2)}, \cdots, \mathbf{z}^{(m)}\}$ from noise prior $p_g(\mathbf{z})$. Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log(1 - D(G(\mathbf{z}^{(i)}))).$$

end for

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

Global optimality of $p_g = p_{\text{data}}$

Proposition 1. *For G fixed, the optimal discriminator D is*

$$D_G^*(\mathbf{x}) = \frac{p_{data}(\mathbf{x})}{p_{data}(\mathbf{x}) + p_g(\mathbf{x})} \tag{2.26}$$

Proof. The training objective for the discriminator D, given any generator G, is to maximize the value function specified by Equation (2.23). This can be rewritten due to the variable substitution theorem,

$$\begin{aligned} V(G, D) &= \int_x p_{\text{data}}(\mathbf{x}) \log D(\mathbf{x}) dx + \int_z p_z(\mathbf{z}) \log(1 - D(g(\mathbf{z}))) dz \\ &= \int_x p_{\text{data}}(\mathbf{x}) \log D(\mathbf{x}) + p_g(\mathbf{x}) \log(1 - D(\mathbf{x})) dx. \end{aligned} \tag{2.27}$$

For any $(a, b) \in \mathbb{R} \setminus \{0, 0\}$ do the function

$$f(y) = a \log(y) + b \log(1 - y)$$

achieve its minimum for $y \in [0, 1]$ at $\frac{a}{a+b}$. This can be seen by differentiating with respect to y and setting equal to zero:

$$\begin{aligned} \frac{df(y)}{dy} &= \frac{a}{y} - \frac{b}{1-y} = 0 \\ &\implies a(1-y) - by = 0 \\ &\implies y = \frac{a}{a+b}. \end{aligned}$$

As $f'' < 0$ must $\frac{a}{a+b}$ be a maximum point. The discriminator D does not need to be defined outside $supp(p_{\text{data}}) \cup supp(p_g)$, and hence is the proof concluded. □

The minmax game in Equation (2.25) can be reformulated using the optimal $D^*(\mathbf{x}) = \frac{p_{\text{data}}(\mathbf{x})}{p_{\text{data}}(\mathbf{x}) + p_g(\mathbf{x})}$. Reformulating yields

$$\begin{aligned} C(G) &= \max_D (V(G, D) \\ &= E_{\mathbf{x} \sim p_{\text{data}}(\mathbf{x})} \big[\log D^*(\mathbf{x} \big] + E_{\mathbf{z} \sim p_{\mathbf{z}}(\mathbf{z})} \big[\log(1 - D^*(G(\mathbf{z}))) \big] \\ &= E_{\mathbf{x} \sim p_{\text{data}}(\mathbf{x})} \big[\log D^*(\mathbf{x} \big] + E_{\mathbf{x} \sim p_g(\mathbf{x})} \big[\log(1 - D^*(\mathbf{x})) \big] \\ &= E_{\mathbf{x} \sim p_{\text{data}}(\mathbf{x})} \big[\log \frac{p_{\text{data}}(\mathbf{x})}{p_{\text{data}}(\mathbf{x}) + p_g(\mathbf{x})} \big] + E_{\mathbf{x} \sim p_g(\mathbf{x})} \big[\log \frac{p_g(\mathbf{x})}{p_{\text{data}}(\mathbf{x}) + p_g(\mathbf{x})} \big]. \end{aligned} \tag{2.28}$$

Theorem 1. *The global minimum of the virtual training criterion $C(G)$ is achieved if and only if $p_g = p_{data}$. At that point, $C(G)$ achieves the value $-\log 4$.*

Proof. Inserting $p_g = p_{\text{data}}$ into Equation (2.30) yields

$$\begin{aligned} C(G) &= E_{\mathbf{x}\sim p_{\text{data}}(\mathbf{x})}\big[\log \frac{1}{2}\big] + E_{\mathbf{x}\sim p_g(\mathbf{x})}\big[\log \frac{1}{2}\big] \\ &= E_{\mathbf{x}\sim p_{\text{data}}(\mathbf{x})}\big[-\log 2\big] + E_{\mathbf{x}\sim p_g(\mathbf{x})}\big[-\log 2\big] = -\log 4. \end{aligned} \tag{2.29}$$

Subtracting the second line in Equation (2.29) from $C(G) = V(D^*, G)$ yields

$$\begin{aligned} C(G) &= E_{\mathbf{x}\sim p_{\text{data}}(\mathbf{x})}\big[\log \frac{p_{\text{data}}(\mathbf{x})}{p_{\text{data}}(\mathbf{x}) + p_g(\mathbf{x})}\big] + E_{\mathbf{x}\sim p_g(\mathbf{x})}\big[\log \frac{p_g(\mathbf{x})}{p_{\text{data}}(\mathbf{x}) + p_g(\mathbf{x})}\big] \\ &\quad - \big(E_{\mathbf{x}\sim p_{\text{data}}(\mathbf{x})}\big[-\log 2\big] + E_{\mathbf{x}\sim p_g(\mathbf{x})}\big[-\log 2\big] + \log 4\big) \\ &= -\log 4 + E_{x\sim p_{\text{data}}}\left[\log \frac{2\cdot\ p_{\text{data}}}{p_{\text{data}} + p_g}\right] + E_{x\sim p_{\text{data}}}\left[\log \frac{2\cdot\ p_g}{p_{\text{data}} + p_g}\right] \\ &= -\log 4 + KL\left(p_{\text{data}}\middle\|\frac{p_{\text{data}} + p_g}{2}\right) + KL\left(p_{\text{g}}\middle\|\frac{p_{\text{data}} + p_g}{2}\right). \end{aligned} \tag{2.30}$$

A brief definition of KL divergence and JS divergence is in the Appendix **??**. Observe that $KL\left(p_{\text{data}}\middle\|\frac{p_{\text{data}}+p_g}{2}\right) + KL\left(p_{\text{g}}\middle\|\frac{p_{\text{data}}+p_g}{2}\right) = 2\cdot JSD\left(p_{\text{data}}\middle\|p_g\right)$.

Jensen Shannon divergence is greater than zero unless $p_g = p_{\text{data}}$. Thereby is

$$\begin{aligned} C(G) &= -\log 4 + 2\cdot JSD\left(p_{\text{data}}\middle\|p_g\right) \\ &\geq -\log 4. \end{aligned} \tag{2.31}$$

Hence is $C^* = -\log 4$ the global minimum of $C(G)$, and the only solution is for $p_g = p_{\text{data}}$. □

In the original article, there is also a proof, that show that under certain conditions, will p_g converges to p_{data}. One condition is that G has enough capacity, and that G is updated to improve the loss function given in Equation (2.25). In practice, does this proof not fully apply, as the adversarial nets represent a limited family of distributions p_g through the function $G(\mathbf{z};\theta_g)$, and it is the parameters θ_g that are optimized rather than p_g itself.

The formulation of the loss function in Equation (2.25), suffer from vanishing gradients, and slow convergence. A range of alternative loss functions has been proposed, among others the least squares loss which will be presented in Section 2.3.7.

2.3.6 Deep Convolutional GANs

Radford et al. (2015) introduced Deep Convolutional GANs (DCGANs) to " bridge the gap between the success of CNNs for supervised learning and unsupervised learning". They propose some architectural changes from the original GAN that lead to more stable training convergence:

- Replace pooling layers by strided convolutions.
- Use batch normalization in discriminator and generator.

- Remove fully connected hidden layers for deeper architectures.
- Use ReLU activation in generator for all layers except for the output, which uses Tanh.
- Use LeakyReLU activation for all layers in the discriminator.

Using strided convolutions instead of pooling layers allows the network to learn its own spatial downsampling. Fully connected hidden layers are removed to increase convergence speed. The learning is stabilized by using batch normalization in the generator and discriminator. This was shown to make deep generators learn better, and to prevent mode collapse. Though, applying batch normalization in all layers led to oscillating samples. This was prohibited by removing the batch normalization to the output layer of the discriminator and the input layer of the generator.

The authors conclude that there is still some instability remaining in the architecture - after training for a long time, there is sometimes collapse of filters to a single oscillating mode.

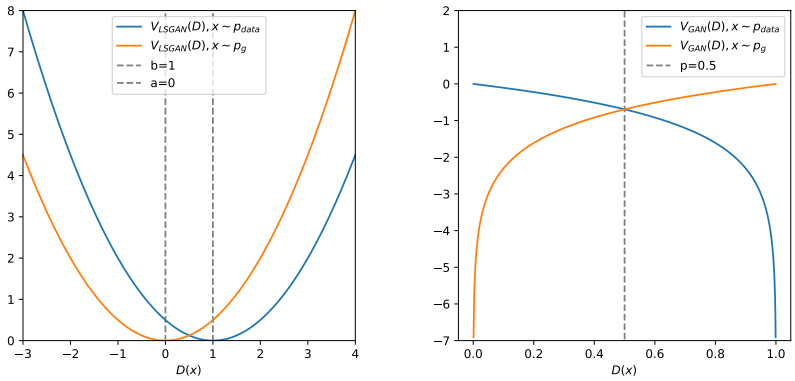
2.3.7 Least Squares GAN

Generative adversarial networks' training abilities were further improved by the introduction of the least squares loss function (LSGAN; Mao et al., 2016). The authors show that LSGANs are able to generate images of higher quality than regular GANs. In addition, LSGANs are more stable during the training process.

Minimization of the loss function used in the original GAN, Equation (2.25), is suffering from vanishing gradients. By introducing the least squares loss, samples are penalized based on their distance to the decision boundary. This is in contrast to the old loss function, which does not have any gradient contribution from the samples on the right side of the decision boundary. In this setup is D no longer outputting a probability of the sample being real. Instead, D wants to learn directly to predict the correct labels: Let a and b denote the labels for fake and real data. The objective function of the LSGAN is defined as follows:

$$\begin{aligned} \min_D V_{\text{LSGAN}}(D) &= \frac{1}{2} E_{\mathbf{x}\sim p_{\text{data}}(\mathbf{x})}\left[(D(\mathbf{x}) - b)^2\right] + \frac{1}{2} E_{\mathbf{z}\sim p_z(\mathbf{z})}\left[(D(G(\mathbf{z})) - a)^2\right] \\ \min_G V_{\text{LSGAN}}(G) &= \frac{1}{2} E_{\mathbf{z}\sim p_z(\mathbf{z})}\left[(D(G(\mathbf{z})) - c)^2\right], \end{aligned} \tag{2.32}$$

where c denotes the label that the generator wants the discriminator to guess for fake data. It is shown that minimizing the objective function Equation (2.32), yields a minimization of the Pearson χ^2 divergence when $b - c = 1$ and $b - a = 2$. In practice, the results with a binary coding scheme $a = 0, b = 1, c = 1$ are showing similar performance. A comparison of D's loss function in a traditional GAN and in a LSGAN with binary coding is shown in Figure 2.9. Observe that the slope is saturating close to the correct probability prediction for the traditional loss. In comparison, LSGAN's loss is only flat when D makes the exact correct prediction, and the slope is higher nearby, also on the right side of the decision boundary. Regular GANs have a very small loss for samples that are on the right side of the decision boundary.



(a) Loss function LSGAN

(b) Loss function GAN

Figure 2.9: The discriminator's loss for LSGAN leads D to predict values close to $b = 1$ for real samples $\mathbf{x} \sim p_{data}$ and values close to $a = 0$ for generated samples $\mathbf{x} \sim p_g$. In the general GAN, the discriminator aims to maximize its loss function, thereby predict probabilities close to 1 for training samples and probabilities close to 0 for generated samples.

2.3.8 Conditional Generative Adversarial Networks

The conditional version of generative adversarial nets was introduced by Mirza and Osindero (2014). In an unconditioned generative model, the goal is to generate samples from the training data's distribution p_{data}. There is no way of controlling which part of p_{data} the generated sample belongs to. Introducing a conditional variable $\mathbf{x}$, is a way of being able to direct the data generation process. Examples of different conditional variables are class labels (Mirza and Osindero, 2014), images (Isola et al., 2016) and audio (Pascual et al., 2017).

The conditional variable is given as input both to the generator and the discriminator: $G : \mathbf{x}, \mathbf{z} \rightarrow \mathbf{y}$ and $D : \mathbf{x}, \mathbf{y} \rightarrow (0, 1)$. The value function from Equation (2.25) can be rewritten as

$$V(G, D) = E_{\mathbf{x},\mathbf{y}}\big[\log D(\mathbf{x}, \mathbf{y})\big] + E_{\mathbf{x},\mathbf{z}}\big[\log(1 - D(\mathbf{x}, G(\mathbf{x}, \mathbf{z})))\big]. \tag{2.33}$$

Previous approaches found it useful to combine the value function with more traditional losses, like MSE or MAE (Isola et al., 2016).

2.4 Generative Adversarial Networks for Speech Enhancement

The first end-to-end Speech Enhancement GAN (SEGAN) was implemented by Pascual et al. (2017). The method has two interesting properties. Firstly, as the model is working

end-to-end, it avoids making any assumptions regarding the raw data. Secondly, the GAN learns a loss function from the data, and hence avoiding to optimize according to a loss function that is not necessarily deduced from the aim of enhancement. The idea is that D learns a loss for G's enhancement to seem accurate.

Both discriminator and generator are conditioned on the noisy speech. The generator is doing the enhancement mapping from latent noise and noisy speech to enhanced speech, while the discriminator takes input pairs of corresponding noisy and clean speech or noisy and enhanced speech and returns a probability of the input pair being from the training data. The architecture is inspired by the DCGAN framework (Section 2.3.6) and the LSGAN loss function (Section 2.3.7) is used.

Model design

Let us use the syntax presented by Pascual et al. (2017). Given a noisy signal $\tilde{\mathbf{x}}$, we want to teach the generator G a mapping from noisy to clean, i.e., we want G to enhance the signal: $G : (\mathbf{z}, \tilde{\mathbf{x}}) \rightarrow \hat{\mathbf{x}}$. The discriminator receives input pairs of either noisy and enhanced speech $(\tilde{\mathbf{x}}, \hat{\mathbf{x}})$ or noisy and clean speech $(\tilde{\mathbf{x}}, \mathbf{x})$. The system is shown in Figure 2.10. We will go further into the specific architecture used to make G and D.

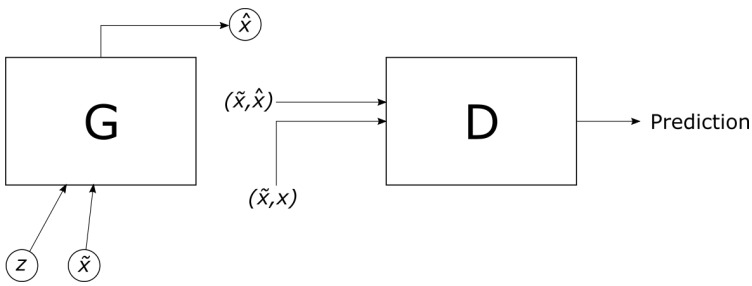


Figure 2.10: The generator G receives latent noise and noisy speech as input and outputs enhanced speech $\hat{\mathbf{x}}$. The discriminator D receives either a real or fake pair, that is either $(\tilde{\mathbf{x}}, \mathbf{x})$ or $\tilde{\mathbf{x}}, \hat{\mathbf{x}})$, and gives out a prediction of the received input, as described in Section 2.3.7.

The G network is fully convolutional, with no dense layers. The input has a window length $L = 16384$ samples, which is slightly more than one second of speech[1] The architecture is illustrated in Figure 2.11. It has an encoder-decoder structure. In the encoder stage there is used one dimensional strided convolutional layers with filter width 31 and stride 2. The resulting layers are of dimensions $16384 \times 1, 8192 \times 16, 4096 \times 32, 2048 \times 32, 1024 \times 64, 512 \times 64, 256 \times 128, 128 \times 128, 64 \times 256, 32 \times 256, 16 \times 512$, and 8×1024. The convolutional layers are followed by parametric rectified linear units (PReLUs; Section 2.3.2). Decimation is done until we have a condensed representation $\mathbf{c}$ of dimension 8×1024. There, the latent noise z is sampled from a standard normal distribution $\mathcal{N}(0, \mathbf{I})$ of dimension 8×1024. The condensed representation $\mathbf{c}$ and the latent noise $\mathbf{z}$ are concatenated before the decoding stage. The decoding stage is a mirrored version of the encoding stage. The only difference is due to the concatenated $\mathbf{z}$, which lead to a doubling of the number of feature maps in every layer.

[1] All speech files are downsampled to 16 kHz before enhancement.

In order to pass fine-grained information of the wave-form do the network use skip-connections between each encoding layer and its corresponding decoding layer. This can also lead to better training performance, as it allows the gradients to flow deeper through the whole network (Ronneberger et al., 2015).

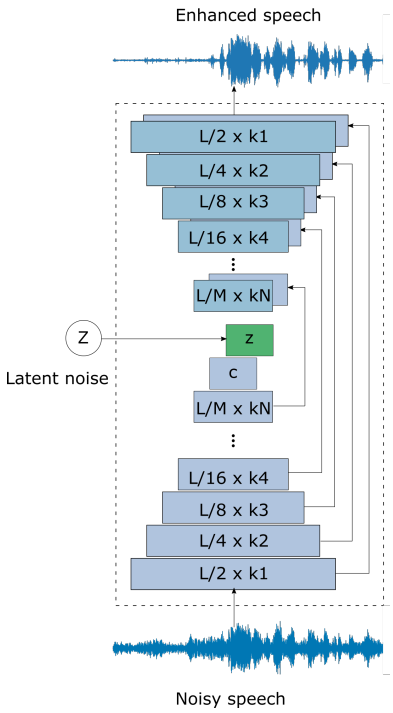


Figure 2.11: The generator is formed as an encoder-decoder. The noisy input of width $L = 16384$ gets downsampled by strided convolutional layer till a condensed representation $\mathbf{c}$, which is concatenated with random noise $\mathbf{z}$. The upsampling layers are a mirrored version of the downsampling layers, with skip connections between corresponding layers. The illustration is inspired by Figure 2 in Pascual et al. (2017).

The D network follows the same convolutional structure as G's encoder stage, but there are some differences. It receives two input channels instead of one, as it gets 16384 samples of noisy and clean or enhanced speech. Virtual batch normalization is used before LeakyReLu non-linearities (Section 2.3.2) with parameter $\alpha = 0.3$. In the last activation layer, there is a one-dimensional convolution layer with one filter of width one that does not downsample the hidden activations. $(1 \times 1$ convolution), which reduces the number of required parameter in the fully connected component from $8 \times 1024 = 8192$ to 8.

Loss function

The loss function chosen is the least-squares GAN described in Equation (2.32), modified for the conditional setting. An L_1 regularization term is added to the generator's loss,

in order to minimize the distance between its generations and the clean samples. The resulting loss functions are

$$\min_{D} V_{\text{LSGAN}}(D) = \frac{1}{2} E_{\mathbf{x},\tilde{\mathbf{x}}}\left[(D(\mathbf{x},\tilde{\mathbf{x}}) - 1)^2\right] + \frac{1}{2} E_{\mathbf{x},\mathbf{z}}\left[D(G(\mathbf{z},\mathbf{x}),\mathbf{x})^2\right] \tag{2.34}$$

$$\min_{G} V_{\text{LSGAN}}(G) = \frac{1}{2} E_{\mathbf{x},\mathbf{z}}\left[(D(G(\mathbf{z},\mathbf{x}),\mathbf{x}) - 1)^2\right] + \lambda \left\|G(\mathbf{z},\mathbf{x}) - \mathbf{x}\right\|_1, \tag{2.35}$$

where the hyperparameter λ were set to 100 during experiments.

The system is trained iteratively like a regular GAN, such that G's weights θ_g are fixed when D's weights θ_d are updated through back-propagation, and θ_g is fixed when θ_d is updated.

Chapter 3

Methods

The project is developed in Python with use of the API Keras (Chollet et al., 2015) with TensorFlow backend (Abadi et al., 2015). The implementation is strongly inspired by SEGAN (Pascual et al., 2017). The full code can be found at `https://github.com/miralv/Speech-Enhancement-with-a-Generative-Adversarial-Network`. NTNU's cluster Idun has been used to train the model on a GPU. We will first go through the datasets that have been used before we go further into the experimental details.

3.1 Dataset

3.1.1 Speech signals

The clean audio files are generated from the Norwegian speech database NB Tale (Nasjonalbiblioteket, 2016), produced by Lingit AS in cooperation with NTNU. The speech database contains recordings of 380 different speakers, from 24 different dialect areas. Each speaker has read 20 sentences, where each sentence was specifically chosen to include different acoustic sounds. The recordings were done with 48 kHz frequency and 16 bit resolution[1]. There have been used two microphones during the recordings, but we will only use one of the microphone recordings due to this project being focused on monaural speech separation. We have used the recordings form Module 1, which contains 4800 recorded sentences read by 240 speakers. The data set is divided into 12 groups, where each group correspond to a geographic area of Norway.

3.1.2 Noise signals

The noise files are generated from a noise database with 100 different environmental sounds (Hu, 2014) and the Demand noise database (Thiemann et al., 2013). The noise

[1] A subset of the recordings have mistakenly been recorded with wrong frequency (44.1 kHz). These have been upsampled to 48 kHz. A log of which files this regards can be found together with the audio records (Nasjonalbiblioteket, 2016)

signals collected by Hu were sampled at 20 kHz with a 16 bit resolution. The files are labeled after which type of noise they contain. Examples are crowd noise, machine noise, traffic, wind, and water noise.

The Demand database contains a collection of recordings of acoustic noise in diverse environments. Also here are the recordings done with several microphones, but we will only use the first channel, labeled "ch01" in the database. The recordings used are sampled at 16 kHz with 16 bit resolution.

3.1.3 Training, validation and test set

The noise files and speech files have been partitioned into a training set, a validation set, and a test set. All speech signals from group 1 - 11 are put in the training set, together with 101 noise signals. 95 are from the 100 environmental sounds, and 6 are from the Demand database. The speech and noise files are combined randomly at SNR $0, 10$ and 15 dB during training.

The validation set contains two sentences from two speakers from group 12, one male and one female. They are combined with two noise signals from the 100 environmental sounds; `n46:Traffic` and `n77:Wind`, and two noise signals from the Demand database: `Station` and `Bus`. The noisy validation files are constructed at the unseen SNR 5 dB, in addition to the levels $0, 10$ and 15 dB.

The test set uses two other unseen speakers from group 12, one male and one female, where both speakers have five sentences. The noise files `Traffic`, `Cafeteria`, `n78:Wind`, `n68:Water` and `n28:Machine` are used. The noisy files in the test set are constructed at the same levels of SNR as the validation set. Figure 3.1 shows the spectrograms of the noise files used for test and validation.

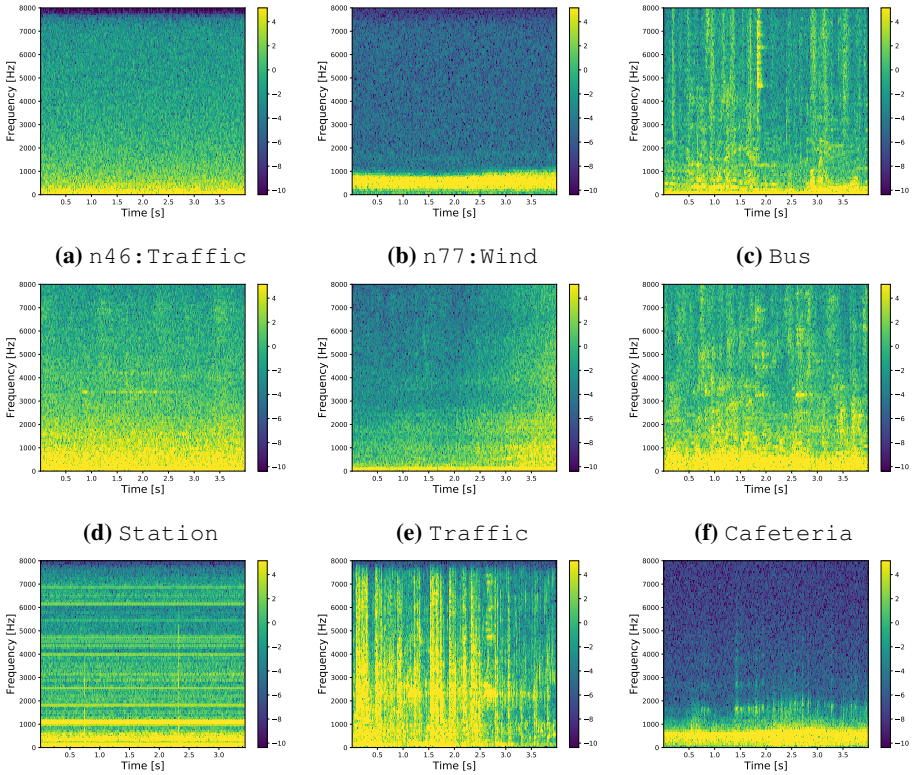
3.2 GAN setup

We have two different versions of the setup: with and without latent noise. The difference is visualized in Figure 3.2. Most other architectural detail is like explained in Section 2.4. The optimizers RMSProp and Adam were both tested during preliminary runs and showed similar results. Hence, we have followed the choice made by the authors of SEGAN and used RMSProp.

G and D are created and compiled separately. Before the GAN is constructed are D's weight fixed, such that D's weights are unchanged when the GAN is trained. The GAN with latent noise is constructed by creating a model that takes a pair of noisy and enhanced or clean speech as input, together with latent noise $\mathbf{z}$, and outputs the predicted label from D and the enhanced speech from G.

```
GAN = Model(inputs=[clean_audio_in, noisy_audio_in, z],
            outputs=[D_out, G_out])
```

Similarly is the GAN without latent nosie constructed by creating a model graph without latent noise:

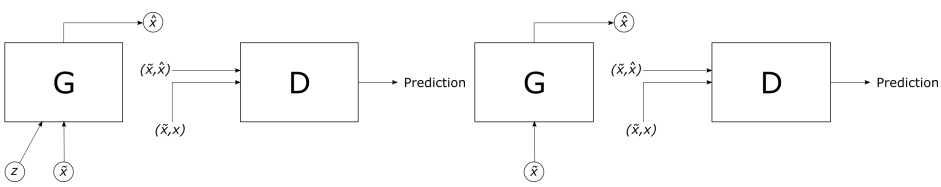


(a) `n46:Traffic` **(b)** `n77:Wind` **(c)** `Bus`

(d) `Station` **(e)** `Traffic` **(f)** `Cafeteria`

(g) `n28:Machine` **(h)** `n68:Water` **(i)** `n78:Wind`

Figure 3.1: Log-density spectrograms of the noise files used to validate and test the GAN. The noises that are labeled with a number are from the 100 environmental sounds (Hu, 2014), while the others are from the Demand database (Thiemann et al., 2013). All files have been scaled to have RMS of 1000 before visualization.



(a) SEGAN with latent noise **z**. **(b)** SEGAN without latent noise **z**.

Figure 3.2: Two different versions of SEGAN have been implemented, one version which uses latent noise like presented in the original algorithm (3.2a), and one version that omits the latent noise (3.2b).

```
GAN = Model(inputs=[clean_audio_in, noisy_audio_in],
            outputs=[D_out, G_out]).
```

The summaries from the compiled GAN models can be found in Appendix B.1. The model with latent noise has a total of $98,498,155$ parameters where only $74,120,049$ of them are trainable (θ_g). D's parameters θ_d are marked as untrainable when `GAN.train_on_batch()` is called, such that only θ_g is updated. D is trained by calling `D.train_on_batch()` separately.

The removal of $\mathbf{z}$ results in a reduction of parameters: the parameters in G is reduced to $57,867,121$ parameters. The discriminator's number of parameters is unchanged.

For additional details, see the implementaion on Github.

3.3 Training procedure

3.3.1 Overview

A float diagram of the training procedure is presented in Figure 3.3. The inner loop presented in the diagram is run for 10 epochs, with 40 batches per epoch and batch size 200. The system uses a window length of 16384 samples, which is slightly more than one second. Thereby is the algorithm trained on $10 \times 40 \times 200 \times 16384/16000 = 81920$ seconds of noisy speech, which is $22,76$ hours. After every epoch is the current version of the generator used to enhance the validation set. Lastly, the noisy test set is enhanced by the trained generator.

Speech and noise files are drawn randomly from the training set before they are preprocessed and added at wanted SNR. Thereafter is G mapping from noisy to enhanced speech, with or without latent noise. D receives one batch of fake pairs (enhanced, clean) and one batch of real pairs (noisy, clean) and predicts labels. Thereafter are D's weights updated based on D's loss, and G's weights updated based on G's loss by use of back-propagation. The different parts of the training procedure will be elaborated in the following.

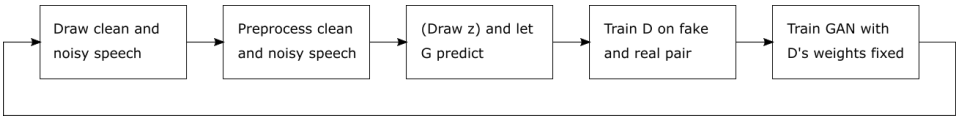


Figure 3.3: An overview of the different steps performed in the inner loop during the training procedure.

3.3.2 Random generation of speech and noise

The speech and noise files are drawn randomly during training. All speech paths and all noise paths in the training set have been gathered in the variables `speech_paths` and `noise_paths`. For each generation of a batch size of speech and noise files, is a random choice of speech paths and noise paths chosen with replacement with the function `random.choice` from `NumPy` (Oliphant, 2006). The procedure is presented in Algorithm 5.

Algorithm 5 Random generation of a batch of speech and noise files.

Require: batch_size: number of elements in each batch
Require: speech_paths: the paths to all speech files in the training set
Require: noise_paths: the paths to all noise files in the training set
  for number of batches **do**
    speech_batch ← a random choice of a batch size of speech paths from speech_paths
    noise_batch ← a random choice of a batch size of noise paths from noise_paths
  end for

3.3.3 Preprocessing

The speech and noise file are both downsampled to 16 kHz by use of the function `resampy.resample` (McFee, 2016) before they are scaled to have values in the range [-1,1]. After that is a random part of one window length drawn randomly from both speech and noise file. The noisy speech is constructed by adding the windows at an SNR level drawn randomly from the SNRs used during training, $0, 10$ and 15 dB. The SNR factor was calculated on the full speech and noise signal before preprocessing as specified in Equation (2.6). Lastly, a high-frequency preemphasis filter is applied, like done by Pascual et al. (2017). A schematic overview of the process is presented in Figure 3.4.

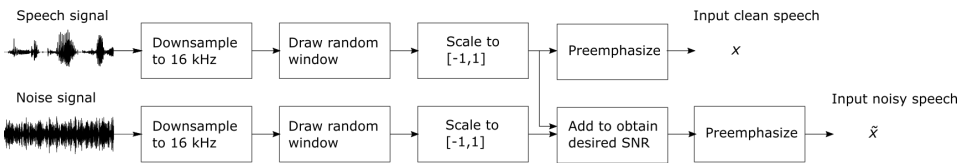


Figure 3.4: A schematic overview of the preprocessing steps done before the clean speech $\mathbf{x}$ and noisy speech $\tilde{\mathbf{x}}$ is given as input to the GAN.

3.3.4 Training the GAN

The original speech enhancement GAN is presented in Figure 2.10. G and D are trained iteratively like explained in Algorithm 4, but the algorithm is modified for the conditional setting. First is a batch of (noisy, clean) pairs ($\tilde{\mathbf{x}}$,$\mathbf{x}$) drawn randomly and preprocessed as described in Section 3.3.3. Thereafter, a batch of the latent noise $\mathbf{z}$ of dimension (8×1024) is drawn from a standard normal distribution $N(0, I)$ if the setup is with $\mathbf{z}$. G performs the mapping from $\mathbf{z}$ and $\tilde{\mathbf{x}}$ or only $\tilde{\mathbf{x}}$ to the enhanced speech signal $\hat{\mathbf{x}}$. The discriminator D is trained on one batch of real pairs $(\tilde{\mathbf{x}}, \mathbf{x})$ and one batch of fake pairs $(\tilde{\mathbf{x}}, \hat{\mathbf{x}})$ at each iteration. I.e. the weights θ_d are updated through back-propagation, where the loss function is as stated in Equation (2.34). After that, the generator is trained such that only θ_g is updated, by using the loss function from Equation (2.35). Note that G's weights are fixed when D is trained and oppositely. The training loss, validation loss, and the enhanced validation set are stored after each epoch.

3.4 Testing procedure

When the GAN has been trained, the generator G has (hopefully) learned to enhance noisy speech. The objective measures PESQ and STOI are used to measure the quality and intelligibility of the enhanced signals. The test float is further elaborated below.

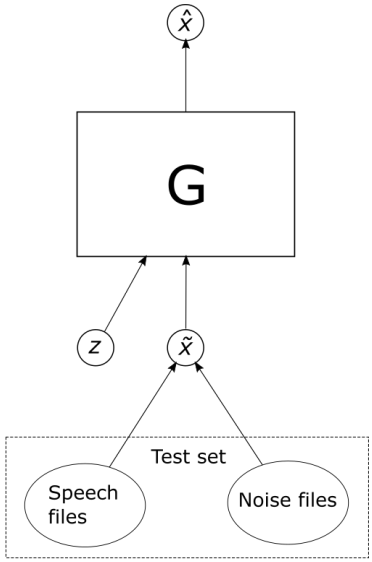


Figure 3.5: When the GAN has been trained, the generator G can be used to enhance speech. G maps from noisy speech $\tilde{\mathbf{x}}$ as input along with random generated latent noise $\mathbf{z}$, and performs the learned mapping to enhanced speech $\hat{\mathbf{x}}$. The setup without latent noise is equal, just without $\mathbf{z}$.

The noisy signal is constructed like in the training phase, the only difference is that instead of drawing windows randomly, they are taken successively from the noisy signal. The generator enhances the signal window by window, and the outputs are later concatenated. The concatenated enhanced windows are further postprocessed by inverting the steps performed in the preprocessing phase, i.e., the signals are deemphasized, before they are upscaled to `int16`. We let the enhanced files remain in 16 kHz.

One could have enhanced noisy recordings directly. The advantage of using a constructed combination of speech and noise to construct the noisy input, is that the corresponding clean version is known. The noisy and enhanced version of a file is compared with the clean reference version when the quality is assessed through PESQ and STOI.

3.4.1 Experiments

Because there are several elements in the model that include randomness (SGD, random generation of noise, random initialization of weights), the results may differ between sessions with completely equal setup. Due to high computational requirements is only one configuration of hyperparameters tested. The model will therefore be trained 3 times with

latent noise and 3 times without latent noise, before their results are compared. Motivated by their results, one shorter run where the number of batches per epoch is reduced from 40 to 10 was run for both setups. The shorter run's enhancement results are mainly in Appendix B.4.

Chapter 4

Results

We will go through the different runs' progress during training, by examining the PESQ and STOI scores of the enhanced validation set. We will also look at the training and validation loss of the loss functions used. The test set is enhanced with each of the trained models, and the enhancement results are evaluated objectively by PESQ and STOI. The models' ability to enhance the different noise types will be examined by looking at statistics for the different noise signals separately. The PESQ and STOI results are compared with the scores obtained by the SEGAN article of inspiration and a different GAN setup, where the enhancement is done by enhancing spectrograms. Lastly, we have provided some dropbox links to enhanced signals, with some minor subjective analysis.

4.1 Training progress

The validation set has been enhanced after each epoch, with use of the model G at that time. I.e. there is the differences in G that lead to the differences in the enhanced output from epoch to epoch.

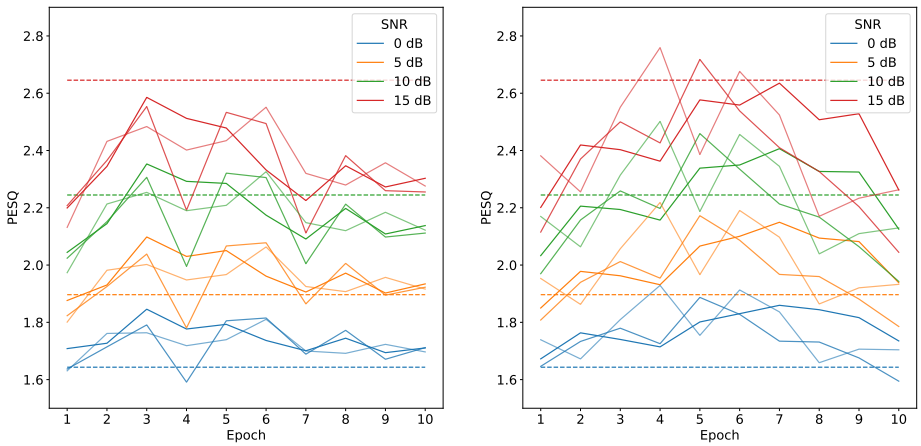
4.1.1 PESQ

In Figure 4.1, the enhancement results for the validation set for each of the three runs with and without latent noise are plotted. The stippled lines represent the PESQ score of the noisy validation of given SNR level before enhancement. The three different runs are distinguished by different nyances of each color. The blue lines are from signals with SNR 0 dB, the yellow 5 dB, the green 10, dB and the red 15 dB. The different runs are plotted separately in Appendix B.3.

We can start by exploring the PESQ scores from the setup with latent noise $\mathbf{z}$. At SNR 0 dB, the majority of the enhanced scores are higher than the reference score. There is an increasing trend for the first 3 epochs. One of the runs gets a drop at epoch 4, before the trend is decreasing from epoch $5-6$. The different runs converge towards the score of 1.7. At SNR 5 dB, the PESQ scores start under the reference line. The different runs

converge towards a value of 1.9, slightly above the reference score. The scores at SNR 10 dB, are mostly under the reference line, and the values converged to the point 2.1 for epoch 10, which is visibly below the noisy reference line. At SNR 15 dB are none of the runs achieving PESQ scores over the reference line.

The three realizations without PESQ are displayed in Figure 4.1b. Compared to the PESQ scores with $\mathbf{z}$, the maximum values for each SNR are higher, achieving some scores over the reference line also for SNR 15 dB. All levels of SNR start with an increase until epoch $2 - 4$, before there eventually is a decreasing trend (for two of three runs) the last epochs. The end score is more diverse for the different runs, compared to the model with $\mathbf{z}$.



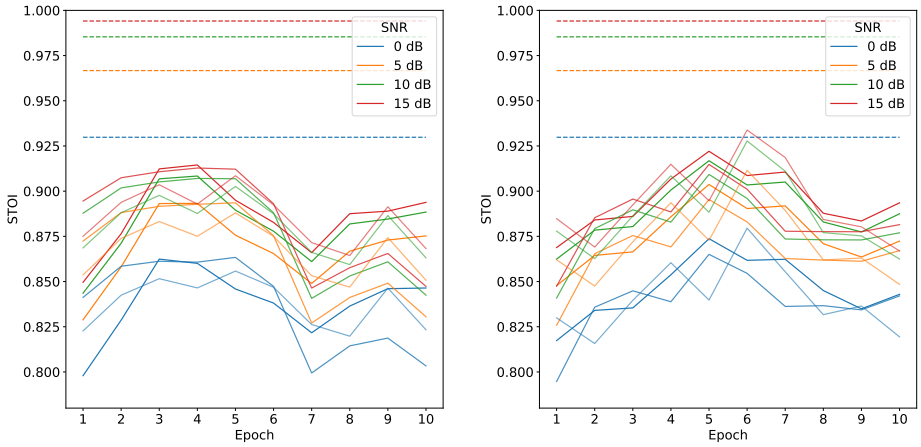
(a) With $\mathbf{z}$ **(b)** Without $\mathbf{z}$

Figure 4.1: The validation set have been enhanced after each epoch, with the then version of the generator. The PESQ results are displayed here for epoch 1 - 10 for the tree runs with and without latent noise. The dashed lines are the PESQ scores obtained by the noisy validation set before enhancement.

The highest PESQ scores with $\mathbf{z}$ are achieved at epoch 3, run 1, while the highest scores without $\mathbf{z}$ are achieved at epoch 4, run 3. Some of these enhanced files are provided on Dropbox, and briefly commented in Section 4.2.2.

4.1.2 STOI

None of the runs lead to an improvement in terms of STOI, at any point, as illustrated in Figure 4.2. Observe that the noisy reference lines are above the corresponding lines of the same color at all points. There is a smaller difference between different SNR-levels here, and the curves are intersecting for SNR $0, 5$ and 10.



(a) With **z** **(b)** Without **z**

Figure 4.2: The validation set have been enhanced after each epoch, with the current version of the generator. The STOI results are displayed here for epoch 1 - 10 for the tree runs with and without latent noise. The dashed lines represent the STOI scores obtained by the noisy validation set before enhancement.

4.1.3 Training and validation loss

The training and validation losses of G and D during the three training runs are displayed in Figure 4.3 for the setup with latent noise, and in Figure 4.4 for the setup without latent noise. G's error is dominated by the term from the discriminator's loss in Equation (2.35).

In the first run with **z**, the discriminator's training loss is decreasing rapidly from epoch 2 to 4, and thereafter stabilizing at a low level. The validation loss is stable at a higher level. D's training loss is decreasing until epoch 5 before it flattens out.

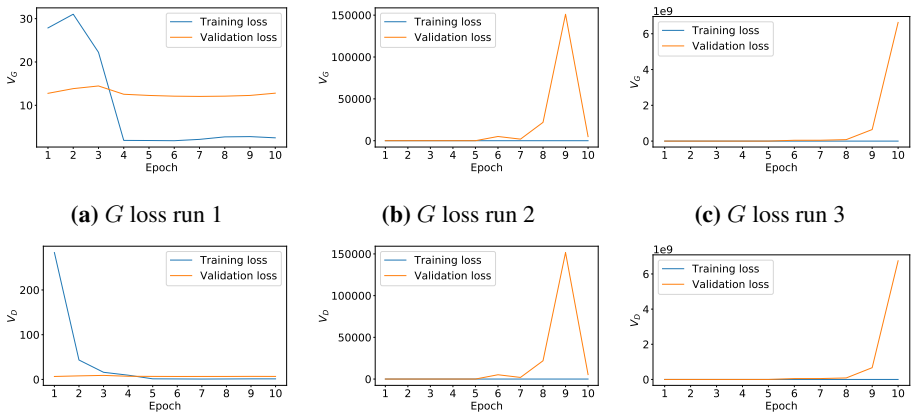
In run 2 and 3 are the scale a lot higher, due to very high loss contributions from the discriminant - the values predicted are far away from the given labels 0 and 1. In run 2, the values are decreasing after the sudden increase, while run 3 ends at a maximum point.

The runs without **z** have some peaks in the first run, a very high maximum loss in the second run, where the value decrease in the following epoch and the third run ends at a maximum point also here.

4.2 Enhancement results

4.2.1 Objective evaluation

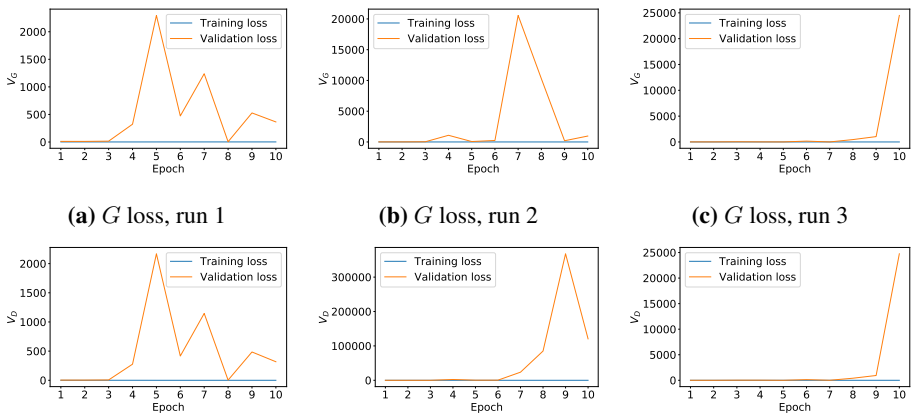
The trained models' ability of enhancement of diverse speakers and noise sources will be evaluated by use of the objective measures PESQ and STOI. The first run's PESQ results are additionally visualized in histograms together with the PESQ scores of the noisy test set before enhancement. (The three runs gave similar results, we are therefore only showing



(a) G loss run 1 **(b)** G loss run 2 **(c)** G loss run 3

(d) D loss run 1 **(e)** D loss run 2 **(f)** D loss run 3

Figure 4.3: Training and validation loss for G and D are plotted for the three runs with latent noise $\mathbf{z}$.



(a) G loss, run 1 **(b)** G loss, run 2 **(c)** G loss, run 3

(d) D loss, run 1 **(e)** D loss, run 2 **(f)** D loss, run 3

Figure 4.4: Training and validation loss for G and D are plotted for the three runs without latent noise $\mathbf{z}$.

the results for the first run for both methods).

Enhancement with latent noise

The PESQ scores of the noisy test set and the enhanced test set after run 1 have been visualized in a histogram in Figure 4.5. Every bar has a width of 0.1, and the height is the relative frequency. The noisy bars are colored red, while the enhanced scores are displayed in blue. There is a visible shift to the right from the noisy to the enhanced files. The noisy

files have an average score of 1.32, while the enhanced files have an average score of 1.53.

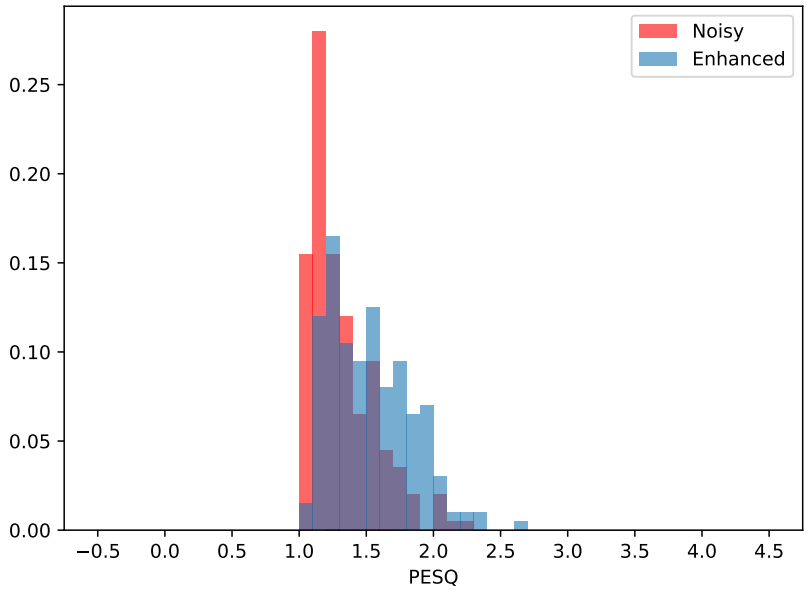
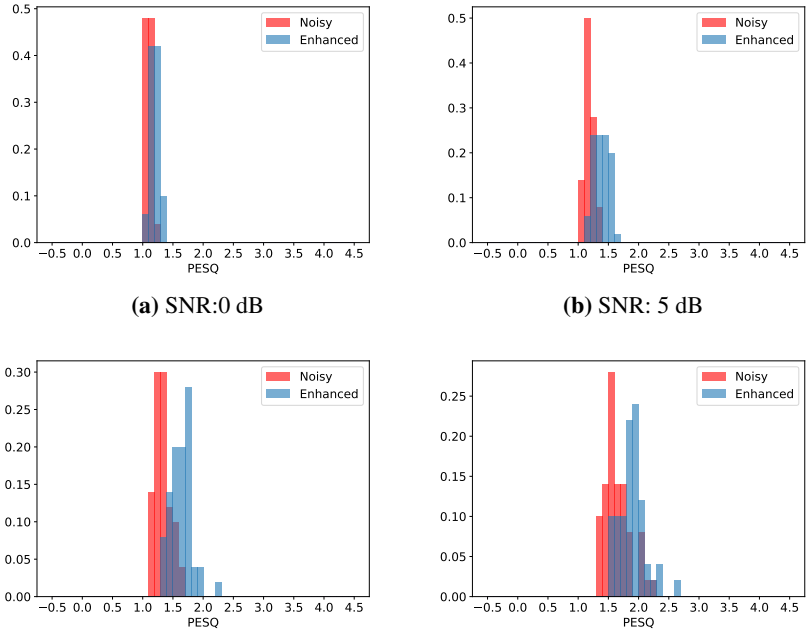


Figure 4.5: The PESQ score of the noisy test set and enhanced test set is displayed in a relative frequency histogram. The setup with latent noise **z** was used.

The PESQ scores of the test set and the enhanced test set distributed after SNR is visualized in Figure 4.6. The corresponding mean scores can be found in Table 4.1, Run 1. All SNRs have a shift towards higher values from noisy to enhanced, but the difference is most visible for SNR 10 and 15. For SNR 0 dB, there is a small shift to the right from noisy to enhanced. The mean value of the noisy files with SNR 0 is 1.11 while it is increased to 1.22 for the enhanced set. For SNR 5 dB is there a more clear shift to the right for the enhanced test set, and the mean score increases from 1.18 to 1.38. There is a clear difference between the noisy and enhanced set for SNR 10 dB too. The mean score increases from 1.34 to 1.62. For SNR 15 dB, the increase from noisy to enhanced is from 1.64 to 1.85.

In Table 4.4 are the STOI scores of noisy and enhanced test set shown. The noisy STOI score is higher than the average obtained for each of the runs at all levels of SNR. At SNR 0 dB, the enhanced set with the generator from the first run is only 0.1 below, while the distance increases to 0.3 for SNR 5 dB, 0.4 for SNR 10 dB and 0.6 for SNR 15 dB. The average noisy score is 0.90, while the highest average score for the enhanced test set is 0.87.



(a) SNR:0 dB

(b) SNR: 5 dB

(c) SNR: 10 dB

(d) SNR: 15 dB

Figure 4.6: The PESQ score of the noisy test set and the enhanced test set with latent noise **z** is visualized in relative frequency histograms separated according to SNR.

Enhancement without latent noise

The histograms resulting from runs without latent noise are very similar to the histograms obtained with latent noise, as shown in Figure 4.8 and Table 4.3. The noisy files have an average score of 1.32, while the enhanced files have an average score of 1.50, which is 0.3 lower than the average score with latent noise. The average scores for all three runs for with and without **z** are shown in Table 4.1 and Table 4.3, respectively. The runs with **z** have a higher average for all levels of SNR, though the difference is only 0.2 for SNR 0 and 5 dB. The difference is 0.6 for SNR 10 dB. For SNR 15 dB there is a clear difference, of 0.12.

Noise statistics

How do the models perform on the different types of noise tested on? The results from the setup with latent noise, organized by type of noise are displayed in Table 4.5 (PESQ) and Table 4.6 (STOI). In terms of PESQ, there are improvements from noisy to enhanced speech for all types of noise, at all the levels of SNR. At 0 and 5 dB is the largest increase from noisy to enhanced for cafeteria noise, where the increases are 0.15 and 0.29. At SNR 10 dB is the largest increase in PESQ for machine noise, from 1.32 to 1.68, that is an increase of 0.36. The largest increase at SNR 15 dB is also for machine noise, from 1.58

Table 4.1: PESQ scores for different levels of SNR is calculated for the noisy test set and the enhanced test set, where the set has been enhanced by the resulting G after three runs of the training period with equal parameters. The setup with latent noise $\mathbf{z}$ was used for training and testing.

	0 dB	**5 dB**	**10 dB**	**15 dB**	**Mean**
Noisy	1.11	1.18	1.34	1.64	1.32
Run 1	1.21	1.38	1.63	1.90	1.53
Run 2	1.22	1.39	1.61	1.80	1.51
Run 3	1.21	1.37	1.61	1.85	1.51
Average	1.22	1.38	1.62	1.85	1.52

Table 4.2: Average STOI scores for different levels of SNR is calculated for the noisy test set and the enhanced test set, where the set has been enhanced by the resulting G after three runs of the training period with equal parameters. The setup with latent noise $\mathbf{z}$ was used for training and testing.

	0 dB	**5 dB**	**10 dB**	**15 dB**	**Mean**
Noisy	0.81	0.89	0.94	0.97	0.90
Run 1	0.80	0.86	0.90	0.91	0.87
Run 2	0.77	0.82	0.85	0.87	0.83
Run 3	0.78	0.83	0.87	0.89	0.84
Average	0.78	0.84	0.87	0.89	0.85

to 1.94, 0.36.

In terms of STOI, there are mostly a decrease from noisy to enhanced. At SNR 0 dB there is one improvement, which is for water noise, increasing the STOI score from 0.70 to 0.73. For traffic noise, the score is unchanged, while for cafeteria and machine noise, the results are only decreasing by 0.1. Water noise gets an improved STOI score also for SNR 5 dB, where the score increases from 0.79 to 0.80. For the rest of the comparisons, there is a decrease between 0.02 and 0.06.

The noise statistics for the setup without latent noise $\mathbf{z}$ is displayed in Table 4.5 (PESQ) and Table 4.8(STOI). The scores are compared with the corresponding scores with latent nosise. The PESQ scores at 0 dB are equal for machine and wind noise, and slightly higher for the other types of noise. At 5 dB SNR, the score is still equal for wind, while there is a small increase in the other values. The trend shifts at SNR 10 dB. The scores without $\mathbf{z}$ are lower for all types of noise except water noise. At the highest value of SNR, the enhanced PESQ is lower than for the other model for all the different noise types. The STOI differences between the two model setups are minor, with the largest differences being on 0.02.

Results in context

Michelsanti and Tan (2017) produced a noise specific table where their conditional spectrogram based GAN method was compared against baseline methods. They obtained en-

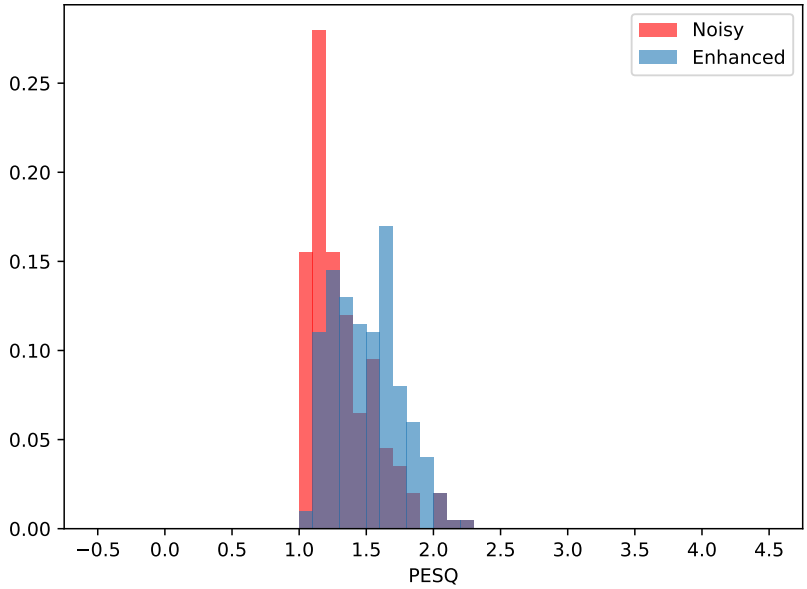
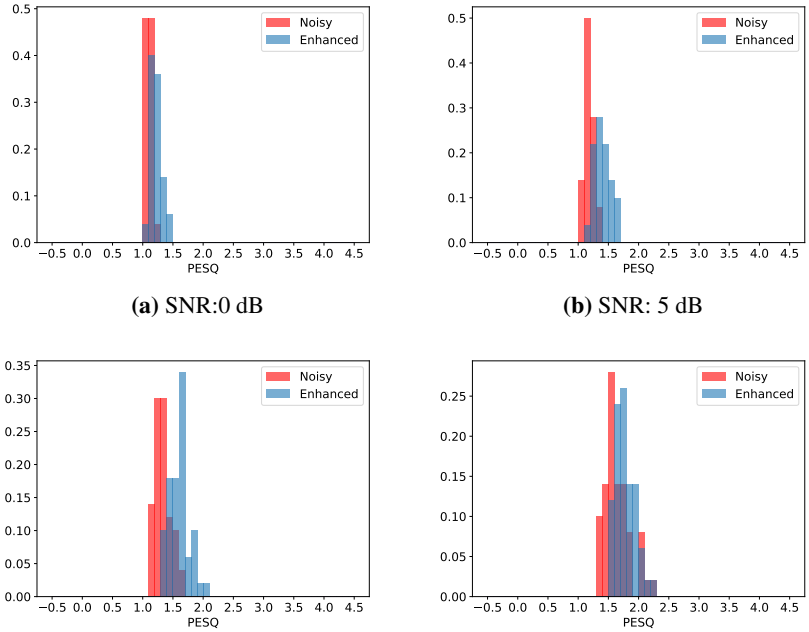


Figure 4.7: The PESQ score of the noisy test set and enhanced test set is displayed in a relative frequency histogram. The setup without latent noise $\mathbf{z}$ was used for training and testing.

hancement results comparable to the results of a DNN. Their models were tested on airplane, babble, cantine, market, and white noise. Their results for their noise general model on cantine noise can be compared with our results with cafeteria noise (well aware that there might be differences in the complexity of the noise signals). Their improvement from noisy to enhanced in terms of PESQ are of the same magnitudes for SNR $0, 5$ and 10 dB. The differences in improvement are of $0.01, 0.03$ and -0.04, respectively, where the sign correspond to improvement or decline of our model. For SNR 10 dB they have achieved an improvement of 0.36, from 2.07 to 2.43, which is a 0.04 higher difference than we obtained. At SNR 15 dB do their improved difference exceed our by 0.11, obtaining an improvement from 2.57 to 2.81.

Their obtained results in terms of STOI are also comparable. The noisy files with input STOI score above 0.80 get a reduced STOI score for the enhanced file for all the tested noise types. The difference is between 0.02 and 0.07. Files with lower input STOI are achieving improvements. (We have not tested with any input STOI lower than 0.77).

The SEGAN article used as the main motivation for this project compared SEGAN-enhanced signals with Wiener-enhanced signals. The PESQ and MOS score reported are regiven in Table 4.9. The PESQ improvement from noisy to SEGAN-enhanced is on 0.19, which is comparable with our results with and without latent noise. The best run with $\mathbf{z}$ achieve an improvement of 0.21, while the best run without $\mathbf{z}$ improves the noisy PESQ score by 0.18.



(a) SNR:0 dB

(b) SNR: 5 dB

(c) SNR: 10 dB

(d) SNR: 15 dB

Figure 4.8: The PESQ score of the noisy test set and the enhanced test set with latent noise **z** is visualized in relative frequency histograms separated according to SNR.

Observe that the PESQ score obtained for the wiener-enhanced signal is higher than the score obtained by SEGAN. A total of 16 listeners were presented with 20 sentences from their test set, to rate the set according to the MOS scale from Table 2.1. SEGAN enhanced signals obtained a higher mean score. As PESQ is used as an estimator of the MOS scores, the MOS scores are perhaps more interesting.

4.2.2 Subjective evaluation

Each of the six runs have enhanced the test set of 200 files, which means that there has not been time to get a full overview of the audible difference between the implementation with and without latent noise. Some sample files are available on Dropbox[1]. The validation scores in terms of PESQ have a non-increasing trend during training, and we have therefore included the enhanced validation set from the run and epoch with the overall highest average score for both models. Lastly, the enhanced test set files are compared with the enhanced results of a shorter run with latent noise, where the number of batches per epoch have been reduced from 40 to 10.

[1] `https://www.dropbox.com/sh/gps8xzvya9cftp9/AAAp6f7eGHCmoC3MFqeSrXiYa?dl=0`

Table 4.3: Average PESQ scores for different levels of SNR is calculated for the noisy test set and three enhanced versions of the test set. The setup without latent noise $\mathbf{z}$ was used for training and testing.

	0 dB	**5 dB**	**10 dB**	**15 dB**	**Mean**
Noisy	1.11	1.18	1.34	1.64	1.32
Run 1	1.23	1.40	1.60	1.78	1.50
Run 2	1.19	1.31	1.48	1.64	1.40
Run 3	1.20	1.38	1.60	1.79	1.49
Average	1.20	1.36	1.56	1.73	1.46

Table 4.4: Average STOI scores for different levels of SNR is calculated for the noisy test set and three enhanced versions of the test set. The setup without latent noise $\mathbf{z}$ was used for training and testing.

	0 dB	**5 dB**	**10 dB**	**15 dB**	**Mean**
Noisy	0.81	0.89	0.94	0.97	0.90
Run 1	0.80	0.85	0.88	0.90	0.86
Run 2	0.80	0.85	0.88	0.90	0.85
Run 3	0.78	0.83	0.86	0.88	0.84
Average	0.79	0.84	0.88	0.89	0.85

Validation set

The noisy files `noisy_f1_3_t-c1019_n46_snr_0.wav` (traffic, female speaker), `noisy_m1_5_x-c172` (station) and `noisy_m1_5_x-c1745_n46_snr_0.wav` (traffic, male speaker) and their enhancements are available on Dropbox. There is no audible difference between the enhancements of the file with traffic noise and male speaker. Both files have a reduced noise level without any additional speech distortion. The enhanced files with traffic noise and female speaker have musical noise for both enhanced versions, though the enhanced file without latent noise have a more intense ringing. The ringing gets more evident during training, which one can hear by listening to the enhanced validation set from the last epoch (epoch 10). The station noise is highly reduced in the enhanced versions of both models.

Test set

One noisy file with each type of the test set noises have been listened to at SNR 0 dB. One additional file have been analyzed for SNR $0, 5, 10$ and 15 dB. The noise signals `n68` and `n28` have enhanced files with a weak musical noise. The noise level is reduced, but at the cost of some degradation of the speech. Both models have similar results. `n78` have enhanced files with more distorted noise. `Traffic` do also result in a combination of noise reduction, speech degradation and musical noise.

The enhanced versions can be compared with the results of a more recent run with

Table 4.5: The PESQ scores of the trained model G for different noise sources have been compared. The model with latent noise $\mathbf{z}$ was used.

	0 dB		**5 dB**		**10 dB**		**15 dB**	
	Noisy	Enhanced	Noisy	Enhanced	Noisy	Enhanced	Noisy	Enhanced
Traffic	1.06	1.14	1.10	1.28	1.22	1.52	1.46	1.79
Cafeteria	1.12	1.27	1.24	1.53	1.50	1.82	1.95	2.08
Machine	1.12	1.24	1.18	1.43	1.32	1.68	1.58	1.94
Water	1.07	1.14	1.12	1.27	1.22	1.48	1.48	1.73
Wind	1.17	1.26	1.26	1.41	1.43	1.67	1.75	1.96

Table 4.6: The STOI scores of the trained model G for different noise sources have been compared. The model with latent noise $\mathbf{z}$ was used.

	0 dB		**5 dB**		**10 dB**		**15 dB**	
	Noisy	Enhanced	Noisy	Enhanced	Noisy	Enhanced	Noisy	Enhanced
Traffic	0.77	0.77	0.86	0.84	0.92	0.89	0.96	0.91
Cafeteria	0.85	0.84	0.93	0.89	0.97	0.92	0.99	0.93
Machine	0.86	0.85	0.92	0.89	0.95	0.91	0.97	0.92
Water	0.70	0.73	0.79	0.80	0.87	0.85	0.93	0.89
Wind	0.88	0.83	0.93	0.88	0.97	0.91	0.98	0.92

latent noise with a lower amount of training. Observe that the voices are more comfortable to listen to.

Restaurant noise have been considered for all levels of SNR, with and without latent noise. The noisy files have higher levels of noise compared to the enhanced files, but they are still more comfortable to listen to. The ehanced files have some level of musical noise, in addition to more uneven background noise. The speech have some distortion. Both models seem to have removed the background noise at SNR level 10 and 15 dB.

Spectrograms illustrating the enhancement of a noisy speech signal with `n28:Machine noise` are shown in Figure 4.9. The horizontal patterns are noise, as can be seen in Figure 3.1g. Observe that the enhancement with $\mathbf{z}$ seem to have removed more of the noise at frequency 1000 and in frequency range $3000 - 5000$ Hz. The version enhanced with $\mathbf{z}$ have recovered stronger vertical stripes between 3 and 4 seconds. The files can be listened to at Dropbox. Both GAN-setups have been successful in reducing the noise level. The speech is only weakly distorted for the setup with $\mathbf{z}$, while for the setup without $\mathbf{z}$, the distortion is more severe. The version with $\mathbf{z}$ have some musical noise.

Table 4.7: The PESQ scores of the trained model G for different noise sources have been compared. The model without latent noise $\mathbf{z}$ was used.

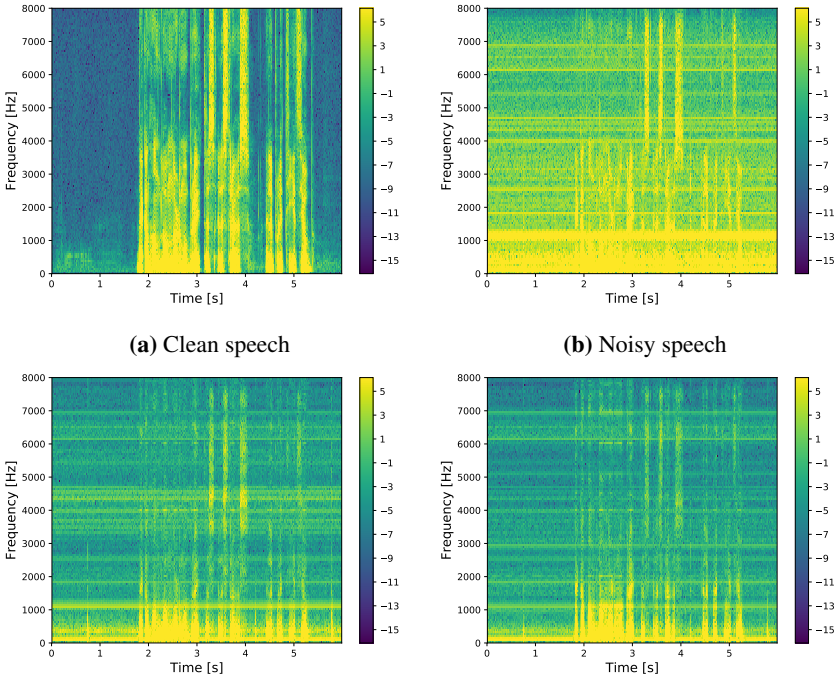
	0 dB		**5 dB**		**10 dB**		**15 dB**	
	Noisy	Enhanced	Noisy	Enhanced	Noisy	Enhanced	Noisy	Enhanced
Traffic	1.06	1.16	1.10	1.29	1.22	1.48	1.46	1.68
Cafeteria	1.12	1.31	1.24	1.54	1.50	1.76	1.95	1.89
Machine	1.12	1.24	1.18	1.43	1.32	1.63	1.58	1.80
Water	1.07	1.16	1.12	1.30	1.22	1.51	1.48	1.69
Wind	1.17	1.26	1.26	1.43	1.43	1.64	1.75	1.82

Table 4.8: The STOI scores of the trained model G for different noise sources have been compared. The model without latent noise $\mathbf{z}$ was used.

	0 dB		**5 dB**		**10 dB**		**15 dB**	
	Noisy	Enhanced	Noisy	Enhanced	Noisy	Enhanced	Noisy	Enhanced
Traffic	0.77	0.77	0.86	0.84	0.92	0.88	0.96	0.90
Cafeteria	0.85	0.83	0.93	0.88	0.97	0.90	0.99	0.91
Machine	0.86	0.83	0.92	0.87	0.95	0.89	0.97	0.90
Water	0.70	0.73	0.79	0.80	0.87	0.85	0.93	0.89
Wind	0.88	0.82	0.93	0.87	0.97	0.90	0.98	0.91

Table 4.9: Noisy signals and Wiener- and SEGAN-enhanced signals were compared objectively in terms of PESQ and subjectively in terms of MOS by Pascual et al. (2017). The results are regiven here.

Metric	**Noisy**	**Wiener**	**SEGAN**
PESQ	1.97	2.22	2.16
MOS	2.09	2.70	3.18



(a) Clean speech

(b) Noisy speech

(c) Enhanced with **z**

(d) Enhanced without **z**

Figure 4.9: Log-density spectrograms of clean speech, noisy speech and the GAN-enhanced versions, from one setup with latent noise **z** and one setup without latent noise **z**. The noisy speech is constructed by adding `n28:Machine noise` at SNR 0 dB.

Chapter 5

Discussion

5.1 Comparison of the models with and without latent noise

The enhancement results obtained with and without use of latent noise are in general very similar. The setup with latent noise achieve generally slightly better scores in terms of STOI and PESQ, but the differences in average are very small. There has not been time to listen to all the different enhanced files, but the impression is that the setup without latent noise more often get musical noise. The data material and number of experiments is too limited to conclude that a setup without latent noise could not be an alternative setup.

Why is the latent noise leading to better results? The setup with latent noise have higher representational power, since it has more free parameters than the model without latent noise. It adds randomness to the model, which means that it might be interpreted as some sort of regulizer. When pix2pix removed latent noise from their setup, dropout(Srivastava et al., 2014) was added instead. They reported that the weights corresponding to z eventually became zero, such that the random noise did not influence the final result. We did a minor experiment to check this, and enhanced a noisy signal 100 times, with new randomly drawn z's for each enhancement. The resulting enhanced files are plotted in Appendix B.1, where the plots show each realization subtracted the mean value of the runs. The maximum distance from 0 is 55.2, which is a small number compared to the range in the enhanced file being from -11728.0 to 16765.0 (1.9%). The average variance of each sample value is only 1.43. Subjectively, I could not here any difference between the different enhancements. This suggests that z do not lead to any large variation of the enhanced output, even if there is some randomness.

If we take a look at the validation scores in epoch 10, it looks like the validation score correlates with the test score. That is not surprising, as both scores are trying to measure the model's ability to generalize. This suggests that an implementation of early stopping based on the validation set's PESQ score could be beneficial to improve the end result.

5.1.1 Overall evaluation

The PESQ score curves and STOI score curves for the three runs during training presented, separated by run, are shown in Figure B.2 and Figure B.3. The curves for each run have a similar shape for the different levels of SNR. That is, the minimum and maximum points, etc. occur at the same epoch for the different SNRs. The extreme points for PESQ and STOI seem visually to have some correlation. This may indicate that when the GAN is ”good”, it is good for all the four SNRs, and oppositely, when it performs poorly, it performs poorly for all SNRs.

The model is trained on a large variety of speakers and noise signals. The model is able to decrease the noise level for all the noise sources tested on, though it sometimes leads to distorted speech. The training procedure has not been optimized, so improved results might be possible with other hyperparameters, initializers and optimizers. The hyperparameters and architecture used is based on SEGAN's article for the setup with latent noise, and is therefore not necessarily the optimal choice for the model without latent noise. It is therefore not possible to conclude that the model with latent noise in general is better than the model without latent noise.

The 5 dB SNR level was not used during training. There was no evident difference in the enhancement performed at this level compared to the other levels of SNR. The model is probably already used to a range of different levels since the levels of SNR are calculated on the full speech and noise signals.

5.2 Exploding loss function and GAN training

Exploding loss functions in the discriminator are appearing for some runs, as shown in Figure 4.3 and Figure 4.4. However, the exploding loss is on the discriminator's loss function, i.e. the discriminator's predictions distance to the correct labels/values 0 and 1. These exploding loss functions do not seem to correlate with the enhancement performance. It would be interesting to see what happens to the gradients when the loss function is exploding.

The training PESQ scores for the model without latent noise are generally decreasing the last epochs - maybe the model gets overfitted? Due to computational requirements, it is not possible to do all the tests again for reduced training time. For the effect of comparison, we did one shorter run, where the amount of training data per epoch were reduced by a factor of 4. The STOI and PESQ scores per epoch are displayed in Figure B.4 and Figure B.5, for the setup with and without latent noise, respectively. One can observe that the scores are almost monotonically increasing in time for both PESQ and STOI. 4 epochs in this plot corresponds to the amount of data trained on for 1 epoch in the longer runs. Run 1, 2 and 3 with latent noise did also have a positive trend the first 3 epochs, while the trend without z is more mixed. The average scores are displayed in Table B.1 and Table B.2. The shorter run with latent noise achieve significantly higher scores in terms of PESQ, specifically for the higher SNR levels. The results on STOI are also weakly increased. The overall average is higher in both measures. The shorter run without latent noise get the same average score for SNR 0 and 5 dB as the best long run without latent noise, while there is improvement for SNR 10 and for 15 dB. In STOI is there a slight decrease in the

score for SNR 0 and 5 dB and a slight increase for 10 and 15 dB. The overall average is highest for the short run for both PESQ and STOI.

5.3 Future work

5.3.1 Early stopping

It would be interesting to implement early stopping based on the validation set scores in terms of PESQ. The overall highest test scores in terms of PESQ are obtained for the model with the highest validation set score at epoch 10, which is the model with a shorter training time, with latent noise. The differences between the different end scores in PESQ for the other runs with latent noise are minor, and so are the average PESQ scores of the enhanced test set. The overall means differs only by 0.02., while the mean for the shorter run is increased by 0.1.

5.3.2 Larger input windows

The model is only enhancing approximately one second at a time. Speech signals are time series and have strong temporal dependencies. Enhancing only one second at a time might lead to shifts at the boundary points between different time windows. We have not explored whether this actually is a problem. This is a common setup in DNN methods, among others tested during my project thesis (Vik, 2018), where two windows before and after the current enhanced window ere given as input to the network.

5.3.3 Features or other training tricks

Some of the enhanced signals have a typical ”musical noise”. Wang and Chen (2018) concluded in their overview article on supervised speech separation with deep learning that a combination of deep learning and feature extraction is smart. A recent article by the authors of SEGAN (Pascual et al., 2019) include tricks to avoid highfrequency artifacts and exploding gradient in the discriminator's loss function. Phase shift (Donahue et al., 2018) can reduce high-frequency artifacts in the outputs of G, while spectral normalization (Miyato et al., 2018; Zhang et al., 2018) is used to avoid exploding gradients in D.

Chapter 6

Conclusion

In this master thesis, generative adversarial nets have been implemented and used to do end-to-end speech enhancement. The proposed setup without latent noise have been compared to the original setup with latent noise. A large variety of speakers and noise signals have been used during training and testing. The enhancement results show an improvement in terms of PESQ, while the STOI score in general decline slightly. The original setup performs slightly better in average in both evaluation metrics. Our implemented models perform comparable to the other speech enhancement methods based on GAN in literature. The GAN's training have been unstable, and there have been high-valued loss functions in the discriminator.

A recent article by the authors of SEGAN seem to have solved some of the problems we have encountered. Including phase shuffle, spectral normalization and features in the model might prevent exploding gradients in the discriminator and high-frequency artifacts in the generator-enhanced speech.

Bibliography

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., Zheng, X., 2015. TensorFlow: Large-scale machine learning on heterogeneous systems. Last accessed on 2018-12-11.
URL `https://www.tensorflow.org/`

Audio Software Engineering and Siri Speech Team, 2018. Optimizing siri on homepod in far-field settings. Apple Machine Learning Journal 1 (12).
URL `https://machinelearning.apple.com/2018/12/03/optimizing-siri-on-homepod-in-far-field-settings.html#3`

Benesty, J., Makino, S., Chen, J., 2005. Speech Enhancement. Signals and Communication Technology. Springer.
URL `https://books.google.no/books?id=7yJXMJUw03oC`

Benesty, J., Sondhi, M. M., Huang, Y. A., 2008. Springer Handbook of Speech Processing. Springer-Verlag, Berlin, Heidelberg.

Chollet, F., et al., 2015. Keras. Last accessed on 2018-12-11.
URL `https://keras.io`

Donahue, C., McAuley, J., Puckette, M., 2018. Synthesizing audio with generative adversarial networks. CoRR abs/1802.04208.
URL `http://arxiv.org/abs/1802.04208`

Erdogan, H., Hershey, J., Watanabe, S., Le Roux, J., 04 2015. Phase-sensitive and recognition-boosted speech separation using deep recurrent neural networks. pp. 708–712.

Fu, S., Wang, T., Tsao, Y., Lu, X., Kawai, H., Sep. 2018. End-to-end waveform utterance enhancement for direct evaluation metrics optimization by fully convolutional neural

networks. IEEE/ACM Transactions on Audio, Speech, and Language Processing 26 (9), 1570–1584.

Fu, S.-W., Tsao, Y., Lu, X., 2016. Snr-aware convolutional neural network modeling for speech enhancement. In: Interspeech 2016. pp. 3768–3772.
URL http://dx.doi.org/10.21437/Interspeech.2016-211

Gelderblom, F. B., Tronstad, T. V., Viggen, E. M., 2017. Subjective intelligibility of deep neural network-based speech enhancement. In: Proc. Interspeech 2017. pp. 1968–1972.
URL http://dx.doi.org/10.21437/Interspeech.2017-1041

Glorot, X., Bordes, A., Bengio, Y., 2011. Deep sparse rectifier neural networks. In: Gordon, G. J., Dunson, D. B., Dudík, M. (Eds.), AISTATS. Vol. 15 of JMLR Proceedings. JMLR.org, pp. 315–323.
URL http://dblp.uni-trier.de/db/journals/jmlr/jmlrp15.html#GlorotBB11

Goodfellow, I., Bengio, Y., Courville, A., 2016. Deep Learning. MIT Press, http://www.deeplearningbook.org.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y., 2014. Generative adversarial nets. In: Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N. D., Weinberger, K. Q. (Eds.), Advances in Neural Information Processing Systems 27. Curran Associates, Inc., pp. 2672–2680.
URL http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf

He, K., Zhang, X., Ren, S., Sun, J., 2015. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. CoRR abs/1502.01852.
URL http://arxiv.org/abs/1502.01852

Hinton, G., 2012. Neural networks for machine learning. Coursera, video lectures.

Hinton, G., Deng, l., Yu, D., E. Dahl, G., Mohamed, A.-r., Jaitly, N., Senior, A., Vanhoucke, V., Nguyen, P., Sainath, T., Kingsbury, B., 11 2012. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. Signal Processing Magazine, IEEE 29, 82–97.

Hu, G., 2014. 100 nonspeech sounds. Last accessed on 2018-12-06.
URL http://web.cse.ohio-state.edu/pnl/corpus/HuNonspeech/HuCorpus.html

Hui, L., Cai, M., Guo, C., He, L., Zhang, W.-Q., Liu, J., 12 2015. Convolutional maxout neural networks for speech separation. pp. 24–27.

Ioffe, S., Szegedy, C., 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. CoRR abs/1502.03167.
URL http://arxiv.org/abs/1502.03167

Isola, P., Zhu, J., Zhou, T., Efros, A. A., 2016. Image-to-image translation with conditional adversarial networks. CoRR abs/1611.07004.
URL http://arxiv.org/abs/1611.07004

ITU, 01 1996. Itu-t: Methods for subjective determination of transmission quality. rec. p.800. ITU-T P.

James, G., Witten, D., Hastie, T., Tibshirani, R., 2014. An Introduction to Statistical Learning: with Applications in R. Springer Texts in Statistics. Springer New York.

Jarrett, K., Kavukcuoglu, K., Ranzato, M., LeCun, Y., 2009a. What is the best multi-stage architecture for object recognition? In: ICCV. IEEE, pp. 2146–2153.
URL http://dblp.uni-trier.de/db/conf/iccv/iccv2009.html#JarrettKRL09

Jarrett, K., Kavukcuoglu, K., Ranzato, M., Lecun, Y., 09 2009b. What is the best multi-stage architecture for object recognition? Vol. 12.

Keintz, C. K., Bunton, K., Hoit, J. D., 2007. Influence of visual information on the intelligibility of dysarthric speech. American Journal of Speech-Language Pathology 16 (3), 222–234.
URL https://pubs.asha.org/doi/abs/10.1044/1058-0360%282007/027%29

Kingma, D., Ba, J., 2015. Adam: A method for stochastic optimization. ArXiv:1412.6980v9.

Lecun, Y., 1989. Generalization and network design strategies. Elsevier.

Lecun, Y., Bottou, L., Bengio, Y., Haffner, P., 1998. Gradient-based learning applied to document recognition. In: Proceedings of the IEEE. pp. 2278–2324.

Leshno, M., Lin, V. Y., Pinkus, A., Schocken, S., 1993. Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. Neural Networks 6 (6), 861 – 867.
URL http://www.sciencedirect.com/science/article/pii/S0893608005801315

Maas, A. L., 2013. Rectifier nonlinearities improve neural network acoustic models.

Mao, X., Li, Q., Xie, H., Lau, R. Y. K., Wang, Z., 2016. Multi-class generative adversarial networks with the L2 loss function. CoRR abs/1611.04076.
URL http://arxiv.org/abs/1611.04076

McFee, B., 2016. resampy: efficient sample rate conversion in Python. Last accessed on 2018-12-10.
URL https://resampy.readthedocs.io/en/stable/index.html#

Michelsanti, D., Tan, Z.-H., 2017. Conditional generative adversarial networks for speech enhancement and noise-robust speaker verification. In: INTERSPEECH.

Mirza, M., Osindero, S., 2014. Conditional generative adversarial nets. CoRR abs/1411.1784.
URL http://arxiv.org/abs/1411.1784

Mitchell, T., 1997. Machine Learning. McGraw-Hill.

Miyato, T., Kataoka, T., Koyama, M., Yoshida, Y., 2018. Spectral normalization for generative adversarial networks. CoRR abs/1802.05957.
URL http://arxiv.org/abs/1802.05957

Nair, V., Hinton, G. E., 2010. Rectified linear units improve restricted boltzmann machines. In: Proceedings of the 27th International Conference on International Conference on Machine Learning. ICML'10. Omnipress, USA, pp. 807–814.
URL http://dl.acm.org/citation.cfm?id=3104322.3104425

Nasjonalbiblioteket, 2016. "nb tale - a basic acoustic phonetic speech database for norwegian". Last accessed on 2018-12-26.
URL https://www.nb.no/sprakbanken/show?serial=oai%3Anb.no%3Asbr-31&lang=en

Oliphant, T., 2006. A Guide to NumPy. Trelgol Publishing.
URL https://books.google.no/books?id=fKulSgAACAAJ

Ortega-Garcia, J., Gonzalez-Rodriguez, J., Oct 1996. Overview of speech enhancement techniques for automatic speaker recognition. In: Proceeding of Fourth International Conference on Spoken Language Processing. ICSLP '96. Vol. 2. pp. 929–932 vol.2.

Paliwal, K., Wójcicki, K., Shannon, B., 2011. The importance of phase in speech enhancement. Speech Communication 53 (4), 465 – 494.
URL http://www.sciencedirect.com/science/article/pii/S0167639310002086

Park, S. R., Lee, J., 2016. A fully convolutional neural network for speech enhancement. CoRR abs/1609.07132.
URL http://arxiv.org/abs/1609.07132

Pascual, S., Bonafonte, A., Serrà, J., 2017. SEGAN: speech enhancement generative adversarial network. CoRR abs/1703.09452.
URL http://arxiv.org/abs/1703.09452

Pascual, S., Serrà, J., Bonafonte, A., 2019. Towards generalized speech enhancement with generative adversarial networks. CoRR abs/1904.03418.
URL http://arxiv.org/abs/1904.03418

Radford, A., Metz, L., Chintala, S., 2015. Unsupervised representation learning with deep convolutional generative adversarial networks. arXiv preprint arXiv:1511.06434.

Reddi, S. J., Kale, S., Kumar, S., 2018. On the convergence of adam and beyond. In: International Conference on Learning Representations.
URL https://openreview.net/forum?id=ryQu7f-RZ

Rix, A. W., Beerends, J. G., Hollier, M. P., Hekstra, A. P., 2001. Perceptual evaluation of speech quality (pesq)-a new method for speech quality assessment of telephone networks and codecs.

Ronneberger, O., Fischer, P., Brox, T., 2015. U-net: Convolutional networks for biomedical image segmentation. CoRR abs/1505.04597.
URL http://arxiv.org/abs/1505.04597

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

Salimans, T., Goodfellow, I. J., Zaremba, W., Cheung, V., Radford, A., Chen, X., 2016. Improved techniques for training gans. CoRR abs/1606.03498.
URL http://arxiv.org/abs/1606.03498

Schaul, T., Antonoglou, I., Silver, D., 2014. Unit tests for stochastic optimization.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R., 2014. Dropout: A simple way to prevent neural networks from overfitting. J. Mach. Learn. Res. 15 (1), 1929–1958.
URL http://dl.acm.org/citation.cfm?id=2627435.2670313

Taal, C. H., Hendriks, R. C., Heusdens, R., Jensen, J., Sept 2011. An algorithm for intelligibility prediction of time–frequency weighted noisy speech. IEEE Transactions on Audio, Speech, and Language Processing 19 (7), 2125–2136.

Tamura, S., Waibel, A., April 1988. Noise reduction using connectionist models. In: ICASSP-88., International Conference on Acoustics, Speech, and Signal Processing. pp. 553–556 vol.1.

Thiemann, J., Ito, N., Vincent, E., Jun. 2013. DEMAND: a collection of multi-channel recordings of acoustic noise in diverse environments. Supported by Inria under the Associate Team Program VERSAMUS.
URL https://doi.org/10.5281/zenodo.1227121

Vik, M. L., 2018. Deep Learning for Speech Separation. Project Thesis. Norwegian University of Science and Technology (NTNU).

Wang, D., Chen, J., 2018. Supervised speech separation based on deep learning: An overview. IEEE/ACM Transactions on Audio, Speech, and Language Processing 26 (10), 1702–1726.

Weninger, F., Erdogan, H., Watanabe, S., Vincent, E., Le Roux, J., Hershey, J. R., Schuller, B., 2015a. Speech enhancement with lstm recurrent neural networks and its application to noise-robust asr. In: Vincent, E., Yeredor, A., Koldovský, Z., Tichavský, P. (Eds.), Latent Variable Analysis and Signal Separation. Springer International Publishing, Cham, pp. 91–99.

Weninger, F., Hershey, J., Le Roux, J., Schuller, B., 02 2015b. Discriminatively trained recurrent neural networks for single-channel speech separation. 2014 IEEE Global Conference on Signal and Information Processing, GlobalSIP 2014, 577–581.

Yang, L.-P., Fu, Q.-J., 2005. Spectral subtraction-based speech enhancement for cochlear implant patients in background noise. The Journal of the Acoustical Society of America 117 (3), 1001–1004.
URL https://doi.org/10.1121/1.1852873

Zhang, H., Goodfellow, I., Metaxas, D., Odena, A., 2018. Self-attention generative adversarial networks. arXiv preprint arXiv:1805.08318.

Appendix A

Information Theory

A.1 Kullback-Leibler divergence

Kullback-Leibler (KL) divergence is a measure of the difference between two distributions. Let $P(x)$ and $Q(x)$ be distributed over the same random variable x. The KL divergence is given as

$$D_{\mathrm{KL}}(P\Big\|Q) = E_{x\sim P}\left[\log\frac{P(x)}{Q(x)}\right]. \tag{A.1}$$

The KL-divegence is non-negative, and zero only if P and Q have the same distributions in the case of discrete distributions or almost equal in the case of continuous distributions. However, KL divergence is not symmetric, $D_{\mathrm{KL}}(P\Big\|Q) \neq D_{\mathrm{KL}}(Q\Big\|P)$.

A.2 Jensen-Shannon divergence

Jensen-Shannon divergence (JSD) is a different measure of the similarity between two probability distributions $P(x)$ and $Q(x)$. It is a symmetric and smoother version of KL divergence. It is defined by

$$JSD(P\Big\|Q) = \frac{1}{2}(P\Big\|M) + \frac{1}{2}(Q\Big\|M), \tag{A.2}$$

where $M = \frac{1}{2}(P+Q)$. Note that JS divergence has a lower bound of 0, since it is a sum of two elements with minimum value 0.

Appendix B

Further experimental details

B.1 Model summaries

GAN setup, with z:

```
__________________________________________________________________________________________
Layer (type)                 Output Shape         Param #     Connected to
==========================================================================================
in_noisy (InputLayer)        (None, 16384, 1)     0
__________________________________________________________________________________________
noise_input (InputLayer)     (None, 8, 1024)      0
__________________________________________________________________________________________
model_1 (Model)              (None, 16384, 1)     74120049    in_noisy[0][0]
                                                              noise_input[0][0]
__________________________________________________________________________________________
model_2 (Model)              (None, 1)            24378106    model_1[1][0]
                                                              in_noisy[0][0]
==========================================================================================
Total params: 98,498,155
Trainable params: 74,120,049
Non-trainable params: 24,378,106
__________________________________________________________________________________________
```

GAN setup, without z:

```
Layer (type)                 Output Shape         Param #     Connected to
==========================================================================================
in_noisy (InputLayer)        (None, 16384, 1)     0
__________________________________________________________________________________________
model_1 (Model)              (None, 16384, 1)     57867121    in_noisy[0][0]
__________________________________________________________________________________________
model_2 (Model)              (None, 1)            24378106    model_1[1][0]
                                                              in_noisy[0][0]
==========================================================================================
Total params: 82,245,227
Trainable params: 57,867,121
Non-trainable params: 24,378,106
__________________________________________________________________________________________
```

B.2 Variation due to latent noise

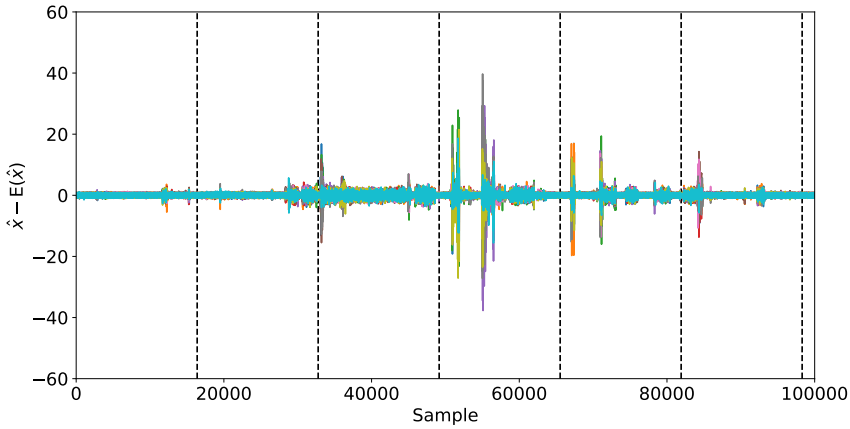
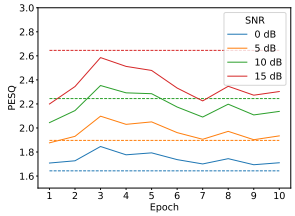
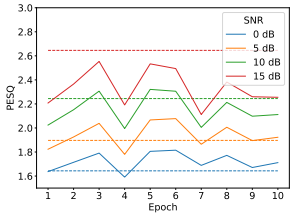


Figure B.1: A noisy file with `n28 : machine noise` have been enhanced $N = 100$ times with new randomly drawn latent noise $\mathbf{z}$. Each enhanced sequence minus the average of all the sequences $\hat{\mathbf{x}}_i - \mathrm{E}[\mathbf{x}]$ is plotted. The dashed lines represent the window limits.

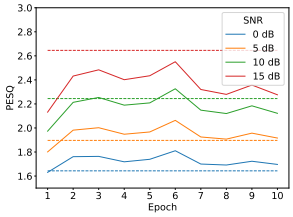
B.3 Validation set scores for the longer runs



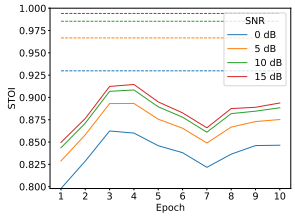
(a) PESQ scores run 1



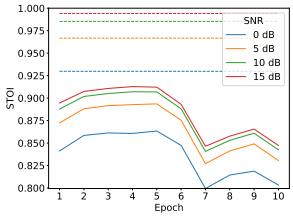
(b) PESQ scores run 2



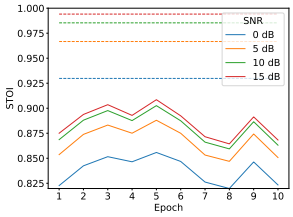
(c) PESQ scores run 3



(d) STOI scores run 1

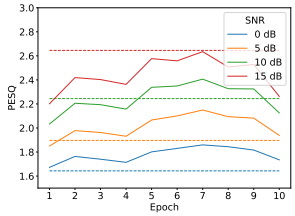


(e) STOI scores run 2

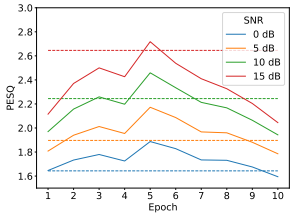


(f) STOI scores run 3

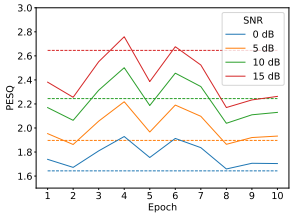
Figure B.2: PESQ and STOI scores during training for the enhanced validation set, for the three runs with latent noise **z**.



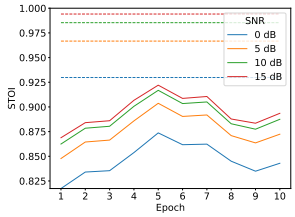
(a) PESQ scores run 1



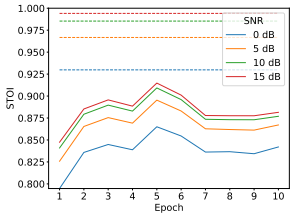
(b) PESQ scores run 2



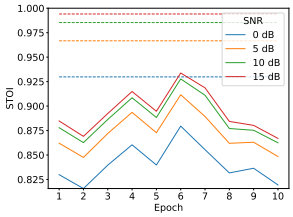
(c) PESQ scores run 3



(d) STOI scores run 1



(e) STOI scores run 2



(f) STOI scores run 3

Figure B.3: PESQ and STOI scores during training for the enhanced validation set, for the three runs without latent noise **z**.

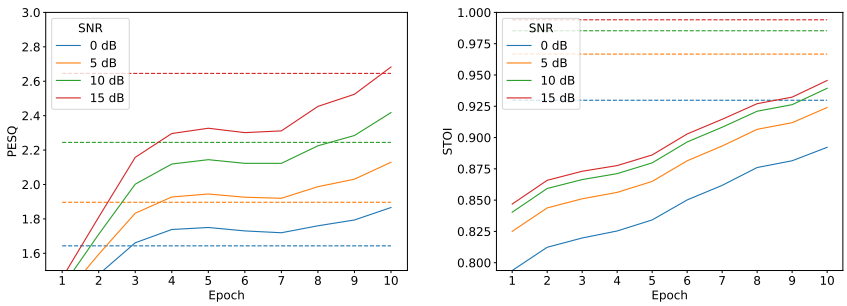
B.4 Results after a shorter run

Table B.1: Average PESQ and STOI scores for different levels of SNR is calculated for the enhanced version of the test set. The setup with latent noise **z** was used for training and testing. The GAN was trained with 10 batches per epoch, which is $1/4$ of the amount used in the other runs.

	0 dB	**5 dB**	**10 dB**	**15 dB**	**Mean**
PESQ	1.26	1.45	1.75	2.11	1.64
STOI	0.81	0.88	0.92	0.94	0.89

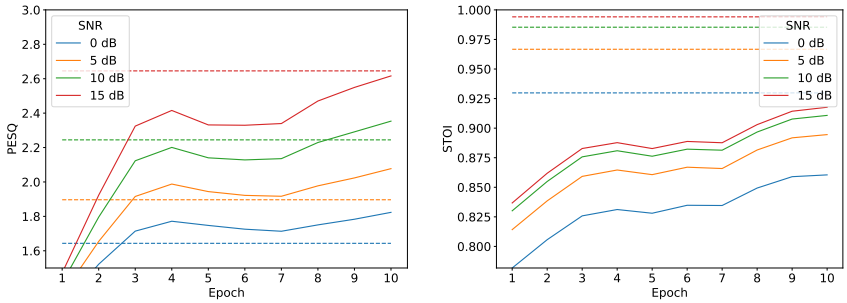
Table B.2: Average PESQ and STOI scores for different levels of SNR is calculated for the enhanced version of the test set. The setup without latent noise **z** was used for training and testing. The GAN was trained with 10 batches per epoch, which is $1/4$ of the amount used in the other runs.

	0 dB	**5 dB**	**10 dB**	**15 dB**	**Mean**
PESQ	1.23	1.40	1.68	2.01	1.58
STOI	0.79	0.86	0.90	0.92	0.87



(a) PESQ **(b)** STOI

Figure B.4: PESQ and STOI scores during training, after enhancement of the validation set. The GAN was trained with 10 batches per epoch, which is $1/4$ of the amount used in the other runs. The setup with latent noise **z** was used.



(a) PESQ **(b)** STOI

Figure B.5: PESQ and STOI scores during training, after enhancement of the validation set. The GAN was trained with 10 batches per epoch, which is $1/4$ of the amount used in the other runs. The setup without latent noise $\mathbf{z}$ was used.