SPEECH TECHNOLOGY PROJECT FINAL REPORT

SEGAN

(Speech Enhancement Generative Adversarial Network)

Under the Guidance of

Prof. Hemant Patil

Author

Kaushik Makwana

(201501118)

Abstract

The aim of the speech enhancement is to improve the intelligibility and quality of the speech. By recording the speech signal in the noisy environment, clean speech signals are degraded. Speech enhancement reduces the noise without distorting the original (clean). I have worked on SEGAN (Speech Enhancement using Generative Adversarial Network) in course project of CT478 (Speech Technology), so this is the final report of that course project.

1. Introduction

Speech enhancement aims to improve speech quality by using various algorithms. The main objective of enhancement is improvement in intelligibility and/or overall perceptual quality of degraded speech signal using audio signal processing techniques. Enhancing of speech degraded by noise is the most important field of speech enhancement, and used for many applications such as mobile phones, VoIP, teleconferencing systems, speech recognition, and hearing aids. We can use speech enhancement before amplification so it will significantly reduce the discomfort and it will also increase intelligibility of given signal. Speech enhancement is also being used in preprocessing stage of Speaker Identification and Speaker recognition systems.

There are many classical method for speech enhancement like spectral subtraction, Wiener filtering, statistical model-based methods, and subspace algorithms. Neural networks and also Recurrent Neural Networks (RNNs) have been also applied to speech enhancement in recent times. Most of current systems are STFT based analysis and they only modify spectrum magnitude. However further studies show that significant improvements of speech quality are possible, especially when a clean phase spectrum is known. There are many researches are going on in field of deep learning and generative networks field. GAN has shown good level of success in generating realistics images. We can use GAN to implement speech enhancement system.

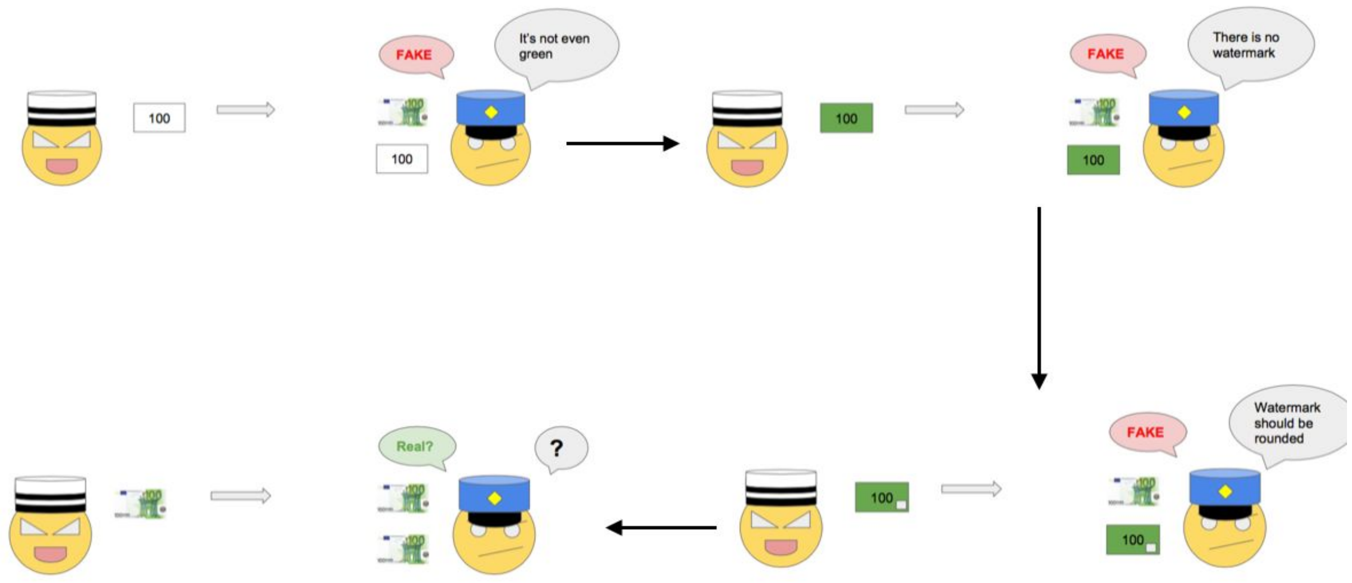
2. GAN

2.1 Introduction

Generative adversarial networks (GANs) are deep neural net architectures comprised of two nets, pitting one against the other (thus the “adversarial”).
GANs’ potential is huge, because they can learn to mimic any distribution of data. That is, GANs can be taught to create worlds almost similar to our own in any domain: images, music, speech, etc. They are robot artists in a sense, and their output is impressive.
GANs can also be implemented for speech enhancement system where we can train GANs by giving noisy and corresponding clean speech data and then after that in testing phase if we give noisy speech it will give enhanced output.

2.2 Understanding

There is very interesting illustration to understand GAN in simple words. We take will take an example of counterfeiter trying to fool bank employee. Here we can think of bank employee as discriminator and counterfeiter as Generator. So generator will try to make as realistic data as possible so in this case counterfeiter will give currency to bank employee and there is some mechanism(called backpropagation) in GAN that will try to improve generator output. And after several iteration counterfeiter will be able to confuse bank employee. So now Generator is able to produce close to realistic data that discriminator can not identify it as fake data.



2.3 GAN Working

GAN is basically combination of two neural network in which one neural network, called the *generator(*counterfeiter) , generates new data(fake currency) instances, while the other, the *discriminator(Bank employee)*, evaluates them for authenticity; i.e. the discriminator decides whether each instance of data it reviews belongs to the actual training dataset or not.

Here are the steps a GAN takes:

- The generator takes in random numbers and returns an output.
- This generated output is fed into the discriminator which is trained using real and fake database.
- The discriminator takes output from Generator and gives probability between 0 and 1, with 1 represtaning real signal and 0 representing fake signal.

3. implementation of GAN

GAN is used in image processing for many years. In recent year it is started to be used in speech processing also. For the implementation of GAN and to understand internal functions of generator, discriminator and loss function of GANs, I decided to first implement it for MNIST database. As we train our GAN with this database our GANs would be able to generate some realistic handwritten numbers. Then I tried to implement for speech database.

3.1 Batch normalization

Batch normalization is very important preprocessing step. We have to normalize data before feeding them into our GAN. It is very important for us that all our data is on same scale before as fed them to GAN, Batch normalization ensures that. This is the basic algorithm for batch normalization. After normalization of every data we train GAN using that data.

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\ldots m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

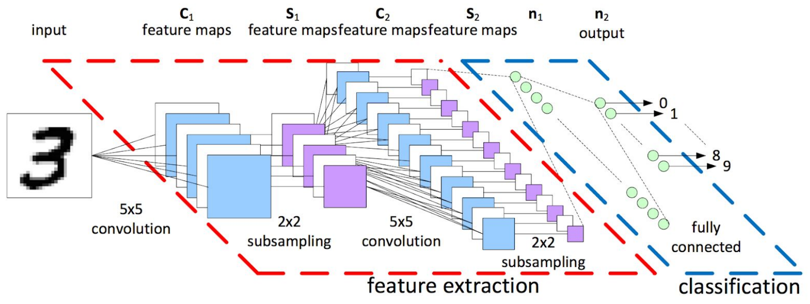
$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

source

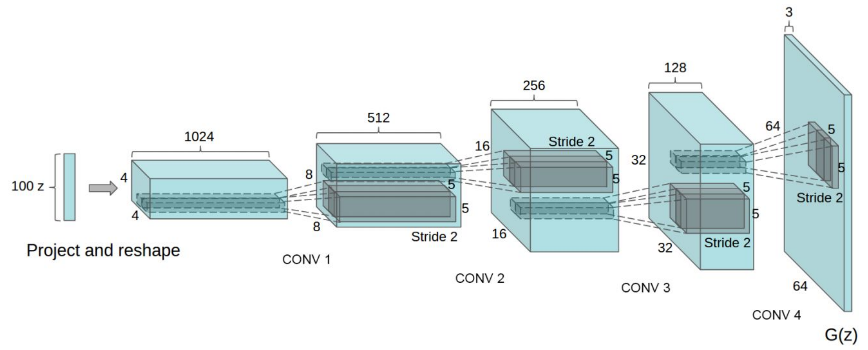
3.2 Discriminator



source

Discriminator is convolution network. what it does is it takes input as whole image by Strided convolution method it creates strides in image and from that it extracts features. So from given input images it create some data that can be used to classify between different numbers as we would have different clusters for different numbers. So if input is not close to any cluster than it means that it’s not representing any numbers.

3.3 Generator



Generator is deconvolution network. so what generator does is in beginning it starts with random data and from that data it builds an image, that image is sent to discriminator which extracts features from that and see whether it is near to specified cluster or not, if not than using back tracing generator try to generate that specified handwritten number.

3.4 Optimisation Algorithm

We use loss function for both generator and discriminator to optimize our algorithm for that we have to have some function that helps both to give better performance.

3.4.1 Loss function for Discriminator

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

source

This function optimize the probability that real data x is rated highly and output of generator G(z) is generated poorly. This is the basic need for discriminator that it can identify real data highly and make sure that generator data is poorly generated.

3.4.2 Loss function for Generator

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

source

This function ensures that output(G(z)) from generator is generated highly.

4. Implementation for speech data

Now as I had explored GAN and implemented it for MNIST database. Now I have to use that knowledge to implement that for speech data. So to use GAN for speech enhancement I had to do following task

- Feature Extraction from speech
- Train DNN from extracted features
- Implement GAN from DNN
- Enhance speech using GAN

4.1 Feature Extraction

The information in speech signal is actually represented by short term amplitude spectrum of the speech waveform. This allows us to extract features based on the short term amplitude spectrum from speech (phonemes). Feature Extraction plays very crucial role in the overall performance of the system. There are many techniques for feature extraction is available like

- Linear predictive analysis (LPC)
- Linear Predictive Cepstral Coefficients(LPCC)
- Mel-frequency cepstral coefficients(MFCC)
- Mel scale cepstral analysis (MEL)
- Gammatone Frequency spectral Coefficients (GFSC)

4.2 Gammatone Frequency spectral Coefficients

I have used GFSC for feature extraction because GFSCs are biologically inspired acoustic features because it gives good approximation of the human auditory filter. It has also shown better recognition performance at low SNRs. It is also easy to inverse of GFSCs.
It uses Set of Gammatone filters(often referred as channels) with different center frequencies, which is used to create a Gammatone filter bank which is shown here.so it works in bandpass like our hearing mechanism.

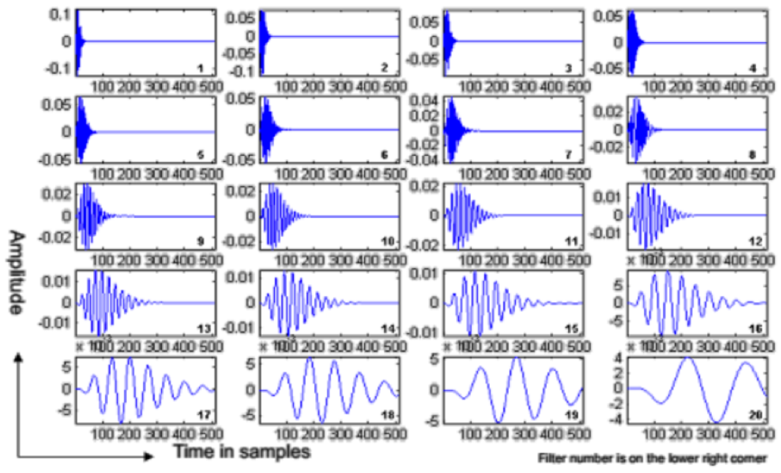
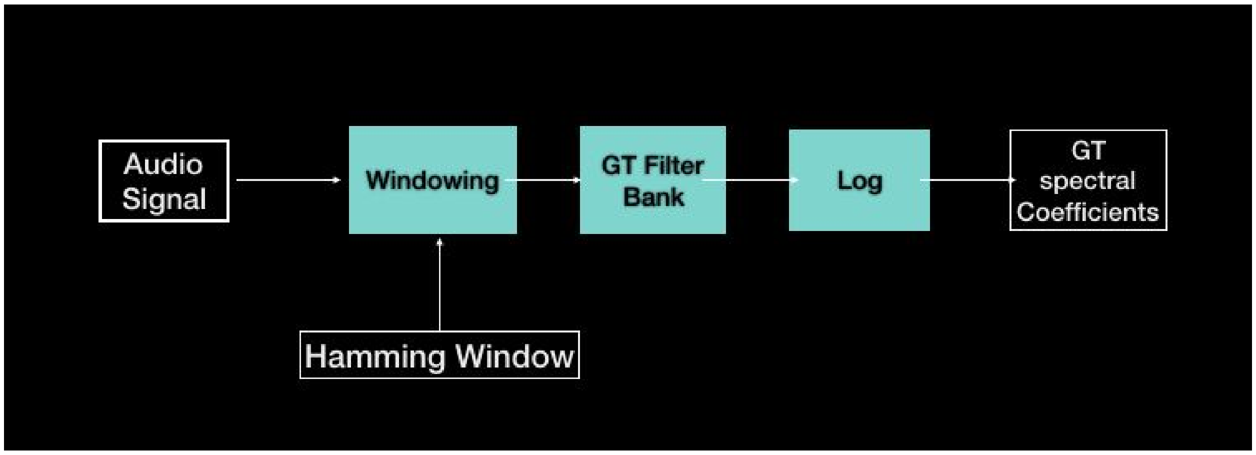


Figure 5-4 Impulse responses of individual filters of a 20 channel filter-bank **[44]**

source

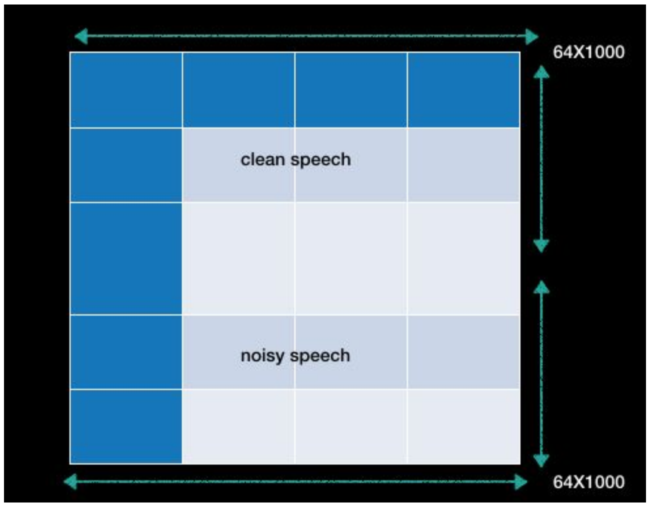
4.2.1 Algorithm for filtering

To extract GFSC from our speech signal I will first window it using hamming window function(for almost 10ms) than I will gave that windowed speech to gammatone filterbank. Speech will have something like [N] X [1] dimensioned speech and [64] X [n] filter bank (I am are using 64 channel filter bank). So I will have [64] X [n] dimensioned output for every speech sample.



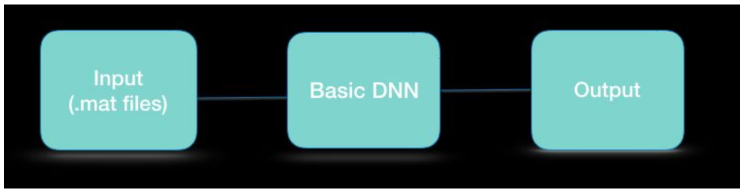
4.3 Create Input to train DNN

As I have extracted features from given training speech signals now I have to create batches from that extracted features. Those features will be in dimension of [64] X [some lac.]. I have extracted features for noisy speech and corresponding clean speech. For creating a batch i have to take [64] X [1000] noisy speech and [64] X [1000] corresponding clean speech.so final .mat file will be [128] X [1000] dimension.



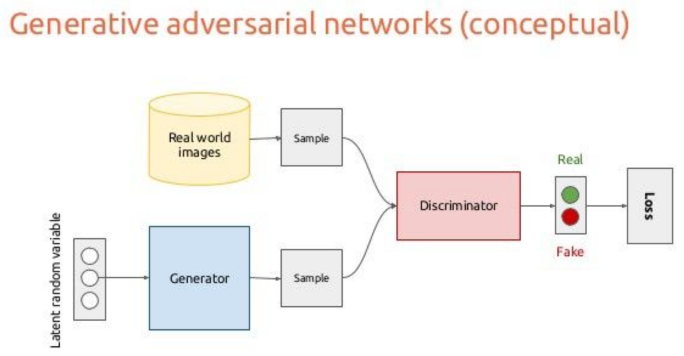
4.4 Deep Neural Network

Now I have got .mat files from extracted features so now I can use those files to train DNN which will give enhanced speech as output



4.4.1 GAN ARCHITECTURE

As I mentioned earlier also GAN is combination of two DNNs. In given time I was only able to implement Speech Enhancement system from single DNN. we can extend this to implement GAN, As it uses two DNN namely Generator and Discriminator. So all this function that I have used to implement DNN can furthermore extended to implement GAN. But in given time I was only able to create DNN speech enhancement system.

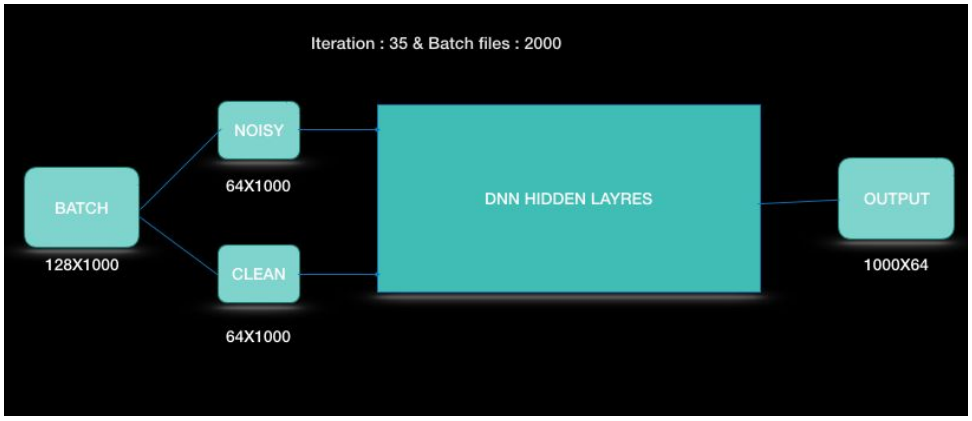


5

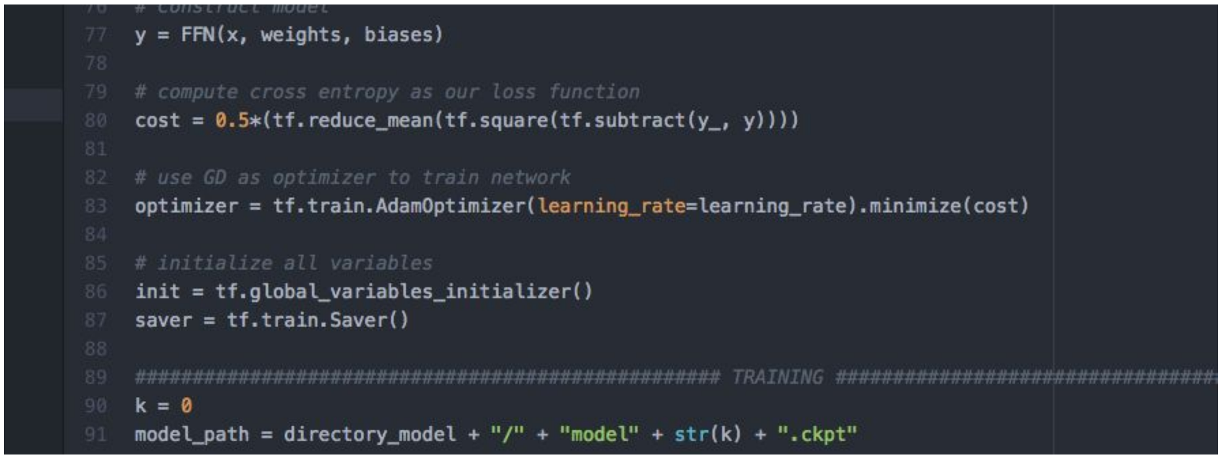
source

4.4.2 Training phase of DNN

Batches will be used to train DNN. I have implemented 3 layer DNN. I have given noisy and corresponding clean speech as input I have done 35 iteration for 2000 batches. For every iteration it will create corresponding weights and bias.



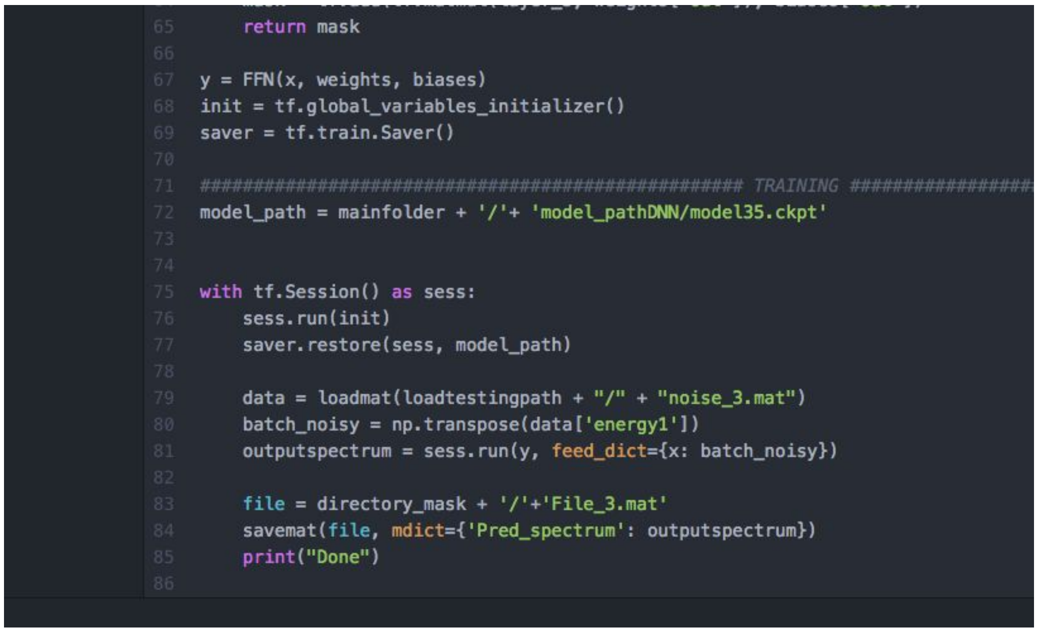
There is function optimizer in “testdnn” code which will use square subtraction method to optimize weight and bias in every iteration.Every iteration this equation will try to optimize loss function and corresponding details will be stored as model for every iteration. This model will be used to generate enhanced speech from noisy speech in training phase. (it took almost 2.5 hour to train DNN for 35 iteration)



```
# construct model
y = FFN(x, weights, biases)

# compute cross entropy as our loss function
cost = 0.5*(tf.reduce_mean(tf.square(tf.subtract(y_, y))))

# use GD as optimizer to train network
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate).minimize(cost)

# initialize all variables
init = tf.global_variables_initializer()
saver = tf.train.Saver()

#################################################### TRAINING ##################################
k = 0
model_path = directory_model + "/" + "model" + str(k) + ".ckpt"
```

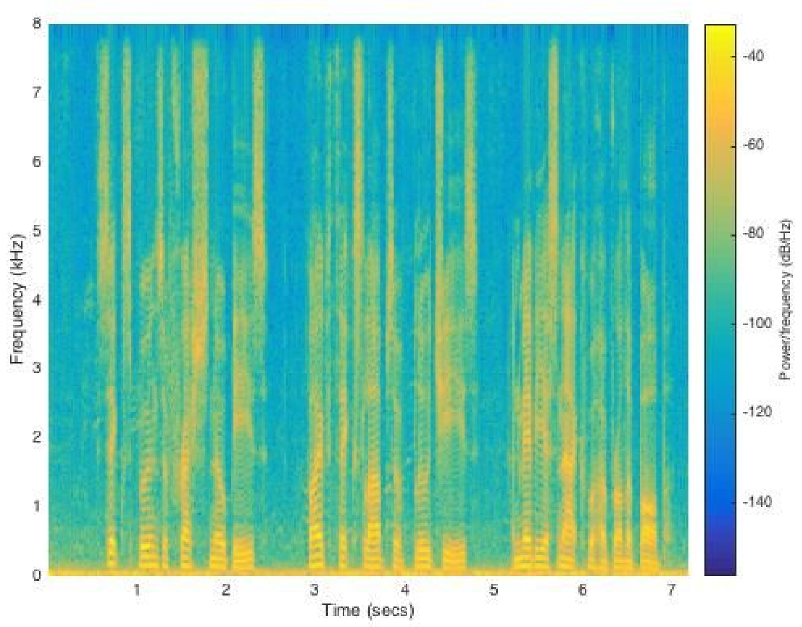
4.4.2 Testing Phase of DNN

Now I have got trained DNN,so now I can give testing wav file to DNN and DNN will generate enhanced speech using most last trained model (I have 35th model as I have done 35 iteration). Using this model it will generate “File_3.mat” which is our enhanced speech.

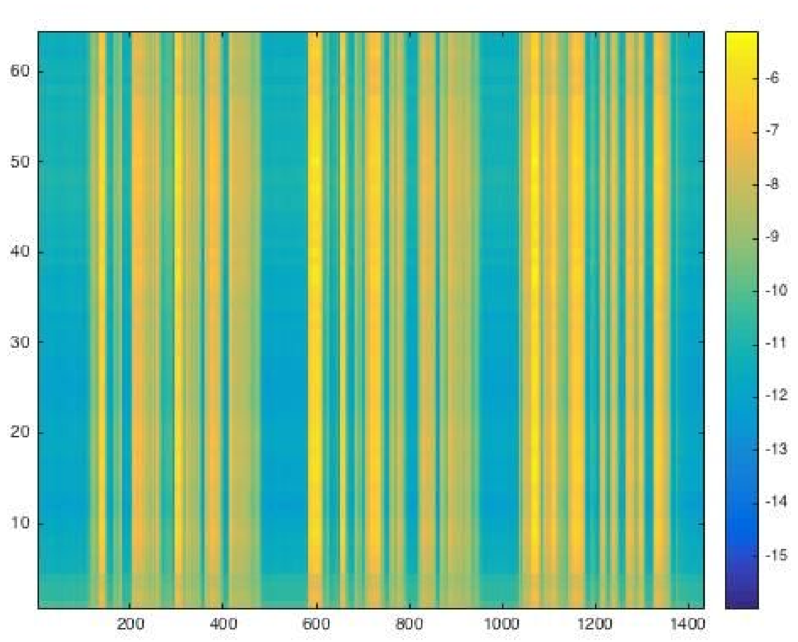


```
    return mask

y = FFN(x, weights, biases)
init = tf.global_variables_initializer()
saver = tf.train.Saver()

#################################################### TRAINING ################
model_path = mainfolder + '/'+ 'model_pathDNN/model35.ckpt'


with tf.Session() as sess:
    sess.run(init)
    saver.restore(sess, model_path)

    data = loadmat(loadtestingpath + "/" + "noise_3.mat")
    batch_noisy = np.transpose(data['energy1'])
    outputspectrum = sess.run(y, feed_dict={x: batch_noisy})

    file = directory_mask + '/'+'File_3.mat'
    savemat(file, mdict={'Pred_spectrum': outputspectrum})
    print("Done")
```

4.5 Results



Input file spectrogram



Enhanced output spectrogram

4.6 Analysis

As we can see from this result that noise in this speech is reduced but many information is also lost. This is because of less training of DNN we can give more training data and also we can increase iteration in training phase which will result in more trained DNN. And from that we can have output signal with less noise and enhanced quality. But just to train 4000 speech with 30 iteration took almost 2.5 hour to train and for feature extraction it took almost 2 hour time. so it is using high computing power,electricity and time to improve quality of speech.

References

1. Generative Adversarial Networks: Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yosh
2. SEGAN: Speech Enhancement Generative Adversarial Network Santiago Pascual, Antonio Bonafonte, Joan Serrà
3. A Guide to TF Layers: Building a Convolutional Neural Network
4. GAN: A Beginner’s Guide to Generative Adversarial Networks
5. Convolutional Neural Network with TensorFlow implementation
6. TECHNIQUES FOR FEATURE EXTRACTION IN SPEECH RECOGNITION SYSTEM : A COMPARATIVE STUDY Urmila Shrawankar
7. An Efficient Implementation of Gammatone Filters
8. GAMMATONE AND MFCC FEATURES IN SPEAKER RECOGNITION
9. Speech Enhancement In Multiple-Noise Conditions using Deep Neural Networks Anurag Kumar, Dinei Florencio

Appendix

Code for training DNN:

```
import os
os.environ['TF_CPP_MIN_LOG_LEVEL']='2'
import tensorflow as tf
import numpy as np
import math
import matplotlib.pyplot as plt
import random
import scipy
from scipy import io as sio
from scipy.io import loadmat
from scipy.io import savemat

mainfolder = "/Users/kaushik/Documents/sppech_dnn"

directory_mask = mainfolder+'/'+'MASKSDNN'
if not os.path.exists(directory_mask):
        os.makedirs(directory_mask)

directory_model = mainfolder+'/'+'model_pathDNN'
if not os.path.exists(directory_model):
        os.makedirs(directory_model)

loadtrainingpath = mainfolder+'/'+'features' # training features

data = loadmat(loadtrainingpath + "/" + "Batch_0.mat")
ip = np.transpose(data['noisy'])             # incoming total context of 11
op = np.transpose(np.log(data['clean'])) # b_s*64, # log-true labels

n_input = ip.shape[1]
print ("Input :" + str(n_input))
n_hidden1 = 512
n_hidden2 = 512
n_hidden3 = 512
n_output = op.shape[1]
print ("Input :" + str(n_output))
learning_rate = 0.0001

training_epochs = 35
training_batches = 2000

# Placeholders
x = tf.placeholder(tf.float32, [None,n_input], name="noisybatch") # b_s x ip
y_ = tf.placeholder(tf.float32, [None,n_output], name="centralcleanframe") # b_s x op

# Create model
def FFN(x, weights, biases):
        layer_1 = tf.add(tf.matmul((x), weights['h1']), biases['b1'])
        layer_1 = tf.nn.relu(layer_1)

        layer_2 = tf.add(tf.matmul(layer_1, weights['h2']), biases['b2'])
        layer_2 = tf.nn.relu(layer_2)
        layer_3 = tf.add(tf.matmul(layer_2, weights['h3']), biases['b3'])
        layer_3 = tf.nn.relu(layer_3)

        mask = tf.add(tf.matmul(layer_3, weights['out']), biases['out'])
        return mask

weights = {
        'h1': tf.Variable(tf.random_normal([n_input, n_hidden1])*.0001),
        'h2': tf.Variable(tf.random_normal([n_hidden1, n_hidden2])*.0001),
        'h3': tf.Variable(tf.random_normal([n_hidden2, n_hidden3])*.0001),
        'out': tf.Variable(tf.random_normal([n_hidden3, n_output])*.0001)
}

biases = {
        'b1': tf.Variable(tf.random_normal([n_hidden1])*0.01),
        'b2': tf.Variable(tf.random_normal([n_hidden2])*0.01),
        'b3': tf.Variable(tf.random_normal([n_hidden3])*0.01),
        'out': tf.Variable(tf.random_normal([n_output]))
}

# Construct model
y = FFN(x, weights, biases)

# compute cross entropy as our loss function
cost = 0.5*(tf.reduce_mean(tf.square(tf.subtract(y_, y))))

# use GD as optimizer to train network
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate).minimize(cost)

# initialize all variables
init = tf.global_variables_initializer()
saver = tf.train.Saver()

##################################################### TRAINING
##################################################################
k = 0
model_path = directory_model + "/" + "model" + str(k) + ".ckpt"

with tf.Session() as sess:
        sess.run(init)
        save_path = saver.save(sess, model_path)
        for epoch in range(0,training_epochs):
        saver.restore(sess, model_path)

        Rand_files = np.random.permutation(training_batches)
        batch_index = 0;
        for batch in Rand_files:

        data = loadmat(loadtrainingpath + "/" + "Batch_" + str(batch)+".mat")
        batch_noisy = np.transpose(data['noisy']) #ip
        batch_cleancentral = np.transpose(np.log(data['clean'])) # label

        costs,_ = sess.run([cost,optimizer], feed_dict={x: batch_noisy, y_: batch_cleancentral})

        print ("Epoch: "+str(epoch)+" Batch_index: "+str(batch_index)+" Cost= "+str(costs))
        batch_index = batch_index+1
        k = k+1
        model_path = directory_model + "/" + "model" + str(k) + ".ckpt"
        save_path = saver.save(sess, model_path)
```

Code for testing DNN:

```
import os
os.environ['TF_CPP_MIN_LOG_LEVEL']='2'
import tensorflow as tf
import numpy as np
import math
import matplotlib.pyplot as plt
import random
import scipy
from scipy import io as sio
from scipy.io import loadmat
from scipy.io import savemat

mainfolder = '/Users/kaushik/Documents/sppech_dnn'

directory_mask = mainfolder+'/'+'MASKSDNN'
if not os.path.exists(directory_mask):
        os.makedirs(directory_mask)

loadtestingpath = mainfolder+'/'+'test_samples' # training features

data = loadmat(loadtestingpath + "/" + "noise_3.mat")
ip = np.transpose(data['energy1'])        # incoming total context of 11
op = 64

n_input = ip.shape[1]
n_hidden1 = 512
n_hidden2 = 512
n_hidden3 = 512
n_output = op
learning_rate = 0.0001


# Placeholders
x = tf.placeholder(tf.float32, [None,n_input], name="noisybatch") # b_s x ip
#y_ = tf.placeholder(tf.float32, [None,n_output], name="centralcleanframe") # b_s x op

weights = {
        'h1': tf.Variable(tf.random_normal([n_input, n_hidden1])*.0001),
        'h2': tf.Variable(tf.random_normal([n_hidden1, n_hidden2])*.0001),
        'h3': tf.Variable(tf.random_normal([n_hidden2, n_hidden3])*.0001),
        'out': tf.Variable(tf.random_normal([n_hidden3, n_output])*.0001)
}

biases = {
        'b1': tf.Variable(tf.random_normal([n_hidden1])*0.01),
        'b2': tf.Variable(tf.random_normal([n_hidden2])*0.01),
        'b3': tf.Variable(tf.random_normal([n_hidden3])*0.01),
        'out': tf.Variable(tf.random_normal([n_output]))
}

# Create model
def FFN(x, weights, biases):
        layer_1 = tf.add(tf.matmul((x), weights['h1']), biases['b1'])
        layer_1 = tf.nn.relu(layer_1)

        layer_2 = tf.add(tf.matmul(layer_1, weights['h2']), biases['b2'])
        layer_2 = tf.nn.relu(layer_2)
```

```
		layer_3 = tf.add(tf.matmul(layer_2, weights['h3']), biases['b3'])
		layer_3 = tf.nn.relu(layer_3)

		mask = tf.add(tf.matmul(layer_3, weights['out']), biases['out'])
		return mask

y = FFN(x, weights, biases)
init = tf.global_variables_initializer()
saver = tf.train.Saver()

#################################################### TRAINING
################################################################
model_path = mainfolder + '/'+ 'model_pathDNN/model35.ckpt'

with tf.Session() as sess:
		sess.run(init)
		saver.restore(sess, model_path)

		data = loadmat(loadtestingpath + "/" + "noise_3.mat")
		batch_noisy = np.transpose(data['energy1'])
		outputspectrum = sess.run(y, feed_dict={x: batch_noisy})

		file = directory_mask + '/'+'File_3.mat'
		savemat(file, mdict={'Pred_spectrum': outputspectrum})
		print("Done")
```