Speech Diarization using Deep Neural Networks and Machine Learning

Clint Ferrin, Madi Mickelsen, Chui Vanfleet, Daniel Mortensen

December 13, 2017

Contents

List of Figures

1 Introduction

"Audio diarization is the process of annotating an input audio channel with information that attributes (possibly overlapping) temporal regions of signal energy to their specific sources" [1]. This report represents an effort to solve speaker diarization problem using deep learning. Many other methods of speech diarization rely heavily one signal processing techniques, but fewer attempt to use neural networks [2, 3, 4].

Dr. Stephanie A. Borrie from the Human Interaction Lab at Utah State University has provided a number of audio files from her research. The audio files contain a recording of two individuals conversing. Each individual speaks into their own microphone. But, crosstalk from each speaker is picked up on the other microphone.

Along with the audio files, Dr. Borrie has provided TextGrid files which contain the diarization annotations. These TextGrid files were created by Dr. Borrie's students and are to be used as training and testing data for a deep learning algorithm.

1.1 Problem

Using the audio and TextGrid files provided by Dr. Borrie, a deep learning algorithm must be developed to automatically perform speaker diarization

2 Requirements

2.1 Algorithm Requirements

1. The project must use a neural network at some point. If additional processing steps are used, such as in preparation of the data, or organizing the output response, that is also fine.
2. The training code should provide some indication of how the training is progressing.
3. The code should write a file indicating the diarization.

2.2 Report Requirements

1. Clearly describe the architecture used in processing.
2. Clearly describe the neural architecture, such as number of layers, neurons in each layer, type of neuron, number of weights, stride, etc.
3. Clearly describe what the inputs are.
4. Clearly describe how many outputs there are, and what the output layer function is (sigmoidal, softmax, linear, etc.).

5. Clearly describe the cost function the neuron is training against, and the kind of optimization used.

6. Clearly describe any other processing steps used in conjunction with the neural net.

7. Describe how to train the neural net — how it is presented input data, how the output data is presented.

8. Present data, preferably learning curves, demonstrating the neural network is learning. Indicate how many training iterations used, and why.

9. Present results quantifying how well the diarizer works. Ideally, this would be compared to the original training data. Provide a description of how the score is derived.

10. Provide a description of what was learned.

11. Provide a printout of the code.

3 Methods

Several different diarization methods were explored to meet the requirements discussed in subsection 2.1. This section will discuss, in detail, the several diarization methods created, presented in no particular order.

3.1 Fully Connected Neural Network

The fully connected version of the Neural Network can be seen in Figure 1. The Network has 5 layers, it receives a total of 3400 inputs (1600 inputs from channel 1, 1600 inputs from channel 2, 100 inputs from the `fft` of channel 1, and 100 inputs from the `fft` of channel 2), and has four possible outputs (00-no speech, 01-speaker one, 10-speaker two, and 11-both speakers).

The input layer is stacked horizontally with channel 1 sampled at 8000 Hertz for a total length of 0.2 seconds, channel 2 sampled at 8000 Hertz, and finally both channels are followed by their corresponding `fft` results with a size of 100 each. This yields a total input size of 3600; each new sample of training or testing data is stacked horizontally on top off each input sample to be fed into the network—the stacking of input data in matrix form allows TensorFlow to speed up the training process.

In order from left to right, each layer has 300, 200, 200, and 500 neurons. All of the hidden layers utilize ReLU activation functions, and the densely connected layers 2 through 4 have a dropout rate of 0.8. The output layer uses the softmax activation function because there are 4 different classes that the loss function needs to categorize.

To train the network, a file path is given to the main function. The number of `.wav` files in the audio files directory must match the number of `.TextGrid` files in the labels

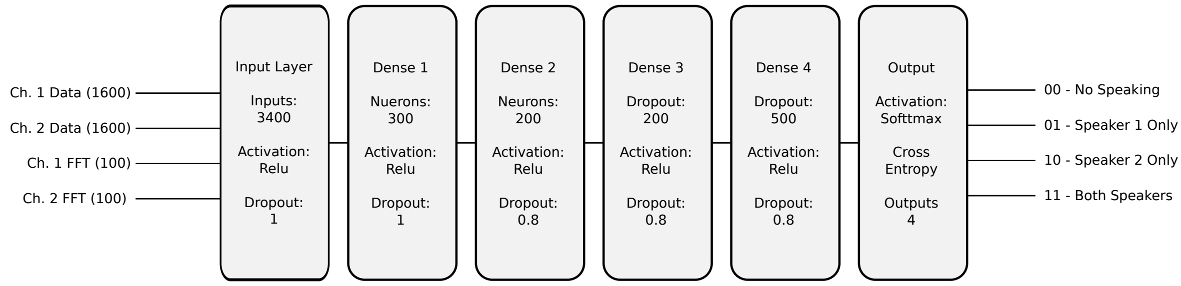

Figure 1: Block Diagram of Neural Network

directory. The label **numpy** array is created by sampling the **TextGrid** file to create an output list that corresponds to every Fs (samples per second) in the audio file.

The cost function utilized in the network design is cross-entropy, and the output of the network is compared to a one-hot-encoded **Y** vector that contains the actual class that was marked by humans. The optimization function that performed best with this setup was the **AdamOptimizer** that is included in the TensorFlow parameters. For more details reference the TensorBoard printout in Figure 2

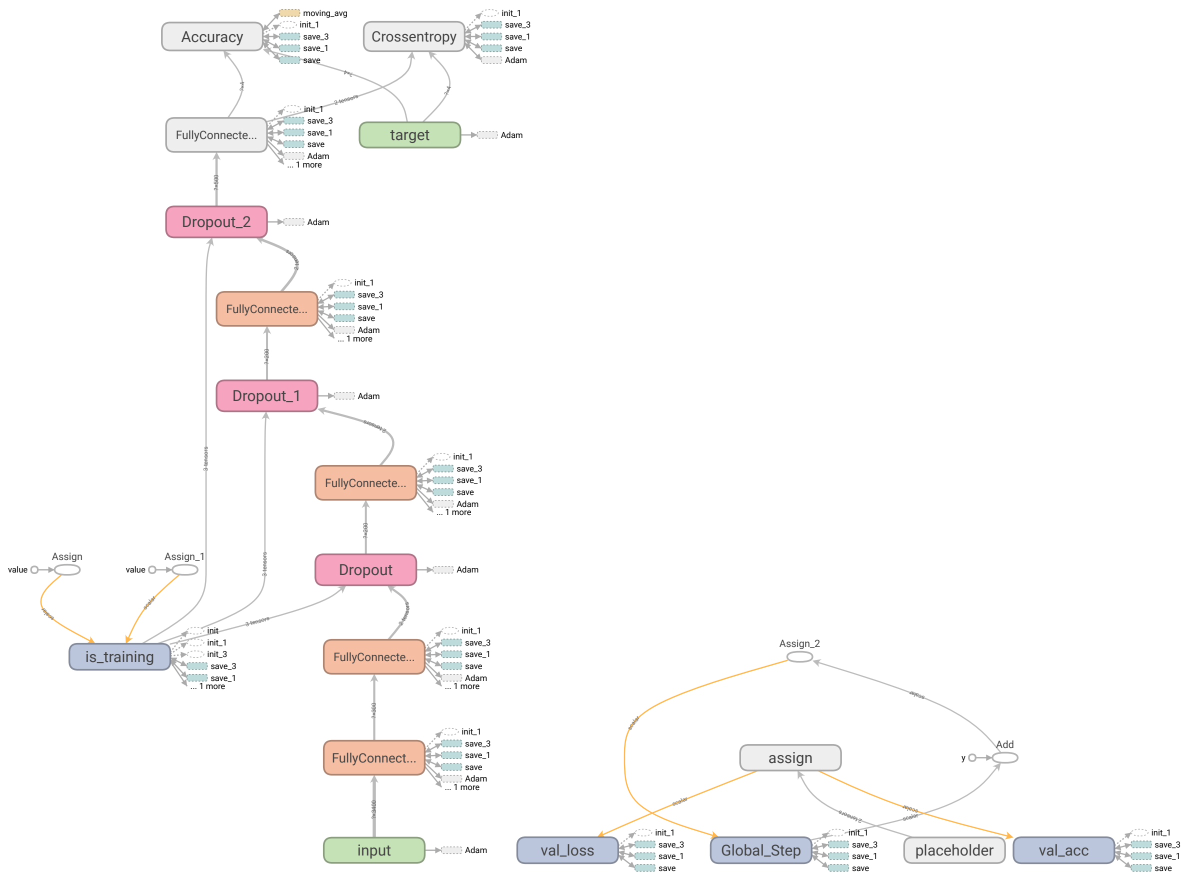

Figure 2: Fully Connected Neural Network

While implementing the network, different numbers of neurons, dropout rates, and network types were experimented with. The 1D convolutional network did not produce better results, and if the dropout rate was too high, the loss function would not converge. The parameters were changed and re-run to identify the best network system.

Introducing the FFT significantly improved the accuracy of the network. Before the FFT, the network had a very difficult time identifying silence, but the added information from the FFT improved the overall performance. For the results of this method, see Section 4

3.2 Blind Source Separation

The original design included a preprocessing portion using blind source separation (BSS). The goal of BSS, also known as independent component analysis, is to determine a matrix which forces the elements of the output to be statistically independent. The matrix is determined using an iterative process of maximum likelihood independent component analysis. When multiplied by the output, each element should be separated from the others.

For the purpose of this project, BSS would be used to separate the two speakers when speech overlap occurs, removing crosstalk. The BSS program was developed for another class using MATLAB. Before implementing BSS into the design fully (converting the MATLAB program into python), segments of an audio file with overlapped speech were run through the BSS program developed in MATLAB. This would test the BSS method to determine if it would add, or detract from the system.

Initially, the program would not separate the signals whatsoever. The audio appeared to be of worse quality, with more crosstalk rather than less. More iterations were run, the step size was changed, and more segments were concatenated to see if more data worked, but none of these possible *improvements* did not appear to change anything.

The reason BSS did not work for this instance could be due to the delay between microphones. For example, speaker one speaks directly into microphone one, whereas speaker two speaks directly into microphone two. The distance between the speakers would cause a delay to occur between speaking into one microphone and the crosstalk on the second. The BSS algorithm used in the program does not account for the delay. As a result this method was not incorporated into the final design of the system.

3.3 Transfer Function Estimation

Based on previous results, the following is a writeup of the next step that is predicted to create better results then the previous. In past attempts, windows of data were inserted into a neural network, along with their corresponding FFTs, to determine where changes in speaking would occur. This was complicated by microphone recording speech signals from several participants at once. This section goes over a method designed to show which microphone is being used during a given time interval.

3.3.1 Methodology

This method operates under the assumption that the input to a single microphone consists of the input from the nearest speaker and the output from the second speaker after it has been run through a transfer function $H(Z)$ as seen in Figure 3. This portion

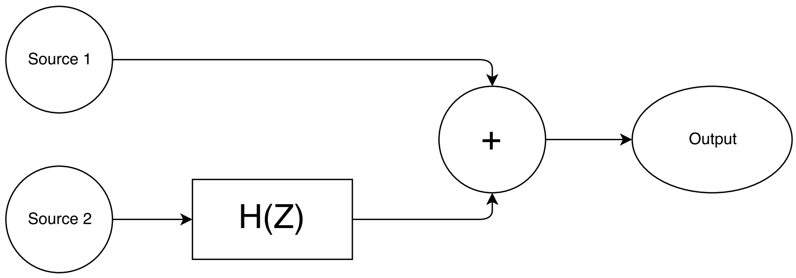

Figure 3: Overall System Diagram

of the speech diarization would be purposed towards detecting who is speaking: person one, person two, or both. For intents of the following setup, it is assumed that one person is speaking at a time. This algorithm requires two channels, one for microphone one, and the other for microphone two. In this setup, if only one person is speaking, then one signal will represent the most accurate data from the source, and the other will represent the same data after it has passed through a system and been modified by the system's transfer function.

The proposed solution is to develop a neural network that would simulate the transfer function that operates on the non-dominant dataset. Once this is accomplished, channel A is run through the simulated transfer function and subtracted from the other channel. If the output is minimal, then the channel that was operated on matches its user. In other words, when person one's data is run through the transfer function, and subtracted from person two's data, if the output is close to zero, then person two was not talking, and person one provided input to the microphone and vice versa. Figure 4 demonstrates a system view if the input signal was pre-transformed. Figure 5 shows the

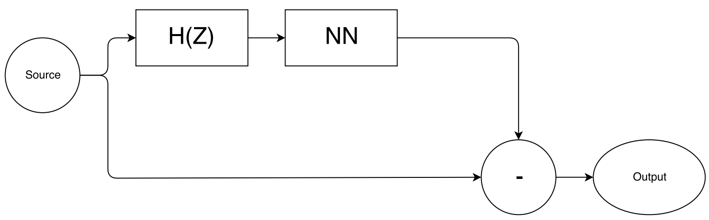

Figure 4: Non-Zero Error System

system view if the input signal is not passed through the transfer function, for example, it is person one's signal when person one was speaking. Channel two should then be run on input data. The results from both systems should be compared. If non-zero values

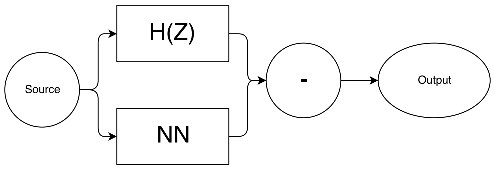

Figure 5: Zero Error System

are detected after running both channels through the transfer function, then both were talking. An example of the proposed system is shown in Figure 6. Both the original sig-

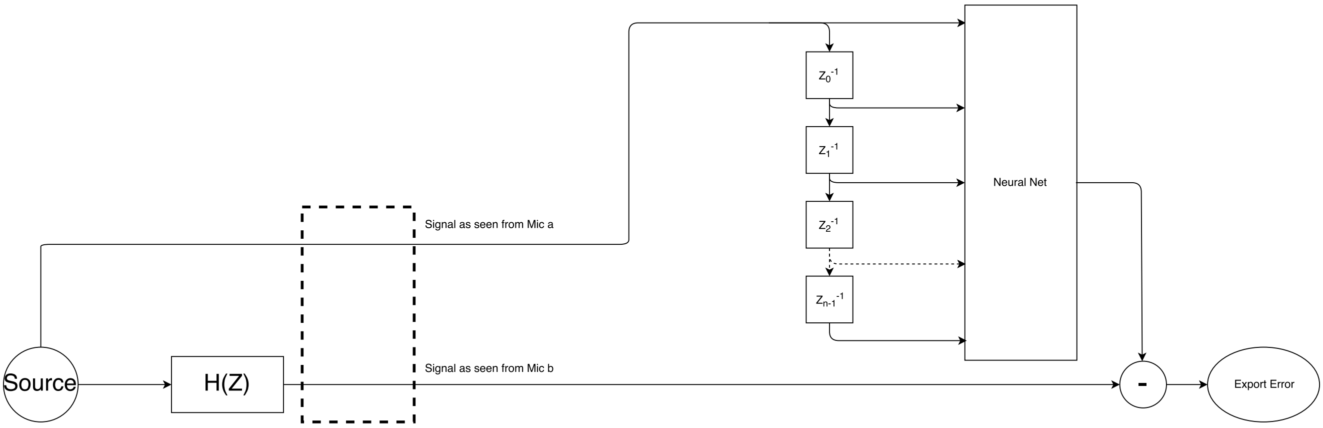

Figure 6: Neural Network System Design

.

nal is provided as input ,and the time delayed versions of the same signal. It is assumed when the signal undergoes a transformation, the transformed signal is consists of linear combinations of time delayed source samples.

3.3.2 System Parameters

The neural network will be comprised of a semi-connected layer where each neuron has access to a time delayed sample of the input data. The data is arranged as a Toeplitz matrix, A, with each row representing the input data of each neuron. The output of the network has equivalent dimensions as the output of the transfer function. There should be between 0 - 500 sample delays based on cross correlation, as shown in Figure 7. Stronger correlations between signals are seen from as far as $\tau = 500$ samples out.

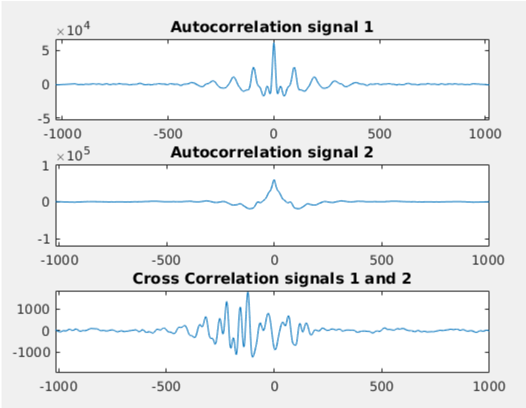

Figure 7: Cross Correlation Data

4 Results

Using the network described in Section 3.1, the network correctly labeled a file it had not seen with 75% accuracy. The training produced the following loss function graph after 25 epochs. 25 epochs was chosen, with batch sizes of 100 because larger batches or epochs cause the network start to decline in accuracy due to over-fitting. The loss plot in Figure 8 converges to a value; with more epochs, little progress is made in reducing the loss.

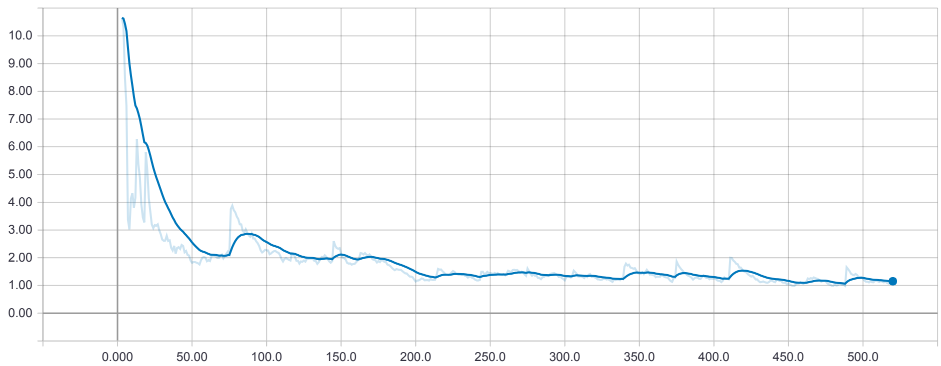

Figure 8: TensorBoard Loss Training Plot

The output of the accuracy plot can be seen in Figure 9. The accuracy was obtained by splitting the data into random training and testing sets of length 0.2 seconds. To create the training and testing data sets, all of the data was placed into a long array of data and shuffled in the same way as the labels. The data was down-sampled to a new rate of 8000 Hz, and the labels were sampled at 5 Hz. After the shuffling, the data was split into the proportion indicated by the variable `percent_train`.

As a result, the accuracy represents categorizing data that has not been trained by the Neural Network.

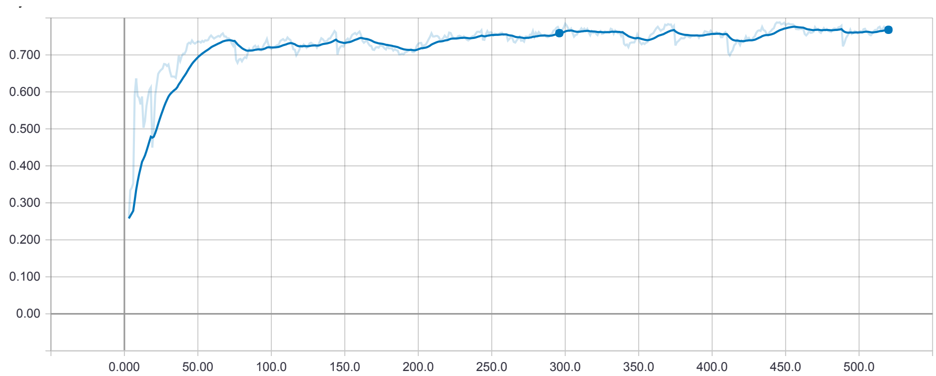

Figure 9: TensorBoard Accuracy Training Plot

It should be noted that the human markings were incorrect in many of the text grids. For example, the file `HS_D09.wav` seen in Figure 10 shows the first 10 seconds of a file that has been poorly marked. When listening to the file with both channels, it is obvious that no one is speaking for the majority of the 10 seconds. In that audio portions that people are speaking, the markings are incorrect. If the TextGrid files were more accurately labeled, the output of the programs would be more accurate as well.

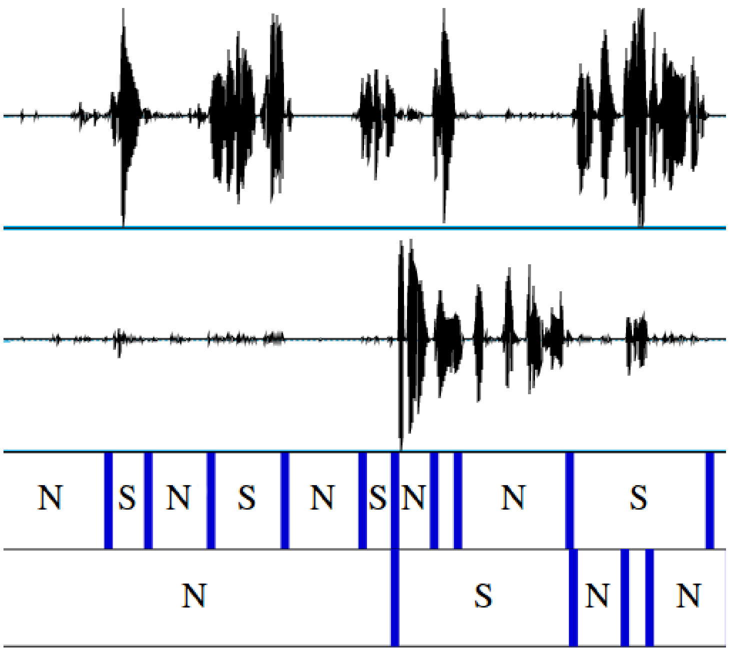

Figure 10: Example of Poor Diarization

To test the effectiveness of the training, the network was given a complete file of

4:00 minutes, and told to predict the labels. Despite the poorly marked training data, the network achieves 75% accuracy on an entire data file that the network had not seen previously. To test the accuracy, the diarization state was predicted every 0.2 seconds and compared to the markings on each tier of the TextGrid files. As a visual representation, the output after the network predicted the labels of the file was graphed. Figure 11 shows all of the necessary information to view how the network performed.

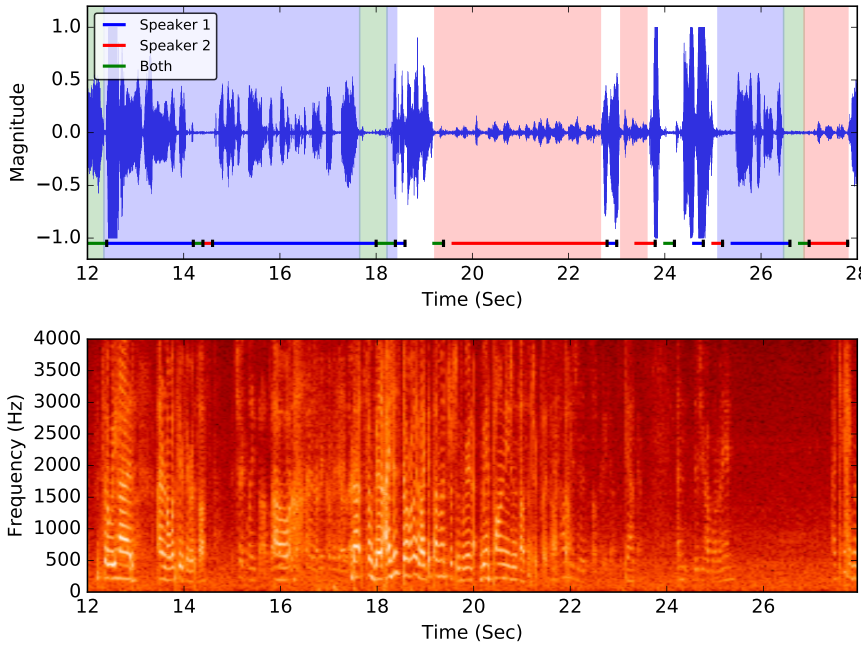

Figure 11: Results of Network on Untrained Data

The solid fill colors represent where the human labeled the speaking of both people. The areas with no color represents when neither person one or person two are speaking, blue fill represents when person one is speaking, red fill represents when person two is speaking, and green fill represents when both people are speaking at the same time.

The solid lines at the bottom of the graph are the predictions that the network produces. The predictions of the network line up closely to the human markings. In many cases, the neural network identifies small sounds that people make that are not recorded as speech in the TextGrid file (such as brief laughs). This can be filtered out at a future date to improve accuracy.

5 Conclusion

The fully connected neural network with the fft inputs was used to label audio and performed with 75% accuracy. The fully connected neural network was chosen as our final design because it yielded the best results. TensorFlow was a challenge to use by itself and it was difficult to quickly change network design, so using APIs like TFLearn and Keras made the process much easier.

Alternate methods to the speech diarization problem were discussed once the initial design was implemented and tested. These would be implemented in future iterations of this project to determine if they improved the accuracy.

The first alternate approach would use the Neural Network for the BSS. The NN should be able to incorporate the delay, making it more robust then the BSS algorithm used initially. This would allow easy separation of the two channels. After filtering out the noise, the speech on each channel would be easily detectable using the power of the signal.

Section 3.3 discussed an another method of approaching the problem, estimating the transfer function. The transfer function estimation method would help to identify when each microphone is being used but may prove difficult to implement depending on the different recording environments.

References

[1] S. E. Tranter and D. A. Reynolds. “An overview of automatic speaker diarization systems”. In: *IEEE Transactions on Audio, Speech, and Language Processing* 14.5 (Sept. 2006), pp. 1557–1565. ISSN: 1558-7916. DOI: 10.1109/TASL.2006.878256.

[2] K. Boakye et al. “Overlapped speech detection for improved speaker diarization in multiparty meetings”. In: *2008 IEEE International Conference on Acoustics, Speech and Signal Processing*. Mar. 2008, pp. 4353–4356. DOI: 10.1109/ICASSP.2008.4518619.

[3] X. Anguera et al. “Speaker Diarization: A Review of Recent Research”. In: *IEEE Transactions on Audio, Speech, and Language Processing* 20.2 (Feb. 2012), pp. 356–370. ISSN: 1558-7916. DOI: 10.1109/TASL.2011.2125954.

[4] C. Barras et al. “Multistage speaker diarization of broadcast news”. In: *IEEE Transactions on Audio, Speech, and Language Processing* 14.5 (Sept. 2006), pp. 1505–1512. ISSN: 1558-7916. DOI: 10.1109/TASL.2006.878261.

A Code Listings

A.1 main.py

```
import tensorflow as tf

from scipy.io import wavfile
from textgrid import TextGrid
from diarization_methods.TfLearnNN import TfLearnNN
from os import walk
from os.path import splitext
from numpy import array, vstack, hstack, zeros, ceil
from numpy.random import shuffle, randint
import time

audio_sample_rate = 8000

# Parameters to be set in some more elegant way later on.
 audio_files_path = '../media/Full_Test_Files_8000/'
text_grids_path = '../media/Full_Test_TextGrids/'
load_all_to_memory = True
window_size = int(audio_sample_rate * 0.2)
num_classes = 4
num_channels= 2
# training_flag = True
 training_flag = False
epochs = 50
batch_size = 100
percent_train = .8
params = {
    'num_channels' : 2,
     'window_size' : window_size,
     'num_classes' : 4,
            'nfft' : 100
}
diarization_method = TfLearnNN(params)
load_path = '../media/tflearn_nn.save'

# run_audio_file_path = '../media/Full Test Files 8000/HS_D01.wav'
# text_grid_load_file_path = '../media/Full Test TextGrids/HS_D01.TextGrid'
# text_grids_save_file_path = '../media/TexGrid Output/'

# save_path = '../media/tflearn_nn_2.save'
save_path = None
# load_path = None
load_path = '../media/tflearn_nn.save'

# End parameters section


def get_file_list(directory):
    """
    Used to get all the file names in the directory specified.
    :param directory: the path to the directory to get the file names from (ending in '/'
    :return: a list of all files names in directory starting with directory
    """
```

```python
    f = []
    for (dirpath, dirnames, filenames) in walk(directory):
        f = filenames
        break
    f.sort()
    for i in range(len(f)):
        f[i] = directory + f[i]
    return f


def get_label(path):
    """
    Used to load a TextGrid from disk and return the array representation of it.
    :pram path: Path to the TextGrid
    :return: Array representation of the TextGrid.
    """
    grid = TextGrid(name=path)
    grid.read(path, Fs=int(audio_sample_rate / window_size))
    return grid.FsArrayCombined


class Trainer:
    def __init__(self):
        self.data_paths = None
        self.label_paths = None
        self.training_data = None
        self.training_labels = None
        self.testing_data = None
        self.testing_labels = None
        self.imax = -1

        self.data_paths = array([get_file_list(audio_files_path)])
        self.label_paths = array([get_file_list(text_grids_path)])
        if len(self.data_paths) != len(self.label_paths):
            print('Error! Data and labels do not match up!')
            ValueError()

        self.split_train_test()

        if load_all_to_memory:
            print('Loading Data...', end='', flush=True)
            self.load_all_data_to_memory()
            print('Done!')
        else:
            print("Training is going to take a bit longer without loading all data first..")

    def get_chunk(self, is_train=True):
        data = self.training_data if is_train else self.testing_data
        labels = self.training_labels if is_train else self.testing_labels

        i = randint(0, len(data))
        x = data[i]  # should be a list at this point
        label = labels[i]

        if not load_all_to_memory:  # means that data is a path instead
            fs, x = wavfile.read(x[0])
            label = get_label(label[0])
        start = randint(0, len(label) - 1)
```

```python
        return x.T[:, start * window_size:start * window_size + window_size], label[start]

    def get_complete_file_chunk(self, is_train=True):
        data = self.training_data if is_train else self.testing_data
        labels = self.training_labels if is_train else self.testing_labels

        i = randint(0, len(data))
        x = data[i]  # should be a list at this point
        label = labels[i]

        if not load_all_to_memory:  # means that data is a path instead
            fs, x = wavfile.read(x[0])
            label = get_label(label[0])
        start = randint(0, len(label) - 1)
        return x.T[:, start * window_size:start * window_size + window_size], label[start]

    def load_all_data_to_memory(self):
        paths = array(self.training_data, copy=True)
        self.training_data = []
        for path in paths:
            fs, x = wavfile.read(path[0])
            self.training_data.append(x)

        paths = array(self.testing_data, copy=True)
        self.testing_data = []
        for path in paths:
            fs, x = wavfile.read(path[0])
            self.testing_data.append(x)

        paths = array(self.training_labels, copy=True)
        self.training_labels = []
        for path in paths:
            self.training_labels.append(get_label(path[0]))

        paths = array(self.testing_labels, copy=True)
        self.testing_labels = []
        for path in paths:
            self.testing_labels.append(get_label(path[0]))

    def split_train_test(self):
        shuf = vstack((self.data_paths, self.label_paths)).T
        shuffle(shuf)

        self.imax = i_max = int(len(self.data_paths[0]) * percent_train) + 1
        self.training_data = shuf[0:i_max, [0]]
        self.training_labels = shuf[0:i_max, [1]]
        self.testing_data = shuf[i_max - 1:-1, [0]]
        self.testing_labels = shuf[i_max - 1:-1, [1]]
        # At this point the training and testing sets contain a list of paths to data and labels.

    def train(self):
        try:
            if load_path is not None:
                print('Loading Diarization')
                diarization_method.load(load_path)


            X = array([], dtype=float).reshape(0,window_size)
```

```
            Y = array([], dtype=int).reshape(0,1)
            for i in range(epochs):
                for j in range(batch_size):
                    x, label = self.get_chunk()
                    X = vstack((X,x))
                    Y = vstack((Y,label))
                diarization_method.train_on_data(X, Y)

            print('Finished Training... ')
            print('Testing... ')

            # Testing section
            l = len(self.testing_data)
            for i in range(l):
                x, label = self.get_chunk(is_train=False)
                X = vstack((X,x))
                Y = vstack((Y,label))
            num_error = diarization_method.get_train_error(X, Y)
            num_train = Y.shape[0]
            training_accuracy = (num_train-num_error)/num_train*100
            print("%d/%d Mistakes. Training Accuracy: %.2f%%"%(int(num_error),num_train,
                training_accuracy))

        except Exception as ex:
            print("ERROR! And exception Occurred!")
            print(ex)
        except KeyboardInterrupt:
            print("\n Interrupted..")
        finally:
            if save_path is not None:
                print('Saving Diarization to %s...'%(save_path))
                diarization_method.save(save_path)


class Predictor:
    def __init__(self):
        self.data_paths = run_audio_file_path
        self.label_paths = None
        self.data = []
        self.labels = []
        self.imax = -1
        self.textgrid = None

        if text_grids_path is not None:
            self.labels = get_label( text_grid_load_file_path )[1:-1]

        fs, x = wavfile.read(self.data_paths)
        X0 = self.create_data_chunks(x[:,0], window_size,window_size)
        X1 = self.create_data_chunks(x[:,1], window_size,window_size)
        self.data = hstack((X0,X1))

    def create_data_chunks(self,x,window_size,step_size):
        window_size = int(window_size)
        step_size = int(step_size)

        # drop elements outside window
        if int(len(x)%(step_size)) != 0:
            x = x[:-int(len(x)%(step_size))]
```

```python

        # ending
        data_nb = int(ceil((len(x) - window_size)/step_size))

        start=0
        end=window_size
        data = zeros([data_nb,window_size])

        for i in range(data_nb):
            data[i,:] = x[start:end]
            start=start+step_size
            end=end+step_size

        return data

    def run(self):
        try:
            if load_path is not None:
                print('Loading Diarization')
                diarization_method.load(load_path)

            # diarization_method.run_on_data(self.data_paths)
            num_error = diarization_method.get_train_error(self.data, self.labels)
            num_train = self.labels.shape[0]
            training_accuracy = (num_train-num_error)/num_train*100
            print("%d/%d Mistakes. Training Accuracy: %.2f%%"%(int(num_error),num_train, training_accuracy))

        except Exception as ex:
            print("ERROR! and exception Occurred!")
            print(ex)
        except KeyboardInterrupt:
            print("\n Interrupted..")
        finally:
            print("Done")


if __name__ == '__main__':
    # TODO: command line argument parsing. Use are parse
    if training_flag is True:
        trainer = Trainer()
        trainer.train()
    else:
        predictor = Predictor()
        predictor.run()
```

A.2 DiarizationBaseClass.py

```python
from abc import ABC, abstractmethod
import tensorflow as tf


class DiarizationBaseClass(ABC):
    """
    The DiarizationBaseClass provides the interfaces between custom diarization
    methods and the wrapper code. It provides a tensorflow session available as
    `self.sess` that is usable within the `train_on_data`, `get_train_error`, and
```

```
    `run_on_data` methods.

    You may add other variables needed within your own implementation. Not here
    please. You can pass parameters into your custom diarization method via the
    `params` init parameter. This can be an array, dictionary, or object. In
    other words, that is the only parameter you'll ever need. You can pack
    everything into it.
    """

    def __init__(self, params=None):
        """
         Initialization for the base diarization class.
        :param params: the tuple, dictonary, or object used to initialize the
                       diarization method.
        """
        self.tf_inititializer = tf.global_variables_initializer()
        self.sess = None
        self.init_diarization_method(params)
        pass

    @abstractmethod
    def init_diarization_method(self, params):
        """
        This method is called once to initialize the speech diarization algorithm.
        :param params: the tuple, dictonary, or object used to initialize the
                       diarization method.
        """
        pass

    @abstractmethod
    def train_on_data(self, data, label):
        """
        This method is meant to be called once per iteration during the training cycle.

        The tensorflow session is available as `self.sess`. tensorflow session initialization
        has already been taken care of with the `tf.global_variables_initializer()`

        :param data: This is the single set of data to be used by a single training
                     operation. This will change with each call of `train_on_data`.
        :param label: This is the textgrid label for this data sample.
        """
        pass

    @abstractmethod
    def run_on_data(self, data):
        """
        This method will be used to run the diarization method on a set of data and
        returns the TextGrid result.
        :param data: the audiofile to run the diarization on.
        :return: the TextGrid object.
        """
        pass

    @abstractmethod
    def get_train_error(self, test_data, test_label):
        """
        This method will be called at the end of each epoch to get the error of the
         diarization method.
```

```
        :param test_data: the data to use to evaluate the accuracy of the model.
        :param test_label: the label to use to evaluate the accuracy of the model.
        :return: The error of the diarization method.
        """

    @abstractmethod
    def load(self, path):
        """
        This method will be called when we want to load the diarization method from disk
        :param path: the path to the storage file that is to be loaded.
        :return: True on success, False on failure
        """
        pass

    @abstractmethod
    def save(self, path):
        """
        This method will be called when we want to save the diarization method to disk to
        be recalled later.
        :param path: the path to where the storage file should be put.
        :return: True on success, False on failure
        """
```

A.3 TFLearnNN.py

```
from diarization_methods.DiarizationBaseClass import DiarizationBaseClass
import tensorflow as tf
import tflearn
from tflearn.layers.core import input_data, dropout, fully_connected
from tflearn.layers.conv import conv_2d, max_pool_2d
from tflearn.layers.normalization import local_response_normalization
from tflearn.layers.estimator import regression
import time
import numpy as np

from numpy import array, zeros


class TfLearnNN(DiarizationBaseClass):
    def run_on_data(self, data):
        # create fft
        if len(test_data) != len(test_label):
            test_data = test_data.reshape(len(test_label),-1)

        Xfft = self.__get_fft(test_data)
        X = np.hstack((test_data,Xfft))

        # create one-hot label
        Y = self.label_to_one_hot(test_label, self.num_classes)

        yhat = self.model.predict(X)
        yhat = np.argmax(yhat, axis=1)

        return self.__validate_results(yhat, test_label)

    def label_to_one_hot(self, label, num_classes):
        """
```

```
        converts
        into
        :pram label:   [0, 3, 2, 1,  ...]
        :return:      [[1, 0, 0, 0,  ...],
                       [0, 0, 0, 1,  ...],
                       [0, 0, 1, 0,  ...],
                       [0, 1, 0, 0,  ...]]
        """
        y = np.eye(self.num_classes)[label.astype(int)].reshape(-1,num_classes)
        return y

    def init_diarization_method(self, params):
        self.window_size  = params['window_size']
        self.nfft         = params['nfft']
        self.num_channels = params['num_channels']
        self.num_classes  = params['num_classes']
        self.total_width  = (self.window_size + self.nfft)*self.num_channels

        input_layer = tflearn.input_data(shape=[None, self.total_width], name='input')
        dense1 = tflearn.fully_connected(input_layer, 300, activation='relu')

        dense2 = tflearn.fully_connected(dense1, 200, activation='relu')
        dropout2 = tflearn.dropout(dense2, 0.8)

        dense3 = tflearn.fully_connected(dropout2, 200, activation='relu')
        dropout3 = tflearn.dropout(dense3, 0.8)

        dense4 = tflearn.fully_connected(dropout3, 500, activation='relu')
        dropout4 = tflearn.dropout(dense4, 0.8)

        softmax = tflearn.fully_connected(dropout4, 4, activation='softmax')

        self.net = tflearn.regression(softmax, loss='categorical_crossentropy',name='target')
        self.model = tflearn.DNN(self.net, tensorboard_verbose=2)

    def train_on_data(self, data, label):
        if len(data) != len(label):
            data = data.reshape(len(label),-1)
            test_data = test_data.reshape(len(test_label),-1)

        # create fft
        Xfft = self.__get_fft(data)
        X = np.hstack((data,Xfft))

        # create one-hot label
        Y = self.label_to_one_hot(label, self.num_classes)

        self.model.fit(X, Y, n_epoch=1, show_metric=True, \
                       validation_set=(X, Y),
                       snapshot_epoch=False,run_id='TfLearnNNk')

    def get_train_error(self, test_data, test_label):
        # create fft
        if len(test_data) != len(test_label):
            test_data = test_data.reshape(len(test_label),-1)

        Xfft = self.__get_fft(test_data)
        X = np.hstack((test_data,Xfft))
```

```
        # create one-hot label
        Y = self.label_to_one_hot(test_label, self.num_classes)

        yhat = self.model.predict(X)
        yhat = np.argmax(yhat, axis=1)

        return self.__validate_results(yhat, test_label)

    def load(self, path):
        self.model.load(path)

    def save(self, path):
        self.model.save(path)

    def __validate_results(self, yhat, y):
        num_error = 0
        for i in range(len(yhat)):
            if yhat[i] != y[i]:
                num_error += 1
        return num_error

    def __get_fft(self, data):
        if self.num_channels is 2:
            Xfft0 = abs(np.fft.fft(data[:,:self.window_size], self.nfft))
            Xfft1 = abs(np.fft.fft(data[:, self.window_size:], self.nfft))
            Xfft = np.hstack((Xfft0,Xfft1))
        else:
            Xfft = abs(np.fft.fft(data[:, self.window_size:], self.nfft))
        return Xfft
```

A.4 plot.py

```
import matplotlib.pyplot as plt
import numpy as np
from scipy.io import wavfile
import scipy.fftpack as fftpack
from scipy import signal
import sounddevice as sd
import matplotlib.animation as animation
from matplotlib import rcParams

rcParams.update({'figure.autolayout': True})


def track_audio(fs, start=0, end="end"):
    Y_MIN = -2
    Y_MAX = 2

    x = np.arange(start, end + 1, 0.01);

    def update_line(num, line):
        i = x[num]
        line.set_data([i, i], [Y_MIN, Y_MAX])
        return line,

    l, _ = plt.plot(start, -1, end, 1, linewidth=2, color='red')
```

```python
    line_anim = animation.FuncAnimation(fig, update_line, len(x), fargs=(l,), interval=1 / fs, blit=True, repeat=False)
    plt.show()


def plot_spectrogram(x, fs, start=0, end="end"):
    if end is "end":
        Pxx, freqs, bins, im = plt.specgram(x[start * fs:-1],
                                            NFFT=512, Fs=fs, noverlap=100, cmap=plt.cm.gist_heat)
        end = max(bins)

    else:
        Pxx, freqs, bins, im = plt.specgram(x[start * fs:start * fs + end * fs],
                                            NFFT=512, Fs=fs, noverlap=100, cmap=plt.cm.gist_heat)
        if end > max(bins):
            end = max(bins)

    plt.ylim(0, max(freqs))
    plt.xlim([start, end])
    plt.ylabel("Frequency (Hz)")
    plt.xlabel("Time (Sec)")
    # plt.colorbar()


def play_audio(data, fs, start=0, end="end", blocking=True):
    if end is "end":
        sd.play(data[int(start * fs):-1], fs, blocking=blocking)
    else:
        sd.play(data[start * fs:int(start * fs + end * fs)], fs, blocking=blocking)


def plot_audio(x, fs, start=0, end="end"):
    if end is "end":
        xaxis = np.linspace(start, len(x) / fs, num=len(x[start * fs:]))
        plt.plot(xaxis, x[start * fs:] / float(max(x)), linewidth=0.25)
        end = len(x) / float(fs)

    else:
        if end > len(x) / fs:
            end = len(x) / fs
        xaxis = np.linspace(start, end, num=len(x[int(start * fs):int(end * fs)]))
        plt.plot(xaxis, x[int(start * fs):int(end * fs)] / float(max(x)), color='#3030e0', linewidth=0.15)

    plt.ylim(-1.2, 1.2)
    plt.xlim([start, end])
    plt.ylabel("Magnitude")
    plt.xlabel("Time (Sec)")


def plot_bounds_lines(changes, marks, start=0, end="end"):
    x = changes
    if end is "end":
        last = x.shape[0] + 1
    else:
        last = np.argmax(x > end)
```

```python

    data_range = range(np.argmin(x < start), last)

    if max(marks) > 2:
        plt.plot((-1, -1), (-1, -1), 'b-', linewidth=2, label="Speaker 1")
        plt.plot((-1, -1), (-1, -1), 'r-', linewidth=2, label="Speaker 2")
        plt.plot((-1, -1), (-1, -1), 'g-', linewidth=2, label="Both")
    else:
        plt.plot((-1, -1), (-1, -1), 'b-', linewidth=2, label="Speech")

    plt.legend(loc=2, fancybox=True, framealpha=0.8, prop={'size': 9})

    for j, mark in enumerate(marks):
        if mark == 1:
            plt.plot((x[j], x[j + 1]), (-1.05, -1.05), 'b-', linewidth=2)
            plt.plot((x[j + 1], x[j + 1]), (-1.03, -1.07), 'k-', linewidth=2, zorder=10)

        elif mark == 2:
            plt.plot((x[j], x[j + 1]), (-1.05, -1.05), 'r-', linewidth=2)
            plt.plot((x[j + 1], x[j + 1]), (-1.03, -1.07), 'k-', linewidth=2, zorder=10)

        elif mark == 3:
            plt.plot((x[j], x[j + 1]), (-1.05, -1.05), 'g-', linewidth=2)
            plt.plot((x[j + 1], x[j + 1]), (-1.03, -1.07), 'k-', linewidth=2, zorder=10)

        else:
            pass

    plt.ylim(-1.2, 1.2)
    plt.xlim([start, end])
    plt.xlabel("Time (Sec)")


def plot_bounds_fill(changes, marks, start=0, end="end"):
    x = changes
    if end is "end":
        last = x.shape[0] + 1
    else:
        last = np.argmax(x >= end) + 1

    data_range = range(np.argmin(x < start), last)

    # create legend information
    # plt.axvspan(0, 0, alpha=0.5, color='b', label="\"N\" Human")
    # plt.axvspan(0, 0, alpha=0.5, color='r', label="\"S\" Human")
    # plt.axvspan(0, 0, alpha=0.5, color='g', label="\"S\" Human")
    plt.legend(loc=2, fancybox=True, framealpha=0.8, prop={'size': 9})

    for j, mark in enumerate(marks):
        if mark == 1:
            plt.axvspan(x[j], x[j + 1], alpha=0.2, color='b')

        elif mark == 2:
            plt.axvspan(x[j], x[j + 1], alpha=0.2, color='r')

        elif mark == 3:
            plt.axvspan(x[j], x[j + 1], alpha=0.2, color='g')

```

```
        else:
            pass


def plot_fft():
    N = 600 # sample points
    T = 1 / 800.0  # sample spacing
    x = np.linspace(0, N * T, N)
    y = np.sin(50.0 * 2.0 * np.pi * x) + 0.5 * np.sin(80.0 * 2.0 * np.pi * x)
    yf = fftpack.fft(y)
    xf = np.linspace(0.0, 1.0 / (2.0 * T), N / 2)
    fig, ax = plt.subplots()
    ax.plot(xf, 2.0 / N * np.abs(yf[:N // 2]))


if __name__ == '__main__':
    main()
```

A.5 textgrid.py

An existing TextGrid parser was used from `https://github.com/kylebgorman/textgrid/blob/master/textgrid/textgrid.py`. Three additional functions were added to assist in neural network training:

```
def get_bin_changes(x, Fs):
    FsTimeChanges = [0]
    FsChangeMarks = [x[0]]
    for i in range(len(x)):
        if x[i] != x[i - 1]:
            FsTimeChanges.append(i)
            FsChangeMarks.append(x[i])

    FsTimeChanges.append(i)
    FsTimeChanges = np.array(FsTimeChanges)*1/float(Fs)
    FsChangeMarks = np.array(FsChangeMarks)
    return FsTimeChanges, FsChangeMarks
```

```
class IntervalTier(object):
...
  def tier_to_array(self, Fs, maxTime):
    array = np.zeros(int(np.ceil(Fs * maxTime)))
    current_interval = 0
    for i in range(array.shape[0]):
      if i/float(Fs) < self[current_interval].maxTime:
        if self[current_interval].mark == 'N':
          array[i] = 1
        else:
      current_interval += 1

    self.FsArray = array
    self.FsTimeChanges, self.FsChangeMarks = get_bin_changes(array, Fs)
...
```

```
class TextGrid(object):
...

  def combine_grids(self):
```

```
        self.FsArrayCombined = self.tiers[0].FsArray + self.tiers[1].FsArray * 2
        self.FsTimeChangesCombined, self.FsChangeMarksCombined = get_bin_changes(self.
            FsArrayCombined, self.Fs)
...
```

```
class TextGrid(object):
...
    def read(self, f, round_digits=DEFAULT_TEXTGRID_PRECISION, Fs=None):
        """
        Read the tiers contained in the Praat-formatted TextGrid file
        indicated by string f. Times are rounded to the specified precision.
        """
        self.Fs = Fs
        encoding = detectEncoding(f)
        with codecs.open(f, 'r', encoding=encoding) as source:
            source.readline()  # header junk
            source.readline()  # header junk
            source.readline()  # header junk

            self.minTime = round(float(source.readline().split()[2]), round_digits)
            self.maxTime = round(float(source.readline().split()[2]), round_digits)
            source.readline()  # more header junk
            m = int(source.readline().rstrip().split()[2])  # will be self.n
            source.readline()
            for i in range(m):  # loop over grids
                source.readline()
                if source.readline().rstrip().split()[2] == '"IntervalTier"':
                    inam = source.readline().rstrip().split(' = ')[1].strip('"')
                    imin = round(float(source.readline().rstrip().split()[2]), round_digits)
                    imax = round(float(source.readline().rstrip().split()[2]), round_digits)
                    itie = IntervalTier(inam)
                    for j in range(int(source.readline().rstrip().split()[3])):
                        source.readline().rstrip().split()  # header junk
                        jmin = round(float(source.readline().rstrip().split()[2]), round_digits)
                        jmax = round(float(source.readline().rstrip().split()[2]), round_digits)
                        jmrk = _getMark(source)
                        if jmin < jmax:  # non-null
                            itie.addInterval(Interval(jmin, jmax, jmrk))
                    self.append(itie)
                else:  # pointTier
                    inam = source.readline().rstrip().split(' = ')[1].strip('"')
                    imin = round(float(source.readline().rstrip().split()[2]), round_digits)
                    imax = round(float(source.readline().rstrip().split()[2]), round_digits)
                    itie = PointTier(inam)
                    n = int(source.readline().rstrip().split()[3])
                    for j in range(n):
                        source.readline().rstrip()  # header junk
                        jtim = round(float(source.readline().rstrip().split()[2]), round_digits)
                        jmrk = _getMark(source)
                        itie.addPoint(Point(jtim, jmrk))
                    self.append(itie)

        if Fs != None:
            for tier in self.tiers:
                tier.tier_to_array(Fs, self.maxTime)
            if len(self.tiers) == 2:
                self.combine_grids()
...
```