令和元年度　卒業研究

# AI を用いた日本語音声合成の研究

函館工業高等専門学校

生産システム工学科　情報コース　５年

１１番　久米田羽月

指導教員　東海林智也

目次

第1章　序論
　　第1節　英文アブストラクト
　　第2節　研究背景
　　第3節　研究目的
　　第4節　開発環境
　　第5節　データセット

第2章　関連技術
　　第1節　Merlin
　　第2節　OpenJTalk
　　第3節　segmentation-kit
　　第4節　WORLD
　　第5節　SPTK
　　第6節　Melin の問題点

第3章　プログラムの開発
　　第1節　Merlin の設定
　　第2節　モデル
　　第3節　ラベル生成プログラム
　　第4節　任意テキストの音声合成プログラム

第4章　比較

第5章　結果

第6章　課題と考察

参考文献

付録
　　ソースコード

# 第1章　序論

## 第1節　　英文アブストラクト

Currently, there are several Text-to-Speech implementations that use deep learning, but the sound quality and intonation quality are not perfect and there is room for improvement. The purpose of this research is to develop a model that can obtain higher quality synthesized speech by improving the existing implementation.

Specifically, improve the network model part of Merlin, a speech synthesis framework written in Python3 and TensorFlow, and compare the speech output using the model before improvement with the speech output using the new model Check if the model change was effective.

Key words: Text-to-Speech, Deeplearning, TensorFlow, Keras

## 第2節　　研究背景

近年、人工知能技術は盛んに研究され、様々な分野で一定の成果を上げている。中でも生成分野では、本物の写真と見分けがつかない高精度な画像が作られるなど、その可能性に注目を集めている。また特に、Text-to-Speech などの音声合成は AI アシスタントなどの登場で需要が高まっていると思われ、多くの手法・実装が公開されている。

しかしながら、合成音声の音質やイントネーションの品質は完璧ではなく改善の余地があるのではないかと考えた。

## 第3節　　研究目的

本研究は既存の実装である音声合成フレームワーク Merlin[1]を改善することによって、より高品質な合成音声が得られるモデルの開発を目的とする。

# 第4節　開発環境

本研究で使用した開発環境は以下の通りである。

第1項　コンピュータスペック
OS：Linux Mint 19.3 64bit
CPU：AMD Ryzen7 2700X 4.3GHz
RAM：32GB + Swap32GB
GPU：MSI Radeon RX Vega 56 Air Boost 8G OC

第2項　開発言語
Python 3.6

第3項　依存ライブラリ
numpy = "==1.16.2"
scipy = "==1.1.0"
theano = "==0.8"
keras = "==2.0.5"
h5py = "==2.8.0"
scikit-learn = "*"
bandmat = "*"
matplotlib = "*"
tensorflow-rocm = "==1.13.3"
OpenJTalk

# 第5節　データセット

深層学習での音声合成を行うにあたって、データセットが必要となる。ここで言うデータセットとは、音声ファイルとそれに対応するテキストファイルである。

本研究では音声ファイルに、東京大学が公開している音声コーパスである JUST[2]の basic5000、対応するラベルファイルは jsut-lab[3]で公開されているものを使用した。

# 第2章　関連技術

## 第1節　　Merlin

Merlinとは、エディンバラ大学音声技術研究センターで開発されたニューラルネットワークベースの音声合成フレームワークであり、Python3 で記述されている。音響特徴量を用いた統計的パラメトリック音声合成方式を用いている。Merlinでは教師データに音声ファイル(.wav)と音素ラベルファイル(.lab)を用い、音響特徴量と言語特徴量を抽出する。それらを用いて音響・継続長モデルを学習し、音響特徴量と言語特徴量を生成する。

Merlin はフレームワークであるため、処理の流れがつかみにくい。以下の図は Merlin で学習から生成を行うまでのおおよその流れを示したものである。なお、継続長の学習と音響特徴の学習とはモデルは共通であるが、それぞれの学習は別々に行われる。

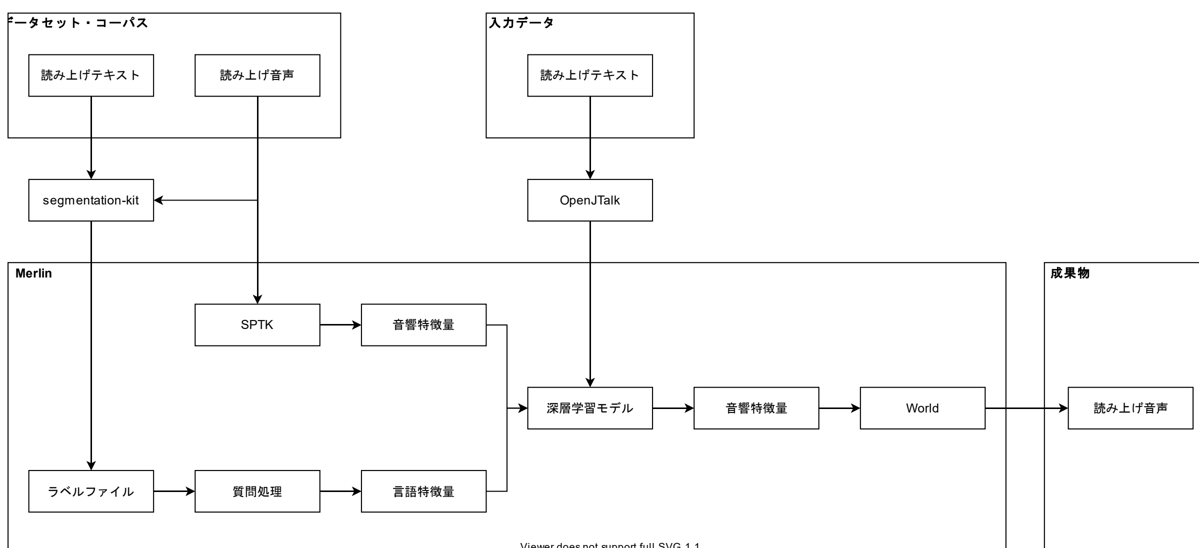


**図 1　Merlin での処理の流れ**

また、学習済みのモデルを用いて任意のテキストから音声を合成したい場合、図のように読み上げテキストから OpenJTalk[4]などを用いてラベルファイルを用意する必要がある。この処理については 2 章プログラムの開発で解説する。

Merlin では設定ファイルにより層の数や層の種類、層ごとのノード数等を変更することができるほか、バックエンドとなる DNN フレームワークの変更も行うことができる。

## 第2節　OpenJTalk

OpenJTalk は名古屋工業大学で開発されている、HMM 方式に基づいた日本語音声合成システムである。Merlin と同じく統計的パラメトリック合成方式であるが、OpenJTalk では隠れマルコフモデルでの実装、Merlin では隠れマルコフモデルにあたる部分を DNN で実現する方式をとっている。

本研究では、任意の音声を合成する際に必要となるラベルファイルを生成するのみに使用している。

## 第3節　segmentation-kit

データの学習には音声ファイルと、時間情報が含まれた対応するラベルファイルが必要である。segmentation-kit[5]はデータセットに含まれる読み上げテキストと、対応する音声ファイルを用いて音素ごとの時間情報が記述されたラベルファイルを生成するプログラムである。Julius[6]と呼ばれるオープンソースの日本語認識プログラムが使用されている。

今回はオープンソースで公開されているラベルファイルを用いたため使用していないが、データセットにラベルファイルが提供されていない場合や、独自のデータセットを用いる場合はこのプログラムを用いてラベルファイルを作成する必要がある。

## 第4節　WORLD

WORLD[7]は、山梨大学で開発されている、ボコーダーと呼ばれるプラグラムである。Merlin 内部では音響特徴量を抽出するほか、推論された音響特徴量から実際の音声ファイルを生成するために使用される。なお、Merlin ではメルケプストラム,F0, 有声・無声フラグ, 非周期成分を学習に用いている。

## 第5節　SPTK

SPTK[8]は東京工業大学で開発されている、音声ファイルから音響特徴量を抽出するためのソフトである。Merlin 内部では WORLD と合わせて特徴量を用いた音声の合成に使用される。

## 第6節　Merlinの問題点

Merlinではモデル構造の定義は設定ファイルのパラメータ調整によって行われるが、設定ファイルで設定できる構造には限度がある。具体的にはドロップアウトの割合の設定は可能だが、層ごとの詳細な設定は不可能である。

さらに、バックエンドとなるDNNフレームワークの変更も可能だが、フレームワークごとに実装が完全に分けられており、バックエンドにより利用可能な設定項目も限られる。例としてTheano[9]をバックエンドに使った場合ではドロップアウト層の追加を行えるが、Keras[10]を使用する場合ではドロップアウト層を任意の位置に追加することができない。

本研究では環境の都合上、バックエンドにKerasを選択したが、この場合ドロップアウト層を任意の位置に追加することができず、ドロップアウトの割合を設定することができない。

# 第3章　プログラムの開発

## 第1節　Merlinの設定

実験にあたり、Merlinの設定を行う。具体的に必要な作業は日本語音声用プロジェクトディレクトリの作成、バックエンドにKerasを使用するための設定の追記、学習時にTensorBoard[11]を使うためのMerlin本体のコードの書き換えである。また、Kerasのバックエンドには Tensorflow[12]をAMDのGPUで利用できるようにしたTensorflow-ROCm[13]を使用した。

Merlinではegs/に言語ごとにプロジェクトディレクトリが配置されている。そのため、このディレクトリにjapanese_voiceというディレクトリを配置する。

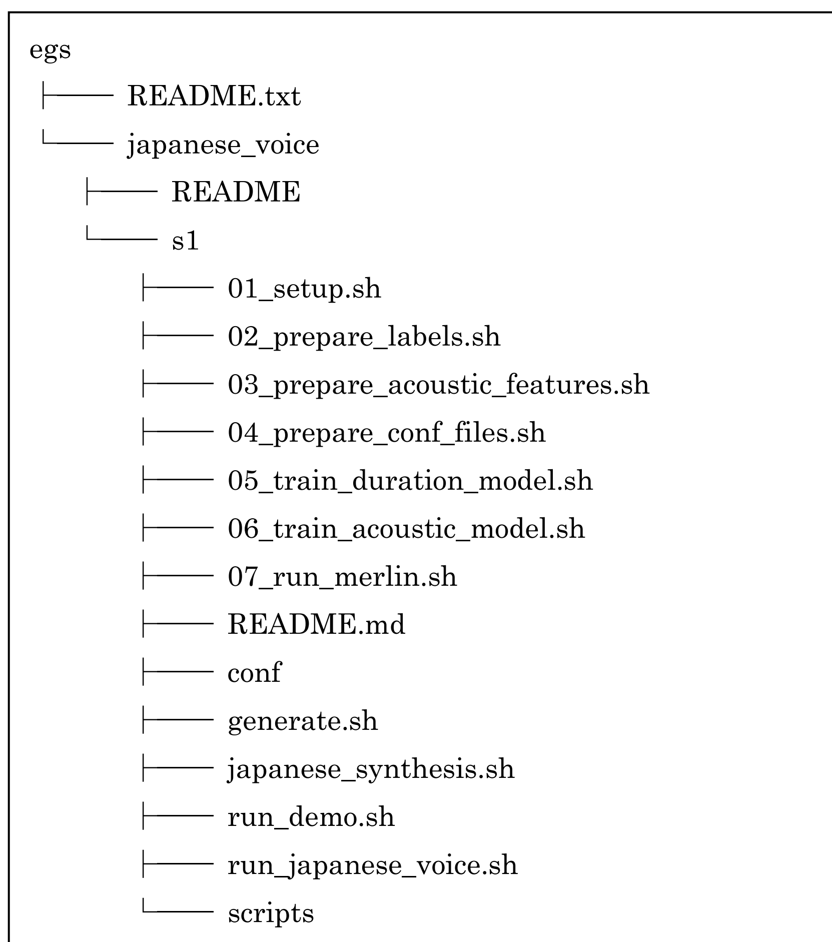


**図 2　egs/以下のディレクトリ構造（一部省略）**

こちらは付属しているmandarin_voiceをベースに、参考文献[14]に従って構成を行った。

次に Keras を使うための設定である。学習・生成に Keras を使用するには prepare_config_files.sh、prepare_config_files_for_synthesis.sh の 2 ファイルについて、それぞれの Architecture セクションに下記の設定を追記する。

```
########## duration config file #########
[Architecture]
$SED -i s#'switch_to_keras¥s*:.*'#'switch_to_keras: True'# $duration_config_file

########## acoustic config file ##########
[Architecture]
$SED -i s#'switch_to_keras¥s*:.*'#'switch_to_keras: True'# $acoustic_config_file
```

図 3　追記する設定項目（中略）

TensorBoard は学習途中の精度などを記録・閲覧することができるプログラムである。TensorBoard を使うために変更・追記したソースコードについては付録の学習用プログラムを参照されたい。このほか、追記した学習用プログラムの関数を実行するようにソースコードを変更した。

また、egs/japanese_voice/s1/run_demo.sh を実行することで学習を開始することができるようにしている。

## 第 2 節 モデル

本研究では、設定ファイルのみでは表現不可能なモデル構造を用いることで、品質の向上を期待する。具体的には、設定ファイルでも変更可能な隠れ層の数や種類、ノード数に加え、設定ファイルのみでは詳細な設定が不可能であったドロップアウト層[15]の追加を直接 Keras のコードを記述し、比較を行う。ドロップアウトとは、多層ニューラルネットワークにおいて、ある層のノードを一定の割合でランダムに無効化することで、汎化性能の向上と過学習の防止を目的とした仕組みである。以下の図はある層に対してドロップアウトを行った場合、一部のノードが無効になるところを表したものである。モデルや学習用プログラムについては付録を参照されたい。

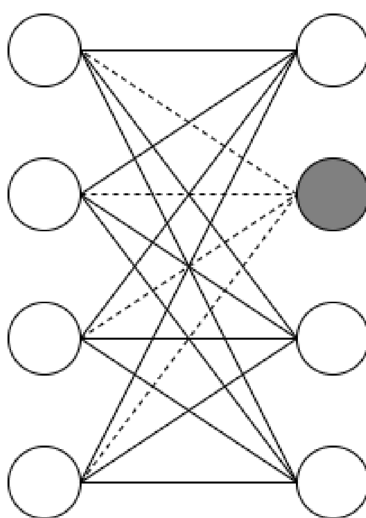

**図 4　ドロップアウト層の構造**

学習結果を比較するため、モデル構造を変更した複数のモデルを用意し、同じデータセットで学習させたのちにテストデータを出力した。モデルの条件を以下で示す。なお、以下に示すノード数は一層あたりのノード数である。

1. 基本設定（13 層、ノード数は 1024）
2. 独自設定・ドロップアウトなし（5 層、ノード数は 512）
3. 独自設定・ドロップアウトあり（5 層、ノード数は 512）

**共通の条件**

- 学習データ = 4800
- テストデータ = 200
- バッチサイズ = 32
- エポック数 = 25

### 基本設定のモデル構造

設定ファイルを基本設定にした場合に使用されるモデル構造である。ドロップアウトの割合は0である。そのためすべての層が全結合層である単純なモデルであるといえる。

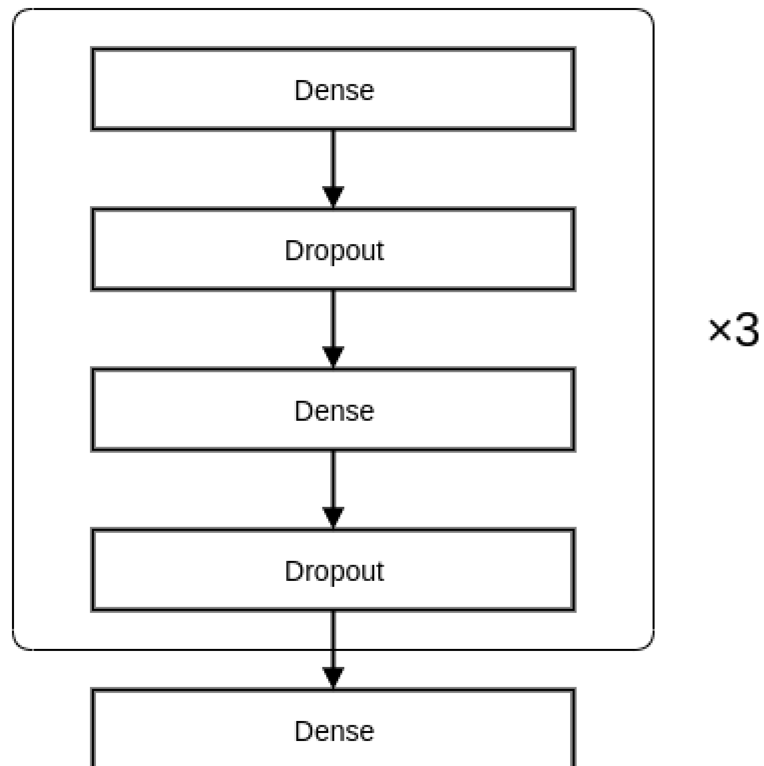


図 5　基本設定のモデル構造

### ドロップアウトなしのモデル構造

Keras を用いて直接記述を行ったモデルである。中間層に 1 つの LSTM 層がある 5 層の構造である。直接記述しても問題なく動作するかを確認するために作成したモデルであるため、この構造は設定ファイルからも再現可能である。

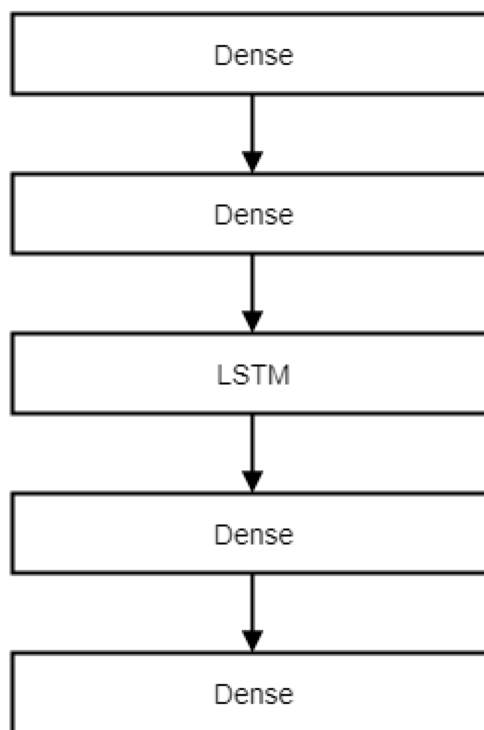


図 6　ドロップアウトなしのモデル構造

**ドロップアウトありのモデル構造**

ドロップアウトのないモデルの一部の全結合層をドロップアウト層に変更したモデルである。ドロップアウトの割合は上から 0.2, 0.5 である。設定ファイルのみではこの構造を再現することはできない。

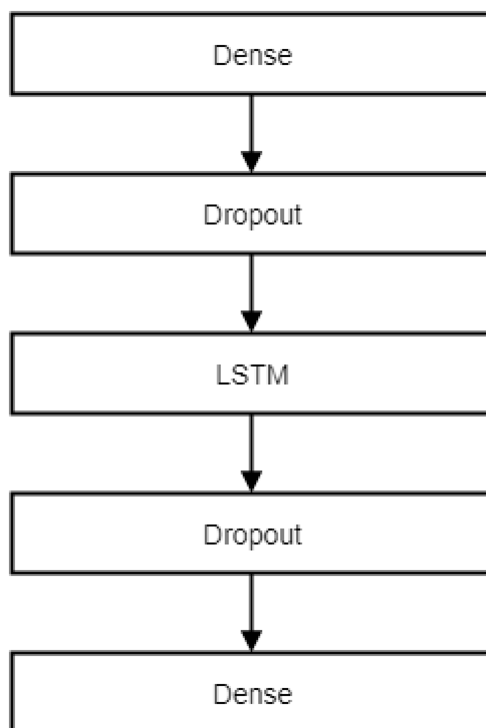


**図 7　ドロップアウトありのモデル構造**

## 第３節 ラベル生成プログラム

Merlin では学習を行った後、テスト・バリデーション用に指定したラベルファイルについては自動で合成・出力する。学習を行ったモデルで任意の読み上げ文を合成するには、別途元となるフルコンテキストラベルが必要となる。ラベルを用意するにはいくつか方法があるが、本研究では OpenJTalk にログファイルに含まれるものを正規表現で抽出する方法での実装を行った。こちらも詳細なプログラムは付録を参照されたい。

**ディレクトリ構成**

dest/ディレクトリはプログラムによって自動生成され、フルコンテキストラベルとモノラベルをそれぞれ格納する。ラベルファイル名は文章数に合わせてゼロフィルされた連番となる。

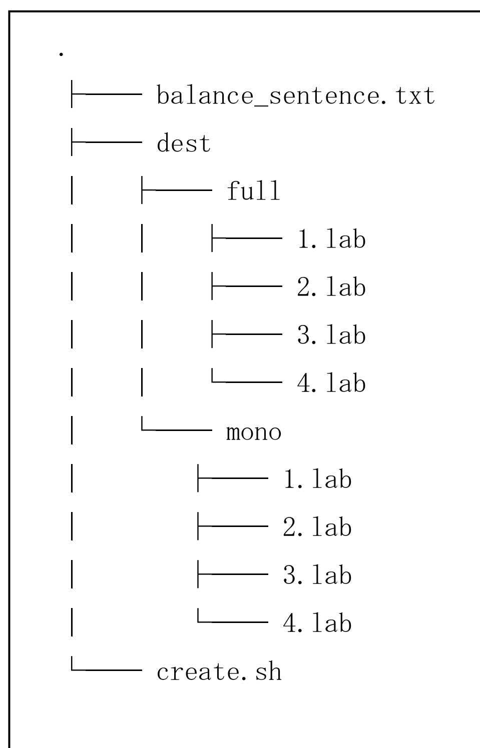


**図 8　ラベル生成プログラムのディレクトリ構造**

**balance_sentence.txt**

ラベルファイルの生成元となる文章を記述するファイルである。研究初期ではデータセットに声優統計コーパスを使用する予定だったため、声優統計コーパス[16]に付属している balance_sentence.txt と同形式である。そのため、以下の書式をとる。

```
sentence_id    sentence        yomi    monophone
```

**図 9　balance_sentence.txt の書式**

しかしながら、プログラム中でラベルファイルの生成に使用するのは読み上げ文章である sentence 項のみであるため、任意のテキストのラベルファイルを生成したい場合は sentence 項のみを記述し、その他は以下のように適当なプレースホルダを記述するだけでよい。

```
.	なりたい自分に近づける自己分析	.	.
```

**図 10　balance_sentence.txt の例**

# 第 4 節 任意テキストの音声合成プログラム

すでにモデルは学習済みであると仮定する。このとき生成したフルコンテキストラベルファイルを、
egs/japanese_voice/s1/experiments/japanese_voice/test_synthesis/prompt-lab
に配置し、
egs/japanese_voice/s1/experiments/japanese_voice/test_synthesis/test_is_list.scp
にファイル名を記述する。この状態で egs/japanese_voice/s1/japanese_synthesis.sh を実行することで任意のテキストファイルから音声を合成できることになる。

しかしながら実際にはファイルの退避を行う必要があるなど、この手順には非常に手間がかかるため、一連の作業を自動化するプログラムを作成した。このプログラムは以下のように実行する

```
$ cd egs/japanese_voice/s1/
$ ./generate.sh “任意のテキスト”
```

**図 11　音声合成プログラムの実行**

これを行うことで、任意のテキスト一文から音声を合成することができる。実際のプログラムは付録を参照されたい。

# 第4章　比較

実際にそれぞれのモデルで学習・生成した場合の結果である。なお、生成ではすべてのモデルで「水をマレーシアから買わなくてはならないのです」という文の読み上げを合成している。この文はbasic5000に含まれるものであるが、先頭の100ファイルをテスト・バリデーション用に除外しているため学習には使用していない。

**基本設定のモデル**

まずは基本設定のモデルである。合成した音声はイントネーションが弱く不自然さが見受けられた。

Accuracyに着目すると、いったんは精度が向上しているものの、エポックが進むにつれて徐々に降下している。また、Lossに着目すると、Accuracyと同じく最初は向上し、徐々に降下していることがわかる。これにはいくつか考えられるが、基本設定のモデルが時系列的な要素を学習するのに十分な構造を持っていなかったことが考えられる。

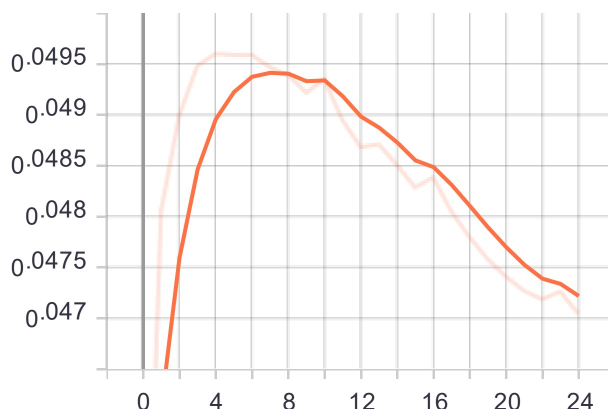


図 12　基本設定モデルの Accuracy（音響モデル）

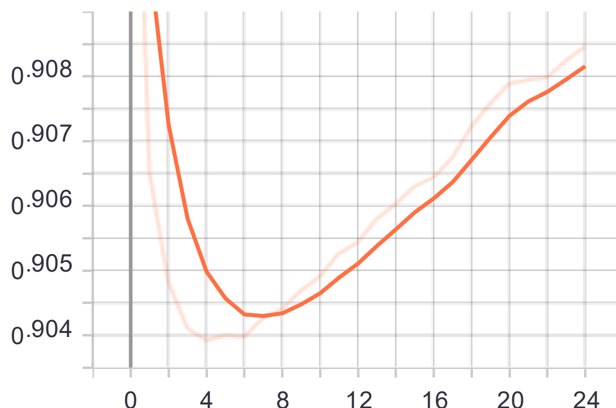


図 13　基本設定モデルの Loss（音響モデル）

**ドロップアウト無し**

合成した音声はイントネーションが強いがやや不自然に感じられた。

はっきりとした発音にはなっているが、とげとげしい印象がある。これは、学習データの影響を受けすぎてしまったのではないかと考えられる。

Accuracy に注目すると、順調に精度が上がっていることがわかる。

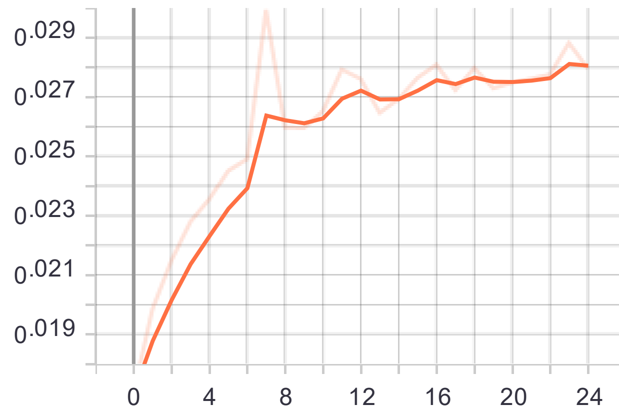


図 14　ドロップアウトなしの Accuracy（音響モデル）

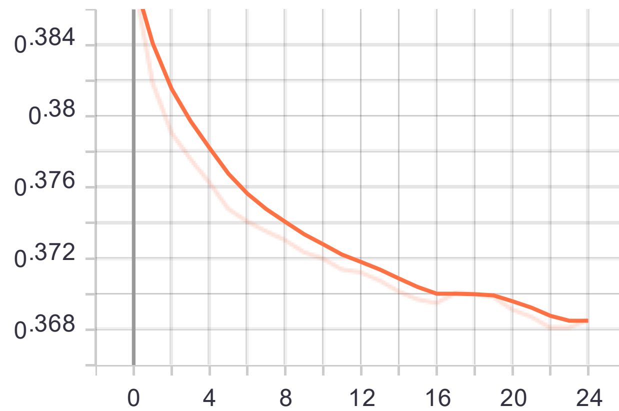


図 15　ドロップアウトなしの Loss（音響モデル）

**ドロップアウトあり**

合成した音声は、比較対象の 3 つの中で最も自然に感じられた。一方でイントネーションがやや弱い印象を受けるが、これはドロップアウトの汎化性能によって全体的な適応度が上がったためと考えられる。

Accuracy に注目すると、基本設定のモデル・ドロップアウトのないモデルと比較して学習上では精度がかなり高いことがわかる。

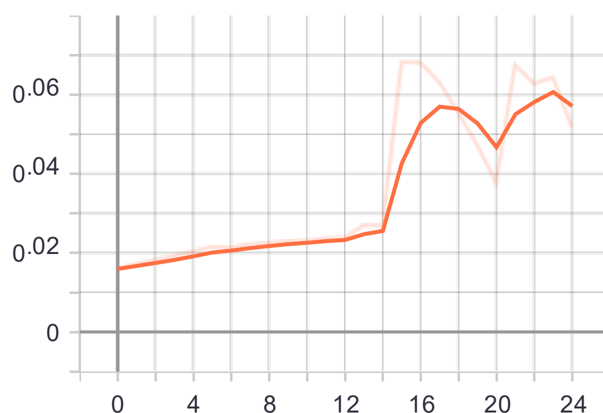


図 16　ドロップアウトありの Accuracy（音響モデル）

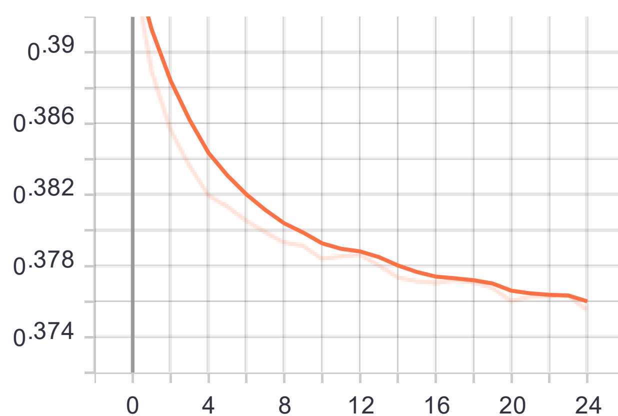


図 17　ドロップアウトありの Loss（音響モデル）

# 第5章　結果

どのモデルを使っても学習・生成は問題なく完了した。実行結果より、基本設定のモデルよりも本研究で作成したモデルの方が効果的であることがわかった。ドロップアウトのないモデルではイントネーションははっきりしているが、所々に不自然さが見られた。またドロップアウトのあるモデルでは、ドロップアウトのないモデルと比べるとイントネーションがやや弱いが、全体的に不自然さの少ない発話になっていることがわかる。これはドロップアウトの汎化性能によって全体的な適応度が向上したからだと予想できる。今後エポック数を増やした場合には過学習の影響が懸念されるため、ドロップアウトの効果がより高まると考えられる。

また比較には記載していないが、ドロップアウトのあるモデルで中間層のノード数を倍の 1024 個にした場合、よりイントネーションの自然さが増したように感じられた。

本研究から、モデルの構造次第で学習結果には大きく差が出ることが分かり、より品質の良い結果を生み出すことが可能であると考えられる。

# 第6章　今後の課題

本研究では、同じ条件下でドロップアウトを使用した場合、過学習が防止され結果の自然さが上昇することに加え、中間層のノードを増やすことでイントネーションの質が上がることが分かった。しかしながら層の追加や、層ごとのノード数の追加を行えば、学習や生成にかかる時間はより長くなる。品質と学習・生成時間は一般的にトレードオフであるが、より特徴や時系列情報を保持しやすいモデルを用いることによって、単純に層を重ねた場合と比較して学習や生成時間を抑えつつ、よりイントネーションや発話の品質を向上させることができると思われる。具体的な方法としては、ResidualNetwork[17]など入力の特徴を失いにくいモデルを用いることで品質の向上が見込まれる。ResidualNetwork は層をまたいで短絡させた入力を足し合わせることで、深いネットワークでも精度が落ちることなく学習できることが知られている。

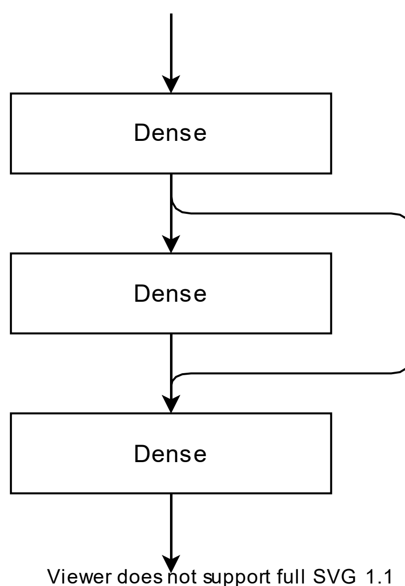


**図 18　単純な RedisualNet の構造**

また現状、テキスト合成までの手順が非常に複雑でありエンドユーザには扱いにくい。そのため今後もし音声合成ソフトウェアとして発展させる場合には、手順の改善や GUI フロントエンドの作成、もしくはモデルのみを利用したソフトウェアを新たに作る必要があると考えられる。

# 参　考　文　献

[1] Zhizheng Wu, Oliver Watts, Simon King, "Merlin: An Open Source Neural Network Speech Synthesis System" in Proc. 9th ISCA Speech Synthesis Workshop (SSW9), September 2016, Sunnyvale, CA, USA.
[2] Ryosuke Sonobe, Shinnosuke Takamichi and Hiroshi Saruwatari, "*JSUT corpus: free large-scale Japanese speech corpus for end-to-end speech synthesis*," arXiv preprint, 1711.00354, 2017.
[3] just-lab https://github.com/r9y9/jsut-lab
[4] OpenJTalk http://open-jtalk.sourceforge.net/
[5] segmentation-kit https://github.com/julius-speech/segmentation-kit
[6] A. Lee and T. Kawahara: Julius v4.5 (2019) https://doi.org/10.5281/zenodo.2530395
[7] M. Morise, F. Yokomori, and K. Ozawa, ``WORLD: a vocoder-based high-quality speech synthesis system for real-time applications,'' IEICE transactions on information and systems, vol. E99-D, no. 7, pp. 1877-1884, 2016.
[8] SPTK http://sp-tk.sourceforge.net/
[9] Theano https://github.com/Theano/Theano
[10] Keras https://github.com/keras-team/keras
[11] Tensorboard https://github.com/tensorflow/tensorboard
[12] Tensorflow https://github.com/tensorflow/tensorflow
[13] Tensorflow-ROCm https://github.com/ROCmSoftwarePlatform/tensorflow-upstream
[14] iBuddle https://www.ibuddie.site/blog/545
[15] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov, Dropout: A simple way to prevent neural networks from overfitting, Journal of Machine Learning Research 15 (2014), 1929-1958.
[16] y benjo and MagnesiumRibbon, Voice-actress corpus.
[17] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, Deep residual learning for image recognition, 2015.

付録

**モデル定義用プログラム**

```
def define_custom_model(self):
    print('--- define custom model ---')
    print('--- debug n_in: ' + str(self.n_in))
    print('--- debug n_out: ' + str(self.n_out))

    seed = 12345
    np.random.seed(seed)

    n_hidden = 512
    use_dropout = False
    dropout_rate1 = 0.2
    dropout_rate2 = 0.5

    self.model.add(
        Dense(
            input_shape=(None, self.n_in),
            units=n_hidden,
            kernel_initializer="normal"))
    self.model.add(
        Dropout(dropout_rate1) if use_dropout else Dense(n_hidden))
    self.model.add(
        LSTM(
            units=n_hidden,
            input_shape=(None, n_hidden),
            kernel_initializer='glorot_uniform',
            return_sequences=True))
    self.model.add(
        Dropout(dropout_rate2) if use_dropout else Dense(n_hidden))
    self.model.add(
        Dense(
            units=self.n_out,
            input_dim=n_hidden,
            kernel_initializer='normal',
            activation='linear'))
```

```
    # Compile the model
    self.compile_model()
```

学習用プログラム

train_sequence_model を改変し、TensorBoard を使用するように変更したプログラムである。このプログラムは新たに関数として定義される。

```
def train_custom_model(self, train_x, train_y, valid_x, valid_y, train_flen, batch_size=1, num_of_epochs=10, shuffle_data=True, training_algo=1):
    print('--- train_custom_model ---')
    batch_size = 32
    num_of_epochs = 25

    tensorboard = TensorBoard(
        log_dir='./logs',
        histogram_freq=0,
        batch_size=batch_size,
        write_graph=True,
        write_grads=True,
    )
    tensorboard.set_model(self.model)

    train_id_list = list(train_flen['utt2framenum'].keys())
    if shuffle_data:
        random.seed(271638)
        random.shuffle(train_id_list)

    train_file_number = len(train_id_list)
    for epoch_num in range(num_of_epochs):
        print(('Epoch: %d/%d ' %(epoch_num+1, num_of_epochs)))
        file_num = 0
        while file_num < train_file_number:
            train_idx_list = train_id_list[file_num: file_num + batch_size]
            seq_len_arr    = [train_flen['utt2framenum'][filename] for filename in train_idx_list]
            max_seq_length = max(seq_len_arr)
            sub_train_x    = dict((filename, train_x[filename]) for filename in train_idx_list)
```

```
            sub_train_y      = dict((filename, train_y[filename]) for filename in
train_idx_list)
            temp_train_x      =  data_utils.transform_data_to_3d_matrix(sub_train_x,
max_length=max_seq_length)
            temp_train_y      =  data_utils.transform_data_to_3d_matrix(sub_train_y,
max_length=max_seq_length)
            logs = self.model.train_on_batch(temp_train_x, temp_train_y)
            file_num += len(train_idx_list)
            data_utils.drawProgressBar(file_num, train_file_number)

        tensorboard.on_epoch_end(epoch_num, dict(zip(self.model.metrics_names, logs)))
        print(" Validation error: %.3f" % (self.get_validation_error(valid_x,
valid_y)))

    tensorboard.on_train_end(None)
```

フルコンテキストラベル作成用プログラム

```
#!/usr/bin/env python3
import os
import re
import subprocess

balance_sentence = 'balance_sentence.txt'
log_name = 'open_jtalk.log'
dest = 'dest/'
full_dest = dest + 'full/'
mono_dest = dest + 'mono/'

os.makedirs(full_dest, exist_ok=True)
os.makedirs(mono_dest, exist_ok=True)

# [Output label]から次の空行までを最短一致でキャプチャする
output_label_pattern = r'¥[Output label¥]¥n([¥s¥S]+?)¥n^¥n'

# - から + までを最短一致でキャプチャする
mono_label_pattern = r'¥-(.+?)¥+'

openjtalk = 'echo {} | open_jtalk -x /var/lib/mecab/dic/open-jtalk/naist-jdic -m
```

```
/usr/share/hts-voice/nitech-jp-atr503-m001/nitech_jp_atr503_m001.htsvoice -r 1.0 -ot '
+ log_name

  # バランス文を読み込み
  sentences = ()
  with open(balance_sentence, 'r', encoding='utf-8') as f:
      sentences = f.read().strip().split('¥n')

  # 文ごとにフルラベルを生成し配列に格納
  full_labels = []
  for i in range(1, len(sentences)):
      id, sentence, yomi, monophone = sentences[i].split('¥t')

      subprocess.run(openjtalk.format(sentence), shell=True)
      with open(log_name, 'r', encoding='utf-8') as f:
          full_labels.append(re.search(output_label_pattern,                    f.read(),
flags=re.MULTILINE).group(1))
  os.remove(log_name)

  # フルラベルからモノラベルを生成し配列に格納
  mono_labels = []
  for i in range(0, len(full_labels)):
      label = []
      for line in full_labels[i].split('¥n'):
          start, end, _label = line.split(' ')
          label.append('     '.join([start,      end,      re.search(mono_label_pattern,
line).group(1)]))
      mono_labels.append('¥n'.join(label))

  # フルラベルを連番ファイルとして出力
  for i in range(len(full_labels)):
      number = str(i + 1)
      digits = len(str(len(full_labels)))
      with open(full_dest+number.zfill(digits)+'.lab', 'w', encoding='utf-8') as f:
          f.write(full_labels[i])

  # モノラベルを連番ファイルとして出力
  for i in range(len(mono_labels)):
```

```
    number = str(i + 1)
    digits = len(str(len(mono_labels)))
    with open(mono_dest+number.zfill(digits)+'.lab', 'w', encoding='utf-8') as f:
        f.write(mono_labels[i])
```

任意テキストの音声合成プログラム

```
#!/usr/bin/env python3
import os
import re
import sys
import shutil
import subprocess

# 出力ファイル名
test_name = 'test'
test_file_name = test_name + '.lab'

log_name = 'open_jtalk.log'
voice_name = 'japanese_voice'

work_dir = 'experiments/' + voice_name
tmp_dir = os.path.join(work_dir, '_test_synthesis')
target_dir = os.path.join(work_dir, 'test_synthesis')

wav_dir = os.path.join(target_dir, 'wav')
gen_lab_dir = os.path.join(target_dir, 'gen-lab')
prompt_lab_dir = os.path.join(target_dir, 'prompt-lab')
list_file = os.path.join(target_dir, 'test_id_list.scp')


# [Output label]から次の空行までを最短一致でキャプチャする正規表現
output_label_pattern = r'¥[Output label¥]¥n([¥s¥S]+?)¥n^¥n'

# OpenJTalk 実行用文字列
openjtalk = 'echo {} | open_jtalk -x /var/lib/mecab/dic/open-jtalk/naist-jdic -m
/usr/share/hts-voice/nitech-jp-atr503-m001/nitech_jp_atr503_m001.htsvoice -r 1.0 -ot '
+ log_name
```

```
# 音声合成実行用文字列
text_to_speech = 'time -p ./07_run_merlin.sh conf/test_dur_synth_{}.conf conf/test_synth_{}.conf'.format(voice_name, voice_name)


if __name__ == '__main__':

    if len(sys.argv) < 2:
        print('error: text is required')
        print('try: ./generate.sh <text>')
        sys.exit(1)

    # 既存のファイルを退避
    shutil.move(target_dir, tmp_dir)

    # ディレクトリを作る
    os.makedirs(target_dir, exist_ok=True)
    os.makedirs(wav_dir, exist_ok=True)
    os.makedirs(gen_lab_dir, exist_ok=True)
    os.makedirs(prompt_lab_dir, exist_ok=True)

    # ラベルテキストの作成
    subprocess.run(openjtalk.format(sys.argv[1]), shell=True)
    with open(log_name, 'r', encoding='utf-8') as f:
        full_label = re.search(
            output_label_pattern,
            f.read(),
            flags=re.MULTILINE
        ).group(1)
    os.remove(log_name)

    # ラベルファイルの書き込み
    with open(os.path.join(prompt_lab_dir, test_file_name), 'w', encoding='utf-8') as f:
        f.write(full_label)

    # リストファイルに書き込み
```

```
with open(os.path.join(list_file), 'w', encoding='utf-8') as f:
    f.write(test_name)

# 生成
subprocess.run(text_to_speech, shell=True)

# カレントディレクトリに合成音声ファイルを移動
shutil.move(os.path.join(wav_dir, test_name + '.wav'), test_name + '.wav')

# ディレクトリをもとに戻す
shutil.rmtree(target_dir)
shutil.move(tmp_dir, target_dir)
```