Artificial Intelligence Nanodegree

Voice User Interfaces

Project: Speech Recognition with Neural Networks

In this notebook, some template code has already been provided for you, and you will need to implement additional functionality to successfully complete this project. You will not need to modify the included code beyond what is requested. Sections that begin with **'(IMPLEMENTATION)'** in the header indicate that the following blocks of code will require additional functionality which you must provide. Please be sure to read the instructions carefully!

> **Note**: Once you have completed all of the code implementations, you need to finalize your work by exporting the Jupyter Notebook as an HTML document. Before exporting the notebook to html, all of the code cells need to have been run so that reviewers can see the final implementation and output. You can then export the notebook by using the menu above and navigating to \n", "**File -> Download as -> HTML (.html).** Include the finished document along with this notebook as your submission.

In addition to implementing code, there will be questions that you must answer which relate to the project and your implementation. Each section where you will answer a question is preceded by a **'Question X'** header. Carefully read each question and provide thorough answers in the following text boxes that begin with **'Answer:'**. Your project submission will be evaluated based on your answers to each of the questions and the implementation you provide.

> **Note:** Code and Markdown cells can be executed using the **Shift + Enter** keyboard shortcut. Markdown cells can be edited by double-clicking the cell to enter edit mode.

The rubric contains *optional* "Stand Out Suggestions" for enhancing the project beyond the minimum requirements. If you decide to pursue the "Stand Out Suggestions", you should include the code in this Jupyter notebook.

Introduction

In this notebook, you will build a deep neural network that functions as part of an end-to-end automatic speech recognition (ASR) pipeline! Your completed pipeline will accept raw audio as input and return a predicted transcription of the spoken language. The full pipeline is summarized in the figure below.

- **STEP 1** is a pre-processing step that converts raw audio to one of two feature representations that are commonly used for ASR.
- **STEP 2** is an acoustic model which accepts audio features as input and returns a probability distribution over all potential transcriptions. After learning about the basic types of neural networks that are often used for acoustic modeling, you will engage in your own investigations, to design your own acoustic model!
- **STEP 3** in the pipeline takes the output from the acoustic model and returns a predicted transcription.

Feel free to use the links below to navigate the notebook:

- The Data
- **STEP 1**: Acoustic Features for Speech Recognition
- **STEP 2**: Deep Neural Networks for Acoustic Modeling

- Model 0: RNN
- Model 1: RNN + TimeDistributed Dense
- Model 2: CNN + RNN + TimeDistributed Dense
- Model 3: Deeper RNN + TimeDistributed Dense
- Model 4: Bidirectional RNN + TimeDistributed Dense
- Models 5+
- Compare the Models
- Final Model

- **STEP 3**: Obtain Predictions

The Data

We begin by investigating the dataset that will be used to train and evaluate your pipeline. LibriSpeech (http://www.danielpovey.com/files/2015_icassp_librispeech.pdf) is a large corpus of English-read speech, designed for training and evaluating models for ASR. The dataset contains 1000 hours of speech derived from audiobooks. We will work with a small subset in this project, since larger-scale data would take a long while to train. However, after completing this project, if you are interested in exploring further, you are encouraged to work with more of the data that is provided online (http://www.openslr.org/12/).

In the code cells below, you will use the `vis_train_features` module to visualize a training example. The supplied argument `index=0` tells the module to extract the first example in the training set. (You are welcome to change `index=0` to point to a different training example, if you like, but please **DO NOT** amend any other code in the cell.) The returned variables are:

- `vis_text` - transcribed text (label) for the training example.
- `vis_raw_audio` - raw audio waveform for the training example.
- `vis_mfcc_feature` - mel-frequency cepstral coefficients (MFCCs) for the training example.
- `vis_spectrogram_feature` - spectrogram for the training example.
- `vis_audio_path` - the file path to the training example.

In [1]:

```
MFCC = False          # MFCC or spectogram

if MFCC:
    inp_dim = 13
    is_spectrogram = False
else:
    inp_dim = 161
    is_spectrogram = True
```

In [2]:

```
#!pip install python_speech_features
#!conda install -y -c conda-forge librosa
```

In [ ]:

```

```

In [3]:

```
from data_generator import vis_train_features

# extract label and audio features for a single training example
vis_text, vis_raw_audio, vis_mfcc_feature, vis_spectrogram_feature, vis_audio_path = vis_tra
in_features()
```

```
Bad key "text.kerning_factor" on line 4 in
/home/ec2-user/anaconda3/envs/tensorflow_p36/lib/python3.6/site-packages/matplot
lib/mpl-data/stylelib/_classic_test_patch.mplstyle.
You probably need to get an updated matplotlibrc file from
https://github.com/matplotlib/matplotlib/blob/v3.1.3/matplotlibrc.template
or from the matplotlib source distribution
```

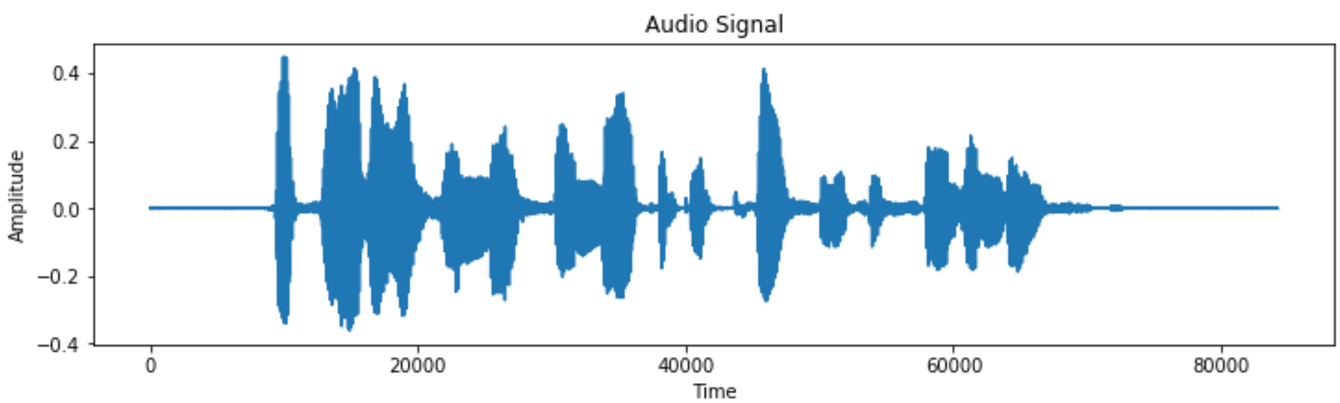
```
There are 2023 total training examples.
```

The following code cell visualizes the audio waveform for your chosen example, along with the corresponding transcript. You also have the option to play the audio in the notebook!

In [4]:

```
from IPython.display import Markdown, display
from data_generator import vis_train_features, plot_raw_audio
from IPython.display import Audio
%matplotlib inline

# plot audio signal
plot_raw_audio(vis_raw_audio)
# print length of audio signal
display(Markdown('**Shape of Audio Signal** : ' + str(vis_raw_audio.shape)))
# print transcript corresponding to audio clip
display(Markdown('**Transcript** : ' + str(vis_text)))
# play the audio file
Audio(vis_audio_path)
```



Shape of Audio Signal : (84231,)

Transcript : her father is a most remarkable person to say the least

Out[4]:

0:00 / 0:03

STEP 1: Acoustic Features for Speech Recognition

For this project, you won't use the raw audio waveform as input to your model. Instead, we provide code that first performs a pre-processing step to convert the raw audio to a feature representation that has historically proven successful for ASR models. Your acoustic model will accept the feature representation as input.

In this project, you will explore two possible feature representations. *After completing the project*, if you'd like to read more about deep learning architectures that can accept raw audio input, you are encouraged to explore this research paper (https://pdfs.semanticscholar.org/a566/cd4a8623d661a4931814d9dffc72ecbf63c4.pdf).

Spectrograms

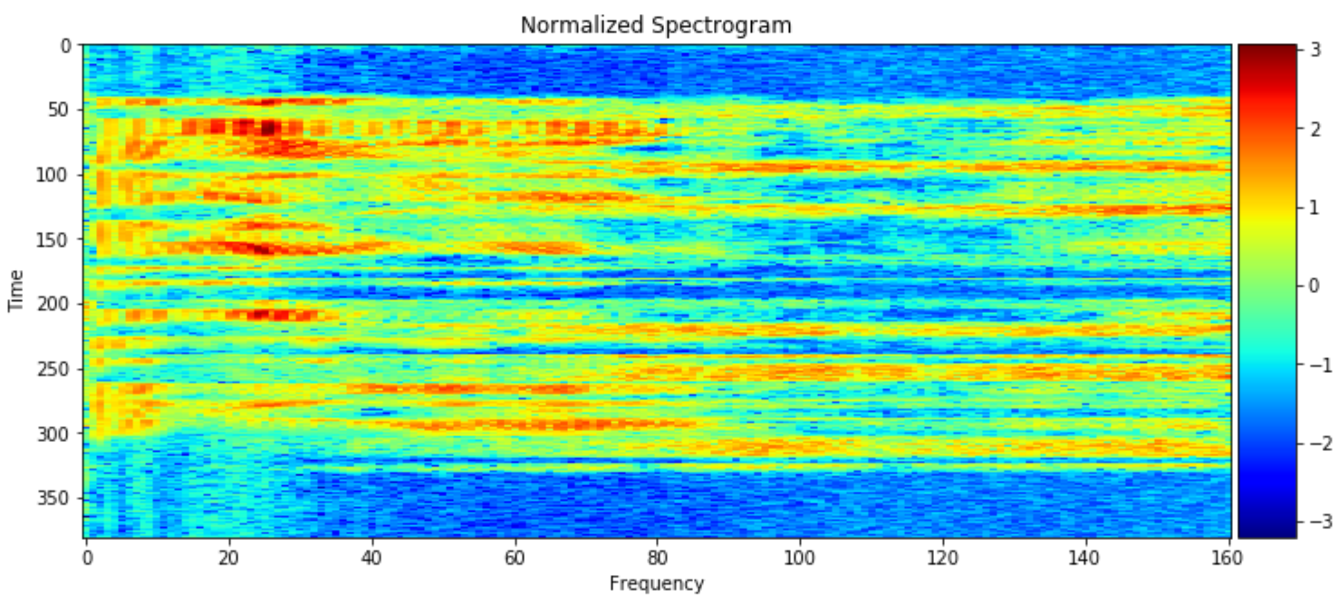
The first option for an audio feature representation is the spectrogram (https://www.youtube.com/watch?v=_FatxGN3vAM). In order to complete this project, you will **not** need to dig deeply into the details of how a spectrogram is calculated; but, if you are curious, the code for calculating the spectrogram was borrowed from this repository (https://github.com/baidu-research/ba-dls-deepspeech). The implementation appears in the `utils.py` file in your repository.

The code that we give you returns the spectrogram as a 2D tensor, where the first (*vertical*) dimension indexes time, and the second (*horizontal*) dimension indexes frequency. To speed the convergence of your algorithm, we have also normalized the spectrogram. (You can see this quickly in the visualization below by noting that the mean value hovers around zero, and most entries in the tensor assume values close to zero.)

In [5]:

```
from data_generator import plot_spectrogram_feature

# plot normalized spectrogram
plot_spectrogram_feature(vis_spectrogram_feature)
# print shape of spectrogram
display(Markdown('**Shape of Spectrogram** : ' + str(vis_spectrogram_feature.shape)))
```



Shape of Spectrogram : (381, 161)

Mel-Frequency Cepstral Coefficients (MFCCs)

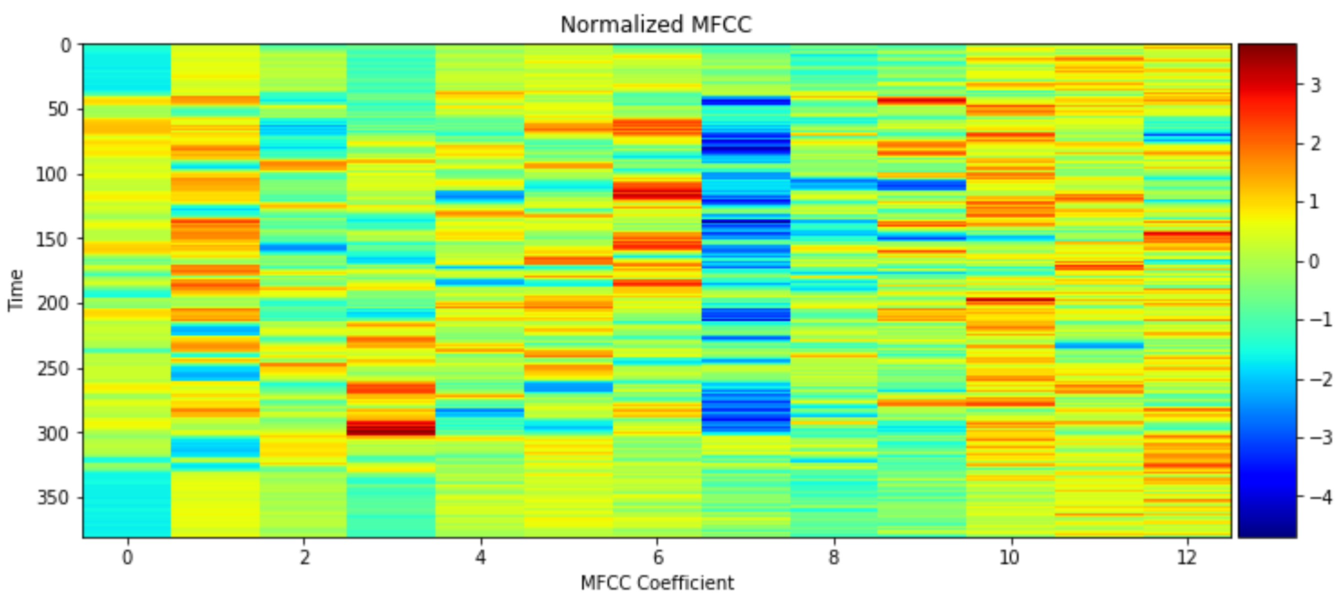
The second option for an audio feature representation is [MFCCs (https://en.wikipedia.org/wiki/Mel-frequency_cepstrum)](https://en.wikipedia.org/wiki/Mel-frequency_cepstrum). You do **not** need to dig deeply into the details of how MFCCs are calculated, but if you would like more information, you are welcome to peruse the [documentation (https://github.com/jameslyons/python_speech_features)](https://github.com/jameslyons/python_speech_features) of the `python_speech_features` Python package. Just as with the spectrogram features, the MFCCs are normalized in the supplied code.

The main idea behind MFCC features is the same as spectrogram features: at each time window, the MFCC feature yields a feature vector that characterizes the sound within the window. Note that the MFCC feature is much lower-dimensional than the spectrogram feature, which could help an acoustic model to avoid overfitting to the training dataset.

In [6]:

```
from data_generator import plot_mfcc_feature

# plot normalized MFCC
plot_mfcc_feature(vis_mfcc_feature)
# print shape of MFCC
display(Markdown('**Shape of MFCC** : ' + str(vis_mfcc_feature.shape)))
```



Shape of MFCC : (381, 13)

When you construct your pipeline, you will be able to choose to use either spectrogram or MFCC features. If you would like to see different implementations that make use of MFCCs and/or spectrograms, please check out the links below:

- This [repository (https://github.com/baidu-research/ba-dls-deepspeech)](https://github.com/baidu-research/ba-dls-deepspeech) uses spectrograms.
- This [repository (https://github.com/mozilla/DeepSpeech)](https://github.com/mozilla/DeepSpeech) uses MFCCs.
- This [repository (https://github.com/buriburisuri/speech-to-text-wavenet)](https://github.com/buriburisuri/speech-to-text-wavenet) also uses MFCCs.
- This [repository (https://github.com/pannous/tensorflow-speech-recognition/blob/master/speech_data.py)](https://github.com/pannous/tensorflow-speech-recognition/blob/master/speech_data.py) experiments with raw audio, spectrograms, and MFCCs as features.

STEP 2: Deep Neural Networks for Acoustic Modeling

In this section, you will experiment with various neural network architectures for acoustic modeling.

You will begin by training five relatively simple architectures. **Model 0** is provided for you. You will write code to implement **Models 1**, **2**, **3**, and **4**. If you would like to experiment further, you are welcome to create and train more models under the **Models 5+** heading.

All models will be specified in the `sample_models.py` file. After importing the `sample_models` module, you will train your architectures in the notebook.

After experimenting with the five simple architectures, you will have the opportunity to compare their performance. Based on your findings, you will construct a deeper architecture that is designed to outperform all of the shallow models.

For your convenience, we have designed the notebook so that each model can be specified and trained on separate occasions. That is, say you decide to take a break from the notebook after training **Model 1**. Then, you need not re-execute all prior code cells in the notebook before training **Model 2**. You need only re-execute the code cell below, that is marked with **`RUN THIS CODE CELL IF YOU ARE RESUMING THE NOTEBOOK AFTER A BREAK`**, before transitioning to the code cells corresponding to **Model 2**.

In [7]:

```
#####################################################################
# RUN THIS CODE CELL IF YOU ARE RESUMING THE NOTEBOOK AFTER A BREAK #
#####################################################################

# allocate 50% of GPU memory (if you like, feel free to change this)
from keras.backend.tensorflow_backend import set_session
import tensorflow as tf
config = tf.ConfigProto()
#config.gpu_options.per_process_gpu_memory_fraction = 0.5
config.gpu_options.per_process_gpu_memory_fraction = 1.0
set_session(tf.Session(config=config))

# watch for any changes in the sample_models module, and reload it automatically
%load_ext autoreload
%autoreload 2
# import NN architectures for speech recognition
from sample_models import *
# import function for training acoustic model
from train_utils import train_model
```

```
Using TensorFlow backend.

WARNING:tensorflow:From /home/ec2-user/anaconda3/envs/tensorflow_p36/lib/python
3.6/site-packages/tensorflow_core/__init__.py:1473: The name tf.estimator.inputs
is deprecated. Please use tf.compat.v1.estimator.inputs instead.
```

Model 0: RNN

Given their effectiveness in modeling sequential data, the first acoustic model you will use is an RNN. As shown in the figure below, the RNN we supply to you will take the time sequence of audio features as input.

At each time step, the speaker pronounces one of 28 possible characters, including each of the 26 letters in the English alphabet, along with a space character (" "), and an apostrophe (').

The output of the RNN at each time step is a vector of probabilities with 29 entries, where the i-th entry encodes the probability that the i-th character is spoken in the time sequence. (The extra 29th character is an empty "character" used to pad training examples within batches containing uneven lengths.) If you would like to peek under the hood at how characters are mapped to indices in the probability vector, look at the `char_map.py` file in the repository. The figure below shows an equivalent, rolled depiction of the RNN that shows the output layer in greater detail.

The model has already been specified for you in Keras. To import it, you need only run the code cell below.

In [8]:

```
model_0 = simple_rnn_model(input_dim=inp_dim) # change to 13 if you would like to use MFCC f
eatures
```

```
WARNING:tensorflow:From /home/ec2-user/anaconda3/envs/tensorflow_p36/lib/python
3.6/site-packages/keras/backend/tensorflow_backend.py:517: The name tf.placehold
er is deprecated. Please use tf.compat.v1.placeholder instead.

WARNING:tensorflow:From /home/ec2-user/anaconda3/envs/tensorflow_p36/lib/python
3.6/site-packages/keras/backend/tensorflow_backend.py:74: The name tf.get_defaul
t_graph is deprecated. Please use tf.compat.v1.get_default_graph instead.

WARNING:tensorflow:From /home/ec2-user/anaconda3/envs/tensorflow_p36/lib/python
3.6/site-packages/keras/backend/tensorflow_backend.py:4138: The name tf.random_u
niform is deprecated. Please use tf.random.uniform instead.


_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
the_input (InputLayer)       (None, None, 161)         0         
_________________________________________________________________
rnn (GRU)                    (None, None, 29)          16617     
_________________________________________________________________
softmax (Activation)         (None, None, 29)          0         
=================================================================
Total params: 16,617
Trainable params: 16,617
Non-trainable params: 0
_________________________________________________________________
None
```

As explored in the lesson, you will train the acoustic model with the [CTC loss (http://www.cs.toronto.edu/~graves/icml_2006.pdf)](http://www.cs.toronto.edu/~graves/icml_2006.pdf) criterion. Custom loss functions take a bit of hacking in Keras, and so we have implemented the CTC loss function for you, so that you can focus on trying out as many deep learning architectures as possible :). If you'd like to peek at the implementation details, look at the `add_ctc_loss` function within the `train_utils.py` file in the repository.

To train your architecture, you will use the `train_model` function within the `train_utils` module; it has already been imported in one of the above code cells. The `train_model` function takes three **required** arguments:

- `input_to_softmax` - a Keras model instance.
- `pickle_path` - the name of the pickle file where the loss history will be saved.
- `save_model_path` - the name of the HDF5 file where the model will be saved.

If we have already supplied values for `input_to_softmax` , `pickle_path` , and `save_model_path` , please **DO NOT** modify these values.

There are several **optional** arguments that allow you to have more control over the training process. You are welcome to, but not required to, supply your own values for these arguments.

- `minibatch_size` - the size of the minibatches that are generated while training the model (default: `20`).
- `spectrogram` - Boolean value dictating whether spectrogram (`True`) or MFCC (`False`) features are used for training (default: `True`).
- `mfcc_dim` - the size of the feature dimension to use when generating MFCC features (default: `13`).
- `optimizer` - the Keras optimizer used to train the model (default: `SGD(lr=0.02, decay=1e-6, momentum=0.9, nesterov=True, clipnorm=5)`).
- `epochs` - the number of epochs to use to train the model (default: `20`). If you choose to modify this parameter, make sure that it is *at least* 20.
- `verbose` - controls the verbosity of the training output in the `model.fit_generator` method (default: `1`).
- `sort_by_duration` - Boolean value dictating whether the training and validation sets are sorted by (increasing) duration before the start of the first epoch (default: `False`).

The `train_model` function defaults to using spectrogram features; if you choose to use these features, note that the acoustic model in `simple_rnn_model` should have `input_dim=161` . Otherwise, if you choose to use MFCC features, the acoustic model should have `input_dim=13` .

We have chosen to use `GRU` units in the supplied RNN. If you would like to experiment with `LSTM` or `SimpleRNN` cells, feel free to do so here. If you change the `GRU` units to `SimpleRNN` cells in `simple_rnn_model` , you may notice that the loss quickly becomes undefined (`nan`) - you are strongly encouraged to check this for yourself! This is due to the [exploding gradients problem (http://www.wildml.com/2015/10/recurrent-neural-networks-tutorial-part-3-backpropagation-through-time-and-vanishing-gradients/)](http://www.wildml.com/2015/10/recurrent-neural-networks-tutorial-part-3-backpropagation-through-time-and-vanishing-gradients/). We have already implemented [gradient clipping (https://arxiv.org/pdf/1211.5063.pdf)](https://arxiv.org/pdf/1211.5063.pdf) in your optimizer to help you avoid this issue.

IMPORTANT NOTE: If you notice that your gradient has exploded in any of the models below, feel free to explore more with gradient clipping (the `clipnorm` argument in your optimizer) or swap out any `SimpleRNN` cells for `LSTM` or `GRU` cells. You can also try restarting the kernel to restart the training process.

In [9]:

```
train_model(input_to_softmax=model_0,
            pickle_path='model_0.pickle',
            save_model_path='model_0.h5',
            spectrogram=is_spectrogram) # change to False if you would like to use MFCC feat
ures
```

```
WARNING:tensorflow:From /home/ec2-user/anaconda3/envs/tensorflow_p36/lib/python
3.6/site-packages/keras/backend/tensorflow_backend.py:4249: to_int32 (from tenso
rflow.python.ops.math_ops) is deprecated and will be removed in a future versio
n.
Instructions for updating:
Use `tf.cast` instead.
WARNING:tensorflow:From /home/ec2-user/anaconda3/envs/tensorflow_p36/lib/python
3.6/site-packages/tensorflow_core/python/ops/array_ops.py:1475: where (from tens
orflow.python.ops.array_ops) is deprecated and will be removed in a future versi
on.
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
WARNING:tensorflow:From /home/ec2-user/anaconda3/envs/tensorflow_p36/lib/python
3.6/site-packages/keras/backend/tensorflow_backend.py:4229: to_int64 (from tenso
rflow.python.ops.math_ops) is deprecated and will be removed in a future versio
n.
Instructions for updating:
Use `tf.cast` instead.
WARNING:tensorflow:From /home/ec2-user/anaconda3/envs/tensorflow_p36/lib/python
3.6/site-packages/keras/backend/tensorflow_backend.py:4253: The name tf.log is d
eprecated. Please use tf.math.log instead.

WARNING:tensorflow:From /home/ec2-user/anaconda3/envs/tensorflow_p36/lib/python
3.6/site-packages/keras/optimizers.py:790: The name tf.train.Optimizer is deprec
ated. Please use tf.compat.v1.train.Optimizer instead.

WARNING:tensorflow:From /home/ec2-user/anaconda3/envs/tensorflow_p36/lib/python
3.6/site-packages/keras/backend/tensorflow_backend.py:986: The name tf.assign_ad
d is deprecated. Please use tf.compat.v1.assign_add instead.

WARNING:tensorflow:From /home/ec2-user/anaconda3/envs/tensorflow_p36/lib/python
3.6/site-packages/keras/backend/tensorflow_backend.py:973: The name tf.assign is
deprecated. Please use tf.compat.v1.assign instead.

Epoch 1/20
WARNING:tensorflow:From /home/ec2-user/anaconda3/envs/tensorflow_p36/lib/python
3.6/site-packages/keras/backend/tensorflow_backend.py:174: The name tf.get_defau
lt_session is deprecated. Please use tf.compat.v1.get_default_session instead.

WARNING:tensorflow:From /home/ec2-user/anaconda3/envs/tensorflow_p36/lib/python
3.6/site-packages/keras/backend/tensorflow_backend.py:190: The name tf.global_va
riables is deprecated. Please use tf.compat.v1.global_variables instead.

WARNING:tensorflow:From /home/ec2-user/anaconda3/envs/tensorflow_p36/lib/python
3.6/site-packages/keras/backend/tensorflow_backend.py:199: The name tf.is_variab
le_initialized is deprecated. Please use tf.compat.v1.is_variable_initialized in
stead.

WARNING:tensorflow:From /home/ec2-user/anaconda3/envs/tensorflow_p36/lib/python
3.6/site-packages/keras/backend/tensorflow_backend.py:206: The name tf.variables
_initializer is deprecated. Please use tf.compat.v1.variables_initializer instea
d.

101/101 [==============================] - 145s 1s/step - loss: 866.2867 - val_l
oss: 757.9863
Epoch 2/20
101/101 [==============================] - 144s 1s/step - loss: 777.3285 - val_l
oss: 754.6674
Epoch 3/20
101/101 [==============================] - 145s 1s/step - loss: 776.2074 - val_l
oss: 755.6445
```

```
Epoch 4/20
101/101 [==============================] - 143s 1s/step - loss: 776.4205 - val_l
oss: 753.9339
Epoch 5/20
101/101 [==============================] - 144s 1s/step - loss: 776.6825 - val_l
oss: 754.4528
Epoch 6/20
101/101 [==============================] - 145s 1s/step - loss: 776.7298 - val_l
oss: 754.9810
Epoch 7/20
101/101 [==============================] - 144s 1s/step - loss: 776.4158 - val_l
oss: 754.9687
Epoch 8/20
101/101 [==============================] - 144s 1s/step - loss: 776.3394 - val_l
oss: 754.6494
Epoch 9/20
101/101 [==============================] - 144s 1s/step - loss: 776.5875 - val_l
oss: 755.4400
Epoch 10/20
101/101 [==============================] - 144s 1s/step - loss: 776.5479 - val_l
oss: 755.4603
Epoch 11/20
101/101 [==============================] - 144s 1s/step - loss: 776.0625 - val_l
oss: 755.1984
Epoch 12/20
101/101 [==============================] - 144s 1s/step - loss: 776.6882 - val_l
oss: 754.8328
Epoch 13/20
101/101 [==============================] - 145s 1s/step - loss: 776.6617 - val_l
oss: 755.2595
Epoch 14/20
101/101 [==============================] - 146s 1s/step - loss: 776.5068 - val_l
oss: 753.7011
Epoch 15/20
101/101 [==============================] - 144s 1s/step - loss: 775.8658 - val_l
oss: 754.9205
Epoch 16/20
101/101 [==============================] - 143s 1s/step - loss: 776.2729 - val_l
oss: 755.8515
Epoch 17/20
101/101 [==============================] - 144s 1s/step - loss: 776.0766 - val_l
oss: 755.4950
Epoch 18/20
101/101 [==============================] - 145s 1s/step - loss: 776.4214 - val_l
oss: 755.1528
Epoch 19/20
101/101 [==============================] - 145s 1s/step - loss: 776.0873 - val_l
oss: 754.7725
Epoch 20/20
101/101 [==============================] - 144s 1s/step - loss: 776.9167 - val_l
oss: 755.4400
```

(IMPLEMENTATION) Model 1: RNN + TimeDistributed Dense

Read about the [TimeDistributed (https://keras.io/layers/wrappers/)](https://keras.io/layers/wrappers/) wrapper and the [BatchNormalization (https://keras.io/layers/normalization/)](https://keras.io/layers/normalization/) layer in the Keras documentation. For your next architecture, you will add [batch normalization (https://arxiv.org/pdf/1510.01378.pdf)](https://arxiv.org/pdf/1510.01378.pdf) to the recurrent layer to reduce training times. The `TimeDistributed` layer will be used to find more complex patterns in the dataset. The unrolled snapshot of the architecture is depicted below.

The next figure shows an equivalent, rolled depiction of the RNN that shows the (`TimeDistrbuted`) dense and output layers in greater detail.

Use your research to complete the `rnn_model` function within the `sample_models.py` file. The function should specify an architecture that satisfies the following requirements:

- The first layer of the neural network should be an RNN (`SimpleRNN` , `LSTM` , or `GRU`) that takes the time sequence of audio features as input. We have added `GRU` units for you, but feel free to change `GRU` to `SimpleRNN` or `LSTM` , if you like!
- Whereas the architecture in `simple_rnn_model` treated the RNN output as the final layer of the model, you will use the output of your RNN as a hidden layer. Use `TimeDistributed` to apply a `Dense` layer to each of the time steps in the RNN output. Ensure that each `Dense` layer has `output_dim` units.

Use the code cell below to load your model into the `model_1` variable. Use a value for `input_dim` that matches your chosen audio features, and feel free to change the values for `units` and `activation` to tweak the behavior of your recurrent layer.

In [10]:

```
model_1 = rnn_model(input_dim=inp_dim, # change to 13 if you would like to use MFCC features
                    units=200,
                    activation='relu')
```

```
WARNING:tensorflow:From /home/ec2-user/anaconda3/envs/tensorflow_p36/lib/python
3.6/site-packages/keras/backend/tensorflow_backend.py:133: The name tf.placehold
er_with_default is deprecated. Please use tf.compat.v1.placeholder_with_default
instead.

_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
the_input (InputLayer)       (None, None, 161)         0         
_________________________________________________________________
rnn (GRU)                    (None, None, 200)         217200    
_________________________________________________________________
batch_normalization_1 (Batch (None, None, 200)         800       
_________________________________________________________________
time_distributed_1 (TimeDist (None, None, 29)          5829      
_________________________________________________________________
softmax (Activation)         (None, None, 29)          0         
=================================================================
Total params: 223,829
Trainable params: 223,429
Non-trainable params: 400
_________________________________________________________________
None
```

Please execute the code cell below to train the neural network you specified in `input_to_softmax`. After the model has finished training, the model is [saved (https://keras.io/getting-started/faq/#how-can-i-save-a-keras-model)](https://keras.io/getting-started/faq/#how-can-i-save-a-keras-model) in the HDF5 file `model_1.h5`. The loss history is [saved (https://wiki.python.org/moin/UsingPickle)](https://wiki.python.org/moin/UsingPickle) in `model_1.pickle`. You are welcome to tweak any of the optional parameters while calling the `train_model` function, but this is not required.

In [11]:

```
train_model(input_to_softmax=model_1,
            pickle_path='model_1.pickle',
            save_model_path='model_1.h5',
            spectrogram=is_spectrogram) # change to False if you would like to use MFCC feat
ures
```

```
Epoch 1/20
101/101 [==============================] - 145s 1s/step - loss: 319.1284 - val_l
oss: 240.5761
Epoch 2/20
101/101 [==============================] - 148s 1s/step - loss: 215.8050 - val_l
oss: 203.6293
Epoch 3/20
101/101 [==============================] - 147s 1s/step - loss: 189.7974 - val_l
oss: 182.2804
Epoch 4/20
101/101 [==============================] - 146s 1s/step - loss: 171.8792 - val_l
oss: 168.4159
Epoch 5/20
101/101 [==============================] - 146s 1s/step - loss: 160.5964 - val_l
oss: 162.6995
Epoch 6/20
101/101 [==============================] - 146s 1s/step - loss: 153.3744 - val_l
oss: 162.5235
Epoch 7/20
101/101 [==============================] - 146s 1s/step - loss: 149.1239 - val_l
oss: 152.5143
Epoch 8/20
101/101 [==============================] - 146s 1s/step - loss: 145.0830 - val_l
oss: 151.1848
Epoch 9/20
101/101 [==============================] - 146s 1s/step - loss: 142.1548 - val_l
oss: 150.1639
Epoch 10/20
101/101 [==============================] - 147s 1s/step - loss: 140.3679 - val_l
oss: 148.3844
Epoch 11/20
101/101 [==============================] - 146s 1s/step - loss: 138.8557 - val_l
oss: 148.6453
Epoch 12/20
101/101 [==============================] - 147s 1s/step - loss: 138.5161 - val_l
oss: 154.4533
Epoch 13/20
101/101 [==============================] - 146s 1s/step - loss: 136.9926 - val_l
oss: 148.4276
Epoch 14/20
101/101 [==============================] - 146s 1s/step - loss: 136.5615 - val_l
oss: 152.6069
Epoch 15/20
101/101 [==============================] - 146s 1s/step - loss: 142.1180 - val_l
oss: 215.0649
Epoch 16/20
101/101 [==============================] - 145s 1s/step - loss: 142.1640 - val_l
oss: 152.3363
Epoch 17/20
101/101 [==============================] - 146s 1s/step - loss: 137.6352 - val_l
oss: 152.1380
Epoch 18/20
101/101 [==============================] - 147s 1s/step - loss: 139.5080 - val_l
oss: 156.0970
Epoch 19/20
101/101 [==============================] - 145s 1s/step - loss: 164.9945 - val_l
oss: 166.5801
Epoch 20/20
101/101 [==============================] - 148s 1s/step - loss: 150.8622 - val_l
oss: 163.5211
```

(IMPLEMENTATION) Model 2: CNN + RNN + TimeDistributed Dense

The architecture in `cnn_rnn_model` adds an additional level of complexity, by introducing a [1D convolution layer (https://keras.io/layers/convolutional/#conv1d)](https://keras.io/layers/convolutional/#conv1d).

This layer incorporates many arguments that can be (optionally) tuned when calling the `cnn_rnn_model` module. We provide sample starting parameters, which you might find useful if you choose to use spectrogram audio features.

If you instead want to use MFCC features, these arguments will have to be tuned. Note that the current architecture only supports values of `'same'` or `'valid'` for the `conv_border_mode` argument.

When tuning the parameters, be careful not to choose settings that make the convolutional layer overly small. If the temporal length of the CNN layer is shorter than the length of the transcribed text label, your code will throw an error.

Before running the code cell below, you must modify the `cnn_rnn_model` function in `sample_models.py`. Please add

In [12]:

```
model_2 = cnn_rnn_model(input_dim=inp_dim, # change to 13 if you would like to use MFCC feat
ures
                        filters=200,
                        kernel_size=11,
                        conv_stride=2,
                        conv_border_mode='valid',
                        units=200)
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
the_input (InputLayer)       (None, None, 161)         0         
_________________________________________________________________
conv1d (Conv1D)              (None, None, 200)         354400    
_________________________________________________________________
bn_conv_1d (BatchNormalizati (None, None, 200)         800       
_________________________________________________________________
rnn (SimpleRNN)              (None, None, 200)         80200     
_________________________________________________________________
batch_normalization_2 (Batch (None, None, 200)         800       
_________________________________________________________________
time_distributed_2 (TimeDist (None, None, 29)          5829      
_________________________________________________________________
softmax (Activation)         (None, None, 29)          0         
=================================================================
Total params: 442,029
Trainable params: 441,229
Non-trainable params: 800
_________________________________________________________________
None

/home/ec2-user/anaconda3/envs/tensorflow_p36/lib/python3.6/site-packages/keras/l
ayers/recurrent.py:1031: UserWarning: The `implementation` argument in `SimpleRN
N` has been deprecated. Please remove it from your layer call.
  warnings.warn('The `implementation` argument '
```

Please execute the code cell below to train the neural network you specified in `input_to_softmax`. After the model has finished training, the model is [saved (https://keras.io/getting-started/faq/#how-can-i-save-a-keras-model)](https://keras.io/getting-started/faq/#how-can-i-save-a-keras-model) in the HDF5 file `model_2.h5`. The loss history is [saved (https://wiki.python.org/moin/UsingPickle)](https://wiki.python.org/moin/UsingPickle) in `model_2.pickle`. You are welcome to tweak any of the optional parameters while calling the `train_model` function, but this is not required.

In [13]:

```
train_model(input_to_softmax=model_2,
            pickle_path='model_2.pickle',
            save_model_path='model_2.h5',
            spectrogram=is_spectrogram) # change to False if you would like to use MFCC feat
ures
```

```
Epoch 1/20
101/101 [==============================] - 69s 684ms/step - loss: 230.1543 - val
_loss: 274.6845
Epoch 2/20
101/101 [==============================] - 51s 503ms/step - loss: 174.4596 - val
_loss: 239.4043
Epoch 3/20
101/101 [==============================] - 50s 492ms/step - loss: 153.6233 - val
_loss: 201.7013
Epoch 4/20
101/101 [==============================] - 50s 494ms/step - loss: 141.5411 - val
_loss: 180.3221
Epoch 5/20
101/101 [==============================] - 50s 498ms/step - loss: 133.1013 - val
_loss: 182.4572
Epoch 6/20
101/101 [==============================] - 50s 496ms/step - loss: 126.7930 - val
_loss: 180.8457
Epoch 7/20
101/101 [==============================] - 50s 494ms/step - loss: 121.4214 - val
_loss: 180.8668
Epoch 8/20
101/101 [==============================] - 50s 493ms/step - loss: 116.0126 - val
_loss: 177.4860
Epoch 9/20
101/101 [==============================] - 50s 494ms/step - loss: 112.1215 - val
_loss: 172.2320
Epoch 10/20
101/101 [==============================] - 50s 497ms/step - loss: 108.3723 - val
_loss: 177.7072
Epoch 11/20
101/101 [==============================] - 50s 492ms/step - loss: 105.1491 - val
_loss: 177.0357
Epoch 12/20
101/101 [==============================] - 51s 504ms/step - loss: 101.0396 - val
_loss: 169.9418
Epoch 13/20
101/101 [==============================] - 50s 497ms/step - loss: 98.3054 - val_
loss: 164.2682
Epoch 14/20
101/101 [==============================] - 51s 501ms/step - loss: 95.3459 - val_
loss: 160.0009
Epoch 15/20
101/101 [==============================] - 50s 498ms/step - loss: 92.5009 - val_
loss: 157.8294
Epoch 16/20
101/101 [==============================] - 50s 495ms/step - loss: 89.4362 - val_
loss: 150.9977
Epoch 17/20
101/101 [==============================] - 50s 496ms/step - loss: 86.8358 - val_
loss: 151.8741
Epoch 18/20
101/101 [==============================] - 50s 498ms/step - loss: 84.8296 - val_
loss: 148.2647
Epoch 19/20
101/101 [==============================] - 51s 506ms/step - loss: 81.7720 - val_
loss: 148.5194
Epoch 20/20
101/101 [==============================] - 50s 496ms/step - loss: 79.6976 - val_
loss: 147.6594
```

(IMPLEMENTATION) Model 3: Deeper RNN + TimeDistributed Dense

Review the code in `rnn_model`, which makes use of a single recurrent layer. Now, specify an architecture in `deep_rnn_model` that utilizes a variable number `recur_layers` of recurrent layers. The figure below shows the architecture that should be returned if `recur_layers=2`. In the figure, the output sequence of the first recurrent layer is used as input for the next recurrent layer.

Feel free to change the supplied values of `units` to whatever you think performs best. You can change the value of `recur_layers`, as long as your final value is greater than 1. (As a quick check that you have implemented the additional functionality in `deep_rnn_model` correctly, make sure that the architecture that you specify here is identical to `rnn_model` if `recur_layers=1`.)

In [8]:

```
model_3 = deep_rnn_model(input_dim=inp_dim, # change to 13 if you would like to use MFCC fea
tures
                         units=200,
                         recur_layers=2)
```

```
WARNING:tensorflow:From /home/ec2-user/anaconda3/envs/tensorflow_p36/lib/python
3.6/site-packages/keras/backend/tensorflow_backend.py:517: The name tf.placehold
er is deprecated. Please use tf.compat.v1.placeholder instead.

WARNING:tensorflow:From /home/ec2-user/anaconda3/envs/tensorflow_p36/lib/python
3.6/site-packages/keras/backend/tensorflow_backend.py:74: The name tf.get_defaul
t_graph is deprecated. Please use tf.compat.v1.get_default_graph instead.

WARNING:tensorflow:From /home/ec2-user/anaconda3/envs/tensorflow_p36/lib/python
3.6/site-packages/keras/backend/tensorflow_backend.py:4138: The name tf.random_u
niform is deprecated. Please use tf.random.uniform instead.

WARNING:tensorflow:From /home/ec2-user/anaconda3/envs/tensorflow_p36/lib/python
3.6/site-packages/keras/backend/tensorflow_backend.py:133: The name tf.placehold
er_with_default is deprecated. Please use tf.compat.v1.placeholder_with_default
instead.

_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
the_input (InputLayer)       (None, None, 161)         0         
_________________________________________________________________
rnn0 (GRU)                   (None, None, 200)         217200    
_________________________________________________________________
batch_normalization_1 (Batch (None, None, 200)         800       
_________________________________________________________________
rnn1 (GRU)                   (None, None, 200)         240600    
_________________________________________________________________
batch_normalization_3 (Batch (None, None, 200)         800       
_________________________________________________________________
time_distributed_1 (TimeDist (None, None, 29)          5829      
_________________________________________________________________
softmax (Activation)         (None, None, 29)          0         
=================================================================
Total params: 465,229
Trainable params: 464,429
Non-trainable params: 800
_________________________________________________________________
None
```

Please execute the code cell below to train the neural network you specified in `input_to_softmax`. After the model has finished training, the model is [saved (https://keras.io/getting-started/faq/#how-can-i-save-a-keras-model)](https://keras.io/getting-started/faq/#how-can-i-save-a-keras-model) in the HDF5 file `model_3.h5`. The loss history is [saved (https://wiki.python.org/moin/UsingPickle)](https://wiki.python.org/moin/UsingPickle) in `model_3.pickle`. You are welcome to tweak any of the optional parameters while calling the `train_model` function, but this is not required.

In [9]:

```
train_model(input_to_softmax=model_3,
            pickle_path='model_3.pickle',
            save_model_path='model_3.h5',
            spectrogram=is_spectrogram) # change to False if you would like to use MFCC feat
ures
```

```
WARNING:tensorflow:From /home/ec2-user/anaconda3/envs/tensorflow_p36/lib/python
3.6/site-packages/keras/backend/tensorflow_backend.py:4249: to_int32 (from tenso
rflow.python.ops.math_ops) is deprecated and will be removed in a future versio
n.
Instructions for updating:
Use `tf.cast` instead.
WARNING:tensorflow:From /home/ec2-user/anaconda3/envs/tensorflow_p36/lib/python
3.6/site-packages/tensorflow_core/python/ops/array_ops.py:1475: where (from tens
orflow.python.ops.array_ops) is deprecated and will be removed in a future versi
on.
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
WARNING:tensorflow:From /home/ec2-user/anaconda3/envs/tensorflow_p36/lib/python
3.6/site-packages/keras/backend/tensorflow_backend.py:4229: to_int64 (from tenso
rflow.python.ops.math_ops) is deprecated and will be removed in a future versio
n.
Instructions for updating:
Use `tf.cast` instead.
WARNING:tensorflow:From /home/ec2-user/anaconda3/envs/tensorflow_p36/lib/python
3.6/site-packages/keras/backend/tensorflow_backend.py:4253: The name tf.log is d
eprecated. Please use tf.math.log instead.

WARNING:tensorflow:From /home/ec2-user/anaconda3/envs/tensorflow_p36/lib/python
3.6/site-packages/keras/optimizers.py:790: The name tf.train.Optimizer is deprec
ated. Please use tf.compat.v1.train.Optimizer instead.

WARNING:tensorflow:From /home/ec2-user/anaconda3/envs/tensorflow_p36/lib/python
3.6/site-packages/keras/backend/tensorflow_backend.py:986: The name tf.assign_ad
d is deprecated. Please use tf.compat.v1.assign_add instead.

WARNING:tensorflow:From /home/ec2-user/anaconda3/envs/tensorflow_p36/lib/python
3.6/site-packages/keras/backend/tensorflow_backend.py:973: The name tf.assign is
deprecated. Please use tf.compat.v1.assign instead.

Epoch 1/20
WARNING:tensorflow:From /home/ec2-user/anaconda3/envs/tensorflow_p36/lib/python
3.6/site-packages/keras/backend/tensorflow_backend.py:174: The name tf.get_defau
lt_session is deprecated. Please use tf.compat.v1.get_default_session instead.

WARNING:tensorflow:From /home/ec2-user/anaconda3/envs/tensorflow_p36/lib/python
3.6/site-packages/keras/backend/tensorflow_backend.py:190: The name tf.global_va
riables is deprecated. Please use tf.compat.v1.global_variables instead.

WARNING:tensorflow:From /home/ec2-user/anaconda3/envs/tensorflow_p36/lib/python
3.6/site-packages/keras/backend/tensorflow_backend.py:199: The name tf.is_variab
le_initialized is deprecated. Please use tf.compat.v1.is_variable_initialized in
stead.

WARNING:tensorflow:From /home/ec2-user/anaconda3/envs/tensorflow_p36/lib/python
3.6/site-packages/keras/backend/tensorflow_backend.py:206: The name tf.variables
_initializer is deprecated. Please use tf.compat.v1.variables_initializer instea
d.

101/101 [==============================] - 277s 3s/step - loss: 286.7486 - val_l
oss: 244.5906
Epoch 2/20
101/101 [==============================] - 279s 3s/step - loss: 214.7585 - val_l
oss: 215.4660
Epoch 3/20
101/101 [==============================] - 280s 3s/step - loss: 182.3231 - val_l
oss: 169.4458
```

```
Epoch 4/20
101/101 [==============================] - 277s 3s/step - loss: 160.7220 - val_l
oss: 166.1831
Epoch 5/20
101/101 [==============================] - 279s 3s/step - loss: 147.8013 - val_l
oss: 153.1725
Epoch 6/20
101/101 [==============================] - 280s 3s/step - loss: 138.8754 - val_l
oss: 146.7674
Epoch 7/20
101/101 [==============================] - 280s 3s/step - loss: 132.5091 - val_l
oss: 142.0255
Epoch 8/20
101/101 [==============================] - 279s 3s/step - loss: 128.4342 - val_l
oss: 141.6528
Epoch 9/20
101/101 [==============================] - 279s 3s/step - loss: 125.5494 - val_l
oss: 140.4157
Epoch 10/20
101/101 [==============================] - 278s 3s/step - loss: 122.8605 - val_l
oss: 139.5184
Epoch 11/20
101/101 [==============================] - 277s 3s/step - loss: 118.7043 - val_l
oss: 137.9088
Epoch 12/20
101/101 [==============================] - 280s 3s/step - loss: 116.4419 - val_l
oss: 135.2083
Epoch 13/20
101/101 [==============================] - 280s 3s/step - loss: 115.1324 - val_l
oss: 136.7850
Epoch 14/20
101/101 [==============================] - 280s 3s/step - loss: 112.6076 - val_l
oss: 132.2179
Epoch 15/20
101/101 [==============================] - 280s 3s/step - loss: 111.5549 - val_l
oss: 139.0041
Epoch 16/20
101/101 [==============================] - 279s 3s/step - loss: 109.4689 - val_l
oss: 135.2965
Epoch 17/20
101/101 [==============================] - 281s 3s/step - loss: 109.5502 - val_l
oss: 133.2925
Epoch 18/20
101/101 [==============================] - 280s 3s/step - loss: 106.1687 - val_l
oss: 130.6368
Epoch 19/20
101/101 [==============================] - 280s 3s/step - loss: 107.4090 - val_l
oss: 130.2043
Epoch 20/20
101/101 [==============================] - 278s 3s/step - loss: 105.2228 - val_l
oss: 132.1762
```

(IMPLEMENTATION) Model 4: Bidirectional RNN + TimeDistributed Dense

Read about the Bidirectional (https://keras.io/layers/wrappers/) wrapper in the Keras documentation. For your next architecture, you will specify an architecture that uses a single bidirectional RNN layer, before a (`TimeDistributed`) dense layer. The added value of a bidirectional RNN is described well in this paper (http://www.cs.toronto.edu/~hinton/absps/DRNN_speech.pdf).

> One shortcoming of conventional RNNs is that they are only able to make use of previous context. In speech recognition, where whole utterances are transcribed at once, there is no reason not to exploit future context as well. Bidirectional RNNs (BRNNs) do this by processing the data in both directions with two separate hidden layers which are then fed forwards to the same output layer.

Before running the code cell below, you must complete the `bidirectional_rnn_model` function in `sample_models.py` . Feel free to use `SimpleRNN` , `LSTM` , or `GRU` units. When specifying the `Bidirectional` wrapper, use `merge_mode='concat'` .

In [10]:

```
model_4 = bidirectional_rnn_model(input_dim=inp_dim, # change to 13 if you would like to use MFCC features
                                  units=200)
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
the_input (InputLayer)       (None, None, 161)         0
_________________________________________________________________
bidirectional_1 (Bidirection (None, None, 400)         434400
_________________________________________________________________
time_distributed_2 (TimeDist (None, None, 29)          11629
_________________________________________________________________
softmax (Activation)         (None, None, 29)          0
=================================================================
Total params: 446,029
Trainable params: 446,029
Non-trainable params: 0
_________________________________________________________________
None
```

Please execute the code cell below to train the neural network you specified in `input_to_softmax` . After the model has finished training, the model is saved (https://keras.io/getting-started/faq/#how-can-i-save-a-keras-model) in the HDF5 file `model_4.h5` . The loss history is saved (https://wiki.python.org/moin/UsingPickle) in `model_4.pickle` . You are welcome to tweak any of the optional parameters while calling the `train_model` function, but this is not required.

In [11]:

```
train_model(input_to_softmax=model_4,
            pickle_path='model_4.pickle',
            save_model_path='model_4.h5',
            spectrogram=is_spectrogram) # change to False if you would like to use MFCC feat
ures
```

```
Epoch 1/20
101/101 [==============================] - 268s 3s/step - loss: 409.4747 - val_l
oss: 363.0292
Epoch 2/20
101/101 [==============================] - 273s 3s/step - loss: 351.6253 - val_l
oss: 414.4104
Epoch 3/20
101/101 [==============================] - 275s 3s/step - loss: 264.9631 - val_l
oss: 229.3404
Epoch 4/20
101/101 [==============================] - 273s 3s/step - loss: 235.5862 - val_l
oss: 220.5213
Epoch 5/20
101/101 [==============================] - 272s 3s/step - loss: 227.3716 - val_l
oss: 216.1990
Epoch 6/20
101/101 [==============================] - 272s 3s/step - loss: 216.5937 - val_l
oss: 208.0499
Epoch 7/20
101/101 [==============================] - 272s 3s/step - loss: 203.2311 - val_l
oss: 194.7064
Epoch 8/20
101/101 [==============================] - 273s 3s/step - loss: 192.7776 - val_l
oss: 185.8051
Epoch 9/20
101/101 [==============================] - 272s 3s/step - loss: 183.8745 - val_l
oss: 178.5403
Epoch 10/20
101/101 [==============================] - 272s 3s/step - loss: 176.6127 - val_l
oss: 176.3437
Epoch 11/20
101/101 [==============================] - 274s 3s/step - loss: 169.8963 - val_l
oss: 171.0576
Epoch 12/20
101/101 [==============================] - 275s 3s/step - loss: 164.2097 - val_l
oss: 167.8033
Epoch 13/20
101/101 [==============================] - 272s 3s/step - loss: 158.7754 - val_l
oss: 162.8397
Epoch 14/20
101/101 [==============================] - 271s 3s/step - loss: 153.6119 - val_l
oss: 158.0490
Epoch 15/20
101/101 [==============================] - 273s 3s/step - loss: 149.2559 - val_l
oss: 159.5662
Epoch 16/20
101/101 [==============================] - 283s 3s/step - loss: 145.3604 - val_l
oss: 156.1159
Epoch 17/20
101/101 [==============================] - 276s 3s/step - loss: 141.4196 - val_l
oss: 155.2330
Epoch 18/20
101/101 [==============================] - 272s 3s/step - loss: 138.4830 - val_l
oss: 154.6619
Epoch 19/20
101/101 [==============================] - 272s 3s/step - loss: 135.7417 - val_l
oss: 148.5893
Epoch 20/20
101/101 [==============================] - 272s 3s/step - loss: 132.7727 - val_l
oss: 151.6678
```

(OPTIONAL IMPLEMENTATION) Models 5+

If you would like to try out more architectures than the ones above, please use the code cell below. Please continue to follow the same convention for saving the models; for the i-th sample model, please save the loss at **`model_i.pickle`** and saving the trained model at **`model_i.h5`** .

In [ ]:

```
## (Optional) TODO: Try out some more models!
### Feel free to use as many code cells as needed.
```

Compare the Models

Execute the code cell below to evaluate the performance of the drafted deep learning models. The training and validation loss are plotted for each model.

In [18]:

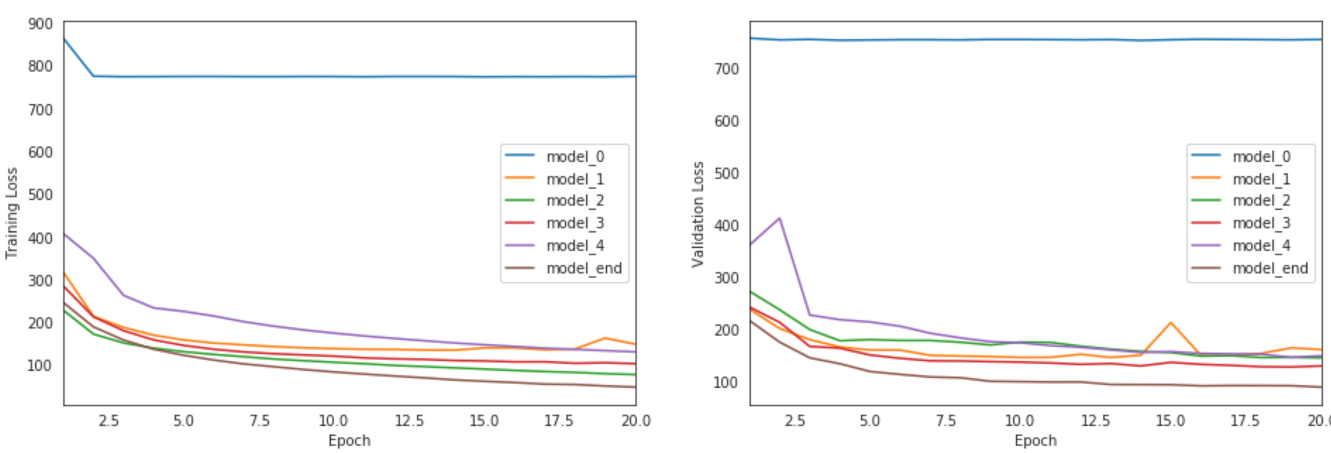
```
from glob import glob
import numpy as np
import _pickle as pickle
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
sns.set_style(style='white')

# obtain the paths for the saved model history
all_pickles = sorted(glob("results/*.pickle"))
# extract the name of each model
model_names = [item[8:-7] for item in all_pickles]
# extract the loss history for each model
valid_loss = [pickle.load( open( i, "rb" ) )['val_loss'] for i in all_pickles]
train_loss = [pickle.load( open( i, "rb" ) )['loss'] for i in all_pickles]
# save the number of epochs used to train each model
num_epochs = [len(valid_loss[i]) for i in range(len(valid_loss))]

fig = plt.figure(figsize=(16,5))

# plot the training loss vs. epoch for each model
ax1 = fig.add_subplot(121)
for i in range(len(all_pickles)):
    ax1.plot(np.linspace(1, num_epochs[i], num_epochs[i]),
            train_loss[i], label=model_names[i])
# clean up the plot
ax1.legend()
ax1.set_xlim([1, max(num_epochs)])
plt.xlabel('Epoch')
plt.ylabel('Training Loss')

# plot the validation loss vs. epoch for each model
ax2 = fig.add_subplot(122)
for i in range(len(all_pickles)):
    ax2.plot(np.linspace(1, num_epochs[i], num_epochs[i]),
            valid_loss[i], label=model_names[i])
# clean up the plot
ax2.legend()
ax2.set_xlim([1, max(num_epochs)])
plt.xlabel('Epoch')
plt.ylabel('Validation Loss')
plt.show()
```



Question 1: Use the plot above to analyze the performance of each of the attempted architectures. Which performs best? Provide an explanation regarding why you think some models perform better than others.

Answer:

Model	Total Params	Performance	Description
0: RNN	16,617		This model has only one RNN layer. We can see that this model **does not have the capacity** to find the complex patterns in the data with its small number of parameters. In my experment, the validation loss stayed steady at 750, which means this model cannot learn through multiple epochs as shown in the **blue** line in the above figures. I guess that increasing the number of layers, increasing the number of hidden units, or adding a fully connected network at the last part can significantly improve the performance of this model.
1: RNN + TimeDistributed Dense	223,829		This model adds a BatchNormalization after GRU and a TimeDistributed Dense layer. This increases the modeling capacity significantly to 223k parameters. In the experiment (**orange** line in the figure), the validation loss stayed at 150-200 range. We can observe that this model **overfitted** after 15 epochs.
2: CNN + RNN + TimeDistributed Dense	442,029	Best Training Loss (among model 0-4)	With Conv1D at the beginning of the network, this model can utilize information from long-distance input samples. "How long" factor is determined by kernel_size and can be optimized for better performance (in my final model). Although this model shows **the best training loss** (about 80) among model 0-4 (**green** line in the figure) , it suffers from **overfitting** (as noticed in the right validation loss figure), which can be reduced by regularization techniques such as Dropout.
3: Deeper RNN + TimeDistributed Dense	465,229	Best Validation Loss (among model 0-4)	With this model, I added more RNN layers. As shown in the deep_rnn_model() code of simple_models.py, I used for-loop to construct RNN with BatchNormalization layers with variable number of layers. This model (**red** line in the figure) showed **the best validation loss** (about 130) among model 0-4. The modeling capacity of this model from **multiple RNN layers** outperformed other approaches.
4: Bidirectional RNN + TimeDistributed Dense	446,029		This model has **only one bidirectional RNN layer** without batch normalization. As shown in the **purple** line in the figure, the performance of this model is worse than multi-layer models such as model 2 or 3. I guess that adding more layers with batch normalization will boost the performance of this model significantly.

- I used Spectrogram for audio feature representation instead of MFCC. From my experiments, the performance differences between the two approaches were not significant.
- In summary, **Model 3** (Deeper RNN + TimeDistributed Dense) performed the best among model 0-4. Based on these results, I will propose a final model for better performance.

(IMPLEMENTATION) Final Model

Now that you've tried out many sample models, use what you've learned to draft your own architecture! While your final acoustic model should not be identical to any of the architectures explored above, you are welcome to merely combine the explored layers above into a deeper architecture. It is **NOT** necessary to include new layer types that were not explored in the notebook.

However, if you would like some ideas for even more layer types, check out these ideas for some additional, optional extensions to your model:

- If you notice your model is overfitting to the training dataset, consider adding **dropout**! To add dropout to [recurrent layers (https://faroit.github.io/keras-docs/1.0.2/layers/recurrent/)](https://faroit.github.io/keras-docs/1.0.2/layers/recurrent/), pay special attention to the `dropout_W` and `dropout_U` arguments. This [paper (http://arxiv.org/abs/1512.05287)](http://arxiv.org/abs/1512.05287) may also provide some interesting theoretical background.
- If you choose to include a convolutional layer in your model, you may get better results by working with **dilated convolutions**. If you choose to use dilated convolutions, make sure that you are able to accurately calculate the length of the acoustic model's output in the `model.output_length` lambda function. You can read more about dilated convolutions in Google's [WaveNet paper (https://arxiv.org/abs/1609.03499)](https://arxiv.org/abs/1609.03499). For an example of a speech-to-text system that makes use of dilated convolutions, check out this GitHub [repository (https://github.com/buriburisuri/speech-to-text-wavenet)](https://github.com/buriburisuri/speech-to-text-wavenet). You can work with dilated convolutions [in Keras (https://keras.io/layers/convolutional/)](https://keras.io/layers/convolutional/) by paying special attention to the `padding` argument when you specify a convolutional layer.
- If your model makes use of convolutional layers, why not also experiment with adding **max pooling**? Check out [this paper (https://arxiv.org/pdf/1701.02720.pdf)](https://arxiv.org/pdf/1701.02720.pdf) for example architecture that makes use of max pooling in an acoustic model.
- So far, you have experimented with a single bidirectional RNN layer. Consider stacking the bidirectional layers, to produce a [deep bidirectional RNN (https://www.cs.toronto.edu/~graves/asru_2013.pdf)](https://www.cs.toronto.edu/~graves/asru_2013.pdf)!

All models that you specify in this repository should have `output_length` defined as an attribute. This attribute is a lambda function that maps the (temporal) length of the input acoustic features to the (temporal) length of the output softmax layer. This function is used in the computation of CTC loss; to see this, look at the `add_ctc_loss` function in `train_utils.py`. To see where the `output_length` attribute is defined for the models in the code, take a look at the `sample_models.py` file. You will notice this line of code within most models:

```
model.output_length = lambda x: x
```

The acoustic model that incorporates a convolutional layer (`cnn_rnn_model`) has a line that is a bit different:

```
model.output_length = lambda x: cnn_output_length(
        x, kernel_size, conv_border_mode, conv_stride)
```

In the case of models that use purely recurrent layers, the lambda function is the identity function, as the recurrent layers do not modify the (temporal) length of their input tensors. However, convolutional layers are more complicated and require a specialized function (`cnn_output_length` in `sample_models.py`) to determine the temporal length of their output.

You will have to add the `output_length` attribute to your final model before running the code cell below. Feel free to use the `cnn_output_length` function, if it suits your model.

In [8]:

```
# specify the model
model_end = final_model(input_dim=inp_dim, # change to 13 if you would like to use MFCC feat
ures
                        filters=200,
                        kernel_size=7,
                        conv_stride=2,
                        conv_border_mode='valid',
                        units=200,
                        dropout_rate=0.2,
                        recur_layers=4
                        )
```

```
WARNING:tensorflow:From /home/ec2-user/anaconda3/envs/tensorflow_p36/lib/python
3.6/site-packages/keras/backend/tensorflow_backend.py:517: The name tf.placehold
er is deprecated. Please use tf.compat.v1.placeholder instead.

WARNING:tensorflow:From /home/ec2-user/anaconda3/envs/tensorflow_p36/lib/python
3.6/site-packages/keras/backend/tensorflow_backend.py:4138: The name tf.random_u
niform is deprecated. Please use tf.random.uniform instead.

WARNING:tensorflow:From /home/ec2-user/anaconda3/envs/tensorflow_p36/lib/python
3.6/site-packages/keras/backend/tensorflow_backend.py:131: The name tf.get_defau
lt_graph is deprecated. Please use tf.compat.v1.get_default_graph instead.

WARNING:tensorflow:From /home/ec2-user/anaconda3/envs/tensorflow_p36/lib/python
3.6/site-packages/keras/backend/tensorflow_backend.py:133: The name tf.placehold
er_with_default is deprecated. Please use tf.compat.v1.placeholder_with_default
instead.

WARNING:tensorflow:From /home/ec2-user/anaconda3/envs/tensorflow_p36/lib/python
3.6/site-packages/keras/backend/tensorflow_backend.py:3445: calling dropout (fro
m tensorflow.python.ops.nn_ops) with keep_prob is deprecated and will be removed
in a future version.
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 - keep
_prob`.
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
the_input (InputLayer)       (None, None, 161)         0
_________________________________________________________________
conv1d (Conv1D)              (None, None, 200)         225600
_________________________________________________________________
bn_conv_1d (BatchNormalizati (None, None, 200)         800
_________________________________________________________________
activation_1 (Activation)    (None, None, 200)         0
_________________________________________________________________
dropout_1 (Dropout)          (None, None, 200)         0
_________________________________________________________________
bidirectional_1 (Bidirection (None, None, 400)         481200
_________________________________________________________________
batch_normalization_1 (Batch (None, None, 400)         1600
_________________________________________________________________
dropout_2 (Dropout)          (None, None, 400)         0
_________________________________________________________________
bidirectional_2 (Bidirection (None, None, 400)         721200
_________________________________________________________________
batch_normalization_2 (Batch (None, None, 400)         1600
_________________________________________________________________
dropout_3 (Dropout)          (None, None, 400)         0
_________________________________________________________________
bidirectional_3 (Bidirection (None, None, 400)         721200
_________________________________________________________________
batch_normalization_3 (Batch (None, None, 400)         1600
_________________________________________________________________
dropout_4 (Dropout)          (None, None, 400)         0
_________________________________________________________________
bidirectional_4 (Bidirection (None, None, 400)         721200
_________________________________________________________________
batch_normalization_4 (Batch (None, None, 400)         1600
_________________________________________________________________
time_distributed_1 (TimeDist (None, None, 29)          11629
_________________________________________________________________
```

```
softmax (Activation)         (None, None, 29)          0
=================================================================
Total params: 2,889,229
Trainable params: 2,885,629
Non-trainable params: 3,600
_________________________________________________________________
None
```

Please execute the code cell below to train the neural network you specified in `input_to_softmax`. After the model has finished training, the model is [saved (https://keras.io/getting-started/faq/#how-can-i-save-a-keras-model)](https://keras.io/getting-started/faq/#how-can-i-save-a-keras-model) in the HDF5 file `model_end.h5`. The loss history is [saved (https://wiki.python.org/moin/UsingPickle)](https://wiki.python.org/moin/UsingPickle) in `model_end.pickle`. You are welcome to tweak any of the optional parameters while calling the `train_model` function, but this is not required.

In [9]:

```
from keras.optimizers import Adam

train_model(input_to_softmax=model_end,
            pickle_path='model_end.pickle',
            save_model_path='model_end.h5',
            optimizer=Adam(0.001),
            spectrogram=is_spectrogram) # change to False if you would like to use MFCC feat
ures
```

```
WARNING:tensorflow:From /home/ec2-user/anaconda3/envs/tensorflow_p36/lib/python
3.6/site-packages/keras/backend/tensorflow_backend.py:4249: to_int32 (from tenso
rflow.python.ops.math_ops) is deprecated and will be removed in a future versio
n.
Instructions for updating:
Use `tf.cast` instead.
WARNING:tensorflow:From /home/ec2-user/anaconda3/envs/tensorflow_p36/lib/python
3.6/site-packages/tensorflow_core/python/ops/array_ops.py:1475: where (from tens
orflow.python.ops.array_ops) is deprecated and will be removed in a future versi
on.
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
WARNING:tensorflow:From /home/ec2-user/anaconda3/envs/tensorflow_p36/lib/python
3.6/site-packages/keras/backend/tensorflow_backend.py:4229: to_int64 (from tenso
rflow.python.ops.math_ops) is deprecated and will be removed in a future versio
n.
Instructions for updating:
Use `tf.cast` instead.
WARNING:tensorflow:From /home/ec2-user/anaconda3/envs/tensorflow_p36/lib/python
3.6/site-packages/keras/backend/tensorflow_backend.py:4253: The name tf.log is d
eprecated. Please use tf.math.log instead.

WARNING:tensorflow:From /home/ec2-user/anaconda3/envs/tensorflow_p36/lib/python
3.6/site-packages/keras/optimizers.py:790: The name tf.train.Optimizer is deprec
ated. Please use tf.compat.v1.train.Optimizer instead.

WARNING:tensorflow:From /home/ec2-user/anaconda3/envs/tensorflow_p36/lib/python
3.6/site-packages/keras/backend/tensorflow_backend.py:986: The name tf.assign_ad
d is deprecated. Please use tf.compat.v1.assign_add instead.

WARNING:tensorflow:From /home/ec2-user/anaconda3/envs/tensorflow_p36/lib/python
3.6/site-packages/keras/backend/tensorflow_backend.py:973: The name tf.assign is
deprecated. Please use tf.compat.v1.assign instead.

Epoch 1/20
WARNING:tensorflow:From /home/ec2-user/anaconda3/envs/tensorflow_p36/lib/python
3.6/site-packages/keras/backend/tensorflow_backend.py:174: The name tf.get_defau
lt_session is deprecated. Please use tf.compat.v1.get_default_session instead.

WARNING:tensorflow:From /home/ec2-user/anaconda3/envs/tensorflow_p36/lib/python
3.6/site-packages/keras/backend/tensorflow_backend.py:190: The name tf.global_va
riables is deprecated. Please use tf.compat.v1.global_variables instead.

WARNING:tensorflow:From /home/ec2-user/anaconda3/envs/tensorflow_p36/lib/python
3.6/site-packages/keras/backend/tensorflow_backend.py:199: The name tf.is_variab
le_initialized is deprecated. Please use tf.compat.v1.is_variable_initialized in
stead.

WARNING:tensorflow:From /home/ec2-user/anaconda3/envs/tensorflow_p36/lib/python
3.6/site-packages/keras/backend/tensorflow_backend.py:206: The name tf.variables
_initializer is deprecated. Please use tf.compat.v1.variables_initializer instea
d.

101/101 [==============================] - 564s 6s/step - loss: 248.1270 - val_l
oss: 218.7572
Epoch 2/20
101/101 [==============================] - 559s 6s/step - loss: 191.3083 - val_l
oss: 177.6582
Epoch 3/20
101/101 [==============================] - 562s 6s/step - loss: 160.4549 - val_l
oss: 147.6792
```

```
Epoch 4/20
101/101 [==============================] - 563s 6s/step - loss: 138.9748 - val_l
oss: 136.4192
Epoch 5/20
101/101 [==============================] - 560s 6s/step - loss: 124.7880 - val_l
oss: 121.5731
Epoch 6/20
101/101 [==============================] - 566s 6s/step - loss: 113.6764 - val_l
oss: 116.0945
Epoch 7/20
101/101 [==============================] - 557s 6s/step - loss: 104.5944 - val_l
oss: 111.1734
Epoch 8/20
101/101 [==============================] - 563s 6s/step - loss: 98.0297 - val_lo
ss: 109.7098
Epoch 9/20
101/101 [==============================] - 563s 6s/step - loss: 91.4321 - val_lo
ss: 102.8987
Epoch 10/20
101/101 [==============================] - 565s 6s/step - loss: 85.7002 - val_lo
ss: 102.2668
Epoch 11/20
101/101 [==============================] - 562s 6s/step - loss: 80.8976 - val_lo
ss: 101.5173
Epoch 12/20
101/101 [==============================] - 561s 6s/step - loss: 76.2315 - val_lo
ss: 101.5946
Epoch 13/20
101/101 [==============================] - 564s 6s/step - loss: 72.0080 - val_lo
ss: 96.7778
Epoch 14/20
101/101 [==============================] - 563s 6s/step - loss: 67.2627 - val_lo
ss: 96.6097
Epoch 15/20
101/101 [==============================] - 560s 6s/step - loss: 64.0360 - val_lo
ss: 96.3504
Epoch 16/20
101/101 [==============================] - 564s 6s/step - loss: 61.0853 - val_lo
ss: 94.2196
Epoch 17/20
101/101 [==============================] - 560s 6s/step - loss: 57.3232 - val_lo
ss: 94.7048
Epoch 18/20
101/101 [==============================] - 563s 6s/step - loss: 56.4837 - val_lo
ss: 94.6473
Epoch 19/20
101/101 [==============================] - 565s 6s/step - loss: 53.0156 - val_lo
ss: 94.4202
Epoch 20/20
101/101 [==============================] - 560s 6s/step - loss: 50.4019 - val_lo
ss: 91.8004
```

Question 2: Describe your final model architecture and your reasoning at each step.

Answer:

Model	Total Params	Performance	Description
Final	2,889,229	Best of all models (training loss: 50.4, validation loss: 91.8)	My final model is based on the above model 2, 3, and 4 with a few additional techniques. - **1D Convolution**: this layer is reused from model 2, but this time I used **kernel_size=7** instead of 11 (model 2) because information captured by too big kernel (= information from too far samples) will not be very useful. - **Batch Normalization**: I used Batch Normalization after all CNN and RNN layers for faster learning by reducing the covariate shift. I arranged the order of layers as suggested by Stanford CS231n: **CNN -> Batch Normalization -> Activation -> Dropout** - **Dropout**: I used Dropout to avoid overfitting with **rate=0.2** after CNN and RNN layers. - **Bidirectional RNN layers**: I used **4 layers** of Bidirectional RNN (GRU) to capture rich context information from temporal dependencies. - **TimeDistributed Dense**: this layer is reused from model 2, 3, and 4. - **Adam optimizer**: I used Adam optimizer instead of the default SGD for faster convergence. **Learning rate of 0.001** was chosen by trial and error experiments. As a result, this model achieves the **training loss of 50.4** and the **validation loss of about 91.8**, which is the best performance of all models in this project.

STEP 3: Obtain Predictions

We have written a function for you to decode the predictions of your acoustic model. To use the function, please execute the code cell below.

In [12]:

```
import numpy as np
from data_generator import AudioGenerator
from keras import backend as K
from utils import int_sequence_to_text
from IPython.display import Audio

def get_predictions(index, partition, input_to_softmax, model_path):
    """ Print a model's decoded predictions
    Params:
        index (int): The example you would like to visualize
        partition (str): One of 'train' or 'validation'
        input_to_softmax (Model): The acoustic model
        model_path (str): Path to saved acoustic model's weights
    """
    # load the train and test data
    data_gen = AudioGenerator()
    data_gen.load_train_data()
    data_gen.load_validation_data()

    # obtain the true transcription and the audio features
    if partition == 'validation':
        transcr = data_gen.valid_texts[index]
        audio_path = data_gen.valid_audio_paths[index]
        data_point = data_gen.normalize(data_gen.featurize(audio_path))
    elif partition == 'train':
        transcr = data_gen.train_texts[index]
        audio_path = data_gen.train_audio_paths[index]
        data_point = data_gen.normalize(data_gen.featurize(audio_path))
    else:
        raise Exception('Invalid partition!  Must be "train" or "validation"')

    # obtain and decode the acoustic model's predictions
    input_to_softmax.load_weights(model_path)
    prediction = input_to_softmax.predict(np.expand_dims(data_point, axis=0))
    output_length = [input_to_softmax.output_length(data_point.shape[0])]
    pred_ints = (K.eval(K.ctc_decode(
                prediction, output_length)[0][0])+1).flatten().tolist()

    # play the audio file, and display the true and predicted transcriptions
    print('-'*80)
    Audio(audio_path)
    print('True transcription:\n' + '\n' + transcr)
    print('-'*80)
    print('Predicted transcription:\n' + '\n' + ''.join(int_sequence_to_text(pred_ints)))
    print('-'*80)
```

Use the code cell below to obtain the transcription predicted by your final model for the first example in the training dataset.

In [16]:

```
get_predictions(index=0,
                partition='train',
                input_to_softmax=final_model(input_dim=inp_dim, # change to 13 if you would
 like to use MFCC features
                        filters=200,
                        kernel_size=7,
                        conv_stride=2,
                        conv_border_mode='valid',
                        units=200,
                        dropout_rate=0.2,
                        recur_layers=4
                        ),
                model_path='results/model_end.h5')
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
the_input (InputLayer)       (None, None, 161)         0         
_________________________________________________________________
conv1d (Conv1D)              (None, None, 200)         225600    
_________________________________________________________________
bn_conv_1d (BatchNormalizati (None, None, 200)         800       
_________________________________________________________________
activation_4 (Activation)    (None, None, 200)         0         
_________________________________________________________________
dropout_16 (Dropout)         (None, None, 200)         0         
_________________________________________________________________
bidirectional_13 (Bidirectio (None, None, 400)         481200    
_________________________________________________________________
batch_normalization_13 (Batc (None, None, 400)         1600      
_________________________________________________________________
dropout_17 (Dropout)         (None, None, 400)         0         
_________________________________________________________________
bidirectional_14 (Bidirectio (None, None, 400)         721200    
_________________________________________________________________
batch_normalization_14 (Batc (None, None, 400)         1600      
_________________________________________________________________
dropout_18 (Dropout)         (None, None, 400)         0         
_________________________________________________________________
bidirectional_15 (Bidirectio (None, None, 400)         721200    
_________________________________________________________________
batch_normalization_15 (Batc (None, None, 400)         1600      
_________________________________________________________________
dropout_19 (Dropout)         (None, None, 400)         0         
_________________________________________________________________
bidirectional_16 (Bidirectio (None, None, 400)         721200    
_________________________________________________________________
batch_normalization_16 (Batc (None, None, 400)         1600      
_________________________________________________________________
time_distributed_4 (TimeDist (None, None, 29)          11629     
_________________________________________________________________
softmax (Activation)         (None, None, 29)          0         
=================================================================
Total params: 2,889,229
Trainable params: 2,885,629
Non-trainable params: 3,600
_________________________________________________________________
None
WARNING:tensorflow:From /home/ec2-user/anaconda3/envs/tensorflow_p36/lib/python
3.6/site-packages/keras/backend/tensorflow_backend.py:4303: sparse_to_dense (fro
m tensorflow.python.ops.sparse_ops) is deprecated and will be removed in a futur
e version.
Instructions for updating:
Create a `tf.sparse.SparseTensor` and use `tf.sparse.to_dense` instead.
--------------------------------------------------------------------------------
True transcription:

her father is a most remarkable person to say the least
--------------------------------------------------------------------------------
Predicted transcription:

her fathere s a mos remarkable person to say the least
--------------------------------------------------------------------------------
```

Use the next code cell to visualize the model's prediction for the first example in the validation dataset.

In [17]:

```
get_predictions(index=0,
                partition='validation',
                input_to_softmax=final_model(input_dim=inp_dim, # change to 13 if you would
 like to use MFCC features
                        filters=200,
                        kernel_size=7,
                        conv_stride=2,
                        conv_border_mode='valid',
                        units=200,
                        dropout_rate=0.2,
                        recur_layers=4
                        ),
                model_path='results/model_end.h5')
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
the_input (InputLayer)       (None, None, 161)         0         
_________________________________________________________________
conv1d (Conv1D)              (None, None, 200)         225600    
_________________________________________________________________
bn_conv_1d (BatchNormalizati (None, None, 200)         800       
_________________________________________________________________
activation_5 (Activation)    (None, None, 200)         0         
_________________________________________________________________
dropout_21 (Dropout)         (None, None, 200)         0         
_________________________________________________________________
bidirectional_17 (Bidirectio (None, None, 400)         481200    
_________________________________________________________________
batch_normalization_17 (Batc (None, None, 400)         1600      
_________________________________________________________________
dropout_22 (Dropout)         (None, None, 400)         0         
_________________________________________________________________
bidirectional_18 (Bidirectio (None, None, 400)         721200    
_________________________________________________________________
batch_normalization_18 (Batc (None, None, 400)         1600      
_________________________________________________________________
dropout_23 (Dropout)         (None, None, 400)         0         
_________________________________________________________________
bidirectional_19 (Bidirectio (None, None, 400)         721200    
_________________________________________________________________
batch_normalization_19 (Batc (None, None, 400)         1600      
_________________________________________________________________
dropout_24 (Dropout)         (None, None, 400)         0         
_________________________________________________________________
bidirectional_20 (Bidirectio (None, None, 400)         721200    
_________________________________________________________________
batch_normalization_20 (Batc (None, None, 400)         1600      
_________________________________________________________________
time_distributed_5 (TimeDist (None, None, 29)          11629     
_________________________________________________________________
softmax (Activation)         (None, None, 29)          0         
=================================================================
Total params: 2,889,229
Trainable params: 2,885,629
Non-trainable params: 3,600
_________________________________________________________________
None
--------------------------------------------------------------------------------
True transcription:

the bogus legislature numbered thirty six members
--------------------------------------------------------------------------------
Predicted transcription:

the bidles lyjeslaured nuberds gerty six limbers
--------------------------------------------------------------------------------
```

One standard way to improve the results of the decoder is to incorporate a language model. We won't pursue this in the notebook, but you are welcome to do so as an *optional extension*.

If you are interested in creating models that provide improved transcriptions, you are encouraged to download more data (http://www.openslr.org/12/) and train bigger, deeper models. But beware - the model will likely take a long while to train. For instance, training this state-of-the-art (https://arxiv.org/pdf/1512.02595v1.pdf) model would take 3-6 weeks on a single GPU!