```
================================================================================
====================
📄 File: .gitignore
📁 Path: .gitignore
--------------------------------------------------------------------------------
--------------------
__pycache__/
.cache/
*.pyc
.DS_Store
run*.sh

================================================================================
====================

================================================================================
====================
📄 File: demo_server.py
📁 Path: demo_server.py
--------------------------------------------------------------------------------
--------------------
# import argparse
# import falcon
# from hparams import hparams, hparams_debug_string
# import os
# from synthesizer import Synthesizer


# html_body = '''<html><title>Demo</title>
# <style>
# body {padding: 16px; font-family: sans-serif; font-size: 14px; color: #444}
# input {font-size: 14px; padding: 8px 12px; outline: none; border: 1px solid
#ddd}
# input:focus {box-shadow: 0 1px 2px rgba(0,0,0,.15)}
# p {padding: 12px}
# button {background: #28d; padding: 9px 14px; margin-left: 8px; border: none;
outline: none;
#         color: #fff; font-size: 14px; border-radius: 4px; cursor: pointer;}
# button:hover {box-shadow: 0 1px 2px rgba(0,0,0,.15); opacity: 0.9;}
# button:active {background: #29f;}
# button[disabled] {opacity: 0.4; cursor: default}
# </style>
# <body>
# <form>
#   <input id="text" type="text" size="40" placeholder="Enter Text">
#   <button id="button" name="synthesize">Speak</button>
# </form>
# <p id="message"></p>
# <audio id="audio" controls autoplay hidden></audio>
# <script>
# function q(selector) {return document.querySelector(selector)}
# q('#text').focus()
# q('#button').addEventListener('click', function(e) {
#   text = q('#text').value.trim()
#   if (text) {
#     q('#message').textContent = 'Synthesizing...'
#     q('#button').disabled = true
#     q('#audio').hidden = true
#     synthesize(text)
#   }
#   e.preventDefault()
#   return false
# })
# function synthesize(text) {
#   fetch('/synthesize?text=' + encodeURIComponent(text), {cache: 'no-cache'})
#     .then(function(res) {
#       if (!res.ok) throw Error(res.statusText)
#       return res.blob()
#     }).then(function(blob) {
#       q('#message').textContent = ''
#       q('#button').disabled = false
#       q('#audio').src = URL.createObjectURL(blob)
#       q('#audio').hidden = false
#     }).catch(function(err) {
#       q('#message').textContent = 'Error: ' + err.message
#       q('#button').disabled = false
#     })
# }
# </script></body></html>
# '''


# class UIResource:
#   def on_get(self, req, res):
#     res.content_type = 'text/html'
#     res.body = html_body


# class SynthesisResource:
#   def on_get(self, req, res):
#     if not req.params.get('text'):
#       raise falcon.HTTPBadRequest()
#     res.data = synthesizer.synthesize(req.params.get('text'))
#     res.content_type = 'audio/wav'


# synthesizer = Synthesizer()
# api = falcon.API()
# api.add_route('/synthesize', SynthesisResource())
# api.add_route('/', UIResource())


# if __name__ == '__main__':
#   from wsgiref import simple_server
#   parser = argparse.ArgumentParser()
#   parser.add_argument('--checkpoint', required=True, help='Full path to model
checkpoint')
#   parser.add_argument('--port', type=int, default=9000)
#   parser.add_argument('--hparams', default='',
#     help='Hyperparameter overrides as a comma-separated list of name=value
pairs')
#   args = parser.parse_args()
#   os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'
#   hparams.parse(args.hparams)
#   print(hparams_debug_string())
#   synthesizer.load(args.checkpoint)
#   print('Serving on port %d' % args.port)
#   simple_server.make_server('0.0.0.0', args.port, api).serve_forever()
# else:
#   synthesizer.load(os.environ['CHECKPOINT'])

import argparse
import falcon
from hparams import hparams, hparams_debug_string
import os
from synthesizer import Synthesizer
```

```
html_body = '''<html><title>Demo</title>
<style>
body {padding: 16px; font-family: sans-serif; font-size: 14px; color: #444}
input {font-size: 14px; padding: 8px 12px; outline: none; border: 1px solid
#ddd}
input:focus {box-shadow: 0 1px 2px rgba(0,0,0,.15)}
p {padding: 12px}
button {background: #28d; padding: 9px 14px; margin-left: 8px; border: none;
outline: none;
        color: #fff; font-size: 14px; border-radius: 4px; cursor: pointer;}
button:hover {box-shadow: 0 1px 2px rgba(0,0,0,.15); opacity: 0.9;}
button:active {background: #29f;}
button[disabled] {opacity: 0.4; cursor: default}
</style>
<body>
<form>
  <input id="text" type="text" size="40" placeholder="Enter Text">
  <button id="button" name="synthesize">Speak</button>
</form>
<p id="message"></p>
<audio id="audio" controls autoplay hidden></audio>
<script>
function q(selector) {return document.querySelector(selector)}
q('#text').focus()
q('#button').addEventListener('click', function(e) {
  text = q('#text').value.trim()
  if (text) {
    q('#message').textContent = 'Synthesizing...'
    q('#button').disabled = true
    q('#audio').hidden = true
    synthesize(text)
  }
  e.preventDefault()
  return false
})
function synthesize(text) {
  fetch('/synthesize?text=' + encodeURIComponent(text), {cache: 'no-cache'})
    .then(function(res) {
      if (!res.ok) throw Error(res.statusText)
      return res.blob()
    }).then(function(blob) {
      q('#message').textContent = ''
      q('#button').disabled = false
      q('#audio').src = URL.createObjectURL(blob)
      q('#audio').hidden = false
    }).catch(function(err) {
      q('#message').textContent = 'Error: ' + err.message
      q('#button').disabled = false
    })
}
</script></body></html>
'''

class UIResource:
    def on_get(self, req, resp):
        resp.content_type = falcon.MEDIA_HTML
        resp.text = html_body

class SynthesisResource:
    def on_get(self, req, resp):
        if not req.params.get('text'):
            raise falcon.HTTPBadRequest()
        resp.data = synthesizer.synthesize(req.params.get('text'))
        resp.content_type = falcon.MEDIA_WAV
```

```
synthesizer = Synthesizer()
app = falcon.App()  # Updated to new Falcon API
app.add_route('/synthesize', SynthesisResource())
app.add_route('/', UIResource())

if __name__ == '__main__':
    from wsgiref import simple_server
    parser = argparse.ArgumentParser()
    parser.add_argument('--checkpoint', required=True, help='Full path to model
checkpoint')
    parser.add_argument('--port', type=int, default=9000)
    parser.add_argument('--hparams', default='',
                        help='Hyperparameter overrides as a comma-separated list
of name=value pairs')
    args = parser.parse_args()
    os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'
    hparams.parse(args.hparams)
    print(hparams_debug_string())
    synthesizer.load(args.checkpoint)
    print('Serving on port %d' % args.port)
    simple_server.make_server('0.0.0.0', args.port, app).serve_forever()
else:
    synthesizer.load(os.environ['CHECKPOINT'])
================================================================================
====================

================================================================================
====================
📄 File: eval.py
📁 Path: eval.py
--------------------------------------------------------------------------------
--------------------
import argparse
import os
import re
from hparams import hparams, hparams_debug_string
from synthesizer import Synthesizer


sentences = [
  # From July 8, 2017 New York Times:
  'Scientists at the CERN laboratory say they have discovered a new particle.',
  'There’s a way to measure the acute emotional intelligence that has never gone
out of style.',
  'President Trump met with other leaders at the Group of 20 conference.',
  'The Senate\'s bill to repeal and replace the Affordable Care Act is now
imperiled.',
  # From Google's Tacotron example page:
  'Generative adversarial network or variational auto-encoder.',
  'The buses aren\'t the problem, they actually provide a solution.',
  'Does the quick brown fox jump over the lazy dog?',
  'Talib Kweli confirmed to AllHipHop that he will be releasing an album in the
next year.',
]


def get_output_base_path(checkpoint_path):
  base_dir = os.path.dirname(checkpoint_path)
  m = re.compile(r'.*?\.ckpt\-([0-9]+)').match(checkpoint_path)
  name = 'eval-%d' % int(m.group(1)) if m else 'eval'
  return os.path.join(base_dir, name)
```

```
def run_eval(args):
  print(hparams_debug_string())
  synth = Synthesizer()
  synth.load(args.checkpoint)
  base_path = get_output_base_path(args.checkpoint)
  for i, text in enumerate(sentences):
    path = '%s-%d.wav' % (base_path, i)
    print('Synthesizing: %s' % path)
    with open(path, 'wb') as f:
      f.write(synth.synthesize(text))


def main():
  parser = argparse.ArgumentParser()
  parser.add_argument('--checkpoint', required=True, help='Path to model
checkpoint')
  parser.add_argument('--hparams', default='',
    help='Hyperparameter overrides as a comma-separated list of name=value
pairs')
  args = parser.parse_args()
  os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'
  hparams.parse(args.hparams)
  run_eval(args)


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

==
====================

==
====================
📄 File: hparams.py
📁 Path: hparams.py


```
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()

import tensorflow as tf


# Default hyperparameters:
hparams = tf.contrib.training.HParams(
  # Comma-separated list of cleaners to run on text prior to training and eval.
For non-English
  # text, you may want to use "basic_cleaners" or "transliteration_cleaners" See
TRAINING_DATA.md.
  cleaners='transliteration_cleaners',

  # Audio:
  num_mels=80,
  num_freq=1025,
  sample_rate=20000,
  frame_length_ms=50,
  frame_shift_ms=12.5,
  preemphasis=0.97,
  min_level_db=-100,
  ref_level_db=20,

  # Model:
  outputs_per_step=5,
  embed_depth=256,
  prenet_depths=[256, 128],
  encoder_depth=256,
  postnet_depth=256,
  attention_depth=256,
  decoder_depth=256,

  # Training:
  batch_size=32,
  adam_beta1=0.9,
  adam_beta2=0.999,
  initial_learning_rate=0.002,
  decay_learning_rate=True,
  use_cmudict=False,  # Use CMUDict during training to learn pronunciation of
ARPAbet phonemes

  # Eval:
  max_iters=200,
  griffin_lim_iters=60,
  power=1.5,              # Power to raise magnitudes to prior to Griffin-Lim
)


def hparams_debug_string():
  values = hparams.values()
  hp = ['  %s: %s' % (name, values[name]) for name in sorted(values)]
  return 'Hyperparameters:\n' + '\n'.join(hp)
```

==
====================

==
====================
📄 File: LICENSE
📁 Path: LICENSE

==
====================

==
====================
📄 File: preprocess.py
📁 Path: preprocess.py

```
--------------------
import argparse
import os
from multiprocessing import cpu_count
from tqdm import tqdm
from datasets import blizzard, ljspeech, nepali
from hparams import hparams


def preprocess_blizzard(args):
  in_dir = os.path.join(args.base_dir, 'Blizzard2012')
  out_dir = os.path.join(args.base_dir, args.output)
  os.makedirs(out_dir, exist_ok=True)
  metadata = blizzard.build_from_path(in_dir, out_dir, args.num_workers, 
tqdm=tqdm)
  write_metadata(metadata, out_dir)


def preprocess_ljspeech(args):
  in_dir = os.path.join(args.base_dir, 'LJSpeech-1.1')
  out_dir = os.path.join(args.base_dir, args.output)
  os.makedirs(out_dir, exist_ok=True)
  metadata = ljspeech.build_from_path(in_dir, out_dir, args.num_workers, 
tqdm=tqdm)
  write_metadata(metadata, out_dir)

def preprocess_nepali(args):
  in_dir = os.path.join(args.base_dir, 'nepali')
  out_dir = os.path.join(args.base_dir, args.output)
  os.makedirs(out_dir, exist_ok=True)
  metadata = nepali.build_from_path(in_dir, out_dir, args.num_workers, 
tqdm=tqdm)
  write_metadata(metadata, out_dir)


def write_metadata(metadata, out_dir):
  with open(os.path.join(out_dir, 'train.txt'), 'w', encoding='utf-8') as f:
    for m in metadata:
      f.write('|'.join([str(x) for x in m]) + '\n')
  frames = sum([m[2] for m in metadata])
  hours = frames * hparams.frame_shift_ms / (3600 * 1000)
  print('Wrote %d utterances, %d frames (%.2f hours)' % (len(metadata), frames, 
hours))
  print('Max input length:  %d' % max(len(m[3]) for m in metadata))
  print('Max output length: %d' % max(m[2] for m in metadata))


def main():
  parser = argparse.ArgumentParser()
  parser.add_argument('--base_dir', 
default=os.path.expanduser('~/PycharmProjects/tacotron'))
  parser.add_argument('--output', default='training')
  parser.add_argument('--dataset', required=True, choices=['blizzard', 
'ljspeech', 'nepali'])
  parser.add_argument('--num_workers', type=int, default=cpu_count())
  args = parser.parse_args()
  if args.dataset == 'blizzard':
    preprocess_blizzard(args)
  elif args.dataset == 'ljspeech':
    preprocess_ljspeech(args)
  elif args.dataset == 'nepali':
    preprocess_nepali(args)
```

~~~
if __name__ == "__main__":
  main()

================================================================================
====================

================================================================================
====================
📄 File: README.md
📁 Path: README.md
--------------------------------------------------------------------------------
--------------------
# Tacotron

An implementation of Tacotron speech synthesis in TensorFlow (Modified to use
for Nepali Dataset.)

## Quick Start

### Installing dependencies

1. Install Python 3.

2. Install the latest version of
[TensorFlow](https://www.tensorflow.org/install/) for your platform. For better
   performance, install with GPU support if it's available. This code works with
TensorFlow 1.3 and later.

3. Install requirements:
   ```
   pip install -r requirements.txt
   ```


### Using a pre-trained model

1. **Download and unpack a model**:
   FOR NEPALI:
   Download [NEPALI PRETRAINED MODEL](https://drive.google.com/open?
id=1P6tyIYZiTG2_6wPim4IRGZrNEYadJrU6) on 30, 50 and 75k iterations. For the
better results use the model checkpoint trained on 75k iteration.

2. **Run the demo server**:
   FOR NEPALI DEMO:
   ```
   python3 demo_server.py --checkpoint
<full_path_to_pretrained_model>/model.ckpt-30000
    ```


3. **Point your browser at localhost:9000**
   * Type what you want to synthesize
   * FOR NEPALI USE NEPALI UNICODE SENTENCES.

### Training
1. **Download a speech dataset.**

    * [ne_np_female](https://research.google/tools/datasets/nepali-tts/)
(Creative Commons Attribution Share-Alike)

2. **Unpack the dataset into `~/PycharmProjects/tacotron`**
   For nepali TTS the folder should look like(the extracted folder is renamed to
`nepali` for simplicity)
      ```
   tacotron
~~~

~~~
      |- nepali
          |- line_index.tsv
          |- wavs
   ```

3. **Preprocess the data**
   ```
   python3 preprocess.py --dataset nepali
   ```
     * For nepali dataset [hparams.py](hparams.py) is set to
`cleaners='transliteration_cleaners'`.
      If you are using other dataset, change it to default.

4. **Train a model**
   ```
   python3 train.py
   ```

   Tunable hyperparameters are found in [hparams.py](hparams.py). You can adjust
these at the command
   line using the `--hparams` flag, for example `--
hparams="batch_size=16,outputs_per_step=2"`.
   Hyperparameters should generally be set to the same values at both training
and eval time.

   * For nepali dataset use `python3 train.py --hparams="max_iters=300"`. See
`Notes and Common Issues` for details.


5. **Monitor with Tensorboard** (optional)
   ```
   tensorboard --logdir ~/PycharmProjects/tacotron/logs-tacotron
   ```

   The trainer dumps audio and alignments every 1000 steps. You can find these
in
   `~/PycharmProjects/tacotron/logs-tacotron`.

6. **Synthesize from a checkpoint**
   ```
   python3 demo_server.py --checkpoint ~/PycharmProjects/tacotron/logs-
tacotron/model.ckpt-50000
   ```
   Replace "50000" with the checkpoint number that you want to use, then open a
browser
   to `localhost:9000` and type what you want to speak. Alternately, you can
   run [eval.py](eval.py) at the command line:
   ```
   python3 eval.py --checkpoint
~/PycharmProjects/tacotron/logs-tacotron/model.ckpt-50000
   ```
   If you set the `--hparams` flag when training, set the same value here.


## Notes and Common Issues

  * During eval and training, audio length is limited to `max_iters *
outputs_per_step * frame_shift_ms`
    milliseconds. With the defaults (max_iters=200, outputs_per_step=5,
frame_shift_ms=12.5), this is
    12.5 seconds.

    If your training examples are longer, you will see an error like this:
    `Incompatible shapes: [32,1340,80] vs. [32,1000,80]`
~~~

```
    To fix this, you can set a larger value of `max_iters` by passing `--
hparams="max_iters=300"` to
    train.py (replace "300" with a value based on how long your audio is and the
formula above).

  * In this fork the basedir should be noted as `~/PycharmProjects/tacotron`,
please modify path location according to your cloning dir.

### For more information please refer to [original
implementation](https://github.com/keithito/tacotron)


================================================================================
====================

================================================================================
====================
📄 File: requirements.txt
📁 Path: requirements.txt
--------------------------------------------------------------------------------
--------------------
# # Note: this doesn't include tensorflow or tensorflow-gpu because the package
you need to install
# # depends on your platform. It is assumed you have already installed
tensorflow.
# falcon==1.2.0
# inflect==0.2.5
# librosa==0.5.1
# matplotlib==2.0.2
# numpy==1.14.3
# scipy==0.19.0
# tqdm==4.11.2
# Unidecode==0.4.20
tensorflow==2.17.0  # Provides tensorflow.compat.v1 for TF 1.x APIs
falcon==4.0.1       # Latest version of Falcon
inflect==7.3.1      # Latest version for number-to-word conversion
librosa==0.10.2     # Latest stable version for audio processing
matplotlib==3.9.2   # Latest version for plotting
numpy==1.26.4       # Latest version compatible with TensorFlow 2.17
scipy==1.14.1       # Latest version compatible with librosa
tqdm==4.66.5        # Latest version for progress bars
Unidecode==1.3.8    # Latest version for ASCII transliteration
================================================================================
====================

================================================================================
====================
📄 File: synthesizer.py
📁 Path: synthesizer.py
--------------------------------------------------------------------------------
--------------------
import io
import numpy as np

import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()

import tensorflow as tf
from hparams import hparams
from librosa import effects
from models import create_model
from text import text_to_sequence
from util import audio
```

```
class Synthesizer:
  def load(self, checkpoint_path, model_name='tacotron'):
    print('Constructing model: %s' % model_name)
    inputs = tf.placeholder(tf.int32, [1, None], 'inputs')
    input_lengths = tf.placeholder(tf.int32, [1], 'input_lengths')
    with tf.variable_scope('model') as scope:
      self.model = create_model(model_name, hparams)
      self.model.initialize(inputs, input_lengths)
      self.wav_output =
audio.inv_spectrogram_tensorflow(self.model.linear_outputs[0])

    print('Loading checkpoint: %s' % checkpoint_path)
    self.session = tf.Session()
    self.session.run(tf.global_variables_initializer())
    saver = tf.train.Saver()
    saver.restore(self.session, checkpoint_path)


  def synthesize(self, text):
    cleaner_names = [x.strip() for x in hparams.cleaners.split(',')]
    seq = text_to_sequence(text, cleaner_names)
    feed_dict = {
      self.model.inputs: [np.asarray(seq, dtype=np.int32)],
      self.model.input_lengths: np.asarray([len(seq)], dtype=np.int32)
    }
    wav = self.session.run(self.wav_output, feed_dict=feed_dict)
    wav = audio.inv_preemphasis(wav)
    wav = wav[:audio.find_endpoint(wav)]
    out = io.BytesIO()
    audio.save_wav(wav, out)
    return out.getvalue()

================================================================================
====================

================================================================================
====================
📄 File: train.py
📁 Path: train.py
--------------------------------------------------------------------------------
--------------------
import argparse
from datetime import datetime
import math
import os
import subprocess
import time

import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()

import tensorflow as tf
import traceback

from datasets.datafeeder import DataFeeder
from hparams import hparams, hparams_debug_string
from models import create_model
from text import sequence_to_text
from util import audio, infolog, plot, ValueWindow
log = infolog.log
```

```
def get_git_commit():
  subprocess.check_output(['git', 'diff-index', '--quiet', 'HEAD'])   # Verify
client is clean
  commit = subprocess.check_output(['git', 'rev-parse',
'HEAD']).decode().strip()[:10]
  log('Git commit: %s' % commit)
  return commit


def add_stats(model):
  with tf.variable_scope('stats') as scope:
    tf.summary.histogram('linear_outputs', model.linear_outputs)
    tf.summary.histogram('linear_targets', model.linear_targets)
    tf.summary.histogram('mel_outputs', model.mel_outputs)
    tf.summary.histogram('mel_targets', model.mel_targets)
    tf.summary.scalar('loss_mel', model.mel_loss)
    tf.summary.scalar('loss_linear', model.linear_loss)
    tf.summary.scalar('learning_rate', model.learning_rate)
    tf.summary.scalar('loss', model.loss)
    gradient_norms = [tf.norm(grad) for grad in model.gradients]
    tf.summary.histogram('gradient_norm', gradient_norms)
    tf.summary.scalar('max_gradient_norm', tf.reduce_max(gradient_norms))
    return tf.summary.merge_all()


def time_string():
  return datetime.now().strftime('%Y-%m-%d %H:%M')


def train(log_dir, args):
  commit = get_git_commit() if args.git else 'None'
  checkpoint_path = os.path.join(log_dir, 'model.ckpt')
  input_path = os.path.join(args.base_dir, args.input)
  log('Checkpoint path: %s' % checkpoint_path)
  log('Loading training data from: %s' % input_path)
  log('Using model: %s' % args.model)
  log(hparams_debug_string())

  # Set up DataFeeder:
  coord = tf.train.Coordinator()
  with tf.variable_scope('datafeeder') as scope:
    feeder = DataFeeder(coord, input_path, hparams)

  # Set up model:
  global_step = tf.Variable(0, name='global_step', trainable=False)
  with tf.variable_scope('model') as scope:
    model = create_model(args.model, hparams)
    # print(f"{feeder.inputs}---{feeder.input_lengths}--{feeder.mel_targets},
{feeder.linear_targets}")
    model.initialize(feeder.inputs, feeder.input_lengths, feeder.mel_targets,
feeder.linear_targets)
    model.add_loss()
    model.add_optimizer(global_step)
    stats = add_stats(model)

  # Bookkeeping:
  step = 0
  time_window = ValueWindow(100)
  loss_window = ValueWindow(100)
  saver = tf.train.Saver(max_to_keep=5, keep_checkpoint_every_n_hours=2)

  # Train!
  with tf.Session() as sess:
    try:
      summary_writer = tf.summary.FileWriter(log_dir, sess.graph)
      sess.run(tf.global_variables_initializer())

      if args.restore_step:
        # Restore from a checkpoint if the user requested it.
        restore_path = '%s-%d' % (checkpoint_path, args.restore_step)
        saver.restore(sess, restore_path)
        log('Resuming from checkpoint: %s at commit: %s' % (restore_path,
commit), slack=True)
      else:
        log('Starting new training run at commit: %s' % commit, slack=True)

      feeder.start_in_session(sess)

      while not coord.should_stop():
        start_time = time.time()
        step, loss, opt = sess.run([global_step, model.loss, model.optimize])
        time_window.append(time.time() - start_time)
        loss_window.append(loss)
        message = 'Step %-7d [%.03f sec/step, loss=%.05f, avg_loss=%.05f]' % (
          step, time_window.average, loss, loss_window.average)
        log(message, slack=(step % args.checkpoint_interval == 0))

        if loss > 100 or math.isnan(loss):
          log('Loss exploded to %.05f at step %d!' % (loss, step), slack=True)
          raise Exception('Loss Exploded')

        if step % args.summary_interval == 0:
          log('Writing summary at step: %d' % step)
          summary_writer.add_summary(sess.run(stats), step)

        if step % args.checkpoint_interval == 0:
          log('Saving checkpoint to: %s-%d' % (checkpoint_path, step))
          saver.save(sess, checkpoint_path, global_step=step)
          log('Saving audio and alignment...')
          input_seq, spectrogram, alignment = sess.run([
            model.inputs[0], model.linear_outputs[0], model.alignments[0]])
          waveform = audio.inv_spectrogram(spectrogram.T)
          audio.save_wav(waveform, os.path.join(log_dir, 'step-%d-audio.wav' %
step))
          plot.plot_alignment(alignment, os.path.join(log_dir, 'step-%d-
align.png' % step),
            info='%s, %s, %s, step=%d, loss=%.5f' % (args.model, commit,
time_string(), step, loss))
          log('Input: %s' % sequence_to_text(input_seq))

    except Exception as e:
      log('Exiting due to exception: %s' % e, slack=True)
      traceback.print_exc()
      coord.request_stop(e)


def main():
  parser = argparse.ArgumentParser()
  parser.add_argument('--base_dir',
default=os.path.expanduser('~/PycharmProjects/tacotron'))
  parser.add_argument('--input', default='training/train.txt')
  parser.add_argument('--model', default='tacotron')
  parser.add_argument('--name', help='Name of the run. Used for logging.
Defaults to model name.')
  parser.add_argument('--hparams', default='',
    help='Hyperparameter overrides as a comma-separated list of name=value
pairs')
  parser.add_argument('--restore_step', type=int, help='Global step to restore
from checkpoint.')
  parser.add_argument('--summary_interval', type=int, default=100,
    help='Steps between running summary ops.')
  parser.add_argument('--checkpoint_interval', type=int, default=1000,
    help='Steps between writing checkpoints.')
  parser.add_argument('--slack_url', help='Slack webhook URL to get periodic
reports.')
  parser.add_argument('--tf_log_level', type=int, default=1, help='Tensorflow C+
+ log level.')
  parser.add_argument('--git', action='store_true', help='If set, verify that
the client is clean.')
  args = parser.parse_args()
  os.environ['TF_CPP_MIN_LOG_LEVEL'] = str(args.tf_log_level)
  run_name = args.name or args.model
  log_dir = os.path.join(args.base_dir, 'logs-%s' % run_name)
  os.makedirs(log_dir, exist_ok=True)
  infolog.init(os.path.join(log_dir, 'train.log'), run_name, args.slack_url)
  hparams.parse(args.hparams)
  train(log_dir, args)


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

==
====================

==
====================
📄 File: TRAINING_DATA.md
📁 Path: TRAINING_DATA.md

# Training Data

This repo supports the following speech datasets:
  * [LJ Speech](https://keithito.com/LJ-Speech-Dataset/) (Public Domain)
  * [Blizzard 2012](http://www.cstr.ed.ac.uk/projects/blizzard/2012/phase_one)
(Creative Commons Attribution Share-Alike)

You can use any other dataset if you write a preprocessor for it.

### Writing a Preprocessor

Each training example consists of:
  1. The text that was spoken
  2. A mel-scale spectrogram of the audio
  3. A linear-scale spectrogram of the audio

The preprocessor is responsible for generating these. See [ljspeech.py]
(datasets/ljspeech.py) for a
commented example.

For each training example, a preprocessor should:

  1. Load the audio file:
     ```python
     wav = audio.load_wav(wav_path)
     ```

  2. Compute linear-scale and mel-scale spectrograms (float32 numpy arrays):
     ```python
     ```

```
     spectrogram = audio.spectrogram(wav).astype(np.float32)
     mel_spectrogram = audio.melspectrogram(wav).astype(np.float32)
     ```

  3. Save the spectrograms to disk:
     ```python
     np.save(os.path.join(out_dir, spectrogram_filename), spectrogram.T,
allow_pickle=False)
     np.save(os.path.join(out_dir, mel_spectrogram_filename), mel_spectrogram.T,
allow_pickle=False)
     ```
     Note that the transpose of the matrix returned by `audio.spectrogram` is
saved so that it's
     in time-major format.

  4. Generate a tuple `(spectrogram_filename, mel_spectrogram_filename,
n_frames, text)` to
     write to train.txt. n_frames is just the length of the time axis of the
spectrogram.

After you've written your preprocessor, you can add it to [preprocess.py]
(preprocess.py) by
following the example of the other preprocessors in that file.

### Non-English Data

If your training data is in a language other than English, you will probably
want to change the
text cleaners by setting the `cleaners` hyperparameter.

  * If your text is in a Latin script or can be transliterated to ASCII using
the
    [Unidecode](https://pypi.python.org/pypi/Unidecode) library, you can use the
transliteration
    cleaners by setting the hyperparameter `cleaners=transliteration_cleaners`.

  * If you don't want to transliterate, you can define a custom character set.
    This allows you to train directly on the character set used in your data.

    To do so, edit [symbols.py](text/symbols.py) and change the `_characters`
variable to be a
    string containing the UTF-8 characters in your data. Then set the
hyperparameter `cleaners=basic_cleaners`.

  * If you're not sure which option to use, you can evaluate the transliteration
cleaners like this:

    ```python
    from text import cleaners
    cleaners.transliteration_cleaners('Здравствуйте')   # Replace with the text
you want to try
    ```

==
====================

==
====================
📄 File: config
📂 Path: .git\config


```
```

```
[core]
	repositoryformatversion = 0
	filemode = false
	bare = false
	logallrefupdates = true
	symlinks = false
	ignorecase = true
[remote "origin"]
	url = https://github.com/silencedsre/tacotron.git
	fetch = +refs/heads/*:refs/remotes/origin/*
[branch "master"]
	remote = origin
	merge = refs/heads/master
	vscode-merge-base = origin/master
	vscode-merge-base = origin/master

================================================================================
====================

================================================================================
====================
📄 File: description
📁 Path: .git\description
--------------------------------------------------------------------------------
--------------------
Unnamed repository; edit this file 'description' to name the repository.

================================================================================
====================

================================================================================
====================
📄 File: FETCH_HEAD
📁 Path: .git\FETCH_HEAD
--------------------------------------------------------------------------------
--------------------
e198a486a0c61187a630994b53947a43f70449a3		branch 'master' of
https://github.com/silencedsre/tacotron
421770db81dd3b3ce28d537760e6520a5ed188d1	not-for-merge	branch 'multispeaker'
of https://github.com/silencedsre/tacotron
065982647cad2b19b642bb2bc9ef431ba95a6c4a	not-for-merge	branch 'tacotron2-
work-in-progress' of https://github.com/silencedsre/tacotron

================================================================================
====================

================================================================================
====================
📄 File: HEAD
📁 Path: .git\HEAD
--------------------------------------------------------------------------------
--------------------
ref: refs/heads/master

================================================================================
====================

[Skipping binary or unreadable file: .git\index]

================================================================================
====================
📄 File: packed-refs
📁 Path: .git\packed-refs
--------------------------------------------------------------------------------
```

```
--------------------
# pack-refs with: peeled fully-peeled sorted
e198a486a0c61187a630994b53947a43f70449a3 refs/remotes/origin/master
421770db81dd3b3ce28d537760e6520a5ed188d1 refs/remotes/origin/multispeaker
065982647cad2b19b642bb2bc9ef431ba95a6c4a refs/remotes/origin/tacotron2-work-in-
progress
d715298713b9bfc818f2118e2ef323fe4cc2d1c3 refs/tags/pretrained-model-1
6a793ba8c42f01f034293c4bd2b95623ad0da34d refs/tags/v0.1.0
522826dd771671f3c91374c22f94a7af6c237e15 refs/tags/v0.2.0

================================================================================
====================

================================================================================
====================
📄 File: applypatch-msg.sample
📂 Path: .git\hooks\applypatch-msg.sample
--------------------------------------------------------------------------------
--------------------
#!/bin/sh
#
# An example hook script to check the commit log message taken by
# applypatch from an e-mail message.
#
# The hook should exit with non-zero status after issuing an
# appropriate message if it wants to stop the commit.  The hook is
# allowed to edit the commit message file.
#
# To enable this hook, rename this file to "applypatch-msg".

. git-sh-setup
commitmsg="$(git rev-parse --git-path hooks/commit-msg)"
test -x "$commitmsg" && exec "$commitmsg" ${1+"$@"}
:

================================================================================
====================

================================================================================
====================
📄 File: commit-msg.sample
📂 Path: .git\hooks\commit-msg.sample
--------------------------------------------------------------------------------
--------------------
#!/bin/sh
#
# An example hook script to check the commit log message.
# Called by "git commit" with one argument, the name of the file
# that has the commit message.  The hook should exit with non-zero
# status after issuing an appropriate message if it wants to stop the
# commit.  The hook is allowed to edit the commit message file.
#
# To enable this hook, rename this file to "commit-msg".

# Uncomment the below to add a Signed-off-by line to the message.
# Doing this in a hook is a bad idea in general, but the prepare-commit-msg
# hook is more suited to it.
#
# SOB=$(git var GIT_AUTHOR_IDENT | sed -n 's/^\(.*>\).*$/Signed-off-by: \1/p')
# grep -qs "^$SOB" "$1" || echo "$SOB" >> "$1"

# This example catches duplicate Signed-off-by lines.

test "" = "$(grep '^Signed-off-by: ' "$1" |
	 sort | uniq -c | sed -e '/^[	 ]*1[  ]/d')" || {
	echo >&2 Duplicate Signed-off-by lines.
	exit 1
}
```

==
====================

==
====================
📄 File: fsmonitor-watchman.sample
📁 Path: .git\hooks\fsmonitor-watchman.sample


```
#!/usr/bin/perl

use strict;
use warnings;
use IPC::Open2;

# An example hook script to integrate Watchman
# (https://facebook.github.io/watchman/) with git to speed up detecting
# new and modified files.
#
# The hook is passed a version (currently 2) and last update token
# formatted as a string and outputs to stdout a new update token and
# all files that have been modified since the update token. Paths must
# be relative to the root of the working tree and separated by a single NUL.
#
# To enable this hook, rename this file to "query-watchman" and set
# 'git config core.fsmonitor .git/hooks/query-watchman'
#
my ($version, $last_update_token) = @ARGV;

# Uncomment for debugging
# print STDERR "$0 $version $last_update_token\n";

# Check the hook interface version
if ($version ne 2) {
	die "Unsupported query-fsmonitor hook version '$version'.\n" .
	    "Falling back to scanning...\n";
}

my $git_work_tree = get_working_dir();

my $retry = 1;

my $json_pkg;
eval {
	require JSON::XS;
	$json_pkg = "JSON::XS";
	1;
} or do {
	require JSON::PP;
	$json_pkg = "JSON::PP";
};

launch_watchman();

sub launch_watchman {
	my $o = watchman_query();
	if (is_work_tree_watched($o)) {
		output_result($o->{clock}, @{$o->{files}});
	}
```

```
}

sub output_result {
      my ($clockid, @files) = @_;

      # Uncomment for debugging watchman output
      # open (my $fh, ">", ".git/watchman-output.out");
      # binmode $fh, ":utf8";
      # print $fh "$clockid\n@files\n";
      # close $fh;

      binmode STDOUT, ":utf8";
      print $clockid;
      print "\0";
      local $, = "\0";
      print @files;
}

sub watchman_clock {
      my $response = qx/watchman clock "$git_work_tree"/;
      die "Failed to get clock id on '$git_work_tree'.\n" .
            "Falling back to scanning...\n" if $? != 0;

      return $json_pkg->new->utf8->decode($response);
}

sub watchman_query {
      my $pid = open2(\*CHLD_OUT, \*CHLD_IN, 'watchman -j --no-pretty')
      or die "open2() failed: $!\n" .
      "Falling back to scanning...\n";

      # In the query expression below we're asking for names of files that
      # changed since $last_update_token but not from the .git folder.
      #
      # To accomplish this, we're using the "since" generator to use the
      # recency index to select candidate nodes and "fields" to limit the
      # output to file names only. Then we're using the "expression" term to
      # further constrain the results.
      my $last_update_line = "";
      if (substr($last_update_token, 0, 1) eq "c") {
            $last_update_token = "\"$last_update_token\"";
            $last_update_line = qq[\n"since": $last_update_token,];
      }
      my $query = <<"   END";
            ["query", "$git_work_tree", {$last_update_line
                  "fields": ["name"],
                  "expression": ["not", ["dirname", ".git"]]
            }]
      END

      # Uncomment for debugging the watchman query
      # open (my $fh, ">", ".git/watchman-query.json");
      # print $fh $query;
      # close $fh;

      print CHLD_IN $query;
      close CHLD_IN;
      my $response = do {local $/; <CHLD_OUT>};

      # Uncomment for debugging the watch response
      # open ($fh, ">", ".git/watchman-response.json");
      # print $fh $response;
      # close $fh;
      die "Watchman: command returned no output.\n" .
      "Falling back to scanning...\n" if $response eq "";
      die "Watchman: command returned invalid output: $response\n" .
      "Falling back to scanning...\n" unless $response =~ /^\{/;

      return $json_pkg->new->utf8->decode($response);
}

sub is_work_tree_watched {
      my ($output) = @_;
      my $error = $output->{error};
      if ($retry > 0 and $error and $error =~ m/unable to resolve root .*
directory (.*) is not watched/) {
            $retry--;
            my $response = qx/watchman watch "$git_work_tree"/;
            die "Failed to make watchman watch '$git_work_tree'.\n" .
                "Falling back to scanning...\n" if $? != 0;
            $output = $json_pkg->new->utf8->decode($response);
            $error = $output->{error};
            die "Watchman: $error.\n" .
            "Falling back to scanning...\n" if $error;

            # Uncomment for debugging watchman output
            # open (my $fh, ">", ".git/watchman-output.out");
            # close $fh;

            # Watchman will always return all files on the first query so
            # return the fast "everything is dirty" flag to git and do the
            # Watchman query just to get it over with now so we won't pay
            # the cost in git to look up each individual file.
            my $o = watchman_clock();
            $error = $output->{error};

            die "Watchman: $error.\n" .
            "Falling back to scanning...\n" if $error;

            output_result($o->{clock}, ("/"));
            $last_update_token = $o->{clock};

            eval { launch_watchman() };
            return 0;
      }

      die "Watchman: $error.\n" .
      "Falling back to scanning...\n" if $error;

      return 1;
}

sub get_working_dir {
      my $working_dir;
      if ($^O =~ 'msys' || $^O =~ 'cygwin') {
            $working_dir = Win32::GetCwd();
            $working_dir =~ tr/\\/\//;
      } else {
            require Cwd;
            $working_dir = Cwd::cwd();
      }

      return $working_dir;
}

================================================================================
====================
================================================================================
====================
📄 File: post-update.sample
📂 Path: .git\hooks\post-update.sample
--------------------------------------------------------------------------------
--------------------
#!/bin/sh
#
# An example hook script to prepare a packed repository for use over
# dumb transports.
#
# To enable this hook, rename this file to "post-update".

exec git update-server-info

================================================================================
====================

================================================================================
====================
📄 File: pre-applypatch.sample
📂 Path: .git\hooks\pre-applypatch.sample
--------------------------------------------------------------------------------
--------------------
#!/bin/sh
#
# An example hook script to verify what is about to be committed
# by applypatch from an e-mail message.
#
# The hook should exit with non-zero status after issuing an
# appropriate message if it wants to stop the commit.
#
# To enable this hook, rename this file to "pre-applypatch".

. git-sh-setup
precommit="$(git rev-parse --git-path hooks/pre-commit)"
test -x "$precommit" && exec "$precommit" ${1+"$@"}
:

================================================================================
====================

================================================================================
====================
📄 File: pre-commit.sample
📂 Path: .git\hooks\pre-commit.sample
--------------------------------------------------------------------------------
--------------------
#!/bin/sh
#
# An example hook script to verify what is about to be committed.
# Called by "git commit" with no arguments.  The hook should
# exit with non-zero status after issuing an appropriate message if
# it wants to stop the commit.
#
# To enable this hook, rename this file to "pre-commit".

if git rev-parse --verify HEAD >/dev/null 2>&1
then
	against=HEAD
else
	# Initial commit: diff against an empty tree object
	against=$(git hash-object -t tree /dev/null)
```

```
fi

# If you want to allow non-ASCII filenames set this variable to true.
allownonascii=$(git config --type=bool hooks.allownonascii)

# Redirect output to stderr.
exec 1>&2

# Cross platform projects tend to avoid non-ASCII filenames; prevent
# them from being added to the repository. We exploit the fact that the
# printable range starts at the space character and ends with tilde.
if [ "$allownonascii" != "true" ] &&
	# Note that the use of brackets around a tr range is ok here, (it's
	# even required, for portability to Solaris 10's /usr/bin/tr), since
	# the square bracket bytes happen to fall in the designated range.
	test $(git diff-index --cached --name-only --diff-filter=A -z $against |
	  LC_ALL=C tr -d '[ -~]\0' | wc -c) != 0
then
	cat <<\EOF
Error: Attempt to add a non-ASCII file name.

This can cause problems if you want to work with people on other platforms.

To be portable it is advisable to rename the file.

If you know what you are doing you can disable this check using:

  git config hooks.allownonascii true
EOF
	exit 1
fi

# If there are whitespace errors, print the offending file names and fail.
exec git diff-index --check --cached $against --

================================================================================
====================

================================================================================
====================
📄 File: pre-merge-commit.sample
📂 Path: .git\hooks\pre-merge-commit.sample
--------------------------------------------------------------------------------
--------------------
#!/bin/sh
#
# An example hook script to verify what is about to be committed.
# Called by "git merge" with no arguments.  The hook should
# exit with non-zero status after issuing an appropriate message to
# stderr if it wants to stop the merge commit.
#
# To enable this hook, rename this file to "pre-merge-commit".

. git-sh-setup
test -x "$GIT_DIR/hooks/pre-commit" &&
        exec "$GIT_DIR/hooks/pre-commit"
:

================================================================================
====================

================================================================================
====================
📄 File: pre-push.sample
```

📁 Path: .git\hooks\pre-push.sample

```
#!/bin/sh

# An example hook script to verify what is about to be pushed.  Called by "git
# push" after it has checked the remote status, but before anything has been
# pushed.  If this script exits with a non-zero status nothing will be pushed.
#
# This hook is called with the following parameters:
#
# $1 -- Name of the remote to which the push is being done
# $2 -- URL to which the push is being done
#
# If pushing without using a named remote those arguments will be equal.
#
# Information about the commits which are being pushed is supplied as lines to
# the standard input in the form:
#
#   <local ref> <local oid> <remote ref> <remote oid>
#
# This sample shows how to prevent push of commits where the log message starts
# with "WIP" (work in progress).

remote="$1"
url="$2"

zero=$(git hash-object --stdin </dev/null | tr '[0-9a-f]' '0')

while read local_ref local_oid remote_ref remote_oid
do
	if test "$local_oid" = "$zero"
	then
		# Handle delete
		:
	else
		if test "$remote_oid" = "$zero"
		then
			# New branch, examine all commits
			range="$local_oid"
		else
			# Update to existing branch, examine new commits
			range="$remote_oid..$local_oid"
		fi

		# Check for WIP commit
		commit=$(git rev-list -n 1 --grep '^WIP' "$range")
		if test -n "$commit"
		then
			echo >&2 "Found WIP commit in $local_ref, not pushing"
			exit 1
		fi
	fi
done

exit 0
```

==

==
📄 File: pre-rebase.sample
📁 Path: .git\hooks\pre-rebase.sample

```
--------------------------------------------------------------------------------
--------------------
#!/bin/sh
#
# Copyright (c) 2006, 2008 Junio C Hamano
#
# The "pre-rebase" hook is run just before "git rebase" starts doing
# its job, and can prevent the command from running by exiting with
# non-zero status.
#
# The hook is called with the following parameters:
#
# $1 -- the upstream the series was forked from.
# $2 -- the branch being rebased (or empty when rebasing the current branch).
#
# This sample shows how to prevent topic branches that are already
# merged to 'next' branch from getting rebased, because allowing it
# would result in rebasing already published history.

publish=next
basebranch="$1"
if test "$#" = 2
then
      topic="refs/heads/$2"
else
      topic=`git symbolic-ref HEAD` ||
      exit 0 ;# we do not interrupt rebasing detached HEAD
fi

case "$topic" in
refs/heads/??/*)
      ;;
*)
      exit 0 ;# we do not interrupt others.
      ;;
esac

# Now we are dealing with a topic branch being rebased
# on top of master.  Is it OK to rebase it?

# Does the topic really exist?
git show-ref -q "$topic" || {
      echo >&2 "No such branch $topic"
      exit 1
}

# Is topic fully merged to master?
not_in_master=`git rev-list --pretty=oneline ^master "$topic"`
if test -z "$not_in_master"
then
      echo >&2 "$topic is fully merged to master; better remove it."
      exit 1 ;# we could allow it, but there is no point.
fi

# Is topic ever merged to next?  If so you should not be rebasing it.
only_next_1=`git rev-list ^master "^$topic" ${publish} | sort`
only_next_2=`git rev-list ^master           ${publish} | sort`
if test "$only_next_1" = "$only_next_2"
then
      not_in_topic=`git rev-list "^$topic" master`
      if test -z "$not_in_topic"
      then
            echo >&2 "$topic is already up to date with master"
            exit 1 ;# we could allow it, but there is no point.
      else
            exit 0
      fi
else
      not_in_next=`git rev-list --pretty=oneline ^${publish} "$topic"`
      /usr/bin/perl -e '
            my $topic = $ARGV[0];
            my $msg = "* $topic has commits already merged to public branch:\n";
            my (%not_in_next) = map {
                  /^([0-9a-f]+) /;
                  ($1 => 1);
            } split(/\n/, $ARGV[1]);
            for my $elem (map {
                        /^([0-9a-f]+) (.*)$/;
                        [$1 => $2];
                  } split(/\n/, $ARGV[2])) {
                  if (!exists $not_in_next{$elem->[0]}) {
                        if ($msg) {
                              print STDERR $msg;
                              undef $msg;
                        }
                        print STDERR " $elem->[1]\n";
                  }
            }
      ' "$topic" "$not_in_next" "$not_in_master"
      exit 1
fi

<<\DOC_END

This sample hook safeguards topic branches that have been
published from being rewound.

The workflow assumed here is:

 * Once a topic branch forks from "master", "master" is never
   merged into it again (either directly or indirectly).

 * Once a topic branch is fully cooked and merged into "master",
   it is deleted.  If you need to build on top of it to correct
   earlier mistakes, a new topic branch is created by forking at
   the tip of the "master".  This is not strictly necessary, but
   it makes it easier to keep your history simple.

 * Whenever you need to test or publish your changes to topic
   branches, merge them into "next" branch.

The script, being an example, hardcodes the publish branch name
to be "next", but it is trivial to make it configurable via
$GIT_DIR/config mechanism.

With this workflow, you would want to know:

(1) ... if a topic branch has ever been merged to "next".  Young
    topic branches can have stupid mistakes you would rather
    clean up before publishing, and things that have not been
    merged into other branches can be easily rebased without
    affecting other people.  But once it is published, you would
    not want to rewind it.

(2) ... if a topic branch has been fully merged to "master".
    Then you can delete it.  More importantly, you should not
    build on top of it -- other people may already want to
    change things related to the topic as patches against your
    "master", so if you need further changes, it is better to
    fork the topic (perhaps with the same name) afresh from the
    tip of "master".

Let's look at this example:

               o---o---o---o---o---o---o---o---o---o "next"
              /       /           /           /
             /   a---a---b A     /           /
            /   /               /           /
             /   /   c---c---c---c B         /
            /   /   /             \         /
           /   /   /   b---b C     \       /
          /   /   /   /             \     /
    ---o---o---o---o---o---o---o---o---o---o---o "master"


A, B and C are topic branches.

 * A has one fix since it was merged up to "next".

 * B has finished.  It has been fully merged up to "master" and "next",
   and is ready to be deleted.

 * C has not merged to "next" at all.

We would want to allow C to be rebased, refuse A, and encourage
B to be deleted.

To compute (1):

      git rev-list ^master ^topic next
      git rev-list ^master        next

      if these match, topic has not merged in next at all.

To compute (2):

      git rev-list master..topic

      if this is empty, it is fully merged to "master".

DOC_END

================================================================================
====================

================================================================================
====================
📄 File: pre-receive.sample
📁 Path: .git\hooks\pre-receive.sample
--------------------------------------------------------------------------------
--------------------
#!/bin/sh
#
# An example hook script to make use of push options.
# The example simply echoes all push options that start with 'echoback='
# and rejects all pushes when the "reject" push option is used.
#
# To enable this hook, rename this file to "pre-receive".

if test -n "$GIT_PUSH_OPTION_COUNT"
then
      i=0
	while test "$i" -lt "$GIT_PUSH_OPTION_COUNT"
	do
		eval "value=\$GIT_PUSH_OPTION_$i"
		case "$value" in
		echoback=*)
			echo "echo from the pre-receive-hook: ${value#*=}" >&2
			;;
		reject)
			exit 1
		esac
		i=$((i + 1))
	done
fi
```

==
====================

==
====================
📄 File: prepare-commit-msg.sample
📁 Path: .git\hooks\prepare-commit-msg.sample


```
#!/bin/sh
#
# An example hook script to prepare the commit log message.
# Called by "git commit" with the name of the file that has the
# commit message, followed by the description of the commit
# message's source.  The hook's purpose is to edit the commit
# message file.  If the hook fails with a non-zero status,
# the commit is aborted.
#
# To enable this hook, rename this file to "prepare-commit-msg".

# This hook includes three examples. The first one removes the
# "# Please enter the commit message..." help message.
#
# The second includes the output of "git diff --name-status -r"
# into the message, just before the "git status" output.  It is
# commented because it doesn't cope with --amend or with squashed
# commits.
#
# The third example adds a Signed-off-by line to the message, that can
# still be edited.  This is rarely a good idea.

COMMIT_MSG_FILE=$1
COMMIT_SOURCE=$2
SHA1=$3

/usr/bin/perl -i.bak -ne 'print unless(m/^. Please enter the commit
message/..m/^#$/)' "$COMMIT_MSG_FILE"

# case "$COMMIT_SOURCE,$SHA1" in
#  ,|template,)
#    /usr/bin/perl -i.bak -pe '
#       print "\n" . `git diff --cached --name-status -r`
#      if /^#/ && $first++ == 0' "$COMMIT_MSG_FILE" ;;
#  *) ;;
# esac

# SOB=$(git var GIT_COMMITTER_IDENT | sed -n 's/^\(.*>\).*$/Signed-off-by:
\1/p')
# git interpret-trailers --in-place --trailer "$SOB" "$COMMIT_MSG_FILE"
# if test -z "$COMMIT_SOURCE"
# then
#   /usr/bin/perl -i.bak -pe 'print "\n" if !$first_line++' "$COMMIT_MSG_FILE"
# fi
```

==
====================

==
====================
📄 File: push-to-checkout.sample
📁 Path: .git\hooks\push-to-checkout.sample


```
#!/bin/sh

# An example hook script to update a checked-out tree on a git push.
#
# This hook is invoked by git-receive-pack(1) when it reacts to git
# push and updates reference(s) in its repository, and when the push
# tries to update the branch that is currently checked out and the
# receive.denyCurrentBranch configuration variable is set to
# updateInstead.
#
# By default, such a push is refused if the working tree and the index
# of the remote repository has any difference from the currently
# checked out commit; when both the working tree and the index match
# the current commit, they are updated to match the newly pushed tip
# of the branch. This hook is to be used to override the default
# behaviour; however the code below reimplements the default behaviour
# as a starting point for convenient modification.
#
# The hook receives the commit with which the tip of the current
# branch is going to be updated:
commit=$1

# It can exit with a non-zero status to refuse the push (when it does
# so, it must not modify the index or the working tree).
die () {
      echo >&2 "$*"
      exit 1
}

# Or it can make any necessary changes to the working tree and to the
# index to bring them to the desired state when the tip of the current
# branch is updated to the new commit, and exit with a zero status.
#
# For example, the hook can simply run git read-tree -u -m HEAD "$1"
# in order to emulate git fetch that is run in the reverse direction
# with git push, as the two-tree form of git read-tree -u -m is
# essentially the same as git switch or git checkout that switches
# branches while keeping the local changes in the working tree that do
# not interfere with the difference between the branches.

# The below is a more-or-less exact translation to shell of the C code
# for the default behaviour for git's push-to-checkout hook defined in
# the push_to_deploy() function in builtin/receive-pack.c.
#
# Note that the hook will be executed from the repository directory,
# not from the working tree, so if you want to perform operations on
# the working tree, you will have to adapt your code accordingly, e.g.
# by adding "cd .." or using relative paths.

if ! git update-index -q --ignore-submodules --refresh
then
```

```
      die "Up-to-date check failed"
fi

if ! git diff-files --quiet --ignore-submodules --
then
      die "Working directory has unstaged changes"
fi

# This is a rough translation of:
#
#   head_has_history() ? "HEAD" : EMPTY_TREE_SHA1_HEX
if git cat-file -e HEAD 2>/dev/null
then
      head=HEAD
else
      head=$(git hash-object -t tree --stdin </dev/null)
fi

if ! git diff-index --quiet --cached --ignore-submodules $head --
then
      die "Working directory has staged changes"
fi

if ! git read-tree -u -m "$commit"
then
      die "Could not update working tree to new HEAD"
fi
```

==
====================

==
====================
📄 File: sendemail-validate.sample
📁 Path: .git\hooks\sendemail-validate.sample


```
#!/bin/sh

# An example hook script to validate a patch (and/or patch series) before
# sending it via email.
#
# The hook should exit with non-zero status after issuing an appropriate
# message if it wants to prevent the email(s) from being sent.
#
# To enable this hook, rename this file to "sendemail-validate".
#
# By default, it will only check that the patch(es) can be applied on top of
# the default upstream branch without conflicts in a secondary worktree. After
# validation (successful or not) of the last patch of a series, the worktree
# will be deleted.
#
# The following config variables can be set to change the default remote and
# remote ref that are used to apply the patches against:
#
#   sendemail.validateRemote (default: origin)
#   sendemail.validateRemoteRef (default: HEAD)
#
# Replace the TODO placeholders with appropriate checks according to your
# needs.

validate_cover_letter () {
      file="$1"
      # TODO: Replace with appropriate checks (e.g. spell checking).
```

```
	true
}

validate_patch () {
	file="$1"
	# Ensure that the patch applies without conflicts.
	git am -3 "$file" || return
	# TODO: Replace with appropriate checks for this patch
	# (e.g. checkpatch.pl).
	true
}

validate_series () {
	# TODO: Replace with appropriate checks for the whole series
	# (e.g. quick build, coding style checks, etc.).
	true
}

# main -------------------------------------------------------------------------

if test "$GIT_SENDEMAIL_FILE_COUNTER" = 1
then
	remote=$(git config --default origin --get sendemail.validateRemote) &&
	ref=$(git config --default HEAD --get sendemail.validateRemoteRef) &&
	worktree=$(mktemp --tmpdir -d sendemail-validate.XXXXXXX) &&
	git worktree add -fd --checkout "$worktree" "refs/remotes/$remote/$ref" &&
	git config --replace-all sendemail.validateWorktree "$worktree"
else
	worktree=$(git config --get sendemail.validateWorktree)
fi || {
	echo "sendemail-validate: error: failed to prepare worktree" >&2
	exit 1
}

unset GIT_DIR GIT_WORK_TREE
cd "$worktree" &&

if grep -q "^diff --git " "$1"
then
	validate_patch "$1"
else
	validate_cover_letter "$1"
fi &&

if test "$GIT_SENDEMAIL_FILE_COUNTER" = "$GIT_SENDEMAIL_FILE_TOTAL"
then
	git config --unset-all sendemail.validateWorktree &&
	trap 'git worktree remove -ff "$worktree"' EXIT &&
	validate_series
fi
```

==
====================

==
====================
📄 File: update.sample
📂 Path: .git\hooks\update.sample


```
#!/bin/sh
#
# An example hook script to block unannotated tags from entering.
# Called by "git receive-pack" with arguments: refname sha1-old sha1-new
```

```
#
# To enable this hook, rename this file to "update".
#
# Config
# ------
# hooks.allowunannotated
#   This boolean sets whether unannotated tags will be allowed into the
#   repository.  By default they won't be.
# hooks.allowdeletetag
#   This boolean sets whether deleting tags will be allowed in the
#   repository.  By default they won't be.
# hooks.allowmodifytag
#   This boolean sets whether a tag may be modified after creation. By default
#   it won't be.
# hooks.allowdeletebranch
#   This boolean sets whether deleting branches will be allowed in the
#   repository.  By default they won't be.
# hooks.denycreatebranch
#   This boolean sets whether remotely creating branches will be denied
#   in the repository.  By default this is allowed.
#

# --- Command line
refname="$1"
oldrev="$2"
newrev="$3"

# --- Safety check
if [ -z "$GIT_DIR" ]; then
      echo "Don't run this script from the command line." >&2
      echo " (if you want, you could supply GIT_DIR then run" >&2
      echo "  $0 <ref> <oldrev> <newrev>)" >&2
      exit 1
fi

if [ -z "$refname" -o -z "$oldrev" -o -z "$newrev" ]; then
      echo "usage: $0 <ref> <oldrev> <newrev>" >&2
      exit 1
fi

# --- Config
allowunannotated=$(git config --type=bool hooks.allowunannotated)
allowdeletebranch=$(git config --type=bool hooks.allowdeletebranch)
denycreatebranch=$(git config --type=bool hooks.denycreatebranch)
allowdeletetag=$(git config --type=bool hooks.allowdeletetag)
allowmodifytag=$(git config --type=bool hooks.allowmodifytag)

# check for no description
projectdesc=$(sed -e '1q' "$GIT_DIR/description")
case "$projectdesc" in
"Unnamed repository"* | "")
      echo "*** Project description file hasn't been set" >&2
      exit 1
      ;;
esac

# --- Check types
# if $newrev is 0000...0000, it's a commit to delete a ref.
zero=$(git hash-object --stdin </dev/null | tr '[0-9a-f]' '0')
if [ "$newrev" = "$zero" ]; then
      newrev_type=delete
else
      newrev_type=$(git cat-file -t $newrev)
fi
```

```
case "$refname","$newrev_type" in
      refs/tags/*,commit)
            # un-annotated tag
            short_refname=${refname##refs/tags/}
            if [ "$allowunannotated" != "true" ]; then
                  echo "*** The un-annotated tag, $short_refname, is not allowed
in this repository" >&2
                  echo "*** Use 'git tag [ -a | -s ]' for tags you want to
propagate." >&2
                  exit 1
            fi
            ;;
      refs/tags/*,delete)
            # delete tag
            if [ "$allowdeletetag" != "true" ]; then
                  echo "*** Deleting a tag is not allowed in this repository"
>&2
                  exit 1
            fi
            ;;
      refs/tags/*,tag)
            # annotated tag
            if [ "$allowmodifytag" != "true" ] && git rev-parse $refname >
/dev/null 2>&1
            then
                  echo "*** Tag '$refname' already exists." >&2
                  echo "*** Modifying a tag is not allowed in this repository."
>&2
                  exit 1
            fi
            ;;
      refs/heads/*,commit)
            # branch
            if [ "$oldrev" = "$zero" -a "$denycreatebranch" = "true" ]; then
                  echo "*** Creating a branch is not allowed in this repository"
>&2
                  exit 1
            fi
            ;;
      refs/heads/*,delete)
            # delete branch
            if [ "$allowdeletebranch" != "true" ]; then
                  echo "*** Deleting a branch is not allowed in this repository"
>&2
                  exit 1
            fi
            ;;
      refs/remotes/*,commit)
            # tracking branch
            ;;
      refs/remotes/*,delete)
            # delete tracking branch
            if [ "$allowdeletebranch" != "true" ]; then
                  echo "*** Deleting a tracking branch is not allowed in this
repository" >&2
                  exit 1
            fi
            ;;
      *)
            # Anything else (is there anything else?)
            echo "*** Update hook: unknown type of update to ref $refname of
type $newrev_type" >&2
            exit 1
            ;;
esac

# --- Finished
exit 0

================================================================================
====================

================================================================================
====================
📄 File: exclude
📂 Path: .git\info\exclude
--------------------------------------------------------------------------------
--------------------
# git ls-files --others --exclude-from=.git/info/exclude
# Lines that start with '#' are comments.
# For a project mostly in C, the following would be a good set of
# exclude patterns (uncomment them if you want to use them):
# *.[oa]
# *~

================================================================================
====================

================================================================================
====================
📄 File: HEAD
📂 Path: .git\logs\HEAD
--------------------------------------------------------------------------------
--------------------
0000000000000000000000000000000000000000
e198a486a0c61187a630994b53947a43f70449a3 Dipesh <deepeshpixar123@gmail.com>
1749089531 +0545  clone: from https://github.com/silencedsre/tacotron.git

================================================================================
====================

================================================================================
====================
📄 File: master
📂 Path: .git\logs\refs\heads\master
--------------------------------------------------------------------------------
--------------------
0000000000000000000000000000000000000000
e198a486a0c61187a630994b53947a43f70449a3 Dipesh <deepeshpixar123@gmail.com>
1749089531 +0545  clone: from https://github.com/silencedsre/tacotron.git

================================================================================
====================

================================================================================
====================
📄 File: HEAD
📂 Path: .git\logs\refs\remotes\origin\HEAD
--------------------------------------------------------------------------------
--------------------
0000000000000000000000000000000000000000
e198a486a0c61187a630994b53947a43f70449a3 Dipesh <deepeshpixar123@gmail.com>
1749089531 +0545  clone: from https://github.com/silencedsre/tacotron.git

================================================================================
====================
[Skipping binary or unreadable file: .git\objects\pack\pack-
c1832090f19484dc3898461279601b4a1d95e2be.idx]

[Skipping binary or unreadable file: .git\objects\pack\pack-
c1832090f19484dc3898461279601b4a1d95e2be.pack]

[Skipping binary or unreadable file: .git\objects\pack\pack-
c1832090f19484dc3898461279601b4a1d95e2be.rev]

================================================================================
====================
📄 File: master
📁 Path: .git\refs\heads\master
--------------------------------------------------------------------------------
--------------------
e198a486a0c61187a630994b53947a43f70449a3

================================================================================
====================

================================================================================
====================
📄 File: HEAD
📁 Path: .git\refs\remotes\origin\HEAD
--------------------------------------------------------------------------------
--------------------
ref: refs/remotes/origin/master

================================================================================
====================

================================================================================
====================
📄 File: blizzard.py
📁 Path: datasets\blizzard.py
--------------------------------------------------------------------------------
--------------------
from concurrent.futures import ProcessPoolExecutor
from functools import partial
import numpy as np
import os
from hparams import hparams
from util import audio


_max_out_length = 700
_end_buffer = 0.05
_min_confidence = 90

# Note: "A Tramp Abroad" & "The Man That Corrupted Hadleyburg" are higher
quality than the others.
books = [
  'ATrampAbroad',
  'TheManThatCorruptedHadleyburg',
  # 'LifeOnTheMississippi',
  # 'TheAdventuresOfTomSawyer',
]

def build_from_path(in_dir, out_dir, num_workers=1, tqdm=lambda x: x):
  executor = ProcessPoolExecutor(max_workers=num_workers)
  futures = []
  index = 1
  for book in books:
    with open(os.path.join(in_dir, book, 'sentence_index.txt')) as f:
      for line in f:
        parts = line.strip().split('\t')
        if line[0] is not '#' and len(parts) == 8 and float(parts[3]) >
_min_confidence:
          wav_path = os.path.join(in_dir, book, 'wav', '%s.wav' % parts[0])
          labels_path = os.path.join(in_dir, book, 'lab', '%s.lab' % parts[0])
          text = parts[5]
          task = partial(_process_utterance, out_dir, index, wav_path,
labels_path, text)
          futures.append(executor.submit(task))
          index += 1
  results = [future.result() for future in tqdm(futures)]
  return [r for r in results if r is not None]


def _process_utterance(out_dir, index, wav_path, labels_path, text):
  # Load the wav file and trim silence from the ends:
  wav = audio.load_wav(wav_path)
  start_offset, end_offset = _parse_labels(labels_path)
  start = int(start_offset * hparams.sample_rate)
  end = int(end_offset * hparams.sample_rate) if end_offset is not None else -1
  wav = wav[start:end]
  max_samples = _max_out_length * hparams.frame_shift_ms / 1000 *
hparams.sample_rate
  if len(wav) > max_samples:
    return None
  spectrogram = audio.spectrogram(wav).astype(np.float32)
  n_frames = spectrogram.shape[1]
  mel_spectrogram = audio.melspectrogram(wav).astype(np.float32)
  spectrogram_filename = 'blizzard-spec-%05d.npy' % index
  mel_filename = 'blizzard-mel-%05d.npy' % index
  np.save(os.path.join(out_dir, spectrogram_filename), spectrogram.T,
allow_pickle=False)
  np.save(os.path.join(out_dir, mel_filename), mel_spectrogram.T,
allow_pickle=False)
  return (spectrogram_filename, mel_filename, n_frames, text)


def _parse_labels(path):
  labels = []
  with open(os.path.join(path)) as f:
    for line in f:
      parts = line.strip().split(' ')
      if len(parts) >= 3:
        labels.append((float(parts[0]), ' '.join(parts[2:])))
  start = 0
  end = None
  if labels[0][1] == 'sil':
    start = labels[0][0]
  if labels[-1][1] == 'sil':
    end = labels[-2][0] + _end_buffer
  return (start, end)

================================================================================
====================

================================================================================
====================
📄 File: datafeeder.py
📁 Path: datasets\datafeeder.py
--------------------------------------------------------------------------------
--------------------
import numpy as np
import os
import random
import tensorflow as tf
import threading
import time
import traceback
from text import cmudict, text_to_sequence
from util.infolog import log


_batches_per_group = 32
_p_cmudict = 0.5
_pad = 0


class DataFeeder(threading.Thread):
  '''Feeds batches of data into a queue on a background thread.'''

  def __init__(self, coordinator, metadata_filename, hparams):
    super(DataFeeder, self).__init__()
    self._coord = coordinator
    self._hparams = hparams
    self._cleaner_names = [x.strip() for x in hparams.cleaners.split(',')]
    self._offset = 0

    # Load metadata:
    self._datadir = os.path.dirname(metadata_filename)
    with open(metadata_filename, encoding='utf-8') as f:
      self._metadata = [line.strip().split('|') for line in f]
      hours = sum((int(x[2]) for x in self._metadata)) *
hparams.frame_shift_ms / (3600 * 1000)
      log('Loaded metadata for %d examples (%.2f hours)' % (len(self._metadata),
hours))

    # Create placeholders for inputs and targets. Don't specify batch size
because we want to
    # be able to feed different sized batches at eval time.
    self._placeholders = [
      tf.placeholder(tf.int32, [None, None], 'inputs'),
      tf.placeholder(tf.int32, [None], 'input_lengths'),
      tf.placeholder(tf.float32, [None, None, hparams.num_mels], 'mel_targets'),
      tf.placeholder(tf.float32, [None, None, hparams.num_freq],
'linear_targets')
    ]

    # Create queue for buffering data:
    queue = tf.FIFOQueue(8, [tf.int32, tf.int32, tf.float32, tf.float32],
name='input_queue')
    self._enqueue_op = queue.enqueue(self._placeholders)
    self.inputs, self.input_lengths, self.mel_targets, self.linear_targets =
queue.dequeue()
    self.inputs.set_shape(self._placeholders[0].shape)
    self.input_lengths.set_shape(self._placeholders[1].shape)
    self.mel_targets.set_shape(self._placeholders[2].shape)
    self.linear_targets.set_shape(self._placeholders[3].shape)

    # Load CMUDict: If enabled, this will randomly substitute some words in the
training data with
    # their ARPABet equivalents, which will allow you to also pass ARPABet to
the model for
    # synthesis (useful for proper nouns, etc.)
    if hparams.use_cmudict:
      cmudict_path = os.path.join(self._datadir, 'cmudict-0.7b')
      if not os.path.isfile(cmudict_path):
        raise Exception('If use_cmudict=True, you must download ' +
          'http://svn.code.sf.net/p/cmusphinx/code/trunk/cmudict/cmudict-0.7b to
%s'  % cmudict_path)
      self._cmudict = cmudict.CMUDict(cmudict_path, keep_ambiguous=False)
      log('Loaded CMUDict with %d unambiguous entries' % len(self._cmudict))
    else:
      self._cmudict = None


  def start_in_session(self, session):
    self._session = session
    self.start()


  def run(self):
    try:
      while not self._coord.should_stop():
        self._enqueue_next_group()
    except Exception as e:
      traceback.print_exc()
      self._coord.request_stop(e)


  def _enqueue_next_group(self):
    start = time.time()

    # Read a group of examples:
    n = self._hparams.batch_size
    r = self._hparams.outputs_per_step
    examples = [self._get_next_example() for i in range(n * _batches_per_group)]

    # Bucket examples based on similar output sequence length for efficiency:
    examples.sort(key=lambda x: x[-1])
    batches = [examples[i:i+n] for i in range(0, len(examples), n)]
    random.shuffle(batches)

    log('Generated %d batches of size %d in %.03f sec' % (len(batches), n,
time.time() - start))
    for batch in batches:
      feed_dict = dict(zip(self._placeholders, _prepare_batch(batch, r)))
      self._session.run(self._enqueue_op, feed_dict=feed_dict)


  def _get_next_example(self):
    '''Loads a single example (input, mel_target, linear_target, cost) from
disk'''
    if self._offset >= len(self._metadata):
      self._offset = 0
      random.shuffle(self._metadata)
    meta = self._metadata[self._offset]
    self._offset += 1

    text = meta[3]
    if self._cmudict and random.random() < _p_cmudict:
      text = ' '.join([self._maybe_get_arpabet(word) for word in text.split('
')])

    input_data = np.asarray(text_to_sequence(text, self._cleaner_names),
dtype=np.int32)
    linear_target = np.load(os.path.join(self._datadir, meta[0]))
    mel_target = np.load(os.path.join(self._datadir, meta[1]))
    return (input_data, mel_target, linear_target, len(linear_target))


  def _maybe_get_arpabet(self, word):
    arpabet = self._cmudict.lookup(word)
    return '{%s}' % arpabet[0] if arpabet is not None and random.random() < 0.5
else word


def _prepare_batch(batch, outputs_per_step):
  random.shuffle(batch)
  inputs = _prepare_inputs([x[0] for x in batch])
  input_lengths = np.asarray([len(x[0]) for x in batch], dtype=np.int32)
  mel_targets = _prepare_targets([x[1] for x in batch], outputs_per_step)
  linear_targets = _prepare_targets([x[2] for x in batch], outputs_per_step)
  return (inputs, input_lengths, mel_targets, linear_targets)


def _prepare_inputs(inputs):
  max_len = max((len(x) for x in inputs))
  return np.stack([_pad_input(x, max_len) for x in inputs])


def _prepare_targets(targets, alignment):
  max_len = max((len(t) for t in targets)) + 1
  return np.stack([_pad_target(t, _round_up(max_len, alignment)) for t in
targets])


def _pad_input(x, length):
  return np.pad(x, (0, length - x.shape[0]), mode='constant',
constant_values=_pad)


def _pad_target(t, length):
  return np.pad(t, [(0, length - t.shape[0]), (0,0)], mode='constant',
constant_values=_pad)


def _round_up(x, multiple):
  remainder = x % multiple
  return x if remainder == 0 else x + multiple - remainder

================================================================================
====================

================================================================================
====================
📄 File: ljspeech.py
📂 Path: datasets\ljspeech.py
--------------------------------------------------------------------------------
--------------------
from concurrent.futures import ProcessPoolExecutor
from functools import partial
import numpy as np
import os
from util import audio


def build_from_path(in_dir, out_dir, num_workers=1, tqdm=lambda x: x):
  '''Preprocesses the LJ Speech dataset from a given input path into a given
output directory.

    Args:
      in_dir: The directory where you have downloaded the LJ Speech dataset
      out_dir: The directory to write the output into
      num_workers: Optional number of worker processes to parallelize across
      tqdm: You can optionally pass tqdm to get a nice progress bar
    Returns:
      A list of tuples describing the training examples. This should be written
to train.txt
  '''

  # We use ProcessPoolExecutor to parallelize across processes. This is just an
optimization and you
  # can omit it and just call _process_utterance on each input if you want.
  executor = ProcessPoolExecutor(max_workers=num_workers)
  futures = []
  index = 1
  with open(os.path.join(in_dir, 'metadata.csv'), encoding='utf-8') as f:
    for line in f:
      parts = line.strip().split('|')
      wav_path = os.path.join(in_dir, 'wavs', '%s.wav' % parts[0])
      text = parts[2]
      futures.append(executor.submit(partial(_process_utterance, out_dir, index,
wav_path, text)))
      index += 1
  return [future.result() for future in tqdm(futures)]


def _process_utterance(out_dir, index, wav_path, text):
  '''Preprocesses a single utterance audio/text pair.

  This writes the mel and linear scale spectrograms to disk and returns a tuple
to write
  to the train.txt file.

  Args:
    out_dir: The directory to write the spectrograms into
    index: The numeric index to use in the spectrogram filenames.
    wav_path: Path to the audio file containing the speech input
    text: The text spoken in the input audio file

  Returns:
    A (spectrogram_filename, mel_filename, n_frames, text) tuple to write to
train.txt
  '''

  # Load the audio to a numpy array:
  wav = audio.load_wav(wav_path)

  # Compute the linear-scale spectrogram from the wav:
  spectrogram = audio.spectrogram(wav).astype(np.float32)
  n_frames = spectrogram.shape[1]

  # Compute a mel-scale spectrogram from the wav:
  mel_spectrogram = audio.melspectrogram(wav).astype(np.float32)

  # Write the spectrograms to disk:
  spectrogram_filename = 'ljspeech-spec-%05d.npy' % index
  mel_filename = 'ljspeech-mel-%05d.npy' % index
  np.save(os.path.join(out_dir, spectrogram_filename), spectrogram.T,
allow_pickle=False)
  np.save(os.path.join(out_dir, mel_filename), mel_spectrogram.T,
allow_pickle=False)

  # Return a tuple describing this training example:
  return (spectrogram_filename, mel_filename, n_frames, text)

================================================================================
====================
```

==
====================
📄 File: nepali.py
📁 Path: datasets\nepali.py


```
from concurrent.futures import ProcessPoolExecutor
from functools import partial
import numpy as np
import os
from util import audio


def build_from_path(in_dir, out_dir, num_workers=1, tqdm=lambda x: x):
  '''Preprocesses the ne_np_female dataset from a given input path into a given
output directory.

    Args:
      in_dir: The directory where you have downloaded the ne_np_female Speech
dataset
      out_dir: The directory to write the output into
      num_workers: Optional number of worker processes to parallelize across
      tqdm: You can optionally pass tqdm to get a nice progress bar

    Returns:
      A list of tuples describing the training examples. This should be written
to train.txt
  '''

  # We use ProcessPoolExecutor to parallelize across processes. This is just an
optimization and you
  # can omit it and just call _process_utterance on each input if you want.
  executor = ProcessPoolExecutor(max_workers=num_workers)
  futures = []
  index = 1
  with open(os.path.join(in_dir, 'line_index.tsv'), encoding='utf-8') as f:
    for line in f:
      parts = line.strip().split('\t')
      wav_path = os.path.join(in_dir, 'wavs', '%s.wav' % parts[0])
      text = parts[1]
      futures.append(executor.submit(partial(_process_utterance, out_dir, index,
wav_path, text)))
      index += 1
  return [future.result() for future in tqdm(futures)]


def _process_utterance(out_dir, index, wav_path, text):
  '''Preprocesses a single utterance audio/text pair.

  This writes the mel and linear scale spectrograms to disk and returns a tuple
to write
  to the train.txt file.

  Args:
    out_dir: The directory to write the spectrograms into
    index: The numeric index to use in the spectrogram filenames.
    wav_path: Path to the audio file containing the speech input
    text: The text spoken in the input audio file

  Returns:
    A (spectrogram_filename, mel_filename, n_frames, text) tuple to write to
train.txt
  '''
  # Load the audio to a numpy array:
  wav = audio.load_wav(wav_path)
  # Compute the linear-scale spectrogram from the wav:
  spectrogram = audio.spectrogram(wav).astype(np.float32)
  n_frames = spectrogram.shape[1]

  # Compute a mel-scale spectrogram from the wav:
  mel_spectrogram = audio.melspectrogram(wav).astype(np.float32)

  # Write the spectrograms to disk:
  spectrogram_filename = 'nepali-spec-%05d.npy' % index
  mel_filename = 'nepali-mel-%05d.npy' % index
  np.save(os.path.join(out_dir, spectrogram_filename), spectrogram.T, 
allow_pickle=False)
  np.save(os.path.join(out_dir, mel_filename), mel_spectrogram.T, 
allow_pickle=False)

  # Return a tuple describing this training example:
  return (spectrogram_filename, mel_filename, n_frames, text)

================================================================================
====================

================================================================================
====================
📄 File: __init__.py
📁 Path: datasets\__init__.py
--------------------------------------------------------------------------------
--------------------

================================================================================
====================

================================================================================
====================
📄 File: helpers.py
📁 Path: models\helpers.py
--------------------------------------------------------------------------------
--------------------
import numpy as np
import tensorflow as tf
from tensorflow.contrib.seq2seq import Helper


# Adapted from tf.contrib.seq2seq.GreedyEmbeddingHelper
class TacoTestHelper(Helper):
  def __init__(self, batch_size, output_dim, r):
    with tf.name_scope('TacoTestHelper'):
      self._batch_size = batch_size
      self._output_dim = output_dim
      self._end_token = tf.tile([0.0], [output_dim * r])

  @property
  def batch_size(self):
    return self._batch_size

  @property
  def sample_ids_shape(self):
    return tf.TensorShape([])

  @property
  def sample_ids_dtype(self):
    return np.int32
  def initialize(self, name=None):
    return (tf.tile([False], [self._batch_size]), _go_frames(self._batch_size,
self._output_dim))

  def sample(self, time, outputs, state, name=None):
    return tf.tile([0], [self._batch_size])  # Return all 0; we ignore them

  def next_inputs(self, time, outputs, state, sample_ids, name=None):
    '''Stop on EOS. Otherwise, pass the last output as the next input and pass
through state.'''
    with tf.name_scope('TacoTestHelper'):
      finished = tf.reduce_all(tf.equal(outputs, self._end_token), axis=1)
      # Feed last output frame as next input. outputs is [N, output_dim * r]
      next_inputs = outputs[:, -self._output_dim:]
      return (finished, next_inputs, state)


class TacoTrainingHelper(Helper):
  def __init__(self, inputs, targets, output_dim, r):
    # inputs is [N, T_in], targets is [N, T_out, D]
    with tf.name_scope('TacoTrainingHelper'):
      self._batch_size = tf.shape(inputs)[0]
      self._output_dim = output_dim

      # Feed every r-th target frame as input
      self._targets = targets[:, r-1::r, :]

      # Use full length for every target because we don't want to mask the
padding frames
      num_steps = tf.shape(self._targets)[1]
      self._lengths = tf.tile([num_steps], [self._batch_size])

  @property
  def batch_size(self):
    return self._batch_size

  @property
  def sample_ids_shape(self):
    return tf.TensorShape([])

  @property
  def sample_ids_dtype(self):
    return np.int32

  def initialize(self, name=None):
    return (tf.tile([False], [self._batch_size]), _go_frames(self._batch_size,
self._output_dim))

  def sample(self, time, outputs, state, name=None):
    return tf.tile([0], [self._batch_size])  # Return all 0; we ignore them

  def next_inputs(self, time, outputs, state, sample_ids, name=None):
    with tf.name_scope(name or 'TacoTrainingHelper'):
      finished = (time + 1 >= self._lengths)
      next_inputs = self._targets[:, time, :]
      return (finished, next_inputs, state)


def _go_frames(batch_size, output_dim):
  '''Returns all-zero <GO> frames for a given batch size and output dimension'''
  return tf.tile([[0.0]], [batch_size, output_dim])
```

==
====================

==
====================
📄 File: modules.py
📁 Path: models\modules.py


```
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior() #NEW
# import tensorflow as tf #OLD


# from tensorflow.contrib.rnn import GRUCell  OLD

# New import
from tensorflow.compat.v1.nn.rnn_cell import GRUCell


def prenet(inputs, is_training, layer_sizes, scope=None):
  x = inputs
  drop_rate = 0.5 if is_training else 0.0
  with tf.variable_scope(scope or 'prenet'):
    for i, size in enumerate(layer_sizes):
      dense = tf.layers.dense(x, units=size, activation=tf.nn.relu,
name='dense_%d' % (i+1))
      x = tf.layers.dropout(dense, rate=drop_rate, training=is_training,
name='dropout_%d' % (i+1))
  return x


def encoder_cbhg(inputs, input_lengths, is_training, depth):
  input_channels = inputs.get_shape()[2]
  return cbhg(
    inputs,
    input_lengths,
    is_training,
    scope='encoder_cbhg',
    K=16,
    projections=[128, input_channels],
    depth=depth)


def post_cbhg(inputs, input_dim, is_training, depth):
  return cbhg(
    inputs,
    None,
    is_training,
    scope='post_cbhg',
    K=8,
    projections=[256, input_dim],
    depth=depth)


def cbhg(inputs, input_lengths, is_training, scope, K, projections, depth):
  with tf.variable_scope(scope):
    with tf.variable_scope('conv_bank'):
      # Convolution bank: concatenate on the last axis to stack channels from
all convolutions
      conv_outputs = tf.concat(
        [conv1d(inputs, k, 128, tf.nn.relu, is_training, 'conv1d_%d' % k) for k
in range(1, K+1)],
        axis=-1
      )

    # Maxpooling:
    maxpool_output = tf.layers.max_pooling1d(
      conv_outputs,
      pool_size=2,
      strides=1,
      padding='same')

    # Two projection layers:
    proj1_output = conv1d(maxpool_output, 3, projections[0], tf.nn.relu,
is_training, 'proj_1')
    proj2_output = conv1d(proj1_output, 3, projections[1], None, is_training,
'proj_2')

    # Residual connection:
    highway_input = proj2_output + inputs

    half_depth = depth // 2
    assert half_depth*2 == depth, 'encoder and postnet depths must be even.'

    # Handle dimensionality mismatch:
    if highway_input.shape[2] != half_depth:
      highway_input = tf.layers.dense(highway_input, half_depth)

    # 4-layer HighwayNet:
    for i in range(4):
      highway_input = highwaynet(highway_input, 'highway_%d' % (i+1),
half_depth)
    rnn_input = highway_input

    # Bidirectional RNN
    outputs, states = tf.nn.bidirectional_dynamic_rnn(
      GRUCell(half_depth),
      GRUCell(half_depth),
      rnn_input,
      sequence_length=input_lengths,
      dtype=tf.float32)
    return tf.concat(outputs, axis=2)  # Concat forward and backward


def highwaynet(inputs, scope, depth):
  with tf.variable_scope(scope):
    H = tf.layers.dense(
      inputs,
      units=depth,
      activation=tf.nn.relu,
      name='H')
    T = tf.layers.dense(
      inputs,
      units=depth,
      activation=tf.nn.sigmoid,
      name='T',
      bias_initializer=tf.constant_initializer(-1.0))
    return H * T + inputs * (1.0 - T)


def conv1d(inputs, kernel_size, channels, activation, is_training, scope):
  with tf.variable_scope(scope):
    conv1d_output = tf.layers.conv1d(
      inputs,
      filters=channels,
      kernel_size=kernel_size,
      activation=activation,
      padding='same')
    return tf.layers.batch_normalization(conv1d_output, training=is_training)

================================================================================
====================

================================================================================
====================
📄 File: rnn_wrappers.py
📁 Path: models\rnn_wrappers.py
--------------------------------------------------------------------------------
--------------------
import numpy as np


# import tensorflow as tf #OLD
# New import
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()


# from tensorflow.contrib.rnn import RNNCell  OLD
# New import
from tensorflow.compat.v1.nn.rnn_cell import RNNCell

from .modules import prenet


class DecoderPrenetWrapper(RNNCell):
  '''Runs RNN inputs through a prenet before sending them to the cell.'''
  def __init__(self, cell, is_training, layer_sizes):
    super(DecoderPrenetWrapper, self).__init__()
    self._cell = cell
    self._is_training = is_training
    self._layer_sizes = layer_sizes

  @property
  def state_size(self):
    return self._cell.state_size

  @property
  def output_size(self):
    return self._cell.output_size

  def call(self, inputs, state):
    prenet_out = prenet(inputs, self._is_training, self._layer_sizes,
scope='decoder_prenet')
    return self._cell(prenet_out, state)

  def zero_state(self, batch_size, dtype):
    return self._cell.zero_state(batch_size, dtype)



class ConcatOutputAndAttentionWrapper(RNNCell):
  '''Concatenates RNN cell output with the attention context vector.

  This is expected to wrap a cell wrapped with an AttentionWrapper constructed
with
  attention_layer_size=None and output_attention=False. Such a cell's state will
include an
  "attention" field that is the context vector.
  '''
  def __init__(self, cell):
    super(ConcatOutputAndAttentionWrapper, self).__init__()
    self._cell = cell

  @property
  def state_size(self):
    return self._cell.state_size

  @property
  def output_size(self):
    return self._cell.output_size + self._cell.state_size.attention

  def call(self, inputs, state):
    output, res_state = self._cell(inputs, state)
    return tf.concat([output, res_state.attention], axis=-1), res_state

  def zero_state(self, batch_size, dtype):
    return self._cell.zero_state(batch_size, dtype)
```

==
====================

==
====================
📄 File: tacotron.py
📂 Path: models\tacotron.py


```
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()#NEW
# import tensorflow as tf #

# from tensorflow.contrib.rnn import GRUCell, MultiRNNCell,
OutputProjectionWrapper, ResidualWrapper
# from tensorflow.contrib.seq2seq import BasicDecoder, BahdanauAttention,
AttentionWrapper  OLD
# New imports
from tensorflow.compat.v1.nn.rnn_cell import GRUCell, MultiRNNCell,
OutputProjectionWrapper, ResidualWrapper
from tensorflow.compat.v1.seq2seq import BasicDecoder, BahdanauAttention,
AttentionWrapper

from text.symbols import symbols
from util.infolog import log
from .helpers import TacoTestHelper, TacoTrainingHelper
from .modules import encoder_cbhg, post_cbhg, prenet
from .rnn_wrappers import DecoderPrenetWrapper, ConcatOutputAndAttentionWrapper



class Tacotron():
  def __init__(self, hparams):
    self._hparams = hparams


  def initialize(self, inputs, input_lengths, mel_targets=None,
linear_targets=None):
    '''Initializes the model for inference.

    Sets "mel_outputs", "linear_outputs", and "alignments" fields.

    Args:
      inputs: int32 Tensor with shape [N, T_in] where N is batch size, T_in is
number of
        steps in the input time series, and values are character IDs
      input_lengths: int32 Tensor with shape [N] where N is batch size and
values are the lengths
        of each sequence in inputs.
      mel_targets: float32 Tensor with shape [N, T_out, M] where N is batch
size, T_out is number
        of steps in the output time series, M is num_mels, and values are
entries in the mel
        spectrogram. Only needed for training.
      linear_targets: float32 Tensor with shape [N, T_out, F] where N is
batch_size, T_out is number
        of steps in the output time series, F is num_freq, and values are
entries in the linear
        spectrogram. Only needed for training.
    '''
    with tf.variable_scope('inference') as scope:
      is_training = linear_targets is not None
      batch_size = tf.shape(inputs)[0]
      hp = self._hparams

      # Embeddings
      embedding_table = tf.get_variable(
        'embedding', [len(symbols), hp.embed_depth], dtype=tf.float32,
        initializer=tf.truncated_normal_initializer(stddev=0.5))
      embedded_inputs = tf.nn.embedding_lookup(embedding_table, inputs)
# [N, T_in, embed_depth=256]

      # Encoder
      prenet_outputs = prenet(embedded_inputs, is_training, hp.prenet_depths)
# [N, T_in, prenet_depths[-1]=128]
      encoder_outputs = encoder_cbhg(prenet_outputs, input_lengths, is_training,
# [N, T_in, encoder_depth=256]
                                     hp.encoder_depth)

      # Attention
      attention_cell = AttentionWrapper(
        GRUCell(hp.attention_depth),
        BahdanauAttention(hp.attention_depth, encoder_outputs),
        alignment_history=True,
        output_attention=False)
# [N, T_in, attention_depth=256]

      # Apply prenet before concatenation in AttentionWrapper.
      attention_cell = DecoderPrenetWrapper(attention_cell, is_training,
hp.prenet_depths)

      # Concatenate attention context vector and RNN cell output into a
2*attention_depth=512D vector.
      concat_cell = ConcatOutputAndAttentionWrapper(attention_cell)
# [N, T_in, 2*attention_depth=512]

      # Decoder (layers specified bottom to top):
      decoder_cell = MultiRNNCell([
          OutputProjectionWrapper(concat_cell, hp.decoder_depth),
          ResidualWrapper(GRUCell(hp.decoder_depth)),
          ResidualWrapper(GRUCell(hp.decoder_depth))
        ], state_is_tuple=True)
# [N, T_in, decoder_depth=256]

      # Project onto r mel spectrograms (predict r outputs at each RNN step):
      output_cell = OutputProjectionWrapper(decoder_cell, hp.num_mels *
hp.outputs_per_step)
      decoder_init_state = output_cell.zero_state(batch_size=batch_size,
dtype=tf.float32)
```

```
      if is_training:
        helper = TacoTrainingHelper(inputs, mel_targets, hp.num_mels,
hp.outputs_per_step)
      else:
        helper = TacoTestHelper(batch_size, hp.num_mels, hp.outputs_per_step)

      (decoder_outputs, _), final_decoder_state, _ =
tf.contrib.seq2seq.dynamic_decode(
        BasicDecoder(output_cell, helper, decoder_init_state),
        maximum_iterations=hp.max_iters)
# [N, T_out/r, M*r]

      # Reshape outputs to be one output per entry
      mel_outputs = tf.reshape(decoder_outputs, [batch_size, -1, hp.num_mels])
# [N, T_out, M]

      # Add post-processing CBHG:
      post_outputs = post_cbhg(mel_outputs, hp.num_mels, is_training,
# [N, T_out, postnet_depth=256]
                               hp.postnet_depth)
      linear_outputs = tf.layers.dense(post_outputs, hp.num_freq)
# [N, T_out, F]

      # Grab alignments from the final decoder state:
      alignments =
tf.transpose(final_decoder_state[0].alignment_history.stack(), [1, 2, 0])

      self.inputs = inputs
      self.input_lengths = input_lengths
      self.mel_outputs = mel_outputs
      self.linear_outputs = linear_outputs
      self.alignments = alignments
      self.mel_targets = mel_targets
      self.linear_targets = linear_targets
      log('Initialized Tacotron model. Dimensions: ')
      log('  embedding:               %d' % embedded_inputs.shape[-1])
      log('  prenet out:              %d' % prenet_outputs.shape[-1])
      log('  encoder out:             %d' % encoder_outputs.shape[-1])
      log('  attention out:           %d' % attention_cell.output_size)
      log('  concat attn & out:       %d' % concat_cell.output_size)
      log('  decoder cell out:        %d' % decoder_cell.output_size)
      log('  decoder out (%d frames):  %d' % (hp.outputs_per_step,
decoder_outputs.shape[-1]))
      log('  decoder out (1 frame):   %d' % mel_outputs.shape[-1])
      log('  postnet out:             %d' % post_outputs.shape[-1])
      log('  linear out:              %d' % linear_outputs.shape[-1])


  def add_loss(self):
    '''Adds loss to the model. Sets "loss" field. initialize must have been
called.'''
    with tf.variable_scope('loss') as scope:
      hp = self._hparams
      self.mel_loss = tf.reduce_mean(tf.abs(self.mel_targets -
self.mel_outputs))
      l1 = tf.abs(self.linear_targets - self.linear_outputs)
      # Prioritize loss for frequencies under 3000 Hz.
      n_priority_freq = int(3000 / (hp.sample_rate * 0.5) * hp.num_freq)
      self.linear_loss = 0.5 * tf.reduce_mean(l1) + 0.5 *
tf.reduce_mean(l1[:,:,0:n_priority_freq])
      self.loss = self.mel_loss + self.linear_loss


  def add_optimizer(self, global_step):
    '''Adds optimizer. Sets "gradients" and "optimize" fields. add_loss must
have been called.

    Args:
      global_step: int32 scalar Tensor representing current global step in
training
    '''
    with tf.variable_scope('optimizer') as scope:
      hp = self._hparams
      if hp.decay_learning_rate:
        self.learning_rate = _learning_rate_decay(hp.initial_learning_rate,
global_step)
      else:
        self.learning_rate = tf.convert_to_tensor(hp.initial_learning_rate)
      optimizer = tf.train.AdamOptimizer(self.learning_rate, hp.adam_beta1,
hp.adam_beta2)
      gradients, variables = zip(*optimizer.compute_gradients(self.loss))
      self.gradients = gradients
      clipped_gradients, _ = tf.clip_by_global_norm(gradients, 1.0)

      # Add dependency on UPDATE_OPS; otherwise batchnorm won't work correctly.
See:
      # https://github.com/tensorflow/tensorflow/issues/1122
      with tf.control_dependencies(tf.get_collection(tf.GraphKeys.UPDATE_OPS)):
        self.optimize = optimizer.apply_gradients(zip(clipped_gradients,
variables),
          global_step=global_step)


def _learning_rate_decay(init_lr, global_step):
  # Noam scheme from tensor2tensor:
  warmup_steps = 4000.0
  step = tf.cast(global_step + 1, dtype=tf.float32)
  return init_lr * warmup_steps**0.5 * tf.minimum(step * warmup_steps**-1.5,
step**-0.5)

================================================================================
====================

================================================================================
====================
📄 File: __init__.py
📁 Path: models\__init__.py
--------------------------------------------------------------------------------
--------------------
from .tacotron import Tacotron


def create_model(name, hparams):
  if name == 'tacotron':
    return Tacotron(hparams)
  else:
    raise Exception('Unknown model: ' + name)

================================================================================
====================

================================================================================
====================
📄 File: cmudict_test.py
📁 Path: tests\cmudict_test.py
--------------------------------------------------------------------------------
--------------------
import io
```

```
from text import cmudict


test_data = '''
;;; # CMUdict  --  Major Version: 0.07
)PAREN  P ER EH N
'TIS  T IH Z
ADVERSE  AE0 D V ER1 S
ADVERSE(1)  AE1 D V ER2 S
ADVERSE(2)  AE2 D V ER1 S
ADVERSELY  AE0 D V ER1 S L IY0
ADVERSITY  AE0 D V ER1 S IH0 T IY2
BARBERSHOP  B AA1 R B ER0 SH AA2 P
YOU'LL  Y UW1 L
'''


def test_cmudict():
  c = cmudict.CMUDict(io.StringIO(test_data))
  assert len(c) == 6
  assert len(cmudict.valid_symbols) == 84
  assert c.lookup('ADVERSITY') == ['AE0 D V ER1 S IH0 T IY2']
  assert c.lookup('BarberShop') == ['B AA1 R B ER0 SH AA2 P']
  assert c.lookup("You'll") == ['Y UW1 L']
  assert c.lookup("'tis") == ['T IH Z']
  assert c.lookup('adverse') == [
    'AE0 D V ER1 S',
    'AE1 D V ER2 S',
    'AE2 D V ER1 S',
  ]
  assert c.lookup('') == None
  assert c.lookup('foo') == None
  assert c.lookup(')paren') == None


def test_cmudict_no_keep_ambiguous():
  c = cmudict.CMUDict(io.StringIO(test_data), keep_ambiguous=False)
  assert len(c) == 5
  assert c.lookup('adversity') == ['AE0 D V ER1 S IH0 T IY2']
  assert c.lookup('adverse') == None
```

==
====================

==
====================

📄 File: numbers_test.py

📁 Path: tests\numbers_test.py


```
from text.numbers import normalize_numbers


def test_normalize_numbers():
  assert normalize_numbers('1') == 'one'
  assert normalize_numbers('15') == 'fifteen'
  assert normalize_numbers('24') == 'twenty-four'
  assert normalize_numbers('100') == 'one hundred'
  assert normalize_numbers('101') == 'one hundred one'
  assert normalize_numbers('456') == 'four hundred fifty-six'
  assert normalize_numbers('1000') == 'one thousand'
  assert normalize_numbers('1800') == 'eighteen hundred'
  assert normalize_numbers('2,000') == 'two thousand'
  assert normalize_numbers('3000') == 'three thousand'
  assert normalize_numbers('18000') == 'eighteen thousand'
  assert normalize_numbers('24,000') == 'twenty-four thousand'
  assert normalize_numbers('124,001') == 'one hundred twenty-four thousand one'
  assert normalize_numbers('6.4 sec') == 'six point four sec'


def test_normalize_ordinals():
  assert normalize_numbers('1st') == 'first'
  assert normalize_numbers('2nd') == 'second'
  assert normalize_numbers('9th') == 'ninth'
  assert normalize_numbers('243rd place') == 'two hundred and forty-third place'


def test_normalize_dates():
  assert normalize_numbers('1400') == 'fourteen hundred'
  assert normalize_numbers('1901') == 'nineteen oh one'
  assert normalize_numbers('1999') == 'nineteen ninety-nine'
  assert normalize_numbers('2000') == 'two thousand'
  assert normalize_numbers('2004') == 'two thousand four'
  assert normalize_numbers('2010') == 'twenty ten'
  assert normalize_numbers('2012') == 'twenty twelve'
  assert normalize_numbers('2025') == 'twenty twenty-five'
  assert normalize_numbers('September 11, 2001') == 'September eleven, two
thousand one'
  assert normalize_numbers('July 26, 1984.') == 'July twenty-six, nineteen
eighty-four.'


def test_normalize_money():
  assert normalize_numbers('$0.00') == 'zero dollars'
  assert normalize_numbers('$1') == 'one dollar'
  assert normalize_numbers('$10') == 'ten dollars'
  assert normalize_numbers('$.01') == 'one cent'
  assert normalize_numbers('$0.25') == 'twenty-five cents'
  assert normalize_numbers('$5.00') == 'five dollars'
  assert normalize_numbers('$5.01') == 'five dollars, one cent'
  assert normalize_numbers('$135.99.') == 'one hundred thirty-five dollars,
ninety-nine cents.'
  assert normalize_numbers('$40,000') == 'forty thousand dollars'
  assert normalize_numbers('for £2500!') == 'for twenty-five hundred pounds!'
```

==
====================

==
====================

📄 File: text_test.py

📁 Path: tests\text_test.py


```
from text import cleaners, symbols, text_to_sequence, sequence_to_text
from unidecode import unidecode


def test_symbols():
  assert len(symbols) >= 3
  assert symbols[0] == '_'
  assert symbols[1] == '~'


def test_text_to_sequence():
  assert text_to_sequence('', []) == [1]
  assert text_to_sequence('Hi!', []) == [9, 36, 54, 1]
  assert text_to_sequence('"A"_B', []) == [2, 3, 1]
  assert text_to_sequence('A {AW1 S} B', []) == [2, 64, 83, 132, 64, 3, 1]
  assert text_to_sequence('Hi', ['lowercase']) == [35, 36, 1]
  assert text_to_sequence('A {AW1 S}  B', ['english_cleaners']) == [28, 64, 83,
132, 64, 29, 1]


def test_sequence_to_text():
  assert sequence_to_text([]) == ''
  assert sequence_to_text([1]) == '~'
  assert sequence_to_text([9, 36, 54, 1]) == 'Hi!~'
  assert sequence_to_text([2, 64, 83, 132, 64, 3]) == 'A {AW1 S} B'


def test_collapse_whitespace():
  assert cleaners.collapse_whitespace('') == ''
  assert cleaners.collapse_whitespace('  ') == ' '
  assert cleaners.collapse_whitespace('x') == 'x'
  assert cleaners.collapse_whitespace(' x.  y,  \tz') == ' x. y, z'


def test_convert_to_ascii():
  assert cleaners.convert_to_ascii("raison d'être") == "raison d'etre"
  assert cleaners.convert_to_ascii('grüß gott') == 'gruss gott'
  assert cleaners.convert_to_ascii('안녕') == 'annyeong'
  assert cleaners.convert_to_ascii('Здравствуйте') == 'Zdravstvuite'


def test_lowercase():
  assert cleaners.lowercase('Happy Birthday!') == 'happy birthday!'
  assert cleaners.lowercase('CAFÉ') == 'café'


def test_expand_abbreviations():
  assert cleaners.expand_abbreviations('mr. and mrs. smith') == 'mister and
misess smith'


def test_expand_numbers():
  assert cleaners.expand_numbers('3 apples and 44 pears') == 'three apples and
forty-four pears'
  assert cleaners.expand_numbers('$3.50 for gas.') == 'three dollars, fifty
cents for gas.'


def test_cleaner_pipelines():
  text = 'Mr. Müller ate  2 Apples'
  assert cleaners.english_cleaners(text) == 'mister muller ate two apples'
  assert cleaners.transliteration_cleaners(text) == 'mr. muller ate 2 apples'
  assert cleaners.basic_cleaners(text) == 'mr. müller ate 2 apples'


================================================================================
====================

================================================================================
====================
📄 File: __init__.py
📁 Path: tests\__init__.py
--------------------------------------------------------------------------------
--------------------

================================================================================
====================
```

==
====================
📄 File: cleaners.py
📁 Path: text\cleaners.py


```
'''
Cleaners are transformations that run over the input text at both training and
eval time.

Cleaners can be selected by passing a comma-delimited list of cleaner names as
the "cleaners"
hyperparameter. Some cleaners are English-specific. You'll typically want to
use:
  1. "english_cleaners" for English text
  2. "transliteration_cleaners" for non-English text that can be transliterated
to ASCII using
     the Unidecode library (https://pypi.python.org/pypi/Unidecode)
  3. "basic_cleaners" if you do not want to transliterate (in this case, you
should also update
     the symbols in symbols.py to match your data).
'''

import re
from unidecode import unidecode
from .numbers import normalize_numbers


# Regular expression matching whitespace:
_whitespace_re = re.compile(r'\s+')

# List of (regular expression, replacement) pairs for abbreviations:
_abbreviations = [(re.compile('\\b%s\\.' % x[0], re.IGNORECASE), x[1]) for x in
[
  ('mrs', 'misess'),
  ('mr', 'mister'),
  ('dr', 'doctor'),
  ('st', 'saint'),
  ('co', 'company'),
  ('jr', 'junior'),
  ('maj', 'major'),
  ('gen', 'general'),
  ('drs', 'doctors'),
  ('rev', 'reverend'),
  ('lt', 'lieutenant'),
  ('hon', 'honorable'),
  ('sgt', 'sergeant'),
  ('capt', 'captain'),
  ('esq', 'esquire'),
  ('ltd', 'limited'),
  ('col', 'colonel'),
  ('ft', 'fort'),
]]


def expand_abbreviations(text):
  for regex, replacement in _abbreviations:
    text = re.sub(regex, replacement, text)
  return text


def expand_numbers(text):
  return normalize_numbers(text)
```

```
def lowercase(text):
  return text.lower()


def collapse_whitespace(text):
  return re.sub(_whitespace_re, ' ', text)


def convert_to_ascii(text):
  return unidecode(text)


def basic_cleaners(text):
  '''Basic pipeline that lowercases and collapses whitespace without 
transliteration.'''
  text = lowercase(text)
  text = collapse_whitespace(text)
  return text


def transliteration_cleaners(text):
  '''Pipeline for non-English text that transliterates to ASCII.'''
  text = convert_to_ascii(text)
  text = lowercase(text)
  text = collapse_whitespace(text)
  return text


def english_cleaners(text):
  '''Pipeline for English text, including number and abbreviation expansion.'''
  text = convert_to_ascii(text)
  text = lowercase(text)
  text = expand_numbers(text)
  text = expand_abbreviations(text)
  text = collapse_whitespace(text)
  return text

================================================================================
====================

================================================================================
====================
📄 File: cmudict.py
📁 Path: text\cmudict.py
--------------------------------------------------------------------------------
--------------------
import re


valid_symbols = [
  'AA', 'AA0', 'AA1', 'AA2', 'AE', 'AE0', 'AE1', 'AE2', 'AH', 'AH0', 'AH1', 
'AH2',
  'AO', 'AO0', 'AO1', 'AO2', 'AW', 'AW0', 'AW1', 'AW2', 'AY', 'AY0', 'AY1', 
'AY2',
  'B', 'CH', 'D', 'DH', 'EH', 'EH0', 'EH1', 'EH2', 'ER', 'ER0', 'ER1', 'ER2', 
'EY',
  'EY0', 'EY1', 'EY2', 'F', 'G', 'HH', 'IH', 'IH0', 'IH1', 'IH2', 'IY', 'IY0', 
'IY1',
  'IY2', 'JH', 'K', 'L', 'M', 'N', 'NG', 'OW', 'OW0', 'OW1', 'OW2', 'OY', 'OY0',
  'OY1', 'OY2', 'P', 'R', 'S', 'SH', 'T', 'TH', 'UH', 'UH0', 'UH1', 'UH2', 'UW',
  'UW0', 'UW1', 'UW2', 'V', 'W', 'Y', 'Z', 'ZH'
]
```

```
_valid_symbol_set = set(valid_symbols)


class CMUDict:
  '''Thin wrapper around CMUDict data.
http://www.speech.cs.cmu.edu/cgi-bin/cmudict'''
  def __init__(self, file_or_path, keep_ambiguous=True):
    if isinstance(file_or_path, str):
      with open(file_or_path, encoding='latin-1') as f:
        entries = _parse_cmudict(f)
    else:
      entries = _parse_cmudict(file_or_path)
    if not keep_ambiguous:
      entries = {word: pron for word, pron in entries.items() if len(pron) == 1}
    self._entries = entries


  def __len__(self):
    return len(self._entries)


  def lookup(self, word):
    '''Returns list of ARPAbet pronunciations of the given word.'''
    return self._entries.get(word.upper())



_alt_re = re.compile(r'\([0-9]+\)')


def _parse_cmudict(file):
  cmudict = {}
  for line in file:
    if len(line) and (line[0] >= 'A' and line[0] <= 'Z' or line[0] == "'"):
      parts = line.split('  ')
      word = re.sub(_alt_re, '', parts[0])
      pronunciation = _get_pronunciation(parts[1])
      if pronunciation:
        if word in cmudict:
          cmudict[word].append(pronunciation)
        else:
          cmudict[word] = [pronunciation]
  return cmudict


def _get_pronunciation(s):
  parts = s.strip().split(' ')
  for part in parts:
    if part not in _valid_symbol_set:
      return None
  return ' '.join(parts)

================================================================================
====================

================================================================================
====================
📄 File: numbers.py
📁 Path: text\numbers.py
--------------------------------------------------------------------------------
--------------------
import inflect
import re
```

```
_inflect = inflect.engine()
_comma_number_re = re.compile(r'([0-9][0-9\,]+[0-9])')
_decimal_number_re = re.compile(r'([0-9]+\.[0-9]+)')
_pounds_re = re.compile(r'£([0-9\,]*[0-9]+)')
_dollars_re = re.compile(r'\$([0-9\.\,]*[0-9]+)')
_ordinal_re = re.compile(r'[0-9]+(st|nd|rd|th)')
_number_re = re.compile(r'[0-9]+')


def _remove_commas(m):
  return m.group(1).replace(',', '')


def _expand_decimal_point(m):
  return m.group(1).replace('.', ' point ')


def _expand_dollars(m):
  match = m.group(1)
  parts = match.split('.')
  if len(parts) > 2:
    return match + ' dollars'  # Unexpected format
  dollars = int(parts[0]) if parts[0] else 0
  cents = int(parts[1]) if len(parts) > 1 and parts[1] else 0
  if dollars and cents:
    dollar_unit = 'dollar' if dollars == 1 else 'dollars'
    cent_unit = 'cent' if cents == 1 else 'cents'
    return '%s %s, %s %s' % (dollars, dollar_unit, cents, cent_unit)
  elif dollars:
    dollar_unit = 'dollar' if dollars == 1 else 'dollars'
    return '%s %s' % (dollars, dollar_unit)
  elif cents:
    cent_unit = 'cent' if cents == 1 else 'cents'
    return '%s %s' % (cents, cent_unit)
  else:
    return 'zero dollars'


def _expand_ordinal(m):
  return _inflect.number_to_words(m.group(0))


def _expand_number(m):
  num = int(m.group(0))
  if num > 1000 and num < 3000:
    if num == 2000:
      return 'two thousand'
    elif num > 2000 and num < 2010:
      return 'two thousand ' + _inflect.number_to_words(num % 100)
    elif num % 100 == 0:
      return _inflect.number_to_words(num // 100) + ' hundred'
    else:
      return _inflect.number_to_words(num, andword='', zero='oh',
group=2).replace(', ', ' ')
  else:
    return _inflect.number_to_words(num, andword='')


def normalize_numbers(text):
  text = re.sub(_comma_number_re, _remove_commas, text)
  text = re.sub(_pounds_re, r'\1 pounds', text)
  text = re.sub(_dollars_re, _expand_dollars, text)
  text = re.sub(_decimal_number_re, _expand_decimal_point, text)
  text = re.sub(_ordinal_re, _expand_ordinal, text)
  text = re.sub(_number_re, _expand_number, text)
  return text

================================================================================
====================

================================================================================
====================
📄 File: symbols.py
📂 Path: text\symbols.py
--------------------------------------------------------------------------------
--------------------
'''
Defines the set of symbols used in text input to the model.

The default is a set of ASCII characters that works well for English or text
that has been run
through Unidecode. For other data, you can modify _characters. See
TRAINING_DATA.md for details.
'''
from text import cmudict

_pad        = '_'
_eos        = '~'
_characters = 'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz!\'(),-.:;? '

# Prepend "@" to ARPAbet symbols to ensure uniqueness (some are the same as
uppercase letters):
_arpabet = ['@' + s for s in cmudict.valid_symbols]

# Export all symbols:
symbols = [_pad, _eos] + list(_characters) + _arpabet

================================================================================
====================

================================================================================
====================
📄 File: __init__.py
📂 Path: text\__init__.py
--------------------------------------------------------------------------------
--------------------
import re
from text import cleaners
from text.symbols import symbols


# Mappings from symbol to numeric ID and vice versa:
_symbol_to_id = {s: i for i, s in enumerate(symbols)}
_id_to_symbol = {i: s for i, s in enumerate(symbols)}

# Regular expression matching text enclosed in curly braces:
_curly_re = re.compile(r'(.*?)\{(.+?)\}(.*)')


def text_to_sequence(text, cleaner_names):
  '''Converts a string of text to a sequence of IDs corresponding to the symbols
in the text.

    The text can optionally have ARPAbet sequences enclosed in curly braces
embedded
    in it. For example, "Turn left on {HH AW1 S S T AH0 N} Street."
    Args:
      text: string to convert to a sequence
      cleaner_names: names of the cleaner functions to run the text through

    Returns:
      List of integers corresponding to the symbols in the text
  '''
  sequence = []

  # Check for curly braces and treat their contents as ARPAbet:
  while len(text):
    m = _curly_re.match(text)
    if not m:
      sequence += _symbols_to_sequence(_clean_text(text, cleaner_names))
      break
    sequence += _symbols_to_sequence(_clean_text(m.group(1), cleaner_names))
    sequence += _arpabet_to_sequence(m.group(2))
    text = m.group(3)

  # Append EOS token
  sequence.append(_symbol_to_id['~'])
  return sequence


def sequence_to_text(sequence):
  '''Converts a sequence of IDs back to a string'''
  result = ''
  for symbol_id in sequence:
    if symbol_id in _id_to_symbol:
      s = _id_to_symbol[symbol_id]
      # Enclose ARPAbet back in curly braces:
      if len(s) > 1 and s[0] == '@':
        s = '{%s}' % s[1:]
      result += s
  return result.replace('}{', ' ')


def _clean_text(text, cleaner_names):
  for name in cleaner_names:
    cleaner = getattr(cleaners, name)
    if not cleaner:
      raise Exception('Unknown cleaner: %s' % name)
    text = cleaner(text)
  return text


def _symbols_to_sequence(symbols):
  return [_symbol_to_id[s] for s in symbols if _should_keep_symbol(s)]


def _arpabet_to_sequence(text):
  return _symbols_to_sequence(['@' + s for s in text.split()])


def _should_keep_symbol(s):
  return s in _symbol_to_id and s is not '_' and s is not '~'
```

==
====================

==
====================
📄 File: audio.py
📁 Path: util\audio.py

```
--------------------------------------------------------------------------------
--------------------
import librosa
import librosa.filters
import math
import numpy as np

# Old import
# import tensorflow as tf
# New import
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()
# Update stft/istft calls
from tensorflow.compat.v1.signal import stft as tf_stft, inverse_stft as
tf_istft



import scipy
from hparams import hparams


def load_wav(path):
  return librosa.core.load(path, sr=hparams.sample_rate)[0]


def save_wav(wav, path):
  wav *= 32767 / max(0.01, np.max(np.abs(wav)))
  scipy.io.wavfile.write(path, hparams.sample_rate, wav.astype(np.int16))


def preemphasis(x):
  return scipy.signal.lfilter([1, -hparams.preemphasis], [1], x)


def inv_preemphasis(x):
  return scipy.signal.lfilter([1], [1, -hparams.preemphasis], x)


def spectrogram(y):
  D = _stft(preemphasis(y))
  S = _amp_to_db(np.abs(D)) - hparams.ref_level_db
  return _normalize(S)


def inv_spectrogram(spectrogram):
  '''Converts spectrogram to waveform using librosa'''
  S = _db_to_amp(_denormalize(spectrogram) + hparams.ref_level_db)  # Convert
back to linear
  return inv_preemphasis(_griffin_lim(S ** hparams.power))          #
Reconstruct phase


def inv_spectrogram_tensorflow(spectrogram):
  '''Builds computational graph to convert spectrogram to waveform using
TensorFlow.

  Unlike inv_spectrogram, this does NOT invert the preemphasis. The caller
should call
  inv_preemphasis on the output after running the graph.
  '''
  S = _db_to_amp_tensorflow(_denormalize_tensorflow(spectrogram) +
hparams.ref_level_db)
  return _griffin_lim_tensorflow(tf.pow(S, hparams.power))
```

```
def melspectrogram(y):
  D = _stft(preemphasis(y))
  S = _amp_to_db(_linear_to_mel(np.abs(D))) - hparams.ref_level_db
  return _normalize(S)


def find_endpoint(wav, threshold_db=-40, min_silence_sec=0.8):
  window_length = int(hparams.sample_rate * min_silence_sec)
  hop_length = int(window_length / 4)
  threshold = _db_to_amp(threshold_db)
  for x in range(hop_length, len(wav) - window_length, hop_length):
    if np.max(wav[x:x+window_length]) < threshold:
      return x + hop_length
  return len(wav)


def _griffin_lim(S):
  '''librosa implementation of Griffin-Lim
  Based on https://github.com/librosa/librosa/issues/434
  '''
  angles = np.exp(2j * np.pi * np.random.rand(*S.shape))
  S_complex = np.abs(S).astype(np.complex)
  y = _istft(S_complex * angles)
  for i in range(hparams.griffin_lim_iters):
    angles = np.exp(1j * np.angle(_stft(y)))
    y = _istft(S_complex * angles)
  return y


def _griffin_lim_tensorflow(S):
  '''TensorFlow implementation of Griffin-Lim
  Based on
https://github.com/Kyubyong/tensorflow-exercises/blob/master/Audio_Processing.ip
ynb
  '''
  with tf.variable_scope('griffinlim'):
    # TensorFlow's stft and istft operate on a batch of spectrograms; create
batch of size 1
    S = tf.expand_dims(S, 0)
    S_complex = tf.identity(tf.cast(S, dtype=tf.complex64))
    y = _istft_tensorflow(S_complex)
    for i in range(hparams.griffin_lim_iters):
      est = _stft_tensorflow(y)
      angles = est / tf.cast(tf.maximum(1e-8, tf.abs(est)), tf.complex64)
      y = _istft_tensorflow(S_complex * angles)
    return tf.squeeze(y, 0)


def _stft(y):
  n_fft, hop_length, win_length = _stft_parameters()
  return librosa.stft(y=y, n_fft=n_fft, hop_length=hop_length,
win_length=win_length)


def _istft(y):
  _, hop_length, win_length = _stft_parameters()
  return librosa.istft(y, hop_length=hop_length, win_length=win_length)


# def _stft_tensorflow(signals):
#   n_fft, hop_length, win_length = _stft_parameters()
#   return tf.contrib.signal.stft(signals, win_length, hop_length, n_fft,
pad_end=False)
# def _istft_tensorflow(stfts):
#   n_fft, hop_length, win_length = _stft_parameters()
#   return tf.contrib.signal.inverse_stft(stfts, win_length, hop_length, n_fft)
def _stft_tensorflow(signals):
    n_fft, hop_length, win_length = _stft_parameters()
    return tf.signal.stft(signals, win_length, hop_length, n_fft, pad_end=False)

def _istft_tensorflow(stfts):
    n_fft, hop_length, win_length = _stft_parameters()
    return tf.signal.inverse_stft(stfts, win_length, hop_length, n_fft)


def _stft_parameters():
  n_fft = (hparams.num_freq - 1) * 2
  hop_length = int(hparams.frame_shift_ms / 1000 * hparams.sample_rate)
  win_length = int(hparams.frame_length_ms / 1000 * hparams.sample_rate)
  return n_fft, hop_length, win_length


# Conversions:

_mel_basis = None

def _linear_to_mel(spectrogram):
  global _mel_basis
  if _mel_basis is None:
    _mel_basis = _build_mel_basis()
  return np.dot(_mel_basis, spectrogram)

def _build_mel_basis():
  n_fft = (hparams.num_freq - 1) * 2
  return librosa.filters.mel(hparams.sample_rate, n_fft,
n_mels=hparams.num_mels)

def _amp_to_db(x):
  return 20 * np.log10(np.maximum(1e-5, x))

def _db_to_amp(x):
  return np.power(10.0, x * 0.05)

def _db_to_amp_tensorflow(x):
  return tf.pow(tf.ones(tf.shape(x)) * 10.0, x * 0.05)

def _normalize(S):
  return np.clip((S - hparams.min_level_db) / -hparams.min_level_db, 0, 1)

def _denormalize(S):
  return (np.clip(S, 0, 1) * -hparams.min_level_db) + hparams.min_level_db

def _denormalize_tensorflow(S):
  return (tf.clip_by_value(S, 0, 1) * -hparams.min_level_db) +
hparams.min_level_db

================================================================================
====================

================================================================================
====================
📄 File: infolog.py
📁 Path: util\infolog.py
--------------------------------------------------------------------------------
--------------------
import atexit
```

```
from datetime import datetime
import json
from threading import Thread
from urllib.request import Request, urlopen


_format = '%Y-%m-%d %H:%M:%S.%f'
_file = None
_run_name = None
_slack_url = None


def init(filename, run_name, slack_url=None):
  global _file, _run_name, _slack_url
  _close_logfile()
  _file = open(filename, 'a', encoding="utf-8")

_file.write('\n-----------------------------------------------------------------
\n')
  _file.write('Starting new training run\n')
  _file.write('-----------------------------------------------------------------
\n')
  _run_name = run_name
  _slack_url = slack_url


def log(msg, slack=False):
  print(msg)
  if _file is not None:
    _file.write('[%s]  %s\n' % (datetime.now().strftime(_format)[:-3], msg))
  if slack and _slack_url is not None:
    Thread(target=_send_slack, args=(msg,)).start()


def _close_logfile():
  global _file
  if _file is not None:
    _file.close()
    _file = None


def _send_slack(msg):
  req = Request(_slack_url)
  req.add_header('Content-Type', 'application/json')
  urlopen(req, json.dumps({
    'username': 'tacotron',
    'icon_emoji': ':taco:',
    'text': '*%s*: %s' % (_run_name, msg)
  }).encode())


atexit.register(_close_logfile)

================================================================================
====================

================================================================================
====================
📄 File: plot.py
📂 Path: util\plot.py
--------------------------------------------------------------------------------
--------------------
import matplotlib
matplotlib.use('Agg')
```

```
import matplotlib.pyplot as plt


def plot_alignment(alignment, path, info=None):
  fig, ax = plt.subplots()
  im = ax.imshow(
    alignment,
    aspect='auto',
    origin='lower',
    interpolation='none')
  fig.colorbar(im, ax=ax)
  xlabel = 'Decoder timestep'
  if info is not None:
    xlabel += '\n\n' + info
  plt.xlabel(xlabel)
  plt.ylabel('Encoder timestep')
  plt.tight_layout()
  plt.savefig(path, format='png')
```

==
====================

==
====================
📄 File: __init__.py
📁 Path: util__init__.py


```
class ValueWindow():
  def __init__(self, window_size=100):
    self._window_size = window_size
    self._values = []

  def append(self, x):
    self._values = self._values[-(self._window_size - 1):] + [x]

  @property
  def sum(self):
    return sum(self._values)

  @property
  def count(self):
    return len(self._values)

  @property
  def average(self):
    return self.sum / max(1, self.count)

  def reset(self):
    self._values = []
```

==
====================