shabda Documentation

Shabda Team

Nov 13, 2018

Contents

CHAPTER 1

About

1.1 Introduction

In recent past, Deep Learning models have proven their potential in many application areas, however its entry into embedded world has its own twists and practical difficulties.

1.2 Problem Statement

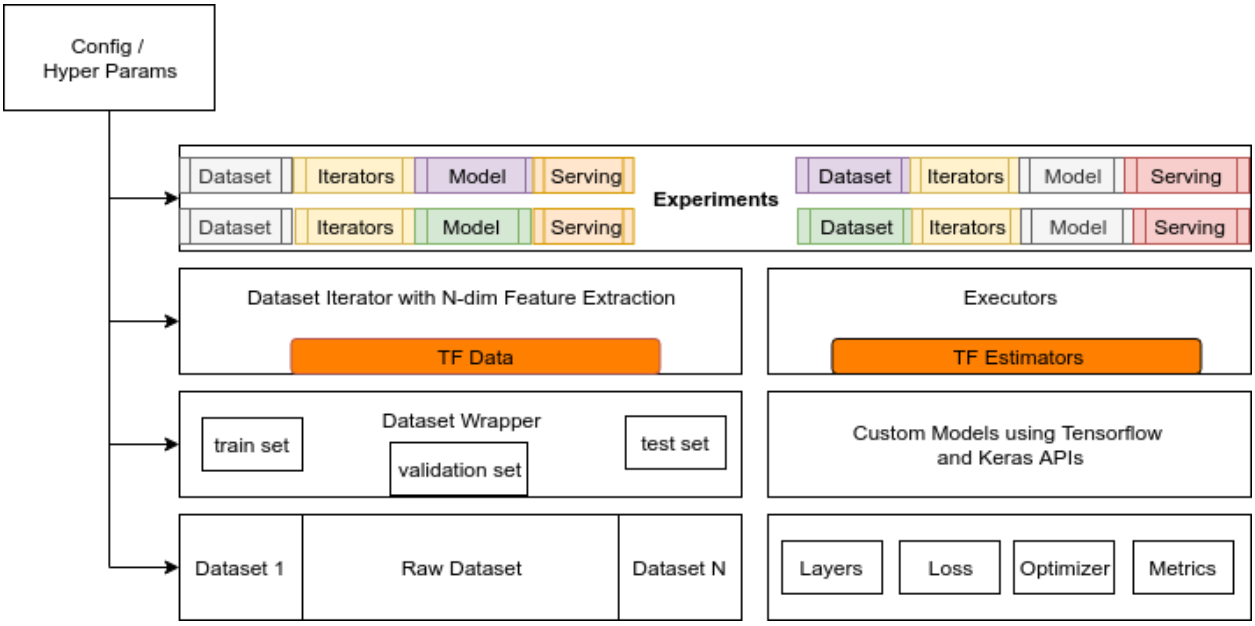
To come up with a framework that enables a fast prototyping of Deep Learning models for Audio (to start with!) and provides an easy way to port the models on to Android using TFLite.

1.3 Proposed Solution

Come up with following modular components which can be then used as plug and play components:

- Dataset modules with preprocessing modules
- DataIterator modules
- Tensorflow Models (Estimators)
- Engine to run the models
- Tensorflow model serving using TFLite
  - Web app
  - Mobile

1.4 Architecture



1.5 Dataset

- FreeSound from Kaggle
- Speech Recognition
  - Kaggle
  - Google

1.6 Notebooks

1.7 Python Environment

```
conda create -n shabda python=3.6
source activate shabda
pip install -r requirements.txt
```

1.8 Examples

CHAPTER 2

Audio Basics

What Does the Unit kHz Mean in Digital Music?

`kHz` is short for kilohertz, and is a measurement of frequency (cycles per second). In digital audio, this measurement describes the number of data chunks used per second to represent an analog sound in digital form. These data chunks are known as the sampling rate or sampling frequency.

This definition is often confused with another popular term in digital audio, called `bitrate (measured in kbps)`. However, the difference between these two terms is that bitrate measures how much is sampled every second (size of the chunks) rather than the number of chunks (frequency).

Note: kHz is sometimes referred to as sampling rate, sampling interval, or cycles per second.

What is the Mel scale?

The Mel scale relates perceived frequency, or pitch, of a pure tone to its actual measured frequency. Humans are much better at discerning small changes in pitch at low frequencies than they are at high frequencies. Incorporating this scale makes our features match more closely what humans hear.

Audio Features

- We start with a speech signal, we'll assume sampled at 16kHz.
- Frame the signal into 20-40 ms frames. 25ms is standard.
  - This means the frame length for a 16kHz signal is 0.025*16000 = 400 samples.
  - Frame step is usually something like 10ms (160 samples), which allows some overlap to the frames.
  - The first 400 sample frame starts at sample 0, the next 400 sample frame starts at sample 160 etc. until the end of the speech file is reached.
  - If the speech file does not divide into an even number of frames, pad it with zeros so that it does.
- Audio Signal File : 0 to N seconds
- Audio Frame : Interval of 20 - 40 ms —> default 25 ms —> 0.025 * 16000 = 400 samples
- Frame step : Default 10 ms —> 0.010 * 16000 —> 160 samples
  - First sample: 0 to 400 samples

- Second sample: 160 to 560 samples etc.,

```
25ms    25ms   25ms   25ms ...  Frames
400     400    400    400  ...  Samples/Frame

|----------|----------|----------|----------|----------|----------|----------|---------
↪|

|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|

10 10 10 10 ... Frame Step
```

Bit-depth = 16: The amplitude of each sample in the audio is one of 2^16 (=65536) possible values. **Samplig rate = 44.1 kHz:** Each second in the audio consists of 44100 samples. So, if the duration of the audio file is 3.2 seconds, the audio will consist of 44100*3.2 = 141120 values.

Still dont get it? Consider the audio signal to be a time series sampled at an interval of 25ms with step size of 10ms

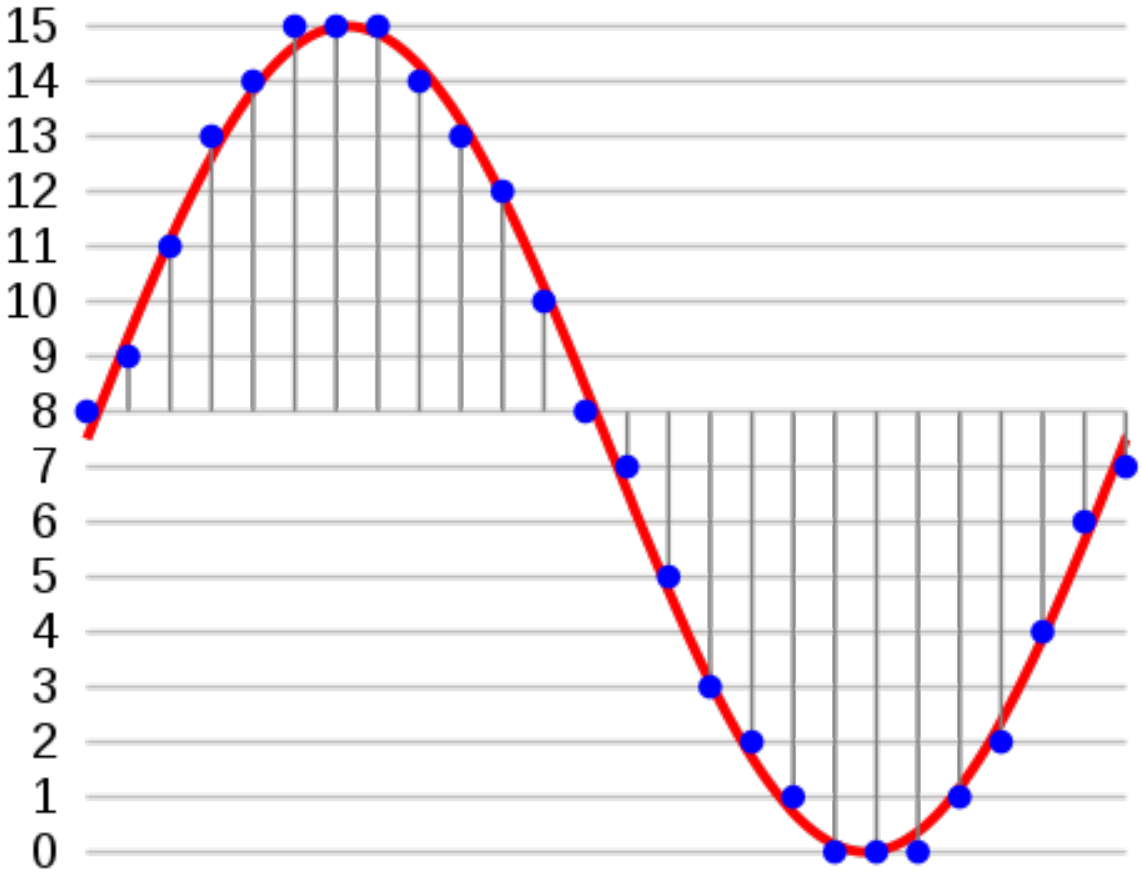
Check out this jupyet notebook @ https://timsainb.github.io/spectrograms-mfccs-and-inversion-in-python.html

Forked version @ https://github.com/dhiraa/python_spectrograms_and_inversion

Mel Frequency Cepstral Coefficient (MFCC) tutorial

2.1 Reading Audio Files

The audios are Pulse-code modulated with a bit depth of 16 and a sampling rate of 44.1 kHz



- **Bit-depth = 16**: The amplitude of each sample in the audio is one of 2^16 (=65536) possible values.
- **Samplig rate = 44.1 kHz**: Each second in the audio consists of 44100 samples. So, if the duration of the audio file is 3.2 seconds, the audio will consist of 44100*3.2 = 141120 values.

2.2 Audio Features

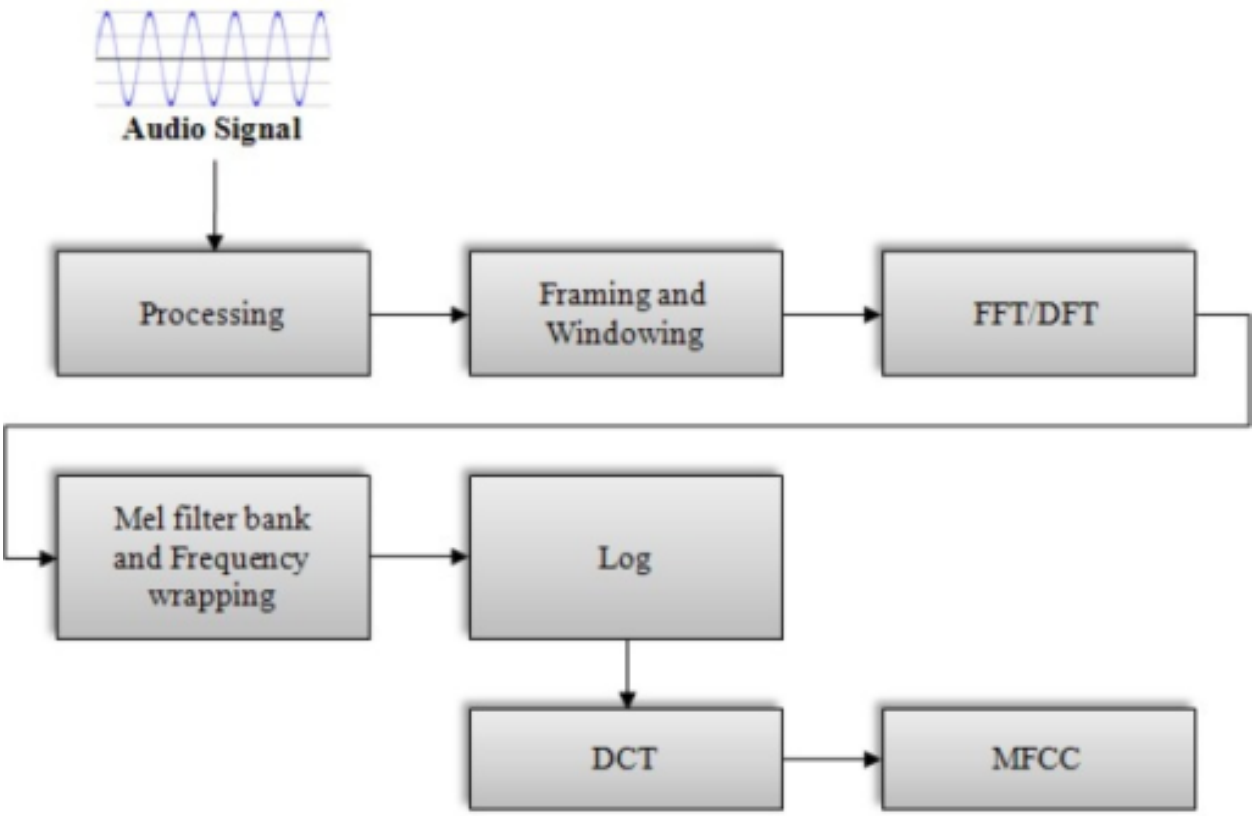
- Zero Cross Rate
- Energy
- Entropy of Energy
- Spectral Centroid
- Spectral Spread
- Spectral Entropy
- Spectral Flux
- Spectral Roll off
- MFCC
- Chroma Vector
- Chroma Deviation

2.3 Introduction to MFCC

Before the Deep Learning era, people developed techniques to extract features from audio signals. It turns out that these techniques are still useful. One such technique is computing the MFCC (Mel Frquency Cepstral Coefficients) from the raw audio. Before we jump to MFCC, let's talk about extracting features from the sound.

If we just want to classify some sound, we should build features that are **speaker independent**. Any feature that only gives information about the speaker (like the pitch of their voice) will not be helpful for classification. In other words, we should extract features that depend on the "content" of the audio rather than the nature of the speaker. Also, a good feature extraction technique should mimic the human speech perception. We don't hear loudness on a linear scale. If we want to double the perceived loudness of a sound, we have to put 8 times as much energy into it. Instead of a linear scale, our perception system uses a log scale.

Taking these things into account, Davis and Mermelstein came up with MFCC in the 1980's. MFCC mimics the logarithmic perception of loudness and pitch of human auditory system and tries to eliminate speaker dependent characteristics by excluding the fundamental frequency and their harmonics. The underlying mathematics is quite complicated and we will skip that. For those interested, here is the detailed explanation.



2.4 FFT/STFT Cheat Sheet:

- **FFT:** Fast Fourier transformA method for computing the discrete Fourier transform of a signal. Its "fastness" relies on size being a power of 2.
- **STFT: Short-time Fourier transform** A method for analyzing a signal whose frequency content is changing over time. The signal is broken into small, often overlapping frames, and the FFT is computed for each frame

(i.e., the frequency content is assumed not to change within a frame, but subsequent analysis frames can be compared to understand how the frequency content changes over time).

- **IFFT: Inverse Fast Fourier transform** Takes a spectrum buffer (a complex vector) of N bins and transforms it into N audio samples.
- **FFT size:** The number of samples over which the FFT is computed; also the number of "bins" that comprise the analysis output.
- **Bin:** The content of a bin denotes the magnitude (and phase) of the frequency corresponding to the bin number. The N bins of an N-sample FFT evenly (linearly) partition the spectrum from 0Hz to the sample rate. Note that for real signals (including audio), we can discard the latter half of the bins, using only the bins from 0Hz to the Nyquist frequency.
- **Window function:** Before computing the FFT, the signal is multiplied by a window function. The simplest window is a rectangular window, which multiplies everything inside the frame by 1 and everything outside the frame by 0. However, in practice, we choose a smoother window function that is 1 in the middle of the window and tapers to 0 or near-0 at the edges. The choice of window depends on the application.
- **Zero-padding:** It is common practice to use a smaller window size than FFT size, then "zeropad" all the samples that lie in between the edges of the window and the edges of the FFT frame.
- **Hop size:** In STFT, you must decide how frequently to perform FFT computations on the signal. If your FFT size is 512 samples, and you have a hop size of 512 samples, you are sliding the analysis frame along the signal with no overlap, nor any space between analyses. If your hop size is 256 samples, you are using 50% overlap. The hop size can be small (high overlap) if you want to very faithfully recreate the sound using an IFFT, or very large if you're only concerned about the spectrum's or spectral features' values every now and then.

2.5 Videos

- https://youtu.be/1RIA9U5oXro
- https://youtu.be/PjlIKVnKe8I

CHAPTER 3

Related Work

- https://research.googleblog.com/2017/08/launching-speech-commands-dataset.html
- https://www.tensorflow.org/versions/master/tutorials/audio_recognition
- https://github.com/tensorflow/tensorflow/tree/master/tensorflow/examples/speech_commands/

3.1 Android App

- https://github.com/tensorflow/tensorflow/tree/master/tensorflow/examples/android/
- ci.tensorflow.org/view/Nightly/job/nightly-android/lastSuccessfulBuild/artifact/out/tensorflow_demo.apk
- https://github.com/petewarden/open-speech-recording
- https://github.com/tensorflow/tensorflow/tree/master/tensorflow/contrib/android
- https://github.com/tensorflow/tensorflow/tree/master/tensorflow/examples/android

3.1.1 References:

- https://heartbeat.fritz.ai/intro-to-machine-learning-on-android-how-to-convert-a-custom-model-to-tensorflow-lite-e07d2d9d50e3
- https://sourcediving.com/machine-learning-on-mobile-fc34be69df1a
- https://blog.insightdatascience.com/ok-google-how-do-you-run-deep-learning-inference-on-android-using-tensorflow-c39fd00c427b **Java MFCC**
- https://www.simplifiedcoding.net/android-speech-to-text-tutorial/ **Android UI**
- https://towardsdatascience.com/how-to-use-dataset-in-tensorflow-c758ef9e4428

CHAPTER 4

Setup

4.1 Ubuntu Environment Setup

4.1.1 Java

```
sudo apt-get update
sudo apt-get install openjdk-8-jdk -y
sudo apt-get install unzip
#check java version
java -version
```

4.1.2 Gradle

Before downloading the gradle pacakge check the official site for the latest version here

For tutorials check here!

```
mkdir /opt/gradle
wget https://services.gradle.org/distributions/gradle-4.10.2-bin.zip
unzip gradle-4.10.2-bin.zip

or

curl -s https://get.sdkman.io | bash - install sdk
#install gradle 3.5 (or any version 3.0+ or 4.0+)
sdk install gradle 3.5
#check the installed version
gradle -v
#switching between versions
sdk use gradle 4.0
```

Add the binaries to user environment path:

```
vim ~/.bashrc
    # add following to the file
    export PATH=$PATH:/opt/gradle/gradle-4.10.2/bin
source ~/.bashrc
#test the installation
gradle -v
```

4.1.3 Python Environment

```
conda create -n shabda python=3.6
source activate shabda
cd path/to/shabda/
pip install -e .[tensorflow-cpu]
#or
pip install -e .[tensorflow-gpu]
pip install -r requirements.txt
```

4.1.4 Git Configure

https://help.github.com/articles/setting-your-commit-email-address-in-git/

```
git clone https://github.com/dhiraa/shabda

#to push without entering password everytime
git remote rm origin
git remote add origin  https://USERNAME:PASSWORD@github.com/dhiraa/shabda.git

#checkout remote branch
git checkout -b branch_name origin/branch_name
```

4.2 IDE Setup

A decent IDE integration is a good start for any development.

Since our motive of supporting different platforms, naturally we end up in dealing with different programing languages.

- **Python** for Tensorflow
- **Kotlin** for Android
- **Java** for Audio libraries
- **Scala** for Web developemnt or Big data, if any.
- **C++** for any performance requirements

4.2.1 Build Tool

- Gradle has been identifed the build tool for JVM languages. (Android by default uses Gradle)

4.2.2 Python Support

Intellij Module configuration is provided along with this repo @

```
/path/to/shabda/src/main/python/shabda/python.iml

IntelliJ
File -> Project Structure -> Modules
Select Shabda (2nd one)
Go to Dependencies tab -> Select a python environment eg:shabda
```

4.3 Shabda Project Structure

Project has three components

- Python based Model development framework using Tensorflow
- Java/Kotlin framework to imitate Python pre-processing and postprocessing steps
- Android application that uses the model using TFLite and the Jar from above step.

```
|
|- android : Android Applications
|- bin : Any runnable
|- build : Gradle build output
|- data : Open Datasets
|- docs : Documentation
|- gradle : Gradle Wrapper executable JAR & configuration properties
|- intellij :
|- notebooks :
|- shabda : Python source code
|- src : JVM source code
|- .gitignore :
|- .pylint.rc :
|- .travis.yml : Travis CI build script
|- build.gradle : Gradle build script for configuring the current project
|- gradlew : Gradle Wrapper script for Unix-based systems
|- gradle.bat : Gradle Wrapper script for Windows
|- LICENSE
|- README.md
|- readthedocs.yml : Readthedocs build script
|- requirements.txt : Python library requirements
|- settings.gradle : Gradle settings script for configuring the Gradle build
|- setup.py
```

CHAPTER 5

API

5.1 Executor

class shabda.run.**Executor**(*model*, *data_iterator*, *config*, *model_hparams=None*, *train_hooks=None*, *eval_hooks=None*, *session_config=None*)

Bases: object

Class that executes training, evaluation, prediction, export, and other actions of tf.estimator.Estimator.

Args:

model: An instance of a subclass of ModelBase.

data_hparams: A *dict* or an instance of HParams containing the hyperparameters of data. It must contain *train* and/or *eval* fields for relevant processes. For example, for *train_and_evaluate()*, both fields are required.

config: An instance of tf.estimator.RunConfig, used as the config argument of Estimator.

model_hparams (optional): A *dict* or an instance of HParams containing the hyperparameters of the model. If *None*, uses model.hparams. Used as the params argument of Estimator.

train_hooks (optional): Iterable of tf.train.SessionRunHook objects to run during training.

eval_hooks (optional): Iterable of tf.train.SessionRunHook objects to run during evaluation.

session_config (optional): An instance of tf.ConfigProto, used as the config argument of tf session.

Example:

```
TODO
```

See *bin/train.py* for the usage in detail.

evaluate(*steps=None*, *checkpoint_path=None*)

Evaluates the model. See tf.estimator.Estimator.evaluate for more details.

Args:

steps (int, optional): Number of steps for which to evaluate model. If *None*, evaluates until the eval data raises an OutOfRange exception.

checkpoint_path (str, optional): Path of a specific checkpoint to evaluate. If *None*, the the latest checkpoint in `config.model_dir` is used. If there are no checkpoints in `model_dir`, evaluation is run with newly initialized variables instead of restored from checkpoint.

train(*max_steps=None*)

Trains the model. See tf.estimator.Estimator.train for more details.

Args:

max_steps (int, optional): Total number of steps for which to train model. If *None*, train forever or until the train data generates the OutOfRange exception. If OutOfRange occurs in the middle, training stops before `max_steps` steps.

train_and_evaluate(*max_train_steps=None*, *eval_steps=None*)

Trains and evaluates the model. See tf.estimator.train_and_evaluate for more details.

Args:

max_train_steps (int, optional): Total number of steps for which to train model. If *None*, train forever or until the train data generates the OutOfRange exception. If OutOfRange occurs in the middle, training stops before `max_steps` steps.

eval_steps (int, optional): Number of steps for which to evaluate model. If *None*, evaluates until the eval data raises an OutOfRange exception.

5.2 DatasetFactory

5.3 AudioDatasetBase

5.4 FreeSoundAudioDataset

5.5 HParams

class shabda.hyperparams.**HParams**(*hparams*, *default_hparams*, *allow_new_hparam=False*)

Bases: `object`

A class to maintains hyperparameters for configing Shabda modules.

The class has several useful features:

- **Auto-completion of missing values.** Users can specify only a subset of hyperparameters they care about. Other hyperparameters will automatically take the default values. The auto-completion performs **recursively** so that hyperparameters taking *dict* values will also be auto-completed **All Shabda modules** provide a `default_hparams()` containing allowed hyperparameters and their default values. For example

```
## Recursive auto-completion
default_hparams = {"a": 1, "b": {"c": 2, "d": 3}}
hparams = {"b": {"c": 22}}
hparams_ = HParams(hparams, default_hparams)
hparams_.todict() == {"a": 1, "b": {"c": 22, "d": 3}}
    # "a" and "d" are auto-completed
```

(continues on next page)

(continued from previous page)

```
## All Shabda modules have built-in `default_hparams`
hparams = {"dropout_rate": 0.1}
emb = tx.modules.WordEmbedder(hparams=hparams, )
emb.hparams.todict() == {
    "dropout_rate": 0.1,  # provided value
    "dim": 100            # default value

}
```

- **Automatic typecheck.** For most hyperparameters, provided value must have the same or compatible dtype with the default value. HParams does necessary typecheck, and raises Error if improper dtype is provided. Also, hyperparameters not listed in *default_hparams* are not allowed, except for "kwargs" as detailed below.
- **Flexible dtype for specified hyperparameters.** Some hyperparameters may allow different dtypes of values.
  - Hyperparameters named "type" are not typechecked. For example, in `get_rnn_cell()`, hyperparameter *"type"* can take value of an RNNCell class, its string name of module path, or an RNNCell class instance. (String name or module path is allowd so that users can specify the value in YAML config files.)
  - For other hyperparameters, list them in the "@no_typecheck" field in *default_hparams* to skip typecheck. For example, in `get_rnn_cell()`, hyperparameter "_keep_prob" can be set to either a *float* or a *tf.placeholder*.
- **Special flexibility of keyword argument hyparameters.** Hyperparameters named "kwargs" are used as keyword arguments for a class constructor or a function call. Such hyperparameters take a *dict*, and users can add arbitrary valid keyword arguments to the dict. For example:

```
default_rnn_cell_hparams = {
    "type": "BasicLSTMCell",
    "kwargs": { "num_units": 256 }
    # Other hyperparameters

}
my_hparams = {
    "kwargs" {
        "num_units": 123,
        "forget_bias": 0.0         # Other valid keyword arguments
        "activation": "tf.nn.relu" # for BasicLSTMCell constructor
    }
}
_ = HParams(my_hparams, default_rnn_cell_hparams)
```

- **Rich interfaces.** An HParams instance provides rich interfaces for accessing, updating, or adding hyperparameters.

```
hparams = HParams(my_hparams, default_hparams)
# Access
hparams.type == hparams["type"]
# Update
hparams.type = "GRUCell"
hparams.kwargs = { "num_units": 100 }
hparams.kwargs.num_units == 100
# Add new
hparams.add_hparam("index", 1)
```

(continues on next page)

(continued from previous page)

```
hparams.index == 1

# Convert to `dict` (recursively)
type(hparams.todic()) == dict

# I/O
pickle.dump(hparams, "hparams.dump")
with open("hparams.dump", 'rb') as f:
    hparams_loaded = pickle.load(f)
```

Args:

hparams: A *dict* or an *HParams* instance containing hyperparameters. If *None*, all hyperparameters are set to default values.

default_hparams (dict): Hyperparameters with default values. If *None*, Hyperparameters are fully defined by `hparams`.

allow_new_hparam (bool): If *False* (default), `hparams` cannot contain hyperparameters that are not included in `default_hparams`, except for the case of `"kwargs"` as above.

`add_hparam`(*name*, *value*)
Adds a new hyperparameter.

`get`(*name*, *default=None*)
Returns the hyperparameter value for the given name. If name is not available then returns `default`.

Args: name (str): the name of hyperparameter. default: the value to be returned in case name does not exist.

`items`()
Returns the list of hyperparam *(name, value)* pairs

`keys`()
Returns the list of hyperparam names

`todict`()
Returns a copy of hyperparameters as a dictionary.

5.6 ModelBase

5.7 ModelsFactory

5.8 ClassifierBase

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

Python Module Index

S

Index