Kaldi Pybind

Mar 24, 2020

Contents


KALDI

CHAPTER 1

About Kaldi Pybind

Kaldi Pybind is a Python wrapper for Kaldi using Pybind11. It is still under active development.

Everything related to Kaldi Pybind is put in the pybind11 branch.

CHAPTER 2

Getting Started

2.1 Compiling Kaldi Pybind

First, you have to install Kaldi. You can find detailed information for Kaldi installation from http://kaldi-asr.org/doc/install.html.

Note: Kaldi Pybind is still under active development and has not yet been merged into the master branch. You should checkout the `pybind11` branch before compilation.

Note: We support **ONLY** Python3. If you are still using Python2, please upgrade to Python3. Python3.5 is known to work.

The following is a quick start:

```
git clone https://github.com/kaldi-asr/kaldi.git
cd kaldi
git checkout pybind11
cd tools
extras/check_dependencies.sh
make -j4
cd ../src
./configure --shared
make -j4
cd pybind
pip install pybind11
make
make test
```

After a successful compilation, you have to modify the environment variable `PYTHONPATH`:

```
export KALDI_ROOT=/path/to/your/kaldi
export PYTHONPATH=$KALDI_ROOT/src/pybind:$PYTHONPATH
```

Hint: There is no `make install`. Once compiled, you are ready to use Kaldi Pybind.

CHAPTER 3

Working with Kaldi's Matrices

This tutorial demonstrates how to use Kaldi's matrices in Python.

The following table summarizes the matrix types in Kaldi that have been wrapped to Python.

Kaldi Types	Python Types
`Vector<float>`	`FloatVector`
`SubVector<float>`	`FloatSubVector`
`Matrix<float>`	`FloatMatrix`
`SubMatrix<float>`	`FloatSubMatrix`

All of the Python types above can be converted to Numpy arrays without copying the underlying memory buffers. In addition, `FloatSubVector` and `FloatSubMatrix` can be constructed directly from Numpy arrays without data copying.

Note: Only the **single** precision floating point type has been wrapped to Python.

3.1 FloatVector

The following code shows how to use `FloatVector` in Python.

Listing 1: Example usage of `FloatVector`

```
#!/usr/bin/env python3

import kaldi

f = kaldi.FloatVector(3)
f[0] = 10
print(f)
```

(continues on next page)

(continued from previous page)

```

g = f.numpy()
g[1] = 20
print(f)
```

Its output is

```
[ 10 0 0 ]

[ 10 20 0 ]
```

```
#!/usr/bin/env python3
```

This is a hint that it needs `python3`. At present, we support only `python3` in Kaldi Pybind.

```
import kaldi
```

This imports the Kaldi Pbyind package. If you encounter an import error, please make sure that `PYTHONPATH` has been set to point to `KALDI_ROOT/src/pybind`.

```
f = kaldi.FloatVector(3)
```

This creates an object of `FloatVector` containing 3 elements which are by default initialized to zero.

```
print(f)
```

This prints the value of the `FloatVector` object to the console. Note that you use operator `()` in C++ to access the elements of a `Vector<float>` object; Python code uses `[]`.

```
g = f.numpy()
```

This creates a Numpy array object `g` from `f`. No memory is copied here. `g` shares the underlying memory with `f`.

```
g[1] = 20
```

This also changes `f` since it shares the same memory with `g`. You can verify that `f` is changed from the output.

Hint: We recommend that you invoke the `numpy()` method of a `FloatVector` object to get a Numpy ndarray object and to manipulate this Numpy object. Since it shares the underlying memory with the `FloatVector` object, every operation you perform to this Numpy ndarray object is visible to the `FloatVector` object.

3.2 FloatSubVector

The following code shows how to use `FloatSubVector` in Python.

Listing 2: Example usage of `FloatSubVector`

```
#!/usr/bin/env python3

import kaldi
import numpy as np
```

(continues on next page)

(continued from previous page)

```
5
6  v = np.array([10, 20, 30], dtype=np.float32)
7  f = kaldi.FloatSubVector(v)
8
9  f[0] = 0
10 print(v)
11
12 g = f.numpy()
13 g[1] = 100
14 print(v)
```

Its output is

```
[ 0. 20. 30.]
[  0. 100.  30. ]
```

```
6  v = np.array([10, 20, 30], dtype=np.float32)
7  f = kaldi.FloatSubVector(v)
```

This creates a `FloatSubVector` object `f` from a Numpy ndarray object `v`. No memory is copied here. `f` shares the underlying memory with `v`. Note that the `dtype` of `v` has to be `np.float32`; otherwise, you will get a runtime error when creating `f`.

```
9  f[0] = 0
```

This uses `[]` to access the elements of `f`. It also changes `v` since `f` shares the same memory with `v`.

```
12 g = f.numpy()
```

This create a Numpy ndarray object `g` from `f`. No memory is copied here. `g` shares the same memory with `f`.

```
13 g[1] = 100
```

This also changes `v` because of memory sharing.

3.3 FloatMatrix

The following code shows how to use `FloatMatrix` in Python.

Listing 3: Example usage of `FloatMatrix`

```
1  #!/usr/bin/env python3
2
3  import kaldi
4
5  f = kaldi.FloatMatrix(2, 3)
6  f[1, 2] = 100
7  print(f)
8
9  g = f.numpy()
10 g[0, 0] = 200
11 print(f)
```

Its output is

```
[
 0 0 0
 0 0 100 ]

[
 200 0 0
 0 0 100 ]
```

```
f = kaldi.FloatMatrix(2, 3)
```

This creates an object `f` of `FloatMatrix` with 2 rows and 3 columns.

```
f[1, 2] = 100
```

This uses `[]` to access the elements of `f`.

```
print(f)
```

This prints the value of `f` to the console.

```
g = f.numpy()
```

This creates a Numpy ndarray object `g` from `f`. No memory is copied here. `g` shares the underlying memory with `f`.

```
g[0, 0] = 200
```

This also changes `f` due to memory sharing.

3.4 FloatSubMatrix

The following code shows how to use `FloatSubMatrix` in Python.

Listing 4: Example usage of `FloatSubMatrix`

```
#!/usr/bin/env python3

import kaldi
import numpy as np

m = np.array([[1, 2, 3], [10, 20, 30]], dtype=np.float32)
f = kaldi.FloatSubMatrix(m)

f[1, 2] = 100
print(m)
print()

g = f.numpy()
g[0, 0] = 200
print(m)
```

Its output is

```
[[  1.   2.   3.]
 [ 10.  20. 100.]]
```

(continues on next page)

(continued from previous page)

```
[[200.   2.   3.]
 [ 10.  20. 100.]]
```

```
m = np.array([[1, 2, 3], [10, 20, 30]], dtype=np.float32)
f = kaldi.FloatSubMatrix(m)
```

This creates an object `f` of `FloatSubMatrix` from a Numpy ndarray object `m`. `f` shares the underlying memory with `m`. Note that the `dtype` of `m` has to be `np.float32`. Otherwise you will get a runtime error.

```
f[1, 2] = 100
print(m)
```

This uses `[]` to access the elements of `f`. Note that `m` is also changed due to memory sharing.

```
g = f.numpy()
```

This creates a Numpy ndarray object from `f`. No memory is copied here. `g` shares the underlying memory with `f`.

```
g[0, 0] = 200
```

This changes `f` and `m`.

CHAPTER 4

Working with Kaldi's IO

This tutorial shows how to read and write ark/scp files in Python.

4.1 Reading and Writing Alignment Information

The following class can be used to write alignment information to files:

- `IntVectorWriter`

And the following classes can be used to read alignment information from files:

- `SequentialIntVectorReader`
- `RandomAccessIntVectorReader`

The following code shows how to write and read alignment information.

Listing 1: Example of reading and writing align information

```
#!/usr/bin/env python3

import kaldi

wspecifier = 'ark,scp:/tmp/ali.ark,/tmp/ali.scp'

writer = kaldi.IntVectorWriter(wspecifier)
writer.Write(key='foo', value=[1, 2, 3])
writer.Write('bar', [10, 20])
writer.Close()

rspecifier = 'scp:/tmp/ali.scp'
reader = kaldi.SequentialIntVectorReader(rspecifier)

for key, value in reader:
    print(key, value)
```

(continues on next page)

(continued from previous page)

```

reader.Close()

reader = kaldi.RandomAccessIntVectorReader(rspecifier)
value1 = reader['foo']
print(value1)

value2 = reader['bar']
print(value2)
reader.Close()
```

Its output is

```
foo [1, 2, 3]
bar [10, 20]
[1, 2, 3]
[10, 20]
```

The output of the following command

```
$ copy-int-vector scp:/tmp/ali.scp ark,t:-
```

is

```
copy-int-vector scp:/tmp/ali.scp ark,t:-
foo 1 2 3
bar 10 20
LOG (copy-int-vector[5.5.792~1-f5875b]:main():copy-int-vector.cc:83) Copied 2 vectors␣
↪of int32.
```

```
wspecifier = 'ark,scp:/tmp/ali.ark,/tmp/ali.scp'
```

It creates a write specifier `wspecifier` indicating that the alignment information is going to be written into files `/tmp/ali.ark` and `/tmp/ali.scp`.

```
writer.Write(key='foo', value=[1, 2, 3])
```

It writes a list `[1, 2, 3]` to file with `key == foo`. Note that you can use keyword arguments while writing.

```
writer.Write('bar', [10, 20])
```

It writes a list `[10, 20]` to file with `key == bar`.

```
writer.Close()
```

It closes the writer.

Note: It is a best practice to close the file when it is no longer needed.

```
rspecifier = 'scp:/tmp/ali.scp'
reader = kaldi.SequentialIntVectorReader(rspecifier)
```

It creates a **sequential** reader.

```
for key, value in reader:
    print(key, value)
```

It uses a `for` loop to iterate the reader.

```
reader.Close()
```

It closes the reader.

```
reader = kaldi.RandomAccessIntVectorReader(rspecifier)
```

It creates a **random access** reader.

```
value1 = reader['foo']
print(value1)
```

It reads the value of `foo` and prints it out.

```
value2 = reader['bar']
print(value2)
```

It reads the value of `bar` and prints it out.

```
reader.Close()
```

Finally, it closes the reader.

The following code example achieves the same effect as the above one except that you do not need to close the file manually.

Listing 2: Example of reading and writing align information using `with`

```
#!/usr/bin/env python3

import kaldi

wspecifier = 'ark,scp:/tmp/ali.ark,/tmp/ali.scp'

with kaldi.IntVectorWriter(wspecifier) as writer:
    writer.Write(key='foo', value=[1, 2, 3])
    writer.Write('bar', [10, 20])

# Note that you do NOT need to close the file.

rspecifier = 'scp:/tmp/ali.scp'
with kaldi.SequentialIntVectorReader(rspecifier) as reader:
    for key, value in reader:
        print(key, value)

rspecifier = 'scp:/tmp/ali.scp'
with kaldi.RandomAccessIntVectorReader(rspecifier) as reader:
    value1 = reader['foo']
    print(value1)

    value2 = reader['bar']
    print(value2)
```

4.2 Reading and Writing Matrices

4.2.1 Using xfilename

The following code demonstrates how to read and write `FloatMatrix` using `xfilename`.

Listing 3: Example of reading and writing matrices with xfilename

```
#!/usr/bin/env python3

import kaldi

m = kaldi.FloatMatrix(2, 2)
m[0, 0] = 10
m[1, 1] = 20

xfilename = '/tmp/lda.mat'
kaldi.write_mat(m, xfilename, binary=True)

g = kaldi.read_mat(xfilename)
print(g)
```

The output of the above program is

```
[
 10 0
 0 20 ]
```

```
m = kaldi.FloatMatrix(2, 2)
m[0, 0] = 10
m[1, 1] = 20
```

It creates a `FloatMatrix` and sets its diagonal to `[10, 20]`.

```
xfilename = '/tmp/lda.mat'
kaldi.write_mat(m, xfilename, binary=True)
```

It writes the matrix to `/tmp/lda.mat` in binary format. `kaldi.write_mat` is used to write the matrix to the specified file. You can specify whether it is written in binary format or text format.

```
g = kaldi.read_mat(xfilename)
print(g)
```

It reads the matrix back and prints it to the console. Note that you do not need to specify whether the file to read is in binary or not. `kaldi.read_mat` will figure out the format automatically.

4.2.2 Using specifier

The following code demonstrates how to read and write `FloatMatrix` using `specifier`.

Listing 4: Example of reading and writing matrices with specifier

```
#!/usr/bin/env python3

import numpy as np
```

(continues on next page)

(continued from previous page)

```
import kaldi

wspecifier = 'ark,scp:/tmp/feats.ark,/tmp/feats.scp'

writer = kaldi.MatrixWriter(wspecifier)

m = np.arange(6).reshape(2, 3).astype(np.float32)
writer.Write(key='foo', value=m)

g = kaldi.FloatMatrix(2, 2)
g[0, 0] = 10
g[1, 1] = 20
writer.Write('bar', g)

writer.Close()

rspecifier = 'scp:/tmp/feats.scp'
reader = kaldi.SequentialMatrixReader(rspecifier)
for key, value in reader:
    assert key in ['foo', 'bar']
    if key == 'foo':
        np.testing.assert_array_equal(value.numpy(), m)
    else:
        np.testing.assert_array_equal(value.numpy(), g.numpy())

reader.Close()

reader = kaldi.RandomAccessMatrixReader(rspecifier)
assert 'foo' in reader
assert 'bar' in reader
np.testing.assert_array_equal(reader['foo'].numpy(), m)
np.testing.assert_array_equal(reader['bar'].numpy(), g.numpy())
reader.Close()
```

```
wspecifier = 'ark,scp:/tmp/feats.ark,/tmp/feats.scp'

writer = kaldi.MatrixWriter(wspecifier)
```

This creates a matrix writer.

```
m = np.arange(6).reshape(2, 3).astype(np.float32)
writer.Write(key='foo', value=m)
```

It creates a Numpy array object of type `np.float32` and writes it to file with the key `foo`. Note that the type of the Numpy array has to be of type `np.float32`. The program throws if the type is not `np.float32`.

```
g = kaldi.FloatMatrix(2, 2)
g[0, 0] = 10
g[1, 1] = 20
writer.Write('bar', g)
```

It creates a `FloatMatrix` and writes it to file with the key `bar`.

Hint: `kaldi.MatrixWriter` accepts Numpy array objects of type `np.float32` as well as `kaldi.FloatMatrix` objects.

```
writer.Close()
```

It closes the writer.

```
rspecifier = 'scp:/tmp/feats.scp'
reader = kaldi.SequentialMatrixReader(rspecifier)
```

It creates a **sequential** matrix reader.

```
reader = kaldi.SequentialMatrixReader(rspecifier)
for key, value in reader:
    assert key in ['foo', 'bar']
    if key == 'foo':
        np.testing.assert_array_equal(value.numpy(), m)
    else:
        np.testing.assert_array_equal(value.numpy(), g.numpy())
```

It uses `for` loop to iterate the sequential reader.

```
reader.Close()
```

It closes the sequential reader.

```
reader = kaldi.RandomAccessMatrixReader(rspecifier)
```

It creates a **random access** matrix reader.

```
assert 'foo' in reader
assert 'bar' in reader
```

It uses `in` to test whether the reader contains a given key.

```
np.testing.assert_array_equal(reader['foo'].numpy(), m)
np.testing.assert_array_equal(reader['bar'].numpy(), g.numpy())
```

It uses `[]` to read the value of a specified key.

```
reader.Close()
```

It closes the random access reader.

The following code example achieves the same effect as the above one except that you do not need to close the file manually.

Listing 5: Example of reading and writing FloatMatrix using `with`

```
#!/usr/bin/env python3

import numpy as np
import kaldi

wspecifier = 'ark,scp:/tmp/feats.ark,/tmp/feats.scp'

with kaldi.MatrixWriter(wspecifier) as writer:
    m = np.arange(6).reshape(2, 3).astype(np.float32)
    writer.Write(key='foo', value=m)
```

(continues on next page)

(continued from previous page)

```
    g = kaldi.FloatMatrix(2, 2)
    g[0, 0] = 10
    g[1, 1] = 20
    writer.Write('bar', g)

rspecifier = 'scp:/tmp/feats.scp'
with kaldi.SequentialMatrixReader(rspecifier) as reader:
    for key, value in reader:
        assert key in ['foo', 'bar']
        if key == 'foo':
            np.testing.assert_array_equal(value.numpy(), m)
        else:
            np.testing.assert_array_equal(value.numpy(), g.numpy())

with kaldi.RandomAccessMatrixReader(rspecifier) as reader:
    assert 'foo' in reader
    assert 'bar' in reader
    np.testing.assert_array_equal(reader['foo'].numpy(), m)
    np.testing.assert_array_equal(reader['bar'].numpy(), g.numpy())
```