# test_netcdf_stream

July 14, 2022

[2]:
```
import os
import shutil
import glob
import time
from tqdm import tqdm
import random
import numpy as np
import xarray as xr
import pandas as pd
import tensorflow as tf
```

## 1 Build up efficient TF data pipelines from netCDF-files

We test two different approaches to build up the dataset input streams. The first one is based on `open_dataset` and requires a large buffer size to enable proper sampling (buffer size $\mathcal{O}(10^4)$ so that at minimum 10 files are buffered). This is due to the fact that only the data files are randomized, not the data samples itself as in the next approach. The second approach is based on `open_mfdatset` which makes the data sampling much easier since it allows randomization on an index-list for the time dimension to build up the iterator.

[3]:
```
def load_nc_dir_with_generator(dir_, patt, shuffle=True, seed=42):
    """
    Opens datafiles via looping in the generator-method. This implies a larger␣
 ↪buffer_size (> 12*744 where 12 corresponds to months per year
    and 744 is the number of time steps in a monthly datafile) when shuffling␣
 ↪since the buffer gets filled up with data from sequential (unordered) files.
    :param dir_: The directory where the netCDF-files are located.
    :param patt: Substring-pattern to identify the desired netCDF-files␣
 ↪("{patt}*.nc" is applied for searching)
    :param shuffle: flag to enable shuffling
    :param seed: seed for random shuffling
    :return: tf.Dataset for data streain in neural networks
    """

    nc_files = glob.glob(os.path.join(dir_, f"{patt}*.nc"))

    if shuffle:
```

```
        random.seed(seed)
        random.shuffle(nc_files)

    def gen(nc_files, shuffle=True, seed=42):

        for file in nc_files:
            ds = xr.open_dataset(file, engine='netcdf4')
            ntimes = len(ds["time"])
            for t in range(ntimes):
                ds_t = ds.isel({"time": t})
                data_dict = {key: tf.convert_to_tensor(val) for key, val in␣
 ↪ds_t.items()}
                data_dict["time"]= np.array([pd.to_datetime(ds_t["time"].
 ↪values).strftime("%Y-%m-%d %H:%M")])
                yield data_dict


    sample = next(iter(gen(nc_files, shuffle, seed=seed)))

    gen_mod = gen(nc_files, shuffle, seed)

    return tf.data.Dataset.from_generator(
        lambda: gen_mod,
        output_signature={
            key: tf.TensorSpec(val.shape, dtype=val.dtype)
            for key, val in sample.items()
        }
    )

def load_mfnc_dir_with_generator(dir_: str, patt: str, shuffle: bool = True,␣
 ↪seed: int = 42):
    """
    Opens netCDF-files using xarray's open_mfdataset-method. Shuffling of the␣
 ↪data is achieved by shuffling over the time step-indices.
    For efficiency, decoding of the time is disabled (implying shared␣
 ↪time-units for all netCDF-data to avoid overwriting with open_mfdataset!!!)
    since this information is not required for data streaming.
    :param dir_: The directory where the netCDF-files are located.
    :param patt: Substring-pattern to identify the desired netCDF-files␣
 ↪("{patt}*.nc" is applied for searching)
    :param shuffle: flag to enable shuffling
    :param seed: seed for random shuffling
    :return: tf.Dataset for data streain in neural networks
    """
    ds_all = xr.open_mfdataset(os.path.join(dir_, f"{patt}*.nc"), cache=False,␣
 ↪decode_cf=False)
```

```
    ntimes = len(ds_all["time"])
    if shuffle:
        random.seed(seed)
        time_list = random.sample(range(ntimes), ntimes)
    else:
        time_list = range(ntimes)


    def gen(ds_all):
        #ds_all = xr.open_mfdataset(os.path.join(dir_, f"{patt}*.nc"),␣
 ↪cache=False, decode_cf=False)#, parallel=True)#, decode_times=False)
        for t in time_list:
            # ds = xr.decode_cf(ds_all.isel({"time": t}))
            ds = ds_all.isel({"time": t})
            data_dict = {key: tf.convert_to_tensor(val) for key, val in ds.
 ↪items()}
            # data_dict["time"]= np.array([pd.to_datetime(ds["time"].values).
 ↪strftime("%Y-%m-%d %H:%M")])
            yield data_dict


    sample = next(iter(gen(ds_all)))

    gen_mod = gen(ds_all)

    return tf.data.Dataset.from_generator(
        lambda: gen_mod,
        output_signature={
            key: tf.TensorSpec(val.shape, dtype=val.dtype)
            for key, val in sample.items()
        }
    )


### highly in efficient in terms of memory -> not tested subsequently!!!
def load_data(dir_, patt) -> xr.DataArray:
    """
    Obtain the data and meta information from the netcdf files, including the␣
 ↪len of the samples, mim and max values
    return: data as xarray's DataArray with dimensions [channels, time, lat,␣
 ↪lon]
    """

    def reshape_ds(ds):
        da = ds.to_array(dim="variables")
        da = da.transpose(..., "variables")
        return da
```

```
    ds = xr.open_mfdataset(os.path.join(dir_, f"{patt}*.nc"), cache=False,␣
 ↪parallel=True)
    da = reshape_ds(ds)
    init_times = da["time"]

    nvars = len(da["variables"])

    return da, init_times #da.chunk(chunks={"time": 744, "variables": nvars}),␣
 ↪init_times
```

After setting up the data directory, both strategies are benchmarked.

[4]:
```
datadir = "/p/scratch/deepacf/maelstrom/maelstrom_data/ap5_michael/
 ↪preprocessed_era5_ifs/netcdf_data/all_files/"
pattern = "preproc_"
```

We create the respective TF datasets, ...

[5]:
```
tfds_test = load_nc_dir_with_generator(datadir, pattern)
tfds_test_mf = load_mfnc_dir_with_generator(datadir, pattern)
tfds_test_mf2 = load_mfnc_dir_with_generator(datadir, pattern)
```

```
2022-07-14 13:30:41.079229: I tensorflow/core/platform/cpu_feature_guard.cc:142]
This TensorFlow binary is optimized with oneAPI Deep Neural Network Library
(oneDNN) to use the following CPU instructions in performance-critical
operations:  AVX512F
To enable them in other operations, rebuild TensorFlow with the appropriate
compiler flags.
2022-07-14 13:30:41.607481: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1510] Created device
/job:localhost/replica:0/task:0/device:GPU:0 with 30194 MB memory:  -> device:
0, name: Tesla V100-PCIE-32GB, pci bus id: 0000:5e:00.0, compute capability: 7.0
```

... configure them and...

[6]:
```
sleep_sec = 0
ap1 = iter(tfds_test.shuffle(buffer_size=20000).batch(32).prefetch(100).
 ↪repeat(1))
ap2 = iter(tfds_test_mf.batch(32).prefetch(100).repeat(1))
ap3 = iter(tfds_test_mf2.shuffle(buffer_size=1000).batch(32).prefetch(100).
 ↪repeat(1))
```

```
2022-07-14 13:31:22.314399: I
tensorflow/compiler/mlir/mlir_graph_optimization_pass.cc:185] None of the MLIR
Optimization Passes are enabled (registered 2)
```

then run both approaches. We start with approach 1:

[7]:
```
%%time

sleep_sec = 0
ntakes = 1000

for i in tqdm(range(ntakes)):
    if i == 1:
        time_s = time.time()

    batch = ap1.get_next()
    #print(batch["2t_in"])
    #print("***************")

load_time_ap1 = (time.time() - time_s)/float(ntakes-1)
print("After filling the buffer, retrieving each minibatch took: {0:5.04f}s".
 ↪format(load_time_ap1))
```

```
  0%|          | 0/1000 [00:00<?, ?it/s]2022-07-14 13:31:32.390863: I
tensorflow/core/kernels/data/shuffle_dataset_op.cc:175] Filling up shuffle
buffer (this may take a while): 1176 of 20000
2022-07-14 13:31:42.393018: I
tensorflow/core/kernels/data/shuffle_dataset_op.cc:175] Filling up shuffle
buffer (this may take a while): 2364 of 20000
2022-07-14 13:31:52.389588: I
tensorflow/core/kernels/data/shuffle_dataset_op.cc:175] Filling up shuffle
buffer (this may take a while): 3568 of 20000
2022-07-14 13:32:02.390630: I
tensorflow/core/kernels/data/shuffle_dataset_op.cc:175] Filling up shuffle
buffer (this may take a while): 4759 of 20000
2022-07-14 13:32:12.393867: I
tensorflow/core/kernels/data/shuffle_dataset_op.cc:175] Filling up shuffle
buffer (this may take a while): 5950 of 20000
2022-07-14 13:32:22.391581: I
tensorflow/core/kernels/data/shuffle_dataset_op.cc:175] Filling up shuffle
buffer (this may take a while): 7137 of 20000
2022-07-14 13:32:32.392505: I
tensorflow/core/kernels/data/shuffle_dataset_op.cc:175] Filling up shuffle
buffer (this may take a while): 8338 of 20000
2022-07-14 13:32:42.394676: I
tensorflow/core/kernels/data/shuffle_dataset_op.cc:175] Filling up shuffle
buffer (this may take a while): 9564 of 20000
2022-07-14 13:32:52.389783: I
tensorflow/core/kernels/data/shuffle_dataset_op.cc:175] Filling up shuffle
buffer (this may take a while): 10800 of 20000
2022-07-14 13:33:02.401542: I
tensorflow/core/kernels/data/shuffle_dataset_op.cc:175] Filling up shuffle
buffer (this may take a while): 12002 of 20000
2022-07-14 13:33:12.394499: I
```

```
tensorflow/core/kernels/data/shuffle_dataset_op.cc:175] Filling up shuffle
buffer (this may take a while): 13200 of 20000
2022-07-14 13:33:22.396244: I
tensorflow/core/kernels/data/shuffle_dataset_op.cc:175] Filling up shuffle
buffer (this may take a while): 14405 of 20000
2022-07-14 13:33:32.394863: I
tensorflow/core/kernels/data/shuffle_dataset_op.cc:175] Filling up shuffle
buffer (this may take a while): 15560 of 20000
2022-07-14 13:33:42.395892: I
tensorflow/core/kernels/data/shuffle_dataset_op.cc:175] Filling up shuffle
buffer (this may take a while): 16762 of 20000
2022-07-14 13:33:52.394951: I
tensorflow/core/kernels/data/shuffle_dataset_op.cc:175] Filling up shuffle
buffer (this may take a while): 17951 of 20000
2022-07-14 13:34:02.403250: I
tensorflow/core/kernels/data/shuffle_dataset_op.cc:175] Filling up shuffle
buffer (this may take a while): 19141 of 20000
2022-07-14 13:34:09.537888: I
tensorflow/core/kernels/data/shuffle_dataset_op.cc:228] Shuffle buffer filled.
100%|      | 1000/1000 [06:30<00:00,  2.56it/s]

After filling the buffer, retrieving each minibatch took: 0.2235s
CPU times: user 5min 41s, sys: 1min, total: 6min 42s
Wall time: 6min 30s
```

Now, we continue with approach 2:

[8]:
```
%%time

time_s = time.time()

for i in tqdm(range(ntakes)):
    if i == 0:
        print(f"Sleeping for {sleep_sec}s...")
        time.sleep(sleep_sec)

    batch = ap2.get_next()
    #print(batch["2t_in"])
    #print("***************")

load_time_ap2 = (time.time() - time_s)/float(ntakes)
```

```
  0%|          | 0/1000 [00:00<?, ?it/s]

Sleeping for 0s…

100%|      | 1000/1000 [25:34<00:00,  1.53s/it]

CPU times: user 18min 49s, sys: 2min 38s, total: 21min 27s
```

```
Wall time: 25min 34s
```

[9]:
```
%%time

sleep_sec = 0

for i in tqdm(range(ntakes)):
    if i == 1:
        time_s = time.time()

    batch = ap3.get_next()
    #print(batch["2t_in"])
    #print("***************")

load_time_ap3 = (time.time() - time_s)/float(ntakes-1)
print("After filling the buffer, retrieving each minibatch took: {0:5.04f}s".
 ↪format(load_time_ap3))
```

```
  0%|          | 0/1000 [00:00<?, ?it/s]2022-07-14 14:03:37.568508: I
tensorflow/core/kernels/data/shuffle_dataset_op.cc:175] Filling up shuffle
buffer (this may take a while): 110 of 1000
2022-07-14 14:03:47.533671: I
tensorflow/core/kernels/data/shuffle_dataset_op.cc:175] Filling up shuffle
buffer (this may take a while): 218 of 1000
2022-07-14 14:03:57.521254: I
tensorflow/core/kernels/data/shuffle_dataset_op.cc:175] Filling up shuffle
buffer (this may take a while): 325 of 1000
2022-07-14 14:04:07.606089: I
tensorflow/core/kernels/data/shuffle_dataset_op.cc:175] Filling up shuffle
buffer (this may take a while): 434 of 1000
2022-07-14 14:04:17.535844: I
tensorflow/core/kernels/data/shuffle_dataset_op.cc:175] Filling up shuffle
buffer (this may take a while): 544 of 1000
2022-07-14 14:04:27.577850: I
tensorflow/core/kernels/data/shuffle_dataset_op.cc:175] Filling up shuffle
buffer (this may take a while): 651 of 1000
2022-07-14 14:04:37.580522: I
tensorflow/core/kernels/data/shuffle_dataset_op.cc:175] Filling up shuffle
buffer (this may take a while): 763 of 1000
2022-07-14 14:04:47.548819: I
tensorflow/core/kernels/data/shuffle_dataset_op.cc:175] Filling up shuffle
buffer (this may take a while): 873 of 1000
2022-07-14 14:04:57.547731: I
tensorflow/core/kernels/data/shuffle_dataset_op.cc:175] Filling up shuffle
buffer (this may take a while): 987 of 1000
2022-07-14 14:04:58.724628: I
tensorflow/core/kernels/data/shuffle_dataset_op.cc:228] Shuffle buffer filled.
```

```
100%|      | 1000/1000 [28:15<00:00,  1.70s/it]

After filling the buffer, retrieving each minibatch took: 1.6026s
CPU times: user 21min 44s, sys: 3min 7s, total: 24min 51s
Wall time: 28min 15s
```

### 1.0.1 Results

[10]:
```
print("After filling the buffer, retrieving each minibatch with APPROACH 1 took:
 ↪ {0:5.04f}s".format(load_time_ap1))
print("Retrieving each minibatch with APPROACH 2 took: {0:5.04f}s".
 ↪format(load_time_ap2))
print("After filling the buffer, retrieving each minibatch with APPROACH 3 took:
 ↪ {0:5.04f}s".format(load_time_ap3))
```

```
After filling the buffer, retrieving each minibatch with APPROACH 1 took:
0.2235s
Retrieving each minibatch with APPROACH 2 took: 1.5344s
After filling the buffer, retrieving each minibatch with APPROACH 3 took:
1.6026s
```

Thus, we see that the first approach outperforms the second approach (at least after the buffer has been filled once).

[ ]:
```

```