Spherical Fourier Neural Operators

A simple notebook to showcase spherical Fourier Neural Operators

Preparation

In [1]:
```
import torch
import torch.nn as nn
from torch.utils.data import DataLoader
from torch.cuda import amp
from torch.optim.lr_scheduler import OneCycleLR

import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation

from math import ceil, sqrt

import time

cmap='twilight_shifted'
```

In [2]:
```
enable_amp = False

# set device
device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
if torch.cuda.is_available():
    torch.cuda.set_device(device.index)
```

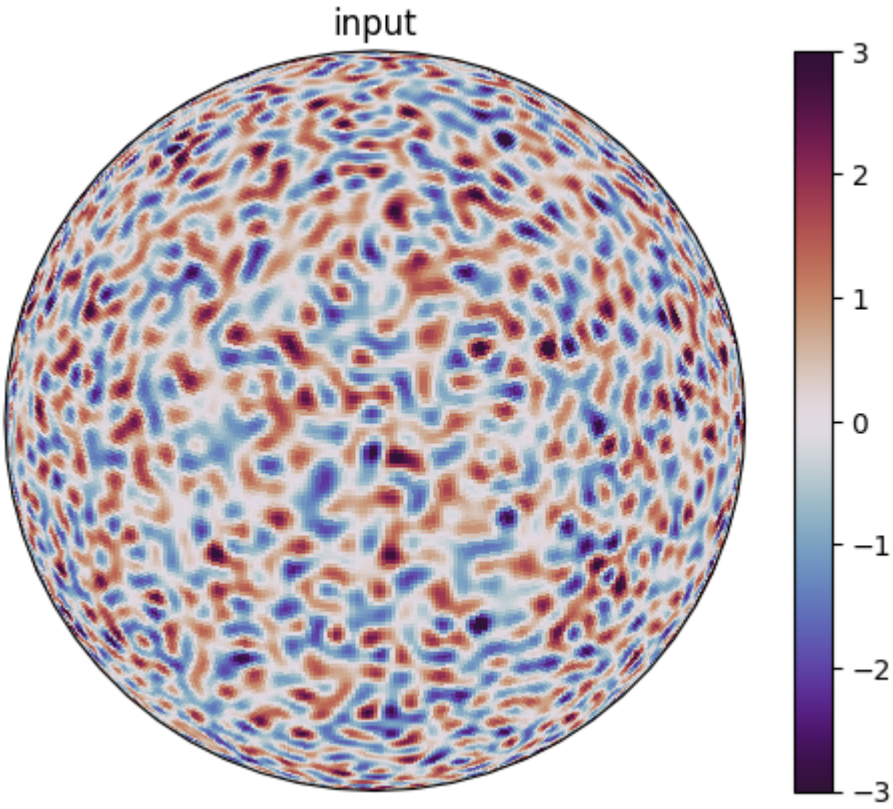
Training data

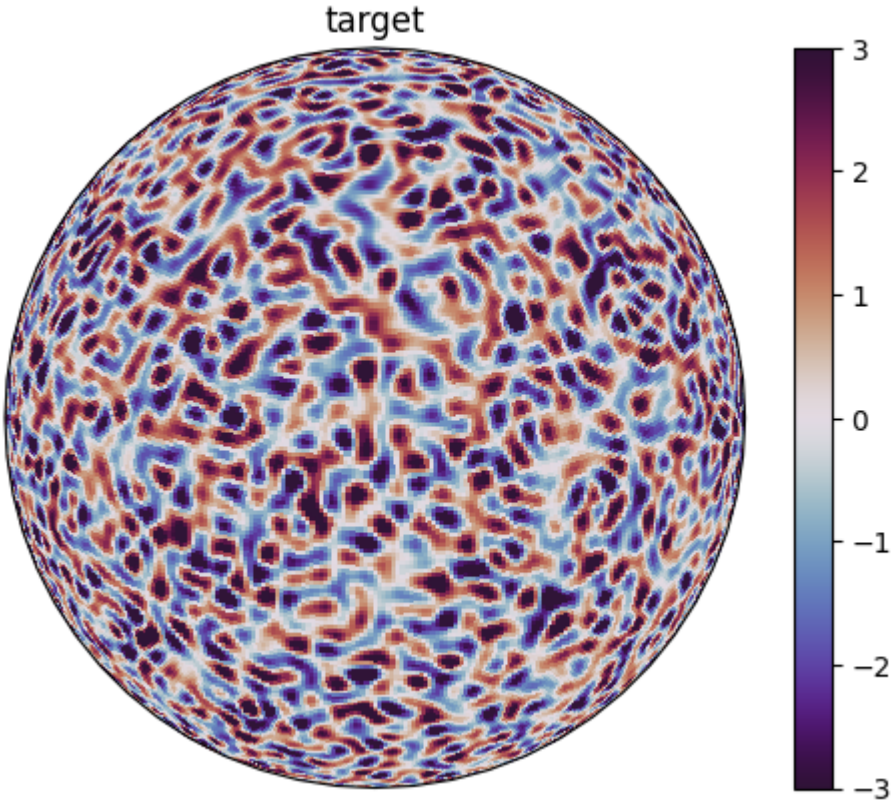
to train our geometric FNOs, we require training data. To this end let us prepare a Dataloader which computes results on the fly:

In [3]:
```
# dataset
from torch_harmonics.examples.sfno import PdeDataset

# 1 hour prediction steps
dt = 1*3600
dt_solver = 150
nsteps = dt//dt_solver
dataset = PdeDataset(dt=dt, nsteps=nsteps, dims=(256, 512), device=device
# There is still an issue with parallel dataloading. Do NOT use it at the
dataloader = DataLoader(dataset, batch_size=4, shuffle=True, num_workers=
solver = dataset.solver.to(device)

nlat = dataset.nlat
nlon = dataset.nlon
```

In [4]:
```
torch.manual_seed(0)
inp, tar = dataset[0]
```

```
fig = plt.figure()
im = solver.plot_griddata(inp[2], fig, vmax=3, vmin=-3)
plt.title("input")
plt.colorbar(im)
plt.show()

fig = plt.figure()
im = solver.plot_griddata(tar[2], fig, vmax=3, vmin=-3)
plt.title("target")
plt.colorbar(im)
plt.show()
```





Defining the geometric Fourier Neural Operator

In [6]:
```
from torch_harmonics.examples.sfno import SphericalFourierNeuralOperatorN
```

In [7]:
```
model = SFNO(spectral_transform='sht', operator_type='driscoll-healy', im
                num_layers=4, scale_factor=3, embed_dim=16, big_skip=Tru
```

In [8]:
```
# pointwise model for sanity checking
# class MLP(nn.Module):
#     def __init__(self,
#                  input_dim = 3,
#                  output_dim = 3,
#                  num_layers = 2,
#                  hidden_dim = 32,
#                  activation_function = nn.ReLU,
#                  bias = False):
#         super().__init__()

#         current_dim = input_dim
#         layers = []
#         for l in range(num_layers-1):
#             fc = nn.Conv2d(current_dim, hidden_dim, 1, bias=True)
#             # initialize the weights correctly
#             scale = sqrt(2. / current_dim)
#             nn.init.normal_(fc.weight, mean=0., std=scale)
#             if fc.bias is not None:
#                 nn.init.constant_(fc.bias, 0.0)
#             layers.append(fc)
#             layers.append(activation_function())
#             current_dim = hidden_dim
#         fc = nn.Conv2d(current_dim, output_dim, 1, bias=False)
#         scale = sqrt(1. / current_dim)
```

```
#         nn.init.normal_(fc.weight, mean=0., std=scale)
#         if fc.bias is not None:
#             nn.init.constant_(fc.bias, 0.0)
#         layers.append(fc)
#         self.mlp = nn.Sequential(*layers)

#     def forward(self, x):
#         return self.mlp(x)

# model = MLP(num_layers=10).to(device)
```

Training the model

In [9]:
```
def l2loss_sphere(solver, prd, tar, relative=False, squared=True):
    loss = solver.integrate_grid((prd - tar)**2, dimensionless=True).sum(
    if relative:
        loss = loss / solver.integrate_grid(tar**2, dimensionless=True).s

    if not squared:
        loss = torch.sqrt(loss)
    loss = loss.mean()

    return loss

def spectral_l2loss_sphere(solver, prd, tar, relative=False, squared=True
    # compute coefficients
    coeffs = torch.view_as_real(solver.sht(prd - tar))
    coeffs = coeffs[..., 0]**2 + coeffs[..., 1]**2
    norm2 = coeffs[..., :, 0] + 2 * torch.sum(coeffs[..., :, 1:], dim=-1)
    loss = torch.sum(norm2, dim=(-1,-2))

    if relative:
        tar_coeffs = torch.view_as_real(solver.sht(tar))
        tar_coeffs = tar_coeffs[..., 0]**2 + tar_coeffs[..., 1]**2
        tar_norm2 = tar_coeffs[..., :, 0] + 2 * torch.sum(tar_coeffs[...,
        tar_norm2 = torch.sum(tar_norm2, dim=(-1,-2))
        loss = loss / tar_norm2

    if not squared:
        loss = torch.sqrt(loss)
    loss = loss.mean()

    return loss
```

In [10]:
```
# training function
def train_model(model, dataloader, optimizer, scheduler=None, nepochs=20,

    train_start = time.time()

    for epoch in range(nepochs):

        # time each epoch
        epoch_start = time.time()

        dataloader.dataset.set_initial_condition('random')
        dataloader.dataset.set_num_examples(num_examples)

        optimizer.zero_grad(set_to_none=True)
```

```
        # do the training
        acc_loss = 0
        model.train()
        for inp, tar in dataloader:
            with amp.autocast(enabled=enable_amp):
                prd = model(inp)
                for _ in range(nfuture):
                    prd = model(prd)
                if loss_fn == 'l2':
                    loss = l2loss_sphere(solver, prd, tar)
                elif loss_fn == "spectral l2":
                    loss = spectral_l2loss_sphere(solver, prd, tar)

            acc_loss += loss.item() * inp.size(0)

            optimizer.zero_grad(set_to_none=True)
            # gscaler.scale(loss).backward()
            loss.backward()
            optimizer.step()
            # gscaler.update()

        if scheduler is not None:
            scheduler.step()

        acc_loss = acc_loss / len(dataloader.dataset)

        dataloader.dataset.set_initial_condition('random')
        dataloader.dataset.set_num_examples(num_valid)

        # perform validation
        valid_loss = 0
        model.eval()
        with torch.no_grad():
            for inp, tar in dataloader:
                prd = model(inp)
                for _ in range(nfuture):
                    prd = model(prd)
                loss = l2loss_sphere(solver, prd, tar, relative=True)

                valid_loss += loss.item() * inp.size(0)

        valid_loss = valid_loss / len(dataloader.dataset)

        epoch_time = time.time() - epoch_start

        print(f'------------------------------------------------------------
        print(f'Epoch {epoch} summary:')
        print(f'time taken: {epoch_time}')
        print(f'accumulated training loss: {acc_loss}')
        print(f'relative validation loss: {valid_loss}')

    train_time = time.time() - train_start

    print(f'------------------------------------------------------------
    print(f'done. Training took {train_time}.')
    return valid_loss
```

In [11]:

```
# set seed
torch.manual_seed(333)
```

```
torch.cuda.manual_seed(333)

optimizer = torch.optim.Adam(model.parameters(), lr=3E-3, weight_decay=0.
gscaler = amp.GradScaler(enabled=enable_amp)
train_model(model, dataloader, optimizer, nepochs=10)

# multistep training
# learning_rate = 5e-4
# optimizer = torch.optim.Adam(fno_model.parameters(), lr=learning_rate)
# dataloader.dataset.nsteps = 2 * dt//dt_solver
# train_model(fno_model, dataloader, optimizer, nepochs=10, nfuture=1)
# dataloader.dataset.nsteps = 1 * dt//dt_solver
```

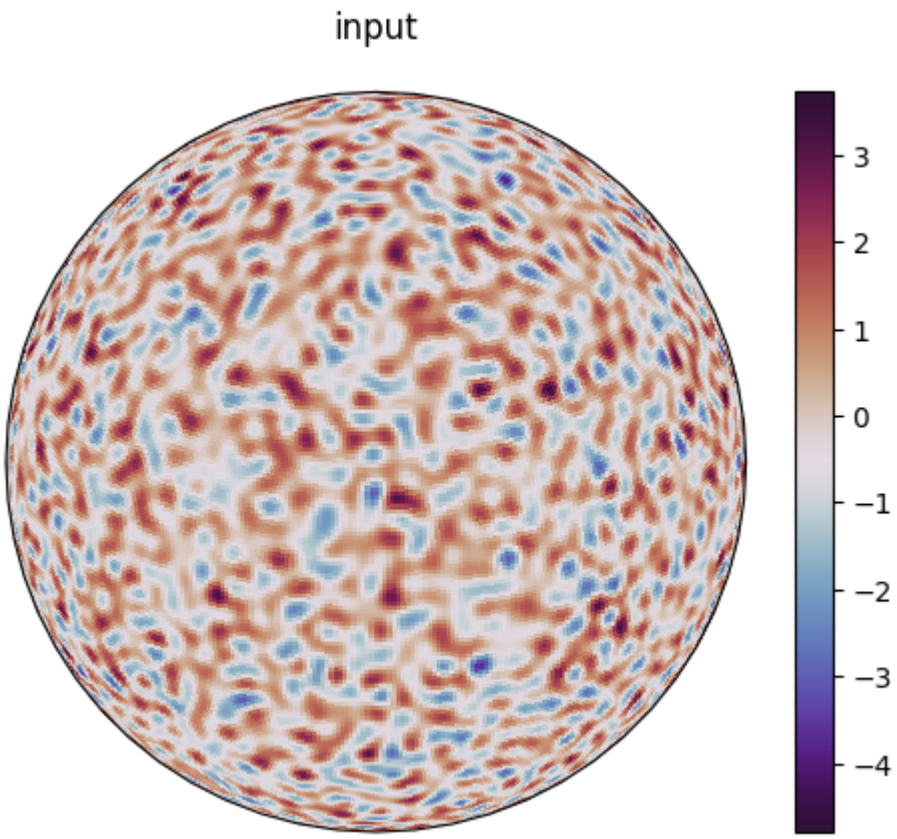
```
/usr/local/lib/python3.10/dist-packages/tqdm/auto.py:21: TqdmWarning: IPro
gress not found. Please update jupyter and ipywidgets. See https://ipywidg
ets.readthedocs.io/en/stable/user_install.html
  from .autonotebook import tqdm as notebook_tqdm
```

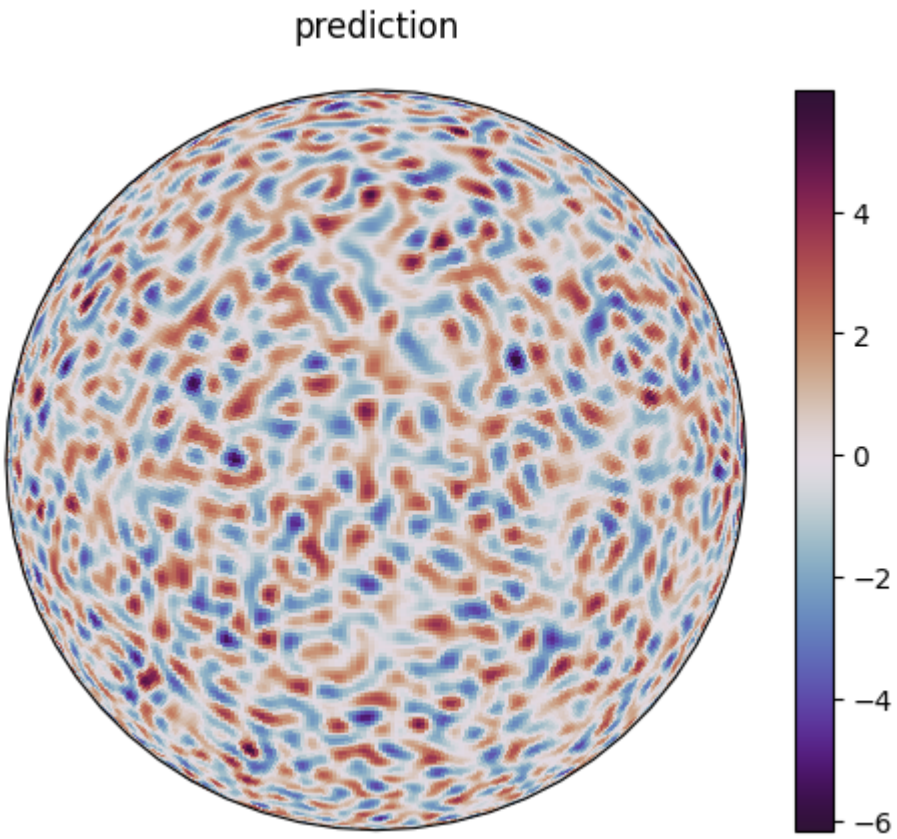
```
--------------------------------------------------------------------------------
------
Epoch 0 summary:
time taken: 16.38287854194641
accumulated training loss: 162.7428795993328
relative validation loss: 0.48357778787612915
--------------------------------------------------------------------------------
------
Epoch 1 summary:
time taken: 16.396941423416138
accumulated training loss: 17.312004193663597
relative validation loss: 0.2579451650381088
--------------------------------------------------------------------------------
------
Epoch 2 summary:
time taken: 15.734673500061035
accumulated training loss: 10.220486253499985
relative validation loss: 0.16569001227617264
--------------------------------------------------------------------------------
------
Epoch 3 summary:
time taken: 15.891995191574097
accumulated training loss: 6.851904459297657
relative validation loss: 0.11703711748123169
--------------------------------------------------------------------------------
------
Epoch 4 summary:
time taken: 15.897265911102295
accumulated training loss: 4.94706941395998
relative validation loss: 0.08656011149287224
--------------------------------------------------------------------------------
------
Epoch 5 summary:
time taken: 15.805784940719604
accumulated training loss: 3.802315466105938
relative validation loss: 0.06816881895065308
--------------------------------------------------------------------------------
------
Epoch 6 summary:
time taken: 16.1594181060791
accumulated training loss: 3.0218399725854397
relative validation loss: 0.05504310131072998
--------------------------------------------------------------------------------
------
Epoch 7 summary:
time taken: 15.709057569503784
accumulated training loss: 2.477198362350464
relative validation loss: 0.04607727192342281
--------------------------------------------------------------------------------
------
Epoch 8 summary:
time taken: 16.11759305000305
accumulated training loss: 2.083927759900689
relative validation loss: 0.039428114891052246
--------------------------------------------------------------------------------
------
Epoch 9 summary:
time taken: 16.217236042022705
accumulated training loss: 1.7926970832049847
relative validation loss: 0.0340796560049057
```

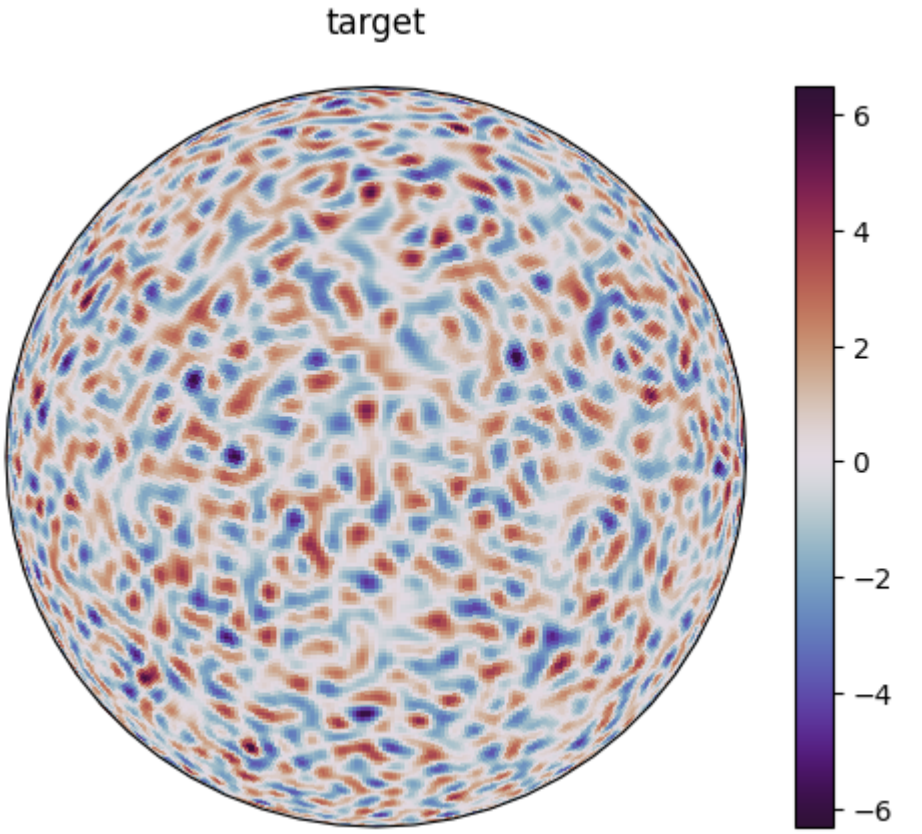
```
------------------------------------------------------------------------
------
done. Training took 160.31519532203674.
```

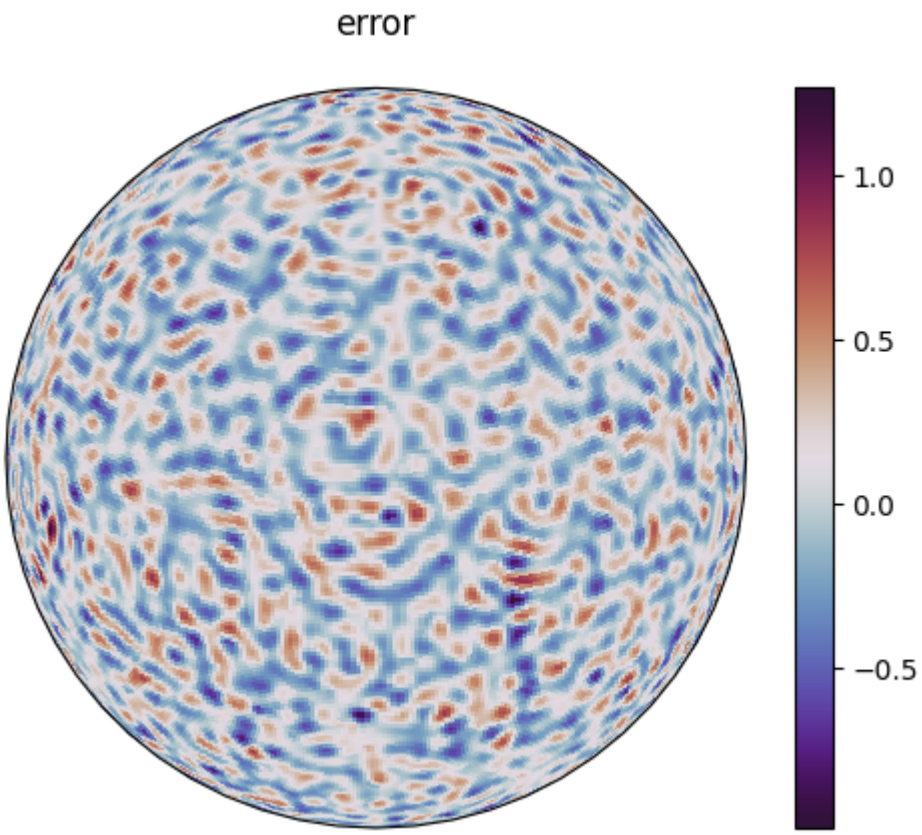
Out[11]: 0.0340796560049057

In [12]:
```
dataloader.dataset.set_initial_condition('random')

torch.manual_seed(0)

with torch.inference_mode():
    inp, tar = next(iter(dataloader))
    out = model(inp).detach()

s = 0; ch = 2

fig = plt.figure()
im = solver.plot_griddata(inp[s, ch], fig, projection='3d', title='input'
plt.colorbar(im)
plt.show()

fig = plt.figure()
im = solver.plot_griddata(out[s, ch], fig, projection='3d', title='predic
plt.colorbar(im)
plt.show()

fig = plt.figure()
im = solver.plot_griddata(tar[s, ch], fig, projection='3d', title='target
plt.colorbar(im)
plt.show()

fig = plt.figure()
im = solver.plot_griddata((tar-out)[s, ch], fig, projection='3d', title='
plt.colorbar(im)
plt.show()
```


input
3
2
1
0
−1
−2
−3
−4


prediction
4
2
0
−2
−4
−6





In [ ]: