Informer Demo

Download code and dataset

```
!git clone https://github.com/zhouhaoyi/Informer2020.git
!git clone https://github.com/zhouhaoyi/ETDataset.git
!ls
```

```
fatal: destination path 'Informer2020' already exists and is not an empty directory.
fatal: destination path 'ETDataset' already exists and is not an empty directory.
checkpoints  ETDataset  Informer2020  results  sample_data
```

```
import sys
if not 'Informer2020' in sys.path:
    sys.path += ['Informer2020']
```

```
# !pip install -r ./Informer2020/requirements.txt
```

Experiments: Train and Test

```
from utils.tools import dotdict
from exp.exp_informer import Exp_Informer
import torch
```

```
args = dotdict()

args.model = 'informer' # model of experiment, options: [informer, informerstack, informer

args.data = 'ETTh2' # data
args.root_path = './ETDataset/ETT-small/' # root path of data file
args.data_path = 'ETTh2.csv' # data file
args.features = 'M' # forecasting task, options:[M, S, MS(TBD)]; M:multivariate predict mu
args.target = 'OT' # target feature in S or MS task
args.freq = 'h' # freq for time features encoding

args.seq_len = 168 # input sequence length of Informer encoder
args.label_len = 48 # start token length of Informer decoder
args.pred_len = 48 # prediction sequence length
# Informer decoder input: concat[start token series(label_len), zero padding series(pred_l

args.enc_in = 7 # encoder input size
args.dec_in = 7 # decoder input size
args.c_out = 7 # output size
args.factor = 5 # probsparse attn factor
```

```
    args.d_model = 512 # dimension of model
    args.n_heads = 8 # num of heads
    args.e_layers = 3 # num of encoder layers
    args.d_layers = 2 # num of decoder layers
    args.d_ff = 512 # dimension of fcn in model
    args.dropout = 0.05 # dropout
    args.attn = 'prob' # attention used in encoder, options:[prob, full]
    args.embed = 'fixed' # time features encoding, options:[timeF, fixed, learned]
    args.activation = 'gelu' # activation
    args.distil = True # whether to use distilling in encoder
    args.output_attention = False # whether to output attention in ecoder

    args.batch_size = 32
    args.learning_rate = 0.0001
    args.loss = 'mse'
    args.lradj = 'type2'

    args.num_workers = 0
    args.itr = 1
    args.train_epochs = 6
    args.patience = 3
    args.des = 'exp'

    args.use_gpu = True if torch.cuda.is_available() else False
    args.gpu = 0



    # Set augments by using data name
    data_parser = {
        'ETTh1':{'data':'ETTh1.csv','T':'OT','M':[7,7,7],'S':[1,1,1],'MS':[7,7,1]},
        'ETTh2':{'data':'ETTh2.csv','T':'OT','M':[7,7,7],'S':[1,1,1],'MS':[7,7,1]},
        'ETTm1':{'data':'ETTm1.csv','T':'OT','M':[7,7,7],'S':[1,1,1],'MS':[7,7,1]},
        'ETTm2':{'data':'ETTm2.csv','T':'OT','M':[7,7,7],'S':[1,1,1],'MS':[7,7,1]},
    }
    if args.data in data_parser.keys():
        data_info = data_parser[args.data]
        args.data_path = data_info['data']
        args.target = data_info['T']
        args.enc_in, args.dec_in, args.c_out = data_info[args.features]


    print('Args in experiment:')
    print(args)


         odel': 512, 'n_heads': 8, 'e_layers': 3, 'd_layers': 2, 'd_ff': 512, 'dropout': 0.05,



    Exp = Exp_Informer


    for ii in range(args.itr):
        # setting record of experiments
        setting = '{}_{}_ft{}_sl{}_ll{}_pl{}_dm{}_nh{}_el{}_dl{}_df{}_at{}_eb{}_dt{}_{}_{}'.fo
```

```
            args.seq_len, args.label_len, args.pred_len,
            args.d_model, args.n_heads, args.e_layers, args.d_layers, args.d_ff, args.

    # set experiments
    exp = Exp(args)

    # train
    print('>>>>>>>start training : {}>>>>>>>>>>>>>>>>>>>>>>>>>>'.format(setting))
    exp.train(setting)

    # test
    print('>>>>>>>testing : {}<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<'.format(setting))
    exp.test(setting)

    torch.cuda.empty_cache()
```

```
 Use GPU: cuda:0
 >>>>>>>start training : informer_ETTh2_ftM_sl168_ll48_pl48_dm512_nh8_el3_dl2_df512_a
 train 8425
 val 2833
 test 2833
        iters: 100, epoch: 1 | loss: 0.4019844
        speed: 1.6514s/iter; left time: 2442.3852s
        iters: 200, epoch: 1 | loss: 0.2388133
        speed: 1.6601s/iter; left time: 2289.2645s
 Epoch: 1, Steps: 263 | Train Loss: 0.3225119 Vali Loss: 1.1832668 Test Loss: 3.70465
 Validation loss decreased (inf --> 1.183267).  Saving model ...
        iters: 100, epoch: 2 | loss: 0.2530593
        speed: 3.7682s/iter; left time: 4582.1903s
        iters: 200, epoch: 2 | loss: 0.2107259
        speed: 1.6605s/iter; left time: 1853.0956s
 Epoch: 2, Steps: 263 | Train Loss: 0.2021973 Vali Loss: 1.4130711 Test Loss: 5.15107
 EarlyStopping counter: 1 out of 3
 Updating learning rate to 5e-05
        iters: 100, epoch: 3 | loss: 0.1721831
        speed: 3.7655s/iter; left time: 3588.5574s
        iters: 200, epoch: 3 | loss: 0.1690238
        speed: 1.6603s/iter; left time: 1416.2225s
 Epoch: 3, Steps: 263 | Train Loss: 0.1594383 Vali Loss: 1.3281847 Test Loss: 4.42330
 EarlyStopping counter: 2 out of 3
        iters: 100, epoch: 4 | loss: 0.1203492
        speed: 3.7654s/iter; left time: 2598.1427s
        iters: 200, epoch: 4 | loss: 0.1144888
        speed: 1.6599s/iter; left time: 979.3229s
 Epoch: 4, Steps: 263 | Train Loss: 0.1454202 Vali Loss: 1.1759692 Test Loss: 4.10739
 Validation loss decreased (1.183267 --> 1.175969).  Saving model ...
 Updating learning rate to 1e-05
        iters: 100, epoch: 5 | loss: 0.1190969
        speed: 3.7666s/iter; left time: 1608.3196s
        iters: 200, epoch: 5 | loss: 0.0972755
        speed: 1.6607s/iter; left time: 543.0547s
 Epoch: 5, Steps: 263 | Train Loss: 0.1236629 Vali Loss: 1.2202918 Test Loss: 4.59752
 EarlyStopping counter: 1 out of 3
        iters: 100, epoch: 6 | loss: 0.1017893
        speed: 3.7640s/iter; left time: 617.2965s
        iters: 200, epoch: 6 | loss: 0.1223550
        speed: 1.6594s/iter; left time: 106.2046s
 Epoch: 6, Steps: 263 | Train Loss: 0.1191037 Vali Loss: 1.2033780 Test Loss: 4.51555
 EarlyStopping counter: 2 out of 3
```

```
Updating learning rate to 5e-06
>>>>>>>testing : informer_ETTh2_ftM_sl168_ll48_pl48_dm512_nh8_el3_dl2_df512_atprob_e
test 2833
test shape: (88, 32, 48, 7) (88, 32, 48, 7)
test shape: (2816, 48, 7) (2816, 48, 7)
mse:4.102377383754676, mae:1.6822883001914781
```

Prediction

```
import os

# set model path
setting = 'informer_ETTh2_ftM_sl168_ll48_pl48_dm512_nh8_el3_dl2_df512_atprob_ebfixed_dtTru
path = os.path.join('./checkpoints/',setting,'checkpoint.pth')


from data.data_loader import Dataset_ETT_hour
from torch.utils.data import DataLoader

# set prediction dataloader (using test dataloader here)
Data = Dataset_ETT_hour
timeenc = 0 if args.embed!='fixed' else 1
flag = 'test'; shuffle_flag = False; drop_last = True; batch_size = 1

data_set = Data(
    root_path=args.root_path,
    data_path=args.data_path,
    flag=flag,
    size=[args.seq_len, args.label_len, args.pred_len],
    features=args.features,
    target=args.target,
    timeenc=timeenc,
    freq=args.freq
)
data_loader = DataLoader(
    data_set,
    batch_size=batch_size,
    shuffle=shuffle_flag,
    num_workers=args.num_workers,
    drop_last=drop_last)


from models.model import Informer

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

args.output_attention = True

# build model
model = Informer(
    args.enc_in,
```

```
        args.dec_in,
        args.c_out,
        args.seq_len,
        args.label_len,
        args.pred_len,
        args.factor,
        args.d_model,
        args.n_heads,
        args.e_layers,
        args.d_layers,
        args.d_ff,
        args.dropout,
        args.attn,
        args.embed,
        args.freq,
        args.activation,
        args.output_attention,
        args.distil,
        device
    )


    # load parameters
    model.load_state_dict(torch.load(path))

         <All keys matched successfully>


    model = model.double().to(device)
    model.eval()
                  (value_projection): Linear(in_features=512, out_features=512, bias=True
                  (out_projection): Linear(in_features=512, out_features=512, bias=True)
                )
                (conv1): Conv1d(512, 512, kernel_size=(1,), stride=(1,))
                (conv2): Conv1d(512, 512, kernel_size=(1,), stride=(1,))
                (norm1): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
                (norm2): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
                (dropout): Dropout(p=0.05, inplace=False)
              )
            )
            (conv_layers): ModuleList(
              (0): ConvLayer(
                (downConv): Conv1d(512, 512, kernel_size=(3,), stride=(1,), padding=(2,),
                (norm): BatchNorm1d(512, eps=1e-05, momentum=0.1, affine=True, track_runn
                (activation): ELU(alpha=1.0)
                (maxPool): MaxPool1d(kernel_size=3, stride=2, padding=1, dilation=1, ceil
              )
              (1): ConvLayer(
                (downConv): Conv1d(512, 512, kernel_size=(3,), stride=(1,), padding=(2,),
                (norm): BatchNorm1d(512, eps=1e-05, momentum=0.1, affine=True, track_runn
                (activation): ELU(alpha=1.0)
                (maxPool): MaxPool1d(kernel_size=3, stride=2, padding=1, dilation=1, ceil
              )
            )
            (norm): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
          )
          (decoder): Decoder(
            (layers): ModuleList(
              (0): DecoderLayer(
```

```
            (0): DecoderLayer(
              (self_attention): AttentionLayer(
                (inner_attention): ProbAttention(
                  (dropout): Dropout(p=0.05, inplace=False)
                )
                (query_projection): Linear(in_features=512, out_features=512, bias=True
                (key_projection): Linear(in_features=512, out_features=512, bias=True)
                (value_projection): Linear(in_features=512, out_features=512, bias=True
                (out_projection): Linear(in_features=512, out_features=512, bias=True)
              )
              (cross_attention): AttentionLayer(
                (inner_attention): FullAttention(
                  (dropout): Dropout(p=0.05, inplace=False)
                )
                (query_projection): Linear(in_features=512, out_features=512, bias=True
                (key_projection): Linear(in_features=512, out_features=512, bias=True)
                (value_projection): Linear(in_features=512, out_features=512, bias=True
                (out_projection): Linear(in_features=512, out_features=512, bias=True)
              )
              (conv1): Conv1d(512, 512, kernel_size=(1,), stride=(1,))
              (conv2): Conv1d(512, 512, kernel_size=(1,), stride=(1,))
              (norm1): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
              (norm2): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
              (norm3): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
              (dropout): Dropout(p=0.05, inplace=False)
            )
            (1): DecoderLayer(
              (self_attention): AttentionLayer(
                (inner_attention): ProbAttention(
                  (dropout): Dropout(p=0.05, inplace=False)
                )
```

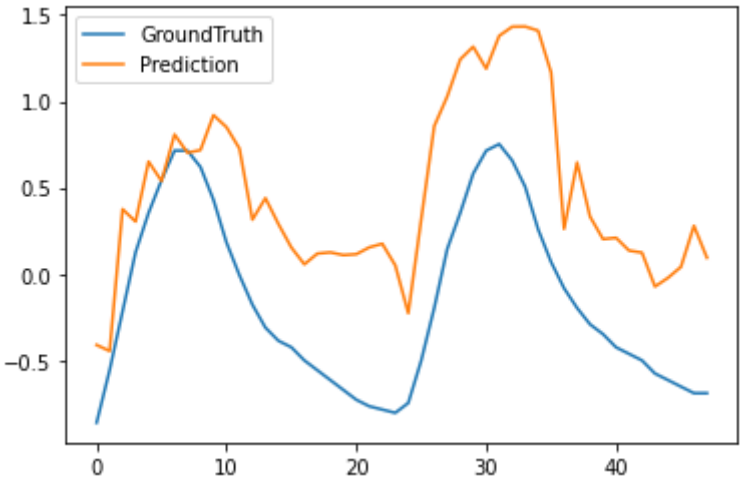
```
preds = []
trues = []

for i, (batch_x,batch_y,batch_x_mark,batch_y_mark) in enumerate(data_loader):
    batch_x = batch_x.double().to(device)
    batch_y = batch_y.double()
    batch_x_mark = batch_x_mark.double().to(device)
    batch_y_mark = batch_y_mark.double().to(device)

    # decoder input = concat[start token series(label_len), zero padding series(pred_len)]
    dec_inp = torch.zeros_like(batch_y[:,-args.pred_len:,:]).double()
    dec_inp = torch.cat([batch_y[:,:args.label_len,:], dec_inp], dim=1).double().to(device

    # encoder - decoder
    if args.output_attention:
        outputs = model(batch_x, batch_x_mark, dec_inp, batch_y_mark)[0]
    else:
        outputs = model(batch_x, batch_x_mark, dec_inp, batch_y_mark)
    batch_y = batch_y[:,-args.pred_len:,:]

    pred = outputs.detach().cpu().numpy()#.squeeze()
    true = batch_y.detach().cpu().numpy()#.squeeze()

    preds.append(pred)
    trues.append(true)
```

```
import numpy as np

preds = np.array(preds)
trues = np.array(trues)

print('prediction shape:', preds.shape, trues.shape) # [num_samples//batch_size, batch_siz
preds = preds.reshape(-1, preds.shape[-2], preds.shape[-1])
trues = trues.reshape(-1, trues.shape[-2], trues.shape[-1])
print('prediction shape:', preds.shape, trues.shape) # [num_samples, pred_len, c_out]
```

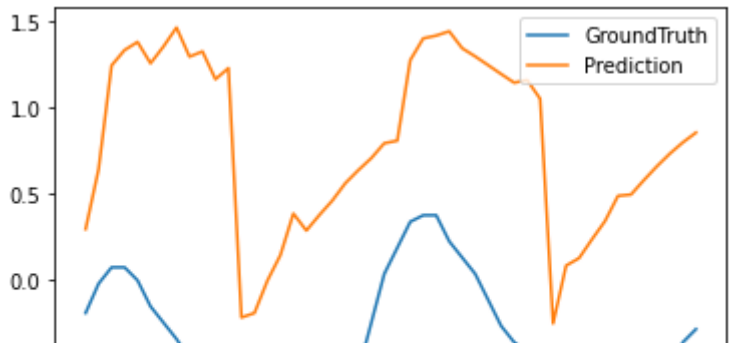
```
prediction shape: (2833, 1, 48, 7) (2833, 1, 48, 7)
prediction shape: (2833, 48, 7) (2833, 48, 7)
```

Visualization

```
import matplotlib.pyplot as plt
import seaborn as sns


# draw OT prediction
plt.figure()
plt.plot(trues[200,:,-1], label='GroundTruth')
plt.plot(preds[200,:,-1], label='Prediction')
plt.legend()
plt.show()
```




```
# draw HUFL prediction
plt.figure()
plt.plot(trues[300,:,-1], label='GroundTruth')
plt.plot(preds[300,:,-1], label='Prediction')
plt.legend()
plt.show()
```




```
# attention visualization
idx = 0
for i, (batch_x,batch_y,batch_x_mark,batch_y_mark) in enumerate(data_loader):
    if i!=idx:SS
        continue
    batch_x = batch_x.double().to(device)
    batch_y = batch_y.double()

    batch_x_mark = batch_x_mark.double().to(device)
    batch_y_mark = batch_y_mark.double().to(device)

    dec_inp = torch.zeros_like(batch_y[:,-args.pred_len:,:]).double()
    dec_inp = torch.cat([batch_y[:,:args.label_len,:], dec_inp], dim=1).double().to(device

    outputs,attn = model(batch_x, batch_x_mark, dec_inp, batch_y_mark)
```

```
   File "<ipython-input-20-a279ffab0ca0>", line 5
     continue
     ^
 IndentationError: unexpected indent
```

SEARCH STACK OVERFLOW

```
attn[0].shape, attn[1].shape, attn[2].shape


layer = 0
distil = 'Distil' if args.distil else 'NoDistil'
for h in range(0,8):
    plt.figure(figsize=[10,8])
    plt.title('Informer, {}, attn:{} layer:{} head:{}'.format(distil, args.attn, layer, h)
    A = attn[layer][0,h].detach().cpu().numpy()
    ax = sns.heatmap(A, vmin=0, vmax=A.max()+0.01)
    plt.show()


layer = 1
distil = 'Distil' if args.distil else 'NoDistil'
for h in range(0,8):
    plt.figure(figsize=[10,8])
    plt.title('Informer, {}, attn:{} layer:{} head:{}'.format(distil, args.attn, layer, h)
    A = attn[layer][0,h].detach().cpu().numpy()
    ax = sns.heatmap(A, vmin=0, vmax=A.max()+0.01)
    plt.show()
```

Custom Data

Custom data (xxx.csv) has to include at least 2 features: `date` (format: `YYYY-MM-DD hh:mm:ss`) and `target feature`.

```
from data.data_loader import Dataset_Custom
from torch.utils.data import DataLoader
import pandas as pd
import os


# custom data: xxx.csv
# data features: ['date', ...(other features), target feature]

# we take ETTh2 as an example
args.root_path = './ETDataset/ETT-small/'
args.data_path = 'ETTh2.csv'

df = pd.read_csv(os.path.join(args.root_path, args.data_path))


df.head()


'''
We set 'HULL' as target instead of 'OT'

The following frequencies are supported:
        Y   - yearly
            alias: A
        M   - monthly
        W   - weekly
        D   - daily
        B   - business days
        H   - hourly
        T   - minutely
            alias: min
'''


args.target = 'HULL'
args.freq = 'h'

Data = Dataset_Custom
timeenc = 0 if args.embed!='timeF' else 1
flag = 'test'; shuffle_flag = False; drop_last = True; batch_size = 1

data_set = Data(
    root_path=args.root_path,
    data_path=args.data_path,
    flag=flag,
    size=[args.seq_len, args.label_len, args.pred_len],
    features=args.features,
    timeenc=timeenc,
```

```
        target=args.target, # HULL here
        freq=args.freq # 'h': hourly, 't':minutely
    )
    data_loader = DataLoader(
        data_set,
        batch_size=batch_size,
        shuffle=shuffle_flag,
        num_workers=args.num_workers,
        drop_last=drop_last)


    batch_x,batch_y,batch_x_mark,batch_y_mark = data_set[0]


    batch_x, batch_x_mark
```