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.
ETDataset  Informer2020  informer_checkpoints  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, informerlight(TBD)]
```

```
args.data = 'ETTh1' # data
args.root_path = './ETDataset/ETT-small/' # root path of data file
args.data_path = 'ETTh2.csv' # data file
args.features = 'S' # forecasting task, options:[M, S, MS]; M:multivariate predict multivariate, S:univariate predict univariate, MS:
args.target = 'OT' # target feature in S or MS task
args.freq = 'h' # freq for time features encoding, options:[s:secondly, t:minutely, h:hourly, d:daily, b:business days, w:weekly, m:m
args.checkpoints = './informer_checkpoints' # location of model checkpoints

args.seq_len = 336 # input sequence length of Informer encoder
args.label_len = 336 # start token length of Informer decoder
args.pred_len = 168 # prediction sequence length
# Informer decoder input: concat[start token series(label_len), zero padding series(pred_len)]

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 = 2 # num of encoder layers
args.d_layers = 1 # num of decoder layers
args.d_ff = 1024 # dimension of fcn in model
args.dropout = 0.05 # dropout
args.attn = 'prob' # attention used in encoder, options:[prob, full]
args.embed = 'timeF' # 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 = 'type1'
args.use_amp = False # whether to use automatic mixed precision training

args.num_workers = 0
args.itr = 1
```

```
    args.train_epochs = 5
    args.patience = 3
    args.des = 'exp'

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

    args.use_multi_gpu = False
    args.devices = '0,1,2,3'



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

    if args.use_gpu and args.use_multi_gpu:
        args.dvices = args.devices.replace(' ','')
        device_ids = args.devices.split(',')
        args.device_ids = [int(id_) for id_ in device_ids]
        args.gpu = args.device_ids[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]


    args.detail_freq = args.freq
    args.freq = args.freq[-1:]
```

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

```
Args in experiment:
{'model': 'informer', 'data': 'ETTh1', 'root_path': './ETDataset/ETT-small/', 'data_path': 'ETTh1.csv', 'features': 'S', 'targe
```

```
Exp = Exp_Informer


for ii in range(args.itr):
    # setting record of experiments
    setting = '{}_{}_ft{}_sl{}_ll{}_pl{}_dm{}_nh{}_el{}_dl{}_df{}_at{}_fc{}_eb{}_dt{}_{}_{}'.format(args.model, args.data, args.featu
                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.attn, args.factor, args.embed, args.distil,

    # 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_ETTh1_ftS_sl336_ll336_pl168_dm512_nh8_el2_dl1_df1024_atprob_fc5_ebtimeF_dtTrue_exp_0>>>>>>>>>>
train 8137
val 2713
test 2713
        iters: 100, epoch: 1 | loss: 0.1744897
        speed: 0.1042s/iter; left time: 121.9991s
        iters: 200, epoch: 1 | loss: 0.1609611
        speed: 0.1043s/iter; left time: 111.7024s
```

```
Epoch: 1 cost time: 26.57021474838257
Epoch: 1, Steps: 254 | Train Loss: 0.1977907 Vali Loss: 0.2164623 Test Loss: 0.1741498
Validation loss decreased (inf --> 0.216462).  Saving model ...
Updating learning rate to 0.0001
        iters: 100, epoch: 2 | loss: 0.1235687
        speed: 0.2339s/iter; left time: 214.5123s
        iters: 200, epoch: 2 | loss: 0.1140385
        speed: 0.1042s/iter; left time: 85.1268s
Epoch: 2 cost time: 26.576510429382324
Epoch: 2, Steps: 254 | Train Loss: 0.1166397 Vali Loss: 0.2104181 Test Loss: 0.2314121
Validation loss decreased (0.216462 --> 0.210418).  Saving model ...
Updating learning rate to 5e-05
        iters: 100, epoch: 3 | loss: 0.0791930
        speed: 0.2337s/iter; left time: 154.9438s
        iters: 200, epoch: 3 | loss: 0.0823716
        speed: 0.1045s/iter; left time: 58.8256s
Epoch: 3 cost time: 26.57927441596985
Epoch: 3, Steps: 254 | Train Loss: 0.0877812 Vali Loss: 0.1973333 Test Loss: 0.2568179
Validation loss decreased (0.210418 --> 0.197333).  Saving model ...
Updating learning rate to 2.5e-05
        iters: 100, epoch: 4 | loss: 0.0698325
        speed: 0.2329s/iter; left time: 95.2703s
        iters: 200, epoch: 4 | loss: 0.0772824
        speed: 0.1040s/iter; left time: 32.1405s
Epoch: 4 cost time: 26.454782962799072
Epoch: 4, Steps: 254 | Train Loss: 0.0770904 Vali Loss: 0.2114129 Test Loss: 0.2677161
EarlyStopping counter: 1 out of 3
Updating learning rate to 1.25e-05
        iters: 100, epoch: 5 | loss: 0.0799557
        speed: 0.2318s/iter; left time: 35.9231s
        iters: 200, epoch: 5 | loss: 0.0727806
        speed: 0.1044s/iter; left time: 5.7447s
Epoch: 5 cost time: 26.51810050010681
Epoch: 5, Steps: 254 | Train Loss: 0.0721366 Vali Loss: 0.2111876 Test Loss: 0.2788309
EarlyStopping counter: 2 out of 3
Updating learning rate to 6.25e-06
>>>>>>>testing : informer_ETTh1_ftS_sl336_ll336_pl168_dm512_nh8_el2_dl1_df1024_atprob_fc5_ebtimeF_dtTrue_exp_0<<<<<<<<<<<<<<<<
test 2713
test shape: (84, 32, 168, 1) (84, 32, 168, 1)
test shape: (2688, 168, 1) (2688, 168, 1)
mse:0.25672146677970886, mae:0.43616965413093567
```

Prediction

```
import os

# set saved model path
setting = 'informer_ETTh1_ftS_sl336_ll336_pl168_dm512_nh8_el2_dl1_df1024_atprob_fc5_ebtimeF_dtTrue_exp_0'
# path = os.path.join(args.checkpoints,setting,'checkpoint.pth')


# If you already have a trained model, you can set the arguments and model path, then initialize a Experiment and use it to predict
# Prediction is a sequence which is adjacent to the last date of the data, and does not exist in the data
# If you want to get more information about prediction, you can refer to code `exp/exp_informer.py function predict()` and `data/data

exp = Exp(args)

exp.predict(setting, True)
```

```
     Use GPU: cuda:0
     pred 1
```

```
# the prediction will be saved in ./results/{setting}/real_prediction.npy
import numpy as np

prediction = np.load('./results/'+setting+'/real_prediction.npy')

prediction.shape
```

```
     (1, 168, 1)
```

More details about Prediction - prediction function

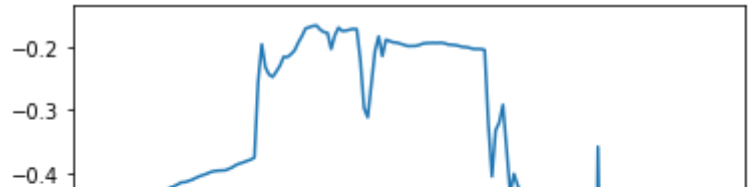
```
# here is the detailed code of function predict
```

```
    # here is the detailed code of function predict

    def predict(exp, setting, load=False):
        pred_data, pred_loader = exp._get_data(flag='pred')

        if load:
            path = os.path.join(exp.args.checkpoints, setting)
            best_model_path = path+'/'+'checkpoint.pth'
            exp.model.load_state_dict(torch.load(best_model_path))

        exp.model.eval()

        preds = []

        for i, (batch_x,batch_y,batch_x_mark,batch_y_mark) in enumerate(pred_loader):
            batch_x = batch_x.float().to(exp.device)
            batch_y = batch_y.float()
            batch_x_mark = batch_x_mark.float().to(exp.device)
            batch_y_mark = batch_y_mark.float().to(exp.device)

            # decoder input
            dec_inp = torch.zeros_like(batch_y[:,-exp.args.pred_len:,:]).float()
            dec_inp = torch.cat([batch_y[:,:exp.args.label_len,:], dec_inp], dim=1).float().to(exp.device)
            # encoder - decoder
            if exp.args.use_amp:
                with torch.cuda.amp.autocast():
                    if exp.args.output_attention:
                        outputs = exp.model(batch_x, batch_x_mark, dec_inp, batch_y_mark)[0]
                    else:
                        outputs = exp.model(batch_x, batch_x_mark, dec_inp, batch_y_mark)
            else:
                if exp.args.output_attention:
                    outputs = exp.model(batch_x, batch_x_mark, dec_inp, batch_y_mark)[0]
                else:
                    outputs = exp.model(batch_x, batch_x_mark, dec_inp, batch_y_mark)
            f_dim = -1 if exp.args.features=='MS' else 0
            batch_y = batch_y[:,-exp.args.pred_len:,f_dim:].to(exp.device)
```

```
            pred = outputs.detach().cpu().numpy()#.squeeze()

            preds.append(pred)

        preds = np.array(preds)
        preds = preds.reshape(-1, preds.shape[-2], preds.shape[-1])

        # result save
        folder_path = './results/' + setting +'/'
        if not os.path.exists(folder_path):
            os.makedirs(folder_path)

        np.save(folder_path+'real_prediction.npy', preds)

        return preds



    # you can also use this prediction function to get result
    prediction = predict(exp, setting, True)
```

pred 1

```
    import matplotlib.pyplot as plt

    plt.figure()
    plt.plot(prediction[0,:,-1])
    plt.show()
```



More details about Prediction - prediction dataset

You can give a `root_path` and `data_path` of the data you want to forecast, and set `seq_len`, `label_len`, `pred_len` and other arguments as other Dataset. The difference is that you can set a more detailed freq such as `15min` or `3h` to generate the timestamp of prediction series.

`Dataset_Pred` only has one sample (including `encoder_input: [1, seq_len, dim]`, `decoder_token: [1, label_len, dim]`, `encoder_input_timestamp: [1, seq_len, date_dim]`, `decoder_input_timstamp: [1, label_len+pred_len, date_dim]`). It will intercept the last sequence of the given data (seq_len data) to forecast the unseen future sequence (pred_len data).

```
from data.data_loader import Dataset_Pred
from torch.utils.data import DataLoader


Data = Dataset_Pred
timeenc = 0 if args.embed!='timeF' else 1
flag = 'pred'; shuffle_flag = False; drop_last = False; batch_size = 1

freq = args.detail_freq

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=freq
)
data_loader = DataLoader(
```

```
        data_set,
        batch_size=batch_size,
        shuffle=shuffle_flag,
        num_workers=args.num_workers,
        drop_last=drop_last)


    len(data_set), len(data_loader)
```

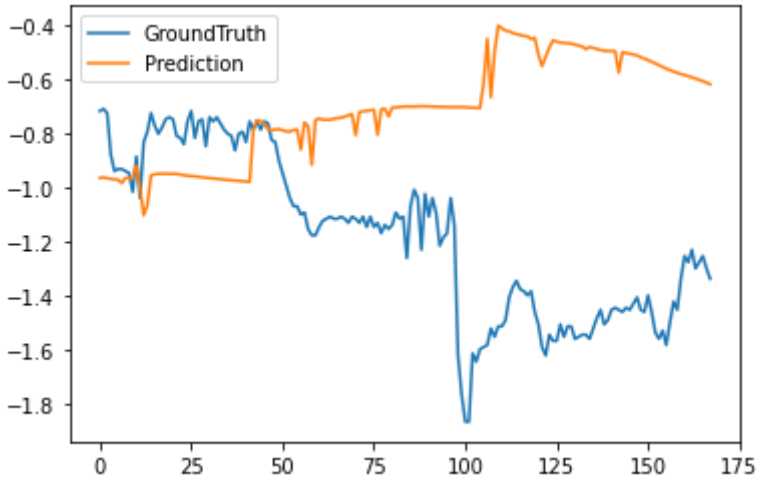
```
    (1, 1)
```

Visualization

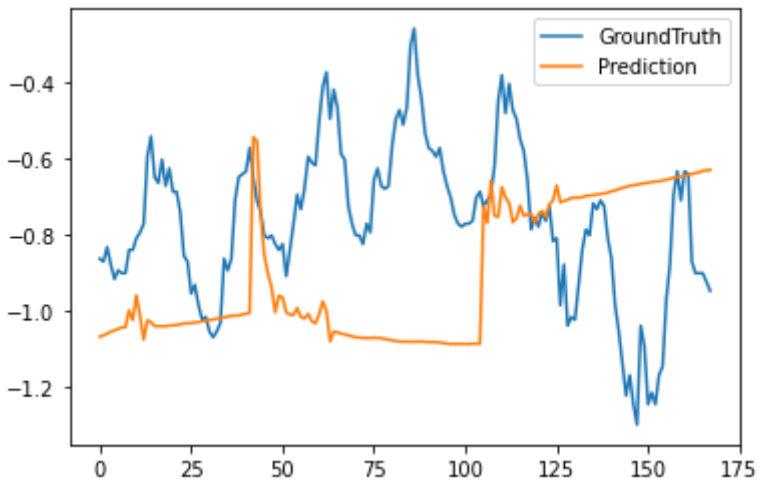
```
    # When we finished exp.train(setting) and exp.test(setting), we will get a trained model and the results of test experiment
    # The results of test experiment will be saved in ./results/{setting}/pred.npy (prediction of test dataset) and ./results/{setting}/t

    preds = np.load('./results/'+setting+'/pred.npy')
    trues = np.load('./results/'+setting+'/true.npy')

    # [samples, pred_len, dimensions]
    preds.shape, trues.shape
```

```
    ((2688, 168, 1), (2688, 168, 1))
```

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


    # draw OT prediction
    plt.figure()
    plt.plot(trues[500,:,-1], label='GroundTruth')
    plt.plot(preds[500,:,-1], label='Prediction')
    plt.legend()
    plt.show()
```



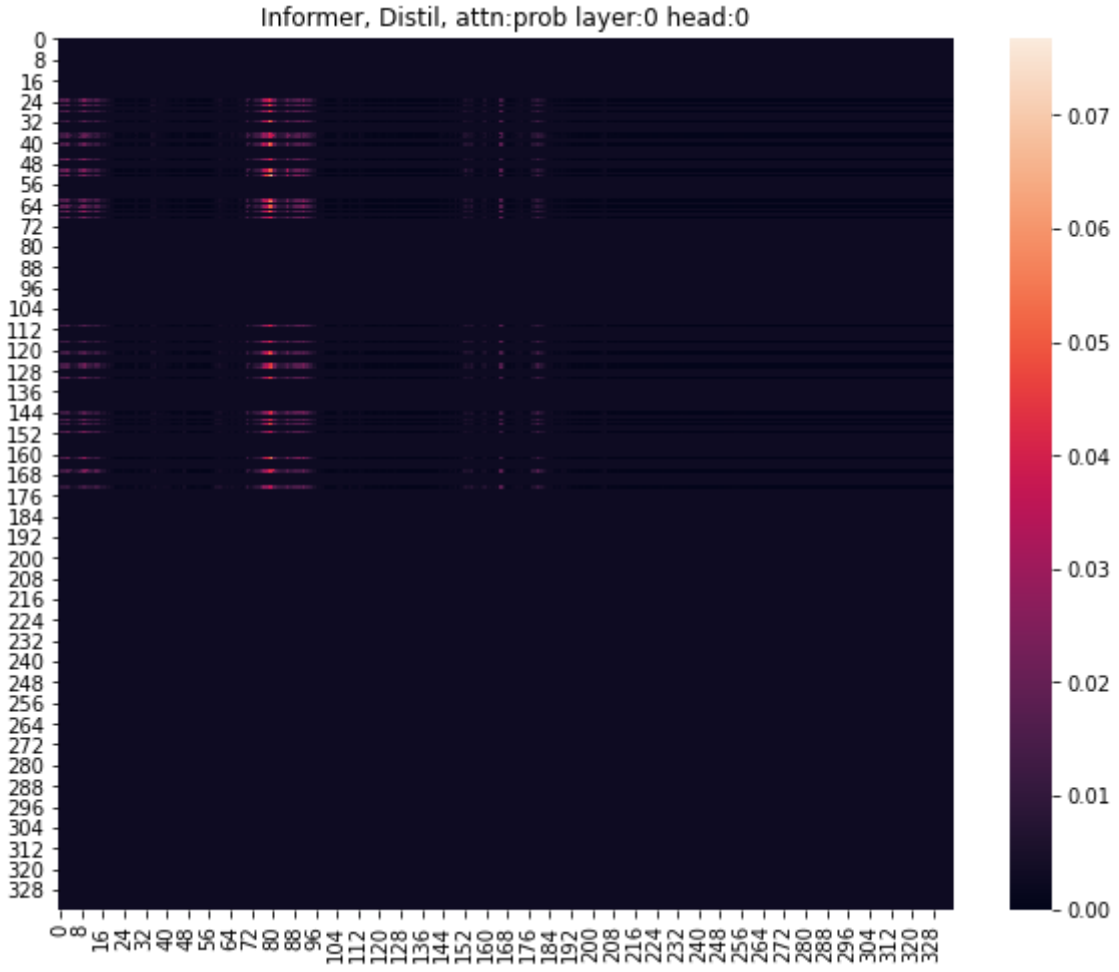

```
# draw HUFL prediction
plt.figure()
plt.plot(trues[0,:,0], label='GroundTruth')
plt.plot(preds[0,:,0], label='Prediction')
plt.legend()
plt.show()
```

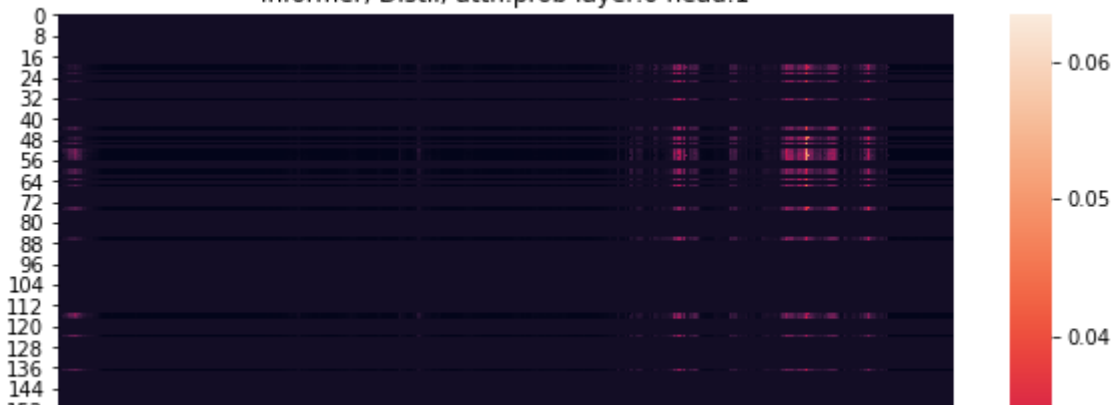


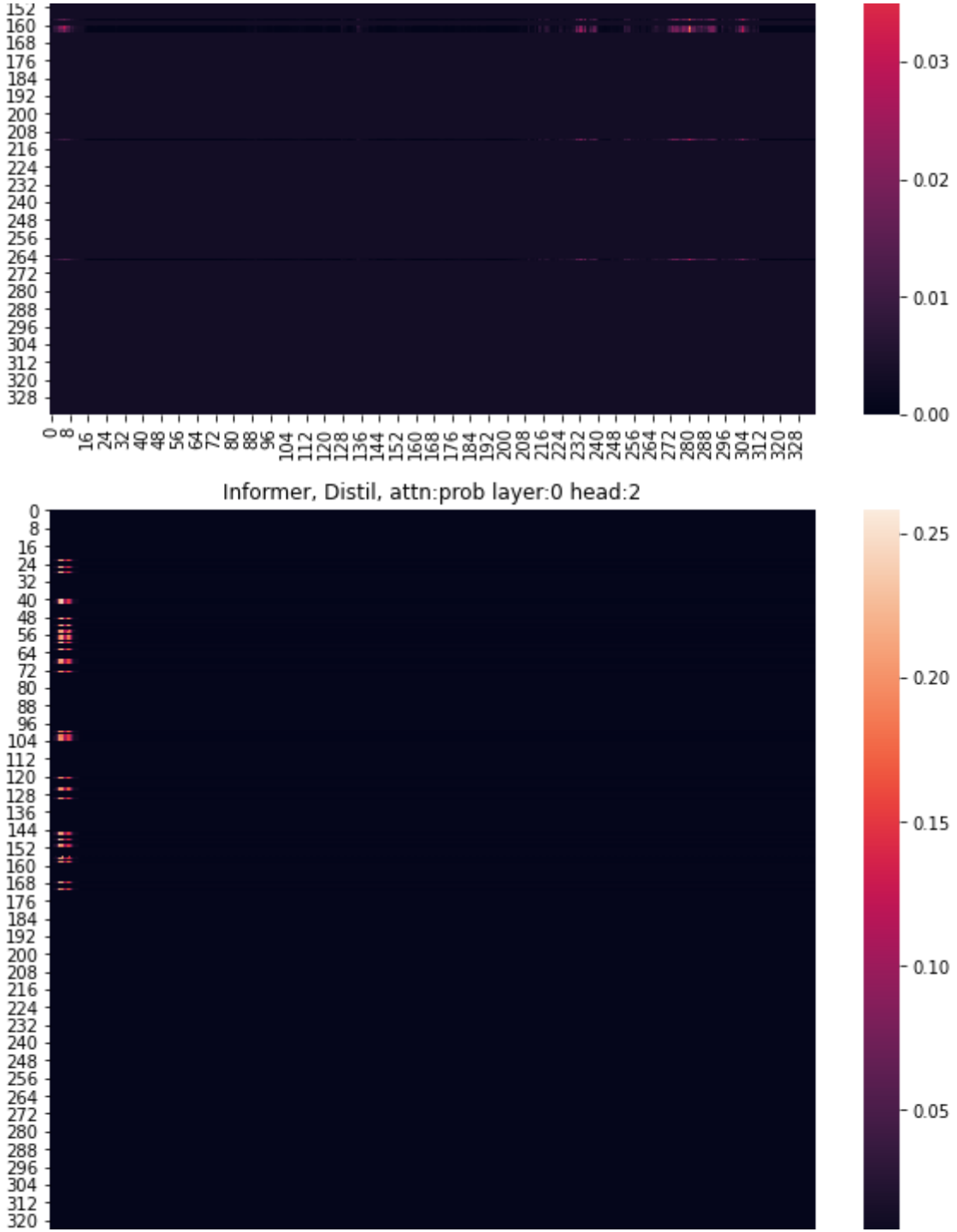

```
from data.data_loader import Dataset_ETT_hour
from torch.utils.data import DataLoader
```

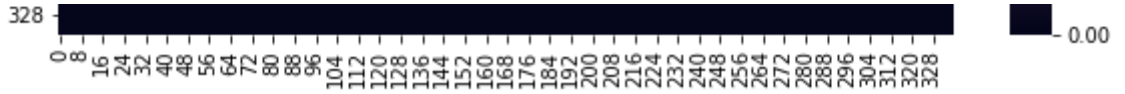
```
Data = Dataset_ETT_hour
timeenc = 0 if args.embed!='timeF' else 1
flag = 'test'; shuffle_flag = False; drop_last = True; batch_size = 1

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,
    freq=args.freq
)
data_loader = DataLoader(
    data_set,
    batch_size=batch_size,
    shuffle=shuffle_flag,
    num_workers=args.num_workers,
    drop_last=drop_last)


import os

args.output_attention = True

exp = Exp(args)

model = exp.model

setting = 'informer_ETTh1_ftS_sl336_ll336_pl168_dm512_nh8_el2_dl1_df1024_atprob_fc5_ebtimeF_dtTrue_exp_0'
path = os.path.join(args.checkpoints,setting,'checkpoint.pth')
model.load_state_dict(torch.load(path))

     Use GPU: cuda:0
     <All keys matched successfully>
```

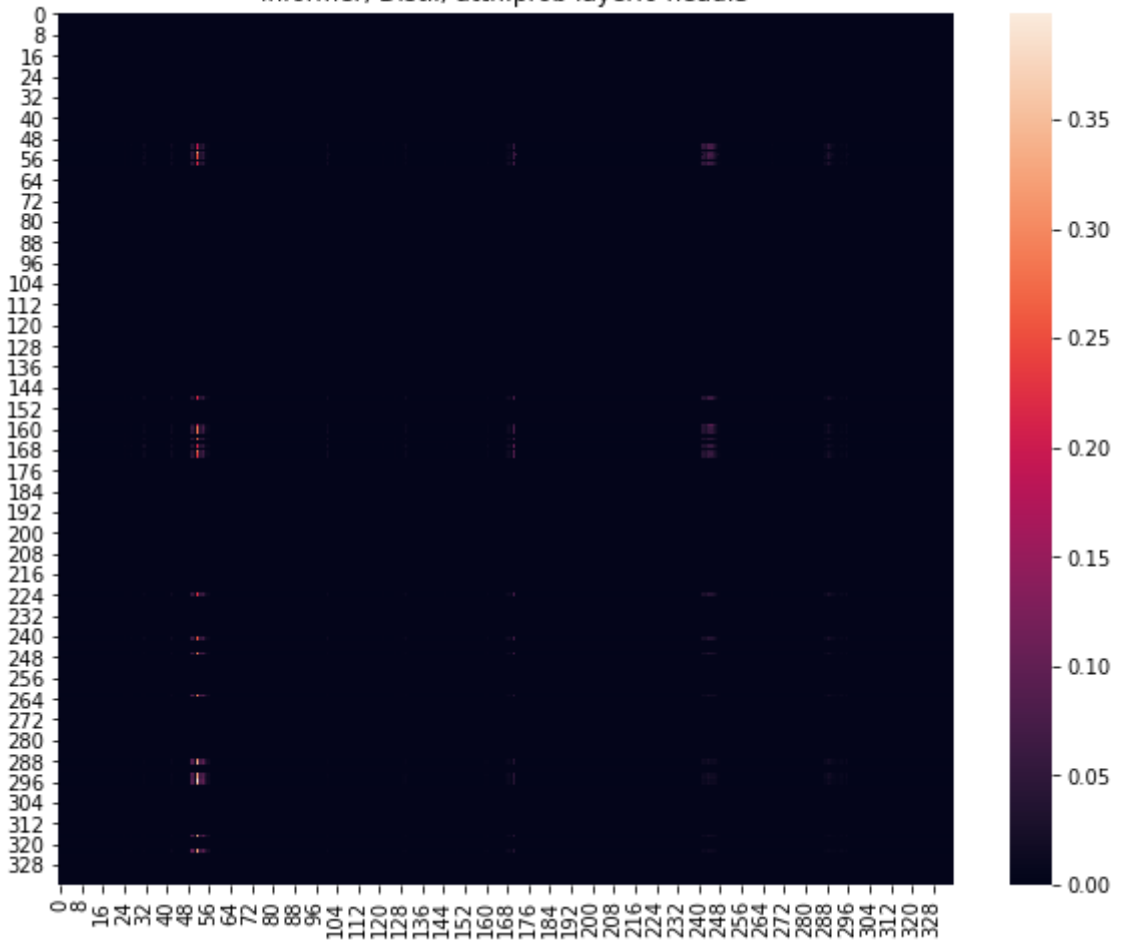
```
# attention visualization
idx = 0
for i, (batch_x,batch_y,batch_x_mark,batch_y_mark) in enumerate(data_loader):
    if i!=idx:
        continue
    batch_x = batch_x.float().to(exp.device)
    batch_y = batch_y.float()

    batch_x_mark = batch_x_mark.float().to(exp.device)
    batch_y_mark = batch_y_mark.float().to(exp.device)

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

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


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

     (torch.Size([1, 8, 336, 336]), torch.Size([1, 8, 169, 169]))


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()
```

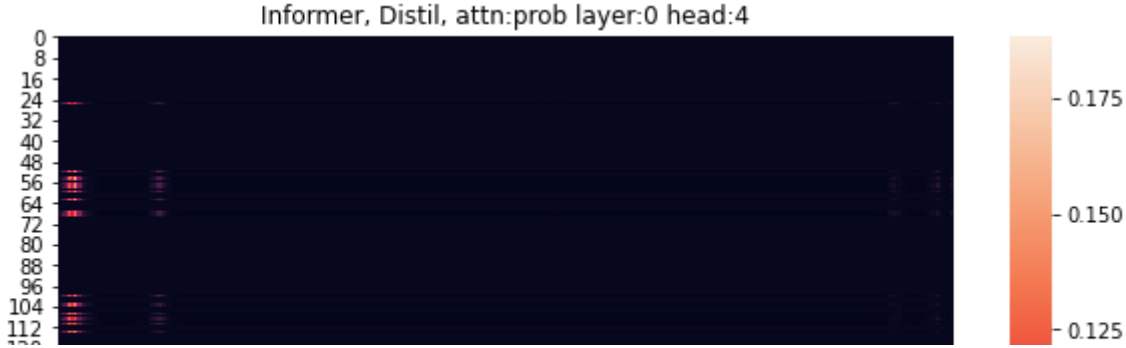

Informer, Distil, attn:prob layer:0 head:0
0 8 16 24 32 40 48 56 64 72 80 88 96 104 112 120 128 136 144 152 160 168 176 184 192 200 208 216 224 232 240 248 256 264 272 280 288 296 304 312 320 328
0.07
0.06
0.05
0.04
0.03
0.02
0.01
0.00

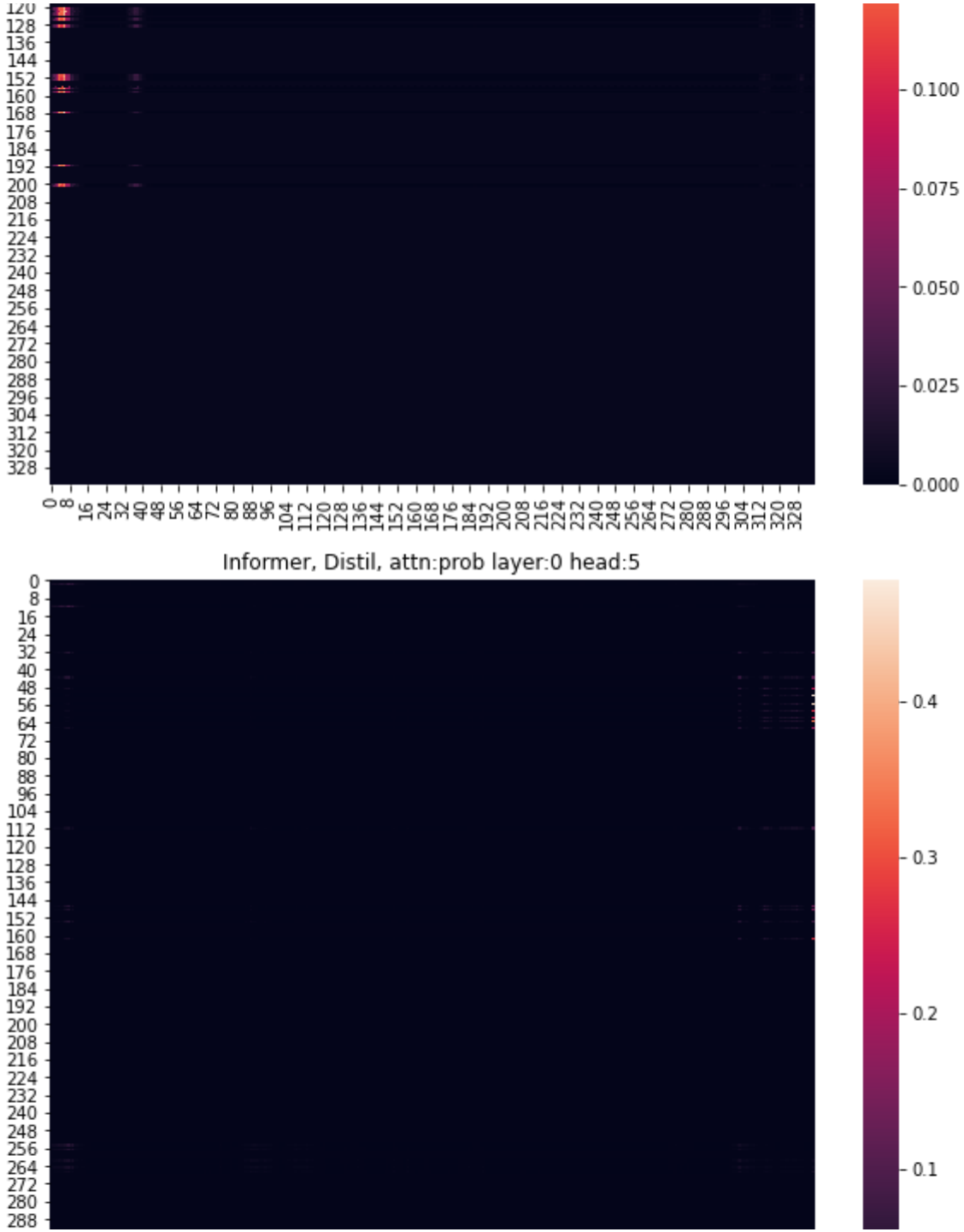

Informer, Distil, attn:prob layer:0 head:1
0 8 16 24 32 40 48 56 64 72 80 88 96 104 112 120 128 136 144
0.06
0.05
0.04

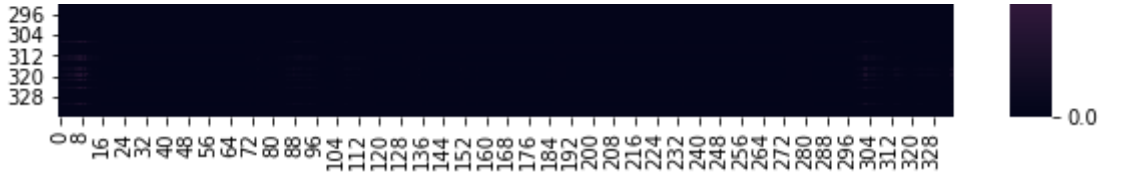

Informer, Distil, attn:prob layer:0 head:2

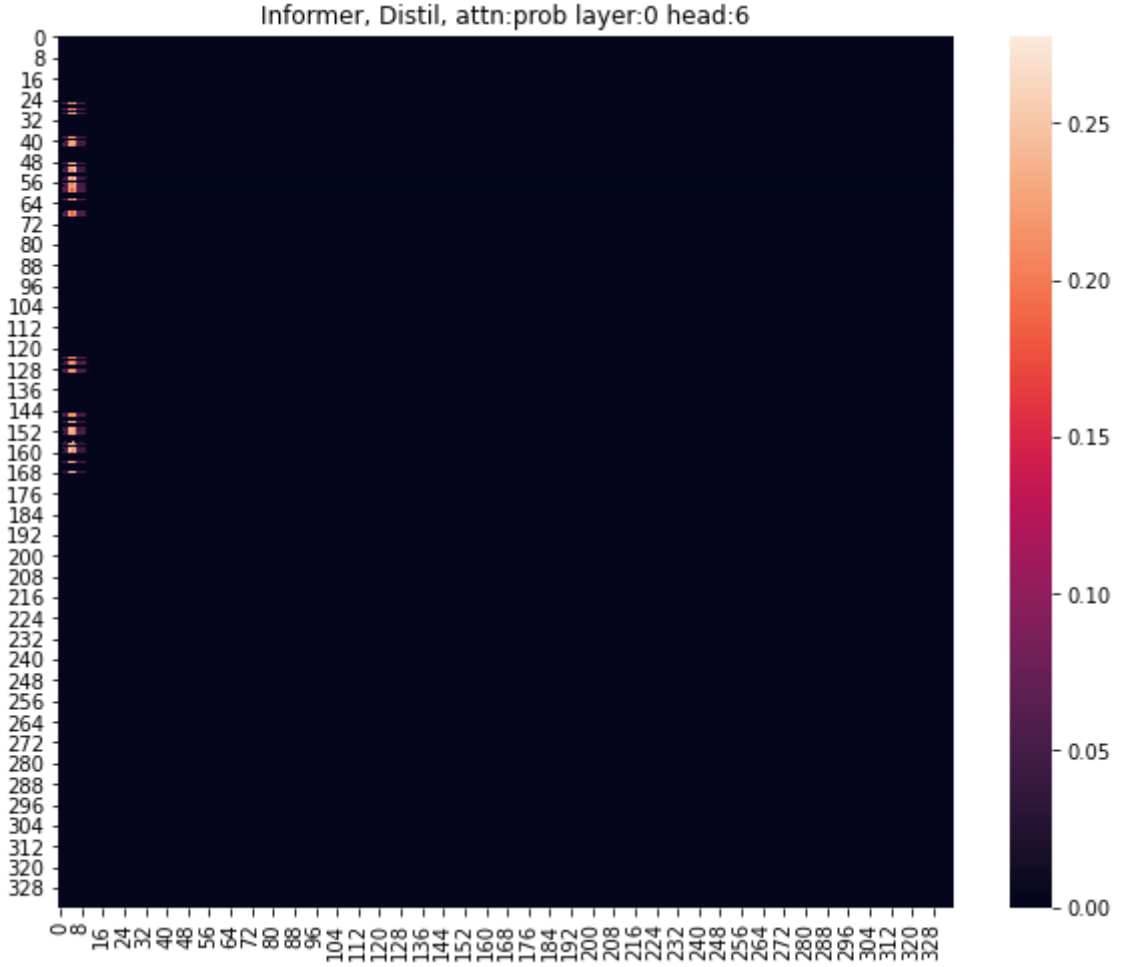

328
0 8 16 24 32 40 48 56 64 72 80 88 96 104 112 120 128 136 144 152 160 168 176 184 192 200 208 216 224 232 240 248 256 264 272 280 288 296 304 312 320 328
0.00

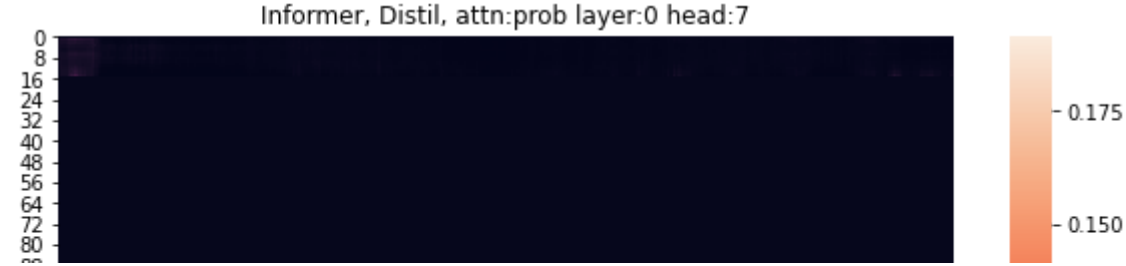

Informer, Distil, attn:prob layer:0 head:3
0 8 16 24 32 40 48 56 64 72 80 88 96 104 112 120 128 136 144 152 160 168 176 184 192 200 208 216 224 232 240 248 256 264 272 280 288 296 304 312 320 328
0 8 16 24 32 40 48 56 64 72 80 88 96 104 112 120 128 136 144 152 160 168 176 184 192 200 208 216 224 232 240 248 256 264 272 280 288 296 304 312 320 328
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00


Informer, Distil, attn:prob layer:0 head:4
0 8 16 24 32 40 48 56 64 72 80 88 96 104 112
0.175
0.150
0.125


Informer, Distil, attn:prob layer:0 head:5


296
304
312
320
328
0
8
16
24
32
40
48
56
64
72
80
88
96
104
112
120
128
136
144
152
160
168
176
184
192
200
208
216
224
232
240
248
256
264
272
280
288
296
304
312
320
328
0.0


Informer, Distil, attn:prob layer:0 head:6
0
8
16
24
32
40
48
56
64
72
80
88
96
104
112
120
128
136
144
152
160
168
176
184
192
200
208
216
224
232
240
248
256
264
272
280
288
296
304
312
320
328
0.25
0.20
0.15
0.10
0.05
0.00


Informer, Distil, attn:prob layer:0 head:7
0
8
16
24
32
40
48
56
64
72
80
0.175
0.150


```
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()
```