run.py

```
import argparse
import os
import torch
from exp.exp_main import Exp_Main
import random
import numpy as np

fix_seed = 2021
random.seed(fix_seed)
torch.manual_seed(fix_seed)
np.random.seed(fix_seed)

parser = argparse.ArgumentParser(description='Autoformer & Transformer family for Time Series
Forecasting')

# basic config
parser.add_argument('--is_training', type=int, required=True, default=1, help='status')
parser.add_argument('--model_id', type=str, required=True, default='test', help='model id')
parser.add_argument('--model', type=str, required=True, default='Autoformer',
help='model name, options: [Autoformer, Informer, Transformer]')

# data loader
parser.add_argument('--data', type=str, required=True, default='EETm1', help='dataset type')
parser.add_argument('--root_path', type=str, default='./data/ETT', help='root path of the data
file')
parser.add_argument('--data_path', type=str, default='dataset4.csv', help='data file')
parser.add_argument('--features', type=str, default='M',
help='forecasting task, options:[M, S, MS]; M:multivariate predict multivariate, S:univariate
predict univariate, MS:multivariate predict univariate')
parser.add_argument('--target', type=str, default='Voltage', help='target feature in S or MS
task')
parser.add_argument('--freq', type=str, default='s',
help='freq for time features encoding, options:[s:secondly, t:minutely, h:hourly, d:daily,
b:business days, w:weekly, m:monthly], you can also use more detailed freq like 15min or 3h')
parser.add_argument('--checkpoints', type=str, default='./checkpoints/', help='location of model
checkpoints')

# forecasting task
parser.add_argument('--seq_len', type=int, default=96, help='input sequence length')
parser.add_argument('--label_len', type=int, default=48, help='start token length')
parser.add_argument('--pred_len', type=int, default=24, help='prediction sequence length')

# model define
parser.add_argument('--enc_in', type=int, default=3, help='encoder input size')
parser.add_argument('--dec_in', type=int, default=3, help='decoder input size')
parser.add_argument('--c_out', type=int, default=3, help='output size')
parser.add_argument('--d_model', type=int, default=512, help='dimension of model')
parser.add_argument('--n_heads', type=int, default=8, help='num of heads')
parser.add_argument('--e_layers', type=int, default=2, help='num of encoder layers')
parser.add_argument('--d_layers', type=int, default=1, help='num of decoder layers')
parser.add_argument('--d_ff', type=int, default=2048, help='dimension of fcn')
parser.add_argument('--moving_avg', type=int, default=25, help='window size of moving average')
parser.add_argument('--factor', type=int, default=5, help='attn factor')
parser.add_argument('--distil', action='store_false',
help='whether to use distilling in encoder, using this argument means not using distilling',
default=True)
parser.add_argument('--dropout', type=float, default=0.05, help='dropout')
parser.add_argument('--embed', type=str, default='timeF',
help='time features encoding, options:[timeF, fixed, learned]')
parser.add_argument('--activation', type=str, default='gelu', help='activation')
parser.add_argument('--output_attention', action='store_true', help='whether to output attention
in ecoder')
parser.add_argument('--do_predict', action='store_true', help='whether to predict unseen future
```

```
data')

# optimization
parser.add_argument('--num_workers', type=int, default=0, help='data loader num workers')
parser.add_argument('--itr', type=int, default=2, help='experiments times')
parser.add_argument('--train_epochs', type=int, default=2, help='train epochs')
parser.add_argument('--batch_size', type=int, default=32, help='batch size of train input data')
parser.add_argument('--patience', type=int, default=10, help='early stopping patience')
parser.add_argument('--learning_rate', type=float, default=0.0001, help='optimizer learning
rate')
parser.add_argument('--des', type=str, default='test', help='exp description')
parser.add_argument('--loss', type=str, default='mse', help='loss function')
parser.add_argument('--lradj', type=str, default='type1', help='adjust learning rate')
parser.add_argument('--use_amp', action='store_true', help='use automatic mixed precision
training', default=False)
parser.add_argument('--inverse', action='store_true', help='inverse output data', default=False)

# GPU
parser.add_argument('--use_gpu', type=bool, default=True, help='use gpu')
parser.add_argument('--gpu', type=int, default=0, help='gpu')
parser.add_argument('--use_multi_gpu', action='store_true', help='use multiple gpus',
default=False)
parser.add_argument('--devices', type=str, default='0,1,2,3', help='device ids of multile gpus')

args = parser.parse_args()

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]

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

Exp = Exp_Main

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

exp = Exp(args) # set experiments
print('>>>>>>>start training : {}>>>>>>>>>>>>>>>>>>>>>>>>>>'.format(setting))
exp.train(setting)

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

```
if args.do_predict:
print('>>>>>>>predicting : {}<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<'.format(setting))
exp.predict(setting, True)

torch.cuda.empty_cache()
else:
ii = 0
setting =
'{}_{}_{}_ft{}_sl{}_ll{}_pl{}_dm{}_nh{}_el{}_dl{}_df{}_fc{}_eb{}_dt{}_{}_{}'.format(args.model_id,
args.model,
args.data,
args.features,
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.factor,
args.embed,
args.distil,
args.des, ii)

exp = Exp(args) # set experiments
print('>>>>>>>testing : {}<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<'.format(setting))
exp.test(setting, test=1)
torch.cuda.empty_cache()
```