```
"""
Created on Mon Mar 01 14:47:27 2024
@author: Bing
The core code of the LBDSAC algorithm
"""
import os
os.environ["CUDA_VISIBLE_DEVICES"] = '0'
import tensorflow as tf
import shutil
import numpy as np
import pandas as pd
import time
import datetime
import seaborn as sns
from logx import EpochLogger
from mpi_tf import sync_all_params, MpiAdamOptimizer
from mpi_tools import mpi_fork, mpi_sum, proc_id, mpi_statistics_scalar, num_procs
from scipy.stats import norm
import globalvar as gl
import json
import argparse
from run_utils import setup_logger_kwargs
import save_data
import matplotlib.pyplot as plt
from env2 import unPower
from model_modules import placeholder,placeholders,cnn_mlp,get_vars,count_vars,gaussian_likelihood,\
    get_target_update,mlp_gaussian_policy,apply_squashing_func,mlp_actor,mlp_critic,Fig
import ReplayBuffer

n_T = gl.get_value('n_T')
n_wind = gl.get_value('n_wind')
n_load = gl.get_value('n_load')
n_gen = gl.get_value('n_gen')
n_wind_data = gl.get_value('n_wind_data')
obs_dim = gl.get_value('state_dim')
act_dim = gl.get_value('action_dim')
P_load = gl.get_value('P_load')
P_wind = gl.get_value('P_wind')
load_capacity = gl.get_value('load_capacity')
wind_capacity = gl.get_value('wind_capacity')
EPS = 1e-8

snow = datetime.datetime.now()
seed_now = str(snow.month)+str(snow.day)+str(snow.hour)
seed_now = int(seed_now)

parser = argparse.ArgumentParser()
parser.add_argument('--Train', type=bool, default=True, help='True or False')
parser.add_argument('--Continue', type=bool, default=False, help='True or False')
parser.add_argument('--Portion', type=bool, default=False, help='True or False')
parser.add_argument('--SaveFig', type=bool, default=False, help='True or False')
parser.add_argument('--load_buffer', type=bool, default=False, help='True or False')
parser.add_argument('--Model_version', type=int, default=8000, help='8000')
parser.add_argument('--epochs', type=int, default=8000, help='800')
parser.add_argument('--steps_per_epoch', type=int, default=20, help='20')
parser.add_argument('--update_freq', type=int, default=720, help='720')
parser.add_argument('--save_freq', type=int, default=1000,  help='500')
parser.add_argument('--cost_lim', type=float, default=1e-3, help='1e-3')
parser.add_argument('--lr', type=float, default=1e-3,help='1e-3')
parser.add_argument('--hidden_sizes_actor', type=list, default=[128, 256, 256, 64], help='actor')
parser.add_argument('--hidden_sizes_critic', type=list, default=[128, 256, 64], help='critic')
parser.add_argument('--hidden_sizes_var', type=list, default=[128, 256, 64], help='var')
parser.add_argument('--cnn_actor', type=list, default=[], help='CNNs of policy network actor')
parser.add_argument('--cnn_critic', type=list, default=[], help='CNNs of  network critic')
parser.add_argument('--cnn_var', type=list, default=[], help='CNNs of network var')
parser.add_argument('--hid', type=int, default=512)
parser.add_argument('--l', type=int, default=2)
parser.add_argument('--gamma', type=float, default=0.99)
parser.add_argument('--seed', '-s', type=int, default=seed_now)
parser.add_argument('--exp_name', type=str, default='sac')
parser.add_argument('--cpu', type=int, default=1)
parser.add_argument('--render', default=False, action='store_true')
parser.add_argument('--local_start_steps', default=3600, type=int, help='3600')
parser.add_argument('--local_update_after', default=3600, type=int, help='3600')
parser.add_argument('--batch_size', default=256, type=int)  # 256
parser.add_argument('--fixed_entropy_bonus', default=None, type=float)
parser.add_argument('--entropy_constraint', type=float, default=-1)
parser.add_argument('--fixed_cost_penalty', default=None, type=float)
parser.add_argument('--cost_constraint', type=float, default=None)
parser.add_argument('--lr_s', type=int, default=50)
parser.add_argument('--damp_s', type=int, default=10)
parser.add_argument('--logger_kwargs_str', type=json.loads, default='{"output_dir": "./logger"}')
args = parser.parse_args()

"""
Soft Actor-Critic with Logic-Based Benders Decomposition Algorithm
"""

```

```
def LBDSAC(env_fn, actor_fn=mlp_actor, critic_fn=mlp_critic,
        ac_kwargs_actor=dict(),ac_kwargs_critic=dict(),ac_kwargs_var=dict(),
        seed=0, steps_per_epoch=1200, epochs=101, replay_size=500 * 2 * 720, gamma=0.99, cl=0.5,
        polyak=0.995, lr=1e-4, batch_size=1024, local_start_steps=600,
        max_ep_len=n_T, logger_kwargs=dict(), save_freq=50, local_update_after=int(1e3),
        update_freq=120, render=False,
        fixed_entropy_bonus=None, entropy_constraint=-1.0,
        fixed_cost_penalty=None, cost_constraint=None, cost_lim=None,
        reward_scale=1, lr_scale=1, damp_scale=0,Train=True,Continue_Training=False,
        Model_Portion=False,load_buffer=True
        ):

    use_costs = fixed_cost_penalty or cost_constraint or cost_lim
    logger = EpochLogger(**logger_kwargs)
    logger.save_config(locals())

    # Env instantiation
    env, test_env = env_fn(), env_fn()
    #Setting seeds
    seed += 200 * proc_id()
    tf.set_random_seed(seed)
    np.random.seed(seed)

    # Inputs to computation graph
    x_ph, a_ph, x2_ph, r_ph, d_ph, c1_ph, c2_ph = placeholders(obs_dim, act_dim, obs_dim, None, None, None, None)

    # Main outputs from computation graph
    with tf.variable_scope('main'):
        mu, pi, logp_pi = actor_fn(x_ph, a_ph, **ac_kwargs_actor)
        qr1, qr1_pi = critic_fn(x_ph, a_ph, pi, name='qr1', **ac_kwargs_critic)
        qr2, qr2_pi = critic_fn(x_ph, a_ph, pi, name='qr2', **ac_kwargs_critic)
        qc1, qc1_pi = critic_fn(x_ph, a_ph, pi, name='qc1', **ac_kwargs_critic)
        qc2, qc2_pi = critic_fn(x_ph, a_ph, pi, name='qc2', **ac_kwargs_critic)

    with tf.variable_scope('main', reuse=True):
        # Additional policy output from a different observation placeholder
        _, pi2, logp_pi2 = actor_fn(x2_ph, a_ph, **ac_kwargs_actor)

    # Target value network
    with tf.variable_scope('target'):
        _, qr1_pi_targ = critic_fn(x2_ph, a_ph, pi2, name='qr1', **ac_kwargs_critic)
        _, qr2_pi_targ = critic_fn(x2_ph, a_ph, pi2, name='qr2', **ac_kwargs_critic)
        _, qc1_pi_targ = critic_fn(x2_ph, a_ph, pi2, name='qc1', **ac_kwargs_critic)
        _, qc2_pi_targ = critic_fn(x2_ph, a_ph, pi2, name='qc2', **ac_kwargs_critic)

    # Entropy bonus
    if fixed_entropy_bonus is None:
        with tf.variable_scope('entreg'):
            soft_alpha = tf.get_variable('soft_alpha',
                                         initializer=0.0,
                                         trainable=True,
                                         dtype=tf.float32)
        alpha = tf.nn.softplus(soft_alpha)
    else:
        alpha = tf.constant(fixed_entropy_bonus)
    log_alpha = tf.log(tf.clip_by_value(alpha, 1e-8, 1e8))

    # Cost penalty
    if use_costs:
        if fixed_cost_penalty is None:
            with tf.variable_scope('costpen1'):
                soft_beta1 = tf.get_variable('soft_beta1',
                                             initializer=0.0,
                                             trainable=True,
                                             dtype=tf.float32)
            beta1 = tf.nn.softplus(soft_beta1)
            log_beta1 = tf.log(tf.clip_by_value(beta1, 1e-8, 1e8))
        else:
            beta1 = tf.constant(fixed_cost_penalty)
            log_beta1 = tf.log(tf.clip_by_value(beta1, 1e-8, 1e8))
    else:
        beta1 = 0.0  # costs do not contribute to policy optimization
        print('Not using costs')

    if use_costs:
        if fixed_cost_penalty is None:
            with tf.variable_scope('costpen2'):
                soft_beta2 = tf.get_variable('soft_beta2',
                                             initializer=0.0,
                                             trainable=True,
                                             dtype=tf.float32)
            beta2 = tf.nn.softplus(soft_beta2)
            log_beta2 = tf.log(tf.clip_by_value(beta2, 1e-8, 1e8))
        else:
            beta2 = tf.constant(fixed_cost_penalty)
            log_beta2 = tf.log(tf.clip_by_value(beta2, 1e-8, 1e8))
    else:
        beta2 = 0.0  # costs do not contribute to policy optimization
        print('Not using costs')

    # Experience buffer
    replay_buffer = ReplayBuffer(obs_dim=obs_dim, act_dim=act_dim, size=replay_size, load_buffer=load_buffer)

    # Count variables
    if proc_id() == 0:
        var_counts = tuple(count_vars(scope) for scope in
                           ['main/pi', 'main/qr1', 'main/qr2', 'main/qc1', 'main/qc2', 'main'])
        print((
                  '\nNumber of parameters: \t pi: %d, \t qr1: %d, \t qr2: %d, \t qc1: %d, \t qc2: %d, \t total: %d\n') % var_counts)
    # Min Double-Q:
    min_q_pi = tf.minimum(qr1_pi, qr2_pi)
    min_q_pi_targ = tf.minimum(qr1_pi_targ, qr2_pi_targ)

    # Targets for Q and V regression
    q_backup = tf.stop_gradient(r_ph + gamma * (1 - d_ph) * (min_q_pi_targ - alpha * logp_pi2))
    qc1_backup = tf.stop_gradient(c1_ph + gamma * (1 - d_ph) * qc1_pi_targ)
    qc2_backup = tf.stop_gradient(c2_ph + gamma * (1 - d_ph) * qc2_pi_targ)

    cost_constraint = cost_lim * (1 - gamma ** max_ep_len) / (1 - gamma) / max_ep_len
    damp1 = damp_scale * tf.reduce_mean(cost_constraint - qc1)
    damp2 = damp_scale * tf.reduce_mean(cost_constraint - qc2)

    # LBDSAC losses
    pi_loss = tf.reduce_mean(alpha * logp_pi - min_q_pi + (beta1 - damp1) * qc1_pi + (beta2 - damp2) * qc2_pi)
    qr1_loss = 0.5 * tf.reduce_mean((q_backup - qr1) ** 2)
    qr2_loss = 0.5 * tf.reduce_mean((q_backup - qr2) ** 2)
    qc1_loss = 0.5 * tf.reduce_mean((qc1_backup - qc1) ** 2)
    qc2_loss = 0.5 * tf.reduce_mean((qc2_backup - qc2) ** 2)
    q_loss = qr1_loss + qr2_loss + qc1_loss + qc2_loss

    # Loss for alpha
    entropy_constraint *= act_dim
    pi_entropy = -tf.reduce_mean(logp_pi)
    alpha_loss = - alpha * (entropy_constraint - pi_entropy)
    print('using entropy constraint', entropy_constraint)

    # Loss for beta
    if use_costs:
        if cost_constraint is None:
            cost_constraint = cost_lim * (1 - gamma ** max_ep_len) / (1 - gamma) / max_ep_len
        print('using cost constraint', cost_constraint)
        beta1_loss = beta1 * (cost_constraint - qc1)
        beta2_loss = beta2 * (cost_constraint - qc2)

    # Policy train op
    train_pi_op = MpiAdamOptimizer(learning_rate=lr).minimize(pi_loss, var_list=get_vars('main/pi'), name='train_pi')

    # Value train op
    with tf.control_dependencies([train_pi_op]):
        train_q_op = MpiAdamOptimizer(learning_rate=lr).minimize(q_loss, var_list=get_vars('main/q'), name='train_q')

    if fixed_entropy_bonus is None:
        entreg_optimizer = MpiAdamOptimizer(learning_rate=lr)
        with tf.control_dependencies([train_q_op]):
            train_entreg_op = entreg_optimizer.minimize(alpha_loss, var_list=get_vars('entreg'))

    if use_costs and fixed_cost_penalty is None:
        costpen_optimizer = MpiAdamOptimizer(learning_rate=lr * lr_scale)

        if fixed_entropy_bonus is None:
            with tf.control_dependencies([train_entreg_op]):
                train_costpen1_op = costpen_optimizer.minimize(beta1_loss, var_list=get_vars('costpen1'))
        else:
            with tf.control_dependencies([train_q_op]):
                train_costpen1_op = costpen_optimizer.minimize(beta1_loss, var_list=get_vars('costpen1'))

        if fixed_entropy_bonus is None:
            with tf.control_dependencies([train_entreg_op]):
                train_costpen2_op = costpen_optimizer.minimize(beta2_loss, var_list=get_vars('costpen2'))
        else:
            with tf.control_dependencies([train_q_op]):
                train_costpen2_op = costpen_optimizer.minimize(beta2_loss, var_list=get_vars('costpen2'))

    target_update = get_target_update('main', 'target', polyak)

    with tf.control_dependencies([train_pi_op]):
        with tf.control_dependencies([train_q_op]):
            grouped_update = tf.group([target_update])

    if fixed_entropy_bonus is None:
        grouped_update = tf.group([grouped_update, train_entreg_op])
    if use_costs and fixed_cost_penalty is None:
        grouped_update = tf.group([grouped_update, train_costpen1_op])
        grouped_update = tf.group([grouped_update, train_costpen2_op])

    def get_action(o, deterministic=False):
        act_op = mu if deterministic else pi
        return sess.run(act_op, feed_dict={x_ph: o.reshape(1, -1)})[0]

    config = tf.ConfigProto()
    config.gpu_options.per_process_gpu_memory_fraction = 0.8
    sess = tf.Session(config=config)

    writer = tf.summary.FileWriter(".logs/",sess.graph)
    # Initializing targets to match main variables
    target_init = get_target_update('main', 'target', 0.0)
    sess.run(tf.global_variables_initializer())
    sess.run(target_init)
    sess.run(sync_all_params())
    # Setup model saving
    logger.setup_tf_saver(sess, inputs={'x': x_ph, 'a': a_ph},
                          outputs={'mu': mu, 'pi': pi, 'qr1': qr1, 'qr2': qr2, 'qc1': qc1, 'qc2': qc2})
    start_time = time.time()
    o, r, d, ep_ret, ep_cost1, ep_cost2, ep_len= env.reset(), 0, False, 0, 0, 0, 0
    total_steps = steps_per_epoch * epochs * max_ep_len

    # variables to measure in an update
    vars_to_get = dict(LossPi=pi_loss, LossQR1=qr1_loss, LossQR2=qr2_loss, LossQC1=qc1_loss, LossQC2=qc2_loss,
                       QR1Vals=qr1, QR2Vals=qr2, QC1Vals=qc1, QC2Vals=qc2, LogPi=logp_pi, PiEntropy=pi_entropy,
                       Alpha=alpha, LogAlpha=log_alpha, LossAlpha=alpha_loss)
    if use_costs:
        vars_to_get.update(dict(Beta1=beta1, LogBeta1=log_beta1, LossBeta1=beta1_loss,
                                Beta2=beta2, LogBeta2=log_beta2, LossBeta2=beta2_loss,))

    print('starting training', proc_id())
    cum_cost1 = 0
    cum_cost2 = 0
    local_steps = 0
    local_steps_per_epoch = steps_per_epoch // num_procs()
    local_batch_size = batch_size // num_procs()
    epoch_start_time = time.time()
    for t in range(total_steps // num_procs()):
        if t > local_start_steps:
            a = get_action(o)
        else:
```

```
    a = env.action_sample()
# Step the env
o2, r, c1, c2, d = env.step(a,o,ep_len)
r *= reward_scale  # yee-haw
ep_ret += r
ep_cost1 += c1
ep_cost2 += c2
ep_len += 1
local_steps += 1
cum_cost1 += c1
cum_cost2 += c2
replay_buffer.store(o, a, r, o2, d, c1, c2)
o = o2
if d or (ep_len == max_ep_len):
    logger.store(EpRet=ep_ret, EpCost1=ep_cost1, EpCost2=ep_cost2)
    o, r, d, ep_ret, ep_cost1, ep_cost2, ep_len = env.reset(n = np.random.randint(n_wind_data)),\
        0, False, 0, 0, 0, 0

if t > 0 and t % update_freq == 0:
    for j in range(200):
        batch = replay_buffer.sample_batch(local_batch_size)
        feed_dict = {x_ph: batch['obs1'],
                     x2_ph: batch['obs2'],
                     a_ph: batch['acts'],
                     r_ph: batch['rews'],
                     c1_ph: batch['costs1'],
                     c2_ph: batch['costs2'],
                     d_ph: batch['done'],
                     }
        if t < local_update_after:
            logger.store(**sess.run(vars_to_get, feed_dict))
        else:
            values, _ = sess.run([vars_to_get, grouped_update], feed_dict)
            logger.store(**values)
    ETA = (time.time()-epoch_start_time)*(total_steps//update_freq-t//update_freq)
    print(('Training: [epoch:%d|%d] [batch:%d|%d] ETA %d:%d:%d')\
      % (t // (local_steps_per_epoch * max_ep_len),epochs-1,\
          (t%(local_steps_per_epoch * max_ep_len)//max_ep_len),steps_per_epoch-1,\
          ETA//3600, ETA%3600//60, ETA%60))
    epoch_start_time = time.time()

# End of epoch wrap-up
if t > 0 and t % (local_steps_per_epoch * max_ep_len) == 0:
    epoch = t // (local_steps_per_epoch * max_ep_len)
```

```python
        cumulative_cost1 = mpi_sum(cum_cost1)
        cumulative_cost2 = mpi_sum(cum_cost2)
        cost_rate1 = cumulative_cost1 / ((epoch + 1) * steps_per_epoch)
        cost_rate2 = cumulative_cost2 / ((epoch + 1) * steps_per_epoch)

        # Save model
        if (epoch % save_freq == 0) or (epoch == epochs - 1):
            saver = tf.train.Saver()
            saver.save(sess,'./saved_models/Case118_%d/Model'%epoch)
        logger.log_tabular('Epoch', epoch)
        logger.log_tabular('EpRet', with_min_and_max=True)
        logger.log_tabular('EpCost1', with_min_and_max=True)
        logger.log_tabular('EpCost2', with_min_and_max=True)
        logger.log_tabular('CumulativeCost1', cumulative_cost1)
        logger.log_tabular('CumulativeCost2', cumulative_cost2)
        logger.log_tabular('CostRate1', cost_rate1)
        logger.log_tabular('CostRate2', cost_rate2)
        logger.log_tabular('LossPi', with_min_and_max=True)
        logger.log_tabular('LossQR1', with_min_and_max=True)
        logger.log_tabular('LossQC1', with_min_and_max=True)
        logger.log_tabular('LossQC2', with_min_and_max=True)
        logger.log_tabular('PiEntropy', with_min_and_max=True)
        logger.log_tabular('TotalTime', time.time() - start_time)
        logger.dump_tabular()
    writer.close()
```