```
#!/usr/bin/env python
"""CEVAE model
"""

import edward as ed
import tensorflow as tf
import sys

from edward.models import Bernoulli, Normal
from progressbar import ETA, Bar, Percentage, ProgressBar

import numpy as np
import time
from scipy.stats import sem
from sklearn.model_selection import train_test_split
from argparse import ArgumentParser

from tensorflow.contrib.layers.python.layers import initializers
from tensorflow.contrib import slim

from generate_data import gen_lrmf, ampute, gen_dlvm


def fc_net(inp, layers, out_layers, scope, lamba=1e-3, activation=tf.nn.relu, reuse=None,
           weights_initializer=initializers.xavier_initializer(uniform=False)):
    # utils for cevea
    with slim.arg_scope([slim.fully_connected],
                        activation_fn=activation,
                        normalizer_fn=None,
                        weights_initializer=weights_initializer,
                        reuse=reuse,
                        weights_regularizer=slim.l2_regularizer(lamba)):

        if layers:
            h = slim.stack(inp, slim.fully_connected, layers, scope=scope)
            if not out_layers:
                return h
        else:
            h = inp
        outputs = []
        for i, (outdim, activation) in enumerate(out_layers):
            o1 = slim.fully_connected(h, outdim, activation_fn=activation, scope=scope +
'_{}'.format(i + 1))
            outputs.append(o1)
        return outputs if len(outputs) > 1 else outputs[0]


def get_y0_y1(sess, y, f0, f1, shape=(), L=1, verbose=True):
    # utils for cevea
    y0, y1 = np.zeros(shape, dtype=np.float32), np.zeros(shape, dtype=np.float32)
    ymean = y.mean()
    for l in range(L):
        if L > 1 and verbose:
            sys.stdout.write('\r Sample {}/{}'.format(l + 1, L))
            sys.stdout.flush()
        y0 += sess.run(ymean, feed_dict=f0) / L
        y1 += sess.run(ymean, feed_dict=f1) / L

    if L > 1 and verbose:
        print()
    return y0, y1


def cevae_tf(X, T, Y, n_epochs=100, early_stop = 10, d_cevea=20):

    T, Y = T.reshape((-1,1)), Y.reshape((-1,1))
    args = dict()
    args['earl'] = early_stop
    args['lr'] = 0.001
    args['opt'] = 'adam'
```

```
    args['epochs'] = n_epochs
    args['print_every'] = 10
    args['true_post'] = True

    M = None  # batch size during training
    d = d_cevea  # latent dimension
    lamba = 1e-4  # weight decay
    nh, h = 3, 200  # number and size of hidden layers

    contfeats = list(range(X.shape[1])) # all continuous
    binfeats = []

    # need for early stopping
    xtr, xva, ttr, tva, ytr, yva = train_test_split(X, T, Y)

    # zero mean, unit variance for y during training
    ym, ys = np.mean(Y), np.std(Y)
    ytr, yva = (ytr - ym) / ys, (yva - ym) / ys
    best_logpvalid = - np.inf

    with tf.Graph().as_default():
        sess = tf.InteractiveSession()

        ed.set_seed(1)
        np.random.seed(1)
        tf.set_random_seed(1)

        # x_ph_bin = tf.placeholder(tf.float32, [M, len(binfeats)], name='x_bin')  # bina
ry inputs
        x_ph_cont = tf.placeholder(tf.float32, [M, len(contfeats)], name='x_cont')  # con
tinuous inputs
        t_ph = tf.placeholder(tf.float32, [M, 1])
        y_ph = tf.placeholder(tf.float32, [M, 1])

        # x_ph = tf.concat([x_ph_bin, x_ph_cont], 1)
        x_ph = x_ph_cont
        activation = tf.nn.elu

        # CEVAE model (decoder)
        # p(z)
        z = Normal(loc=tf.zeros([tf.shape(x_ph)[0], d]), scale=tf.ones([tf.shape(x_ph)[0]
, d]))

        # p(x|z)
        hx = fc_net(z, (nh - 1) * [h], [], 'px_z_shared', lamba=lamba, activation=activat
ion)
        # logits = fc_net(hx, [h], [[len(binfeats), None]], 'px_z_bin', lamba=lamba, acti
vation=activation)
        # x1 = Bernoulli(logits=logits, dtype=tf.float32, name='bernoulli_px_z')

        mu, sigma = fc_net(hx, [h], [[len(contfeats), None], [len(contfeats), tf.nn.softp
lus]], 'px_z_cont', lamba=lamba,
                           activation=activation)
        x2 = Normal(loc=mu, scale=sigma, name='gaussian_px_z')

        # p(t|z)
        logits = fc_net(z, [h], [[1, None]], 'pt_z', lamba=lamba, activation=activation)
        t = Bernoulli(logits=logits, dtype=tf.float32)

        # p(y|t,z)
        mu2_t0 = fc_net(z, nh * [h], [[1, None]], 'py_t0z', lamba=lamba, activation=activ
ation)
        mu2_t1 = fc_net(z, nh * [h], [[1, None]], 'py_t1z', lamba=lamba, activation=activ
ation)
        y = Normal(loc=t * mu2_t1 + (1. - t) * mu2_t0, scale=tf.ones_like(mu2_t0))

        # CEVAE variational approximation (encoder)
        # q(t|x)
        logits_t = fc_net(x_ph, [d], [[1, None]], 'qt', lamba=lamba, activation=activatio
n)
```

```
        qt = Bernoulli(logits=logits_t, dtype=tf.float32)
        # q(y|x,t)
        hqy = fc_net(x_ph, (nh - 1) * [h], [], 'qy_xt_shared', lamba=lamba, activation=ac
tivation)
        mu_qy_t0 = fc_net(hqy, [h], [[1, None]], 'qy_xt0', lamba=lamba, activation=activa
tion)
        mu_qy_t1 = fc_net(hqy, [h], [[1, None]], 'qy_xt1', lamba=lamba, activation=activa
tion)
        qy = Normal(loc=qt * mu_qy_t1 + (1. - qt) * mu_qy_t0, scale=tf.ones_like(mu_qy_t0
))
        # q(z|x,t,y)
        inpt2 = tf.concat([x_ph, qy], 1)
        hqz = fc_net(inpt2, (nh - 1) * [h], [], 'qz_xty_shared', lamba=lamba, activation=
activation)
        muq_t0, sigmaq_t0 = fc_net(hqz, [h], [[d, None], [d, tf.nn.softplus]], 'qz_xt0',
lamba=lamba,
                                   activation=activation)
        muq_t1, sigmaq_t1 = fc_net(hqz, [h], [[d, None], [d, tf.nn.softplus]], 'qz_xt1',
lamba=lamba,
                                   activation=activation)
        qz = Normal(loc=qt * muq_t1 + (1. - qt) * muq_t0, scale=qt * sigmaq_t1 + (1. - qt
) * sigmaq_t0)

        # Create data dictionary for edward
        data = {x2: x_ph_cont, y: y_ph, qt: t_ph, t: t_ph, qy: y_ph}

        # sample posterior predictive for p(y|z,t)
        y_post = ed.copy(y, {z: qz, t: t_ph}, scope='y_post')
        # crude approximation of the above
        y_post_mean = ed.copy(y, {z: qz.mean(), t: t_ph}, scope='y_post_mean')
        # construct a deterministic version (i.e. use the mean of the approximate posteri
or) of the lower bound
        # for early stopping according to a validation set
        y_post_eval = ed.copy(y, {z: qz.mean(), qt: t_ph, qy: y_ph, t: t_ph}, scope='y_po
st_eval')
        # x1_post_eval = ed.copy(x1, {z: qz.mean(), qt: t_ph, qy: y_ph}, scope='x1_post_e
val')
        x2_post_eval = ed.copy(x2, {z: qz.mean(), qt: t_ph, qy: y_ph}, scope='x2_post_eva
l')
        t_post_eval = ed.copy(t, {z: qz.mean(), qt: t_ph, qy: y_ph}, scope='t_post_eval')
        logp_valid = tf.reduce_mean(tf.reduce_sum(y_post_eval.log_prob(y_ph) + t_post_eva
l.log_prob(t_ph), axis=1) +
                                    tf.reduce_sum(x2_post_eval.log_prob(x_ph_cont), axis=
1) +
                                    tf.reduce_sum(z.log_prob(qz.mean()) - qz.log_prob(qz.
mean()), axis=1))

        inference = ed.KLqp({z: qz}, data)
        optimizer = tf.train.AdamOptimizer(learning_rate=args['lr'])
        inference.initialize(optimizer=optimizer)

        saver = tf.train.Saver(tf.contrib.slim.get_variables())
        tf.global_variables_initializer().run()

        n_epoch, n_iter_per_epoch, idx = args['epochs'], 10 * int(xtr.shape[0] / 100), np
.arange(xtr.shape[0])

        # # dictionaries needed for evaluation
        t0, t1 = np.zeros((X.shape[0], 1)), np.ones((X.shape[0], 1))
        # tr0t, tr1t = np.zeros((xte.shape[0], 1)), np.ones((xte.shape[0], 1))
        f1 = {x_ph_cont: X, t_ph: t1}
        f0 = {x_ph_cont: X, t_ph: t0}
        # f1t = {x_ph_bin: xte[:, 0:len(binfeats)], x_ph_cont: xte[:, len(binfeats):], t_
ph: tr1t}
        # f0t = {x_ph_bin: xte[:, 0:len(binfeats)], x_ph_cont: xte[:, len(binfeats):], t_
ph: tr0t}

        for epoch in range(n_epoch):
            avg_loss = 0.0
```

```
            widgets = ["epoch #%d|" % epoch, Percentage(), Bar(), ETA()]
            pbar = ProgressBar(n_iter_per_epoch, widgets=widgets)
            pbar.start()
            np.random.shuffle(idx)
            for j in range(n_iter_per_epoch):
                # print('j', j)
                # pbar.update(j)
                batch = np.random.choice(idx, 100)
                x_train, y_train, t_train = xtr[batch], ytr[batch], ttr[batch]
                info_dict = inference.update(feed_dict={x_ph_cont: x_train,
                                                        t_ph: t_train, y_ph: y_train})
                avg_loss += info_dict['loss']

            avg_loss = avg_loss / n_iter_per_epoch
            avg_loss = avg_loss / 100

            if epoch % args['earl'] == 0 or epoch == (n_epoch - 1):
                logpvalid = sess.run(logp_valid, feed_dict={x_ph_cont: xva,
                                                            t_ph: tva, y_ph: yva})
                if logpvalid >= best_logpvalid:
                    print('Improved validation bound, old: {:0.3f}, new: {:0.3f}'.format(
best_logpvalid, logpvalid))
                    best_logpvalid = logpvalid
                    saver.save(sess, 'data/cevea_models/dlvm')


        saver.restore(sess, 'data/cevea_models/dlvm')
        y0, y1 = get_y0_y1(sess, y_post, f0, f1, shape=Y.shape, L=100)
        y0, y1 = y0 * ys + ym, y1 * ys + ym

        sess.close()

    return y0.reshape((-1)), y1.reshape((-1))

if __name__ == '__main__':

    Z, X, w, y, ps = gen_dlvm()
    y0, y1 = cevae_tf(X, w, y, n_epochs=10)
    print('cevea_tf OKAY !')
    print(y0.shape, y1.shape)
```