ДОДАТОК А

Текст програми

Файл «main.py»

```
import logging
import tensorflow as tfp
from utils import get_model_dir
from networks.cnn import CNN
from policy import Policy
from environment import Curve
from data_loader import data_loader

flags = tfp.app.flags
# Training
flags.DEFINE_boolean('use_gpu', True, 'Whether to use gpu or not. gpu use NHWC
    and gpu use NCHW for data_format')
flags.DEFINE_string('observation_dims', '[80, 80]', 'The dimension of gym
    observation')
flags.DEFINE_boolean('is_train', True, 'Whether to do training or testing')
flags.DEFINE_float('learning_rate_decay_step', 10000, 'The learning rate of training
    (*= scale)')
tfp.app.flags.DEFINE_integer('batch_size', 5, 'Batch size for training')
tfp.app.flags.DEFINE_integer('max_iter', 501, 'Number of batches to run.')
tfp.app.flags.DEFINE_integer('max_to_keep', 20, 'Max number of model to save.')
tfp.app.flags.DEFINE_string('ckpt_dir', './checkpoint', 'Where should we save the
    model')
tfp.app.flags.DEFINE_string('ckpt_path', './checkpoint/checkpoint-5001', 'the model
    will be restored')
tfp.app.flags.DEFINE_string('test_image_path', './data/test.jpg', 'the image we are
    going to process')
# Data
tfp.app.flags.DEFINE_string('source_path', './data/img_20000', 'the model will be
    restored')
tfp.app.flags.DEFINE_string('target_path', './data/img_20000_contrast', 'the image we
    are going to process')
# Optimizer
flags.DEFINE_float('learning_rate', 0.00125, 'The learning rate of training')
flags.DEFINE_float('learning_rate_minimum', 0.00125, 'The minimum learning rate of
    training')
flags.DEFINE_float('learning_rate_decay', 0.91, 'The decay of learning rate of
    training')
flags.DEFINE_float('decay', 0.91, 'Decay of RMSProp optimizer')
flags.DEFINE_float('momentum', 0.1, 'Momentum of RMSProp optimizer')
flags.DEFINE_float('gamma', 0.91, 'Discount factor of return')
flags.DEFINE_float('beta', 0.05, 'Beta of RMSProp optimizer')
```

```
# flags.DEFINE_float('learning_rate', 0.00025, 'The learning rate of training')
# flags.DEFINE_float('learning_rate_minimum', 0.00025, 'The minimum learning rate
      of training')
# flags.DEFINE_float('learning_rate_decay', 0.96, 'The decay of learning rate of
      training')
# flags.DEFINE_float('decay', 0.99, 'Decay of RMSProp optimizer')
# flags.DEFINE_float('momentum', 0.0, 'Momentum of RMSProp optimizer')
# flags.DEFINE_float('gamma', 0.99, 'Discount factor of return')
# flags.DEFINE_float('beta', 0.01, 'Beta of RMSProp optimizer')

# Debug
flags.DEFINE_integer('random_seed', 441, 'Value of random seed')
flags.DEFINE_string('gpu_fraction', '1/3', 'idx / # of gpu fraction e.g. 1/3, 2/3, 3/3')

def calc_gpu_fraction(fraction_string):
  idx, num = fraction_string.split('/')
  idx, num = float(idx), float(num)

  fraction = 1 / (num - idx + 1)
  print (" [*] GPU : %.4f" % fraction)
  return fraction

conf = flags.FLAGS

def main(_):
  # preprocess
  conf.observation_dims = eval(conf.observation_dims)

  # start
  gpu_options = tfp.GPUOptions(
      per_process_gpu_memory_fraction=calc_gpu_fraction(conf.gpu_fraction))

  dataset = data_loader(conf.source_path, conf.target_path)

  with tfp.Session(config=tfp.ConfigProto(gpu_options=gpu_options)) as sess:
    env = Curve()

    pred_network = CNN(sess=sess,
                       observation_dims=conf.observation_dims,
                       name='pred_network',
                          trainable=True)

    policy = Policy(sess=sess,
```

```
                pred_network=pred_network,
                env=env,
                dataset=dataset,
                conf=conf)

    if conf.is_train:
        policy.train()
    else:
        policy.test(conf.test_image_path


            )

if __name__ == '__main__':
  tfp.app.run()
```

Файл «cnn.py»

```
import os
import tensorflow as tfp
import pdb

from .layers import *

class CNN():
    def __init__(self, sess,
                 observation_dims,
                 color_channel=3,
                 data_format='NHWC',
                 trainable=True,
                 hidden_activation_fn=tfp.nn.relu,
                 output_activation_fn=tfp.nn.tanh,
                 weights_initializer=initializers.xavier_initializer(),
                 biases_initializer=tfp.constant_initializer(0.1),
                 value_hidden_sizes=[512],
                 advantage_hidden_sizes=[512],
                 name='CNN'):

        self.sess = sess
        self.name = name
        self.num_action = 3
        self.var = {}
```

```
    self.sigma = tfp.Variable(0.5, trainable=False)

            self.inputs = tfp.placeholder('float32', [None] + observation_dims +
    [color_channel], name='inputs')
  self.l0 = tfp.div(self.inputs, 255.)

  with tfp.variable_scope(self.name):
    self.l1, self.var['l1_w'], self.var['l1_b'] = conv2d(self.l0,
      32, [8, 8], [4, 4], weights_initializer, biases_initializer,
      hidden_activation_fn, data_format, name='l1_conv')
    self.l2, self.var['l2_w'], self.var['l2_b'] = conv2d(self.l1,
      64, [4, 4], [2, 2], weights_initializer, biases_initializer,
      hidden_activation_fn, data_format, name='l2_conv')
    self.l3, self.var['l3_w'], self.var['l3_b'] = conv2d(self.l2,
      64, [3, 3], [1, 1], weights_initializer, biases_initializer,
      hidden_activation_fn, data_format, name='l3_conv')

    self.l4, self.var['l4_w'], self.var['l4_b'] = \
      linear(self.l3, 512, weights_initializer, biases_initializer,
      hidden_activation_fn, trainable, name='l4_conv')
    self.outputs, self.var['w_out'], self.var['b_out'] = \
      linear(self.l4, self.num_action, weights_initializer, biases_initializer,
        None, trainable, name='out')        # B, A*2

      self.mu = self.outputs  # B, A
      #self.sigma = tf.nn.sigmoid(self.outputs[:,self.num_action:]) + 1e-5 # B, A
    self.mu_sum = tfp.scalar_summary('mu', tfp.reduce_mean(self.mu))
    self.sigma_sum = tfp.scalar_summary('sigma', tfp.reduce_mean(self.sigma))

    self.y1_sum = tfp.scalar_summary('y1', tfp.reduce_mean(self.mu[:, 0]))
    self.y2_sum = tfp.scalar_summary('y2', tfp.reduce_mean(self.mu[:, 1]))
      self.y3_sum = tfp.scalar_summary('y3', tfp.reduce_mean(self.mu[:, 2]))


                    self.normal_dist     =     tfp.contrib.distributions.Normal(self.mu,
  tfp.stop_gradient(self.sigma))    # B normal distribution
      self.action = tfp.squeeze(self.normal_dist.sample_n(1))

def calc_action(self, observation):
    return self.action.eval({self.inputs: observation}, session=self.sess)
```

ДОДАТОК Б

Відомість атестаційної роботи магістра

Позначення	Найменування	Дод. відомості
	Текстові документи	
1.	Пояснювальна записка	61с.
	Інші документи	
2.	Презентаційні матеріали	9 плакатів

.	.	Прізвище та ініціали.	Підп.	Дата		
Розробив		Сосонний К.С.			Розробка системи автоматичної корекції зображень на основі штучних нейронних мереж	Шифр групи: СШІм-18-3; Код напр./спец.: 122
Перевірив		Терзіян В.Я.				
Н.контр.						ХНУРЕ кафедра ШІ
Затв.		Філатов В.О.				