This repository Search Pull requests Issues Marketplace Gist

articuno144 / **DMT2017**

Unwatch 2 Star 1

Code Issues 0 Pull requests 0 Projects 0 Wiki Settings Insights

Branch: master DMT2017 / main / **assemb_MMGonly_aggressive.py** Find file

articuno144 Finished project 69c5d8f 3

1 contributor

225 lines (199 sloc) 7.53 KB Raw Blame History

```
import time
import struct
import tensorflow as tf
import numpy as np
import random
import time
from threading import Thread

import drone_control as dc
from CNN_functions_MMGonly import conv_net
from load_data import noise_values, noised_values

n_classes = 9  # 5 gestures, 1 noise
x = tf.placeholder("float", [None, 50, 3])
y = tf.placeholder("float", [None, n_classes])
keep_prob = tf.placeholder("float")
n_gesture = 3
n_trial = 5
smp_per_trial = 8
ready = 0
new_noise = None
training_set = None

roll, pitch, yaw = None, None, None

weights = {
    'wc3': tf.Variable(tf.random_normal([3, 3, 1, 32])),
    # 5x5 conv, 32 inputs, 64 outputs
    'wc4': tf.Variable(tf.random_normal([5, 1, 32, 64])),
    # fully connected, 7*7*64 inputs, 1024 outputs
    'wd2': tf.Variable(tf.random_normal([10*64, 1024])),
    'out': tf.Variable(tf.random_normal([1024, 9]))
}

biases = {
    'bc3': tf.Variable(tf.random_normal([32])),
    'bc4': tf.Variable(tf.random_normal([64])),
    'bd1': tf.Variable(tf.random_normal([1024])),
    'bd2': tf.Variable(tf.random_normal([1024])),
    'out': tf.Variable(tf.random_normal([9]))
}

pred = conv_net(x, weights, biases, keep_prob)
cost = tf.reduce_mean(
    tf.nn.softmax_cross_entropy_with_logits(logits=pred, labels=y))
with tf.device('/cpu:0'):
    c_argmax = tf.argmax(pred, 1)
```

```python
    c_sigmoid_pred = tf.sigmoid(pred)

m = np.zeros([1, 50, 3], dtype=float)
ctr = 0


def process(sess, m):
    "m is a 6 by 50 matrix"
    return list([ready, int(sess.run(c_argmax, feed_dict={x: m, keep_prob: 1.0}))])


def standardize(s):
    s = list(s)
    assert len(s) == 6, "LengthError"
    ss = []
    ss.append((s[0]*256+s[1]-250)/100)
    ss.append((s[2]*256+s[3]-250)/100)
    ss.append((s[4]*256+s[5]-250)/100)
    return ss

learning_rate = tf.placeholder("float")
sess = tf.InteractiveSession()
saver = tf.train.Saver()
saver.restore(sess, 'Saved\\CNN_MMGonly50_pretrain')
second_optimizer = tf.train.GradientDescentOptimizer(learning_rate=learning_rate).minimize(
    cost, var_list=[weights['wd2'], weights['out'], biases['bd2'], biases['out']])
correct_pred = tf.equal(tf.argmax(pred, 1), tf.argmax(y, 1))
accuracy = tf.reduce_mean(tf.cast(correct_pred, tf.float32))

tmp = input("key in anything to start ")
f = open(r'\\.\pipe\GesturePipe', 'r+b', 0)
n = struct.unpack('I', f.read(4))[0]    # Read str length
s = f.read(n)                           # Read str
f.seek(0)                               # Important!!!
print('Read:', list(s))


r_prev = 10
c = 0
# get training_set
while True:
    c += 1
    s = list(s)
    roll = (s[9]-100)*10/3.14
    pitch = (s[10]-100)*10/3.14
    m[0, :49, :] = m[0, 1:, :]
    m[0, 49, :] = standardize(s[3:9])
    if s[0] == 0 and r_prev == 1:
        g, t = s[1]-1, s[2]-1
        if training_set == None:
            training_set = np.array(m)
            cali_d = np.zeros(9)
            cali_d[s[1]] = 1
            cali_d = cali_d.reshape((1, 9))
        else:
            training_set = np.concatenate((training_set, np.array(m)), axis=0)
            cal = np.zeros(9)
            cal[s[1]] = 1
            cali_d = np.concatenate((cali_d, cal.reshape(1, 9)), axis=0)
        c = 0
    if training_set != None and c < 5:
        training_set = np.concatenate((training_set, np.array(m)), axis=0)
        cal = np.zeros(9)
        cal[s[1]] = 1
        cali_d = np.concatenate((cali_d, cal.reshape(1, 9)), axis=0)
    if s[0] == 0:
        if new_noise == None and c > 5 and c < 95:
            new_noise = np.array(m)
            new_noised = np.array([0, 0, 0, 0, 0, 0, 0, 0, 1])
            new_noised = new_noised.reshape((1, 9))
        elif c > 20 and random.randint(0, 20) > 12:
            new_noise = np.concatenate((new_noise, np.array(m)), axis=0)
            new_noised = np.concatenate(
                (new_noised, np.array([0, 0, 0, 0, 0, 0, 0, 0, 1]).reshape((1, 9))), axis=0)
    r_prev = s[0]
    if s[0] > 1:  # training_set ready
        break
    p = process(sess, m)
    p = bytes(p)
    f.write(struct.pack('I', len(p)) + p)   # Write str length and str
    f.seek(0)                               # EDIT: This is also necessary

    n = struct.unpack('I', f.read(4))[0]    # Read str length
    s = f.read(n)                           # Read str
    f.seek(0)                               # Important!!!
    buf = p[1]
    if buf != 8:
        print('pred: ', buf)

step = 0
batch_size = 200
display_step = 100
# train
training_iters = 200000
batch_x, batch_y = training_set, cali_d
while step * batch_size < training_iters:
    n = random.randint(0, noise_values.shape[0]-batch_size-5)
    noise_x, noise_y = noise_values[
        n:n+batch_size, :, :], noised_values[n:n+batch_size, :]

    # Reshape data to get 28 seq of 28 elements
    #batch_x = batch_x.reshape((batch_size, n_steps, n_input))
    # Run optimization op (backprop)
    sess.run(second_optimizer, feed_dict={
             x: batch_x, y: batch_y, keep_prob: 0.5, learning_rate: 0.0001})
    sess.run(second_optimizer, feed_dict={
             x: noise_x, y: noise_y, keep_prob: 0.5, learning_rate: 0.0005})
    sess.run(second_optimizer, feed_dict={
             x: new_noise, y: new_noised, keep_prob: 0.5, learning_rate: 0.00005})
    if step % display_step == 0:
        # Calculate batch accuracy
        acc = sess.run(accuracy, feed_dict={x: np.concatenate(
            (batch_x, new_noise), axis=0), y: np.concatenate((batch_y, new_noised), axis=0), keep_prob: 1.0})
        print("Iter " + str(step*batch_size) + ", Minibatch Accuracy= " +
              "{:.6f}".format(acc))
    step += 1

target = [[0.15, 0, -0.05],[-0.15,0,-0.15]]
start_signal = [0]
target_locked = True
control_Thread = Thread(target=dc.control,
                        args=(target, ["radio://0/80/250K","radio://0/12/1M"], start_signal))
control_Thread.start()
new_gesture_counter = 0


def enter_start(start_signal):
    while True:
        time.sleep(0.1)
        input("press enter to start or stop")
        start_signal[0] = 1 - start_signal[0]

enter_start_thread = Thread(target=enter_start, args=(start_signal,))
enter_start_thread.start()

gesture_window = [8, 8, 8, 8, 8, 8, 8, 8]
while True:
    print(start_signal, target_locked, target)
    s = list(s)
    roll = (s[9]-100)*10/3.14
    pitch = (s[10]-100)*10/3.14
    m[0, :49, :] = m[0, 1:, :]
    m[0, 49, :] = standardize(s[3:9])
    p = process(sess, m)
    p = bytes(p)
    f.write(struct.pack('I', len(p)) + p)   # Write str length and str
    f.seek(0)                               # EDIT: This is also necessary

    n = struct.unpack('I', f.read(4))[0]    # Read str length
    s = f.read(n)                           # Read str
    f.seek(0)                               # Important!!!
    gesture_window[:7] = gesture_window[1:]
    gesture_window[7] = p[1]
    if new_gesture_counter > 0:
        new_gesture_counter += 1
    if new_gesture_counter > 20:
        new_gesture_counter = 0
    if gesture_window[6] == 8 and gesture_window[7] == 8 and gesture_window[5] != 8:
        if new_gesture_counter == 0:
            new_gesture_counter += 1
            if gesture_window[0] == 0:
                target[0][2] = 0 - target[0][2]
            elif gesture_window[0] == 2:
                target_locked = not target_locked
            elif gesture_window[0] == 1:
                pass
                # if start_signal[0] == 0:
                #     start_signal[0] = 1
                # else:
                #     print("drone landing")

    buf = p[1]
    # if buf != 8:
    #     print('pred: ', buf)
    if not target_locked:
        target[0][0] = min(max(target[0][0]+pitch/1000, -0.2), 0.2)
        target[0][1] = min(max(target[0][1]+roll/1000, -0.2), 0.2)
```