# Document：MazeFinder

# 基于DQN的快速避障模型代码详解

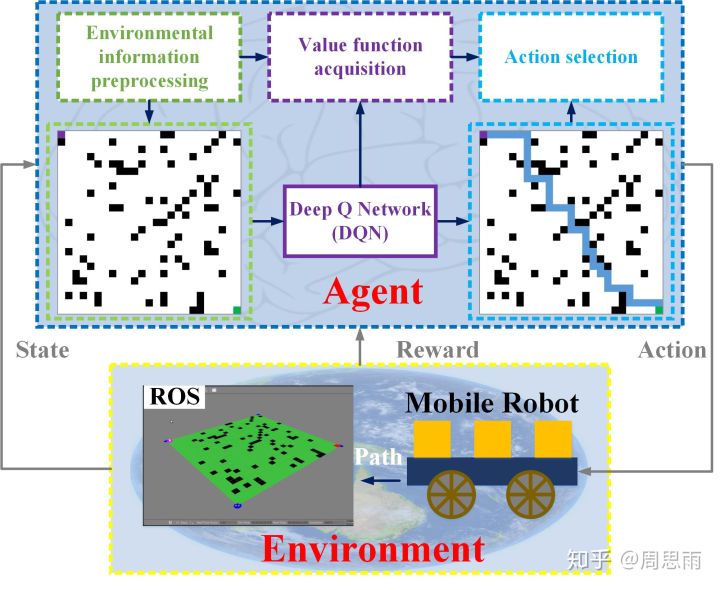
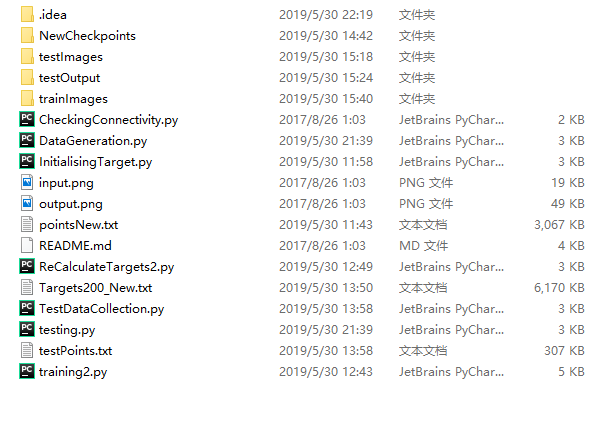
该程序将由几个Blockes（由黑色块颜色表示）组成的图像作为输入，起始点由蓝色表示，目的地由绿色表示。它输出一个由输入到输出的可能路径之一组成的图像。下面显示的是程序的输入和输出。

输入1 输出1

输入图像被馈送到由2个conv和2个fc层组成的模型，其输出对应于底部和右侧动作的Q值。Agent根据哪个Q值更大而向右或向下移动，并且使用代理的新位置生成的相应新图像再次馈送到模型中。获得输出状态并反馈新图像，重复该过程直到代理到达目的地。

代码整体架构：

算法流程 代码架构

## 1.数据生成

代码DataGeneration.py为任务生成必需的数据集。它在25X25大小的图像中随机分配1X1像素大小的块。生成对应于每个不同起始位置的所有625个图像并存储在文件夹中。生成具有随机变化的障碍物位置的200个不同的图像，因此总训练图像达200 \* 625=125000。还生成与每个不同状态（图像）相关联的分数并将其存储在txt文件中，其中起始点与障碍物重合的状态获得-100的分数，当起始点与目的地重合时，其获得分数100，剩余的其他状态得分为0。

DataGeneration.py代码详解：

## 2.训练

代码InitialisingTarget.py生成与每个训练图像相关联的初始Q值，并将它们存储在txt文件-Targets200\_New.txt中。生成的Q值只不过是随机初始化模型的输出。

InitialisingTarget.py代码详解：

**2.1函数的定义**

#分别定义权重，偏差函数，卷积网络以及批量获取数据的函数

def weight\_variable(shape):

initial = tf.truncated\_normal(shape, stddev=0.05)

return tf.Variable(initial)

def bias\_variable(shape):

initial = tf.constant(0.05, shape=shape)

return tf.Variable(initial)

def conv2d(x, W):

return tf.nn.conv2d(x, W, strides=[1, 1, 1, 1], padding='SAME')

def getBatchInput(inputs,start,batchSize) :

first = start

start = start + batchSize

end = start

return inputs[first:end],start

def max\_pool(x):

return tf.nn.max\_pool(x, ksize=[1, 2, 2, 1],strides=[1, 2, 2, 1], padding='SAME')

**2.2定义所需要的参数**

imageSize = [25,25,3] #图片大小

batchSize = 100 #训练批次大小

games = 200 #200个不同的Maze

totalImages = games\*imageSize[0]\*imageSize[1] #总共图片的数量

learningRate = 0.001 #学习率的设定

lr\_decay\_rate = 0.9

lr\_decay\_step = 2000

folderName = 'NewCheckpoints'

if os.path.exists(folderName):

shutil.rmtree(folderName) #递归删除文件夹下的所有子文件夹和子文件

os.mkdir(folderName) #else创建文件夹

checkpointFile = 'NewCheckpoints/Checkpoint3.ckpt'

2.3网络结构的定义

x = tf.placeholder(tf.float32, shape=[None, imageSize[0], imageSize[1], imageSize[2]]) #设置占位符，大小为输入大小 None为自 动查看相应的样本数

W\_conv1 = weight\_variable([2, 2, 3, 10]) #fliter的大小 卷积核2\*2 输入3通道，输出10通道

b\_conv1 = bias\_variable([10]) # 一维10个变量的偏置

h\_conv1 = tf.nn.relu(conv2d(x, W\_conv1) + b\_conv1) #第一层卷积 使用relu

W\_conv2 = weight\_variable([2, 2, 10, 20])#卷积核为2\*2 输入10通道 输出20通道

b\_conv2 = bias\_variable([20]) #输出一维20个变量的偏置

h\_conv2 = tf.nn.relu(conv2d(h\_conv1, W\_conv2) + b\_conv2)#第二层神经网络

h\_conv2\_flat = tf.reshape(h\_conv2, [-1, 25\*25\*20])# -1代表的含义是不用我们自己指定大小，函数会自动计算，将25\*25\*20个参数reshape成1维向量

W\_fc1 = weight\_variable([25\*25\*20, 100])

b\_fc1 = bias\_variable([100])

h\_fc1 = tf.nn.relu(tf.matmul(h\_conv2\_flat, W\_fc1) + b\_fc1)#第一个全连接层

W\_fc2 = weight\_variable([100, 2])

b\_fc2 = bias\_variable([2])

y\_out=tf.matmul(h\_fc1, W\_fc2) + b\_fc2#第二个全连接层

loss = tf.reduce\_mean(tf.square(y\_out),1)#square取平方和，指定第二个参数为1，则第二维的元素取平均值，即每一行求平均值

avg\_loss = tf.reduce\_mean(loss) #不指定参数1对每个元素取均值

global\_step = tf.Variable(0, trainable=False)

lr = tf.train.exponential\_decay(learningRate,

global\_step,

lr\_decay\_step,

lr\_decay\_rate,

staircase=True)

首先使用较大学习率(目的：为快速得到一个比较优的解)；

然后通过迭代逐步减小学习率(目的：为使模型在训练后期更加稳定)。

每隔lr\_decay\_step步，learningRate要乘以lr\_decay\_rate；

staircase = True，每decay\_steps次计算学习速率变化，更新原始学习速率。

train\_step = tf.train.AdamOptimizer(lr).minimize(avg\_loss)

sess = tf.InteractiveSession()

saver = tf.train.Saver()

sess.run(tf.initialize\_all\_variables())

inputs = np.zeros([totalImages,imageSize[0],imageSize[1],imageSize[2]])

print('reading inputs')

for i in range(totalImages) :

temp = imageSize[0]\*imageSize[1]

inputs[i] = cv2.imread('trainImages/image\_'+str(int(i/temp))+'\_'+str(i%temp)+'.png')

print('inputs read')

start = 0

initialTarget = []

iterations = int(totalImages/batchSize)

save\_path = saver.save(sess, checkpointFile)

print("Model saved in file: %s" % save\_path)

print('number of iterations is %d'%iterations)

for i in range(iterations) :

batchInput,start = getBatchInput(inputs,start,batchSize)

batchOutput = sess.run(y\_out,feed\_dict={x: batchInput})

if i%50 == 0 :

print('%d iterations reached'%i)

for j in range(batchSize) :

initialTarget.append(batchOutput[j])

print("stop")

np.savetxt('Targets200\_New.txt',initialTarget)

代码Training2.py开始训练模型。从training数据中选择随机批量大小的图像集并将其馈送到模型。根据损失Loss更新模型权重，该损失是输出Q值和预期Q值之间的平方差。预期的Q值：

Q（s，a）= r（s，a）+ gamma \* max（Q1（s1，a）），

其中max取2个动作。这里Q1对应于存储在'Targets200\_New.txt'文件中的初始Q值。奖励r也是下一个状态和当前状态之间得分的差异。几个训练Q1值再次更新并存储在相同的txt文件中，输出来自训练模型。新的Q值再次用于训练模型。生成目标Q值和训练模型的步骤重复几个步骤，直到模型学习到所需特征。

## 3.测试

就像DataGeneration.py一样，代码TestDataCollection.py也以相同的格式生成图像，除了它不是200个game，而是在单独的文件夹中生成仅对应20个game的图像。对于20个game中的每个game，代码testing.py获取对应于Agent位于Maze中位置的图像，并将最终路径的图像输出到单独的文件夹中。对于由于与障碍物重合而无法找到路径的game，则不会生成图像。