

songrotek的专栏

知乎专栏：<https://zhuanlan.zhihu.com/intelligentunit>

目录视

个人资料



songrotek

关注

发私信

访问：533415次

积分：6084

等级：

BLOG

6

排名：第2606名

原创：107篇

转载：5篇

译文：2篇

评论：197条

文章搜索

博客专栏

深度增强学习DRL

文章：18篇

阅读：48877

iOS与LEGO EV3机器人

文章：15篇

阅读：53995

iOS 开发从入门到超级

文章：20篇

阅读：246465

文章分类

Deep Reinforcement Learning (15)

Artificial Intelligence (13)

Deep Learning (12)

Robotics (9)

Reinforcement Learning (4)

Machine Learning (2)

Computer Vision (9)

iOS Development (40)

TechYY Series (6)

iOS与LEGO EV3混合机器人编程 (14)

【专家问答】韦玮：Python基础编程实战专题

【知识库】Swift资源大集合

【公告】博客新皮肤上线啦

CSDN福利第二期

用Tensorflow基于Deep Q Learning DQN 玩Flappy Bird

标签：

深度增强学习

DQN

2016-03-22 00:11

3524人阅读

评论

分类：

Deep Reinforcement Learning ( 14 )

版权声明：本文为博主原创文章，未经博主允许不得转载。

目录(?)

[+]

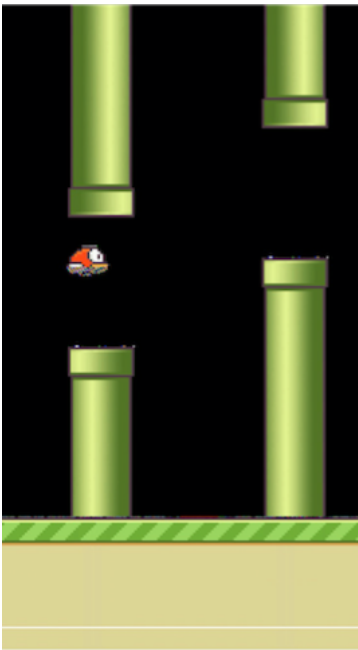
前言

2013年DeepMind 在NIPS上发表Playing Atari with Deep Reinforcement Learning 一文，提出了DQN ( Deep Q Network 玩Atari游戏，即只有像素输入，看着屏幕玩游戏。Deep Mind就凭借这个应用以6亿美元被Google收购。由于DQN的开源，在版本的DQN程序。但大多是复现Atari的游戏，代码量很大，也不好理解。

Flappy Bird是个极其简单又困难的游戏，风靡一时。在很早之前，就有人使用Q-Learning 算法来实现完Flappy Bird。<http://sarvagyaish.github.io/FlappyBirdRL/>  
但是这个的实现是通过获取小鸟的具体位置信息来实现的。

能否使用DQN来实现通过屏幕学习玩Flappy Bird是一个有意思的挑战。（话说本人和朋友在去年年底也考虑了这个idea，但游戏屏幕只能使用具体位置来学习，不过其实也成功了）

最近，github上有人放出使用DQN玩Flappy Bird的代码，<https://github.com/yenchenlin1994/DeepLearningFlappyBird> 该repo通过结合之前的repo成功实现了这个想法。这个repo对整个实现过程进行了较详细的分析，但是由于其DQN算法的代码较为混乱，不易理解。



为此，本人改写了一个版本<https://github.com/songrotek/DRL-FlappyBird>

对DQN代码进行了重新改写。本质上对其做了类的封装，从而使代码更具通用性。可以方便移植到其他应用。

Python (4)

Others (6)

ROS (1)

QT (0)

文章存档

2016年05月 (7)

2016年04月 (5)

2016年03月 (7)

2016年01月 (4)

2015年08月 (8)

展开

阅读排行

iOS 开发 入门：使用Ad Hoc ... (39482)

iOS 开发 初级：应用内购买 I... (32873)

iOS 开发 入门：发布应用到A... (20294)

iOS 开发 高级：通过AirServ... (19513)

iOS 开发 入门：加入iOS Dev... (18427)

iOS 开发 中级：HTTP请求 G... (18318)

iOS 开发 初级：应用多语言... (17794)

解密Google Deepmind Alp... (16731)

iOS开发 之 可穿戴设备 蓝牙4... (13881)

iOS 开发 初级：插入Admob ... (12822)

最新评论

AR Drone系列之：使用ROS catkin创建...  
stihy : @songrotek:我调用image\_view ro  
srun image\_view imag...

深度学习之Matlab 转C++在iOS上测试C...  
songrotek : @xiongcelail:是的

AR Drone系列之：使用ROS catkin创建...  
songrotek : @stihy:用cv\_bridge很快  
的,和ardrone\_autonomy没什么关系

AR Drone系列之：使用ROS catkin创建...  
songrotek : 用cv\_bridge很快的,和ardr  
one\_autonomy没什么关系

深度学习之Matlab 转C++在iOS上测试C...  
xccoming : 您好,请问任意matlab代码都可  
以用coder工具转成C++么?

AR Drone系列之：使用ROS catkin创建...  
stihy : 我读取/ardrone/image\_raw 话  
题,并用Opencv把图像显示出来,但是好  
卡。一会又图...

增强学习Reinforcement Learning经典...  
LosinueriS : 看了博主写的关于RL的文  
章,写得很好,希望以后有机会能多多交流

你是这样获取人工智能AI前沿信息的吗？  
Alex-zhai : 不错的指导, 谢谢

DRL前沿之：Benchmarking Deep Rein...  
Alex-zhai : 楼主,我在ubantu下配置好了  
rllab包。但是在运行stub 例子程序时,卡在  
了using se...

深度学习 Deep Learning UFLDL 最新Tu...  
derk1992 : 谢谢楼主, 运行成功, 学到不  
少！

当然，本文的目的是借Flappy Bird DQN这个代码来详细分析一下DQN算法极其使用。

## DQN 伪代码

这个是NIPS13版本的伪代码：

```
1 Initialize replay memory D to size N
2 Initialize action-value function Q with random weights
3 for episode = 1, M do
4     Initialize state s_1
5     for t = 1, T do
6         With probability  $\epsilon$  select random action a_t
7         otherwise select  $a_t = \arg\max_a Q(s_t, a)$ 
8         Execute action a_t in emulator and observe r_t and s_{t+1}
9         Store transition (s_t, a_t, r_t, s_{t+1}) in D
10        Sample a minibatch of transitions (s_j, a_j, r_j, s_{j+1}) from D
11        Set y_j :=
12            r_j for terminal s_{j+1}
13            r_j +  $\gamma \max_{a'} Q(s_{j+1}, a')$  for non-terminal s_{j+1}
14        Perform a gradient step on (y_j - Q(s_j, a_j))^2 with respect to Q
15    end for
16 end for
```

基本的分析详见[Paper Reading 1 - Playing Atari with Deep Reinforcement Learning](#)

基础知识详见[Deep Reinforcement Learning 基础知识 \( DQN方面 \)](#)

本文主要从代码实现的角度来分析如何编写Flappy Bird DQN的代码

## 编写FlappyBirdDQN.py

首先，FlappyBird的游戏已经编写好，是现成的。提供了很简单的接口：

```
1 nextObservation, reward, terminal = game.frame_step(action)
```

即输入动作，输出执行完动作的屏幕截图，得到的反馈reward，以及游戏是否结束。

那么，现在先把DQN想象为一个大脑，这里我们也用BrainDQN类来表示，这个类只需获取感知信息也就是上面说的观察（截图）然后输出动作即可。

完美的代码封装应该是这样。具体DQN里面如何存储。如何训练是外部不关心的。

因此，我们的FlappyBirdDQN代码只有如下这么短：

```
1 # -----
2 # Project: Deep Q-Learning on Flappy Bird
3 # Author: Flood Sung
4 # Date: 2016.3.21
5 # -----
6
7 import cv2
8 import sys
9 sys.path.append("game/")
10 import wrapped_flappy_bird as game
11 from BrainDQN import BrainDQN
12 import numpy as np
13
14 # preprocess raw image to 80*80 gray image
15 def preprocess(observation):
16     observation = cv2.cvtColor(cv2.resize(observation, (80, 80)), cv2.COLOR_BGR2GRAY)
17     ret, observation = cv2.threshold(observation, 1, 255, cv2.THRESH_BINARY)
18     return np.reshape(observation, (80, 80, 1))
19
20 def playFlappyBird():
21     # Step 1: init BrainDQN
22     brain = BrainDQN()
23     # Step 2: init Flappy Bird Game
24     flappyBird = game.GameState()
25     # Step 3: play game
26     # Step 3.1: obtain init state
27     action0 = np.array([1, 0]) # do nothing
```

http://blog.csdn.net/songrotek/article/details/50951537

2/8

```
28 observation0, reward0, terminal = flappyBird.frame_step(action0)
29 observation0 = cv2.cvtColor(cv2.resize(observation0, (80, 80)), cv2.COLOR_BGR2GRAY)
30 ret, observation0 = cv2.threshold(observation0,1,255, cv2.THRESH_BINARY)
31 brain.setInitState(observation0)
32
33 # Step 3.2: run the game
34 while l!= 0:
35     action = brain.getAction()
36     nextObservation, reward, terminal = flappyBird.frame_step(action)
37     nextObservation = preprocess(nextObservation)
38     brain.setPerception(nextObservation, action, reward, terminal)
39
40 def main():
41     playFlappyBird()
42
43 if __name__ == '__main__':
44     main()
```

核心部分就在while循环里面，由于要讲图像转换为80x80的灰度图，因此，加了一个preprocess预处理函数。

这里，显然只有有游戏引擎，换一个游戏是一样的写法，非常方便。

接下来就是编写BrainDQN.py 我们的游戏大脑

编写BrainDQN

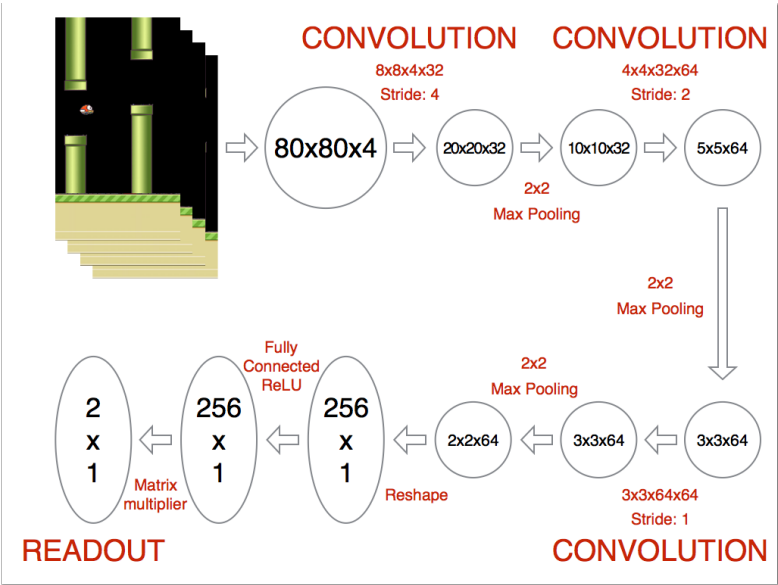
基本架构：

```
1 class BrainDQN:
2     def __init__(self):
3         # init replay memory
4         self.replayMemory = deque()
5         # init Q network
6         self.createQNetwork()
7     def createQNetwork(self):
8
9     def trainQNetwork(self):
10
11     def setPerception(self, nextObservation, action, reward, terminal):
12     def getAction(self):
13     def setInitState(self, observation):
```

基本的架构也就只需要上面这几个函数，其他的都是多余了，接下来就是编写每一部分的代码。

CNN代码

也就是createQNetwork部分，这里采用如下图的结构（转自【1】）：



这里就不讲解整个流程了。主要是针对具体的输入类型和输出设计卷积和全连接层。

代码如下：

```
1 def createQNetwork(self):
2     # network weights
3     W_conv1 = self.weight_variable([8, 8, 4, 32])
4     b_conv1 = self.bias_variable([32])
5
6     W_conv2 = self.weight_variable([4, 4, 32, 64])
7     b_conv2 = self.bias_variable([64])
8
9     W_conv3 = self.weight_variable([3, 3, 64, 64])
10    b_conv3 = self.bias_variable([64])
11
12    W_fc1 = self.weight_variable([1600, 512])
13    b_fc1 = self.bias_variable([512])
14
15    W_fc2 = self.weight_variable([512, self.ACTION])
16    b_fc2 = self.bias_variable([self.ACTION])
17
18    # input layer
19
20    self.stateInput = tf.placeholder("float", [None, 80, 80, 4])
21
22    # hidden layers
23    h_conv1 = tf.nn.relu(self.conv2d(self.stateInput, W_conv1, 4) + b_conv1)
24    h_pool1 = self.max_pool_2x2(h_conv1)
25
26    h_conv2 = tf.nn.relu(self.conv2d(h_pool1, W_conv2, 2) + b_conv2)
27
28    h_conv3 = tf.nn.relu(self.conv2d(h_conv2, W_conv3, 1) + b_conv3)
29
30    h_conv3_flat = tf.reshape(h_conv3, [-1, 1600])
31    h_fc1 = tf.nn.relu(tf.matmul(h_conv3_flat, W_fc1) + b_fc1)
32
33    # Q Value layer
34    self.QValue = tf.matmul(h_fc1, W_fc2) + b_fc2
35
36    self.actionInput = tf.placeholder("float", [None, self.ACTION])
37    self.yInput = tf.placeholder("float", [None])
38    Q_action = tf.reduce_sum(tf.mul(self.QValue, self.actionInput), reduction_indices = 1)
39    self.cost = tf.reduce_mean(tf.square(self.yInput - Q_action))
40    self.trainStep = tf.train.AdamOptimizer(1e-6).minimize(self.cost)
```

记住输出是Q值，关键要计算出cost，里面关键是计算Q\_action的值，即该state和action下的Q值。由于actionInput是one hot， $\text{tf.mul}(\text{self.QValue}, \text{self.actionInput})$ 正好就是该action下的Q值。

training 部分。

这部分是代码的关键部分，主要是要计算y值，也就是target Q值。

```
1 def trainQNetwork(self):
2     # Step 1: obtain random minibatch from replay memory
3     minibatch = random.sample(self.replayMemory, self.BATCH_SIZE)
4     state_batch = [data[0] for data in minibatch]
5     action_batch = [data[1] for data in minibatch]
6     reward_batch = [data[2] for data in minibatch]
7     nextState_batch = [data[3] for data in minibatch]
8
9     # Step 2: calculate y
10    y_batch = []
11    QValue_batch = self.QValue.eval(feed_dict={self.stateInput:nextState_batch})
12    for i in range(0, self.BATCH_SIZE):
13        terminal = minibatch[i][4]
14        if terminal:
15            y_batch.append(reward_batch[i])
16        else:
17            y_batch.append(reward_batch[i] + GAMMA * np.max(QValue_batch[i]))
18
19    self.trainStep.run(feed_dict={
20        self.yInput : y_batch,
21        self.actionInput : action_batch,
22        self.stateInput : state_batch
23    })
```

## 其他部分

其他部分就比较容易了，这里直接贴出完整的代码：

```
1  # -----
2  # File: Deep Q-Learning Algorithm
3  # Author: Flood Sung
4  # Date: 2016.3.21
5  # -----
6
7  import tensorflow as tf
8  import numpy as np
9  import random
10 from collections import deque
11
12 class BrainDQN:
13
14     # Hyper Parameters:
15     ACTION = 2
16     FRAME_PER_ACTION = 1
17     GAMMA = 0.99 # decay rate of past observations
18     OBSERVE = 100000. # timesteps to observe before training
19     EXPLORE = 150000. # frames over which to anneal epsilon
20     FINAL_EPSILON = 0.0 # final value of epsilon
21     INITIAL_EPSILON = 0.0 # starting value of epsilon
22     REPLAY_MEMORY = 50000 # number of previous transitions to remember
23     BATCH_SIZE = 32 # size of minibatch
24
25     def __init__(self):
26         # init replay memory
27         self.replayMemory = deque()
28         # init Q network
29         self.createQNetwork()
30         # init some parameters
31         self.timeStep = 0
32         self.epsilon = self.INITIAL_EPSILON
33
34     def createQNetwork(self):
35         # network weights
36         W_conv1 = self.weight_variable([8, 8, 4, 32])
37         b_conv1 = self.bias_variable([32])
38
39         W_conv2 = self.weight_variable([4, 4, 32, 64])
40         b_conv2 = self.bias_variable([64])
41
42         W_conv3 = self.weight_variable([3, 3, 64, 64])
43         b_conv3 = self.bias_variable([64])
44
45         W_fc1 = self.weight_variable([1600, 512])
46         b_fc1 = self.bias_variable([512])
47
48         W_fc2 = self.weight_variable([512, self.ACTION])
49         b_fc2 = self.bias_variable([self.ACTION])
50
51         # input layer
52
53         self.stateInput = tf.placeholder("float", [None, 80, 80, 4])
54
55         # hidden layers
56         h_conv1 = tf.nn.relu(self.conv2d(self.stateInput, W_conv1, 4) + b_conv1)
57         h_pool1 = self.max_pool_2x2(h_conv1)
58
59         h_conv2 = tf.nn.relu(self.conv2d(h_pool1, W_conv2, 2) + b_conv2)
60
61         h_conv3 = tf.nn.relu(self.conv2d(h_conv2, W_conv3, 1) + b_conv3)
62
63         h_conv3_flat = tf.reshape(h_conv3, [-1, 1600])
64         h_fc1 = tf.nn.relu(tf.matmul(h_conv3_flat, W_fc1) + b_fc1)
65
66         # Q Value layer
67         self.QValue = tf.matmul(h_fc1, W_fc2) + b_fc2
68
69         self.actionInput = tf.placeholder("float", [None, self.ACTION])
70         self.yInput = tf.placeholder("float", [None])
71         Q_action = tf.reduce_sum(tf.mul(self.QValue, self.actionInput), reduction_indices = 1)
72         self.cost = tf.reduce_mean(tf.square(self.yInput - Q_action))
73         self.trainStep = tf.train.AdamOptimizer(1e-6).minimize(self.cost)
```

```

74
75     # saving and loading networks
76     saver = tf.train.Saver()
77     self.session = tf.InteractiveSession()
78     self.session.run(tf.initialize_all_variables())
79     checkpoint = tf.train.get_checkpoint_state("saved_networks")
80     if checkpoint and checkpoint.model_checkpoint_path:
81         saver.restore(self.session, checkpoint.model_checkpoint_path)
82         print "Successfully loaded:", checkpoint.model_checkpoint_path
83     else:
84         print "Could not find old network weights"
85
86     def trainQNetwork(self):
87         # Step 1: obtain random minibatch from replay memory
88         minibatch = random.sample(self.replayMemory, self.BATCH_SIZE)
89         state_batch = [data[0] for data in minibatch]
90         action_batch = [data[1] for data in minibatch]
91         reward_batch = [data[2] for data in minibatch]
92         nextState_batch = [data[3] for data in minibatch]
93
94         # Step 2: calculate y
95         y_batch = []
96         QValue_batch = self.QValue.eval(feed_dict={self.stateInput:nextState_batch})
97         for i in range(0, self.BATCH_SIZE):
98             terminal = minibatch[i][4]
99             if terminal:
100                 y_batch.append(reward_batch[i])
101             else:
102                 y_batch.append(reward_batch[i] + GAMMA * np.max(QValue_batch[i]))
103
104         self.trainStep.run(feed_dict={
105             self.yInput : y_batch,
106             self.actionInput : action_batch,
107             self.stateInput : state_batch
108         })
109
110         # save network every 100000 iteration
111         if self.timeStep % 10000 == 0:
112             saver.save(self.session, 'saved_networks/' + 'network' + '-dqn', global_step = self.timeStep)
113
114     def setPerception(self, nextObservation, action, reward, terminal):
115         newState = np.append(nextObservation, self.currentState[:, :, 1:], axis = 2)
116         self.replayMemory.append((self.currentState, action, reward, newState, terminal))
117         if len(self.replayMemory) > self.REPLAY_MEMORY:
118             self.replayMemory.popleft()
119         if self.timeStep > self.OBSERVE:
120             # Train the network
121             self.trainQNetwork()
122
123         self.currentState = newState
124         self.timeStep += 1
125
126     def getAction(self):
127         QValue = self.QValue.eval(feed_dict= {self.stateInput:[self.currentState]})[0]
128         action = np.zeros(self.ACTION)
129         action_index = 0
130         if self.timeStep % self.FRAME_PER_ACTION == 0:
131             if random.random() <= self.epsilon:
132                 action_index = random.randrange(self.ACTION)
133                 action[action_index] = 1
134             else:
135                 action_index = np.argmax(QValue)
136                 action[action_index] = 1
137         else:
138             action[0] = 1 # do nothing
139
140         # change epsilon
141         if self.epsilon > self.FINAL_EPSILON and self.timeStep > self.OBSERVE:
142             self.epsilon -= (self.INITIAL_EPSILON - self.FINAL_EPSILON)/self.EXPLORE
143
144         return action
145
146     def setInitState(self, observation):
147         self.currentState = np.stack((observation, observation, observation, observation), axis = 2)
148
149     def weight_variable(self, shape):
150         initial = tf.truncated_normal(shape, stddev = 0.01)

```

```
152         return tf.Variable(initial)
153
154     def bias_variable(self, shape):
155         initial = tf.constant(0.01, shape = shape)
156         return tf.Variable(initial)
157
158     def conv2d(self, x, W, stride):
159         return tf.nn.conv2d(x, W, strides = [1, stride, stride, 1], padding = "SAME")
160
161     def max_pool_2x2(self, x):
162         return tf.nn.max_pool(x, ksize = [1, 2, 2, 1], strides = [1, 2, 2, 1], padding = "SAME")
163
164 ..
```

一共也只有160代码。

如果这个任务不使用深度学习，而是人工的从图像中找到小鸟，然后计算小鸟的轨迹，然后计算出应该怎么按键，那么代码没有深度学习大大减少了代码工作。

小结

本文从代码角度对于DQN做了一定的分析，对于DQN的应用，大家可以在此基础上做各种尝试。

顶

1

踩

0

- 上一篇

Paper Reading 4:Massively Parallel Methods for Deep Reinforcement Learning
- 下一篇

深度解读 AlphaGo 算法原理

关闭

我的同类文章



轻钢结构房屋



Reinforcement Learning ( 14 )

Reinf...	2016-05-12	阅读 835	• 增强学习Reinforcement Learning经典...	2016-05-
ng经典...	2016-05-12	阅读 841	• 增强学习Reinforcement Learning经典...	2016-05-
ng for...	2016-05-03	阅读 836	• DRL前沿之：Benchmarking Deep Re...	2016-04-
沿	2016-04-14	阅读 5238	• 深度解读 AlphaGo 算法原理	2016-04-
parallel ...	2016-03-18	阅读 747	• Paper Reading 3:Continuous control...	2016-03-

更多文章

参考知识库



算法与数据结构知识库  
2409 关注 | 4174 收录



大型网站架构知识库  
2253 关注 | 532 收录

猜你在找

- Part 1: 基础语言-Cocos2d-x手...

高并发集群架构超细精讲

Android入门实战教程

拥抱开源： Github使用指南

韦东山嵌入式Linux第一期视频
- cocos2d-x 《Flappy Bird 》三...

Flappy Bird

Unity3D基础教程四通过制作Flap...

下坠的小鸟flappy bird速算电脑版

树莓派制作敲击式Flappy-Bird远...



查看评论



lt2fish1楼 2016-06-19 10:00

博主你好，看了你写的几篇文章收获很大，不知道 `W_fc1 = self.weight_variable([1600,512])`这个地方的1600和512是怎么定的？



songrotekRe: 2016-06-19 10:00

回复lt2fish: 1600是前面的卷积之后全连接的长度，即5x5x64，512是后面隐藏层的神经元数量

您还没有登录,请[\[登录\]](#)或[\[注册\]](#)

\* 以上用户言论只代表其个人观点，不代表CSDN网站的观点或立场

核心技术类目

全部主题	Hadoop	AWS	移动游戏	Java	Android	iOS	Swift	智能硬件	Docker	OpenStack	V	
IE10	Eclipse	CRM	JavaScript	数据库	Ubuntu	NFC	WAP	jQuery	BI	HTML5	Spring	Ap
API	HTML	SDK	IIS	Fedora	XML	LBS	Unity	Splashtop	UML	components	Windows Mob	
QEMU	KDE	Cassandra	CloudStack	FTC	coremail	OPhone	CouchBase	云计算		iOS6	Rackspa	
SpringSide	Maemo	Compuware	大数据	aptech	Perl	Tornado	Ruby	Hibernate		ThinkPHP	HB	
Angular	Cloud Foundry	Redis	Scala	Django	Bootstrap							