**算法源代码说明文档模板**

修订记录

| 日期 | 修订版本 | 修改章节 | 修改描述 | 作者 |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

目录

[1. 原理介绍 4](#_Toc47537820)

[2. 总流程图 4](#_Toc47537821)

[3. 分模块介绍 5](#_Toc47537822)

[3.1 Packages（引用的包）: 5](#_Toc47537823)

[3.1.1 代码 5](#_Toc47537824)

[3.1.2 详细说明 6](#_Toc47537825)

[3.2 逻辑模块1: Replay Memory 6](#_Toc47537826)

[3.2.1代码 6](#_Toc47537827)

[3.2.2流程图 6](#_Toc47537828)

[3.2.3详细说明 7](#_Toc47537829)

[3.3 逻辑模块2: DQN神经网络 7](#_Toc47537830)

[3.3.1代码 7](#_Toc47537831)

[3.3.2流程图 8](#_Toc47537832)

[3.3.3详细说明 8](#_Toc47537833)

[3.4 逻辑模块3: 输入提取 10](#_Toc47537834)

[3.4.1代码 10](#_Toc47537835)

[3.4.2详细说明 11](#_Toc47537836)

[3.5 接口模块1: 实例模型及优化器 12](#_Toc47537837)

[3.5.1代码 12](#_Toc47537838)

[3.5.2功能描述 13](#_Toc47537839)

[3.5.3接口描述 14](#_Toc47537840)

[3.6 逻辑模块3: 训练 14](#_Toc47537841)

[3.6.1代码 15](#_Toc47537842)

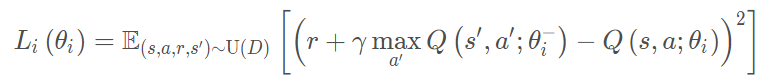
[3.6.2详细说明 16](#_Toc47537843)

# 原理介绍

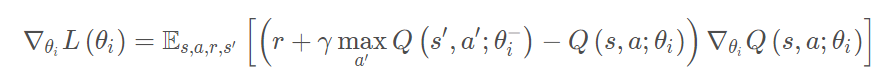
用深度卷积神经网络近似值函数手机屏幕截图

描述已自动生成

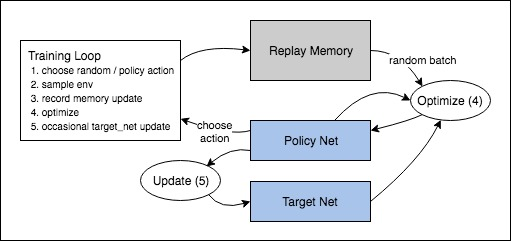
执行Replay Memory，将每一时间步t的经验存储到经验池（Memory）中，在学习的过程中，对从经验池中随机抽取的样本采用Q-learning更新。第i次迭代，获得损失函数：



对损失函数求权重上的导数，可以得到梯度：



# 总流程图



具体步骤如下：

1）初始化Memory，用作Replay Memory；

2）设状态值函数Q作为预测网络（Policy Net），随机初始化权重参数。设状态值函数作为目标网络（target Net），随机初始化权重参数；

3）以概率eps随机选择执行动作a，获得环境反馈r和下一时间的游戏图像

4）将获得的状态转移存入Memory中

5）随机从Memory里面采样，计算更新Q值，基于梯度下降更新Q网络（Policy Net）权重参数。

6）每经过C次迭代后，更新target Net的网络参数为Policy Net的网络参数。

# 分模块介绍

## Packages（引用的包）:

### 代码

|  |
| --- |
| **import** gym  **import** math  **import** random  **import** numpy **as** np  **import** matplotlib  **import** matplotlib.pyplot **as** plt  **from** collections **import** namedtuple  **from** itertools **import** count  **from** PIL **import** Image  **import** torch  **import** torch.nn **as** nn  **import** torch.optim **as** optim  **import** torch.nn.functional **as** F  **import** torchvision.transforms **as** T  env **=** gym**.**make('CartPole-v0')**.**unwrapped  *# set up matplotlib*  is\_ipython **=** 'inline' **in** matplotlib**.**get\_backend()  **if** is\_ipython:  **from** IPython **import** display  plt**.**ion()  *# if gpu is to be used*  device **=** torch**.**device("cuda" **if** torch**.**cuda**.**is\_available() **else** "cpu") |
|  |

### 详细说明

gym: 模拟环境；

Pytorch中使用模块：神经网络（torch.nn），优化（torch.optim）,自动微分（torch，autograd），图像操作（torchvision）

## 逻辑模块1: Replay Memory

### 3.2.1代码

|  |
| --- |
| Transition **=** namedtuple('Transition',  ('state', 'action', 'next\_state', 'reward'))  **class** **ReplayMemory**(object):  **def** \_\_init\_\_(self, capacity):  self**.**capacity **=** capacity  self**.**memory **=** []  self**.**position **=** 0  **def** **push**(self, **\***args):  """Saves a transition."""  **if** len(self**.**memory) **<** self**.**capacity:  self**.**memory**.**append(**None**)  self**.**memory[self**.**position] **=** Transition(**\***args)  self**.**position **=** (self**.**position **+** 1) **%** self**.**capacity  **def** **sample**(self, batch\_size):  **return** random**.**sample(self**.**memory, batch\_size)  **def** \_\_len\_\_(self):  **return** len(self**.**memory) |

### 3.2.2流程图

|  |
| --- |
|  |
|  |
|  |
| …………… |
|  |

### 3.2.3详细说明

用于存储agent观察到的变化（transition），随机抽取小批量进行训练。目的是，打破数据之间的关联性，并做到有效利用历史数据。代码中用到了两个类：

1. Replay Memory - 一个有限大小的缓存区域，用于保存transition。Sample()这个函数，用于随机选择一批transition，用于训练。
2. Transition - 一个命名的元组，表示环境中的单个过渡。实质上，它将（state，action）对映射到它的（next\_sate，reward）结果。这里的状态是屏幕的差异图像。

## 逻辑模块2: DQN神经网络

### 3.3.1代码

|  |
| --- |
| **class** **DQN**(nn**.**Module):  **def** \_\_init\_\_(self, h, w, outputs):  super(DQN, self)**.**\_\_init\_\_()  self**.**conv1 **=** nn**.**Conv2d(3, 16, kernel\_size**=**5, stride**=**2)  self**.**bn1 **=** nn**.**BatchNorm2d(16)  self**.**conv2 **=** nn**.**Conv2d(16, 32, kernel\_size**=**5, stride**=**2)  self**.**bn2 **=** nn**.**BatchNorm2d(32)  self**.**conv3 **=** nn**.**Conv2d(32, 32, kernel\_size**=**5, stride**=**2)  self**.**bn3 **=** nn**.**BatchNorm2d(32)  *# Number of Linear input connections depends on output of conv2d layers*  *# and therefore the input image size, so compute it.*  **def** **conv2d\_size\_out**(size, kernel\_size **=** 5, stride **=** 2):  **return** (size **-** (kernel\_size **-** 1) **-** 1) **//** stride **+** 1  convw **=** conv2d\_size\_out(conv2d\_size\_out(conv2d\_size\_out(w)))  convh **=** conv2d\_size\_out(conv2d\_size\_out(conv2d\_size\_out(h)))  linear\_input\_size **=** convw **\*** convh **\*** 32  self**.**head **=** nn**.**Linear(linear\_input\_size, outputs)  *# Called with either one element to determine next action, or a batch*  *# during optimization. Returns tensor([[left0exp,right0exp]...]).*  **def** **forward**(self, x):  x **=** F**.**relu(self**.**bn1(self**.**conv1(x)))  x **=** F**.**relu(self**.**bn2(self**.**conv2(x)))  x **=** F**.**relu(self**.**bn3(self**.**conv3(x)))  **return** self**.**head(x**.**view(x**.**size(0), **-**1)) |

### 3.3.2流程图

下一层

ReLU层

BN层

卷积层

层的输入

### 3.3.3详细说明

输入：

**def** \_\_init\_\_(self, h, w, outputs):

h: screen height,

w: screen width,

outputs: number of actions from gym action space,

卷积层：特征提取

class torch.nn.Conv2d(in\_channels: int, out\_channels: int, kernel\_sizes: Union[T, Tuple[T,T]], stride: Union[ T,Tuple[T,T]])

in\_channels:输入数据体的深度。当图片为RGB时为3，否则为1。

Out\_channels:输出数据体的深度，与使用的滤波器数量一致。

Kernel\_sizes:滤波器(filter)的大小，单位为pixel。

Stride： 步长。每次滤波器在输入矩阵上移动，进行点积运算的长度。

Batch Normalization层：对输入激活函数的输入进行归一化，解决输入数据发生偏移和增大的影响

class torch.nn.BatchNorm2d(num\_features)

num\_features:特征数量，上一层卷积层的输出。

输出大小计算：

**def** **conv2d\_size\_out**(size, kernel\_size **=** 5, stride **=** 2)

参数：滤波器数量K,空间尺寸F(filter)，滑动步长S(stride)，零填充数量P（padding）

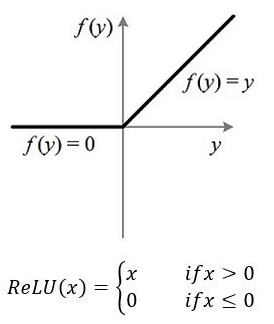
一次过滤后输出体的尺寸Width2\*Height2\*Depth2

width2 = [(width1-F+2P)/S] +1

height2 = [(height1-F+2P)/S]+1

前向传播：

激活函数：ReLU



注：（不需要池化层的原因）

池化函数使用某一位置的相邻输出的总体统计特征来代替网络在该位置的输出。当输入做出平移的时候，池化帮助输入的表示近似不变（平移不变性）。这个性质，在我们关心某个特征是否出现，而不关心具体位置的时候有用。但是在DQN的游戏图像里面，是需要知道特征的具体位置的。

## 逻辑模块3: 输入提取

### 3.4.1代码

|  |
| --- |
| resize **=** T**.**Compose([T**.**ToPILImage(),  T**.**Resize(40, interpolation**=**Image**.**CUBIC),  T**.**ToTensor()])  **def** **get\_cart\_location**(screen\_width):  world\_width **=** env**.**x\_threshold **\*** 2  scale **=** screen\_width **/** world\_width  **return** int(env**.**state[0] **\*** scale **+** screen\_width **/** 2.0) *# MIDDLE OF CART*  **def** **get\_screen**():  *# Returned screen requested by gym is 400x600x3, but is sometimes larger*  *# such as 800x1200x3. Transpose it into torch order (CHW).*  screen **=** env**.**render(mode**=**'rgb\_array')**.**transpose((2, 0, 1))  *# Cart is in the lower half, so strip off the top and bottom of the screen*  \_, screen\_height, screen\_width **=** screen**.**shape  screen **=** screen[:, int(screen\_height**\***0.4):int(screen\_height **\*** 0.8)]  view\_width **=** int(screen\_width **\*** 0.6)  cart\_location **=** get\_cart\_location(screen\_width)  **if** cart\_location **<** view\_width **//** 2:  slice\_range **=** slice(view\_width)  **elif** cart\_location **>** (screen\_width **-** view\_width **//** 2):  slice\_range **=** slice(**-**view\_width, **None**)  **else**:  slice\_range **=** slice(cart\_location **-** view\_width **//** 2,  cart\_location **+** view\_width **//** 2)  *# Strip off the edges, so that we have a square image centered on a cart*  screen **=** screen[:, :, slice\_range]  *# Convert to float, rescale, convert to torch tensor*  *# (this doesn't require a copy)*  screen **=** np**.**ascontiguousarray(screen, dtype**=**np**.**float32) **/** 255  screen **=** torch**.**from\_numpy(screen)  *# Resize, and add a batch dimension (BCHW)*  **return** resize(screen)**.**unsqueeze(0)**.**to(device)  env**.**reset()  plt**.**figure()  plt**.**imshow(get\_screen()**.**cpu()**.**squeeze(0)**.**permute(1, 2, 0)**.**numpy(),interpolation**=**'none')  plt**.**title('Example extracted screen')  plt**.**show() |

### 3.4.2详细说明

以上代码用于从环境中提取和处理渲染图像。使用了torchvision程序包。

## 接口模块1: 实例模型及优化器

### 3.5.1代码

GAMMA **=** 0.999

EPS\_START **=** 0.9

EPS\_END **=** 0.05

EPS\_DECAY **=** 200

TARGET\_UPDATE **=** 10

*# Get screen size so that we can initialize layers correctly based on shape*

*# returned from AI gym. Typical dimensions at this point are close to 3x40x90*

*# which is the result of a clamped and down-scaled render buffer in get\_screen()*

init\_screen **=** get\_screen()

\_, \_, screen\_height, screen\_width **=** init\_screen**.**shape

n\_actions **=** env**.**action\_space**.**n

policy\_net **=** DQN(screen\_height, screen\_width, n\_actions)**.**to(device)

target\_net **=** DQN(screen\_height, screen\_width, n\_actions)**.**to(device)

target\_net**.**load\_state\_dict(policy\_net**.**state\_dict())

target\_net**.**eval()

optimizer **=** optim**.**RMSprop(policy\_net**.**parameters())

memory **=** ReplayMemory(10000)

steps\_done **=** 0

**def** **select\_action**(state):

**global** steps\_done

sample **=** random**.**random()

eps\_threshold **=** EPS\_END **+** (EPS\_START **-** EPS\_END) **\*** \

math**.**exp(**-**1. **\*** steps\_done **/** EPS\_DECAY)

steps\_done **+=** 1

**if** sample **>** eps\_threshold:

**with** torch**.**no\_grad():

**return** policy\_net(state)**.**max(1)[1]**.**view(1, 1)

**else**:

**return** torch**.**tensor([[random**.**randrange(n\_actions)]], device**=**device, dtype**=**torch**.**long)

episode\_durations **=** []

**def** **plot\_durations**():

plt**.**figure(2)

plt**.**clf()

durations\_t **=** torch**.**tensor(episode\_durations, dtype**=**torch**.**float)

plt**.**title('Training...')

plt**.**xlabel('Episode')

plt**.**ylabel('Duration')

plt**.**plot(durations\_t**.**numpy())

*# Take 100 episode averages and plot them too*

**if** len(durations\_t) **>=** 100:

means **=** durations\_t**.**unfold(0, 100, 1)**.**mean(1)**.**view(**-**1)

means **=** torch**.**cat((torch**.**zeros(99), means))

plt**.**plot(means**.**numpy())

plt**.**pause(0.001) *# pause a bit so that plots are updated*

**if** is\_ipython:

display**.**clear\_output(wait**=True**)

display**.**display(plt**.**gcf())

### 3.5.2功能描述

get\_screen()-获取输入图像的大小

n\_actions **=** env**.**action\_space**.**n-从gym action space里面获取一系列动作

optimizer **=** optim**.**RMSprop(policy\_net**.**parameters())-优化器

memory **=** ReplayMemory(10000)-记录历史游戏帧

policy\_net = DQN(screen\_height, screen\_width, n\_actions).to(device)-预测网络

target\_net = DQN(screen\_height, screen\_width, n\_actions).to(device)-目标网络

def select\_action(state)-根据epsilon贪婪策略选择一个操作。选择动作的概率开始于EPS\_START并将朝指数衰减EPS\_END，EPS\_DECAY控制衰减率。

def plot\_durations – 绘制episode（情景）持续时间及最近100个episodes的平均值。在训练循环下面绘制，每个episode更新一次。

### 3.5.3接口描述

|  |  |  |  |
| --- | --- | --- | --- |
| 名称 | In/Out/parameter | Size/precision | Description/Value |
| Batch\_size | Parameter | 128 | 随机梯度下降最小批量采样样本大小 |
| GAMMA | Parameter | 0.999 | 折扣系数 |
| EPS\_START | Parameter | 0.9 | 选择随即动作的概率 |
| EPS\_END | Parameter | 0.05 | 指数衰减结束的概率 |
| EPS\_DECAY | Parameter | 200 | 衰减率 |
| TARGET\_UPDATE | parameter | 10 | 目标网络更新频率 |
| sreen\_height | In | get\_screen() | 图像高度 |
| screen\_width | In | get\_screen() | 图像宽度 |
| n\_actions | in | env**.**action\_space**.**n | 一系列动作 |

## 逻辑模块3: 训练

### 3.6.1代码

**def** **optimize\_model**():

**if** len(memory) **<** BATCH\_SIZE:

**return**

transitions **=** memory**.**sample(BATCH\_SIZE)

batch **=** Transition(**\***zip(**\***transitions))

non\_final\_mask **=** torch**.**tensor(tuple(map(**lambda** s: s **is** **not** **None**,

batch**.**next\_state)), device**=**device, dtype**=**torch**.**bool)

non\_final\_next\_states **=** torch**.**cat([s **for** s **in** batch**.**next\_state

**if** s **is** **not** **None**])

state\_batch **=** torch**.**cat(batch**.**state)

action\_batch **=** torch**.**cat(batch**.**action)

reward\_batch **=** torch**.**cat(batch**.**reward)

state\_action\_values **=** policy\_net(state\_batch)**.**gather(1, action\_batch)

next\_state\_values **=** torch**.**zeros(BATCH\_SIZE, device**=**device)

next\_state\_values[non\_final\_mask] **=** target\_net(non\_final\_next\_states)**.**max(1)[0]**.**detach()

*# Compute the expected Q values*

expected\_state\_action\_values **=** (next\_state\_values **\*** GAMMA) **+** reward\_batch

*# Compute Huber loss*

loss **=** F**.**smooth\_l1\_loss(state\_action\_values, expected\_state\_action\_values**.**unsqueeze(1))

*# Optimize the model*

optimizer**.**zero\_grad()

loss**.**backward()

**for** param **in** policy\_net**.**parameters():

param**.**grad**.**data**.**clamp\_(**-**1, 1)

optimizer**.**step()

num\_episodes **=** 50

**for** i\_episode **in** range(num\_episodes):

env**.**reset()

last\_screen **=** get\_screen()

current\_screen **=** get\_screen()

state **=** current\_screen **-** last\_screen

**for** t **in** count():

action **=** select\_action(state)

\_, reward, done, \_ **=** env**.**step(action**.**item())

reward **=** torch**.**tensor([reward], device**=**device)

last\_screen **=** current\_screen

current\_screen **=** get\_screen()

**if** **not** done:

next\_state **=** current\_screen **-** last\_screen

**else**:

next\_state **=** **None**

memory**.**push(state, action, next\_state, reward)

state **=** next\_state

optimize\_model()

**if** done:

episode\_durations**.**append(t **+** 1)

plot\_durations()

**break**

**if** i\_episode **%** TARGET\_UPDATE **==** 0:

target\_net**.**load\_state\_dict(policy\_net**.**state\_dict())

print('Complete')

env**.**render()

env**.**close()

plt**.**ioff()

plt**.**show()

### 3.6.2详细说明

def optimize\_model():

随机取样-transitions **=** memory**.**sample(BATCH\_SIZE)

计算- state\_action\_values **=** policy\_net(state\_batch)**.**gather(1, action\_batch)

计算-

next\_state\_values **=** torch**.**zeros(BATCH\_SIZE, device**=**device)

next\_state\_values[non\_final\_mask] **=** target\_net(non\_final\_next\_states)**.**max(1)[0]**.**detach()

计算loss-

loss **=** F**.**smooth\_l1\_loss(state\_action\_values, expected\_state\_action\_values**.**unsqueeze(1))

反向传播-loss**.**backward()

训练的循环：

1.随机选择一个动作执行- action **=** select\_action(state)

2.获得环境反馈和下一个状态- last\_screen **=** current\_screen；current\_screen **=** get\_screen()

3.将获得的状态转移，存储到经验池（Memory）-memory**.**push(state, action, next\_state, reward)

4.执行优化函数（optimize\_model()），更新Q网络和权重