بسمه تعالی

Simulation of Pole Balancing Problem

تهیه کننده : ایمان دهقان ابراهیمی

دانشجوی دکتری هوش مصنوعی و رباتیکز

شماره دانشجویی: ۹۶۰۲۸۶۵۱۸

مربوط به درس یادگیری تقویتی

دکتر علی احمدی

مساله pole balancing که به نام های زیر نیز شناخته می شود، یکی از بهترین مسائل یادگیری تقویتی است که در ادامه به تعریف مساله پرداخته و سپس چگونگی پیاده سازی آن در متلب توضیح داده می شود.

- پاندول معکوس (Inverted Pendulum)
- آونگ معکوس
- cart Pole Balancing

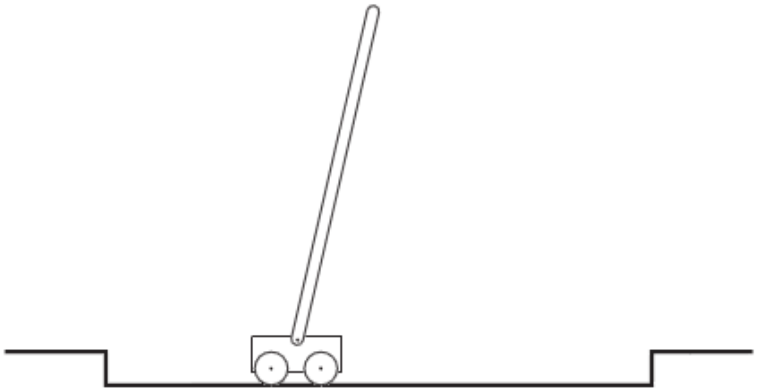


Figure 3.2: The pole-balancing task.

سیستم پاندول معکوس از یک آونگ متصل به یک ارابه (cart) تشکیل شده است، به طوریکه ارابه با نیروی اعمال شونده به سمت چپ و راست حرکت می کند.هدف قراردادن آونگ در وضعیت قائم رو به بالا و حفظ آن در همان موقعیت می باشد، ضمن اینکه ارابه نیز باید حتی الامکان در مرکز خط ریل قرار گیرد و به گوشه ها برخورد نکند.

در این مثال failure بدین صورت تعریف می شود:اگر زاویه آونگ از یک حدی تجاوز کرده یا ارابه از محدوده تعریف شده خارج شود (به اطراف برخورد کند) .دراین صورت ارابه به صورت عمودی در می آید. به تعادل رساندن ارابه می تواند اپیزودیک (مرحله به مرحله) باشد و این اپیزودها تکرار می شوند تا پارامترهای مرتبط با action به درستی تعیین شده و ارابه به تعادل برسد.

پاداش در هر مرحله می تواند عدد ۱+ به ازای هر مرحله (step) تعیین شود تا زمانیکه failure اتفاق نیفتد. درنتیجه جمع پاداش در حقیقت تعداد مرحله ها (step ها) تا رخداد failure تعریف می شود.

بمنظور توسعه یادگیری تقویتی می توان بعدها به صورت مداوم (continuing task) از روش discounting نیز استفاده کرد. در این روش هربار رخداد failure معادل با ۱– و سایر حالات معادل ۰ می باشد. در این حالت پارامتر returnبه صورت $-\gamma^{K}$تعریف می شود که در آن K تعداد مرحله ها قبل از رخداد failure می باشد.

در هردوروش به جهت عمود نگه داشته شدن آونگ برای بیشترین مدت زمان ممکن ، return بیشینه میشود.

شبیه سازی به زبان متلب این مساله را پیاده سازی کرده و از روش sarsa استفاده کرده است و متعلق به آقای مارتین می باشد. (Jose Antonio Martin H. <jamartinh@fdi.ucm.es>)

اصلی ترین متدی که شبیه سازی را اجرامی کند Demo.m می باشد و توسط تایپ عبارت Demo() نیز شبیه سازی اجرا خواهد شد.

پس از تعریف مقادیر اولیه متد اصلی این شبیه سازی با عنوان Cart_PoleDemo(maxepisodes) اجرا می شود و به عنوان پارامتر ورودی مقدار حداکثر تعداد مراحل قابل اجرا برای آن ارسال می شود .

پارامترهای اولیه در این قسمت مقدار دهی می شوند،

```
maxsteps    = 2000;            % maximum number of steps per episode (1000)
statelist   = BuildStateList();  % the list of states
actionlist  = BuildActionList(); % the list of actions

nstates     = size(statelist,1);
nactions    = size(actionlist,1);
Q           = BuildQTable( nstates,nactions );  % the Qtable

alpha       = 0.3;    % learning rate
gamma       = 1.0;    % discount factor
epsilon     = 0.001;  % probability of a random action selection
grafica     = false; % indicates if display the graphical interface
```

alpha : نرخ یادگیری

gamma : فاکتور discount

epsilon : احتمال انتخاب عمل (action) تصادفی

grafica : تعیین اینکه محیط گرافیکی نمایش داده شود یا خیر

در ادامه کد به تعداد اپیزودها به ازای هر اپیزود متد Episode فراخوانی می شود و در ادامه جزئیات این متد و متدهای دیگر به صورت مجزا توضیح داده خواهد شد.

```
for i=1:maxepisodes
      [total_reward,steps,Q ] = Episode( maxsteps, Q , alpha,
      gamma,epsilon,statelist,actionlist,grafica );

    disp(['Espisode: ',int2str(i),'  Steps:',int2str(steps),'
Reward:',num2str(total_reward),' epsilon: ',num2str(epsilon)])

    epsilon = epsilon * 0.99;
    xpoints(i)=i-1;
    ypoints(i)=steps;
    subplot(2,1,2);
    plot(xpoints,ypoints)
    title(['Episode: ',int2str(i),' epsilon: ',num2str(epsilon)])
    xlabel('Episodes')
    ylabel('Steps')
    drawnow

    if (i>1000) grafica=true;//درصورتیکه به نتیجه رسید محیط گرافیکی فعال میشود.
    end
end
```

متد Episode: تمامی پارامترها را به عنوان ورودی گرفته و در آن یک عمل انجام می دهد، پاداش می گیرد، پاداش تجمیعی را محاسبه می کند

```
function [ total_reward,steps,Q ] = Episode( maxsteps, Q , alpha,
gamma,epsilon,statelist,actionlist,grafic )

global grafica
% state variables x,x_dot,theta,theta_dot
x            = [0 0 0 0.01];
steps        = 0;
total_reward = 0;

% convert the continous state variables to an index of the statelist
s   = DiscretizeState(x,statelist);
%s    = get_box(x);
% selects an action using the epsilon greedy selection strategy
a   = e_greedy_selection(Q,s,epsilon);

for i=1:maxsteps

    % convert the index of the action into an action value
    action = actionlist(a);

    %do the selected action and get the next car state
    xp  = DoAction( action , x );

    % observe the reward at state xp and the final state flag
    [r,f]   = GetReward(xp);
    total_reward = total_reward + r;

    % convert the continous state variables in [xp] to an index of the statelist
    sp  = DiscretizeState(xp,statelist);
    %sp  = get_box(xp);
    % select action prime
    ap = e_greedy_selection(Q,sp,epsilon);

    % Update the Qtable, that is,  learn from the experience
    Q = UpdateSARSA( s, a, r, sp, ap, Q , alpha, gamma );

    %update the current variables
    s = sp;     a = ap;     x = xp;

    %increment the step counter.
    steps=steps+1;

   grafic=grafica;
    % Plot of the mountain car problem
    if (grafic==true)
       plot_Cart_Pole(x,action,steps);
    end

    % if the car reachs the goal breaks the episode
    if (f==true)
        break
    end

end
```

دراین قسمت عمل بعدی که قرار است انجام شود توسط DoAction(action , x) انجام شده و پس از انجام عمل پاداش عمل انجام شده توسط [r,f] = GetReward(x) محاسبه شده و به total_reward اضافه می شود. سپس اطلاعات مربوط به تجربه کسب شده بروزرسانی می شود. و این عمل برای هرمرحله تکرار می‌شود.

انجام عملیات نمایش حالت ارابه، پاندول و عمل انجام شده به صورت گرافیکی توسط plot_Cart_Pole صورت می‌پذیرد.

برای محاسبه دقیق عمل انجام شده نیاز به پارامترهای زیر داریم که توضیح هر پارامتر در جدول زیر آورده شده است:

Table 1: System parameters and variable names for the pole balancing problem.

Symbol	Name	Description
θ	Pole Angle	measured (in radians) relatively to the upright position, initial value in $[-0.1, +0.1]$
$\dot{\theta}$	Pole Velocity	angular velocity of the pole in rad/s
$\ddot{\theta}$	Pole Accelaration	acceleration of the pole in rad/s^2
x	Card Position	measured relatively to the middle of the track (in m), initial value in $[-1, +1]$
$\dot{x}$	Card Velocity	velocity of the cart (in m/s)
$\ddot{x}$	Card Acceleration	acceleration of the cart (in m/s)
g	Gravitational Acceleration	acceleration due to gravity (g =9.81 m/s^2)
m_c	Cart Mass	1.0 kg
m_p	Pole Mass	0.1 kg
l	Pole Length	distance from pivot to the pole's center of mass (l=0.5m)
t	Time	measured in s
F_t	Force	force applied to the cart at time t (in N, always $F_t \neq 0$ for a bang-bang controller)
h	Track Limit	$\pm$2.4m from track center
r	Pole Failure Angle	$\pm 12°$ from vertical ($12° \approx 0.209$rad)
τ	Time Step	discrete integration time step for the simulation ($\tau = 0.02$s)
F_m	Controller Constant	constant of linear controller (set to F_m = 100N)
k_1, k_2, k_3, k_4	Controller Constants	further constants of controller (in $[0, 1]$, to be optimized)

متد DoAction عمل مورد نظر را انجام داده و نتیجه را طبق محاسبات فیزیک محاسبه کرده و در متغیری با نام s قرار می دهد.

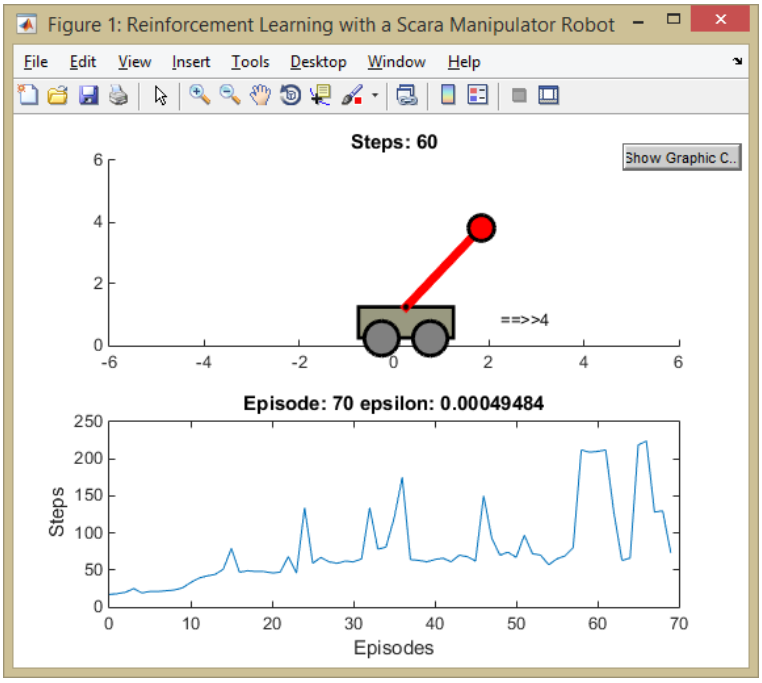
```
function [s] = DoAction(action,s)
%  Cart_Pole:  Takes an action (0 or 1) and the current values of the
%  four state variables and updates their values by estimating the state
%  TAU seconds later.

% Parameters for simulation
x          = s(1);
x_dot      = s(2);
theta      = s(3);
theta_dot  = s(4);


g               = 9.8;      %Gravity
Mass_Cart       = 1.0;      %Mass of the cart is assumed to be 1Kg
Mass_Pole       = 0.1;      %Mass of the pole is assumed to be 0.1Kg
Total_Mass      = Mass_Cart + Mass_Pole;
Length          = 0.5;      %Half of the length of the pole
PoleMass_Length = Mass_Pole * Length;
Force_Mag       = 10.0;
Tau             = 0.02;     %Time interval for updating the values
Fourthirds      = 4.0/3.0;

force = action * Force_Mag;

temp     = (force + PoleMass_Length * theta_dot * theta_dot * sin(theta)) /
Total_Mass;
thetaacc = (g * sin(theta) - cos(theta) * temp) / (Length * (Fourthirds -
Mass_Pole * cos(theta) * cos(theta) / Total_Mass));
xacc     = temp - PoleMass_Length * thetaacc * cos(theta) / Total_Mass;

% Update the four state variables, using Euler's method.
x         = x + Tau * x_dot;
x_dot     = x_dot + Tau * xacc;
theta     = theta + Tau * theta_dot;
theta_dot = theta_dot+Tau*thetaacc;

s = [x x_dot theta theta_dot];
```

متد GetReward(s): مقدار پاداش را تعیین می کند.

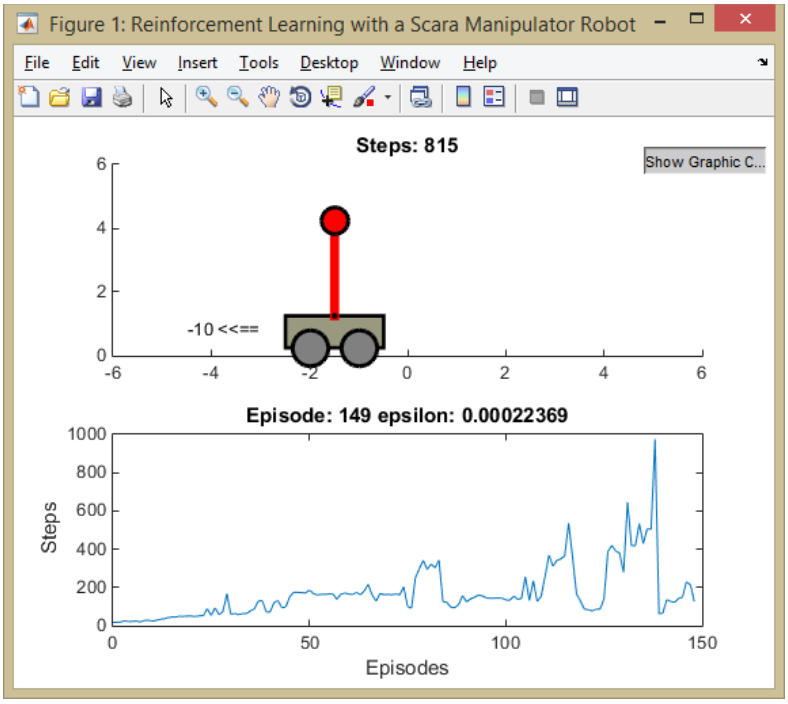
```
function [ r,f ] = GetReward( s )
% r: the returned reward.
% f: true if the car reached the goal, otherwise f is false

x         = s(1);
x_dot     = s(3);
theta     = s(3);
theta_dot = s(4);

r = 10 - 10*abs(10*theta)^2 - 5*abs(x) - 10*theta_dot;
f = false;


twelve_degrees     = deg2rad(12); % 12°
fourtyfive_degrees = deg2rad(45); % 45°
%if (x < -4.0 | x > 4.0  | theta < -twelve_degrees | theta >
twelve_degrees)
if (x < -4.0 | x > 4.0  | theta < -fourtyfive_degrees | theta >
fourtyfive_degrees)
    r = -10000 - 50*abs(x) - 100*abs(theta);
    f = true;
end
```

اجرای شبیه سازی

همانطور که در تصویر مشاهده می کنید در مرحله های اول شبیه سازی زاویه پاندول به مقدار حداکثر مجاز خود رسیده و به failure می رسیم که بلافاصله ارابه به حالت اولیه باز می‌گردد. مثلا در مثال زیر طی سپری کردن ۶۰ قدم به failure رسیدیم.



در یکی از اپیزودهای زیر (اپیزود ۱۵۰) پس از طی کردن ۸۱۵ عمل انجام شده، ارابه به تعادل نسبی رسیده است.



نتایج اجرا در مرحله های اولیه : همانطور که مشاهده می شود، پاداش منفی می باشد.

Espisode: 95 Steps:58 Reward:-12741.9602 epsilon: 0.00038878
Espisode: 96 Steps:70 Reward:-15814.2442 epsilon: 0.0003849
Espisode: 97 Steps:71 Reward:-14239.0772 epsilon: 0.00038105
Espisode: 98 Steps:60 Reward:-13091.5997 epsilon: 0.00037724
Espisode: 99 Steps:80 Reward:-14161.8783 epsilon: 0.00037346
Espisode: 100 Steps:67 Reward:-13333.6392 epsilon: 0.00036973

نتایج اجرا در مراحل پایانی : همانطور که مشاهده می شود پاداش مثبت و به حداکثر مقدار خود رسیده است.

Espisode: 495 Steps:2000 Reward:19004.1705 epsilon: 6.9789e-06
Espisode: 496 Steps:2000 Reward:19004.1705 epsilon: 6.9091e-06
Espisode: 497 Steps:2000 Reward:19004.1705 epsilon: 6.84e-06
Espisode: 498 Steps:2000 Reward:19004.1705 epsilon: 6.7716e-06
Espisode: 499 Steps:2000 Reward:19004.1705 epsilon: 6.7039e-06
Espisode: 500 Steps:2000 Reward:19004.1705 epsilon: 6.6369e-06

زمانیکه ارابه به تعادل می رسد در حقیقت step ها به بینهایت میل می کند به همین منظور یک آستانه بالا برای تعداد عملیات موفق در نظر گرفته شده است. در این شبیه سازی آستانه ۲۰۰۰ در نظر گرفته شده و زمانیکه عملیات موفق بدون رخداد failure به ۲۰۰۰ می رسد، اپیزود به اتمام رسیده و از آن جا به بعد سیستم فرا گرفته است با چه پارامترهایی ارابه را به حالت تعادل نگهدارد.