Learning to Balance an Inverted Pendulum

Nanda Kishore Mallapragada, Sai Keerthy Kakarla, Venkata Yuktal Bobba
Arizona State University

***Abstract*—** Design of a learning algorithm to balance an inverted pendulum by using Particle Swam Optimization algorithm. The learning performance of the system is tested for several cases and the results are updated.

Key words: We used OpenAI to design our project.

I. Introduction

Reinforcement learning is an area of machine learning where software agent learns how to take actions in the interactive environment with trial and error method to increase the cumulative reward [6]. It has become very popular because of its wide spread applications and the recent breakthroughs it has achieved. Applications range from Game playing (Atari) to Self-Driving Cars [5]. It also is convenient because constant supervision is not required because of its efficient algorithms and its trial and error learning method. In our proposed method, we used Particle Swarm Optimization to solve this problem. Swarm algorithm depends on heuristic searching principle in the solution process. The environment model also is not necessary to be known beforehand.

II. CART POLE PROBLEM

The cart pole problem or the inverted pendulum problem is an environment consisting of a cart, a pole and forces and constraints on the model.

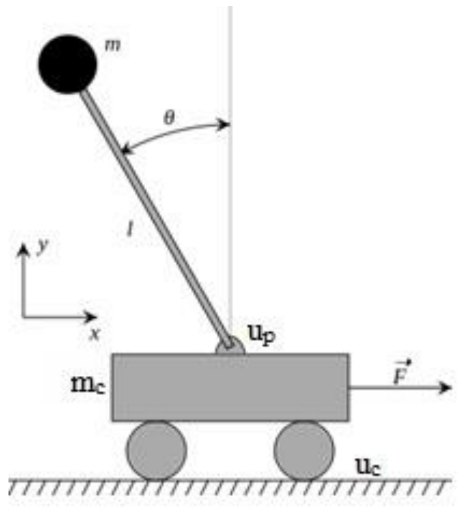


Fig 1. Cart pole diagram

The challenge is to self-balance the pole on the cart (pole must be perpendicular to the cart) during forces applied on the cart. The design of the cart pole has been taken from [1]. Fig. 1 defines a cart pole diagram for which we are trying to balance the pole using an optimization algorithm.

$$\frac{d^2\theta}{d^2t} = \frac{g\sin\theta + \cos\theta[-F - ml\dot{\theta}^2\sin\theta + \mu_c\,\mathrm{sgn}(\dot{x})] - \dfrac{\mu_p\dot{\theta}}{ml}}{l\left(\dfrac{4}{3} - \dfrac{m\cos^2\theta}{m_c + m}\right)} \quad (1)$$

$$\frac{d^2x}{d^2t} = \frac{F + ml[\dot{\theta}^2\sin\theta - \ddot{\theta}\cos\theta] - \mu_c\,\mathrm{sgn}(\dot{x})}{m_c + m} \quad (2)$$

The equations (1) & (2) pertaining to the cart-pole problem as described above where

g = 9.8 m/s^2, acceleration due to gravity
m_c=1.0 kg, mass of cart(M indicated in the diagram)
m = 0.1 kg, mass of pole
l= 0.5 m, half-pole length
u_c = 0.0005, coefficient of friction of cart on track
u_p =0.000002, coefficient of friction of pole on cart
F= + or - 10 Newton's, force applied to cart's center of mass
where the sgn(x) is defined as

$$\mathrm{sgn}(x) = \begin{cases} 1, & \text{if } x > 0 \\ 0, & \text{if } x = 0 \\ -1, & \text{if } x < 0. \end{cases} \quad (3)$$

The nonlinear differential equations (1) and (2) are solved numerically by Runge-Kutta method.
This model provides four state variables
x(t) = position of the cart on the track
Θ(t)=angle of the pole with respect to the vertical position
x_dot(t)=cart velocity
Θ_dot(t)=angular velocity of the pole

III. STATES AND CONDITIONS DEFINED

1. Each time step is 0.02 sec.
2. A pole is considered fallen when the pole is outside the range of [-12deg, 12deg] and if the cart is beyond the

range of [-2.4, 2.4] m in reference to the central position on the track.
3.A run consists of 1000 consecutive trails.
4.Trail is the process from start to fall.
5.If the last trail of the run has lasted for 60,000 timesteps, then it is considered as a successful trial.
6.If the controller is unable to learn to balance the cart-pole within 1000 trails, then the run is considered unsuccessful[5].

IV. PARTICLE SWARM OPTIMIZATION (PSO)

Particle swarm optimization is a population based stochastic optimization technique.

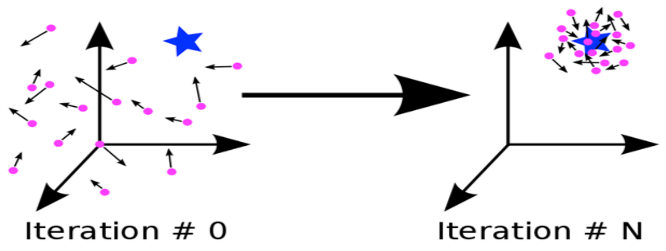
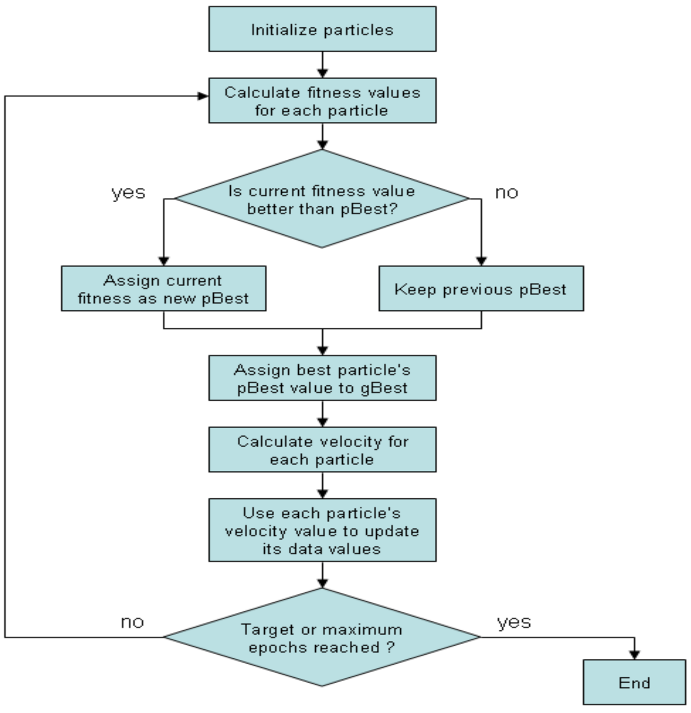


Fig 2. Particle swarm reaching the Global Minimum



Initialize particles
Calculate fitness values for each particle
Is current fitness value better than pBest?
yes
no
Assign current fitness as new pBest
Keep previous pBest
Assign best particle's pBest value to gBest
Calculate velocity for each particle
Use each particle's velocity value to update its data values
Target or maximum epochs reached ?
no
yes
End

Fig 3. Algorithm Flowchart

It is similar to the social behavior of bird flocking. The birds are in random positions and the food is available at a position whose location is unknown but they are clear how far it is present. All the birds follow the one closest to the food and reach the endpoint as in the Fig. 2. Similarly, PSO solves a problem by iteratively trying to improve a candidate solution i.e. moving the candidate solutions around in the search space[3]. Each particle is characterized by position and velocity. The position is evaluated as shown in pseudo code below.

Pseudo code:

```
For each particle
{
        Initialize particle
}
Do until maximum iterations or minimum error criteria
{
        For each particle
        {
                Calculate data fitness value
                If the fitness value is better that Pid
                {
                Set Pid = current fitness value
                }
        If Pid is better than Pgd
        {
                Set Pgd = Pid
        }
}
        For each particle
        {
        Calculate particle velocity
        Use Pgd and Velocity to update particle data.
        }
}
```

p_{id} = *best fitness value of each particle*
p_{gd} = *global best fitness value of all particles*
If local particles arrive at the best solution of the population then p_{gd} is replaced with the local best value p_{Id}. After finding p_{id} and p_{gd} a particle updates its velocity v_{id} and position of current particle x_{id} according to the following equations.

$$v_{id} = wv_{id} + c_1r_1(p_{id} - x_{id}) + c_2r_2(p_{gd} - x_{id}) \qquad (4)$$

$$x_{id} = x_{id} + v_{id} \qquad (5)$$

r1 and r2 are random numbers between 0 and 1. C1 and c2 are acceleration constants (learning factors) w = weights.
Velocity v gives information about the resolution about the solution regions. Higher velocity v may result in the particle missing good solutions whereas low velocity v

may result in particles not sufficiently exploring the solution space [2]. Acceleration constants c_1 and c_2 represent the weighting of the stochastic acceleration terms that pull each particle towards p_{id} and p_{gb} positions. [3]

PSO Algorithm and working of controller:

In our system, we have taken parallel environments with certain set of random parameters initialized for each environment for one single trial. Hence, we run parallel environments to update rewards of the model in a single trial and see if the rewards of every single iteration matches with the best rewards that corresponds to 60000 timesteps. If it reaches that reward, we will stop the iterations and record the index of the trial and we consider it as the number of trials for that particular run. We will repeat this over 20 runs and take the average to decide total number of trials to reach 60000 timesteps.

Algorithm we implemented:

1. We start with class that called EnvWorker and this class inherits Process class and has the attributes and functions

- env_name -> The name of the gym environment we are using, i.e. CartPole-v0.
- pipe -> A shared data structure used to exchange information between the EnvWorker object process and the parent process.
- name -> A string to name the Process.
- NotDisplay -> Boolean value to enable/disable simulation output.
- Initialization function to initialize variables that are to be used.
- Run function is used to start the action of the environment and when the done flag is set, i.e. if the pole tipped off or if it reaches 60000 timesteps. We will update the parameters accordingly.

2. We build a second class in which we execute parallel environments which is similar to previous class. This parallel environment will update the batch parameters. Taking a set of rewards out of parallel environments is the main moto of this project.

3. The function updates the velocities of the particles as mentioned in PSO which can be related to updating the rewards based on the velocities in p [4]. The best coordinates of the particles are changed if the new position gives a higher value of the objective function, and the group best coordinate and objective function value is also updated. So we can finally converge at global optimum.

V. SIMULATION RESULTS

We modified the code obtained from ** and acquired the following results.

Noise type	success rate	# of trials	PSO Parameters
Noise free	100%	24.5	velocity = 0.5, envs = 7;#iterations=100
Uniform 5% Actuator	100%	33.65	velocity = 0.5, envs = 7;#iterations=100
Uniform 10% Actuator	100%	37.8	velocity = 0.5, envs = 7;#iterations=100
Uniform 5% sensor	100%	30.05	velocity = 0.5, envs = 7,#iterations=100
Uniform 10% sensor	100%	31.65	velocity = 0.7, envs = 7,#iterations=500
Gaussian Var = 0.1 sensor	100%	39.85	velocity = 0.5, envs = 5,#iterations=100
Gaussian Var = 0.2 sensor	80%	153	velocity = 0.9, envs = 10, #iterations= 1000

** https://rishav1.github.io/rein_learning/2017/01/05/simple-swarm-intelligence-optimization-for-cartpole-balancing-problem.html

We have added actuator noise using u(t) = u(t) + p, where p is uniformly distributed random variable and u(t) is action network. We have changed the force variable in cartpole.py for actuator noise. We have added sensor noise using theta = theta* (1+random(-noise percentage, noise percentage)) and recorded the simulation results in the above table.

The objective is to balance the pole for 60000 time steps for all the conditions defined in the above table. For a Noise free environment taking the number of parallel environments as 7, the velocity parameter as 0.5 and the number of iterations to update the parameters of parallel environment model as 100 we obtained the average number of trails as 24.5 and a success rate of 100% . .For a uniform 5% Actuator noise environment taking the number of parallel environments as 7 , the velocity parameter as 0.5 and the number of iterations to update the parameters of parallel environment model as 100 we obtained the average number of trails as 33.65 and a success rate of 100%. For a uniform 10% Actuator noise environment taking the number of parallel environments as 7 , the velocity parameter as 0.5 and the number of iterations to update the parameters of parallel environment model as 100 we obtained the average number of trails as 37.8 and a success rate of 100%. For a uniform 5% sensor noise environment taking the number of parallel environments as 7, the velocity parameter as 0.5 and the number of iterations to update the parameters of parallel environment model as 100 we obtained the average number of trails as 30.05 and a success rate of 100%. For a uniform 10% sensor noise environment, taking the number of parallel environments as 7, the velocity parameter as 0.7 and the number of iterations to update the parameters of parallel environment model as 500 we obtained the average number of trails as 31.65 and a success rate of 100%. For a Gaussian noise of variance 0.1 for sensor environment, taking the number of parallel environments as 5 , the velocity parameter as 0.5 and the number of iterations to update the parameters of parallel environment model as 100 we obtained the average number of trails as 39.85 and a success rate of 100%. But for the last case where we have Gaussian noise with variance of 0.2 the algorithm took lot of time to execute for even 20 runs and average number of trials is around 153 with success rate of 80%. We tried it with multiple velocities for better results, but we ended up with the parameters as specified and the results are updated in Table.

VI. CONCLUSION

We have successfully implemented the Cart-pole problem with the parameters defined in the [1] and have obtained the results accordingly.

VII. ACKNOWLEDGMENT

We would like to thank the Professor Jenni Si for her constant support during the project. We would also like to thank her for her interesting coursework, the materials she provided and her effort to engage us with the latest technology. A final thanks to all our classmates who have helped us during various situations.

REFERENCES

[1]. Jennie Si, Yu-Tsung Wang, “Online learning control by Association and Reinforcement” , IEEE Transactions on Neural Networks vol. 12, pp. 264-276, ISSN: 1941-0093.

[2].Particle Swarm Optimized Adaptive Dynamic Liu, Proceedings of the 2007 IEEE Symposium on ADPRL 2007Programming, Dongbin Zhao, Jianqiang Yi, Derong.

[3]. Iima, H., Kuroe, Y.: Swarm Reinforcement Learning Algorithms Based on Particle Swarm Optimization. In: IEEE International Conference on Systems, Man and Cybernetics, pp. 1110–1115 (2008).

[4]. Daniel, H., Alexander, H., Thomas A. Runkler, Stefen, U.: Reinforcement Learning wit Particle Swarm Optimization Policy (PSOP) in Continuous state and Action Spaces. In: International Journal of Swarm Intelligence Research, Volume 7, Issue 3, July – September (2016)

[5]. R. Ozakar, G. T. Ozyer and B. Ozyer, "Balancing inverted pendulum using reinforcement algorithms," *2016 24th Signal Processing and Communication Application Conference (SIU)*, Zonguldak, 2016, pp. 1569-1572.

[6]. V. Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, Martin Riedmiller “Playing Atari with Deep Reinforcement Learning”, NIPS DeepLearning workshop 2013.