# Genetic Approach to Actor Critic Algorithm in Ping Pong

[1]Deepak Singh ,B.E Student at GCET Jammu, India.

**Abstract - Actor Critic method where the “Critic” estimates the value function and the “Actor” updates the policy distribution in the direction suggested by the Critic (such as with policy gradients). I will initially discuss about my approach with a simple actor critic solution for the pong game , And then I will go on to discuss some shortcomings , And end with my solution to overcome this issue which is having a genetic approach on top of actor critic model, In which models diverging from the solution are rejected . Initially, I will create 10 actor critic models and then based on the rewards after running an episode for them , I will substitute the poorest performing models with new models with random weights and run optimiser on the top six , until I find a model that crosses the reward threshold . I conclude by discussing how this approach out performs the basic actor critic algorithm and eliminates the issue of diverging too far.**

**Keywords – Critic Algorithm, Genetic, Random model.**

## I. INTRODUCTION

Actor Critic Algorithm was first introduced in 1983 as a Reinforcement Learning Algorithm. In Reinforcement learning , an Agent learns to make correct decisions based on the rewards which it gets from interacting with the environment . Which is in contrast to supervised learning where an agent is given a dataset of correct behaviour to learn from. In the actor critic algorithm, Based on the action taken by the agent, the critic makes an approximation of the possible future rewards. The actor uses the approximation to update its policy in an approximate gradient direction. I will be running this algorithm in a pong game to test how well it does . In the pong environment , I will be collecting episodes with respect to both the paddles in order to have a more diverse set to train the policy on and reach convergence faster.

## II. PONG ENVIRONMENT

In the 2 player pong environment,
I will be Collecting trajectories in the form of state ,reward and if the episode ended(lost the game) .

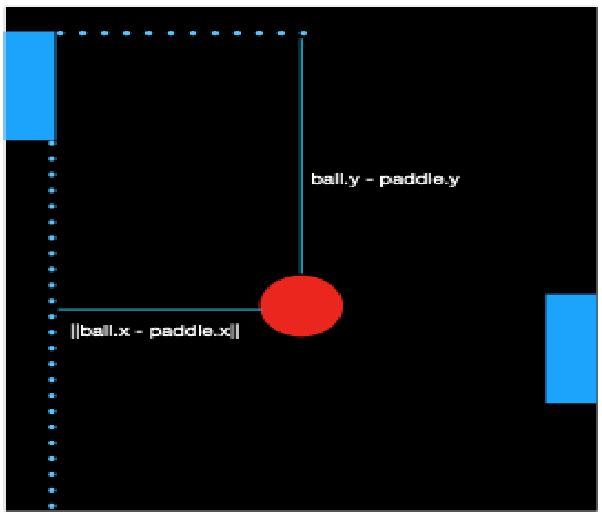


Figure : 1

State will have 3 parameter,Y position of paddle , ball.y - paddle.y and ||ball.x - paddle.x|| .
Reward is calculated based on the fact
1)if it hit the paddle the rewards= 1000 ,
2)if it just in the environment not touching wall or paddles reward =1 ,
3) if it hit the walls=
-((||paddle.y- ball.y||*||paddle.y-ball.y|| )/10)
Giving a -1000 reward for both when the difference between y position of paddle and ball was 2 px and 200px doesn’t make sense so squaring the difference between their y values makes much more sense as negative reward. Episode end is a Boolean if hit the wall is True, else it is False. Trajectories are collected for the 2 paddle separately and later they will be appended together, to get a better collection of trajectories to train the policy.

## III. ACTOR CRITIC ALGORITHM

Actor-Critic methods are temporal difference (TD) learning methods that represent the policy function independent of the value function. The actor is the policy $\pi\theta(a|s)$ with parameters $\theta$ which conducts actions in an environment. The critic computes value functions to help assist the actor in learning. These are usually the state value, state-action value, or advantage value, denoted as V(s) Q(s,a), andA(s,a), respectively. We will be using a basic actor critic method -> Vanilla Policy Gradient with learned baseline function.

**Vanilla Policy Gradient Algorithm**

```
Initialise policy parameter θ ,baseline b For iteration
I=1,2,......do
        Collect a set of trajectory by executing the current
        policy
        At each time step in each trajectory compute and,
        Then return R = $\sum T \, \gamma t'{-}trt'$
        the advantage estimate At = Rt -b(st)
        Refit the baseline by minimising the
        || Rt - b(st)||2 ,

        summed over all trajectories and time step
        Update the policy ,using a policy gradient estimate g,
        which is a sum of terms
        ∇ θlogπ(at ,st , θ )At
End for
```

**The actor-critic loss**

Since a hybrid actor-critic model is used, the chosen loss function is a combination of actor and critic losses for training, as shown below:

$L = L_{actor} + L_{critic}$

The actor loss is based on policy gradients with the critic as a state dependent baseline and computed with single-sample (per- episode) estimates.

$L_{actor} = -\sum_{t=1}^{T} \log \pi(a_t|s_t)[G(a_t|s_t) - V_\pi^\theta(s_t)]$

**Issues with this approach**

Running the particular algorithm over and over in the pong environment , I have seen that some time models will converge in a few iterations of training while in some cases , it will diverge and never reach the optimum solution. I figured that the reason is possibly due to how the weights and biases are assigned , when the model is created each time I run the script, Also there is how well it interact with environment and is able to recognise how to gain positive rewards in the game that can affect how soon it converge or would just go on a tangent , also there is fine tuning of discount factor and learning rate

**Genetic Approach on Actor Critic Model:**

So far I have establish that one of the reasons to reach convergence faster is getting lucky with right weights and biases for our model ,So I figured out an approach that can speed up the process , I will create a list of models , train each model using traditional approach and then test these in the environment and store the score for each model and sort them in descending order. I will eliminate the worst performing model and replace them with the newly initialised model which should remove the possibility of models diverging too far from the solution and can help reach solution faster.

It's Survival of The Fittest , the models that are doing good in comparison to others will be passed to the next generation , while poorly performing ones will be discarded and replaced with new models with randomised policy as to stop wasting time on models that will never converge.

So I created a list of models and scores each model hold .

```
def __init__(self, numActions ,hidden_units )
        self.common = layers.Dense(1024 ,activation = 'relu')
        self.common1= layers.Dense(num_hidden_units ,activation = 'relu')
        self.actor = layers.Dense(num_actions ,activation = "tanh")
        self.critic = layers.Dense(1 )
def call(self,inputs):
        x = self.common(inputs)
        x1 =self.common1(inputs)
        return self.actor(x) , self.critic(x1)

Lt = []
score=0
numAction =2 // move paddle up or down
hidden_units = 32
for i in range(10)
        Lt.append([score ,Model(numActions ,hidden_units),i+1])
```

Also I started Collecting multiple episode for one model to have better set and also I had one policy for for both paddle so I had shared experience to train my model

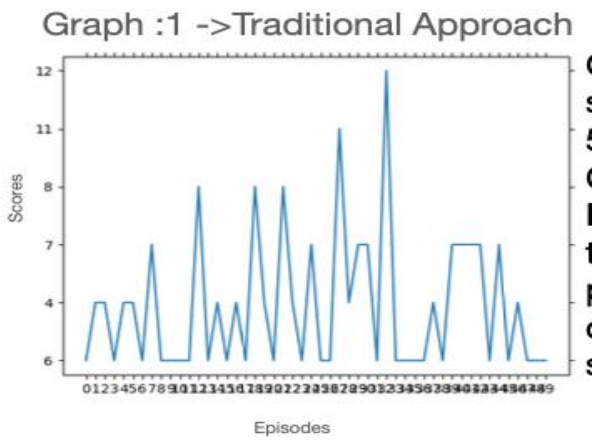
```
For model in the list Run the algorithm
        Initialise policy parameter θ ,baseline b For iteration I=1,2,......do
                Collect a set of trajectory by executing the current policy
                At each time step in each trajectory compute,
                , and
                The return R = ∑T γt′−trt′
                t t′=t
                the advantage estimate At = Rt - b(st)
                Refit the baseline by minimising the || Rt - b(st)||2 , summed over all trajectories and time step
                Update the policy ,using a policy gradient estimate g,
                which is a sum of terms ∇ θlogπ(at ,st , θ )
        At End for
```

After running an episode for each model and updating the policy ,I sorted the model list based on the scores , removing the last 4 models and creating 4 new ones . Resetting the scores to zeroes and running the episode again for each model until a model crosses the reward threshold.

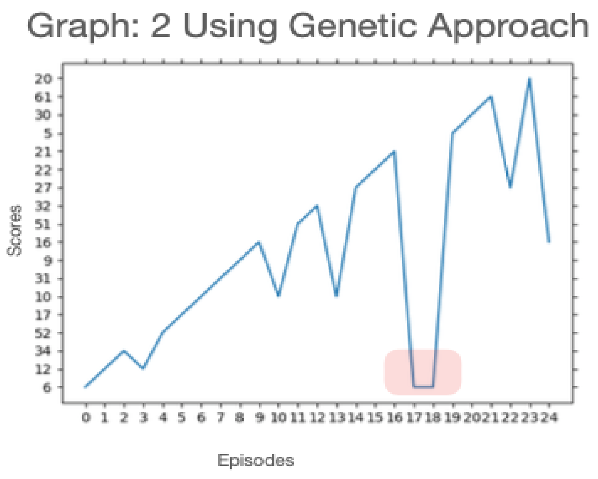
```
Lt = sorted(Lt , key = lambda l : l[0] , reverse = True) Lt = Lt[:5]
for i in range(5):
        Lt.append([0 , Actor_Critic_A(numActions ,hidden_units) , 6+i])
for i in range(10):
        Lt[i][0]=0
```

## IV. Result

So based on my approach lets see how well our model plays the game in comparison to the traditional approach . We will be using rewards as the number of times the pedal successfully sends the ball to the other side in an episode, which ends when the ball touches the wall.After each episode we train the model as discussed above.



Graph 1, depicts the score pattern over 50 episodes in Classical Approach. Here you can notice that the score peaked at 12 and diverged from the solution .



Graph 2 ,depicts the score pattern over 24 episodes using Genetic Approach on Actor Critic.

Here we can see that if the model diverges from the solution they are discarded and replaced with a better performing model.

## V. Conclusion

In Conclusion , Using actor critic algorithm without some sort of clipping will lead to model diverging from the solution , In my approach I let the model explore without clipping which will cause some models to diverge from the solution but will also increase the chances of model doing better and figuring out a way to maximise the reward . Also sharing experience between the paddles made the dataset better on which I was training my model. Since I discard the model deviating from the solution and focus on improving the model doing good .It speeds up the process of finding a solution significantly and even in the worst case scenario you will have a solution but it may take some time.

## REFERENCES

[1] Actor Critic Algorithm,Vijay R. Konda John N. Tsitsiklis Laboratory for Information and Decision Systems, Massachusetts Institute of Technology, Cambridge, MA, 02139.

[2] Playing CartPole with the Actor-Critic Method TensorFlow.

[3]Evaluation of Policy Gradient Methods and Variants on the Cart-Pole Benchmark

[4]Proximal Policy Optimisation Algorithms

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, Oleg Klimov. OpenAI

[5] Reinforcement learning

From Wikipedia, the free encyclopaedia

[6]Policy Gradient Methods for Reinforcement Learning with Function Approximation

[7]SAMPLE EFFICIENT ACTOR-CRITIC WITH EXPERIENCE REPLAY

[8]A Bounded Actor-Critic Reinforcement Learning Algorithm Applied to Airline Revenue Management Ryan J. Lawhead and Abhijit Gosavi Department of Engineering Management and Systems Engineering Missouri University of Science and Technology.