

Tutorial: Actor Critic Implementation

- ME20B072

```
#Import required libraries
```

```
import argparse
import gym
import numpy as np
from itertools import count
from collections import namedtuple

import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torch.distributions import Categorical
```

```
#Set constants for training
```

```
seed = 543
log_interval = 10
gamma = 0.99
```

```
env = gym.make('CartPole-v1')
env.reset(seed=seed)
torch.manual_seed(seed)
```

```
SavedAction = namedtuple('SavedAction', ['log_prob', 'value'])
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283:
DeprecationWarning: `should_run_async` will not call `transform_cell`
automatically in the future. Please pass the result to
`transformed_cell` argument and any exception that happen during
thetransform in `preprocessing_exc_tuple` in IPython 7.17 and above.
    and should_run_async(code)
```

```
/usr/local/lib/python3.10/dist-packages/gym/core.py:317:
DeprecationWarning: WARN: Initializing wrapper in old step API which
returns one bool instead of two. It is recommended to set
`new_step_api=True` to use new step API. This will be the default
behaviour in future.
```

```
    deprecation(
/usr/local/lib/python3.10/dist-packages/gym/wrappers/step_api_compatib
ility.py:39: DeprecationWarning: WARN: Initializing environment in old
step API which returns one bool instead of two. It is recommended to
set `new_step_api=True` to use new step API. This will be the default
behaviour in future.
    deprecation(
```

```

env = gym.make('CartPole-v1')
env.reset(seed=seed)
torch.manual_seed(seed)

SavedAction = namedtuple('SavedAction', ['log_prob', 'value'])

class Policy(nn.Module):
    """
    implements both actor and critic in one model
    """
    def __init__(self):
        super(Policy, self).__init__()
        self.affine1 = nn.Linear(4, 128)

        # actor's layer
        self.action_head = nn.Linear(128, 2)

        # critic's layer
        self.value_head = nn.Linear(128, 1)

        # action & reward buffer
        self.saved_actions = []
        self.rewards = []

    def forward(self, x):
        """
        forward of both actor and critic
        """
        x = F.relu(self.affine1(x))

        # actor: choses action to take from state s_t
        # by returning probability of each action
        action_prob = F.softmax(self.action_head(x), dim=-1)

        # critic: evaluates being in the state s_t
        state_values = self.value_head(x)

        # return values for both actor and critic as a tuple of 2
        values:
            # 1. a list with the probability of each action over the
            action space
            # 2. the value from state s_t
            return action_prob, state_values

model = Policy()
optimizer = optim.Adam(model.parameters(), lr=3e-2)
eps = np.finfo(np.float32).eps.item()

```

```

def select_action(state):
    state = torch.from_numpy(state).float()
    probs, state_value = model(state)

    # create a categorical distribution over the list of probabilities of actions
    m = Categorical(probs)

    # and sample an action using the distribution
    action = m.sample()

    # save to action buffer
    model.saved_actions.append(SavedAction(m.log_prob(action), state_value))

    # the action to take (left or right)
    return action.item()

def finish_episode():
    """
    Training code. Calculates actor and critic loss and performs backprop.
    """
    R = 0
    saved_actions = model.saved_actions
    policy_losses = [] # list to save actor (policy) loss
    value_losses = [] # list to save critic (value) loss
    returns = [] # list to save the true values

    # calculate the true value using rewards returned from the environment
    for r in model.rewards[::-1]:
        # calculate the discounted value
        R = r + gamma * R
        returns.insert(0, R)

    returns = torch.tensor(returns)
    returns = (returns - returns.mean()) / (returns.std() + eps)

    for (log_prob, value), R in zip(saved_actions, returns):
        advantage = R - value.item()

        # calculate actor (policy) loss
        policy_losses.append(-log_prob * advantage)

        # calculate critic (value) loss using L1 smooth loss
        value_losses.append(F.smooth_l1_loss(value, torch.tensor([R])))

```

```

    # reset gradients
    optimizer.zero_grad()

    # sum up all the values of policy_losses and value_losses
    loss = torch.stack(policy_losses).sum() +
    torch.stack(value_losses).sum()

    # perform backprop
    loss.backward()
    optimizer.step()

    # reset rewards and action buffer
    del model.rewards[:]
    del model.saved_actions[:]

def train():
    running_reward = 10

    # run infinitely many episodes
    for i_episode in range(2000):

        # reset environment and episode reward
        state = env.reset()
        ep_reward = 0

        # for each episode, only run 9999 steps so that we don't
        # infinite loop while learning
        for t in range(1, 10000):

            # select action from policy
            action = select_action(state)

            # take the action
            state, reward, done, _ = env.step(action)

            model.rewards.append(reward)
            ep_reward += reward
            if done:
                break

        # update cumulative reward
        running_reward = 0.05 * ep_reward + (1 - 0.05) *
running_reward

        # perform backprop
        finish_episode()

        # log results
        if i_episode % log_interval == 0:

```

```

        print('Episode {} \t Last reward: {:.2f} \t Average reward:
{:.2f}'.format(
            i_episode, ep_reward, running_reward))

```

```

    # check if we have "solved" the cart pole problem
    if running_reward > env.spec.reward_threshold:
        print("Solved! Running reward is now {} and "
            "the last episode runs to {} time
steps!".format(running_reward, t))
        break

```

```

train()

```

```

/usr/local/lib/python3.10/dist-packages/gym/utils/
passive_env_checker.py:241: DeprecationWarning: `np.bool8` is a
deprecated alias for `np.bool_`. (Deprecated NumPy 1.24)
    if not isinstance(terminated, (bool, np.bool8)):

```

```

Episode 0   Last reward: 22.00   Average reward: 10.60
Episode 10  Last reward: 28.00   Average reward: 16.78
Episode 20  Last reward: 42.00   Average reward: 33.66
Episode 30  Last reward: 21.00   Average reward: 31.73
Episode 40  Last reward: 26.00   Average reward: 29.00
Episode 50  Last reward: 150.00  Average reward: 64.74
Episode 60  Last reward: 85.00   Average reward: 87.97
Episode 70  Last reward: 234.00  Average reward: 153.18
Episode 80  Last reward: 44.00   Average reward: 144.46
Episode 90  Last reward: 44.00   Average reward: 101.80
Episode 100      Last reward: 133.00   Average reward: 89.91
Episode 110      Last reward: 60.00   Average reward: 88.15
Episode 120      Last reward: 339.00  Average reward: 178.64
Episode 130      Last reward: 105.00  Average reward: 177.38
Episode 140      Last reward: 34.00   Average reward: 138.97
Episode 150      Last reward: 500.00  Average reward: 155.96
Episode 160      Last reward: 500.00  Average reward: 267.45
Episode 170      Last reward: 500.00  Average reward: 360.77
Episode 180      Last reward: 140.00  Average reward: 312.40
Episode 190      Last reward: 119.00  Average reward: 242.36
Episode 200      Last reward: 154.00  Average reward: 202.31
Episode 210      Last reward: 240.00  Average reward: 192.62
Episode 220      Last reward: 500.00  Average reward: 307.03
Episode 230      Last reward: 500.00  Average reward: 357.78
Episode 240      Last reward: 500.00  Average reward: 390.83
Episode 250      Last reward: 500.00  Average reward: 421.68
Episode 260      Last reward: 165.00  Average reward: 428.27
Episode 270      Last reward: 500.00  Average reward: 438.85
Episode 280      Last reward: 500.00  Average reward: 452.60
Episode 290      Last reward: 500.00  Average reward: 471.62
Solved! Running reward is now 475.6650523591305 and the last episode
runs to 500 time steps!

```

TODO: Write a policy class similar to the above, without using shared features for the actor and critic and compare their performance.

```
#TODO: Write a policy class similar to the above, without using shared features for the actor and critic and compare their performance.
```

```
class UnsharedPolicy(nn.Module):
    def __init__(self):
        super(UnsharedPolicy, self).__init__()
        #TODO: Fill in.
        self.affine1 = nn.Linear(4, 128)
        self.affine2 = nn.Linear(4, 128)

        # actor's layer
        self.action_head = nn.Linear(128, 2)

        # critic's layer
        self.value_head = nn.Linear(128, 1)

        # action & reward buffer
        self.saved_actions = []
        self.rewards = []

    def forward(self, x):
        # TODO: Fill in. For your networks, use the same hidden_size for the layers as the previous policy, that is 128.
        a = F.relu(self.affine1(x))
        action_prob = F.softmax(self.action_head(a), dim=-1)

        b = F.relu(self.affine2(x))
        state_values = self.value_head(b)

        return (action_prob, state_values)

model = UnsharedPolicy()
optimizer = optim.Adam(model.parameters(), lr=3e-2)
eps = np.finfo(np.float32).eps.item()
train()
```

```
/usr/local/lib/python3.10/dist-packages/gym/utils/
passive_env_checker.py:241: DeprecationWarning: `np.bool8` is a
deprecated alias for `np.bool_`. (Deprecated NumPy 1.24)
    if not isinstance(terminated, (bool, np.bool8)):
```

Episode 0	Last reward: 12.00	Average reward: 10.10
Episode 10	Last reward: 15.00	Average reward: 11.25
Episode 20	Last reward: 27.00	Average reward: 23.25
Episode 30	Last reward: 65.00	Average reward: 38.56
Episode 40	Last reward: 58.00	Average reward: 52.85
Episode 50	Last reward: 69.00	Average reward: 63.75
Episode 60	Last reward: 70.00	Average reward: 74.61
Episode 70	Last reward: 90.00	Average reward: 96.04
Episode 80	Last reward: 91.00	Average reward: 94.37
Episode 90	Last reward: 45.00	Average reward: 104.10
Episode 100	Last reward: 92.00	Average reward: 102.63
Episode 110	Last reward: 33.00	Average reward: 76.98
Episode 120	Last reward: 48.00	Average reward: 62.95
Episode 130	Last reward: 38.00	Average reward: 58.60
Episode 140	Last reward: 39.00	Average reward: 52.37
Episode 150	Last reward: 97.00	Average reward: 58.02
Episode 160	Last reward: 92.00	Average reward: 61.71
Episode 170	Last reward: 90.00	Average reward: 78.17
Episode 180	Last reward: 236.00	Average reward: 146.49
Episode 190	Last reward: 204.00	Average reward: 187.63
Episode 200	Last reward: 157.00	Average reward: 197.31
Episode 210	Last reward: 87.00	Average reward: 161.21
Episode 220	Last reward: 76.00	Average reward: 126.82
Episode 230	Last reward: 107.00	Average reward: 103.50
Episode 240	Last reward: 246.00	Average reward: 111.63
Episode 250	Last reward: 106.00	Average reward: 117.63
Episode 260	Last reward: 242.00	Average reward: 132.45
Episode 270	Last reward: 90.00	Average reward: 127.58
Episode 280	Last reward: 127.00	Average reward: 118.08
Episode 290	Last reward: 87.00	Average reward: 118.48
Episode 300	Last reward: 108.00	Average reward: 115.81
Episode 310	Last reward: 71.00	Average reward: 107.64
Episode 320	Last reward: 70.00	Average reward: 94.11
Episode 330	Last reward: 33.00	Average reward: 78.84
Episode 340	Last reward: 35.00	Average reward: 63.81
Episode 350	Last reward: 47.00	Average reward: 60.76
Episode 360	Last reward: 96.00	Average reward: 64.94
Episode 370	Last reward: 87.00	Average reward: 77.99
Episode 380	Last reward: 96.00	Average reward: 87.71
Episode 390	Last reward: 86.00	Average reward: 103.24
Episode 400	Last reward: 114.00	Average reward: 119.55
Episode 410	Last reward: 103.00	Average reward: 121.33
Episode 420	Last reward: 81.00	Average reward: 105.80
Episode 430	Last reward: 88.00	Average reward: 98.00
Episode 440	Last reward: 66.00	Average reward: 90.93
Episode 450	Last reward: 82.00	Average reward: 84.59
Episode 460	Last reward: 72.00	Average reward: 81.34
Episode 470	Last reward: 89.00	Average reward: 87.53
Episode 480	Last reward: 95.00	Average reward: 107.84
Episode 490	Last reward: 125.00	Average reward: 120.12

Episode 500	Last reward: 89.00	Average reward: 118.53
Episode 510	Last reward: 131.00	Average reward: 113.88
Episode 520	Last reward: 61.00	Average reward: 97.19
Episode 530	Last reward: 67.00	Average reward: 84.57
Episode 540	Last reward: 69.00	Average reward: 76.17
Episode 550	Last reward: 66.00	Average reward: 72.13
Episode 560	Last reward: 69.00	Average reward: 69.94
Episode 570	Last reward: 82.00	Average reward: 72.41
Episode 580	Last reward: 84.00	Average reward: 75.45
Episode 590	Last reward: 98.00	Average reward: 82.45
Episode 600	Last reward: 96.00	Average reward: 88.78
Episode 610	Last reward: 93.00	Average reward: 91.20
Episode 620	Last reward: 96.00	Average reward: 92.69
Episode 630	Last reward: 108.00	Average reward: 96.02
Episode 640	Last reward: 102.00	Average reward: 98.25
Episode 650	Last reward: 105.00	Average reward: 100.21
Episode 660	Last reward: 106.00	Average reward: 101.20
Episode 670	Last reward: 40.00	Average reward: 96.00
Episode 680	Last reward: 114.00	Average reward: 101.49
Episode 690	Last reward: 123.00	Average reward: 109.67
Episode 700	Last reward: 130.00	Average reward: 117.51
Episode 710	Last reward: 117.00	Average reward: 128.61
Episode 720	Last reward: 203.00	Average reward: 137.54
Episode 730	Last reward: 500.00	Average reward: 229.63
Episode 740	Last reward: 343.00	Average reward: 317.24
Episode 750	Last reward: 243.00	Average reward: 282.71
Episode 760	Last reward: 245.00	Average reward: 262.58
Episode 770	Last reward: 500.00	Average reward: 312.15
Episode 780	Last reward: 37.00	Average reward: 220.87
Episode 790	Last reward: 23.00	Average reward: 143.95
Episode 800	Last reward: 27.00	Average reward: 97.70
Episode 810	Last reward: 36.00	Average reward: 71.99
Episode 820	Last reward: 39.00	Average reward: 57.92
Episode 830	Last reward: 67.00	Average reward: 56.49
Episode 840	Last reward: 87.00	Average reward: 63.98
Episode 850	Last reward: 116.00	Average reward: 79.84
Episode 860	Last reward: 111.00	Average reward: 93.91
Episode 870	Last reward: 128.00	Average reward: 105.67
Episode 880	Last reward: 140.00	Average reward: 115.14
Episode 890	Last reward: 122.00	Average reward: 121.13
Episode 900	Last reward: 138.00	Average reward: 125.79
Episode 910	Last reward: 137.00	Average reward: 129.77
Episode 920	Last reward: 140.00	Average reward: 133.04
Episode 930	Last reward: 122.00	Average reward: 129.83
Episode 940	Last reward: 125.00	Average reward: 127.59
Episode 950	Last reward: 128.00	Average reward: 127.51
Episode 960	Last reward: 133.00	Average reward: 129.60
Episode 970	Last reward: 141.00	Average reward: 132.99
Episode 980	Last reward: 137.00	Average reward: 134.18
Episode 990	Last reward: 131.00	Average reward: 134.05

Episode 1000	Last reward: 136.00	Average reward: 133.26
Episode 1010	Last reward: 154.00	Average reward: 137.79
Episode 1020	Last reward: 149.00	Average reward: 142.09
Episode 1030	Last reward: 157.00	Average reward: 146.40
Episode 1040	Last reward: 146.00	Average reward: 150.64
Episode 1050	Last reward: 159.00	Average reward: 153.75
Episode 1060	Last reward: 157.00	Average reward: 156.99
Episode 1070	Last reward: 157.00	Average reward: 156.94
Episode 1080	Last reward: 169.00	Average reward: 159.15
Episode 1090	Last reward: 227.00	Average reward: 168.14
Episode 1100	Last reward: 500.00	Average reward: 255.90
Episode 1110	Last reward: 500.00	Average reward: 353.85
Episode 1120	Last reward: 500.00	Average reward: 412.49
Episode 1130	Last reward: 500.00	Average reward: 447.61
Episode 1140	Last reward: 500.00	Average reward: 468.63

Solved! Running reward is now 475.7266036258587 and the last episode runs to 500 time steps!

Observations :

- From the above results we can observe that it takes longer for the unshared method (1140 episodes) to converge as compared to the shared method (290) under similar circumstances (having same number of nodes and hidden layers)
- This result could be attributed to the simplicity of the environment (cartpole), allowing the shared network to learn the common features efficiently as compared to separate networks
- Moreover the suitability of a type of network can be ascertained from the complexity of environment and agent models that we use