Tutorial: Actor Critic Implementation

```
#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/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 {}\tLast reward: {:.2f}\tAverage 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
```

```
/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)
```

```
train()
```

```
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: 110.00   Average reward: 105.82
Episode 80 Last reward: 500.00   Average reward: 165.57
Episode 90 Last reward: 358.00   Average reward: 255.62
Episode 100     Last reward: 227.00   Average reward: 269.72
Episode 110     Last reward: 500.00   Average reward: 274.02
Episode 120     Last reward: 223.00   Average reward: 312.00
Episode 130     Last reward: 131.00   Average reward: 268.42
Episode 140     Last reward: 140.00   Average reward: 208.19
Episode 150     Last reward: 209.00   Average reward: 193.07
Episode 160     Last reward: 399.00   Average reward: 215.42
Episode 170     Last reward: 405.00   Average reward: 321.84
Episode 180     Last reward: 55.00    Average reward: 262.47
Episode 190     Last reward: 17.00    Average reward: 164.84
Episode 200     Last reward: 500.00   Average reward: 186.86
Episode 210     Last reward: 500.00   Average reward: 255.60
Episode 220     Last reward: 30.00    Average reward: 213.30
Episode 230     Last reward: 191.00   Average reward: 184.22
Episode 240     Last reward: 137.00   Average reward: 173.53
Episode 250     Last reward: 107.00   Average reward: 151.88
Episode 260     Last reward: 118.00   Average reward: 134.86
Episode 270     Last reward: 149.00   Average reward: 132.75
Episode 280     Last reward: 156.00   Average reward: 137.39
Episode 290     Last reward: 176.00   Average reward: 148.45
Episode 300     Last reward: 215.00   Average reward: 157.83
Episode 310     Last reward: 214.00   Average reward: 175.97
Episode 320     Last reward: 230.00   Average reward: 201.26
Episode 330     Last reward: 306.00   Average reward: 230.41
Episode 340     Last reward: 310.00   Average reward: 270.20
Episode 350     Last reward: 227.00   Average reward: 267.24
Episode 360     Last reward: 258.00   Average reward: 263.74
Episode 370     Last reward: 257.00   Average reward: 268.51
```

```
Episode 380     Last reward: 330.00   Average reward: 282.38
Episode 390     Last reward: 332.00   Average reward: 331.86
Episode 400     Last reward: 224.00   Average reward: 309.06
Episode 410     Last reward: 222.00   Average reward: 284.05
Episode 420     Last reward: 258.00   Average reward: 275.17
Episode 430     Last reward: 249.00   Average reward: 267.44
Episode 440     Last reward: 324.00   Average reward: 272.00
Episode 450     Last reward: 272.00   Average reward: 291.95
Episode 460     Last reward: 380.00   Average reward: 311.38
Episode 470     Last reward: 391.00   Average reward: 339.02
Episode 480     Last reward: 500.00   Average reward: 402.23
Episode 490     Last reward: 500.00   Average reward: 441.46
Episode 500     Last reward: 500.00   Average reward: 464.95
Episode 510     Last reward: 169.00   Average reward: 360.36
Episode 520     Last reward: 101.00   Average reward: 254.31
Episode 530     Last reward: 132.00   Average reward: 193.35
Episode 540     Last reward: 131.00   Average reward: 162.21
Episode 550     Last reward: 164.00   Average reward: 153.38
Episode 560     Last reward: 387.00   Average reward: 180.62
Episode 570     Last reward: 208.00   Average reward: 178.01
Episode 580     Last reward: 130.00   Average reward: 165.46
Episode 590     Last reward: 120.00   Average reward: 149.50
Episode 600     Last reward: 119.00   Average reward: 139.55
Episode 610     Last reward: 113.00   Average reward: 131.34
Episode 620     Last reward: 131.00   Average reward: 128.18
Episode 630     Last reward: 135.00   Average reward: 133.05
Episode 640     Last reward: 193.00   Average reward: 148.19
Episode 650     Last reward: 207.00   Average reward: 172.14
Episode 660     Last reward: 237.00   Average reward: 200.35
Episode 670     Last reward: 329.00   Average reward: 240.80
Episode 680     Last reward: 500.00   Average reward: 324.67
Episode 690     Last reward: 500.00   Average reward: 395.02
Episode 700     Last reward: 500.00   Average reward: 437.15
Episode 710     Last reward: 500.00   Average reward: 458.53
Episode 720     Last reward: 197.00   Average reward: 425.72
Episode 730     Last reward: 500.00   Average reward: 455.52
Episode 740     Last reward: 500.00   Average reward: 473.37
Solved! Running reward is now 475.9663174259033 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.
    # Actor Network
    self.fc1_a = nn.Linear(4, 128)
    self.fc2_a = nn.Linear(128, 2)

    # Critic Network
    self.fc1_v = nn.Linear(4, 128)
    self.fc2_v = 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.

      # Actor Network that outputs probabilities for each action
      a = F.relu(self.fc1_a(x))
      action_prob = F.softmax(self.fc2_a(a), dim=-1)

      # Critic Network that outputs the value of the state
      v = F.relu(self.fc1_v(x))
      state_values = self.fc2_v(v)

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

Episode 0  Last reward: 31.00    Average reward: 11.05
Episode 10 Last reward: 14.00    Average reward: 11.20
Episode 20 Last reward: 32.00    Average reward: 17.87
Episode 30 Last reward: 35.00    Average reward: 37.54
Episode 40 Last reward: 33.00    Average reward: 37.17
Episode 50 Last reward: 58.00    Average reward: 41.45
Episode 60 Last reward: 170.00   Average reward: 57.76
```

```
Episode 70 Last reward: 63.00    Average reward: 60.17
Episode 80 Last reward: 51.00    Average reward: 63.85
Episode 90 Last reward: 42.00    Average reward: 55.47
Episode 100     Last reward: 68.00    Average reward: 57.01
Episode 110     Last reward: 127.00   Average reward: 76.25
Episode 120     Last reward: 176.00   Average reward: 140.95
Episode 130     Last reward: 135.00   Average reward: 136.06
Episode 140     Last reward: 113.00   Average reward: 131.74
Episode 150     Last reward: 131.00   Average reward: 132.20
Episode 160     Last reward: 112.00   Average reward: 118.88
Episode 170     Last reward: 158.00   Average reward: 132.44
Episode 180     Last reward: 315.00   Average reward: 207.03
Episode 190     Last reward: 113.00   Average reward: 288.66
Episode 200     Last reward: 94.00    Average reward: 210.65
Episode 210     Last reward: 95.00    Average reward: 161.30
Episode 220     Last reward: 117.00   Average reward: 138.60
Episode 230     Last reward: 114.00   Average reward: 129.34
Episode 240     Last reward: 143.00   Average reward: 130.54
Episode 250     Last reward: 152.00   Average reward: 136.87
Episode 260     Last reward: 99.00    Average reward: 123.65
Episode 270     Last reward: 72.00    Average reward: 105.14
Episode 280     Last reward: 78.00    Average reward: 92.11
Episode 290     Last reward: 100.00   Average reward: 91.30
Episode 300     Last reward: 189.00   Average reward: 108.90
Episode 310     Last reward: 500.00   Average reward: 256.03
Episode 320     Last reward: 500.00   Average reward: 353.93
Episode 330     Last reward: 500.00   Average reward: 412.54
Episode 340     Last reward: 500.00   Average reward: 447.64
Episode 350     Last reward: 500.00   Average reward: 468.65
Solved! Running reward is now 475.7397831754416 and the last episode
runs to 500 time steps!
```

Inferences

- We observe that using two seperate networks allows us to converge faster (350 episodes) when comapared to using the same network for feature representation (740 episodes).
- Although the above result is true in most cases, but there are instances where the two seperate network model does not converge in 2000 episodes. We hypothesize this is because the weight initialization that happens randomly and this adds external stochasticities that cannot be predicted.
- Also, the learning rates of the critic and actor network are the same. But we know from theory we prefer to choose learning rate of the critic network to be higher than that of the actor network. This might also contribute to the stochastic results we observe.
- The seperate networks can be optimzed by reducing the number of hidden layer nodes used. By using the same number of hidden layer nodes as the case of same network for actor and critic, we are doubling the number of parameters to be estimated. I tried this experiment and the model converges in 240 episodes (experiment results not provided above).