Tutorial: Actor Critic Implementation

In [1]:

```
# Import required libraries

import argparse
import gymnasium as gym
import numpy as np
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
```

In [2]:

```
# Set constants for training # DO NOT CHANGE
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'])
```

In [3]:

```

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
```

In [4]:

```python
# DO NOT Modify Training Code

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()[0]
        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, truncated, _ = 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
```

In [14]:

```
%%time
# Trail 1
model = Policy()
optimizer = optim.Adam(model.parameters(), lr=1.5e-2)
eps = np.finfo(np.float32).eps.item()
train()
```

```
Episode 0       Last reward: 28.00      Average reward: 10.90
Episode 10      Last reward: 63.00      Average reward: 19.10
Episode 20      Last reward: 322.00     Average reward: 59.17
Episode 30      Last reward: 82.00      Average reward: 110.03
Episode 40      Last reward: 98.00      Average reward: 101.12
Episode 50      Last reward: 92.00      Average reward: 92.00
Episode 60      Last reward: 256.00     Average reward: 176.70
Episode 70      Last reward: 268.00     Average reward: 188.68
Solved! Running reward is now 670.8592141216325 and the last episode runs to 9319 time steps!
CPU times: user 20.2 s, sys: 35.1 ms, total: 20.3 s
Wall time: 20.9 s
```

In [15]:

```
%%time
# Trail 2
model = Policy()
optimizer = optim.Adam(model.parameters(), lr=1.5e-2)
eps = np.finfo(np.float32).eps.item()
train()
```

```
Episode 0       Last reward: 14.00      Average reward: 10.20
Episode 10      Last reward: 11.00      Average reward: 12.16
Episode 20      Last reward: 10.00      Average reward: 14.00
Episode 30      Last reward: 15.00      Average reward: 15.94
Episode 40      Last reward: 34.00      Average reward: 19.84
Episode 50      Last reward: 81.00      Average reward: 38.05
Episode 60      Last reward: 197.00     Average reward: 63.05
Episode 70      Last reward: 200.00     Average reward: 109.05
Episode 80      Last reward: 41.00      Average reward: 87.67
Episode 90      Last reward: 242.00     Average reward: 89.43
Episode 100     Last reward: 97.00      Average reward: 170.49
Episode 110     Last reward: 311.00     Average reward: 212.84
Episode 120     Last reward: 437.00     Average reward: 261.66
Episode 130     Last reward: 219.00     Average reward: 316.11
Episode 140     Last reward: 44.00      Average reward: 249.87
Episode 150     Last reward: 106.00     Average reward: 197.43
Episode 160     Last reward: 213.00     Average reward: 219.95
Solved! Running reward is now 538.783586169595 and the last episode runs to 6095 time steps!
CPU times: user 29 s, sys: 48.9 ms, total: 29 s
Wall time: 29.1 s
```

In [17]:

```
%%time
# Trail 3
model = Policy()
optimizer = optim.Adam(model.parameters(), lr=1.5e-2)
eps = np.finfo(np.float32).eps.item()
train()
```

```
Episode 0       Last reward: 17.00      Average reward: 10.35
Episode 10      Last reward: 38.00      Average reward: 16.29
Episode 20      Last reward: 104.00     Average reward: 34.82
Episode 30      Last reward: 28.00      Average reward: 61.61
Episode 40      Last reward: 97.00      Average reward: 71.99
Episode 50      Last reward: 87.00      Average reward: 77.92
Episode 60      Last reward: 23.00      Average reward: 76.47
Episode 70      Last reward: 28.00      Average reward: 66.68
Episode 80      Last reward: 158.00     Average reward: 77.86
Episode 90      Last reward: 59.00      Average reward: 90.38
Episode 100     Last reward: 99.00      Average reward: 95.70
Episode 110     Last reward: 529.00     Average reward: 135.16
Episode 120     Last reward: 312.00     Average reward: 178.71
Episode 130     Last reward: 619.00     Average reward: 251.69
Episode 140     Last reward: 926.00     Average reward: 300.43
Solved! Running reward is now 554.9558954068989 and the last episode runs to 5285 time steps!
CPU times: user 24.3 s, sys: 107 ms, total: 24.4 s
Wall time: 24.7 s
```

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

In [8]:

```

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

class UnsharedPolicy(nn.Module):
    def __init__(self):
        super(UnsharedPolicy, self).__init__()
        # TODO: Fill in.
        hidden_size = 128
        # Actor network
        self.actor_affine1 = nn.Linear(4, hidden_size)
        self.action_head = nn.Linear(hidden_size, 2)

        # Critic network
        self.critic_affine1 = nn.Linear(4, hidden_size)
        self.value_head = nn.Linear(hidden_size, 1)

        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
        # Actor forward pass
        actor_x = F.relu(self.actor_affine1(x))
        action_prob = F.softmax(self.action_head(actor_x), dim=-1)

        # Critic forward pass
        critic_x = F.relu(self.critic_affine1(x))
        state_values = self.value_head(critic_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


```

In [9]:

```
%%time
# Trail 1
model = UnsharedPolicy()
optimizer = optim.Adam(model.parameters(), lr=1.5e-2)
eps = np.finfo(np.float32).eps.item()
train()
```

```
Episode 0       Last reward: 12.00      Average reward: 10.10
Episode 10      Last reward: 72.00      Average reward: 21.01
Episode 20      Last reward: 78.00      Average reward: 37.09
Episode 30      Last reward: 32.00      Average reward: 48.21
Episode 40      Last reward: 131.00     Average reward: 91.93
Episode 50      Last reward: 80.00      Average reward: 87.19
Episode 60      Last reward: 105.00     Average reward: 83.80
Episode 70      Last reward: 438.00     Average reward: 134.90
Episode 80      Last reward: 661.00     Average reward: 359.26
Episode 90      Last reward: 215.00     Average reward: 326.31
Episode 100     Last reward: 195.00     Average reward: 289.66
Episode 110     Last reward: 189.00     Average reward: 264.00
Episode 120     Last reward: 108.00     Average reward: 233.10
Episode 130     Last reward: 39.00      Average reward: 182.92
Episode 140     Last reward: 103.00     Average reward: 139.45
Episode 150     Last reward: 104.00     Average reward: 124.55
Episode 160     Last reward: 88.00      Average reward: 111.15
Episode 170     Last reward: 123.00     Average reward: 110.90
Episode 180     Last reward: 154.00     Average reward: 123.97
Episode 190     Last reward: 212.00     Average reward: 149.61
Episode 200     Last reward: 377.00     Average reward: 208.81
Episode 210     Last reward: 406.00     Average reward: 294.45
Solved! Running reward is now 844.579265464063 and the last episode runs to 9999 time steps!
CPU times: user 52.2 s, sys: 148 ms, total: 52.3 s
Wall time: 52.5 s
```

In [11]:

```
%%time
# Trail 2
model = UnsharedPolicy()
optimizer = optim.Adam(model.parameters(), lr=1.5e-2)
eps = np.finfo(np.float32).eps.item()
train()
```

```
Episode 0       Last reward: 10.00      Average reward: 10.00
Episode 10      Last reward: 9.00       Average reward: 10.57
Episode 20      Last reward: 12.00      Average reward: 11.45
Episode 30      Last reward: 10.00      Average reward: 11.09
Episode 40      Last reward: 9.00       Average reward: 11.12
Episode 50      Last reward: 11.00      Average reward: 12.45
Episode 60      Last reward: 46.00      Average reward: 17.77
Episode 70      Last reward: 34.00      Average reward: 27.80
Episode 80      Last reward: 77.00      Average reward: 47.31
Episode 90      Last reward: 54.00      Average reward: 49.86
Episode 100     Last reward: 75.00      Average reward: 59.78
Episode 110     Last reward: 101.00     Average reward: 69.92
Episode 120     Last reward: 67.00      Average reward: 73.71
Episode 130     Last reward: 74.00      Average reward: 80.61
Episode 140     Last reward: 40.00      Average reward: 74.51
Episode 150     Last reward: 44.00      Average reward: 67.89
Episode 160     Last reward: 46.00      Average reward: 61.33
Episode 170     Last reward: 106.00     Average reward: 68.41
Episode 180     Last reward: 181.00     Average reward: 92.58
Episode 190     Last reward: 120.00     Average reward: 98.85
Episode 200     Last reward: 133.00     Average reward: 105.84
Episode 210     Last reward: 197.00     Average reward: 135.52
Episode 220     Last reward: 131.00     Average reward: 127.97
Episode 230     Last reward: 152.00     Average reward: 130.90
Solved! Running reward is now 669.9642084534141 and the last episode runs to 9999 time steps!
CPU times: user 31 s, sys: 62.5 ms, total: 31.1 s
Wall time: 33.9 s
```

In [13]:

```
%%time
# Trail 3
model = UnsharedPolicy()
optimizer = optim.Adam(model.parameters(), lr=1.5e-2)
eps = np.finfo(np.float32).eps.item()
train()
```

```
Episode 0       Last reward: 42.00      Average reward: 11.60
Episode 10      Last reward: 10.00      Average reward: 11.06
Episode 20      Last reward: 10.00      Average reward: 10.51
Episode 30      Last reward: 10.00      Average reward: 10.33
Episode 40      Last reward: 9.00       Average reward: 9.96
Episode 50      Last reward: 9.00       Average reward: 9.69
Episode 60      Last reward: 10.00      Average reward: 9.59
Episode 70      Last reward: 11.00      Average reward: 9.73
Episode 80      Last reward: 11.00      Average reward: 9.56
Episode 90      Last reward: 11.00      Average reward: 9.60
Episode 100     Last reward: 12.00      Average reward: 10.66
Episode 110     Last reward: 17.00      Average reward: 13.32
Episode 120     Last reward: 15.00      Average reward: 16.55
Episode 130     Last reward: 49.00      Average reward: 29.21
Episode 140     Last reward: 10.00      Average reward: 24.73
Episode 150     Last reward: 19.00      Average reward: 24.10
Episode 160     Last reward: 104.00     Average reward: 40.09
Episode 170     Last reward: 48.00      Average reward: 43.23
Episode 180     Last reward: 50.00      Average reward: 50.56
Episode 190     Last reward: 48.00      Average reward: 50.56
Episode 200     Last reward: 189.00     Average reward: 65.73
Episode 210     Last reward: 106.00     Average reward: 144.54
Episode 220     Last reward: 93.00      Average reward: 124.08
Episode 230     Last reward: 78.00      Average reward: 106.24
Episode 240     Last reward: 75.00      Average reward: 91.86
Episode 250     Last reward: 78.00      Average reward: 85.44
Episode 260     Last reward: 106.00     Average reward: 82.58
Episode 270     Last reward: 119.00     Average reward: 94.06
Episode 280     Last reward: 115.00     Average reward: 103.90
Episode 290     Last reward: 147.00     Average reward: 110.33
Episode 300     Last reward: 118.00     Average reward: 114.31
Episode 310     Last reward: 113.00     Average reward: 120.09
Episode 320     Last reward: 93.00      Average reward: 115.12
Episode 330     Last reward: 191.00     Average reward: 127.87
Solved! Running reward is now 698.7663821435183 and the last episode runs to 9999 time steps!
CPU times: user 35.7 s, sys: 75.8 ms, total: 35.8 s
Wall time: 36 s
```

By Running the experiments for 3 consecutive trails, It is observed that the Actor Critic model with `shared features` perform better than the Actor critic model with `unshared features` .

The model with shared features learns a common feature representation by which the actor and critic update their parameters. Due to this there is stable updates and faster learning. The trails conducted also indicate the same, where model with shared features learns to solve the environment in lesser number of epochs compared to model with unshared features.

The parameters:

1. CPU time: Total time spent by CPU in executing the task.

- For Model with Shared features:
  - Avg CPU time over 3 trails: `24.57s`
- For Model with Unshared features:
  - Avg CPU time over 3 trails: `39.73s`

2. Wall time: The actual real world time spent in executing the task.

- For Model with Shared features:
  - Avg Wall time over 3 trails: `24.9s`
- For Model with Unshared features:
  - Avg Wall time over 3 trails: `40.8s`

These parameters supports the claim that the Actor critic model with shared features is better than the model with shared features.

In [ ]:

In [ ]: