4-1

Describe your Policy Gradient model

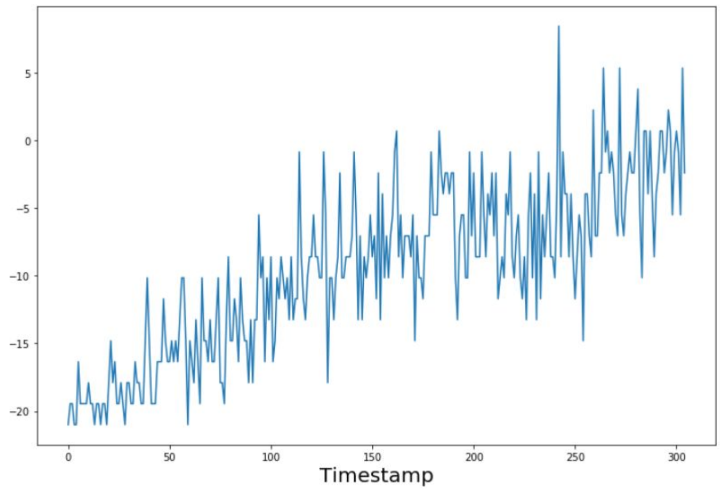
我使用的model相當單純，只是用一個CNN將環境的圖片作為輸入，並輸出各action的機率。該CNN的架構如下:

```python
class Model(torch.nn.Module):
    def __init__(self, observ_dim, action_num):
        super(Model, self).__init__()
        self.cnn1 = nn.Conv2d(1, 16, 8, stride=4)
        self.cnn2 = nn.Conv2d(16, 32, 4, stride=2)
        self.linear1 = nn.Linear(2048, 128)
        self.linear2 = nn.Linear(128, action_num)

    def forward(self, frames):
        frames = frames.unsqueeze(-3)
        x = F.relu(self.cnn1(frames))
        x = F.relu(self.cnn2(x))

        x = x.view(x.size(0), -1)

        x = F.relu(self.linear1(x))
        x = F.softmax(self.linear2(x), dim=1)

        return x
```

policy gradient部分，透過上面CNN可以得到6個action的probability，用該機率分布sample下一步，並得到reward。有使用到add baseline的tip，而這個baseline是直接取目前記錄中前兩千筆reward的平均。

```python
loss = -(score - mean) * log_probs
```

Plot the learning curve to show the performance of your Policy Gradient on Pong



Implement 1 improvement method on page 8

- Describe your tips for improvement
- Learning curve
- Compare to the vallina policy gradient

4-2

Describe your DQN model

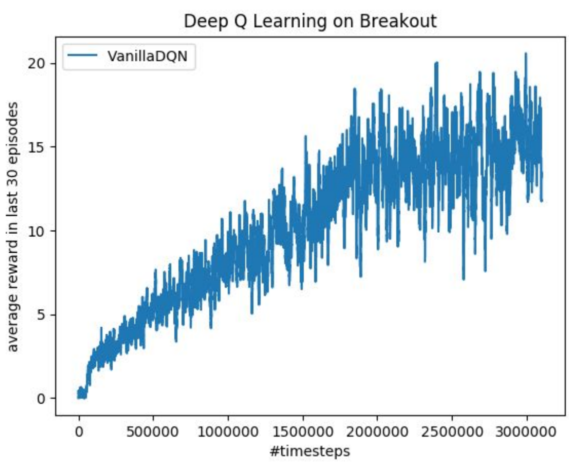
- model architecture

```python
class VanillaDQN(nn.Module):

    def __init__(self, input_shape, n_actions):
        super(VanillaDQN, self).__init__()
        self.conv1 = nn.Conv2d(input_shape[-1], 32, kernel_size=8, stride=4)
        #self.bn1 = nn.BatchNorm2d(16)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=4, stride=2)
        #self.bn2 = nn.BatchNorm2d(32)
        self.conv3 = nn.Conv2d(64, 64, kernel_size=3, stride=1)
        #self.bn3 = nn.BatchNorm2d(32)
        self.linear1 = nn.Linear(7*7*64, 512)
        self.linear2 = nn.Linear(512, n_actions)

    def forward(self, x):
        x = self.conv1(x)
        x = F.relu(x)
        x = self.conv2(x)
        x = F.relu(x)
        x = self.conv3(x)
        x = F.relu(x)
        #x = F.relu(self.bn3(self.conv3(x)))
        x = x.view(-1, 7*7*64)
        x = self.linear1(x)
        x = F.leaky_relu(x)
        x = self.linear2(x)
        return x
```

大致上為通過三層CNN，依序產生32個channels的feature map，再來是64個channels的feature map，中間的activation function為Relu，之後再通過兩層DNN，中間的activation function為LeakyRelu。

- optimizer
  - Adam，learning rate = 0.00015
- other hyperparameters
  1. γ = 0.99
  2. batch size = 32
  3. buffer size = 10000
  4. update current network step = 4
  5. update target network step = 1000
  6. epsilon-greedy採用exponential decline，epsilon start = 1，epsilon end = 0.025
  7. timesteps = 3100000 iterations

Plot the learning curve to show the performance of your Deep Q Learning on Breakout



這個圖是在training時reward為unclip的狀態下畫的。

Implement 1 improvement method on page 6

- Describe your tips for improvement

- Double DQN

```python
if self.args.DoubleDQN == True:
    next_q_values = self.eval_net(next_state)
    next_q_state_values = self.target_net(next_state).detach()
    next_q_value = next_q_state_values.gather(1, torch.max(next_q_values, 1)[1].unsqueeze(1)).squeeze(1)
else:
    next_q_values = self.target_net(next_state).detach()
    next_q_value = next_q_values.max(1)[0]
```

改成由current net predict最大的Q value的action當成target net input時的action。

- Dueling DQN

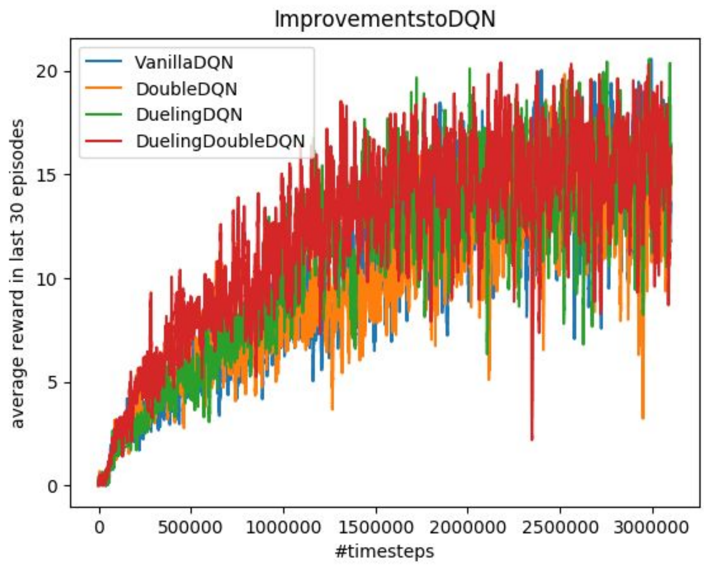
```python
class DuelingDQN(nn.Module):
    def __init__(self, input_shape, n_actions):
        super(DuelingDQN, self).__init__()

        self.features = nn.Sequential(
            nn.Conv2d(input_shape[-1], 32, kernel_size=8, stride=4),
            nn.ReLU(),
            nn.Conv2d(32, 64, kernel_size=4, stride=2),
            nn.ReLU(),
            nn.Conv2d(64, 64, kernel_size=3, stride=1),
            nn.ReLU()
        )

        self.advantage = nn.Sequential(
            nn.Linear(7*7*64, 512),
            nn.ReLU(),
            nn.Linear(512, n_actions)
        )

        self.value = nn.Sequential(
            nn.Linear(7*7*64, 512),
            nn.ReLU(),
            nn.Linear(512, 1)
        )

    def forward(self, x):
        x = self.features(x)
        x = x.view(x.size(0), -1)
        advantage = self.advantage(x)
        value     = self.value(x)
        return value + advantage - advantage.mean()
```

最後多計算個別action的advantage和value，當成最後的Q value。

- Dueling DQN + Double DQN

將上述兩個tips合在一起train。

- Learning curve



這個圖是在training時reward為unclip的狀態下畫的。

- Compare to origin Deep Q Learning
  - 從上面的圖可以看出，在training時，reward上升的速度，依序為Dueling Double DQN、Dueling DQN、Double DQN、Vanilla DQN。
  - testing時，各個model在100個episodes的average reward:
    - Vanilla DQN：49.86
    - Double DQN：65.38
    - Dueling DQN：45.4

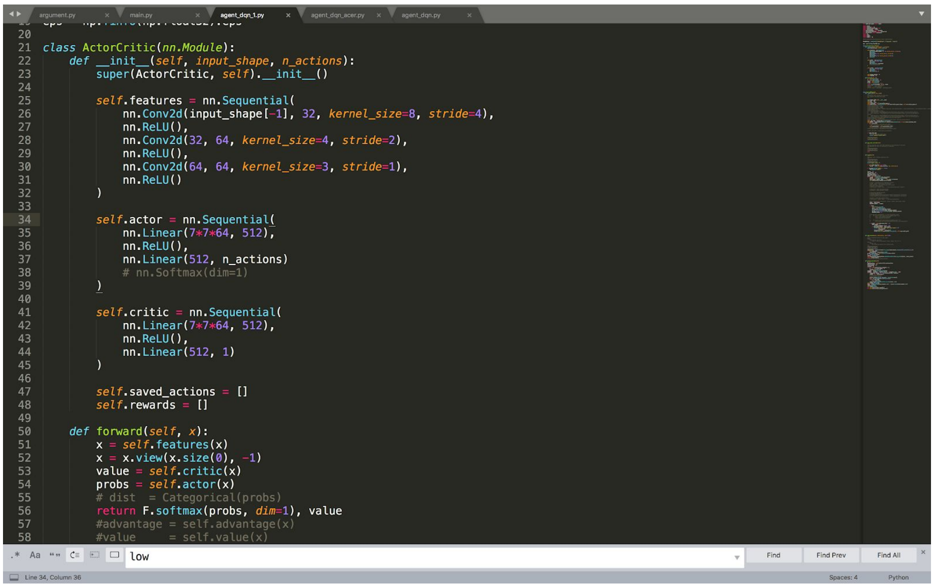
- Dueling Double DQN：50.68

其中Double DQN表現最好，Dueling DQN表現最差。

4-3

Describe your actor-critic model on Pong and Breakout

- Pong
- Breakout



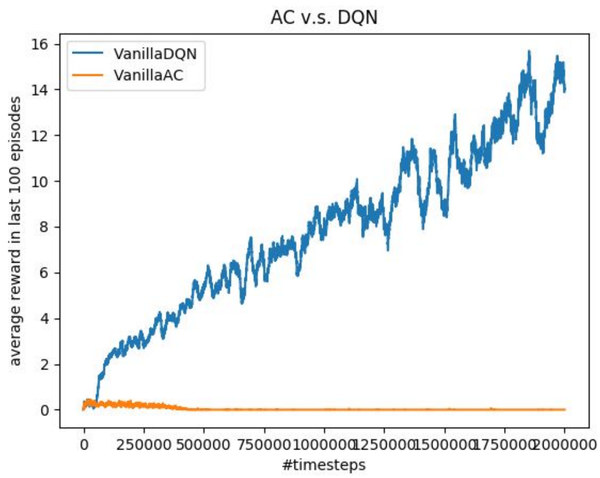
```python
class ActorCritic(nn.Module):
    def __init__(self, input_shape, n_actions):
        super(ActorCritic, self).__init__()

        self.features = nn.Sequential(
            nn.Conv2d(input_shape[-1], 32, kernel_size=8, stride=4),
            nn.ReLU(),
            nn.Conv2d(32, 64, kernel_size=4, stride=2),
            nn.ReLU(),
            nn.Conv2d(64, 64, kernel_size=3, stride=1),
            nn.ReLU()
        )

        self.actor = nn.Sequential(
            nn.Linear(7*7*64, 512),
            nn.ReLU(),
            nn.Linear(512, n_actions)
            # nn.Softmax(dim=1)
        )

        self.critic = nn.Sequential(
            nn.Linear(7*7*64, 512),
            nn.ReLU(),
            nn.Linear(512, 1)
        )

        self.saved_actions = []
        self.rewards = []

    def forward(self, x):
        x = self.features(x)
        x = x.view(x.size(0), -1)
        value = self.critic(x)
        probs = self.actor(x)
        # dist  = Categorical(probs)
        return F.softmax(probs, dim=1), value
        #advantage = self.advantage(x)
        #value     = self.value(x)
```

Actor和Critic共用前三層CNN，依序輸出32個channels的feature map和64個channels的feature map，其activation function為Relu。之後各別讓Actor和Critic通過各自的兩層DNN，其activation function為Relu。

- optimizer
    - Adam，learning rate = 0.00015
- other hyperparameters
    1. γ = 0.9
    2. timesteps = 2000000 iterations

Plot the learning curve and compare with 4-1 and 4-2 to show the performance of your actor-critic model on Pong & Breakout

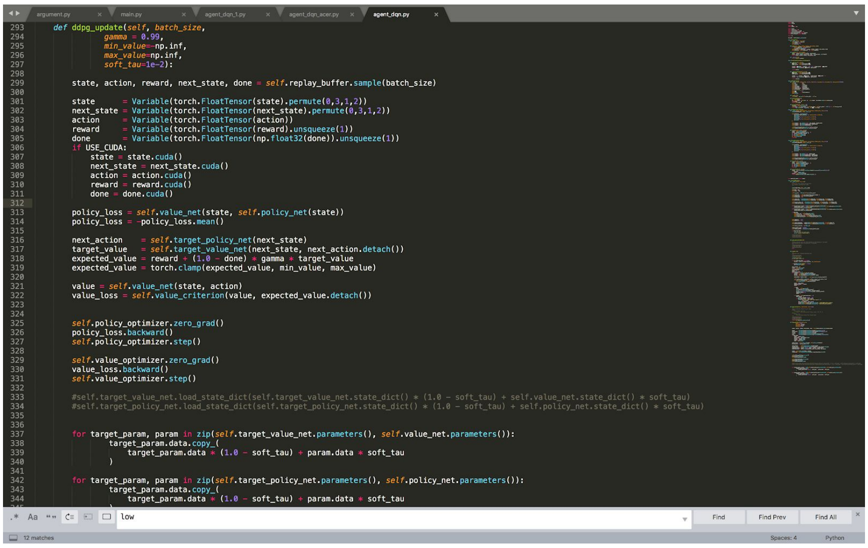
- Pong
- Breakout



這個圖是在training時reward為unclip的狀態下畫的。在Breakout這個task中，後期很明顯VanillaDQN表現的比VanillaAC好。

Reproduce 1 improvement method of actor-critic (Allow any resource)

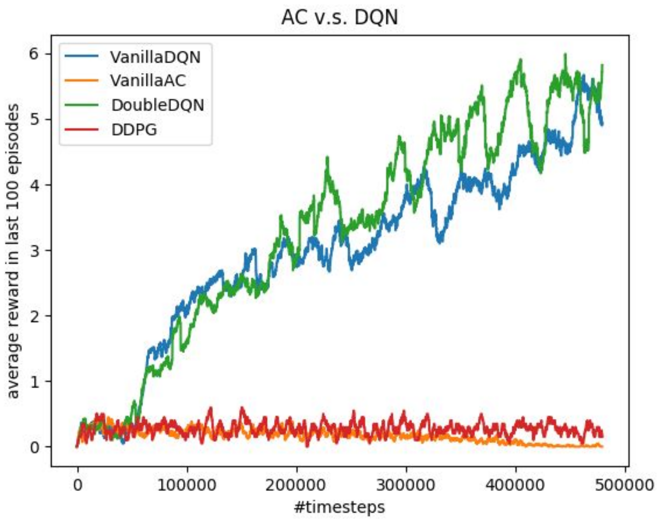
- Describe the method
  - DDPG



```
    def ddpg_update(self, batch_size,
                    gamma = 0.99,
                    min_value=-np.inf,
                    max_value=np.inf,
                    soft_tau=1e-2):

        state, action, reward, next_state, done = self.replay_buffer.sample(batch_size)

        state      = Variable(torch.FloatTensor(state).permute(0,3,1,2))
        next_state = Variable(torch.FloatTensor(next_state).permute(0,3,1,2))
        action     = Variable(torch.FloatTensor(action))
        reward     = Variable(torch.FloatTensor(reward).unsqueeze(1))
        done       = Variable(torch.FloatTensor(np.float32(done)).unsqueeze(1))
        if USE_CUDA:
            state = state.cuda()
            next_state = next_state.cuda()
            action = action.cuda()
            reward = reward.cuda()
            done = done.cuda()

        policy_loss = self.value_net(state, self.policy_net(state))
        policy_loss = -policy_loss.mean()

        next_action    = self.target_policy_net(next_state)
        target_value   = self.target_value_net(next_state, next_action.detach())
        expected_value = reward + (1.0 - done) * gamma * target_value
        expected_value = torch.clamp(expected_value, min_value, max_value)

        value = self.value_net(state, action)
        value_loss = self.value_criterion(value, expected_value.detach())


        self.policy_optimizer.zero_grad()
        policy_loss.backward()
        self.policy_optimizer.step()

        self.value_optimizer.zero_grad()
        value_loss.backward()
        self.value_optimizer.step()

        #self.target_value_net.load_state_dict(self.target_value_net.state_dict() * (1.0 - soft_tau) + self.value_net.state_dict() * soft_tau)
        #self.target_policy_net.load_state_dict(self.target_policy_net.state_dict() * (1.0 - soft_tau) + self.policy_net.state_dict() * soft_tau)


        for target_param, param in zip(self.target_value_net.parameters(), self.value_net.parameters()):
                target_param.data.copy_(
                    target_param.data * (1.0 - soft_tau) + param.data * soft_tau
                )

        for target_param, param in zip(self.target_policy_net.parameters(), self.policy_net.parameters()):
                target_param.data.copy_(
                    target_param.data * (1.0 - soft_tau) + param.data * soft_tau
```

個別計算policy和value的loss，在算expected target value時會clip到最小值和最大值之間。另外更新target policy net和target value net會和各自current net有一個比例的更新。

- Plot the learning curve and compare with 4-1 and 4-2, 4-3 to show the performance of your improvement
  - Pong
  - Breakout



這個圖是在training時reward為unclip的狀態下畫的。在Breakout這個task中，後期表現DoubleDQN最好，其次為VanillaDQN、DDPG，VanillaAC最差。