DS595/CS525 - Reinforcement Learning

Project 3: Atari Breakout using Deep Q-Learning Network

By: Abhishek Jain

Objective

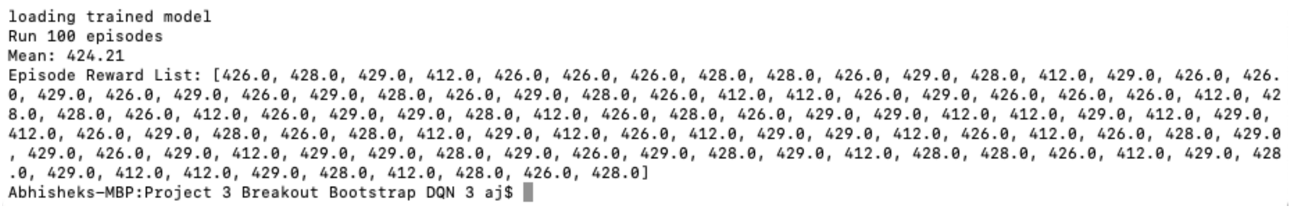
The aim of the project was to implement Deep Q-Learning Network(DQN). The network was to be implemented using OpenAI Gym on '*Atari Breakout'* environment.

Target

The bare minimum target for the project was to achieve an average score of 40 point in 100 episodes.

Results

The maximum average test score that was achieved over 100 episodes was **424.21** using Double-Dueling-DQN framework.

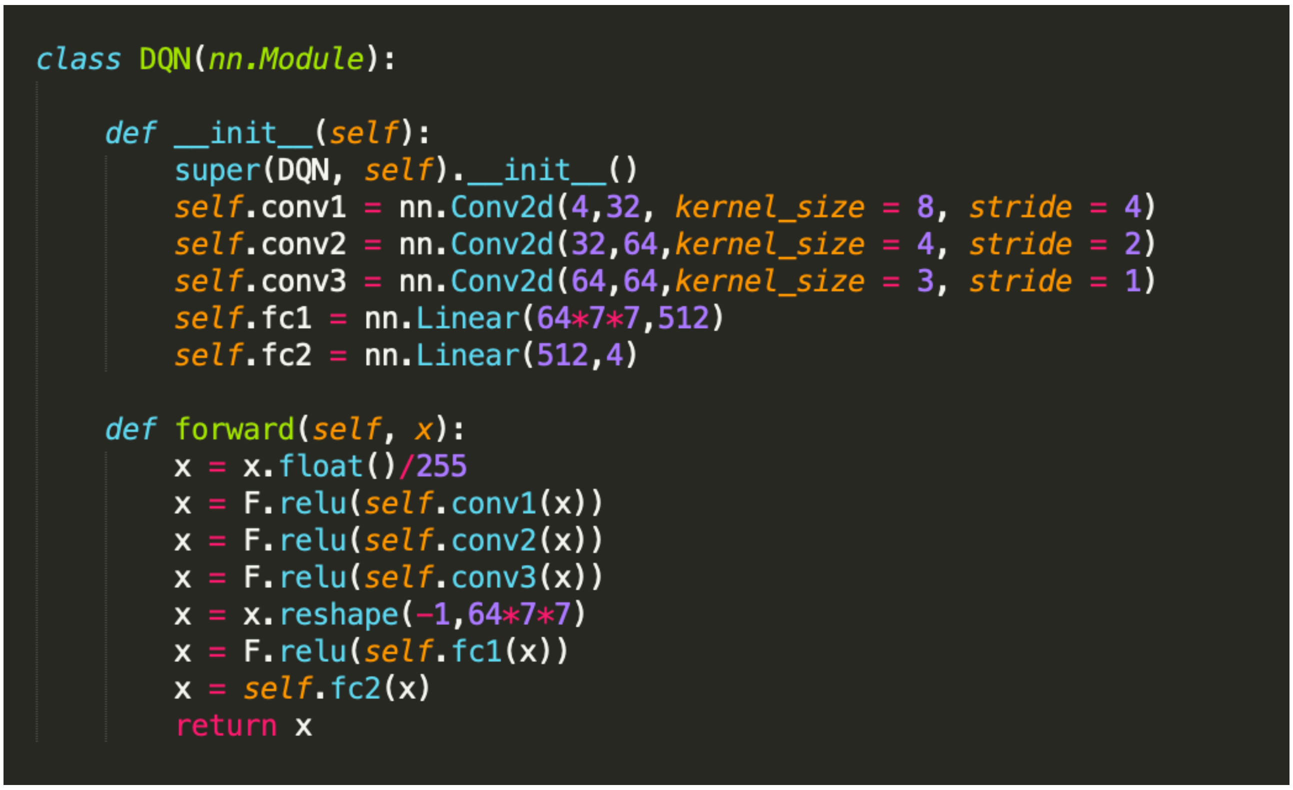


```
loading trained model
Run 100 episodes
Mean: 424.21
Episode Reward List: [426.0, 428.0, 429.0, 412.0, 426.0, 426.0, 426.0, 428.0, 428.0, 426.0, 429.0, 428.0, 412.0, 429.0, 426.0, 426.
0, 429.0, 426.0, 429.0, 426.0, 429.0, 428.0, 426.0, 429.0, 428.0, 426.0, 412.0, 412.0, 426.0, 429.0, 426.0, 426.0, 426.0, 412.0, 42
8.0, 428.0, 426.0, 412.0, 426.0, 429.0, 429.0, 428.0, 412.0, 426.0, 428.0, 426.0, 429.0, 429.0, 412.0, 412.0, 429.0, 412.0, 429.0,
412.0, 426.0, 429.0, 428.0, 426.0, 428.0, 412.0, 429.0, 412.0, 426.0, 412.0, 429.0, 429.0, 412.0, 426.0, 412.0, 426.0, 428.0, 429.0
, 429.0, 426.0, 429.0, 412.0, 429.0, 429.0, 428.0, 429.0, 426.0, 429.0, 428.0, 429.0, 412.0, 428.0, 428.0, 426.0, 412.0, 429.0, 428
.0, 429.0, 412.0, 412.0, 429.0, 428.0, 412.0, 428.0, 426.0, 428.0]
Abhisheks-MBP:Project 3 Breakout Bootstrap DQN 3 aj$ 
```

Approach and set of experiments performed:

Networks used:

1. **Normal DQN**:



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

    def __init__(self):
        super(DQN, self).__init__()
        self.conv1 = nn.Conv2d(4,32, kernel_size = 8, stride = 4)
        self.conv2 = nn.Conv2d(32,64,kernel_size = 4, stride = 2)
        self.conv3 = nn.Conv2d(64,64,kernel_size = 3, stride = 1)
        self.fc1 = nn.Linear(64*7*7,512)
        self.fc2 = nn.Linear(512,4)

    def forward(self, x):
        x = x.float()/255
        x = F.relu(self.conv1(x))
        x = F.relu(self.conv2(x))
        x = F.relu(self.conv3(x))
        x = x.reshape(-1,64*7*7)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return x
```

The training was started on a normal DQN network. The above mentioned network was used where there were :

- 3 Convilutional Layers: For frame processing
- 2 Linear Fully Connected Layers: For reaching the desired output
- Activation Function: ReLU

Results: The maximum average test score that was achieved over 100 episodes was 121 using Normal-DQN framework.

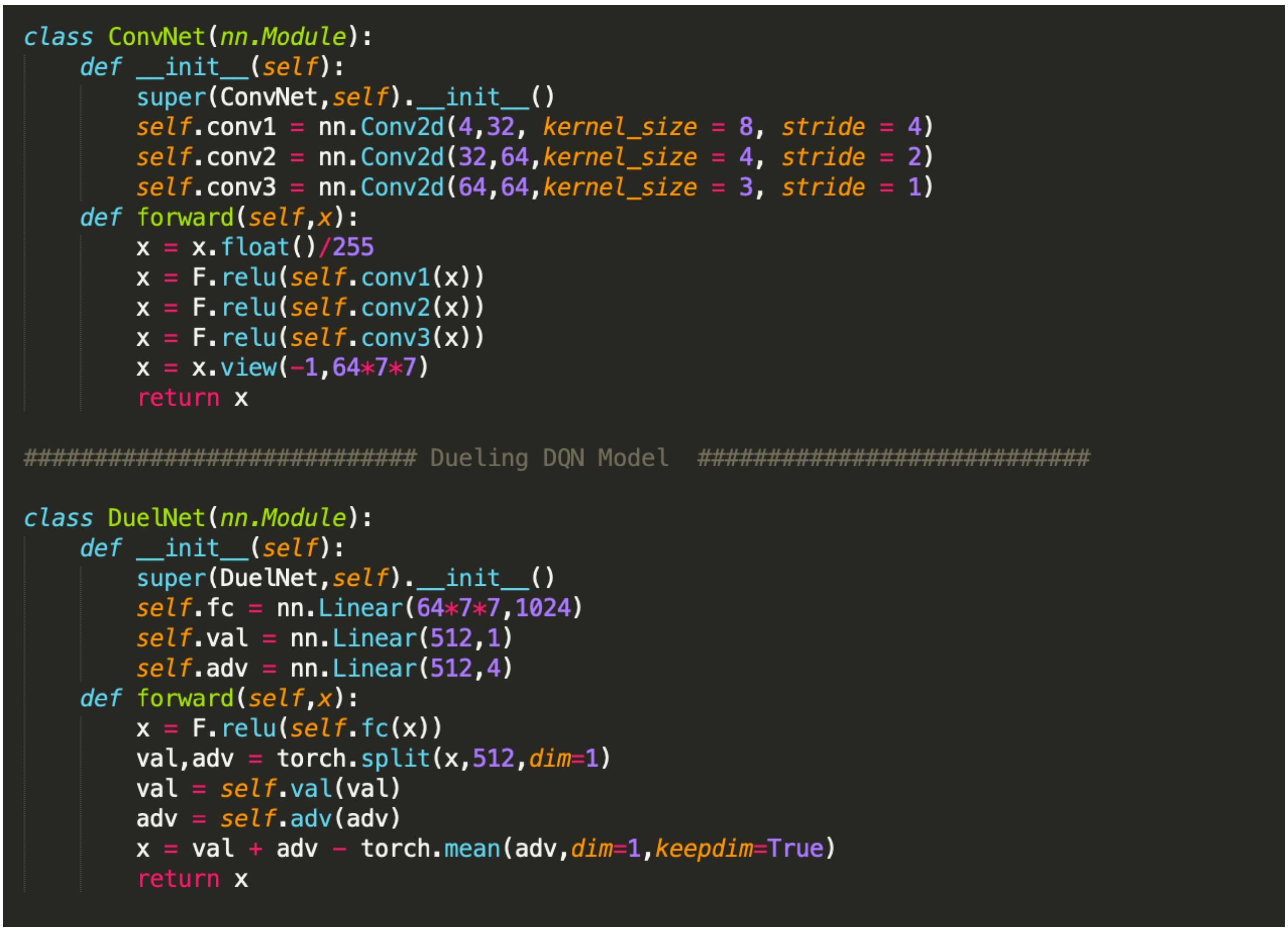
2. <u>Double-DQN:</u>

The same network architecture was used that was used for normal DQN. But the only difference was that the actions being selected for getting the Q-values of the next-immediate for bootstrapping was according to:

- Argmax of the target network : For Normal DQN
- Argmax of the current network: For Double DQN

Results: The maximum average test score that was achieved over 100 episodes was 178 using Double-DQN framework.

3. <u>Dueling DQN:</u>



```
class ConvNet(nn.Module):
    def __init__(self):
        super(ConvNet,self).__init__()
        self.conv1 = nn.Conv2d(4,32, kernel_size = 8, stride = 4)
        self.conv2 = nn.Conv2d(32,64,kernel_size = 4, stride = 2)
        self.conv3 = nn.Conv2d(64,64,kernel_size = 3, stride = 1)
    def forward(self,x):
        x = x.float()/255
        x = F.relu(self.conv1(x))
        x = F.relu(self.conv2(x))
        x = F.relu(self.conv3(x))
        x = x.view(-1,64*7*7)
        return x

############################ Dueling DQN Model  ############################

class DuelNet(nn.Module):
    def __init__(self):
        super(DuelNet,self).__init__()
        self.fc = nn.Linear(64*7*7,1024)
        self.val = nn.Linear(512,1)
        self.adv = nn.Linear(512,4)
    def forward(self,x):
        x = F.relu(self.fc(x))
        val,adv = torch.split(x,512,dim=1)
        val = self.val(val)
        adv = self.adv(adv)
        x = val + adv - torch.mean(adv,dim=1,keepdim=True)
        return x
```

Here, The network architecture was changed to have a split after the output from the first linear layer. This was done to generate two branches:

- A value network that would estimate the value of being in a state.
- An advantage network that would give the estimate of relative advantages of taking each action

The final output was adding the output of these two branches and subtracting the mean of advantage network to get the relative advantage of actions over the mean advantage.

The maximum average test score that was achieved over 100 episodes was **409.5** using Dueling-DQN framework.

4. **Double-Dueling DQN:**

The same network architecture was used that was used for Dueling DQN. But the only difference was that the actions being selected for getting the Q-values of the next-immediate for bootstrapping was according to:

- Argmax of the target network : For Normal Dueling DQN
- Argmax of the current network: For Double Dueling DQN

Results: The maximum average test score that was achieved over 100 episodes was **424.21** using Double-DQN framework.

Hyper-Parameter Tuning:

1) **Epsilon Decay:**
    1) Final Strategy used (best results):
        - Start epsilon from 1 and decay linearly to 0.1 over 1000000 (1 Million) steps. Then decay from 0.1 to 0.01 over 23.5 million steps.
    2) Other Experiments:
        1) Decay linearly from 1 to 0.01 over 1000000 (1 Million) steps
        2) Decay linearly from 1 to 0.01 over 500000 (0.5 Million) steps
        3) Decay linearly from 1 to 0.01 over 100000 (0.1 Million) steps
2) **Batch Size:**
    1) Final Strategy used (best results):
        - 32
    2) Other Experiments:
        1) 64 (Longer Time with not much change in results)
        2) 128 (Very long time with not much change in results)
3) **Loss Function:**
    1) Final Strategy used (best results):
        - Huber Loss: SmoothL1Loss
    2) Other Experiments:
        1) MSELoss (Was giving a bit more variance)
4) **Learning Rate:**
    1) Final Strategy used (best results):
        - 0.0000625
    2) Other Experiments:
        1) 0.001 (High Variation in later stages of training)
        2) 0.0001 (Relatively stable in training)
        3) 0.00001 (very slow convergence but very stable)
5) **Optimiser:**
    1) Final Strategy used (best results):
        - ADAM Optimiser

2) Other Experiments:

   1) SGD Optimiser (a bit slower)

6) **Minimum Buffer Size:** Minimum size of buffer for training to start

   1) Final Strategy used (best results):

      - 50,000

   2) Other Experiments:

      1) 10,000 (Provided similar bet slightly lesser results)

7) **Buffer Memory Size:**

   1) Final Strategy used (best results):

      - 500000 (0.5 Million) (The more the better, This was max according to available RAM - 27 GB)

   2) Other Experiments:

      1) 200000(0.2 Million)

      2) 100000(0.1 Million)

8) **Update Target network Frequency:**

   1) Final Strategy used (best results):

      - 10,000

   2) Other Experiments:

      1) 20,000 (Provided similar but took longer time to converge)

      2) 50,000 (Took a very long time to converge with ineffective learning)

9) **Learning Frequency:** Steps to skip after each gradient update

   1) Final Strategy used (best results):

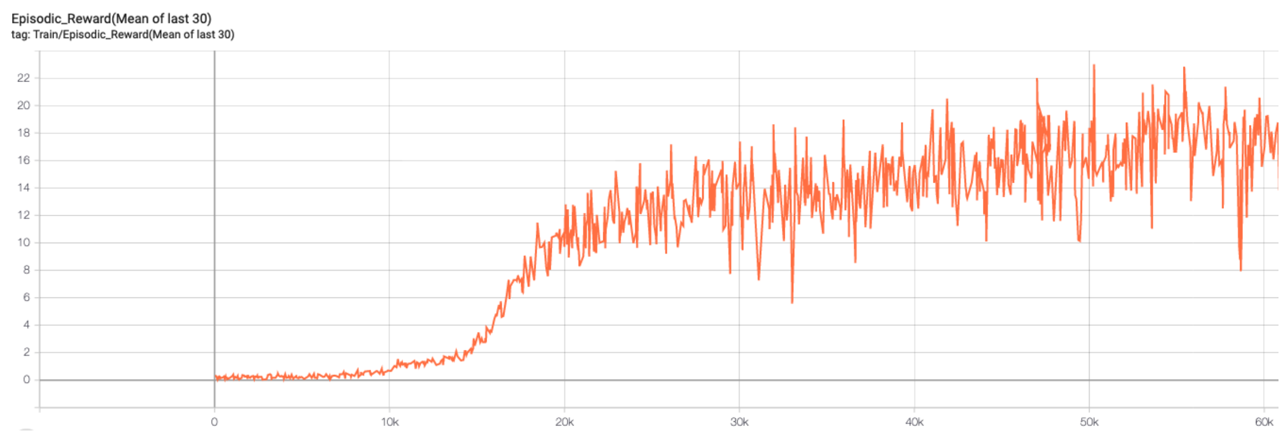
      - 4

   2) Other Experiments:

      1) 1 (Was very slow and was a bit unstable)

Visualisation

The X-axis represents the number of episodes, while the Y-axis reports the mean reward of last 30 episodes


Episodic_Reward(Mean of last 30)
tag: Train/Episodic_Reward(Mean of last 30)
10k
20k
30k
40k
50k
60k