Assignment 3: Deep Q-Learning

Architecture

Input

The network receives the current state of the environment as input. The state is usually shown as a vector in such a scenario (or, in environments such as Atari, as a sequence of stacked frames). In typical image-based settings, the input layer's size could be (84, 84, 4) for stacked frames (four frames layered together), however this would depend on the environment.

Convolutional Layers

```
class DQNetwork(nn.Module):
    def __init__(self, action_size):
        super(DQNetwork, self).__init__()
        self.conv1 = nn.Conv2d(in_channels=4, out_channels=32, kernel_size=8, stride=4)
        self.conv2 = nn.Conv2d(in_channels=32, out_channels=64, kernel_size=4, stride=2)
        self.conv3 = nn.Conv2d(in_channels=64, out_channels=64, kernel_size=3, stride=1)
        self.fc1 = nn.Linear(in_features=self._get_conv_output_size(), out_features=512)
        self.out = nn.Linear(in_features=512, out_features=action_size)

    def _get_conv_output_size(self):
        # Create a dummy input tensor with the expected shape
        input = torch.zeros(1, 4, 84, 80) # input is 4 frames, 84x80
        # Pass the dummy input through the convolutional layers
        output = self.conv1(input)
        output = self.conv2(output)
        output = self.conv3(output)
        # Flatten the output and return the size
        return output.view(1, -1).size(1)
```

Convolutional Layer 1:

- Filters: 32
- Kernel size: (8, 8)
- Stride: (4, 4)
- Activation: ReLU
- Purpose: Extracts low-level features like edges, textures, and basic shapes.

Convolutional Layer 2:

- Filters: 64
- Kernel size: (4, 4)
- Stride: (2, 2)
- Activation: ReLU
- Purpose: Extracts higher-level features from the input images.

Convolutional Layer 3:

- Filters: 64
- Kernel size: (3, 3)
- Stride: (1, 1)
- Activation: ReLU
- Purpose: Further reduces the spatial dimensions and extracts more complex features.

Fully Connected Layer 1:

- Number of neurons: 512
- Activation: ReLU
- Purpose: This layer takes the output from the convolutional layers (or directly from the input if not using convolutions) and processes it into a flattened vector. This layer allows the model to learn more complex, non-linear relationships in the data.

Fully Connected Layer 2 (optional):

- Number of neurons: 256
- Activation: ReLU
- Purpose: Another fully connected layer to help the model learn better abstract features.

Training:

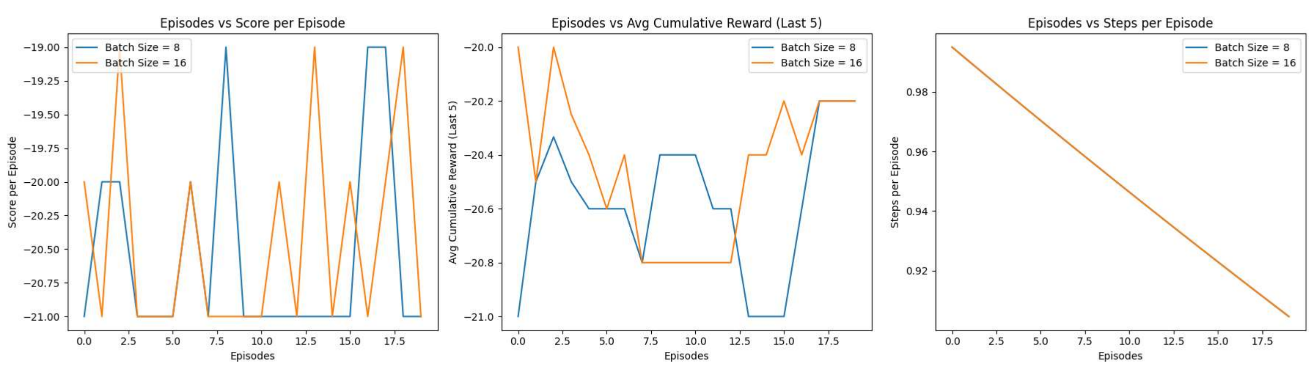
The DQN algorithm, which updates the Q-values according to the Bellman equation, is used to train the model using Q-learning:

$$Q(s,a)=r+\gamma a' \max Q(s',a')$$

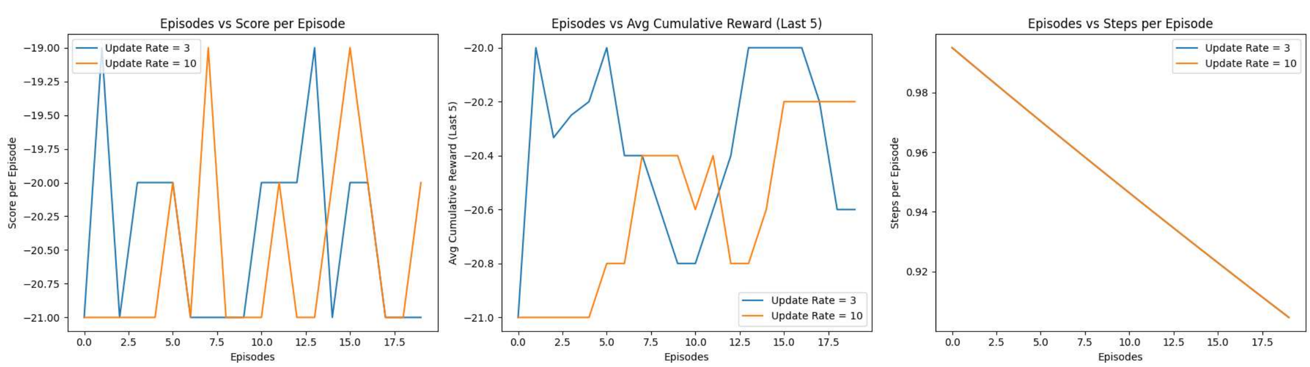
- **Maximum Q(s ′,a ′) :** The Mean Squared Error (MSE) between the target and forecasted Q-values is used to calculate the loss.
- This architecture is fairly flexible and can be changed based on the particular purpose or environment

Metrices, Observations and Comments

1. **Effect of 'Batch Size' on DQN Training**



2. **Effect of 'Update Rate' on DQN Training**



Metrices

1. **Score per Episode**
   - Significant variation in the scores between episodes suggests that learning is unstable.
   - Variations in batch size and update rate exhibit comparable trends, with no discernible improvement over the episodes.

2. **Average Cumulative Reward**
   - The fact that the average cumulative payment is still modest and fluctuates indicates that the agent hasn't picked up any useful policies.
   - Within the few episodes, the average cumulative prize is not significantly affected by batch sizes or update rates.
3. **Epsilon Decay**
   - Effective exploration is limited by the small number of episodes, although epsilon drops linearly as predicted.

Observation

1. **Batch Size**
   - Within 20 episodes, there was no discernible difference in performance between batches of 8 and 16.
   - More noisy updates can result from smaller batch sizes, however because of the short training period, this wasn't evident.
2. **Update Rate**
   - Performance was not significantly impacted by update rates of 3 or 10 throughout the brief training time.
   - Performance did not improve with more frequent updates (update rate of 3), which may have been caused by inadequate training time.

Comments

Training Duration:

These trials' main drawback is their brief (20-episode) training period. It usually takes a lot more time for DQNs to converge on challenging problems like Pong.

Exploration vs. Exploitation:

Due to the limited number of episodes, the epsilon decay may be excessively rapid, which would restrict exploration.

Hyperparameter Sensitivity:

The trials indicate that longer training times or other parameters may be more important, as neither batch size nor update rate significantly affected the limited episodes.

Best Combination of Batch Size and Update Rate

1. **Batch Size**: If training continues, a batch size of 16 may be ideal for more reliable updates.
2. **Update Rate:** A more stable update rate of 10 might enable the target network to update less frequently, which could result in more stable learning over extended training times.