COMS7071A Lab3: DQN Summary

Willem Van Der Merwe 2914429

University of the Witwatersrand, Johannesburg

18 October 2024

Chapter 1

Training Process

The training process follows the loop as shown in Figure 3:

1. **Replay Memory:** The agent stores transitions (state, action, reward, next state) in a replay buffer. This memory allows the agent to break correlations between sequential states by sampling random batches during the optimization step.

2. **Policy Network:** The policy network is the Q-network that chooses actions based on the current state of the environment. Actions can be selected using an epsilon-greedy strategy to balance exploration and exploitation.

3. **Target Network:** The target network is a delayed copy of the policy network, updated periodically. This helps stabilize training by providing fixed targets during the optimization process.

4. **Optimization:** At each step, a random batch of transitions is sampled from the replay memory. The policy network is optimized by minimizing the loss between predicted Q-values and target Q-values obtained from the target network.

5. **Target Network Update:** Occasionally, the target network is updated to the current policy network to maintain stable learning.

The general training loop follows these steps:

- Choose random or policy action based on the current state.
- Sample the environment based on the action.
- Record the transition (state, action, reward, next state) in replay memory.

- Optimize the policy network using a random batch from the replay memory.
- Occasionally update the target network with the policy network.

Chapter 2

Q-Network Architecture

The Q-Network is a neural network designed to approximate the Q-value function, $Q(s, a)$. In this implementation, I adopt an architecture similar to the one described in the original DQN paper, tailored for processing image inputs from the Atari Pong environment. Below is a detailed explanation of its architecture:

Input Layer: I apply several preprocessing steps and wrappers to the environment which processes input from the Atari environemnt, which is the current state s_t, which consists of a stack of recent frames. After preprocessing, the input state s_t is a tensor with shape $[C, H, W]$, where $C = 5$ (number of stacked frames), and $H = W = 84$.

- **NoopResetEnv:** Introduces a random number of no-op actions at the beginning to randomize initial conditions.
- **MaxAndSkipEnv:** Skips frames to reduce computation and takes the maximum of consecutive frames to handle flickering.
- **EpisodicLifeEnv:** Treats loss of lives as terminal states to provide more frequent learning signals.
- **FireResetEnv:** Ensures the game is properly initialized if a 'FIRE' action is required to start.
- **ClipRewardEnv:** Clamps rewards to the range $[-1, 1]$ to stabilize training.
- **WarpFrame:** Converts frames to grayscale and resizes them to 84×84 pixels to reduce computational complexity.

- **PyTorchFrame:** Transposes frame dimensions to match PyTorch's $C \times H \times W$ format.
- **FrameStack:** Stacks the last $k = 5$ frames along the channel dimension to capture motion.

Hidden Layers: which extract spatial and temporal features from the stacked frames, allowing the network to understand motion and object positions in the game.

- **Convolutional Layers:**
  - **First Convolutional Layer:**
    * **Input Channels:** 5
    * **Output Channels:** 32
    * **Kernel Size:** 8×8
    * **Stride:** 4
    * **Activation:** ReLU
  - **Second Convolutional Layer:**
    * **Input Channels:** 32
    * **Output Channels:** 64
    * **Kernel Size:** 4×4
    * **Stride:** 2
    * **Activation:** ReLU
  - **Third Convolutional Layer:**
    * **Input Channels:** 64
    * **Output Channels:** 64
    * **Kernel Size:** 3×3
    * **Stride:** 1
    * **Activation:** ReLU
- **Flattening:** The output of the final convolutional layer is flattened into a 1D vector to be fed into the fully connected layers.

- **Fully Connected Layers:** The fully connected layer serves to combine the features extracted by the convolutional layers and to learn higher-level representations. First Fully Connected Layer, has a input size which is calculated based on the output of the convolutional layers. It has a output size of 512 neurons which is then activated with ReLU.

Output Layer:

- **Action-Value Outputs:** Each output neuron corresponds to the Q-value $Q(s_t, a)$ for a specific action a. No activation function is applied to the output layer since Q-values are real numbers representing expected rewards. Ouput size is 6 discrete actions.

- **Action Selection:** The agent selects the action a_t by choosing the action with the highest Q-value: $a_t = \arg\max_a Q(s_t, a)$, unless exploring.

Loss Function:

- The network is trained to minimize the Temporal Difference (TD) error using the Mean Squared Error (MSE) loss.

- For a minibatch of transitions sampled from the replay buffer, the loss function is computed as:

$$L(\theta) = \frac{1}{N} \sum_{i=1}^{N} \left(r_i + \gamma \max_{a'} Q_{\text{target}}(s_{i+1}, a'; \theta^-) - Q(s_i, a_i; \theta) \right)^2$$

  where:

  - N is the batch size.
  - r_i is the reward received after taking action a_i in state s_i.
  - γ is the discount factor.
  - $Q_{\text{target}}(s_{i+1}, a'; \theta^-)$ is the Q-value from the target network for the next state s_{i+1}.
  - $Q(s_i, a_i; \theta)$ is the predicted Q-value from the policy network for the current state-action pair.

  - θ are the weights of the policy network, and θ^- are the weights of the target network.

- The network weights θ are updated by minimizing this loss using the Adam optimizer with the specified learning rate.

Optimization: Gradients are computed via backpropagation, and the optimizer adjusts the weights accordingly.

Target Network: A separate target network Q_{target} is maintained to provide stable targets during training. The weights θ^- of the target network are periodically updated from the policy network θ every specified number of steps (e.g., every 1,000 steps). This decoupling helps mitigate oscillations and divergence during training.

Epsilon-Greedy Exploration: The agent employs an epsilon-greedy strategy for action selection, with probability ϵ, a random action is selected (exploration), or with probability $1-\epsilon$, the action with the highest Q-value is selected (exploitation).

Epsilon Decay: Epsilon ϵ is decayed linearly from an initial value $\epsilon_{\text{start}} = 1.0$ to a minimum value $\epsilon_{\text{end}} = 0.01$ over a fraction of the total training steps. This strategy allows the agent to explore sufficiently while gradually focusing on exploiting learned behaviors. The decay is computed as:

$$\epsilon_t = \epsilon_{\text{start}} + \left(\frac{t}{\epsilon_{\text{decay\_steps}}}\right)(\epsilon_{\text{end}} - \epsilon_{\text{start}})$$

where t is the current timestep and $\epsilon_{\text{decay\_steps}}$ is the total number of steps over which epsilon is decayed.

Chapter 3

Hyperparameters

The performance of the DQN agent heavily depends on the choice of hyperparameters. Below is a detailed list of the hyperparameters used in this implementation, along with their descriptions:

- **Seed:** 42, Used to initialize random number generators for reproducibility.
- **Replay Buffer Size:** 5,000 transitions, The maximum number of past transitions stored in the replay buffer. A smaller buffer size speeds up training but may limit the diversity of experiences.
- **Batch Size:** 256 transitions, the number of transitions sampled from the replay buffer for each training update. A larger batch size provides more stable gradient estimates.
- **Learning Rate:** 0.0001, The step size used by the Adam optimizer to update the network weights. A lower learning rate helps in stable convergence.
- **Discount Factor (γ):** 0.99, Balances immediate and future rewards in the Q-value updates. A value close to 1.0 considers future rewards more heavily.
- **Number of Steps:** 1,000,000, the total number of training steps.
- **Learning Starts:** 10,000 steps, the number of initial steps where the agent only collects experiences without updating the network. Allows the replay buffer to fill up before training begins.
- **Learning Frequency:** Every 5 steps, specifies that the network is updated every 5 steps. Frequent updates can accelerate learning.

- **Target Network Update Frequency:** 1,000 steps, the frequency (in steps) at which the target network θ^- is updated with the policy network weights θ. Regular updates help stabilize training.
- **Exploration Rate (ϵ) Start Value:** 1.0, the initial probability of selecting a random action.
- **Exploration Rate (ϵ) End Value:** 0.01, the minimum probability of selecting a random action after decay.
- **Exploration Fraction:** 0.1, the fraction of total training steps over which ϵ is decayed from the start value to the end value. For example, if the total steps are 1,000,000, then ϵ is decayed over the first 100,000 steps.
- **Print Frequency:** Every 10 episodes, how often training statistics are printed to monitor progress.
- **Optimizer:** Adam, the optimization algorithm used for training the network. Chosen for its adaptive learning rate properties.
- **Frame Stack Size (k):** 5, the number of frames stacked together to form the state input. Provides temporal context to the agent.
- **Learning Rate Scheduler:** Not used, A constant learning rate is maintained throughout training.