Human-level control through deep reinforcement learning

Atul Balaji - EE16B002

March 5, 2019

1 Introduction

In general, Reinforcement Learning approaches work well only in low dimensional spaces and in domains where feature engineering can be done. It is difficult to scale them to solve real world problems, where there is high dimensional input. This paper introduces a powerful method called Deep Q-Network (DQN), which is capable of learning a successful policy from a high-dimensional sensory input. This has turned out to be the first successful combination of deep learning (deep CNN) and reinforcement learning (Q-learning). This network has been more successful than previous methods, aimed at playing Atari Games. It uses only the pixel data (frames of 210 x 160 video of the game), game score, number of actions and life count (if present in game) as input.

2 Approach

2.1 Preprocessing

Each frame is of shape 210x160. For each pixel, a max is taken over the pixel value currently and that of the previous frame. The luminance (Y) channel is extracted and then rescaled to 84x84. This preprocessing (denoted by function ϕ) is applied to the m (usually m = 4) most recent frames. These are then stacked and passed as input to the network.

2.2 Architecture

To approximate the Q-function, a **convolutional neural network** is used. It consists of 3 Convolutional layers (with a ReLU activation), 1 Fully connected hidden layer and 1 Fully connected output layer. It generates a single output for each valid action.

2.3 Training settings

- All rewards are clipped to -1 or 1, to limit the scale of the error derivatives and makes it possible to use the same learning rate across all games.
- RMSProp optimizer is used, with the batch size being 32.
- Frame-skipping is used, where we select actions every on k^{th} frame and this action is repeated on the skipped frames, so as to reduce computations.

2.4 Algorithm

- At each time step t, the agent selects action a_t and this is sent to the emulator. The internal state and the game score in the emulator are changed. Then, the image x_t and the reward (change in game score) r_t are sent to the agent. Since at any given time, the agent can only observe the current screen, this is a Partially Observable task.
- In this problem, the sequence of actions and observations is the state: $s_t = (x_1, a_1, ..., x_{t-1}, a_{t-1}, x_t)$. The goal of the agent is to select actions so as to maximize the total discounted return $R_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_t$.
- We need to find the optimal action-value function Q*(s,a) which achieves maximum expected return. If we know the optimal value Q*(s',a') i.e next time step, we can find Q*(s,a) using the Bellman Equation:

$$Q^*(s,a) = \mathbf{E}\left[r + \gamma \underset{a'}{max} Q^*(s',a') | s, a\right] \tag{1}$$

- In the DQN approach, Q* is approximated by Q(s,a;θ), where θ are the weights of the neural network. Training is done so as to reduce the mean squared error in the Bellman equation.

- The loss function at iteration i can be written as:

$$L_i(\theta_i) = \mathbf{E}_{s,a,r,s'}\left[(y - Q(s,a;\theta_i))^2\right] + \mathbf{E}_{s,a,r}[V_{s'}(y)] \tag{2}$$

where $y = r + \gamma \underset{a'}{max} Q(s', a'; \theta_i^-)$ and θ_i^- are the parameters from the previous iteration. The second term in the loss function is the variance term and is not dependent on parameter θ_i. After the end of an iteration, we update $\theta_i^- = \theta_{i-1}$.

- The above algorithm is a model-free and off-policy method. This is because it learns about the optimal (greedy) policy while following a behaviour policy which is ϵ-greedy.

This algorithm makes the following additions to the standard Q-learning approach:

- **Experience replay** - we store the experiences of the agent $e_t = (s_t, a_t, r_t, s_{t+1})$ in a data set $D_t = (e_1, e_2, ..., e_t)$. While applying Q-learning updates in the algorithm, we draw a random sample from U(D). Random sampling is done to avoid the correlation between consecutive samples which is observed in the case of standard Q-learning. It also avoids oscillations and divergence of parameters.

- **Separate network for targets** - Every C iterations, we set the target network $\hat{Q} \leftarrow Q$. This $\hat{Q}$ is used to generate the targets y_j for the next C updates to Q. This periodic update increases the stability of the algorithm.

3 Results

- The network is evaluated by playing each game for up to 30 times for up to 5 min, with different initial random conditions and an ϵ-greedy policy with $\epsilon = 0.05$. The agent chooses actions at a rate of 10 Hz, which is the fastest rate for human players.

- The algorithm was tested on 49 Atari Games. On 43 games, the DQN outperforms all previous RL methods and on 29 games, it achieved more than 75% of the human score.

- Interestingly, in the game "Breakout", the agent learns the long-term optimal strategy, which is to dig a tunnel through the bricks so that a large number of them will be destroyed, even though this strategy is not obvious at first.