CS786 TERM PAPER
Comparing 4 Deep Q-Learning

Nishant Patel - 210673

Abstract

This paper presents a comparative study of four influential Deep Q-Learning models: Deep Q-Networks (DQN), Deep Recurrent Q-Learning (DRQN), Double DQN, and Dueling DQN. Each model represents a significant advancement in reinforcement learning, addressing unique challenges such as stability, partial observability, and overestimation bias. The study delves into their respective architectures, objectives, and mathematical formulations, highlighting innovations like target networks, LSTM layers, and decoupled Q-value evaluation.

1 Deep Q-Networks (DQN) - Playing Atari with Deep Reinforcement Learning (2013)

Objective

This paper introduced the basic DQN model, which was a deep learning-based reinforcement. learning algorithm that can play Atari games directly from pixels. The model is a convolutional neural network, trained using a variant of Q-learning, whose input is raw pixels and whose output is a value function estimating future re- wards.

Model Architecture

- **Convolutional Neural Network (CNN)**: The network architecture starts with series of convolutional layers which extract spatial and temporal features from input frames:
  - The first convolutional layer has 32 filters with kernel size of 8x8 and a stride of 4, followed by a ReLU activation function.
  - The second convolutional layer has 64 filters with kernel size of 4x4 and a stride of 2, also followed by ReLU activation.
  - The third convolutional layer has 64 filters with a kernel size of 3x3 and a stride of 1, again with ReLU activation.

- **Target Network**: To further stabilize the training, DQN introduces a separate target network. This target network is copy of the online Q-network but is updated less frequently. The target network helps reduce oscillations in Q-value updates by providing stable targets:
  - The target Q-values used in the Bellman update equation are computed using the target network instead of the online Q-network.
  - The target network's parameters are periodically updated to match online Q-network's parameters (e.g., every 10,000 steps), allowing the online network to learn towards a stable target.
- **Optimization and Loss Function**: The DQN model is trained to minimize the Mean Squared Error (MSE) between the Q-value predicted by the online network and the target Q-value obtained from the target network:
  - The loss function is defined as:

    $$L(\theta) = \mathbb{E}_{(s,a,r,s')\sim U(D)}\left[\left(r + \gamma \max_{a'} Q(s', a'; \theta^-) - Q(s, a; \theta)\right)^2\right]$$

    where θ are the parameters of the online network, and θ^- are the parameters of the target network.
  - The gradient of this loss function with respect to θ is computed using backpropagation, and the network is optimized with the RMSProp optimizer to adjust the weights.

Mathematical Formulation

The Q-learning update rule used in DQN is as follows:

$$Q(s, a) \leftarrow Q(s, a) + \alpha \left(r + \gamma \max_{a'} Q(s', a') - Q(s, a)\right) \tag{1}$$

where:

- α is the learning rate,
- r is the reward received after taking action a in state s,
- γ is the discount factor, controlling the importance of future rewards,
- s' is the next state.

The loss function for optimizing Q-network is defined as:

$$L(\theta) = \mathbb{E}_{(s,a,r,s')\sim U(D)}\left[\left(r + \gamma \max_{a'} Q(s', a'; \theta^-) - Q(s, a; \theta)\right)^2\right] \tag{2}$$

2 Deep Recurrent Q-Learning for Partially Observable MDPs (2015)

Objective

It addresses a partially observable Markov decision processes (POMDPs) where the agent doesn't have full visibility of the state.

Model Architecture

- **Convolutional Layers for Feature Extraction**: The first stage of the network consists of convolutional layers, which extract spatial features from each frame:
  - These convolutional layers are similar to those in DQN, with first layer having 32 filters with a kernel size of 8x8 and a stride of 4, followed by a ReLU activation.
  - The second convolutional layer has 64 filters with a kernel size of 4x4 and a stride of 2, followed by ReLU activation.
  - The third convolutional layer has 64 filters with a kernel size of 3x3 and a stride of 1 followed by ReLU activation.
- **Recurrent Layer (LSTM)**: To capture temporal dependencies, DRQN adds a Long Short-Term Memory (LSTM) layer after the convolutional layers The hidden state h_t in the LSTM cell stores the "memory" of previous observations and actions, helping agent track the history of states that would no longer be visible.
- **Fully Connected Layers and Action Selection**: After the LSTM layer, the output is passed through fully connected layers to compute Q-values for each possible action:
  - The output of the LSTM layer is flattened and connected to a fully connected layer with 512 units and ReLU activation.
  - The final output layer has one node per possible action, each representing the Q-value of the corresponding action given the current state and history.

Mathematical Formulation

The Q-value function in DRQN is defined with a recurrent dependency on hidden states:

$$Q(s_t, a_t; \theta, h_t) = Q(h_t, a_t; \theta) \tag{3}$$

The hidden state h_t at time t is computed recursively:

$$h_t = f(h_{t-1}, x_t; \theta_h) \tag{4}$$

Performance

DRQN performs better than DQN in POMDP tasks by maintaining a sense of memory, outperforming DQN in tasks with incomplete state information.

3 Double DQN - Deep Reinforcement Learning with Double Q-learning (2015)

Objective

This paper tries to solve overestimation bias in Q-learning, which can result in suboptimal policies.

Model Architecture

- **Convolutional and Fully Connected Layers**:
  - DDQN uses convolutional layers to process visual inputs and extract spatial features. These layers are identical to DQN's, with the three convolutional layers followed by fully connected layers:
    - The first convolutional layer has 32 filters with a kernel size of 8x8 and stride of 4.
    - The second convolutional layer has 64 filters with a kernel size of 4x4 and a stride of 2.
    - The third convolutional layer has 64 filters with a kernel size of 3x3 and a stride of 1.
- **Double Q-Learning Mechanism**:
  - DDQN addresses this by decoupling action selection and evaluation, using two networks:
    - The *online network* (main network) $Q(s, a; \theta)$ selects the action that maximizes the Q-value.
    - The *target network* $Q(s', a; \theta^-)$ evaluates the Q-value of the selected action.
  - The target for Q-value update in DDQN is defined as:

    $$y = r + \gamma Q(s', \arg\max_{a'} Q(s', a'; \theta); \theta^-)$$

    where:
    - r is the reward,
    - γ is the discount factor,
    - $\arg\max_{a'} Q(s', a'; \theta)$ selects the action using the online network,
    - $Q(s', a; \theta^-)$ evaluates the action using the target network.

- **Loss Function and Optimization**:
  - The loss function in DDQN is based on the Mean Squared Error (MSE) between the Q-value predicted by the online network and decoupled target Q-value from the target network:

$$L(\theta) = \mathbb{E}_{(s,a,r,s')\sim U(D)}\left[(y - Q(s,a;\theta))^2\right]$$

Mathematical Formulation

The target y in Double Q-Learning is defined as:

$$y = r + \gamma Q(s', \arg\max_{a'} Q(s', a'; \theta); \theta^-) \tag{5}$$

The loss function in DDQN is given by:

$$L(\theta) = \mathbb{E}_{(s,a,r,s')\sim U(D)}\left[(y - Q(s,a;\theta))^2\right] \tag{6}$$

Performance

Double DQN (DDQN) demonstrates more stable learning and improved performance compared to DQN, especially in environments where overestimation is problematic.

4 Dueling DQN - Dueling Network Architectures for Deep Reinforcement Learning (2016)

Objective

This paper introduces ADAM (Adaptive Moment Estimation) for stochastic objective functions. It combines the benefits of RMSProp and AdaGrad to improve optimization in large-scale deep learning problems.

Model Architecture

- **AdaGrad:** Provides adaptive learning rates for each parameter based on the accumulation of squared gradients, making it effective for the sparse gradients.
- **RMSProp:** Normalizes learning rates using an exponential moving average of squared gradients, allowing better handling of non-stationary objectives.
- Adam integrates these concepts with momentum to balance fast convergence and stability, ensuring efficient training on large-scale datasets with noisy gradients.

- **Moment Estimates:**
  - **First Moment (Mean):**

    $$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$

    This estimate averages the gradients which helps the optimizer navigate noisy gradients and avoid small oscillations in the loss landscape.
  - **Second Moment (Variance):**

    $$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

    This estimate normalizes the updates by accounting for the magnitude of past gradients, preventing overly large updates in dimensions with high curvature.
  - **Bias Correction:** Both moment estimates are biased toward zero, especially during the initial steps. Bias correction addresses this by scaling the moments:

    $$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

- **Parameter Update Rule:** Adam updates parameters using the corrected moments:

  $$\theta_{t+1} = \theta_t - \frac{\alpha}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t$$

- **Hyperparameters:** Adam's performance is largely robust to its hyperparameters, but typical values are:
  - $\beta_1 = 0.9$: Controls the decay rate of the first moment estimate (momentum-like behavior).
  - $\beta_2 = 0.999$: Controls the decay rate of the second moment estimate (scale adjustment).
  - $\epsilon = 10^{-8}$: A small constant for numerical stability.
  - α: Learning rate, which can be tuned based on the specific task.

- **Implementation Simplicity:** Adam is computationally efficient, requiring constant memory overhead, and is straightforward to implement, making it a popular choice in modern deep learning frameworks.

References

[1] Matthew Hausknecht and Peter Stone. Deep recurrent q-learning for partially observable mdps. *CoRR*, abs/1507.06527, 2015.

[2] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[3] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[4] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *CoRR*, abs/1511.05952, 2015.

[5] John Schulman, Sergey Levine, Pieter Abbeel, Michael I Jordan, and Philipp Moritz. Trust region policy optimization. *arXiv preprint arXiv:1502.05477*, 2015.

[6] Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. *CoRR*, abs/1509.06461, 2015.

[7] Ziyu Wang, Nando de Freitas, and Marc Lanctot. Dueling network architectures for deep reinforcement learning. *arXiv preprint arXiv:1511.06581*, 2015.