PLAYING ATARI WITH DEEP REINFORCEMENT LEARNING

Kyriaki D. Zafeiroudi

INTRODUCTION

Successful DL applications need large amounts of hand-labelled training data.

RL algorithms are assumed to be able to learn from a scalar reward signal that is frequently sparse, noisy and delayed.

DL algorithms assume the data samples to be independent, while in RL the sequences of states are usually highly correlated.

In RL the data distribution changes as the algorithm learns new behaviors; wouldn't work for DL methods that assume a fixed underlying distribution.

INTRODUCTION

In this paper, a convolutional neural network is shown to overcome the aforementioned challenges, learning successful control policies from raw video data in complex RL environments.

The network is trained with a variant of the Q-learning algorithm, using stochastic gradient descent to update the weights.

Experience replay mechanism is used to randomly sample previous transitions, alleviating the problems of correlated data and non-stationary distributions.

The developed network outperformed all previous RL algorithms on six out of the seven Atari 2600 games that were used in the experiments, plus a human expert in three of them.

BACKGROUND

Environment: Atari emulator

Agent selects an action a_t from a set of legal game actions $A = \{1, \dots, K\}$

Agent does not observe internal state; observes an image $x_t \in \mathbb{R}^d$ from the emulator

Network receives reward r_t representing the change in the game score

The task is partially observed

Game strategies are learned depending on sequences of actions and observations $s_t = x_1, a_1, x_2, \dots, a_{t-1}, x_t$; standard RL methods for Markov decision processes apply by using the complete sequence s_t as the state representation at time t

The goal of the agent is to interact with the emulator by selecting actions in a way that maximizes future rewards; these are discounted by a factor of γ per time-step

The algorithm is model-free & off-policy.

BACKGROUND

The future discounted return at time t is $R_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}$

The optimal action-value function obeys an important identity known as the Bellman equation

$$Q^*(s,a) = \mathbb{E}_{s' \sim \mathcal{E}} \left[r + \gamma \max_{a'} Q^*(s',a') \middle| s, a \right]$$

The Q-network is trained by minimizing a sequence of loss functions

$$L_i(\theta_i) = \mathbb{E}_{s,a \sim \rho(\cdot)} \left[(y_i - Q(s,a;\theta_i))^2 \right]$$

The gradient occurs by differentiating the loss function with respect to the weights

$$\nabla_{\theta_i} L_i(\theta_i) = \mathbb{E}_{s,a \sim \rho(\cdot); s' \sim \mathcal{E}} \left[\left(r + \gamma \max_{a'} Q(s',a';\theta_{i-1}) - Q(s,a;\theta_i) \right) \nabla_{\theta_i} Q(s,a;\theta_i) \right].$$

DEEP REINFORCEMENT LEARNING

The goal here is to connect a reinforcement learning algorithm to a deep neural network which operates directly on RGB images and efficiently process training data by using stochastic gradient updates.

Contrary to known similar systems, for this goal experience replay is used; at each time-step the agent's experiences are stored $e_t = (s_t, a_t, r_t, s_{t+1})$ in a dataset $D = e_1, \ldots, e_N$, and they are then pooled over many episodes into a replay memory.

After experience replay is performed, the agent selects and executes an action according to an ε-greedy policy. The Q-function used here works on fixed length representation of histories produced by a function φ.

ALGORITHM: DEEP Q-LEARNING

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
  end for
end for

ADVANTAGES

Greater data efficiency

- Each step of experience is potentially used in many weight updates

Reduced variance of the updates

- Learning directly from consecutive samples is inefficient, due to the strong correlations between samples; by randomizing them these correlations break

The behavior distribution is averaged over many of its previous states, smoothing out learning and avoiding oscillations or divergence in the parameters because of using experience replay.

In practice, the algorithm only stores the last N experience tuples in the replay memory.

PREPROCESSING

The input is Atari frames: 210 x 160 pixel images with a 128 color palette. Reducing the dimensionality is essential.

RGB converted to gray-scale and then down-sampling was applied resulting in an 110 x 84 pixel image.

Final input was reduced to 84 x 84 due to the requirement for a square input.

Function φ applies this preprocessing to the last 4 frames of a history and stacks them to produce the input to the Q -function.

ARCHITECTURE

Input: 84 x 84 x 4 image produced by φ.

First hidden layer convolves 16 8 x 8 filters with stride 4 with the input image and applies a rectifier nonlinearity.

Second hidden layer convolves 32 4 x 4 filters with stride 2, again followed by a rectifier nonlinearity.

Final hidden layer is fully-connected and consists of 256 rectifier units.

Output layer is a fully-connected linear layer with a single output for each valid action; number of valid actions varied between 4 and 18 on the seven Atari games.

EXPERIMENTS

Same network was used across all seven games.

Positive rewards were fixed to be 1 and negative rewards to be -1.

RMSProp algorithm was used in minibatches of size 32.

Behavior policy was ε-greedy, ε annealed linearly from 1 to 0.1.

Training for a total of 10 million frames, using replay memory of 1 million.

Frame-skipping technique also applied.

EVALUATION METRICS

The total reward the agent collects in an episode or game averaged over a number of games, periodically computed during training.

The policy's estimated action-value function Q, which provides an estimate of how much discounted reward the agent can obtain by following its policy from any given state.

EVALUATION

	B. Rider	Breakout	Enduro	Pong	Q*bert	Seaquest	S. Invaders
Random	354	1.2	0	−20.4	157	110	179
Sarsa [3]	996	5.2	129	−19	614	665	271
Contingency [4]	1743	6	159	−17	960	723	268
DQN	**4092**	**168**	**470**	**20**	**1952**	**1705**	**581**
Human	7456	31	368	−3	18900	28010	3690
HNeat Best [8]	3616	52	106	19	1800	920	**1720**
HNeat Pixel [8]	1332	4	91	−16	1325	800	1145
DQN Best	**5184**	**225**	**661**	**21**	**4500**	**1740**	1075

EVALUATION

	B. Rider	Breakout	Enduro	Pong	Q*bert	Seaquest	S. Invaders
Random	354	1.2	0	−20.4	157	110	179
Sarsa [3]	996	5.2	129	−19	614	665	271
Contingency [4]	1743	6	159	−17	960	723	268
DQN	**4092**	**168**	**470**	**20**	**1952**	**1705**	**581**
Human	7456	31	368	−3	18900	28010	3690
HNeat Best [8]	3616	52	106	19	1800	920	**1720**
HNeat Pixel [8]	1332	4	91	−16	1325	800	1145
DQN Best	**5184**	**225**	**661**	**21**	**4500**	**1740**	1075