"Playing Atari with Deep Reinforcement Learning"
or "DQN"

Introduction

Background
↳ Bellman Eq^n
↳ optimal action value f^n
$Q^*(s,a) = \max_\pi E[R_t \mid s_t = s, a_t = a, \pi]$
↳ neural nets → Q-network
↓
minimising loss f^n
↳ model free & policy free

Introduced what the paper is going to present
↳ DQN outperformed all previous RL algos on 6/7 games and surpassed human expert players on 3 of them

Algorithm - DQN → using experience replay

① → Initialize replay memory D to capacity N
→ Initialize action-value f^n Q with random weights (θ)

② Episode Loop → training process iterates over a set no of episodes, M
↳ at beginning of each episode → environment is reset & initial sequence of observations(s_1) is obtained
↳ preprocessed by ϕ to create fixed length representation ϕ_1
→ Here preprocess to last 4 frames & stacks them

③ Time Step Loop → within each episode, agent interacts with the environment over time steps, t

Action Selection

agent

with $P(\varepsilon)$

↓

selects random action from set of legal game actions, A

Otherwise $(P(1-\varepsilon))$

↓

selects action "a" that maximises the predicted Q value for current pre-processed state $\phi(s_t)$

Q network weights θ : $\underbrace{a = \operatorname{argmax}_a Q^*(\phi(s_t), a; \theta)}_{\varepsilon\text{-greedy strategy}}$

Environment Interaction

↳ selected action (a_t) executed in emulator (Atari) emulator updates its ~~initial time~~ internal state, returns a reward (r_t) and next image observation (x_{t+1})

State Update & Preprocessing

↳ actions & observations updated to → $\underbrace{s_{t+1} = s_t, a_t, x_{t+1}}$

sequence preprocessed by ϕ to obtain next state ϕ_{t+1}

Store transition

↳ agent's experience, represented as transitions $(\phi_t, a_t, r_t, \phi_{t+1})$ stored in replay memory D. Old ~~memory~~ experiences replaced with new ones (replay buffer has limited memory)

Sample Mini Batch

↳ random sampling from replay memory D.

↳ breaks strong correlation among consecutive samples and reduces variance of updates

↳ more stable training

Calculate Target y_j → for each ~~target~~ transition in mini-batch, target value y_j is calculated.

if

ϕ_{t+1} is terminal → immediate reward r_j

ϕ_{t+1} is non terminal state → calculated upon Bellman Eqⁿ, incorporating max Q value for next state,

$$y_j = r_j + \underbrace{\gamma}_{\text{Discount factor}} \max_{a'} Q(\phi_{j+1}, a'; \theta)$$

Perform Gradient Descent

GD performed on loss f^n → squared difference b/w predicted Q value and calculated targeted value y_j summed over the minibatch, using SGD. The gradient update is proportional to → $(y_j - Q(\phi_j, a_j; \theta))^2$

(4) Repeat (2 to 11)

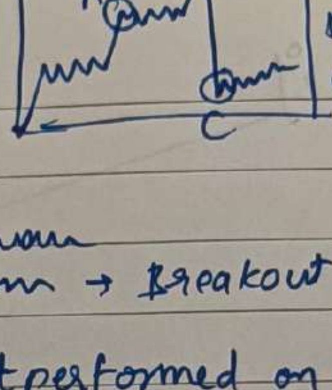
Experiments

↳ Beam Rider, Breakout, Enduro, Pong, Q*bert, Seaquest, Space Invaders

↳ same network architecture for all 7 games

fixed all positive rewards ⇒ 1 & -ve rewards as -1

5.1 → figure 3 →



A → aiming correctly

B → ammo shot

C → the aftermath, drop

Human

→ outperformed on → Breakout, Pong, Enduro

HNeat → outperformed on Space Invaders