Human level control through deep reinforcement learning

Presented by: Carsten Lüth
Supervisor: Professor U. Köthe
1. Why do we play video games?

- Humans play to:
  - learn
  - test their abilities
  - compete with each other

- Agents are trained on games:
  - because it is easier, faster and safer than in the real world
  - to benchmark their performance in different environments

ATARI2600 games

Taken from: https://gym.openai.com/envs/#atari
1. How do we play Video Games?

- **Humans:**
  1. See and hear the video game
  2. Interpret the input
  3. Decide to do something

- **Agents before DQN:**
  1. Get the pixels of the video input
  2. Interpret the input
  3. Decide to do something

- Get the pixels of the video input
  - Derive features from the pixels
    - Hand-crafted features
  - Linear value functions, policy representations etc.
1. MARI\0 - NEAT

Taken from: https://www.youtube.com/watch?v=qv6UVOQ0F44&t=224s
1. Hand-Crafted Features

- Why are they used?
  - Computer vision: raw pixels of videos or images are bad features
    - high dimensional, highly correlated
    - distill human knowledge into the agents

- Problems:
  - costly and often not generalizable from a game like pac-man to Super Mario
  - agents performance is limited by the features
Fundamental Concept:
- self learning feature maps
- CNN extracts features
- features used for a learned Q-function
- Fully connected network
- Agent trainable via backpropagation
2. A DQN - playing Seaquest

Taken from: https://www.youtube.com/watch?v=XjsY8-P4WHM
2. B Convolutional Neural Network

- multiple convolutional Layers, each of those Layers uses a filter or kernel to create a new feature-map
- convolutions use the same filters on different locations of the image
2.B Features of a CNN

Filters of a VGG- Network trained on IMAGENET visualized

Taken from: Visualizing and Understanding Convolutional Networks - Matthew D Zeiler, Rob Fergus
2.B Value Function

- Optimal Value function:

\[ Q^*(s, a) = \max_{\pi} \mathbb{E}[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots | s_t = s, a_t = a, \pi] \]

- maximum expected Reward given any policy, state and action

- Q-learning:

\[ Q^{i+1}(s_t, a_t) \leftarrow Q^i(s_t, a_t) + \alpha (r_t + \gamma \max_{a_{t+1}} Q^i(s_{t+1}, a_{t+1}) - Q^i(s_t, a_t)) \]

\[ Q^i(s, a) \rightarrow_{i \rightarrow \infty} Q^*(s, a) \]
2. B Designing the Input

- The state is a concatenation of consecutive downsampled and grayscaled frames

- Reward is the score difference of the ATARI games reached within the frames

\[ r_t = \text{Score}_{t+1} - \text{Score}_t \]

Taken from: https://www.freecodecamp.org/news/an-introduction-to-deep-q-learning-lets-play-doom-54d02d8017d8/
2.B State: Concept of Time

Multiple frames allow the agent to learn a concept of movement and time

$s_t$: Stack of consecutive frames

Taken from: https://towardsdatascience.com/self-learning-ai-agents-part-ii-deep-q-learning-b5ac60c3f47
2. B Experience Replay

❖ Intuition:
❖ Save past experiences so that the agent can learn from them

Taken from: https://www.freecodecamp.org/news/an-introduction-to-deep-q-learning-lets-play-doom-54d02d8017d8/

\[ e_t = (s_t, a_t, r_t, s_{t+1}) \quad D_t = \{e_1, e_2, e_3, \ldots, e_t\} \quad (s, a, r, s') \sim U(D) \quad \text{Q-learning updates} \]

❖ Benefits:
❖ Avoids forgetting previous episodes
❖ Reduce correlations between experiences
2.B Experience Replay

Forgetting Previous Episodes

Taken from: https://www.freecodecamp.org/news/an-introduction-to-deep-q-learning-lets-play-doom-54d02d8017d8/
2. B Training schedule

- Minimization Objective: \( \hat{\theta} = \operatorname{argmin}_\theta \mathcal{L}(\theta) \)

- Loss:
\[
\mathcal{L}(\theta_i) = \mathbb{E}_{(s_t, a_t, r_t, s_{t+1}) \sim U(D)}[(r_t + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}; \theta_i^-) - Q(s_t, a_t; \theta_i))^2]
\]

\( \theta_i \): Value function (iterative updates)  \( \theta_i^- \): Target-Value function (periodic updates)

\( D \): Experience Dataset

- Define 2 value functions with different updates:
  - value function is updated by gradient descent w.r.t. target-value function
  - target value function gets updated periodically every \( C \) iterations
\[
\theta_C^- = \theta_C
\]

- Benefit: training is more stable
2.B Exploration vs. Exploitation

\( \epsilon \)-Greedy

Describes the probability between exploration and exploitation by going off-policy during training of the agent.

\[
p(a | s) = \begin{cases} 
1 - \epsilon + \epsilon/n & : a = \text{argmax}_{a_0} Q(a_0 | s) \\
\epsilon/n & : \text{else}
\end{cases}
\]

\( n \) : \# actions
\( \epsilon \) gets annealed during the training from 1 to 0.1

2.C Algorithm

Algorithm 1: deep Q-learning with experience replay.

- Initialize replay memory $D$ to capacity $N$
- Initialize action-value function $Q$ with random weights $\theta$
- Initialize target action-value function $\hat{Q}$ with weights $\theta^- = \theta$

For episode $= 1, M$ do
  - Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
  - For $t = 1, T$ do
    - With probability $\varepsilon$ select a random action $a_t$
    - otherwise select $a_t = \arg \max_{a} Q(\phi(s_t), a; \theta)$
    - Exploration vs. Exploitation
    - 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 $D$
    - Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $D$
    - Set $y_j = \begin{cases} r_j & \text{if episode terminates at step } j+1 \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta^-) & \text{otherwise} \end{cases}$
    - Update Q-function
      - Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ with respect to the network parameters $\theta$
      - Every $C$ steps reset $\hat{Q} = Q$
  - End For
- End For

Experience Replay
update Q-function
3. Performance on ATARI 2600

- Agents trained on 49 different games

\[
\text{Performance} = \frac{R_A - R_R}{R_H - R_R} \cdot 100
\]

- $R_H$: human gaming tester score
- $R_R$: random play score
- $R_A$: agent score

- Agents have an action frequency of 10 Hz
3. DQN - playing Breakout

Taken from: https://www.youtube.com/watch?v=TmPftjtdgg
4. Discussion

✧ Pros:
  ✧ generalizes well to different environments
  ✧ learns important features itself
    ✧ minimizes human knowledge
  ✧ outperforms most other algorithms (before 2015) and has the best overall score

✧ Cons:
  ✧ decision horizon is rather short
  ✧ algorithm learns slower than a human
Sources

- Scientific:
  - Human level control through deep reinforcement learning - V. Mnih, K. Kavukcuoglu, D. Silver et al.
  - Playing Atari with Deep Reinforcement Learning - Volodymyr Mnih, Koray Kavukcuoglu, David Silver et al.
  - Visualizing and Understanding Convolutional Networks - Matthew D Zeiler, Rob Fergus

- Further Reading:
  - Reinforcement Learning, Fast and Slow - M. Botvinick, Sam Ritter, Jane X. Wang et al.