Deep Reinforcement Learning: Rainbow in Atari Simulator

Deep Learning Seminar WS 19/20

Aylin Haskioglu

Hochschule Offenburg
University of Applied Sciences

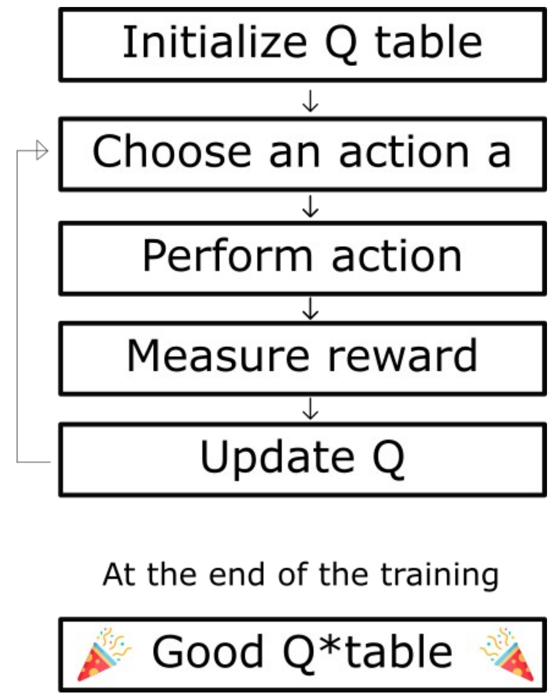
Agenda

- Rainbow Agent
  - DQN + Variants/Extensions
  - Results
- Experiments
  - Frameworks
  - Results

Q-learning

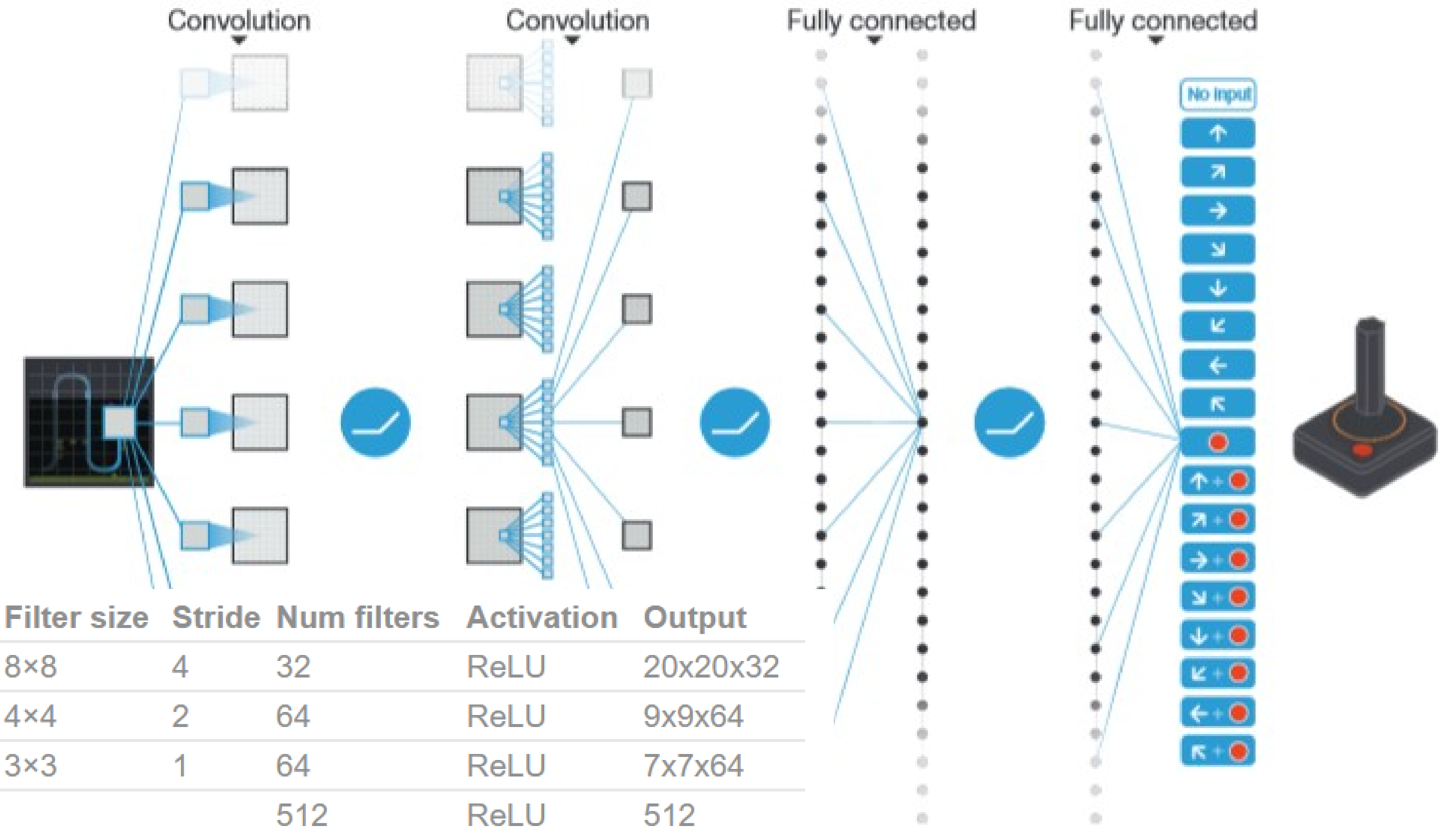
- Learning action-value function
- Determines how good a action at a particular state is
- Bellman equation:

$$Q(s,a) = r + \gamma max_{a'} Q(s', a')$$



https://medium.com/@SmartLabAI/reinforcement-learning-algorithms-an-intuitive-overview-904e2dff5bbc

DQN – Deep Q-learning Network



Layer	Input	Filter size	Stride	Num filters	Activation	Output
conv1	84x84x4	8×8	4	32	ReLU	20x20x32
conv2	20x20x32	4×4	2	64	ReLU	9x9x64
conv3	9x9x64	3×3	1	64	ReLU	7x7x64
fc4	7x7x64			512	ReLU	512
fc5	512			18	Linear	18

https://neuro.cs.ut.ee/demystifying-deep-reinforcement-learning/
https://web.stanford.edu/class/psych209/Readings/MnihEtAlHassibis15NatureControlDeepRL.pdf

DQN - Variants/Extensions

Double DQN (DDQN)

- Countermeasurment for overestimation bias
- Decouples action selection from its evaluation

Multi-Step Q-learning

- Calculate Q-values with N-step Return

DQN - Variants/Extensions

Prioritized Experience Replay

- Selectes samples with most probability to learn from
- Learning potential -> Q-Loss

Dueling Network Architecture

- One stream to calculate value of being at specific state
- One stream to calculate advantage of action over other actions at specific state
- Combines at the end to get a state action value

DQN - Variants/Extensions

Distributional Q-learning

- Normally using average estimated Q-value as target

  -> average Q-values not accurate
- Learn distribution of Q-Values instead of average
- KL divergence as loss

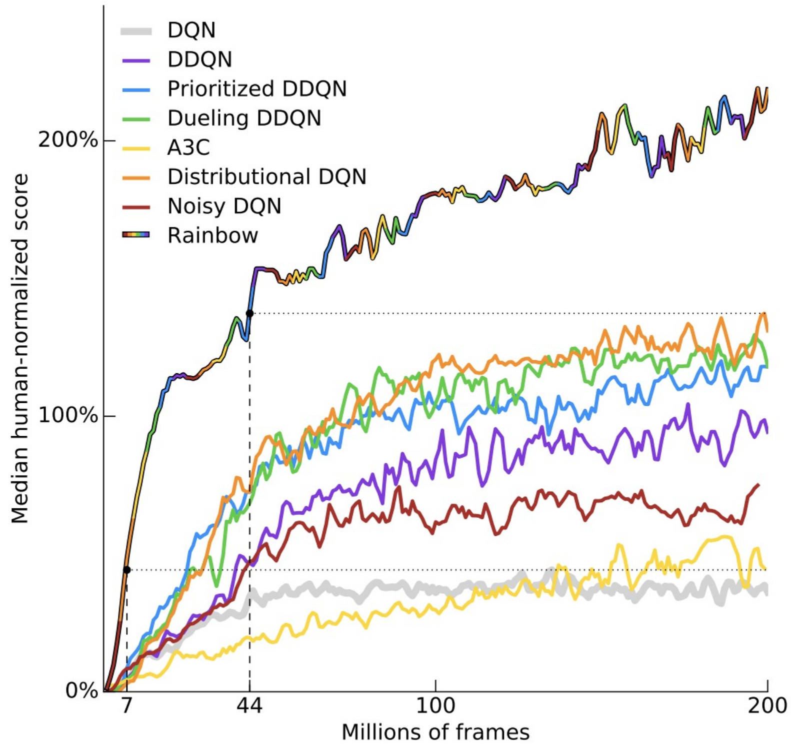
Noisy Nets

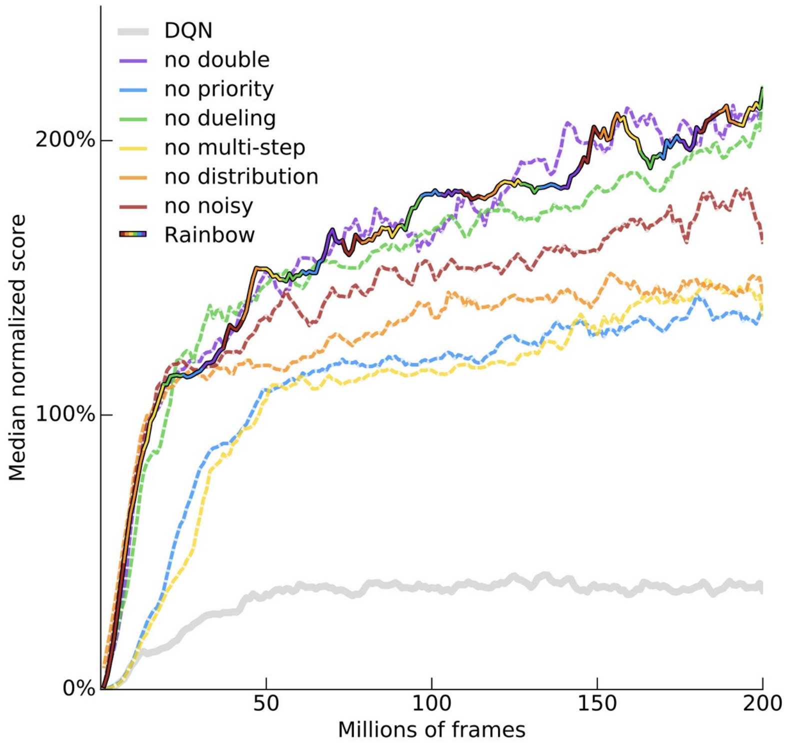
- Combining final output linear layer of Q-Network with noisy stream
- Network can learn to ignore noisy stream during training

Rainbow

- Replace 1-step distributional loss with multi-step variant
- Combine with DDQN
- Proportional prioritized replay → prioritizes transistions by the KL loss
- Dueling network adapted for use with return distributions
- Replace all linear layers with noisy equivalent

Rainbow


DQN
DDQN
Prioritized DDQN
Dueling DDQN
A3C
Distributional DQN
Noisy DQN
Rainbow
Median human-normalized score
200%
100%
0%
7
44
100
200
Millions of frames


DQN
no double
no priority
no dueling
no multi-step
no distribution
no noisy
Rainbow
Median normalized score
200%
100%
0%
50
100
150
200
Millions of frames

Experiments

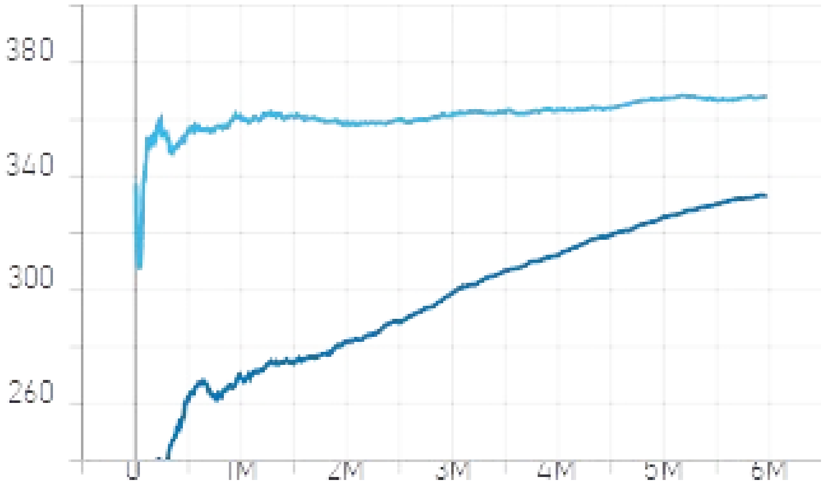
- Different frameworks, different implementations
- Colab
- Anyrl-py
  - Used by OpenAI Baseline Retro Contest
- Dopamine
  - Only using 3 variants not all

Parameter	Value
Min history to start learning	80K frames
Adam learning rate	0.0000625
Exploration ϵ	0.0
Noisy Nets σ_0	0.5
Target Network Period	32K frames
Adam ϵ	1.5×10^{-4}
Prioritization type	proportional
Prioritization exponent ω	0.5
Prioritization importance sampling β	$0.4 \rightarrow 1.0$
Multi-step returns n	3
Distributional atoms	51
Distributional min/max values	$[-10, 10]$

Table 1: Rainbow hyper-parameters

Anyrl – Space Invaders

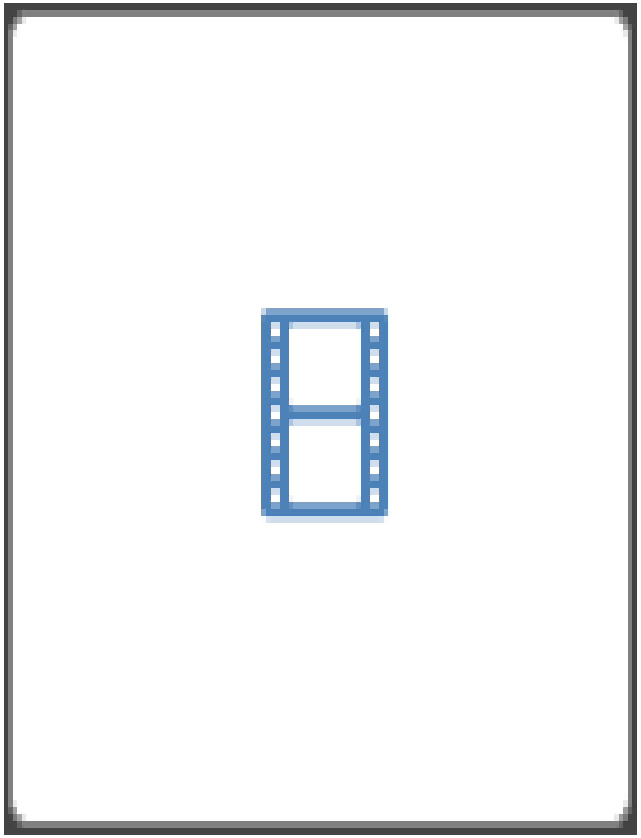
Rewards



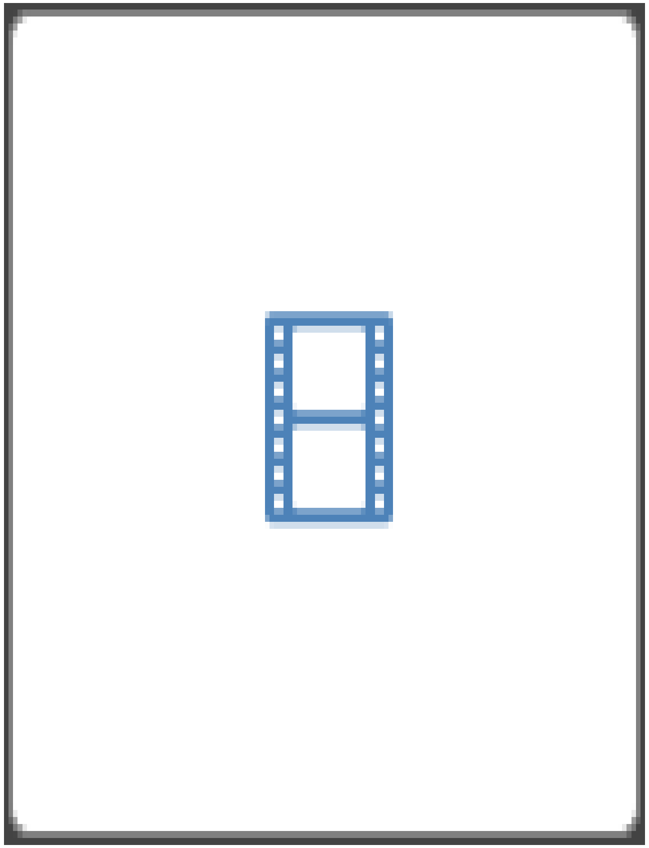
Losses

Anyrl – Space Invaders

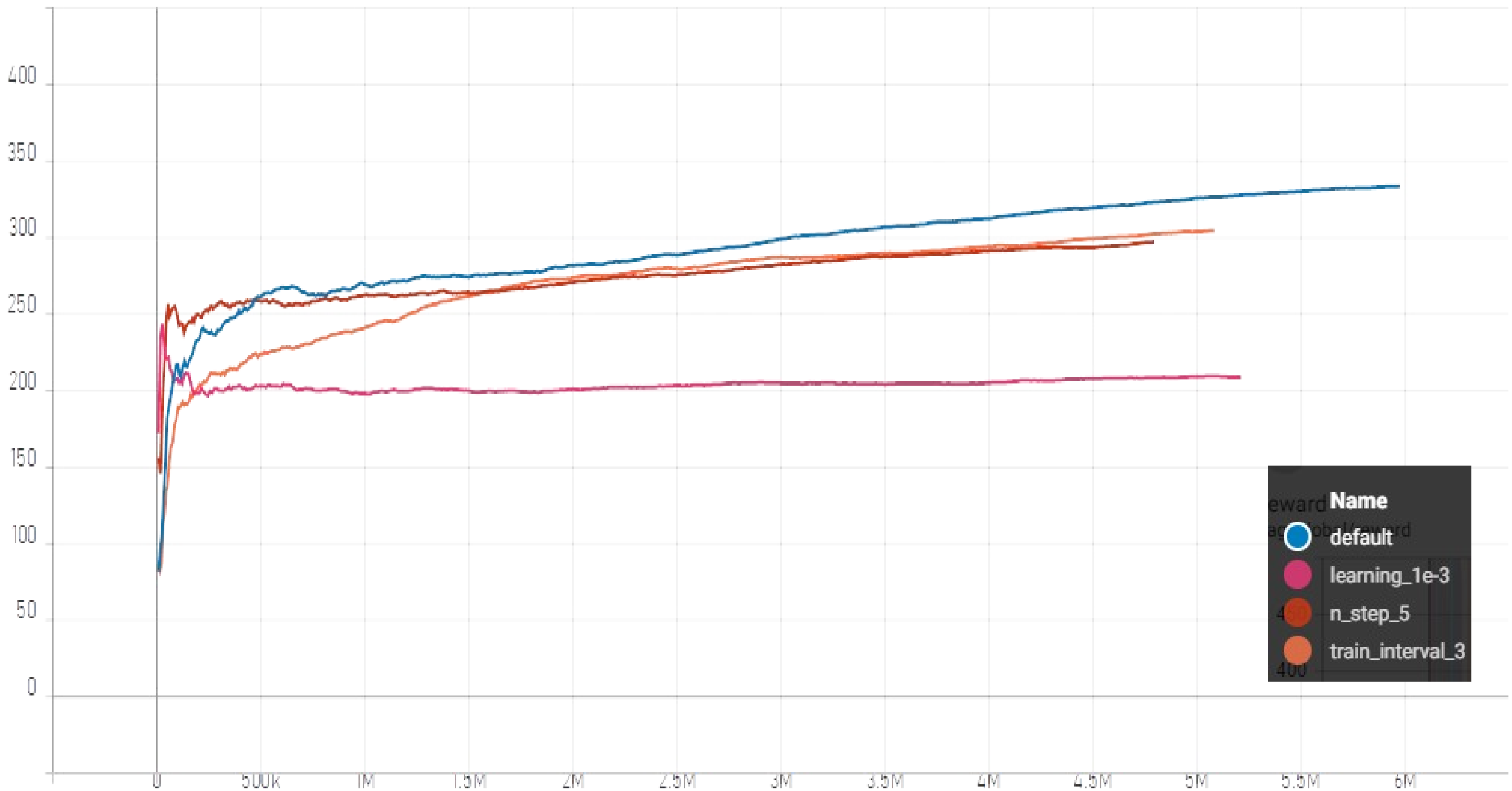
Episode 0



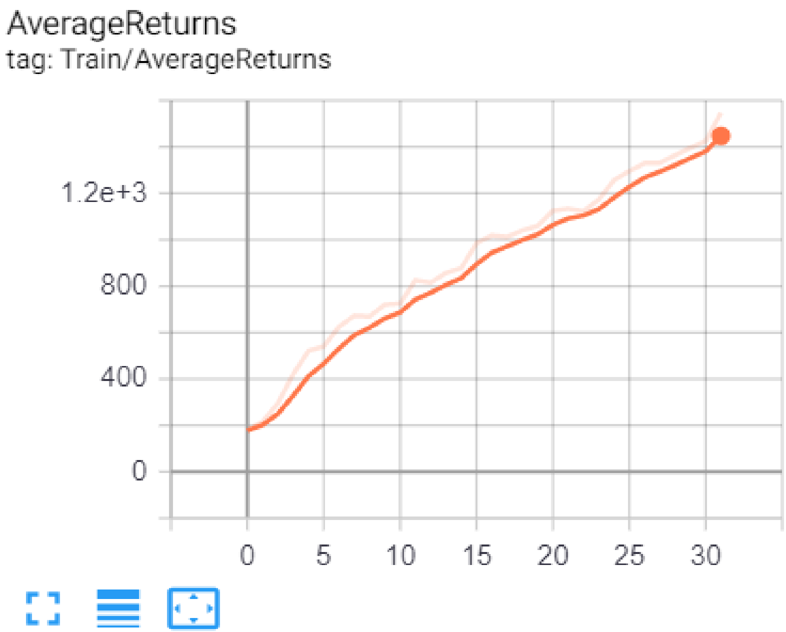
~ Episode 14000

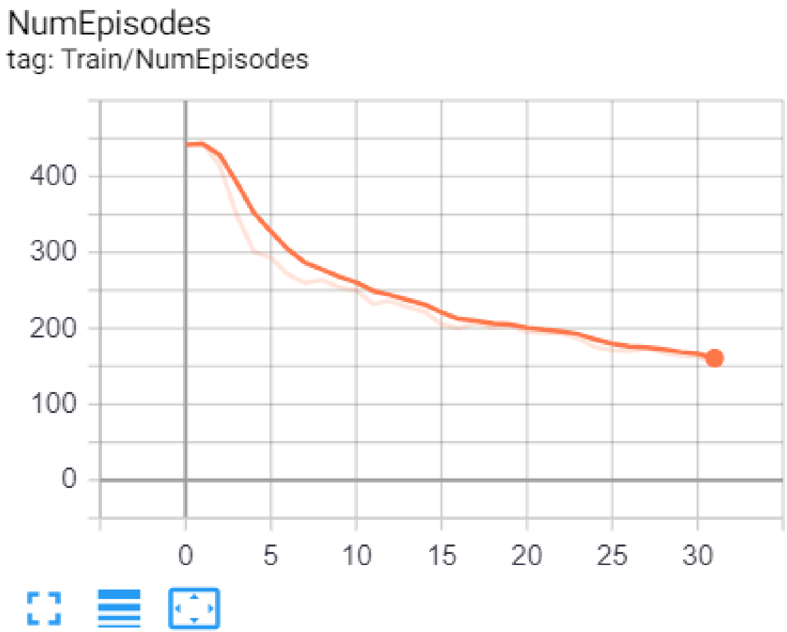


Anyrl – Space Invaders

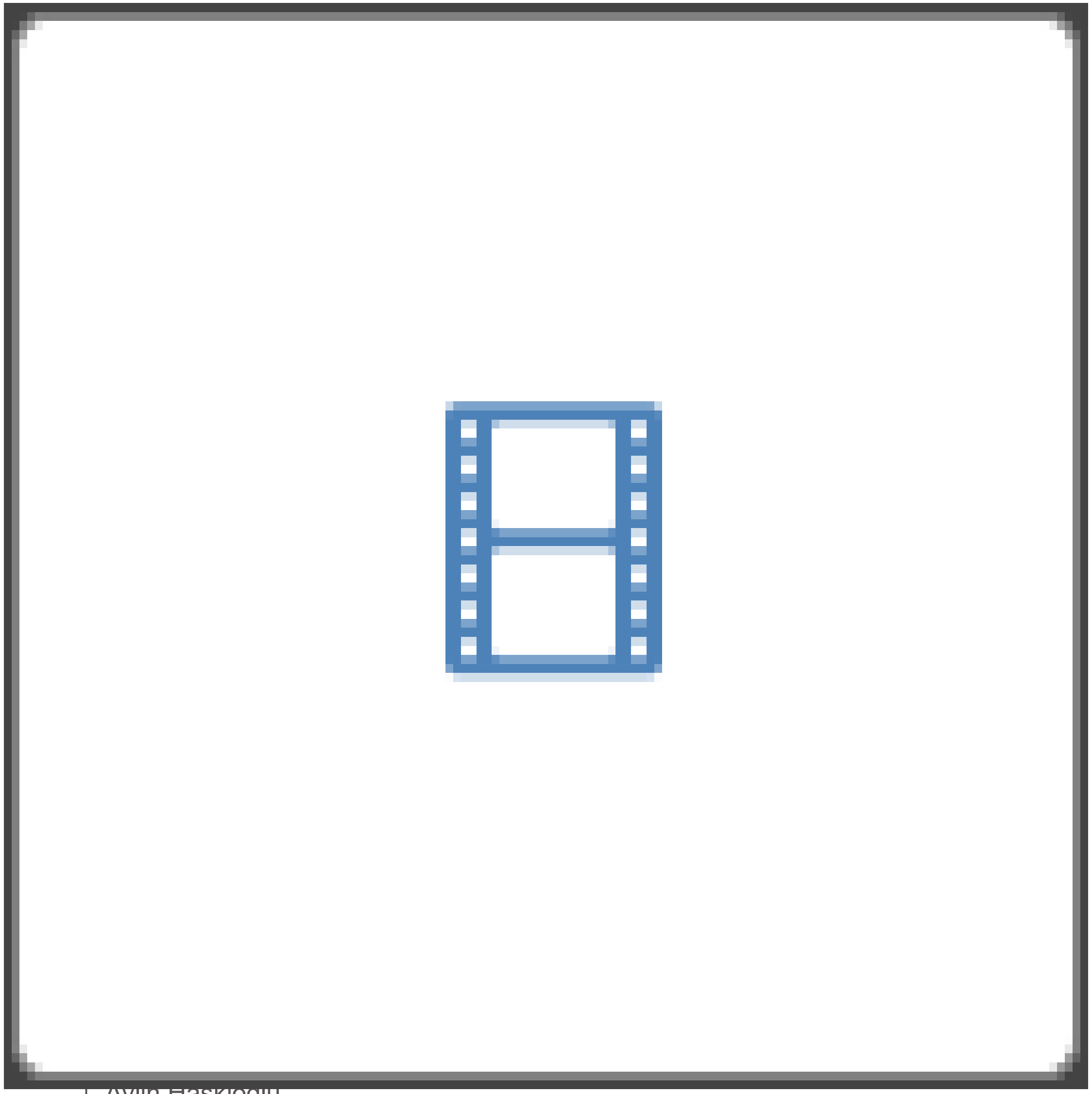


Dopamine


AverageReturns
tag: Train/AverageReturns
1.2e+3
800
400
0
0
5
10
15
20
25
30


NumEpisodes
tag: Train/NumEpisodes
400
300
200
100
0
0
5
10
15
20
25
30

Dopamine



Sources

- Paper:

https://arxiv.org/pdf/1710.02298.pdf

- Github:

https://github.com/AyHaski/DL_AtariRainbow

- Frameworks:

https://github.com/unixpickle/anyrl-py

https://github.com/google/dopamine

https://github.com/astooke/rlpyt

https://github.com/Kaixhin/Rainbow

https://github.com/ray-project/ray/tree/master/rllib

https://github.com/openai/retro-baselines

- Links:

https://www.freecodecamp.org/news/diving-deeper-into-reinforcement-learning-with-q-learning-c18d0db58efe/

https://medium.com/intelligentunit/conquering-openai-retro-contest-2-demystifying-rainbow-baseline-9d8dd258e74b

DQN – Deep Q-learning Network

- Epsilon-greedy strategy
  - Exploration at the beginning
  - Exploitation with time
- Experience Replay
  - All experience is stored in replay memory
  - Random samples are used instead of most recent transistions
- Online/Target Network
  - Online is periodically copied to target network
  - Target → For future rewards computations

Formulas

- Dqn Loss

$$\left(R_{t+1} + \gamma_{t+1} \max_{a'} q_{\bar{\theta}}(S_{t+1}, a') - q_\theta(S_t, A_t)\right)^2$$

- DDQN

$$\left(R_{t+1} + \gamma_{t+1} q_{\bar{\theta}}(S_{t+1}, \operatorname*{argmax}_{a'} q_\theta(S_{t+1}, a')) - q_\theta(S_t, A_t)\right)^2.$$

- Prioritized

$$p_t \propto \left| R_{t+1} + \gamma_{t+1} \max_{a'} q_{\bar{\theta}}(S_{t+1}, a') - q_\theta(S_t, A_t) \right|^\omega$$

- Dueling

$$q_\theta(s, a) = v_\eta(f_\xi(s)) + a_\psi(f_\xi(s), a) - \frac{\sum_{a'} a_\psi(f_\xi(s), a')}{N_{\text{actions}}}$$

- MutliStep

$$R_t^{(n)} \equiv \sum_{k=0}^{n-1} \gamma_t^{(k)} R_{t+k+1} \,. \qquad (2)$$

A multi-step variant of DQN is then defined by minimizing the alternative loss,

$$\left(R_t^{(n)} + \gamma_t^{(n)} \max_{a'} q_{\bar{\theta}}(S_{t+n}, a') - q_\theta(S_t, A_t)\right)^2.$$

Formulas - Integrated Agent

- Distributional $$D_{\mathrm{KL}}(\Phi_{\boldsymbol{z}} d_t^{(n)} || d_t)$$
- Prioritized $$p_t \propto \left(D_{\mathrm{KL}}(\Phi_{\boldsymbol{z}} d_t^{(n)} || d_t)\right)^{\omega}$$
- Dueling $$p_\theta^i(s,a) = \frac{\exp(v_\eta^i(\phi) + a_\psi^i(\phi,a) - \overline{a}_\psi^i(s))}{\sum_j \exp(v_\eta^j(\phi) + a_\psi^j(\phi,a) - \overline{a}_\psi^j(s))}$$

where $\phi = f_\xi(s)$ and $\overline{a}_\psi^i(s) = \frac{1}{N_{\text{actions}}} \sum_{a'} a_\psi^i(\phi, a')$.