Value-Based Reinforcement Learning

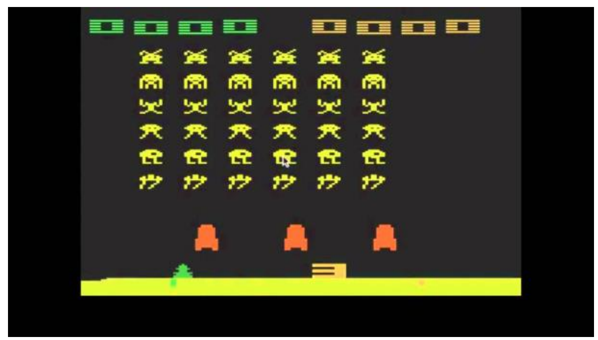
- DQN
- DDQN (Double DQN)
- DRQN
- Dueling Network (with Advantage)

Deep Q Network (DQN)

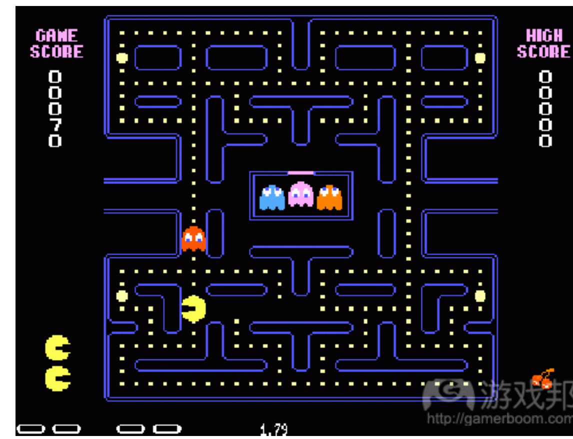
Atari 2600 Games – a Big Success of DQN

- Learn to play Atari games from video only (without knowing the game a priori) by DeepMind, 2013. (in Nature, 2015)

- Atari 2600

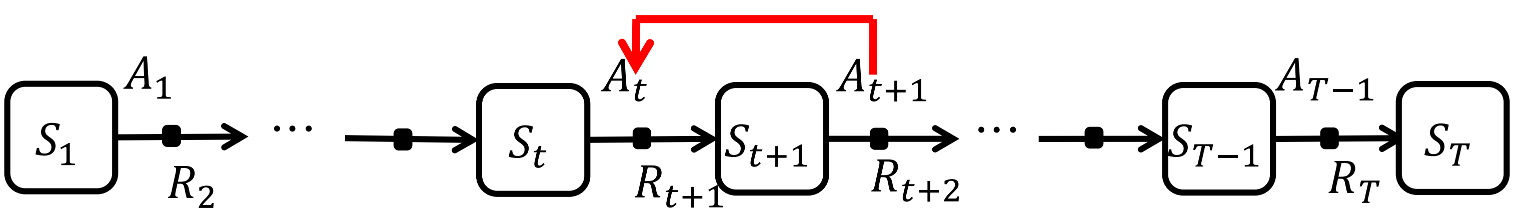


- Space Invader

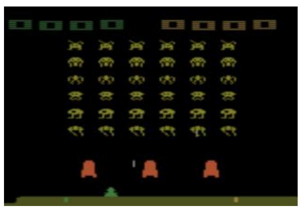


- PacMan

$R_{t+1} + Q(S_{t+1}, A_{t+1})$



Illustrations of DQN



State s → DQN θ

$Q(s, \Leftarrow |\theta) = 200$

$Q(s, \Rightarrow |\theta) = 150$

$Q(s, \Uparrow |\theta) = 300$

......

State s → DQN θ

$Q(s, \Uparrow |\theta) = 50$

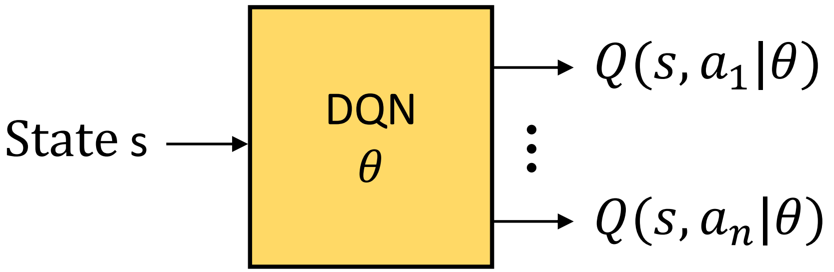
$Q(s, \Downarrow |\theta) = 20$

$Q(s, \Leftarrow |\theta) = 100$

$Q(s, \Rightarrow |\theta) = 70$

Deep Q Network (DQN)

- Single deep network estimates the action value function of each discrete action
  - Action Value: $Q(s_t, a_t|\theta)$
  - Select action: $\arg\max_{a'} Q(s_t, a'|\theta)$
- Target Q (A real number):
  - $Y_t^Q = r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a'|\theta)$
- Loss Function:
  - $L_Q(s_t, a_t|\theta) = \left(Y_t^Q - Q(s_t, a_t|\theta)\right)^2$
- Gradient descent:
  - $\nabla_\theta L_Q(s_t, a_t|\theta) = \left(Y_t^Q - Q(s_t, a_t|\theta)\right) \nabla_\theta Q(s_t, a_t|\theta)$



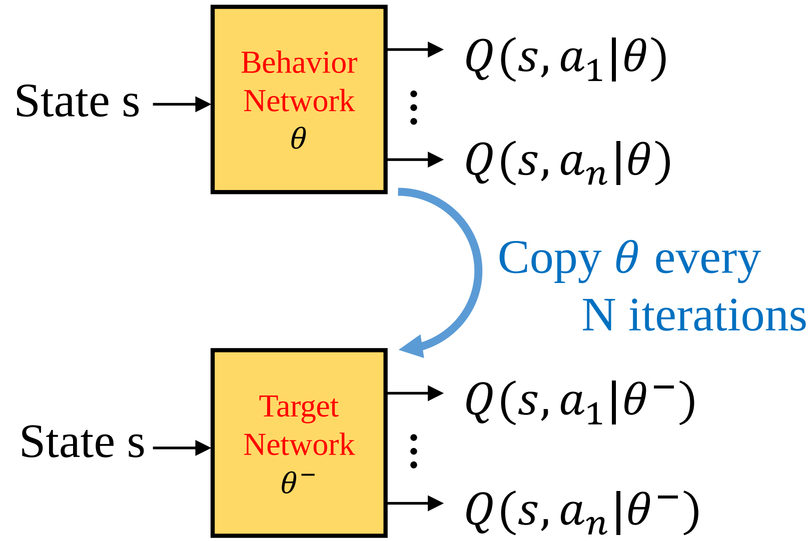
Stability Issues with Deep RL

- Data is sequential (overfitting)
  - Successive samples are correlated, non-iid
- Policy changes rapidly with slight changes to Q-values (hard to converge)
  - Policy may oscillate
  - Distribution of data can swing from one extreme to another
- Scale of rewards and Q-values is unknown (not normalized)
  - Naive Q-learning gradients can be large and unstable when backpropagated

Solution for Stability

- Use experience replay
  - Break correlations in data, bring us back to iid setting
  - Learn from all past policies
- Freeze target Q-network
  - Avoid oscillations
  - Break correlations between Q-network and target
- Clip rewards or normalize network adaptively to sensible range (simply normalization, not discussed here)
  - Robust gradients

Deep Q Network (DQN)



- Techniques
  1. Target Network with parameters θ^-
  2. Experience Replay
     - Sample experiences at random.

- Apply Target Network on DQN:
  - $Y_t^Q = r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a'|\theta^-)$
  - Gradient descent on behavior network:
    - $\nabla_\theta L_Q(s_t, a_t|\theta) = \left(Y_t^Q - Q(s_t, a_t|\theta)\right) \nabla_\theta Q\left(s_y, a_t|\theta\right)$
  - Copy parameters from θ to θ^- every N iterations (updates).
    - Ex. N=1000

Behavior and target network

Algorithm 1: deep Q-learning with experience replay.

Initialize replay memory D to capacity N
Initialize action-value function Q with random weights θ
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 ε select a random action a_t
    otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$
    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 D
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D
    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^-) & \text{otherwise} \end{cases}$
    Perform a gradient descent step on $\left(y_j - Q(\phi_j, a_j; \theta)\right)^2$ with respect to the network parameters θ
    Every C steps reset $\hat{Q} = Q$
  End For
End For

Algorithm 1: deep Q-learning with experience replay.
Initialize replay memory D to capacity N
Initialize action-value function Q with random weights θ
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**

ε-greedy based on behavior network

    $\boxed{\text{With probability } \varepsilon \text{ select a random action } a_t \text{ otherwise select } a_t = \mathrm{argmax}_a Q(\phi(s_t), a; \theta)}$
    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 D
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D
    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step } j+1 \\ r_j + \gamma \max_{a'} \hat{Q}\left(\phi_{j+1}, a'; \theta^{-}\right) & \text{otherwise} \end{cases}$
    Perform a gradient descent step on $\left(y_j - Q\left(\phi_j, a_j; \theta\right)\right)^2$ with respect to the network parameters θ
    Every C steps reset $\hat{Q} = Q$
  End For
End For

Algorithm 1: deep Q-learning with experience replay.

Initialize replay memory D to capacity N

Initialize action-value function Q with random weights θ

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 ε select a random action a_t

otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$

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 $\phi_{t+1} = \phi(s_{t+1})$

Experience replay

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

$$\text{Set } y_j = \begin{cases} r_j & \text{if episode terminates at step } j+1 \\ r_j + \gamma \max_{a'} \hat{Q}\left(\phi_{j+1}, a'; \theta^-\right) & \text{otherwise} \end{cases}$$

Perform a gradient descent step on $\left(y_j - Q\left(\phi_j, a_j; \theta\right)\right)^2$ with respect to the network parameters θ

Every C steps reset $\hat{Q} = Q$

End For

End For

Algorithm 1: deep Q-learning with experience replay.

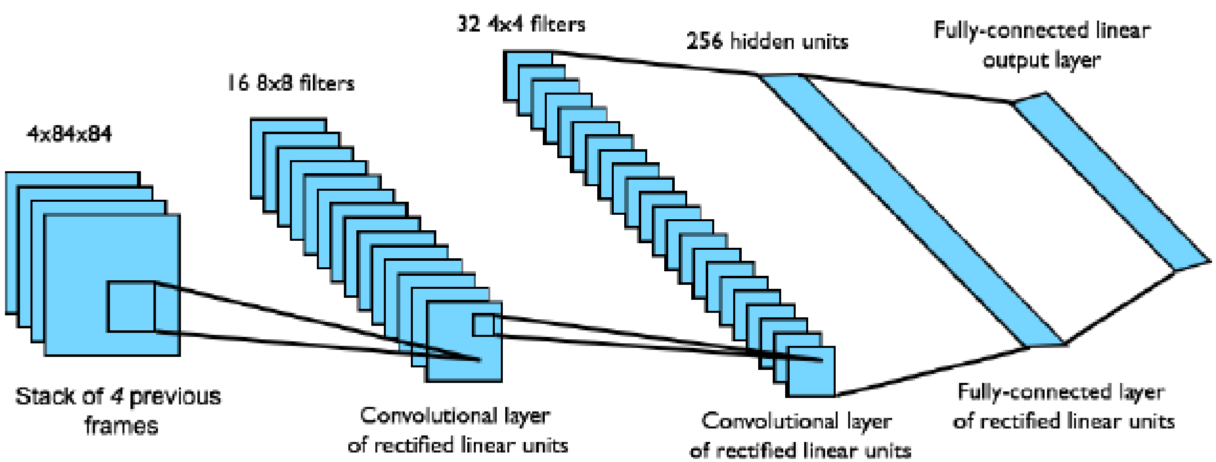
Initialize replay memory D to capacity N
Initialize action-value function Q with random weights θ
Initialize target action-value function $\hat{Q}$ with weights $\theta^{-} = \theta$
For episode = 1, M **do**
    Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
    For t = 1,T **do**
        With probability ε select a random action a_t
        otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$
        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 D
        Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D
        Set $y_j = \begin{cases} r_j & \text{if episode terminates at step } j+1 \\ r_j + \gamma \max_{a'} \hat{Q}\left(\phi_{j+1}, a'; \theta^{-}\right) & \text{otherwise} \end{cases}$
        Perform a gradient descent step on $\left(y_j - Q\left(\phi_j, a_j; \theta\right)\right)^2$ with respect to the network parameters θ
        Every C steps reset $\hat{Q} = Q$
    End For
End For

Update the behavior network

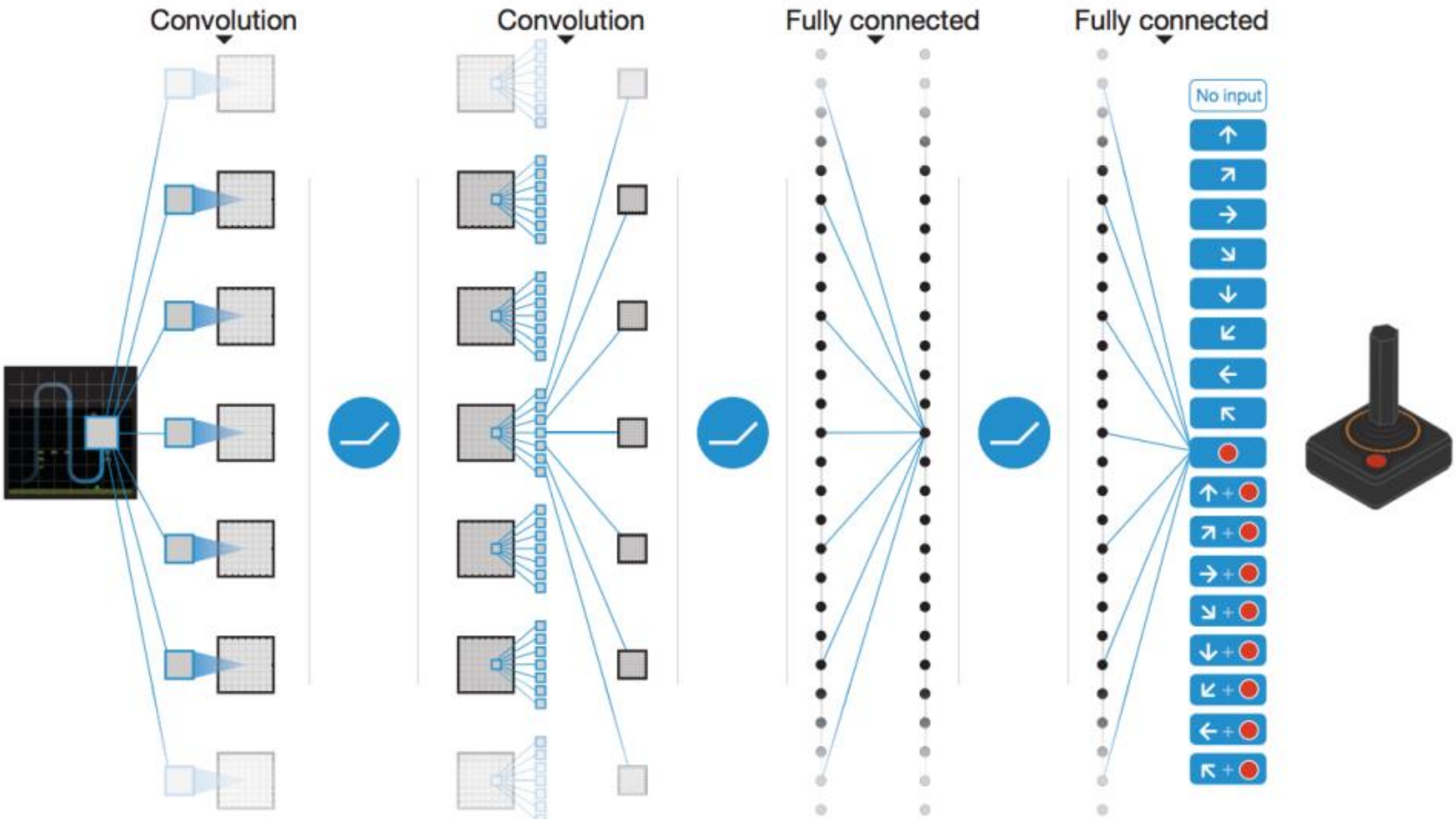
I-Chen Wu

DQN in Atari

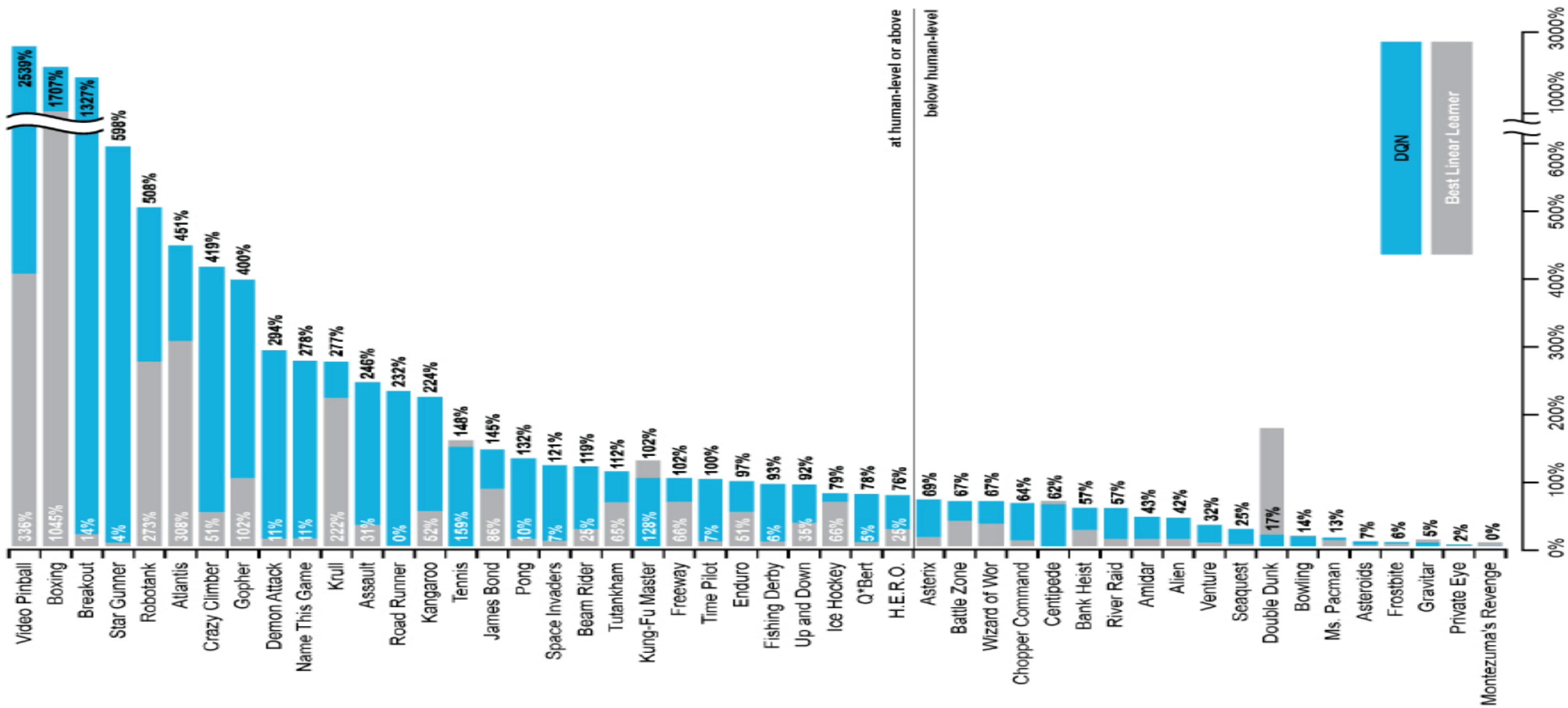
- End-to-end learning of values $Q(s, a)$ from pixels s
- Input state s is stack of raw pixels from last 4 frames
- Output is $Q(s, a)$ for 18 joystick/button positions
- Reward is change in score for that step



Network architecture and hyperparameters fixed across all games


Convolution
Convolution
Fully connected
Fully connected
No input

DQN Results in Atari



How much does DQN help?

	Q-learning	Q-learning +Target Q	Q-learning + Replay	Q-learning + Replay +Target Q
Breakout	3	10	241	317
Enduro	29	142	831	1006
River	1453	2868	4103	7447
Seaquest	276	1003	823	2894
Space Invaders	302	373	826	1089

Experiments - DQN

	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

- The upper table compares average total reward for various learning methods by running an ε-greedy policy with $\varepsilon = 0.05$ for a fixed number of steps.

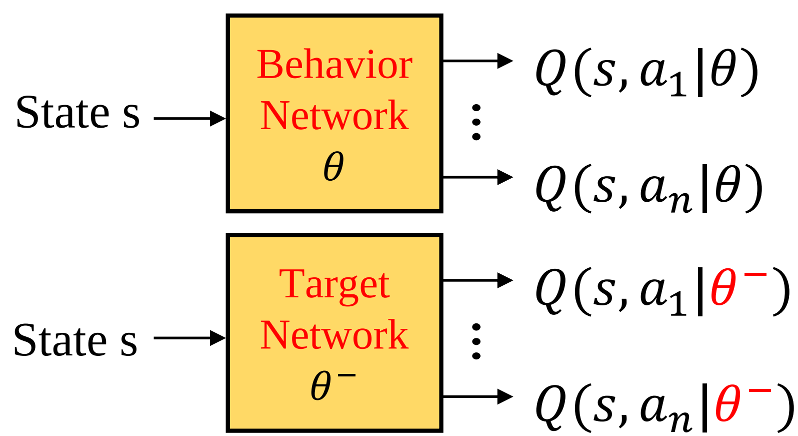
Double DQN

Double DQN (DDQN)

- Prevent over-optimistic value estimates on DQN.
- Decouple the selection from the evaluation.

$$Y_t^Q = r_{t+1} + \gamma \max_a Q(S_{t+1}, \boxed{a} | \theta^-)$$

$$Y_t^{DoubleQ} = r_{t+1} + \gamma Q\left(S_{t+1}, \boxed{\operatorname*{argmax}_a Q(S_{t+1}, a | \theta)} | \theta^-\right)$$



Overestimation Problem

- Q-Learning update

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

Algorithm 1: Double DQN Algorithm.

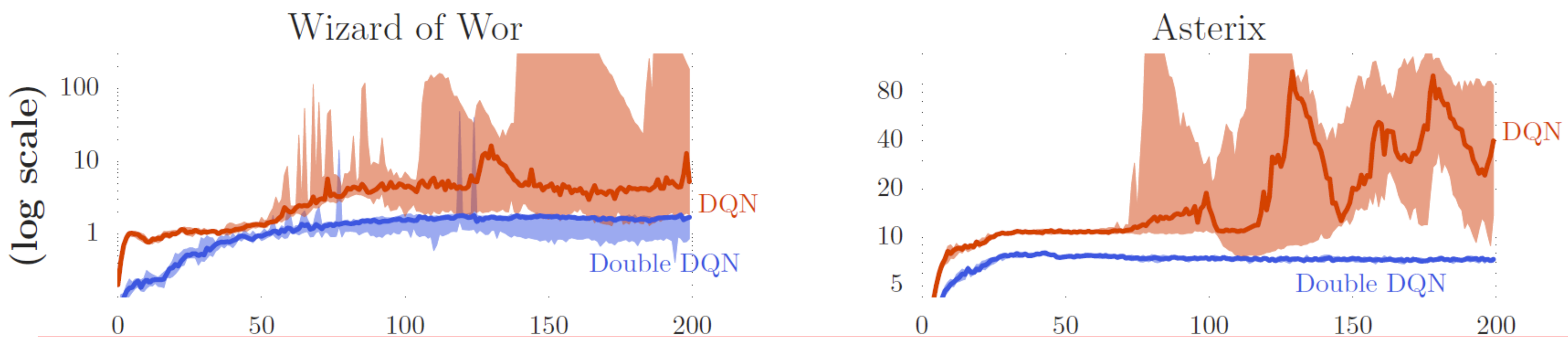
input : $\mathcal{D}$ – empty replay buffer; θ – initial network parameters, θ^- – copy of θ
input : N_r – replay buffer maximum size; N_b – training batch size; N^- – target network replacement freq.
for *episode* $e \in \{1, 2, \ldots, M\}$ **do**
    Initialize frame sequence $\mathbf{x} \leftarrow ()$
    for $t \in \{0, 1, \ldots\}$ **do**
        Set state $s \leftarrow \mathbf{x}$, sample action $a \sim \pi_{\mathcal{B}}$
        Sample next frame x^t from environment $\mathcal{E}$ given (s, a) and receive reward r, and append x^t to $\mathbf{x}$
        if $|\mathbf{x}| > N_f$ **then** delete oldest frame $x_{t_{min}}$ from $\mathbf{x}$ **end**
        Set $s' \leftarrow \mathbf{x}$, and add transition tuple (s, a, r, s') to $\mathcal{D}$,
            replacing the oldest tuple if $|\mathcal{D}| \geq N_r$
        Sample a minibatch of N_b tuples $(s, a, r, s') \sim \mathrm{Unif}(\mathcal{D})$
        Construct target values, one for each of the N_b tuples:
        Define $a^{\max}(s'; \theta) = \arg\max_{a'} Q(s', a'; \theta)$

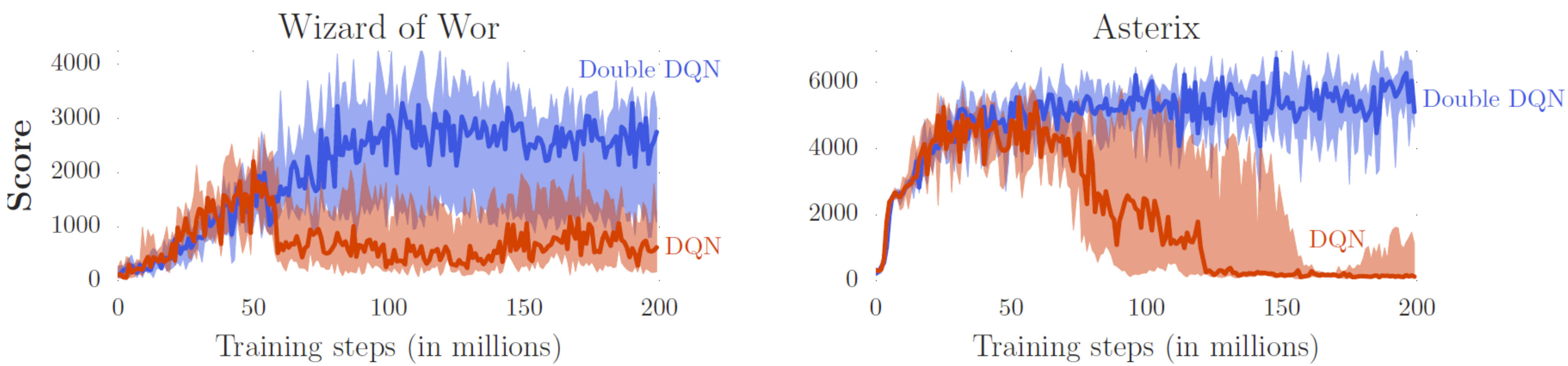
$$y_j = \begin{cases} r & \text{if } s' \text{ is terminal} \\ r + \gamma Q(s', a^{\max}(s'; \theta); \theta^-), & \text{otherwise.} \end{cases}$$

        Do a gradient descent step with loss $\|y_j - Q(s, a; \theta)\|^2$
        Replace target parameters $\theta^- \leftarrow \theta$ every N^- steps
    end
end

Experiments - DDQN

Predicted Q-value at training (showing over-optimism)



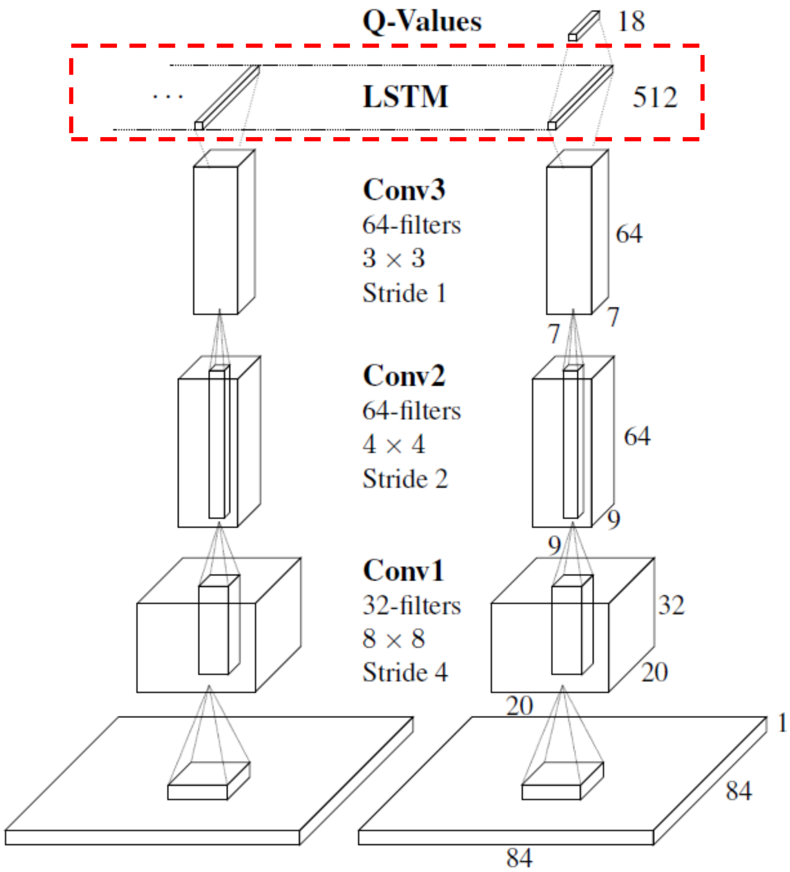


Real Scores

DRQN

Deep Recurrent Q-Network (DRQN)

- Replace only its first fully-connected layer with a LSTM.
- Take a single 84 x 84 preprocessed image (not 4 consecutive images)
- Finally, LSTM outputs become Q-values for each action after passing through a fully-connected layer.



Update for DRQN

Update: episodes are selected randomly from the replay memory (for example, choose an episode with $s_1, \dots, s_{200}$ states)

- Sequential:
  - Updates begin at the beginning of the episode to the end of the episode (always start from s_1)
  - Good for LSTM, but violate DQN's random sampling policy
- Random:
  - Updates begin at random points in the episode and proceed for only *unroll iterations* time-steps (randomly pick s_i)
  - LSTM's hidden state must be zeroed at the start of each update. Harder for the LSTM to learn.
- Both have similar performance (use random here)

DRQN Results

- It can generalize its policies to the case of complete observations (on Flickering Pong)
- DRQN's performance generalizes better than DQN's at all levels of partial information

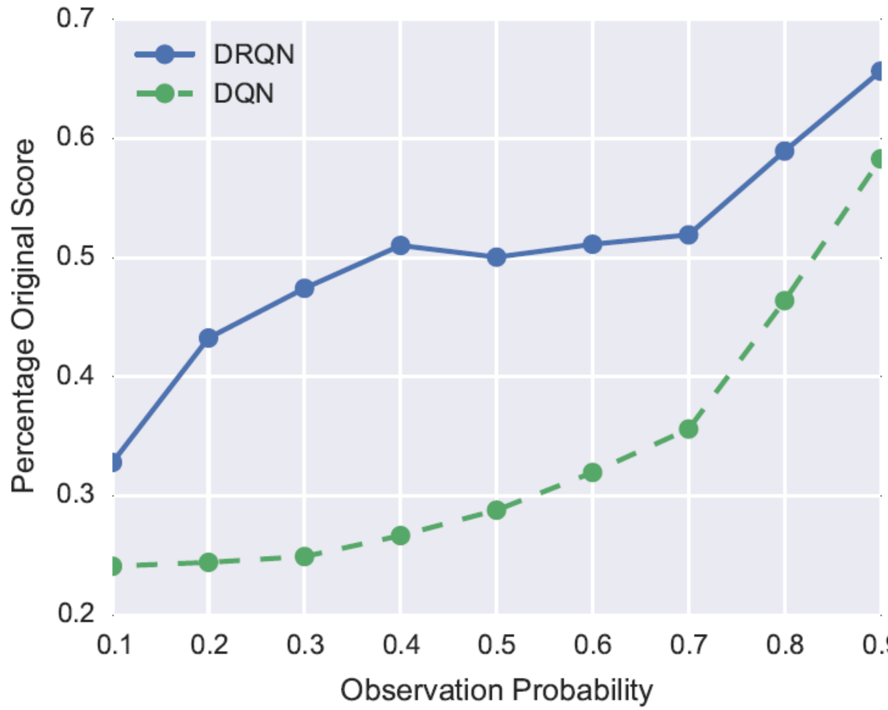


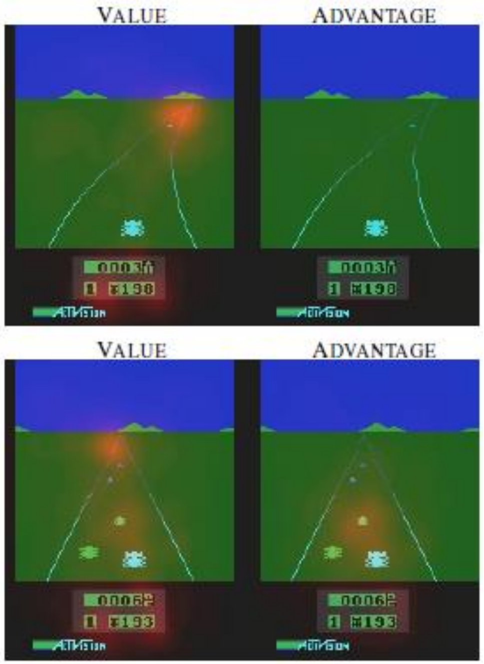
Figure 5: When trained on normal games (MDPs) and then evaluated on flickering games (POMDPs), DRQN's performance degrades more gracefully than DQN's.

I-Chen Wu

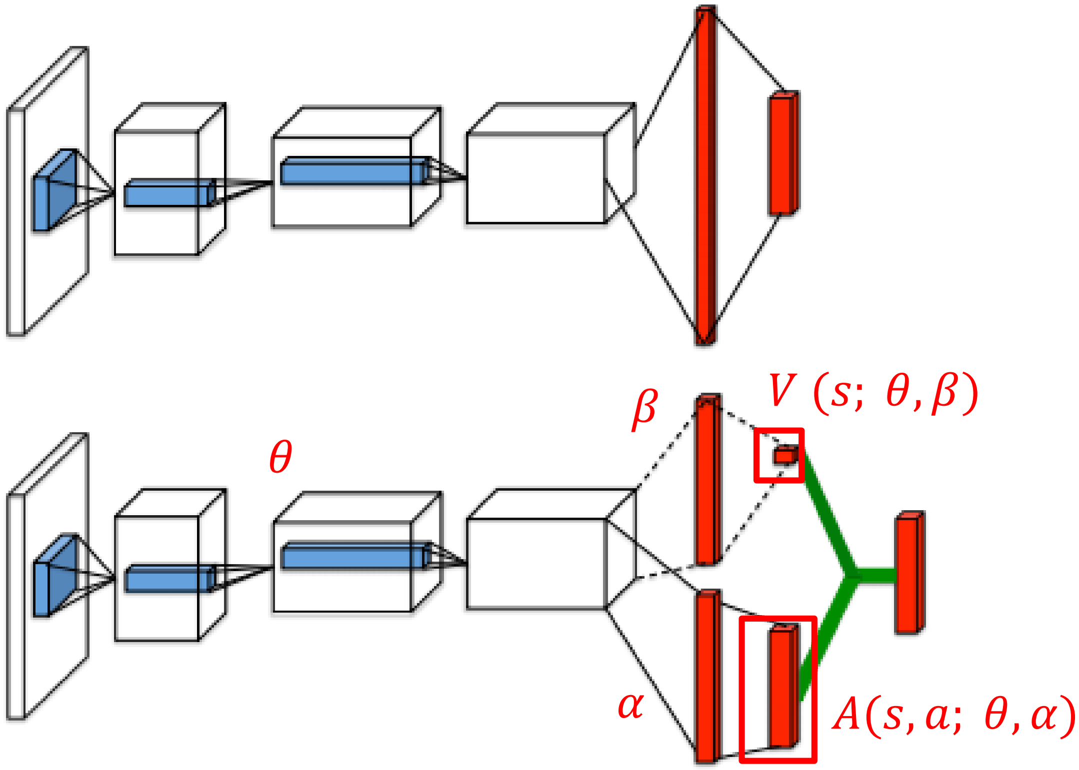
Dueling Network (with Advantage)

Dueling Network

- In most states, learning the effect of each action is not necessary.
  - Actions do not affect the environment in any relevant way
- Intuitively, the dueling architecture can learn whether states are valuable (or not).
  - Advantage stream
  - Value stream



Dueling Network



Q-network (top) and the dueling Q-network (bottom). The dueling network has two streams to separately estimate (scalar) state-value and the advantages for each action; the green output module combines them by the equation $Q(s,a;\,\theta,\alpha,\beta) = V\,(s;\,\theta,\beta) + A(s,a;\,\theta,\alpha)$. Both networks output Q-values for each action.

Dueling Network

- A relative measure of the importance of each action
  - $Q(s,a;\ \theta,\alpha,\beta) = V\ (s;\ \theta,\beta)\ + A(s,a;\ \theta,\alpha)$
    - Unidentifiable in the sense that given Q we cannot recover V and A uniquely.
- Address the issue of identifiability
  - $Q(s,a;\ \theta,\alpha,\beta) =\ V\ (s;\ \theta,\beta)\ + (A(s,a;\ \theta,\alpha) - \max_{a' \in |A|} A(s,a';\ \theta,\alpha))$
    - Force the advantage function estimator have zero advantage at the chosen action
    - When $a^* = \max_{a'} Q(s,a')$, $Q(s,a^*) = V(s)$
- Improvement (increase stability)
  - $Q(s,a;\ \theta,\alpha,\beta)\ =\ V\ (s;\ \theta,\beta)\ + (A(s,a;\ \theta,\alpha) - \frac{1}{|A|}\sum_{a'} A(s,a'\ ;\ \theta,\alpha))$
    - When $a^* = \max_{a'} Q(s,a')$, $Q(s,a^*) \neq V(s)$
    - The advantages only need to change as fast as the mean.

Experiments – Dueling Network

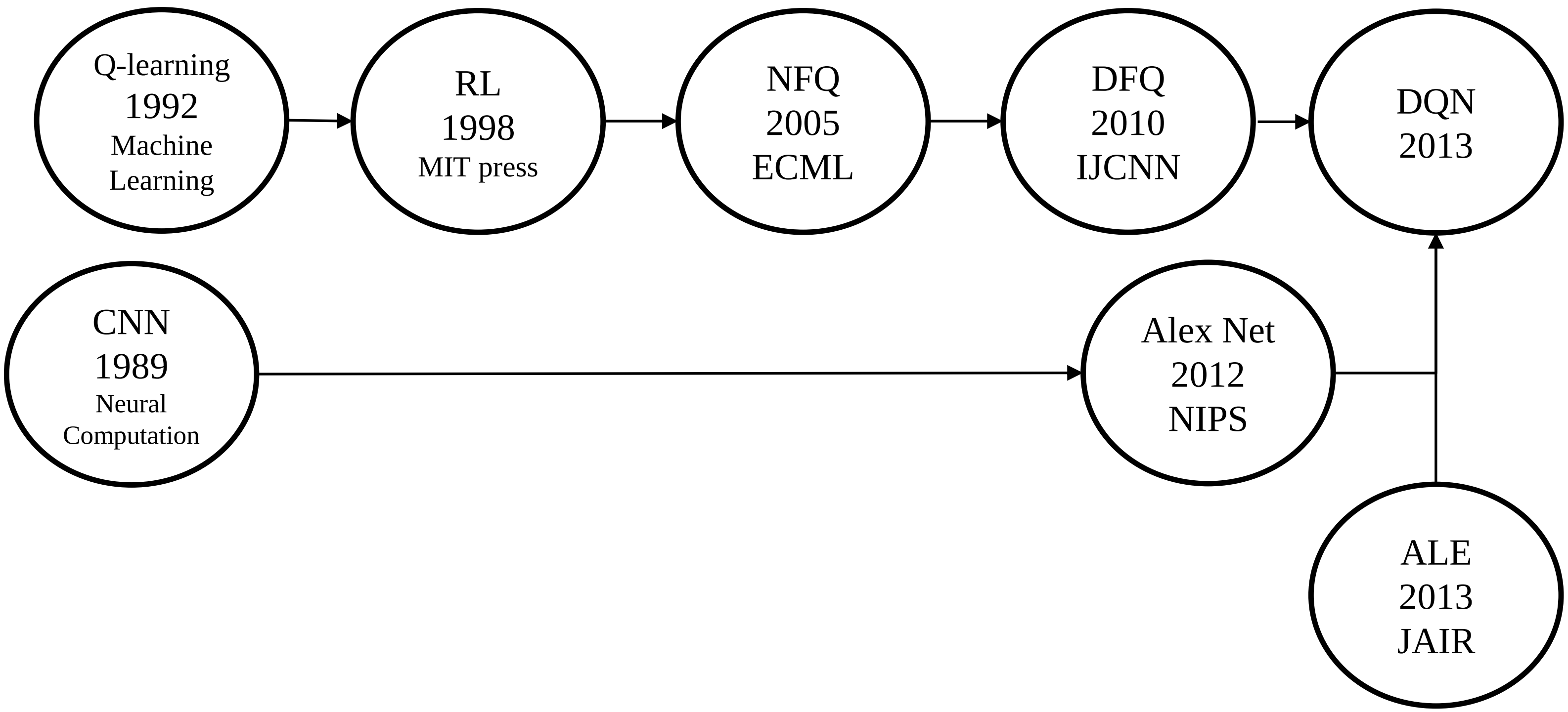
- Achieve human level performance on **42 out of 57** games

	30 no-ops		Human Starts	
	Mean	**Median**	**Mean**	**Median**
Prior. Duel Clip	**591.9%**	**172.1%**	**567.0%**	**115.3%**
Prior. Single	434.6%	123.7%	386.7%	112.9%
Duel Clip	**373.1%**	**151.5%**	**343.8%**	**117.1%**
Single Clip	341.2%	132.6%	302.8%	114.1%
Single	307.3%	117.8%	332.9%	110.9%
Nature DQN	227.9%	79.1%	219.6%	68.5%

Measured in percentage of human performance

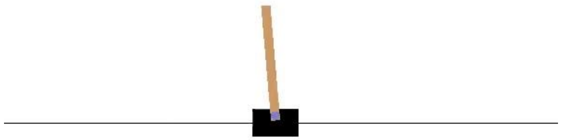
DQN – Genealogy

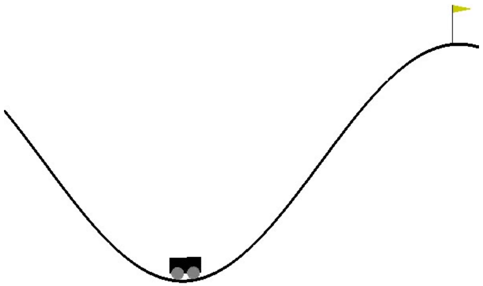
DQN – Genealogy



Before DQN

- NFQ (Neural Fitted Q Iteration, 2005)
  - *First Experiences with a Data Efficient Neural Reinforcement Learning Method*
  - Using neural network (MLP)
    - 2 hidden layers with 5 neurons
  - Using experience replay
    - collected in triples of the form (s,a,s′)
  - Internal state:





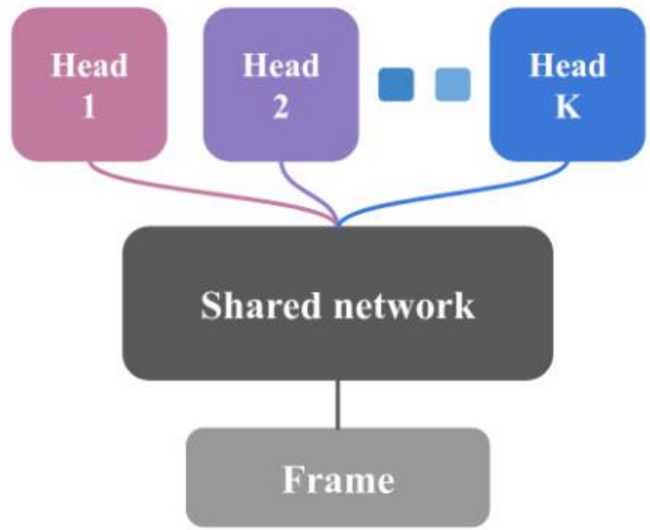
- DFQ (Deep Fitted Q Iteration, 2010)
  - Applying deep learning (MLP, but not CNN)
    - Deep auto-encoders
      - 21 layers
        - 900-900-484-225-121-113-57-29-15-8-2-8-15-29-57-113-121-225-484-900-900 neurons
  - AutoEncoder visualization (2D latent space) to train policy.

DQN – Summary

- Using convolutional neural network (CNN)
  - Alex Net (2012)
    - ReLU
    - GPU
- Arcade Learning Environment (ALE), 2013
  - For Atari games
  - An Evaluation Platform for General Agents
- Using experience replay

Bootstrapped DQN – Summary

- Bootstrapped with k-heads DQN
- No ϵ-greedy
  - ϵ select a head at episode initial
    - Greedy with this head
  - Deep exploration
- * A way go to distribution
  - Bootstrapped distribution

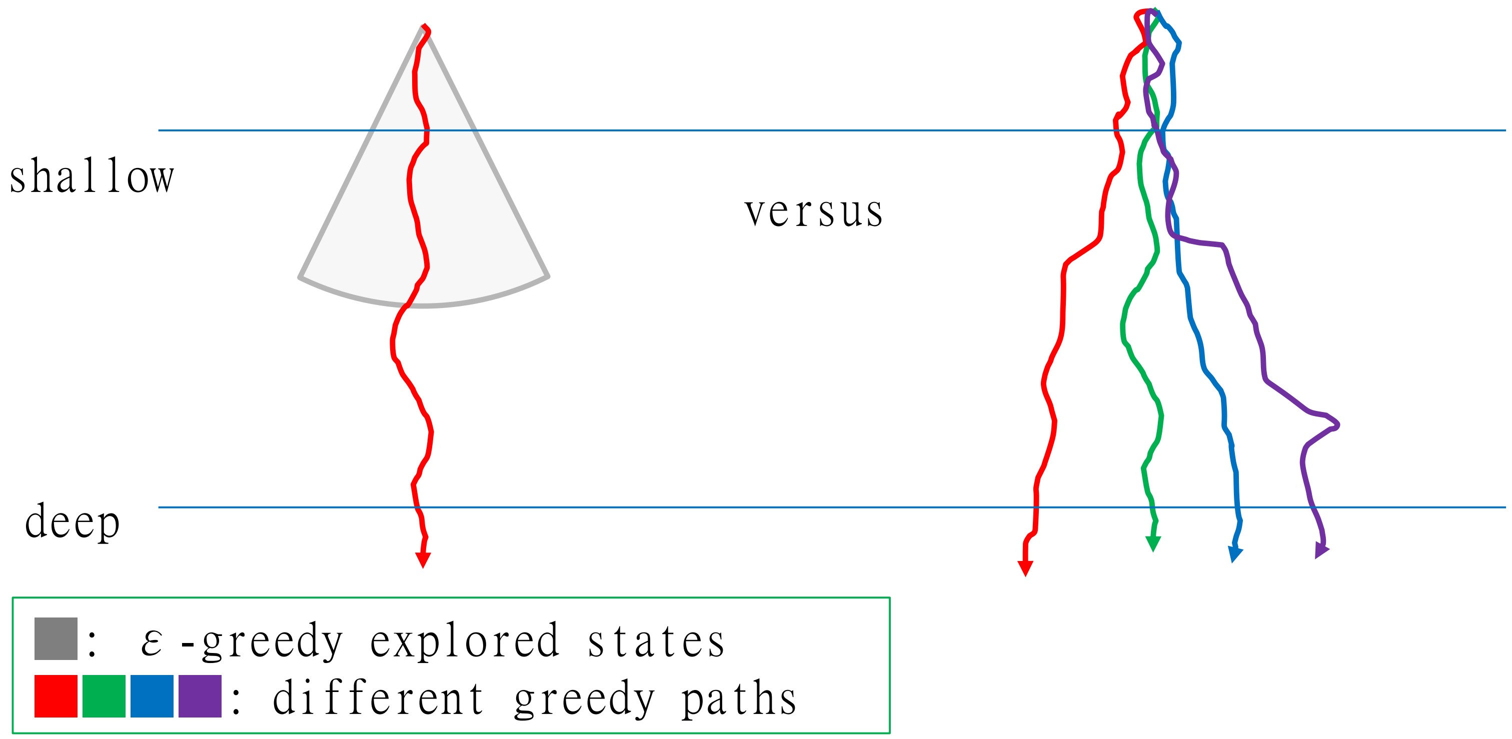


Bootstrapped DQN – Method

Algorithm 1 Bootstrapped DQN

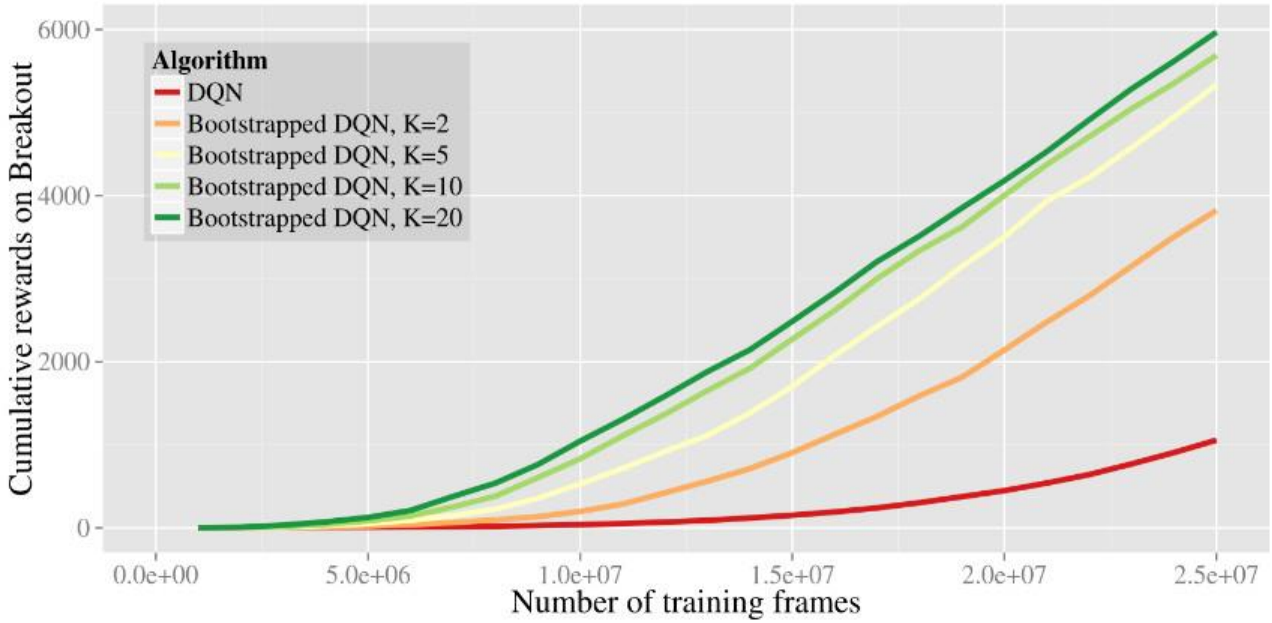
Input: neural network $(Q_k)_{k=1}^{K}$, sampling distribution P
for each episode **do**
    Update network parameters via minibatches
    Sample $k \sim \text{Uniform}\{1,\ldots,K\}$
    while not end of episode **do**
        Choose $a_t \in \text{argmax}_a Q_k(s_t, a)$
        Receive state s_{t+1} and reward r_t from environment
        Sample bootstrap mask $m_t^k \sim P$ for all k
        Add (a_t, r_t, s_{t+1}, m_t) to replay buffer
    end while
end for

Bootstrapped DQN – Deep Exploration



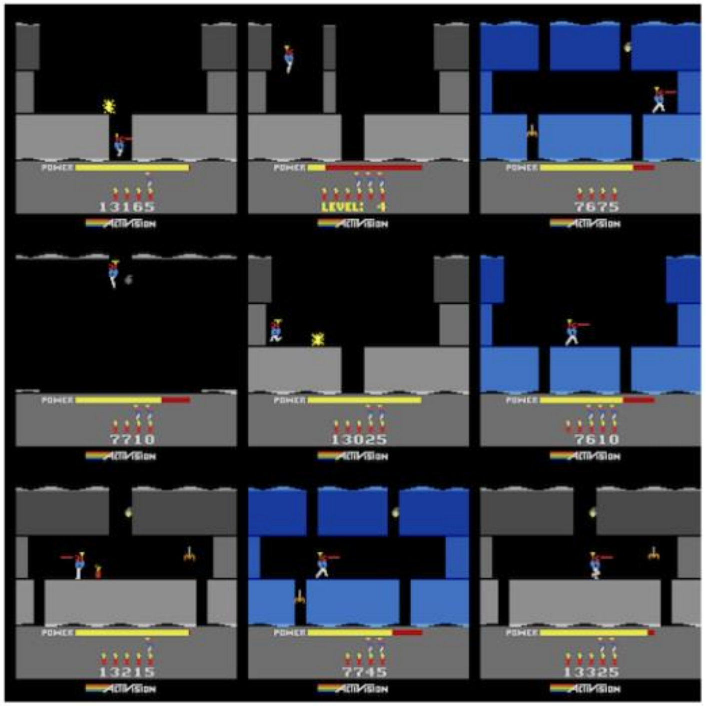
Bootstrapped DQN – Result

- Compare to DQN (Different head count, in Breakout)

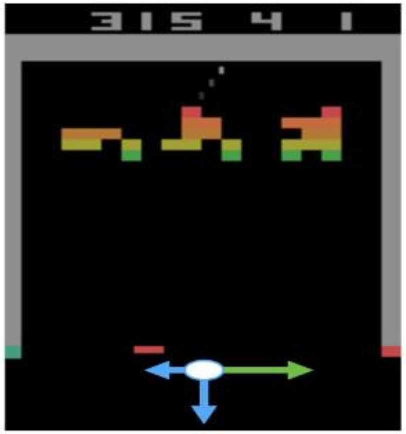


Bootstrapped DQN – Diverse exploration policies

- Compare to DQN (faster, stronger)



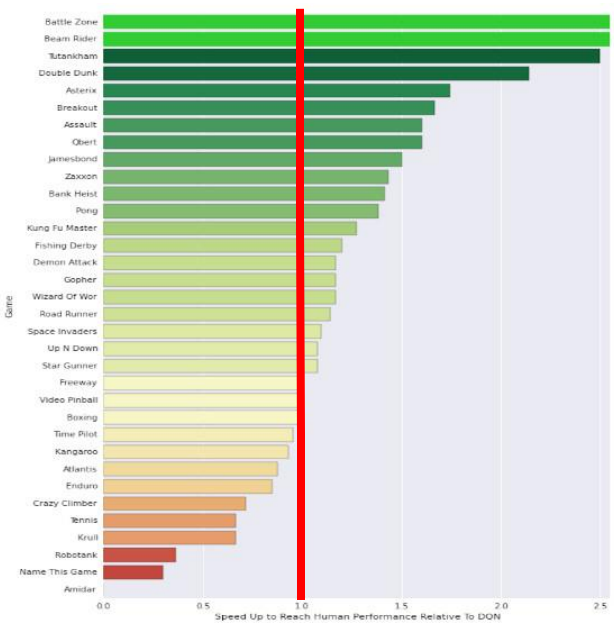
(a) All heads vote right.



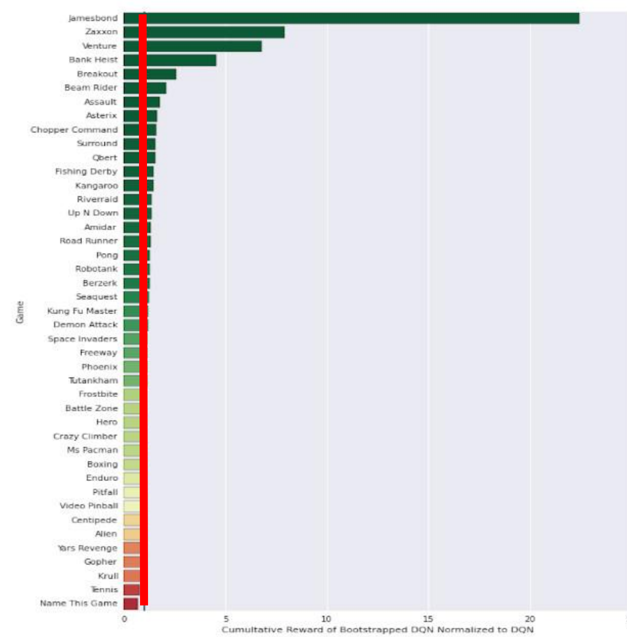
(b) Heads disagree on policy.

Bootstrapped DQN - Result

- Compare to DQN (faster, stronger)



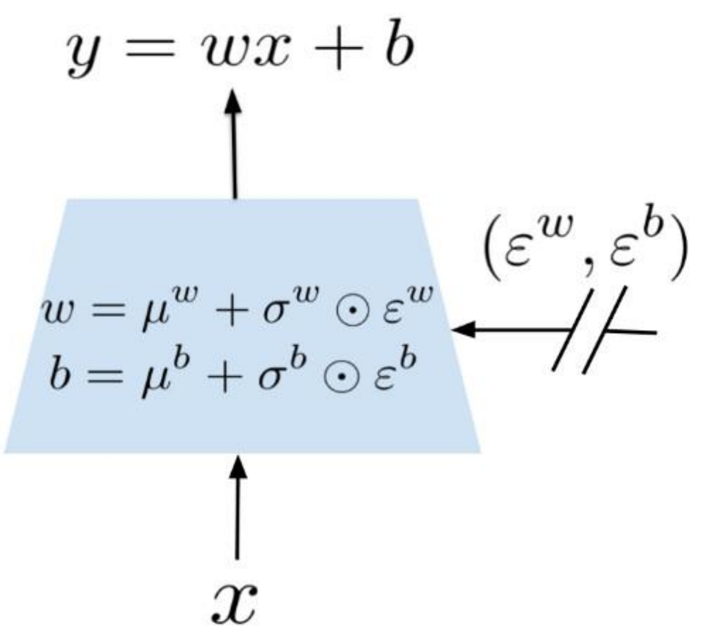
Bootstrapped DQN at human level faster than DQN.



Bootstrapped DQN improves cumulative rewards.

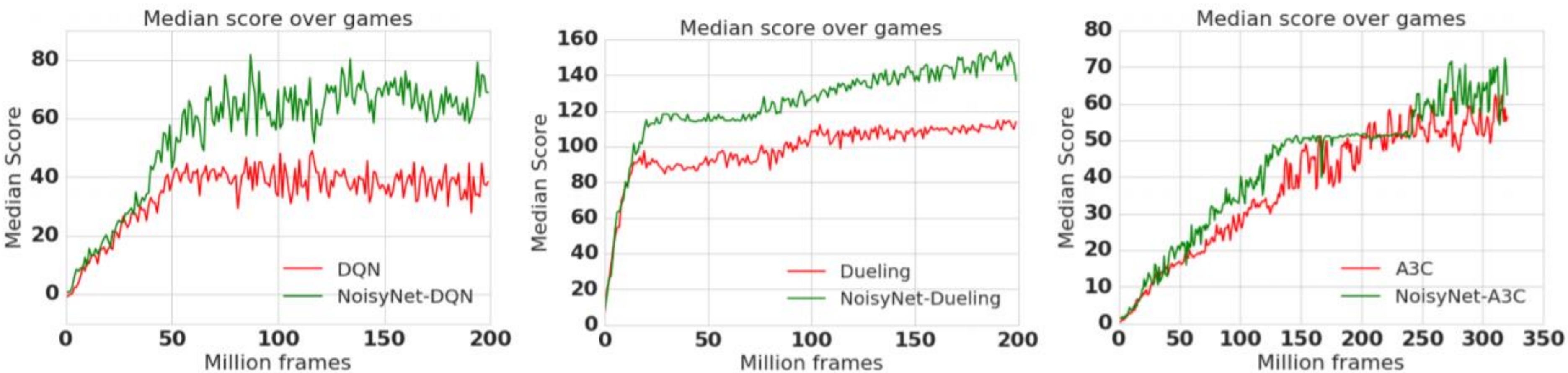
NoisyNet – Summary

- Add trainable noise to neural network
  - Stochastic policy (through latent state space)
    - No ϵ-greedy
  - Better optimization
- A way go to distribution
  - [ICLR 2018]

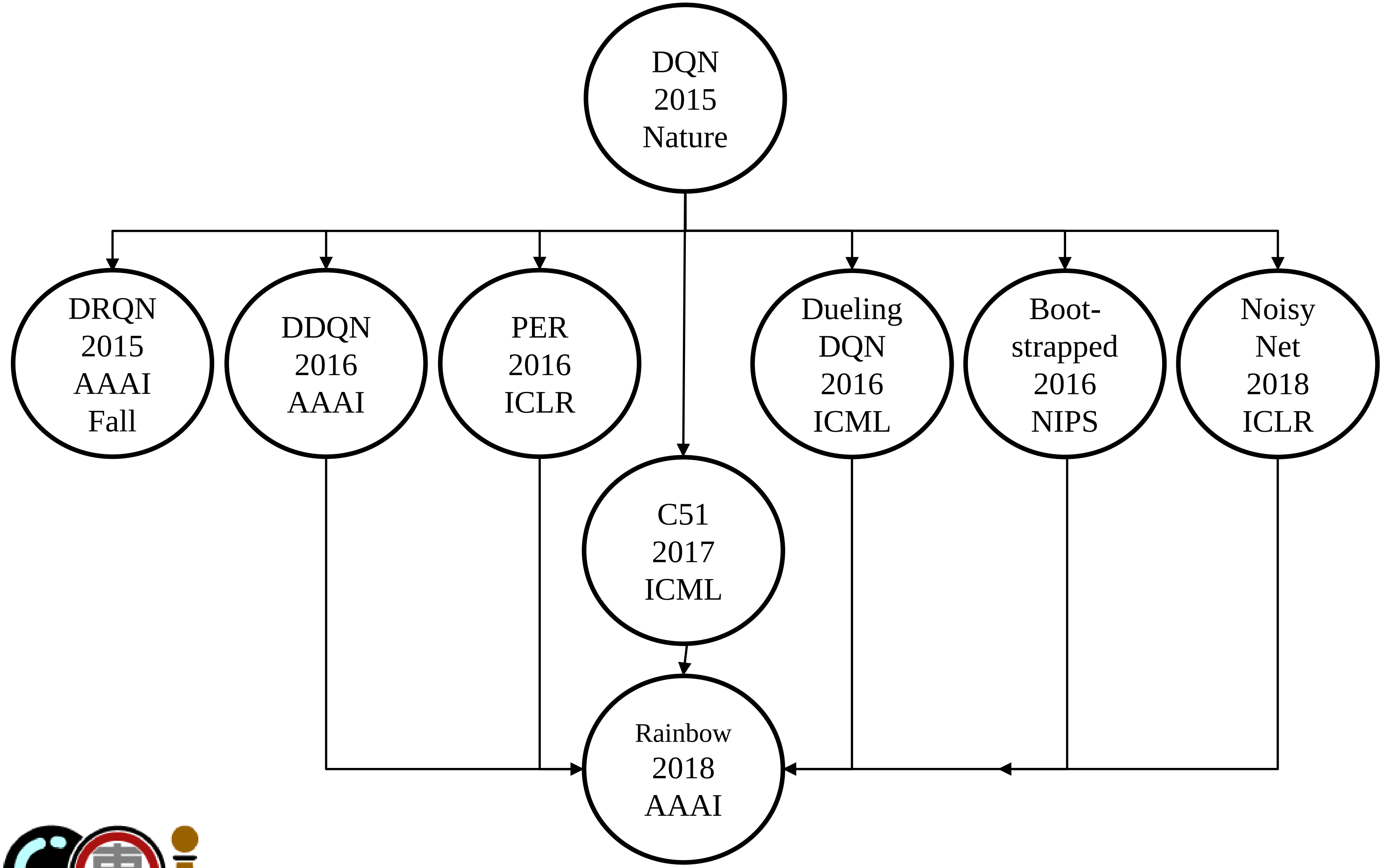


NoisyNet – Result

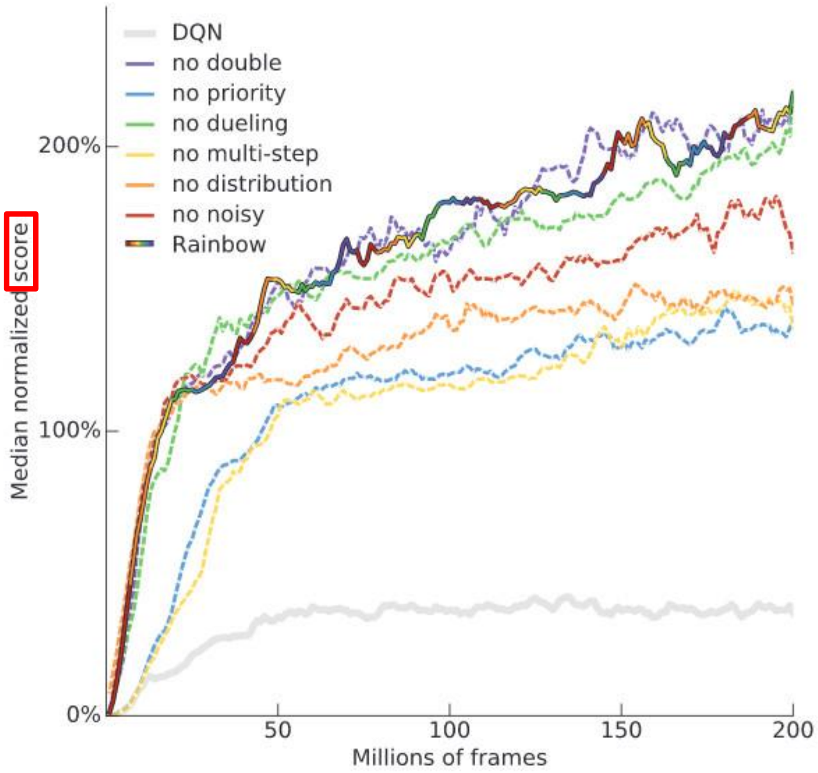
- Better than DQN & Dueling (using e-greedy)
- But, close to A3C.



Rainbow - Genealogy



Rainbow – Ablation studies



- Prioritized replay and multi-step
  - the two most crucial components
- Distributional Q-learning
  - Perform after 40 million frames
  - Relatively to human performance
- Noisy Nets
  - ϵ- greedy when removed
  - large drop in performance for several games
- Dueling network
  - median score/above-human performance levels may hide the impact
- Double Q-learning
  - Actual returns are often higher than10
    - Underestimated
  - May increase if the support of the distributions is expanded