Deep Q-Networks

Christopher Mutschler

Value Function Approximation (VFA)

- Idea: Why don‘t we replace linear approximation with NNs?
  - Because theory tells us that this doesn‘t work out

Algorithm	Table Lookup	Linear	Non-linear
Monte-Carlo Control	✓	(✓)	✗
SARSA	✓	(✓)	✗
Q-learning	✓	✗	✗

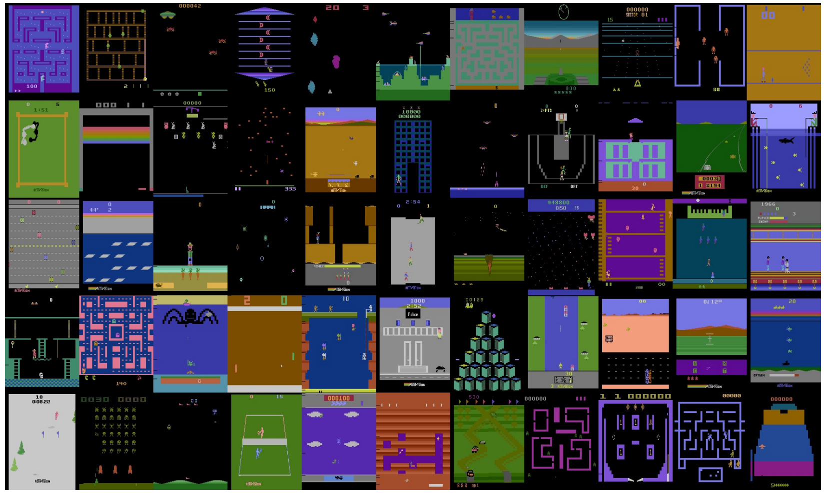
(✓) =chatters around near-optimal value function

- Besides some few hand-crafted and tuned successes NNs have not been managed to be applied “as is” to RL problems

Deep Q-Networks (DQNs)

- Then a game-changing result was published in Nature:
  DQN from DeepMind (now Google DeepMind)
  - Surpassed human player in 49 games of the Atari 2600 series
  - Same RL algorithm to learn a policy in each game
  - **End-to-end:** Only image pixels as input
  - “The“ contribution that initiated a round of huge investments in RL

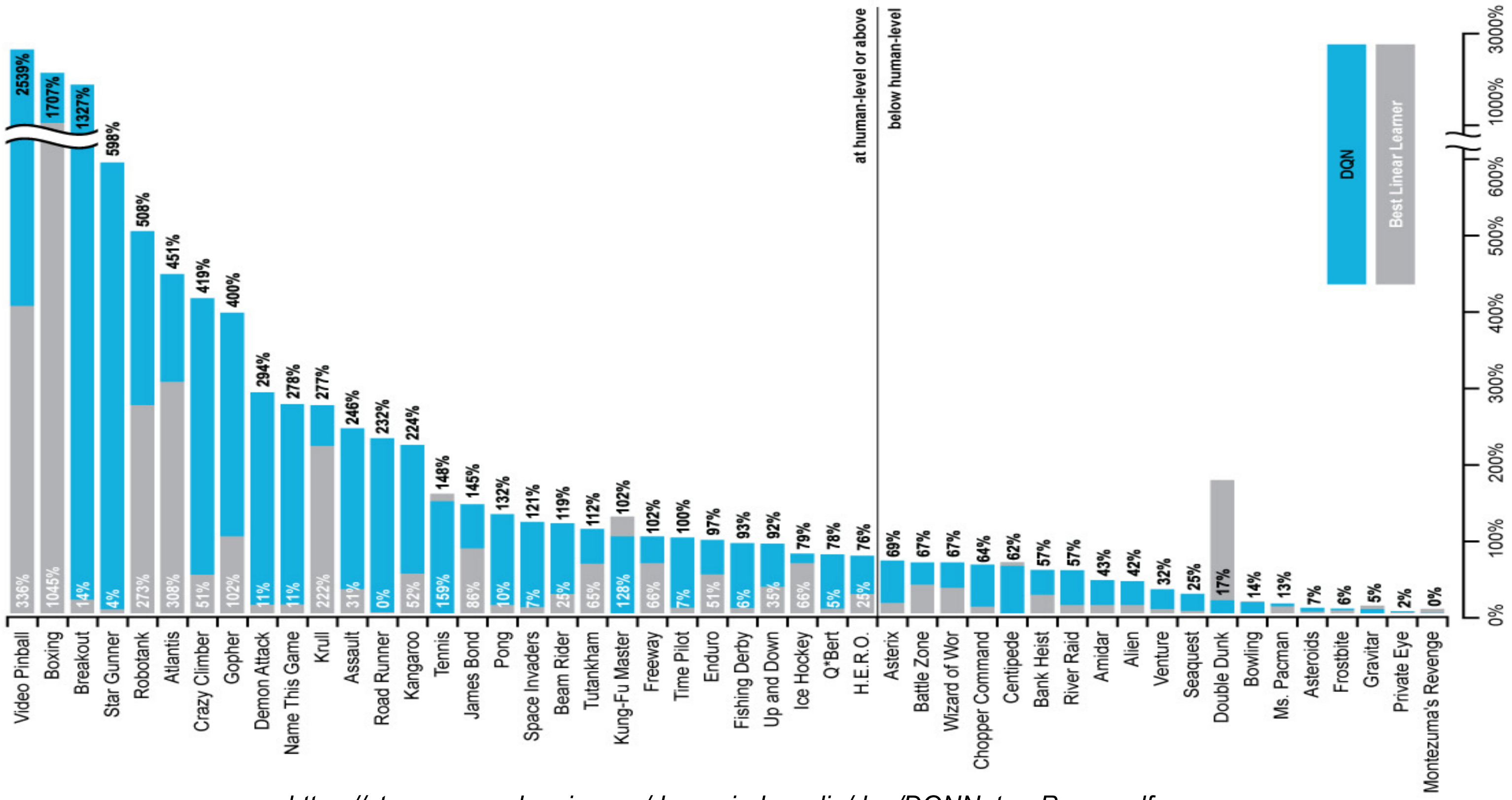
https://storage.googleapis.com/deepmind-media/dqn/DQNNaturePaper.pdf



https://www.aarondefazio.com/adefazio-rl2014.pdf

Deep Q-Networks (DQNs)

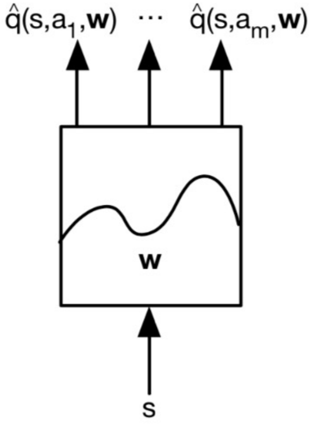
- Then a game-changing result was published in Nature:
  DQN from DeepMind (now Google DeepMind)



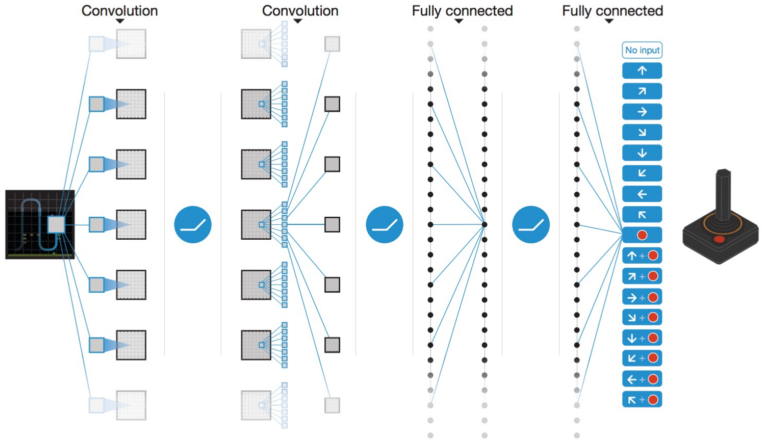
https://storage.googleapis.com/deepmind-media/dqn/DQNNaturePaper.pdf

Deep Q-Networks (DQNs)

- How does it work?
  - A convolutional neural network reads the image from the game (i.e., a framestack that uses the last $N = 4$ frames).
  - The CNN is a value function approximator for the $Q(s, a)$ function.
  - The reward is the game score.
  - The network weights are tuned using backpropagation signals of the rewards.



http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching_files/FA.pdf



https://storage.googleapis.com/deepmind-media/dqn/DQNNaturePaper.pdf

Deep Q-Networks (DQNs)

- How does it work?
  - DQNs are „Q-Learning on steroids“ (Deep NN as VFA)
  - Training possible in Tensorflow (or Pytorch, Keras, …)
  - Objective function for gradient descent:

$$L(w_i) = \mathbb{E}_{s,a,r,s' \sim D_i}[(y_i - Q(s,a,w))^2]$$

Q-Learning with NN VFA

- Approximated Q-function with NN and parameters w:

$$Q(s,a) \approx \hat{Q}(s,a;w^-)$$

- Target value:

$$y_i = r + \gamma \max_{a' \in \mathcal{A}} \hat{Q}(s',a';w^-)$$

- Updating:
  $w_{i+1} = w_i + \alpha\big[y_i - \hat{Q}(s,a;w_i)\big]\nabla_{w_i}\hat{Q}(s,a;w_i)$
  Every k steps: $w^- \leftarrow w_i$

Q-Learning with Linear VFA

- Approximated Q-function with parameters w and feature vector ϕ:

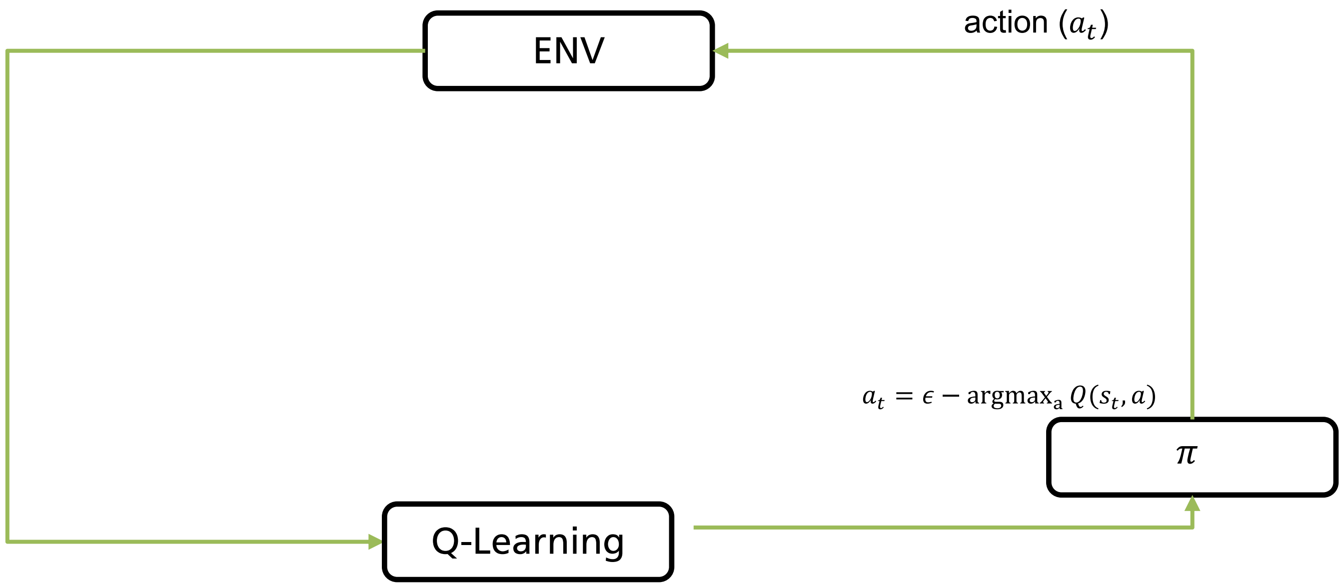
$$Q(s,a) \approx w^T\phi(s,a)$$

- Target value:

$$y_i = r + \gamma \max_{a' \in \mathcal{A}} w_i^T\phi(s',a')$$

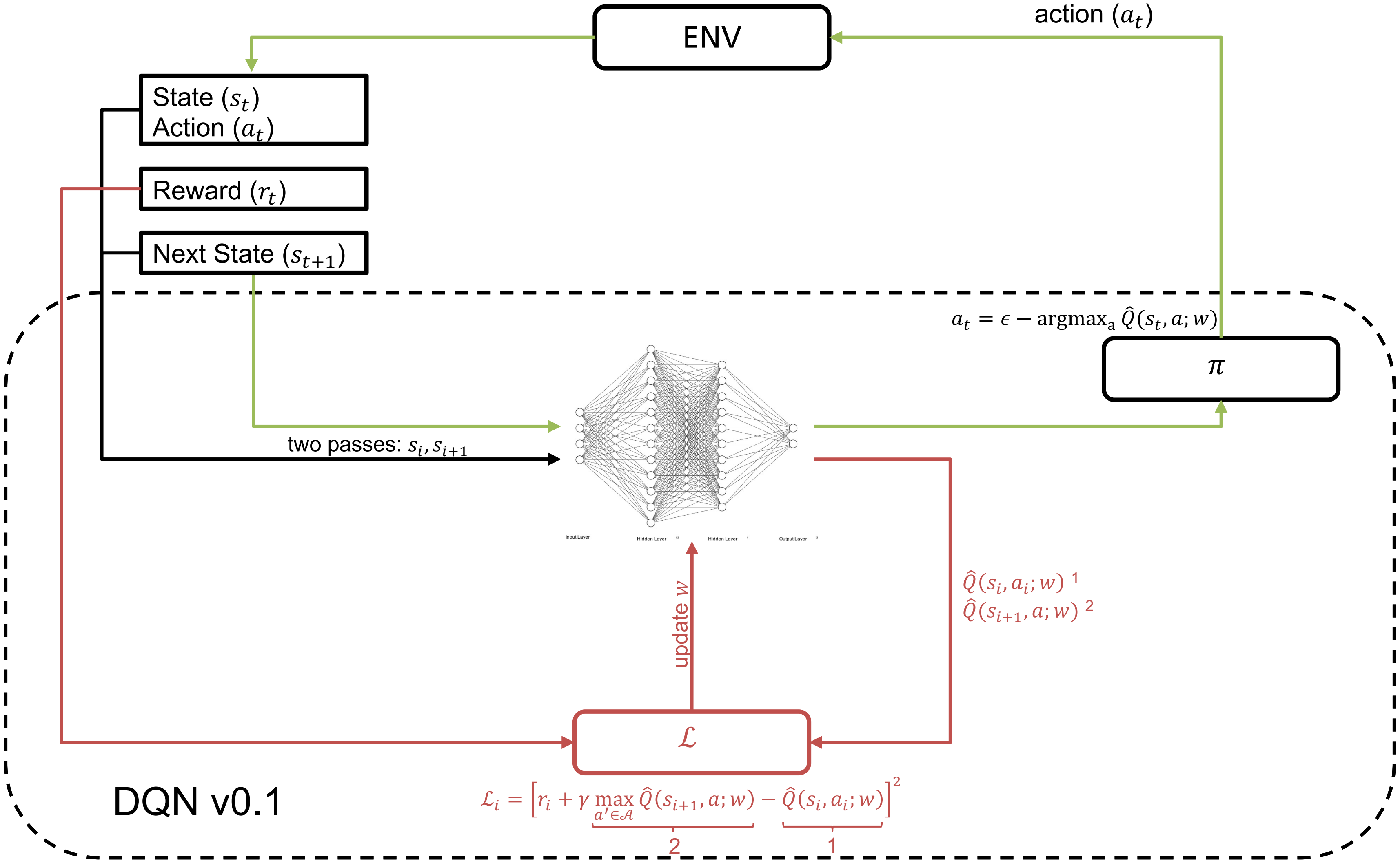
- Updating:
  $w_{i+1} = w_i + \alpha[y_i - w_i^T\phi(s,a)]\phi(s,a)$

Deep Q-Networks (DQNs)



$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_t + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right]$$

Deep Q-Networks (DQNs)



Deep Q-Networks (DQNs)

- How does it work?
- A **bag of tricks** for stabilizing learning:

  1. Experience Replay:
     - **Problem:** Experiences are correlated over time.
       → *Oscillations* and *divergence* during learning.
     - **Solution:** Random sampling of experience mini-batches from a memory.
       → Samples can be re-used to *increase data efficiency*.
       → Breaking correlations by randomization *reduces variance*.

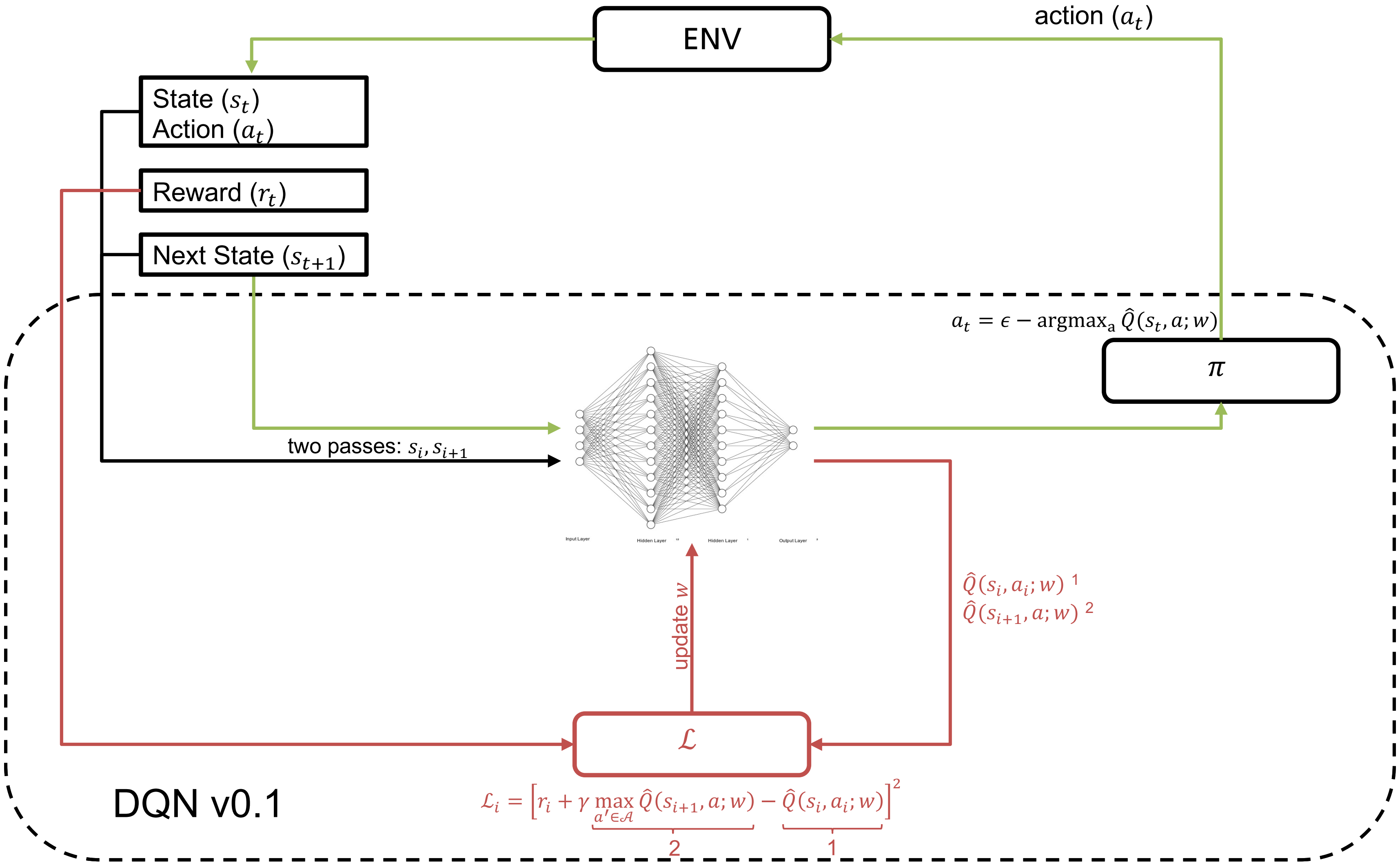
s_1, a_1, r_2, s_2
s_2, a_2, r_3, s_3
s_3, a_3, r_4, s_4
$\dots$
$s_t, a_t, r_{t+1}, s_{t+1}$

$\rightarrow s, a, r, s'$

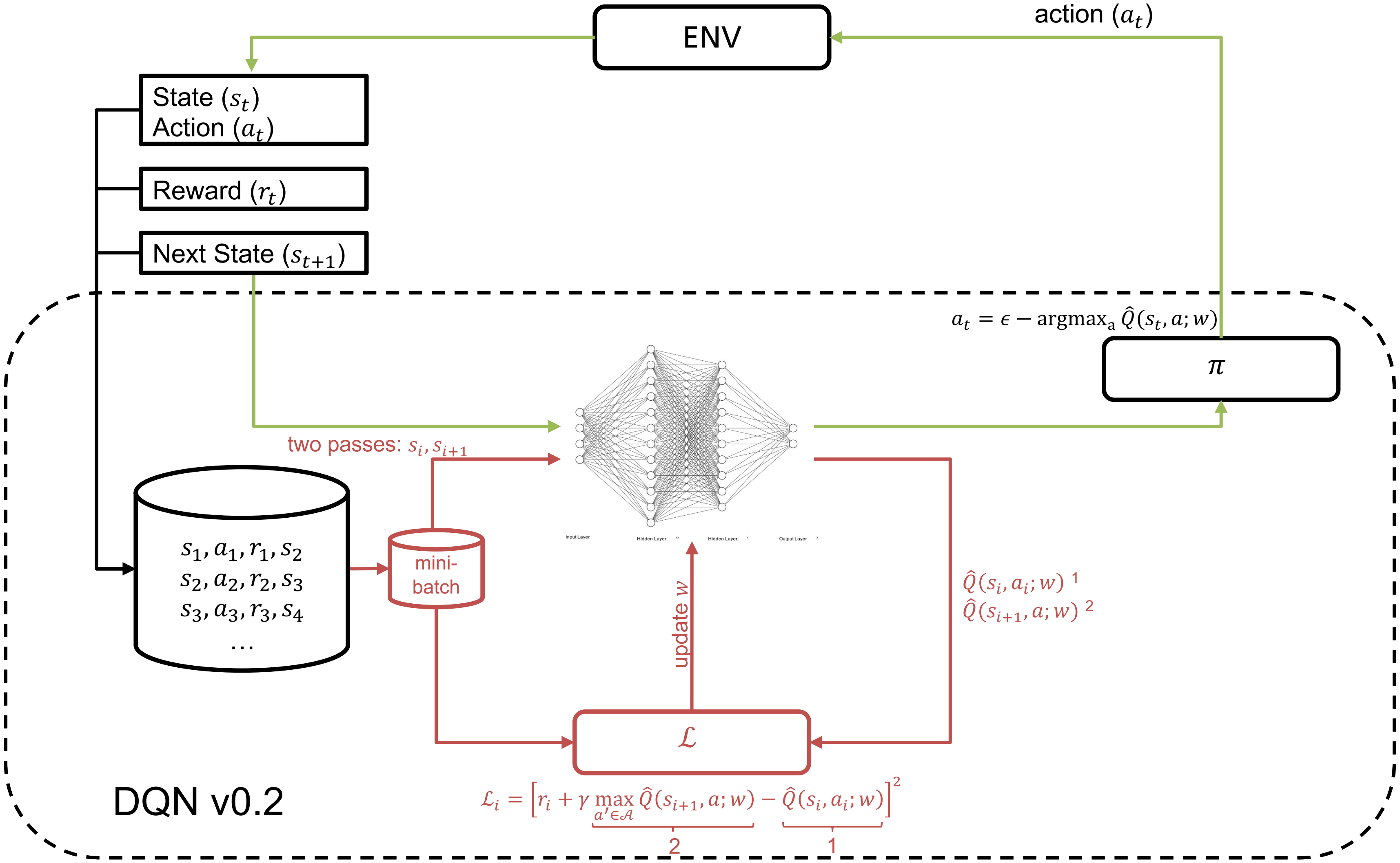
$s_t, a_t, r_{t+1}, s_{t+1} \rightarrow$

http://www0.cs.ucl.ac.uk/staff/d.silver/web/Resources_files/deep_rl.pdf

Deep Q-Networks (DQNs)



Deep Q-Networks (DQNs)



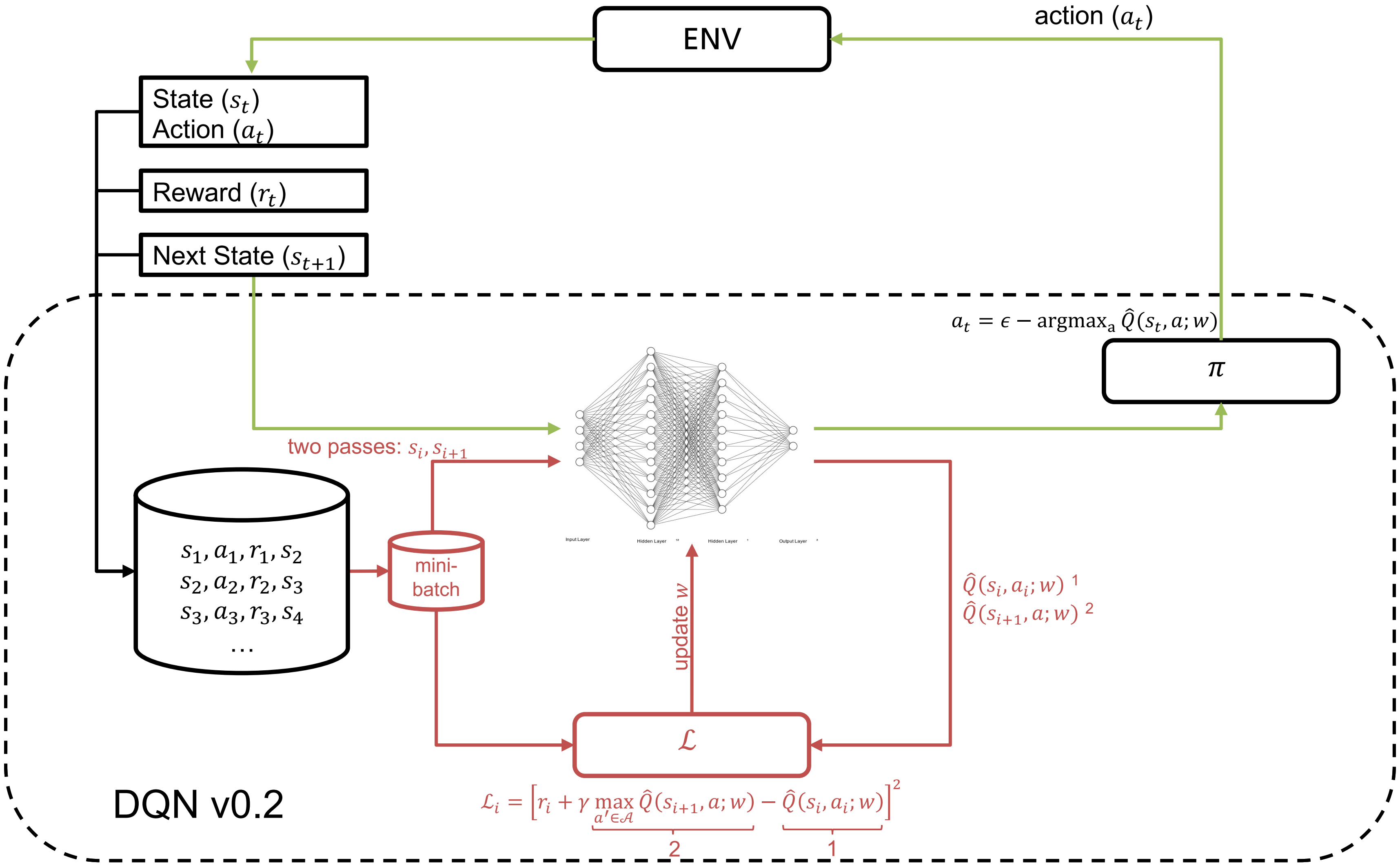
Deep Q-Networks (DQNs)

- How does it work?
- A **bag of tricks** for stabilizing learning:
  1. Experience Replay.
  2. Separate, frozen target Q-network:
     - **Problem:** Target Q-values $y_i = r + \gamma \max_{a' \in \mathcal{A}} \hat{Q}(s', a'; w_{i-1})$ change constantly.
       → *Oscillations* and *divergence* during learning.

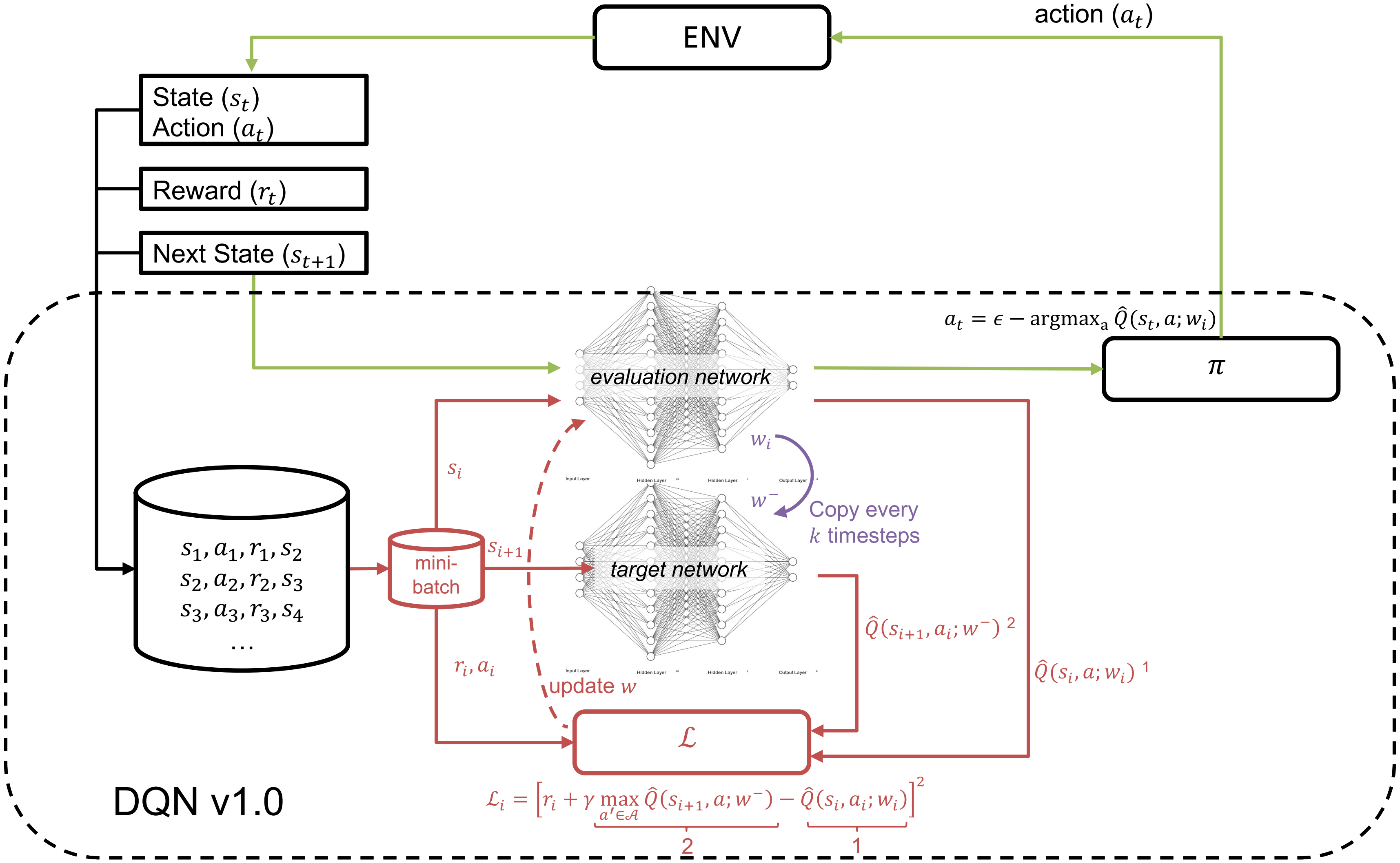
Deep Q-Networks (DQNs)

- How does it work?
- A **bag of tricks** for stabilizing learning:
  1. Experience Replay.
  2. Separate, frozen target Q-network:
     - **Solution:** Two Q-networks:
       → Frozen Target Q-network with parameters w^- predicts Q-learning targets $\hat{Q}(s', a'; w_i^-)$.
       → Dynamic Main Q-network with parameters w evaluates current Q-values $\hat{Q}(s', a'; w_{i+1})$.
       → Perform a gradient descent step (w.r.t. w) towards $\left(y_i - \hat{Q}(s, a; w_i)\right)^2$
     - Added delay breaks correlations between Q-network and target.
     - *Avoids oscillations* by having fixed targets.
       (Note: We periodically update the target Q-network by copying the weights $w^- \leftarrow w_i$.)
     - *Reduces* chance of *divergence*.

Deep Q-Networks (DQNs)



Deep Q-Networks (DQNs)



Deep Q-Networks (DQNs)

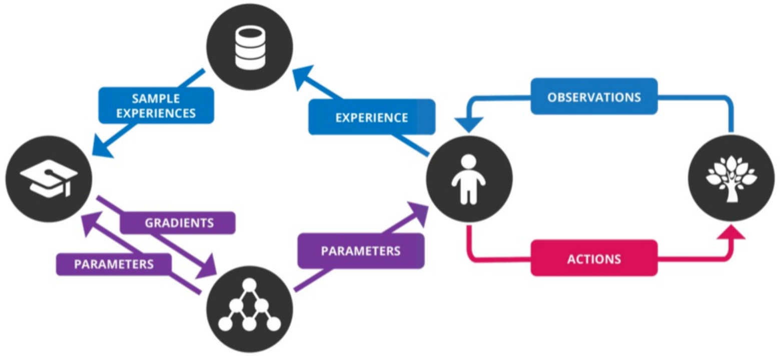
- How does it work?
- A **bag of tricks** for stabilizing learning:
  1. Experience Replay.
  2. Separate, frozen target Q-network.
  3. Apply reward clipping:
     - **Problem:** Large rewards result in large variances in Q-values.
     - Different games have different reward values.
     - *Oscillations* and *divergence* during learing.

     - **Solution:** Clip the rewards (and loss terms) to a range $[-1.0, 1.0]$.
     - Avoids *oscillations* by normalizing rewards when training for multiple games.
     - Prevents Q-values from becoming too large.

Deep Q-Networks (DQNs)

- How does it work?

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



https://sites.google.com/view/deep-rl-bootcamp/lectures

Deep Q-Networks (DQNs)

- **Extension #1: Double DQN (DDQN)**
- **Problem:** Upward positive bias (overestimation of Q-values) in targets:

$$y_i = r + \gamma \max_{a' \in \mathcal{A}} \hat{Q}(s', a'; \vartheta_i)$$

- DDQN splits action selection and action evaluation:

$$y_i = r + \gamma \hat{Q}\left(s', \underset{a' \in \mathcal{A}}{argmax}\, \hat{Q}(s', a'; \theta_i); \vartheta_i\right)$$

  - Use estimations from main Q-network to select actions
  - Use estimations from target Q-network to evaluate actions

Deep Q-Networks (DQNs)

- **Extension #1: Double DQN (DDQN)**
- DDQN splits action selection and action evaluation:

$$y_i = r + \gamma \hat{Q}\left(s', \underset{a' \in \mathcal{A}}{\operatorname{argmax}} \hat{Q}(s', a'; \theta_i); \vartheta_i\right)$$

  - Use estimations from main Q-network to select actions
  - Use estimations from target Q-network to evaluate actions

Algorithm 1 Double Q-learning

1: Initialize Q^A,Q^B,s
2: **repeat**
3: Choose a, based on $Q^A(s,\cdot)$ and $Q^B(s,\cdot)$, observe r, s'
4: Choose (e.g. random) either UPDATE(A) or UPDATE(B)
5: **if** UPDATE(A) **then**
6: Define $a^* = \arg\max_a Q^A(s', a)$
7: $Q^A(s,a) \leftarrow Q^A(s,a) + \alpha(s,a)\left(r + \gamma Q^B(s', a^*) - Q^A(s,a)\right)$
8: **else if** UPDATE(B) **then**
9: Define $b^* = \arg\max_a Q^B(s', a)$
10: $Q^B(s,a) \leftarrow Q^B(s,a) + \alpha(s,a)(r + \gamma Q^A(s', b^*) - Q^B(s,a))$
11: **end if**
12: $s \leftarrow s'$
13: **until** end

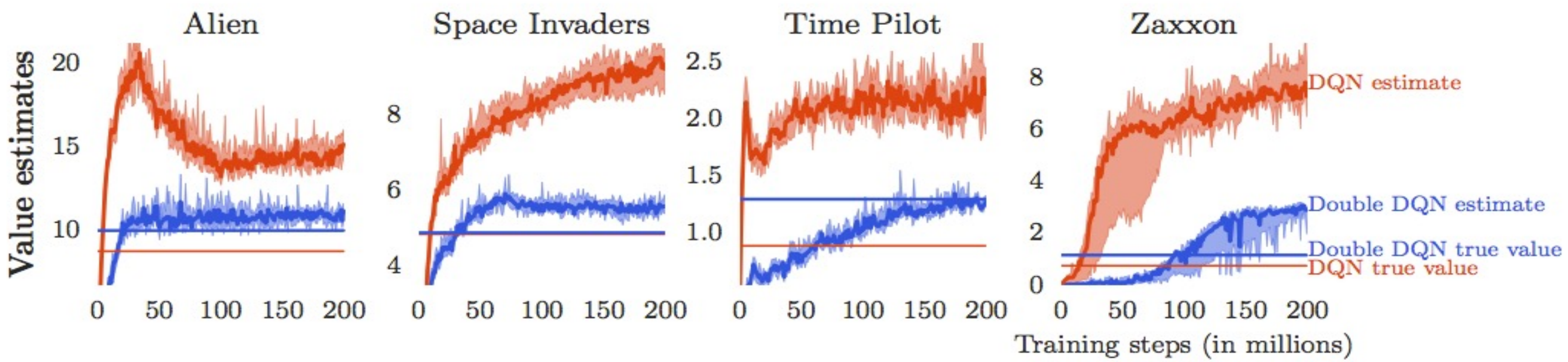
Hasselt, H. V. (2010). Double Q-learning. In *Advances in Neural Information Processing Systems* (pp. 2613-2621).

Deep Q-Networks (DQNs)

- **Extension #1: Double DQN (DDQN)**
- DDQN splits action selection and action evaluation:

$$y_i = r + \gamma \hat{Q}(s', \underset{a' \in \mathcal{A}}{\operatorname{argmax}} \hat{Q}(s', a'; \theta_i); \vartheta_i)$$

  - Use estimations from main Q-network to select actions
  - Use estimations from target Q-network to evaluate actions



Hasselt, H. V. (2015). Deep Reinforcement Learning with Double Q-learning. In *AAAI* (pp. 2094-2100).

Deep Q-Networks (DQNs)

- **Extension #2: Prioritized Experience Replay**
- Some experiences retain more information for learning than others
- **Problem:** Experience Replay Sampling is uniform sampling
- **Prioritized Experience Replay** samples mini-batches of based on their absolute Bellman error e:

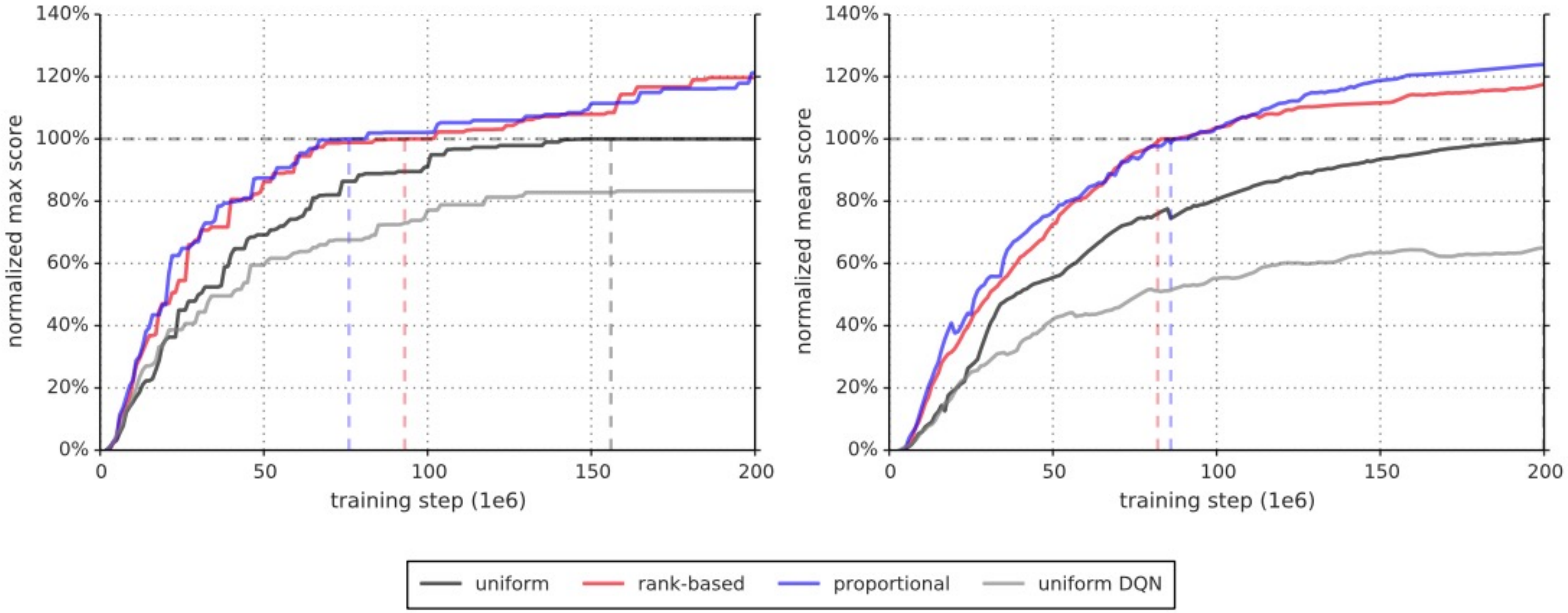
$$\delta = r + \gamma \max_{a' \in \mathcal{A}} Q(s', a') - Q(s, a)$$
$$e = |\delta|$$

- Using DDQN notation:

$$y_i = r + \gamma \hat{Q}\left(s', \underset{a' \in \mathcal{A}}{argmax}\, \hat{Q}(s', a'; \theta_i)\, ; \vartheta_i\right)$$
$$\delta = y_i - Q(s, a; \theta_i)$$
$$e = |\delta|$$

Deep Q-Networks (DQNs)

- **Extension #2: Prioritized Experience Replay**
- Some experiences retain more information for learning than others
- **Problem:** Experience Replay Sampling is uniform sampling
- **Prioritized Experience Replay** leads to much faster learning



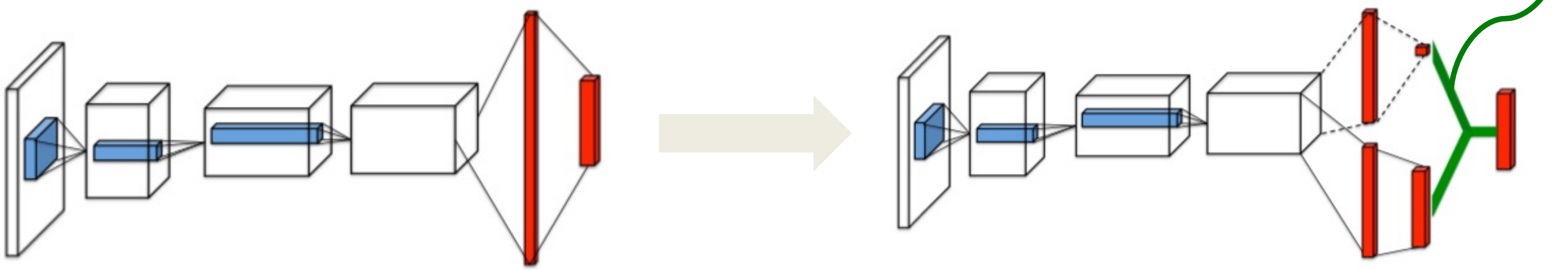
https://arxiv.org/pdf/1511.05952.pdf

Deep Q-Networks (DQNs)

- **Extension #3: Dueling Architectures**
- Split Q-network into two channels:
  - Action-independent value function $V(s;\mathbf{v})$
  - Action-dependent advantage function $A(s,a;\mathbf{w})$

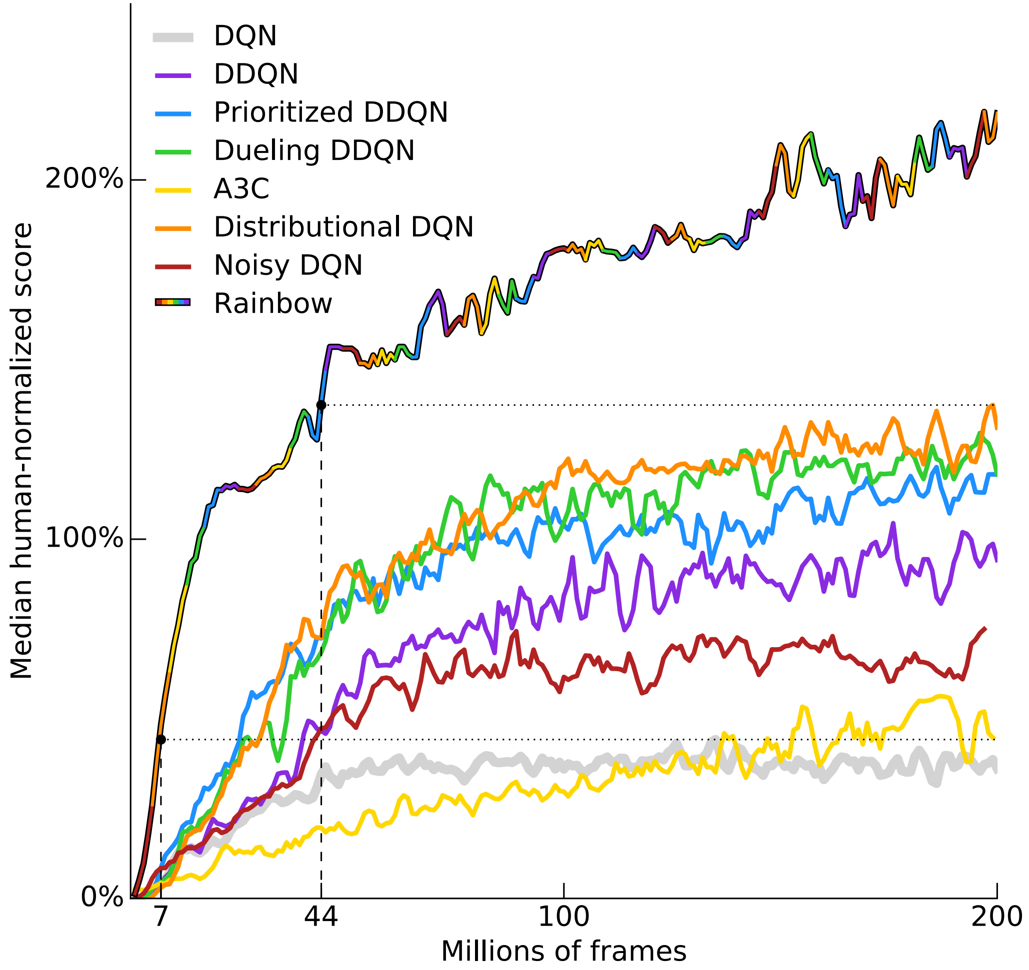
  → $Q(s,a) = V(s;\mathbf{v}) + A(s,a;\mathbf{w})$

$$Q(s,a;\theta,\alpha,\beta) = V(s;\theta,\beta) + \left(A(s,a;\theta,\alpha) - \frac{1}{|\mathcal{A}|}\sum_{a'} A(s,a';\theta,\alpha)\right)$$



Wang et al.: *Dueling Network Architectures for Deep Reinforcement Learning*

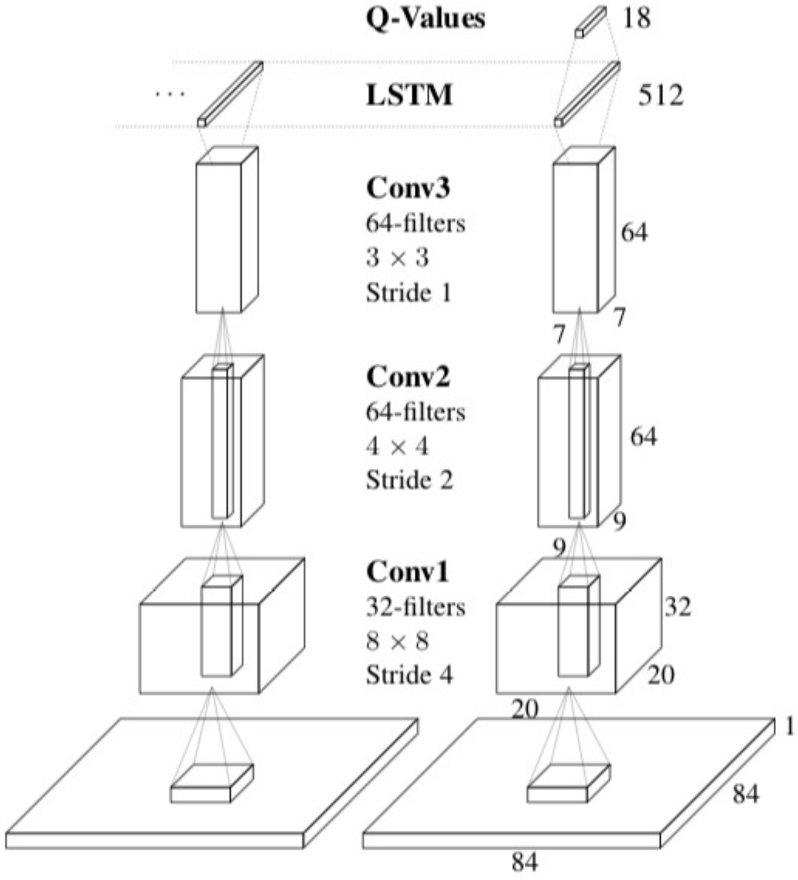
Deep Q-Networks (DQNs)



Hessel et al.: Rainbow: Combining Improvements in Deep Reinforcement Learning

Deep Q-Networks (DQNs)

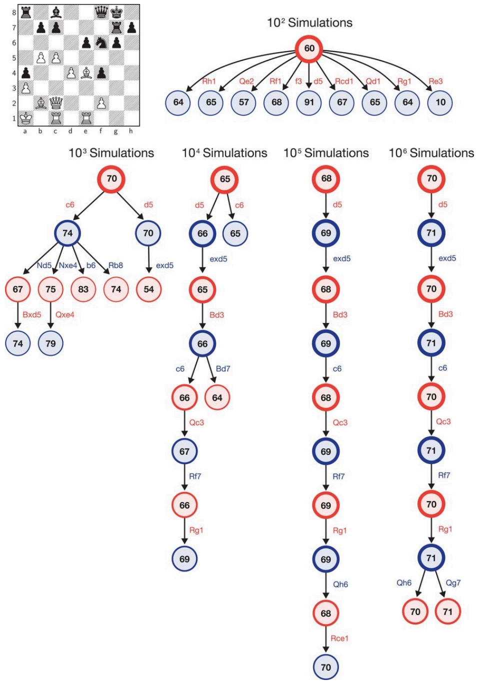
- **Extension: Deep Recurrent Q-Learning Networks (DRQNs)**
- DQN in Atari needed to process more frames (4) in order to get an estimate of hidden state (e.g., ball velocity in pong)
- Many real-world problems have partially observable states (POMDPs)
- So instead of augmenting states in the POMDP case, why don't we use an RNN (LSTM) instead?
- Caution: use only when you have a true POMDP problem as it adds significant complexity during training. If you are not sure, try augmenting states first!

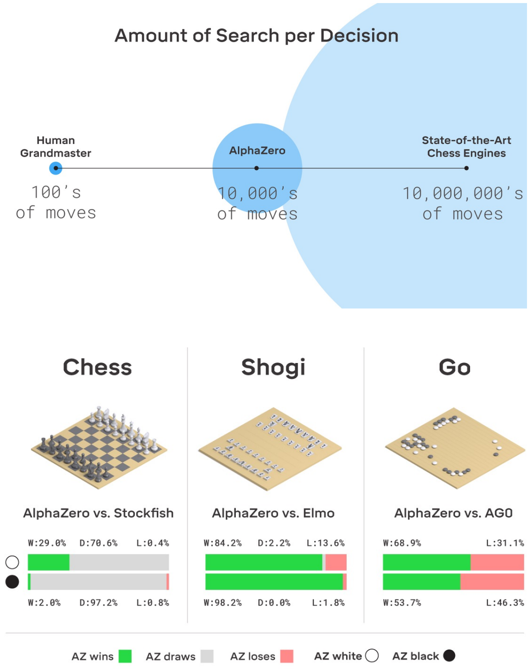


https://arxiv.org/pdf/1507.06527.pdf

Deep Q-Networks (DQNs)

- **Beyond just an extension: AlphaZero**





https://deepmind.com/blog/alphazero-shedding-new-light-grand-games-chess-shogi-and-go/

References

- Watkins, C. J. C. H. (1989). Learning from delayed rewards(Doctoral dissertation, King's College, Cambridge): https://link.springer.com/content/pdf/10.1007/BF00992698.pdf
- Lin, L. J. (1993). Reinforcement learning for robots using neural networks (No. CMU-CS-93-103). Carnegie-Mellon Univ Pittsburgh PA School of Computer Science: http://www.dtic.mil/dtic/tr/fulltext/u2/a261434.pdf
- Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., ... & Petersen, S. (2015). Human-level control through deep reinforcement learning. Nature, 518(7540), 529: http://web.stanford.edu/class/psych209/Readings/MnihEtAlHassibis15NatureControlDeepRL.pdf
- Van Hasselt, H., Guez, A., & Silver, D. (2016, February). Deep Reinforcement Learning with Double Q-Learning. In AAAI (Vol. 2, p. 5): http://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/download/12389/11847
- Wang, Z., Schaul, T., Hessel, M., Van Hasselt, H., Lanctot, M., & De Freitas, N. (2015). Dueling network architectures for deep reinforcement learning. arXiv preprint arXiv:1511.06581: https://arxiv.org/abs/1511.06581
- Hausknecht, M., & Stone, P. (2015). Deep recurrent q-learning for partially observable mdps. CoRR, abs/1507.06527, 7(1): http://www.aaai.org/ocs/index.php/FSS/FSS15/paper/download/11673/11503
- Van Hasselt, H., Doron, Y., Strub, F., Hessel, M., Sonnerat, N., & Modayil, J. (2018). Deep Reinforcement Learning and the Deadly Triad. arXiv preprint arXiv:1812.02648.: https://arxiv.org/abs/1812.02648