
# Deep Reinforcement Learning

## Beyond DQN

Julien Vitay
Professur für Künstliche Intelligenz - Fakultät für Informatik

# 1 - Distributional learning : Categorical DQN

---

## A Distributional Perspective on Reinforcement Learning

---

**Marc G. Bellemare** [*1] **Will Dabney** [*1] **Rémi Munos** [1]

# Distributional learning

- Until now, we have only cared about the **expectation** of the returns, i.e. their mean value:

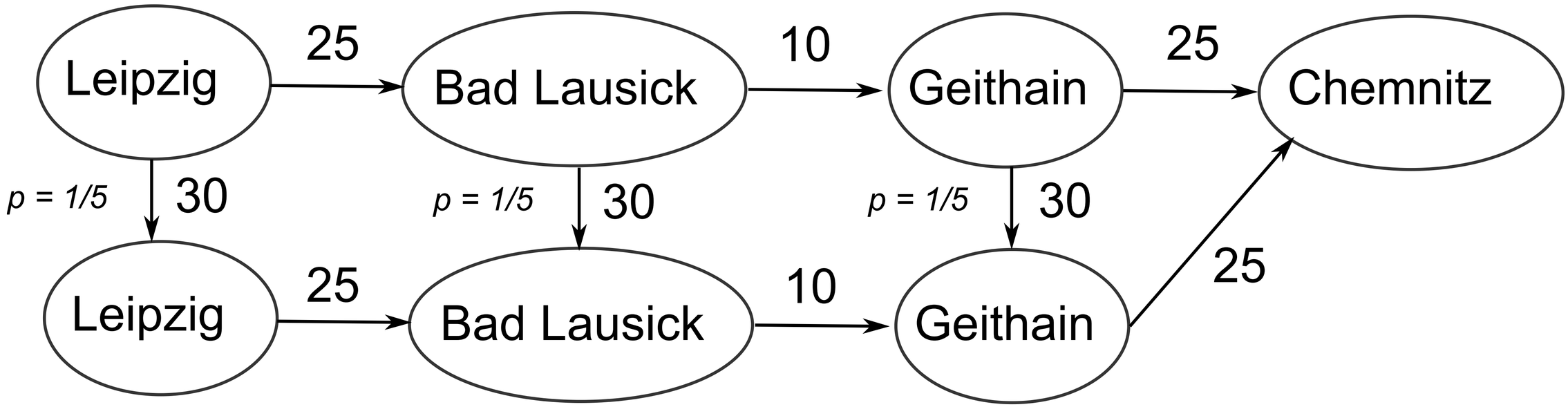
$$V^\pi(s) = \mathbb{E}_\pi[R_t|s_t = s]$$

$$Q^\pi(s,a) = \mathbb{E}_\pi[R_t|s_t = s, a_t = a]$$

- We select actions with the highest **expected return**, which makes sense **on the long term**.
- Suppose we have two actions $a_1$ and $a_2$, which provide different returns with the same probability:
  - $R(a_1) = \{100, 200\}$
  - $R(a_2) = \{-100, 400\}$
- Their Q-value is the same: $Q(a_1) = Q(a_2) = 150$, so if you play them an **infinity** of times, they are both optimal.
- But suppose that, after learning, you can only try a **single** action. Which one do you chose?
- RL does not distinguish safe from risky actions.

# Distributional learning

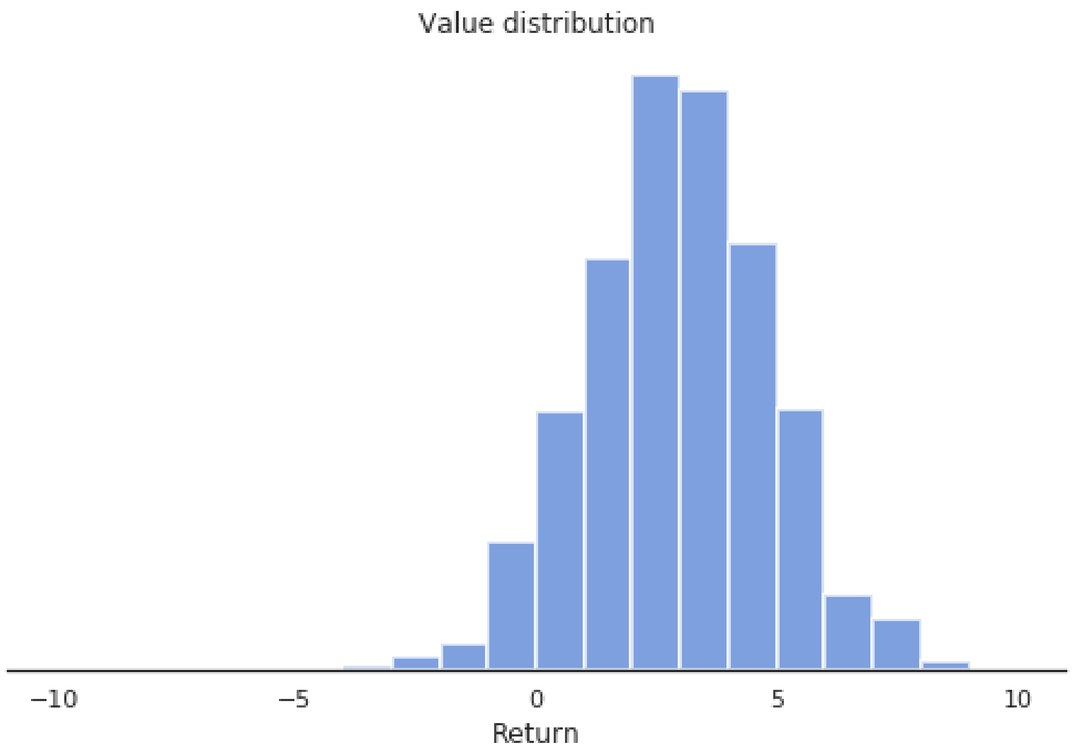
- The trip by train from Leipzig to Chemnitz takes 1 hour if everything goes well.
- Once a week on average, the train will get stuck on the way for 30 minutes.
- The expected duration of the trip is 1h + 1/5*30 = 1h06.
- But in practice it takes either 1h or 1h30, never 1h06.
- If driving by car always takes 1h15, it might be worth it if you have an urgent appointment that day.



# Distributional learning

- The idea of **distributional RL** is to learn the **distribution of returns** $\mathcal{Z}^\pi$ directly instead of its expectation:

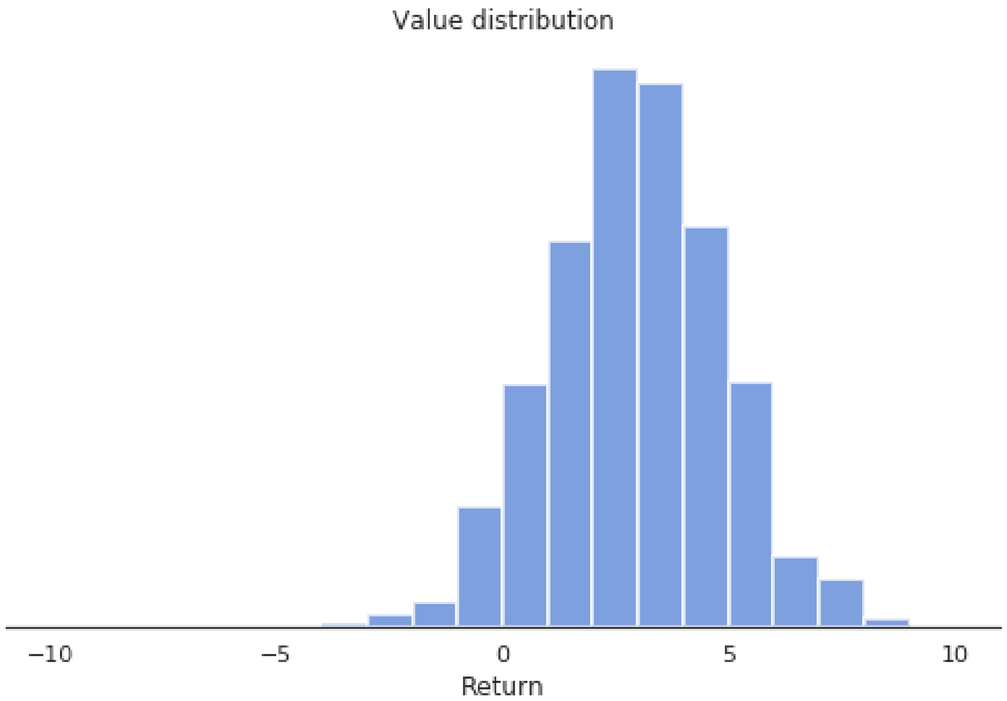
$$R_t \sim \mathcal{Z}^\pi(s_t, a_t)$$




- Note that we can always obtain the Q-values back:

$$Q^\pi(s, a) = \mathbb{E}_\pi[\mathcal{Z}^\pi(s, a)]$$

# Categorical DQN

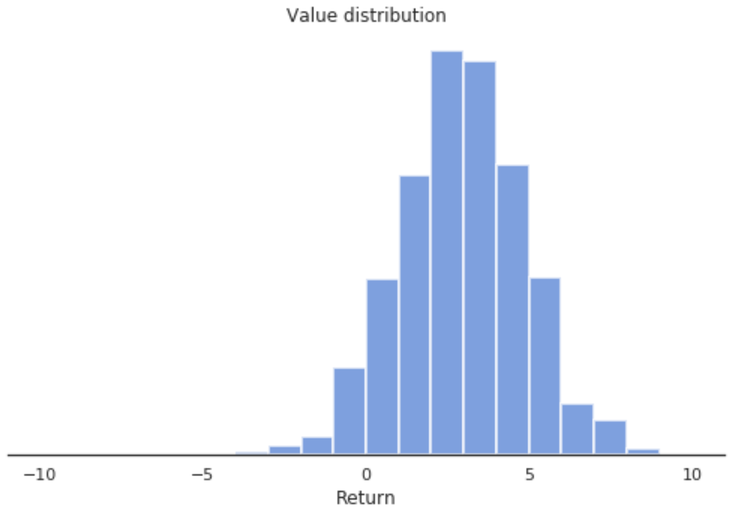



- In categorical DQN (Bellemare et al., 2017), we model the distribution of returns as a **discrete probability distribution**.
  - **categorical** or **multinouilli** distribution.
- We first need to identify the minimum and maximum returns $R_{\min}$ and $R_{\max}$ possible in the problem.
- We then split the range $\left[R_{\min}, R_{\max}\right]$ in $n$ **discrete bins** centered on the atoms $\{z_i\}_{i=1}^{n}$.

- The probability that the return obtained the action $(s, a)$ lies in the bin of the atom $z_i$ is noted $p_i(s, a)$.
- It can be approximated by a neural network $F$ with parameters $\theta$, using a **softmax output layer**:

$$p_i(s, a; \theta) = \frac{\exp F_i(s, a; \theta)}{\sum_{j=1}^{n} \exp F_j(s, a; \theta)}$$

# Categorical DQN




- The $n$ probabilities $\{p_i(s, a; \theta)\}_{i=1}^{n}$ completely define the parameterized distribution $\mathcal{Z}_\theta(s, a)$.

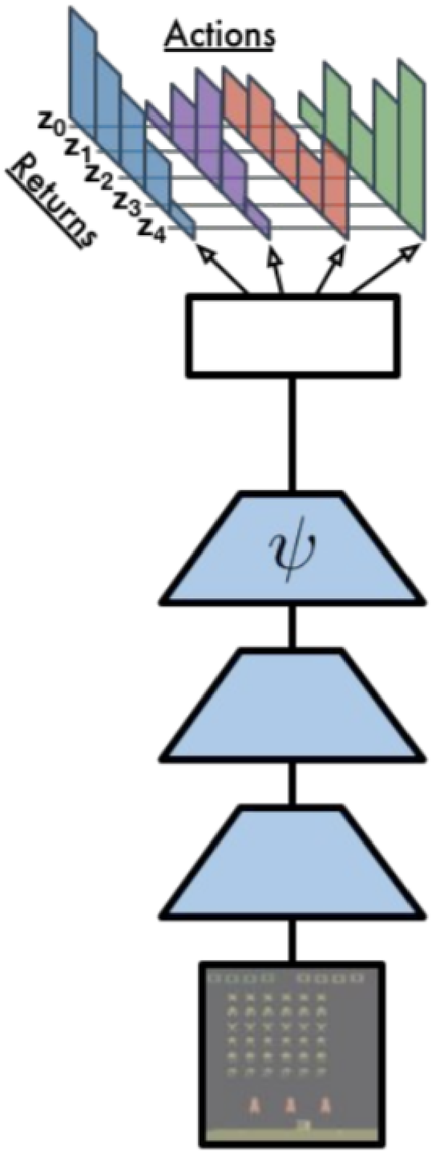
$$\mathcal{Z}_\theta(s, a) = \sum_a p_i(s, a; \theta)\, \delta_{z_i}$$

where $\delta_{z_i}$ is a Dirac distribution centered on the atom $z_i$.

- The Q-value of an action can be obtained by:

$$Q_\theta(s, a) = \mathbb{E}[\mathcal{Z}_\theta(s, a)] = \sum_{i=1}^{n} p_i(s, a; \theta)\, z_i$$

# Categorical DQN




- The only thing we need is a neural network $\theta$ returning for each action $a$ in the state $s$ a discrete probability distribution $\mathcal{Z}_\theta(s, a)$ instead of a single Q-value $Q_\theta(s, a)$.
- The NN uses a **softmax activation function** for each action.
- Action selection is similar to DQN: we first compute the $Q_\theta(s, a)$ and apply greedy / $\epsilon$-greedy / softmax over the actions.

$$Q_\theta(s, a) = \sum_{i=1}^{n} p_i(s, a; \theta)\, z_i$$

- The number $n$ of atoms for each action should be big enough to represent the range of returns.
- A number that works well with Atari games is $n = 51$:
  - Categorical DQN is often noted **C51**.

Source:
https://physai.sciencesconf.org/data/pages/distributional_RL_Remi_Munos.pdf

## Categorical DQN

- How do we learn the distribution of returns $\mathcal{Z}_\theta(s, a)$ of parameters $\{p_i(s, a; \theta)\}_{i=1}^n$?
- In Q-learning, we minimize the mse between the prediction $Q_\theta(s, a)$ and the **target**:

$$\mathcal{T} Q_\theta(s, a) = r + \gamma Q_\theta(s', a')$$

where $\mathcal{T}$ is the Bellman operator.

$$\min_\theta (\mathcal{T} Q_\theta(s, a) - Q_\theta(s, a))^2$$

- We do the same here:
  - we apply the **Bellman operator** on the distribution $\mathcal{Z}_\theta(s, a)$.

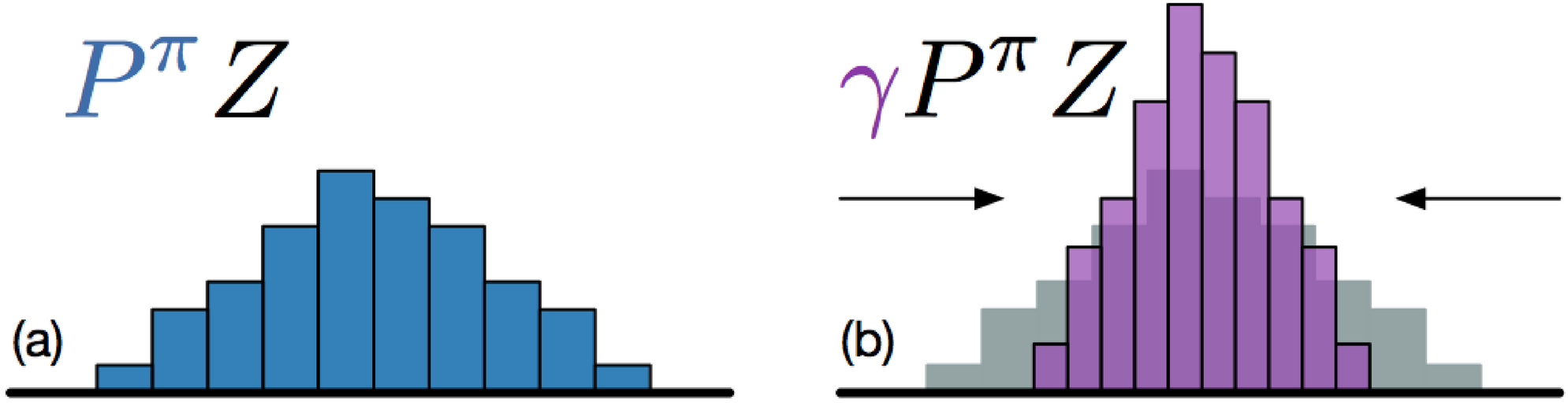
$$\mathcal{T} \mathcal{Z}_\theta(s, a) = r(s, a) + \gamma \mathcal{Z}_\theta(s', a')$$

  - we then minimize the statistical "distance" between the distributions $\mathcal{Z}_\theta(s, a)$ and $\mathcal{T} \mathcal{Z}_\theta(s, a)$.

$$\min_\theta \mathrm{KL}(\mathcal{T} \mathcal{Z}_\theta(s, a) || \mathcal{Z}_\theta(s, a))$$

## Categorical DQN

- Let's note $P^\pi \mathcal{Z}$ the return distribution of the greedy action in the next state $\mathcal{Z}_\theta(s', a')$.
- Multiplying the returns by the discount factor $\gamma < 1$ **shrinks** the return distribution (its support gets smaller).
- The **atoms** $z_i$ of $\mathcal{Z}_\theta(s', a')$ now have the position $\gamma\, z_i$, but the probabilities stay the same.



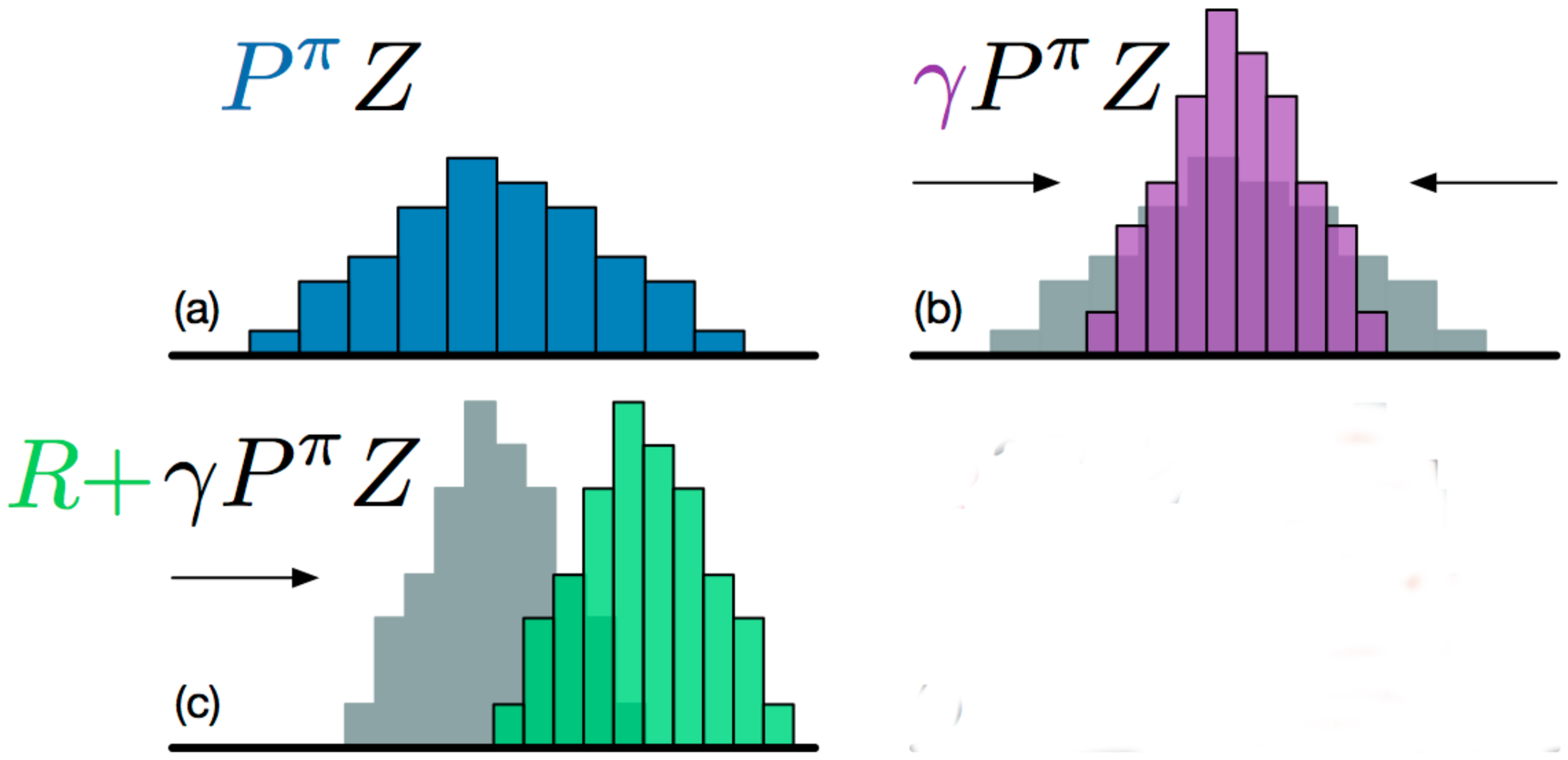

Source: https://physai.sciencesconf.org/data/pages/distributional_RL_Remi_Munos.pdf

# Categorical DQN

- Adding a reward $r$ **translates** the distribution. The new position of the atoms is:

$$z_i' = r + \gamma\, z_i$$

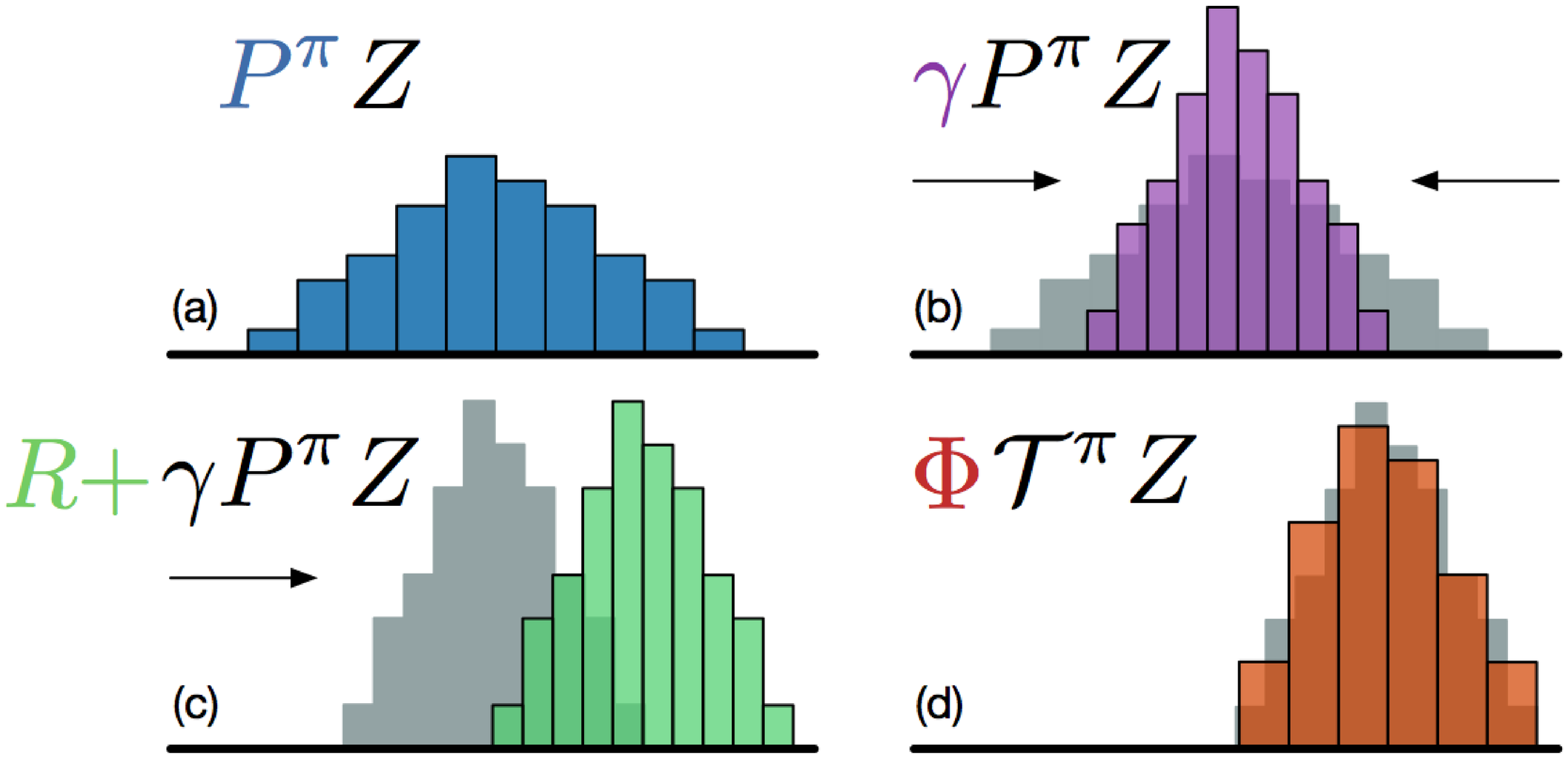
- The corresponding probabilities have not changed.




Source: https://physai.sciencesconf.org/data/pages/distributional_RL_Remi_Munos.pdf

## Categorical DQN

- But now we have a problem: the atoms $z_i'$ of $\mathcal{T}\ \mathcal{Z}_\theta(s,a)$ do not match with the atoms $z_i$ of $\mathcal{Z}_\theta(s,a)$.
- We need to **interpolate** the target distribution to compare it with the predicted distribution.

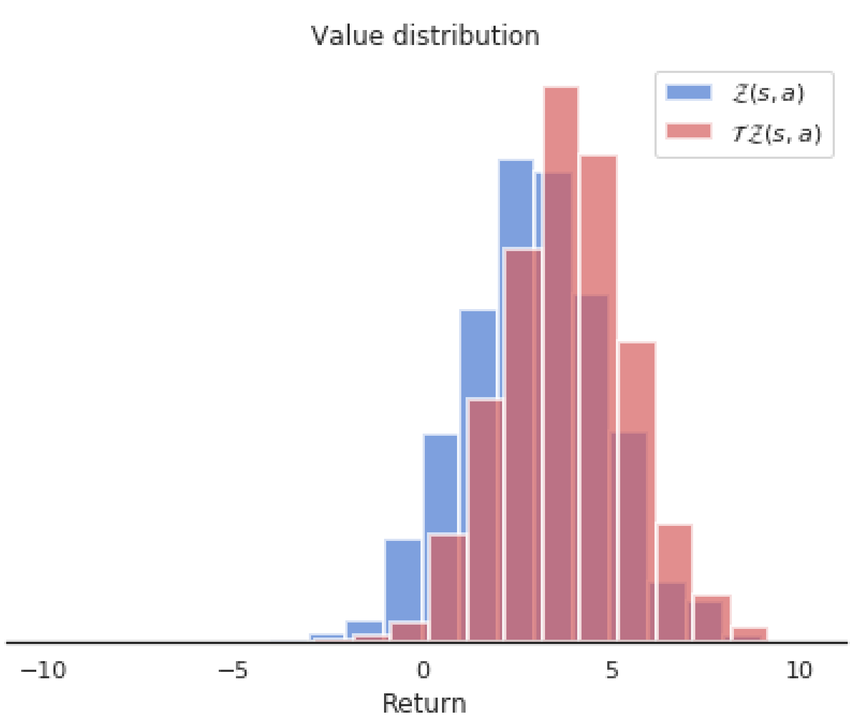


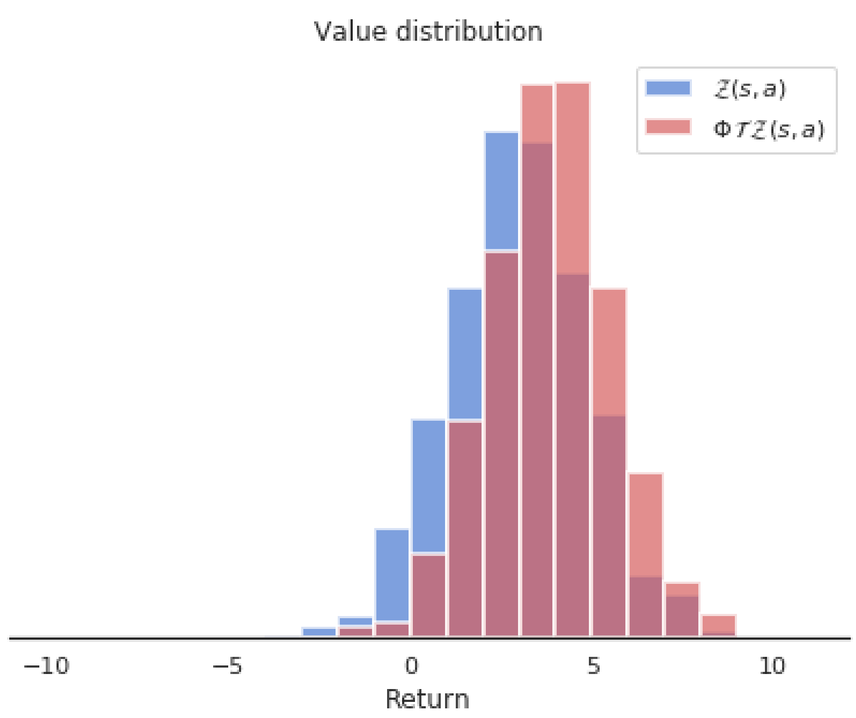

Source: https://physai.sciencesconf.org/data/pages/distributional_RL_Remi_Munos.pdf

# Categorical DQN

- We need to apply a **projection** $\Phi$ so that the bins of $\mathcal{T}\,\mathcal{Z}_\theta(s,a)$ are the same as the ones of $\mathcal{Z}_\theta(s,a)$.
- The formula sounds complicated, but it is basically a linear interpolation:

$$(\Phi\,\mathcal{T}\,\mathcal{Z}_\theta(s,a))_i = \sum_{j=1}^{n} \Big[1 - \frac{\big|[\mathcal{T}\,z_j]_{R_{\min}}^{R_{\max}} - z_i\big|}{\Delta z}\Big]_0^1 p_j(s',a';\theta)$$



# Categorical DQN


- We now have two distributions $\mathcal{Z}_\theta(s,a)$ and $\Phi\,\mathcal{T}\,\mathcal{Z}_\theta(s,a)$ sharing the same support.
- We now want to have the prediction $\mathcal{Z}_\theta(s,a)$ close from the target $\Phi\,\mathcal{T}\,\mathcal{Z}_\theta(s,a)$.
- These are probability distributions, not numbers, so we cannot use the mse.
- We instead minimize the **Kullback-Leibler (KL) divergence** between the two distributions.

# Kullback-Leibler (KL) divergence

- Let's consider a parameterized discrete distribution $X_\theta$ and a discrete target distribution $T$.
- The KL divergence between the two distributions is:

$$\mathrm{KL}(T||X_\theta) = \mathbb{E}_{t\sim T}[-\log \frac{X_\theta}{T}]$$

- It can be rewritten as the sum of the **cross-entropy** and the entropy of $T$:

$$\mathrm{KL}(X_\theta||T) = \mathbb{E}_{t\sim T}[-\log X_\theta + \log T] = H(X_\theta, T) - H(T)$$

- As $T$ does not depend on $\theta$, the gradient of the KL divergence w.r.t to $\theta$ is the same as the gradient of the cross-entropy.

$$\nabla_\theta \,\mathrm{KL}(X_\theta||T) = \mathbb{E}_{t\sim T}[-\nabla_\theta \log X_\theta]$$

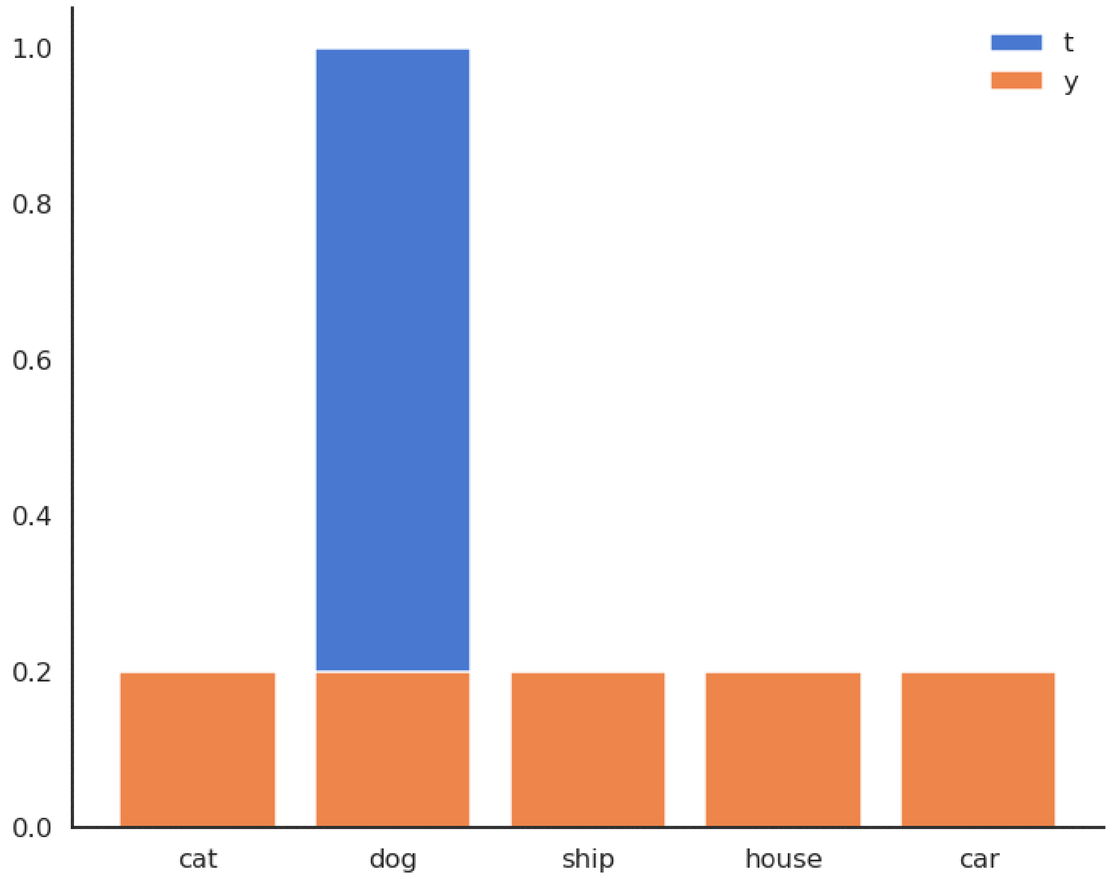
- **Minimizing the KL divergence is the same as minimizing the cross-entropy.**
- Neural networks with a softmax output layer and the cross-entropy loss function can do that.

# Cross-entropy

- In supervised learning, the targets $\mathbf{t}$ are fixed **one-hot encoded vectors**.

$$\mathcal{L}(\theta) = \mathbb{E}_{\mathcal{D}}[-\mathbf{t} \log \mathbf{y}]$$

- But it could be any target distribution, as long as $\mathbf{t}$ and $\mathbf{y}$ share the same support.



# Reminder: DQN

- Initialize value network $Q_\theta$ and target network $Q_{\theta'}$.
- Initialize experience replay memory $\mathcal{D}$ of maximal size $N$.
- for $t \in [0, T_{\text{total}}]$:
  - Select an action $a_t$ based on $Q_\theta(s_t, a)$, observe $s_{t+1}$ and $r_{t+1}$.
  - Store $(s_t, a_t, r_{t+1}, s_{t+1})$ in the experience replay memory.
  - Every $T_{\text{train}}$ steps:
    - Sample a minibatch $\mathcal{D}_s$ randomly from $\mathcal{D}$.
    - For each transition $(s_k, a_k, r_k, s'_k)$ in the minibatch:
      - Compute the target value $t_k = r_k + \gamma \max_{a'} Q_{\theta'}(s'_k, a')$ using the target network.
    - Update the value network $Q_\theta$ on $\mathcal{D}_s$ to minimize:

$$\mathcal{L}(\theta) = \mathbb{E}_{\mathcal{D}_s}\left[(t_k - Q_\theta(s_k, a_k))^2\right]$$

  - Every $T_{\text{target}}$ steps:
    - Update target network: $\theta' \leftarrow \theta$.

# Categorical DQN

- Initialize distributional value network $Z_\theta$ and target network $Z_{\theta'}$, experience replay memory $\mathcal{D}$.
- Every $T_{\text{train}}$ steps:
  - Sample a minibatch $\mathcal{D}_s$ randomly from $\mathcal{D}$.
  - For each transition $(s_k, a_k, r_k, s'_k)$ in the minibatch:
    - Select the greedy action in the next state using the target network:

$$a'_k = \text{argmax}_a \, \mathbb{E}[Z_{\theta'}(s'_k, a)]$$

    - Apply the Bellman operator on the distribution of the next greedy action:

$$TZ_k = r_k + \gamma \, Z_{\theta'}(s'_k, a'_k)$$

    - Project this distribution to the support of $Z_\theta(s_k, a_k)$.

$$\mathbf{t}_k = \text{Projection}(TZ_k, Z_\theta(s_k, a_k))$$

  - Update the value network $Q_\theta$ on $\mathcal{D}_s$ to minimize the cross-entropy:

$$\mathcal{L}(\theta) = \mathbb{E}_{\mathcal{D}_s}[-\mathbf{t}_k \, \log Z_\theta(s_k, a_k)]$$

## Categorical DQN

**Algorithm 1** Categorical Algorithm

**input** A transition $x_t, a_t, r_t, x_{t+1}, \gamma_t \in [0, 1]$

$Q(x_{t+1}, a) := \sum_i z_i p_i(x_{t+1}, a)$

$a^* \leftarrow \arg\max_a Q(x_{t+1}, a)$

$m_i = 0, \quad i \in 0, \ldots, N-1$

**for** $j \in 0, \ldots, N-1$ **do**

# Compute the projection of $\hat{\mathcal{T}} z_j$ onto the support $\{z_i\}$

$\hat{\mathcal{T}} z_j \leftarrow [r_t + \gamma_t z_j]_{V_{\text{MIN}}}^{V_{\text{MAX}}}$

$b_j \leftarrow (\hat{\mathcal{T}} z_j - V_{\text{MIN}})/\Delta z \quad \# \, b_j \in [0, N-1]$

$l \leftarrow \lfloor b_j \rfloor, u \leftarrow \lceil b_j \rceil$

# Distribute probability of $\hat{\mathcal{T}} z_j$

$m_l \leftarrow m_l + p_j(x_{t+1}, a^*)(u - b_j)$

$m_u \leftarrow m_u + p_j(x_{t+1}, a^*)(b_j - l)$

**end for**

**output** $-\sum_i m_i \log p_i(x_t, a_t)$ # Cross-entropy loss

# Categorical DQN

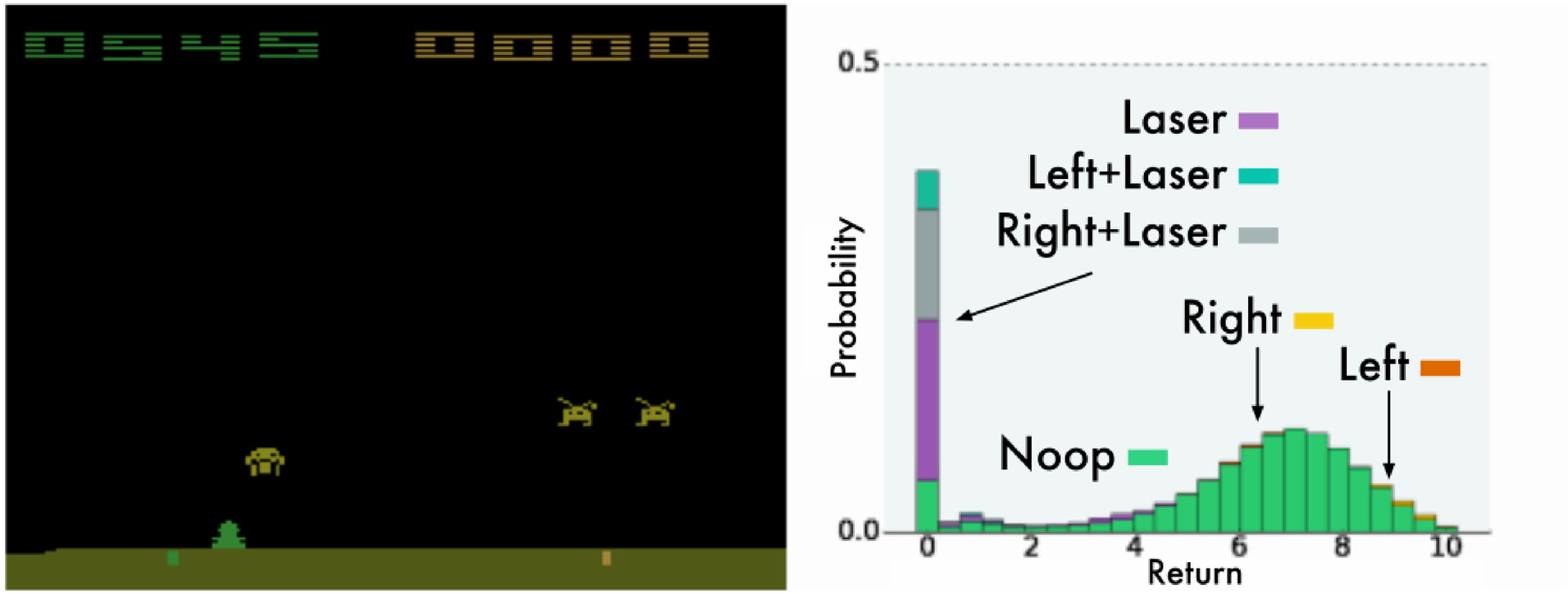
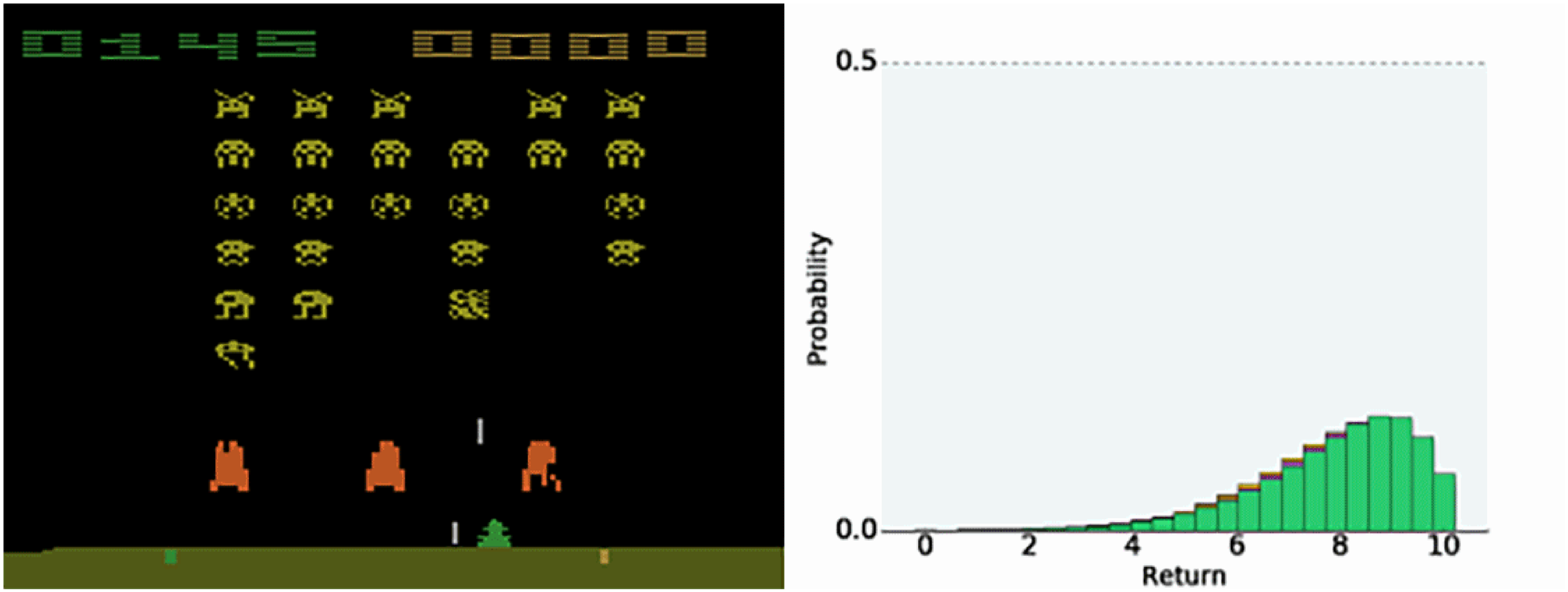



*Figure 4.* Learned value distribution during an episode of SPACE INVADERS. Different actions are shaded different colours. Returns below 0 (which do not occur in SPACE INVADERS) are not shown here as the agent assigns virtually no probability to them.

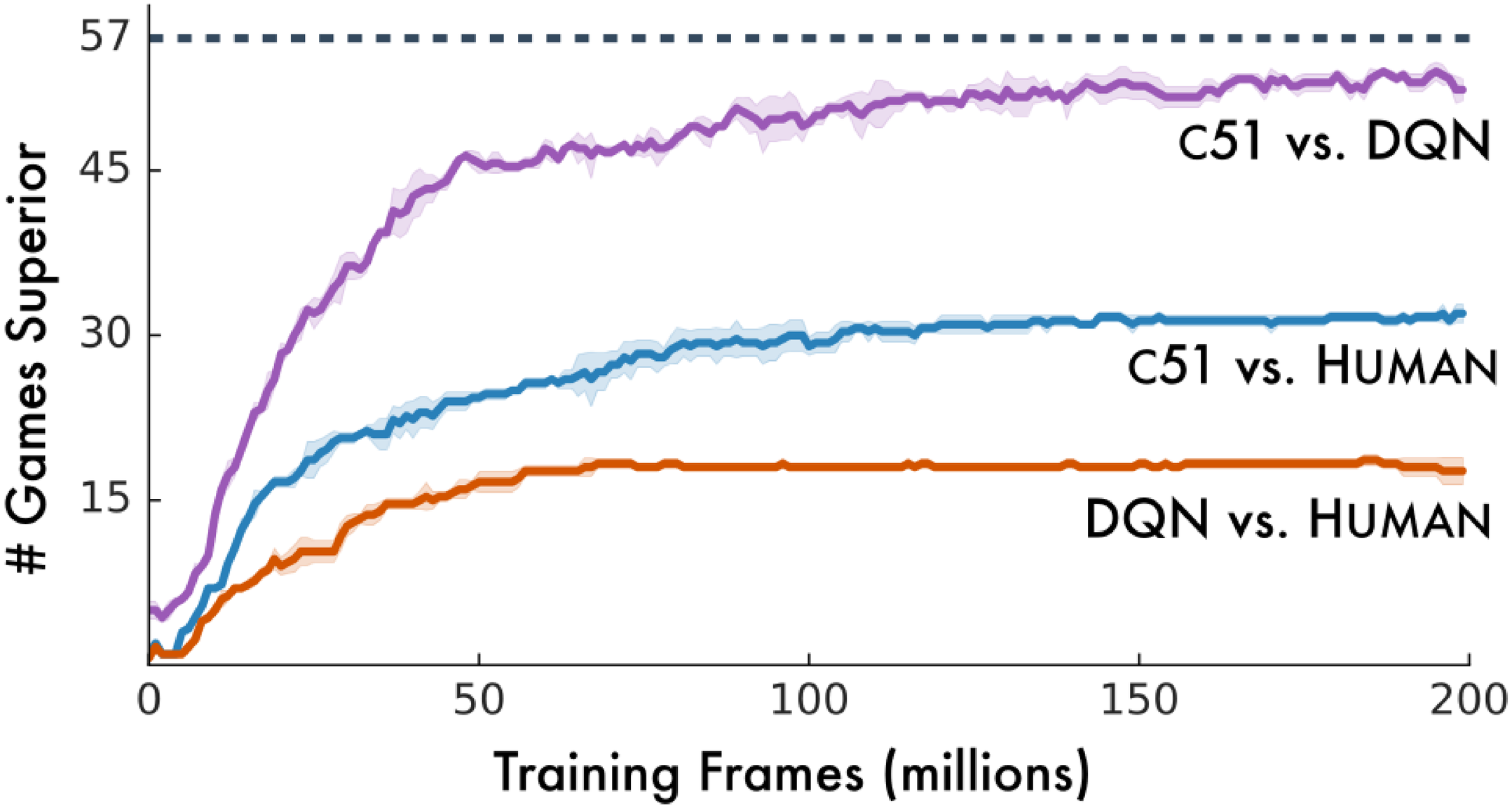
# Categorical DQN

- Having the full distribution of returns allow to deal with **uncertainty**.
- For certain actions in critical states, one could get a high return (killing an enemy) or no return (death).
- The distribution reflects that the agent is not certain of the goodness of the action. Expectations would not provide this information.




Source: https://deepmind.com/blog/article/going-beyond-average-reinforcement-learning

## Categorical DQN



# Categorical DQN

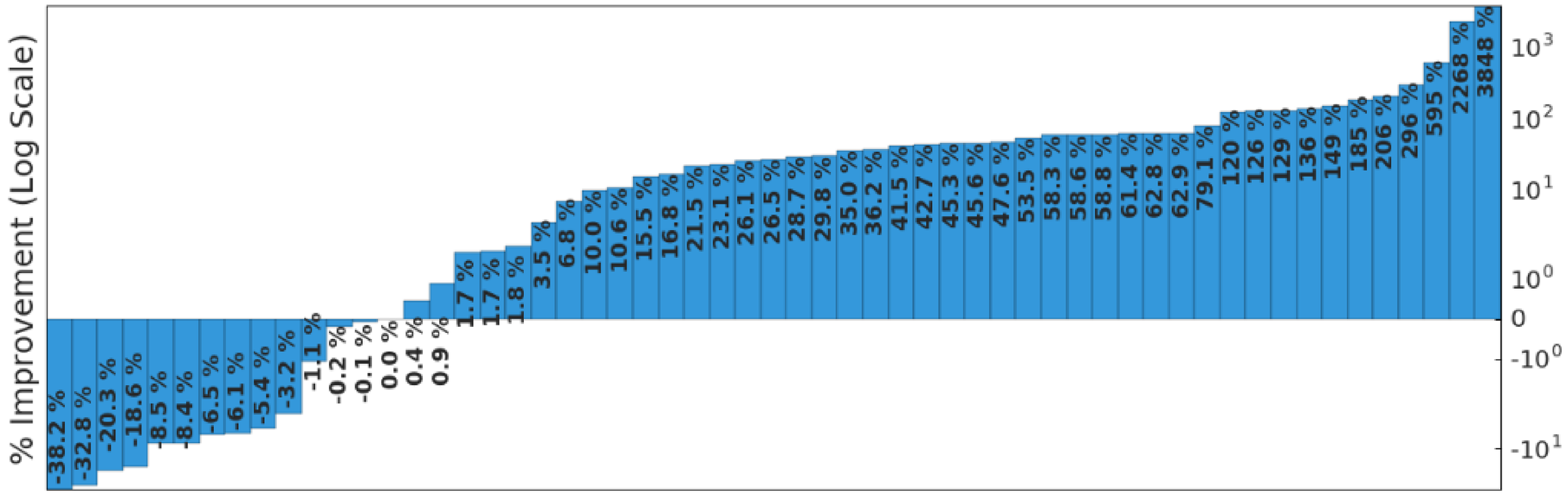
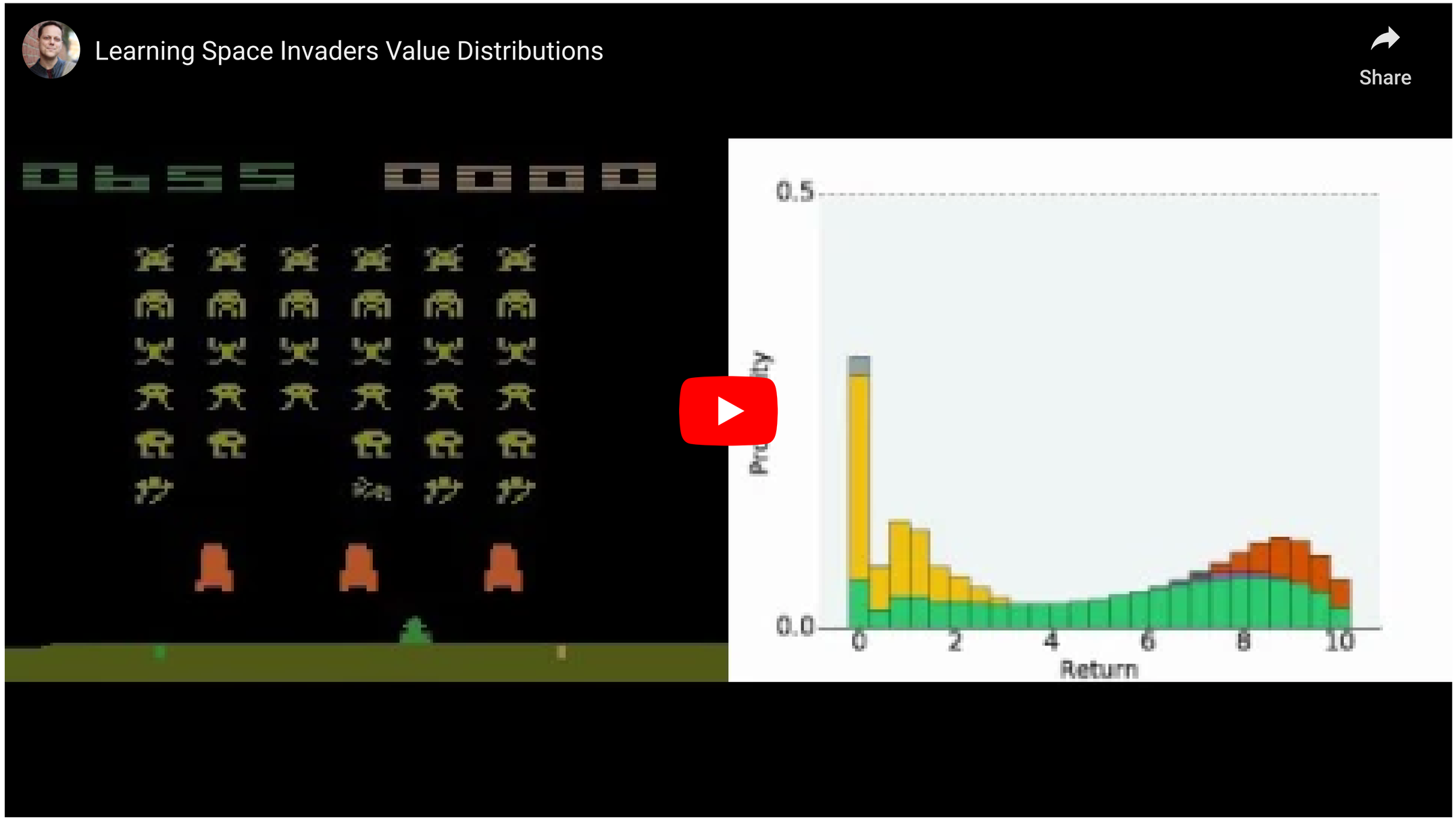
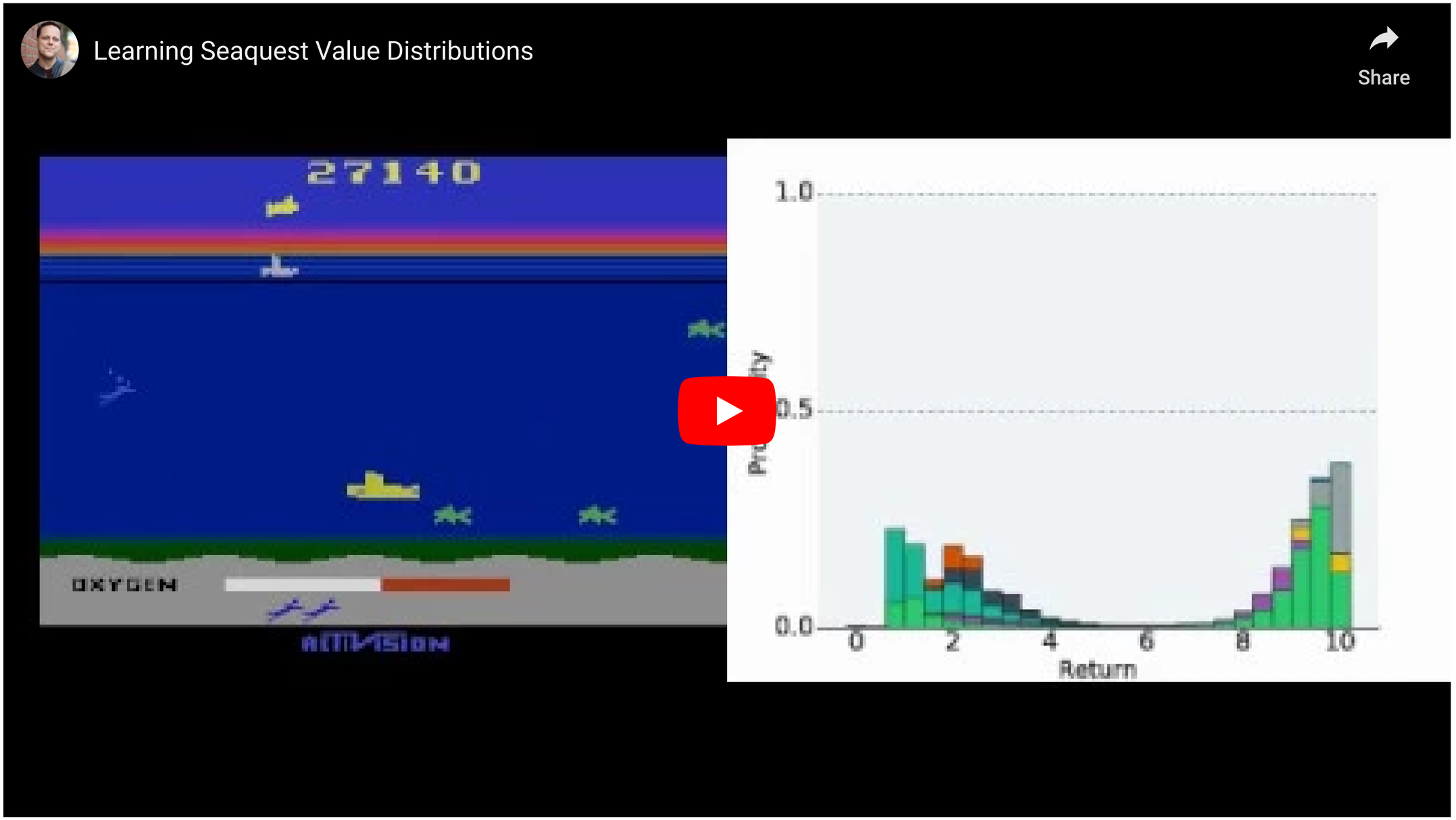



*Figure 7.* Percentage improvement, per-game, of C51 over Double DQN, computed using van Hasselt et al.'s method.

# Categorical DQN



# Categorical DQN



## Other variants of distributional learning

- **QR-DQN:** Dabney, W., Rowland, M., Bellemare, M. G., and Munos, R. (2017). Distributional Reinforcement Learning with Quantile Regression. arXiv:1710.10044
- **IQN:** Dabney, W., Ostrovski, G., Silver, D., and Munos, R. (2018). Implicit Quantile Networks for Distributional Reinforcement Learning. arXiv:1806.06923.
- **The Reactor:** Gruslys, A., Dabney, W., Azar, M. G., Piot, B., Bellemare, M., and Munos, R. (2017). The Reactor: A fast and sample-efficient Actor-Critic agent for Reinforcement Learning. arXiv:1704.04651.

# 2 - Noisy DQN

Published as a conference paper at ICLR 2018

## NOISY NETWORKS FOR EXPLORATION

**Meire Fortunato*** **Mohammad Gheshlaghi Azar*** **Bilal Piot ***

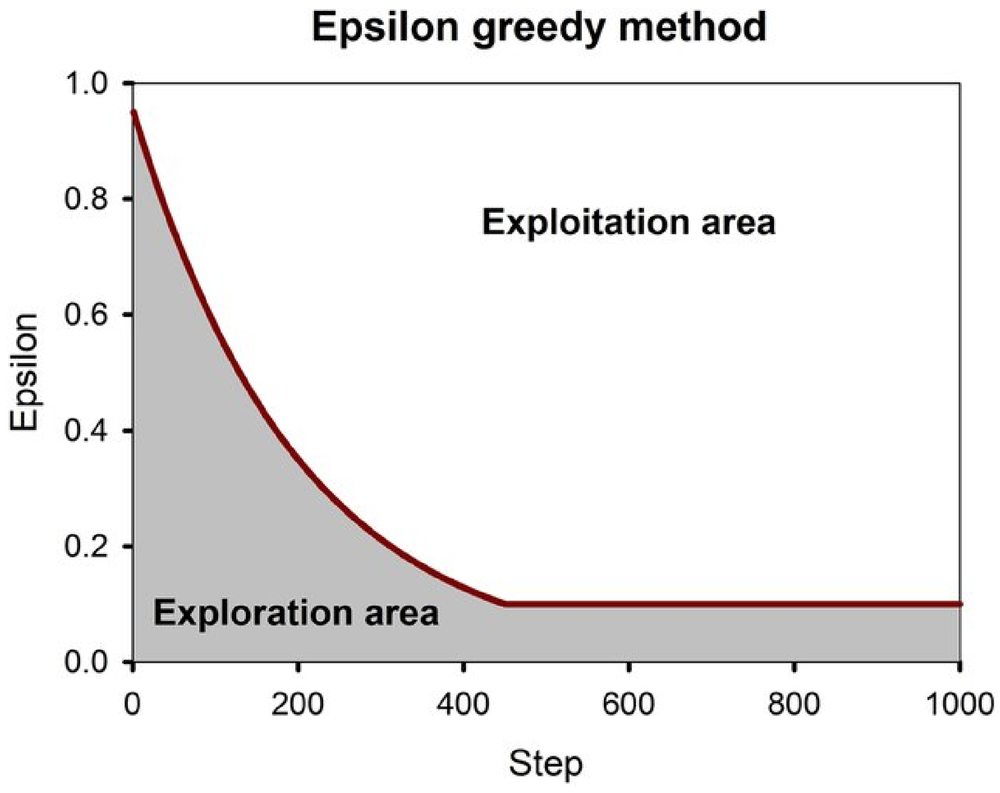
**Jacob Menick** **Matteo Hessel** **Ian Osband** **Alex Graves** **Volodymyr Mnih**

**Remi Munos** **Demis Hassabis** **Olivier Pietquin** **Charles Blundell** **Shane Legg**

DeepMind {meirefortunato,mazar,piot,
jmenick,mtthss,iosband,gravesa,vmnih,
munos,dhcontact,pietquin,cblundell,legg}@google.com

# Noisy DQN

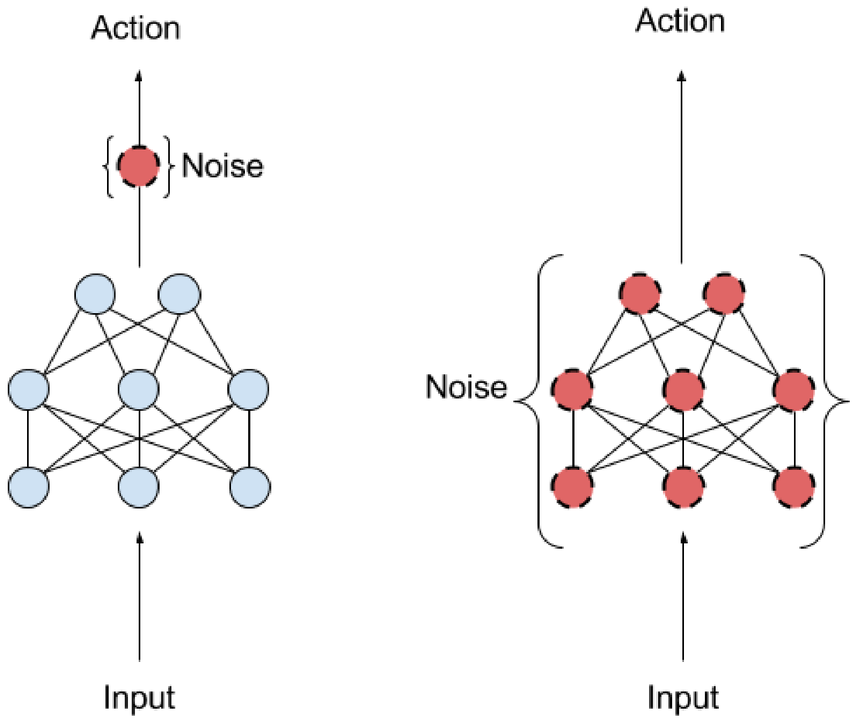
- DQN and its variants rely on $\epsilon$-greedy action selection over the Q-values to **explore**.
- The exploration parameter $\epsilon$ is **annealed** during training to reach a final minimal value.
- It is preferred to **softmax** action selection, where $\tau$ scales with the unknown Q-values.
- The problem is that it is a global exploration mechanism: well-learned states do not need as much exploration as poorly explored ones.




Source: https://www.researchgate.net/publication/334741451/figure/fig2/AS:786038515589120@1564417594220/Epsilon-greedy-method-At-each-step-a-random-number-is-generated-by-the-model-If-the_W640.jpg

# Noisy DQN

- $\epsilon$-greedy and softmax add **exploratory noise** to the output of DQN:
    - The Q-values predict a greedy action, but another action is taken.
- What about adding noise to the **parameters** (weights and biases) of the DQN, what would change the greedy action everytime?
- Controlling the level of noise inside the neural network indirectly controls the exploration level.




Source: https://openai.com/blog/better-exploration-with-parameter-noise/

- Note: a very similar idea was proposed by OpenAI at the same ICLR conference:

> Plappert, M., Houthooft, R., Dhariwal, P., Sidor, S., Chen, R. Y., Chen, X., et al. (2018). Parameter Space Noise for Exploration. arXiv:1706.01905.

# Noisy DQN

- Parameter noise builds on the idea of **Bayesian deep learning**.
- Instead of learning a single value of the parameters:

$$y = \theta_1 \, x + \theta_0$$

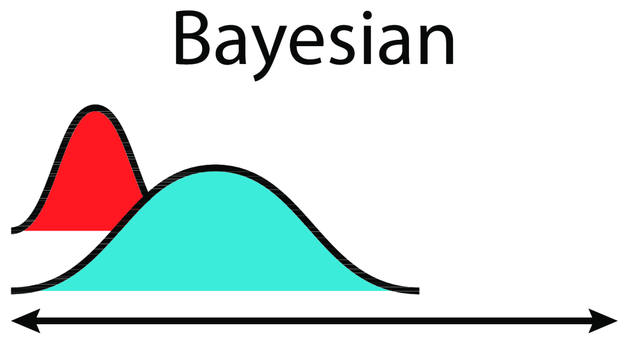
we learn the **distribution** of the parameters, for example by assuming they come from a normal distribution:

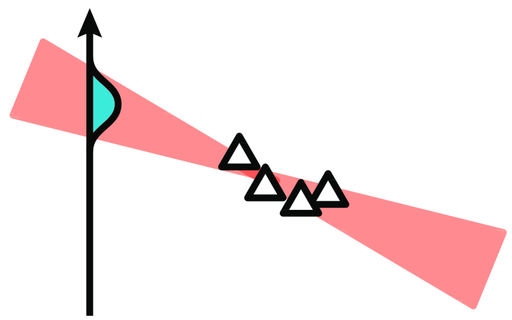
$$\theta \sim \mathcal{N}(\mu_\theta, \sigma_\theta^2)$$

- For each new input, we **sample** a value for the parameter:

$$\theta = \mu_\theta + \sigma_\theta \, \epsilon$$

with $\epsilon \sim \mathcal{N}(0, 1)$ a random variable.



Source: https://ericmjl.github.io/bayesian-deep-learning-demystified

- The prediction $y$ will vary for the same input depending on the variances:

$$y = (\mu_{\theta_1} + \sigma_{\theta_1} \, \epsilon_1) \, x + \mu_{\theta_0} + \sigma_{\theta_0} \, \epsilon_0$$

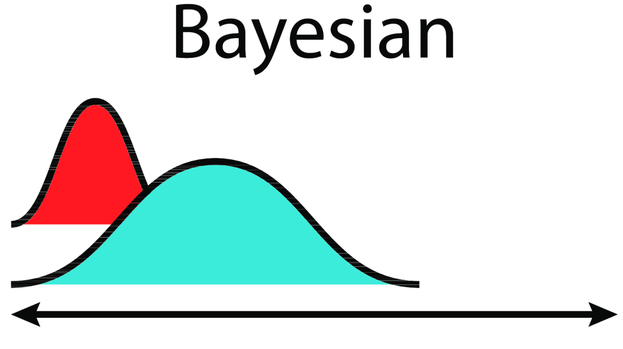
- The mean and variance of each parameter can be learned through backpropagation!

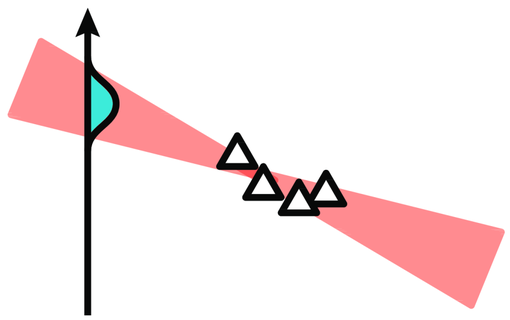
# Noisy DQN

- Probabilistic weights:

$$\theta \sim \mathcal{N}(\mu_\theta, \sigma_\theta^2)$$

- As the random variables $\epsilon_i \sim \mathcal{N}(0, 1)$ are not correlated with anything, the variances $\sigma_\theta^2$ should decay to 0.
- The variances $\sigma_\theta^2$ represent the **uncertainty** about the prediction $y$.
- Applied to DQN, this means that a state which has not been visited very often will have a high uncertainty:
  - The predicted Q-values will change a lot between two evaluations.
  - The greedy action might change: **exploration**.
- Conversely, a well-explored state will have a low uncertainty:
  - The greedy action stays the same: **exploitation**.

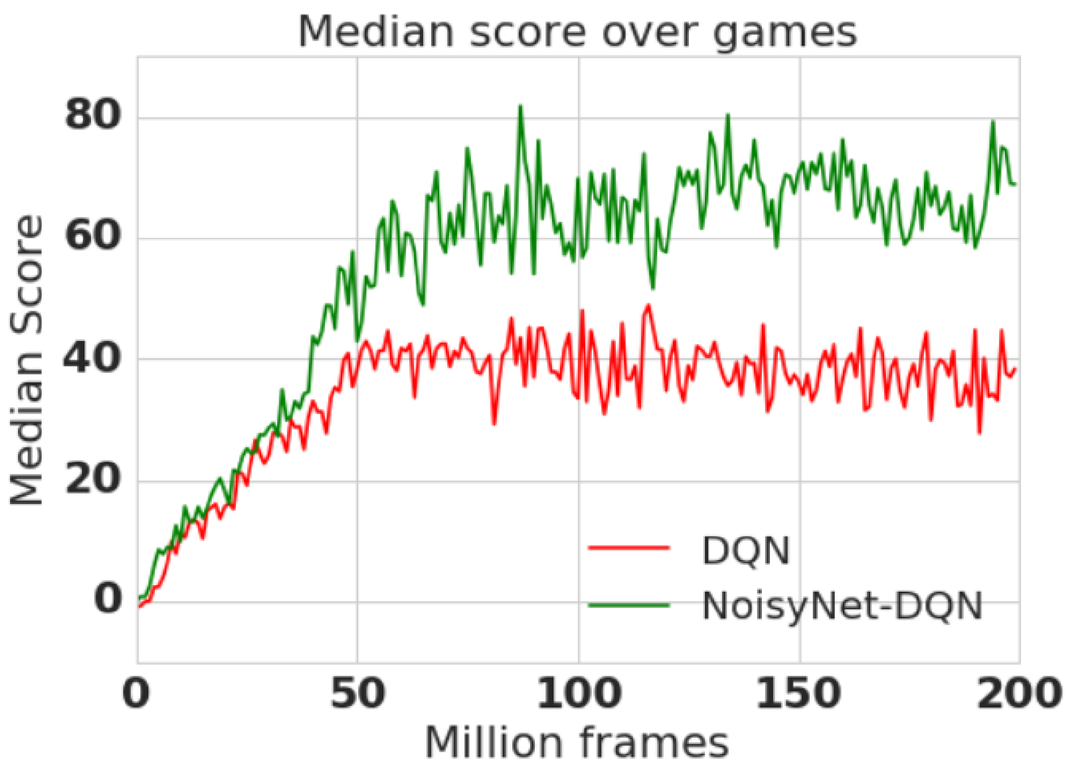


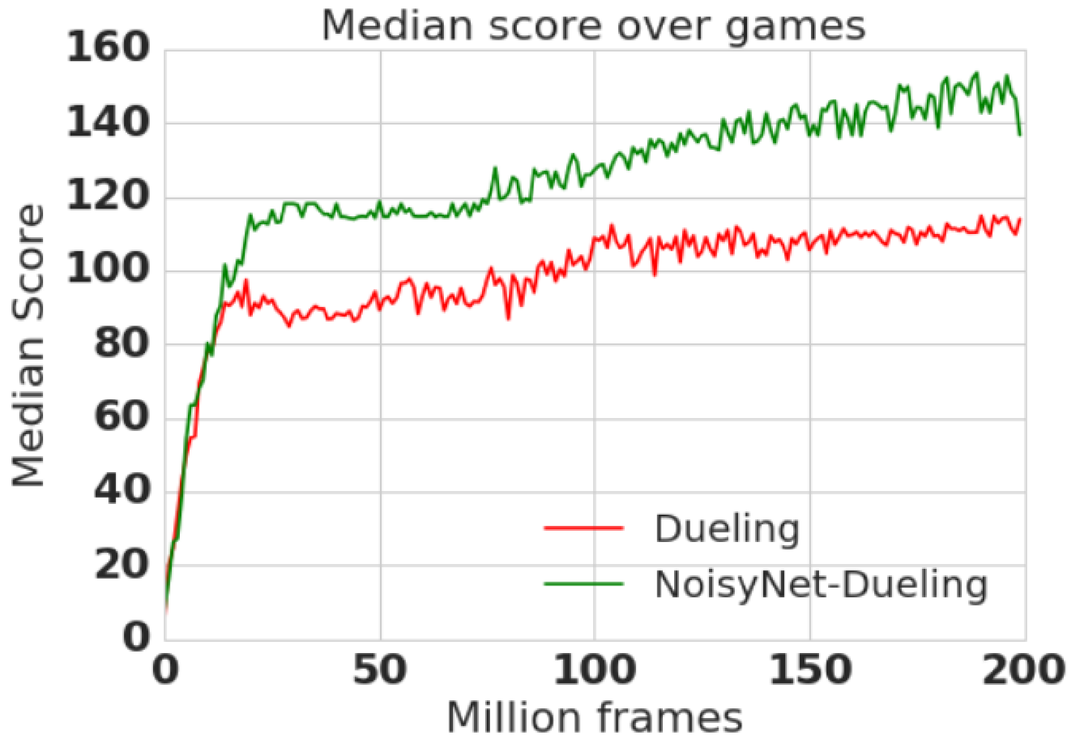



Source: https://ericmjl.github.io/bayesian-deep-learning-demystified

# Noisy DQN

- Noisy DQN uses **greedy action selection** over noisy Q-values.
- The level of exploration is **learned** by the network on a per-state basis. No need for scheduling!
- **Parameter noise** improves the performance of $\epsilon$-greedy-based methods, including DQN, dueling DQN, A3C, DDPG (see later), etc.



# 3 - Rainbow network

## Rainbow: Combining Improvements in Deep Reinforcement Learning

**Matteo Hessel**
DeepMind

**Joseph Modayil**
DeepMind

**Hado van Hasselt**
DeepMind

**Tom Schaul**
DeepMind

**Georg Ostrovski**
DeepMind

**Will Dabney**
DeepMind

**Dan Horgan**
DeepMind

**Bilal Piot**
DeepMind

**Mohammad Azar**
DeepMind

**David Silver**
DeepMind

# Raindow network

We have seen various improvements over a few years (2013-2017):

- Original DQN (Mnih et al., 2013)

$$\mathcal{L}(\theta) = \mathbb{E}_{\mathcal{D}}[(r + \gamma\, Q_{\theta'}(s', \text{argmax}_{a'} Q_{\theta'}(s', a')) - Q_{\theta}(s, a))^2]$$

- Double DQN (van Hasselt, Guez and Silver, 2015)

$$\mathcal{L}(\theta) = \mathbb{E}_{\mathcal{D}}[(r + \gamma\, Q_{\theta'}(s', \text{argmax}_{a'} Q_{\theta}(s', a')) - Q_{\theta}(s, a))^2]$$

- Prioritized Experience Replay (Schaul et al., 2015)

$$P(k) = \frac{(|\delta_k| + \epsilon)^\alpha}{\sum_k (|\delta_k| + \epsilon)^\alpha}$$

- Dueling DQN (Wang et al., 2016)

$$Q_\theta(s, a) = V_\alpha(s) + A_\beta(s, a)$$

- Categorical DQN (Bellemare, Dabney and Munos, 2017)

$$\mathcal{L}(\theta) = \mathbb{E}_{\mathcal{D}_s}\left[-\mathbf{t}_k \, \log Z_\theta(s_k, a_k)\right]$$

- NoisyNet (Fortunato et al., 2017)

$$\theta = \mu_\theta + \sigma_\theta \, \epsilon$$

Which of these improvements should we use?

# Rainbow network

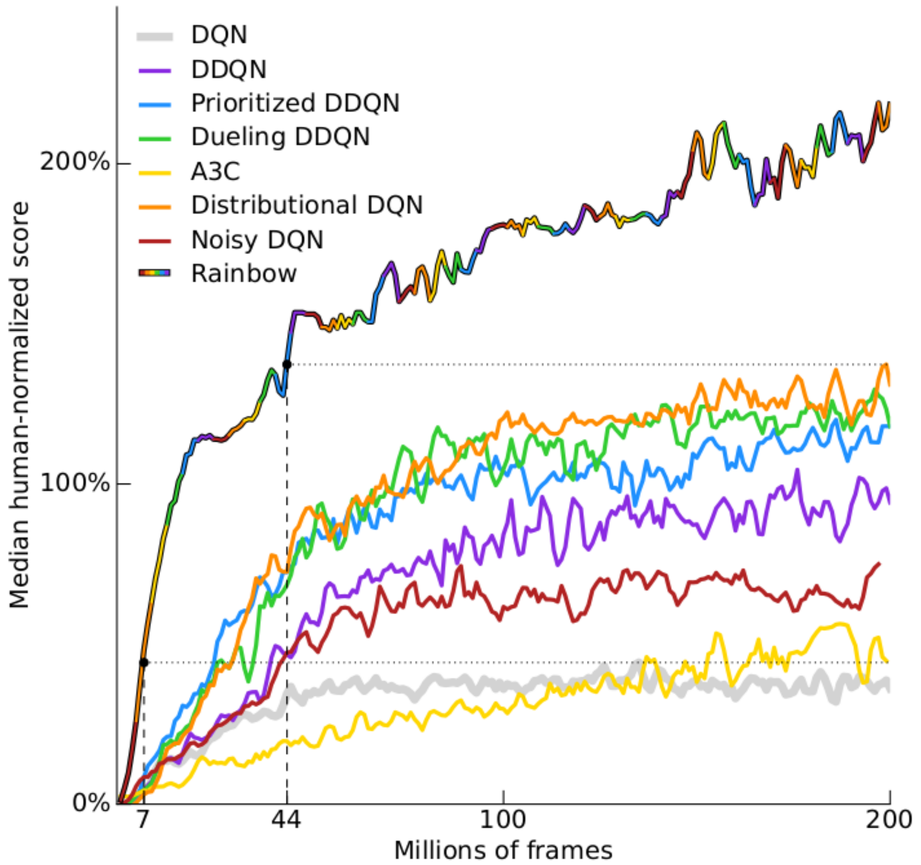



Figure 1: **Median human-normalized performance** across 57 Atari games. We compare our integrated agent (rainbow-colored) to DQN (grey) and six published baselines. Note that we match DQN's best performance after 7M frames, surpass any baseline within 44M frames, and reach substantially improved final performance. Curves are smoothed with a moving average over 5 points.

- Answer: all of them.
- The **rainbow network** combines :
  - double dueling DQN with PER.
  - categorical learning of return distributions.
  - parameter noise for exploration.
  - n-step return (n=3) for the bias/variance trade-off:

$$R_t = \sum_{k=0}^{n-1} \gamma^k r_{t+k+1} + \gamma^n \max_a Q(s_{t+n}, a)$$

- It outperforms any of the single improvements.

# Rainbow network

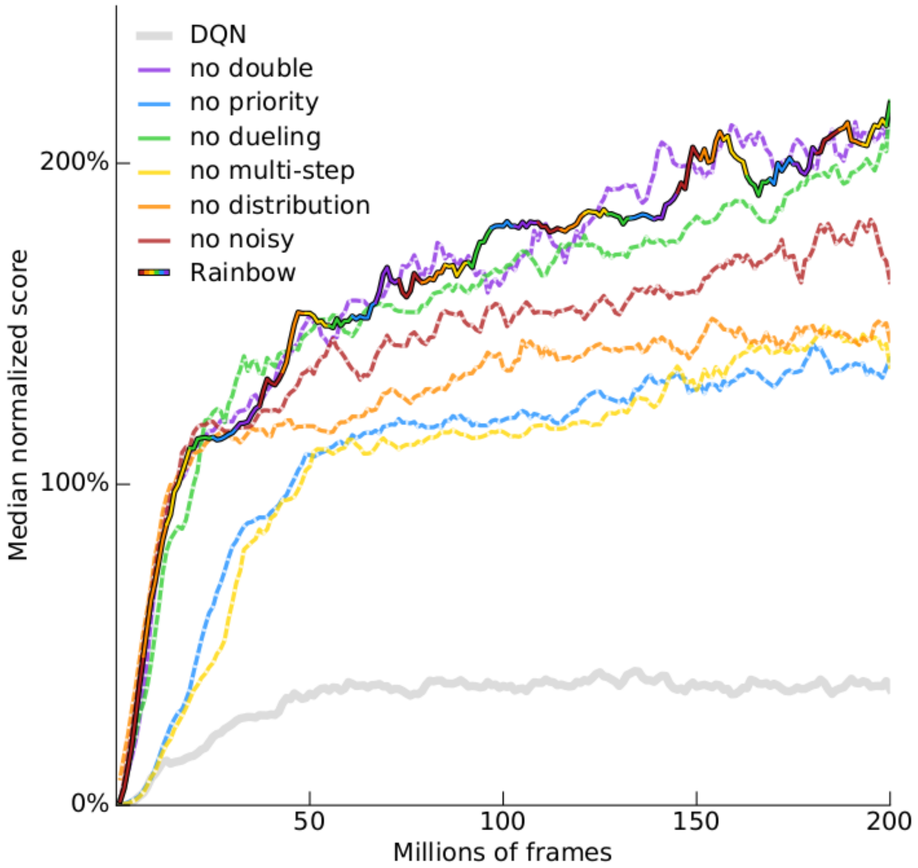



Figure 3: **Median human-normalized performance** across 57 Atari games, as a function of time. We compare our integrated agent (rainbow-colored) to DQN (gray) and to six different ablations (dashed lines). Curves are smoothed with a moving average over 5 points.

- Most of these mechanisms are necessary to achieve optimal performance (**ablation studies**).
- n-step returns, PER and distributional learning are the most critical.
- Interestingly, double Q-learning does not have a huge effect on the Rainbow network:
  - The other mechanisms (especially distributional learning) already ensure that Q-values are not over-estimated.
- You can find good implementations of Rainbow DQN on all major frameworks, for example on `rllib`:

https://docs.ray.io/en/latest/rllib-algorithms.html#deep-q-networks-dqn-rainbow-parametric-dqn

Hessel et al. (2017). Rainbow: Combining Improvements in Deep Reinforcement Learning. arXiv:1710.02298

# 4 - Gorila - General Reinforcement Learning Architecture

## Massively Parallel Methods for Deep Reinforcement Learning

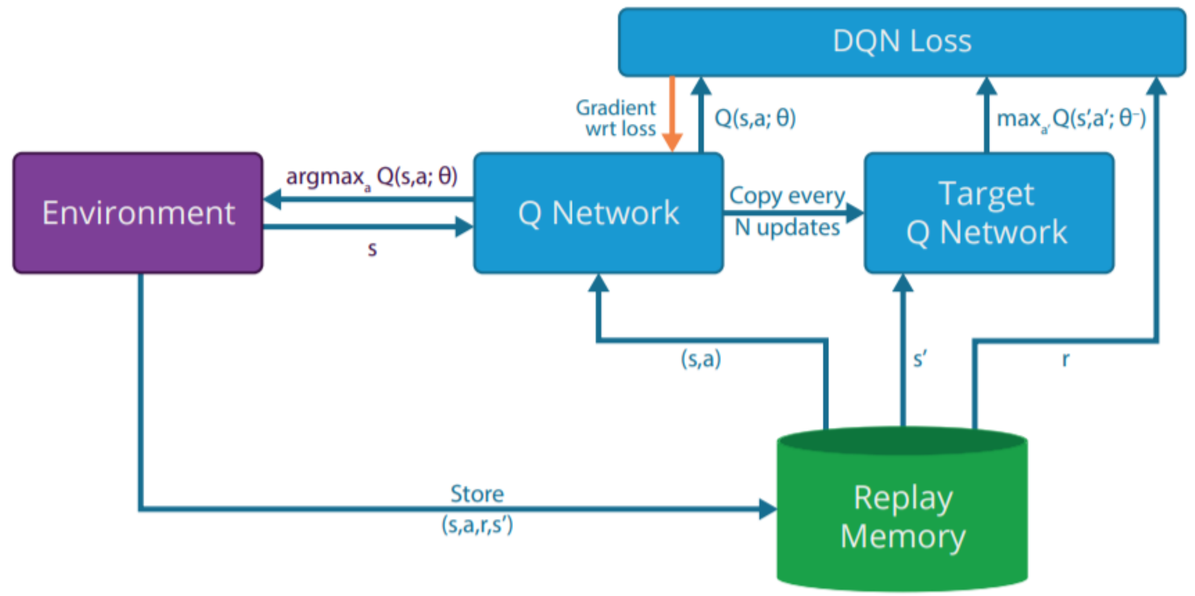
**Arun Nair, Praveen Srinivasan, Sam Blackwell, Cagdas Alcicek, Rory Fearon, Alessandro De Maria, Vedavyas Panneershelvam, Mustafa Suleyman, Charles Beattie, Stig Petersen, Shane Legg, Volodymyr Mnih, Koray Kavukcuoglu, David Silver**

{ARUNSNAIR, PRAV, BLACKWELLS, CAGDASALCICEK, RORYF, ADEMARIA, DARTHVEDA, MUSTAFASUL, CBEATTIE, SVP, LEGG, VMNIH, KORAYK, DAVIDSILVER @GOOGLE.COM }

Google DeepMind, London

# Gorila

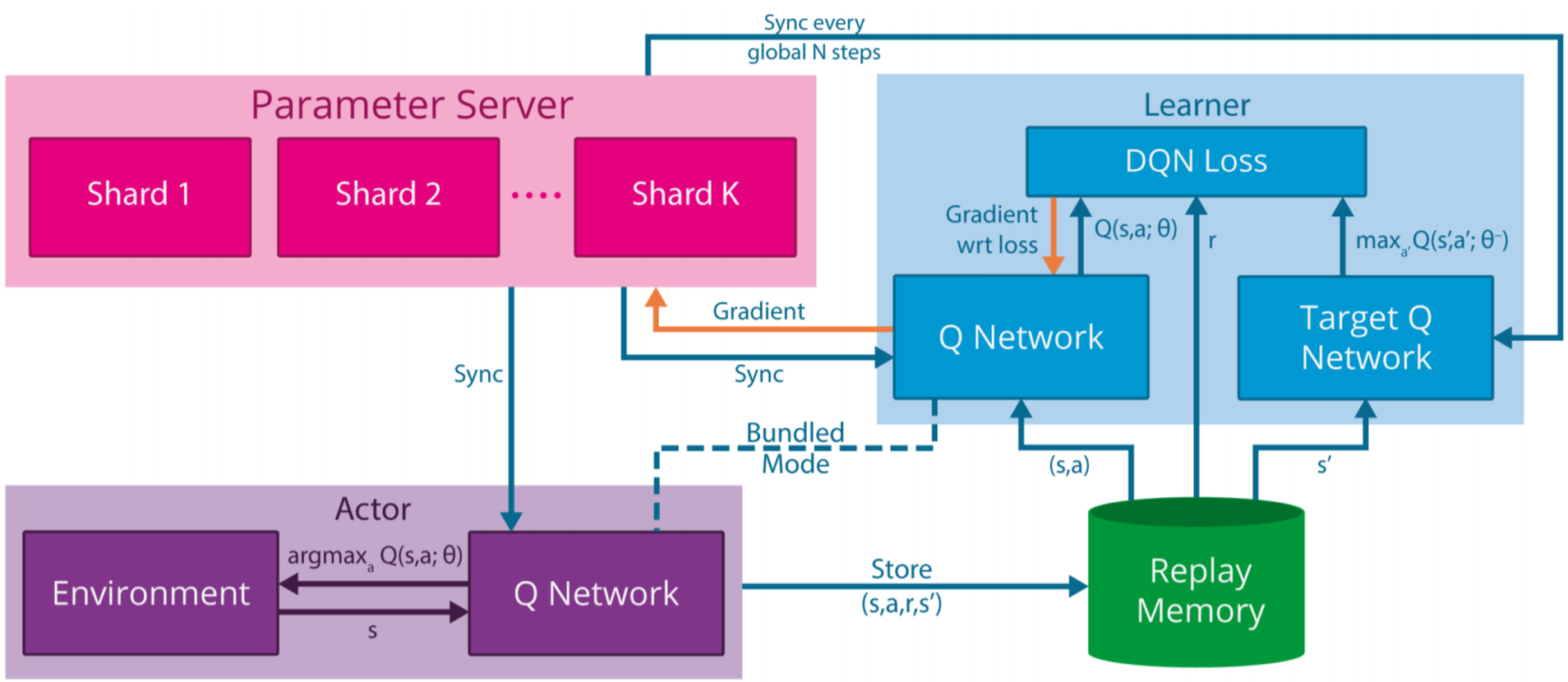
- The DQN value network $Q_\theta(s, a)$ has two jobs:
  - **actor:** it interacts with the environment to sample $(s, a, r, s')$ transitions.
  - **learner:** it learns from minibatches out of the replay memory.




- The weights of the value network lie on the same CPU/GPU, so the two jobs have to be done sequentially: **computational bottleneck**.
- DQN cannot benefit from **parallel computing**: multi-core CPU, clusters of CPU/GPU, etc.

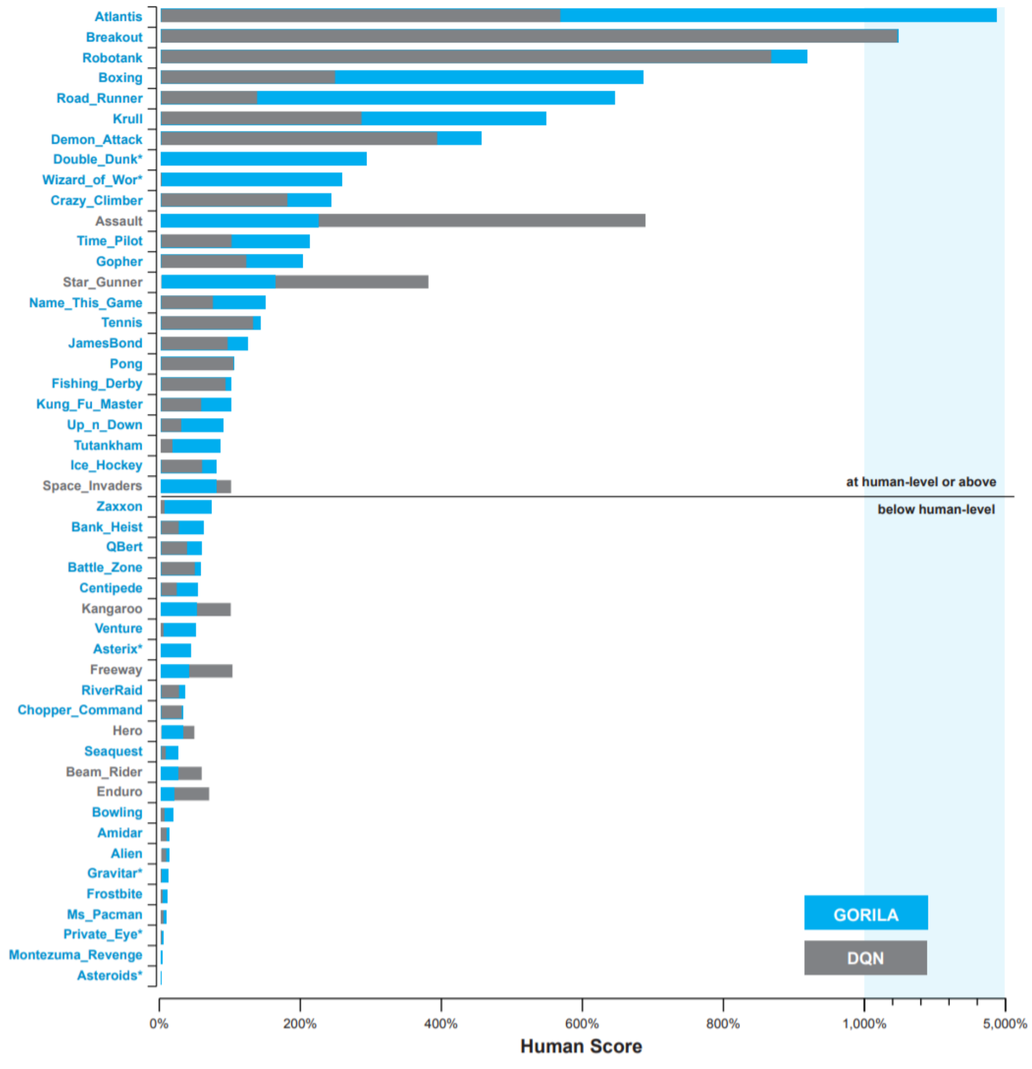
# Gorila

- The Gorila framework splits DQN into **multiple actors** and **multiple learners**.
- Each actor (or worker) **interacts** with its copy of the environment and stores transitions in a distributed replay buffer.
- Each learner samples minibatches from the replay buffer and computes **gradients** w.r.t the DQN loss.
- The parameter server (**master network**) applies the gradients on the parameters and frequently **synchronizes** the actors and learners.



# Gorila

- Gorila allows to train DQN on parallel hardware (e.g. clusters of GPU) as long as the environment can be copied (simulation).




- The final performance is not incredibly better than single-GPU DQN, but obtained much faster in wall-clock time (2 days instead of 12-14 days on a single GPU).

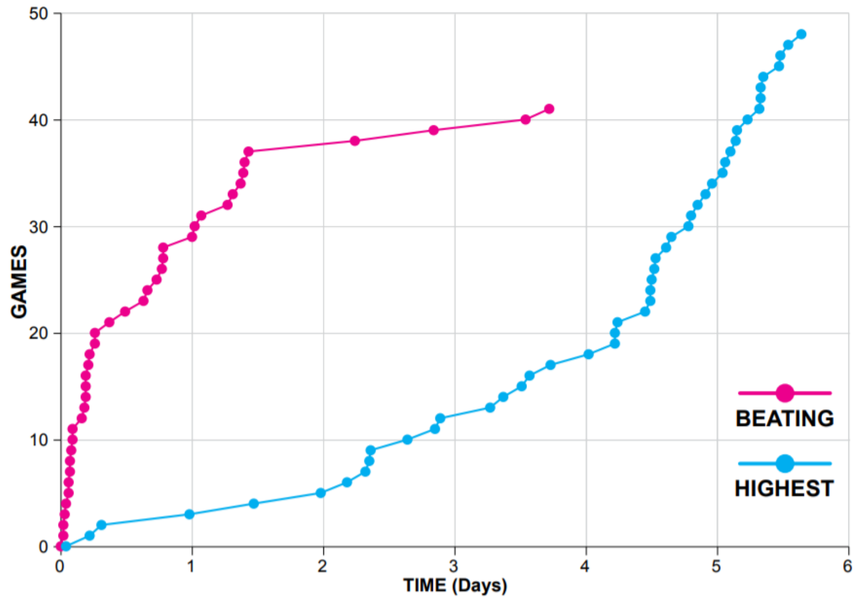



*Figure 5.* The time required by Gorila DQN to surpass single DQN performance (red curve) and to reach its peak performance (blue curve).

# Ape-X

- With more experience, Deepmind realized that a single learner is better. Distributed SGD (computing gradients with different learners) is not very efficient.
- What matters is collecting transitions very quickly (multiple workers) but using **prioritized experience replay** to learn from the most interesting ones.

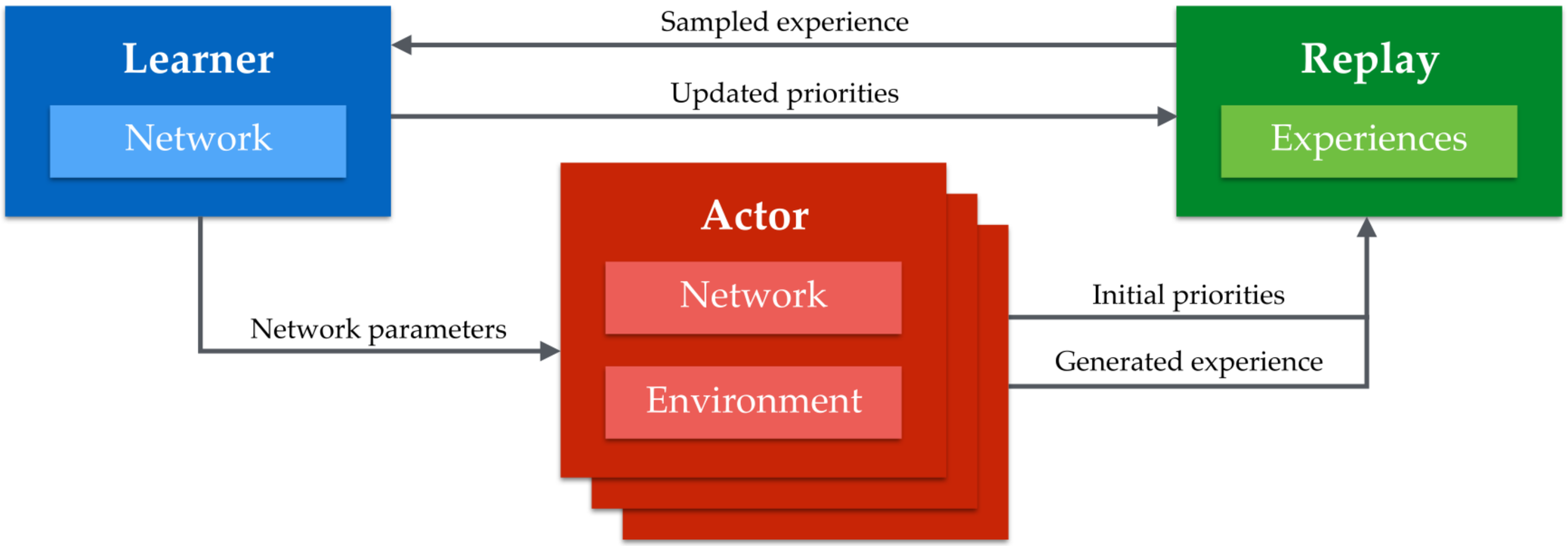



Figure 1: The Ape-X architecture in a nutshell: multiple actors, each with its own instance of the environment, generate experience, add it to a shared experience replay memory, and compute initial priorities for the data. The (single) learner samples from this memory and updates the network and the priorities of the experience in the memory. The actors' networks are periodically updated with the latest network parameters from the learner.

# Ape-X

- Using 360 workers (1 per CPU core), Ape-X reaches super-human performance for a fraction of the wall-clock training time.

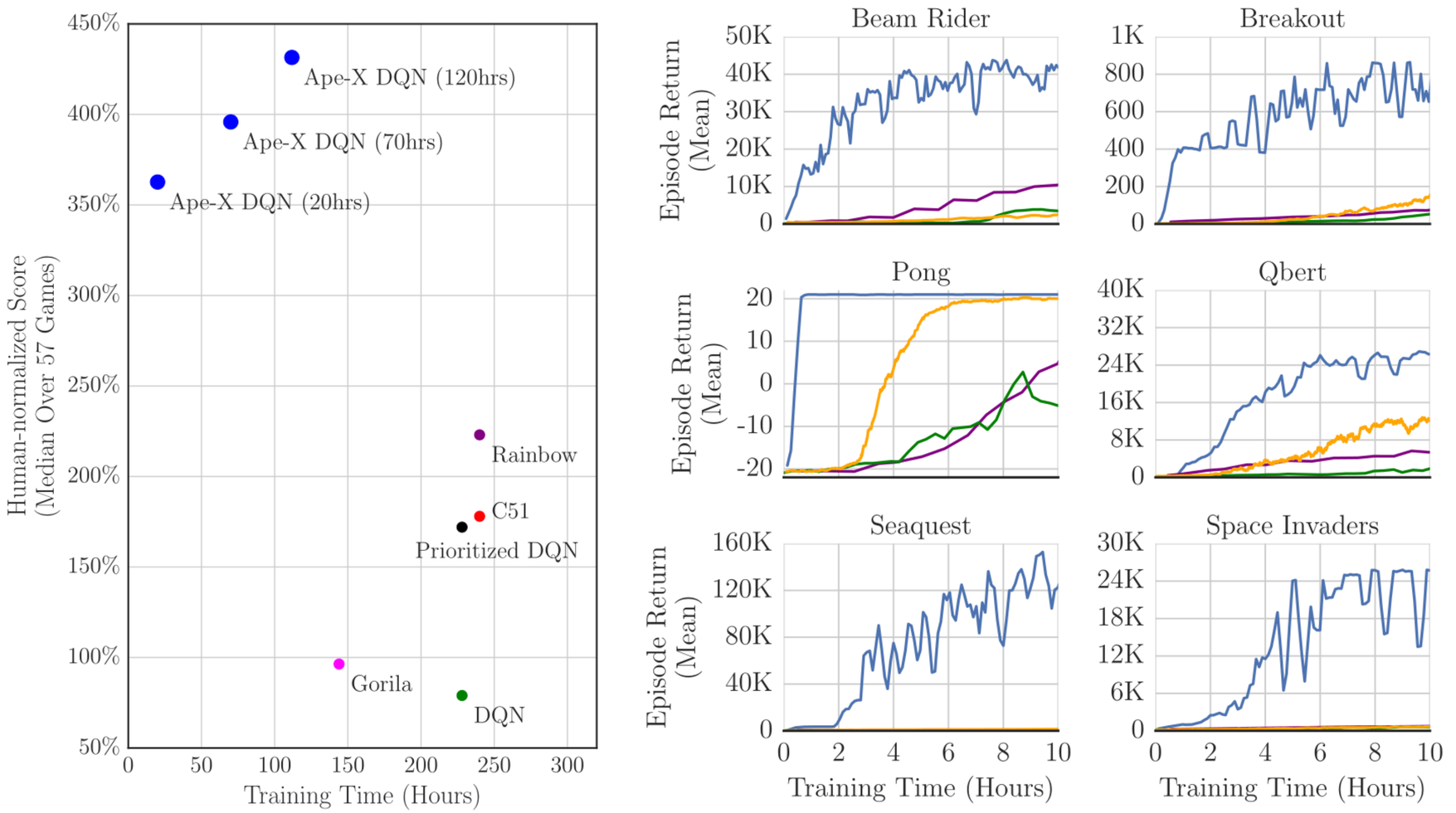



Figure 2: Left: Atari results aggregated across 57 games, evaluated from random no-op starts. Right: Atari training curves for selected games, against baselines. Blue: Ape-X DQN with 360 actors; Orange: A3C; Purple: Rainbow; Green: DQN. See appendix for longer runs over all games.

# Ape-X

- The multiple parallel workers can collect much more frames, leading to the better performance.
- The learner uses n-step returns and the double dueling DQN network architecture, so it is not much different from Rainbow DQN internally.

| Algorithm | Training Time | Environment Frames | Resources (per game) | Median (no-op starts) | Median (human starts) |
|---|---|---|---|---|---|
| Ape-X DQN | 5 days | 22800M | 376 cores, 1 GPU [a] | **434%** | **358%** |
| Rainbow | 10 days | 200M | 1 GPU | 223% | 153% |
| Distributional (C51) | 10 days | 200M | 1 GPU | 178% | 125% |
| A3C | 4 days | — | 16 cores | — | 117% |
| Prioritized Dueling | 9.5 days | 200M | 1 GPU | 172% | 115% |
| DQN | 9.5 days | 200M | 1 GPU | 79% | 68% |
| Gorila DQN [c] | ∼4 days | — | unknown [b] | 96% | 78% |
| UNREAL [d] | — | 250M | 16 cores | 331% [d] | 250% [d] |

Table 1: Median normalized scores across 57 Atari games. [a] Tesla P100. [b] >100 CPUs, with a mixed number of cores per CPU machine. [c] Only evaluated on 49 games. [d] Hyper-parameters were tuned per game.

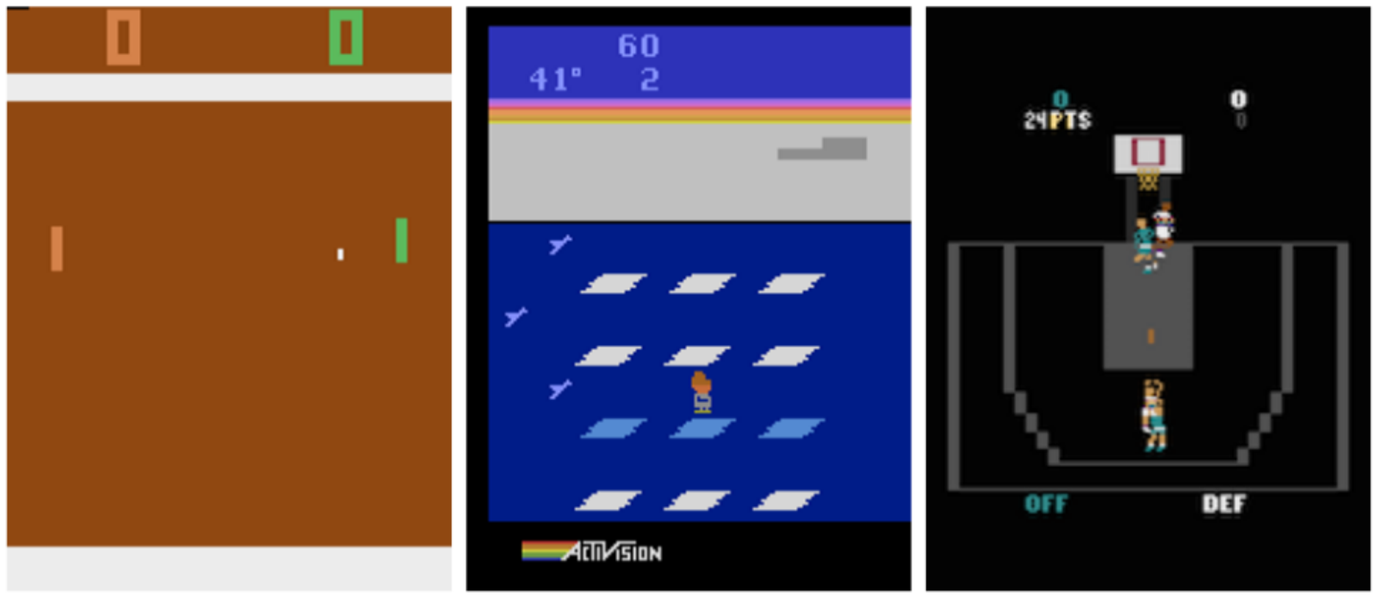
# 5 - DRQN: Deep Recurrent Q-network

## Deep Recurrent Q-Learning for Partially Observable MDPs

**Matthew Hausknecht and Peter Stone**
Department of Computer Science
The University of Texas at Austin
{mhauskn, pstone}@cs.utexas.edu

# DRQN: Deep Recurrent Q-network

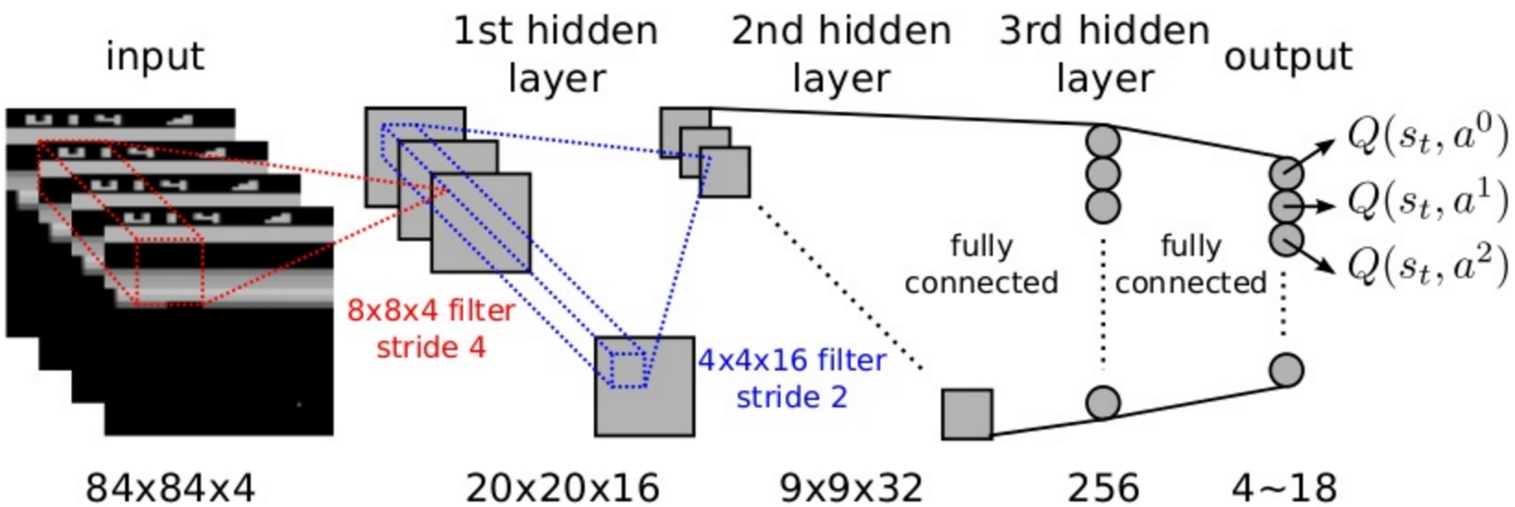
- Atari games are POMDP: each frame is a **partial observation**, not a Markov state.
- One cannot infer the velocity of the ball from a single frame.



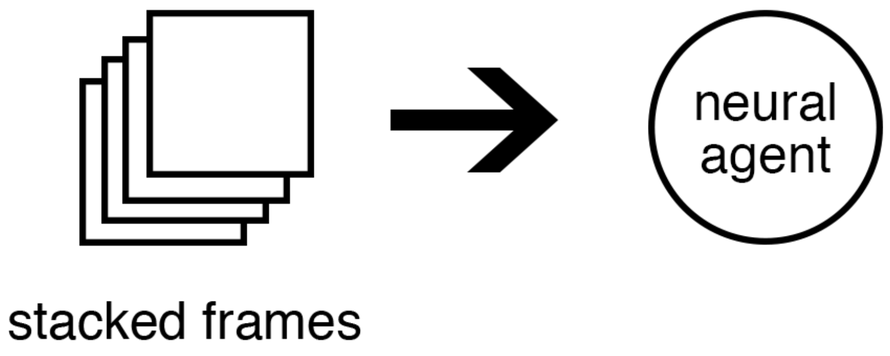

(a) Pong (b) Frostbite (c) Double Dunk

Figure 1: Nearly all Atari 2600 games feature moving objects. Given only one frame of input, Pong, Frostbite, and Double Dunk are all POMDPs because a single observation does not reveal the velocity of the ball (Pong, Double Dunk) or the velocity of the icebergs (Frostbite).

# DRQN: Deep Recurrent Q-network

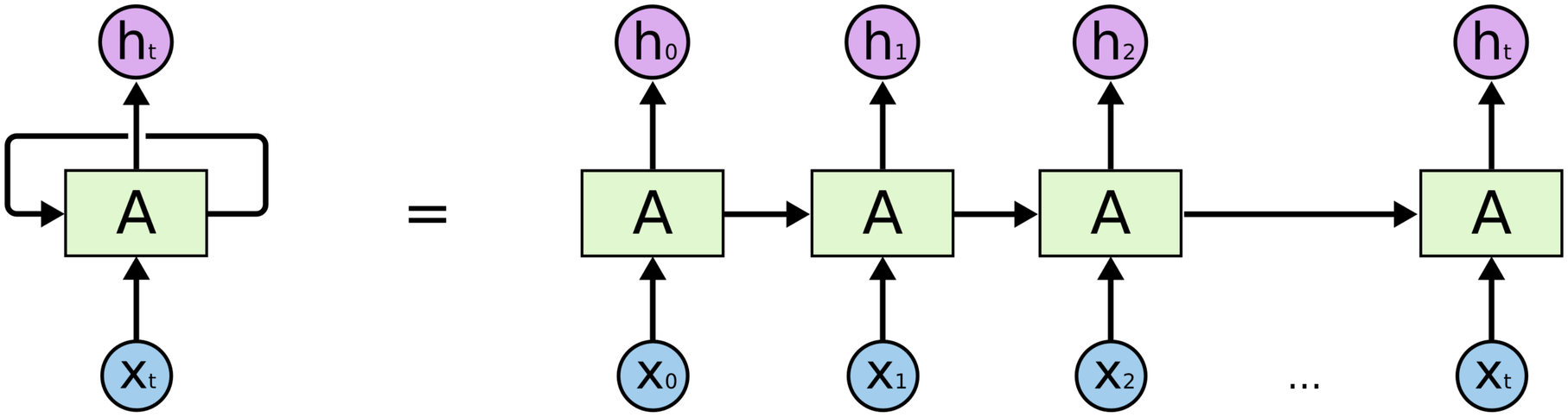


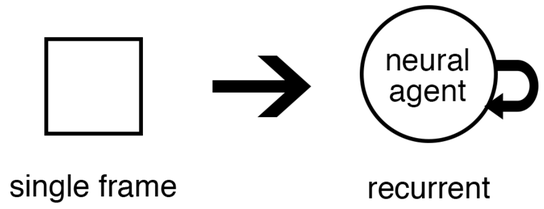

Source: https://medium.com/emergent-future/simple-reinforcement-learning-with-tensorflow-part-6-partial-observability-and-deep-recurrent-q-68463e9aeefc




- The trick used by DQN and its variants is to **stack** the last four frames and provide them as inputs to the CNN.
- The last 4 frames have (almost) the Markov property.

# DRQN: Deep Recurrent Q-network




Source: https://medium.com/emergent-future/simple-reinforcement-learning-with-tensorflow-part-6-partial-observability-and-deep-recurrent-q-68463e9aeefc

- The alternative is to use a **recurrent neural network** (e.g. LSTM) to encode the **history** of single frames.

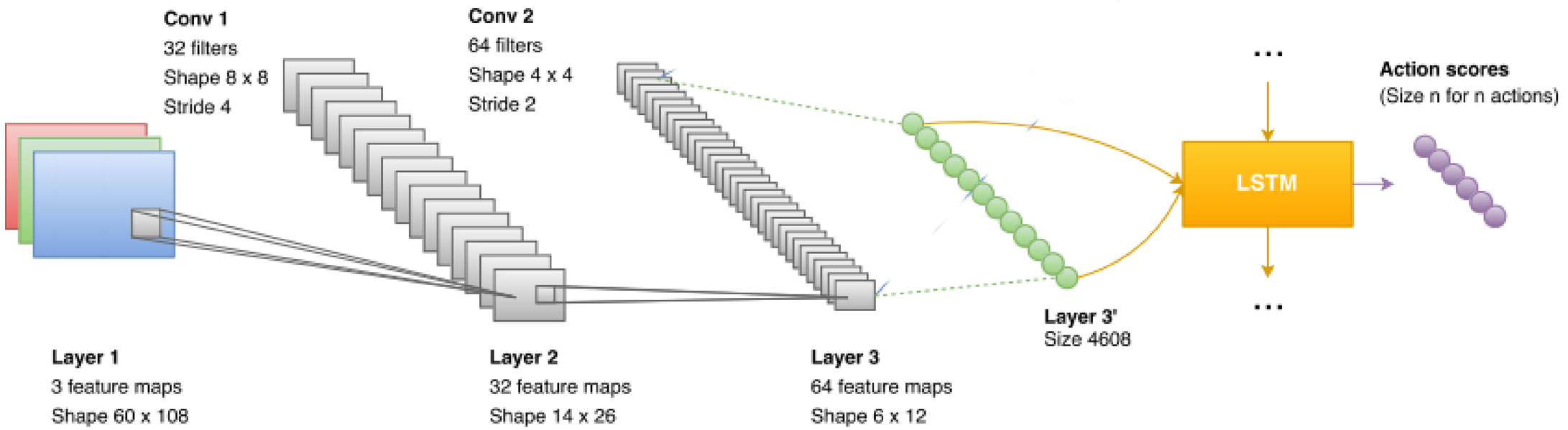
$$\mathbf{h}_t = f(W_x \times \mathbf{x}_t + W_h \times \mathbf{h}_{t-1} + \mathbf{b})$$

- The output at time $t$ depends on the whole history of inputs $(\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_t)$.

- Using the output of a LSTM as a state, we make sure that we have the Markov property, RL will work:

$$P(\mathbf{h}_{t+1}|\mathbf{h}_t) = P(\mathbf{h}_{t+1}|\mathbf{h}_t, \mathbf{h}_{t-1}, \ldots, \mathbf{h}_0)$$

# DRQN: Deep Recurrent Q-network

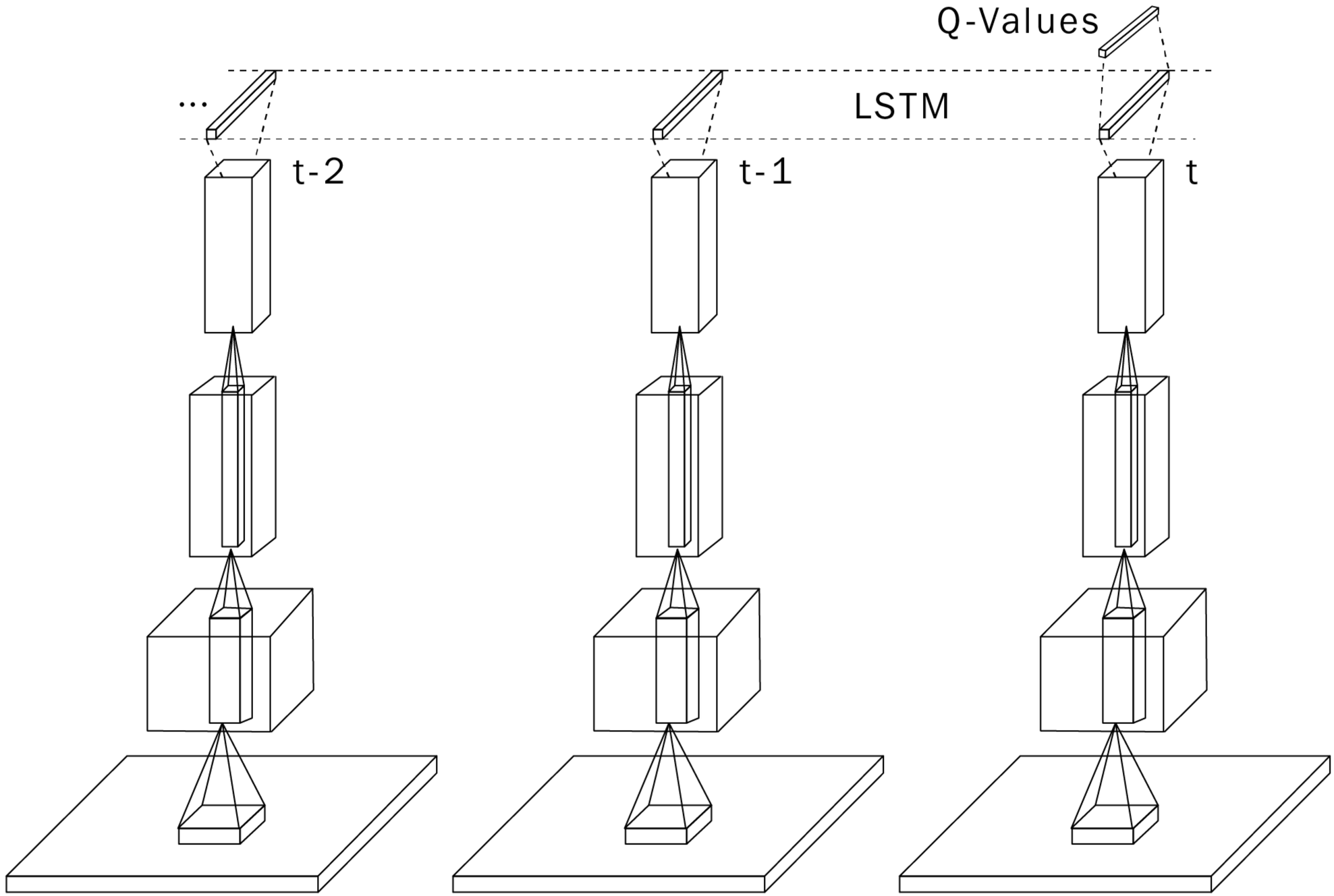



Source: https://blog.acolyer.org/2016/11/23/playing-fps-games-with-deep-reinforcement-learning/

- For the neural network, it is just a matter of adding a LSTM layer before the output layer.
- The convolutional layers are **feature extractors** for the LSTM layer.
- The loss function does not change: backpropagation (through time) all along.

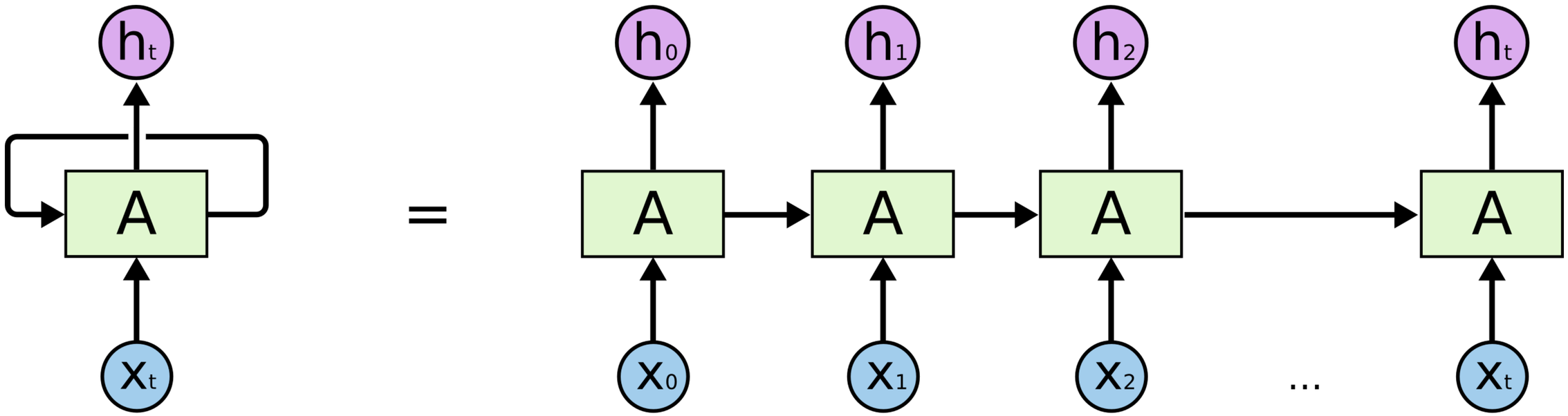
$$\mathcal{L}(\theta) = \mathbb{E}_{\mathcal{D}}[(r + \gamma\, Q_{\theta'}(s', \text{argmax}_{a'} Q_{\theta}(s', a')) - Q_{\theta}(s, a))^2]$$

# DRQN: Deep Recurrent Q-network




Hausknecht, M., and Stone, P. (2015). Deep Recurrent Q-Learning for Partially Observable MDPs. arXiv:1507.06527

## DRQN: Deep Recurrent Q-network




- The only problem is that RNNs are trained using truncated **backpropagation through time** (BPTT).
- One needs to provide a partial history of $T = 10$ inputs to the network in order to learn one output:

$$\left(\mathbf{x}_{t-T}, \mathbf{x}_{t-T+1}, \ldots, \mathbf{x}_t\right)$$

- The **experience replay memory** should not contain single transitions $\left(s_t, a_t, r_{t+1}, s_{t+1}\right)$, but a partial history of transitions.

$$\left(s_{t-T}, a_{t-T}, r_{t-T+1}, s_{t-T+1}, \ldots, s_t, a_t, r_{t+1}, s_{t+1}\right)$$

# DRQN: Deep Recurrent Q-network

- Using a LSTM layer helps on certain games, where temporal dependencies are longer that 4 frames, but impairs on others.

| | DRQN $\pm std$ | DQN $\pm std$ | |
|---|---|---|---|
| Game | | Ours | Mnih et al. |
| Asteroids | 1020 ($\pm$312) | 1070 ($\pm$345) | 1629 ($\pm$542) |
| Beam Rider | 3269 ($\pm$1167) | **6923** ($\pm$1027) | 6846 ($\pm$1619) |
| Bowling | 62 ($\pm$5.9) | 72 ($\pm$11) | 42 ($\pm$88) |
| Centipede | 3534 ($\pm$1601) | 3653 ($\pm$1903) | 8309 ($\pm$5237) |
| Chopper Cmd | 2070 ($\pm$875) | 1460 ($\pm$976) | 6687 ($\pm$2916) |
| Double Dunk | **-2** ($\pm$7.8) | -10 ($\pm$3.5) | -18.1 ($\pm$2.6) |
| Frostbite | **2875** ($\pm$535) | 519 ($\pm$363) | 328.3 ($\pm$250.5) |
| Ice Hockey | -4.4 ($\pm$1.6) | -3.5 ($\pm$3.5) | -1.6 ($\pm$2.5) |
| Ms. Pacman | 2048 ($\pm$653) | 2363 ($\pm$735) | 2311 ($\pm$525) |

Table 1: On standard Atari games, DRQN performance parallels DQN, excelling in the games of Frostbite and Double Dunk, but struggling on Beam Rider. Bolded font indicates statistical significance between DRQN and our DQN.[5]

# DRQN: Deep Recurrent Q-network

- Beware: LSTMs are extremely slow to train (but not to use).
- Stacking frames is still a reasonable option.

| | Backwards (ms) | | | Forwards (ms) | | |
|---|---|---|---|---|---|---|
| Frames | 1 | 4 | 10 | 1 | 4 | 10 |
| Baseline | 8.82 | 13.6 | 26.7 | 2.0 | 4.0 | 9.0 |
| Unroll 1 | 18.2 | 22.3 | 33.7 | 2.4 | 4.4 | 9.4 |
| Unroll 10 | 77.3 | 111.3 | 180.5 | 2.5 | 4.4 | 8.3 |
| Unroll 30 | 204.5 | 263.4 | 491.1 | 2.5 | 3.8 | 9.4 |

Table 2: Average milliseconds per backwards/forwards pass. Frames refers to the number of channels in the input image. Baseline is a non recurrent network (e.g. DQN). Unroll refers to an LSTM network backpropagated through time 1/10/30 steps.

# 6 - R2D2: Recurrent Replay Distributed DQN

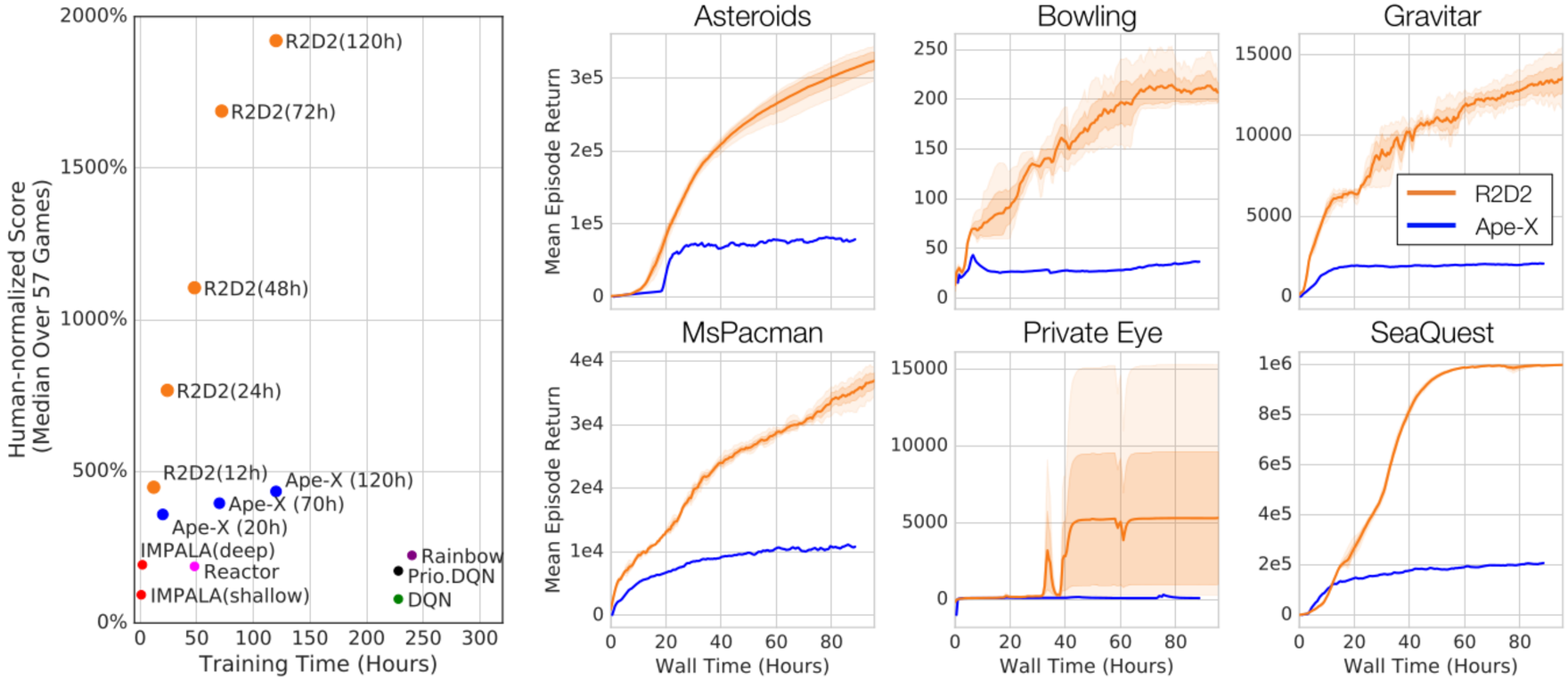
Published as a conference paper at ICLR 2019

# RECURRENT EXPERIENCE REPLAY IN DISTRIBUTED REINFORCEMENT LEARNING

**Steven Kapturowski, Georg Ostrovski, John Quan, Rémi Munos, Will Dabney**
DeepMind, London, UK
{skapturowski,ostrovski,johnquan,munos,wdabney}@google.com

# R2D2: Recurrent Replay Distributed DQN

- R2D2 builds on Ape-X and DRQN:
  - double dueling DQN with n-step returns (n=5) and prioritized experience replay.
  - 256 actors, 1 learner.
  - 1 LSTM layer after the convolutional stack.
- Additionally solving practical problems with LSTMs (initial state), it became the state of the art on Atari-57 until November 2019…




Kapturowski, S., Ostrovski, G., Quan, J., Munos, R., and Dabney, W. (2019). Recurrent experience replay in distributed reinforcement learning. ICLR.