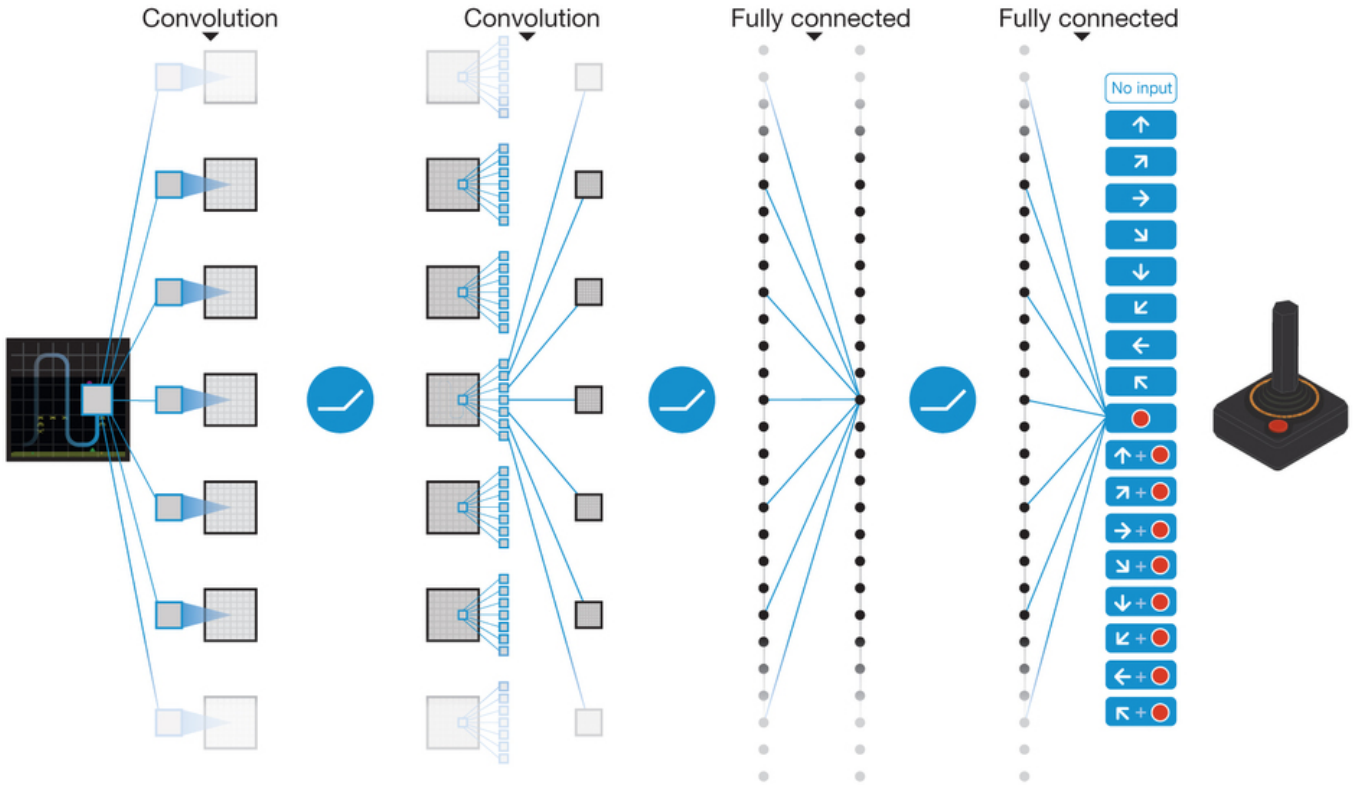
Human-level control through deep reinforcement learning

Volodymyr Mnih, Koray Kavukcuoglu, David Silver et. al.

Google Deepmind



Presented by
Muhammed Kocabas

Figure from paper

Outline

- Introduction
- What is Reinforcement Learning?
- Q-Learning
- Background and related work
- Methods
- Experiments
- Results
- Conclusion
- References

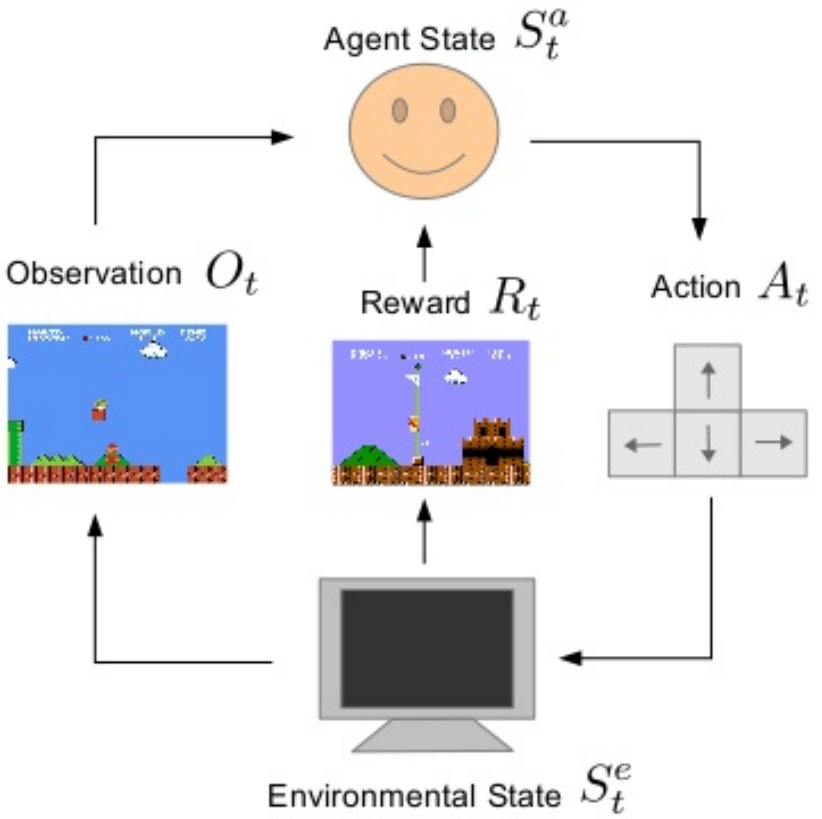
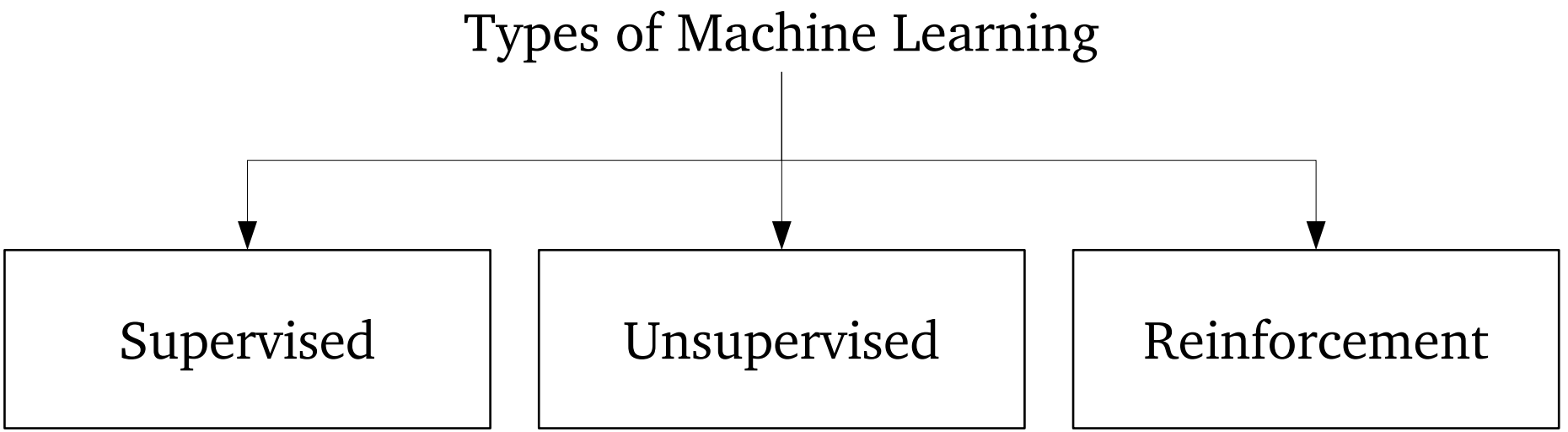


Figure from link

Introduction

- Former reinforcement learning agents are successful in some domains in which useful features can be ***handcrafted***, or in ***fully observed***, ***low dimensional*** state spaces.
- Authors used the recent advances in deep neural network training, and developed ***Deep-Q-Network***.
- Deep-Q-Network:
  - Learns successful policies
  - End-to-end RL method.
  - In Atari 2600 games, receives only the ***pixels and the game score*** as inputs just like a human learns and perform super-human capabilities in half of the games.
- **Comparison:** Human players, linear learners

What is reinforcement learning?



Supervised	Unsupervised	Reinforcement
Task driven	Data driven	Learns to react an environment
(Regression/Classification)	(Clustering)	Sparse and time delayed labels (Rewards)

RL is the problem of getting an agent to act in the world so as to ***maximize its rewards.***

What is reinforcement learning?

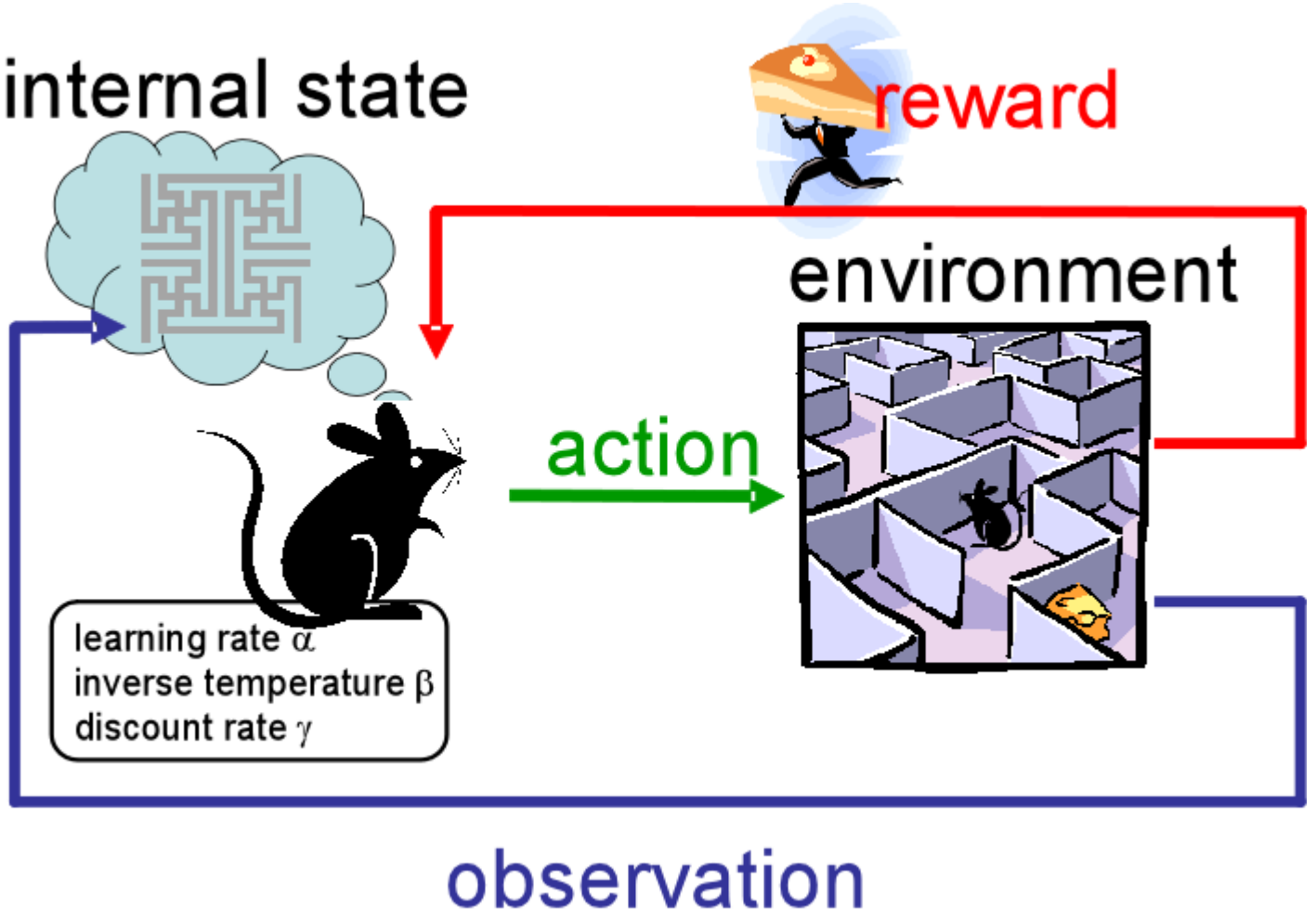
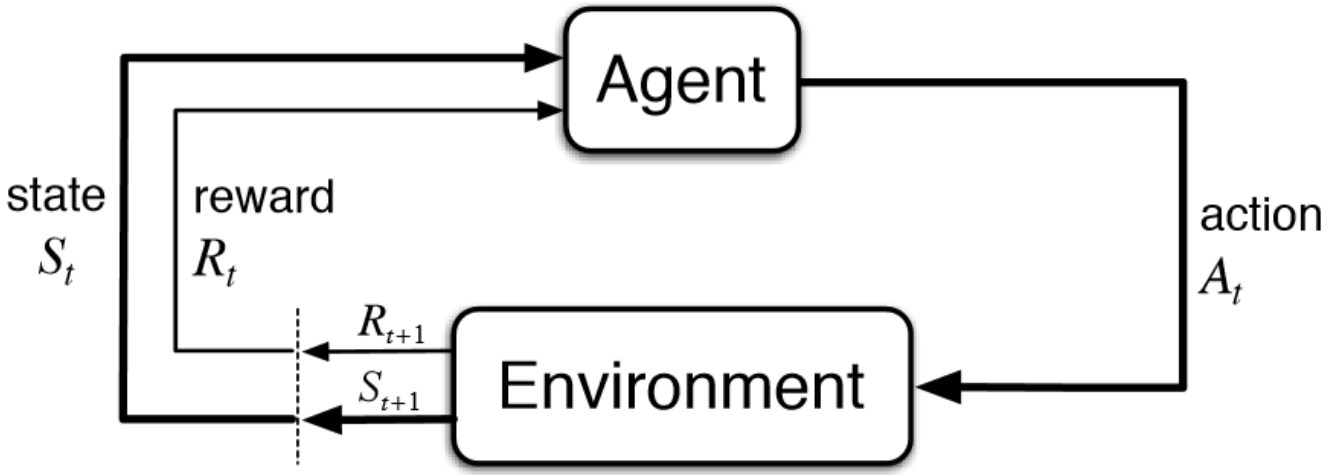


Figure from link

What is reinforcement learning?

- No explicit training data set.
- Nature provides reward for each of the learners actions.
- At each time,
  - Learner has a state and choses an action.
  - Nature responds with new state and a reward.
  - Learner learns from reward and makes better decisions.



What is reinforcement learning?

- Main goal is to maximizing the reward R_t
- Looking at only immediate rewards wouldn't work well.
- We need to take into account "future" rewards.
- At time t, the total future reward is:

$$R_t = r_t + r_{t+1} + \ldots + r_n$$

- We want to take the action that maximizes R_t
- But we have to consider the fact that:

The environment is stochastic

Discounted future rewards

- We can never be sure, if we will get the same rewards the next time we perform the same actions. The more into the future we go, the more it may diverge.
- Main goal is to maximizing the reward R_t

$$R_t = r_t + \gamma\, r_{t+1} + \gamma^2\, r_{t+2} + \ldots + \gamma^{n-t}\, r_n$$

- γ *is* the discount factor between 0 and 1.
- The more into the future the reward is, the less we take it into consideration.
- Simply:

$$R_t = r_t + \gamma\, (r_{t+1} + \gamma\, (r_{t+2} + \ldots\,))$$

$$\boldsymbol{R_t = r_t + \gamma\, R_{t+1}}$$

Q-Function

$$Q(s_t,\ a_t) = max\ R_{t+1}$$

- The maximum discounted future reward when we perform **action a** in **state s**, and continue optimally from that point on.
- The way to think about $Q(s_t,\ a_t)$ is that it is "*the best possible score at the end of the game after performing action a in state s*".
- It is called **Q-function**, because it represents the "quality" of a certain action in a given state.

Policy & Bellman equation

$$\pi(s) = argmax_a \; Q(s, \; a)$$

- π represents the **policy**, the rule how we choose an action in each state.

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

- This is called the **Bellman equation**.
- Maximum **future reward** for this **state** and **action** is the **immediate reward** plus maximum **future reward** for the **next state**.
- The basic idea behind many reinforcement learning algorithms is to estimate the action-value function by using the Bellman equation as an iterative update

Q-Learning

```
initialize Q[num_states,num_actions] arbitrarily
observe initial state s
repeat
      select and carry out an action a
      observe reward r and new state s'
      Q[s,a] = Q[s,a] + α(r + γ max_a' Q[s',a'] - Q[s,a])
      s = s'
until terminated
```

$$Q_{k+1}(s,a) := (1-\alpha)Q(s,a) + \alpha(c(s,a) + \gamma \min_b Q_k(s',b))$$

- α in the algorithm is a **learning rate** that controls how much of the difference between previous Q-value and newly proposed Q-value is taken into account.
- The $max_{a'} Q(s', a')$ that we use to update $Q(s, a)$ is only an approximation and in early stages of learning it may be completely wrong. However the approximation get more and more accurate with every iteration and it has been shown, that if we perform this update enough times, then the Q-function will converge and represent the true Q-value.

Related Work

- Neural fitted Q-iteration (2005)
  - Riedmiller, M. *Neural fitted Q iteration - first experiences with a data efficient neural reinforcement learning method*. Mach. Learn.: ECML
- Deep auto-encoder NN in RL (2010)
  - Lange, S. & Riedmiller, M. *Deep auto-encoder neural networks in reinforcement learning.* Proc. Int. Jt. Conf. Neural. Netw.

Comparison Papers:

- The arcade learning environment: An evaluation platform for general agents (2013)
  - Bellemare, M. G., Naddaf, Y., Veness, J. & Bowling, M. *The arcade learning environment: An evaluation platform for general agents.* J. Artif. Intell. Res.
- Investigating contingency awareness using Atari 2600 games (2012)
  - Bellemare, M. G., Veness, J. & Bowling, M. *Investigating contingency awareness using Atari 2600 games.* Proc. Conf. AAAI. Artif. Intell.

Neural Fitted Q-Iteration

- This paper introduces NFQ, an algorithm for efficient and effective training of a Q-value function represented by a multi-layer perceptron.
- The main drawback of this type of architecture is that a separate forward pass is required to compute the Q-value of each action, resulting in a cost that scales linearly with the number of actions.
- This method involve the repeated training of networks de novo on hundreds of iterations.

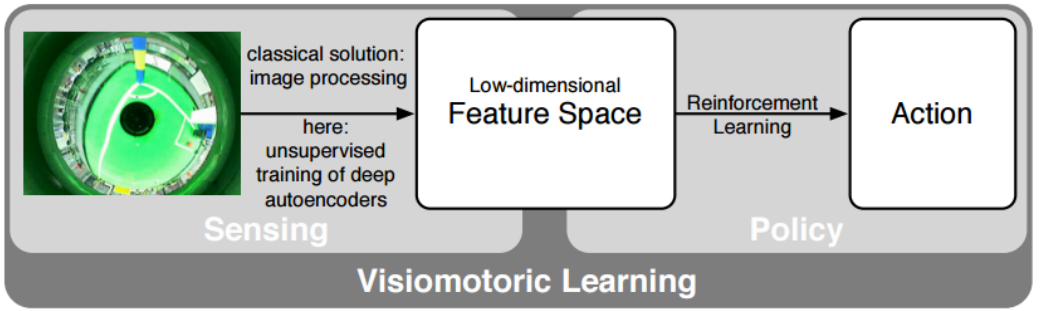
NFQ_main() {
input: a set of transition samples D; output: Q-value function Q_N
k=0
init_MLP() $\rightarrow Q_0$;
Do {
generate_pattern_set $P = \{(input^l, target^l), l = 1, \ldots, \#D\}$ where:
$input^l = s^l, u^l,$
$target^l = c(s^l, u^l, s'^l) + \gamma \, min_b Q_k(s'^l, b)$
Rprop_training(P) $\rightarrow Q_{k+1}$
k:= k+1
} WHILE ($k < N$)

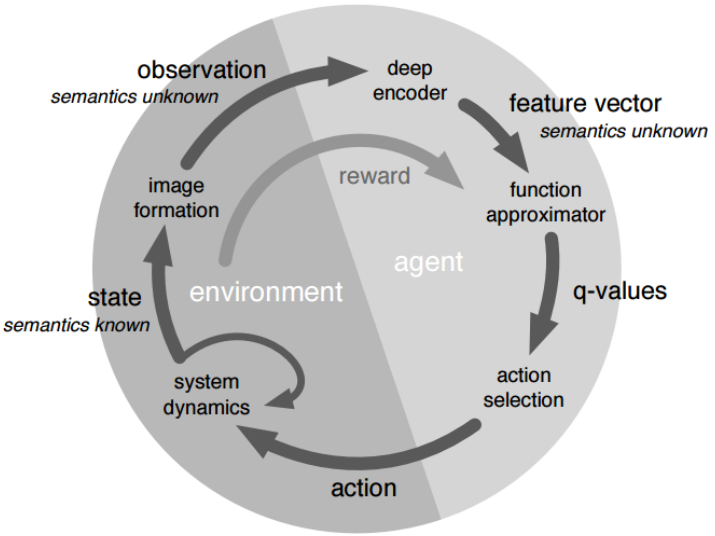
Fig. 1. Main loop of NFQ

Figure from paper

Deep Auto-Encoder NN in RL

- This paper tries to solve visual reinforcement learning problems e.g. simple mazes.
- Propose to use deep auto-encoder nn's to to obtain low dimensional feature space by extracting representative features from states.
- Then, uses kernel based approximators e.g. FQI (Fitted Q Iterations) to approximate Q-function over feature vectors outputted by DAE.





Figures from paper

Investigating contingency awareness using Atari 2600 games

- Contingency awareness is the recognition that a future observation is under an agent's control and not solely determined by the environment.
- The contingent regions of an observation are the components whose value is dependent on the most recent choice of action.
- They tries to learn the contingent regions in game play with a contingency learning method.
- Contingency learning method is a **logistic regression classifier** that can predict whether or not a pixel belongs to the contingent regions within an arbitrary Atari 2600 game

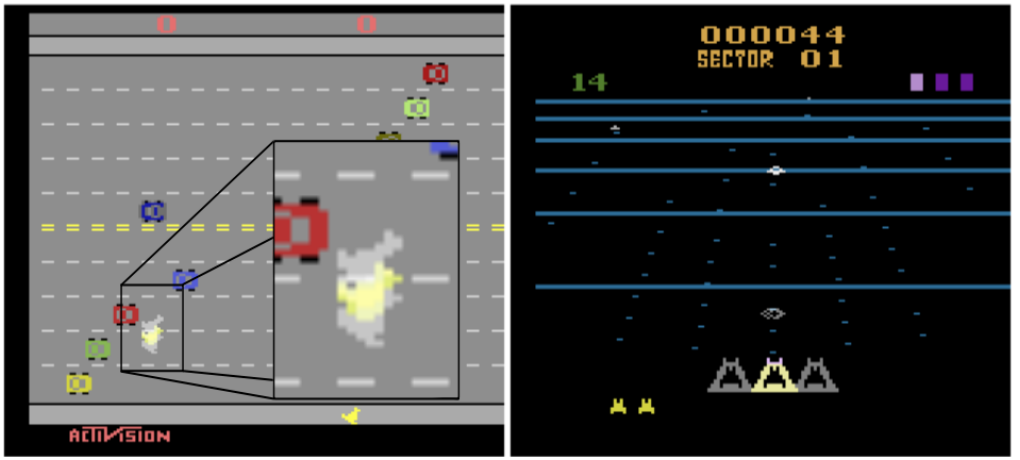


Figure from paper

The arcade learning environment: An evaluation platform for general agents

- This paper introduces ALE – Arcade Learning Environment as a testbed for RL algorithms.
- Paper is basically a benchmark paper of feature generation methods with linear function approximation method.
- Feature generation methods are:

Basic. A binary encoding of the presence of colours within the Atari 2600 screen.

BASS. The Basic method augmented with pairwise feature combinations. A restricted colour space (3-bit) is used to minimize the size of the resulting feature set.

DISCO. A heuristic object detection pipeline trained on offline data.

LSH. A method encoding the Atari 2600 screen into a small set of binary features via Locally Sensitive Hashing [Gionis *et al.*, 1999].

RAM. An encoding of the Atari 2600's 1024 bits of memory together with pairwise combinations.

Deep Q-Learning

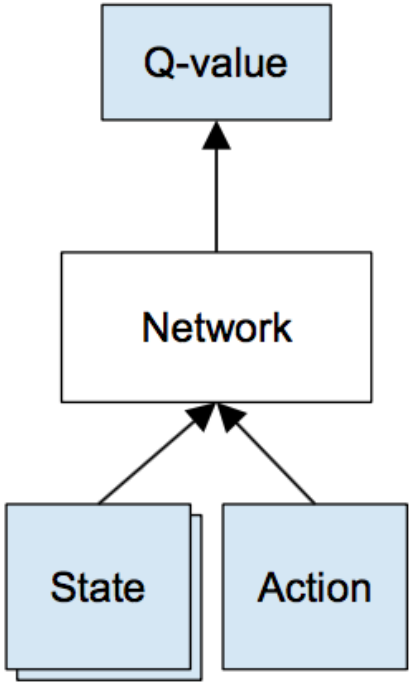
- The basic idea behind many reinforcement learning algorithms is to estimate the action-value function by using the Bellman equation as an iterative update.

$$Q_{i+1}(s,\ a) = r + \gamma\, max_{a'} Q_i(s',\ a')$$

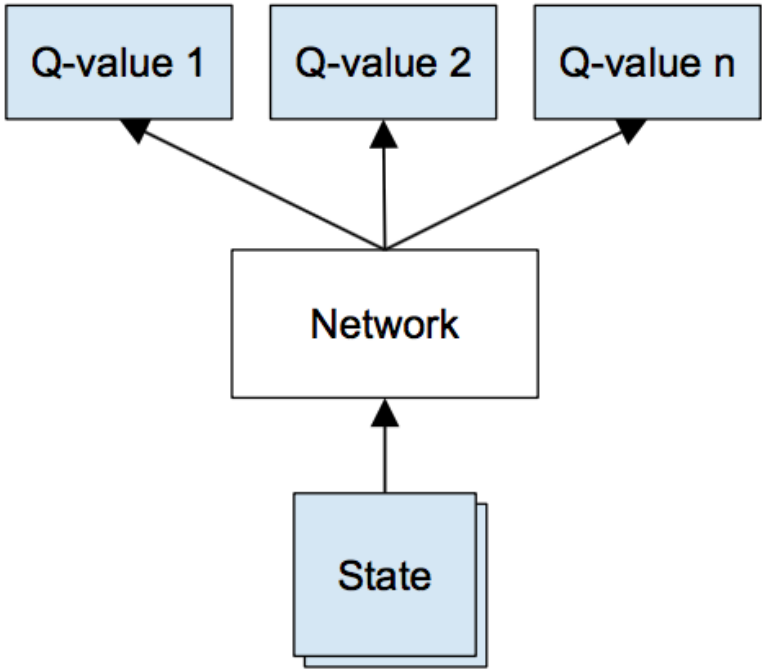
- Such value iteration algorithms converge to the optimal action-value function, $Q_{i+1} \rightarrow Q^*$ as $i \rightarrow \infty$
- In practice, this basic approach is impractical, because the action-value function is estimated separately for each sequence, without any generalization.
- It is common to use a function approximators to estimate the action-value function as linear or nonlinear function approximators e.g. neural networks.

Deep Q-Learning

- The Q-function can be approximated using a neural network model.



Naive formulation of deep Q-network.



More optimized architecture of deep Q-network, used in DeepMind paper.

Figures from link

Deep Q-Learning

- To efficiently evaluate $max_{a'} Q(s', a')$, one should use the below architecture.

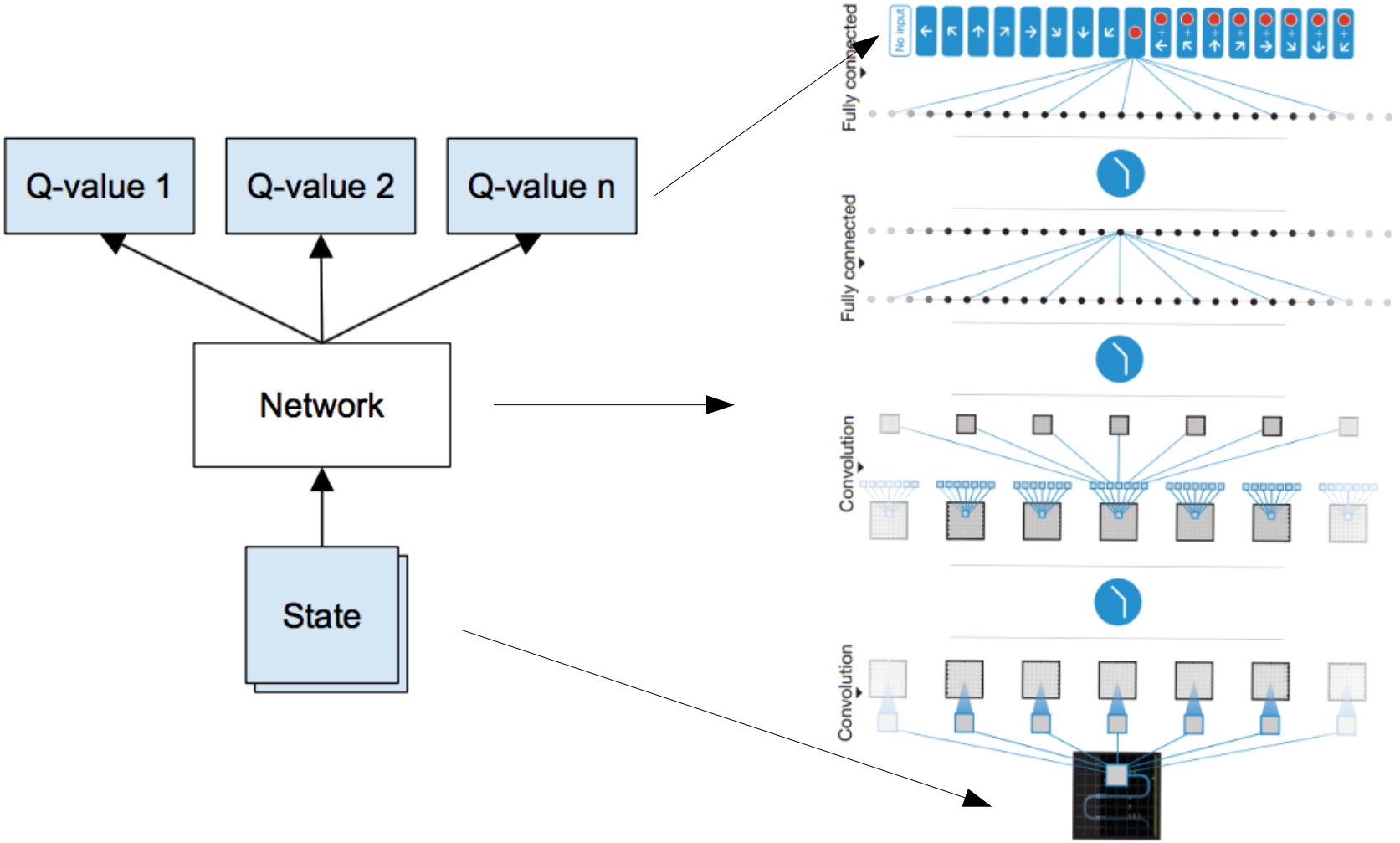


Figure from link

Figure from paper

Loss Function

Q-values can be any real values, which makes it a **regression task**, that can be optimized with simple **squared error loss**.

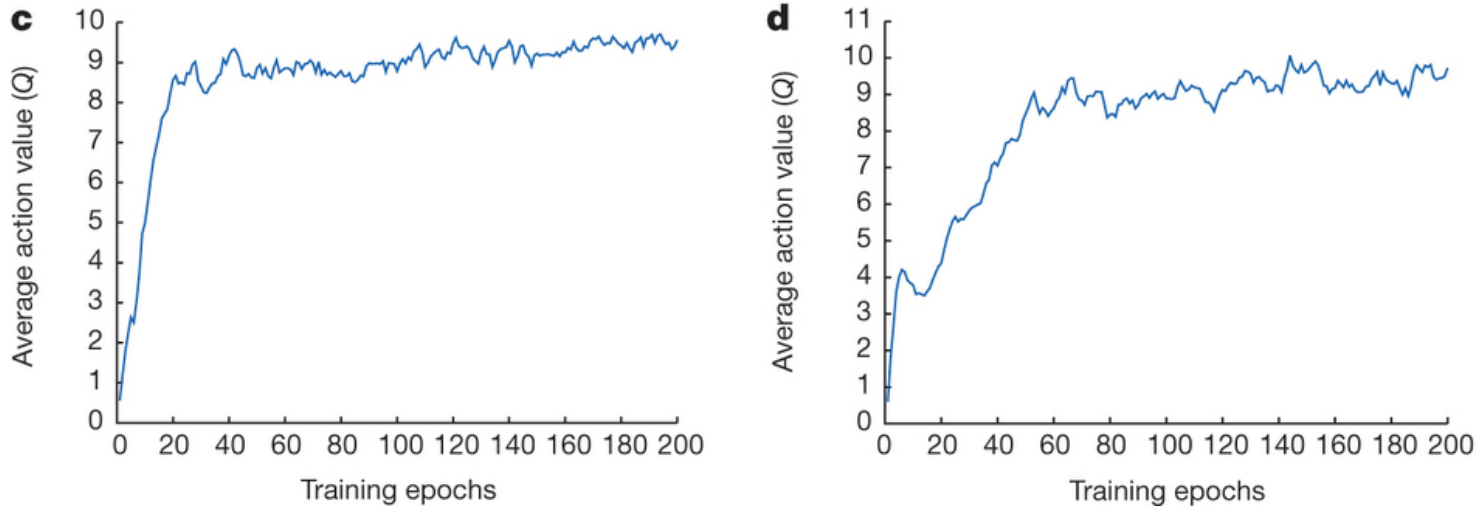
$$L_i(\theta_i) = \underbrace{\mathbb{E}_{(s,a,r,s') \sim \mathrm{U}(D)}}_{\text{Exp. Replay}} \Bigg[\Bigg(r + \gamma \underbrace{\max_{a'} Q(s', a'; \theta_i^-)}_{\text{Target}} - \underbrace{Q(s, a; \theta_i)}_{\text{Prediction}} \Bigg)^2 \Bigg]$$

- Targets depend on the network weights; this is in contrast with the targets used for supervised learning, which are fixed before learning begins.
- we hold the parameters from the previous iteration θ_i^- fixed when optimizing the *i*th loss function $L_i(\theta_i)$

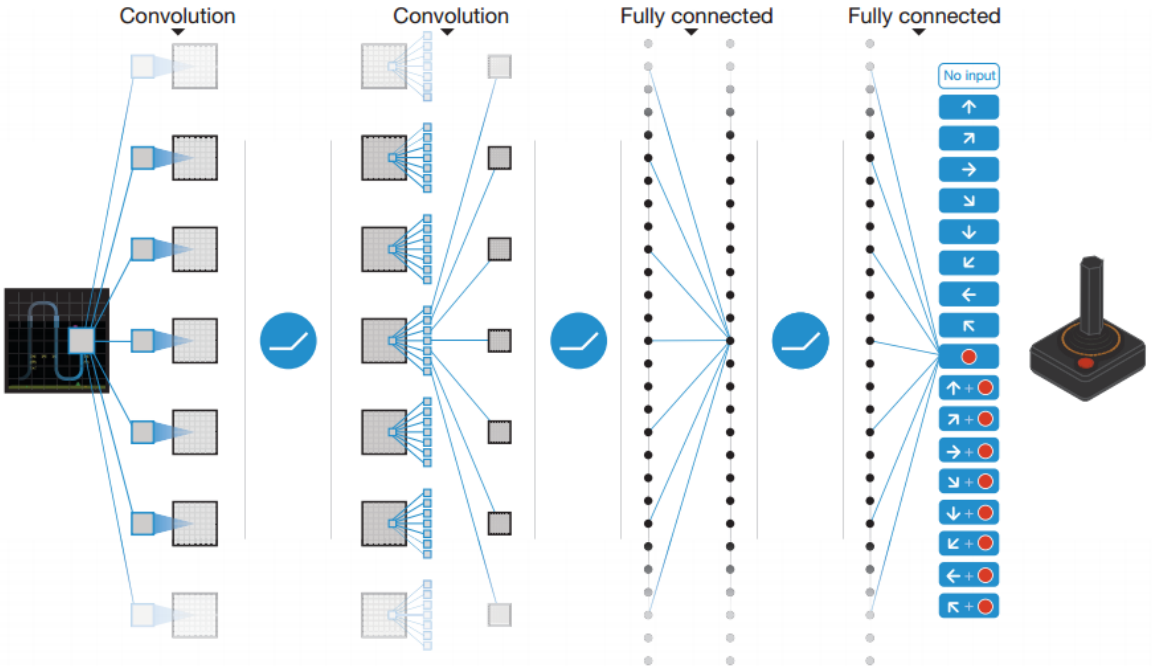
Gradient Update Rule

$$\nabla_{\theta_i} L(\theta_i) = \mathbb{E}_{s,a,r,s'}\left[\left(r + \gamma \max_{a'} Q(s',a';\theta_i^-) - Q(s,a;\theta_i)\right)\nabla_{\theta_i} Q(s,a;\theta_i)\right].$$

- Stochastic gradient descent was used to optimize loss function



Network Model



Layer	Input	Filter size	Stride	Num filters	Activation	Output
conv1	84x84x4	8x8	4	32	ReLU	20x20x32
conv2	20x20x32	4x4	2	64	ReLU	9x9x64
conv3	9x9x64	3x3	1	64	ReLU	7x7x64
fc4	7x7x64			512	ReLU	512
fc5	512			18	Linear	18

Figure from paper

Network Model

- Network is a classical convolutional neural network with **three convolutional layers**, followed by **two fully connected layers**.
- There are no pooling layers:
  - Pooling is for translation invariance.
  - For games, the location of the ball i.e states is crucial in determining the potential reward.
- 4 last frames, each 84x84:
  - To understand the last taken action e.g. ball speed, agent direction etc.
- Sequences of actions and observations, are input to the algorithm, which then learns game strategies depending upon these sequences.
- Discount factor γ $w$as set to 0.99 throughout
- Outputs of the network are Q-values for each possible action (18 in Atari).

Training Details

- 49 Atari 2600 games.
- A different network for each game
- **Reward clipping:**
  - As the scale of scores varies greatly from game to game, we clipped all positive rewards at 1 and all negative rewards at -1, leaving 0 rewards unchanged.
  - Clipping the rewards in this manner limits the **scale of the error derivatives** and makes it easier to use the **same learning rate** across multiple games.
  - It could affect the performance of our agent since it **can't differentiate between rewards of different magnitude.**
- Minibatch size 32.
- The behaviour policy during training was ε-greedy.
- ε decreases over time from 1 to 0.1 – in the beginning the system makes completely random moves to explore the state space maximally, and then it settles down to a fixed exploration rate.

Training Details

- Frame skipping:
  - the agent sees and selects actions on every *k*th frame instead of every frame, and its last action is repeated on skipped frames.
  - This technique allows the agent to play roughly *k* times more games without significantly increasing the runtime.

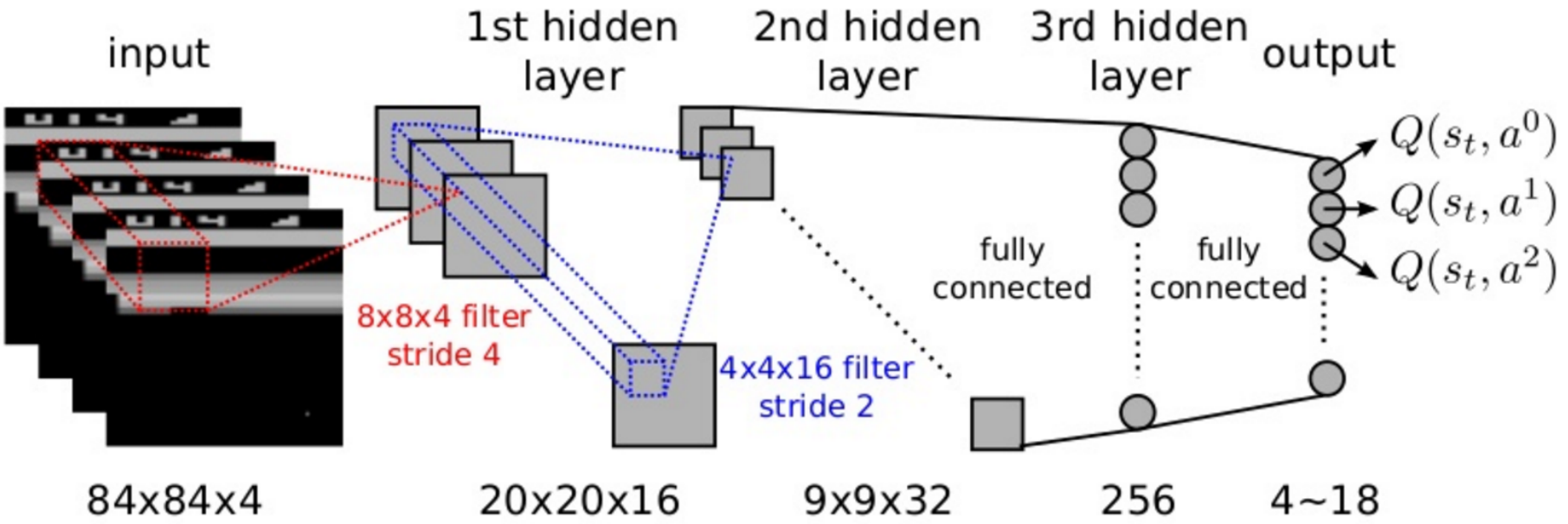


Figure from link

Unstable Non-linear Q-Learning

- **Problem:** Approximation of Q-values using non-linear functions is not very stable.
- **Reason:** Correlation present in the sequence of observations.
- **How:** Small updates to Q may significantly change the policy and data distribution.
- **Solution:** *Experience replay* to remove correlations in the ***observation.***

  Seperate target network to remove correlations with the ***target.***

Experience Replay

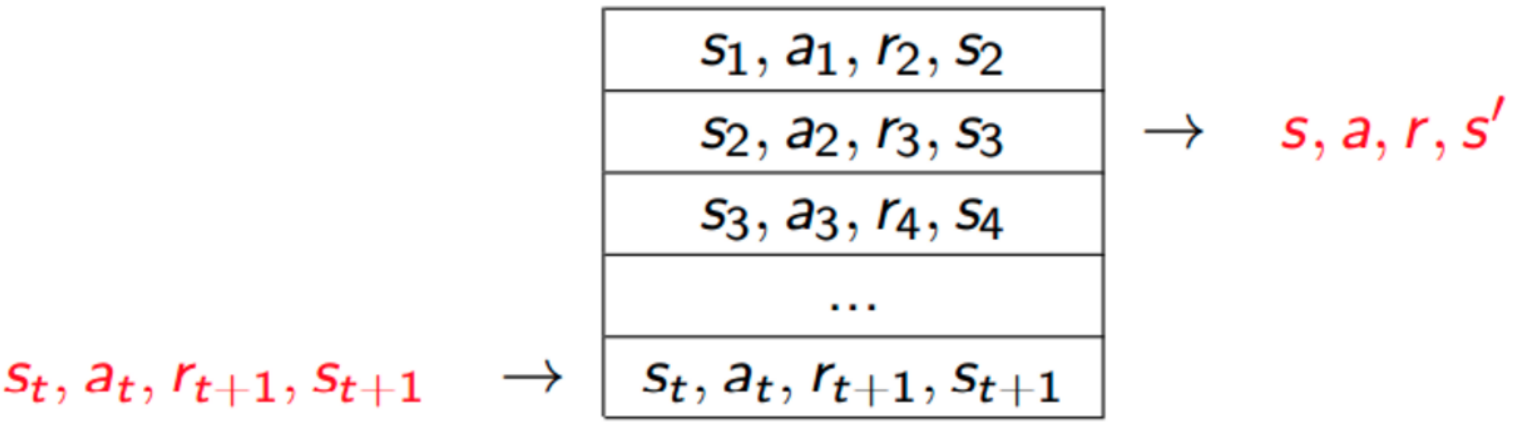
- During gameplay all the experiences $<s,a,r,s´>$ are stored in a replay memory.
- When training the network, random minibatches from the replay memory are used instead of the most recent transition.
- This breaks the similarity of subsequent training samples, which otherwise might drive the network into a local minimum.
- Experience replay makes the training task more similar to usual supervised learning, which simplifies debugging and testing the algorithm.
- One could actually collect all those experiences from human gameplay and then train network on these.

Experience Replay

- Each step of experience is potentially used in many weight updates, which allows for greater data efficiency.
- Learning directly from consecutive samples is inefficient, owing to the strong correlations between the samples.
- Randomizing the samples breaks these correlations and therefore reduces the **variance of the updates**.
- By using experience replay the behaviour distribution is averaged over many of its previous states, smoothing out learning and avoiding oscillations or divergence in the parameters.
- Algorithm only stores the last N experience tuples in the replay memory, and samples uniformly at random from D when performing updates.

Experience Replay

- To remove correlations, build data-set from agent's own experience.
- Sample experiences from data-set and apply update.



Replay Buffer – fixed size

Experience Replay - Analogy

Sequential-Correlated

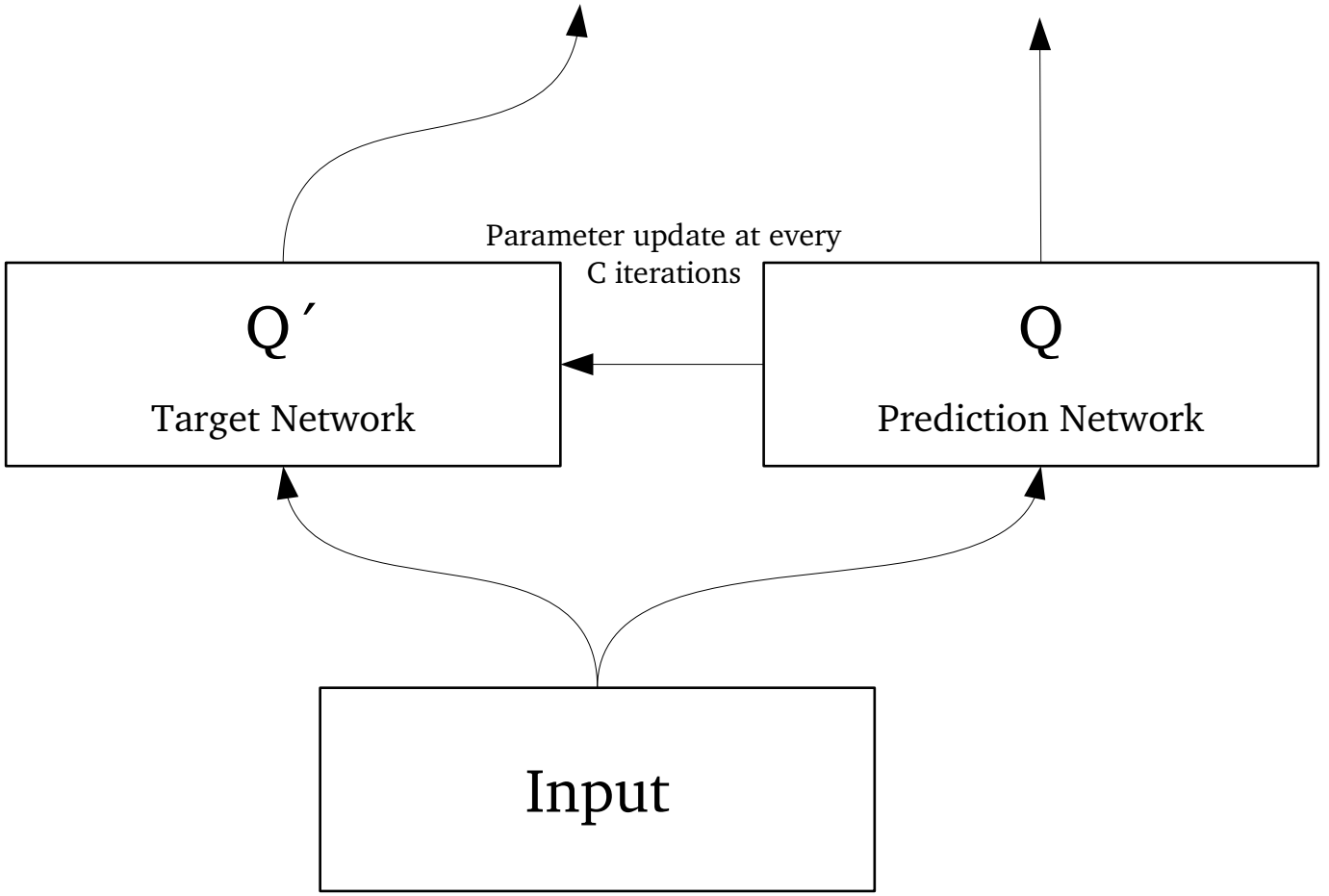
Varied Data Dist.

Seperate Target Network

- To improve the stability of method with neural networks is to use a **separate network** for generating the targets in the Q-learning update.
- Every **C updates**, clone the network Q to obtain a target network Q´ and use Q´ for generating the Q-learning targets for the following C updates to Q.
- This modification makes the algorithm **more stable** compared to standard online Q-learning
- Reduces oscillations or divergence of the policy.
- Generating the targets using an older set of parameters adds a **delay** between the time an update to Q is made and the time the update affects the targets, **making divergence or oscillations much more unlikely**.

Seperate Target Network

$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s') \sim \mathrm{U}(D)} \left[\left(r + \gamma \max_{a'} Q(s', a'; \theta_i^-) - Q(s, a; \theta_i) \right)^2 \right]$$



Exploration-Exploitation

- Firstly observe, that when a Q-table or Q-network is initialized randomly, then its predictions are initially random as well.
- If we pick an action with the highest Q-value, the action will be random and the agent performs crude “exploration”.
- As a Q-function converges, it returns more consistent Q-values and the amount of exploration decreases.
- So one could say, that Q-learning incorporates the exploration as part of the algorithm.
- A simple and effective fix for the above problem is ε-greedy exploration – with probability ε choose a random action, otherwise go with the “greedy” action with the highest Q-value.
- In their system DeepMind actually decreases ε over time from 1 to 0.1 – in the beginning the system makes completely random moves to explore the state space maximally, and then it settles down to a fixed exploration rate.

Algorithm

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 $\left(\phi_t, a_t, r_t, \phi_{t+1}\right)$ in D

Sample random minibatch of transitions $\left(\phi_j, a_j, r_j, \phi_{j+1}\right)$ 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

Experiments

- Atari 2600 platform, 49 games.
- Same network architecture for all tasks.
- **Input:** *visual images & number of actions*
- Results compared with:
  - Bellemare, M. G., Naddaf, Y., Veness, J. & Bowling, M. *The arcade learning environment: An evaluation platform for general agents.* J. Artif. Intell. Res.
  - Bellemare, M. G., Veness, J. & Bowling, M. *Investigating contingency awareness using Atari 2600 games.* Proc. Conf. AAAI. Artif. Intell.
  - Human players.

Results

- “Our DQN method outperforms the best existing reinforcement learning methods on 43 of the games without incorporating any of the additional prior knowledge about Atari 2600 games used by other approaches.”
- “Our DQN agent performed at a level that was comparable to that of a professional human games tester across the set of 49 games, achieving more than 75% of the human score on more than half of the games. (29 games)

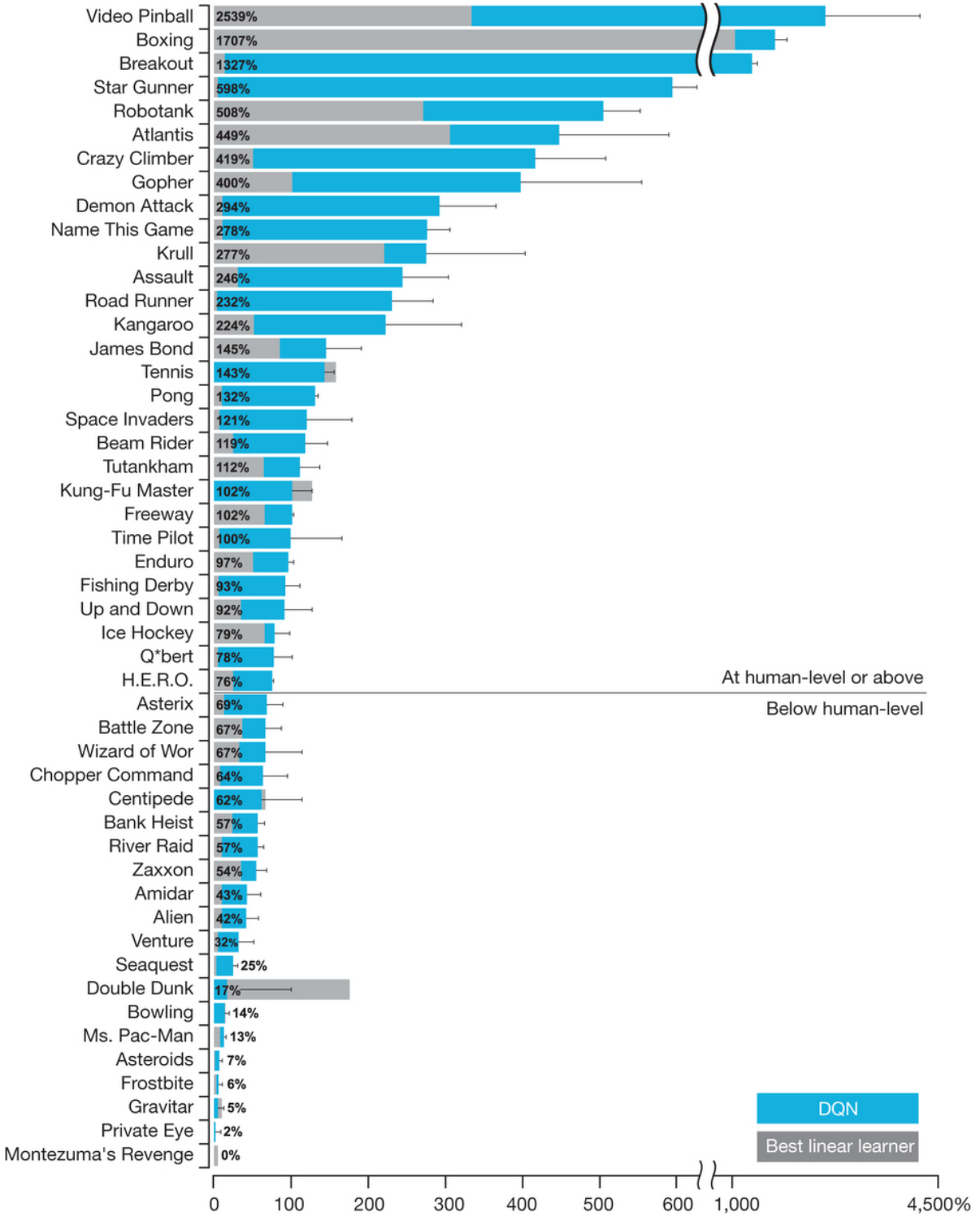
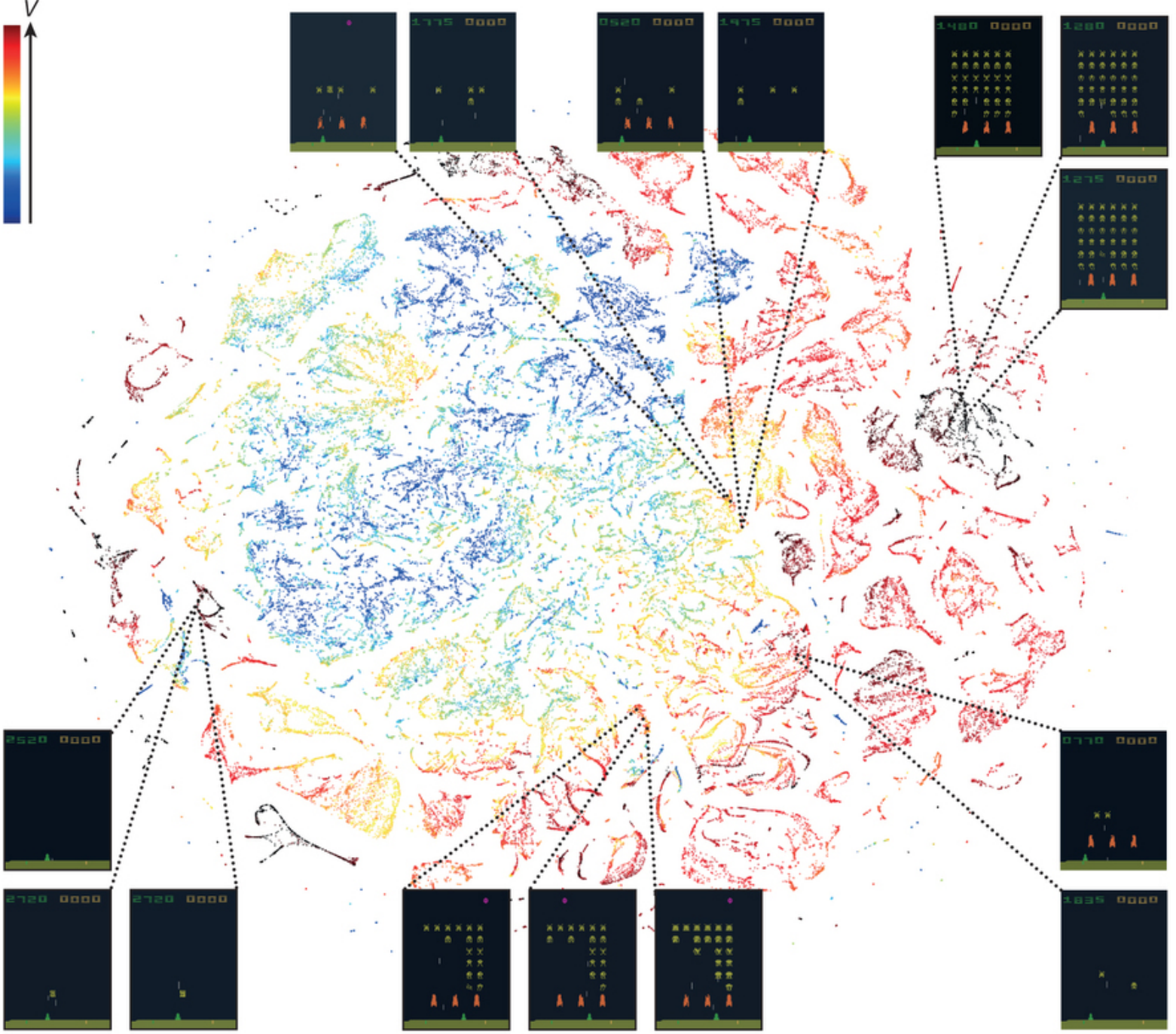


Figure from paper



States that are close in terms of **reward expectation** but perceptually **dissimilar**.

Figure from paper

Results

- In certain games DQN is able to discover a relatively long-term strategy.
  - Breakout game: first dig a tunnel around the side of the wall, send the ball back.

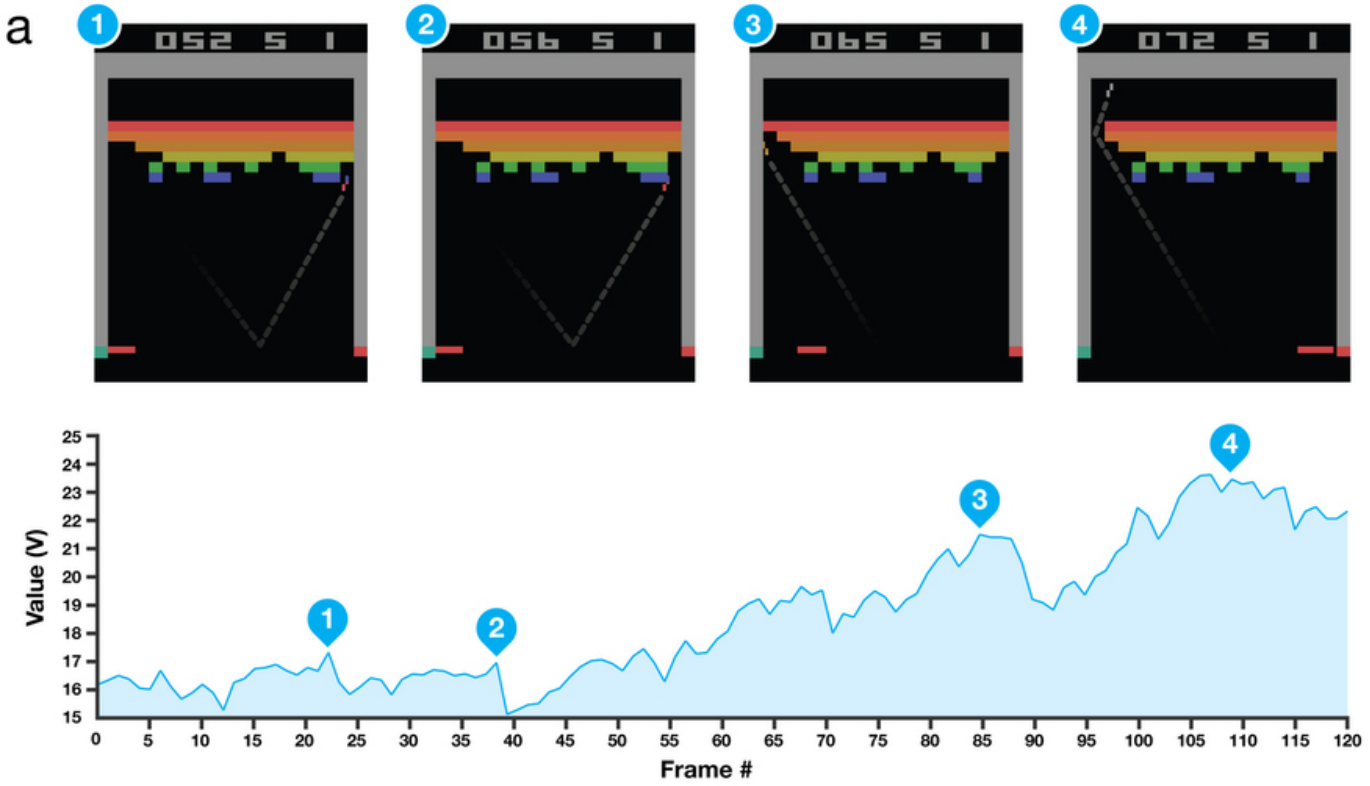


Figure from paper

Results

- Nevertheless, games demanding more temporally extended planning strategies still constitute a major challenge for all existing agents including DQN
  - Montezuma's Revenge

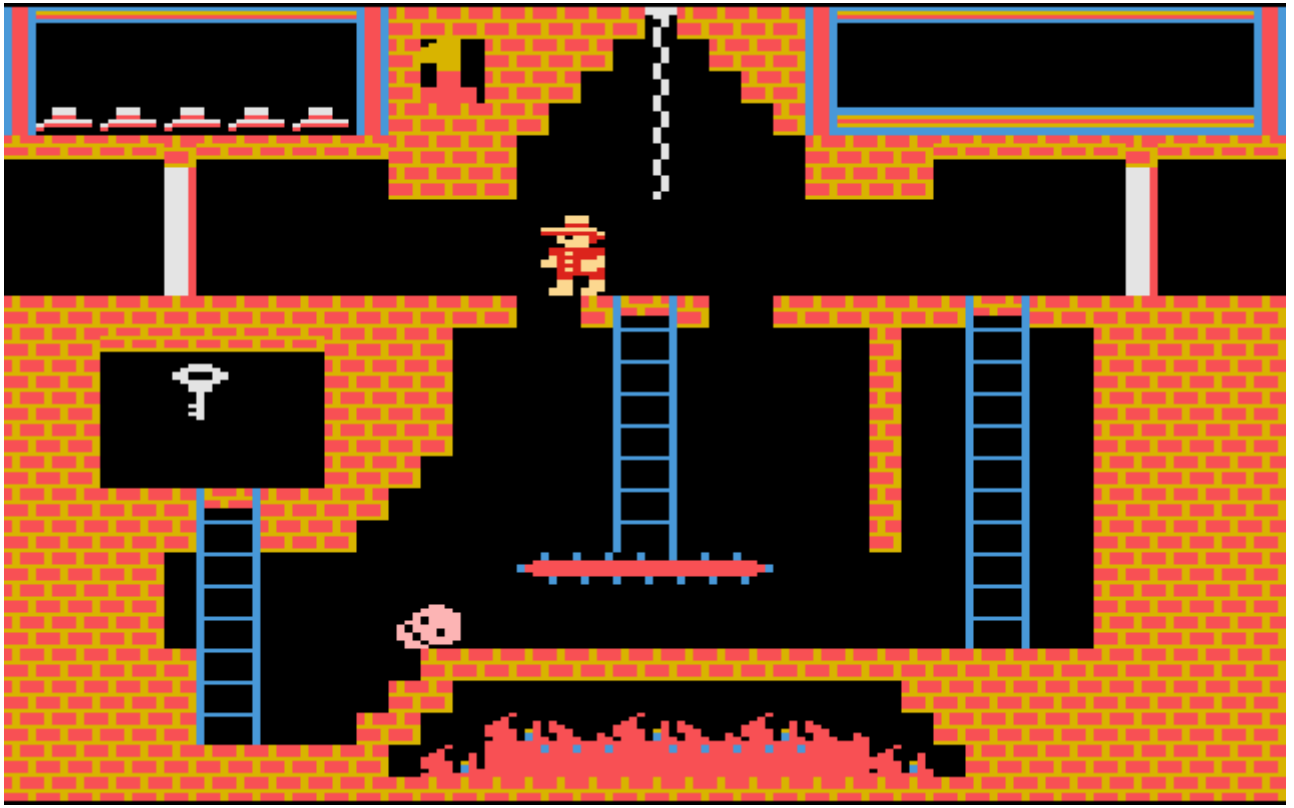


Figure from link

Results

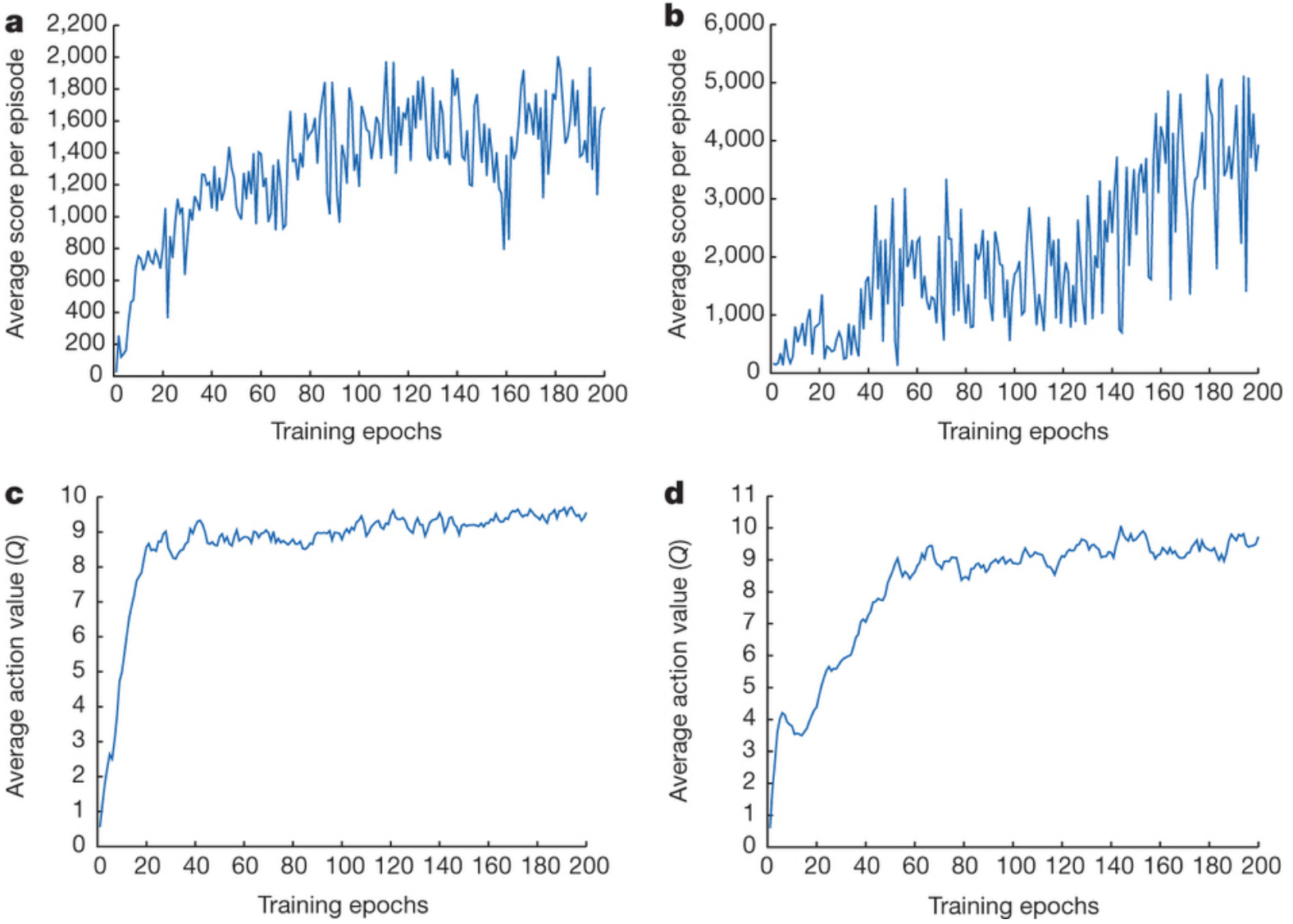


Figure from paper

Results

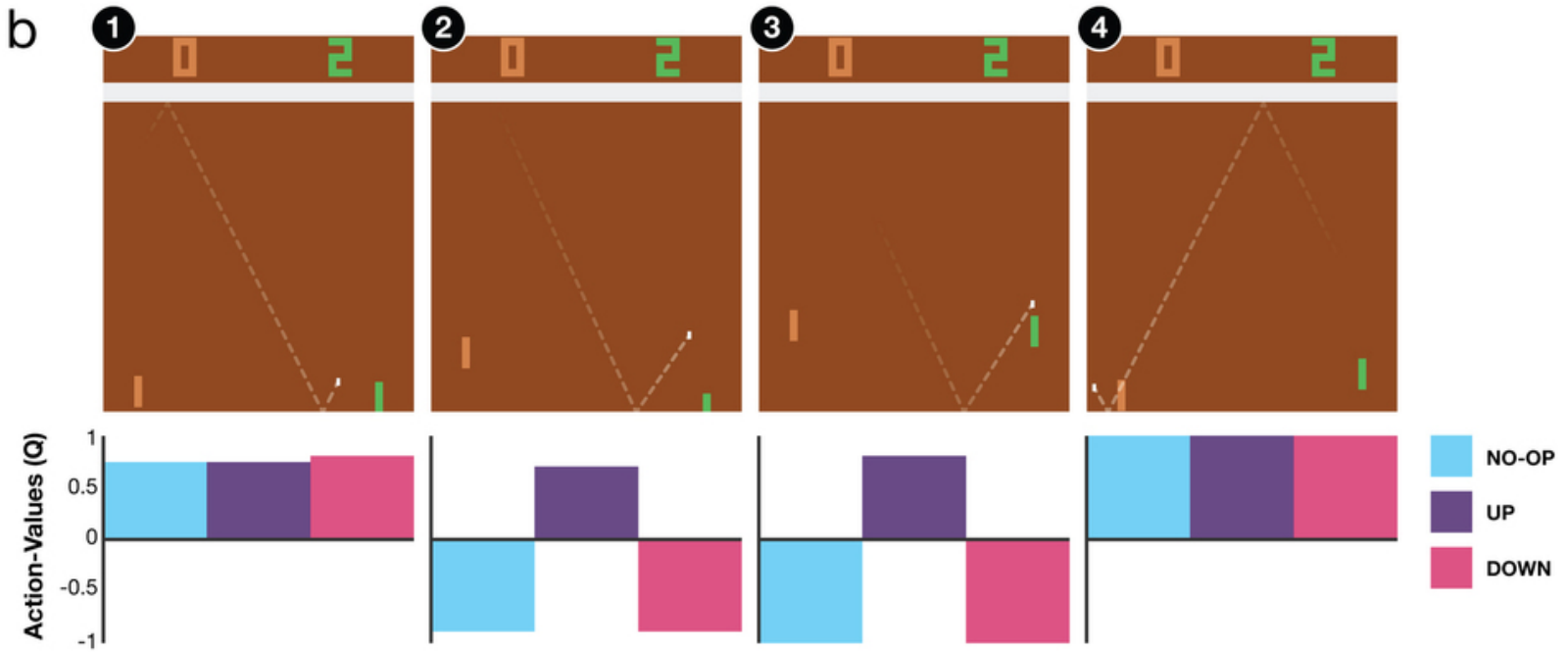


Figure from paper

Results

Extended Data Table 3 | The effects of replay and separating the target Q-network

Game	With replay, with target Q	With replay, without target Q	Without replay, with target Q	Without replay, without target Q
Breakout	316.8	240.7	10.2	3.2
Enduro	1006.3	831.4	141.9	29.1
River Raid	7446.6	4102.8	2867.7	1453.0
Seaquest	2894.4	822.6	1003.0	275.8
Space Invaders	1088.9	826.3	373.2	302.0

Figure from paper

Extended Data Table 2 | Comparison of games scores obtained by DQN agents with methods from the literature[12,15] and a professional human games tester

Game	Random Play	Best Linear Learner	Contingency (SARSA)	Human	DQN (± std)	Normalized DQN (% Human)
Alien	227.8	939.2	103.2	6875	3069 (±1093)	42.7%
Amidar	5.8	103.4	183.6	1676	739.5 (±3024)	43.9%
Assault	222.4	628	537	1496	3359(±775)	246.2%
Asterix	210	987.3	1332	8503	6012 (±1744)	70.0%
Asteroids	719.1	907.3	89	13157	1629 (±542)	7.3%
Atlantis	12850	62687	852.9	29028	85641(±17600)	449.9%
Bank Heist	14.2	190.8	67.4	734.4	429.7 (±650)	57.7%
Battle Zone	2360	15820	16.2	37800	26300 (±7725)	67.6%
Beam Rider	363.9	929.4	1743	5775	6846 (±1619)	119.8%
Bowling	23.1	43.9	36.4	154.8	42.4 (±88)	14.7%
Boxing	0.1	44	9.8	4.3	71.8 (±8.4)	1707.9%
Breakout	1.7	5.2	6.1	31.8	401.2 (±26.9)	1327.2%
Centipede	2091	8803	4647	11963	8309(±5237)	63.0%
Chopper Command	811	1582	16.9	9882	6687 (±2916)	64.8%
Crazy Climber	10781	23411	149.8	35411	114103 (±22797)	419.5%
Demon Attack	152.1	520.5	0	3401	9711 (±2406)	294.2%
Double Dunk	-18.6	-13.1	-16	-15.5	-18.1 (±2.6)	17.1%
Enduro	0	129.1	159.4	309.6	301.8 (±24.6)	97.5%
Fishing Derby	-91.7	-89.5	-85.1	5.5	-0.8 (±19.0)	93.5%
Freeway	0	19.1	19.7	29.6	30.3 (±0.7)	102.4%
Frostbite	65.2	216.9	180.9	4335	328.3 (±250.5)	6.2%
Gopher	257.6	1288	2368	2321	8520 (±3279)	400.4%
Gravitar	173	387.7	429	2672	306.7 (±223.9)	5.3%
H.E.R.O.	1027	6459	7295	25763	19950 (±158)	76.5%

Figure from paper

Conclusion

- Single architecture can successfully learn control policies in a range of different environments
  - Minimal prior knowledge,
  - Only pixels and game score as input,
  - Same algorithm,
  - Same architecture,
  - Same hyperparameters,
  - “Just like a human player”

Thank you!

? / + / -