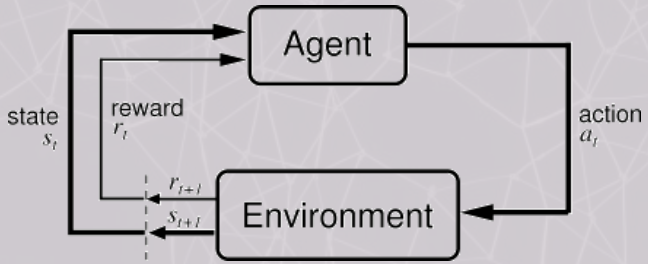
Artificial Intelligence: Playing ATARI with Deep Reinforcement Learning

Demis Hassabis (co-founder of DeepMind)

https://youtu.be/rbsqaJwpu6A

Preliminaries: Reinforcement Learning & Q-Learning



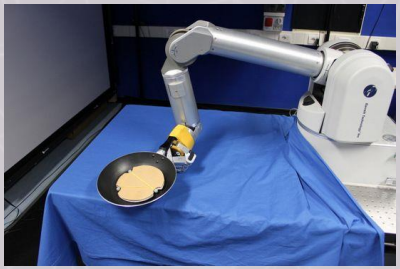
- Reinforcement learning (RL) is centered around the premise of a subject learning from interacting with its surroundings. More formally, an *agent* in (or observing) a particular *state* at time *t:* s_t , performs *action* a_t, receiving a *reward*, r_t.
- Importantly, RL is different from supervised learning. Here there is no 'external supervisor' or labeled data *per se*; instead, an agent is tasked with learning from its own experience.
- Accordingly, there exists a fundamental tradeoff in the domain of RL wherein the agent must choose between exploration and exploitation; in some instances the agent must *exploit* what it already knows in order to obtain a reward – but it must, nevertheless, also *explore* its environment in order to make better action selections in the future. This basic tension makes RL both challenging and potent as a paradigm in AI.
- Another key feature of RL is that it explicitly considers the *whole* problem of a goal-oriented agent interacting with an uncertain (moreover: stochastic) environment. By contrast, oftentimes supervised learning problems *implicitly* focus on the solution of a sub-problem (without specification of how the solution to the sub-problem is useful, holistically).

Preliminaries: Reinforcement Learning & Q-Learning (cont'd)

Some Elements of RL

- In addition to the agent and environment, there are (4) key ingredients to RL:
- (1) A **policy** (usually denoted π) is a mapping, $\pi: S \rightarrow A$. The policy is the decision-making function for the agent.
- (2) A **reward function** maps each perceived state (or state-action pair) of the environment to its corresponding reward value: $r: S \times A \rightarrow \mathbb{R}; r(s_t, a_t) = r_t$. Most often the reward function is unknown to the agent.
- (3) A **value function** specifies what is 'good' in the long-run for the agent. In this sense, the value of a state is the total amount of reward an agent can expect to accumulate over the future, starting from that state.

  Formally, $V^{\pi}(s) = E\left[\sum_{i=0}^{\infty} \gamma^i r_{t+i+1} \middle| s_t = s\right]$, with 'discount' $0 \leq \gamma \leq 1$.
- (4) A **model** of the environment, consisting of the triple: $(S, A, \delta: S \times A \rightarrow S)$. Where S is the 'state space' (assumed finite), A is the 'action space' (normatively, $|A| \ll |S|$) and δ is the transition function inherent to the environment.
- Put simply, the goal of the learning task is to learn an optimal policy, π^*, that maximizes $V^{\pi}(s)$ for all states.



Preliminaries: Reinforcement Learning & Q-Learning (cont'd II)

- At first blush, it seems as though the best strategy for learning the optimal policy, π^*, might be to directly learn $V^*(s)$.
- To do so, we could solve the recurrence relation:

$$V^{\pi}(s) = E\left[\sum_{i=0}^{\infty} \gamma^i r_{t+i+1} \middle| s_t = s\right] = \ldots = \sum_a \underbrace{\pi(s,a)}_{\substack{\text{prob. of taking} \\ \text{action } a \text{ in state } s}} \sum_{s'} \underbrace{P_{ss'}^a}_{\substack{\text{transition} \\ \text{probability}}} \left[R_{ss'}^a + V^{\pi}(s')\right]$$

- This equation is known as the **Bellman Equation for V^{π}**. The Bellman equation averages over all the possibilities, weighting each by its probability of occurring. It states that the value of the start state must equal the (discounted) value of the expected next state, plus the reward expected along the way.
- When the agent has **complete knowledge** of its environment, then, in theory, we could directly solve (or even approximate) for the value function.
- In this fashion, the optimal policy would be defined as follows: $\pi^*(s) = \arg\max_a \left[r(s,a) + \gamma V^*(\delta(s,a))\right]$, so that action a is chosen in such a way so as to maximize the sum of the immediate reward and the discounted expected reward for the successor state.
- The problem, however, with this approach is that it requires full knowledge of the transition and reward functions which are unknown to the agent.
- Alternatively, a common approach for learning an optimal policy in RL with incomplete environmental knowledge is to use **temporal-difference learning** (TD), including **Q learning**.

Preliminaries: (II) Reinforcement Learning & Q-Learning (cont'd III)

- Define the *value* of the state-action pair as the mapping: $Q : S \times A \to \mathbb{R}$; let $Q^{\pi}(s,a) = E\left[\sum_{i=0}^{\infty} \gamma^{i} r_{t+i+1} \middle| s_t = s, a_t = a\right]$.
- Q^{π} denotes the action-value for policy π, which is to say, the *expected return* starting from state s, taking action a and following policy π henceforth.
- One-step Q learning is defined as the following iterative update:

$$Q(s_t,a_t) \leftarrow \underbrace{Q(s_t,a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \left(r_{t+1} + \gamma \cdot \underbrace{\max_a Q(s_{t+1},a)}_{\substack{\text{estimate of optimal} \\ \text{future value}}} - \underbrace{Q(s_t,a_t)}_{\text{old value}} \right)$$

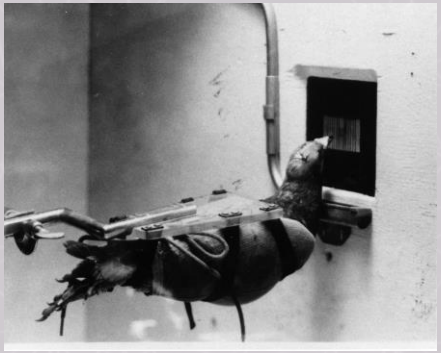
- Here the learned action-value function, Q, directly approximates Q*, the optimal action-value function, independent of the policy being followed (we say the method is 'off-policy').
- Q learning algorithm schematic:

```
Initialize Q(s, a) arbitrarily
Repeat (for each episode):
    Initialize s
    Repeat (for each step of episode):
        Choose a from s using policy derived from Q (e.g., ε-greedy)
        Take action a, observe r, s'
        Q(s, a) ← Q(s, a) + α[r + γ max_a' Q(s', a') − Q(s, a)]
        s ← s';
    until s is terminal
```

- Famously, Tesauro (early 1990s) developed **TD-Gammon** using Q-learning (computed with NN equipped with 50 hidden units; trained originally on 300,00 games against itself); by 1995 TD-Gammon was competitive with best human players in the world.

Human-level control through deep reinforcement learning

- RL theory is rooted in psychological and neuroscientific perspectives on animal behavior, particularly instances in which an agent optimizes their control of/interaction with an environment.



- Future breakthroughs in '**strong AI**' will conceivably necessitate the coordination of techniques in reinforcement learning and high-dimensional, hierarchical sensory processing systems.
- Despite a myriad of impressive results due to RL in the past several decades, to date, most RL applications have been, nonetheless, **limited to domains** where features are **handcrafted**, domains that are **full-observed** and or regimes with **low-dimensional** state spaces.
- Hassabis *et al*. (Google DeepMind) develop a novel artificial agent, termed a 'Q-network', that can learn successful policies directly from high-dimensional sensory inputs using end-to-end reinforcement learning.

Human-level control through deep reinforcement learning (cont'd)

- The test domain chosen for this novel, state-of-the-art AI agent is, naturally, classic ATARI games!

- Crucially, this problem domain is very challenging because it involves high-dimensional data, an incomplete environment knowledge-base (plus the environment is non-deterministic), minimal prior information, and the absence of intuitive/hand-crafted features.

- Moreover, the developers used a **single algorithm** (with shared parameters) to build an agent with a wide range of competencies on a varied range of challenging tasks – a central goal of AI.

Human-level control through deep reinforcement learning (cont'd II)

- Methodological details
- **Preprocessing/data**: Atari 2600 frames are of dimension 210x160 pixels, over a palette of 128 'vibrant & ultra-realistic' colors.
- A pooling over colors was performed to obviate 'flickering' issues; 'luminance' channel (i.e. the brightness/intensity of an image) was rescaled to 84x84; frames were stacked (by fours) when fed into the algorithm; notation: $\phi(s_t)$ denotes this pre-processing applied to the sequence of images s_t.

Human-level control through deep reinforcement learning (cont'd III)

- Methodological details

- **Model Architecture**: The researchers developed a novel agent, a **deep Q-network** (*DQN*) which is able to combine RL with 'deep' NNs (that is to say they have many layers).

- Notably, recent research [see Bengio, for instance] has revealed that it is possible for deep NNs to 'learn'/infer progressively more abstract representations of data.

- Hassabis *et al*. use a deep CNN, which employs hierarchical layers of tiled convolutional filters to mimic the effects of 'receptive fields'.

- The goal of the game-playing agent is to select actions in a fashion that maximizes cumulative feature rewards. Formally, the deep CNN is used to approximate the optimal action-value, which is the maximum sum of discounted rewards achievable by a behavior policy $\pi = P(a \mid s)$:

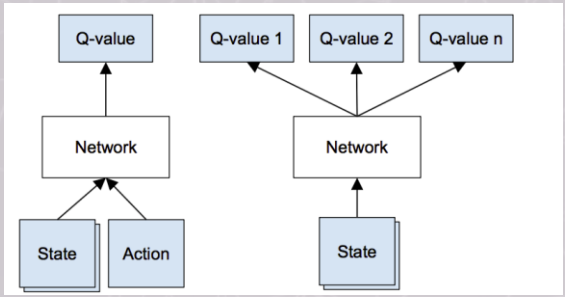
$$Q^*(s,a) = \max_{\pi} E\left[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots \middle| s_t = s, a_t = a, \pi \right]$$

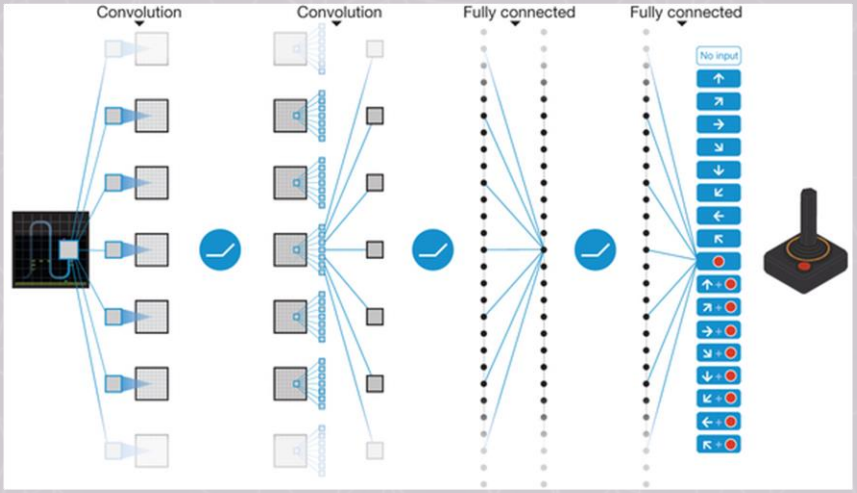
- RL is known to be **numerically unstable** when NNs are used to approximate Q functions; this is largely due to (2) issues: (1) many data sequences of state action pairs are highly correlated; (2) minute updates in Q-value approximations can significantly impact the behavior of an optimal policy.

- To get around these potential shortcomings, the authors propose: (1) the use of 'experience replay' that randomizes over the data and thus removes many data correlations; and (2) the use of iterative updates to Q-values that are only periodically updated.

Human-level control through deep reinforcement learning (cont'd IV)

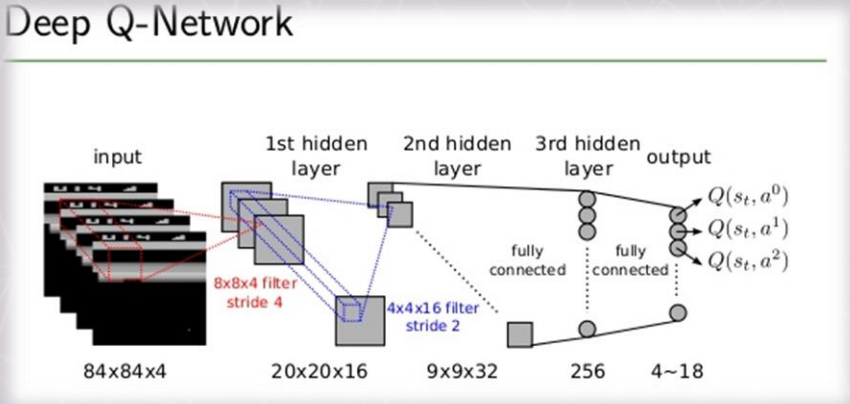
- **Model Architecture (cont'd)**:
- In previous approaches, researchers applied NNs to approximate Q-values using histories + actions as inputs to the NN. This scheme presents a significant drawback, however, since a separate forward pass is required to compute the Q-value for each individual action.
- Instead, in the current method, the outputs correspond to the predicted Q-values of the individual actions for the input state. This presents a significant computational advantage over previous methods; Q-values are accordingly computed for **all possible actions** in a given state with only a single forward pass through the network.





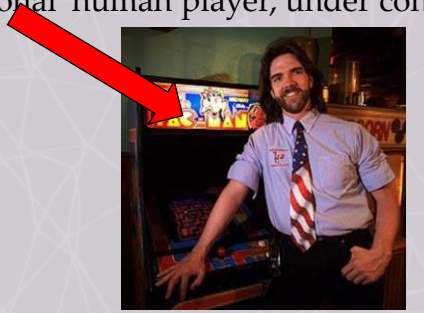
Human-level control through deep reinforcement learning (cont'd V)

- **Model Architecture (cont'd)**:
- The input to the NN consists of an 84x84x4 image produced by the preprocessing map.
- The first hidden layer convolves 32 filters of size 8x8 with stride 4 and applies a RELU.
- The second hidden layer convolves 64 filters of size 4x4 with stride 2, again followed by a RELU.
- The third hidden layer convolves 64 filters of size 3x3 with stride 1, with RELU.
- The final hidden layer is fully-connected and consists of 512 rectifier units. The output layer is a fully-connected layer with an output for each action. The number of valid actions varies between 4 and 18 in the games considered.



Human-level control through deep reinforcement learning (cont'd VI)

- **Training Details:**
- Experiments were performed across 49 games; a different network was trained on each game. Nevertheless, the same architecture, learning algorithm and hyperparameters were used across all games.
- Rewards were based on in-game scoring and were 'clipped' in order to reasonably limit the scale of derivatives.
- A gradient-based backprop (with minibatches of size 32) was used to train the CNN.
- The behavior policy during training was ε–greedy with ε annealed linearly from 1.0 to 0.1.
- **Evaluation Details:**
- Trained agents were evaluated by playing each game 30 times for up to 5 minutes each time, with different random initial conditions and an ε–greedy policy with ε=.05; a 'random' agent was provided as a baseline.
- In addition to the learned agents, scores were also reported for a 'professional' human player, under controlled conditions (*e.g.* same emulator).



Human-level control through deep reinforcement learning (cont'd VII)

- **Algorithm Details:**
- Sequences of actions and observations, $s_t = x_1, a_1, x_2, \ldots, a_{t-1}, x_t$, are input to the algorithm, which then learns game strategies depending upon these sequences.
- This formalism gives rise to a large (but finite) Markov decision process (*MDP*).
- The optimal action-value function in this setting obeys the aforementioned Bellman equation. Normally, where possible, one would use the Bellman equation as an iterative update for the action-value approximation.
- In practice for large sequence MDPs, this approach is impractical because it requires estimating the action-value function for each sequence separately, without any generalization.
- Alternatively, the authors use a NN, *viz.*, a Q-Network (with parameter set θ) for the approximation: $Q(s,a;\theta) \approx Q^*(s,a)$.
- Note that without an efficient state-action value approximation, the number of action pair values is astronomically large ($\sim 10^{67970}$)!
- The Q-Network is trained by adjusting the parameters θ_i at each iteration to reduce the MSE in the Bellman equation, this yields the loss function: $L(\theta_i) = E\left[\left(y - Q(s,a;\theta_i)\right)^2\right] + E\left[V[y]\right]^2, \; with: y = r + \gamma \max_{a'} Q^*(s',a';\theta_i^-)$
- Differentiating this loss function wrt the weights yields a gradient used in stochastic gradient descent. Note that state-action sequences are generated off-policy; the behavior distribution is ε-greedy.

Human-level control through deep reinforcement learning (cont'd VIII)

- **Putting it all together…**
- The agent selects and executes actions according to an ε–greedy policy based on Q. The Q-function works on fixed length representations of histories produced by the pre-processing function ϕ.
- The algorithm modifies standard online Q-learning in (2) ways to make it suitable for training a large NN.
- (1) The authors employ a technique called 'experience replay', in which the agent's experiences at each time step $e_t=(s_t,a_t,r_t,s_{t+1})$ are stored in a data set $D_t=\{e_1,\ldots,e_t\}$ pooled over many episodes.
- During the inner loop of the algorithm the authors apply Q-learning updates to samples of experience, (s,a,r,s')~U(D), drawn at random from the pool of stored samples. (this improves data efficiency and reduces correlations between samples and the presence of feedback loops in the in training process).
- By using experience replay, the behavior distribution is averaged over many of its previous states, thereby smoothing out learning and avoiding oscillations or avoidance in the parameters.
- Note that the uniform sampling gives equal importance to all transitions in the replay memory (a possible improvement would be to apply a more sophisticated sampling strategy similar to prioritized sweeping).
- (2) To further improve stability, a separate network for generating the targets (y_i's) in the Q-learning update. More precisely, every C updates the authors cloned the network Q to obtain a target network that is used for generating Q-learning targets for the following C updates to Q.

Human-level control through deep reinforcement learning (cont'd IX)

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

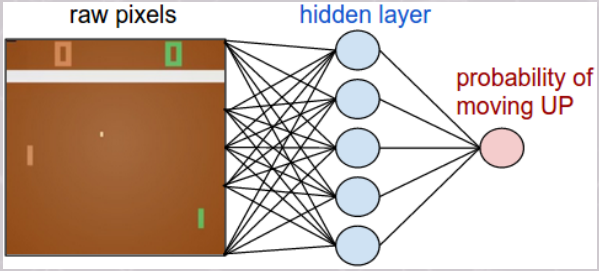
    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step } j+1 \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^{-}) & \text{otherwise} \end{cases}$

    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 $\hat{Q} = Q$

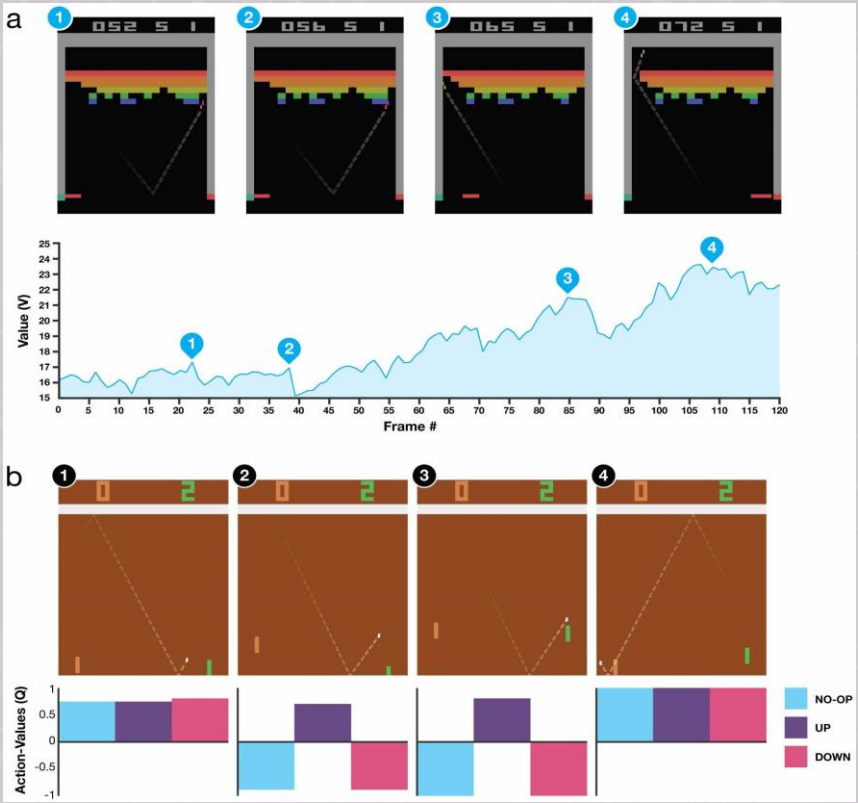
  End For

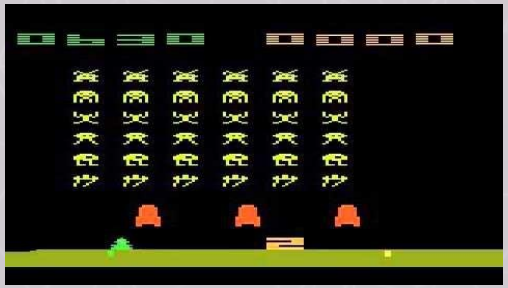
End For



Human-level control through deep reinforcement learning (cont'd X)

- Visualization of learned value functions on two games.



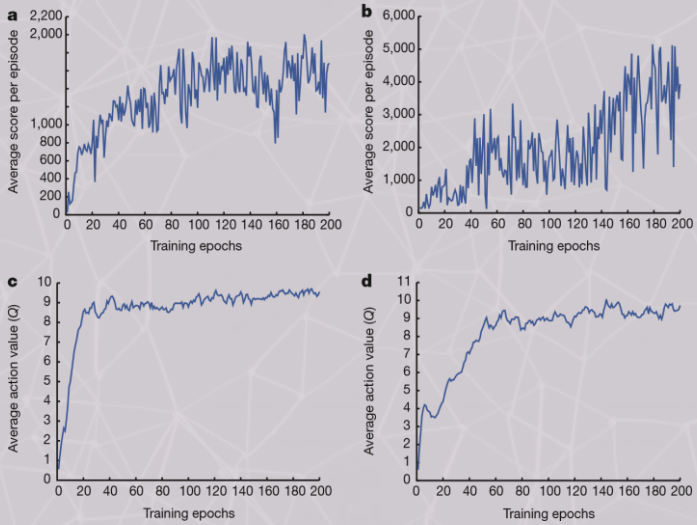


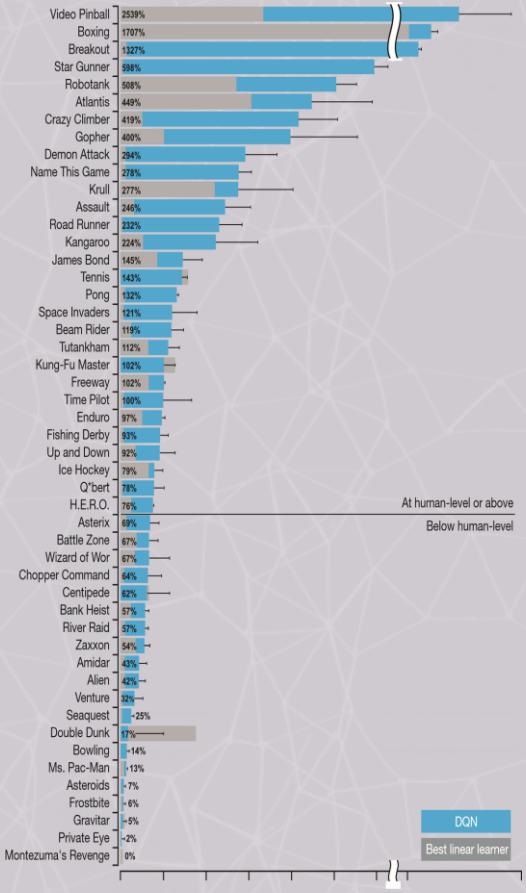
Human-level control through deep reinforcement learning (cont'd XI)

- **Results**

- The DQN agent performed at a level comparable to that of a professional human games test across the set of 49 games, achieving more than 75% of the human score on more than half the games.

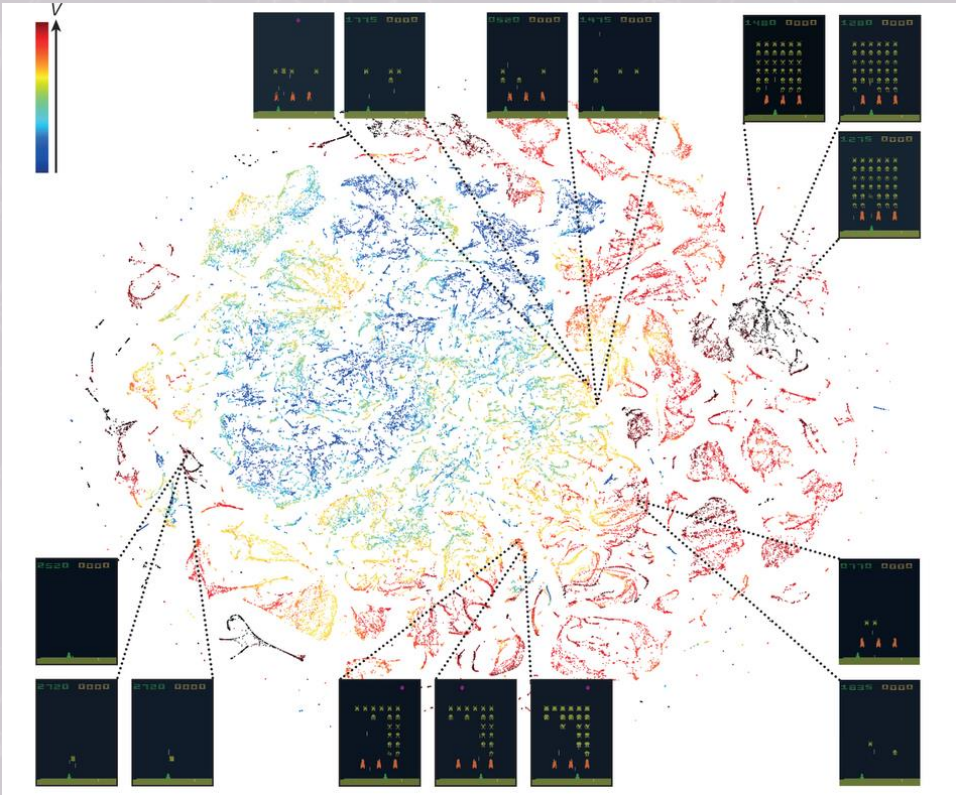
- The authors' method was able to train large NNs using RL with stochastic gradient descent in a stable manner – illustrated by the temporal evolution of two indices of learning (the agent's average score-per-episode and average predicted Q-values).





Human-level control through deep reinforcement learning (cont'd XII)

- 2-D t-SNE embedding of the representations in the last hidden layer assigned by DQN to game states experienced while playing Space Invaders. Points are colored according to state values predicted by DQN for the corresponding game states (dark red: high V, dark blue: low V).



Human-level control through deep reinforcement learning (cont'd XIII)

- **Conclusion**:
- The authors demonstrate that a **single architecture** and **single algorithm** can successfully learn control policies in a rage of different environments.
- Input only consisted of scores and pixels, prior knowledge and hand-crafted features were essentially non-existent; no internal knowledge of the game algorithm itself was utilized; environment model is thus unknown and effectively non-deterministic.
- RL was used end-to-end for training and evaluation.
- Methodology is rooted in neurobiological evidence that reward signals during perceptual learning may influence the characteristics of representations within primate visual cortex.
- The successful integration of RL with deep network architectures was critically dependent on incorporate of a **replay algorithm** which involves the storage and representation of recently experienced transitions.
- This work reveals the potential of marrying cutting-edge ML techniques with biologically-inspired mechanisms to create agents capable of learning to master a multitude of diverse and challenging tasks.
- **Some discussion questions**:
- (*) How does this work compare (as an achievement in the domain of AI), with previous landmark results and models, *e.g.* Deep Blue, Watson, AlphaGo? While these other models are conventionally considered as solving 'narrow' AI tasks, can the same be said for the current paper?
- (*) Does the algorithm's ability to 'play' (some would consider play synonymous with instantiations of creativity) shed any light on AI-based 'creativity'?
- (*) Can the model be said to 'learn' the rules/objectives of the game in a way comparable to our own understanding?
- (*) Do you consider this research a milestone in the development of AI?
- (*) For students familiar with these structures, do you think the use of an RNN/LSTM model might be beneficial for this paradigm?