Neural Episodic Control

Alexander Pritzel APRITZEL@GOOGLE.COM
Benigno Uria BURIA@GOOGLE.COM
Sriram Srinivasan SRSRINIVASAN@GOOGLE.COM
Adrià Puigdomènech ADRIAP@GOOGLE.COM
Oriol Vinyals VINYALS@GOOGLE.COM
Demis Hassabis DEMISHASSABIS@GOOGLE.COM
Daan Wierstra WIERSTRA@GOOGLE.COM
Charles Blundell CBLUNDELL@GOOGLE.COM
DeepMind, London UK

170419
Song, Junho

Abstract

Deep reinforcement learning methods attain super-human performance in a wide range of environments. Such methods are grossly inefficient, often taking orders of magnitudes more data than humans to achieve reasonable performance. We propose Neural Episodic Control: a deep reinforcement learning agent that is able to rapidly assimilate new experiences and act upon them. Our agent uses a semi-tabular representation of the value function: a buffer of past experience containing slowly changing state representations and rapidly updated estimates of the value function. We show across a wide range of environments that our agent learns significantly faster than other state-of-the-art, general purpose deep reinforcement learning agents.

1. introduction

Atari 2600 set of environments (Bellemare et al., 2013), deep Q-networks (Mnih et al., 2016) require more than 200 hours of gameplay in order to achieve scores similar to those a human player achieves after two hours (Lake et al., 2016).

problem with training time and process in RL

1. introduction

focus on addressing these 3 issues

1. Stochastic gradient descent optimisation requires the use of small learning rates. Due to the global approximation nature of neural networks, high learning rates cause catastrophic interference (McCloskey & Cohen, 1989). Low learning rates mean that experience can only be incorporated into a neural network slowly.

2. Environments with a sparse reward signal can be difficult for a neural network to model as there may be very few instances where the reward is non-zero. This can be viewed as a form of class imbalance where low-reward samples outnumber high-reward samples by an unknown number. Consequently, the neural network disproportionately underperforms at predicting larger rewards, making it difficult for an agent to take the most rewarding actions.

3. Reward signal propagation by value-bootstrapping techniques, such as Q-learning, results in reward information being propagated one step at a time through the history of previous interactions with the environment. This can be fairly efficient if updates happen in reverse order in which the transitions occur. However, in order to train on uncorrelated minibatches DQN-style, algorithms train on randomly selected transitions, and, in order to further stabilise training, require the use of a slowly updating *target network* further slowing down reward propagation.

1. introduction

In this work we shall focus on addressing the three concerns listed above; we must note, however, that other recent advances in exploration (Osband et al., 2016), hierarchical reinforcement learning (Vezhnevets et al., 2016) and transfer learning (Rusu et al., 2016; Fernando et al., 2017) also

Deep Exploration via Bootstrapped DQN

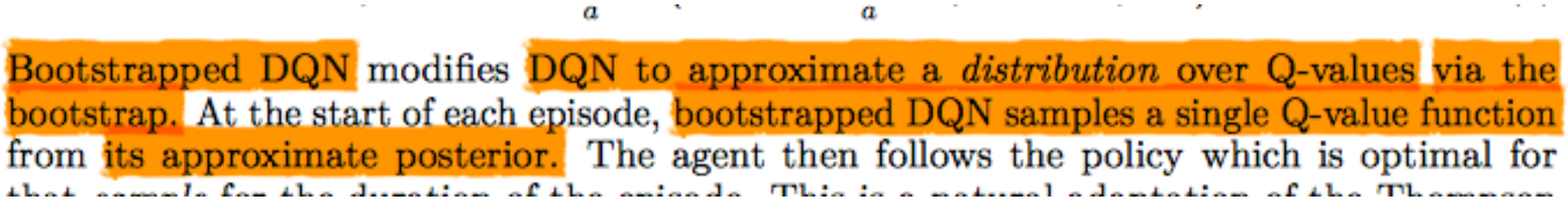
Ian Osband[1,2], **Charles Blundell**[2], **Alexander Pritzel**[2], **Benjamin Van Roy**[1]
[1]Stanford University, [2]Google DeepMind
{iosband, cblundell, apritzel}@google.com, bvr@stanford.edu

Abstract

Efficient exploration remains a major challenge for reinforcement learning (RL). Common dithering strategies for exploration, such as ϵ-greedy, do not carry out temporally-extended (or deep) exploration; this can lead to exponentially larger data requirements. However, most algorithms for statistically efficient RL are not computationally tractable in complex environments. Randomized value functions offer a promising approach to efficient exploration with generalization, but existing algorithms are not compatible with nonlinearly parameterized value functions. As a first step towards addressing such contexts we develop *bootstrapped DQN*. We demonstrate that bootstrapped DQN can combine deep exploration with deep neural networks for exponentially faster learning than any dithering strategy. In the Arcade Learning Environment bootstrapped DQN substantially improves learning speed and cumulative performance across most games.

1. introduction

In this work we shall focus on addressing the three concerns listed above; we must note, however, that other recent advances in exploration (Osband et al., 2016), hierarchical reinforcement learning (Vezhnevets et al., 2016) and transfer learning (Rusu et al., 2016; Fernando et al., 2017) also



Bootstrapped DQN modifies DQN to approximate a *distribution* over Q-values via the bootstrap. At the start of each episode, bootstrapped DQN samples a single Q-value function from its approximate posterior. The agent then follows the policy which is optimal for

1. introduction

In this work we shall focus on addressing the three concerns listed above; we must note, however, that other recent advances in exploration (Osband et al., 2016), hierarchical reinforcement learning (Vezhnevets et al., 2016) and transfer learning (Rusu et al., 2016; Fernando et al., 2017) also

Strategic Attentive Writer for Learning Macro-Actions

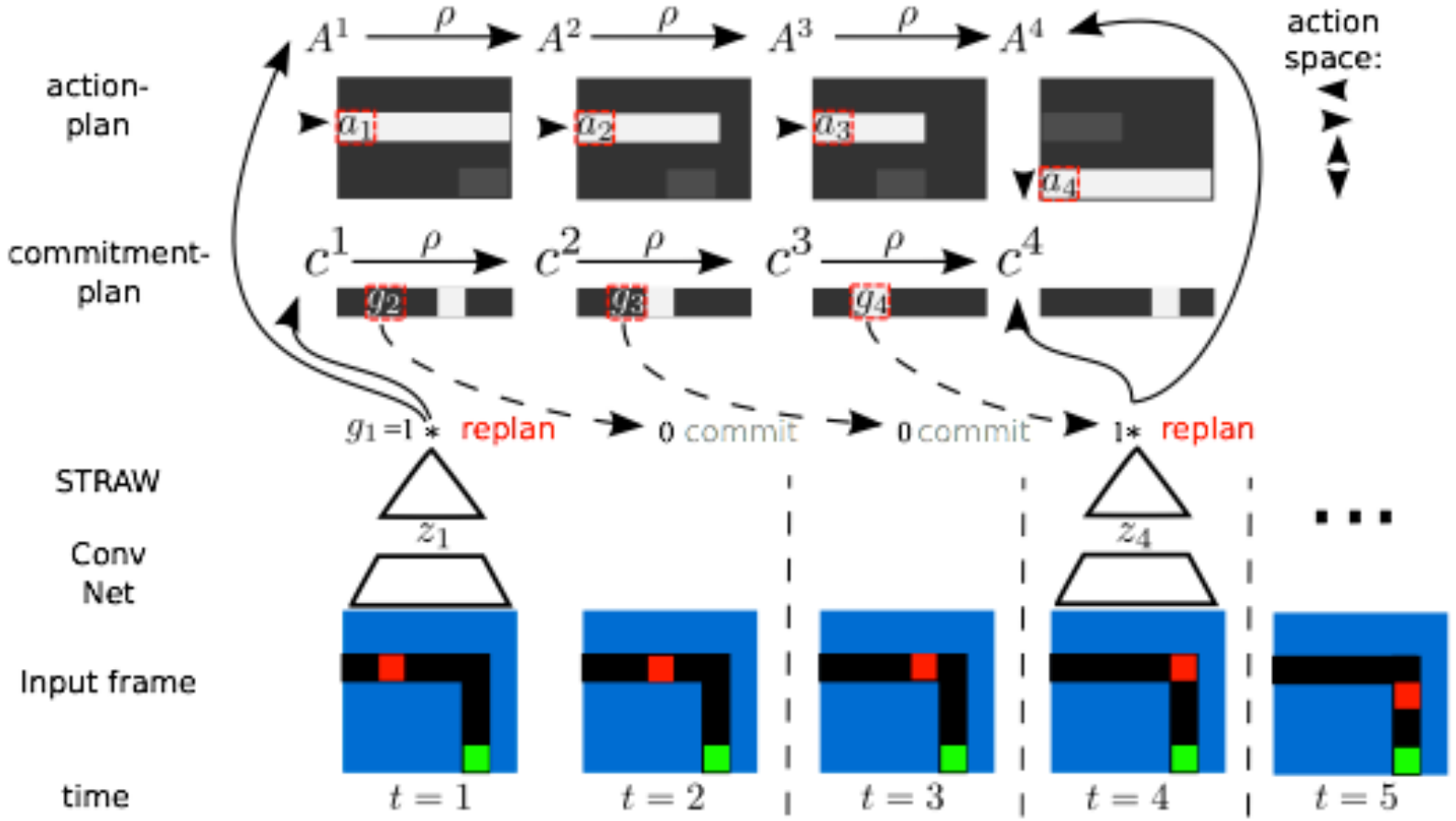
Alexander (Sasha) Vezhnevets, Volodymyr Mnih, John Agapiou, Simon Osindero, Alex Graves, Oriol Vinyals, Koray Kavukcuoglu
Google DeepMind
{vezhnick,vmnih,jagapiou,osindero,gravesa,vinyals,korayk}@google.com

Abstract

We present a novel deep recurrent neural network architecture that learns to build *implicit plans* in an end-to-end manner purely by interacting with an environment in reinforcement learning setting. The network builds an internal plan, which is continuously updated upon observation of the next input from the environment. It can also partition this internal representation into contiguous sub-sequences by learning for how long the plan can be committed to – i.e. followed without replaning. Combining these properties, the proposed model, dubbed STRategic Attentive Writer (STRAW) can learn high-level, temporally abstracted macro-actions of varying lengths that are solely learnt from data without any prior information. These macro-actions enable both structured exploration and economic computation. We experimentally demonstrate that STRAW delivers strong improvements on several ATARI games by employing temporally extended planning strategies (e.g. Ms. Pacman and Frostbite). It is at the same time a general algorithm that can be applied on any sequence data. To that end, we also show that when trained on text prediction task, STRAW naturally predicts frequent n-grams (instead of macro-actions), demonstrating the generality of the approach.

1. introduction

In this work we shall focus on addressing the three concerns listed above; we must note, however, that other recent advances in exploration (Osband et al., 2016), hierarchical reinforcement learning (Vezhnevets et al., 2016) and transfer learning (Rusu et al., 2016; Fernando et al., 2017) also



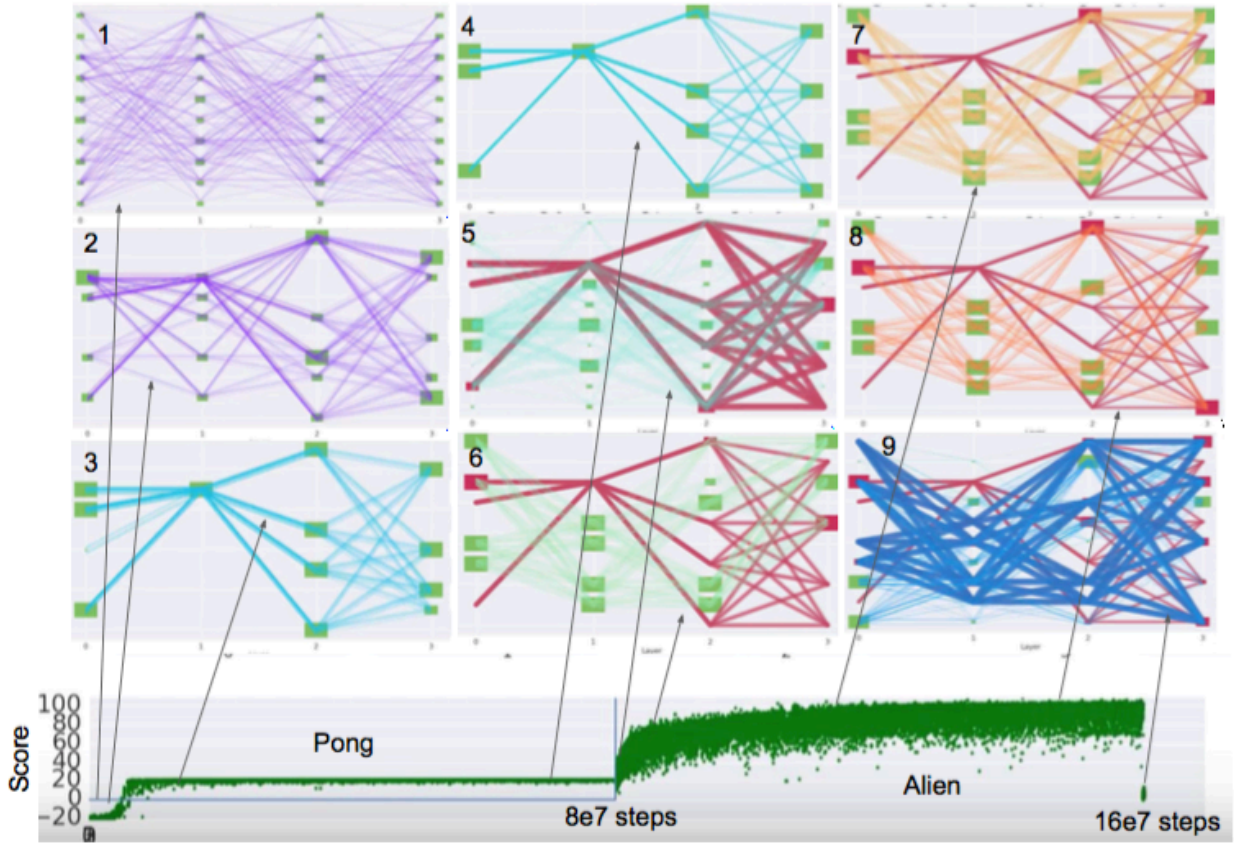
training the plaining

1. introduction

In this work we shall focus on addressing the three concerns listed above; we must note, however, that other recent advances in exploration (Osband et al., 2016), hierarchical reinforcement learning (Vezhnevets et al., 2016) and transfer learning (Rusu et al., 2016; Fernando et al., 2017) also

PathNet: Evolution Channels Gradient Descent in Super Neural Networks

Chrisantha Fernando, Dylan Banarse, Charles Blundell, Yori Zwols, David Ha†, Andrei A. Rusu, Alexander Pritzel, Daan Wierstra
Google DeepMind, London, UK. †Google Brain
chrisantha@google.com



2. Deep Reinforcement Learning

that the policy π is followed thereafter. The discount factor $\gamma \in (0, 1)$ trades off favouring short vs. long term rewards.

state s_t at step t. The agent then executes an ϵ-greedy policy based upon this value function to trade-off exploration and exploitation: with probability ϵ the agent picks an action uniformly at random, otherwise it picks the action $a_t = \arg\max_a Q(s_t, a)$.

When the agent observes a transition, DQN stores the (s_t, a_t, r_t, s_{t+1}) tuple in a *replay buffer*, the contents of which are used for training. This neural network is trained

buffer. The *target network* $\tilde{Q}(s_{t+1}, a)$ is an older version of the value network that is updated periodically. The use of a target network and uncorrelated samples from the replay buffer are critical for stable training.

A3C (Mnih et al., 2016) is another well known deep reinforcement learning algorithm that is very different from DQN. It is based upon a policy gradient, and learns both a policy and its associated value function, which is learned

3.1. Differentiable Neural Dictionary

For each action $a \in \mathcal{A}$, NEC has a simple memory module $M_a = (K_a, V_a)$, where K_a and V_a are dynamically sized arrays of vectors, each containing the same number of vectors. The memory module acts as an arbitrary association from keys to corresponding values, much like the dictionary data type found in programs. Thus we refer to this kind of memory module as a *differentiable neural dictionary* (DND). There are two operations possible on a DND: *lookup* and *write*, as depicted in Figure 1. Performing a lookup on a

$$o = \sum_i w_i v_i, \tag{1}$$

where v_i is the ith element of the array V_a and

$$w_i = k(h, h_i) / \sum_j k(h, h_j), \tag{2}$$

where h_i is the ith element of the array K_a and $k(x, y)$ is a kernel between vectors x and y, e.g., Gaussian or inverse kernels. Thus the output of a lookup in a DND is a weighted sum of the values in the memory, whose weights are given by normalised kernels between the lookup key and the corresponding key in memory. To make queries into very large memories scalable we shall make two approximations in practice: firstly, we shall limit (1) to the top p-nearest neighbours (typically $p = 50$). Secondly, we use an approximate nearest neighbours algorithm to perform the lookups, based upon kd-trees (Bentley, 1975).

memory module
- key : Ka
- corresponding value : Va
- operation : lookup and write

each action -> Va, Ka

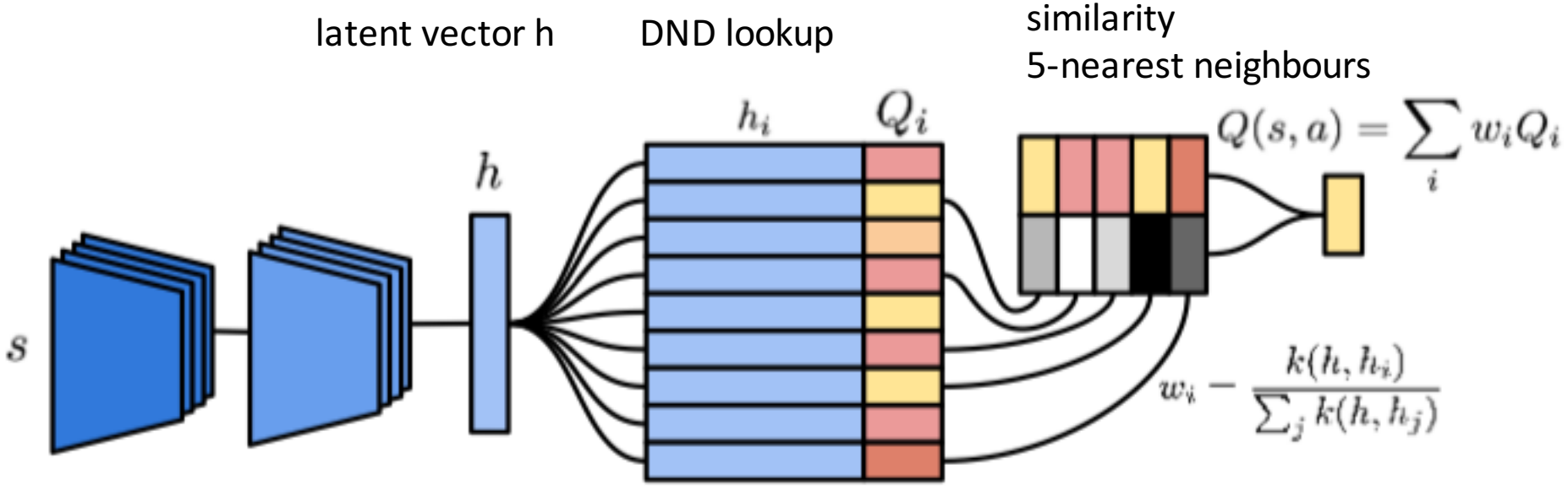
weight : normalized kernel
between lookup key and corresponding key

h

3.1. Differentiable Neural Dictionary

After a DND is queried, a new key-value pair is written into the memory. The key written corresponds to the key that was looked up. The associated value is application-specific (below we specify the update for the NEC agent). Writes to a DND are append-only: keys and values are written to the memory by appending them onto the end of the arrays K_a and V_a respectively. If a key already exists in the memory, then its corresponding value is updated, rather than being duplicated.

3.1. Differentiable Neural Dictionary



$$k(h, h_i) = \frac{1}{\|h - h_i\|_2^2 + \delta}.$$

h

3.2. Agent Architecture

Algorithm 1 Neural Episodic Control

$\mathcal{D}$: replay memory.
M_a: a DND for each action a.
N: horizon for N-step Q estimate.
for each episode **do**
  for $t = 1, 2, \ldots, T$ **do**
    Receive observation s_t from environment with embedding h.
    Estimate $Q(s_t, a)$ for each action a via (1) from M_a
    $a_t \leftarrow \epsilon$-greedy policy based on $Q(s_t, a)$
    Take action a_t, receive reward r_{t+1}
    Append $(h, Q^{(N)}(s_t, a_t))$ to M_{a_t}.
    Append $(s_t, a_t, Q^{(N)}(s_t, a_t))$ to $\mathcal{D}$.
    Train on a random minibatch from $\mathcal{D}$.
  end for
end for

h

3.3. Adding (s, a) pairs to memory

3.4. Learning

N-step Q-value estimate

$$Q^{(N)}(s_t, a) = \sum_{j=0}^{N-1} \gamma^j r_{t+j} + \gamma^N \max_{a'} Q(s_{t+N}, a') \,.$$

When a state-action value is already present in a DND (*i.e* the exact same key h is already in K_a), the corresponding value present in V_a, Q_i, is updated in the same way as the classic tabular Q-learning algorithm:

$$Q_i \leftarrow Q_i + \alpha(Q^{(N)}(s, a) - Q_i) \,.$$

h

4. Experiments

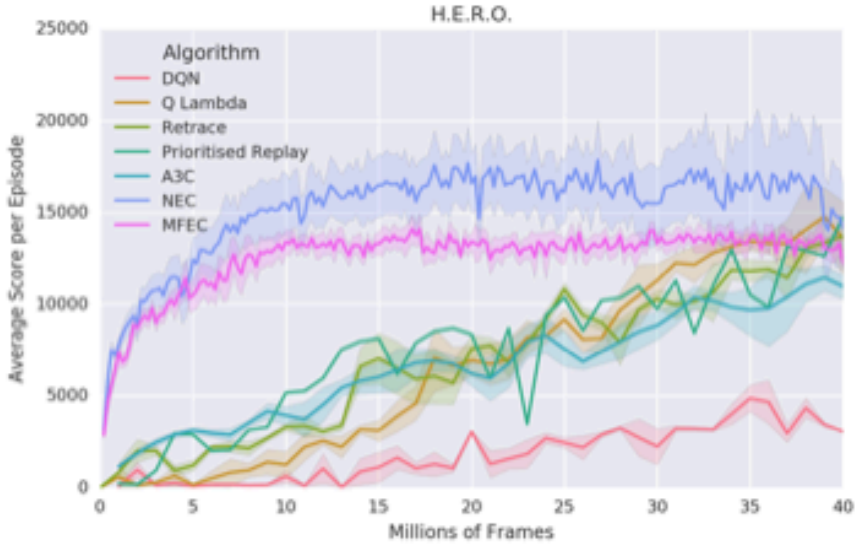


Figure 5. Learning curve on H.E.R.O.

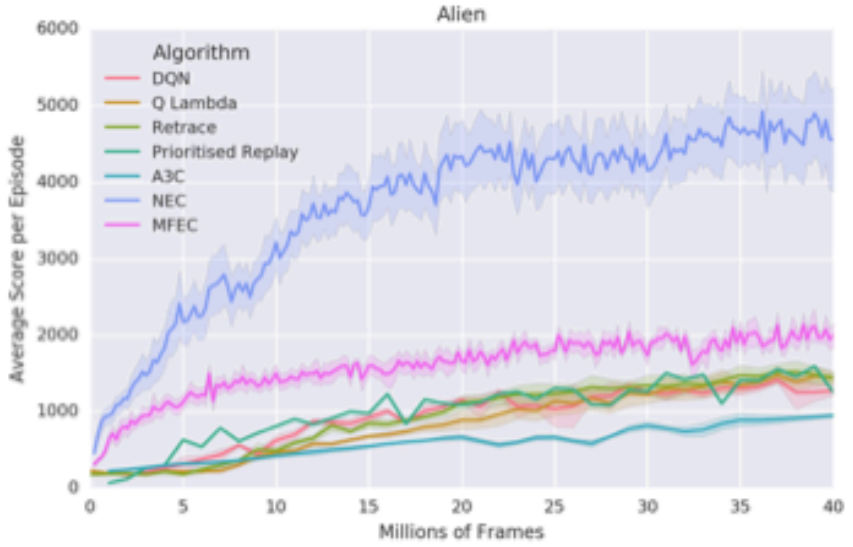
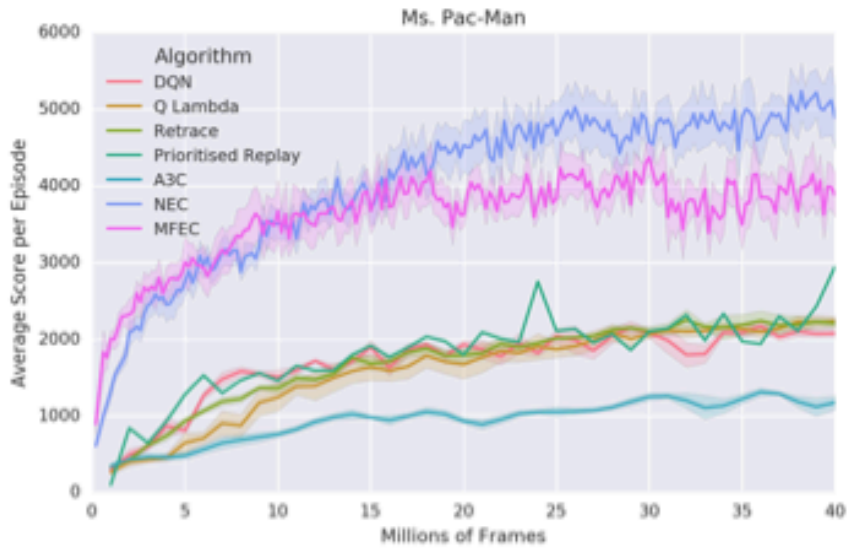
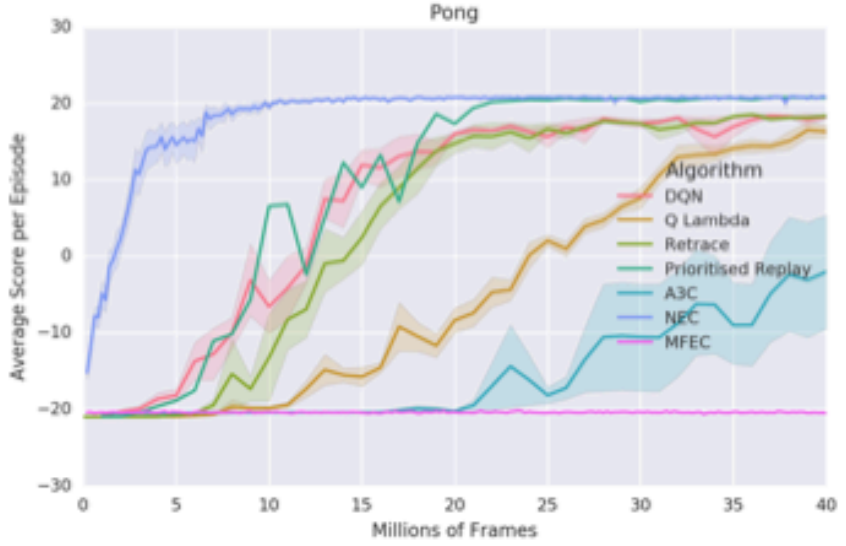


Figure 7. Learning curve on Alien.





h

summary

- Deep learning is currently hindered by the amount of data necessary to perform well

- Computational neuroscience research suggests that episodic memory may excel at selecting good actions when data are noisy or scarce

- Greedy non-parametric tabular-memory agents like MFEC can outperform model-based agents when data are noisy or scarce

- NEC outperforms MFEC by creating an end-to-end trainable learning system using differentiable neural dictionaries and a convolutional neural network

- A representation of the environment as generated by the mammalian brain's ventral stream can be approximated with random projections, a variational autoencoder, or a convolutional neural network

source：https://rylanschaeffer.github.io/content/research/neural_episodic_control/main.html