Reinforcement Learning with Unsupervised Auxiliary Tasks

Presenter: Ceyer Wakilpoor

Max Jaderberg, Volodymyr Mnih, Wojciech Marian Czarnecki, Tom Schaul, Joel Z Leibo, David Silver, and Koray Kavukcuoglu

DeepMind

Nov 2016

Outline

Outline

Introduction

- Both natural and artificial agents line in a stream of sensorimotor data
  - At each time step t, the agent receives observations o_t and executes action a_t
  - These actions influence the course of the sensorimotor stream
- Classical reinforcement learning (RL) paradigm focuses on the maximization of extrinsic reward, but in many cases there are more goals than the immediate extrinsic rewards
- This paper introduces an agent they refer to as the UNsupervised REinforcement and Auxiliary Learning agent (UNREAL)

Introduction

- Imagine a child trying to maximize the total amount of red it sees
  - The baby must understand how to increase "redness" through various means
  - This includes the ability to bring the red object close o its eyes, crying to get red items from parents, and moving towards red objects
- These ideas aren't crucial to maximizing the reward, but understanding these concepts are helpful for future tasks
- These abstractions can be reused for different tasks in the long run

Outline

Overview

- The architecture utilizes auxiliary tasks that provide pseudo rewards, leading the agent to make auxiliary predictions that serve to focus the agent on important aspects of the task
- There are replay mechanisms to provide additional updates, much like how animals and humans dream about positively and negatively rewarding events more frequently
- Both the auxiliary control and auxiliary prediction tasks share the convolutional neural network and LSTM that the base agent uses to act
  - This leads the agent to create jointly learned representations that greatly improve the policies at the end of training
- This paper brings together Asynchronous Advantage Actor-Critic (A3C) with auxiliary control tasks and auxiliary reward tasks

Overview

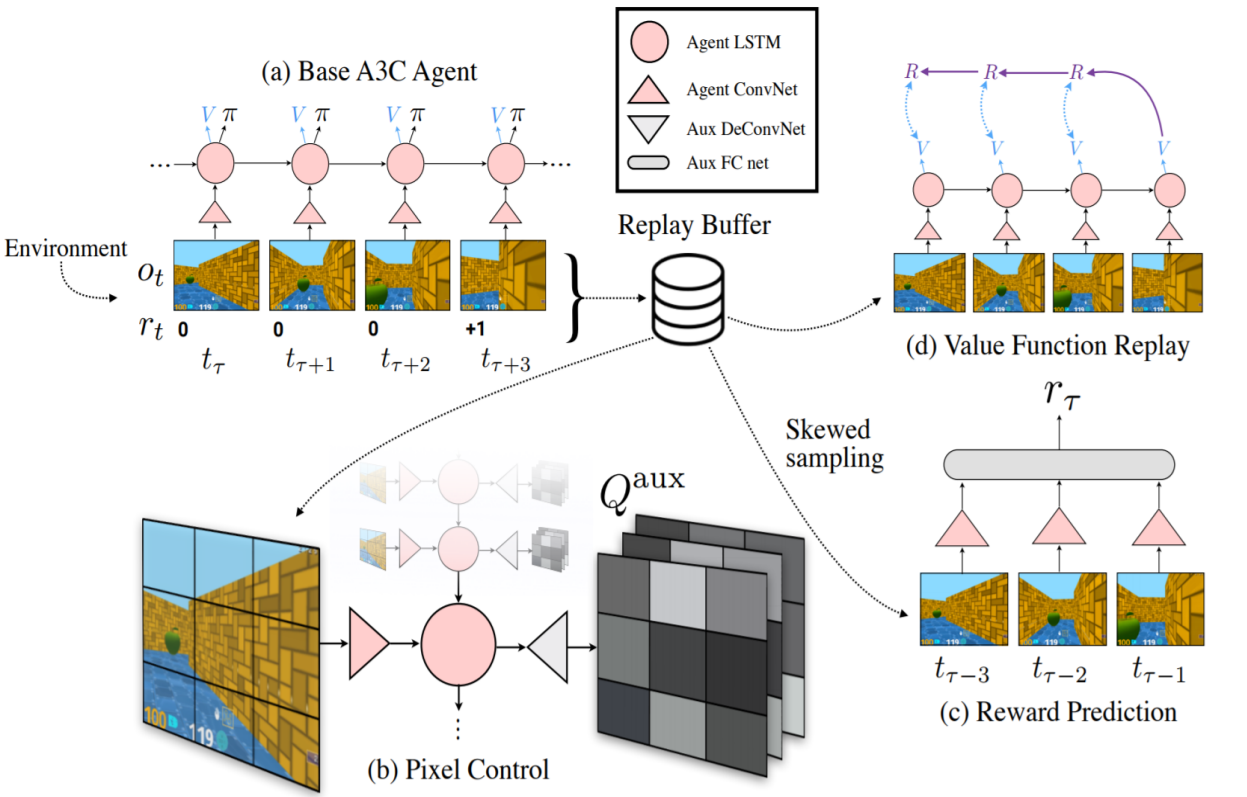

Figure 1: Overview of the *UNREAL* agent. (a) The base agent is a CNN-LSTM agent trained on-policy with the A3C loss (Mnih et al., 2016). Observations, rewards, and actions are stored in a small replay buffer which encapsulates a short history of agent experience. This experience is used by auxiliary learning tasks. (b) Pixel Control – auxiliary policies Q^{aux} are trained to maximise change in pixel intensity of different regions of the input. The agent CNN and LSTM are used for this task along with an auxiliary deconvolution network. This auxiliary control task requires the agent to learn how to control the environment. (c) Reward Prediction – given three recent frames, the network must predict the reward that will be obtained in the next unobserved timestep. This task network uses instances of the agent CNN, and is trained on reward biased sequences to remove the perceptual sparsity of rewards. (d) Value Function Replay – further training of the value function using the agent network is performed to promote faster value iteration. Further visualisation of the agent can be found in `https://youtu.be/Uz-zGYrYEjA`

Outline

Background

- Standard RL setting where agent interacts with an environment over a number of discrete time steps
- Agent's state s_t is a function of its experience up until time t

$$s_t = f(o_1, r_1, a_1, ..., o_t, r_t)$$

- n-step return $R_{t:t+n}$ is a discontinued sum of rewards

$$R_{t:t+n} = \sum_{i=1}^{n} \gamma^i r_{t+1}$$

Background

- Value function is the expected return from state s, when actions are selected according to a policy $\pi(a|s)$

$$V^{\pi}(s) = \mathbb{E}[R_{t:\infty}|s_t = s, \pi]$$

- The action-value function is the expected return following action a from state s

$$Q^{\pi}(s, a) = \mathbb{E}[R_{t:\infty}|s_t = s, a_t = a, \pi]$$

- The goal in Q-learning and asynchronous Q-learning is to approximate the action-value function, $Q(s, a; \theta)$ using parameters θ
- Update parameters to minimize the mean-squared error, for example using an n-step lookahead loss:

$$\mathcal{L}_Q = \mathbb{E}[(R_{t:t+n} + \gamma^n \max_{a'} Q(s', a'; \theta^-) - Q(s, a; \theta))^2]$$

Background

- In practice, these formulas are approximated using some function of the states (which are usually parametrized by variables)
- The Asynchronous Advantage Actor-Critic (A3C) algorithm approximates the policy, $\pi(a|s,\theta)$ and the value function $V(s,\theta)$
- The policy and the value approximation are adjusted towards an n-step lookahead loss using an entropy regularization penalty, so the loss is:

$$\mathcal{L}_{A3C} \approx \mathcal{L}_{VR} + \mathcal{L}_{\pi} - \mathbb{E}_{s\sim\pi}[\alpha H(\pi(s,;\theta)]$$

where

$$\mathcal{L}_{VR} = \mathbb{E}_{s\sim\pi}[(R_{t:t+n} + \gamma^n V(s_{t+n+1},\theta^-) - V(s_t,\theta))^2]$$

Background

- A3C uses many instances of the agent to interact in parallel with the environment, this accelerates and stabilizes the learning
- They utilize an LSTM to jointly approximate both policy, π and value function, V using the entire history of experience as inputs

Outline

Outline

Auxiliary Control Tasks

- Define an auxiliary control task, c by a reward function $r^{(c)} : \mathcal{S}x\mathcal{A} \rightarrow \mathbb{R}$
  - Where $\mathcal{S}$ is the space of possible states including the history of observations and rewards, and the current state of the agent (as in the activation of the hidden units of the network)
  - $\mathcal{A}$ is the space of available actions
- Let $\pi^{(c)}$ be the agent's policy for each auxiliary task, $c \in \mathcal{C}$, the objective is to maximize total performance across all tasks:

$$\arg\max_{\theta} \mathbb{E}_{\pi}[R_{1:\infty}] + \lambda_c \sum_{c \in \mathcal{C}} \mathbb{E}_{\pi_c}[R^{(c)}_{1:\infty}]$$

- $R^{(c)} = \sum_{k=1}^{n} \gamma^k r_t^{(c)}$ is the discounted return for auxiliary reward $r^{(c)}$, and θ is the set of parameters of π and all $\pi^{(c)\prime}s$
- By sharing some of the parameters of π and $\pi^{(c)}$ the agent must balance improving the global reward and auxiliary tasks

Auxiliary Control Tasks

- Any RL method could be applied to maximize these objectives, but they go with Q-learning, which is off-policy
  - This is crucial to allow the agent to learn many different pseudo-rewards simultaneously from a single stream of experience
- For each control task, they optimize an n-step Q-learning loss:

$$\mathcal{L}_Q^{(c)} = \mathbb{E}[(R_{t:t+n} + \gamma^n \max_{a'} Q^{(c)}(s', a', \theta^-) - Q^{(c)}(s, a, \theta))^2]$$

Auxiliary Control Tasks

- They focus on two main auxiliary reward functions that others are based off of
  - Pixel changes - changes in the perceptual stream often correspond to important events in an environment, so one auxiliary task is maximally observe changes in the input image pixels
  - Network features - in order to extract task-relevant high-level features of the environment, the activation of hidden units itself is an auxiliary reward

Auxiliary Control Tasks

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Initialize $Q(s, a)$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
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'
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
    $S \leftarrow S'$
  until S is terminal

Sarsa (on-policy TD control) for estimating $Q \approx q_*$

Initialize $Q(s, a)$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
  Initialize S
  Choose A from S using policy derived from Q (e.g., ϵ-greedy)
  Repeat (for each step of episode):
    Take action A, observe R, S'
    Choose A' from S' using policy derived from Q (e.g., ϵ-greedy)
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma Q(S', A') - Q(S, A)\big]$
    $S \leftarrow S'$; $A \leftarrow A'$;
  until S is terminal

Outline

Auxiliary Reward Tasks

- An agent wants to optimize its reward, but isn't necessarily trained to know when it might get reward, so one of the auxiliary reward tasks is *reward prediction*
- This task is done by processing a sequence of consecutive observations, and requiring the agent to predict the reward picked up in the unseen frame
- This simply going to help shape the features
- This affects the network and the representation of the features and predictor without shaping the value function or policy

Outline

Experience Replay

- Empirically proven to be useful in many other uses
- Main idea: store transitions in a replay buffer, and then apply learning updates to sampled transitions from this buffer
- This allows a training bias towards positive reward, compared to directly sampling from behaviour policy
- Equally sample rewarding and non-rewarding replays
- Use the replay buffer to perform value function replay
- Sparsity over time of replay allows for new features to be found
- Since there is already a large bias for positive value evaluation, they don't skew the sampling for value function replay

Experience Replay

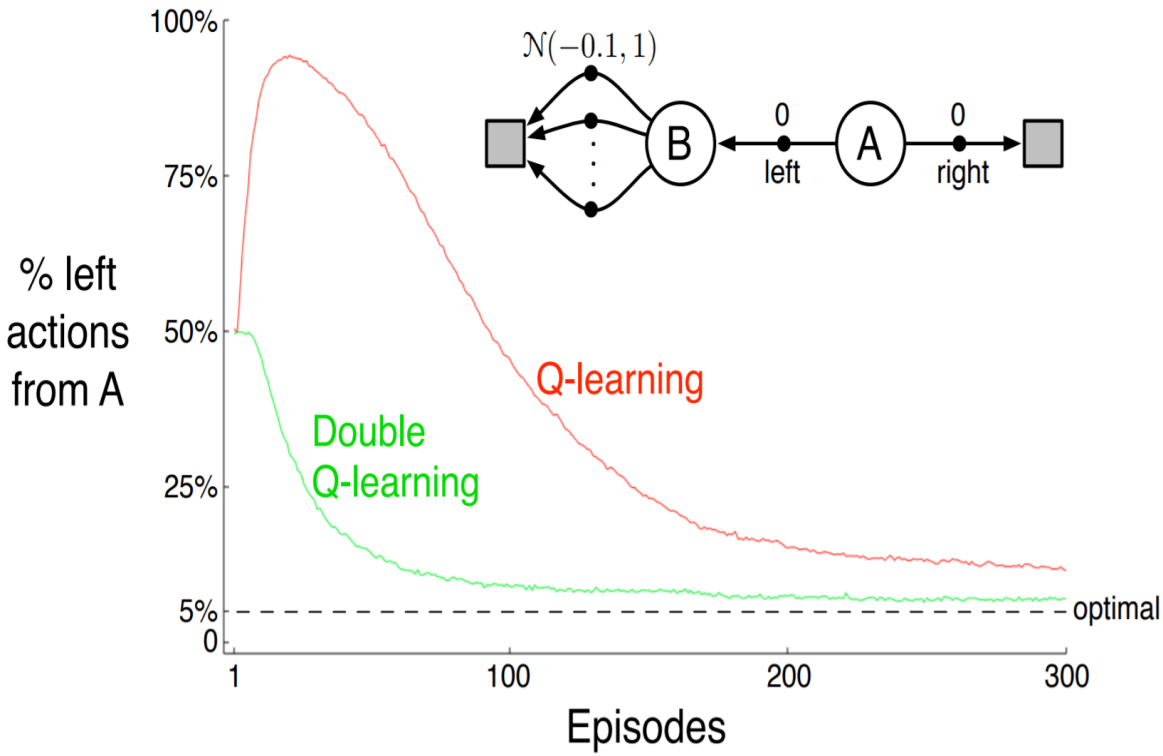

Figure 6.7: Comparison of Q-learning and Double Q-learning on a simple episodic MDP (shown inset). Q-learning initially learns to take the left action much more often than the right action, and always takes it significantly more often than the 5% minimum probability enforced by ε-greedy action selection with $\varepsilon = 0.1$. In contrast, Double Q-learning is essentially unaffected by maximization bias. These data are averaged over 10,000 runs. The initial action-value estimates were zero. Any ties in ε-greedy action selection were broken randomly.

Outline

UNREAL Agent

- Bringing everything together, UNREAL:
  - Utilizes A3C, learning from parallel streams of experience to gain efficiency and stability
  - Uses an RNN to encode the complete history
- The auxiliary tasks are trained on recent sequences of experience and randomly sampled
  - These targets are trained off-policy by Q-learning using a simple feedforward architecture

UNREAL Agent

- The UNREAL algorithm, as a whole, optimizes a single combined loss function wrt to the joint parameters and using λ as the weighting terms on each loss component, θ:

$$\mathcal{L}_{UNREAL}(\theta) = \mathcal{L}_{A3C} + \lambda_{VR}\mathcal{L}_{VR} + \lambda_{PC}\sum_{c}\mathcal{L}_{Q}^{(c)} + \lambda_{RP}\mathcal{L}_{RP}$$

- A3C, $\mathcal{L}_{A3C}$, is optimized on-policy from direct experience
- The value function loss, $\mathcal{L}_{VR}$, is optimized from replay data, but also on-policy for the component in A3C
- The auxiliary control loss, $\mathcal{L}_{PC}$, is optimized off-policy from replayed data, by n-step Q-learning
- The reward loss, $\mathcal{L}_{RP}$, is optimized from re-balanced replay data

Outline

Experiments

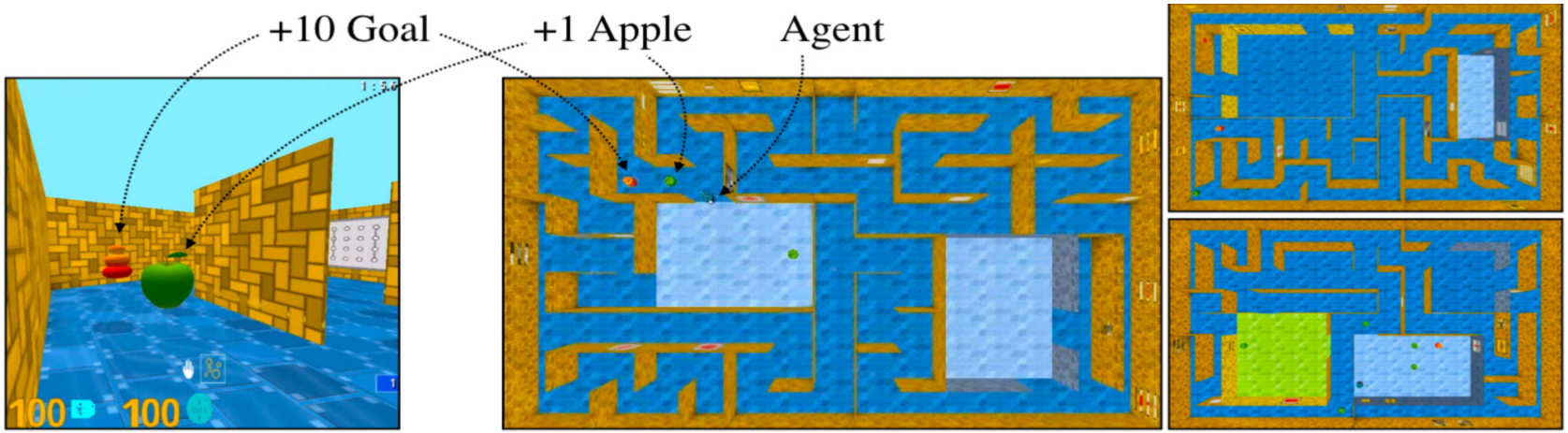

Figure 2: The raw RGB frame from the environment is the observation that is given as input to the agent, along with the last action and reward. This observation is shown for a sample of a maze from the nav_maze_all_random_02 level in Labyrinth. The agent must navigate this unseen maze and pick up apples giving +1 reward and reach the goal giving +10 reward, after which it will respawn. Top down views of samples from this maze generator show the variety of mazes procedurally created. A video showing the agent playing Labyrinth levels can be viewed at `https://youtu.be/Uz-zGYrYEjA`

Experiments and Results

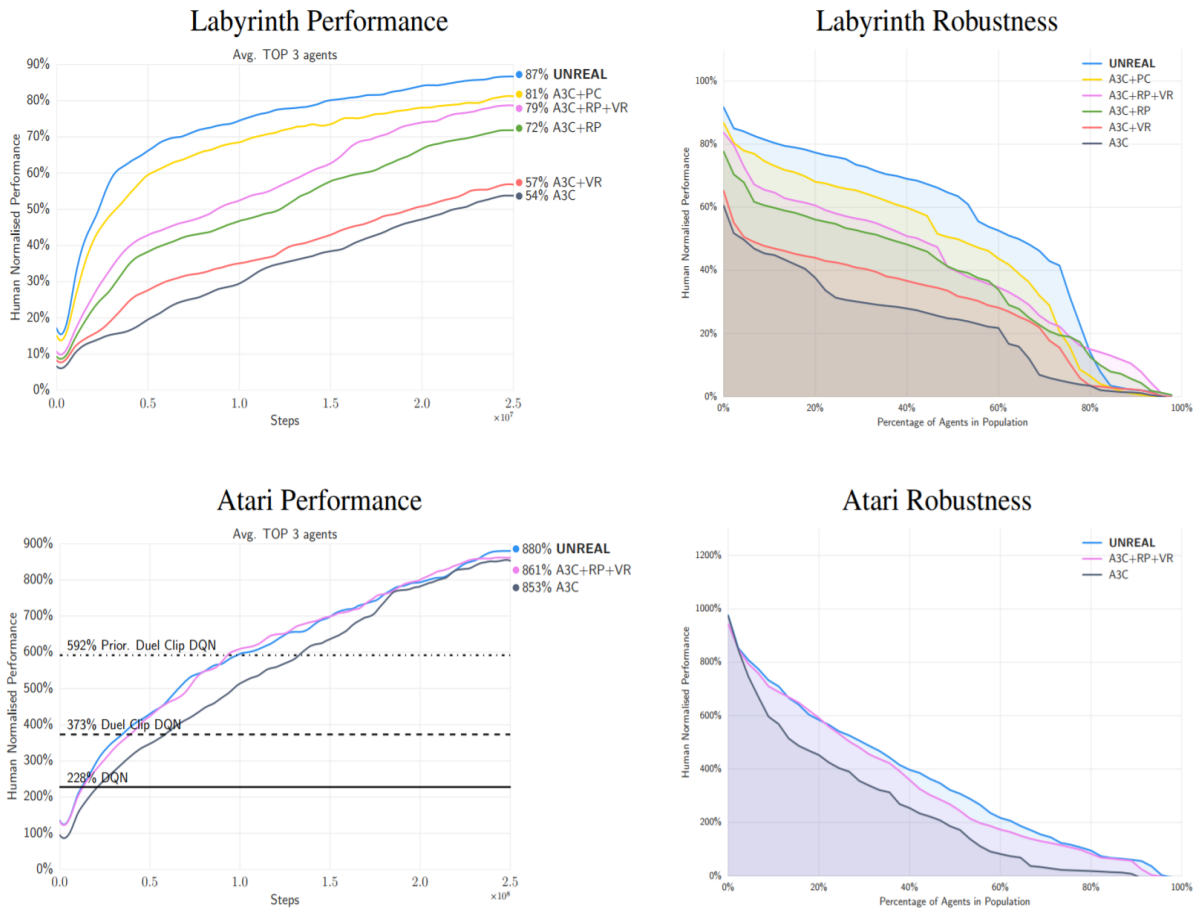

Figure 3: An overview of performance averaged across all levels on Labyrinth (Top) and Atari (Bottom). In the ablated versions RP is reward prediction, VR is value function replay, and PC is pixel control, with the *UNREAL* agent being the combination of all. *Left*: The mean human-normalised performance over last 100 episodes of the top-3 jobs at every point in training. We achieve an average of 87% human-normalised score, with every element of the agent improving upon the 54% human-normalised score of vanilla A3C. *Right*: The final human-normalised score of every job in our hyperparameter sweep, sorted by score. On both Labyrinth and Atari, the *UNREAL* agent increases the robustness to the hyperparameters (namely learning rate and entropy cost).

Experiments and Results

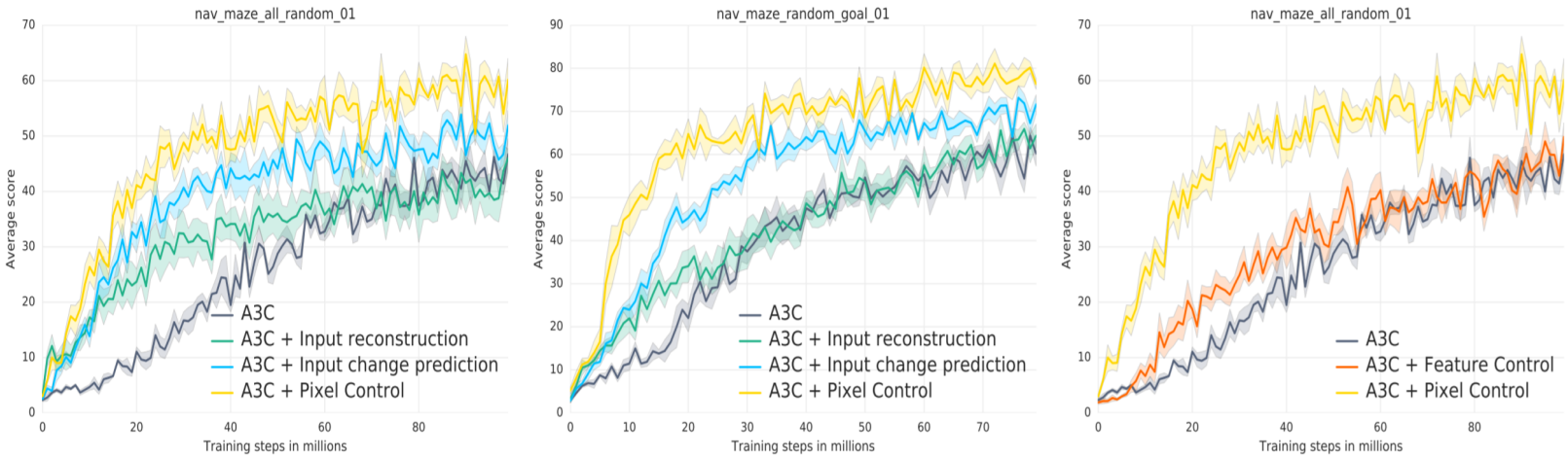

Figure 5: Comparison of various forms of self-supervised learning on random maze navigation. Adding an input reconstruction loss to the objective leads to faster learning compared to an A3C baseline. Predicting changes in the inputs works better than simple image reconstruction. Learning to control changes leads to the best results.

Experiments and Results

- They found that the unsupervised RL helped a lot, showing that learning to control the pixel changes helped more than just predicting the immediate pixel changes
- This paper has shown that augmenting a deep RL agent with auxiliary control and reward prediction tasks can drastically improve both data efficiency and robustness to hyperparameter settings
- Double the previous state-of-the-art results, and increased learning speed significantly
- https://www.youtube.com/watch?v=Uz-zGYrYEjA

Citations

Reinforcement Learning: An Introduction by Richard S. Sutton and Andrew G. Barto