Auto-Perceptive Reinforcement Learning (APRiL)

Rebecca Allday, Simon Hadfield, and Richard Bowden
Center for Vision, Speech and Signal Processing (CVSSP)
University of Surrey, Guildford, United Kingdom
{r.allday, s.hadfield, r.bowden}@surrey.ac.uk

Abstract

The relationship between the feedback given in Reinforcement Learning (RL) and visual data input is often extremely complex. Given this, expecting a single system trained end-to-end to learn both how to perceive and interact with its environment is unrealistic for complex domains. In this paper we propose Auto-Perceptive Reinforcement Learning (APRiL), separating the perception and the control elements of the task. This method uses an auto-perceptive network to encode a feature space. The feature space may explicitly encode available knowledge from the semantically understood state space but the network is also free to encode unanticipated auxiliary data. By decoupling visual perception from the RL process, APRiL can make use of techniques shown to improve performance and efficiency of RL training, which are often difficult to apply directly with a visual input. We present results showing that APRiL is effective in tasks where the semantically understood state space is known. We also demonstrate that allowing the feature space to learn auxiliary information, allows it to use the visual perception system to improve performance by approximately 30%. We also show that maintaining some level of semantics in the encoded state, which can then make use of state-of-the art RL techniques, saves around 75% of the time that would be used to collect simulation examples.

CEUR-WS.org/Vol-2487/anpaper3.pdf

1 Introduction

Unifying deep reinforcement learning with visual perception is often slow and ineffective for high dimensional problems with continuous action spaces. This is perhaps unsurprising as training directly from percepts through to actions for control is a complex relationship with poorly constrained supervision. This is especially true due to the high dimensionality of the image domain and the fact that many techniques used to speed up learning cannot be applied in image based systems. In nature, learning to interpret our surroundings and interact with them are learnt simultaneously but not necessarily as one continuous system [1]. Inspired by this, we use interaction with the world to co-train separate visual perception and control systems (Fig. 1).

This work shows how an auto-perception network can be used to learn an effective state space for reinforcement learning. Unlike previous RL techniques which try to learn an encoded state space, the proposed solution generalises across all levels of state observability. When the state space is completely or partially observable at training time then it is used to condition the learning of a representative feature space. The conditioned auto-encoder can be used to create a feature space which includes this knowledge but is not limited to it - allowing retrieval of other auxiliary information from the observations which may not have been considered by the developer.

Copyright © by the paper's authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

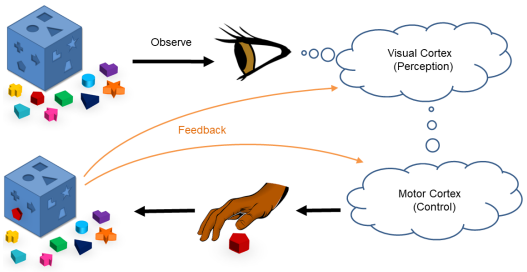


Figure 1: Perception and control in the brain are linked but separate systems requiring different feedback.

2 Related Work

Deep RL has seen advances recently with work like Deep Q-Networks [2] which uses a deep convolutional neural network (CNN) to approximate the action-value function in a Q-learning method to learn to play Atari games. There have since been many variations on DQNs such as using recurrent neural networks in place of a standard feed-forward CNN [3] and adaptations for use with continuous action spaces [4]. Whilst these value based methods for RL have proved popular, policy based and actor-critic methods have also been successfully adapted for deep learning. In this work we use a synchronous version of Mnih et al.'s Asynchronous Advantage Actor-Critic (A3C) method [5].

RL algorithms are often tested using simple software simulators such as video games or simple physics problems (e.g. cart-pole). This makes it easy to accumulate the number of episodes required to train the networks, which is not practical for more realistic robotics applications. Many techniques for approaching the issue of data collection have been suggested. For example, Hindsight Experience Replay (HER) [6] allows RL to learn from unsuccessful episodes by changing the goal and hence the reward feedback. However, in order to apply this to the image domain, a method for synthesising images is required to change the goal. There have also been model based techniques aimed at reducing the number of experiences needed for training. Black-DROPS (Black-box Data-efficient RObot Policy Search) [7], for example, uses Gaussian Processes (GPs) to learn the dynamics of a system with a small number of experiences and then produces experiences for training the RL directly from the GPs. This accelerates the RL process but is focused on systems where the state is fully observed and has a small number of dimensions. The large dimensionality of observations only available as images are not suitable for GP dynamics modeling.

Advances in deep learning has meant that feature spaces can be created which represent the important aspects of a visual observation. Deep auto-encoders [8] have been used extensively to reduce dimensionality of data and have been used with CNNs [9] to help retain the spatial relationships in images. As well as providing a low-dimensional feature space they are also used to create generative models, for example in image restoration [10].

Considering the problems high dimensional spaces cause in RL, it is not surprising that attempts to use auto-encoder networks with RL have been made. Table 1 compares the uses of auto-encoders in RL systems. Finn et al. [13] use an auto-encoder to create a set of feature points representing positions in the image that describe the environment, for example where objects are. Stadie et al. [15] encode a state for training a dynamics model in order to improve exploration by increasing curiosity, but still use the raw observation as the input to the learning system. Lange and Riedmiller [11] use a deep auto-encoder to compress a visual input to a low dimensional feature space, which is not semantically understood. This improves the reinforcement learning data-efficiency. Kimura [16] uses auto-encoders as pre-training for a DQN system. However, none of these approaches can exploit valuable RL techniques, such as HER. Lange and Reidmiller's work does not have the semantic understanding required in the features in order to adapt the episode with a new goal. Kimura's requires images, for fine-tuning of the network, which cannot be adapted for a new goal.

Nair et al. [17] propose a solution to goal-conditioned RL, using an encoder-decoder system to learn a latent space which can be used to sample goals, provide a lower dimensional, structured input for RL, and to compute a reward signal. Although this allows HER to be used for visual problems, it introduces its own limitations. In using an image as an explicit goal, the agent's flexibility is limited. For example, in a pick and place problem it restrains the final position of the robot when the final position of the object is more important. They also assume that only the image is available to the RL system at train time, they do not consider cases where we may want to make use of the state that is available - meaning information is wasted.

In contrast APRiL makes use of whatever semantically understood state information is available at train time,

Table 1: Comparison of different works using Reinforcement Learning (RL) and Auto-encoders (AE)

	Use of AE	**RL method**	**RL Input Space**	**Goal conditioned**	**Semantics in RL input**
Lange and Riedmiller [11]	Encode image	FQI [12]	Latent space	No	None
Finn et al. [13]	Encode image	Gaussian Controller + Guided Policy Search [14]	Robot state + latent space	Implicit in image - encoded into latent space	Partially
Stadie et al. [15]	Encode image for augmented reward	DQN [2]	Images	Implicit in image	-
Kimura [16]	Pretrain RL network	DQN [2]	Images	Implicit in image	-
Nair et al. [17]	Encode image, Calc reward, Generate data	TD3 [18]	Latent space	Conditioned with a point in the state space	None
APRiL (Ours)	Encode image	A2C [5]	Available env and robot state + latent space	Implicit in input (state or latent space)	Variable

whilst still allowing additional auxiliary information to be encoded from the visual input. This gives a system which makes full use of the information and RL techniques available at train time but can still be deployed using vision as the input.

3 Methodology

Fig. 2 shows the outline of the proposed APRiL approach. The black arrows show the data flow at deployment. The observation image of the agent and environment is passed to the encoder which provides an encoded state, which may be completely or partially semantically understood. This is then passed to the trained reinforcement learning system which selects an action which is passed back to the agent to be executed.

The flow of the data when the system is being trained can also be seen in Fig. 2. The optional loss can be used if semantically understood knowledge of the state is partly or fully available. The perception network is trained independently on data collected with an initial random walk policy from the RL system. The RL block is trained using data both from the agent (in this case a physics simulator) and from a pre-trained Gaussian Process which models the dynamics of the system. This means that the RL system can obtain vast quantities of data points without having to run them all through a physics simulator, speeding up the process. The following sections elaborate on the individual elements and how they are trained.

3.1 Reinforcement Learning

A formalisation of episodic reinforcement learning is used where an agent interacts with an environment at discrete time steps, t, with a maximum number of steps T. There is a set of states $s_t \in \mathcal{S}$ and a set of actions the agent can perform $a_t \in \mathcal{A}$. The goal is to maximise the discounted sum of reward signal r_t over time,

$$R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k} \tag{1}$$

where $\gamma \in [0, 1]$ is the discount factor for future rewards. In order to maximise R_t we learn a policy $\pi(a \mid s_t)$, which estimates a distribution over the possible actions, $a \in \mathcal{A}$, conditioned on the current state s_t. We sample a_t from this distribution $\pi(a \mid s_t)$. The value is defined as $V^\pi(s_t) = \mathbb{E}\left(R_t \mid s_t, \pi\right)$, the expected return R_t given a particular policy starting in a particular state s_t. Finally, the action-value function is defined as $Q^\pi(s_t, a_t) = \mathbb{E}\left(R_t \mid s_t, a_t, \pi\right)$,

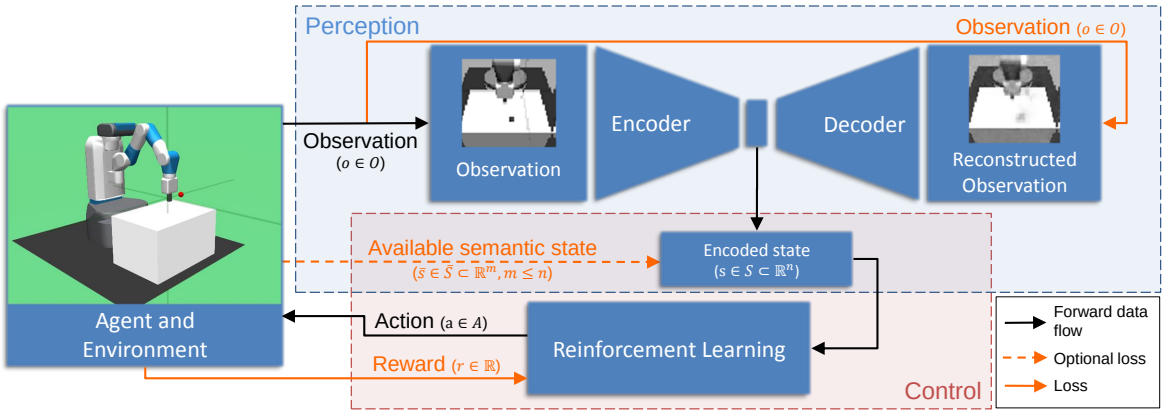


Figure 2: Overview of APRiL. The optional loss and the $|\mathcal{S}|$ determines how much of the encoded latent space is semantically understood. Black arrows: the data flow in the forward pass, Orange arrows: the data flow in the backward pass.

the expected return R_t given a particular policy, starting with a particular action a_t from a specified stated s_t. For the visual aspect we define $o_t \in \mathcal{O}$ as an image of the system.

In this work we use Advantage-Actor-Critic style reinforcement learning [5]. This system produces two outputs - a stochastic policy (the actor) and an estimate of the value function (the critic). The ground truth value R_t is used to calculate the value loss

$$L_v = R_t - V(s_t). \tag{2}$$

The policy loss is calculated using the advantage, given by

$$A(s_t, a_t) = Q(s_t, a_t) - V(s_t). \tag{3}$$

The advantage gives the difference between the expected return given the action taken and the expected return of the state itself given the current policy - showing how much better or worse the action performed than expected. This can be approximated as the discounted rewards minus the predicted value for the current policy, taking the form $A(s_t, a_t) \approx R_t - V(s_t)$. The policy used is in the form of a Gaussian distribution, such that $\pi(a \mid s_t) = \mathcal{N}(\mu_a, \sigma_a^2)$. Given that an action a_t is then sampled and executed, the policy loss is then calculated as

$$L_p = \log \pi(a = a_t | s_t) A(s_t, a_t). \tag{4}$$

This means that an action which is better than expected will be made more likely, with a weighting of how likely it was in the first place. In contrast an action which performed worse will be made less likely for that state. The full loss for the RL network then takes the form

$$L_{\text{RL}} = \alpha L_v + \beta L_p + \epsilon H(\pi(s_t)) \tag{5}$$

where H is the entropy - which is included to encourage exploration - and α, β, ϵ are hyper-parameters which control the strength of each loss term. To ensure that the initial random value estimate is sensible and does not skew the policy loss, we train with $\alpha = 1, \beta = 0, \epsilon = 0$ for a small number of iterations.

We use a batch-style off-policy approach by storing up experience in a replay buffer and sampling from this to train the RL algorithm. We set a limit to our experience replay buffer to some value M so that as learning progresses, the oldest experiences are forgotten and replaced with more recent ones. The replay buffer is of the form $\Omega = \{e : |\Omega| < M\}$, where each episode of experiences is of the form $e = \{(s_t, a_t, R_t) : t = 1, .., j \text{ and } j \leq T\}$ where j is the terminating step for that episode. The probability of a_t being selected from the current policy and the value of the s_t for the current policy is found at training time.

3.1.1 Hindsight Experience Replay

Hindsight Experience Replay (HER) [6] is a powerful technique which allows us to learn from unsuccessful episodes in learning, especially where rewards are sparse and success from random exploration may be limited. Using HER we can adjust the goal for our system to a state it achieved in the current episode - meaning we artificially

create successful episodes. For a given episode $s_1, ..., s_T$ where a goal $g \neq s_1, ..., s_t$, we may "replay" this episode with $g = s_i$ for some $1 < i < T$ knowing that it will achieve the goal. Adding these adapted episodes to the experience replay, Ω, means the episode buffer then has more episodes to learn from and has a more balanced ratio of successful episodes without needing excessive exploration.

3.1.2 Gaussian Process Model

In order to reduce the number of costly agent-environment interactions we use Gaussian Processes (GPs) to approximate the dynamics of our system and give uncertainty information. We use a small number of interactions with the agent and environment to train the GP - this takes in the current state, s_t, and the action to be taken, a_t. It is then optimized to output a Gaussian distribution which estimates the next state s_{t+1} with uncertainty.

We represent the dynamics of our system as

$$s_{t+1} = s_t + D(s_t, a_t) + \mathrm{w}, \tag{6}$$

with w (Gaussian system noise) and D (unknown transition dynamics). Given that $x_t = (s_t, a_t)$, the GP is computed as

$$\hat{D}(x_t) \sim \mathcal{GP}(\mu_{\hat{\mathrm{D}}}(x_t), k_{\hat{\mathrm{D}}}(x_t, x_t')), \tag{7}$$

where $\mu_{\hat{\mathrm{D}}}$ is the mean function and $k_{\hat{\mathrm{D}}}$ is the kernel function. With a set of observed transitions $Y_{1:t} = D(x_1), ..., D(x_t)$, we can query our GP at a new data point x_* to obtain a distribution over expected state updates:

$$p(\hat{D}(x_*) \mid Y_{1:t}, x_*) = \mathcal{N}(\mu_{\hat{\mathrm{D}}}(x_*), \sigma^2_{\hat{\mathrm{D}}}(x_*)). \tag{8}$$

Sampling from this Gaussian allows the rapid creation of more episodes to train the RL system. The same reward calculations as the normal environment are used so these episodes can be added directly to Ω as before.

3.2 Auto-Perceptive Network

The perception part of our system is an auto-encoder. This allows us to encode a feature space to use as the state space, $\mathcal{S}$, which is the input to the RL system. The encoder uses the observations of the agent and environment in the form of an image, transforming it to the feature space as the function $\phi_{enc} : \mathcal{O} \to \mathcal{S}$, whilst the decoder arm transforms from the feature space to a reconstructed image $\phi_{dec} : \mathcal{S} \to \mathcal{O}$.

The auto-encoder takes the image observation of the system as an input and compresses it down to the feature space $s_t = \phi_{enc}(o_t)$ and the output is a reconstruction of that image $\hat{o}_t = \phi_{dec} \circ \phi_{enc}(o_t)$. The reconstruction loss is a pixel-wise loss against the input

$$L_r = |o_t - \hat{o}_t|. \tag{9}$$

We denote the space of available information from the environment, which has a predefined semantic meaning, as $\bar{s}_t \in \bar{\mathcal{S}}$. The optional conditioning loss is the absolute difference between a section of the encoded state space and the semantically understood state. In the case where $\mathcal{S} \subset \mathbb{R}^n$ and $\bar{\mathcal{S}} \subset \mathbb{R}^m$, with $m \leq n$, then the conditioning loss is

$$L_c = |s_t^{1:m} - \bar{s}_t|. \tag{10}$$

The full loss for the visual perception network is

$$L_{\mathrm{VP}} = L_r + \omega L_c, \tag{11}$$

where ω is a weighting which determines how strong the conditioning is. The learnt feature space can be:

1. entirely conditioned to be semantically understood as the observable state ($m = n$, $\omega \neq 0$),
2. partially conditioned with some learnt features relating to the observable state and some auxiliary features with no predetermined semantic meaning ($m < n$, $\omega \neq 0$),
3. or not conditioned with learnt features having no predetermined semantic meaning ($\omega = 0$).

This network can be trained using data from initial random exploration and fine-tuned during reinforcement learning.

3.3 Auto-Perceptive Reinforcement Learning (APRiL)

The RL system and the auto-perception network are independent networks, which can be trained concurrently with much of the same data but do not need to be trained end-to-end as they exploit different types of supervision.

In the case of the encoded feature space being entirely semantically understood the auto-encoder is trained with data collected for the initial random exploration - the same data can be used to train the GP to learn the dynamics. These may be co-trained in parallel and tested individually before being integrated. The perception network infers an approximated state from an observation and then passes this approximate state to the RL network without the RL needing to see any images during training. This still provides a system which does not need access to the robot state at run time and can predict actions with only visual input, but does not require it to be trained in an end-to-end manner, allowing RL to benefit from HER and GP modelled transition dynamics.

When the encoded feature space is partially semantically understood then the auto-encoder will still be pre-trained on random data but the encoder arm will be used to get the encoded state for input to the RL system. Therefore the RL system only has to interpret the low dimensional feature space coming from the auto-encoder and does not need to process the images. This means that the training is more focused on solving the control problem. Techniques such as HER are still feasible since we have a predetermined understanding of some of the feature space being used by the RL.

The final case is where there is no semantically understood state available. This is similar to Lange and Riedmiller's work [11] where the encoder feature space had no predetermined semantic meaning. This case still allows a lower dimensional state space to be learnt from the visual input even when there is no semantic state available during training.

4 Experiments and Results

To evaluate APRiL we use the OpenAI [19] framework with the Mujoco physics simulator [20]. We use a variation of the Fetch robot reach environment because it has a continuous action space and has a visually interesting environment to test the auto-perceptive system. The aim is to direct the end-effector of the Fetch arm to a goal g_x, g_y, g_z - represented visually by a red sphere. The action space is defined with actions $(\Delta x, \Delta y, \Delta z)$ where (x, y, z) is the position of the end-effector and the maximum episode length is set as $T = 50$. We train the networks using Adam optimizers [21] and a Tensorflow [22] implementation of our system will be available at `https://github.com/rebecca-allday/APRiL`.

4.1 Fully Semantic Features

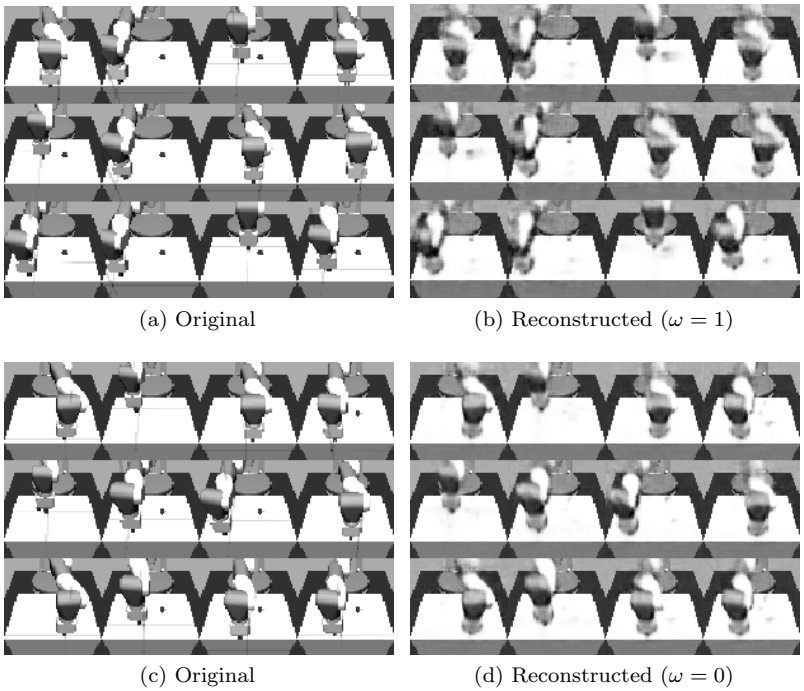
The first experiment uses a fully observed, semantically understood state $\bar{s}_t = (x_t, y_t, z_t, g_x, g_y, g_z)$. Firstly, we use a random policy to collect an initial experience replay buffer. This data can be used to train multiple aspects of the system. Initially we train a GP on the transitions taking in $(x_t, y_t, z_t, \Delta x, \Delta y, \Delta z)$ and outputting $(x_{t+1}, y_{t+1}, z_{t+1})$. This allows us to create extra episodes to train our RL system as described in Section 3.1.2. The advantage actor-critic RL system is trained with data created from both the GP and from the agent, including the HER additions to the replay buffer. The data from the random policy and any episodes collected using the simulator are used to train the perception network. In this case the perception network is co-trained such that $\bar{s}_t = s_t = \phi_{enc}(o_t)$, which is the first case from Section 3.2, when $m = n$ and $\omega \neq 0$. Finally, at test time the networks can be used together to go directly from vision to actions, following the data flow shown by the black arrows in Fig. 2. We compare this to a latent space with no conditioning loss, where $\omega = 0$, which is similar to [11].

The training of the RL system, using the semantically understood state space directly, converges with only 15 episodes of random policy interactions with the simulator, the rest of the data used is collected from our trained GP. It takes approximately 0.01 seconds per rendered simulation step, but only 0.0025 seconds to sample a single step from the GP. This equates to saving 75% of the time that would have been spent on collecting simulation examples. This is a saving that would not be possible using a traditional end-to-end visual RL algorithm.

Table 2 shows the policy achieves an average episode length of 3.1 actions when using the ground truth state space as input. The perception network is trained alongside this. Examples of the reconstructions from the auto-encoder can be seen in Fig. 3, along with reconstructions from the auto-encoder without the semantic conditioning ($\omega = 0$). Even though we fully constrain the encoded feature space, and do not enable the system to encode many visual properties, the decoder arm is still able to learn how to produce realistic images of the scene from a non-visual intermediate state, including how to correctly place a fully textured robotic arm. They

Table 2: Average episode length (actions to complete task) of system trained on the Fetch Reach env (no obstacle).

Runtime RL Input	Average Episode Length
Ground truth $\bar{\mathcal{S}}$	3.10
Perceived $\mathcal{S}$ ($\omega = 1.0$)	12.42
Perceived $\mathcal{S}$ ($\omega = 0.5$)	17.06
Perceived $\mathcal{S}$ ($\omega = 0.0$) [11]	30.94



(a) Original (b) Reconstructed ($\omega = 1$)

(c) Original (d) Reconstructed ($\omega = 0$)

Figure 3: Reconstructions from the auto-perceptive network

are certainly comparable to the reconstructions without the conditioning loss. However, reconstruction accuracy is unimportant, the key is the reconstruction loss aids encoding meaningful information into the latent space for the RL.

At test time we can see the performance of the system using the visual encoder network to produce the feature space, which is an approximation of the semantically understood state space, given to the RL network. The policy achieves an average episode length of 12.4 actions. This is largely due to the goal or end point being occluded or out of the field of view, in which case the arm must move to attempt to gather more information about its current state. In these situations, the ground truth algorithm is an unrealistic comparison for a vision based system which will never have full access to the state. However, this is still much more effective than the case when the perceived state, $\mathcal{S}$, is not conditioned on the semantically understood state, $\bar{\mathcal{S}}$, which is similar to [11].

4.2 Partially Semantic Features

The next set of experiments introduces an element such that the state is not be fully observed via a semantically understood state space. A randomly placed obstacle (box) is added which can affect exploration and potential solutions for getting to the goal (red sphere), see Fig. 4. Again we compare the results in this section to a network trained with no access to the available semantic state which is similar to [11]. We also train a system which takes the ground truth semantic state and a separate latent space (in a similar way to [13]) to show that if both are available the system has all the information it needs.

We first train APRiL on the same state that was available in the previous set-up. This means that the RL

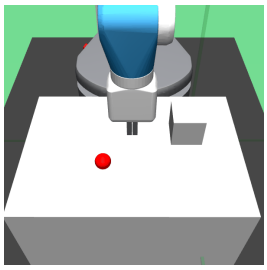


Figure 4: Fetch simulation with obstacle (box) and goal (sphere)

Table 3: Average episode length of system trained on an environment with a randomly placed obstacle.

Runtime RL Input	n	m	Average Episode Length
Ground Truth $\bar{\mathcal{S}}$	-	-	8.45
GT $\bar{\mathcal{S}}$ and Percieved $\mathcal{S}$ [13]	6	0	5.44
Perceived $\mathcal{S}$ [11]	16	0	37.04
Perceived $\mathcal{S}$	6	6	28.55
Perceived $\mathcal{S}$	8	6	20.83

system is not receiving any information about the obstacle. As expected, we see a reduction in performance compared to the environment with no obstacle. From 3.10 average actions per episode with no obstacle to 8.45 with obstacles - this equates to approximately a 2.5 times increase in the number of actions. Examples of the reconstructions from the perception network are seen in Fig. 5b. These reconstructions are comparable to those in Fig. 3b, with some slight degradation because the scene is more complex yet we have not given it any additional degrees of freedom in the latent space. It is interesting to note that the decoder arm attempts to reconstruct the obstacle even though it is theoretically not present in the intermediate state.

When testing with the perception to action system we see that this gives much worse performance with an average episode length of 28.55 actions (See Table 3). It is good to note that this is in comparison to 12.42 actions with no obstacles, equating to approximately a 2.5 times increase in the number of actions which is similar in scale to the decrease in performance seen without perception. This is likely because it has no way of knowing about the obstacle in the encoded state and often mistakes it for the goal, especially if the goal is occluded by the arm.

Next we allowed the encoded feature space to be only partially semantically understood. We used a feature space of size $n = 8$, with the semantically understood state $\bar{s}_t$ conditioning only the first 6 elements (i.e. $m = 6$). The rest were driven purely by the reconstruction loss, allowing it to learn whatever was relevant to the understanding of the environment. Example reconstructions from the perception network can be seen in Fig. 5d. This trained perception network does a better job of modelling the obstacle and goal as independent objects, however the robot arm has lost a significant amount of visual fidelity. This may be because all systems have been trained for the same number of iterations, despite this one having more network parameters. Regardless, a high fidelity image of the robotic arm is not important for RL, as long as the position is known.

The proposed RL system using our partially semantically understood feature space as input performs better than the system using just the semantic state, with an average of 20.83 actions (See Table 3). In comparison to the 12.42 actions in the environment with no obstacles, this is only a 1.68 times increase for what is a more difficult problem. This is approximately a 30% improvement compared to 28.55 average actions taken when using the semantic feature space. This shows that when we do not have access to the full semantically understood state our feature space can encode the additional auxiliary information necessary to solve the task better than just with the semantic state based perception.

Finally we give the perception network complete freedom to encode a state space based purely on the reconstruction loss in a similar manner to [11]. Fig. 5f shows that this improves the reconstruction as expected since that is the only feedback given to the encoder-decoder network. However, as we can see from Table 3 the performance of the system with no use of the semantically understood data available to it at train time performs much worse than those which do.

(a) Original (b) Reconstructed ($n = m = 6$)

(c) Original (d) Reconstructed ($n = 8, m = 6$)

(e) Original (f) Reconstructed ($n = 16, m = 0$)

Figure 5: Reconstructions from the auto-perceptive network for the env with obstacles - top: semantic features, middle: partially semantic features, bottom: non-semantic features.

5 Conclusion

In this paper we have shown that the bio-inspired separation of percepts and control at training time allows reinforcement learning to be trained effectively and still gives a system that can predict actions purely from visual data. We showed that allowing the perception system to encode additional properties into the feature space improved the performance over a system using only the approximate state.

This demonstrates the value in allowing the visual system to encode additional features into the input of our RL algorithms. In addition, the splitting of perception and control allows other techniques to be used, which are typically challenging to implement in the high dimensional image domain, such as HER and modelling transition dynamics with GPs. Whilst we still have a system which allows us to go from visual observation to action - the training does not need to be end-to-end.

References

[1] M. Land and B. Tatler, *Looking and acting: vision and eye movements in natural behaviour*. Oxford University Press, 2009.

[2] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, 2015.

[3] M. Hausknecht and P. Stone, "Deep Recurrent Q-Learning for Partially Observable MDPs," *AAAI*, pp. 29–37, 2015.

[4] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, pp. 1–14, 2015.

[5] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *ICML*, 2016, pp. 1928–1937.

[6] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, P. Abbeel, and W. Zaremba, "Hindsight experience replay," *CoRR*, vol. abs/1707.01495, 2017.

[7] K. I. Chatzilygeroudis, R. Rama, R. Kaushik, D. Goepp, V. Vassiliades, and J. Mouret, "Black-box data-efficient policy search for robotics," *CoRR*, vol. abs/1703.07261, 2017.

[8] G. E. Hinton and R. R. Salakhutdinov, "Reducing the Dimensionality of Data with Neural Networks," *Science*, vol. 313, pp. 504–507, 2006.

[9] J. Masci, U. Meier, D. Cireşan, and J. Schmidhuber, "Stacked convolutional auto-encoders for hierarchical feature extraction," in *International Conference on Artificial Neural Networks*. Springer, 2011, pp. 52–59.

[10] X. Mao, C. Shen, and Y.-B. Yang, "Image restoration using very deep convolutional encoder-decoder networks with symmetric skip connections," in *Advances in Neural Information Processing Systems 29*. Curran Associates, Inc., 2016, pp. 2802–2810.

[11] S. Lange and M. Riedmiller, "Deep auto-encoder neural networks in reinforcement learning," in *The 2010 International Joint Conference on Neural Networks (IJCNN)*, July 2010, pp. 1–8.

[12] D. Ernst, P. Geurts, and L. Wehenkel, "Tree-based batch mode reinforcement learning," *Journal of Machine Learning Research*, vol. 6, no. Apr, pp. 503–556, 2005.

[13] C. Finn, X. Tan, Y. Duan, T. Darrell, S. Levine, and P. Abbeel, "Deep spatial autoencoders for visuomotor learning," in *ICRA*, 2016.

[14] S. Levine and P. Abbeel, "Learning neural network policies with guided policy search under unknown dynamics," in *Advances in Neural Information Processing Systems*, 2014, pp. 1071–1079.

[15] B. C. Stadie, S. Levine, and P. Abbeel, "Incentivizing exploration in reinforcement learning with deep predictive models," *CoRR*, vol. abs/1507.00814, 2015.

[16] D. Kimura, "DAQN: Deep Auto-encoder and Q-Network," *arXiv*, vol. abs/1710.06542, 2018.

[17] A. V. Nair, V. Pong, M. Dalal, S. Bahl, S. Lin, and S. Levine, "Visual reinforcement learning with imagined goals," in *Advances in Neural Information Processing Systems*, 2018, pp. 9209–9220.

[18] S. Fujimoto, H. van Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," *arXiv:1802.09477*, 2018.

[19] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," 2016.

[20] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*. IEEE, 2012, pp. 5026–5033.

[21] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *International Conference on Learning Representations (ICLR)*, 2015.

[22] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org.