# Learning From Alternative Sources of Supervision

*Harrison Edwards*

Doctor of Philosophy
Institute for Language, Cognition and Computation
School of Informatics
University of Edinburgh
2022

# Abstract

With the rise of the internet, data of many varieties including: images, audio, text and video are abundant. Unfortunately for a very specific task one might have, the data for that problem is not typically abundant unless you are lucky. Typically one might have only a small amount of labelled data, or only noisy labels, or labels for a different task, or perhaps a simulator and reward function but no demonstrations, or even a simulator but no reward function at all. However, arguably no task is truly novel and so it is often possible for neural networks to benefit from the abundant data that is related to your current task. This thesis documents three methods for learning from alternative sources of supervision, an alternative to the more preferable case of simply having unlimited amounts of direct examples of your task. Firstly we show how having data from many related tasks could be described with a simple graphical model and fit using a Variational-Autoencoder - directly modelling and representing the relations amongst tasks. Secondly we investigate various forms of prediction-based intrinsic rewards for agents in a simulator with no extrinsic rewards. Thirdly we introduce a novel intrinsic reward and investigate how to best combine it with an extrinsic reward for best performance.

# Acknowledgements

I dedicate this thesis to my wife Sarah, my daughter Ada, my unborn son Dylan and, just to be safe, any other children I may have in the future who might one day read this and wonder why they were left out. I would like to especially thank my advisor Amos Storkey for his mentorship and stimulating discussions. I would also like to thank all my coauthors for their hard work and insights, and my colleagues at the University of Edinburgh and OpenAI for their support and community.

# Declaration

I declare that the thesis has been composed by myself and that the work has not be submitted for any other degree or professional qualification. I confirm that the work submitted is my own, except where work which has formed part of jointly-authored publications has been included. My contribution and those of the other authors to this work have been explicitly indicated below. I confirm that appropriate credit has been given within this thesis where reference has been made to the work of others.

The work presented in Chapter 3 was previously published in *International Conference on Representation Learning* as 'Towards a Neural Statistician' by Harrison Edwards (thesis author) and Amos Storkey (supervisor). This study was conceived by all of the authors. I carried out all experimentation and jointly wrote the paper with my supervisor.

The work presented in Chapter 4 was previously published in *International Conference on Representation Learning* as 'Large-Scale Study of Curiosity-Driven Learning' by Harrison Edwards (thesis author), Yuri Burda, Deepak Pathak, Amos Storkey (supervisor), Trevor Darrell and Alexei A. Efros. Yuri Burda, Deepak Pathak and myself were joint first-authors on this work. My contribution was to initiate the project with an initial experimental proof of concept, followed by refining the approach with Yuri and writing and running all the code with Yuri and Deepak. The paper writing was a collaborative effort with all the coauthors.

The work presented in Chapter 5 was previously published in *International Conference on Representation Learning* as 'Exploration by Random Network Distillation' by Harrison Edwards (thesis author), Yuri Burda, Amos Storkey (supervisor) and Oleg Klimov. Yuri Burda and myself were joint first-authors on this work. My contribution was to propose the algorithm and run some initial experiments, followed by refining the approach with Yuri and writing and running all the code with Yuri. The paper writing was a collaborative effort with all the coauthors.

(*Harrison Edwards*)

# Table of Contents

# Chapter 1

# Introduction

Deep learning is by now a well-established umbrella of methods for turning a large pile of labelled data into, for instance, a highly accurate classifier. The day to day business of a deep-learning practitioner or researcher is in eking out a few more percentage points of accuracy by improvements to the optimisation procedure or architecture, but often times the labelled data you wish you had is just not available. In these cases sometimes a more creative approach is called for and you might consider unsupervised (for example learning representations with a Variational Autoencoder Kingma and Welling (2013a), self-supervised (for example using a pretext task like rearranging a shuffled image Noroozi and Favaro (2016)), semi-supervised (combining both labelled and unlabelled data, for example using the Kingma et al. (2014)) or weakly-supervised methods (for example making use of noisy image tagging labels as pretraining for classification Mahajan et al. (2018)). This involves learning from an alternative source of supervision to do useful representation learning which can then later be used for transfer learning to the main task of interest, or, in the case of reinforcement learning, instead of representation learning the useful product of alternatively-supervised learning might be the discovery of interesting and useful behaviours or states in the environment (for example a robot may in an unsupervised way through exploration learn useful behaviors such as "go to the kitchen", "go up the stairs" etc).

This thesis will present three papers treating the problem of limited or unusual forms of supervision. We will present each paper as a chapter. Each chapter will begin with the paper as written in its published form, followed by a retrospective section where I will reflect on the paper and discuss how it has interacted with later work in the field. On each of these papers I was a first author.

The first paper ‘Towards a Neural Statistician’ (Edwards and Storkey, 2016) concerns the setting where data points are presented as collections of sets or datasets and is an early example of the deep-learning approach to meta-learning. The premise of transfer learning is that datasets and tasks are more or less related and so by first training on a related task we might improve performance on a target task. In this paper we directly model the space of datasets, aiming to learn a latent space where each point is a dataset or distribution and thereby be able to ask questions like how similar are two datasets? We can also generate new examples from a novel dataset at test time, or sample new datasets. As a concrete example, consider various 1 dimensional distributions that might be introduced in a probability course: the Gaussian, Laplacian, Exponential and Uniform distributions. Each kind of distribution has various parameters like mean and variance that characterize them. Each choice of family and parameters will specify a distribution and if we sample from it a dataset. The Neural Statistician method would learn a representation space where each dataset is a point, and would arrange them in representation space in ways intuitively obvious, the means of the dataset would be one direction, the dataset family (Exponential vs Gaussian for example) would represent well-defined clusters, and so on. The idea to directly represent the space of datasets was a novel contribution of this work. Technically it also demonstrated how to implement neural generative models for processes with hierarchical and non-trivial graphical models.

The second paper 'Large-Scale Study of Curiosity-Driven Learning' (Burda et al., 2018a) concerns the setting where we have access to an environment, be it a simulation of a robot, or an Atari game, but no reward function. We consider a large number of different environments and several intrinsic rewards derived from prediction problems and study the resulting behaviours exhibited by agents optimising these intrinsic rewards. We find that useful behaviors emerge from these intrinsic reward functions, in fact they often end up (in videogames) collecting a nontrivial amount of extrinsic reward, since typically the richest dynamics in a simple game are observed by working towards the game's objective. The main contribution of this work was demonstrating the breadth of behaviors one could learn in a purely unsupervised manner, and set the stage for later work which would aim to quantify the and optimize the extent to which one could learn useful behaviors or representations for downstream tasks.

The third paper 'Exploration by Random Network Distillation' (Burda et al., 2018c) builds on some of the ideas in the second paper. We propose a new prediction task from which derive an exploration bonus designed to avoid problems we observed with stochastic environments in the earlier work. In this paper we consider the sparse-reward setting rather than the no-reward setting, and show how combining the proposed intrinsic reward function with a sparse extrinsic reward function leads to improvements on several environments known to be difficult exploration problems. Prior to this work the state of the art on a particular environment called Montezuma's Revenge had been stagnant for several years, and had come to be synonymous with the difficulty of exploration in reinforcement learning, and so a primary contribution of this work was making progress on a problem widely considered to be difficult by the community. Another contribution was its utility as an easy-to-implement component and baseline. It now features as a standard baseline for new exploration methods, and is an important part of current state of the art methods as a component.

# Chapter 2

# Background

In this chapter I will provide useful background material for understanding the chapters that follow. This thesis concerns approaches for dealing with the difference between the data you wish you had and the data you actually have. Let's first begin by discussing the ideal case.

## 2.1 Supervised Learning

A common task is classification where we map inputs $x \in \mathbb{R}^d$ to a label $y \in \mathbb{R}^l$. In this case the input space may be images of a given resolution, and the label space may consist of 1-hot vectors (vectors taking values in $0, 1$ and having exactly one non-zero entry) where the non-zero element indicates the class of the image. The generative model underlying this mapping in most cases is the result of asking humans to label an input which gives a conditional distribution over the label space for each input $p(y|x)$. In the completely deterministic case this distribution will have all mass on one label, but for more realistic cases there is irreducible randomness resulting from human error, or genuine ambiguity as to which label is most appropriate. In supervised learning we parameterize a classifier, a neural network taking an input to a distribution over labels $p(y|x;\theta)$, and use optimization to find a good value for $\theta$.

The most common approach to optimization is to use a gradient-based optimizer such as Stochastic Gradient Descent (SGD) to minimize the log-loss

$$\mathcal{L}(\theta) = -\mathbb{E}_{x,y} \log p(y|x;\theta) \tag{2.1}$$

where the expectation is taken over the joint distribution over inputs and labels, in practice this means the sample of $(x, y)$ pairs in your training data. If using Gradient

Descent as an optimizer, one would calculate the gradient $\nabla_\theta \mathcal{L}(\theta)$ and follow the rule:

$$\theta_{t+1} = \theta_t - \lambda \nabla_{\theta_t} \mathcal{L}(\theta_t)$$

where $\lambda \in \mathbb{R}^+$ is the learning rate. More likely one would instead use SGD where one uses a Monte-Carlo estimate of the true gradient by sampling a minibatch of examples from the training data.

In the case where you have a large number (millions) of labelled examples in a well-understood modality (text, images) then fitting a deep neural network with parameters $\theta$ using this approach will often give you a reasonable level of performance. For example modern ImageNet classifiers (Deng et al., 2009) are able to achieve a top-1 accuracy of around 90% (Zhai et al., 2021). However this performance will rapidly deteriorate when tested under non-standard conditions that differ from the curated training data (for example with objects not being in the center of the frame.)

In the case where you have a small number of labelled examples then fitting a neural network from scratch on only these labels is not feasible, and alternative approaches must be considered.

## 2.2 Transfer Learning

In some situations you may have a small dataset of interest, for example you may wish to estimate the dimensions of a Welsh corgi from a photo for the purpose of producing a tailored coat for a dog. If you have had thousands of previous customers then you will have thousands of tuples of the form (photos of dog, measured dimensions of dog) which you could train a neural network to predict. Doing this from scratch would be hopeless, but if instead of randomly initializing the parameters of your neural network in the optimization process, you used the parameters of a neural network trained for a related task with more abundant data, then often this will lead to a large gain in performance over the baseline of training from randomly initialized parameters. In this example using a model trained to do ImageNet classification might be a good idea because there are a large number of images of dogs in the dataset. There might not be a large number of Welsh corgis specifically, and the task is classification and not predicting the dimensions of the object, but the intuition is that many of the features useful for doing the more general task can be repurposed for your task.

Typically the transfer learning training process is done in two stages: a *pretraining* phase where a neural network is trained on a task with abundant data from a random initialization, and a *finetuning* phase where the neural network is then fit to a task with scarce data with the previous parameter vector used as an initialization. An alternative to the pretraining then finetuning procedure could be to do multitask training, whereby one trains the network on the mixture of the abundant data and the small dataset. In practice this is not done as often since the computational expense of training on the abundant data is large, and having a parameter vector trained only on the abundant pretraining data can be reused for many different downstream tasks.

The pretraining task is typically broader than the finetuning task. Examples include:

| *Pretrain task* | *Finetune task* |
|---|---|
| Imagenet classification | Corgi pose estimation |
| Next word prediction | Amazon review sentiment prediction |
| Next frame video prediction | Towel folding robot action prediction |

A common source of pretraining tasks comes from data that can be scraped from the internet and social media since it is enormous. For example Common Crawl (com, 2021) is a project that archives a portion of the internet. It contains petabytes of data, and adds several billion pages per month to its collection.

## 2.3 Meta-learning

Instead of pretraining on one task and finetuning on another, one could attempt to optimize directly for quickly learning a new task. Since this can be considered learning how to learn it is often called ‘metalearning’. This can be done if you have a large collection of tasks, typically with small amounts of data per task. In this case we would have training datasets $D_1,\dots,D_n$ where each dataset is a set of tuples $D_i = \{(x_1,y_1),\dots,(x_{N_i},y_{N_i})\}$. At test time we would be given a few labelled examples from a novel task $D_{\text{test}}$ and be asked to make predictions about held-out examples $x_{\text{test}} \notin D_{\text{test}}$ from the same task.

The typical setup for this in metalearning papers is to create few-shot classification datasets, that is image classification tasks with a small number of categories and only a few examples per class. At test time the model will be asked to make predictions about previously unseen object categories.

In order to accomplish this goal a neural network is required that can be rapidly fit to a new task. This can be done in a variety of ways. One straightforward approach is to treat each dataset as a sequence and use architectures capable of dealing with sequences such as recurrent neural networks (Santoro et al., 2016a), (Duan et al., 2016). Another is to observe that the ordering of a dataset is not important and to learn an order-invariant representation appropriate for sets by using symmetric pooling operations (Vinyals et al., 2016b), (Snell et al., 2017). Yet another is to directly optimize for $\theta$ such that taking a few gradient steps with respect to the loss on $D_i$ improves the loss on held out examples from $D_i$, this involves differentiating through a small number of gradient steps (Finn et al., 2017).

Finally although I have presented metalearning as being about learning supervised tasks quickly it can just as easily be applied to generative modelling where the task is to quickly learn a density, or a model capable of producing samples from a distribution, or a policy that quickly learns to maximize reward in a new environment. The reader wishing for a more detailed overview of meta-learning in the deep learning era is referred to Hospedales et al. (2021).

## 2.4 Self-supervised Learning

Self-supervised learning describes a situation where you have a large quantity of unlabelled data and you create tasks through various means to learn representations on this data.

Standard approaches for creating such tasks are to learn generative models of the data. A common approach is to use an autoregressive model that predicts each coordinate of the datum given the previous ones fitting $p(x_{d+1}|x_d,\dots,x_1)$ for each index $d$. The ordering of the data dimensions can be arbitrary but there are often reasonable choices suggested by the modality such as left to right for English text and raster order for images. Examples of such works for images include: NADE (Uria et al., 2016), MADE (Germain et al., 2015), PixelCNN and PixelRNN (Van Oord et al., 2016) and GPT-I (Chen et al., 2020). For text data the GPT papers (Radford et al., 2018), (Radford et al., 2019), (Brown et al., 2020) are prominent examples using the transformer architecture (Vaswani et al., 2017) and there is a long history of using recurrent neural networks as language models (Mikolov et al., 2010). Other kinds of generative models are also possible, one of which, Variational Autoencoders, are treated in the next section.

An alternative approach is create pretext tasks such as: denoising noised images (Vincent et al., 2008), infilling patches of an image (Pathak et al., 2016), rearranging patches of an image into the correct spatial relationship (Noroozi and Favaro, 2016), predicting the color of a black and white image (Larsson et al., 2016), predicting whether an image has been rotated, infilling words (Mikolov et al., 2013) (Devlin et al., 2019) etc. More recent approaches to self-supervised learning typically take a contrastive approach: given a datum $x$, create several views of $x$, $p_1, \dots, p_k$ by applying data augmentations appropriate to the modality (for images this means rotations, crops, etc) then learn an encoder with a representation that is required to be similar for $x$ and $p_i$ and dissimilar to other randomly chosen encoded data points. Examples of this include Oord et al. (2018), Grill et al. (2020), He et al. (2020).

All of these are examples of transfer learning where the pretraining task is generic to the modality, as opposed to doing supervised learning on a different labelled dataset.

## 2.5 Variational Autoencoders

A commonly used class of generative models parameterized by deep neural networks are variational autoencoders Kingma and Welling (2013a), Rezende et al. (2014a). These are latent variable models that jointly learn a distribution $p(x,z;\theta)$ over observed data $x$ and latent variables $z$. Sampling from such a distribution is done by first sampling a latent variable from the prior distribution $z \sim p(z;\theta)$, often a simple isotropic Gaussian distribution with no learnable parameters, then sampling from the conditional distribution $p(x|z;\theta)$. The neural network parameterizing the conditional distribution is called the *decoder*. The variational autoencoder is central to the work in Chapter 3 where it is extended in a hierarchical fashion. Although other generative models could potentially have been used, variational inference handles latent variables in a natural and flexible way that lent itself to this work.

A straightforward approach to fit the parameters of a generative model is called maximum likeihood estimation. The likelihood $\mathcal{L}(\theta)$ of data $x_1, \dots, x_n$ is the density of the data

$$\mathcal{L}(\theta) := p(x_1, \dots, x_n; \theta)$$

which, assuming the data are drawn independently from the distribution, can be written as the product

$$\prod_{i=1}^{n} p(x_i;\theta).$$

Since the logarithm function is monotonic, maximizing the log of likelihood $l(\theta) :=$

$\log(\mathcal{L}(\theta))$ is equivalent to maximizing the likelihood where

$$l(\theta) = \sum_{i=1}^{n} \log p(x_i;\theta)$$

becomes a sum. To optimize the log-likelihood we take gradients of this with respect to $\theta$

$$\nabla_\theta l(\theta) = \sum_{i=1}^{n} \nabla_\theta \log p(x_i;\theta).$$

Hence to fit $\theta$ we just need to compute the gradient of the log-density of a data point.

For latent variable models the log-density $\log p(x;\theta)$ is not straightforward to calculate, but there is a well-known way to compute a lower bound. First we observe that

$$\log(p(x;\theta)) = \log\left(\int_z p(x,z;\theta)\,dz\right)$$

by integrating out the latent variable $z$. Next we decompose the joint distribution

$$\log(p(x;\theta)) = \log\left(\int_z p(x|z;\theta)p(z;\theta)\,dz\right)$$

into the decoder and prior parts. Now we introduce a density $q(z)$ called the approximate posterior

$$\log(p(x;\theta)) = \log\left(\int_z p(x|z;\theta)p(z;\theta)\frac{q(z)}{q(z)}\,dz\right)$$

then rewrite the integral as an expectation under $q$, and apply Jensen's inequality (since the logarithm is concave) to take the logarithm inside the integral

$$\log\left(\int_z \left(p(x|z;\theta)\frac{p(z;\theta)}{q(z)}\right) q(z)\,dz\right) \geq \int_z \log\left(p(x|z;\theta)\frac{p(z;\theta)}{q(z)}\right) q(z)\,dz.$$

The final step is to split the integral into two pieces

$$\int_z \log(p(x|z;\theta))\,q(z)\,dz + \int_z \log\left(\frac{p(z;\theta)}{q(z)}\right) q(z)\,dz.$$

The first term is the expected log-likelihood of $x$ given $z$ where the expectation is taken under $q$. The second term is Kullback-Leibler divergence of $q(z)$ and $p(z;\theta)$. Since this is a lower-bound, maximizing the lower-bound will also increase the log-likeihood. In fact this bound will be maximized when $q(z)$ is equal to the true posterior $p(z|x;\theta)$. Optimizing this term with respect to $q$ must balance two forces, from the first term the pressure to concentrate $q(z)$ over $z$ that decode to $x$, from the second term the pressure to match the prior. The lower-bound given is called the *variational lower bound*. Standard practice before the VAE paper was to learn a parametric $q(z)$ for each data point $x$ separately, the chief innovation of the paper was to introduce amortized variational inference with an encoder network $q(z|x;\phi)$ with parameters $\phi$. This encoder network must produce for $x$ an approximate posterior over $z$, mostly commonly the encoder outputs vectors parameterizing a diagonal Gaussian distribution.

## 2.6 Reinforcement Learning

In reinforcement learning an agent interacts with an environment by applying actions, the environment in turn responds with observations and rewards. For example if the environment were a simple videogame then the observations would be the pixels of the screen rendering it, the actions might be up, down, left, right, A and B and the rewards would be changes to your score.

More formally there is a transition distribution

$$p(x_{t+1}, r_{t+1}|x_t, r_t, a_t, \ldots, x_0, r_0, a_0)$$

where $x_t$ is observation at time $t$, $r_t$ is the reward at time $t$ and $a_t$ is the action at time $t$.

A policy is a distribution over actions for each possible history of actions, observations and rewards

$$\pi(a_{t+1}|x_{t+1}, r_{t+1}, a_t, \ldots, x_0, r_0, a_0).$$

Commonly there is also a termination state like "Game Over" or simply a time limit, and the goal of optimization is to produce a policy $\pi_\theta$ that maximizes the expected sum of rewards. The total reward collected by a policy in an episode is called the return. A summary of the history of the trajectory sufficient for making all future predictions is called the state. A value function is a prediction of the expected future return of a policy starting in a particular state $V_\pi(s)$.

Reinforcement learning is a classic case of the data you have being rather different than the data you wish you had. Typically you wish you had many examples showing exactly which action should be taken in each situation in order to maximize the reward. You could then learn to predict these actions via supervised learning. In this context this is known as behavior cloning. What you have instead is a way of measuring the quality of a trajectory (the rewards), but the relationship of the rewards to the actions taken is opaque.

There are many approaches in the literature for optimizing $\pi$, a straightforward way of proceeding is to use policy gradient methods (and this is the approach we take in all experiments in the thesis, in particular in Chapters 5 and 4). The basic version is simply to optimize the expected reward with SGD. If we use $\tau$ to denote a trajectory and $R(\tau)$ to be the return, then we are trying to maximize

$$\mathbb{E}_{p,\pi}\left[R(\tau)\right]$$

where the expectation is both over the dynamics of the environment $p$ and the policy $\pi$. For convienience let's introduce a notation for the probability of the trajectory

$$P(\tau) := \pi(a_0|s_0) \times p(s_0) \times p(s_1|a_0,s_0) \times p(a_1|s_1) \times \cdots \times p(s_T|s_{T-1},a_{T-1}) \times \pi(a_{T-1}|s_{T-1}),$$

now we can rewrite the expected return as

$$\mathbb{E}_P\left[R(\tau)\right] = \int R(\tau)P(\tau)\,d\tau$$

Now if we let $\theta$ be a parameterization of the policy and take gradients with respect to it we find:

$$\begin{aligned}\nabla_\theta \mathbb{E}_{p,\pi}\left[R(\tau)\right] &= \nabla_\theta \int R(\tau)P(\tau)\,d\tau \\ &= \int \nabla_\theta \left(R(\tau)P(\tau)\right)\,d\tau \\ &= \int R(\tau)\nabla_\theta P(\tau)\,d\tau\end{aligned}$$

noting that we can take the return outside the gradient since $R(\tau)$ has no dependence on the policy. We then proceed to use the 'log derivative trick' ( $x' = x \cdot \log'(x)$)

$$\begin{aligned}&= \int R(\tau)P(\tau)\nabla_\theta \log P(\tau)\,d\tau \\ &= \mathbb{E}_P\left[R(\tau) \cdot \nabla_\theta \log P(\tau)\right].\end{aligned}$$

This is often called the policy gradient theorem. So what have we achieved by this manipulation? The utility is that we have written the gradient as an expectation (under the policy) of the return times the gradient of the log-prob of the trajectory. This means we can approximate the gradient by simply sampling trajectories using the policy (often called doing a rollout) several times and computing the return and gradient of the log-prob of the trajectory.

Policy gradient methods can be contrasted with value-based methods, where instead of optimizing a parameterization of the *policy* one instead learns to predict the *value* (expected return) of each state and action pair under the optimal policy. Since the experiments in this thesis all concern policy gradient methods we won't go into more detail on value-based approaches, but I will mention that there is no clear consensus in the literature about which approach may be better and when, and that it is extremely likely that all the experiments in the thesis could be redone with this approach with little important change to the results.

Also note the difference to supervised learning. In a supervised learning case we would just have a sum of gradients of log-probs of expert trajectories. Here we weight these terms by the returns, and the trajectories themselves are sampled with the policy given by $\theta$. The practical importance of sampling from the current policy is that the variance of the gradient can be extremely large, so that a large batch size, meaning sampling many trajectories, would be required in order to adequately approximate the gradient.

For example imagine an environment which is a simple 1 dimensional grid of states $s_1, \ldots, s_N$ and two actions 'left' and 'right' which respectively decrement and increment your state index. This has the topology of a line segment with boundaries at $s_1$ and $s_N$. Further suppose the environment starts in state $s_1$, an episode lasts $N$ timesteps, and the reward is 1 when the state is $s_N$ and is zero otherwise. Then the only trajectory that achieves a non-zero reward is one where every action is 'right'.

If we assume that the policy is initialized to be uniformly random, then we can see that the probability of sampling a sequence with a positive reward is $2^{-N}$. All other trajectories contribute nothing to the policy gradient integral, since they have zero reward. Any monte-carlo approximation to the gradient then will be highly sensitive to whether we sampled that trajectory or not, and to expect to see such a trajectory we will need on the order of $2^N$ trajectories. This simple example illustrates what is also called the exploration problem.

The papers in this thesis use a variant of the policy gradient called 'Proximal Policy Optimization' Schulman et al. (2017). This method uses a learned value function as a baseline to reduce the variance of the gradient, and it adds clipping of the policy to prevent the policy from diverging too much from the previous iteration of the policy during optimization. The clipping increases the stability of optimization.

# Chapter 3

# Towards a Neural Statistician

## 3.1 Introduction

The machine learning community is well-practised at learning representations of *data-points* and *sequences*. A middle-ground between these two is representing, or summarizing, *datasets* - unordered collections of vectors, such as photos of a particular person, recordings of a given speaker or a document as a bag-of-words. Where these sets take the form of i.i.d samples from some distribution, such summaries are called *statistics*. We explore the idea of using neural networks to learn statistics and we refer to our approach as a *neural statistician*.

The key result of our approach is a *statistic network* that takes as input a set of vectors and outputs a vector of summary statistics specifying a generative model of that set - a mean and variance specifying a Gaussian distribution in a latent space we term the context. The advantages of our approach are that it is:

- *Unsupervised*: It provides principled and unsupervised way to learn summary statistics as the output of a variational encoder of a generative model.
- *Data efficient*: If one has a large number of small but related datasets, modelling the datasets jointly enables us to gain statistical strength.
- *Parameter Efficient*: By using summary statistics instead of say categorical labellings of each dataset, we decouple the number of parameters of the model from the number of datasets.
- *Capable of few-shot learning*: If the datasets correspond to examples from different classes, *class embeddings* (summary statistics associated with examples

from a class), allow us to handle new classes at test time.

## 3.2 Problem Statement

We are given datasets $D_i$ for $i \in I$. Each dataset $D_i = \{x_1, \dots, x_{k_i}\}$ consists of a number of i.i.d samples from an associated distribution $p_i$ over $\mathbb{R}^n$. The task can be split into learning and inference components. The learning component is to produce a generative model $\hat{p}_i$ for each dataset $D_i$. We assume there is a common underlying generative process $p$ such that $p_i = p(\cdot|c_i)$ for $c_i \in \mathbb{R}^l$ drawn from $p(c)$. We refer to $c$ as the *context*. The inference component is to give an approximate posterior over the context $q(c|D)$ for a given dataset produced by a *statistic network*.

## 3.3 Neural Statistician

In order to exploit the assumption of a hierarchical generative process over datasets we will use a 'parameter-transfer approach' (see Pan and Yang, 2010) to extend the variational autoencoder model of Kingma and Welling (2013a).

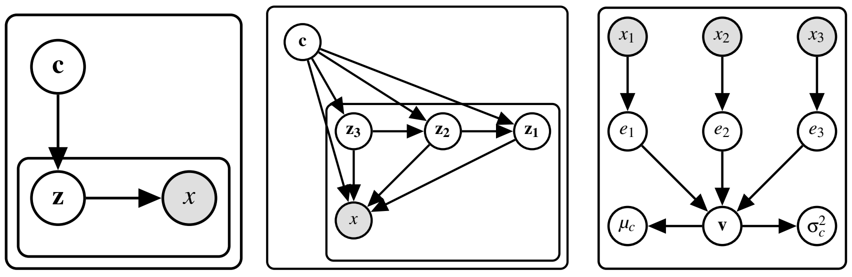


Figure 3.1: *Left*: basic hierarchical model, where the plate encodes the fact that the context variable $c$ is shared across each item in a given dataset. *Center*: full neural statistician model with three latent layers $z_1, z_2, z_3$. Each collection of incoming edges to a node is implemented as a neural network, the input of which is the concatenation of the edges' sources, the output of which is a parameterization of a distribution over the random variable represented by that node. *Right*: The statistic network, which combines the data via an exchangeable statistic layer.

### 3.3.1 Variational Autoencoder

The variational autoencoder is a latent variable model $p(x|z;\theta)$ (often called the decoder) with parameters $\theta$. For each observed $x$, a corresponding latent variable $z$ is drawn from

$p(z)$ so that

$$p(x) = \int p(x|z;\theta)p(z)\,dz. \tag{3.1}$$

The generative parameters $\theta$ are learned by introducing a recognition network (also called an encoder) $q(z|x;\phi)$ with parameters $\phi$. The recognition network gives an approximate posterior over the latent variables that can then be used to give the standard variational lower bound (Saul and Jordan, 1996) on the single-datum log-likelihood. I.e. $\log P(x|\theta) \geq \mathcal{L}_x$, where

$$\mathcal{L}_x = \mathbb{E}_{q(z|x,\phi)}\left[\log p(x|z;\theta)\right] - D_{KL}\left(q(z|x;\phi)\|p(z)\right). \tag{3.2}$$

Likewise the full-data log likelihood is lower bounded by the sum of the $\mathcal{L}_x$ terms over the whole dataset. We can then optimize this lower bound with respect to $\phi$ and $\theta$ using the reparameterization trick introduced by Kingma and Welling (2013a) and Rezende et al. (2014b) to get a Monte-Carlo estimate of the gradient.

### 3.3.2 Basic Model

We extend the variational autoencoder to the model depicted on the left in Figure 3.1. This includes a latent variable $c$, the context, that varies between different datasets but is constant, a priori, for items within the same dataset. Now, the likelihood of the parameters $\theta$ for one single particular dataset $D$ is given by

$$p(D) = \int p(c)\left[\prod_{x\in D}\int p(x|z;\theta)p(z|c;\theta)\,dz\right]dc. \tag{3.3}$$

The prior $p(c)$ is chosen to be a spherical Gaussian with zero mean and unit variance. The conditional $p(z|c;\theta)$ is Gaussian with diagonal covariance, where all the mean and variance parameters depend on $c$ through a neural network. Similarly the observation model $p(x|z;\theta)$ will be a simple likelihood function appropriate to the data modality with dependence on $z$ parameterized by a neural network. For example, with real valued data, a diagonal Gaussian likelihood could be used where the mean and log variance of $x$ are created from $z$ via a neural network.

We use approximate inference networks $q(z|x,c;\phi)$, $q(c|D;\phi)$, with parameters collected into $\phi$, to once again enable the calculation and optimization of a variational lower bound on the log-likelihood. The single dataset log likelihood lower bound is given by

$$\mathcal{L}_D = \mathbb{E}_{q(c|D;\phi)}\left[\sum_{x\in d}\mathbb{E}_{q(z|c,x;\phi)}\left[\log p(x|z;\theta)\right] - D_{KL}\left(q(z|c,x;\phi)\|p(z|c;\theta)\right)\right] \\ - D_{KL}\left(q(c|D;\phi)\|p(c)\right). \quad (3.4)$$

As with the generative distributions, the likelihood forms for $q(z|x,c;\phi)$ and $q(c|D;\phi)$ are diagonal Gaussian distributions, where all the mean and log variance parameters in each distribution are produced by a neural network taking the conditioning variables as inputs. Note that $q(c|D;\phi)$ accepts as input a *dataset* $D$ and we refer to this as the statistic network. We describe this in Subsection 3.3.4.

The full-data variational bound is given by summing the variational bound for each dataset in our collection of datasets. It is by learning the difference of the within-dataset and between-dataset distributions that we are able to discover an appropriate statistic network.

### 3.3.3 Full Model

The basic model works well for modelling simple datasets, but struggles when the datasets have complex internal structure. To increase the sophistication of the model we use multiple stochastic layers $z_1,\ldots,z_k$ and introduce skip-connections for both the inference and generative networks. The generative model is shown graphically in Figure 3.1 in the center. The probability of a dataset $D$ is then given by

$$p(D) = \int p(c)\prod_{x\in D}\int p(x|c,z_{1:L};\theta)p(z_L|c;\theta)\prod_{i=1}^{L-1}p(z_i|z_{i+1},c;\theta)\,dz_{1:L}\,dc \quad (3.5)$$

where the $p(z_i|z_{i+1},c,\theta)$ are again Gaussian distributions where the mean and log variance are given as the output of neural networks. The generative process for the full model is described in Algorithm 1.

The full approximate posterior factorizes analogously as

$$q(c,z_{1:L}|D;\phi) = q(c|D;\phi)\prod_{x\in D}q(z_L|x,c;\phi)\prod_{i=1}^{L-1}q(z_i|z_{i+1},x,c;\phi). \quad (3.6)$$

For convenience we give the variational lower bound as sum of a three parts, a reconstruction term $R_D$, a context divergence $C_D$ and a latent divergence $L_D$:

$$\mathcal{L}_D = R_D + C_D + L_D \text{ with} \tag{3.7}$$

$$R_D = \mathbb{E}_{q(c|D;\phi)} \sum_{x \in D} \mathbb{E}_{q(z_{1:L}|c,x;\phi)} \log p(x|z_{1:L}, c; \theta) \tag{3.8}$$

$$C_D = D_{KL}\left(q(c|D;\phi) \| p(c)\right) \tag{3.9}$$

$$L_D = \mathbb{E}_{q(c,z_{1:L}|D;\phi)} \left[ \sum_{x \in D} D_{KL}\left(q(z_L|c,x;\phi) \| p(z_L|c;\theta)\right) + \sum_{i=1}^{L-1} D_{KL}\left(q(z_i|z_{i+1}, c, x; \phi) \| p(z_i|z_{i+1}, c; \theta)\right) \right]. \tag{3.10}$$

The skip-connections $p(z_i|z_{i+1}, c; \theta)$ and $q(z_i|z_{i+1}, x; \phi)$ allow the context to specify a more precise distribution for each latent variable by explaining-away more generic aspects of the dataset at each stochastic layer. This architecture was inspired by recent work on probabilistic ladder networks in Kaae Sønderby et al. (2016). Complementing these are the skip-connections from each latent variable to the observation $p(x|z_{1:L}, c; \theta)$, the intuition here is that each stochastic layer can focus on representing a certain level of abstraction, since its information does not need to be copied into the next layer, a similar approach was used in Maaløe et al. (2016).

Once again, note that we are maximizing the lower bound to the log likelihood over many datasets $D$: we want to maximize the expectation of $\mathcal{L}_D$ over all datasets. We do this optimization using stochastic gradient descent. In contrast to a variational autoencoder where a minibatch would consist of a subsample of *datapoints* from the dataset, we use minibatches consisting of a subsample of *datasets* - tensors of shape `(batch size, sample size, number of features)`.

### 3.3.4 Statistic Network

In addition to the standard inference networks we require a *statistic network* $q(c|D;\phi)$ to give an approximate posterior over the context $c$ given a dataset $D = \{x_1, \ldots, x_k\}$ . This inference network must capture the exchangeability of the data in $D$.

We use a feedforward neural network consisting of three main elements:

- An instance encoder $E$ that takes each individual datapoint $x_i$ to a vector $e_i = E(x_i)$.

- An exchangeable instance pooling layer that collapses the matrix $(e_1, \ldots, e_k)$ to a single pre-statistic vector $v$. Examples include elementwise means, sums, products, geometric means and maximum. We use the sample mean for all experiments.

- A final post-pooling network that takes $v$ to a parameterization of a diagonal Gaussian.

The graphical model for this is given at the right of Figure 3.1.

We note that the humble sample mean already gives the statistic network a great deal of representational power due to the fact that the instance encoder can learn a representation where averaging makes sense. For example since the instance encoder can approximate a polynomial on a compact domain, and so can the post-pooling network, a statistic network can approximate any moment of a distribution.

## 3.4 Related Work

Due to the general nature of the problem considered, our work touches on many different topics which we now attempt to summarize.

**Topic models and graphical models** The form of the graphical model in Figure 3.1 on the left is equivalent to that of a standard topic model. In contrast to traditional topic models we do not use discrete latent variables, or restrict to discrete data. Work such as that by Ranganath et al. (2014) has extended topic models in various directions, but importantly we use flexible conditional distributions and dependency structures parameterized by deep neural networks. Recent work has explored neural networks for document models (see e.g. Miao et al., 2016) but has been limited to modelling datapoints with little internal structure. Along related lines are 'structured variational autoencoders' (see Johnson et al., 2016), where they treat the general problem of integrating graphical models with variational autoencoders.

**Transfer learning** There is a considerable literature on transfer learning, for a survey see Pan and Yang (2010). There they discuss 'parameter-transfer' approaches whereby parameters or priors are shared across datasets, and our work fits into that paradigm. For examples see Lawrence and Platt (2004) where share they priors between Gaussian processes, and Evgeniou and Pontil (2004) where they take an SVM-like approach to share kernels.

**One-shot Learning** Learning quickly from small amounts of data is a topic of great interest. Lake et al. (2015) use Bayesian program induction for one-shot generation and classification, and Koch (2015) train a Siamese (Chopra et al. (2005)) convolutional network for one-shot image classification. We note the relation to the recent work (Rezende et al., 2016) in which the authors use a *conditional* recurrent variational autoencoder capable of one-shot generalization by taking as extra input a conditioning data point. The important differences here are that we *jointly* model datasets and datapoints and consider datasets of any size. Recent approaches to one-shot classification are matching networks (Vinyals et al., 2016b) (which was concurrent with the initial preprint of this work), and related previous work (Santoro et al., 2016b). The former can be considered a kind of differentiable nearest neighbour classifier, and the latter augments their network with memory to store information about the classification problem. Both are trained end-to-end for the classification problem, whereas the present work is a general approach to learning representations of datasets. Probably the closest previous work is by Salakhutdinov et al. (2012) where the authors learn a topic model over the activations of a DBM for one-shot learning. Compared with their work we use modern architectures and easier to train VAEs, in particular we have fast and amortized feedforward inference for test (and training) datasets, avoiding the need for MCMC.

**Multiple-Instance Learning** There is previous work on classifying sets in multiple-instance learning, for a useful survey see Cheplygina et al. (2015). Typical approaches involve adapting kernel based methods such as *support measure machines* (Muandet et al., 2012), *support distribution machines* (Póczos et al., 2012) and *multiple-instance-kernels* (Gartner et al., 2002). We do not consider applications to multiple-instance learning type problems here, but it may be fruitful to do so in the future.

**Set2Seq** In very related work, Vinyals et al. (2016a) explore architectures for mapping sets to sequences. There they use an LSTM to repeatedly compute weighted-averages of the datapoints and use this to tackle problems such as sorting a list of numbers. The main difference between their work and ours is that they primarily consider supervised problems, whereas we present a general unsupervised method for learning representations of sets of i.i.d instances. In future work we may also explore recurrently computing statistics.

**ABC** There has also been work on learning summary statistics for Approximate Bayesian Computation by either learning to predict the parameters generating a sample as a supervised problem, or by using kernel embeddings as infinite dimensional summary statistics. See the work by Fukumizu et al. (2013) for an example of kernel-based approaches. More recently Jiang et al. (2017) used deep neural networks to predict the parameters generating the data. The crucial differences are that their problem is supervised, they do not leverage any exchangeability properties the data may have, nor can it deal with varying sample sizes.

## 3.5 Experimental Results

Given an input set $x_1, \dots x_k$ we can use the statistic network to calculate an approximate posterior over contexts $q(c|x_1, \dots, x_k; \phi)$. Under the generative model, each context $c$ specifies a conditional model $p(x|c; \theta)$. To get samples from the model corresponding to the most likely posterior value of $c$, we set $c$ to the mean of the approximate posterior and then sample directly from the conditional distributions. This is described in Algorithm 2. We use this process in our experiments to show samples. In all experiments, we use the Adam optimization algorithm (Kingma and Ba, 2014) to optimize the parameters of the generative models and variational approximations. Batch normalization (Ioffe and Szegedy, 2015a) is implemented for convolutional layers and we always use a batch size of 16. We primarily use the Theano (Theano Development Team, 2016) framework with the Lasagne (Dieleman et al., 2015) library, but the final experiments with face data were done using Tensorflow (Abadi et al., 2015). In all cases experiments were terminated after a given number of epochs when training appeared to have sufficiently converged (300 epochs for omniglot, youtube and spatial MNIST examples, and 50 epochs for the synthetic experiment).

### 3.5.1 Simple 1-D Distributions

In our first experiment we wanted to know if the neural statistician will learn to cluster synthetic 1-D datasets by distribution family. We generated a collection of synthetic 1-D datasets each containing 200 samples. Datasets consist of samples from either an Exponential, Gaussian, Uniform or Laplacian distribution with equal probability. Means and variances are sampled from $U[-1, 1]$ and $U[0.5, 2]$ respectively. The training data contains $10K$ sets.

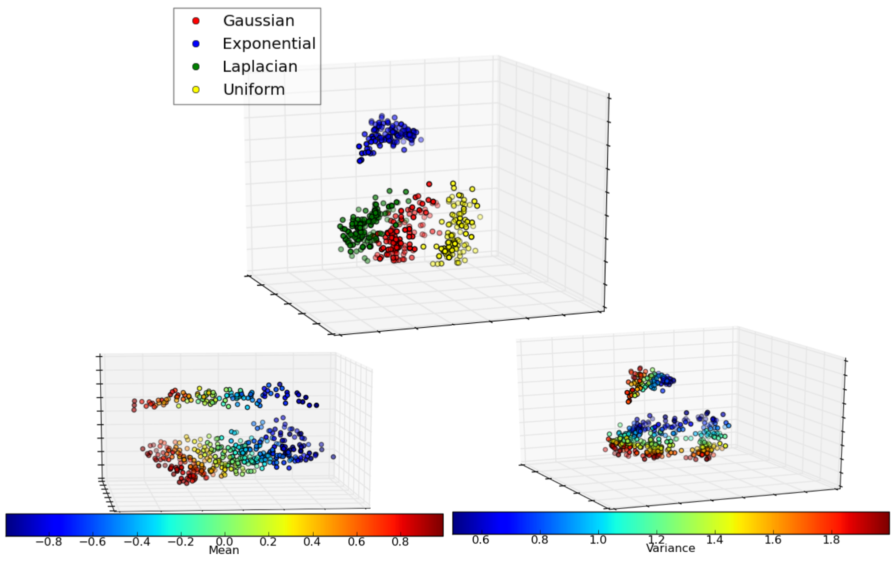


Figure 3.2: Three different views of the same data. Each point is the mean of the approximate posterior over the context $q(c|D;\phi)$ where $c \in \mathbb{R}^3$. Each point is a summary statistic for a single dataset with 200 samples. Top plot shows points colored by distribution family, left plot colored by the mean and right plot colored by the variance. The plots have been rotated to illustrative angles.

The architecture for this experiment contains a single stochastic layer with 32 units for $z$ and 3 units for $c$, . The model $p(x|z,c;\theta)$ and variational approximation $q(z|x,c;\phi)$ are each a diagonal Gaussian distribution with all mean and log variance parameters given by a network composed of three dense layers with ReLU activations and 128 units. The statistic network determining the mean and log variance parameters of posterior over context variables is composed of three dense layers before and after pooling, each with 128 units with Rectified Linear Unit (ReLU) activations.

Figure 3.2 shows 3-D scatter plots of the summary statistics learned. Notice that the different families of distribution cluster. It is interesting to observe that the Exponential cluster is differently orientated to the others, perhaps reflecting the fact that it is the only non-symmetric distribution. We also see that between the Gaussian and Laplacian clusters there is an area of ambiguity which is as one might expect. We also see that within each cluster the mean and variance are mapped to orthogonal directions.

### 3.5.2 Spatial MNIST

Building on the previous experiments we investigate 2-D datasets that have complex structure, but the datapoints contain little information by themselves, making it a good

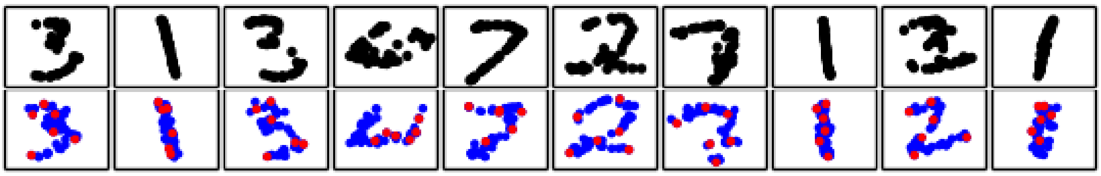


Figure 3.4: Conditioned samples from spatial MNIST data. Blue and red digits are the input sets, black digits above correspond to samples given the input. Red points correspond to a 6-sample summary of the dataset

test of the statistic network. We created a dataset called *spatial MNIST*. In spatial MNIST each image from MNIST (LeCun et al., 1998) is turned into a dataset by interpreting the normalized pixel intensities as a probability density and sampling *coordinate values*. An example is shown in Figure 3.3. This creates two-dimensional spatial datasets. We used a sample size of 50. Note that since the pixel coordinates are discrete, it is necessary to dequantize them by adding uniform noise $u \sim U[0,1]$ to the coordinates if one models them as real numbers, else you can get arbitrarily high densities (see Theis et al. (2016) for a discussion of this point).

The generative architecture for this experiment contains 3 stochastic $z$ layers, each with 2 units, and a single $c$ layer with 64 units. The means and log variances of the Gaussian likelihood for $p(x|z_{1:3}, c; \theta)$, and each subnetwork for $z$ in both the encoder and decoder contained 3 dense layers with 256 ReLU units each. The statistic network also contained 3 dense layers pre-pooling and 3 dense layers post pooling with 256 ReLU units.

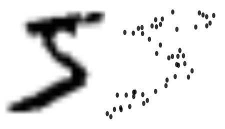


Figure 3.3: An *image* from MNIST on the left, transformed to a *set* of 50 $(x,y)$ coordinates, shown as a scatter plot on the right.

In addition to being able to sample from the model conditioned on a set of inputs, we can also summarize a dataset by choosing a subset $S \subseteq D$ to minimise the KL divergence of $q(C|D;\phi)$ from $q(C|S;\phi)$. We do this greedily by iteratively discarding points from the full sample. Pseudocode for this process is given in Algorithm 3. The results are shown in Figure 3.4. We see that the model is capable of handling complex arrangements of datapoints. We also see that it can select sensible subsets of a dataset as a summary.

### 3.5.3 Omniglot

Next we work with the OMNIGLOT data (Lake et al., 2015). This contains 1628 classes of handwritten characters but with just 20 examples per class. This makes it an excellent test-bed for transfer / few-shot learning. We constructed datasets by splitting each class into datasets of size 5. We train on datasets drawn from 1200 classes and reserve the remaining classes to test few-shot sampling and classification. We created new classes by rotating and reflecting characters. We resized the images to $28 \times 28$. We sampled a binarization of each image for each epoch. We also randomly applied the dilation operator from computer vision as further data augmentation since we observed that the stroke widths are quite uniform in the OMNIGLOT data, whereas there is substantial variation in MNIST, this augmentation improved the visual quality of the few-shot MNIST samples considerably and increased the few-shot classification accuracy by about 3 percent. Finally we used 'sample dropout' whereby a random subset of each dataset was removed from the pooling in the statistic network, and then included the number of samples remaining as an extra feature. This was beneficial since it reduced overfitting and also allowed the statistic network to learn to adjust the approximate posterior over $c$ based on the number of samples.

We used a single stochastic layer with 16 units for $z$, and 512 units for $c$. We used a shared convolutional encoder between the inference and statistic networks and a deconvolutional decoder network. Full details of the networks are given in Appendix A.2.1. The decoder used a Bernoulli likelihood.

In Figure 3.5 we show two examples of few-shot learning by conditioning on samples of *unseen* characters from OMNIGLOT, and conditioning on samples of digits from MNIST. The samples are mostly of a high-quality, and this shows that the neural statistician can generalize even to new datasets.

As a further test we considered few-shot classification of both unseen OMNIGLOT characters and MNIST digits. Given a sets of labelled examples of each class $D_0, \ldots, D_9$ (for MNIST say), we computed the approximate posteriors $q(C|D_i;\phi)$ using the statistic network. Then for each test image $x$ we also computed the posterior $q(C|x;\phi)$ and classified it according to the training dataset $D_i$ minimizing the KL divergence *from* the test context *to* the training context. This process is described in Algorithm 4. We tried this with either 1 or 5 labelled examples per class and either 5 or 20 classes. For each trial we randomly select $K$ classes, randomly select training examples for each class, and test on the remaining examples. This process is repeated 100 times and the

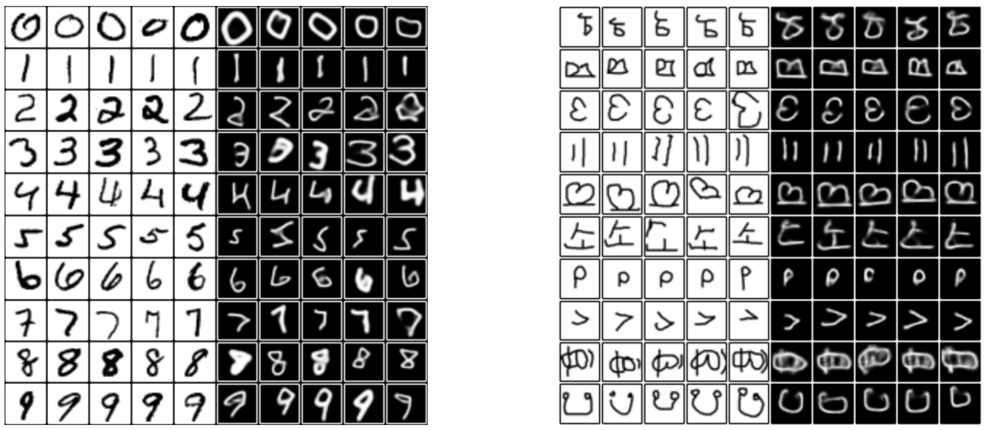


Figure 3.5: Few-shot learning *Left*: Few-shot learning from OMNIGLOT to MNIST. Left rows are input sets, right rows are samples given the inputs. *Right*: Few-shot learning from with OMNIGLOT data to unseen classes. Left rows are input sets, right rows are samples given the inputs. Black-white inversion is applied for ease of viewing.

results averaged. The results are shown in Table 3.1. We compare to a number of results reported in Vinyals et al. (2016b) including Santoro et al. (2016b) and Koch (2015). Overall we see that the neural statistician model can be used as a strong classifier, particularly for the 5-way tasks, but performs worse than matching networks for the 20-way tasks. One important advantage that matching networks have is that, whilst each class is processed independently in our model, the representation in matching networks is conditioned on all of the classes in the few-shot problem. This means that it can exaggerate differences between similar classes, which are more likely to appear in a 20-way problem than a 5-way problem.

### 3.5.4 Youtube Faces

Finally, we provide a proof of concept for generating faces of a particular person. We use the Youtube Faces Database from Wolf et al. (2011). It contains $3,245$ videos of $1,595$ different people. We use the aligned and cropped to face version, resized to $64 \times 64$. The validation and test sets contain 100 unique people each, and there is no overlap of persons between data splits. The sets were created by sampling frames randomly without replacement from each video, we use a set size of 5 frames. We resample the sets for the training data each epoch.

Our architecture for this problem is based on one presented in Lamb et al. (2016). We used a single stochastic layer with 500 dimensional latent $c$ and 16 dimensional $z$ variable. The statistic network and the inference network $q(z|x,c;\phi)$ share a common

| Task | | | Method | | | |
|---|---|---|---|---|---|---|
| Test Dataset | K Shot | K Way | Siamese | MANN | Matching | Ours |
| MNIST | 1 | 10 | 70 | - | 72 | **78.6** |
| MNIST | 5 | 10 | - | - | - | 93.2 |
| OMNIGLOT | 1 | 5 | 97.3 | 82.8 | **98.1** | **98.1** |
| OMNIGLOT | 5 | 5 | 98.4 | 94.9 | 98.9 | **99.5** |
| OMNIGLOT | 1 | 20 | 88.1 | - | **93.8** | 93.2 |
| OMNIGLOT | 5 | 20 | 97.0 | - | **98.7** | 98.1 |

Table 3.1: The table shows the classification accuracies of various few-shot learning tasks. Models are trained on OMNIGLOT data and tested on either unseen OMNIGLOT classes or MNIST with varying numbers of samples per class (K-shot) with varying numbers of classes (K-way). Comparisons are to Vinyals et al. (2016b) (Matching), Santoro et al. (2016b) (MANN) and Koch (2015) (Siamese). 5-shot MNIST results are included for completeness.

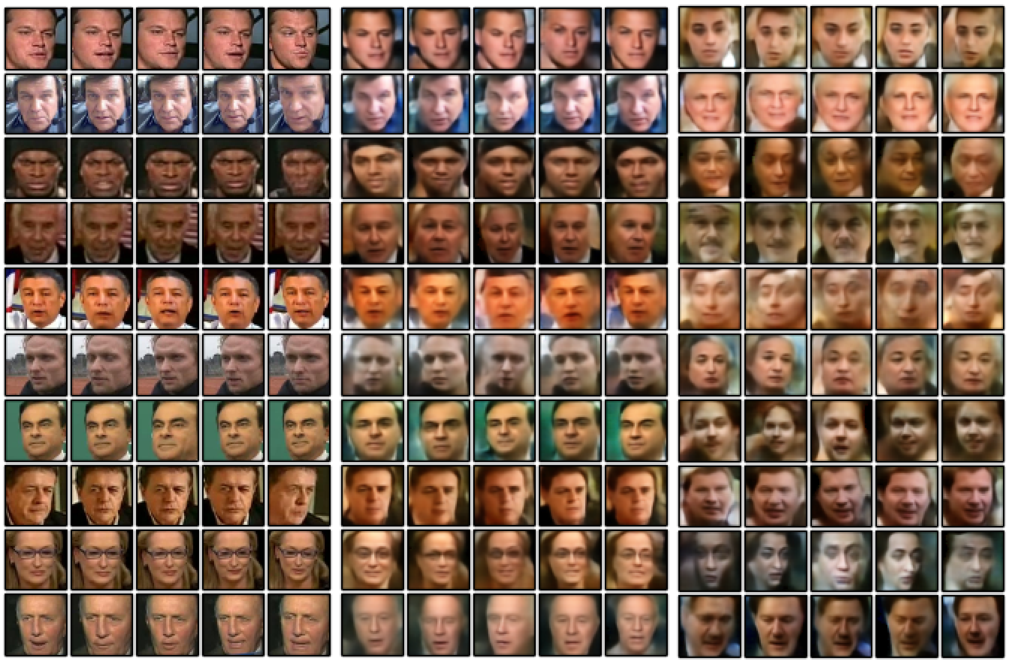


Figure 3.6: Few-shot learning for face data. Samples are from model trained on Youtube Faces Database. *Left*: Each row shows an input set of size 5. *Center*: Each row shows 5 samples from the model corresponding to the input set on the left. *Right*: Imagined new faces generated by sampling contexts from the prior. Each row consists of 5 samples from the model given a particular sampled context.

convolutional encoder, and the deocder uses deconvolutional layers. For full details see Appendix A.2.2. The likelihood function is a Gaussian, but where the variance parameters are shared across all datapoints, this was found to make training faster and more stable.

The results are shown in Figure 3.6. Whilst there is room for improvement, we see that it is possible to specify a complex distribution on-the-fly with a set of photos of a previously unseen person. The samples conditioned on an input set have a reasonable likeness of the input faces. We also show the ability of the model to generate new datasets and see that the samples have a consistent identity and varied poses.

## 3.6 Conclusion

We have demonstrated a highly flexible model on a variety of tasks. Going forward our approach will naturally benefit from advances in generative models as we can simply upgrade our base generative model, and so future work will pursue this. Compared with some other approaches in the literature for few-shot learning, our requirement for supervision is weaker: we only ask at training time that we are given datasets, but we do not need labels for the datasets, nor even information on whether two datasets represent the same or different classes. It would be interesting then to explore application areas where only this weaker form of supervision is available. There are two important limitations to this work, firstly that the method is dataset hungry: it will likely not learn useful representations of datasets given only a small number of them. Secondly at test time the few-shot fit of the generative model will not be greatly improved by using larger datasets unless the model was also trained on similarly large datasets. The latter limitation seems like a promising future research direction - bridging the gap between fast adaptation and slow training.

### Acknowledgments

This work was supported in part by the EPSRC Centre for Doctoral Training in Data Science, funded by the UK Engineering and Physical Sciences Research Council (grant EP/L016427/1) and the University of Edinburgh.

## 3.7 2021 Retrospective

In the paper the motivation and experiments were mostly discussed in abstract terms or in terms of commonly used academic benchmarks (Omniglot for example), but it may be more useful to consider a more vivid and practical vision. A holy grail in robotics has long been a practical domestic assistant, a general purpose machine that doesn't assume a special and static layout of a house, doesn't assume special amenities, one that is safe around pets and children, a machine that can do the dishes, make a cup of coffee, do the laundry and vacuum. Every house, every apartment, every room has its own unique layout, both in terms of where say the downstairs bathroom is, and in terms of where the remote control for a TV may reside that day. In this sense every home is its own dataset, and for a general purpose robot to work it must be capable of explicitly or implicitly inferring the parameters describing the house in order to do its tasks. In this setting we can imagine using the Neural Statistician approach to learn representations of homes which would have a number of uses. Firstly by learning the relations amongst homes, the representation for a new home could be learned with a small amount of data. Secondly the uncertainty in the distribution over the home latent variable could be used to guide an exploration policy that may be used when a robot first arrives in a new home, or do a survey of the home's current state. Thirdly the representation itself could be used to condition a robot policy, or perhaps to condition a world model of the dynamics that could be used to do planning. Finally generation of novel home latent variables could be used to create synthetic training data for training robot policies.

This paper was one of the early approaches to meta-learning using deep learning, and, as it was put on arxiv more than four years ago at the time of writing, it is interesting to reflect on where the field has gone since then. In the paper I considered the most interesting part to be the idea of learning representations of datasets rather than datapoints as is usual practice. As such I emphasised qualitative investigations of these representations such as visualising the representations and looking for expected factors of variation, and showing few-shot generation samples on unseen visual datasets. The results on few-shot classification were added for completeness. Since then the field has focused heavily on few-shot classification benchmarks and less on few-shot generation.

The most direct follow-up work was the 'Variational Homoencoder' (Hewitt et al., 2018) that, amongst other contributions, shows how to modify the objective to deal with datasets that are too large to be passed through the neural network all at once at training time, and instead makes a stochastic lower bound that only requires sub-samples of datasets. Another line of work building on the Neural Statistician are 'Neural Processes' (Garnelo et al., 2018b), (Garnelo et al., 2018a), (Kim et al., 2018) which can be a seen as a natural generalisation to the case of learning representations of *functions* rather than sets.

I think that one of the most interesting prospects for structured representations or latent variables could be in situations where one has a distribution over environments. A simple example could be distribution over 2d mazes that an agent can move through. One could apply a similar approach to learning a generative model of forward dynamics, that is a mapping from a history of states and actions to a distribution over the next state. By doing so one could have a latent variable representing the specifics of one maze versus another. An approximate posterior over such a latent variable could provide the basis for a principled method for 'meta-exploration', that is where the task is to quickly explore a novel environment at test-time. For example if $D_t$ is the information collected from the environment over the first $t$ timesteps, then a divergence between the approximate posteriors $q(C|D_{t+1})$ and $q(C|D_t)$, where $C$ is the latent variable representing the environment, would be a measurement of the information gained about the environment, in our example it would be information about the layout of the maze.

# Chapter 4

# Large-Scale Study of Curiosity-Driven Learning

## 4.1 Introduction

Reinforcement learning (RL) has emerged as a popular method for training agents to perform complex tasks. In RL, the agent policy is trained by maximizing a reward function that is designed to align with the task. The rewards are extrinsic to the agent and specific to the environment they are defined for. Most of the success in RL has been achieved when this reward function is dense and well-shaped, e.g., a running “score” in a video game (Mnih et al., 2015a). However, designing a well-shaped reward function is a notoriously challenging engineering problem. An alternative to “shaping” an extrinsic reward is to supplement it with dense intrinsic rewards (Oudeyer and Kaplan, 2009), that is, rewards that are generated by the agent itself. Examples of intrinsic reward include “curiosity” (Mohamed and Rezende, 2015; Schmidhuber, 1991b; Singh et al., 2005; Houthooft et al., 2016; Pathak et al., 2017) which uses prediction error as reward signal, and “visitation counts” (Bellemare et al., 2016; Ostrovski et al., 2018a; Poupart et al., 2006; Lopes et al., 2012) which discourages the agent from revisiting the same states. The idea is that these intrinsic rewards will bridge the gaps between sparse extrinsic rewards by guiding the agent to efficiently explore the environment to find the next extrinsic reward.

But what about scenarios with no extrinsic reward at all? This is not as strange as it sounds. Developmental psychologists talk about *intrinsic motivation* (i.e., curiosity) as the primary driver in the early stages of development (Smith and Gasser, 2005; Ryan, 2000): babies appear to employ goal-less exploration to learn skills that will be useful

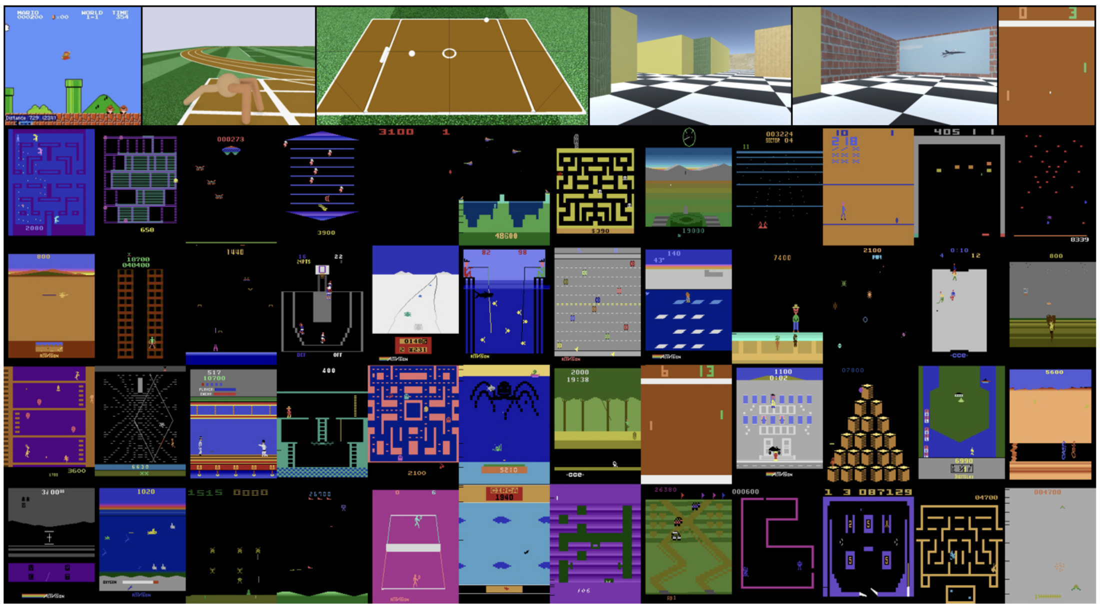


Figure 4.1: A snapshot of the 54 environments investigated in the paper. We show that agents are able to make progress using no extrinsic reward, or end-of-episode signal, and only using curiosity. Video results, code and models at `https://pathak22.github.io/large-scale-curiosity/`.

later on in life. There are plenty of other examples, from playing Minecraft to visiting your local zoo, where no extrinsic rewards are required. Indeed, there is evidence that pre-training an agent on a given environment using only intrinsic rewards allows it to learn much faster when fine-tuned to a novel task in a novel environment (Pathak et al., 2017, 2018). Yet, so far, there has been no systematic study of learning with only intrinsic rewards.

In this paper, we perform a large-scale empirical study of agents driven purely by intrinsic rewards across a range of diverse simulated environments. In particular, we choose the dynamics-based curiosity model of intrinsic reward presented in Pathak et al. (2017) because it is scalable and trivially parallelizable, making it ideal for large-scale experimentation. The central idea is to represent intrinsic reward as the error in predicting the consequence of the agent's action given its current state, i.e., the prediction error of learned forward-dynamics of the agent. We thoroughly investigate the dynamics-based curiosity across 54 environments: video games, physics engine simulations, and virtual 3D navigation tasks, shown in Figure 4.1.

To develop a better understanding of curiosity-driven learning, we further study the crucial factors that determine its performance. In particular, predicting the future state in the high dimensional raw observation space (e.g., images) is a challenging problem and, as shown by recent works (Pathak et al., 2017; Stadie et al., 2015), learning dynamics in an auxiliary feature space leads to improved results. However, how one should choose such an embedding space is a critical, yet open research problem. Through a systematic ablation, we examine the role of different ways to encode agent's observation such that an agent can perform well driven purely by its own curiosity. To ensure stable online training of dynamics, we argue that the desired embedding space should: (a) be *compact* in terms of dimensionality, (b) preserve *sufficient* information about the observation, and (c) be a *stationary* function of the observations. We show that encoding observations via a random network turn out to be a simple, yet effective technique for modeling curiosity across many popular RL benchmarks. This might suggest that many popular RL video game test-beds are not as visually sophisticated as commonly thought. Interestingly, we discover that although random features are sufficient for good performance at training, the learned features appear to generalize better (e.g., to novel game levels in Super Mario Bros.).

In summary: (a) We perform a large-scale study of curiosity-driven exploration across a variety of environments including: the set of Atari games (Bellemare et al., 2013), Super Mario Bros., virtual 3D navigation in Unity (Juliani et al., 2018), multi-player Pong, and Roboschool (Schulman et al., 2017) environments. (b) We extensively investigate different feature spaces for learning the dynamics-based curiosity: random features, pixels, inverse-dynamics (Pathak et al., 2017) and variational auto-encoders (Kingma and Welling, 2013b) and evaluate generalization to unseen environments. (c) We conclude by discussing some limitations of a direct prediction-error based curiosity formulation. We observe that if the agent itself is the source of stochasticity in the environment, it can reward itself without making any actual progress. We empirically demonstrate this limitation in a 3D navigation task where the agent controls different parts of the environment.

## 4.2 Dynamics-based Curiosity-driven Learning

Consider an agent that sees an observation $x_t$, takes an action $a_t$ and transitions to the next state with observation $x_{t+1}$. We want to incentivize this agent with a reward $r_t$ relating to how informative the transition was. To provide this reward, we use an exploration bonus involving the following elements: (a) a network to embed observations into representations $\phi(x)$, (b) a forward dynamics network to predict the representation of the next state conditioned on the previous observation and action

$p(\phi(x_{t+1})|x_t, a_t)$. Given a transition tuple $\{x_t, x_{t+1}, a_t\}$, the exploration reward is then defined as $r_t = -\log p(\phi(x_{t+1})|x_t, a_t)$, also called the *surprisal* (Achiam and Sastry, 2017).

An agent trained to maximize this reward will favor transitions with high prediction error, which will be higher in areas where the agent has spent less time, or in areas with complex dynamics. Such a dynamics-based curiosity has been shown to perform quite well across scenarios (Pathak et al., 2017) especially when the dynamics are learned in an embedding space rather than raw observations. In this paper, we explore dynamics-based curiosity and use mean-squared error corresponding to a fixed-variance Gaussian density as surprisal, i.e., $\|f(x_t, a_t) - \phi(x_{t+1})\|_2^2$ where f is the learned dynamics model. However, any other density model could be used.

### 4.2.1 Feature spaces for forward dynamics

Consider the representation $\phi$ in the curiosity formulation above. If $\phi(x) = x$, the forward dynamics model makes predictions in the observation space. A good choice of feature space can make the prediction task more tractable and filter out irrelevant aspects of the observation space. But, what makes a good feature space for dynamics driven curiosity? We narrow down a few qualities that a good feature space should have:

- *Compact*: The features should be easy to model by being low(er)-dimensional and filtering out irrelevant parts of the observation space.
- *Sufficient*: The features should contain all the important information. Otherwise, the agent may fail to be rewarded for exploring some relevant aspect of the environment.
- *Stable*: Non-stationary rewards make it difficult for reinforcement agents to learn. Exploration bonuses by necessity introduce non-stationarity since what is new and novel becomes old and boring with time. In a dynamics-based curiosity formulation, there are two sources of non-stationarity: the forward dynamics model is evolving over time as it is trained and the features are changing as they learn. The former is intrinsic to the method, and the latter should be minimized where possible

In this work, we systematically investigate the efficacy of a number of feature learning methods, summarized briefly as follows:

**Pixels** The simplest case is where $\phi(x) = x$ and we fit our forward dynamics model in the observation space. Pixels are sufficient, since no information has been thrown away, and stable since there is no feature learning component. However, learning from pixels is tricky because the observation space may be high-dimensional and complex.

**Random Features (RF)** The next simplest case is where we take our embedding network, a convolutional network, and fix it after random initialization. Because the network is fixed, the features are stable. The features can be made compact in dimensionality, but they are not constrained to be. However, random features may fail to be sufficient. As discussed in 4.4 randomly initialized convnets have been shown to give useful features for image classification task since the induction bias of the architecture is appropriate for images (in particular the meaning a patch in one location is often highly related to the meaning of that same patch in a different spatial location, and the convnet encodes these equivariances to some extent). Because of this the dynamics model will be able to generalize when making predictions about an object it has previously seen in one location to another, without having to relearn this object in every possible location. For this reason the prediction error will not lead to the agent being endlessly fascinated with every patch of pixels appearing in every location.

**Variational Autoencoders (VAE)** VAEs were introduced in (Kingma and Welling, 2013b; Rezende et al., 2014a) to fit latent variable generative models $p(x,z)$ for observed data $x$ and latent variable $z$ with prior $p(z)$ using variational inference. The method calls for an inference network $q(z|x)$ that approximates the posterior $p(z|x)$. This is a feedforward network that takes an observation as input and outputs a mean and variance vector describing a Gaussian distribution with diagonal covariance. We can then use the mapping to the mean as our embedding network $\phi$. These features will be a low-dimensional approximately sufficient summary of the observation since they are trained to reconstruct the input (although it is possibly that the encoder may elect to remove some important aspects in order to balance the objective in principle), but they may still contain some irrelevant details such as noise, and the features will change over time as the VAE trains.

| | VAE | IDF | RF | Pixels |
|---|---|---|---|---|
| Stable | No | No | Yes | Yes |
| Compact | Yes | Yes | Maybe | No |
| Sufficient | Yes | Maybe | Maybe | Yes |

Table 4.1: Table summarizing the categorization of

different kinds of feature spaces considered.

**Inverse Dynamics Features (IDF)** Given a transition $(s_t, s_{t+1}, a_t)$ the inverse dynamics task is to predict the action $a_t$ given the previous and next states $s_t$ and $s_{t+1}$. Features are learned using a common neural network $\phi$ to first embed $s_t$ and $s_{t+1}$. The intuition is that the features learned should correspond to aspects of the environment that are under the agent's immediate control. This feature learning method is easy to implement and in principle should be invariant to certain kinds of noise (see (Pathak et al., 2017) for a discussion). A potential downside could be that the features learned may not be sufficient, that is they do not represent important aspects of the environment that the agent cannot immediately affect.

A summary of these characteristics is provided in Table 4.1. Note that the learned features are not stable because their distribution changes as learning progresses. One way to achieve stability could be to pre-train VAE or IDF networks. However, unless one has access to the internal state of the game, it is not possible to get a representative data of the game scenes to train the features. One way is to act randomly to collect data, but then it will be biased to where the agent started, and won't generalize further. Since all the features involve some trade-off of desirable properties, it becomes an empirical question as to how effective each of them is across environments.

### 4.2.2 Practical considerations in training an agent driven purely by curiosity

Deciding upon a feature space is only first part of the puzzle in implementing a practical system. Here, we detail the critical choices we made in the learning algorithm. Our goal was to reduce non-stationarity in order to make learning more stable and consistent across environments. Through the following considerations outlined below, we are able to get exploration to work reliably for different feature learning methods and environments with minimal changes to the hyper-parameters.

- *PPO*. In general, we have found the PPO algorithm (Schulman et al., 2017) to be a robust learning algorithm that requires little hyper-parameter tuning and so we stick to it for our experiments.

- *Reward normalization*. Since the reward function is non-stationary, it is useful to normalize the scale of the rewards so that the value function can learn quickly. We did this by dividing the rewards by a running estimate of the standard deviation of the sum of discounted rewards.

- *Advantage normalization.* While training with PPO, we normalize the advantages (Sutton and Barto, 1998) in a batch to have a mean of 0 and a standard deviation of 1.
- *Observation normalization.* We run a random agent on our target environment for 10000 steps, then calculate the mean and standard deviation of the observation and use these to normalize the observations when training. This is useful to ensure that the features do not have very small variance at initialization, and also ensure features have less variation across different environments.
- *More actors.* The stability of the method is greatly increased by increasing the number of parallel actors (which affects the batch-size) used. We typically use 128 parallel runs of the same environment for data collection while training an agent.
- *Normalizing the features.* In combining intrinsic and extrinsic rewards, we found it useful to ensure that the scale of the intrinsic reward was consistent across state space. We achieved this by using batch-normalization (Ioffe and Szegedy, 2015b) in the feature embedding network.

### 4.2.3 'Death is not the end': discounted curiosity with infinite horizon

One important point is that the use of an end-of-episode signal, sometimes called a 'done', can often leak information about the true reward function. If we don't remove the 'done' signal, many of the Atari games become too simple. For example, a simple strategy of giving +1 artificial reward at every time-step when the agent is alive and 0 on death is sufficient to obtain a high score in some games, for instance, the Atari game 'Breakout' where it will seek to maximize the episode length and hence its score. In the case of negative rewards, the agent will try to end the episode as quickly as possible.

In light of this, if we want to study the behavior of pure exploration agents, we should not bias the agent. In the infinite horizon setting (i.e., the discounted returns are not truncated at the end of the episode and always bootstrapped using the value function), death is just another transition to the agent, to be avoided only if it is boring. Therefore, we removed 'done' to separate the gains of an agent's exploration from merely that of the death signal. In practice, we do find that the agent avoids dying in the games since that brings it back to the beginning of the game, an area it has already

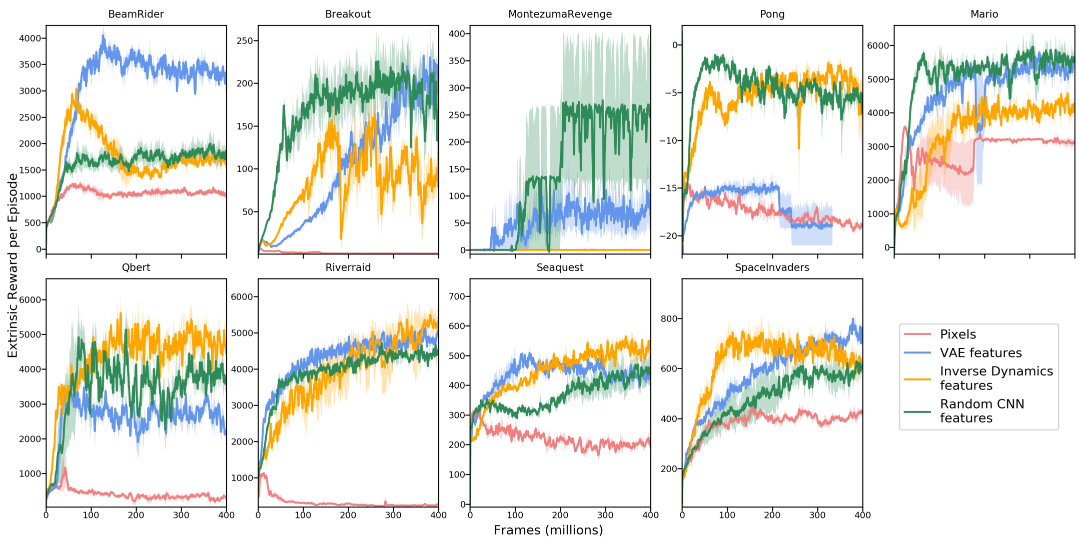


Figure 4.2: A comparison of feature learning methods on 8 selected Atari games and the Super Mario Bros. These evaluation curves show the mean reward (with standard error) of agents trained purely by curiosity, without reward or an end-of-episode signal. We see that our purely curiosity-driven agent is able to gather rewards in these environments without using any extrinsic reward at training. Results on all of the Atari games are in the appendix in Figure B.2. We find curiosity model trained on pixels does not work well across any environment and VAE features perform either same or worse than random and inverse dynamics features. Further, inverse dynamics-trained features perform better than random features in $55\%$ of the Atari games. An interesting outcome of this analysis is that random features for modeling curiosity are a simple, yet surprisingly strong baseline and likely to work well in half of the Atari games.

seen many times and where it can predict the dynamics well. This subtlety has been neglected by previous works showing experiments without extrinsic rewards.

## 4.3 Experiments

In all of our experiments, both the policy and the embedding network work directly from pixels. For our implementation details including hyper-parameters and architectures, please refer to the Appendix B.1. Unless stated otherwise, all curves are the average of three runs with different seeds, and the shaded areas are standard errors of the mean. We have released the code and videos of a purely curious agent playing across all environments on the website [1].

[1] https://pathak22.github.io/large-scale-curiosity/

### 4.3.1 Curiosity-driven learning without extrinsic rewards

We begin by scaling up a pure curiosity-driven learning to a large number of environments without using any extrinsic rewards. We pick a total of 54 diverse simulated environments, as shown in Figure 4.1, including 48 Atari games, Super Mario Bros., 2 Roboschool scenarios (learning Ant controller and Juggling), Two-player Pong, 2 Unity mazes (with and without a TV controlled by the agent). The goal of this large-scale analysis is to investigate the following questions: (a) What actually happens when you run a pure curiosity-driven agent on a variety of games without any extrinsic rewards? (b) What kinds of behaviors can you expect from these agents? (c) What is the effect of the different feature learning variants in dynamics-based curiosity on these behaviors?

**Atari Games** To answer these questions, we began with a collection of well-known Atari games and ran a suite of experiments with different feature learning methods. One way to measure how well a purely curious agent performs is to measure the extrinsic reward it is able to achieve, i.e. how good is the agent at playing the game. We show the evaluation curves of mean extrinsic reward in on 8 common Atari games in Figure 4.2 and all 48 Atari suite in Figure B.2 in the appendix. It is important to note that the extrinsic reward is only used for evaluation, not for training. However, this is just a proxy for pure exploration because the game rewards could be arbitrary and might not align at all with how the agent explores out of curiosity.

The first thing to notice from the curves is: *most of them are going up*. This shows that a pure curiosity-driven agent can learn to obtain external rewards even without using any extrinsic rewards during training. It is remarkable that agents with no extrinsic reward and no end of episode signal can learn to get scores comparable in some cases to learning with the extrinsic reward. For instance, in Breakout, the game score increases on hitting the ball with the paddle into bricks which disappear and give points when struck. The more times the bricks are struck in a row by the ball, the more complicated the pattern of bricks remaining becomes, making the agent more curious to explore further, hence, collecting points as a bi-product. Further, when the agent runs out of lives, the bricks are reset to a uniform structure again that has been seen by the agent many times before and is hence very predictable, so the agent tries to stay alive to be curious by avoiding reset by death.

This is an unexpected result and might suggest that many popular RL test-beds do not need an external reward. Perhaps game designers (similar to architects, urban planners, gardeners, etc.) are good at setting up curricula to guide agents through the task. This could explain the reason curiosity-like objective decently aligns with the extrinsic reward in many human-designed environments (Lazzaro, 2004; Costikyan, 2013; Hunicke et al., 2004; Wouters et al., 2011). However, this is not always the case, and sometimes a curious agent can even do worse than random agent! This happens when the extrinsic reward has little correlation with the agent's exploration, or when the agent fails to explore efficiently (e.g. see games 'Atlantis', 'IceHockey' in Figure B.2). We further encourage the reader to refer to the game-play videos of the agent available on the website for a better understanding of the learned skills.

**Comparison of feature learning methods:** We compare four feature learning methods in Figure 4.2: raw pixels, random features, inverse dynamics features and VAE features. Training dynamics on raw-pixels performs bad across all the environments, while encoding pixels into features does better. This is likely because it is hard to learn a good dynamics model in pixel space, and prediction errors may be dominated by small irrelevant details.

Surprisingly, random features (RF) perform quite well across tasks and sometimes better than using learned features. One reason for good performance is that the random features are kept frozen (stable), the dynamics model learned on top of them has an easier time because of the stationarity of the target. In general, random features should work well in the domains where visual observations are simple enough, and random features can preserve enough information about the raw signal, for instance, Atari games. Interestingly, we find that while random features work well at training, IDF learned features appear to generalize better in Mario Bros. (see Section 4.3.2 for details).

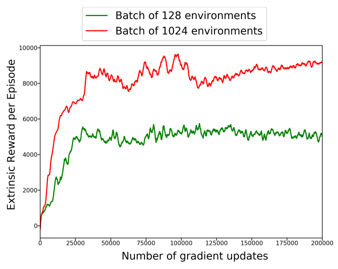
The VAE method also performed well but was somewhat unstable, so we decided to use RF and IDF for further experiments. The detailed result in appendix Figure B.2 compares IDF vs. RF across the full Atari suite. To quantify the learned behaviors, we compared our curious agents to a randomly acting agent. We found that an IDF-curious agent collects more game reward than a random agent in 75% of the Atari games, an RF-curious agent does better in 70%. Further, IDF does better than RF in 55% of the games. Overall, random features and inverse dynamics features worked well in general. Further details in the appendix.

**Super Mario Bros.** We compare different feature learning methods in Mario Bros. in Figure 4.2. Super Mario Bros has already been studied in the context of extrinsic reward free learning (Pathak et al., 2017) in small-scale experiments, and so we were keen to see how far curiosity alone can push the agent. We use an efficient version of Mario simulator faster to scale up for longer training keeping observation space, actions, dynamics of the game intact. Due to 100x longer training and using PPO for optimization, our agent is able to pass several levels of the game, significantly improving over prior exploration results on Mario Bros.

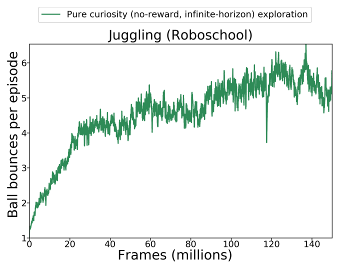
Could we further push the performance of a purely curious agent by making the underlying optimization more stable? One way is to scale up the batch-size. We do so by increasing the number of parallel threads for running environments from 128 to 1024. We show the comparison between training using 128 and 1024 parallel environment threads in Figure 4.3(a). As apparent from the graph, training with large batch-size using 1024 parallel environment threads performs much better. In fact, the agent is able to explore much more of the game: *discovering 11 different levels of the game*, finding secret rooms and defeating bosses. Note that the x-axis in the figure is the number of gradient steps, not the number of frames, since the point of this large-scale experiment is not a claim about sample-efficiency, but performance with respect to training the agent. This result suggests that the performance of a purely curiosity-driven agent would improve as the training of base RL algorithm (which is PPO in our case) gets better. The video is on the website.

**Roboschool Juggling** We modified the Pong environment from the Roboschool framework to only have one paddle and to have two balls. The action space is continuous with two-dimensions, and we discretized the action space into 5 bins per dimension giving a total of 25 actions. Both the policy and embedding network are trained on pixel observation space (note: not state space). This environment is more difficult to control than the toy physics used in games, but the agent learns to intercept and strike the balls when it comes into its area. We monitored the number of bounces of the balls as a proxy for interaction with the environment, as shown in Figure 4.3(b). See the video on the project website.

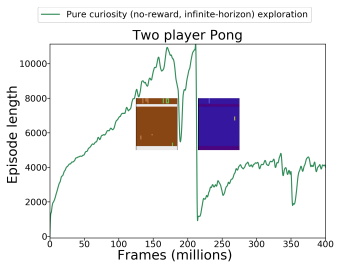
**Roboschool Ant Robot** We also explored using the Ant environment which consists of an Ant with 8 controllable joints on a track. We again discretized the action space and trained policy and embedding network on raw pixels (not state space). However, in this case, it was less easy to measure exploration because the extrinsic distance reward measures progress along the racetrack, but a purely curious agent is free to move in any direction. We find that a walking like behavior emerges purely out of a curiosity-driven training. We refer the reader to the result video showing that the agent is meaningfully



(a) Mario w/ large batch



(b) Juggling (Roboschool)



(c) Two-player Pong

Figure 4.3: (a) Left: A comparison of the RF method on Mario with different batch sizes. Results are without using extrinsic reward. (b) Center: Number of ball bounces in the Juggling (Roboschool) environment. (c) Right: Mean episode length in the multiplayer Pong environment. The discontinuous jump on the graph corresponds to the agent reaching a limit of the environment - after a certain number of steps in the environment the Atari Pong emulator starts randomly cycling through background colors and becomes unresponsive to agent's actions

interacting with the environment.

**Multi-agent curiosity in Two-player Pong** We have already seen that a purely curiosity-driven agent learns to play several Atari games without reward, but we wonder how much of that behavior is caused by the fact that the opposing player is a computer-agent with hardcoded strategy. What would happen if we were to make both the teams playing against each other to be curious? To find out, we take Two-player Pong game where both the sides (paddles of pong) of the game are controlled by curiosity-driven agents. We share the initial layers of both the agent and have different action heads, i.e., total action space is now the cross product of the actions of player 1 by the actions of player 2.

Note that the extrinsic reward is meaningless in this context since the agent is playing both sides, so instead, we show the length of the episode. The results are shown in Figure 4.3(c). We see from the episode length that the agent learns to have more and longer rallies over time, learning to play pong without any teacher – purely by curiosity on both sides. In fact, *the game rallies eventually get so long that they break our Atari emulator* causing the colors to change radically, which crashes the policy as shown in the plot.

### 4.3.2 Generalization across novel levels in Super Mario Bros.

In the previous section, we showed that our purely curious agent can learn to explore efficiently and learn useful skills, e.g., game playing behaviour in games, walking behaviour in Ant etc. So far, these skills were shown in the environment where the agent was trained on. However, one advantage of developing reward-free learning is that one should then be able to utilize abundant "unlabeled" environments without reward functions by showing generalization to novel environments. To test this, we first pre-train our agent using curiosity only in the Level 1-1 of Mario Bros. We investigate how well RF and IDF-based curiosity agents generalize to novel levels of Mario. In Figure 4.4, we show two examples of training on one level of Mario and finetuning on another testing level, and compare to learning on the testing level from scratch. The training signal in all the cases is only curiosity reward. In the first case, from Level 1-1 to Level 1-2, the global statistics of the environments match (both are 'day' environment in games, i.e., blue background) but levels have different enemies, geometry and difficulty level. We see that there is strong transfer from for both methods in this scenario. However, the transfer performance is weaker in the second scenario from Level 1-1 to Level 1-3. This is so because the problem is considerably harder for the latter level pairing as there is a color scheme shift from day to night, as shown in Figure 4.4.

We further note that IDF-learned features transfer in both the cases and random features transfer in the first case, but do not transfer in the second scenario from day to night. These results might suggest that while random features perform well on training environments, learned features appear to generalize better to novel levels. However, this needs more analysis in the future across a large variety of environments. Overall, we find some promising evidence showing that skills learned by curiosity help our agent explore efficiently in novel environments.

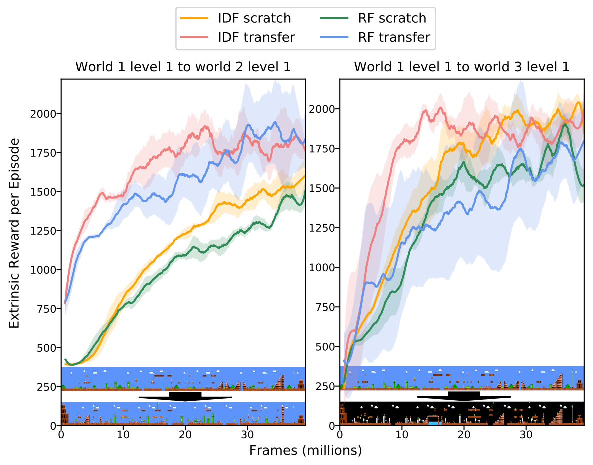


Figure 4.4: Mario generalization experiments. On the left we show transfer results from Level 1-1 to Level 1-2, and on the right we show transfer results from Level 1-1 to Level 1-3. Underneath each plot is a map of the source and target environments. All agents are trained without extrinsic reward.

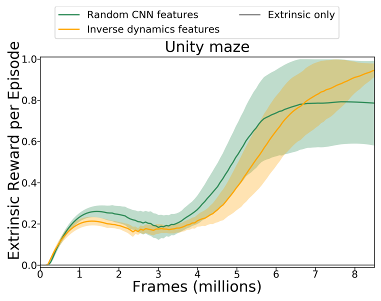


Figure 4.5: Mean extrinsic reward in the Unity environment while training with terminal extrinsic + curiosity reward. Note that the curve for extrinsic reward only training is constantly zero.

## 4.3.3 Curiosity with Sparse External Reward

In all our experiments so far, we have shown that our agents can learn useful skills without any extrinsic rewards driven purely by curiosity. However, in many scenarios, we might want the agent to perform some particular task of interest. This is usually conveyed to the agent by defining extrinsic rewards. When rewards are dense (e.g. game score at every frame), classic RL works well and intrinsic rewards generally should not help performance. However, designing dense rewards is a challenging engineering problem (see introduction for details). In this section, we evaluate how well curiosity can help an agent perform a task in presence of sparse, or just terminal, rewards.

**Terminal reward setting**: For many real problems, e.g. navigation, the only terminal reward is available, a setting where classic RL typically performs poorly. Hence, we consider the 3D navigation in a maze designed in the Unity ML-agent framework with 9 rooms and a sparse terminal reward. There is a discrete action space consisting of: move forwards, look left 15 degrees, look right 15 degrees and no-op. The agent starts in the room-1, which is furthest away from room-9 which contains the goal of the agent. We compare an agent trained with extrinsic reward (+1 when the goal is reached, 0 otherwise) to an agent trained with extrinsic + intrinsic reward. Extrinsic only (classic RL) never finds the goal in all our trials which means it is impossible to get any meaningful gradients. Whereas extrinsic+intrinsic typically converges to getting the reward every time. Results in Figure 4.5 show results for vanilla PPO, PPO + IDF-curiosity and PPO + RF-curiosity.

**Sparse reward setting**: In preliminary experiments, we picked 5 Atari games which have sparse rewards (as categorized by (Bellemare et al., 2016)), and compared extrinsic (classic RL) vs. extrinsic+intrinsic (ours) reward performance. In 4 games out of 5, curiosity bonus improves performance (see Table B.1 in the appendix, the higher score is better). We would like to emphasize that this is not the focus of the paper, and these experiments are provided just for completeness. We just combined extrinsic (coefficient 1.0) and intrinsic reward (coefficient 0.01) directly without any tuning. We leave the question on how to optimally combine extrinsic and intrinsic rewards as a future direction.

## 4.4 Related Work

**Intrinsic Motivation:** A family of approaches to intrinsic motivation reward an agent based on prediction error (Schmidhuber, 1991c; Stadie et al., 2015; Pathak et al., 2017; Achiam and Sastry, 2017), prediction uncertainty (Still and Precup, 2012; Houthooft et al., 2016), or improvement (Schmidhuber, 1991a; Lopes et al., 2012) of a forward dynamics model of the environment that gets trained along with the agent's policy. As a result the agent is driven to reach regions of the environment that are difficult to predict for the forward dynamics model, while the model improves its predictions in these regions. This adversarial and non-stationary dynamics can give rise to complex behaviors. Relatively little work has been done in this area on the pure exploration setting where there is no external reward. Of these mostly closely related are those that use a forward dynamics model of a feature space such as Stadie et al. (2015) where they use autoencoder features, and Pathak et al. (2017) where they use features trained with an inverse dynamics task. These correspond roughly to the VAE and IDF methods detailed in Section 4.2.1.

Smoothed versions of state visitation counts can be used for intrinsic rewards (Bellemare et al., 2016; Fu et al., 2017; Ostrovski et al., 2018a; Tang et al., 2017b). Count-based methods have already shown very strong results when combining with extrinsic rewards such as setting the state of the art in the Atari game Montezuma's Revenge (Bellemare et al., 2016), and also showing significant exploration of the game without using the extrinsic reward. It is not yet clear in which situations count-based approaches should be preferred over dynamics-based approaches; we chose to focus on dynamics-based bonuses in this paper since we found them straightforward to scale and parallelize. In our preliminary experiments, we did not have sufficient success with already existing count-based implementations in scaling up for a large-scale study.

Learning without extrinsic rewards or fitness functions has also been studied extensively in the evolutionary computing where it is referred to as 'novelty search' (Lehman and Stanley, 2008, 2011; Stanley and Lehman, 2015). There the novelty of an event is often defined as the distance of the event to the nearest neighbor amongst previous events, using some statistics of the event to compute distances. One interesting finding from this literature is that often much more interesting solutions can be found by not solely optimizing for fitness.

Other methods of exploration are designed to work in combination with maximizing a reward function, such as those utilizing uncertainty about value function estimates (Osband et al., 2016; Chen et al., 2017), or those using perturbations of the policy for exploration (Fortunato et al., 2018; Plappert et al., 2018). Schmidhuber (2010) and Oudeyer and Kaplan (2009); Oudeyer (2018) provide a great review of some of the earlier work on approaches to intrinsic motivation. Alternative methods of exploration include Sukhbaatar et al. (2018) where they utilize an adversarial game between two agents for exploration. In Gregor et al. (2017), they optimize a quantity called empowerment which is a measurement of the control an agent has over the state. In a concurrent work, diversity is used as a measure to learn skills without reward functions Eysenbach et al. (2018).

**Random Features:** One of the findings in this paper is the surprising effectiveness of random features, and there is a substantial literature on random projections and more generally randomly initialized neural networks. Much of the literature has focused on using random features for classification (Saxe et al., 2011; Jarrett et al., 2009; Yang et al., 2015) where the typical finding is that whilst random features can work well for simpler problems, feature learning performs much better once the problem becomes sufficiently complex. Whilst we expect this pattern to also hold true for dynamics-based exploration, we have some preliminary evidence showing that learned features appear to generalize better to novel levels in Mario Bros.

## 4.5 Discussion

We have shown that our agents trained purely with a curiosity reward are able to learn useful behaviours: (a) Agent being able to play many atari games without using any rewards. (b) Mario being able to cross over over 11 levels without reward. (c) Walking like behavior emerged in the Ant environment. (d) Juggling like behavior in Roboschool environment (e) Rally-making behavior in Two-player Pong with curiosity-driven agent on both sides. But this is not always true as there are some Atari games where exploring the environment does not correspond to extrinsic reward.

More generally, these results suggest that, in environments designed by humans, the extrinsic reward is perhaps often aligned with the objective of seeking novelty. The game designers set up curriculums to guide users while playing the game explaining the reason Curiosity-like objective decently aligns with the extrinsic reward in many human-designed games (Costikyan, 2013; Hunicke et al., 2004; Wouters et al., 2011; Lazzaro, 2004).

**Limitation of prediction error based curiosity:** A more serious potential limitation is the handling of stochastic dynamics. If the transitions in the environment are random, then even with a perfect dynamics model, the expected reward will be the entropy of the transition, and the agent will seek out transitions with the highest entropy. Even if the environment is not truly random, unpredictability caused by a poor learning algorithm, an impoverished model class or partial observability can lead to exactly the same problem. We did not observe this effect in our experiments on games so we designed an environment to illustrate the point.

We return to the maze of Section 4.3.3 to empirically validate a common thought experiment called the noisy-TV problem. The idea is that local sources of entropy in an environment like a TV that randomly changes channels when an action is taken should prove to be an irresistible attraction to our agent. We take this thought experiment

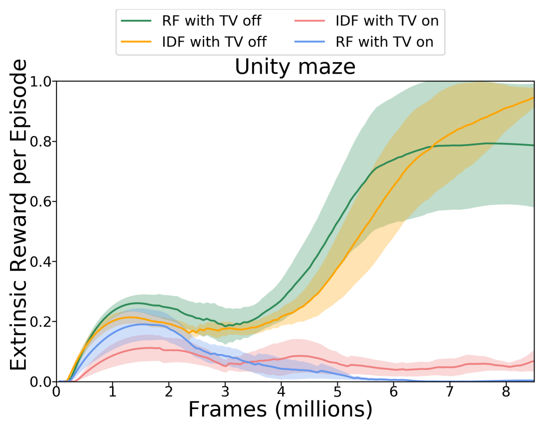


Figure 4.6: We add a noisy TV to the unity environment in Section 4.3.3. We compare IDF and RF with and without the TV.

literally and add a TV to the maze along with an action to change the channel. In Figure 4.6 we show how adding the noisy-TV affects the performance of IDF and RF. As expected the presence of the TV drastically slows down learning, but we note that if you run the experiment for long enough the agents do sometimes converge to getting the extrinsic reward consistently. We have shown empirically that stochasticity can be a problem, and so it is important for future work to address this issue in an efficient manner.

**Future Work:** We have presented a simple and scalable approach that can learn nontrivial behaviors across a diverse range of environments without any reward function or end-of-episode signal. One surprising finding of this paper is that random features perform quite well, but learned features appear to generalize better. Whilst we believe that learning features will become important once the environment is complex enough, we leave that up to future work to explore.

Our wider goal, however, is to show that we can take advantage of many unlabeled (i.e., not having an engineered reward function) environments to improve performance on a task of interest. Given this goal, showing performance in environments with a generic reward function is just the first step, and future work could investigate transfer from unlabeled to labeled environments.

## Acknowledgments

We would like to thank Chris Lu for helping with the Unity environment, Oleg Klimov for helping with the Roboschool environments and Phillip Isola and Alex Nichol for feedback on the paper. We are grateful to the members of BAIR and OpenAI for fruitful discussions. DP is supported by the Facebook graduate fellowship.

## 4.6 2021 Retrospective

Upon reflection I think that the most important contribution of this paper was to focus solely on ‘unsupervised reinforcement learning’ (meaning not making use of any reward functions associated with the environment). Prior to this paper this question had not been investigated extensively, especially across a large variety of environments. Investigating a variety of non-toy environments showed that all sorts of interesting behaviours could come from an unsupervised approach.

Most of the environments considered in this chapter are simple in the sense of having simple visuals, have little or no stochasticity and often being fully-observable (or almost fully-observable). Having richer visuals can presumably be handled through the use of better features, perhaps coming from a larger encoder pretrained on something like Imagenet or video data. Partial-observability could be addressed by using dynamics model that is recurrent. The lack of stochasticity is harder to address and indeed we would expect the baseline methods explored here to fail in such environments. Nevertheless I think we learn a lot about the remarkable behaviors that can be elicited through simple objectives, and so we learn a lot about the kinds of things we can expect to see once that problem is solved. Indeed I suspect that many potential reward functions will cause the agent to learn useful behaviors if they are not completely trivial, since basic things like learning to avoid obstacles, moving between areas, increasing control other the environment etc, are almost universally useful.

Uncertainty in predictions is often described as being epistemic or aleatoric. Epistemic uncertainty comes from uncertainty over the parameters of the model you are fitting translated into predictive uncertainty. Aleatoric uncertainty comes from true stochasticity in the variable you are predicting. So for example if one is fitting a Bernoulli distribution to a sequence of coin-flips, then the epistemic uncertainty comes from not having seen enough coin flips leading to uncertainty in your estimation of the probability of heads. Aleatoric uncertainty comes from the fact that no matter how certain you are about the probability of heads, unless the probability is zero or one you will not be able to guess the outcome correctly every time. Rather than think in terms of epistemic and aleatoric I find it more useful to think in terms of irreducible and reducible error. The reducible error goes to zero as you observe an infinite amount of data, whereas the irreducible error converges to some positive number (unless you are in the rare case with no irreducible error, normally a toy problem of some sort). I prefer this because aleatoric suggests some fact about the environment itself, whereas it could just as well be a fact about the model class you have chosen (for example a non-recurrent versus recurrent dynamics model).

For simple environments where that are deterministic and fully-observable, the irreducible error ought to be close to zero, and so any prediction error observed can be assumed to be ‘interesting’ in the sense of resulting from imperfect estimation of the parameters. For environments like these one can expect simple prediction error to work well as an intrinsic reward. For environments with localized sources of irreducible error, we expect that agents trained with this reward to be drawn to them like a moth to flame.

The disadvantage of a purely unsupervised approach is that it is hard to quantify progress. Subsequent works have adopted an approach analogous to that taken in unsupervised computer vision papers, where the benefits of unsupervised objectives on unlabelled data are demonstrated through their utility as a pre-training step prior to fine-tuning on a potentially small number of labelled examples. In the reinforcement learning setting the analogy would be to first allow a large amount of unsupervised interaction environment, then measure the improvement this gives to sample complexity when used as an initialisation for learning with the extrinsic rewards. A recent example is ‘Fast Task Inference with Variational Intrinsic Successor Features’ (Hansen et al., 2019).

Despite recent progress on unsupervised reinforcement learning I think using it only for feature learning can only improve sample efficiency so far. I think the real breakthrough in unsupervised pre-training will come when we can discover useful behaviours in an unsupervised way and transfer them to a new task. One line of work in this direction is to consider build latent variable models where the variable encodes a policy/trajectory/skill, and where high mutual information between the latent variable and trajectory is enforced. Through appropriate choices of amortized inference networks some degree over the semantics of the latent variable can be enforced. For example if the inference network must infer the latent variable independently from each state of the trajectory (see for example (Eysenbach et al., 2018)), then each latent variable will correspond to a trajectory that preferentially occupies a subset of states. If the inference network must infer the latent variable only from the final state of the trajectory then each latent variable will correspond to trajectories that end in certain states (see for example (Gregor et al., 2017)). We can also go further and ask to maximize the mutual information between the latent variable and the start state *given* the ending state as in (Baumli et al., 2021), which will yield latent variables yielding a predictable change of start from beginning to end.

Another contribution that I think turned out to be useful was making a vivid demonstration of the noisy-TV problem for prediction-based exploration bonuses. Although the problem had been discussed before, I think that the attitude was that although it was theoretically a problem in practice it might not be worth worrying about. On this point a contribution of the paper, and associated blog post [2], was firstly to clearly explain the noisy-TV problem, and show a video of an agent really getting distracted by a noisy-TV in a 3d environment. Secondly we show that even in environments with small amounts of randomness, say Montezuma's Revenge with 'sticky-actions' (this is where with a small probability the previous action taken is repeated), an agent trained only with a prediction error reward can learn to find states that amplify this stochasticity as much as possible (by moving to the boundary of two rooms and hopping back and forth between them, the transition between rooms is hard to predict because it will depend on whether the sticky action condition is triggered). This maximises the change in pixels as a result of the stochasticity, which, for simple dynamics models which cannot handle multi-modal distributions, will reliably give high errors. Thirdly it was suggested in (Pathak et al., 2017) that by predicting the dynamics in an appropriate choice of feature space, the noisy-TV problem can be avoided by using features where stochastic parts of the environment are removed. In our twist on the noisy-TV the agent has a 'remote' that gives a 'change the channel' action, but the channel will be chosen randomly. The point is that the randomness is partially under the agent's control, and so the image on the TV is predictive of the agent's action, which makes it a counter-example for the particular feature space suggested in (Pathak et al., 2017) (features learned by predicting inverse dynamics). A potential remedy for the noisy-TV problem is discussed in the next chapter.

[2] https://openai.com/blog/reinforcement-learning-with-prediction-based-rewards/

# Chapter 5

# Exploration by Random Network Distillation

## 5.1 Introduction

Reinforcement learning (RL) methods work by maximizing the expected return of a policy. This works well when the environment has dense rewards that are easy to find by taking random sequences of actions, but tends to fail when the rewards are sparse and hard to find. In reality it is often impractical to engineer dense reward functions for every task one wants an RL agent to solve. In these situations methods that explore the environment in a directed way are necessary.

Recent developments in RL seem to suggest that solving the most challenging tasks (Silver et al., 2016; Zoph and Le, 2016; Horgan et al., 2018; Espeholt et al., 2018; OpenAI, 2018; Andrychowicz et al., 2020) requires processing large numbers of samples obtained from running many copies of the environment in parallel. In light of this it is desirable to have exploration methods that scale well with large amounts of experience. However many of the recently introduced exploration methods based on counts, pseudo-counts, information gain or prediction gain are difficult to scale up to large numbers of parallel environments.

This paper introduces an exploration bonus that is particularly simple to implement, works well with high-dimensional observations, can be used with any policy optimization algorithm, and is efficient to compute as it requires only a single forward pass of a neural network on a batch of experience. Our exploration bonus is based on the observation that neural networks tend to have significantly lower prediction errors on examples similar to those on which they have been trained. This motivates the use

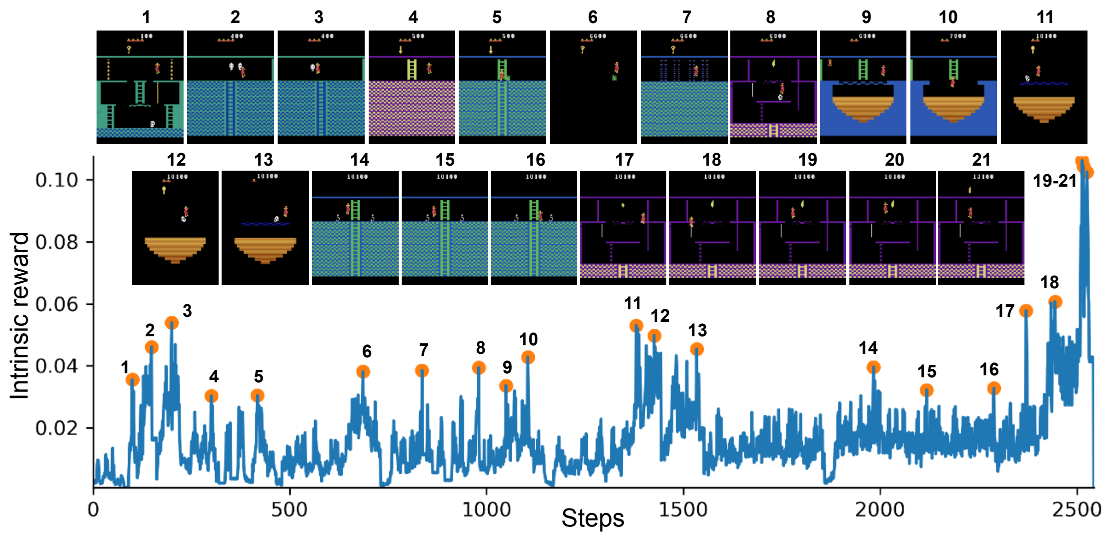


Figure 5.1: RND exploration bonus over the course of the first episode where the agent picks up the torch (19-21). To do so the agent passes 17 rooms and collects gems, keys, a sword, an amulet, and opens two doors. Many of the spikes in the exploration bonus correspond to meaningful events: losing a life (2,8,10,21), narrowly escaping an enemy (3,5,6,11,12,13,14,15), passing a difficult obstacle (7,9,18), or picking up an object (20,21). The large spike at the end corresponds to a novel experience of interacting with the torch, while the smaller spikes correspond to relatively rare events that the agent has nevertheless experienced multiple times. See here for videos.

of prediction errors of networks trained on the agent's past experience to quantify the novelty of new experience.

As pointed out by many authors, agents that maximize such prediction errors tend to get attracted to transitions where the answer to the prediction problem is a stochastic function of the inputs. For example if the prediction problem is that of predicting the next observation given the current observation and agent's action (forward dynamics), an agent trying to maximize this prediction error will tend to seek out stochastic transitions, like those involving randomly changing static noise on a TV, or outcomes of random events such as coin tosses. This observation motivated the use of methods that quantify the relative improvement of the prediction, rather than its absolute error. Unfortunately, as previously mentioned, such methods are hard to implement efficiently.

We propose an alternative solution to this undesirable stochasticity by defining an exploration bonus using a prediction problem where the answer is a deterministic function of its inputs. Namely we predict the output of a fixed randomly initialized neural network on the current observation.

Atari games have been a standard benchmark for deep reinforcement learning algorithms since the pioneering work by Mnih et al. (2013). Bellemare et al. (2016) identified among these games the hard exploration games with sparse rewards: Freeway, Gravitar, Montezuma's Revenge, Pitfall!, Private Eye, Solaris, and Venture. RL algorithms tend to struggle on these games, often not finding even a single positive reward.

In particular, Montezuma's Revenge is considered to be a difficult problem for RL agents, requiring a combination of mastery of multiple in-game skills to avoid deadly obstacles, and finding rewards that are hundreds of steps apart from each other even under optimal play. Significant progress has been achieved by methods with access to either expert demonstrations (Pohlen et al., 2018; Aytar et al., 2018; Garmulewicz et al., 2018), special access to the underlying emulator state (Tang et al., 2017a; Stanton and Clune, 2018), or both (Salimans and Chen, 2018). However without such aids, progress on the exploration problem in Montezuma's Revenge has been slow, with the best methods finding about half the rooms (Bellemare et al., 2016). For these reasons we provide extensive ablations of our method on this environment.

We find that even when disregarding the extrinsic reward altogether, an agent maximizing the RND exploration bonus consistently finds more than half of the rooms in Montezuma's Revenge. To combine the exploration bonus with the extrinsic rewards we introduce a modification of Proximal Policy Optimization (PPO, Schulman et al.

(2017)) that uses two value heads for the two reward streams. This allows the use of different discount rates for the different rewards, and combining episodic and non-episodic returns. With this additional flexibility, our best agent often finds 22 out of the 24 rooms on the first level in Montezuma's Revenge, and occasionally (though not frequently) passes the first level. The same method gets state of the art performance on Venture and Gravitar.

## 5.2 Method

### 5.2.1 Exploration bonuses

Exploration bonuses are a class of methods that encourage an agent to explore even when the environment's reward $e_t$ is sparse. They do so by replacing $e_t$ with a new reward $r_t = e_t + i_t$, where $i_t$ is the exploration bonus associated with the transition at time $t$.

To encourage the agent to visit novel states, it is desirable for $i_t$ to be higher in novel states than in frequently visited ones. Count-based exploration methods provide an example of such bonuses. In a tabular setting with a finite number of states one can define $i_t$ to be a decreasing function of the visitation count $n_t(\boldsymbol{s})$ of the state $\boldsymbol{s}$. In particular $i_t = 1/n_t(\boldsymbol{s})$ and $i_t = 1/\sqrt{n_t(\boldsymbol{s})}$ have been used in prior work (Bellemare et al., 2016; Ostrovski et al., 2018b). In non-tabular cases it is not straightforward to produce counts, as most states will be visited at most once. One possible generalization of counts to non-tabular settings is pseudo-counts (Bellemare et al., 2016) which uses changes in state density estimates as an exploration bonus. In this way the counts derived from the density model can be positive even for states that have not been visited in the past, provided they are similar to previously visited states.

An alternative is to define $i_t$ as the prediction error for a problem related to the agent's transitions. Generic examples of such problems include forward dynamics (Schmidhuber, 1991c; Stadie et al., 2015; Achiam and Sastry, 2017; Pathak et al., 2017; Burda et al., 2018b) and inverse dynamics (Haber et al., 2018). Non-generic prediction problems can also be used if specialized information about the environment is available, like predicting physical properties of objects the agent interacts with (Denil et al., 2016). Such prediction errors tend to decrease as the agent collects more experience similar to the current one. For this reason even trivial prediction problems like predicting a constant zero function can work as exploration bonuses (Fox et al., 2018).

### 5.2.2 Random Network Distillation

This paper introduces a different approach where the prediction problem is randomly generated. This involves two neural networks: a fixed and randomly initialized *target* network which sets the prediction problem, and a *predictor* network trained on data collected by the agent. The target network takes an observation to an embedding $f : O \to \mathbb{R}^k$ and the predictor neural network $\hat{f} : O \to \mathbb{R}^k$ is trained by gradient descent to minimize the expected MSE $\|\hat{f}(\mathrm{x};\theta) - f(\mathrm{x})\|^2$ with respect to its parameters $\theta_{\hat{f}}$. This process distills a randomly initialized neural network into a trained one. The prediction error is expected to be higher for novel states dissimilar to the ones the predictor has been trained on.

To build intuition we consider a toy model of this process on MNIST. We train a predictor neural network to mimic a randomly initialized target network on training data consisting of a mixture of images with the label 0 and of a target class, varying the proportion of the classes, but not the total number of training examples. We then test the predictor network on the unseen test examples of the target class and report the MSE. In this model the zeros are playing the role of states that have been seen many times before, and the target class is playing the role of states that have been visited infrequently. The results are shown in Figure 5.2. The figure shows that test error decreases as a function of the number of training examples in the target class, suggesting that this method can be used to detect novelty. Figure 5.1 shows that the intrinsic reward is high in novel states in an episode of Montezuma's Revenge.

One objection to this method is that a sufficiently powerful optimization algorithm might find a predictor that mimics the target random network perfectly on any input (for example the target network itself would be such a predictor). However the above experiment on MNIST shows that standard gradient-based methods don't overgeneralize in this undesirable way.

#### 5.2.2.1 Sources of prediction errors

In general, prediction errors can be attributed to a number of factors:

1. *Amount of training data*. Prediction error is high where few similar examples were seen by the predictor (epistemic uncertainty).

2. *Stochasticity*. Prediction error is high because the target function is stochastic (aleatoric uncertainty). Stochastic transitions are a source of such error for forward dynamics prediction.

3. *Model misspecification*. Prediction error is high because necessary information is missing, or the model class is too limited to fit the complexity of the target function.

4. *Learning dynamics*. Prediction error is high because the optimization process fails to find a predictor in the model class that best approximates the target function.

Factor 1 is what allows one to use prediction error as an exploration bonus. In practice the prediction error is caused by a combination of all of these factors, not all of them desirable.

For instance if the prediction problem is forward dynamics, then factor 2 results in the ‘noisy-TV’ problem. This is the thought experiment where an agent that is rewarded for errors in the prediction of its forward dynamics model gets attracted to local sources of entropy in the environment. A TV showing white noise would be such an attractor, as would a coin flip.

To avoid the undesirable factors 2 and 3, methods such as those by Schmidhuber (1991a); Oudeyer et al. (2007); Lopes et al. (2012); Achiam and Sastry (2017) instead use a measurement of how much the prediction model improves upon seeing a new datapoint. However these approaches tend to be computationally expensive and hence difficult to scale.

RND obviates factors 2 and 3 since the target network can be chosen to be deterministic and inside the model-class of the predictor network.

#### 5.2.2.2 Relation to uncertainty quantification

RND prediction error is related to an uncertainty quantification method introduced by Osband et al. (2018). Namely, consider a regression problem with data distribution $D = \{x_i, y_i\}_i$. In the Bayesian setting we would consider a prior $p(\theta^*)$ over the parameters of a mapping $f_{\theta^*}$ and calculate the posterior after updating on the evidence.

Let $\mathcal{F}$ be the distribution over functions $g_\theta = f_\theta + f_{\theta^*}$, where $\theta^*$ is drawn from $p(\theta^*)$ and $\theta$ is given by minimizing the expected prediction error

$$\theta = \arg\min_{\theta} \mathbb{E}_{(x_i,y_i)\sim D} \|f_\theta(x_i) + f_{\theta^*}(x_i) - y_i\|^2 + \mathcal{R}(\theta), \tag{5.1}$$

where $\mathcal{R}(\theta)$ is a regularization term coming from the prior (see Lemma 3, Osband et al. (2018)). Osband et al. (2018) argue (by analogy to the case of Bayesian linear regression) that the ensemble $\mathcal{F}$ is an approximation of the posterior.

If we specialize the regression targets $y_i$ to be zero, then the optimization problem $\arg\min_\theta \mathbb{E}_{(x_i,y_i)\sim D}\|f_\theta(x_i)+f_{\theta^*}(x_i)\|^2$ is equivalent to distilling a randomly drawn function from the prior. Seen from this perspective, each coordinate of the output of the predictor and target networks would correspond to a member of an ensemble (with parameter sharing amongst the ensemble), and the MSE would be an estimate of the predictive variance of the ensemble (assuming the ensemble is unbiased). In other words the distillation error could be seen as a quantification of uncertainty in predicting the constant zero function.

### 5.2.3 Combining intrinsic and extrinsic returns

In preliminary experiments that used only intrinsic rewards, treating the problem as non-episodic resulted in better exploration. In that setting the return is not truncated at “game over". We argue that this is a natural way to do exploration in simulated environments, since the agent’s intrinsic return should be related to all the novel states that it could find in the future, regardless of whether they all occur in one episode or are spread over several. It is also argued in (Burda et al., 2018b) that using episodic intrinsic rewards can leak information about the task to the agent.

We also argue that this is closer to how humans explore games. For example let’s say Alice is playing a videogame and is attempting a tricky maneuver to reach a suspected secret room. Because the maneuver is tricky the chance of a game over is high, but the payoff to Alice’s curiosity will be high if she succeeds. If Alice is modelled as an episodic reinforcement learning agent, then her future return will be exactly zero if she gets a game over, which might make her overly risk averse. The real cost of a game over to Alice is the opportunity cost incurred by having to play through the game from the beginning (which is presumably less interesting to Alice having played the game for some time).

However using non-episodic returns for extrinsic rewards could be exploited by a strategy that finds a reward close to the beginning of the game, deliberately restarts the game by getting a game over, and repeats this in an endless cycle.

It is not obvious how to estimate the combined value of the non-episodic stream of intrinsic rewards $i_t$ and the episodic stream of extrinsic rewards $e_t$. Our solution is to observe that the return is linear in the rewards and so can be decomposed as a sum $R = R_E + R_I$ of the extrinsic and intrinsic returns respectively. Hence we can fit two value heads $V_E$ and $V_I$ separately using their respective returns, and combine them to give the value function $V = V_E + V_I$. This same idea can also be used to combine reward streams with different discount factors.

Note that even where one is not trying to combine episodic and non-episodic reward streams, or reward streams with different discount factors, there may still be a benefit to having separate value functions since there is an additional supervisory signal to the value function. This may be especially important for exploration bonuses since the extrinsic reward function is stationary whereas the intrinsic reward function is non-stationary.

### 5.2.4 Reward and Observation Normalization

One issue with using prediction error as an exploration bonus is that the scale of the reward can vary greatly between different environments and at different points in time, making it difficult to choose hyperparameters that work in all settings. In order to keep the rewards on a consistent scale we normalized the intrinsic reward by dividing it by a running estimate of the standard deviations of the intrinsic returns.

Observation normalization is often important in deep learning but it is crucial when using a random neural network as a target, since the parameters are frozen and hence cannot adjust to the scale of different datasets. Lack of normalization can result in the variance of the embedding being extremely low and carrying little information about the inputs. To address this issue we use an observation normalization scheme often used in continuous control problems whereby we whiten each dimension by subtracting the running mean and then dividing by the running standard deviation. We then clip the normalized observations to be between -5 and 5. We initialize the normalization parameters by stepping a random agent in the environment for a small number of steps before beginning optimization. We use the same observation normalization for both predictor and target networks but not the policy network.

## 5.3 Experiments

We begin with an intrinsic reward only experiment on Montezuma's Revenge in Section 5.3.1 to isolate the inductive bias of the RND bonus, follow by extensive ablations of RND on Montezuma's Revenge in Sections 5.3.2-5.3.4 to understand the factors that contribute to RND's performance, and conclude with a comparison to baseline methods on 6 hard exploration Atari games in Section 5.3.6. For details of hyperparameters and

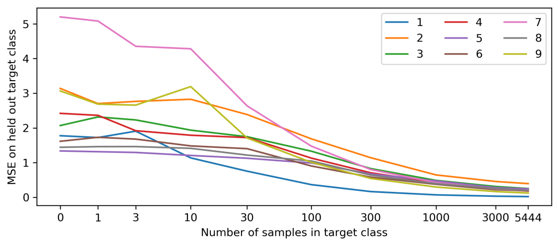


Figure 5.2: Novelty detection on MNIST: a predictor network mimics a randomly initialized target network. The training data consists of varying proportions of images from class "0" and a target class. Each curve shows the test MSE on held out target class examples plotted against the number of training examples of the target class (log scale).

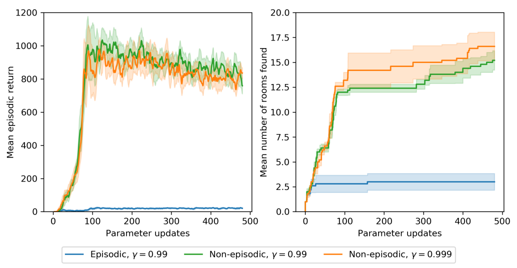


Figure 5.3: Mean episodic return and number of rooms found by pure exploration agents on Montezuma's Revenge trained without access to the extrinsic reward. The agents explores more in the non-episodic setting (see also Section 5.2.3)

architectures we refer the reader to Appendices C.3 and C.4. Most experiments are run for 30K rollouts of length 128 per environment with 128 parallel environments, for a total of 1.97 billion frames of experience.

## 5.3.1 Pure exploration

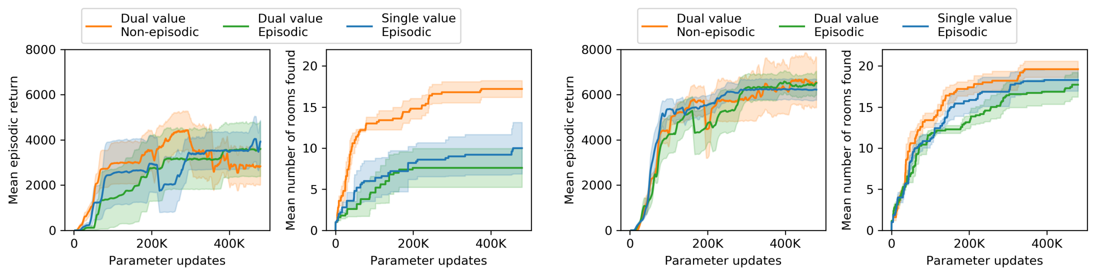
In this section we explore the performance of RND in the absence of any extrinsic reward. In Section 5.2.3 we argued that exploration with RND might be more natural in the non-episodic setting. By comparing the performance of the pure exploration agent in episodic and non-episodic settings we can see if this observation translates to improved exploration performance.

We report two measures of exploration performance in Figure 5.3: mean episodic return, and the number of rooms the agent finds over the training run. Since the pure exploration agent is not aware of the extrinsic rewards or number of rooms, it is not directly optimizing for any of these measures. However obtaining some rewards in Montezuma's Revenge (like getting the key to open a door) is required for accessing more interesting states in new rooms, and hence we observe the extrinsic reward increasing over time up to some point. The best return is achieved when the agent interacts with some of the objects, but the agent has no incentive to keep doing the same once such interactions become repetitive, hence returns are not consistently high.

We clearly see in Figure 5.3 that on both measures of exploration the non-episodic agent performs best, consistent with the discussion in Section 5.2.3. The non-episodic setting with $\gamma_I = 0.999$ explores more rooms than $\gamma_I = 0.99$, with one of the runs exploring 21 rooms. The best return achieved by 4 out 5 runs of this setting was 6,700.

## 5.3.2 Combining episodic and non-episodic returns

In Section 5.3.1 we saw that the non-episodic setting resulted in more exploration than the episodic setting when exploring without any extrinsic rewards. Next we consider whether this holds in the case where we combine intrinsic and extrinsic rewards. As discussed in Section 5.2.3 in order to combine episodic and non-episodic reward streams we require two value heads. This also raises the question of whether it is better to have two value heads even when both reward streams are episodic. In Figure 5.4 we compare episodic intrinsic rewards to non-episodic intrinsic rewards combined with episodic extrinsic rewards, and additionally two value heads versus one for the episodic case. The discount factors are $\gamma_I = \gamma_E = 0.99$.



(a) RNN policies (b) CNN policies

Figure 5.4: Different ways of combining intrinsic and extrinsic rewards. Combining non-episodic stream of intrinsic rewards with the episodic stream of extrinsic rewards outperforms combining episodic versions of both steams in terms of number of explored rooms, but performs similarly in terms of mean return. Single value estimate of the combined stream of episodic returns performs a little better than the dual value estimate. The differences are more pronounced with RNN policies. CNN runs are more stable than the RNN counterparts.

In Figure 5.4 we see that using a non-episodic intrinsic reward stream increases the number of rooms explored for both CNN and RNN policies, consistent with the experiments in Section 5.3.1, but that the difference is less dramatic, likely because the extrinsic reward is able to preserve useful behaviors. We also see that the difference is less pronounced for the CNN experiments, and that the RNN results tend to be less stable and perform worse for $\gamma_E = 0.99$.

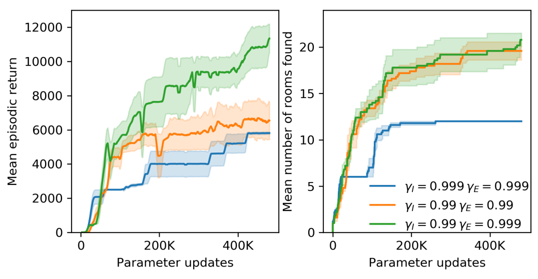


Figure 5.5: Performance of different discount factors for intrinsic and extrinsic reward streams. A higher discount factor for the extrinsic rewards leads to better performance, while for intrinsic rewards it hurts exploration.

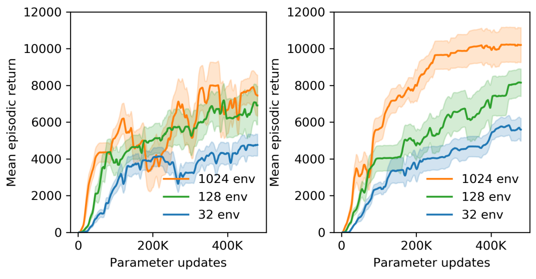


Figure 5.6: Mean episodic return improves as the number of parallel environments used for collecting the experience increases for both the CNN policy (left) and the RNN policy (right). The runs have processed 0.5,2, and 16B frames.

Contrary to our expectations (Section 5.2.3) using two value heads did not show any benefit over a single head in the episodic setting. Nevertheless having two value heads is necessary for combining reward streams with different characteristics, and so all further experiments use two value heads.

### 5.3.3 Discount factors

Previous experiments (Salimans and Chen, 2018; Pohlen et al., 2018; Garmulewicz et al., 2018) solving Montezuma's Revenge using expert demonstrations used a high discount factor to achieve the best performance, enabling the agent to anticipate rewards far into the future. We compare the performance of the RND agent with $\gamma_E \in \{0.99, 0.999\}$ and $\gamma_I = 0.99$. We also investigate the effect of increasing $\gamma_I$ to 0.999. The results are shown in Figure 5.5.

In Figure 5.5 we see that increasing $\gamma_E$ to 0.999 while holding $\gamma_I$ at 0.99 greatly improves performance. We also see that further increasing $\gamma_I$ to 0.999 hurts performance. This is at odds with the results in Figure 5.3 where increasing $\gamma_I$ did not significantly impact performance.

### 5.3.4 Scaling up training

In this section we report experiments showing the effect of increased scale on training. The intrinsic rewards are non-episodic with $\gamma_I = 0.99$, and $\gamma_E = 0.999$.

To hold the rate at which the intrinsic reward decreases over time constant across experiments with different numbers of parallel environments, we downsample the batch size when training the predictor to match the batch size with 32 parallel environments (for full details see Appendix C.4). Larger numbers of environments results in larger batch sizes per update for training the policy, whereas the predictor network batch size remains constant. Since the intrinsic reward disappears over time it is important for the policy to learn to find and exploit these transitory rewards, since they act as stepping-stones to nearby novel states.

Figure 5.6 shows that agents trained with larger batches of experience collected from more parallel environments obtain higher mean returns after similar numbers of updates. They also achieve better final performance. This effect seems to saturate earlier for the CNN policy than for the RNN policy.

We allowed the RNN experiment with 32 parallel environments to run for more time, eventually reaching a mean return of 7,570 after processing 1.6 billion frames over 1.6 million parameter updates. One of these runs visited all 24 rooms, and passed the first level once, achieving a best return of 17,500. The RNN experiment with 1024 parallel environments had mean return of 10,070 at the end of training, and yielded one run with mean return of 14,415.

### 5.3.5 Recurrence

Montezuma's Revenge is a partially observable environment even though large parts of the game state can be inferred from the screen. For example the number of keys the agent has appears on the screen, but not where they come from, how many keys have been used in the past, or what doors have been opened. To deal with this partial observability, an agent should maintain a state summarizing the past, for example the state of a recurrent policy. Hence it would be natural to hope for better performance from agents with recurrent policies. Contrary to expectations in Figure 5.4 recurrent policies performed worse than non-recurrent counterparts with $\gamma_E = 0.99$. However in Figure 5.6 the RNN policy with $\gamma_E = 0.999$ outperformed the CNN counterpart at each scale[1]. Comparison of Figures 5.7 and C.2 shows that across multiple games the RNN

[1]The results in Figure 5.5 for the CNN policy were obtained as an average of 5 random seeds. When we ran 10 different seeds for the best performing setting for Figure 5.6 we found a large discrepancy in performance. This discrepancy is likely explained by the fact that the distribution of results on Montezuma's Revenge dominated by effects of discrete choices (such as going left or right from the first room), and hence contains a preponderance of outliers. In addition, the results in Figure 5.5 were run with an earlier version of our code base and it is possible that subtle differences between that version and

policy outperforms the CNN more frequently than the other way around.

### 5.3.6 Comparison to baselines

In this section we compare RND to two baselines: PPO without an exploration bonus and an alternative exploration bonus based on forward dynamics error. We evaluate RND's performance on six hard exploration Atari games: Gravitar, Montezuma's Revenge, Pitfall!, Private Eye, Solaris, and Venture. We first compare to the performance of a baseline PPO implementation without intrinsic reward. For RND the intrinsic rewards are non-episodic with $\gamma_I = 0.99$, while $\gamma_E = 0.999$ for both PPO and RND. The results are shown in Figure 5.7 for the RNN policy and summarized in Table 5.1 (see also Figure C.2 for the CNN policy).

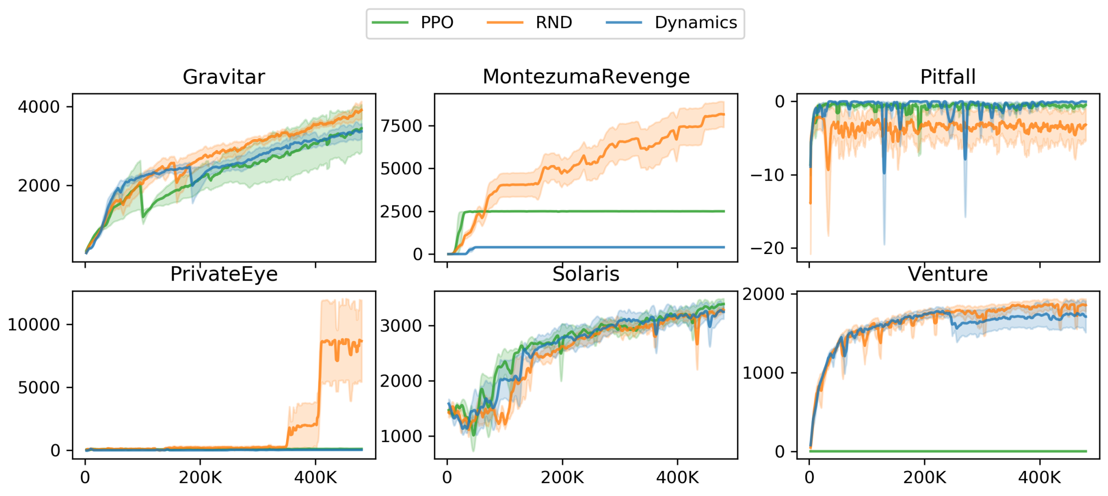


Figure 5.7: Mean episodic return of RNN-based policies: RND, dynamics-based exploration method, and PPO with extrinsic reward only on 6 hard exploration Atari games. RND achieves state of the art performance on Gravitar, Montezuma's Revenge, and Venture, significantly outperforming PPO on the latter two.

In Gravitar we see that RND does not consistently exceed the performance of PPO. However both exceed average human performance with an RNN policy, as well as the previous state of the art. On Montezuma's Revenge and Venture RND significantly outperforms PPO, and exceeds state of the art performance and average human performance. On Pitfall! both algorithms fail to find any positive rewards. This is a typical result for this game, as the extrinsic positive reward is very sparse. On

the publicly released one have contributed to the discrepancy. The results in Figure 5.6 were reproduced with the publicly released code and so we suggest that future work compares against these results.

Private Eye RND's performance exceeds that of PPO. On Solaris RND's performance is comparable to that of PPO.

Next we consider an alternative exploration bonus based on forward dynamics error. There are numerous previous works using such a bonus (Schmidhuber, 1991c; Stadie et al., 2015; Achiam and Sastry, 2017; Pathak et al., 2017; Burda et al., 2018b). Fortuitously Burda et al. (2018b) show that training a forward dynamics model in a random feature space typically works as well as any other feature space when used to create an exploration bonus. This means that we can easily implement an apples to apples comparison and change the loss in RND so the predictor network predicts the random features of the next observation given the current observation and action, while holding fixed all other parts of our method such as dual value heads, non-episodic intrinsic returns, normalization schemes etc. This provides an ablation of the prediction problem defining the exploration bonus, while also being representative of a class of prior work using forward dynamics error. Our expectation was that these methods should be fairly similar except where the dynamics-based agent is able to exploit non-determinism in the environment to get intrinsic reward.

Figure 5.7 shows that dynamics-based exploration performs significantly worse than RND with the same CNN policy on Montezuma's Revenge, PrivateEye, and Solaris, and performs similarly on Venture, Pitfall, and Gravitar. By analyzing agent's behavior at convergence we notice that in Montezuma's Revenge the agent oscillates between two rooms. This leads to an irreducibly high prediction error, as the non-determinism of sticky actions makes it impossible to know whether, once the agent is close to crossing a room boundary, making one extra step will result in it staying in the same room, or crossing to the next one. This is a manifestation of the 'noisy TV' problem, or aleatoric uncertainty discussed in Section 5.2.2.1. Similar behavior emerges in PrivateEye and Pitfall!. In Table 5.1 the final training performance for each algorithm is listed, alongside the state of the art from previous work and average human performance.

### 5.3.7 Qualitative Analysis: Dancing with skulls

By observing the RND agent, we notice that frequently once it obtains all the extrinsic rewards that it knows how to obtain reliably (as judged by the extrinsic value function), the agent settles into a pattern of behavior where it keeps interacting with potentially dangerous objects. For instance in Montezuma's Revenge the agent jumps back and forth over a moving skull, moves in between laser gates, and gets on and off disappearing

| | Gravitar | Montezuma's Revenge | Pitfall! | PrivateEye | Solaris | Venture |
|---|---|---|---|---|---|---|
| RND | **3,906** | **8,152** | -3 | 8,666 | 3,282 | **1,859** |
| PPO | 3,426 | 2,497 | 0 | 105 | 3,387 | 0 |
| Dynamics | 3,371 | 400 | 0 | 33 | 3,246 | 1,712 |
| SOTA | 2,209[1] | 3,700[2] | **0** | **15,806**[2] | **12,380**[1] | **1,813**[3] |
| Avg. Human | 3,351 | 4,753 | 6,464 | 69,571 | 12,327 | 1,188 |

Table 5.1: Comparison to baselines results. Final mean performance for various methods. State of the art results taken from: [1] (Fortunato et al., 2018) [2] (Bellemare et al., 2016) [3] (Horgan et al., 2018)

bridges. We also observe similar behavior in Pitfall!. It might be related to the very fact that such dangerous states are difficult to achieve, and hence are rarely represented in agent's past experience compared to safer states.

## 5.4 Related Work

**Exploration.** Count-based exploration bonuses are a natural and effective way to do exploration (Strehl and Littman, 2008) and a lot of work has studied how to tractably generalize count bonuses to large state spaces (Bellemare et al., 2016; Fu et al., 2017; Ostrovski et al., 2018a; Tang et al., 2017a; Machado et al., 2020; Fox et al., 2018).

Another class of exploration methods rely on errors in predicting dynamics (Schmidhuber, 1991c; Stadie et al., 2015; Achiam and Sastry, 2017; Pathak et al., 2017; Burda et al., 2018b). As discussed in Section 5.2.2, these methods are subject to the 'noisy TV' problem in stochastic or partially-observable environments. This has motivated work on exploration via quantification of uncertainty (Still and Precup, 2012; Houthooft et al., 2016) or prediction improvement measures (Schmidhuber, 1991a; Oudeyer et al., 2007; Lopes et al., 2012; Achiam and Sastry, 2017).

Other methods of exploration include adversarial self-play (Sukhbaatar et al., 2018), maximizing empowerment (Gregor et al., 2017), parameter noise (Plappert et al., 2018; Fortunato et al., 2018), identifying diverse policies (Eysenbach et al., 2018; Achiam et al., 2018), and using ensembles of value functions (Osband et al., 2018, 2016; Chen et al., 2017).

**Montezuma's Revenge.** Early neural-network based reinforcement learning algorithms that were successful on a significant portion of Atari games (Mnih et al., 2015b, 2016; Hessel et al., 2018) failed to make meaningful progress on Montezuma's Revenge, not finding a way out of the first room reliably. This is not necessarily a failure of exploration, as even a random agent finds the key in the first room once every few hundred thousand steps, and escapes the first room every few million steps. Indeed, a mean return of about 2,500 can be reliably achieved without special exploration methods (Horgan et al., 2018; Espeholt et al., 2018; Oh et al., 2018).

Combining DQN with a pseudo-count exploration bonus Bellemare et al. (2016) set a new state of the art performance, exploring 15 rooms and getting best return of 6,600. Since then a number of other works have achieved similar performance (O'Donoghue et al., 2018; Ostrovski et al., 2018b; Machado et al., 2020; Osband et al., 2018), without exceeding it.

Special access to the underlying RAM state can also be used to improve exploration by using it to hand-craft exploration bonuses (Kulkarni et al., 2016; Tang et al., 2017a; Stanton and Clune, 2018). Even with such access previous work achieves performance inferior to average human performance.

Expert demonstrations can be used effectively to simplify the exploration problem in Montezuma's Revenge, and a number of works (Salimans and Chen, 2018; Pohlen et al., 2018; Aytar et al., 2018; Garmulewicz et al., 2018) have achieved performance comparable to or better than that of human experts. Learning from expert demonstrations benefits from the game's determinism. The suggested training method (Machado et al., 2018) to prevent an agent from simply memorizing the correct sequence of actions is to use sticky actions (i.e. randomly repeating previous action) has not been used in these works. In this work we use sticky actions and thus don't rely on determinism.

**Random features.** Features of randomly initialized neural networks have been extensively studied in the context of supervised learning (Rahimi and Recht, 2008; Saxe et al., 2011; Jarrett et al., 2009; Yang et al., 2015). More recently they have been used in the context of exploration (Osband et al., 2018; Burda et al., 2018b). The work Osband et al. (2018) provides motivation for random network distillation as discussed in Section 5.2.2.

**Vectorized value functions.** Pong et al. (2018) find that a vectorized value function (with coordinates corresponding to additive factors of the reward) improves their method. Bellemare et al. (2017) parametrize the value as a linear combination of value heads that estimate probabilities of discretized returns. However the Bellman backup equation used there is not itself vectorized.

## 5.5 Discussion

This paper introduced an exploration method based on random network distillation and experimentally showed that the method is capable of performing directed exploration on several Atari games with very sparse rewards. These experiments suggest that progress on hard exploration games is possible with relatively simple generic methods, especially when applied at scale. They also suggest that methods that are able to treat the stream of intrinsic rewards separately from the stream of extrinsic rewards (for instance by having separate value heads) can benefit from such flexibility.

We find that the RND exploration bonus is sufficient to deal with local exploration, i.e. exploring the consequences of short-term decisions, like whether to interact with a particular object, or avoid it. However global exploration that involves coordinated decisions over long time horizons is beyond the reach of our method.

To solve the first level of Montezuma's Revenge, the agent must enter a room locked behind two doors. There are four keys and six doors spread throughout the level. Any of the four keys can open any of the six doors, but are consumed in the process. To open the final two doors the agent must therefore forego opening two of the doors that are easier to find and that would immediately reward it for opening them.

To incentivize this behavior the agent should receive enough intrinsic reward for saving the keys to balance the loss of extrinsic reward from using them early on. From our analysis of the RND agent's behavior, it does not get a large enough incentive to try this strategy, and only stumbles upon it rarely.

Solving this and similar problems that require high level exploration is an important direction for future work.

## 5.6 2021 Retrospective

The paper introduced an easy to implement and computationally cheap method for computing an exploration bonus that gave good results in Atari games. For this reason it is now a commonly used baseline in exploration papers (a recent example would be (Guo et al., 2021)), and also used as a component of more complex state of the art systems (see for example (Badia et al., 2019) where it is used as their 'life-long novelty module' and the follow-up work 'Agent 57' the first paper to exceed the human baseline on all Atari games (Badia et al., 2020)).

A number of papers have considered within-episode exploration an alternative or complement to across-episode exploration. Across-episode exploration tries to maximize the information gain or novelty across all the trajectories that the agent accumulates throughout optimization. One issue with this approach is, by design, things that were previously rewarding to agent become unrewarding as they get observed multiple times, and in principle in the limit all intrinsic reward would go to zero and one would end up only with an agent that acted randomly. Even before this theoretical behavior, in practice the agent may lose track of the frontier of interesting states because the reward for getting there is decreasing, a problem referred to as 'detachment' (Ecoffet et al., 2021). By contrast a within-episode exploration objective, such as maxmizing the number of distinct states seen in an episode. An advantage of this approach is that the reward can lead to a stable behavior at convergence (for example a maximally efficient tour of the environment) that be more useful as an initialization for finetuning. One example of within-episode exploration could be (Savinov et al., 2018). State of the art approaches to curiosity often combine these two signals, the across-episode signal leading to eventually seeing everything there is to see, and the within-episode signal stabilizing the learning and preserving useful exploratory behaviors (Badia et al., 2019).

In addition to using RND for exploration in reinforcement learning there have been a number of papers using it for outlier detection. A recent example of outlier detection would be (Choi and Chung, 2019) where they use an RND-derived score to detect out of distribution images. A related recent work (Fowl et al., 2020) uses an RND score to detect the distance between distributions of text, and show that you can use this score to tell apart real text from model generated text, for example.

In the paper we noted the connection to a work on randomized prior functions (Osband et al., 2018) which shows that regressing on random networks, in the case of Bayesian linear regression, recovers the correct uncertainty. This result is only shown for linear networks however and so its use for deep nonlinear networks is justified only by analogy to this case. In (Ciosek et al., 2019) they show an interesting extension of the linear case to arbitrary neural networks and show that the uncertainty estimates from RND-like bonuses are conservative in expectation, that is the uncertainty estimates are higher than the true uncertainty, but at the same time they will eventually go to zero in the limit of infinite data. The strongest results in the paper rely on the idea that large randomly initialised networks can be approximated by Gaussian processes which is reasonable for randomly initialised networks since there are a number of papers showing this connection, but not for the case of trained neural networks. Using an ensemble of trained neural networks is one natural way to think about having an informative prior in this setting which could be one way to encode extra information about the task if we have any. This would be an interesting direction to explore in future work.

# Chapter 6

# Conclusion

In this thesis I have given three examples of learning from alternative sources of supervision. The first involved learning when data is given as a collection of sets. The second was unsupervised reinforcement learning where predicting the dynamics of the environment gave a pretext task from which to derive an exploration bonus. The third used an alternative pretext task in the reinforcement learning setting combined with a sparse extrinsic reward signal. The overall contribution was to push the frontier of what was known to be possible at the time when dealing with extreme cases of learning from alternative supervision, as well as practical and technical tips and tricks for achieving such results.

Whilst it is gratifying to see interesting results from unsupervised learning, from a more practical point of view, purely unsupervised learning rarely makes sense. Except in rare cases, it is always possible to invest resources into acquiring labelled examples. Thus I think that the key challenge for research in this area is how to best improve the sample complexity of learning a particular task through the use of unsupervised learning, or alternative sources of supervision, and how to most efficiently allocate one's resources to the acquisition of labelled data. The former case is the purview of semi-supervised learning, transfer learning and multi-task learning and was partially explored in the RND paper (Burda et al., 2018c). A systematic study of the transfer sample-efficiency gains from first training on dataset A and finetuning on dataset B for the case of language models was recently investigated in (Hernandez et al., 2021). The latter case is active learning which has been extensively studied for the case of classification but less so for generative modelling and reinforcement learning.

A related point on exploration objectives is that although progress has been made on exploration bonuses for reinforcement learning that are computationally tractable and tend to improve sample-complexity on highly sparse reward tasks, I have come to think it unlikely that we will be able to find a perfect unsupervised objective that acts as expected in all cases. For example various thought-experiments and simulations show that using the prediction error of a forward dynamics model as an exploration bonus can have undesirable side effects such as the 'noisy-TV' problem, since there will be irreducible sources of entropy in any realistic environment that will persist no matter how much data the agent is able to gather. Thus to the extent that such approaches work we have gotten lucky.

The natural principled approach to exploration is to use Bayesian methods for learning a dynamics model and using as a reward the amount of information gained about the parameters of the dynamics model. In practice this approach is rarely taken because of the computational expense of Bayesian neural networks. Even if we were able to implement this approach efficiently, I argue that we would still often be disappointed with the results since there is no reason why a generic information gathering objective will prioritise the most important bits in an environment, and for more realistic environments, the amount of things that could be learned about it is, to practical intents and purposes, unbounded. Thus whilst we might wish a curious agent to learn the most natural things like how to navigate, the layout of its environment and how to manipulate a variety of objects effectively, it may instead choose to specialise in predicting the way dust moves in the air.

Given this I think we will eventually need to move beyond generic approaches to objectives flexibly tailored to human wishes. One simple way to begin such a research direction to elicit a novelty kernel from human experts, that is a mapping $k(s,s')$ that specifies how novel or interestingly different state/observation/trajectory $s$ is from $s'$. This can then be used to derive simple exploration bonuses. This idea relates to work on learning reward functions from human preferences (Christiano et al., 2017), but is a dynamic reward that is a function of the agent's whole collection of experiences rather than a function of a transition or state.

Overall, now that we have seen in many areas, including in the papers contributed to this thesis, that weakly-supervised, unsupervised and semi-supervised learning is having a beneficial effect, especially for modalities that are 'internet-scale' such as images and text, I hope that attention will turn to maximizing use of our most limited resource: human time.

# Appendix A

# Towards a Neural Statistician

## A.1 Pseudocode

**Algorithm 1** Sampling a dataset of size $k$

```
sample $c \sim p(c)$
for $i = 1$ to $k$ do
    sample $z_{i,L} \sim p(z_L|c;\theta)$
    for $j = L-1$ to $1$ do
        sample $z_{i,j} \sim p(z_j|z_{i,j+1},c;\theta)$
    end for
    sample $x_i \sim p(x|z_{i,1},\ldots,z_{i,L},c;\theta)$
end for
```

**Algorithm 2** Sampling a dataset of size $k$ conditioned on a dataset of size $m$

```
$\mu_c, \sigma_c^2 \leftarrow q(c|x_1,\ldots,x_m;\phi)$ {Calculate approximate posterior over $c$ using statistic network.}
$c \leftarrow \mu_c$ {Set $c$ to be the mean of the approximate posterior.}
for $i = 1$ to $k$ do
    sample $z_{i,L} \sim p(z_L|c;\theta)$
    for $j = L-1$ to $1$ do
        sample $z_{i,j} \sim p(z_j|z_{i,j+1},c;\theta)$
    end for
    sample $x_i \sim p(x|z_{i,1},\ldots,z_{i,L},c;\theta)$
end for
```

**Algorithm 3** Selecting a representative sample of size $k$

```
$S \leftarrow \{x_1, \ldots, x_m\}$
$I \leftarrow \{1, \ldots, m\}$
$S_I = \{x_i \in S : i \in I\}$
$N_{S_I} \leftarrow q(c|S_I; \phi)$ {Calculate approximate posterior over $c$ using statistic network.}
for $i = 1$ to $k$ do
    $t \leftarrow argmin_{j \in I} D_{KL}\left(N_S \| N_{S_{I-j}}\right)$
    $I \leftarrow I - t$
end for
```

**Algorithm 4** $K$-way few-shot classification

```
$D_0, \ldots, D_K \leftarrow$ sets of labelled examples for each class
$x \leftarrow$ datapoint to be classified
$N_x \leftarrow q(c|x; \phi)$ {approximate posterior over $c$ given query point}
for $i = 1$ to $K$ do
    $N_i \leftarrow q(c|D_i; \phi)$
end for
$\hat{y} \leftarrow argmin_i D_{KL}(N_i \| N_x)$
```

## A.2 Further Experimental Details

### A.2.1 Omniglot

**Shared encoder** $x \rightarrow h$

$2\times$ { *conv2d* 64 feature maps with $3 \times 3$ kernels and ELU activations }

*conv2d* 64 feature maps with $3 \times 3$ kernels, stride 2 and ELU activations

$2\times$ {*conv2d* 128 feature maps with $3 \times 3$ kernels and ELU activations }

*conv2d* 128 feature maps with $3 \times 3$ kernels, stride 2 and ELU activations

$2\times$ { *conv2d* 256 feature maps with $3 \times 3$ kernels and ELU activations }

*conv2d* 256 feature maps with $3 \times 3$ kernels, stride 2 and ELU activations

**Statistic network** $q(c|D;\phi) : h_1, \ldots, h_k \rightarrow \mu_c, \sigma_c^2$

*fully-connected* layer with 256 units and ELU activations
*sample-dropout* and *concatenation* with number of samples
*average pooling* within each dataset
$2\times$ {*fully-connected* layer with 256 units and ELU activations }
*fully-connected* linear layers to $\mu_c$ and $\log\sigma_c^2$

**Inference network** $q(z|x,c;\phi) : h, c \rightarrow \mu_z, \sigma_z^2$

*concatenate c* and *h*
$3\times$ {*fully-connected* layer with 256 units and ELU activations }
*fully-connected* linear layers to $\mu_z$ and $\log\sigma_z^2$

**Latent decoder network** $p(z|c;\theta) : c \rightarrow \mu_z, \sigma_z^2$

$3\times$ {*fully-connected* layer with 256 units and ELU activations }
*fully-connected* linear layers to $\mu_z$ and $\log\sigma_z^2$

**Observation decoder network** $p(x|c,z;\theta) : c, z \rightarrow \mu_x$

*concatenate z* and *c*
*fully-connected* linear layers with $4 \cdot 4 \cdot 256$ units
$2\times$ { *conv2d* 256 feature maps with $3 \times 3$ kernels and ELU activations }
*deconv2d* 256 feature maps with $2 \times 2$ kernels, stride 2, ELU activations
$2\times$ { *conv2d* 128 feature maps with $3 \times 3$ kernels and ELU activations }
*deconv2d* 128 feature maps with $2 \times 2$ kernels, stride 2, ELU activations
$2\times$ { *conv2d* 64 feature maps with $3 \times 3$ kernels and ELU activations }
*deconv2d* 64 feature maps with $2 \times 2$ kernels, stride 2, ELU activations
*conv2d* 1 feature map with $1 \times 1$ kernels, sigmoid activations

### A.2.2 Youtube faces

| **Shared encoder** $x \to h$ |
|---|
| $2\times$ { *conv2d* 32 feature maps with $3 \times 3$ kernels and ELU activations } |
| *conv2d* 32 feature maps with $3 \times 3$ kernels, stride 2 and ELU activations |
| $2\times$ {*conv2d* 64 feature maps with $3 \times 3$ kernels and ELU activations } |
| *conv2d* 64 feature maps with $3 \times 3$ kernels, stride 2 and ELU activations |
| $2\times$ { *conv2d* 128 feature maps with $3 \times 3$ kernels and ELU activations } |
| *conv2d* 128 feature maps with $3 \times 3$ kernels, stride 2 and ELU activations |
| $2\times$ { *conv2d* 256 feature maps with $3 \times 3$ kernels and ELU activations } |
| *conv2d* 256 feature maps with $3 \times 3$ kernels, stride 2 and ELU activations |

| **Statistic network** $q(c\|D,\phi) : h_1,\ldots,h_k \to \mu_c, \sigma_c^2$ |
|---|
| *fully-connected* layer with 1000 units and ELU activations |
| *average pooling* within each dataset |
| *fully-connected* linear layers to $\mu_c$ and $\log\sigma_c^2$ |

| **Inference network** $q(z\|x,c,\phi) : h, c \to \mu_z, \sigma_z^2$ |
|---|
| *concatenate* $c$ and $h$ |
| *fully-connected* layer with 1000 units and ELU activations |
| *fully-connected* linear layers to $\mu_z$ and $\log\sigma_z^2$ |

| **Latent decoder network** $p(z\|c,;\theta) : c \to \mu_z, \sigma_z^2$ |
|---|
| *fully-connected* layer with 1000 units and ELU activations |
| *fully-connected* linear layers to $\mu_z$ and $\log\sigma_z^2$ |

**Observation decoder network** $p(x|c,z;\theta) : c,z \to \mu_x$

*concatenate* $z$ and $c$

*fully-connected* layer with 1000 units and ELU activations

*fully-connected* linear layer with $8 \cdot 8 \cdot 256$ units

$2\times$ { *conv2d* 256 feature maps with $3 \times 3$ kernels and ELU activations }

*deconv2d* 256 feature maps with $2 \times 2$ kernels, stride 2, ELU activations

$2\times$ { *conv2d* 128 feature maps with $3 \times 3$ kernels and ELU activations }

*deconv2d* 128 feature maps with $2 \times 2$ kernels, stride 2, ELU activations

$2\times$ { *conv2d* 64 feature maps with $3 \times 3$ kernels and ELU activations }

*deconv2d* 64 feature maps with $2 \times 2$ kernels, stride 2, ELU activations

$2\times$ { *conv2d* 32 feature maps with $3 \times 3$ kernels and ELU activations }

*deconv2d* 32 feature maps with $2 \times 2$ kernels, stride 2, ELU activations

*conv2d* 3 feature maps with $1 \times 1$ kernels, sigmoid activations

# Appendix B

# Large-Scale Study of Curiosity-Driven Learning

## B.1 Implementation Details

We have released the training code and environments on our website [1]. For full details, we refer the reader to our code and video results in the website.

**Pre-processing:** All experiments were done with pixels. We converted all images to grayscale and resized to size 84x84. We learn the agent's policy and forward dynamics function both on a stack of historical observations $[x_{t-3}, x_{t-2}, x_{t-1}, x_t]$ instead of only using the current observation. This is to capture partial observability in these games. In the case of Super Mario Bros and Atari experiments, we also used a standard frameskip wrapper that repeats each action 4 times.

**Architectures:** Our embedding network and policy networks had identical architectures and were based on the standard convolutional networks used in Atari experiments. The layer we take as features in the embedding network had dimension 512 in all experiments and no nonlinearity. To keep the scale of the prediction error consistent relative to extrinsic reward, in the Unity experiments we applied batchnorm to the embedding network. We also did this for the Mario generalization experiments to reduce covariate shift from level to level. For the VAE auxiliary task and pixel method, we used a similar deconvolutional architecture the exact details of which can be found

[1] Website at https://pathak22.github.io/large-scale-curiosity/

in our code submission. The IDF and forward dynamics networks were heads on top of the embedding network with several extra fully-connected layers of dimensionality 512.

**Hyper-parameters:** We used a learning rate of 0.0001 for all networks. In most experiments, we used 128 parallel environments with the exceptions of the Unity and Roboschool experiments where we could only run 32 parallel environments, and the large scale Mario experiment where we used 1024. We used rollouts of length 128 in all experiments except for the Unity experiments where we used 512 length rollouts so that the network could quickly latch onto the sparse reward. In the initial 9 experiments on Mario and Atari, we used 3 optimization epochs per rollout in the interest of speed. In the Mario scaling, generalization experiments, as well as the Roboschool experiments, we used 6 epochs. In the Unity experiments, we used 8 epochs, again to more quickly take advantage of sparse rewards.

## B.2 Additional Results

### B.2.1 Atari

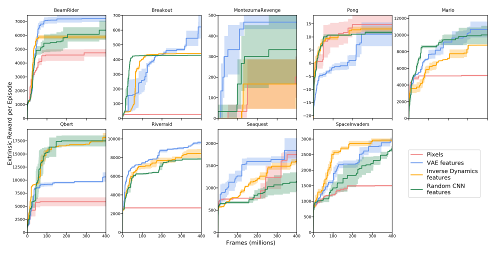
To better measure the amount of exploration, we provide the best return of curiosity-driven agents in figure B.1(a) and the episode lengths in figure B.1(b). Notably on Pong the increasing episode length combined with a plateau in returns shows that the agent maximizes the number of ball bounces, rather than the reward.

Figure B.2 shows the performance of curiosity-driven agents based on Inverse Dynamics and Random features on 48 Atari games.

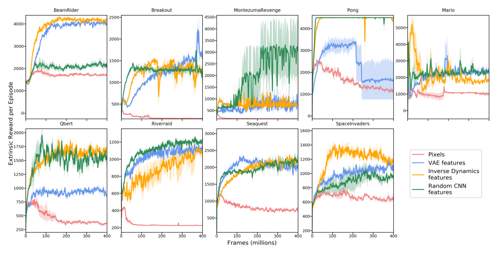
Although not the focus of this paper, for completeness we include some results on combining intrinsic and extrinsic reward on several sparse reward Atari games. When combining with extrinsic rewards, we use the end of the episode signal. The reward used is the extrinsic reward plus 0.01 times the intrinsic reward. The results are shown in Table B.1. We don't observe a large difference between the settings, likely because the combination of intrinsic and extrinsic reward needs to be tuned. We did observe that one of the intrinsic+extrinsic runs on Montezuma's Revenge explored 10 rooms.

### B.2.2 Mario

We show the analogue of the plot shown in Figure 4.3(a) showing max extrinsic returns. See Figure B.3.



(a) Best returns



(b) Episode length

Figure B.1: (a) Left: Best extrinsic returns on eight Atari games and Mario. (c) Right: Mean episode lengths on eight Atari games and Mario.

| Reward | Gravitar | Freeway | Venture | PrivateEye | MontezumaRevenge |
|---|---|---|---|---|---|
| Ext Only | $999.3 \pm 220.7$ | $33.3 \pm 0.6$ | $0 \pm 0$ | $5020.3 \pm 395$ | $1783 \pm 691.7$ |
| Ext + Int | $1165.1 \pm 53.6$ | $32.8 \pm 0.3$ | $416 \pm 416$ | $3036.5 \pm 952.1$ | $2504.6 \pm 4.6$ |

Table B.1: These results compare the mean reward ($\pm$ std-error) after 100 million frames across 3 seeds for an agent trained with intrinsic plus extrinsic reward versus extrinsic reward only. The extrinsic (coefficient $1.0$) and intrinsic reward (coefficient $0.01$) were directly combined without any hyper-parameter tuning. We leave the question on how to optimally combine extrinsic and intrinsic rewards up to future work. This is to emphasize that combining extrinsic with intrinsic rewards is not the focus of the paper, and these experiments are provided just for completeness.

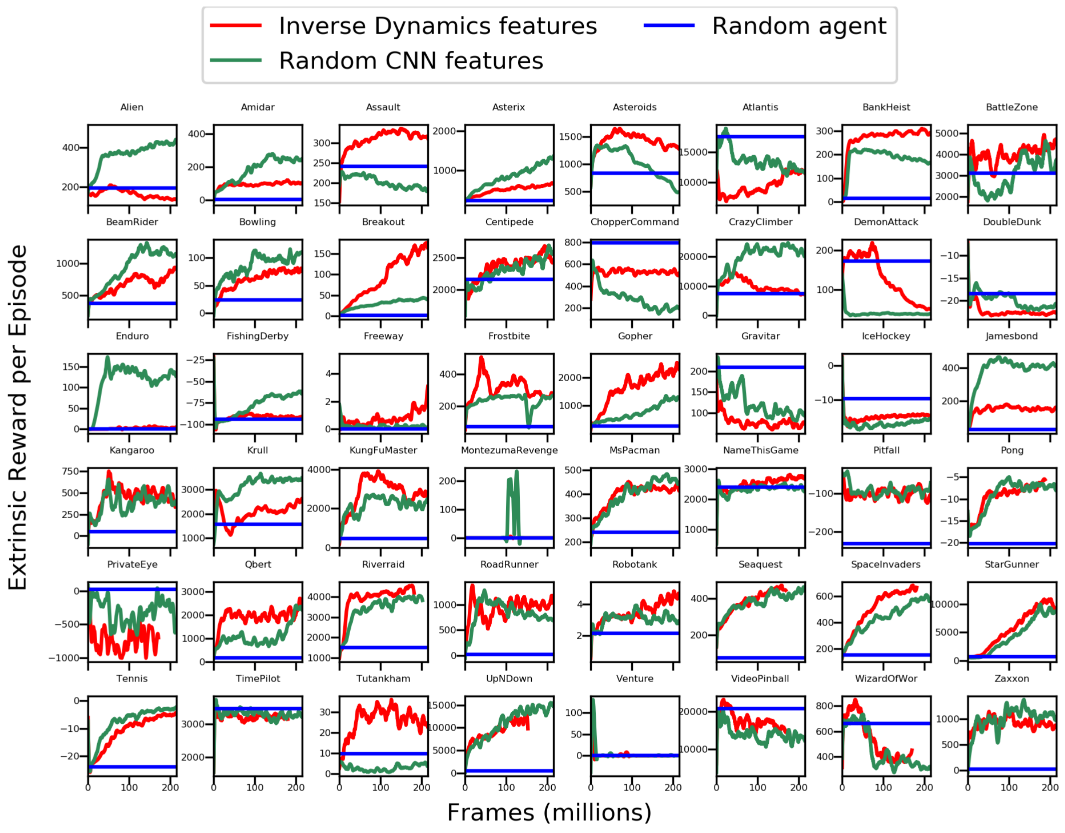


Figure B.2: Pure curiosity-driven exploration (no extrinsic reward, or end-of-episode signal) on 48 Atari games. We observe that the extrinsic returns of curiosity-driven agents often increases despite the agents having no access to the extrinsic return or end of episode signal. In multiple environments, the performance of the curiosity-driven agents is significantly better than that of a random agent, although there are environments where the behavior of the agent is close to random, or in fact seems to minimize the return, rather than maximize it. For the majority of the training process RF perform better than a random agent in about 67% of the environments, while IDF perform better than a random agent in about 71% of the environments.

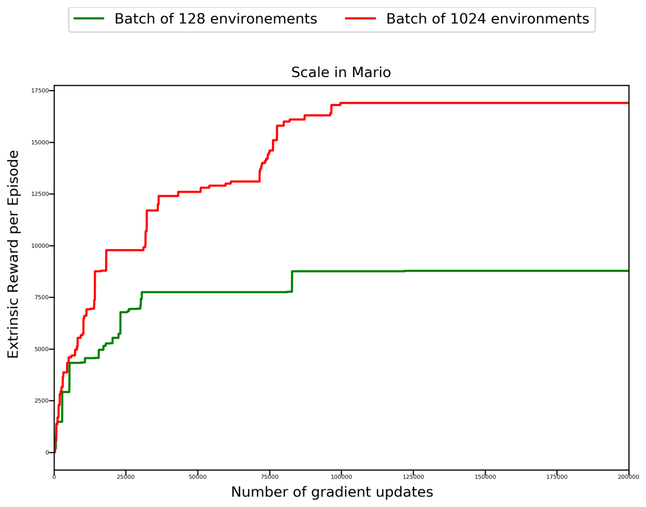


Figure B.3: Best extrinsic returns on the Mario scaling experiments. We observe that larger batches allow the agent to explore more effectively, reaching the same performance in less parameter updates, and also achieving better ultimate scores.

**Algorithm 5** Curiosity-driven Learning

```
Initialize the networks $f(x_t, a_t; \theta_f)$, $\pi(x_t; \theta_\pi)$ and $\phi(x; \theta_\phi)$
$D = \{\}$
for iteration i = 1 to ... do
    for envs in parallel t = 1 to 128 do
        for iteration t = 1 to 128 do
            Sample $a \sim \pi(x_t; \theta_\pi)$ and act using $a$ in the environment
            $D \Leftarrow D + (x_t, a_t, x_{t+1}, r_t)$ where $r_t = \|f(x_t, a_t; \theta_f) - \phi(x_{t+1}; \theta_\phi)\|_2^2$
        end
    end
    for steps k = 1 to 64 do
        Sample batch size of 2048 from $D$ and update using ADAM as follows:
        $\theta_f' := \theta_f - \eta_1 \nabla_{\theta_f} \mathbb{E}\left[\|f(x_t, a_t; \theta_f) - \phi(x_{t+1}; \theta_\phi)\|_2^2\right]$
        $\theta_\phi' := \theta_\phi - \eta_2 \nabla_{\theta_\phi} \mathbb{E}\left[\| \dots \|_2^2\right]$: some auxiliary task
        $\theta_\pi' := \theta_\pi + \eta_3 \nabla_{\theta_\pi} \mathbb{E}_{\pi(x_t; \theta_\pi)}\left[\sum_t r_t\right]$: use PPO with discounted returns
        $\theta_f \Leftarrow \theta_f'$
        $\theta_\phi \Leftarrow \theta_\phi'$
        $\theta_\pi \Leftarrow \theta_\pi'$
    end
end
```

# Appendix C

# Exploration by Random Distillation

## C.1 Reinforcement Learning Algorithm

An exploration bonus can be used with any RL algorithm by modifying the rewards used to train the model (i.e., $r_t = i_t + e_t$). We combine our proposed exploration bonus with a baseline reinforcement learning algorithm PPO (Schulman et al., 2017). PPO is a policy gradient method that we have found to require little tuning for good performance. For algorithmic details see Algorithm 6.

## C.2 RND Pseudo-code

Algorithm 6 gives an overall picture of the RND method. Exact details of the method can be found in the code accompanying this paper.

## C.3 Preprocessing details

Table C.1 contains details of how we preprocessed the environment for our experiments. We followed the recommendations in Machado et al. (2018) in using sticky actions in order to make the environments non-deterministic so that memorization of action sequences is not possible. In Table C.2 we show additional preprocessing details for the policy and value networks. In Table C.3 we show additional preprocessing details for the predictor and target networks.

**Algorithm 6** RND pseudo-code

```
$N \leftarrow$ number of rollouts
$N_{\text{opt}} \leftarrow$ number of optimization steps
$K \leftarrow$ length of rollout
$M \leftarrow$ number of initial steps for initializing observation normalization
$t = 0$
Sample state $s_0 \sim p_0(s_0)$
for $m = 1$ to $M$ do
    sample $a_t \sim \text{Uniform}(a_t)$
    sample $s_{t+1} \sim p(s_{t+1}|s_t, a_t)$
    Update observation normalization parameters using $s_{t+1}$
    t += 1
end for
for $i = 1$ to $N$ do
    for $j = 1$ to $K$ do
        sample $a_t \sim \pi(a_t|s_t)$
        sample $s_{t+1}, e_t \sim p(s_{t+1}, e_t|s_t, a_t)$
        calculate intrinsic reward $i_t = \|\hat{f}(s_{t+1}) - f(s_{t+1})\|^2$
        add $s_t, s_{t+1}, a_t, e_t, i_t$ to optimization batch $B_i$
        Update reward normalization parameters using $i_t$
        t += 1
    end for
    Normalize the intrinsic rewards contained in $B_i$
    Calculate returns $R_{I,i}$ and advantages $A_{I,i}$ for intrinsic reward
    Calculate returns $R_{E,i}$ and advantages $A_{E,i}$ for extrinsic reward
    Calculate combined advantages $A_i = A_{I,i} + A_{E,i}$
    Update observation normalization parameters using $B_i$
    for $j = 1$ to $N_{\text{opt}}$ do
        optimize $\theta_\pi$ wrt PPO loss on batch $B_i, R_i, A_i$ using Adam
        optimize $\theta_{\hat{f}}$ wrt distillation loss on $B_i$ using Adam
    end for
end for
```

| Hyperparameter | Value |
|---|---|
| Grey-scaling | True |
| Observation downsampling | (84,84) |
| Extrinsic reward clipping | $[-1, 1]$ |
| Intrinsic reward clipping | False |
| Max frames per episode | 18K |
| Terminal on loss of life | False |
| Max and skip frames | 4 |
| Random starts | False |
| Sticky action probability | 0.25 |

Table C.1: Preprocessing details for the environments for all experiments.

| Hyperparameter | Value |
|---|---|
| Frames stacked | 4 |
| Observation normalization | $x \mapsto x/255$ |

Table C.2: Preprocessing details for policy and value network for all experiments.

| Hyperparameter | Value |
|---|---|
| Frames stacked | 1 |
| Observation normalization | $x \mapsto \text{CLIP}\left((x-\mu)/\sigma, [-5,5]\right)$ |

Table C.3: Preprocessing details for target and predictor networks for all experiments.

## C.4 PPO and RND hyperparameters

In Table C.4 the hyperparameters for the PPO RL algorithm along with any additional hyperparameters used for RND are shown. Complete details for how these hyperparameters are used can be found in the code accompanying this paper.

Initial preliminary experiments with RND were run with only 32 parallel environments. We expected that increasing the number of parallel environments would improve performance by allowing the policy to adapt more quickly to transient intrinsic rewards. This effect could have been mitigated however if the predictor network also learned more quickly. To avoid this situation when scaling up from 32 to 128 environments we kept the effective batch size for the predictor network the same by randomly dropping out elements of the batch with keep probability 0.25. Similarly in our experiments with 256 and 1,024 environments we dropped experience for the predictor with respective probabilities 0.125 and 0.03125.

| Hyperparameter | Value |
|---|---|
| Rollout length | 128 |
| Total number of rollouts per environment | 30K |
| Number of minibatches | 4 |
| Number of optimization epochs | 4 |
| Coefficient of extrinsic reward | 2 |
| Coefficient of intrinsic reward | 1 |
| Number of parallel environments | 128 |
| Learning rate | 0.0001 |
| Optimization algorithm | Adam (Kingma and Ba (2015)) |
| $\lambda$ | 0.95 |
| Entropy coefficient | 0.001 |
| Proportion of experience used for training predictor | 0.25 |
| $\gamma_E$ | 0.999 |
| $\gamma_I$ | 0.99 |
| Clip range | $[0.9, 1.1]$ |
| Policy architecture | CNN |

Table C.4: Default hyperparameters for PPO and RND algorithms for experiments where applicable. Any differences to these defaults are detailed in the main text.

## C.5 Architectures

In this paper we use two policy architectures: an RNN and a CNN. Both contain convolutional encoders identical of those in the standard architecture from (Mnih et al., 2015b). The RNN architecture additionally contains GRU (Cho et al., 2014) cells to capture longer contexts. The layer sizes of the policies were chosen so that the number of parameters matches closely. The architectures of the target and predictor networks also have convolutional encoders identical to the ones in (Mnih et al., 2015b) followed by dense layers. Exact details are given in the code accompanying this paper.

## C.6 Additional experimental results

Figure C.1 compares the performance of RND with an identical algorithm, but with the exploration bonus defined as the reconstruction error of an autoencoder. The

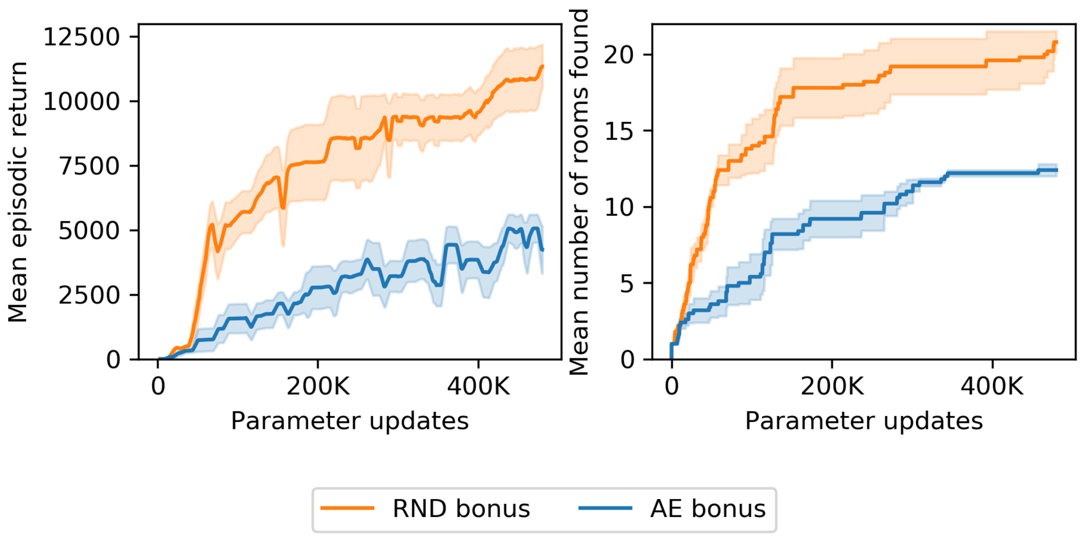


Figure C.1: Comparison of RND with a CNN policy with $\gamma_I = 0.99$ and $\gamma_E = 0.999$ with an exploration defined by the reconstruction error of an autoencoder, holding all other choices constant (e.g. using dual value, treating intrinsic return as non-episodic etc). The performance of the autoencoder-based agent is worse than that of RND, but exceeds that of baseline PPO.

autoencoding task is similar in nature to the random network distillation, as it also obviates the second (though not necessarily the third) sources of prediction error from section 5.2.2.1. The experiment shows that the autoencoding task can also be successfully used for exploration.

Figure C.2 compares the performance of RND to PPO and dynamics prediction-based baselines for CNN policies.

## C.7 Additional Experimental Details

In Table C.5 we show the number of seeds used for each experiment, indexed by figure.

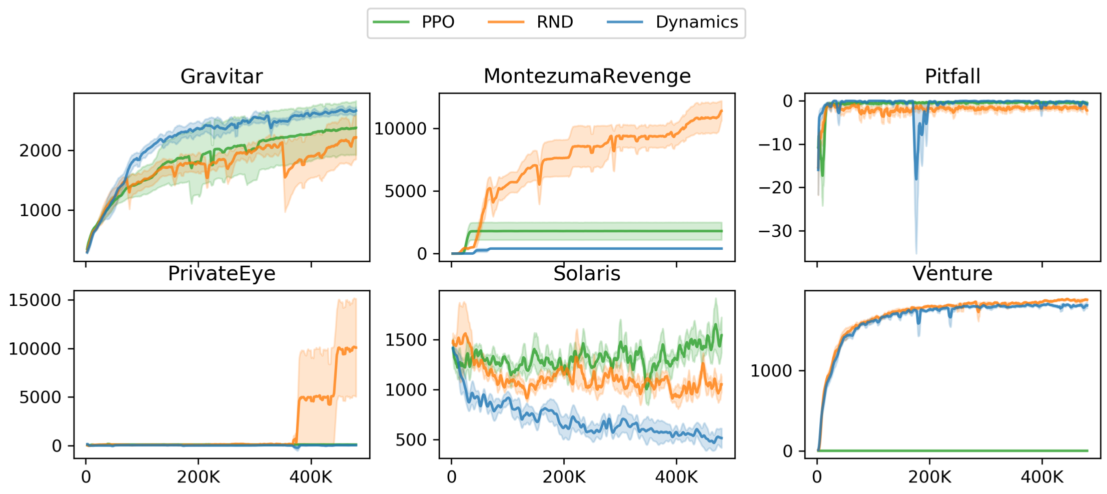


Figure C.2: Mean episodic return of CNN-based policies: RND, dynamics-based exploration method, and PPO with extrinsic reward only on 6 hard exploration Atari games. RND significantly outperforms PPO on Montezuma's Revenge, Private Eye, and Venture.

| Figure number | Number of seeds |
|---|---|
| 1 | NA |
| 2 | 10 |
| 3 | 5 |
| 4 | 5 |
| 5 | 10 |
| 6 | 5 |
| 7 | 3 |
| 8 | 5 |
| 9 | 5 |

Table C.5: The numbers of seeds run for each experiment is shown in the table. The results of each seed are then averaged to provide a mean curve in each figure, and the standard error is used make the shaded region surrounding each curve.

# Bibliography

(2021). http://commoncrawl.org/. 7

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., and et al, Z. C. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org. 22

Achiam, J., Edwards, H., Amodei, D., and Abbeel, P. (2018). Variational option discovery algorithms. *arXiv preprint arXiv:1807.10299*. 67

Achiam, J. and Sastry, S. (2017). Surprise-based intrinsic motivation for deep reinforcement learning. *arXiv:1703.01732*. 34, 45, 56, 58, 66, 67

Andrychowicz, O. M., Baker, B., Chociej, M., Józefowicz, R., McGrew, B., Pachocki, J., Petron, A., Plappert, M., Powell, G., Ray, A., et al. (2020). Learning dexterous in-hand manipulation. *The International Journal of Robotics Research*, 39(1):3–20. 53

Aytar, Y., Pfaff, T., Budden, D., Paine, T., Wang, Z., and de Freitas, N. (2018). Playing hard exploration games by watching YouTube. *Advances in Neural Information Processing Systems*, 31. 55, 68

Badia, A. P., Piot, B., Kapturowski, S., Sprechmann, P., Vitvitskyi, A., Guo, Z. D., and Blundell, C. (2020). Agent57: Outperforming the Atari human benchmark. In *International Conference on Machine Learning*, pages 507–517. PMLR. 69

Badia, A. P., Sprechmann, P., Vitvitskyi, A., Guo, D., Piot, B., Kapturowski, S., Tieleman, O., Arjovsky, M., Pritzel, A., Bolt, A., et al. (2019). Never give up: Learning directed exploration strategies. In *International Conference on Learning Representations*. 69, 70

Baumli, K., Warde-Farley, D., Hansen, S., and Mnih, V. (2021). Relative variational intrinsic control. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 6732–6740. 51

Bellemare, M., Srinivasan, S., Ostrovski, G., Schaul, T., Saxton, D., and Munos, R. (2016). Unifying count-based exploration and intrinsic motivation. *Advances in neural information processing systems*, 29:1471–1479. 31, 45, 46, 55, 56, 67, 68

Bellemare, M. G., Dabney, W., and Munos, R. (2017). A distributional perspective on reinforcement learning. In *International Conference on Machine Learning*, pages 449–458. PMLR. 68

Bellemare, M. G., Naddaf, Y., Veness, J., and Bowling, M. (2013). The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279. 33

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. (2020). Language models are few-shot learners. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M. F., and Lin, H., editors, *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc. 8

Burda, Y., Edwards, H., Pathak, D., Storkey, A., Darrell, T., and Efros, A. A. (2018a). Large-scale study of curiosity-driven learning. In *International Conference on Learning Representations*. 2

Burda, Y., Edwards, H., Pathak, D., Storkey, A., Darrell, T., and Efros, A. A. (2018b). Large-scale study of curiosity-driven learning. In *International Conference on Learning Representations*. 56, 59, 66, 67, 68

Burda, Y., Edwards, H., Storkey, A., and Klimov, O. (2018c). Exploration by random network distillation. In *International Conference on Learning Representations*. 2, 73

Chen, M., Radford, A., Child, R., Wu, J., Jun, H., Luan, D., and Sutskever, I. (2020). Generative pretraining from pixels. In *International Conference on Machine Learning*, pages 1691–1703. PMLR. 8

Chen, R. Y., Schulman, J., Abbeel, P., and Sidor, S. (2017). UCB and infogain exploration via $q$-ensembles. *arXiv:1706.01502*. 46, 67

Cheplygina, V., Tax, D. M., and Loog, M. (2015). On classification with bags, groups and sets. *Pattern Recognition Letters*, 59:11 – 17. 21

Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning phrase representations using RNN encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*. 90

Choi, S. and Chung, S.-Y. (2019). Novelty detection via blurring. In *International Conference on Learning Representations*. 70

Chopra, S., Hadsell, R., and LeCun, Y. (2005). Learning a similarity metric discriminatively, with application to face verification. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, pages 539–546 Vol. 1. 21

Christiano, P., Leike, J., Brown, T. B., Martic, M., Legg, S., and Amodei, D. (2017). Deep reinforcement learning from human preferences. *arXiv preprint arXiv:1706.03741*. 74

Ciosek, K., Fortuin, V., Tomioka, R., Hofmann, K., and Turner, R. (2019). Conservative uncertainty estimation by fitting prior networks. In *International Conference on Learning Representations*. 70

Costikyan, G. (2013). *Uncertainty in games*. MIT Press. 40, 47

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee. 6

Denil, M., Agrawal, P., Kulkarni, T. D., Erez, T., Battaglia, P., and de Freitas, N. (2016). Learning to perform physics experiments via deep reinforcement learning. *arXiv preprint arXiv:1611.01843*. 56

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2019). Bert: Pre-training of deep bidirectional transformers for language understanding. In *NAACL-HLT (1)*. 9

Dieleman, S., Schlüter, J., Raffel, C., Olson, E., Sønderby, S., Nouri, D., Maturana, D., Thoma, M., Battenberg, E., Kelly, J., et al. (2015). Lasagne: First release. *Zenodo: Geneva, Switzerland*. 22

Duan, Y., Schulman, J., Chen, X., Bartlett, P. L., Sutskever, I., and Abbeel, P. (2016). RL$^2$: Fast reinforcement learning via slow reinforcement learning. *arXiv preprint arXiv:1611.02779*. 8

Ecoffet, A., Huizinga, J., Lehman, J., Stanley, K. O., and Clune, J. (2021). First return, then explore. *Nature*, 590(7847):580–586. 70

Edwards, H. and Storkey, A. (2016). Towards a neural statistician. *International Conference on Learning Representations*. 1

Espeholt, L., Soyer, H., Munos, R., Simonyan, K., Mnih, V., Ward, T., Doron, Y., Firoiu, V., Harley, T., Dunning, I., et al. (2018). Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. In *International Conference on Machine Learning*, pages 1407–1416. PMLR. 53, 68

Evgeniou, T. and Pontil, M. (2004). Regularized multi–task learning. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 109–117. ACM. 20

Eysenbach, B., Gupta, A., Ibarz, J., and Levine, S. (2018). Diversity is all you need: Learning skills without a reward function. In *International Conference on Learning Representations*. 46, 51, 67

Finn, C., Abbeel, P., and Levine, S. (2017). Model-agnostic meta-learning for fast adaptation of deep networks. In *International Conference on Machine Learning*, pages 1126–1135. PMLR. 8

Fortunato, M., Azar, M. G., Piot, B., Menick, J., Hessel, M., Osband, I., Graves, A., Mnih, V., Munos, R., Hassabis, D., et al. (2018). Noisy networks for exploration. In *International Conference on Learning Representations*. 46, 67

Fowl, L., Goldblum, M., Gupta, A., Sharaf, A., and Goldstein, T. (2020). Random network distillation as a diversity metric for both image and text generation. *arXiv preprint arXiv:2010.06715*. 70

Fox, L., Choshen, L., and Loewenstein, Y. (2018). Dora the explorer: Directed outreaching reinforcement action-selection. *International Conference on Learning Representations*. 56, 67

Fu, J., Co-Reyes, J. D., and Levine, S. (2017). EX2: Exploration with exemplar models for deep reinforcement learning. *NIPS*. 46, 67

Fukumizu, K., Song, L., and Gretton, A. (2013). Kernel Bayes' rule: Bayesian inference with positive definite kernels. *The Journal of Machine Learning Research*, 14(1):3753–3783. 22

Garmulewicz, M., Michalewski, H., and Miłoś, P. (2018). Expert-augmented actor-critic for ViZDoom and Montezuma's Revenge. *arXiv preprint arXiv:1809.03447*. 55, 63, 68

Garnelo, M., Rosenbaum, D., Maddison, C., Ramalho, T., Saxton, D., Shanahan, M., Teh, Y. W., Rezende, D., and Eslami, S. A. (2018a). Conditional neural processes. In *International Conference on Machine Learning*, pages 1704–1713. PMLR. 30

Garnelo, M., Schwarz, J., Rosenbaum, D., Viola, F., Rezende, D. J., Eslami, S., and Teh, Y. W. (2018b). Neural processes. *arXiv preprint arXiv:1807.01622*. 30

Gartner, T., Flach, P. A., Kowalczyk, A., and Smola, A. J. (2002). Multi-instance kernels. In *In Proc. 19th International Conf. on Machine Learning*, pages 179–186. Morgan Kaufmann. 21

Germain, M., Gregor, K., Murray, I., and Larochelle, H. (2015). Made: Masked autoencoder for distribution estimation. In *International Conference on Machine Learning*, pages 881–889. PMLR. 8

Gregor, K., Rezende, D. J., and Wierstra, D. (2017). Variational intrinsic control. *ICLR Workshop*. 46, 51, 67

Grill, J.-B., Strub, F., Altché, F., Tallec, C., Richemond, P., Buchatskaya, E., Doersch, C., Pires, B., Guo, Z., Azar, M., et al. (2020). Bootstrap your own latent: A new approach to self-supervised learning. In *Neural Information Processing Systems*. 9

Guo, Z. D., Azar, M. G., Saade, A., Thakoor, S., Piot, B., Pires, B. A., Valko, M., Mesnard, T., Lattimore, T., and Munos, R. (2021). Geometric entropic exploration. *arXiv preprint arXiv:2101.02055*. 69

Haber, N., Mrowca, D., Wang, S., Fei-Fei, L., and Yamins, D. L. (2018). Learning to play with intrinsically-motivated, self-aware agents. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pages 8398–8409. 56

Hansen, S., Dabney, W., Barreto, A., Warde-Farley, D., Van de Wiele, T., and Mnih, V. (2019). Fast task inference with variational intrinsic successor features. In *International Conference on Learning Representations*. 51

He, K., Fan, H., Wu, Y., Xie, S., and Girshick, R. (2020). Momentum contrast for unsupervised visual representation learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9729–9738. 9

Hernandez, D., Kaplan, J., Henighan, T., and McCandlish, S. (2021). Scaling laws for transfer. *arXiv preprint arXiv:2102.01293*. 73

Hessel, M., Modayil, J., Van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M., and Silver, D. (2018). Rainbow: Combining improvements in deep reinforcement learning. In *Thirty-second AAAI conference on artificial intelligence*. 68

Hewitt, L. B., Nye, M. I., Gane, A., Jaakkola, T., and Tenenbaum, J. B. (2018). The variational homoencoder: Learning to learn high capacity generative models from few examples. *arXiv preprint arXiv:1807.08919*. 29

Horgan, D., Quan, J., Budden, D., Barth-Maron, G., Hessel, M., van Hasselt, H., and Silver, D. (2018). Distributed prioritized experience replay. In *International Conference on Learning Representations*. 53, 67, 68

Hospedales, T. M., Antoniou, A., Micaelli, P., and Storkey, A. J. (2021). Meta-learning in neural networks: A survey. 8

Houthooft, R., Chen, X., Duan, Y., Schulman, J., De Turck, F., and Abbeel, P. (2016). Vime: Variational information maximizing exploration. In *NIPS*. 31, 45, 67

Hunicke, R., LeBlanc, M., and Zubek, R. (2004). Mda: A formal approach to game design and game research. In *AAAI Workshop on Challenges in Game AI*. 40, 47

Ioffe, S. and Szegedy, C. (2015a). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of The 32nd International Conference on Machine Learning*, pages 448–456. 22

Ioffe, S. and Szegedy, C. (2015b). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*. 37

Jarrett, K., Kavukcuoglu, K., LeCun, Y., et al. (2009). What is the best multi-stage architecture for object recognition? In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 2146–2153. IEEE. 47, 68

Jiang, B., Wu, T.-y., Zheng, C., and Wong, W. H. (2017). Learning summary statistic for approximate bayesian computation via deep neural network. *Statistica Sinica*, pages 1595–1618. 22

Johnson, M. J., Duvenaud, D., Wiltschko, A. B., Datta, S. R., and Adams, R. P. (2016). Structured vaes: Composing probabilistic graphical models and variational autoencoders. *arXiv preprint arXiv:1603.06277*. 20

Juliani, A., Berges, V.-P., Vckay, E., Gao, Y., Henry, H., Mattar, M., and Lange, D. (2018). Unity: A general platform for intelligent agents. *arXiv:1809.02627*. 33

Kaae Sønderby, C., Raiko, T., Maaløe, L., Kaae Sønderby, S., and Winther, O. (2016). How to train deep variational autoencoders and probabilistic ladder networks. *arXiv preprint arXiv:1602.02282*. 19

Kim, H., Mnih, A., Schwarz, J., Garnelo, M., Eslami, A., Rosenbaum, D., Vinyals, O., and Teh, Y. W. (2018). Attentive neural processes. In *International Conference on Learning Representations*. 30

Kingma, D. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*. 22

Kingma, D. and Ba, J. (2015). Adam: A method for stochastic optimization. *ICLR*. 90

Kingma, D. P., Mohamed, S., Jimenez Rezende, D., and Welling, M. (2014). Semi-supervised learning with deep generative models. *Advances in neural information processing systems*, 27. 1

Kingma, D. P. and Welling, M. (2013a). Auto-encoding variational Bayes. In *Proceedings of the 2nd International Conference on Learning Representations (ICLR)*, number 2014. 1, 9, 16, 17

Kingma, D. P. and Welling, M. (2013b). Auto-encoding variational Bayes. *arXiv preprint arXiv:1312.6114*. 33, 35

Koch, G. (2015). Siamese neural networks for one-shot image recognition. *Doctoral dissertation, University of Toronto*. 21, 26, 27

Kulkarni, T. D., Narasimhan, K., Saeedi, A., and Tenenbaum, J. (2016). Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. In *Advances in neural information processing systems*, pages 3675–3683. 68

Lake, B. M., Salakhutdinov, R., and Tenenbaum, J. B. (2015). Human-level concept learning through probabilistic program induction. *Science*, 350(6266):1332–1338. 21, 25

Lamb, A., Dumoulin, V., and Courville, A. (2016). Discriminative regularization for generative models. *arXiv preprint arXiv:1602.03220*. 26

Larsson, G., Maire, M., and Shakhnarovich, G. (2016). Learning representations for automatic colorization. In *European conference on computer vision*, pages 577–593. Springer. 9

Lawrence, N. D. and Platt, J. C. (2004). Learning to learn with the informative vector machine. In *Proceedings of the twenty-first international conference on Machine learning*, page 65. ACM. 20

Lazzaro, N. (2004). Why we play games: Four keys to more emotion in player experiences. In *Proceedings of GDC*. 40, 47

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324. 24

Lehman, J. and Stanley, K. O. (2008). Exploiting open-endedness to solve problems through the search for novelty. In *ALIFE*. 46

Lehman, J. and Stanley, K. O. (2011). Abandoning objectives: Evolution through the search for novelty alone. *Evolutionary computation*. 46

Lopes, M., Lang, T., Toussaint, M., and Oudeyer, P.-Y. (2012). Exploration in model-based reinforcement learning by empirically estimating learning progress. In *NIPS*. 31, 45, 58, 67

Maaløe, L., Sønderby, C. K., Sønderby, S. K., and Winther, O. (2016). Auxiliary deep generative models. In *International conference on machine learning*, pages 1445–1453. PMLR. 19

Machado, M. C., Bellemare, M. G., and Bowling, M. (2020). Count-based exploration with the successor representation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 5125–5133. 67, 68

Machado, M. C., Bellemare, M. G., Talvitie, E., Veness, J., Hausknecht, M., and Bowling, M. (2018). Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. *Journal of Artificial Intelligence Research*, 61:523–562. 68, 87

Mahajan, D. K., Girshick, R. B., Ramanathan, V., He, K., Paluri, M., Li, Y., Bharambe, A., and van der Maaten, L. (2018). Exploring the limits of weakly supervised pretraining. In *ECCV*. 1

Miao, Y., Yu, L., and Blunsom, P. (2016). Neural variational inference for text processing. In *International conference on machine learning*, pages 1727–1736. PMLR. 20

Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*. 9

Mikolov, T., Karafiát, M., Burget, L., Černockỳ, J., and Khudanpur, S. (2010). Recurrent neural network based language model. In *Eleventh annual conference of the international speech communication association*. 8

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., and Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. In *ICML*. 68

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing Atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*. 55

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015a). Human-level control through deep reinforcement learning. *Nature*. 31

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. (2015b). Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533. 67, 90

Mohamed, S. and Rezende, D. J. (2015). Variational information maximisation for intrinsically motivated reinforcement learning. In *NIPS*. 31

Muandet, K., Fukumizu, K., Dinuzzo, F., and Schölkopf, B. (2012). Learning from distributions via support measure machines. In Bartlett, P., Pereira, F., Burges, C., Bottou, L., and Weinberger, K., editors, *Advances in Neural Information Processing Systems 25*, pages 10–18. 21

Noroozi, M. and Favaro, P. (2016). Unsupervised learning of visual representations by solving jigsaw puzzles. In *European conference on computer vision*, pages 69–84. Springer. 1, 9

Oh, J., Guo, Y., Singh, S., and Lee, H. (2018). Self-imitation learning. In *International Conference on Machine Learning*, pages 3878–3887. PMLR. 68

Oord, A. v. d., Li, Y., and Vinyals, O. (2018). Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*. 9

OpenAI (2018). OpenAI Five. `https://blog.openai.com/openai-five/`. 53

Osband, I., Aslanides, J., and Cassirer, A. (2018). Randomized prior functions for deep reinforcement learning. *Advances in Neural Information Processing Systems*, 31. 58, 67, 68, 70

Osband, I., Blundell, C., Pritzel, A., and Van Roy, B. (2016). Deep exploration via bootstrapped DQN. In *NIPS*. 46, 67

Ostrovski, G., Bellemare, M. G., Oord, A. v. d., and Munos, R. (2018a). Count-based exploration with neural density models. *International Conference for Machine Learning*. 31, 46, 67

Ostrovski, G., Bellemare, M. G., Oord, A. v. d., and Munos, R. (2018b). Count-based exploration with neural density models. *International Conference for Machine Learning*. 56, 68

Oudeyer, P.-Y. (2018). Computational theories of curiosity-driven learning. *arXiv preprint arXiv:1802.10546*. 46

Oudeyer, P.-Y. and Kaplan, F. (2009). What is intrinsic motivation? A typology of computational approaches. *Frontiers in neurorobotics*. 31, 46

Oudeyer, P.-Y., Kaplan, F., and Hafner, V. V. (2007). Intrinsic motivation systems for autonomous mental development. *Evolutionary Computation*. 58, 67

O'Donoghue, B., Osband, I., Munos, R., and Mnih, V. (2018). The uncertainty bellman equation and exploration. In *International Conference on Machine Learning*, pages 3836–3845. 68

Pan, S. J. and Yang, Q. (2010). A survey on transfer learning. *Knowledge and Data Engineering, IEEE Transactions on*, 22(10):1345–1359. 16, 20

Pathak, D., Agrawal, P., Efros, A. A., and Darrell, T. (2017). Curiosity-driven exploration by self-supervised prediction. In *ICML*. 31, 32, 33, 34, 36, 40, 45, 46, 52, 56, 66, 67

Pathak, D., Krahenbuhl, P., Donahue, J., Darrell, T., and Efros, A. A. (2016). Context encoders: Feature learning by inpainting. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2536–2544. 8

Pathak, D., Mahmoudieh, P., Luo, G., Agrawal, P., Chen, D., Shentu, Y., Shelhamer, E., Malik, J., Efros, A. A., and Darrell, T. (2018). Zero-shot visual imitation. In *ICLR*. 32

Plappert, M., Houthooft, R., Dhariwal, P., Sidor, S., Chen, R. Y., Chen, X., Asfour, T., Abbeel, P., and Andrychowicz, M. (2018). Parameter space noise for exploration. In *International Conference on Learning Representations*. 46, 67

Póczos, B., Xiong, L., Sutherland, D. J., and Schneider, J. (2012). Support distribution machines. *Technical Report*. 21

Pohlen, T., Piot, B., Hester, T., Azar, M. G., Horgan, D., Budden, D., Barth-Maron, G., van Hasselt, H., Quan, J., Večerík, M., et al. (2018). Observe and look further: Achieving consistent performance on Atari. *arXiv preprint arXiv:1805.11593*. 55, 63, 68

Pong, V., Gu, S., Dalal, M., and Levine, S. (2018). Temporal difference models: Model-free deep rl for model-based control. In *International Conference on Learning Representations*. 68

Poupart, P., Vlassis, N., Hoey, J., and Regan, K. (2006). An analytic solution to discrete bayesian reinforcement learning. In *ICML*. 31

Radford, A., Narasimhan, K., Salimans, T., and Sutskever, I. (2018). Improving language understanding by generative pre-training. 8

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I., et al. (2019). Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9. 8

Rahimi, A. and Recht, B. (2008). Random features for large-scale kernel machines. In *Advances in neural information processing systems*, pages 1177–1184. 68

Ranganath, R., Gerrish, S., and Blei, D. M. (2014). Black box variational inference. In *AISTATS*, pages 814–822. 20

Rezende, D., Danihelka, I., Gregor, K., Wierstra, D., et al. (2016). One-shot generalization in deep generative models. In *International Conference on Machine Learning*, pages 1521–1529. PMLR. 21

Rezende, D. J., Mohamed, S., and Wierstra, D. (2014a). Stochastic backpropagation and approximate inference in deep generative models. *arXiv preprint arXiv:1401.4082*. 9, 35

Rezende, D. J., Mohamed, S., and Wierstra, D. (2014b). Stochastic backpropagation and approximate inference in deep generative models. In *Proceedings of The 31st International Conference on Machine Learning*, pages 1278–1286. 17

Ryan, Richard; Deci, E. L. (2000). Intrinsic and extrinsic motivations: Classic definitions and new directions. *Contemporary Educational Psychology*. 31

Salakhutdinov, R., Tenenbaum, J. B., and Torralba, A. (2012). One-shot learning with a hierarchical nonparametric bayesian model. In *ICML Unsupervised and Transfer Learning*, pages 195–206. 21

Salimans, T. and Chen, R. (2018). Learning Montezuma's Revenge from a single demonstration. *https://blog.openai.com/learning-montezumas-revenge-from-a-single-demonstration/*. 55, 63, 68

Santoro, A., Bartunov, S., Botvinick, M., Wierstra, D., and Lillicrap, T. (2016a). Meta-learning with memory-augmented neural networks. In *International conference on machine learning*, pages 1842–1850. PMLR. 8

Santoro, A., Bartunov, S., Botvinick, M., Wierstra, D., and Lillicrap, T. (2016b). One-shot learning with memory-augmented neural networks. *arXiv preprint arXiv:1605.06065*. 21, 26, 27

Saul, L. K. and Jordan, M. I. (1996). Exploiting tractable substructures in intractable networks. In *Advances in Neural Processing Systems 8*. 17

Savinov, N., Raichuk, A., Marinier, R., Vincent, D., Pollefeys, M., Lillicrap, T., and Gelly, S. (2018). Episodic curiosity through reachability. *arXiv preprint arXiv:1810.02274*. 70

Saxe, A. M., Koh, P. W., Chen, Z., Bhand, M., Suresh, B., and Ng, A. Y. (2011). On random weights and unsupervised feature learning. In *ICML*, pages 1089–1096. 47, 68

Schmidhuber, J. (1991a). Curious model-building control systems. In *Neural Networks, 1991. 1991 IEEE International Joint Conference on*, pages 1458–1463. IEEE. 45, 58, 67

Schmidhuber, J. (1991b). A possibility for implementing curiosity and boredom in model-building neural controllers. In *From animals to animats: Proceedings of the first international conference on simulation of adaptive behavior*. 31

Schmidhuber, J. (1991c). A possibility for implementing curiosity and boredom in model-building neural controllers. 45, 56, 66, 67

Schmidhuber, J. (2010). Formal theory of creativity, fun, and intrinsic motivation (1990–2010). *IEEE Transactions on Autonomous Mental Development*. 46

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*. 13, 33, 36, 55, 87

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., and Hassabis, D. (2016). Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489. 53

Singh, S. P., Barto, A. G., and Chentanez, N. (2005). Intrinsically motivated reinforcement learning. In *NIPS*. 31

Smith, L. and Gasser, M. (2005). The development of embodied cognition: Six lessons from babies. *Artificial life*. 31

Snell, J., Swersky, K., and Zemel, R. (2017). Prototypical networks for few-shot learning. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 4080–4090. 8

Stadie, B. C., Levine, S., and Abbeel, P. (2015). Incentivizing exploration in reinforcement learning with deep predictive models. *NIPS Workshop*. 33, 45, 46, 56, 66, 67

Stanley, K. O. and Lehman, J. (2015). *Why greatness cannot be planned: The myth of the objective*. Springer. 46

Stanton, C. and Clune, J. (2018). Deep curiosity search: Intra-life exploration improves performance on challenging deep reinforcement learning problems. *arXiv preprint arXiv:1806.00553*. 55, 68

Still, S. and Precup, D. (2012). An information-theoretic approach to curiosity-driven reinforcement learning. *Theory in Biosciences*. 45, 67

Strehl, A. L. and Littman, M. L. (2008). An analysis of model-based interval estimation for markov decision processes. *Journal of Computer and System Sciences*, 74(8):1309–1331. 67

Sukhbaatar, S., Kostrikov, I., Szlam, A., and Fergus, R. (2018). Intrinsic motivation and automatic curricula via asymmetric self-play. In *ICLR*. 46, 67

Sutton, R. S. and Barto, A. G. (1998). *Reinforcement learning: An introduction*. MIT press Cambridge. 37

Tang, H., Houthooft, R., Foote, D., Stooke, A., Chen, O. X., Duan, Y., Schulman, J., DeTurck, F., and Abbeel, P. (2017a). # Exploration: A study of count-based exploration for deep reinforcement learning. In *NIPS*. 55, 67, 68

Tang, H., Houthooft, R., Foote, D., Stooke, A., Chen, X., Duan, Y., Schulman, J., De Turck, F., and Abbeel, P. (2017b). #Exploration: A study of count-based exploration for deep reinforcement learning. *Advances in Neural Information Processing Systems*. 46

Theano Development Team (2016). Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*, abs/1605.02688. 22

Theis, L., van den Oord, A., and Bethge, M. (2016). A note on the evaluation of generative models. In *International Conference on Learning Representations (ICLR)*. 24

Uria, B., Côté, M.-A., Gregor, K., Murray, I., and Larochelle, H. (2016). Neural autoregressive distribution estimation. *The Journal of Machine Learning Research*, 17(1):7184–7220. 8

Van Oord, A., Kalchbrenner, N., and Kavukcuoglu, K. (2016). Pixel recurrent neural networks. In *International Conference on Machine Learning*, pages 1747–1756. PMLR. 8

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008. 8

Vincent, P., Larochelle, H., Bengio, Y., and Manzagol, P.-A. (2008). Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pages 1096–1103. 8

Vinyals, O., Bengio, S., and Kudlur, M. (2016a). Order matters: sequence to sequence for sets. In *International Conference on Learning Representations (ICLR)*. 21

Vinyals, O., Blundell, C., Lillicrap, T., Wierstra, D., et al. (2016b). Matching networks for one shot learning. *Advances in neural information processing systems*, 29:3630–3638. 8, 21, 26, 27

Wolf, L., Hassner, T., and Maoz, I. (2011). Face recognition in unconstrained videos with matched background similarity. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 529–534. IEEE. 26

Wouters, P., Van Oostendorp, H., Boonekamp, R., and Van der Spek, E. (2011). The role of game discourse analysis and curiosity in creating engaging and effective serious games by implementing a back story and foreshadowing. *Interacting with Computers*. 40, 47

Yang, Z., Moczulski, M., Denil, M., de Freitas, N., Smola, A., Song, L., and Wang, Z. (2015). Deep fried convnets. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1476–1483. 47, 68

Zhai, X., Kolesnikov, A., Houlsby, N., and Beyer, L. (2021). Scaling vision transformers. *arXiv preprint arXiv:2106.04560*. 6

Zoph, B. and Le, Q. V. (2016). Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*. 53