Count-Based Exploration with Neural Density Models

Georg Ostrovski [1] **Marc G. Bellemare** [1] **Aäron van den Oord** [1] **Rémi Munos** [1]

Abstract

Bellemare et al. (2016) introduced the notion of a pseudo-count, derived from a density model, to generalize count-based exploration to non-tabular reinforcement learning. This pseudo-count was used to generate an exploration bonus for a DQN agent and combined with a mixed Monte Carlo update was sufficient to achieve state of the art on the Atari 2600 game Montezuma's Revenge. We consider two questions left open by their work: First, how important is the quality of the density model for exploration? Second, what role does the Monte Carlo update play in exploration? We answer the first question by demonstrating the use of PixelCNN, an advanced neural density model for images, to supply a pseudo-count. In particular, we examine the intrinsic difficulties in adapting Bellemare et al.'s approach when assumptions about the model are violated. The result is a more practical and general algorithm requiring no special apparatus. We combine PixelCNN pseudo-counts with different agent architectures to dramatically improve the state of the art on several hard Atari games. One surprising finding is that the mixed Monte Carlo update is a powerful facilitator of exploration in the sparsest of settings, including Montezuma's Revenge.

1. Introduction

Exploration is the process by which an agent learns about its environment. In the reinforcement learning framework, this involves reducing the agent's uncertainty about the environment's transition dynamics and attainable rewards. From a theoretical perspective, exploration is now well-understood (e.g. Strehl & Littman, 2008; Jaksch et al., 2010; Osband et al., 2016), and Bayesian methods have been successfully demonstrated in a number of settings (Deisenroth & Rasmussen, 2011; Guez et al., 2012). On the other hand, practical algorithms for the general case remain scarce; fully Bayesian approaches are usually intractable in large state spaces, and the count-based method typical of theoretical results is not applicable in the presence of value function approximation.

Recently, Bellemare et al. (2016) proposed the notion of *pseudo-count* as a reasonable generalization of the tabular setting considered in the theory literature. The pseudo-count is defined in terms of a density model ρ trained on the sequence of states experienced by an agent:

$$\hat{\mathrm{N}}(x) = \rho(x)\hat{n}(x),$$

where $\hat{n}(x)$ can be thought of as a total pseudo-count computed from the model's *recoding probability* $\rho'(x)$, the probability of x computed immediately after training on x. As a practical application the authors used the pseudo-counts derived from the simple CTS density model (Bellemare et al., 2014) to incentivize exploration in Atari 2600 agents. One of the main outcomes of their work was substantial empirical progress on the infamously hard game MONTEZUMA'S REVENGE.

Their method critically hinged on several assumptions regarding the density model: 1) the model should be *learning-positive*, i.e. the probability assigned to a state x should increase with training; 2) it should be trained online, using each sample exactly once; and 3) the effective model step-size should decay at a rate of n^{-1}. Part of their empirical success also relied on a mixed Monte Carlo/Q-Learning update rule, which permitted fast propagation of the exploration bonuses.

In this paper, we set out to answer several research questions related to these modelling choices and assumptions:

1. To what extent does a better density model give rise to better exploration?
2. Can the above modelling assumptions be relaxed without sacrificing exploration performance?
3. What role does the mixed Monte Carlo update play in successfully incentivizing exploration?

[1] DeepMind, London, UK. Correspondence to: Georg Ostrovski <ostrovski@google.com>.

Proceedings of the 34^{th} International Conference on Machine Learning, Sydney, Australia, PMLR 70, 2017. Copyright 2017 by the author(s).

In particular, we explore the use of PixelCNN (van den Oord et al., 2016b;a), a state-of-the-art neural density model. We examine the challenges posed by this approach:

Model choice. Performing two evaluations and one model update *at each agent step* (to compute $\rho(x)$ and $\rho'(x)$) can be prohibitively expensive. This requires the design of a simplified – yet sufficiently expressive and accurate – PixelCNN architecture.

Model training. A CTS model can naturally be trained from sequentially presented, correlated data samples. Training a *neural* model in this online fashion requires more careful attention to the optimization procedure to prevent overfitting and catastrophic forgetting (French, 1999).

Model use. The theory of pseudo-counts requires the density model's rate of learning to decay over time. Optimization of a neural model, however, imposes constraints on the step-size regime which cannot be violated without deteriorating effectiveness and stability of training.

The concept of intrinsic motivation has made a recent resurgence in reinforcement learning research, in great part due to a dissatisfaction with ϵ-greedy and Boltzmann policies. Of note, Tang et al. (2016) maintain an approximate count by means of hash tables over features, which in the pseudo-count framework corresponds to a hash-based density model. Houthooft et al. (2016) used a second-order Taylor approximation of the prediction gain to drive exploration in continuous control. As research moves towards ever more complex environments, we expect the trend towards more intrinsically motivated solutions to continue.

2. Background

2.1. Pseudo-Count and Prediction Gain

Here we briefly introduce notation and results, referring the reader to (Bellemare et al., 2016) for technical details.

Let ρ be a density model on a finite space $\mathcal{X}$, and $\rho_n(x)$ the probability assigned by the model to x after being trained on a sequence of states $x_1, \ldots, x_n$. Assume $\rho_n(x) > 0$ for all x, n. The *recoding probability* $\rho'_n(x)$ is then the probability the model would assign to x if it were trained on that same x one more time. We call ρ *learning-positive* if $\rho'_n(x) \geq \rho_n(x)$ for all $x_1, \ldots, x_n, x \in \mathcal{X}$. The *prediction gain* (PG) of ρ is

$$\mathrm{PG}_n(x) = \log \rho'_n(x) - \log \rho_n(x). \tag{1}$$

A learning-positive ρ implies $\mathrm{PG}_n(x) \geq 0$ for all $x \in \mathcal{X}$. For learning-positive ρ, we define the *pseudo-count* as

$$\hat{\mathrm{N}}_n(x) = \frac{\rho_n(x)(1 - \rho'_n(x))}{\rho'_n(x) - \rho_n(x)},$$

derived from postulating that a single observation of $x \in \mathcal{X}$ should lead to a unit increase in pseudo-count:

$$\rho_n(x) = \frac{\hat{\mathrm{N}}_n(x)}{\hat{n}}, \quad \rho'_n(x) = \frac{\hat{\mathrm{N}}_n(x) + 1}{\hat{n} + 1},$$

where $\hat{n}$ is the *pseudo-count total*. The pseudo-count generalizes the usual state visitation count function $\mathrm{N}_n(x)$. Under certain assumptions on ρ_n, pseudo-counts grow approximately linearly with real counts. Crucially, the pseudo-count can be approximated using the prediction gain of the density model:

$$\hat{\mathrm{N}}_n(x) \approx \left(e^{\mathrm{PG}_n(x)} - 1\right)^{-1}.$$

Its main use is to define an *exploration bonus*. We consider a reinforcement learning (RL) agent interacting with an environment that provides observations and extrinsic rewards (see Sutton & Barto, 1998, for a thorough exposition of the RL framework). To the reward at step n we add the bonus

$$r^+(x) := (\hat{\mathrm{N}}_n(x))^{-1/2},$$

which incentivizes the agent to try to re-experience surprising situations. Quantities related to prediction gain have been used for similar purposes in the intrinsic motivation literature (Lopes et al., 2012), where they measure an agent's *learning progress* (Oudeyer et al., 2007). Although the pseudo-count bonus is close to the prediction gain, it is asymptotically more conservative and supported by stronger theoretical guarantees.

2.2. Density Models for Images

The CTS density model (Bellemare et al., 2014) is based on the namesake algorithm, Context Tree Switching (Veness et al., 2012), a Bayesian variable-order Markov model. In its simplest form, the model takes as input a 2D image and assigns to it a probability according to the product of location-dependent L-shaped filters, where the prediction of each filter is given by a CTS algorithm trained on past images. In Bellemare et al. (2016), this model was applied to 3-bit greyscale, 42 × 42 downsampled Atari 2600 frames (Fig. 1). The CTS model presents advantages in terms of simplicity and performance but is limited in expressiveness, scalability, and data efficiency.

In recent years, neural generative models for images have achieved impressive successes in their ability to generate diverse images in various domains (Kingma & Welling, 2013; Rezende et al., 2014; Gregor et al., 2015; Goodfellow et al., 2014). In particular, van den Oord et al. (2016b;a) introduced *PixelCNN*, a fully convolutional neural network composed of residual blocks with multiplicative gating units, which models pixel probabilities conditional on previous pixels (in the usual top-left to bottom-right raster-scan order) by using masked convolution fil-

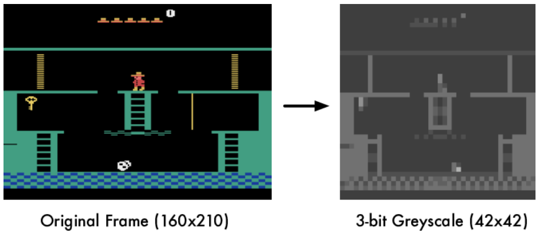

Figure 1. Atari frame preprocessing (Bellemare et al., 2016).

ters. This model achieved state-of-the-art modelling performance on standard datasets, paired with the computational efficiency of a convolutional feed-forward network.

2.3. Multi-Step RL Methods

A distinguishing feature of reinforcement learning is that the agent "learns on the basis of interim estimates" (Sutton, 1996). For example, the Q-Learning update rule is

$$\begin{aligned} Q(x,a) &\leftarrow Q(x,a) \\ &\quad + \alpha \underbrace{\left[r(x,a) + \gamma \max\nolimits_{a'} Q(x',a') - Q(x,a)\right]}_{\delta(x,a)}, \end{aligned}$$

linking the reward r and next-state value function $Q(x', a')$ to the current state value function $Q(x,a)$. This particular form is the stochastic update rule with step-size α and involves the *TD-error* δ. In the approximate reinforcement learning setting, such as when $Q(x,a)$ is represented by a neural network, this update is converted into a loss to be minimized, most commonly the squared loss $\delta^2(x,a)$.

It is well known that better performance, both in terms of learning efficiency and approximation error, is attained by multi-step methods (Sutton, 1996; Tsitsiklis & van Roy, 1997). These methods interpolate between one-step methods (Q-Learning) and the Monte-Carlo update

$$Q(x,a) \leftarrow Q(x,a) + \alpha \underbrace{\left[\sum_{t=0}^{\infty} \gamma^t r(x_t, a_t) - Q(x,a)\right]}_{\delta_{\text{MC}}(x,a)},$$

where $x_0, a_0, x_1, a_1, \ldots$ is a sample path through the environment beginning in (x,a). To achieve their success on the hardest Atari 2600 games, Bellemare et al. (2016) used the *mixed Monte-Carlo update* (MMC)

$$Q(x,a) \leftarrow Q(x,a) + \alpha\left[(1-\beta)\delta(x,a) + \beta\delta_{\text{MC}}(x,a)\right],$$

with $\beta \in [0,1]$. This choice was made for "computational and implementational simplicity", and is a particularly coarse multi-step method. A better multi-step method is the recent Retrace(λ) algorithm (Munos et al., 2016). Retrace(λ) uses a product of truncated importance sampling ratios $c_1, c_2, \ldots$ to replace δ with the error term

$$\delta_{\text{RETRACE}}(x,a) := \sum_{t=0}^{\infty} \gamma^t \left(\prod_{s=1}^{t} c_s\right) \delta(x_t, a_t),$$

effectively mixing in TD-errors from all future time steps. Munos et al. showed that Retrace(λ) is safe (does not diverge when trained on data from an arbitrary behaviour policy), and efficient (makes the most of multi-step returns).

3. Using PixelCNN for Exploration

As mentioned in the Introduction, the theory of using density models for exploration makes several assumptions that translate into concrete requirements for an implementation:

(a) The density model should be trained completely online, i.e. exactly once on each state experienced by the agent, in the given sequential order.

(b) The prediction gain (PG) should decay at a rate n^{-1} to ensure that pseudo-counts grow approximately linearly with real counts.

(c) The density model should be learning-positive.

Simultaneously, a partly competing set of requirements are posed by the practicalities of training a neural density model and using it as part of an RL agent:

(d) For stability, efficiency, and to avoid catastrophic forgetting in the context of a drifting data distribution, it is advantageous to train a neural model in mini-batches, drawn randomly from a diverse dataset.

(e) For effective training, a certain optimization regime (e.g. a fixed learning rate schedule) has to be followed.

(f) The density model must be computationally lightweight, to allow computing the PG (two model evaluations and one update) as part of every training step of an RL agent.

We investigate how to best resolve these tensions in the context of the Arcade Learning Environment (Bellemare et al., 2013), a suite of benchmark Atari 2600 games.

3.1. Designing a Suitable Density Model

Driven by (f) and aiming for an agent with computational performance comparable to DQN, we design a slim variant of the PixelCNN network. Its core is a stack of 2 gated residual blocks with 16 feature maps (compared to 15 residual blocks with 128 feature maps in vanilla PixelCNN). As was done with the CTS model, images are downsampled to 42 × 42 and quantized to 3-bit greyscale. See Appendix A for technical details.

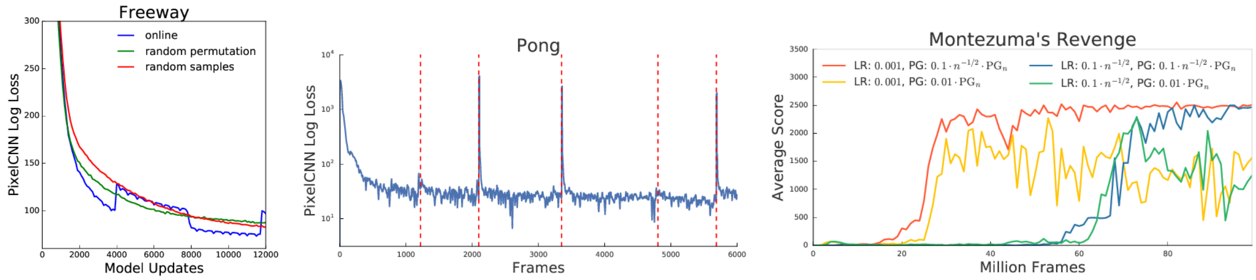

Figure 2. **Left:** PixelCNN log loss on FREEWAY, when trained *online*, on a *random permutation* (single use of each frame) or on *randomly drawn samples* (with replacement, potentially using same frame multiple times) from the state sequence. To simulate the effect of non-stationarity, the agent's policy changes every 4K updates. All training methods show qualitatively similar learning progress and stability. **Middle:** PixelCNN log loss over first 6K training frames on PONG. Vertical dashed lines indicate episode ends. The coinciding loss spikes are the density model's 'surprise' upon observing the distinctive green frame that sometimes occurs at the episode start. **Right:** DQN-PixelCNN training performance on MONTEZUMA'S REVENGE as we vary learning rate and PG decay schedules.

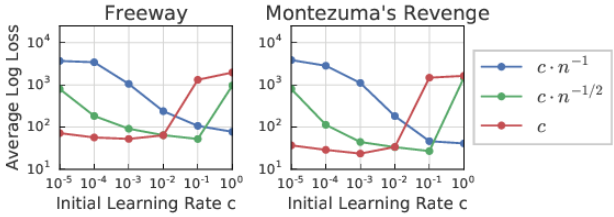

Figure 3. Model loss averaged over 10K frames, after 1M training frames, for constant, n^{-1}, and $n^{-1/2}$ learning rate schedules. The smallest loss is achieved by a constant learning rate of 10^{-3}.

3.2. Training the Density Model

Instead of using randomized mini-batches, we train the density model completely online on the sequence of experienced states. Empirically we found that with minor tuning of optimization hyper-parameters we could train the model as robustly on a temporally correlated sequence of states as on a sequence with randomized order (Fig. 2(left)).

Besides satisfying the theoretical requirement (a), completely online training of the density model has the advantage that $\rho'_n = \rho_{n+1}$, so that the model update performed for computing the PG need not be reverted[1].

Another more subtle reason for avoiding mini-batch updates of the density model (despite (d)) is a practical optimization issue. The (necessarily online) computation of the PG involves a model update and hence the use of an optimizer. Advanced optimizers used with deep neural networks, like the RMSProp optimizer (Tieleman & Hinton, 2012) used in this work, are stateful, tracking running averages of e.g. mean and variance of the model parameters. If the model is *additionally* trained from mini-batches, the two streams of updates may show different statistical characteristics (e.g. different gradient magnitudes), invalidating the assumptions underlying the optimization algorithm and leading to slower or unstable training.

To determine a suitable online learning rate schedule, we train the model on a sequence of 1M frames of experience of a random-policy agent. We compare the loss achieved by training procedures following constant or decaying learning rate schedules, see Fig. 3. The lowest final training loss is achieved by a constant learning rate of 0.001 or a decaying learning rate of $0.1 \cdot n^{-1/2}$. We settled our choice on the constant learning rate schedule as it showed greater robustness with respect to the choice of initial learning rate.

PixelCNN rapidly learns a sensible distribution over state space. Fig. 2(left) shows the model's loss decaying as it learns to exploit image regularities. Spikes in its loss function quickly start to correspond to visually meaningful events, such as the starts of episodes (Fig. 2(middle)). A video of early density model training is provided in http://youtu.be/T6iaa8Z4eyE.

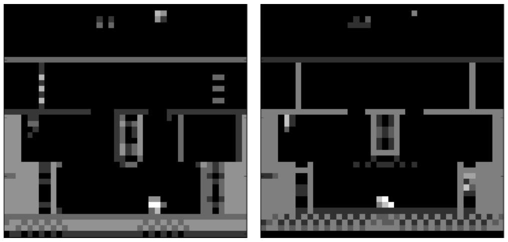

Figure 4. Samples after 25K steps. **Left**: CTS, **right**: PixelCNN.

3.3. Computing the Pseudo-Count

From the previous section we obtain a particular learning rate schedule that cannot be arbitrarily modified without deteriorating the model's training performance or stability. To achieve the required PG decay (b), we instead replace PG_n by $c_n \cdot \mathrm{PG}_n$ with a suitably decaying sequence c_n.

[1] The CTS model allows querying the PG cheaply, without incurring an actual update of model parameters.

In experiments comparing actual agent performance we empirically determined that in fact the constant learning rate 0.001, paired with a PG decay $c_n = c \cdot n^{-1/2}$, obtains the best exploration results on hard exploration games like MONTEZUMA'S REVENGE, see Fig. 2(right). We find the model to be robust across 1-2 orders of magnitude for the value of c, and informally determine $c = 0.1$ to be a sensible configuration for achieving good results on a broad range of Atari 2600 games (see also Section 7).

Regarding (c), it is hard to ensure learning-positiveness for a deep neural model, and a negative PG can occur whenever the optimizer 'overshoots' a local loss minimum. As a workaround, we threshold the PG value at 0. To summarize, the computed pseudo-count is

$$\hat{\mathrm{N}}_n(x) = \Big(\exp\Big(c \cdot n^{-1/2} \cdot (\mathrm{PG}_n(x))_+\Big) - 1\Big)^{-1}.$$

4. Exploration in Atari 2600 Games

Having described our pseudo-count friendly adaptation of PixelCNN, we now study its performance on Atari games. To this end we augment the environment reward with a pseudo-count exploration bonus, yielding the combined reward $r(x, a) + (\hat{\mathrm{N}}_n(x))^{-1/2}$. As usual for neural network-based agents, we ensure the total reward lies in $[-1, 1]$ by clipping larger values.

4.1. DQN with PixelCNN Exploration Bonus

Our first set of experiments provides the PixelCNN exploration bonus to a DQN agent (Mnih et al., 2015)[2]. At each agent step, the density model receives a single frame, with which it simultaneously updates its parameters and outputs the PG. We refer to this agent as *DQN-PixelCNN*.

The *DQN-CTS* agent we compare against is derived from the one in (Bellemare et al., 2016). For better comparability, it is trained in the same online fashion as DQN-PixelCNN, i.e. the PG is computed whenever we train the density model. By contrast, the original DQN-CTS queried the PG at the end of each episode.

Unless stated otherwise, we always use the mixed Monte Carlo update (MMC) for the intrinsically motivated agents[3], but regular Q-Learning for the baseline DQN.

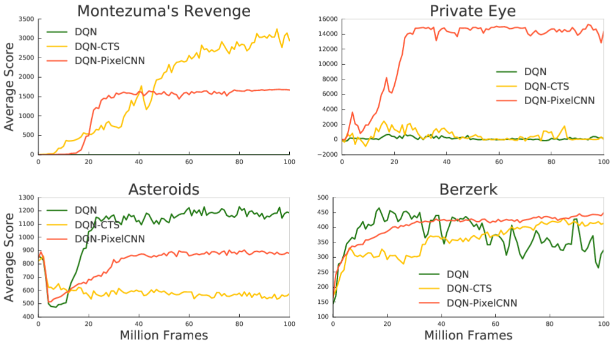

Figure 5. DQN, DQN-CTS and DQN-PixelCNN on hard exploration games (**top**) and easier ones (**bottom**).

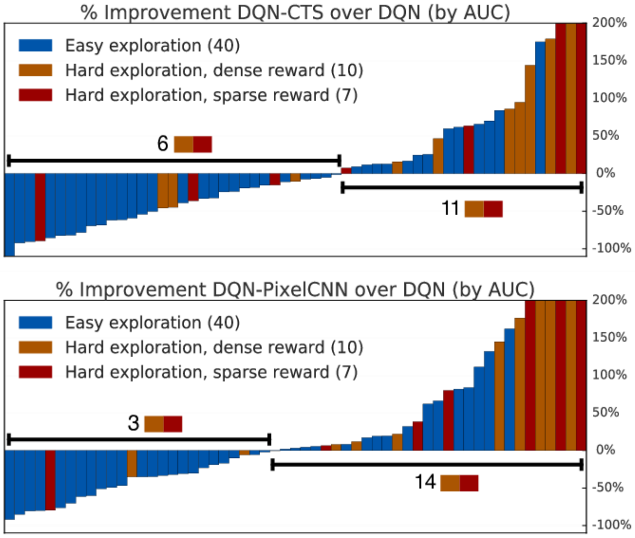

Figure 6. Improvements (in % of AUC) of DQN-PixelCNN and DQN-CTS over DQN in 57 Atari games. Annotations indicate the number of hard exploration games with positive (right) and negative (left) improvement, respectively.

Fig. 5 shows training curves of DQN compared to DQN-CTS and DQN-PixelCNN. On the famous MONTEZUMA'S REVENGE, both intrinsically motivated agents vastly outperform the baseline DQN. On other hard exploration games (PRIVATE EYE; or VENTURE, appendix Fig. 15), DQN-PixelCNN achieves state of the art results, substantially outperforming DQN and DQN-CTS. The other two games shown (ASTEROIDS, BERZERK) pose easier exploration problems, where the reward bonus should not provide large improvements and may have a negative effect by skewing the reward landscape. Here, DQN-PixelCNN behaves more gracefully and still outperforms DQN-CTS. We hypothesize this is due to a qualitative difference between the models, see Section 5.

Overall PixelCNN provides the DQN agent with a larger advantage than CTS, and often accelerates or stabilizes training even when not affecting peak performance. Out of

[2] Unlike Bellemare et al. we use regular Q-Learning instead of Double Q-Learning (van Hasselt et al., 2016), as our early experiments showed no significant advantage of DoubleDQN with the PixelCNN-based exploration reward.

[3] The use of MMC in a replay-based agent poses a minor complication, as the MC return is not available for replay until the end of an episode. For simplicity, in our implementation we disregard this detail and set the MC return to 0 for transitions from the most recent episode.

57 Atari games, DQN-PixelCNN outperforms DQN-CTS in 52 games by maximum achieved score, and 51 by AUC (methodology in Appendix B). See Fig. 6 for a high level comparison (appendix Fig. 15 for full training graphs). The greatest gains from using either exploration bonus are observed in games categorized as *hard exploration* games in the 'taxonomy of exploration' in (Bellemare et al., 2016, reproduced in Appendix D), specifically in the most challenging *sparse reward* games (e.g. MONTEZUMA'S REVENGE, PRIVATE EYE, VENTURE).

4.2. A Multi-Step RL Agent with PixelCNN

Empirical practitioners know that techniques beneficial for one agent architecture often can be detrimental for a different algorithm. To demonstrate the wide applicability of the PixelCNN exploration bonus, we also evaluate it with the more recent *Reactor* agent[4] (Gruslys et al., 2017). This replay-based actor-critic agent represents its policy and value function by a recurrent neural network and, crucially, uses the multi-step Retrace(λ) algorithm for policy evaluation, replacing the MMC we use in DQN-PixelCNN.

To reduce impact on computational efficiency of this agent, we sub-sample intrinsic rewards: we perform updates of the PixelCNN model and compute the reward bonus on (randomly chosen) 25% of all steps, leaving the agent's reward unchanged on other steps. We use the same PG decay schedule of $0.1n^{-1/2}$, with n the number of model updates.

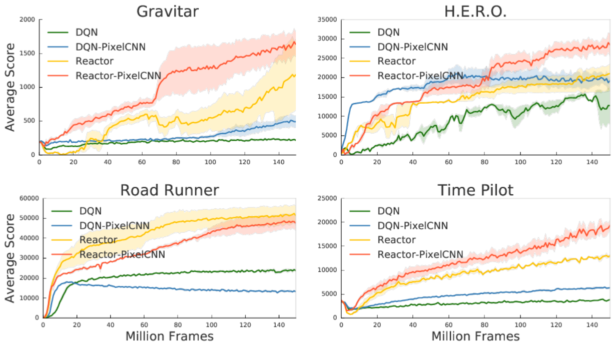

Figure 7. Reactor/Reactor-PixelCNN and DQN/DQN-PixelCNN training performance (averaged over 3 seeds).

Training curves for the Reactor/Reactor-PixelCNN agent compared to DQN/DQN-PixelCNN are shown in Fig. 7. The baseline Reactor agent is superior to the DQN agent, obtaining higher scores and learning faster in about 50 out of 57 games. It is further improved on a large fraction of games by the PixelCNN exploration reward, see Fig. 8 (full training graphs in appendix Fig. 16).

The effect of the exploration bonus is rather uniform, yielding improvements on a broad range of games. In particular, Reactor-PixelCNN enjoys better sample efficiency (in terms of area under the curve, AUC) than vanilla Reactor. We hypothesize that, like other policy gradient algorithms, Reactor generally suffers from weaker exploration than its value-based counterpart DQN. This aspect is much helped by the exploration bonus, boosting the agent's sample efficiency in many environments.

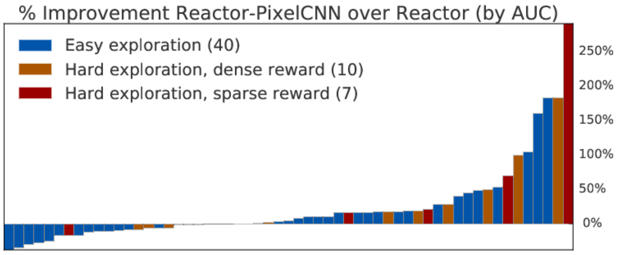

Figure 8. Improvements (in % of AUC) of Reactor-PixelCNN over Reactor in 57 Atari games.

However, on hard exploration games with sparse rewards, Reactor seems unable to make full use of the exploration bonus. We believe this is because, in very sparse settings, the propagation of reward information across long horizons becomes crucial. The MMC takes one extreme of this view, directly learning from the observed returns. The Retrace(λ) algorithm, on the other hand, has an effective horizon which depends on λ and, critically, the truncated importance sampling ratio. This ratio results in the discarding of trajectories which are off-policy, i.e. unlikely under the current policy. We hypothesize that the very goal of the Retrace(λ) algorithm to learn cautiously is what prevents it from taking full advantage of the exploration bonus!

5. Quality of the Density Model

PixelCNN can be expected to be more expressive and accurate than the less advanced CTS model, and indeed, samples generated after training are somewhat higher quality (Fig. 4). However, we are not using the generative function of the models when computing an exploration bonus, and a better generative model does not necessarily give rise to better probability estimates (Theis et al., 2016).

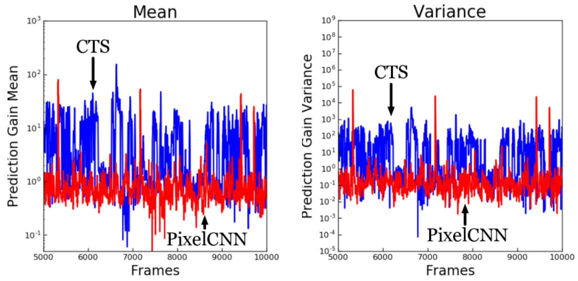

Figure 9. PG on MONTEZUMA'S REVENGE (log scale).

[4] The exact agent variant is referred to as 'β-LOO' with $\beta = 1$.

In Fig. 9 we compare the PG produced by the two models throughout 5K training steps. PixelCNN consistently produces PGs lower than CTS. More importantly, its PGs are *smoother*, exhibiting less variance between successive states, while showing more pronounced peaks at certain infrequent events. This yields a reward bonus that is less harmful in easy exploration games, while providing a strong signal in the case of novel or rare events.

Another distinguishing feature of PixelCNN is its non-decaying step-size. The per-step PG never completely vanishes, as the model tracks the most recent data. This provides an unexpected benefit: the agent remains mildly surprised by significant state changes, e.g. switching rooms in MONTEZUMA'S REVENGE. These persistent rewards act as *milestones* that the agent learns to return to. This is illustrated in Fig. 10, depicting the intrinsic reward over the course of an episode. The agent routinely revisits the right-hand side of the torch room, not because it leads to reward but just to "take in the sights". A video of the episode is provided at `http://youtu.be/232tOUPKPoQ`.[5]

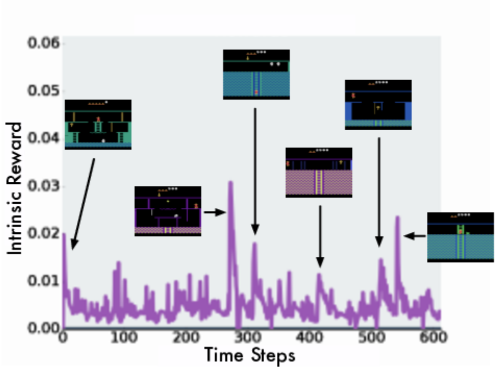

Figure 10. Intrinsic reward in MONTEZUMA'S REVENGE.

Lastly, PixelCNN's convolutional nature is expected to be beneficial for its sample efficiency. In Appendix C we compare to a convolutional CTS and confirm that this explains part, but not all of PixelCNN's advantage over vanilla CTS.

6. Importance of the Monte Carlo Return

Like for DQN-CTS, the success of DQN-PixelCNN hinges on the use of the mixed Monte Carlo update. The transient and vanishing nature of the exploration rewards requires the learning algorithm to latch on to these rapidly. The MMC serves this end as a simple multi-step method, helping to propagate reward information faster. An additional benefit lies in the fact that the Monte Carlo return helps bridging long horizons in environments where rewards are far apart and encountered rarely. On the other hand, it is important to note that the Monte Carlo return's on-policy nature increases variance in the learning algorithm, and can prevent the algorithm's convergence to the optimal policy when training off-policy. It can therefore be expected to adversely affect training performance in some games.

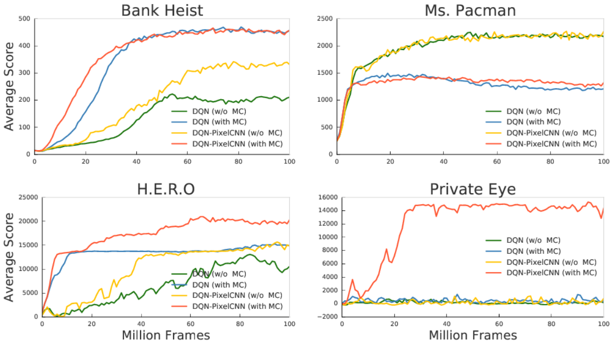

Figure 11. **Top**: games where MMC completely explains the improved/decreased performance of DQN-PixelCNN compared to DQN. **Bottom-left**: MMC and PixelCNN show additive benefits. **Bottom-right**: hard exploration, sparse reward game – only combining MMC and PixelCNN bonus achieves training progress.

To distill the effect of the MMC on performance, we compare all four combinations of DQN with/without PixelCNN exploration bonus and with/without MMC. Fig. 11 shows the performance of these four agent variants (graphs for all games are shown in Fig. 17). These games were picked to illustrate several commonly occurring cases:

- MMC speeds up training and improves final performance significantly (examples: BANK HEIST, TIME PILOT). In these games, MMC alone explains most or all of the improvement of DQN-PixelCNN over DQN.
- MMC hurts performance (examples: MS. PAC-MAN, BREAKOUT). Here too, MMC alone explains most of the difference between DQN-PixelCNN and DQN.
- MMC and PixelCNN reward bonus have a compounding effect (example: H.E.R.O.).

Most importantly, the situation is rather different when we restrict our attention to the hardest exploration games with sparse rewards. Here the baseline DQN agent fails to make any training progress, and neither Monte Carlo return nor the exploration bonus alone provide any significant benefit. Their combination however grants the agent rapid training progress and allows it to achieve high performance.

One effect of the exploration bonus in these games is to provide a denser reward landscape, enabling the agent to learn meaningful policies. Due to the transient nature of the exploration bonus, the agent needs to be able to learn from this reward signal faster than regular one-step methods allow, and MMC proves to be an effective solution.

[5] Another agent video on the game PRIVATE EYE can be found at `http://youtu.be/kNyFygeUa2E`.

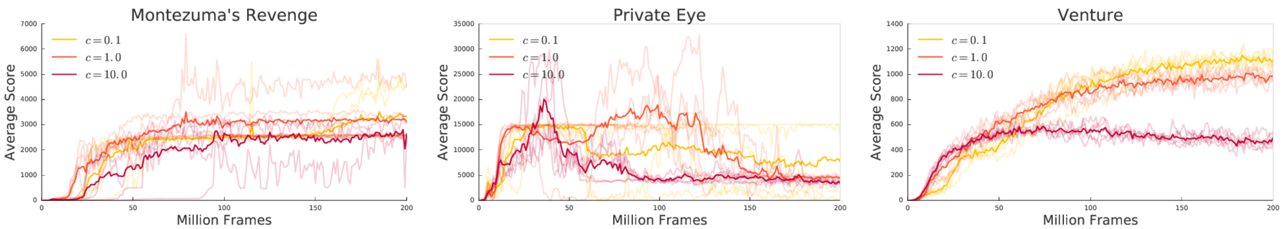

Figure 12. DQN-PixelCNN, hard exploration games, different PG scales $c \cdot n^{-1/2} \cdot \mathrm{PG}_n$ ($c = 0.1, 1, 10$) (5 seeds each).

7. Pushing the Limits of Intrinsic Motivation

In this section we explore the idea of a ‘maximally curious’ agent, whose reward function is dominated by the exploration bonus. For that we increase the PG scale, previously chosen conservatively to avoid adverse effects on easy exploration games.

Fig. 12 shows DQN-PixelCNN performance on the hardest exploration games when the PG scale is increased by 1-2 orders of magnitude. The algorithm seems fairly robust across a wide range of scales: the main effect of increasing this parameter is to trade off exploration (seeking maximal reward) with exploitation (optimizing the current policy).

As expected, a higher PG scale translates to stronger exploration: several runs obtain record peak scores (900 in GRAVITAR, 6,600 in MONTEZUMA’S REVENGE, 39,000 in PRIVATE EYE, 1,500 in VENTURE) surpassing the state of the art by a substantial margin (for previously published results, see Appendix D). Aggressive scaling speeds up the agent’s exploration and achieves peak performance rapidly, but can also deteriorate its stability and long-term performance. Note that in practice, because of the non-decaying step-size the PG does not vanish. After reward clipping, an overly inflated exploration bonus can therefore become essentially constant, no longer providing a useful intrinsic motivation signal to the agent.

Another way of creating an entirely curiosity-driven agent is to ignore the environment reward altogether and train based on the exploration reward only, see Fig. 13. Remarkably, the curiosity signal alone is sufficient to train a high-performing agent (measured by environment reward!).

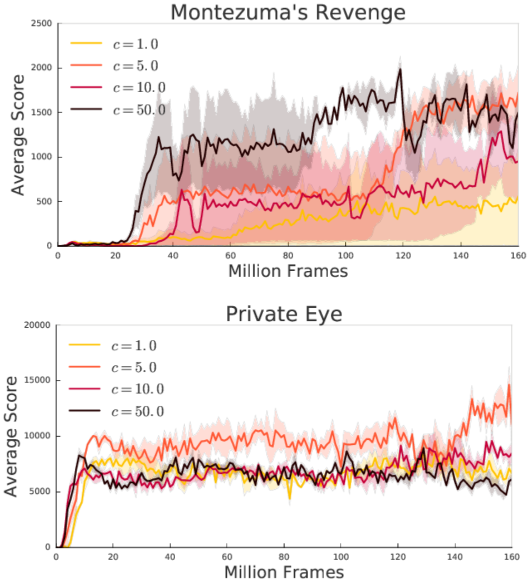

Figure 13. DQN-PixelCNN trained from **intrinsic reward only** (3 seeds for each configuration).

It is worth noting that agents with exploration bonus seem to ‘never stop exploring’: for different seeds, the agents make learning progress at very different times during training, a qualitative difference to vanilla DQN.

8. Conclusion

We demonstrated the use of PixelCNN for exploration and showed that its greater accuracy and expressiveness translate into a more useful exploration bonus than that obtained from previous models. While the current theory of pseudo-counts puts stringent requirements on the density model, we have shown that PixelCNN can be used in a simpler and more general setup, and can be trained completely online. It also proves to be widely compatible with both value-function and policy-based RL algorithms.

In addition to pushing the state of the art on the hardest exploration problems among the Atari 2600 games, PixelCNN improves speed of learning and stability of baseline RL agents across a wide range of games. The quality of its reward bonus is evidenced by the fact that on sparse reward games, this signal alone suffices to learn to achieve significant scores, creating a truly intrinsically motivated agent.

Our analysis also reveals the importance of the Monte Carlo return for effective exploration. The comparison with more sophisticated but fixed-horizon multi-step methods shows that its significance lies both in faster learning in the context of a useful but transient reward function, as well as bridging reward gaps in environments where extrinsic and intrinsic rewards are, or quickly become, extremely sparse.

Acknowledgements

The authors thank Tom Schaul, Olivier Pietquin, Ian Osband, Sriram Srinivasan, Tejas Kulkarni, Alex Graves, Charles Blundell, and Shimon Whiteson for invaluable feedback on the ideas presented here, and Audrunas Gruslys especially for providing the Reactor agent.

References

Bellemare, Marc, Veness, Joel, and Talvitie, Erik. Skip context tree switching. *Proceedings of the International Conference on Machine Learning*, 2014.

Bellemare, Marc G., Naddaf, Yavar, Veness, Joel, and Bowling, Michael. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.

Bellemare, Marc G., Srinivasan, Sriram, Ostrovski, Georg, Schaul, Tom, Saxton, David, and Munos, Rémi. Unifying count-based exploration and intrinsic motivation. *Advances in Neural Information Processing Systems*, 2016.

Deisenroth, Marc P. and Rasmussen, Carl E. PILCO: A model-based and data-efficient approach to policy search. In *Proceedings of the International Conference on Machine Learning*, 2011.

French, Robert M. Catastrophic forgetting in connectionist networks. *Trends in cognitive sciences*, 3(4):128–135, 1999.

Goodfellow, Ian, Pouget-Abadie, Jean, Mirza, Mehdi, Xu, Bing, Warde-Farley, David, Ozair, Sherjil, Courville, Aaron, and Bengio, Yoshua. Generative adversarial nets. In *Advances in Neural Information Processing Systems*, 2014.

Gregor, Karol, Danihelka, Ivo, Graves, Alex, Rezende, Danilo, and Wierstra, Daan. Draw: A recurrent neural network for image generation. In *Proceedings of the International Conference on Machine Learning*, 2015.

Gruslys, Audrunas, Azar, Mohammad Gheshlaghi, Bellemare, Marc G., and Munos, Rémi. The Reactor: A sample-efficient actor-critic architecture. *arXiv preprint arXiv:1704.04651*, 2017.

Guez, Arthur, Silver, David, and Dayan, Peter. Efficient bayes-adaptive reinforcement learning using sample-based search. In *Advances in Neural Information Processing Systems*, 2012.

Houthooft, Rein, Chen, Xi, Duan, Yan, Schulman, John, De Turck, Filip, and Abbeel, Pieter. Variational information maximizing exploration. In *Advances in Neural Information Processing Systems (NIPS)*, 2016.

Jaksch, Thomas, Ortner, Ronald, and Auer, Peter. Near-optimal regret bounds for reinforcement learning. *Journal of Machine Learning Research*, 11:1563–1600, 2010.

Kingma, Diederik P. and Welling, Max. Auto-encoding variational bayes. In *Proceedings of the International Conference on Learning Representations*, 2013.

Lopes, Manuel, Lang, Tobias, Toussaint, Marc, and Oudeyer, Pierre-Yves. Exploration in model-based reinforcement learning by empirically estimating learning progress. In *Advances in Neural Information Processing Systems*, 2012.

Mnih, Volodymyr, Kavukcuoglu, Koray, Silver, David, Rusu, Andrei A, Veness, Joel, Bellemare, Marc G., Graves, Alex, Riedmiller, Martin, Fidjeland, Andreas K., Ostrovski, Georg, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

Munos, Rémi, Stepleton, Tom, Harutyunyan, Anna, and Bellemare, Marc G. Safe and efficient off-policy reinforcement learning. In *Advances in Neural Information Processing Systems*, 2016.

Osband, Ian, van Roy, Benjamin, and Wen, Zheng. Generalization and exploration via randomized value functions. In *Proceedings of the International Conference on Machine Learning*, 2016.

Oudeyer, Pierre-Yves, Kaplan, Frédéric, and Hafner, Verena V. Intrinsic motivation systems for autonomous mental development. *IEEE Transactions on Evolutionary Computation*, 11(2):265–286, 2007.

Rezende, Danilo Jimenez, Mohamed, Shakir, and Wierstra, Daan. Stochastic backpropagation and approximate inference in deep generative models. In *Proceedings of The International Conference on Machine Learning*, 2014.

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

Sutton, Richard S. Generalization in reinforcement learning: Successful examples using sparse coarse coding. In *Advances in Neural Information Processing Systems*, 1996.

Sutton, Richard S. and Barto, Andrew G. *Reinforcement learning: An introduction*. MIT Press, 1998.

Tang, Haoran, Houthooft, Rein, Foote, Davis, Stooke, Adam, Chen, Xi, Duan, Yan, Schulman, John, De Turck, Filip, and Abbeel, Pieter. #Exploration: A study of count-based exploration for deep reinforcement learning. *arXiv preprint arXiv:1611.04717*, 2016.

Theis, Lucas, van den Oord, Aäron, and Bethge, Matthias. A note on the evaluation of generative models. In *Proceedings of the International Conference on Learning Representations*, 2016.

Tieleman, Tijmen and Hinton, Geoffrey. RMSProp: divide the gradient by a running average of its recent magnitude. *COURSERA. Lecture 6.5 of Neural Networks for Machine Learning*, 2012.

Tsitsiklis, John N. and van Roy, Benjamin. An analysis of temporal-difference learning with function approximation. *IEEE Transactions on Automatic Control*, 42(5): 674–690, 1997.

van den Oord, Aaron, Kalchbrenner, Nal, Espeholt, Lasse, Vinyals, Oriol, Graves, Alex, et al. Conditional image generation with PixelCNN decoders. In *Advances in Neural Information Processing Systems*, 2016a.

van den Oord, Aaron, Kalchbrenner, Nal, and Kavukcuoglu, Koray. Pixel recurrent neural networks. In *Proceedings of the International Conference on Machine Learning*, 2016b.

van Hasselt, Hado, Guez, Arthur, and Silver, David. Deep reinforcement learning with Double Q-learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2016.

Veness, Joel, Ng, Kee Siong, Hutter, Marcus, and Bowling, Michael H. Context tree switching. In *Proceedings of the Data Compression Conference*, 2012.

Wang, Ziyu, Schaul, Tom, Hessel, Matteo, van Hasselt, Hado, Lanctot, Marc, and de Freitas, Nando. Dueling network architectures for deep reinforcement learning. In *Proceedings of The 33rd International Conference on Machine Learning*, pp. 1995–2003, 2016.