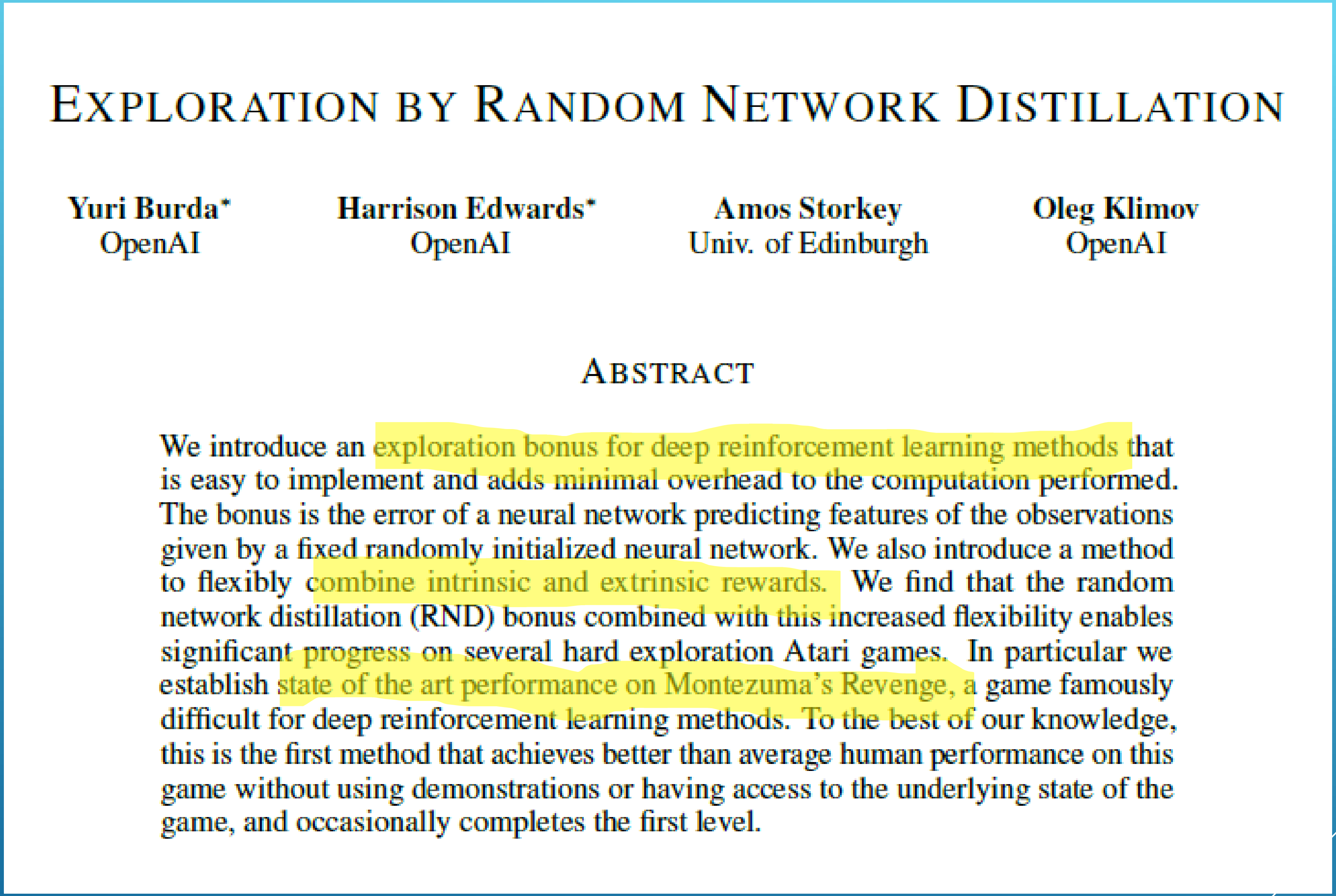
GROUP DISCUSSION: *"EXPLORATION BY RANDOM NETWORK DISTILLATION"*

Scott O'Hara

Metrowest Developers Machine Learning Group

01/16/2018



EXPLORATION BY RANDOM NETWORK DISTILLATION

Yuri Burda*
OpenAI

Harrison Edwards*
OpenAI

Amos Storkey
Univ. of Edinburgh

Oleg Klimov
OpenAI

ABSTRACT

We introduce an exploration bonus for deep reinforcement learning methods that is easy to implement and adds minimal overhead to the computation performed. The bonus is the error of a neural network predicting features of the observations given by a fixed randomly initialized neural network. We also introduce a method to flexibly combine intrinsic and extrinsic rewards. We find that the random network distillation (RND) bonus combined with this increased flexibility enables significant progress on several hard exploration Atari games. In particular we establish state of the art performance on Montezuma's Revenge, a game famously difficult for deep reinforcement learning methods. To the best of our knowledge, this is the first method that achieves better than average human performance on this game without using demonstrations or having access to the underlying state of the game, and occasionally completes the first level.

https://arxiv.org/pdf/1810.12894.pdf

REINFORCEMENT LEARNING: THE BASIC IDEA

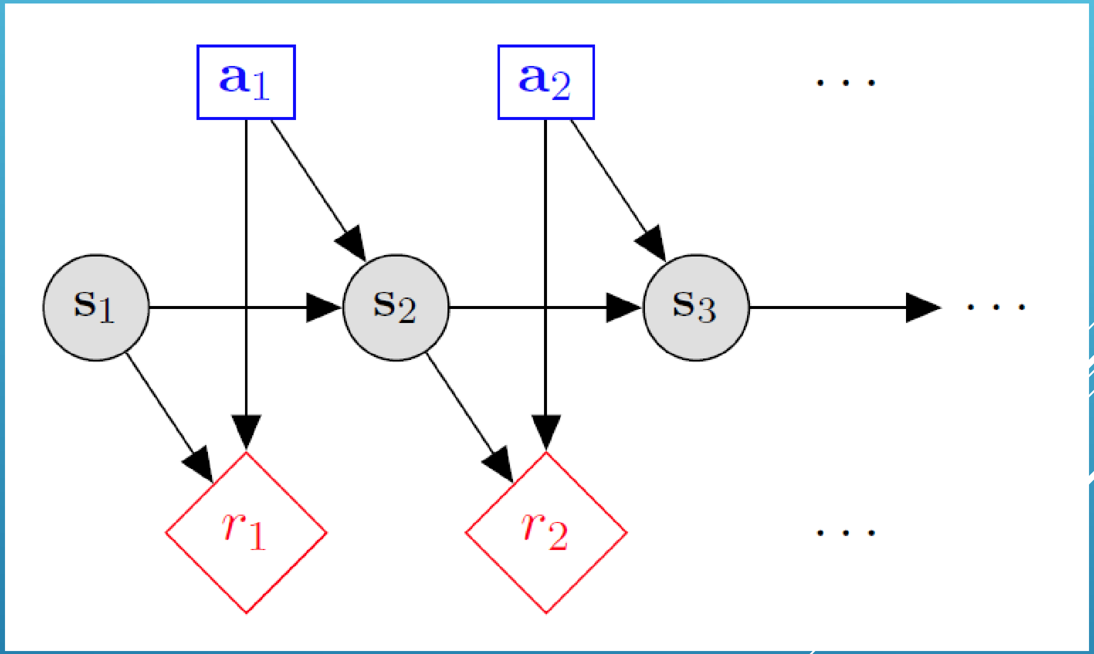
- Select an action.
- If action leads to reward, reinforce that action.
- If action leads to punishment, avoid that action.
- Basically, a computational form of Behaviorism (Pavlov, B. F. Skinner).

MARKOV DECISION PROCESSES

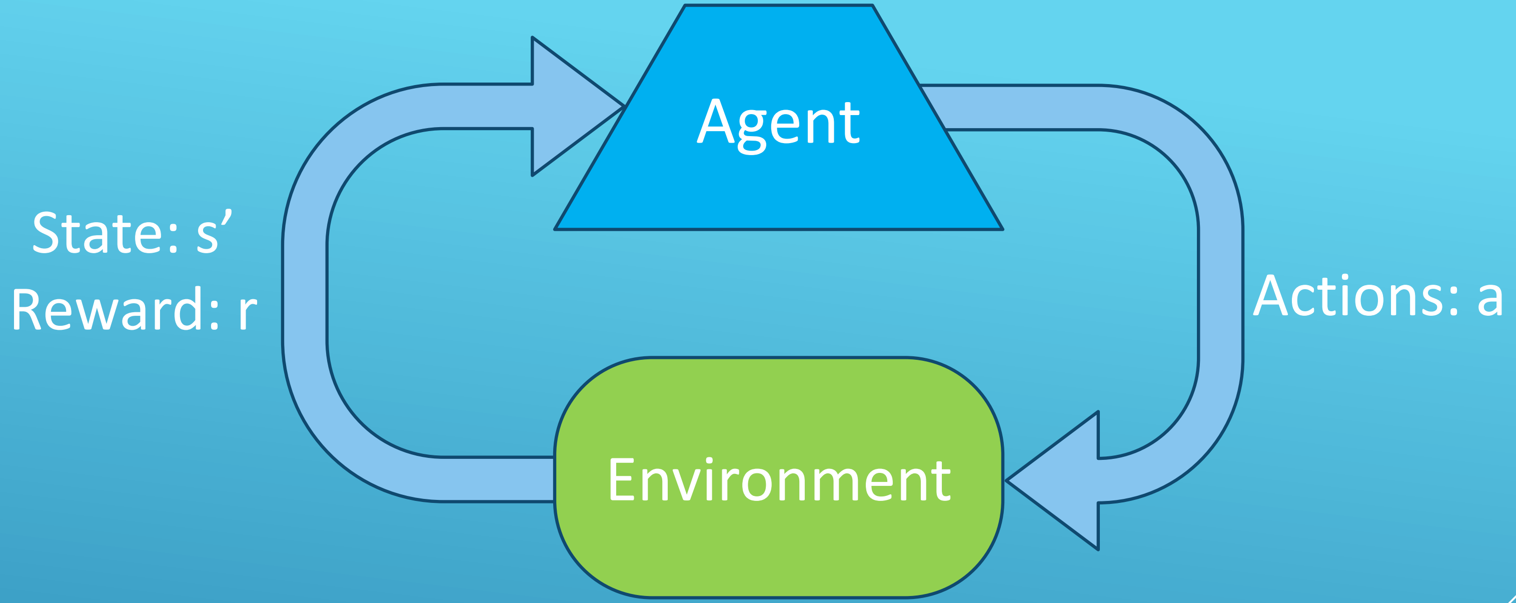
- **States:** $s_1, \ldots, s_n$
- **Actions:** $a_1, \ldots, a_m$
- **Reward Function:**

  $r(s, a, s') \in R$
- **Transition model:**

  $T(s,a,s') = P(s'|s, a)$
- **Discount factor:** $\gamma \in [0, 1]$



REINFORCEMENT LEARNING



- Agent knows the current state *s*, takes action *a*, receives a reward *r* and observes the next state *s'*
- Agent has **no access** to reward model *r(s,a)* or transition model *p(s' | s,a)*
- Agent must learn to act so as to maximize expected rewards.
- All learning is based on observed samples of outcomes!

CHOOSING AN ACTION: EXPLORATION VS EXPLOITATION

- How should an agent choose an action? An obvious answer is simply follow the current policy. However, this is often the best way to improve your model.
- **Exploit:** use your current model to maximize the expected utility now.
- **Explore:** choose an action that will help you improve your model.

HOW/WHEN SHOULD WE EXPLORE / EXPLOIT ?

- **How to Exploit?** use the current policy.

- **How to Explore?**
  - choose an action randomly
  - choose an action you haven't chosen yet
  - choose an action that will take you to an unexplored state.

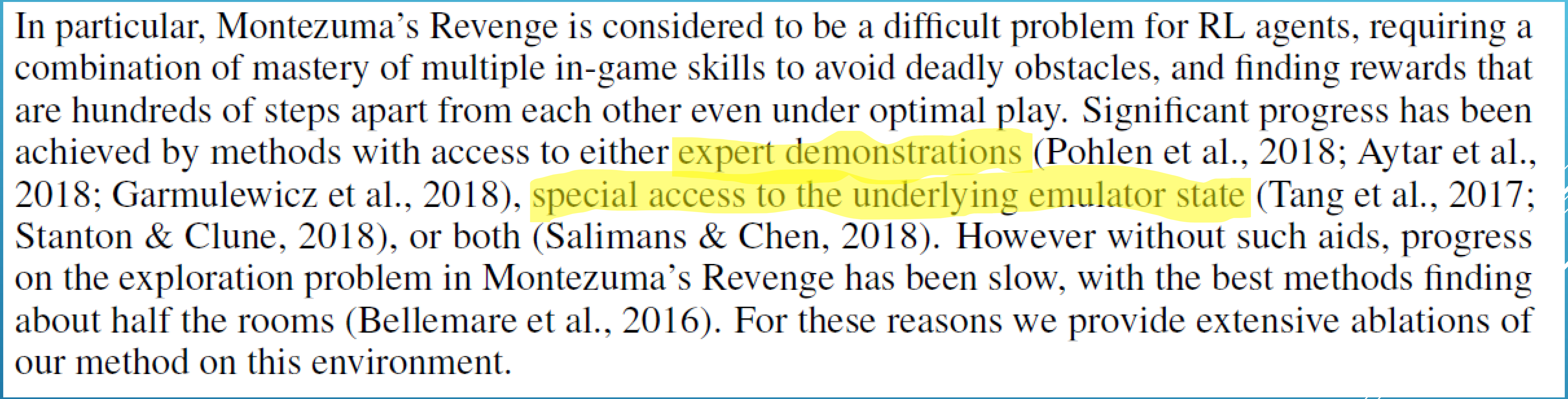
- **When to exploit? When to explore? How much time in exploration vs time in exploitation?**

Reinforcement Learning with Prediction-Based Rewards

OCTOBER 31, 2018

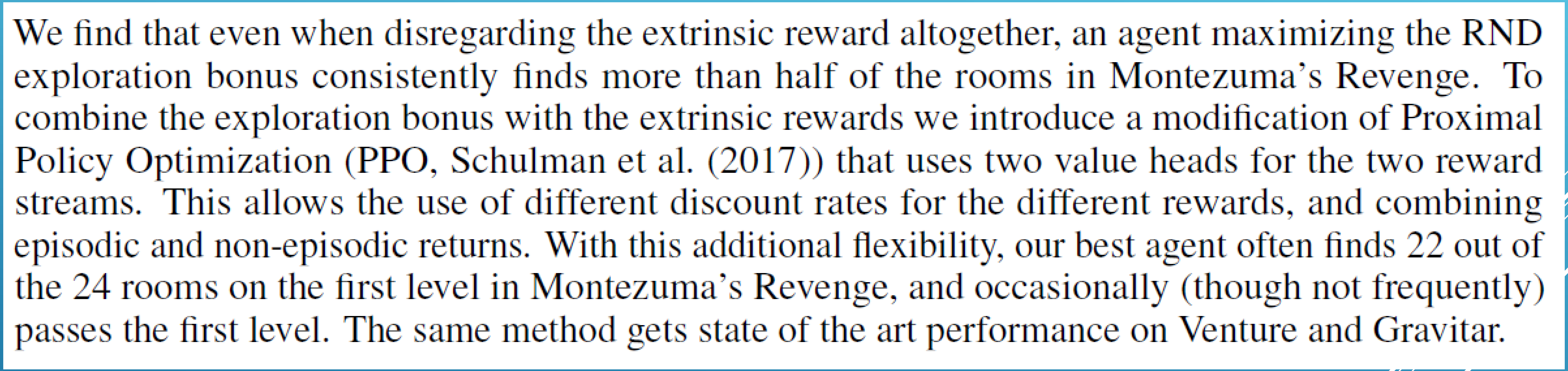
https://blog.openai.com/reinforcement-learning-with-prediction-based-rewards/

WHY MONTEZUMA'S REVENGE IS HARD



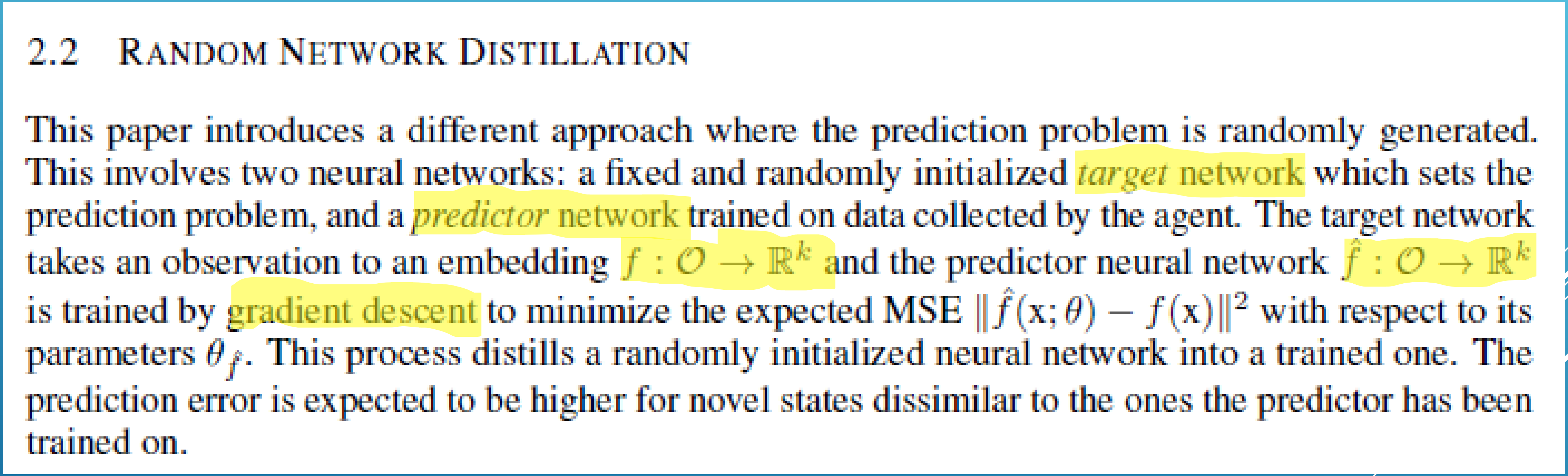
In particular, Montezuma's Revenge is considered to be a difficult problem for RL agents, requiring a combination of mastery of multiple in-game skills to avoid deadly obstacles, and finding rewards that are hundreds of steps apart from each other even under optimal play. Significant progress has been 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., 2017; Stanton & Clune, 2018), or both (Salimans & 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.

WHAT RND CONTRIBUTES



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.

RANDOM NETWORK DISTILLATION



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 $f : \mathcal{O} \to \mathbb{R}^k$ and the predictor neural network $\hat{f} : \mathcal{O} \to \mathbb{R}^k$ 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.

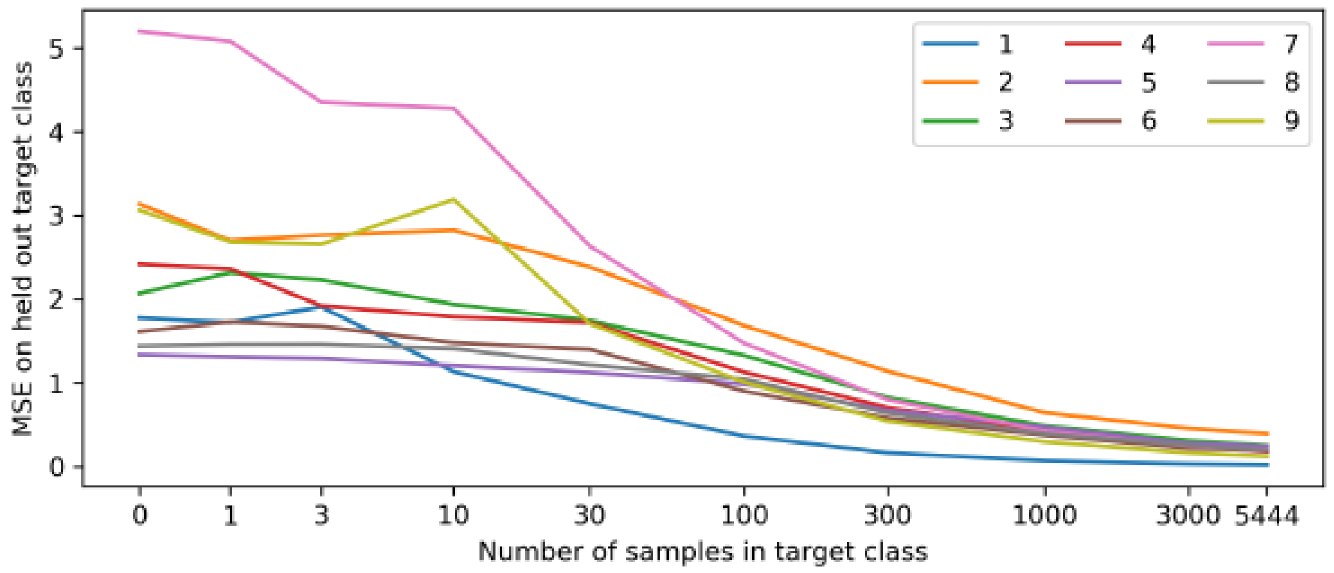
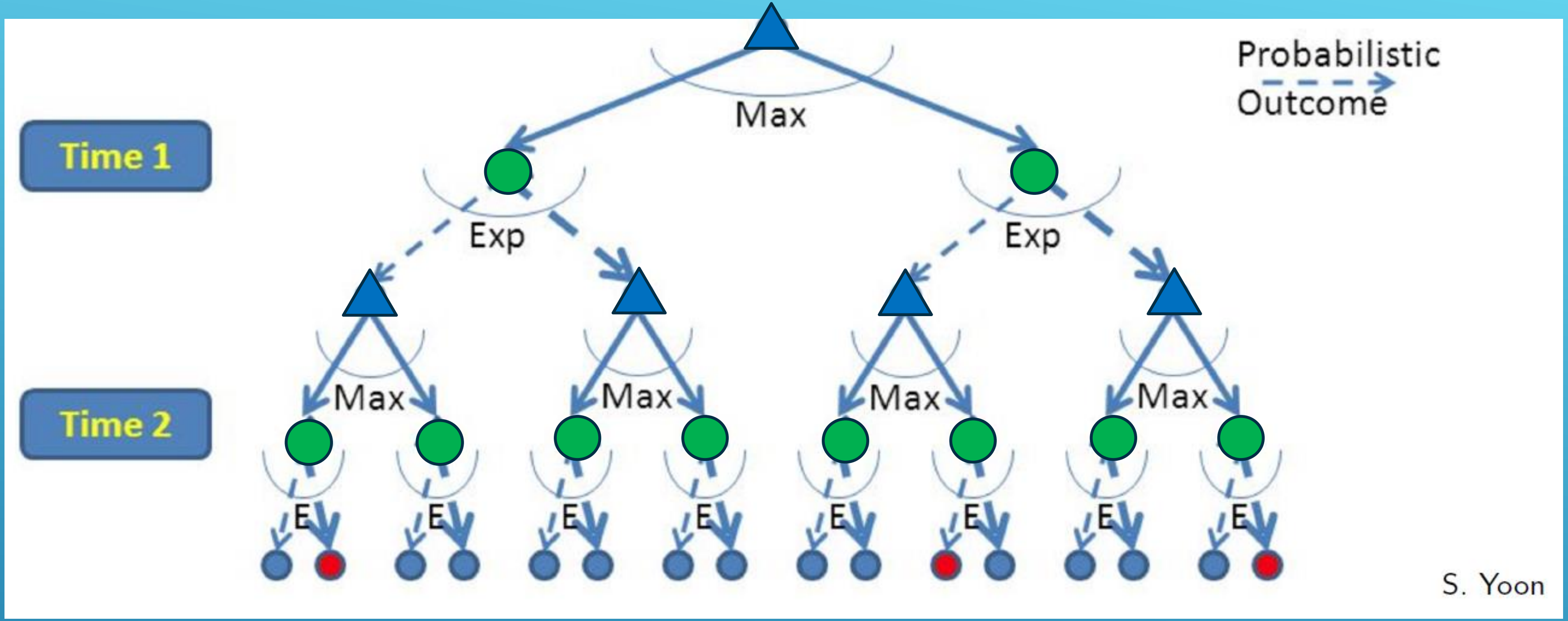


Figure 2: Novelty detection on MNIST: a predictor network mimics a randomly initialized target network. The training data consists of varying proportions of images from class "0" and a target class. Each curve shows the test MSE on held out target class examples plotted against the number of training examples of the target class (log scale).

SUPPLEMENTAL SLIDES

RL IS LIKE A GAME AGAINST NATURE



- Reinforcement learning is like a game-playing algorithm.
- Nodes where you move are called **states**: S (▲)
- Nodes where nature “moves” are called **Q-states**: <S,A> (●)

Credit: CS188 – Intro to AI, UC Berkeley, ai.berkely.edu

Q-LEARNING ALGORITHM

```
For each state s and action a:
    $\boxed{Q(s,a) \leftarrow 0}$
Begin in state s:
Repeat:
    Given s and the set of actions $A_s$ associated with s:
        ▸ **CHOOSE ACTION a** $\in A_s$ ← ***HOW?***
        ▸ Apply action a.
        ▸ Receive reward r and new state s'.
        ▸ With transition <s,a,r,s'>, update Q(s,a)
        $\boxed{Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha\left[r + \gamma \max_{a'} Q_k(s',a')\right]}$
    $\boxed{\mathrm{S} \leftarrow s'}$
```