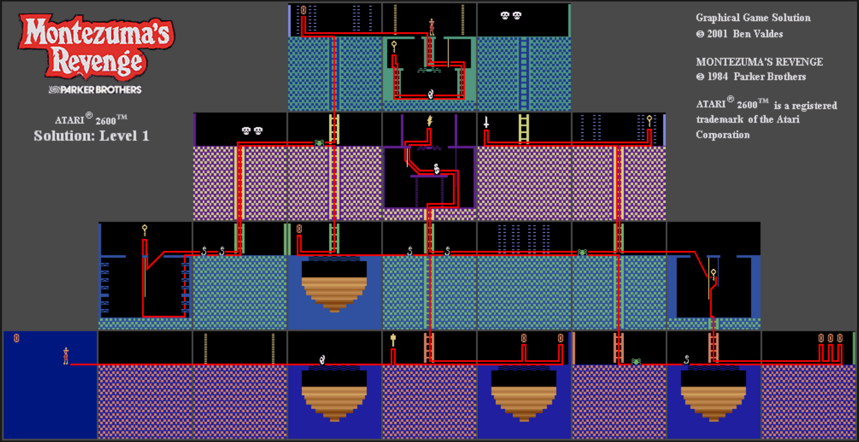
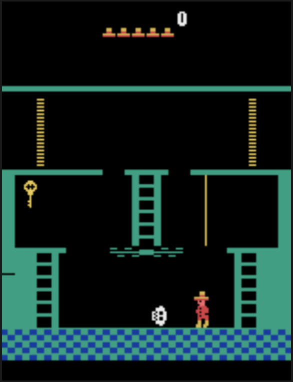
challenging Montezuma's Revenge
intrinsic motivation in RL
Michal CHOVANEC

Montezuma’s Revenge



Montezuma's Revenge



- **very sparse rewards** - hundrets of steps
- **huge state space**
- **hard exploration**
- **needs returns back**

state of the art score

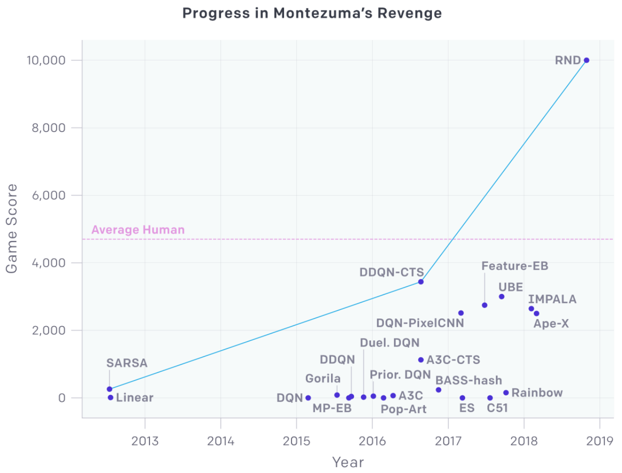
source : `https://paperswithcode.com/sota/atari-games-on-atari-2600-montezumas-revenge`

year	name	score
2015	Deep Reinforcement Learning with Double Q-learning	0
2017	Curiosity-driven Exploration by Self-supervised Prediction [a]	0
2021	MuZero	2500
2018	Count-Based Exploration with Neural Density Models [b]	3705
2019	**Exploration by Random Network Distillation** [c]	**8152**
2021	GoExplore* [d]	43 000

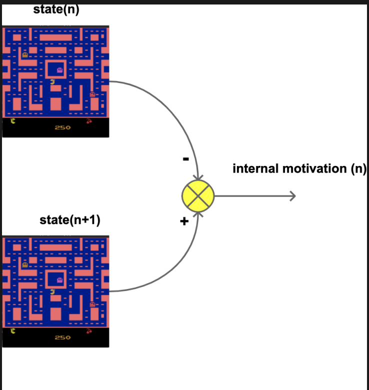
*** requires environment state saving/loading**

[a] https://arxiv.org/abs/1705.05363
[b] https://arxiv.org/abs/1703.01310
[c] https://arxiv.org/abs/1810.12894
[d] https://arxiv.org/abs/2004.12919

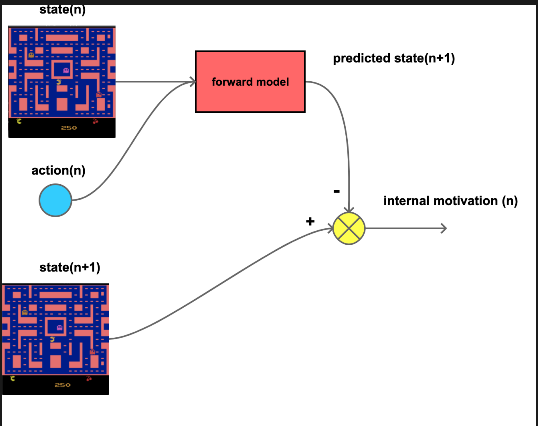
state of the art score



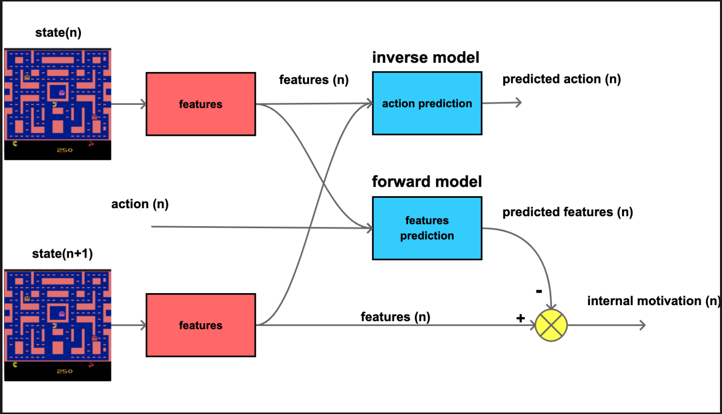
pixel change motivation



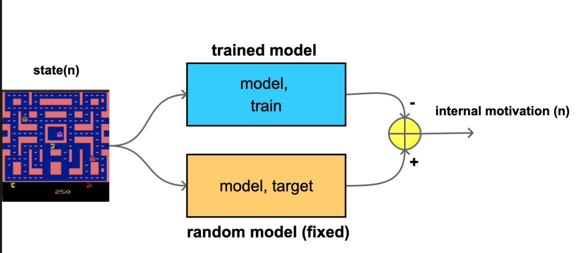
next state prediction



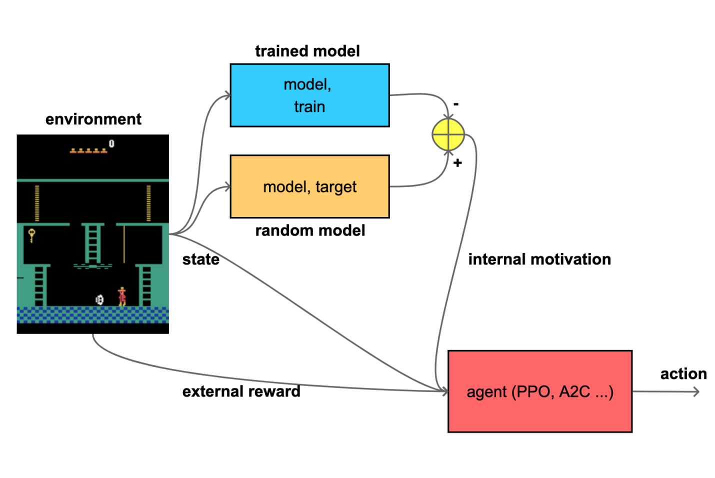
intrinsic curiosity module



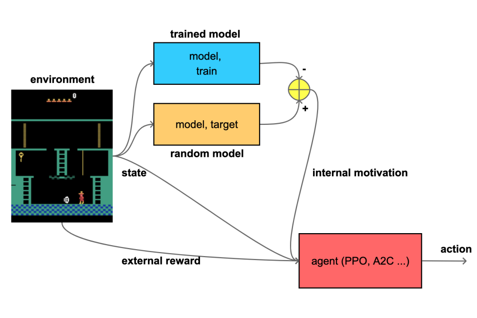
random network distillation



random network distillation

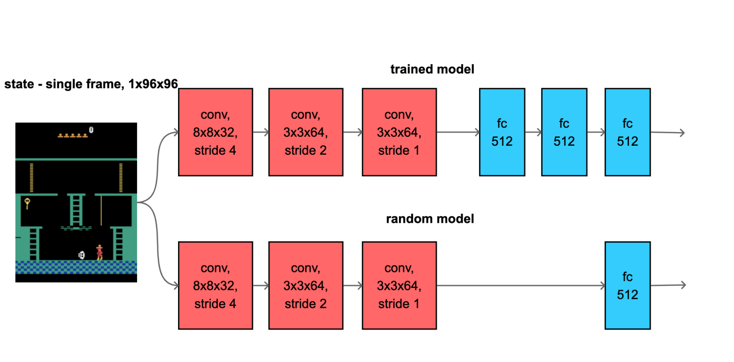


random network distillation

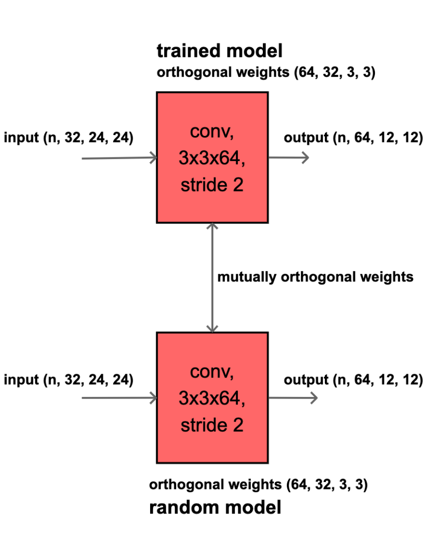


- neural network works as **novelty detector**
- model learns to imitate random (target) model
- **less visited states produce bigger motivation signal**
- orthogonal weights initialisation ($g = 2^{0.5}$) for strong signal
- lot of fully connected layers **to avoid generalisation**
- **coupled orthogonal models**

random network distillation architecture

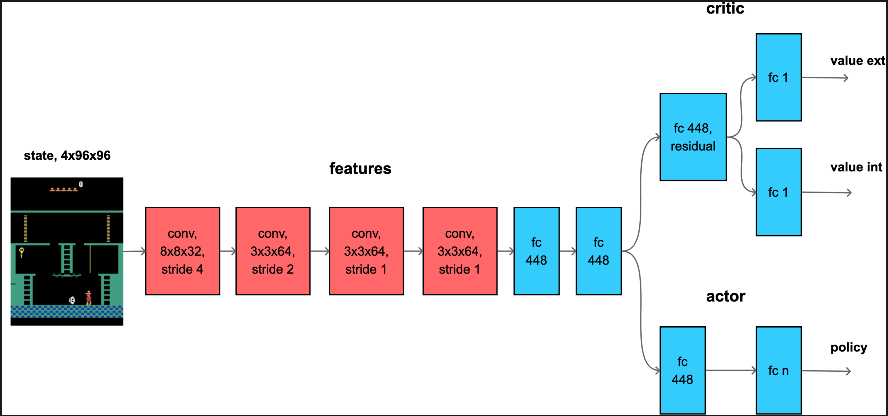


coupled RND architecture

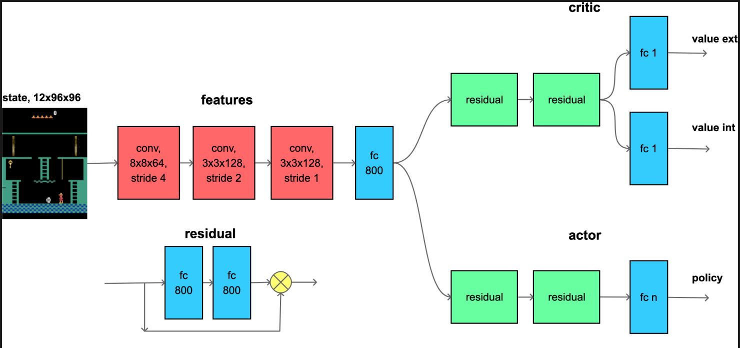



```python
def coupled_ortohogonal_init(shape, gain):
    w = torch.zeros((2*shape[0], ) + shape[1:])
    torch.nn.init.orthogonal_(w, gain)

    w = w.reshape((2, ) + shape)
    return w[0], w[1]

wa, wb = coupled_ortohogonal_init((64, 32, 3, 3), 2.0**0.5)
```

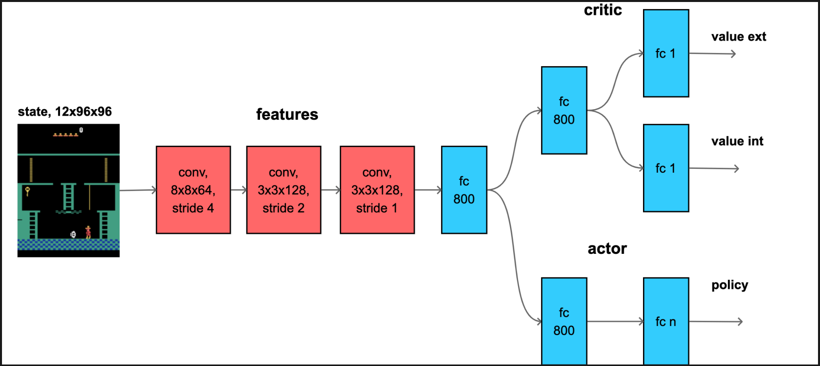
ppo model architecture A



ppo model architecture B

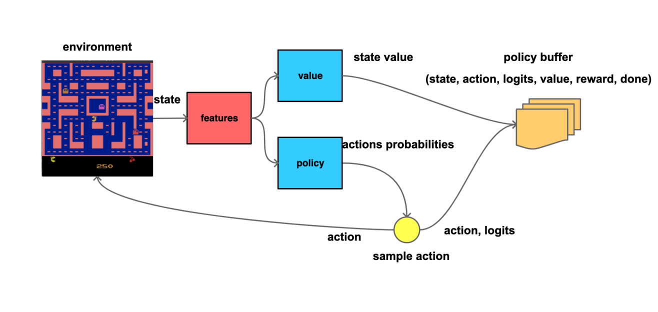


ppo model architecture C

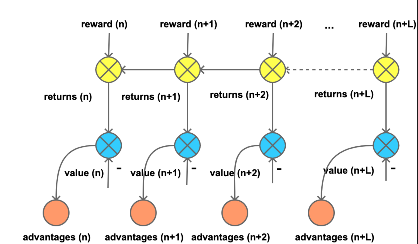


proximal policy optimization - PPO

Schulman, 2017, `https://arxiv.org/pdf/1707.06347.pdf`

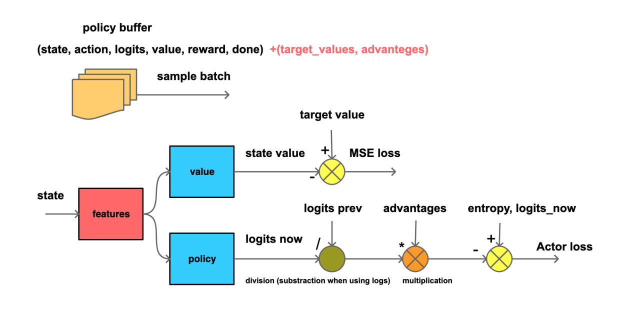


PPO - computing target values and returns



1. compute returns (target values), using Q-learning
2. compute advantages as difference between returns and current values

PPO - training models



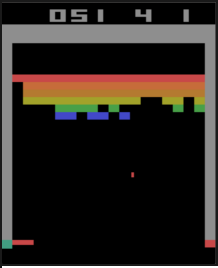
- critic uses common MSE loss
- actors uses loss (excluding clipping terms)

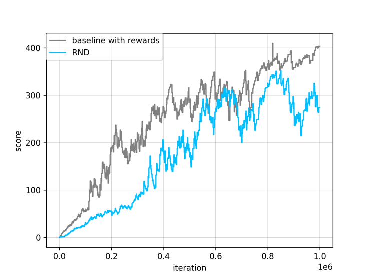
$$\mathcal{L} = \frac{1}{N}\sum_{n}^{N}\frac{\pi^{now}(a_n|s_n)}{\pi^{prev}(a_n|s_n)}A_n^{\pi^{old}}$$

experiments

- Breakout, without rewards
- Pacman, without rewards
- Montezuma's rewenge (tons of experiments)

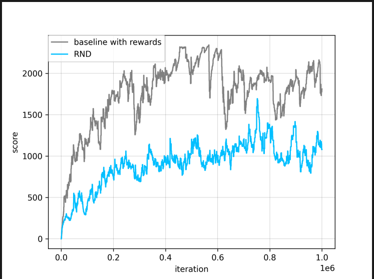
breakout





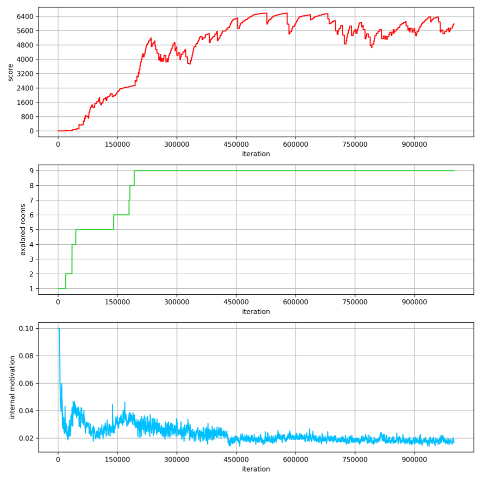
- 1M, 32parallel envs, total 32M steps
- PPO model A
- ext reward weight 2.0
- int reward weight 1.0

pacman



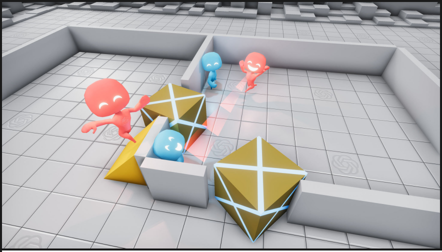
- 1M, 32parallel envs, total 32M steps
- PPO model A
- ext reward weight 2.0
- int reward weight 1.0

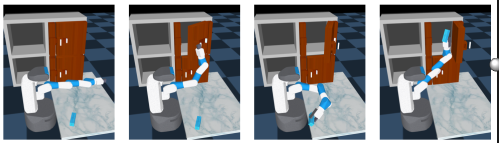
results



- 1M steps - **20% of original paper**
- 128 parallel envs = total 128M steps
- **score 6400**
- **9 rooms explored**

Emergent Tool Use From Multi-Agent Autocurricula





- multi-agent robotic environment
- hide and seek
- https://openai.com/blog/emergent-tool-use/
- https://arxiv.org/abs/1909.07528

Q&A