Improving sample-efficiency of model-free reinforcement learning algorithms on image inputs with representation learning

Marko Guberina
Betelhem Dejene Desta

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg

Gothenburg, Sweden 2022

UNIVERSITY OF GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Presentation structure

Project overview

Hypothesis

- state (feature) extraction
- dynamics modelling
- reward dynamics modelling

Joint training hypothesis

Joint training is better than pretraining.

Joint training hypothesis

Joint training is better than pretraining.

Better features hypothesis.

The more features are aligned with the underlying Markov chain, the better they work as state representations.

Joint training hypothesis

Joint training is better than pretraining.

Better features hypothesis.

The more features are aligned with the underlying Markov chain, the better they work as state representations.

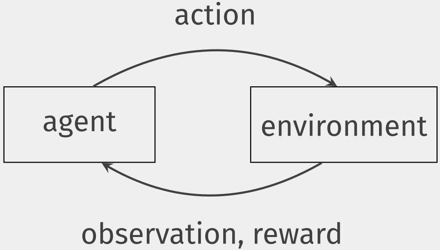
Regularization hypothesis.

Proper regularization helps when learning different objectives.

Reinforcement learning

What is reinforcement learning?

- formalized "trial-and-error" learning
- needs a **reward function**
- trade-off between exploration and exploitation



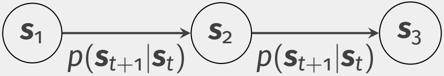


Figure: Schematic of a Markov chain.

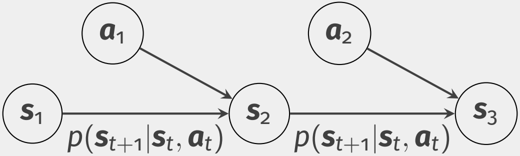


Figure: Schematic of a Markov decision process.

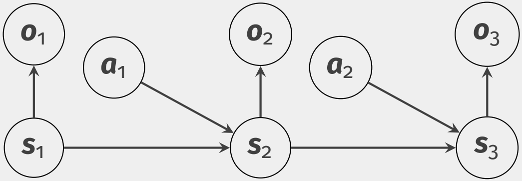


Figure: Schematic of a partially observable Markov decision process.

Markov decision process with a policy

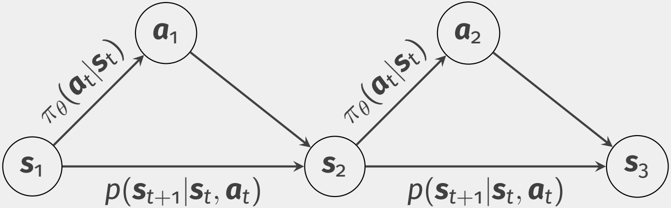


Figure: Schematic of a Markov decision process with a policy π.

$$\underbrace{p_\theta(\boldsymbol{s}_1, \boldsymbol{a}_1, \ldots, \boldsymbol{s}_T, \boldsymbol{a}_T)}_{p_\theta(\tau)} = p(\boldsymbol{s}_1) \prod_{t=1}^{T} \underbrace{\pi_\theta(\boldsymbol{a}_t | \boldsymbol{s}_t) p(\boldsymbol{s}_{t+1} | \boldsymbol{s}_t, \boldsymbol{a}_t)}_{\text{Markov chain on } (\boldsymbol{s},\boldsymbol{a})}$$

The goal of reinforcement learning

Find policy parameters $\theta^\star$ such that:

$$
\begin{aligned}
\theta^\star &= \operatorname*{argmax}_\theta \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[\sum_t r(\boldsymbol{s}_t, \boldsymbol{a}_t) \right] \\
&= \operatorname*{argmax}_\theta \sum_t^T \mathbb{E}_{(\boldsymbol{s}_t, \boldsymbol{a}_t) \sim p_\theta(\boldsymbol{s}_t, \boldsymbol{a}_t)} \left[r(\boldsymbol{s}_t, \boldsymbol{a}_t) \right]
\end{aligned}
$$

Q-function:

$$Q^{\pi}(\boldsymbol{s}_t, \boldsymbol{a}_t) = \sum_{t'=t}^{T} \mathbb{E}_{\pi_\theta}\left[r(\boldsymbol{s}_{t'}, \boldsymbol{a}_{t'})|\boldsymbol{s}_t, \boldsymbol{a}_t\right]$$

State value function:

$$V^{\pi}(\boldsymbol{s}_t) = \sum_{t'=t}^{T} \mathbb{E}_{\pi_\theta}\left[r(\boldsymbol{s}_{t'}, \boldsymbol{a}_{t'}|\boldsymbol{s}_t)\right]$$

Their connection:

$$V^{\pi}(\boldsymbol{s}_t) = \mathbb{E}_{\boldsymbol{a}_t \sim \pi(\boldsymbol{s}_t, \boldsymbol{a}_t)}\left[Q^{\pi}(\boldsymbol{s}_t, \boldsymbol{a}_t)\right]$$

Classes of reinforcement learning algorithms

Based on objective:

- policy gradient algorithms
- actor-critic algorithms
- value iteration algorithms

Based on sampling strategy:

- on-policy
- off-policy

REINFORCE algorithm:

1. sample $\{\tau^i\}$ from $\pi_\theta(\boldsymbol{a}_t|\boldsymbol{s}_t)$ by running the policy
2. use the samples to estimate the gradient of the objective:
   $\nabla_\theta J(\theta) \approx \sum_i \left(\sum_t^T \nabla_\theta \log \pi_\theta(\boldsymbol{a}_{i,t}|\boldsymbol{s}_{i,t})\right) \left(\sum_t r(\boldsymbol{s}_{i,t}, \boldsymbol{a}_{i,t})\right)$
3. update the policy function by performing a step of gradient ascent:
   $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

Actor-critic algorithm template

1. take action $\boldsymbol{a} \sim \pi_\theta(\boldsymbol{a}|\boldsymbol{s})$, observe transition $(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s'}, r)$ and store it in the replay buffer $\mathcal{R}$
2. sample a batch $\big\{(\boldsymbol{s}_i, \boldsymbol{a}_i, \boldsymbol{s}'_i, r_i)\big\}$ from buffer $\mathcal{R}$
3. update the Q-value estimator $\hat{Q}^\pi_\theta$ by using the target: $y_i = r_i + \gamma \hat{Q}^\pi_\theta(\boldsymbol{s}'_i, \boldsymbol{a}'_i) \forall \boldsymbol{s}_i, \boldsymbol{a}_i$
4. compute the policy gradient estimate with: $\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_i \nabla_\theta \log \pi_\theta(\boldsymbol{a}^\pi_i|\boldsymbol{s}_i) \hat{Q}^\pi(\boldsymbol{s}_i, \boldsymbol{a}^\pi_i)$, where $\boldsymbol{a}^\pi_i \sim \pi_\theta(\boldsymbol{a}|\boldsymbol{s}_i)$
5. update the policy function by performing a gradient step: $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

$$\pi_{\text{greedy}}(\boldsymbol{s}_t|\boldsymbol{a}_t) = \begin{cases} 1 & \text{if } \boldsymbol{a}_t = \operatorname{argmax}_{\boldsymbol{a}_t} A^{\pi}(\boldsymbol{s}_t, \boldsymbol{a}_t) \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

Bootstrap update for the value function:

$$V^{\pi}(\boldsymbol{s}) \leftarrow E_{\boldsymbol{a} \sim \pi(\boldsymbol{a}|\boldsymbol{s})} \left[r(\boldsymbol{s}, \boldsymbol{a}) + \gamma E_{\boldsymbol{s}' \sim p(\boldsymbol{s}'|\boldsymbol{a},\boldsymbol{s})}[V^{\pi}(\boldsymbol{s}')] \right] \tag{2}$$

Value iteration

1. set $Q(\boldsymbol{s}, \boldsymbol{a}) \leftarrow r(\boldsymbol{s}, \boldsymbol{a}) + \gamma E[V(\boldsymbol{s}')]$
2. set $V(\boldsymbol{s}) \leftarrow \max_{\boldsymbol{a}} Q(\boldsymbol{s}, \boldsymbol{a})$

DQN

"Classic" DQN

1. take some action $\boldsymbol{a}_i$, observe $(\boldsymbol{s}_i, \boldsymbol{a}_i, \boldsymbol{s}'_i, r_i)$ and add it to $\mathcal{B}$
2. sample a mini-batch $\left(\boldsymbol{s}_j, \boldsymbol{a}_j, \boldsymbol{s}'_j, r_j\right)$ from $\mathcal{B}$ uniformly
3. compute $y_j = r_j + \gamma \max_{\boldsymbol{a}'_j} Q_{\phi'}(\boldsymbol{s}'_j, \boldsymbol{a}'_j)$ using the *target* network $Q_{\phi'}$
4. $\phi \leftarrow \phi - \alpha \sum_j \frac{dQ_\phi}{d\phi}(\boldsymbol{s}_j, \boldsymbol{a}_j) \left(Q_\phi(\boldsymbol{s}_i, \boldsymbol{a}_i) - y_j\right)$
5. update ϕ': copy ϕ every N steps

DQN with the following improvements:

- double Q-networks
- multi-step returns
- prioritized replay buffer
- dueling network
- noisy networks

Representation learning

General remarks

- goal is to learn a parametric mapping from raw input data to a feature vector in order to capture and extract useful abstract information
- works with unsupervised learning
- generative and discriminative models

- deterministic:
  - autoencoders (AEs)
- probabilistic:
  - variational autoencoders (VAEs)
  - generative adversarial networks (GANs)

Discriminative models

- have only encoders
- trained with:
  - contrastive learning
  - bootstrapping
  - ...

Regularization for autoencoders

Known from [BCV13], [Gho+19] and others:

- on input:
  - denoising autoencoders
- on bottleneck:
  - noise injection
  - Tikhonov regularization (L2 regularization)
- other:
  - gradient penalty (weight decay)
  - spectral normalization

Representation learning for control

Desirable properties

On top of general desirable properties for representations:

- having the Markov property
- represent states well enough for policy improvement
- generalize in the stateful sense

Types of models

- autoencoders
- forward models
- inverse models
- hybrid models

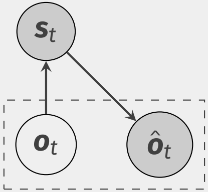


Figure: Auto-encoder: learned by reconstructing the observation (one-to-one). The observation is the input and the computed state is the vector at the auto-encoder's bottleneck layer.

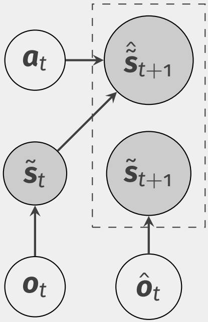


Figure: Forward model: predicting the future state from the state-action pair.

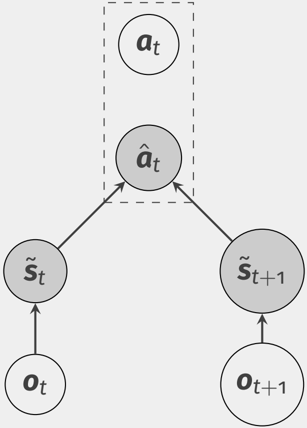


Figure: Inverse model: predicting the action between two consecutive states.

Related work

Reinforcement learning on Atari games

- started with DQN [Mni+13]
- many improvements:
  - algorithm fundamentals, combination in [Hes+18]
  - exploration schemes: [Pat+17], [Eco+21]
  - better sampling: [And+17], [Kap+18]
- model based algorithms: [Sch+20a]
- all solved on human level in [Bad+20]

- started with DQN [Mni+13]
- many improvements:
  - algorithm fundamentals, combination in [Hes+18]
  - exploration schemes: [Pat+17], [Eco+21]
  - better sampling: [And+17], [Kap+18]
- model based algorithms: [Sch+20a]
- all solved on human level in [Bad+20]

Next challenge

Making reinforcement learning more sample-efficient.

Using data augmentation for regularization

- simply adding data augmentation to RL: [Las+20]
- using it to regularize RL: [KYF20], [Yar+21]

Leveraging unsupervised representation learning

- early efforts for state representation learning did not work well
- initially used to help exploration: [She+16], [Jad+16] or [Pat+17]
- recent efforts use both deterministic and stochastic generative models (mostly stochastic)
- most recent works focus on using discriminative models
- ideally obtained solely via pretrainining; recent efforts include [Seo+22]

Idea introduced in [LR10]. Most importantly used in [Yar+19]. Authors identify the following for success:

- only value function gradients update the encoder
- same update rate for autoencoder and RL updates
- using L2 regularization

Possible improvements:

- prediction architecture from [Oh+15]
- optical or latent flow: [Sha+21]

Theoretically more interesting because:

- can be integrated into the underlying Markov chain: [Lee+20]
- can be used as models in model-based RL

Despite large interest they are hard to get to work due to their stochasticity (elaborated and tested in [Yar+19]).

More practical as there is no decoder (which is unnecessary for the purpose). Trained using different objectives:

- contrastive loss: [LSA20]
- mutual information: [Rak+21], [Ana+19], [Maz+20]
- bisimulation metrics: [Zha+20]
- bootstrapped self-predictions (introduced in [Gri+20]): [Sch+20b], and in [Mer+22]

Methods

Our hypotheses

We want to test the following claims:

1. joint representation and reinforcement training performs better than pretraining
2. representation better incentivised to learn stateful information will yield better features
3. regularization is important for joint training stabilization and final effectiveness

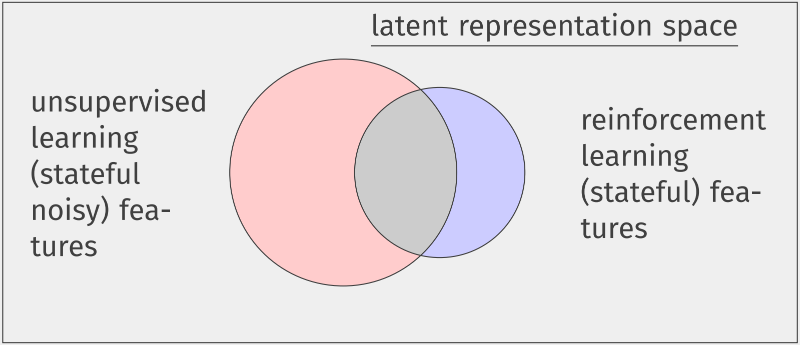


Figure: Schematic of the latent representation space.

Testing hypothesis 1

We can only indirectly test the hypothesis by observing the obtained returns on different games. We run the following experiments and observe the results:

1. only RL
2. only RL, but on encoders from a finished RL run
3. only RL, but on encoders pretrained with pixel reconstruction loss
4. joint training from scratch

1. joint training where unsupervised learning task is only compression
2. joint training where unsupervised learning task is compression and one-step forward prediction in pixel space

Testing hypothesis 3

1. joint training with no regularization
2. joint training with L2 regularization
3. joint training with L2 regularization and data augmentation

Module implementation

We really implement our add-on module as an add-on module in the reinforcement learning library Tianshou.
This is possible because Tianshou abstract different parts of reinforcement learning.
We implement our module as a policy wrapper.

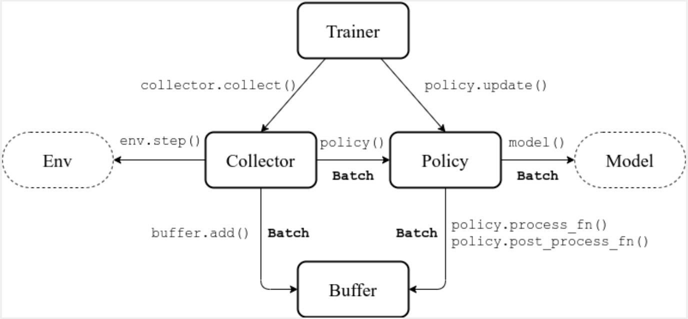


Figure: Tianshou abstractions.

Network architectures and hyperparameters

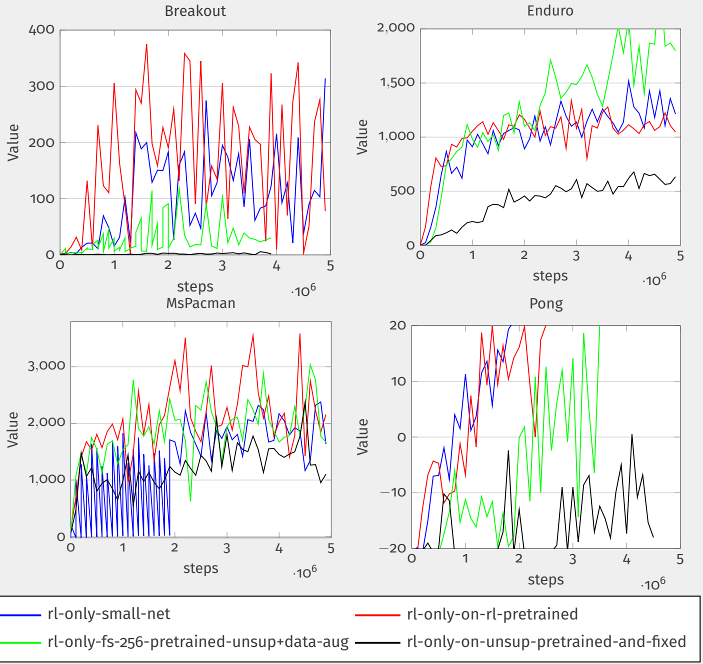
- all reinforcement learning parameters are kept equal and correspond to those in [Hes+18]
- two sized encoders, 2-D convolutional layers specified as (number of input channels, number of output channels, kernel size, stride, padding):
  - smaller, same as in [Mni+15]:
    (number of stacked frames, 32, 8, 4, 0), (32, 64, 4, 2, 0), (64, 64, 3, 1, 0)
  - bigger, same as in [Yar+19]:
    (number of stacked frames, 32, 3, 2, 0), (32, 32, 3, 2, 0), (32, 32, 3, 2, 0), (32, 32, 3, 2, 0)
    followed by a linear layer of shape ($32 \times 35 \times 35$, features dimension)

Results

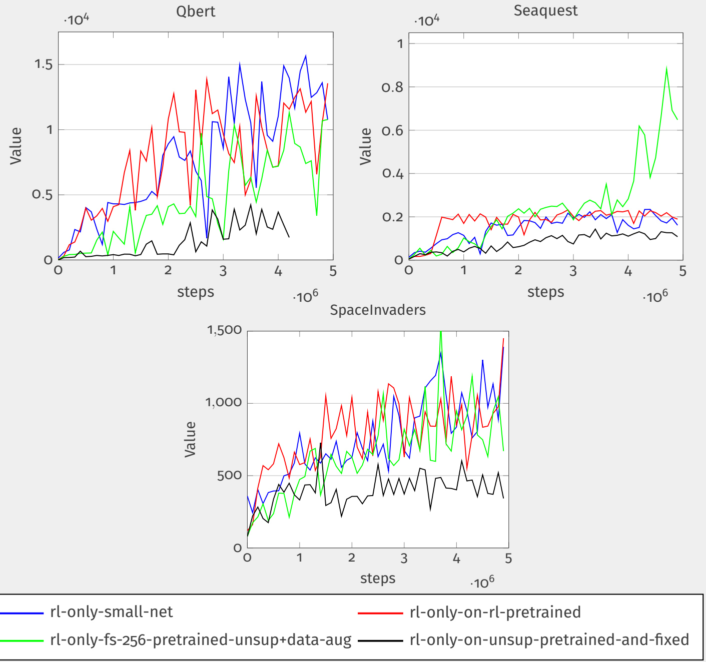
Games

1. Breakout
2. Enduro
3. Ms Pacman
4. Pong
5. Qbert
6. Seaquest
7. Space Invaders

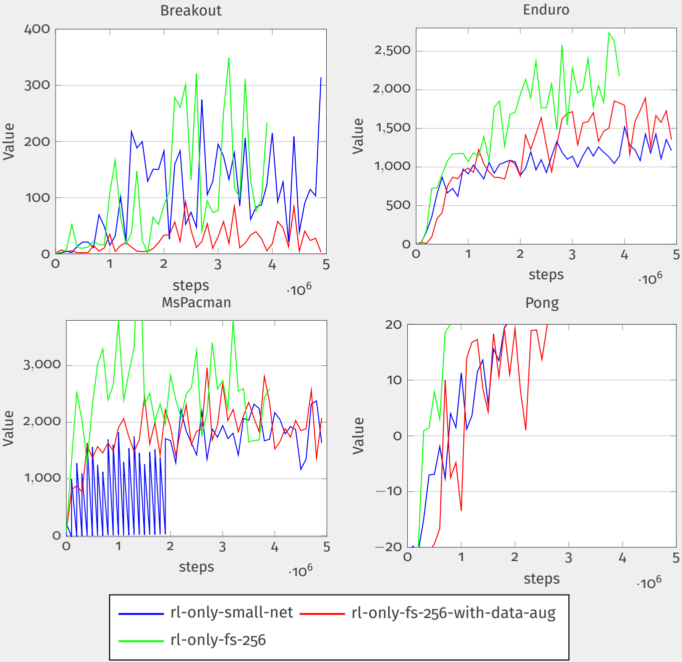
Effectiveness of pretraining



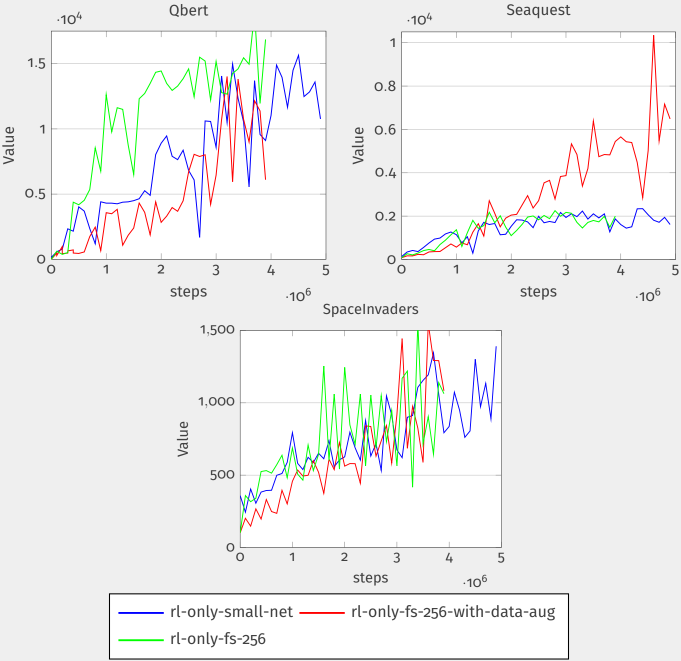
Effectiveness of pretraining — continued

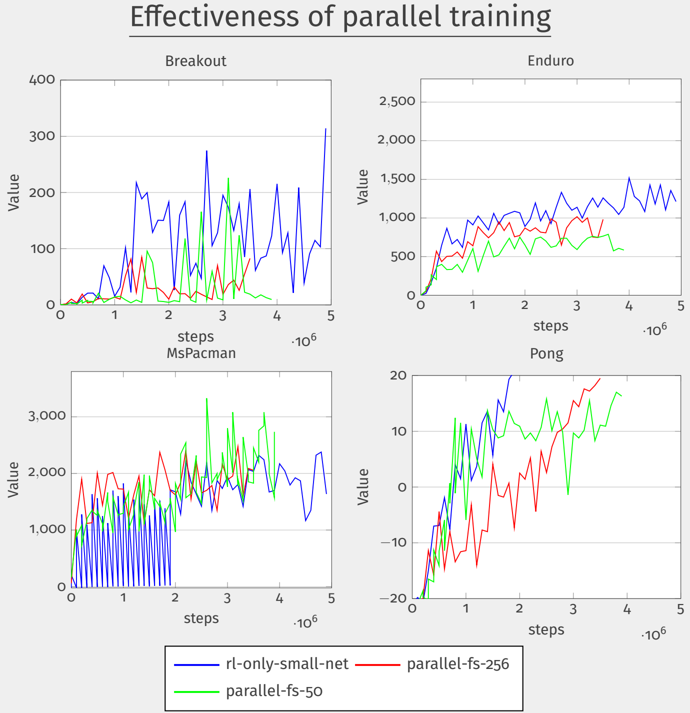


Effect of different encoder sizes on RL



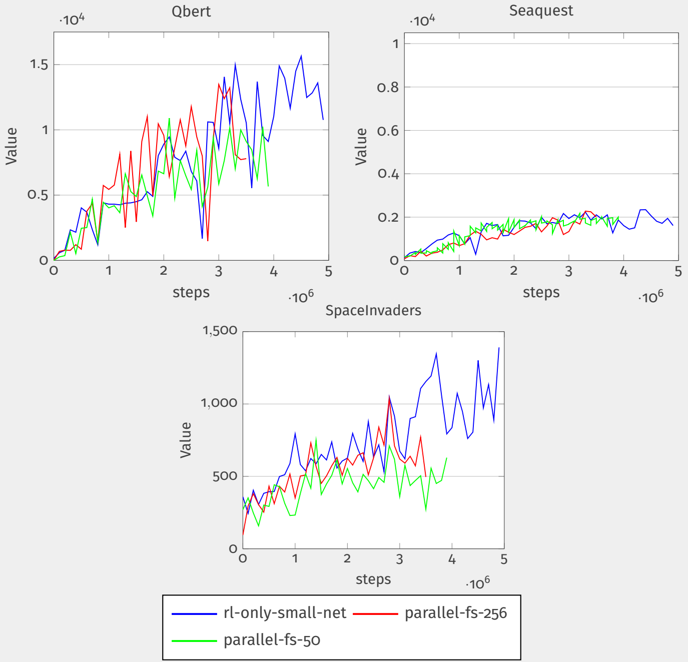
Effect of different encoder sizes on RL — continued



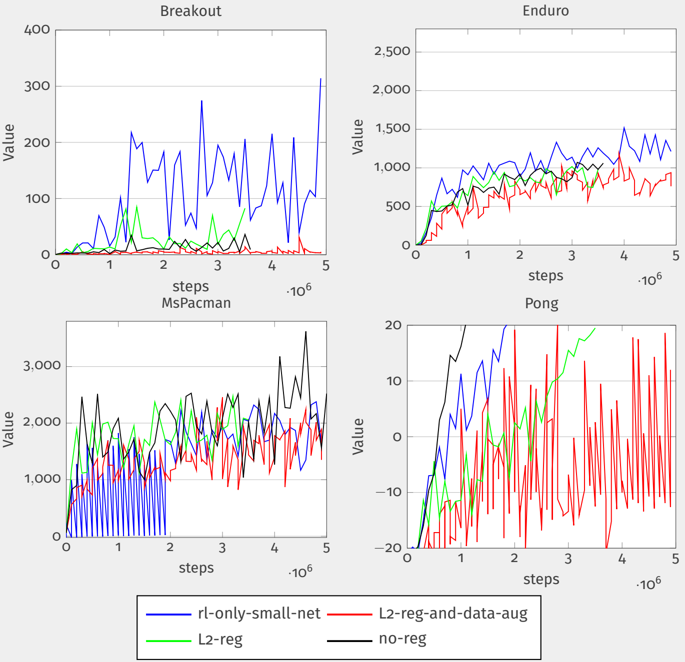


Effectiveness of parallel training
Breakout
Enduro
MsPacman
Pong
Value
steps
·10^6
0
1
2
3
4
5
100
200
300
400
500
1,000
1,500
2,000
2,500
3,000
20
10
−10
−20
rl-only-small-net
parallel-fs-256
parallel-fs-50

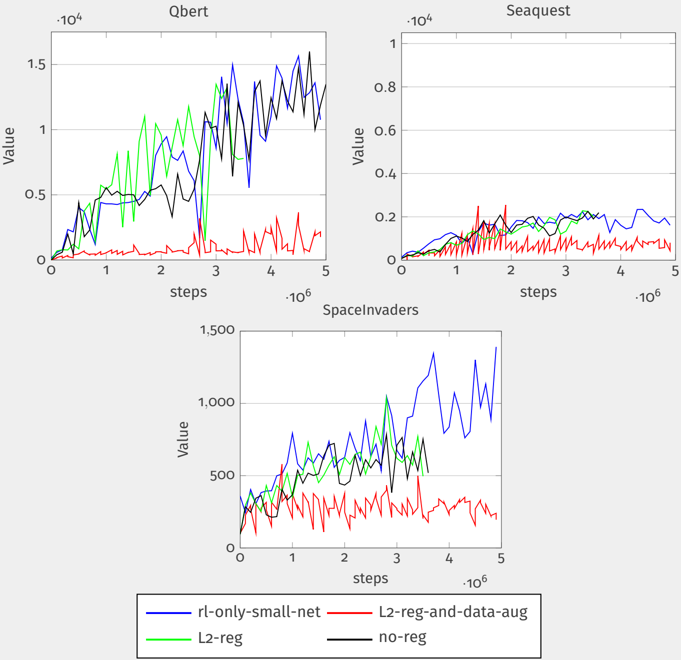
Effectiveness of parallel training — continued



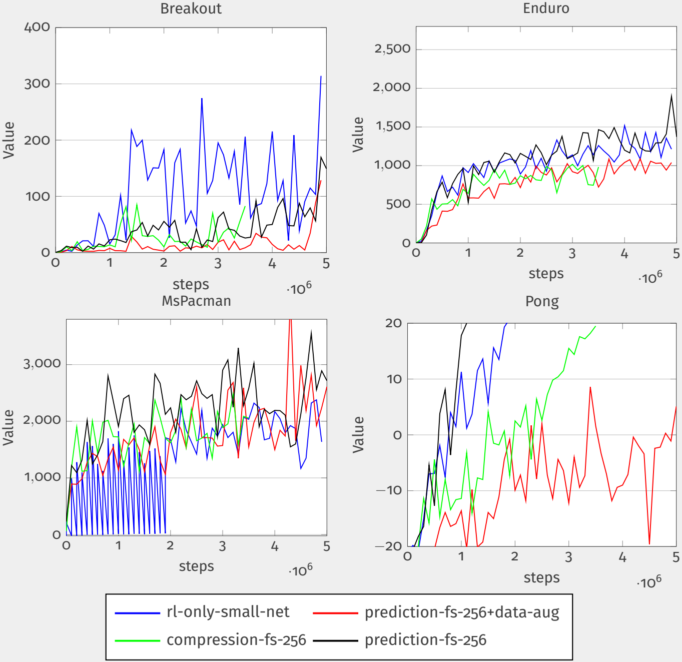
Effectiveness of regularization



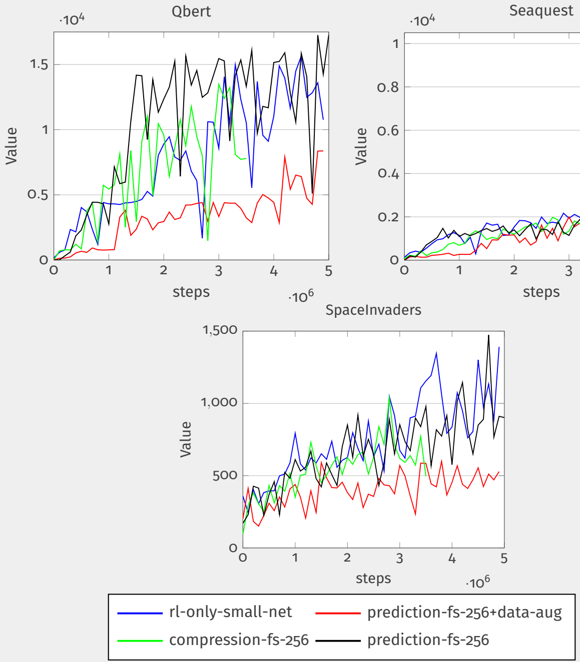
Effectiveness of regularization — continued



Effectiveness of predicting



Effectiveness of predicting — continued



Discussion

Key questions

- What are the differences between features and states and how important are they to the final performance?
- Why is reconstruction loss particularly bad at representing stateful information?
- What could be the characteristics of more successful approaches to using unsupervised learning for state representation learning?
- What role does regularization play in reinforcement learning, unsupervised learning and their combination?

- in Atari games states are primarily positions and velocities of objects

Representation learning methods do not learn this

Thus, to work as intended they should either be specialized to the problem, or truly made general across many problems.

- better understanding of what neural network features are would greatly help in designing them

Differences between features and states

- in Atari games states are primarily positions and velocities of objects

Representation learning methods do not learn this

Thus, to work as intended they should either be specialized to the problem, or truly made general across many problems.

- better understanding of what neural network features are would greatly help in designing them

Incentivising learning stateful information helps

Despite an order of magnitude large unsupervised loss (also destabilizes), forward prediction makes a difference.

Reconstruction loss is bad

Reconstruction loss is ill-suited for state representation learning

- MSE loss misses important details
- MSE loss learns unimportant details

Reconstruction loss is ill-suited for state representation learning

- MSE loss misses important details
- MSE loss learns unimportant details

Discriminative models are more promising

- they allow for better losses
- are less computationally expensive

Data augmentation hurt joint learning

Although it, interestingly, did not hurt either reinforcement nor representation learning individually.

Not all regularization is the same

Data augmentation hurt joint learning

Although it, interestingly, did not hurt either reinforcement nor representation learning individually.

L2 regularization provided a small benefit

Tested separately, it made the reconstruction features more stable, but that was not crucial.

Conclusion

Our approach did not work

Simply adding pixel reconstruction, and other naive approaches do not increase sample-efficiency.

Our approach did not work

Simply adding pixel reconstruction, and other naive approaches do not increase sample-efficiency.

Our implementation and takeaways help further research

The pull request will be made once code is cleaned up.

Thank you for your attention!