Increased Reinforcement Learning Performance through Transfer of Representation Learned by State Prediction Model

Alperen Tercan Charles W. Anderson, Senior Member, IEEE

Colorado State University

July 2021

Table of Contents

Deep Reinforcement Learning

- A promising approach for optimal control in unknown environments
- Learns optimal control from experiences accumulated from interactions with the environment
- Can achieve superhuman performance in certain tasks, eg. Atari games
- Suffers from high sample complexity.

Transfer Learning

- A potential way to solve sample complexity issues
- Exploits prior knowledge of a similar task or a similar environment
- A hot topic in RL community
- What if this is a novel task in an unfamiliar environment?

Our Work

Goal: Increase sample efficiency without relying on prior knowledge

Idea: Spend some of your time on learning dynamics and transfer the representations you've learned to Q-value prediction

Question: How can this be better than trying to learn Q-function from the beginning?

- Temporal Difference(TD) targets that we need to use are very biased at the beginning
- Bootstrapping from these biased targets can be very unstable in a function approximation setting

Table of Contents

Architecture

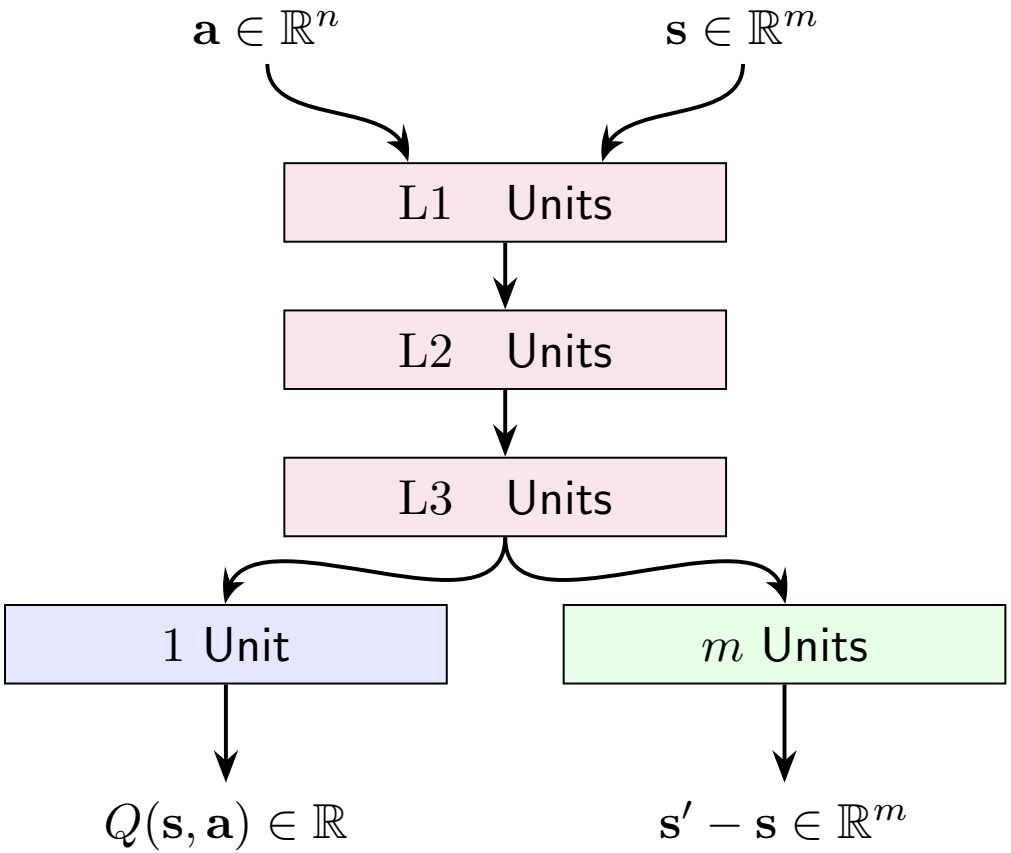

Figure: A four-layer example of the network architecture.

- Inputs:
  - State: $\mathrm{s} \in \mathbb{R}^m$
  - Action: $\mathrm{a} \in \mathbb{R}^n$
- Outputs:
  - State change: $\mathrm{s}' - \mathrm{s} \in \mathbb{R}^m$
  - Q-value: $Q(\mathrm{s}, \mathrm{a}) \in \mathbb{R}$
- Starts with shared layers
  - Parameters; θ^C
- Forks into two output heads
  - θ^S and θ^Q

How to train the network? - Loss Functions

1. Minimize TD-error:
   - Trains θ^C and θ^Q - Common layers and Q-head
   - Learns Q-function
2. Minimize State Change Error
   - Trains θ^C and θ^S - Common layers and state-chage head
   - Learns state dynamics
   - We will call this model prediction error.

Table of Contents

How to combine these? - The Schedules

Different schedules of using the loss functions give rise to different algorithms.

We consider two of them:

1. Pretraining
2. Simultaneous dual-training

Schedule 1: Pretraining

Out of K iterations of the whole training:

- Use only model prediction error in the first K_p iterations.
- Use only TD-error in the remaining $K - K_p$ iterations
- K and K_p are hyperparameters.

Schedule 2: Dual-training

At every training iteration:

- First, update θ^C and θ^Q to minimize TD-error.
- Then, update θ^C and θ^S to minimize model prediction error.

Using the schedules

These schedules can be applied on top of different RL algorithms and can be bundled with other RL "tricks".

To illustrate this, we apply our method to two different RL algorithms:

- Double DQN(DDQN): An enhanced version of Deep Q-learning.
- Twin Delayed DDPG(TD3): A state-of-the-art actor-critic algorithm.

Table of Contents

Used benchmarks

We have tested our approach in several settings:

With Double DQN:

- CartPole
- Acrobot
- MountainCar

With TD3:

- HalfCheetah
- Walker2D

For CartPole, we used our own implementation. Other DQN tasks are from OpenAI Gym and TD3 ones are from PyBullet.

Our CartPole Implementation

Original task:

- The pole starts upright.
- Goal is to keep it upright.

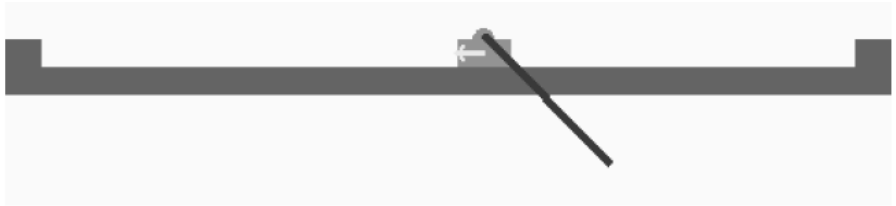

Figure: Our CartPole Implementation

Our Implementation

- The pole starts downright.
- Goal is to first swing it up and keep it there.
- Can utilize elastic collisions at the boundaries.

Our CartPole Implementation - Customizations

Parameterized state observability and reward sparsity in CartPole:

(Direct) Observability:

- **Fully Observable**: Cart position, cart velocity, pole angle, and pole angular velocity.
- **Hidden Cart Velocity:** Replaces cart velocity with the previous cart position. Requires approximating velocity from consecutive frames.
- **Hidden Cart and Pole Velocities:** Also replaces the pole angular velocity.

Our CartPole Implementation - Customizations

Reward Sparsity:

- Controlled via the number of consecutive upright steps necessary for a reward.
- Reward is 1 iff the pole was upright for the last k steps, 0 otherwise.
- State space augmented to include the number of consecutive upright steps, to keep the task Markovian.
- Experimented with $k = 1, 5, 15, 25$ steps.

Table of Contents

CartPole

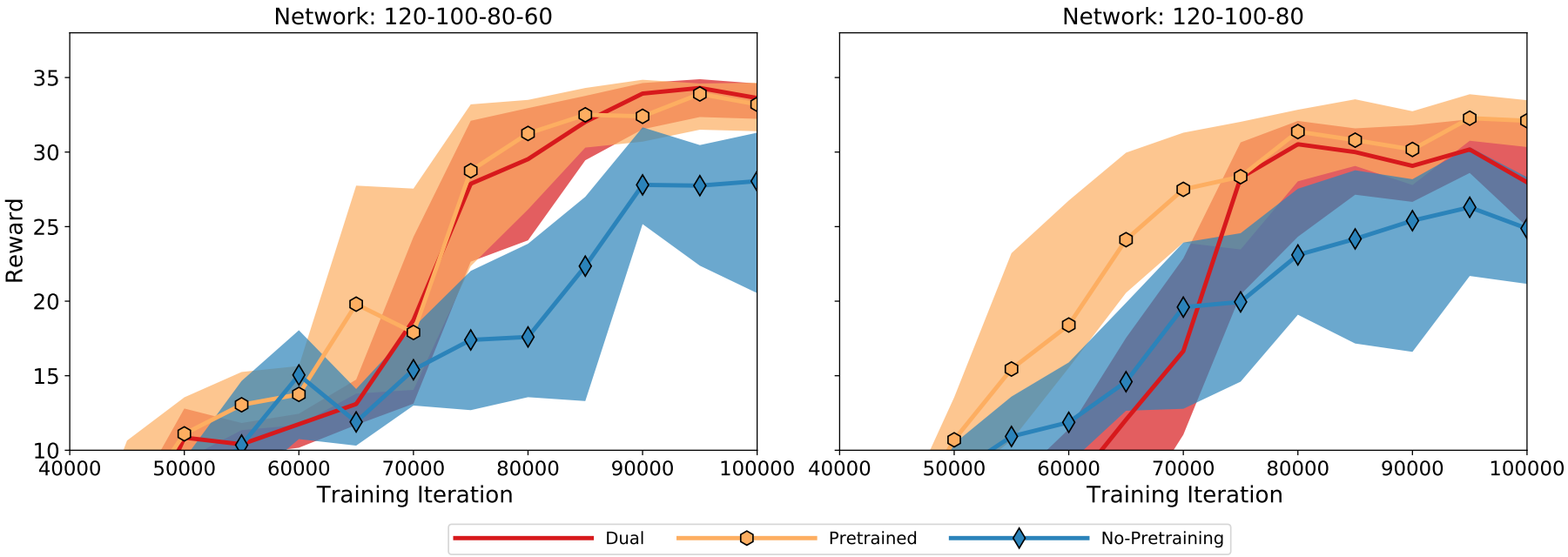

Figure: Learning curves for CartPole with Hidden Cart and Pole velocity with two different architectures. Median performance.

Walker2D

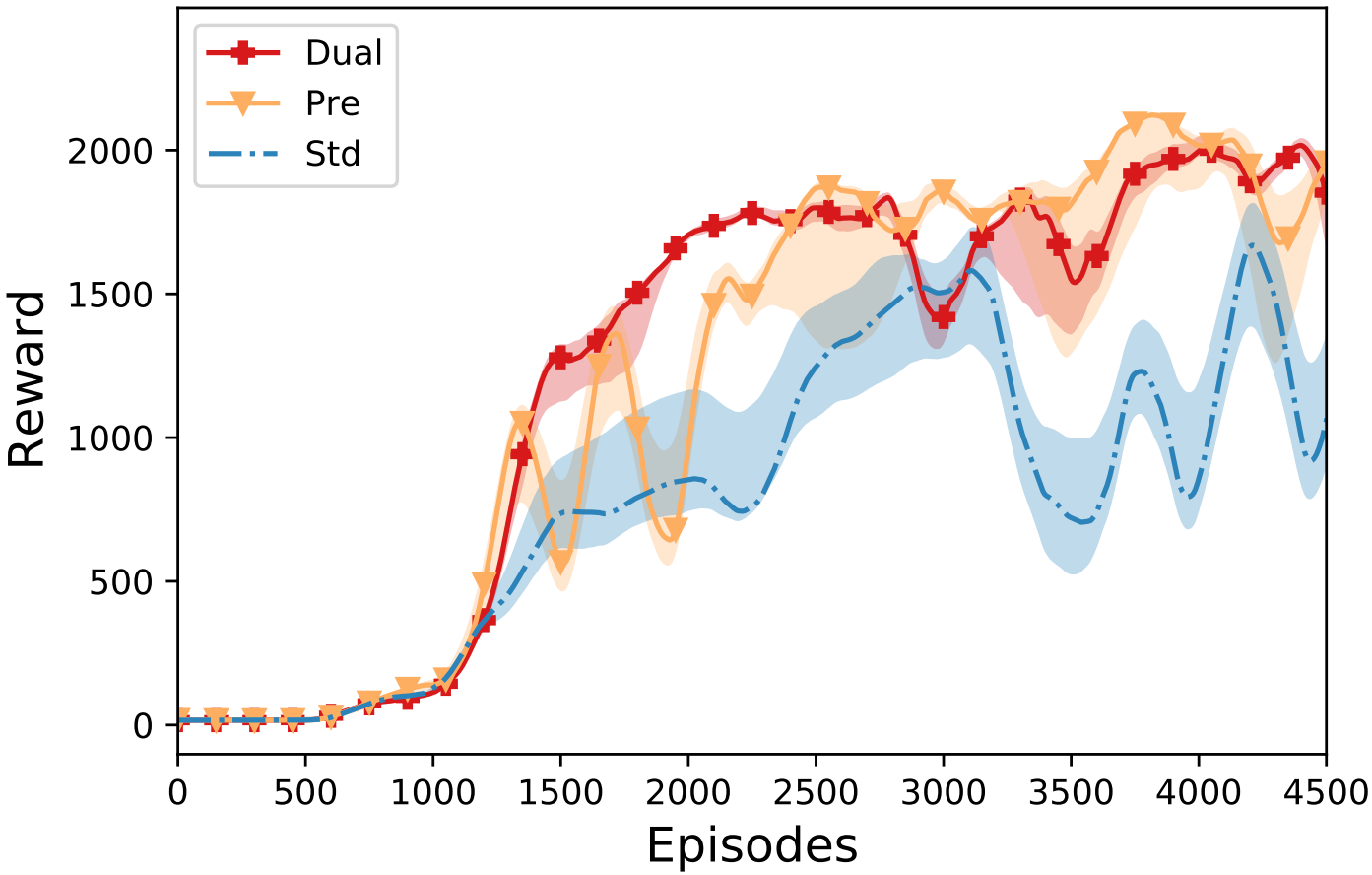

Figure: Comparison of learning curves for two variants with TD3 baseline. Median performance.

Impacts of Observability

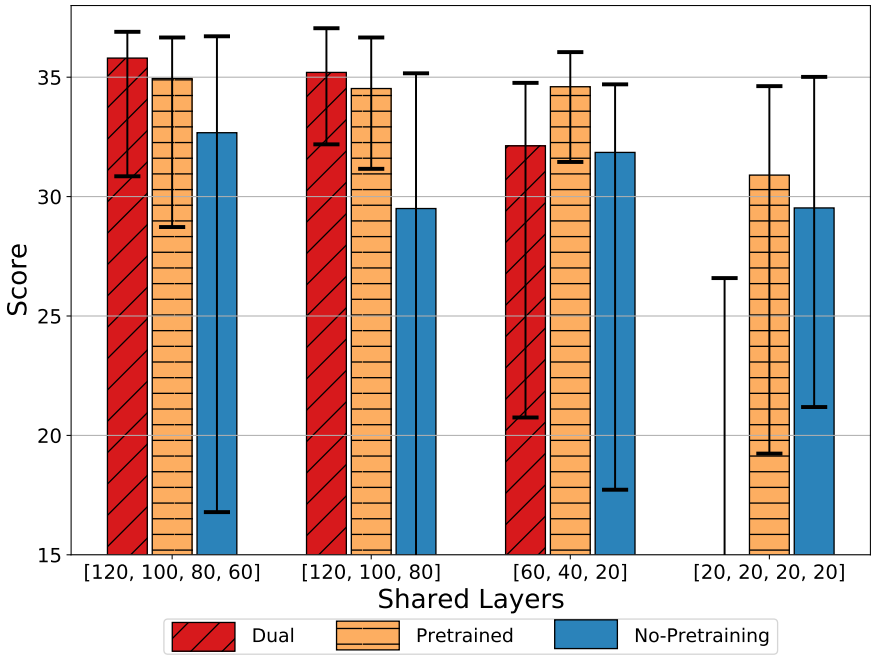

Figure: Only Cart Velocity Hidden

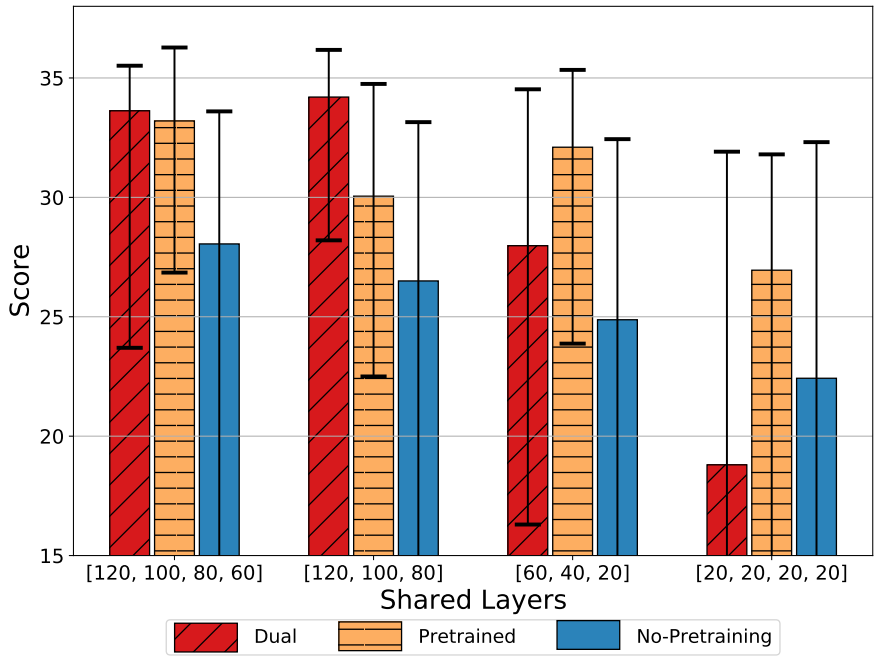

Figure: Both Cart and Pole Velocities Hidden

Less observability increases the benefits from using our method.

Impacts of Sparsity

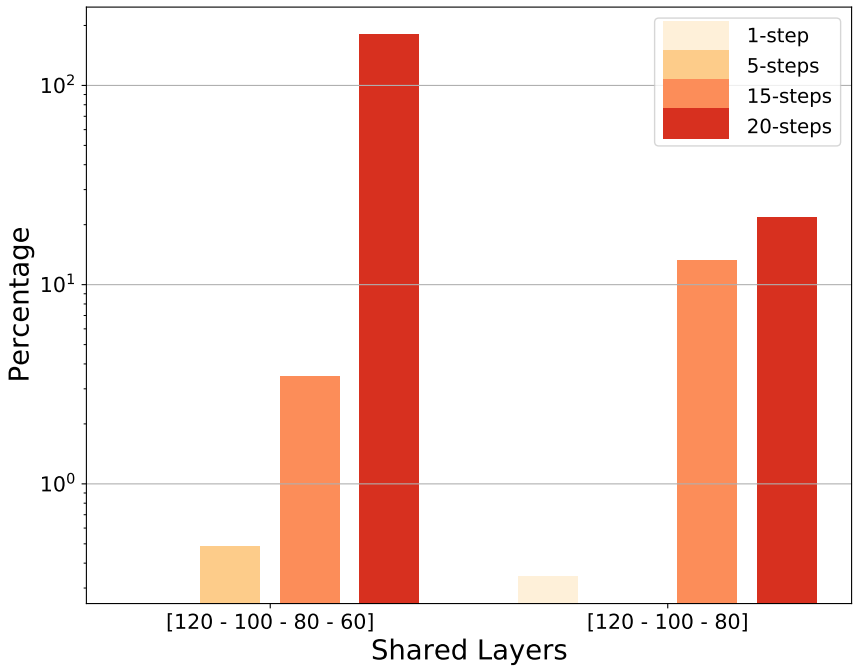

Figure: Comparison of relative gains from using dual-training in CartPole.

- Rewards given for keeping pole upright consecutive 1, 5, 15, 20 steps.
- When reward sparsity increases, relative gains increase too.
- Promising for harder problems.

Transfer from Value Function to Model Dynamics

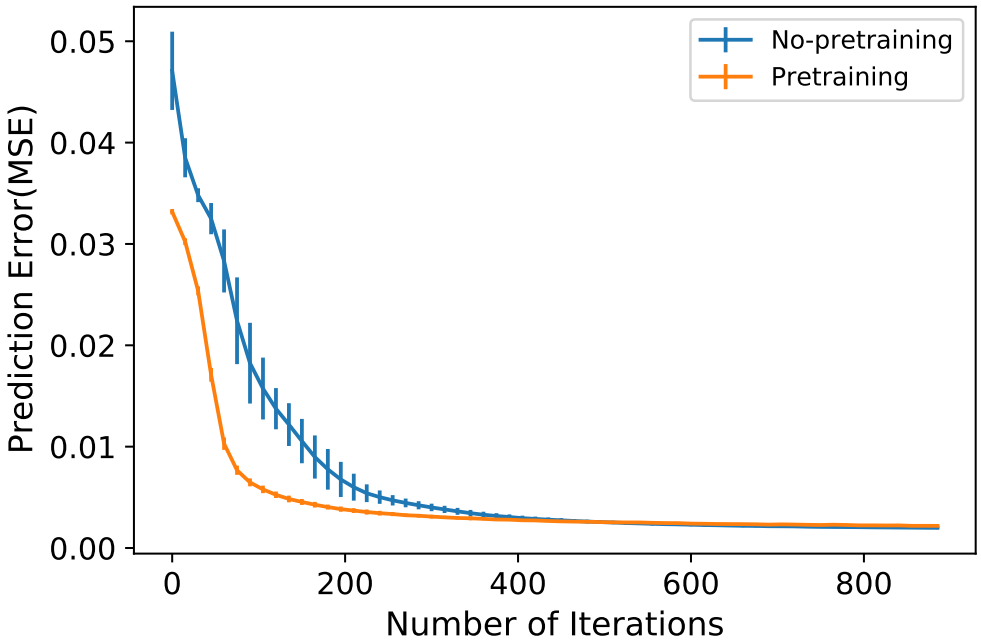

Figure: Supervised learning of state change prediction

- Blue is initialized randomly.
- Orange is initialized with weights of a Q-function learned in the same environment.
- Orange converges much faster.
- Further validates the effectiveness of sharing features.

Table of Contents

Key Points

- Transferring representations learned by state prediction to Q-value prediction is useful - expected
- **Pretraining:** Training the network for state prediction in the early iterations is better than training directly for Q-value prediction - less intuitive
- **Dual-training:** It is possible/faster to train large networks for both tasks simultaneously
- Benefits increase in **less observable** and **sparser** settings
- A simple approach that can be easily combined with methods

Future Work

- Apply this technique to new RL algorithms
- Test this in higher dimensional and more complex control tasks - we believe that the benefits will be amplified
- Investigate combinations with more common uses of learned models like generating virtual interactions

Thank You

Thank you for listening!