Learning State Correspondence of Reinforcement Learning Tasks for Knowledge Transfer

Marko Ruman[1*] and Tatiana V. Guy[1]

[1]Department of Adaptive Systems, Institute of Information Theory and Automation, The Czech Academy of Sciences, Pod Vodárenskou věží 4, Prague, 182 00, Czech Republic.

*Corresponding author(s). E-mail(s): ruman@utia.cas.cz;
Contributing authors: guy@utia.cas.cz;

Abstract

Deep reinforcement learning has shown an ability to achieve super-human performance in solving complex reinforcement learning tasks only from raw-pixels. However, it fails to reuse knowledge from previously learnt tasks to solve new, unseen ones. Generalizing and reusing knowledge are the fundamental requirements for creating a truly intelligent agent. This work focuses on transfer learning in reinforcement learning tasks. It offers a general method for one-to-one transfer learning based on generative adversarial network model tailored to reinforcement learning task.

Keywords: reinforcement learning, transfer learning, Markov decision process, deep learning

1 Introduction

The field of deep reinforcement learning has received much attention recently. Model-free methods for solving reinforcement learning (RL) tasks have shown ability to achieve super-human performance in Atari games, [23], [36], where the Deep Q-network (DQN) algorithm find optimal discrete actions. Moreover, they are successfully used also e.g. in MuJoCo environments with continuous actions, [19], or in the real-world robotic applications, [21]. Advances were made also in model-based deep RL methods such as AlphaZero, [28], or PlaNET, [12].

However, unlike humans, all the methods lack an important ability to *generalize* across different, but similar tasks. Humans have a remarkable ability to learn motor skills by mimicking behaviors of others. For example, developmental psychologists has shown that 18-month-old children are able to infer the intentions and imitate behaviors of adults, [22]. Imitation is not easy: children likely need to infer correspondences between their observations and their internal representations, which effectively aligns the two domains.

In machine learning, AI or robotics, using previously learnt knowledge in new, different, but related tasks is called *transfer learning* (TL), [24]. In the RL context, transfer learning is crucially important for developing agents with lifelong learning, [1], for simulation-to-real knowledge transfer used in robotics, [17], [13], [27], or for developing the general artificial intelligence, [5].

The problems of transfer learning in RL and especially deep RL contexts often stem from inability to generalize. For example, when deep learning was used with image data, it was shown that the output of deep convolutional network can be dramatically changed by 1-pixel perturbations of the input image, [29]. Naturally, as image data often form the observable states in RL tasks, this problem occurs also in RL domain, for instance 1-pixel perturbations lead to useless policies, [26]. The methods fail to reuse previously learnt knowledge even in very similar tasks, i.e. when the original image is rotated, or some colors have been changed. It was also shown that learning from scratch can be more optimal than fine-tuning the previously gained model, [8]. All that is in huge contrast with human abilities to generalize.

This work proposes a transfer learning approach with an emphasis on finding correspondence between *unpaired* data. It formalizes the problem of transfer learning in RL tasks, provides survey of state-of-the-art methods and offers a novel method for one-to-one transfer learning based on unsupervised data-record correspondence search.

Related works

Other approaches: Besides learning the task correspondence, some other approaches for transfer learning in deep RL are being researched. One research direction aims to learn general features of RL tasks transferable across multiple tasks. Work [4] showed that learning policies on so-called mid-level features can generalize better than learning policies directly on image observations. In [11] the authors learnt general features for two tasks with different dynamics, however, their method requires paired image observations which are hard to obtain in practice. Work [6] achieved success in tasks differing in reward functions by maintaining so-called successor features and by decoupling environment dynamics and reward function.

Correspondence search: The pioneering works that used task correspondence were based on unsupervised image-to-image translation models CycleGAN[1],

[1]Cycle generative adversarial network

[33], and UNIT[2], [20]. Work [8] achieved partial results by finding correspondence of states of two RL tasks. Their application potential is rather limited as problems like mode-collapse were present. Works [27] and [13] improved the approach by introducing learnt Q-function or object detection into the learning of the task correspondence. One of the most recent work, [32], introduces dynamics model in the learning. This approach is strongly influenced by the video-to-video translation model, [2].

Contributions

- The work introduces a method for *knowledge transfer between two similar RL tasks* based on RL-specific modification of CycleGAN.
- The work establishes a *corresponding function finding similarity between the data* from different RL tasks. The method is able to successfully transfer the knowledge between RL tasks while respecting their similarity.
- The work proposes a new loss function consisting of several meaningful components reflecting different type of losses. The modification *introduces Q-function and environment model into the learning.* It is shown that each component plays a significant role.
- The method is general and *does not require any extra task-specific* manual engineering. It works with RL tasks regardless of data format of states (e.g. images, sounds, numerical vectors, etc.).

Experiments with Atari game Pong demonstrated that the proposed method notably speeds up the learning and increases the average reward. The proposed approach is able to cope with the tasks for which standard approaches (GAN, CycleGAN) fail and the most efficient way remains to learn from scratch.

The paper layout is as follows: Section 2 recalls the required background and formulates the problem. Section 3 introduces the correspondence function and proposes a novel method of its learning. Section 4 describes experimental evaluation of the proposed method and compares it with baseline methods. Section 5 provides concluding remarks and outlines the future research directions.

2 Background and notation

This section briefly recalls RL formalism and introduces the considered problem.

Throughout the text sets are denoted by bold capital letters (e.g. $\mathbf{X}$), functions by capital letters (e.g. X), variables by lower-case letters (e.g. x). $\|x\|_1$ is the L1 norm of x. x_t is the value of x at discrete time instant $t \in \mathbf{N}$. $E_p[x]$ denotes expected value of x with respect to a probability density p (if provided). No notational distinction is made between a random variable and its realization.

[2] Unsupervised Image-to-Image Translation Networks

2.1 Reinforcement learning

Reinforcement learning (RL) considers an *agent* purposefully interacting with an *environment* by selecting actions. This scenario is typically formulated as a *Markov decision process* (MDP), [25].

Definition 1 (Markov decision process) MDP is a tuple

$$(\mathbf{S}, \mathbf{A}, T, R, \gamma), \tag{1}$$

where $\mathbf{S}$ is a set of environment states, $\mathbf{A}$ is a set of actions, $T(s_{t+1}, s_t, a_t)$ is a *transition function* from state $s_t \in \mathbf{S}$ to state $s_{t+1} \in \mathbf{S}$ after taking action $a_t \in \mathbf{A}$; $R(s_{t+1}, a_t, s_t)$ is a function that gives *reward* received by the transition $(s_t, a_t) \to s_{t+1}$; and $\gamma \in [0, 1]$ is a *discount factor*.

The set $\mathbf{S} \times \mathbf{A}$ will be referred to as a *task domain*.

At each time step t, the agent observes environment state $s_t \in \mathbf{S}$ and takes action $a_t \in \mathbf{A}$. Then the environment transits to a new state s_{t+1}[3] and the agent receives reward $r_t = R(s_{t+1}, a_t, s_t)$. The *goal* of the agent is to learn policy $\pi : \mathbf{S} \mapsto \mathbf{A}$ that maximizes the accumulated rewards. The solution of a MDP task is the *optimal* π^* such that:

$$\pi^* = \underset{\pi \in \boldsymbol{\pi}}{\arg\max}\, E\left[\sum_{t=1}^{N} \gamma^t R(s_{t+1}, \pi(s_t), s_t)\right], \tag{2}$$

where $N \in \mathbb{N} \cup \{\infty\}$ is decision horizon.

RL aims to train the agent to act optimally within MDP when either transition function or reward (or both) are unknown. Reasonable RL algorithm modifies π over time to gradually get it closer to an optimal policy. Majority of RL approaches are based on dynamic programming [25] and use Q-function (aka *action-value function*)

$$Q(s, a) = E_{\pi^*}\left[\sum_{t=1}^{N} \gamma^t R_t(s_{t+1}, \pi^*(s_t), s_t) \mid s_1 = s, a_1 = a\right]. \tag{3}$$

Q-function measures the expected future discounted reward the agent can obtain by taking an action in a given state.

Equation (3) *evaluates* policy π^*, i.e. gives the expected value of future discounted reward over the states induced by π^* for given starting state s and action a.

The estimate of Q-function (3) for s and a, denoted as $\hat{Q}(s, a)$, can be gradually learnt on the stream of data records (s_t, a_t, r_t, s_{t+1}) using for instance temporal difference learning, [31]:

$$\hat{Q}_{t+1}(s_t, a_t) = (1 - \alpha)\hat{Q}_t(s_t, a_t) + \alpha(r_t + \gamma \max_{a \in \mathbf{A}} \hat{Q}_t(s_{t+1}, a)), \tag{4}$$

[3] observable by the agent

Algorithm 1 Q-learning

Input: initial Q-function $Q_0(s,a)$, learning rate $\alpha \in (0,1)$, discount factor $\gamma \in (0,1)$

```
get the initial state s_0
for t = 1, 2, ..., till convergence do
    select action a_t = argmax_{a∈A} Q(s_t, a)
    get next state s_{t+1} and reward r_t
    if s_t is terminal then
        target = r_t
    else
        target = r_t + γ max_{a∈A} Q(s_{t+1}, a)
    end if
    Q_new(s_t, a_t) = (1 − α)Q(s_t, a_t) + α(target)
end for
```

Output: Q-function $Q(s,a)$

where $\alpha \in (0,1)$ is a parameter called *learning rate*, $\gamma \in (0,1)$ is a *discount factor* and $r_t = R(s_{t+1}, a_t, s_t)$. The learning starts with initial estimate of the Q-function $\hat{Q}_0(s,a)$. The learned optimal decision rule is then

$$\pi^*\left(s \mid \hat{Q}\right) = \operatorname*{argmax}_{a\in\mathbf{A}} \hat{Q}(s,a). \tag{5}$$

2.2 Deep Q-learning

Whenever the state space is huge, for instance when state is given by video frame, efficient learning of Q-function calls for approximation. The state-of-the-art in function approximation points deep neural networks as a suitable methodology, [7]. Deep Q-Networks (DQN) use a standard off-policy Q-learning algorithm [31] (see Algorithm 1) and deep neural networks to estimate the Q-function (3).

DQN approximates Q-function by a deep neural network with parameters, which can be trained similarly to the supervised learning, [23]. However, in the supervised learning, i. i. d. input data are assumed. Moreover, output values are expected to be the same for the same inputs, [9]. Neither of these assumptions is met in RL tasks. The consecutive states (e.g. video frames) are usually highly correlated and thus very far from being i. i. d. Also, output values contain learned Q-function which evolves during the learning. This makes the learning process unstable. To enable data reuse and stabilize the learning, DQN uses an *experience replay technique* to remove correlations in the observed sequences, and employs additional *target network* to stabilize the output values, see [23] for details.

Experience replay considers that the last n_M data records (so-called *experience memory*, denoted as $\mathbf{M}$) are stored in a memory buffer. At each learning step, mini-batch of data of length $n_B \in \mathbb{N}$ is randomly sampled from the memory buffer and used for updating the neural network that approximates Q-function. This makes the learning data closer to being i. i. d.

Target network is an additional network serving for stabilizing the learning. The idea is as follows. The parameters of the *original network* are updated at every learning step, while *target network* is used to retrieve output values and stays static, i.e. its parameters do not change in every learning step. Every $n_U \in \mathbb{N}$ steps, parameters of the original network are copied to the parameters of the target network, i.e. the original and target networks are synchronised.

The DQN algorithm is summarized in Algorithm 2. θ denotes parameters of the original network and θ^T are parameters of the target network.

Algorithm 2 DQN

Input: initial parameters θ of the Q-function $Q(s, a, \theta)$, learning rate α, discount factor γ, exploration rate ϵ, size of the experience memory n_M, size of the learning mini-batch n_B, number of steps for target network synchronization n_U

Initialize the experience memory to the size n_M
Set the parameters θ^T of the target network as $\theta^T = \theta$
for $t = 1, 2, ...,$ till convergence **do**
  With probability ϵ perform random action a_t otherwise select action $a_t = \operatorname{argmax}_{a \in \mathbf{A}} Q(s, a \mid \theta)$
  Get next state s_{t+1} and receive reward r_t
  Check if the memory is full. If so, remove the oldest data record
  Store (s_t, a_t, r_t, s_{t+1}) in the experience memory $\mathbf{M}$
  Sample random mini-batch of size n_B $(s_j, a_j, r_j, s_{j+1})_{j \in Random(n_B)} \in \mathbf{M}$
  for every j **do**
    if s_{j+1} is terminal **then**
      $\text{target}_j = r_j$
    else
      $\text{target}_j = r_j + \gamma \max_{a' \in \mathbf{A}} Q(s_{j+1}, a' \mid \theta^T)$
    end if
  end for
  Perform a gradient descent step on $\left(\left(\text{target}_j - Q(s_j, a_j \mid \theta)\right)^2\right)_{j \in Random(n_B)}$ with Huber loss, [14], with respect to parameters θ
  Every n_U steps set $\theta^T = \theta$
end for

Output: Q-function $Q(s, a)$, experience memory $\mathbf{M}$

2.3 Problem description

Assume two RL tasks on the same or similar environment. Generally the tasks are related in some way, while may differ in state/action spaces, reward function, environment dynamics, prior knowledge, etc. It is expected that experience gained during solving one RL task will help to improve learning performance in another RL task.

Let us consider one to be a *source task* with the domain $\mathbf{S}_S \times \mathbf{A}_S$ and another be a *target task* with the domain $\mathbf{S}_T \times \mathbf{A}_T$.

Our aim is to improve the results of target task learning via transfer of knowledge from the source task. Clearly the success of the transfer will depend on how similar these tasks are. Section 3 introduces correspondence function that reflects their similarity and proposes how to learn it.

3 Transfer learning: the correspondence function

3.1 Problem definition

Consider a *source task* and similar, but somehow different, *target task*. The source task has been solved in the past by DQN algorithm (see Section 2.2). The obtained knowledge $\mathbf{K}_S = (Q_S, \mathbf{M}_S)$ consists of learned Q-function Q_S and experience memory $\mathbf{M}_S = ((s_t, a_t, s_{t+1}, r_t)_{i=1}^{n_M})$.

Now the agent needs to solve the target task. First the agent can apply a random decision rule to the target task and collect experience memory $\mathbf{M}_T$. Our aim is to use knowledge $\mathbf{K}_S$ gained by solving the source task to leverage learning the target task.

Let there is a correspondence function $\mathcal{C}$ that establishes a pairwise connection between similar instances of the target and source tasks. It implies that for each state-action pair (s_T, a_T) from the target task function $\mathcal{C}$ gives a corresponding state-action pair (s_S, a_S) from the source task, i.e. $(s_S, a_S) = \mathcal{C}(s_T, a_T)$. Intuitively: these pairs should represent the same or very similar "agent-environment" state.[4] Then Q-function Q_T of the target task can be written using Q_S and correspondence function $\mathcal{C}$ as follows:

$$Q_T(s_T, a_T) = Q_S(\mathcal{C}(s_T, a_T)), \quad \forall (s_T, a_T) \in (\mathbf{S}_T \times \mathbf{A}_T)\,. \tag{6}$$

Hence the correspondence function can be used to transform state-action pairs from the target task to the source task. Thus the agent can use Q-function Q_S from the source task to choose the optimal policy of the target task. It is assumed that Q-function Q_T (6) solves (at least partially) the target task. Note that the correspondence function can be unknown to the agent. The next section describes how to learn the it.

[4] For example pulling break levers (a_T) when seeing an animal on the road (s_T) while driving a motorcycle corresponds to pressing the brake pedal (a_S) when seeing a person crossing the road (s_S) while driving a car.

3.2 Correspondence function learning

Let us assume (for brevity) that the action spaces of source and target task are identical, i.e. $\mathbf{A}_S = \mathbf{A}_T$, and that the mutually corresponding actions are found by an identity mapping independently of the current state. Thus, we only need to learn a mapping that finds the corresponding states, i.e. correspondence function for states.

The proposed learning is inspired by CycleGAN, [33], and extends it by employing the environment dynamics and the optimal strategy for the correspondence function learning. The approach is applicable for knowledge transfer between the related RL tasks.

CycleGAN is based on GAN[5], [10], and was developed for learning image-to-image transformations in an unsupervised way. The similar situation occurs here - there are two experience memories $\mathbf{M}_S$ and $\mathbf{M}_T$ with mutually unpaired entries. The task is to learn the correspondence function $\mathcal{C}$ that will help to match them.

In CycleGAN, two mappings G_S and G_T called *generators* are learnt,

$$G_S:\ \mathbf{S}_S \to \mathbf{S}_T \quad \text{and} \quad G_T:\ \mathbf{S}_T \to \mathbf{S}_S. \tag{7}$$

They are learnt as two GANs, thus generators G_S and G_T are learned simultaneously with respective discriminators D_S and D_T. The generators learn to map the states from $\mathbf{S}_S$ to $\mathbf{S}_T$ and vice-versa, while the discriminators learn to *distinguish* a real state from a state mapped by a generator.

Mappings G_S, G_T, D_S and D_T are constructed as neural networks with architecture depending on the data format, for instance for image states convolutional layers are often used. Learning in CycleGAN minimizes a loss consisting of two parts. The first part of the loss, $\mathcal{L}_{GAN}$, comes from GAN and it is given by the adversarial loss for generators G_S, G_T and their respective discriminators D_S, D_T as follows:

$$\begin{aligned}\mathcal{L}_{GAN} =& E_{p_{s_S}}\left[\log D_S(s_S)\right] + E_{p_{s_T}}\left[\log\left(1 - D_S\left(G_S\left(s_T\right)\right)\right)\right] \\ &+ E_{p_{s_T}}\left[\log D_T(s_T)\right] + E_{p_{s_S}}\left[\log\left(1 - D_T\left(G_T\left(s_S\right)\right)\right)\right],\end{aligned} \tag{8}$$

The adversarial training can learn mappings G_S and G_T (7) to produce outputs undistinguishable from respective sets $\mathbf{S}_S$ and $\mathbf{S}_T$. However, a network can learn to map the same set of input images to any random permutation of images in the target domain while minimizing the adversarial loss $\mathcal{L}_{GAN}$ at the same time. This part of the loss alone thus does not guarantee finding the desired correspondence function even if its global minimum is found.

To further reduce the search space of possible mappings G_S and G_T, CycleGAN introduces a so-called cycle-consistency requirement: every state $s_S \in \mathbf{S}_S$ must be recoverable after mapping it to $\mathbf{S}_T$, i.e. $G_T(G_S(s_S)) \approx s_S$. The same

[5]Generative adversarial network

requirement applies to every state $s_T \in \mathbf{S}_T$. This cycle-consistency requirement can be expressed by minimising the second part of the loss, $\mathcal{L}_{Cyc}$:

$$\mathcal{L}_{Cyc} = E_{p_{s_S}} \left[\| G_T \left(G_S \left(s_S \right) \right) - s_S \|_1 \right] + E_{p_{s_T}} \left[\| G_S \left(G_T \left(s_T \right) \right) - s_T \|_1 \right] \tag{9}$$

(9) is referred to as *cycle-consistency* loss, [33]:

3.2.1 Relation to RL

Even direct application of CycleGAN to the states brought some success in policy transfer, see for instance [8]. However, data records in experience memories comprise additional (richer) yet unused information that may be helpful for transfer of knowledge between tasks.

Respecting this information by including additional components into the loss function that is minimized in CycleGAN learning will make the learned correspondence even more relevant to RL tasks.

This work proposes to add other two components of the loss, that reflect

- Q-function learned from the source task,
- Environment model of the source task that reflect source task dynamics.

3.2.2 Q loss

The available Q-function, Q_S, should be incorporated in learning of the correspondence $\mathcal{C}$ as it determines the optimal policy. The *cycle-consistency* loss (9) forces generators G_S and G_T to be inverses of each other. By requiring that, the values of Q-function of state s_S and the state mapped to the other domain and back, $G_T(G_S(s_S))$, are the same. So, it is assumed that the learning of the generators will focus on the states that are relevant for the Q-function. Thus we propose to introduce the loss $\mathcal{L}_Q$ associated with the Q-function in the following form:

$$\mathcal{L}_Q = E_{p_{s_S}} \left[\| Q_S \left(G_T \left(G_S \left(s_S \right) \right) \right) - Q_S \left(s_S \right) \|_1 \right] \tag{10}$$

The loss (10) will make the learned correspondence more suitable for transferring knowledge between RL tasks as it indirectly indicates similarity of the tasks goals.

3.2.3 Model loss

So far, all considered losses (8) - (10) are associated with state values. However, any RL task is dynamic and the time sequence of states is important. Consider states of the target and the source tasks at times t and $t+1$. If generator G_T ensures mapping $s_{T;t}$ on $s_{S;t+1}$ and generator G_S maps $s_{S;t+1}$ back to $s_{T;t}$, then losses $\mathcal{L}_{GAN}, \mathcal{L}_{Cyc}$ and $\mathcal{L}_Q$ (8) - (10) are minimal. However, it would not be of help for solving the target RL task.

Therefore, to reflect the underlying dynamics, it is also vital to consider the loss respecting environment model F of the source task:

$$\mathcal{L}_M = E_{p_{s_{T;t}a_{T;t}s_{T;t+1}}} \left[\left\| F\left(G_T\left(s_{T;t}\right), a_{T;t}\right) - G_T\left(s_{T;t+1}\right) \right\|_1 \right] \tag{11}$$

The correspondence function minimising (11) will respect the environment dynamics. In particular, whenever action $a_{T;t}$ is applied at state $s_{T;t}$ resulting in state $s_{T;t+1}$, the triplet $(\bar{s}_{S;t}, a_{T;t}, \bar{s}_{S;t+1})$ should be observable in the source task, where $\bar{s}_{S;t} = G_T\left(s_{T;t}\right)$ and $\bar{s}_{S;t+1} = G_T\left(s_{T;t+1}\right)$.

3.2.4 Final loss

The proposed loss comprises all the components (8), (9), (10) and (11) and has the following form:

$$\mathcal{L} = \mathcal{L}_{GAN} + \lambda_{Cyc}\mathcal{L}_{Cyc} + \lambda_Q\mathcal{L}_Q + \lambda_M\mathcal{L}_M, \tag{12}$$

where λ_{Cyc}, λ_Q and λ_M are *loss parameters* that define the relative influence (weight) of the respective component of the loss.

The searched correspondence function $\mathcal{C}$ is then obtained as follows:

$$\mathcal{C}(s_T, a_T) = (G_T(s_T), I(a_T)), \qquad \forall (s_T, a_T) \in \mathbf{S}_T \times \mathbf{A}_T, \tag{13}$$

where G_T is the generator from (7) mapping states from the *target task* to states from the *source task* and I is an identity mapping.

4 Experimental part

To test efficiency of the proposed approach, two experiments on Atari game Pong, [3], were conducted. Performance of the approach was evaluated based on an average accumulated reward per game. GAN and CycleGAN were used as baseline methods.

4.1 Domain description

Pong is a two-dimensional game simulating table tennis. There are 6 available actions (do nothing, fire, move up, move down, move up fast, move down fast) and the last 4 image frames served as a state. The agent learned to play the game using the DQN algorithm from Section 2.2 and, thus, learned the Q-function. To test the approach described in Section 3.2, the agent also learned environment model F, see Section 3.2.1.

4.2 Experiment description

The proposed method for TL was tested in two experiments.

Experiment 1: The source and target tasks are the same, i.e. game Pong (screenshot is shown on Fig. 1). The main aim of this experiment was to use the proposed approach to find the identity transformation.

Experiment 2: The source task was the original Pong while the target task was rotated Pong (see screenshot at Fig. 2). The game remained the same, but all of the image frames were all rotated by 90 degrees.

The states in the experiments were the last 4 frames (images) of the game.

Each experiment included the following steps:

1) The agent played the *source task* (standard Pong), learned the optimal policy by DQN and obtained the optimal Q-function Q_S, environment model F and experience memory $\mathbf{M}_S$ that consists of 10 000 data entries collected at the end of the game.
2) The agent played the *target task* (standard Pong in Experiment 1, rotated Pong in Experiment 2) using random actions and obtained experience memory $\mathbf{M}_T$ consisting of 10 000 data entries.
3) The agent started learning the correspondence function $\mathcal{C}$ from the *target task* to the *source task* using the method from Section 3 with the Q-function Q_S, environment model F and experience memories $\mathbf{M}_S$ and $\mathbf{M}_T$,
4) For every 1000 steps of correspondence function learning:
    - The agent played 5 games of the *target task* using the correspondence function that was being learned and the transformed Q-function from the *source task* $Q_S(\mathcal{C}(.,.))$ (6)
    - Average accumulated reward per game was computed.
5) The agent played the *target task* while using the learned correspondence[6] and Q-function transferred from the *source* task (while continuously fine-tuning the Q-function with fixed correspondence function using DQN).

The *key metric* to evaluate the success of the knowledge transfer was the average accumulated reward per game while playing the *target task* with the transformed $Q_S(\mathcal{C}(.,.))$ with learned correspondence function $\mathcal{C}$.

Baselines

The results are compared with two baselines - using GAN and CycleGAN methods, [10], [33], which brought some success for knowledge transfer in similar settings, [8].

Easy to see that GAN (CycleGAN) can be obtained from (12) for the loss parameters set to the following values:

- $\lambda_Q = \lambda_M = \lambda_{Cyc} = 0$ (for GAN),
- $\lambda_Q = \lambda_M = 0$ (for CycleGAN).

[6]the correspondence that achieved the best average accumulated reward per game in the previous step was used here

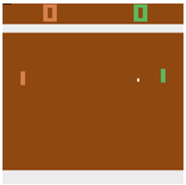


Fig. 1: **Standard Pong**, [3]

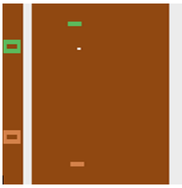


Fig. 2: **Rotated Pong by 90 degrees**, [3]

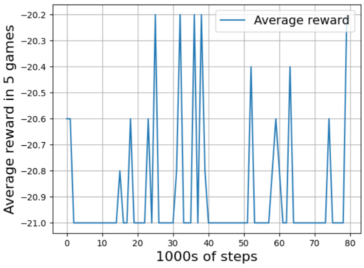
Thus the proposed method is more general and includes baselines as particular cases.

4.3 Experiment 1

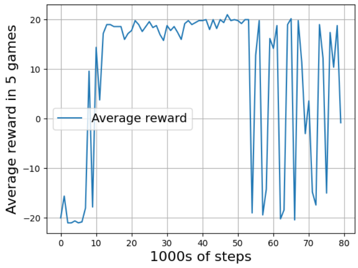
This experiment tested transfer learning when *source* and *target* tasks are the same.

The generators G_S and G_T (see Section 3.2) in this experiment were constructed as neural networks with convolutional layers. Their specific architecture was taken from [18]. The discriminators D_S and D_T were also constructed as neural networks with convolutional layers with the same architecture as in [15].

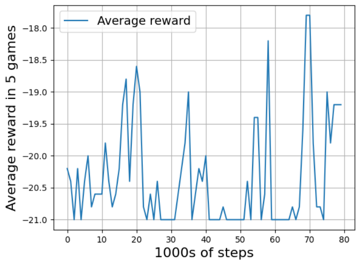
The parameters of all of the networks were initialized from Gaussian distribution $N(0, 0.02)$. The approach was tested for all the combinations of the parameters: $\lambda_{Cyc} \in \{0, 1, 10\}$, $\lambda_Q \in \{0, 1\}$ and $\lambda_M \in \{0, 1, 10\}$ (12).



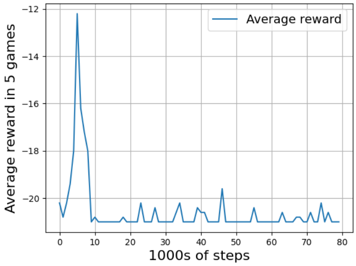
a) **GAN**
$\lambda_{Cyc} = \lambda_Q = \lambda_M = 0$



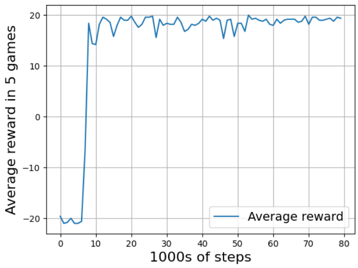
b) **CycleGAN**
$\lambda_{Cyc} = 10, \lambda_Q = \lambda_M = 0$



c) $\lambda_{Cyc} = 0,\ \lambda_Q = 1,\ \lambda_M = 0$



d) $\lambda_{Cyc} = 0,\ \lambda_Q = 0,\ \lambda_M = 10$



e) $\lambda_{Cyc} = 1,\ \lambda_Q = 1,\ \lambda_M = 1$

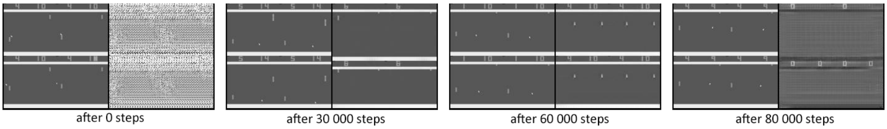
Fig. 3: **Experiment 1**: Average accumulated reward per game when playing 5 games with the transformed Q-function (6). The reward is computed each 1000 steps of correspondence function learning. The results are shown for different values of loss parameters λ_{Cyc}, λ_Q and λ_M. Fig. 3a) and 3b) show the **baselines** using *GAN* and *CycleGAN* methods.

The results are presented in Fig. 3 - Fig. 5.

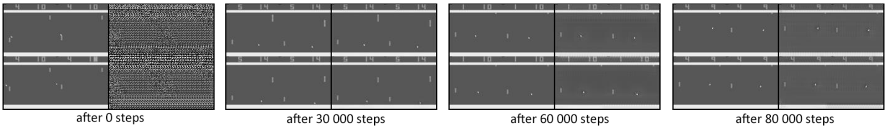
Fig. 3 shows the average reward gained in 5 games that the agent played each 1000 steps of the learning of correspondence function for different λ_{Cyc}, λ_Q and λ_M. The maximum possible reward in one game was 21 and Fig. 3 shows that the best results were achieved with settings involving all components of the loss $\lambda_{Cyc} = \lambda_Q = \lambda_M = 1$. *GAN baseline* (Fig. 3a) did not produce good result. *CycleGAN* performance (Fig. 3b) soon became unstable though it provided good reward at the beginning.

Fig. 4 shows the progress correspondence function learning. It confirms that the best results were achieved with all the loss components active, $\lambda_{Cyc} = \lambda_Q = \lambda_M = 1$ (12), while visually good results were achieved also for the CycleGAN baseline with $\lambda_{Cyc} = 10$, $\lambda_Q = \lambda_M = 0$ (although as Fig. 3 showed, the learned correspondence function oscillated between good and a useless one). In other settings no useful correspondence function has been learned.

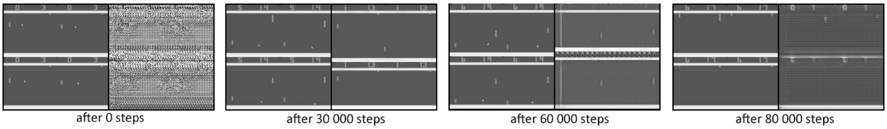
Lastly, Fig. 5 compares the performance of the agent learning to play *Pong* from scratch and the agent that learned the correspondence function and employed the previously learned Q-function. Fig. 5 displays a moving average of accumulated reward per game computed from the last 20 games. As Fig. 5 shows, when the target and source tasks are the same, the agent was able to fully re-use the previously acquired knowledge.



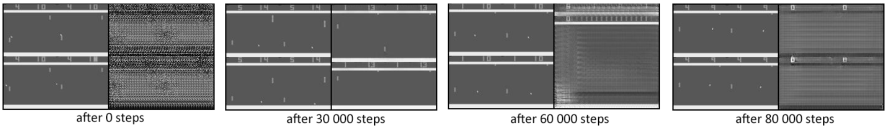
a) **GAN** $\lambda_{Cyc} = 0,\ \lambda_Q = 0,\ \lambda_M = 0$



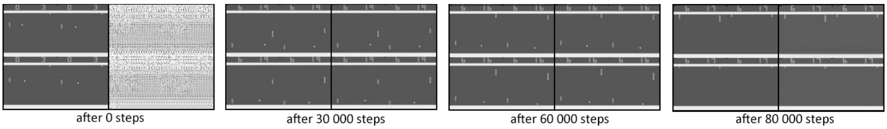
b) **CycleGAN** $\lambda_{Cyc} = 10, \lambda_Q = 0,\ \lambda_M = 0$



c) $\lambda_{Cyc} = 0,\ \lambda_Q = 1,\ \lambda_M = 0$



d) $\lambda_{Cyc} = 0, \lambda_Q = 0,\ \lambda_M = 10$



e) $\lambda_{Cyc} = 1, \lambda_Q = 1,\ \lambda_M = 1$

Fig. 4: **Experiment 1**: Progress of learning the correspondence function $\mathcal{C}$ (13) after 0, 20000, 40000 and 60000 steps). The results are shown for different values of loss parameters λ_{Cyc}, λ_Q and λ_M (12). The left parts of the pictures are frames serving as states of the *target* task, the right parts are the same states mapped by the learned correspondence function, $\mathcal{C}$.

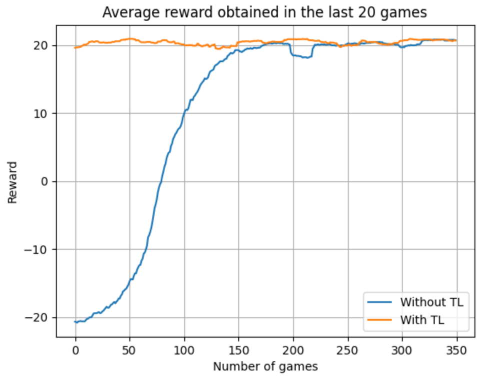


Fig. 5: Moving average of reward per game computed from the last 20 games depending on the number of played Pong games. Orange line denotes the case hen the agent learned from scratch, i.e. no TL was used. Blue line denotes the case with TL, i.e. when the agent learned the correspondence function and used the transformed Q-function (6). The agent was continuously learning the Q-function in both cases.

4.4 Experiment 2

In this Experiment, the *target task* is rotated Pong, i.e. the original Pong with image frames rotated by 90 degrees (see Fig. 2).

Generators G_S and G_T (7) (see Section 3.2) were constructed as neural networks. Two types of generators were tested in this experiment. The architecture of the first one, referred to here as **resnet generator**, was taken from [18] and followed by a rotation layer as in [16]. The second type, referred to as **rotation generator**, was composed of the mentioned rotation layer only.

Discriminators D_S and D_T were constructed, same as in the Experiment 1 by neural networks with convolutional layers with the same architecture as in [15].

The method was tested for various settings of the loss parameters λ_{Cyc}, λ_Q and λ_M (12). Settings providing the best performance are presented (Fig. 6 - Fig. 8) together with GAN and CycleGAN baselines.

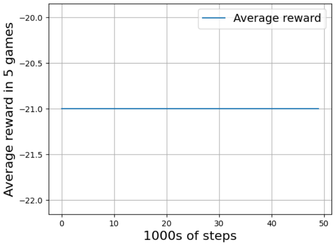
Fig. 6 shows the average reward gained in 5 games that the agent played each 1000 steps of the learning of correspondence function for both types of the generators and for the best settings of λ_{Cyc}, λ_Q and λ_M. The maximum possible reward in one game was 21 and Fig. 7 shows that the perfect correspondence was found with **rotation generator**. Learning of correspondence function with **resnet generator** showed also some success and the gained correspondence was subsequently used for fine-tuning the Q-function. At the same time, the GAN and CycleGAN baselines did not achieve reasonable result. Fig. 6a) and Fig. 6b) show that.

Fig. 7 shows the progress in correspondence function learning. It shows the successful learning in the case of **rotation generator** (Fig. 7a), however the correspondence learned in the case of **resnet generator** (Fig. 7b) also looked visually close to the optimal one (especially after 50 000 steps).

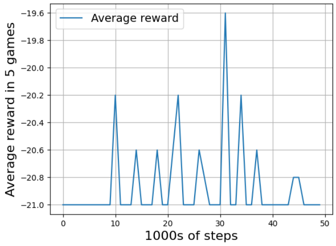
Lastly, Fig. 8 shows agent's performance when playing the *target task* - rotated Pong. Fig. 8 displays a moving average of accumulated reward per game computed from the last 20 games. Fig. 8 compares the agent's results in the target task in the following cases:

1. When the agent played the *target task* from scratch.
2. The agent learned the correspondence function $\mathcal{C}$ using resnet generator.
3. The agent learned the correspondence function using rotation generator.
4. The agent re-used the Q-function from the source task without respecting the correspondence.

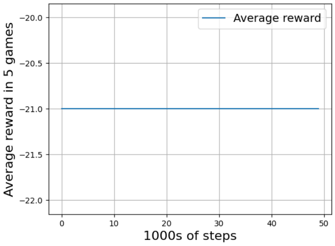
As Fig. 8 shows, the agent with rotation generator was able to fully reuse the previously gained knowledge. Use of resnet generator gave significantly better performance at the beginning compared with learning from scratch. Re-use of the Q-function gave worse results even compare to the case of learning from scratch.



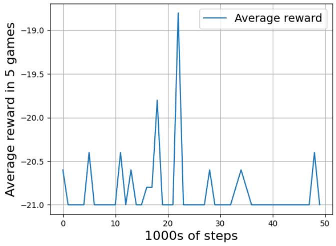
a) **Rotation generator** using **GAN**



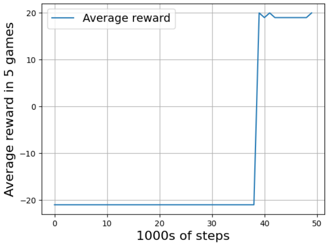
b) **Resnet generator** using **GAN**



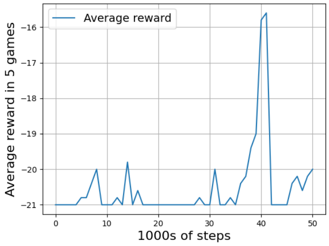
a) **Rotation generator** using **CycleGAN**



b) **Resnet generator** using **CycleGAN**



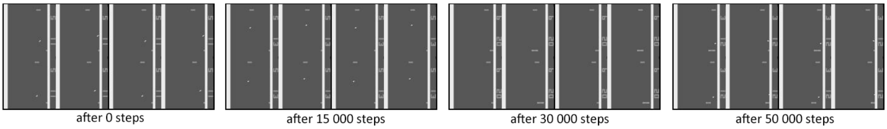
a) **Rotation generator** with $\lambda_{Cyc} = \lambda_Q = 0,\ \lambda_M = 10$



b) **Resnet generator** with $\lambda_{Cyc} = 1, \lambda_Q = 0,\ \lambda_M = 1$

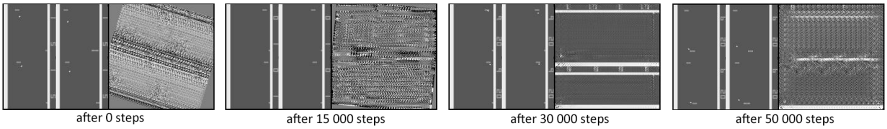
Fig. 6: Average accumulated reward in 5 games when playing Rotated Pong with the transformed Q-function (6). The reward is computed each 1000 steps of the learning of correspondence function. The results are shown for **rotation** and **resnet** generator with the best settings of the loss parameters in each case as well as with using **GAN** and **CycleGAN** baselines.

a) Rotation generator using **GAN**

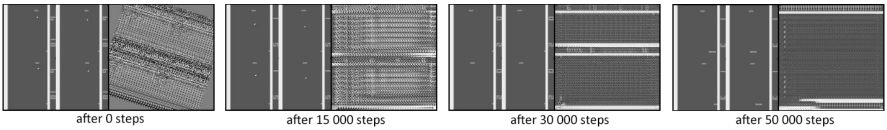


b) Rotation generator using **CycleGAN**

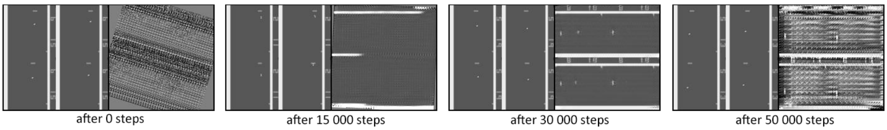
c) Rotation generator with $\lambda_{Cyc} = 0$, $\lambda_Q = 0$, $\lambda_M = 10$



d) Resnet generator using **GAN**



e) Resnet generator using **CycleGAN**



f) Resnet generator with $\lambda_{Cyc} = 1, \lambda_Q = 0$, $\lambda_M = 1$

Fig. 7: Progress in learning the correspondence function $\mathcal{C}$ (13). The results are shown for rotation and resnet generators with the best settings of the loss parameters λ_{Cyc}, λ_Q and λ_M (12) as well as with using **GAN** and **CycleGAN** baselines. The left parts of the pictures are frames representing the states of the *target* task, the right parts are the same states transformed by the correspondence function $\mathcal{C}$.

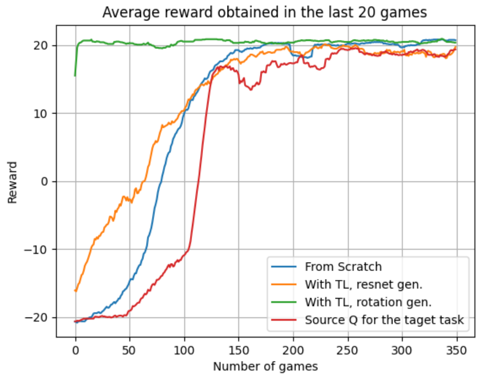


Fig. 8: Moving average of reward per game computed from the last 20 games depending on the number of played games for the game rotated Pong for four different agents - an agent learning the game from scratch (blue line), an agent using the correspondence function learned with resnet generator (orange line), an agent using the correspondence function learned with rotation generator (green line) and an agent reusing only the Q-function without any correspondence function (red line). The agents were continuously learning the Q-function.

5 Conclusion

The paper considers transfer learning in RL tasks. The work establishes a correspondence function indicating similarity of the *source* and *target task*. The work proposes a novel correspondence function that respects RL elements like Q-function and transfers model of the source task into learning. It gives the agent an ability to gradually create a set of skills, adapt and use them while interacting with the dynamically changing environment.

The method was evaluated on simulated experiments involving 2-D Atari game Pong and compared against two baselines using GAN and CycleGAN method. Experiment 1 with identical source and target tasks served verification of the proposed method. It was shown that the method succeeded and learned the tasks correspondence. It also illustrated importance of all proposed loss components as TL was the most successful when all the components were respected. Comparing to baselines shows that using the loss relevant only to images, TL was either unsuccessful (using only GAN loss) or unstable (using GAN and Cycle-consistency loss).

In Experiment 2, the target task was different from the source task. Two types of network architecture for generators in the learning of correspondence function were tested. It was shown that choice of the architecture is crucial for achieving good results. The generator composed of rotation and convolutional layers yielded worse performance (but still usable) compare to the generator with only rotation layer. It shows that the convolutional layers traditionally used in image processing may not be good for this scenario. Thus other architectures like transformers, [30], should be investigated.

The proposed method was successful to learn the mapping between the states from the *source task* and the states from the *target task* in both of the experiments. Subsequently, TL was successful and the agent was able to play the target task very well from the very beginning.

Many open problems remain unsolved, in particular: i) transfer learning for tasks having low similarity, ii) using several source tasks, iii) choosing only relevant source tasks similarly to [37], iv) find a better network architecture for the correspondence function learning, v) testing the method on a real-world example.

Data availability statement: The datasets generated during and/or analysed during the current study are available from the corresponding author on reasonable request.

Ethical approval: This article does not contain any studies with human participants performed by any of the authors.

References

[1] Haitham Bou Ammar, Eric Eaton, José Marcio Luna, and Paul Ruvolo. Autonomous cross-domain knowledge transfer in lifelong policy gradient reinforcement learning. In *Twenty-fourth international joint conference on artificial intelligence*, 2015.

[2] Aayush Bansal, Shugao Ma, Deva Ramanan, and Yaser Sheikh. Recycle-gan: Unsupervised video retargeting. In *Proceedings of the European conference on computer vision (ECCV)*, pages 119–135, 2018.

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

[4] Bryan Chen, Alexander Sax, Gene Lewis, Iro Armeni, Silvio Savarese, Amir Zamir, Jitendra Malik, and Lerrel Pinto. Robust policies via mid-level visual representations: An experimental study in manipulation and navigation. *arXiv preprint arXiv:2011.06698*, 2020.

[5] Jeff Clune. Ai-gas: Ai-generating algorithms, an alternate paradigm for producing general artificial intelligence. *arXiv preprint arXiv:1905.10985*, 2019.

[6] André Barreto, Will Dabney, Rémi Munos, Jonathan J Hunt, Tom Schaul, Hado P Hasselt and David Silver Successor features for transfer in reinforcement learning. *Advances in neural information processing systems*, 30, 2017

[7] Balázs Csanád Csáji et al. Approximation with artificial neural networks. *Faculty of Sciences, Etvs Lornd University, Hungary*, 24(48):7, 2001.

[8] Shani Gamrian and Yoav Goldberg. Transfer learning for related reinforcement learning tasks via image-to-image translation. In *International Conference on Machine Learning*, pages 2063–2072. PMLR, 2019.

[9] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[10] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014.

[11] Abhishek Gupta, Coline Devin, YuXuan Liu, Pieter Abbeel, and Sergey Levine. Learning invariant feature spaces to transfer skills with reinforcement learning. *arXiv preprint arXiv:1703.02949*, 2017.

[12] Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. In *International conference on machine learning*, pages 2555–2565. PMLR, 2019.

[13] Daniel Ho, Kanishka Rao, Zhuo Xu, Eric Jang, Mohi Khansari, and Yunfei Bai. Retinagan: An object-aware approach to sim-to-real transfer. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 10920–10926. IEEE, 2021.

[14] Peter J Huber. Robust estimation of a location parameter. In *Breakthroughs in statistics*, pages 492–518. Springer, 1992.

[15] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1125–1134, 2017.

[16] Max Jaderberg, Karen Simonyan, Andrew Zisserman, et al. Spatial transformer networks. *Advances in neural information processing systems*, 28, 2015.

[17] Stephen James, Paul Wohlhart, Mrinal Kalakrishnan, Dmitry Kalashnikov, Alex Irpan, Julian Ibarz, Sergey Levine, Raia Hadsell, and Konstantinos Bousmalis. Sim-to-real via sim-to-sim: Data-efficient robotic grasping via randomized-to-canonical adaptation networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12627–12637, 2019.

[18] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *European conference on computer vision*, pages 694–711. Springer, 2016.

[19] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[20] Ming-Yu Liu, Thomas Breuel, and Jan Kautz. Unsupervised image-to-image translation networks. *Advances in neural information processing systems*, 30, 2017.

[21] A Rupam Mahmood, Dmytro Korenkevych, Gautham Vasan, William Ma, and James Bergstra. Benchmarking reinforcement learning algorithms on real-world robots. In *Conference on robot learning*, pages 561–591. PMLR, 2018.

[22] Andrew N Meltzoff. Understanding the intentions of others: re-enactment of intended acts by 18-month-old children. *Developmental psychology*, 31(5):838, 1995.

[23] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[24] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2009.

[25] Martin L Puterman. Markov decision processes. *Handbooks in operations research and management science*, 2:331–434, 1990.

[26] Xinghua Qu, Zhu Sun, Yew-Soon Ong, Abhishek Gupta, and Pengfei Wei. Minimalistic attacks: How little it takes to fool deep reinforcement learning policies. *IEEE Transactions on Cognitive and Developmental Systems*, 13(4):806–817, 2020.

[27] Kanishka Rao, Chris Harris, Alex Irpan, Sergey Levine, Julian Ibarz, and Mohi Khansari. Rl-cyclegan: Reinforcement learning aware simulation-to-real. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11157–11166, 2020.

[28] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.

[29] Jiawei Su, Danilo Vasconcellos Vargas, and Kouichi Sakurai. One pixel attack for fooling deep neural networks. *IEEE Transactions on Evolutionary Computation*, 23(5):828–841, 2019.

[30] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[31] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3):279–292, 1992.

[32] Qiang Zhang, Tete Xiao, Alexei A Efros, Lerrel Pinto, and Xiaolong Wang. Learning cross-domain correspondence for control with dynamics cycle-consistency. *arXiv preprint arXiv:2012.09811*, 2020.

[33] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Proceedings of the IEEE international conference on computer vision*, pages 2223–2232, 2017.

[34] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[35] Sebastian Ruder. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1609.04747*, 2016.

[36] Yueyue Hu, Shiliang Sun, Xin Xu and Jing Zhao. International Journal of Machine Learning and Cybernetics. *International Journal of Machine Learning and Cybernetics*, 11(11), 2020, pages 2461–2474.

[37] Marzieh Davoodabadi Farahani and Nasser Mozayani. Evaluating skills in hierarchical reinforcement learning. *International Journal of Machine Learning and Cybernetics*, 11(10), 2020, pages 2407–2420

6 Appendix - implementation details

6.1 Network architectures

CycleGAN generator architectures

The architectures of generators G_S and G_T in the experiment 1 (Section 4.3) and the resnet generators G_S and G_T in the experiment 2 (Section 4.4) were taken from [33]. The 9 residual blocks version was used. Below, we follow the naming convention used in the mentioned work.

Let `c7s1-f` denote a 7×7 Convolution-BatchNorm-ReLU layer with f filters and stride 1. `df` denotes a 3×3 Convolution-BatchNorm-ReLU layer with f filters and stride 2. Reflection padding was used to reduce artefacts.

`Rf` denotes a residual block that contains two 3×3 con- volutional layers with the same number of filters (f) on both layers. `uf` denotes a 3×3 fractional-strided-Convolution- BatchNorm-ReLU layer with f filters and stride 1 2.

The network architecture consisted of:

`c7s1-64,d128,d256,R256,R256,R256,R256,R256,R256,R256,R256, R256,u128,u64,c7s1-3`

Discriminator architectures

For discriminator networks D_S and D_T in all the experiments, 70×70 Patch-GAN was used, [15]. Let `Cf` denote a $4{\times}4$ Convolution-BatchNorm-LeakyReLU layer with f filters and stride 2. After the last layer, a convolution to produce a 1-dimensional output was used. Leaky ReLUs were used with a slope of 0.2.

The discriminator architecture was:

`C64,C128,C256,C512`

Q-function architecture

Q-function had architecture taken from [23]. Let `c-k-s-f` denote a $k \times k$ Convolution-ReLU layer with stride s and f filters and `f-o` is a Fully connected-ReLU layer with o outputs.

The Q-function architecture was:

`c-8-1-32,c-4-2-64,c-3-1-64,f-512,f-6`

Environment model architecture

The environment model F had the same architecture as the generators G_S and G_T with one difference - the fifth residual block received one-hot encoded actions as an additional input.

The architecture of environment model was then as follows:

`c7s1-64,d128,d256,R256,R256,R256,R256,R262,R262,R262,R262, R262,u128,u64,c7s1-3`

6.2 Training

All the networks are trained from scratch with weights are initialized from a Gaussian distribution $N(0, 0.02)$.

The environment model F was trained with Adam optimizer, [34], with learning rate of 0.001, batch size of 16 and it was trained for 50 epochs.

For the training of Q-function, RMSprop optimizer, [35], was used, the learning rate was set to 0.0001 and batch size was set to 32. The other parameters of Q-learning were identical as in [23].

Generators G_S and G_T and discriminators D_S and D_T were jointly trained using Adam optimizer with initial learning rate of 0.0002 which was linearly decayed to zero, the training took 4 epochs.