# Transfer learning in Reinforcement learning tasks - learning the task correspondence

Marko Ruman

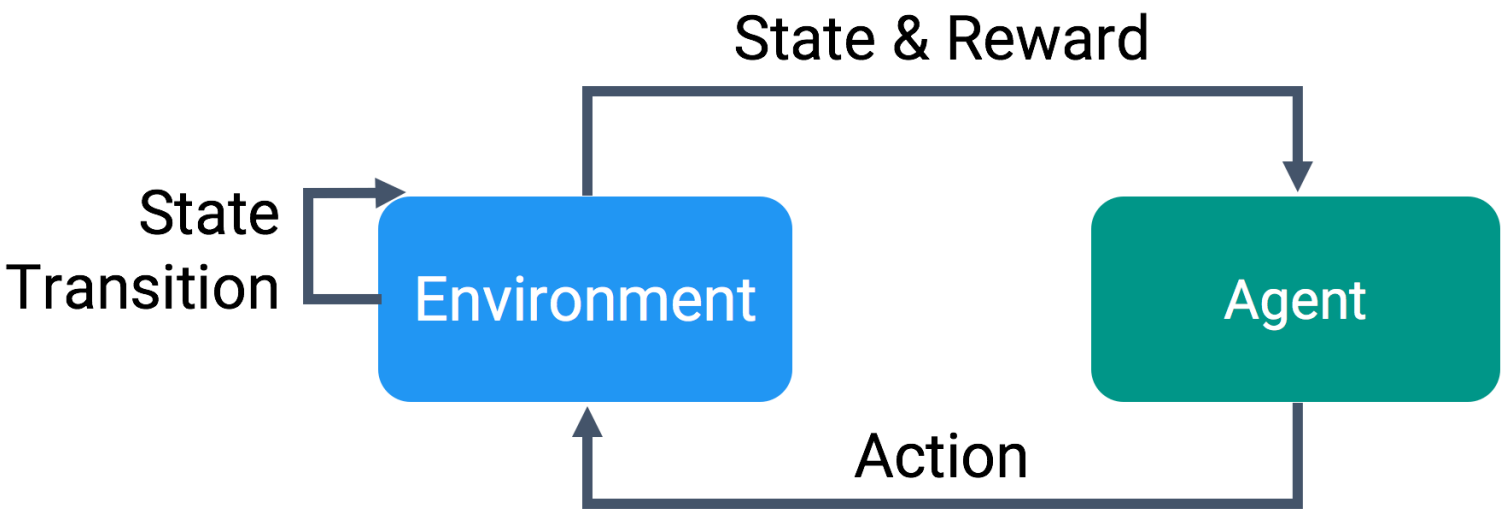
**supervisor**: Ing. Tatiana Valentine Guy, Ph.D.

# Outline

1. What is reinfocement learning (RL)?
2. Deep Q-learning
3. Knowledge transfer between tasks - transfer learning (TL)
4. Proposed TL method for RL
5. Initial results and conclusion

# What is *Reinforcement learning* (RL)?

An agent interacts with an environment through actions and receives reward.



# RL and Markov decision processes (MDP)

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

- **S** - state space,
- **A** - action space,
- $T = T(s_{t+1}, s_t, a_t)$ - transition function,
- $R = R(s_{t+1}, a_t, s_t)$ - reward function,
- $\gamma$ - discount factor

Functions $R$ a $T$ are **unknown**.
The goal is to find the optimal decision rule $\pi^*$, i.e.

$$\pi^* = \underset{\pi}{argmax}\, E\left[\sum_t \gamma^t R(s_{t+1}, \pi(s_t), s_t)\right] \tag{2}$$

# Q-learning

- $Q$-function gives value of the pair $(s_t, a_t)$,

$$Q(s,a) = E_{\pi^*}\left[\sum_k \gamma^k R_{t+k+1}(\cdot)|s_t = s, a_t = a\right] \quad (3)$$

- $Q$-function is learnt using the temporal difference error:

$$Q^{new}(s_t,a_t) = (1-\alpha)Q^{old}(s_t,a_t) + \alpha(r_t + \gamma \max_{a\in\mathbf{A}} Q^{old}(s_{t+1},a)) \quad (4)$$

$r_t$ is the received value of reward at the time $t$

# Deep Q-learning

- A method for tasks with large state space **S**
  - e.g. images as states
- $Q$-function is approximated by a neural network
- The learning uses:
  - **Experience memory** - the network is not learned directly on the stream of new data, but we sample it from finite *experience memory* $M$ in order to be closer to i.i.d
  - **Target network** - one network is learned on the background, another one (its copy) network is used to choose actions and their parameters are synchronized each $N_S$ steps
- Effective way of solving individual RL tasks, however, often fails even when the task is slightly changed (e.g. only couple of pixels are different)

# Transfer learning (TL) between RL tasks

*"If you know how to drive a car, you will learn to drive a motorcycle very fast"*

TL is an ability to reuse knowledge gained in the past RL tasks to solve new RL task more efficiently.

**Motivation**

- Lifelong learning
- Simulation-to-real
- General intelligence

# The TL

$\mathbf{S}_1, \mathbf{A}_1$ - source
$\mathbf{S}_2, \mathbf{A}_2$ - target

- **source** and **target** tasks are assumed
- The agent solved the **source** task and has:
  - $Q$-function $Q_1(s, a)$,
  - Experience memory $\mathbf{M}_1$.
- The agent has experience memory $\mathbf{M}_2$ from **target** task acquired by initial interaction using random decision rule.

The goal is to learn a *correspondence function* $\mathcal{C} : \mathbf{S}_2 \times \mathbf{A}_2 \mapsto \mathbf{S}_1 \times \mathbf{A}_1$ so that the transformed $Q$-function

$$Q_1\left(\mathcal{C}(s, a)\right)$$

describes the **target** task.

# CycleGAN for RL tasks

- Unsupervised method for learning transformation between two sets.
- Can be directly used on experience memories - $M_1$, $M_2$,
- Two mappings are learned simultaneously

$$G_1 : \mathbf{S}_1 \mapsto \mathbf{S}_2, \qquad G_2 : \mathbf{S}_2 \mapsto \mathbf{S}_1$$

- The mappings $G_1$ a $G_2$ are learned as GAN models with extra Cycle-consistency error:

$$\mathcal{L}_{Cyc} = \|G_2(G_1(s)) - s\| + \|G_1(G_2(\bar{s})) - \bar{s}\|$$

where $s \in M_1$, $\bar{s} \in M_2$.

# CycleGAN is not tailored for RL tasks

- CycleGAN can be applied directly on individual states, but RL tasks provide more information then which states were present in the task and how they look like.
- The agent learned the $Q$-function $Q_1$ and is able to pick optimal actions.
- The experience memory $M$ contains tuples $(s_t, a_t, s_{t+1})$ that give information about transitions from state $s_t$ to $s_{t+1}$ depending on the action $a_t$.
- The aim of the study was to make the TL method **specific for RL** and, thus, include the extra information from RL tasks in the learning of $\mathcal{C}$.

# Proposed TL method in RL

To make the learned correspondence $\mathcal{C}$ more relevant to RL, two extra errors were introduced

$$\mathcal{L}_Q = \|Q_1\left(G_2\left(G_1(s)\right)\right) - Q_1(s)\|$$

$$\mathcal{L}_M = \|F\left(G_2\left(s_t\right), a_t\right)) - G_2(s_{t+1})\|$$

where $s, s_t, s_{t+1}, a_t \in M_1$,
$F$ is *environment model* learned on $M_1$.

The loss for learning $G_1$ and $G_2$ is the as follows:

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

$\lambda_i$ - error weights

# The method was tested in Pong game

## The first case

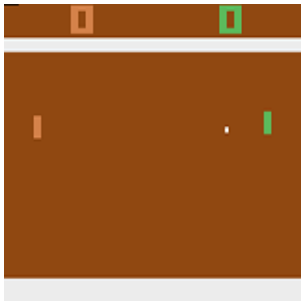


Figure: **source task**

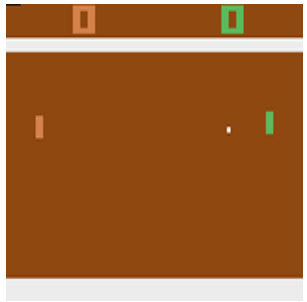


Figure: **target task**

## The second case (rotated)

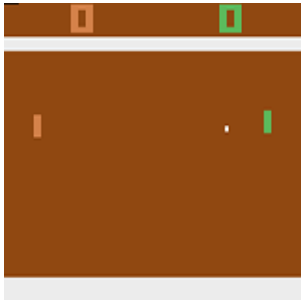


Figure: **source task**

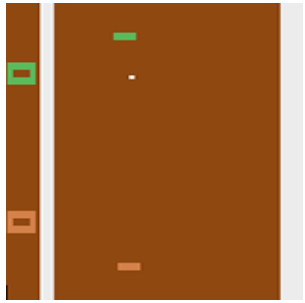


Figure: **target task**

# The first case

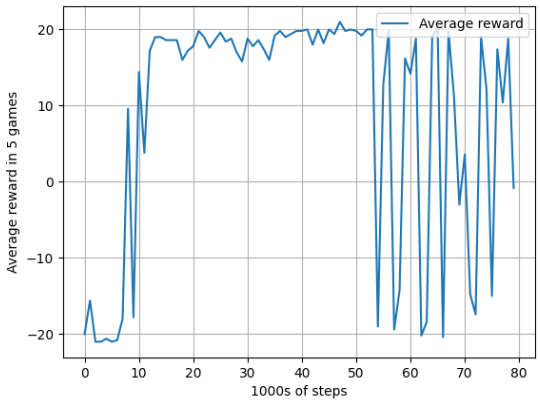
The **source** and **target** tasks are the same.

The method was tested for several error weights:

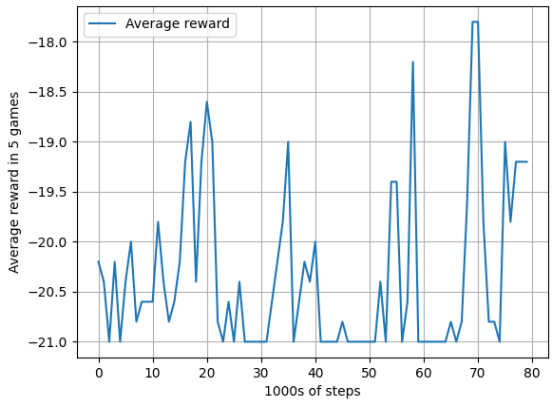
- $\lambda_{Cyc} \in \{0, 1, 10\}$
- $\lambda_Q \in \{0, 1\}$
- $\lambda_M \in \{0, 1, 10\}$

After each 1000 steps of correspondence learning, the agent played 5 games of the target task with transformed $Q$-function and the **average reward** was evaluated.

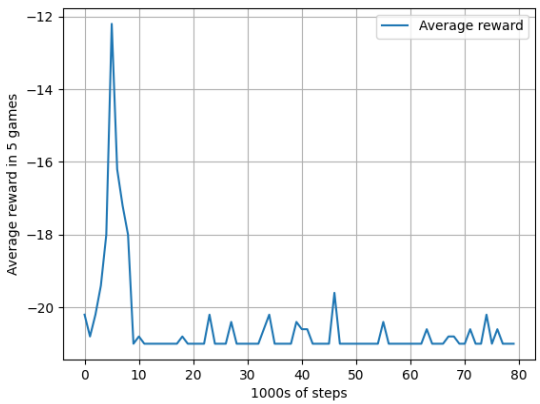
# The first case - results - average reward



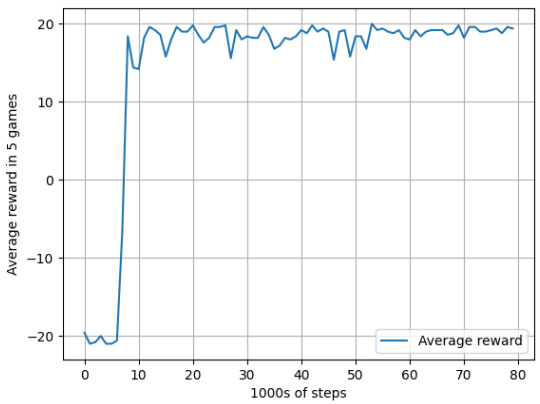

$\lambda_{Cyc} = 10, \lambda_Q = 0, \lambda_M = 0$




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




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




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

# The first case - results - learning of $\mathcal{C}$

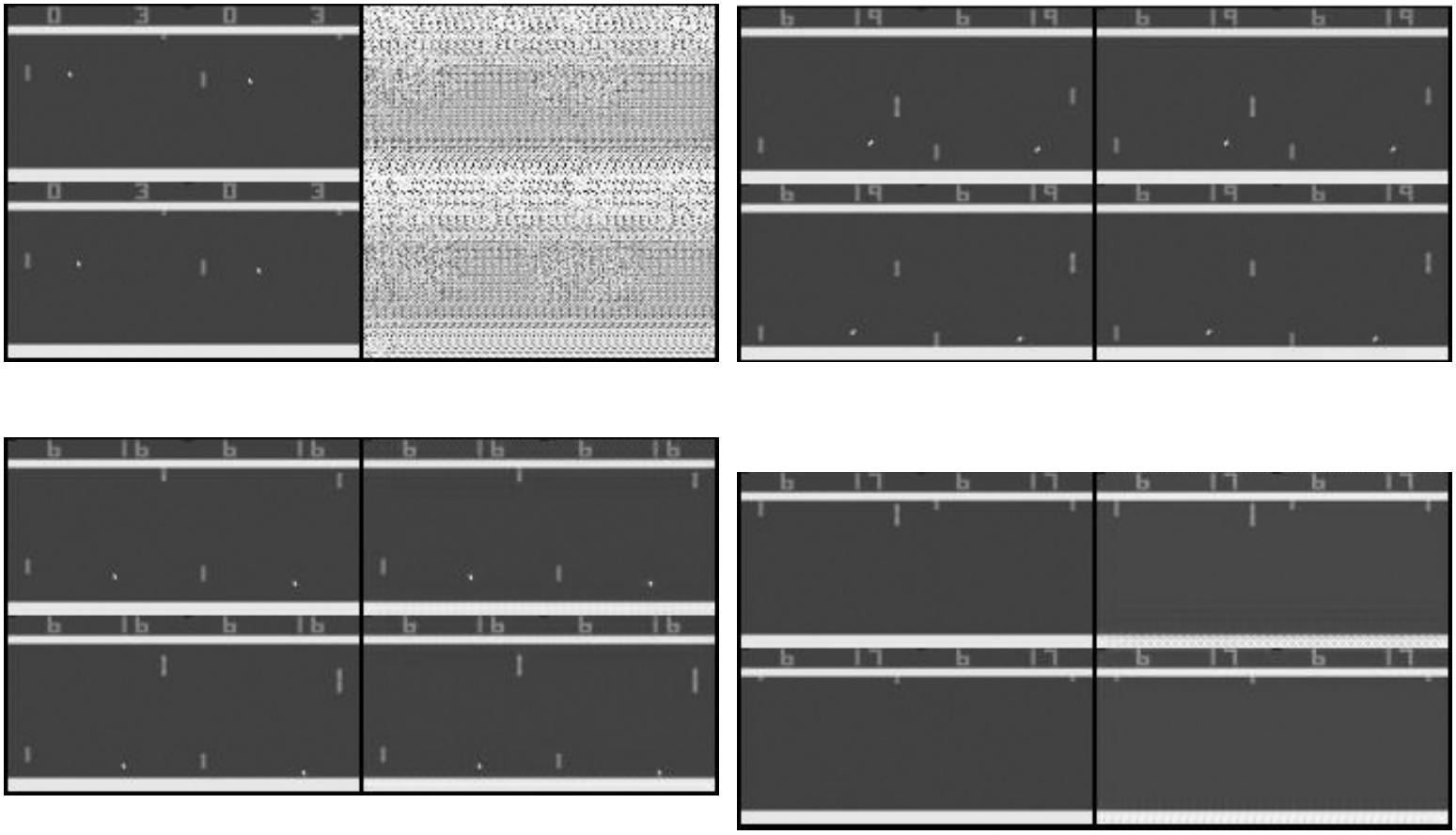


Figure: Learning of the correspondence function $\mathcal{C}$ after 0, 30000, 60000 and 80000 iterations for $\lambda_{Cyc} = 1$, $\lambda_Q = 1$, $\lambda_M = 1$

# The first case - results - learning of $\mathcal{C}$

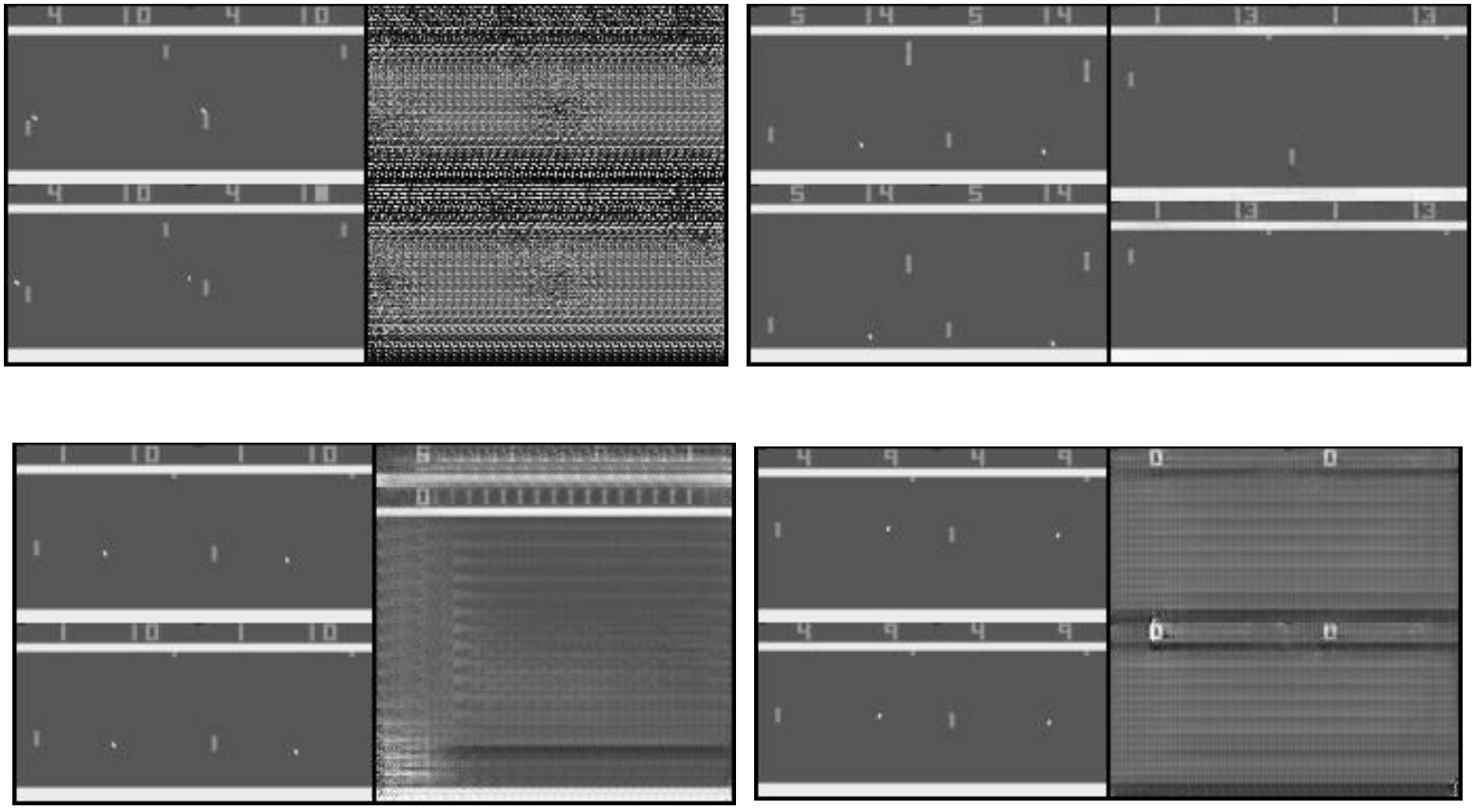


Figure: Learning of the correspondence $\mathcal{C}$ after 0, 30000, 60000 and 80000 iterations for $\lambda_{Cyc} = 0$, $\lambda_Q = 0$, $\lambda_M = 10$

# The second case

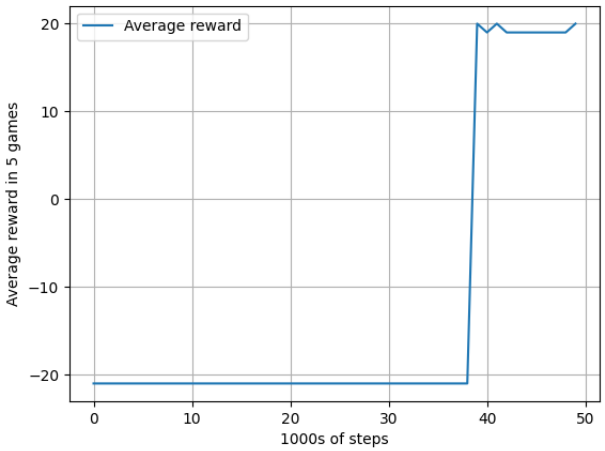
**Source** task is normal *Pong*, **target** task is *Rotated Pong*.

The method was tested two types of architecture of the mappings $G_1$, $G_2$:

- Resnet with rotation layer
- Only rotation layer

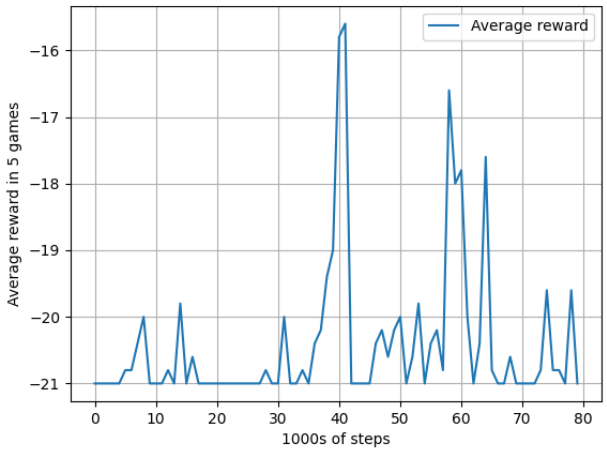
After each 1000 steps of correspondence learning, the agent played 5 games of the target task with transformed $Q$-function and the **average reward** was being evaluated.

# The second case - results - average reward




**Rotation** architecture with

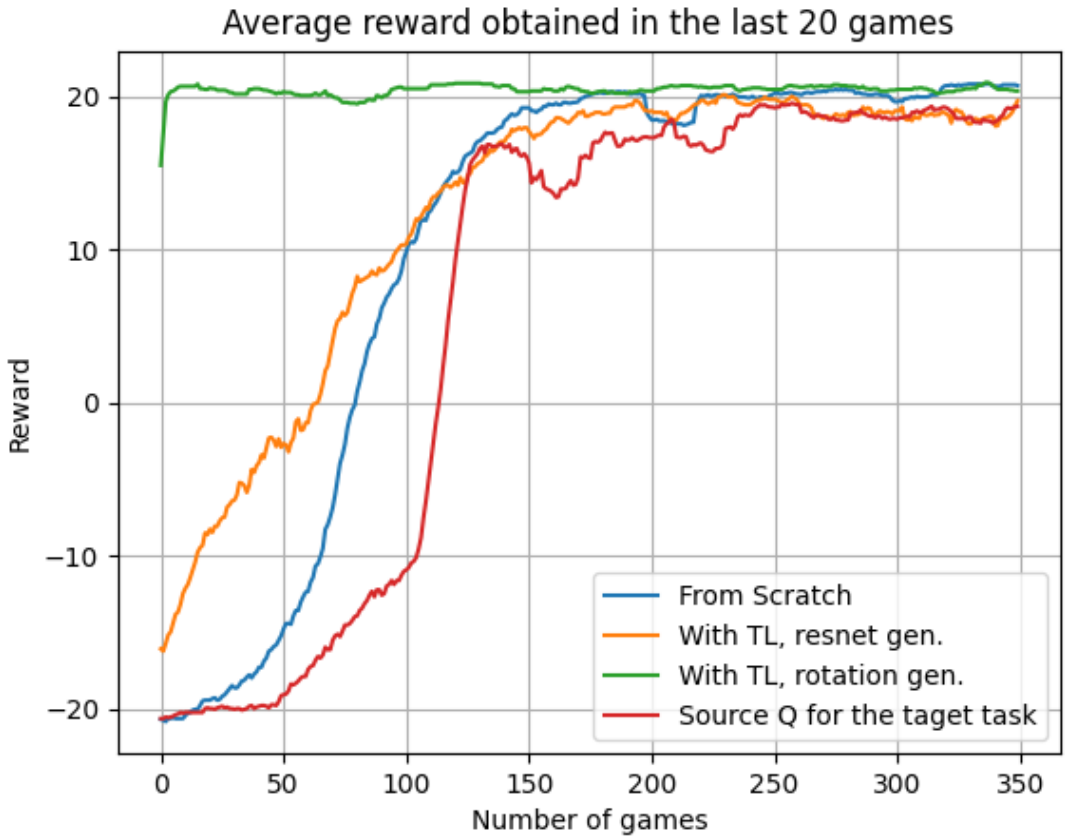
$$\lambda_{Cyc} = \lambda_Q = 0, \lambda_M = 10$$




**Resnet** architecture with rotation layer with

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

# The second case - results - learning the game




Average reward of the last 20 games depending on the number of played games for the game Pong without using TL (learning from sratch) vs. with using TL

# Conclusion and further steps

- Promising method
- All the proposed error parts showed to be important in correspondence learning
- The method depends heavily on the network architecture

**Future plans**

- Test different network architectures such as Transformers etc.
- Further develop the method for:
  - Partially similar tasks only
  - $N$-to-1 transfer learning tasks
- Test in a real-world scenario (e.g. sim-to-real transfer)