RL-Course 2025/26: Final Project Report

Swabian-Olympics: Niklas Ehrenfried, Simon Rappenecker, Fabian Holzwarth

February 22, 2026

1 Introduction

This is the story on how we became the greatest hockey players in the whole course (maybe)

us
(before the tournament)

Reinforcement learning is a branch of machine learning in which an agent learns to take actions to maximize its cumulative reward in an environment. One of the most well-known examples of reinforcement learning is AlphaGo [3], groundbreaking research by Google DeepMind that was the first to defeat a professional Go player.

This project investigates reinforcement learning in a simple, physics-based two player environment called Hockey-Game. The task models a rudimentary competitive game in which two players face each other in a rectangular arena. Each of them defending their own goal while attempting to score against the opponent by shooting the puck. The players are confined to their own half of the field and cannot cross the vertical center line. At every time step, a player can apply a movement and a rotation force to their agent, as well as kicking the puck if it currently is held. If the puck is not kicked after 15 steps, this action is forced automatically. Whoever scores the first goal is declared the winner of the game. If no goal was scored after 250 steps, the game will end in a draw.

In this continuous and dynamic environment, the agent receives a complete observation of the game and selects actions for its player every step. By default, rewards are granted mostly on goals. Some smaller metrics are already rewarded during the gameplay and encourage puck possession and motion towards the opponent's goal. Due to the fully observable environment, this can be extended for custom requirements.

On the following pages, we will present our agent implementations to learn a beneficial policy to navigate this game and try to achieve mastery over the opponent strategies:

- **N-PACT** (**N**-step, **P**rioritized, **A**uxiliary, **C**EM, **T**QC) - Niklas
- **Quantile Soft Actor-Critic (SAC)** - Simon
- **Model Based Soft Actor-Critic (MBPO + SAC)** - Fabian

2 N-PACT (N-step, Prioritized, Auxiliary, CEM, TQC)

2.1 Methods and Implementation Details

2.1.1 Base Algorithm: Truncated Quantile Critics (TQC)

The core of the agent relies on Truncated Quantile Critics (TQC), a distributional reinforcement learning algorithm. Instead of estimating a single expected return, TQC estimates the distribution of the return.

$$L(\theta) = ... \tag{1}$$

2.1.2 N-Step Returns

To improve sample efficiency and accelerate the propagation of delayed rewards, we incorporate N-step returns into the value estimation.

$$R_t^{(n)} = \sum_{k=0}^{n-1} \gamma^k r_{t+k} + \gamma^n \max_a Q(s_{t+n}, a) \tag{2}$$

2.1.3 Prioritized Experience Replay (PER)

$$w_j = \left(\frac{1}{N \cdot P(j)} \right)^{\beta} \tag{3}$$

2.1.4 Auxiliary Tasks

$$L_{aux} = ... \tag{4}$$

2.1.5 Cross-Entropy Method (CEM) Planning

$$\mu_{new} = ..., \quad \sigma_{new} = ... \tag{5}$$

2.2 Experimental Evaluation

2.2.1 Simple Environment Validation

Figure 1: Training performance of N-TQC-PER-Aux-CEM on simple gymnasium environments.

2.2.2 Feature Ablation Study

Figure 2: Ablation study demonstrating the performance gain from each added feature.

2.2.3 Performance Against Basic Opponents

2.2.4 Self-Play and Training Dynamics

Hyperparameter	**Value**
N-step size	...
CEM iterations	...
PER α	...
PER β	...

Table 1: Key hyperparameters used during the final training phase.

3 Quantile-SAC

Here write a short introduction

3.1 Soft-Actor Critic

Descibe SAC and quantile critic in more detail

3.2 Quantile Critic

Describe your non trivial modification

3.3 Self-Play Strategy

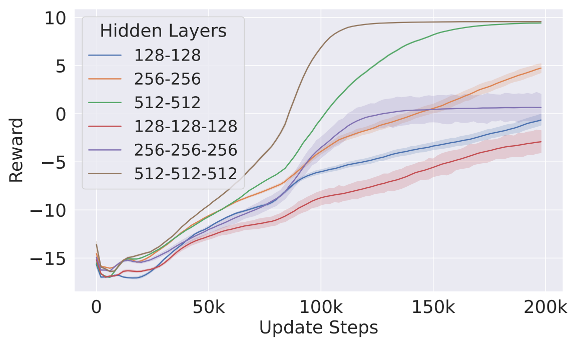
Describe your non trivial modification

3.4 Experiments

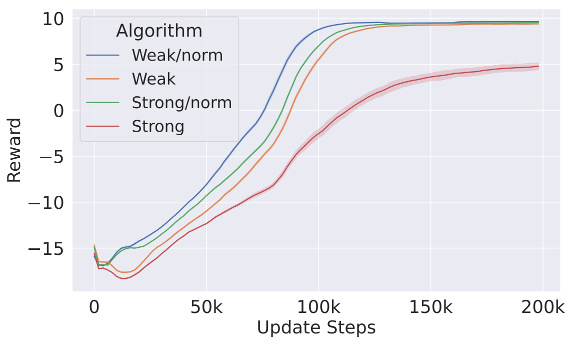
3.4.1 Network Architecture and Normalization

Different hidden layer configurations for both actor and critic were explored, resulting in the final configuration 512-512-512. Note that higher capacity networks increase convergence speed and reduce the variance across random seeds (Fig. 3a). However, when training against the strong opponent, observation normalization is necessary to stabilize training (Fig. 3b).

Training against the weak or the strong opponent takes 20 minutes and results in a win rate of $> 99\%$.



(a)



(b)

Figure 3: **Network Architecture and Observation Normalization.** **(a)** Training against strong with varying critic and actor architectures. **(b)** Training against weak and strong with and without observation normalization. Both plots show the mean and 95% CI over three runs.

3.4.2 Pink Noise

By default, the Soft Actor-Critic uses Gaussian noise to sample actions in the environment. This noise can be replaced by other noise processes, for example with Pink Noise [1]. However, this consistently resulted in slightly reduced convergence speed during training and was therefore not considered further.

3.4.3 Quantile Critic and Self-Play

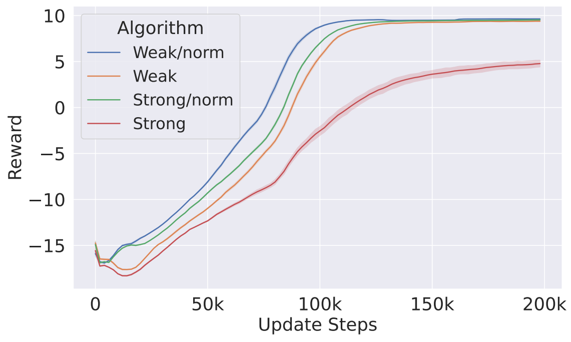
The Quantile-SAC converges slightly slower than the default SAC, but only when training against a single opponent with a fixed strategy (e.g., weak or strong). However, in a self-play setting involving a pool of opponents, the default SAC struggles to find a robust strategy. Q-SAC on the other hand, with a critic aware of the distribution proves beneficial and stabilizes the training. Notably, the Q-SAC finds a counter-strategy considerably faster. This allows it to reach the performance threshold required to add a new agent to the self-play pool sooner, resulting in more experience and therefore better performance in the same amount of time (Tab. 4a).

Win Rates	Weak*	Strong*	SAC**
Q-SAC	100%	98.3%	75.33%
SAC	97.6%	80.6%	–

* Mean over 3 trained models, 100 games per model

** Mean over 900 games (100 games for each model pair)

(a)



(b)

Figure 4: **Self-Play Ablation and Training Run.** **(a)** Win rates after 1M steps of self-play comparing Q-SAC and default SAC. **(b)** Training over 20M steps with Q-SAC.

The final agent used in the competition was trained over 20M steps. To ensure a challenging environment, the basic opponent pool contained checkpoints from team members and previous runs. During its journey to perfection, it accumulated over 170 opponents. To fully exploit this diversity, the pool was frozen at 19M steps. This allowed the model to focus entirely on optimizing its policy against a static, yet diverse set of opponents.

In the competition, the Q-SAC reached ...

4 MBPO + SAC

Even with the very simple environment of the hockey game, we can try to improve sample efficiency further with a combination of a Soft-Actor-Critic (SAC) agent with model based policy optimization (MBPO). The SAC consists of multiple neural networks:

1. Policy Network (Actor) $\pi_\theta(a \mid s)$
   For learning the actual policy we use a neural network with Gaussian policy and tanh squashing. Given the current observation, we receive the action of our agent.

$$\mathcal{L}_\pi = \mathbb{E}\left[\alpha \cdot log(\pi(a \mid s)) - min_i(Q_i(s,a))\right] \tag{6}$$

2. Q-Networks (Two Critics) Q_1 and Q_2
   Two independent neural networks are used for estimating the soft action-value function $Q(s,a)$. By the principle of double Q-learning, we can reduce overestimation bias and provide gradients to train the policy.

$$y = r + \gamma\left(min_i(Q_{target,i})(s',a') - \alpha \cdot log(\pi(a' \mid s'))\right) \tag{7}$$

$$\mathcal{L}_{Q_i} = \mathbb{E}\left[(Q_i(s,a) - y)^2\right] \tag{8}$$

3. Entropy temperature α
   To balance between maximizing reward and encouraging exploration, another network is used that learns and adapts a scalable parameter for the desired policy entropy $\mathcal{H}_{target}$ (Hyperparameter = $-dim(a)$ [2]).

$$\mathcal{L}_\alpha = \mathbb{E}\left[-log(\alpha \cdot (log(\pi(a \mid s) + \mathcal{H}_target)))\right] \tag{9}$$

This alone already learns the behavior of the game and training opponents quite well. To improve the learning process and also introduce some stochastic variety, this SAC Agent is wrapped by the MBPO construct using

1. A dynamic world model
   By trying to learn how the world steps based on an observation and an action, we create this feedforward neural network. This gives us an approximation of the expected next state and reward for any given combination of state and action $(s,a) \to (\hat{s}', \hat{r})$. By using this, we can generate more transitions without stepping the actual environment and exploring actions safely. By chaining the generated states, we can also estimate more into the future with increasing uncertainty.

$$\mathcal{L}_{worldmodel} = \mathbb{E}\left[mse(\hat{s}', s') + \lambda \cdot mse(\hat{r}, r)\right] \tag{10}$$

2. A replay buffer
   To train the world model continuously and keep using some real world data for the SAC training, the experienced transitions (s, a, r, s', d) are stored in a buffer. Training data can now be generated by using random samples from this buffer and extending them with imagined next states from the world model.

5 Comparison

Write something here

References

[1] O. Eberhard, J. Hollenstein, C. Pinneri, and G. Martius. Pink noise is all you need: Colored noise exploration in deep reinforcement learning. In *Proceedings of the Eleventh International Conference on Learning Representations (ICLR 2023)*, May 2023.

[2] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, and S. Levine. Soft actor-critic algorithms and applications. 2019.

[3] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–503, 2016.