# Simulating Anticipatory Centering Behavior in a Robotic Sequential Reaching Task using Deep Reinforcement Learning

Sinttruije, Deborah van

# Simulating Anticipatory Centering Behavior in a Robotic Sequential Reaching Task using Deep Reinforcement Learning

Deborah van Sinttruije

University Leiden

**Abstract**

Humans use inferred statistical properties of sequential events to smoothen subsequent actions by anticipatory movements. These anticipatory movements have been studied in the serial reaction time (SRT) task, in which participants anticipate the target stimuli in learned sequences, however, under uncertainty, the participants seem to adhere to a centering strategy. It remains unclear whether this centering behavior is a statistically inferred way to compensate for the absence of sequence knowledge, using the center as an optimal anticipatory position. In this study, two state-of-the-art Deep Reinforcement Learning (Deep RL) algorithms (Proximal Policy Optimization (PPO) & Soft Actor-Critic (SAC)) are compared and employed to train artificial agents to investigate the scope of centering behavior, by manipulating the frequency distributions of target stimuli. While SAC evidently outperformed PPO in terms of performance and stability, both algorithms displayed an effect of frequency distribution on centering position. Specifically, a proportional shift toward more probable target stimuli, suggesting that centering behavior is indeed anticipatory behavior as a way to compensate for the absence of explicit sequence knowledge.

*Keywords:* Sequential actions, sequential decision making, statistical learning, anticipatory behavior, Reinforcement learning, deep reinforcement learning, PPO, SAC

# Simulating Anticipatory Centering Behavior in a Robotic Sequential Reaching Task using Deep Reinforcement Learning

## Introduction

Implicit or explicit, humans are rather good at inferring the statistical properties of sequential events (Duran & Dale, 2009). It is a good thing we are, since the majority of human activity consists of sequential decision making, and we rely on this skill to execute our actions smoothly. These activities vary from complex endeavors, like navigating, to the most basic human functioning, like motor control. Yet, how exactly humans learn these actions remains an interesting research subject in psychology and neuroscience and is currently exultant by the opportunities Artificial Intelligence (AI) has to offer to the field (Botvinick et al., 2020), which will be elaborated on further below.

In sequence learning inferred statistical properties are used in order to make implicit and explicit expectations and predictions about subsequent actions (Hunt & Aslin, 2001). This behavior is already seen in 8-month-olds, who, are capable of word segmentation from fluent speech based on the statistical properties of adjacent speech sounds (Saffran et al., 1996). This statistical learning effect is driven by mere exposure (Aslin, 2017), an idea that already stems from behaviorists Pavlov, 1927 and Skinner, 1938, nevertheless a powerful mechanism to smoothen our subsequent actions with anticipatory movement. These anticipatory movements have been observed in studies as anticipatory eye movement before stimulus manipulation (Mennie et al., 2007) and anticipatory lip movement before speech (Bell-Berti & Harris, 1979).

A motor learning paradigm to analyze sequence learning is the serial reaction time (SRT) task (Nissen & Bullemer, 1987). In this task, target stimuli will be presented in a repeated sequence at varying locations. Sequence learning was originally demonstrated by comparing the mean response times of structured sequences to random sequences.
de Kleijn et al., 2018 adapted the SRT task to computer-mouse movement with four possible symmetric target locations around the middle. Target stimuli would be

highlighted after a short inter-stimulus interval (ISI). It was found that when there was a repeated structured sequence, participants, after training, were not only faster in terms of response time but also displayed anticipatory movement towards the next highlighted target during the ISI. When a repeated structure was absent, and targets were highlighted at random, participants seemed to conform to a centering strategy, in which the resting position of the mouse in the ISI was in the middle of the four possible target locations. This suggests that even in the absence of sequence knowledge, the participants might still use the statistical properties of the task in order to find the most optimal anticipatory position, equally distant to all possible target locations (Dale et al., 2012; de Kleijn et al., 2018; Duran & Dale, 2009). If participants indeed use statistical learning in order to display this centering behavior, that would suggest that the optimal anticipatory position can be manipulated, along with the centering behavior of the participant, by moving the locations or the probability distributions of the target stimuli.

Humans are not the only statistical learners we know. With the rise of AI Machine Learning (ML) algorithms arose, inspired by human learning and neuroscience (Hassabis et al., 2017), using statistical learning theory (Vapnik, 1999). Meaning these algorithms use statistical inference in order to make estimates and predictions. Because of the similarities between human and machine learning, both fields can benefit from synergism with the other. The biology of the human brain offers ML inspiration for optimizing learning algorithms, while contrariwise ML offers psychology and neuroscience a framework to model and study human learning (Botvinick et al., 2020; Hassabis et al., 2017).

## Human learning to Machine learning

A type of Machine Learning (ML), which is inspired by sequential decision making, is Reinforcement Learning (RL) (Barto et al., 1989; Niv, 2009). RL is a machine learning method in which an artificial agent interacts with an environment, gathers experiences, and learns through trial and error over time. Desirable actions are rewarded and undesirable

actions punished, while the agents try to optimize objectives by maximizing the cumulative reward (Sutton, Barto, et al., 1998). The basics of learning through reinforcement originate from classical conditioning and operant conditioning. Classical conditioning, also called Pavlovian conditioning, is a behavioral procedure that occurs when a neutral stimulus is paired with a potent stimulus; the neural stimulus will elicit a response similar to the response the potent stimulus would elicit (Pavlov, 1927). In operant conditioning, also known as instrumental conditioning, behavior is paired with a consequence; behavior can be modified by either reinforcement or punishment (Skinner, 1938). Building further on the work of Pavlov, 1927, Rescorla, 1972 created a model in which learning was driven by the difference between the predicted and received reward. This learning difference ($\Delta V$) can be symbolized by the following equation:

$$\Delta V = \alpha\beta(\lambda - \sum V), \tag{1}$$

where $\lambda$ represents the received reward and $\sum V$ the summarized total of associative value, in other words, the expectation of the reward. Together $(\lambda - \sum V)$ is the difference in expectation and actual reward, or error. This difference is used to update the summarized total of associative value. Alpha $\alpha$ and beta $\beta$ are weights that depict the salience of the conditioned stimulus ($\alpha$) and learning-speed given the unconditioned stimulus ($\beta$). These weights determine the magnitude of adjustment the difference will have on the previously established associative value. Together, these concepts have become the foundation of RL (François-Lavet et al., 2018; Jozefowiez, 2002), which is elaborated at a computational level below.

## Reinforcement Learning

A RL problem can be mathematically modeled as a policy search in a Markov decision process model (MDP), defined by a tuple $(S, A, P, R, \gamma)$, which consists of:

- Discrete time $t = 0, 1, 2, ... T$
- A discrete set of states $s \in S$
- A discrete set of actions $a \in A(s)$ for each $s$
- A transition function $p(s'|s, a)$: the probability of transitioning from state $s$ to state $s'$ while taking action $a$
- A reward function $r(s, a, s') = \mathbb{E}[r|s, a, s']$: expected reward of taking action $a$ at state $s$ transitioning to $s'$
- Discount factor $\gamma$: values the importance of future rewards

In RL, the agent in this attempts to learn policy $\pi$ in order to maximize the sum of expected rewards. Policy $\pi$ specifies what actions should be taken in a given state. This can either be deterministic $a = \pi(s)$ or stochastic $\pi(a|s) := P(A_{t+1} = a|S_t = s)$. The policy is determined by value functions $v(s)$ and $q(s, a)$, which in turn are determined by the policy. The state value function $v_\pi(s)$ is the expected total return of state $s$ when actions are specified by policy $\pi$. The state-action value function $q(s, a)$ is the expected total return of state $s$, given $a$ under policy $\pi$, meaning first action $a$ is taken independently of the policy $\pi$. The state value function can be written as: $v_\pi(s) = \sum_a \pi(a|s)q(s, a)$ and the state-action value function as: $q(s, a) = \sum_{s'} p(s'|s, a)[r(s, a, s') + \gamma v_\pi(s')]$. Rewritten this gives us the following Bellman Equations to describe the value of a state:

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s'} p(s'|s, a)[r(s, a, s') + \gamma v_\pi(s')] \tag{2}$$

$$q_\pi(s, a) = \sum_{s'} p(s'|s, a)[r(s, a, s') + \gamma \sum_{a'} \pi(a'|s')q(s', a')]. \tag{3}$$

Ultimately agents search for the optimal policy $\pi^*$, this is done by maximizing the value functions: $v_{\pi*}(s) = v^*(s) := \max_\pi v_\pi(s)$ and $q_{\pi*}(s, a) = q^*(s, a) := \max_\pi q_\pi(s, a)$.

There are two types of methods to learn the optimal policy: On-policy and off-policy. On-policy methods use the policy that was made to make decisions to evaluate or improve the policy, off-policy methods evaluate or improve the policy independently of

the policy used to make these decisions (Sutton, Barto, et al., 1998). The trade-off between the two is usually characterized by stability versus data efficiency. Off-policy learners tend to be less data hungry but have to relinquish stability, as opposed to on-policy learners that retain more stability while being less data efficient (Haarnoja, Pong, et al., 2018; Juliani et al., 2020; Nachum et al., 2018).

Furthermore, there are two types of RL problems: Model-based, where the MDP is completely specified, and Model-free, for which the MDP is unknown. In model-free RL problems, the agent can only learn from direct experiences, like sample paths. This means that the agent will have to execute a random search, however, can still use the Bellman equations the propagate values. Exploiting these equations, Temporal difference (TD) learning is a method in which predictions are made over successive timesteps to drive the learning process (Sutton & Barto, 1987). An update step through TD can be displayed as the following equation:

$$v_\pi(S_t) \leftarrow v_\pi(S_t) + \alpha[R_{t+1} + \gamma v_\pi(S_{t-1}) - v_\pi(S_t)], \tag{4}$$

where the value of a state is updated by taking the old value and adding the weighted difference between the old value and the new information found. This weighted difference: $\alpha[R_{t+1} + \gamma v_\pi(S_{t-1}) - v_\pi(S_t)]$, corresponds to $\Delta V$, the difference between the expected and actual value, or error, seen in the Rescorla-Wagner model in Equation 1. In order to improve the policy in a Model-free problem, the agent will have to balance exploration versus exploitation of the state-space. The agent needs to explore enough state-actions in order to improve its knowledge about the value of a state, contrastingly, it will need to exploit its current estimated values to get the most reward (Sutton, Barto, et al., 1998).

### *Deep RL*

When transferring RL to real-world problems, we are struck by the curse of dimensionality, meaning that, in order to obtain reliable results in situations that approach real-world complexity, the state-action space and amount of data needed will exponentially grow with the dimensionality (Bellman, 1957; Perrusquia et al., 2019). When challenged with these problems, it is important that the agents use the high-dimensional inputs efficiently to derive representations of the environment (Mnih et al., 2015). Therefore, if the state-space is high-dimensional combining RL with the representational power of deep learning, named deep RL is most useful (Botvinick et al., 2020; François-Lavet et al., 2018).

Deep learning is a machine learning network inspired by the biological neural networks in the brain, in which neurons are connected through synapses, which are used to receive and transmit signals in order to transmit the information from one neuron to another (Goodfellow et al., 2016). In deep learning artificial neural networks are utilized, which transform a set of inputs into a set of outputs. The network consists of layers of units; an input layer, an output layer, and one or more hidden layers in between. The connections between there units are weighted and except for the input layer, at every layer, the input of each unit is computed as the weighted sum of the units of the previous layer. After the computation from input to output, we can backpropagate gradients from the output to the input by computing the error derivatives backward. Like this, the weights can be updated to optimize some loss function (Li, 2017).

In deep RL, deep learning is incorporated to solve high-dimensional RL problems. In these models, the policy $\pi$ or other learned functions are represented inside a deep neural network. These deep RL systems typically use the neural network in order to compute non-linear mappings from observational input to action output, while also using RL signals to update the network weights (Botvinick et al., 2020; Mnih et al., 2015). When extending RL with deep learning, the following hyperparameters are enforced in the RL model (Juliani et al., 2020):

1. Max steps: The Total amount of steps the simulation entails.

2. Batch size: The number of experiences (agent observations, actions, and rewards obtained) used for one iteration of a gradient descent update. The batch size should always be a fraction of the buffer size.

3. Buffer size: The function of the Buffer size differs for on-policy and off-policy algorithms. For on-policy algorithms, it is the amount of experiences that should be collected before learning or updating the model. For off-policy algorithms, it is the maximum number of experiences that can be stored in the experience replay buffer. Off-policy algorithms use this experience buffer to train from past experiences.

4. Time horizon: How many steps of experience are needed before adding it to the experience buffer. It is important that the time horizon is big enough to capture the sequence behavior of the agent.

5. Learning rate: The strength of each gradient descent update step.

6. Learning schedule: Either constant or linear (the learning rate decreases linearly towards 0 reaching the end of the training).

Two state-of-the-art deep RL algorithms are Proximal Policy Optimization (PPO) (Schulman et al., 2017) and Soft Actor-critic (SAC) (Haarnoja, Zhou, Abbeel, et al., 2018) which will be explained in detail below. Both of these algorithms are Actor-Critic methods, in which the policy and value function are represented independently and TD learning methods are used to drive the learning process (Konda & Tsitsiklis, 1999). In this method, the actor and critic are separate entities. The actor represents the policy for the possible actions given a state, the critic represent the value function, which evaluates the actions of the actor.

### *Proximal Policy Optimization*

PPO (Schulman et al., 2017) is an on-policy family of policy optimization methods that combine the stability and reliability of trust region policy optimization (TRPO) with the benefits of first-order optimization. The key feature of the PPO algorithm is a clipped "surrogate" objective function for computing policy updates. More basic policy gradient implementations use an objective function whose gradient is the policy gradient estimator:

$$L^{PG}(\theta) = \hat{\mathbb{E}}_t[log\pi_\theta(a_t|s_t)\hat{A}_t], \tag{5}$$

in which the log probability of stochastic policy $\pi_\theta$ is multiplied by the estimation of the advantage function $\hat{A}_t$ at timestep t to evaluate the impact of action $a$ in state $t$ (Mnih et al., 2016). Estimates $\hat{A}_t$, which tell us how "good" a certain decision is, rely upon parameters $\gamma \in [0, 1]$ and $\lambda \in [0, 1]$. Parameter $\gamma$ is the discount factor, which controls how much future rewards are valued. Parameter $\lambda$ makes a compromise between bias and variance (Schulman, Moritz, et al., 2015). Since this policy gradient is on-policy, it will search actions from the current state. Therefore, if it takes large steps, it can end up in bad states and will have to resume from these bad states with locally bad policies. Therefore, multiple steps of optimization with this loss $L^{PG}$ function with the same trajectory can lead to uncontrollably large policy changes, which can hurt performance badly (Schulman et al., 2017).

TRPO tackles this problem by maximizing a surrogate objective to a constraint on the size of the policy update, namely:

$$\underset{\theta}{\text{Maximize}}\ \hat{\mathbb{E}}_t\left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta old}(a_t|s_t)}\hat{A}_t\right] \tag{6}$$

$$\text{Subject to } \hat{\mathbb{E}}_t[\text{KL}[\pi_{\theta old}(\cdot|s_t), \pi_\theta(\cdot|s_t)]] \leq \delta. \tag{7}$$

In Equation 6 the probability ratio $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta old}(a_t|s_t)}$ of $\pi_{\theta old}$, the vector of policy

parameters before the update and current policy vectors $\pi_\theta$, is utilized to find the impact of certain actions in certain states. If $\pi_\theta = \pi_{\theta old}$, actions are as likely under the current policy as they were under the old policy, therefore we have $r(\theta_o ld) = 1$. When actions under the current policy are more likely, the ratio will become $> 1$. Equation 7 exhibits the constraint the objective is subjected to. With the Kullback–Leibler (KL) divergence, the difference between the data distribution of the old policy and the current policy is measured. This distribution must the lower than $\delta$, the size of the region (Schulman, Levine, et al., 2015). Since hard constraints heavily impact performance, a penalty is suggested instead. However, finding the appropriate coefficient for this penalty is hard to determine (Schulman, Levine, et al., 2015; Schulman et al., 2017). Without constraint, maximization of the $L^{CPI}$ loss function:

$$L^{CPI}(\theta) = \hat{\mathbb{E}}_t \left[ \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta old}(a_t|s_t)} \hat{A}_t \right] = \hat{\mathbb{E}}_t[r_t(\theta)\hat{A}_t], \tag{8}$$

will also lead to unreasonably large policy changes. The PPO-clip algorithm addresses this by clipping the probability ratio of the surrogate objective:

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)], \tag{9}$$

where the minimum is taken of $L^{CPI}$ and $\text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)$. In the second term, the ratio $r_t(\theta)$ is limited from moving outside of interval $[1-\epsilon, 1+\epsilon]$ using hyperparameter epsilon. Because the minimum is taken, the final objective will never be higher than the unclipped objective, meaning that the final objective will be a lower bound on the unclipped objective. Like this, the change in probability ratio is only included if it makes the objective worse. Pseudo-code of the clipped PPO from Schulman et al., 2017 can be found in Algorithm 1.

**Algorithm 1** Algorithm 1 PPO, Actor-Critic Style

**for** each iteration **do**

**for** agent $1, ..., N$ **do**

Run policy $\pi_{\theta old}$ in environment for $T$ timesteps

Compute advantage estimates $\hat{A}_1, ..., \hat{A}_T$

**end for**

Optimize surrogate $L$ wrt $\theta$, with $K$ epochs and minibatch size $M \leq NT$

$\theta_{old} \leftarrow \theta$

**end for**

### *Soft Actor-Critic*

SAC (Haarnoja, Zhou, Abbeel, et al., 2018) is an off-policy algorithm based on the maximum entropy RL framework. The objective in this framework combines the standard objective with an entropy term, such that the optimal policy is found while also maximizing entropy (Ziebart, 2010; Ziebart et al., 2008). Meaning that the agent will try to succeed in the task while acting the most random. This gives us the following equation:

$$\pi^* = \underset{\pi}{\text{argmax}} \sum_t \mathbb{E}_{(s_t,a_t)\sim\rho_\pi}[r(s_t, a_t) + \alpha\mathcal{H}(\pi(\cdot|s_t))], \tag{10}$$

where $\pi^*$ is the optimal policy and $\alpha$ is the temperature parameter which determines the gravity of the entropy term $\mathcal{H}(\pi(\cdot|s_t)$ versus reward $r(s_t, a_t)$, therefore controlling the randomness for the optimal policy. Because of the entropy term, the policy is encouraged to explore more and even capture various approaches of near-optimal solutions (Haarnoja, Zhou, Abbeel, et al., 2018; Ziebart, 2010). Since it is hard to choose a certain temperature $\alpha$ Haarnoja, Zhou, Hartikainen, et al., 2018 automated this process for the user with the following Equation:

$$\alpha_t^* = \underset{\alpha_t}{\text{argmax}} \mathbb{E}_{a_t\sim\pi_t^*}[-\alpha_t log\pi_t^*(a_t|s_t;\alpha_t) - \alpha_t\bar{\mathcal{H}}]. \tag{11}$$

SAC is rooted in Soft Policy Iteration, in which a soft Q-function including entropy term is defined by:

$$\tau^{\pi}Q(s_t, a_t) \triangleq r(s_t, a_t) + \gamma \mathbb{E}_{(s_t+1)\sim p}[V(s_(t+1)], \tag{12}$$

where $\tau$ is a backup operator, corresponding to the gravity of the Q-update (Juliani et al., 2020) and $V(s_t)$ can be defined as:

$$V(s_t) = \mathbb{E}_{(s_t)\sim\pi}[Q(s_t, a_t) - \alpha log \pi(a_t|s_t)], \tag{13}$$

where the entropy term $\alpha$ now gives weight to the log probability of the stochastic policy. The policy update is defined by the KL-divergence:

$$\pi_{new} = \underset{\pi'\in\Pi}{\operatorname{argmin}} D_{KL}\left(\pi'(\cdot|s_t)||\frac{\exp\frac{1}{\alpha}Q^{\pi old}(s_t,\cdot))}{Z^{\pi old(s_t)}}\right), \tag{14}$$

where the distribution is normalized by partition function $Z^{\pi old}(s_t)$ and $\Pi$ is a restricted set of policies, containing only traceable policies. This projected policy update makes sure that the new policy has a higher value in terms of maximum entropy objective than the old policy (Haarnoja, Zhou, Abbeel, et al., 2018).

Since these functions can only find the optimal solutions in tabular form, SAC will need to use function approximation for the continuous domains. For both the Q-function and policy, function approximators will be utilized and alternated between updating these function approximators with stochastic gradients and using the current policy to collect experiences from the environment. The soft Q-function will be modeled as a neural network with network-parameter $\theta$, and trained by using stochastic gradients to minimize the soft Bellman residual:

$$J_Q(\theta) = \mathbb{E}_{(s_t,a_t)\sim\mathcal{D}}[\frac{1}{2}(Q_\theta(s_t, a_t) - (r(s_t, a_t) + \gamma \mathbb{E}_{st+1\sim p}[V_{\bar{\theta}}(s_{t+1})]))^2]. \tag{15}$$

The policy can be modeled as Gaussian distribution with a mean and covariance

with network parameters $\phi$. These parameters can be trained by minimizing the expectation of the KL-divergence seen in Equation 14 with the following formula:

$$J_\pi(\phi) = \mathbb{E}_{(s_t)\sim\mathcal{D}}[\mathbb{E}_{(a_t)\sim\pi_\phi}[\alpha \log(\pi_\phi(a_t|s_t)) - Q_\phi(s_t, a_t)]] \tag{16}$$

.

Because the target density is a neural network (Q-function), the policy can be re-parameterized by using $a_t = f_\phi(\epsilon_t; s_t)$. As with the Q function approximate, we can approximate Equation 16 with gradient estimators (Haarnoja, Zhou, Abbeel, et al., 2018). Because the algorithm is off-policy, the value estimators and policy can be trained from the replay pool $\mathcal{D}$. A pseudo-code overview for SAC from Haarnoja, Zhou, Hartikainen, et al., 2018 can be found in Algorithm 2.

**Algorithm 2** Algorithm 2 SAC, Soft Actor-Critic

**for** each iteration **do**
  **for** each environment step **do**
    Sample $a_t$ action for policy $\pi_\phi(a_t|s_t)$
    Sample transition $s_{t+1}$ from environment $p(s_{t+1}|s_t, a_t)$
    Store transition in replay pool $\mathcal{D}$
  **end for**
  **for** each gradient step **do**
    Update Q-function parameters $\theta_i$
    Update policy weights $\phi$
    Adjust temperature $\alpha$
    Update target network weights $\bar{\theta}_i$ using $\tau$
  **end for**
**end for**

Human learning has been and still is a huge inspiration for Machine learning, subsequently, machine learning can now help us in our quest for deeper understanding of

human learning. Computational modeling, formulating and simulating cognitive processes in terms of mathematical models, had become increasingly popular in the field of psychology and neuroscience over the last decade (Lewandowsky & Farrell, 2010; Wilson & Collins, 2019). Fitting these computational models to behavioral data can help us better understand the underlying behavior. RL models have been used method to explain decision making and accompanying cognitive processes (François-Lavet et al., 2018; Niv, 2009; Zhang et al., 2020).

To help analyze sequence decision making behavior Kachergis et al., 2016 used RL to model the motor learning paradigm in the SRT task. They found two RL models, Q-learning and SARSA, which could partially explain the effect of rewards in sequence learning behavior in humans. Q-leaning (Watkins, 1989) and SARSA (Rummery & Niranjan, 1994) are two popular model-free RL methods that use TD learning. The difference between the two lies in their value function and policy updating strategy. Q-learning is off-policy, whereas SARSA is on-policy. Extending this research, in a later study (de Kleijn et al., 2018) the same RL models were also effectively used to help model reactive responses in sequence learning behavior. In terms of maximum score, fit and variability Q-learning seems to outperform SARSA in modeling sequence decision making behavior (de Kleijn et al., 2018; Kachergis et al., 2016).

However, these models only partially explained sequence learning behavior. With RL alone, we can analyze how rewards shape decision making, deep RL opens up new explanatory principles in neuroscience. By the notion of superadditivity, the integrated aspects of deep learning and RL, do not only give us the power to model meaningful abstract representation plus reward-driven decision making behavior of RL, but also provides us with a framework to study how reward-driven learning builds these representations, and, in turn, how these representations drive decision making and learning (Botvinick et al., 2020).

Building on RL models to explain sequence learning in the SRT task, studies with

deep RL algorithm PPO were conducted in a more realistic 3D environment, simulation the real-world complexity of the task (de Kleijn et al., 2022; Sen et al., 2022). These studies made use of an agent resembling a human arm, tasked to touch target stimuli. This agent consists of a simulated robotic arm with 4 degrees of freedom to move around (DoF) controlled by a neural network with two fully connected hidden layers, each containing 128 units. In line with human studies of sequence learning behavior, it was found that deep RL agents made use of predictive information to optimize behavior in the SRT task (Sen et al., 2022), would adhere to a centering strategy under uncertainty (de Kleijn et al., 2022; Sen et al., 2022) and displayed more flexibility in critical learning periods (de Kleijn et al., 2022). Coincidentally, these results, show promising prospects of further investigating and modeling sequence decision making behavior with the deep RL agent.

**Contributions**

The current study extends on the centering behavior of the Reacher agent in absence of explicit predictive information in the studies of Sen et al., 2022 and de Kleijn et al., 2022. In this experiment, the Reacher will be simulated again, performing the SRT task, in a more challenging environment. Instead of below, the target stimuli will appear above the Reacher agent, urging the agent to actively keep its arm up in order to perform the task. Additionally, alongside PPO, the newer SAC algorithm will be employed to investigate its performance on the task. And foremost, the scope of centering behavior itself will be investigated by moving the optimal centering position by changing the frequency distribution of the target stimuli. This work aims to answer the following research questions:

1. How does the performance of the algorithms PPO and SAC compare on the Reacher SRT task?

2. How does the frequency distribution of target stimuli influence the centering behavior of the Reacher agent?

Concerning performance, there are no prior results with the SAC algorithm on the SRT task, and the performance of algorithms tends to be problem specific, therefore this research will be mostly exploratory. However, in SRT studies using RL algorithms (de Kleijn et al., 2018; Kachergis et al., 2016), it was found that the off-policy learner outperformed the on-policy learning, which is why it might be hypothesized that the SAC algorithm will indeed outperform PPO in terms of performance on the SRT task.

Regarding centering behavior, previous research suggested that the centering strategy was a way to compensate for the absence of sequence knowledge, using the center as an optimal anticipatory position to unknown subsequent stimuli (Dale et al., 2012; de Kleijn et al., 2022; Duran & Dale, 2009). This would mean that if the optimal anticipatory position changes, like when the probability distribution of the target stimuli is skewed, this might also result in a shift of centering location towards stimuli that appear more frequent. Since, like in humans, less initial distance to the stimuli results in faster response times and therefore more rewards, it is expected that both PPO and SAC algorithms display a similar pattern. Hence, it is hypothesized that the probability distribution of target stimuli influences the centering behavior of the Reacher agent.

## Method

### Implementation

For this study ML-Agents release 14 (Juliani et al., 2020) was used in the Unity 3D version 2019.4.31f1 with the ML agents 1.8.1 and ML-Agents Extensions 0.2.0 toolkits as physics simulators. The agents were trained with reinforcement algorithms PPO and SAC included in ML-Agents release 14, using Python 3.8.8 with library mlagents 0.24.1. The source code for the implementation can be found at https://github.com/deborahvans/Reacher. The experiment was conducted using ALICE High-Performance Computing facility of Leiden University.

## Experiment

In this study, the scope of centering behavior was extended and compared among reinforcement algorithms PPO and SAC. To investigate centering behavior, the location of the hand of the agent was measured right before the end of the inter-stimulus interval (ISI). The "center", meaning the most profitable resting state in terms of least movement acquired to reach the targets, was manipulated by changing the frequency at which certain targets appeared. The frequency distribution was manipulated among one axis, meaning that either on the right-side or the left-side targets were more likely to appear. Per reinforcement trainer, 11 different frequency conditions were created ranging from a distribution of 0-100 left-right to 100-0 left-right. For the first condition, 0-100, this means that there is a 0% chance of a stimulus appearing on the left side of the agent and 100% chance of a stimulus appearing on the right side. An overview of all conditions can be found in Table 1. Every condition consists of 20 runs, meaning a total of 20*11*2 runs, which approximately took 0.5 hours per run for the PPO algorithm and 2 hours for SAC.

**Table 1**

**Conditions for each reinforcement learner.**

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|
| 0-100 | 10-90 | 20-80 | 30-70 | 40-60 | 50-50 | 60-40 | 70-30 | 80-20 | 90-10 | 100-0 |

### *Setup*

***Agents:*** The agent consists of an arm constructed of two arm segments and a hand. In Figure 1 the agent and the goal can be seen. The robot arm is centered at the shoulder (black sphere) and has two actuators, both retaining 3 degrees of freedom to move around. The first actuator is positioned at the shoulder and the second one at the elbow between the two arm segments (white spheres). The hand (blue sphere) of the agent embodies a sensor and can destroy the goal (green sphere) on collision. When torque is

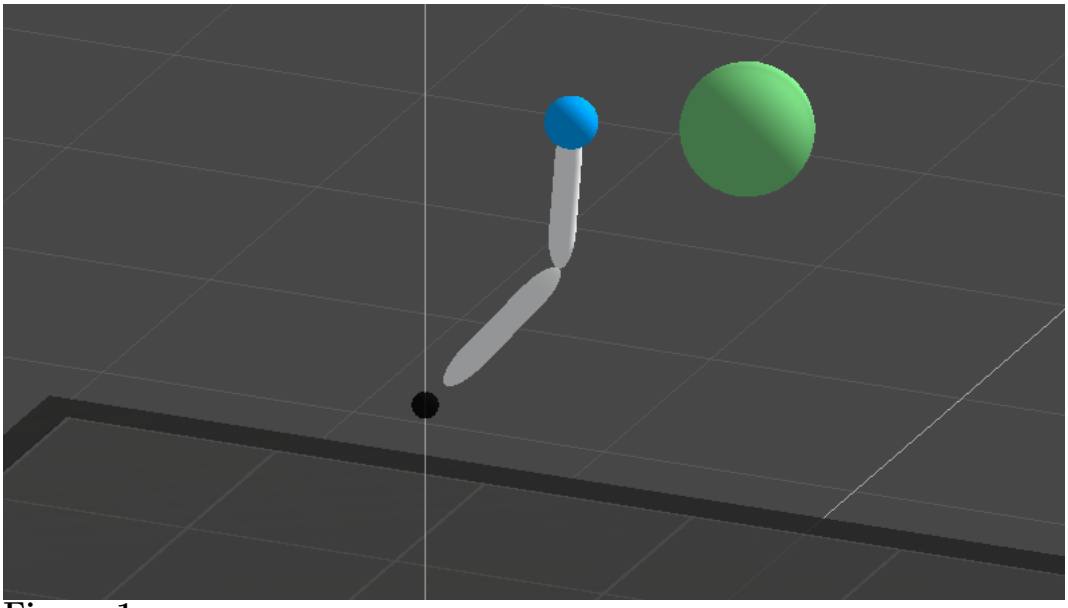
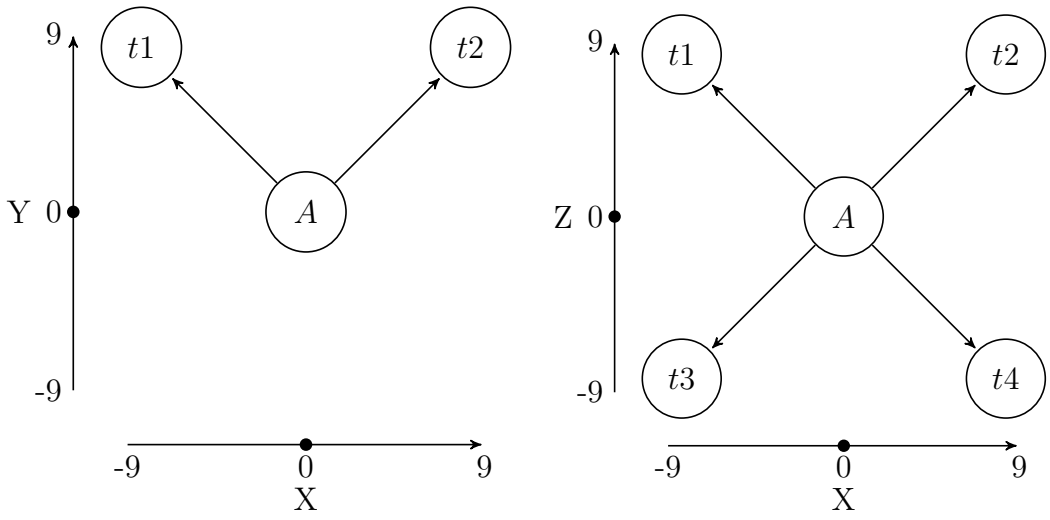


**Figure 1**

*Robot arm and goal (green).*

applied to the actuators to move the joints, axial motion is ignored. Therefore this results in four degrees of freedom in total.

***Task design:*** Representations of the current reaching task are shown in Figure 2. For this serial response time (SRT) task, stimuli will appear that the agent has to touch. Figure 2a contains a representation from the side, in which can be seen that the stimuli appear above the agent. By presenting the stimuli above the agent, it is avoided that the agent can display centering behavior by "letting loose" and hanging down its arm. Now, the agents need to actively keep its arm up to display centering behavior. In Figure 2b we see a Bird's-eye view of the locations at which a stimulus can appear. Target stimuli will appear in random order at one of these four locations. The stimuli will remain for a maximum of 200 timesteps and will either disappear if this limit is reached or if the stimulus is touched by the hand of the agent. After the stimulus disappears there is an ISI of 50 timesteps after which a new stimulus appears. The reward for touching target stimuli starts at 1 with a decay of 0.001 per timestep. This gives us a reward function of reward = 1-0.001*timestep. Meaning that if the stimulus would be touched at timestep 0 this results in a reward of 1




(a) *View from the side.* (b) *View from the top.*

**Figure 2**

*Visual representation on the serial response time (SRT) task. Node A is the anchor point of the agents (the shoulder). Nodes* $t1 - t4$ *are the possible coordinates at which a target stimulus can appear. The distance between the anchor point and stimuli can be calculated via the Pythagorean theorem:* $\sqrt{9^2 + 9^2 + 9^2} = 15.59$.

and the lowest possible reward would be at the limit of 200 timesteps would remain 0.8. This is reset for every new target. To discourage inefficient movements and mimic biological energy consumption a small penalty of 0.0001 * absolute distance moved per timestep was imposed. The overall performance of touching targets is quantified as cumulative rewards (minus penalties) per episode. An episode is a total of 4000 timesteps.

Where the target stimulus appears is generated by discrete uniform distribution U(0,1) per axis. Through curriculum learning the training is divided into two equal fractions. For the first half of the training, for both axes, the frequency distribution of where the stimuli can appear is 50-50. This means that for both axes there is a threshold of 0.5. For each axis, a discrete uniform number is generated. If the number is below the threshold, the coordinate on the axis will be 9, otherwise, the coordinate will be -9 for this axis. Like this, all locations have the same likelihood of being the target location. In the

second half of the training, the frequency distribution for the x-axis is changed. The Z-coordinate will still have an equal chance of being 9 or -9, however, the chance of the X-coordinate being 9 or -9 now depends on the frequency distribution condition. If the frequency condition is for example 70-30, the threshold will be 0.7. This results in a 70% chance of the X-coordinate being 9 and a 30% chance of it being -9. Centering behavior is then determined by the location of the hand of the agent at the end of the ISI, 1 timestep before the new target stimulus appears.

***Neural Controller and hyperparameters:*** The "brain" of an agent is the neural controller consisting of a feedforward network. A more complex model was used compared to previous experiments with the Reacher (de Kleijn et al., 2022; Sen et al., 2022), consisting of three hidden layers which all contained 512 units. The input vector holds 33 elements, which consisted of the following observations:

- Position of both arm segments (2 × 3 elements)
- Rotation of both arm segments (2 × 4 elements)
- Velocity of both arm segments (2 × 3 elements)
- Angular velocity of both arm segments (2 × 3 elements)
- Position of the hand (3 elements)
- Position of the goal (3 elements)
- Presence of the goal (1 element)

The output layer determines the "action" of the agents. In this case, the torques applied to the actuators of the arm segments, causing them to rotate and therefore move. PyTorch was used for the implementation and underlying representation of the deep learning model. After training an agent the output model was provided as a .onnx file, which could be used as a brain.

Most parameters for both PPO and SAC are based on a more complex model that was provided by Juliani et al., 2020. For comparability reasons, the learning rate schedule of both PPO and SAC was set to constant, meaning that the learning rate (of 0.0003) would remain the same for the whole training. The max steps for PPO (4000000) were

doubled compared to SAC (2000000). This decision was made because off-policy algorithms tend to be more data-efficient and therefore require fewer steps, but spend more time performing updates (Haarnoja, Zhou, Hartikainen, et al., 2018; Juliani et al., 2020; Nachum et al., 2018). This was also confirmed by several test runs. Since for this task, the agents need to capture the underlying distribution of the stimuli, the time horizon to learn was expanded for both PPO and SAC. The time horizon was set to match the episode length (4000), as this is common practice (Juliani et al., 2020). More details of the parameter settings can be found in Appendix A, Table A1.

### *Data accumulation and statistics*

After conducting the experiment and collecting the data, the data was aggravated to an .csv file in order to be processed in R studio.

***SRT performance:*** To explore the overall performance of both PPO and SAC algorithms on the SRT task, the cumulative rewards were plotted over time. In order to investigate a statistical difference between the general performance on both algorithms, the mean cumulative rewards at the end of the training were compared by applying an independent sample t-test.

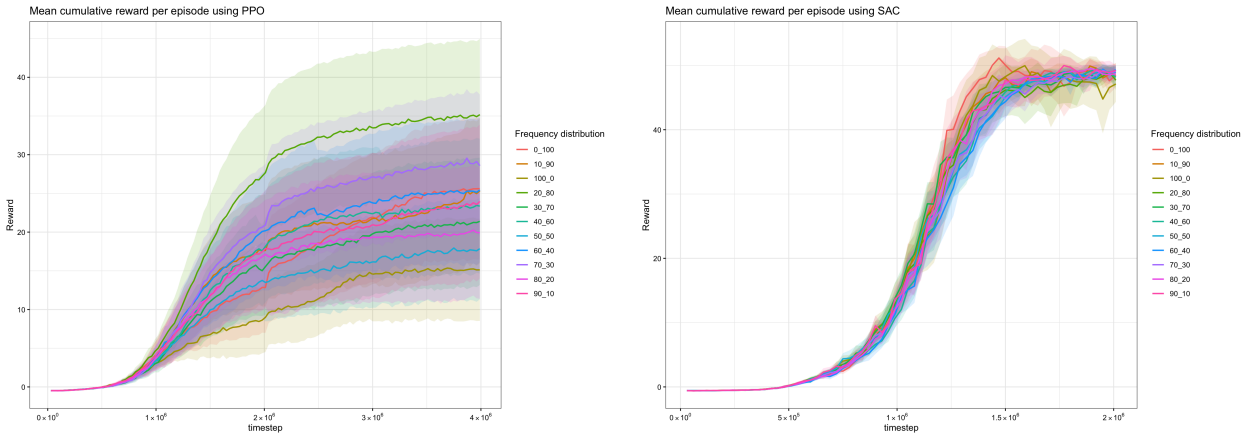
***Centering behavior:*** First, centering behavior was explored and examined separately for PPO and SAC. The locations of the X-coordinates of the agent at the end of the ISI were visualized over time and grouped by their frequency distribution. Additionally, the final X-coordinates at the end of the ISI are plotted against the frequency distribution that was used in the training. Next, to analyze the effect of frequency distribution on centering behavior, for each algorithm a linear regression was performed with the frequency of the X-coordinates being -9 as continuous variable as independent variable and the X-coordinates of the hand agent at the end of the ISI as dependent variable. Likewise, an Analysis of Variance (ANOVA) was performed with the same variables, alternately with the frequency distribution as a factor instead of a numeric variable. Finally, to investigate the difference between centering behavior between the algorithms, the euclidean distances

between the position of the hand of the agent and optimal centering position are calculated and compared by applying an independent sample t-test.

## Results

### *SRT performance*

Figure 3a exhibits the mean cumulative rewards per episode and 95% confidence interval (CI) for each distribution group of PPO over time. Likewise, Figure 3b displays the same for the SAC trainer.




(a) *PPO trainer results.* (b) *SAC trainer results.*

**Figure 3**

*Mean and 95% CI per distribution group of cumulative rewards per episode (4000 timesteps) over time.*

For the PPO algorithm in Figure 3a we see relative broad CI's, indicating the estimate and therefore trainer is not very stable. Since the CI's of the different frequency distribution groups are mostly overlapping it seems most likely that the groups are from the same population in terms of general SRT performance. In comparison, the SAC algorithm in 3b shows a lot more stability over the different runs within and between the different frequency distribution groups. Since the CI's are very narrow and still overlapping it can be deduced with more precision, that the means are likely from the same population

and very close to the actual population mean. Since these measures showed no evidence of statistical differences between frequency groups (within the algorithm) they are combined for the following measure.

The overall means and standard deviations (SD) combined of all frequency conditions per algorithm can be found in Figure 4. The SD's in the Figure show that the SAC algorithm results in reduced variability compared to the PPO algorithm. Since the SD's are not overlapping and the mean for SAC is higher it seems probable that the performance on the SRT task is statistically higher for the SAC conditions.

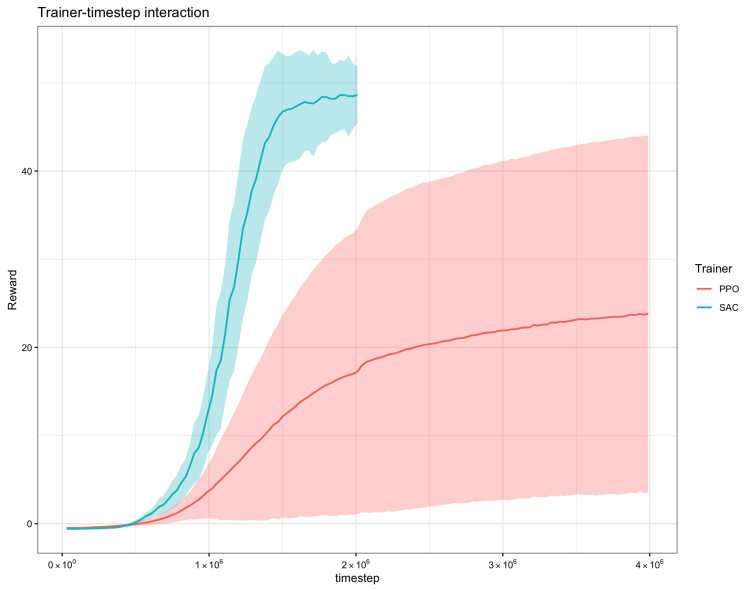



**Figure 4**

*Total mean and SD of cumulative rewards per episode over time.*

Given the visualizations, the SAC algorithm seems to exceed the PPO algorithm in performance on the SRT task. Therefore, a one-sided $t-$test was conducted. The independent sample $t-$test confirmed that, at the end of the training, the SAC algorithm ($M$ = 48.49) surpassed the PPO algorithm ($M$ = 23.81) in performance, $t(235.81)$ = -17.847, $p <$ 2.2e-16. This indicated that overall the SAC trainer exceeds the PPO trainer in training an agent on the SRT task.

### *Centering behavior*

For this part, we start by exploring centering behavior for PPO and SAC separately. Starting with PPO, in Figure 5 the effect of the frequency distribution of the X-coordinate of the target stimulus on the centering behavior of the agent can be seen over time. In the first half of the training, the frequency distribution of a stimulus being either at X-coordinate 9 or -9 is 50-50. In the Figure, this results in the means of all conditions being spread around 0 with their CI's almost completely overlapping. After the curriculum is used in the second half of the training this spread seems to expand and the more skewed the frequency distribution, the bigger the deviation of the mean from 0 seems to be. The clear distinction in the graph between the first and second half of the training suggests an effect of frequency distribution on the centering position behavior of the agent.

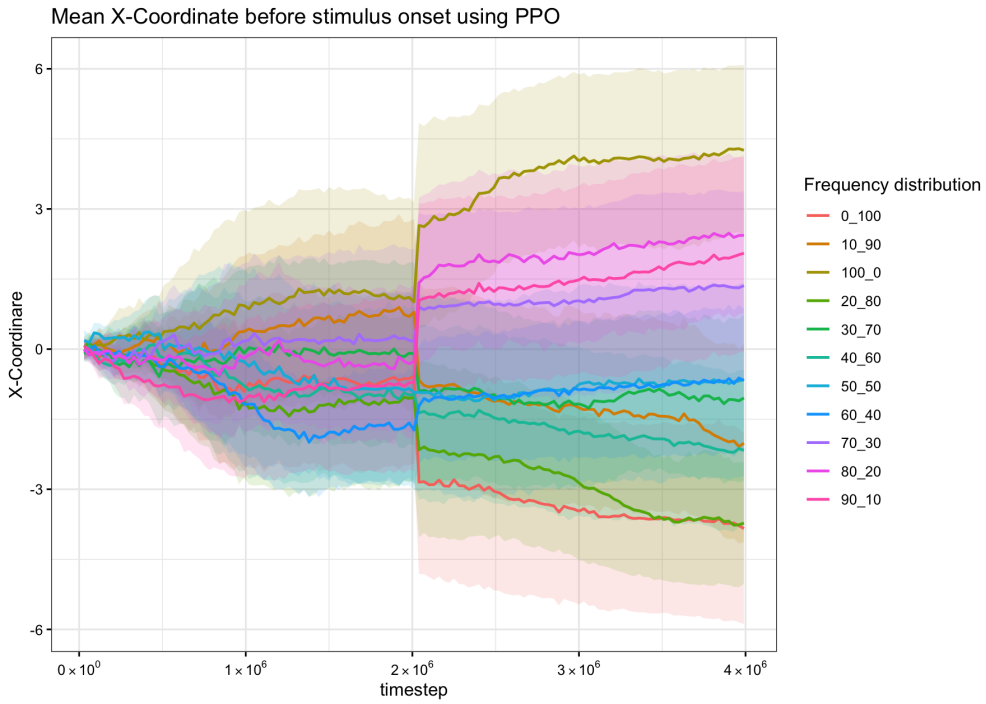



**Figure 5**

*Mean X-coordinate and 95% CI of the agent hand at the end of the ISI over time. The change in frequency distribution was applied at timestep* $2x10^6$ *for PPO.*

Figure 6 displays the X-coordinate of the agent's hand at the end of the ISI at the

end of the training. The different colors in the graphs represent different frequency conditions. For every condition, there seems to be a cluster around a certain X-coordinate, which seems to increase as the probability of the X-coordinate for target stimuli being 9 increases.

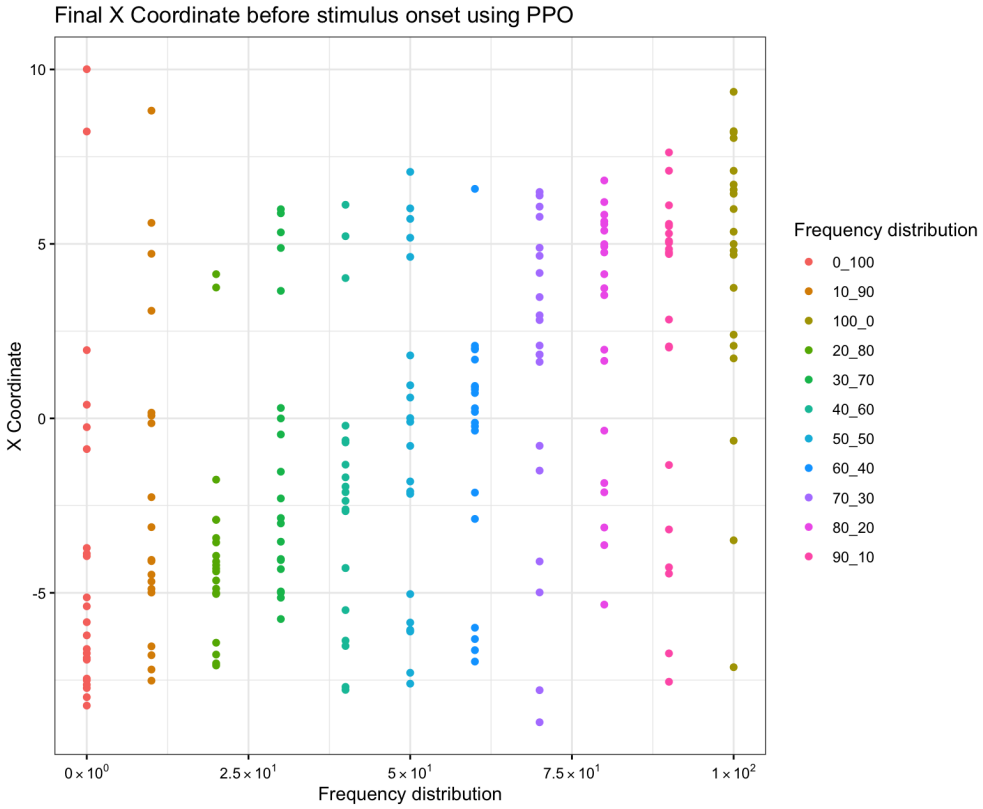



**Figure 6**

*X-Coordinates at the end of the ISI at the end of the training over frequency of the X-coordinate for target stimuli being 9 for PPO.*

First, a linear regression was performed with the frequency of the X-coordinate for target stimuli being 9 as predictor and the X-coordinate of the agent before stimulus onset after training as response variable. This resulted in the following fitted linear regression model: $\hat{y} = -4.07 + 0.074 * x$. The overall model was found to be statistically significant ($R2 = 0.24$, $F(1, 220) = 68.88$, $p = 1.056\text{e-}14$). Predictor variable X-coordinate for target stimuli being 9 significantly predicted X-coordinate of the agent before stimulus onset after

training ($\beta = 0.074$, $p = 1.06\text{e-}14$) This indicated that frequency distribution is a significant predictor for the centering position of the agent.

Next, an ANOVA was conducted with frequency distribution as factor and position of the mean X-coordinate of the agents before stimulus onset after training as response variable. The ANOVA revealed that there was a statistical significant difference in the centering position between at least two groups ($F(10, 211) = 7.689$, $p = 1.858\text{e-}10$). Table 2 contains a complete summary of the model. In this table it can be seen that, for most conditions, the further away the frequency distribution coefficient is from the base (the 0-100 condition in this case), the higher the t-value seems to be in general.

**Table 2**

**ANOVA summary for the PPO algorithm**

Coefficients:

| | Estimate | Std. Error | t value | Pr(>\|t\|) |
|---|---|---|---|---|
| (Intercept) | -3.8406 | 0.8874 | -4.328 | 2.32e-05 *** |
| 10-90 | 1.8269 | 1.3193 | 1.385 | 0.1676 |
| 20-80 | 0.1171 | 1.3011 | 0.090 | 0.9283 |
| 30-70 | 2.7954 | 1.3011 | 2.148 | 0.0328 * |
| 40-60 | 1.6744 | 1.3011 | 1.287 | 0.1996 |
| 50-50 | 3.1935 | 1.3011 | 2.454 | 0.0149 * |
| 60-40 | 3.1676 | 1.3011 | 2.434 | 0.0157 * |
| 70-30 | 5.1995 | 1.3011 | 3.996 | 8.90e-05 *** |
| 80-20 | 6.2762 | 1.3011 | 4.824 | 2.70e-06 *** |
| 90-10 | 5.8931 | 1.3011 | 4.529 | 9.90e-06 *** |
| 100-0 | 8.0962 | 1.3011 | 6.222 | 2.60e-09 *** |
| Signif. codes: | 0 '***' | 0.001 '**' | 0.01 '*' | 0.05 '.' |

Now for the SAC algorithm, Figure 7 displays the effect of the frequency

distribution of the X-coordinate of the target stimulus on the centering behavior of the agent over time. As for PPO, the frequency distribution of a stimulus being either at X-coordinate 9 or -9 is 50-50 for the first half of the training. The figure shows means around 0 and overlapping CI's. After the curriculum onset, the graph shows a very clear distinction between the different frequency groups. In comparison to Figure 5, Figure 7 shows relatively narrow CI's, suggesting that the centering behavior for this trainer is more stable and precise. Since the CI's are very narrow and barely overlapping, it is very likely that the different frequency conditions come from different populations, suggesting a clear effect of frequency distribution on centering position behavior of the agent.

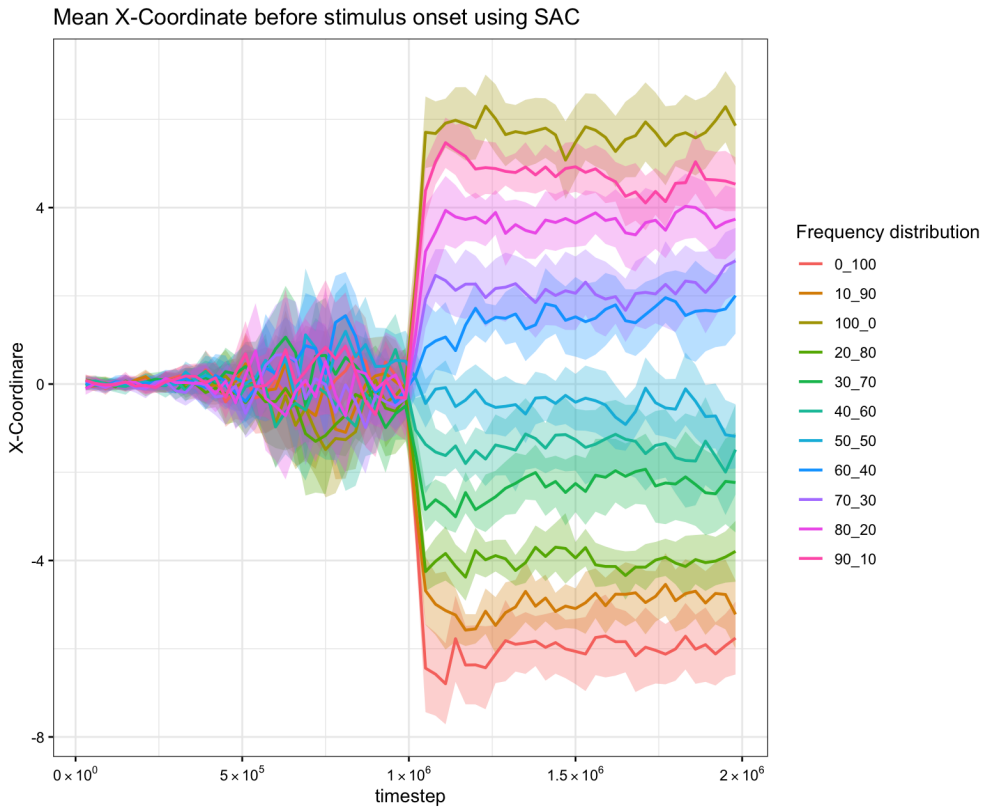



**Figure 7**

*Mean X-coordinate and 95% CI of the agent hand at the end of the ISI over time. The change in frequency distribution was applied at timestep* $1x10^6$ *for SAC.*

Figure 8 displays the X-coordinate of the agent's hand at the end of the ISI at the

end of the training. Again, the different colors in the graphs represent different frequency conditions. In this graph, we see very clear clusters around certain X-coordinates compared to Figure 6. Individual points deviate less from the overall clusters and the centers of these clusters increase as the probability of the X-coordinate for target stimuli being 9 increases.

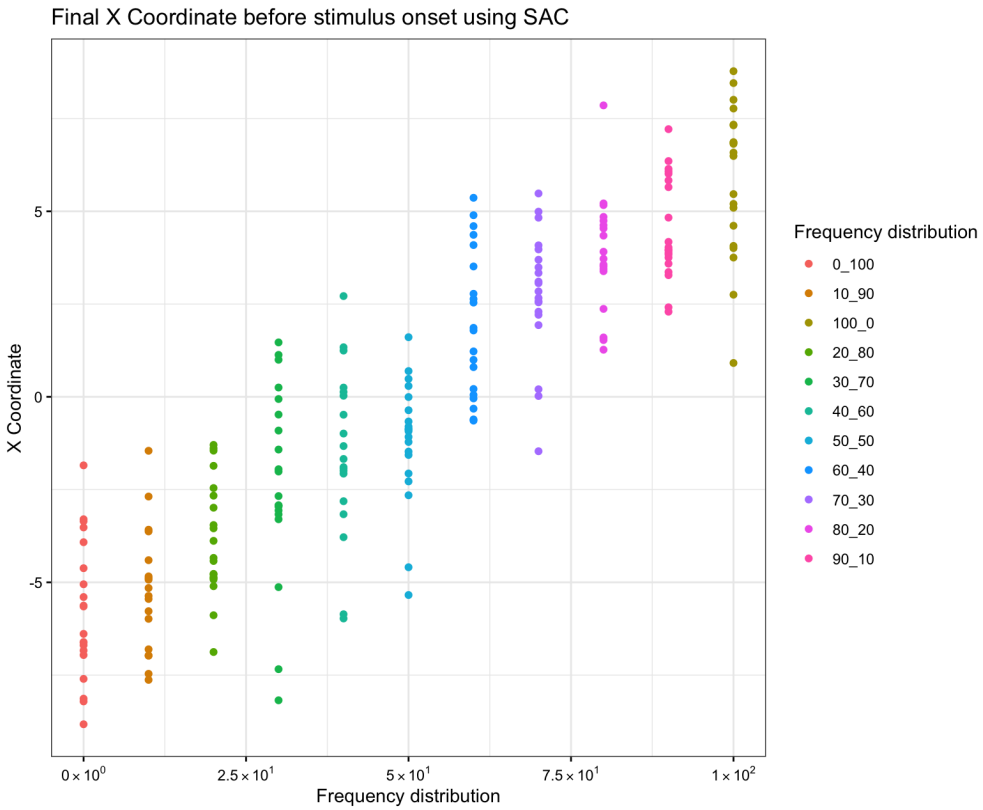



**Figure 8**

*X-Coordinates at the end of the ISI at the end of the training over frequency of the X-coordinate for target stimuli being 9 for PPO.*

As for the PPO, for SAC a linear regression was performed with the frequency of the X-coordinate for target stimuli being 9 as predictor and the X-coordinate of the agent before stimulus onset after training as response variable. This resulted in the following fitted linear regression model: $\hat{y} = -6.113972 + 0.121 * x$. The overall model was found to be statistically significant ($R2 = 0.80$, $F(1, 215) = 877.2$, $p <$ 2e-16). Predictor variable X-coordinate for target stimuli being 9 significantly predicted the X-coordinate of the agent

before stimulus onset after training ($\beta = 0.121$, $p <$ 2e-16). This indicated that frequency distribution is a significant predictor for the centering position of the agent.

Likewise, an ANOVA was conducted with frequency distribution as factor and position of the mean X-coordinate of the agents before stimulus onset after training as response variable. The ANOVA revealed that there was a statistical significant difference in the centering position between at least two groups ($F(10, 206) = 91.63$, $p <$ 2.2e-16). Table 3 contains a complete summary of the model. In line with the CI's in Figure 7, Table 3 shows that, except for the direct neighboring frequency distributions, the different frequency conditions significantly differ from each other.

**Table 3**

**ANOVA summary for the SAC algorithm**

Coefficients:

| | Estimate | Std. Error | t value | Pr(>\|t\|) |
|---|---|---|---|---|
| (Intercept) | -5.7586 | 0.4195 | -13.729 | < 2e-16 *** |
| 10-90 | 0.5372 | 0.6094 | 0.881 | 0.37908 |
| 20-80 | 1.9663 | 0.5932 | 3.315 | 0.00108 ** |
| 30-70 | 3.5261 | 0.5932 | 5.944 | 1.17e-08 *** |
| 40-60 | 4.2716 | 0.6009 | 7.108 | 1.89e-11 *** |
| 50-50 | 4.5734 | 0.5932 | 7.710 | 5.28e-13 *** |
| 60-40 | 7.7639 | 0.5932 | 13.088 | < 2e-16 *** |
| 70-30 | 8.5524 | 0.5932 | 14.418 | < 2e-16 *** |
| 80-20 | 9.4969 | 0.5932 | 16.010 | < 2e-16 *** |
| 90-10 | 10.2907 | 0.5932 | 17.348 | < 2e-16 *** |
| 100-0 | 11.6154 | 0.5932 | 19.581 | < 2e-16 *** |
| Signif. codes: | 0 '***' | 0.001 '**' | 0.01 '*' | 0.05 '.' |

Of course, the actual most profitable resting state in terms of least movement

acquired to reach the targets (the optimal centering position) can be calculated by adding the probability of the target being -9 multiplied by -9 to the probability of the target being 9 multiplied by 9. Table 4 contains these numbers together with the found X-coordinates before target onset at the end of the training of PPO and SAC. A visual comparison of the found X-coordinates can be seen in Figure 9. Respectively, the optimal position for the Y-coordinate and Z-coordinate are 9 and 0. The euclidean distance between the hand of the agent and the optimal centering position can be calculated by:

$$\sqrt{(x_{hand} - x_{opt})^2 + (x_{hand} - x_{opt})^2 + (x_{hand} - x_{opt})^2}. \tag{17}$$

Since previous evidence suggested more reliable centering behaviour for the SAC algorithm, a one sided $t-$test was conducted. The independent sample $t-$test confirmed that, at the end of the training, the SAC algorithm ($M = 3.35$) displayed a smaller euclidean distance from the optimal centering position than the PPO algorithm ($M = 8.84$), $t(260) = 15.754$, $p < 2.2\text{e-}16$. This indicates that overall, the SAC trainer is superior to the PPO trainer in terms of approaching the optimal learnt centering behavior.

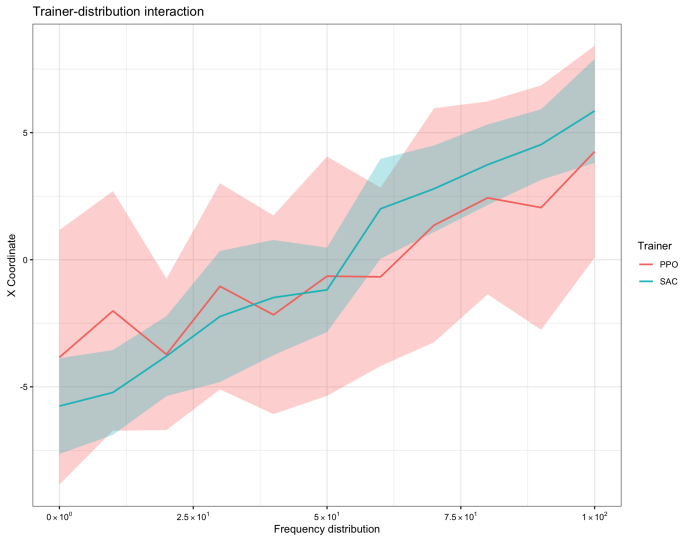



**Figure 9**

*Mean and SD of X-coordinates before stimulus onset after training over frequency distribution for PPO and SAC.*

**Table 4**

**Final centering position for each frequency distribution.**

| Distribution | PPO | SAC | Optimal |
|---|---|---|---|
| 0-100 | -3.8405977 | -5.758629 | -9.0 |
| 10-90 | -2.0137252 | -5.221407 | -7.2 |
| 20-80 | -3.7234513 | -3.792315 | -5.4 |
| 30-70 | -1.0452287 | -2.232568 | -3.6 |
| 40-60 | -2.1662039 | -1.487052 | -1.8 |
| 50-50 | -0.6471243 | -1.185278 | 0.0 |
| 60-40 | -0.6730011 | 2.005317 | 1.8 |
| 70-30 | 1.3589230 | 2.793791 | 3.6 |
| 80-20 | 2.4355640 | 3.738274 | 5.4 |
| 90-10 | 2.0525024 | 4.532027 | 7.2 |
| 100-0 | 4.2556401 | 5.856760 | 9.0 |

## Discussion

In this study, the scope of centering behavior on the SRT task was modeled and investigated using 3D deep RL agents. The performance of two different algorithms: PPO and SAC, was explored and compared on the task. Foremost, the influence of changing the probability distribution of the target stimuli on resting position during the ISI of the Reacher agents was investigated; Which was expected to shift from the center towards the stimuli that appeared more frequent.

Comparing overall SRT performance in terms of cumulative reward per episode, in this experiment, SAC clearly outperformed PPO with a higher mean cumulative reward and less variability. In Figures 3a & 3b, both algorithms seemed to plateau nearing the end of the training, while maintaining a constant learning rate, suggesting that this is indeed

the number of steps needed for training. As expected, given the known data efficiency of off-policy algorithms versus on-policy (Haarnoja, Zhou, Hartikainen, et al., 2018; Nachum et al., 2018), SAC converged a lot faster –in terms of timesteps– than the PPO algorithm; in real-time, the PPO algorithm was faster which was also expected since on-policy algorithms usually spend less time updating. What was not expected, was the high variability of the PPO algorithm, since on-policy algorithms tend to be more stable (Nachum et al., 2018). However, the SAC algorithm also aims for stability by using the exponential moving average to update the soft Q-function (Haarnoja, Zhou, Hartikainen, et al., 2018; Mnih et al., 2015) also seen in Equation 15. The success of this intervention could be the explanation for why the SAC algorithm is less variable than the PPO algorithm in this paradigm. Interestingly, the on-policy algorithm SARSA was also found to be more variable than its off-policy counterpart Q-learning (Kachergis et al., 2016). This might hint that human learning might be more related to off-policy learning.

For both the PPO and SAC algorithms frequency distribution of the target stimuli was a significant predictor of the resting position of the hand of the agent. This indicates that the centering position the agent displayed in previous research (de Kleijn et al., 2022; Sen et al., 2022) was not just a trivial resting state, but that the Reacher agents use their implicit acquired knowledge about the statistical properties –i.e. probability distributions– of the target stimuli to anticipate these stimuli and predict the ISI position to adopt in order to maximize the cumulative rewards. This reinforces the belief that centering behavior is indeed used as a way to compensate for the absence of sequence knowledge, using the center as optimal anticipatory position, as speculated in previous human research (Dale et al., 2012; de Kleijn et al., 2022; Duran & Dale, 2009).

For the SAC algorithm the distinction between frequency conditions is abundantly clearer as can be seen in both the Figures 5 & 7 and ANOVA tables 2 & 3. While the SAC algorithm has no trouble differentiating non-direct neighboring conditions, the PPO algorithm seems to struggle more in distinguishing the different frequency conditions.

Again the PPO algorithm shows high variability, which was also noticed comparing overall SRT performance. The euclidean distance to the optimal anticipatory position was found to be significantly less for the SAC algorithm compared to PPO. This effect, together with the lower rewards for global performance of PPO, is in line with human trials where it was found that humans that adhered to a better centering strategy under uncertainty would perform better on the overall SRT task (Dale et al., 2012; de Kleijn et al., 2018; Duran & Dale, 2009).

An explanation of SAC performing better than PPO might be the entropy term that is added to the model. Because of the entropy term, the policy is able to capture various approaches of near-optimal solutions, which makes the agent less vulnerable to environmental changes we also see in real-world problems (Tang & Haarnoja, 2017). In the current simulation, the order of the target stimuli is randomized and keeps changing, therefore the entropy term of SAC might also help the algorithm switch tactics in this case. A limitation, however, is that a lot of hyperparameters of SAC are less critical to the learning process than for PPO (Juliani et al., 2020). SAC specifically aims to counter the brittleness of hyperparameters, making them less critical and easier to tune than other algorithms like PPO (Haarnoja, Zhou, Hartikainen, et al., 2018). Meaning that it is more likely that PPO performance might be compromised by its hyperparametrisation than SAC. Therefore, the difference if performance in this study may be partially influenced by the hyperparametrization of said algorithms. More experimenting and tuning of the PPO hyperparameters algorithm might produce different results in further research.

Also noticeable, when looking at the differences in actual optimal anticipatory position and found anticipatory position for both PPO and SAC in Table 4, is that the more extreme the frequency distance becomes, the larger the distance between the optimal and found position becomes. One might expect that it is compelling the move the resting position more toward certain stimuli when it is clearly more likely to be (or even always) activated, both algorithms seem to be more cautious.

A possible explanation lies in the curriculum that was employed. The current curriculum first teaches the agent with an evenly balanced distribution in order to create a baseline. Research of de Kleijn et al., 2022 demonstrated a critical learning period for deep RL agents, after which the agents are less flexible to incorporate new objectives. This might imply that in the current study the critical learning period had passed and the agent was not flexible enough to incorporate objectives differing too much from the initial one. However, a striking difference is that the current study uses a constant learning rate instead of linearly declining towards 0 at the end of the training. Meaning that in this study experiences learned at later moments in training weigh as much as earlier experiences. Nonetheless, future research could verify this by manipulating the curriculum and/or changing the learning rate to linear as seen in de Kleijn et al., 2022.

In this study, the frequency distribution of stimuli was manipulated on one axis only. In future studies, the probability distribution of more axes could be manipulated or even the positions of the target stimuli to further investigate the scope of centering behavior or the simulated Reacher agents. Another way to investigate centering behavior would be to give them explicit knowledge about the probability distribution as predictive information instead of training the agents to implicitly learn the distribution, as is done with sequence information in the study of Sen et al., 2022. In said study, the agents receive information about the upcoming target stimulus by hot-coding its coordinates in the input vector of the agent. In further research the agents could be given the probability distribution of the target stimulus –while changing the probability distribution during the trial– and thus investigated if explicit knowledge of this probability distribution influences the centering behavior. Furthermore, to help model sequential decision making processes in humans, parallel Reacher and human studies should be conducted and compared. The synergy between simulated and human studies is important as this could help understanding human learning, along with the development of AI algorithms, and therefore, bridge the gap between human and machine learning (Botvinick et al., 2020; François-Lavet et al., 2018).

## Conclusion

Driven by the quest of modeling and explaining human sequential learning behavior, this study investigated and compared the scope of centering behavior for deep RL algorithms PPO and SAC in a 3D simulated environment of the SRT task. Although SAC evidently outperformed PPO in terms of performance and stability, it was found that in both algorithms the frequency distribution of the target stimuli was a vital influence on the resting position –which shifts proportionally towards more probable target stimuli– of the agent in the ISI. This indicates that centering behavior is not just a resting position, but also acts as an optimal anticipatory position, caused by predictive processes, between the target stimuli under uncertainty. Future research and comparison with human anticipatory actions should be able to shed a light on how exactly these results relate to human sequential learning behavior.

## References

Aslin, R. N. (2017). Satistical learning: A powerful mechanism that operates by mere exposure. *Wiley Interdisciplinary Reviews: Cognitive Science*, *8*(1-2), e1373.

Barto, A. G., Sutton, R. S., & Watkins, C. (1989). Sequential decision problems and neural networks. *Advances in Neural Information Processing Systems*, *2*.

Bell-Berti, F., & Harris, K. S. (1979). Anticipatory coarticulation: Some implications from a study of lip rounding. *The Journal of the Acoustical Society of America*, *65*(5), 1268–1270.

Bellman, R. (1957). *Dynamic programming*. Princeton University Press.

Botvinick, M., Wang, J. X., Dabney, W., Miller, K. J., & Kurth-Nelson, Z. (2020). Deep reinforcement learning and its neuroscientific implications. *Neuron*, *107*(4), 603–616. https://doi.org/https://doi.org/10.1016/j.neuron.2020.06.014

Dale, R., Duran, N. D., & Morehead, J. R. (2012). Prediction during statistical learning, and implications for the implicit/explicit divide. *Advances in Cognitive Psychology*, *8*(2), 196.

de Kleijn, R., Kachergis, G., & Hommel, B. (2018). Predictive movements and human reinforcement learning of sequential action. *Cognitive Science*, *42*, 783–808.

de Kleijn, R., Sen, D., & Kachergis, G. (2022). A critical period for robust curriculum-based deep reinforcement learning of sequential action in a robot arm. *Topics in Cognitive Science*, *14*(2), 311–326.

Duran, N., & Dale, R. (2009). Predictive arm placement in the statistical learning of position sequences. *Proceedings of the Annual Meeting of the Cognitive Science Society*, *31*(31).

François-Lavet, V., Henderson, P., Islam, R., Bellemare, M. G., Pineau, J., et al. (2018). An introduction to deep reinforcement learning. *Foundations and Trends® in Machine Learning*, *11*(3-4), 219–354.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT Press.

Haarnoja, T., Pong, V., Hartikainen, K., Zhou, A., Dalal, M., & Levine, S. (2018). Soft actor critic—deep reinforcement learning with real-world robots.

Haarnoja, T., Zhou, A., Abbeel, P., & Levine, S. (2018). Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *International Conference on Machine Learning*, 1861–1870.

Haarnoja, T., Zhou, A., Hartikainen, K., Tucker, G., Ha, S., Tan, J., Kumar, V., Zhu, H., Gupta, A., Abbeel, P., et al. (2018). Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*.

Hassabis, D., Kumaran, D., Summerfield, C., & Botvinick, M. (2017). Neuroscience-inspired artificial intelligence. *Neuron*, *95*(2), 245–258. https://doi.org/https://doi.org/10.1016/j.neuron.2017.06.011

Hunt, R. H., & Aslin, R. N. (2001). Statistical learning in a serial reaction time task: Access to separable statistical cues by individual learners. *Journal of Experimental Psychology: General*, *130*(4), 658.

Jozefowiez, J. (2002). Reinforcement learning and conditioning: An overview.

Juliani, A., Berges, V., Teng, E., Cohen, A., Harper, J., Elion, C., Goy, C., Gao, Y., Henry, H., Mattar, M., & Lange, D. (2020). Unity: A general platform for intelligent agents. *arXiv preprint arXiv:1809.02627*. https://doi.org/https://github.com/Unity-Technologies/ml-agents.

Kachergis, G., Berends, F., Kleijn, R. d., & Hommel, B. (2016). Human reinforcement learning of sequential action.

Konda, V., & Tsitsiklis, J. (1999). Actor-critic algorithms. *Advances in Neural Information Processing Systems*, *12*.

Lewandowsky, S., & Farrell, S. (2010). *Computational modeling in cognition: Principles and practice*. SAGE publications.

Li, Y. (2017). Deep reinforcement learning: An overview. *arXiv preprint arXiv:1701.07274*.

Mennie, N., Hayhoe, M., & Sullivan, B. (2007). Look-ahead fixations: Anticipatory eye movements in natural tasks. *Experimental Brain Research*, *179*(3), 427–442.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., & Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. *International Conference on Machine Learning*, 1928–1937.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015). Human-level control through deep reinforcement learning. *Nature*, *518*(7540), 529–533.

Nachum, O., Gu, S. S., Lee, H., & Levine, S. (2018). Data-efficient hierarchical reinforcement learning. *Advances in Neural Information Processing Systems*, *31*.

Nissen, M. J., & Bullemer, P. (1987). Attentional requirements of learning: Evidence from performance measures. *Cognitive Psychology*, *19*(1), 1–32.

Niv, Y. (2009). Reinforcement learning in the brain. *Journal of Mathematical Psychology*, *53*(3), 139–154.

Pavlov, I. P. (1927). *Conditioned reflexes*. New York: Dover.

Perrusquia, A., Yu, W., & Soria, A. (2019). Large space dimension reinforcement learning for robot position/force discrete control. *2019 6th International Conference on Control, Decision and Information Technologies (CoDIT)*, 91–96.

Rescorla, R. A. (1972). A theory of pavlovian conditioning: Variations in the effectiveness of reinforcement and nonreinforcement. *Current Research and Theory*, 64–99.

Rummery, G. A., & Niranjan, M. (1994). *On-line q-learning using connectionist systems* (Vol. 37). University of Cambridge, Department of Engineering Cambridge, UK.

Saffran, J. R., Aslin, R. N., & Newport, E. L. (1996). Statistical learning by 8-month-old infants. *Science*, *274*(5294), 1926–1928.

Schulman, J., Levine, S., Abbeel, P., Jordan, M., & Moritz, P. (2015). Trust region policy optimization. *International Conference on Machine Learning*, 1889–1897.

Schulman, J., Moritz, P., Levine, S., Jordan, M., & Abbeel, P. (2015). High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Sen, D., de Kleijn, R., & Kachergis, G. (2022). Behavioral optimization in a robotic serial reaching task using predictive information. *IEEE Transactions on Cognitive and Developmental Systems*.

Skinner, B. (1938). The behavior of organisms. *American Psychologist*, *221*, 233.

Sutton, R. S., & Barto, A. G. (1987). A temporal-difference model of classical conditioning. *Proceedings of the Ninth Annual Conference of the Cognitive Science Society*, 355–378.

Sutton, R. S., Barto, A. G. et al. (1998). Introduction to reinforcement learning.

Tang, H., & Haarnoja, T. (2017). Learning diverse skills via maximum entropy deep reinforcement learning. *URL http://bair. berkeley. edu/blog/2017/10/06/soft-q-learning*.

Vapnik, V. N. (1999). An overview of statistical learning theory. *IEEE Transactions on Neural Networks*, *10*(5), 988–999.

Watkins, C. J. C. H. (1989). *Learning from delayed rewards*. King's College, Cambridge United Kingdom.

Wilson, R. C., & Collins, A. G. (2019). Ten simple rules for the computational modeling of behavioral data. *Elife*, *8*, e49547.

Zhang, L., Lengersdorff, L., Mikus, N., Gläscher, J., & Lamm, C. (2020). Using reinforcement learning models in social neuroscience: Frameworks, pitfalls and suggestions of best practices. *Social Cognitive and Affective Neuroscience*, *15*(6), 695–707.

Ziebart, B. D. (2010). *Modeling purposeful adaptive behavior with the principle of maximum causal entropy*. Carnegie Mellon University.

Ziebart, B. D., Maas, A. L., Bagnell, J. A., Dey, A. K., et al. (2008). Maximum entropy inverse reinforcement learning. *Aaai*, *8*, 1433–1438.

# Appendix

## Appendix A

**Table A1**

**Hyperparameters of PPO and SAC**

| Parameters | PPO | SAC |
|---|---|---|
| Batch size | 2024 | 256 |
| Buffer size | 20240 | 500000 |
| Learning rate | 0.0003 | 0.0003 |
| Learning schedule | constant | constant |
| Beta | 0.005 | - |
| Epsilon | 0.2 | - |
| lambd | 0.95 | - |
| Tau | - | 0.005 |
| Init Alpha | - | 1.0 |
| Gamma | 0.995 | 0.995 |
| Max steps | 4000000 | 2000000 |
| Time Horizon | 4000 | 4000 |