Specialization project

# Evaluation of REINFORCE on benchmark tasks for Panda in RLBench

Krishnan Jothi Ramalingam
Matrikelnummer: 4997812

Technische Universität Braunschweig
Institut für Robotik und Prozessinformatik
Mühlenpfordtstraße 23 · 38106 Braunschweig

Prüfer: Prof. Dr. Jochen Steil
Betreuer: Prof. Dr. Jochen Steil

**Why Reinforcement Learning?**

- Model-based methods like Dynamic Programming cannot be used to train the agent, since a complete and accurate model of the environment is not available.

- Reinforcement Learning - An agent interacts with its environment and tries to maximize the accumulated reward over its lifetime without relying on exemplary supervision or complete knowledge of the environment model.

- MDP formally describe an environment for reinforcement learning.

  - ❖ Assumption : The state and reward are Markovian.
  - ❖ MDP policies fully define the behaviour of an agent, and it depend only on the current state.
  - ❖ Extensions to MDPs also exist. For ex., POMDP

# Background

**Policy Gradient Methods:**

- It uses function approximators to directly parameterize the following stochastic policy.

$$\pi(a|s,\theta) = Pr\{A_t = a|S_t = s, \theta_t = \theta\}$$

which gives the probability of choosing the action $a$ at time $t$ given that the environment is in state $s$, at time $t$, with parameter $\theta$, such that $\theta \in \mathbb{R}^d$, $\pi(a|s,\theta) \in (0,1)$ and $\nabla\pi(a|s,\theta)$ exists [SB18]. Policy gradient reinforcement learning is an optimization problem and the goal is to search for a local maximum in the policy objective function $J(\theta)$ by ascending the gradient of the policy with respect to parameter $\theta$.

- It can handle high-dimensional or even continuous state and action spaces.
- Whereas value function approximation leads to memory issues, and also learning the value of each state individually becomes too slow.

[SB18] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. second edition. Cambridge, MA, USA: A Bradford Book, 2018.

# Background

**Stochastic Policy:**

- It represents a conditional probability distribution.
- More suitable for continuous-control environments.
- Can handle the exploration/exploitation trade-off.
- Capable of handling the problem of perceptual aliasing (multiple indistinguishable states require different optimal actions).

**The Policy Gradient Theorem:** It gives an exact expression whose expectation approximates the gradient of the objective function.

$$\nabla J(\theta) = \mathbb{E}_\pi \left[ \sum_a q_\pi(S_t, a) \nabla \pi(a|S_t, \theta) \right]$$

$$q_\pi(S_t, a) = \mathbb{E}_\pi \left[ G_t | S_t, A_t \right]$$

where $\pi(a|s, \theta)$ is the parameterized policy and $q_\pi(S_t, a)$ is the action-value function, i.e., the expected return from the state $S_t$, by taking the action $A_t$, and then following policy $\pi(a|s, \theta)$.

**REINFORCE:**

- The classical algorithm in the family of policy gradient methods.
- It uses the policy gradient theorem to approximate the gradient of the objective function.
- The algorithm is well defined only for the episodic case.
- The main issue with this algorithm is high variance and thus produces slow learning.
- It is possible to introduce a baseline to decrease the variance and improve the learning speed.
- The baseline can be any function that is not a function of action $a_t$
- When baseline is introduced in an expectation, it does not modify the expected value and not introduces bias.
- Research shows that a good choice for the baseline function is an estimate of the state-value [Lex13]

[Lex13] Nigel Tao Lex Weaver. „The Optimal Reward Baseline for Gradient Based Reinforcement Learning“. In: *arXiv:1301.2315* (2013), pages 538–545.

**Pseudocode for REINFORCE**

**Input:** A differentiable parameterized policy $\pi(a|s,\theta), \theta \in \mathbb{R}^d$

**Initialization:**
Algorithm parameters: step size $\alpha > 0$, $\gamma \in [0,1]$
Initialize policy parameter $\theta$ arbitrarily.

**for** each episode:
  Generate an episode $S_0, A_0, R_1, ..., S_{T-1}, A_{T-1}, R_T \sim \pi(.|.,\theta)$
  **for** each step of the generated episode $t = 0, 1, ..., T-1$:
    $G_t \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$
    $\theta_{t+1} \leftarrow \theta_t + \alpha G_t \nabla \ln \pi(A_t|S_t,\theta_t)$
  **end for**
**end for**

**Pseudocode for REINFORCE with Baseline**

**Input:**
A differentiable parameterized policy $\pi(a|s,\theta), \theta \in \mathbb{R}^d$
A differentiable parameterized state-value function $\hat{v}(S_t, \mathbf{w}), \mathbf{w} \in \mathbb{R}^{d'}$

**Initialization:**
Algorithm parameters: step sizes $\alpha > 0$, $\beta > 0$, $\gamma \in [0, 1]$
Initialize policy parameter $\theta$ and state-value weights $\mathbf{w}$ arbitrarily.

**for** each episode:
  Generate an episode $S_0, A_0, R_1, ..., S_{T-1}, A_{T-1}, R_T \sim \pi(.|., \theta)$
  **for** each step of the generated episode $t = 0, 1, ..., T-1$:
    $G_t \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$
    $\delta \leftarrow G_t - \hat{v}(S_t, \mathbf{w})$
    $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + \beta\delta\nabla\hat{v}(S_t, \mathbf{w})$
    $\theta_{t+1} \leftarrow \theta_t + \alpha\delta\nabla \ln \pi(A_t|S_t, \theta_t)$
  **end for**
**end for**

# Motivation

- It is feasible to approach simple problems with traditional reinforcement learning algorithms without the need for large size visual data sets and high computational power since not all day-to-day simple tasks are automated by a robot yet, which reflects the necessity of better robot control policy call for better approaches.

- Even with proprioceptive inputs, the agent can learn complex behaviours and demonstrates remarkable zero-shot generalization from simulation to the real world [al 5], which motivated me to evaluate the performance of the classical policy gradient algorithm, on a 7 degree of freedom robot in a simulated environment using proprioceptive inputs to perform a benchmark task.

- This research is conducted on a ’blind’ Frank Emika Panda agent, using the Monte Carlo policy gradient REINFORCE algorithm on a benchmark task in RLBench environment.

[al 5] Joonho Lee et al. „Learning Quadrupedal Locomotion over Challenging Terrain“. In: *Science Robotics* (2020, Volume 5), pages 1–22.

# Motivation

- The motive is to understand the ability of the agent to learn to map optimal actions with only proprioceptive inputs, the limitations of REINFORCE, the possible measures to improve the capabilities, and the directions of future research.

- The agent is initially trained on a simplified benchmark task, with the prime motivation to validate the developed algorithm. Experiments are carried out on the simplified benchmark task to gain insights into the learning dynamics. Finally, the agent is trained on the actual task in RLBench, and the results are presented and discussed.

- In the field of reinforcement learning, the effect of learning rate decay has hardly been studied [CSW21], which further motivated me to compare and analyse the learning dynamics of REINFORCE algorithm, with both constant and decaying learning rates.

Yifei Chen, Lambert Schomaker, and Marco Wiering. „An Investigation Into the Effect of the Learning Rate on Overestimation Bias of Connectionist Q-learning“. In: *13th International Conference on Agents and Artificial Intelligence* (2021).

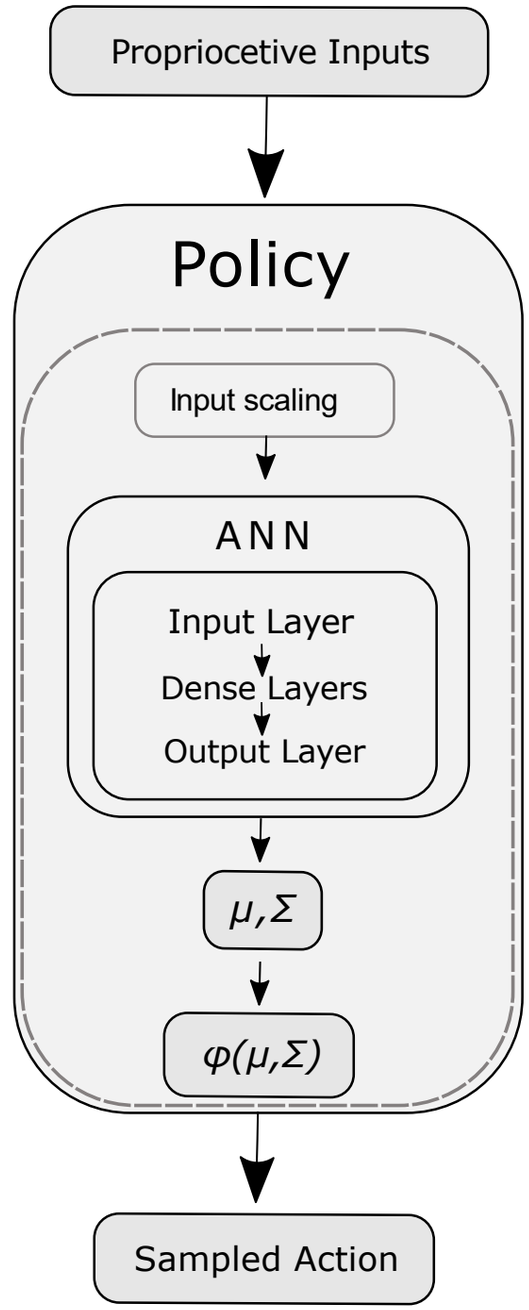
**Simulation Environment:**

- RLBench is a benchmark and learning environment for robot learning which can be used as a standard in place for comparing robotic manipulation methods.
- It includes 100 unique, hand-designed tasks, with arrays of both proprioceptive observations and visual observations.
- One of the key properties of RLBench is the tiered difficulty, i.e., the tasks that are designed are of different difficulty levels starting from easy tasks like reaching to challenging tasks, which can even stress-test well-known state-of-the-art algorithms in use today.
- The proprioceptive data of the agent include joint angles, velocities, torques, and the gripper pose.
- By default, the agent in the scene is Frank Emika Panda, a 7 DoF arm.
- Since REINFORCE is a classical approach and converges to a local optimum rather than a global optimum, the reach-target task is chosen.

# Implementation

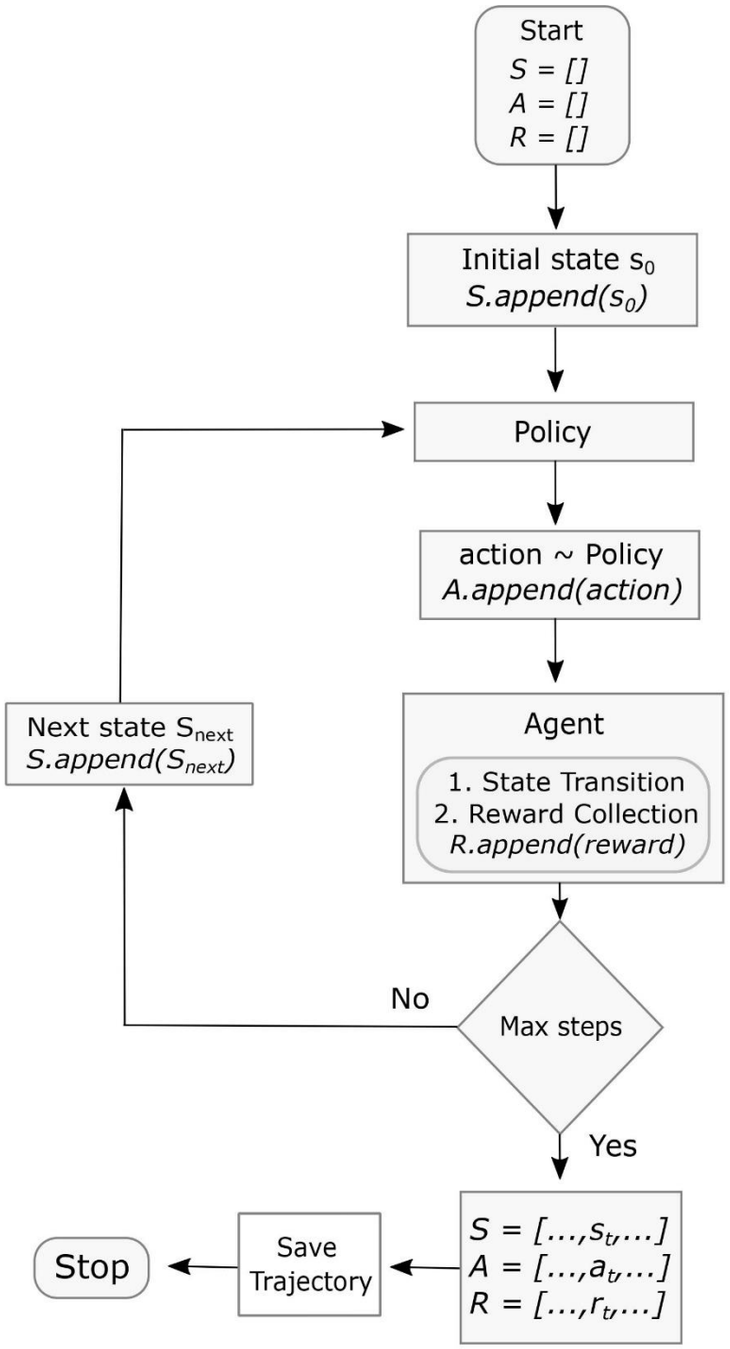
## Stochastic Policy Representation:

- The proprioceptive inputs are initially scaled and then passed forward through the dense layers of the feed-forward neural network.
- Assuming Gaussian distribution with diagonal covariance, the mean vector, and the covariance matrix are built from the output of the ANN.
- The action is sampled from the normal distribution built from the obtained mean vector and covariance matrix.
- ANN combined with the process of building the distribution refers to the policy of the agent.

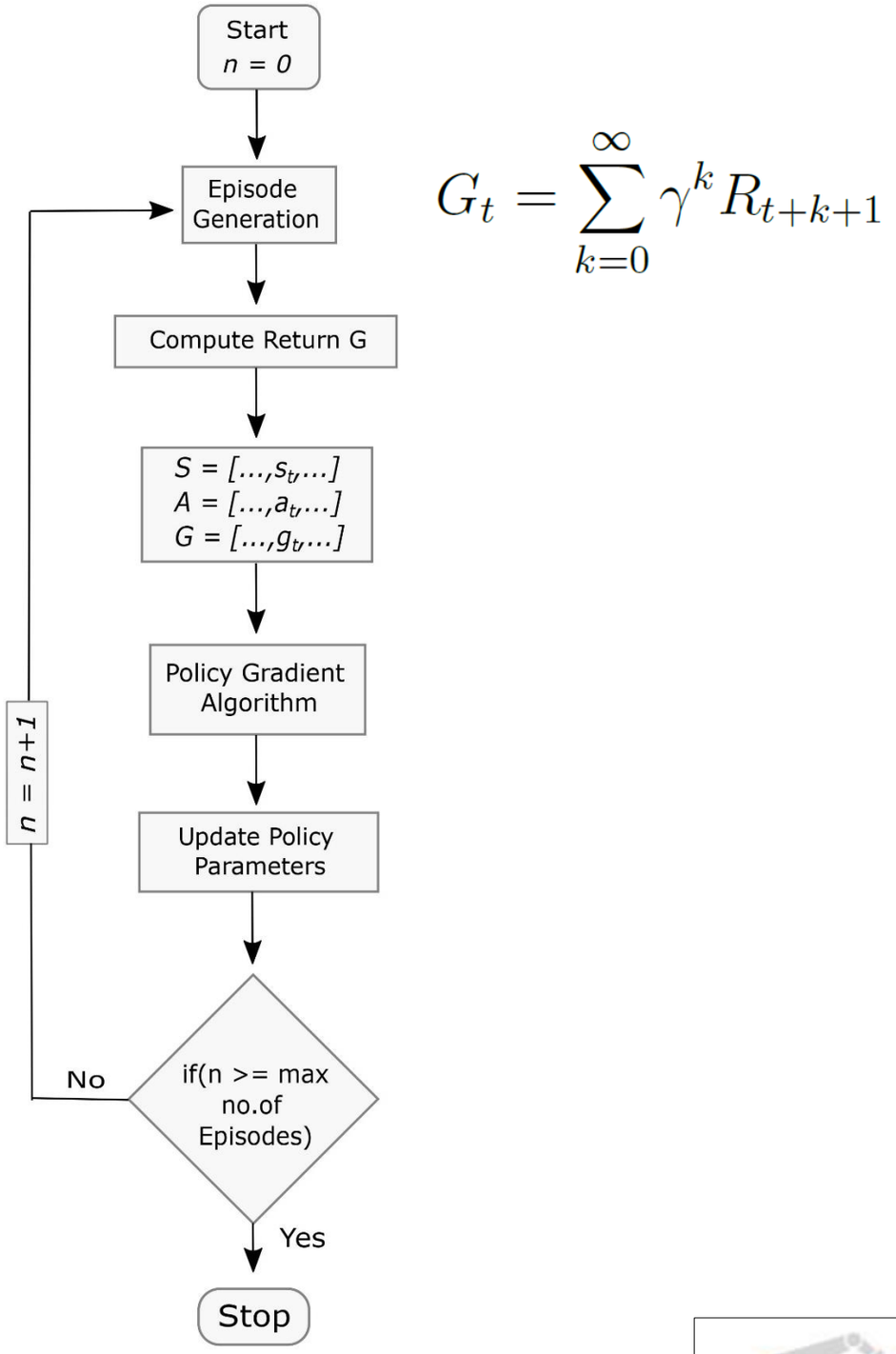


# Implementation

## Trajectory Generation:




## Algorithm Deployment:




$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

# Results and Discussion

**Architecture Details:**

- The inputs are normalized in [-1, 1]

- Actor network: The input layer size is 40. Three hidden layers, each containing 100 neurons, are connected using the relu activation function. The first 7 neurons of the output layer correspond to the mean vector elements of the Gaussian distribution, with the tanh activation. The last 7 neurons of the output layer (out of total 14 neurons) represent the diagonal elements of the covariance matrix, with the softplus activation since the covariance matrix is symmetric positive semidefinite with strictly-positive diagonal variances of the random vector [Bek88].

- Baseline network: The input layer size is 40. Three hidden layers, each containing 20 neurons, are connected using the relu activation function. The output is a single node, with softplus activation since it approximates the value of the state, which cannot be negative.

[Bek88] Paul A. Bekker. „The Positive Semidefiniteness of Partitioned Matrices “. In: *Elsevier Science Publishing Co., Inc* (1988), pages 261–278.

# Results and Discussion

**Architecture Details:**

- Optimizer: Adam
- Maximum steps per episode: 40
- Discount factor: 0.99
- Validation roll-outs: 500 roll-out
- Smoothing: Exponential moving average with a factor of 0.99
- Learning rates of actor and baseline networks are given in the following table:

| Actor | Baseline |
|---|---|
| $\alpha_1 = 1\text{e-}5$ | $\beta_1 = 1\text{e-}3$ |
| $\alpha_2 = 1\text{e-}6$ | $\beta_2 = 1\text{e-}4$ |
| $\alpha_3 = 1\text{e-}5$ | - |
| $\alpha_4 = 1\text{e-}6$ | - |
| $\alpha_5 = 1\text{e-}7$ | - |
| $\alpha_6 = 5\text{e-}7$ | $\beta_6 = 5\text{e-}5$ |

**Plots Description:**

- Success rate:

The success rate is measured on the fly during training, with 0.99 as a smoothing factor (exponential moving average). Each episode bin in the x-axis represents 20 episodes, starting from ((Episode bin*20)-20) to (Episode bin*20), e.g., the 5th episode bin represents the episodes from 80 to 100, and the respective y-axis value represents the percentage of those 20 episodes that succeeded the task. Even if the agent reaches the target once in a trajectory, then it is considered a success.

- Average Episodic Sampling Uncertainty:

In the y-axis, $-ln\ L(\boldsymbol{\mu}_{S_t}, \boldsymbol{\Sigma}_{S_t}|A_t)$ is averaged over all the actions $A_t$ of an episode, where $\boldsymbol{\mu}_{S_t}$ and $\boldsymbol{\Sigma}_{S_t}$ represents the mean vector and the covariance matrix built from the actor network predictions, for the given proprioceptive input $S_t$, and $L(\boldsymbol{\mu}_{S_t}, \boldsymbol{\Sigma}_{S_t}|A_t)$ represents the likelihood of a Gaussian with mean vector $\boldsymbol{\mu}_{S_t}$ and covariance matrix $\boldsymbol{\Sigma}_{S_t}$, given the action $A_t$. The exponential moving average with a factor of 0.99 is used to smooth the data.

# Results and Discussion

**Plots Description:**

- Validating the trained agent: For every n episodes, the agent is validated on 500 roll-outs, and the success rate is evaluated. Even if the agent reaches the target once in a roll-out, then it is considered a success.

- Average episodic return: The average of all the returns computed in a episode.

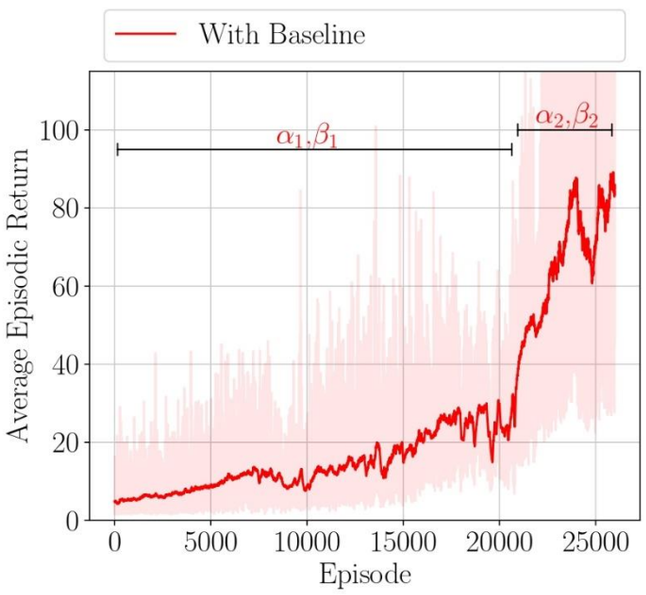
**Training on a simplified benchmark task:**

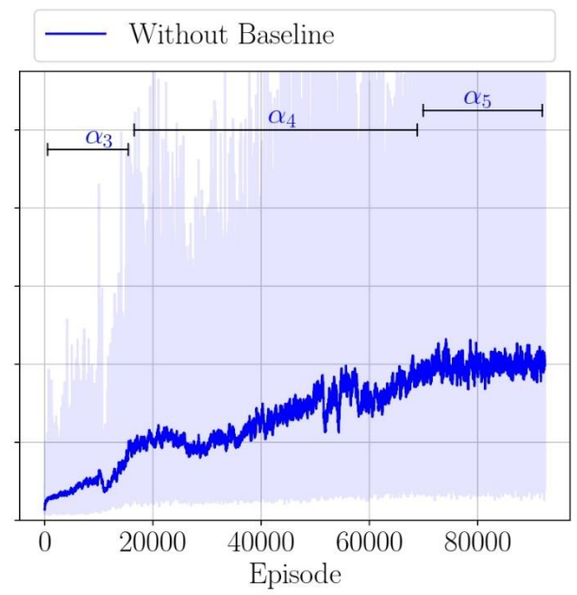
The following experiments are carried out:

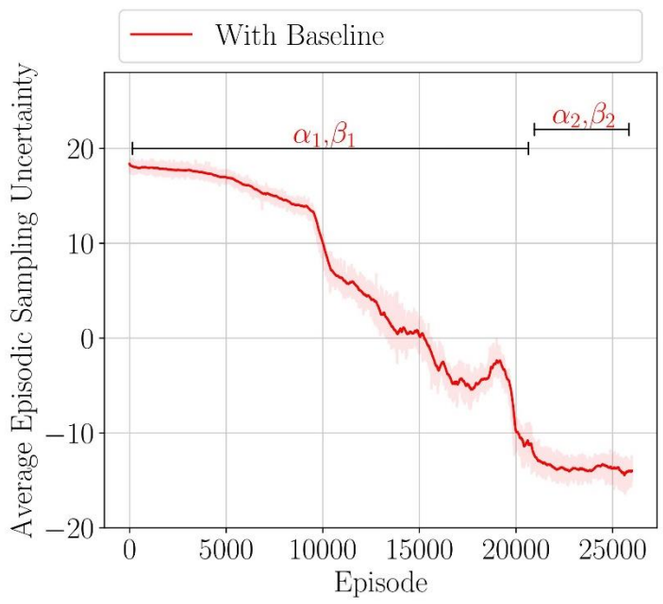
- Effect of baseline (validate the algorithm)
- Effect of learning rate decay

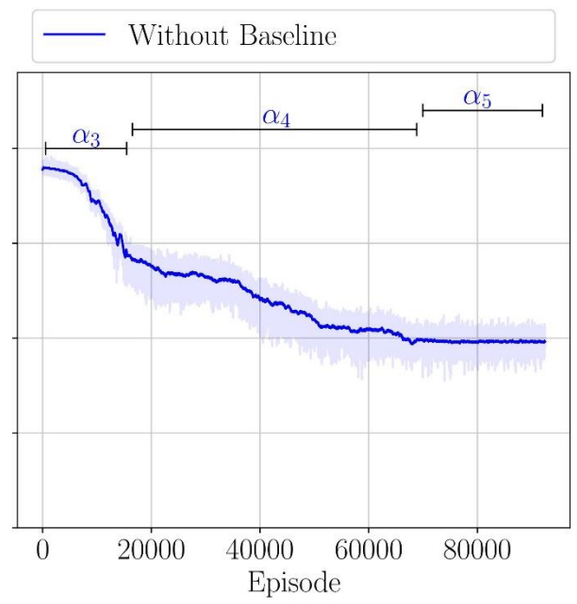
# Results and Discussion

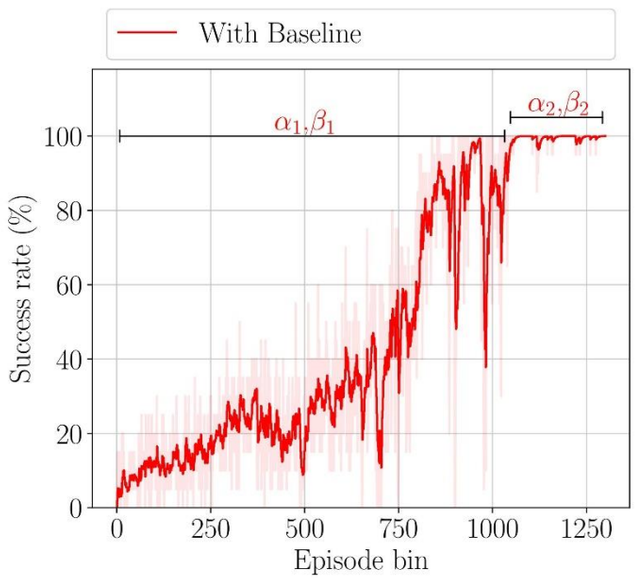


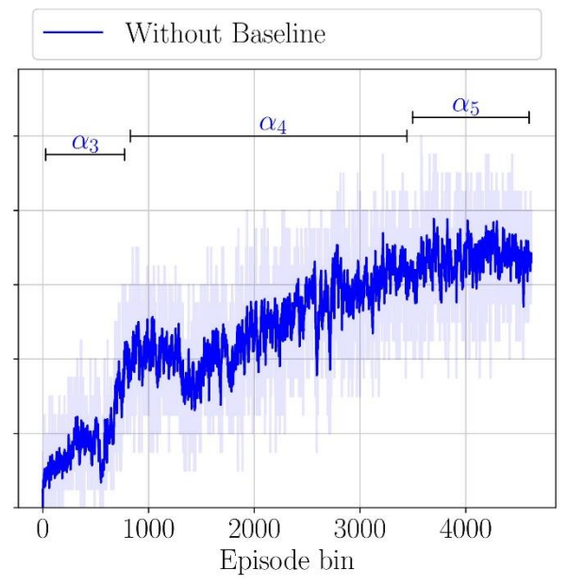



While training with baseline, the learning rate is modified at the 20800th episode. In the absence of a baseline, the learning rate is modified at 16000th and 69400th episodes.

# Results and Discussion

**Effect of baseline:**

- Plots show a significant reduction in the variance of the average episodic return due to the addition of the baseline function, and as a result, learning speed is notably improved.
- With a baseline, the agent was able to learn the target behaviour almost within 26000 episodes with a 100% success rate, but whereas in the absence of baseline, even after 90000 episodes, the success rate is around 70%.
- Even after attaining a 100% success rate, the agent can maintain it, which can be observed in the last 5000 episodes.
- The agent became more certain about its actions since the variance of the action distribution became low, leading to a high likelihood value.
- In the presence of a baseline, the rate of reduction in uncertainty of action sampling is comparatively high.
- The expected behaviour further validates the algorithm implementation.

# Results and Discussion

**Learning rate decay in reinforcement learning:**

- Recent findings in the field of reinforcement learning using Q-learning and Double Q-learning algorithms show that the best performances are achieved by decaying the learning rates [CSW21].

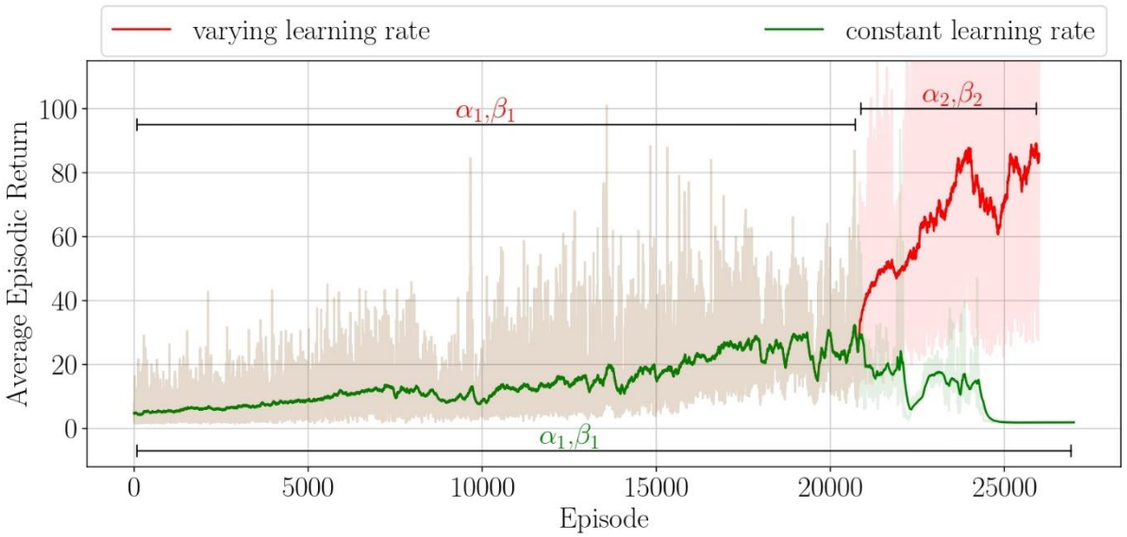
- The experiment is, if there is a continuous drop in the performance from a particular point in time during training, then the agent is retrained from that point with a lower learning rate (red curve). In deep learning, a similar technique is deployed to train the model for the image recognition task. The learning rate is divided by 10 when the error plateaus, i.e., if the change in the loss with respect to the training iterations is less than a threshold value. This learning rate decay scheduler resulted in better performance [He+15].

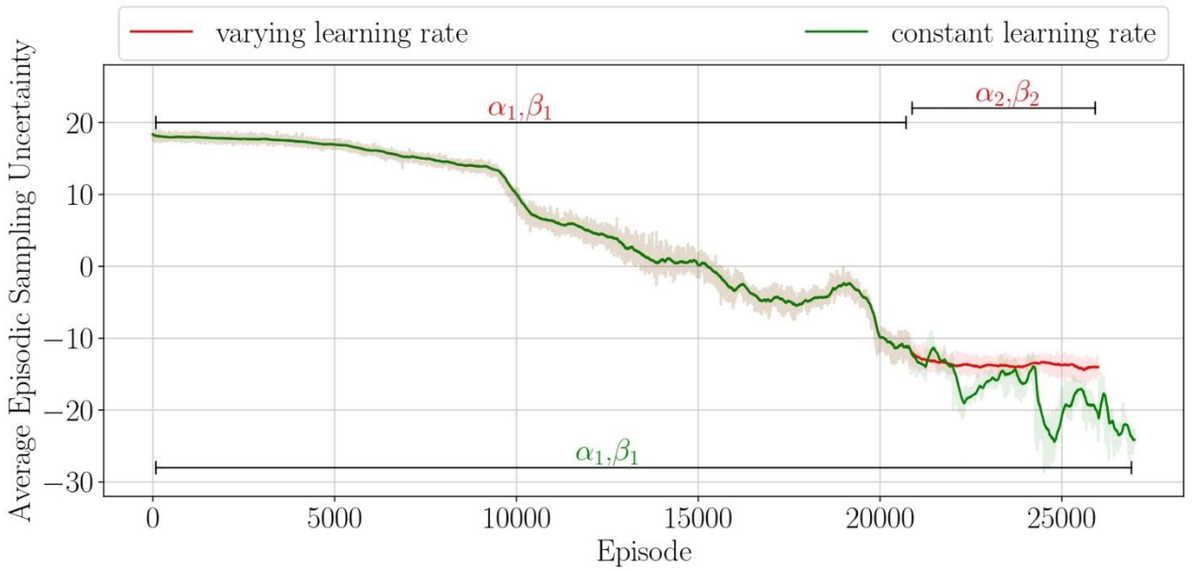
[CSW21] Yifei Chen, Lambert Schomaker, and Marco Wiering. „An Investigation Into the Effect of the Learning Rate on Overestimation Bias of Connectionist Q-learning“. In: *13th International Conference on Agents and Artificial Intelligence* (2021).

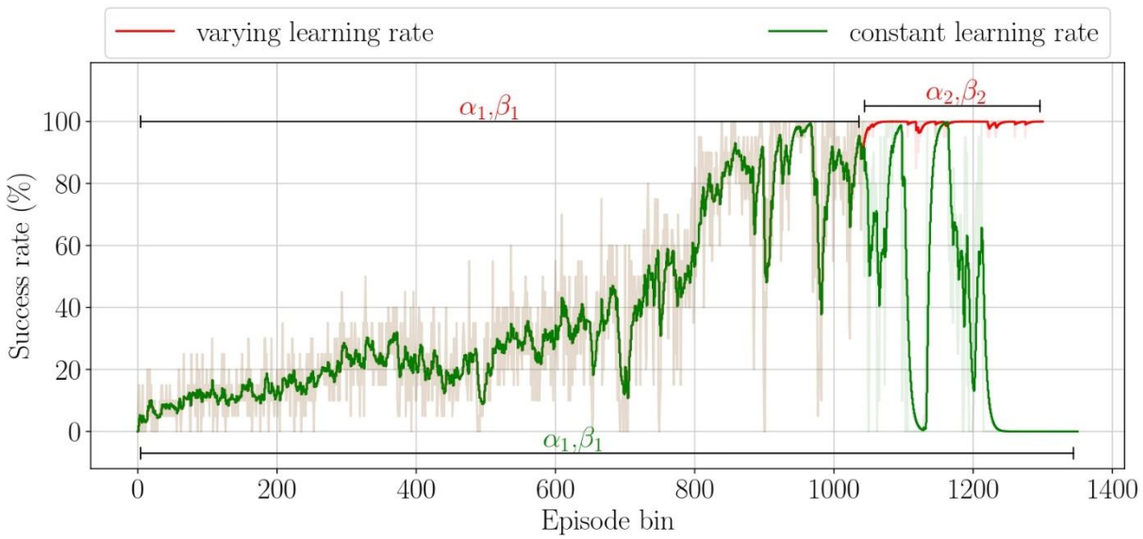
[He+15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. „Deep Residual Learning for Image Recognition“. In: *arXiv:1512.03385v1* (2015).

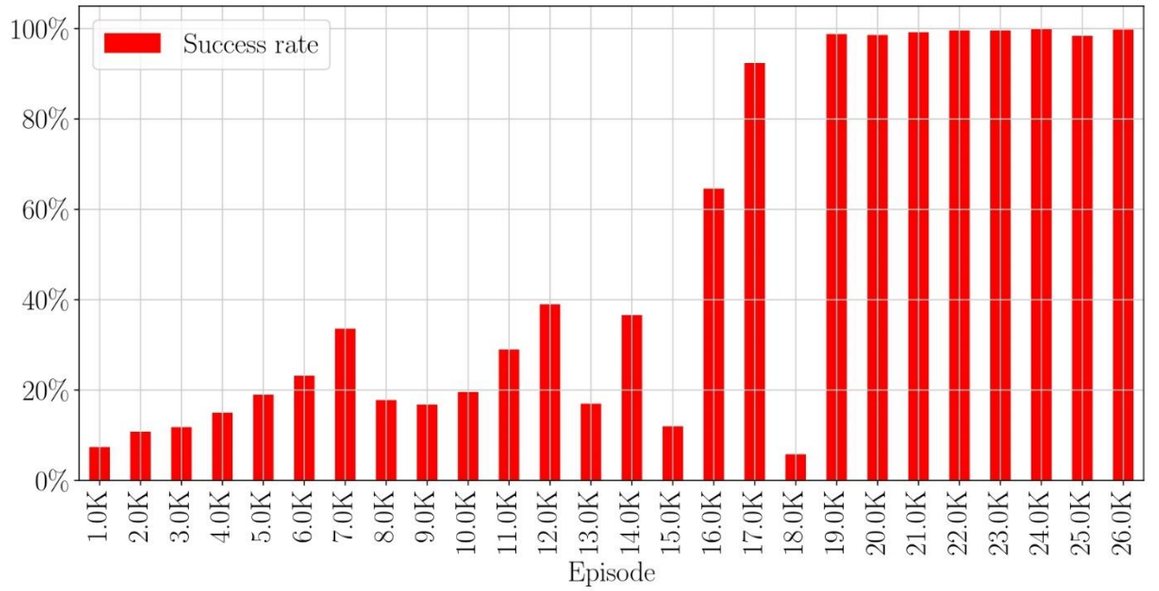
# Results and Discussion

## Effect of learning rate decay:



# Results and Discussion

**Effect of learning rate decay:**

- The plots show the change in learning dynamics after reducing the learning rate at the 20800th episode, and the curves in the plots overlap when they both have the same learning rate.

- After a certain point in time during training, if the learning rate is high, then the agent takes big steps and jumps around, and it becomes almost impossible for the agent to learn minute variations of the network parameters to fine-tune the model. Due to the big steps taken, the optimization problem diverges, and the agent learns a non-sense policy.

- Even if the agent is able to obtain 100% success rate with a constant learning rate (green curve), it is unable to maintain its performance since the learning rate became high after some point in time, and started to fluctuate and ended at 0%.

# Results and Discussion



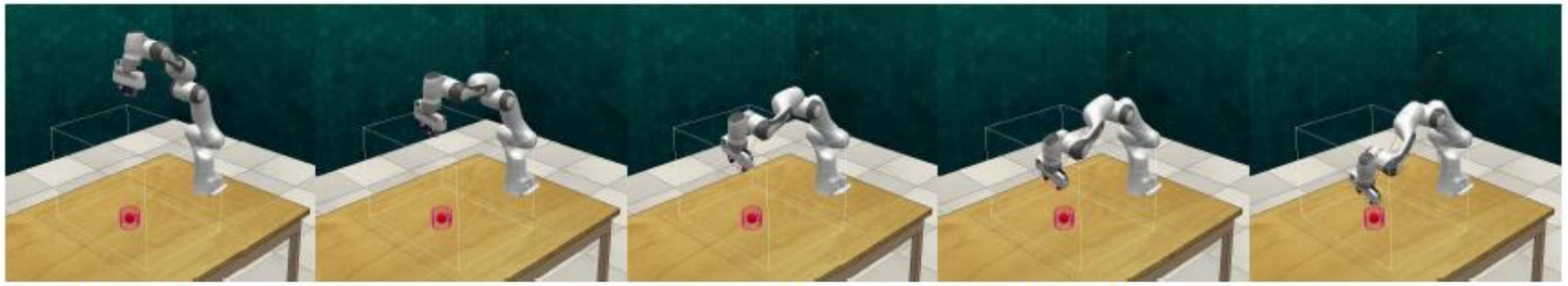
- In the case of the constant learning rate, around 25000 episodes, the variance of the average episodic return started to become very small, and hence it is barely visible, and after reaching a 0% success rate at the end of the training, there is no further improvement in its performance. Because after 20800 episodes, due to big learning steps (green curve), the agent is not only learning non-optimal policy but also becoming highly certain with its non-sense actions, i.e., in other words, the policy quickly becomes deterministic without sufficiently exploring.

- But even after obtaining a cent percent success rate, why there is a significant increase in average episodic return? What kind of additional behaviour did the agent learn in between 19000 and 26000 episodes?

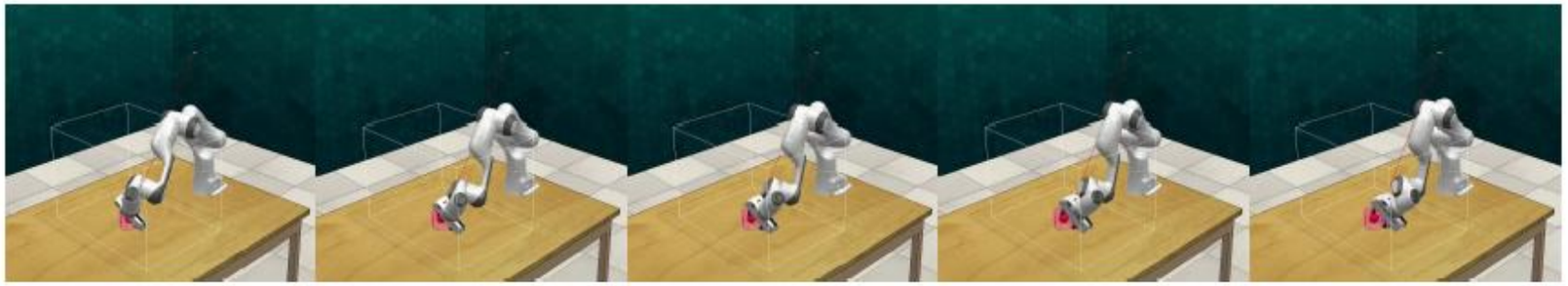
  ➢ Even after reaching the target ($8^{th}$ timeframe), the agent learned to stay at the target till the termination of the episode, in order to maximize the average episodic return. This was the behaviour the agent leaned in between 19000 and 26000 episodes, and as a result, there was a significant increase in the average episodic return (refer next slide).

# Results and Discussion

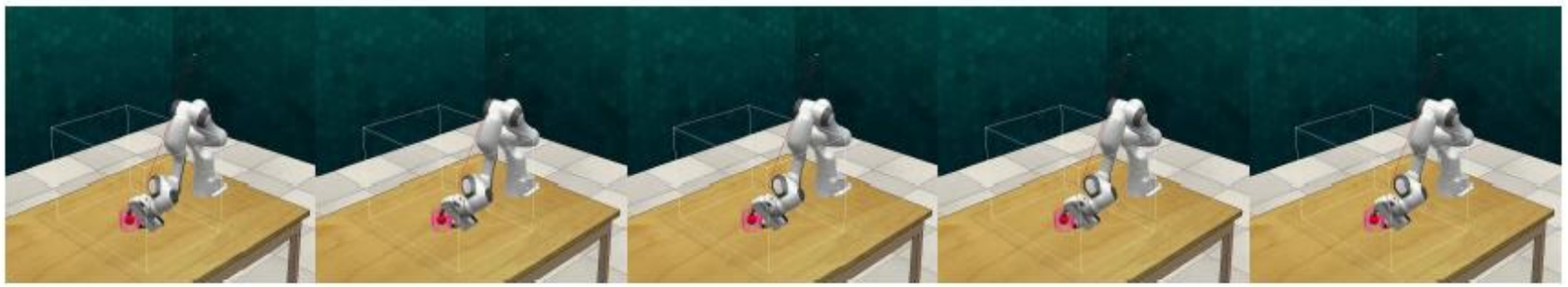
Timeframes of a CoppeliaSim simulation of a sample roll-out executed by an agent, which was successfully trained with baseline, on 26000 episodes. The red ball represents the target.



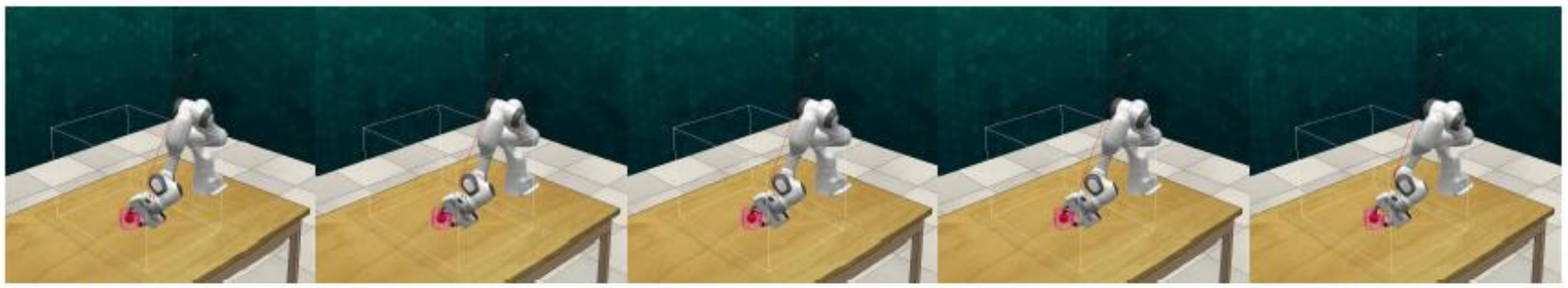
(a) Timeframe 1 (b) Timeframe 2 (c) Timeframe 3 (d) Timeframe 4 (e) Timeframe 5



(f) Timeframe 6 (g) Timeframe 7 (h) Timeframe 8 (i) Timeframe 9 (j) Timeframe 10



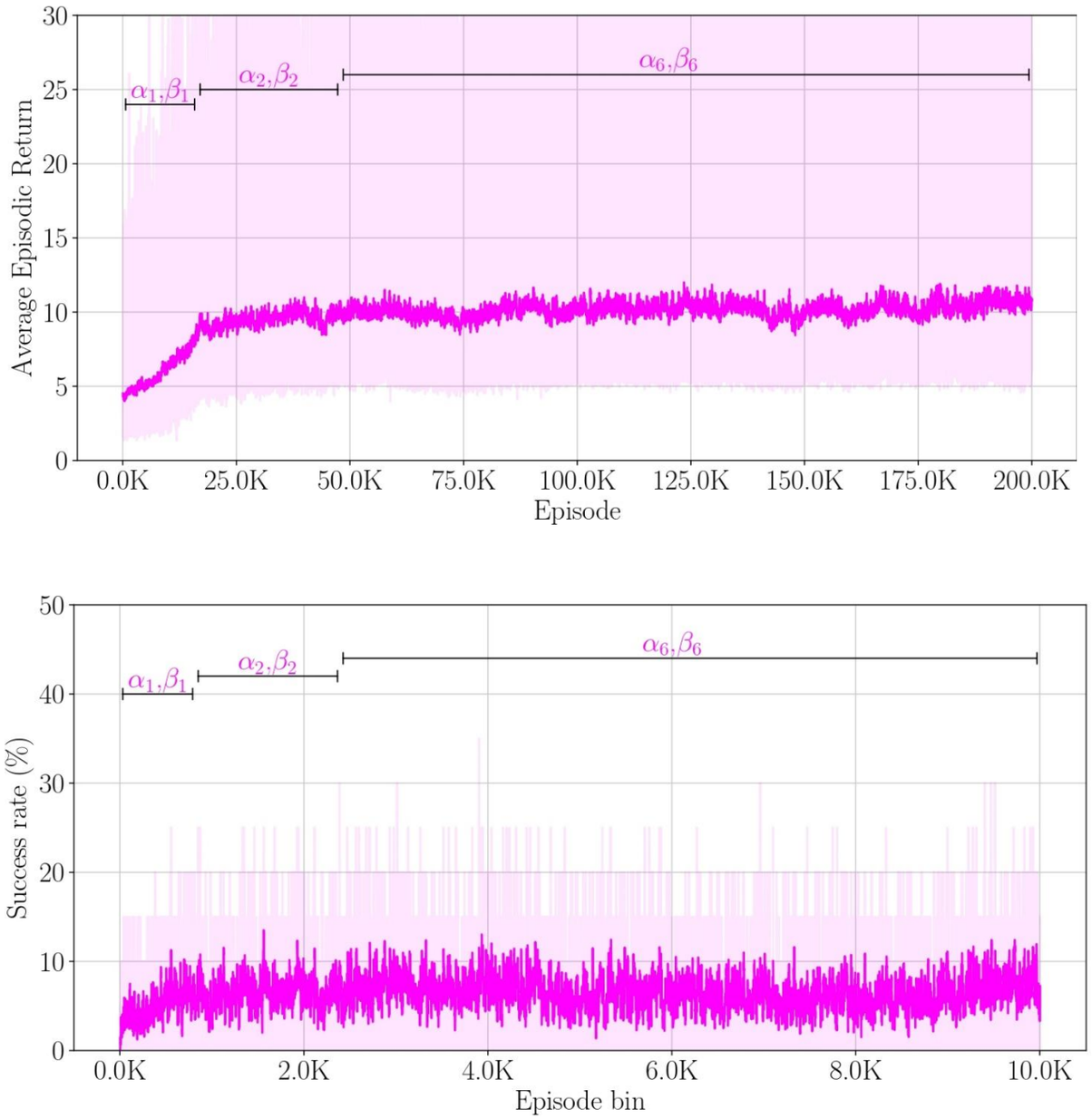
(k) Timeframe 11 (l) Timeframe 12 (m) Timeframe 13 (n) Timeframe 14 (o) Timeframe 15

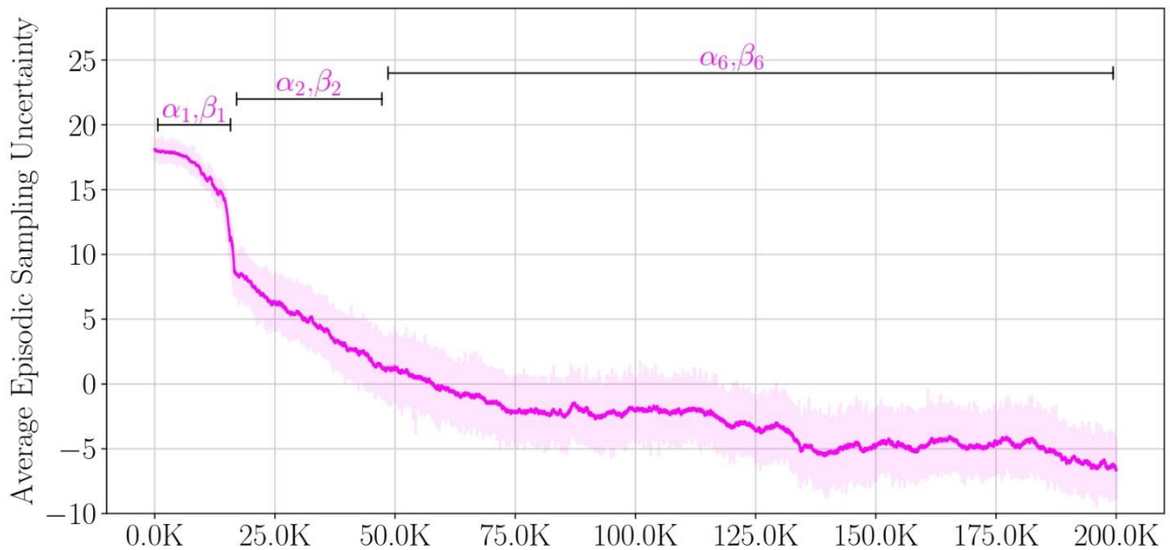


(p) Timeframe 16 (q) Timeframe 17 (r) Timeframe 18 (s) Timeframe 19 (t) Timeframe 20

# Results and Discussion

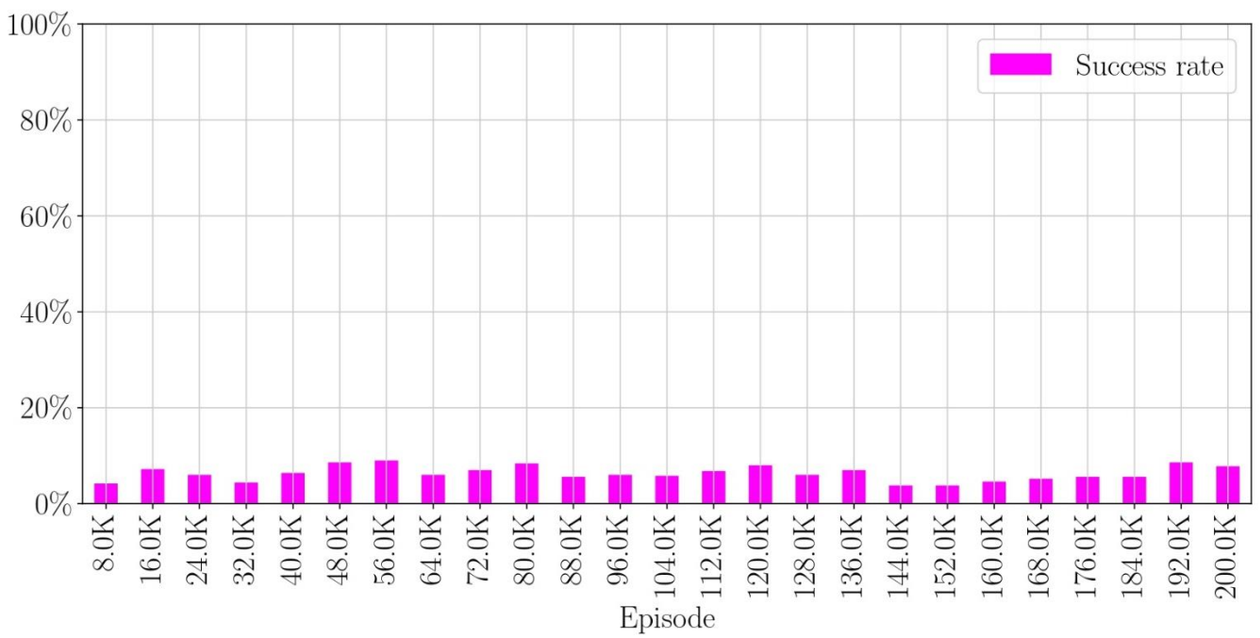
## Training on the actual benchmark task:









**Need to Maintain Stochasticity:**

- The learning rate is modified at 16400th and 47900th episodes during training in the presence of baseline.

- The maximum success rate measured on the fly during training is only around 10%, and the maximum success rate measured on 500 roll-outs during validation is also around 10%, which reflects that there is still room for improvement.

- But unfortunately, the uncertainty of action sampling became very low, and the learning rates $\alpha_6 \ and \ \beta_6$ are also very low, which makes it difficult for the agent to explore unseen regions in the search space.

- That's where the need to maintain the stochasticity of the action sampling during training arises, to explore the action space properly, without becoming certain about its non-sense actions in advance.

# Results and Discussion



Premature convergence is a common problem, which means that the agent got stuck in a local optimum during optimization. Let us look at a few of the approaches, which are more relevant to our use case in the context of reinforcement learning, to overcome premature convergence.

- In **off-policy learning** since the behaviour policy is different from the target, the agent can explore continuously, and hence off-policy algorithms give flexibility in dealing with the trade-off between exploration and exploitation. Off-policy learning also facilitates multiprocessing, multitask learning and kinesthetic teaching.
- Adding the entropy of the policy to the objective function improves exploration by discouraging premature convergence to suboptimal policies [Mni+16]. The optimization problem with the additional entropy term results in a smoother objective function that can be optimized with a larger learning rate, and hence it speeds up the training [Ahm+18]. **Entropy regularization** facilitates the agent to re-utilize the learned behaviour to fine-tune the policy for a more advanced task, instead of learning from scratch [TL17].

# Results and Discussion

- The influence of entropy is not guaranteed to change the landscape of the objective function significantly, in order to find better solutions at all times. Its because the impact of entropy on the objective function depends both on the problem and the environment [Ahm+18].

[Mni+16] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, et al. „Asynchronous Methods for Deep Reinforcement Learning“. In: *arXiv:1602.01783v2 [cs.LG]* (2016).
[Ahm+18] Zafarali Ahmed, Nicolas Le Roux, Mohammad Norouzi, and Dale Schuurmans. „Understanding the impact of entropy on policy optimization“. In: *arXiv:1811.11214v5* (2018), pages 1–25.
[TL17] Pieter Abbeel Tuomas Haarnoja Haoran Tang and Sergey Levine. „Reinforcement Learning with Deep Energy-Based Policies“. In: *International Conference on Machine Learning* (2017), pages 1–16.

# Conclusion and Future Works



- Experiments show that decaying the learning rate during training results in better performance and helps to avoid early deterministic behaviour. In the future, the agent performance needs to be analysed with different learning rate schedulers to understand its impact on training.

- Even though the simulated agent performed well in the simplified setting of the task, as a part of the future work, the feasibility of executing the learned behaviour on a real Panda robot needs to be verified.

- Since the optimization landscapes are highly problem and environment dependent, the increased complexity in the actual problem case might have generated a more complicated optimization landscape with many local optima. In that case, other techniques to maintain the stochasticity of action sampling during training might help the agent to explore unseen regions of search space.

# Conclusion and Future Works



- It is highly challenging to develop general-purpose optimization algorithms, which work well for all optimization landscapes. This emphasizes the need to better understand the underlying optimization landscape in direct policy optimization problems by investigating the aspects of the problem and the environment, which induces difficulties in the objective function.

- Similar to entropy regularization, batch normalization is an objective function smoothing technique, which might lead to improvement in performance [SM19]. Some research should be directed on alternative smoothing techniques to develop advanced methods for smooth convergence even with large learning rates.

[SM19] Andrew Ilyas Shibani Santurkar Dimitris Tsipras and Aleksander Madry. „How Does Batch Normalization Help Optimization?“ In: *arXiv:1805.11604v5* (2019), pages 1–26.