Trajectory Planning in Dynamic Environments

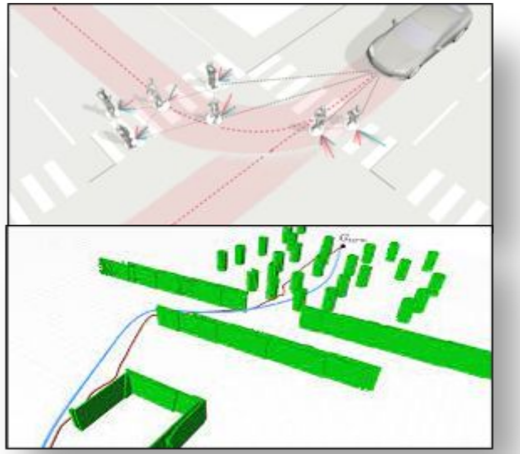
TUM

Proposal

Goal:
- Trajectory planning on **dynamic environments**, as it has a lack of research
    - Currently only in static and algorithms for this exists

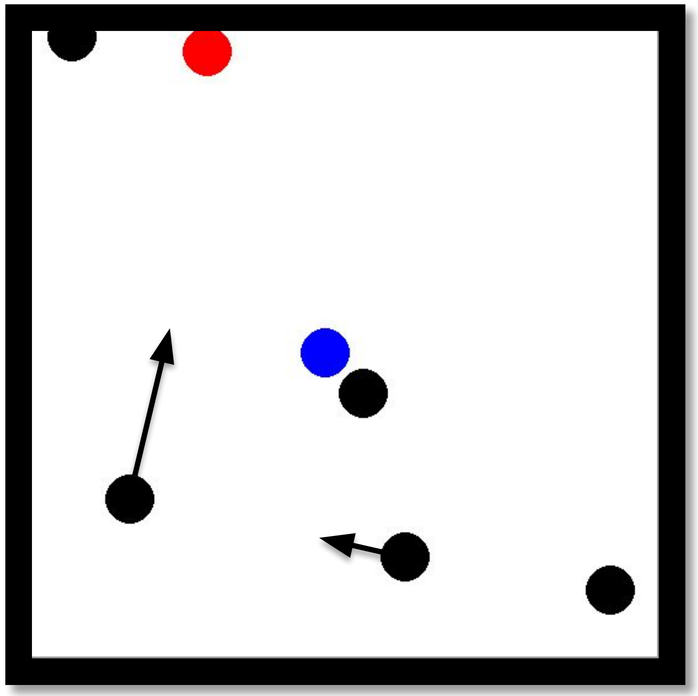
Related work:
- **MPNet** -> State of the art in Trajectory Planning with RL
    - Target data is traditional methods
    - Calculations are using an NNs
- **SAC + SAC-X (Scheduled Auxiliary Control)** -> implemented
- ~~**Meta-RL** -> 2nd priority~~

Example Usage Case:



Environment

- ~~Grid~~ **Continious** environment in Open AI Gym
    - 512 x 512 pixel size
    - 5 obstacles
        - **2 moving obstacles**



e.g. "a Museum room"

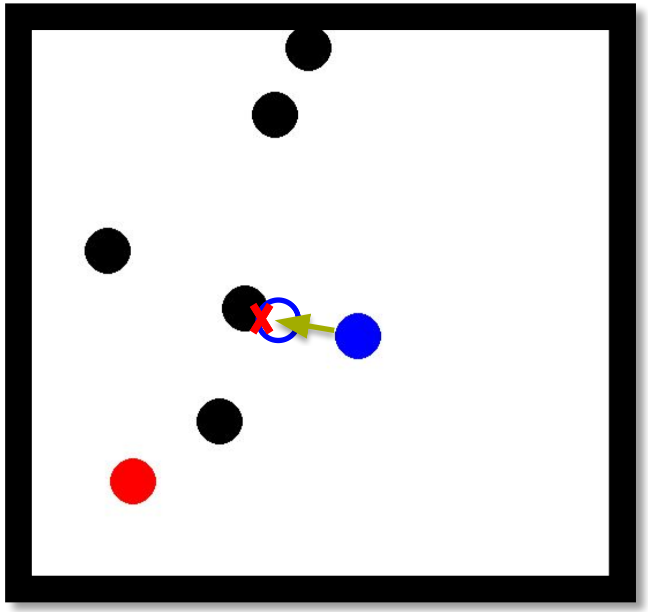
OpenAI Gym

Weights & Biases

Google Cloud Platform

Our Skills/Subtasks

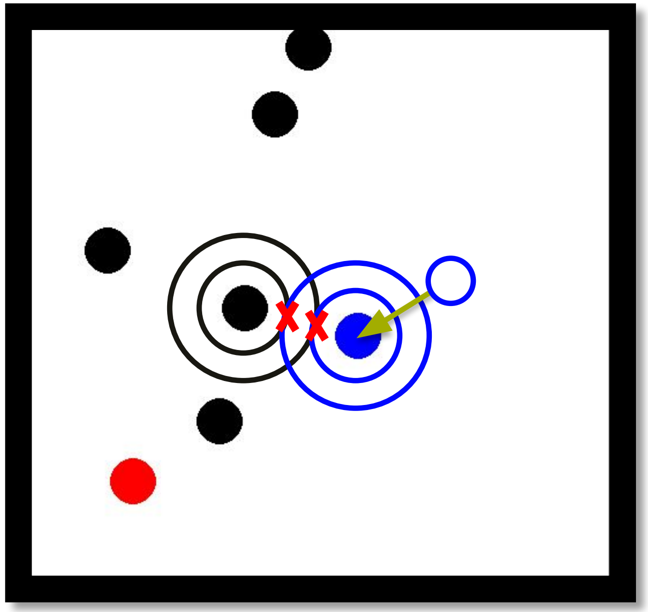
1. Obstacle Avoidance
- **Collision prediction**: We look into the next step and give penalty for collision



Our Skills/Subtasks

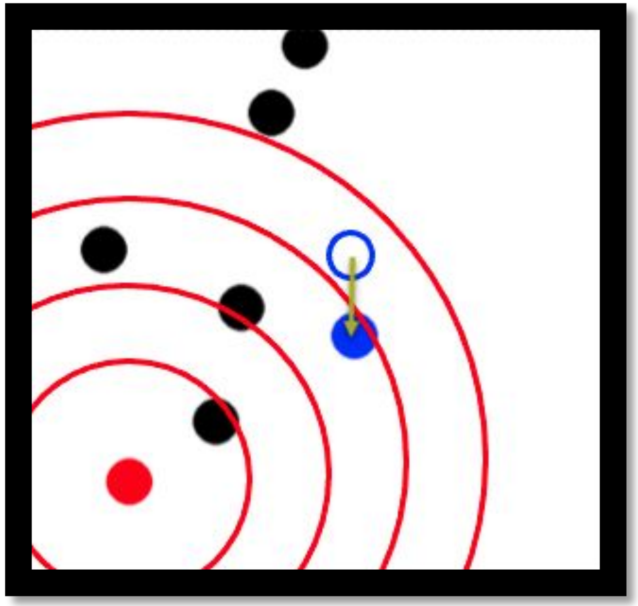
1. (Sparse) Obstacle Avoidance

- **Predictive obstacle avoidance**: We look into the next step and give increasing penalty for getting close to an obstacle



Our Skills/Subtasks

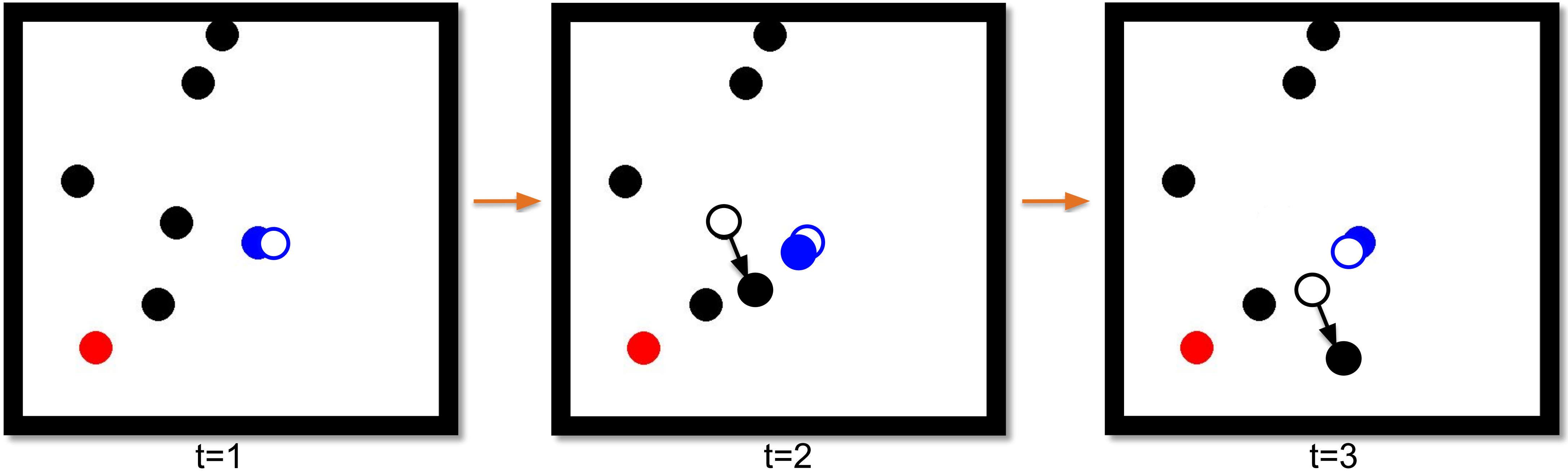
2. Target Seeking
- **Target proximity checkpoints**: If the agent reaches a distance threshold to the target it gets a reward



Our Skills/Subtasks

(**History:** We track the step history of the agent for subtask 3 and 4)

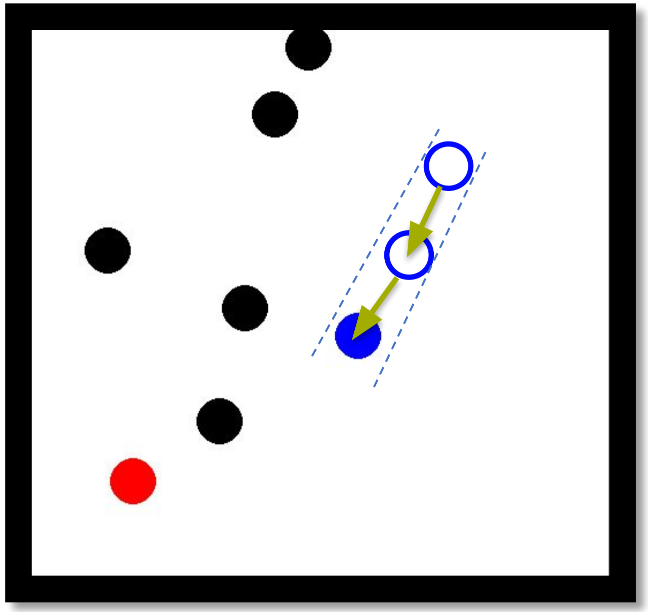
3. Waiting: If the agent waits (e.g. moving obstacle passes) for certain amount fo steps it gets a reward



Our Skills/Subtasks

(**History:** We track the step history of the agent for subtask 3 and 4)

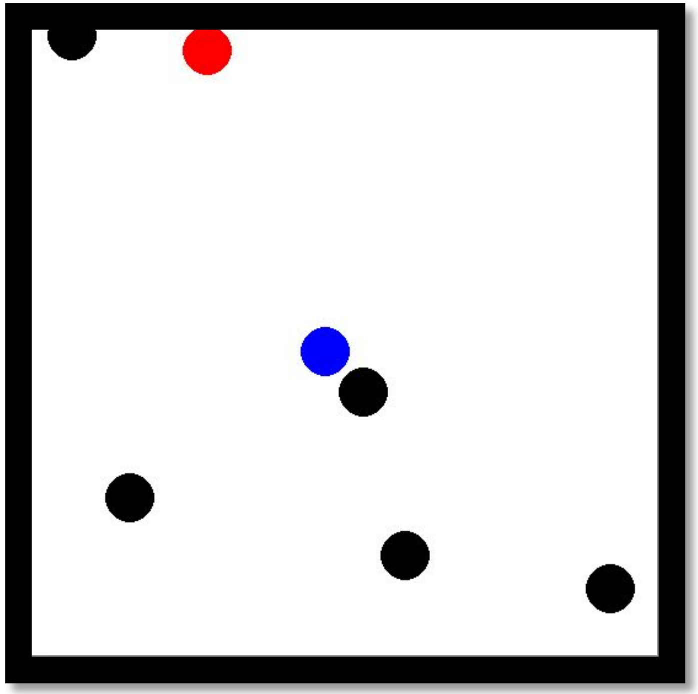
4. Consistency: If the agents behavior is consistent (e.g. moving in one direction consecutively) for certain amount of steps it gets a reward



Our Assumptions

For environment:
- We know **obstacle IDs** and use them in **obstacle sorting**
- We use super sparse rewards:
  - (+1) for **reaching the target**
  - (-5) for **collision**
- All **subtask rewards** are in [0,1] interval

For features:
- We use an **environment seed**
  - we train on **same set of environments** for every version
  - we **test on unseen** environments

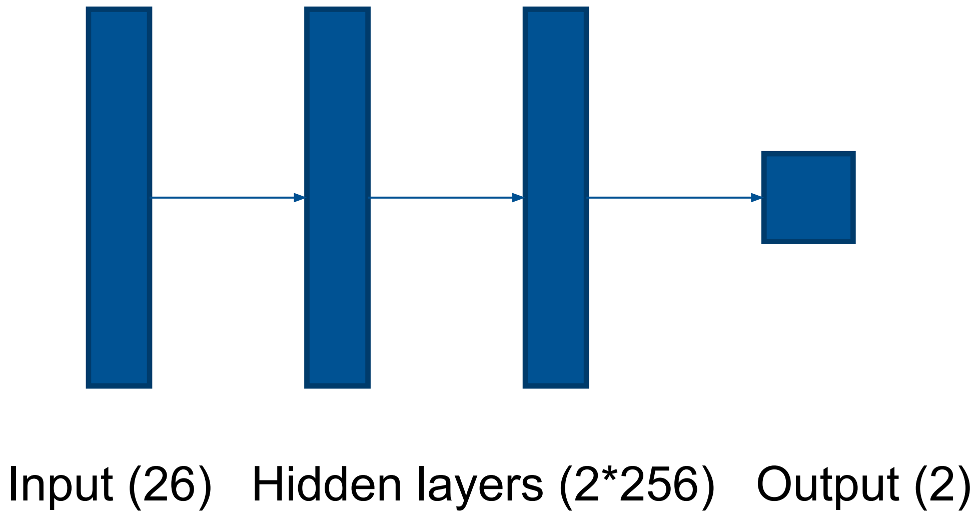


Neural Network structure for SAC

Same structure for ActorNet, CriticNet and ValueNet ->

Input:
- Position of Agent, Target, Obstacles
- Velocity of Agent, Obstacles

Output (ActorNet):
- Velocity of Agent (mean and std)

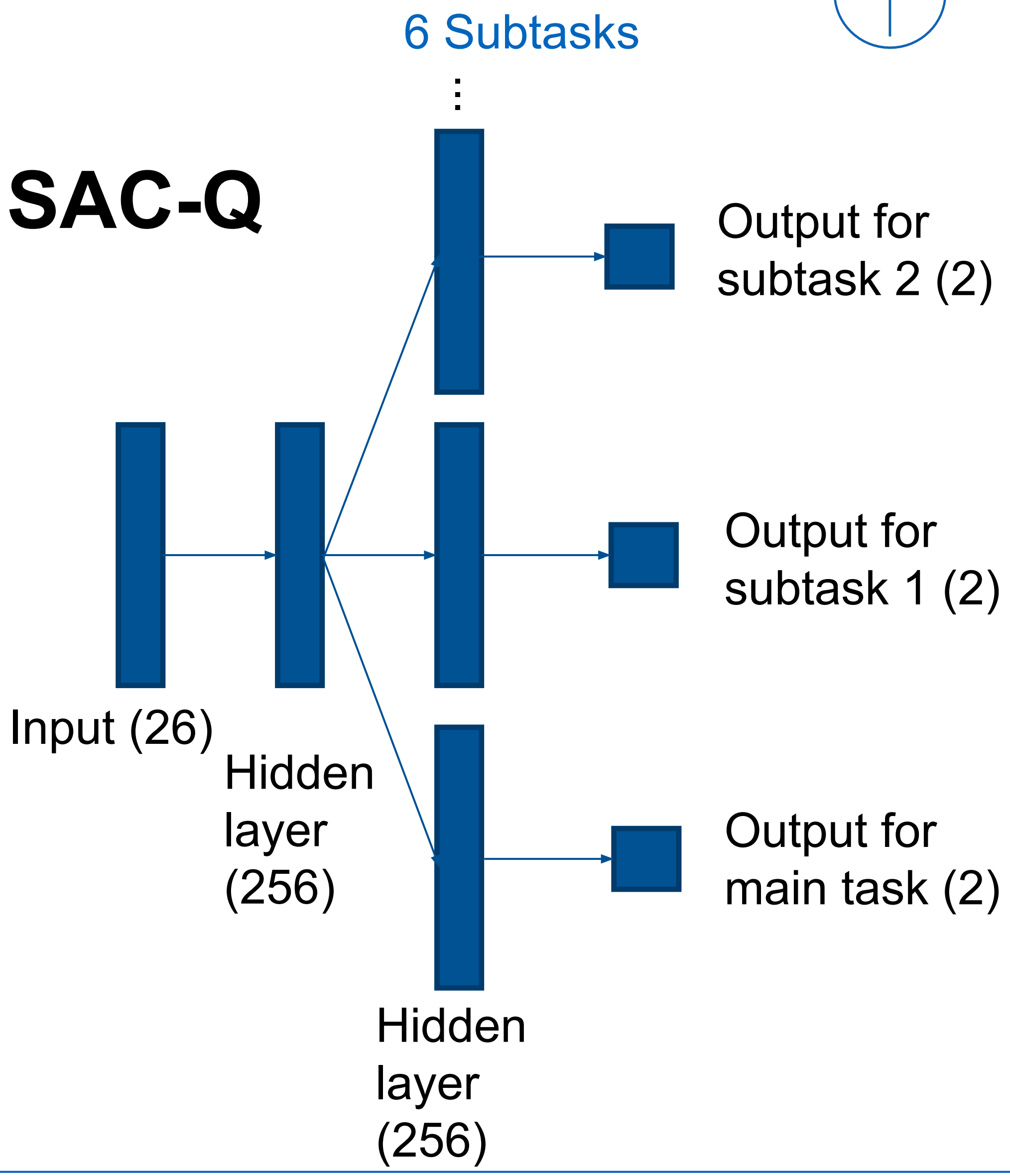


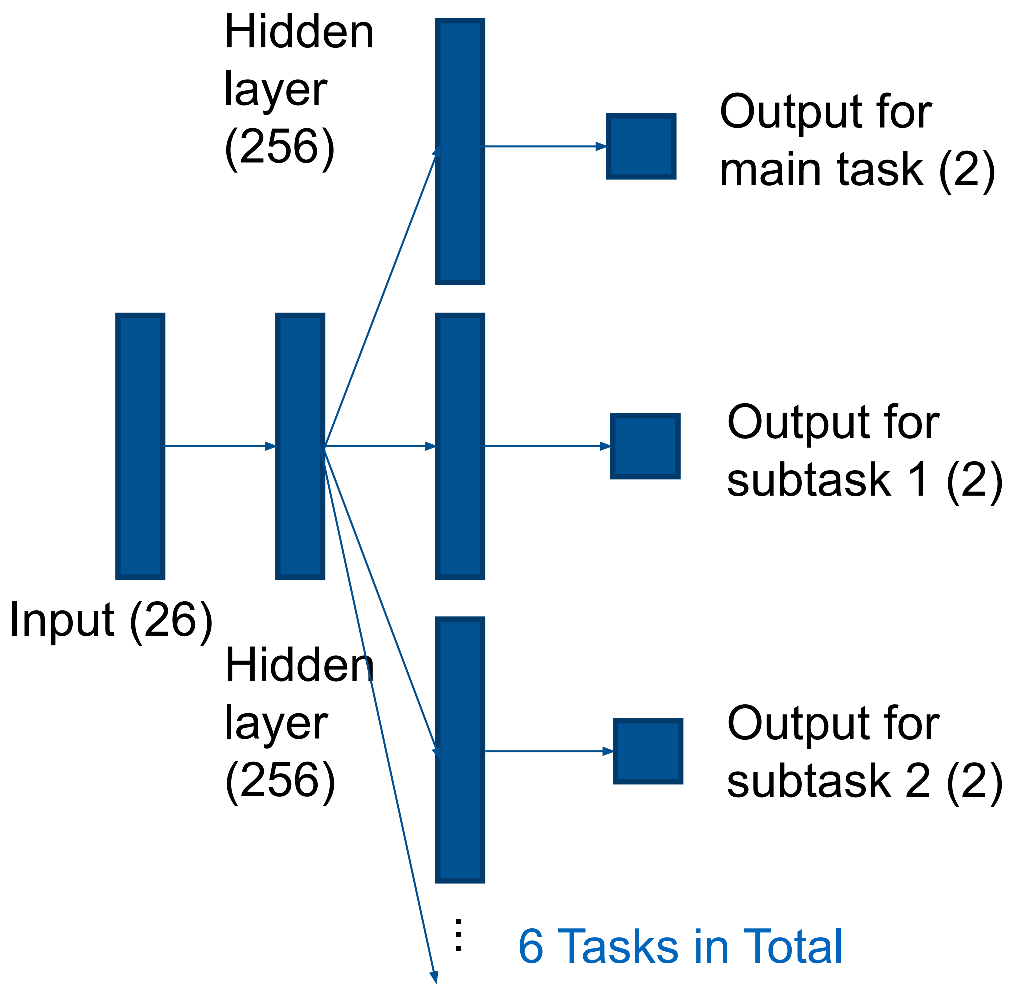
Neural Network structure for SAC-Q

Same structure for ActorNet, CriticNet and ValueNet ->

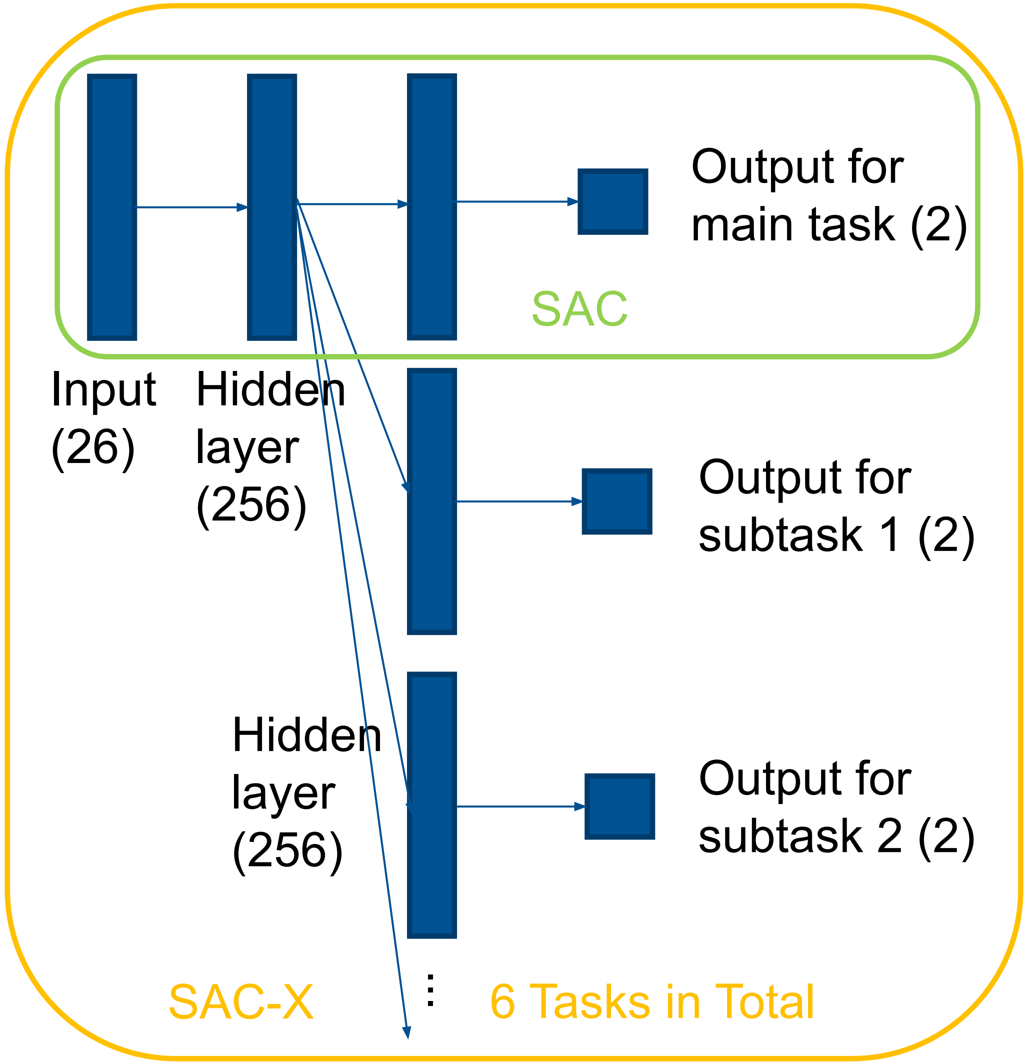
Input:
- Position of Agent, Target, Obstacles
- Velocity of Agent, Obstacles

Output (ActorNet):
- Velocity of Agent (mean and std)



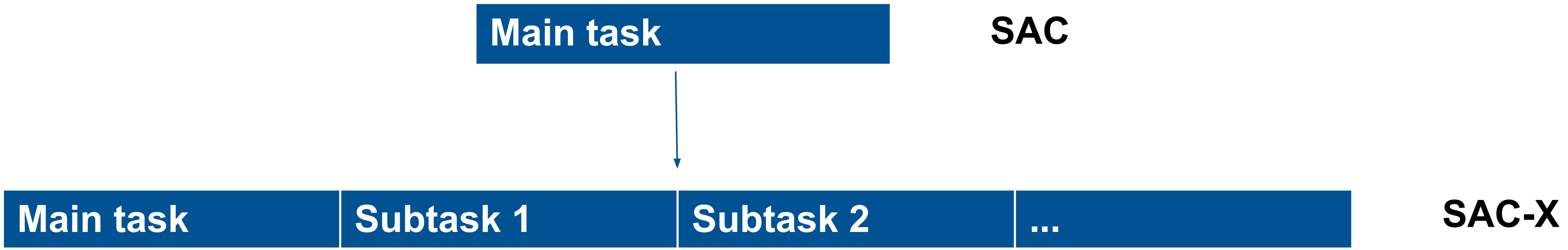


Hidden layer (256)
Output for main task (2)
Output for subtask 1 (2)
Input (26)
Hidden layer (256)
Output for subtask 2 (2)
6 Tasks in Total


Output for
main task (2)
SAC
Input
(26)
Hidden
layer
(256)
Output for
subtask 1 (2)
Hidden
layer
(256)
Output for
subtask 2 (2)
SAC-X
6 Tasks in Total

SAC to SAC-X

- We added different **subtasks** and we **train them separately with same replay buffer**
- We use a **scheduler to switch between** different subtasks
- Example for subtask: moving toward target, moving away from obstacles ... (more later)



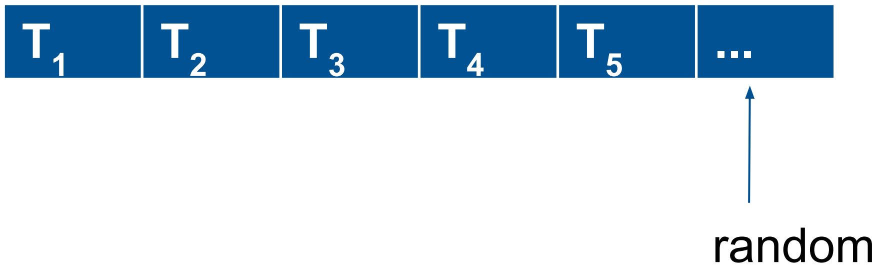
Scheduling Strategies

- SAC
    - **no subtasks -> no scheduling**
- SAC-X
    - SAC-U
        - **random**
    - SAC-Q
        - **depending on previous subtasks**

Scheduling Strategies

SAC-U

Choose a new subtask **randomly** from a **uniform** distribution



Scheduling Strategies

T_t (Task in time t) = Task ID

T_1	T_2	T_3	T_4	T_5	T_6=1	**Future Reward**: 3 pts
T_1	T_2	T_3	T_4	T_5	T_6=2	0 pts
T_1	T_2	T_3	T_4	T_5	T_6=3	1 pts

SAC-Q

Choose a new subtask **based on all past subtasks and potential future reward**

$$P(\text{subtask} = i) = \frac{exp(R(\text{subtask} = i)/\eta)}{\sum_{k=1}^{N} exp(R(\text{subtask} = k)/\eta)}$$

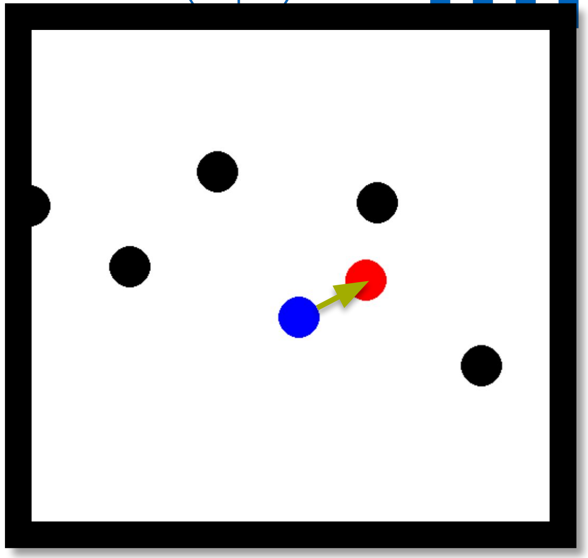
T_1	T_2	T_3	T_4	T_5	T_6=...

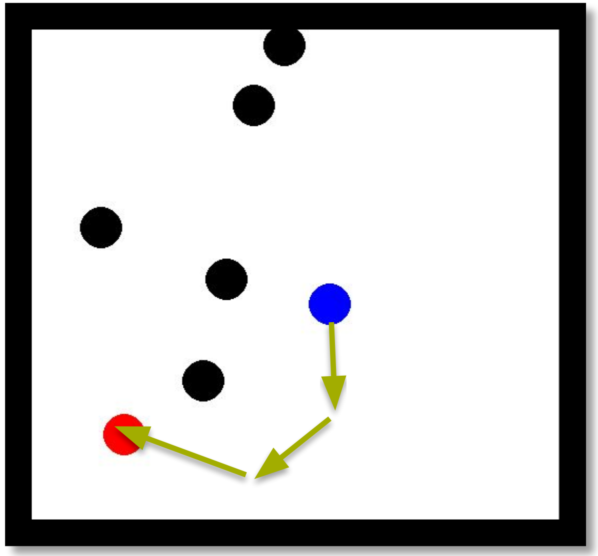
Choose **subtask 1** as its with **higher probability**

Why SAC-Q

- In **different situations**, **different skills** can be utilized
- **SAC-U does not work well with our main task**, because we want to **choose a new subtask based on current situation** and quickly go back to main task

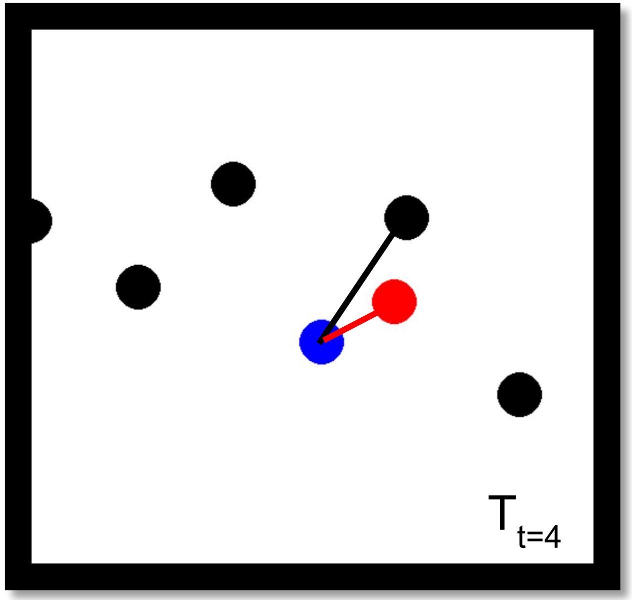
Subtask 2: Target Seeking





Subtask 1: Obstacle Avoidance

Our modification on SAC-Q



Problem of SAC-Q:
- It is **impossible to store** and train all possible subtask combinations

- We can just **take several past subtasks** (here 3)
- But now we may **lose a lot of state information**

\>> **Add state information into scheduler!**

- We classify the distance into "**close**", "**middle**" and "**far**".

Last subtasks of agent**:** $T_{t=1}$ $T_{t=2}$ $T_{t=3}$

State -> Distance to closest **obstacle**: **middle**

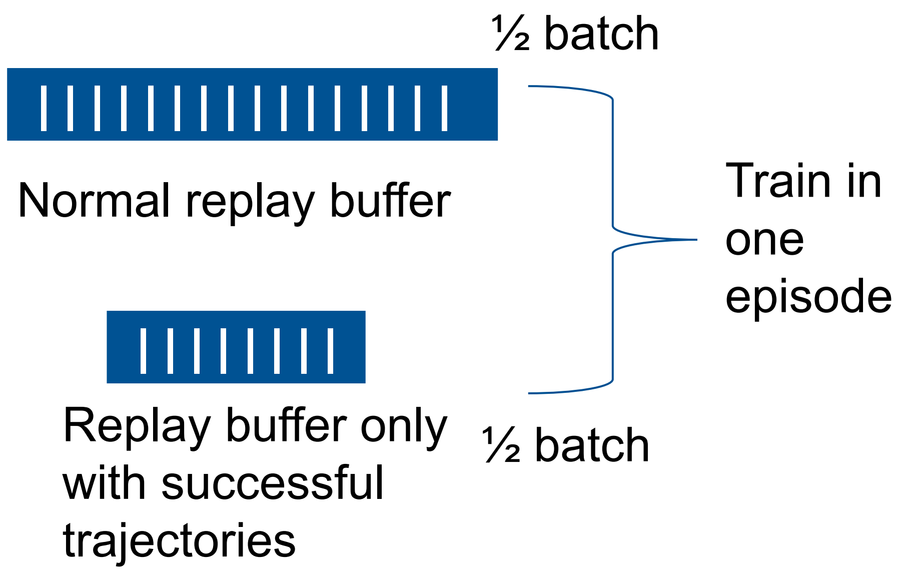
State -> Distance to **target**: **close**

T_4 Choose a new subtask (here **seek target**)

Sample balancing

Problem:
- It is **hard** for agent to **reach target** at the early stages of training
- Agent has far more **negative samples** than **positive samples**

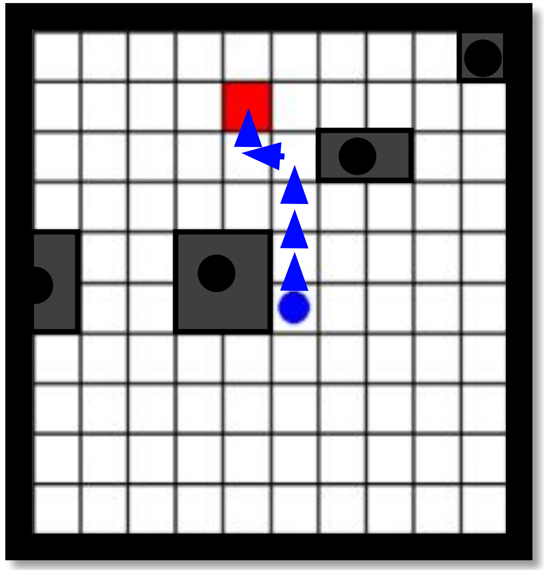
Solution 1:
- **Store** the **trajectory of successful runs** into a new replay buffer
- In every training episode, randomly take **half data from this replay buffer with successful trajectories**



Sample balancing

Problem:
- It is hard for agent to reach target at the early stages of training
- Agent has far more negative samples than positive samples

Solution 2:
- **Pretraining** is a good method to generate **positive samples** and we still use it.
    - Generated previously with **A-Star planner** and fed to the replay buffer



A-Star planner trajectory

SAC for continuous dynamic environment

Statistics	Base model (no features, only super-sparse rewards)	w/ Action smoothing (history size:3)	w/ Obstacle Sort	w/ Time Penalty	w/ Radius Checkpoints	w/o pretrain and w/ obstacle sort
Accuracy	0.51	0.37	**0.66**	0.38	**0.58**	0.09
Mean ± Std reward	-19.4±29.99	-27.3±29.01	-9.9±28.16	-27.2±29.12	-15.2±29.61	-45.05±15.74
Mean steps to finish	20.77	33.98	27.65	36.78	19.88	7.42

SAC-Q for continuous dynamic environment

Statistics	Base model (all features, only super-sparse rewards)	w/o Action smoothing (history size:3)	w/o Action filtering	w/o Obstacle sorting	w/o Time penalty
Accuracy	0.03	**0.18**	0.12	0.05	0.13
Mean ± Std main reward	-1.43±2.28	-1.61±2.51	-1.49±2.39	-1.73±2.43	-0.74±1.91
Mean ± Std of steps to finish	5.79±5.49	19.81±28.78	34.50±44.42	9.23±8.88	18.52±23.29

Moving Quickly

Movement in target direction

We need lots of steps

SAC-Q for continuous dynamic environment

Statistics	Base model (all subtasks, only super-sparse rewards)	w/o Collision prediction	w/o Predictive obstacle avoidance	w/o Target proximity checkpoints	w/o Waiting	w/o Step consistency
Accuracy	0.03	0.10	0.03	0.09	0.08	0.04
Mean ± Std main reward	-1.430±2.28	-1.39±2.32	-1.28±2.21	-1.21±2.23	-1.34±2.29	-1.55±2.34
Mean ± Std of steps to finish	5.78±5.48	13.59±18.04	6.25±5.16	36.19±47.40	7.31±8.75	12.55±16.06

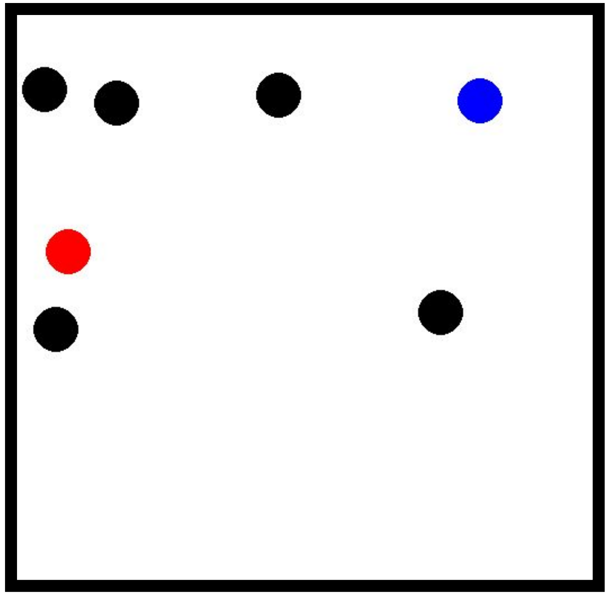
- Randomness in our scheduler
- Many skills

Exploit vs explore

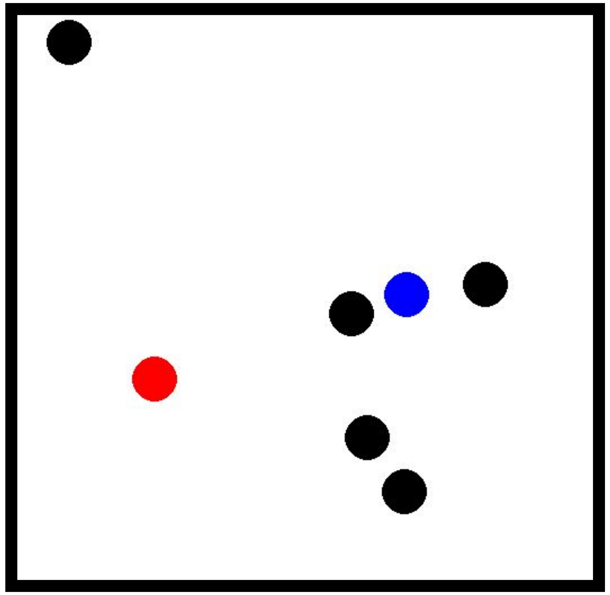
SAC vs SAC-Q

- SAC-Q can **learn different methods to avoid** obstacles
- SAC is more **greedy**, has **better accuracy**

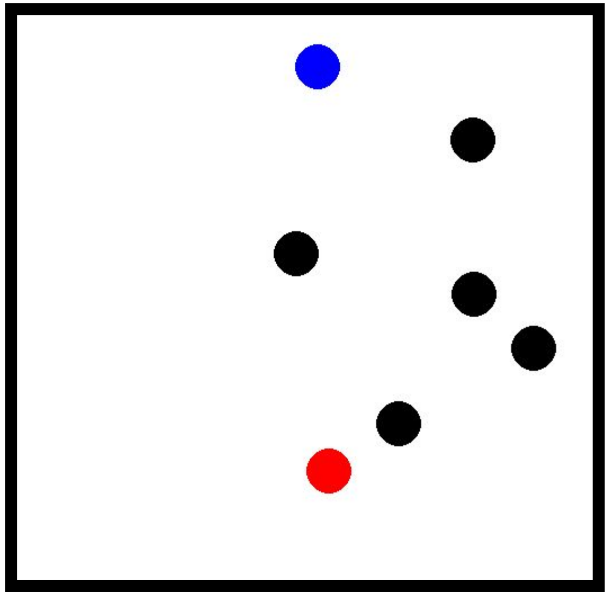
*Model used for gifs:
- SAC-Q w/o Action Smoothing
- with all skills



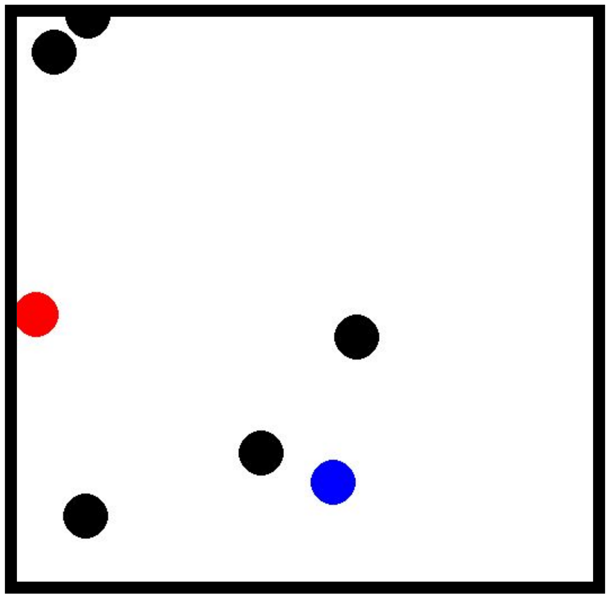
SAC
- would **not work if the obstacle was on the way to the target**
- avoids moving obstacle
- **does not wait around** the target for long



SAC-Q
- avoids moving obstacle
- exploits "**waiting**" reward

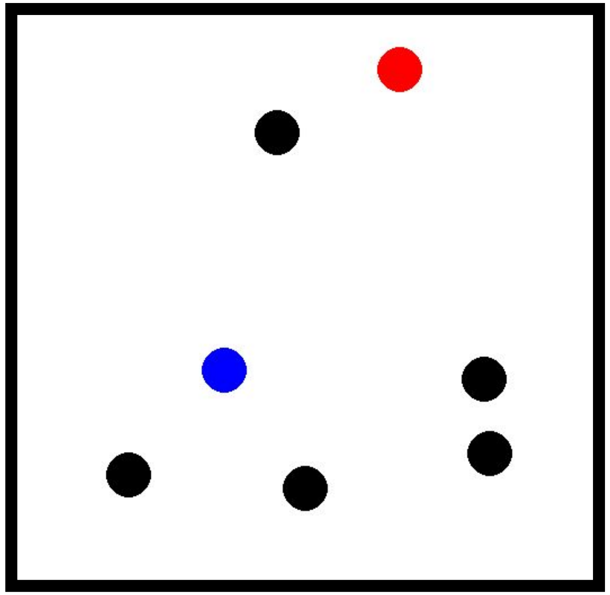


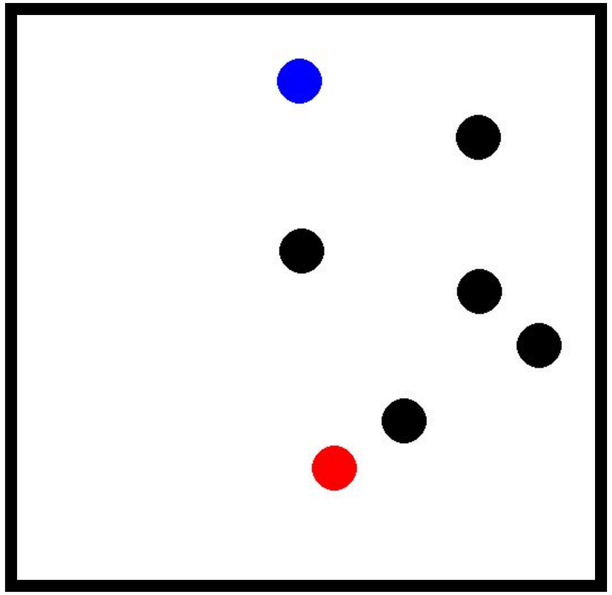
SAC-Q
- avoids obstacles or not based on chosen skills
- hardly reaches target



SAC-Q
- moving **obstacle coming from the target direction** is confusing

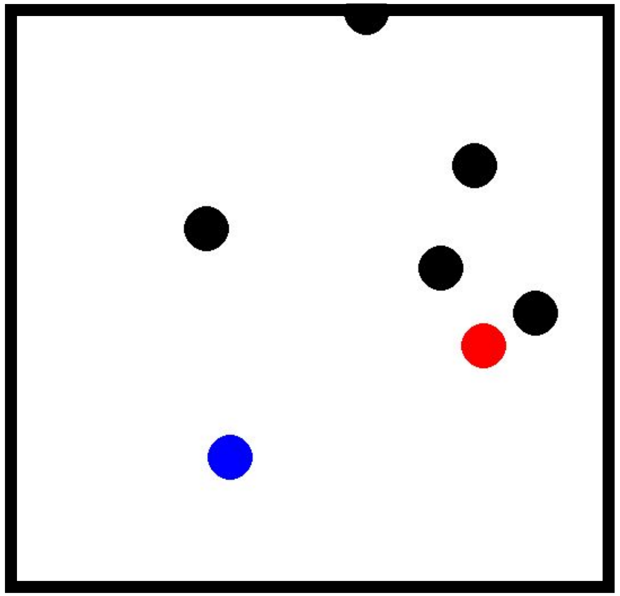
SAC-Q
- can **run away** from a moving obstacle





SAC-Q
- **fast reaction** to avoid moving obstacle

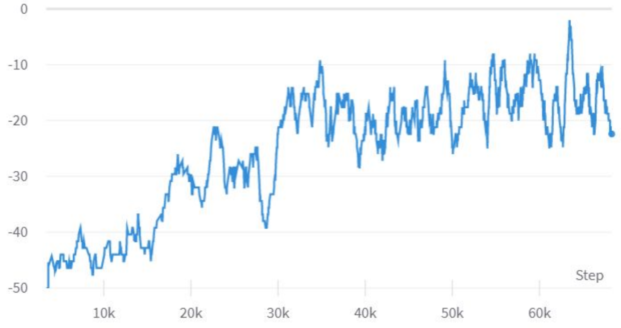
SAC-Q
- can learn to **wait**



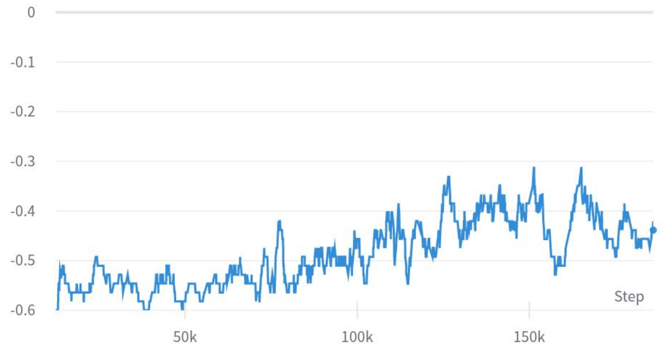
Conclusions

- **Obstacle sorting** makes agent pay more attention to closest obstacle
- **Pretraining** and **sample balancing** play an important role on making the training faster
- All the **subtasks** we use **increase the performance individually**
    - Where "**Predictive obstacle avoidance"** and **"Step consistency"** have the biggest effect

- Our experiments show that **SAC-Q** doesnt work well <u>yet</u>
    - increase **training time**
    - make the model more **complex**



mean reward curve of **SAC** during training

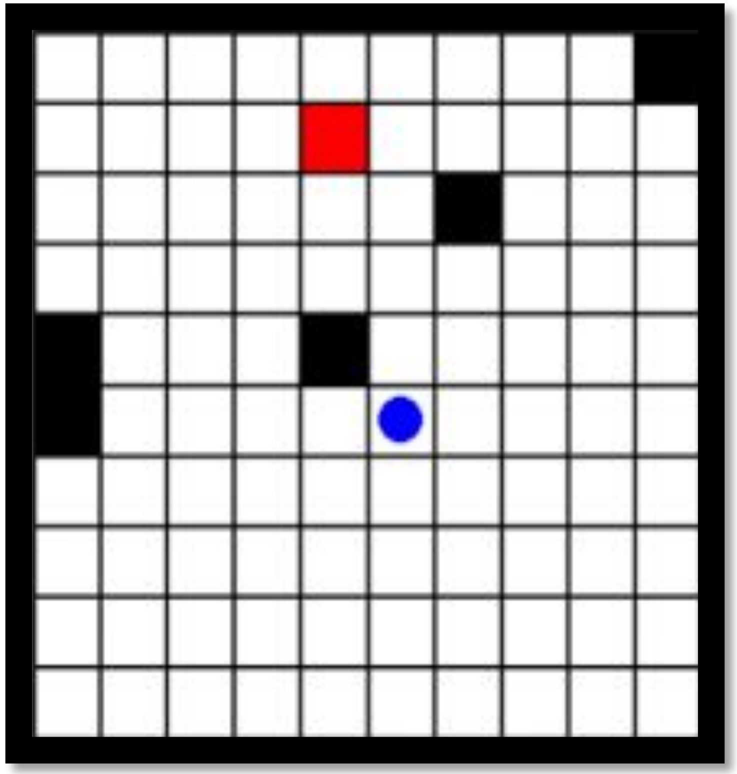


mean reward curve of **SAC-Q** during training

Thank you for your attention :)
Questions?

APPENDIX

Environment (before)



- **Grid environment** in Open AI Gym
  - 10x10 size
  - 5 obstacles