10/04/24 Lecture 06 → Recent advances & miscellaneous

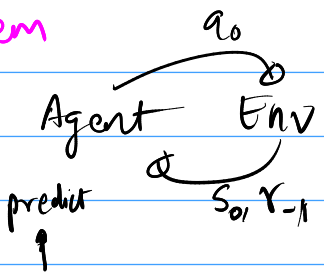
① RL as a sequence prediction problem
- $s_t, a_t, r_t, s_{t+1}, \ldots$
- Attraction of using large supervised models

② RL as a search heuristic for hard optimisation problems
- solve specific instances from scratch

③ RL in multi-agent settings
- Apart from adversarial ones such as games

④ Improving RL convergence
- reward shaping
- advanced sampling methods

RL as a sequence prediction problem



Upside-down RL

consider trajectory : $s_0, a_0, r_0, s_1, a_1, r_1, s_2, a_2, \ldots$

Traditional RL converts this to an optimisation problem :

$$a_0 = \max_a E[G_0 \mid s_0]$$

$G_0 = r_0 + \gamma r_1 + \ldots + \gamma^n$ [terminal reward]

$[s_0, a_0, r_0, \ldots s_{t-1}, a_{t-1}, r_{t-1}, s_t][\mathcal{L}][R_t] \rightarrow a_t \quad \ldots \quad T$

Challenges → ① Scalar reward, minimal feedback signal (no error is available)

② Necessity of discount factor for convergence, especially in non-terminating problems

$s_t, a_t, r_t, \ldots$

$[t, t+\tau]$

R

③ Sample efficiency : n step episode gives n samples

④ Difficulty of working with traditional deep architectures

⑤ Generalisation to out-of-ditribution areas, rather than learning a narrow behaviour distribution

⑥ Credit assignment in case of delayed rewards

Solution : Can we think of the trajectory (above) as a sequence, and use an LLM (essentially causal tranformer) to generate actions?

① conditioning on full trajectory

② Supervised training

③ Credit assignment with self-attention (causal → previous tokens only)

④ approx $\frac{n^2}{2}$ samples per episode

⑤ Highly capable pretrained architectures

Diversion: Quick recap of tranformers

Assume input encoding $x \rightarrow p$ tokens of n size each $\Rightarrow x_{p \times n}$

Query $Q = x W_Q$

Key $K = x W_K$

Values $V = x W_V$

matrices size $n \times m$

p, p, p

$$\text{attention} = \text{softmax}\left(\frac{QK^T}{\sqrt{m}}\right)$$

m

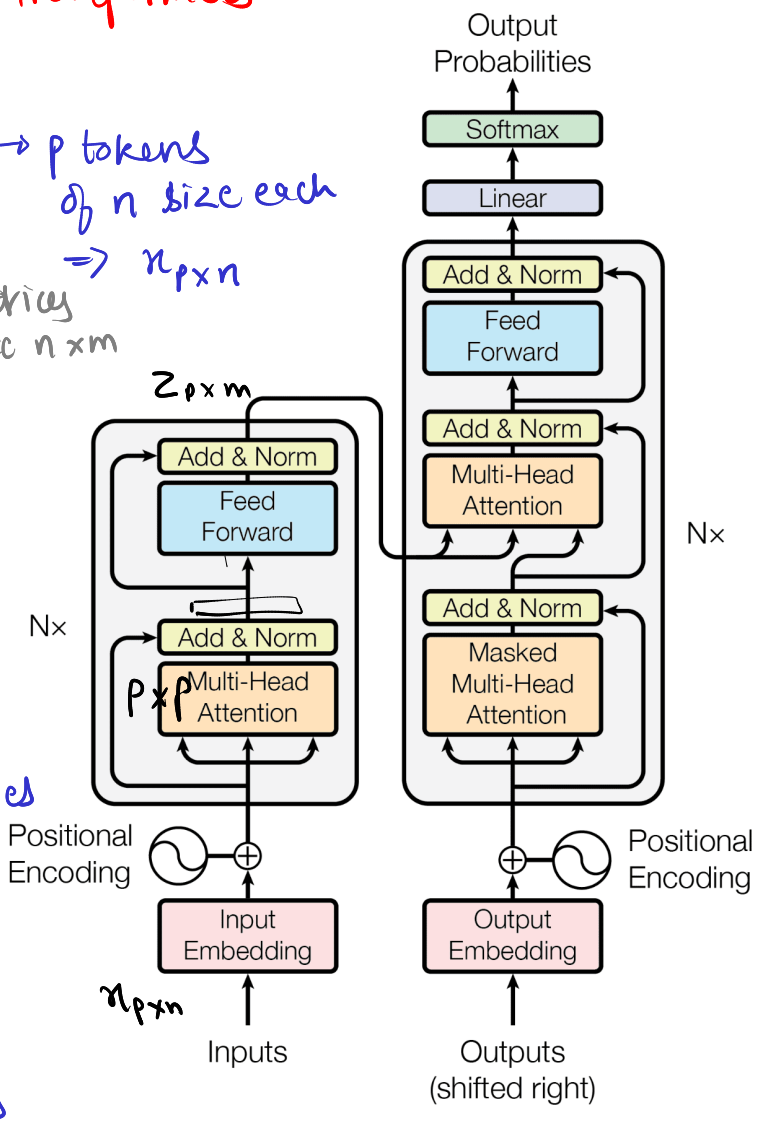
Note: attention size $p \times p$

Output Z = attention x values

$p \times m$

Z is passed to next block

→ Note number of samples stays same, only embedding size may be changed



So how does upside-down RL work?

Inputs →
- state measurements
- reward vector (current)
- previous action(s)
- desired reward
- time horizon to obtain this reward
- other extra context, e.g. goal state

upside down RL (UDRL) paper uses RNN instead of xformers

Phase 1 : Experience collection

inputs → action probabilities → action → new inputs

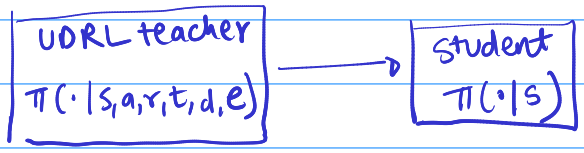
Phase 2: training

2a) Experience re-labeling → replace by actual observations instead of desired ones

2b) Trajectory subsampling → pick every pair of time instants and compute $\hat{r}$, Δt

2c) Supervised training of policy output

Final point on running in real world without reward conditioning



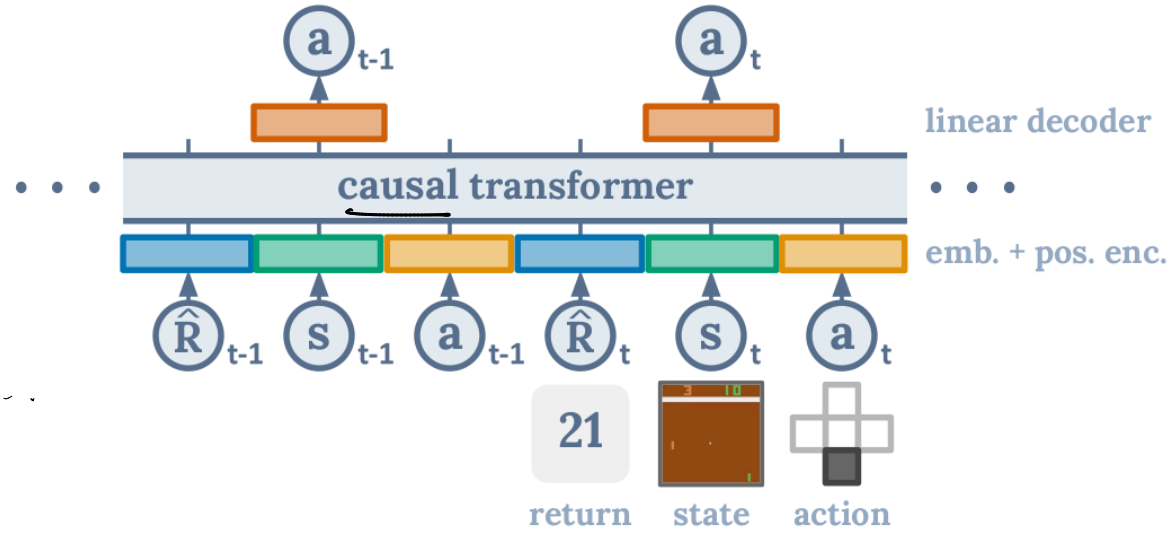
use the student network (trained in supervised fashion)

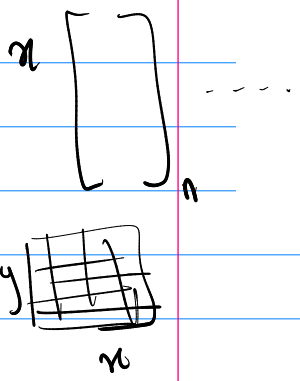
Use in learning by demonstration

① Let robot take random actions
② Get humans to imitate, record video
③ Train using UDRL with ② as "desired"
④ Take videos of actually useful demonstrations
⑤ Fine-tune using UDRL

Decision transformer & Trajectory transformer

Think of $r, s, a, r, s, a, \ldots$ as a sequence, and use a transformer to produce the next token.





Decision transformer

Set returns target $\hat{R}_t = \sum_{t'=t}^{T} r_{t'}$

Trajectory $\tau = (\hat{R}_1, s_1, a_1, \ldots)$

K time steps = 3K tokens

s, a, r have separate embeddings for matching dimensions

In-context learning
Algo distillation

Output = action embedding, to be decoded to action

Trajectory transformer

Use step rewards

Trajectory $\tau = (s_t^1, s_t^2, \ldots, a_t^1, a_t^2 \ldots r_t)$

Each token is one (scalar) dimension of state/action or reward, discretised

⇒ No embedding needed apart from normalisation

output → similar to NLP models, generate T length trajectories by sampling and choose action corresponding to most likely trajectory (beam search)

Discovering matrix multiplication using RL

Conceptual steps:

① Simplest matrix multiplication has 3 'for' loops, hence around k^3 scalar multiplication operations

② Multiplication operations more expensive than addition operations

③ # multiplications can be reduced by clever & selective multiplications of elements, followed by simple addition / subtraction
- generic for any $(n \times m) \times (m \times p)$ multiplication
- every set (n, m, p) has unique formula

we want to discover this

④ The multiplication of matrices, $C_{n \times p} = A_{n \times m} \times B_{m \times p}$ can be considered to be a tensor of dimension $n \times m \times p$

⑤ Just like a matrix of rank R has R independent eigenvectors, this tensor $\mathcal{T}_{n,m,p}$ also has R independent terms. For a square operation,

$$\mathcal{T}_{n,n,p} := \mathcal{T}_n = \sum_{r=1}^{R} U^{(r)} \otimes V^{(r)} \otimes W^{(r)}$$

outer product

each a vector of length n^2

⑥ Tensor $R \leq n^2$, since trivially you could have individual $a_i b_j$ terms (total n^2) combined to give elements of C

⑦ Goal of RL is to select $[U^{(t)}, V^{(t)}, W^{(t)}]$ in each step, to go from $\mathcal{T}_n$ to 0 in as few steps as possible.

Multi-Agent RL

Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments

Ryan Lowe* McGill University OpenAI

Yi Wu* UC Berkeley

Aviv Tamar UC Berkeley

Jean Harb McGill University OpenAI

Pieter Abbeel UC Berkeley OpenAI

Igor Mordatch OpenAI

Multi-agent RL similar to stochastic games

aggregate state + transition

For N agents, MDP → $(S, \{a_n\}, \{r_n\}, P, \gamma)$ $P: S \times A^N \to S$

individual actions — local rewards

Markov Games In this work, we consider a multi-agent extension of Markov decision processes (MDPs) called partially observable Markov games [19]. A Markov game for N agents is defined by a set of states $\mathcal{S}$ describing the possible configurations of all agents, a set of actions $\mathcal{A}_1, ..., \mathcal{A}_N$ and a set of observations $\mathcal{O}_1, ..., \mathcal{O}_N$ for each agent. To choose actions, each agent i uses a stochastic policy $\pi_{\theta_i} : \mathcal{O}_i \times \mathcal{A}_i \mapsto [0, 1]$, which produces the next state according to the state transition function $\mathcal{T} : \mathcal{S} \times \mathcal{A}_1 \times ... \times \mathcal{A}_N \mapsto \mathcal{S}$.[2] Each agent i obtains rewards as a function of the state and agent's action $r_i : \mathcal{S} \times \mathcal{A}_i \mapsto \mathbb{R}$, and receives a private observation correlated with the state $\mathbf{o}_i : \mathcal{S} \mapsto \mathcal{O}_i$. The initial states are determined by a distribution $\rho : \mathcal{S} \mapsto [0, 1]$. Each agent i aims to maximize its own total expected return $R_i = \sum_{t=0}^{T} \gamma^t r_i^t$ where γ is a discount factor and T is the time horizon.

Individual returns

local observations

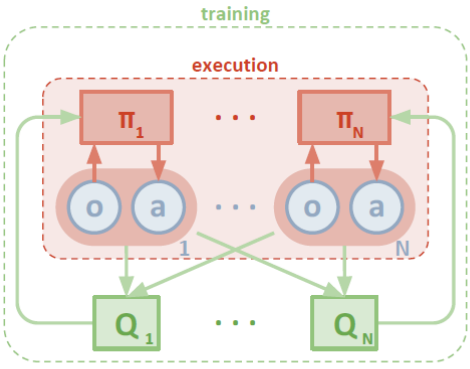


Figure 1: Overview of our multi-agent decentralized actor, centralized critic approach.

Gradient for agent i:

$$\nabla_{\theta_i} J(\theta_i) = \mathbb{E}_{s\sim p^{\boldsymbol{\mu}}, a_i\sim\boldsymbol{\pi}_i}[\nabla_{\theta_i} \log \boldsymbol{\pi}_i(a_i|o_i) Q_i^{\boldsymbol{\pi}}(\mathbf{x}, a_1, ..., a_N)].$$

- ∇_{θ_i}: policy parameters for agent i, resulting in policy $\pi_i(\cdot|o_i)$
- $\log \boldsymbol{\pi}_i(a_i|o_i)$: actions depend only on o_i
- $Q_i^{\boldsymbol{\pi}}(\mathbf{x}, a_1, ..., a_N)$: Q values have access to all agent actions and true state
- $\mathbf{x}$: true state

However note that each agent i builds its own Q_i, allowing us to handle non-cooperative / adversarial settings

In multi-agent DDPG (MA-DDPG), replace $\pi_i(\cdot|o_i)$ by $\mu_{\theta_i}(o_i)$

$$\mathcal{L}(\theta_i) = \mathbb{E}_{\mathbf{x},a,r,\mathbf{x}'}[(Q_i^{\boldsymbol{\mu}}(\mathbf{x}, a_1, \ldots, a_N) - y)^2], \quad y = r_i + \gamma\, Q_i^{\boldsymbol{\mu}'}(\mathbf{x}', a_1', \ldots, a_N')\big|_{a_j'=\boldsymbol{\mu}_j'(o_j)}, \tag{6}$$

where $\boldsymbol{\mu}' = \{\boldsymbol{\mu}_{\theta_1'}, ..., \boldsymbol{\mu}_{\theta_N'}\}$ is the set of target policies with delayed parameters θ_i'. As shown in

This is a problem, since D_2 depends on knowing all agent policies

To remove the assumption of knowing other agents' policies, as required in Eq. 6, each agent i can additionally maintain an approximation $\hat{\boldsymbol{\mu}}_{\phi_i^j}$ (where ϕ are the parameters of the approximation; henceforth $\hat{\boldsymbol{\mu}}_i^j$) to the true policy of agent j, $\boldsymbol{\mu}_j$. This approximate policy is learned by maximizing the log probability of agent j's actions, with an entropy regularizer:

$$\mathcal{L}(\phi_i^j) = -\mathbb{E}_{o_j,a_j}\left[\log \hat{\boldsymbol{\mu}}_i^j(a_j|o_j) + \lambda H(\hat{\boldsymbol{\mu}}_i^j)\right], \tag{7}$$

where H is the entropy of the policy distribution. With the approximate policies, y in Eq. 6 can be replaced by an approximate value $\hat{y}$ calculated as follows:

$$\hat{y} = r_i + \gamma Q_i^{\boldsymbol{\mu}'}(\mathbf{x}', \hat{\boldsymbol{\mu}}_i'^1(o_1), \ldots, \boldsymbol{\mu}_i'(o_i), \ldots, \hat{\boldsymbol{\mu}}_i'^N(o_N)), \tag{8}$$

Policy invariance under reward transformations: Theory and application to reward shaping

Reward shaping

In general, changing reward ⇒ changing optimal policy

↓

not desirable

Notable exception: Potential based rewards

$r_{new} = a r_{old} + b$ where $a > 0$

timal policy. It is shown that, besides the positive linear transformation familiar from utility theory, one can add a reward for transitions between states that is expressible as the difference in value of an *arbitrary* potential function applied to those states. Further-

than running our reinforcement learning algorithm on $M = (S, A, T, \gamma, R)$, we will run it on some *transformed* MDP $M' = (S, A, T, \gamma, R')$, where $R' = R + F$ is the reward function in the transformed MDP, and $F : S \times A \times S \mapsto \mathbb{R}$ is a bounded real-valued function called the **shaping reward function**. (Simi-

For any fixed MDP and assuming additive, memoryless shaping reward functions, this $R' = R + F$ is the most general possible form of shaping rewards.[3] Moreover, they cover a fairly large range of possible shaping rewards one might come up with. For example, to encourage moving towards a goal, a shaping-reward function that one might choose is $F(s, a, s') = r$ whenever s' is closer (in whatever appropriate sense) to the goal than s, and $F(s, a, s') = 0$ otherwise, where r is some positive reward. Or, to encourage taking action a_1 in some set of states S_0, one might set $F(s, a, s') = r$ whenever $a = a_1, s \in S_0$, and $F(s, a, s') = 0$ otherwise.

Theorem 1 *Let any S, A, γ, and any shaping reward function $F : S \times A \times S \mapsto \mathbb{R}$ be given. We say F is a* **potential-based** *shaping function if there exists a real-valued function $\Phi : S \mapsto \mathbb{R}$ such that for all $s \in S - \{s_0\}, a \in A, s' \in S$,*

$$F(s, a, s') = \gamma\Phi(s') - \Phi(s), \qquad (2)$$

(where $S - \{s_0\} = S$ if $\gamma < 1$). Then, that F is a potential-based shaping function is a necessary and sufficient condition for it to guarantee consistency with the optimal policy (when learning from $M' = (S, A, T, \gamma, R + F)$ rather than from $M = (S, A, T, \gamma, R)$), in the following sense:

- *(Sufficiency) If F is a potential-based shaping function, then every optimal policy in M' will also be an optimal policy in M (and vice versa).*
- *(Necessity) If F is not a potential-based shaping function (e.g. no such Φ exists satisfying Equation (2)), then there exist (proper) transition functions T and a reward function $R : S \times A \mapsto \mathbb{R}$, such that no optimal policy in M' is optimal in M.*

How? $Q^*(s,a) = R + \gamma \max_{a'} Q^*(s', a')$

whatever policy maximises Q^* must also maximise $\hat{Q}^*$

$\Rightarrow \underbrace{Q^*(s,a) - \Phi(s)}_{\hat{Q}^*(s,a)} = R + \underbrace{\gamma\Phi(s') - \Phi(s)}_{F} + \gamma\underbrace{[\max_{a'} Q(s',a') - \Phi(s')]}_{Q^*(s',a')}$

STAS: Spatial-Temporal Return Decomposition for Solving Sparse Rewards Problems in Multi-agent Reinforcement Learning

tionality to model complicated relations of the delayed global reward in the temporal dimension and suffer from inefficiencies. To tackle this, we introduce Spatial-Temporal Attention with Shapley (**STAS**), a novel method that learns credit assignment in both temporal and spatial dimensions. It first decomposes the global return back to each time step, then utilizes the Shapley Value to redistribute the individual payoff from the decomposed global reward. To mitigate the com-

$R_T \rightarrow r_0, r_1, \cdots, r_T$ time decomposition

$r_{i,0}, r_{j,0}, \cdots$ Agent level decomposition

the marginal contribution of player i in C can be defined as:

$$v_i(C) = v(C \cup \{i\}) - v(C). \tag{1}$$

The Shapley Value of player i can be computed from the marginal contribution of player i in all subsets of N:

$$\Phi_v(i) = \sum_{C \subseteq N \setminus \{i\}} \frac{|C|!(|N| - |C| - 1)!}{|N|!} v_i(C). \tag{2}$$

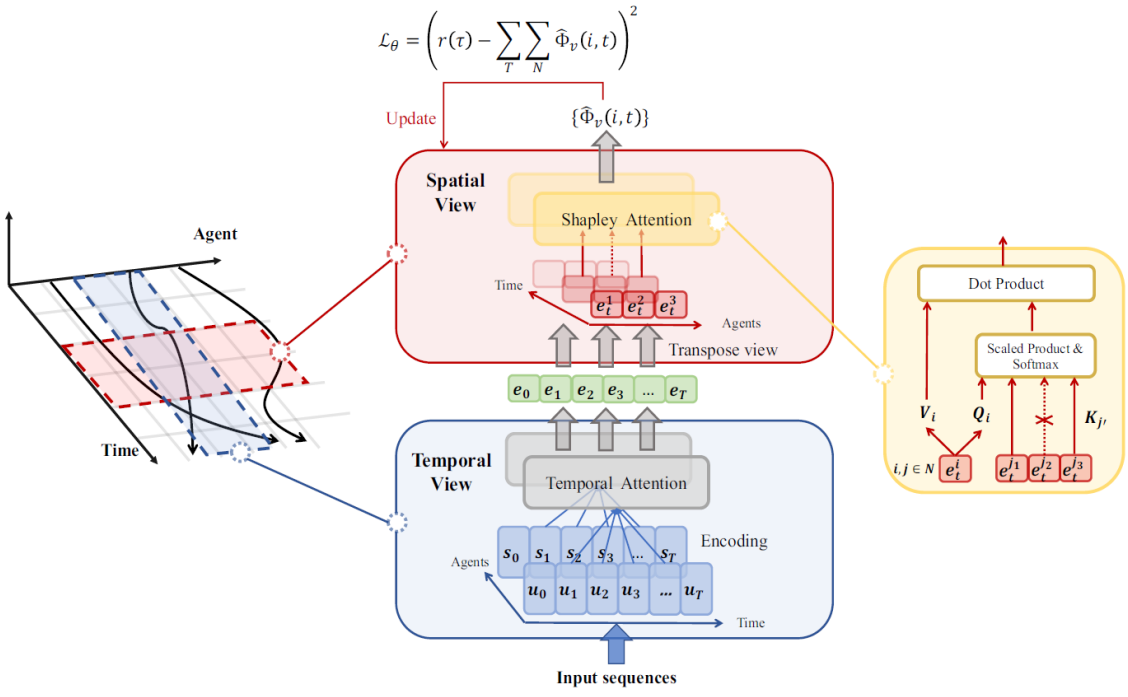
True return

$$r_{\text{ep}}(\tau) \approx \hat{r}_{\text{ep}}(\tau) = \sum_{t=0}^{T-1} \hat{r}(\boldsymbol{s}_t, \boldsymbol{u}_t). \qquad r_{\text{ep}}(\tau) = \sum_{t=0}^{T-1} r(\boldsymbol{s}_t, \boldsymbol{u}_t) = \sum_{t=0}^{T-1} \sum_{i=1}^{N} \Phi_v(i,t).$$

shaped return

Assumed to be sum decomposable

further decompose to agents



Using Contrastive Samples for Identifying and Leveraging Possible Causal Relationships in Reinforcement Learning

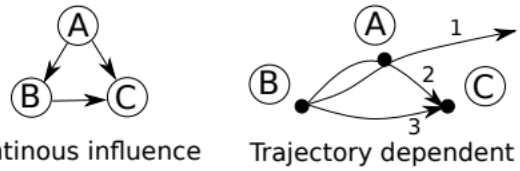


Fig. 1. Difference between a one-step inference problem and a sequential decision-making problem. In the latter case, the influence of A on the long term rewards is more subtle.

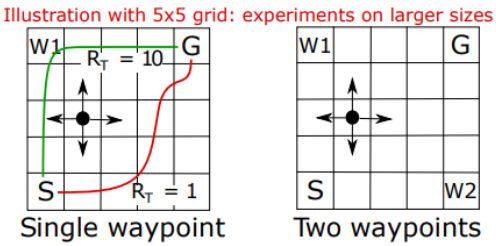


Fig. 2. Schematic of the environments used in this work. Waypoints `W1` and `W2` influence the terminal rewards if they are visited on the way to `G`. Size of the environment is illustrative; experiments use 8x8 and 12x12 grids.

Create two buffers

regular memory

contrastive memory

If $\Delta(s_t, s_{t+1}) \geq \delta$

then $(s_t, a_t, r_t, s_{t+1}) \rightarrow$ CER

Additionally,

$(s_t, a_t^* \neq a_t, r_t^*, s_{t+1}^*) \rightarrow$ CER

for training, sample fixed proportion from regular buffer and CER