## Wisely select which experience to feed to the agent

Practically speaking we want explanations to improve learning.
So, we also want to find a reasonable way to effectively provide them to a RL agent.
Considering that a standard RL agent learns from experience (that is past or present state transitions correlated with rewards) **only**, then the most **generic** way to provide explanations to a RL agent is via some sophisticated experience feeding mechanism. **Ad hoc** RL agents exploiting different mechanisms might also exist.

This is why the idea, we are currently trying, consists in shaping the experience buffer properly, so that a set of experience batches might serve as an explanation for the agent to learn faster in a complex-enough environment.
In order to shape the experience buffer we might:

- Augment experience:
    - Augment the state representations, adding to them extra information.
    - Change the reward function. In fact, we can see the set of constraints (that rules the environment) as a culture (in the sense given by Alex's work). Thus, if the environment is defined by a culture (a hierarchy of rules), we have that a complex-enough culture is supposed to shape/improve the behaviour of the agent. In RL, the reward function is the one that defines the behaviour of the agent, shaping the agent objective. It may follow that the explanations might be embedded in the reward function.
- Wisely select which experience to feed to the agent:
    - Cluster and prioritise experience in order to feed the agent with the most relevant, succinct and abstract amount of information that might help it to learn faster. -> *This is inspired by my current work on interactive explanatory tools for humans*.

Experience augmentation is not straightforward, and one might easily make the learning procedure harder/slower.
For example, what about "explanation-awareness" as "reward function engineering"?

In the case of a very complex culture (e.g. one that is not a Direct Acyclic Graph), it is hard (if not impossible) to engineer the reward function as to mimic the culture.
Then, what about "augmenting the state representations"?
In a complex enough culture, the agent might be required to learn in a curricular fashion, that is: learn first plain rules and then exceptions. According to literature <Isele, David, and Akansel Cosgun. "Selective experience replay for lifelong learning." *Thirty-second AAAI conference on artificial intelligence*. 2018.>, this might lead to catastrophic forgetting. So, simply augmenting the state representations does not seem to be enough, unless the extra information carries details of past.

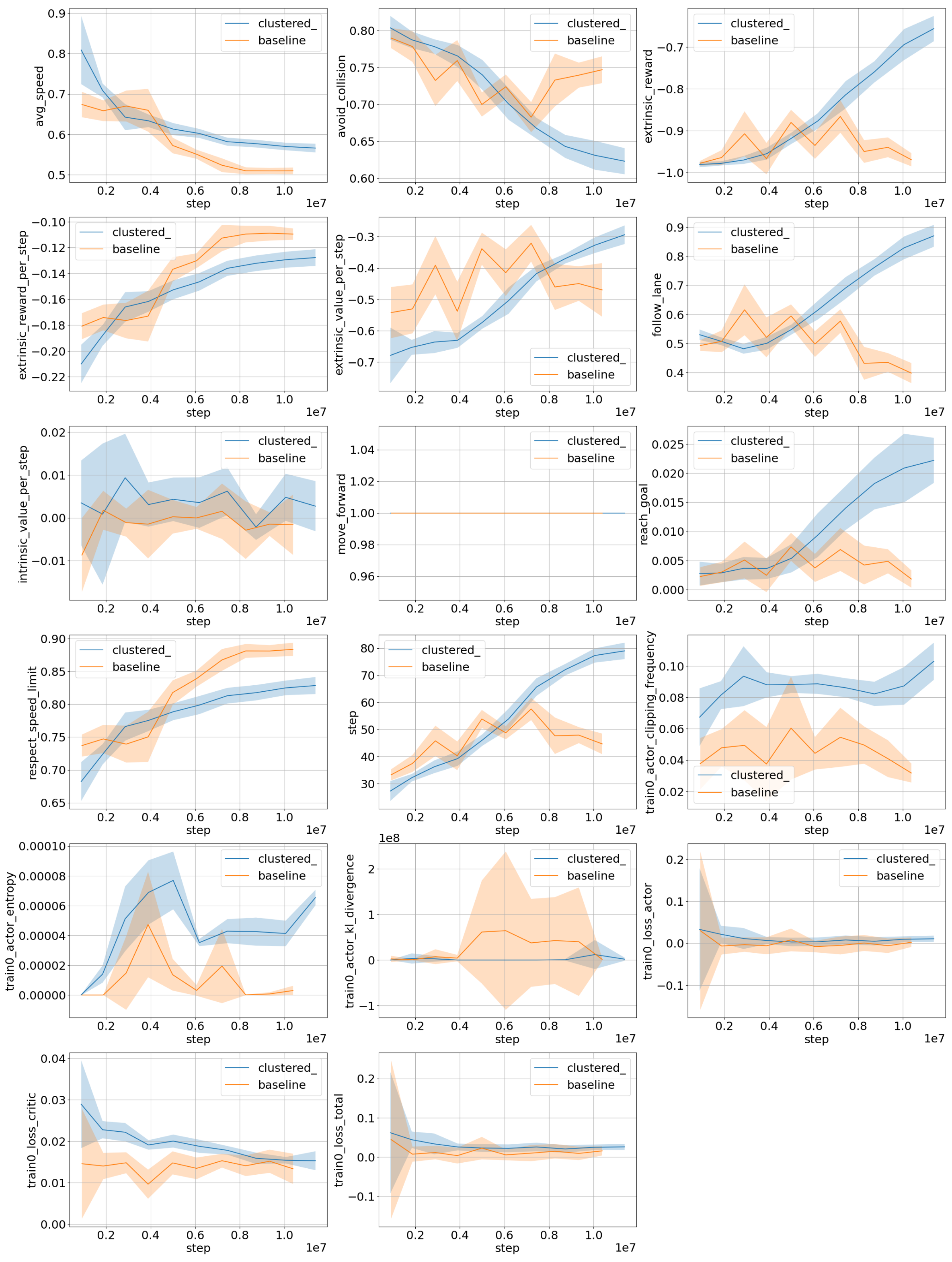
This is why wisely select which experience to feed to the agent seems to be the simplest and most sample efficient strategy left.
To this end we define a very simple reward function: -1 if a rule is broken, 1 if the goal is reached, 0 otherwise. Game is immediately terminated when a rule is broken.
Then we cluster experience mimicking the culture, so that to every cluster is associated a subset of rules (those responsible for the rewards stored in the state transitions of the cluster).
Then to every piece of experience, in every cluster, is given a relevance score estimated by the loss function (the agent objective following the reward function).
Then experience in clusters is prioritised by relevance and clusters are prioritised by their average relevance.
During learning, the agent samples the most relevant pieces of experience from the most relevant clusters, dropping the oldest and/or the less relevant ones.

I tested almost every part of it (with PPO), but the relevance-based cluster sampling (that currently is uniform rather then prioritised).
I performed tests on CescoDrive. The set of rules (the culture) for driving is super simple:

- move_forward
- avoid_collision
- follow_lane
- respect_speed_limit

The follow_lane is attacked by the avoid_collision rule, so that the agent can move away from the lane for avoiding obstacles on the road.

Initial **results** show that experience clustering by reward type (types: 'move_forward', 'avoid_collision', 'follow_lane', 'respect_speed_limit', 'reach_goal') is likely to be more effective than simple prioritised experience replay:



What's interesting is probably that for CescoDrive we also have an engineered reward function (I spent a few days shaping it). So that we can do an actual comparison between reward engineering and our strategy.

What's left to do:

- Tests on DDPG (continuous action spaces) or DQN (discrete action spaces).
- Tests on different environments.
- Tests on more complex cultures.
- Better benchmarks

Answering to Nik, I think that discretising CescoDrive's action space might be easy. Just need to create a grid and define the actions set (for steering and acceleration).