eXplanation Aware DQN

- The difference with vanilla (duelling) DQN (https://docs.ray.io/en/master/rllib-algorithms.html#dqn) is in the replay strategy: instead of having a single experience buffer, we may have multiple buffers, one per explanation. With this simple technique we can encode explanations in the order experience is presented to the learning agent.

- To every batch of consecutive state transitions of the same episode we assign an explanation, according to many possible strategies.
- An explanation is a label attached to the batch, explaining something about that batch.
- We might have different types of explanations:
    - "Why" explanations: telling why the batch has the rewards it has. This type of explanations can be produced by an argumentation framework.
    - "How" explanations: telling how a batch (e.g. an episode; a single transition) is behaving. This type of explanations can be produced by studying the average behaviour of the agent, e.g. if an episode has a cumulative reward that is greater than the running mean, then the agent is behaving better than average on that episode.
    - "Why-How" explanations: a concatenation of a why and a how explanation.
- A clustering scheme is a scheme adopted to group experience into multiple (prioritised) buffers, according to their explanations.
- **Assumption:** Let an explanation be an answer to a question.
- The idea is that every cluster represents an explanation, every explanation is an answer to a question, every question represents a task, therefore every explanation represents a different task. Keeping track of all the possible useful (for the sake of learning faster) tasks would prevent the agent to catastrophically forget any of them and thus to generalise better. Furthermore it would be possible for the agent to automatically learn tasks in a curricular fashion if it samples clusters in a prioritised fashion, by using the sum of the priorities of a cluster as cluster's priority (we will refer to this technique as cluster_prioritisation).

- We devised the following clustering strategies, but many others are possible:
    - epsd: how explanations containing information about the quality of the episode; if an episode has a cumulative reward that is greater than the running mean, then the transitions of that episode are labeled as “better” otherwise “worse”. With this scheme we have only two different clusters.
    - arg: why explanations containing explanations about the reasons behind the received rewards. Every reward can have multiple different explanations, where an explanation is an argument. If a transition has more than one label, then that transition is duplicated and inserted in multiple clusters at once. This would allow a maximum number of clusters that is linear in the number of different arguments.
    - rwrd: how explanations containing information about the quality of both the episode and the transition; if the transition has a reward greater than 0, then it is said to be “good” otherwise “bad”, this information is concatenated to the episode-level explanation. The maximum number of different clusters can be 2*2=4.
    - epsd_arg: why-how explanations where arg-explanations and epsd-explanations are concatenated. The max. number of different clusters is double the number of arguments.

- Along with all the previous stuff, we devised also an ad-hoc strategy for dropping experience from a cluster, given that a cluster can only contain a fixed amount of batches (N.B. in vanilla DQN each batch contains only one transition).
- The drop strategies we found useful are the following:
    - PDrop: Drop the batch with the lowest priority with probability p otherwise drop the oldest one (as in vanilla DQN).
    - Global Distribution Matching: To every batch assign a random number in [0,1] during insertion. Drop the batch with the lowest random number. Idea from: *Isele, David, and Akansel Cosgun. "Selective experience replay for lifelong learning." Thirty-second AAAI conference on artificial intelligence. 2018.*

- We run some experiments. Apparently, in XADQN (but not in XADDPG and XAPPO), cluster_prioritisation is heavily worsening performance by causing value over-estimation. The best clustering scheme in many experiments seems to be epsd_arg, combining how and why explanations. The best drop strategy seems to be PDrop with p=0.5; a p=1 is apparently causing a heavy value over-estimation (worse performance).