Offered Term Project Topics and Criteria for Alternatives

Project 1: Neural episodic locomotion control

Develop a variant of the Neural Episodic Control (NEC) algorithm for locomotion control with continuous state and action sets. Modify the algorithm in an original way that will make it feasible also for continuous action spaces. As a hint, aim to remove the need for a perpetually growing neural dictionary. You can for instance fix its size and interact with it via an attention mechanism. Evaluate your solution on at least three Mujoco environments, for instance Hopper, Half Cheetah, and Ant. Replicate your experiments multiple times and report the evolution of the evaluation-time total reward as a function of the gradient-descent steps. Your report should contain:

i) All the properties of the NEC algorithm that limit its applicability to locomotion control in addition to the ones hinted above.
ii) The modifications you make in the algorithm and how they address the limitations you stated in item (i).
iii) The resulting algorithm as a clean and descriptive pseudo-code
iv) How well the resulting model is solving the tasks compared to the state of the art (e.g. SAC, TD3).
v) What the limitations of your own algorithm are. How can they be improved further in future work.
vi) How sensitive the performance of your algorithm is to the episodic memory size.

Project 2: Learning to play Atari games with Bayesian Deep Expected Sarsa

Devise an Expected Sarsa variant to solve the pixel-to-control Atari game playing task that uses a Bayesian neural network to model the Q factor. The algorithm should derive a soft policy directly from the Bayesian Q-network, owing its softness to the randomness of the network weights. The model should then integrate out this uncertainty in the target calculation. Sticking to the original Sarsa approach, the algorithm should be on-policy, i.e. it should not have a replay buffer that contains data points collected from a different policy from the one being learned. One way to handle this issue is to interact with the environment with a fixed policy for a tunable number of steps (e.g. 100 steps), then training the Bayesian Q-network on the newly collected (100) data points and using the current Bayesian Q-network as the prior distribution for the next inference round. Evaluate your model on three Atari games (e.g. “Breakout-v0”, “SpaceInvaders-v0”, “Tennis-v0”). Replicate your experiments multiple times and report the evolution of the evaluation-time total reward as a function of the gradient-descent steps. Your report should contain:

i) Your Bayesian Q-factor network design
ii) The steps you take towards obtaining an Expected Sarsa variant.
iii) The resulting algorithm as a clean and descriptive pseudo-code
iv) How well the resulting model is solving the tasks compared to the state of the art (e.g. DQN, DDQN, Soft Q-learning etc.)
v) What the limitations of your own algorithm are. How can they be improved further in future work.
vi) How sensitive the performance of your algorithm is to the model update interval, after how many environment interactions with a fixed policy the Bayesian Q-network is updated.

Project 3: Solving the overestimation bias in Deep RL.

Find existing and understandable PyTorch implementations of the Soft Actor Critic (SAC) and Twin Delayed Deep Deterministic Policy Gradient (TD3) algorithms. Run them on at least three locomotion control tasks, e.g. Hopper, Half Cheetah, Ant and plot their overestimation curves, i.e. the difference

between the discounted total reward predicted by the critic (Q) networks at the beginning of an episode and the observed discounted total reward for that episode (see Figure 2 of the TD3 paper). Devise a new target estimator that could mitigate the overestimation bias. You can consider the following options: i) building an ensemble of critic networks, i.e. using more than two of them, ii) building a "bootstrap" ensemble of the critic networks, i.e. training them concurrently but on different minibatches, iii) as the target value of a state-action pair using the mean minus the standard deviation of the ensemble outcomes. iv) trying item (iii) for both (i) and (ii). Replicate your experiments multiple times and report the evolution of the evaluation-time total reward as a function of the gradient-descent steps as well as the overestimation curves. This is only a suggestion; you can also come up with other solutions. Your report should contain:

i) A rigorous description of your target estimator design
ii) A justification about why it should mitigate the overestimation bias
iii) The total episode reward and overestimation curves for updated versions of SAC and TD3 on each environment separately
iv) An interpretation that addresses the match of results in item (iii) with the justifications provided in (ii)
v) The limitations of the developed estimator referring to the situations where it may overestimate.
vi) The sensitivity of the results on the hyperparameters of the estimator, such as the number of ensemble elements.

In all projects, train the models as long as your computation resources permit. If a run result is predicted to take unacceptably long, feel free to reduce the maximum number of episodes or the capacities of the used function approximators.

Your own project idea is suitable as a course project if your answers to the questions below are "yes":

i) Is your project topic relevant to the course material comparable to the projects proposed above?
ii) Do you modify the existing RL algorithm you build on as much as the projects proposed above?
iii) Do you evaluate the performance of your algorithm on comparably many different settings as the projects above?
iv) Do you explain the outcome of your evaluation using the course material at a level of depth comparable to the examples above?