Week 7 Thursday Paper 1: Soft Actor-Critic

Explained by: Soft Actor-Critic - Spinning Up documentation

Supplemental: AlphaGo

1. What does the paper's introduction / related work section say about drawbacks for on-policy learning, off-policy learning, and their combination, respectively?

Ans: On-policy learning suffers from poor sample efficiency since it requires new samples to be collected for each gradient step, which becomes extremely expensive as the number of gradient steps needed to learn an effective policy increases with task complexity.

Off-policy learning, while it aims to reuse past experiences for efficiency, combining off-policy learning with high-dimensional nonlinear function approximation presents significant challenges for stability and convergence, especially in continuous state and action spaces.

Each has its limitations which might not be easily overcome by simply combining them without addressing the fundamental issues of sample efficiency and stability.

2. What role does the Entropy term play in policy iteration? Explain in your words why it is called a "soft" actor-critic?

Ans: The entropy term incentivizes the policy to explore more widely and act as randomly as possible within the constraints of succeeding at the task, which is why it's called soft actor-critic. This entropy maximization leads to improved exploration and robustness, and prevents premature convergence by encouraging the policy to explore different strategies.

3. What does it mean by using "two Q-functions"? Why is it helpful?

Ans: The approach of using two Q-functions in SAC is to mitigate the positive bias in the policy improvement step, which is known to degrade the performance of value-based methods. By parameterizing two Q-functions and training them independently, then taking the minimum of these two for the value gradient and policy gradient, the algorithm can improve stability and learning efficiency.

4. What are the main hyperparameters used in Soft Actor-Critic training? What purposes do they serve?

Ans: Temperature parameter: Controls the relative importance of the entropy term against the reward, hence controlling the stochasticity of the optimal policy.

Learning rates: For updating the policy, value, and Q-functions.

Replay buffer size: Affects the variety and number of past experiences available for learning.

Batch size: Influences the sample variance of the gradient estimates.

Target smoothing coefficient and update intervals: Used for updating the target networks to improve learning stability.

5. Compare SAC with DDPG: which one should one use when?

Ans: SAC should be preferred when sample efficiency and stability are of high concern and dealing with complex, high-dimensional tasks where exploration is crucial.

DDPG might still be used in scenarios where a deterministic policy is preferred, and the task environment is less complex or when computational resources are limited. However, SAC generally outperforms DDPG, especially on more difficult and higher-dimensional tasks, due to its better exploration strategies and stability.

Week 7 Thursday Paper 2: Decision Transformer

Explained by: Blog

Supplemental: Q-Learning Decision Transformer

1. What are the major components of the Decision Transformer? How is it different from the traditional transformer architecture?

Ans: The Decision Transformer is a variant of the transformer architecture used in language modeling. It employs modality-specific linear embeddings for states, actions, and returns, which are then combined with a positional episodic timestep encoding. The model uses a GPT architecture to predict actions autoregressively, leveraging a causal self-attention mask. This is different from traditional transformers as it is specifically adapted for reinforcement learning by incorporating returns-to-go (sum of future rewards) and focusing on generating actions that lead to desired outcomes.

Traditional transformer architectures, like those used in language processing, do not typically incorporate returns-to-go or are not directly aimed at decision-making tasks. In contrast, the Decision Transformer specifically models trajectories (sequences of rewards, states and actions) to generate actions that lead to desired returns, adapting the transformer architecture to the reinforcement learning context

2. Explain why using a sequential modeling approach makes sense in the context of Reinforcement Learning.

Ans: Sequential modeling aligns naturally with reinforcement learning as both involve making decisions over time. In RL, decisions taken at each step depend on the current state and potentially previous states and actions, which is inherently sequential. The sequential modeling approach, exemplified by the Decision Transformer, aligns with this by treating RL as a sequence modeling problem, thus leveraging the power of transformers in handling long-range dependencies and contexts, which is beneficial for tasks requiring long-term credit assignment and decision-making based on historical data.

3. How do we train a Decision Transformer? What are the main hyperparameters and design options we need to choose from?

Ans: The Decision Transformer is trained on sequences of states, actions, and returns from an offline dataset using a sequence modeling objective. This approach bypasses the need for conventional RL techniques like temporal difference learning, thus avoiding issues like the deadly triad and bootstrapping errors. Instead, the model learns to generate actions leading to desired outcomes by being conditioned on past states, actions, and desired returns.
The main hyperparameters include the number of transformer layers, the number of attention heads, the embedding dimension, and the sequence length. Additionally, design choices like the specific linear embeddings for states, actions, and returns, and whether to use additional encodings like positional embeddings, are crucial

4. What are some potential problems of using a transformer instead of value functions in the RL environment?

Ans: Using transformers instead of traditional value functions in RL can lead to challenges such as managing the computational complexity due to the self-attention mechanism, especially in environments with long sequences or large state spaces. Additionally, transformers might overfit to the offline data they were trained on, potentially leading to poor generalization in environments not covered by the training data. Moreover, the lack of explicit exploration mechanisms, typical in model-free RL algorithms, might limit the model's ability to learn optimal policies when transitioning from offline to online settings

5. Is this algorithm model-dependent? Is it on or off-policy?

Ans: The Decision Transformer is model-agnostic in the sense that it does not explicitly learn a model of the environment's dynamics but instead focuses on generating action sequences that lead to desired outcomes. It is trained off-policy since it learns from a fixed dataset of collected experiences rather than from online interactions with the environment