Terminology:

- Scene: a specific time step t and its corresponding state s_t (within an episode)
- Episode: Full sequence of scenes
- Memory: a 4-tuple (s_t, a_t, r_t, s_{t+1})
- Trajectory: synonymous with episode. I.e., a trajectory is a sequence of memories that reaches a terminal state or some t_{max}.

System overview: Actor-Critic

- Critic $\mathcal{C}$ is learning a value $V_\theta(s)$ that critiques the actor. The critic $\mathcal{C}$ uses attention to train on trajectories from prior experiences. This would happen through self-supervised "leave one out" predictions in simialar fashion to the BERT / GPT-2 text generation tasks.
- TD-error is computed from $V_\theta(s)$:

$$E_{TD} = r_t + \gamma V_\theta(s_{t+1}) - V_\theta(s_t)$$
$$R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$$

- Actor $\mathcal{A}$ (an MLP or something) acts based on sampled actions from a policy. The state s_t and weights $\boldsymbol{\theta}$ are taken in by the actor $\mathcal{A}$ to output a probability distribution of actions $\pi_\theta(s_t)$. Sample from that probability distribution to select an action $a_t \sim \pi_\theta(s_t)$.

Algorithm 1: Minibatch SGD training of GAN. The number of steps to apply the discriminator, k, is a hyperparameter. $k = 1$ is the least expensive option.

for number of training episodes **do**
    while not in terminal state $(t < T)$ **do**
        In state s_t, select action from the policy: $a_t \sim \pi_\theta(s_t)$
        Perform action a_t to retrieve s_{t+1}, r_t.
        Memorize the memory $M_t := (s_t, a_t, s_{t+1}, r_t)$ for training the critic.
        if terminal state **then**
            Compute discounted rewards vector $\boldsymbol{R}$ with discount factor γ: $\boldsymbol{R} = \sum_{t=0}^{T} \gamma^t r_t \hat{e}_t$
            Save the trajectory sequence $\boldsymbol{\tau} = [M_0, \cdots, M_T]$
            break
        else
            $s_t \leftarrow s_{t+1}$
            $t \leftarrow t + 1$
        end
    end
    # Train critic in order to update the policy. Compute values vector
    $\boldsymbol{V} = [V_\theta(s_0), V_\theta(s_1), \ldots, V_\theta(s_{t_f})]$ Value = BERT($\boldsymbol{\tau}$) Policy
end
The gradient-based updates can use any standard gradient-based learning rule.

Attention: Self-supervised "leave one out" training on trajectory inputs. This is similar to how BERT / GPT-2 do self-supervised training on sentences to build context for words. We would do this with trajectories to build context on memories, where the attention mechanism predicts

- Hidden layer embeddings are able to learn context-dependent information.

Training the attention part:

Input time (t): 0 1 2 3 _ 5 6 7 8

For predicting t=4, we could input s_4, a_4, and predicts s_5

What's missing at $t = 4$: s_4, a_4

Predicts $mathcalR$ -¿ expected reward γr_{4-8}

RL agent (live) :

Then, when we give at seq like below: Input time(t): 0 1 2 3 4 _

Right, we're in s_4 and want to pick a_4 and it would be helpful to know r_4 and s_5.

$Q_{att}(s_t, a)$: attention-value of state-action pair

$pi(s_t) = max_a[Q(s_4, a) + Q_{att}(s_4, a)]$

—-

Generate N trajectories, where N is a hyperparameter.

Attention trains on those trajectories in self-supervised setting to predict the expected reward?