Additional Details for Presentation

Nikhil Challa

June 8, 2022

1 Formulas

Given the agent in state s_t it executes the action

$$a_t \sim \pi(s_t; \theta_P) \tag{1}$$

θ_P is optimized to maximize the expected sum of rewards using A3C policy gradient

$$\max_{\theta_P} \mathbb{E}_{\pi(s_t;\theta_P)} \left[\sum_t r_t \right] \tag{2}$$

With taking an action, we get s_{t+1}, but as we indicated earlier, we cannot use image directly!
Instead we transform raw states into feature vector

$$s_t \to \phi(s_t); \quad s_{t+1} \to \phi(s_{t+1}) \tag{3}$$

Inverse Dynamics Model
Given the states, we attempt to predict the action a_t taken by agent to move from state s_t to s_{t+1}.
Since we have trainable parameters θ_I, we also need to define the loss function. Note that only Inverse dynamics model can influence the state to feature vector conversion.

$$\hat{a}_t = g\left(s_t, s_{t+1}; \theta_I\right) \tag{4}$$

$$\min_{\theta_I} L_I(\hat{a}_t, a_t) \tag{5}$$

Forward Dynamics Model
Given the current feature and action, we attempt to predict feature encoding at time $t+1$
Since we have trainable parameters θ_F, we also need to define the loss function.

$$\hat{\phi}(s_{t+1}) = f\left(\phi(s_t), a_t; \theta_F\right) \tag{6}$$

$$\min_{\theta_F} L_F\left(\hat{\phi}(s_{t+1}), \phi(s_{t+1})\right) \tag{7}$$

We need to determine the intrinsic reward!

$$r_t^i = \frac{\eta}{2} \left|\left| \hat{\phi}(s_{t+1}) - \phi(s_{t+1}) \right|\right|_2^2 \tag{8}$$

Overall optimization problem

$$\min_{\theta_P \theta_I \theta_F} \Bigg[-\lambda \underbrace{\mathbb{E}_{\pi(s_t;\theta_P)} \left[\sum_t r_t \right]}_{\text{A3C Model}} + (1-\beta) \underbrace{L_I\Big(\overbrace{g(s_t, s_{t+1}; \theta_I)}^{\hat{a}_t}, a_t \Big)}_{\text{Inversion Dynamics Model}} + \beta \underbrace{L_F\Big(\overbrace{f\big(\phi(s_t), a_t; \theta_F\big)}^{\hat{\phi}(s_{t+1})}, \phi(s_{t+1}) \Big)}_{\text{Forward Dynamics Model}} \Bigg] \tag{9}$$

$\mathbb{E}_{\pi(s_t;\theta_P)}$: Expected sum of rewards
L_I : Loss function for Inverse Dynamics Model
g : Function to predict action given current and new feature vectors
L_F : Loss function for Inverse Dynamics Model
f : Function to predict new state vector
η : Scaling factor
β : weights the inverse model loss against the forward model loss
λ : weights the policy gradient loss against the importance of learning the intrinsic reward signal.

2 Full Flow

1. Randomly initialize the weights/parameters for the global network. For each worker, we create a thread thereby creating agents that explore/learn indepedently in their own environment. The parameter values are synchronized from global network to each thread.

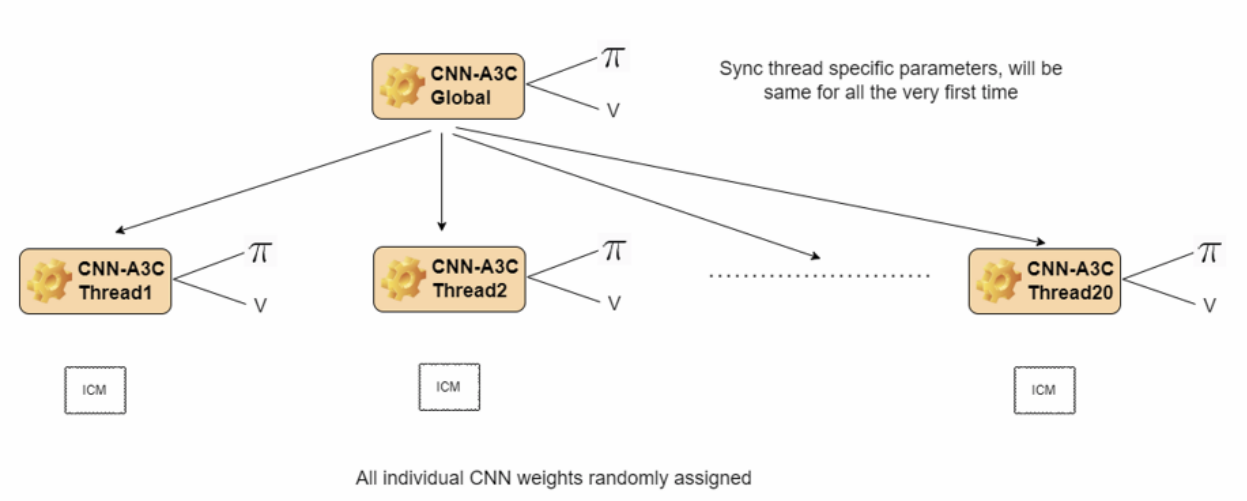

Figure 1: Initialization

Relevant pseudo code: [Algorithm S3 from Asynchronous Methods for Deep Reinforcement Learning]

Initialize thread step counter $t \leftarrow 1$
repeat
Reset gradients: $d\theta \leftarrow 0$ and $d\theta_v \leftarrow 0$
Synchronize thread-specific parameters $\theta' = \theta$ and $\theta'_v = \theta_v$
$t_{start} = t$

---

2. Now we focus on one thread, but same can be thought of for all threads running independently. Given the current image (current state), we get the policy values which is probability of each action. The agent picks and performs the action (sample from distribution) in the world. This gives us new image (new state).

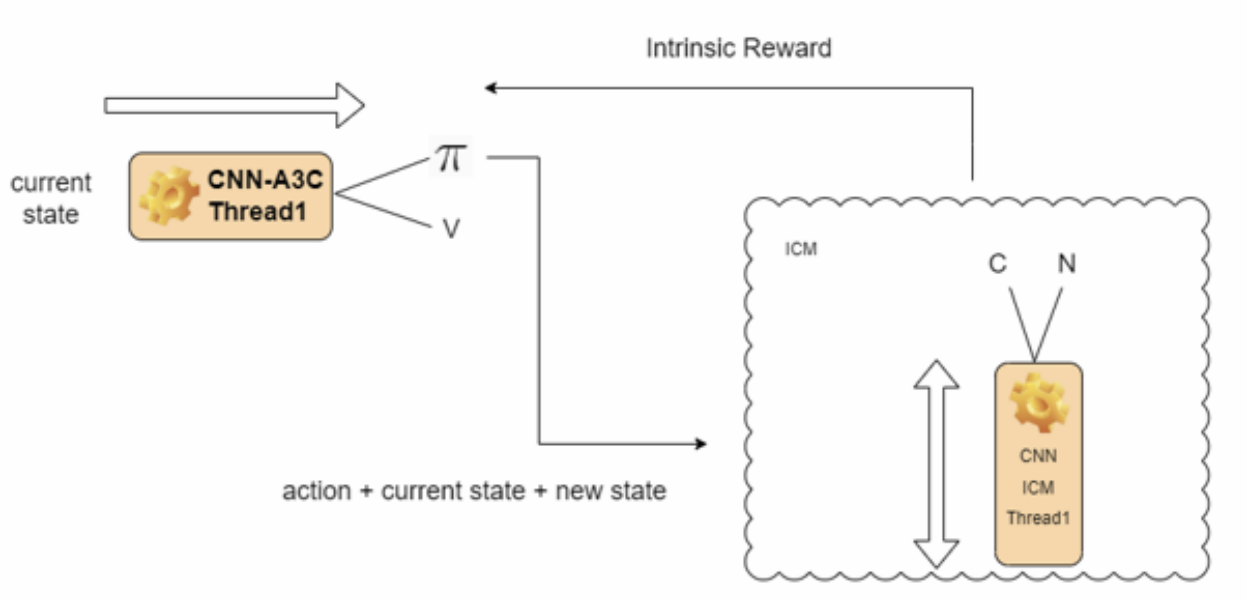

Figure 2: Training in ICM Module

Using the action/current state and new state, the CNN within ICM module undergoes training to minimize equation (9).

Relevant pseudo code: [Algorithm S3 from Asynchronous Methods for Deep Reinforcement Learning]

---

repeat
  Perform a_t according to policy $\pi(a_t|s_t;\theta')$
  Receive reward r_{t+1} and new state s_{t+1}
  $t \leftarrow t+1$
  $T \leftarrow T+1$
until terminal s_t or $t - t_{start} == t_{max}$

---

3. Once any of the threads either hits terminal state or reaches end of episodic time limit, the worker calculates the accumulated gradient. The worker then applies Adam optimizer and uses final gradient value to update the global network.

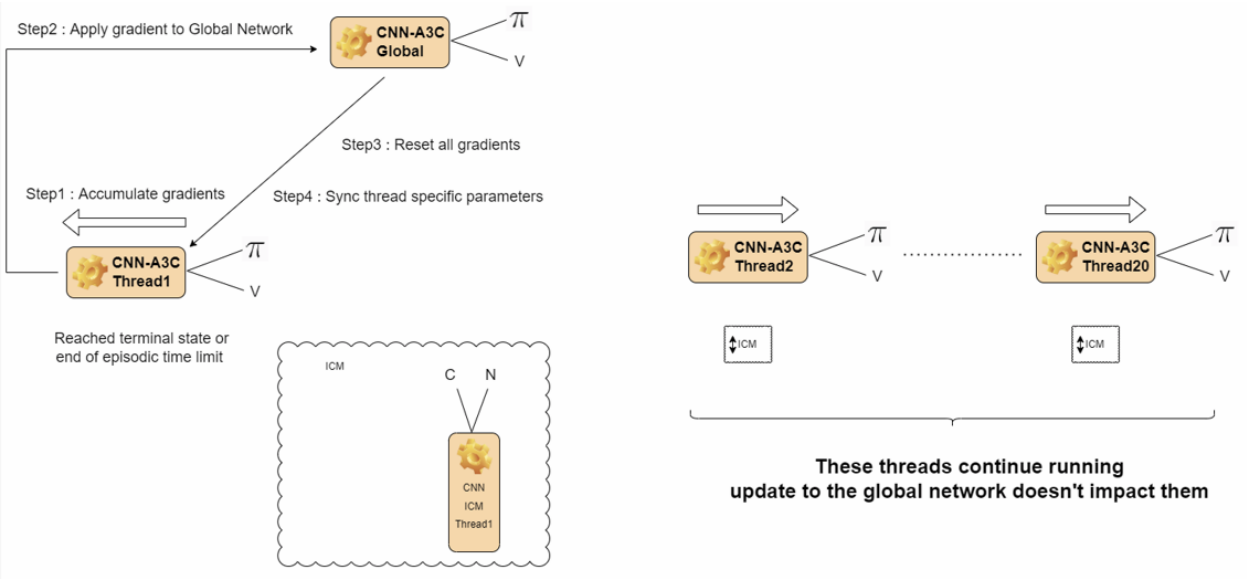

Figure 3: Accumulate gradient

Relevant pseudo code: [Algorithm S3 from Asynchronous Methods for Deep Reinforcement Learning]

$R = \begin{cases} 0 & \text{for terminal } s_t \\ V(s_t, \theta'_v) & \text{for non-terminal } s_t \text{ Boot strap from last state} \end{cases}$
for $i \in \{t-1, \cdots, t_{start}\}$ **do**
  $R \leftarrow r_i + \gamma R$
  Advantage : $A = (R - V(s_i; \theta'_v))$
  Accumulate gradients wrt θ' : $d\theta + \leftarrow d\theta + \nabla\theta' \log \pi(a_i|s_i; \theta')(A)$
  Accumulate gradients wrt θ'_v : $d\theta_v + \leftarrow d\theta_v + \partial(A)^2/\partial\theta'_v$
end for

We need to factor in Adam optimizer, below are equations

Assume $g_t \leftarrow d\theta$

$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$

$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2$

$\hat{m}_t = \frac{m_t}{1-\beta_1^t}$

$\hat{v}_t = \frac{v_t}{1-\beta_2^t}$

new $d\theta = \frac{\alpha \hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$

Do similar steps for dθ_v

We then perform aynchronous update of θ using d θ and of θ_v using d θ_v

4. Each thread can apply the gradient to the global network aynchronously upon hitting terminal state or reaching end of episodic time limit and start fresh by resetting the gradient values and syncing to global network parameters.

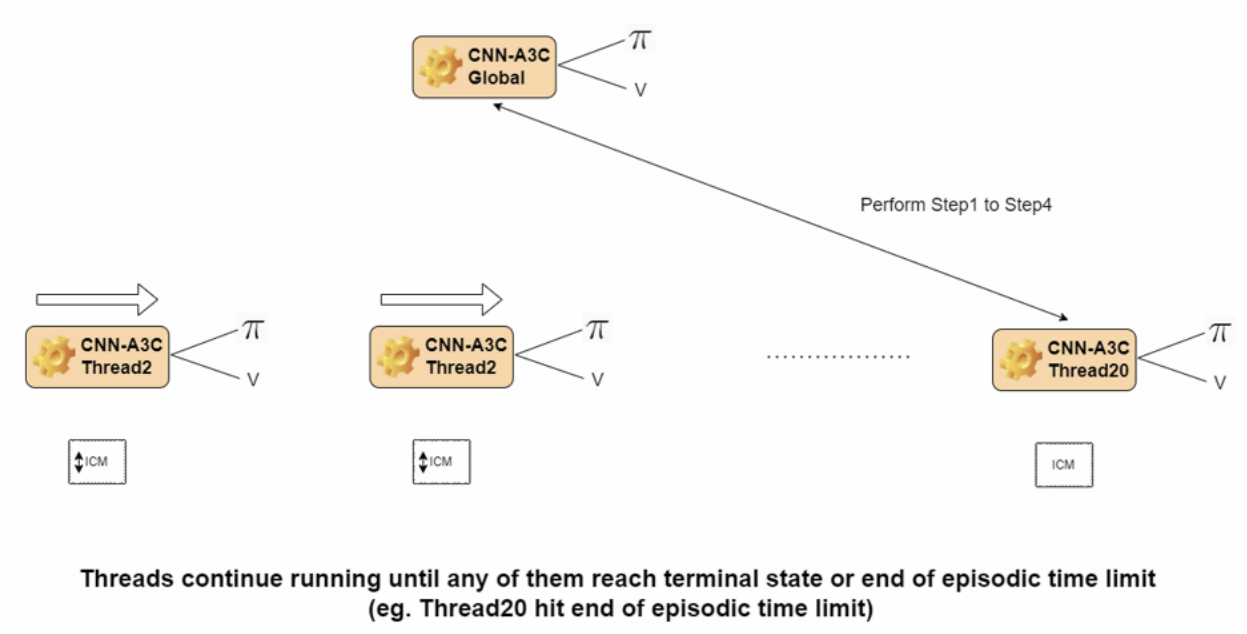

Figure 4: Accumulate gradient

Relevant pseudo code: [Algorithm S3 from Asynchronous Methods for Deep Reinforcement Learning]

Perform asynchronous update of θ using d θ and of θ_v using d θ_v
until $T > T_{max}$

This process continues until we reach end of training period denoted as T_{max}. For subsequent testing we use the global network directly and apply greedy approach!