Blockwise Sequential Model Learning for Partially Observable Reinforcement Learning

Giseung Park, Sungho Choi, Youngchul Sung

School of Electrical Engineering, KAIST, Korea
{gs.park, sungho.choi, ycsung}@kaist.ac.kr

Abstract

This paper proposes a new sequential model learning architecture to solve partially observable Markov decision problems. Rather than compressing sequential information at every timestep as in conventional recurrent neural network-based methods, the proposed architecture generates a latent variable in each data block with a length of multiple timesteps and passes the most relevant information to the next block for policy optimization. The proposed blockwise sequential model is implemented based on self-attention, making the model capable of detailed sequential learning in partial observable settings. The proposed model builds an additional learning network to efficiently implement gradient estimation by using self-normalized importance sampling, which does not require the complex blockwise input data reconstruction in the model learning. Numerical results show that the proposed method significantly outperforms previous methods in various partially observable environments.

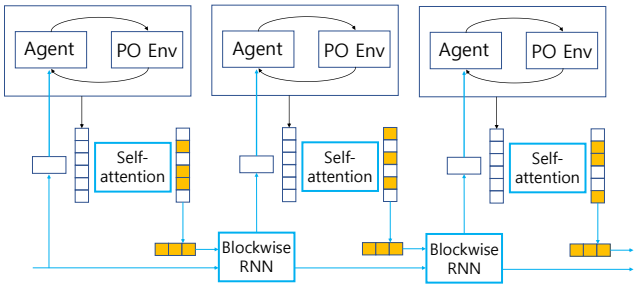


Figure 1: Overview: self-attention and RNN are used to process each block of sequential data instead of processing at every timestep in partially observable (PO) environments.

Introduction

Reinforcement learning (RL) in partially observable environments is usually formulated as partially observable Markov decision processes (POMDPs). RL solving POMDPs is a challenging problem since the Markovian assumption on observation is broken. The information from the past should be extracted and exploited during the learning phase to compensate for the information loss due to partial observability. Partially observable situations are prevalent in real-world problems such as control tasks when observations are noisy, some part of the underlying state information is deleted, or long-term information needs to be estimated (Han, Doya, and Tani 2020b; Meng, Gorbet, and Kulic 2021).

Although many RL algorithms have been devised and state-of-the-art algorithms provide outstanding performance in fully observable environments, relatively fewer methods have been proposed to solve POMDPs. Previous POMDP methods use a recurrent neural network (RNN) either to compress the information from the past in a model-free manner (Hausknecht and Stone 2015; Zhu, Li, and Poupart 2017; Goyal et al. 2021) or to estimate the underlying state information and use the estimation result as an input to the RL agent (Igl et al. 2018; Han, Doya, and Tani 2020b). These methods compress observations in a step-by-step sequential order in time, which may be inefficient when partiality in observation is high and less effective in extracting *contextual* information within a time interval.

We conjecture that observations at specific timesteps in a given time interval contain more information about decision-making. We propose a new architecture to solve partially observable RL problems by formalizing this intuition into a mathematical framework. Our contributions are as follows:

- As shown in Fig. 1, we propose a new learning architecture based on a **block** of sequential input rather than estimating a latent variable at every timestep by jointly using self-attention (Vaswani et al. 2017) and RNN and exploiting the advantage of each structure.
- To learn the proposed architecture, we present a blockwise sequential model learning based on direct gradient estimation using self-normalized importance sampling (Bornschein and Bengio 2015; Le et al. 2019), which does not require input data reconstruction in contrast to usual variational methods to POMDPs (Chung et al. 2015; Han, Doya, and Tani 2020b).
- Using the proposed blockwise representations of the proposed model and feeding the learned block variables to the RL agent, we significantly improved the performance over existing methods in several POMDP environments.

Related Work

In partially observable RL, past information should be exploited appropriately to compensate for the information loss in the partial observation. RNN and its variants (Hochreiter and Schmidhuber 1997; Cho et al. 2014) have been used to process the past information. The simplest way is that the output of RNN driven by the sample sequence is directly fed into the RL agent as the input capturing the past information without further processing, as considered in previous works (Hausknecht and Stone 2015; Zhu, Li, and Poupart 2017). The main drawback of these end-to-end approaches is that it requires considerable data for training RNN and is suboptimal in some complicated environments (Igl et al. 2018; Han, Doya, and Tani 2020b).

Goyal et al. (2021) proposed a variant of RNN in which the hidden variable is divided into multiple segments with equal length. First, a fixed number of the segments are selected using attention (Vaswani et al. 2017). Then, only the selected segments are updated with independent RNNs followed by self-attention, and the remaining segments are not changed. Our approach is substantially different from this method in that we use attention over a time interval, while the structure of Goyal et al. (2021) is updated stepwise by using the attention over the segments at the same timestep.

Other methods estimate state information or belief state by learning a sequential model of stepwise latent variables. The inferred latent variables are then used as input to the RL agent. Igl et al. (2018) proposed estimating the belief state by applying a particle filter (Maddison et al. 2017; Le et al. 2018; Naesseth et al. 2018) in variational learning. Han, Doya, and Tani (2020b) proposed a Soft Actor-Critic (Haarnoja et al. 2018) based method (VRM) focusing on solving partially observable continuous action control tasks. VRM adds action sequence as additional input and uses samples from the replay buffer to maximize the variational lower bound (Chung et al. 2015). Then, latent variables are generated as input to the RL agent. To solve the stability issue, however, VRM concatenates (i) a pre-trained and frozen variable d_{freeze}, and (ii) a learned variable d_{keep} from the distinct model because using only d_{keep} as input to the RL agent does not yield performance improvement. In contrast, our method only uses one block model for learning, which is more efficient.

While previous methods improved performance in partially observable environments, they mostly use RNN. The RNN architecture suffers two problems when partiality in observation is high: (i) the forgetting problem and (ii) the inefficiency of stepwise compressing all the past samples, including unnecessary information such as noise. Our work solves these problems by learning our model blockwise by passing the most relevant information to the next block.

Background

Setup We consider a discrete-time POMDP denoted by $(\mathcal{S}, \mathcal{A}, P, r, \gamma, \mathcal{O}, \Omega)$, where $\mathcal{S}$ and $\mathcal{A}$ are the state and action spaces, respectively, $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}^+$ is the state transition probability distribution, $r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the reward function, $\gamma \in [0, 1]$ is the discounting factor, and $\mathcal{O}$ and Ω are the observation space and observation probability, respectively. Unlike in a usual MDP setting, the agent cannot observe the state s_t at timestep t in POMDP, but receives an observation $o_t \in \mathcal{O}$ which is generated by the observation probability $\Omega : \mathcal{S} \times \mathcal{A} \times \mathcal{O} \to \mathbb{R}^+$. Our goal is to optimize policy π to maximize the expected discounted return $\mathbb{E}_\pi[\sum_{t=0}^{\infty} \gamma^t r_t]$ by learning π with a properly designed input variable to π in addition to o_t in place of the unknown true state s_t at each timestep t.

Self-attention Self-attention (Vaswani et al. 2017) is an architecture that can perform a detailed process within a time interval by considering contextual information among sequential input data in the interval. Consider a sequential input data of length L, denoted by $B = x_{1:L} \triangleq [x_1, x_2, \cdots, x_L]^\top \in \mathbb{R}^{L \times d}$, where $x_i \in \mathbb{R}^d$ (column vector), $1 \le i \le L$, and $(\cdot)^\top$ denotes matrix transpose. (The notation $A_{m_1:m_2} \triangleq [A_{m_1}, A_{m_1+1}, \cdots, A_{m_2}]$ for any quantity A will be used in the rest of the paper.) Self-attention architecture transforms each input data x_i in B into y_i so that the transformed representation y_i contains information in not only x_i but also all other $x_j \in B$, reflecting the relevance to the target task. (See Appendix A for the structure.)

To improve the robustness of learning, self-attention is usually implemented with m (> 1) multi-head transformation. Let the $d \times d$ transform matrices of query, key, and value be $M^Q = [M_1^Q, M_2^Q, \cdots, M_m^Q]$, $M^K = [M_1^K, M_2^K, \cdots, M_m^K]$, $M^V = [M_1^V, M_2^V, \cdots, M_m^V]$, respectively, where $d = mh_{\text{head}}$ so that $M_l^Q, M_l^K, M_l^V \in \mathbb{R}^{d \times h_{\text{head}}}$ for each $1 \le l \le m$. The l-th query, key, and value are defined as $BM_l^Q, BM_l^K, BM_l^V \in \mathbb{R}^{L \times h_{\text{head}}}$, respectively. Using an additional transform matrix $M^O \in \mathbb{R}^{mh_{\text{head}} \times d} = \mathbb{R}^{d \times d}$, the output of multi-head self-attention $\text{MHA}(B) \in \mathbb{R}^{L \times d}$ is given by

$$\begin{aligned} &\text{MHA}(B) = [A_1, A_2, \cdots, A_m]M^O, \ \text{ where} \\ &A_l = f\left(\frac{(BM_l^Q)(BM_l^K)^\top}{\sqrt{h_{\text{head}}}}\right)(BM_l^V) \in \mathbb{R}^{L \times h_{\text{head}}} \end{aligned} \tag{1}$$

and f is a row-wise softmax function (other pooling methods can be used in some cases (Richter and Wattenhofer 2020)).

In practice, residual connection, layer normalization (Ba, Kiros, and Hinton 2016), and a feed-forward neural network g are used to produce the final $L \times d$ representation $Y = [y_1, y_2, \cdots, y_L]^\top$:

$$\begin{aligned} Y &= \text{LayerNormalize}(g(U) + U), \ \text{ where} \\ U &= \text{LayerNormalize}(\text{MHA}(B) + B). \end{aligned} \tag{2}$$

Note that the self-attention architecture of $B \to Y$ can further be stacked multiple times for deeper representation.

Unlike RNN, however, in self-attention, each data block is processed without consideration of the previous blocks, and hence each transformed block data is disconnected. Therefore, information from the past is not used to process the current block. In contrast, RNN uses past information by stepwise accumulation, but RNN suffers the forgetting problem when the data sequence becomes long.

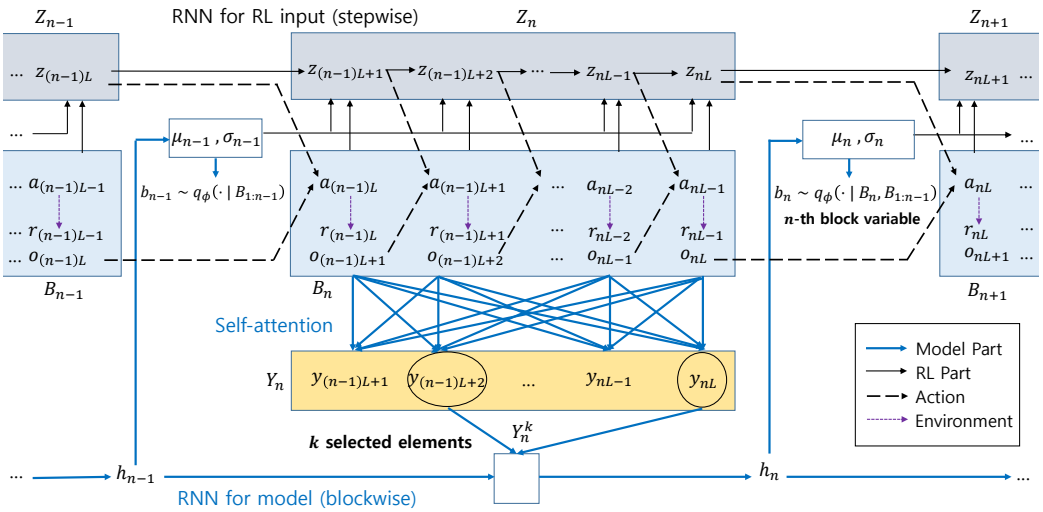


Figure 2: Detailed architecture of the proposed model: An analogy to filter theory can be drawn. RNN corresponds to autoregressive (AR) filtering, which performs recursive filtering only, attention corresponds to moving-average (MA) filtering, which performs block processing, and the proposed new architecture corresponds to autoregressive–moving-average (ARMA) filtering.

Proposed Method

We present a new architecture for POMDPs, modeling blockwise latent variables by jointly using self-attention and RNN and exploiting the advantage of each structure. The proposed architecture consists of (i) stepwise RNN for RL input and (ii) block model. If only the stepwise RNN is used, it corresponds to the naive RNN method. As shown in Figs. 1 and 2, the block model consists of self-attention and blockwise RNN. After the self-attention compresses block information, the blockwise RNN passes the information to the next block.

We describe the block model structure including how the block model is used for the trajectory generation and how the block information is compressed. We then explain how the block model is efficiently learned to help the RL update.

Proposed Architecture

We consider a sample sequence $\{x_t, t = 1, \cdots, T\}$ of length $T = NL$, where the sample $x_t \in \mathbb{R}^d$ at timestep t is given by the column-wise concatenation $x_t = [a_{t-1}; r_{t-1}; o_t]$ of action a_{t-1}, reward r_{t-1}, and the partial observation o_t. We partition the sample sequence $\{x_t, t = 1, \cdots, T\}$ into N blocks $B_1, \cdots, B_N$, where the n-th block $B_n \in \mathbb{R}^{L \times d}$ $(1 \le n \le N)$ is given by

$$B_n = x_{(n-1)L+1:nL} = [x_{(n-1)L+1}, x_{(n-1)L+2}, \cdots, x_{nL}]^\top.$$

We then decompose the model-learning objective as

$$\log p_\theta(x_{1:T}) = \sum_{n=1}^{N} \log p_\theta(B_n | B_{1:n-1}), \tag{3}$$

instead of conventional sample-by-sample decomposition $\log p_\theta(x_{1:T}) = \sum_{t=1}^{T} \log p_\theta(x_t | x_{1:t-1})$. Note that the conditioning term in (3) is $B_{1:n-1}$ not B_{n-1} since the full Markovian assumption is broken in partially observable environments. Based on this block-based decomposition, we define the generative model $p_\theta(B_{1:N}, b_{1:N})$ and the inference model $q_\phi(b_{1:N} | B_{1:N})$ as

$$p_\theta(B_{1:N}, b_{1:N}) = \prod_{n=1}^{N} p_\theta(B_n, b_n | B_{1:n-1}),$$
$$q_\phi(b_{1:N} | B_{1:N}) = \prod_{n=1}^{N} q_\phi(b_n | B_n, B_{1:n-1}), \tag{4}$$

where b_n is a latent variable containing the information of the n-th block B_n.

For each block index n, we want to infer the n-th block variable b_n from the amortized posterior distribution q_ϕ in (4) by using (i) the information of the current block B_n and (ii) the information from past blocks $B_{1:n-1}$ before B_n. After inferring b_n at the n-th block, the information of b_n is used to generate the input variables $Z_n \triangleq z_{(n-1)L+1:nL}$ to the RL agent. Then, the RL agent learns policy π based on $\{(z_t, o_t, a_t, r_t),\ t = 1, 2, 3, \cdots\}$, where z_t is extracted from Z_n containing z_t. Action a_t is taken by the agent based on partial observation o_t and additional input z_t compensating for partiality in observation o_t according to $a_t \sim \pi(\cdot | z_t, o_t)$.

Trajectory Generation Fig. 2 shows the proposed architecture with the n-th block processing as reference. The blue arrows in the lower part show the block variable inference network q_ϕ, and the solid black arrows in the upper part represent the processing network for RL learning.

Until block index $n-1$ (i.e., timestep $t = (n-1)L$), the information from the previous blocks $B_{1:n-1}$ is compressed into the variable h_{n-1}. The $(n-1)$-th block latent variable b_{n-1} is generated according to $b_{n-1} \sim q_\phi(\cdot | B_{1:n-1}) = \mathcal{N}(\mu_{n-1}, \mathrm{diag}(\sigma_{n-1}^2))$, where μ_{n-1} and σ_{n-1} are the outputs of two neural networks with input

h_{n-1}. The information of stochastic b_{n-1} is summarized in μ_{n-1} and σ_{n-1}, so these two variables together with samples $B_n = x_{(n-1)L+1:nL}$ are fed into the RL input generation RNN to sequentially generate the RL input variables $Z_n = z_{(n-1)L+1:nL}$ during the n-th block period. (Note that μ_{n-1} and σ_{n-1} capture the information in h_{n-1}.)

The stepwise RL processing is as follows: Given the RL input $z_{(n-1)L}$ and the observation $o_{(n-1)L}$ from the last timestep $t = (n-1)L$ of B_{n-1}, the RL agent selects an action $a_{(n-1)L} \sim \pi(\cdot|z_{(n-1)L}, o_{(n-1)L})$ (see the dashed black arrows). Then, the environment returns the reward $r_{(n-1)L}$ and the next observation $o_{(n-1)L+1}$. Then, the sample $x_{(n-1)L+1} = [a_{(n-1)L}; r_{(n-1)L}; o_{(n-1)L+1}]$ at timestep $t = (n-1)L+1$ together with μ_{n-1} and σ_{n-1} is fed into the stepwise RNN to produce the next RL input $z_{(n-1)L+1}$. This execution is repeated at each timestep until $t = nL$ to produce $Z_n = z_{(n-1)L+1:nL}$, and each sample x_t is stored in a current batch (on-policy) or a replay memory (off-policy).

At the last timestep $t = nL$ of the n-th block, the n-th block data $B_n = x_{(n-1)L+1:nL}$ is fed into the *self-attention network* to produce the output $Y_n \triangleq y_{(n-1)L+1:nL}$ capturing the contextual information in B_n. The procedure extracting Y_n from B_n follows the standard self-attention processing. However, instead of using all Y_n to represent the information in B_n, we select k ($< L$) elements in Y_n for data compression and efficiency. We denote the concatenated vector of the k elements by Y_n^k. Y_n^k is fed into the blockwise RNN, which compresses Y_n^k together with h_{n-1} to produce h_n. Thus, h_n has the compressed information up to the n-th block $B_{1:n}$. The impact of the self-attention network and the blockwise RNN is analyzed in the Ablation Study section.

Block Information Compression In order to select k elements from Y_n, we exploit the self-attention structure and the weighting matrix $W_l \triangleq f\left(\frac{(B_n M_l^Q)(B_n M_l^K)^T}{\sqrt{h_{\text{head}}}}\right)$ ($1 \le l \le m$, m is the number of multi-heads) appearing in (1). The p-th column of the j-th row of W_l determines the importance of the p-th row of $B_n M_l^V$ (i.e., data at timestep $(n-1)L+p$) to produce the j-th row of the attention A_l in (1).

Hence, adding all elements in the p-th column of whole matrix W_l, we can determine the overall contribution (or importance) of the p-th row of $B_n M_l^V$ to generate A_l. We choose the k positions with largest k contributions in column-wise summation of $\frac{1}{m}\sum_{l=1}^{m} W_l$ in timestep and choose the corresponding k positions in Y_n as our representative attention messages, considering the one-to-one mapping from B_n to Y_n. The effect of the proposed compression method is analyzed the Ablation Study section.

Efficient Block Model Learning

The overall learning is composed of two parts: block model learning and RL policy learning. First, we describe block model learning. Basically, the block model learning is based on maximum likelihood estimation (MLE) to maximize the likelihood (3) for given data $x_{1:T}$ with the generative model p_θ and the inference model q_ϕ defined in (4).

In conventional variational approach cases such as variational RNN (Chung et al. 2015), the generative model p_θ is implemented as the product of a prior latent distribution and a decoder distribution, and the decoder is learned to reconstruct each input sample x_t given the latent variable at each timestep t. In our blockwise setting allowing attention processing, however, learning to reconstruct the block input $B_n = x_{(n-1)L+1:nL}$ with a decoder given a single block variable b_n is challenging and complex compared to the stepwise variational RNN case.

In order to circumvent this difficulty, we approach the MLE problem based on *self-normalized importance sampling* (Bornschein and Bengio 2015; Le et al. 2019), which does not require an explicit reconstruction procedure. Instead of estimating the value of $p_\theta(B_n|B_{1:n-1})$, we directly estimate the gradient $\nabla_\theta \log p_\theta(B_n|B_{1:n-1})$ by using self-normalized importance sampling to update the generative model parameter θ (as $\theta \to \theta + c\nabla_\theta \log p_\theta(B_n|B_{1:n-1})$) to maximize the log-likelihood $\log p_\theta(B_n|B_{1:n-1})$ in (3).

The detailed procedure is as follows. (The overall learning procedure is detailed in Appendix B.) To estimate the gradient $\nabla_\theta \log p_\theta(B_n|B_{1:n-1})$, we construct a neural network parameterized by θ which produces the value of $\log p_\theta(B_n, b_n|B_{1:n-1})$. (The output of this neural network is the logarithm of $p_\theta(B_n, b_n|B_{1:n-1})$ not $p_\theta(B_n, b_n|B_{1:n-1})$ for convenience.) Using the formula $\nabla_\theta p_\theta(B_n|B_{1:n-1}) = \int \nabla_\theta p_\theta(B_n, b_n|B_{1:n-1}) db_n$, we can express the gradient $g_\theta^n := \nabla_\theta \log p_\theta(B_n|B_{1:n-1})$ as

$$g_\theta^n \approx \sum_{j=1}^{K_{\text{sp}}} \frac{w_n^j}{\sum_{j'=1}^{K_{\text{sp}}} w_n^{j'}} \nabla_\theta \log p_\theta(B_n, b_n^j|B_{1:n-1}), \qquad (5)$$

where $w_n^j = \frac{p_\theta(B_n, b_n^j|B_{1:n-1})}{q_\phi(b_n^j|B_n, B_{1:n-1})}$ and $b_n^j \sim q_\phi(b_n|B_n, B_{1:n-1})$ for $1 \le j \le K_{\text{sp}}$. (See Appendix B for the full derivation.) The numerator of the importance sampling ratio w_n^j can be computed as $\exp(\log p_\theta(B_n, b_n^j|B_{1:n-1}))$ based on the output of the constructed generative model yielding $\log p_\theta(B_n, b_n|B_{1:n-1})$. The denominator $q_\phi(b_n|B_n, B_{1:n-1})$ is modeled as Gaussian distribution $b_n \sim \mathcal{N}(\mu_n, \text{diag}(\sigma_n^2))$, where μ_n and σ_n are functions of h_n. h_n itself is a function of the blockwise RNN and the self-attention module, as seen in Fig. 2. Thus, the parameters of the blockwise RNN and the self-attention module in addition to the parameters of the μ_n and σ_n neural network with input h_n constitute the whole model parameter ϕ.

Note that proper learning of q_ϕ is required to estimate $\nabla_\theta \log p_\theta(B_n|B_{1:n-1})$ accurately in (5). For this, we learn q_ϕ to minimize

$$D_n \triangleq D_{KL}[p_\theta(b_n|B_n, B_{1:n-1})||q_\phi(b_n|B_n, B_{1:n-1})]$$

with respect to ϕ, where $D_{KL}(\cdot||\cdot)$ is the Kullback-Leibler divergence and $p_\theta(b_n|B_n, B_{1:n-1}) = \frac{p_\theta(B_n, b_n|B_{1:n-1})}{\int p_\theta(B_n, b|B_{1:n-1}) db}$ is the intractable posterior distribution of b_n from $p_\theta(B_n, b_n|B_{1:n-1})$. To circumvent the intractability of the posterior distribution $p_\theta(b_n|B_n, B_{1:n-1})$, we again use the self-normalized importance sampling technique with the constructed neural network for $\log p_\theta(B_n, b_n|B_{1:n-1})$. We estimate the negative gradient $-\nabla_\phi D_n$ in a similar way to the

gradient estimation in (5):

$$-\nabla_\phi D_n \approx \sum_{j=1}^{K_{\text{sp}}} \frac{w_n^j}{\sum_{j'=1}^{K_{\text{sp}}} w_n^{j'}} \nabla_\phi \log q_\phi(b_n^j | B_n, B_{1:n-1}), \tag{6}$$

where the samples and importance sampling ratio b_n^j and w_n^j in (5) can be used. (See Appendix B for the full derivation.)

One issue regarding the actual implementation of (5) and (6) is how to feed the current block information from B_n into p_θ and q_ϕ. In the case of q_ϕ modeled as Gaussian distribution $b_n \sim \mathcal{N}(\mu_n, \text{diag}(\sigma_n^2))$, the dependence on $B_n, B_{1:n-1}$ is through μ_n, σ_n which are functions of h_n. h_n is a function of Y_n^k and h_{n-1}, where Y_n^k is a function B_n and h_{n-1} is a function of $B_{1:n-1}$, as seen in Fig. 2. On the other hand, in the case of the neural network $\log p_\theta(B_n, b_n | B_{1:n-1})$, we choose to feed Y_n^k and b_n. Note that Y_n^k is a function of B_n, b_n is a function of $B_{1:n-1}$, and hence both are already conditioned on $B_{1:n-1}$.

The part of RL learning is described as follows. With the sample $x_t = [a_{t-1}; r_{t-1}; o_t]$ and μ_n, σ_n from the learned q_ϕ, the z_t-generation RNN is run to generate z_t at each timestep t. The input μ_n, σ_n is common to the block, whereas x_t changes at each timestep inside the block. Then, the RL policy π is learned using the sequence $\{(z_t, o_t, a_t, r_t), t = 1, 2, 3, \cdots\}$ based on standard RL learning (either on-policy or off-policy), where z_t is the side information compensating for the partiality in observation from the RL agent perspective.

The RL part and the model part are segregated by stopping the gradient from RL learning through μ_n and σ_n to improve stability when training the RL agent (Han, Doya, and Tani 2020b). The pseudocode of the algorithm and the details are described in Appendix C and D, respectively. Our source code is provided at https://github.com/Giseung-Park/BlockSeq.

Experiments

In this section, we provide some numerical results to evaluate the proposed block model learning scheme for POMDPs. In order to test the algorithm in various partially observable environments, we considered the following four types of partially observable environments:

- Random noise is added to each state: Mountain Hike (Igl et al. 2018)
- Some part of each state is missing: Pendulum - random missing version (Meng, Gorbet, and Kulic 2021)
- Memorizing long history is required: Sequential target-reaching task (Han, Doya, and Tani 2020a)
- Navigating agent cannot observe the whole map in maze: Minigrid (https://github.com/maximecb/gym-minigrid)

Note that the proposed method (denoted by Proposed) can be combined with any general RL algorithm. For the first three continuous action control tasks, we use the Soft Actor-Critic (SAC) algorithm (Haarnoja et al. 2018), which is an off-policy RL algorithm, as the background RL algorithm. Then, we compare the performance of the proposed method with (i) SAC with raw observation input (SAC), (ii) SAC aided by the output of LSTM (Hochreiter and Schmidhuber 1997), a variant of RNN, driven by observation sequences (LSTM), (iii) VRM, which is a SAC-based method for partially observable continuous action control tasks (Han, Doya, and Tani 2020b), and (iv) RIMs as a drop-in replacement of LSTM. The y-axis in the three performance comparison figures represent the mean value of the returns of the most recent 100 episodes averaged over five random seeds.

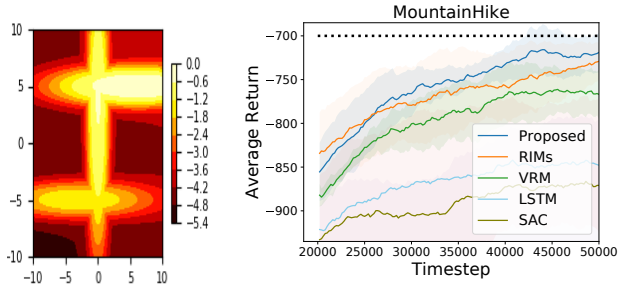


Figure 3: (a) Distribution of reward function in the Mountain Hike environment (b) Performance comparison in the Mountain Hike environment (The horizontal black dotted line shows the mean performance of SAC over five seeds at 50000 timesteps when each observation is fully observable.)

Since the Minigrid environment has discrete action spaces, we cannot use SAC and VRM, but instead, we use PPO (Schulman et al. 2017) as the background algorithm. Then the proposed algorithm is compared with PPO, PPO with LSTM, and PPO with RIMs over five seeds. (The details of the implementations are described in Appendix D.)

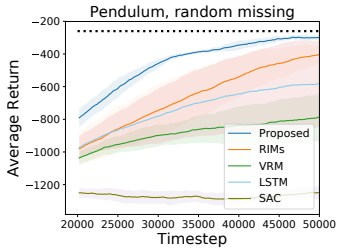
Mountain Hike

The goal of the agent in Mountain Hike is to maximize the cumulative return by moving along the path of the high reward region, as shown in Fig. 3(a). Each state is a position of the agent, but the observation is received with the addition of Gaussian noise. (See Appendix E for the details.) In Fig. 3(b), it is seen that the proposed method outperforms the baselines in the Mountain Hike environment. The horizontal black dotted line shows the mean SAC performance over five seeds at 50000 steps without noise. Hence, the performance of the proposed method nearly approaches the SAC performance in the fully observable setting.

We applied Welch's t-test at the end of the training to statistically check the proposed method's gain over the baselines. This test is robust for comparison of different RL algorithms (Colas, Sigaud, and Oudeyer 2019). Each p-value is the probability that the proposed algorithm does not outperform the compared baseline. Then the proposed algorithm outperforms the compared baseline with a $100(1-p)\%$ confidence level. The proposed method outperforms RIMs and VRM with 73 % and 98 % confidence levels, respectively.

Pendulum - Random Missing Version

We conducted experiments on the Pendulum control problem, where the pendulum is learned to swing up and stay upright during every episode. Unlike the original fully-observable

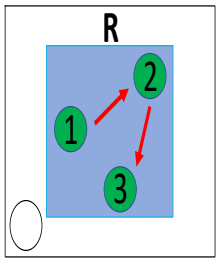


(a)

Method	p-value
Proposed	-
RIMs	0.011
VRM	0.001
LSTM	0.051
SAC	4.57×10^{-8}

(b)

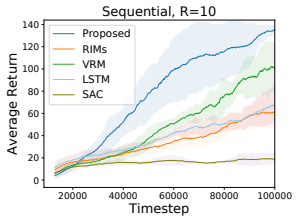
Figure 4: (a) Performance comparison in the Pendulum environment (the horizontal black dotted line shows the mean performance of SAC over five seeds at convergence when each observation is fully observable) and (b) p-values of the null hypothesis $H_0 : \mu_{\text{proposed}} \leq \mu_{\text{baseline}}$ based on Welch's t-test, where μ_{proposed} and μ_{baseline} are the performance means of the proposed method and each baseline, respectively



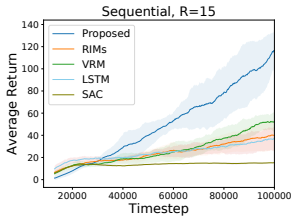
(a)

Method	Success Rate $R = 10$ (%)	Success Rate $R = 15$ (%)
Proposed	**98.4**	**91.4**
RIMs	32.4	4.2
VRM	68.8	15.6
LSTM	33.8	4.8
SAC	1.2	0.0

(b)



(c)



(d)

Figure 5: (a) Sequential target-reaching task, (b) success rate in the task with $R = 10$ and $R = 15$, respectively, and (c) and (d) performance comparison with $R = 10$ and $R = 15$, respectively

version, each dimension of every state is converted to zero with probability $p_{\text{miss}} = 0.1$ when the agent receives observation (Meng, Gorbet, and Kulic 2021). This random missing setting induces partial observability and makes a simple control problem challenging.

It is seen in Fig. 4(a) that the proposed method outperforms the baselines. The horizontal black dotted line shows the mean SAC performance at convergence when $p_{\text{miss}} = 0.0$. The performance of the proposed method nearly approaches the SAC performance in the fully observable setting, as seen in Fig. 4(a). Fig. 4(b) shows that the proposed method outperforms LSTM and RIMs with 95 % and 99 % confidence levels, respectively. Note in Fig. 4(a) that the performance variance of the proposed method (which is 12.8) is significantly smaller than that of VRM and LSTM (147.6 and 264.4, respectively). This implies that the proposed method is learned more stably than the baselines.

Sequential Target-reaching Task

To verify that the proposed model can learn long-term information effectively, we conducted experiments in the sequential target-reaching task (Han, Doya, and Tani 2020a). The sequential target-reaching task is shown in Fig. 5(a). The agent has to visit three targets in order of $1 \rightarrow 2 \rightarrow 3$ as shown in Fig. 5(a). Visiting the first target only yields r^1_{seq}, visiting the first and the second target yields r^2_{seq}, and visiting all the three target in order of $1 \rightarrow 2 \rightarrow 3$ yields r^3_{seq}, where $r^3_{\text{seq}} > r^2_{\text{seq}} > r^1_{\text{seq}} > 0$. Otherwise, the agent receives zero reward. When $R (0 < R \leq 15)$ increases, the distances among the three targets become larger, and the task becomes more challenging. The agent must memorize and properly use the past information to get the full reward.

In Figs. 5(c) and 5(d), it is seen that the proposed method significantly outperforms the baselines. Note that the performance gap between the proposed method and the baselines becomes large as the task becomes more difficult by increasing $R = 10$ to $R = 15$. Welch's t-test shows that the proposed method outperforms VRM with 98 % confidence level when $R = 10$. The p-value compared to VRM is 2.84×10^{-4} when $R = 15$.

The success rate is an alternative measure other than the average return, removing the overestimation by reward function choice in the sequential target-reaching task. After training the block model and the RL agent, we loaded the trained models, evaluated 100 episodes for each model, and checked how many times the models successfully reached all three targets in the proper order. In Fig. 5(b), it is seen that the proposed method drastically outperforms the other baselines.

Minigrid

We considered partially observable maze navigation environments with sparse reward, as seen in Figs. 6(a) and 6(c). The agent (red triangle) receives a nonzero reward $1 - 0.9 \frac{t_{\text{step}}}{N_{\text{max}}}$ only when it reaches the green square, where N_{max} is the maximum episode length and $t_{\text{step}} (\leq N_{\text{max}})$ is the total timestep before success. Otherwise, the agent receives zero reward. A new map with the same size but different shapes is generated at every episode, and the agent starts to navigate again. The agent must learn to cross the narrow path with partial observation and sparse reward. (See Appendix E for more details.)

In Figs. 6(b) and 6(d), it is seen that the proposed method outperforms the considered baselines even when the size of map increases and the difficulty becomes higher. According to Welch's t-test, the proposed method outperforms PPO with RIMs (RIMs), PPO with LSTM (LSTM), and PPO in 6(b) with 93%, 98%, and 94%, respectively. The p-value of LSTM in Fig. 6(d) is 0.159.

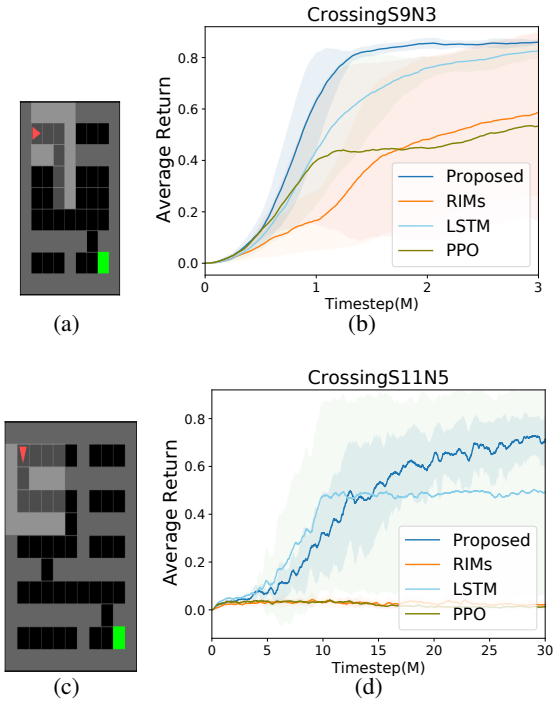


Figure 6: (a) A sample map of Minigrid CrossingS9N3 task and (b) performance comparison in CrossingS9N3. (c) A sample map of Minigrid CrossingS11N5 task and (d) performance comparison in CrossingS11N5. The y-axis represents the mean value of the returns of the most recent two hundred iterations over the five seeds

Ablation Study

The proposed block model consists of the blockwise RNN and self-attention. We investigate the contribution to the performance improvement of the blockwise RNN and the self-attention. We then replace the proposed compression method with other methods while using the same self-attention. (See Appendix F for the effect of hyperparameters L and k.)

Effect of Components

We include the method using only self-attention without blockwise RNN (denoted by 'Self-attention only'). $Y_n^k \in \mathbb{R}^{k \cdot d}$, a single vector from concatenation of k selected elements, is fed into RL agent instead of μ_n and σ_n. The self-attention is trained end-to-end with the RL agent.

We also add the method using only blockwise RNN without self-attention ('Blockwise RNN only') by replacing the self-attention with a feedforward neural network (FNN). The replaced FNN maps each d-dimensional input $x_t \in B_n$ in a block to an S_{FNN}-dimensional vector. Instead of Y_n^k, $(L \cdot S_{\text{FNN}})$-dimensional transformed block is used for the blockwise RNN input. For fair comparison, we set S_{FNN} such that $L \cdot S_{\text{FNN}}$ is equal to the dimension of $Y_n^k (= k \cdot d)$.

In Tab. 1, we observe that blockwise RNN plays an essential role for performance improvement in both sequential target-reaching task and Pendulum. In Pendulum, the effect of self-attention is critical for performance improvement.

Method	Success Rate $R = 15$ (%)	Average Return in Pendulum
Proposed	**91.4**	**-300.2**
Self-attention only	21.4	-342.9
Blockwise RNN only	90.8	-467.7
Best baseline	15.6 (VRM)	-402.7 (RIMs)

Table 1: Ablation on effect of components: Sequential target-reaching task with $R = 15$ (middle) and Pendulum (right)

Method	Average Return	p-value
Proposed	**-300.2**	-
Pooling	-354.1	0.001
Top-k Average	-352.0	0.011
Linear	-346.5	0.007
Random	-349.6	0.024

Table 2: Performance comparison with other compression methods in the considered Pendulum environment

Effect of Compression Methods

To check the effectiveness of the proposed compression method which selects Y_n^k from Y_n, we conducted performance comparison with other compression methods in the Pendulum environment. Instead of using Y_n^k as an input to the blockwise RNN in the proposed method, the considered baselines use (i) $\sum_{i=(n-1)L+1}^{nL} y_i$ (Pooling), (ii) averaging over the k selected elements in Y_n weighted by normalized corresponding contributions (Top-k Average), (iii) $[M_{\text{comp}} y_{(n-1)L+1}; \cdots ; M_{\text{comp}} y_{nL}]$ with a trainable matrix $M_{\text{comp}} \in \mathbb{R}^{\frac{kd}{L} \times d}$ (Linear), or (iv) k randomly chosen elements in Y_n (Random).

The comparison result is shown in Tab. 2. In Pendulum, self-attention has effective compression ability since all the considered compression methods using self-attention outperform the 'Blockwise RNN only' method (with average -467.7) in Tab. 1. Among the considered compression methods with self-attention, the proposed method induces the least relevant information loss.

Conclusion

In this paper, we have proposed a new blockwise sequential model learning for POMDPs. The proposed model compresses the input sample sequences using self-attention for each data block and passes the compressed information to the next block using RNN. The compressed information from the block model is fed into the RL agent with the corresponding data block to improve the RL performance in POMDPs. The proposed architecture is learned based on direct gradient estimation using self-normalized importance sampling, making the learning efficient. By exploiting the advantages of self-attention and RNN, the proposed method outperforms the previous approaches to POMDPs in the considered partially observable environments.

Acknowledgments

This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science, ICT & Future Planning (NRF-2021R1A2C2009143).

References

Ba, L. J.; Kiros, J. R.; and Hinton, G. E. 2016. Layer Normalization. *CoRR*, abs/1607.06450.

Bornschein, J.; and Bengio, Y. 2015. Reweighted Wake-Sleep. In Bengio, Y.; and LeCun, Y., eds., *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Cho, K.; van Merrienboer, B.; Gülçehre, Ç.; Bahdanau, D.; Bougares, F.; Schwenk, H.; and Bengio, Y. 2014. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. In Moschitti, A.; Pang, B.; and Daelemans, W., eds., *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, 1724–1734. ACL.

Chung, J.; Kastner, K.; Dinh, L.; Goel, K.; Courville, A. C.; and Bengio, Y. 2015. A Recurrent Latent Variable Model for Sequential Data. In Cortes, C.; Lawrence, N. D.; Lee, D. D.; Sugiyama, M.; and Garnett, R., eds., *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, 2980–2988.

Colas, C.; Sigaud, O.; and Oudeyer, P. 2019. A Hitchhiker's Guide to Statistical Comparisons of Reinforcement Learning Algorithms. In *Reproducibility in Machine Learning, ICLR 2019 Workshop, New Orleans, Louisiana, United States, May 6, 2019*. OpenReview.net.

Goyal, A.; Lamb, A.; Hoffmann, J.; Sodhani, S.; Levine, S.; Bengio, Y.; and Schölkopf, B. 2021. Recurrent Independent Mechanisms. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net.

Haarnoja, T.; Zhou, A.; Abbeel, P.; and Levine, S. 2018. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. In Dy, J. G.; and Krause, A., eds., *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, 1856–1865. PMLR.

Han, D.; Doya, K.; and Tani, J. 2020a. Self-organization of action hierarchy and compositionality by reinforcement learning with recurrent neural networks. *Neural Networks*, 129: 149–162.

Han, D.; Doya, K.; and Tani, J. 2020b. Variational Recurrent Models for Solving Partially Observable Control Tasks. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.

Hausknecht, M. J.; and Stone, P. 2015. Deep Recurrent Q-Learning for Partially Observable MDPs. In *2015 AAAI Fall Symposia, Arlington, Virginia, USA, November 12-14, 2015*, 29–37. AAAI Press.

Hochreiter, S.; and Schmidhuber, J. 1997. Long Short-Term Memory. *Neural Comput.*, 9(8): 1735–1780.

Igl, M.; Zintgraf, L.; Le, T. A.; Wood, F.; and Whiteson, S. 2018. Deep Variational Reinforcement Learning for POMDPs. In Dy, J.; and Krause, A., eds., *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, 2117–2126. PMLR.

Le, T. A.; Igl, M.; Rainforth, T.; Jin, T.; and Wood, F. 2018. Auto-Encoding Sequential Monte Carlo. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net.

Le, T. A.; Kosiorek, A. R.; Siddharth, N.; Teh, Y. W.; and Wood, F. 2019. Revisiting Reweighted Wake-Sleep for Models with Stochastic Control Flow. In Globerson, A.; and Silva, R., eds., *Proceedings of the Thirty-Fifth Conference on Uncertainty in Artificial Intelligence, UAI 2019, Tel Aviv, Israel, July 22-25, 2019*, volume 115 of *Proceedings of Machine Learning Research*, 1039–1049. AUAI Press.

Maddison, C. J.; Lawson, D.; Tucker, G.; Heess, N.; Norouzi, M.; Mnih, A.; Doucet, A.; and Teh, Y. W. 2017. Filtering Variational Objectives. In Guyon, I.; von Luxburg, U.; Bengio, S.; Wallach, H. M.; Fergus, R.; Vishwanathan, S. V. N.; and Garnett, R., eds., *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, 6573–6583.

Meng, L.; Gorbet, R.; and Kulic, D. 2021. Memory-based Deep Reinforcement Learning for POMDP. *CoRR*, abs/2102.12344.

Naesseth, C. A.; Linderman, S. W.; Ranganath, R.; and Blei, D. M. 2018. Variational Sequential Monte Carlo. In Storkey, A. J.; and Pérez-Cruz, F., eds., *International Conference on Artificial Intelligence and Statistics, AISTATS 2018, 9-11 April 2018, Playa Blanca, Lanzarote, Canary Islands, Spain*, volume 84 of *Proceedings of Machine Learning Research*, 968–977. PMLR.

Richter, O.; and Wattenhofer, R. 2020. Normalized Attention Without Probability Cage. *CoRR*, abs/2005.09561.

Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; and Klimov, O. 2017. Proximal Policy Optimization Algorithms. *CoRR*, abs/1707.06347.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, L.; and Polosukhin, I. 2017. Attention is All you Need. In Guyon, I.; von Luxburg, U.; Bengio, S.; Wallach, H. M.; Fergus, R.; Vishwanathan, S. V. N.; and Garnett, R., eds., *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, 5998–6008.

Zhu, P.; Li, X.; and Poupart, P. 2017. On Improving Deep Reinforcement Learning for POMDPs. *CoRR*, abs/1704.07978.