Particle Filter Network

Pengfei Gao, Junzi Zhang

03/14/2018

Motivation and Main Problem

- Typical POMDP algorithms are model-based or domain specific
- What if we only have a black-box simulator for the observations **(and not the states)**? Then usual MCTS-type methods (e.g. POMCP) don't work!
- Can we have a model-free approach for POMDP? I.e., can we have a general POMDP solver to make decision given a black-box simulator only for the observations, which is the usual practical case?
- Or consider a simpler problem, how can we even understand an unknown hidden Markov decision process given this its black-box simulator?

Prior Work

- Decision Making:
  - For MDP, we have model-free MDP: Q-learning, Policy Gradient, etc.
  - For POMDP, we can learn with external-memory; or we can build a framework with strong prior knowledge like in QMDP-net, so as to learn unknown underlying process from observations
- Understanding Unknown HMM:
  - Recurrent Neural Network or other deep neural-network structure seems to be the only methods that can learn from black-box hidden Markov model
  - Drawback: RNN can only give conditional observation forecast at next time step! No understanding of the hidden state or its **unconditional propagation (without the inputs/observations)**
- We will focus on how to learn and interpret hidden state for an unknown HMM.

Underlying Model Assumption

True Model Definition

Hidden states start from some prior distribution:

$$s_0 \sim p(\cdot)$$

Hidden states transit as in the dynamics:

$$s_t \sim p(\cdot \mid s_{t-1})$$

Observations are generated from:

$$o_t \sim o(\cdot \mid s_t)$$

Particle Propagation

Transit to next state:

Transition Network

$$\tilde{s}_{t,i} \sim p(\cdot \mid s_{t-1,i}) \qquad \tilde{s}_{t,i} := T_\theta(z, s_{t-1,i}) \quad z \sim N(0,1)$$

Reweight particle:

Reweight Network

$$w_{t,i} = p(o_t \mid s_{t,i}) \qquad W_{t,i} := O_\theta(\tilde{s}_{t,i}, o_t) \quad w_{t,i} := \frac{e^{W_{t,i}}}{\sum_{j=1}^{M} e^{W_{t,j}}}$$

Bootstrap:

$$\text{Boostrap resample } \{s_{t,i}\} \text{ from } \{\tilde{s}_{t,i}, w_{t,i}\}$$

Observation Forecast

$$\hat{o}_t = F^o_\theta(\tilde{s}_t)$$

Estimation Network

Residual Definition

- k-step pre-fit residual: $y_{t|t-k} = o_t - \mathbb{E}\left[O_t|o_{1:t-k}\right] = o_t - \mathbb{E}\left[\mathbb{E}\left[O_t|s_t\right]|o_{1:t-k}\right]$
- post-fit residual: $y_{t|t} = o_t - \mathbb{E}\left[O_t|o_{1:t}\right] = o_t - \mathbb{E}\left[\mathbb{E}\left[O_t|s_t\right]|o_{1:t}\right]$

Residual from PF Network

The K-step pre-fit residual from PF network is:

$$y_{t|t-K} = o_t - \frac{1}{M}\sum_{i=1}^{M} F^o_\theta(\hat{s}_{t,i})$$

where $\hat{s}_{t,i} = T_\theta(\cdot, \hat{s}_{t-1,i})$ recursively for K times, until $\hat{s}_{t-K+1,i} = T_\theta(\cdot, s_{t-K,i})$

The post-fit residual from PF network is:

$$y_{t|t} = o_t - \sum_{i=1}^{M} w_{t,i} F^o_\theta(\tilde{s}_{t,i})$$

Loss Function

Related to reweight network

Reweight Network

$$W_{t,i} := O_\theta(\tilde{s}_{t,i}, o_t)$$

Related to transition network

Transition Network

$$\tilde{s}_{t,i} := T_\theta(z, s_{t-1,i})$$

$$\frac{1}{T}\sum_{t=1}^{T}\left(y_{t|t}^2 + \sum_{k=0}^{K-1}\frac{\lambda}{(k+1)^2}y_{t-k|t-K}^2\right)$$

Both Related to estimation network

$$\hat{o}_t = F_\theta^o(\tilde{s}_t)$$

Estimation Network

Training Process

Back Propagate The following is how to train PF network from a simulator:

1. Generate $o_{0:T}$ from simulator.
2. Generate $s_{0,i} \sim$ prior, use forward propagate to get $\{s_{t,i}\}$ for $t = 1, \ldots, T; i = 1, \ldots, M$.
3. The following steps are repeated $\frac{T}{L}$ times in tensorflow:
   (a) Random pick time index $\tau_1, \ldots, \tau_L$ from $t = 1, \ldots, T$.
   (b) Apply one gradient descend by minimizing the following loss in tensorflow

$$\text{Loss}(\theta; \{s_{\tau_j,i}\}) := \sum_{j=1}^{L} \left(y^2_{\tau_j|\tau_j} + \sum_{k=0}^{K-1} \frac{\lambda}{(k+1)^2} y^2_{\tau_j-k|\tau_j-K} \right)$$

   Specifically, for each j compute independently,

   i. Compute $\{\tilde{s}_{\tau_j,i}, w_{\tau_j,i}\}_{i=1}^{M}$ via forward propagate from $\tau_j - 1$ to τ_j, given $\{s_{\tau_j-1,i}\}_{i=1}^{M}$ from step 2
   ii. $y_{\tau_j|\tau_j} := o_{\tau_j} - \sum_{i=1}^{M} w_{\tau_j,i} F^o_\theta(\tilde{s}_{\tau_j,i})$
   iii. For all $i = 1, \ldots, M$, compute independently: $\hat{s}_{\tau_j,i} := T_\theta(\cdot, \hat{s}_{\tau_j-1,i})$ recursively for K times, until $\hat{s}_{\tau_j-K+1,i} := T_\theta(\cdot, s_{\tau_j-K,i})$, given $\{s_{\tau_j-K,i}\}_{i=1}^{M}$ from step 2
   iv. For all $k = 0, \ldots, K-1$, $y_{\tau_j-k|\tau_j-K} := o_{\tau_j-k} - \frac{1}{M} \sum_{i=1}^{M} F^o_\theta(\hat{s}_{\tau_j-k,i})$

4. goto step 1.

Experiment 1

3.2 Experiment 1: Linear Gaussian Model

True model:

$$s_t = F s_{t-1} + c_1 + \epsilon_{1,t}$$

$$o_t = H s_t + c_2 + \epsilon_{2,t}$$

where $s_t, c_1, \epsilon_{1,t}$ are dimension d_s vectors; $o_t, c_2, \epsilon_{2,t}$ are dimension d_o vectors. $\epsilon_{1,t} \sim N(0, Q)$, $\epsilon_{2,t} \sim N(0, R)$. And for initial state, we assume $s_0 \sim N(c_1, Q)$.

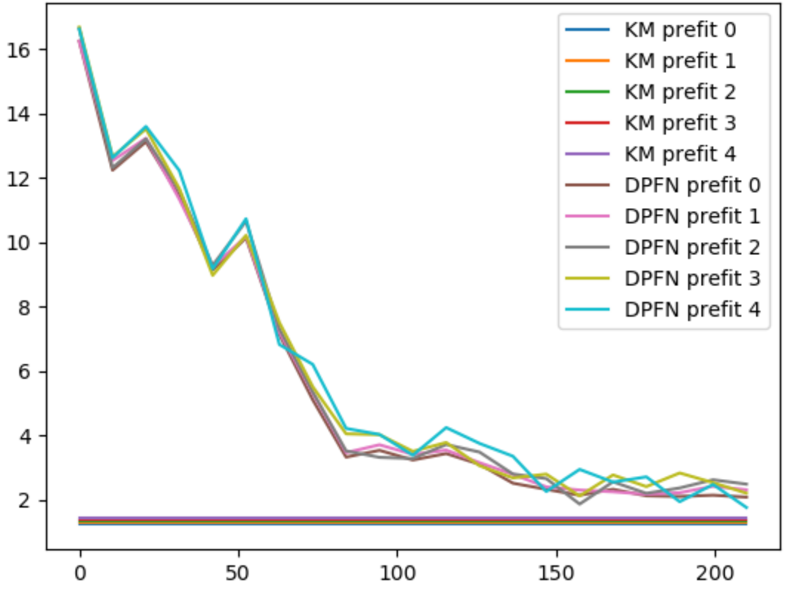
PF Network:

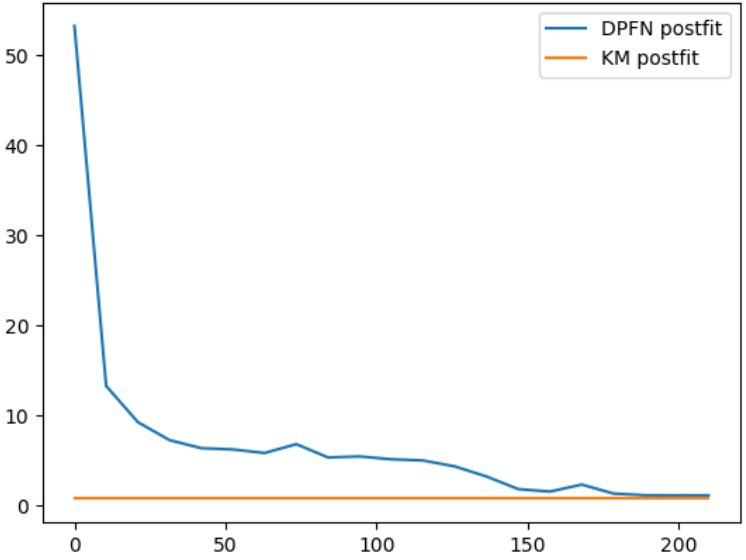
$$\tilde{s}_t = T_\theta(z, s_{t-1}) = W_1[s_{t-1}, z] + b_1$$

$$\hat{o}_t = F_\theta^o(\tilde{s}_t) = W_2 \tilde{s}_t + b_2$$

$$W_t = O_\theta(\tilde{s}_t, o_t) = \sigma(W_4 \sigma(W_3[\tilde{s}_t, o_t] + b_3) + b_4)$$

Experiment 1 Results





Experiment 2

3.3 Experiment 2: Classic Non-Linear Model

True model:

$$s_t = \frac{s_{t-1}}{2} + 25\frac{s_{t-1}}{1+s_{t-1}^2} + 8\cos(1.2t) + \epsilon_{1,t}$$

$$o_t = \frac{s_t^2}{2} + \epsilon_{2,t}$$

where $s_t, \epsilon_{1,t}, o_t, \epsilon_{2,t}$ are all scalers. $s_0 = 0$, $\epsilon_{1,t} \sim N(0, 0.1^2)$, $\epsilon_{2,t} \sim N(0,1)$

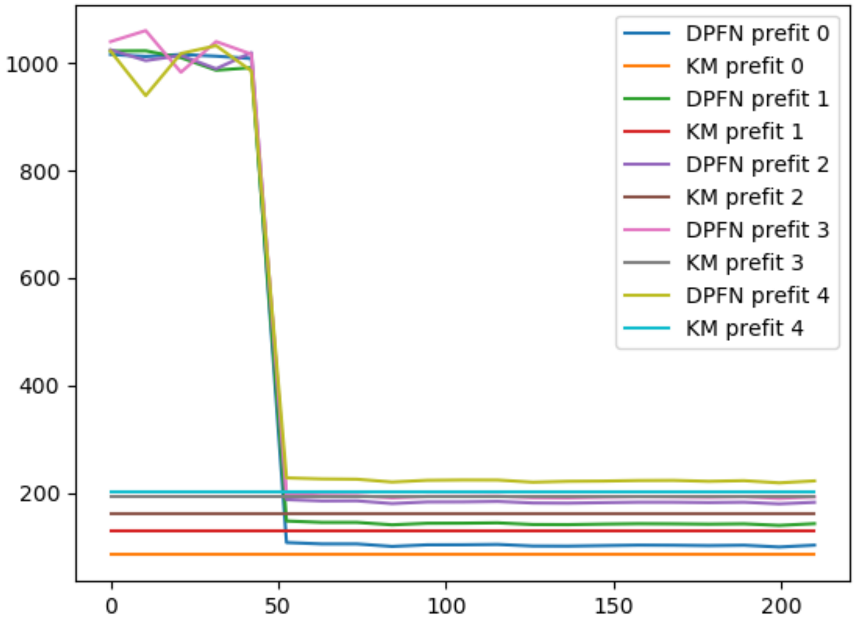
PF Network:

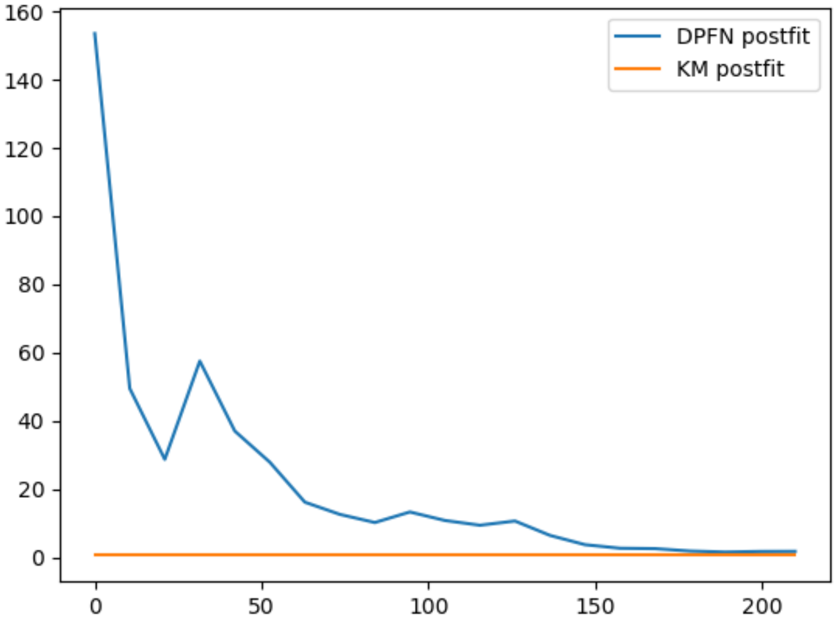
$$\tilde{s}_t = T_\theta(z, s_{t-1}) = \text{fully connected two layer}$$

$$\hat{o}_t = F_\theta^o(\tilde{s}_t) = \text{fully connected two layer}$$

$$W_t = O_\theta(\tilde{s}_t, o_t) = \text{fully connected two layer}$$

Experiment 2 Results





Discussion of results

- Our preliminary results show the training works.
- Delicate tuning of hyper-parameters is needed to increase training accuracies.

Future Work

1. Replace current transition/reweight network to posterior sampling to increase particle efficiencies
2. Test on more complex models
3. Include (historical) actions in the trajectories, and add QMDP-module on learned hidden states to have a general model-free POMDP methods

Closely Related Papers

References

[1] Peshkin, Leonid, Nicolas Meuleau, and Leslie Kaelbling. "Learning policies with external memory." arXiv preprint cs/0103003 (2001).

[2] Mnih, Volodymyr, et al. "Playing atari with deep reinforcement learning." arXiv preprint arXiv:1312.5602 (2013).

[3] Tamar, Aviv, et al. "Value iteration networks." Advances in Neural Information Processing Systems. 2016.

[4] Zhang, Marvin, et al. "Learning deep neural network policies with continuous memory states." Robotics and Automation (ICRA), 2016 IEEE International Conference on. IEEE, 2016.

[5] Karkus, Peter, David Hsu, and Wee Sun Lee. "QMDP-Net: Deep Learning for Planning under Partial Observability." arXiv preprint arXiv:1703.06692 (2017).

[6] Osband, Ian, et al. "Deep exploration via randomized value functions." arXiv preprint arXiv:1703.07608 (2017).