Towards Stochastic Recurrent Neural Control Policy

Tianhao Zhang

Sequential Monte Carlo variant of Generalized EM algorithm

Training Objective: variational lower bound of data log-likelihood

$$l(\theta) = \log p_\theta(u|x) = \log \sum_h p_\theta(u,h|x) = \log \mathbb{E}_{h\sim q} \frac{p_\theta(u,h|x)}{q(h)}$$

$$\geq \mathbb{E}_{h\sim q} \log \frac{p_\theta(u,h|x)}{q(h)} = Q(\theta) - \mathcal{H}(q) = \mathcal{L}(q,\theta) \qquad (3)$$

Approximate Distribution: the tighest choice is $q^*(h) = p(h|u,x)$

$$\mathcal{L}(q,\theta) = \mathbb{E} \log \frac{p(h|u,x)p_\theta(u|x)}{q(h)}$$

$$= \log p_\theta(u|x) - D_{KL}\big(q(h) \parallel p_\theta(h|u,x)\big) \qquad (4)$$

EM Algorithm:

$$\begin{cases} \text{E-step:} & q^* = \max_q \mathcal{L} = p(h|u,x) \qquad (5a) \\ \text{M-step:} & \theta^* = \max_\theta \mathcal{L} = \max_\theta Q(\theta) \qquad (5b) \end{cases}$$

Sequential Monte Carlo (E-step): importance sampling is needed because the true posterior distribution is hard to compute

$$w(\mathbf{h}^{(i)}_{0:T}) = \frac{p(\mathbf{h}^{(i)}_{0:T}|\mathbf{u}_{0:T},\mathbf{x}_{0:T})}{p(\mathbf{h}^{(i)}_{0:T}|\mathbf{x}_{0:T})}$$

$$= \frac{p(h^{(i)}_0|x_0,u_0)}{p(h^{(i)}_0|x_0)} \prod_{t=1}^{T} \frac{p(h^{(i)}_t|\mathbf{h}^{(i)}_{0:t-1},\mathbf{u}_{0:t},\mathbf{x}_{0:t})}{p(h^{(i)}_t|\mathbf{h}^{(i)}_{0:t-1},\mathbf{x}_{0:t})} \qquad (6)$$

$$w^i_T \triangleq \zeta^i_0 \prod_{t=1}^{T} \zeta^i_t = w^i_{T-1}\zeta^i_T \qquad (7)$$

The last equation suggests that the importance weights can be computed recursively by:

$$\zeta^i_t = \frac{p(h^{(i)}_t|\mathbf{h}^{(i)}_{0:t-1},\mathbf{u}_{0:t},\mathbf{x}_{0:t})}{p(h^{(i)}_t|\mathbf{h}^{(i)}_{0:t-1},\mathbf{x}_{0:t})}$$

$$= \frac{p(\mathbf{h}^{(i)}_{0:t},\mathbf{u}_{0:t},\mathbf{x}_{0:t})}{p(\mathbf{h}^{(i)}_{0:t},\mathbf{x}_{0:t})p(\mathbf{u}_{0:t}|\mathbf{h}^{(i)}_{0:t-1},\mathbf{x}_{0:t})} = \frac{p(\mathbf{u}_{0:t}|\mathbf{h}^{(i)}_{0:t},\mathbf{x}_{0:t})}{p(\mathbf{u}_{0:t}|\mathbf{h}^{(i)}_{0:t-1},\mathbf{x}_{0:t})}$$

$$= \frac{p(u_0|h_0,x_0)p(u_t|\mathbf{h}^{(i)}_{0:t},\mathbf{x}_{0:t})\prod_{\tau=1}^{t-1}p(u_\tau|\mathbf{h}_{0:\tau},\mathbf{x}_{0:\tau})}{p(u_0|h_0,x_0)p(u_t|\mathbf{h}_{0:t-1},\mathbf{x}_{0:t})\prod_{\tau=1}^{t-1}p(u_\tau|\mathbf{h}_{0:\tau},\mathbf{x}_{0:\tau})}$$

$$= \frac{p(u_t|\mathbf{h}^{(i)}_{0:t},\mathbf{x}_{0:t})}{p(u_t|\mathbf{h}_{0:t-1},\mathbf{x}_{0:t})} \simeq \frac{p(u_t|\mathbf{h}^{(i)}_{0:t},\mathbf{x}_{0:t})}{\frac{1}{M}\sum_j p(u_t|\mathbf{h}^{(j)}_{0:t},\mathbf{x}_{0:t})} \qquad (8)$$

where M is the total number of samples from q_t for each t and the last equation follows from a standard Monte Carlo:

$$p(u_t|\mathbf{h}_{0:t-1},\mathbf{x}_{0:t}) = \mathbb{E}_{h_t\sim q_t} p(u_t|h_t,\mathbf{h}_{0:t-1},\mathbf{x}_{0:t})$$

By now we should have collected:

$$q^i_t \triangleq p(h^{(i)}_t|\mathbf{h}_{0:t-1},\mathbf{x}_{0:t}) \qquad l^i_t \triangleq p(u_t|\mathbf{h}^{(i)}_{0:t},\mathbf{x}_{0:t}) \qquad (9)$$

Gradient Asent (M-step):

$$Q(\theta) = \mathbb{E}_{\mathbf{h}_{0:T}\sim q^*} \log p_\theta(\mathbf{u}_{0:T},\mathbf{h}_{0:T}|\mathbf{x}_{0:T})$$

$$\simeq \sum_{i=1}^{M} \tilde{w}^i_T \big[\log p_\theta(\mathbf{u}_{0:T}|\mathbf{h}^{(i)}_{0:T},\mathbf{x}_{0:T}) + \log p_\theta(\mathbf{h}^{(i)}_{0:T}|\mathbf{x}_{0:T})\big]$$

$$= \sum_{i=1}^{M} \Big(\prod_{t=0}^{T} \frac{q^i_t}{\sum_j q^j_t}\Big)\Big[\sum_{t=0}^{T} \log q^i_t + \log l^i_t\Big]$$

$$\frac{\partial Q}{\partial \theta} \simeq \sum_{i=1}^{M} \Big(\prod_{t=0}^{T} \frac{q^i_t}{\sum_j q^j_t}\Big) \frac{\partial}{\partial \theta}\Big[\sum_{t=0}^{T} \log q^i_t + \log l^i_t\Big] \qquad (10)$$

Abstract

Goal: MLPs are popular control policies for, and achieve state-of-the-art results in, many robotics tasks. However, the feedforward architecture of MLPs limits their ability to extend further than a sophisticated reflexive agent. We introduce stochastic units into the neural network control policies, with an ultimate goal of achieving a stochastic recurrent neural network.

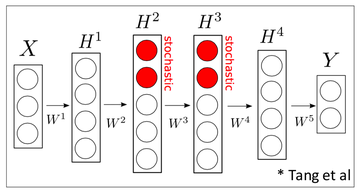
Method: We derive a Sequential Monte Carlo variant of Generalized EM algorithm for learning stochastic recurrent neural policy.

Graphical Models

Hybrid Stochastic Feed-forward Network (SFNN): Shares same structure with MLPs, except that some hidden units are replaced by Bernoulli stochastic neurons. In the extreme case, this becomes a Sigmoid Belief Network.

Bernoulli units: $p(h_{out} = 1 \mid h_{in}) = \sigma(h_{in})$

Recurrent Stocastic units: We can readily add additional dependency passed through time.



* Tang et al

The below algorithm summarizes the overall learning procedure of a hybrid stochastic recurrent neural control policy:

Algorithm 1 SIS Generalized EM

Input: training trajectories $\{(\mathbf{x}^{(n)}_{0:T}, \mathbf{u}^{(n)}_{0:T}, \mathbf{\Sigma}^{(n)}_{\mathbf{u}0:T})\}_{n=1}^{N}$, hidden stochastic units for each step unrolled $\{\mathbf{h}_t\}_{t=1}^{T}$, network parameter θ, number of samples M

repeat

{E-step}

for $n = 1$ **to** N **do**

for $i = 1$ **to** M **do**

$h^{(i)}_1 \sim p(h_1|h_0,x_1)$

$l^i_1 = p(h^{(i)}_1|h_0,x_1)$

$w^i_1 = p(u_1|\mathbf{h}^{(i)}_{0:1},x_1)$

end for

$\tilde{w}^i_1 = w^i_1 / \sum_i w^i_1$

for $t = 2$ **to** T **do**

for $i = 1$ **to** M **do**

$h^{(i)}_t \sim p(\mathbf{h}_{0:t}|\mathbf{h}_{0:t-1},\mathbf{x}_{0:t})$

$l^i_t = p(\mathbf{h}^{(i)}_{0:t}|\mathbf{h}_{0:t-1},\mathbf{x}_{0:t})$

$w^i_t = p(u_t|\mathbf{h}^{(i)}_{0:t},\mathbf{x}_{0:t})$

end for

$\tilde{w}^i_t = w^i_t / \sum_i w^i_t$

end for

end for

{M-step}

for $i = 1$ **to** M **do**

compute $\Delta Q^{(i)} \triangleq \frac{\partial}{\partial\theta}\big[\sum_{t=0}^{T} \log q^i_t + \log l^i_t\big]$ using standard back-propagation

end for

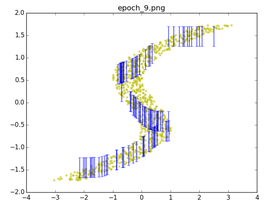
sga_update($\theta, \sum_{i=1}^{M} \tilde{w}^i_T \Delta Q^{(i)}$) {any stochastic gradient ascent algorithm such as ADAM or RMSProp}

until convergence

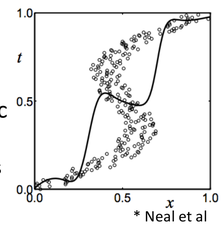
Experimental Evaluation

Implementation: We independently implemented harness3, based on Computation Graph Toolkit, that allows convenient construction of arbitrary structure of neural network. A highlight of our harness is that it supports layer with mixed types of units, including standard fully-connected units, Bernoulli stochastic units, long-short-term-memory block, and com- mon non-linearities.

Synthetic Datasets: Here we review the capabilities of SFNN on 1D dataset



Right: optimal solution (conditional average) of a deterministic network
Left: error bars represent the range of samples from the learnt stochastic network
* Horizontal/vertical axis is inputs/outputs

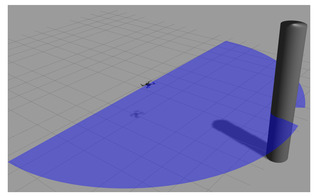


* Neal et al

Quadrotor Control Task: cylindrical obstacle avoidance

Data Generation:

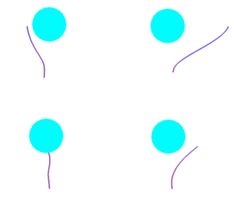
- Total of 15 randomized initial positions roughly in front of the obstacle
- For each scenario, a trajectory optimization algorithm (iLQG) is used to generate optimal (time-variant Gaussian) control policy given perfect state information (i.e. position, etc)
- Rollout each optimal policy 5 times in simulation and collect the raw observations (i.e. laser ranges) along the resulting trajectories as our supervised training set



* a 3D view of the simulator

* typical training trajectories

Set-up: we trained MLP and SFNN on same training set with conflicting examples which often arise in typical data collection process.
For SFNN, use M = 30 samples for gradient estimate. For both, apply weight decay (5e-3) and allow objective to plateau with proper number of epochs.
Both have fully-connected layers of size 40 (inputs) -> 40 -> 20 -> 15 -> 4 (outputs); SFNN has additional one Bernoulli node at each of the last two layers.



Result: MLP control policy is permanently biased towards dodging right and leads to frequent crashes, whereas SNN retains the variety and keeps a lower crash rate.

Above: SNN; **Below**: MLP
Typical rollouts of learnt policy

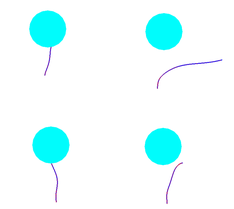


Table 1. Comparison of stochastic and deterministic neural network control policies (numbers in parenthesis represent training epochs

METHOD	%CRASH	%LEFT
SNN(5)	32.0	22.9
SNN(10)	49.3	15.2
SNN(15)	33.3	11.3
MLP(15)	48.0	0