Towards Stochastic Neural Network Control Policies

Tianhao Zhang <OMITTED>@BERKELEY.EDU
University of California, Berkeley, Department of Electrical Engineering and Computer Sciences

Abstract

Deterministic neural networks (DNNs) are powerful models in many artificial intelligence tasks. In robotics, DNNs prove expressive and flexible control policies and often show good generalizability in novel scenarios. However, the deterministic nature of computations in DNNs effectively restricts DNNs to only modelling unimodal control policies. Training DNNs on multimodal data can cause inefficiencies or even lead to garbage results. In contrast, stochastic neural networks (SNNs) are able to learn one-to-many mappings. In this paper, we introduce SNNs as control policies and extend existing learning algorithm for feed-forward SNNs to recurrent ones. Our experiments on the task of quadrotor obstacle avoidance demonstrate that the use of SNNs instead of DNNs are crucial when multiple competing controls exist. We also sketch the future direction towards using recurrent SNNs.

1. Introduction

Deterministic neural networks are powerful and universal models and prove applicable in many domains in artificial intelligence, such as computer vision and natural language processing. In robotics control, recent works show that using deep neural networks as control policies lead to promising results. The learnt policy demonstrates good generalizability in novel test scenarios, proves capable of low-level controls, and, under instrumented training environment, is able to learn to directly operate on raw observations (Levine et al., 2015). The success is much owing to the expressiveness and flexibility of neural networks.

However, the deterministic nature of computations, including forward pass and gradient back-propagation, in standard neural networks practically restricts DNNs to modelling only uni-modal control policies. Given fixed inputs, like a reflex agent with one-to-one condition-action rules set, the standard neural network policy always outputs the same values. This determinism is not always sufficient when the optimal policy is multi-modal and conflicting, or multi-modal, examples exist in the training set. For example, in the context of obstacle avoidance, a robot, initially positioned straight in front of a tall symmetrical obstacle, faces two equally optimal control policies. During the supervised training, the neural network may receive examples with competing controls at identical states. A deterministic neural policy, without explicitly incorporating this binary information into the state, either leads to inefficient training, when the left- and right-trajectory are not equal in number, resulting in a biased policy, or, more fatally, produces pathological policies, such as crashing straight into the obstacle, as it learns to average trajectories of both turning directions. If explicitly separating different modes in training examples is difficult, any deterministic structure fails to model the desired underlying policy. This is only made worse in partially observable environments, where the optimal policy is locally ambiguous. As in the above example, the robot, now equipped with collision detection of limited range, is unable to locally determine the optimal control to circumvent an obstacle that extends beyond this range.

To our knowledge, prior works have relied solely on deterministic structures for control policies. We propose to use a stochastic neural network instead when the optimal policy is multi-modal. In such networks, some, or all, units are stochastic. One popular choice is Bernoulli random variable whose parameter is supplied by the outputs of its previous layer, as in Sigmoid Belief Network.

$$h \sim \text{Bernoulli}(\sigma(x)), \quad \text{or} \quad p(h = 1|x) = \sigma(x)$$

Then multi-modal outputs are expected from multiple runs of the forward pass with the same input. Tang et al (2013) demonstrated that hybrid networks, with a mix of deterministic units and a relatively few few number of stochastic units, achieve superior performance than other appoaches. Considering faster training process with fewer stochastic units and the exponentially many modes a given number of Bernoulli units can represent, we adopt similar architecture for our SNN policies. Indeed, the mix of deterministic and stochastic units should lend the network more power

This is a report of an ongoing research effort. Also as final projects of CompSci 287 and 281A, Fall 2015.

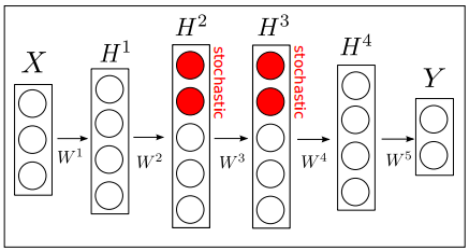

Figure 1. An example of hybrid feed-forward network where the red units are Bernoulli random variables as described above.[2]

than a purely stochastic or deterministic one. For control task, the deterministic units, owing to their computational delicacy, are responsible for producing meaningful control outputs, whereas the stochastic ones provide higher level signals to account for the many modes, if any, of the training data. Throughout this paper, we refer hybrid stochastic network as SNN for clarity. Further, we anticipate that a feed-forward SNNs may lack the ability to consistently commit to a decision through time. In the obstacle avoidance example, having moved left at the previous time step, the robot may decide, unawares of its previous decision, move right at the current step and may hence oscillate indefinitely. Without experimentation, we speculate that using *recurrent* stochastic units may address this issue. We envision that the resulting recurrent SNN policies can successfully apply to different robotics control tasks where ambiguity prevails and/or variety of controls is beneficial.

Towards that end, in this paper, we derive a learning algorithm for recurrent SNNs using a Sequential Monte Carlo variant of Generalized EM algorithm, which extends the algorithm proposed in Tang et al (2013) for feed-forward SNNs. For experimental evaluation, we compare the performance of feed-forward SNNs, as the base case of recurrent SNNs, and DNNs in the context of quadrotor obstacle avoidance, and demonstrate that even without the recurrent part SNNs critically outperform DNNs.

2. Preliminaries

2.1. Guided Policy Search

We follow a simplified version of guided policy search (GPS) algorithm for learning control policies (Levine & Abbeel, 2014). Guided policy search is an efficient reinforcement learning framework that decomposes general policy search problem, which is often intractable, into two much easier sub-problems: trajectory optimization and supervised training of the global control policy. During the trajectory optimization phrase, given prescribed initial conditions of environment and robot, an efficient algorithm is used to produce local optimal policy for each condition. A common choice is differential dynamic programming (DDP) algorithm which produces time-varying linear-Gaussian policies (Tassa et al., 2012). In practice, each local optimal policy is allowed to roll out multiple times to produce trajectories, which are used as training set in the supervised training phrase. The GPS proceeds in a iterative manner, where the successive phrase of trajectory optimization considers the partially learnt global policy at its current state so as to generate training trajectories that match the latter. This is enforced by a set of constraints on KL divergence between the global policy and local optimal policies. In this paper, we allow the GPS to run for only one iteration for simplicity as it suffices to illustrate our proposed control policy.

One may wonder why we want to train a neural network policy since we can acquire optimal policies for arbitrary initial conditions via algorithms such as DDP. First, these policies are only locally optimal along the resulting trajectories and have no means of generalization. We cannot expect these policies to work in novel scenarios. Instead, neural networks represent a class of highly expressive, nonlinear models. Trained with representative trajectories from a variety of conditions, the neural network often shows good generalizability. Second, the optimization algorithm is often not able to directly optimize upon high-dimensional raw observation inputs such as images, whereas neural networks, as shown in many computer vision works, perform well for such low-level inputs.

2.2. Neural Network Policy

Here we introduce the parametrization of our resulting control policy. The notations closely follow Zhang et al (Zhang et al., 2015). Let $\pi^*(\mathbf{u}|\mathbf{x})$ represent the true global optimal policy we wish the neural network learns. We represent the neural network policy as $\pi_\theta(\mathbf{u}|\mathbf{x})$ where θ is the parameter to learn. A reasonable choice for π_θ is a conditional Gaussian distribution where the network realizes its mean, i.e.

$$\pi_\theta(\mathbf{u}|\mathbf{x}) = \mathcal{N}\big(\text{net}_\theta(\cdot|\mathbf{x}), \Sigma^\pi(\mathbf{x})\big)$$

Since both the local optimal policies and the global policy are Gaussians, we can analytically calculate $\Sigma^\pi(\mathbf{x})$ in close form for best performance. For notational convenience, we drop the variance term and assume the network directly outputs the distribution $\pi_\theta(\mathbf{u}|\mathbf{x})$.

3. Stochastic Control Policies

3.1. Parametrization

Recent works have shown that stochastic networks can efficiently model one-to-many mappings, and recurrent networks are known able to carry over useful information

[2]The figure originally appeared in Tang et al (2013)

through time. We propose to introduce stochastic units into neural network control policies to effectively render the resulting policies multi-modal. Concretely, let $\mathbf{h}$ denote all the hidden stochastic units and $\mathbf{h}^{(i)}$ one of the possible realizations. The proposed stochastic network then has potentially distinctive mode for each hidden realization, i.e.

$$\pi_\theta(\mathbf{u}|\mathbf{x}) = \mathcal{N}\big(\{\text{net}_\theta(\cdot|\mathbf{x}, \mathbf{h}^{(i)}) : \forall i\}, \Sigma^\pi(\mathbf{x})\big) \tag{1}$$

With sigmoid nonlinearity, we can easily sample $\mathbf{h}$ given $\mathbf{x}$. Concretely, for a hybrid layer $\mathbf{h}^l = \begin{pmatrix}\mathbf{h}_s^l & \mathbf{h}_d^l\end{pmatrix}$ where s stands for stochastic units and d deterministic ones, we have

$$\begin{aligned}\mathbf{h}^l &= \sigma(W^{l-1}\mathbf{h}^{l-1} + b^{l-1}) \\ \mathbf{h}_s^l &= \text{Bernoulli}(\mathbf{h}_s^l)\end{aligned}$$

Towards recurrent SNNs, we can additionally introduce time dependencies; a potential choice is

$$\mathbf{h}_t^l = \sigma(W^{l-1}\mathbf{h}_t^{l-1} + b_t^{l-1} + H^l\mathbf{h}_{t-1}^l)$$

3.2. Learning Objective

Given a set of training trajectories $\{(\mathbf{x}_{0:T}^{(n)}, \mathbf{u}_{0:T}^{(n)})\}_{n=1}^N$, we wish to fit a recurrent SNN $p_\theta(u_t|\mathbf{x}_{0:t})$ to model the underlying global optimal policy $\pi^*(u_t|\mathbf{x}_{0:t})$. Ideally, we would like to directly maximize the data log-likelihood $l(\theta)$; however, the summand inside the logarithm is generally intractable. Instead, we follow the variational inference approach (Jordan et al., 1999) and maximize its lower bound $\mathcal{L}$ (below we drop the subscript $(\cdot)_{0:T}$ for clarity):

$$\begin{aligned}l(\theta) = \log p_\theta(u|x) &= \log \sum_h p_\theta(u, h|x) = \log \mathbb{E}_{h\sim q}\frac{p_\theta(u,h|x)}{q(h)} \\ &\geq \mathbb{E}_{h\sim q}\log\frac{p_\theta(u,h|x)}{q(h)} = Q(\theta) - \mathcal{H}(q) = \mathcal{L}(q,\theta)\end{aligned} \tag{3}$$

where $\mathcal{H}$ is the cross entropy and Q is the expected complete data log-likelihood.

Although the variational lower bound holds for any $q(h)$, in practice, without a careful choice the learning can hardly proceed. To see the choice of $q^*(h) = p(h|u,x)$ leads to the tightest bound, rewrite the lower bound as:

$$\begin{aligned}\mathcal{L}(q,\theta) &= \mathbb{E}\log\frac{p(h|u,x)p_\theta(u|x)}{q(h)} \\ &= \log p_\theta(u|x) - D_{KL}\big(q(h) \parallel p_\theta(h|u,x)\big)\end{aligned} \tag{4}$$

We can then follow the standard EM algorithm:

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

If maximizing Q is difficult, we can take a generalized M-step and perform a gradient ascent, i.e. $\Delta\theta = \alpha\frac{\partial Q}{\partial\theta}$. This is proved to converge in similar way as does standard EM.

3.3. Sequential Monte Carlo

For network with more than one hidden stochastic layers, calculating $p(h|u,x)$ is intractable. For training feed-forward SNNs, Tang et al (2013) employed importance sampling with $q(h) = p(h|x)$ to approximate the true posterior distribution $q^*(h) = p(h|u,x)$. Note that $p(h|x)$ constitutes the forward pass of the network and is easy to compute. In this paper, we generalize this idea to recurrent SNNs using a Sequential Monte Carlo (SMC) variant.

A natural choice of the proposal distribution for the recurrent setting is $q(\mathbf{h}_{0:T}) = p(\mathbf{h}_{0:T}|\mathbf{x}_{0:T})$. Observe that $q_t(h) \triangleq p(h_t|\mathbf{h}_{0:t-1}, \mathbf{x}_{0:t})$ is analogos to $p(h|x)$ in the feed-forward setting. We factorize the importance weights favorably for ease of sampling using Bayes' rule. For the i^{th} sample, we have:

$$\begin{aligned}w(\mathbf{h}_{0:T}^{(i)}) &= \frac{p(\mathbf{h}_{0:T}^{(i)}|\mathbf{u}_{0:T}, \mathbf{x}_{0:T})}{p(\mathbf{h}_{0:T}^{(i)}|\mathbf{x}_{0:T})} \\ &= \frac{p(h_0^{(i)}|x_0, u_0)}{p(h_0^{(i)}|x_0)}\prod_{t=1}^T\frac{p(h_t^{(i)}|\mathbf{h}_{0:t-1}^{(i)}, \mathbf{u}_{0:t}, \mathbf{x}_{0:t})}{p(h_t^{(i)}|\mathbf{h}_{0:t-1}^{(i)}, \mathbf{x}_{0:t})}\end{aligned} \tag{6}$$

$$w_T^i \triangleq \zeta_0^i\prod_{t=1}^T\zeta_t^i = w_{T-1}^i\zeta_T^i \tag{7}$$

where we introduce ζ_t^i and w_t^i to lighten notations and where (7) is equivalent to (6) and suggests that we can compute the importance weights recursively.

Now we focus on calculating w_t^i. The base case $w_0^i = \zeta_0^i$ corresponds to Tang et al (2013). For the general case, first assume w_{t-1}^i is known, then it suffices to calcualte ζ_t^i:

$$\begin{aligned}\zeta_t^i &= \frac{p(h_t^{(i)}|\mathbf{h}_{0:t-1}^{(i)}, \mathbf{u}_{0:t}, \mathbf{x}_{0:t})}{p(h_t^{(i)}|\mathbf{h}_{0:t-1}^{(i)}, \mathbf{x}_{0:t})} \\ &= \frac{p(\mathbf{h}_{0:t}^{(i)}, \mathbf{u}_{0:t}, \mathbf{x}_{0:t})}{p(\mathbf{h}_{0:t}^{(i)}, \mathbf{x}_{0:t})p(\mathbf{u}_{0:t}|\mathbf{h}_{0:t-1}^{(i)}, \mathbf{x}_{0:t})} = \frac{p(\mathbf{u}_{0:t}|\mathbf{h}_{0:t}^{(i)}, \mathbf{x}_{0:t})}{p(\mathbf{u}_{0:t}|\mathbf{h}_{0:t-1}^{(i)}, \mathbf{x}_{0:t})} \\ &= \frac{p(u_0|h_0,x_0)p(u_t|\mathbf{h}_{0:t}^{(i)}, \mathbf{x}_{0:t})\prod_{\tau=1}^{t-1}p(u_\tau|\mathbf{h}_{0:\tau}, \mathbf{x}_{0:\tau})}{p(u_0|h_0,x_0)p(u_t|\mathbf{h}_{0:t-1}, \mathbf{x}_{0:t})\prod_{\tau=1}^{t-1}p(u_\tau|\mathbf{h}_{0:\tau}, \mathbf{x}_{0:\tau})} \\ &= \frac{p(u_t|\mathbf{h}_{0:t}^{(i)}, \mathbf{x}_{0:t})}{p(u_t|\mathbf{h}_{0:t-1}, \mathbf{x}_{0:t})} \simeq \frac{p(u_t|\mathbf{h}_{0:t}^{(i)}, \mathbf{x}_{0:t})}{\frac{1}{M}\sum_j p(u_t|\mathbf{h}_{0:t}^{(j)}, \mathbf{x}_{0:t})}\end{aligned} \tag{8}$$

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

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

Note that the normalization constant $Z_t = \sum_i \zeta_t^i = M$ for all t. We can readily define $\tilde{\zeta}_t^i = \frac{1}{M}\zeta_t^i$ and hence the normalized weights $\tilde{w}_t^i = \tilde{w}_{t-1}^i\tilde{\zeta}_t^i = \prod_{\tau=0}^t\tilde{\zeta}_\tau^i$.

The above algorithm is known as Sequential Importance Sampling (SIS). A common pitfall of directly applying SIS is that for large T, as t increases many samples are likely to have negligible weights, resulting in numerical instabilities and degenerate weight distributions. Another variant

Algorithm 1 SIS Generalized EM

Input: training trajectories $\{(\mathbf{x}_{0:T}^{(n)}, \mathbf{u}_{0:T}^{(n)}, \mathbf{\Sigma}_{\mathbf{u}0:T}^{(n)})\}_{n=1}^{N}$, 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**
      $h_1^{(i)} \sim p(h_1|h_0, x_1)$
      $l_1^i = p(h_1^{(i)}|h_0, x_1)$
      $w_1^i = p(u_1|\mathbf{h}_{0:1}^{(i)}, x_1)$
    end for
    $\tilde{w}_1^i = w_1^i / \sum_i w_1^i$
    for $t = 2$ **to** T **do**
      for $i = 1$ **to** M **do**
        $h_t^{(i)} \sim p(\mathbf{h}_{0:t}|\mathbf{h}_{0:t-1}, \mathbf{x}_{0:t})$
        $l_t^i = p(\mathbf{h}_{0:t}^{(i)}|\mathbf{h}_{0:t-1}, \mathbf{x}_{0:t})$
        $w_t^i = p(u_t|\mathbf{h}_{0:t}^{(i)}, \mathbf{x}_{0:t})$
      end for
      $\tilde{w}_t^i = w_t^i / \sum_i w_t^i$
    end for
  end for
  {M-step}
  for $i = 1$ **to** M **do**
    compute $\Delta Q^{(i)} \triangleq \frac{\partial}{\partial \theta}\big[\sum_{t=0}^{T} \log q_t^i + \log l_t^i\big]$ using standard back-propagation
  end for
  sga_update($\theta, \sum_{i=1}^{M} \tilde{w}_T^i \Delta Q^{(i)}$) {any stochastic gradient ascent algorithm such as ADAM or RMSProp}
until convergence

of SMC, called Sequential Importance Resampling (SIR), mitigates this issue by resampling $h_t^{(i)}$ at each t based on the weight distributions formed by samples at previous step. However, for recurrent neural networks trained with small number of steps unrolled, SIR is optional.

3.4. Training Procedure

In E-step, we perform SIS or SIR to approximate the true posterior distribution of $\mathbf{h}_{0:T}$ using our proposal distribution $q(\mathbf{h}_{0:T}) = p(\mathbf{h}_{0:T}|\mathbf{x}_{0:T})$ which is easy to compute. In this process, without additional efforts, we retain the below values useful for M-step:

$$q_t^i \triangleq p(h_t^{(i)}|\mathbf{h}_{0:t-1}, \mathbf{x}_{0:t}) \qquad l_t^i \triangleq p(u_t|\mathbf{h}_{0:t}^{(i)}, \mathbf{x}_{0:t}) \tag{9}$$

In M-step, we perform gradient ascent on Q approximated by SMC in E-step:

$$\begin{aligned} Q(\theta) &= \mathbb{E}_{\mathbf{h}_{0:T}\sim q^*} \log p_\theta(\mathbf{u}_{0:T}, \mathbf{h}_{0:T}|\mathbf{x}_{0:T}) \\ &\simeq \sum_{i=1}^{M} \tilde{w}_T^i \big[\log p_\theta(\mathbf{u}_{0:T}|\mathbf{h}_{0:T}^{(i)}, \mathbf{x}_{0:T}) + \log p_\theta(\mathbf{h}_{0:T}^{(i)}|\mathbf{x}_{0:T})\big] \\ &= \sum_{i=1}^{M} \Big(\prod_{t=0}^{T} \frac{q_t^i}{\sum_j q_t^j}\Big) \big[\sum_{t=0}^{T} \log q_t^i + \log l_t^i\big] \end{aligned}$$

Note that the importance weights are unrelated to θ; hence the gradient step:

$$\frac{\partial Q}{\partial \theta} \simeq \sum_{i=1}^{M} \Big(\prod_{t=0}^{T} \frac{q_t^i}{\sum_j q_t^j}\Big) \frac{\partial}{\partial \theta} \big[\sum_{t=0}^{T} \log q_t^i + \log l_t^i\big] \tag{10}$$

Algorithm 1 summaries the overall algorithm of training a recurrent SNN. The reference control covariance matrix $\mathbf{\Sigma}_{\mathbf{u}0:T}$ is for calculating l_t terms and is assumed easy to acquire by trajectory optimization algorithm. (Indeed, the iLQG algorithm, for example, outputs $\mathbf{\Sigma}_{\mathbf{u}0:T}^{-1}$ as a by-product.)

4. Experimental Evaluation

4.1. Implementation

We independently implemented flexible software package [3], based on Computation Graph Toolkit[4], that allows convenient construction of arbitrary structure of neural network. A highlight of our software is that it supports layer with mixed types of units, including standard fully-connected units, Bernoulli stochastic units, long-short-term-memory block, and common non-linearities. We enjoyed this flexibility to carry out agile experiments. For learning, we implemented the base case of our proposed Sequential Importance Sampling variant of Generalized EM algorithm for an arbitrary (feed-forward) SNN. Training and experimentation of recurrent SNN is left as a future work.

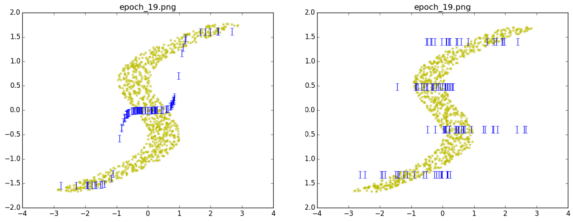

Figure 2. Samples from the learnt DNN (left) and the learnt SNN (right).

4.2. Synthetic Dataset

Here we reproduce, using our implementation, known results of (feed-forward) SNNs based on a one-dimensional

[3] available at https://github.com/TZ2016/snncp/
[4] http://rll.berkeley.edu/cgt/

synthetic dataset to motivate more complex example in next section. The synthetic data is given by:

$$x = y + 0.3\sin(2\pi y) + \epsilon \tag{11}$$

where $\epsilon \sim \text{Unif}(-0.1, 0.1)$ and $y \in [0, 1]$. If we take x as inputs and y outputs, the dataset realizes one-to-many mappings for some x, i.e. $p(y|x)$ is multi-modal.

For simplicity, we take constant variance $\Sigma^\pi(x) = \sigma^2$ for the output distribution $p_\theta(y|x)$. We separately trained a feed-forward DNN and SNN (with 4 stochastic units) on the above dataset. For hybrid SNN, we use $M = 30$ to estimate gradient and ADAM (Kingma & Ba, 2014) as gradient ascent algorithm in the generalized M-step. All other training hyperparams are held fixed.

In Figure 2, yellow dots represent data points in the training set after normalization. The blue vertical segments represent samples from the learnt neural network with the length representing variance. Note that DNN outputs the conditional average of y given x as expected and fails to account for the many modes of the true distribution. In contrast, SNN is able to learn different conditional density masses. Due to the limited number of nodes in this case, at most 4 modes are learnt.

4.3. Quadrotor Control

Zhang et al (2015) demonstrated in their work that a feed-forward DNN is able to successfully perform complex control policies under raw observation inputs (see Figure 3). Due to the limitation of DNN, we verify below that the method no longer works with conflicting examples in the training set.

A total of 15 randomized initial positions at roughly in front of the obstacle are pre-generated. For each condition, we used iLQG algorithm to general a locally optimal time-varying Gaussian policy based on full state information, i.e. position, orientation, velocity. Then each local optimal policy is allowed to roll out 5 times in simulation; the resulting trajectories of optimal controls (of dimension 4) and corresponding raw sensor observations (of dimension 40) are collected as the training set. Note that during trajectory optimization, some local optimal policies choose to dodge left and the others dodge right, as in Figure 4. This dataset, though much more complex, is analogous to that in Figure 2, in that controls are uni-modal for some states and two-modal for others.

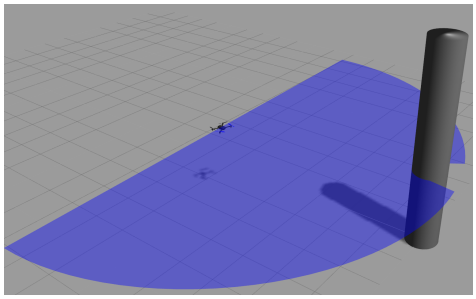

Figure 3. A 3D view of the quadrotor and a cylinder obstacle in simulation. The blue region are the limited range laser inputs the neural network operates on.

Figure 4. 2D view (from above) of two typical training trajectories generated from local optimal policies. (Note that the simulation is still in 3D)

For training both DNN and SNN, we use three hidden fully-connected inner-product layers of sizes 20, 15, and 4, respectively. A small weight decay (5×10^{-3}) is used to prevent overfitting. We allow learning objective to plateau with sufficient number of epochs (for both cases, 15 epochs are used).

Trained using the existing method, at test time, the learnt deterministic neural network policy results in high ratio (approx. 50%) of crashes (Figure 5). Again notice the analogy to the simple synthetic dataset; in both cases, the deterministic network learns the conditional average of examples from distinct modes. The presence of right turning trajectories are probably caused by the unimportant nuances in the training trajectories, which the deterministic learner associates with different turning directions. Under ideal set up where all controls except the turning point are the same, however, the deterministic policy should result in nearly 100% crashes.

Figure 5. Representative trajectories generated by deterministic neural network control policies. (existing method)

For SNN, we replace two of the deterministic units by Bernoulli stochastic ones. In contrast to DNN, we arrive at a more satisfactory result (Figure 6). At test time, not only fewer rollouts result in crashes, but also a meaningful variety of modes are acquired.

Both experiments are run multiple times with shuffled training data, and both results presented above are consistently reproduced. Table 1 summarizes the results.

Figure 6. Representative trajectories generated by SNN control policies. (our method)

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

5. Conclusions and Future Work

We demonstrated that, in scenarios where multiple competing controls exist, a deterministic neural network as the global control policy is no longer a reasonable option. Considering the many features neural networks enjoy, such as good generalizability, we propose to address this issue by introducing stochastic neural networks as control policies with potentials of multi-modality.

Using feed-forward SNN, however, does not completely solve the problem. Still, we have, though reduced, significant percentage of crashes. More work should be done to analyze the reason of these crashes. One potential reason is that, due to lack of mechanism to make consistent decisions through time, a hidden stochastic units may contradict its output at prior time steps. In the case of obstacle avoidance, contradicting decisions can lead to horizontal oscillation while the quadrotor approaches the cylinder. To address this issue, a promising direction is to integrate recurrence into stochastic units, allowing consistent decision making. With the learning algorithm outlined in this paper, we plan to experiment with a recurrent SNN in the future.

Another potential future work is on the learning algorithm itself. For SNNs, sampling multiple times is crucial to arrive at an useful gradient estimate. Depending on the complexity of the structure, the number of samples required may grow indefinitely and ultimately render the learning algorithm impractical. Several recent works propose to use a separate neural network to model the true posterior distribution, called recognition model (Kingma & Welling, 2013) (Gu et al., 2015). We can work towards generalizing this idea to recurrent SNNs.

Specifically about Sequential Monte Carlo, recent works propose to use a separate neural network which is updated at every time step for a much more accurate approximate distribution to sample from at the next step (Gu et al., 2015).

References

Gu, Shixiang, Levine, Sergey, Sutskever, Ilya, and Mnih, Andriy. Muprop: Unbiased backpropagation for stochastic neural networks. *arXiv preprint arXiv:1511.05176*, 2015.

Jordan, Michael I, Ghahramani, Zoubin, Jaakkola, Tommi S, and Saul, Lawrence K. An introduction to variational methods for graphical models. *Machine learning*, 37(2):183–233, 1999.

Kingma, Diederik and Ba, Jimmy. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Kingma, Diederik P and Welling, Max. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Levine, Sergey and Abbeel, Pieter. Learning neural network policies with guided policy search under unknown dynamics. In *Advances in Neural Information Processing Systems*, pp. 1071–1079, 2014.

Levine, Sergey, Finn, Chelsea, Darrell, Trevor, and Abbeel, Pieter. End-to-end training of deep visuomotor policies. *arXiv preprint arXiv:1504.00702*, 2015.

Tang, Yichuan and Salakhutdinov, Ruslan R. Learning stochastic feedforward neural networks. In *Advances in Neural Information Processing Systems*, pp. 530–538, 2013.

Tassa, Yuval, Erez, Tom, and Todorov, Emanuel. Synthesis and stabilization of complex behaviors through online trajectory optimization. In *IROS 2012*, 2012.

Zhang, Tianhao, Kahn, Gregory, Levine, Sergey, and Abbeel, Pieter. Learning deep control policies for autonomous aerial vehicles with mpc-guided policy search. *arXiv preprint arXiv:1509.06791*, 2015.