ENSAE SMC Project Report

January 6, 2023

Abstract

In this project, we mainly reproduce the method proposed by [1] based on our understanding.[1]

1 Introduction

Modern robotics often apply neural network for state prediction and control. In practice, people usually train a neural network over a large, comprehensive data and then deploy it. However, a such pretrained model cannot adapt to changes in the environment. Many adaptive control techniques have been considered for parametric models, but as to neural models, current adaptive approaches are still not easy to train and implement.

Given a pretrained neural network, model adaptation is usually fine-tuning by gradient descent, i.e. updating neural weights on a small dataset emitted by the current environment. However, gradient descent is not always feasible in production when it comes to a giant model or in online setting due to expensive gradient computations. Instead, [1] uses a particle filter in the dropout layer of the neural network to imitate the adaptation process to the changes of environment. The authors think that the pretrained model should have learned sufficient knowledge and be able to adapt to changes by masking some of the neurons. In other words, there is no update of knowledge but only a focus on knowledge for a specific task. The proposed adaptation approach is called **Sequential Monte Carlo Dropout (SMCD)**. Experiments on simulated and real data show that SMCD adapts better than the gradient method with less computations.

2 Problem

To better understand the task on which the SMCD applies, we give a brief introduction to robotics, in particular the state estimation problem[4]. The state estimation problem aims at estimating the state x of a system given its noisy observations z and controls u. In probabilistic language, the goal is to estimate $p(x_t|z_{1:t}, u_{1:t})$.

$$\begin{tikzcd}
 & u_{t-1} \arrow[d] & u_t \arrow[d] & \\
\cdots \arrow[r] & x_{t-1} \arrow[r] \arrow[d] & x_t \arrow[r] \arrow[d] & \cdots \\
 & z_{t-1} & z_t &
\end{tikzcd}$$

Figure 1: Graph model for evolution of states, observations and controls

The recursive Bayes filter (framework) was invented to solve this problem. Its basic philosophy is to update our belief of the current state based on our belief of the previous state and the current observation. The framework can be expressed as the following formula:

[1]Codes is provided here: https://github.com/alephpi/M2-SMC

$$\begin{aligned}
p(x_t|z_{1:t}, u_{1:t}) &= \eta p(z_t|x_t, z_{1:t-1}, u_{1:t})p(x_t|z_{1:t-1}, u_{1:t}) && (1)\\
&= \eta p(z_t|x_t)p(x_t|z_{1:t-1}, u_{1:t}) && (2)\\
&= \eta p(z_t|x_t)\int p(x_t|x_{t-1}, z_{1:t-1}, u_{1:t})p(x_{t-1}|z_{1:t-1}, u_{1:t})dx_{t-1} && (3)\\
&= \eta p(z_t|x_t)\int p(x_t|x_{t-1}, u_t)p(x_{t-1}|z_{1:t-1}, u_{1:t})dx_{t-1} && (4)\\
&= \eta p(z_t|x_t)\int p(x_t|x_{t-1}, u_t)p(x_{t-1}|z_{1:t-1}, u_{1:t-1})dx_{t-1} && (5)
\end{aligned}$$

where we apply the Markovian assumption from (1) to (2) and (2) to (3), and an assumption that future control will not affect the belief on the past state from (4) to (5).

If we denote $p(x_t|z_{1:t}, u_{1:t})$ as $bel(x_t)$, and decompose the equation into two steps, we will get the following:

$$\begin{aligned}
\overline{bel}(x_t) &= \int p(x_t|x_{t-1}, u_t)bel(x_{t-1})dx_{t-1} && (6)\\
bel(x_t) &= \eta p(z_t|x_t)\overline{bel}(x_t) && (7)
\end{aligned}$$

where (6) is called the motion/prediction step, and (7) is called the observation/correction step. Respectively, $p(x_t|x_{t-1}, u_t)$ is called the prediction model and $p(z_t|x_t)$ is called the observation model.

We point out that in [1] settings, the prediction model $p(x_t|x_{t-1}, u_t)$ is a pretrained network $x_t \sim f_\theta(x_{t-1}, u_t|M_t)$ whereas the observation model is a common Gaussian model $p(z_t|x_t) = \mathcal{N}(x_t, \Sigma)$.

3 Method

3.1 Dropout

Let us assume that $f_\theta(\cdot|M)$ is a neural network trained using a dropout regularization M and with neural weights θ. A dropout layer is a mask which values are either 0 or 1 that acts on a hidden layer. In common deep learning setting, the dropout technique is only applied during training which randomly zeros out neurons in the forward passing step. This is to alleviate co-adaptation between neurons, so that each neuron depends less on others and holds more effective information[2]. As to the prediction stage, the dropout layer is usually disabled.

However [1] tries to also activate dropout masks to serve model adaptation and then state prediction. Mathematically, dropout mask M also takes part in the forward passing during the prediction stage, i.e. $x_t \sim f_\theta(x_{t-1}, u_t|M_t)$. We can think that such a step performs a feature selection (M_t applies on θ) during the prediction on x_t, which adapts to the specific prediction task (finds the most salient neurons relevant to the environment at time t).

3.2 Recursive mask adaptation

The paper mainly studies how to update neural prediction model based on observations. In order to perform adaptation in terms of mask, the paper introduces mask M into the graph model (Fig.2).

$$\begin{tikzcd}
& u_{t-1} \arrow[d] & u_t \arrow[d] & \\
{} \arrow[r] & x_{t-1} \arrow[r] \arrow[d] & x_t \arrow[r] \arrow[d] & {} \\
{} \arrow[d] & z_{t-1} \arrow[d] & z_t \arrow[d] & \\
M_{t-1} \arrow[uur] & M_t \arrow[uur] & {} &
\end{tikzcd}$$

Figure 2: Graph model for evolution of states, observations, controls and dropout masks

Mathematically, our goal is to estimate $p(M_t|z_{1:t})$. Similar to the deduction in recursive Bayes filter:

$$\begin{aligned}
p(M_t|z_{1:t}) &= \eta p(z_t|M_t, z_{1:t-1})p(M_t|z_{1:t-1}) && (8)\\
&= \eta p(z_t|M_t)p(M_t|z_{1:t-1}) && (9)\\
&= \eta p(z_t|M_t)\int p(M_t|M_{t-1}, z_{1:t-1})p(M_{t-1}|z_{1:t-1})dM_{t-1} && (10)\\
&= \eta p(z_t|M_t)\int p(M_t|M_{t-1})p(M_{t-1}|z_{1:t-1})dM_{t-1} && (11)
\end{aligned}$$

where $p(z_t|M_t) = \mathcal{N}(f_\theta(x_{t-1}, u_t|M_t), \Sigma)$ according to our assumption on the observation model, and $p(M_t|M_{t-1})$ is modeled by a multivariate Bernoulli distribution with probability d, i.e. a bit-flip with probability d for each binary value in the mask.

To summarize, the major difference between the recursive Bayes filter for state estimation and the recursive mask adaptation is that we update based on observation our prediction model rather than simply the prediction itself (focusing on different paths in the graph model).

4 Implementation

We have implemented the Two-Link Arm Task to verify the proposed method. The system is illustrated in Fig.4.1. As described in [1], the two-link arm follows the following dynamics and forward kinematics:

$$\dot{q_1} = u_1, \dot{q_2} = u_2 \tag{12}$$

$$x = l_1\cos(q_1) + l_2\cos(q_1+q_2), y = l_1\sin(q_1) + l_2\sin(q_1+q_2) \tag{13}$$

We try to build a neural network that learns the forward kinematics (13), namely $x_t \sim f_\theta(q_t|M_t)$[2], where $x_t = (x, y)$ represents the end-effector position at time t. There are three stages to reproduce the proposed method: we first pretrain a neural network on the simulated data, by two different pretraining approaches (multi-task and reptile [3]). Then, we adopted multiple adaptation ways (SMCD proposed by [1], Gradient descent and No adaptation) to fine-tune the neural network. In the end, the predictions were performed using fine-tuned network, with different look-ahead steps.

4.1 Data simulation

The task of neural network is to perform forward kinematics: predict the end-effector position $[x, y]^T$ given joint angles $q = [q_1, q_2]^T$ for variable limb lengths l_1, l_2.

[2]Notice here the formulation is different from section 3 where $x_t \sim f_\theta(x_{t-1}, u_t|M_t)$. This is because x_{t-1} contains the information of q_{t-1} and $u_t = \dot{q}_t = q_t - q_{t-1}$, while directly feed x_{t-1} to the neural network doesn't work well since the inverse kinematic (solve q from x) is too complex to be captured by a 3-layer neural network.

In pretraining stage, we expect the neural network learn forward kinematic (13) from a large collection of (q, x) pair generated with different limb length. In order to do that, we generated 150 episodes for 1000 tasks, each task representing a different limb lengths setting l_1, l_2 sampled from $\mathcal{N}(1, 0.3^2)$. Each episode is randomly initialized with $q_1, q_2 \sim U(-\pi, \pi)$, and evolving for 10 timesteps under the motor babbling control $u_1, u_2 \sim \mathcal{N}(0, 1)$.[3]

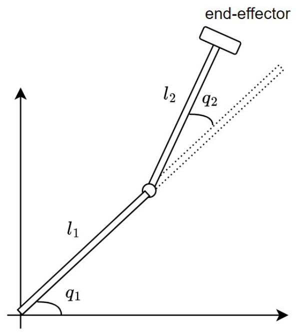

Figure 3: Two-Link Arm System

In adaptation stage, we expect the pretrained neural network adapts to a new task (with unseen limb lengths), we generate 100 different tasks for 200 timesteps under motor babbling control as well.

In prediction stage, we expect the adapted neural network continue to perform predictions on the adapting task, by predicting the end-effector position under a PD control which drives the end-effector to a specific reference variable smoothly. The ordinary PD control is: $\tau = -k_p(q - q_d) - k_d\dot{q}$ where k_p is the proportional gain and k_d is the derivative gain. Here in [1], the proportional term is adapted as: $u = -K_1 J(q)(x - x_g) - K_2\dot{q}$ where $J(q)$ corresponds to the Jacobian of (13) and x_g is the referenced position. Here, x_g is randomly chosen within 5cm from the initial position.[4] We simulate for each tasks in adaptation stage for 200 timesteps under the PD control, where $K_1 = 1, K_2 = 0.01$.[5]

4.2 Pretraining

A 3-layer neural network (illustrated in Fig.4) is constructed in Pytorch with the following hyperparameters: hidden dimension 512, dropout probability 0.5, Adam optimizer with learning rate $1e^{-4}$, batch size 128, number of epochs 50.

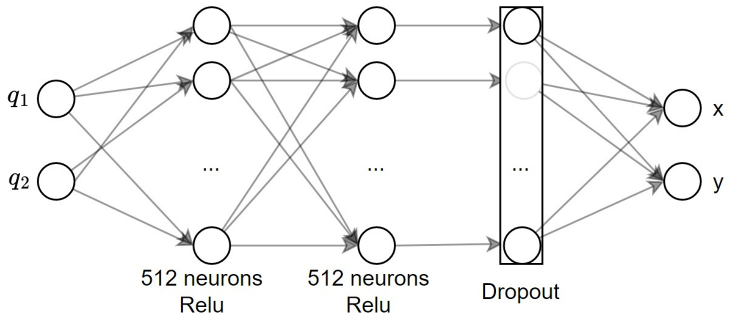

Figure 4: Neural network structure

We adopted two training strategies: multi-task and Reptile. In multi-task training, data from different tasks are mixed up to feed into network, while in reptile training data are fed into network task by task, and the weights are updated using interpolation between current weights and trained weights, i.e. $w = w_{\text{before}} + (w_{\text{before}} - w_{\text{after}}) * lr_{\text{inner}}$. Here we set the inner learning rate to 0.02 and the inner number of epochs to 1. Refer to this blog for more details on Reptile.

4.3 Adaptation

In this section, we explain several implementation details of the mask adaptation method. Overall, the prediction step is implemented by a bit-flipping step, and the correction step is implemented by an importance sampling and resampling step.

In the bit-flipping step we generate a binary array from a multivariate Bernouilli distribution with probability d, and apply it on the mask with an xor operation.[6] Typical candidates for d are 0.05, 0.1, 0.15. An aggressive

[3]To ensure the convergence of neural network, all the angles are normalized into $[-\pi, \pi]$

[4]As stated in [1], we focus on state prediction and not the policy decision, here we always follow the PD control policy where the control is made upon the prediction of state.

[5]hyperparameters need to be tuned manually according to the relative magnitude of quantities that appears in the task.

[6]Another understanding is to bit-flip a fixed $d\%$ of the masks, which forces to leave from the current mask configuration, proved to be sub-optimal in practice.

d tends to explore the configuration space, while a conservative d tends to exploit around the neighborhood. In practice, it should be fine-tuned depends on the task.

In importance sampling step, we generate importance weights from the observation model with uncertainty Σ.[7] A big Σ is more tolerable to the particles far from the optimal while a small Σ will strictly rule out unsuitable particles. In practice it also need to be manually tuned, we choose $\Sigma = 0.001$ based on [1].

In importance resampling step, we adopt the systematic resampling strategy (also known as stochastic universal sampling or Low-variance resampling) due to its computation efficiency (linear complexity) More importantly, it does not introduce bias as sampling with replacement.[4].

We adapt the pseudo code in [1] here for convenience. [h] Sequential Monte Carlo Dropout (SMCD) Model Adaptation

```
Input: Trained net $f_\theta$, Observation sequence $z_{1:T}$,
Transition probability $d$, Measurement uncertainty $\Sigma$
Output: Mask estimate sequence $\overline{M}_{1:T}$
Initialise N random masks $\{M_0^i\}_{i=1...N}$
for $t = 1...T$ do
  for $i = 0...N$ do
    Sample $M_t^i \sim p(M_t^i | M_{t-1}^i)$ by bit-flipping
    Predict $x_t^i = f_\theta(x_{t-1}, u_t | M_t^i)$
    Generate importance weight $w_i = \mathcal{N}(z_t | x_t^i, \Sigma)$
    Resample masks: $\{M_t^i\}_{i=1...N} \sim \frac{\sum_{i=1}^N w_i \delta(M_t^i)}{\sum_{i=1}^N w_i}$
    Compute best mask estimate $\overline{M}_t = \frac{1}{N}\sum_{i=1}^N M_t^i$ and binarize with threshold 0.5
  end for
end for
```

4.4 Prediction

We take the last best mask after adaptation to use it on the prediction of the end-effector trace under PD control. We only feed in the initial angle of the system and perform predictions in look-ahead scheme as illustrated in Fig.5.

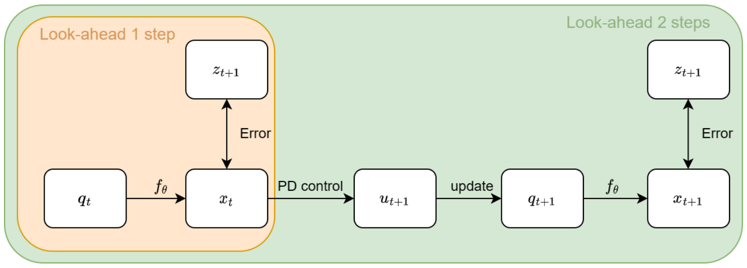

Figure 5: Look ahead scheme

5 Evaluation

All data are pretrained for 50 epochs with two strategies. We notice that pretraining with multi-task strategy converges faster than the reptile strategy, which proves what the original paper says. Here, M stands for Multi-task, and R stands for Reptile.

As to adaptation, we replicate 10 times SMCD adaptation on 100 timesteps with 1000 particles on 100 tasks. We compute the mean of adaptation loss on each task and sort them in increasing order, compare to another

[7] In practice, we replace $p(x_i|z_t)$ by $p(z_t|x_i)$ to compute the PDF of observation model to benefit from the vectorization of Numpy, without significant change in performance. Also, we apply the log-scale trick to alleviate numerical problem.

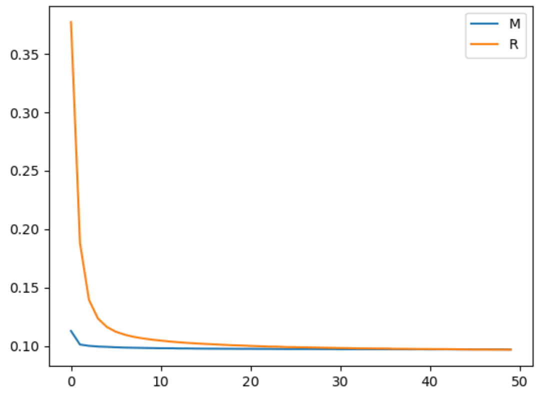

Figure 6: loss plot in pretraining stage

two adaptation baselines (no adaptation and gradient descent)

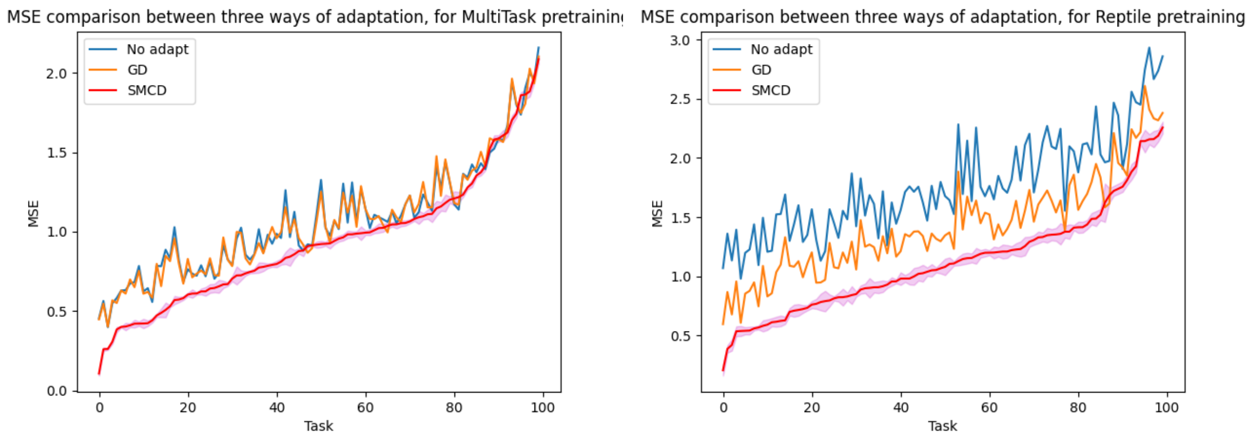

Figure 7: MSE comparison for two pretraining strategies

We notice that in general, the better initial prediction before adaptation is, the better SMCD adaptation loss will improve. This is quite reasonable as if the initial loss is low i.e. the initial mask configuration is closer to the optimal then the particles are more likely to reach there, while on the contrary, the particles are struggled to find a path to the mode of PDF, the loss sometimes become even worse due to the bad initialization.

For multi-task pretraining, we can see the gradient descent adaptation hardly improves from the original prediction, this can be interpreted by the lack of data (here only 100 time steps per task). For Reptile pretraining, there is an obvious improvement of gradient descent adaptation with respect to the original prediction, but the improvements doesn't excess the one in multi-task training, we think it's because the model is not sufficiently trained. Above all, in adaptation stage, we reproduce that the proposed method does outperform the gradient based method, while we failed to get a good result for every tasks in general.

As to the prediction, the result is not promising. It seems that the PD control is very sensitive to the predicted position, therefore there is not much meaningful prediction compared to the oracle trace. Here we pick two traces that seems reasonable, notice that since the model is not adapting in an online fashion, the error accumulates. Again this is one try for downstream tasks after adaptation since [1] doesn't make it very clear.

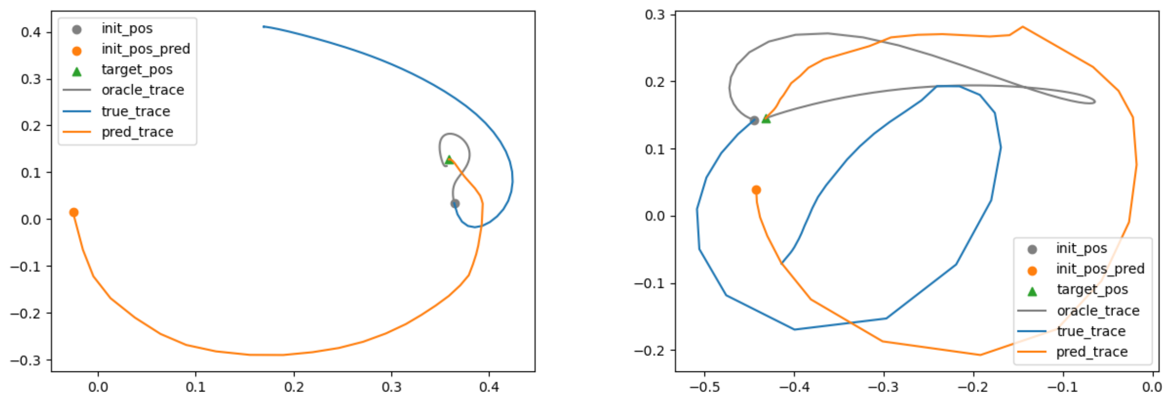

Figure 8: Two traces of look ahead prediction for 200 timesteps
The orange trace is what the model perceives, the blue trace is the trace under control in practice and the grey curve is the oracle one

6 Conclusion

We have partially reproduced the experimental results in [1] with our own understandings. The proposed method SMCD adaptation method does outperform the traditional gradient descent method in condition of little adaptation data. However, we didn't get a good result on each tasks in general. As to [1], future work may focus on realistic data instead of simulated one, and also we hope the experimental data can be released to facilitate community's reproduction and verification.

References

[1] Pamela Carreno-Medrano, Dana Kulić, and Michael Burke. *Adapting Neural Models with Sequential Monte Carlo Dropout*. 2022. DOI: 10.48550/ARXIV.2210.15779. URL: https://arxiv.org/abs/2210.15779.

[2] Geoffrey E. Hinton et al. *Improving neural networks by preventing co-adaptation of feature detectors*. 2012. DOI: 10.48550/ARXIV.1207.0580. URL: https://arxiv.org/abs/1207.0580.

[3] Alex Nichol, Joshua Achiam, and John Schulman. "On First-Order Meta-Learning Algorithms". In: *CoRR* abs/1803.02999 (2018). arXiv: 1803.02999. URL: http://arxiv.org/abs/1803.02999.

[4] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005. ISBN: 0262201623.