Contents

Chapter 1

Model

1.1 Notations

We consider the prediction task of a set of observations $(y^0, \cdots y^T)$ given a set of input $(u^0, \cdots u^T)$. The model can be built on top of a arbitrary encoder model, denoted h, that will be trained by gradient descent. Through this encoder, inputs are mapped to latent vectors:

$$h : (u^0, \cdots u^T) \mapsto (\tilde{u}^0, \cdots \tilde{u}^T)$$

1.2 Model

Our model can be viewed as a RNN layer followed by a fully connected layer. At time step t,

$$\begin{cases} y_{t+1} = \tanh(W_y x_{t+1}^L + b_y) \\ x_{t+1} = \tanh(W_{xx} x_t + W_{xu} \tilde{u}_{t+1} + b_x) \end{cases}$$

with $x_0 \equiv 0$.

Let's consider the weights of the last RNN and fully connected layers as $\theta \equiv (W_{xx}, W_{xu}, b_x, W_y, b_y)$. We introduce two sequences of random noises as i.i.d real valued random variables ϵ and η, with covariance matrices Σ_y and Σ_x. We can now write our model in terms of two functions f and g as:

$$\begin{cases} y_t = f_\theta(x_t) + \epsilon_t & \text{observation model} \\ x_{t+1} = g_\theta(x_t, \tilde{u}_{t+1}) + \eta_{t+1} & \text{state model} \\ \tilde{u}_t = h(u_t) & \text{command model} \end{cases} \tag{1.1}$$

In the following section, we will focus on maximizing the joint log likelihood:

$$\log\ p_\theta(X_{0:T}, y_{0:T}, u_{0:T}) \tag{1.2}$$

1.3 Minimization

We can start by developing the log likelihood:

$$\begin{aligned}
\log p_\theta(X_{0:T}, y_{0:T}, u_{0:T}) &= \frac{1}{T}\log\left(p_\theta(x_0)p_\theta(y_0|x_0)\prod_{k=1}^{T} p_\theta(x_k|x_{k-1},u_k)p_\theta(y_k|x_k)\right) \\
&= \frac{1}{T}\log p_\theta(x_0) \\
&+ \frac{1}{T}\sum_{k=1}^{T}\log\ p_\theta(x_k|x_{k-1},u_k) + \frac{1}{T}\sum_{k=0}^{T}\log\ p_\theta(y_k|x_k) \\
&= \frac{1}{T}\log p_\theta(x_0) - \frac{1}{2}\log|\Sigma_x| - \frac{1}{2}\log|\Sigma_y| + Cst \\
&- \frac{1}{2T}\sum_{k=1}^{T}(x_k - g_\theta(x_{k-1},u_k))'\Sigma_x^{-1}(x_k - g_\theta(x_{k-1},u_k)) \\
&- \frac{1}{2T}\sum_{k=0}^{T}(y_k - f_\theta(x_k))'\Sigma_y^{-1}(y_k - f_\theta(x_k))
\end{aligned}$$

We aim at maximizing 1.2 by gradient descent, by leveraging fisher's identity:

$$\nabla \log p_\theta(x_{0:T}, y_{0:T}, u_{0:T}) = \mathbb{E}_\theta\left[\nabla \log p_\theta(x_{0:T}, y_{0:T}, u_{0:T})|Y_{0:T}\right]$$

In order to approximate this expectation, we need to sample from the posterior distribution $p_\theta(x|y)$. In Section 1.4, we detail a Sequential Monte Carlo approach to this end. In Section 1.5, we describe the algorithm to train our model through gradient descent.

1.4 Sequential Monte Carlo Approach

1.4.1 Filter

In order to compute the conditional expectations in the previous expressions, we will iteratively sample trajectories $\xi^i_{1:T}$ associated with weights ω^i with respect to the density $p_\theta(x|y)$, using a sequential Monte Carlo particle filter.

At time step $k = 0$, $(\xi^l_0)_{l=1}^N$ are sampled independently from the first hidden state, and associated with sampling weights proportional to the observation density:

$$\begin{aligned}
\xi^i_0 &\sim \mathcal{N}(x_0, \Sigma_x) \\
\omega^i_0 &\sim p_\theta(y_0|\xi^i_0)
\end{aligned}$$

At time step $k+1$, we sample indices I of the particles to propagate, based on their previous weights. After propagation, particles weights are computed following the observation density:

$$\mathbb{P}(I^i_{k+1} = j) = \omega^j_k \quad \forall 1 \leq j \leq N$$

$$\xi_{k+1}^i \sim p_\theta(x_{k+1}|\xi_k^{I_{k+1}^i})$$
$$\omega_{k+1}^i \sim p_\theta(y_{k+1}|\xi_{k+1}^i)$$

Algorithm 1: Particle filter

Input: $\theta, y_{1:T}, u_{1:T}$
Output: $\xi_{1:T}^{1:N}$
$\xi_0^i \leftarrow \eta^i$;
for $k \leftarrow 0$ **to** T **do**
  $\omega_k^i \leftarrow \varphi_{y_k,\Sigma_y}(f_\theta(\xi_k^i))$;
  $I_{k+1}^i \sim \mathbb{P}(I_{k+1}^i = j) = \omega_k^j$;
  $\xi_{k+1}^i \leftarrow g_\theta(\xi_k^{I_{k+1}^i}, u_{k+1}) + \eta^i$;

1.4.2 Smoother

Using the poor man filter, we get N trajectories:

$$\xi_{1:k+1}^i = (\xi_{1:k}^{I_{k+1}^i}, \xi_{k+1}^i)$$

1.4.3 Approximation

We can now approximate this conditional expectation for any measurable bounded function h:

$$\mathbb{E}_\theta\left[h(x)|y_{1:T}\right] = \sum_{i=1}^N \omega_T^i h(\xi_{0:T}^i)$$

1.5 Gradient descent

1.5.1 Forward pass

During the forward pass, we generate a set of N particles under the law $p(x|y)$ for fixed values of θ, Σ_x and Σ_y.

1.5.2 Loss function

Considering that we have computed a set of N trajectories $(\xi_{0:T}^i)$, $1 \leq i \leq N$, associated with weights (ω^i), we can approximate the gradient of the log likelihood by computing the gradient of:

$$\begin{aligned}\mathbb{J}(\theta) &= \log|\Sigma_x| + \log|\Sigma_y| \\ &+ \frac{1}{T}\sum_{k=1}^T\sum_{i=1}^N \omega^i (y_k - f_\theta(\xi_k^i))'\Sigma_y^{-1}(y_k - f_\theta(\xi_k^i)) \\ &+ \frac{1}{T}\sum_{k=0}^T\sum_{i=1}^N \omega^i (\xi_k^i - g_\theta(\xi_{k-1}^i, u_k))'\Sigma_x^{-1}(\xi_k^i - g_\theta(\xi_{k-1}^i, u_k))\end{aligned}$$

1.5.3 Backward pass

During this step, each parameter of the model is updated by gradient descent.

1.5.4 Algorithm

Algorithm 2: Gradient descent

Input: $\theta_0, y_{1:T}, u_{1:T}$
Output: θ_p
for $p \leftarrow 0$ **to** n_{epochs} **do**
  $\xi_{1:T}^{1:N} \leftarrow$ `particule_filter`$(y_{1:T})$;
  $\theta_{p+1} \leftarrow \theta_p - \alpha \nabla \mathbb{J}(\theta_p)$;

1.6 Two-step training

In this section, we aim at improving the weights of a model already trained in a traditional fashion.

1.6.1 Initial training

Given a dataset of samples $(u_{0:T}^{(i)}, y_{0:T}^{(i)})_{i=1}^{m}$, we consider a training of our model as defined in 1.1 without the added noise. For each sample, we simply compute a prediction $\hat{y}_{0:T}$ by running the input through each layer, and minimize the discrepancy to the target values by gradient descent.

$$\mathbb{J} = \|u_{0:T} - y_{0:T}\|^2$$

This training results in an initial set of weights θ_0.

1.6.2 Finetuning

Because we trust the initial training has successfully extracted relevant information from the command u, and reached satisfactory parameters for the hidden state model, we will only adapt the parameters of the observation model:

$$\theta_{finetune} \equiv (\cancel{W_{xx}}, \cancel{b_x}, W_y, b_y)$$

Σ_x is set to a small value and will not be updated during the finetuning.

1.7 Prediction at $t+1$

Suppose that at time t, we have access to past observations $y_{1:t}$, along with current weighted particles trajectories $\{\xi_t^i, \omega_t^i\}_{1\leq i\leq N}$. Then,

$$\begin{aligned} p(y_{t+1}|y_{1:t}) &= \int p(y_{t+1}, x_{t+1}, x_t|y_{1:t})\mathrm{d}x_{t+1}\mathrm{d}x_t \\ &= \int p(y_{t+1}|x_{t+1})p(x_{t+1}|x_t)p(x_t|y_{1:t})\mathrm{d}x_{t+1}\mathrm{d}x_t \\ &\approx \sum_{i=1}^{N} \omega_t^i \int p(y_{t+1}|x_{t+1})p(x_{t+1}|\xi_t^i)\mathrm{d}x_{t+1} \end{aligned}$$

In order to approximate the last integral, we sample for each particle $1 \leq i \leq N$ a set of M particles under the law $\hat{\xi}_{t+1}^{i,j} \sim p(x_{t+1}|\xi_t^i)_{1\leq j\leq M}$. We write:

$$p(y_{t+1}|y_{1:t}) \approx \sum_{i=1}^{N} \omega_t^i \frac{1}{M} \sum_{j=j}^{M} p(y_{t+1}|\hat{\xi}_{t+1}^{i,j}) \tag{1.3}$$

We want a point estimation of the observation at $t+1$ along with a quantification of the uncertainty, so we compute the mean and variance of $p(y_{t+1}|y_{1:t})$.

$$\mathbb{E}[y_{t+1}|y_{1:t}] \approx \sum_{i=1}^{N} \omega_t^i \frac{1}{M} \sum_{j=j}^{M} \mathbb{E}[y_{t+1}|\xi_{t+1}^{i,j}] \tag{1.4}$$

$$\mathrm{Var}[y_{t+1}|y_{1:t}] \approx \sum_{i=1}^{N} \omega_t^{i\,2} \frac{1}{M} \sum_{j=j}^{M} \Sigma_y = \Sigma_y \sum_{i=1}^{N} \omega_t^{i\,2} \tag{1.5}$$

Algorithm 3: Prediction at time t+1

Input: $\xi_t^{1:N}, y_{1:t}$
Output: μ_{t+1}, σ_{t+1}
$\hat{\xi}_{t+1}^{i,j} = g_\theta(\xi_t^i) + \eta_t^{i,j}$;
$\mu_{t+1} = \sum_{i=1}^{N} \omega_t^i \frac{1}{M} \sum_{j=j}^{M} f_\theta(\hat{\xi}_{t+1}^{i,j})$;
$\sigma_{t+1}^2 = \Sigma_y \sum_{i=1}^{N} \omega_t^{i\,2}$;

Chapter 2

Simulation plan

2.1 Finetuning for noisy dataset

For this simulation, our objective is to generate a noisy dataset of pairs $(u_{0:T}^{(i)}, y_{0:T}^{(i)})_{i=1}^{m}$, to obtain an initial set of parameters by traditional gradient descent minimizing a loss criterion, then fine-tune by adding noise to the model and maximizing the log likelihood. In this section, we assume that h is the identity function, meaning the latent vector is equal to the input:

$$\tilde{u} \equiv u$$

2.1.1 Dataset

We sample the dataset using the model defined in 1.1. W_{xi} is set to a high value, and W_{xx} to a small value, for the observation to carry more information from the input than the hidden state. This will prevent the particle filter to simply learn a good posterior. Σ_x and Σ_y are set to small values in order to keep the model's dependency on input and hidden state, while still preventing a traditional training from obtaining zero loss. We use $T = 50$ and generate $m = 100$ samples.

2.1.2 Pre training

Pre training is implemented as described in Section 1.6.1. In our experimentation, we were able to recover the simulation parameters, although the model does not require to be identifiable, as demonstrated in the next simulation plan. During prediction, the model matches our simulated dataset to some extent. The general trend were correctly followed, while some of the most extremal points could not be reached, as this simple model doesn't account for model or observation noise.

2.1.3 Finetuning

We now freeze the model's parameters, except for W_y, b_y and Σ_y. Σ_x is set to a small value. We maximize the log likelihood as previously described in the

model definition. We found W_y and b_y to vary during training, but eventually reached the original values at convergence. We were also able to recover the original value of Σ_y.

2.1.4 Training from scratch

As a comparison, we trained the model by maximizing the log likelihood without any prior trainings. Weights are not frozen. We found convergence to be much harder to achieve than both previous methods, but we were still able to recover the value of Σ_y.

2.2 Finetuning advanced models

In this section, we reiterate the experiment of the previous section while considering a model where h is an arbitrary non linear function, such as a neural network. We use the same dataset. Similarly, the static model is able to capture trends, without being able to match all points due to observation noise. The Bayesian model is able to take advantage of the retrain, and recover the observation noise while only slightly straying from the original values of W_y and b_y.

2.3 Real data

This section addresses the training protocol on real data. We follow the same steps as for synthetic data:

1. A training step expected to reach a satisfactory set of parameters for the encoder h and the state model g_θ.
2. A finetuning step, during which only W_y, b_y and Σ_y are learned via Fisher's identity, for a set value of Σ_x.