Deep Filtering with DNN, CNN and RNN

Bin Xie¹, Qing Zhang²

1. School of Mathematics and Statistics, Hubei Minzu University, Enshi, Hubei 445000, P. R. China
   E-mail: xiebinmm@gmail.com
2. Mathematics Department, University of Georgia, Athens, GA 30602, USA
   E-mail: qz@uga.edu

Abstract: This paper is about a deep learning approach for general filtering. The idea is to train a neural network with Monte Carlo samples generated from a nominal dynamic model. Then the network weights are applied to Monte Carlo samples from an actual dynamic model. The main focus of this paper is on the deep filters with three major neural network architectures, Dense Neural Network (DNN), Convolution Neural Network (CNN), and Recurrent Neural Network (RNN). Our deep filter compares favorably to the traditional Kalman filter in linear cases and outperforms the extended Kalman filter in nonlinear cases. Then a switching model with jumps is studied to show the adaptiveness and the power of our deep filtering. Among the three major neural networks, CNN outperforms the others on average. The RNN does not seem to be suitable for the filtering problem. One advantage of the deep filter is its robustness when the nominal model and actual model differ. The other advantage of deep filtering is real data can be used directly to train the deep neural network. Therefore, model calibration can be by-passed altogether.

Key Words: Deep Filtering, Neural Network, Switching Model

1 Introduction

This paper discusses a deep learning approach to filtering using various neural networks (NNs). Filtering involves state estimation in systems that are partially observable, with widespread applications in engineering and finance. Traditional methods typically rely on least squares estimators and assume Gaussian distributions. The main challenge in filtering is to determine the conditional mean of the state based on observations up to a specific time $n$. The most well-known solution for linear models is the Kalman filter (KF), while the extended Kalman filter (EKF) offers effective approximations for certain nonlinear models. For further background and technical details, readers are directed to Anderson and Moore [1].

Key classical results in nonlinear filtering include Duncan [4], which addresses conditional densities for diffusion processes; Mortensen [13], focusing on the most probable trajectories; Kushner [10], deriving nonlinear filtering equations; and Zakai [21], dealing with equations of unnormalized conditional expectations.

Building on these classical works, significant progress has been made over the past decades. Those advancements include hybrid filtering approaches, as explored by Hijab [8] with an unknown constant, Zhang [22] with small observation noise, and Miller and Runggaldier [12] focusing on Markovian jump times. Blom and Bar-Shalom [2] have contributed to discrete-time hybrid models and the Interactive Multiple Model algorithm. Furthermore, Dufour et al. [5, 6] and Dufour and Elliott [7] have developed models incorporating regime switching. Additional advancements along this line can be found in Zhang’s later works [22–24]. Despite these important advances, the computation of filtering remains a daunting task, particularly for nonlinear filtering, where there is still a need for feasible and efficient schemes to address the high computational complexity associated with infinite dimensionality. Efforts continue to focus on developing computable approximation schemes.

Recently, Wang et al. [18] developed a deep learning framework for general filtering based on partially observed systems. The approach involves training a deep neural network using Monte Carlo samples generated from the system, with the observation process serving as inputs and the states from the Monte Carlo samples as targets. The objective is to minimize a least squares loss function between the target and the calculated output, enabling the optimization of weight vectors that are then applied to a new set of Monte Carlo samples from the actual dynamic model. This method, termed a deep filtering (DF), has demonstrated adaptiveness, robustness, and effectiveness in computational experiments with linear, nonlinear, and switching models. A significant advantage of the DF is its robustness in scenarios where the nominal model differs from the actual model. Additionally, deep filtering allows for the direct use of real-world data to train the neural network, eliminating the need for model calibration in practical applications.

Romero et al. [16] introduced a deep learning method to remove baseline wander from electrocardiogram signals, employing a fully convolutional network with multi-kernel linear and non-linear filters, which significantly enhances diagnostic accuracy by preserving signal integrity and effectively reducing noise. Schroter et al. [17] developed a deep learning-based speech enhancement framework utilizing CNNs for full-band audio processing, achieving substantial noise reduction and superior performance across various frequency resolutions and latency conditions. Qian et al. [15] introduced a deep learning framework for nonlinear filtering that manages random switching diffusions by adaptively approximating neural network weights and learn-

This research was supported in part by the Simons Foundation (849576).
ing rates, demonstrating its effectiveness with complex dynamic systems.

We note that the experiments reported by Wang et al. [18], Romero et al. [16] and Schrotter et al. [17] were solely conducted using DNNs or CNNs. Exploring how these filtering schemes perform with different types of neural networks would be intriguing. The purpose of this paper is to expand our investigation of deep filtering using DNNs, CNNs, and RNNs, and to compare their efficacy. We examine both linear and non-linear systems and evaluate the performance of the deep filter across these various neural networks. Our results indicate that the deep filter generally performs comparably to the Kalman filter in linear scenarios and surpasses the extended Kalman filter in non-linear cases. We also investigate a switching model with jumps to demonstrate the adaptiveness of the deep filtering technique. Among the three major neural network types, CNNs tend to outperform the others on average, while RNNs appear less suited for filtering tasks.

The structure of this paper is organized as follows: Section 2 details our methodology and describes the various neural networks under consideration. Section 3 presents the results of our computational experiments. Finally, Section 4 concludes the paper with final remarks and observations.

2 Methodology

Let \( x_n \in \mathbb{R}^{m_1} \) denote the state process with system equation

\[
x_{n+1} = f_n(x_n, u_n), \quad x_0 = x, \quad n = 0, 1, 2, \ldots, \quad (2.1)
\]

for some suitable functions \( f_n : \mathbb{R}^{m_1} \times \mathbb{R}^{l_1} \rightarrow \mathbb{R}^{m_1} \) and system noise \( \{u_n\} \) with \( u_n \in \mathbb{R}^{l_1} \). A function of \( x_n \) can be observed with possible noise corruption. In particular, the observation process \( y_n \in \mathbb{R}^{m_2} \) is given by

\[
y_n = h_n(x_n, v_n), \quad (2.2)
\]

with noise \( \{v_n\}, v_n \in \mathbb{R}^{l_2} \), and \( h_n : \mathbb{R}^{m_1} \times \mathbb{R}^{l_2} \rightarrow \mathbb{R}^{m_2} \).

Deep Filter. The DF for (2.1) and (2.2) can be given as follows: Let \( N_{seed} \) denote the number of training sample paths and let \( n_0 \) denote the training window for each sample path. For any fixed \( \kappa = n_0, \ldots, N \) with a fixed \( \omega \), we take \( \{y_{\kappa}(\omega), y_{\kappa-1}(\omega), \ldots, y_{n_0+n_0+1}(\omega)\} \) as the input vector to the neural network and \( x_n(\omega) \) as the target. In what follows, we shall suppress the \( \omega \) dependence. Fix \( x_0, \) let \( \xi_{\ell} \) denote the neural network output at iteration \( \ell \), which depends on the parameter \( \theta \). Our goal is to find an NN weight \( \theta \in \mathbb{R}^{m_3} \) so as to minimize the loss function

\[
L(\theta) = \frac{1}{2} E|\xi_{\ell} - x_{\kappa}|^2. \quad (2.3)
\]

We use the stochastic gradient descent (SGD) method to find the NN parameters \( \theta \) by minimizing the loss function with learning rate \( \gamma \in (0, 1) \).

\[
\theta_{n+1} = \theta_n - \gamma \frac{\partial L(\theta_n)}{\partial \theta}. \quad (2.4)
\]

Then these weights \( \theta \) are used to out-of-sample data \( \{\omega\} \) with the actual observation \( y_n(\omega) \) as inputs in the subsequent testing stage which leads to neural network output \( \tilde{x}_n(\omega) \). Such a state estimation procedure is called deep filtering.

2.1 Network Specifications

Typical neural networks for deep learning applications include dense neural networks (DNN), convolutional neural networks (CNN), and recurrent neural networks (RNN).

In this paper, we first consider the fully connected DNNs. As in Wang et al. [18], we use the DNN with 5 hidden layers (each with 5 neurons). We also use the sigmoid activation function for all hidden layers and the simple activation for the output layer. Figure 1 shows the DNN structure used in this paper.

![DNN Structure](image)

A CNN is another architecture of NNs and it allows to model both time and space correlations in multivariate signals. A CNN utilizes convolution operation to seize the local features with several kernel filters. CNNs are widely used in image classification, image recognition, and computer vision. Figure 2 shows the CNN structure used in this paper.

An RNN is the other architecture of neural network where connections between nodes form a directed graph.
3 Numerical Experiments

Following the system equations (2.1) and (2.2) for \( (x_n, y_n) \), we generate \( N_{\text{seed}} = 5000 \) Monte Carlo sample paths (in-sample) for the neural network training. Then we generate a separate set of \( N_{\text{seed}} = 5000 \) samples (out-of-sample) to test the results. We take the time horizon \( N = 1000 \) and use the window size \( n_0 = 50 \) of input to train the network. We use the stochastic gradient decent (SGD) algorithm with learning rate \( \gamma = 0.1 \).

The following relative error is used in this paper. Given vectors \( \xi^1(\omega) = (\xi^1_{n_0}(\omega), \ldots, \xi^1_N(\omega)) \) and \( \xi^2(\omega) = (\xi^2_{n_0}(\omega), \ldots, \xi^2_N(\omega)) \), define

\[
\|\xi^1 - \xi^2\| = \frac{\sum_{n=n_0}^{N} \sum_{m=1}^{N_{\text{seed}}} |\xi^1_{n}(\omega_m) - \xi^2_{n}(\omega_m)|}{N_{\text{seed}}(N - n_0 + 1)\Sigma},
\]

where

\[
\Sigma = \frac{\sum_{n=n_0}^{N} \sum_{m=1}^{N_{\text{seed}}} (|\xi^1_{n}(\omega_m)| + |\xi^2_{n}(\omega_m)|)}{N_{\text{seed}}(N - n_0 + 1)}.
\]

In this paper, we work with linear, non-linear, and switching models to test deep filtering schemes. We start with linear systems.

Linear Systems. We consider the linear system

\[
\begin{align*}
    x_{n+1} &= F_n x_n + G_n u_n, \quad x_0 = x, \\
    y_n &= H_n x_n + v_n, \quad n = 0, 1, 2, \ldots,
\end{align*}
\]

for some matrices \( F_n, G_n, \) and \( H_n \) of appropriate dimensions. Here \( u_n \) and \( v_n \) are independent random vectors having Gaussian distributions with mean zero and \( E(u_n u_n^T) = \delta_{nl} I, E(v_n v_n^T) = \delta_{nl} I \) for \( u, l = 0, 1, 2, \ldots \), where \( \delta_{nl} = 1 \) if \( n = l \) and 0 otherwise.

1D Linear Model. In particular, the one-dimensional system is given as follows:

\[
\begin{align*}
    x_{n+1} &= (1 + 0.1\eta)x_n + \sqrt{\eta} \sigma u_n, \quad x_0 = 1, \\
    y_n &= x_n + \sigma_0 v_n, \quad n = 0, 1, 2, \ldots,
\end{align*}
\]

with \( u_n \) and \( v_n \) which are independent Gaussian \( N(0, 1) \) random variables. We take \( \sigma = 0.7, \sigma_0 = 0.5 \), and the step size \( \eta = 0.005 \).

2D Linear Model. In two-dimensional setting, we consider the system

\[
\begin{align*}
    x_{n+1} &= (1 + 0.1\eta F_n^0)x_n + \sqrt{\eta} G_n^0 u_n, \quad x_0 = [1, 1], \\
    y_n &= H_n^0 x_n + \sigma_0 v_n, \quad n = 0, 1, 2, \ldots,
\end{align*}
\]

with \( u_n \) and \( v_n \) being independent 2D normal random variables.

We take \( \eta = 0.005, \sigma_0 = 0.5, G_n^0 = I, H_n^0 = I \) and \( F_n^0 = \left( \begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix} \right) \).

With these parameters, the corresponding numerical relative errors are included in Table 1. The DF with neural networks perform comparable to the KF except the one with the RNN in two-dimensional case. The DF with the CNN produces a little better relative errors.

| \( \text{Model} \) | \( \text{KF} \) | \( \text{DNN} \) | \( \text{CNN} \) | \( \text{RNN} \) |
|----------------|----------|----------|----------|----------|
| 1D Linear      | 4.17     | 4.51     | 4.17     | 4.50     |
| 2D Linear      | 4.45     | 4.43     | 4.24     | 8.26     |

Robustness of Deep Filtering. In this section, we examine the robustness of deep filtering. We consider separately the nominal model and the actual model. A nominal model (NM) is an estimated model. It deviates from real data for different applications. In this paper,
it is used to train our NNs, i.e., a selected mathematical model is used to generate Monte Carlo sample paths to train the NN. The coefficients of the mathematical model are also used in Kalman filtering equations for comparison.

An actual model (AM), on the other hand, is about the simulated (Monte Carlo based) environment. It is used in this paper for testing purposes. In real world applications, the observation process is the actual process obtained from real physical process. To test the model robustness, we consider the case when the AM’s observation noise differs from the AM’s observation noise.

To examine the robustness of the DF, we fix $\sigma_0 = \sigma_0^{NM} = 2$ and vary $\sigma_0 = \sigma_0^{AM}$. The relative errors for the one-dimensional model case are reported in Table 2. Again, the DF with each NN perform close to the KF. The DF with the CNN does a little better while the one with the RNN falls behind.

Table 2: Relative Errors (in %) for Different $\sigma_0^{AM}$ (1D Linear Models) with $\sigma_0 = \sigma_0^{NM} = 2$

| $\sigma_0^{AM}$ | 0.1 | 0.5 | 1 | 1.5 | 2 | 2.5 |
|-----------------|-----|-----|---|-----|---|-----|
| KF              | 6.91| 6.28| 6.93| 7.85| 8.94| 10.13|
| DNN 1D Lin      | 6.48| 6.57| 7.09| 8.25| 9.88| 11.74|
| CNN 1D Lin      | 6.69| 5.88| 6.59| 7.75| 9.19| 10.80|
| RNN 1D Lin      | 6.11| 6.32| 7.32| 9.33| 12.12| 15.38|

The results for the 2D linear model with $F_0^0 = \frac{1}{2}$ are provided in Table 3. The robustness of the DF is comparable to that of the KF.

Table 3: Relative Errors (in %) for Different $\sigma_0^{AM}$ (2D Linear Models) with $\sigma_0 = \sigma_0^{NM} = 2$

| $\sigma_0^{AM}$ | 0.1 | 0.5 | 1 | 1.5 | 2 | 2.5 |
|-----------------|-----|-----|---|-----|---|-----|
| KF              | 6.87| 6.97| 7.26| 7.73| 8.35| 9.07|
| DNN 2D Lin      | 5.61| 5.82| 6.45| 7.44| 8.68| 10.07|
| CNN 2D Lin      | 8.49| 7.55| 7.09| 7.49| 8.48| 9.81|
| RNN 2D Lin      | 6.60| 6.63| 6.98| 7.76| 8.89| 10.26|

Non-Linear Systems. We consider nonlinear models and compare the results of three kinds of neural networks with the corresponding extended Kalman filter.

1D Non-Linear Model. First, we consider the following 1D model:

$$
\begin{align*}
    x_{n+1} &= x_n + \eta \sin(5x_n) + \sqrt{\eta} \sigma u_n, \quad x_0 = 1, \\
    y_n &= x_n + \sigma_0 v_n, \quad n = 0, 1, 2, \ldots,
\end{align*}
$$

(3.8)

with $u_n$ and $v_n$ which are independent Gaussian $N(0, 1)$ random variables.

2D Non-Linear Model. The 2D model is given by:

$$
\begin{align*}
    x_{n+1} &= x_n + \eta \sin(5F_0^0 x_n) + \sqrt{\eta} \sigma_0 u_n, \quad x_0 = [1, 1]^T, \\
    y_n &= H^0 x_n + \sigma_0 v_n, \quad n = 0, 1, 2, \ldots,
\end{align*}
$$

(3.9)

with $u_n$ and $v_n$ which are independent 2D normal random variables. We take $\eta = 0.005$, $\sigma_0 = 0.5$, and $F_0^0 = \frac{1}{2}$.

In Table 4, it can be seen that the DF with each NN outperforms the EKF in both the one-dimensional and two dimensional cases.

The corresponding robustness comparisons are reported in Tables 5 and 6. In both the one and two dimensional cases, the DF with each NN shows similar robustness as the EKF except with the RNN which under performs the rest when $\sigma_0^{AM} > 1.5$ for 1D case.

In Figures 4-7, a sample of $x_n$ in the one-dimensional system with $\sigma_0 = \sigma_0^{NM} = 2$ is given along with the corresponding DF $\bar{x}_n$ with DNN, CNN, and RNN. According to these pictures, it appears that the DF with CNN provides smoother path. This is consistent with the results included in Table 4.

Fig. 4: A Sample Path of $x_n$ in 1D Non-Linear Model

Fig. 5: The Corresponding DF $\bar{x}_n$ with DNN

Switching Models. Next, we consider a switching model with jumps and apply the DF to these models. For the jump case, neither the KF nor the EKF can be applied

Table 4: Relative Errors (in %) of Nominal Model for Non-Linear Models

|               | EKF | DNN | CNN | RNN |
|---------------|-----|-----|-----|-----|
| 1D Non-Linear | 10.04| 6.16| 5.68| 6.40|
| 2D Non-Linear | 8.87 | 5.27| 5.13| 5.37|

Table 5: Relative Errors (in %) for Different $\sigma_0^{AM}$ (Non-Linear Models) with $\sigma_0 = \sigma_0^{AM} = 2$

| $\sigma_0^{AM}$ | 0.1 | 0.5 | 1 | 1.5 | 2 | 2.5 |
|-----------------|-----|-----|---|-----|---|-----|
| EKF             | 8.80| 9.05| 9.75| 10.76| 11.97| 13.33|
| DNN 1D Lin      | 9.54| 9.47| 9.87| 11.29| 13.51| 16.04|
| CNN 1D Lin      | 8.27| 8.40| 9.26| 10.77| 12.68| 14.86|
| RNN 1D Lin      | 8.54| 8.81| 10.24| 13.19| 17.13| 21.51|

Table 6: Relative Errors (in %) for Different $\sigma_0^{AM}$ (Non-Linear Models) with $\sigma_0 = \sigma_0^{NM} = 2$

| $\sigma_0^{AM}$ | 0.1 | 0.5 | 1 | 1.5 | 2 | 2.5 |
|-----------------|-----|-----|---|-----|---|-----|
| EKF             | 9.19| 9.34| 9.78| 10.44| 11.28| 12.25|
| DNN 2D Lin      | 7.08| 7.31| 8.07| 9.32| 10.95| 12.81|
| CNN 2D Lin      | 11.12| 9.84| 9.12| 9.53| 10.74| 12.40|
| RNN 2D Lin      | 7.82| 7.88| 8.53| 9.34| 10.74| 12.43|
due to the presence of the switching process and lack of recursive dynamic equations.

1D Switching Model. As in Wang et al.[18] we consider the following model:

\[
\begin{align*}
    x_n &= \sin(n\eta \alpha_n + \sigma u_n), \quad n = 0, 1, 2, \ldots, \\
    y_n &= x_n + \sigma_0 u_n.
\end{align*}
\]

(3.10)

We take \( \alpha(t) \in \{1, 2\} \) to be a continuous-time Markov chain with generator \( Q = \begin{pmatrix} -2 & 2 \\ 2 & -2 \end{pmatrix} \). Using step size \( \eta = 0.005 \) to discretize \( \alpha(t) \) to get \( \alpha_n = \alpha(n\eta) \). We also take \( \sigma = 0.1 \) and \( \sigma_0 = 0.3 \).

2D Switching Model. We consider the following 2D switching system:

\[
\begin{align*}
    x_n &= \sin(n\eta \alpha_n + \sigma u_n), \quad n = 0, 1, 2, \ldots, \\
    \tilde{x}_n &= \cos(n\eta \alpha_n + \sigma u_n), \quad n = 0, 1, 2, \ldots,
\end{align*}
\]

The observation

\[
\begin{pmatrix}
    y_n \\
    y_n
\end{pmatrix} = H \begin{pmatrix}
    x_n \\
    \tilde{x}_n
\end{pmatrix} + \sigma_0 u_n,
\]

(3.11)

with \( u_n \) as 2D white noise. We take \( \eta = 0.005, \sigma = 0.3, \) and \( H = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \). We choose nominal model \( \sigma_0^{NM} = 2 \) for the NNs training.

In these examples, the KF (or the ETF) would not apply due to lack of recursive dynamic systems. We only compare the robustness of the DF with various NNs. In Table 7, the DF with the DNN exhibits better robustness while the one with the RNN underperforms when \( \sigma_0^{AM} > 1.5 \).

Table 7: Relative Errors (in %) for Different \( \sigma_0^{AM} \) (1D Switching Model) with \( \sigma_0 = \sigma_0^{NM} = 2 \)

| \( \sigma_0^{AM} \) | 0.1 | 0.5 | 1 | 1.5 | 2 | 2.5 |
|-------------------|-----|-----|---|-----|---|-----|
| DNN 1D Swt        | 12.67 | 12.64 | 12.92 | 14.28 | 16.79 | 19.92 |
| CNN 1D Swt        | 18.01 | 17.84 | 16.90 | 17.74 | 19.80 | 22.52 |
| RNN 1D Swt        | 13.81 | 14.23 | 16.45 | 21.31 | 27.61 | 33.92 |

The robustness of the DF under the two-dimensional switching model is given in Table 8. In this case, the CNN shows stability and outperforms while the RNN drags much behind for bigger \( \sigma_0^{AM} \).

Table 8: Relative Errors (in %) for Different \( \sigma_0^{AM} \) (2D Switching Models) with \( \sigma_0 = \sigma_0^{NM} = 2 \)

| \( \sigma_0^{AM} \) | 0.1 | 0.5 | 1 | 1.5 | 2 | 2.5 |
|-------------------|-----|-----|---|-----|---|-----|
| DNN 2D Swt        | 37.24 | 36.99 | 36.92 | 37.83 | 39.29 | 40.75 |
| CNN 2D Swt        | 16.38 | 15.38 | 15.48 | 17.16 | 19.68 | 22.47 |
| RNN 2D Swt        | 12.01 | 11.94 | 11.85 | 12.66 | 17.27 | 19.37 |

In Figures 8-11, a sample of \( x_n \) in the one-dimensional switching model with \( \sigma_0^{NM} = \sigma_0^{AM} = 2 \) is plotted along with the corresponding DF \( \tilde{x}_n \) with DNN, CNN, and RNN. In this case, the DNN appears doing better than others with jumps capture.

Fig. 8: A Sample Path of \( x_n \) in the 1D Switching Model

Fig. 9: The Corresponding DF \( \tilde{x}_n \) with DNN

Fig. 10: The Corresponding DF \( \tilde{x}_n \) with CNN

Remark 1 Note that the RNN performs poorly when \( \sigma_0^{AM} = 2.5 \). This appears due to the LSTM’s overfitting feature in the 2D case. One way to reduce such overfitting is to stop early to avoid loss function blowups. By and large, the early stopping will halt the NN’s training by monitoring the loss function and exit when the loss value no longer improves in several consecutive epochs. We refer the reader to Jason [9] for further details. Using such an early stopping technique, the corresponding relative errors are given in Table 9.
Remark 2 In this paper, an iMac desktop with Intel Core i5 processor (3.2 GHz Quad-core, 24G DDR3) with Keras library of Python 3 was used for all numerical experiments. In Table 10, the CPU time for one step calculation with various network architectures are given. The DNN and CNN consume similar amount of time, while the RNN runs much longer.

We would like to comment that the network training is the most time consuming part. Nevertheless, this will not affect much in real time filtering because the training can be done offline.

4 Conclusion

In this paper, we have studied the deep filter with different neural network architectures. The DF performs similarly as the Kalman filter in linear models and outperforms the KF in nonlinear models. In addition, the CNN outperforms the DNN and RNN on average. The DF is applicable to general dynamic systems and it does not require any model specifications. To implement the DF, real data can be used directly to train the deep neural network. Therefore, model calibration can be eliminated altogether. Nevertheless, the reasons for varying performance due to different structures remain unclear and requires further investigation.

References

[1] B.D.O Anderson and J.B. Moore, Optimal Filtering, Prentice-Hall, Englewood Cliffs, NJ, 1979.
[2] H. A. P. Blom and Y. Bar-Shalom, The interacting multiple model algorithm for systems with Markovian switching coefficients, IEEE Trans. Automat. Contr., vol. 33, pp. 780-783, 1988.
[3] F. Chollet, Deep learning with Python, Manning New York, 2018.
[4] T.E. Duncan, Probability densities for diffusion processes with applications to nonlinear filtering theory and detection theory, PhD Diss., Stanford Univ. 1967.
[5] F. Dufour, P. Bertrand, and R. J. Elliott, Filtering for linear systems with jump parameters and high signal-to-noise ratio, Proc. 13th IFAC World Congress, San Francisco, CA, pp. 445-450, 1996.
[6] F. Dufour, P. Bertrand, and R. J. Elliott, Optimal filtering and control of linear systems with Markov perturbations, Proc. 35th IEEE CDC, Kobe, Japan, pp. 4065-4070, 1996.
[7] F. Dufour and R. J. Elliott, Adaptive control of linear systems with Markov perturbations, IEEE Trans. Automat. Contr., vol. 43, pp. 351-372, 1997.
[8] O. Hijab, The adaptive LQG problem - Part I, IEEE, Trans. Automat. Contr., vol. 28, pp. 171-178, 1983.
[9] B. Jason, Better deep learning: train faster, reduce over-fitting, and make better predictions, Machine Learning Mastery, 2018.
[10] H.J. Kushner, On the differential equations satisfied by conditional probability densities of Markov processes, with applications, J. SIAM Control, 2, vol. 2, pp. 106-119, 1964.
[11] H.J. Kushner and G. Yin, Stochastic Approximation Algorithms and Applications, Springer, New York, 1997.
[12] B. M. Miller and W. J. Runggaldier, Kalman filtering for linear systems with coefficients driven by a hidden Markov jump process, Systems & Control Lett., vol. 31, pp. 93-102, 1997.
[13] R. E. Mortensen, Maximum-likelihood recursive nonlinear filtering, J. Optim. Theory Appl., vol. 2, pp. 386-394, 1968.
[14] M. Nielsen. Neural networks and deep learning, michaelnielsen.org.
[15] H. Qian, G. Yin and Q. Zhang, Deep Filtering With Adaptive Learning Rates, IEEE Trans. Autom. Control, vol. 68, no. 6, pp. 3285-3299, 2023.
[16] Francisco P. Romero, David C. Piñol, Carlos R. Vázquez-Seisdedos, DeepFilter: An ECG baseline wander removal filter using deep learning techniques, Biomed. Signal Process. Control,Volume 70, 2021.
[17] H. Schroter, A. N. Escalante-B, T. Rosenkranz and A. Maier, Deepfilternet: A Low Complexity Speech Enhancement Framework for Full-Band Audio Based On Deep Filtering, ICASSP 2022 - 2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp. 7407-7411, 2022.
[18] L.Y. Wang, G. Yin, and Q. Zhang, Deep filtering, Commun. Inf. Syst., Vol. 21, pp. 651-667, 2021.
[19] G. Welch and G. Bishop, An introduction to the Kalman filter, Chapel Hill, NC, 1995.
[20] G. Yin and Q. Zhang, Continuous-Time Markov Chains and Applications: A Two-Time Scale Approach, Second Edition, Springer, New York, 2013.
[21] M. Zakai, On the optimal filtering of diffusion processes, Zeitschrift für Wahrscheinlichkeitstheorie und Verwandte Gebiete, Vol 11, pp. 230-243, 1969.
[22] Q. Zhang, Nonlinear filtering and control of a switching diffusion with small observation noise, SIAM J. Contr. Optimi., vol. 36, pp. 1738-1768, 1998.
[23] Q. Zhang, Optimal filtering of discrete-time hybrid systems, J. Optim. Theory Appl., Vol. 100, pp. 123-144, 1999.
[24] Q. Zhang, Hybrid filtering for linear systems with non-Gaussian disturbances, IEEE Trans. Autom. Control, Vol. 45, pp. 50-61, 2000.

Fig. 11: The Corresponding DF $\hat{x}_n$ with RNN

Table 9: Relative Errors (in %) of Early Stopping for Different $\sigma_{AM}^0$ (Switching Models) with $\sigma_0 = \sigma_{AM}^0 = 2$

| $\%$ | 0.1 | 0.5 | 1 | 1.5 | 2 | 2.5 |
|------|-----|-----|---|-----|---|-----|
| RNN 2D Swt | 12.06 | 11.89 | 11.81 | 12.56 | 14.54 | 17.73 |

Table 10: Running Time (in seconds)

| Time | 1D Lin | 1D NL | 1D Swt | 2D Lin | 2D NL | 2D Swt |
|------|--------|-------|--------|--------|-------|--------|
| DNN  | 11.0   | 10.5  | 11.1   | 14.1   | 13.5  | 14.7   |
| CNN  | 12.5   | 12.1  | 12.3   | 19.5   | 18.6  | 18.8   |
| RNN  | 17.5   | 17.5  | 10.6   | 183.5  | 175.0 | 182.2  |

Authorized licensed use limited to the terms of the applicable license agreement with IEEE. Restrictions apply.