CS230

Deep RNNs for Non-Linear State Estimation

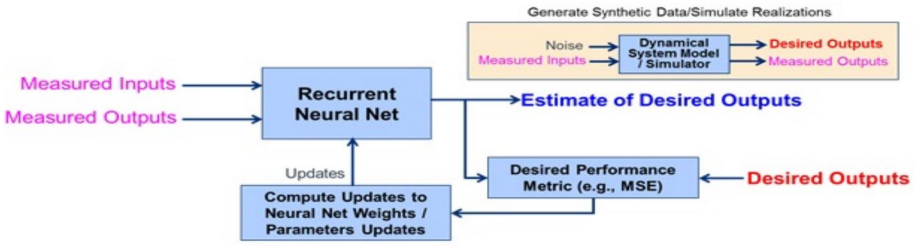
Abstract

Estimating the state of a dynamical system from noise-corrupted indirect measurements is a fundamental problem in many applications, such as navigation, process control, or time series forecasting. This fundamental problem addressed is by Filtering or State Estimation, which employs techniques such as the Extended Kalman (EKF) or Particle Filter (PF.) This paper explores applying a Neural Filter, which is a deep RNN composed of multi-layer LSTM cells as a solution to solving a benchmark non-linear time-series. In contrast to classical techniques such as the EKF and PF techniques, a Neural Filter is applicable to cases where no analytic model of the system is available, and only measured data from experiments is available. We found that a simple deep RNN architecture is able to produce superior RMSE performance on this problem, implying that neural filters closely compare to state-of-the-art filtering techniques.

1 Introduction

This project proposes considering a neural network-based (Neural) Filter as a solution to optimally estimate any non-linear dynamical system. Limitations of current state-of-the-art estimators, like the Particle Filter (PF), are not widely recognized but are mentioned in literature. In general, for a given non-linear system it is difficult to predict which of these estimators will produce the best results. For example the Extended Kalman Filter (EKF) is widely used but is recognized to have degraded accuracy or lose stability for certain non-linear problems (Reference 4.) Furthermore, unlike these traditional techniques, a Neural Filter is applicable to cases where no analytic model of the system is available, and only measured data from experiments is available. Lastly, since the computational intensive portion of the Neural Filter is done offline (training), Neural Filters have the potential to be more computationally efficient in a real-time setting in contrast to a particle filter which is very computationally expensive in a real-time implementation.

Figure 1: Employing a Recurrent Neural Network as "Neural Filter" to solve the state estimate problem. Dynamical System Model/ Simulator is employed to generate synthetic data for training purposes. In contrast to Figure 1 a trained RNN becomes the filter/observer for the system.



CS230: Deep Learning, Winter 2018, Stanford University, CA. (LateX template borrowed from NIPS 2017.)

This project explores training a deep Recurrent Neural Network constructed of stacked LSTM cells (a Neural Filter) to estimate the mapping from measurements to state estimates. The main assumption in deep learning is availability of large amounts of data for effective training - this is easily satisfied since relevant data will be generated by simulation using the dynamical system model. This proposal will consider a benchmark non-linear dynamical system (proposed in Reference 4), which is a one-state problem with a bimodal prior. The inputs to the Neural Filter will be the measured inputs and measured outputs of the dynamical system, and the prediction will be estimates of the desired output of the dynamical system. Since desired outputs are generated in the simulation process, we can visualize the performance of the Neural Filter by plotting the filter-produced estimated desired output against the simulation desired output over the time series of data. Evaluation will be based on minimization of a desired performance metrics (e.g. mean square error) and compared against state-of-the-art filters (e.g. particle filters.)

2 Related work

This project builds off of a published comparison analysis titled "The Blind Tricyclist Problem and a Comparison Study of Nonlinear Filters- A Challenging Benchmark for Evaluation Nonlinear Estimation Methods" (Reference 4.) This paper considers two challenging non-linear dynamical system in which traditional approximate techniques can perform poorly. This project focused on the second of the two benchmarks. For this particular system, the Particle Filter, which is considered "state-of-the-art" for many filtering problems achieves the lowest error. Yet, the results of this comparison suggest that no single filtering technique dominates as a sure solution to all non-linear estimation problems. The solution proposed in this project, a deep recurrent neural network, aims to be a more widely applicable and consistent solution to challenging non-linear dynamical systems.

3 Dataset and Features

The example considered was a nonlinear time series, outlined in equations 1-3 below, which is widely used for bench-marking numerical filtering techniques. Model parameters depicted in the equations below are chosen based on Reference 4 (optimal RMSE ~ 4.68.)

$$x_t = \frac{x_{t-1}}{2} + 25\frac{x_{t-1}}{{x_{t-1}}^2} + 0.8cos(1.2t) + w_t \tag{1}$$

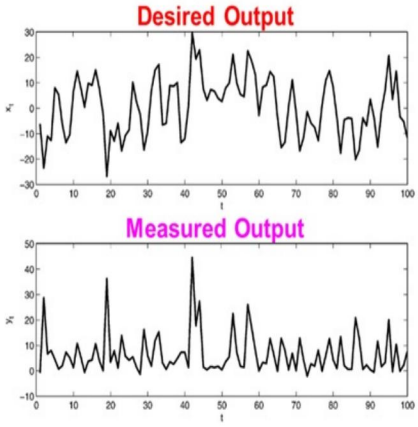
$$y_t = \frac{{x_t}^2}{20} + v_t \tag{2}$$

$$x_o \sim N(0, 25), w_o \sim N(0, 10), v_t \sim N(0, 1) \tag{3}$$

The data required to train and validate a neural filter solution can be simulated using the dynamical system above. Knowing that w_o, v_t, x_o are all normally distributed random variables, we simulated the data required to train the network, which is measured outputs (y_t) and desired outputs (x_t), by sampling from the normal distribution (using np.random.randn) with the appropriate variance outlined in the equations above.

For a single time step t, the measured outputs (y_t) represent our feature vector and the desired outputs (x_t) represent the label or ground truth at time t. This process of constructing measured outputs (y_t) and desired outputs (x_t) from sampling of the normal distribution is repeated for the series length (T), which represents the length of the time series which will be estimated by the sequence model.

Figure 2: Example data of measured outputs (y_t),representing the feature vector, and the desired outputs (x_t), representing the label, drawn from the nonlinear time series model with series length of 100.



For each epoch during training, a batch of fixed sample size is generated via simulation. Each observation of a given batch is a series of length T of measured outputs (y_t), which serve as the features, and desired outputs (x_t), the labels. We selected a nominal length of $T = 100$ time steps for this analysis. The network is trained on many more observations of the systems than what it requires for validation. Nominally, training can take $5000 - 25000$ epochs to train depending on the size of the stacked RNN and hyperparameter selection. We validated on approximately 10 epochs of simulated data, which is supported by the fact the RMSE of the model is fairly stable in the validation results.

4 Methods

The approach of solving the nonlinear dynamical system involves building a sequence to sequence model which takes an input time series of length T of measured outputs and constructing a sequence of estimated outputs of equal length T. As in classical estimation techniques, like the KF, the prior state of the system is important in the current state's prediction. Hence, recurrent neural networks are a natural approach, in particular the Long Term Short Memory (LSTM) cell.

$$\tilde{c}^{<t>} = tanh(W_c[a^{<t-1>}, x^{<t>}] + b_c) \tag{4}$$

$$\Gamma_u = \sigma(W_u[a^{<t-1>}, x^{<t>}] + b_u) \tag{5}$$

$$\Gamma_f = \sigma(W_f[a^{<t-1>}, x^{<t>}] + b_f) \tag{6}$$

$$\Gamma_o = \sigma(W_o[a^{<t-1>}, x^{<t>}] + b_o) \tag{7}$$

$$c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + \Gamma_f * c^{<t-1>} \tag{8}$$

$$a^{<t>} = \Gamma_o * tanh(c^{<t>}) \tag{9}$$

The LSTM cell is a way to incorporate long term dependencies and persistence of information over time into a neural network. Equations 4-9 depict the inner workings of an LSTM cell. The cell consists of 3 gates: $\Gamma_u, \Gamma_f, \Gamma_o$. In short, these gates have the ability to add and remove information to the cell's state, $c^{<t>}$. Γ_f governs how much of the prior state is forgotten, while Γ_u governs how much we update each state value. Lastly Γ_o controls the output of the cell. A deep RNN consists of stacking RNN cells, in our case LSTM cells. Figures 3 and 4 depict the how a sequence of stacked RNN cells constructs a multi-layer network.

Figure 3: Network architecture of stacking RNN cells, in our case LSTM cells, to create a deep RNN framework. This example depicts a network that consists of a 3-layer stacked RNN for each time step.

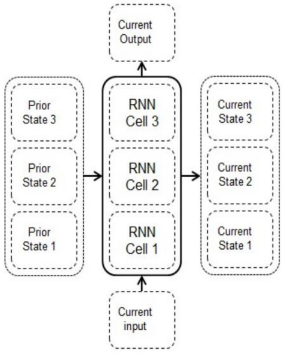
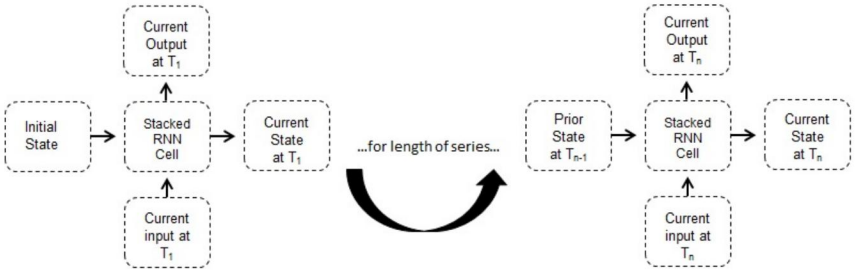


Figure 4: A network of stacked RNN cells estimating a sequence of length T_n. In this analysis T_n is the sequence length 100.



The loss function chosen was Root Mean Squared Error (RMSE, equation 10.) This is a natural choice since the problem statement requires estimation of the state vector which is continuously valued. In our analysis the length of the sequence T is 100 and $\hat{y}_i$ is the estimated desired output and y_i is the ground truth desired output. Mean Absolute Error (MAE) was also considered. However, RMSE is also the metric reported in the literature considered (Reference 4.) The Adam optimization method was selected for training the model and used the TensorFlow default values for hyperparameters except for the learning rate ($\beta_1 = 0.9, \beta_2 = 0.999$ and $\epsilon = 1e-8$.)

$$RMSE = \sqrt{\frac{\sum_{i=1}^{T} (\hat{y}_i - y_i)^2}{T}} \tag{10}$$

5 Experiments/Results/Discussion

We predominantly focused on tuning two hyperparameters for this study: learning rate and number of layers in each stacked LSTM cell cell. Tuning of other hyperparameters (batch size, etc) that were not explored in this project are discussed briefly in the final section. We first considered cases of a static learning rate throughout the training period and varied the learning rate from 1e-2 to 1e-6 and varied the number of LSTM cells from 2 to 4. The metric used to assess performance was RMSE. Since the problem is of the regression variety, RMSE is a natural choice for a metric.

From the results in Table 1, one can see that tuning the learning rate had a greater effect on decreasing RMSE than increasing the number of layers. The best results occurred with a static learning rate of 1e-4 and 3 layers, which resulted in a validation RMSE of 4.74. The transition from 3 layers to 4 did not decrease RMSE for any of the learning rates considered, so larger numbers of layers were

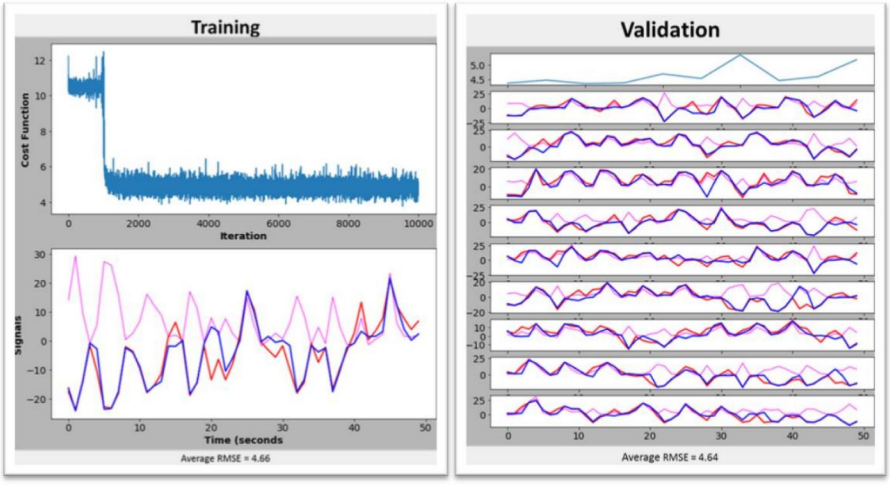
Learning Rate	2 Layers	3 Layers	4 Layers
1e-2	12.89/12.97	5.51/5.57	11.55/11.72
1e-4	5.15/5.11	4.66/4.74	5.17/5.33
1e-6	10.44/10.60	10.50/10.46	10.45/10.35

Table 1: RMSE results for deep LSTM network for static learning rate and variable numbers of layers. First number represents training RMSE averaged over the last 100 epochs. Second number is the average RMSE for 10 validation epochs.

not considered. Since the training and testing RMSE are quite close and the training RMSE was not significantly lower in any cases to the validation RMSE, over fitting did not appear to be a problem. Based on the loss function behavior overtime, we can observe that the loss functions becomes on average static, so the more likely scenario is the optimization process stalls in local minima. Thus, it makes sense consider a learning rate schedule which decreases the learning rate over the duration of the training period. Although the Adam optimizer itself has an adaptive nature to its optimization steps, we still chose to enforce and additional learning rate decay protocol.

We additionally considered an exponential learning rate decay with a staircase function, specifically we chose to to decay the learning rate every 1000 epochs with a base of 0.96. This approach produced an average RMSE of 4.67 in training average over the last 100 epoch, and an RMSE of 4.64 in 10 epochs of validation. Figure 5 depicts the relationship between Desired Outputs (the data labels, shown in red) and the Estimated Desired Outputs (output of the RNN, shown in blue), and the Measured Outputs (inputs to RNN, shown in pink.) By the end of training the RNN estimates the output signal very closely. The top subplot on the right shows the RMSE for the various steps in the series length. The 9 subplots below show the trained RNN's performance on 9 validation epochs. Over the entire series length the Estimated Tutputs (shown in blue) track closely with the Desired Outputs (shown in red).

Figure 5: Left: Training Result of 3-layer RNN, where number of epochs = 10000, series length = 100, batch size =10, with an exponential learning rate schedule with staircase step every 1000 epochs initialized at 1e3 with a base of 0.96. Right: Results of validation of 3-layer RNN on testing data. Average RMSE for series length is shown in the top graph. Shown below is Measured outputs (Pink), Desired Outputs (the data labels, shown in red) and the Estimated Desired Outputs (output of the RNN, shown in blue) for 9 epochs.



Comparing the results of our neural filter performance to the results published in Reference 4 shown in Table 2, the neural filter produces a much smaller RMSE than many of the traditional Kalman filtering techniques. The particle filter performance with a RMSE of 4.68 was the optimal baseline for this problem. The neural filter performed slightly better than the PF with a validation RMSE of 4.64.

Hence when sufficiently accurate measured outputs of a system are available, a neural filter is an alternative to other non-linear filtering techniques. Furthermore unlike traditional non-linear methods, where there is often some level of uncertainty which filter method will produce most optimal results, a neural filter can be seem as a "universal" non-linear estimator.

Performance metrics for filters	
Filter	RMSE
Extended Kalman Filter	21.53
Unscented Kalman Filter A	26.97
Particle Filter B	4.68
Neural Filter	4.64

Table 2: RMSE results for various filter techniques. Hyperparameter details for the Kalman and Particle Filters can be found in Reference 4.

6 Conclusion/Future Work

In summary, the results support the use of a Neural Filter as an alternative to traditional approximate filtering techniques currently used to solve non-linear state estimation problems. The RMSE produced by a Neural Filter on the bench mark problem considered was 4.64 as opposed to a RMSE of 4.68 for a Particle Filter solution. We found highest performing network consisted of a stacked LSTM architecture consisting of 3 layers, with an exponential learning rate decay schedule initialized at 10e-3, and a batch size 10.

Further analysis should include a finer tuning of the learning rate. In addition, a deeper exploration of options such as alternative stepped decreases or a cosine annealing of the learning rate over the training process may be beneficial to model performance. It is also worth exploring optimizing the batch size if more time and resources were invested in the future. In this paper, a static batch size of 10 was used. Similarly, tuning the additional hyperparmeters used in Adam, such as β_1, β_2 or ϵ, could improve model performance. Lastly, the approach chosen selected using stacked LSTM cells. An alternative approach which could be useful to consider in future work would be to employ stacked Gradient Recurrent Units (GRUs) instead of LSTMs. GRUs are less complex; they only contain a single gate, while the LSTM consists of three gates. Hence, a model consisting of GRUs will have fewer learn-able parameters and may be less intensive to train than one consisting of LSTMs.

7 Contributions

This project was completed by Catherine Watkins. Special thanks to my group's chief scientist at JHU APL, Ruchir Saheba for your wisdom and inspiration in all things estimation theory and controls.

References

[1] James T. Lo (1994), "Synthetic Approach to Optimal Filtering", IEEE Transactions on Neural Networks, Vol. 5, No. 5.

[2] Zachary C. Lipton, John Berkowitz, Charles Elkan (2015), "A Critical Review of Recurrent Neural Networks for Sequence Learning", arXiv.org

[3] Cappe et. al. (2007), "An Overview of Existing Methods and Recent Advances in Sequential Monte Carlo", Proceedings of the IEEE, Vol. 95, No. 5.

[4] M. L. Psiaki (2013), "The blind tricyclist problem and a comparative study of nonlinear filters: A challenging benchmark for evaluating nonlinear estimation methods," IEEE Control Systems Magazine, Vol. 33, No. 3.

[5] Ristic et. al. (2004), "Beyond the Kalman Filter. Particle Filters for Tracking Applications", Artech House.

[6] "TensorFlow", tensorflow.org.