Available online at www.sciencedirect.com

ScienceDirect

IFAC Papers Online CONFERENCE PAPER ARCHIVE

IFAC PapersOnLine 51-15 (2018) 1–6

Data-Driven Impulse Response Regularization via Deep Learning

Carl Andersson * **Niklas Wahlström** * **Thomas B. Schön** *

* *Department of Information Technology, Uppsala University, Sweden.*
Email: {carl.andersson, niklas.wahlstrom, thomas.schon}@it.uu.se

Abstract: We consider the problem of impulse response estimation of stable linear single-input single-output systems. It is a well-studied problem where flexible non-parametric models recently offered a leap in performance compared to the classical finite-dimensional model structures. Inspired by this development and the success of deep learning we propose a new flexible data-driven model. Our experiments indicate that the new model is capable of exploiting even more of the hidden patterns that are present in the input-output data as compared to the non-parametric models.

Keywords: Linear system identification, impulse response estimation, flexible models, deep learning, regularization, Gaussian processes.

1. INTRODUCTION

Impulse response estimation has for a long time been at the core of system identification. Up until some five to seven years ago, the generally held belief in the field was indeed that we knew all there was to know about this topic. However, the enlightening work by Pillonetto and De Nicolao (2010) changed this by showing that the estimate can in fact be improved significantly by placing a Gaussian Process (GP) prior on the impulse response, which acts as a regularizer. This model-driven approach has since then been further refined (Pillonetto et al., 2011; Chen et al., 2012; Pillonetto et al., 2014), where the prior in this case could be interpreted to encode not only smoothness information, but also information about the exponential decay of the impulse response. In this paper we employ deep leaning (DL) to find a suitable regularizer via a method that is driven by data. Fig. 1 depicts the general idea and the similarity of our method compared to the method based on Gaussian processes.

Deep learning is a fairly new area of research that continues the work on neural networks from the 1990's. To get a brief, but informative, overview of the field of deep learning we recommend the paper by LeCun et al. (2015) and for a more complete snapshot of the field we refer to the monograph by Goodfellow et al. (2016). Deep learning has recently revolutionized several fields, including image recognition (e.g. Cirean et al. (2011)) and speech recognition (e.g. Hinton et al. (2012)). In both fields, deep learning has surpassed domain specific methods and hand-crafted feature design, by making use of large quantities of data in order to learn data-driven neural network models as general function approximators.

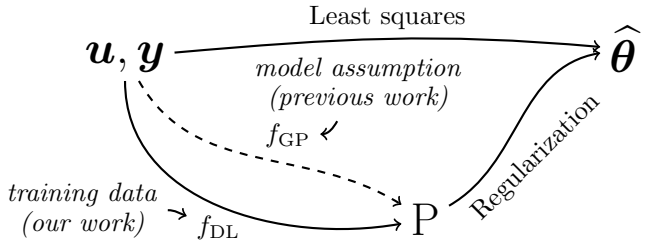


Fig. 1. Schematic figure over the proposed method for impulse response estimation using deep learning in relation to the previous work using Gaussian processes. The functions f_{GP} and f_{DL} maps the input sequence $\boldsymbol{u}$ and the output sequence $\boldsymbol{y}$ of a system to an inverse regularization matrix P for the Gaussian process approach and the deep learning approach, respectively.

The idea of using neural networks within system identification is certainly not new and they have been a standard tool for a long time, see e.g. Sjöberg et al. (1995). However, the current development in deep learning is different from the past due to advances in computational software and hardware. As a consequence, contemporary neural networks are more reproducible than before which has increased the credibility of the area. Finally, the amount of available data has sky-rocketed which has made it possible to train larger models with better results. We believe that system identification still has lots to gain from using deep learning and this paper is just one concrete example of what can be done.

2. BACKGROUND AND PROBLEM FORMULATION

Consider a stable single-input single-output time-invariant linear system G^0 relating an input sequence $u(t)$ to an output sequence $y(t)$ according to

$$y(t) = G^0(q)u(t) + v(t), \tag{1}$$

⋆ This research is financially supported by the Swedish Research Council via the project *NewLEADS - New Directions in Learning Dynamical Systems* (contract number: 621-2016-06079), and the Swedish Foundation for Strategic Research (SSF) via the project *ASSEMBLE* (contract number: RIT15-0012).

2405-8963
Peer review under responsibility of International Federation of Automatic Control.
10.1016/j.ifacol.2018.09.081

where q denotes the shift operator $qu(t) = u(t+1)$ and $v(t)$ denotes additive noise. The noise is assumed to be Gaussian white noise with zero mean and variance σ^2. The system $G^0(q)$ is represented using a transfer function

$$G^0(q) = \sum_{k=1}^{\infty} g_k^0 q^{-k}, \tag{2}$$

where $g_1^0, g_2^0, \dots$ denote the impulse response coefficients of the system. We make use of superscript 0 to denote the true system.

Based on data, i.e. an input sequence $\boldsymbol{u} = (u(1), \dots, u(N))^{\mathrm{T}}$ and an output sequence $\boldsymbol{y} = (y(1), \dots, y(N))^{\mathrm{T}}$ both of length N, the task is to compute an estimate that represents the true system $G^0(q)$ as well as possible.

Traditionally, the transfer function is encoded using a finite number of parameters $\boldsymbol{\theta}$ ($\dim(\boldsymbol{\theta}) = n \ll N$), for example, via a finite impulse response (FIR) model,

$$G(q;\boldsymbol{\theta}) = \mathrm{B}(q) = b_1 q^{-1} + \cdots + b_{n_b} q^{-n_b}, \tag{3}$$

or an output-error (OE) model,

$$G(q;\boldsymbol{\theta}) = \frac{\mathrm{B}(q)}{\mathrm{F}(q)} = \frac{b_1 q^{-1} + \cdots + b_{n_b} q^{-n_b}}{f_1 q^{-1} + \cdots + f_{n_f} q^{-n_f}}, \tag{4}$$

where $\boldsymbol{\theta} = (b_1, \dots, b_{n_b})^{\mathrm{T}}$ or $\boldsymbol{\theta} = (b_1, \dots, b_{n_b}, f_1, \dots, f_{n_f})^{\mathrm{T}}$, respectively. See Ljung (1999) for a comprehensive list of model structures. With these model structures, the prediction error method can be used to compute a point estimate $\widehat{\boldsymbol{\theta}}$ of the unknown parameters by solving the following optimization problem,

$$\widehat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} \sum_{t=1}^{N} (y(t) - G(q,\boldsymbol{\theta})u(t))^2 + \boldsymbol{\theta}^{\mathrm{T}} \mathrm{D} \boldsymbol{\theta}, \tag{5}$$

where $\boldsymbol{\theta}^{\mathrm{T}}\mathrm{D}\boldsymbol{\theta}$ describes the added regularization term, governed by the regularization matrix D. It has been shown by Pillonetto and De Nicolao (2010); Pillonetto et al. (2011); Chen et al. (2012) that the use of an effective regularization is more important than the specific choice of model-order, n. Intuitively the use of a GP model and the regularization term $\boldsymbol{\theta}^{\mathrm{T}}\mathrm{D}\boldsymbol{\theta}$ opens up for model selection at a "finer" scale compared to what is possible with the classical finitely parameterized model structures. Adding this kind of regularization is especially important when the number of samples, N, is low compared to the parameters we wish to estimate, n.

In the case of the FIR model, the optimization problem reduces to a linear least squares problem to find the optimal parameters $\widehat{\boldsymbol{\theta}}$. For numerical reasons the inverse regularization matrix $\mathrm{P} = \mathrm{D}^{-1}$ is often used in place of D. This gives rise to the following analytic solution,

$$\widehat{\boldsymbol{\theta}} = (\mathrm{PR} + \mathrm{I}_n)^{-1} \mathrm{PF}_N, \tag{6a}$$

where

$$\mathrm{R} = \sum_{t=n+1}^{N} \varphi(t)\varphi(t)^{\mathrm{T}}, \tag{6b}$$

$$\varphi(t) = \{u(t-1) \dots u(t-n)\}^{\mathrm{T}}, \tag{6c}$$

$$\mathrm{F}_N = \sum_{t=n+1}^{N} \varphi(t) y(t). \tag{6d}$$

We will throughout this paper denote then estimate (6) by $\widehat{\boldsymbol{\theta}}(\mathrm{P})$ to stress its dependence on the inverse regularization matrix P. As a special case, with $\mathrm{P} = 0$, we have the least squares solution, which we denote $\widehat{\boldsymbol{\theta}}_{\mathrm{LS}}$.

One question still remains though; how do we find the inverse regularization matrix P? One of the most general ideas is to let it depend on $\boldsymbol{u}$ and $\boldsymbol{y}$. This was done in Pillonetto and De Nicolao (2010); Pillonetto et al. (2011); Chen et al. (2012) by modelling the impulse response as a Gaussian process. A Gaussian process is known to be a very flexible prior, even so, since the model only depends on a low number of hyper-parameters, typically one for a lengthscale and one for some noise, it heavily depends on the specific model we choose for the covariance function. These hyper-parameters are replaced by a point estimate obtained by maximizing the marginal likelihood of the observed data, a procedure known as Empirical Bayes (Bishop, 2006). The regularization matrix will thus implicitly depend on $\boldsymbol{u}$ and $\boldsymbol{y}$ via the hyper-parameters. We explicitly denote this dependence by $\mathrm{P} = f_{\mathrm{GP}}(\boldsymbol{u}, \boldsymbol{y})$. This method is also explained in more detail in Section 3.

We instead propose an arguably even more flexible model by parametrizing the inverse regularization matrix P with a neural network $\mathrm{P} = f_{\mathrm{DL}}(\boldsymbol{u}, \boldsymbol{y})$. In contrast to the Gaussian process model these parameters are computed by training the model with lots of training data consisting of an input sequence, an output sequence and the true impulse response for either real or simulated systems. Compared to the GP model this is a more data-driven approach to the problem which also makes it possible to use existing techniques from deep learning when building and training the model.

3. REGULARIZATION USING GAUSSIAN PROCESS

The Gaussian prior offers a natural way of encoding the smoothness and decay characteristics that we find in the impulse response from a stable linear system. The specific details of these characteristics are tuned via the hyper-parameters λ. The resulting GP prior can be written as

$$\boldsymbol{\theta} \sim p_\lambda(\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\theta} \mid 0, \mathrm{P}^\lambda), \tag{7}$$

where P^λ in this Bayesian setting is equal to the inverse regularization matrix in (6). The matrix P^λ is related to the covariance function of the Gaussian Process as $k_\lambda(i,j) = \mathrm{P}^\lambda_{ij}$, where P^λ_{ij} denotes then entry on row i and column j of P^λ. With the aid of measured data we can select a point estimate of the hyper-parameters $\widehat{\lambda}$ by maximizing the marginal log-likelihood,

$$\widehat{\lambda} = \arg\max_{\lambda} \log \int p(\boldsymbol{y} \mid \boldsymbol{\theta}; \boldsymbol{u}) p_\lambda(\boldsymbol{\theta}) d\boldsymbol{\theta}, \tag{8}$$

where

$$p(\boldsymbol{y} \mid \boldsymbol{\theta}; \boldsymbol{u}) = \prod_{t=1}^{N} p(y(t) \mid \boldsymbol{\theta}; \boldsymbol{u}) = \prod_{t=1}^{N} \mathcal{N}(y(t) \mid \boldsymbol{\theta}^{\mathrm{T}} \varphi(t), \sigma^2). \tag{9}$$

As a direct consequence of this, the optimal inverse regularization matrix $\mathrm{P}^{\widehat{\lambda}}$ implicitly depends on the input sequence $\boldsymbol{u}$ and the output sequence $\boldsymbol{y}$. The equation (8) then describe the function $\mathrm{P} = f_{\mathrm{GP}}(\boldsymbol{u}, \boldsymbol{y})$. Using this inverse regularization matrix together with (6), we obtain an estimate of the FIR parameters that has better accuracy and is more robust than the non-regularized approach (Chen et al., 2012).

4. REGULARIZING USING DEEP LEARNING

In contrast to previous work where the regularization matrix only depends on a few hyper-parameters we instead model the regularization matrix directly with a neural network that depends on a large number of parameters η according to

$$\mathrm{P} = f_{\mathrm{DL}}(\boldsymbol{u}, \boldsymbol{y}; \eta). \tag{10}$$

To select specific values for all these parameters we start by formulating the mean squared error (MSE) of the estimate $\widehat{\boldsymbol{\theta}}(\mathrm{P})$ in (6), the so-called estimation error,

$$\mathrm{MSE}\left(\widehat{\boldsymbol{\theta}}\right) = \left\| \widehat{\boldsymbol{\theta}}(f_{\mathrm{DL}}(\boldsymbol{u}, \boldsymbol{y}; \eta)) - \boldsymbol{\theta}_0 \right\|^2, \tag{11}$$

where $\|\cdot\|^2$, denotes the 2-norm and $\boldsymbol{\theta}_0$ denotes the true FIR parameters which corresponds to the truncated true impulse response. Here we make use of the neural network model (10) of P when forming the MSE. The parameters $\widehat{\eta}$ to use in (10) are found by simply minimizing this estimation error,

$$\widehat{\eta} = \arg\min_{\eta} \frac{1}{M} \sum_{i=1}^{M} \left\| \widehat{\boldsymbol{\theta}}\left(f_{\mathrm{DL}}(\boldsymbol{u}^{(i)}, \boldsymbol{y}^{(i)}; \eta)\right) - \boldsymbol{\theta}_0^{(i)} \right\|^2, \tag{12}$$

where M is the number of training examples. To set the terminology straight we use the term *training* for the minimization of the estimation error w.r.t. η whereas *estimation* refers to the computation of a point estimate of the FIR parameters by minimizing the one step prediction error w.r.t. $\boldsymbol{\theta}$. The following sections describe our method and an overview is provided in Algorithm 1.

4.1 Regularization model

We know that P must be positive semi-definite. Inspired by this fact we choose P as a weighted sum of rank-one positive definite matrices. The idea behind this choice is to have a representation that is flexible enough to represent all possible regularization matrices and at the same time encode the knowledge that the optimal regularization matrix is a rank-one matrix (see Appendix A).

Our rank-one matrices are constructed as outer products of a vector $\boldsymbol{s}_i$ with itself where the elements of the vectors are free parameters. The weights, w_i, that weight our rank-one matrices, are modelled as the output of a softmax layer from a neural network. The input to this neural network is $\boldsymbol{u}$ and $\boldsymbol{y}$ as well as the nonregularized least squares solution, which only depends on $\boldsymbol{u}$ and $\boldsymbol{y}$. Hence, we have,

$$\mathrm{P} = \sum_{i=1}^{n_m} w_i(\boldsymbol{u}, \boldsymbol{y}, \widehat{\boldsymbol{\theta}}_{\mathrm{LS}}; \eta') \boldsymbol{s}_i \boldsymbol{s}_i^{\mathrm{T}}, \tag{13}$$

where n_m is the number of rank-one matrices used to represent the regularization matrix and η' is the parameters of the neural network. We use $\eta = \{\eta', \boldsymbol{s}_1, \ldots, \boldsymbol{s}_{n_m}\}$ to collect all the parameters in the regularization matrix model.

4.2 Neural network model

We have made use of four fully connected layers in our neural network, where the final layer is a softmax layer producing weights between 0 and 1. The other activation functions for the fully connected layers are Rectified Linear Units (ReLU) which are defined as $\mathrm{ReLU}(x) = \max(0, x)$. A typical layer of the network is thus written as $\boldsymbol{h}^{(i+1)} = \mathrm{ReLU}(\mathrm{W}\boldsymbol{h}^{(i)} + \boldsymbol{b})$, where $\boldsymbol{h}^{(i)}$ denotes the input to the layer, $\boldsymbol{h}^{(i+1)}$ denotes the output from the layer, W denotes a so-called weight matrix of dimensions $\dim(\boldsymbol{h}^{(i+1)}) \times \dim(\boldsymbol{h}^{(i)})$ and $\boldsymbol{b}$ denotes a so-called bias term of dimension $\dim(\boldsymbol{h}^{(i)})$. Both the weight matrices and the bias terms are part of the parameters η' describing the network, i.e. $\mathrm{W}^{(i)} \in \eta'$ and $\boldsymbol{b}^{(i)} \in \eta'$. To regularize the training procedure we add a dropout layer after the softmax layer which with 30% probability sets a weight to zero during training. This is a standard technique to avoid overfitting in NN (Srivastava et al., 2014). The input to the network is the input $\boldsymbol{u}$ and output $\boldsymbol{y}$ sequence of the system we want to estimate and the corresponding non-regularized least squares solution $\widehat{\boldsymbol{\theta}}_{\mathrm{LS}}$. The resulting network is schematically illustrated in Fig. 2.

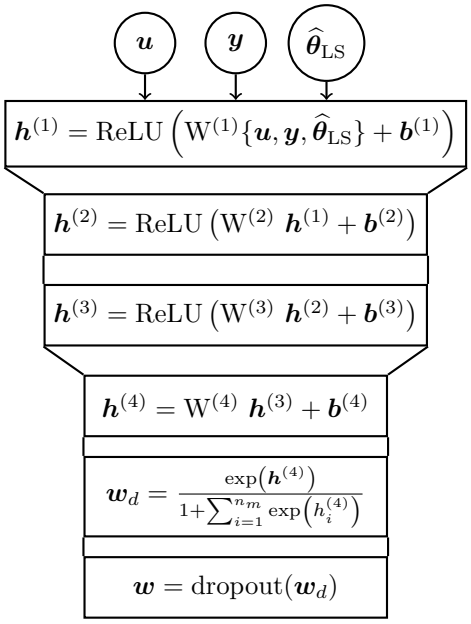


Fig. 2. Schematic description of our neural network. The weights from (13) are denoted by $\boldsymbol{w} = \{w_1, \ldots, w_{n_m}\}$. The dimensions of $\boldsymbol{h}^{(1)}, \boldsymbol{h}^{(2)}, \boldsymbol{h}^{(3)}$ and $\boldsymbol{h}^{(4)}$ are $600, 300, 200$ and 500 respectively. Note that the dimension of the final layer is equal to the number of matrices used.

4.3 Normalizing the data

A key aspect to successfully train neural networks is the normalization of the input and output of the network by subtracting the mean and dividing with the standard deviation. We notice that we can, without loss of generality, normalize each data example of $\boldsymbol{y}$ and $\boldsymbol{u}$ if we at the same time do the corresponding scaling of the impulse response $\boldsymbol{\theta}$ to keep the analytic relationship intact.

The non-regularized least square solution, $\widehat{\boldsymbol{\theta}}_{\mathrm{LS}}$, that we use as an input to the network is also straightforward normalize with the statistics form the true impulse response calculated from the training data. Finally, we want to normalize the networks dependence of the vectors $\boldsymbol{s}_i$. The outer product of these vectors should correspond to the optimal regularization (see Section 4.1 and Appendix A). To enforce that they follow the same statistics, we initialize

the vectors $\boldsymbol{s}_i$ with unit Gaussian and then multiply we the standard deviation and add the mean of the true impulse response.

Algorithm 1 Training procedure

1: Collect evaluation data.
2: Collect training data with similar behaviour as evaluation data (e.g. through simulations).
3: Normalize $\boldsymbol{u}$ and $\boldsymbol{y}$ for each example (Section 4.3).
4: Normalize $\widehat{\boldsymbol{\theta}}_{\mathrm{LS}}$ with statistics from the whole training dataset (Section 4.3).
5: Find $\hat{\eta}$ using training data by minimizing Equation (12).
6: Use $\hat{\eta}$ to predict $\widehat{\boldsymbol{\theta}}$ on evaluation data.

5. EXPERIMENT

The model explained in Section 4 is implemented using Tensorflow (Abadi et al., 2016) and our implementation of the model is available on GitHub [1] .

5.1 Simulate data using `rss`

To train the model, we use an artificial distribution over real systems to produce input and output sequences along with their true impulse responses. The artificial distribution over systems we use is MATLAB's `rss` function. This function is not ideal as was recently pointed out by for example Rojas et al. (2015). Hence, there is potential to further improve the results by making use of better data.

To generate data we use the same method as Chen et al. (2012) with some minor alterations concerning the signal to noise ratio (SNR). The full procedure follows as:

(1) Sample a system of order 30 using a slightly modified version of MATLAB's `rss` where the probability of having an integrating pole is zero.
(2) Sample $N = 125$ timesteps from the system at a sampling rate of 3 times the bandwidth, i.e.,
```
bw = bandwidth(m);
f = 3*(bw*2*pi);
md = c2d(m,1/f,'zoh');
```
(3) Calculate the true FIR parameters as the truncated impulse response to length $n = 50$.
(4) Sample white noise as an input sequence $\boldsymbol{u}$ and get the corresponding output sequence $\boldsymbol{y}^*$. Add noise to the output with the SNR drawn randomly from a uniform distribution between 1 and 10, i.e., the noise added has variance that is between 1 and 1/10 of the variance in the output sequence.
(5) Repeat all steps until we have M examples from M different systems.

These M training examples are then used in Equation (12) and are called the training data. Using the same method we generate a validation set which we also split in two sets depending on the SNR, one with SNR larger than 5.5 and one set with SNR less than 5.5 with roughly $M_v \approx 5\,000$ examples in each.

[1] https://github.com/carl-andersson/impest

5.2 Evaluation metrics

To evaluate the performance of the model we use a metric that is calculated as the mean squared error of the estimate normalized with the mean squared error of the least squares solution without any regularization. We denote this metric by S, i.e.,

$$\mathrm{S} = \frac{1}{M_v}\sum_{i=1}^{M_v}\left[\frac{\left\|\widehat{\boldsymbol{\theta}}\left(f_{\mathrm{DL}}(\boldsymbol{u}^{(i)},\boldsymbol{y}^{(i)};\eta)\right)-\boldsymbol{\theta}_0^{(i)}\right\|^2}{\left\|\widehat{\boldsymbol{\theta}}_{\mathrm{LS}}^{(i)}-\boldsymbol{\theta}_0^{(i)}\right\|^2}\right], \quad (14)$$

where M_v denotes the number of examples in the validation set. This metric makes sure that each impulse response gets equal weighting when computing the performance of the algorithm and measures the average effect of the regularization. A perfect match for this measure corresponds to a measure of 0.

Chen et al. (2012) make use of a slightly different metric defined as

$$\widetilde{\mathrm{S}} = \frac{1}{M_v}\sum_{i=1}^{M_v} 100\left(1-\left[\frac{\left\|\widehat{\boldsymbol{\theta}}\left(f_{\mathrm{DL}}(\boldsymbol{u}^{(i)},\boldsymbol{y}^{(i)};\eta)\right)-\boldsymbol{\theta}_0^{(i)}\right\|^2}{\left\|\boldsymbol{\theta}_0^{(i)}-\bar{\boldsymbol{\theta}}_0^{(i)}\right\|^2}\right]\right)$$

where $\bar{\boldsymbol{\theta}}_0^{(i)}$ is the mean of $\boldsymbol{\theta}_0^{(i)}$. This metric averages over a so-called 'model fit', i.e. how well the estimated parameters fit the true impulse response. Besides the shifting and scaling in $\widetilde{\mathrm{S}}$, the only difference between the two metrics is the normalization factor used, where S is normalized with the least squares estimation error and $\widetilde{\mathrm{S}}$ is normalized with the variance of the true impulse response. We have empirically observed that the terms in $\widetilde{\mathrm{S}}$ might vary a lot between different examples and the average might thus be dominated by a few examples leading to measures that are hard to interpret. Our slightly modified metric S will on the other hand measure the average effect of using a regularization method compared to not using a regularization method, which seem to result in a more stable performance indicator.

5.3 Simulation results

The model is trained using $M = 1\,000\,000$ training examples for roughly 2.5 hours using a desktop computer with an Nvidia Titan Xp GPU. The chosen hyperparameters of the model is described in Figure 2. Note that while training require a GPU with large memory, the evaluation can easily be done in CPU on an ordinary computer. We are using early stopping as a stopping criteria even though the model essentially does not seem to overfit with this amount of training data. The model is not very dependent on the number of examples in the training data either. Even with $M = 10\,000$ it managed to achieve comparable results to previous methods.

Fig. 3 shows a subset of the matrices $\boldsymbol{s}_i\boldsymbol{s}_i^{\mathrm{T}}$ from (13) after training. Note that the matrices have an oscillating pattern with different periods and a decay towards zero for parameters with high index (lower right corner). Fig. 4 shows three different regularization matrices for an example in the validation dataset. We can see that the deep learning regularization seems to capture the behaviour of the optimal regularization matrix fairly well. In Fig. 5 we can

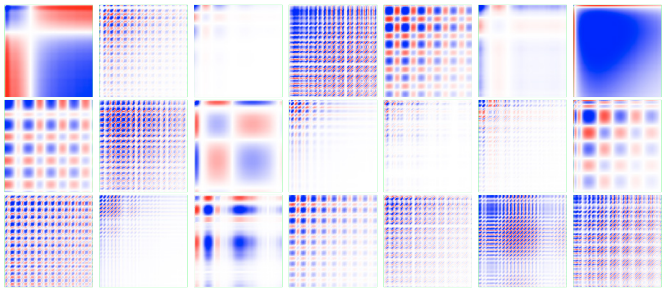


Fig. 3. Illustration of 21 matrices from $\boldsymbol{s}_i \boldsymbol{s}_i^{\mathrm{T}}$ after training. The matrices are rescaled to the interval $[+1, -1]$, where blue indicates a positive value, white around zero and red indicates a negative value. The upper left corner corresponds to the lowest index of the estimate $\boldsymbol{\theta}$.

compare the estimates from the different regularization approaches.

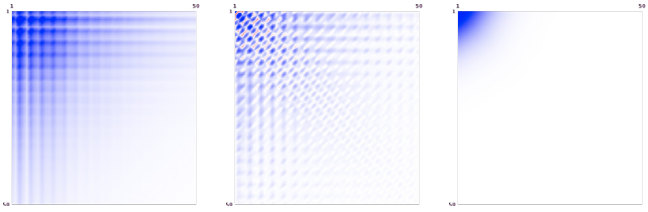
The trained model does not produce a useful regularization matrix for all examples. In cases where it fails the neural network seems to fail in the same way for all examples by producing a similar regularization matrix for each example with a bad performance for S as consequence. Despite this problem, the model manages to produce average results which are comparable or better than previous methods which is reflected in Table 1. We can see that the performance decreases when the SNR gets larger. This is due to the improved effectiveness of the least squares estimate and we are thus less dependent on the regularization. For comparison we also present the result for the optimal regularization which of course is unachievable since it depends on the true impulse response, but it is still an useful lower bound.

Table 1. Comparing the different models using the metric from (14) evaluated using the validation set. LS stands for Least Squares, OR stands for Optimal regularization (see Appendix A), GP stands for Gaussian process regularization and DL stands for deep learning regularization. LS, OR and GP are not data driven approaches and they are thus not dependent on any training data.

Model	SNR < 5.5	SNR > 5.5
LS	1	1
OR	0.04	0.05
GP	0.31	0.40
DL	0.20	0.23

5.4 Real data results

To show that our method at least does not give unreasonable estimates for real systems and data we test our method on data measured at a processing plant. See e.g. Bittencourt et al. (2015) for an introduction to this problem area. We do not know the true parameter values for these systems, implying that we cannot evaluate the performance of the estimates. We use the same trained network as we evaluated in the previous section. In Fig. 6 we simply present an input sequence and the corresponding output sequence from a real system together with the estimates produced by our method compared to using least squares without regularization. We note also that the results seems reasonable and that the regularization removes a lot of noise in the estimation.



(a) Optimal regularization (b) Deep learning regularization (c) Gaussian process regularization

Fig. 4. Comparison between rescaled inverse regularization matrices for a validation example where our method captures the behaviour of the optimal regularization matrix.

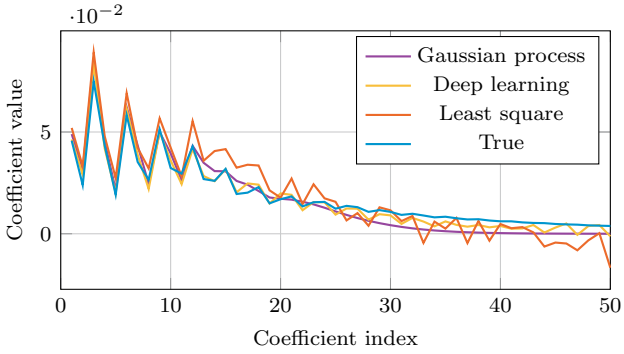
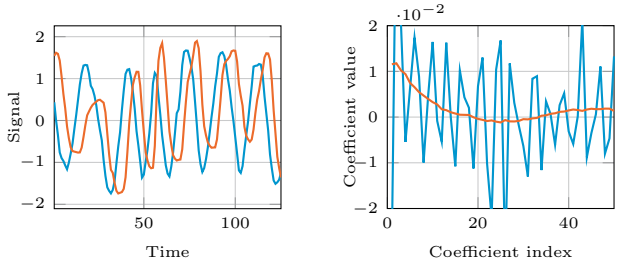


Fig. 5. Estimates of the impulse response coefficients, $\widehat{\boldsymbol{\theta}}$, using the inverse regularization matrices from Fig. 4 and the same input and output sequence.



(a) An input sequence (blue) and the corresponding output (red) for a real system.

(b) Impulse response estimates using the data in Fig. 6a for regular least squares (blue) and our method (red).

Fig. 6. Our method applied to an example from a real system and real measured data.

6. CONCLUSION

In this paper we present a method to regularize an impulse response estimation problem. We train a model with simulated data in a data-driven fashion to encode this regularization with parameters in a neural network. A trained model can then be used to improve the mean squared error of the new estimations. The results of our method seems promising and there is plenty of scope for future work along this line of research, both when it comes to impulse response estimation, but also for other problems. We find it especially interesting that this model can mimic the optimal regularization matrix to higher

degree than previous methods which we believe is the reason for why it sometimes produce better estimates.

Although training our model is quite time consuming, estimating the impulse response using our method is very fast since it only involves a couple of matrix multiplications to compute the regularization matrix, whereas the method of Chen et al. (2012) needs to solve an optimization problem for each example.

7. FUTURE WORK

We are planning to further investigate how one can make use of real data from e.g. the process industry. This would make it possible to use large amounts of collected data to improve the estimated parameters in a data-driven manner. The process industry has a lot of data available and makes extensive use of linear models.

The idea of learning a prior by representing it with a regularization matrix in the form of a neural network is not unique to the problem of estimating the impulse response. It could easily be generalized to other situations where the least squares solution is available but the prior of the solution is either unknown or intractable. If one can simulate many such systems at low cost, or have data from true systems available, formulating a regularization matrix as a neural network might be a tractable way of regularizing the estimate.

The presented approach can easily be extended to multi-input multi-output systems where the only difference is that the dimension of the input and output sequences and the parameters $\boldsymbol{\theta}$ increases. The deep structure of the model automatically induces any relevant connection between the different system input and output components present in the training data.

Finally we want to stress that this is just one example of what one might do with deep learning in system identification. There might and should be other areas where it is possible to make use of either simulated or real data to improve standard methods, or invent new methods for system identification. For example it might be worth looking into Recurrent Neural Networks (RNN) such as Long Short Term Memory (LSTM)(Hochreiter and Urgen Schmidhuber, 1997), Gated Recurrent Units (GRU) (Cho et al., 2014) or Stocastic Recurrent Neural Network (SRNN)(Fraccaro et al., 2016) and apply it in a system identification setting or even to bring some of the system identification knowledge of dynamical systems to the field of deep learning.

ACKNOWLEDGEMENTS

We would like to thank Professor Krister Forsman at Perstorp and the Norwegian University of Science and Technology for providing us with the real world data. We would also like to thank Dr Jack Umenberger for useful feedback on an early draft of this work.

REFERENCES

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mane, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viegas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2016). TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems.

Bishop, C.M. (2006). *Pattern Recognition and Machine Learning*. Springer.

Bittencourt, A.C., Isaksson, A., Peretzki, D., and Forsman, K. (2015). An algorithm for finding process identification intervals from normal operating data. *Processes*, 3(2), 357–383.

Chen, T., Ohlsson, H., and Ljung, L. (2012). On the estimation of transfer functions, regularizations and Gaussian processes - Revisited. *Automatica*, 48(8), 1525–1535.

Cho, K., Merrienboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar.

Cirean, D.C., Meier, U., Masci, J., Gambardella, L.M., and Schmidthuber, J. (2011). Flexible, high performance convolutional neural networks for image classification. In *Proceedings of the Twenty-Second international joint conference on Artificial Intelligence (IJCAI)*. Barcelona, Spain.

Fraccaro, M., Sønderby, S.K., Paquet, U., and Winther, O. (2016). Sequential Neural Models with Stochastic Layers. In *Advances in Neural Information Processing Systems 29*, 2199–2207.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep learning*. MIT Press.

Hinton, G., Deng, L., Yu, D., Dahl, G.E., Mohamed, A.r., Jaitly, N., Senior, A., Vanhoucke, V., Nguyen, P., Sainath, T.N., and Kingsbury, B. (2012). Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups. *IEEE Signal Processing Magazine*, 29(6), 82–97.

Hochreiter, S. and Urgen Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, 9(8), 1735–1780.

LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature*, 521(7553), 436–444.

Ljung, L. (1999). *System identification: Theory for the User*. Wiley Online Library.

Pillonetto, G., Dinuzzo, F., Chen, T., De Nicolao, G., and Ljung, L. (2014). Kernel methods in system identification, machine learning and function estimation: a survey. *Automatica*, 50(3), 657–682.

Pillonetto, G., Chiuso, A., and De Nicolao, G. (2011). Prediction error identification of linear systems: a nonparametric Gaussian regression approach. *Automatica*, 47(2), 291–305.

Pillonetto, G. and De Nicolao, G. (2010). A new kernel-based approach for linear system identification. *Automatica*, 46(1), 81–93.

Rojas, C.R., Valenzuela, P.E., and Rojas, R.A. (2015). A Critical View on Benchmarks based on Randomly Generated Systems. *IFAC-PapersOnLine*, 48(28), 1471–1476.

Sjöberg, J., Zhang, Q., Ljung, L., Benveniste, A., Delyon, B., Glorennec, P.Y., Hjalmarsson, H., and Juditsky, A. (1995). Nonlinear black-box modeling in system identification: a unified overview. *Automatica*, 31(12), 1691–1724.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, 15, 1929–1958.

Appendix A. OPTIMAL REGULARIZATION

The optimal regularization matrix is a term coined by Chen et al. (2012). It corresponds to the regularization matrix that is optimal in the sense that it minimizes the mean squared error. The optimal regularization matrix can be written as, $\mathrm{P} = \sigma^{-2}\boldsymbol{\theta}_0\boldsymbol{\theta}_0^{\mathrm{T}}$ (Chen et al., 2012), where σ^2 is the variance of the additive noise $v(t)$ in (1), $\boldsymbol{\theta}_0$ is the true impulse response of the system without noise.