QUASI-NEWTON METHODS FOR LEARNING NONLINEAR STATE-SPACE MODELS

Alberto Bemporad

imt.lu/ab

SCHOOL FOR ADVANCED STUDIES LUCCA

8th Workshop on Nonlinear System Identification Benchmarks - April 25, 2024

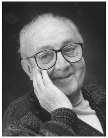
OUTLINE

- **Focus**: training **control-oriented** state-space models
- **Generalized Gauss-Newton** (**Sequential Least-Squares**) methods
- **Extended Kalman Filtering** methods
- **L-BFGS** (Limited-memory Broyden–Fletcher–Goldfarb–Shanno) methods

LEARNING CONTROL-ORIENTED MODELS

"All models are wrong, but some are useful."

(George E. P. Box)



CONTROL-ORIENTED MODELS

- **Complex model = complex controller** (controller design and evaluation)
  Example: Model Predictive Control (MPC)

- Typically look for **small-scale models** (e.g., ≤ 10 states/inputs/outputs) with a **limited number of coefficients** (vs. Large Language Models: 2-300 B params)

- **Limit nonlinearities** as much as possible (e.g., avoid very deep neural networks)

- Need to get the **best model** within a **poor model class** from a rich dataset (= limited risk to overfit)

- **Computation constraints**: solve the learning problem using limited resources (=our laptop, no supercomputing infrastructures)

Solving system identification problems requires different algorithms than in typical machine learning tasks

NONLINEAR SYS-ID BASED ON NEURAL NETWORKS

- **Neural networks** proposed for nonlinear system identification since the '90s

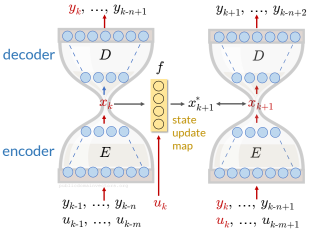
  (Narendra, Parthasarathy, 1990) (Hunt et al., 1992) (Suykens, Vandewalle, De Moor, 1996)

- **NNARX** models: use a **feedforward neural network** to approximate the nonlinear difference equation $y_t \approx \mathcal{N}(y_{t-1}, \ldots, y_{t-n_a}, u_{t-1}, \ldots, u_{t-n_b})$

- **Neural state-space** models:

  - **w/ state data**: fit a neural network model $x_{t+1} \approx \mathcal{N}_x(x_t, u_t), \quad y_t \approx \mathcal{N}_y(x_t)$

  - **I/O data only**: set x_t = value of an inner layer of the network (Prasad, Bequette, 2003), such as an **autoencoder** (Masti, Bemporad, 2021)



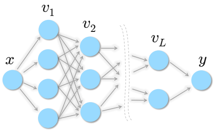
- Often, the **open-loop prediction error** must be minimized to get good models

RECURRENT NEURAL NETWORKS

- **Recurrent Neural Network** (RNN) model:

$$\begin{aligned} x_{k+1} &= f_x(x_k, u_k, \theta_x) \\ y_k &= f_y(x_k, \theta_y) \\ f_x, f_y &= \text{feedforward neural network} \end{aligned}$$

(e.g.: general RNNs, LSTMs, RESNETS, physics-informed NNs, ...)



$$v_j = A_j f_{j-1}(v_{j-1}) + b_j$$

$$\theta = (A_1, b_1, \ldots, A_L, b_L)$$

- **Training problem**: given an I/O dataset $\{u_0, y_0, \ldots, u_{N-1}, y_{N-1}\}$ solve

$$\begin{aligned} \min_{\substack{\theta_x, \theta_y \\ x_0, x_1, \ldots, x_{N-1}}} \quad & r(x_0, \theta_x, \theta_y) + \frac{1}{N} \sum_{k=0}^{N-1} \ell(y_k, f_y(x_k, \theta_y)) \\ \text{s.t.} \quad & x_{k+1} = f_x(x_k, u_k, \theta_x) \end{aligned}$$

- **Main issue**: x_k are **hidden states** and hence also **unknowns** of the problem

GRADIENT DESCENT METHODS FOR TRAINING RNNS

- **Problem condensing**: substitute $x_{k+1} = f_x(x_k, u_k, \theta_x)$ recursively and solve

$$\min_{\theta_x,\theta_y,x_0} r(x_0,\theta_x,\theta_y) + \frac{1}{N}\sum_{k=0}^{N-1} \ell(y_k, f_y(x_k,\theta_y)) = \boxed{\min_{\theta_x,\theta_y,x_0} V(\theta_x,\theta_y,x_0)}$$

- **Gradient descent** (**GD**) methods: update θ_x, θ_y, x_0 by setting

$$\begin{bmatrix} \theta_x^{t+1} \\ \theta_y^{t+1} \\ x_0^{t+1} \end{bmatrix} = \begin{bmatrix} \theta_x^{t} \\ \theta_y^{t} \\ x_0^{t} \end{bmatrix} - \alpha_t \nabla V(\theta_x^t, \theta_y^t, x_0^t)$$

Example: **Adam** uses adaptive moment estimation to set the learning rate α_t

(Kingma, Ba, 2015)

GRADIENT DESCENT METHODS FOR TRAINING RNNS

- **Main issue** with GD methods: **slow convergence** (in theory and in practice)

- **Stochastic** gradient descent (SGD) can be even less efficient with RNNs:
  - collect a high number of short independent experiments (often impossible)
  - create mini-batches by using **multiple-shooting** ideas
    (Forgione, Piga, 2020) (Bemporad, 2023)

- **Newton's method**: very fast (2^{nd}-order) local convergence but difficult to implement, as we need the **Hessian** $\nabla^2 V(\theta_x^t, \theta_y^t, x_0^t)$

- **Quasi-Newton methods**: good tradeoff between convergence speed (=solution quality) and numerical complexity, only the **gradient** $\nabla V(\theta_x^t, \theta_y^t, x_0^t)$ is required

TRAINING RNNS VIA SEQUENTIAL LEAST SQUARES

TRAINING RNNS BY SEQUENTIAL LEAST-SQUARES

(Bemporad, 2023)

- RNN training problem = **optimal control** problem:

$$\begin{array}{rl} \displaystyle\min_{\theta_x,\theta_y,x_0,x_1,\ldots,x_{N-1}} & r(x_0,\theta_x,\theta_y)+\displaystyle\sum_{k=0}^{N-1}\ell(y_k,\hat{y}_k) \\ \text{s.t.} & x_{k+1}=f_x(x_k,u_k,\theta_x) \\ & \hat{y}_k=f_y(x_k,u_k,\theta_y) \end{array}$$

inputs $=\theta_x,\theta_y,x_0$

output $=\hat{y}$

reference $=y_k$

meas. dist. $=u_k$

  - $r(x_0,\theta_x,\theta_y)$ = input penalty
  - $\ell(y_k,\hat{y}_k)$ = output penalty
  - prediction horizon = N steps, control horizon = 1 step

- **Linearized model**: given a current guess $\theta_x^h,\theta_y^h,x_0^h,\ldots,x_{N-1}^h$, approximate

$$\begin{array}{rcl} \Delta x_{k+1} & = & (\nabla_x f_x)'\Delta x_k+(\nabla_{\theta_x} f_x)'\Delta\theta_x \\ \Delta y_k & = & (\nabla_x f_y)'\Delta x_k+(\nabla_{\theta_y} f_y)'\Delta\theta_y \end{array}$$

TRAINING RNNS BY SEQUENTIAL LEAST-SQUARES

- Take 2^{nd}-order expansion of the loss ℓ and regularization term r

- Solve **least-squares** problem to get increments $\Delta x_0, \Delta\theta_x, \Delta\theta_y$

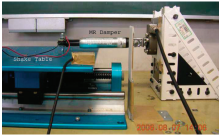
- Update $x_0^{h+1}, \theta_x^{h+1}, \theta_y^{h+1}$ by applying either a
  - **line-search** (LS) method based on Armijo rule
  - or a **trust-region** method (Levenberg-Marquardt) (LM)

- The resulting training method is a **Generalized Gauss-Newton** method
  ⇒ very good convergence properties (Messerer, Baumgärtner, Diehl, 2021)

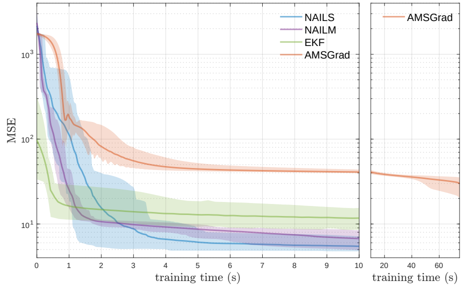
- No guarantee to converge to a global minimum (multiple runs may be required)

- **Example**: **magneto-rheological fluid damper**
  N=2000 data used for training, 1499 for testing the model

  (Wang, Sano, Chen, Huang, 2009)



- RNN model: 4 states, shallow NNs w/ **4 neurons**, **I/O feedthrough**



MSE loss on training data, mean value and range over 20 runs from different random initial weights

$$\text{BFR} = 100\left(1 - \frac{\|Y - \hat{Y}\|_2}{\|Y - \bar{y}\|_2}\right)$$

NAILS = GNN method with **line search**

NAILM = GNN method with **LM steps**

BFR (Best Fit Rate)	training	test
NAILS	**94.41** (0.27)	89.35 (2.63)
NAILM	94.07 (0.38)	89.64 (2.30)
AMSGrad	84.69 (0.15)	80.56 (0.18)
EKF	91.41 (0.70)	87.17 (3.06)

TRAINING RNNS BY SEQUENTIAL LS AND ADMM

- We also want to handle **non-smooth**/**non-convex** regularization terms

$$\begin{array}{rl} \min_{\theta_x,\theta_y,x_0} & r(x_0,\theta_x,\theta_y)+\sum_{k=0}^{N-1}\ell(y_k,f_y(x_k,\theta_y))+g(\theta_x,\theta_y) \\ \text{s.t.} & x_{k+1}=f_x(x_k,u_k,\theta_x) \end{array}$$

E.g.: $g(\theta_x,\theta_y)=\tau(\|\theta_x\|_1+\|\theta_y\|_1)$ (Lasso regularization)

- **Idea**: use **alternating direction method of multipliers** (ADMM) by splitting

$$\begin{array}{rl} \min_{\theta_x,\theta_y,x_0,\nu_x,\nu_y} & r(x_0,\theta_x,\theta_y)+\sum_{k=0}^{N-1}\ell(y_k,f_y(x_k,\theta_y))+g(\nu_x,\nu_y) \\ \text{s.t.} & x_{k+1}=f_x(x_k,u_k,\theta_x) \\ & \begin{bmatrix}\nu_x\\ \nu_y\end{bmatrix}=\begin{bmatrix}\theta_x\\ \theta_y\end{bmatrix} \end{array}$$

TRAINING RNNS BY SEQUENTIAL LS AND ADMM

- ADMM + Seq. LS = **NAILS** algorithm (Nonconvex ADMM Iterations and Sequential LS)

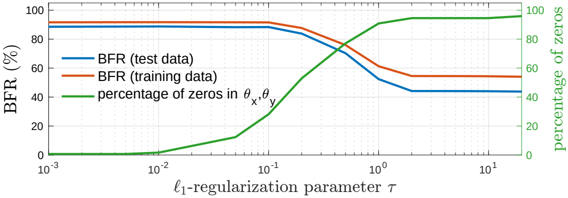
$$\begin{bmatrix} x_0^{t+1} \\ {\theta_x}^{t+1} \\ {\theta_y}^{t+1} \end{bmatrix} = \arg\min_{x_0,\theta_x,\theta_y} V(x_0,\theta_x,\theta_y) + \frac{\rho}{2}\left\|\begin{bmatrix} \theta_x-\nu_x^t+w_x^t \\ \theta_y-\nu_y^t+w_y^t \end{bmatrix}\right\|_2^2$$ (sequential) LS

$$\begin{bmatrix} \nu_x^{t+1} \\ \nu_y^{t+1} \end{bmatrix} = \mathrm{prox}_{\frac{1}{\rho}g}({\theta_x}^{t+1}+w_x^t, {\theta_y}^{t+1}+w_y^t)$$ proximal step

$$\begin{bmatrix} w_x^{t+1} \\ w_y^{t+1} \end{bmatrix} = \begin{bmatrix} w_x^h+{\theta_x}^{t+1}-\nu_x^{t+1} \\ w_y^h+{\theta_y}^{t+1}-\nu_y^{t+1} \end{bmatrix}$$ update dual vars

- ADMM + Levenberg-Marquardt steps = **NAILM** algorithm

- Fluid-damper example: **Lasso regularization** $g(\nu_x,\nu_y)=\tau(\|\nu_x\|_1+\|\nu_y\|_1)$



(mean results over 20 runs from different initial weights)

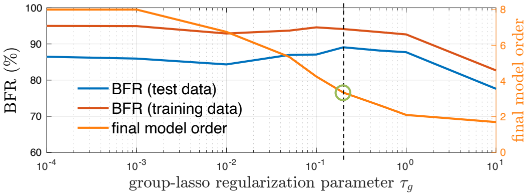
TRAINING RNNS BY SEQUENTIAL LS AND ADMM

- Fluid-damper example: **Lasso regularization** $g(\nu_x, \nu_y) = 0.2(\|\nu_x\|_1 + \|\nu_y\|_1)$

training algorithm	BFR training	BFR test	sparsity %	CPU time	# epochs
NAILS	91.00 (1.66)	87.71 (2.67)	65.1 (6.5)	11.4 s	250
NAILM	91.32 (1.19)	87.80 (1.86)	64.1 (7.4)	11.7 s	250
AMSGrad	91.04 (0.47)	88.32 (0.80)	16.8 (7.1)	64.0 s	2000
Adam	90.47 (0.34)	87.79 (0.44)	8.3 (3.5)	63.9 s	2000
DiffGrad	90.05 (0.64)	87.34 (1.14)	7.4 (4.5)	63.9 s	2000
EKF	89.27 (1.48)	86.67 (2.71)	47.9 (9.1)	13.2 s	50

$\approx$ same fit than SGD/EKF but sparser models and faster (Apple M1 Pro)

- Fluid-damper example: **group-Lasso regularization** $g(\nu_i^g) = \tau_g \sum_{i=1}^{n_x} \|\nu_i^g\|_2$ to zero entire rows/columns and **reduce the state-dimension** automatically

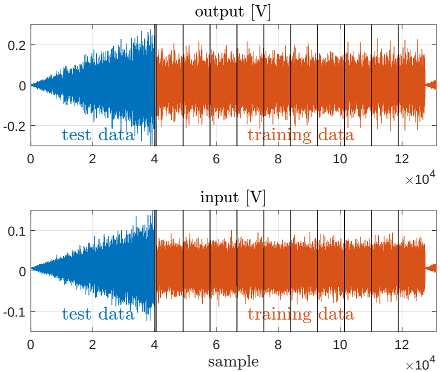


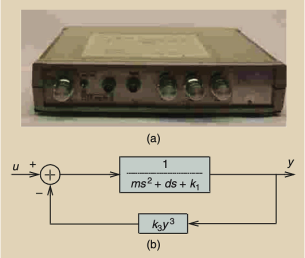
good choice: $n_x = 3$ (best fit on test data)

TRAINING RNNS - SILVERBOX BENCHMARK

(Wigren, Schoukens, 2013)

- **Silverbox benchmark** (Duffin oscillator): 10 traces (≈8600 samples each) used for training, 40000 for testing





(Schoukens, Ljung, 2019)

Data download: `http://www.nonlinearbenchmark.org`

TRAINING RNNS - SILVERBOX BENCHMARK

(Bemporad, 2023)

- **RNN model**: 8 states, 3 layers of 8 neurons, atan activation, no I/O feedthrough
- **Initial-state**: **encode** x_0 as the output of a NN with atan activation, 2 layers of 4 neurons, receiving 8 past inputs and 8 past outputs

$$\begin{array}{rl}\min_{\theta_{x_0},\theta_x,\theta_y} & r(\theta_{x_0},\theta_x,\theta_y)+\sum_{j=1}^{M}\sum_{k=0}^{N-1}\ell(y_k^j,\hat{y}_k^j)\\ \text{s.t.} & x_{k+1}^j=f_x(x_k^j,u_k^j,\theta_x),\ \hat{y}_k^j=f_y(x_k^j,u_k^j,\theta_y)\\ & x_0^j=f_{x_0}(v^j,\theta_{x_0})\end{array} \qquad v=\begin{bmatrix} y_{-1}\\ \vdots\\ y_{-8}\\ u_{-1}\\ \vdots\\ u_{-8}\end{bmatrix}$$

[cf. (Beintema, Toth, Schoukens, 2021)]

- ℓ_2-regularization: $r(\theta_{x_0},\theta_x,\theta_y)=\frac{0.01}{2}(\|\theta_x\|_2^2+\|\theta_y\|_2^2)+\frac{0.1}{2}\|\theta_{x_0}\|_2^2$
- Total number of parameters $n_{\theta_x}+n_{\theta_y}+n_{\theta_{x_0}}=296+225+128=$ **649**
- Training: use NAILM over **150 epochs**

TRAINING RNNS - SILVERBOX BENCHMARK

- Identification results on test data [1]:

identification method	RMSE [mV]	BFR [%]
ARX (ml) [1]	16.29 [4.40]	69.22 [73.79]
NLARX (ms) [1]	8.42 [4.20]	83.67 [92.06]
NLARX (mlc) [1]	1.75 [1.70]	96.67 [96.79]
NLARX (ms8c50) [1]	1.05 [0.30]	98.01 [99.43]
Recurrent LSTM model [2]	2.20	95.83
SS encoder [3] ($n_x = 4$)	[1.40]	[97.35]
NAILM	0.35	99.33

[1] Ljung, Zhang, Lindskog, Juditski, 2004

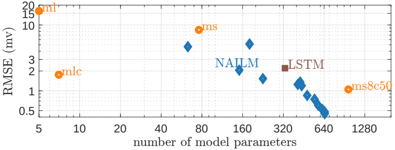
[2] Ljung, Andersson, Tiels, Schön, 2020

[3] Beintema, Toth, Schoukens, 2021

$$\text{RMSE} = \sqrt{\frac{1}{N}\sum_{k=1}^{N}(y_k - \hat{y}_k)^2}$$

- NAILM training time $\approx$ 400 s (MATLAB+CasADi on Apple M1 Max CPU)

- Repeat training with ℓ_1-regularization:



[1]Trained RNN: `http://cse.lab.imtlucca.it/~bemporad/shared/silverbox/rnn888.zip`

EXTENDED KALMAN FILTER FOR TRAINING RNNS

TRAINING RNNS BY EKF

(Puskorius, Feldkamp, 1994) (Wang, Huang, 2011) (Bemporad, 2023)

- Iterating an **Extended Kalman Filter** (**EKF**) based on the following model

$$\begin{cases} x_{k+1} = f_x(x_k, u_k, \theta_{xk}) + \xi_k \\ \begin{bmatrix} \theta_{x(k+1)} \\ \theta_{y(k+1)} \end{bmatrix} = \begin{bmatrix} \theta_{xk} \\ \theta_{yk} \end{bmatrix} + \eta_k \\ y_k = f_y(x_k, \theta_{yk}) + \zeta_k \end{cases}$$

$$Q = \mathrm{Var}[\eta_k] \quad R = \mathrm{Var}[\zeta_k] \quad P_0 = \mathrm{Var}\left[\begin{bmatrix} \theta_x \\ \theta_y \\ x_0 \end{bmatrix}\right]$$

is equivalent to applying Newton's method incrementally to solve the relaxed problem (Humpherys, Redd, West, 2012)

$$\min_{\substack{\theta_x, \theta_y \\ x_0, x_1, \ldots, x_{N-1}}} \left\| \begin{bmatrix} \theta_x \\ \theta_y \\ x_0 \end{bmatrix} \right\|_{P_0^{-1}}^2 + \sum_{k=0}^{N-1} \|y_k - f_y(x_k, \theta_y)\|_{R^{-1}}^2 + \sum_{k=0}^{N-2} \left\| \begin{bmatrix} x_{k+1} - f_x(x_k, u_k, \theta_x) \\ \theta_{k+1} - \theta_k \end{bmatrix} \right\|_{Q^{-1}}^2$$

- The ratio Q/R determines the **learning-rate** of the training algorithm

- **Generalization**: train via **Moving Horizon Estimation** (MHE)
  (Løwenstein, Bernardini, Bemporad, Fagiano, 2023)

(Bemporad, 2023)

- EKF can be generalized to handle **general strongly convex and smooth** losses $\ell(y_k, \hat{y}_k)$ by taking a local quadratic approximation of the loss around $\hat{y}_k$:

$$\begin{aligned} \ell(y_k, \hat{y}) &\approx \tfrac{1}{2}\Delta y' H(k)\Delta y + \phi_k' \Delta y + \text{const} \\ &= \tfrac{1}{2}\left\| y_k - H^{-1}(k)\phi_k - \hat{y}\right\|_{H(k)}^2 + \text{const} \end{aligned}$$

$$\Delta y = \hat{y} - \hat{y}_k,\ \phi_k = \frac{\partial \ell(y_k, \hat{y}_k)}{\partial \hat{y}}$$
$$H(k) = \frac{\partial^2 \ell(y_k, \hat{y}_k)}{\partial \hat{y}_k^2}$$

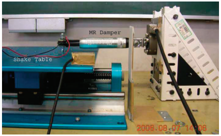
- Strongly convex smooth regularization $r(x_0, \theta_x, \theta_y)$ can be handled similarly

- Can handle **ℓ_1-penalties** $\lambda \left\| \begin{bmatrix} \theta_x \\ \theta_y \end{bmatrix} \right\|_1$, useful to **sparsify** θ_x, θ_y by changing the EKF update into

$$\begin{bmatrix} \hat{x}(k|k) \\ \theta_x(k|k) \\ \theta_y(k|k) \end{bmatrix} = \begin{bmatrix} \hat{x}(k|k-1) \\ \theta_x(k|k-1) \\ \theta_y(k|k-1) \end{bmatrix} + M(k)e(k) - \lambda P(k|k-1) \begin{bmatrix} 0 \\ \mathrm{sign}(\theta_x(k|k-1)) \\ \mathrm{sign}(\theta_y(k|k-1)) \end{bmatrix}$$

The model θ_x, θ_y can be learned offline by processing a given dataset multiple times, and also **adapted on line** from streaming data (u_k, y_k)

TRAINING RNNS BY EKF - EXAMPLES



- **Dataset**: **magneto-rheological fluid damper**
  3499 I/O data (Wang, Sano, Chen, Huang, 2009)

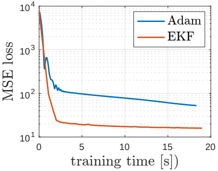
- N=2000 data used for training, 1499 for testing the model

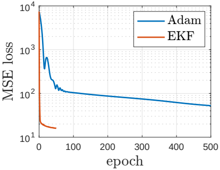
- Same data used in NNARX modeling demo of SYS-ID Toolbox for MATLAB

- **RNN model**: **4** hidden states, shallow state-update and output functions
  6 neurons, **atan** activation, I/O feedthrough

- Compare with **gradient descent** (Adam)

  MATLAB+CasADi implementation (Apple M1 Max CPU)



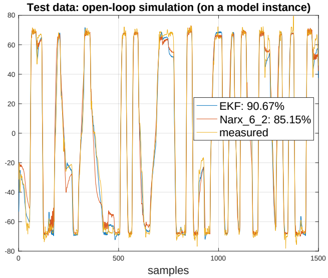


TRAINING RNNS BY EKF - EXAMPLES

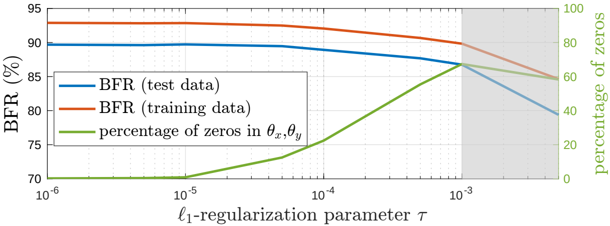
- Compare BFR[2] wrt NNARX model (SYS-ID TBX):

  EKF = **92.82**, Adam = **89.12**, NNARX(6,2) = **88.18** (**training**)

  EKF = **89.78**, Adam = **85.51**, NNARX(6,2) = **85.15** (**test**)



- Repeat training with ℓ_1-penalty $\tau \left\| \begin{bmatrix} \theta_x \\ \theta_y \end{bmatrix} \right\|_1$



[2]Best fit rate BFR=$100(1 - \frac{\|Y-\hat{Y}\|_2}{\|Y-\bar{y}\|_2})$, averaged over 20 runs from different initial weights

TRAINING LSTMS BY EKF - EXAMPLES

- Use EKF to train **Long Short-Term Memory** (**LSTM**) model

(Hochreiter, Schmidhuber, 1997) (Bonassi et al., 2020)

$$
\begin{aligned}
x_a(k+1) &= \sigma_G(W_F u(k) + U_f x_b(k) + b_f) \odot x_a(k) \\
&\quad + \sigma_G(W_I u(k) + U_I x_b(k) + b_I) \odot \sigma_C(W_C u(k) + U_C x_b(k) + b_C) \\
x_b(k+1) &= \sigma_G(W_O u(k) + U_O x_b(k) + b_O) \odot \sigma_C(x_a(k+1)) \\
y(k) &= f_y(x_b(k), u(k), \theta_y)
\end{aligned}
$$

gate activation fcn $\sigma_G(\alpha) = \frac{1}{1+e^{-\alpha}}$, cell activation fcn $\sigma_C(\alpha) = \tanh(\alpha)$

- Training results (mean and std over 20 runs):

	BFR	Adam	EKF
RNN	training	89.12 (1.83)	**92.82** (0.33)
$n_\theta = 107$	test	85.51 (2.89)	**89.78** (0.58)
LSTM	training	89.60 (1.34)	92.63 (0.43)
$n_\theta = 139$	test	85.56 (2.68)	88.97 (1.31)

- EKF training **applicable to arbitrary classes** of black/gray box recurrent models!

TRAINING RNNS BY EKF - EXAMPLES

- Dataset: 2000 I/O data of linear system with **binary outputs**

$$
\begin{aligned}
x(k+1) &= \begin{bmatrix} .8 & .2 & -.1 \\ 0 & .9 & .1 \\ .1 & -.1 & .7 \end{bmatrix} x(k) + \begin{bmatrix} -1 \\ .5 \\ 1 \end{bmatrix} u(k) + \xi(k) \\
y(k) &= \begin{cases} 1 & \text{if } \begin{bmatrix} -2 & 1.5 & 0.5 \end{bmatrix} x(k) - 2 + \zeta(k) \geq 0 \\ 0 & \text{otherwise} \end{cases}
\end{aligned}
$$

$$\mathrm{Var}[\xi_i(k)] = \sigma^2$$

$$\mathrm{Var}[\zeta(k)] = \sigma^2$$

- N=1000 data used for training, 1000 for testing the model

- Train **linear state-space model** with 3 states and **sigmoidal output** function

$$f_1^y(y) = 1/(1 + e^{-A_1^y [x'(k)\ u(k)]' - b_1^y})$$

σ	EKF accuracy [%] test	EKF accuracy [%] training
0.000	98.02	97.91
0.001	95.33	98.66
0.010	97.99	98.52
0.100	94.56	95.44
0.200	93.71	92.22

- Training loss: (modified) **cross-entropy** loss

$$\ell_{\mathrm{CE}\epsilon}(y(k), \hat{y}) = \sum_{i=1}^{n_y} -y_i(k) \log(\epsilon + \hat{y}_i) - (1 - y_i(k)) \log(1 + \epsilon - \hat{y}_i)$$

LINEAR AND NONLINEAR IDENTIFICATION VIA L-BFGS

SYSTEM IDENTIFICATION PROBLEM

- Class of dynamical models with n_x states, n_u inputs, n_y outputs:

$$\begin{aligned} x_{k+1} &= Ax_k + Bu_k + f_x(x_k, u_k; \theta_x) \\ \hat{y}_k &= Cx_k + Du_k + f_y(x_k, u_k; \theta_y) \end{aligned}$$

Special cases:

linear model, RNN, ...

- Loss function (open-loop prediction error + regularization)

$$\begin{array}{ll} \min_{z, x_1, \ldots, x_{N-1}} & r(z) + \dfrac{1}{N} \displaystyle\sum_{k=0}^{N-1} \ell(y_k, Cx_k + Du_k + f_y(x_k, u_k; \theta_y)) \\ \text{s.t.} & x_{k+1} = Ax_k + Bu_k + f_x(x_k, u_k; \theta_x) \\ & k = 0, \ldots, N-2 \end{array}$$

$$z = \left[\begin{smallmatrix} x_0 \\ \Theta \end{smallmatrix}\right]$$

$$\Theta = \begin{bmatrix} A(:) \\ B(:) \\ C(:) \\ D(:) \\ \theta_x \\ \theta_y \end{bmatrix}$$

- Condense the problem by eliminating the hidden states x_k and get

$$\min_z f(z) + r(z)$$

(nonconvex) nonlinear programming (NLP) problem

NLP PROBLEM

- If f and r **differentiable**: use any state-of-the-art unconstrained NLP solver, e.g., **L-BFGS** (Limited-memory Broyden–Fletcher–Goldfarb–Shanno) (Liu, Nocedal, 1989)

- The gradient $\nabla f(z)$ can be computed efficiently by **automatic differentiation**

- However, sparsifying the model requires **non-smooth** regularizers:

$$r_1(z) = \tau \left\|z\right\|_1 \qquad\qquad r_g(z) = \tau_g \sum_{i=1}^{m} \|I_i z\|_2$$

ℓ_1-regularization   group-Lasso penalty

- Examples of **group-Lasso penalties**:

$m = n_x$ and I_i selected to reduce the **number of states**

$m = n_u$ and I_i selected to reduce the **number of inputs**

HANDLING NON-SMOOTH REGULARIZATION TERMS

(Bemporad, 2024)

1. If $r(x) = \sum_{i=1}^{n} r_i(x_i)$ and $r_i : \mathbb{R} \to \mathbb{R}$ is convex and positive semidefinite, the ℓ_1-regularized problem can be recast as a **bound-constrained NLP**:

$$\min_x f(x)+\tau\|x\|_1+r(x) \quad \rightarrow \quad \min_{y,z\geq 0} f(y-z)+\tau[1\ \dots\ 1]\left[\begin{smallmatrix} y\\ z\end{smallmatrix}\right]+r(y)+r(-z)$$

$$x^* = y^* - z^*$$

Example: $r(x) = \|x\|_2^2$ then $r(y) + r(-z) = \|\left[\begin{smallmatrix} y\\ z\end{smallmatrix}\right]\|_2^2$ well-regularized augmented problem

2. If $r(x)$ is convex and symmetric wrt each component x_i and increasing for $x \geq 0$, then we can solve instead

$$\min_{y,z\geq 0} f(y-z)+\tau[1\ \dots\ 1]+r(y+z)$$

differentiable for $y, z > 0$
if $r(x)$ differentiable for $x > 0$

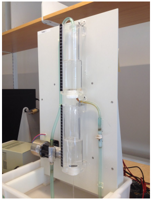
Example: $r(x)$ = group-Lasso penalty + constraint $y, z \geq \epsilon$ = machine precision

EXAMPLE: LINEAR SYSTEM IDENTIFICATION

- **Cascaded-Tanks benchmark**: (Schoukens, Mattson, Wigren, Noël, 2016)

$z = (A, B, C, D, x_0)$, mean-squared error loss + ℓ_2-regularization

	R^2 (training)			R^2 (test)			
n_x	`lbfgs`	`sippy`[3]	`MATLAB`[4]	`lbfgs`	`sippy`	`MATLAB`	
1	87.43	56.24	87.06	83.22	52.38	83.18	(`ssest`)
2	94.07	28.97	93.81	92.16	23.70	92.17	(`ssest`)
3	94.07	74.09	93.63	92.16	68.74	91.56	(`ssest`)
4	94.07	48.34	92.34	92.16	45.50	90.33	(`ssest`)
5	94.07	90.70	93.40	92.16	89.51	80.22	(`ssest`)
6	94.07	94.00	93.99	92.17	92.32	88.49	(`n4sid`)
7	94.07	92.47	93.82	92.17	90.81	< 0	(`ssest`)
8	94.49	< 0	94.00	89.49	< 0	< 0	(`n4sid`)
9	94.07	< 0	< 0	92.17	< 0	< 0	(`ssest`)
10	94.08	93.39	< 0	92.17	92.35	< 0	(`ssest`)



$n_y = n_u = 1$

1024 training data

1024 test data

(standard scaling)

CPU time: 2.4 s (lbfgs), 30 ms (sippy), 50 ms (n4sid/pred.), 0.3 s (n4sid/sim.), 0.5 s (ssest) [Apple M1 Max]

NLP with bounds solved in **JAX/JAXOPT** using the **L-BFGS-B** solver (Byrd, Lu, Nocedal, Zhu, 1995)

`pip install jax-sysid`

`github.com/bemporad/jax-sysid`

[3] (Armenise, Vaccari, Bacci Di Capaci, Pannocchia, 2018)

[4] (Ljung, SYS-ID Toolbox)

EXAMPLE: LINEAR SYSTEM IDENTIFICATION

- Synthetic data generated by the **cascaded 2x2 linear system**

$$x_{k+1} = \begin{bmatrix} 0.96 & 0.26 & 0.04 & 0 & 0 & 0 \\ -0.26 & 0.70 & 0.26 & 0 & 0 & 0 \\ 0 & 0 & 0.93 & 0.32 & 0.07 & 0 \\ 0 & 0 & -0.32 & 0.61 & 0.32 & 0 \\ 0 & 0 & 0 & 0 & 0.90 & 0.38 \\ 0 & 0 & 0 & 0 & -0.38 & 0.52 \end{bmatrix} x_k + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0.07 & 0 \\ 0.32 & 0 \\ 0 & 0.10 \\ 0 & 0.38 \end{bmatrix} u_k + \xi_k$$

$$y_k = \begin{bmatrix} x_1 \\ x_3 \end{bmatrix} + \eta_k$$

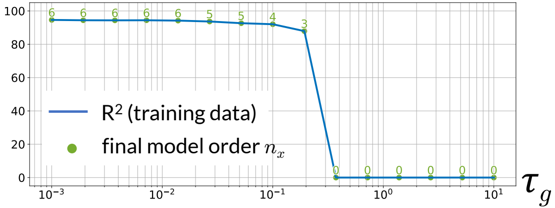
$\xi_{ki}, \eta_{kj} \in \mathcal{N}(0, 0.01)$

N=2000 training data

$\{(u_k, y_k)\}$

- Group-lasso penalty for **model-order reduction**:

$$\min_{\theta_x, \theta_y, x_0} \frac{1}{1000}\|z\|_2^2 + 10^{-16}\|z\|_1 + \tau_g \sum_{i=1}^{n_x} \left\| \begin{bmatrix} A'_{i,:} \\ A_{:,i} \\ B'_{i,:} \\ C_{:,i} \end{bmatrix} \right\|_2 + \frac{1}{N}\sum_{k=0}^{N-1} \|y_k - Cx_k\|_2^2$$



best results out of 10 runs

CPU time ≈ 3.85 s per run

[Apple M1 Max]

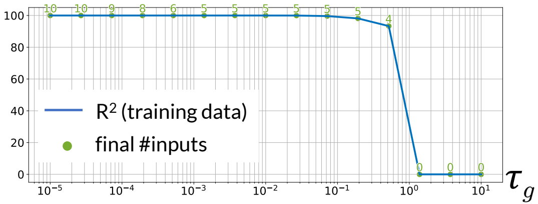
EXAMPLE: LINEAR SYSTEM IDENTIFICATION

- Synthetic data generated by a **random linear system** with $n_x = 3$ states, $n_u = 10$ inputs, $n_y = 1$ outputs, noise in $\mathcal{N}(0, 0.01)$, $N = 10000$ training data

- The last 5 columns of the B matrix are **1000x smaller** than the first 5

- Group-lasso penalty for **input selection**:

$$\min_{\theta_x,\theta_y,x_0} 10^{-8}\|z\|_2^2 + 10^{-16}\|z\|_1 + \tau_g \sum_{i=1}^{n_u} \|B_{:,i}\|_2 + \frac{1}{N}\sum_{k=0}^{N-1} \|y_k - Cx_k\|_2^2$$



best results out of 10 runs

CPU time ≈ 3.71 s per run

[Apple M1 Max]

- Can be useful to identify **Hammerstein models** using basis functions on u

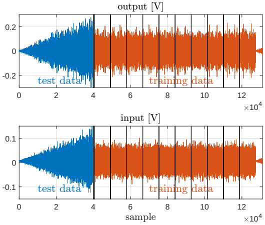
EXAMPLE: QUASI-LPV MODEL OF SILVERBOX BENCHMARK

(Bemporad, this talk)

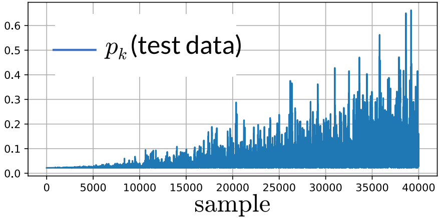
- **Quasi-LPV** model structure ($n_x = 8$ states):

$$
\begin{aligned}
x_{k+1} &= (A_0 + A_1 p_k)x_k + (B_0 + B_1 p_k)u_k \\
y_k &= Cx_k \\
p_k &= \mathrm{swish}(W_2\,\mathrm{swish}(W_1 x_k + b_1) + b_2)
\end{aligned}
$$

$$\mathrm{swish}(x) = \frac{x}{1+e^{-x}}$$



- Training setup:
  - ℓ_2-regularization ($\rho = 10^{-4}$)
  - warm start on first experiment (8,600 samples) 500 Adam + 500 L-BFGS iterations
  - 5000 L-BFGS iterations on full dataset (86,114 samples)
  - CPU time $\approx$ **265 s** [Apple M1 Max]



- RMSE on test data: **0.397 mV** (LTI model: **14.090 mV**)

  ($\|A_0\|_2 = 1.96$, $\|A_1\|_2 = 0.35$, $\|B_0\|_2 = 0.79$, $\|B_1\|_2 = 0.09$)

INDUSTRIAL ROBOT BENCHMARK

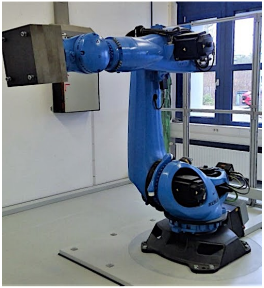
(Weigand, Götz, Ulmen, Ruskowski, 2022)

- KUKA KR300 R2500 ultra SE industrial robot, full robot movement

- **6 inputs** (torques), **6 outputs** (joint angles), w/ backlash, highly **nonlinear** and coupled, **slightly over-sampled** ($\|y_k - y_{k-1}\|$ is often very small)

- Identification benchmark dataset (forward model):
  - Sample time: $T_s = 100$ ms
  - N = 39988 training samples
  - N_{test} = 3636 test samples

- Most challenging benchmark on **`nonlinearbenchmark.org`**



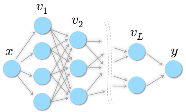
`nonlinearbenchmark.org`

RECURRENT NEURAL NETWORKS IN RESIDUAL FORM

(Bemporad, 2023 - *NLSYS-ID Benchmarks Workshop*)

- **Recurrent Neural Network** (RNN) model in **residual form**:

$$
\begin{aligned}
x_{k+1} &= Ax_k + Bu_k + f_x(x_k, u_k, \theta_x^i) \\
y_k &= Cx_k + f_y(x_k, \theta_y^i) \\
f_x, f_y &= \text{feedforward neural network}
\end{aligned}
$$



$$v_j = A_j f_{j-1}(v_{j-1}) + b_j$$

$$\theta = (A_1, b_1, \ldots, A_L, b_L)$$

- **Goal**: minimize **open-loop simulation error** under **elastic net** regularization

$$
\begin{aligned}
\min_{x_0, A, B, C, \theta_x, \theta_y} \ & \frac{1}{N} \sum_{k=1}^{N} \|y_k - \hat{y}_k\|_2^2 + \frac{1}{2}\rho(\|\theta_x\|_2^2 + \|\theta_y\|_2^2) + \tau(\|\theta_x\|_1 + \|\theta_y\|_1) \\
\text{s.t.} \ & \text{model equations}
\end{aligned}
$$

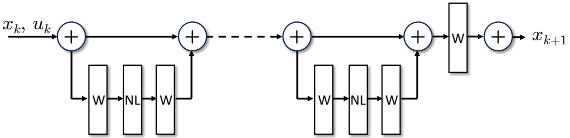
- **ℓ_1-regularization** introduced to reduce # model coefficients (=simpler model)

TRAINING RNN W/ ℓ_1-PENALTIES - INDUSTRIAL ROBOT

- Main **issues** with industrial robot benchmark:
  - **many parameters** to train, **large dataset** $\Rightarrow$ complex NLP
  - **high sensitivity** wrt weights (dynamics gets easily unstable)
  - **local minima** (solution depends on initial guess)
  - cannot easily use **mini-batches**: open-loop simulation cost is not separable, long-term memory effects present due to small sample time

- More general **residual networks** + ℓ_1/group-Lasso regularization possible

(Frascati, Bemporad, 2023)



SOLUTION APPROACH

(Bemporad, 2024)

1. Standard-scale I/O data for numerical reasons $u_i \leftarrow \frac{u_i - \mu_u^i}{\sigma_u^i}$, $y_i \leftarrow \frac{y_i - \mu_y^i}{\sigma_y^i}$
   $i = 1, \ldots, 6$

2. Train (A, B, C, x_0) by **jax-sysid** (1000 L-BFGS iters) w/o ℓ_1-regularization ($x \in \mathbb{R}^{12}$) (CPU time: 9.12 s) [Apple M1 Max]

   For comparison: **n4sid** takes 36.21 s and gives lower R^2-scores on training & test data in MATLAB
   sippy fails

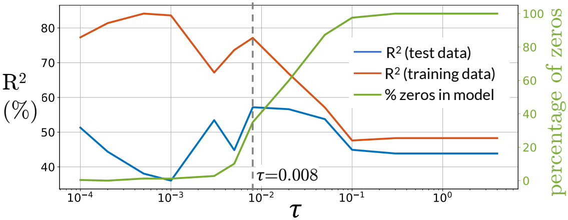
3. Fix (A, B, C) and train simple RESNET model with shallow NNs:

$$x_{k+1} = Ax_k + Bu_k + f_x(x_k, u_k, \theta_x), \qquad y_k = Cx_k + f_y(x_k, \theta_y)$$

- **Optimization**: to handle $\tau\|\theta\|_1$, use **jax-sysid** running 2000 **Adam** iters first (for warm-start) and then 2000 **L-BGFS-B** iters

INDUSTRIAL ROBOT BENCHMARK: RESULTS

- State $x \in \mathbb{R}^{12}$, f_x, f_y with **36** and **24** neurons, **swish** activation fcn $\frac{x}{1+e^{-x}}$
- Total number of training parameters: $\dim(\theta_x) + \dim(\theta_y) = 1590$



(best R^2 in 30 runs)

- Model quality measured by **average R^2-score** on all outputs:

$$R^2 = \frac{1}{n_y} \sum_{i=1}^{n_y} 100 \left(1 - \frac{\sum_{k=1}^{N} (y_{k,i} - \hat{y}_{k,i|0})^2}{\sum_{k=1}^{N} (y_{k,i} - \frac{1}{N} \sum_{i=1}^{N} y_{k,i})^2}\right)$$

- Training time $\approx$ 12 min on a single core per run
  (3192 variables, 2000 Adam iterations + 2000 L-BFGS-B iterations, Apple M1 Max CPU)

INDUSTRIAL ROBOT BENCHMARK: RESULTS

- **Open-loop simulation** errors ($\rho = 0.01, \tau = 0.008$):

	R^2 (training) RNN model	R^2 (test) RNN model	R^2 (training) linear model	R^2 (test) linear model	
average	77.1493	57.1784	48.2789	43.8573	**jax-sysid**
			39.2822	32.0410	**n4sid**

- More parameters/smaller regularization leads to overfitting training data

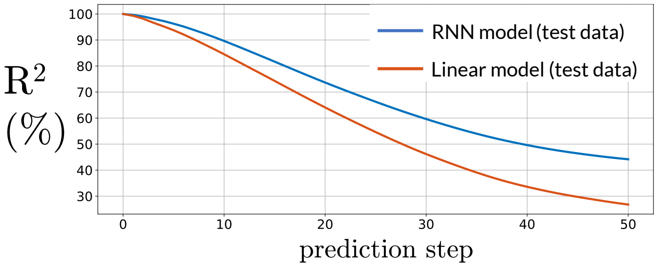
- **Pure Adam** vs **LBFG-B+Adam** vs **OWL-QN** (Andrew, Gao, 2007): ($\tau = 0.008$)

solver	adam iters	fcn evals	$\overline{R^2}$ training	$\overline{R^2}$ test	# zeros (θ_x, θ_y)	CPU time (s)
L-BFGS-B	2000	2000	**77.1493**	**57.1784**	556/1590	**309.87**
OWL-QN	2000	2000	74.7816	54.0531	**736/1590**	449.17
Adam	6000	0	71.0687	54.3636	1/1590	389.39

- Adam is unable to sparsify the model

INDUSTRIAL ROBOT BENCHMARK: RESULTS

- Compute p-step ahead prediction $\hat{y}_{k+p|k}$, with hidden state $x_{k|k}$ estimated by an Extended Kalman Filter based on identified RNN model



- This is a more relevant indicator of model quality for MPC purposes than open-loop simulation error $\hat{y}_{k|0} - y_k$

CONCLUSIONS

CONCLUSIONS

- **Quasi-Newton** methods for SYS-ID enabled by powerful **autodiff libraries**

PyTorch CasADi Zygote

  - ☺ **Extremely flexible** (model structure, loss functions, regularization terms)
  - ☺ **Faster convergence/better models** than with classical GD methods (like Adam)
  - ☺ Numerically very **robust** (even to get **linear state-space models**!)
  - 😐 **Non-convex problem**: multiple runs often required from different initial guesses

- **Open research topics**:
  - ? How to get good-quality training data (**active learning**)
  - ? More efficient methods for **non-smooth nonlinear optimization**

2023 ERC Advanced Grant "COMPACT"

Starting soon … (postdoc positions available)