Sparse neural networks for forward and inverse estimation

Bojana Rosic, Bram van de Weg,

and Wouter van Dijk

(b.rosic@utwente.nl)

Applied Mechanics and Data Analysis

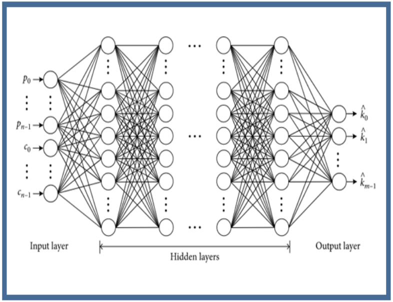
Faculty of Engineering Technology

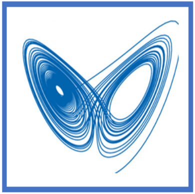
University of Twente, Netherlands

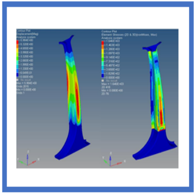
UNIVERSITY OF TWENTE.

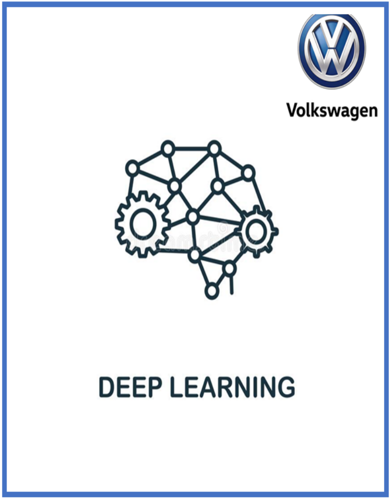
Motivation

Goal: build self-learning neural network based meta-models



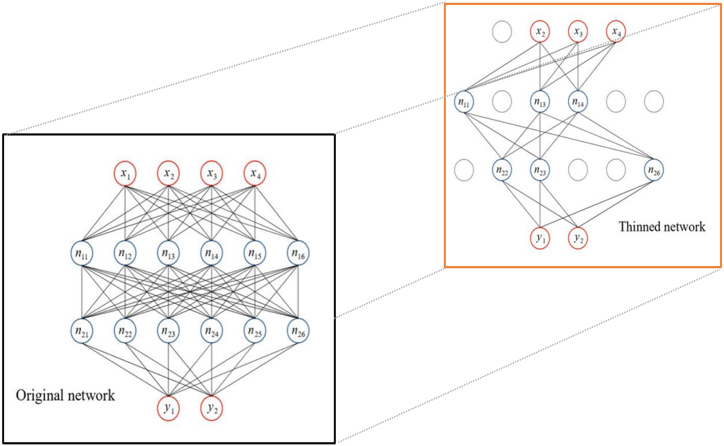






Motivation

Goal: build **self-learning sparse** neural network based meta-models



NN for forward and inverse problems

Inverse problem

Given noisy observations

$$y_{n+1} = Hz_{n+1} + \varepsilon_{n+1}$$

find the uknown z *and* q *modelled by*

$$z_{n+1} = G(z_n, q) + \eta_n, \quad n \in \mathbb{N}$$

in predefined time interval $[0, T]$.

NN for forward and inverse problems

Inverse problem

Given noisy observations

$$y_{n+1} = H z_{n+1} + \varepsilon_{n+1}$$

find the uknown z *and* q *modelled by*

$$z_{n+1} = G(z_n, q) + \eta_n, \quad n \in \mathbb{N}$$

in predefined time interval $[0, T]$.

- the operator $G \in \mathcal{C}(\mathbb{R}^d \times \mathbb{R}^p \mapsto \mathbb{R}^d)$
- the operator $H \in \mathcal{L}(\mathbb{R}^d \mapsto \mathbb{R}^m)$
- $(\varepsilon_n), (\eta_n)$ are i.i.d sequences

NN for forward and inverse problems

Inverse problem

Given noisy observations

$$y_{n+1} = H z_{n+1} + \varepsilon_{n+1}$$

find the uknown z *and* q *modelled by*

$$z_{n+1} = G(z_n, q) + \eta_n, \quad n \in \mathbb{N}$$

in predefined time interval $[0, T]$.

- the operator $G \in \mathcal{C}(\mathbb{R}^d \times \mathbb{R}^p \mapsto \mathbb{R}^d)$
- the operator $H \in \mathcal{L}(\mathbb{R}^d \mapsto \mathbb{R}^m)$
- $(\varepsilon_n), (\eta_n)$ are i.i.d sequences

We may distinguish: state and parameter estimation problems.

NN for forward and inverse problems

In a Bayesian setting one may model q as uncertain following $q_f(\omega) \sim p(q)$ and estimate

$$p(q|y_0, ..., y_N) \propto p(y_0, ..., y_N|q)p(q).$$

NN for forward and inverse problems

In a Bayesian setting one may model q as uncertain following $q_f(\omega) \sim p(q)$ and estimate

$$p(q|y_0, ..., y_N) \propto p(y_0, ..., y_N|q)p(q).$$

For an efficient estimation of the posterior we need two steps:

- Forecast (prediction, uncertainty propagation) step

$$\text{Map } \phi : q_f(\omega) \mapsto y_{n,f}(\omega)$$

- Assimilation (update) phase

$$\text{Map } \varphi : y_{n,f}(\omega) \mapsto q_f(\omega)$$

NN for forward and inverse problems

In a Bayesian setting one may model q as uncertain following $q_f(\omega) \sim p(q)$ and estimate

$$p(q|y_0, ..., y_N) \propto p(y_0, ..., y_N|q)p(q).$$

For an efficient estimation of the posterior we need two steps:

- Forecast (prediction, uncertainty propagation) step

$$\text{Map } \phi : q_f(\omega) \mapsto y_{n,f}(\omega)$$

- Assimilation (update) phase

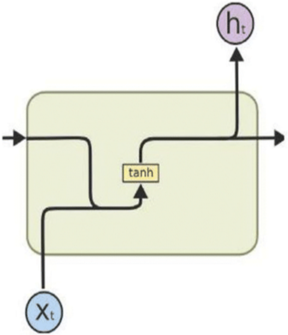
$$\text{Map } \varphi : y_{n,f}(\omega) \mapsto q_f(\omega)$$

Both of these maps can be approximated by neural networks.

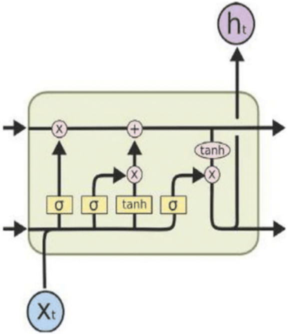
Time-dependent neural networks

In this talk we will look at:

- standard recurrent neural network
- standard long-short term memory network [a]



(a) RNN



(b) LSTM

[a] pics @towardsdatascience

Time-dependent neural network

Let be given the nonlinear dynamical system described by

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{x}_t, \boldsymbol{u}(t), t)$$

in which $\boldsymbol{x} \in \mathbb{R}^d$ is the state of the system, $\boldsymbol{u} \in \mathbb{R}^d$ is the input

Time-dependent neural network

Let be given the nonlinear dynamical system described by

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{x}_t, \boldsymbol{u}(t), t)$$

in which $\boldsymbol{x} \in \mathbb{R}^d$ is the state of the system, $\boldsymbol{u} \in \mathbb{R}^d$ is the input , and

$$\boldsymbol{x}_t = \{\boldsymbol{x}(\tau) : \tau \leq t\}$$

represents the trajectory of the solution in the past.

Time-dependent neural network

Let be given the nonlinear dynamical system described by

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{x}_t, \boldsymbol{u}(t), t)$$

in which $\boldsymbol{x} \in \mathbb{R}^d$ is the state of the system, $\boldsymbol{u} \in \mathbb{R}^d$ is the input , and

$$\boldsymbol{x}_t = \{\boldsymbol{x}(\tau) : \tau \leq t\}$$

represents the trajectory of the solution in the past.

For the discrete delay one has:

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{x}(t-\tau_1), ..., \boldsymbol{x}(t-\tau_m), \boldsymbol{u}(t), t)$$

in which $\tau_1 > \cdots > \tau_m \geq 0$ denotes the memory of the system.

Time dependent neural network

In a special linear case [Sherstinsky,2020] the previous system can be decoupled to

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{A}\boldsymbol{x}(t) + \boldsymbol{B}\boldsymbol{h}(t - \tau_0) + \boldsymbol{C}\boldsymbol{u}(t) + \boldsymbol{a}, \quad \boldsymbol{h} = \boldsymbol{g}(\boldsymbol{x}(t - \tau_0))$$

in which $\boldsymbol{g}$ is a nonlinear, saturating, and invertible function of a state.

Time dependent neural network

In a special linear case [Sherstinsky,2020] the previous system can be decoupled to

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{A}\boldsymbol{x}(t) + \boldsymbol{B}\boldsymbol{h}(t-\tau_0) + \boldsymbol{C}\boldsymbol{u}(t) + \boldsymbol{a}, \quad \boldsymbol{h} = \boldsymbol{g}(\boldsymbol{x}(t-\tau_0))$$

in which $\boldsymbol{g}$ is a nonlinear, saturating, and invertible function of a state.

Taking $\tau_0 = \Delta t$, and after the discretization by the implicit Euler technique, one may rewrite the previous system by

$$\boldsymbol{x}_n = \boldsymbol{W}_x \boldsymbol{x}_{n-1} + \boldsymbol{W}_h \boldsymbol{h}_{n-1} + \boldsymbol{W}_u \boldsymbol{u}_n + \boldsymbol{b}$$

$$\boldsymbol{h}_n = \boldsymbol{g}(\boldsymbol{x}_n)$$

Time dependent neural network

In a special linear case [Sherstinsky,2020] the previous system can be decoupled to

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{A}\boldsymbol{x}(t) + \boldsymbol{B}\boldsymbol{h}(t-\tau_0) + \boldsymbol{C}\boldsymbol{u}(t) + \boldsymbol{a}, \quad \boldsymbol{h} = \boldsymbol{g}(\boldsymbol{x}(t-\tau_0))$$

in which $\boldsymbol{g}$ is a nonlinear, saturating, and invertible function of a state.

Taking $\tau_0 = \Delta t$, and after the discretization by the implicit Euler technique, one may rewrite the previous system by

$$\boldsymbol{x}_n = \boldsymbol{W}_x \boldsymbol{x}_{n-1} + \boldsymbol{W}_h \boldsymbol{h}_{n-1} + \boldsymbol{W}_u \boldsymbol{u}_n + \boldsymbol{b}$$

$$\boldsymbol{h}_n = \boldsymbol{g}(\boldsymbol{x}_n)$$

in which

$$\boldsymbol{W}_x = (I - \Delta t \boldsymbol{A})^{-1}, \quad \boldsymbol{W}_h = \Delta t \boldsymbol{W}_x \boldsymbol{B}$$

$$\boldsymbol{W}_u = \Delta t \boldsymbol{W}_x \boldsymbol{C}, \quad \boldsymbol{b} = \Delta t \boldsymbol{W}_s \boldsymbol{a}.$$

Stability

In order that

$$\boldsymbol{x}_n = \boldsymbol{W}_x \boldsymbol{x}_{n-1} + \boldsymbol{W}_h \boldsymbol{h}_{n-1} + \boldsymbol{W}_u \boldsymbol{u}_n + \boldsymbol{a}$$

$$\boldsymbol{h}_n = \boldsymbol{g}(\boldsymbol{x}_n)$$

is stable every eignevalue of $\boldsymbol{W}_s := \boldsymbol{W}_x + \boldsymbol{W}_h$ must lie within the complex-valued unit circle:

Stability

In order that

$$\boldsymbol{x}_n = \boldsymbol{W}_x \boldsymbol{x}_{n-1} + \boldsymbol{W}_h \boldsymbol{h}_{n-1} + \boldsymbol{W}_u \boldsymbol{u}_n + \boldsymbol{a}$$

$$\boldsymbol{h}_n = \boldsymbol{g}(\boldsymbol{x}_n)$$

is stable every eignevalue of $\boldsymbol{W}_s := \boldsymbol{W}_x + \boldsymbol{W}_h$ must lie within the complex-valued unit circle:

- for $a_{ii} << 0, a_{i \neq j} = 0$ then $\boldsymbol{W}_x = (I - \Delta t \boldsymbol{A})^{-1} \approx -\boldsymbol{A}^{-1} \approx \boldsymbol{0}$

Stability

In order that

$$\boldsymbol{x}_n = \boldsymbol{W}_x \boldsymbol{x}_{n-1} + \boldsymbol{W}_h \boldsymbol{h}_{n-1} + \boldsymbol{W}_u \boldsymbol{u}_n + \boldsymbol{a}$$

$$\boldsymbol{h}_n = \boldsymbol{g}(\boldsymbol{x}_n)$$

is stable every eignevalue of $\boldsymbol{W}_s := \boldsymbol{W}_x + \boldsymbol{W}_h$ must lie within the complex-valued unit circle:

- for $a_{ii} << 0, a_{i \neq j} = 0$ then $\boldsymbol{W}_x = (I - \Delta t \boldsymbol{A})^{-1} \approx -\boldsymbol{A}^{-1} \approx \boldsymbol{0}$
- for $\boldsymbol{B} = \boldsymbol{B}^T, \boldsymbol{B} = \boldsymbol{V}_B \boldsymbol{\Lambda}_B \boldsymbol{V}_B^T$ then the stability comes from

$$\boldsymbol{W}_s := -\boldsymbol{A}^{-1} \boldsymbol{B} = -(\boldsymbol{V}_B^T \boldsymbol{A}^{-1})(\boldsymbol{V}_B \boldsymbol{\Lambda}_B).$$

Stability

In order that

$$\boldsymbol{x}_n = \boldsymbol{W}_x \boldsymbol{x}_{n-1} + \boldsymbol{W}_h \boldsymbol{h}_{n-1} + \boldsymbol{W}_u \boldsymbol{u}_n + \boldsymbol{a}$$

$$\boldsymbol{h}_n = \boldsymbol{g}(\boldsymbol{x}_n)$$

is stable every eignevalue of $\boldsymbol{W}_s := \boldsymbol{W}_x + \boldsymbol{W}_h$ must lie within the complex-valued unit circle:

- for $a_{ii} << 0, a_{i \neq j} = 0$ then $\boldsymbol{W}_x = (I - \Delta t \boldsymbol{A})^{-1} \approx -\boldsymbol{A}^{-1} \approx \mathbf{0}$
- for $\boldsymbol{B} = \boldsymbol{B}^T, \boldsymbol{B} = \boldsymbol{V}_B \boldsymbol{\Lambda}_B \boldsymbol{V}_B^T$ then the stability comes from

$$\boldsymbol{W}_s := -\boldsymbol{A}^{-1} \boldsymbol{B} = -(\boldsymbol{V}_B^T \boldsymbol{A}^{-1})(\boldsymbol{V}_B \boldsymbol{\Lambda}_B).$$

Thus, sufficient condition for stability is

$$0 < \lambda_i (|a_{ii}|)^{-1} < 1, \quad \text{i.e. } 0 < \lambda_i < |a_{ii}|$$

Reccurent neural cell

As $\boldsymbol{W}_x$ vanishes, the previously described dynamical system becomes

$$\boldsymbol{x}_n = \boldsymbol{W}_h \boldsymbol{h}_{n-1} + \boldsymbol{W}_u \boldsymbol{u}_n + \boldsymbol{b}$$

$$\boldsymbol{h}_n = \boldsymbol{g}(\boldsymbol{x}_n).$$

Reccurent neural cell

As $\boldsymbol{W}_x$ vanishes, the previously described dynamical system becomes

$$\boldsymbol{x}_n = \boldsymbol{W}_h \boldsymbol{h}_{n-1} + \boldsymbol{W}_u \boldsymbol{u}_n + \boldsymbol{b}$$

$$\boldsymbol{h}_n = \boldsymbol{g}(\boldsymbol{x}_n).$$

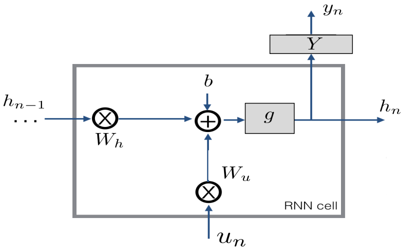
This matches the definition of the standard recurrent neural cell (i.e. the evolution of a hidden state)

$$\boldsymbol{h}_n = \boldsymbol{g}(\boldsymbol{W}_h \boldsymbol{h}_{n-1} + \boldsymbol{W}_u \boldsymbol{u}_n + \boldsymbol{b})$$

with the output (observable) defined as

$$\boldsymbol{y}_n = Y(\boldsymbol{h}_n, \boldsymbol{w}_y)$$

in which Y is possibly a nonlinear observation operator.



Reccurent neural network

Collecting

$$\boldsymbol{w} := \{\boldsymbol{W}_h, \boldsymbol{W}_u, \boldsymbol{b}, \boldsymbol{w}_y\}$$

and applying recurrency in the time interval $[0, \Delta t, ..., n\Delta t]$, one can further introduce a neural network (NN) as a composition of n functions

$$\boldsymbol{g}_n(\boldsymbol{h}, \boldsymbol{u}, \boldsymbol{w}) := \boldsymbol{g}(\boldsymbol{W}_h \boldsymbol{h}_{n-1} + \boldsymbol{W}_u \boldsymbol{u}_n + \boldsymbol{b})$$

Reccurent neural network

Collecting

$$\boldsymbol{w} := \{\boldsymbol{W}_h, \boldsymbol{W}_u, \boldsymbol{b}, \boldsymbol{w}_y\}$$

and applying recurrency in the time interval $[0, \Delta t, ..., n\Delta t]$, one can further introduce a neural network (NN) as a composition of n functions

$$\boldsymbol{g}_n(\boldsymbol{h}, \boldsymbol{u}, \boldsymbol{w}) := \boldsymbol{g}(\boldsymbol{W}_h \boldsymbol{h}_{n-1} + \boldsymbol{W}_u \boldsymbol{u}_n + \boldsymbol{b})$$

such that NN reads

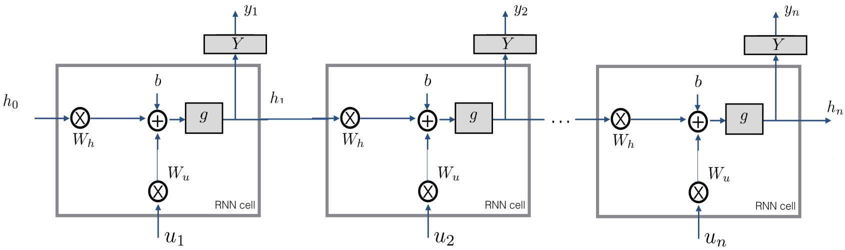
$$\boldsymbol{F}(\boldsymbol{h}, \boldsymbol{u}, \boldsymbol{w}) := (\boldsymbol{Y}_n \circ \boldsymbol{g}_{n-1} \circ \boldsymbol{g}_{n-2} \circ ... \circ \boldsymbol{g}_1)(\boldsymbol{h}, \boldsymbol{u}, \boldsymbol{w}).$$

Here,

$$Y_n(\boldsymbol{h}, \boldsymbol{u}, \boldsymbol{w}) := Y(\boldsymbol{h}_n, \boldsymbol{w}_y).$$

Reccurent neural network

As all cells share weights, we have significant reduction of the parametrisation compared to the feedforward network.



Offline gradient descent

The goal is to estimate $\boldsymbol{w}$ given data $(\boldsymbol{y}_i)_{i=1,n}$ such that

$$\boldsymbol{w}^* = \arg\min \boldsymbol{J}(\boldsymbol{w}), \quad \boldsymbol{J}(\boldsymbol{w}) := \sum_{i=1}^{n} \frac{1}{2} \langle \boldsymbol{y}_i - \hat{\boldsymbol{y}}_i(\boldsymbol{w}), \boldsymbol{y}_i - \underbrace{\hat{\boldsymbol{y}}_i(\boldsymbol{w})}_{NN} \rangle$$

is minimized. The gradient based approach

Offline gradient descent

The goal is to estimate $\boldsymbol{w}$ given data $(\boldsymbol{y}_i)_{i=1,n}$ such that

$$\boldsymbol{w}^* = \arg\min \boldsymbol{J}(\boldsymbol{w}), \quad \boldsymbol{J}(\boldsymbol{w}) := \sum_{i=1}^{n} \frac{1}{2} \langle \boldsymbol{y}_i - \hat{\boldsymbol{y}}_i(\boldsymbol{w}), \boldsymbol{y}_i - \underbrace{\hat{\boldsymbol{y}}_i(\boldsymbol{w})}_{NN} \rangle$$

is minimized. The gradient based approach

$$\boldsymbol{w} = \boldsymbol{w} - \alpha \frac{\partial \boldsymbol{J}}{\partial \boldsymbol{w}}$$

then requires estimation of the gradient $\boldsymbol{J}$ that depends on

$$\|\boldsymbol{W}_h\|^{\ell-n} \|\boldsymbol{g}'(\boldsymbol{z})\|^{\ell-n},$$

and thus on the properties of both $\|\boldsymbol{W}_h\|$ and $\|\boldsymbol{g}'(\boldsymbol{z})\|$.

Offline gradient descent

Looking at

$$\| \|\boldsymbol{W}_h\|^{\ell-n} \|\boldsymbol{g}'(\boldsymbol{z})\|^{\ell-n}$$

one may have the following scenarios

Offline gradient descent

Looking at

$$\||W_h\|^{\ell-n}\|g'(z)\|^{\ell-n}$$

one may have the following scenarios

- If all $0 < \lambda_i(W_h) < 1$ then $\||W_h\| < 1$, and if $\|g'(z)\| < 1$ then **the gradient vanishes**.

Offline gradient descent

Looking at

$$\||\boldsymbol{W}_h\|^{\ell-n}\|\boldsymbol{g}'(\boldsymbol{z})\|^{\ell-n}$$

one may have the following scenarios

- If all $0 < \lambda_i(\boldsymbol{W}_h) < 1$ then $\||\boldsymbol{W}_h\| < 1$, and if $\|\boldsymbol{g}'(\boldsymbol{z})\| < 1$ then **the gradient vanishes**.
- If any $\lambda_i(\boldsymbol{W}_h) > 1$ then the term $\|\boldsymbol{W}_h\|$ will exponentially grow, and thus two scenarios:

Offline gradient descent

Looking at

$$\|\|\boldsymbol{W}_h\|^{\ell-n}\|\boldsymbol{g}'(\boldsymbol{z})\|^{\ell-n}$$

one may have the following scenarios

- If all $0 < \lambda_i(\boldsymbol{W}_h) < 1$ then $\|\|\boldsymbol{W}_h\| < 1$, and if $\|\boldsymbol{g}'(\boldsymbol{z})\| < 1$ then **the gradient vanishes**.
- If any $\lambda_i(\boldsymbol{W}_h) > 1$ then the term $\|\boldsymbol{W}_h\|$ will exponentially grow, and thus two scenarios:
    - If $\|\boldsymbol{g}'(\boldsymbol{z})\| = \boldsymbol{0}$ (the flat regions of the activation function), then **the gradient vanishes**

Offline gradient descent

Looking at

$$\|\|\boldsymbol{W}_h\|^{\ell-n}\|\boldsymbol{g}'(\boldsymbol{z})\|^{\ell-n}$$

one may have the following scenarios

- If all $0 < \lambda_i(\boldsymbol{W}_h) < 1$ then $\|\|\boldsymbol{W}_h\| < 1$, and if $\|\boldsymbol{g}'(\boldsymbol{z})\| < 1$ then **the gradient vanishes**.
- If any $\lambda_i(\boldsymbol{W}_h) > 1$ then the term $\|\boldsymbol{W}_h\|$ will exponentially grow, and thus two scenarios:
  - If $\|\boldsymbol{g}'(\boldsymbol{z})\| = \boldsymbol{0}$ (the flat regions of the activation function), then **the gradient vanishes**
  - If $\|\boldsymbol{g}'(\boldsymbol{z})\| \neq \boldsymbol{0}$ (quasi-linear regions of the activation function), then **the gradient explodes.**

Offline gradient descent

Looking at

$$\| \|\boldsymbol{W}_h\|^{\ell-n} \|\boldsymbol{g}'(\boldsymbol{z})\|^{\ell-n}$$

one may have the following scenarios

- If all $0 < \lambda_i(\boldsymbol{W}_h) < 1$ then $\| \|\boldsymbol{W}_h\| < 1$, and if $\|\boldsymbol{g}'(\boldsymbol{z})\| < 1$ then **the gradient vanishes**.
- If any $\lambda_i(\boldsymbol{W}_h) > 1$ then the term $\|\boldsymbol{W}_h\|$ will exponentially grow, and thus two scenarios:
  - If $\|\boldsymbol{g}'(\boldsymbol{z})\| = \boldsymbol{0}$ (the flat regions of the activation function), then **the gradient vanishes**
  - If $\|\boldsymbol{g}'(\boldsymbol{z})\| \neq \boldsymbol{0}$ (quasi-linear regions of the activation function), then **the gradient explodes.**

Thus, the RNNs suffer from the so-called gradient problem when used in long term integration.

Long-short term memory cell

To make the system robust one may generalize previous equation to

$$\boldsymbol{x}_n = \boldsymbol{g}_{cx}(n) \odot (\boldsymbol{W}_x \boldsymbol{x}_{n-1}) + \boldsymbol{g}_c(n) \odot \boldsymbol{g}(\boldsymbol{s}_n)$$

in which

$$\boldsymbol{s}_n := \boldsymbol{W}_h \boldsymbol{v}_{n-1} + \boldsymbol{g}_{cu}(n) \odot \boldsymbol{W}_u \boldsymbol{u}_n + \boldsymbol{b}$$

and

$$\boldsymbol{v}_{n-1} := \boldsymbol{g}_{ch}(n) \odot \boldsymbol{h}_{n-1}.$$

Long-short term memory cell

To make the system robust one may generalize previous equation to

$$\boldsymbol{x}_n = \boldsymbol{g}_{cx}(n) \odot (\boldsymbol{W}_x \boldsymbol{x}_{n-1}) + \boldsymbol{g}_c(n) \odot \boldsymbol{g}(\boldsymbol{s}_n)$$

in which

$$\boldsymbol{s}_n := \boldsymbol{W}_h \boldsymbol{v}_{n-1} + \boldsymbol{g}_{cu}(n) \odot \boldsymbol{W}_u \boldsymbol{u}_n + \boldsymbol{b}$$

and

$$\boldsymbol{v}_{n-1} := \boldsymbol{g}_{ch}(n) \odot \boldsymbol{h}_{n-1}.$$

Here, controls are continuous, differentiable, monotonically increasing functions that map the domain $(-\infty, \infty)$ into the range $(0, 1)$ (e.g. logistic function), i.e. $\mathbf{0} \leq \boldsymbol{g}_{cx}(n), \boldsymbol{g}_{cu}(n), \boldsymbol{g}_c(n) \leq \mathbf{1}$.

Long-short term memory cell

To make the system robust one may generalize previous equation to

$$\boldsymbol{x}_n = \boldsymbol{g}_{cx}(n) \odot (\boldsymbol{W}_x \boldsymbol{x}_{n-1}) + \boldsymbol{g}_c(n) \odot \boldsymbol{g}(\boldsymbol{s}_n)$$

in which

$$\boldsymbol{s}_n := \boldsymbol{W}_h \boldsymbol{v}_{n-1} + \boldsymbol{g}_{cu}(n) \odot \boldsymbol{W}_u \boldsymbol{u}_n + \boldsymbol{b}$$

and

$$\boldsymbol{v}_{n-1} := \boldsymbol{g}_{ch}(n) \odot \boldsymbol{h}_{n-1}.$$

Here, controls are continuous, differentiable, monotonically increasing functions that map the domain $(-\infty, \infty)$ into the range $(0, 1)$ (e.g. logistic function), i.e. $\boldsymbol{0} \leq \boldsymbol{g}_{cx}(n), \boldsymbol{g}_{cu}(n), \boldsymbol{g}_c(n) \leq \boldsymbol{1}$. Taking $\boldsymbol{W}_x = \boldsymbol{I}$ one obtains

$$\boldsymbol{x}_n = \boldsymbol{g}_{cx}(n) \odot \boldsymbol{x}_{n-1} + \boldsymbol{g}_{cu}(n) \odot \boldsymbol{g}(\boldsymbol{s}_n)$$

which is a core constituent of the set of formulas defining the cell of the Vanilla LSTM network [Hochreiter and Schmidhuber,1997].

Long-short term memory cell

Thus,

$$\boldsymbol{x}_n = \boldsymbol{g}_{cx}(n) \odot \boldsymbol{x}_{n-1} + \boldsymbol{g}_{cu}(n) \odot \boldsymbol{g}(\boldsymbol{s}_n)$$

$$\boldsymbol{h}_n = g(\boldsymbol{x}_n)$$

in which we choose

$$\boldsymbol{g}_{cx}(n) = \boldsymbol{g}_a(\hat{\boldsymbol{W}}_x \boldsymbol{x}_n + \hat{\boldsymbol{W}}_h \boldsymbol{h}_{n-1} + \boldsymbol{b}_{cx})$$

$$\boldsymbol{g}_{cu}(n) = \boldsymbol{g}_a(\tilde{\boldsymbol{W}}_x \boldsymbol{x}_n + \tilde{\boldsymbol{W}}_h \boldsymbol{h}_{n-1} + \boldsymbol{b}_{cu})$$

$$\boldsymbol{g}_c(n) = \boldsymbol{g}_a(\bar{\boldsymbol{W}}_x \boldsymbol{x}_n + \bar{\boldsymbol{W}}_h \boldsymbol{h}_{n-1} + \boldsymbol{b}_c)$$

This matches the definition of the standard LSTM cell with the output (observable) defined as

$$\boldsymbol{y}_n = Y(\boldsymbol{h}_n, \boldsymbol{w}_y)$$

in which Y is a possibly nonlinear observation operator.

Offline gradient descent

Collecting all unknown parameters to

$$\boldsymbol{w} := \{\hat{\boldsymbol{W}}_x, \hat{\boldsymbol{W}}_h, \tilde{\boldsymbol{W}}_x, \tilde{\boldsymbol{W}}, ...\}$$

the goal is to estimate $\boldsymbol{w}$ given data $(\boldsymbol{u}_n, \boldsymbol{y}_n)$ such that

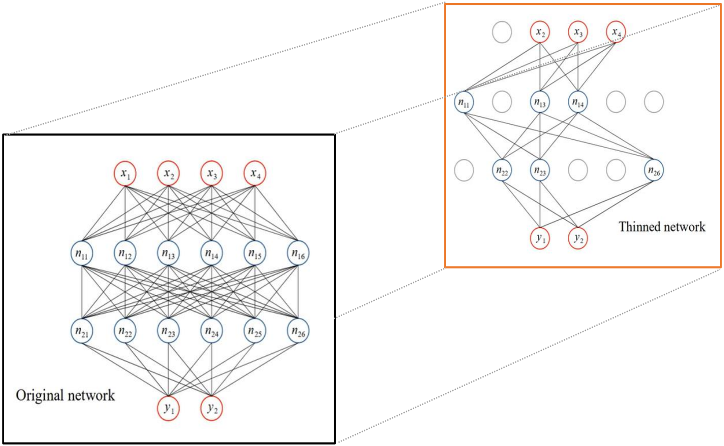
$$\boldsymbol{w}^* = \arg\min \boldsymbol{J}(\boldsymbol{w}), \quad \boldsymbol{J}(\boldsymbol{w}) := \sum_{i=1}^{n} \frac{1}{2} \langle \boldsymbol{y}_i - \hat{\boldsymbol{y}}_i(\boldsymbol{w}), \boldsymbol{y}_i - \hat{\boldsymbol{y}}_i(\boldsymbol{w}) \rangle$$

is minimized by using the gradient based approach

$$\boldsymbol{w} = \boldsymbol{w} - \alpha \frac{\partial \boldsymbol{J}}{\partial \boldsymbol{w}}.$$

But, we dont have sparsity...

and cannot include noise in data, or in the input...



Stochastic RNN formulation

Let the unknown weights $\boldsymbol{w}$ be modelled as uncertain, i.e.

$$\boldsymbol{w}(\omega_w) \in L_2(\Omega_w, \mathfrak{F}_w, \mathbb{P}_w)$$

Stochastic RNN formulation

Let the unknown weights $\boldsymbol{w}$ be modelled as uncertain, i.e.

$$\boldsymbol{w}(\omega_w) \in L_2(\Omega_w, \mathfrak{F}_w, \mathbb{P}_w)$$

such that the RNN cell based dynamical system becomes stochastic

$$\hat{\boldsymbol{x}}_n(\omega_w) = \boldsymbol{W}_h(\omega_w)\hat{\boldsymbol{h}}_{n-1}(\omega_w) + \boldsymbol{W}_u(\omega_w)\boldsymbol{u}_n(\omega_w) + \boldsymbol{b}(\omega_w)$$

$$\hat{\boldsymbol{h}}_n(\omega_w) = \boldsymbol{g}(\hat{\boldsymbol{x}}_n(\omega_w))$$

Stochastic RNN formulation

Let the unknown weights $\boldsymbol{w}$ be modelled as uncertain, i.e.

$$\boldsymbol{w}(\omega_w) \in L_2(\Omega_w, \mathfrak{F}_w, \mathbb{P}_w)$$

such that the RNN cell based dynamical system becomes stochastic

$$\hat{\boldsymbol{x}}_n(\omega_w) = \boldsymbol{W}_h(\omega_w)\hat{\boldsymbol{h}}_{n-1}(\omega_w) + \boldsymbol{W}_u(\omega_w)\boldsymbol{u}_n(\omega_w) + \boldsymbol{b}(\omega_w)$$

$$\hat{\boldsymbol{h}}_n(\omega_w) = \boldsymbol{g}(\hat{\boldsymbol{x}}_n(\omega_w))$$

with the output (observable) defined as

$$\hat{\boldsymbol{y}}_n(\omega_w) = Y(\hat{\boldsymbol{h}}_n(\omega_w), \boldsymbol{w}_y(\omega_w)) + \boldsymbol{\varepsilon}_n(\omega_\varepsilon)$$

in which $\varepsilon_n(\omega_\varepsilon)$ is the prediction of the cell-modelling and/or observation error, here assumed to be independent of $\boldsymbol{w}(\omega_w)$.

Stochastic RNN cell: forward pass

Given $\boldsymbol{w}(\omega_w), \boldsymbol{\varepsilon}(\omega_\varepsilon)$ and

$$\Omega := \Omega_w \times \Omega_\varepsilon, \mathfrak{F} := \sigma(\mathfrak{F}_w \times \mathfrak{F}_\varepsilon), \mathbb{P} = \mathbb{P}_w \mathbb{P}_\varepsilon$$

we may estimate the predicted values

$$\hat{\boldsymbol{x}}_n(\omega) = \boldsymbol{W}_h(\omega)\hat{\boldsymbol{h}}_{n-1}(\omega) + \boldsymbol{W}_u(\omega)\boldsymbol{u}_n(\omega) + \boldsymbol{b}(\omega)$$

$$\hat{\boldsymbol{h}}_n(\omega) = \boldsymbol{g}(\hat{\boldsymbol{x}}_n(\omega))$$

$$\hat{\boldsymbol{y}}_n(\omega) = Y(\hat{\boldsymbol{h}}_n(\omega), \boldsymbol{w}_y(\omega)) + \boldsymbol{\varepsilon}_n(\omega)$$

with one of the following methods

- sampling (e.g. Monte Carlo, quasi-Monte Carlo, etc.)
- approximation based methods (e.g. kernel methods, Gaussian mixture, etc.)

Bayesian RNN

Given observation

$$\boldsymbol{y}_n = Y(\boldsymbol{h}_n(\boldsymbol{w}), \boldsymbol{w}) + \boldsymbol{\varepsilon}(\hat{\omega}),$$

one may estimate the unknown weights $\boldsymbol{w}$ by using Bayes rule

$$p(\boldsymbol{w}|\boldsymbol{y}_n) \propto p(\boldsymbol{y}_n|\boldsymbol{w})p(\boldsymbol{w})$$

in which $p(\boldsymbol{y}_n|\boldsymbol{w})$ denotes the likelihood, and $p(\boldsymbol{w})$ is the a priori distribution.

Bayesian RNN

Given observation

$$\boldsymbol{y}_n = Y(\boldsymbol{h}_n(\boldsymbol{w}), \boldsymbol{w}) + \boldsymbol{\varepsilon}(\hat{\omega}),$$

one may estimate the unknown weights $\boldsymbol{w}$ by using Bayes rule

$$p(\boldsymbol{w}|\boldsymbol{y}_n) \propto p(\boldsymbol{y}_n|\boldsymbol{w})p(\boldsymbol{w})$$

in which $p(\boldsymbol{y}_n|\boldsymbol{w})$ denotes the likelihood, and $p(\boldsymbol{w})$ is the a priori distribution.

Assuming that all activation functions and observation are linear, and

$$p(\boldsymbol{w}) \sim \mathcal{N}(\boldsymbol{w}_f, \boldsymbol{C}_w), \quad \boldsymbol{w}_f \sim \mathcal{N}(\boldsymbol{w}_f(\omega), \boldsymbol{C}_w), \quad p(\boldsymbol{\varepsilon}_n) \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I}_\varepsilon)$$

the Bayes's rule reduces to the regularized RNN-cost function

$$\boldsymbol{J}_{BR} := \left(\boldsymbol{J}(\boldsymbol{w}) + \frac{1}{2}\langle \boldsymbol{w} - \boldsymbol{w}_f, \boldsymbol{w} - \boldsymbol{w}_f \rangle_{\boldsymbol{C}_w} + \frac{1}{2}\langle \boldsymbol{w}_f - \bar{\boldsymbol{w}}, \boldsymbol{w}_f - \bar{\boldsymbol{w}} \rangle_{\boldsymbol{C}_w} \right)$$

Bayesian RNN

From

$$\boldsymbol{w}^* = \arg\min \boldsymbol{J}_{BR}(\boldsymbol{w})$$

the maximum aposterori estimate reads

$$\boldsymbol{w}_a(\omega) = \boldsymbol{w}_f(\omega) + \boldsymbol{K}(\boldsymbol{y}_n - \hat{\boldsymbol{y}}_n(\omega))$$

in which "a" denotes a-posteriori random variable, and

$$\boldsymbol{K} = \boldsymbol{C}_{\boldsymbol{w}(\omega), \boldsymbol{y}_n(\omega)} \boldsymbol{C}^{-1}_{\boldsymbol{y}_n(\omega)}$$

is known as the Kalman gain. The previous formula is also known as a classical Kalman filter estimate.

Nonlinearity issue

However, RNN cell is violating linearity assumption:

$$\boldsymbol{x}_n(\omega) = \boldsymbol{W}_h(\omega)\boldsymbol{h}_{n-1}(\omega) + \boldsymbol{W}_u(\omega)\boldsymbol{u}_n(\omega) + \boldsymbol{b}(\omega)$$

$$\boldsymbol{h}_n(\omega) = \boldsymbol{g}(\boldsymbol{x}_n(\omega))$$

$$\boldsymbol{y}_n(\omega) = Y(\boldsymbol{h}_n(\omega), \boldsymbol{w}_y(\omega)) + \boldsymbol{\varepsilon}_n(\omega)$$

and thus one cannot use the previously described Kalman filter.

Nonlinearity issue

However, RNN cell is violating linearity assumption:

$$\boldsymbol{x}_n(\omega) = \boldsymbol{W}_h(\omega)\boldsymbol{h}_{n-1}(\omega) + \boldsymbol{W}_u(\omega)\boldsymbol{u}_n(\omega) + \boldsymbol{b}(\omega)$$

$$\boldsymbol{h}_n(\omega) = \boldsymbol{g}(\boldsymbol{x}_n(\omega))$$

$$\boldsymbol{y}_n(\omega) = Y(\boldsymbol{h}_n(\omega), \boldsymbol{w}_y(\omega)) + \boldsymbol{\varepsilon}_n(\omega)$$

and thus one cannot use the previously described Kalman filter.

On the other hand, estimating the full posterior using Bayes's rule:

$$p(\boldsymbol{w}|\boldsymbol{y}_n) \propto p(\boldsymbol{y}_n|\boldsymbol{w})p(\boldsymbol{w})$$

would be computationally expensive.

Gauss-Markov-Kalman RNN

Inverse problem

Instead of estimating $p(\boldsymbol{w}|\boldsymbol{y}_n)$, *estimate the conditional expectation* $\mathbb{E}(\boldsymbol{w}|\boldsymbol{y}_n) = \int \boldsymbol{w} p(\boldsymbol{w}|\boldsymbol{y}_n) d\boldsymbol{w}$ *directly without integration.*

Gauss-Markov-Kalman RNN

Inverse problem

Instead of estimating $p(\boldsymbol{w}|\boldsymbol{y}_n)$*, estimate the conditional expectation* $\mathbb{E}(\boldsymbol{w}|\boldsymbol{y}_n) = \int \boldsymbol{w} p(\boldsymbol{w}|\boldsymbol{y}_n) d\boldsymbol{w}$ *directly without integration.*

$$\mathbb{E}(\boldsymbol{w}|\mathfrak{B}) = P_{\mathfrak{B}}(\boldsymbol{w}) = \underset{\eta \in \mathscr{Q}_{\mathfrak{B}}}{\arg\min} \, \|\boldsymbol{w} - \boldsymbol{\eta}\|^2_{\mathscr{Q}}, \quad \mathfrak{B} := \sigma(\boldsymbol{y}_n)$$

Optimality and orthogonality conditions:

$$\forall \tilde{\boldsymbol{w}} \in \mathscr{Q}_{\mathfrak{B}} : \langle\langle \boldsymbol{w} - \mathbb{E}(\boldsymbol{w}|\sigma(\boldsymbol{y})), \tilde{\boldsymbol{w}} \rangle\rangle = 0 \Rightarrow \boldsymbol{w} - \mathbb{E}(\boldsymbol{w}|\sigma(\boldsymbol{y})) \in \mathscr{Q}^{\perp}_{\mathfrak{B}}$$

Gauss-Markov-Kalman RNN

Inverse problem

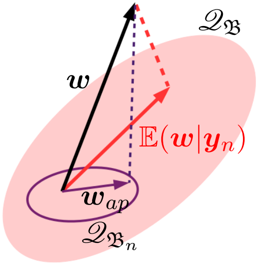
Instead of estimating $p(\boldsymbol{w}|\boldsymbol{y}_n)$, estimate the conditional expectation $\mathbb{E}(\boldsymbol{w}|\boldsymbol{y}_n) = \int \boldsymbol{w}p(\boldsymbol{w}|\boldsymbol{y}_n)d\boldsymbol{w}$ directly without integration.

$$\mathbb{E}(\boldsymbol{w}|\mathfrak{B}) = P_{\mathfrak{B}}(\boldsymbol{w}) = \underset{\eta \in \mathscr{Q}_{\mathfrak{B}}}{\arg\min} \; \|\boldsymbol{w} - \boldsymbol{\eta}\|^2_{\mathscr{Q}}, \quad \mathfrak{B} := \sigma(\boldsymbol{y}_n)$$

Optimality and orthogonality conditions:

$$\forall \tilde{\boldsymbol{w}} \in \mathscr{Q}_{\mathfrak{B}} : \langle\!\langle \boldsymbol{w} - \mathbb{E}(\boldsymbol{w}|\sigma(\boldsymbol{y})), \tilde{\boldsymbol{w}} \rangle\!\rangle = 0 \Rightarrow \boldsymbol{w} - \mathbb{E}(\boldsymbol{w}|\sigma(\boldsymbol{y})) \in \mathscr{Q}_{\mathfrak{B}}^{\perp}$$

$$\boldsymbol{w} = P_{\mathfrak{B}}\boldsymbol{w} + (I - P_{\mathfrak{B}})\boldsymbol{w}$$



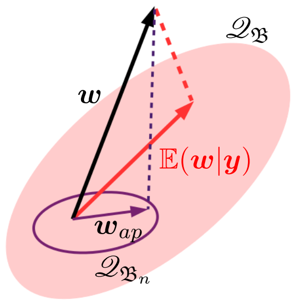
Gauss-Markov-Kalman RNN

Use Doob-Dynkin lemma

$$\mathbb{E}(\boldsymbol{w}|\mathfrak{B}) = P_{\mathfrak{B}}(\boldsymbol{w}) = \boldsymbol{\varphi}(\boldsymbol{y}_n(\boldsymbol{w}))$$

with $\boldsymbol{\varphi} \in L_0(\mathcal{Y}; \mathcal{Q})$ such that

$$\begin{aligned} \boldsymbol{w} &= P_{\mathfrak{B}}\boldsymbol{w} + (I - P_{\mathfrak{B}})\boldsymbol{w} \\ &= \boldsymbol{\varphi}(\boldsymbol{y}_n) + (\boldsymbol{w} - \boldsymbol{\varphi}(\boldsymbol{y}_n)). \end{aligned}$$



Then one has

$$\boldsymbol{w} = \underbrace{\boldsymbol{\varphi}(\boldsymbol{y})}_{data} + \underbrace{(\boldsymbol{w} - \boldsymbol{\varphi}(\boldsymbol{y}))}_{prior}$$

leading to [Rosic et al, 2012]

$$\boldsymbol{w}_a(\omega) = \boldsymbol{w}_f(\omega) + \boldsymbol{\varphi}(\boldsymbol{y}_n) - \boldsymbol{\varphi}(\boldsymbol{y}_n(\omega))$$

Updating more than mean

Inverse problem

Given noisy data estimate the conditional expectation $\mathbb{E}(R(\boldsymbol{w})|\boldsymbol{y}_n)$ *of* $\mathcal{R}$*-valued functions of* $\boldsymbol{w}$*, priorly seen as vectorial RVs* $R(\boldsymbol{w})$ *- in the Hilbert space* $\mathscr{R} := L_2(\Omega, \mathfrak{F}, \mathbb{P}; \mathcal{R})$*, directly without integration.*

- Conditional mean:

$$R(\boldsymbol{w}) = \boldsymbol{w}$$

- Conditional covariance:

$$R(\boldsymbol{w}) = (\boldsymbol{w} - \bar{\boldsymbol{w}}) \otimes (\boldsymbol{w} - \bar{\boldsymbol{w}}), \quad \bar{\boldsymbol{w}} = \mathbb{E}(\boldsymbol{w})$$

Updating more than mean

Hence,

$$\mathbb{E}(R(\boldsymbol{w})|\mathfrak{B}) = P_{\mathfrak{B}}(R(\boldsymbol{w})) = \arg\min_{\eta \in \mathscr{R}_{\mathfrak{B}}} \|R(\boldsymbol{w}) - \eta\|^2_{\mathscr{R}}$$

in which closed subspace

$$\mathscr{R}_{\mathfrak{B}} = L_2(\Omega, \sigma(\boldsymbol{y}_n), \mathbb{P}; \mathcal{R}), \quad \mathfrak{B} := \sigma(\boldsymbol{y}_n).$$

Optimality condition:

$$\forall \eta \in \mathscr{R}_{\mathfrak{B}} : \langle\!\langle R(\boldsymbol{w}) - \mathbb{E}(R(\boldsymbol{w})|\mathfrak{B}), \eta \rangle\!\rangle = 0 \Rightarrow R(\boldsymbol{w}) - \mathbb{E}(R(\boldsymbol{w})|\mathfrak{B}) \in \mathscr{R}_{\mathfrak{B}}^{\perp}$$

Updating more than mean

Hence,

$$\mathbb{E}(R(\boldsymbol{w})|\mathfrak{B}) = P_{\mathfrak{B}}(R(\boldsymbol{w})) = \underset{\eta \in \mathscr{R}_{\mathfrak{B}}}{\arg\min} \|R(\boldsymbol{w}) - \eta\|^2_{\mathscr{R}}$$

in which closed subspace

$$\mathscr{R}_{\mathfrak{B}} = L_2(\Omega, \sigma(\boldsymbol{y}_n), \mathbb{P}; \mathcal{R}), \quad \mathfrak{B} := \sigma(\boldsymbol{y}_n).$$

Optimality condition:

$$\forall \eta \in \mathscr{R}_{\mathfrak{B}} : \langle\langle R(\boldsymbol{w}) - \mathbb{E}(R(\boldsymbol{w})|\mathfrak{B}), \eta \rangle\rangle = 0 \Rightarrow R(\boldsymbol{w}) - \mathbb{E}(R(\boldsymbol{w})|\mathfrak{B}) \in \mathscr{R}_{\mathfrak{B}}^{\perp}$$

Using Doob-Dynkin lemma

$$\mathbb{E}(R(\boldsymbol{w})|\mathfrak{B}) = P_{\mathfrak{B}}(R(\boldsymbol{w})) = \Phi_{R(\boldsymbol{w})}(\boldsymbol{y}_n)$$

leads to [Matthies et al., 2016]

$$R(\boldsymbol{w}_a) = R(\boldsymbol{w}_f) + \Phi_{R(\boldsymbol{w})}(\boldsymbol{y}_n) - \Phi_{R(\boldsymbol{w})}(\boldsymbol{y}_n(\boldsymbol{w}_f))$$

Optimal map for covariance

- Exact posterior mean

$$R(\boldsymbol{w}) := \boldsymbol{w}, \quad \mathbb{E}(\boldsymbol{w}|\boldsymbol{y}_n) = \Phi_{\boldsymbol{w}}(\boldsymbol{y}_n)$$

- Exact posterior correlation

$$R(\boldsymbol{w}) = \boldsymbol{w} \otimes \boldsymbol{w}, \quad C_c := \mathbb{E}(\boldsymbol{w} \otimes \boldsymbol{w}|\boldsymbol{y}_n) = \Phi_{\boldsymbol{w}\otimes\boldsymbol{w}}(\boldsymbol{y}_n)$$

- Exact posterior covariance

$$C_p = C_c - \Phi_{\boldsymbol{w}}(\boldsymbol{y}_n) \otimes \Phi_{\boldsymbol{w}}(\boldsymbol{y}_n)$$

Optimal map for covariance

In Gauss-Markov-Kalman filter

$$\boldsymbol{w}_a = \boldsymbol{w}_f + \varphi(\boldsymbol{y}_n) - \varphi(\boldsymbol{y}_{n,f}), \quad \tilde{\boldsymbol{w}}_a = \boldsymbol{w}_f - \varphi(\boldsymbol{y}_{n,f})$$

Optimal map for covariance

In Gauss-Markov-Kalman filter

$$\boldsymbol{w}_a = \boldsymbol{w}_f + \varphi(\boldsymbol{y}_n) - \varphi(\boldsymbol{y}_{n,f}), \quad \tilde{\boldsymbol{w}}_a = \boldsymbol{w}_f - \varphi(\boldsymbol{y}_{n,f})$$

one has

$$C_{\boldsymbol{w}_a} = \mathbb{E}(\tilde{\boldsymbol{w}}_a \otimes \tilde{\boldsymbol{w}}_a | \boldsymbol{y}_n) = \mathbb{E}((\boldsymbol{w}_f - \varphi(\boldsymbol{y}_{n,f})) \otimes (\boldsymbol{w}_f - \varphi(\boldsymbol{y}_{n,f})) | \boldsymbol{y}_n)$$

Optimal map for covariance

In Gauss-Markov-Kalman filter

$$\boldsymbol{w}_a = \boldsymbol{w}_f + \varphi(\boldsymbol{y}_n) - \varphi(\boldsymbol{y}_{n,f}), \quad \tilde{\boldsymbol{w}}_a = \boldsymbol{w}_f - \varphi(\boldsymbol{y}_{n,f})$$

one has

$$C_{\boldsymbol{w}_a} = \mathbb{E}(\tilde{\boldsymbol{w}}_a \otimes \tilde{\boldsymbol{w}}_a | \boldsymbol{y}_n) = \mathbb{E}((\boldsymbol{w}_f - \varphi(\boldsymbol{y}_{n,f})) \otimes (\boldsymbol{w}_f - \varphi(\boldsymbol{y}_{n,f})) | \boldsymbol{y}_n)$$

which is not same as

$$C_p = C_c - \varphi(\boldsymbol{y}_n) \otimes \varphi(\boldsymbol{y}_n), \quad C_c := \mathbb{E}(\boldsymbol{w} \otimes \boldsymbol{w} | \boldsymbol{y}_n) = \Phi_{\boldsymbol{w}\otimes\boldsymbol{w}}(\boldsymbol{y}_n)$$

Optimal map for covariance

In Gauss-Markov-Kalman filter

$$\boldsymbol{w}_a = \boldsymbol{w}_f + \varphi(\boldsymbol{y}_n) - \varphi(\boldsymbol{y}_{n,f}), \quad \tilde{\boldsymbol{w}}_a = \boldsymbol{w}_f - \varphi(\boldsymbol{y}_{n,f})$$

one has

$$C_{\boldsymbol{w}_a} = \mathbb{E}(\tilde{\boldsymbol{w}}_a \otimes \tilde{\boldsymbol{w}}_a | \boldsymbol{y}_n) = \mathbb{E}((\boldsymbol{w}_f - \varphi(\boldsymbol{y}_{n,f})) \otimes (\boldsymbol{w}_f - \varphi(\boldsymbol{y}_{n,f}))|\boldsymbol{y}_n)$$

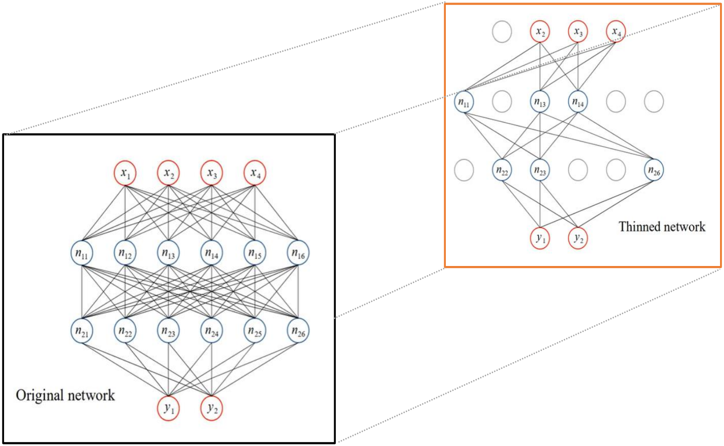
which is not same as

$$C_p = C_c - \varphi(\boldsymbol{y}_n) \otimes \varphi(\boldsymbol{y}_n), \quad C_c := \mathbb{E}(\boldsymbol{w} \otimes \boldsymbol{w} | \boldsymbol{y}_n) = \Phi_{\boldsymbol{w}\otimes\boldsymbol{w}}(\boldsymbol{y}_n)$$

Therefore, the first equation has to be corrected to

$$\boldsymbol{w}_a = \varphi(\boldsymbol{y}_n) + C_p^{1/2} C_{\boldsymbol{w}_a}^{-1/2} \tilde{\boldsymbol{w}}_a.$$

Still, no sparsity only noise



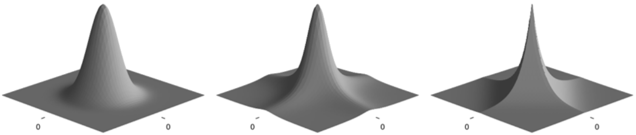
Sparsity inducing prior

In order to introduce sparsity in weights (and thus connections), we may introduce the Laplace prior [Tipping, 2001]:

$$\boldsymbol{w} \sim e^{-\|\boldsymbol{w}\|_1} \Rightarrow p(\boldsymbol{w}|\boldsymbol{\varpi}) \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\varpi}^{-1})$$

in which $\boldsymbol{\varpi}$ is the diagonal matrix with entries ϖ_{ii} (defining precision) corresponding to the Gamma prior $p(\varpi_{ii})$. By marginalizing one obtains

$$p(\boldsymbol{w}) = \int p(\boldsymbol{w}|\boldsymbol{\varpi})p(\boldsymbol{\varpi})d\boldsymbol{\varpi}$$



(a) Multivariate Gaussian. (b) Multivariate Student-t. (c) Multivariate Laplace.

Relevance vector machine

Furthermore, in $\boldsymbol{y}_n(\omega) = Y(\boldsymbol{h}_n(\boldsymbol{w}(\omega)), \boldsymbol{w}(\omega)) + \boldsymbol{\varepsilon}(\omega)$ one assumes that

$$p(\boldsymbol{\varepsilon}) \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\beta}^{-1})$$

with $\boldsymbol{\beta}$ also having Gamma prior, i.e. we assume $\boldsymbol{\beta}$ to be unknown.

Relevance vector machine

Furthermore, in $\boldsymbol{y}_n(\omega) = Y(\boldsymbol{h}_n(\boldsymbol{w}(\omega)), \boldsymbol{w}(\omega)) + \boldsymbol{\varepsilon}(\omega)$ one assumes that

$$p(\boldsymbol{\varepsilon}) \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\beta}^{-1})$$

with $\boldsymbol{\beta}$ also having Gamma prior, i.e. we assume $\boldsymbol{\beta}$ to be unknown. Thus, Bayes rule reads

$$p(\boldsymbol{w}, \boldsymbol{\varpi}, \boldsymbol{\beta}|\boldsymbol{y}_n) \propto p(\boldsymbol{y}_n|\boldsymbol{w}, \boldsymbol{\varpi}, \boldsymbol{\beta})p(\boldsymbol{w}, \boldsymbol{\varpi}, \boldsymbol{\beta})$$

Relevance vector machine

Furthermore, in $\boldsymbol{y}_n(\omega) = Y(\boldsymbol{h}_n(\boldsymbol{w}(\omega)), \boldsymbol{w}(\omega)) + \boldsymbol{\varepsilon}(\omega)$ one assumes that

$$p(\boldsymbol{\varepsilon}) \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\beta}^{-1})$$

with $\boldsymbol{\beta}$ also having Gamma prior, i.e. we assume $\boldsymbol{\beta}$ to be unknown. Thus, Bayes rule reads

$$p(\boldsymbol{w}, \boldsymbol{\varpi}, \boldsymbol{\beta} | \boldsymbol{y}_n) \propto p(\boldsymbol{y}_n | \boldsymbol{w}, \boldsymbol{\varpi}, \boldsymbol{\beta}) p(\boldsymbol{w}, \boldsymbol{\varpi}, \boldsymbol{\beta})$$

The posterior is further decoupled to [Tipping, 2001]

$$p(\boldsymbol{w}, \boldsymbol{\varpi}, \boldsymbol{\beta} | \boldsymbol{y}_n) = \underbrace{p(\boldsymbol{w} | \boldsymbol{y}_n, \boldsymbol{\varpi}, \boldsymbol{\beta})}_{\text{convolution of normals}} \underbrace{p(\boldsymbol{\varpi}, \boldsymbol{\beta} | \boldsymbol{y}_n)}_{\delta(\boldsymbol{\varpi}_{MP}, \boldsymbol{\beta}_{MP})}$$

in which is again assumed that all activation functions and observation operator are linear.

Relevance vector machine

In $p(\boldsymbol{w}, \boldsymbol{\varpi}, \boldsymbol{\beta}|\boldsymbol{y}_n) = \underbrace{p(\boldsymbol{w}|\boldsymbol{y}_n, \boldsymbol{\varpi}, \boldsymbol{\beta})}_{\text{convolution of normals}} \underbrace{p(\boldsymbol{\varpi}, \boldsymbol{\beta}|\boldsymbol{y}_n)}_{\delta(\boldsymbol{\varpi}_{MP}, \boldsymbol{\beta}_{MP})}$ the term

$$\underbrace{p(\boldsymbol{w}|\boldsymbol{y}_n, \boldsymbol{\varpi}, \boldsymbol{\beta})}_{\text{convolution of normals}} = \mathcal{N}(\boldsymbol{w}|\boldsymbol{\mu}_w, \boldsymbol{\Sigma}_w)$$

can be estimated using the classical Kalman filter approach.

Relevance vector machine

In $p(\boldsymbol{w}, \boldsymbol{\varpi}, \boldsymbol{\beta}|\boldsymbol{y}_n) = \underbrace{p(\boldsymbol{w}|\boldsymbol{y}_n, \boldsymbol{\varpi}, \boldsymbol{\beta})}_{\text{convolution of normals}} \underbrace{p(\boldsymbol{\varpi}, \boldsymbol{\beta}|\boldsymbol{y}_n)}_{\delta(\boldsymbol{\varpi}_{MP}, \boldsymbol{\beta}_{MP})}$ the term

$$\underbrace{p(\boldsymbol{w}|\boldsymbol{y}_n, \boldsymbol{\varpi}, \boldsymbol{\beta})}_{\text{convolution of normals}} = \mathcal{N}(\boldsymbol{w}|\boldsymbol{\mu}_w, \boldsymbol{\Sigma}_w)$$

can be estimated using the classical Kalman filter approach.

On the other hand, the maximum point $\delta(\boldsymbol{\varpi}_{MP}, \boldsymbol{\beta}_{MP})$ is obtained given

$$p(\boldsymbol{\varpi}, \boldsymbol{\beta}|\boldsymbol{y}_n) \propto p(\boldsymbol{y}_n|\boldsymbol{\varpi}, \boldsymbol{\beta})p(\boldsymbol{\varpi})p(\boldsymbol{\beta})$$

by maximizing marginal likelihood

$$p(\boldsymbol{y}_n|\boldsymbol{\varpi}, \boldsymbol{\beta}) = \int p(\boldsymbol{y}_n|\boldsymbol{w}, \boldsymbol{\beta})p(\boldsymbol{w}|\boldsymbol{\varpi})d\boldsymbol{w}.$$

Nonlinearity

However, RNN cell is violating linearity assumption:

$$\boldsymbol{x}_n(\omega) = \boldsymbol{W}_h(\omega)\boldsymbol{h}_{n-1}(\omega) + \boldsymbol{W}_u(\omega)\boldsymbol{u}_n(\omega) + \boldsymbol{b}(\omega)$$

$$\boldsymbol{h}_n(\omega) = \boldsymbol{g}(\boldsymbol{x}_n(\omega))$$

$$\boldsymbol{y}_n(\omega) = Y(\boldsymbol{h}_n(\omega), \boldsymbol{w}_y(\omega)) + \boldsymbol{\varepsilon}_n(\omega)$$

and thus in

$$p(\boldsymbol{w}, \boldsymbol{\varpi}, \boldsymbol{\beta}|\boldsymbol{y}_n) = \underbrace{p(\boldsymbol{w}|\boldsymbol{y}_n, \boldsymbol{\varpi}, \boldsymbol{\beta})}_{\neq \text{convolution of normals}} \underbrace{p(\boldsymbol{\varpi}, \boldsymbol{\beta}|\boldsymbol{y}_n)}_{\delta(\boldsymbol{\varpi}_{MP}, \boldsymbol{\beta}_{MP})}$$

is hard to estimate both of posteriors directly.

Nonlinear Relevance Vector Machine

The term

$$p(\boldsymbol{w}|\boldsymbol{y}_n, \boldsymbol{\varpi}, \boldsymbol{\beta})$$

can be estimated by use of the generalized Gauss-Markov Kalman filter:

$$\boldsymbol{w}_a(\omega) = \boldsymbol{w}_f(\omega) + \boldsymbol{\varphi}(\boldsymbol{y}_n) - \boldsymbol{\varphi}(\boldsymbol{y}_n(\omega))$$

in which $\boldsymbol{w}_f(\omega) \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\varpi}^{-1})$, and similarly its covariance [Rosic, 2022, in preparation]:

$$\boldsymbol{w}_a = \varphi(\boldsymbol{y}_n) + C_p^{1/2} C_{\boldsymbol{w}_a}^{-1/2} \tilde{\boldsymbol{w}}_a.$$

Nonlinear Relevance Vector Machine

The term

$$p(\boldsymbol{w}|\boldsymbol{y}_n, \boldsymbol{\varpi}, \boldsymbol{\beta})$$

can be estimated by use of the generalized Gauss-Markov Kalman filter:

$$\boldsymbol{w}_a(\omega) = \boldsymbol{w}_f(\omega) + \boldsymbol{\varphi}(\boldsymbol{y}_n) - \boldsymbol{\varphi}(\boldsymbol{y}_n(\omega))$$

in which $\boldsymbol{w}_f(\omega) \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\varpi}^{-1})$, and similarly its covariance [Rosic, 2022, in preparation]:

$$\boldsymbol{w}_a = \varphi(\boldsymbol{y}_n) + C_p^{1/2} C_{\boldsymbol{w}_a}^{-1/2} \tilde{\boldsymbol{w}}_a.$$

On the other hand, the term

$$\underbrace{p(\boldsymbol{\varpi}, \boldsymbol{\beta}|\boldsymbol{y}_n)}_{\delta(\boldsymbol{\varpi}_{MP}, \boldsymbol{\beta}_{MP})}$$

is hard to estimate directly unless approximating the likelihood.

Approximation

In

$$\boldsymbol{x}_n(\omega) = \boldsymbol{W}_h(\omega)\boldsymbol{h}_{n-1}(\omega) + \boldsymbol{W}_u(\omega)\boldsymbol{u}_n(\omega) + \boldsymbol{b}(\omega)$$

$$\boldsymbol{h}_n(\omega) = \boldsymbol{g}(\boldsymbol{x}_n(\omega))$$

$$\boldsymbol{y}_n(\omega) = Y(\boldsymbol{h}_n(\omega), \boldsymbol{w}_y(\omega)) + \boldsymbol{\varepsilon}_n(\omega)$$

Approximation

In

$$\boldsymbol{x}_n(\omega) = \boldsymbol{W}_h(\omega)\boldsymbol{h}_{n-1}(\omega) + \boldsymbol{W}_u(\omega)\boldsymbol{u}_n(\omega) + \boldsymbol{b}(\omega)$$

$$\boldsymbol{h}_n(\omega) = \boldsymbol{g}(\boldsymbol{x}_n(\omega))$$

$$\boldsymbol{y}_n(\omega) = Y(\boldsymbol{h}_n(\omega), \boldsymbol{w}_y(\omega)) + \boldsymbol{\varepsilon}_n(\omega)$$

one can linearize the last two equations such that

$$\boldsymbol{h}_n^{(\ell)}(\omega) = \boldsymbol{g}^{(\ell)}(\boldsymbol{x}_n(\omega)) = \boldsymbol{J}_x\boldsymbol{x}_n(\omega) + \boldsymbol{z}_h$$

$$\boldsymbol{y}_n^{(\ell)}(\omega) = Y^{(\ell)}(\boldsymbol{h}_n(\omega), \boldsymbol{w}_y(\omega)) + \varepsilon_n(\omega) = \boldsymbol{J}_h\boldsymbol{h}_n^{(\ell)}(\omega) + \boldsymbol{z}_y + \boldsymbol{\varepsilon}_n(\omega)$$

holds. The linearisation can be also achieved by prevously described relevance vector machine [Rosic, 2022, in preparation].

Gaussian approximation of the marginal likelihood

The point $\delta(\boldsymbol{\varpi}_{MP}, \boldsymbol{\beta}_{MP})$ is obtained given

$$p(\boldsymbol{\varpi}, \boldsymbol{\beta}|\boldsymbol{y}_n) \propto p(\boldsymbol{y}_n|\boldsymbol{\varpi}, \boldsymbol{\beta})p(\boldsymbol{\varpi})p(\boldsymbol{\beta})$$

by maximizing $p(\boldsymbol{y}_n|\boldsymbol{\varpi}, \boldsymbol{\beta}) = \int p(\boldsymbol{y}_n|\boldsymbol{w}, \boldsymbol{\beta})p(\boldsymbol{w}|\boldsymbol{\varpi})d\boldsymbol{w}$ in an iterative fashion [Rosic, 2022, in preparation].

Gaussian approximation of the marginal likelihood

The point $\delta(\boldsymbol{\varpi}_{MP}, \boldsymbol{\beta}_{MP})$ is obtained given

$$p(\boldsymbol{\varpi}, \boldsymbol{\beta}|\boldsymbol{y}_n) \propto p(\boldsymbol{y}_n|\boldsymbol{\varpi}, \boldsymbol{\beta})p(\boldsymbol{\varpi})p(\boldsymbol{\beta})$$

by maximizing $p(\boldsymbol{y}_n|\boldsymbol{\varpi}, \boldsymbol{\beta}) = \displaystyle\int p(\boldsymbol{y}_n|\boldsymbol{w}, \boldsymbol{\beta})p(\boldsymbol{w}|\boldsymbol{\varpi})d\boldsymbol{w}$ in an iterative fashion [Rosic, 2022, in preparation]. After linearisation

$$\mathbb{E}(\boldsymbol{y}_n|\boldsymbol{w}, \boldsymbol{\beta}) = \boldsymbol{\Phi}^T\boldsymbol{w}, \quad \boldsymbol{C}(\boldsymbol{y}_n|\boldsymbol{w}, \boldsymbol{\beta}) = \boldsymbol{C}_{\boldsymbol{w}}$$

$$p(\boldsymbol{y}_n|\boldsymbol{w}, \boldsymbol{\beta}) \approx \mathcal{N}(\mu_{\boldsymbol{w}}, \boldsymbol{C}_{\boldsymbol{w}}),$$

the mean vector and the covariance matrix are both the functions of the weights $\boldsymbol{w}$.

Gaussian approximation of the marginal likelihood

The point $\delta(\boldsymbol{\varpi}_{MP}, \boldsymbol{\beta}_{MP})$ is obtained given

$$p(\boldsymbol{\varpi}, \boldsymbol{\beta}|\boldsymbol{y}_n) \propto p(\boldsymbol{y}_n|\boldsymbol{\varpi}, \boldsymbol{\beta})p(\boldsymbol{\varpi})p(\boldsymbol{\beta})$$

by maximizing $p(\boldsymbol{y}_n|\boldsymbol{\varpi}, \boldsymbol{\beta}) = \int p(\boldsymbol{y}_n|\boldsymbol{w}, \boldsymbol{\beta})p(\boldsymbol{w}|\boldsymbol{\varpi})d\boldsymbol{w}$ in an iterative fashion [Rosic, 2022, in preparation]. After linearisation

$$\mathbb{E}(\boldsymbol{y}_n|\boldsymbol{w}, \boldsymbol{\beta}) = \boldsymbol{\Phi}^T\boldsymbol{w}, \quad \boldsymbol{C}(\boldsymbol{y}_n|\boldsymbol{w}, \boldsymbol{\beta}) = \boldsymbol{C}_{\boldsymbol{w}}$$

$$p(\boldsymbol{y}_n|\boldsymbol{w}, \boldsymbol{\beta}) \approx \mathcal{N}(\mu_{\boldsymbol{w}}, \boldsymbol{C}_{\boldsymbol{w}}),$$

the mean vector and the covariance matrix are both the functions of the weights $\boldsymbol{w}$. Thus, one can use the law of the total expectation to get

$$\boldsymbol{\mu} := \mathbb{E}_{p(\boldsymbol{w}|\boldsymbol{\varpi})}(\mathbb{E}(\boldsymbol{y}_n|\boldsymbol{w}, \boldsymbol{\beta})) = \boldsymbol{0}$$

$$\boldsymbol{C} := \mathbb{E}_{p(\boldsymbol{w}|\boldsymbol{\varpi})}(\boldsymbol{C}(\boldsymbol{y}_n|\boldsymbol{w}, \boldsymbol{\beta})) + \boldsymbol{C}_{p(\boldsymbol{w}|\boldsymbol{\varpi})}(\mathbb{E}(\boldsymbol{y}_n|\boldsymbol{w}, \boldsymbol{\beta}))$$

Sparse LSTM

The complete process can be repeated for LSTM model as well [van Weg, Greve, Rosic, 2021]:

$$\boldsymbol{x}_n(\omega) = \boldsymbol{g}_{cx}(n,\omega) \odot \boldsymbol{x}_{n-1}(\omega) + \boldsymbol{g}_{cu}(n,\omega) \odot \boldsymbol{g}(\boldsymbol{s}_n(\omega))$$

$$\boldsymbol{h}_n(\omega) = g(\boldsymbol{x}_n(\omega))$$

in which we choose

$$\boldsymbol{g}_{cx}(n,\omega) = \boldsymbol{g}_a(\hat{\boldsymbol{W}}_x(\omega)\boldsymbol{x}_n(\omega) + \hat{\boldsymbol{W}}_h(\omega)\boldsymbol{h}_{n-1}(\omega) + \boldsymbol{b}_{cx}(\omega))$$

$$\boldsymbol{g}_{cu}(n,\omega) = \boldsymbol{g}_a(\tilde{\boldsymbol{W}}_x(\omega)\boldsymbol{x}_n(\omega) + \tilde{\boldsymbol{W}}_h(\omega)\boldsymbol{h}_{n-1}(\omega) + \boldsymbol{b}_{cu}(\omega))$$

$$\boldsymbol{g}_{c}(n,\omega) = \boldsymbol{g}_a(\bar{\boldsymbol{W}}_x(\omega)\boldsymbol{x}_n(\omega) + \bar{\boldsymbol{W}}_h(\omega)\boldsymbol{h}_{n-1}(\omega) + \boldsymbol{b}_{c}(\omega))$$

$$\boldsymbol{y}_n(\omega) = Y(\boldsymbol{h}_n(\omega), \boldsymbol{w}_y(\omega))$$

Sparse NN for forward and inverse problems

Inverse problem

Given noisy data $\boldsymbol{z} \in \mathcal{Z}$, *i.e.*

$$\boldsymbol{z} = Z(\boldsymbol{q}) + \boldsymbol{\epsilon}$$

estimate the unknown $\boldsymbol{q} \in \mathcal{Q}$.

- $\mathcal{X} := \{\mathcal{Q}, \mathcal{Z}\}$ are Hilbert spaces with the inner product $\langle \cdot, \cdot \rangle_{\mathcal{X}}$
- $\boldsymbol{q} \in \mathcal{Q}$ is the parameter
- $Z : \mathcal{Q} \mapsto \mathcal{Z}$ is possibly nonlinear observation operator
- $\boldsymbol{z}$ are data
- $\boldsymbol{\epsilon}$ are noise realisations

Sparse NN for forward and inverse problems

By using Gauss-Markov-Kalman filter

$$\boldsymbol{q}_a(\omega) = \boldsymbol{q}_f(\omega) + \boldsymbol{\varphi}(\boldsymbol{z}_m) - \boldsymbol{\varphi}(\boldsymbol{y}_f(\omega))$$

we may distinguish two steps [van Dijk, Hakvoort, Rosic, 2022]:

Sparse NN for forward and inverse problems

By using Gauss-Markov-Kalman filter

$$\boldsymbol{q}_a(\omega) = \boldsymbol{q}_f(\omega) + \boldsymbol{\varphi}(\boldsymbol{z}_m) - \boldsymbol{\varphi}(\boldsymbol{y}_f(\omega))$$

we may distinguish two steps [van Dijk, Hakvoort, Rosic, 2022]:

- Forecast (prediction, uncertainty quantification) step

$$\text{Map } \boldsymbol{\phi} : \boldsymbol{q}_f(\omega) \mapsto \boldsymbol{y}_f(\omega)$$

- Assimilation (update) phase

$$\text{Map } \boldsymbol{\varphi} : \boldsymbol{y}_f(\omega) \mapsto \boldsymbol{q}_f(\omega)$$

Sparse NN for forward and inverse problems

By using Gauss-Markov-Kalman filter

$$\boldsymbol{q}_a(\omega) = \boldsymbol{q}_f(\omega) + \boldsymbol{\varphi}(\boldsymbol{z}_m) - \boldsymbol{\varphi}(\boldsymbol{y}_f(\omega))$$

we may distinguish two steps [van Dijk, Hakvoort, Rosic, 2022]:

- Forecast (prediction, uncertainty quantification) step

$$\text{Map } \boldsymbol{\phi} : \boldsymbol{q}_f(\omega) \mapsto \boldsymbol{y}_f(\omega)$$

- Assimilation (update) phase

$$\text{Map } \boldsymbol{\varphi} : \boldsymbol{y}_f(\omega) \mapsto \boldsymbol{q}_f(\omega)$$

Both of these maps can be approximated by sparse NNs such that:

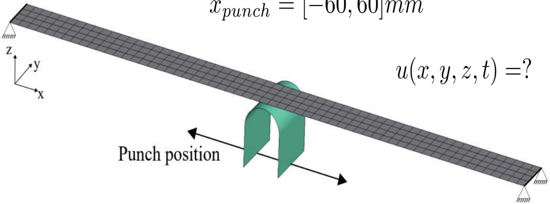
$$\boldsymbol{q}_a(\omega) = \boldsymbol{q}_f(\omega) + \boldsymbol{\varphi}_{sNN}(\boldsymbol{z}_m) - \boldsymbol{\varphi}_{sNN}(\boldsymbol{\phi}_{sNN}(\boldsymbol{q}_f(\omega)) + \boldsymbol{\epsilon}(\omega)$$

Numerical example

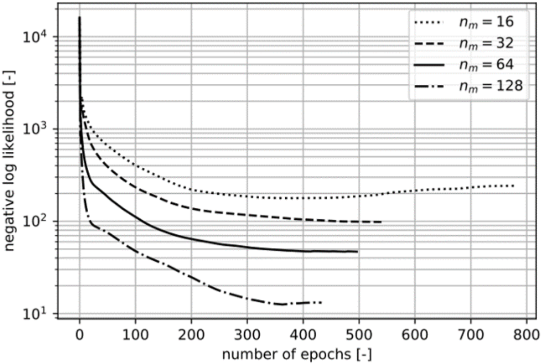
$$m = 10kg \quad v = 2ms^{-1} \quad T = [0, 20]ms, N = 41$$

$$x_{punch} = [-60, 60]mm$$

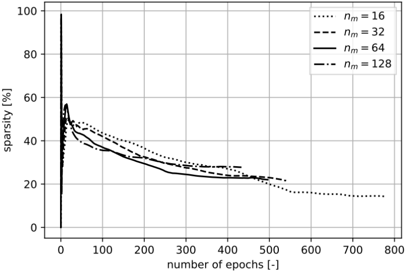
$$u(x, y, z, t) = ?$$

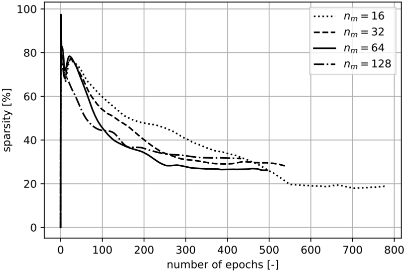


Convergence



Sparsity





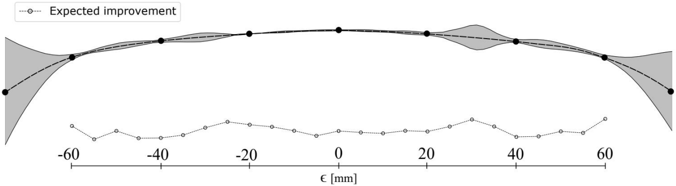
Sparsity of a) Dense layer, b) LSTM cell

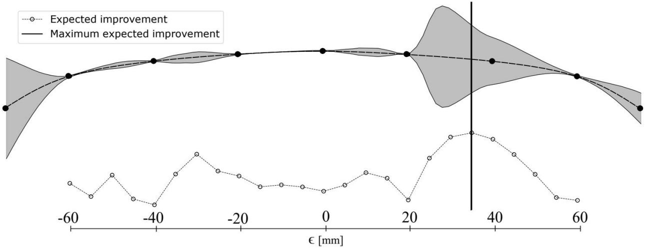
Comparison to point estimate

$$\mathrm{R}^2 = 1 - \frac{\sum_i^m \sum_j^{n_b} \left(\boldsymbol{y}_{ij} - \frac{1}{n_b} \sum_{n_b} (\boldsymbol{y}_i)^* \right)^2}{\sum_i^m \sum_j^{n_b} \left(\boldsymbol{y}_{ij} - \frac{1}{n_b} \sum_{n_b} \boldsymbol{y}_i \right)^2},$$

	n_m	epochs [−]	time [s]	time per epoch [s]	R^2 [−]
Point estimate LSTM	16	4000	189	0.047	0.994
	32	4000	189	0.048	0.993
	64	4000	190	0.048	0.994
	128	4000	191	0.048	0.996
ARD-LSTM	16	779	153	0.20	0.993
	32	541	151	0.28	0.995
	64	497	317	0.64	0.998
	128	434	1403	3.23	0.998

Expected improvement





Identification

$$\frac{\mathrm{d}x}{\mathrm{d}t} = -\sigma x + \sigma y,$$
$$\frac{\mathrm{d}y}{\mathrm{d}t} = \rho x - xz - y,$$
$$\frac{\mathrm{d}z}{\mathrm{d}t} = xy - \beta z.$$

$$\boldsymbol{x}_0 = [1.508870, -1.531271, 25.46091]$$

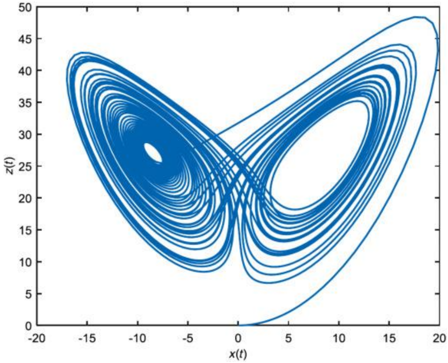
$$\boldsymbol{q} = [\sigma, \rho, \beta] \quad = \quad [10, 28, 8/3]$$

$$\boldsymbol{q}(\omega) \sim \mathcal{U}(\boldsymbol{q}_{min}, \boldsymbol{q}_{max}),$$

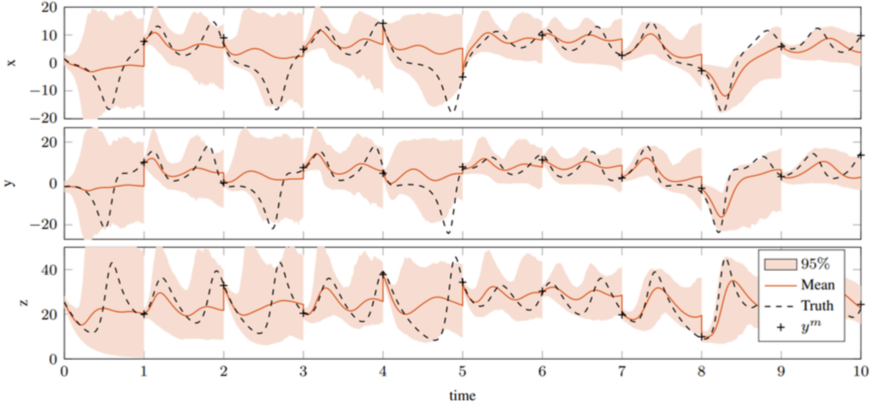
$$\boldsymbol{\mu}_{\boldsymbol{x}_0} = \boldsymbol{x}_0, \qquad \boldsymbol{\sigma}^2_{\boldsymbol{x}_0} = [2, 2, 2],$$

$$\boldsymbol{x}_0(\omega) \sim \mathcal{N}(\boldsymbol{\mu}_{\boldsymbol{x}_0}, \sigma^2_{\boldsymbol{x}_0} I),$$

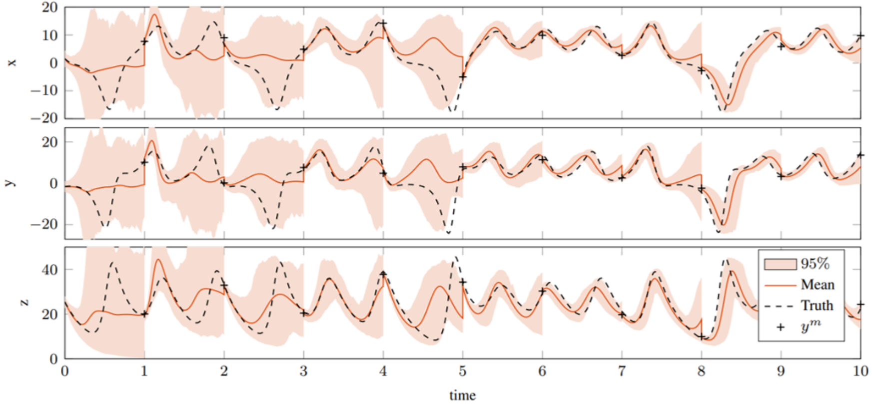
$$\boldsymbol{q}_{min} = [1, 1, 1], \qquad \boldsymbol{q}_{max} = [30, 44.8, 5.3].$$



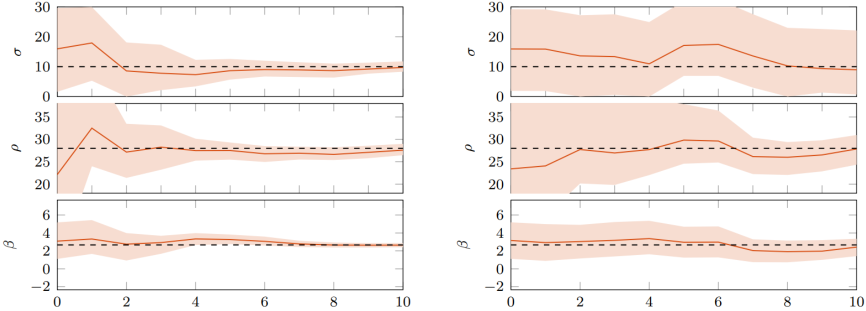
State Identification



State Identification



Parameter Identification



(a) (b)

Conclusion

Currently done:

- Neural networks (NN) can be represented as delayed differential equations
- Classical training is reqiring more data due to higher parametrisation
- Sparse training using relevance vector machine is only for linear case
- We suggest nonlinear releveance vector machine and apply on NN

To be done:

- study the requirements for convergence and stability
- extend this with the model reduction techniques

Thank you: any questions?