Neural ODEs for *real-world*, irregularly-sampled, time series prediction

Emanuele Ballarin
Milton Nicolás Plasencia Palacios
Arianna Tasciotti

Department of Mathematics and Geosciences, University of Trieste

Statistical Machine Learning course
Final Project ∼ June 2020

We are here...

1. *Theory first!*
2. *Deep learning works great, except when it doesn't.*
3. *No data, no learning!*
4. *So, what?*

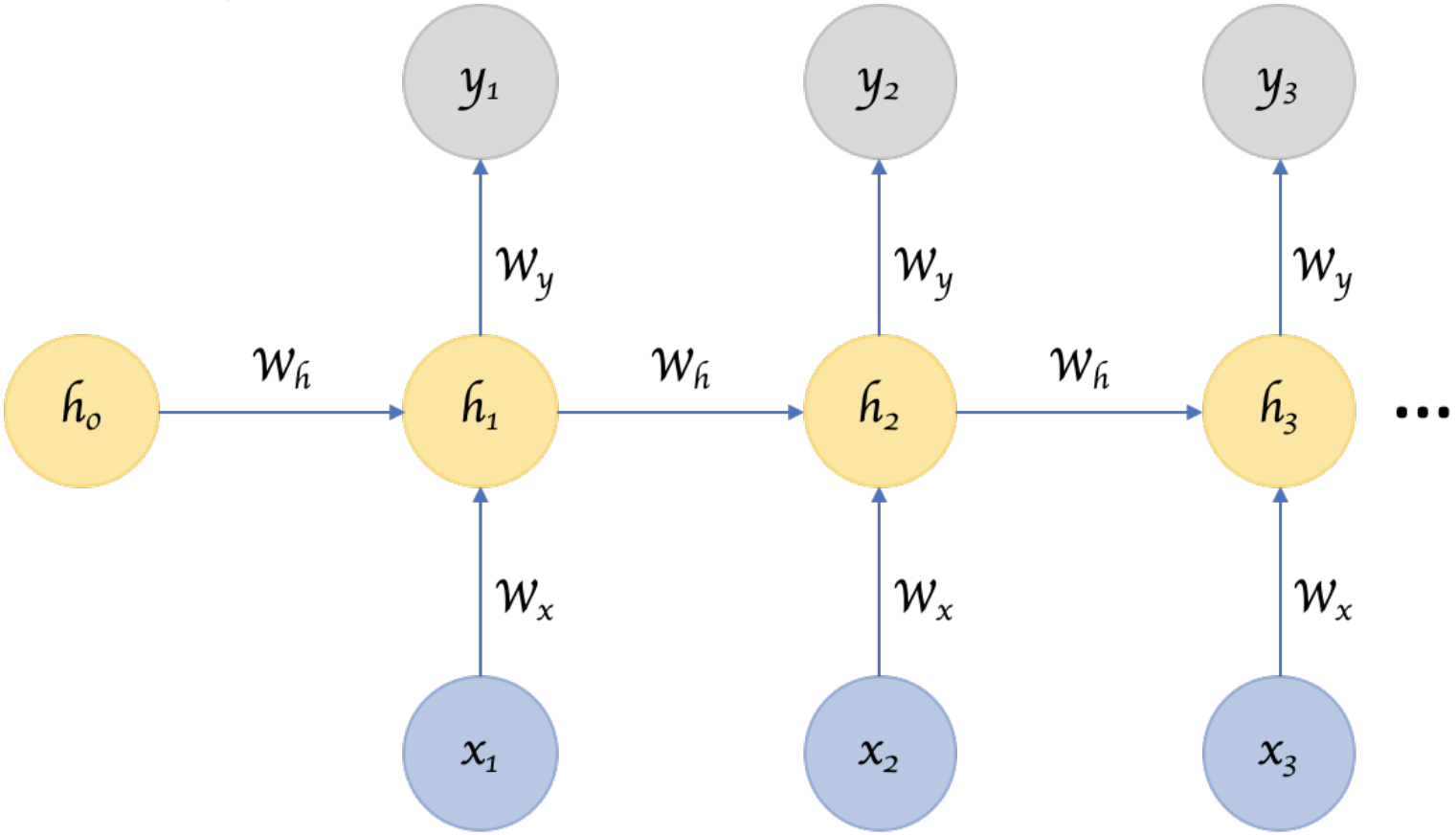
Time Series

A **time series** is a sequence of observations labeled in time order. If the time-dependent variables are more than one we are talking about **multivariate time series**.

This type of data is highly exploited in economics, business and finance where it is important to predict future outcomes that depends on past observations.

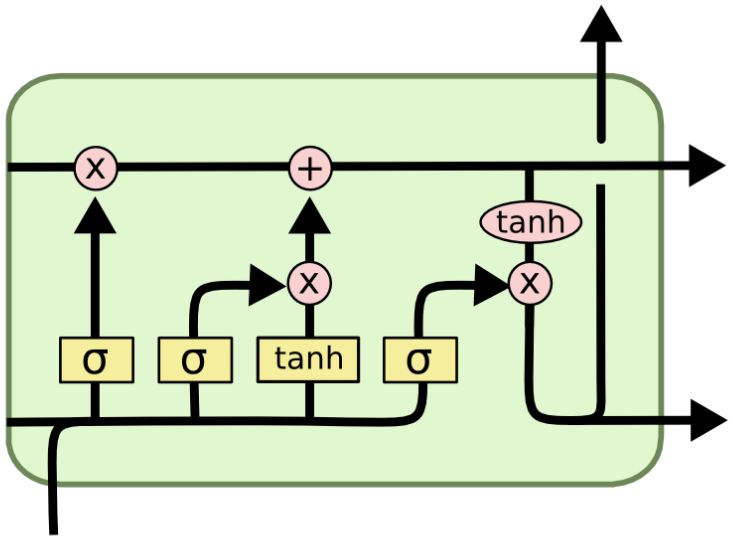
RNN

To study a time series we can use a **Recurrent Neural Network** (RNN), which is a type of neural network in which the current hidden state is computed using the previous hidden state and the current input.



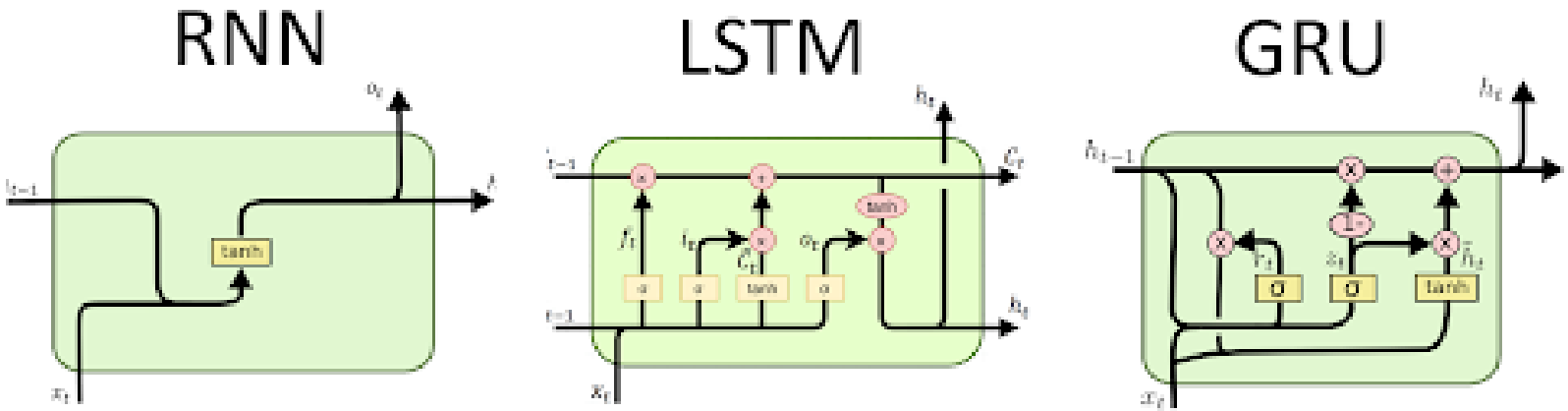
LSTM

The **Long Short Term Memory Unit** is an improvement of the recurrent unit in which the information pass through a **don't forget gate** which decides to keep or throw away data. Input and output data are processed respectively by **input** and **output gate**.



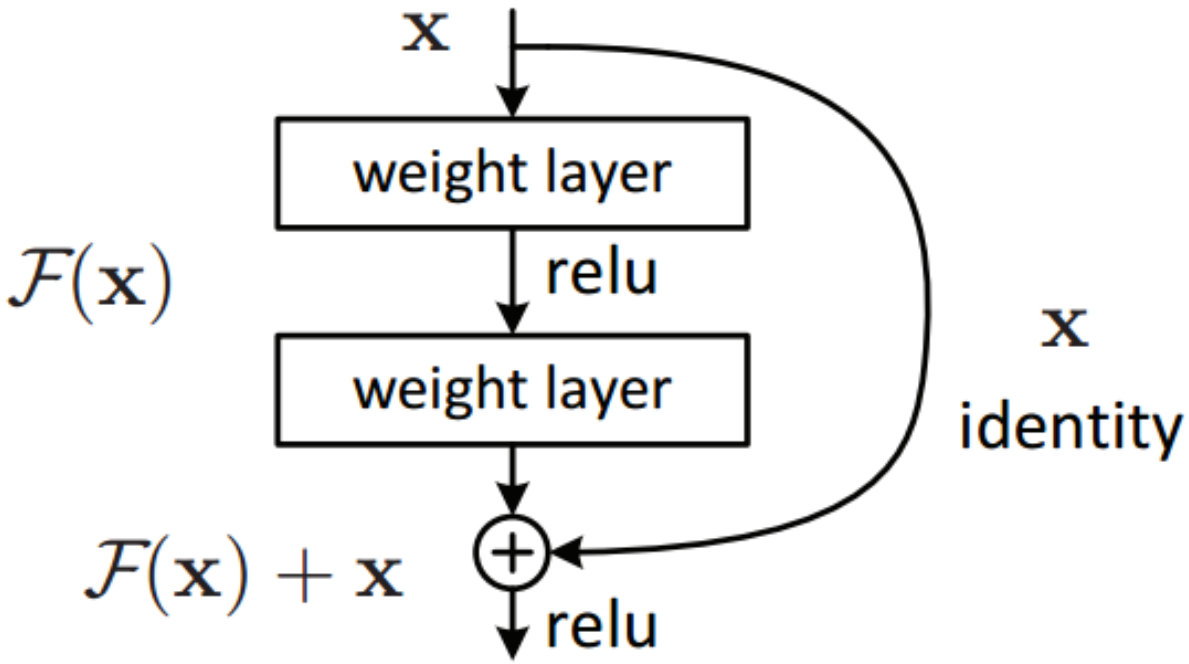
GRU

Gated Recurrent Unit (GRU) is an improved version (in terms of efficiency) of the LSTM. GRU uses **update and reset gates** in order to modulate what information should be passed to the output.



ResNet

Residual Neural Networks (ResNet) use residual blocks and skip connections to solve vanishing gradient problem.



ODE

The ResNet model:

$$\mathbf{h}_{l+1} = \mathbf{h}_l + f(\mathbf{h}_l, \theta_l) \tag{1}$$

can be seen as the Euler method for solving ordinary differential equations. We can exploit this new relation by adding more layers into the network and take smaller time steps. In the limit we can describe our new neural network with the following model (**Neural ODE**):

$$\frac{d\mathbf{h}(t)}{dt} = f(\mathbf{h}(t), t, \theta) \tag{2}$$

with the function f parameterized by a neural network, and solve it with a differentiable discretization method for ODEs.

RNN-ODE

Standard RNNs

'Latent ODEs for Irregularly-Sampled Time Series', Yulia Rubanova and Ricky T. Q. Chen and David Duvenaud, 2019.

RNN-ODE

Standard RNNs

'Latent ODEs for Irregularly-Sampled Time Series', Yulia Rubanova and Ricky T. Q. Chen and David Duvenaud, 2019.

- Constant or undefined hidden states between observations
- Hard to adapt for sparse irregular data...

RNN-ODE

RNNs for sparse data

- Include the time gap $\Delta_t = t_i - t_{i-1}$ between observations into RNN update function:

$$h_i = RNNCell(h_{i-1}, \Delta_t, x_i)$$

RNN-ODE

RNNs for sparse data

- Include the time gap $\Delta_t = t_i - t_{i-1}$ between observations into RNN update function:

$$h_i = RNNCell(h_{i-1}, \Delta_t, x_i)$$

- Exponential decay of the hidden state:

$$h_i = RNNCell(h_{i-1}, \exp(-\tau \Delta_t), x_i)$$

'Latent ODEs for Irregularly-Sampled Time Series', Yulia Rubanova and Ricky T. Q. Chen and David Duvenaud, 2019.

Equivalence between exponential decay approach and solving an initial value problem involving an ODE:

$$\begin{cases} \frac{dh(t)}{dt} & = -\tau h \\ h(t_0) & = h_0 \end{cases}$$

Equivalence between exponential decay approach and solving an initial value problem involving an ODE:

$$\begin{cases} \frac{dh(t)}{dt} & = -\tau h \\ h(t_0) & = h_0 \end{cases}$$

ODE solution: $h(t) = h_0 e^{-\tau \Delta_t}$

Equivalence between exponential decay approach and solving an initial value problem involving an ODE:

$$\begin{cases} \frac{dh(t)}{dt} & = -\tau h \\ h(t_0) & = h_0 \end{cases}$$

ODE solution: $h(t) = h_0 e^{-\tau \Delta_t}$

- hidden state $h(t)$ is an unknown function of time and is the solution of an ODE:

$$\frac{dh(t)}{dt} = f_\theta(h(t), t)$$

Equivalence between exponential decay approach and solving an initial value problem involving an ODE:

$$\begin{cases} \frac{dh(t)}{dt} & = -\tau h \\ h(t_0) & = h_0 \end{cases}$$

ODE solution: $h(t) = h_0 e^{-\tau \Delta_t}$

- hidden state $h(t)$ is an unknown function of time and is the solution of an ODE:

$$\frac{dh(t)}{dt} = f_\theta(h(t), t)$$

- $h(t)$ can be evaluated at any time using a numerical ODE solver.

GRU-ODE architecture

The state between observations is the solution of an ODE:

$$h_i' = ODESolve(f_\theta, h_{i-1}, (t_{i-1}, t_i))$$

GRU-ODE architecture

The state between observations is the solution of an ODE:

$$h_i' = ODESolve(f_\theta, h_{i-1}, (t_{i-1}, t_i))$$

For each observation, the hidden state is updated using a GRU update:

$$h_i = GRUCell(h_i', x_i)$$

'Latent ODEs for Irregularly-Sampled Time Series', Yulia Rubanova and Ricky T. Q. Chen and David Duvenaud, 2019.

Latent ODE model

- Time series is represented by a latent trajectory;

Latent ODE model

- Time series is represented by a latent trajectory;
- Each trajectory is determined from a local initial state z_0 and a global set of latent dynamics shared across time series.

Latent ODE model

- Time series is represented by a latent trajectory;
- Each trajectory is determined from a local initial state z_0 and a global set of latent dynamics shared across time series.

Two tasks:

- extrapolation $\rightarrow$ forwards in time
- interpolation $\rightarrow$ backwards in time

Uncertainty quantifier

x_{in}, x_{out} observed, z latent, $p(x_{in}, x_{out}, z)$ joint

Uncertainty quantifier

x_{in}, x_{out} observed, z latent, $p(x_{in}, x_{out}, z)$ joint

$$q(z_0|x_{in}, x_{out}) = \mathcal{N}(\mu_{z_0}, \sigma_{z_0})$$

Uncertainty quantifier

Feed-forward NN

- Input: final hidden state of GRU-ODE
- Output: μ_{z_0} and σ_{z_0}

Uncertainty quantifier

Feed-forward NN

- Input: final hidden state of GRU-ODE
- Output: μ_{z_0} and σ_{z_0}

Sampler

- Input: μ_{z_0}, σ_{z_0}
- Output: $\bar{z}_0$

Architectural overview

$$h'_i = ODESolve(f, h_{i-1}, (t_{i-1}, t_i))$$

Architectural overview

$$h_i' = ODESolve(f, h_{i-1}, (t_{i-1}, t_i))$$

$$h_i = GRUCell(h_i', x_i)$$

Architectural overview

$$h_i' = ODESolve(f, h_{i-1}, (t_{i-1}, t_i))$$

$$h_i = GRUCell(h_i', x_i)$$

$$\mu_{z_0}, \sigma_{z_0} = g(h_i)$$

Architectural overview

$$h_i' = ODESolve(f, h_{i-1}, (t_{i-1}, t_i))$$

$$h_i = GRUCell(h_i', x_i)$$

$$\mu_{z_0}, \sigma_{z_0} = g(h_i)$$

$$z_0 \sim \mathcal{N}(\mu_{z_0}, \sigma_{z_0})$$

Architectural overview

$$h_i' = ODESolve(f, h_{i-1}, (t_{i-1}, t_i))$$

$$h_i = GRUCell(h_i', x_i)$$

$$\mu_{z_0}, \sigma_{z_0} = g(h_i)$$

$$z_0 \sim \mathcal{N}(\mu_{z_0}, \sigma_{z_0})$$

$$\{z_i\} = ODESolve(z_0, f, (t_0, ..., t_N))$$

Architectural overview

$$h_i' = ODESolve(f, h_{i-1}, (t_{i-1}, t_i))$$

$$h_i = GRUCell(h_i', x_i)$$

$$\mu_{z_0}, \sigma_{z_0} = g(h_i)$$

$$z_0 \sim \mathcal{N}(\mu_{z_0}, \sigma_{z_0})$$

$$\{z_i\} = ODESolve(z_0, f, (t_0, ..., t_N))$$

$$\widetilde{x}_i = OutputNN(z_i)$$

A *bird's eye* architectural recap (1)

A *lato sensu* encoder-decoder structure

$$h_i' = ODESolve(f, h_{i-1}, (t_{i-1}, t_i))$$

$$h_i = GRUCell(h_i', x_i)$$

Encoder. Encodes an entire multidimensional, variable-length time series to fixed-low-dimensional latent representation, loop after loop. Outputs the present *latent state*.

$$\mu_{z_0}, \sigma_{z_0} = g(h_i)$$

$$z_0 \sim \mathcal{N}(\mu_{z_0}, \sigma_{z_0})$$

$$\{z_i\} = ODESolve(z_0, f, (t_0, ..., t_N))$$

$$\widetilde{x_i} = OutputNN(z_i)$$

A *bird's eye* architectural recap (2)

A *lato sensu* encoder-decoder structure

$$h_i' = ODESolve(f, h_{i-1}, (t_{i-1}, t_i))$$

$$h_i = GRUCell(h_i', x_i)$$

$$\mu_{z_0}, \sigma_{z_0} = g(h_i)$$

$$z_0 \sim \mathcal{N}(\mu_{z_0}, \sigma_{z_0})$$

Uncertainty estimator. $\approx$ a VAE. Essentially *BBVI* tricked into making uncertainty estimation (only). Outputs the present *processed* latent state.

$$\{z_i\} = ODESolve(z_0, f, (t_0, ..., t_N))$$

$$\widetilde{x}_i = OutputNN(z_i)$$

A *bird's eye* architectural recap (3)

A *lato sensu* encoder-decoder structure

$$h_i' = ODESolve(f, h_{i-1}, (t_{i-1}, t_i))$$

$$h_i = GRUCell(h_i', x_i)$$

$$\mu_{z_0}, \sigma_{z_0} = g(h_i)$$

$$z_0 \sim \mathcal{N}(\mu_{z_0}, \sigma_{z_0})$$

$$\{z_i\} = ODESolve(z_0, f, (t_0, ..., t_N))$$

$$\widetilde{x}_i = OutputNN(z_i)$$

Decoder. Emits one entire predicted future-time-series, from present *processed* latent state.

We are here...

1. *Theory first!*
2. *Deep learning works great, except when it doesn't.*
3. *No data, no learning!*
4. *So, what?*

The theory seems sound...

So sound it doesn't even work

To make such architecture work in practice, we need to pull some (theoretically-grounded) tricks out of the $\widehat{trick}^{*}$!

[* trick hat]

Batch Normalisation and *Layer Normalisation*

A *batch-wise* and a *feature-wise* approach to training regularisation

Introduced to fight the problem of *internal covariate shift*, they mainly act as *noise-injection regularisers (Leslie, 2019)*.

$$\hat{x}_i^{(j)} = \frac{x_i^{(j)} - \mu^{(j)}}{\sqrt{\sigma^{(j)^2} + \epsilon}}$$

- *BatchNorm (Ioffe, 2015)* acts dimension-wise over a minibatch, during training;
- *LayerNorm (Ba, 2016)* acts dimension-wise and feature-wise, online, also during evaluation.

Gradient Norm Clipping

Keeping seatbelts fastened

The recurrent nature of GRUs (and RNNs in general) may lead to exponentially-varying weights, and subsequently gradients.
$\rightarrow$ Training instability!

- *Vanishing gradients* $\rightarrow$ Gating (Schmidhuber, 1992; Cho, 2015), ReLU networks;
- *Exploding gradients* $\rightarrow$ $||\nabla\boldsymbol{\theta}|| := \min(c, ||\nabla\boldsymbol{\theta}||)$ and rescale *(Graves, 2013)*.

Also: no side effects if unneeded.

AdamW Optimiser (1)

Ridge penalty and *weight decay* are not the same thing

All the *Adam-family* optimisers:

- Are Hessian-free;
- Use single-parameter-adaptive, upper-bound, SGD learning;
- Exploit 2^{nd}-order gradient-momentum information.

However, they regularise differently!

- Adam: $\mathcal{L}_r = \mathcal{L} + \omega \frac{||\boldsymbol{\theta}||_2^2}{2}$ *(Kingma, 2014)*;
- AdamW: $\boldsymbol{\theta} := \boldsymbol{\theta} - \boldsymbol{\rho} \cdot \nabla \boldsymbol{\theta} - \omega \boldsymbol{\rho} \cdot \boldsymbol{\theta}$ *(Loshchilov, 2018)*.

AdamW Optimiser (2)

A deeper dive into Adam & friends

For each scalar weight w, at iteration t, in parallel:

$$v_t = \beta_1 v_{t-1} - (1 - \beta_1)\nabla w$$

$$s_t = \beta_2 s_{t-1} - (1 - \beta_2)(\nabla w)^2$$

$$\rho = \min\left(\text{lr}, \text{lr}\frac{v_t}{\sqrt{s_t + \epsilon}}\right)$$

And then, the usual $\boldsymbol{\theta} := \boldsymbol{\theta} - \boldsymbol{\rho} \cdot \nabla\boldsymbol{\theta}$.

Per-parameter estimation makes the two regularisations different.

L_1 loss

Uniformity is the key

An outright impopular choice for a loss function, however:

- The per-epoch loss average was ≈ 1;
- $\mathcal{L}^2 < \mathcal{L}$ if $\mathcal{L} < 1$, whereas $\mathcal{L}^2 > \mathcal{L}$ if $\mathcal{L} > 1$, thus ruling out as potentially unstable and biased L_2 and *Huber's smooth* losses;
- Also: more straightforward training monitoring ("> 2.5" heuristic)

We are here...

1. *Theory first!*
2. *Deep learning works great, except when it doesn't.*
3. *No data, no learning!*
4. *So, what?*

Dataset

The dataset contains informations about stock prices of many US companies in the period from 2 January 1962 to 27 March 2018.

ticker	date	open	high	low	close	volume
A	1999-11-18	45.50	50.00	40.0000	44.00	44739900.0
A	1999-11-19	42.94	43.00	39.8100	40.38	10897100.0
A	1999-11-22	41.31	44.00	40.0600	44.00	4705200.0
A	1999-11-23	42.50	43.63	40.2500	40.25	4274400.0
A	1999-11-24	40.13	41.94	40.0000	41.06	3464400.0

Windowing

Changing everything to make nothing change

Presenting data in overlapping-window form to RNNs is the first step to effectively *teacher-force* them *(Lamb, 2016; Hafner, 2018)*.

$$
\begin{array}{l}
a\ b\ c \mid d\ e\ f \mid g\ h\ i \mid \ \ldots \\
b\ c\ d \mid e\ f\ g \mid h\ i\ j \mid \ \ldots \\
c\ d\ e \mid f\ g\ h \mid i\ j\ k \mid \ \ldots
\end{array}
$$

(note: data ordered by column!)

However, we often used very large windows to further improve learning on the most recent data.

Data batching

You probably always made batches for RNNs wrong!

The second (and last) step is doing the batches in the right manner! *(Lamb, 2016; Hafner, 2018)*.

$$
\begin{array}{l}
a\ b\ c \mid d\ e\ f \mid g\ h\ i \mid \ \ldots \\
b\ c\ d \mid e\ f\ g \mid h\ i\ j \mid \ \ldots \\
c\ d\ e \mid f\ g\ h \mid i\ j\ k \mid \ \ldots
\end{array}
$$

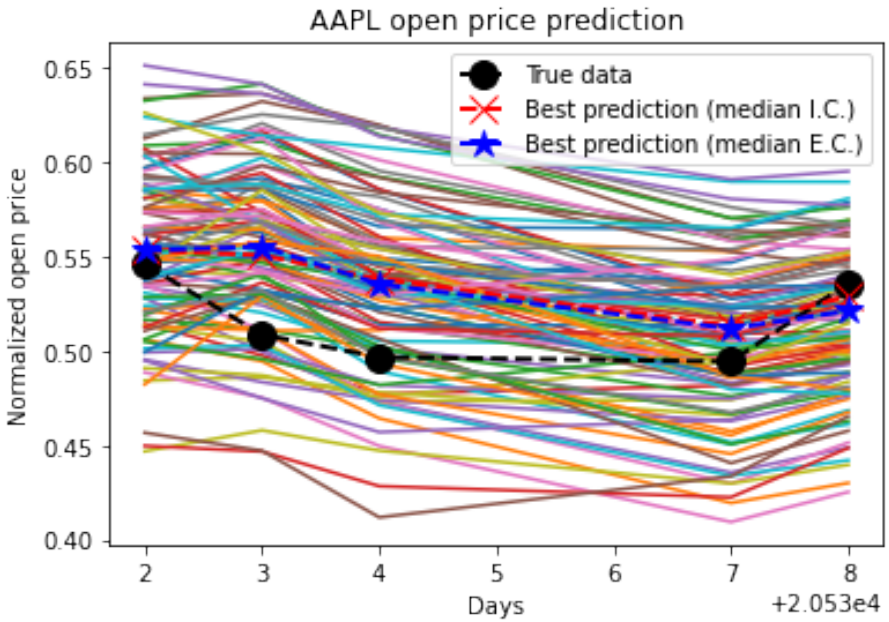
- The usual (and almost-always unchangeable) *first-K-and-batch* approach leads to hidden-state dissociation!
- Hidden-state averaging can be limiting and slow;

$\rightarrow$ Use of the *multi-head-disk seek algorithm*.

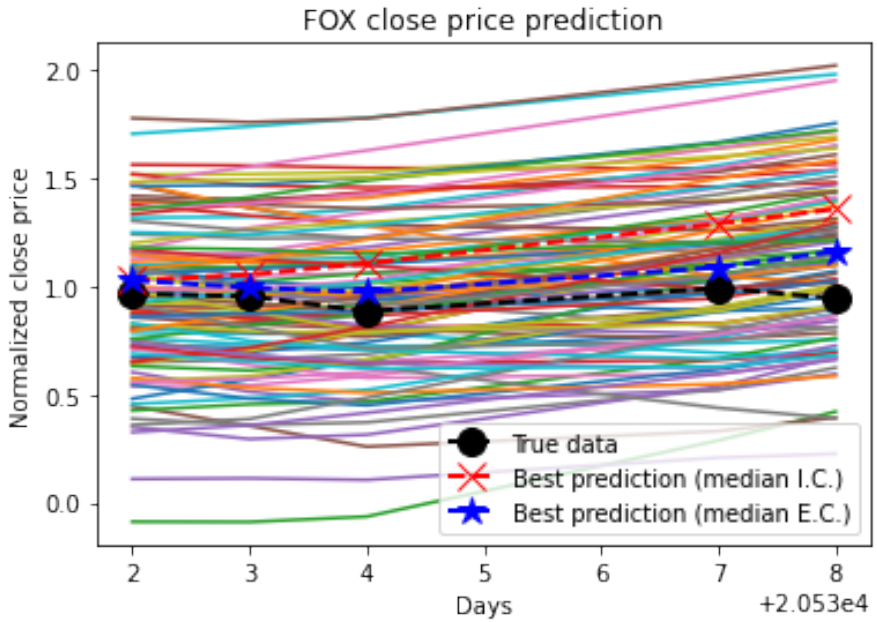
We are here...

1. *Theory first!*
2. *Deep learning works great, except when it doesn't.*
3. *No data, no learning!*
4. *So, what?*

Results



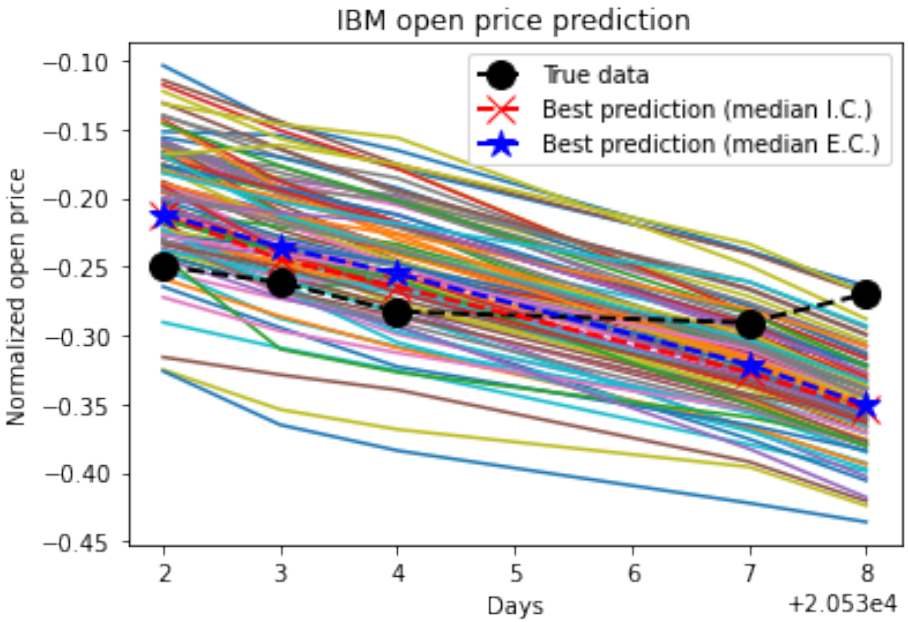
Prediction of AAPL open price.



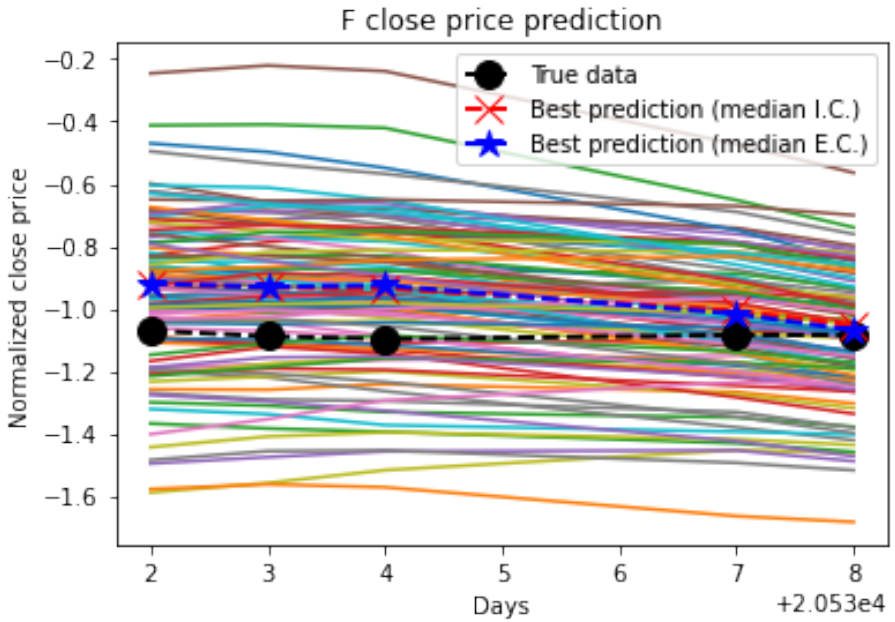
Prediction of FOX close price.

	MAE on Median I.C.	MAE on Median E.C.
AAPL	0.0051	0.0066
FOX	0.1836	0.3016

Results



Prediction of IBM open price.



Prediction of F close price.

	MAE on Median I.C.	MAE on Median E.C.
IBM	0.0086	0.0068
F	0.0311	0.0347

Final remarks and future work (1)

Just one – even successful – try is never enough!

"We never wanted to build an infallible oracle, but just to ease risk management."

Ritchie Ng (Chief of AI – Hessian Matrix Capital)

Final remarks and future work (2)

Just one – even successful – try is never enough!

Overall, a moderately-successful application of *state-of-the-art* NN methods to a real-world problem, despite the lack of settled knowledge on the matter and training instabilities.

- Fairly *close-to-reality* predictions provided a successful training;
- Usable uncertainty estimation for *limit-case* scenarios;
- Fast and introspectable training ($\leftarrow$ heuristics).

Not a perfect one, though...

- Extreme brittleness of hyperparameter tuning;
- Too broad uncertainty for finer-grained analysis;
- Deep $\mathcal{L}$ local minima may require training restart.

Greetings

Thank you for your attention!

[https://github.com/emaballarin/cathode]