Deep Learning in Finance

week 6: prediction

Damien Challet
damien.challet@centralesupelec.fr

10th December 2024

What to predict in Finance

1. Prices (returns) !
2. Volatility
3. Interest rates
4. Trader behaviour
5. Events (e.g. limit order book)
6. ...

Standard approaches

- (V)ARIMA(X) models, exponential smoothing
- Simplest non-linear extension: higher-order factorization [preprint]

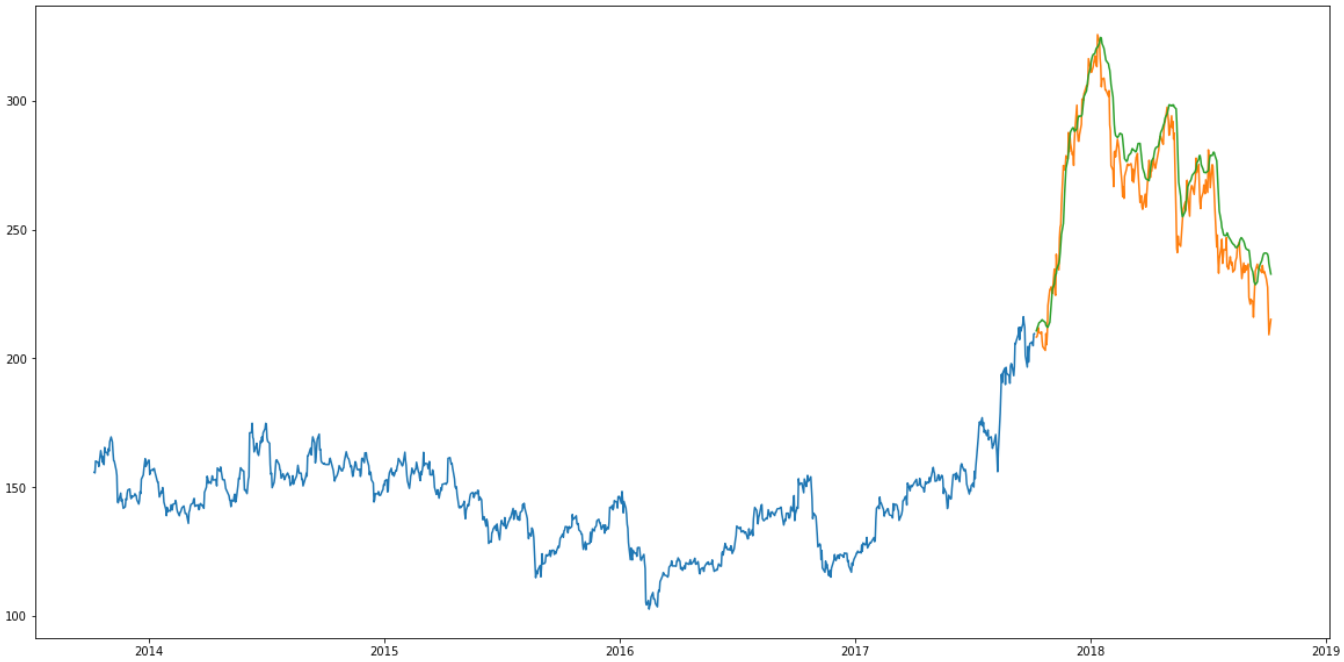
$$\hat{y}(\boldsymbol{x}) := v^{(0)} + \sum_{j_1=1}^{p} x_j v_{j_1}^{(1)} + \sum_{j_1=1}^{p} \sum_{j_2=j_1+1}^{p} x_{j_1} x_{j_2} \sum_{f=1}^{k_2} v_{j_1,f}^{(2)} v_{j_2,f}^{(2)},$$

How to include and extend these models with NNs?

A short history of (wrong) price predictions with machine learning

- 1990s: NNs
- 1995: SVM
- 2001: Random Forests
- 2008: Deep Learning
- 2023: LLMs

Summary (image from [link])



What not to do

1. Take price p_t or price returns R_t
2. Rescale the whole data
3. Cut into 60/20/20 train/validate/test
4. Minimise error of predicted price or returns
5. Be happy

Prediction: what is the question?

- How to invest
  - directional bet on asset / volatility
  - strategy selection
  - portfolio optimisation
- Decision making, e.g. broker
  - keep trade or not
  - interesting assets for clients
  - interesting clients for assets

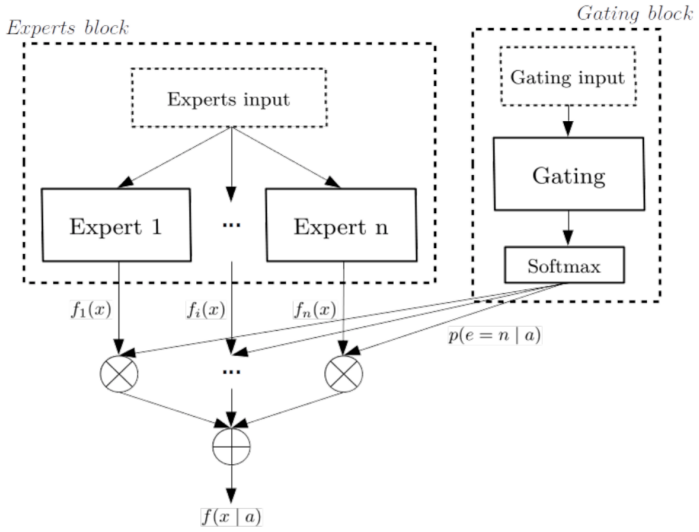
Broker: prediction of interest

DEEP PREDICTION OF INVESTOR INTEREST: A SUPERVISED CLUSTERING APPROACH

Baptiste Barreau[1, 2], Laurent Carlier[2], and Damien Challet[1]

[1]Chair of Quantitative Finance, MICS Laboratory, CentraleSupélec, Université Paris-Saclay, Gif-sur-Yvette, France
[2]BNP Paribas Corporate & Institutional Banking, Global Markets Data & Artificial Intelligence Lab, Paris, France
{baptiste.barreau, laurent.carlier}@bnpparibas.com
damien.challet@centralesupelec.fr}

[link]



Broker: prediction of trade toxicity

Can Deep Learning Predict Risky Retail Investors? A Case Study in Financial Risk Behavior Forecasting

Y. Yang[†,a], A. Kim[†,b], S. Lessmann[†,b], T. Ma[†,c], M.-C. Sung[†,c,*], J.E.V. Johnson[†,c]

[a] *Department of Computer Science, University College London*
[b] *School of Business and Economics, Humboldt-University of Berlin*
[c] *Southampton Business School, University of Southampton*

[link]

CFD: keep trade in own book or send it to the market

Good practices

Raw returns are not very useful

1. Strategy $\equiv$ filter $\equiv$ diagnostic tool
2. Reduce complexity of inputs
3. Reduce complexity of aims

Input: filtered returns (strategies)

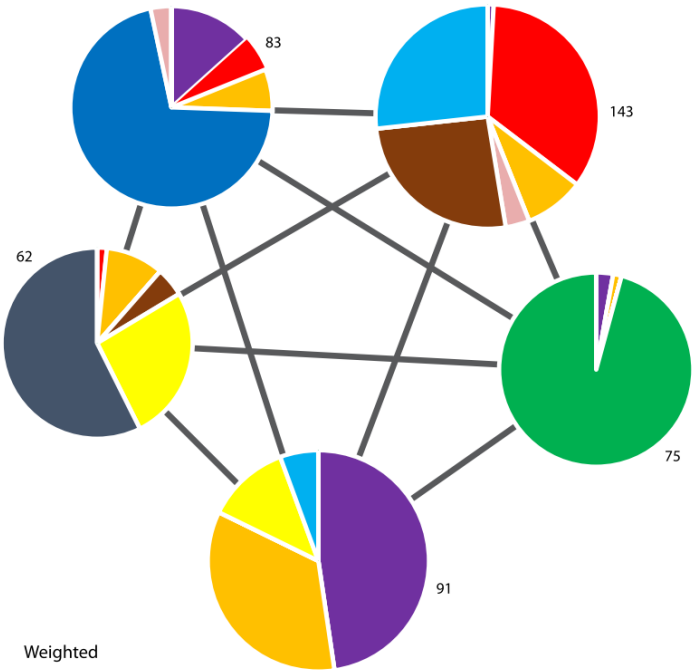
Price returns are mostly noise

no filtering, no hope

some filtering, some hope

Complexity of inputs

How many bits of information in a price return?
Almog *et al.* (2015) [link]



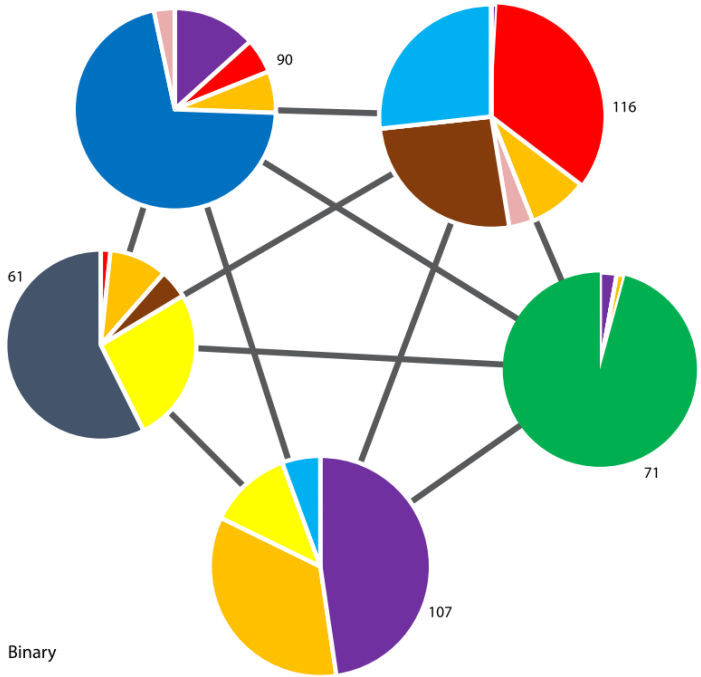


Fig 4. Communities of the S&P 500 (daily closing prices from 2001 to 2011) generated using the modified Louvain algorithm [7]. Each community is labelled with the number of stocks and the pie chart represents the relative composition of each community based on the industry sectors of the constituent stocks (color legend in Table 1). The inter-community link weights are negative, indicating that the communities are all residually anti-correlated.

doi:10.1371/journal.pone.0133679.g004

Return complexity

- Raw returns

$$r_{i,t} \in \mathbb{R}$$

- Sign returns

$$s_{i,t} = \text{sign} r_{i,t} \in \{-1, 0, +1\}$$

- Sign of excess return

$$x_{i,t} = \text{sign}\left[r_{i,t} - r_t^{(market)}\right]$$

- Quantiles
  - fixed
  - volatility-dependent
  - transaction cost-dependent

Some kinds of filtering

- Better estimators (Hurst, robust estimators)
- Better/more data
  - focus on time periods
  - OHLC ratios, volume
  - [...]
- NN do the final filtering

$\rightarrow$ representation?

Further reading

[link]

Enhancing Time Series Momentum Strategies Using Deep Neural Networks

Bryan Lim, Stefan Zohren, Stephen Roberts

- rescaled MACD

Aim: find good representation and time filtering

Representation

- CNN (Hurst, moving averages, 2D images)
- Autoencoder
  - Market states
  - LOB state
- Signature [Bonnier et al. 2019]
  - Sig-WGAN [link]: conditional WGAN in signature space

Predict best/worst k assets

European Journal of Operational Research 270 (2018) 654–669

Contents lists available at ScienceDirect

European Journal of Operational Research

journal homepage: www.elsevier.com/locate/ejor

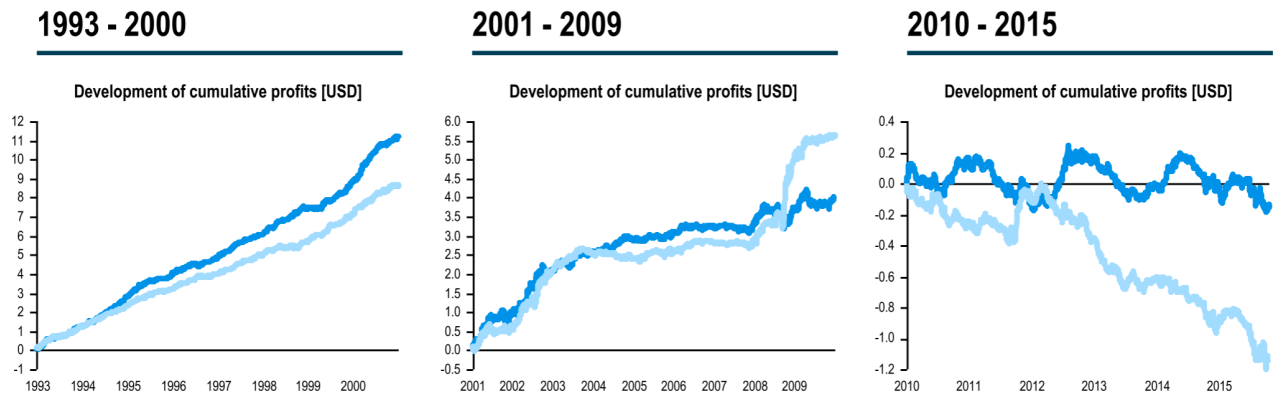
Computational Intelligence and Information Management

Deep learning with long short-term memory networks for financial market predictions

Thomas Fischer[1,*], Christopher Krauss[1]

Department of Statistics and Econometrics, University of Erlangen-Nürnberg, Lange Gasse 20, 90403 Nürnberg, Germany

[link]



Predict monthly returns with fundamental predictors

Deep Learning for Predicting Asset Returns

Guanhao Feng* — *College of Business*, *City University of Hong Kong*

Jingyu He† — *Booth School of Business*, *University of Chicago*

Nicholas G. Polson ‡ — *Booth School of Business*, *University of Chicago*

April 25, 2018

Variable	Description
d/p	dividend price ratio
d/y	dividend yield ratio
e/p	earning price ratio
d/e	dividend payout ratio
svar	stock variance
b/m	book to market ratio
ntis	net issues
tbl	treasury bills rate
ltr	long term rate
tms	term spread
dfy	default spread
dfr	default return spread
infl	consumer price index
cay	consumption, wealth, income ratio

[link]

Spot the mistake

A deep learning framework for financial time series using stacked autoencoders and long-short term memory

Wei Bao[1], **Jun Yue**[2]*, **Yulei Rao**[1]

1 Business School, Central South University, Changsha, China, **2** Institute of Remote Sensing and Geographic Information System, Peking University, Beijing, China

* jyue@pku.edu.cn

Abstract

The application of deep learning approaches to finance has received a great deal of attention from both investors and researchers. This study presents a novel deep learning framework where wavelet transforms (WT), stacked autoencoders (SAEs) and long-short term memory (LSTM) are combined for stock price forecasting. The SAEs for hierarchically extracted deep features is introduced into stock price forecasting for the first time. The deep learning framework comprises three stages. First, the stock price time series is decomposed by WT to eliminate noise. Second, SAEs is applied to generate deep high-level features for predicting the stock price. Third, high-level denoising features are fed into LSTM to forecast the next day's closing price. Six market indices and their corresponding index futures are chosen to examine the performance of the proposed model. Results show that the proposed model outperforms other similar models in both predictive accuracy and profitability

Spot the mistake

A Deep Learning Framework for Univariate Time Series Prediction Using Convolutional LSTM Stacked Autoencoders

Publisher: IEEE

Cite This PDF

Aniekan Essien ; Cinzia Giannetti **All Authors**

Abstract:

This paper proposes a deep learning framework where wavelet transforms (WT), 2-dimensional Convolutional Neural Networks (CNNs) and Long Short-Term Memory (LSTM) stacked autoencoders (SAE) are combined towards single-step time series prediction. Within the framework, the input dataset is denoised using wavelet decomposition, before learning in an unsupervised manner using SAEs comprising bidirectional Convolutional LSTM (ConvLSTM) layers to predict a single-step ahead value. To evaluate our proposed framework, we compared its performance to two (2) state-of-the-art deep learning predictive models using three open-source univariate time series datasets. The experimental results support the value of the approach when applied to univariate time series prediction.

Learning in a non-stationary world: splits

1. Whole time series: $\rightarrow$ obtain a stationary model
   Train / validation / test window

2. Partial time series:

- Expanding windows: if same dynamics, more data

- Rolling windows: reactivity, intrinsically non-stationary world

Note: Repeated learning + evolving model $\equiv$ effectively stationary modelling

Predictions with validation window

Hyperparameter tuning

- architecture
- depth
- activation function
- learning rate schedule
- losses
- ...

Procedure:

1. For each set of parameters, train a network
2. Test each network in validation period
3. Use the best validation window method in the test window

Hyper-parameter optimization

See review, Yu and Zu (2020) [link]

1. random exploration: computation budget or precision goal [talos]

2. grid search [talos]

3. Bayesian parameter tuning [hyperopt]
   - model-based learning of the dependence of loss on parameters
   - common hypothesis: Gaussian model
   - predict next improving parameters

4. ... [hyperopt]

Note: HPO is necessary (in principle) for each rolling window.

Hot-cold-lukewarm training

- Cold training — new model in each rolling window
- Hot training — continue training the previous model
- Lukewarm training — continue training under-optimized model
  - Online early stopping [link]

Prediction of less noisy data: fundamentals to predict returns

Improving Factor-Based Quantitative Investing by Forecasting Company Fundamentals

John Alberg
Euclidean Technologies
john.alberg@euclidean.com

Zachary C. Lipton
Amazon AI
Carnegie Mellon University
zlipton@cmu.edu

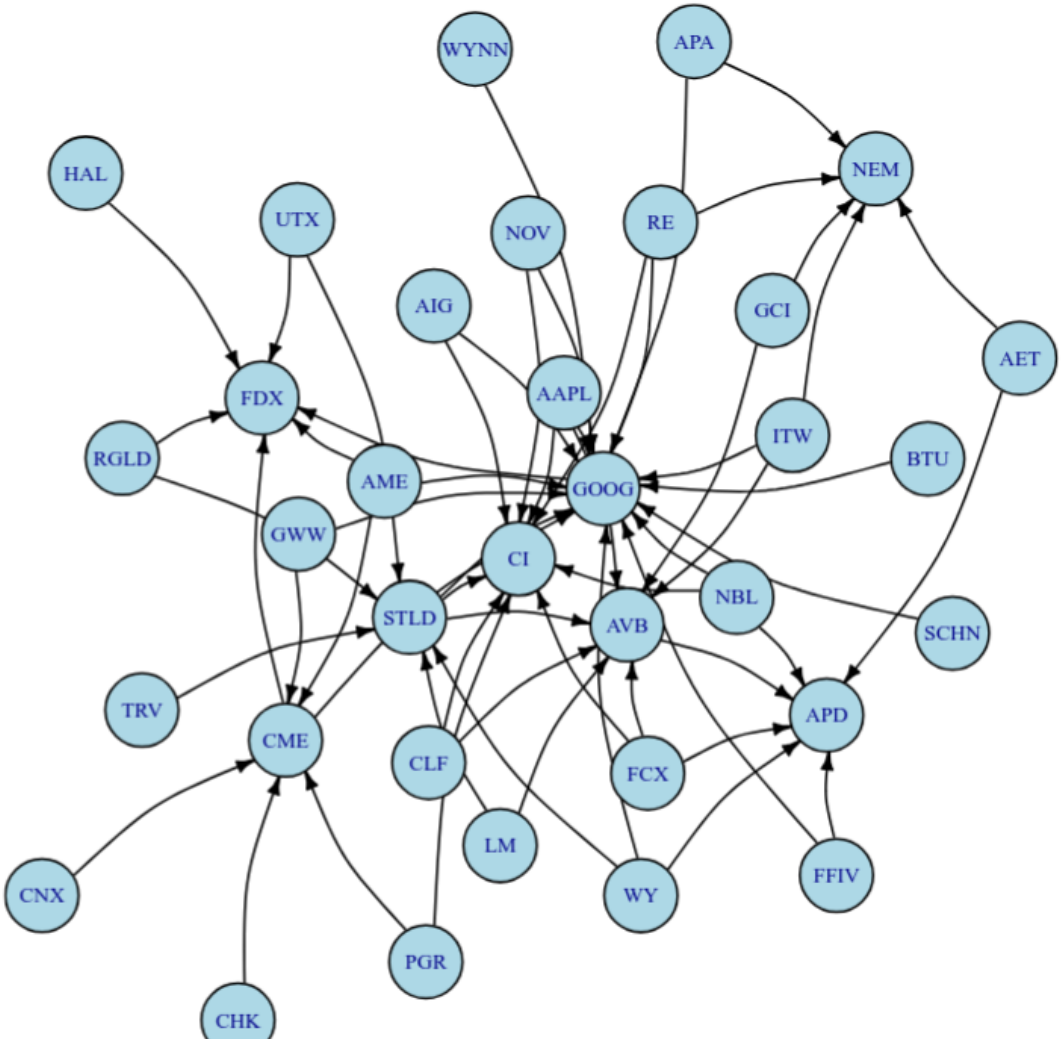
[link]

Predict representation to predict returns

- Factors
- Autoencoder state
- Mamba / Samba
- SimStock [preprint]

Why AE to predict stock returns?

Validated prediction networks (Challet et al. 2022 [preprintpreprint])



Neural architectures for timeseries

Sophisticated moving averages

- CNN: short attention span
- TCN: larger perceptive field

Longer memory: *internal state needed*

Recurrent NNs:

- Vanilla RNNs
- Gated Reccurent Unit (GRU)
- Long-Short-Term Memory (LSTM)

State-space models (attention with linear scaling)

- Mamba [preprint]
- Samba [preprint] (no code)

TCN

Oord et al. 2016 [preprint]

stacked CNN with dilation, no memory

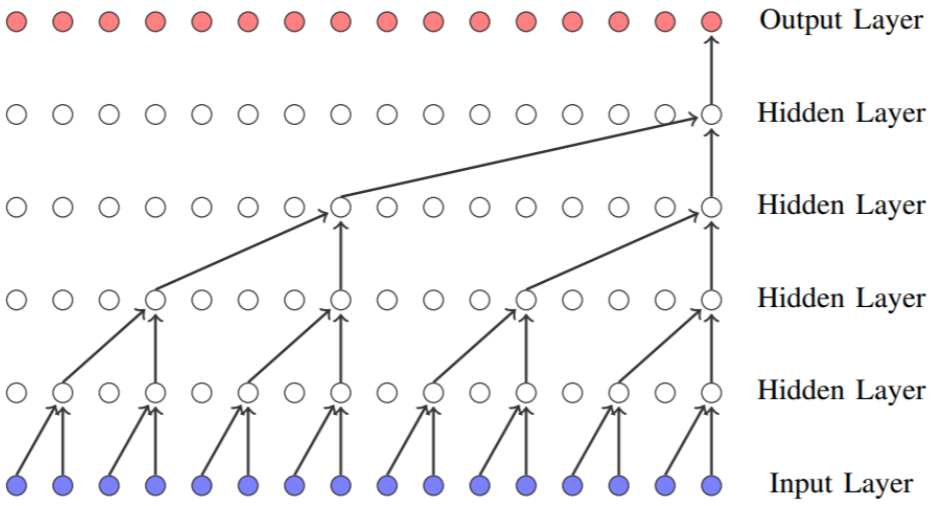
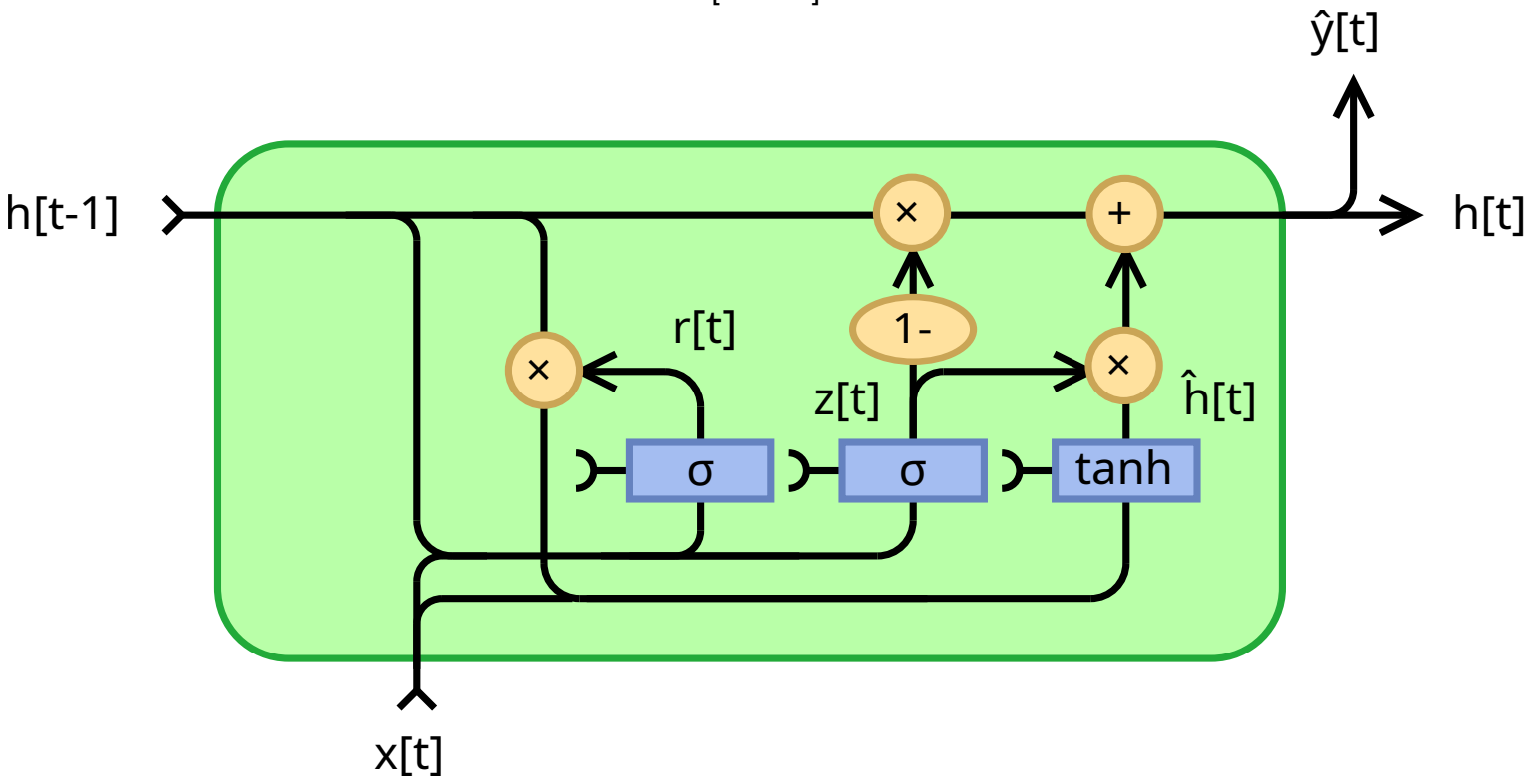


Figure 6: Vanilla TCN with 4 hidden layers, kernel size $K = 2$ and dilation factor $D = 2$ (cf. van den Oord et al. (2016)).

[keras-tcn]

Gated Recurrent Unit (GRU)

[source]



Gated Recurrent Unit (GRU)

Cho et al. (2014) [preprint]

input: $t-$th line of sample i: $x_t \in \mathbb{R}^{N_F}$
output $h_t \in \mathbb{R}^{No}$

$$\begin{aligned} h_t &= (1 - \lambda_t) \odot h_{t-1} + \lambda_t \odot \hat{h}_t \\ &= h_{t-1} + \lambda_t \odot (\hat{h}_t - h_{t-1}) \end{aligned}$$

where

- $\odot$ is the Hadamar (element-wise) product
- $\lambda_t \in \mathbb{R}^{No}$: exponential moving average.
- $\hat{h}_t$: new proposed output
- Note: h_t is the hidden state / memory

Gated Recurrent Unit (GRU)

Input $x_t \in \mathbb{R}^{N_I}$
Output $h_t \in \mathbb{R}^{N_O}$

$$h_t = (1 - \lambda_t) \odot h_{t-1} + \lambda_t \odot \hat{h}_t$$

where

- λ_t : exponential moving average vector.

$$\lambda_t = \sigma_\lambda(W_\lambda x_t + U_\lambda h_{t-1} + b_\lambda)$$

- $\hat{h}_t$: new proposed output vector

$$\hat{h}_t = \sigma_h\left[W x_t + U_h(r_t * h_{t-1}) + b_h\right]$$

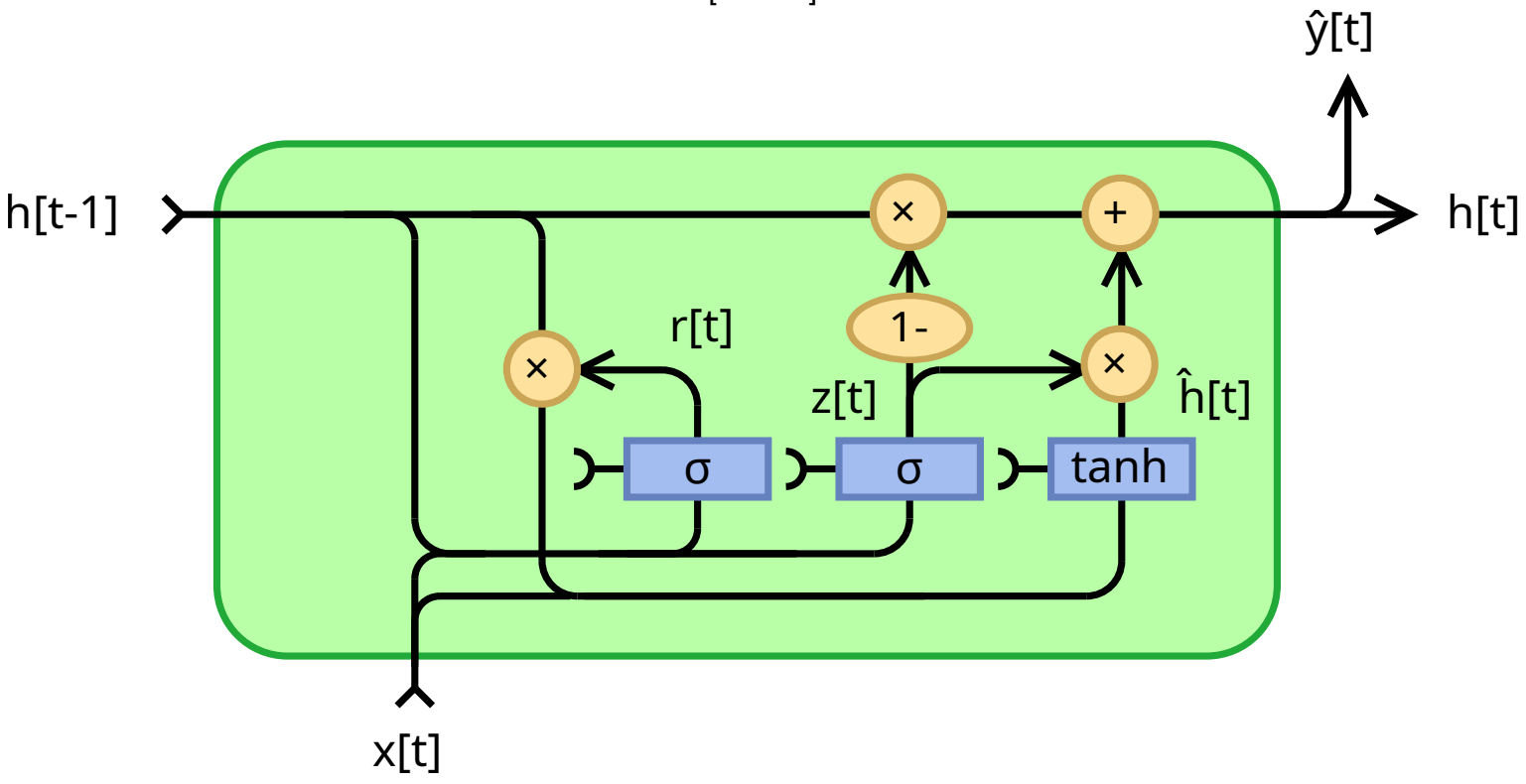
- r_t: reset (when $\simeq 0$) $\equiv$ forget

$$r_t = \sigma_r\left[W_r x_t + U_r h_{t-1} + b_r\right]$$

- Notes:
  - h_t is the output and the state kept in memory
  - λ_t: linear forgetting of h_t
  - r_t: non-linear inhibition within $\hat{h}_t$

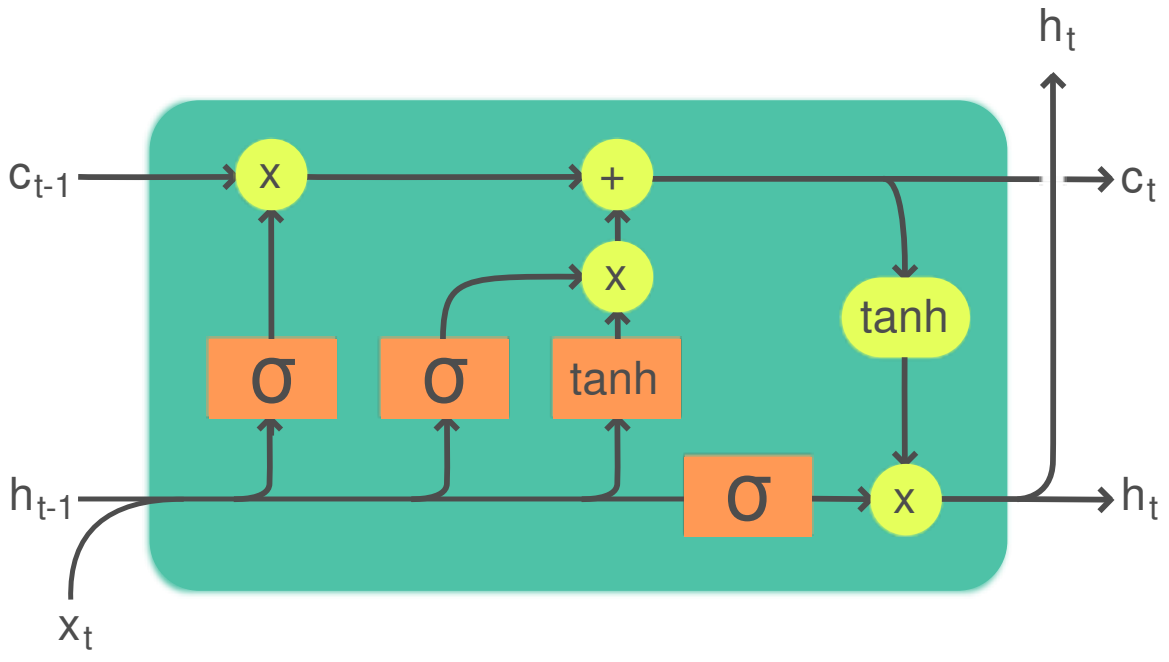
GRU: graphically

[source]

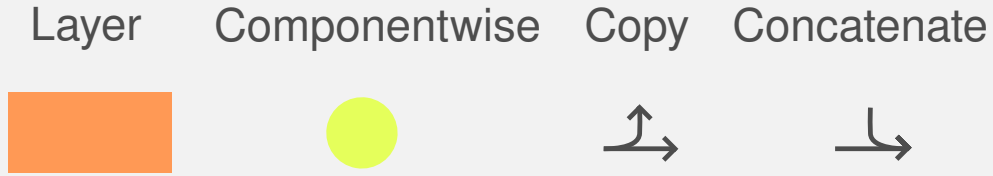


Long- short-term memory (LSTM)

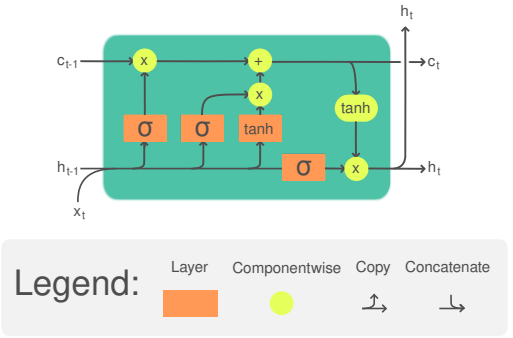
GRU with richer history [source]



Legend:



LSTM



- h_t and $c_t \in \mathbb{R}^{N_H}$
- N_H = dimension of hidden states
  $\rightarrow$ if $N_O \neq N_H$, add another (e.g. Dense) layer
- x_t interacts with h_{t-1} to compute c_t from c_{t-1}
- h_{t-1} obtained from c_t

LSTM dynamics

- Hidden states h_t is modulated by the cell values c_t

$$h_t = A(o_t \odot c_t), \quad A(x) = x \text{ or } A(x) = \tanh x$$

- Output gate: mixing of x_t and h_{t-1}+ nonlinear activation

$$o_t = \text{sigmoid}(W_o x_t + U_o h_{t-1} + b_o)$$

- Activation vector (cell)

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t$$

- Forget gate (remember to me)

$$f_t = \text{sigmoid}(W_f x_t + U_f h_{t-1} + b_f)$$

- Input gate

$$i_t = \text{sigmoid}(W_i x_t + U_i h_{t-1} + b_i)$$

- Input gate activation vector

$$\tilde{c}_t = \tanh(W_c x_t + U_c h_{t-1} + b_c)$$

Training recurrent neural networks

- Moving average

$$h_t = (1 - \lambda_t) h_{t-1} + \lambda_t \hat{h}_t, \quad h_0 = 0$$

- Memory length: $\lambda_t \equiv \lambda$

$$\begin{aligned} h_{t+1} &= (1 - \lambda)^t \hat{h}_1 + \ldots \\ &= e^{t \ln(1-\lambda)} \hat{h}_1 + \ldots \\ &= e^{-t/\tau_0} \hat{h}_1 + \ldots \end{aligned}$$

with

$$\tau_0 = \frac{1}{|\ln(1 - \lambda)|} \simeq \frac{1}{\lambda} \quad \lambda \ll 1$$

- Example

$$\lambda = 0.01 \qquad \tau_0 \simeq 100$$

Data structure for xCNs and RNNs

- Full data
$$X \in \mathbb{R}^{T \times N_F}, \quad Y \in \mathbb{R}^{No}$$
- Single sample: $N_I < T$
$$X_i \in \mathbb{R}^{N_I \times N_F}$$
- Truth
$$Y_i \in \mathbb{R}^{No}$$
- Example ($N_F = 1$, $N_0 = 1$)
$$X_i = (x_{i,1}, \cdots, x_{i,T})^\dagger, Y_i = H_i \text{ (Hurst exponent)}$$
- Example ($N_F = 2$, $N_0 = 3$)
$$X_i = \begin{pmatrix} a_{i,1} & b_{i,1} \\ \vdots & \vdots \\ a_{i,N_I} & b_{i,N_I} \end{pmatrix} \quad Y_i = \begin{matrix} \\ \\ (\alpha_{N_I+1} & \beta_{N_I+1} & \gamma_{N_I+1}) \end{matrix}$$
- Batch: array of samples (sequences), dimension (batch_size, N_I, N_F)

LSTM continued

- LSTM: learn representation that optimises something from h_t (loss)
- Feeding LSTMs with raw price returns
  - hope: useful representation
  - reality: need to learn market state, H, fundamental data, complex estimators, path signatures, ...
- Encoder before LSTMs
  - CNN
  - Autoencoder
- Encoder in LSTMs: [ConvLSTM1D]

LSTM: hidden state/cell dynamics

- Initially, $h = 0$, $c = 0$
- Sample i

$$X_i = \begin{pmatrix} a_{i,1} & b_{i,1} \\ \vdots & \vdots \\ a_{i,N_I} & b_{i,N_I} \end{pmatrix} \quad Y_i = \begin{pmatrix} \alpha_{N_I+1} & \beta_{N_I+1} & \gamma_{N_I+1} \end{pmatrix}$$

- LSTM
  - applies W_i, W_o, ...
  - computes h_t, c_t, $t = 1, \cdots, N_I$
  - outputs final $h_{t=N_I}$
- Batch: $i = 1, \cdots,$ batch_size
  - for $i > 1$, $h_{t=1}^{(i)}$ comes from $h_{t=1}^{(i-1)}$
- After each batch, $h = 0$, $c = 0$

LSTM: read the doc

```
tf.keras.layers.LSTM(
    units,
    activation="tanh",
    recurrent_activation="sigmoid",
    use_bias=True,
    kernel_initializer="glorot_uniform",
    recurrent_initializer="orthogonal",
    bias_initializer="zeros",
    unit_forget_bias=True,
    kernel_regularizer=None,
    recurrent_regularizer=None,
    bias_regularizer=None,
    activity_regularizer=None,
    kernel_constraint=None,
    recurrent_constraint=None,
    bias_constraint=None,
    dropout=0.0,
    recurrent_dropout=0.0,
    return_sequences=False,
    return_state=False,
    go_backwards=False,
    stateful=False,
    time_major=False,
    unroll=False,
    **kwargs
)
```

What to do with $\{h_t\}$: stacked LSTM

- LSTM
  - applies W_i, W_o, ...
  - computes h_t, c_t, $t = 1, \cdots, N_I$
  - outputs $h_{t=N_I}$
- Lost information: $h_{t<N_I}$
- `return_sequences=True` returns $\{h_{1,} \cdots, h_{N_I}\}$
- Input $h-$sequence to another LSTM: stacking
- $n-$times stacked LSTM
- Note: N_H can vary along the LSTM stack

Return sequences: LSTM auto-encoder

- Encoder: LSTM: from N_F features to N_H dimensions
- Return sequences $\equiv$ encodings
- Stacking $N_H^{(2)} = N_F$
- Useful if dynamics of features relevant

What to do with LSTM output

- Dense network $\rightarrow$ output dimension
- Stack LSTMs: h_t is the input of another LSTM
- Attention: conditional weights of the past [link] (LLM, e.g. [alpaca])

Attention is **all you need**

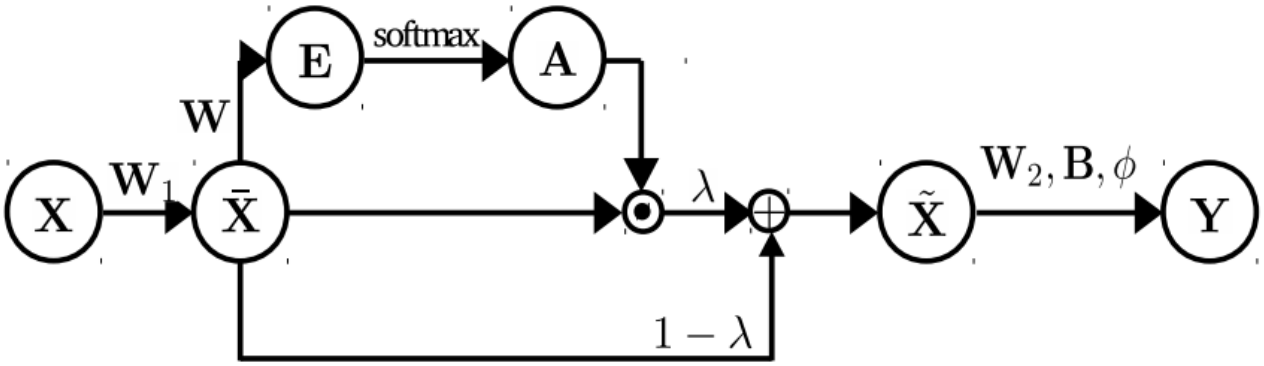
A Vaswani, N Shazeer, N Parmar… - Advances in neural …, 2017 - proceedings.neurips.cc

… the number of **attention** heads and the **attention** key and value dimensions, keeping the amount of computation constant, as described in Section 3.2.2. While single-head **attention** is 0.9 …

☆ Save 99 Cite Cited by 70387 Related articles All 46 versions »

TABL: limit order book prediction

Temporal attention + bilinear filtering, Tran et al (2017) [preprint]



$$\bar{\mathbf{X}} = \mathbf{W}_1\mathbf{X}$$

$$\mathbf{E} = \bar{\mathbf{X}}\mathbf{W}$$

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T}\exp(e_{ik})}$$

$$\tilde{\mathbf{X}} = \lambda(\bar{\mathbf{X}} \odot \mathbf{A}) + (1-\lambda)\bar{\mathbf{X}}$$

$$\mathbf{Y} = \phi\big(\tilde{\mathbf{X}}\mathbf{W}_2 + \mathbf{B}\big)$$

Multi-head TABL

Shabani *et al.* (2022) [link]

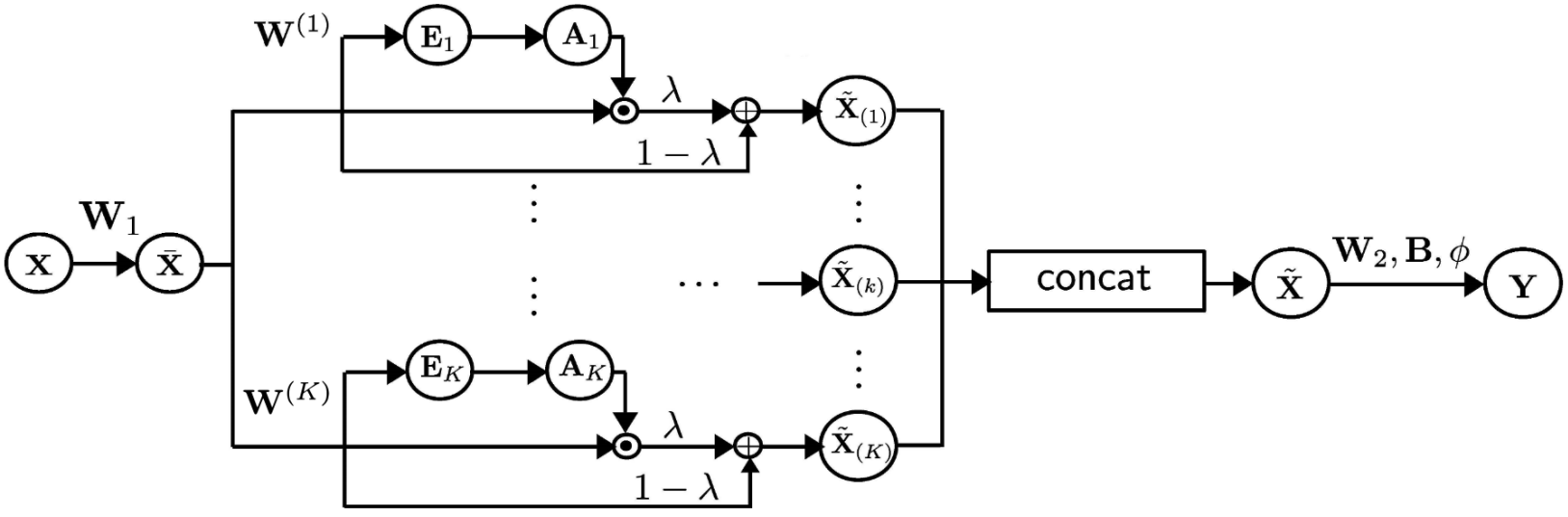
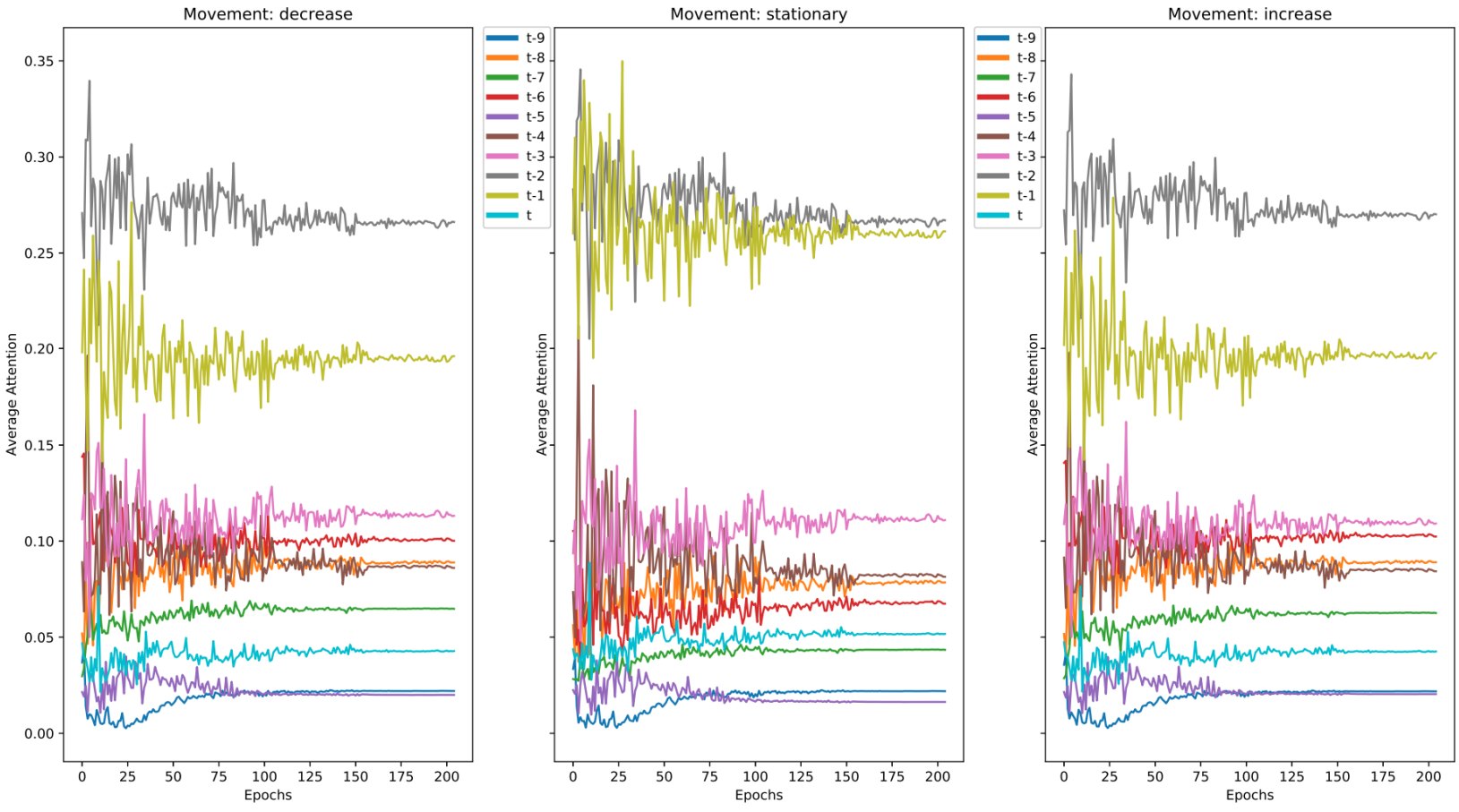


Fig. 1. Schematic illustration of the proposed `MTABL` layer

TABL: attention where you need it

Aim: predict Limit Order Book mid-price move in 10 ticks, tick-by-tick



MTABL: results

Aim: predict Limit Order Book mid-price move in 10 ticks, tick-by-tick

TABLE I
PERFORMANCES OF MULTI-HEAD ATTENTION MODELS USING CONCATENATION (MTABL−C) (MEAN ± STD.)

Topology	Layer	Accuracy (%)	Precision (%)	Recall (%)	F1-Score (%)
A	TABL	67.21±0.045	53.76±0.039	55.47±0.015	54.25±0.03
	MTABL−C-2	69.78±0.029	56.64±0.029	59.58±0.026	57.81±0.029
	MTABL−C-3	72.45±0.009	59.03±0.009	60.41±0.001	59.66±0.005
	MTABL−C-4	72.30±0.007	59.25±0.007	61.60±0.005	60.28±0.006
	MTABL−C-5	**72.57±0.003**	**59.63±0.003**	**62.68±0.005**	**60.90±0.004**
B	TABL	78.56±0.002	67.55±0.003	71.07±0.004	69.10±0.002
	MTABL−C-2	77.68±0.004	66.44±0.004	70.56±0.007	68.18±0.004
	MTABL−C-3	78.13±0.007	67.04±0.009	71.39±0.004	68.89±0.007
	MTABL−C-4	77.63±0.005	66.48±0.006	70.89±0.003	68.35±0.005
	MTABL−C-5	**78.22±0.012**	**67.4±0.017**	**71.52±0.004**	**69.16±0.012**
C	TABL	83.52±0.009	75.12±0.013	77.02±0.006	76.01±0.009
	MTABL−C-2	83.69±0.005	75.21±0.008	77.74±0.004	76.39±0.006
	MTABL−C-3	81.64±0.014	72.16±0.022	75.17±0.015	73.54±0.019
	MTABL−C-4	**83.71±0.01**	**75.37±0.015**	**77.63±0.006**	**76.42±0.011**
	MTABL−C-5	82.63±0.004	73.66±0.005	76.93±0.008	75.16±0.006

Attention on $\{h_t\}$

- by hand [machinelearningmastery.com]
- attention-decoder, [machinelearningmastery.com]

Transformers for time series?

- Not good for long-term predictions [preprint] (1500 citations) [Betteridge law]

The Thirty-Seventh AAAI Conference on Artificial Intelligence (AAAI-23)

Are Transformers Effective for Time Series Forecasting?

Ailing Zeng[1,2*], **Muxi Chen**[1*], **Lei Zhang**[2], **Qiang Xu**[1]
[1]The Chinese University of Hong Kong
[2]International Digital Economy Academy
{zengailing, leizhang}@idea.edu.cn,{mxchen21, qxu}@cse.cuhk.edu.hk

- Scale as N_{inputs}^2 or T_{in}^2

LSTM vs rough volatility

- Rosenbaum et al. [link (2022)] [link (2023)]
  average LSTM and rough volatility yield the same prediction accuracy

- Challet and Ragel [link (2023)]
  best LSTMs better than rough volatility

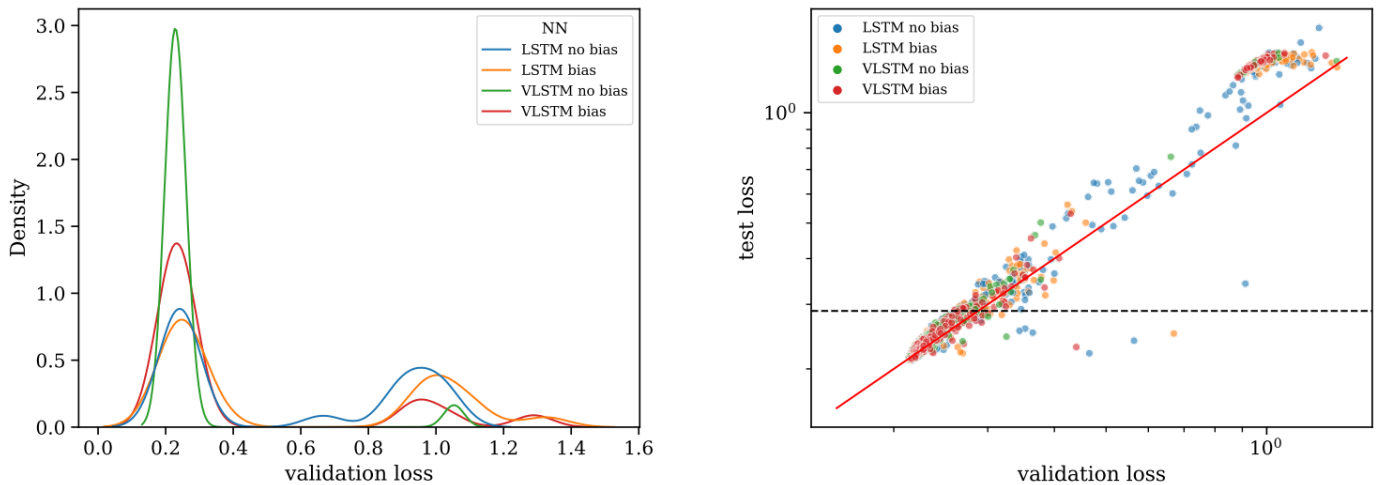
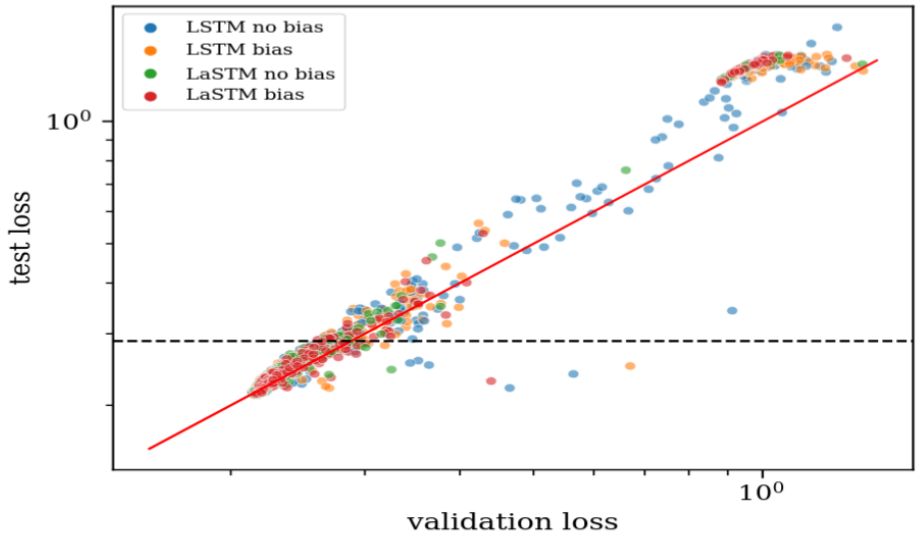


Figure 3: Left plot: density of validation losses by architecture. Right plot: test loss vs validation loss. Multiple volatility time series prediction. The dashed line is the average MSE of predictions made with rough volatility models.

LSTMs: single train/val/test

Hard-to-train NNs

- Single train period
  - population of LSTMs
  - check consistency train/val or val/test + test2



LSTMs: rolling train / val / test

- Population of LSTMs
- Hot training
- Genetic Evolution
  - replace worst %10 ones (e.g.)
  - random start or offspring of the the best one
  - use the $k-$best ones

From 2-D to 3-D data

(source code in provided notebook)

1. Prepare predictor matrix $X \in \mathbb{R}^{T \times N_F}$
   T is the total length of available data
   N_F is the total number of features (predictors)

2. Hyp: NN takes input of N_I timesteps (subset of X)

3. for each $t_1 \in \{N_I, T_{train}\}$,
   - define $X_{t_1}^{(3d)} = X_{\{t_1 - N_{I,}, t_1\},:}$: lines $t_1 - N_i$ to t_1 of matrix X
   - NN learns the correspondence between $X_{t_1}^{(3d)}$ and y_{t_1} (hidden dynamics)
   - Volatility prediction: $y_{t_1} = \sigma_{t_1+1}$.

Further peculiarities of LSTM (in Keras)

- MLP:
  - matrix of $X_{train} \rightarrow$ fit()
  - matrix of $X_{test} \rightarrow$ predict(). Sizes may differ
- LSTM
  - 3D-array of $X_{train} \in \mathbb{R}^{N_{samples} \times T \times N_F} \rightarrow$ fit()
  - 3D-array of $X_{test} \in \mathbb{R}^{N_{samples} \times T \times N_F} \rightarrow$ predict: SAME size
  - Solutions: [link]
    - my favourite: define new model with same T and N_F, but with a different $N_{samples}$, and copy weights from trained model (code provided)

Single-sequence prediction with LSTMs

- Technical limitation of Keras: prediction length = batch size
- Solution: copy weights

```
myLSTM_2 = build_LSTM(neurons_lstm,dT,X.shape)

for t1 in t1s:
    t0=t1-T_in
# ...
    myLSTM=build_LSTM(neuron,batch_size,Xshape)
    myLSTM.fit(X_scaled,y_scaled,
               batch_size=batch_size,epochs=num_epochs,
               callbacks=[es],validation_split=0.2)

    myLSTM_2.set_weights(myLSTM.get_weights())
# ...
    preds=myLSTM_2.predict(X_oos_scaled,
                           batch_size=dT)
```

Conclusions

Beyond non-linear black box:

- be explicit with what you want
- say what you know
- have test cases
- experiment