Deep learning in Finance

Damien Challet
damien.challet@centralesupelec.fr

22nd October 2024

Course philosophy

DL from the point of view of Finance

Course philosophy

- How/when not to use DL
- Methodology:
  - good filtering?
  - pre-knowledge?
- how to talk to DL
  - inputs
  - architecture
  - check default options
  - result interpretation $\leftrightarrow$ filtering
- Mark: 100% from TPs.

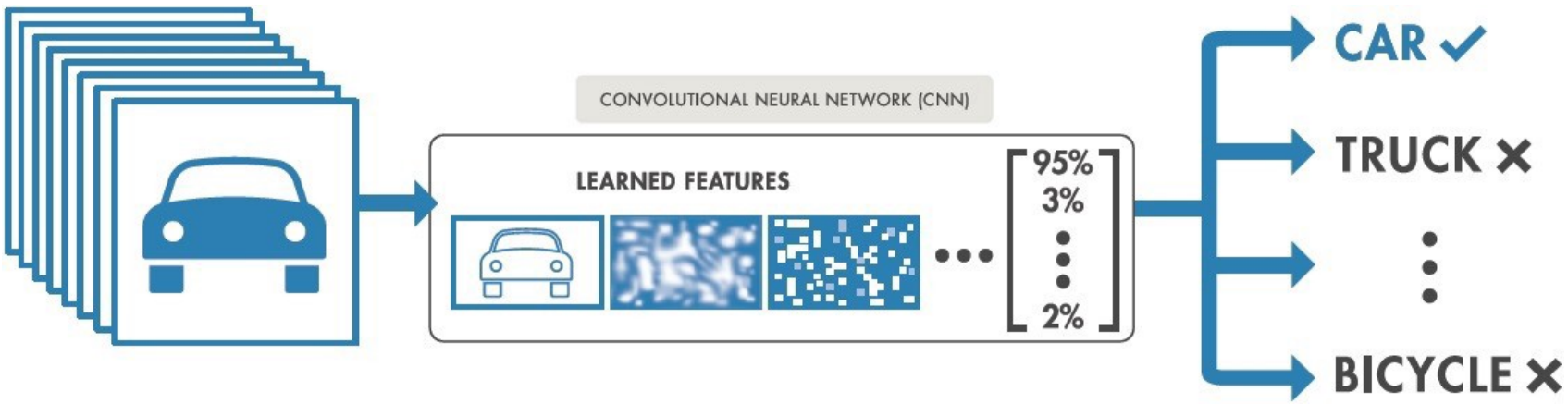
Success of Deep Learning

- Image
- Sound
- Text
- Image to text, text to image, sound to text, ...

  State of AI annual report (206 pages) [link]

- Finance?

DL: image classification



[source]

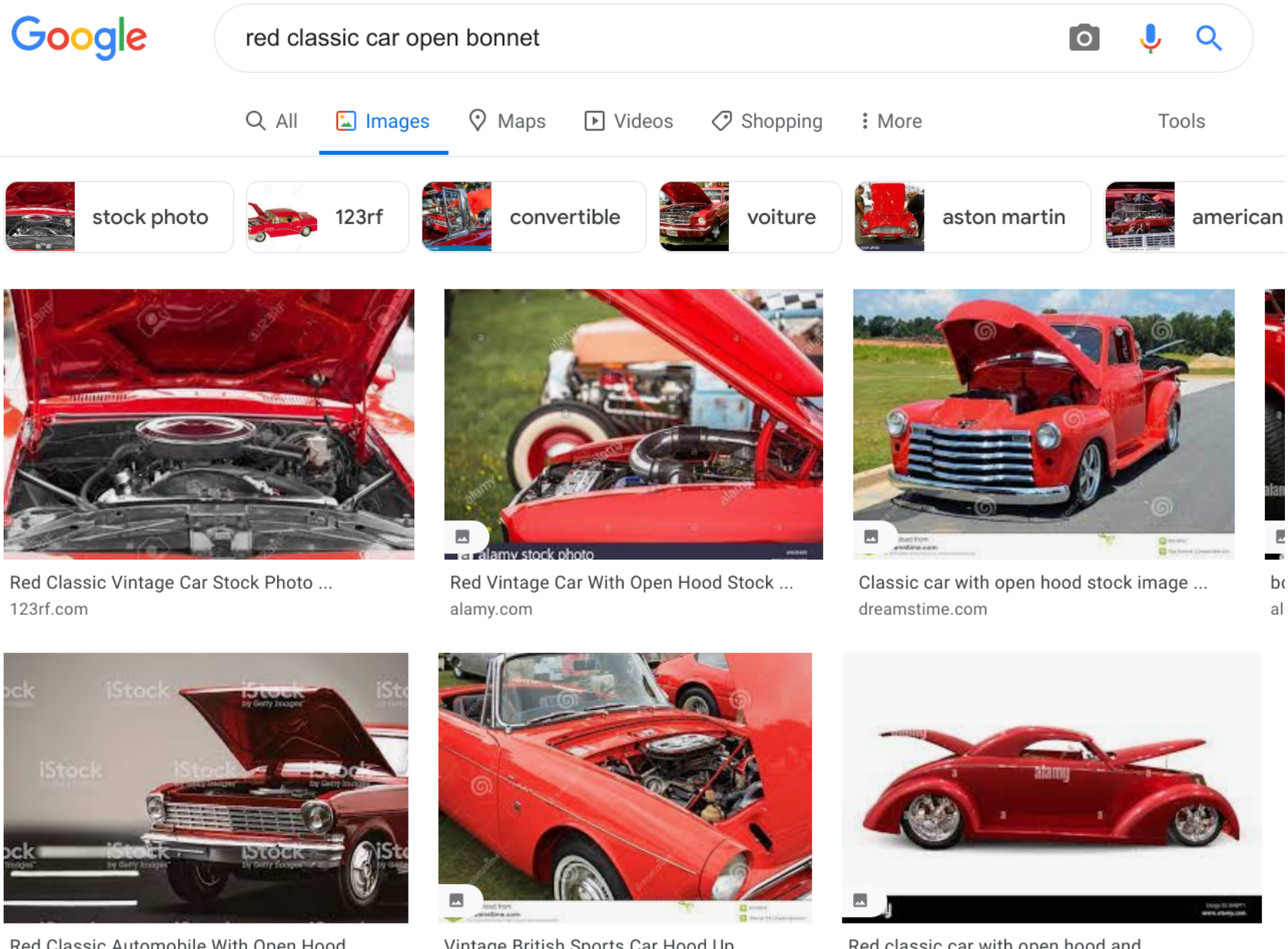
DL: image generation

[source], see also [this person does not exist]

DL: image generation

[source]

DL: text to image (2021)



DL: text to image (2024)

Stable diffusion (automatic111): ‘Beach, morning, seagulls, pizza’

The art of the prompt

glowing galaxy cracks, A cute, glowing version of SpongeBob SquarePants, designed with irresistible, large eyes. His body appears to be cracking like a molten volcano, with cracks glowing in bright, galaxy-like colors—shades of deep purple, blue, and bright orange. The glow emanates from within the cracks, giving an ethereal and mesmerizing effect, as if SpongeBob is infused with cosmic energy. The background is dark, making the glow and molten effect stand out, and small, floating star-like particles surround him, enhancing the galaxy and volcanic theme. The scene blends cuteness with a cosmic, otherworldly vibe

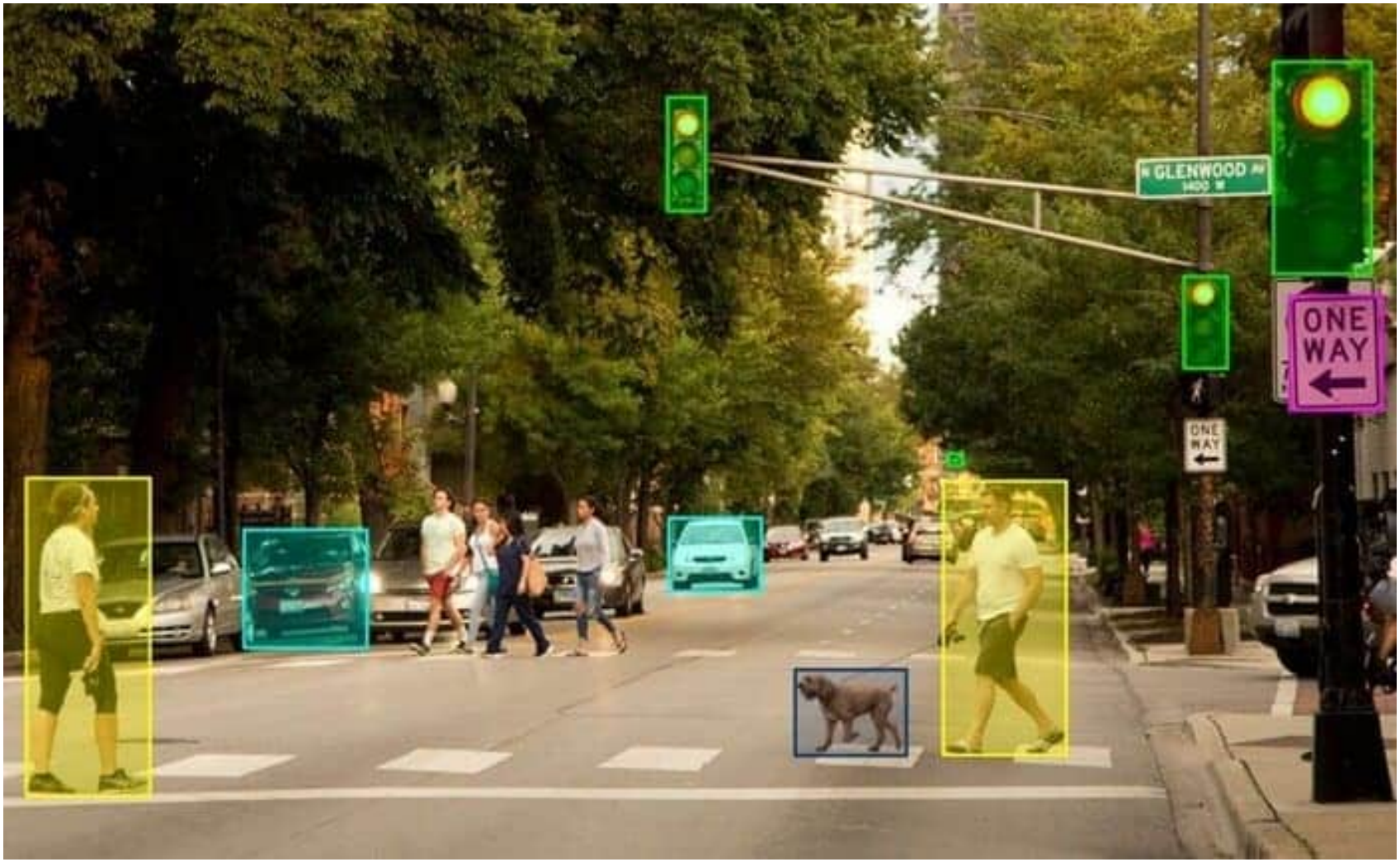
The art of the prompt

[civitai.com]

and of the model

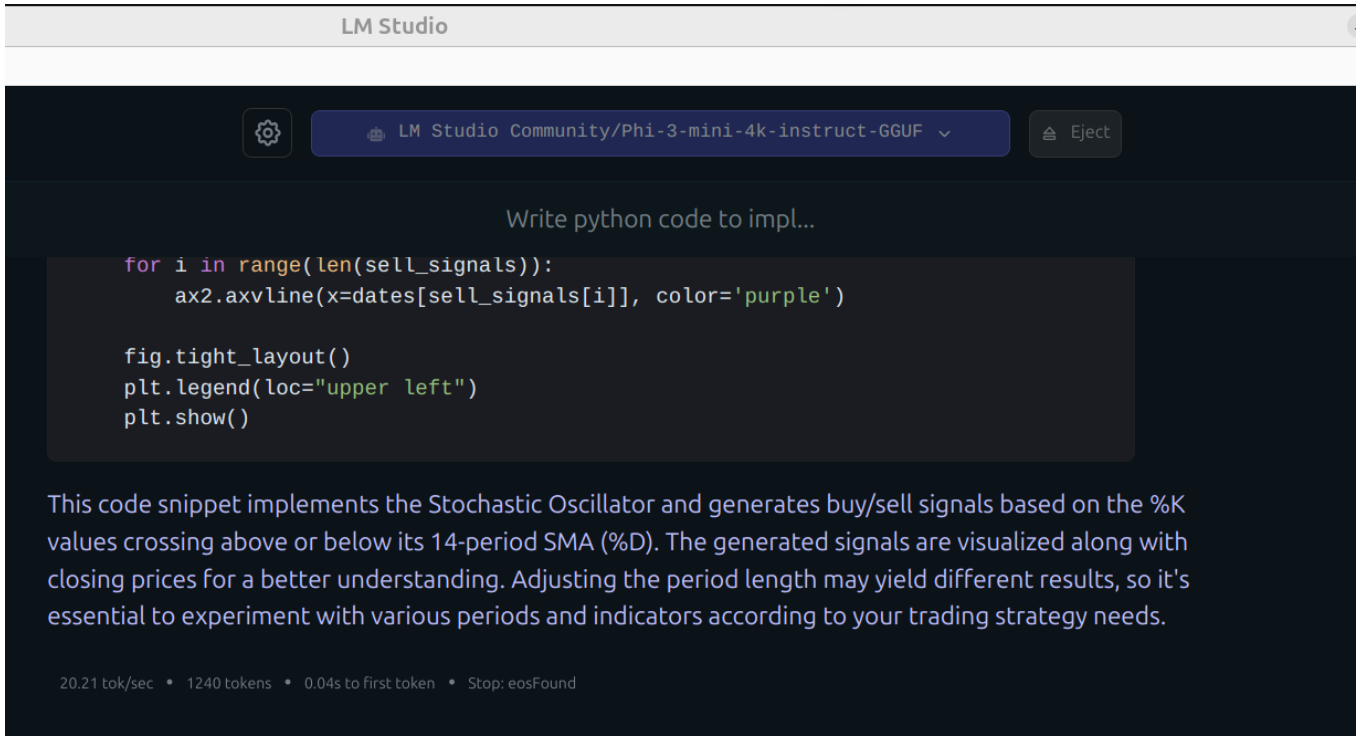
Image annotation

[source]



LLMs on your laptop / (mobile phone)

LMstudio [link]

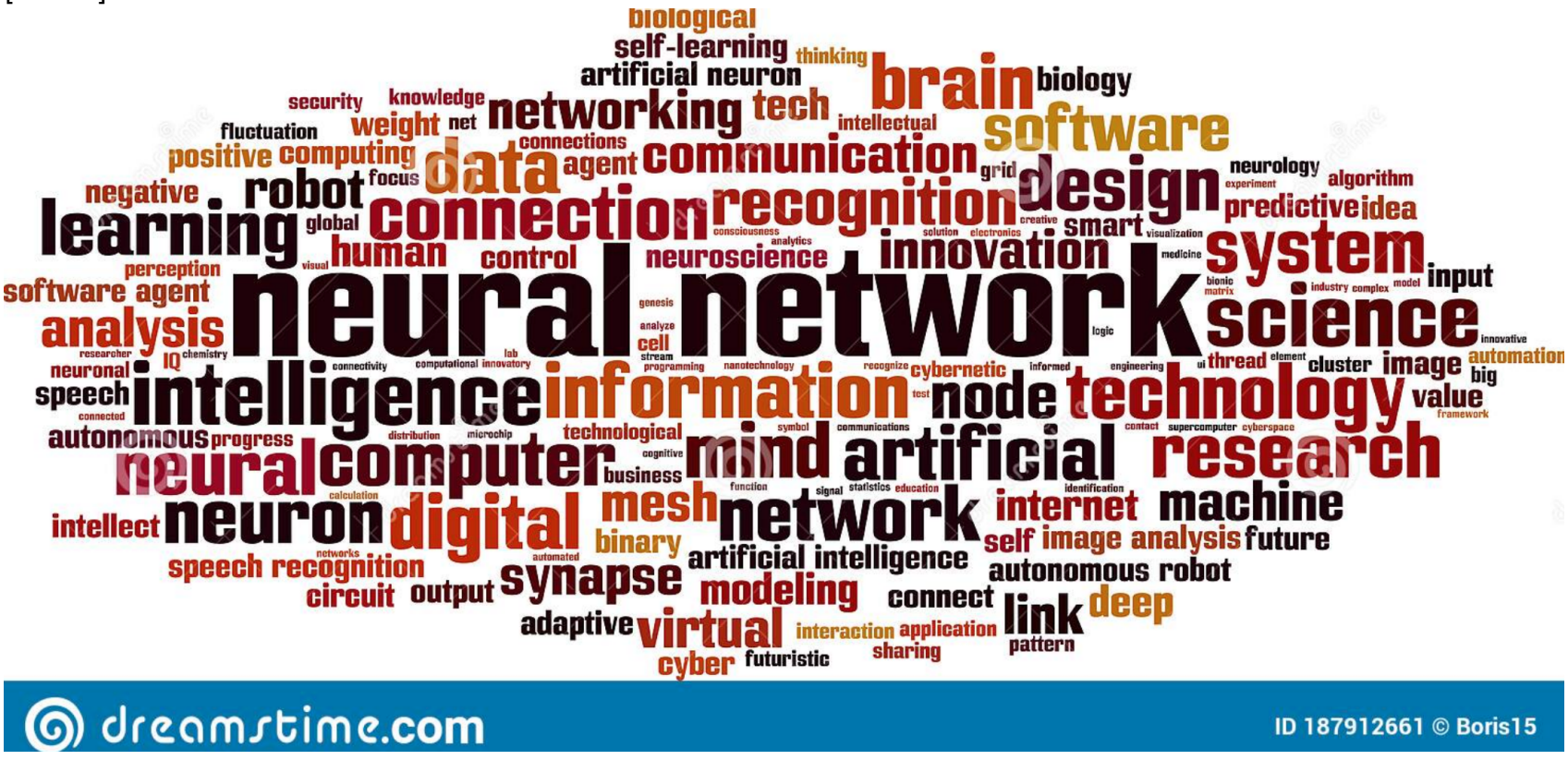


Course plan

1. Estimation/calibration	MLP/CNN
2. Option pricing	model vs DL
3. Market state classification	auto-encoders
4. Timeseries generation	GANs
5. Timeseries generation 2	GANs
6. Prediction	RNN
7. Prediction 2	RNN/CNN/VAE/MLP/LTR
8. Oral feedback on TPs	

Deep learning from the point of view of finance

[source]



YAMLP

Yet Another ML Project

Finance and Machine Learning

- Traditional finance is very linear:

$$\text{CAPM: } \hat{r}_{i,t} = \alpha_i + \beta_i r_{market,t} + \eta_{i,t}$$

$$\text{Factors: } \hat{r}_{i,t} = \alpha_i + \sum_k w_{i,k} f_{i,k,t} + \eta_{i,t}$$

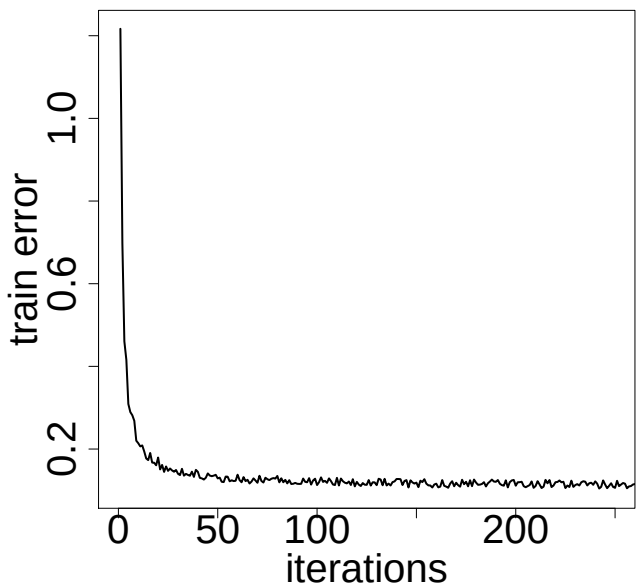
- ML: adds non-linearity
  - Support Vector Machines
  - Genetic programming
  - Random Forests
  - XGboost, LightGBM, ...
  - **DL**

YAMLP: minimal approach to success in ML

1. Split data in train / test sets (80/20)

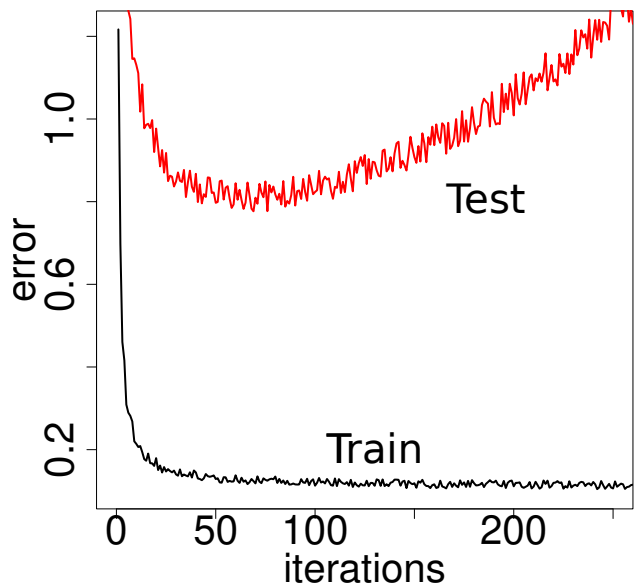
2. Train ML model on train set until error stops decreasing

3. Test model on test set
   (~fine for large stationary synthetic data)



YAMLP: over-fitting

1. Split data in train / test sets (80/20)
2. Train ML model until test error stops decreasing (early stopping)
3. Report test error



YAMLP: a more subtle approach to success in ML

1. Split data in train / validation / test sets (60/20/20)
2. Train ML model until *validation* error stops decreasing
3. Test model

YAMLP: rescaling

Types of rescaling:

- MinMax rescaler

$$\tilde{x}_i = \frac{x_i - \min_k x_k}{\max_j x_j - \min_k x_k} \in [0, 1]$$

- Z-score

$$\tilde{x}_i = \frac{x_i - \text{mean}(\boldsymbol{x})}{\text{std}(\boldsymbol{x})}$$

YAMLP successful: sharp stationary truth

X input, y output

$$y = F(X)$$

1. Sharp X and y
2. Stationary world: F constant

Finance is NOT YAMLP (Yet Another ML Project)

Specific problems

- Heavy tailed data — rescaling?
- Non-stationarity — distribution shift
- Long memory — ??
- Good-enough price predictions are *really hard* [link]

Problem with rescaling: heavy-tailed data

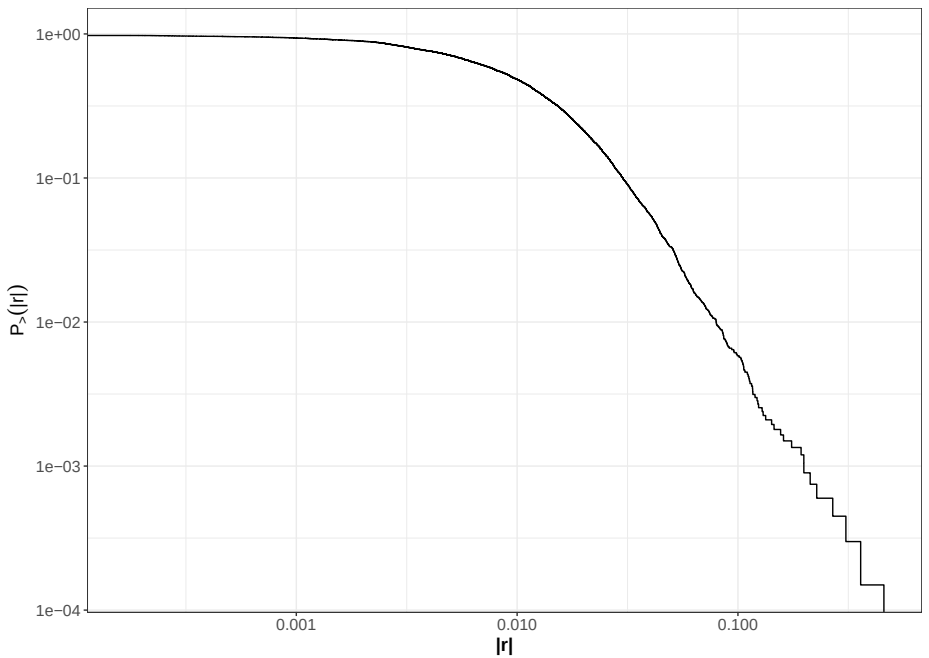
- Price return

$$R_t = p_t / p_{t-1} - 1$$

- Price log return

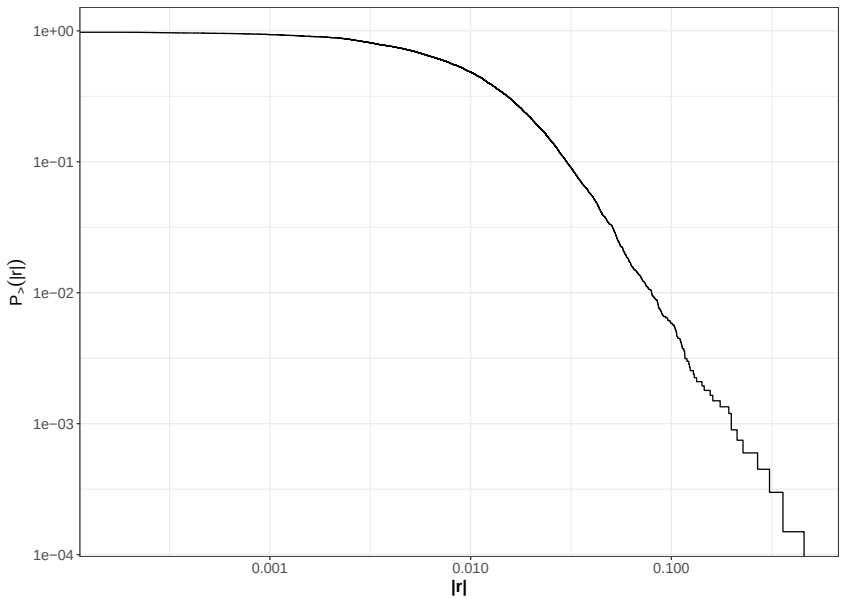
$$r_t = \log p_t - \log p_{t-1} = \log(1 + R_t)$$

- Heavy tail: $P(|\text{return}| > r) \propto |r|^{-3}$



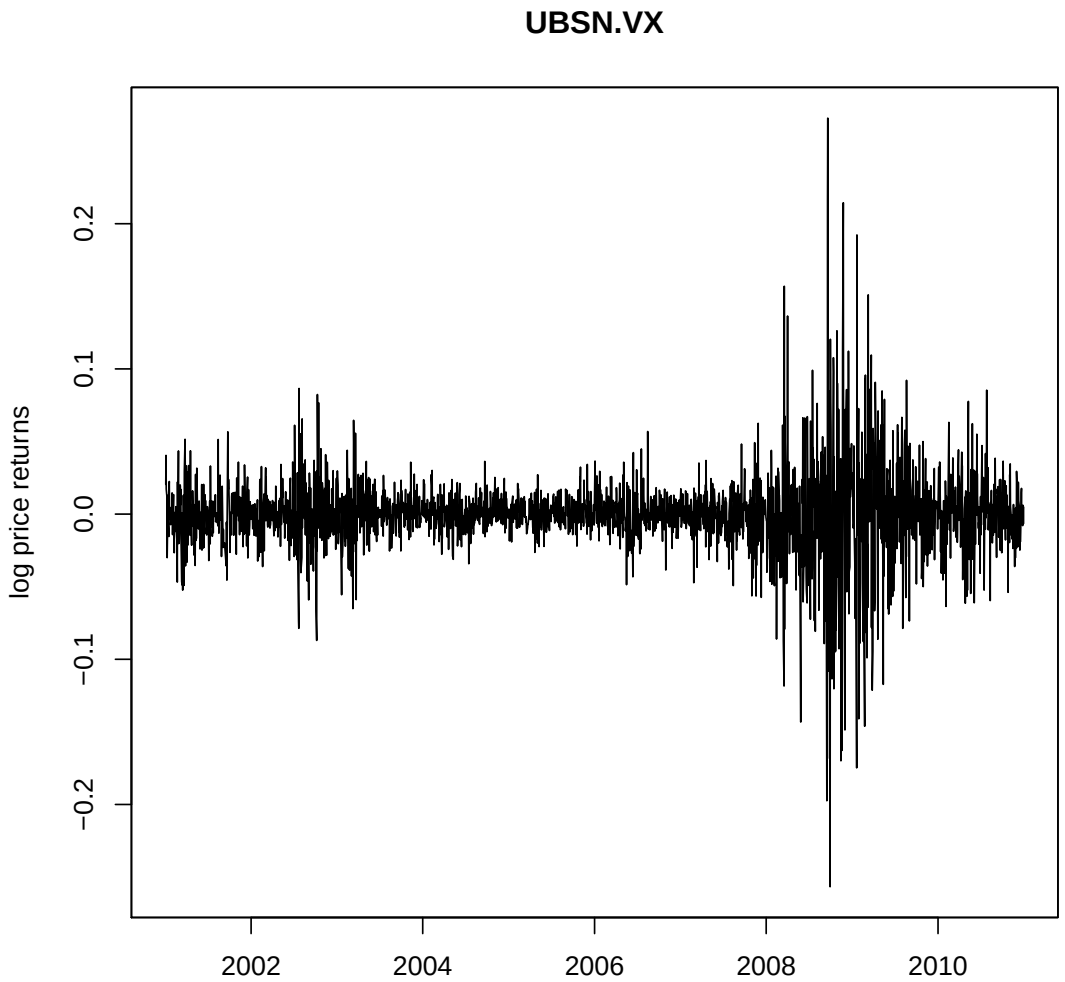
Problem with rescaling: heavy-tailed data

- MinMax:
  - unbounded values
  - max and min are not robust
- Z-score:
  - Usually $E(|r|^3) \to \infty$
  - StdDev estimator is too noisy
  - Average estimator is too noisy

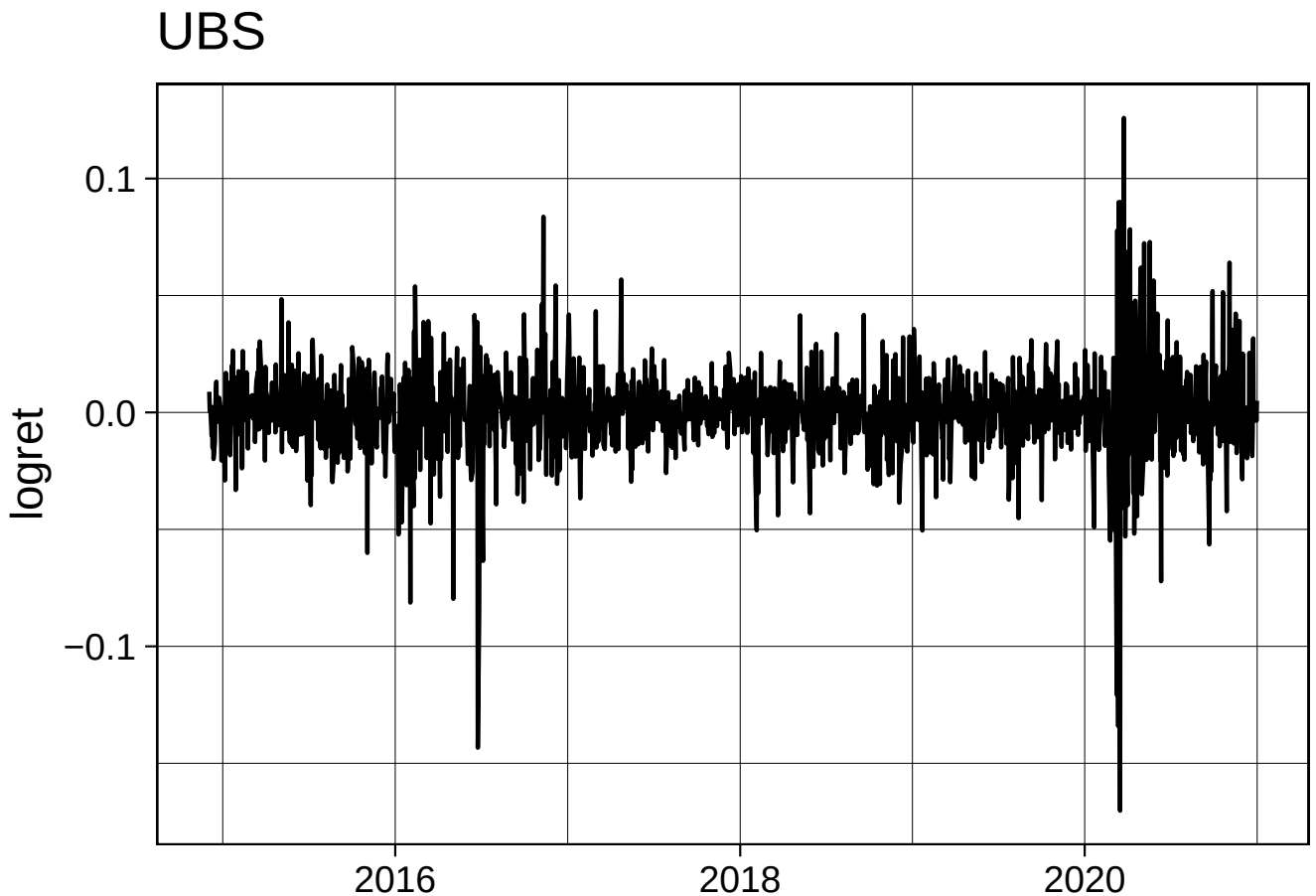


$\to$ naive input rescaling may not be a wise strategy

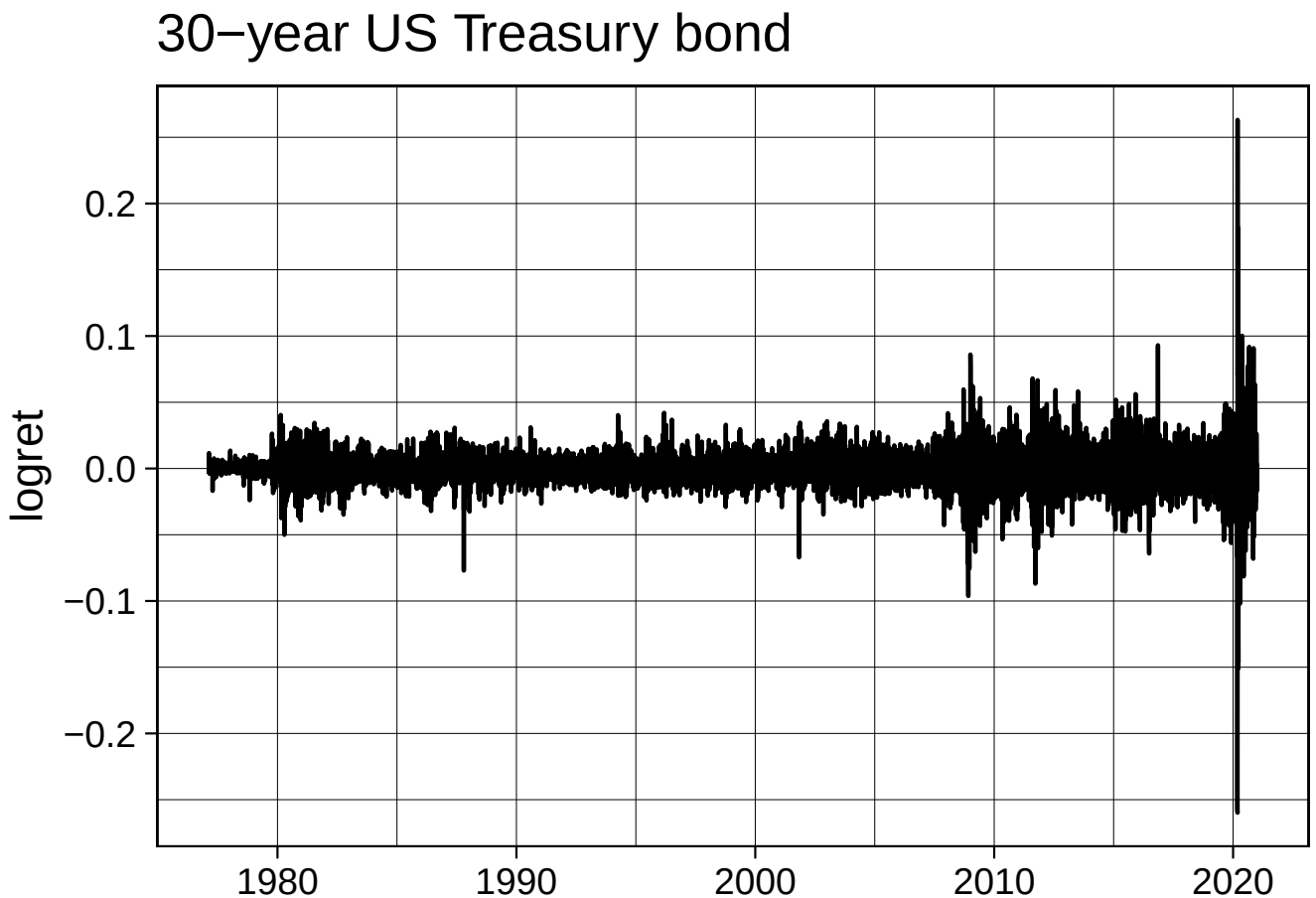
Problem with rescaling: non-stationarity



Non-stationarity is non-stationary

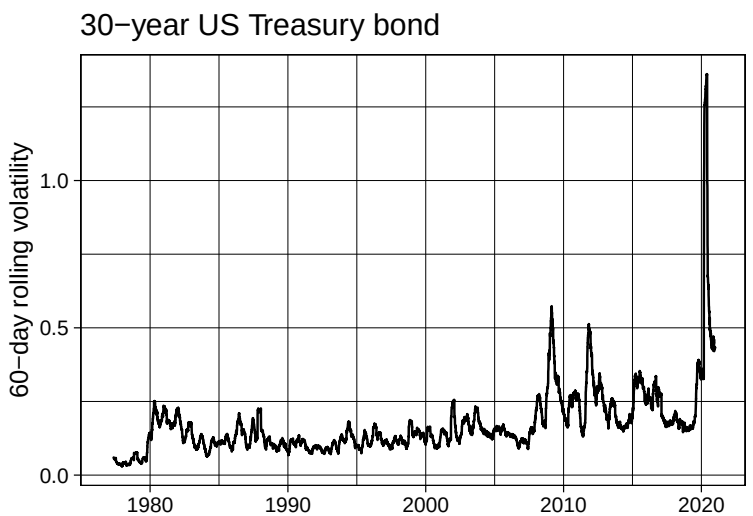


All asset types are non-stationary and heavy-tailed



Problem with full data rescaling: information leakage

- Many ML blogs make this mistake
- Rescaling must be done within each set, not globally
- ANY information from the future makes prediction significantly biased.
- Accounting for future new scale maxima uses the future



Problem with single train window: non-stationarity

- Many papers/blogs only train once
- Dependence structure of financial data is non-stationary
  $\rightarrow$ train/validation/test once is not enough
  $\rightarrow$ test length = real-life trading: as short as possible.
- Problem: DL may be prohibitively expensive
- Solutions:
  - stationarize ML model: very long time series
  - warm start
  - predict distribution shifts

Problem with simple NNs: time-series, long memory

- Problem: Many financial quantities have a long memory
  - volatility
  - volume
  - order flow (e.g. market order sign)
- Solution: use adequate model / NN architecture
  - Temporal Convolution Nets (TCNs) (Wavenets)
  - LSTM / GRU
  - State-space models (Mamba)

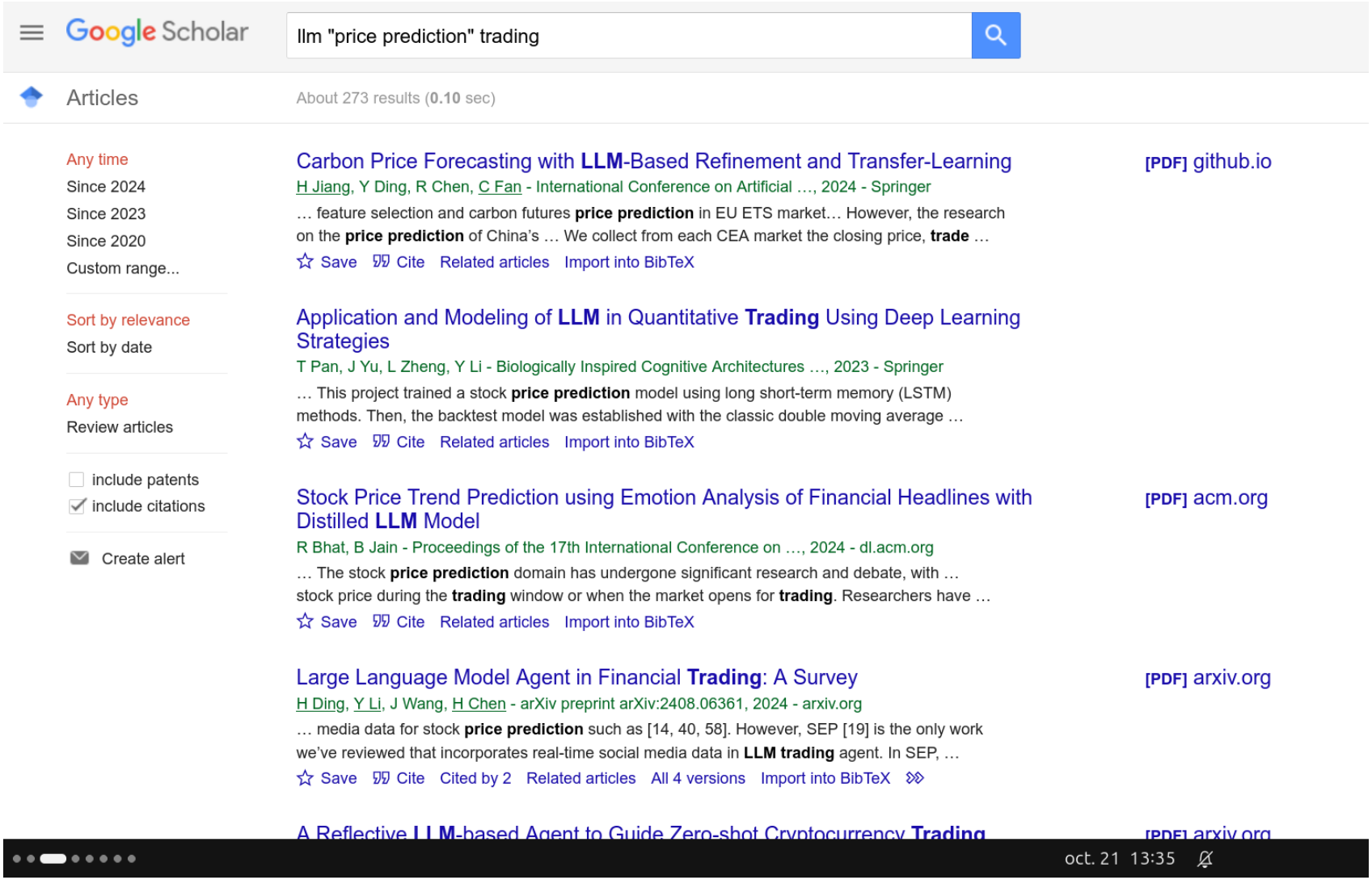
Problem with predictions:

- Many quantities are hard to predict.
- Many ML blogs/articles find $\hat{p}_{t+1} = p_t \iff E(r_t) = 0$: no prediction power
- Any small information leakage from the future changes much the results.
  - scale from the future
  - train/test split not causal
  - over-optimizing ML (data snooping)

Problem with predictions: time series are VERY noisy

- Noise filtering is essential before using a matrix of price returns (e.g.)
- Real information content of a single price return is very small: 1-3 bits, not 64.
- *Purgamentum init, purgamentum exit*

Problem with predictions: literature very noisy



DL for hard problems in Finance

• Exact nonlinear results are rare	approx. solution
• Monte-Carlo simulations are expensive	approx. solution
• Calibration is hard and expensive	DL surrogates
• Prediction is harder	hidden relationships
• Many equations are impossible to solve	PINNs

Artificial Neural Networks: a short introduction

Artificial Neural Networks: a short introduction

- NN= collection of *connected* neurons

- Artificial neuron:
  weighted sum of inputs + non-linear *activation*

$$O = \phi\left(\sum_{i=1}^{N} w_i x_i\right)$$

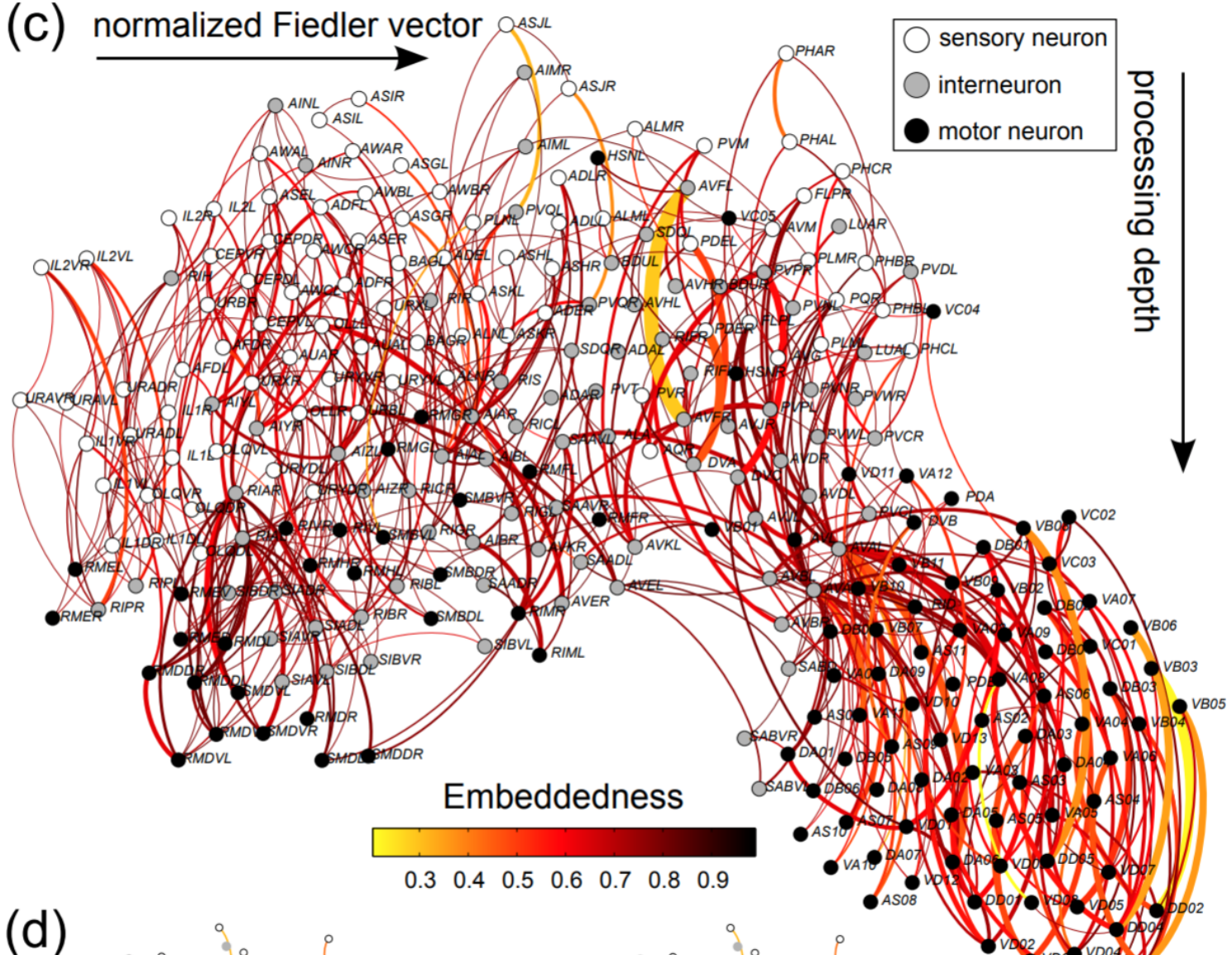
ϕ : non-linear function

- What happens if many non-linear functions interact?

  $\rightarrow$ complex system (simple elements, complex behaviour)
  $\rightarrow$ even more complex if time-dependent with feed-back
  $\rightarrow$ even more complex is connectivity is complex

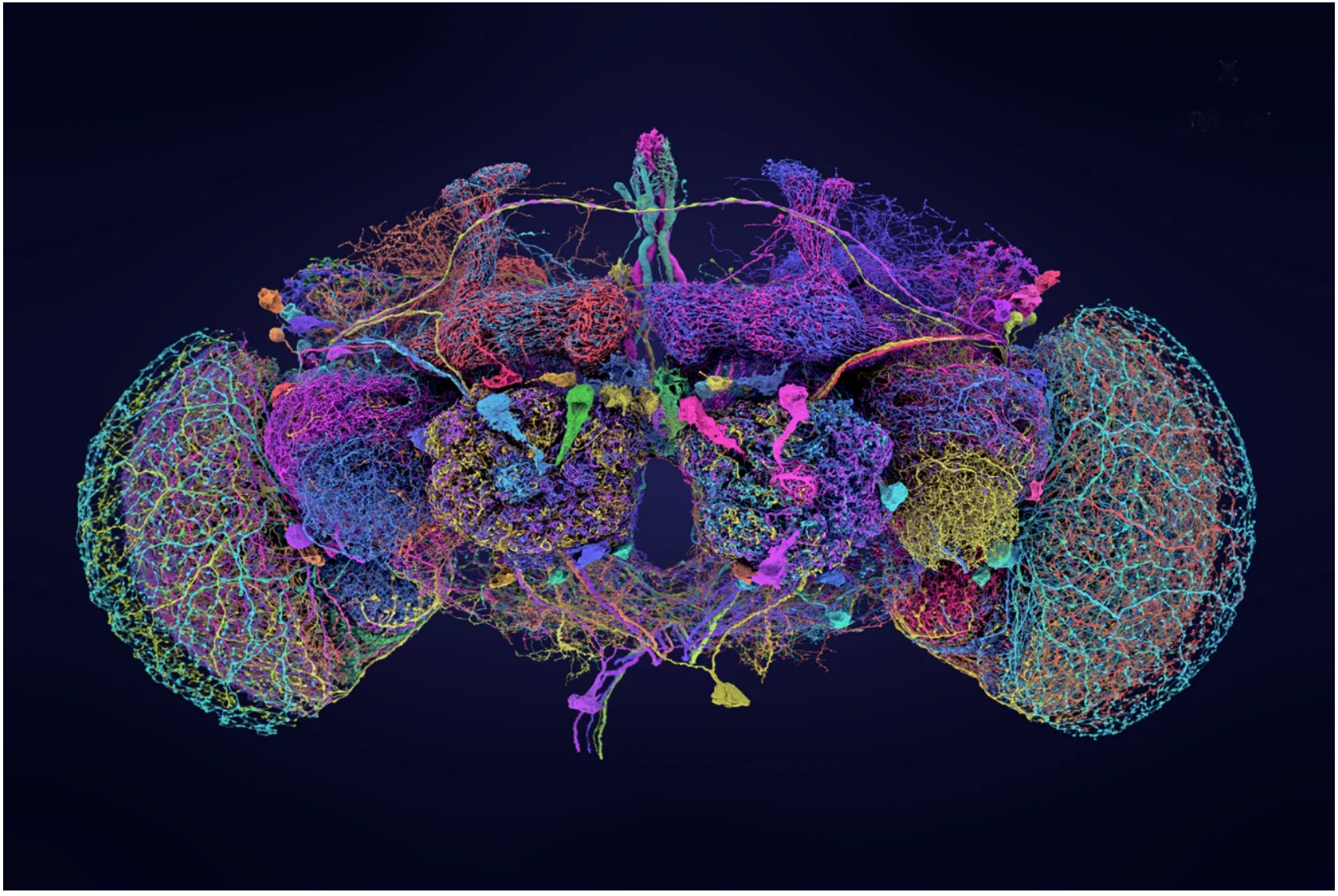
Real-life connectivity

C. Elegans [Schaub et al. (2013)]



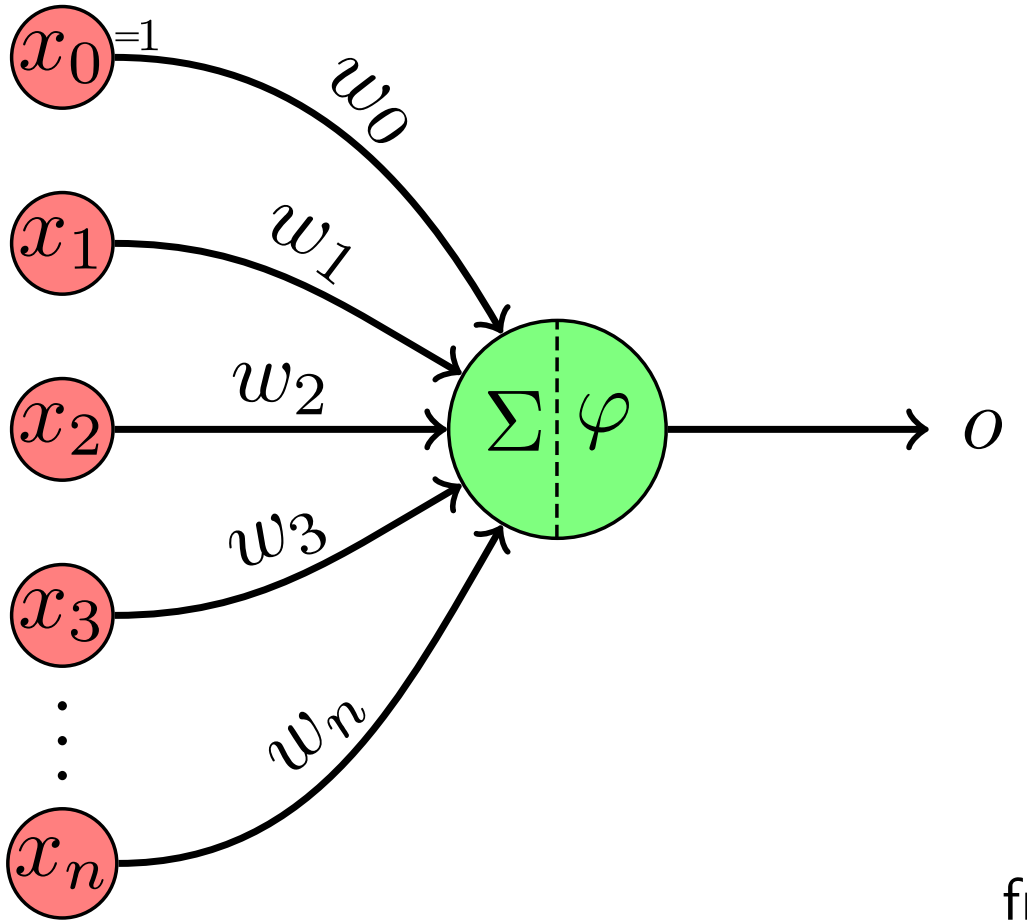
Fruit fly (2024)

Dorkenwald *et al.* (2024)[link]: 139,255 neurons



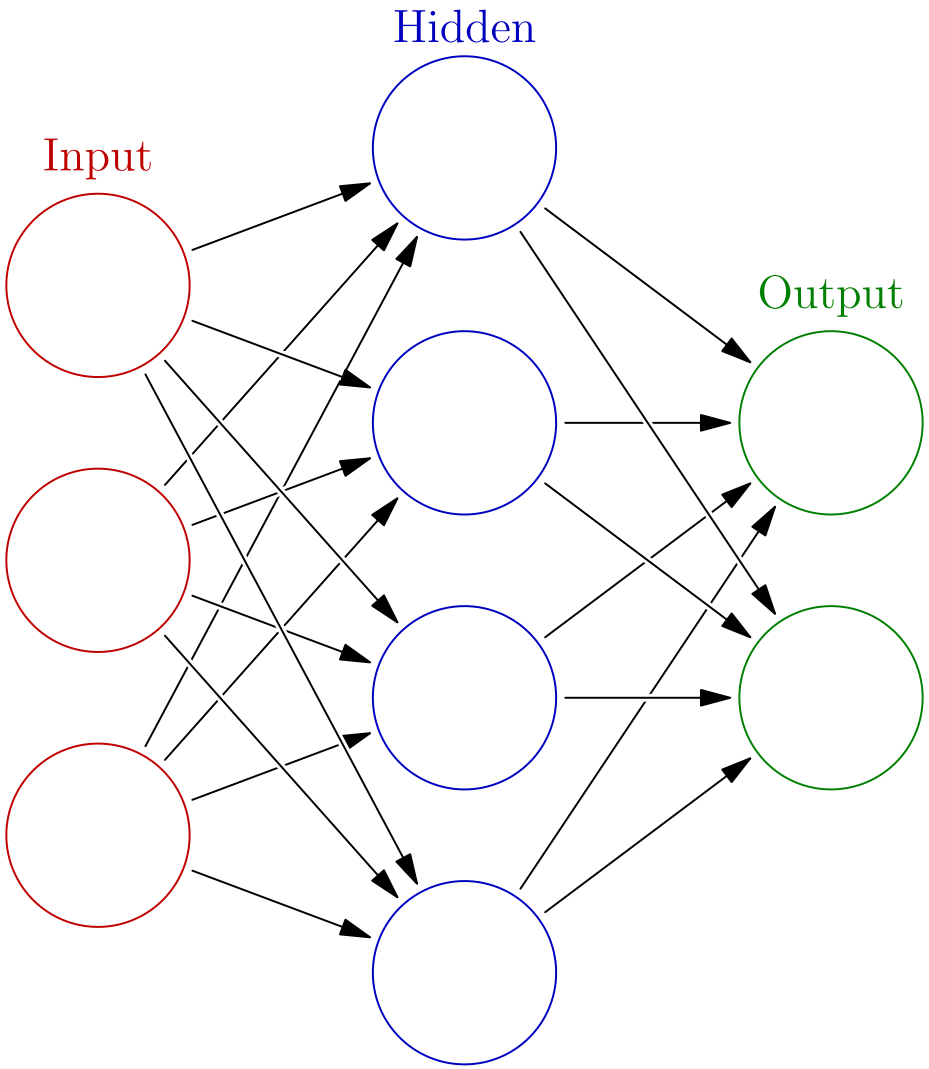
ANNs: simplest perceptron

Single neuron

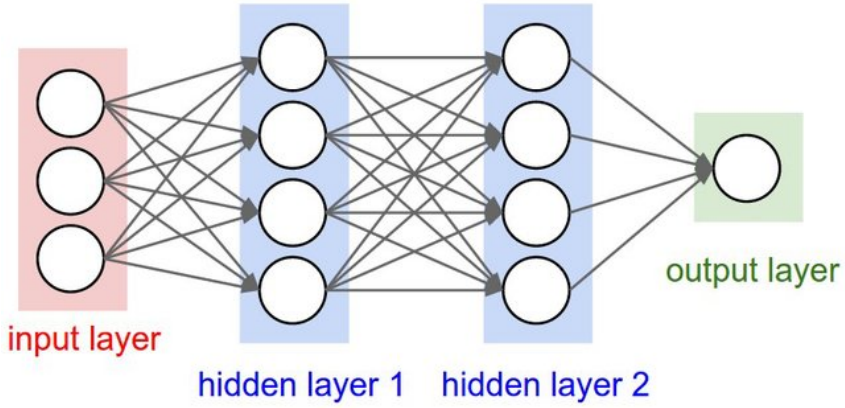


from [link]

Single-hidden layer ANN



Why Multilayer ANNs (a.k.a. Deep)?



from [link]

- Neuron budget: N
- N neurons in a single layer
- K hidden layers with N/K neurons

  $\rightarrow$ complexity $O\left[\left(\frac{N}{K}\right)^K\right] > O(N)$

Biological NNs

More complex architecture

- Shortcuts
- Recurrent
- Stateful neurons
- Time dimension
  - synchronisation?
  - functional programming
- Principle of least action (not least loss)

Deep Neural Network: definition

at least 1 hidden layer + something to optimize

What to optimize?

1. Least-squares
2. Non-linear fit
3. Estimation
4. Distance to complex equation solution
5. Risk minimization
6. Trading performance
7. Terminal wealth / integrated cost
8. Look-alike-ness time series
9. Minimise deviation from constraints

Some notations

- N samples of P features

$$X_{n,p}, \quad n = 1, \cdots, N \quad p = 1, \cdots, P$$

- Output Y of size K

$$Y_n, \; n = 1, \cdots, N$$

$$(X_{n,1} \cdots X_{n,P}) \sim (Y_{n,1} \cdots Y_{n,K})$$

- ML: match lines of X with lines of Y

$$X \sim Y$$

Tasks

- Regression:

$$Y_n = F(X_n) \in \mathbb{R}^K$$

- Classification: S classes

$$Y_n = F(X_n) \in \{1, \cdots, S\}^K$$

- Discrimination

$$F(X_n) \neq F(X_m), \quad n \neq m$$

- Ranking

$$F(X_n) > F(X_m), \quad n \neq M$$

DL $\equiv$ error minimisation

Regression

$$\mathcal{L} = \sum_i \Phi\left[Y_i - F(X_i)\right]$$

- MSE ($\rightarrow$ average)

$$\mathcal{L} = \sum_i \left[Y_i - F(X_i)\right]^2$$

- MAE ($\rightarrow$ median)

$$\mathcal{L} = \sum_i \left|Y_i - F(X_i)\right|$$

- Robust loss (Huber)

$$\Phi(x) = \begin{cases} x & |x| < c \\ K & |x| \geq c \end{cases}$$

How to optimise something with an ANN?

Backpropagation: only weights matters

Nice write-up [link]

- Feed forward:

$$\text{Input} \rightarrow \text{ ANN} \rightarrow \text{output}$$

- error:

$$\mathcal{L} = E(|\text{output - expected output}|\,|\boldsymbol{w})$$

- backpropagation: weight update, layer i

$$w_i' = w_i - \eta \frac{\partial \mathcal{L}}{\partial w_i}$$

η : learning rate

- computing $\frac{\partial \mathcal{L}}{\partial w_i} = \prod_{j \geq i} \frac{\partial \mathcal{L}}{\partial w_j}$: chain rule

Do biological networks backpropagate?

Alternative: predictive configuration/coding [link]

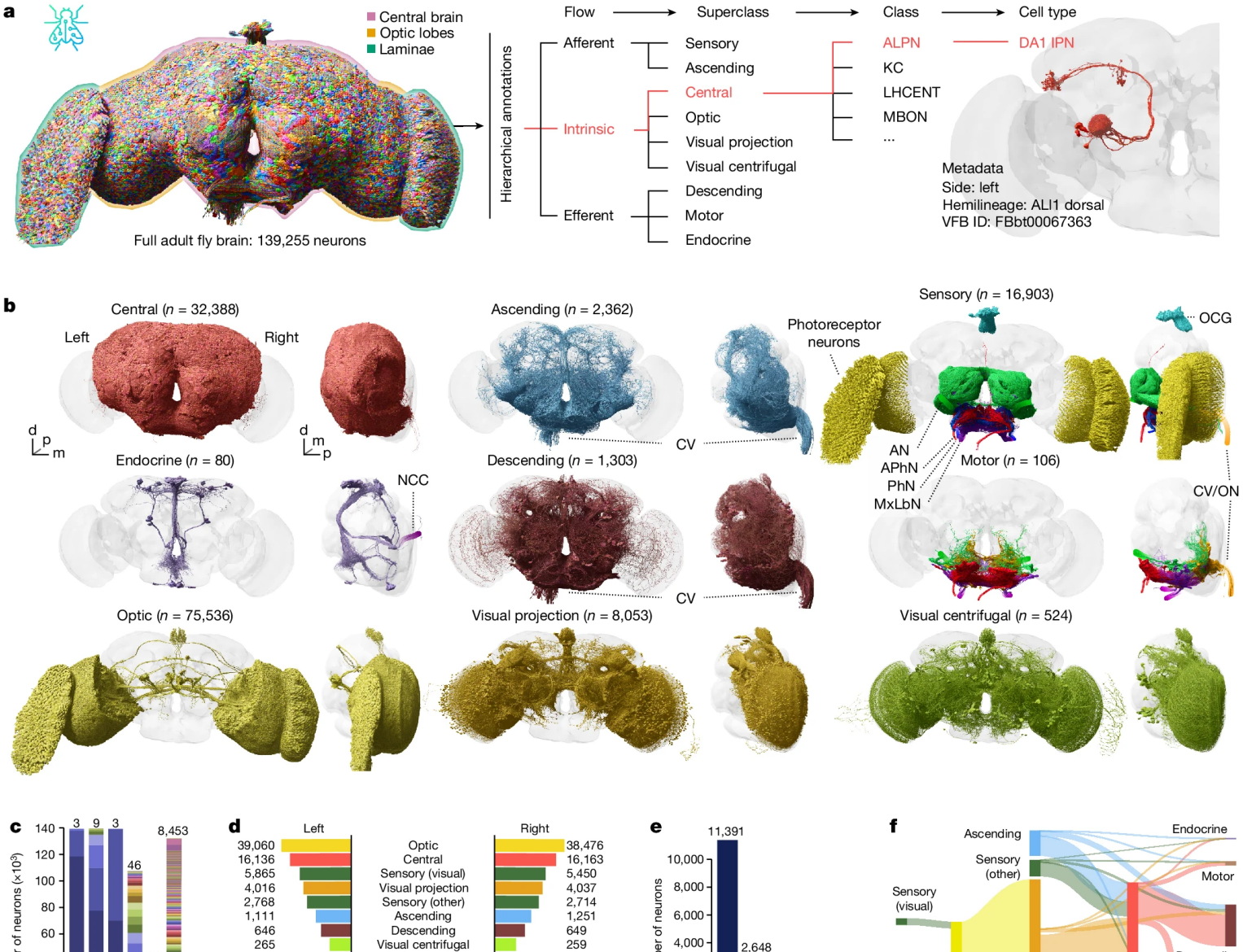
- Activation strength matter
- Find better weights that yield better activation strengths

1. $\rightarrow$ update neuron activation: control variables
2. $\rightarrow$ update weights: parameters

Song et al. (2023) [link]

Backward-forward propagation

Fruit fly: Schlegel et al (2024) [link]



Side note: alternative ways of learning

Bak and Chialvo: learning from mistakes (2001) [link]

- Classification
- Activation: neuron with max w_i is activated
- Learning only from mistakes
  - reduce weights of activated neurons
  - quick re-learning

DL success: learn representations

Convolutional networks [source]

- The deeper, the more and more complex the features
- Interactive visualization [link]

DL programming: crash course

1. No installation: use Google Colab [link]
   free **fast** CPU / GPU / TPU
   (runtime->GPU)

2. Nvidia: use a fully set up Docker container from Nvidia
   AMD and Mac: install tensorflow

3. Install Tensorflow 2.x
   `conda install -c anaconda tensorflow`

   preferably in a virtual environment (`conda create -n DLF`)

My first ANN: single-layer NN

Input: $P = 8$, output $K = 1$

```
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

model = Sequential()

model.add(Dense(12, input_dim=8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

model.compile(loss='mse', optimizer='adam')
```

Multi-layer NN

```
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

model = Sequential()

model.add(Dense(12, input_dim=8, activation='relu'))

model.add(Dense(12, activation='relu'))
model.add(Dense(12, activation='relu'))
model.add(Dense(12, activation='relu'))
model.add(Dense(12, activation='relu'))
model.add(Dense(12, activation='relu'))
model.add(Dense(12, activation='relu'))
model.add(Dense(12, activation='relu'))
model.add(Dense(12, activation='relu'))
model.add(Dense(12, activation='relu'))
model.add(Dense(12, activation='relu'))
model.add(Dense(12, activation='relu'))
model.add(Dense(12, activation='relu'))
model.add(Dense(12, activation='relu'))

model.add(Dense(1, activation='linear'))


model.compile(loss='mse', optimizer='adam')
```

Activation functions: domain of Y!

1. Bounded
   - Heaviside (pre-2000)
   $$\theta(x) \in [0, 1]$$
   - sigmoid
   $$\frac{1 + \tanh(x)}{2} \in [0, 1]$$
2. Unbounded
   - ReLU
   $$\max(0, x) = (x > 0) \cdot x \in [0, \infty]$$
   - LeakyReLU (α may be learned $\rightarrow$ PReLU)
   $$\max(0, x) + (x < 0) \cdot (\alpha x) \in [-\infty, +\infty]$$
   - SELU
   $$\lambda x \cdot (x < 0) + \lambda\alpha(e^x - 1) \cdot (x \geq 0) \in [-1.75, +\infty]$$
   λ, α known
   - Swish [link]
   $$x\text{sigmoid}(x)$$

How to train and predict a NN with Keras

```
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense


P=20
K=1

model = Sequential()
model.add(Dense(12, input_dim=P activation='relu'))
model.add(Dense(12, activation='relu'))
model.add(Dense(12, activation='relu'))
model.add(Dense(K, activation='linear'))


model.compile(loss='mse', optimizer='adam')


# here, generate X, an N*P numpy array
#             and Y, the right answer
# X=...
# Y=...

model.fit(X,Y)

# X_test is a new input (numpy array of length N'*P)
# X_test=...

model.predict(X_test)
```

Training in practice

- $N \gg 1$ samples: computing loss is expensive
- Batch size $B < N$: number of samples

$$\mathcal{L}(X) \simeq \mathcal{L}(X_{batch})$$

  - `for each batch`
    `compute loss`
    `backpropagation #stochastic gradients`

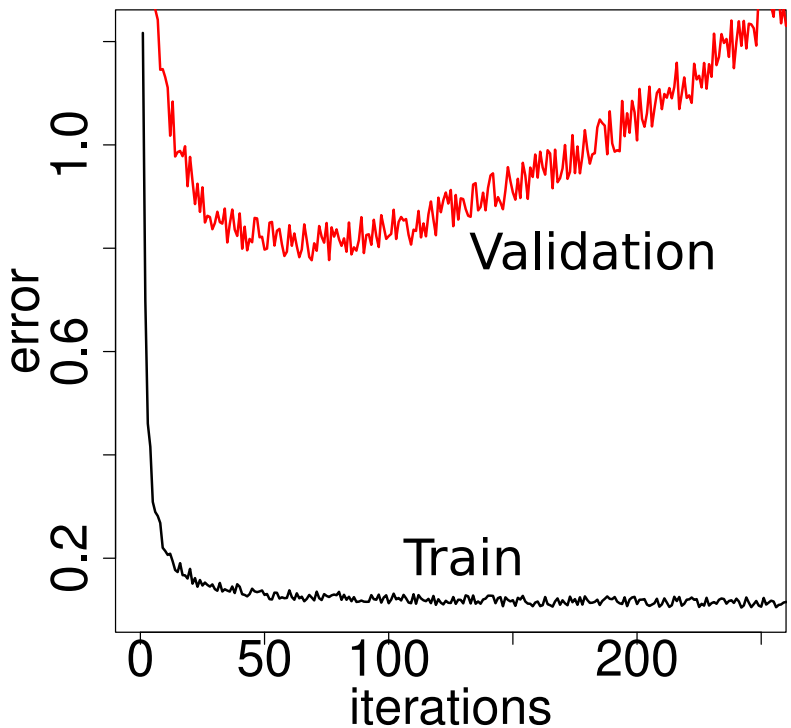
- Number of times whole data set is used: epochs
  - `for each epoch`
    `for each batch`
    `compute loss`
    `backpropagation #stochastic gradients`

How to train and predict a NN with Keras

```
from tensorflow import keras
from keras.models import Sequential
from keras.layers import Dense

P=20
K=1

model = Sequential()
model.add(Dense(12, input_dim=P activation='relu'))
model.add(Dense(12, activation='relu'))
model.add(Dense(12, activation='relu'))
model.add(Dense(K, activation='linear'))


model.compile(loss='mse', optimizer='adam')


# here, generate X, an N*P numpy array
#              and Y, the right answer


model.fit(X,Y,batch_size=32,epochs=10)

# X_test is a new input (numpy array of length N'*P)
# X_test=...

model.predict(X_test)
```

How many epochs? batch size?

1. Split train data into train + validation
2. Early stopping when error on validation stop decreasing



Model calibration: why ML?

- ML requires a stochastic model / candidate distribution
- Some parameters may be hard to estimate with ML
- Estimation is more generic
- Best estimator usually unknown

Model calibration

- Stochastic model $\mathcal{M}(\theta)$, $\theta \in \mathbb{R}^K$

$$\text{noise} \rightarrow \text{time series } x = \{x_1, \cdots, x_T\}$$

- Calibration:

$$\hat{\theta}(x) = \hat{M^{-1}}(x)$$

ML/NN predicts θ from model output $\equiv \mathcal{M}^{-1}(X)$

Model calibration: recipe

- choose J values of θ_i (regularly spread)
- If stochastic model
  - for each θ_i, generate M outputs

$$X^{(\theta_i)} = \begin{pmatrix} X_{1,1}^{(\theta_i)} & \cdots & X_{1,T}^{(\theta_i)} \\ \vdots & \vdots & \vdots \\ X_{M,1}^{(\theta_i)} & & X_{M,T}^{(\theta_i)} \end{pmatrix} \sim \begin{pmatrix} \theta_i \\ \vdots \\ \theta_i \end{pmatrix} = Y^{(\theta_i)}$$

  - stack the matrices

$$X = \begin{pmatrix} X^{(\theta_1)} \\ \vdots \\ X^{(\theta_J)} \end{pmatrix} \in \mathbb{R}^{MJ \times T} \qquad Y = \begin{pmatrix} Y^{(\theta_1)} \\ \vdots \\ Y^{(\theta_K)} \end{pmatrix} \in \mathbb{R}^{MJ \times K}$$

- Else $M = 1$

Reference papers in finance

[link]

Model Calibration with Neural Networks

Andres Hernandez
IBM Risk Analytics
andres.hernandez@gmx.net

July 20, 2016

- Quantity $Q = \mathcal{M}(\theta_{\text{model}}, \xi_{external})$
- Data: $\{Q_{\text{market}}, \xi_{\text{external}}\}$
- Learn

$$\theta^* = \arg\min_{\theta} \text{Cost}(\theta, \{Q_{\text{market}}, \xi_{\text{external}}\}) = \text{fct}(\{Q_{\text{market}}, \xi_{\text{external}}\})$$

Reference papers in finance

[link]

Vol. 21, No. 1, 11–27, https://doi.org/10.1080/14697688.2020.1817974

Check for updates

Deep learning volatility: a deep neural network perspective on pricing and calibration in (rough) volatility models

BLANKA HORVATH*†, AITOR MUGURUZA‡ and MEHDI TOMAS§

†King's College London, The Alan Turing Institute London, UK
‡Imperial College London & NATIXIS London, UK
§CMAP & LadHyx, École Polytechnique, Palaiseau, France

When model $\to$ Q is expensive (e.g. volatility map)

1. train NN_1 $Q[\mathcal{M}(\theta, \xi_{external})] \to \tilde{Q}(\theta, \xi_{external})$ (surrogate)
2. train NN_2 to solve calibration

Case study: Hurst exponent H

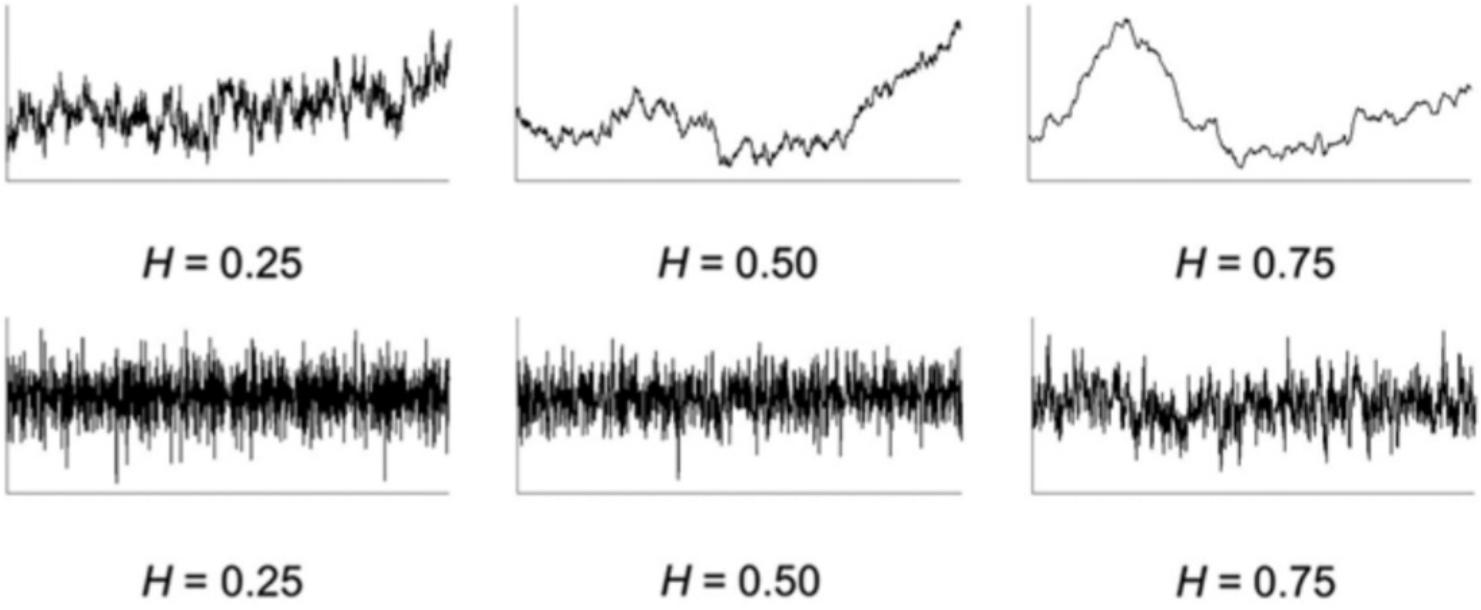


FIGURE 1 | Top row: example series of fractional Brownian motions (fBm) for three typical values of the scaling exponent. The central graph represents an ordinary Brownian motion ($H = 0.5$). The left graph shows an anti-persistent fBm ($H = 0.25$) and the right graph a persistent fBm ($H = 0.75$). The corresponding fractional Gaussian noises series (fGn) are displayed in the bottom row.

Marmelat et al. (2012) [link]

Standard diffusion

- $X_t = X_{t-1} + \epsilon_t$
- If ϵ_t i.i.d.,

$$E\left[(X_t - X_0)^2\right] = cst \times t = cst \times t^{2H}$$

- In physicist's words: envelope growth

$$|X_t - X_0| \sim \sqrt{t} = t^H$$

H : Hurst exponent

- Diffusion / Wiener process:

$$H = 1/2$$

How to generate anomalous diffusion

fractional Brownian motion (fBm)
(Mandelbrot and Van Ness (1968) [link])

- Notation: position W_t
- $W_t - W_s$ is stationary

$$P(W_t - W_s) = P(W_{t-s})$$

- Hyp: Gaussian increments with

$$E(dW_t) = 0, \quad E(dW_t^2) = 1$$

fMB: properties

- Covariance

$$E(W_t W_s) = \frac{1}{2}\left(|t|^{2H} + |s|^{2H} - |t-s|^{2H}\right) \quad \begin{cases} > 0 & H > 1/2 \\ \min(t,s) & H = 1/2 \\ < 0 & H < 1/2 \end{cases}$$

- Self-affine time

$$t \to \lambda t$$

$$W_{\lambda t} \overset{\text{law}}{=} \lambda^H W_t$$

- Integral representation

$$W_t = W_0 + \frac{1}{\Gamma(H+1/2)}\{\int_{-\infty}^{0}\left[(t-s)^{H-1/2} - (-s)^{H-1/2}\right] dW_s + \int_0^t (t-s)^{H-1/2} dW_s\}$$

Simulation of fBm

1. Straightforward integration from Gaussian variables

2. Given that the covariance between B_s and B_t is known, use Cholesky decomposition.

3. ...

How to measure H? Scale plot

Since

$$E\left[(W_t - W_0)^2\right] \propto t^{2H}$$

Fit

$$\log E\left[(W_t - W_0)^2\right] = a + 2H \log t$$

Rescaled Range method (Hurst)

Range method

- Detrend data

$$X_t = W_t - \hat{\mu}, \quad t = 1, \cdots, T$$

- Compute S_t : standard deviation of X_t
- Compute running minimum and maximum

$$m_t = \min(X_1, \cdots, X_t)$$
$$M_t = \max(X_1, \cdots, X_t)$$

$\rightarrow$NOT robust!

- Range

$$R_t = M_t - m_t$$

- Rescaled range

$$R_t / S_t \simeq H$$

Fourier Method

- Fourier transform: from real signal to frequencies $\rightarrow$ spectrum
- Non-local kernel $e^{i\pi kx}$
- Flandrin (1989) [link]: for fBm, average spectrum

$$S(\omega) \propto \frac{1}{|\omega|^{2H+1}}$$

- Compute spectrum, fit with power-law $\rightarrow H$

Wavelets

- Local kernels ϕ

$$\int \Phi(t)dt = 0, \quad \int \Phi(t)^2 dt = 1$$

Orthogonality to low-order polynomials

$$\int t^m \Phi(t)dt = 0 \;\; 0 \leq m \leq n$$

- Example: Morlet (mother) wavelet

$$\Phi(t) = \frac{c}{\pi^{1/4}} e^{-\frac{t^2}{2}} \left(e^{it} - \kappa \right) \sim e^{-\frac{t^2}{2}} e^{it}$$

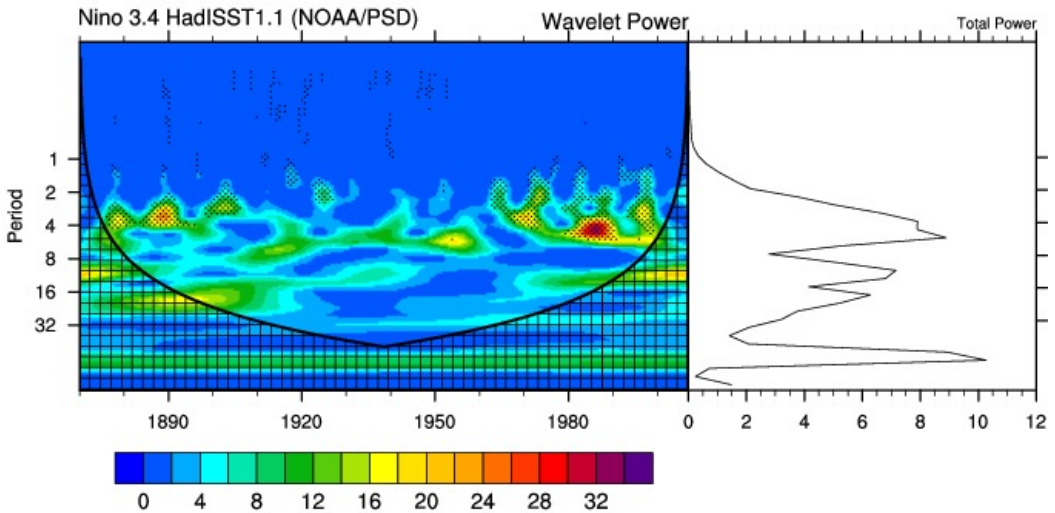
- Vary scale and time shift $\rightarrow$ wavelet basis

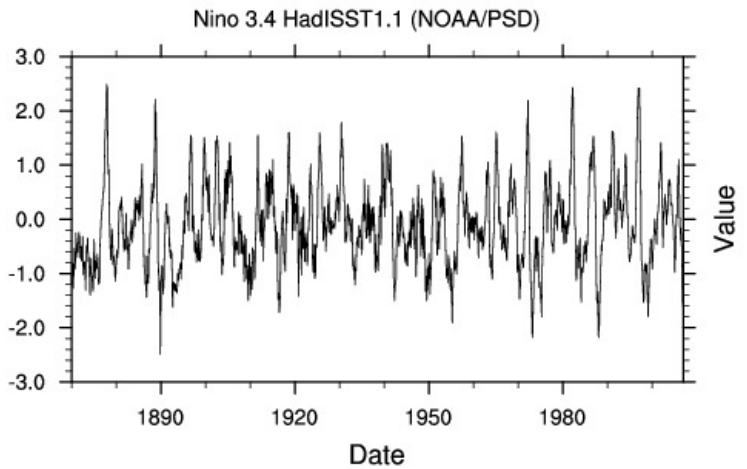
$$\Psi_{a,b}(t) = \Phi\left(\frac{t-b}{a}\right)$$

Wavelet transform

Cf Fourier

$$\mathcal{W}[h](a,b) = \frac{1}{\sqrt{a}} \int \Psi^*_{a,b}(t) h(t) dt$$





[link]

Estimation of H with wavelets

Simonsen *et al.* (1998) [link]

Reminder: if h is self-affine, $h(\lambda t) = \lambda^H h(t)$

$$h(t) = \lambda^{-H} h(\lambda t)$$

thus

$$\begin{aligned}\mathcal{W}[h(t)](a,b) &= \mathcal{W}[\lambda^{-H} h(\lambda t)](a,b) \\ &= \dots \\ &= \lambda^{-H-1/2}\mathcal{W}[h(t)](\lambda a, \lambda b)\end{aligned}$$

thus finally

$$\mathcal{W}[h(t)](\lambda a, \lambda b) = \lambda^{H+1/2}\mathcal{W}[h(t)](a,b)$$

Estimation of H with wavelets

Exploit

$$\mathcal{W}[h(t)](\lambda a, \lambda b) = \lambda^{H+1/2}\mathcal{W}[h(t)](a, b)$$

1. Average time shift out: define

$$W[h](a) = \text{mean}\left(|\mathcal{W}[h](a, b)|\right)_b$$

2. Scaling: take several λs

$$W[h](\lambda a) \simeq \lambda^{H+1/2} W[h](a)$$

3. Fit

$$\log W[h](\lambda a) = (H + 1/2) \log \lambda W[h](a) + cst$$

$$\rightarrow \text{slope } H + 1/2$$

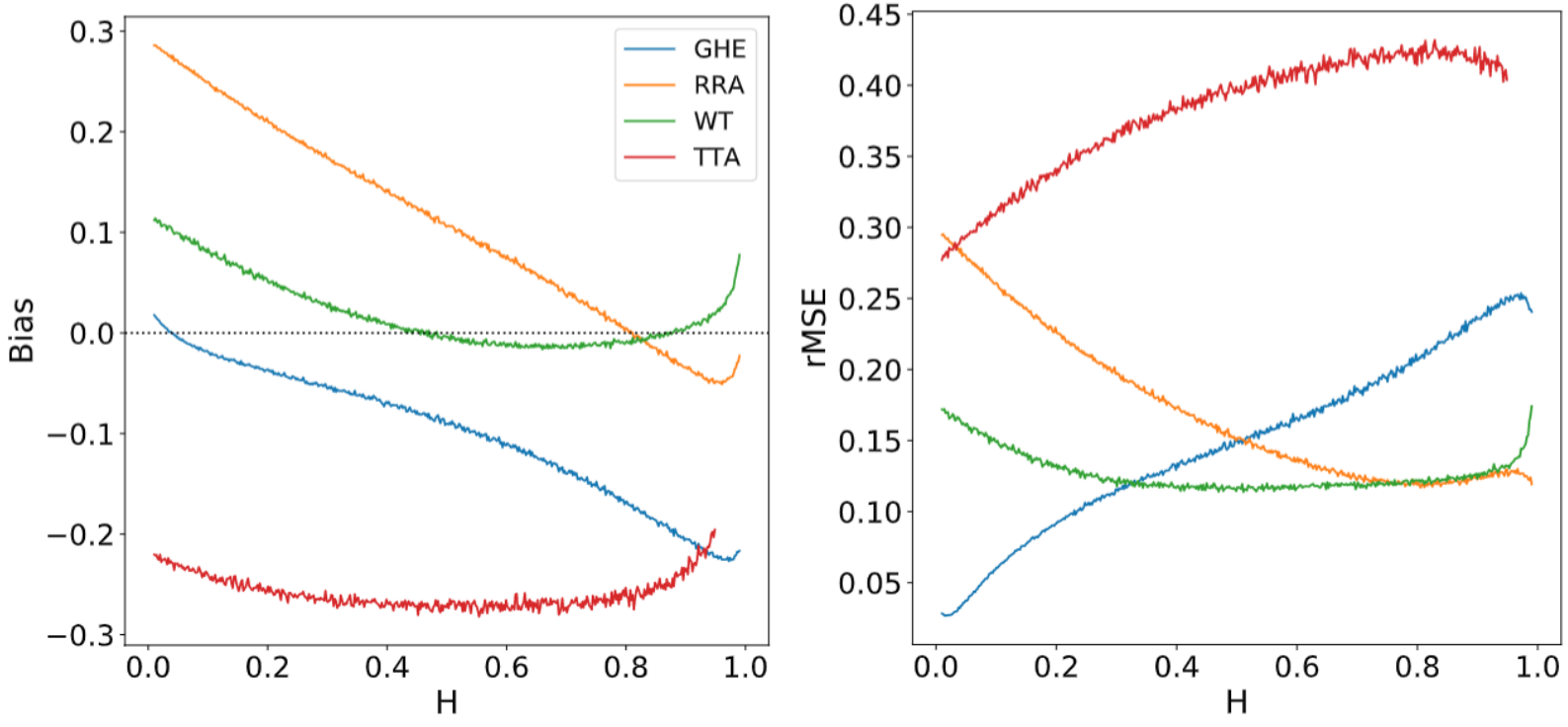
Other methods

- Detrended fluctuation analysis (large biases)
- Total Triangle area (2020) [link]
- Distance with self-affine pdfs.
- Ratio of scaling [Garcin (2021)]

$$\widehat{H}_t = \frac{1}{2\log(\tau_1/\tau_2)} \log\left(\frac{(T-\tau_2)\sum_{i=0}^{T-\tau_1}(p_{t-i}-p_{t-i-\tau_1})^2}{(T-\tau_1)\sum_{i=0}^{T-\tau_2}(p_{t-i}-p_{t-i-\tau_2})^2}\right)$$

Problem: short time series

Nguyen and Challet (2021) $T = 100$.
GHE=moment method, RRA=rescaled range, WT=wavelets, TTA= total triangle area



(a) Bias

(b) *rMSE*

Estimate H with DL

MultiLayerPerceptron (Dense) layers:

Rules of thumb

- # neurons first layer = dim_input
- Layers structure
  - Rectangle □▷
  - Diamond ◁▷
  - Triangle ▷
  - Bottleneck ▷◁▷
- Activation functions: some intuition from first principles

Estimate H with DL: convolutional networks

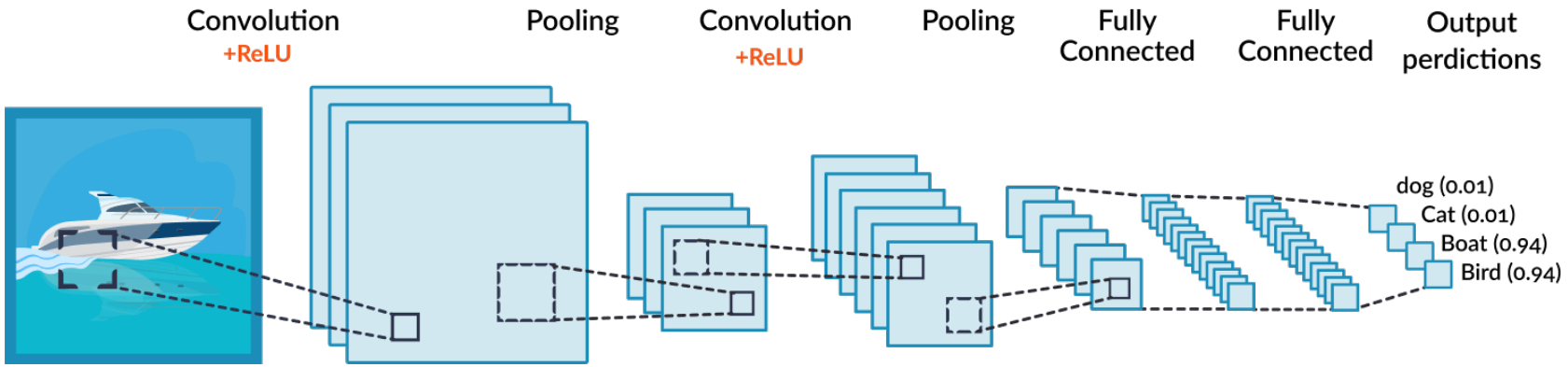
- fBm: convolution
- Generically, convolution between $\{x_i\}$ and kernel ϕ

$$\sum_{k=-K/2}^{K/2} \phi_k x_{i-k} \quad i = 1, \cdots, T$$

- K : kernel size

CNNs for images: typical architecture

[source]



- Kernels in parallel
- Smaller images, more kernels
- One feature per kernel

CNNs for images

[source]

- Kernels in parallel
- One feature per kernel
- The deeper, the more and more complex the features

- Interactive visualization [link]

CNNs

- Convolution between $\{x_i\}$ and kernel ϕ (learned)

$$\sum_{k=-K/2}^{K/2} \phi_k x_{i-k} \quad i = 1, \cdots, T$$

- Then activation of linear convolution

$$\tilde{x}_i = \Phi \left(\sum_{k=-K/2}^{K/2} \phi_k x_{i-k} \right)$$

- From Keras help:

```
tf.keras.layers.Conv1D(
    filters, kernel_size, strides=1, padding='valid',
```

  - filters = number of parallel kernels
  - kernel_size= K (cf above)
  - padding='same' keeps dimension of input by adding zeros at the start and end of x
  - padding='causal' adds zero at the start (see [link]) and only takes input from the past

CNN (1D): other ingredients

Between layers

- pooling: reduce dimension, filter
  - MaxPooling: $\max_{j \in [i-\delta, i+\delta]}(\tilde{x}_j)$; δ: parameter
  - AveragePooling

- dropout: random neurons during training, avoids overfitting
  - rate= probability to disable a given neuron during training
  - rules of thumb: 0.5 in hidden layers, 0.8 in input layers
  - only during `.fit()`, not `.predict()`
  - equivalent to weight constraints

H from CNNs

Stone, Quant. Fin. (2020) [link] CNNs

- the first layer, with 32 kernels;
- max pooling layer;
- a dropout layer, with rate $= 0.25$;
- the second layer, with 64 kernels;
- max pooling layer;
- a dropout layer, with rate $= 0.25$;
- the third layer, with 128 kernels;
- max pooling layer;
- a dropout layer, with rate $= 0.4$;
- a dense layer with 128 units;
- a dropout layer, with rate $= 0.3$.

Leaky ReLU ($\alpha = 0.1$), maxpooling of size 3, 30 epochs, batch size=64

Challenge: image to H

- Time series are images
- Input a time series as its image
- Use Conv2D networks

CNN $\longleftrightarrow$ wavelets

- CNN

$$\Phi\left(\sum_{k=-K/2}^{K/2} \phi_k x_{i-k}\right)$$

  and wavelets are convolutions

- Wavelets fix a mother function

- **Maths**: S. Mallat 'CNNs really are wavelets' [link]
  $\rightarrow$pre-filtering with wavelets helps learning much faster

- **DL:** what is a wavelet? Who cares?

Bonus topic: early stopping

- Until now, fix number of epochs
- Idea: stop when learning is useless
- Early stopping [link]: stop when loss or val_loss does not decrease
  - minimum/average over the last $time_{\text{patience}}$ steps
  - either train or validation loss
- code

```
from tensorflow.keras.callbacks import EarlyStopping
es = EarlyStopping(monitor='val_loss', mode='min', patience=10)
model.fit(        ,callbacks=[es])
```