The *Stock Transformer**

Multi-dimensional financial time-series forecasting via *self-attention*

Davide Roznowicz[◊○] | Emanuele Ballarin[◊]

** which is not actually a Transformer :)*

Deep Learning *end-of-course* project

University of Trieste, 26/07/2021

◊ University of Trieste, Dept. of Mathematics
○ SISSA

Introduction (1)

In order to intuitively figure out **why** this specific problem and this peculiar *Deep Learning* technique have been chosen to dig into, imagine the following **real-world scenario**:

A *trading professional/fund manager* might need support to **make better decisions** at his trading desk, in his day-to-day operations.

As many others, he specializes in **trading few stocks**.
Thus, he needs to have insightful **quantitative indicators about the overall market**, seen as an aggregate of (mainly) stocks to grasp the mood/behaviour of different interacting phenomena.

Introduction (2): *customer* requirements

- Often tends to **rebalance his portfolio weekly**;
- Has **enough computational power** to potentially re-train a model *often*;
- Would like to integrate the model among his **main quantitative indicators**;
- Wants a **relevant subset of the market** to be taken into account by the model: not only to forecast single-stock-prices directions, but also **sectors behaviour as a whole**;
- Believes that not only stock time series should be included, but also some **relevant *macro indicators* about commodities, *market-health* and volatility**.

Why not... recurrent models?

Previous attempts at *end-to-end time series modelling* mainly relied on *deep recurrent models*, eventually *gated*.

- Inability to grasp substantial long-term dependencies *(see Vecoven et al., 2020)*;
- Inability to transparently model hierarchical structures;
- Computational burden in training *(w.r.t. e.g. CNNs, FCNs... even on GPUs!)*;
- Difficulty in handling > *1D*

Outcomes: mixed, generally inconclusive. But huge *online popularity*.

Why not... learning system dynamics?

RNN & friends are generally regarded as *models of dynamical systems*. Once learned, they should be able to map any sequence of *past input* sequences into their (most likely) future evolution. E.g.: learning *laws of physics*.

- The existence of such laws for financial markets is strongly debated;
- Some even question their learnability altogether *(brittleness and the fat tails problem)*;
- Such *laws* may change in time, due to external interaction *(e.g. social, legislative, ...)*

Also, not a problem to *retrain whenever needed (even daily?)*

→ Exploit the *almost-overfitting* regime: learn an *input/output map* conditional on the *present*.

Transforming the transformer for time series data

Transformers (and *transformer-like* architectures) solve or ease many of the issues traditionally associated to *RNNs* (except: multidimensional data handling), and better suit the learning of conditional mappings.

On the other hand, they were conceived for *language modelling tasks*.

- Input to the model must be a sequence of tokens (usually *dictionary-embedded* words);
- It models *output sequence* probabilities, conditioned on previous outputs, in the space of tokens *(a.k.a. proper encoding/decoding)*;
- Embeddings must be easily reversible, and the same for input and output.

¿So what?

The *Stock Transformer*... for real!

Browsable map

of the architecture

Link to the original

transformer architecture

Gathering *data* from *raw data* (1)

Raw data source (API)

Daily, adjusted closing price for the largest stocks by market cap in the *SP500*

\+

Daily, adjusted closing prices for macroeconomic indicators and commodities

Financial time series, subsequently called *financials*

Gathering *data* from *raw data* (2)

Raw data source (API)

Contextual data, giving information about time: e.g. *daily*, *monthly*, *yearly*, *infinite* frequencies

Contextual time series, subsequently called *context*

Pre-processing

Financials:

1. Consider a set of time series *(i.e*. a *multidimensional t.s.*), each containing **daily adjusted closing prices**;
2. Since missing data might occur for some days, **fill them via linear interpolation**;
3. Compute **daily returns**.

Context:

1. Consider a set of time series *(i.e*. a *multidimensional t.s.*), each containing **domain-specific contextual information**;
2. Convert them to the corresponding **frequencies**;
3. When needed: apply specific **token-wise aggregation function**.

Training (1): optimizers, overview

Less *mainstream*, but more *robust* (to gradient noise, to gradient variance, to *loss-landscape raggedness*) optimization techniques have been chosen:

RAdam within a *Lookahead* loop

Why, exactly?

Training (2): optimizers, detail

Why an **adaptive optimization method** (and not, e.g., SGD)?

Great **simplification of *l.r.* scheduling**, which may become beneficial when the parameter space is large.

Why ***RAdam***, among all adaptive optimizers?

- The go-to optimizer *Adam* has scarce generalization ability and stability when gradients are small (in norm);
- *Adam* gradients have **high variance in the initial steps** (often requiring warmup).

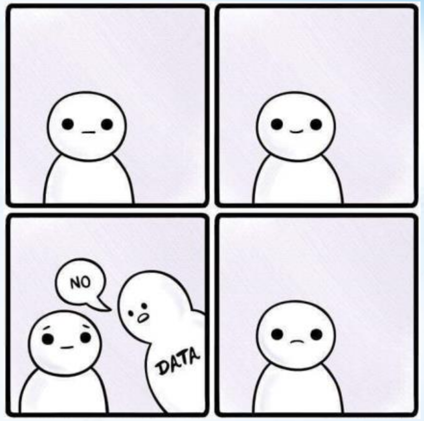
Why additionally adopting inner-loop optimization, i.e. **Lookahead**?

Greater **robustness to loss-landscape noise**.

Training (3): pitfalls

Training *deep neural nets* is undoubtedly hard; training large, complex ones is even harder.

- Large batches (> 128), *small-kernels* regime → collapse to the mean;
- Large batches (> 128), *large-kernels* regime → kernel over-averaging;
- *Un-incisive* CNN compression → parameter imbalance towards the decoder;
- *Flat* decoder → parameter imbalance towards the decoder
  → excessive parameterization
- *Too deep* encoder → *scramble effect*, *gradient decoupling*



Fantastic tricks... and where to find them

The *obvious:*

- *Medium-sized* (~ 5) kernels, mostly overlapping;
- Relatively shallow (~ 4) encoder stacking.

The *less obvious:*

- Small-batch (~ 32) training *(after Luschi and Masters, 2018)*;
- *Compress-in-time*, *expand-in-features* for the CNN featurizer;
- *Convergent halving l.r.* scheduling;
- Unshuffled training *(dynamics-aware* as for RNNs).

The *unorthodox:*

Berenstein-sized kernel-stencil over-featurization $\sim 2 * 5 * \sum_{i=1}^{5} i$;

The actual results (so far)

A *decently*-performing, *medium*-sized, *fast*-training model that has surely room for improvement, needs more analysis and polishing, but also (on average) performs significantly better than pure chance.

- $\sim 80\%$ *return-sign* concordance (vs. $\sim 51\%$ distribution of signs imbalance);
- $\pm 50\%$ average error, increasing with forecast time (vs. $\pm 100\%$ *mean-learning*).

- Total learnable parameters: $< 3\ mln$;
- Training time to full convergence: $\sim 55'$ @ 1 *NVidia V100.*

- Why does it *plateau*?

```
Total params: 2,811,587
Trainable params: 2,811,587
Non-trainable params: 0
Total mult-adds (M): 315.63
==========================================================================================
Input size (MB): 0.38
Forward/backward pass size (MB): 365.36
Params size (MB): 11.25
Estimated Total Size (MB): 376.99
```

Future developments and open problems

Depth-first:

- Principled analysis of *what* the model actually learns;
- Principled analysis of the *nature of the loss plateau*;
- Ablation study on the *correlator*;
- Improvement of contextual information and its processing;
- Exploration of different architectures: *proper Transformer* and *impedance-matching Transformer*, *fully-convolutional* approaches.

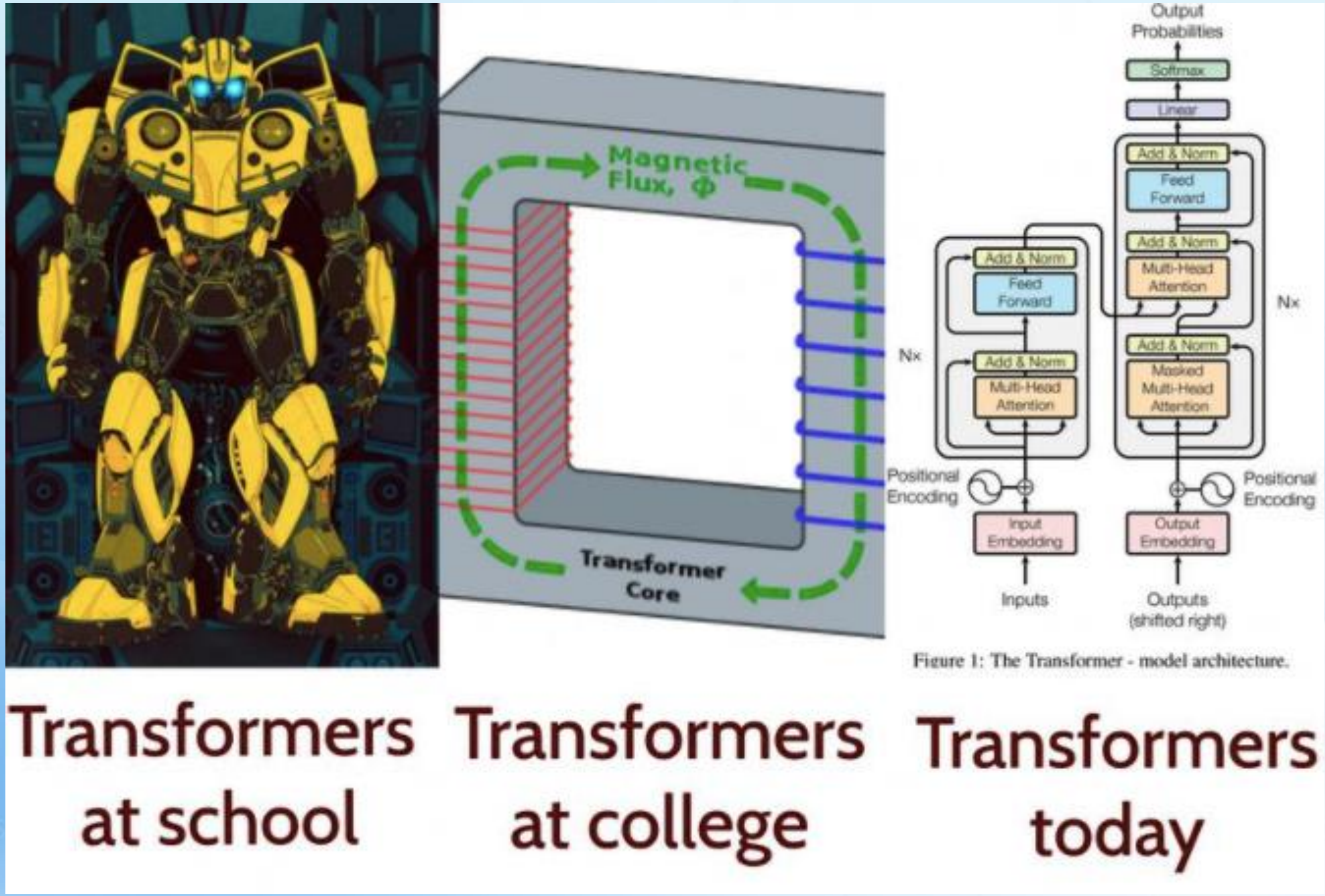
Breadth-first:

- *Frequent* re-training and transfer learning dynamics;
- Integration with pre-existing *"classical" feature banks* or *filters*.

Bibliographic inspiration

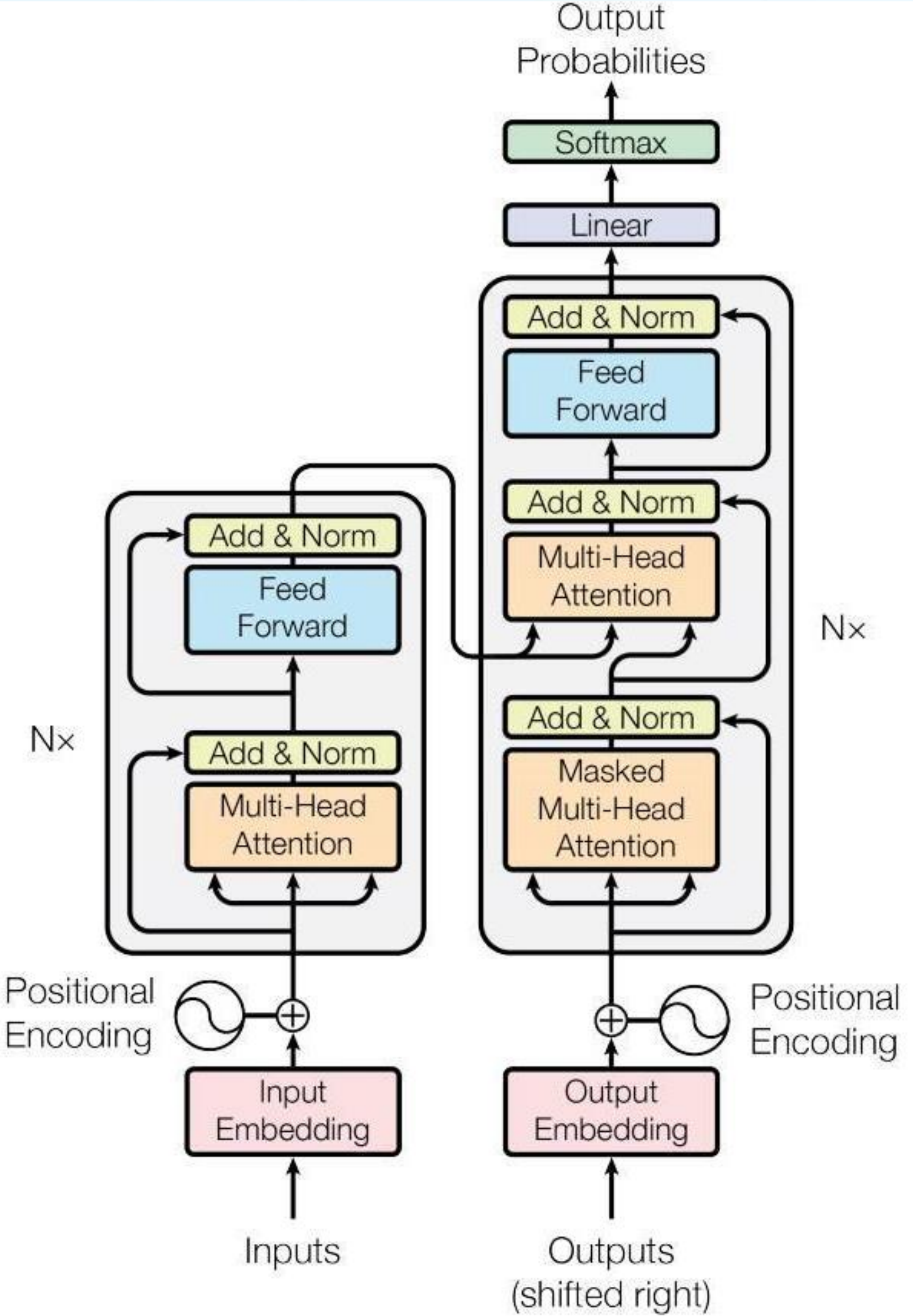
- *"Temperature Forecasting via Convolutional Recurrent Neural Networks Based on Time-Series Data"* (Zhang et al., 2019);
- *"Deep Convolutional Cascade for Face Alignment In The Wild"* (Dapogny et al., 2019);
- *"Attention Is All You Need"* (Vaswani et al., 2017)
- "Sparse attention based separable dilated convolutional neural network for targeted sentiment analysis" (Gan et al., 2019);
- *"Enhancing the Locality and Breaking the Memory Bottleneck of Transformer on Time Series Forecasting"* (Li et al., 2020);
- *Dilated causal convolution with multi-head self attention for sensor human activity recognition* (Hamad et al., 2021);
- *"On the Variance of the Adaptive Learning Rate and Beyond"* (Liu et al., 2020);
- *"Revisiting Small Batch Training for Deep Neural Networks"* (Masters and Luschi, 2018);
- *Training Very Deep Networks* (Srivastava et al., 2011).

Thanks for your... *attention*!



[https://github.com/emaballarin/financial-wholenamycs]

End of slides


Output
Probabilities
Softmax
Linear
Add & Norm
Feed
Forward
Add & Norm
Multi-Head
Attention
Add & Norm
Masked
Multi-Head
Attention
Nx
Add & Norm
Feed
Forward
Add & Norm
Multi-Head
Attention
Nx
Positional
Encoding
Positional
Encoding
Input
Embedding
Output
Embedding
Inputs
Outputs
(shifted right)

[Back](#)

Algorithm 2: Rectified Adam. All operations are element-wise.

Input: $\{\alpha_t\}_{t=1}^{T}$: step size, $\{\beta_1, \beta_2\}$: decay rate to calculate moving average and moving 2nd moment, θ_0: initial parameter, $f_t(\theta)$: stochastic objective function.

Output: θ_t: resulting parameters

$m_0, v_0 \leftarrow 0, 0$ (Initialize moving 1st and 2nd moment)

$\rho_\infty \leftarrow 2/(1-\beta_2) - 1$ (Compute the maximum length of the approximated SMA)

while $t = \{1, \cdots, T\}$ **do**

  $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Calculate gradients w.r.t. stochastic objective at timestep t)

  $v_t \leftarrow 1/\beta_2 v_{t-1} + (1-\beta_2) g_t^2$ (Update exponential moving 2nd moment)

  $m_t \leftarrow \beta_1 m_{t-1} + (1-\beta_1) g_t$ (Update exponential moving 1st moment)

  $\widehat{m_t} \leftarrow m_t/(1-\beta_1^t)$ (Compute bias-corrected moving average)

  $\rho_t \leftarrow \rho_\infty - 2t\beta_2^t/(1-\beta_2^t)$(Compute the length of the approximated SMA)

  if *the variance is tractable, i.e.,* $\rho_t > 4$ **then**

    $l_t \leftarrow \sqrt{(1-\beta_2^t)/v_t}$ (Compute adaptive learning rate)

    $r_t \leftarrow \sqrt{\frac{(\rho_t-4)(\rho_t-2)\rho_\infty}{(\rho_\infty-4)(\rho_\infty-2)\rho_t}}$ (Compute the variance rectification term)

    $\theta_t \leftarrow \theta_{t-1} - \alpha_t r_t \widehat{m_t} l_t$ (Update parameters with adaptive momentum)

  else

    $\theta_t \leftarrow \theta_{t-1} - \alpha_t \widehat{m_t}$ (Update parameters with un-adapted momentum)

return θ_T

Back