Machine Learning

Time series forecasting

Part 3:
neural models

Why deep learning?

Complex tasks

- primarily excel at highly complex tasks with lots of data:
- **big data:**
  - high frequency and long history
- **complex tasks:**
  - complex relations, e.g. multiple variable seasonalities
  - dynamic, requiring adaptation, with changing patterns and noise
- **multivariate time series:**
  - many time series, with cross-series relations
- **long forecasting horizons:**
  - can be quite precise compared to classical methods

Direct multi-step (DMS) forecasts

- statistical models basically always perform **autoregressive forecasting:**
  - forecast 1 step ahead at a time, assume previous forecasts are true
  - this results in error accumulation and higher error bias
  - also known as iterated multi-step (IMS) forecasting (or recursive forecasting)

- neural networks are typically **multioutput**, i.e. can easily have many output neurons
- this results in **direct multi-step (DMS)** forecasts, which:
  - avoids error accumulation, but has higher error variance
  - acts as regularization, since it has to optimize many horizons at a same time

IMS vs DMS forecasting - additional resources

- CrossValidated - Time Series One Step Ahead vs N-Step Ahead
- "Recursive and direct multi-step forecasting: the best of both worlds" S. Taieb, R. Hyndman
- "A comparison of direct and iterated multistep AR methods for forecasting macroeconomic time series" M. Marcellino et al.
- "When are Direct Multi-Step and Iterative Forecasts Identical?" T. McElroy
- "Direct Versus Iterated Multiperiod Volatility Forecasts" E. Ghysels et al.
- "An Empirical Investigation of Direct and Iterated Multistep Conditional Forecasts" M. McCracken, J. McGillicuddy

Pretraining

- **transfer learning** and **pretraining** of time series recently became possible:
  - novel architectures, particularly transformers
  - available massive datasets
  - utilize previous knowledge and reduce overfitting
- **foundational models** start to emerge, with e.g. few-shot and zero-shot capabilities
- does this work?
  - many works lack fair evaluation, or fail against simple baselines
  - comparisons are often artificial and unrealistic, e.g. lots of data, highly multivariate
  - just research currently, not well tested in the industry
- definitely a **research direction** in near future

Deep learning approaches

Deep learning approaches

- **linear networks:**
  - Linear, DLinear, NLinear, RLinear etc.
  - simple, 1-layer networks for univariate time series
- **MLP-based models:**
  - N-BEATS, N-HiTS, TSMixer, TiDE etc.
  - learn complex relations and decompositions by using stacks of MLPs
- **transformers:**
  - PatchTST, Autoformer, FEDformer, Pyraformer etc.
  - pure transformer architectures, often with complex attention modifications

Deep learning approaches

- **pretrained foundational models:**
  - TimesFM, Chronos, Lag-Llama, Moirai, TimeGPT etc.
  - first really successful transfer learning for time series
- **State Space Models (SSMs):**
  - LSSL, MambaTS, Chimera, SpaceTime etc.
  - state-space models theory unifies ETS, CNNs, RNNs, and a few other things
- **graph neural networks (GNNs):**
  - T-GCN, DGSL, GaAN, STGNN etc.
  - typically used for spatio-temporal forecasting, e.g. traffic demand

Deep learning approaches

- **recurrent networks (RNNs):**
    - old (mostly obsolete), e.g. LSTM, GRU, DeepAR
    - modern, e.g. RWKV-TS, TFT, P-sLSTM
    - built for sequence prediction, fast inference, but can be hard to train
- **convolutional networks (CNNs):**
    - old (mostly obsolete), e.g. TCN, DeepTCN
    - modern, e.g. MICN, TimesNet, SCINet
    - typically based on dilated convolutions and causal convolutions

- classical ones are generally obsolete, but modern ones are noteworthy

Agenda

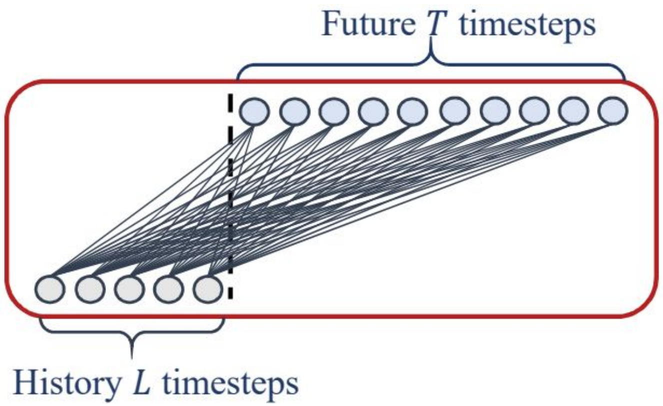
- we will go over **representative architectures** from the most commonly used groups:
    - linear models
    - MLP-based
    - transformers
- lastly, we will cover **important research direction** - pretrained foundation models
- we omit others, because:
    - SSMs are not well proven or popular (yet)
    - GNNs are specific for spatio-temporal forecasting
    - RNNs and CNNs are mostly obsolete (with some notable exceptions)

Linear networks

Linear

["Are Transformers Effective for Time Series Forecasting?" A. Zeng et al.]

- **LTSF-Linear** (Long-term Time Series Forecasting Linear)
- just a linear projection from L to T values, no activation function etc.
- in paper called just **Linear** (or Vanilla Linear)
- naturally performs direct multi-step (DMS) forecasts
- univariate, but performs very well also for multivariate problems
- why does this work?
  - relies **exclusively** on order and magnitude of time series values
  - so only those matter - how long ago something happened and how strongly
  - linearly mixes L last values, which is simple and avoids overfitting



DLinear and NLinear

- **expansions** of the Linear model, from the same paper
- **DLinear (Detrended Linear):**
  - first detrends time series with moving average
  - uses 2 Linear models, for trend and remainder, forecast is their sum
  - performs better for data with clear trend
- **NLinear (Normalized Linear):**
  - first normalizes by subtracting the last value from time series
  - predicts normalized series, adds back value to forecast
  - just a normalization that should stabilize training

Linear model equivalence

"An Analysis of Linear Time Series Forecasting Models" W. Toner, L. Darlow

- authors prove that:
  - DLinear, NLinear and Linear are **equivalent** to OLS linear regression
  - NLinear is just Linear + constraint (rows sum to 1)
  - all equivalent models have **closed formula** from OLS
- uses SVD for training, which is great: optimal, fast, stable
- incredibly simple, but wins in 72% of experiments, and performs great
- just a single hyperparameter - **lookback window L**
- even L2 regularization is not required

Linear networks - pros and cons

Pros:

- closed formula OLS
- great performance
- very fast, stable, simple
- avoids overfitting
- just a single hyperparameter

Cons:

- cannot learn very complex relations
- univariate - performs worse for strong cross-series correlations
- requires long time series for larger lookback L and learning long relations

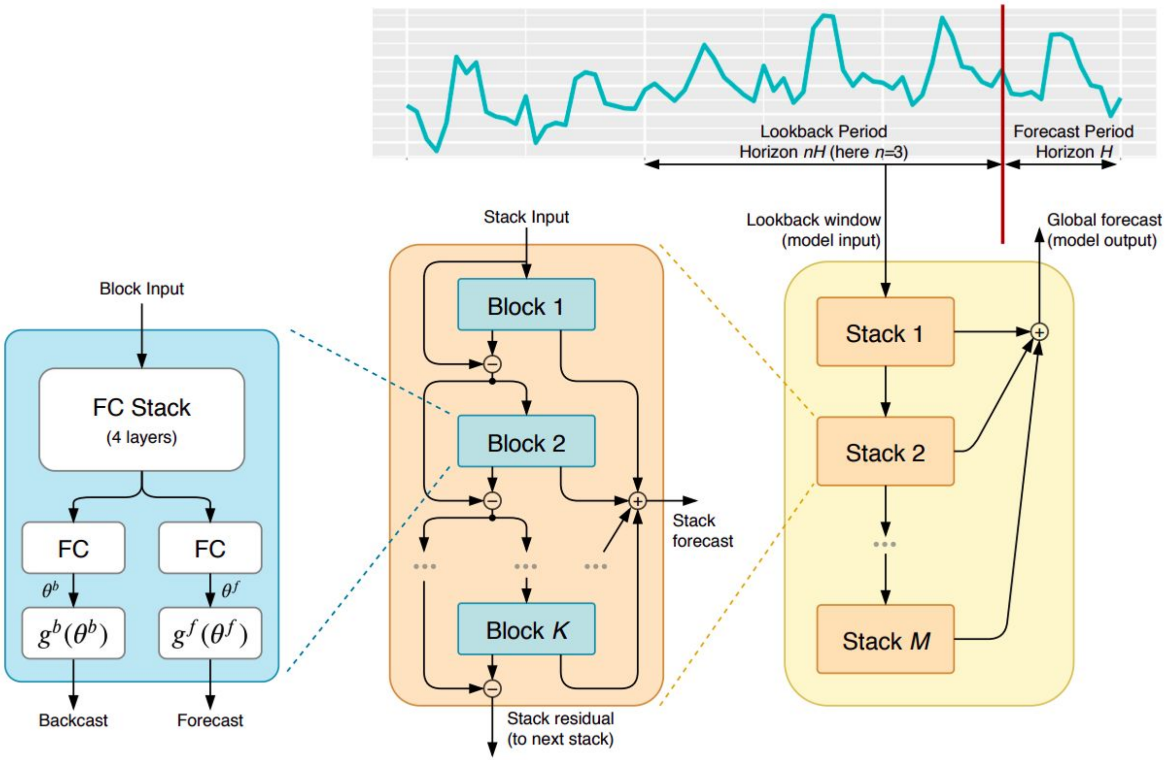
MLP-based models

N-BEATS

["N-BEATS: Neural basis expansion analysis for interpretable time series forecasting"
B. Oreshkin et al.]

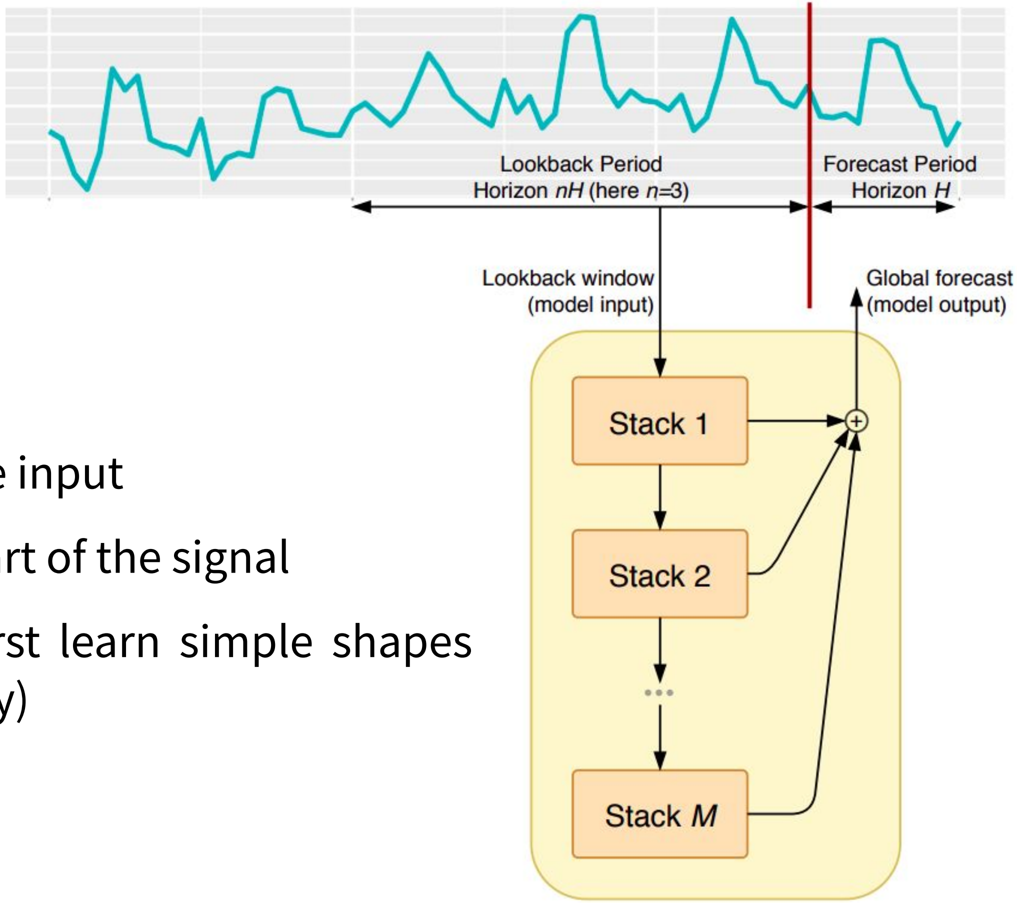
Combines a few key ideas:

- learn everything from raw data
- MLP as a basic building block
- doubly residual stacking
- stacked blocks
- basis expansion
- backcast



N-BEATS - stacked architecture

- first stack gets raw data of length *nH*
- model forecast is a sum from all stacks
- each stack has **2 outputs:**
    - forecast of length *H*
    - **residual** of its inputs
- stack subtracts what it learned from the input
- further ones only have to predict the part of the signal
- encourages **specialized** stacks, e.g. first learn simple shapes (trend), then more complex (seasonality)

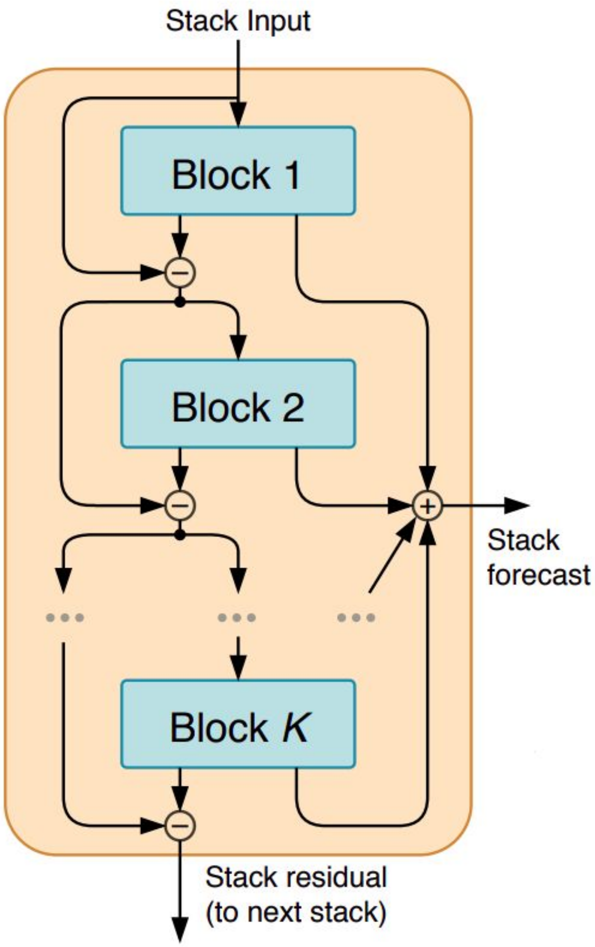


N-BEATS - variants

- paper proposed 2 variants: general and interpretable
- **general (N-BEATS-G):**
  - some N stacks, no structure enforced (linear basis)
  - free to learn arbitrarily complex relations, but need enough data
- **interpretable (N-BEATS-I):**
  - 2 stacks: trend and seasonality
  - trend stack: outputs trend forecast & detrended data
  - seasonality stack: outputs seasonality forecast
  - use dedicated basis functions for inductive bias:
    - polynomial (trend)
    - Fourier (seasonality)

N-BEATS - stack

- similar idea inside each stack, but with **blocks**
- block has 2 outputs:
  - partial forecast
  - **backcast**, estimating (reconstructing) its input data
- **residual connection:**
  - original data - backcast = residual
  - input into the next block
  - makes the job easier - removes parts of signal (data)
- called **doubly residual learning** in the paper (for stacks and for blocks)
- stack forecast = sum of block forecasts

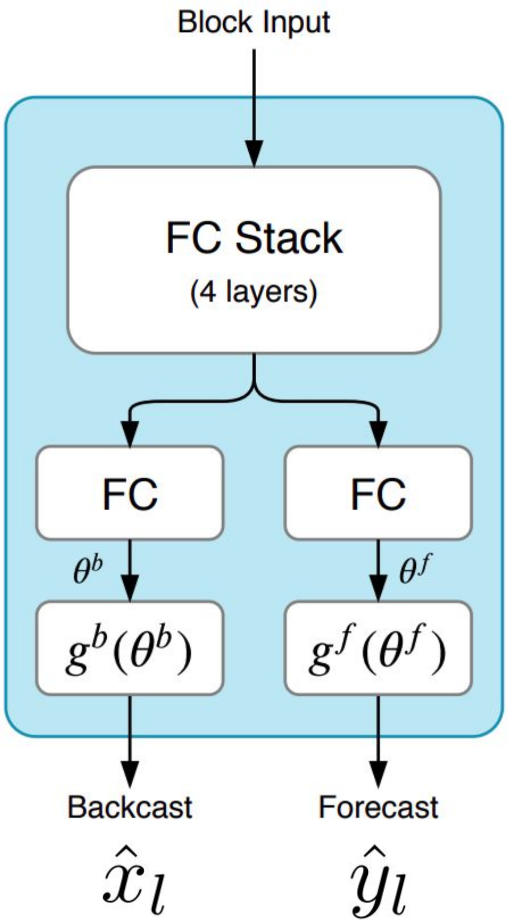


N-BEATS - block

- 2 outputs: forecast $\hat{y}_l$ and backcast $\hat{x}_l$
- major idea is to predict **basis coefficients** of basis *g*
- allows encoding **inductive bias** through basis choice, e.g. seasonality is periodic
- forecast and backcast use the same basis, but separate weights
- generic architecture uses linear basis, which just matrix multiplication, i.e. linear projection

$$\hat{y}_l = W_f \theta_f + b_f$$

$$\hat{x}_l = W_b \theta_b + b_b$$



N-BEATS - block

- interpretable variant encodes information in basis choise
- trend is uses **polynomial basis** of low degree, in paper *p*=2:

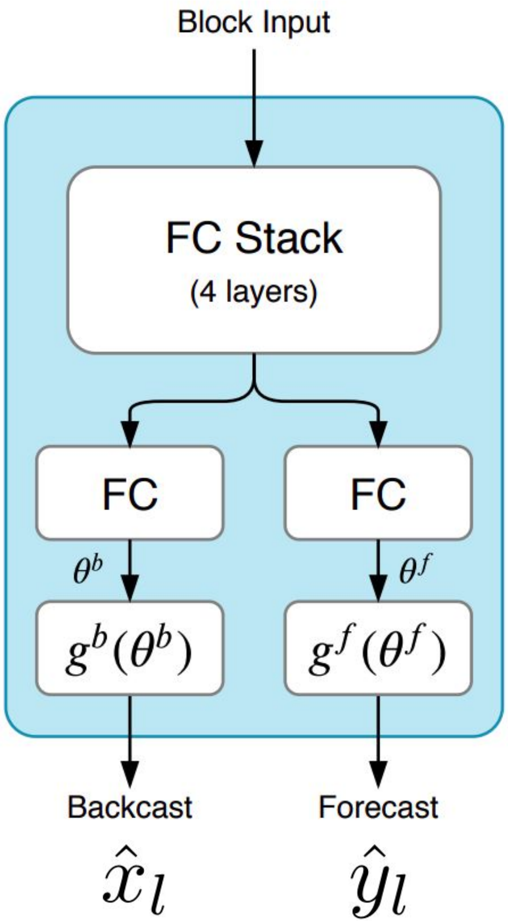
$$\hat{y}_l = \sum_{i=0}^{p} \theta_{f,i} t^i$$

t^i - time steps vector, linear grid raised to a given power

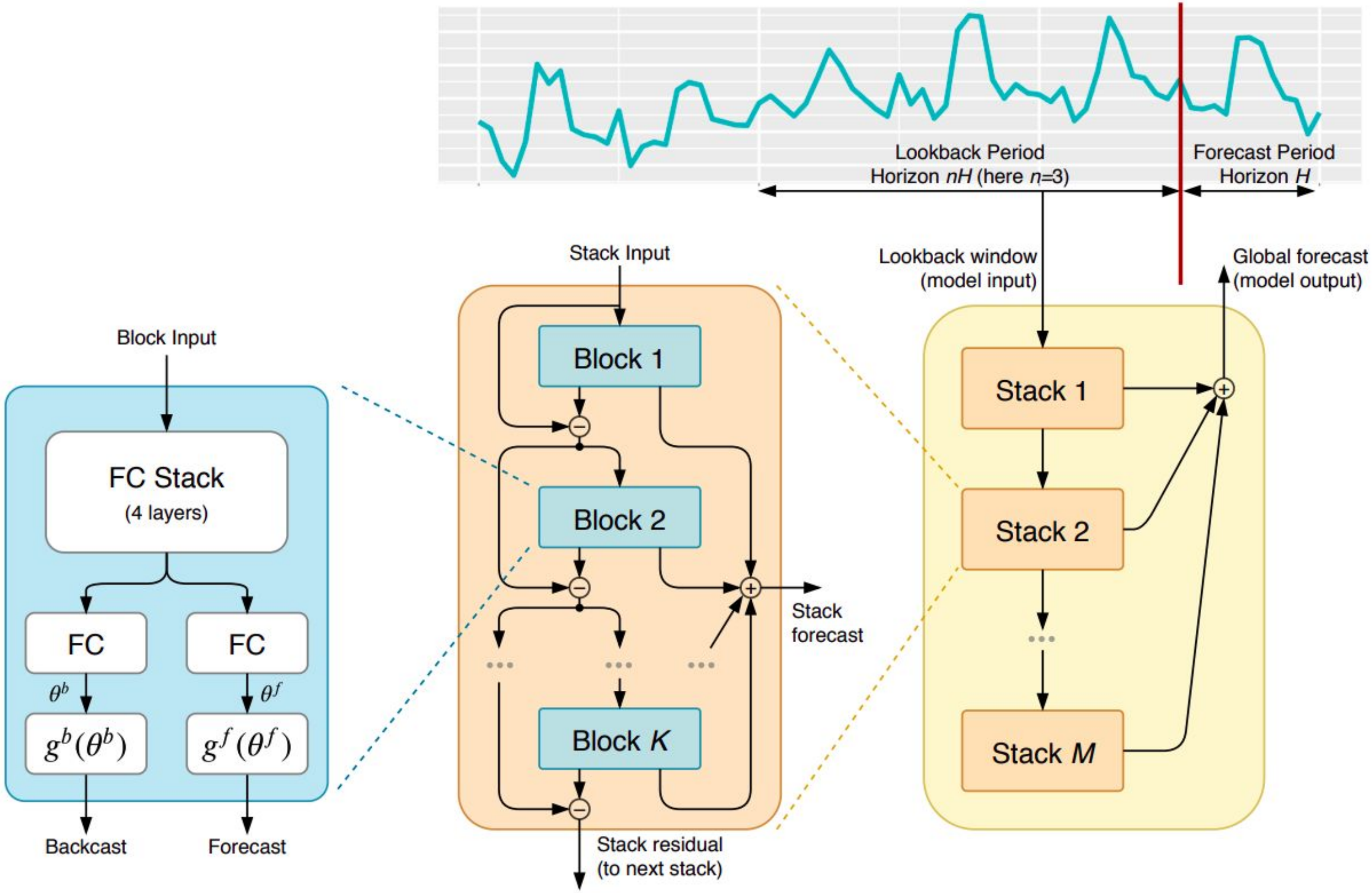
$$t^i = [0, 1, 2, ..., H-2, H-1]^i / H$$

- seasonality uses **Fourier basis**

$$\hat{y}_l = \sum_{i=0}^{\lfloor H/2-1 \rfloor} \theta_{f,i} \cos(2\pi i t) + \theta_{f,i+\lfloor H/2 \rfloor} \sin(2\pi i t)$$



N-BEATS - recap



N-BEATS - pros and cons

Pros:

- very flexible
- good results
- can model very complex seasonality
- interpretable variant

Cons:

- only univariate
- does not scale well to long forecasting horizons (but: N-HiTS)
- no exogenous variables (but: N-BEATSx)

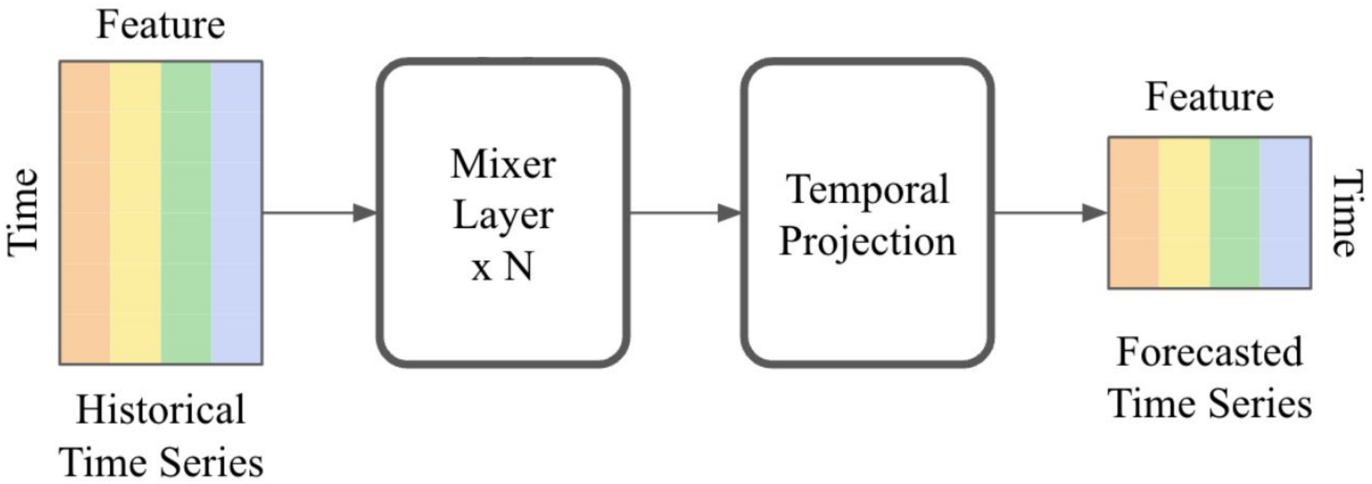
N-BEATS - additional resources

- "N-HiTS: Neural Hierarchical Interpolation for Time Series Forecasting" C. Challu et al.
- "Neural basis expansion analysis with exogenous variables: Forecasting electricity prices with NBEATSx" Kin Olivares et al.
- N-BEATS code in PyTorch Forecasting: blocks, whole model
- alternative explanation:
    - "N-BEATS — The First Interpretable Deep Learning Model That Worked for Time Series Forecasting" J. Dancker
    - "Optimizing Time Series Forecasting: Exploring N-BEATS Architecture for Improved Predictions" G. Sayago

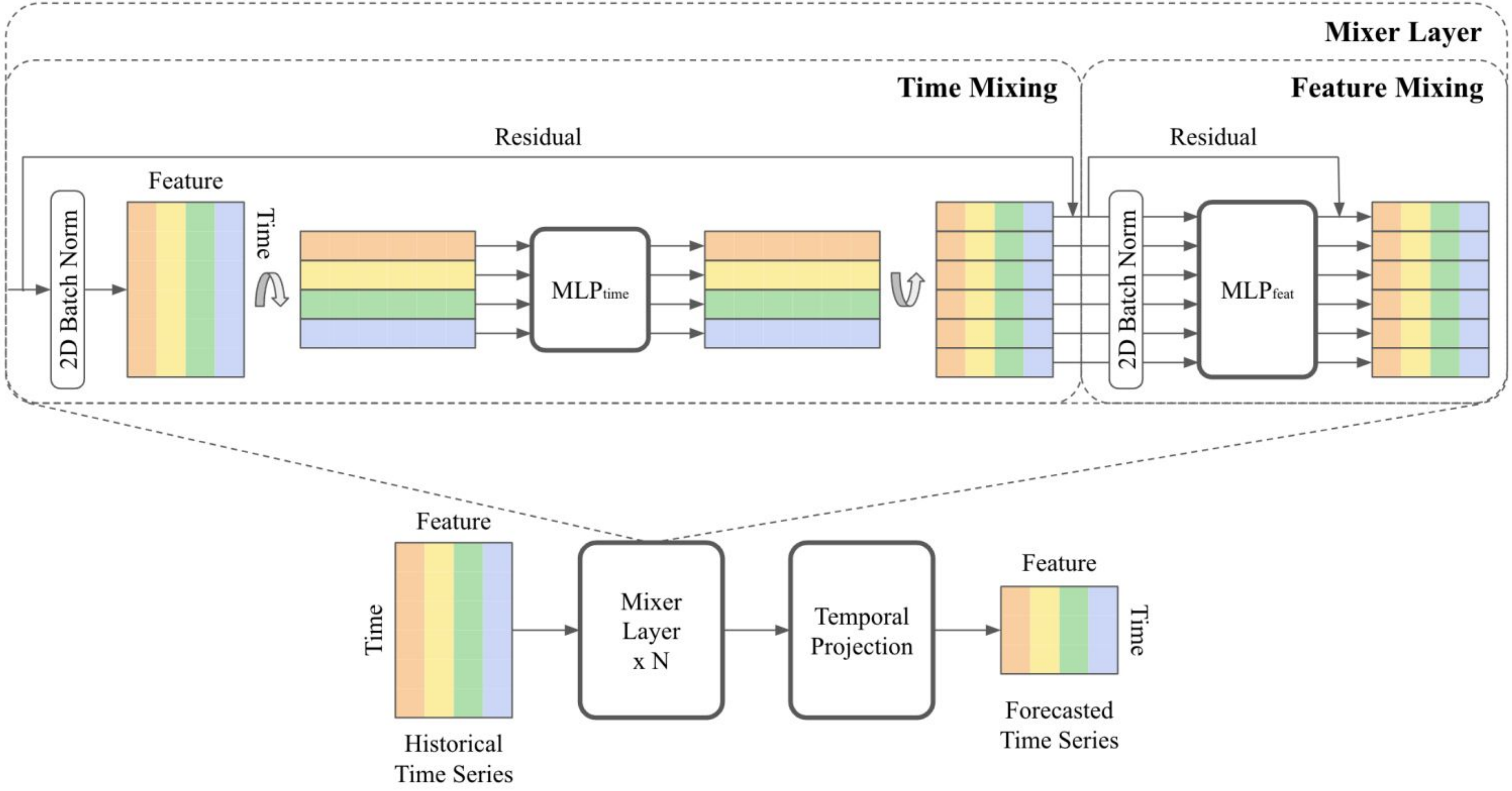
TSMixer

["TSMixer: An All-MLP Architecture for Time Series Forecasting" S. Chen et al.]

- **idea:**
  - "mixing" values in time or feature dimensions with MLPs
  - process separately: univariate time, multivariate features, exogenous variables

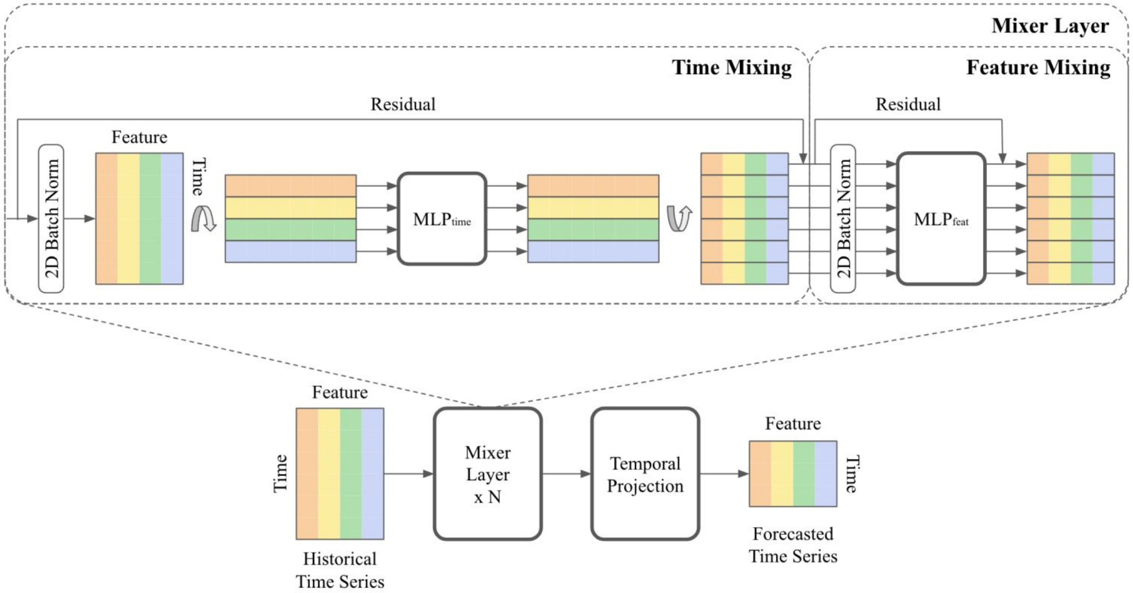


TSMixer - mixer layer



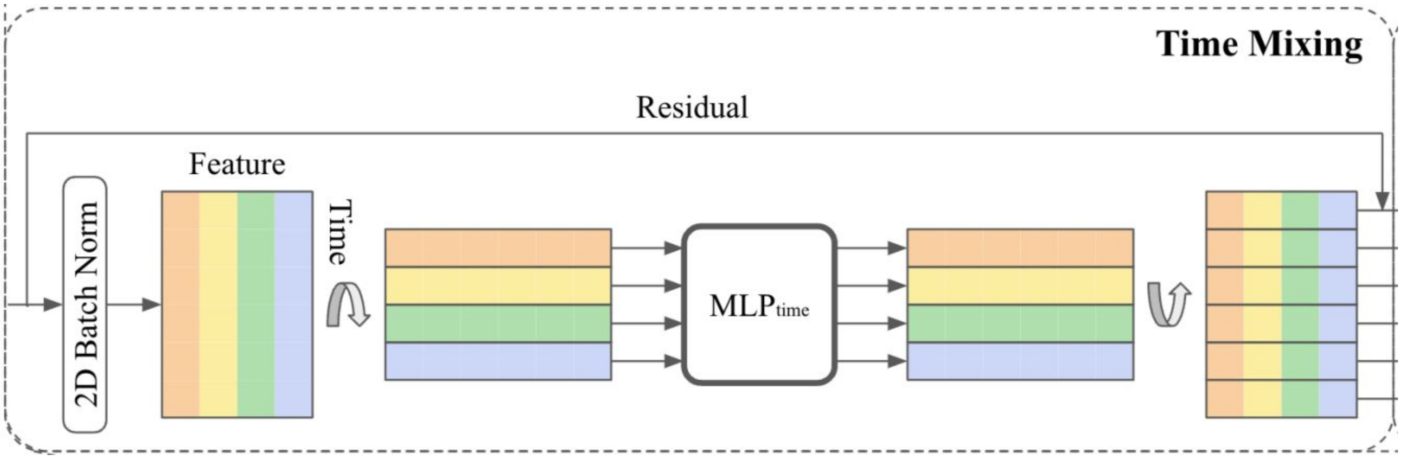
TSMixer - mixer layer

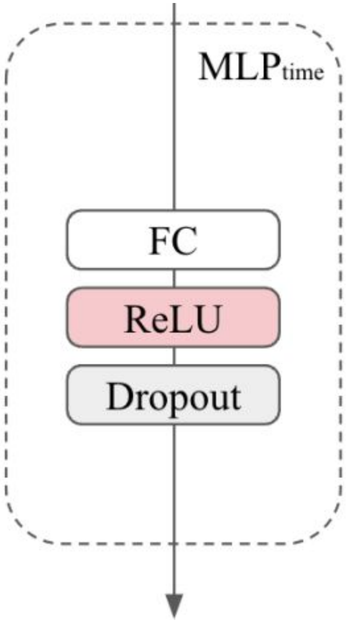
- **time mixing:** learn about time relations inside series
- **feature mixing:** learn about cross-series relations
- **separate** mixing reduces cost and complexity
- typical additions:
  - residuals
  - batch norm
- **temporal projection** is just a linear projection to horizon H



TSMixer - time mixing

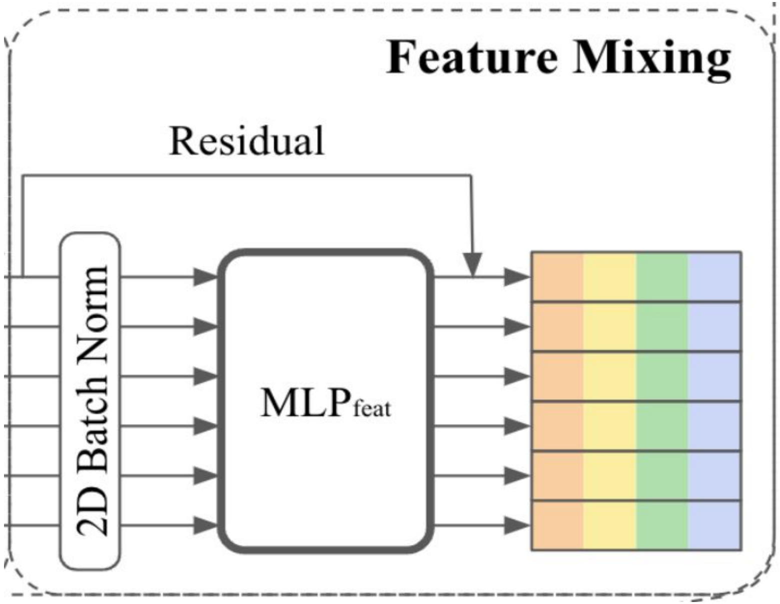
- **inspired by** the Linear model
- the simplest non-linearity: 1-layer MLP
- extracts time-varying information inside a single time series

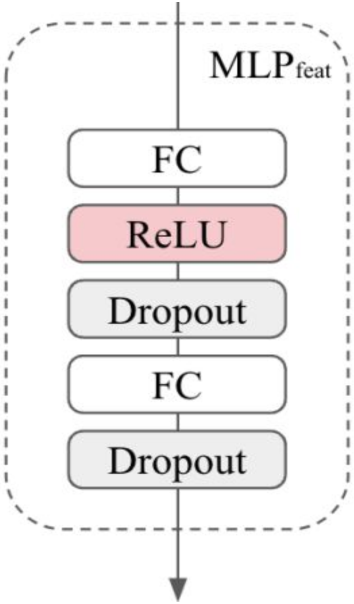




TSMixer - feature mixing

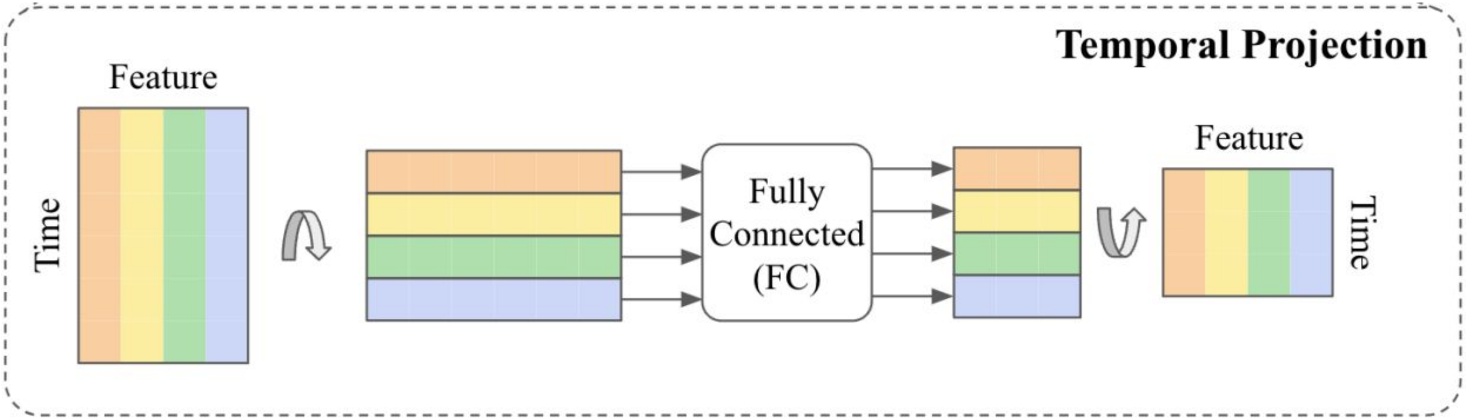
- **inspired by** the feature mixing in Transformer
- 2-layer MLP to learn more complex covariate relations
- extracts cross-series information





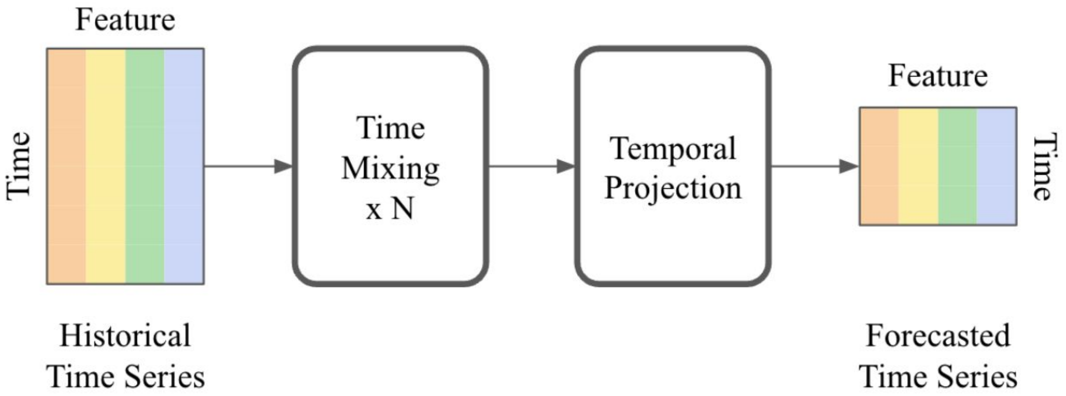
TSMixer - temporal projection

- **inspired by** the Linear model
- literally just a Linear model - simple linear projection from lookback L to to horizon H



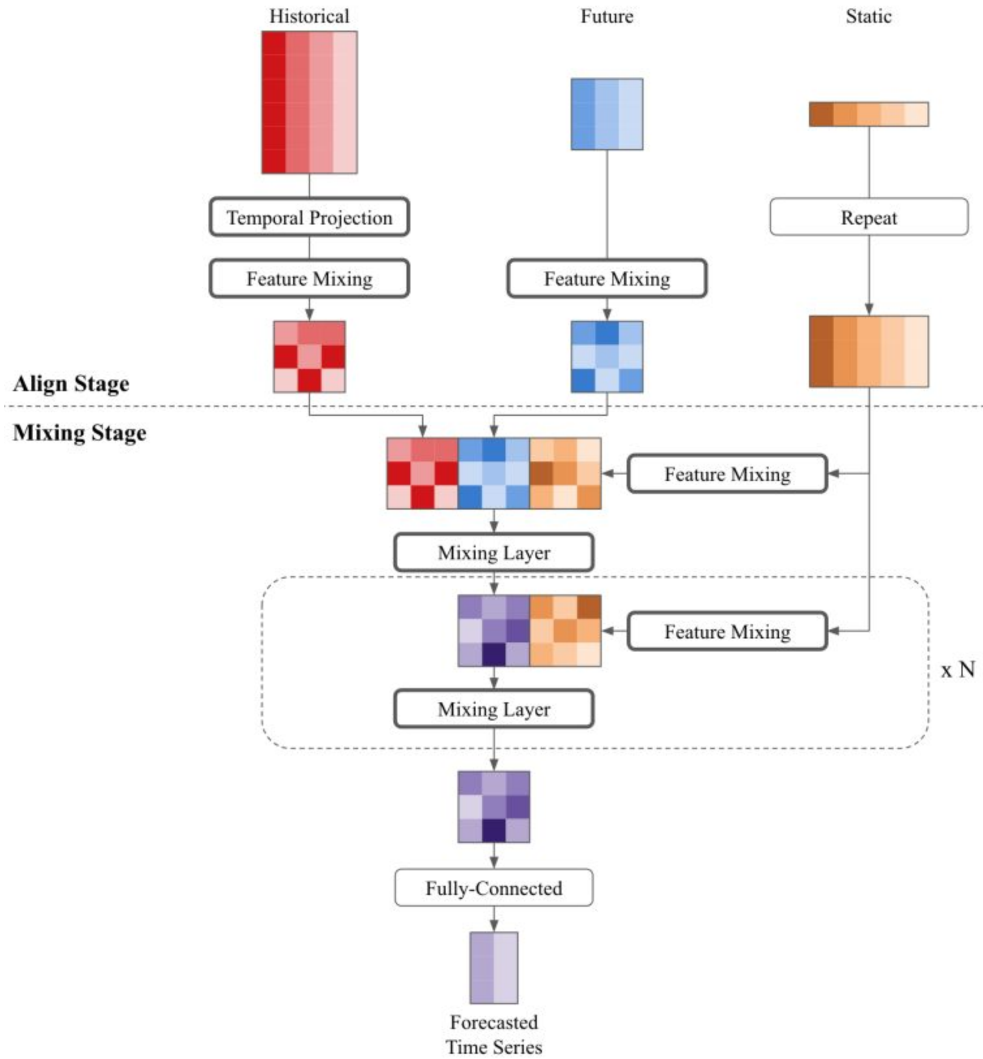
TMix-Only

- TSMixer variant for univariate time series
- only time mixing



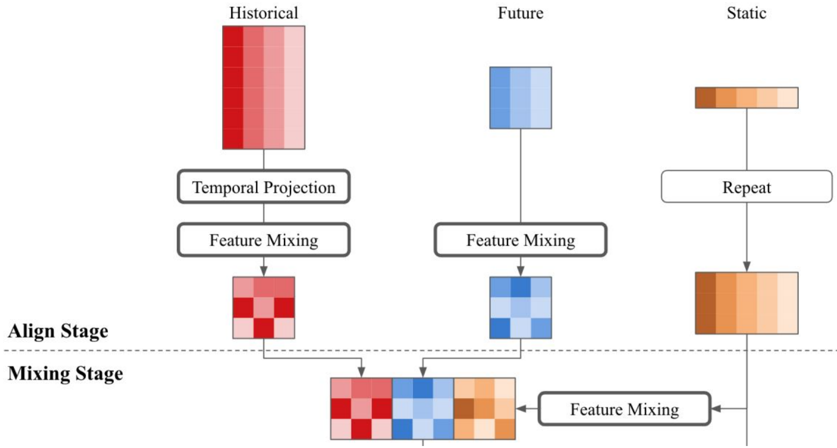
TSMixer - exogenous variables

- "auxiliary variables" in the paper
- can be:
  - **static**, e.g. shop location
  - **dynamic**, e.g. ongoing promotion
- historical data:
  - time series (1 or more)
  - dynamic auxiliary variables
- added after regular TSMixer forecast
- **correct** the pure time series forecast to account for exogenous variables



TSMixer - exogenous variables

- **historical** data:
  - time series values + past auxiliary dynamic features
  - all concatenated as time series
  - temporal projection to get `horizon` length
  - feature mixing learns interactions between series and features
- **future** data:
  - future dynamic auxiliary variables
  - we need to know them!
- **static** data:
  - repeated to `horizon` length



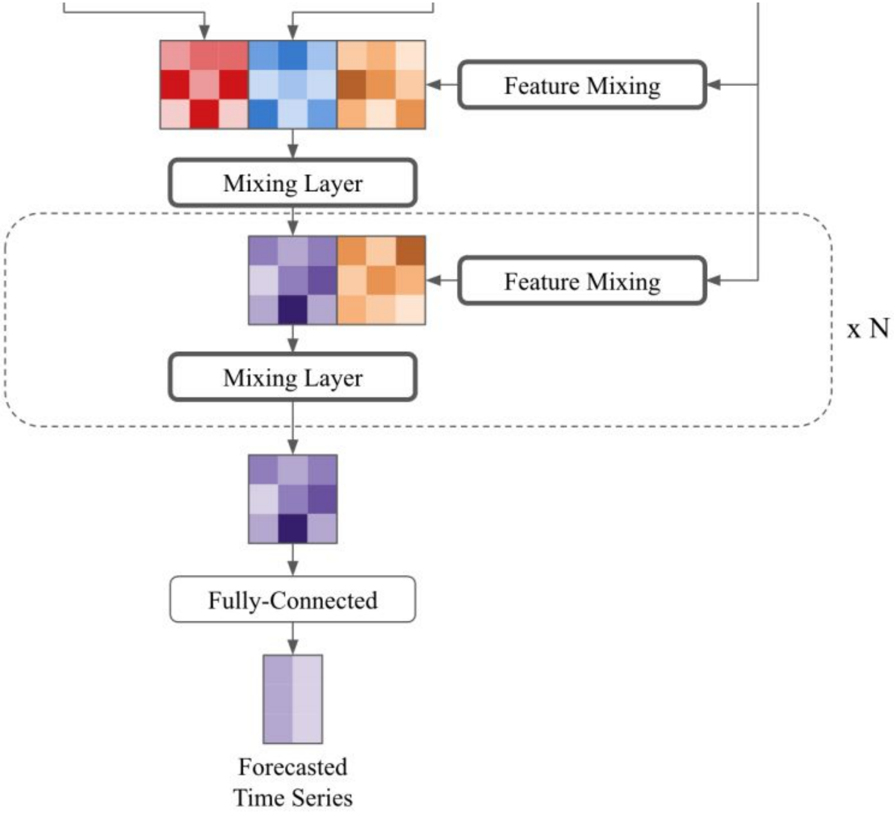
- they create **align** stage
- allows concatenating them like separate time series
- each has length `horizon`

TSMixer - exogenous variables

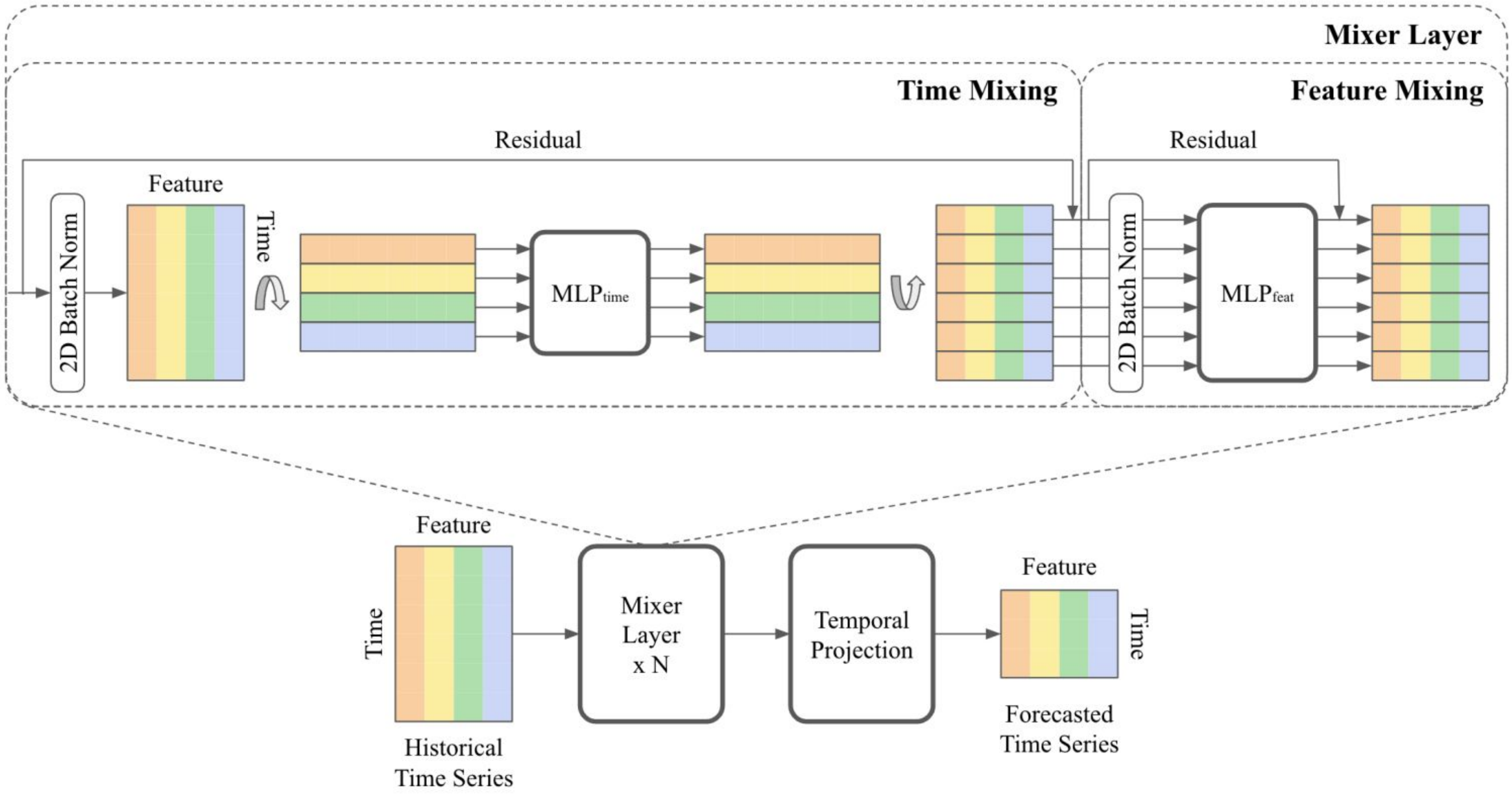
- **mixing** stage, performs actual learning
- remember that:

mixing layer = temporal mixing + feature mixing

- **first** mix to combine concatenated features
- proceeds with regular mixing layers
- static variables are added each time, since they are always present
- output number of features in a layer can be **smaller** to avoid overfitting



TSMixer - recap



TSMixer - pros and cons

Pros:

- simple
- can model very complex relations
- uni- and multivariate
- exogenous variables support

Cons:

- can overfit with too little data
- computational cost (but not too high)
- not interpretable

TSMixer - warning!

- there are **two papers** with name "TSMixer"
- we talked about the one by Google!
- but there is also a one by IBM:

  "TSMixer: Lightweight MLP-Mixer Model for Multivariate Time Series Forecasting" V. Ekambaram et al.

- Google one is better known, much simpler, people generally mean that one

Other interesting MLP-based models

- TimeMixer:

  ["TimeMixer: Decomposable Multiscale Mixing for Time Series Forecasting" S. Wang et al.]()

  - time series downsampling & multiscale structure
  - differentiable trend-seasonality decomposition (borrowed from Autoformer)
  - many different mixings
- TiDE:

  ["Long-term Forecasting with TiDE: Time-series Dense Encoder" A. Das et al.]()

  - MLP-based encoder-decoder
  - flexible: univariate, multivariate, with exogenous variables
  - quite small and very fast

Other mixing architectures

- time series:
  - TimeMixer
  - Tiny Time Mixers (TTMs)
  - U-Mixer
- computer vision:

  "MLP-Mixer: An all-MLP Architecture for Vision" I. Tolstikhin et al.

  "Patches Are All You Need?" A. Trockman, J. Kolter - ConvMixer
- graphs:

  "A Generalization of ViT/MLP-Mixer to Graphs" X. He et al.
- NLP:

  "pNLP-Mixer: an Efficient all-MLP Architecture for Language" F. Fusco et al.

Transformers

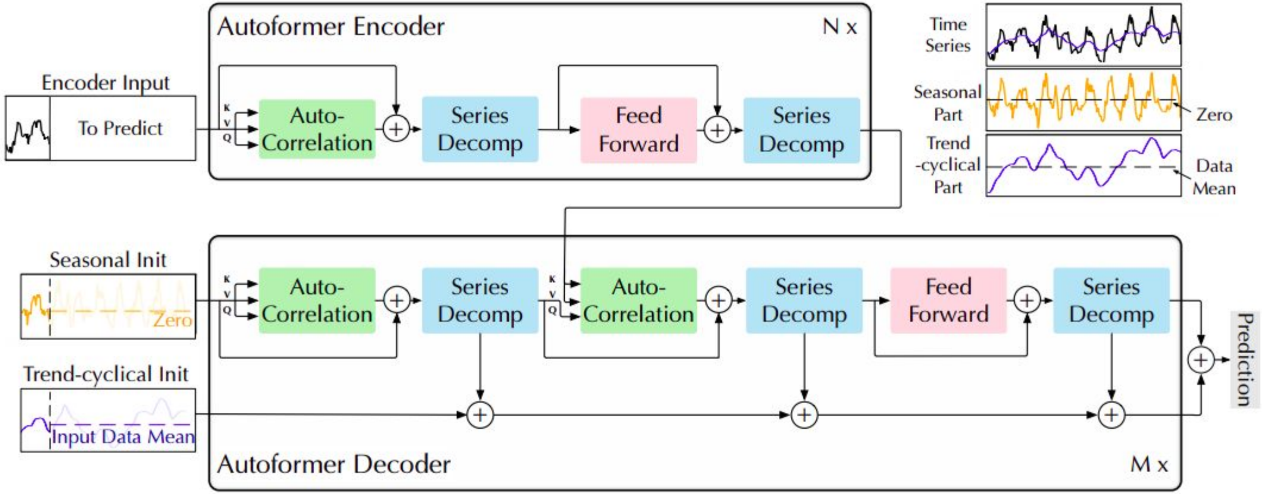
Time series transformers

- NLP-inspired transformer, but with modifications for time series
- often based on quite **complex attention variants**, especially hierarchical ones
- reduces cost and better learns time series information
- **quite varied:**
  - architecture: encoder-decoder / encoder-only
  - dimensionality: univariate / multivariate
  - pretraining: pretrained / trained from scratch (more frequent)

- foundational models are also based on transformers, but have visibly different trends in architecture - see further slides

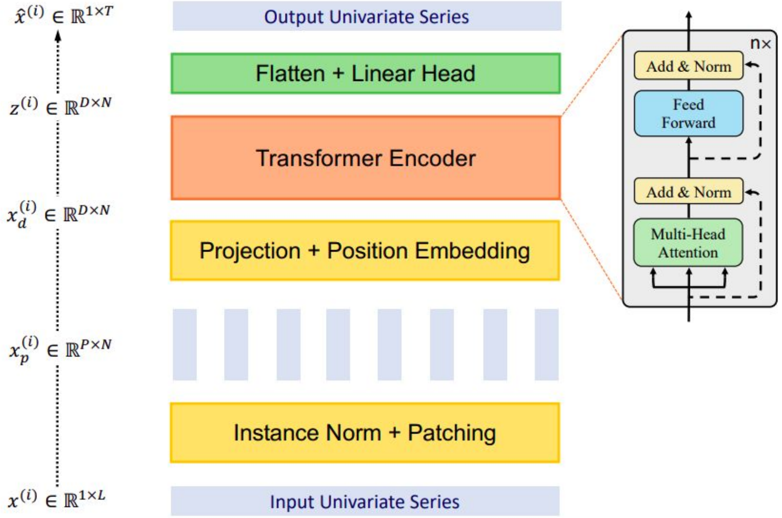
Encoder-decoder:

Autoformer



Encoder-only:

PatchTST



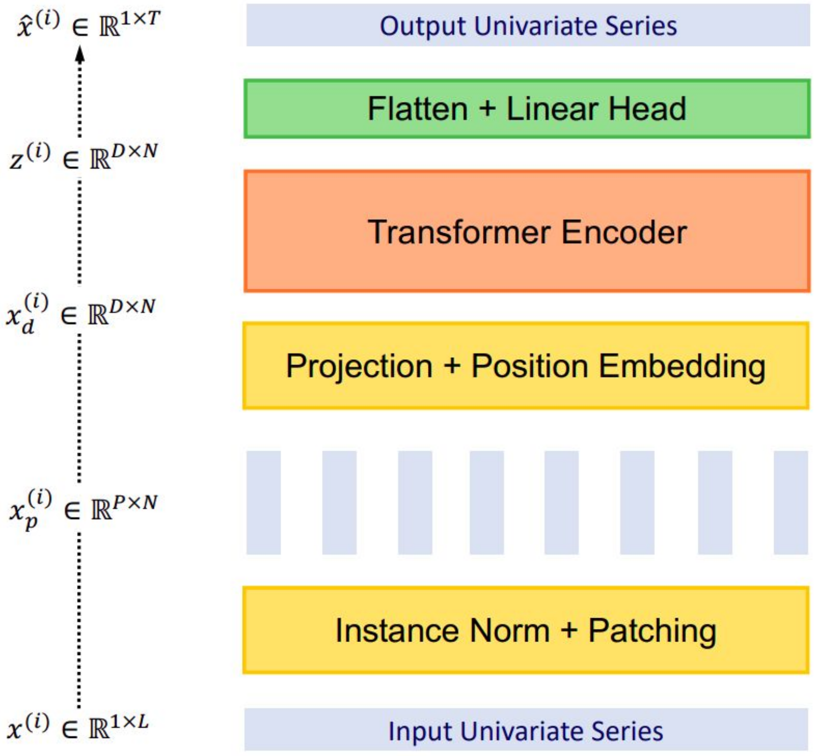
A bunch of transformers

Model	Year	Architecture	Univariate / multivariate	Pretrained	Innovation
LogTrans	2019	enc-dec	uni	no	LogSparse attention
Autoformer	2021	enc-dec	uni	no	Learn trend-seasonality and autocorrelation
Informer	2021	enc-dec	multi	no	Conv. subsampling, ProbSparse attention
FEDformer	2022	enc-dec	multi	no	Frequency enhanced attention
Pyraformer	2022	2 variants	uni	2 variants	Pyramidal attention with subsampling
NonStationary	2022	enc-dec	uni	no	Stationarization, De-Stationary attention
Crossformer	2023	enc-dec	multi	no	Hierarchical learning
PatchTST	2023	encoder-only	uni	yes	Patching, weight sharing, pretraining
iTransformer	2024	encoder-only	multi	no	Inverted tokenization (series-level)

PatchTST

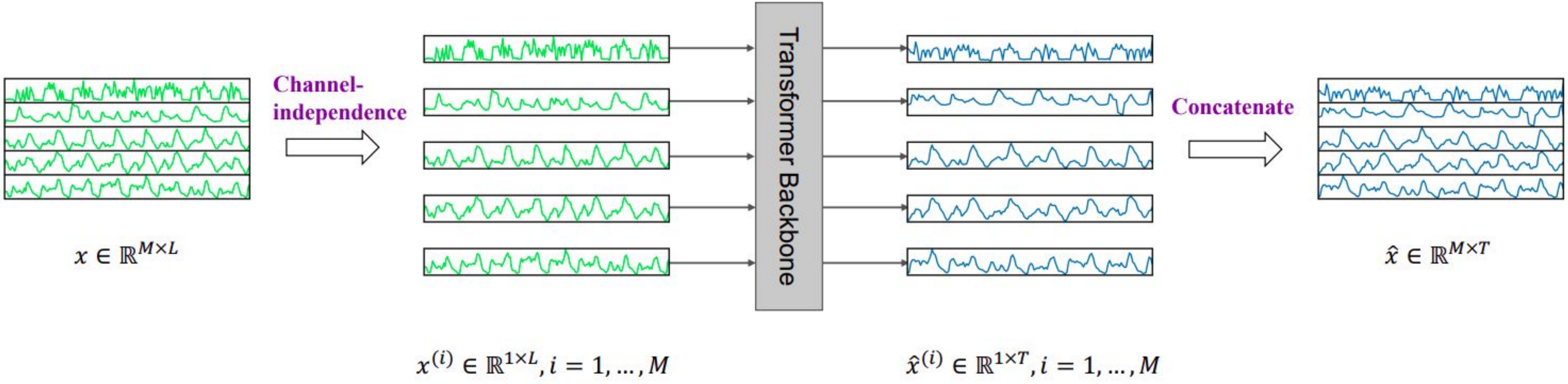
["A Time Series is Worth 64 Words: Long-term Forecasting with Transformers" Y. Nie et al.]

- **encoder-only** transformer, very similar to BERT
- main idea is **patching** - tokenizing time series as "patches" of values
- after this, token sequence is just like a text sentence
- can pretrain with **masked modeling**, simply by masking and reconstructing patches
- uses previous 64 patches, which are equivalent to lookback `L=512`



PatchTST - channel independence

- univariate, processes time series separately - called **channel independence** in the paper
- but with **weight sharing** - all series have the same transformer!
  - regularizes, encourages better generalization
  - isolates effects from noisy channels
  - allows flexible number of time series

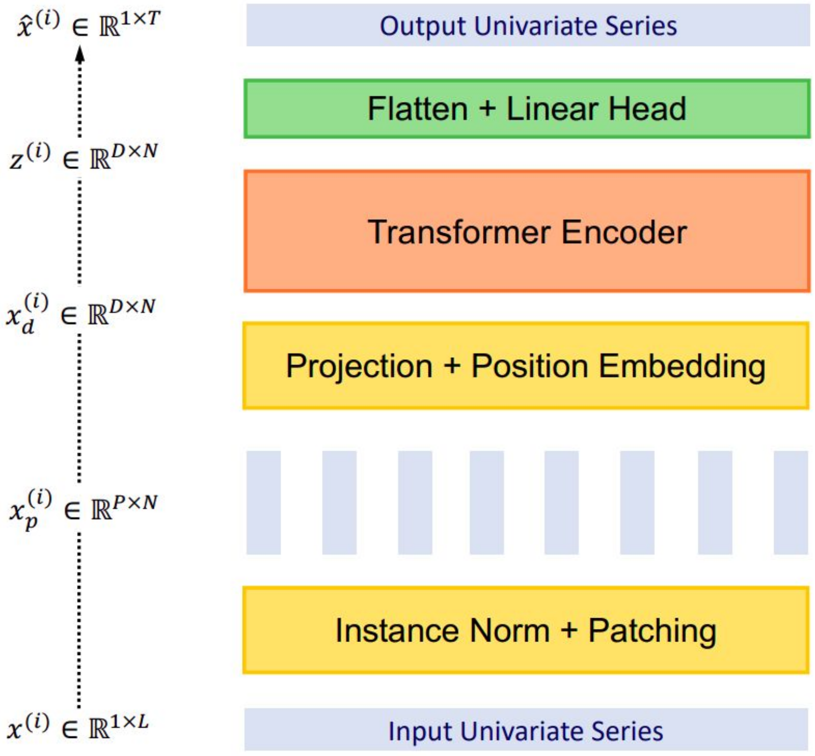


PatchTST - instance norm

- **instance normalization**
- time series standardization, before patching

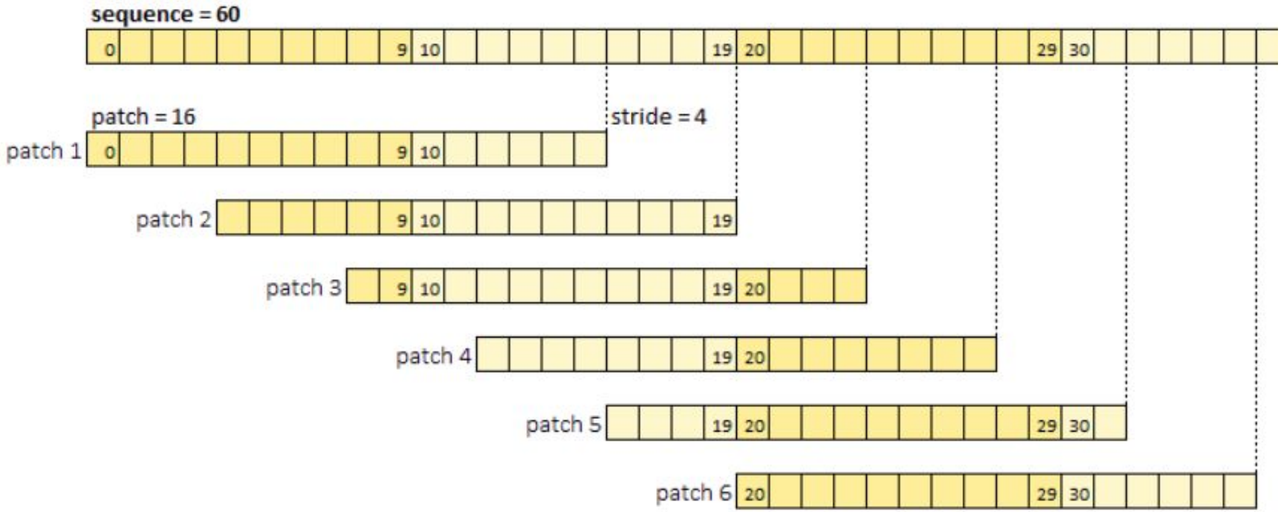
$$x' = \frac{x - \mu}{\sigma}$$

- processes time series separately
- reduces train/test **distribution shift**, making values distributions more similar
- added back to the final forecast



PatchTST - patching

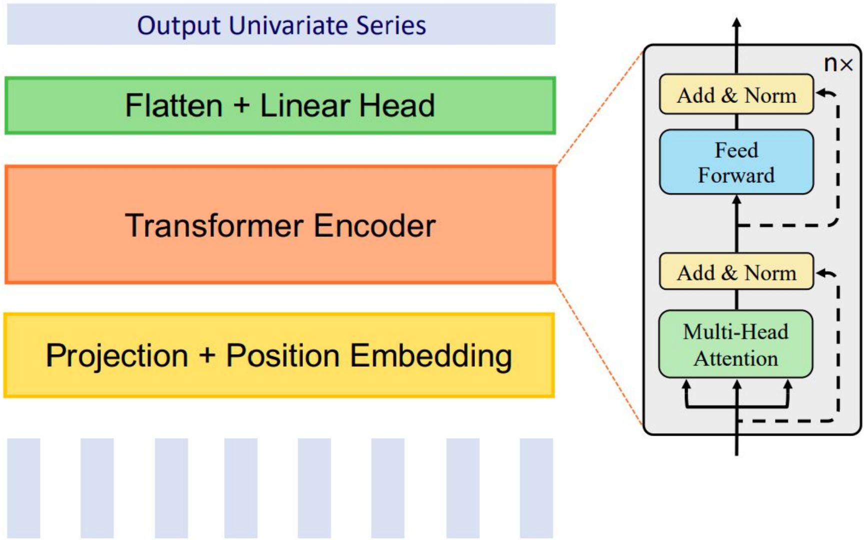
- **patch** is a continuous series of values, creating a **token** for transformer layers
- greatly reduces complexity (L → L/S tokens) and allows longer lookback
- original paper models: patch length `P=16`, stride `S=8`
- **overlap** is similar to CNNs, slightly reduces overfitting due to shared data



https://www.signalpop.com/2023/11/06/understanding-the-patchtst-model-for-time-series-prediction/

PatchTST - transformer

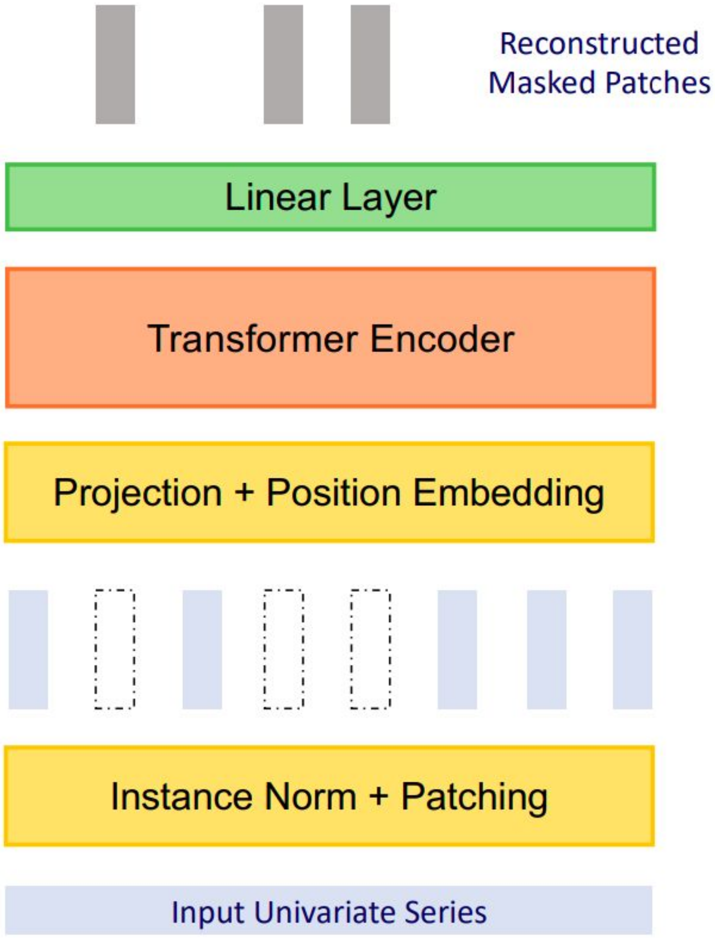
- similar to regular transformer layers, with a few changes
- **position embedding** encodes time and order
- multi-head self attention and 2-layer MLP use **GELU** activation
- **batch norm** instead of layer norm inside, it works better for time series
- paper parameters:
    - 3 layers
    - 16 attention heads
    - latent dimensionality D=128
    - MLP uses 128 → 256 → 128 dimensions



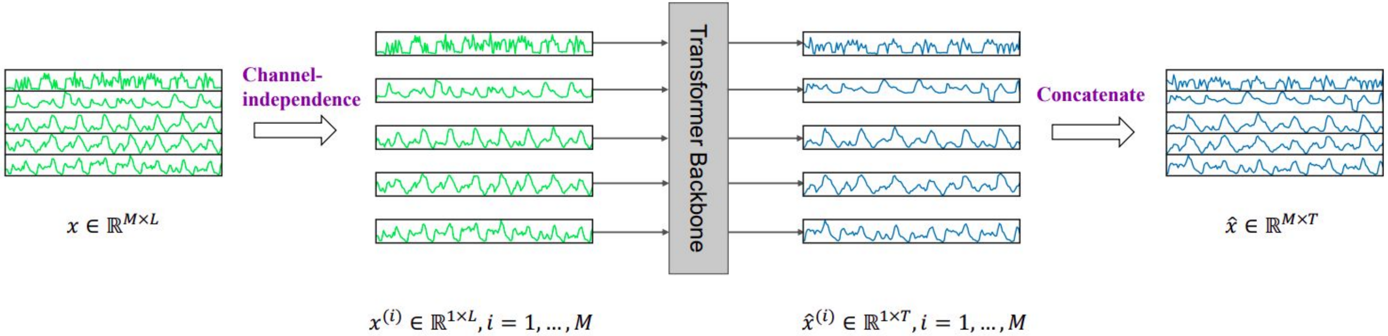
PatchTST - pretraining

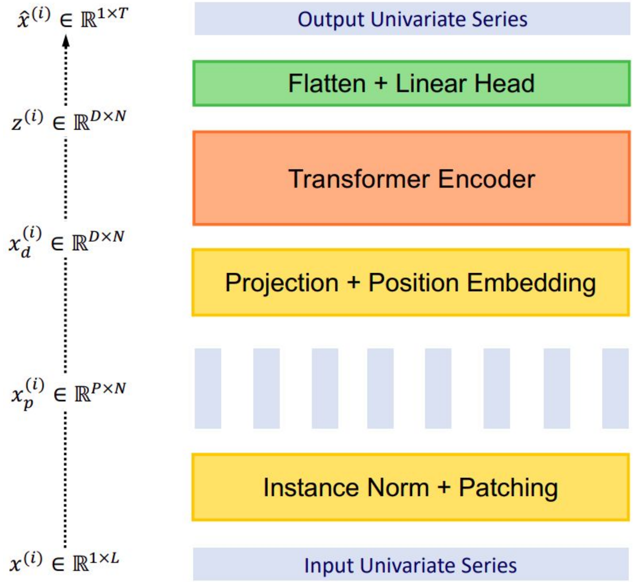
- change head to **masked modeling**
- mask patches randomly and learn to reconstruct them, minimizing MSE
- non-overlapping patches to avoid data leakage, i.e. `P=S=16`

- typically results in better quality (not always!)
- but finetuning is always much faster than training from scratch
- can also finetune just the classification head, known as **linear probing**



PatchTST - recap





PatchTST - pros and cons

Pros:

- simple and fast (for a transformer)
- pretraining
- uni- and multivariate

Cons:

- easily overfits with too little data (but: linear probing)
- hard to pretrain effectively
- not interpretable
- no exogenous variables

Pretrained foundation models

Foundation models

- foundation model (Wikipedia):

"Model that is trained on broad data such that it can be applied across a wide range of use cases"

- **size matters:** large models + massive and diverse datasets + lots of computational power
- based on **representation learning:**
    - creating neural networks encoding general-purpose knowledge
    - internally create useful input representation (at least we hope so)
    - pretrain "domain expert", which can perform well on new tasks
- unique capabilities:
    - **few-shot learning** - with extremely short finetuning
    - **zero-shot forecasting** - no additional training, just input new data and get output

Time series foundation models

- **common features:**
  - transformers
  - quite simple architectures
  - patching (tokenization)
  - pretraining on massive datasets
- **varied:**
  - architecture: encoder-only, enc-dec, decoder-only
  - uni- / multivariate
  - exogenous variables support
- **idea:** rely on data and simple learning, rather than complicated models and handcrafted modules
- novelty: decoder-only, generative pretraining
- we will see if they are worthwhile in the future, for now those are a **research direction**

Time series foundation models - caution

- **be very cautious** when checking those models
- often made by companies, to create "hype" around "ChatGPT for time series"
- frequently not fully open source (incl. data, code, model weights)
- massive data requirement has its own challenges:

  "Models work great when all test datasets are in your proprietary training dataset"
- whitepapers, preprints, technical notes etc. are **not peer-reviewed papers**

A bunch of time series foundation models

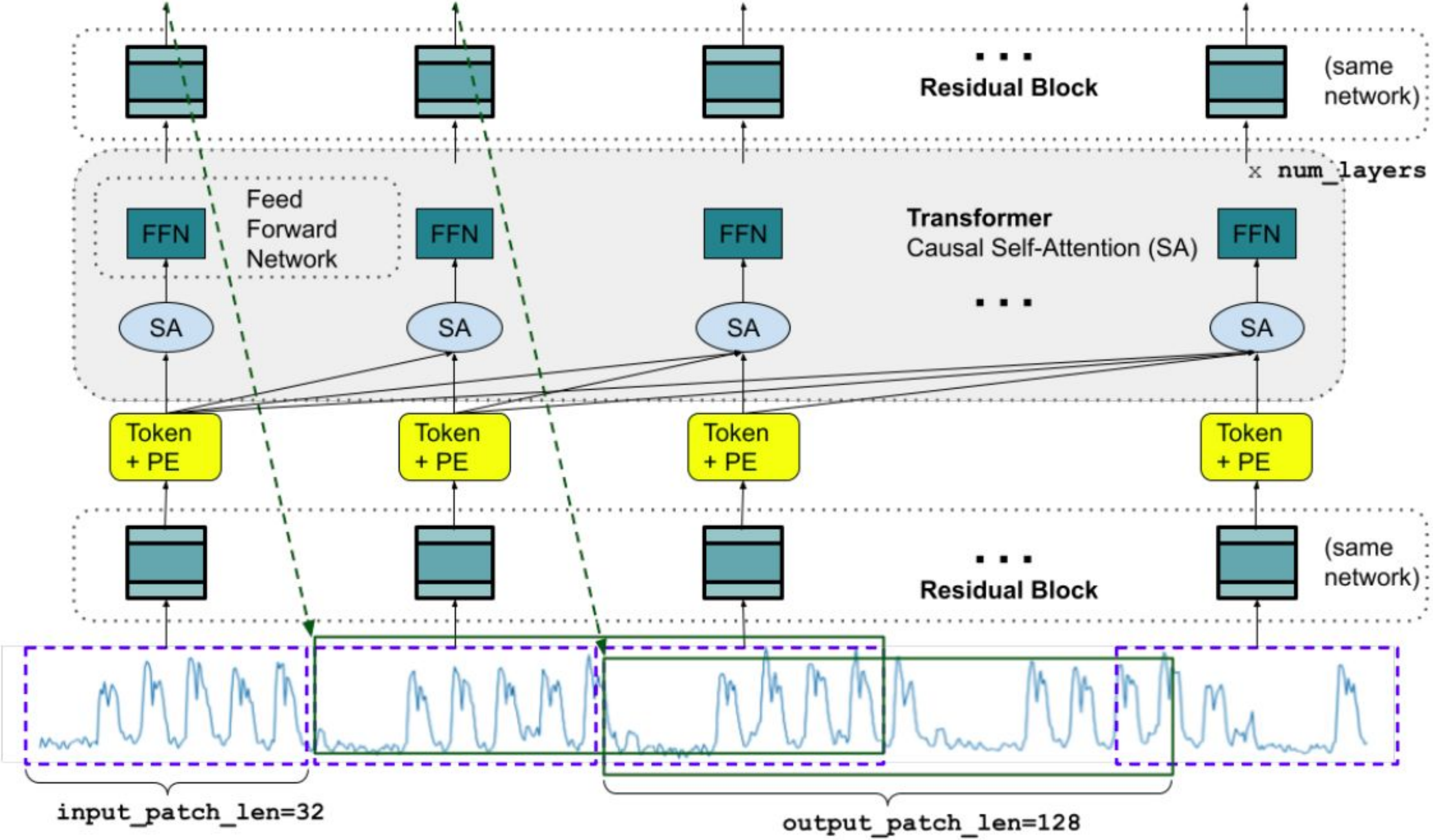
Model	Year	Company / university	Open source?	Published paper?	Architecture	Univariate / multivariate
TimeGPT	2023	Nixtla	no	no	enc-dec	uni
Lag-Llama	2023	Various (both)	yes	yes (NeurIPS workshop)	decoder	uni
TimesFM	2024	Google	yes	yes (ICML)	decoder	uni
Chronos	2024	Amazon	yes	no (TMLR reviews)	both	uni
Moirai	2024	Salesforce	yes	yes (ICML)	encoder	both
UniTS	2024	Harvard & MIT	yes	no	encoder	multi

TimesFM

"A decoder-only foundation model for time-series forecasting" A. Das et al.

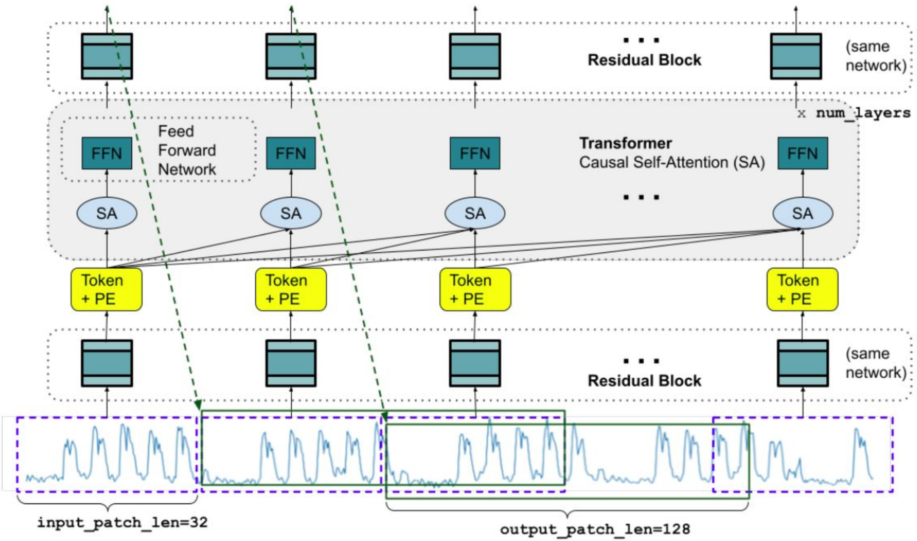
- Time series Foundation Model (TimesFM)
- **decoder-only** transformer, very similar to GPT
- **combination** of a few pretty simple ideas:
    - patching (tokenization)
    - decoder-only, generative pretraining
    - reasonable masking strategy
- in addition, they created a **massive pretraining dataset**, combining, e.g. Google Trends, Wikipedia page views, M4 datasets
- also used **synthetic**, generated datasets, exposing the model to different trends, shocks, seasonalities etc.

TimesFM - architecture



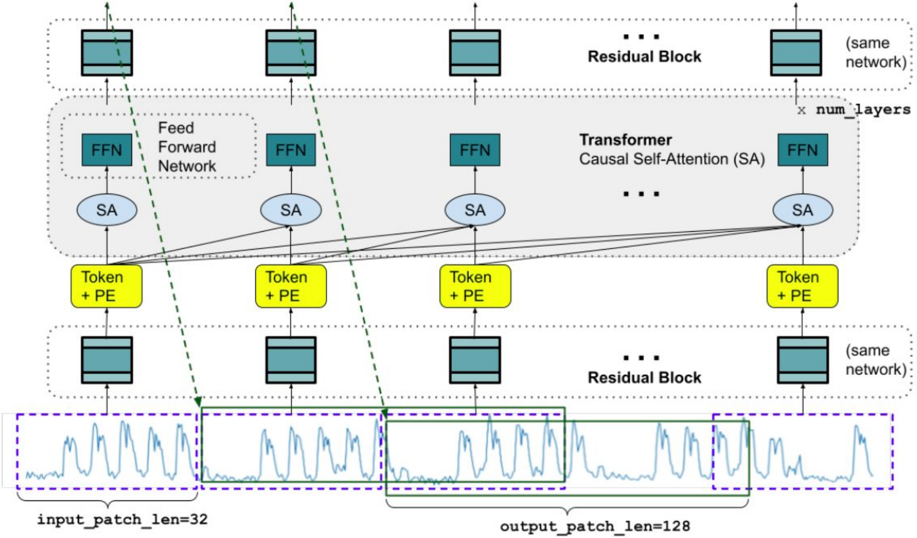
TimesFM - architecture

- basically, a standard **GPT transformer**
- **autoregressive** model that also makes direct multi-step (DMS) forecasts
- input patches are the "prompt"
- generates **long patches** at once, much longer than inputs
- this greatly reduces autoregressive error accumulation



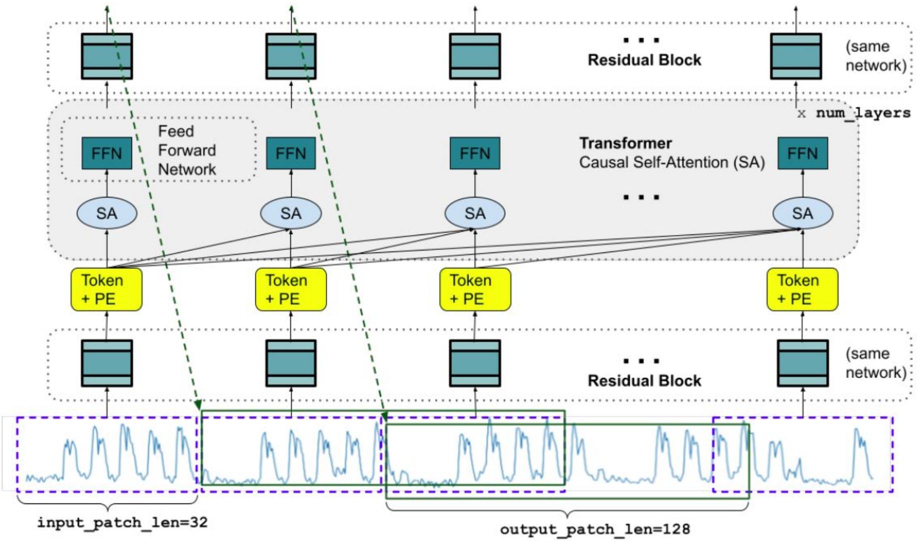
TimesFM - inputs and outputs

- "prompt" = **context**
- forecast = generated "words"
- sequence of **non-overlapping** patches
- input can have variable length, e.g. 32, 64, …, padded when necessary
- **published model:**
  - input patch length 32
  - max context length 512 (16 tokens)
  - output patch length 128
- autoregressive forecast makes "full" steps, i.e. 128 here



TimesFM - transformer

- quite standard transformer
- causal attention with masking
- published model is quite **wide & deep:**
  - 20 layers
  - 16 attention heads
  - hidden size 1280 in all layers
  - 200M parameters in total
- generates 128-element vectors of floats



TimesFM - training

- just a regular training, with one detail - **patch masking**
- **problem:** for naive patches, model might learn to predict well only for context that is multiple of input patch length (e.g. 16, 32, 64, …)
- **patch masking:**
  - for each time series in a batch, get a random number r from $[0, p-1]$
  - mask first r elements of the first patch, reducing the context
  - do this enough times and model will see all possibilities
- example:
  - $p=32$, $r=4$
  - first context is 28 (32-4), second is 60 (28+32), and so on

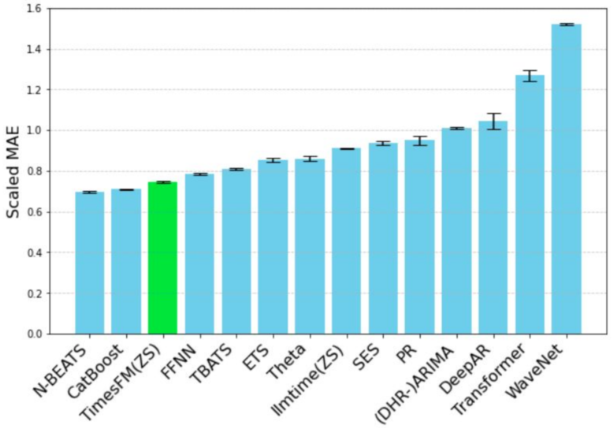
TimesFM - pretraining

- trained on a **mixture** of data
- a lot of **synthetic** data (20%):
  - trends, seasonalities
  - processes, e.g. ARMA
- real data (80%) chosen to give equal weights to different frequencies
- context depended on frequency:
  - 512 where possible
  - 256 for weekly
  - 64 for >= monthly

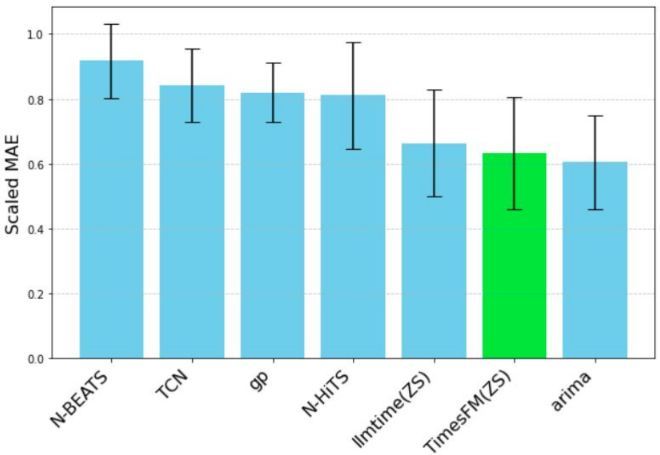
Table 1: Composition of TimesFM pretraining dataset.

Dataset	Granularity	# Time series	# Time points
Synthetic		3,000,000	6,144,000,000
Electricity	Hourly	321	8,443,584
Traffic	Hourly	862	15,122,928
Weather [ZZP+21]	10 Min	42	2,213,232
Favorita Sales	Daily	111,840	139,179,538
LibCity [WJJ+23]	15 Min	6,159	34,253,622
M4 hourly	Hourly	414	353,500
M4 daily	Daily	4,227	9,964,658
M4 monthly	Monthly	48,000	10,382,411
M4 quarterly	Quarterly	24,000	2,214,108
M4 yearly	Yearly	22,739	840,644
Wiki hourly	Hourly	5,608,693	239,110,787,496
Wiki daily	Daily	68,448,204	115,143,501,240
Wiki weekly	Weekly	66,579,850	16,414,251,948
Wiki monthly	Monthly	63,151,306	3,789,760,907
Trends hourly	Hourly	22,435	393,043,680
Trends daily	Daily	22,435	122,921,365
Trends weekly	Weekly	22,435	16,585,438
Trends monthly	Monthly	22,435	3,821,760

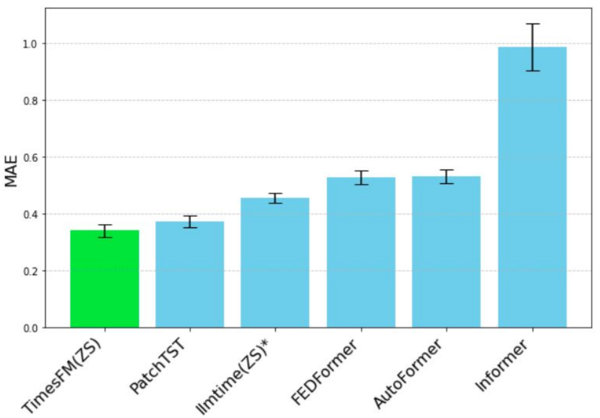
TimesFM - zero-shot results



(a) Monash Archive (Godahewa et al., 2021)



(b) Darts (Herzen et al., 2022)



(c) ETT (Horizons 96 and 192) (Zhou et al., 2021)

TimesFM - pros and cons

Pros:

- simple, yet powerful
- good results
- pretraining on a lot of data
- few-shot and zero-shot capabilities

Cons:

- computational cost
- only univariate
- no exogenous variables
- not interpretable

Questions?