Time series analysis

Runpeng Dai

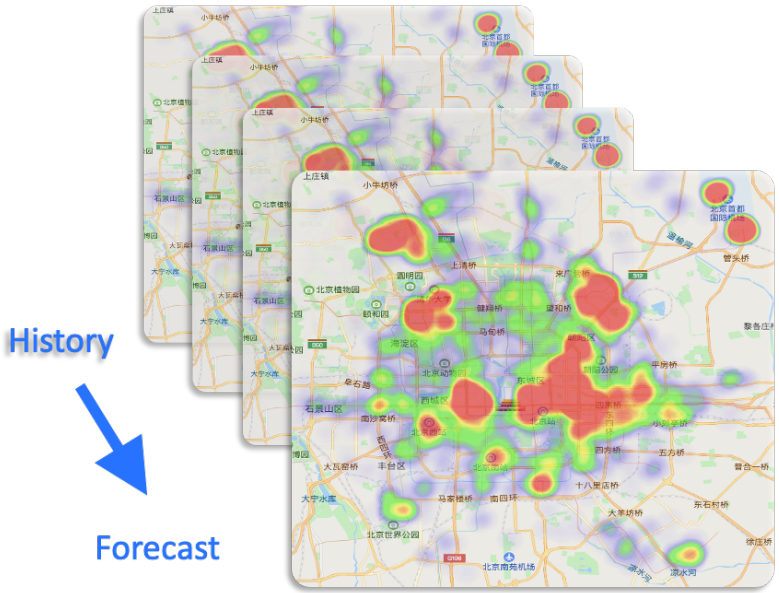
The University of North Carolina at Chapel Hill

June 16, 2024

Table of contents

Overview

- Time series / Spatial Temporal data analysis is a long-standing challenge in statistics society with various applications (e.g., energy, weather, traffic, economics).



If we "foresee" future orders:

- Get prepared for future peak hours.
- Dispatch vehicles for regions with higher demand.
- Detect abnormal road conditions.

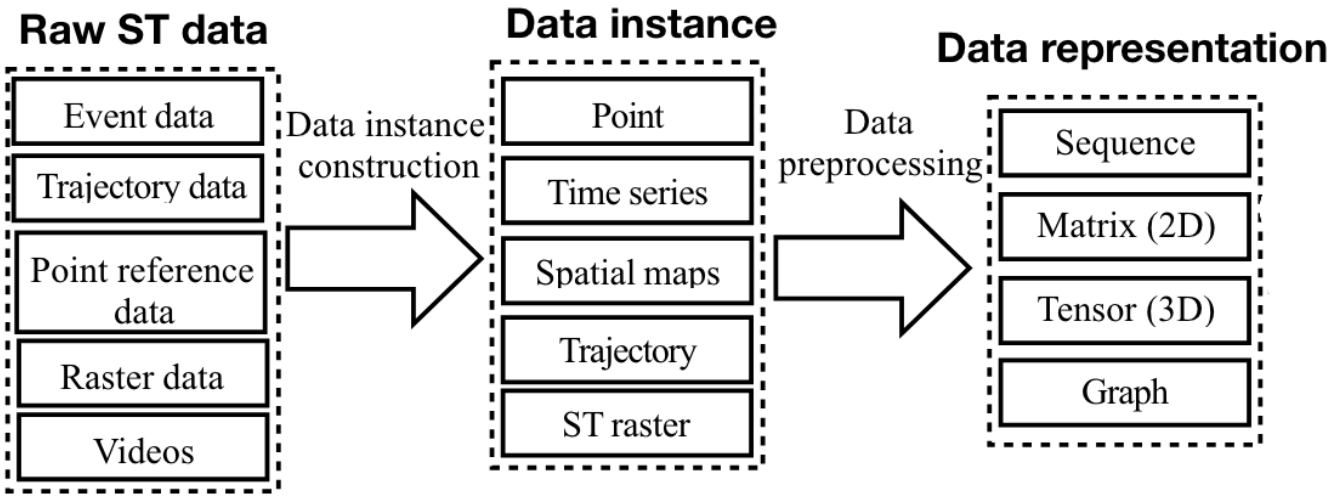
Demand forecasting "empowers" better strategies and services.

Overview-Challenges

- Traditional time series analysis methods, such as ARIMA and exponential smoothing, have been widely used but often fall short in capturing complex patterns and nonlinearities present in modern datasets.
- In the context of time series analysis, deep learning techniques offer several advantages:
  1. Automatic Feature Extraction: Deep learning models can automatically extract relevant features from raw time series data, reducing the need for manual feature engineering.
  2. Handling Nonlinearities: Neural networks excel at capturing nonlinear patterns that traditional methods might miss.
  3. Scalability: Deep learning models can handle large volumes of data and can be scaled to accommodate increasing data sizes.

Overview-Data and task

Given the diverse nature of tasks and data collection methods, real life time series analysis often involves a variety of data structures [WCP20].

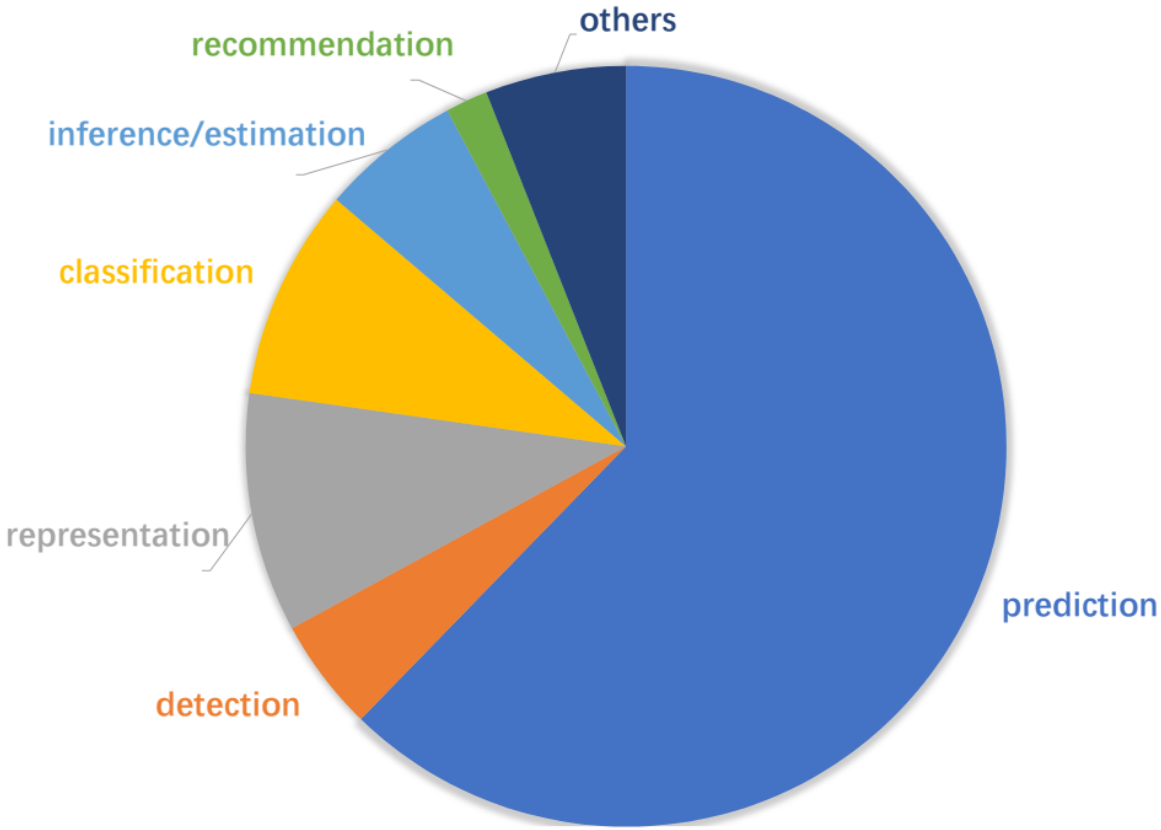


Overview-Data and task

- A ST point can be represented as a tuple containing the spatial and temporal information.
- While trajectories are series of points collected over time.
- Spatial maps contains the data observations at some locations of spatial filed at each time stamp.
- Spatial-Temporal raster contains the measurements spanning the entire set of locations and time stamps.

In the essence, all these data instances can be generalized to time-series data. For the sake of simplicity, we will focus on simple multivariate time series, namely we consider m time series $X_1(t), ..., X_m(t) \quad t \leq T$.

Overview-Data and task



More than 70% related deep learning papers focus on studying the time-series prediction problem due to the predictive capabilities of deep learning models and the broad applicability of prediction tasks. We focus on prediction problem in this section.

Overview-Outline

This section will explore the following topics:

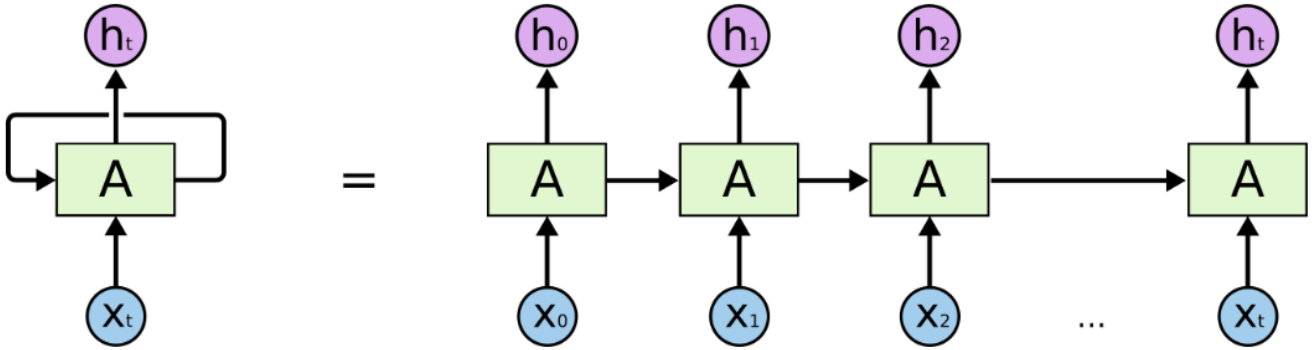
1. Recurrent neural network (RNN) based models.
2. Convolutional neural networks.
3. Transformer based models.
4. Large language model (LLM) for time series.
5. Application1: Origin-destination prediction.
6. Application2: Supply demand forecasting.

References I

[WCP20] Senzhang Wang, Jiannong Cao, and S Yu Philip. "Deep learning for spatio-temporal data mining: A survey". In: *IEEE transactions on knowledge and data engineering* 34.8 (2020), pp. 3681–3700.

Table of contents

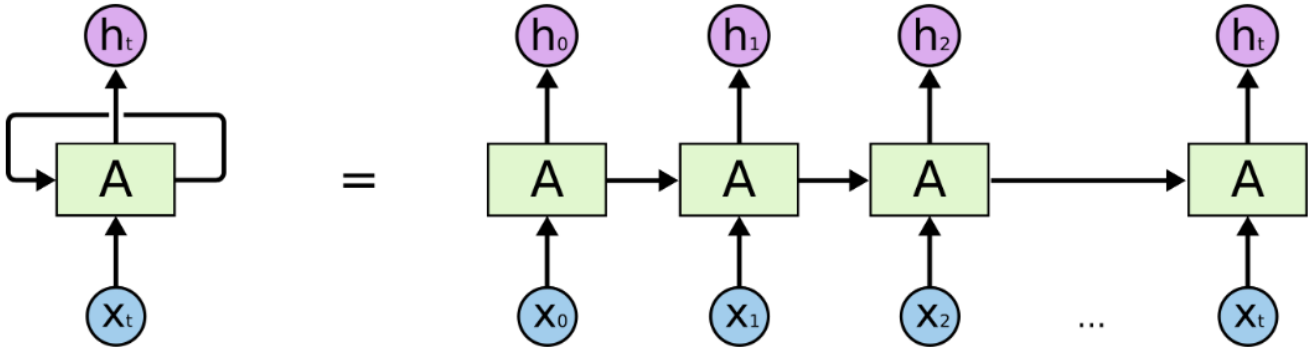
Overview



An unrolled recurrent neural network.

- Before the advent of transformers, sequence modeling was commonly associated with recurrent networks.
- RNNs are often preferred over fully connected neural networks for the following reasons:
  1. Memory consumption remains constant regardless of the history length.
  2. RNNs propagate information through time and generate outputs sequentially, treating historical and current data differently..

How recurrent networks work

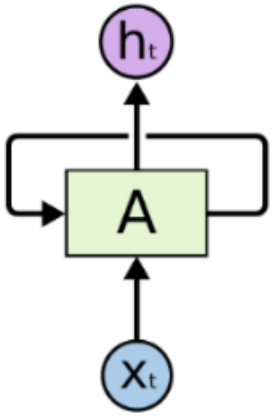


An unrolled recurrent neural network.

- Forward propagation: Begin with the initial hidden state (zero or random). Each unit takes the hidden state from the previous step and the current feature as inputs, outputting the hidden state for the current step.
- Loss function: Use a fully connected network on the final hidden state to project it to the desired future length dimension, then calculate the Mean Squared Error (MSE).

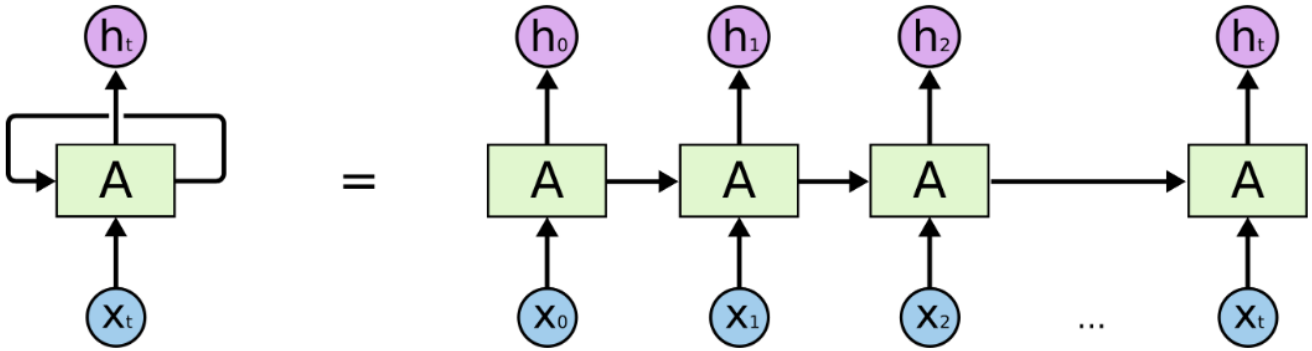
RNN has only one unit

```
# Efficient implementation equivalent to the following with bidirectional=False
def forward(x, h_0=None):
    if batch_first:
        x = x.transpose(0, 1)
    seq_len, batch_size, _ = x.size()
    if h_0 is None:
        h_0 = torch.zeros(num_layers, batch_size, hidden_size)
    h_t_minus_1 = h_0
    h_t = h_0
    output = []
    for t in range(seq_len):
        for layer in range(num_layers):
            h_t[layer] = torch.tanh(
                x[t] @ weight_ih[layer].T
                + bias_ih[layer]
                + h_t_minus_1[layer] @ weight_hh[layer].T
                + bias_hh[layer]
            )
        output.append(h_t[-1])
        h_t_minus_1 = h_t
    output = torch.stack(output)
    if batch_first:
        output = output.transpose(0, 1)
    return output, h_t
```



The description of sequence processing can sometimes be confusing. In reality, an RNN consists of a single unit that takes the value at the current time step and the hidden state from the previous time step as inputs, then outputs a new hidden state. This process loops until the end of the input sequence is reached.

Problem of naive RNN model

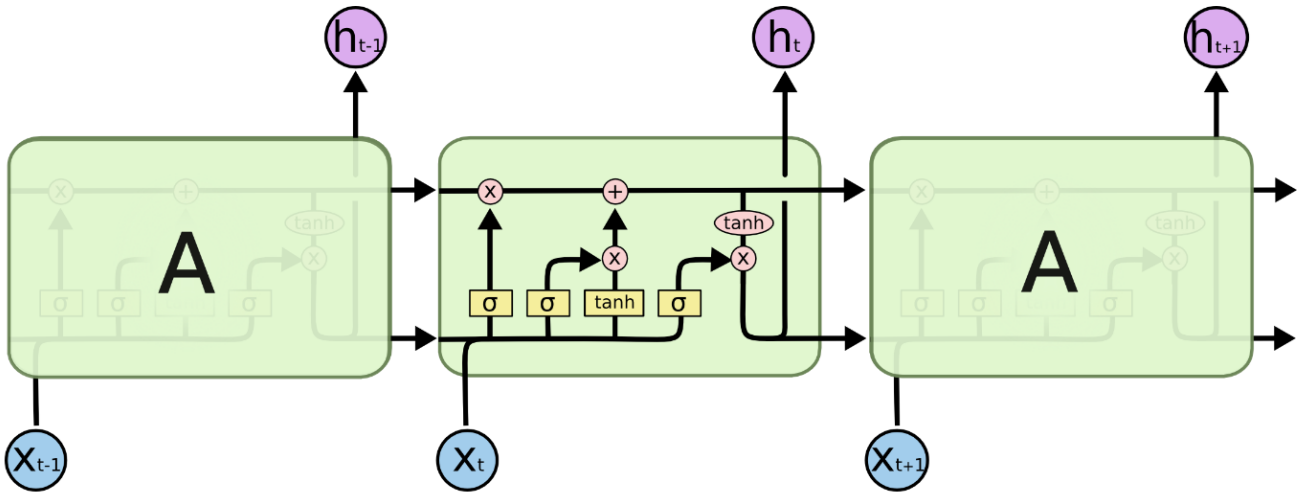


An unrolled recurrent neural network.

- Vanishing or Exploding Gradient:
- Limited Long-Term Memory: Despite theoretically having infinite memory, RNNs often suffer from forgetting issues.
- Lack of Parallelization: As a sequence model, RNNs can only generate hidden states one at a time.

LSTM

To address the issues of vanishing gradients and forgetting, LSTM (Long Short-Term Memory) networks were proposed and have been the dominant method in time series and language modeling for a considerable period[Col15].

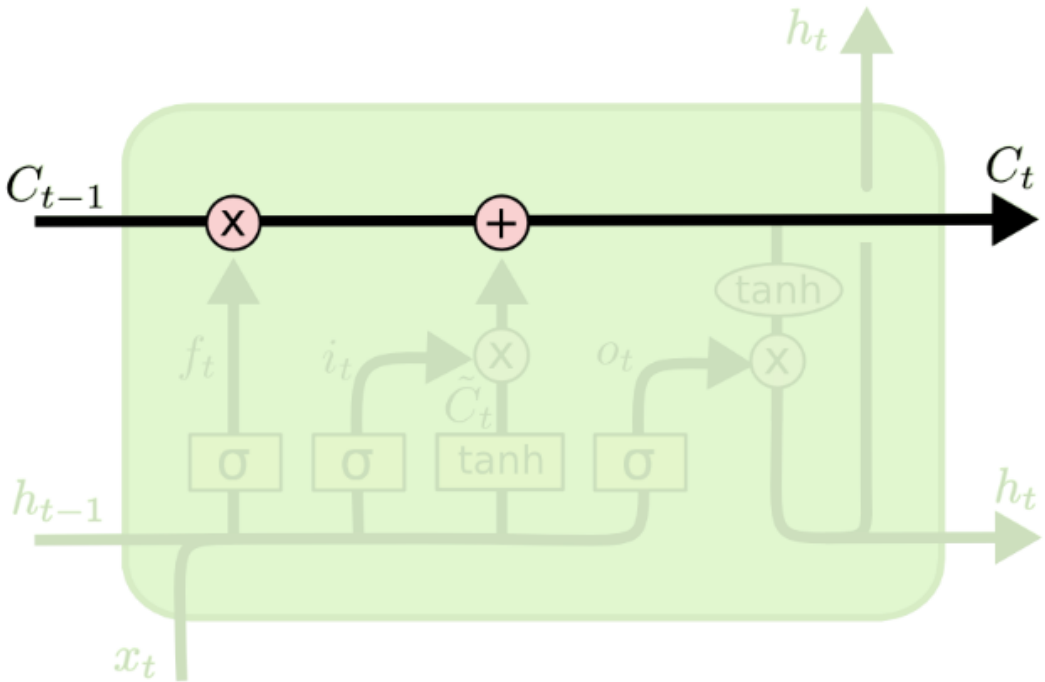


The repeating module in an LSTM contains four interacting layers.

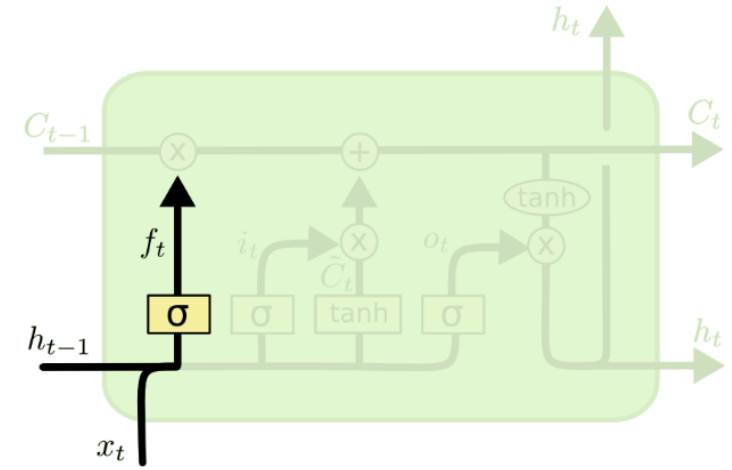
LSTMs have the same sequential structure as RNNs but incorporate more complex units.

LSTM-cell module

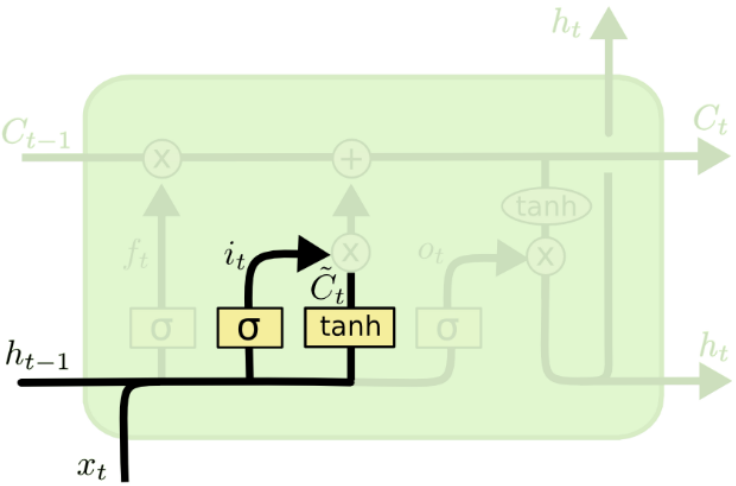
A key difference in LSTMs is the presence of a cell state. Each unit receives the hidden state and cell state from the previous step. Unlike the hidden state, which undergoes multiple layers of projection, the cell state only experiences linear transformations. This design helps alleviate the forgetting issue.



LSTM-message passing logic



Forget Gate: An element-wise multiplication with values ranging from zero to one, determining which parts of the cell state to forget.



Remember Gate: An element-wise multiplication with values ranging from zero to one applied to a non-linear transformation of the hidden state, determining which parts of the hidden state to add to the cell memory

LSTM-cell module

TABLE II: The RMSEs of ARIMA and LSTM models.

Stock	RMSE		% Reduction in RMSE
	ARIMA	LSTM	
N225	766.45	105.315	-86.259
IXIC	135.607	22.211	-83.621
HSI	1,306.954	141.686	-89.159
GSPC	55.3	7.814	-85.869
DJI-Monthly	516.979	77.643	84.981
DJI-Weekly	287.6	30.61	-89.356
Average	511.481	64.213	-87.445
MC	0.81	0.801	-1.111
HO	0.522	0.43	-17.624
ER	1.286	0.251	-80.482
FB	0.478	0.397	-16.945
MS	30.231	3.17	-89.514
TR	2.672	0.569	-78.705
Average	5.999	0.936	-84.394

Figure: An Empirical comparison of ARIMA and LSTM on time series prediction[STN18].

References I

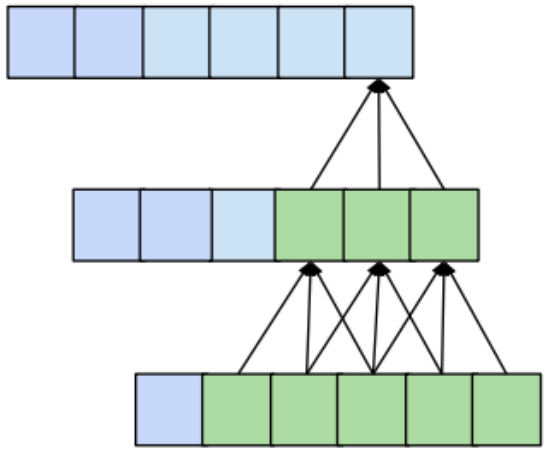
[Col15] Colah. *Understanding LSTM Networks*. 2015. URL: https://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[STN18] Sima Siami-Namini, Neda Tavakoli, and Akbar Siami Namin. "A comparison of ARIMA and LSTM in forecasting time series". In: *2018 17th IEEE international conference on machine learning and applications (ICMLA)*. IEEE. 2018, pp. 1394–1401.

Table of contents

Naive CNN model

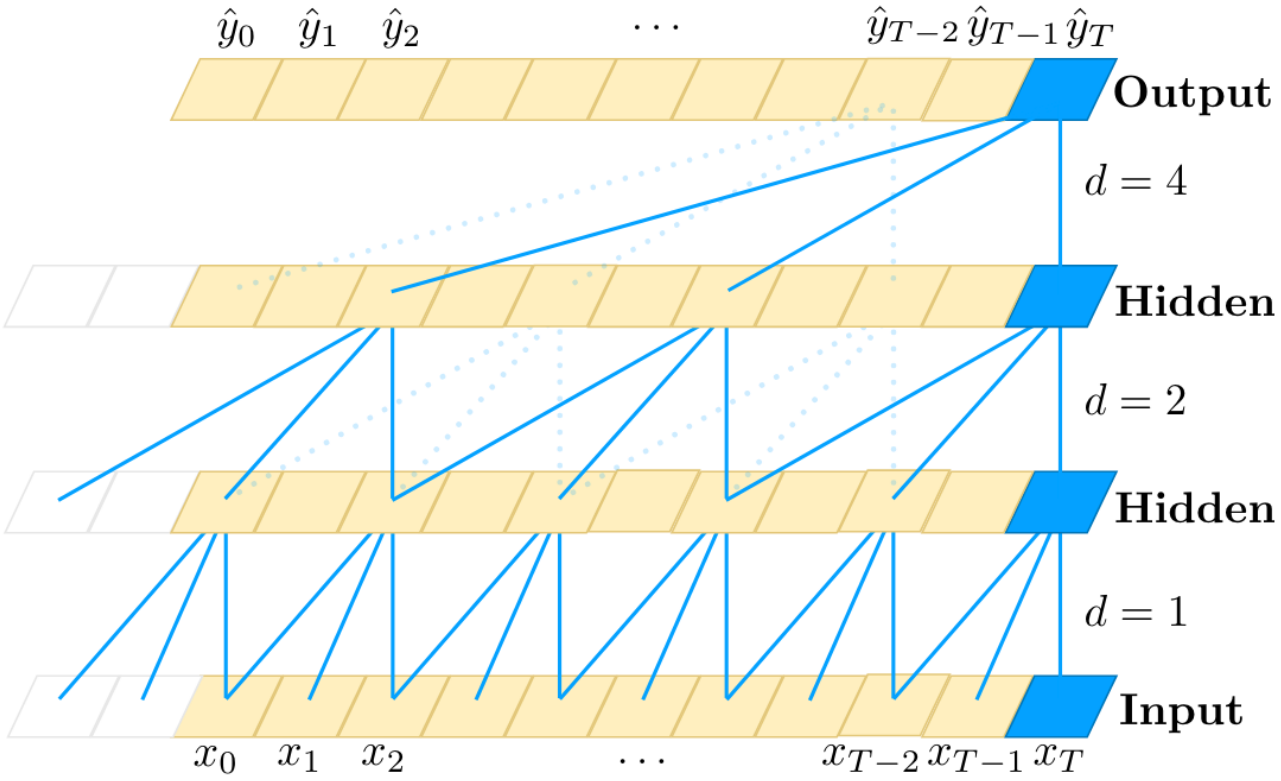
Before the development of Temporal Considering the success of Convolutional Neural Networks (CNNs) in the field of computer vision, there were some early attempts to apply convolutional layers to time series data before the development of Temporal Convolutional Networks (TCNs).



This approach had a significant drawback: to achieve a receptive field that covers all historical data, the model depth had to increase linearly with the time scale of the input.

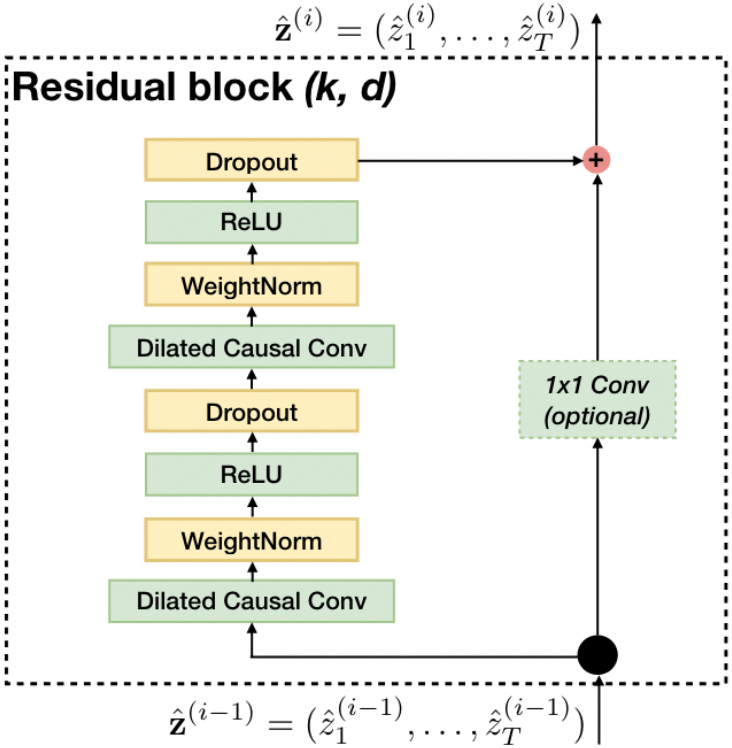
TCN

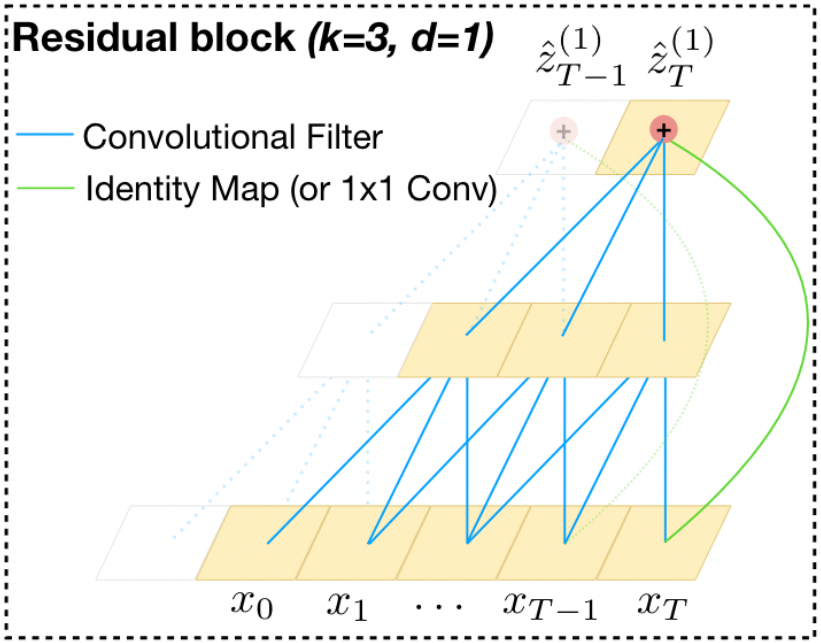
- TCNs [BKK18; Fra21] utilize a dilated causal convolution structure, skipping certain connections within the network.
- This approach allows TCNs to achieve a model depth that scales logarithmically with the time scale, while still ensuring a receptive field that encompasses all historical data.



TCN

To ensure model stability and prevent overfitting, residual blocks, along with non-linear activation and normalization layers, are introduced in addition to the convolutional layers.





References I

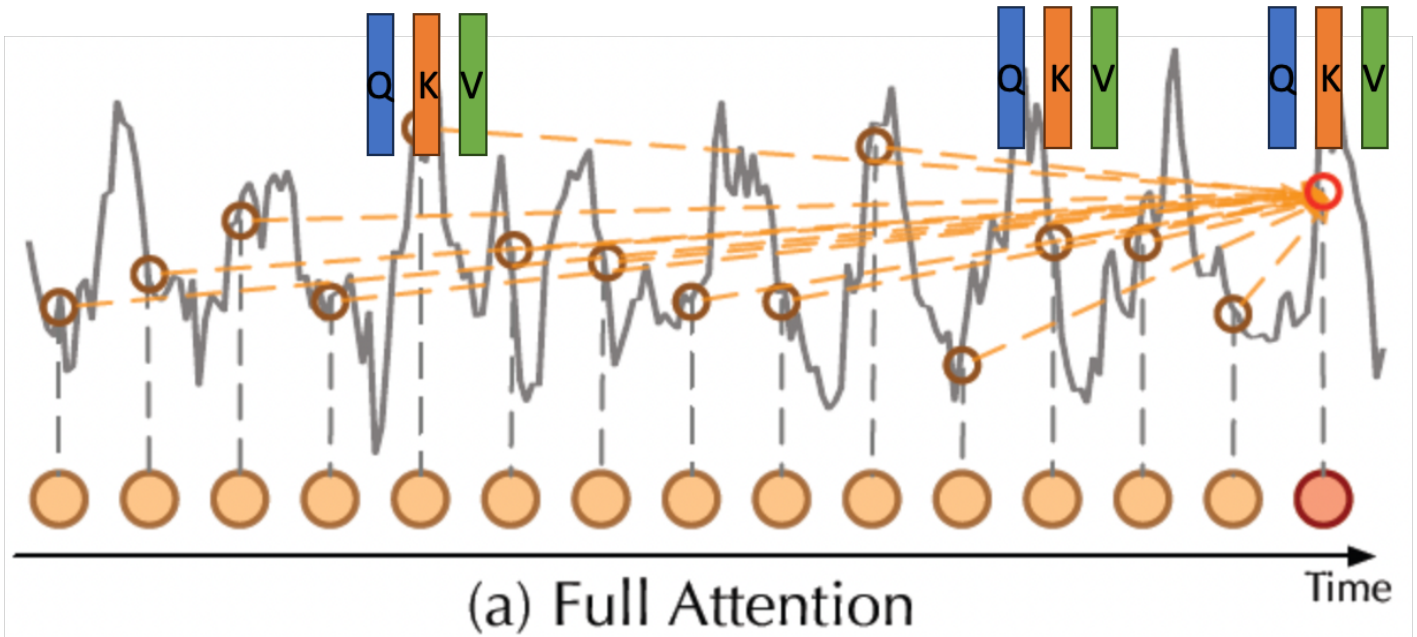
[BKK18] Shaojie Bai, J Zico Kolter, and Vladlen Koltun. "An empirical evaluation of generic convolutional and recurrent networks for sequence modeling". In: *arXiv preprint arXiv:1803.01271* (2018).

[Fra21] Francesco. *Temporal Convolutional Networks and Forecasting*. 2021. URL: https://unit8.com/resources/temporal-convolutional-networks-and-forecasting/.

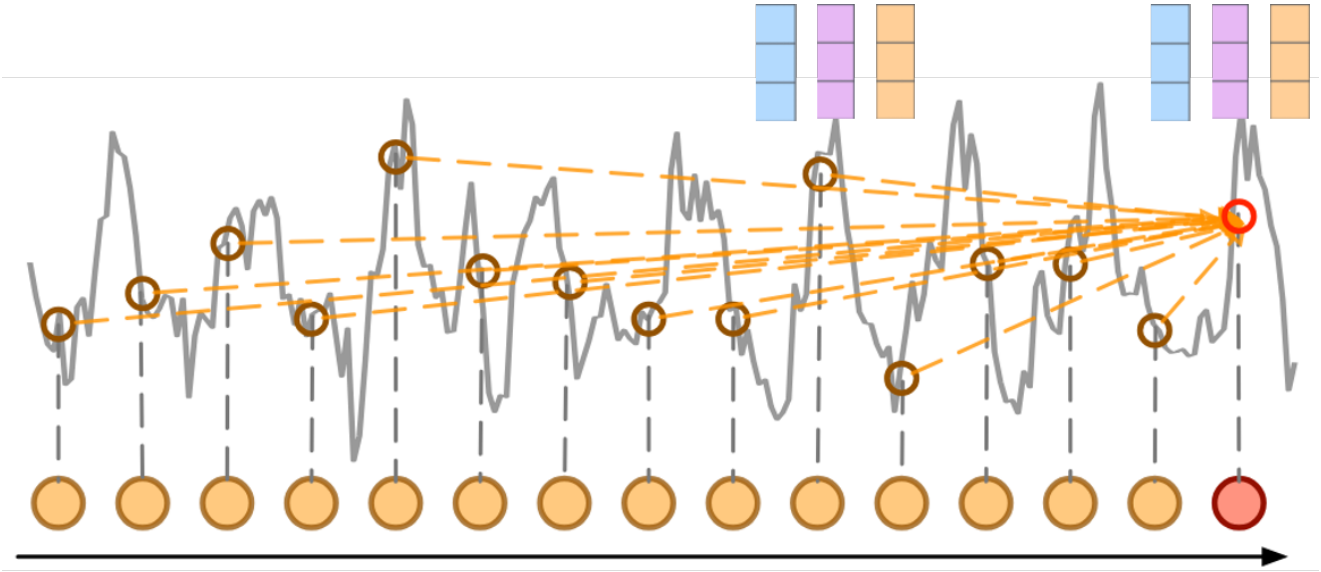
Table of contents

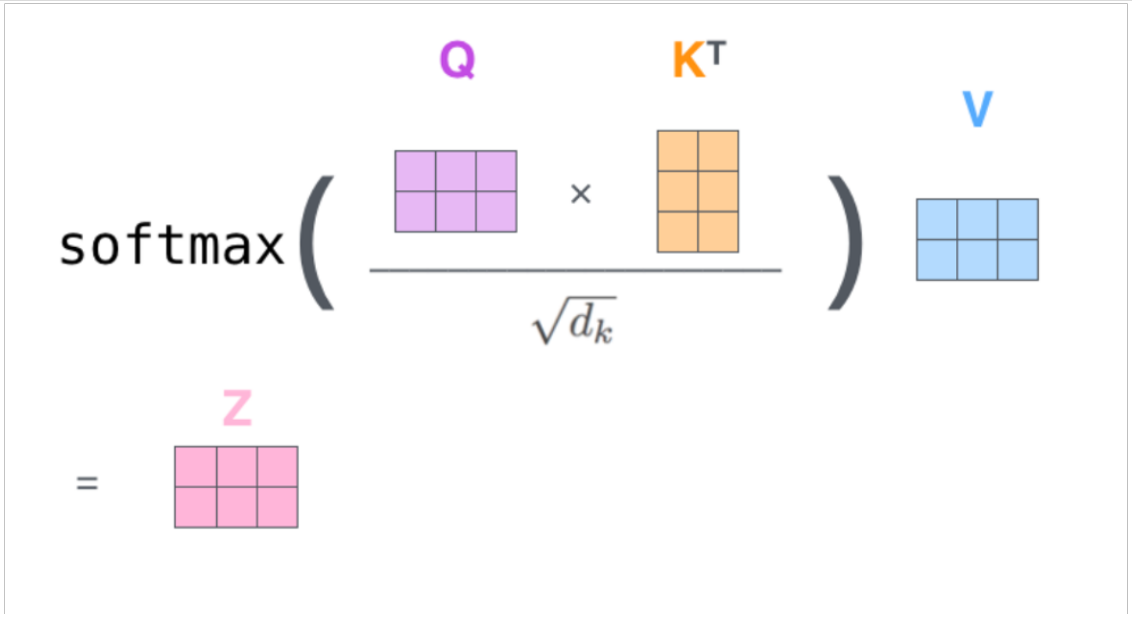
Vanilla time transformer

A key difference among time series models is how they handle long-term dependencies. RNNs use a Markov chain-like structure, while TCNs employ a layered structure.
Transformers[Vas+17; Nie22], on the other hand, use self-attention to capture dependencies over time, which alleviates forgetting issue.



Attention under the hood





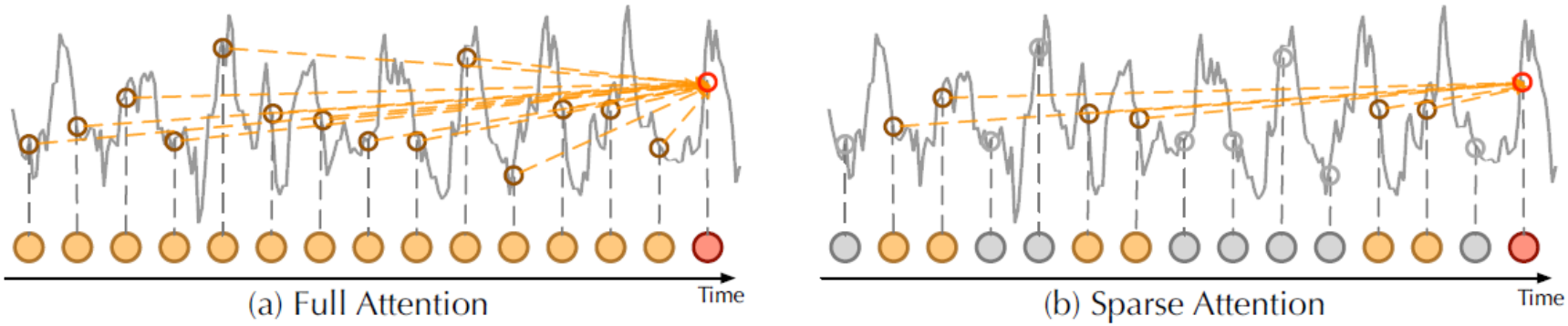
Computation concern

- Attention models are well-optimized for parallel computing.
- For an input length of L, the model size of a Transformer does not scale proportionally with L as it does in RNNs.
- However, the computational cost and memory consumption are L^2 due to the pairwise inner product calculations.

Many work improve over vanilla transformer model to save memory and computation, see [Wen+22].

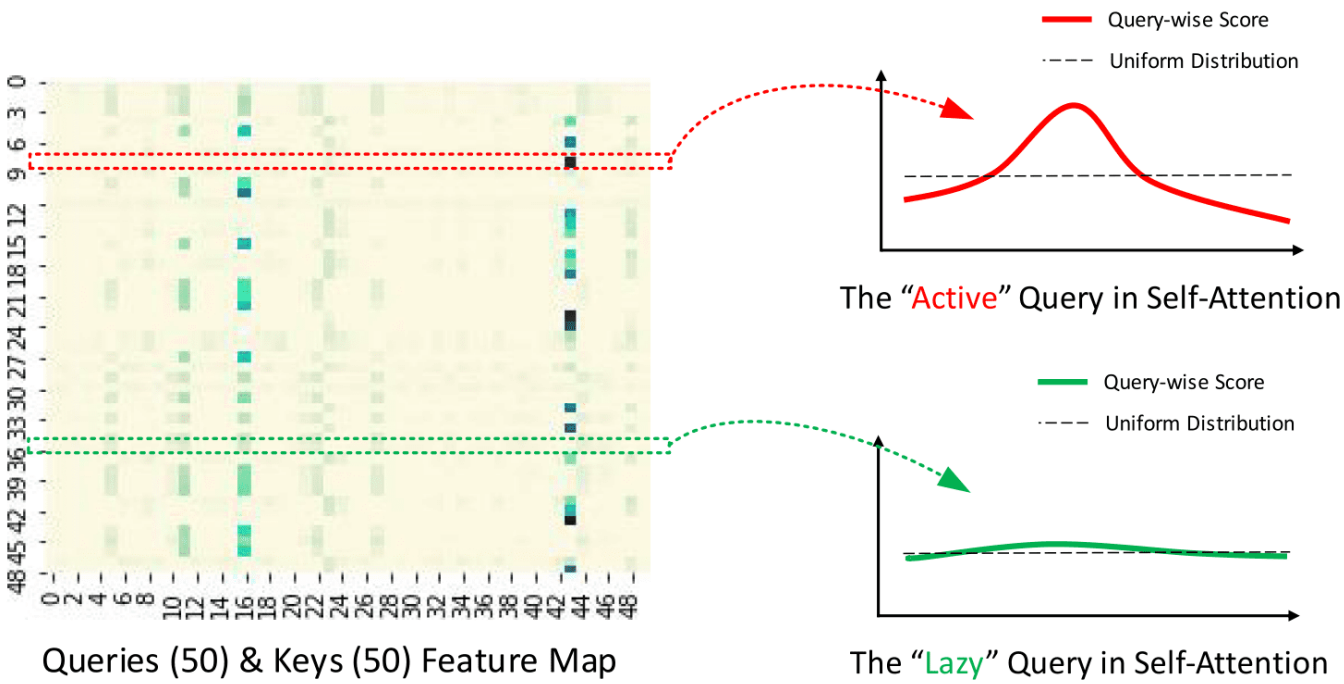
Methods	Training		Testing
	Time	Memory	Steps
Transformer [Vaswani *et al.*, 2017]	$\mathcal{O}\left(N^2\right)$	$\mathcal{O}\left(N^2\right)$	N
LogTrans [Li *et al.*, 2019]	$\mathcal{O}(N \log N)$	$\mathcal{O}\left(N \log N\right)$	1
Informer [Zhou *et al.*, 2021]	$\mathcal{O}(N \log N)$	$\mathcal{O}(N \log N)$	1
Autoformer [Wu *et al.*, 2021]	$\mathcal{O}(N \log N)$	$\mathcal{O}(N \log N)$	1
Pyraformer [Liu *et al.*, 2022a]	$\mathcal{O}(N)$	$\mathcal{O}(N)$	1
Quatformer [Chen *et al.*, 2022]	$\mathcal{O}(2cN)$	$\mathcal{O}(2cN)$	1
FEDformer [Zhou *et al.*, 2022]	$\mathcal{O}(N)$	$\mathcal{O}(N)$	1
Crossformer [Zhang and Yan, 2023]	$\mathcal{O}(\frac{D}{L_{seg}^2} N^2)$	$\mathcal{O}(N)$	1

Informer



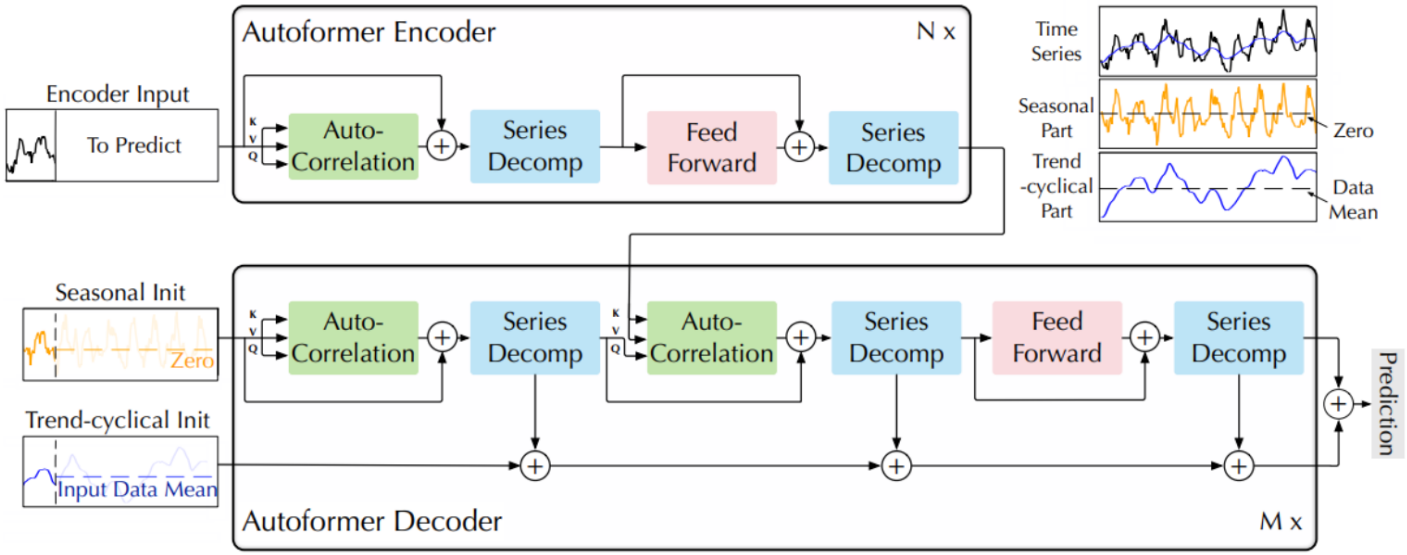
- Instead of using pair-wise attention for all time-step pairs, Informer [Zho+21; Nie23a] and some models utilize sparse attention, which ignores less relevant queries.
- By doing so, Informer achieves a time and memory complexity of $L \log L$.

Informer



- Informer defines two types of queries: 'Active' and 'Lazy' and only save 'Active' queries.
- Since the output of a vector is the weighted average of itself and the rest. The weights of "lazy" locations are nearly uniform, making the output essentially the mean of all values.

Autoformer



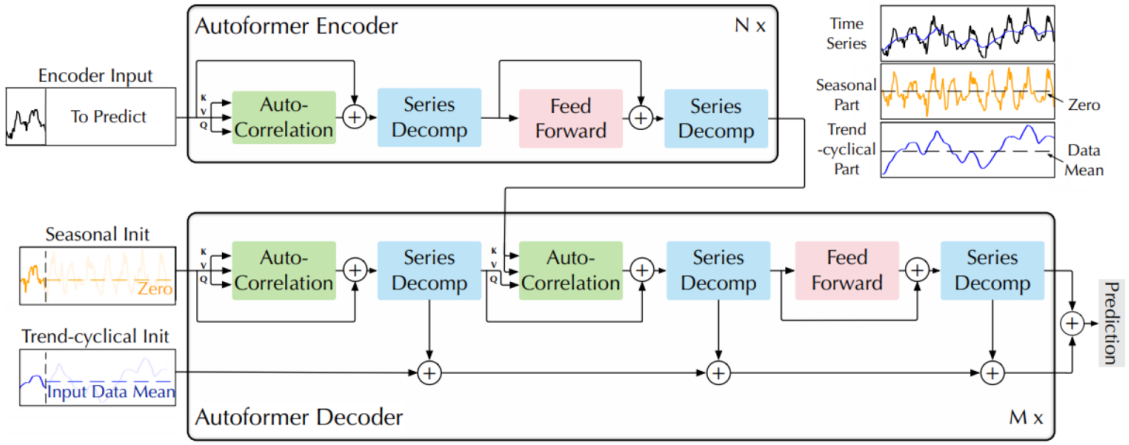
Autoformer[Wu+21; Nie23b] uses time series decomposition, breaking down the series into seasonal, trend, and random components. Decomposition method has proven effective in statistical literature and has been adopted in other time series research, such as in FEDformer.

Autoformer-Decomposition

Specifically, Autoformer employs average pooling (moving average) to separate trend and seasonal signals and model them separately.

$$\mathcal{X}_{\text{trend}} = \text{AvgPool}(\text{Padding}(\mathcal{X}))$$
$$\mathcal{X}_{\text{seasonal}} = \mathcal{X} - \mathcal{X}_{\text{trend}}$$

Autoformer-Decomposition



Encoder:

$$\mathcal{S}_{\text{en}}^{l,1}, \_ = \text{SeriesDecomp}\Big(\text{AutoCorrelation}(\mathcal{X}_{\text{en}}^{l-1}) + \mathcal{X}_{\text{en}}^{l-1}\Big)$$

$$\mathcal{S}_{\text{en}}^{l,2}, \_ = \text{SeriesDecomp}\Big(\text{FeedForward}(\mathcal{S}_{\text{en}}^{l,1}) + \mathcal{S}_{\text{en}}^{l,1}\Big),$$

Decoder:

$$\mathcal{S}_{\text{de}}^{l,1}, \mathcal{T}_{\text{de}}^{l,1} = \text{SeriesDecomp}\Big(\text{AutoCorrelation}(\mathcal{X}_{\text{de}}^{l-1}) + \mathcal{X}_{\text{de}}^{l-1}\Big)$$

$$\mathcal{S}_{\text{de}}^{l,2}, \mathcal{T}_{\text{de}}^{l,2} = \text{SeriesDecomp}\Big(\text{AutoCorrelation}(\mathcal{S}_{\text{de}}^{l,1}, \mathcal{X}_{\text{en}}^{N}) + \mathcal{S}_{\text{de}}^{l,1}\Big)$$

$$\mathcal{S}_{\text{de}}^{l,3}, \mathcal{T}_{\text{de}}^{l,3} = \text{SeriesDecomp}\Big(\text{FeedForward}(\mathcal{S}_{\text{de}}^{l,2}) + \mathcal{S}_{\text{de}}^{l,2}\Big)$$

$$\mathcal{T}_{\text{de}}^{l} = \mathcal{T}_{\text{de}}^{l-1} + \mathcal{W}_{l,1} * \mathcal{T}_{\text{de}}^{l,1} + \mathcal{W}_{l,2} * \mathcal{T}_{\text{de}}^{l,2} + \mathcal{W}_{l,3} * \mathcal{T}_{\text{de}}^{l,3},$$

Autoformer-Correlation

Another advantage of Autoformer is its use of a novel attention module. Specifically, it seamlessly replaces the original dot product attention between two time steps with Pearson correlation, enhancing the modeling of temporal relationships.

$$\mathbf{Autocorrelation}(\tau) = \mathbf{Corr}(y_t, y_{t-\tau})$$

In practice, the autocorrelation of the queries and keys for all lags is calculated simultaneously using FFT. This allows the autocorrelation mechanism to achieve $O(L \log L)$ time/memory complexity.

Comparison

Table 1: Multivariate results with different prediction lengths $O \in \{96, 192, 336, 720\}$. We set the input length I as 36 for ILI and 96 for the others. A lower MSE or MAE indicates a better prediction.

Models		**Autoformer**		Informer[48]		LogTrans[26]		Reformer[23]		LSTNet[25]		LSTM[17]		TCN[4]	
Metric		MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
ETT*	96	**0.255**	**0.339**	0.365	0.453	0.768	0.642	0.658	0.619	3.142	1.365	2.041	1.073	3.041	1.330
	192	**0.281**	**0.340**	0.533	0.563	0.989	0.757	1.078	0.827	3.154	1.369	2.249	1.112	3.072	1.339
	336	**0.339**	**0.372**	1.363	0.887	1.334	0.872	1.549	0.972	3.160	1.369	2.568	1.238	3.105	1.348
	720	**0.422**	**0.419**	3.379	1.388	3.048	1.328	2.631	1.242	3.171	1.368	2.720	1.287	3.135	1.354
Electricity	96	**0.201**	**0.317**	0.274	0.368	0.258	0.357	0.312	0.402	0.680	0.645	0.375	0.437	0.985	0.813
	192	**0.222**	**0.334**	0.296	0.386	0.266	0.368	0.348	0.433	0.725	0.676	0.442	0.473	0.996	0.821
	336	**0.231**	**0.338**	0.300	0.394	0.280	0.380	0.350	0.433	0.828	0.727	0.439	0.473	1.000	0.824
	720	**0.254**	**0.361**	0.373	0.439	0.283	0.376	0.340	0.420	0.957	0.811	0.980	0.814	1.438	0.784
Exchange	96	**0.197**	**0.323**	0.847	0.752	0.968	0.812	1.065	0.829	1.551	1.058	1.453	1.049	3.004	1.432
	192	**0.300**	**0.369**	1.204	0.895	1.040	0.851	1.188	0.906	1.477	1.028	1.846	1.179	3.048	1.444
	336	**0.509**	**0.524**	1.672	1.036	1.659	1.081	1.357	0.976	1.507	1.031	2.136	1.231	3.113	1.459
	720	**1.447**	**0.941**	2.478	1.310	1.941	1.127	1.510	1.016	2.285	1.243	2.984	1.427	3.150	1.458
Traffic	96	**0.613**	**0.388**	0.719	0.391	0.684	0.384	0.732	0.423	1.107	0.685	0.843	0.453	1.438	0.784
	192	**0.616**	**0.382**	0.696	0.379	0.685	0.390	0.733	0.420	1.157	0.706	0.847	0.453	1.463	0.794
	336	**0.622**	**0.337**	0.777	0.420	0.733	0.408	0.742	0.420	1.216	0.730	0.853	0.455	1.479	0.799
	720	**0.660**	**0.408**	0.864	0.472	0.717	0.396	0.755	0.423	1.481	0.805	1.500	0.805	1.499	0.804
Weather	96	**0.266**	**0.336**	0.300	0.384	0.458	0.490	0.689	0.596	0.594	0.587	0.369	0.406	0.615	0.589
	192	**0.307**	**0.367**	0.598	0.544	0.658	0.589	0.752	0.638	0.560	0.565	0.416	0.435	0.629	0.600
	336	**0.359**	**0.395**	0.578	0.523	0.797	0.652	0.639	0.596	0.597	0.587	0.455	0.454	0.639	0.608
	720	**0.419**	**0.428**	1.059	0.741	0.869	0.675	1.130	0.792	0.618	0.599	0.535	0.520	0.639	0.610
ILI	24	**3.483**	**1.287**	5.764	1.677	4.480	1.444	4.400	1.382	6.026	1.770	5.914	1.734	6.624	1.830
	36	**3.103**	**1.148**	4.755	1.467	4.799	1.467	4.783	1.448	5.340	1.668	6.631	1.845	6.858	1.879
	48	**2.669**	**1.085**	4.763	1.469	4.800	1.468	4.832	1.465	6.080	1.787	6.736	1.857	6.968	1.892
	60	**2.770**	**1.125**	5.264	1.564	5.278	1.560	4.882	1.483	5.548	1.720	6.870	1.879	7.127	1.918

* *ETT* means the ETTm2. See Appendix A for the **full benchmark** of ETTh1, ETTh2, ETTm1.

References I

[Nie22] Kashif Rasul Niels Rogge. *Probabilistic Time Series Forecasting with huggingface Transformers*. 2022. URL: https://huggingface.co/blog/time-series-transformers.

[Nie23a] Kashif Rasul Niels Rogge. *Multivariate Probabilistic Time Series Forecasting with Informer*. 2023. URL: https://huggingface.co/blog/informer.

[Nie23b] Kashif Rasul Niels Rogge. *Yes, Transformers are Effective for Time Series Forecasting (+ Autoformer)*. 2023. URL: https://huggingface.co/blog/autoformer.

[Vas+17] Ashish Vaswani et al. "Attention is all you need". In: *Advances in neural information processing systems* 30 (2017).

[Wen+22] Qingsong Wen et al. "Transformers in time series: A survey". In: *arXiv preprint arXiv:2202.07125* (2022).

References II

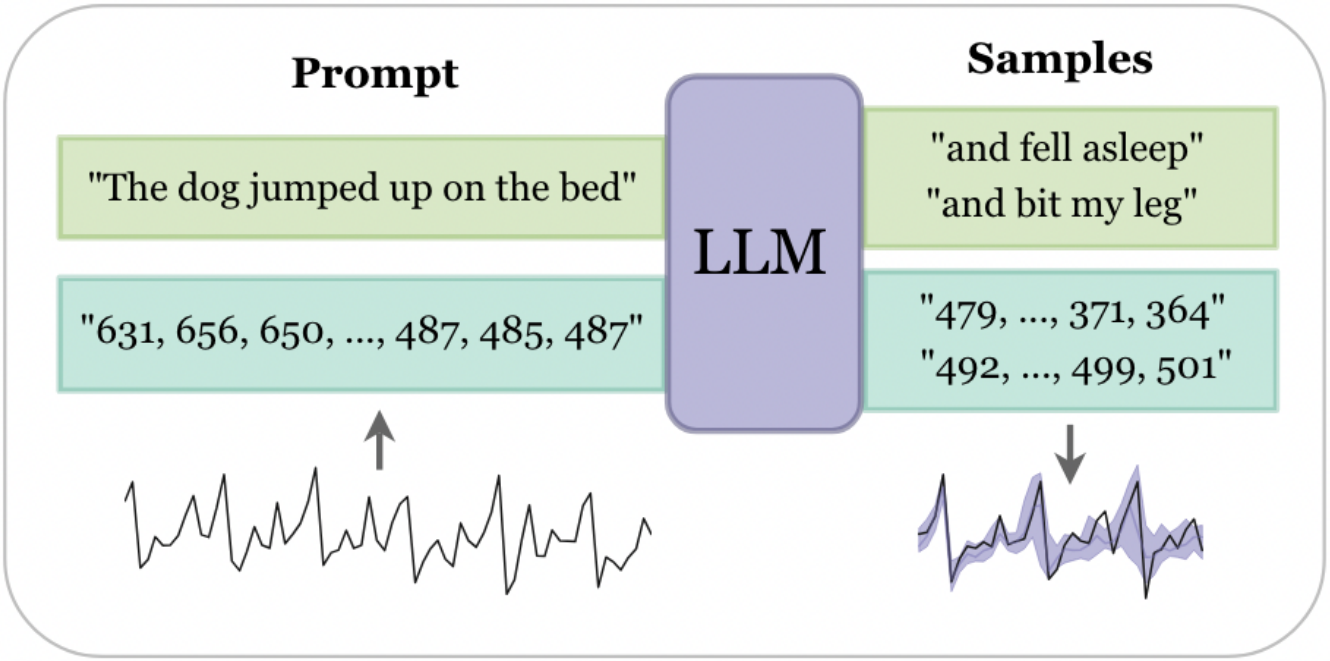
[Wu+21] Haixu Wu et al. “Autoformer: Decomposition transformers with auto-correlation for long-term series forecasting”. In: *Advances in neural information processing systems* 34 (2021), pp. 22419–22430.

[Zho+21] Haoyi Zhou et al. “Informer: Beyond efficient transformer for long sequence time-series forecasting”. In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 35. 12. 2021, pp. 11106–11115.

Table of contents

Large Language Models Are Zero-Shot Time Series Forecasters[Gru+24]

- Why LLM can be used for time series?



Large Language Models Are Zero-Shot Time Series Forecasters

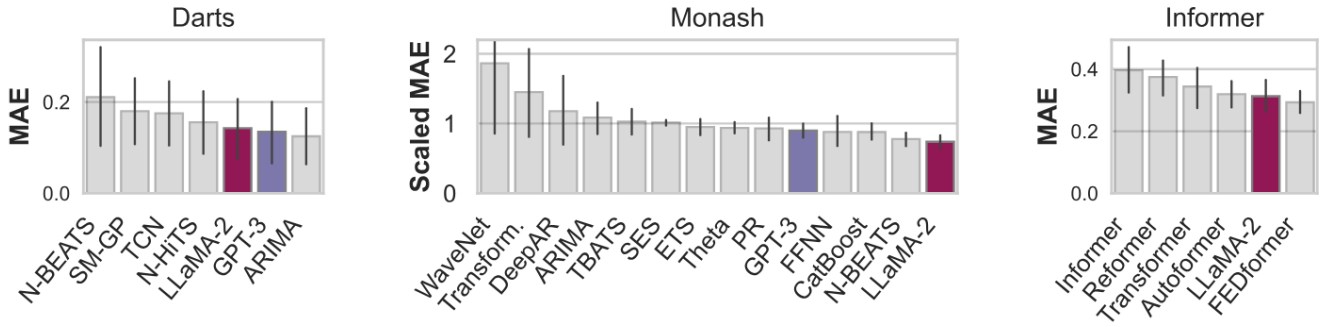
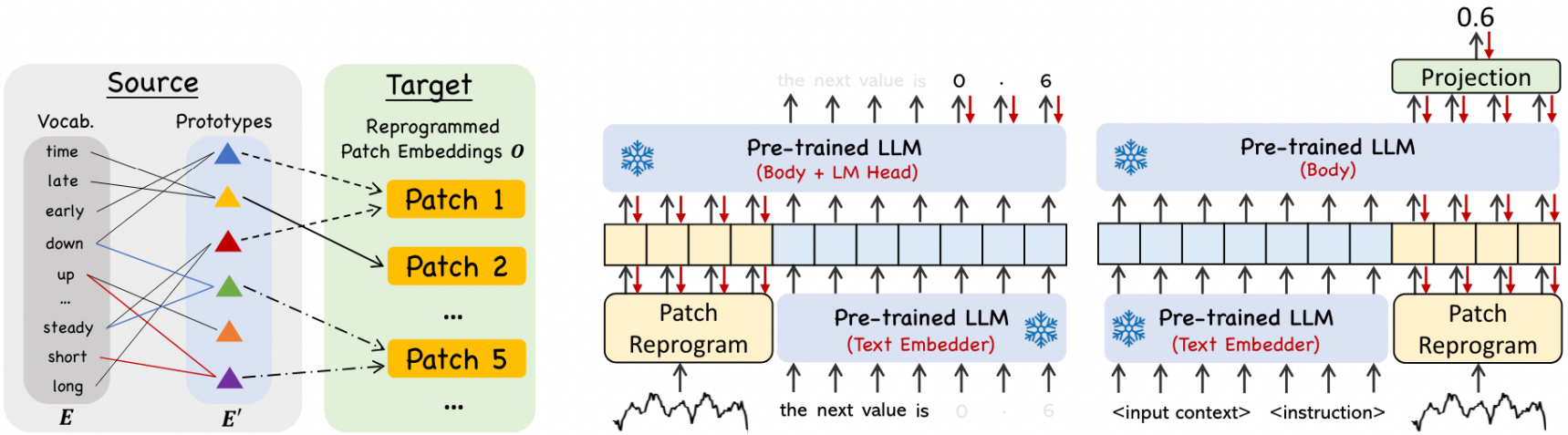


Figure 4: LLMTIME with base model GPT-3 or LLaMA-2 70B has the best or second best aggregated performance on several deterministic time series benchmarks [23, 18, 54] while being entirely zero-shot. Collectively, these benchmarks comprise 29 individual datasets with diverse sources, lengths, and noise levels. For Monash MAE numbers, established results are reported on unnormalized data, so we normalize values before aggregating (Appendix C.2). The informer datasets are multivariate, and we predict each covariate independently with LLMTIME (Appendix C.3). GPT-3 evaluation on the Informer datasets was skipped because of the cost of API queries. Error bars show standard errors over the individual datasets in each benchmark.

- Is the zero-shot here really zero shot?

Time-LLM

- Time-LLM[Jin+23] attempts to align time series patches with word embeddings in LLMs.
- Specifically, Time-LLM utilizes cross-attention, a technique commonly employed in various multimodal models, to capture the correlation between time series data and word embeddings.



(a) Patch Reprogramming

(b) Patch-as-Prefix and Prompt-as-Prefix

Figure 3: Illustration of **(a)** patch reprogramming and **(b)** Patch-as-Prefix versus Prompt-as-Prefix.

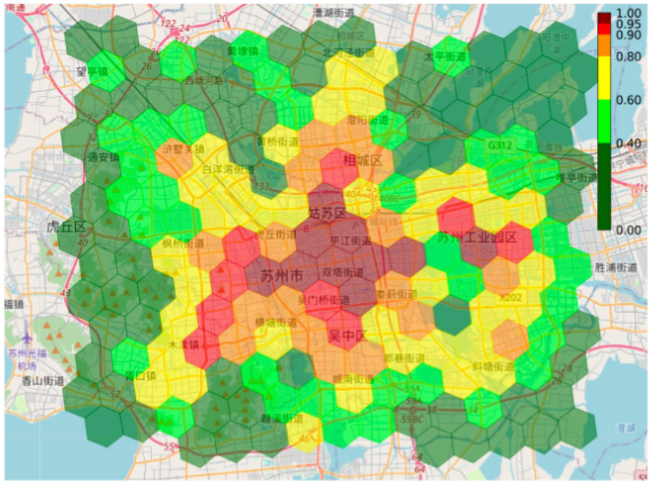
References I

[Gru+24] Nate Gruver et al. "Large language models are zero-shot time series forecasters". In: *Advances in Neural Information Processing Systems* 36 (2024).

[Jin+23] Ming Jin et al. "Time-llm: Time series forecasting by reprogramming large language models". In: *arXiv preprint arXiv:2310.01728* (2023).

Table of contents

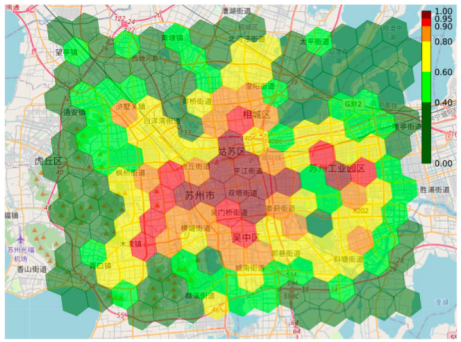
Formulation



$$
\begin{array}{c}
 & \begin{array}{ccccccc} g_1 & g_2 & g_3 & \cdot & g_j & \cdot & g_{183} \end{array} \\
\begin{array}{c} g_1 \\ g_2 \\ g_3 \\ \cdot \\ g_i \\ \cdot \\ g_{183} \end{array} &
\begin{bmatrix}
9 & 0 & 12 & \cdot & \cdot & \cdot & 0 \\
0 & 23 & 0 & \cdot & \cdot & \cdot & 2 \\
1 & 8 & 9 & \cdot & \cdot & \cdot & 4 \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 0 \\
\cdot & \cdot & \cdot & \cdot & x_{i,j} & \cdot & 6 \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 0 \\
5 & 3 & 7 & 0 & 0 & 2 & 0
\end{bmatrix}
\end{array}
$$

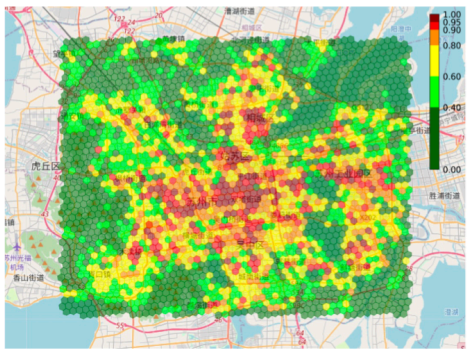
- The city can be divided into multiple grids, with the OD matrix $M^{(t)}$ representing the traffic flow between these grids at time step t. Specifically, $M^{(t)}_{i,j}$ indicates the traffic flow from grid i to grid j at this time step.
- Given the OD matrices from the last K time steps $M^{t-K:t}$, predict the OD matrices for the next τ time steps $M^{t+1:t+\tau}$.

Intuitions



(a)

	g_1	g_2	g_3	$\cdot$	g_j	$\cdot$	g_{183}
g_1	9	0	12	$\cdot$	$\cdot$	$\cdot$	0
g_2	0	23	0	$\cdot$	$\cdot$	$\cdot$	2
g_3	1	8	9	$\cdot$	$\cdot$	$\cdot$	4
$\cdot$	$\cdot$	$\cdot$	$\cdot$	$\cdot$	$\cdot$	$\cdot$	0
g_i	$\cdot$	$\cdot$	$\cdot$	$\cdot$	$x_{i,j}$	$\cdot$	6
$\cdot$	$\cdot$	$\cdot$	$\cdot$	$\cdot$	$\cdot$	$\cdot$	0
g_{183}	5	3	7	0	0	2	0



(b)

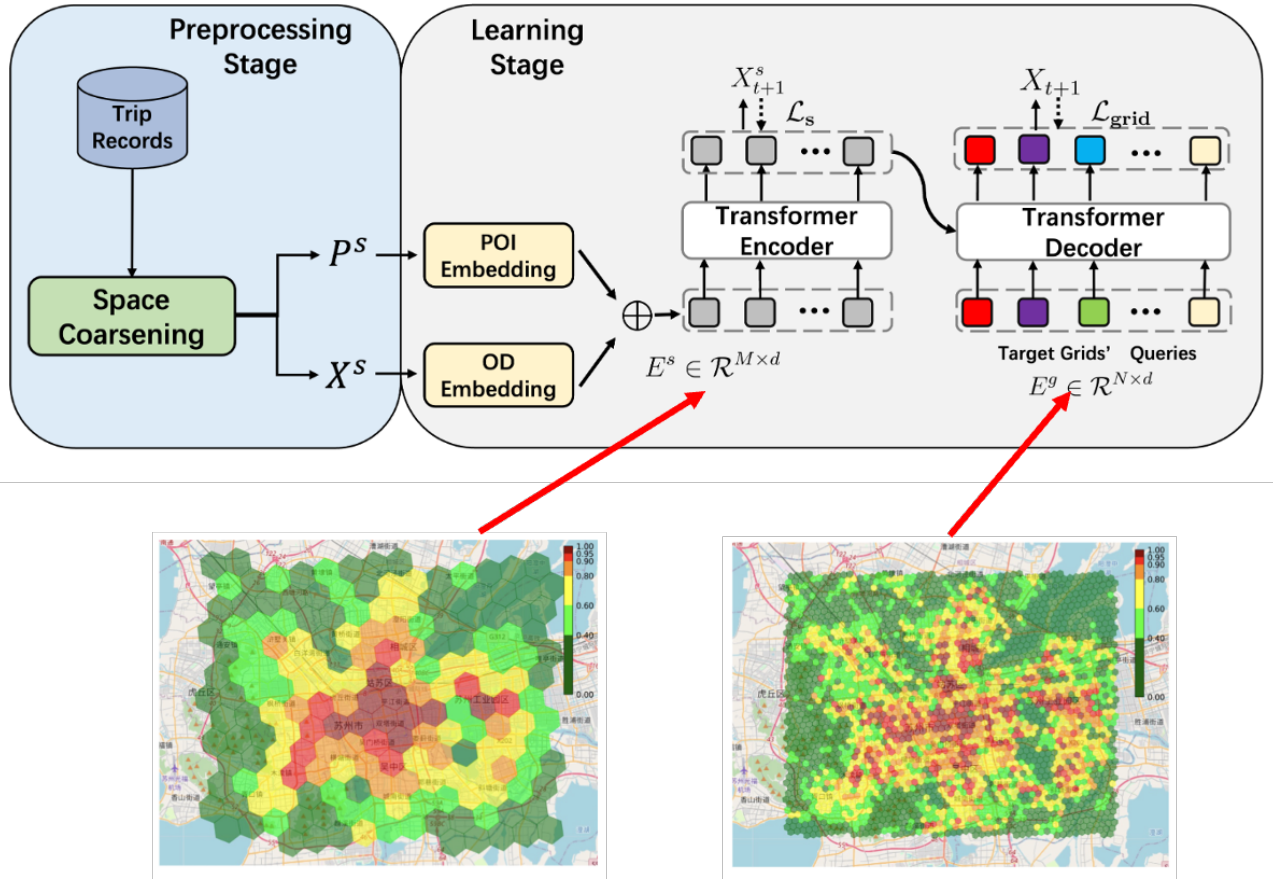
	g_1	g_2	g_3	$\cdot$	g_j	$\cdot$	g_{2531}
g_1	1	0	2	$\cdot$	$\cdot$	$\cdot$	0
g_2	0	0	0	$\cdot$	$\cdot$	$\cdot$	2
g_3	0	0	0	$\cdot$	$\cdot$	$\cdot$	0
$\cdot$	$\cdot$	$\cdot$	$\cdot$	$\cdot$	$\cdot$	$\cdot$	0
g_i	$\cdot$	$\cdot$	$\cdot$	$\cdot$	$x_{i,j}$	$\cdot$	0
$\cdot$	$\cdot$	$\cdot$	$\cdot$	$\cdot$	$\cdot$	$\cdot$	0
g_{2531}	0	0	0	0	0	1	0

A Dilemma

- A coarse-grained OD matrix is undesirable because the large grid sizes (e.g., 1 km) are not helpful for downstream problems.
- A fine-grained OD matrix often experiences high sparsity, as many OD flows are zero (e.g., no one takes a taxi within a hospital).

Can we utilize coarse-grained OD data to assist the prediction of fine-grained OD flows?

Model structure



- Clustering fine grained grids into coarse grained grids and put them into encoder and decoder separately.
- Adopt two separate MSE losses to train the model.

Results

Method	City-C			City-S		
	wMAPE	**RMSE**	**CPC**	**wMAPE**	**RMSE**	**CPC**
HA	0.813	1.442	0.348	0.821	1.435	0.355
OLSR	0.822	1.419	0.324	0.816	1.351	0.333
LASSO	0.807	1.424	0.359	0.813	1.349	0.337
CSTN	0.782	1.370	0.354	0.721	1.217	0.451
MRSTN	0.788	1.380	0.351	0.766	1.253	0.464
GEML	0.667	1.255	0.540	0.605	1.146	0.597
STGCN	0.681	1.337	0.488	0.596	1.210	0.674
OD-CED	**0.411**	**0.905**	**0.776**	**0.323**	**0.740**	**0.889**

Results