

# Advanced Workshop on Machine Learning

# Lecture 3: Recurrent Neural Network

# Agenda

## Part 1

- Project Makridakis
- Recurrent Layer

## Part 2

- Case Study: DeepAR
- Assignment #3: WeatherNet

## Optional Part 3:

- Attention Layer
- Transformer for Time Series

# Part 1

## Recurrent Layer



(Image by a Stable Diffusion model)

# Tasks with Sequences

Task	Example of Input	Example of Output
Churn prediction	Customer's order history	Will the customer churn?
Economic forecasting	Past energy prices	This year's GDP growth
Weather forecasting	Yesterday's temperature	Today's temperature
Stock price prediction	Historical stock prices	Future stock price

Non-sequences

Temporal sequence

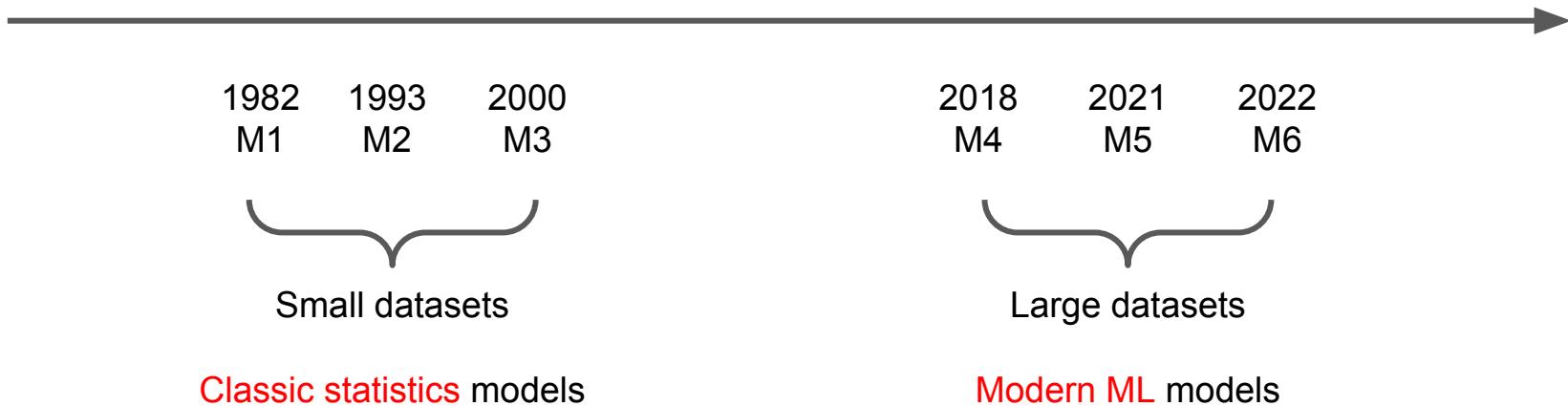
# Temporal Sequences

Measurement	Frequency	Duration	Sequence Shape
Temperature in LA	1 day	10 years	$\sim 3.6K \times 1$
Audio recording	44 kHz	1 minute	$\sim 2.6M \times 1$
Earth vibrations	4 MHz	2 minutes	$480M \times 1$
Price of every stock in S&P 500	1 Hz	1 week	$117K \times 500 (= \sim 60M)$
Purchase activity by 1M customers	1 day	1 year	$365 \times 1M (= 365M)$

1-dimensional sequence

2-dimensional sequence

# Project Makridakis



Summary results published in [International Journal of Forecasting](#).

# Project M4 (2018)

## Dataset

100K time series

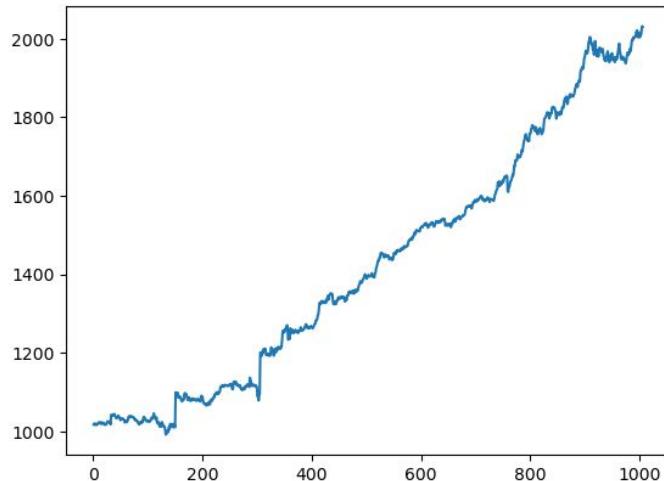
Socio-economic domains

- Economics
- Financial markets
- Industry
- Demographics

Y-Q-M-D frequencies

Continuous time series

Example: Daily time series

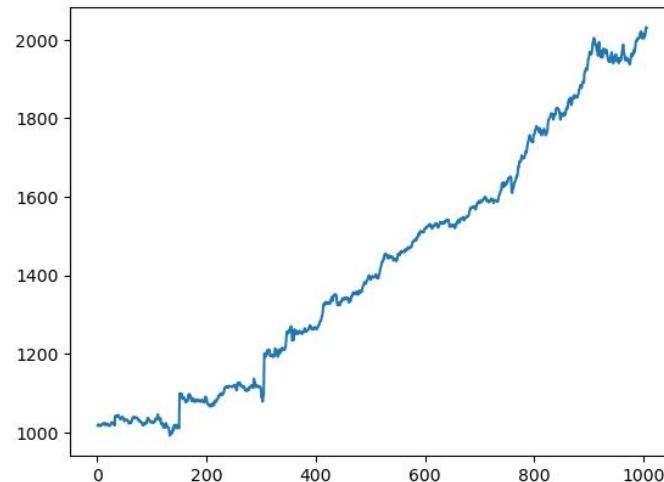


# Project M4 (2018)

## Takeaways

- Ensembles perform better
  - More in Lecture 5
- Top-1 model:
  - **LSTM (RNN)** + Exponential Smoothing
  - **Cross-learning** across time series
- Top-2 model:
  - Ensemble of stats models + meta-learner
  - More in Lecture 5

Example: Daily time series



# Project M5 (2021)

## Dataset

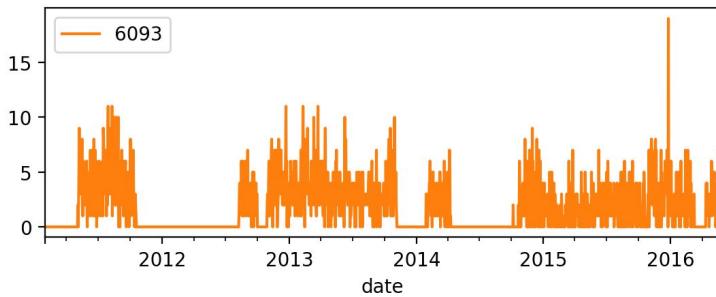
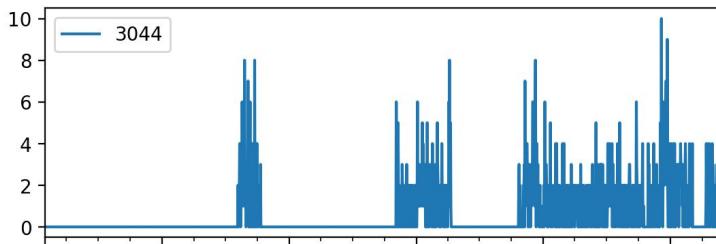
Walmart sales

~42K time series

## Hierarchy:

- State (3)
- Store (10)
- Department
- Category
- Product (~3K)

Example: Same product, different stores



# Project M5 (2021)

## Takeaways

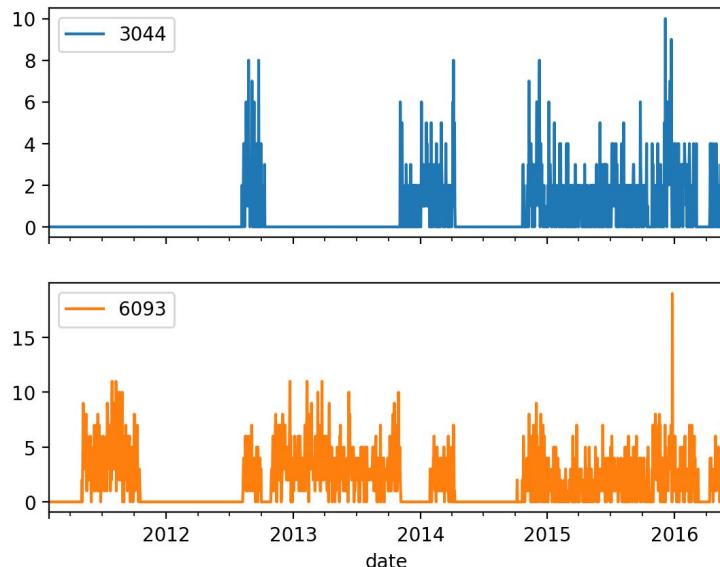
Top-1 model:

- LightGBM
- More in Lecture 5

Top-3 model:

- DeepAR (RNN)
- Cross-learning across time series

Example: Same product, different stores



# Project M6 (2022)

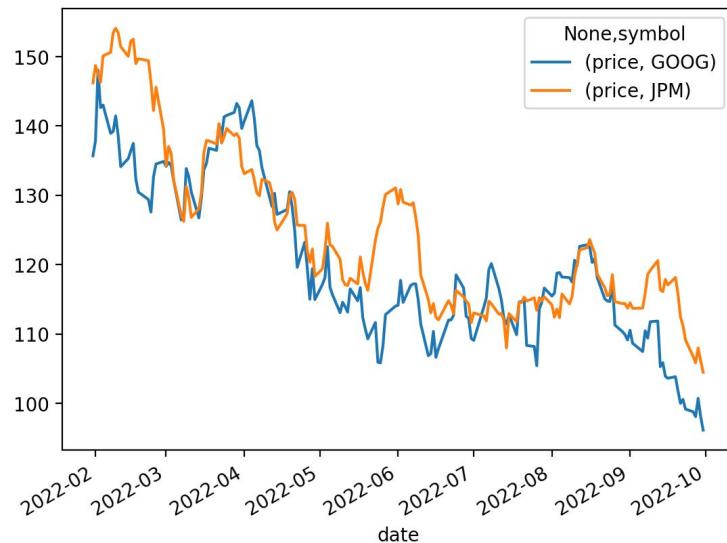
## Dataset

Stock market prices

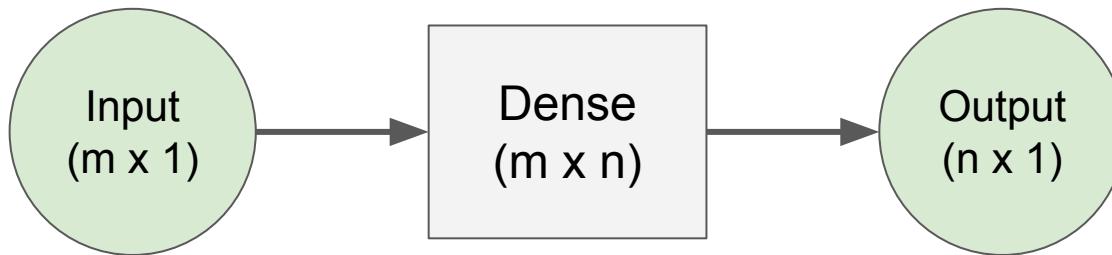
100 tickers

Unlimited data sources

Example: Google and JP Morgan



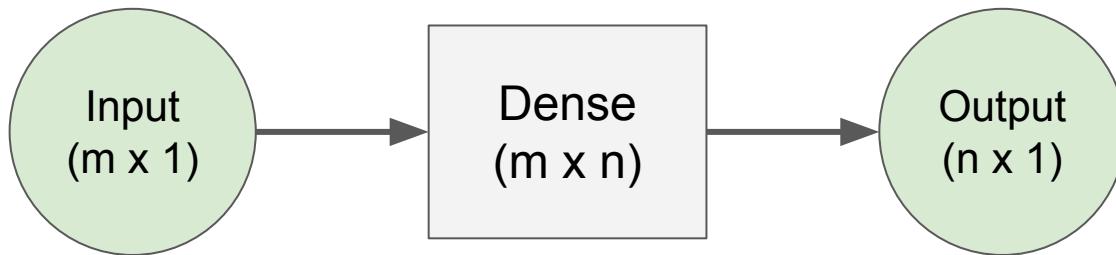
# What about Dense Layer?



## Problem 1

Count of parameters grows **proportionally to the sequence length**. Density of information might be too low to give the model such capacity.

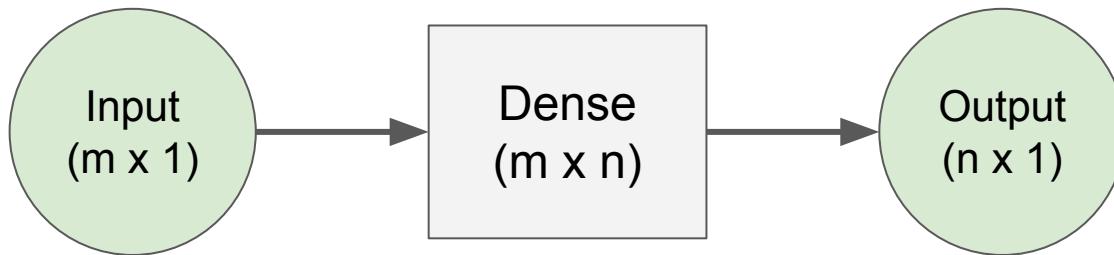
# What about Dense Layer?



## Problem 2

In any sequence, **nearby input dimensions are related**, and their relationship is **location-invariant**. Dense layer treats every dimension independently, until it learns otherwise.

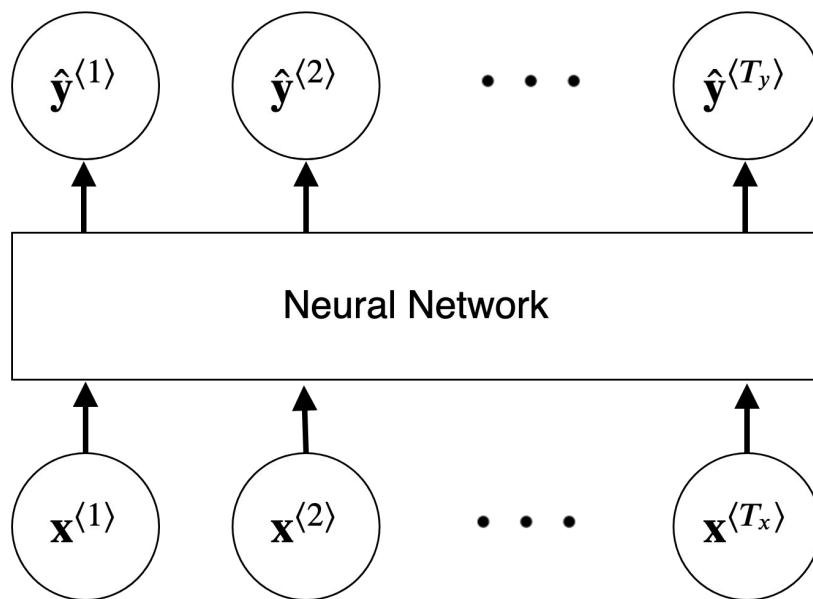
# What about Dense Layer?



## Problem 3

Sequence length can vary across examples within the same task. For dense layers,  
**shape of input and output is frozen** at design time.

# Sequence to Sequence

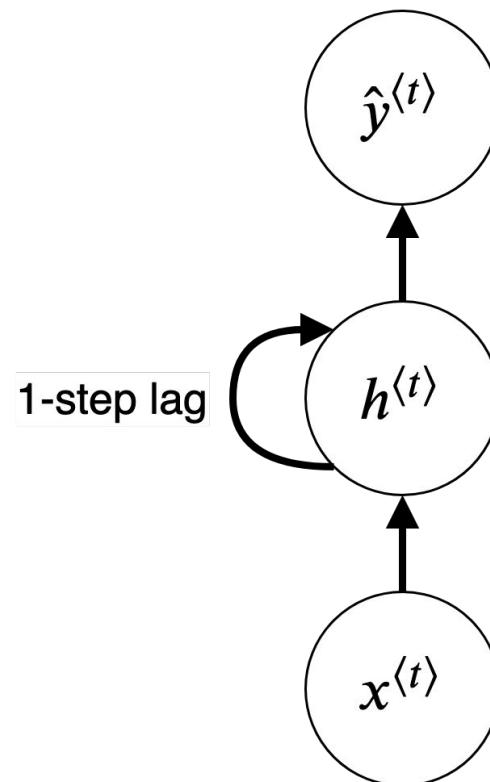


$$\mathbf{x}^{(t)} \in \mathbb{R}^n, t \in \{1, 2, \dots, T_x\}$$

$$\hat{\mathbf{y}}^{(t)} \in \mathbb{R}^m, t \in \{1, 2, \dots, T_y\}$$

# Recurrent Connection

$$\mathbf{h}^{(t)} = f(\mathbf{h}^{(t-1)}, \mathbf{x}^{(t)})$$

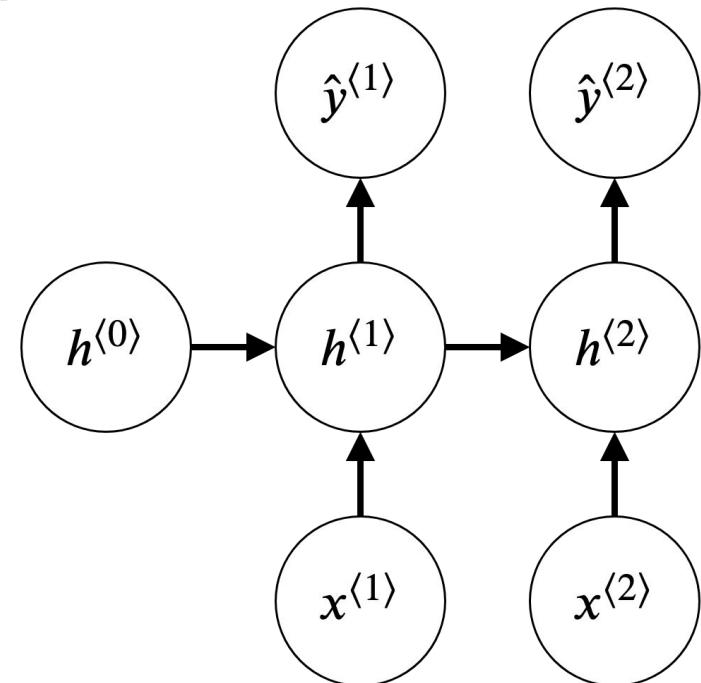


# Unrolled Recurrent Connection

$$\mathbf{h}^{(0)} = \mathbf{0}$$

$$\mathbf{h}^{(1)} = f(\mathbf{h}^{(0)}, \mathbf{x}^{(1)})$$

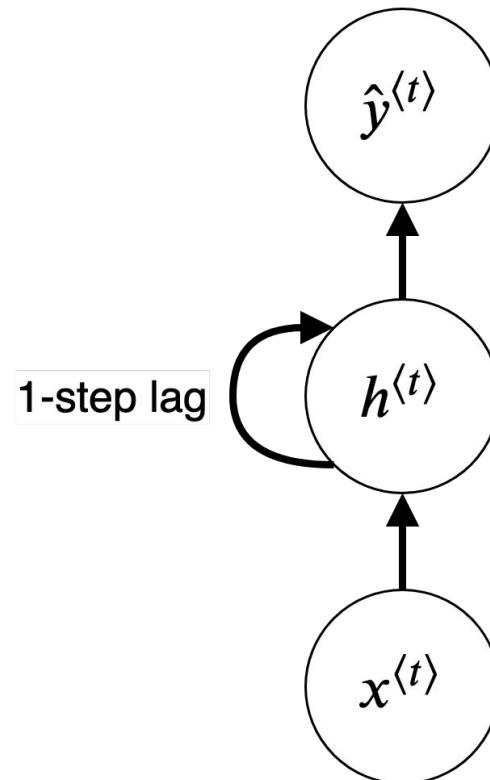
$$\mathbf{h}^{(2)} = f(\mathbf{h}^{(1)}, \mathbf{x}^{(2)})$$



# Example: Dot Product / Tanh

$$\mathbf{z}^{(t)} = \mathbf{V} \cdot \mathbf{h}^{(t-1)} + \mathbf{U} \cdot \mathbf{x}^{(t)} + \mathbf{b}$$

$$\mathbf{h}^{(t)} = \tanh(\mathbf{z}^{(t)})$$



# Dot Product / Tanh: Unrolled

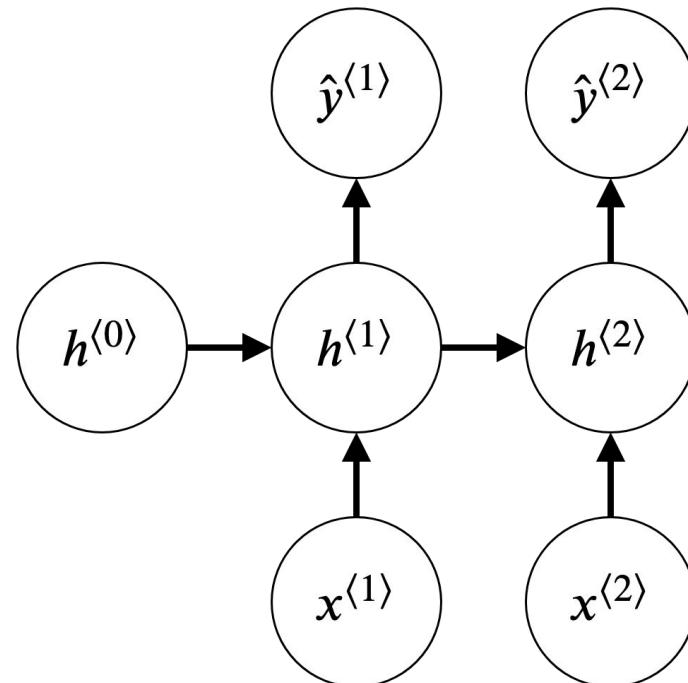
$$\mathbf{h}^{(0)} = \mathbf{0}$$

$$\mathbf{z}^{(1)} = \mathbf{V} \cdot \mathbf{h}^{(0)} + \mathbf{U} \cdot \mathbf{x}^{(1)} + \mathbf{b}$$

$$\mathbf{h}^{(1)} = \tanh(\mathbf{z}^{(1)})$$

$$\mathbf{z}^{(2)} = \mathbf{V} \cdot \mathbf{h}^{(1)} + \mathbf{U} \cdot \mathbf{x}^{(2)} + \mathbf{b}$$

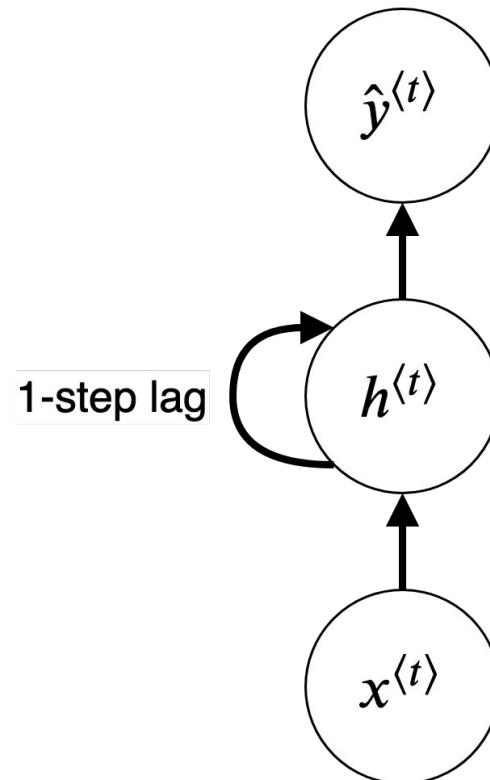
$$\mathbf{h}^{(2)} = \tanh(\mathbf{z}^{(2)})$$



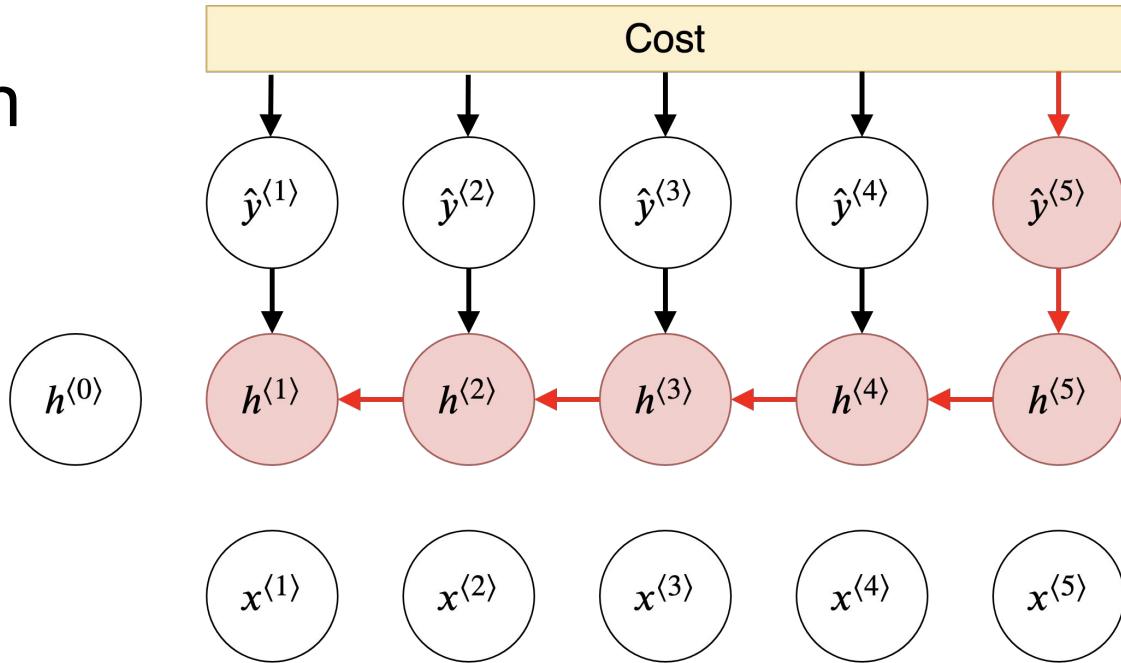
# Stacked Parameters Notation

$$\mathbf{W} = (\mathbf{V} \ \mathbf{U})$$

$$\mathbf{h}^{(t)} = \tanh\left(\mathbf{W} \cdot \begin{pmatrix} \mathbf{h}^{(t-1)} \\ \mathbf{x}^{(t)} \end{pmatrix} + \mathbf{b}\right)$$



# Backpropagation Through Time



$$\frac{\partial C}{\partial \mathbf{W}} = \dots + \frac{\partial C}{\partial \hat{\mathbf{y}}^{(5)}} \frac{\partial \hat{\mathbf{y}}^{(5)}}{\partial \mathbf{h}^{(5)}} \frac{\partial \mathbf{h}^{(5)}}{\partial \mathbf{h}^{(4)}} \frac{\partial \mathbf{h}^{(4)}}{\partial \mathbf{h}^{(3)}} \frac{\partial \mathbf{h}^{(3)}}{\partial \mathbf{h}^{(2)}} \frac{\partial \mathbf{h}^{(2)}}{\partial \mathbf{h}^{(1)}} \frac{\partial \mathbf{h}^{(1)}}{\partial \mathbf{W}} + \dots$$

# Exploding Gradient

## Problem

Repetitive matrix multiplication  
makes gradient very large.

## Solution 1

If the gradient's norm is larger than  
threshold, element-wise rescale  
gradient by (threshold / norm).

$$\frac{dC}{dW} = \nabla, n = \|\nabla\|_2$$
$$\nabla_i \leftarrow \begin{cases} \nabla_i \cdot \frac{t}{n} & \text{if } n > t \\ \nabla_i & \text{otherwise} \end{cases}$$

# Exploding Gradient

## Problem

Repetitive matrix multiplication makes gradient very large.

## Solution 2

Simply element-wise clip gradient to a reasonable interval.

$$\frac{dC}{dW} = \nabla, m > 0$$
$$\nabla_i \leftarrow \begin{cases} \min(\nabla_i, m) & \text{if } \nabla_i > 0 \\ \max(\nabla_i, -m) & \text{otherwise} \end{cases}$$

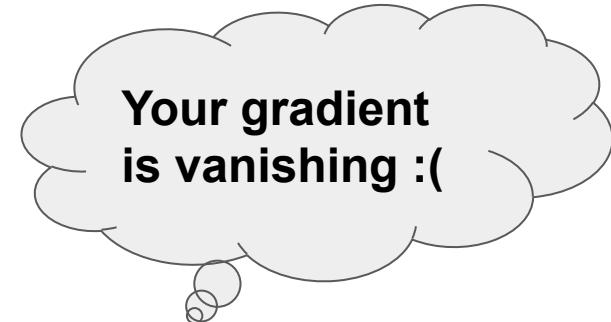
# Vanishing Gradient

## Problem

Repetitive matrix multiplication makes **gradient very small**.

## Solution

Need a different cell architecture, where gradient **is not repetitively multiplied by the same matrix**.

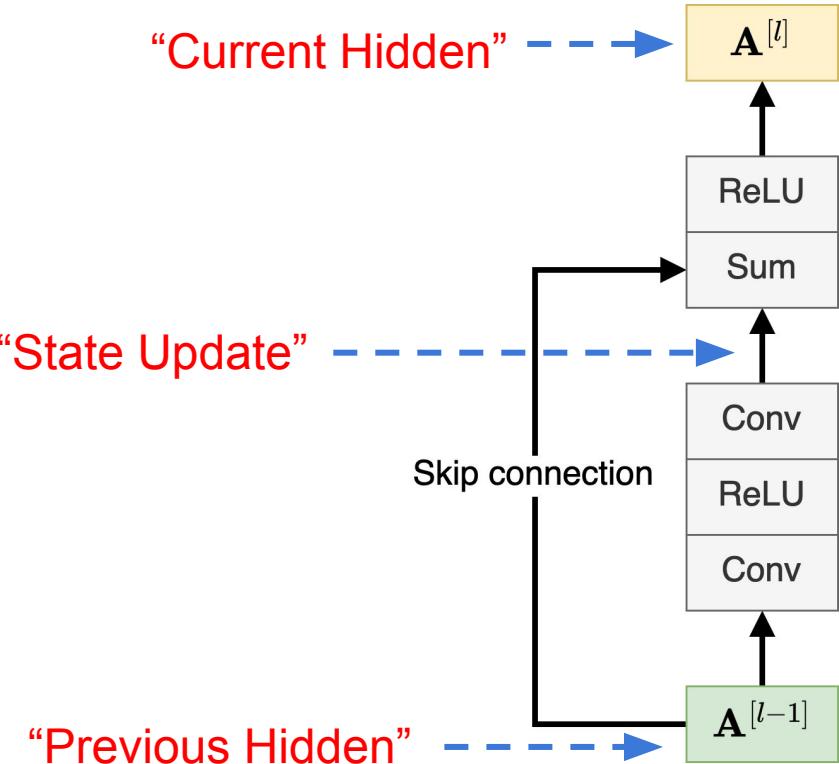


(Image by a Stable Diffusion model)

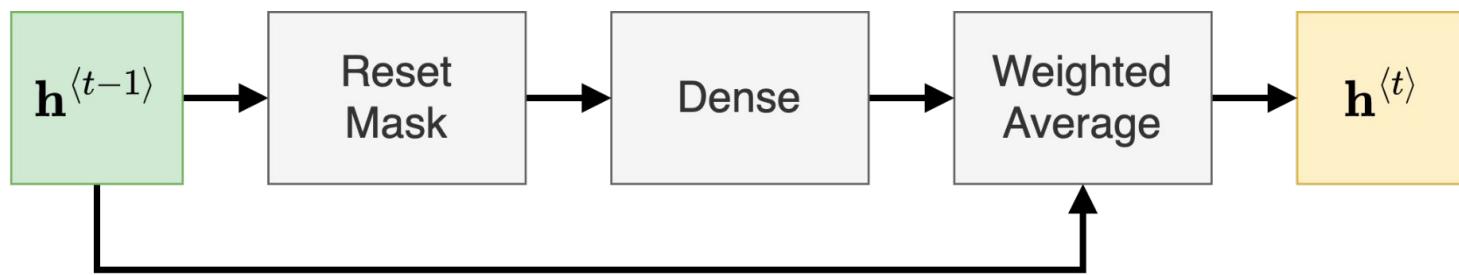
# Analogy with Residual Layer

In ResNet, the skip connection acts along the **depth** dimension.

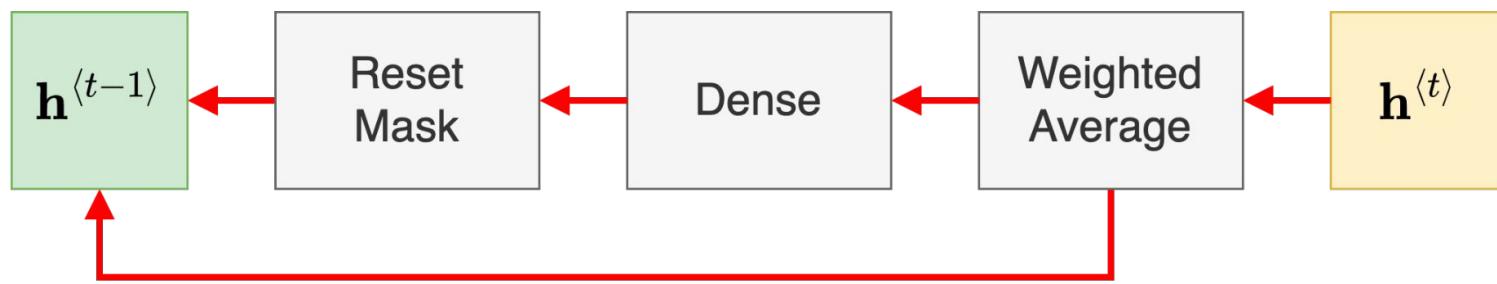
In RNNs, the skip connection acts along the **time** dimension.



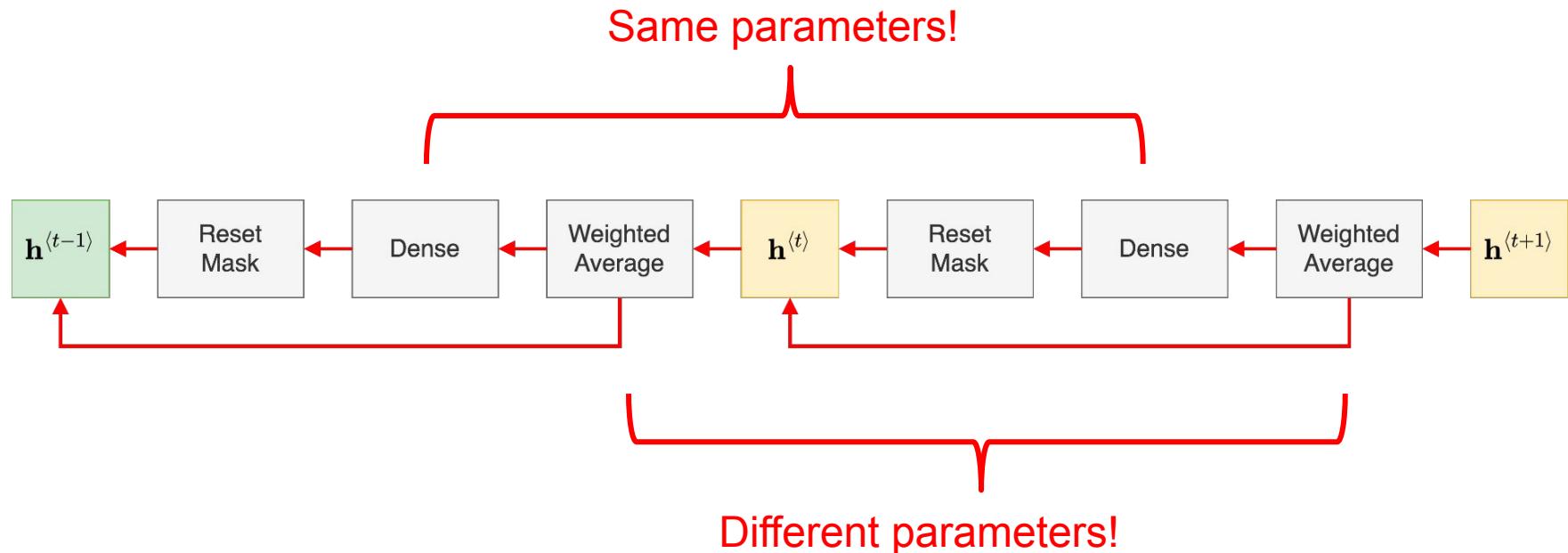
# Gated Recurrent Unit



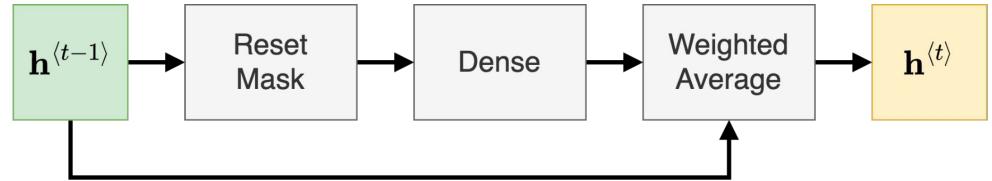
# GRU: Gradient Flow



# GRU: Gradient Flow



# GRU: Equations



$$\begin{pmatrix} \mathbf{r}^{(t)} \\ \mathbf{z}^{(t)} \end{pmatrix} = \text{sigmoid} \left[ \mathbf{U} \cdot \begin{pmatrix} \mathbf{h}^{(t-1)} \\ \mathbf{x}^{(t)} \end{pmatrix} + \mathbf{b} \right]$$

$$\tilde{\mathbf{h}}^{(t)} = \tanh \left[ \mathbf{V} \cdot \begin{pmatrix} \mathbf{r}^{(t)} \odot \mathbf{h}^{(t-1)} \\ \mathbf{x}^{(t)} \end{pmatrix} + \mathbf{c} \right]$$

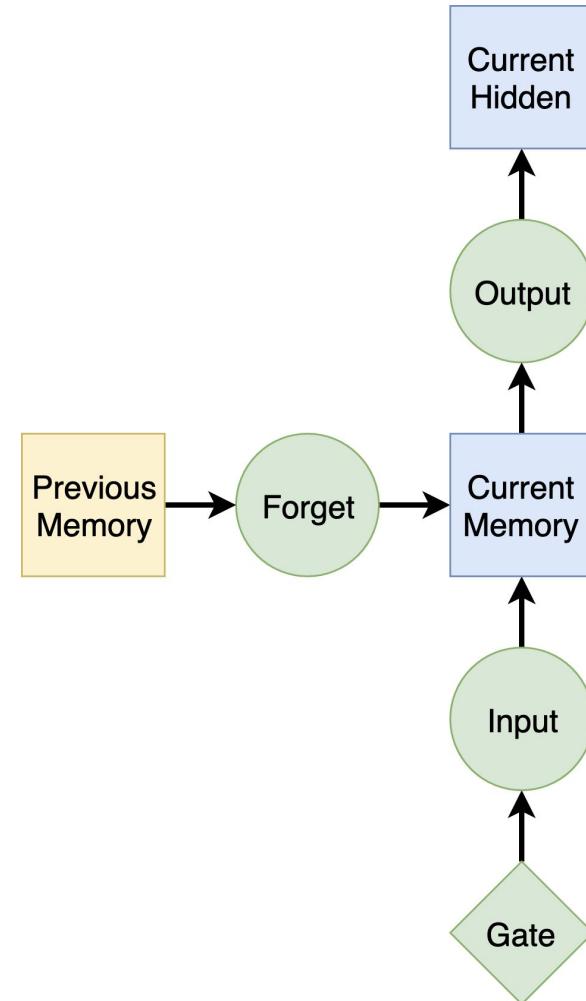
$$\mathbf{h}^{(t)} = \left[ \mathbf{z}^{(t)} \odot \mathbf{h}^{(t-1)} \right] + \left[ (1 - \mathbf{z}^{(t)}) \odot \tilde{\mathbf{h}}^{(t)} \right]$$

# Long Short-Term Memory

$$\begin{pmatrix} \mathbf{i}^{(t)} \\ \mathbf{f}^{(t)} \\ \mathbf{o}^{(t)} \\ \mathbf{g}^{(t)} \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} \cdot \mathbf{W} \cdot \begin{pmatrix} \mathbf{h}^{(t-1)} \\ \mathbf{x}^{(t)} \end{pmatrix} + \mathbf{b}$$

$$\mathbf{c}^{(t)} = \mathbf{f} \odot \mathbf{c}^{(t-1)} + \mathbf{i}^{(t)} \odot \mathbf{g}^{(t)}$$

$$\mathbf{h}^{(t)} = \mathbf{o} \odot \tanh(\mathbf{c}^{(t)})$$



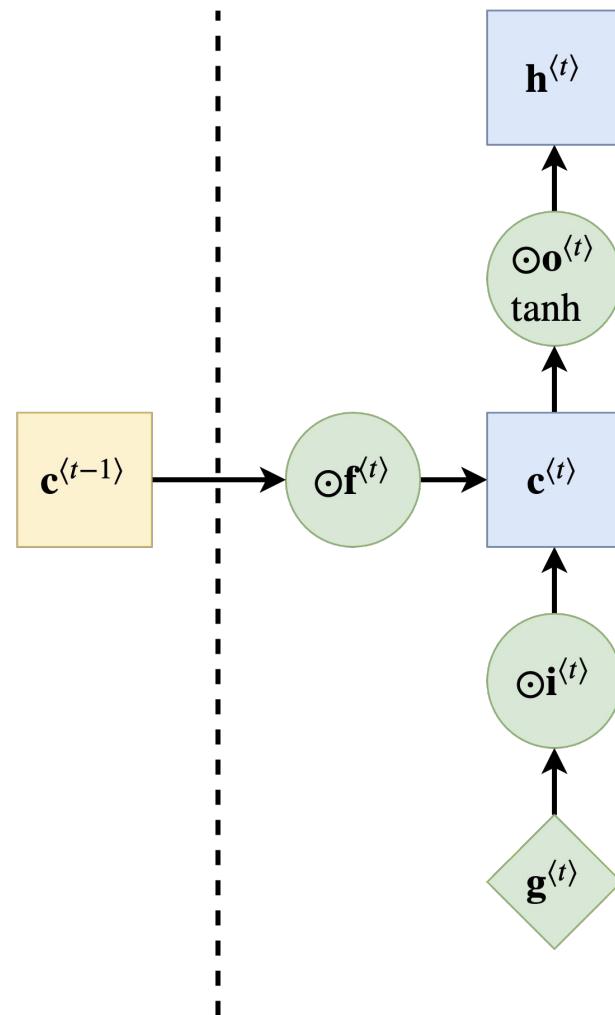
Hochreiter, S, and Schmidhuber, J. (1997)

# Long Short-Term Memory

$$\begin{pmatrix} \mathbf{i}^{(t)} \\ \mathbf{f}^{(t)} \\ \mathbf{o}^{(t)} \\ \mathbf{g}^{(t)} \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} \cdot \mathbf{W} \cdot \begin{pmatrix} \mathbf{h}^{(t-1)} \\ \mathbf{x}^{(t)} \end{pmatrix} + \mathbf{b}$$

$$\mathbf{c}^{(t)} = \mathbf{f} \odot \mathbf{c}^{(t-1)} + \mathbf{i}^{(t)} \odot \mathbf{g}^{(t)}$$

$$\mathbf{h}^{(t)} = \mathbf{o} \odot \tanh(\mathbf{c}^{(t)})$$

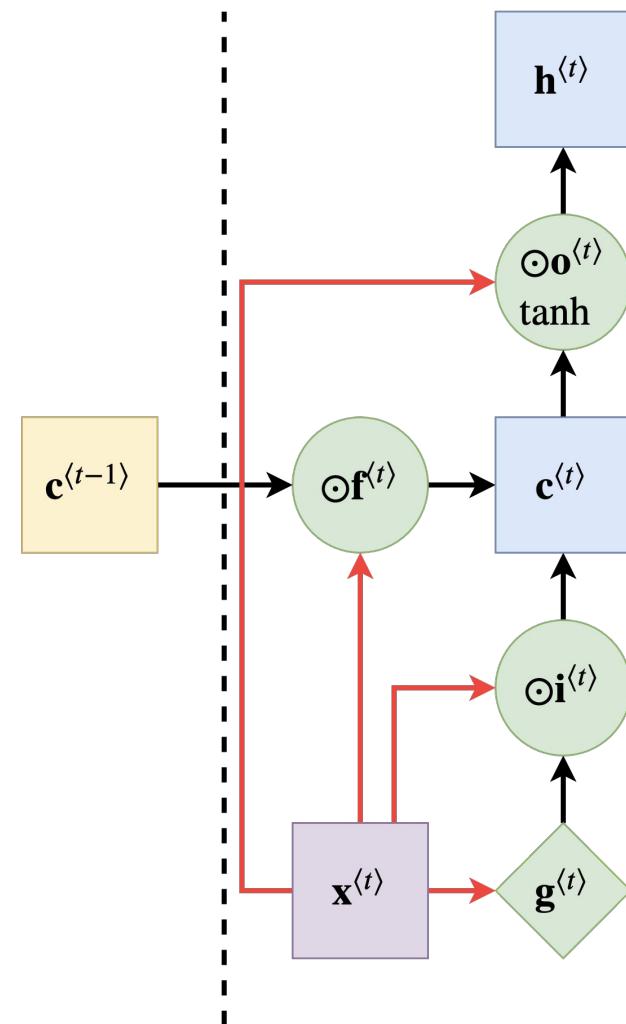


# Long Short-Term Memory

$$\begin{pmatrix} \mathbf{i}^{(t)} \\ \mathbf{f}^{(t)} \\ \mathbf{o}^{(t)} \\ \mathbf{g}^{(t)} \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} \cdot \mathbf{W} \cdot \begin{pmatrix} \mathbf{h}^{(t-1)} \\ \mathbf{x}^{(t)} \end{pmatrix} + \mathbf{b}$$

$$\mathbf{c}^{(t)} = \mathbf{f} \odot \mathbf{c}^{(t-1)} + \mathbf{i}^{(t)} \odot \mathbf{g}^{(t)}$$

$$\mathbf{h}^{(t)} = \mathbf{o} \odot \tanh(\mathbf{c}^{(t)})$$



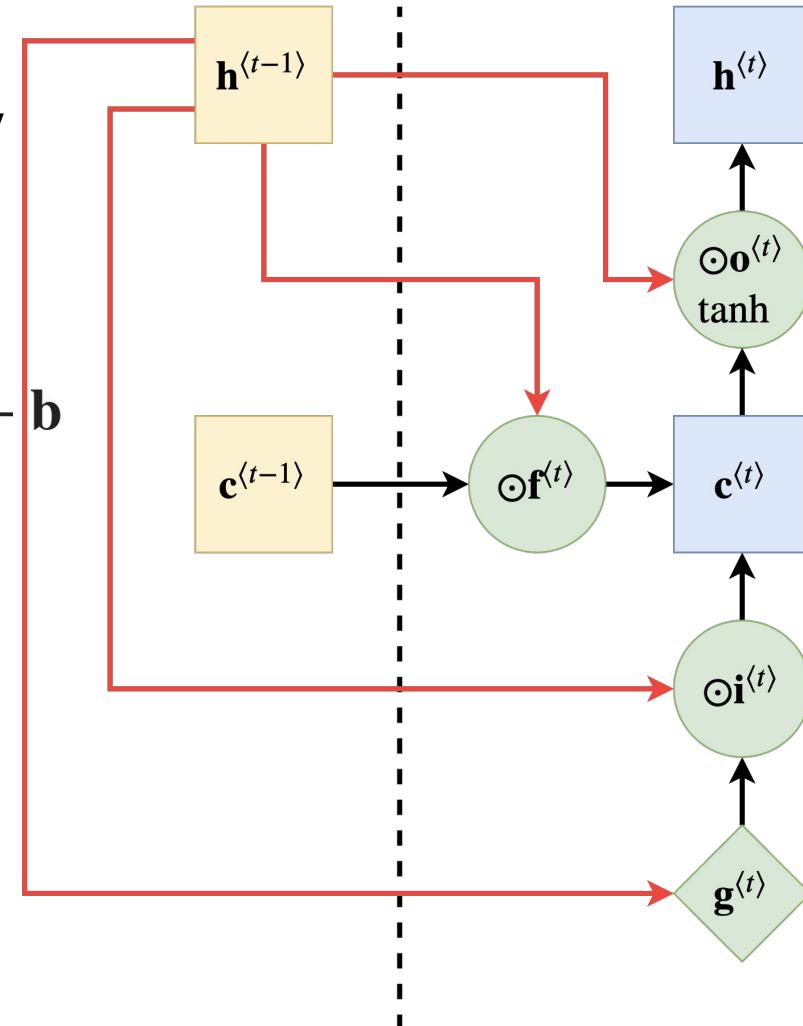
Hochreiter, S, and Schmidhuber, J. (1997)

# Long Short-Term Memory

$$\begin{pmatrix} \mathbf{i}^{(t)} \\ \mathbf{f}^{(t)} \\ \mathbf{o}^{(t)} \\ \mathbf{g}^{(t)} \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} \cdot \mathbf{W} \cdot \begin{pmatrix} \mathbf{h}^{(t-1)} \\ \mathbf{x}^{(t)} \end{pmatrix} + \mathbf{b}$$

$$\mathbf{c}^{(t)} = \mathbf{f} \odot \mathbf{c}^{(t-1)} + \mathbf{i}^{(t)} \odot \mathbf{g}^{(t)}$$

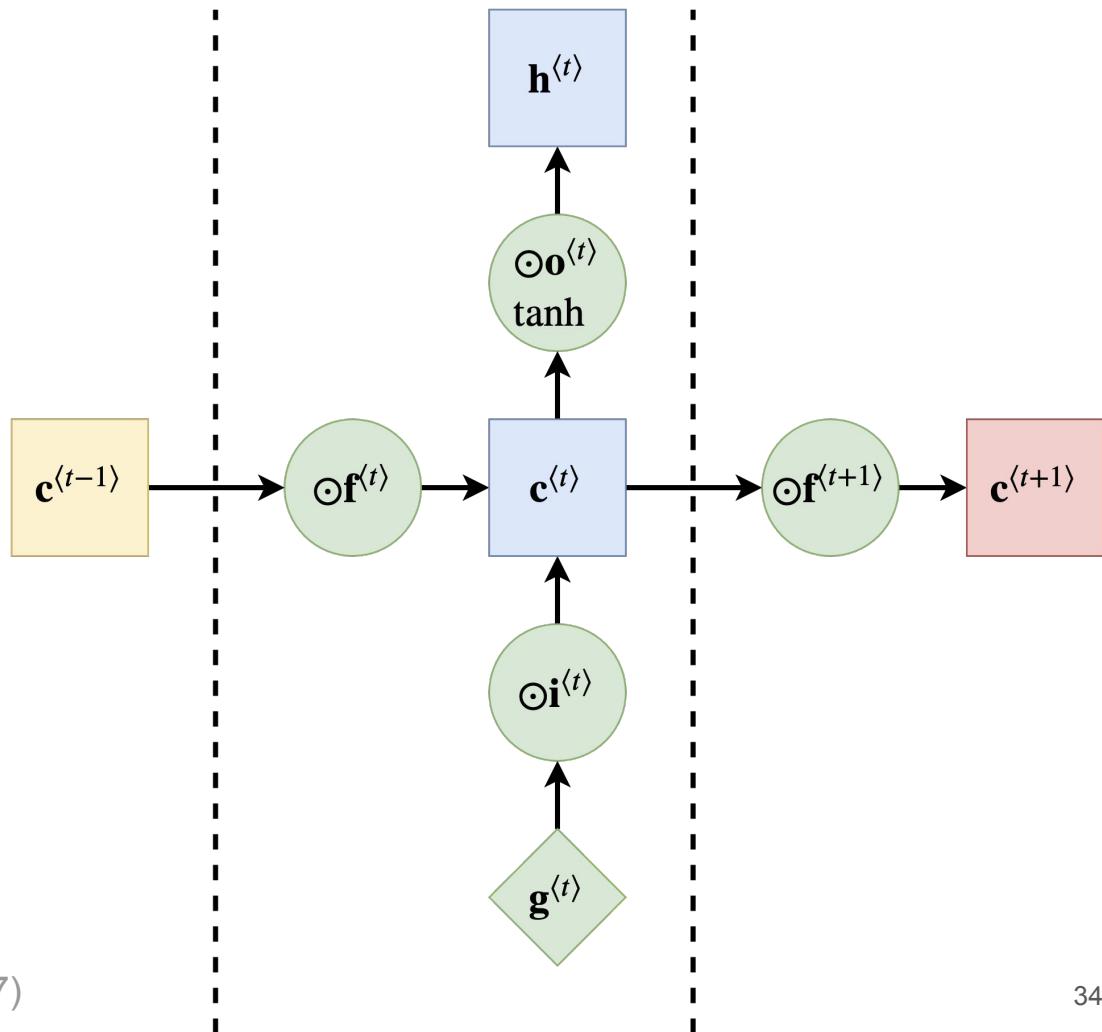
$$\mathbf{h}^{(t)} = \mathbf{o} \odot \tanh(\mathbf{c}^{(t)})$$



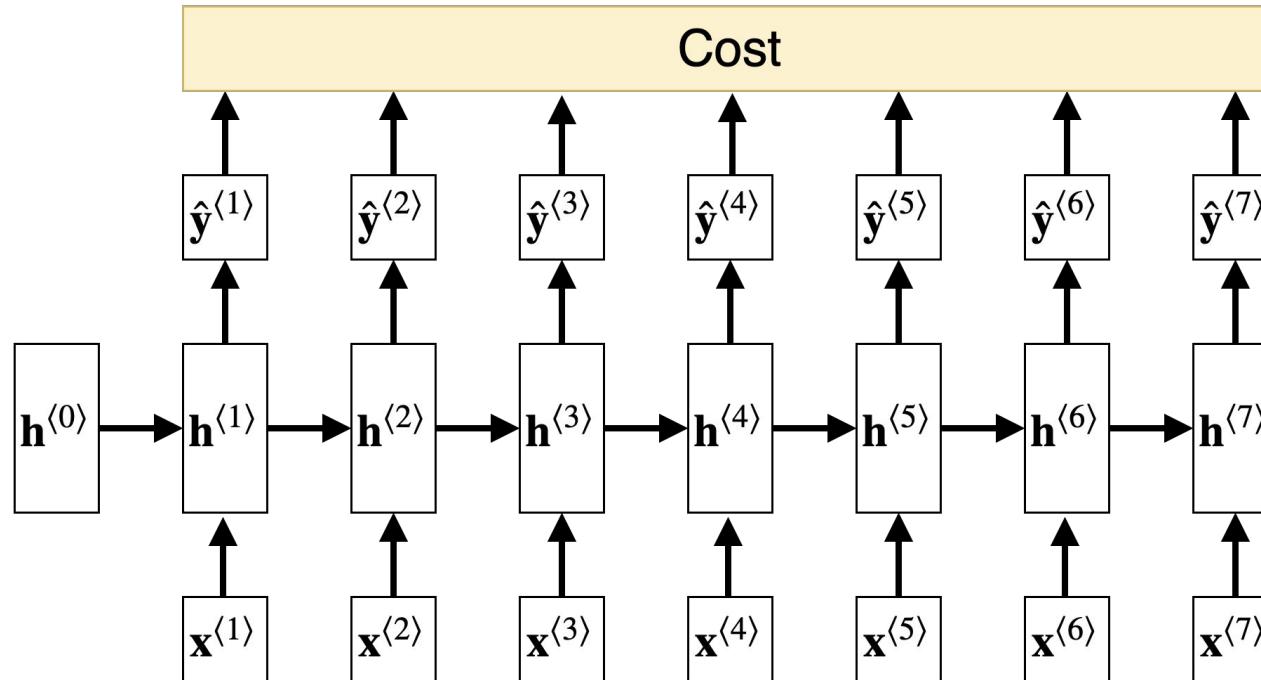
Hochreiter, S, and Schmidhuber, J. (1997)

Gradients flows further in LSTM than in the vanilla architecture due to:

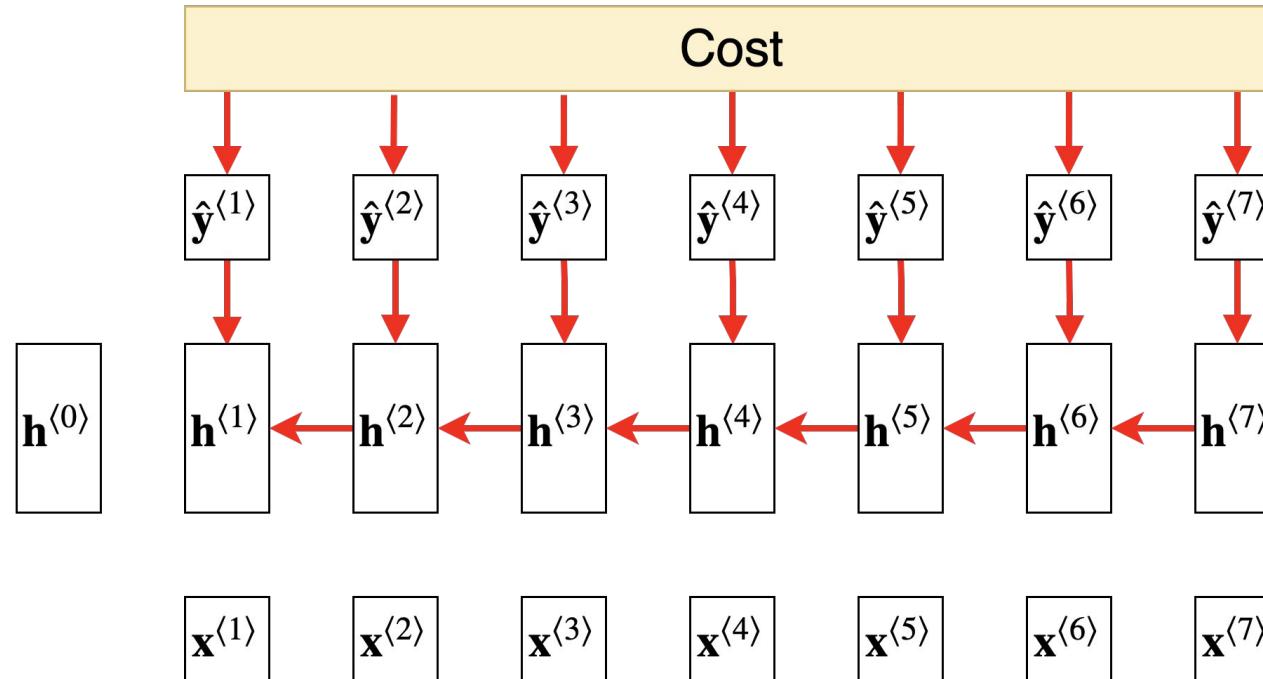
- 1) Between time steps, memory connected through an **element-wise product** rather than a dot product.
- 2) At each step, **forget mask varies**, while the weight matrix does not.



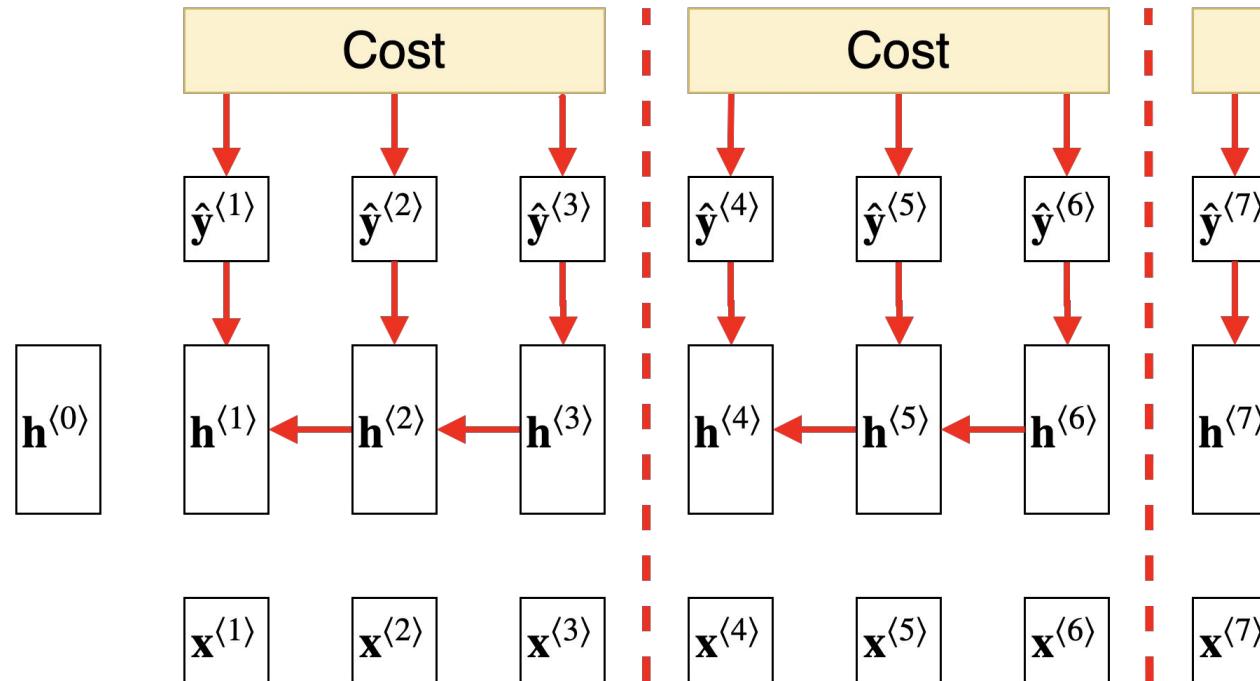
# Long Sequences: Forward Pass



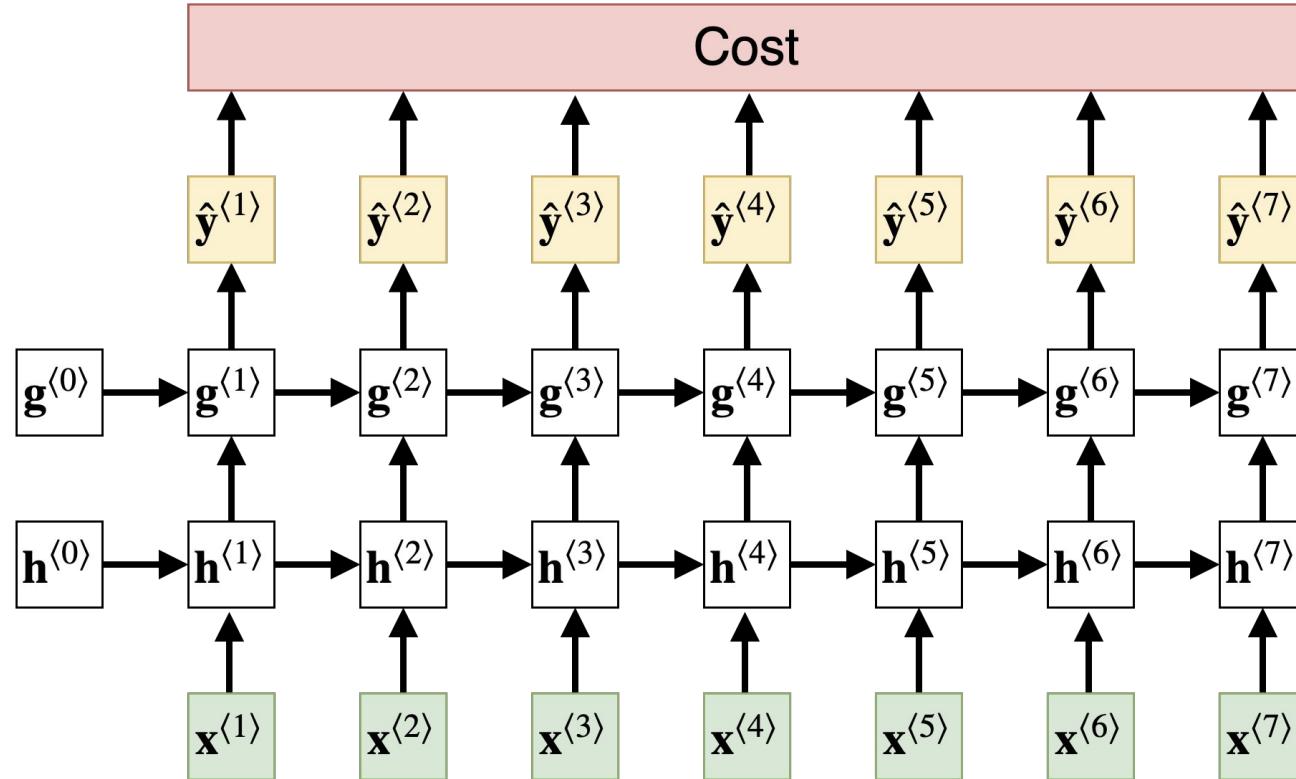
# Long Sequences: Backward Pass



# Long Sequences: Truncated Backprop



# Stacking Recurrent Layers

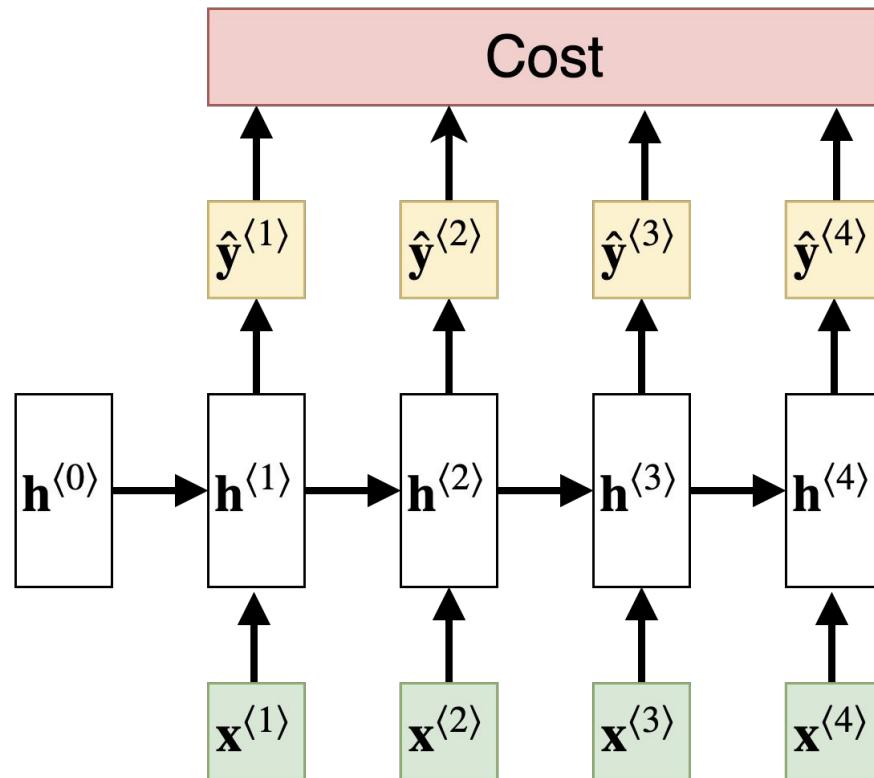


# Sequence to Sequence: Sync / Training

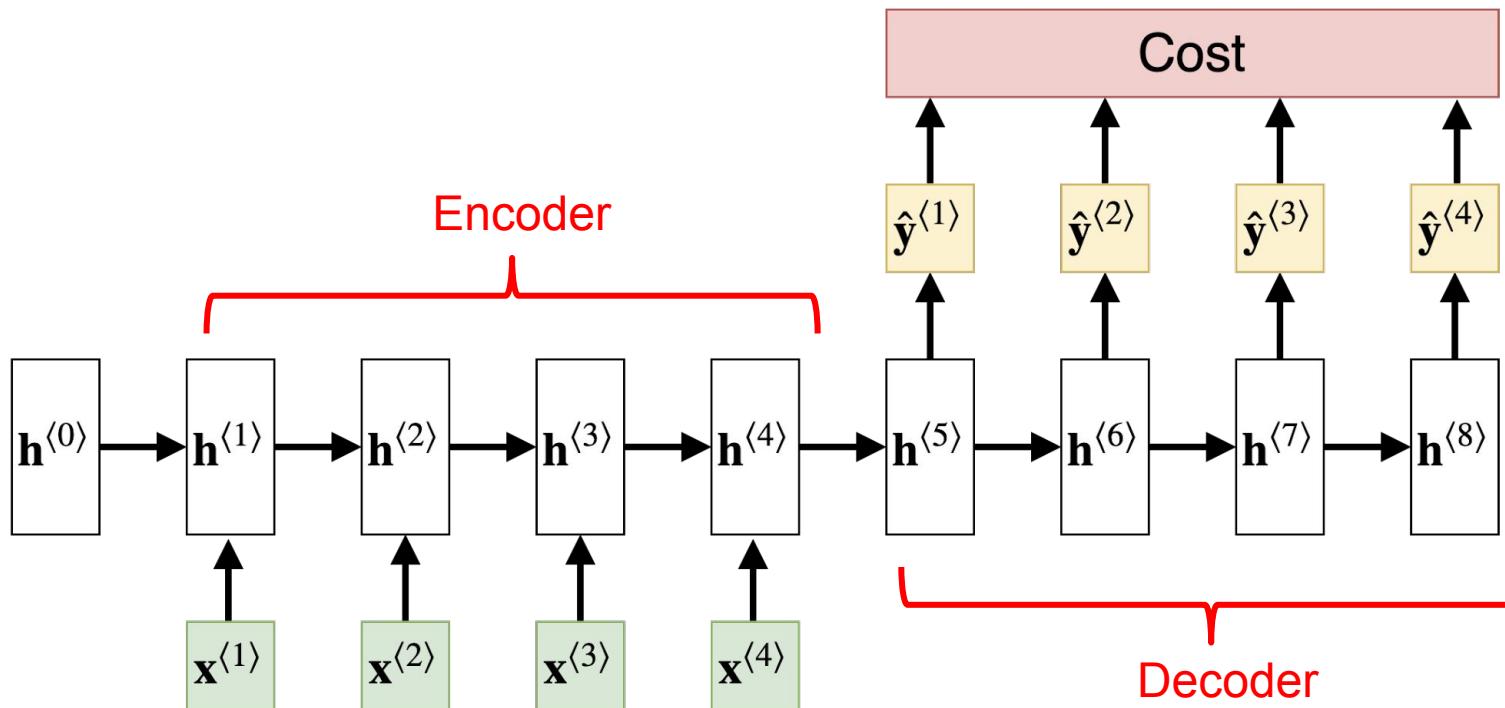
Requires to produce output at each time step, **without seeing the following inputs.**

Examples:

- Speech recognition
- Single-horizon forecasting



# Sequence to Sequence: Async



# Part 2

## DeepAR WeatherNet



(Image by a Stable Diffusion model)

# Case Study: DeepAR

## Task

Time-series forecasting

## Key Attributes

- Autoregressive model
- LSTM encoder/decoder
- Probabilistic output
- Multiple time series



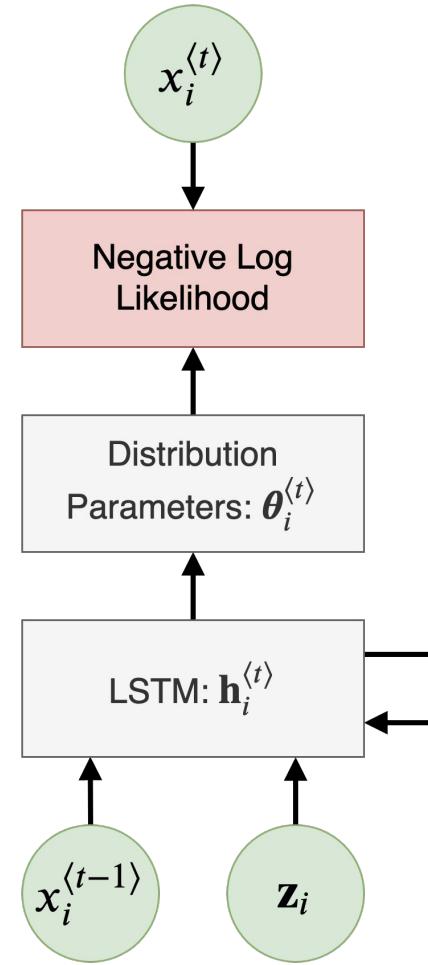
Available via AWS SageMaker and AWS Forecast.

Salinas, D., et al. (2019)

# DeepAR: Model

In training, the model learns to output **distribution parameters** that maximize the **likelihood of the next item** in the sequence's prediction window.

$$C = \sum_{i=1}^N \sum_{t=t_0}^{T_i} -\log \ell(x_i^{(t)} \mid \theta(\mathbf{h}_i^{(t)}))$$

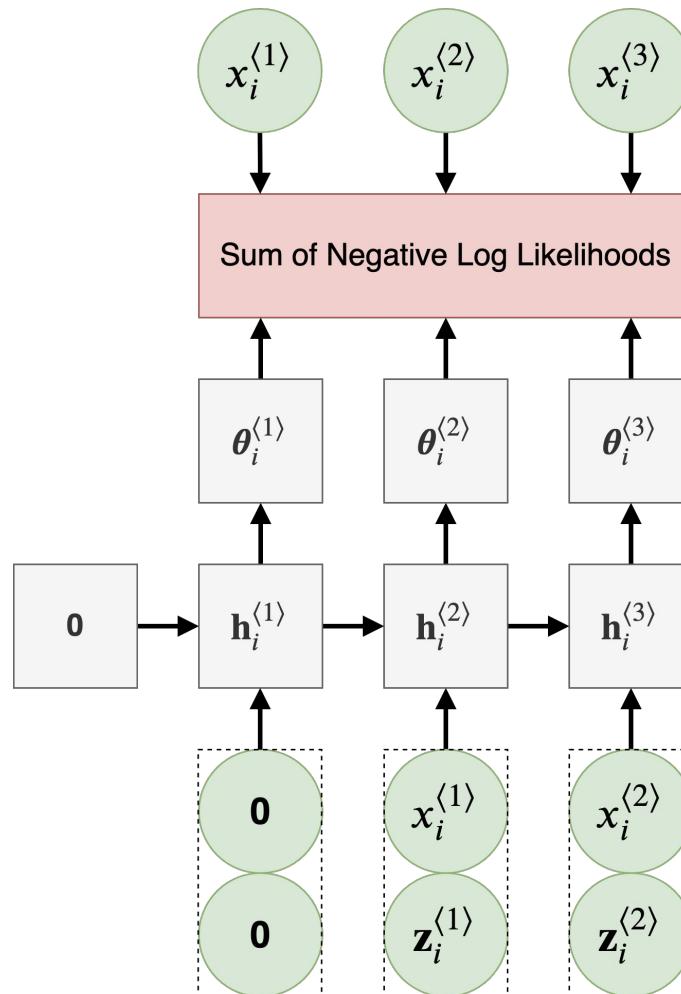


# DeepAR: Training

For each of N time series,  
**multiple start times** are randomly  
chosen (cutting sequences).

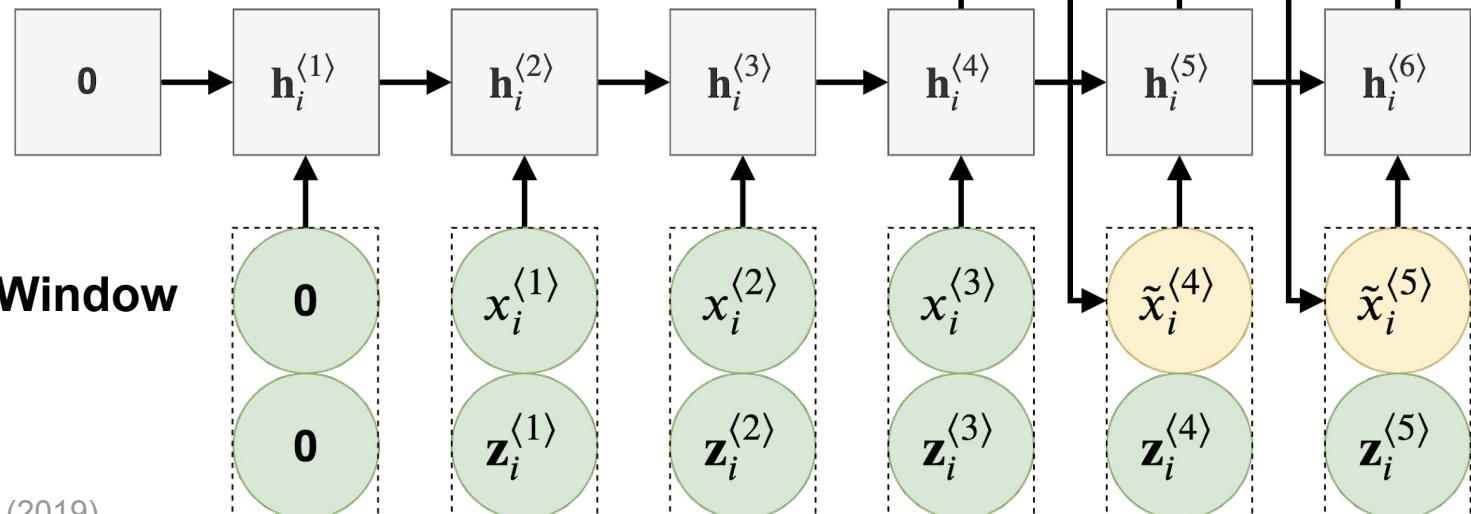
Some start times are negative,  
simulating the **cold start** problem  
during the inference.

All N time series are trained  
**simultaneously** across  
mini-batches.



# DeepAR: Inference

**Conditioning Window**  
3 periods



**Prediction Window**  
3 periods

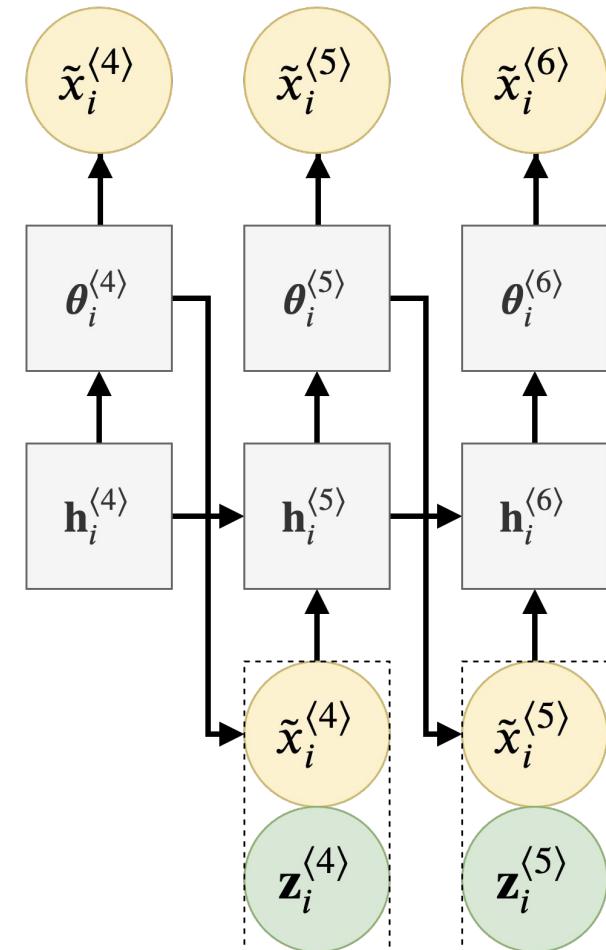
Salinas, D., et al. (2019)

# DeepAR: Ancestral Sampling

Ancestral Sampling for prediction window N:

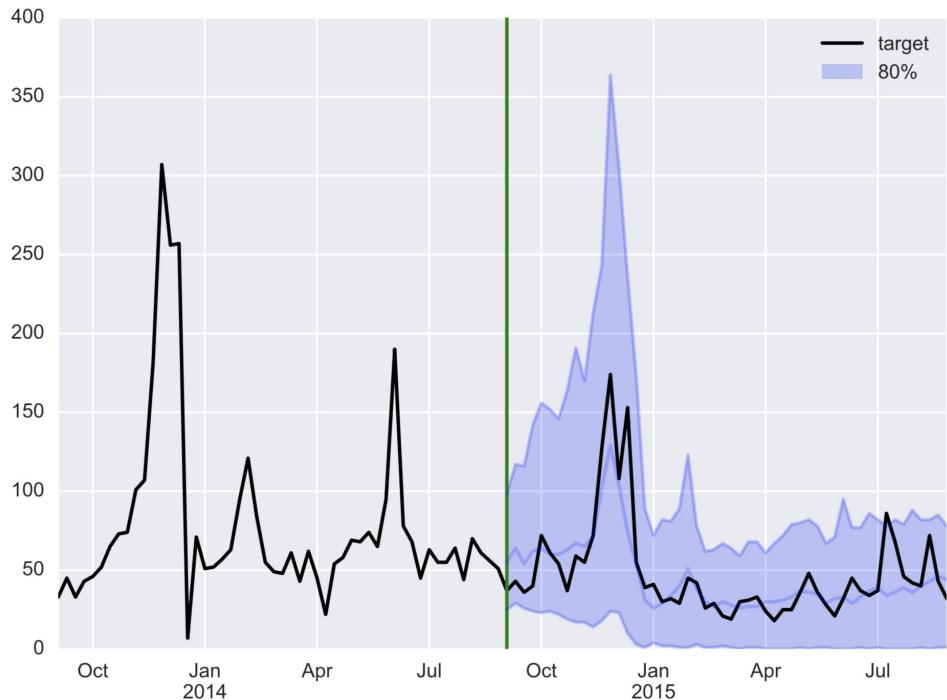
1. Randomly sample the next unknown item
2. Feed into the network as input
3. Repeat for N time steps

To estimate confidence intervals,  
sample many prediction sequences.

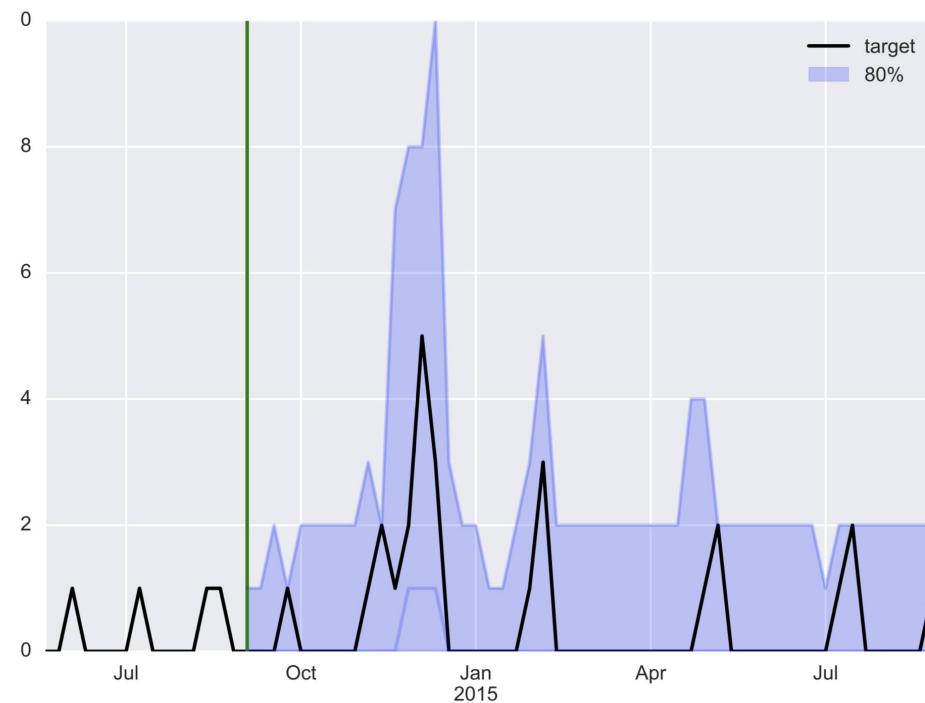


# DeepAR: Performance

Salinas, D., et al. (2019)

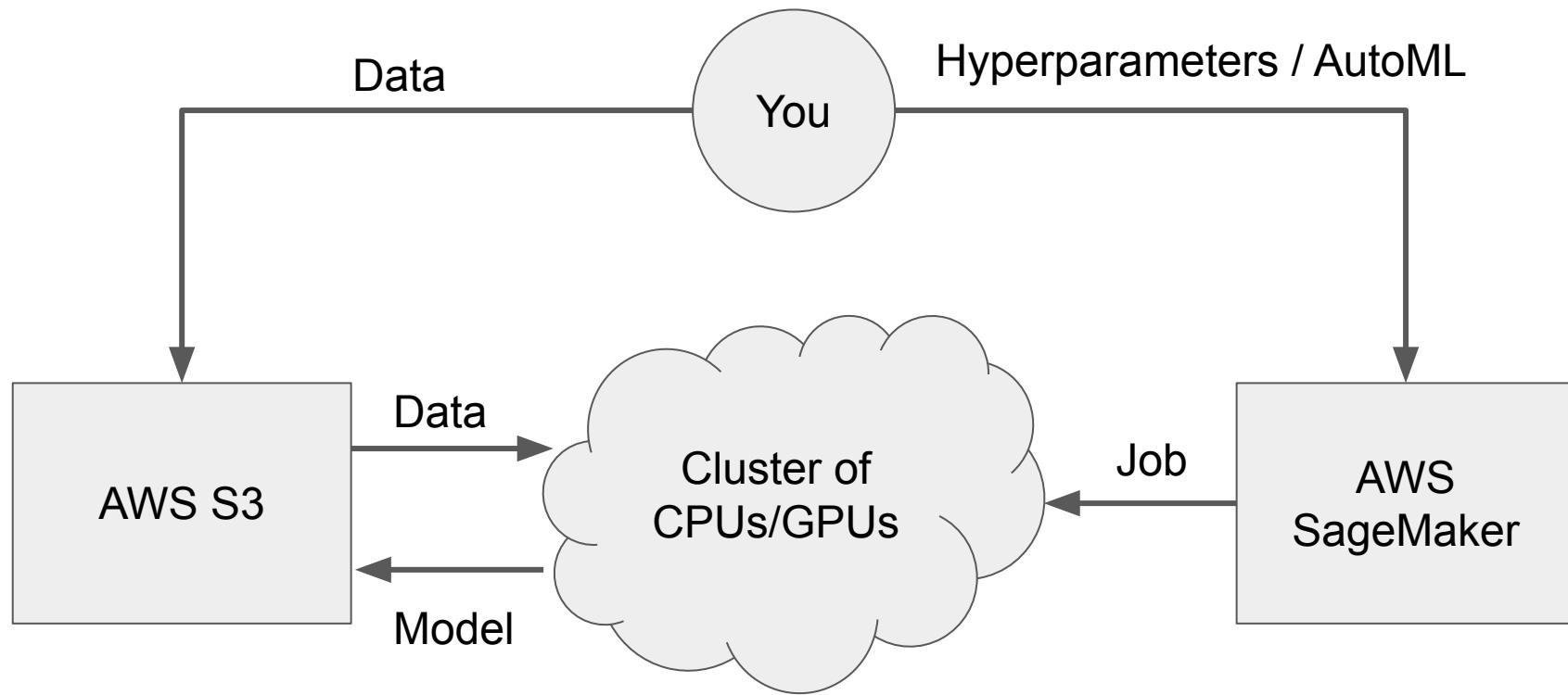


High-demand / warm-start

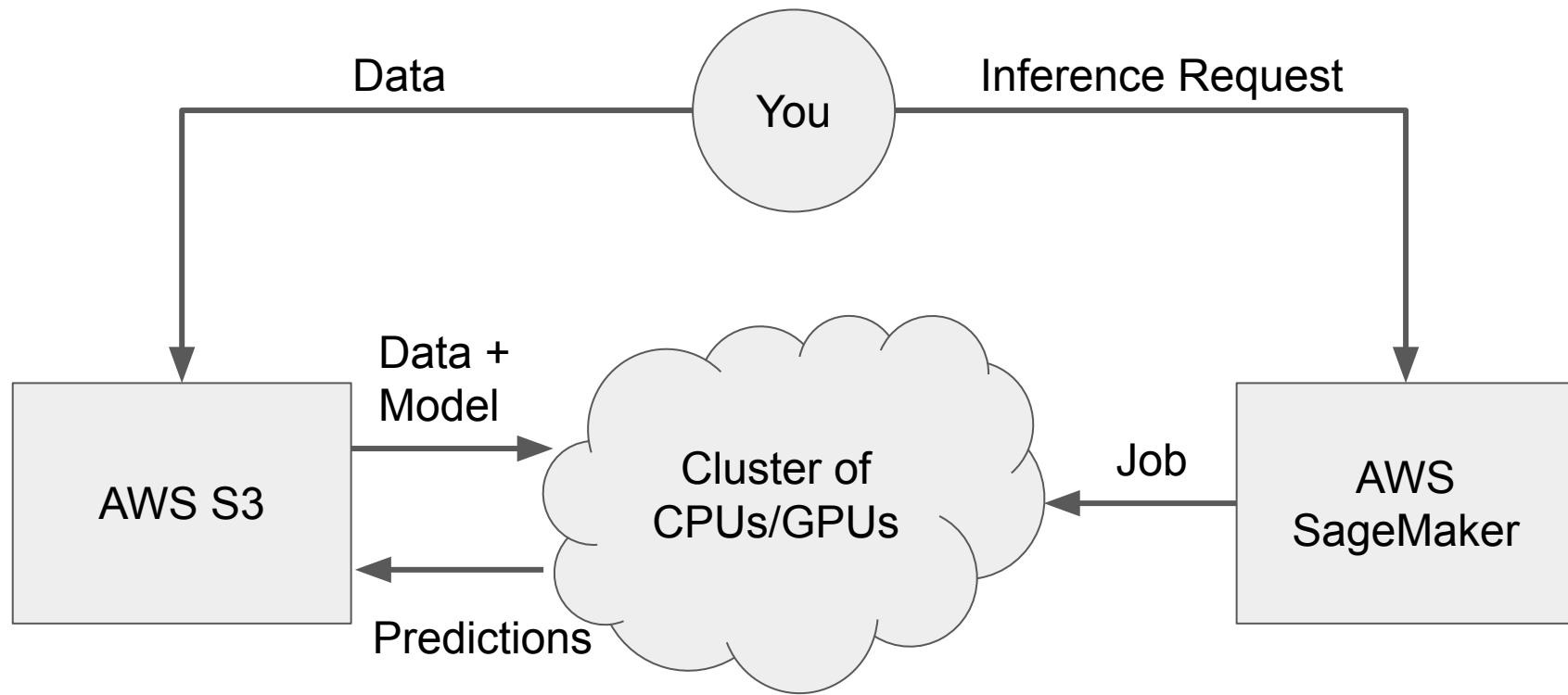


Low-demand / cold-start

# DeepAR: Training in AWS



# DeepAR: Batch Inference in AWS



# Case Study: Weather Forecasts

## Dataset

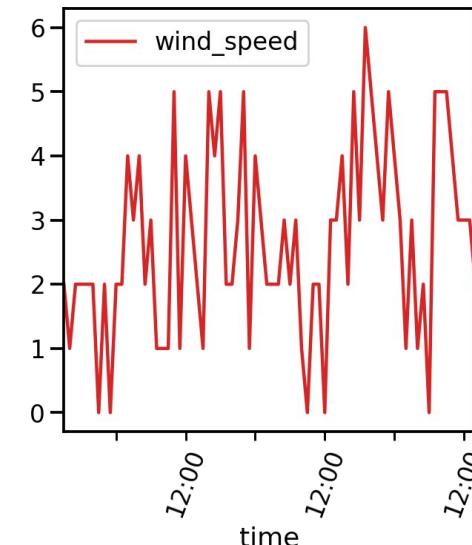
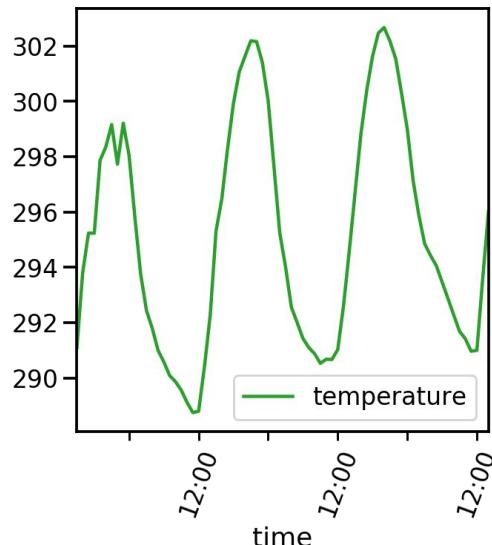
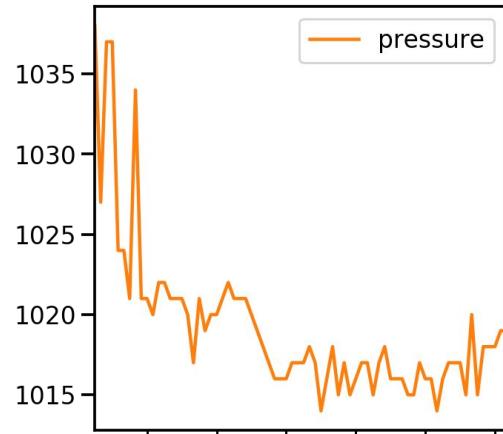
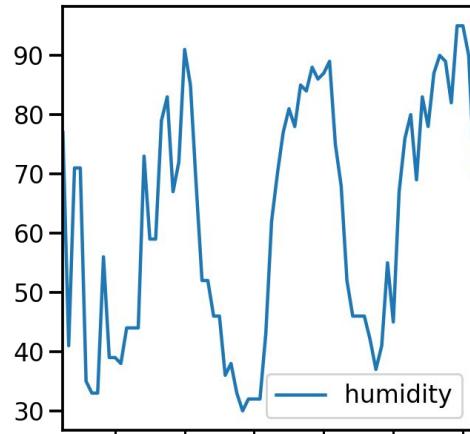
Hourly frequency

36 cities across the world

5 years of non-continuous recordings

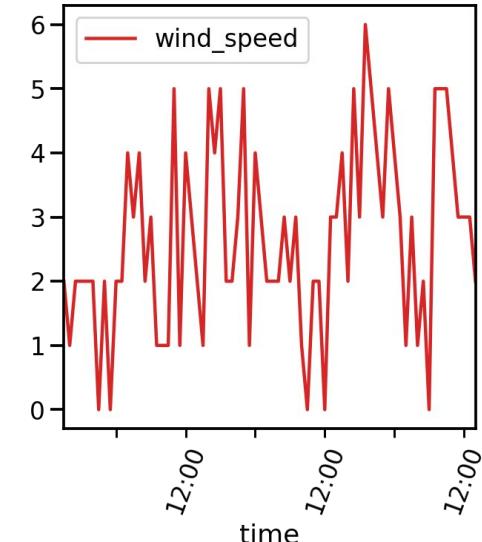
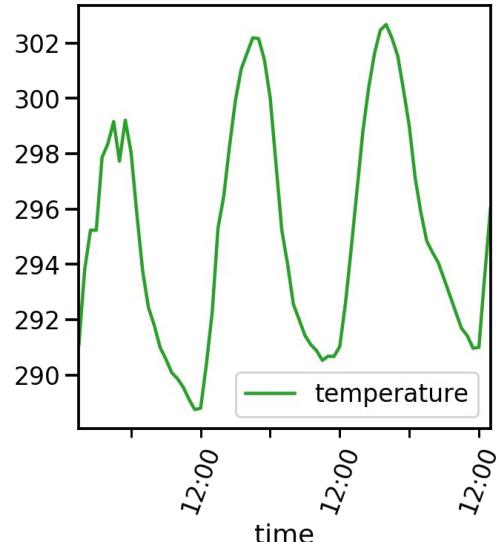
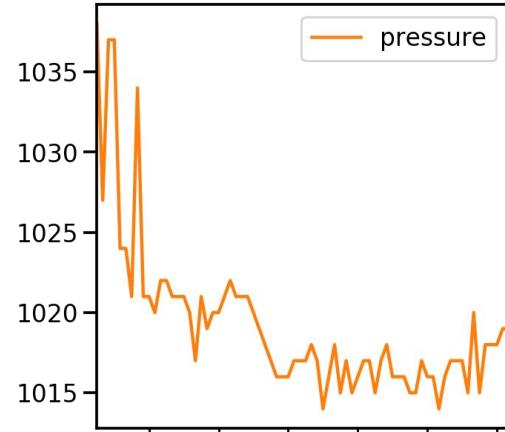
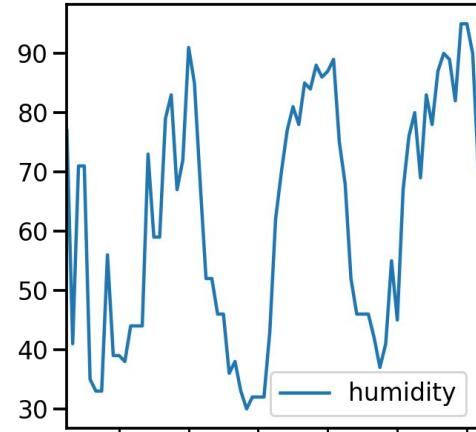
## Features

Temperature, humidity, atmospheric pressure, wind speed



# Processing Pipeline

1. Handle missing data
2. Split into segments
3. Build condition and prediction windows
4. Train baseline models
5. Train RNN model
6. Evaluate results



# Weather: Data Quality

1. Marked no-change observation as N/A
2. Forward-padded N/A observations up to 3 hours
3. Removed remaining N/A observations

		feature	humidity	pressure	temperature	wind_speed
	sample	time				
Albuquerque/1108/0	2015-02-16 09:00:00		47.0	1031.0	276.323000	3.0
	2015-02-16 10:00:00		45.0	1032.0	275.097667	3.0
	2015-02-16 11:00:00		48.0	1032.0	273.931333	2.0
	2015-02-16 12:00:00		46.0	1033.0	274.380000	2.0
	2015-02-16 13:00:00		51.0	1033.0	272.675333	1.0

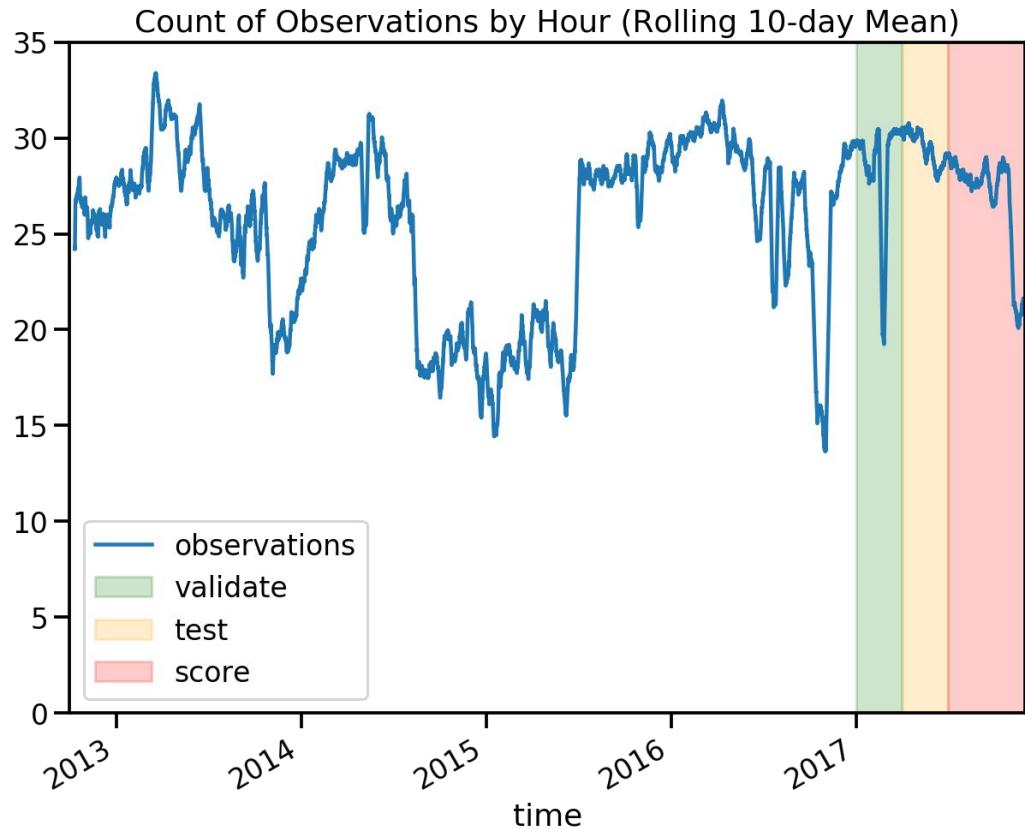
# Weather: Data Splits

Simple time-based split

**Approximate ratios**

80% / 5% / 5% / 10%

	observations	%	date_min	date_max
train	939344	82	2012-10-01	2016-12-31
validate	60785	5	2017-01-01	2017-03-31
test	63502	5	2017-04-01	2017-06-30
score	96635	8	2017-07-01	2017-11-30



# Weather: Batches

Source Sequence = 6

Rolling Window

1	2	3	4	5	6
---	---	---	---	---	---

## Condition Features

Temperature, pressure,  
humidity, wind speed

## Prediction Variables

Temperature

Condition Window = 2

1	2
---	---



3	4
---	---

2	3
---	---



4	5
---	---

3	4
---	---



5	6
---	---

Examples = 3

# Weather: Batches

## Condition Window

24 hours

## Prediction Window

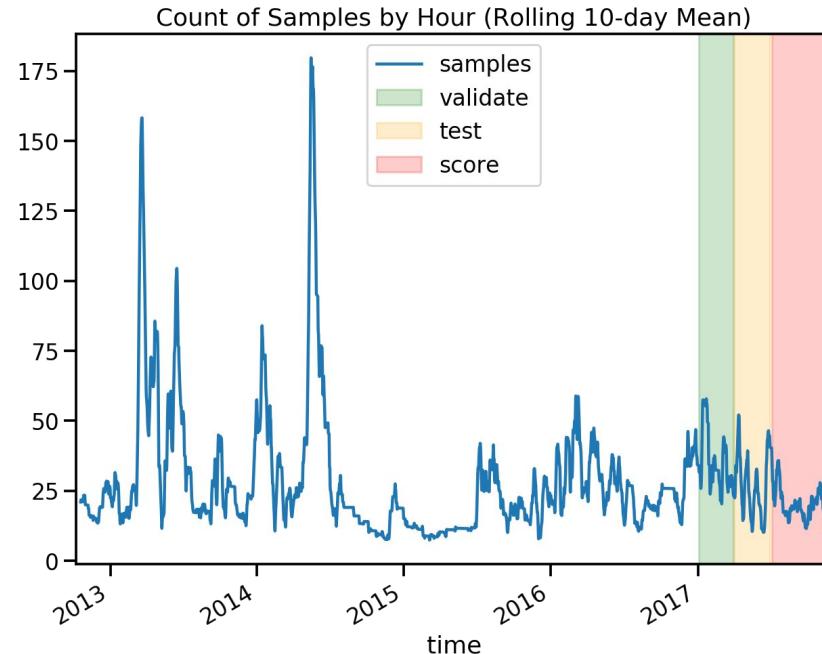
24 hours

## Minimum Sequence Length

48 hours

## Fixed Sample Length

48 hours



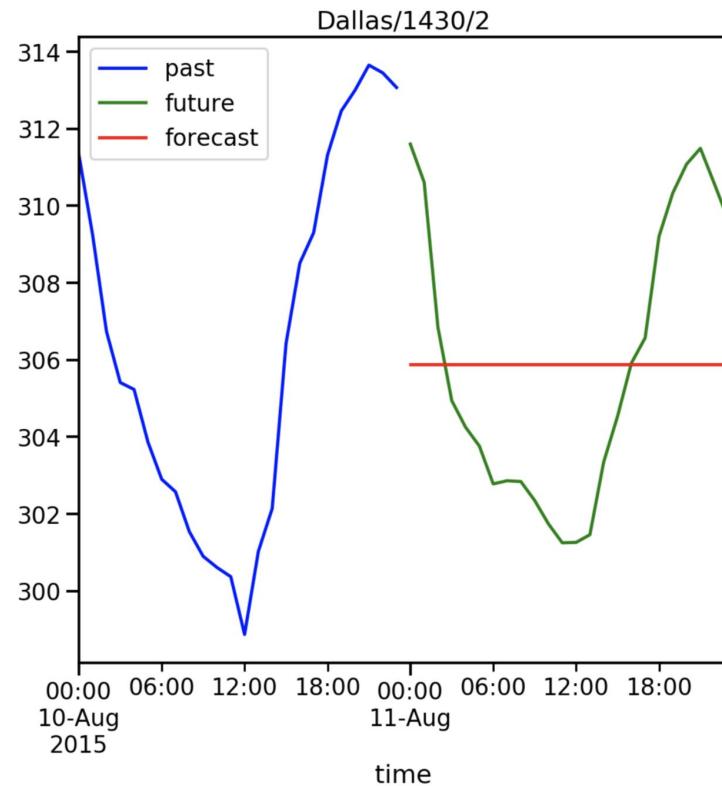
	samples	%	date_min	date_max
train	34436	83	2012-10-08	2016-12-30
validate	2288	6	2017-01-02	2017-03-30
test	2177	5	2017-04-01	2017-06-29
score	2399	6	2017-07-02	2017-11-29

# Weather: Baseline Models

## Mean Model

Forecast = mean of the past 24 hours

	MAE	ME	MSE	R2
<b>train</b>	4.00	-0.49	26.77	0.80
<b>validate</b>	4.62	-0.98	37.23	0.58
<b>test</b>	4.22	-1.07	28.52	0.61
<b>score</b>	4.30	-1.43	28.70	0.78

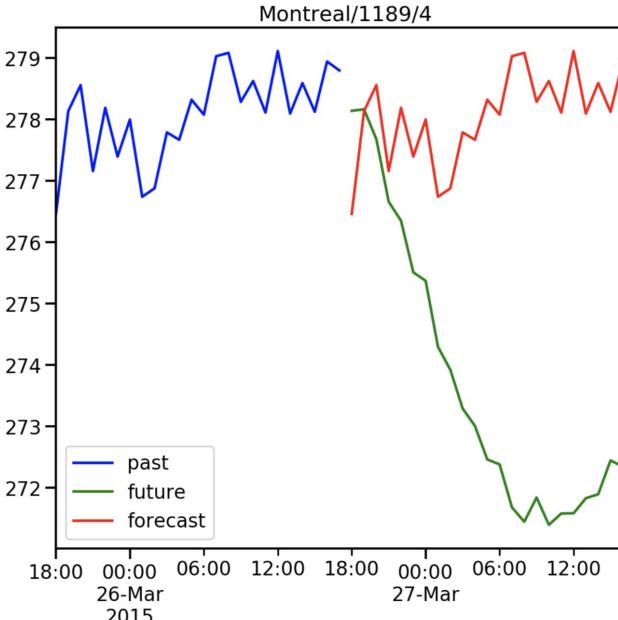
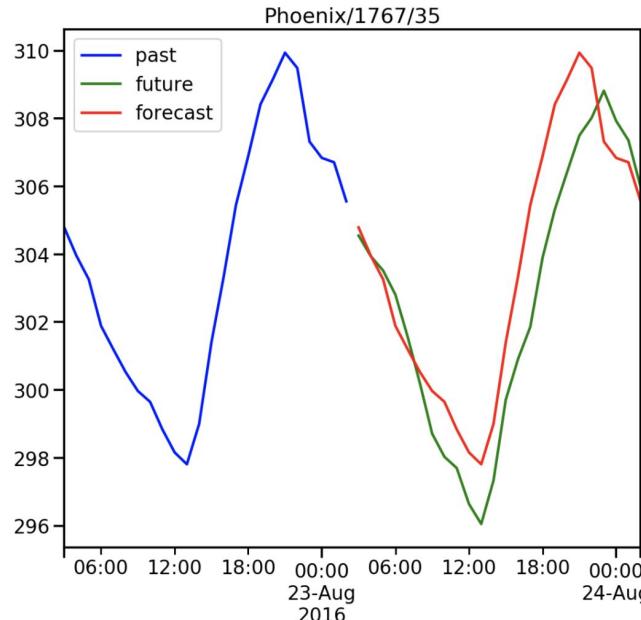


# Weather: Baseline Models

## Constant Model

Forecast = replay of the past 24 hours

	MAE	ME	MSE	R2
train	3.26	-0.49	21.82	0.84
validate	4.76	-0.98	40.25	0.54
test	3.20	-1.07	20.20	0.72
score	3.29	-1.43	22.31	0.83

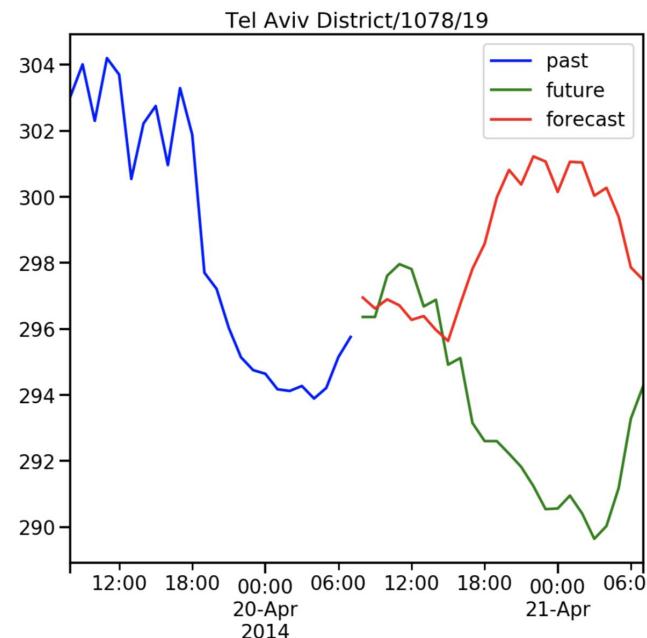
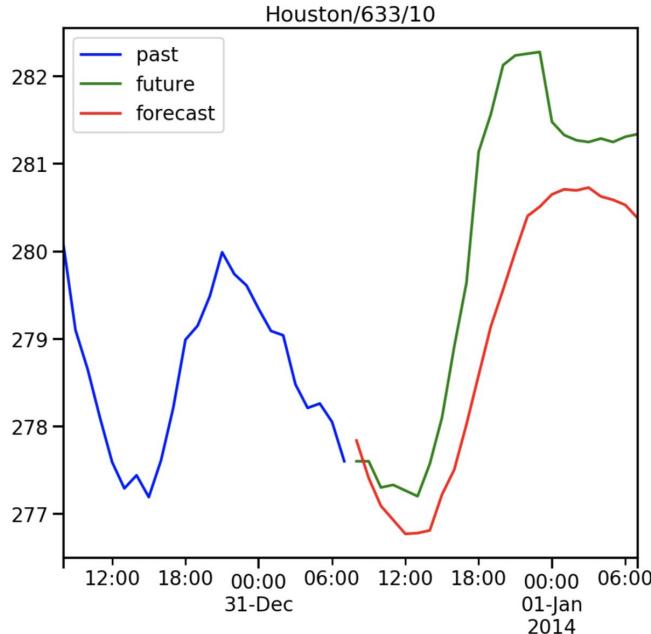


# Weather: Baseline Models

## Regression Model

24 independent linear regressions

	MAE	ME	MSE	R2
train	4.21	0.00	30.53	0.78
validate	4.04	-0.24	29.94	0.66
test	4.51	0.15	33.64	0.54
score	4.49	-0.26	33.45	0.75



# Weather: RNN

## Encoder

Stacked GRUs

32 / 16 units

## Decoder

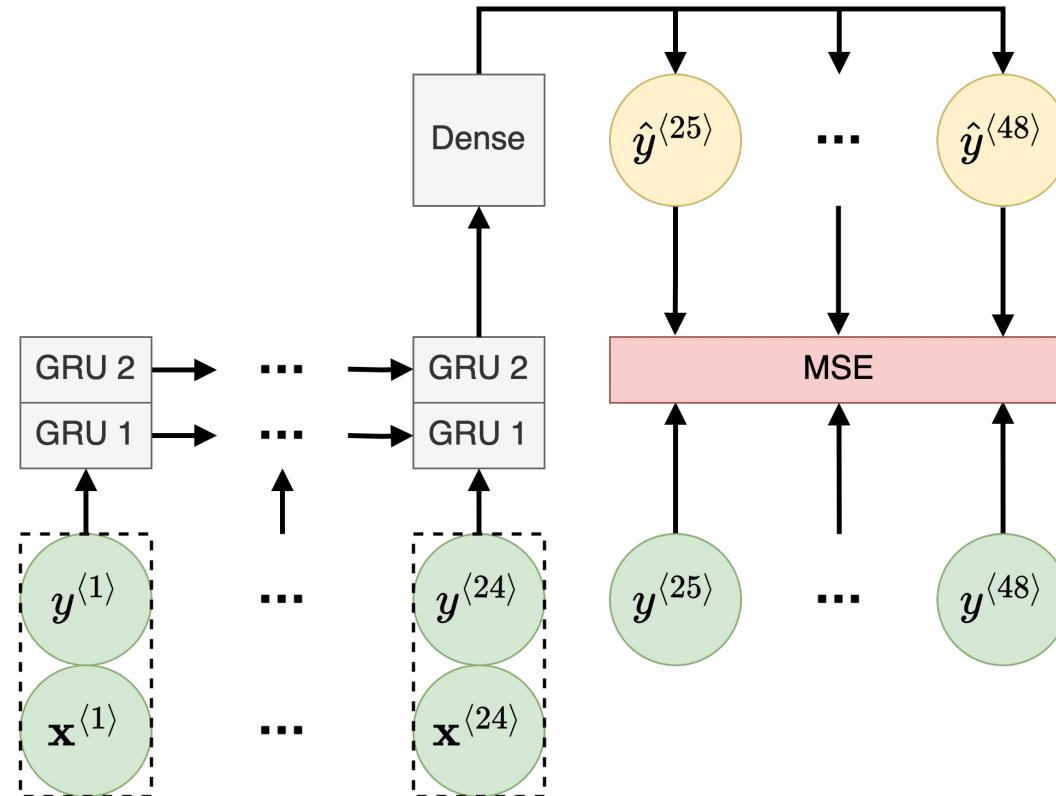
Dense

## Parameters

~6K

## Loss

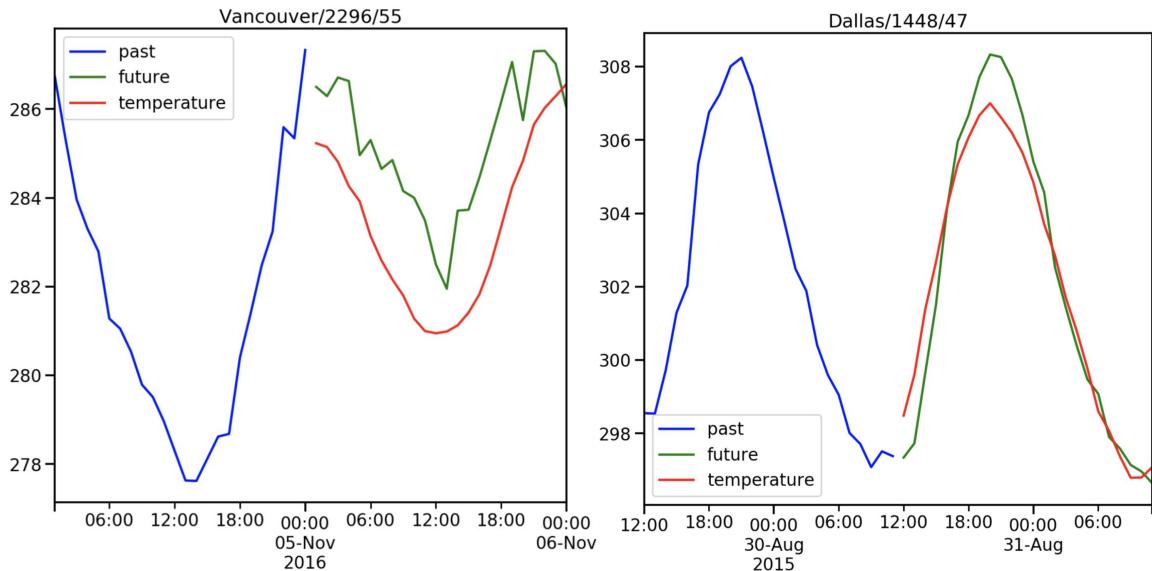
Mean Squared Error



# Weather: RNN

## Inference

1. RNN model  
beats all  
baseline models.



2. The model is  
overfitting on the  
train segment.

	train		validate	
	MSE	R2	MSE	R2
<b>mean</b>	26.77	0.80	37.23	0.58
<b>constant</b>	21.82	0.84	40.25	0.54
<b>regression</b>	30.53	0.78	29.94	0.66
<b>rnn</b>	15.73	0.89	20.37	0.77

# Assignment #3

## Methodology

- Build and train RNN model on the weather forecasting task
- Try LSTM/GRU cell types
- Try deeper encoder / decoder
- Try L1 / L2 regularization
- Try adding calendar / location features
- **Can't submit exactly the same** model as the example
- **Must document changes** to the training code

## Grading

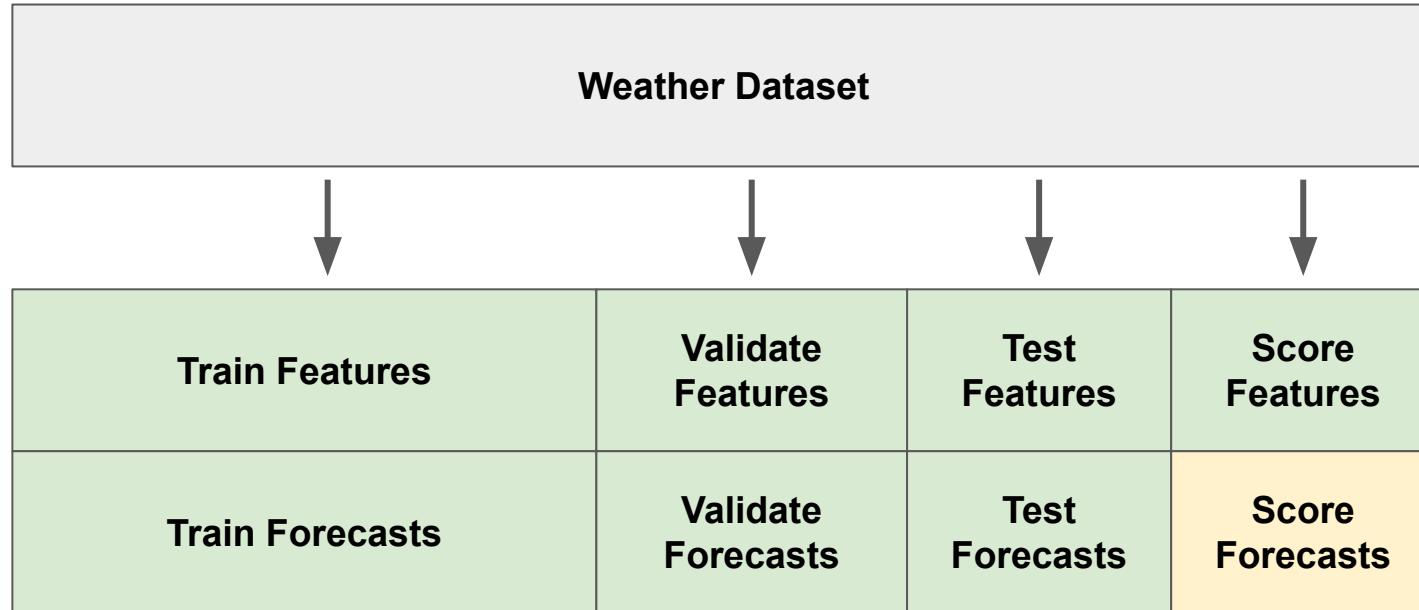
- 10 points for **R2** on Score segment  $> 0.85$
- 10 points **proportionally to the percentile** versus other submissions

# Assignment #3

## Deliverables

1. Jupyter notebook and/or Python script
  - o Data preprocessing
  - o Construction and training of the final model
  - o Inference on the Score dataset
2. Final model definition
  - o Keras serialization in JSON format
3. Final model parameters
  - o Keras serialization in H5 format
4. Temperature forecasts for the Score segment
  - o Shape = 57576 x 1
  - o Data Type = float32
  - o File Type = Parquet

# Assignment Dataset



# Demo

AssignmentTemplate.ipynb

# Case Study: MQ-RNN

## Task

Time-series forecasting

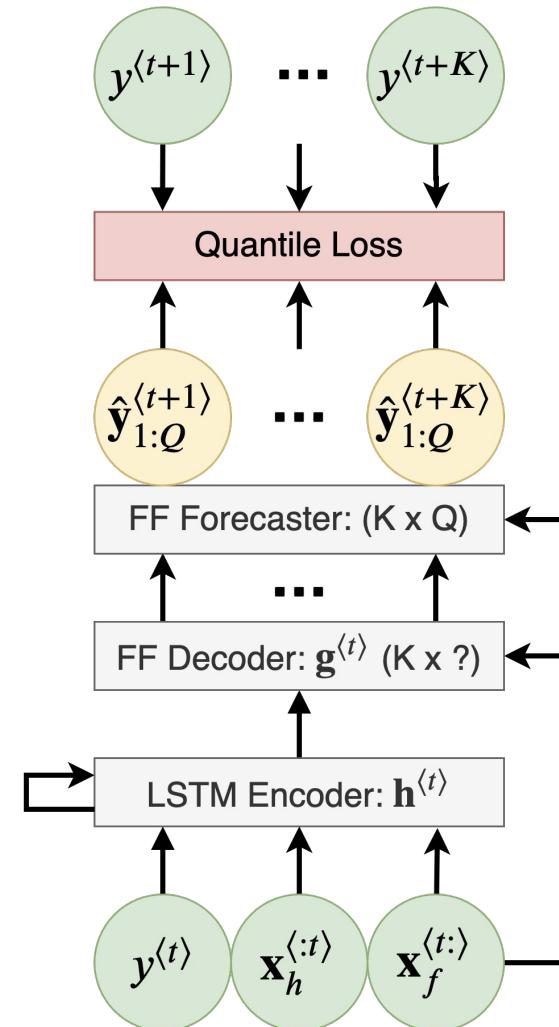
## Key Attributes

- Autoregressive model
- LSTM encoder
- Dense decoder
- Quantile output
- Multiple time series

Time	1	2	3	4	5	6
Window	Conditioning			Prediction		
Horizon				1	2	3
Target						
q = 25						
q = 50						
q = 75						

# MQ-RNN: Model

- Decoder summarizes **horizon-specific context vector** for each of K horizons
- Forecaster outputs prediction for **each of K horizons and Q quantiles**
- Forecaster **shares parameters** across horizons, but not across quantiles

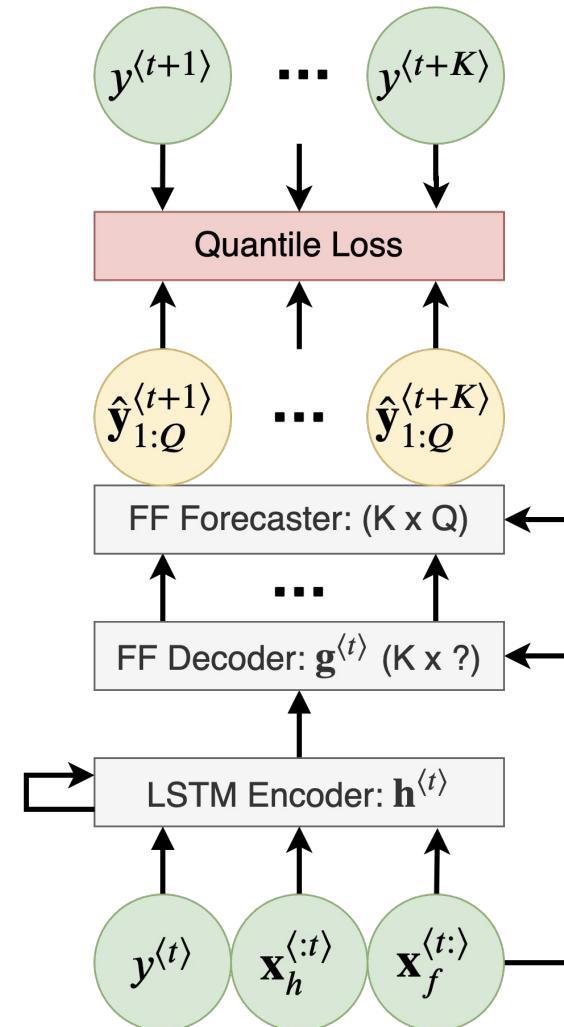


# MQ-RNN: Model

$$\begin{aligned}\mathbf{g}^{(t)} &= (\mathbf{c}^{(t+1)}, \dots, \mathbf{c}^{(t+K)}, \mathbf{c}_a^{(t)}) \\ &= g(\mathbf{h}^{(t)}, \mathbf{x}_f^{(t,:)})\end{aligned}$$

$$\begin{aligned}\hat{\mathbf{y}}^{(t+k)} &= (\hat{y}_1^{(t+k)}, \dots, \hat{y}_Q^{(t+k)}) \\ &= f(\mathbf{c}^{(t+k)}, \mathbf{c}_a^{(t)}, \mathbf{x}_f^{(t+k)})\end{aligned}$$

Dense decoder looks at **all horizons**, while Dense forecaster looks at **one horizon** at a time.

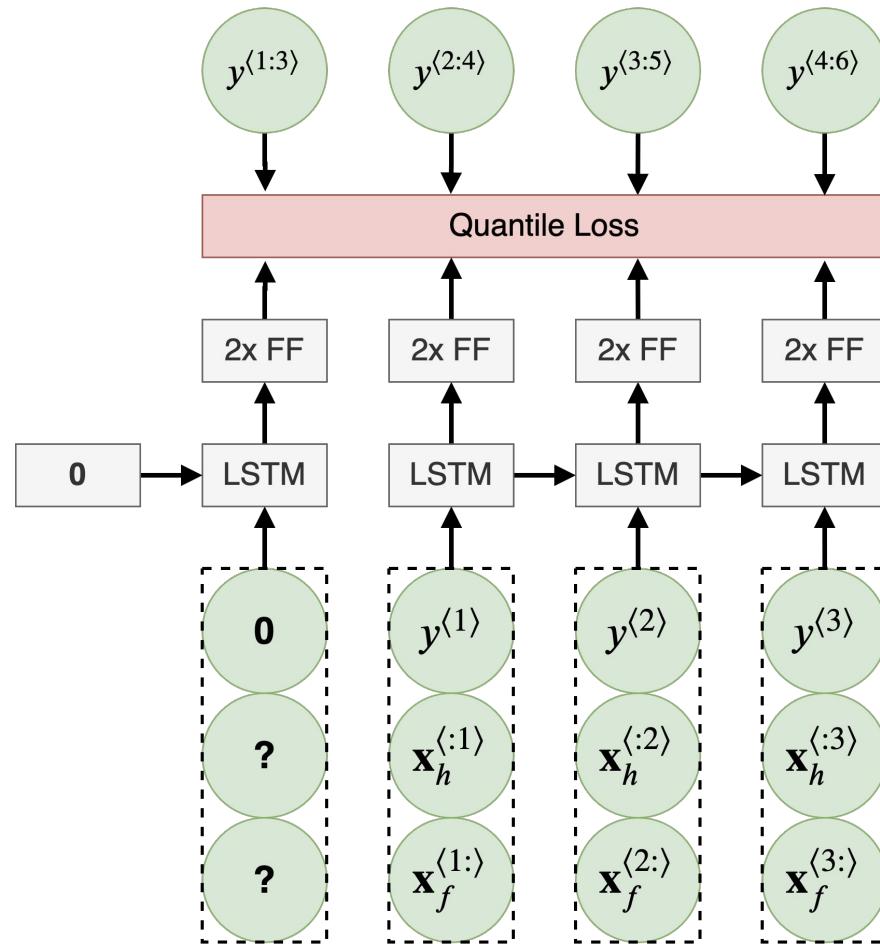


# MQ-RNN: Training

**Conditioning Window**  
Full trailing history

**Prediction Window**  
3 periods

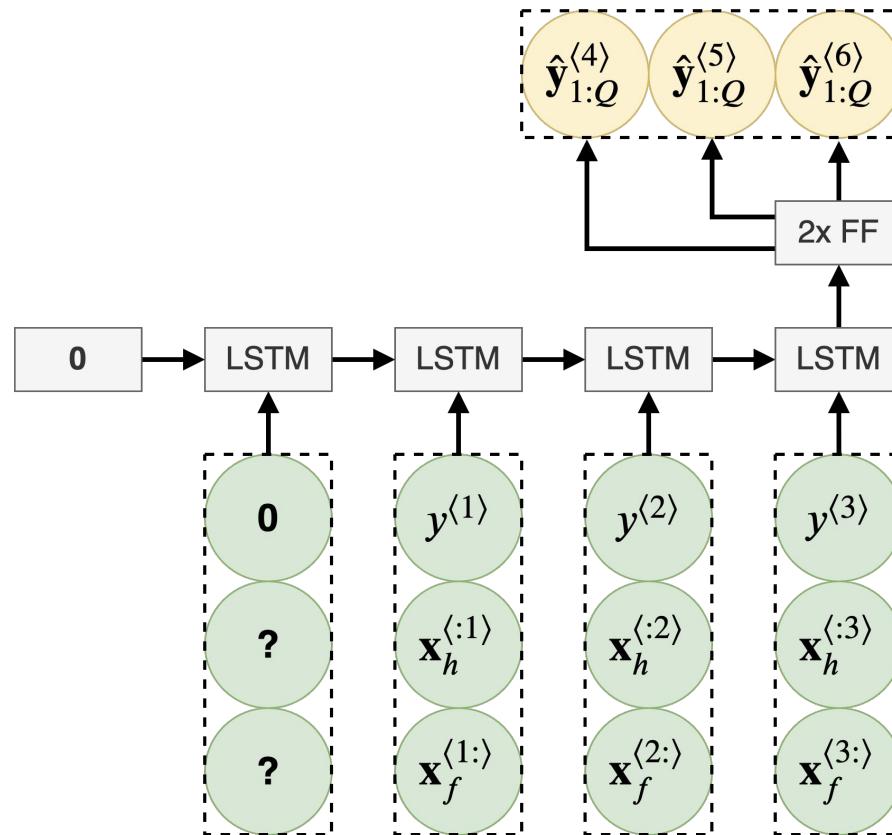
**Forking Sequences**  
Unlike cutting sequences approach of DeepAR, the model outputs multi-horizon prediction at each step.



# MQ-RNN: Inference

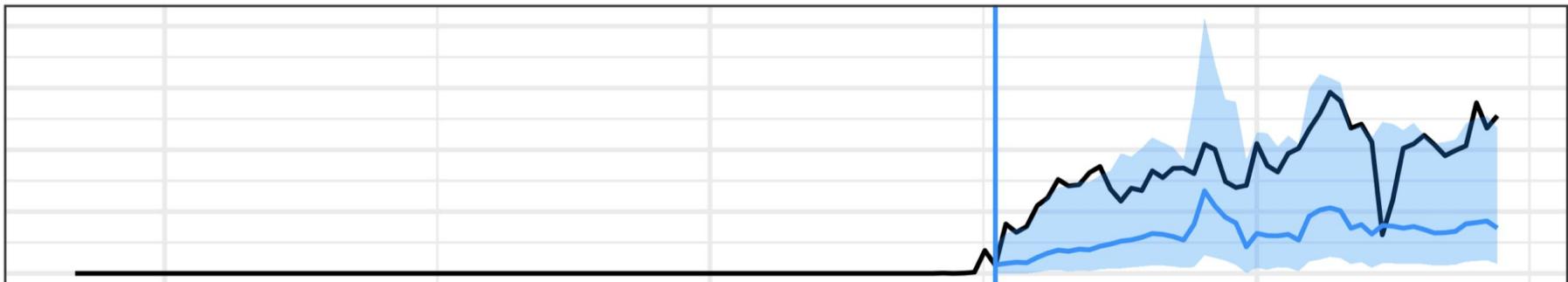
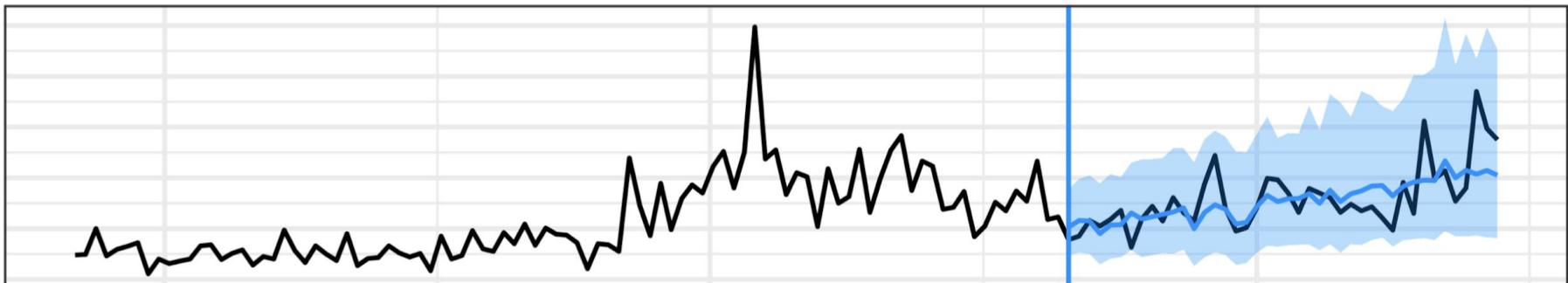
**Conditioning Window**  
Full trailing history

**Prediction Window**  
3 periods



# MQ-RNN: Performance

**Warm start**



# Part 3

## Attention Layer Temporal Transformer



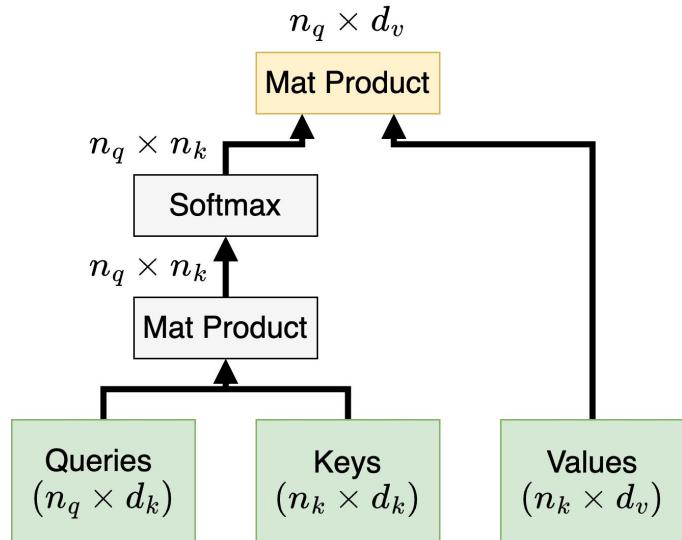
(Image by a Stable Diffusion model)

# Attention

- **Key** ~ a question about the input
- **Value** ~ the answer to a question
- **Query** ~ a question about the input

Output = a linear combination of values  
**weighted by similarity** of keys to queries.

Compute Cost =  $O(n^2)$

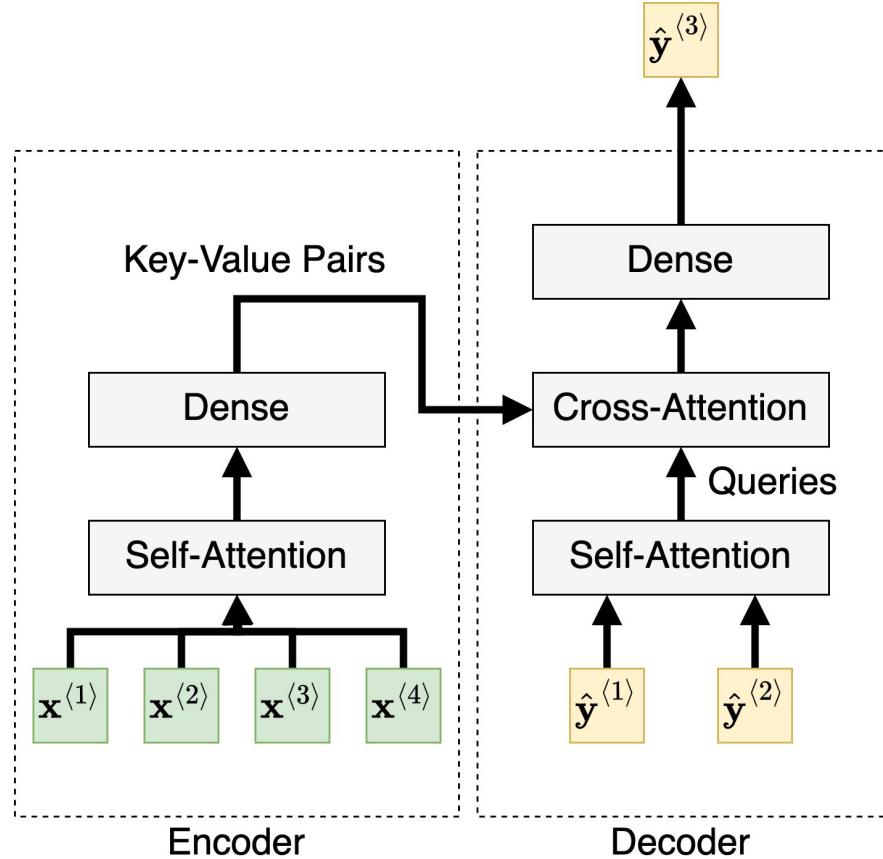


# Transformer

Encoder + Decoder

Example:

- Input
  - Full sequence
  - 4 tokens
- Output
  - Past sequence
  - 2 tokens



# Further Reading

- Reformer (Nikita, K., et al., 2020)
- Informer (Zhou, H, et al., 2021)
- Autoformer (Wu, H., et al., 2022)

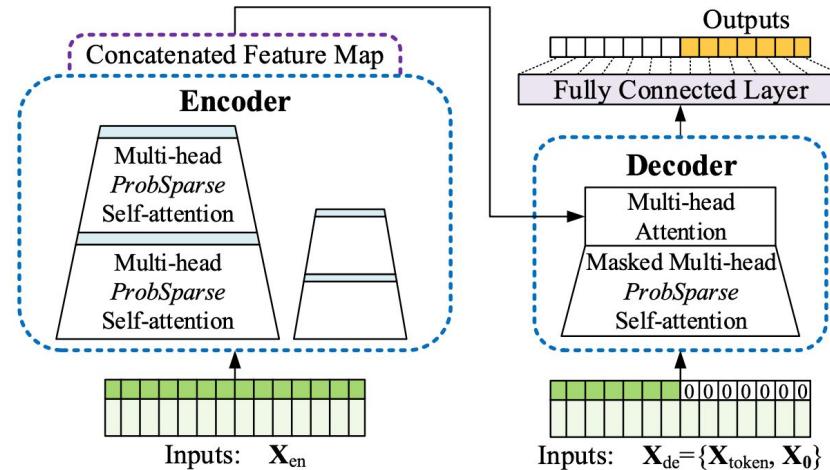
Model	MSE (ETT dataset)
Autoformer	0.255
Informer	0.365
LogTrans	0.768
Reformer	0.658
LSTNet	3.142
LSTM	2.041
TCN	3.041

(Wu, H., et al, 2023)

# Informer

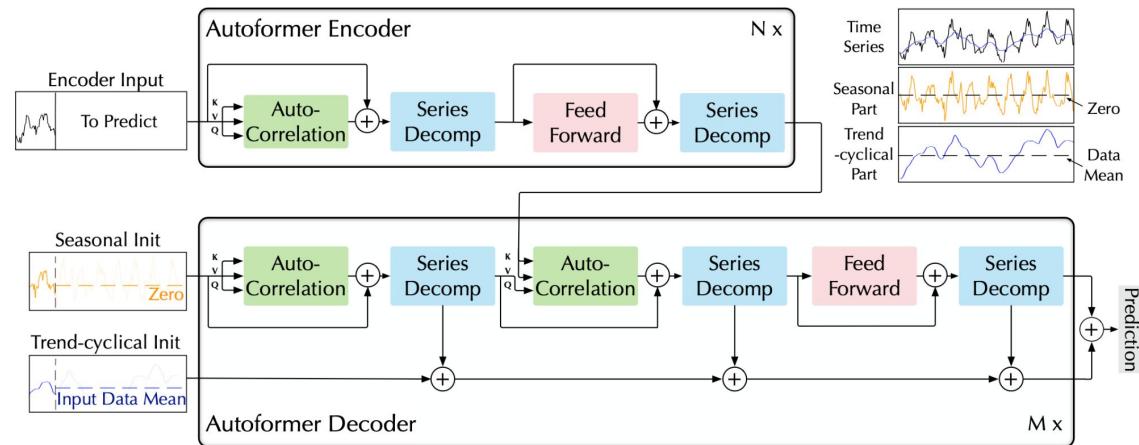
ProbSparse attention reduces the compute cost to  $O(n \log(n))$ .

Dense layer outputs predictions for all horizons.



# Autoformer

- Decomposition
- Lag-series attention



# References

1. A Multi-Horizon Quantile Recurrent Forecaster (Wen, R, et al., 2018)
2. Autoformer: Decomposition Transformers with Auto-Correlation for Long-Term Series Forecasting (Wu, H., et al., 2023)
3. DeepAR: Probabilistic Forecasting with Autoregressive Recurrent Networks (Salinas, D., et al., 2019)
4. **Deep Learning (Goodfellow, I., Bengio, Y, and Courville, A., 2016)**
5. Informer: Beyond Efficient Transformer for Long Sequence Time-Series Forecasting (Zhou, H, et al., 2021)
6. Investigating the accuracy of cross-learning time series forecasting methods (Semenoglou, A., et al., 2021)
7. Long Short-Term Memory (Hochreiter, S, and Schmidhuber, J., 1997)
8. Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation (Cho, K., et al., 2014)
9. Learning Transferable Visual Models From Natural Language Supervision (Radford, A., et al., 2021)
10. M5 accuracy competition: Results, findings, and conclusions (Makridakis, S., et al., 2021)
11. Reformer: The Efficient Transformer (Nikita, K., et al., 2020)
12. The M4 Competition: 100,000 time series and 61 forecasting methods (Makridakis, S., et al., 2020)

# Thanks



(Image by a Stable Diffusion model)