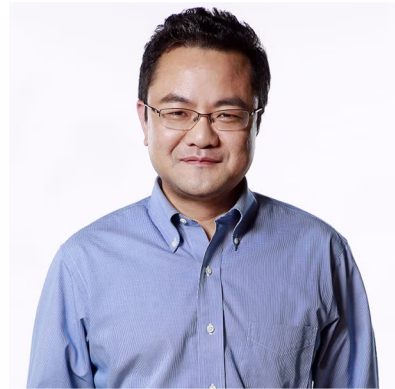


# Deep Sequence Modeling and Spatio-temporal Modeling



**Dr. Hongtu Zhu**  
**University of North Carolina at Chapel Hill**  
**URL: [www.med.unc.edu/bigs2/](http://www.med.unc.edu/bigs2/)**

# Content

**1 Motivation to Sequence Modeling**

**2 Introduction to Recurrent Neural Networks (RNNs)**

**3 Extensions of RNN**

**4 Motivation to Spatio-temporal Modeling**

**5 Deep ST Models and Applications**

# Content

## 1 Motivation to Sequence Modeling

## 2 Introduction to Recurrent Neural Networks (RNNs)

## 3 Extensions of RNN

## 4 Motivation to Spatio-temporal Modeling

## 5 Deep ST Models and Applications

# Motivation

**Recurrent Neural Networks (RNNs)** are motivated by their ability to address challenges inherent in sequential data.



Weather  
forecasting



Stock market  
trends



Autocomplete  
for texting



Genetic  
sequencing

Many real-world datasets are inherently sequential, where **the order of data points is crucial**.

- ❖ **Time series:** Stock prices, weather forecasts, and sensor data require capturing patterns over time.
- ❖ **Text:** The meaning of a sentence depends on word order ("The cat chased the dog" vs. "The dog chased the cat").
- ❖ **Speech:** Phonemes and intonation must be processed sequentially to understand spoken language.
- ❖ **Video:** Frames in a video sequence have temporal relationships that determine the flow of events.



Speech  
recognition



Video frame  
prediction



Music  
composition



...

# Sequences

A **sequence** is an **ordered list of elements**, where the order of the elements matters. They are fundamentally different from unordered data because each element is **dependent** or **influenced** by the previous elements.

## Examples:

- Letters (words)
- Words (sentences)
- Sentences (documents)
- Frames (video)
- Amino-acids (genetic code)
- fMRI/ECG signals

"Hello, how are you?" (chatbot input).

**e.g.**, A security camera capturing a person walking.

**e.g.**, "ACGTAGCTAGT" represents a biological sequence.

## Why Are Sequences Important?

Unlike independent data points, sequences contain **temporal or contextual dependencies**:

- **Future values depend on past values** (e.g., predicting tomorrow's weather).
- **Words in a sentence rely on context** (e.g., in "bank deposit" vs. "river bank").
- **Biological sequences determine genetic functions.**

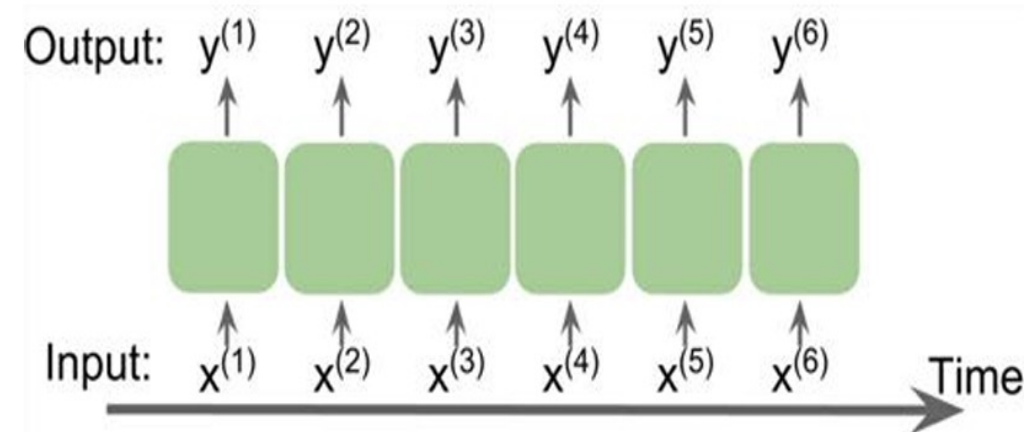
# Sequences

## Challenges in modeling sequential data

- **Infinite number of possible sequences:**
  - ❖ Sequences can vary in **length** (short vs. long sequences).
  - ❖ Sequences can have **variable patterns** (e.g., DNA sequences, language models).
  - ❖ Order **matters**, meaning different orders of the same elements can have different meanings.
- **Need for Probability Distributions Over Sequences:**
  - Since an infinite number of sequences exist, we **cannot store all possible sequences explicitly**.
  - Instead, we model a **probabilistic function** that assigns a likelihood to each possible sequence.
  - Example: Given a sequence  $S=(x_1, x_2, \dots, x_T)$ , we want to learn a **probability distribution**  $P(S)$ .

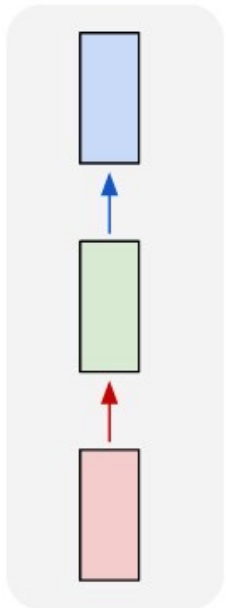
## RNNs are designed for modeling sequences

- Sequences of any length in the input, in the output, or in both
- They can remember past information
- Apply the same weights on each step



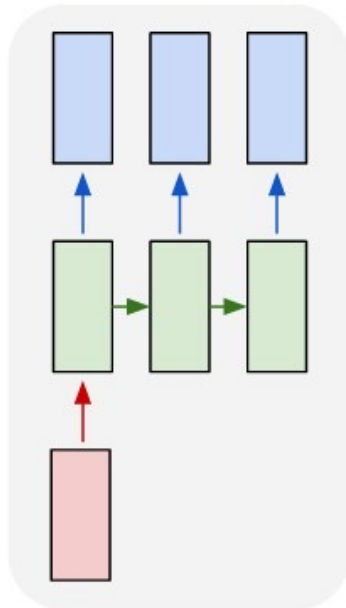
# Different Categories of Sequence Modeling

one to one



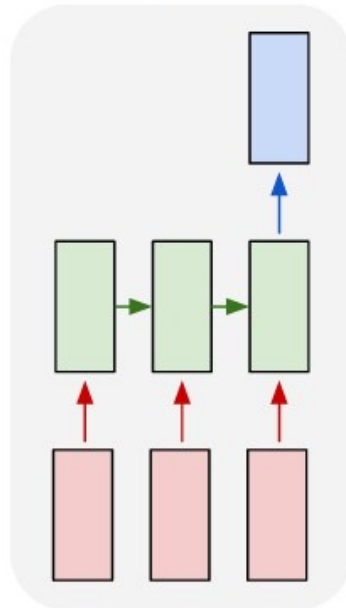
Vanilla mode without RNN  
e.g. image classification

one to many



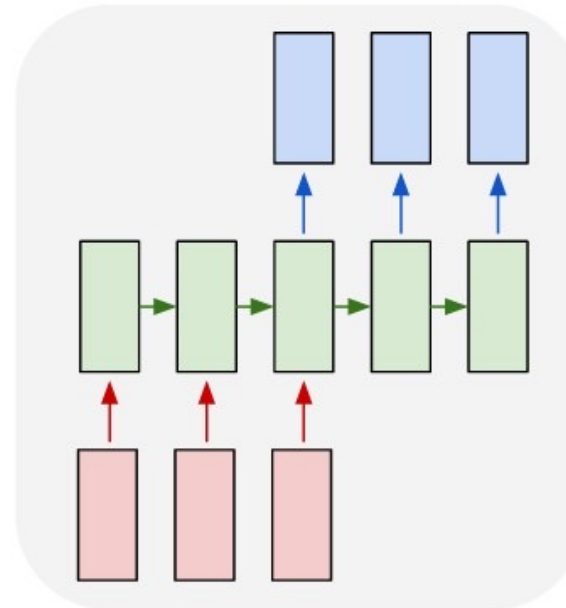
Sequence output  
e.g., image captioning

many to one



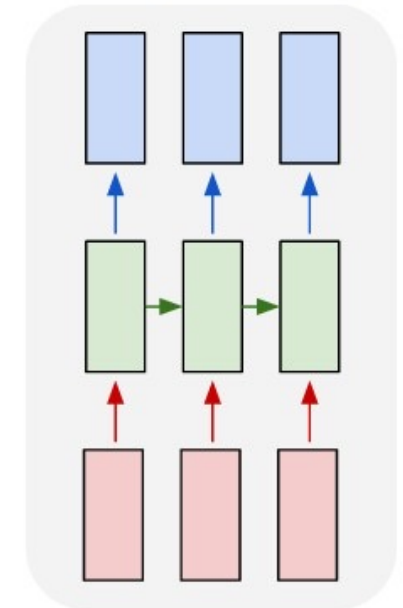
Sequence input  
e.g., sentiment analysis

many to many



Sequence input and output  
e.g., machine translation

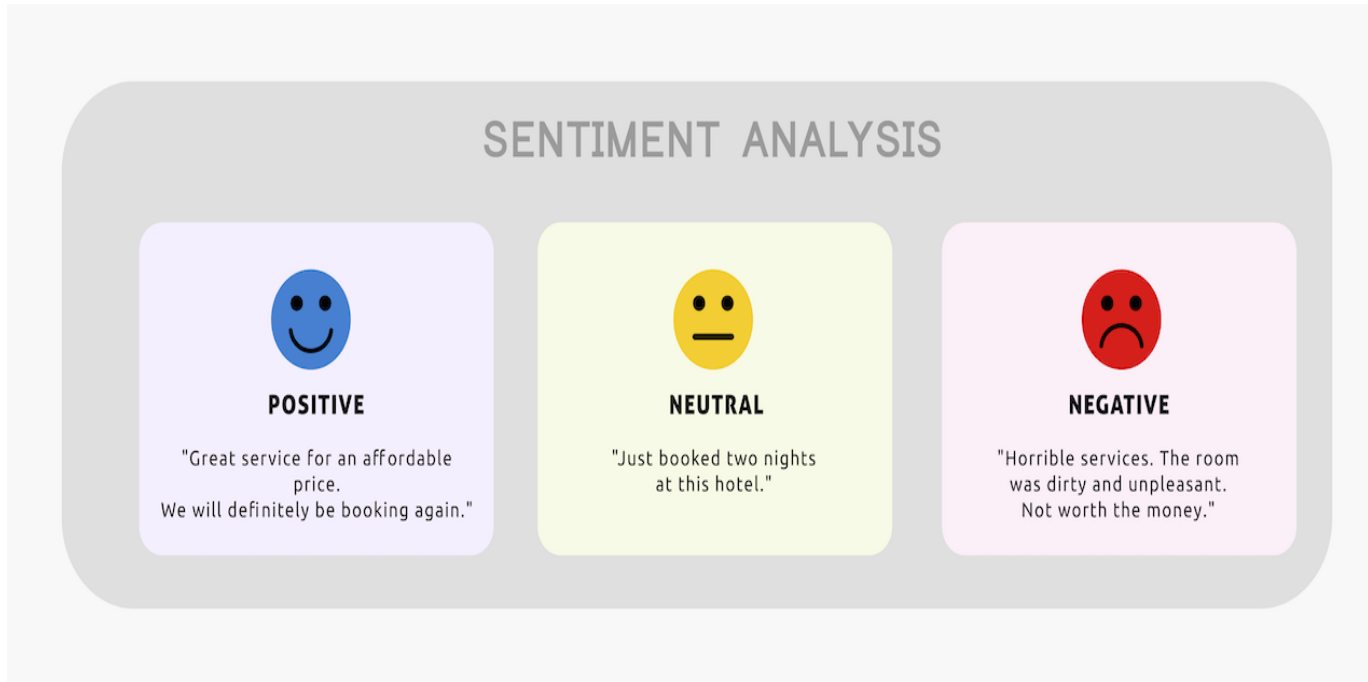
many to many



Synced sequence input and output  
e.g., video classification

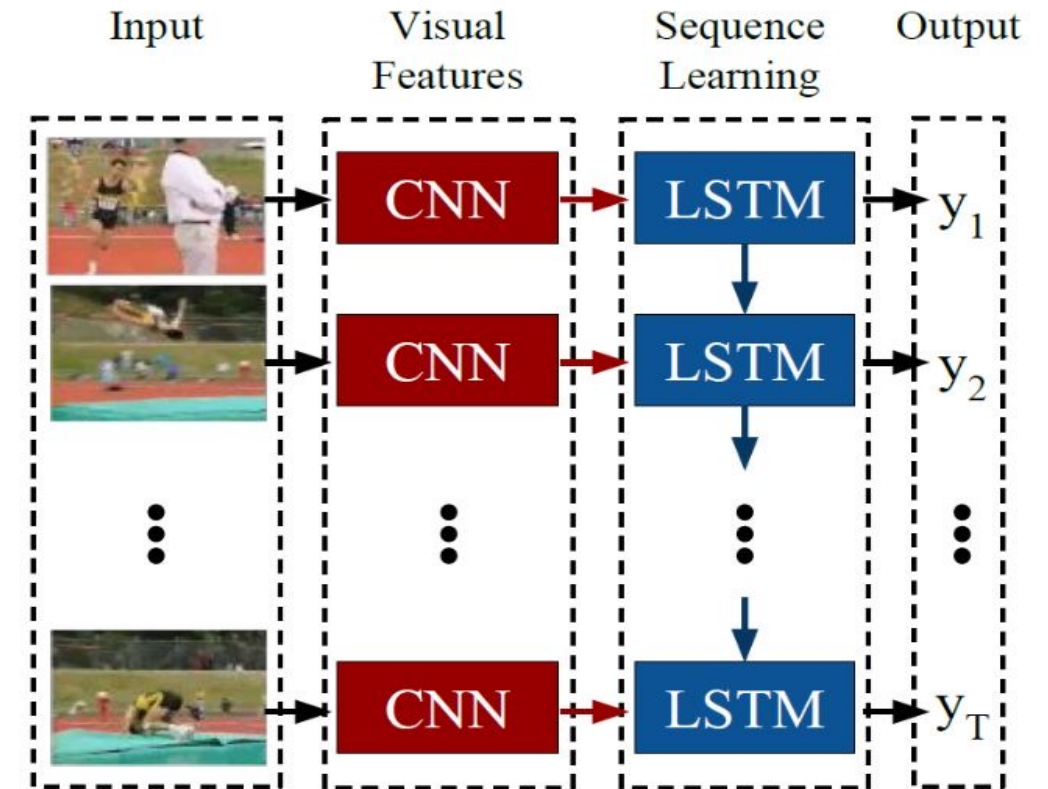
Source: <https://karpathy.github.io/2015/05/21/rnn-effectiveness/>

# Sentiment Analysis and Video Analytics



**Sentiment Analysis** is a **NLP** technique used to determine the emotional tone of a given text. It helps identify whether the sentiment of the text is **positive, negative, or neutral**.

<https://www.gosmar.eu/machinelearning/2020/08/23/recurrent-neural-networks-for-sentiment-analysis/>



**Video analytics** enable machines to **recognize actions, objects, and scenes** in videos.

<https://imerit.net/blog/using-neural-networks-for-video-classification-blog-all-pbm/>

# Machine Translation

**Machine Translation (MT)** is the task of translating a sentence  $x$  from one language (the **source language**) to a sentence  $y$  in another language (the **target language**).

**Goal:** Produce translations that are both fluent and faithful to the meaning of the source text.

**Applications:** Global communication, localization cross-lingual information retrieval, etc.

$x$ : *L'homme est né libre, et partout il est dans les fers*

**English:**

$y$ : Man is born free, but everywhere he is in chains

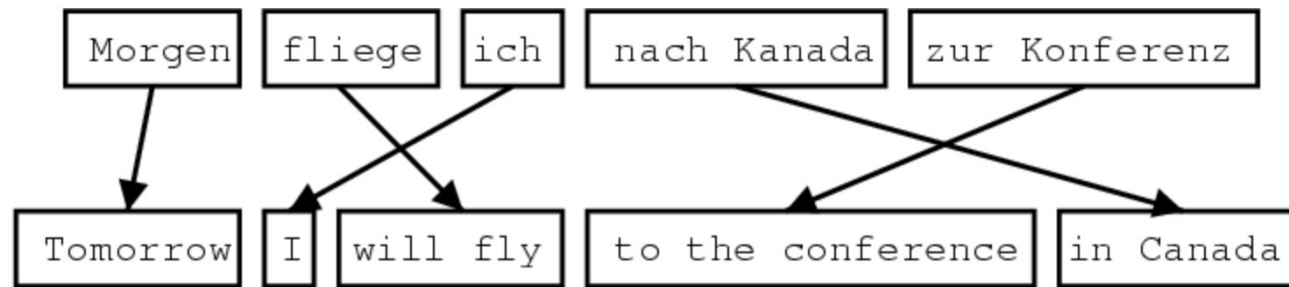
**Chinese:**

“人生而自由，但无处不在被枷锁束缚。”

**Japanese:**

「人間は自由に生まれるが、どこにいても鎖に縛られている。」

# Not trivial to model!



1519年600名西班牙人在墨西哥登陆，去征服**几百万人口**  
**的阿兹特克帝国**，初次交锋他们损兵三分之二。

In 1519, six hundred Spaniards landed in Mexico to conquer **the Aztec Empire with a population of a few million**. They lost two thirds of their soldiers in the first clash.

[translate.google.com](https://translate.google.com) (2009): 1519 600 Spaniards landed in Mexico, **millions of people to conquer the Aztec empire**, the first two-thirds of soldiers against their loss.

[translate.google.com](https://translate.google.com) (2013): 1519 600 Spaniards landed in Mexico **to conquer the Aztec empire, hundreds of millions of people**, the initial confrontation loss of soldiers two-thirds.

[translate.google.com](https://translate.google.com) (2015): 1519 600 Spaniards landed in Mexico, **millions of people to conquer the Aztec empire**, the first two-thirds of the loss of soldiers they clash.

[StanfordCS224n](#)

# NMT: the first big success story

**Neural Machine Translation (NMT):** Uses deep learning and end-to-end training to model translation and offers improved fluency and the ability to capture complex dependencies.

Neural Machine Translation went from a **fringe research attempt** in **2014** to the **leading standard method** in **2016**

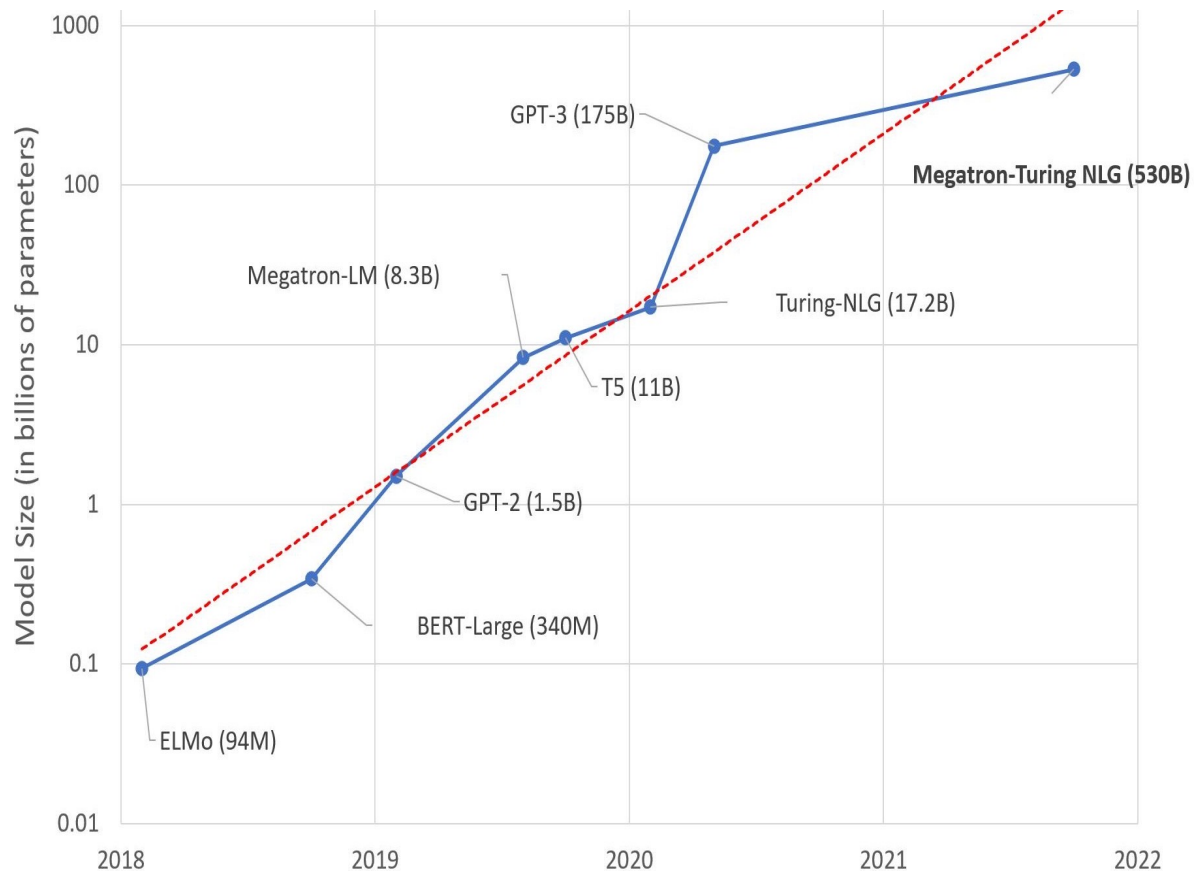
- **2014:** First seq2seq paper published [Sutskever et al. 2014]
- **2016:** Google Translate switches from SMT to NMT – and by 2018 everyone had
  - <https://www.nytimes.com/2016/12/14/magazine/the-great-ai-awakening.html>



- **This was amazing!**
  - **SMT** systems, built by **hundreds** of engineers over many **years**, were outperformed by NMT systems trained by **small groups** of engineers in a few **months**

StanfordCS224n

# Deepseek v.s. OpenAI



## Modern NLP Systems

<https://ai.plainenglish.io/deepseek-r1-vs-chatgpt-01-my-experience-ddbe09e80aa9>

<https://huggingface.co/blog/large-language-models>

# Challenges

- Standard NN models (MLPs, CNNs) are not able to handle sequences of data
  - ❖ They accept a **fixed-sized vector** as input and produce a **fixed-sized vector** as output.
  - ❖ The **weights are updated independently**, meaning there is **no memory** of past computations.
  - ❖ The models **do not have recurrence**, so they cannot learn patterns across time steps.
- Many real-world problems require capturing **context over time**:
  - ❖ **Speech Recognition** – Words depend on previous words.
  - ❖ **Time-series Prediction** – Future values depend on past observations.
  - ❖ **DNA Sequencing** – Genetic patterns unfold over long sequences.
  - ❖ **Natural Language Processing (NLP)** – Meaning depends on word order.
- **Example: Simple Context-Dependent Problem:** Output YES if the number of 1s in the sequence is even; otherwise, output NO.
  - Input: 1000010101 → **YES**; Input: 100011 → **NO**

# Challenges

**High Dimensionality and Complexity** - Sequential data often involves high-dimensional inputs with complex interdependencies:

- ❖ **Text:** Words and phrases have semantic and syntactic relationships across sentences.
- ❖ **Time Series:** Multivariate time series data (e.g., temperature, humidity, and pressure) exhibit interdependencies between variables over time.
- ❖ **Biological Data:** DNA sequences and protein structures involve intricate, sequential patterns.

**Solution:** RNNs address this by learning hierarchical representations through their recurrent structure, encoding both local and global patterns.

**Noise and Missing Data** - Sequential data often contains noise or missing values:

- **Noise:** Sensor readings and time series data may have irregularities or anomalies.
- **Missing values:** Gaps in sequences arise from interruptions in data collection.

**Solution:** RNNs aggregate information over time, making them robust to noise and capable of interpolating missing values using contextual information.

# Challenges

## Temporal Dependencies

- ❖ **Short-term dependencies:** In text, the current word depends on immediately preceding words (e.g., ``I want to eat a...’’).
- ❖ **Long-term dependencies:** Distant elements in the sequence can influence the current state (e.g., in a paragraph, the topic sentence affects subsequent sentences).

**Solution:** RNNs maintain memory through hidden states, enabling them to model temporal dependencies. Variants like Long Short-Term Memory (LSTM) and Gated Recurrent Units (GRU) address challenges such as vanishing gradients, allowing effective modeling of long-term dependencies.

**Variable-Length Inputs and Outputs** - Many real-world tasks involve sequences of varying lengths, which traditional models struggle to handle. RNNs process inputs dynamically, making them ideal for tasks with variable-length data.

### Examples:

- ❖ **Natural Language Processing (NLP):** Sentences have varying word counts, and RNNs can process each word without requiring fixed input dimensions.
- ❖ **Speech Recognition:** Audio recordings vary in duration depending on the speaker or content.
- ❖ **Time Series:** Data collected over irregular time intervals often results in sequences of differing lengths.

# Content

1 Motivation to Sequence Modeling

**2 Recurrent Neural Networks (RNNs)**

3 Extensions of RNN

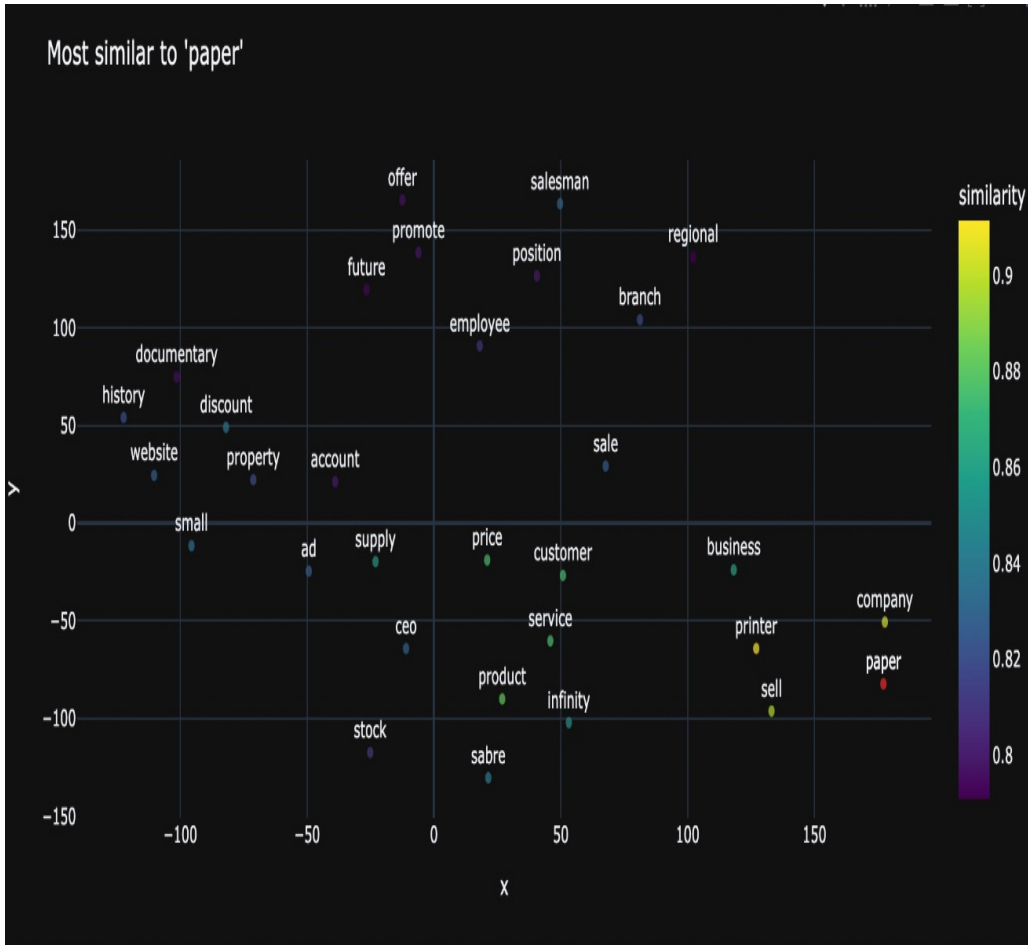
4 Motivation to Spatio-temporal Modeling

5 Deep ST Models and Applications

# Text Processing for RNN

Step	Task	Description
1	Preprocessing Text	Clean text, tokenize, remove stopwords, and normalize case using NLTK.
2	Build Vocabulary	Assign a unique index to each word using a frequency-based vocabulary.
3	Convert Text to Sequences	Map tokenized words to their corresponding integer indices.
4	Padding Sequences	Standardize input sequence lengths by adding padding tokens where necessary.
5	Dataloader Preparation	Create PyTorch dataset and dataloader for mini-batch training.
6	Load Pretrained Word Embeddings	Use GloVe embeddings (100D) for better semantic representation.
7	Define RNN Model	Construct an RNN with an embedding layer, hidden layers, and output layer.
8	Loss and Optimization	Use Binary Cross-Entropy Loss ('BCEWithLogitsLoss') and Adam optimizer.
9	Train Model	Train the RNN model using mini-batches from the dataloader.
10	Make Predictions	Preprocess new text, convert it to sequences, and run inference using the trained model.

# Word Embeddings



**Word embeddings** are a fundamental technique in NLP. They convert words into dense, continuous vector representations. Word embeddings place similar words closer in vector space. Unlike traditional one-hot encoding, embeddings preserve:

- ❖ **Semantic relationships** between words.
- ❖ **Contextual meaning** of words in sentences.
- ❖ **Word similarity and analogies.**

## Types of Word Embeddings

### ➤ **Frequency-Based Methods**

- TF-IDF (Term Frequency-Inverse Document Frequency)
- LSA (Latent Semantic Analysis)

### ➤ **Prediction-Based Methods (Neural Networks)**

- ✓ Word2Vec (CBOW & Skip-gram)
- ✓ GloVe (Global Vectors for Word Representation)
- ✓ FastText (Subword Embeddings)
- ✓ Transformer-Based (BERT, GPT)

# Word Embeddings Dimensions

```
from transformers import BertTokenizer, BertModel
import torch
```

```
# Load BERT model
```

```
tokenizer = BertTokenizer.from_pretrained("bert-base-uncased")
model = BertModel.from_pretrained("bert-base-uncased")
```

```
# Tokenize and get embedding
```

```
text = "Hello world"
tokens = tokenizer(text, return_tensors="pt")
with torch.no_grad():
    output = model(**tokens)
```

```
print("BERT Embedding Dimension:",
      output.last_hidden_state.shape[-1])
```

Word Embeddings	Dimension	Key Features
Word2Vec	50-300	Trained on large corpora like Google News
GloVe	50-300	Uses word co-occurrence statistics
FastText	50-300	Handles subword information
ELMo	1024	Contextual embeddings from bidirectional LSTMs
BERT (base)	768	Transformer-based, context-aware
BERT (large)	1024	More parameters than BERT base
GPT-2 (small)	768	Transformer-based generative model
GPT-2 (medium)	1024	More layers and parameters
GPT-3	12288	High-dimensional transformer model

◆ **For small models or mobile applications** →  
Use **50-300 dimensions** (Word2Vec, GloVe).

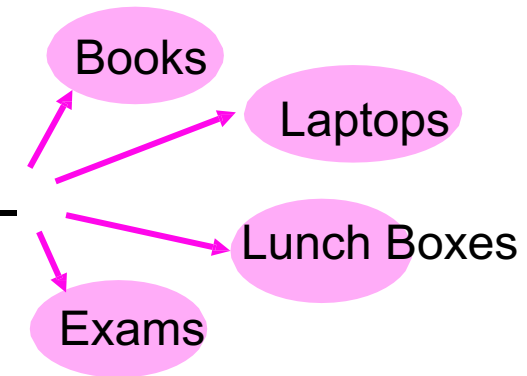
◆ **For NLP applications with context-awareness** →  
Use **512-1024 dimensions** (BERT, ELMo).

◆ **For large-scale generative AI** →  
Use **1024+ dimensions** (GPT-3, Transformers).

# Formulation: Language Modeling (LM)

A **language model (LM)** is a statistical or machine learning model that **predicts the next word in a sequence** or assigns **probabilities** to sequences of words.

*the students opened their* \_\_\_\_\_



- ✓ Predicts the likelihood of a sequence of words
- ✓ Generates human-like text (e.g., GPT models)
- ✓ Understands context and meaning
- ✓ Enables AI systems to process and generate natural language

More formally: given a sequence of words  $x^{(1)}, x^{(2)}, \dots, x^{(t)}$ , compute the probability distribution of the next word  $x^{(t+1)}$ :  $P(x^{(t+1)} | x^{(t)}, \dots, x^{(1)})$  where  $x^{(t+1)}$  can be any word in the vocabulary  $V = \{w_1, \dots, w_{|V|}\}$


## Example:

- **Input:** "I am going to the"
- **Model prediction:** "store" (80%), "beach" (15%), "moon" (5%)
- The model assigns probabilities and selects the most likely next word.

# Language Modeling

- You can also think of a Language Model as a system that assigns a probability to a piece of text
- For example, if we have some text  $\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(T)}$  then the probability of this text (according to the Language Model) is:

$$P(\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(T)}) = P(\mathbf{x}^{(1)}) \times P(\mathbf{x}^{(2)} | \mathbf{x}^{(1)}) \times \dots \times P(\mathbf{x}^{(T)} | \mathbf{x}^{(T-1)}, \dots, \mathbf{x}^{(1)})$$

$$= \prod_{t=1}^T P(\mathbf{x}^{(t)} | \mathbf{x}^{(t-1)}, \dots, \mathbf{x}^{(1)})$$


This is what our LM provides

# History of Deep RNNs

## The Rise of Deep RNNs (2010s - Present)

### ◆ RNNs in NLP and AI

- **2013** – Google used LSTM for **speech recognition**.
- **2014** – **Seq2Seq Models** (Sutskever et al.) used LSTMs for **machine translation**.
- **2015** – Google Translate adopted LSTMs for **neural machine translation (NMT)**.

### ◆ Attention and Transformers Change the Game

- **2015** – **Bahdanau et al.** introduced **Attention Mechanisms**, improving Seq2Seq models.
- **2017** – **Vaswani et al.** introduced **Transformers**, replacing RNNs with a more parallelizable model.

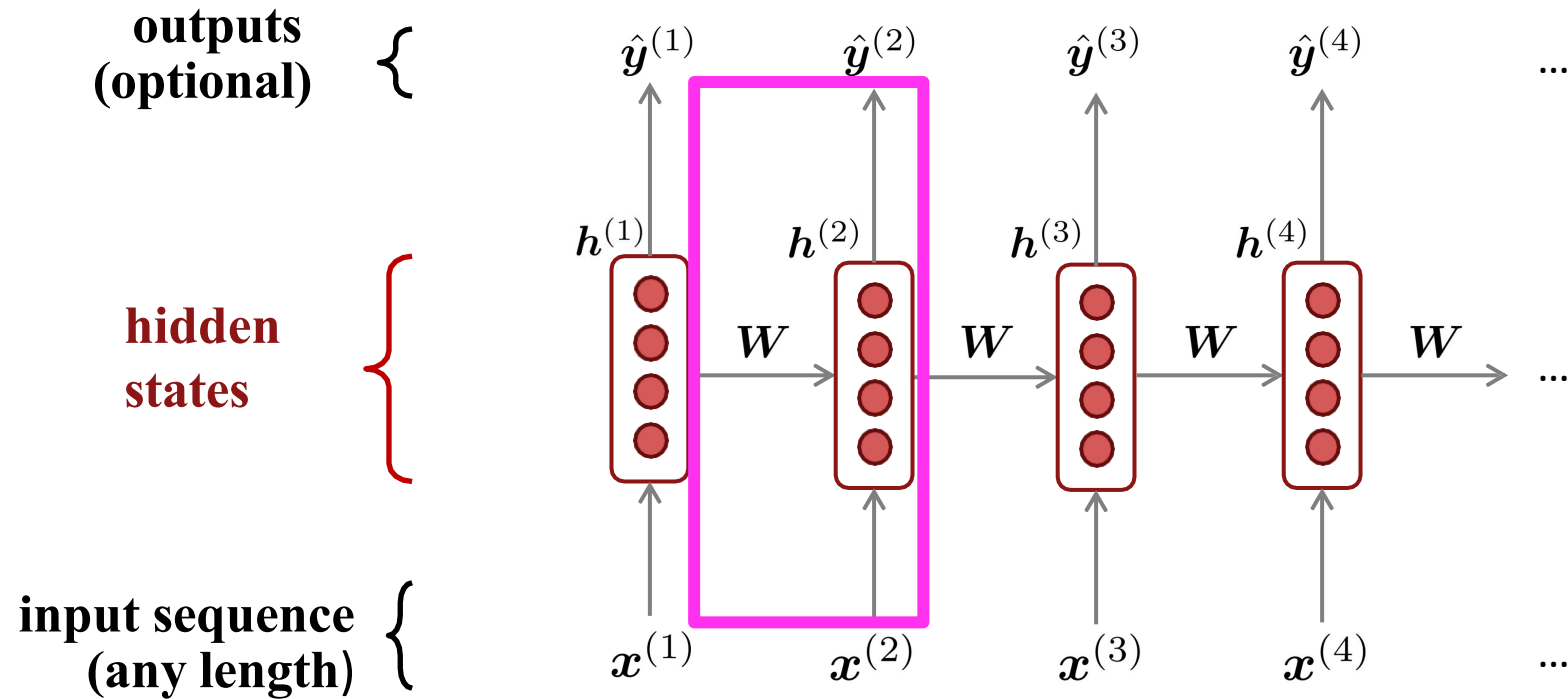
### 🔍 Key Concept:

Transformers like **BERT (2018)**, **GPT-3 (2020)**, and **ChatGPT (2022)** outperformed RNNs, leading to their decline in NLP.

# Recurrent Neural Networks (RNN)

A family of neural architectures

**Core idea:** Apply the same weights  $W$  repeatedly



```
nn.RNN(input_size, hidden_size, num_layers, batch_first=True)
```

StanfordCS224n

# A Simple RNN Language Model

## output distribution

$$\hat{y}^{(t)} = \text{softmax} \left( U h^{(t)} + b_2 \right) \in \mathbb{R}^{|V|}$$

## hidden states

$$h^{(t)} = \sigma \left( W_h h^{(t-1)} + W_e e^{(t)} + b_1 \right)$$

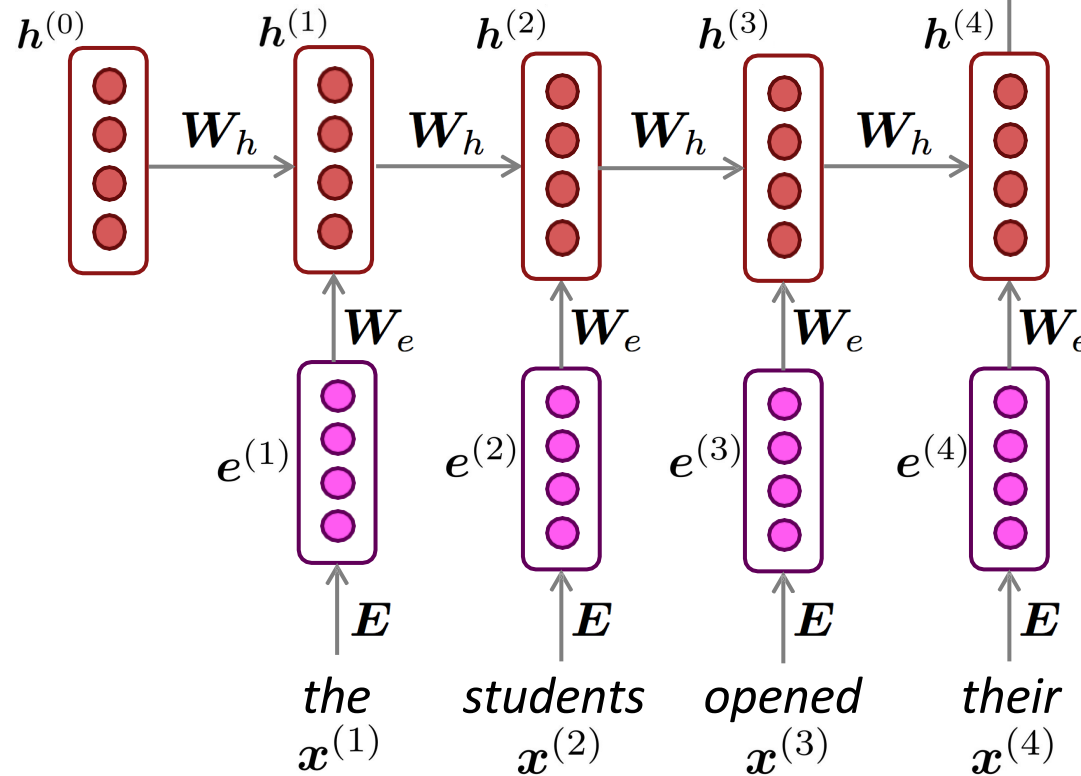
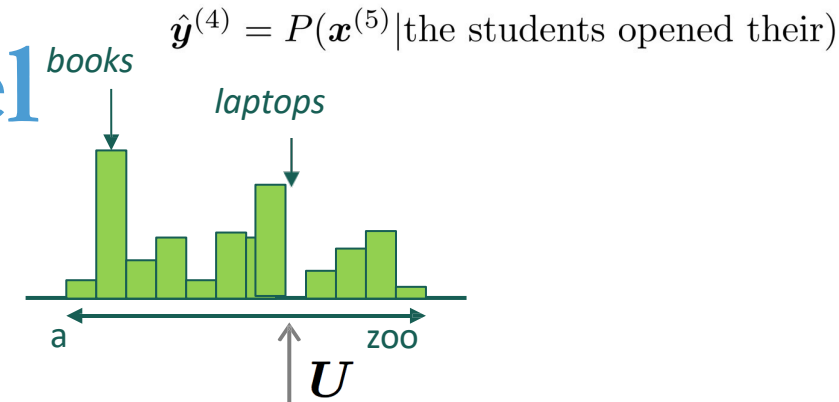
$h^{(0)}$  is the initial hidden state

## word embeddings

$$e^{(t)} = E x^{(t)}$$

## words / one-hot vectors

$$x^{(t)} \in \mathbb{R}^{|V|}$$



Note: this input sequence could be much longer now!

# RNN Language Models

## RNN **Advantages**:

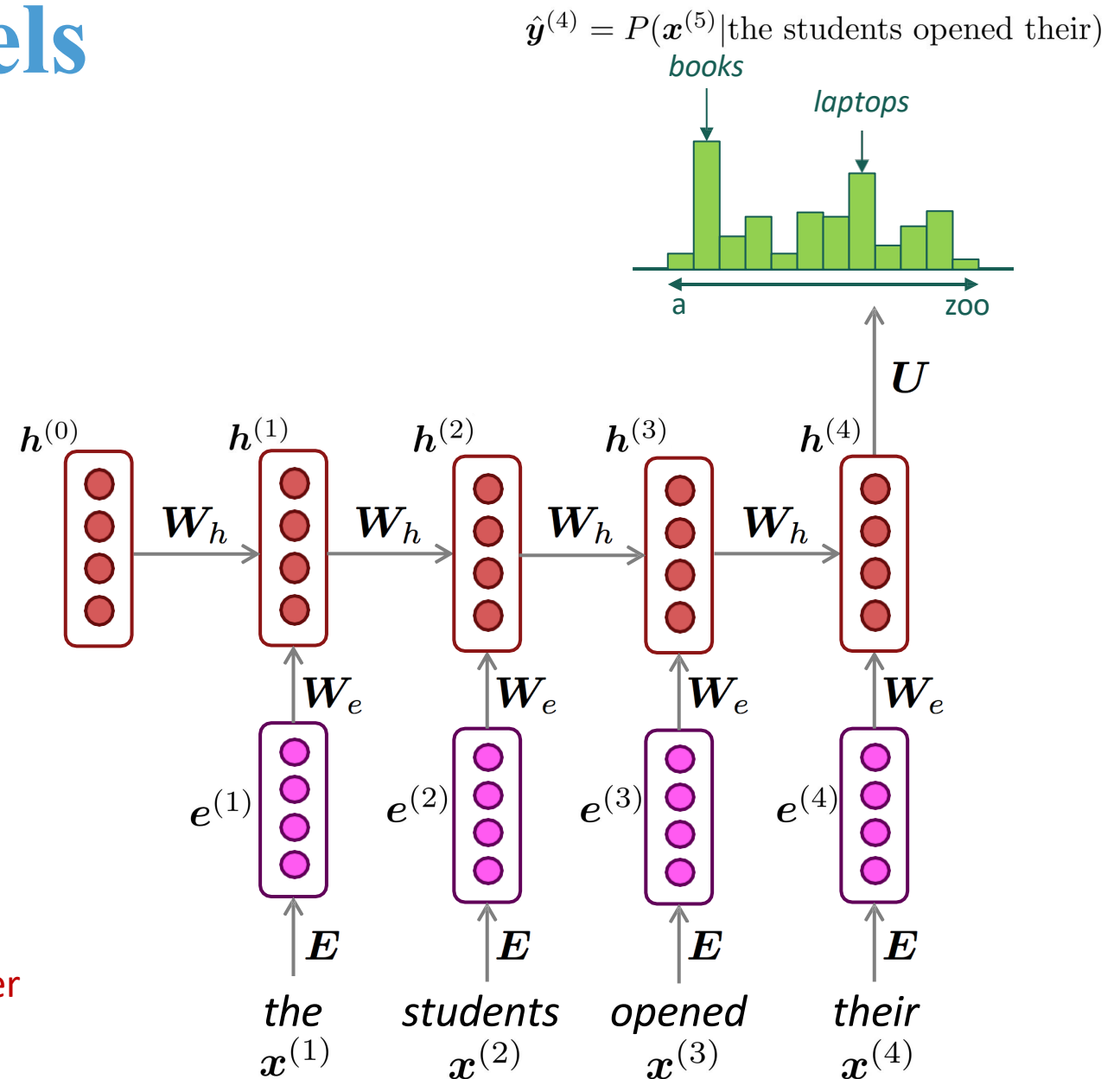
- Can process **any length** input
- Computation for step  $t$  can (in theory) use information from **many steps back**
- **Model size doesn't increase** for longer input context
- Same weights applied on every timestep, so there is **symmetry** in how inputs are processed.

## RNN **Disadvantages**:

- Recurrent computation is **slow**
- In practice, difficult to access information from **many steps back**

More on these later

StanfordCS224n



# Training an RNN Language Model

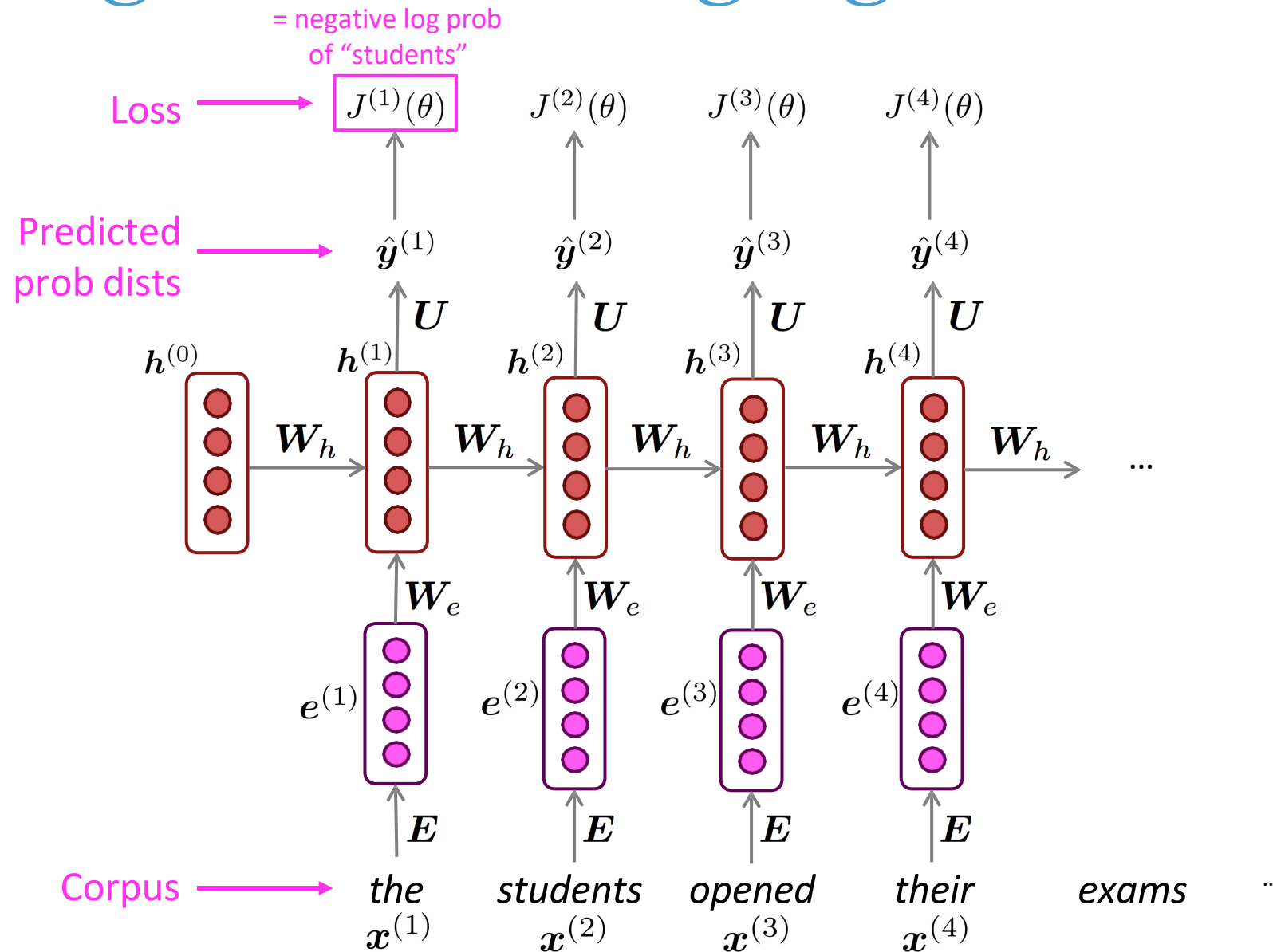
- ❖ Prepare the dataset (tokenize text into words) and convert words to numerical tensors (word embeddings).
- ❖ Build an RNN-LM and compute output distribution  $\hat{\mathbf{y}}^{(t)}$  for every step  $t$ .
- ❖ Loss function on step  $t$  is cross-entropy loss between predicted probability distribution  $\hat{\mathbf{y}}^{(t)}$  and the true next word  $\mathbf{y}^{(t)}$  (one-hot for  $\mathbf{x}^{(t+1)}$ ):

$$J^{(t)}(\theta) = CE(\mathbf{y}^{(t)}, \hat{\mathbf{y}}^{(t)}) = - \sum_{w \in V} \mathbf{y}_w^{(t)} \log \hat{\mathbf{y}}_w^{(t)} = - \log \hat{\mathbf{y}}_{\mathbf{x}_{t+1}}^{(t)}$$

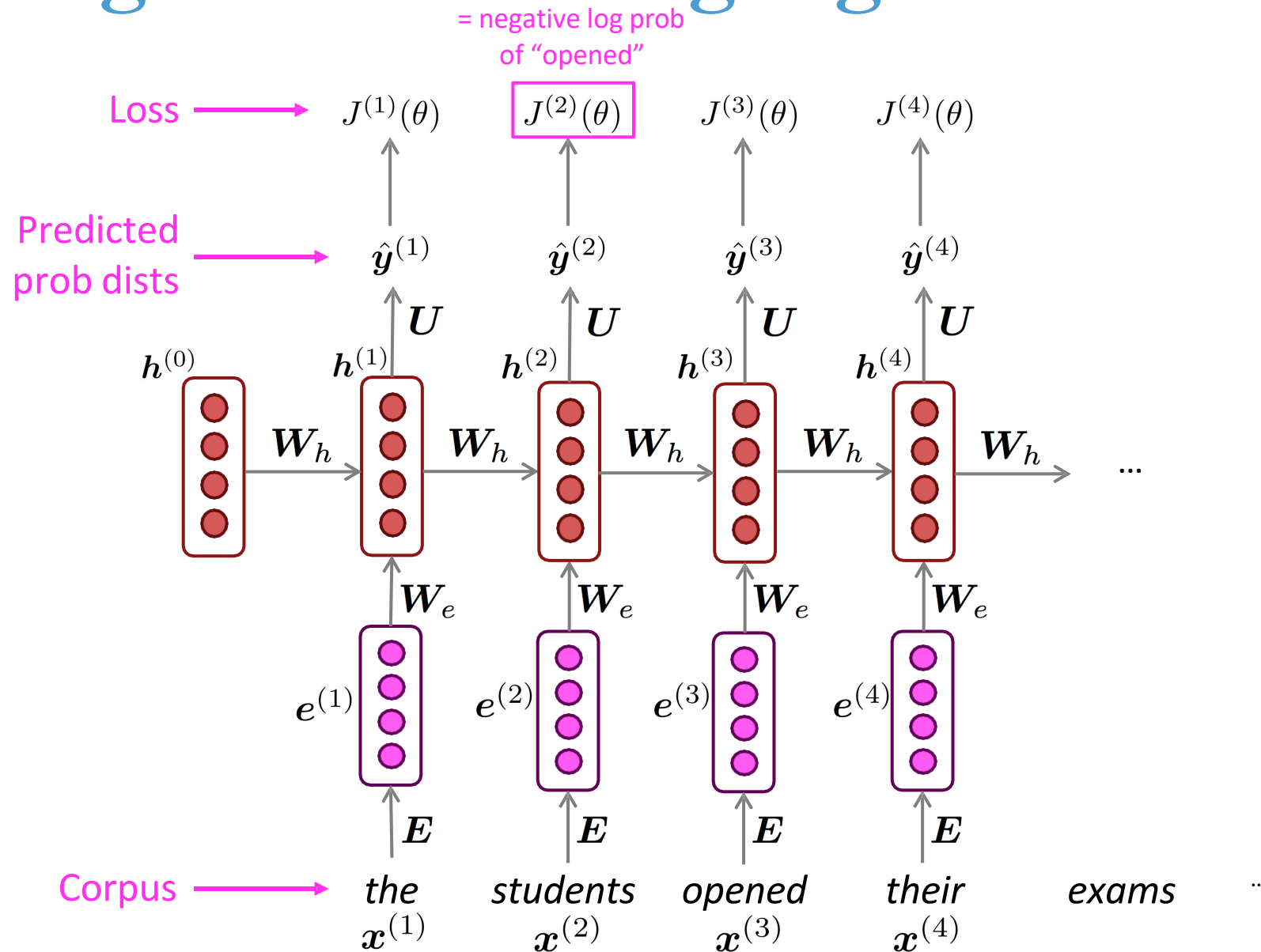
- ❖ Average this to get the overall loss for entire training set:

$$J(\theta) = \frac{1}{T} \sum_{t=1}^T J^{(t)}(\theta) = \frac{1}{T} \sum_{t=1}^T - \log \hat{\mathbf{y}}_{\mathbf{x}_{t+1}}^{(t)}$$

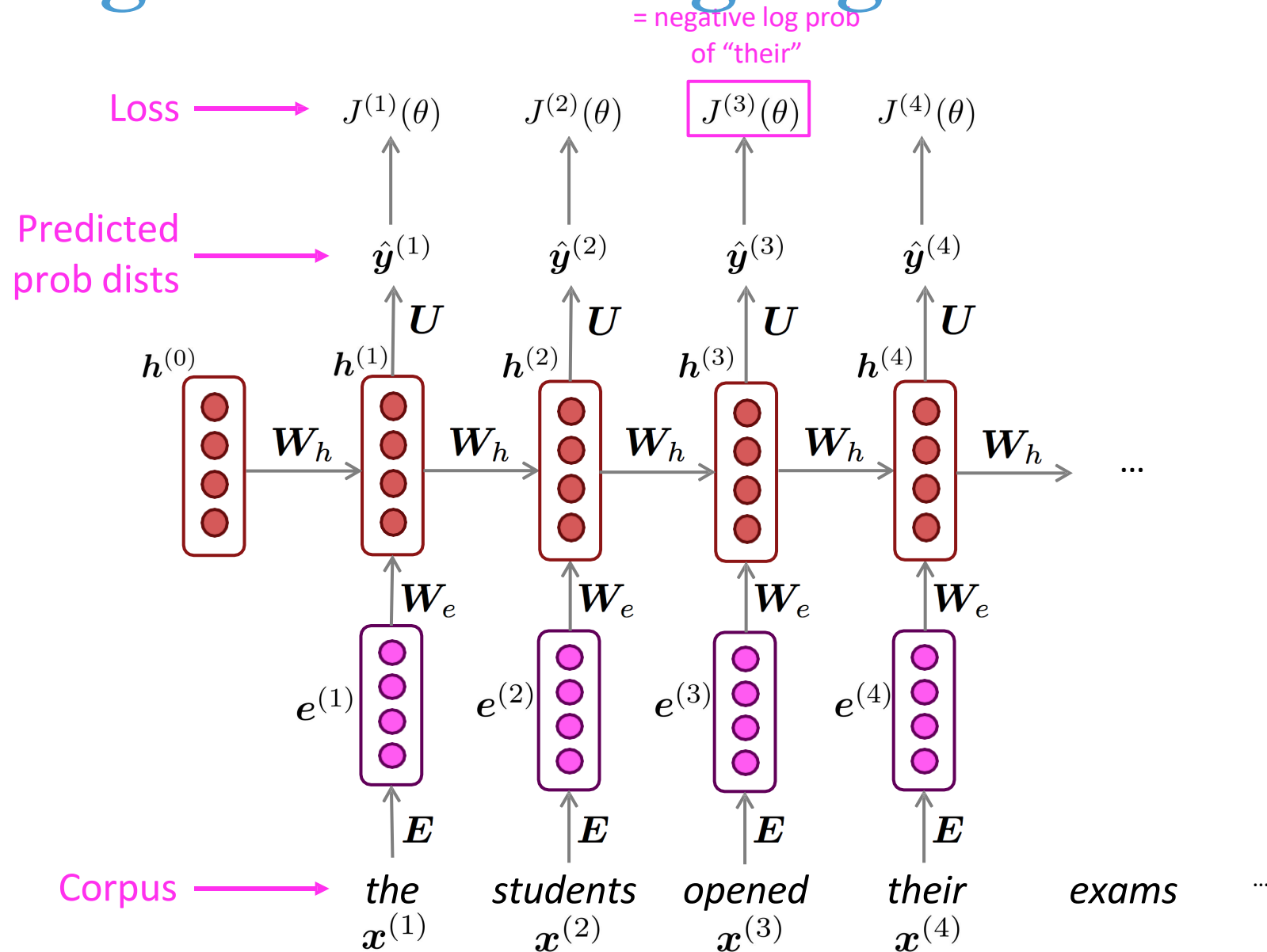
# Training an RNN Language Model



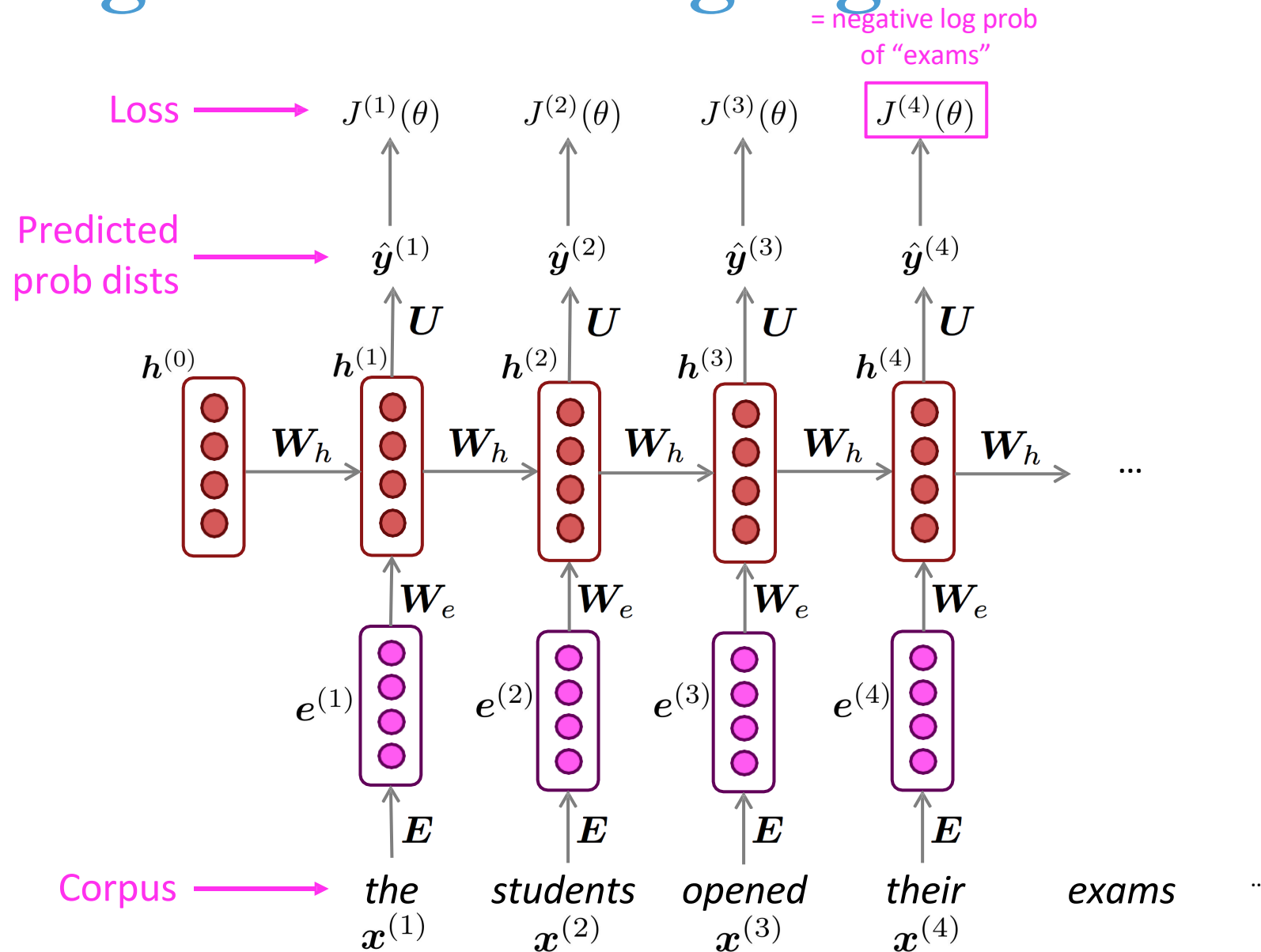
# Training an RNN Language Model



# Training an RNN Language Model

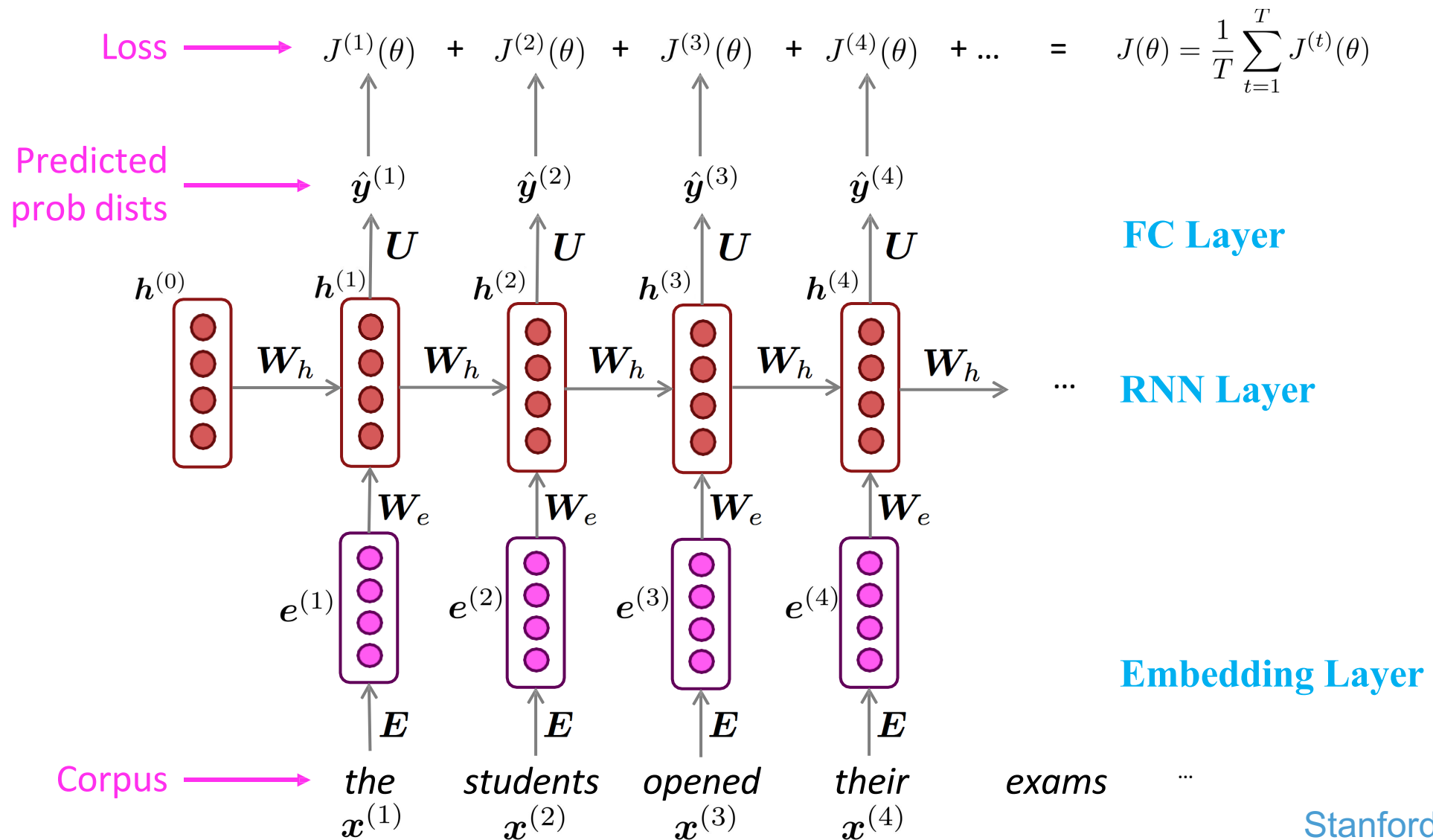


# Training an RNN Language Model



# Training an RNN Language Model

“Teacher forcing”



StanfordCS224n

# Training Challenges

- **Memory Constraints:** Storing all word embeddings, gradients, and activations requires enormous memory.
- **Computational Cost:** Performing **backpropagation over the entire dataset** in a single step is impractical.
- **Batching is Required:** Instead of processing all data at once, models use **mini-batches** to update weights efficiently.

$$J(\theta) = \frac{1}{T} \sum_{t=1}^T J^{(t)}(\theta)$$

## Challenges

## Solution

Too much memory usage

✓ Mini-batch training

Divide the **entire dataset** into **mini-batches** (e.g., **batch size = 32**).

Long sequences overflow memory

✓ Truncated BPTT (TBPTT)

**Truncate** the sequence into smaller **sub-sequences** (e.g., **20 time steps** at a time).

Exploding gradients

✓ Gradient clipping

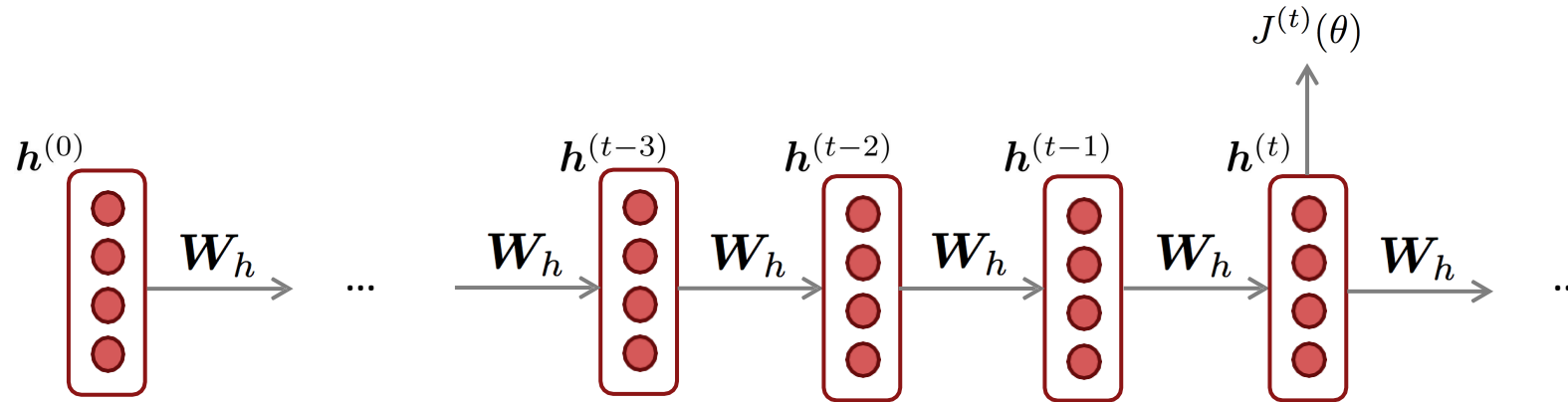
Clip gradients to a maximum norm (e.g., **5**).

Slow training

✓ Efficient batching and parallelization

$$g = g \times \frac{C}{\max(\|g\|, C)}$$

# Backpropagation for RNNs



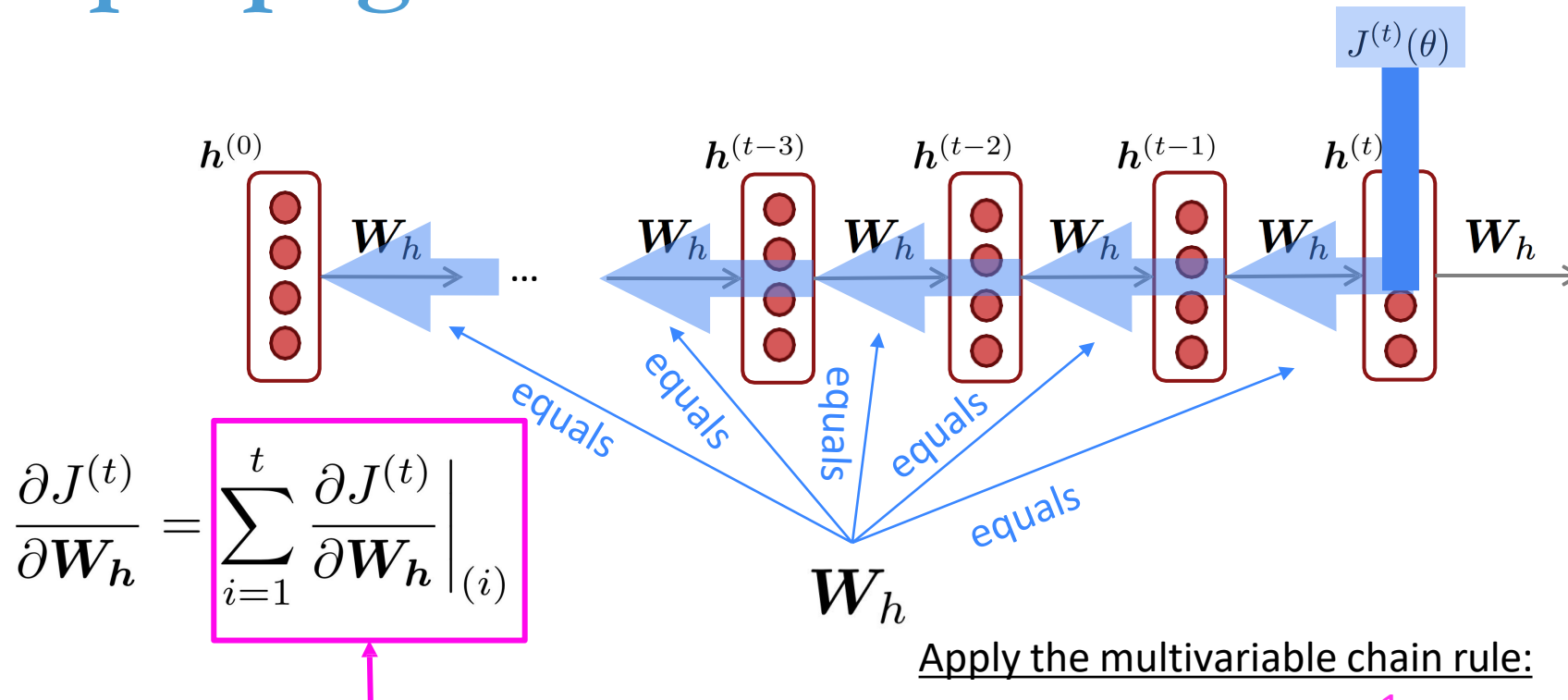
**Question:** What's the derivative of  $J^{(t)}(\theta)$  w.r.t. the **repeated** weight matrix  $W_h$  ?

**Answer:** 
$$\frac{\partial J^{(t)}}{\partial W_h} = \sum_{i=1}^t \frac{\partial J^{(t)}}{\partial W_h} \Big|_{(i)}$$

“The gradient w.r.t. a repeated weight is the sum of the gradient w.r.t. each time it appears”

Why?

# Backpropagation for RNNs



In practice, often “truncated” after ~20 timesteps for training efficiency reasons

**Question:** How do we calculate this?

**Answer:** Backpropagate over timesteps  $i = t, \dots, 0$ , summing gradients as you go. This algorithm is called “**backpropagation through time**” [Werbos, P.G., 1988, *Neural Networks 1*, and others]

$$\begin{aligned} \frac{\partial J^{(t)}}{\partial W_h} &= \sum_{i=1}^t \left. \frac{\partial J^{(t)}}{\partial W_h} \right|_{(i)} \boxed{\frac{\partial W_h}{\partial W_h}} \\ &= \sum_{i=1}^t \left. \frac{\partial J^{(t)}}{\partial W_h} \right|_{(i)} \end{aligned}$$

# RNNs greatly improved perplexity

*n*-gram model →

Increasingly complex RNNs ↓

Model	Perplexity
Interpolated Kneser-Ney 5-gram (Chelba et al., 2013)	67.6
RNN-1024 + MaxEnt 9-gram (Chelba et al., 2013)	51.3
RNN-2048 + BlackOut sampling (Ji et al., 2015)	68.3
Sparse Non-negative Matrix factorization (Shazeer et al., 2015)	52.9
LSTM-2048 (Jozefowicz et al., 2016)	43.7
2-layer LSTM-8192 (Jozefowicz et al., 2016)	30
Ours small (LSTM-2048)	43.9
Ours large (2-layer LSTM-2048)	39.8

Perplexity improves (lower is better) ↓

Source: <https://research.fb.com/building-an-efficient-neural-language-model-over-a-billion-words/>

StanfordCS224n

# Content

1 Motivation to Sequence Modeling

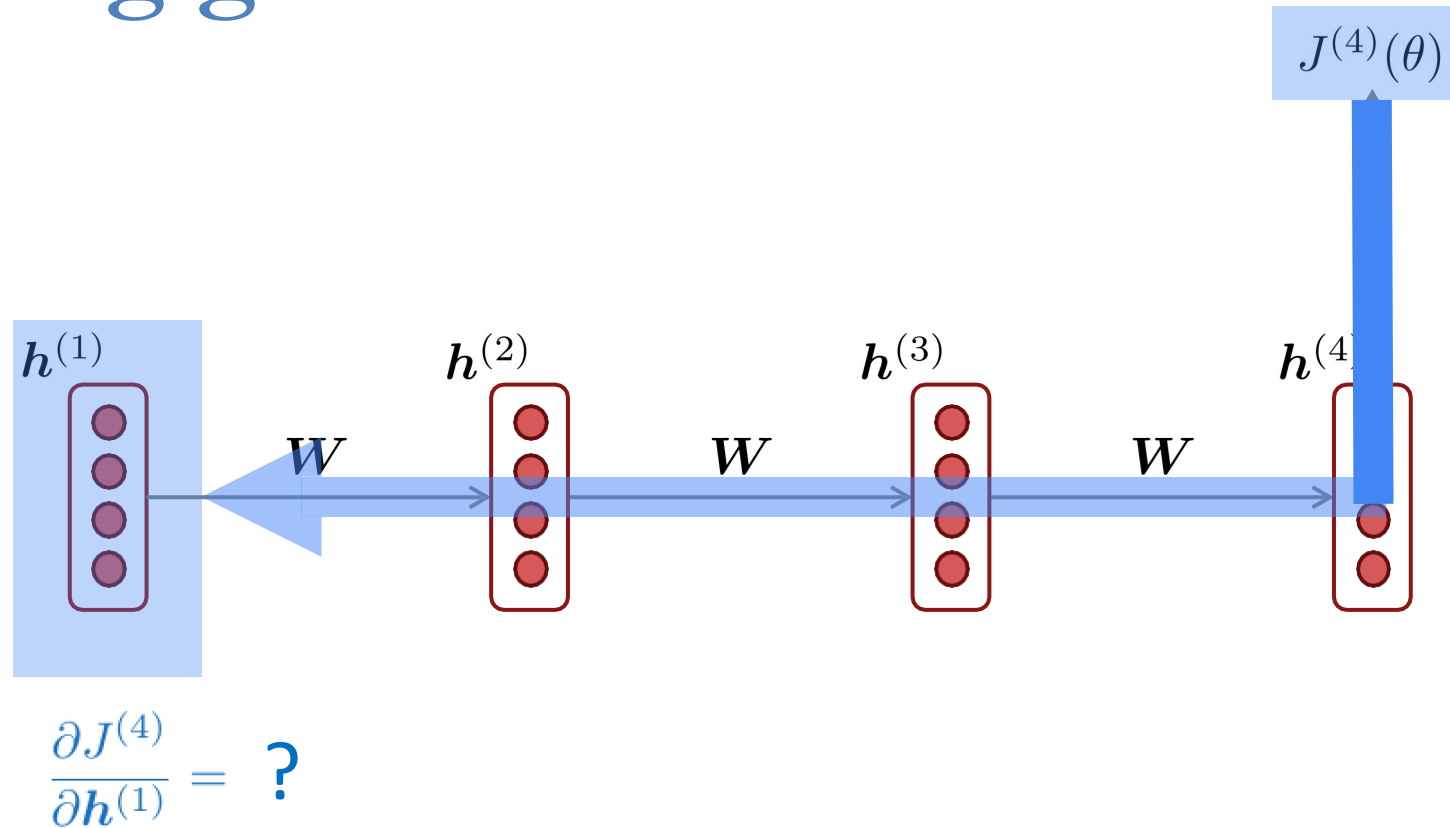
2 Recurrent Neural Networks (RNNs)

**3 Extensions of RNN**

4 Motivation to Spatio-temporal Modeling

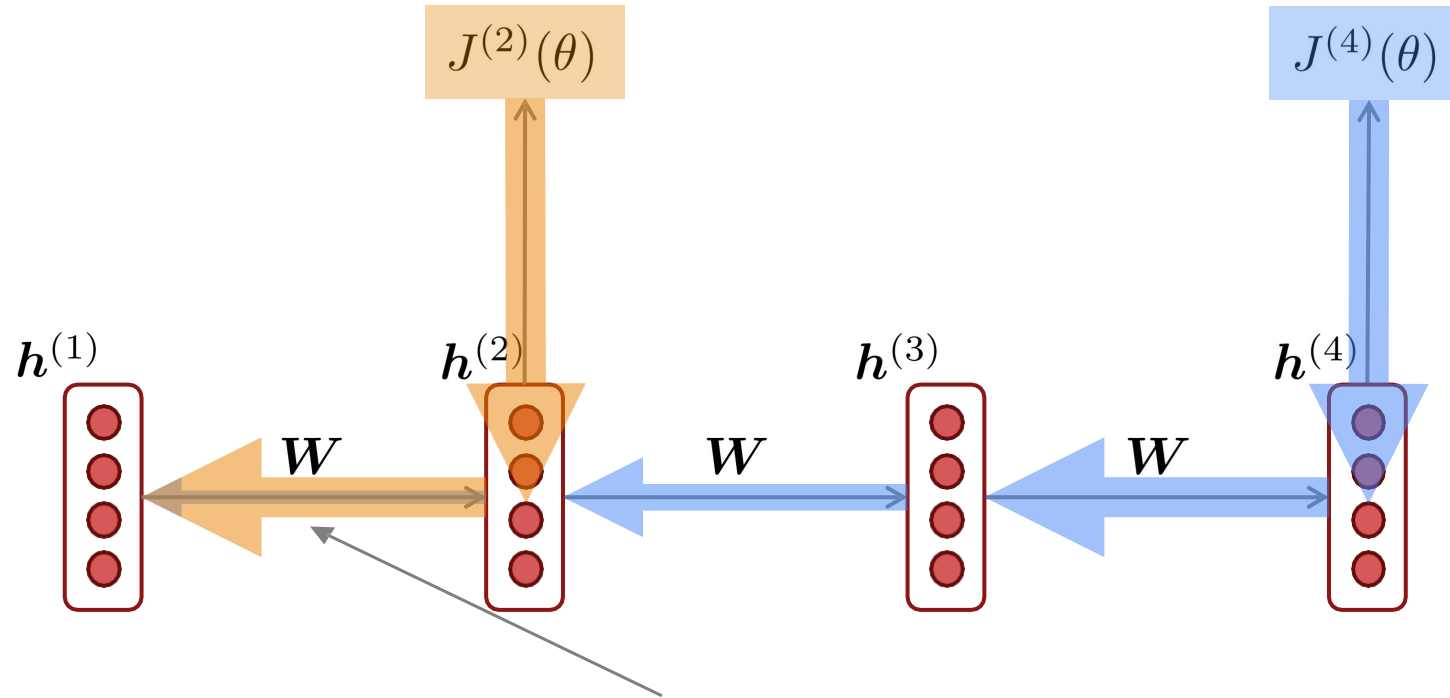
5 Deep ST Models and Applications

# Vanishing gradient intuition



- ▶ This decay makes it difficult for RNNs to learn long-term dependencies.
- ▶ Empirical evidence shows rapid gradient norm decay.

# Why is vanishing gradient a problem?



- **Short-Range vs. Long-Range:** Imagine a long chain of dominos where each domino represents a layer or time step. If the force transferred from one domino to the next diminishes (say by a constant factor each time), then after a long chain, the force reaching the first domino becomes nearly zero. This means the first few dominos (or layers) barely "feel" the impact of the initial force (or error), and thus they do not adjust effectively based on long-term dependencies.
- **Resulting Behavior:** The model ends up "paying attention" only to the parts of the sequence that are immediately relevant (the nearby gradient signals) while ignoring distant context. This leads to challenges such as:
  - ❖ Inability to learn relationships or dependencies that span many time steps.
  - ❖ Poor performance on tasks that require integrating information over long sequences.

# Long Short-Term Memory RNNs (LSTMs)

- At each time step  $t$ , the LSTM maintains a **hidden state**  $h(t)$  and a **cell state**  $c(t)$ , both vectors of length  $n$ .
- The cell state stores **long-term information**, while the hidden state represents **the immediate output**.
- Three gates—the **forget gate**, **input gate**, and **output gate**—dynamically control the **erasing, reading, and writing** of information in the cell state.
- Each gate is computed as a vector with values between 0 and 1, where **values indicate the proportion of information to keep or update**.
- This gating mechanism allows the LSTM to **selectively maintain important information over long sequences**, effectively addressing the vanishing gradient problem encountered in traditional RNNs.

This dynamic gating system is a key innovation that makes LSTMs powerful for tasks requiring long-term memory, such as language modeling, speech recognition, and time-series prediction.

# Long Short-Term Memory (LSTM)

A sequence of inputs  $\mathbf{x}^{(t)}$ . Compute a sequence of hidden states  $\mathbf{h}^{(t)}$  and cell states  $\mathbf{c}^{(t)}$ . On timestep  $t$ :

**Forget gate:** controls what is kept vs forgotten, from previous cell state

**Input gate:** controls what parts of the new cell content are written to cell

**Output gate:** controls what parts of cell are output to hidden state

**New cell content:** this is the new content to be written to the cell

**Cell state:** erase (“forget”) some content from last cell state, and write (“input”) some new cell content

**Hidden state:** read (“output”) some content from the cell

**Sigmoid function:** all gate values are between 0 and 1

$$\mathbf{f}^{(t)} = \sigma \left( \mathbf{W}_f \mathbf{h}^{(t-1)} + \mathbf{U}_f \mathbf{x}^{(t)} + \mathbf{b}_f \right)$$

$$\mathbf{i}^{(t)} = \sigma \left( \mathbf{W}_i \mathbf{h}^{(t-1)} + \mathbf{U}_i \mathbf{x}^{(t)} + \mathbf{b}_i \right)$$

$$\mathbf{o}^{(t)} = \sigma \left( \mathbf{W}_o \mathbf{h}^{(t-1)} + \mathbf{U}_o \mathbf{x}^{(t)} + \mathbf{b}_o \right)$$

$$\tilde{\mathbf{c}}^{(t)} = \tanh \left( \mathbf{W}_c \mathbf{h}^{(t-1)} + \mathbf{U}_c \mathbf{x}^{(t)} + \mathbf{b}_c \right)$$

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

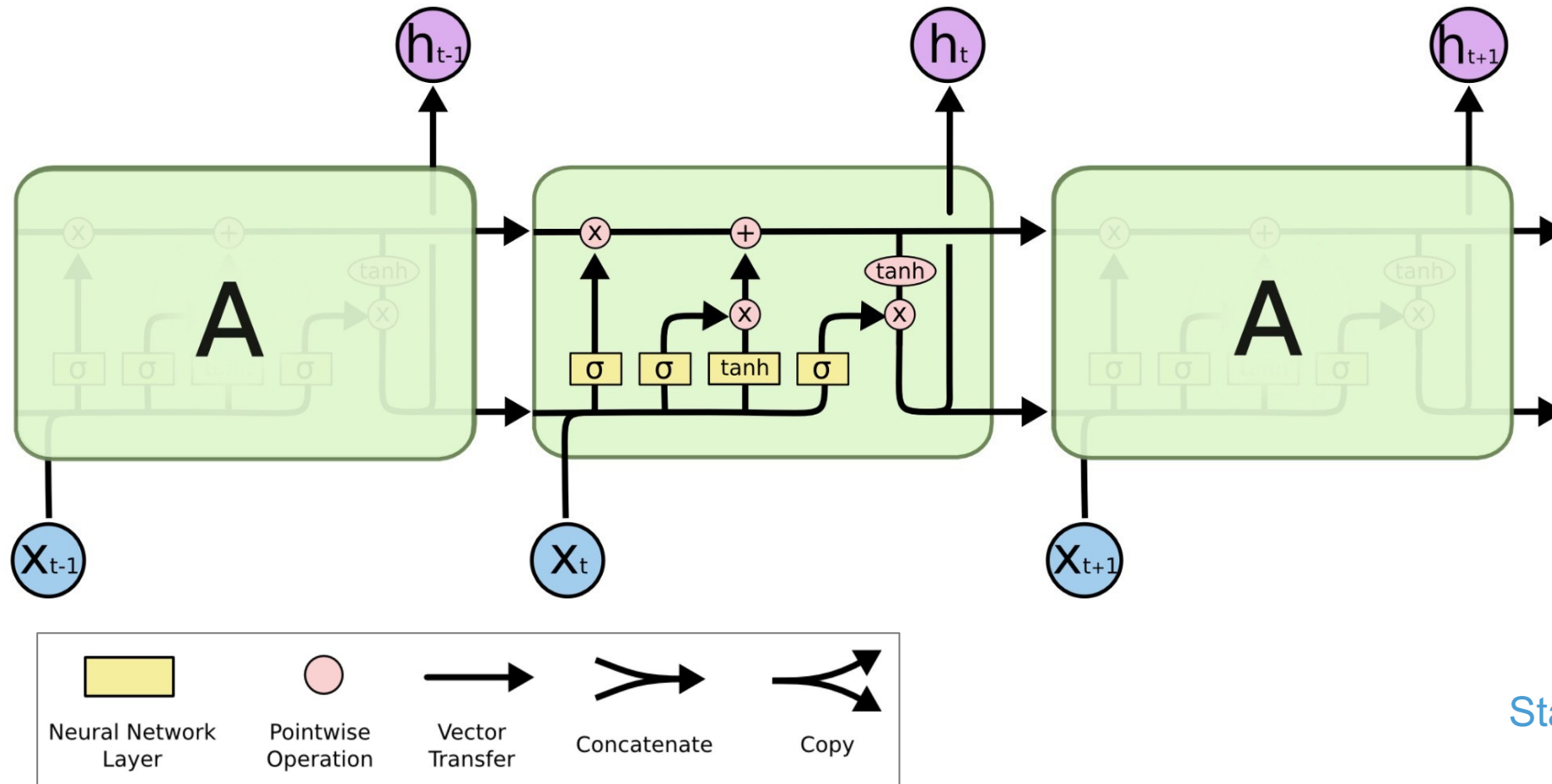
$$\mathbf{h}^{(t)} = \mathbf{o}^{(t)} \odot \tanh \mathbf{c}^{(t)}$$

All these are vectors of same length  $n$

Gates are applied using element-wise (or Hadamard) product:  $\odot$

# Long Short-Term Memory (LSTM)

You can think of the LSTM equations visually like this:

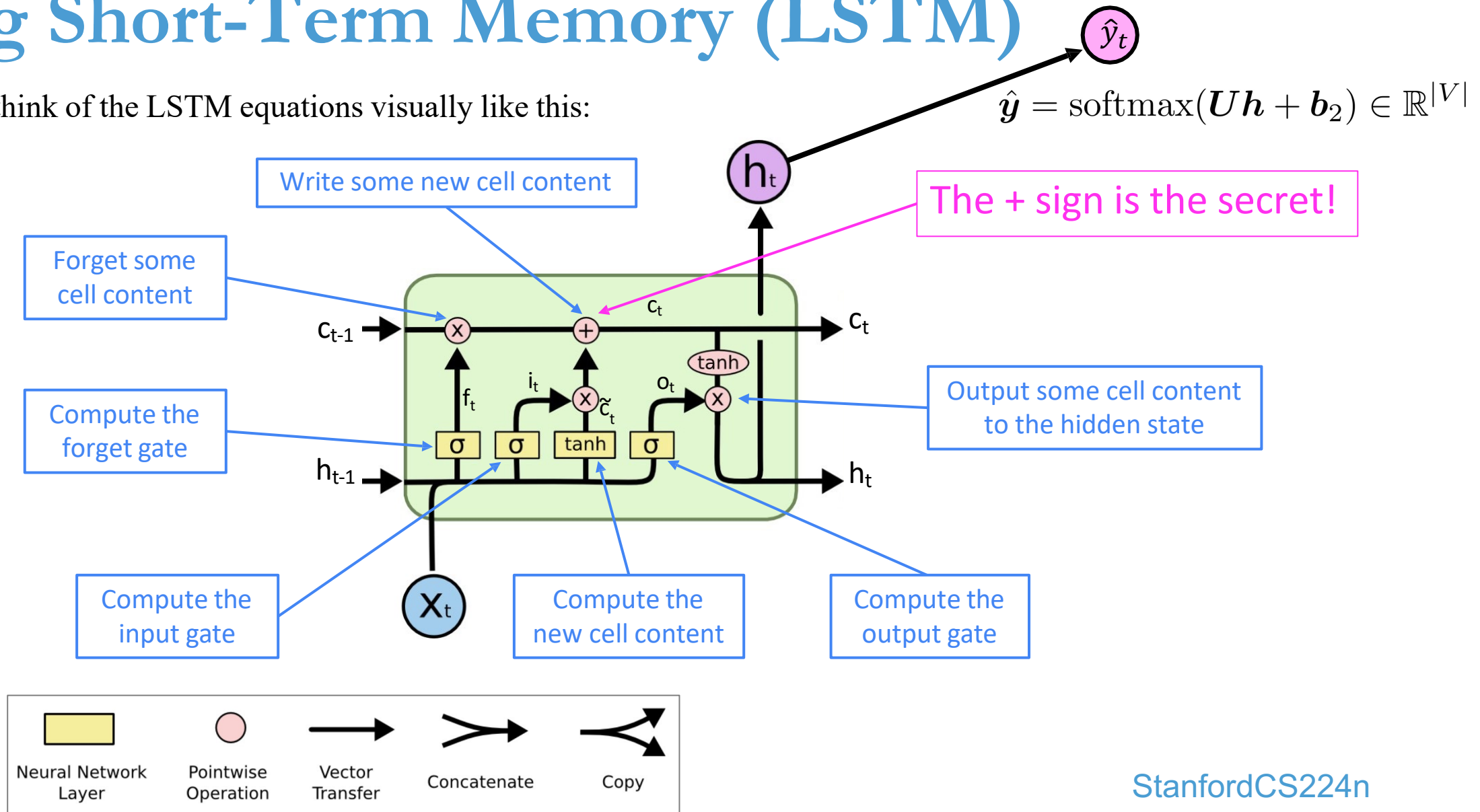


StanfordCS224n

Source: <http://colah.github.io/posts/2015-08-Understanding-LSTMs/>

# Long Short-Term Memory (LSTM)

You can think of the LSTM equations visually like this:



StanfordCS224n

Source: <http://colah.github.io/posts/2015-08-Understanding-LSTMs/>

# Preserving Long-Term Dependencies

## LSTMs vs. Vanilla RNNs

- **LSTM Advantage:**

The LSTM architecture makes it much easier for an RNN to preserve information over many timesteps. Example: If the forget gate is set to 1 and the input gate to 0 for a cell dimension, the corresponding cell state is preserved indefinitely.

- **Vanilla RNN Limitation:**

A vanilla RNN must learn a recurrent weight matrix that preserves information in the hidden state. In practice, vanilla RNNs typically manage to preserve information over only about 7 timesteps.

- **Extended Memory with LSTMs:**

LSTMs can effectively preserve information for around 100 timesteps, greatly enhancing the model's ability to capture long-term dependencies.

- **Alternative Approaches:**

There are other methods to create direct, linear pass-through connections that capture long-distance dependencies.

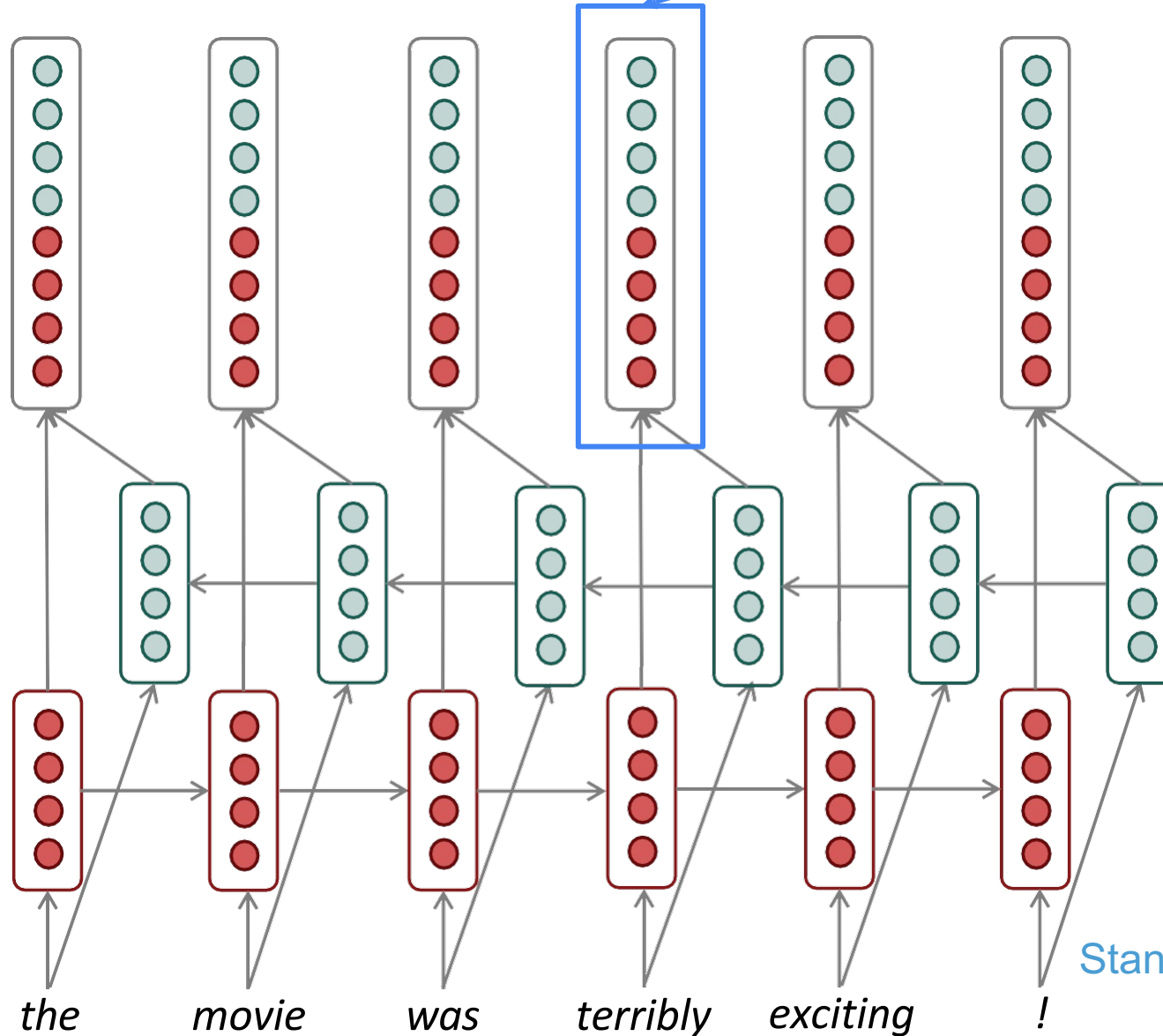
# Bidirectional RNNs

This contextual representation of “terribly” has both left and right context!

Concatenated  
hidden states

Backward RNN

Forward RNN



StanfordCS224n

# Bidirectional RNNs

On timestep  $t$ :

This is a general notation to mean “compute one forward step of the RNN” – it could be a simple RNN or LSTM computation.

Forward RNN  $\vec{h}^{(t)} = \text{RNN}_{\text{FW}}(\vec{h}^{(t-1)}, \mathbf{x}^{(t)})$

Backward RNN  $\overleftarrow{h}^{(t)} = \text{RNN}_{\text{BW}}(\overleftarrow{h}^{(t+1)}, \mathbf{x}^{(t)})$

Generally, these two RNNs have separate weights

Concatenated hidden states  $\mathbf{h}^{(t)} = [\vec{h}^{(t)}; \overleftarrow{h}^{(t)}]$

We regard this as “the hidden state” of a bidirectional RNN. This is what we pass on to the next parts of the network.

StanfordCS224n

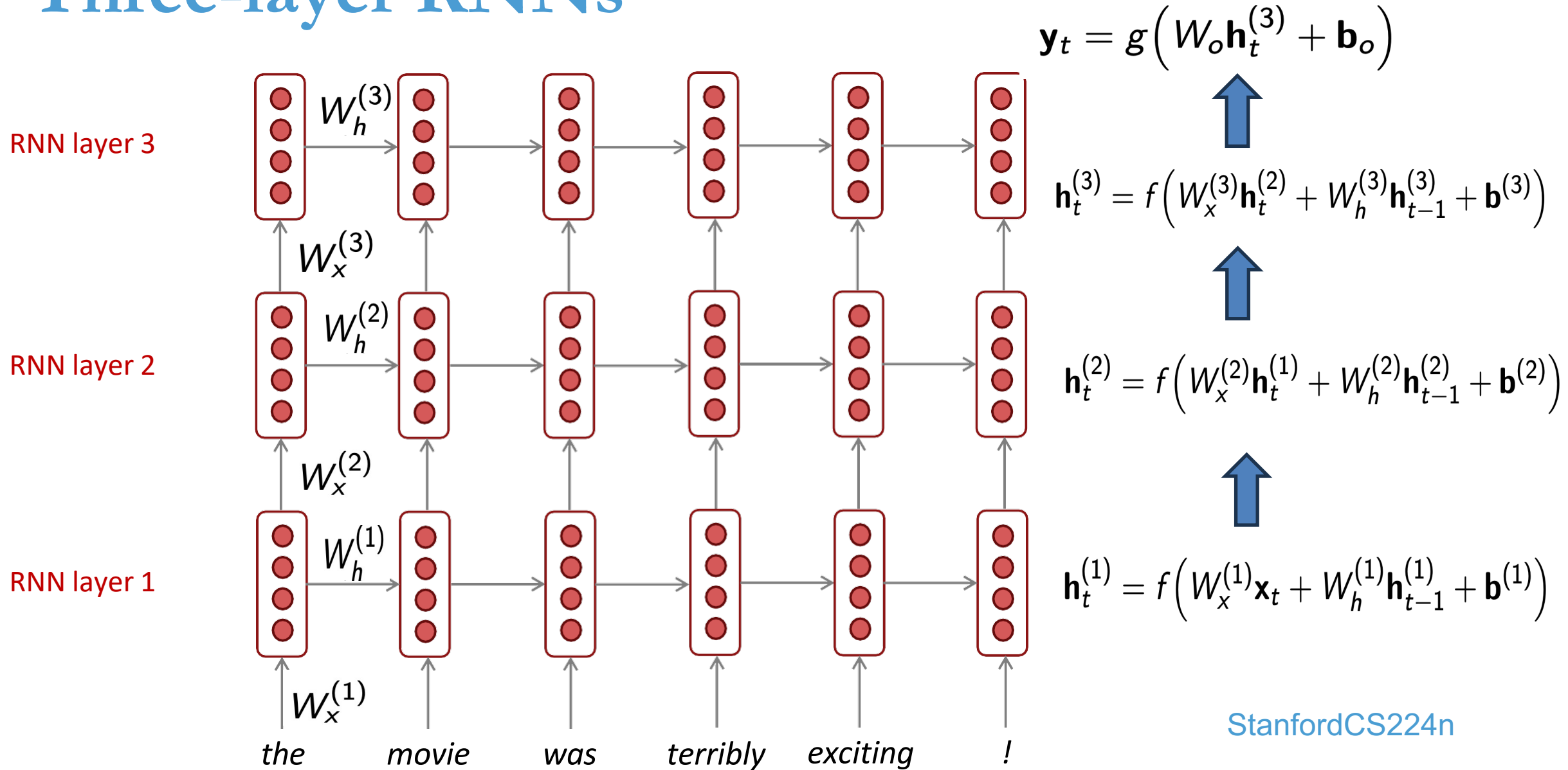
# Stacked RNNs

**Stacked RNNs (also known as deep RNNs):** multiple recurrent layers are placed on top of each other.

- ▶ The output of each RNN layer serves as the input to the next, creating a hierarchical representation of the sequential data. The **lower RNNs** should **compute lower-level features** and the **higher RNNs** should compute **higher-level features**.
- ▶ This architecture is used to capture complex, abstract temporal patterns. Hierarchical layers can learn more abstract features. Better capture complex temporal dependencies.
- ▶ Improved Performance: Often achieve higher accuracy on tasks such as language modeling, speech recognition, and time series prediction.
- ▶ However, they also introduce challenges such as increased training complexity and computational cost.
- ▶ With careful design and tuning, stacked RNNs are a powerful tool for sequence modeling.



# Three-layer RNNs

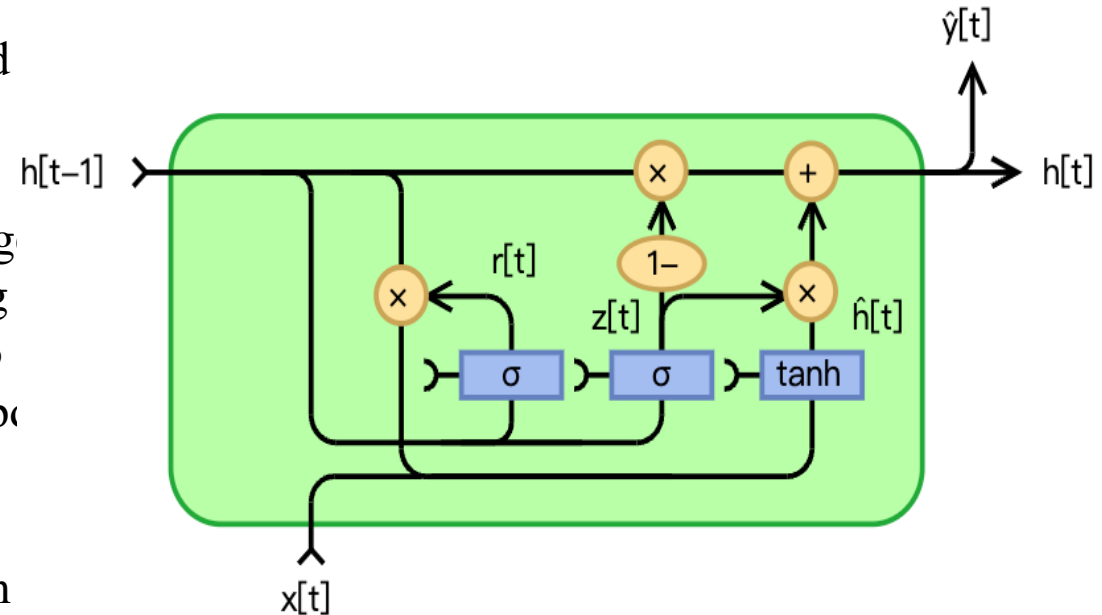


# Multi-layer RNNs in practice

- ❖ **Stacked RNNs allow for more complex, hierarchical representations:** Lower layers capture basic, low-level features, while higher layers integrate these into more abstract, high-level features.'
- ❖ **Performance improvements are observed with moderate stacking:** Empirical results (e.g., from Britz et al., 2017) indicate that 2–4 layers can be optimal for tasks such as machine translation.
- ❖ **Deep RNNs require architectural innovations:** For very deep RNNs (e.g., 8 layers or more), skip-connections or dense connections are necessary to maintain gradient flow and facilitate training.
- ❖ **Transformers push depth further with built-in residual connections:** Models like BERT, which can have 12–24 layers, offer a different approach to capturing long-range dependencies via self-attention and deep stacking.

# GRU (Gate Recurrent Unit)

- **Gate Recurrent Unit** is one of the ideas that has enabled RNN to become much better at capturing very **long-range dependencies** and made RNN much more effective.
- The GRU is like a LSTM with a **gating mechanism** to input or forget certain features, but lacks a context vector or output gate, resulting in **fewer parameters** than LSTM. Proposed as a simpler alternative to LSTM, the GRU merges the forget and input gates into a single update gate.
- GRUs are known for having fewer parameters than LSTMs, which lead to faster training and similar performance in many tasks.



Each GRU cell has two main gates:

- ▶ **Reset Gate ( $r(t)$ )**: Controls how much of the previous hidden state to forget.
- ▶ **Update Gate ( $z(t)$ )**: Decides how much of the candidate activation to use.
- ▶ The GRU combines these gates to update its hidden state without a separate cell state.

1. **Update Gate ( $z_t$ )**: Determines how much of the past hidden state  $h_{t-1}$  should be retained and how much of the new candidate state  $\hat{h}_{t-1}$  should be added to form the current hidden state.

$$z_t = \sigma(W_z \cdot x_t + U_z \cdot h_{t-1} + b_z)$$

2. **Reset Gate ( $r_t$ )**: Determines how much of the past hidden state  $h_{t-1}$  contributes to the computation of the new candidate state  $\hat{h}_{t-1}$ .

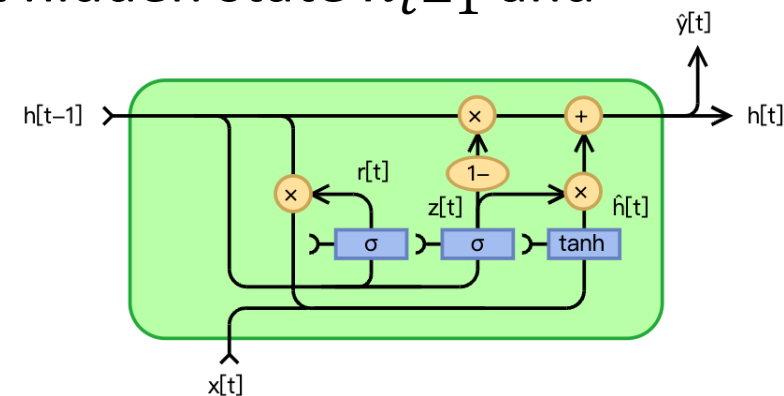
$$r_t = \sigma(W_r \cdot x_t + U_r \cdot h_{t-1} + b_r)$$

3. **Candidate State ( $\hat{h}_t$ )**: New information computed at the current time step.

$$\hat{h}_t = \tanh(W_h \cdot x_t + U_h \cdot (r_t \odot h_{t-1}) + b_h)$$

4. **Hidden State Update**: Combines contributions from the past hidden state  $h_{t-1}$  and the new candidate state  $\hat{h}_{t-1}$  using the update gate  $z_t$ .

$$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot \hat{h}_t$$



# Content

1 Motivation to Sequence Modeling

2 Recurrent Neural Networks (RNNs)

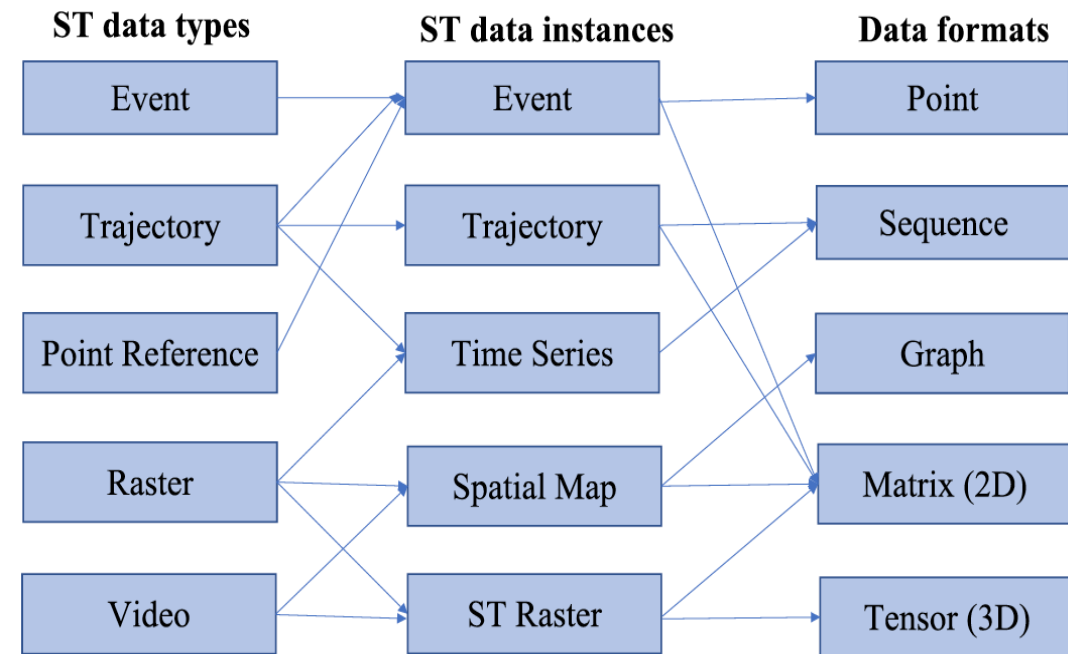
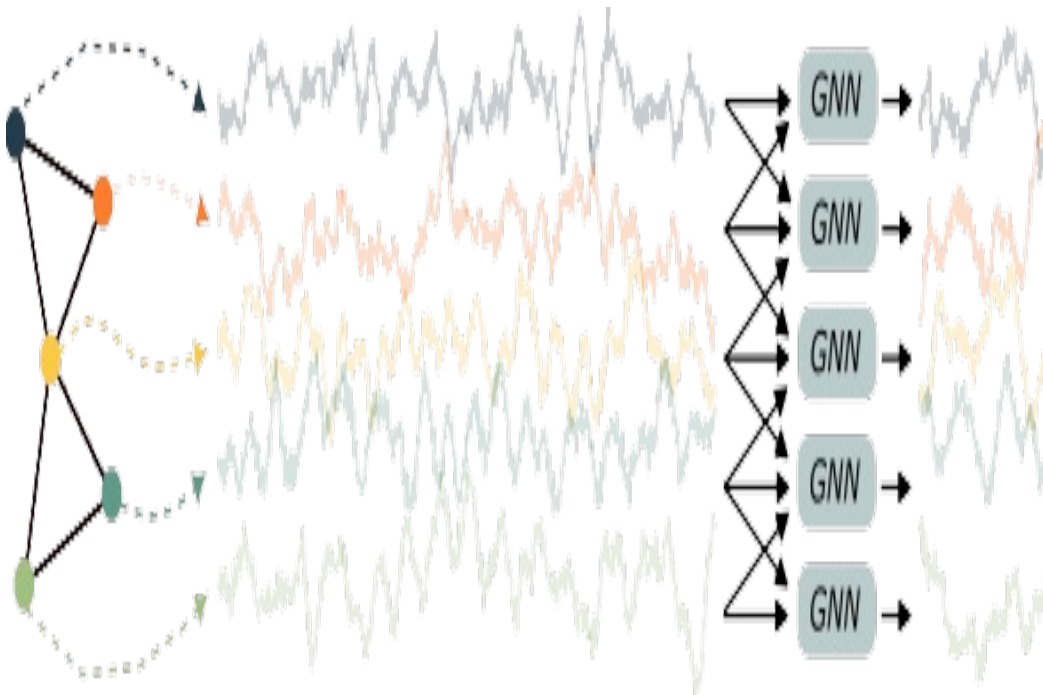
3 Extensions of RNN

**4 Motivation to Spatio-temporal Modeling**

5 Deep ST Models and Applications

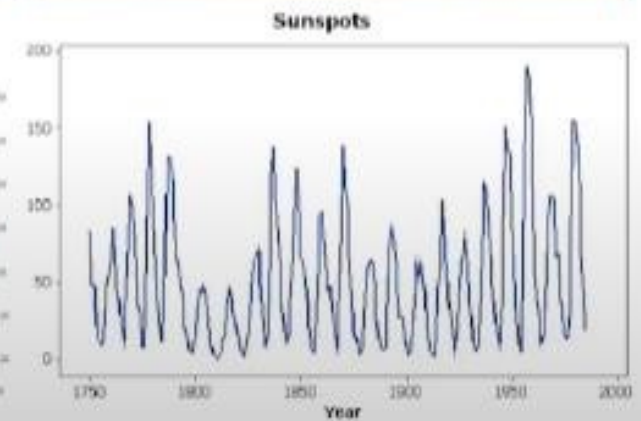
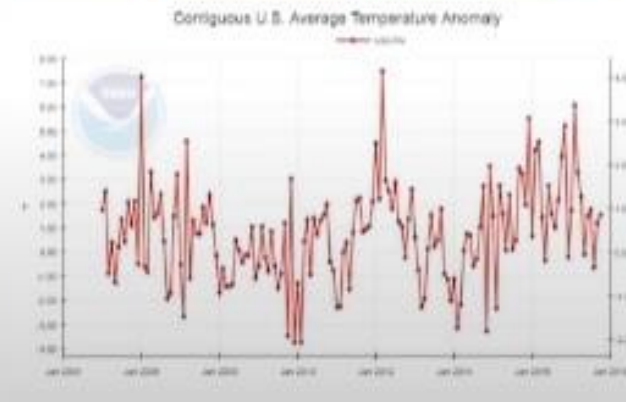
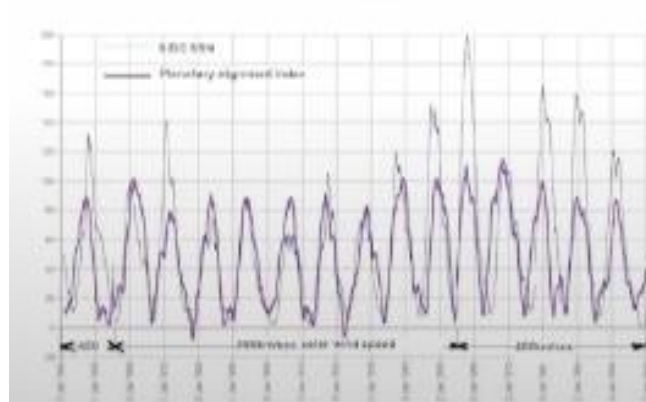
# Spatio-temporal Data

Spatiotemporal data, however, are data derived from measurements, which take into account both the parameters of space and time.



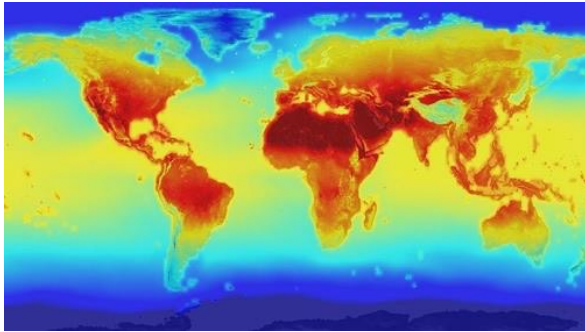
# Spatio-temporal Data – Time series

- Time series is a sequence of data points collected or recorded at specific time intervals, showing how a variable changes over time
- Multivariate time series is a typical spatio-temporal data

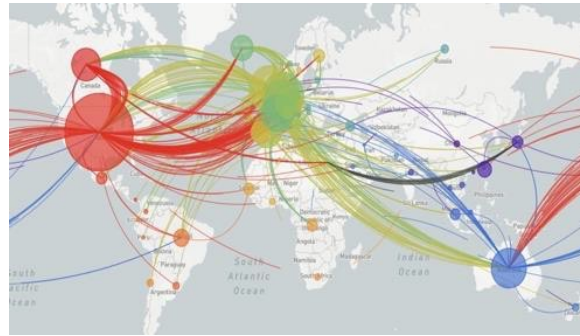


# Spatio-temporal Data – Time series

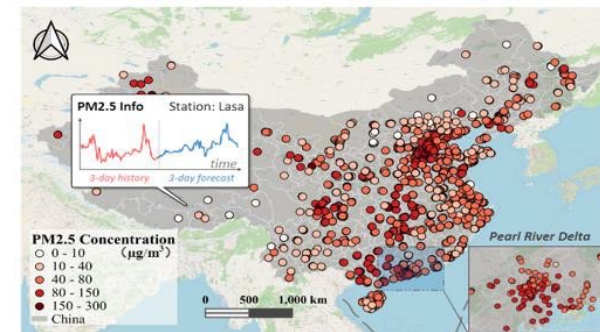
With recent advances in sensing technologies, a myriad of **Time Series (TS) Data** has been collected and contributed to various disciplines



Climate



Epidemiology



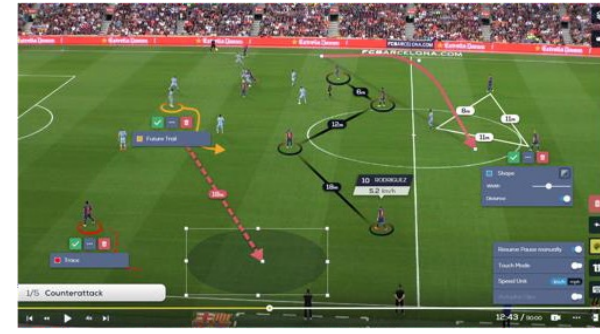
Environment



Social Science



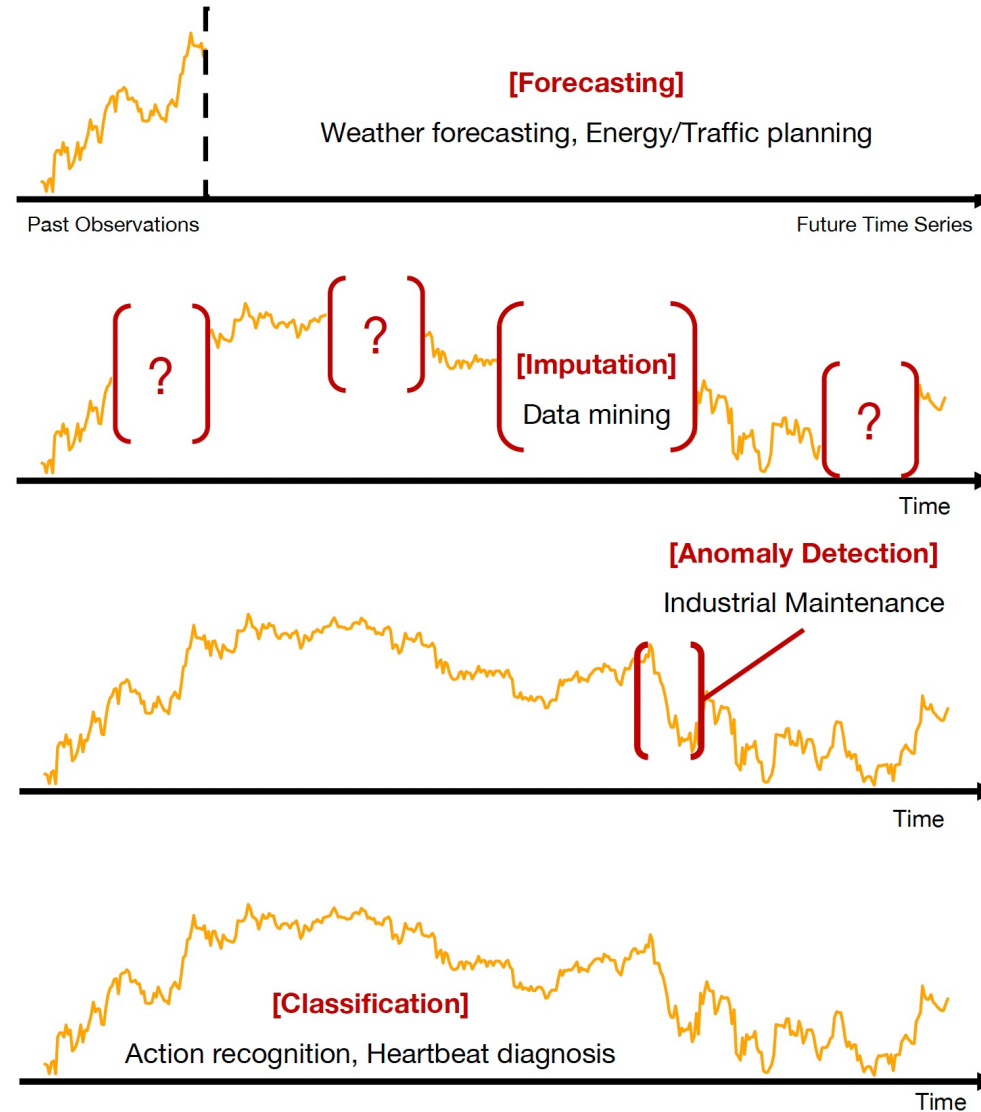
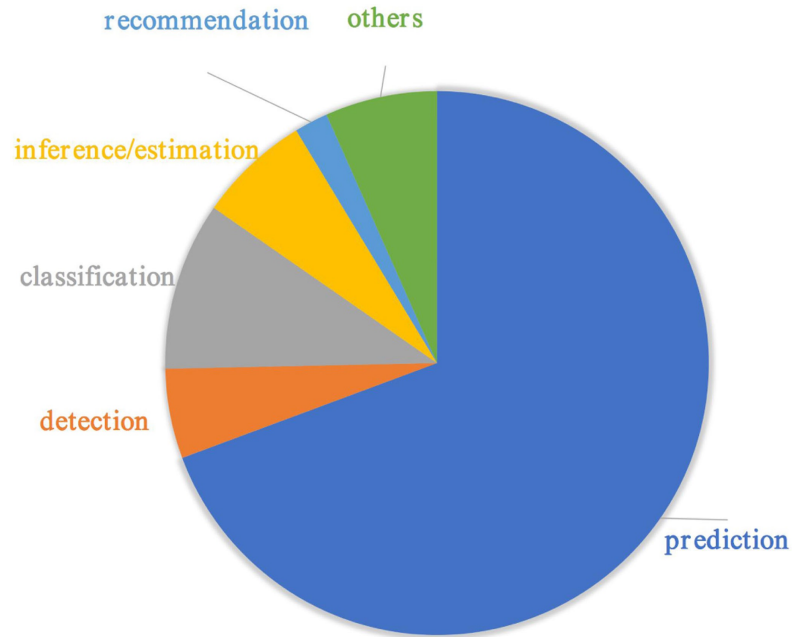
Transportation



Sports Analysis

# Spatio-temporal Tasks

- Predictive learning:  
Prediction; Forecasting
- Classification
- Estimation and Inference
- Anomaly detection



Wang, Y., Wu, H., Dong, J., Liu, Y., Long, M. and Wang, J., 2024. Deep time series models: A comprehensive survey and benchmark. *arXiv preprint arXiv:2407.13278*.

# Content

1 Motivation to Sequence Modeling

2 Recurrent Neural Networks (RNNs)

3 Extensions of RNN

4 Motivation to Spatio-temporal Modeling

**5 Deep ST Models and Applications**

# Problem formulations

Table 1. Summary of deep Forecasting Models based on Forecast and Model Type

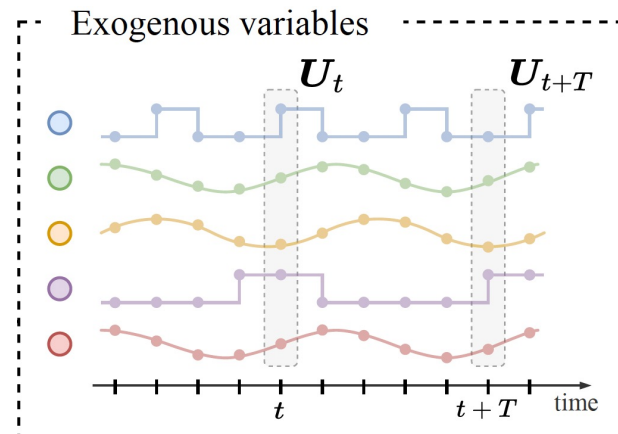
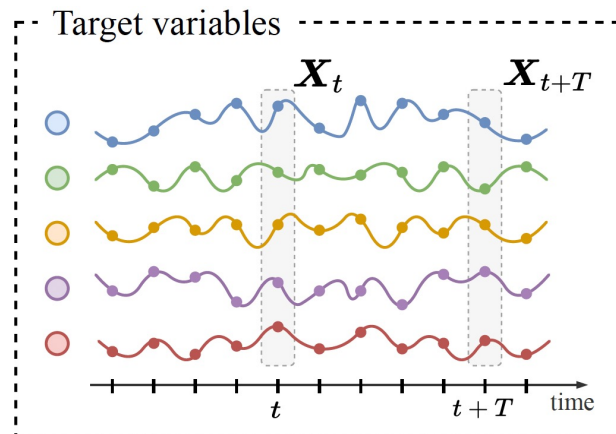
Forecast type	Model type	Formulation
Point	Local univariate	$\hat{z}_{i,t+1:t+h} = \Psi(z_{i,1:t}, X_{i,1:t+h})$
	Global univariate	$\hat{z}_{i,t+1:t+h} = \Psi(z_{i,1:t}, X_{i,1:t+h}, \Phi)$
	Multivariate	$\hat{Z}_{t+1:t+h} = \Psi(Z_{1:t}, X_{1:t+h}, \Phi)$
Probabilistic	Local univariate	$P(z_{i,t+1:t+h}   z_{i,1:t}, X_{i,1:t+h}; \theta_i), \quad \theta_i = \Psi(z_{i,1:t}, X_{i,1:t+h})$
	Global univariate	$P(z_{i,t+1:t+h}   Z_{1:t}, X_{1:t+h}; \theta_i), \quad \theta_i = \Psi(z_{i,1:t}, X_{i,1:t+h}, \Phi)$
	Multivariate	$P(Z_{t+1:t+h}   Z_{1:t}, X_{1:t+h}; \theta), \quad \theta = \Psi(Z_{1:t}, X_{1:t+h}, \Phi)$

For one-step and multi-step forecasting models  $h = 1$  and  $h > 1$ , respectively.

# Setup of multivariate time series

A set of  $N$  correlated time series, where each  $i$ -th time series is associated with:

- an observation vector  $\mathbf{x}_t^i \in R^{d_x}$  at each time step  $t$ ;
- a vector of exogenous variable  $\mathbf{u}_t^i \in R^{d_u}$  at each time step  $t$ ;
- a vector of static (time-independent) attributes  $\mathbf{v}_t^i \in R^{d_v}$ .



$$\mathbf{V} = \begin{bmatrix} \text{blue} & \text{blue} & \text{blue} & \text{blue} \\ \text{green} & \text{green} & \text{green} & \text{green} \\ \text{orange} & \text{orange} & \text{orange} & \text{orange} \\ \text{purple} & \text{purple} & \text{purple} & \text{purple} \\ \text{red} & \text{red} & \text{red} & \text{red} \end{bmatrix} \quad \mathcal{E}_t = \left\{ \begin{bmatrix} \text{green} & \text{green} & \text{blue} \\ \text{green} & \text{yellow} & \text{yellow} \\ \text{blue} & \text{yellow} & \text{yellow} \\ \text{yellow} & \text{purple} & \text{purple} \\ \text{yellow} & \text{orange} & \text{orange} \\ \text{purple} & \text{red} & \text{red} \end{bmatrix} \begin{matrix} \text{green circle} \\ \text{yellow circle} \\ \text{blue circle} \\ \text{orange circle} \\ \text{purple circle} \\ \text{red circle} \end{matrix} \right\}$$

Matrices denote the stacked  $N$  observations at time  $t$ ,

e.g.,  $\mathbf{X}_t \in R^{N \times d_x}$ ,  $\mathbf{U}_t \in R^{N \times d_u}$ .

# Setup of multivariate time series

We consider a setup where observations have been generated by a **time-invariant** spatiotemporal stochastic process such that

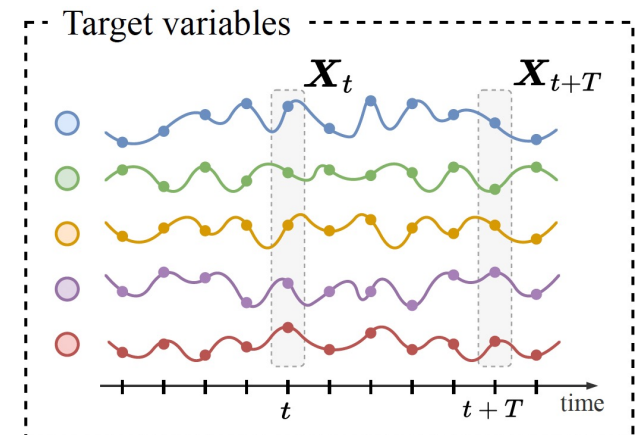
$$\mathbf{x}_t^i \sim p^i(\mathbf{x}_t^i | \mathbf{X}_{<t}, \mathbf{U}_{\leq t}, \mathbf{V}) \quad \forall i = 1, \dots, N;$$

$\mathbf{X}_{t:t+T}$ : the sequence of observations within time interval  $[t, t + T)$ ;

$\mathbf{X}_{<t}$ : observations at time steps up to  $t$  (excluded)

Note that the time series:

- can be generated by **different processes**,
- can **depend on each other**,
- are assumed **homogenous, synchronous, regularly sampled**.



# Relational information

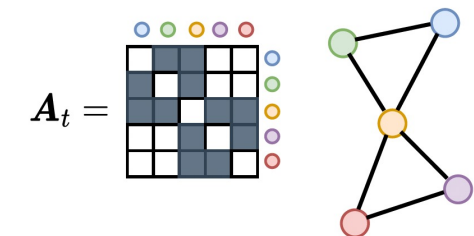
- Pairwise relationships existing among the time series at time step  $t$  can be encoded by a **adjacency matrix**  $\mathbf{A}_t \in \{0, 1\}^{N \times N}$ .
- $\mathbf{A}_t$  can be asymmetric and dynamic (can vary with  $t$ ).
- optional edge attributes  $\mathbf{e}_t^{ij} \in \mathbb{R}^{d_e}$  can be associated to each non-zero entry of  $\mathbf{A}_t$ .
- $\mathcal{E}_t \doteq \{ \langle (i, j), \mathbf{e}_t^{ij} \rangle \mid \forall i, j : \mathbf{A}_t[i, j] \neq 0 \}$ : the set of attributed edges encoding all the available relational information
- Tuple  $\mathcal{G}_t \doteq \langle \mathbf{X}_t, \mathbf{U}_t, \mathcal{E}_t, \mathbf{V} \rangle$  indicates all the available information at time step  $t$ .

- Pairwise relationships existing among the time series at time step  $t$  can be encoded by a **adjacency matrix**  $\mathbf{A}_t \in \{0, 1\}^{N \times N}$ .
- $\mathbf{A}_t$  can be asymmetric and dynamic (can vary with  $t$ ).
- optional edge attributes  $\mathbf{e}_t^{ij} \in \mathbb{R}^{d_e}$  can be associated to each non-zero entry of  $\mathbf{A}_t$ .
- $\mathcal{E}_t \doteq$  : the set of attributed edges encoding all the available relational information
- Tuple indicates all the available information at time step  $t$ .

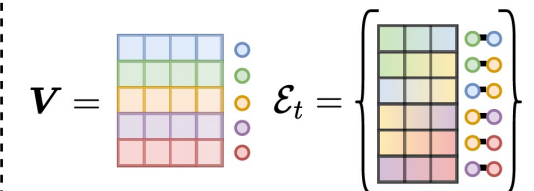
Nodes (sensors)

$$\mathcal{V} = \{ \text{green, blue, yellow, purple, red} \}$$

Edges (functional dependencies)



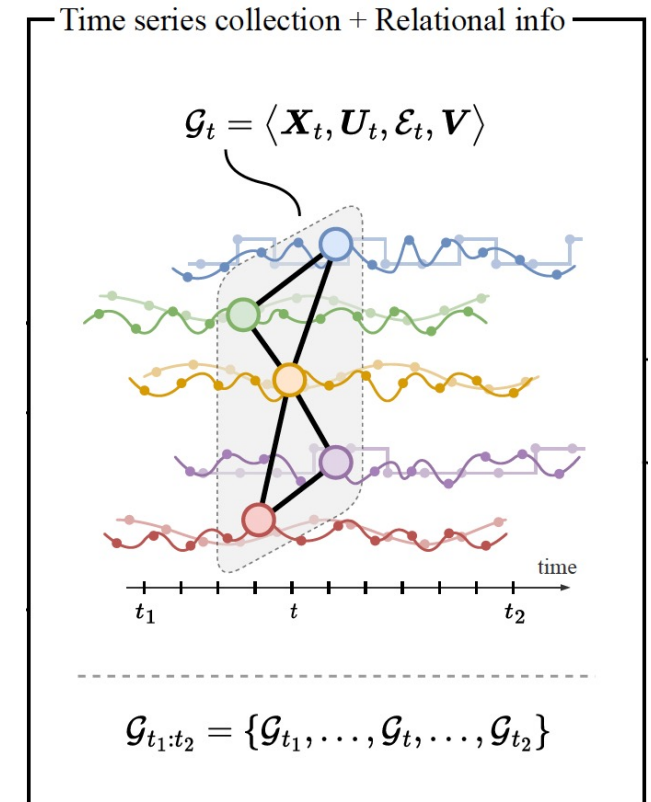
Node/edge attributes



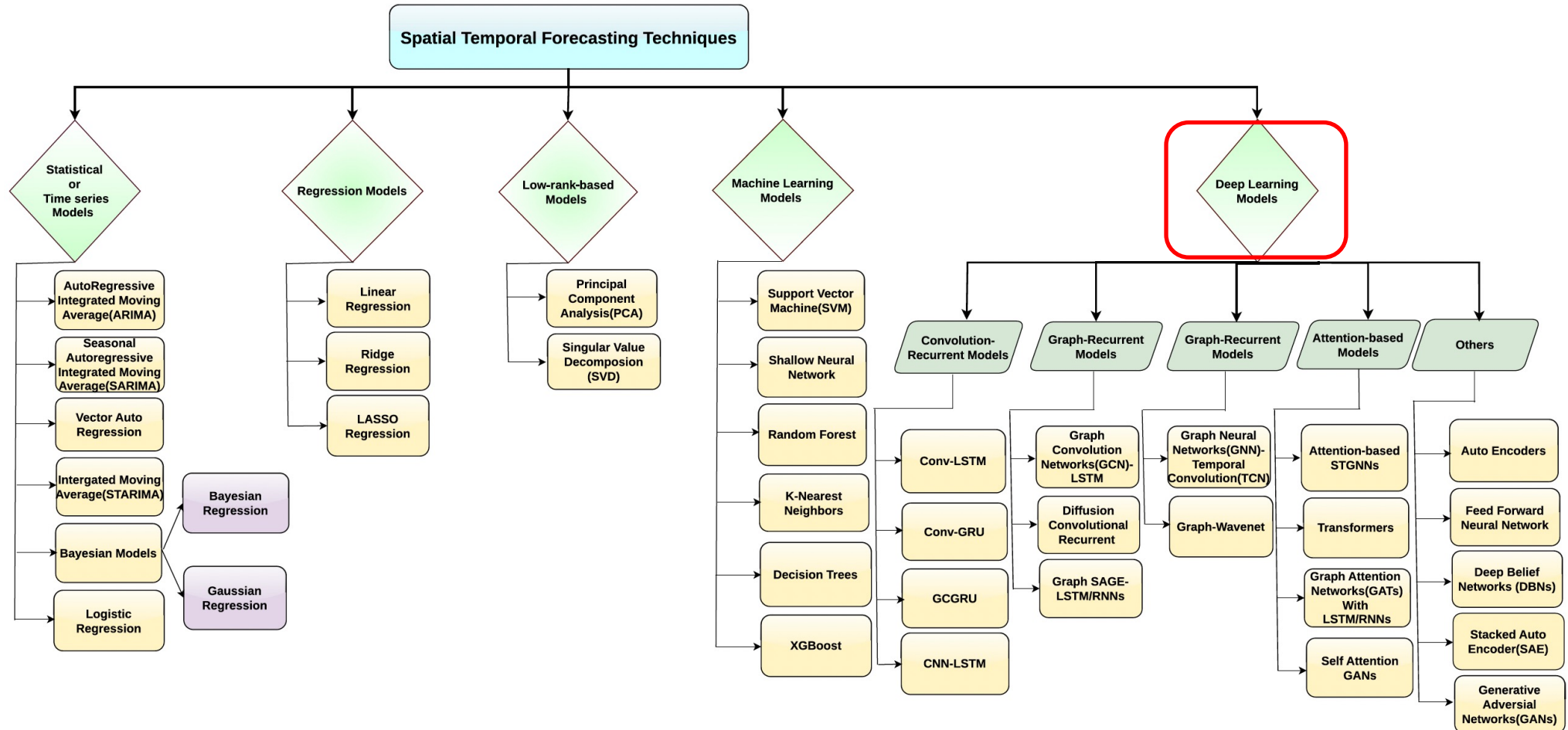
Cini, A., Marisca, I., Zambon, D. and Alippi, C., 2023. Graph deep learning for time series forecasting. arXiv preprint arXiv:2310.15978.

# Relational information

- The term **spatial** refers to the dimension of size  $N$ , that spans the time series collection; in the case of fMRI, the term spatial reflects the fact that each time series might correspond to a different physical location.
- We use the terms **node** and **sensor** to indicate the  $N$  entities generating the time series.
- We assume the existence of **functional dependencies** between the time series.
  - e.g., forecasts for one time series can be improved by accounting for the past values of other time series.

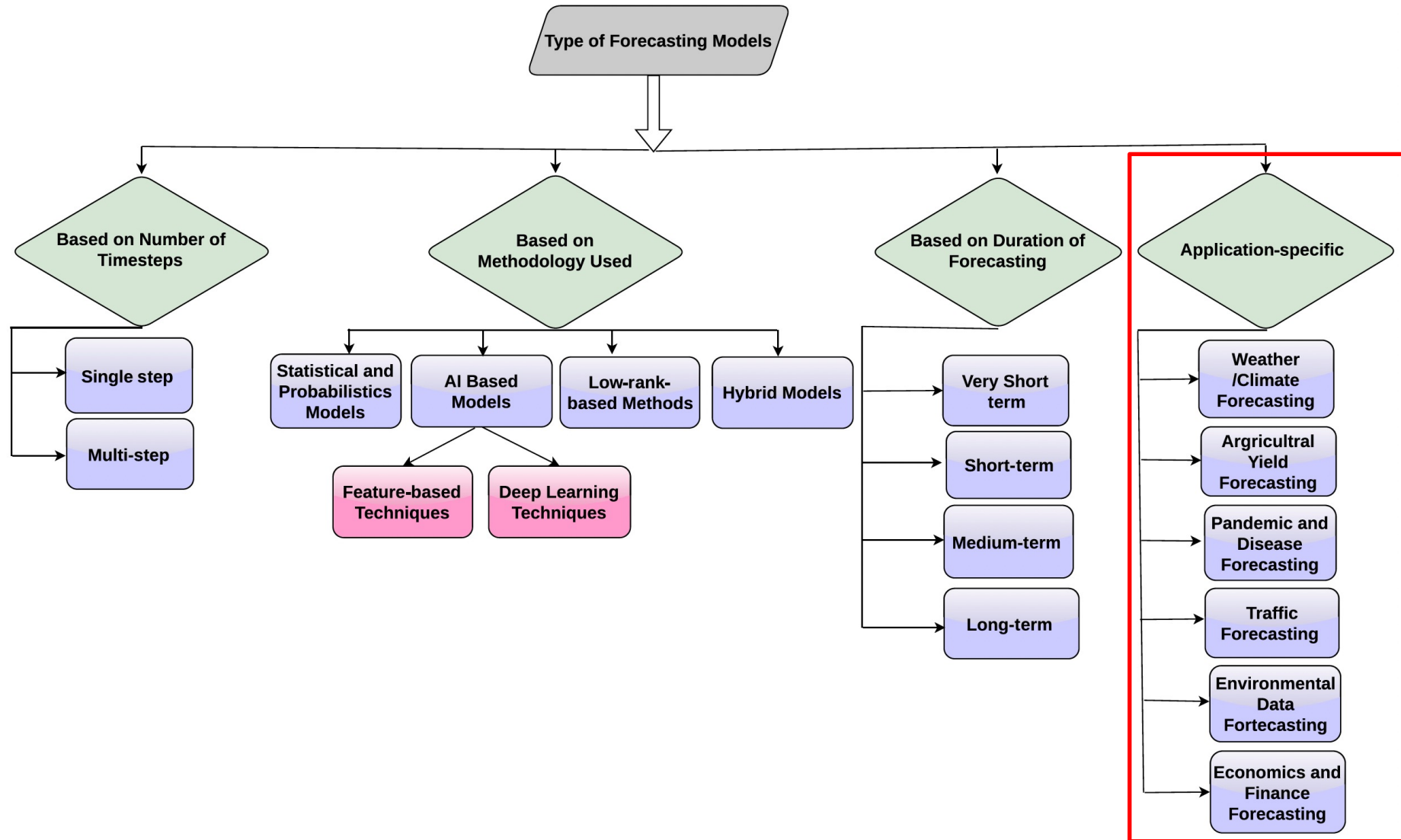


# Spatio-temporal Modeling



Kumar, R., Bhanu, M., Mendes-Moreira, J. and Chandra, J., 2024. Spatio-Temporal Predictive Modeling Techniques for Different Domains: a Survey. *ACM Computing Surveys*, 57(2), pp.1-42.

# Applications



# Application - Transportation

With the rapid growth of transportation data from sensors (e.g., loop detectors, cameras, GPS), there is an urgent need to use deep learning to model the complex spatiotemporal correlations for tasks like:

- Traffic flow prediction
- Traffic incident detection
- Traffic congestion prediction

Transportation data can appear in various spatiotemporal forms:

- ST Raster: Traffic flow matrices (sensor  $\times$  time)
- Graphs: Sensor networks modeled by road connections
- Time Series: Single-road traffic histories

Modeling Approaches:

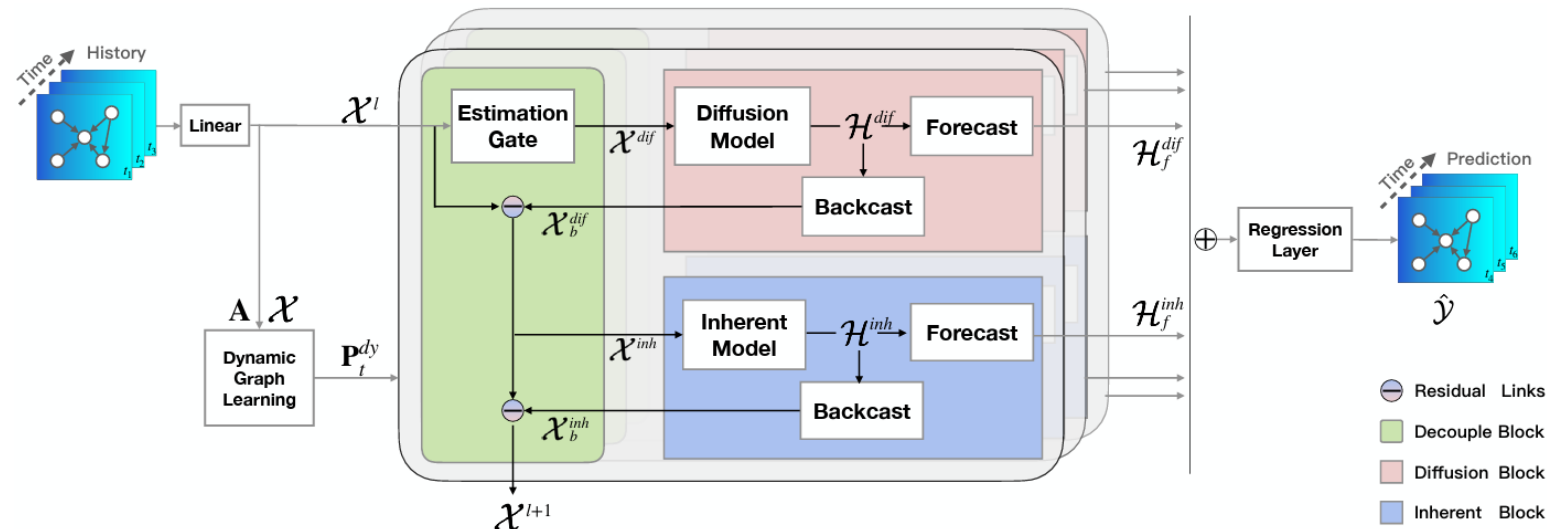
- GraphCNNs for sensor network graphs
- RNN/LSTM for single-road time series

Additionally, transportation data is influenced by external factors (e.g., weather, holidays, events), so models must effectively fuse external features with traffic data for better prediction accuracy.

# Application - Transportation

Introduce a novel framework that decouples traffic data into diffusion and inherent components to better model complex spatial-temporal dependencies in traffic forecasting.

- Decoupled Spatial-Temporal Framework (DSTF): Separates traffic data into diffusion signals (capturing spatial dependencies) and inherent signals (capturing temporal patterns).
- Dynamic Graph Learning Module: Learns dynamic characteristics of traffic networks over time.
- Residual Decomposition Mechanism: Enhances the model's ability to capture complex patterns by decomposing residuals.



# Application - On-Demand Service

With the rise of on-demand service platforms (e.g., Uber, DiDi, Mobike, GoGoVan), a large volume of spatiotemporal (ST) data is generated, involving customer locations and service times.

To better meet real-time demand and optimize services, accurate demand-supply prediction across locations and times is crucial.

## Deep Learning Approaches:

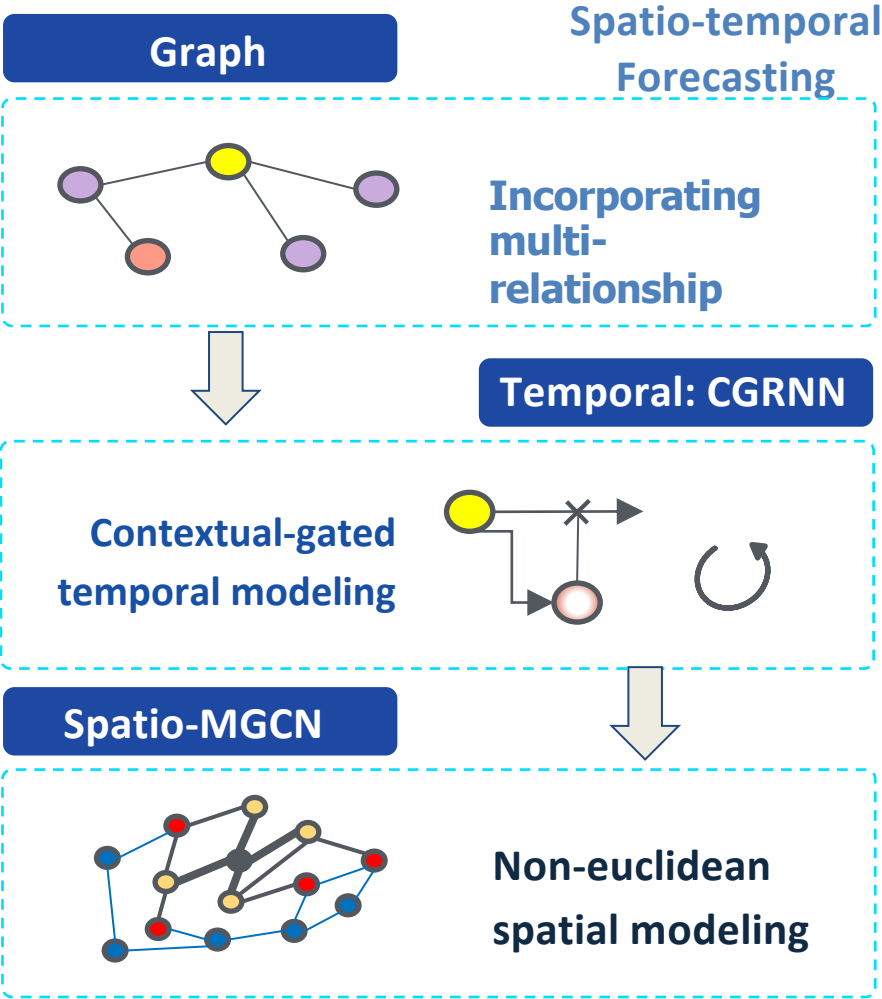
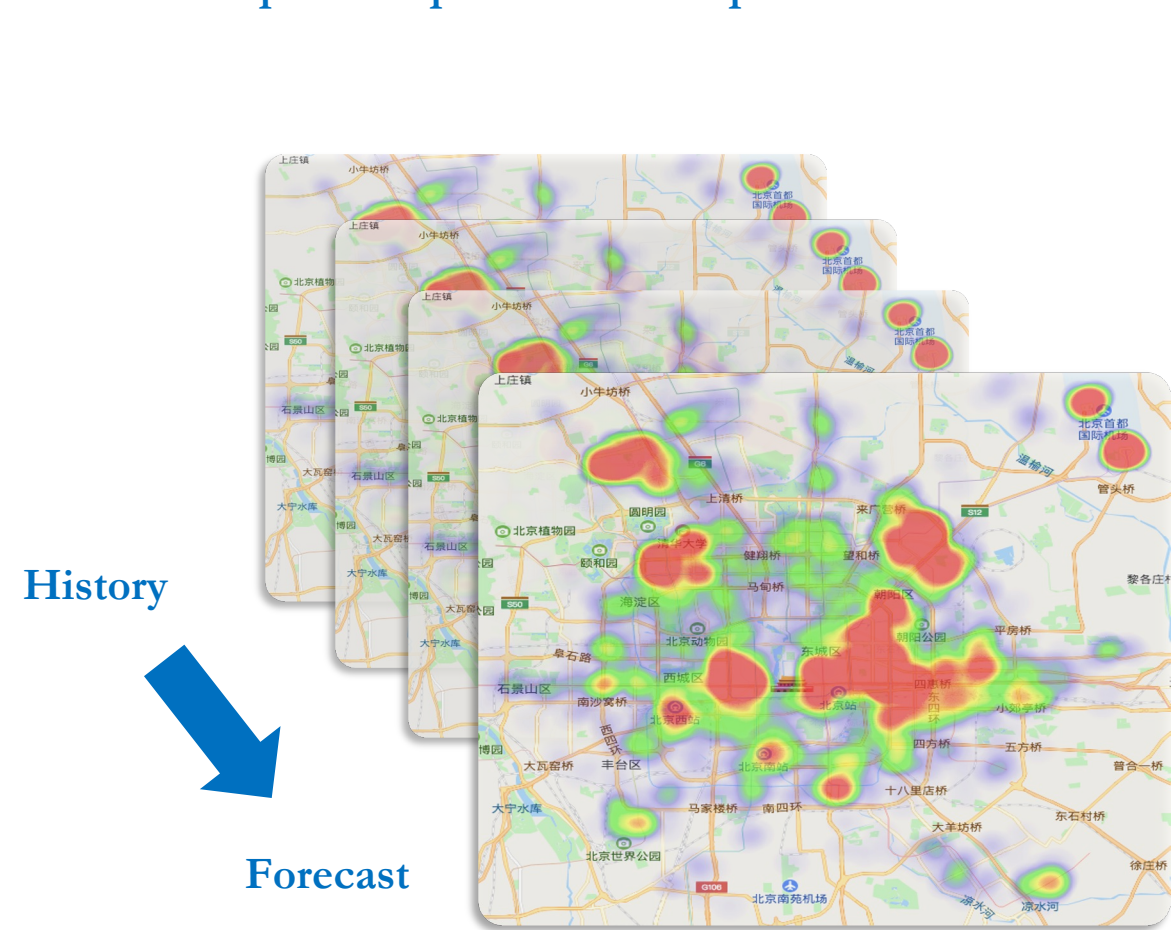
- Dockless bike-sharing: Deep learning methods predict demand-supply distributions.
- Bike-sharing systems: Graph CNN models forecast hourly bike demand at stations by modeling bike flow as a graph.
- Taxi services: LSTM models predict area-specific taxi demand.
- Ride-hailing platforms: ResNet models predict supply-demand patterns.

## Modeling Strategy:

- Represent demand-supply across city regions as spatial maps or raster tensors.
- Apply CNNs, RNNs, or hybrid deep models for feature extraction and future prediction.

# Application - On-Demand Service

ST-MGCN: Spatiotemporal Multi-Graph Convolution Network:



# Application - On-Demand Service

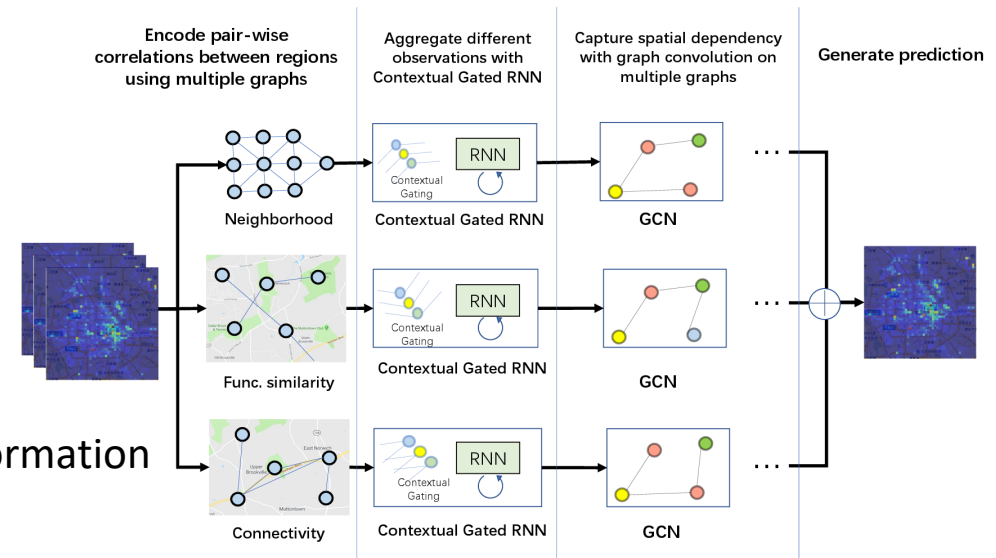
## ST-MGCN: Spatiotemporal Multi-Graph Convolution Network:

### Key Idea:

Model non-Euclidean spatial dependencies and global temporal dynamics simultaneously for accurate region-level ride-hailing demand forecasting.

### Model Architecture:

- **Multi-Graph Construction:**  
Build three graphs to capture different spatial relationships:
  - Neighborhood Graph (adjacent regions)
  - Functional Similarity Graph (similar POI surroundings)
  - Transportation Connectivity Graph (road network links)
- **Multi-Graph Convolution:**  
Perform graph convolutions over multiple graphs to aggregate information from spatially correlated regions (both near and distant).
- **Contextual Gated RNN (CGRNN):**  
Augments RNN with global context-aware gating.  
Dynamically reweights different time steps based on global demand patterns.
- **Prediction Head:**  
Outputs future region-level ride-hailing demand after spatial and temporal aggregation.



# Application - On-Demand Service

ST-MGCN: Spatiotemporal Multi-Graph Convolution Network:

## Graph Generation

Neighborhood

$$A_{N,ij} = \begin{cases} 0, & v_i \text{ and } v_j \text{ are adjacent} \\ 1, & \text{otherwise} \end{cases}$$

POI similarity

$$A_{S,ij} = \text{sim}(P_{v_i}, P_{v_j})$$

Road connectivity

$$A_{C,ij} = \max(0, \text{conn}(v_i, v_j) - A_{N,ij})$$

## Temporal modeling

CGRNN

- Use 1-layer GCN to invoke context information
- Use spatial global pooling to get temporal gate
- Apply gate to input signal
- Aggregate gated signal by share-weight RNN

## Spatial modeling

MGCN

- Use stacked GCN layer to extract spatial information
- The locality is determined by graph Laplacian and convolution degree
- A proper way to extract spatial information under arbitrary relationship

# Application - On-Demand Service

## ST-MGCN: Experiments

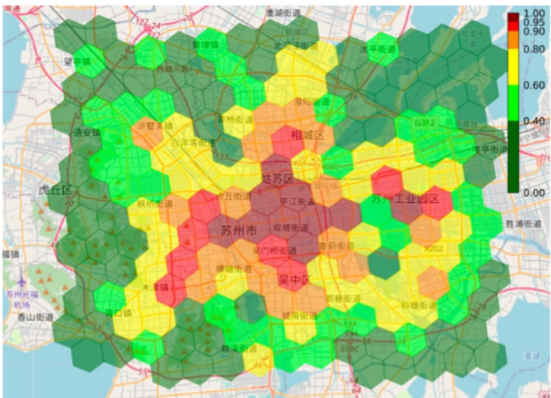
Method	Beijing		Shanghai	
	RMSE	MAPE(%)	RMSE	MAPE(%)
HA	16.14	23.9	17.15	34.8
LASSO	14.24±0.14	23.8±0.8	10.62±0.06	22.9±0.8
Ridge	14.24±0.11	23.8±0.9	10.61±0.04	23.1±0.8
VAR	13.32±0.17	22.4±1.6	10.54±0.18	23.7±1.4
STAR	13.16±0.22	22.2±1.9	10.52±0.21	23.2±1.4
GBM	13.66±0.16	23.1±1.5	10.25±0.11	23.4±1.2
STResNet	11.77±0.95	14.8±6.0	9.87±0.94	14.9±6.0
DMVST-Net	11.62±0.48	12.3±5.5	9.61±0.44	13.8±1.2
ST-GCN	11.62±0.36	10.1±5.1	9.29±0.31	11.2±1.3
<i><b>ST-MGCN</b></i>	<b>10.78±0.25</b>	<b>8.8±3.5</b>	<b>8.30±0.16</b>	<b>9.3±0.9</b>

# Application - On-Demand Service

## OD-CED: Spatio-temporal Prediction of Fine-Grained Origin-Destination Matrices with Applications in Ridesharing

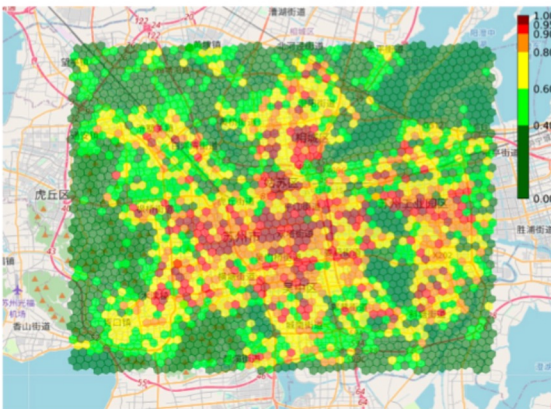
### Challenges:

1. **Scalability** – OD matrices grow exponentially with more spatial divisions.
2. **Data Sparsity** – Over 90% of fine-grained OD flows have zero demand.
3. **Semantic & Geographical Dependencies** – Travel demand is influenced by both regional function (e.g., residential vs. commercial) and spatial proximity.



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

(a)



$$\begin{matrix} & g_1 & g_2 & g_3 & \cdot & g_j & \cdot & g_{2531} \\ \begin{matrix} g_1 \\ g_2 \\ g_3 \\ \cdot \\ g_i \\ \cdot \\ g_{2531} \end{matrix} & \left[ \begin{array}{ccccccc} 1 & 0 & 2 & \cdot & \cdot & \cdot & 0 \\ 0 & 0 & 0 & \cdot & \cdot & \cdot & 2 \\ 0 & 0 & 0 & \cdot & \cdot & \cdot & 0 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 0 \\ \cdot & \cdot & \cdot & \cdot & x_{i,j} & \cdot & 0 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{array} \right]
 \end{matrix}$$

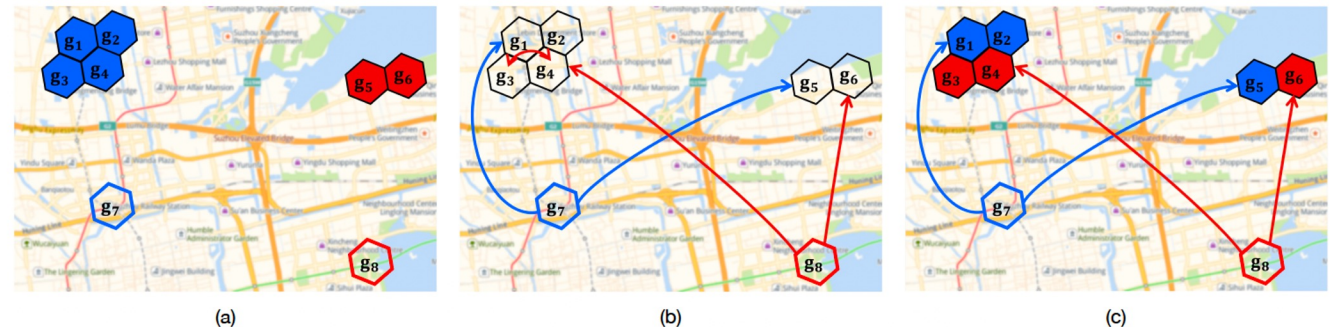
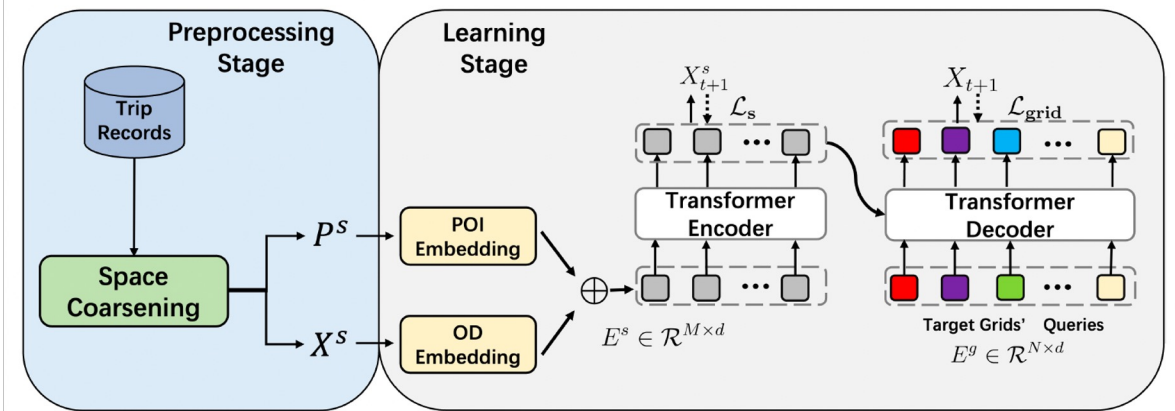
(b)

# Application - On-Demand Service

## OD-CED: Spatio-temporal Prediction of Fine-Grained Origin-Destination Matrices with Applications in Ridesharing

### OD-CED Model: A Novel OD Prediction Framework

- **Space Coarsening Module:** Merges fine-grained cells into super-cells to mitigate sparsity.
- **Encoder-Decoder Architecture:** Captures semantic and geographical dependencies effectively.
- **Permutation-Invariant OD Embedding:** Learns robust representations of OD flows.



# Application - On-Demand Service

## OD-CED: Experiments

### Dataset Performance Comparison (City-C & City-S):

- **City-C:**
  - RMSE reduced from **1.255 (GEML)** → **0.905 (OD-CED)** (~28% improvement).
  - wMAPE reduced from **0.667 (GEML)** → **0.411 (OD-CED)** (~39% improvement).
- **City-S:**
  - RMSE reduced from **1.146 (GEML)** → **0.740 (OD-CED)** (~35% improvement).
  - wMAPE reduced from **0.605 (GEML)** → **0.323 (OD-CED)** (~47% improvement).

### Training Time Comparison (per epoch on V100 GPU):

- OD-CED: **22.12s**
- STGCN: **28.81s**
- GEML (state-of-the-art): **39.63s**
- CSTN & MRSTN: **1200+ seconds**
- OD-CED is **2x faster than GEML** and **over 50x faster** than CNN-based methods.

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
<b>OD-CED</b>	<b>0.411</b>	<b>0.905</b>	<b>0.776</b>	<b>0.323</b>	<b>0.740</b>	<b>0.889</b>

	CSTN	MRSTN	GEML	STGCN	OD-CED
# of Params (in millions)	0.54M	0.67M	2.9M	1.6M	0.1M
Training Time (in seconds)	1222.13s	1602.14s	39.63s	28.81s	22.12s

# Application - Meso Level Supply-Demand Forecasting

## Causal Probabilistic Spatio-Temporal Fusion Transformers in Two-Sided Ride-Hailing Markets

**Goal:** Predicting supply and demand in ride-hailing platforms using a **causal, interpretable,** and **scalable** forecasting framework

**Algorithm 3:** A collaborative causal spatio-temporal fusion transformer (**CausalTrans**).

**Authors:** Wang et al.,

**Journal:** *ACM Transactions on Spatial Algorithms and Systems*, 2024

# Application - Meso Level Supply-Demand Forecasting

## Collaborative Problem

$$P(x_v(t+1:t+\tau_{\max})|x_v(:t), z_v(:t+\tau_{\max})) \\ P(y_v(t+1:t+\tau_{\max})|y_v(:t), x_v(:t+\tau_{\max}), z_v(:t+\tau_{\max}))$$

Where:

$x_v(t)$ : demand at time t in grid v;

$y_v(t)$ : supply at time t in grid v;

$z_v(t)$ : external covariates (e.g., weather, holiday) at time t in grid v;

$\tau_{\max}$  : pre-specified time length

$v \in V$ .

## Probabilistic Forecasting

- Given  $q \in Q = \{10\%, 50\%, 90\%\}$ , then quantile loss  $QL_q$  at each point q is:

$$QL_q(x_t, \hat{x}_{t-\tau}^q) = \{q - I(x_t \leq \hat{x}_{t-\tau}^q)\} * (x_t - \hat{x}_{t-\tau}^q)$$

- Then final quantile loss is:

$$Loss_Q = \sum_{x_t \in \Omega} \sum_{q \in Q} \sum_{\tau=1}^{\tau_{\max}} \frac{QL_q(x_t, \hat{x}_{t-\tau}^q)}{M * \tau_{\max}}$$

- We introduce quantile risk as a key metric:

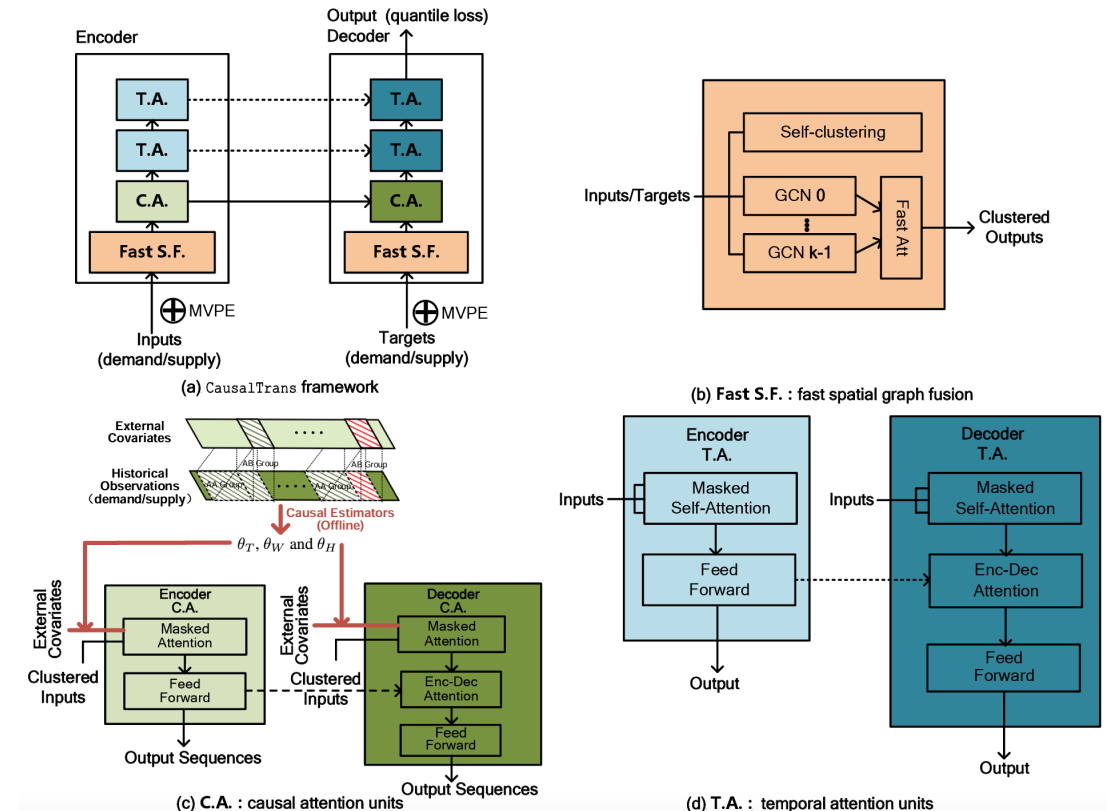
$$Risk_q = \frac{2 \sum_{x_t \in \tilde{\Omega}} \sum_{\tau=1}^{\tau_{\max}} QL_q(x_t, \hat{x}_{t-\tau}^q)}{\sum_{x_t \in \tilde{\Omega}} \sum_{\tau=1}^{\tau_{\max}} |x_t|}$$

where  $\tilde{\Omega}$  is the best dataset.

# Application - Meso Level Supply-Demand Forecasting

The overview of **CausalTrans** framework:

- (a). The framework consists of three essential components: **Fast S.F.** (*fast graph spatial fusion*), **C.A.** (*causal attention*), and **T.A.** (*temporal attention*). Demand and supply are trained separately in sequence.
- (b). The **Fast S.F.** consists of self-clustering with GAT and fast attention.
- (c). The **C.A.** applies offline trained causal weights  $\theta$  to online treatments evaluations.
- (d). The **T.A.** aims to keep ordering self-attentions.



# Application - Meso Level Supply-Demand Forecasting

## CausalTrans - Causal Attention Mechanism

We transfer the weights of external covariates to causal weights by **HTE methods** (e.g. double machine learning).

### Algorithm 1 Causal Attention Algorithm with DML

**Input:** Given demand matrix  $x(:t)$  at a grid  $v$  before time  $t$ , three kinds of treatments includes weekday and hour slots  $T(:t) = \{W(:t), H(:t)\}$ , weather vectors  $W(:t)$ , and holidays one-hot vectors  $H(:t)$

**Output:** causal effect coefficients  $\theta_T$  for  $T(:t)$ ,  $\theta_W$  for  $W(:t)$ , and  $\theta_H$  for  $H(:t)$

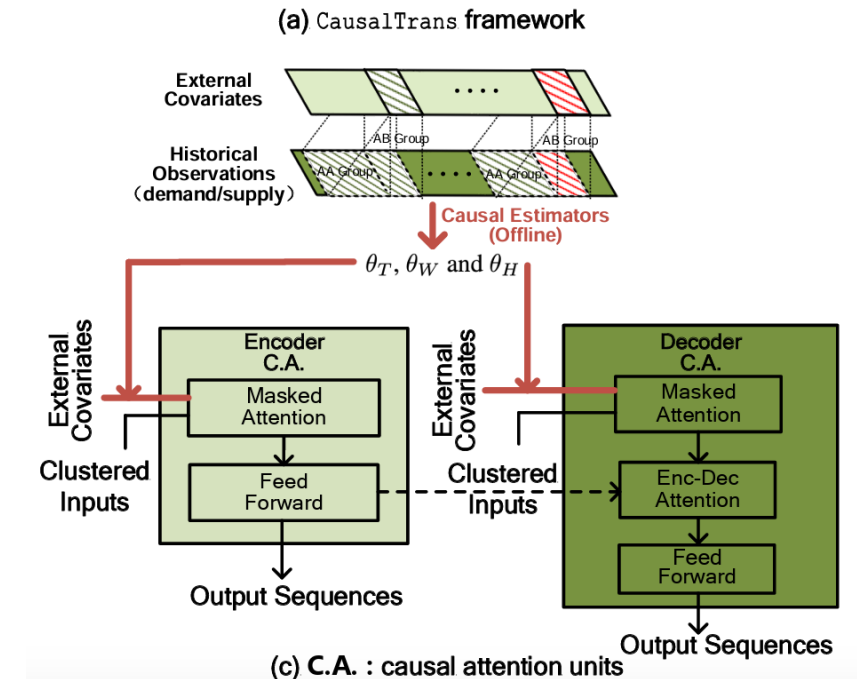
- 1: Take  $\theta_T$  as an example, and suppose that a AA group and AB group on  $T(:t)$  is  $T_{AA} = T_{AB} = \{\}$
- 2: **for all**  $\{T_w(t_0), T_w(t_1)\} \in \{Mon, Tue, \dots, Sun\}, \{T_h(t_0), T_h(t_1)\} \in \{1, \dots, 24\}$  **do**
- 3:   **if**  $T_w(t_0) = T_w(t_1), T_h(t_0) = T_h(t_1), \mathcal{P}_{T-Test}(x(t_0), x(t_1)) < 0.05$  **then**
- 4:     **for all**  $t'_0 \in \{t_0\}$  and  $t'_1 \in \{t_1\}$  **do**
- 5:       Calculate 1st-order differences  $\tilde{x}(t'_0 : t_0)$  and  $\tilde{x}(t'_1 : t_1)$
- 6:       **if**  $\mathcal{P}_{KPSS}(\tilde{x}(t'_0 : t_0)), \mathcal{P}_{KPSS}(\tilde{x}(t'_1 : t_1))$  and  $\mathcal{P}_{T-Test}(\tilde{x}(t'_0 : t_0), \tilde{x}(t'_1 : t_1)) > 0.05$  **then**
- 7:           $T_{AA}.append([x(t'_0 : t_0), x(t'_1 : t_1)])$
- 8:           $T_{AB}.append([x(t_0), x(t_1)])$
- 9:       **end if**
- 10:     **end for**
- 11:   **end if**
- 12: **end for**
- 13: Do DML on  $T_{AA}$  and  $T_{AB}$  datasets and estimate treatment coefficients  $\theta_T$
- 14: Repeat from Step 2 and estimate  $\theta_W$  and  $\theta_H$  by different DML.
- 15: **return**  $\theta_T, \theta_W$ , and  $\theta_H$

#### (a) causal attention algorithm

**step 1:** external covariates: weather, holidays and subsidy;

**step 2:** build various of control groups and treat groups;

**step 3:** do **DML** and get causal attention or weights.



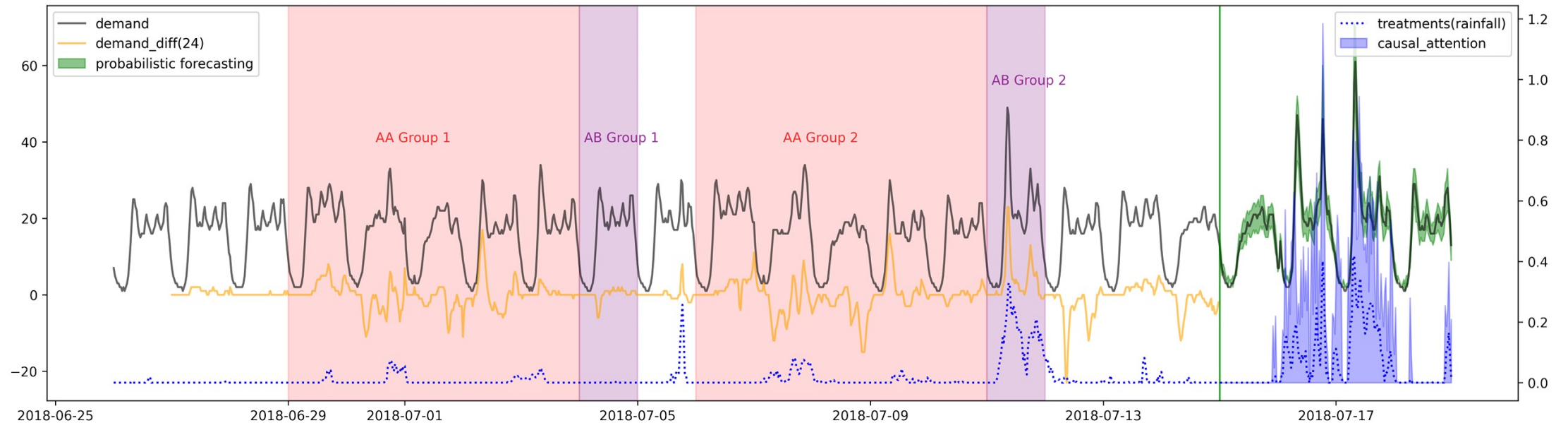
#### (b) how to work in ConvTrans

**step 1:** offline training causal attention;

**step 2:** add above weights in multi-head attention

# Application - Meso Level Supply-Demand Forecasting

## CausalTrans - Causal Attention Visualization



- “AA group 1” and “AA group 2” are regarded as comparable contexts;
- “AB group 1” and “AB group 2” is control group and treatment group;
- Do **DML** and get causal attention weights.

# Application - Meso Level Supply-Demand Forecasting

## CausalTrans - Experiment

(a) Risk\_(50%) losses on the retail and ride-hailing datasets.

	ConvTrans	Seq2Seq	MQRNN	DeepAR	DMVST	ST-MGCN	TFT	CausalTrans
Retail	0.429 <sup>◇</sup>	0.411 <sup>◇</sup>	0.379 <sup>◇</sup>	0.386	0.403	0.395	0.354 <sup>◇</sup>	<b>0.352(-0.6%)</b>
Ride-hailing (1d, city A, Demand)	0.573	0.550	0.495	0.499	0.524	0.482	0.450	<b>0.434(-3.7%)</b>
Ride-hailing (1d, city A, Supply)	0.482	0.453	0.428	0.422	0.443	0.421	0.415	<b>0.393(-5.3%)</b>
Ride-hailing (1d, city B, Demand)	0.470	0.455	0.405	0.400	0.422	0.404	0.370	<b>0.361(-2.5%)</b>
Ride-hailing (1d, city B, Supply)	0.426	0.404	0.388	0.384	0.388	0.378	0.357	<b>0.341(-4.5%)</b>
Ride-hailing (7d, city A, Demand)	0.756	0.717	0.653	0.663	0.664	0.677	0.689	<b>0.613(-6.2%)</b>
Ride-hailing (7d, city A, Supply)	0.612	0.569	0.516	0.519	0.536	0.575	0.583	<b>0.468(-9.3%)</b>
Ride-hailing (7d, city B, Demand)	0.693	0.627	0.574	0.571	0.590	0.588	0.576	<b>0.539(-5.6%)</b>
Ride-hailing (7d, city B, Supply)	0.568	0.519	0.499	0.501	0.503	0.525	0.528	<b>0.454(-9.0%)</b>

(b) Risk\_(90%) losses on the retail and ride-hailing datasets.

	ConvTrans	Seq2Seq	MQRNN	DeepAR	DMVST	ST-MGCN	TFT	CausalTrans
Retail	0.192 <sup>◇</sup>	0.157 <sup>◇</sup>	0.152 <sup>◇</sup>	0.156	0.156	0.155	0.147 <sup>◇</sup>	<b>0.143(-2.8%)</b>
Ride-hailing (1d, city A, Demand)	0.238	0.208	0.205	0.205	0.208	0.195	0.192	<b>0.164(-14.6%)</b>
Ride-hailing (1d, city A, Supply)	0.212	0.177	0.164	0.162	0.173	0.165	0.160	<b>0.142(-11.3%)</b>
Ride-hailing (1d, city B, Demand)	0.208	0.176	0.159	0.158	0.170	0.157	0.155	<b>0.145(-6.5%)</b>
Ride-hailing (1d, city B, Supply)	0.205	0.197	0.157	0.188	0.169	0.151	0.149	<b>0.139(-6.7%)</b>
Ride-hailing (7d, city A, Demand)	0.324	0.306	0.276	0.289	0.286	0.280	0.297	<b>0.244(-11.6%)</b>
Ride-hailing (7d, city A, Supply)	0.259	0.233	0.207	0.204	0.237	0.248	0.237	<b>0.173(-15.2%)</b>
Ride-hailing (7d, city B, Demand)	0.288	0.269	0.241	0.240	0.252	0.255	0.238	<b>0.216(-9.3%)</b>
Ride-hailing (7d, city B, Supply)	0.214	0.184	0.177	0.179	0.168	0.197	0.204	<b>0.153(-8.9%)</b>

- Use grid search to optimize hyperparameters;
- **DeepAR** outperforms **Seq2Seq** and **MQRNN** because of Poisson and weather covariates;
- **CausalTrans** outperforms other methods primarily due to causal estimator **DML**;
- **CausalTrans** achieves lower losses on supply than demand based on both causal relationship;
- Long-term prediction focuses on unbiased distribution estimation.

# Application - Climate & Weather

**Weather and climate data** capture atmospheric and oceanic conditions (e.g., temperature, wind, pressure, precipitation, air quality) via sensors at fixed or mobile locations.

Due to strong **spatiotemporal correlations** in climate data, **spatiotemporal deep modeling (STDM)** techniques are widely used for **short-term and long-term forecasting**.

## Deep Learning Approaches:

- **Air quality inference:** Predict urban air pollution.
- **Precipitation prediction:** Forecast rain using remote sensing.
- **Wind speed prediction:** Model anemometer readings.
- **Extreme weather detection:** Identify severe weather events.

## Data Types:

- **Spatial maps:** e.g., radar reflectivity images
- **Time series:** e.g., wind speed readings
- **Event data:** e.g., extreme weather occurrences

## Example Models:

- **Attention models** for air quality prediction
- **CNNs** for detecting extreme weather and precipitation forecasting

# Application - Neuroscience

Various brain imaging technologies — such as **fMRI**, **EEG**, **MEG**, and **fNIRS** — are widely used in neuroscience research. These technologies differ significantly in spatial and temporal resolution:

- **fMRI**: Millions of spatial locations, lower temporal resolution (~2 seconds per measurement)
- **EEG**: Tens of locations, very high temporal resolution (~1 millisecond)

## Data Representation:

- Brain imaging data (fMRI, EEG) are naturally represented as **spatial maps** or **rasters**, making them suitable for **DL** analysis.

## Deep Learning Applications:

- **Disease classification and diagnosis**: e.g., Autism Spectrum Disorder, amnesic Mild Cognitive Impairment, Schizophrenia
- **Brain function network classification**
- **Brain activation pattern classification**

## Example Models:

- **LSTM** for Autism Spectrum Disorder detection
- **CNN** for diagnosing amnesic Mild Cognitive Impairment
- **FNN** for classifying Schizophrenia

# Application 1: BrainGNN

BrainGNN is a **graph neural network (GNN)** specifically designed for analyzing **functional MRI (fMRI)** brain data. It **predicts cognitive states or disease status** while providing **interpretability** by identifying important brain regions and connections.

## Model Highlights:

### Node-Level Pooling:

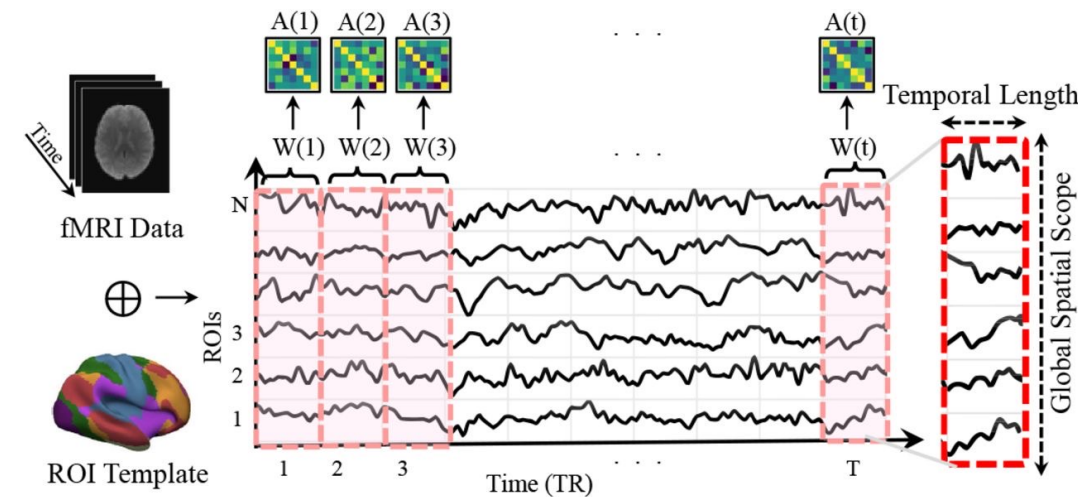
- Groups similar brain regions (ROIs) into clusters based on learned features.

### ROI Selection Layer:

- Automatically selects important brain regions contributing to the prediction.

### Attention Mechanism:

- Highlights key functional connections between selected ROIs.



Dynamic Graph Transformer for Brain Disorder Diagnosis

# Application

Introduce a model that **jointly learns spatial and temporal features** from resting-state fMRI (rs-fMRI) using a combination of **graph convolution** and **recurrent neural networks (RNNs)**.

## Model Highlights:

### Spatial Graph Convolution:

- Models **functional connectivity** between brain regions at each time step, treating brain ROIs as graph nodes.

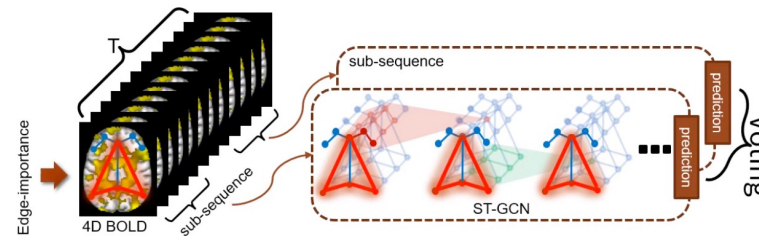
### Temporal RNN:

- Captures the **evolution of brain connectivity over time** by applying an RNN (such as GRU) on node embeddings.

### End-to-End Training:

- Learns both spatial (graph structure) and temporal (dynamic activity) representations directly from raw fMRI sequences.

- Our proposal: ST-GCN
  - › Consider both temporal dependency and functional connectivity
  - › Train on short sub-sequences
  - › Learns the importance of graph edges to the prediction



# Application

**Total Activation (TA)** is a method that **deconvolves fMRI signals** to recover the underlying **neural activity-inducing signals** by applying **spatio-temporal regularization**.

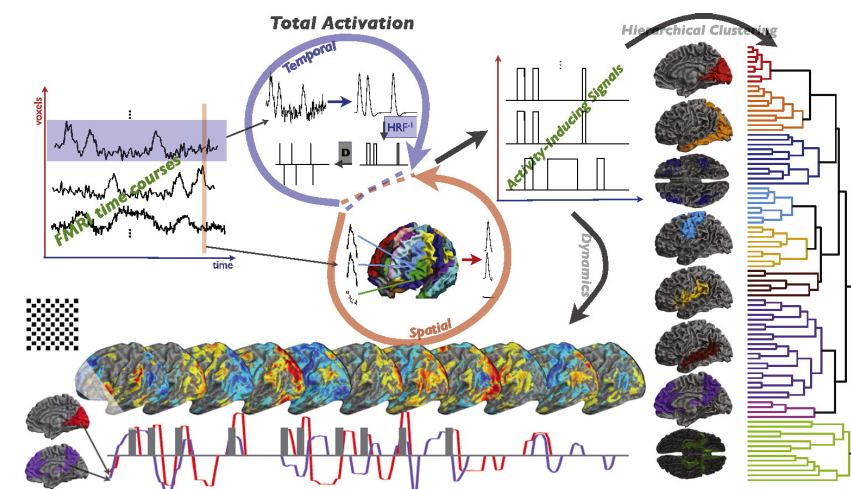
## Model Highlights:

### •Temporal Regularization:

- Promotes **piecewise constant activation patterns** over time (temporal sparsity).

### •Spatial Regularization:

- Enforces **spatial smoothness** across neighboring voxels (nearby brain regions).
- Solves an optimization problem balancing **data fidelity** with **spatio-temporal priors**.



Total activation: fMRI deconvolution through spatio-temporal regularization

# How to succeed in this course?

```
from torch import nn

# Create the neural network
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(-1, 16 * 5 * 5)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

**Practice**



**Explore**



**Visualize**

**Discuss**



**Ask**

