Sequence Modeling

Recurrent Neural Networks and LSTM

N. Rich Nguyen, PhD

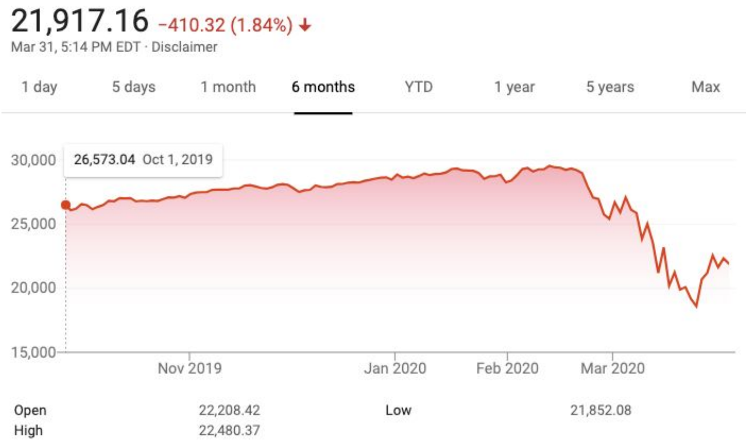
SYS 6016

Sequential Data

Text: "I am comfortably watching this lecture at home"

Audio:

Stock prices:



A Sequence Modeling Problem

"I am comfortably watching this lecture at home"

given these words **predict the next word**

We can use a fixed window:

"I am comfortably watching this lecture at home"

given these two words **predict the next word**

Problem 1: Long-term Dependencies

One-hot encoding tells us what each word is: [1 0 0 0 0 0 1 0 0 0] → prediction

lecture *at*

Using a short window (2 words), we can't model long-term dependencies:

"The coronavirus is originated in China, and …. We should not call it the ______"

not enough info!

We need information from **the distant past**...

Problem 2: Word Order in the Sequence

What if we use the entire sequence as a bag-of-words vector?

"I am comfortably watching this lecture at"

↓

"bag-of-words"

[0 1 0 0 1 0 1 1 ... 0 0 1 1 0 0 1 0 0 0]

However, bag of words does **not preserve the order** of words in the sequence.

The food was good, not bad at all.

vs.

The food was bad, not good at all.

Problem 3: Parameter Sharing

What if we use a very long fixed window:

"I am comfortably watching this lecture at home"

given these words **predict the next word**

[1 0 0 0 0 0 0 0 0 1... 0 1 0 0 0 0 0 0 1 0] → **prediction**

I am ... lecture at

Each of these inputs has separate parameters which *are not shared across the sequence*, so a neural network will likely to learn different parameters to "**I am**" if it appears at different part of the sequence:

[0 1 0 0 0 ...1 0 0 0 0 0 0 0 0 1 ...] → **prediction**?

I am ...

→ needs to share the same parameters across several time steps.

Sequence Modeling Criteria

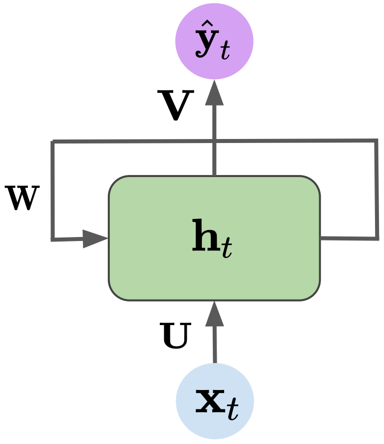
To model sequence, we will need to:

1. Handle variable-length sequence while tracking long-term dependencies
2. Maintain information about order
3. Share parameters across the sequence

Today, we will use **Recurrent Neural Networks (RNNs)** as an approach to *solve sequence modeling* problems.

Recurrent Neural Networks (RNNs)

Recurrent Neural Network



Want to predict $\mathbf{y}$ at some time step `t`

We can process a sequence of vector $\mathbf{x}$ by applying a **recurrence** formula at every time step:

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

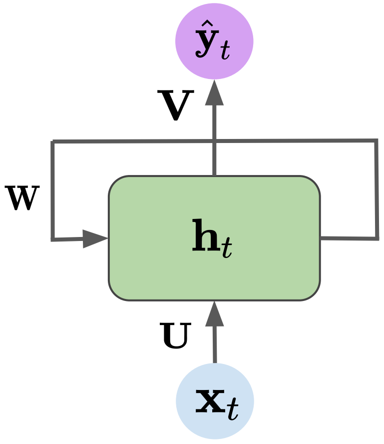
New state

A function with parameters W

Old state

Input vector at this time step

Recurrent Neural Network



Note that **the same** function and set of parameter are used at every time step.

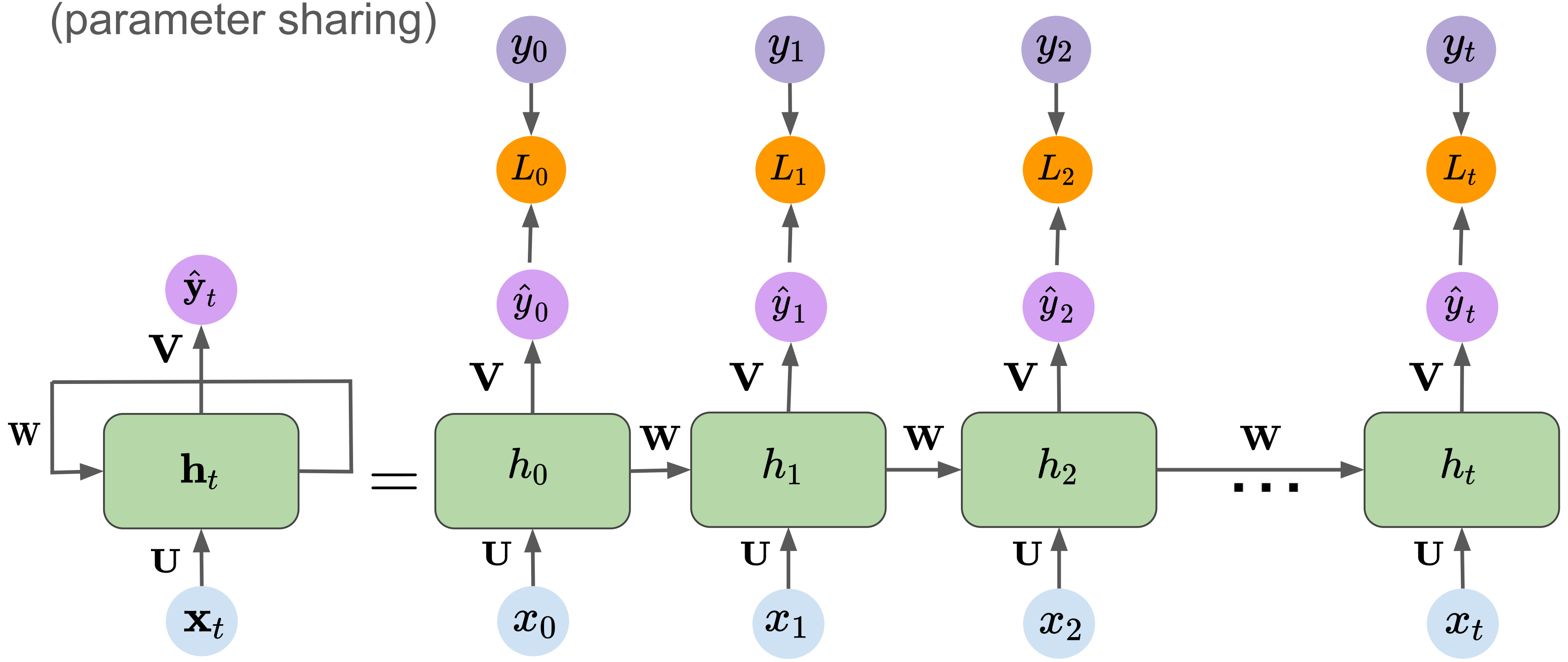
The state consists of a single "hidden" vector $\mathbf{h}_t$:

$$\mathbf{h}_t = \tanh(\mathbf{W}\mathbf{h}_{t-1} + \mathbf{U}\mathbf{x}_t)$$

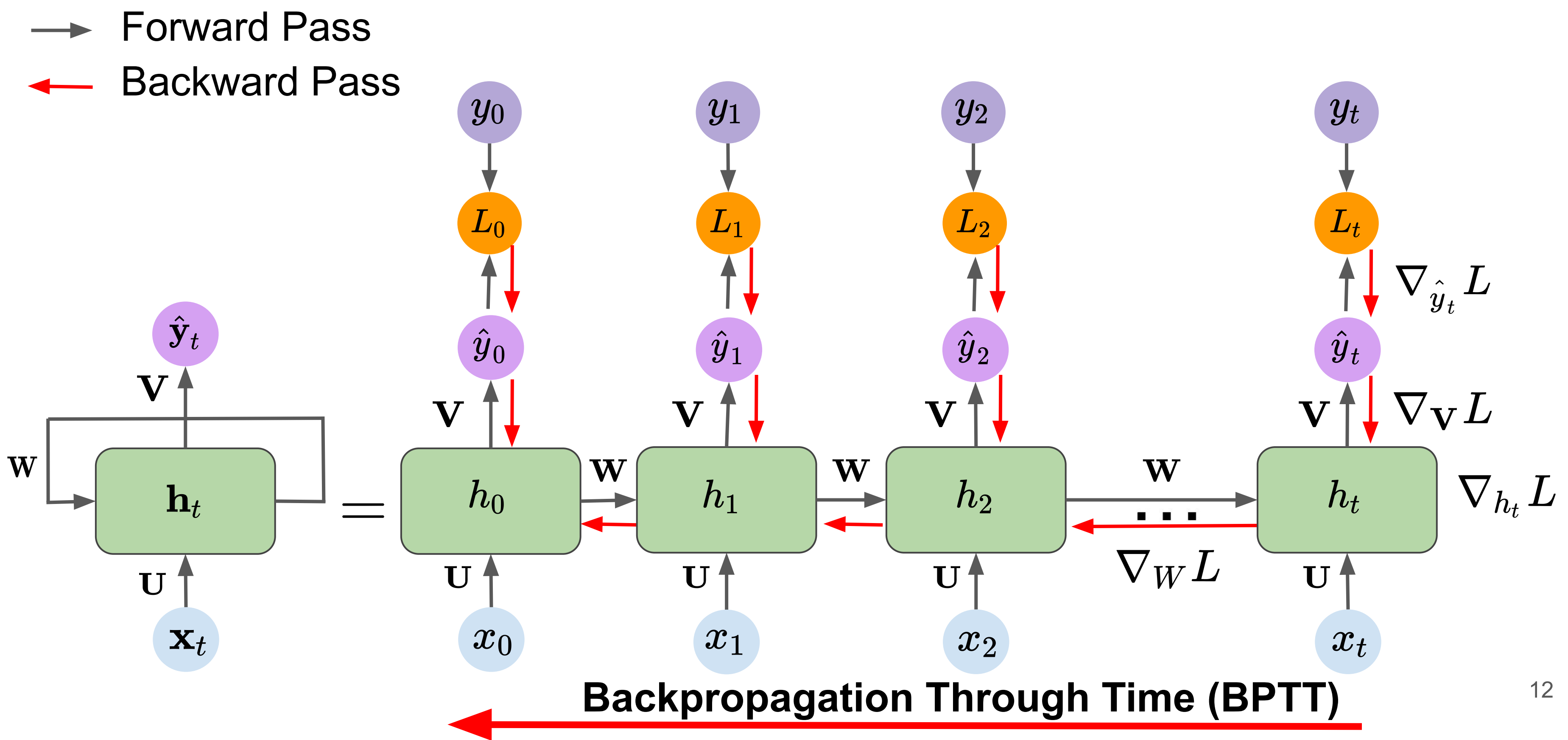
$$\hat{\mathbf{y}}_t = \mathbf{V}\mathbf{h}_t$$

Recurrent Computational Graph

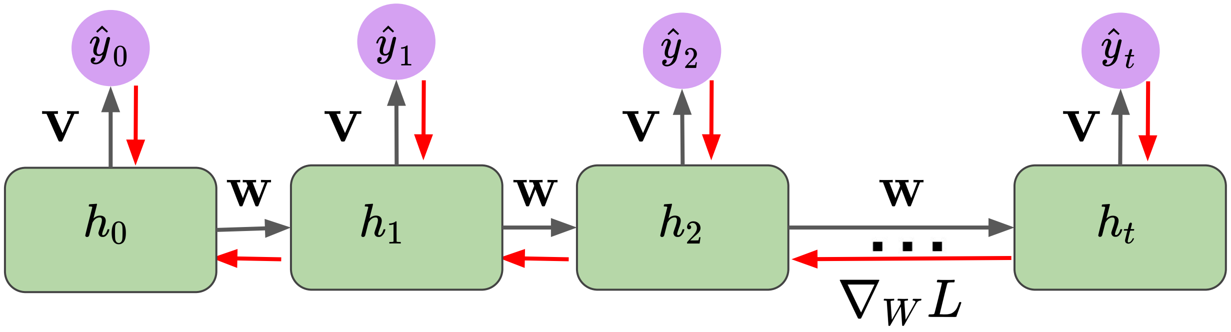
Unfolding the network and **reuse the same weight matrix** at every timestep (parameter sharing)



Training RNNs



Training RNN: Vanishing/exploding gradients



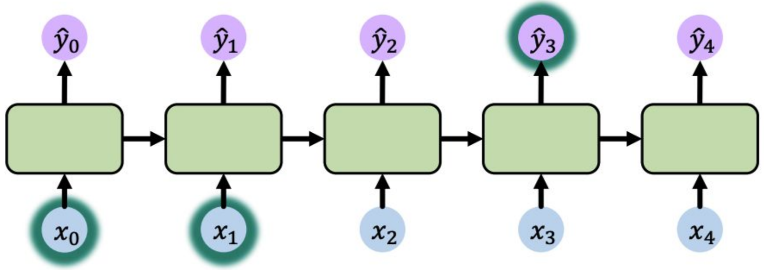
Computing the gradient w.r.t h_0 involves many factors of $\mathbf{W}$ with repeated gradient computation which can be problematics:

- Many values > 1: **exploding** gradients (rarely happen, but damaging) → needs to scale big gradients using gradient clipping?
- Many values < 1: **vanishing** gradients (happen most of the time) → only capture short-term dependencies

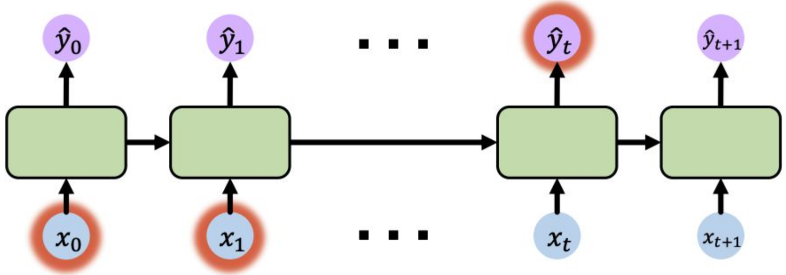
Problem of Training RNN over long sequence

- A standard RNN can only capture short-term dependencies.
- Training a RNN on long sequences suffer from the **vanishing gradients** problem
- ⇒ further back time steps have smaller and smaller gradients
- ⇒ this training produces only parameters that can predict short-term dependencies.

“Watching lectures at home is ______”



“COVID-19 is originated from China, and ... , we should not call it the ______”



Control the flow of gradients

Although RNN should theoretically be able to retain information about inputs seens many timesteps before, in practice, such long-term dependencies are very **difficult to learn**[2]

A solution is to use a more complex recurrent unit with gates to control how the information is passed through

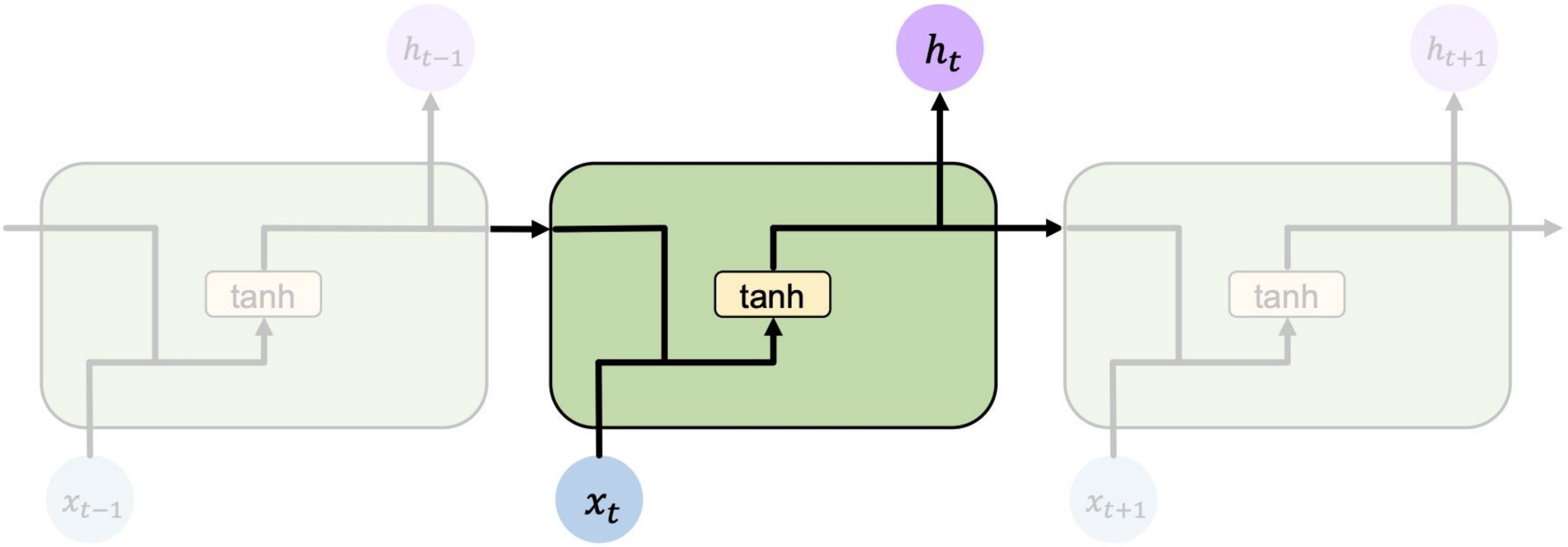
⇒ **Long Short-term Memory** (LSTM)

[2]See, for example, Yoshua Bengio, Patrice Simard, and Paolo Frasconi, "Learning Long-Term Dependencies with Gradient Descent Is Difficult," *IEEE Transactions on Neural Networks* 5, no. 2 (1994).

Long Short-term Memory (LSTM)

Structure of a Standard RNN

In a standard RNN, repeating modules contain a simple computation node

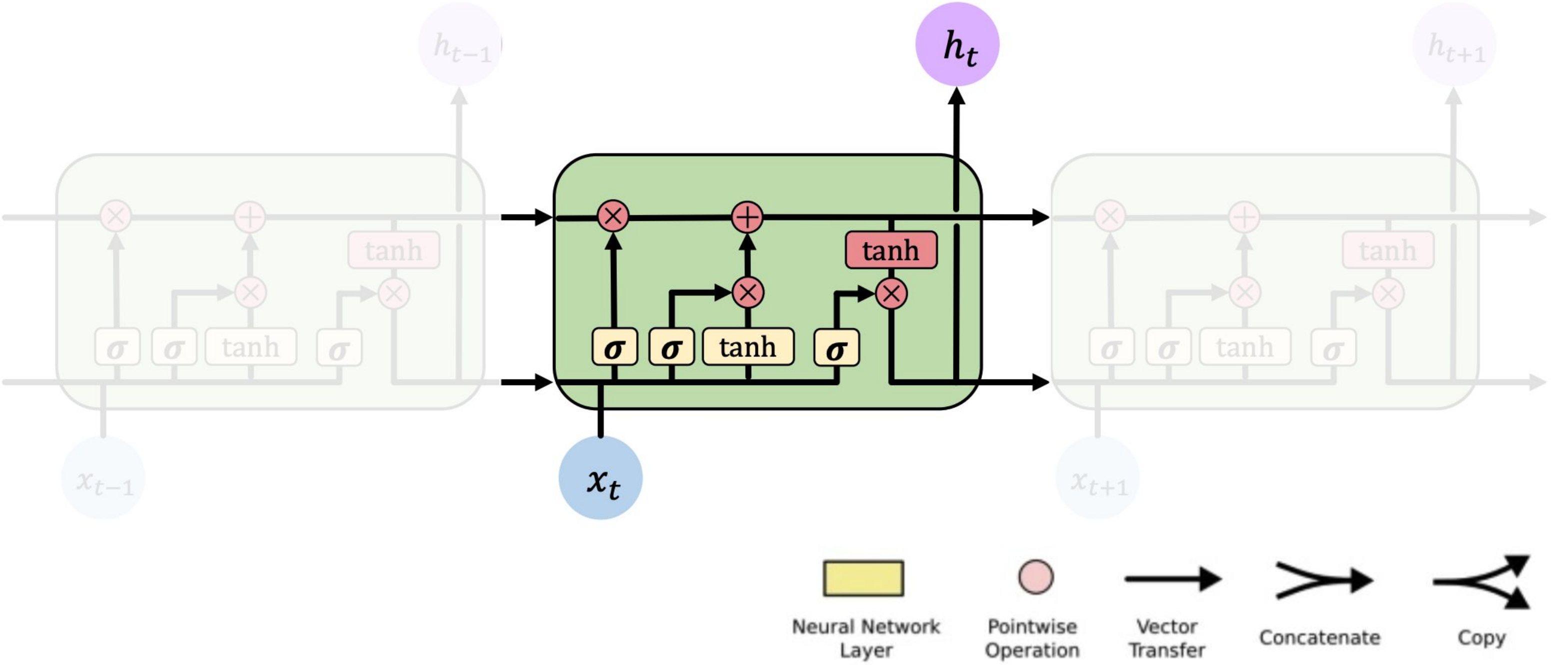


$$\mathbf{h}_t = \tanh(\mathbf{W}\mathbf{h}_{t-1} + \mathbf{U}\mathbf{x}_t)$$

Long Short-Term Memory (LSTM)

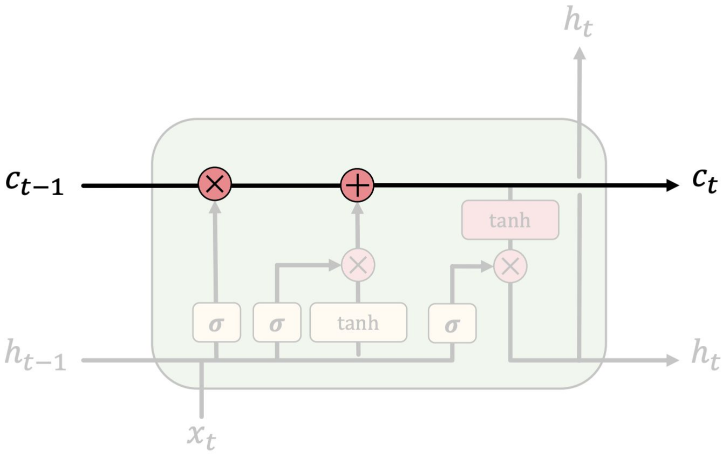
Proposed by **Hochreiter** and **Schmidhuber** in 1997

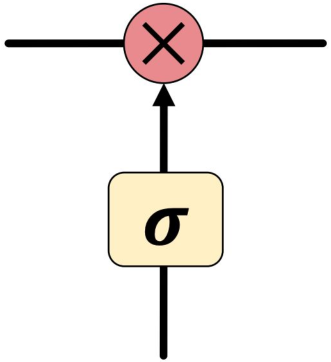
LSTM repeating modules contain interacting layers that control information flow



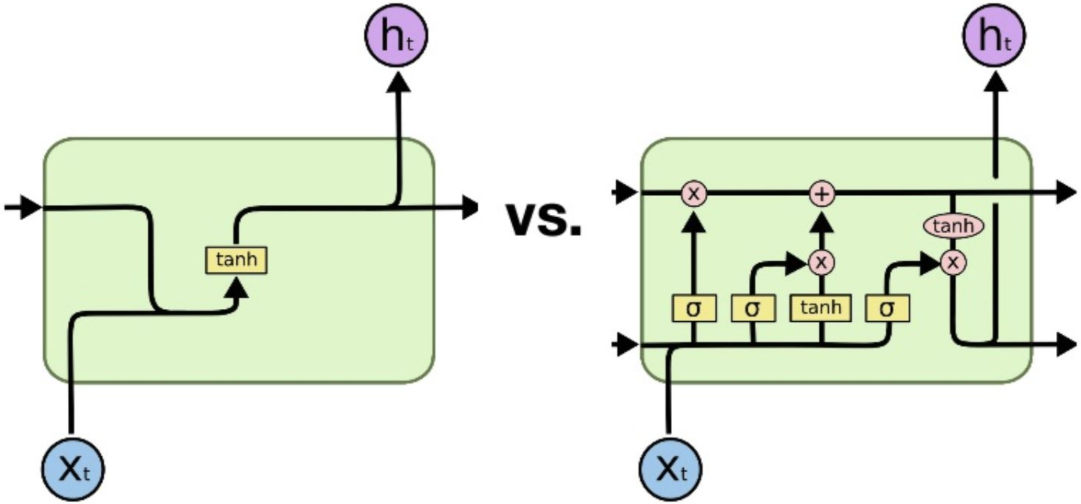
LSTM Structure

- Introduce **self-loops** to produce paths where the gradient can flow for long duration and make the weight of this self-loop **conditioned** on the context.
- LSTM maintains **a cell state** c_t where it's easy for information flow
- Information is added or removed to cell state through **gates**
- Gates let information through via a **sigmoid** neural layer and pointwise multiplication





LSTM Core Idea



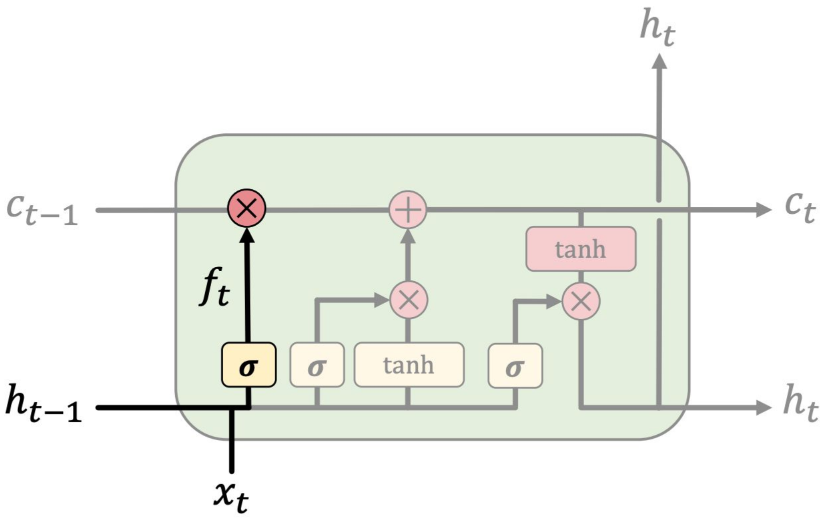
An LSTM cell learns to:

1. **Forget** irrelevant parts of the previous state
2. **Store** relevant parts in a long-term cell state
3. **Update** selectively its cell state
4. **Output** its cell state whenever needed

Example: Bob and Alice are having lunch. Bob likes apples. Alice likes oranges. <u>She</u> is eating an orange.

1. Forget Gate: Forget irrelevant information

Bob and Alice are having lunch. Bob likes apples. Alice likes oranges. **She** is eating an orange.

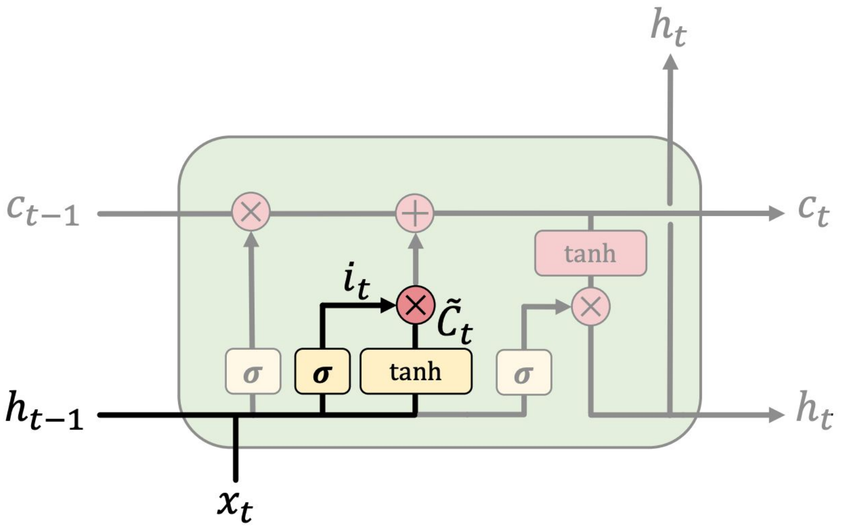


$$\mathbf{f}_t = \sigma(\mathbf{W}_f(\mathbf{h}_{t-1}, \mathbf{x}_t))$$

- Use previous cell output and input
- **Sigmoid** layer: values either 0 or 1 → "completely forget" or "completely keep"

Example: Forget the gender pronoun of previous subject (Bob)

2. Store Gate: Identify new info to be stored



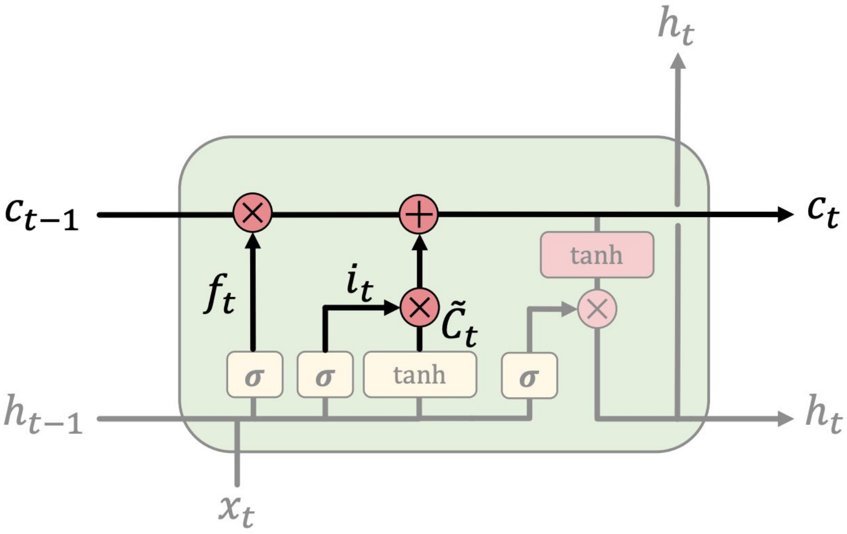
$$\mathbf{i}_t = \sigma(\mathbf{W}_i(\mathbf{h}_{t-1}, \mathbf{x}_t))$$

$$\tilde{\mathbf{C}}_t = \tanh(\mathbf{W}_C(\mathbf{h}_{t-1}, \mathbf{x}_t))$$

- **Sigmoid**: to decide what values to update
- **Tanh**: to generate new vector of values to be added to the cell state.

Example: Add gender of new subject (Alice) to replace that of old subject (Bob).

3. Update Gate: Update cell state

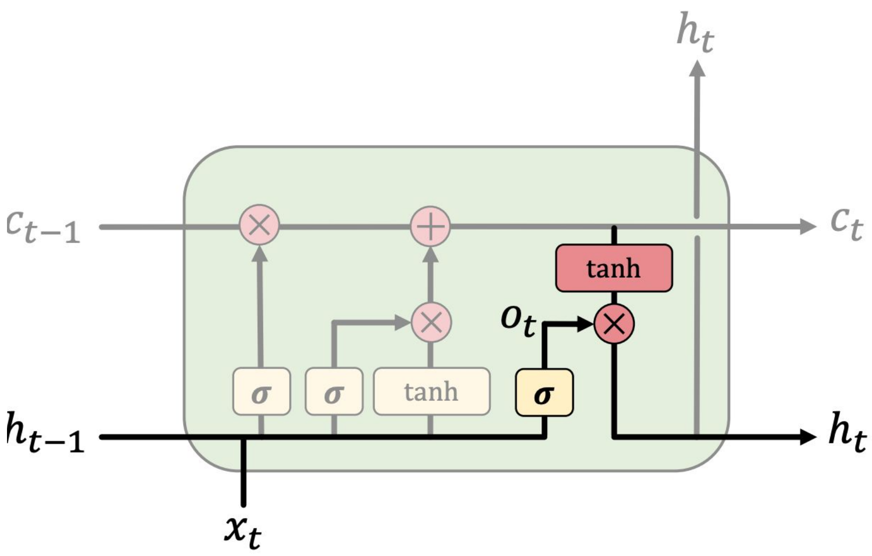


$$\mathbf{C}_t = \mathbf{f}_t * \mathbf{C}_{t-1} + \mathbf{i}_t * \tilde{\mathbf{C}}_t$$

- Apply **forget** operation to previous internal cell state
- Then, **add** new values, scaled by how much we decided to update

Example: *Actually* drop old info and add **new info** about subject's **gender**.

4. Output Gate: Output filtered cell state



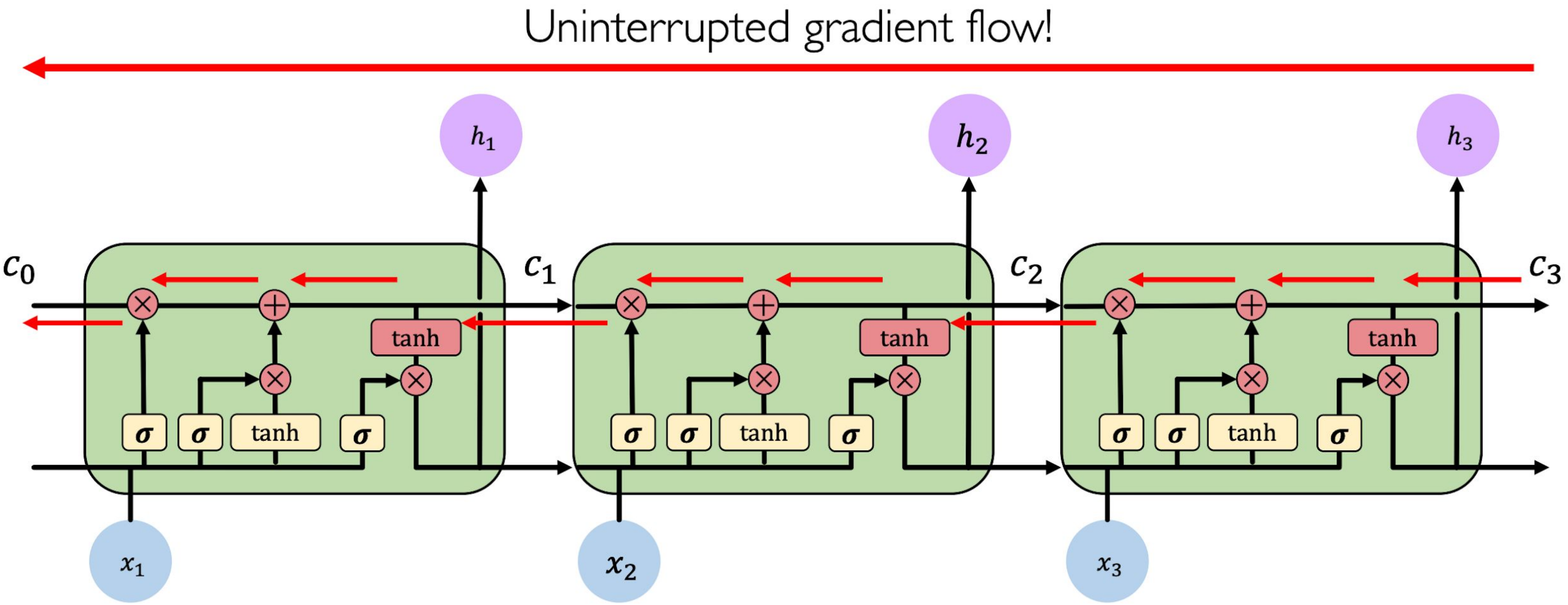
$$\mathbf{o}_t = \sigma(\mathbf{W}_o(\mathbf{h}_{t-1}, \mathbf{x}_t))$$

$$\mathbf{h}_t = \mathbf{o}_t * \tanh(\mathbf{C}_t)$$

- **Sigmoid**: to decide what parts of cell state to output
- **Tanh**: squashes values between -1 and 1
- **Multiplication**: output filtered version of cell state

Example: Having seen a subject (Alice), may output info relating to a pronoun (**She**).

LSTM Gradient Flow



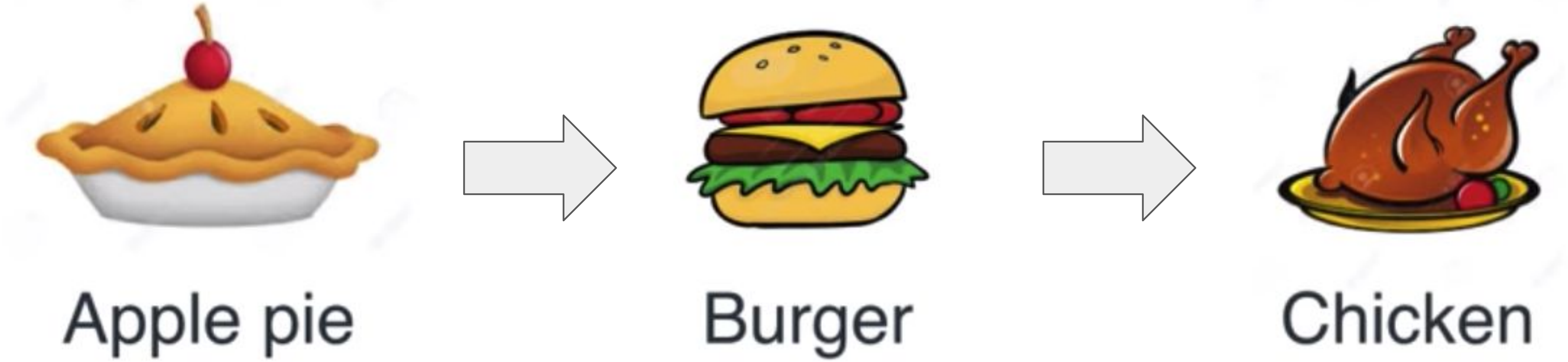
$$\mathbf{C}_t = \mathbf{f}_t * \mathbf{C}_{t-1} + \mathbf{i}_t * \tilde{\mathbf{C}}_t$$

Summary

- **RNNs** is used to solving sequence modeling problem
- **LSTM** maintains a separate cell state and uses gates to control the information flow: forget, store, update, and output gates.
- Backpropagation can be done with **uninterrupted** gradient flow
- Better/simpler architectures are a **hot** topic of research → the rise of **BERT** in 2018 (will be discussed in the next video)

Bonus Slides

Toy Example: A Moody Chef



Moods based on the weather

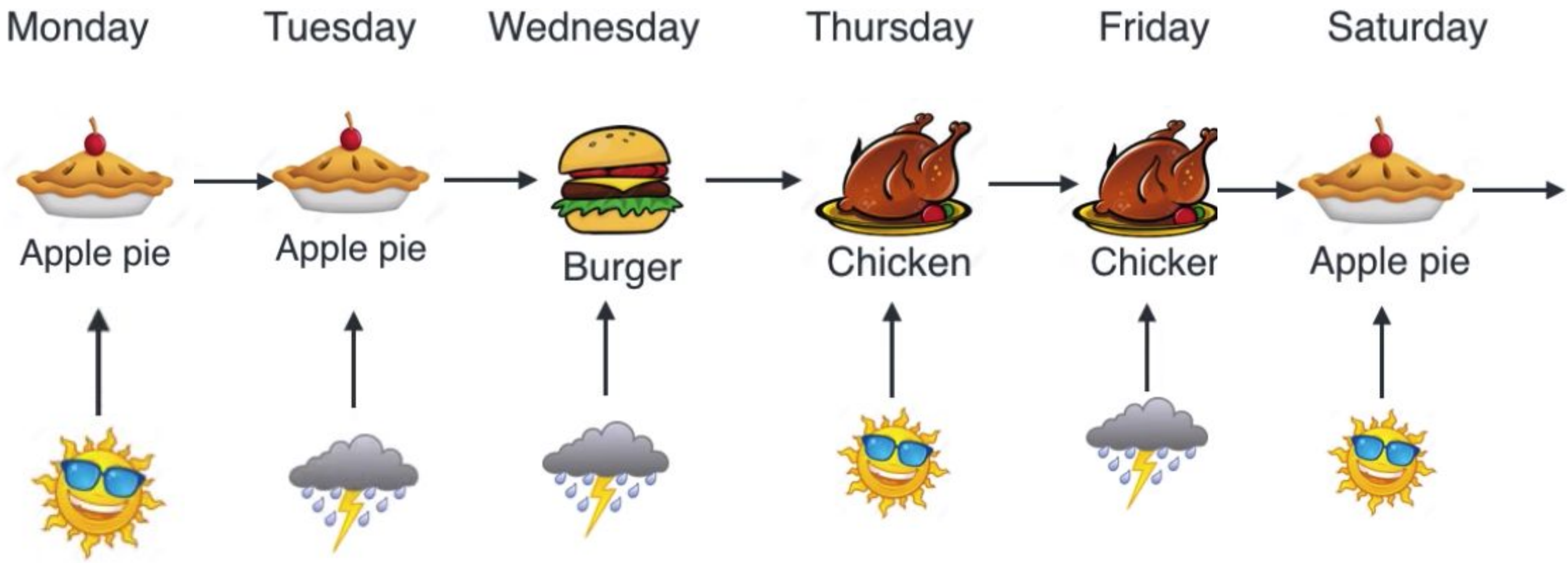
Sunny

Same as yesterday

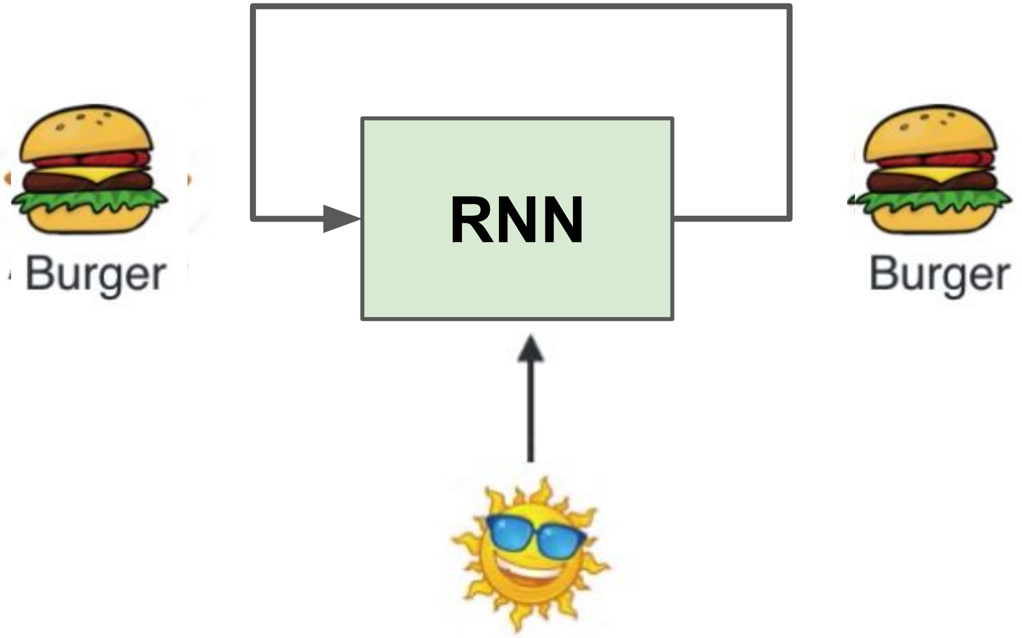
Rain

Next dish

Cooking Schedule

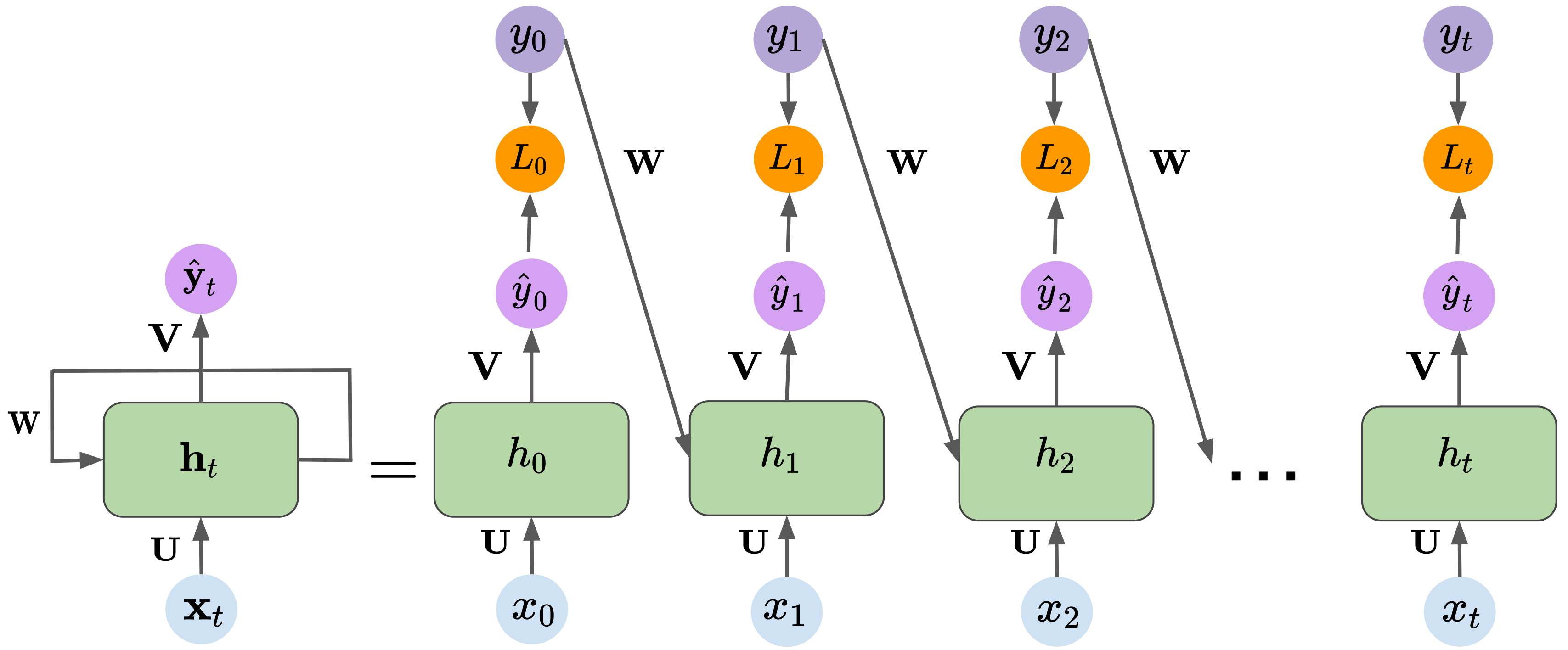


Recurrent Neural Network

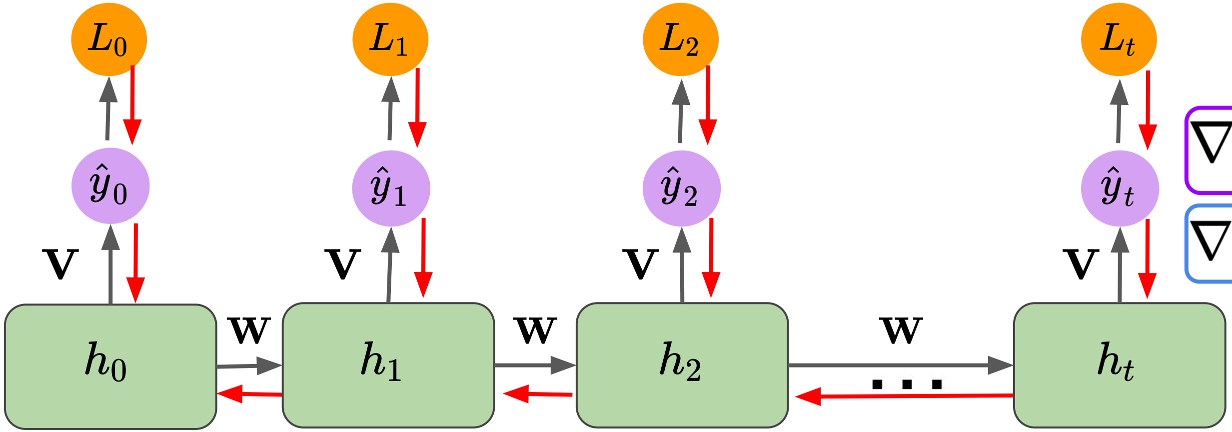


Teacher Forcing Networks

RNN has recurrent connections from **outputs** leading back into **hidden states**



Exploding/vanishing gradients



$$\boxed{\nabla_{\hat{y}_t} L} = \frac{\partial L}{\partial \hat{y}_t} = \hat{y}_t - y_t$$

$$\boxed{\nabla_{\mathbf{V}} L} = \sum_t (\boxed{\nabla_{\hat{y}_t} L}) h_t^\top$$

$$\boxed{\nabla_{h_t} L} = \mathbf{V}^\top \boxed{\nabla_{\hat{y}_t} L}$$

$$\boxed{\nabla_W L} = \sum_t \boxed{\mathrm{diag}(1 - h_t^2)} (\boxed{\nabla_{h_t} L}) h_{t-1}^\top$$

the Jacobian of the tanh associated with the hidden unit at time t