# Recurrent Neural Networks (RNNs) Improvements

Naeemullah Khan

naeemullah.khan@kaust.edu.sa

KAUST Academy

King Abdullah University of Science and Technology

# Table of Contents

# Motivation



**Why Do We Need to Improve RNNs?**

- Vanilla RNNs **fail at capturing long-term dependencies**
- Gradients either **vanish or explode** over long sequences
- This **limits learning** over time-based tasks like translation, conversation modeling, or video understanding

**Solution:** Modify RNN architecture to retain important past information without instability.

# Learning Outcomes

KAUST ACADEMY

By the end of this session, you should be able to:

- Explain why RNNs suffer from vanishing and exploding gradients
- Understand how LSTMs and GRUs solve these problems
- Compare LSTMs and GRUs in terms of performance and complexity
- Identify practical scenarios where each is preferred
- Recognize limitations and future directions in sequence modeling

# Vanishing and Exploding Gradients

**Backpropagation Through Time (BPTT)** spreads gradients across many time steps.

**Vanishing Gradients:**

$$\left\| \frac{\partial L}{\partial h_t} \right\| \to 0$$

- Early layers barely learn
- Forget long-term dependencies

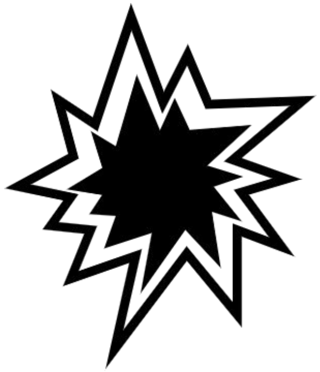
**Exploding Gradients:**

$$\left\| \frac{\partial L}{\partial h_t} \right\| \to \infty$$

- Unstable updates, diverging weights

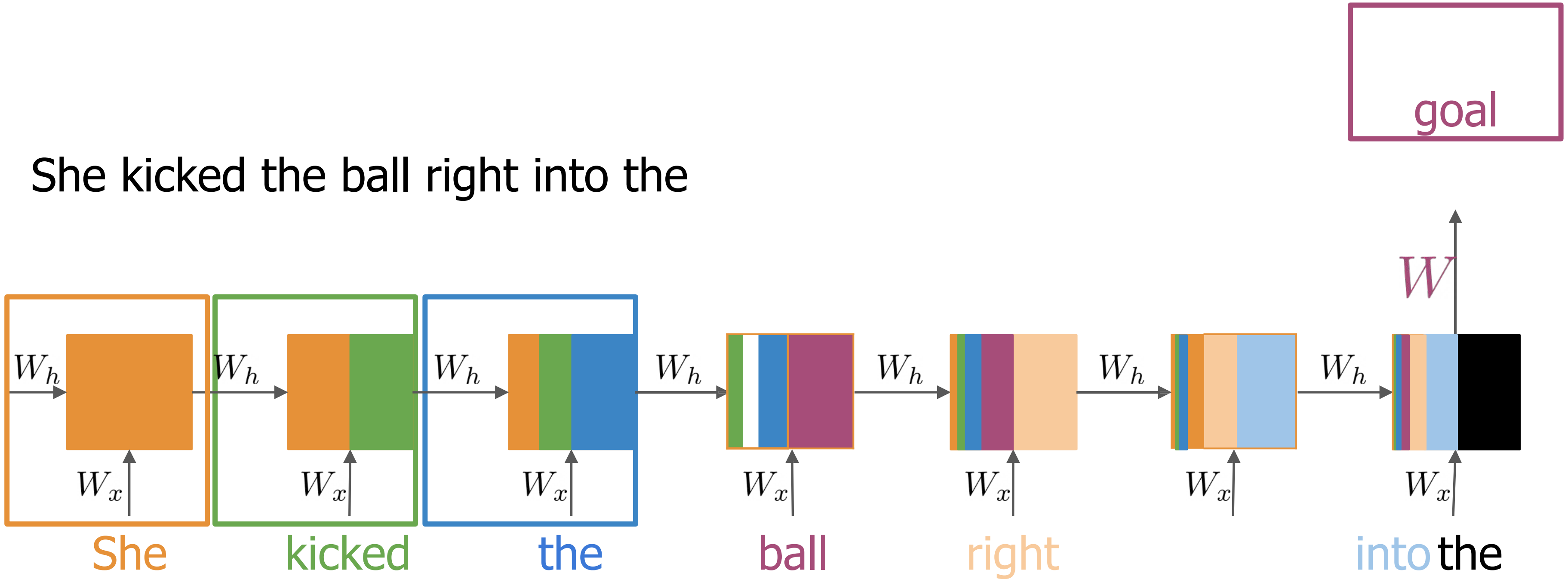
# RNNs and Vanishing Gradients

- Backprop through time
- RNNs and vanishing/exploding gradients
- Solutions



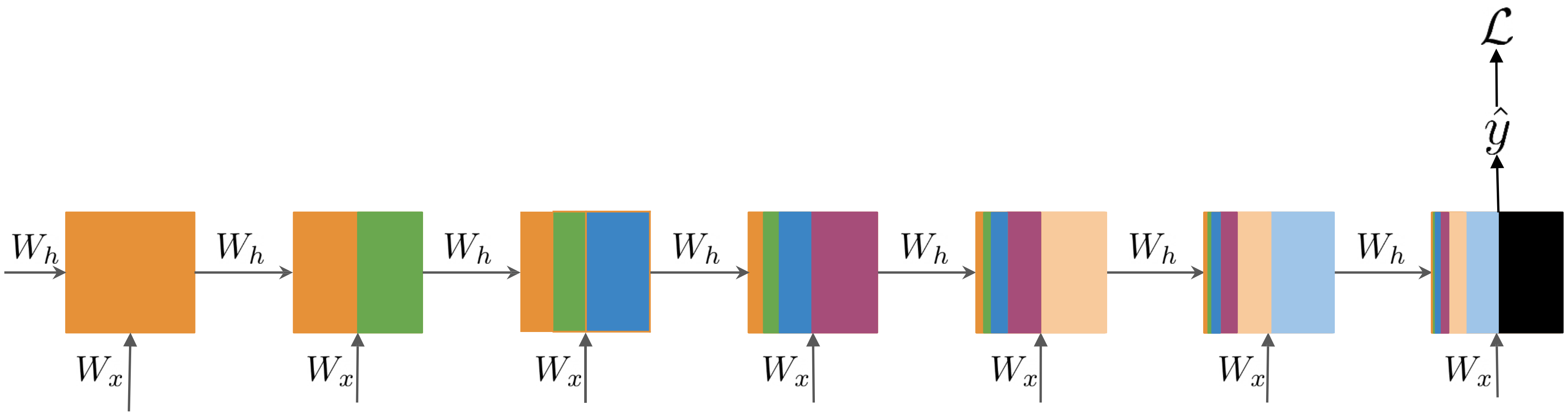
# RNN Basic Structure

She kicked the ball right into the




Learnable
parameters

# Backpropagation through time




$W_x$, $W_h$ → Same at every step

$$\frac{\partial L}{\partial W_h} \propto \sum_{1 \leq k \leq t} \left( \prod_{t \geq i > k} \frac{\partial h_i}{\partial h_{i-1}} \right) \frac{\partial h_k}{\partial W_h}$$

Gradient is proportional to a sum of partial derivative products

# Backpropagation through time

$$\frac{\partial L}{\partial W_h} \propto \sum_{1\leq k\leq t} \boxed{\left(\boxed{\prod_{t\geq i>k} \frac{\partial h_i}{\partial h_{i-1}}}\right) \frac{\partial h_k}{\partial W_h}}$$

Contribution of hidden state ***k***

Length of the product proportional to how far ***k*** is from ***t***

$$\boxed{\frac{\partial h_t}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial h_{t-2}} \frac{\partial h_{t-2}}{\partial h_{t-3}} \frac{\partial h_{t-3}}{\partial h_{t-4}} \frac{\partial h_{t-4}}{\partial h_{t-5}} \frac{\partial h_{t-5}}{\partial h_{t-6}} \frac{\partial h_{t-6}}{\partial h_{t-7}} \frac{\partial h_{t-7}}{\partial h_{t-8}} \frac{\partial h_{t-8}}{\partial h_{t-9}} \frac{\partial h_{t-9}}{\partial h_{t-10}} \frac{\partial h_{t-10}}{\partial W_h}}$$

Contribution of hidden state ***t-10***

# Backpropagation through time

أكاديمية كاوست
KAUST ACADEMY

$$\frac{\partial L}{\partial W_h} \propto \sum_{1\le k\le t} \boxed{\left(\boxed{\prod_{t\ge i>k} \frac{\partial h_i}{\partial h_{i-1}}}\right)\frac{\partial h_k}{\partial W_h}}$$

Contribution of hidden state ***k***

Length of the product proportional to how far ***k*** is from ***t***

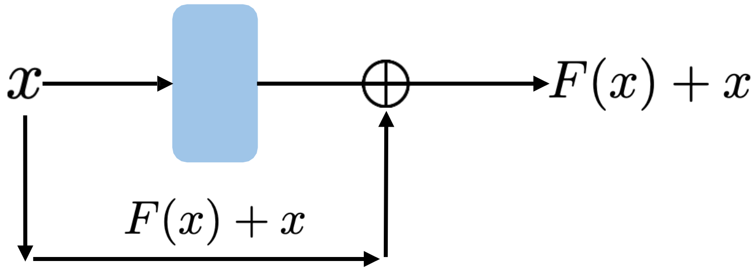
| | | |
|---|---|---|
| Partial derivatives <1 | Contribution goes to 0 | Vanishing Gradient |
| Partial derivatives >1 | Contribution goes to infinity | **Exploding** Gradient |

# Solving for vanishing or exploding gradients

- Identity RNN with ReLU activation

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Gradient clipping
- Skip connections



# RNNs: Advantages



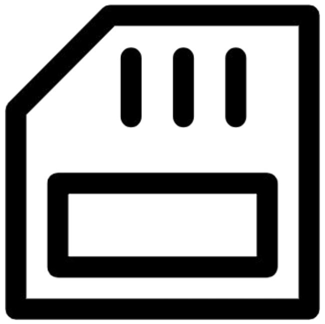
- Captures dependencies within a short range
- Takes up less RAM than other n-gram models

# RNNs: Disadvantages

- Struggles to capture long term dependencies
- Prone to vanishing or exploding gradients

# LSTMs

- Meet the Long short-term memory unit!
- LSTM architecture
- Applications



# LSTM – Long Short-Term Memory

**Introduced by Hochreiter & Schmidhuber (1997)**

**Core Idea:** LSTM uses **gates** to control what to keep, forget, and output.

**Key Components:**

- **Forget Gate** $f_t$
- **Input Gate** $i_t$
- **Cell State** $C_t$
- **Output Gate** $o_t$

**Equations:**

$$
\begin{aligned}
f_t &= \sigma(W_f[x_t, h_{t-1}] + b_f) \\
i_t &= \sigma(W_i[x_t, h_{t-1}] + b_i) \\
\tilde{C}_t &= \tanh(W_C[x_t, h_{t-1}] + b_C) \\
C_t &= f_t * C_{t-1} + i_t * \tilde{C}_t \\
o_t &= \sigma(W_o[x_t, h_{t-1}] + b_o) \\
h_t &= o_t * \tanh(C_t)
\end{aligned}
$$

**Helps retain long-term dependencies**

# LSTMs: a memorable solution

- Learns when to remember and when to forget
- Basic anatomy:
  - A cell state
  - A hidden state
  - Multiple gates
- Gates allow gradients to avoid vanishing and exploding

# LSTMs: Based on previous understanding



Starting point with some irrelevant information

↓

Cell and Hidden States

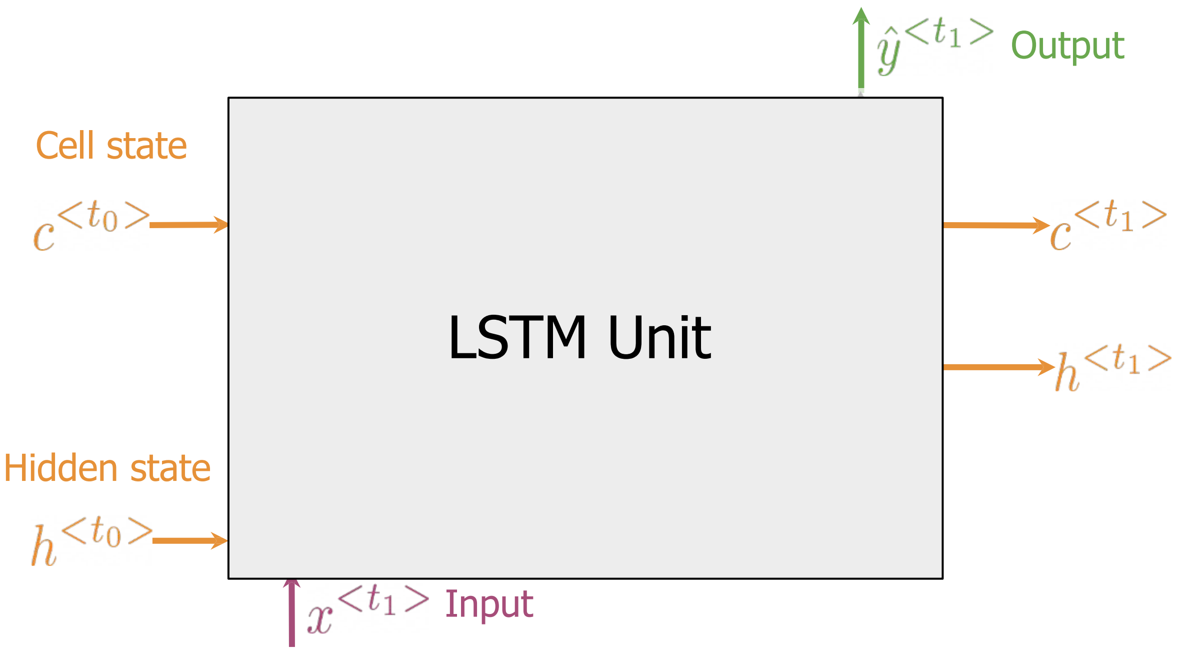
Discard anything irrelevant

Add important new information

Produce output

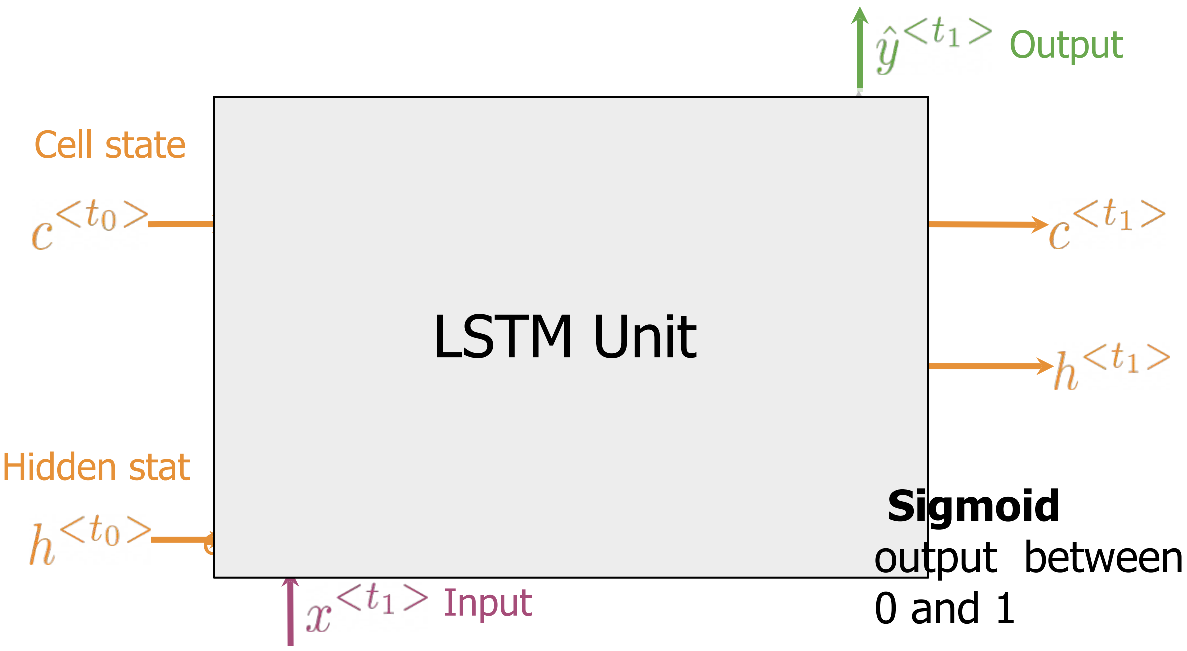
→ Gates

# Gates in LSTM




1. **Forget Gate**: information that is no longer important
2. **Input Gate**: information to be stored
3. **Output Gate**: information to use at current step

# Gates in LSTM

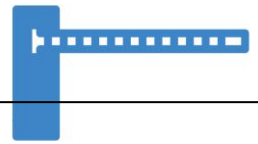



**Sigmoid**
output between
0 and 1

**1. Forget Gate**:
information that is no longer important

**2. Input Gate**:
information
to be stored

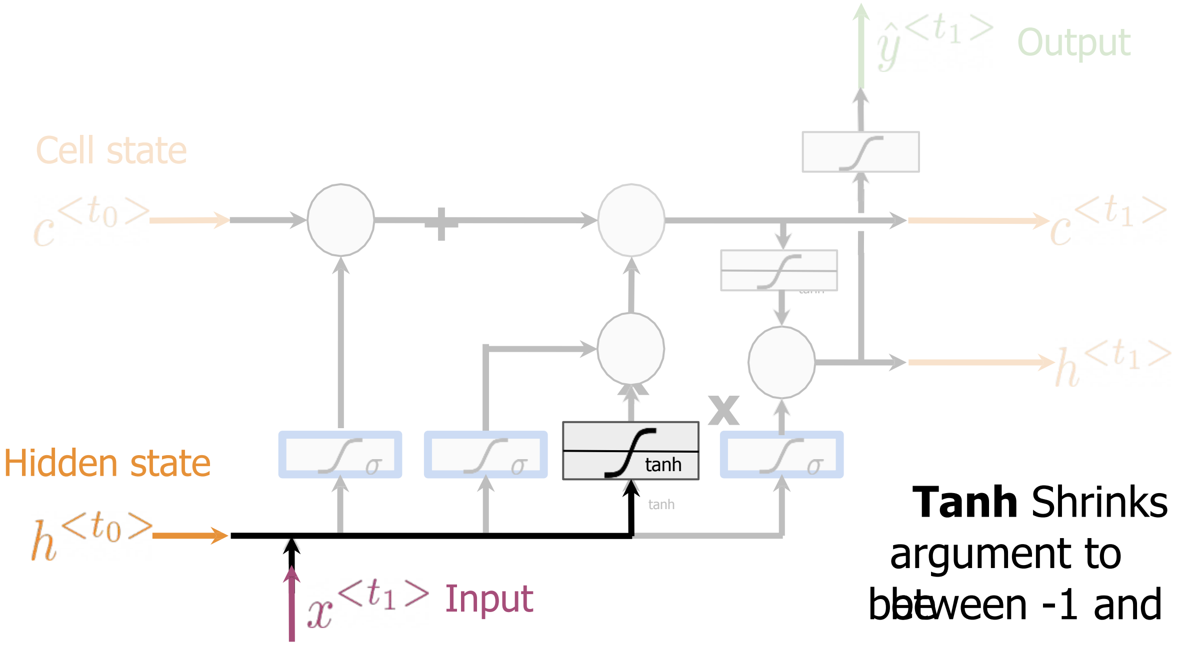
**3. Output Gate**:
information to use at current step



0-Closed

1-Open

# Candidate Cell State

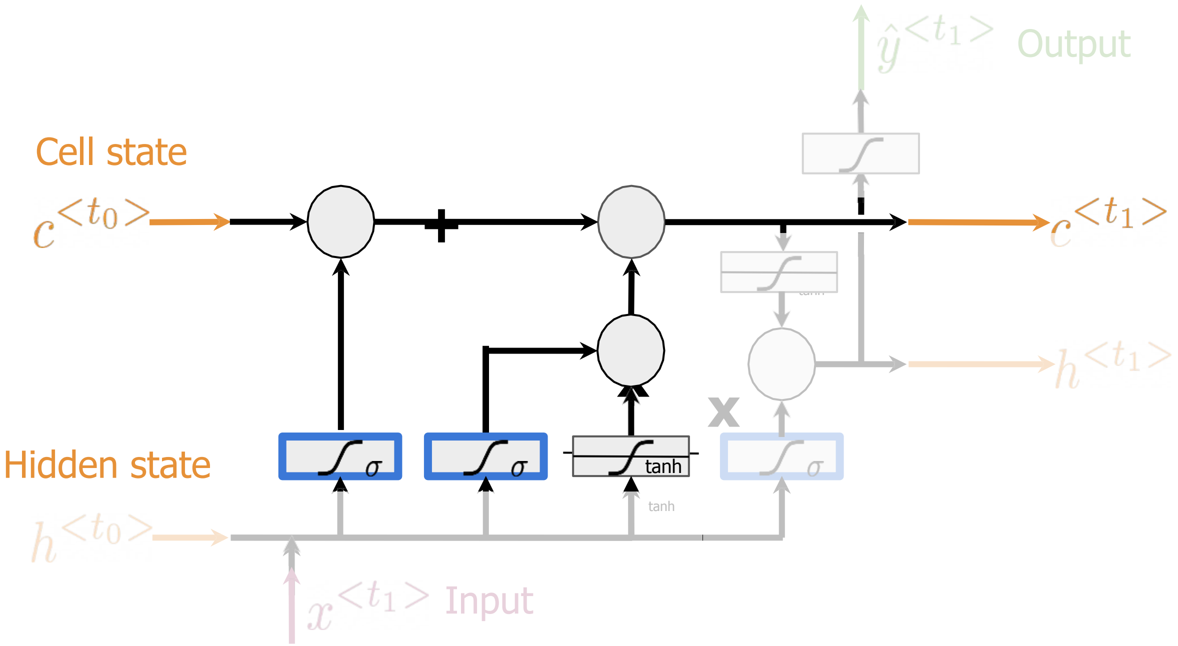



**Candidate cell state**

Information from the previous **hidden state** and current **input**

**Tanh** Shrinks argument to between -1 and 1 → **Other activations could be used**

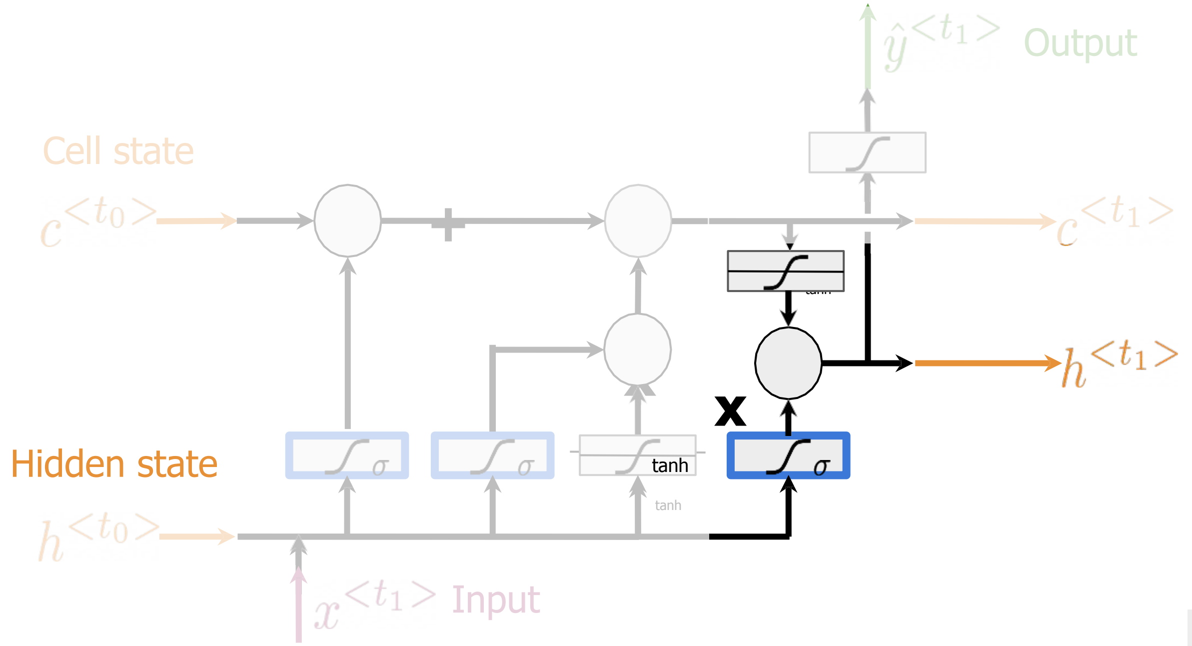
# New Cell State




**New Cell state**

Add information from the **candidate cell state** using the **forget** and **input gates**

# New Hidden State




**New Hidden State**

Select information from the **new cell state** using the **output gate**

The **Tanh** activation could be omitted

# Applications of LSTMs

Next-character prediction

Chatbots

Music composition

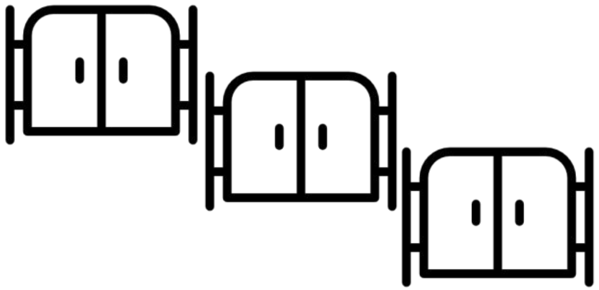
Image captioning

Speech recognition

# Summary

- LSTMs offer a solution to vanishing gradients

- Typical LSTMs have a cell and three gates:
  - Forget gate
  - Input gate
  - Output gate

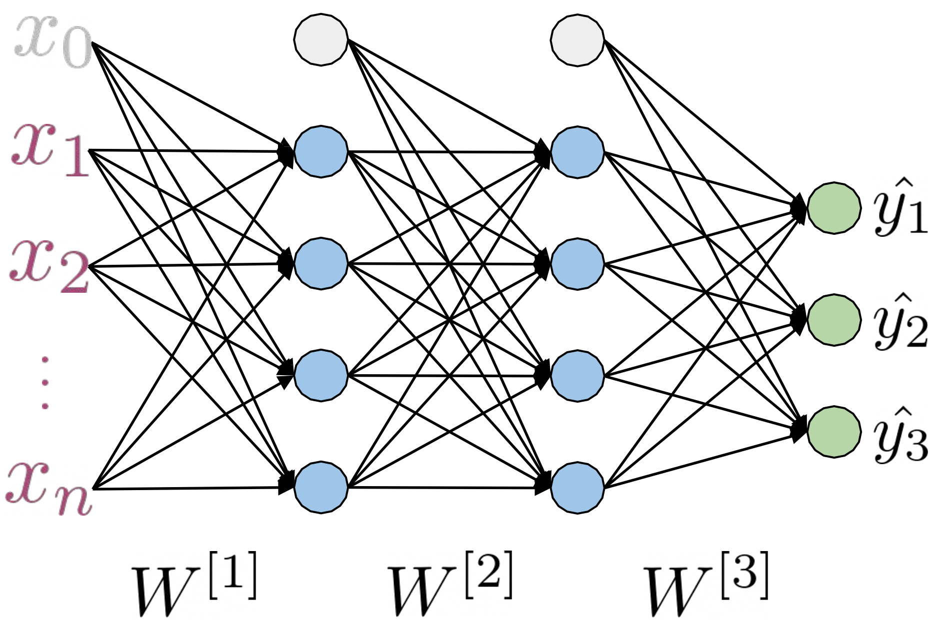


# Summary



- LSTMs use a series of gates to decide which information to keep:
  - Forget gate decides what to keep
  - Input gate decides what to add
  - Output gate decides what the next hidden state will be

# Cross Entropy Loss



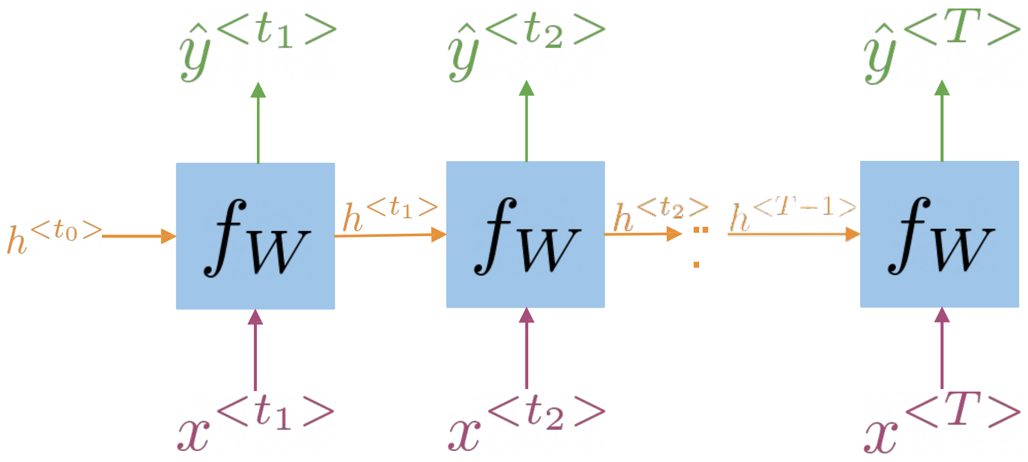

K - classes or possibilities

Either 0 or 1

$$J = -\sum_{j=1}^{\boxed{K}} \boxed{y_j} \log \hat{y}_j$$

Looking at a single example $(x, y)$

# Cross Entropy Loss for RNNs




$$h^{<t>} = g(W_h[h^{<t-1>}, x^{<t>}] + b_h)$$

$$\hat{y}^{<t>} = g(W_{yh}h^{<t>} + b_y)$$

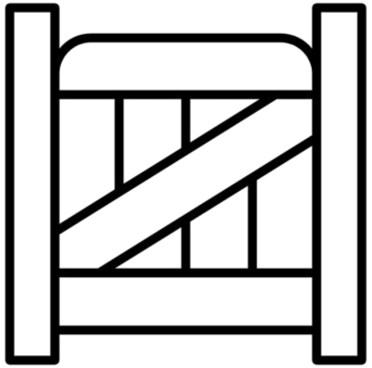
$$J = -\frac{1}{T}\sum_{t=1}^{T}\sum_{j=1}^{K} y_j^{<t>} \log \hat{y}_j^{<t>}$$

Average with respect to time

For RNNs the loss function is just an average through time!

# Gated Recurrent Unit (GRU)

- Gated recurrent unit (GRU) structure
- Comparison between GRUs and vanilla RNNs



# GRU – Gated Recurrent Unit

**Introduced by Cho et al., 2014**

**Simpler than LSTM**, with fewer gates

- No separate memory cell
- Combines forget and input into **update gate**

**Key Components:**

- **Update Gate** $z_t$
- **Reset Gate** $r_t$

# GRU – Gated Recurrent Unit

**Equations:**

$$z_t = \sigma(W_z[x_t, h_{t-1}])$$
$$r_t = \sigma(W_r[x_t, h_{t-1}])$$
$$\tilde{h}_t = \tanh(W_h[x_t, r_t * h_{t-1}])$$
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

**Comparable performance to LSTM**
**Faster training, fewer parameters**

# Gated Recurrent Units

"Ants are really interesting. They are everywhere."

Plural

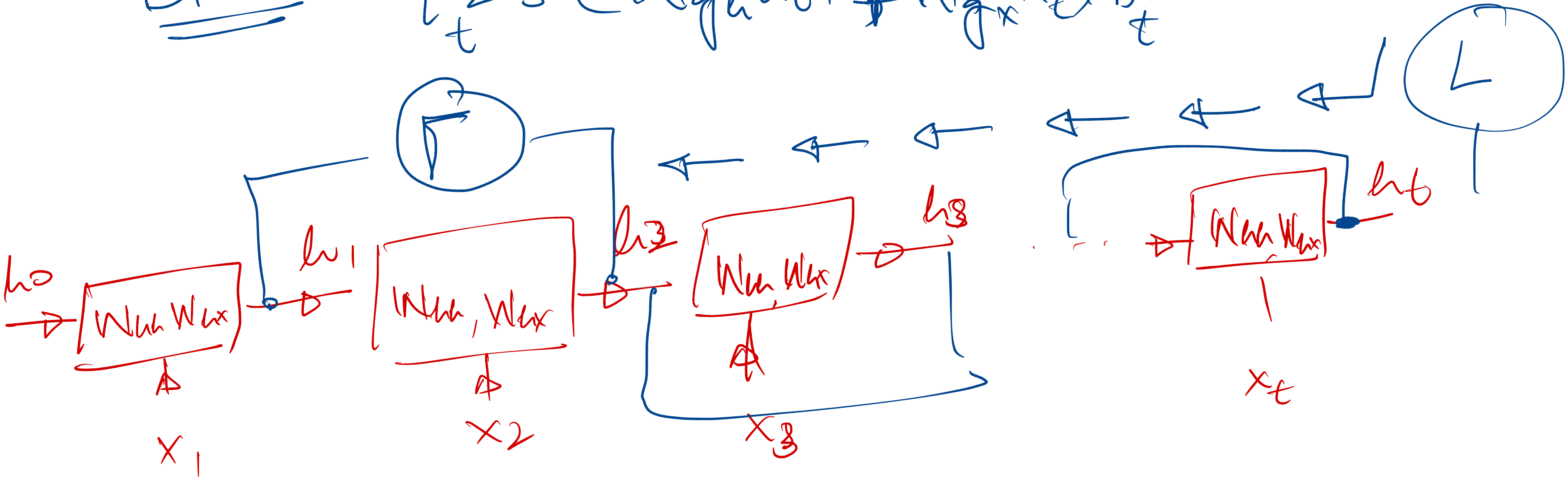
Relevance and update gates to remember important prior information

Vanilla

$$h_t = \tanh(W_{hh} h_{t-1} + W_{hx} x_t + B_h)$$

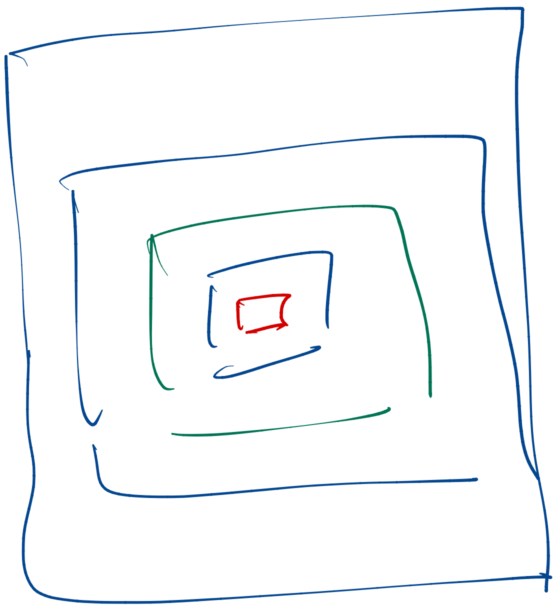
BPTT

$$\Gamma_t = \sigma(W_{gh} h_{t-1} + W_{gx} x_t + B_t)$$

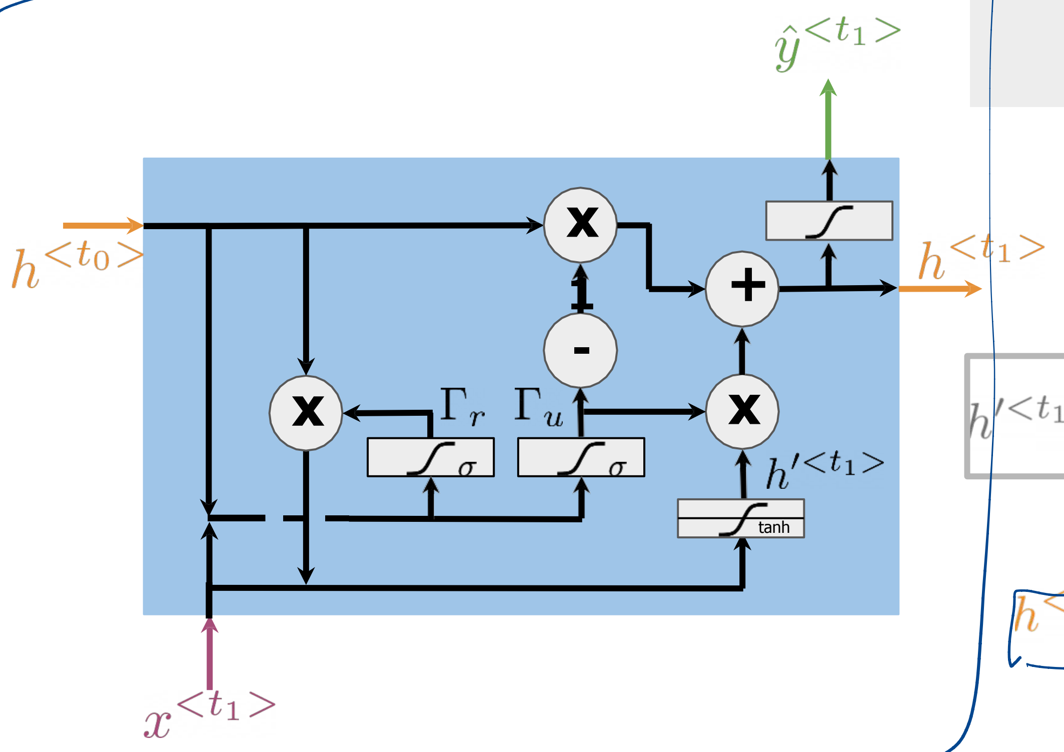


$$\underline{\underline{(n \ln n)^n e^{-n}}} \qquad n >> 1$$

Skip connections ??



# Gated Recurrent Unit




Gates to keep/update relevant information in the hidden state

$$\Gamma_r = \sigma(W_r[h^{<t_0>}, x^{<t_1>}] + b_r)$$

$$\Gamma_u = \sigma(W_u[h^{<t_0>}, x^{<t_1>}] + b_u)$$

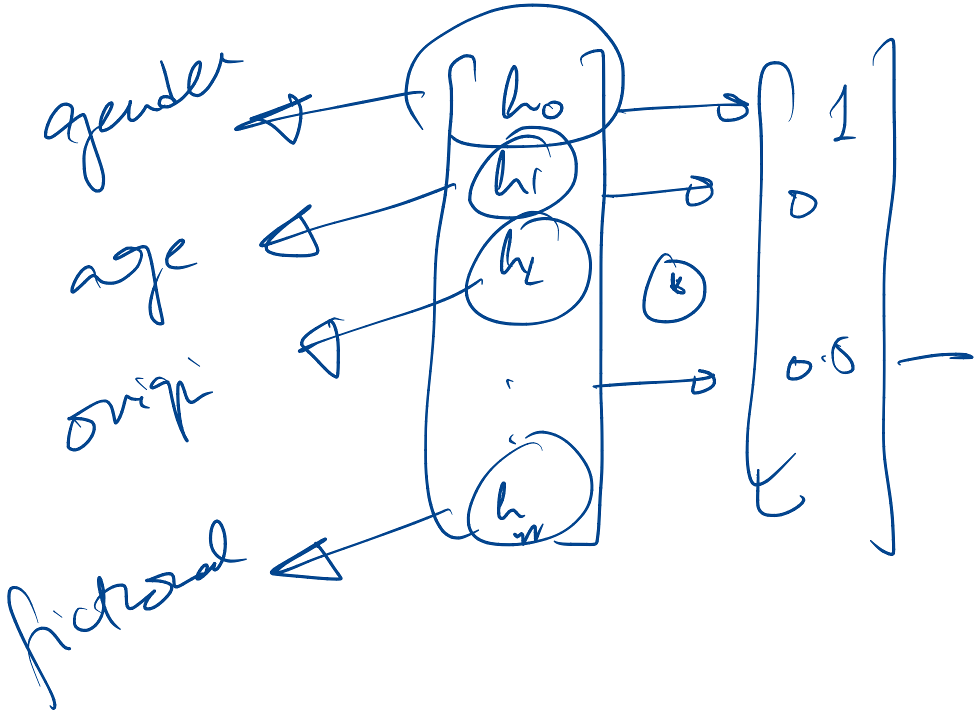
$$h'^{<t_1>} = \tanh\left(W_h[\Gamma_r * h^{<t_0>}, x^{<t_1>}] + b_h\right)$$

Hidden state candidate

$$h^{<t_1>} = (1 - \Gamma_u) * h^{<t_0>} + \Gamma_u * h'^{<t_1>}$$

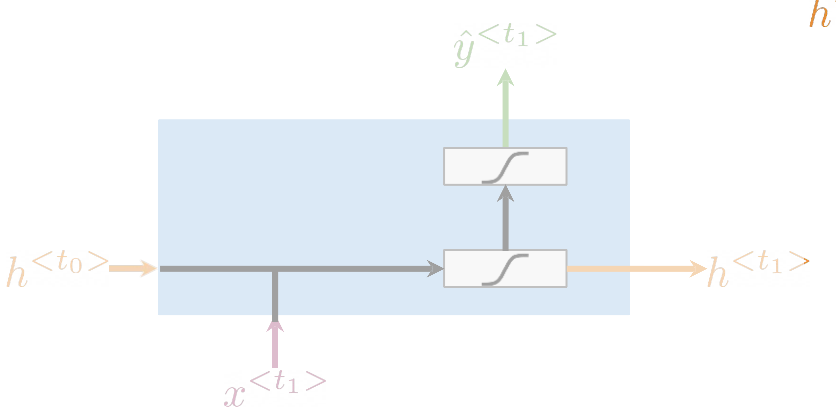
$$\hat{y}^{<t_1>} = g(W_y h^{<t_1>} + b_y)$$

$$tw = \boxed{2(6) - 1}$$

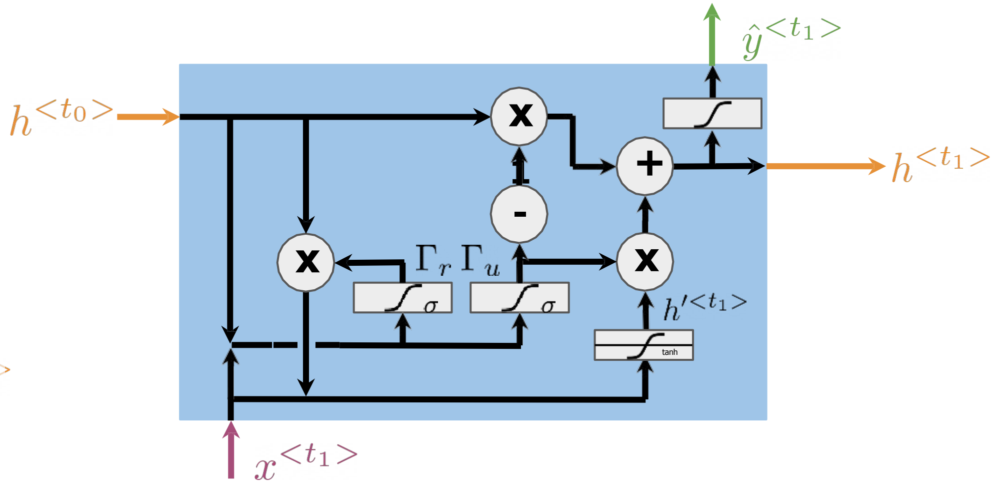


gender
age
origi
fictional
$h_0$
$h_1$
$h_2$
$h_n$
1
0
0.5

# Vanilla RNN vs GRUs



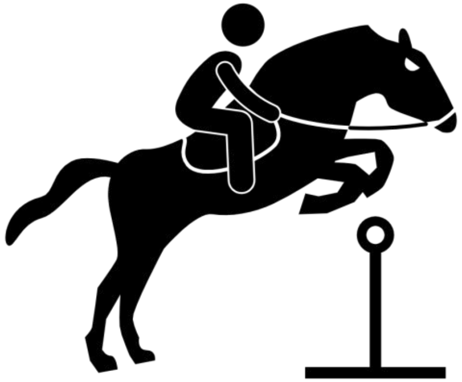

$$h^{<t>} = g(W_h[h^{<t-1>}, x^{<t>}] + b_h)$$
$$\hat{y}^{<t>} = g(W_{yh}h^{<t>} + b_y)$$




$$\Gamma_u = \sigma(W_u[h^{<t_0>}, x^{<t_1>}] + b_u)$$
$$\Gamma_r = \sigma(W_r[h^{<t_0>}, x^{<t_1>}] + b_r)$$
$$h'^{<t_1>} = \tanh\left(W_h[\Gamma_r * h^{<t_0>}, x^{<t_1>}] + b_h\right)$$
$$h^{<t_1>} = (1 - \Gamma_u) * h^{<t_0>} + \Gamma_u * h'^{<t_1>}$$
$$\hat{y}^{<t_1>} = g(W_y h^{<t_1>} + b_y)$$

# Summary

- GRUs “decide” how to update the hidden state
- GRUs help preserve important information



# LSTM vs GRU



| Feature | LSTM | GRU |
|---|---|---|
| Gates | 3 (i, f, o) | 2 (z, r) |
| Memory Cell | Yes | No |
| Complexity | Higher | Lower |
| Training Speed | Slower | Faster |
| Performance | Great for long sequences | Similar or better on short tasks |

**Tip:** Try GRU first for faster results; switch to LSTM if performance suffers.

# Use Cases for GRU/LSTM



**NLP Tasks**

- Language Modeling
- Machine Translation
- Sentiment Analysis
- Chatbots

**Audio & Time Series**

- Music Generation
- Speech Recognition
- Anomaly Detection

**Video & Sequential Vision**

- Action Recognition
- Video Captioning

# Limitations



Even with GRU/LSTM:

- Still sequential $\rightarrow$ Hard to parallelize
- Struggle with very long-range dependencies
- Architectural complexity
- Hard to interpret gate decisions
- Require lots of training data

# Future Directions



- **Transformers:** Fully parallelized sequence modeling using attention
- **Efficient Attention:** Longformer, Linformer, etc. for long sequences
- **Neural Memory Networks:** Explicit memory read/write
- **Recurrent Attention Models**
- **Hybrid Architectures:** RNN + CNN + Attention

RNNs are still used in edge devices for efficient modeling

# Summary

- RNNs struggle with long dependencies due to vanishing/exploding gradients
- GRU and LSTM improve memory retention using gating mechanisms
- GRU is simpler and faster; LSTM is more expressive
- Attention and transformers now dominate, but RNNs remain relevant in many domains

# References

## Foundational Papers:


- Hochreiter, S., & Schmidhuber, J. (1997). *Long Short-Term Memory*. Neural Computation.
- Cho, K., van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014). *Learning Phrase Representations using RNN Encoder–Decoder with GRU*. EMNLP.
- Pascanu, R., Mikolov, T., & Bengio, Y. (2013). *On the difficulty of training RNNs*. ICML.
- Bengio, Y., Simard, P., & Frasconi, P. (1994). *Learning long-term dependencies*. IEEE Transactions on Neural Networks.
- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). *Attention Is All You Need*. NeurIPS.

# References

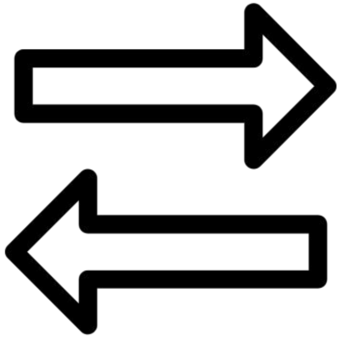


**Resources:**

- Karpathy's RNN Blog:
  https://karpathy.github.io/2015/05/21/rnn-effectiveness/
- CS231n Lecture Notes on RNNs and LSTM
- DeepLearning.ai NLP Specialization – Coursera
- MIT 6.S191 Deep Learning Lecture Slides

# Deep and Bi- directional RNNs

- How bidirectional RNNs propagate information
- Forward propagation in deep RNNs

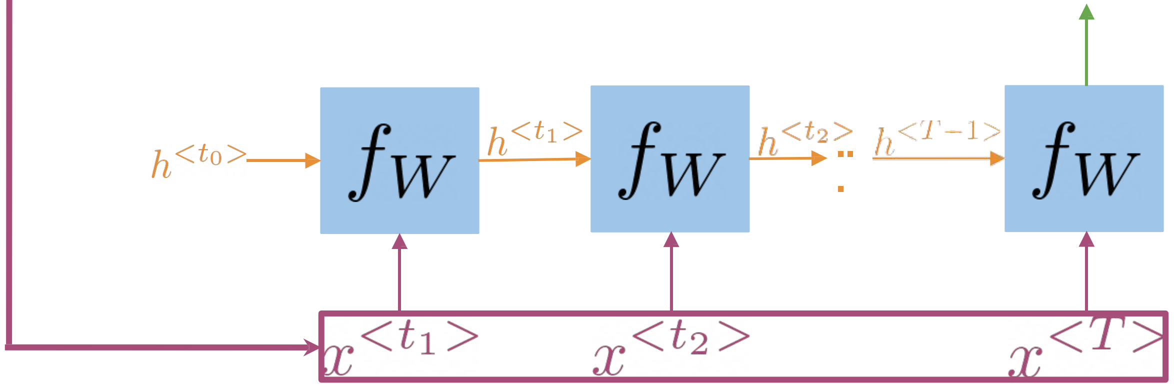


# Bi-directional RNNs

I was trying really hard to get a hold
answered when I was about to give
up.

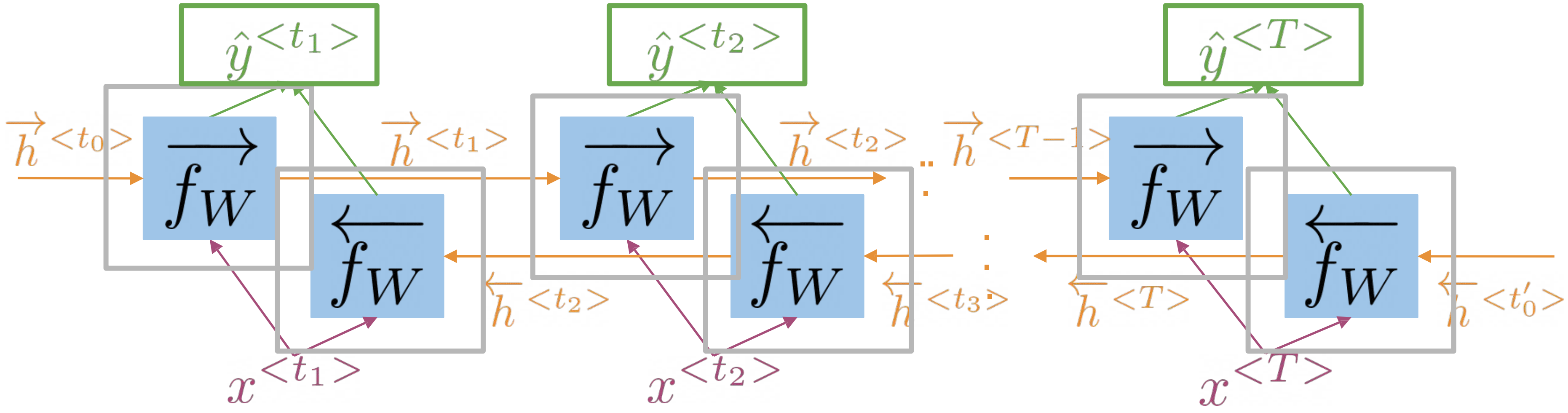
. **Louise,** finally

her him them



# Bi-directional RNNs

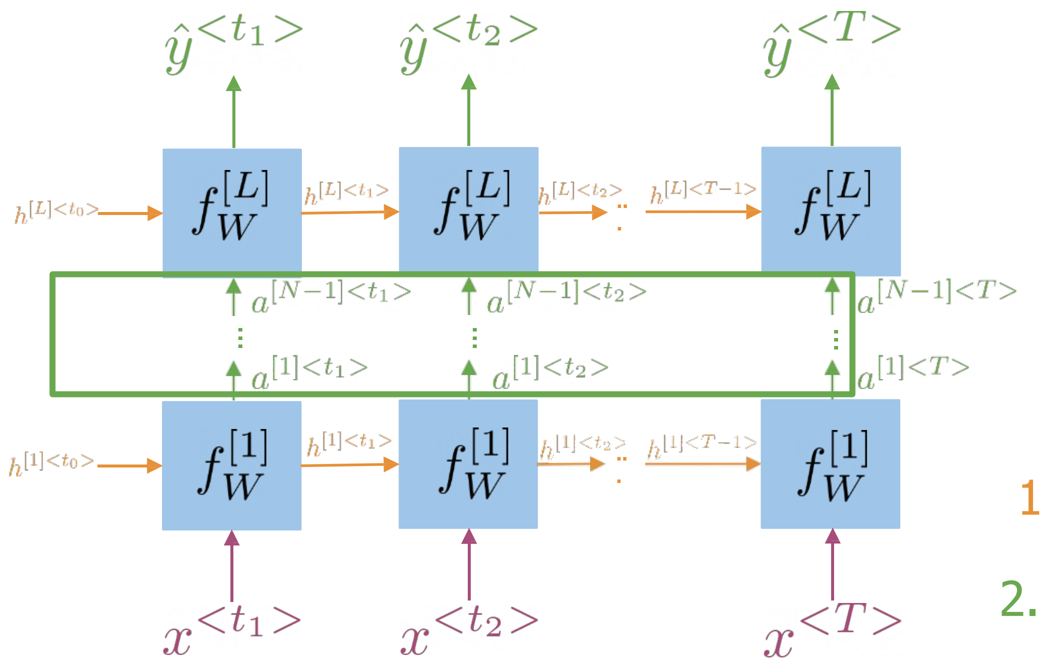





Information flows from the past and from the future
**independently**

$$\hat{y}^{<t>} = g(W_y[\overrightarrow{h}^{<t>}, \overleftarrow{h}^{<t>}] + b_y)$$

# Deep RNNs




$$h^{[l]<t>} = f^{[l]}(W_h^{[l]}[h^{[l]<t-1>}, a^{[l-1]<t>}] + b_h^{[l]})$$

$$a^{[l]<t>} = f^{[l]}(W_a^{[l]} h^{[l]<t>} + b_a^{[l]})$$

Intermediate layers and activations

1. Get hidden states for current layer
2. Pass the activations to the next layer

# Summary

- In bidirectional RNNs, the outputs take information from the past and
  the future
- Deep RNNs have more than one layer, which helps in complex tasks

# References

These slides have been adapted from

- Younes Mourri & Lukasz Kaiser, [Natural Language Processcing Specialization, DeepLearning.Ai]