

RNN & LSTM

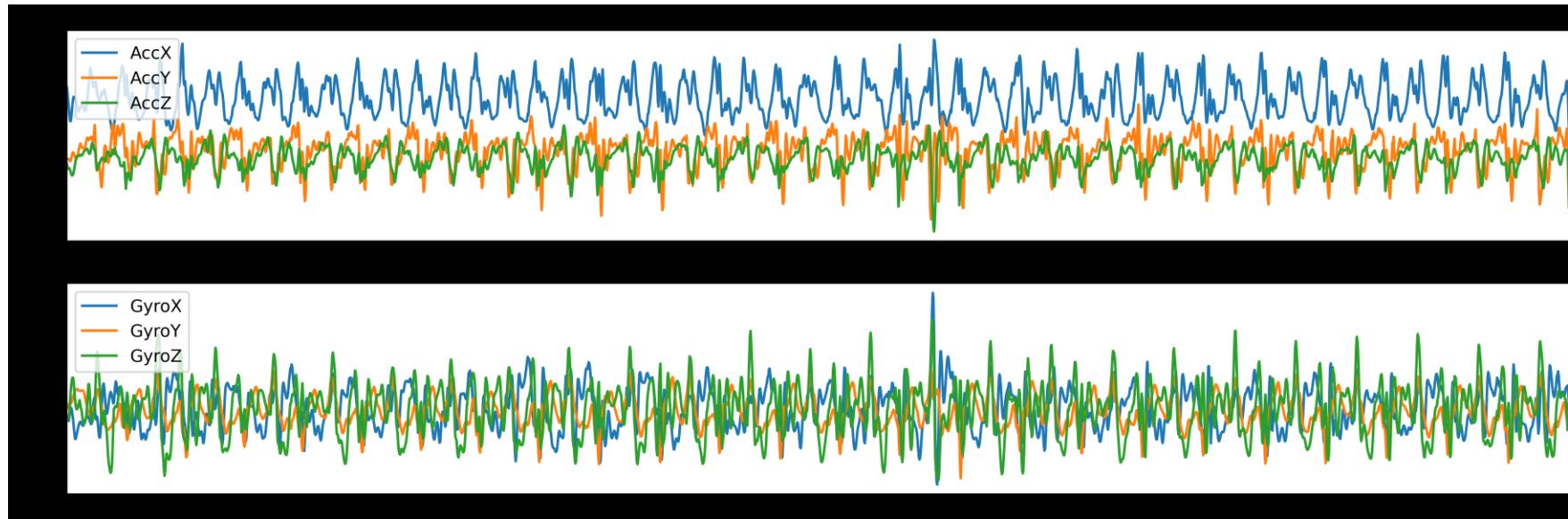
Nguyen Van Vinh - UET, VNU Ha Noi

Content

- Recurrent Neural Network
- The vanishing/exploding gradient problem
- LSTM
- Applications for LSTM

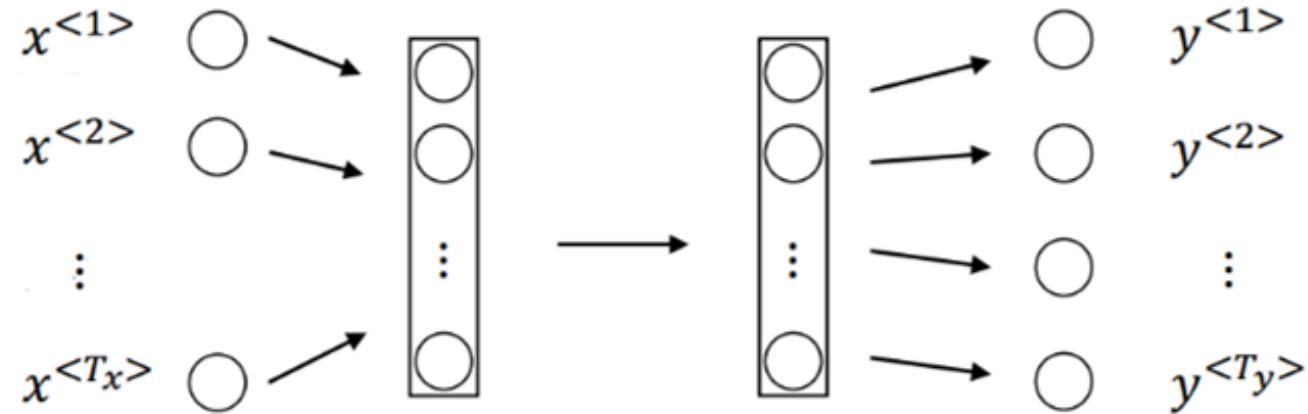
Sequence Data

- Time Series Data
- Natural Language



Data Source: <https://dl.acm.org/doi/10.1145/2370216.2370438>

Why not standard NN?



Problems:

- Inputs, outputs can be different lengths in different examples.
- Doesn't share features learned across different positions of text.

What is RNN?

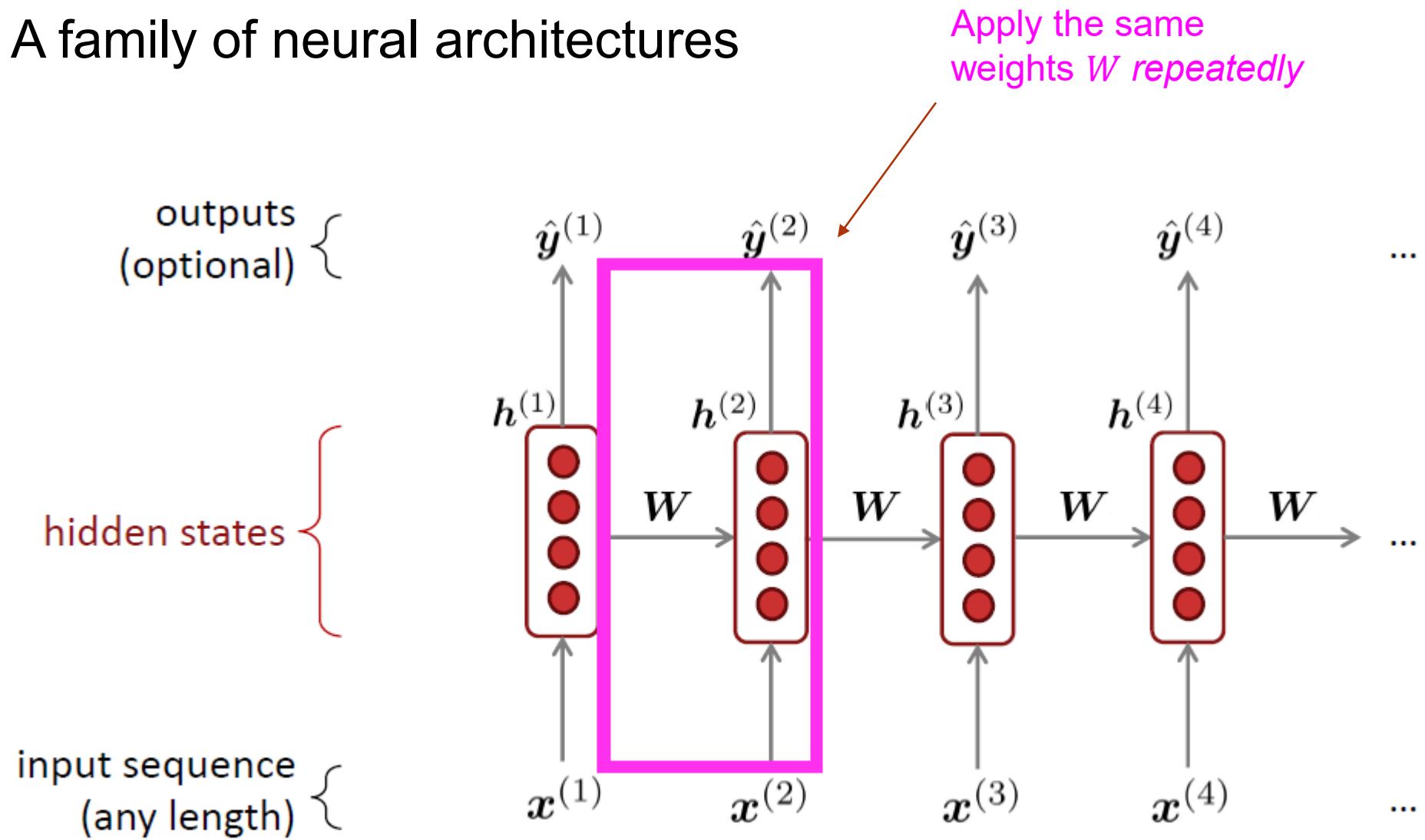
- We consider a class of recurrent networks referred to as **Elman Networks (Elman, 1990)**.
- A **recurrent neural network (RNN)** is a type of artificial neural network which used for sequential data or time series data.

Application:

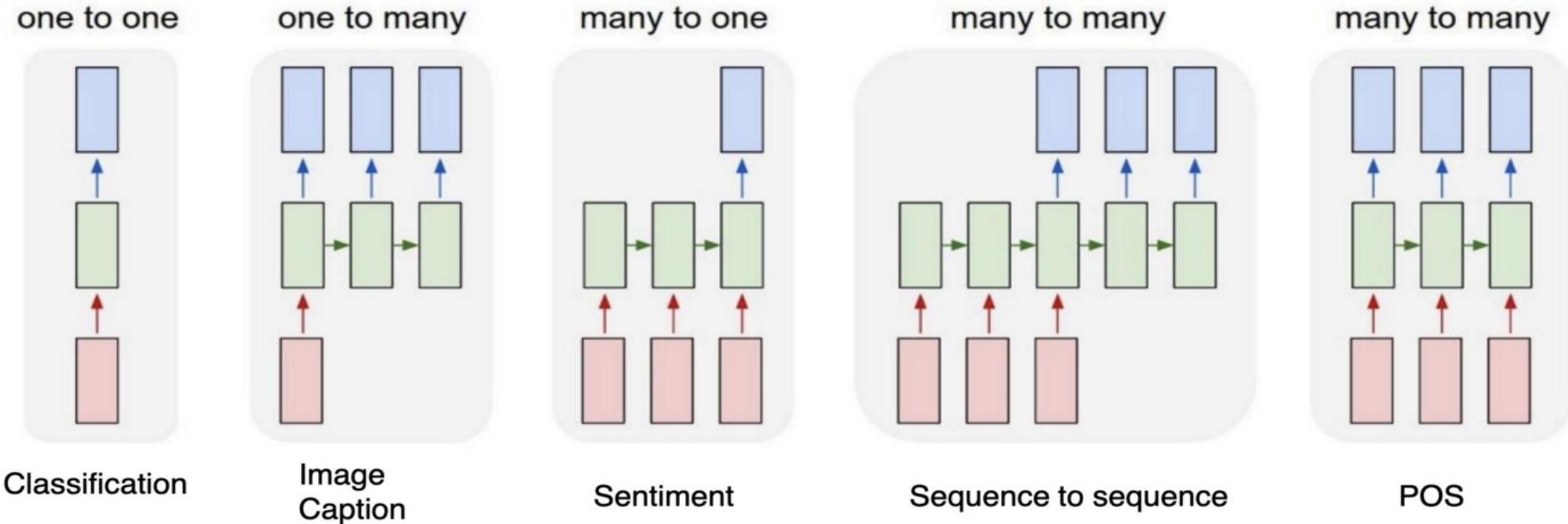
- + Language translation.
- + Natural language processing (NLP).
- + Speech recognition.
- + Image captioning.

Recurrent Neural Networks (RNN)

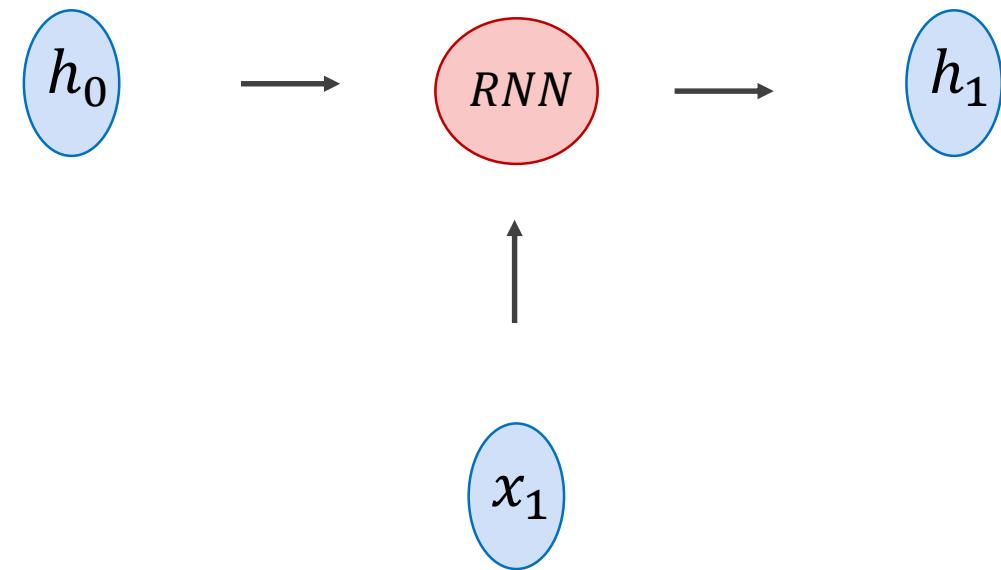
- A family of neural architectures



Types of RNN

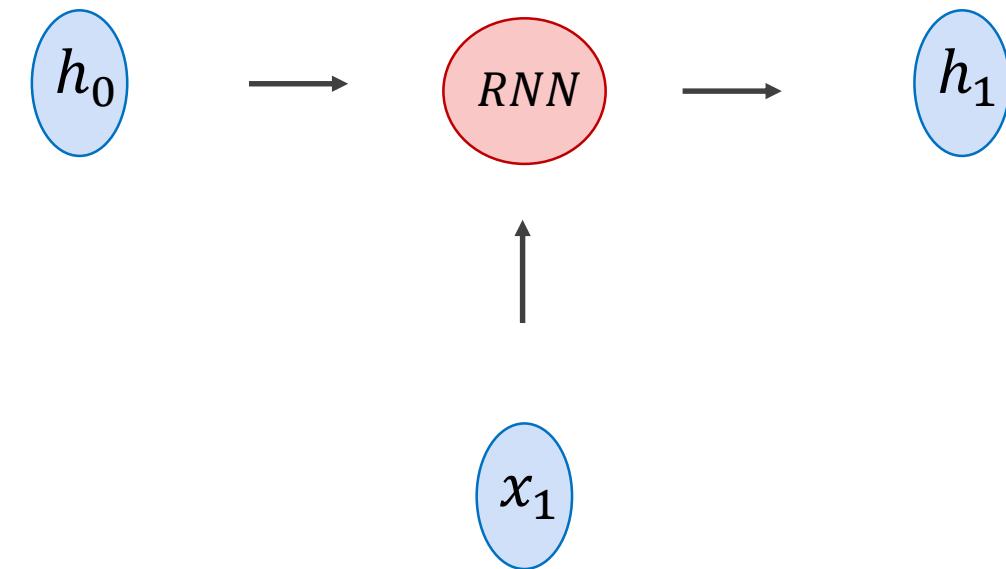


Recurrent Neural Network Cell

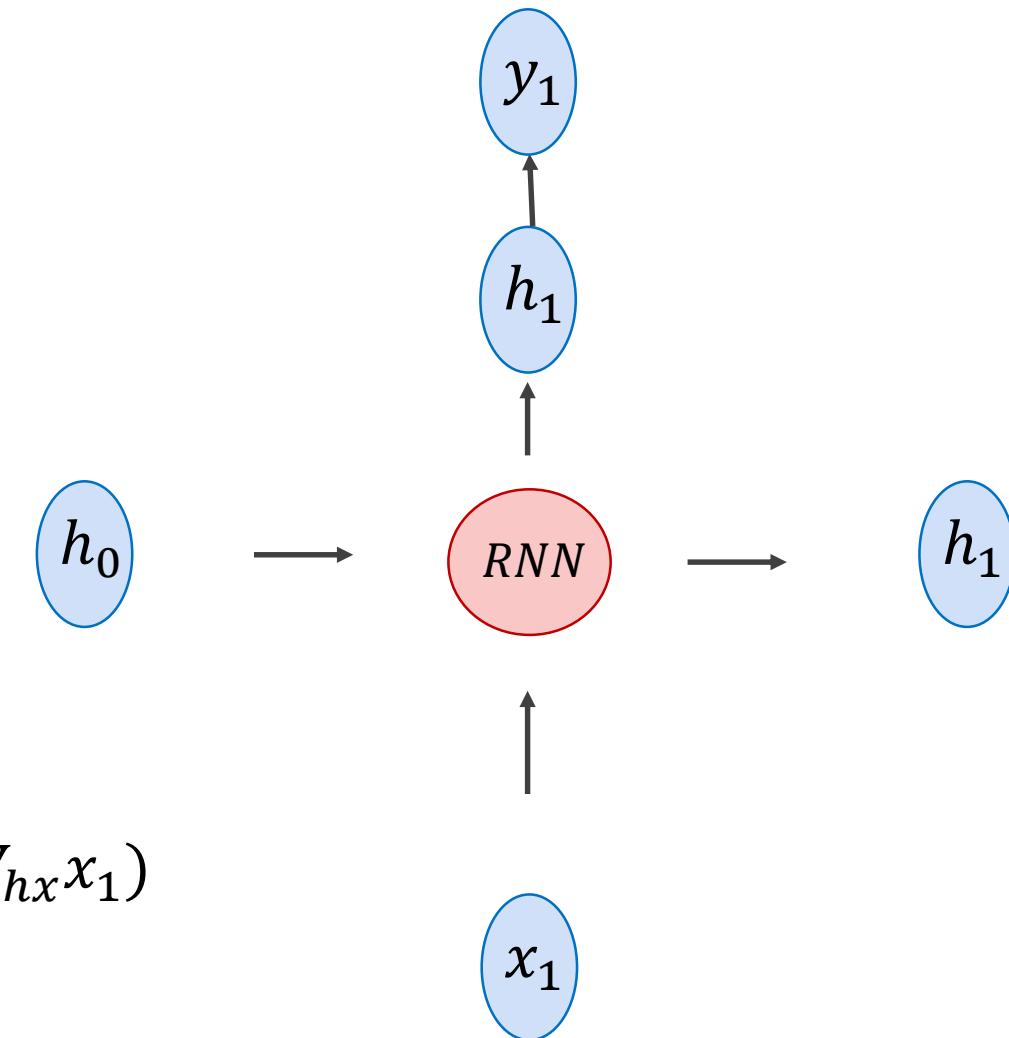


Recurrent Neural Network Cell

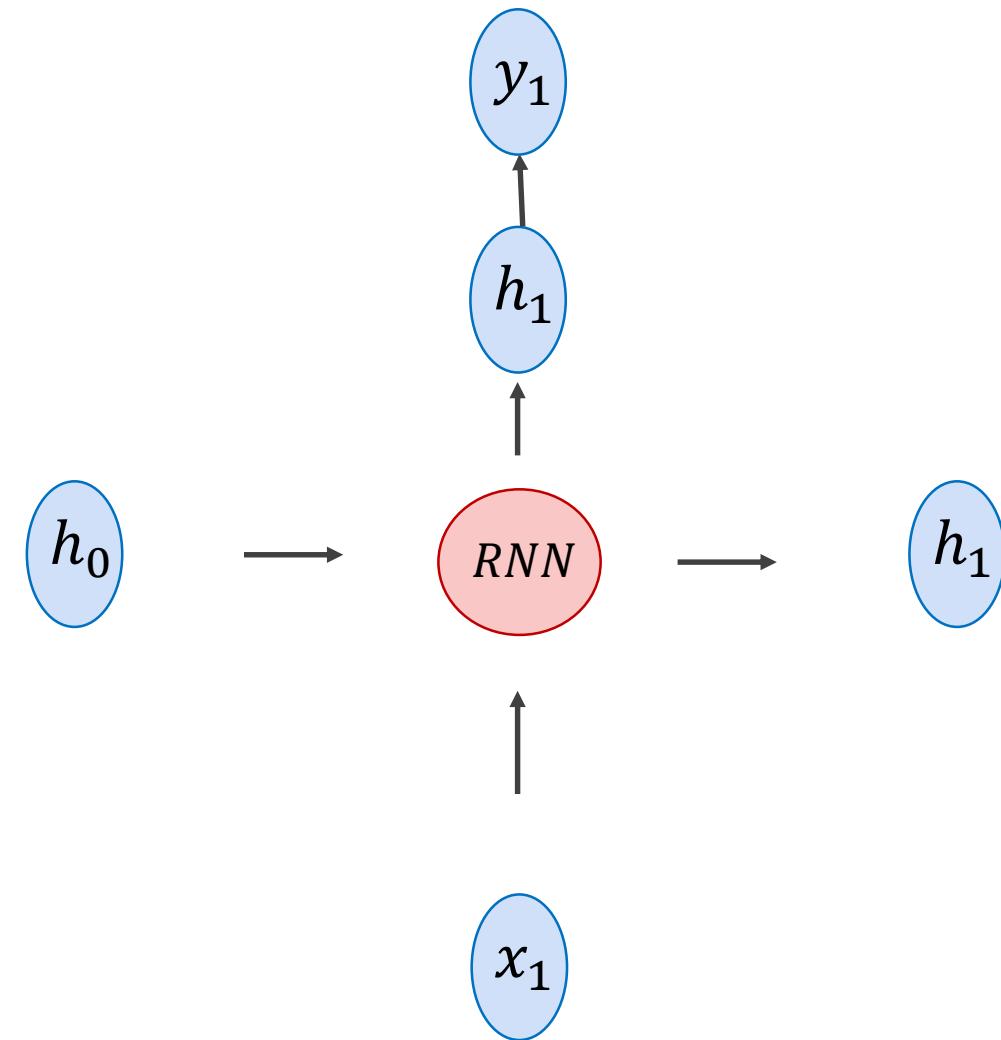
$$h_1 = \tanh(W_{hh}h_0 + W_{hx}x_1)$$



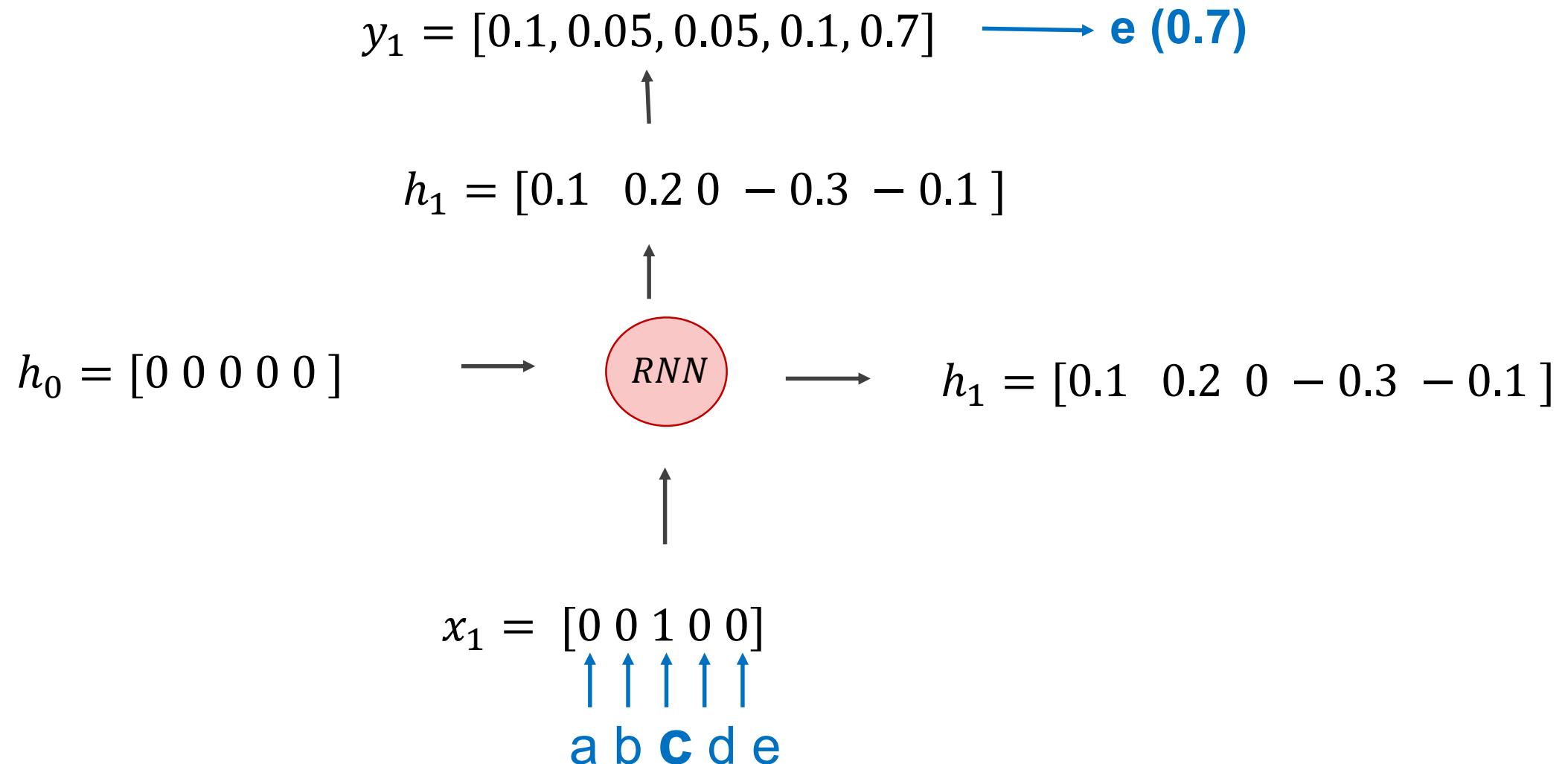
Recurrent Neural Network Cell



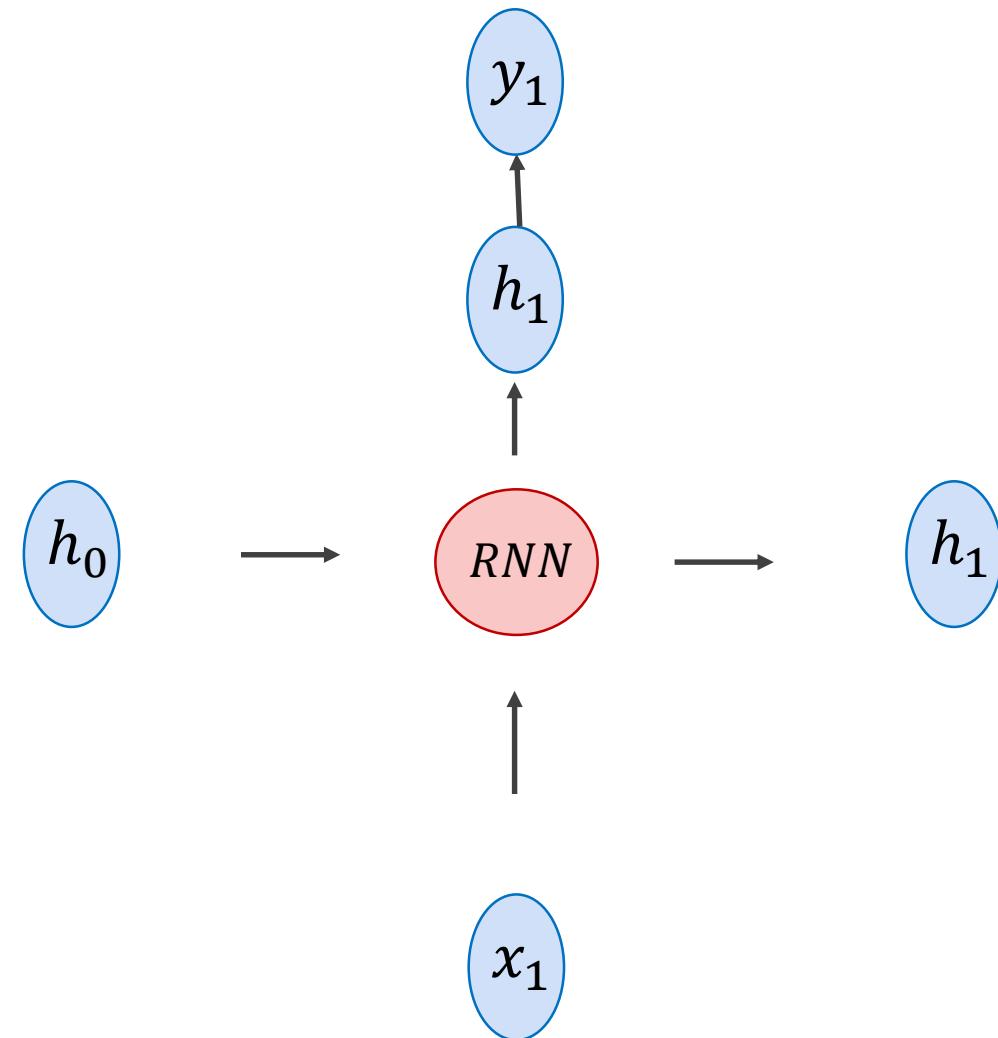
Recurrent Neural Network Cell



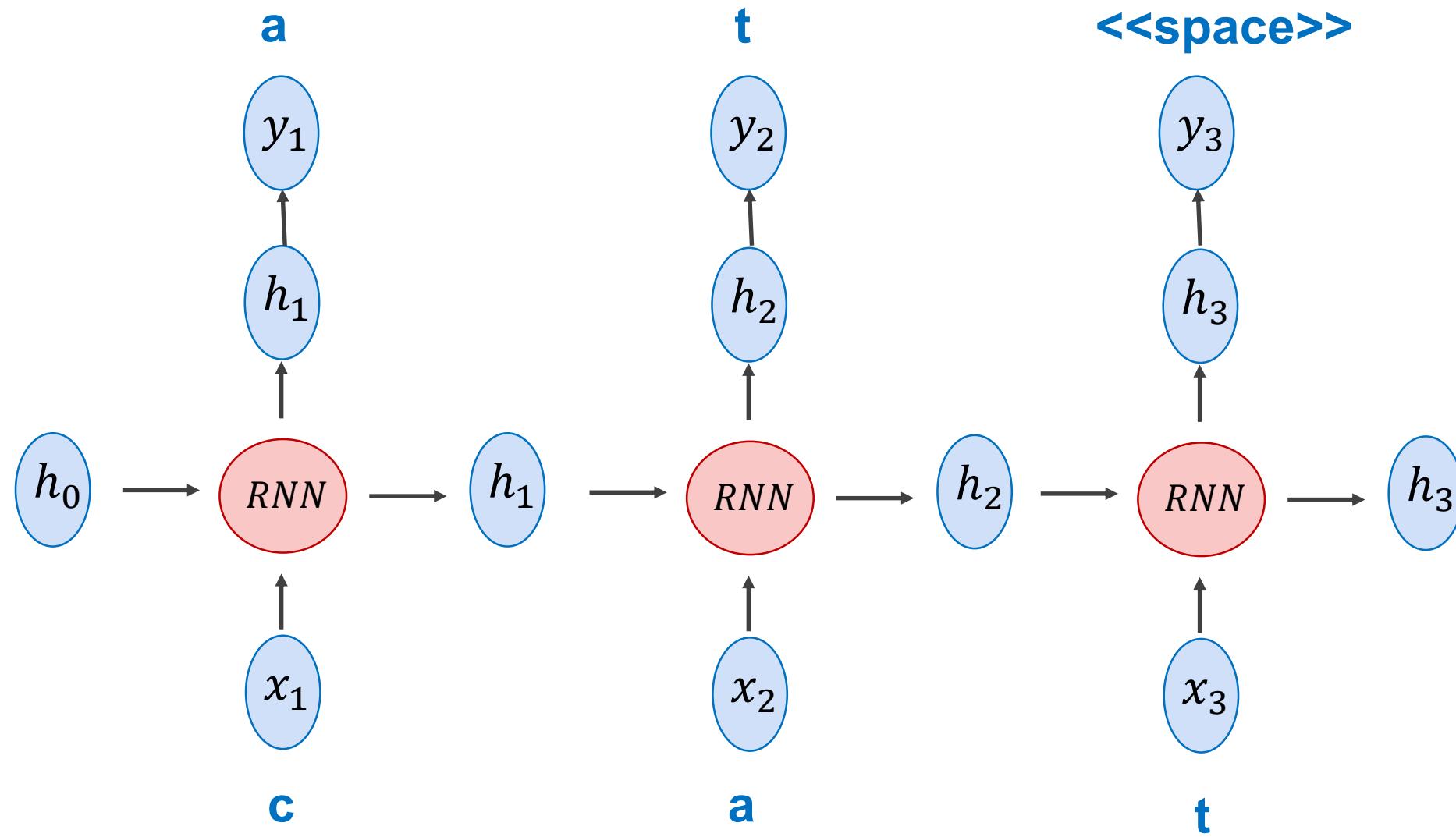
Recurrent Neural Network Cell



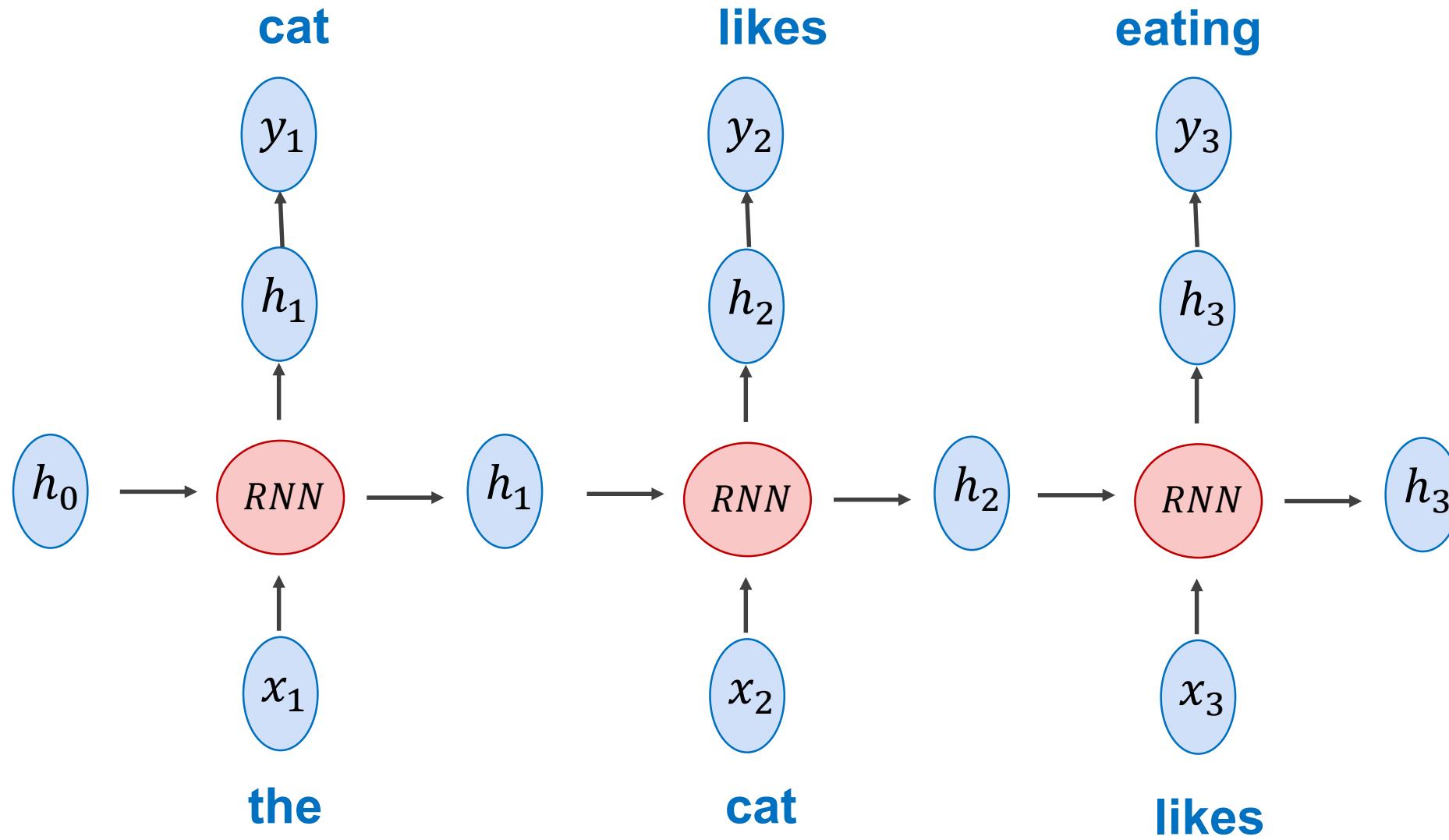
Recurrent Neural Network Cell



(Unrolled) Recurrent Neural Network

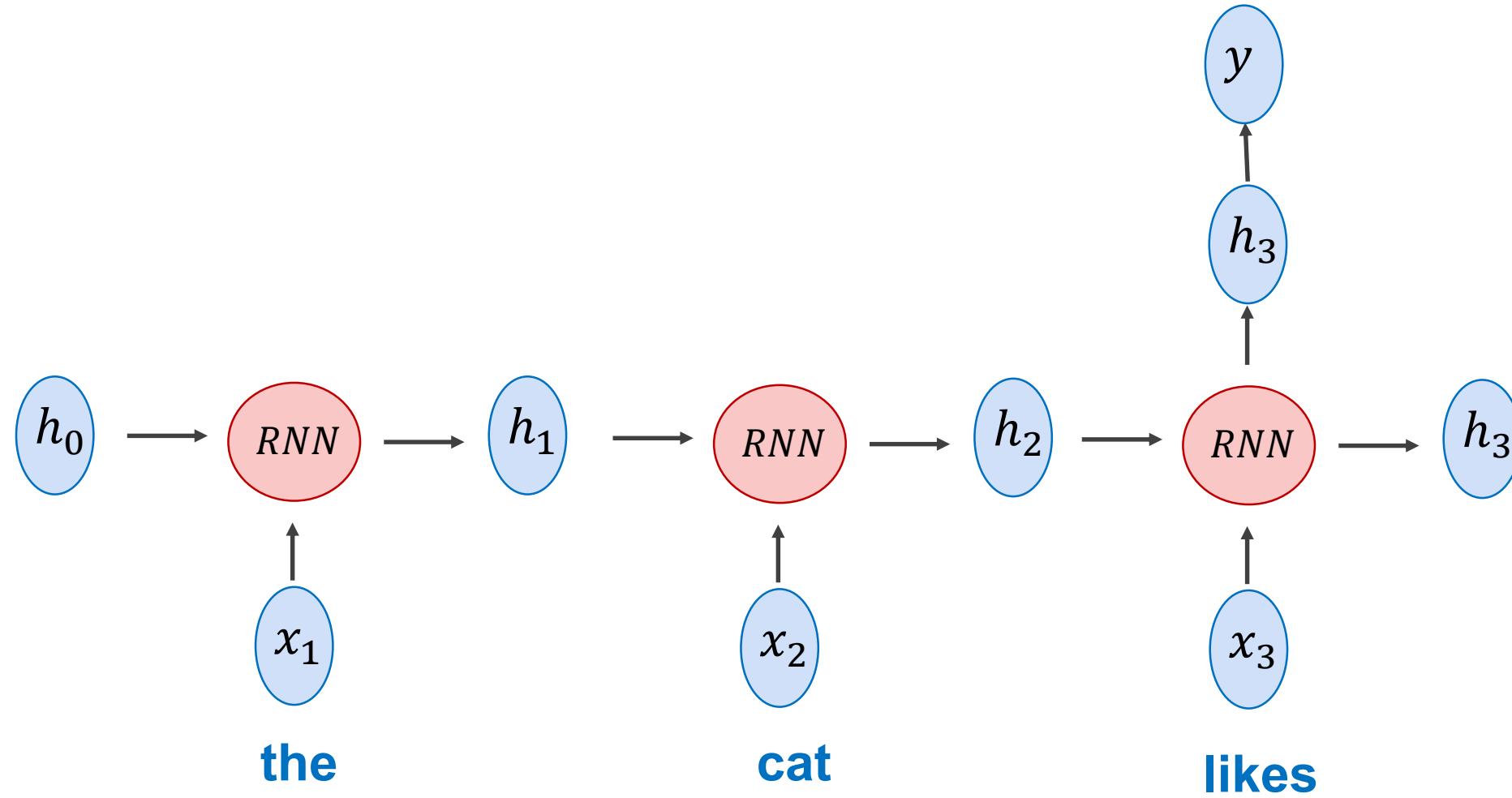


(Unrolled) Recurrent Neural Network

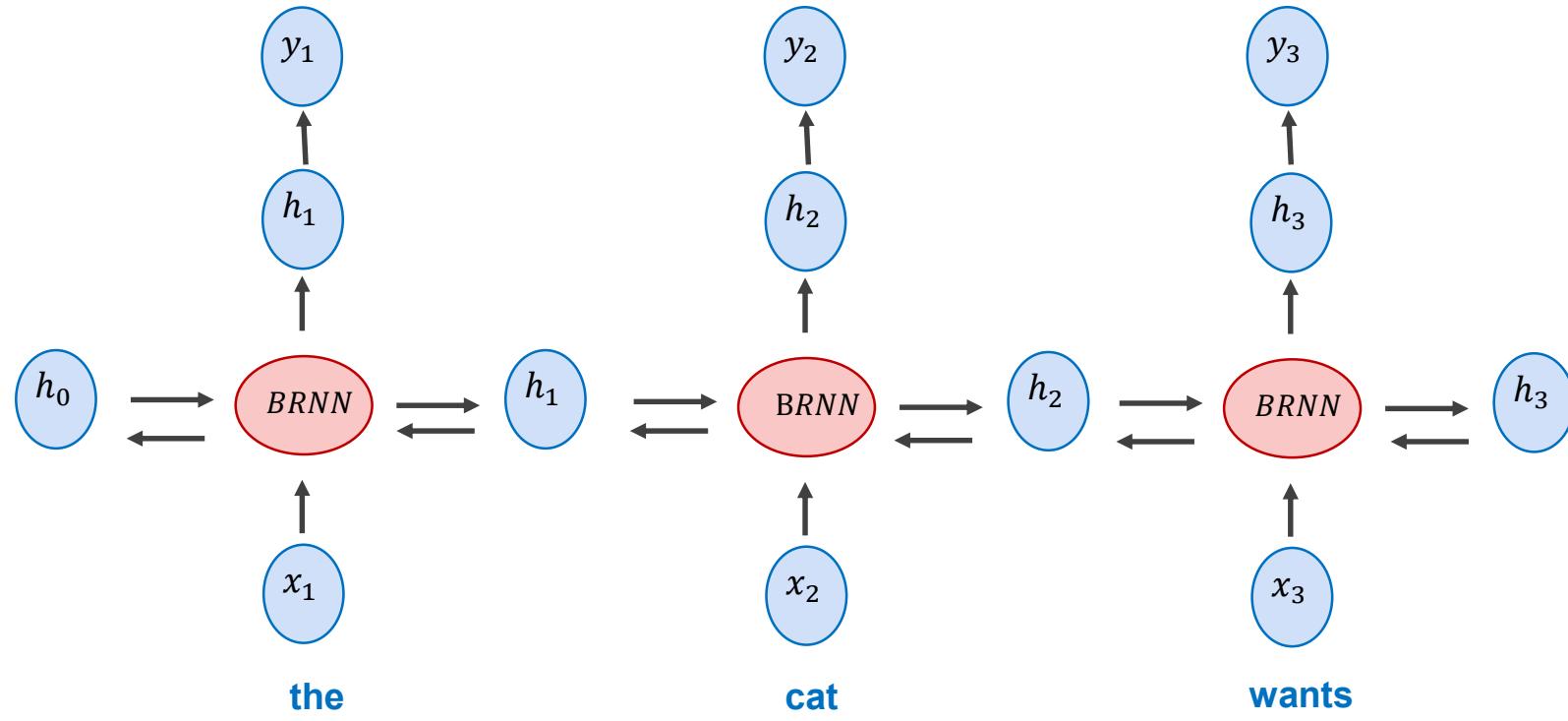


(Unrolled) Recurrent Neural Network

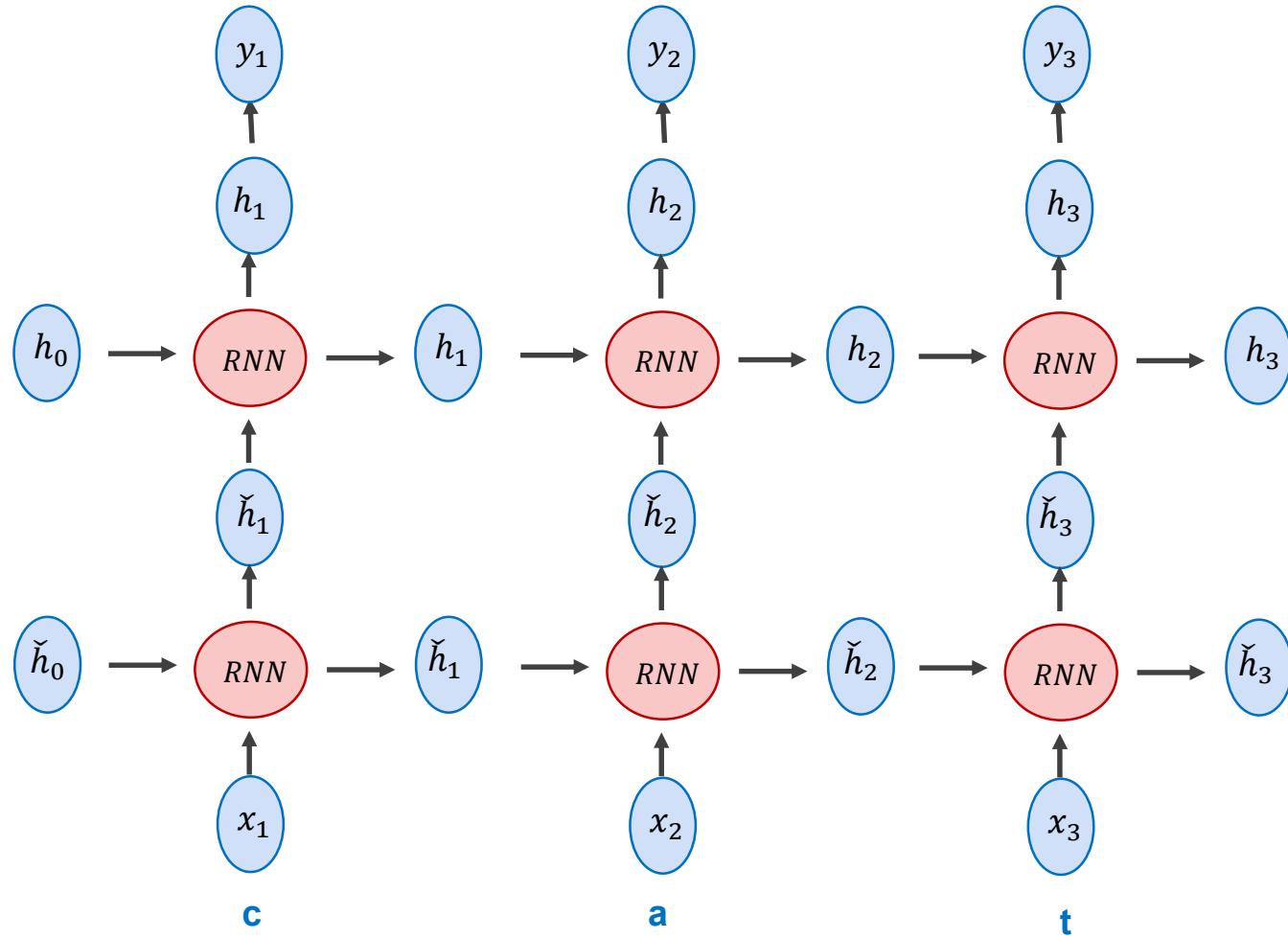
positive / negative sentiment rating



Bidirectional Recurrent Neural Network



Stacked Recurrent Neural Network



A Simple RNN Language Model

output distribution

$$\hat{y}^{(t)} = \text{softmax}(\mathbf{U} \mathbf{h}^{(t)} + \mathbf{b}_2) \in \mathbb{R}^{|V|}$$

hidden states

$$\mathbf{h}^{(t)} = \sigma(\mathbf{W}_h \mathbf{h}^{(t-1)} + \mathbf{W}_e \mathbf{e}^{(t)} + \mathbf{b}_1)$$

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

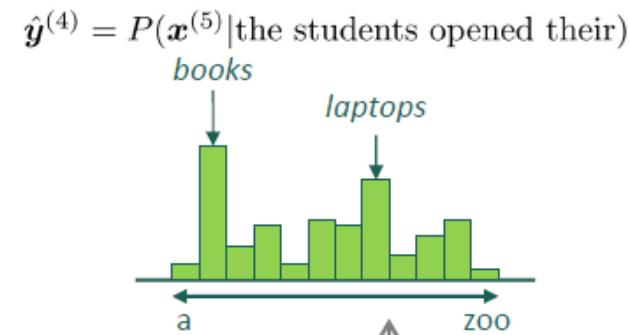
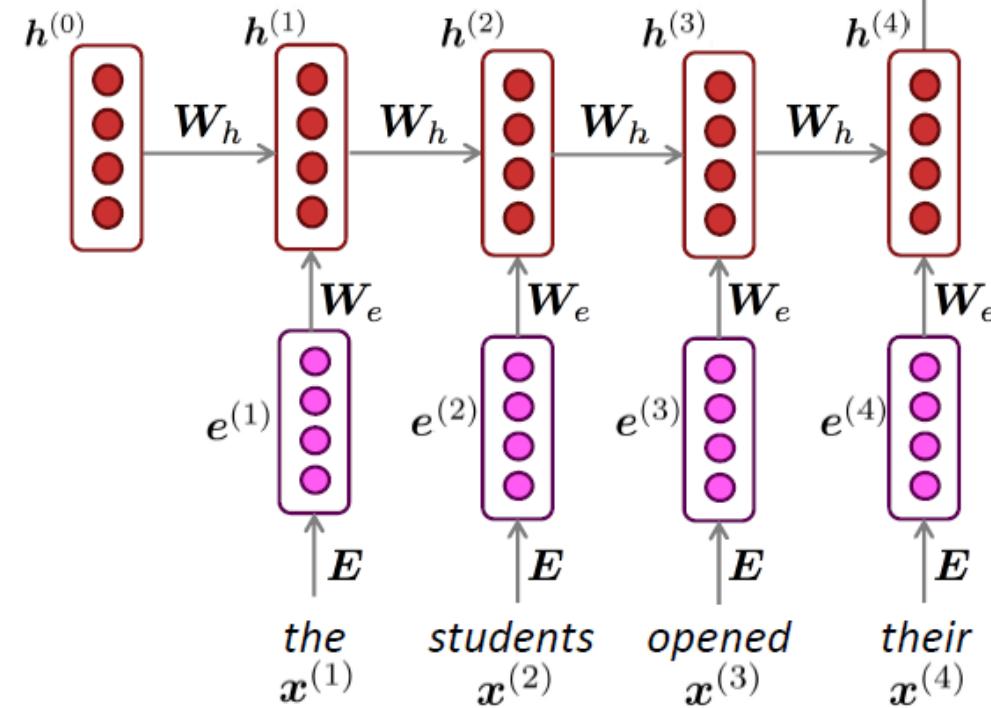
word embeddings

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

words / one-hot vectors

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

Note: this input sequence could be much longer now!



Ac

Training an RNN Language Model

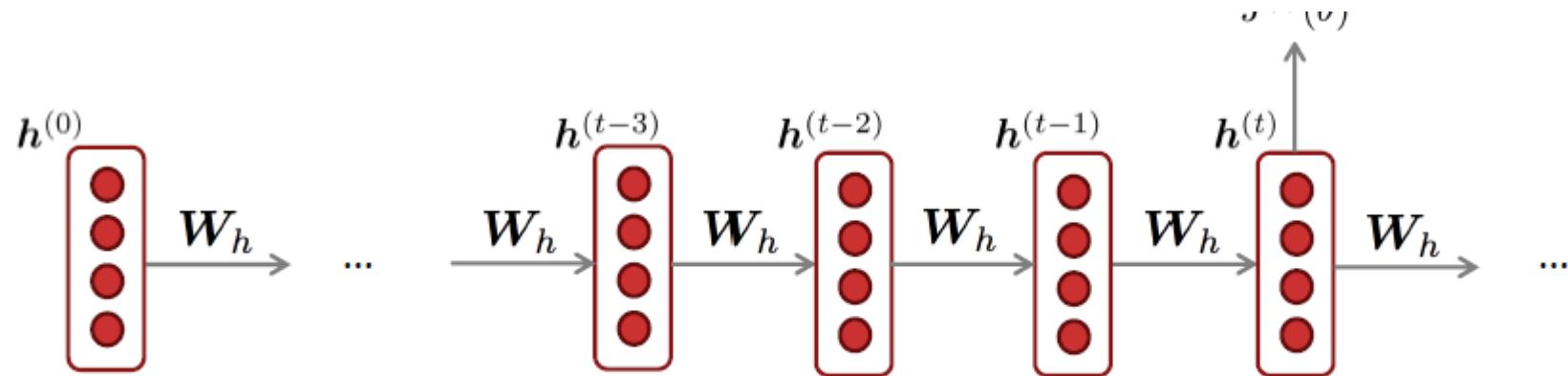
- Get a **big corpus of text** which is a sequence of words $\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(T)}$
- Feed into RNN-LM; compute output distribution $\hat{\mathbf{y}}^{(t)}$ **for every step t .**
 - i.e., predict probability dist of *every word*, given words so far
- **Loss function** on step t is **cross-entropy** 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 **overall loss** for entire training set:

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

Backpropagation for RNNs



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

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

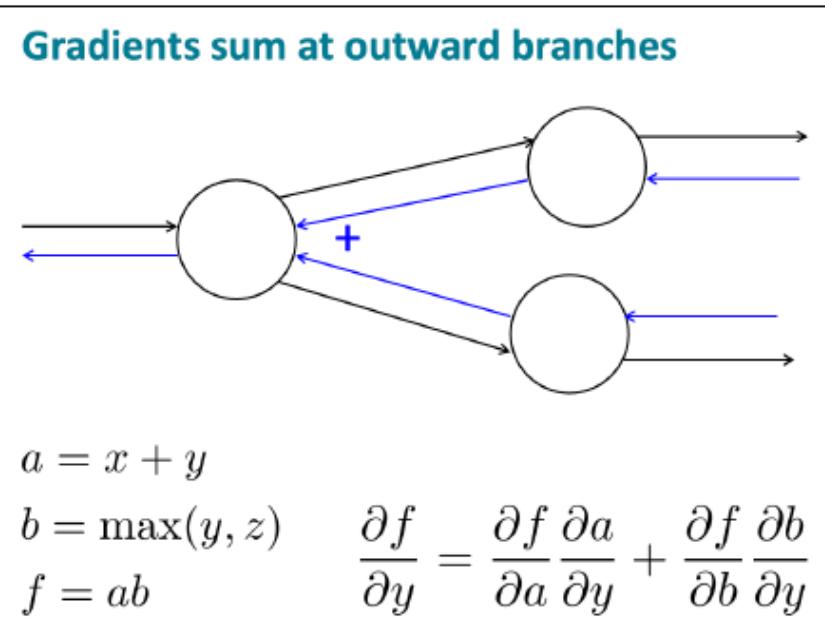
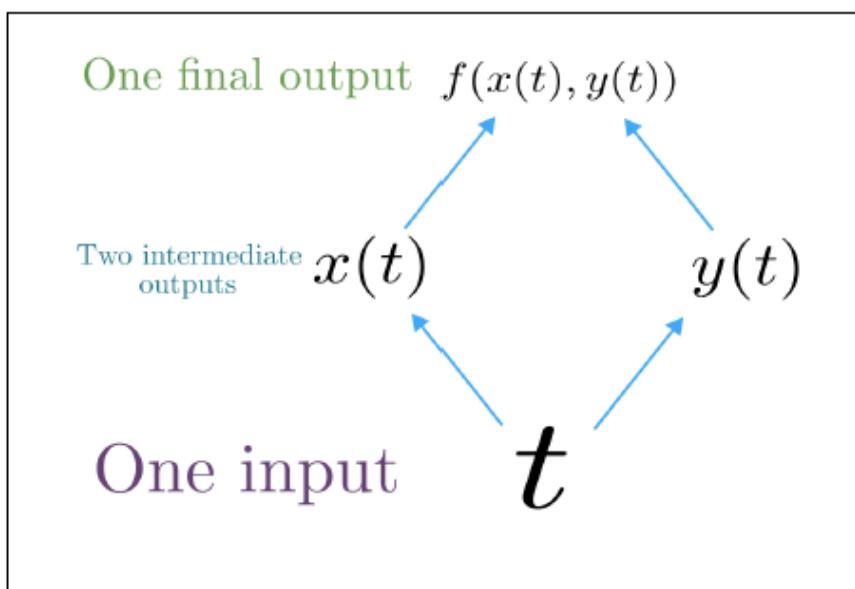
“The gradient w.r.t. a repeated weight
is the sum of the gradient
w.r.t. each time it appears”

Why?

Multivariable Chain Rule

- Given a multivariable function $f(x, y)$, and two single variable functions $x(t)$ and $y(t)$, here's what the multivariable chain rule says:

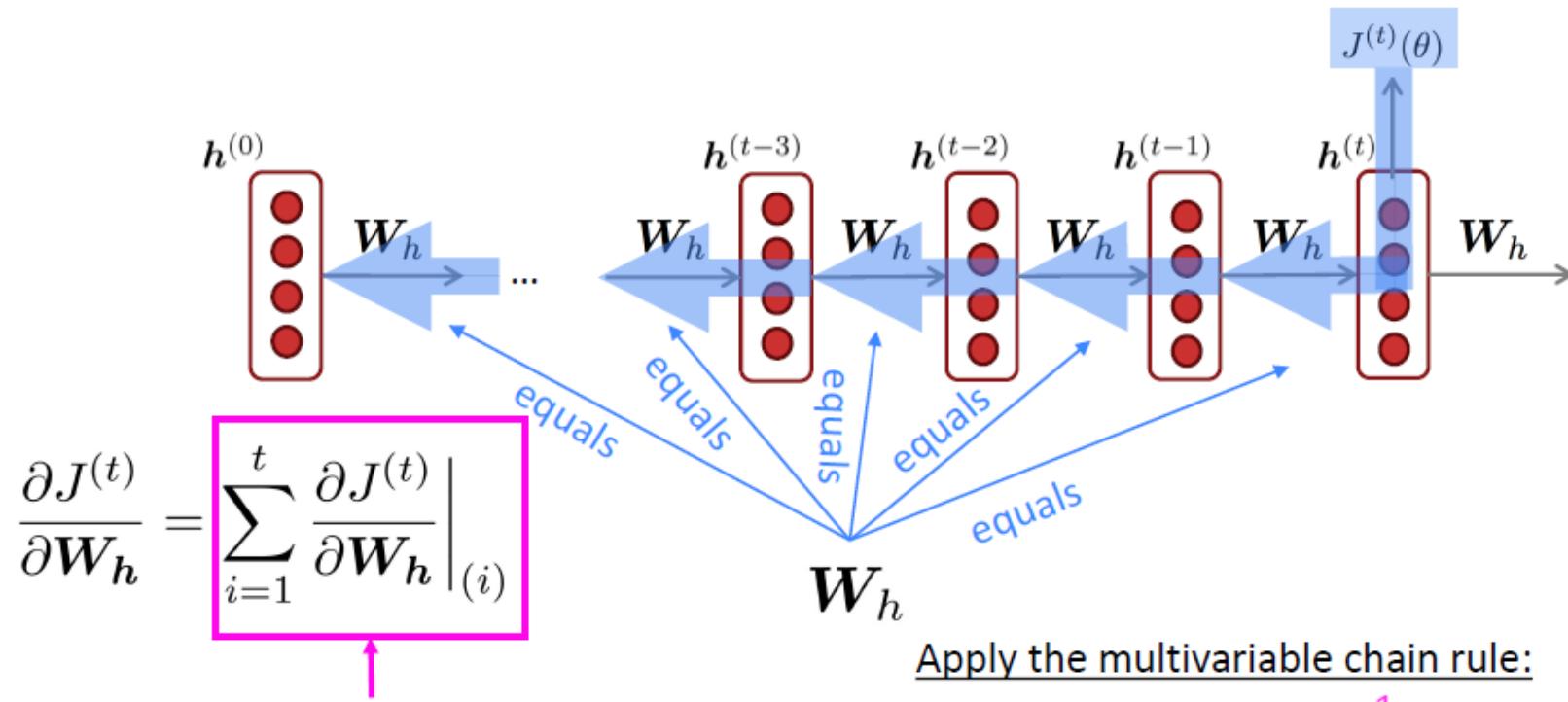
$$\underbrace{\frac{d}{dt} f(x(t), y(t))}_{\text{Derivative of composition function}} = \frac{\partial f}{\partial x} \frac{dx}{dt} + \frac{\partial f}{\partial y} \frac{dy}{dt}$$



Source: <https://www.khanacademy.org/math/multivariable-calculus/multivariable-derivatives/differentiating-vector-valued-functions/a/multivariable-chain-rule-simple-version>

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]

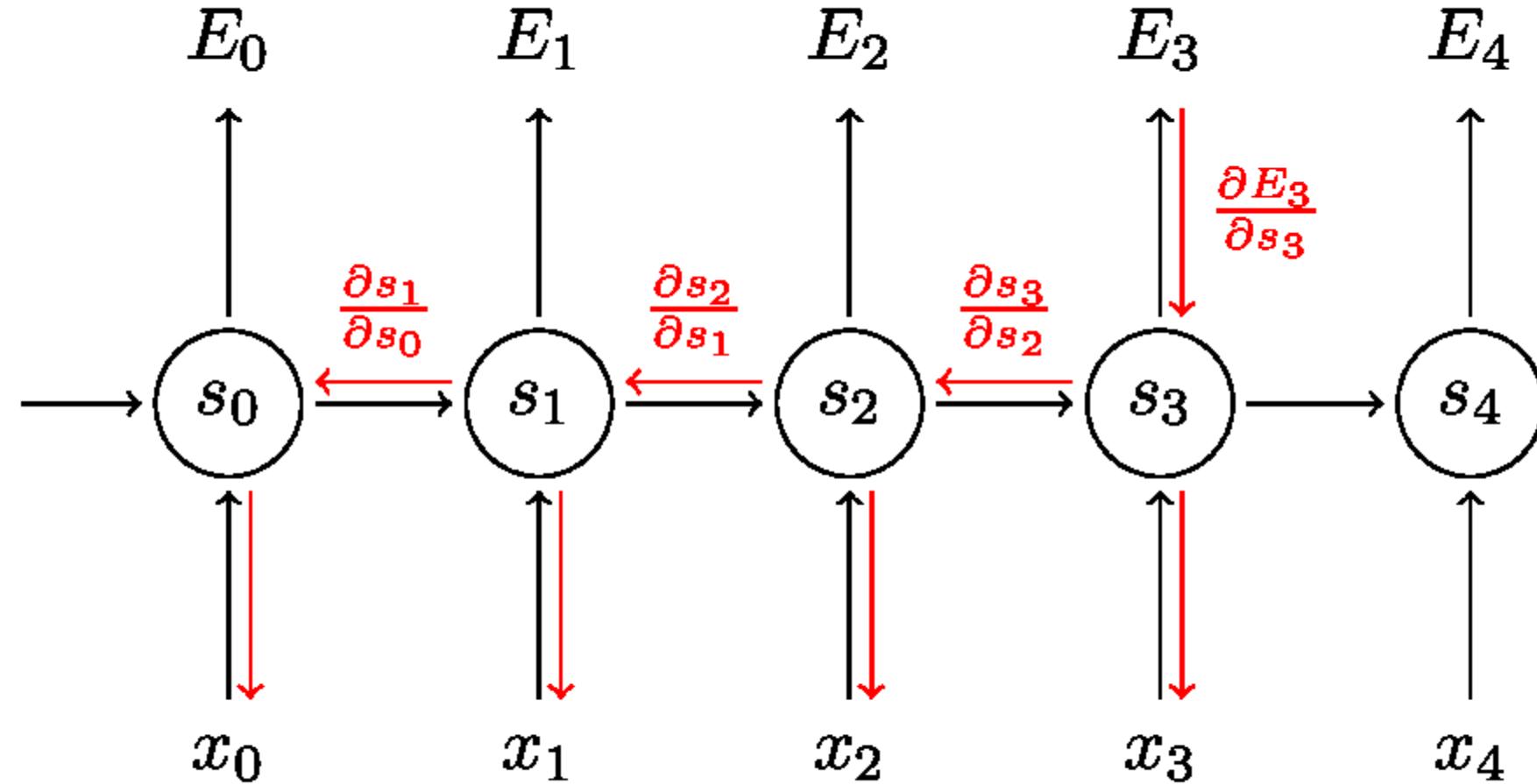
Apply the multivariable chain rule:

= 1

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

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

The vanishing gradient problem



The vanishing gradient problem

- Similar but simpler RNN formulation:

$$\begin{aligned} h_t &= Wf(h_{t-1}) + W^{(hx)}x_{[t]} \\ \hat{y}_t &= W^{(S)}f(h_t) \end{aligned}$$

- Total error is the sum of each error at time steps t

$$\frac{\partial E}{\partial W} = \sum_{t=1}^T \frac{\partial E_t}{\partial W}$$

- Hardcore chain rule application:

$$\frac{\partial E_t}{\partial W} = \sum_{k=1}^t \frac{\partial E_t}{\partial y_t} \frac{\partial y_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} \frac{\partial h_k}{\partial W}$$

The vanishing gradient problem

- Useful for analysis we will look at:

$$\frac{\partial E_t}{\partial W} = \sum_{k=1}^t \frac{\partial E_t}{\partial y_t} \frac{\partial y_t}{\partial h_t} \boxed{\frac{\partial h_t}{\partial h_k}} \frac{\partial h_k}{\partial W}$$

- Remember

$$h_t = Wf(h_{t-1}) + W^{(hx)}x_{[t]}$$

- More chain rule, remember:

$$\frac{\partial h_t}{\partial h_k} = \prod_{j=k+1}^t \frac{\partial h_j}{\partial h_{j-1}}$$

- The gradient is a product of Jacobian matrices, each associated with a step in the forward computation.

$$\left\| \frac{\partial h_t}{\partial h_k} \right\| = \left\| \prod_{j=k+1}^t \frac{\partial h_j}{\partial h_{j-1}} \right\| \leq (\beta_W \beta_h)^{t-k}$$



Vanishing or exploding gradient

The vanishing gradient problem for language models

- In the case of language modeling or question answering words from time steps far away are not taken into consideration when training to predict the next word
- **Example:**

*Jane walked into the room. John walked in too. It was late in the day.
John said hi to _____*

Vanishing/Exploding Solutions

- **Vanishing Gradient:**

- Gating mechanism (LSTM, GRU)
- Attention mechanism (Transformer)
- Adding skip connection through time (Residual Network)
- Better Initialization

Long Short-Term Memory RNNs (LSTMs)

- A type of RNN proposed by **Hochreiter** and **Schmidhuber** in 1997 as a solution to the **vanishing gradients problem**.

LONG SHORT-TERM MEMORY

NEURAL COMPUTATION 9(8):1735–1780, 1997

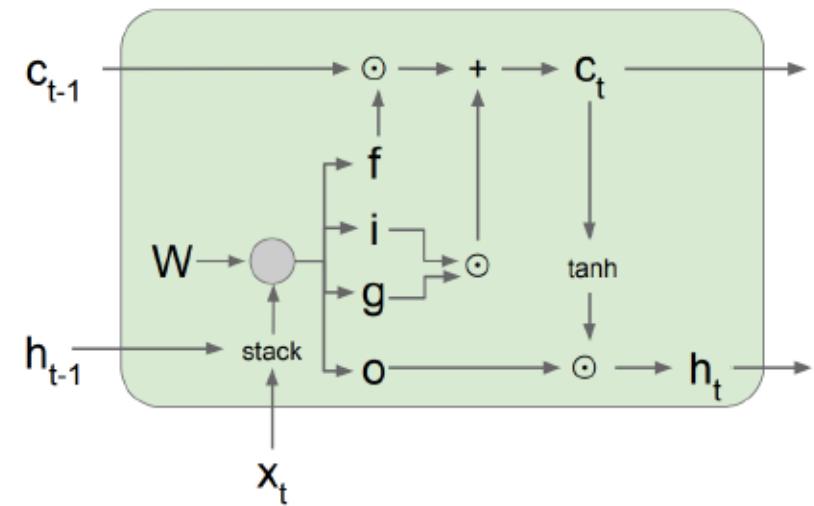
Sepp Hochreiter
Fakultät für Informatik
Technische Universität München
80290 München, Germany
hochreit@informatik.tu-muenchen.de
<http://www7.informatik.tu-muenchen.de/~hochreit>

Jürgen Schmidhuber
IDSIA
Corso Elvezia 36
6900 Lugano, Switzerland
juergen@idsia.ch
<http://www.idsia.ch/~juergen>

LSTMs: The intuition

- **Key idea:** turning **multiplication** into **addition** and using “**gates**” to control how much information to add/erase
- At each time step, instead of re-writing the hidden state $\mathbf{h}_t = g(\mathbf{W}\mathbf{h}_{t-1} + \mathbf{U}\mathbf{x}_t + \mathbf{b})$, there is also a cell state $\mathbf{c}_t \in \mathbb{R}^h$ which stores **long-term information**
 - We write to/erase information from \mathbf{c}_t after each step
 - We read \mathbf{h}_t from \mathbf{c}_t

Example: The flights the airline *was canceling were full.*



LSTMs: the formulation

- Input gate (**how much to write**):

$$\mathbf{i}_t = \sigma(\mathbf{W}^i \mathbf{h}_{t-1} + \mathbf{U}^i \mathbf{x}_t + \mathbf{b}^i) \in \mathbb{R}^h$$

- Forget gate (**how much to erase**):

$$\mathbf{f}_t = \sigma(\mathbf{W}^f \mathbf{h}_{t-1} + \mathbf{U}^f \mathbf{x}_t + \mathbf{b}^f) \in \mathbb{R}^h$$

- Output gate (**how much to reveal**):

$$\mathbf{o}_t = \sigma(\mathbf{W}^o \mathbf{h}_{t-1} + \mathbf{U}^o \mathbf{x}_t + \mathbf{b}^{(o)}) \in \mathbb{R}^h$$

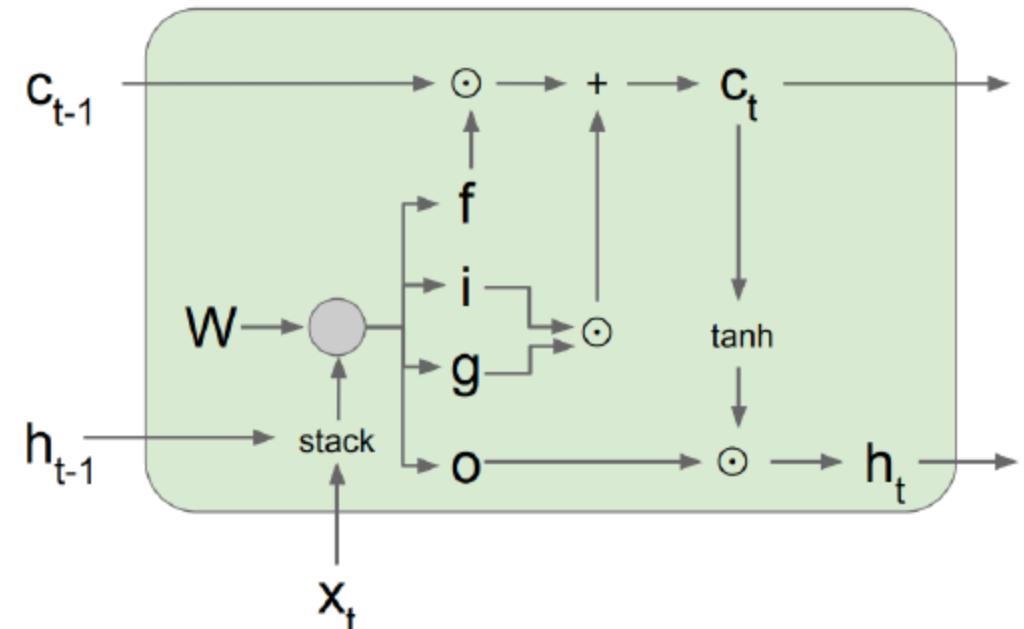
- New memory cell (**what to write**):

$$\mathbf{g}_t = \tanh(\mathbf{W}^g \mathbf{h}_{t-1} + \mathbf{U}^g \mathbf{x}_t + \mathbf{b}^g) \in \mathbb{R}^h$$

- Final memory cell: $\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \mathbf{g}_t$

element-wise product

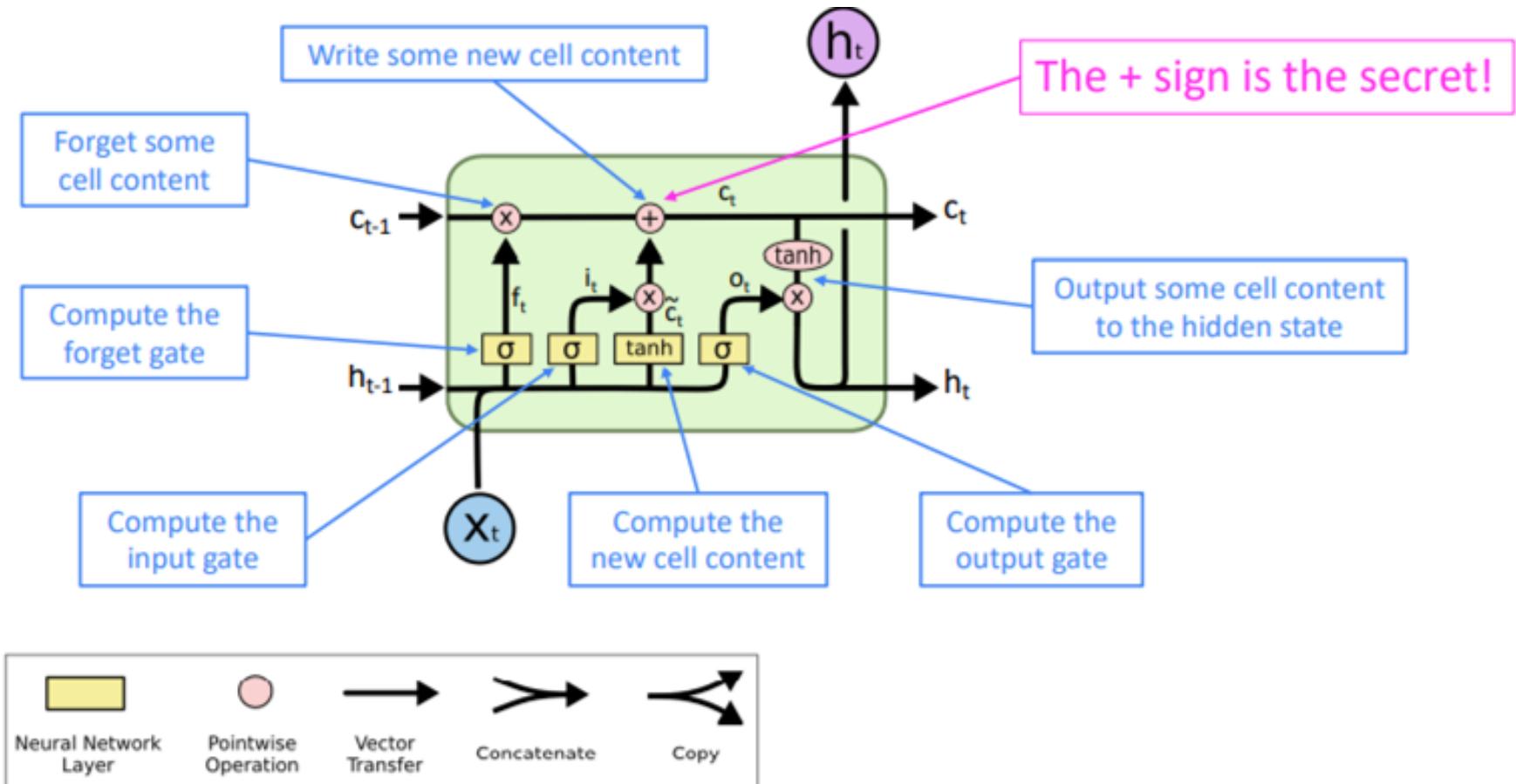
- Final hidden cell: $\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t)$



$\mathbf{h}_0, \mathbf{c}_0 \in \mathbb{R}^h$ are initial states (usually set to $\mathbf{0}$)

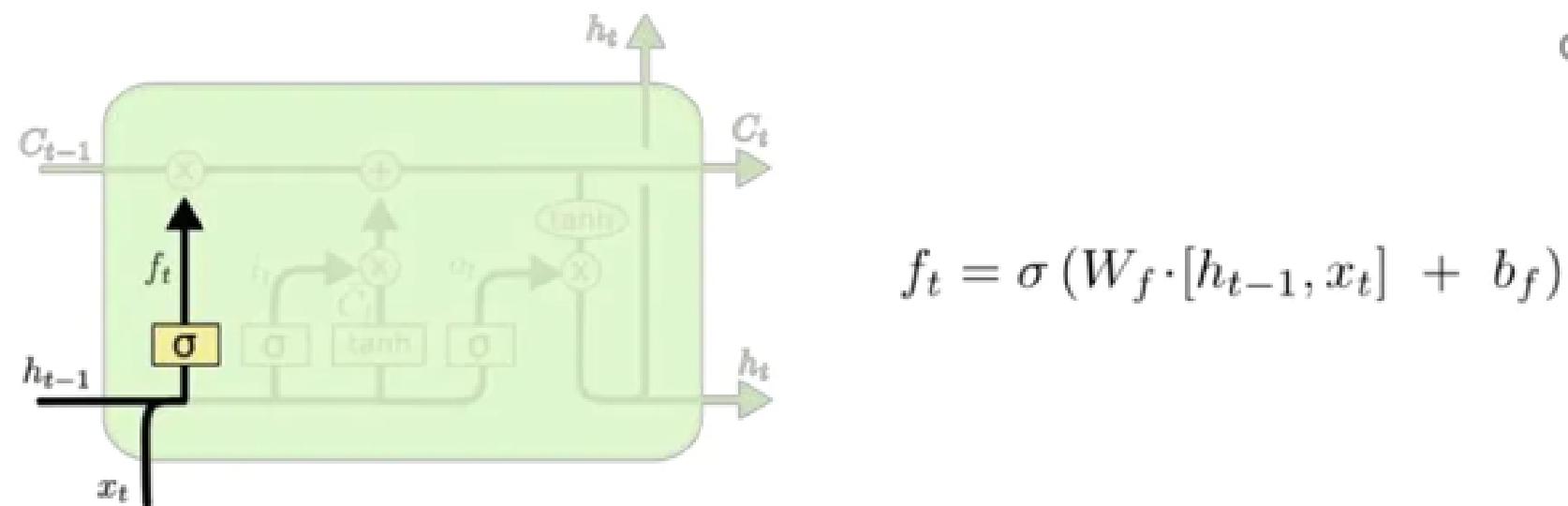
LSTMs

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



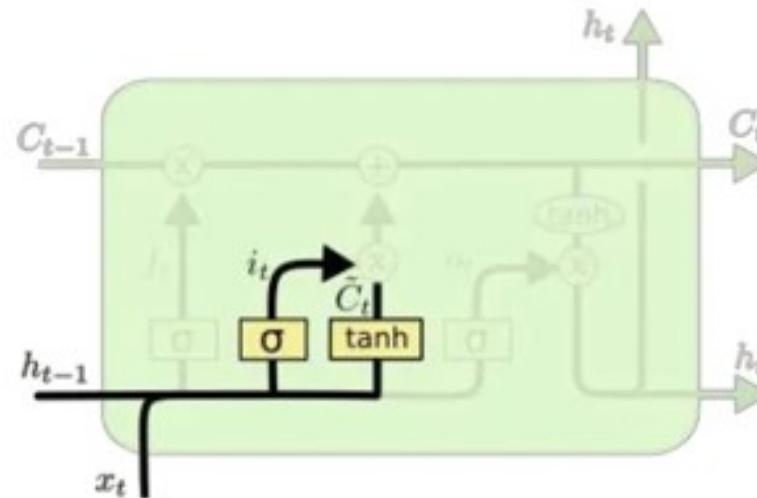
Architecture of LSTM cell

- **Forget information:**
 - Decide what information throw away from the cell state
 - **Forget gate layer:**
 - Output a number between 0 and 1



Architecture of LSTM cell

- **Add new information:**
 - Decide what new information store in the cell state
 - **Input gate layer:**
 - Decides which values we'll update
 - **Tanh layer:**
 - creates a vector of new candidate values, \tilde{C}_t

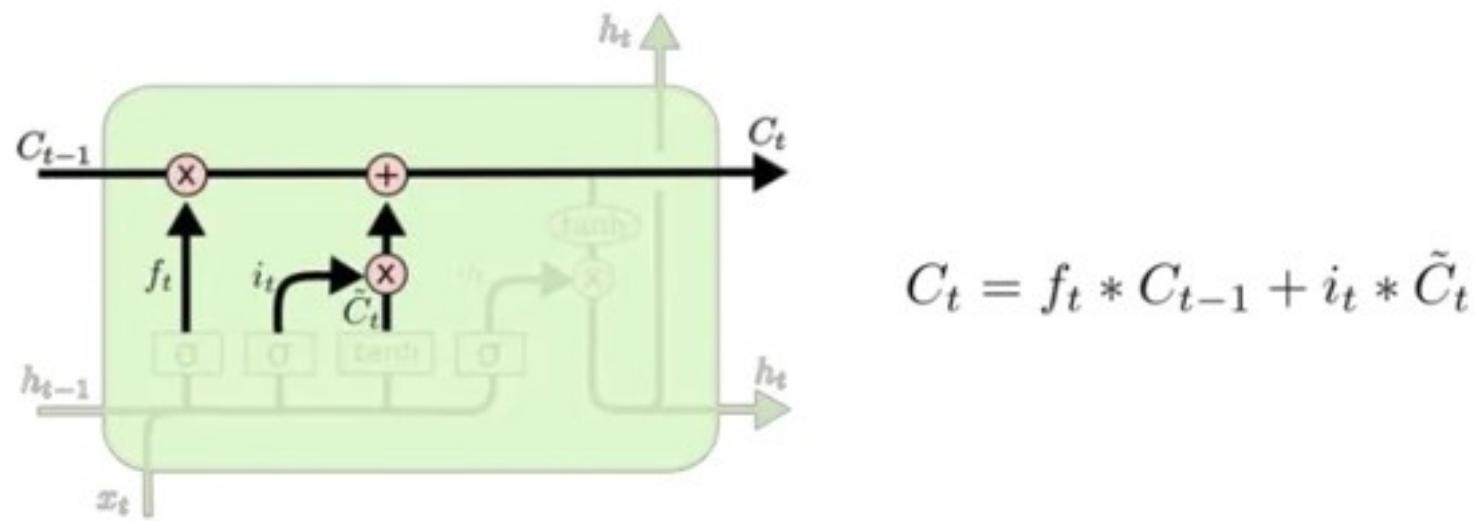


$$i_t = \sigma (W_i \cdot [h_{t-1}, x_t] + b_i)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

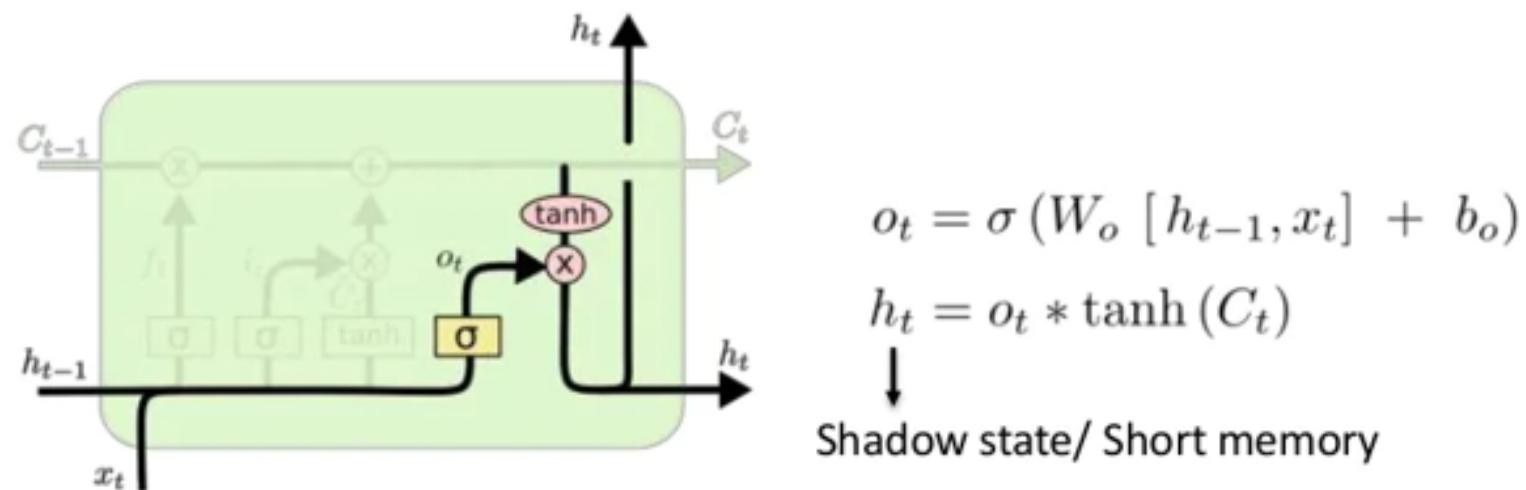
Architecture of LSTM cell

- **Update cell state:**
 - Forgetting the things we decided to forget earlier: $f_t * C_{t-1}$
 - Adding information we decide to add: $i_t * \tilde{C}_t$



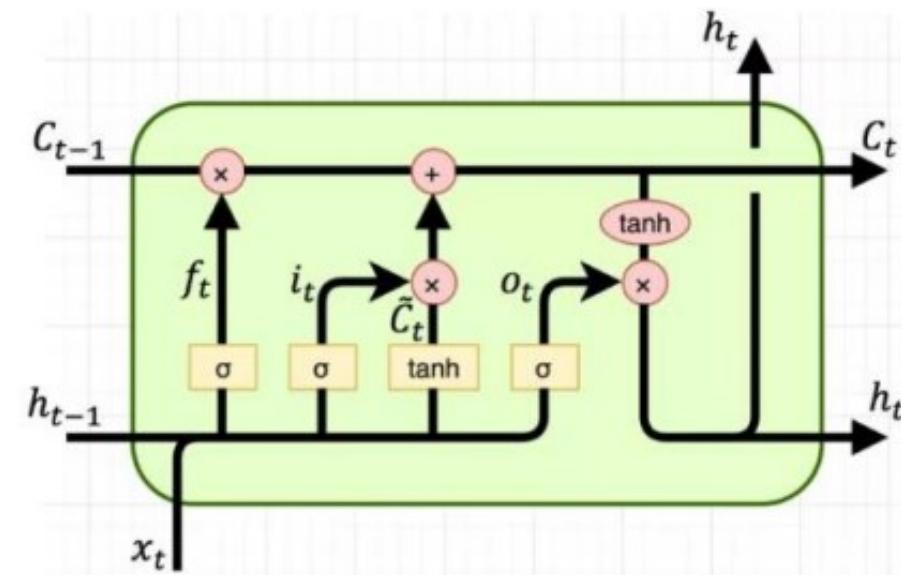
Architecture of LSTM cell

- **Create output:**
 - Decide what we're going to output
 - **Output gate layer:**
 - Decides what parts of the cell state we're going to output
 - **Tanh layer:**
 - Push the values between -1 and +1



Architecture of LSTM cell

- **Conclusion:**
 - Step 1: Forget gate layer.
 - Step 2: Input gate layer.
 - Step 3: Combine step 1 & 2.
 - Step 4: Output the cell state.



Why is LSTM More Efficient than RNN?

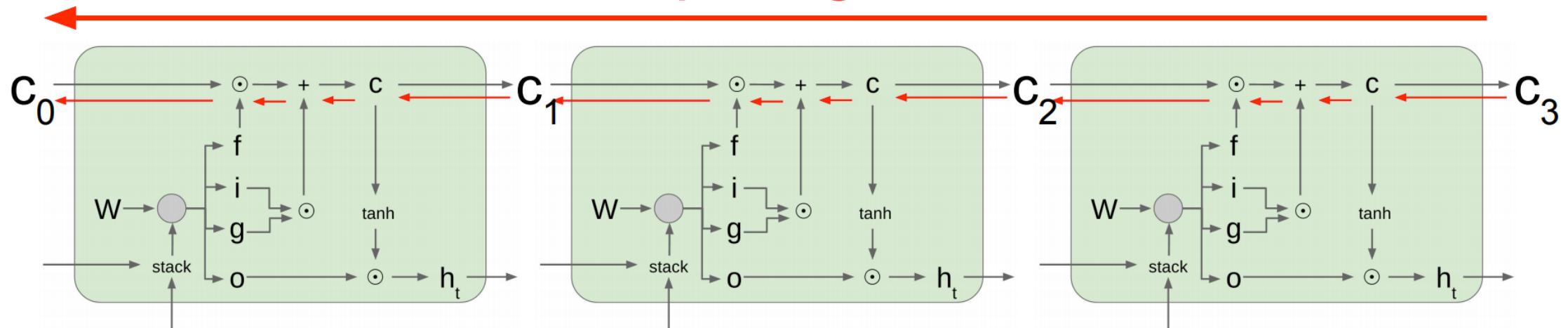
Feature	RNN	LSTM
Memory	Only hidden state (h_t)	Cell state (C_t) + hidden state (h_t)
Information update	Overwrite h_t each step	Selective adjustment for gate
Gradient	Vanishing gradient	Preserved through cell state

How does LSTM can solve vanishing gradient

- The LSTM architecture makes it **easier** for the RNN to **preserve** information **over many timesteps**.
- LSTM ***doesn't guarantee*** that there is **no vanishing/ exploding** gradient.
- LSTM provides an **easier way** for the model to learn **long-distance dependencies**.

How does LSTM can solve vanishing gradient

Uninterrupted gradient flow!

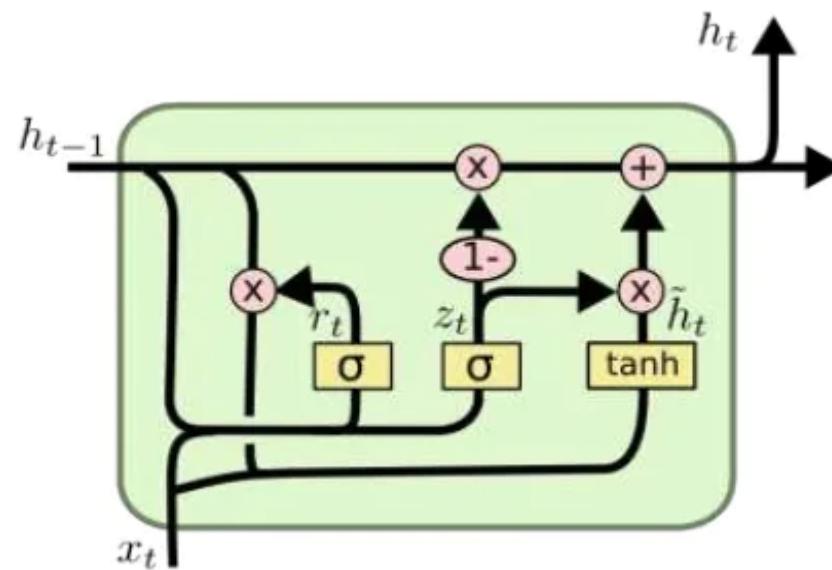


- LSTMs were invented in 1997 but finally got working from 2013-2015.
- These ideas influenced later designs such as “**residual connection**” (**ResNet**).

LSTM Variations (GRU)

- **Gated Recurrent Unit (GRU)**

- Combine the forget and input layer into a single “update gate”
- Merge the cell state and the hidden state
- Simpler.



$$z_t = \sigma (W_z \cdot [h_{t-1}, x_t])$$

$$r_t = \sigma (W_r \cdot [h_{t-1}, x_t])$$

$$\tilde{h}_t = \tanh (W \cdot [r_t * h_{t-1}, x_t])$$

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

Source: Kyunghyun Cho et al, 2014

Compare LSTM vs. GRU

- **GRUs train faster** and perform better than LSTMs on **less training data** if you are doing language modeling (not sure about other tasks).
- **GRUs are simpler** and thus easier to modify, for example adding new gates in case of additional input to the network. It's just less code in general.
- **LSTMs** should in theory **remember longer sequences** than GRUs and outperform them in tasks requiring modeling long-distance relations.

Successful Applications of LSTMs

- Speech recognition: Language and acoustic modeling
- Sequence labeling
 - POS Tagging
[https://www.aclweb.org/aclwiki/index.php?title=POS_Tagging_\(State_of_the_art\)](https://www.aclweb.org/aclwiki/index.php?title=POS_Tagging_(State_of_the_art))
 - NER
 - Phrase Chunking
- Neural syntactic and semantic parsing
- Image captioning: CNN output vector to sequence
- Sequence to Sequence
 - Machine Translation (Sutskever, Vinyals, & Le, 2014)
 - Video Captioning (input sequence of CNN frame outputs)

Summary

- Recurrent Neural Network is one of the best deep NLP model families
- Most important and powerful RNN extensions with LSTMs and GRUs

Homework

- RNN & LSTM for **sentiment analysis**
- Dữ liệu văn bản IMDB: The IMDB movie reviews dataset is a set of 50,000 reviews, half of which are positive and the other half negative
- Compare the results with previous methods (SVM, Logistic Regression)

References

- Speech and Language Processing (3rd ed. draft), chapter 13
- Slide of Stanford NLP course and other documents

Question and Discussion!