COMP 5630/6630:Machine Learning

Lecture 11: Recurrent Neural Networks (RNN), LSTM, GRU

New Topic: Recurrent Neural Networks (RNN)

New Terminologies

- Recurrent Neural Networks (RNNs)

- Recursive Neural Networks
  - General family; think graphs instead of chains

- Types:
  - "Vanilla" RNNs (Elman Networks)
  - Long Short Term Memory (LSTMs)
  - Gated Recurrent Units (GRUs)
  - …

What's wrong with MLPs?

- Problem 1: Can't model sequences
  - Fixed-sized Inputs & Outputs
  - No temporal structure

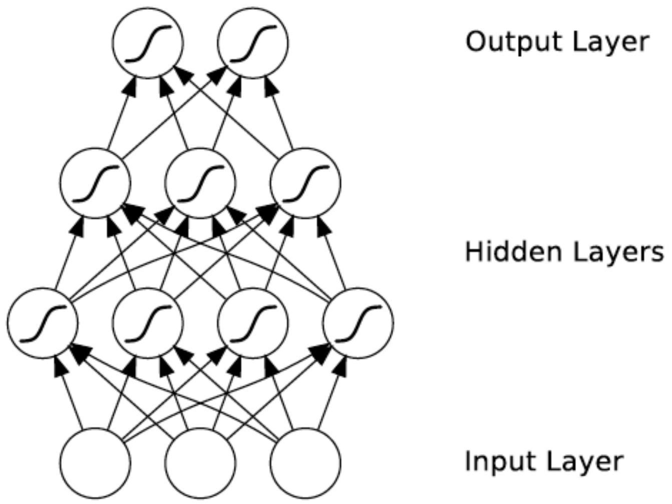


Image Credit: Alex Graves, book

What's wrong with MLPs?

- Problem 1: Can't model sequences
  - Fixed-sized Inputs & Outputs
  - No temporal structure
- Problem 2: Pure feed-forward processing
  - No "memory", no feedback

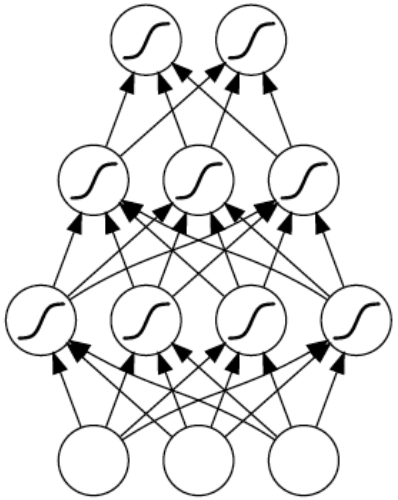
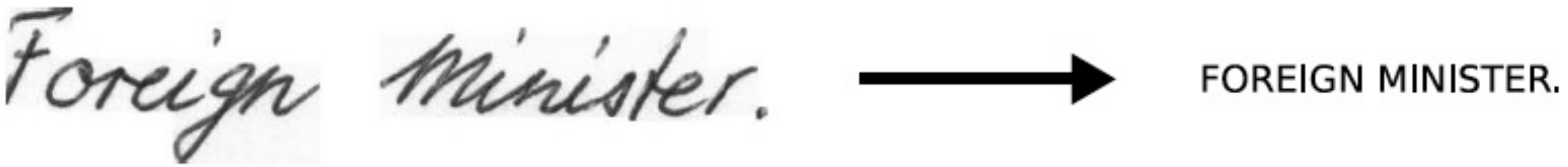


Image Credit: Alex Graves, book

Sequences are everywhere...



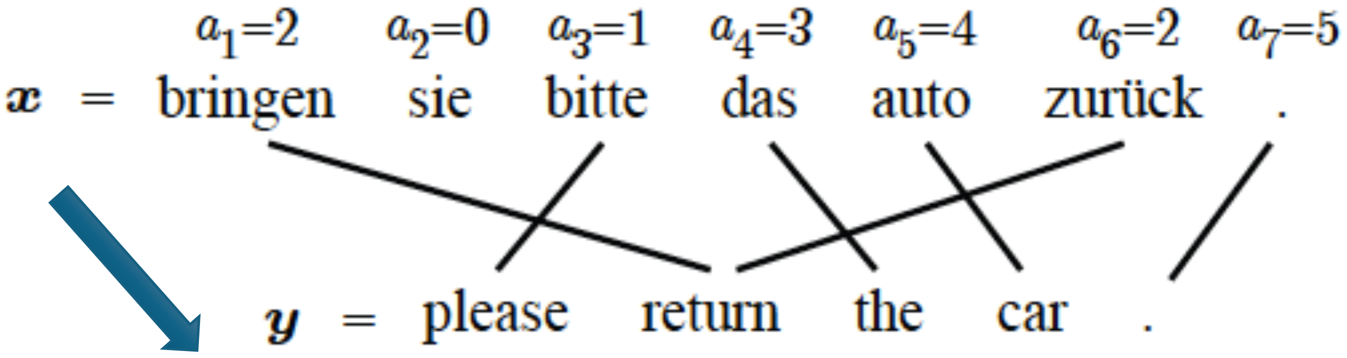
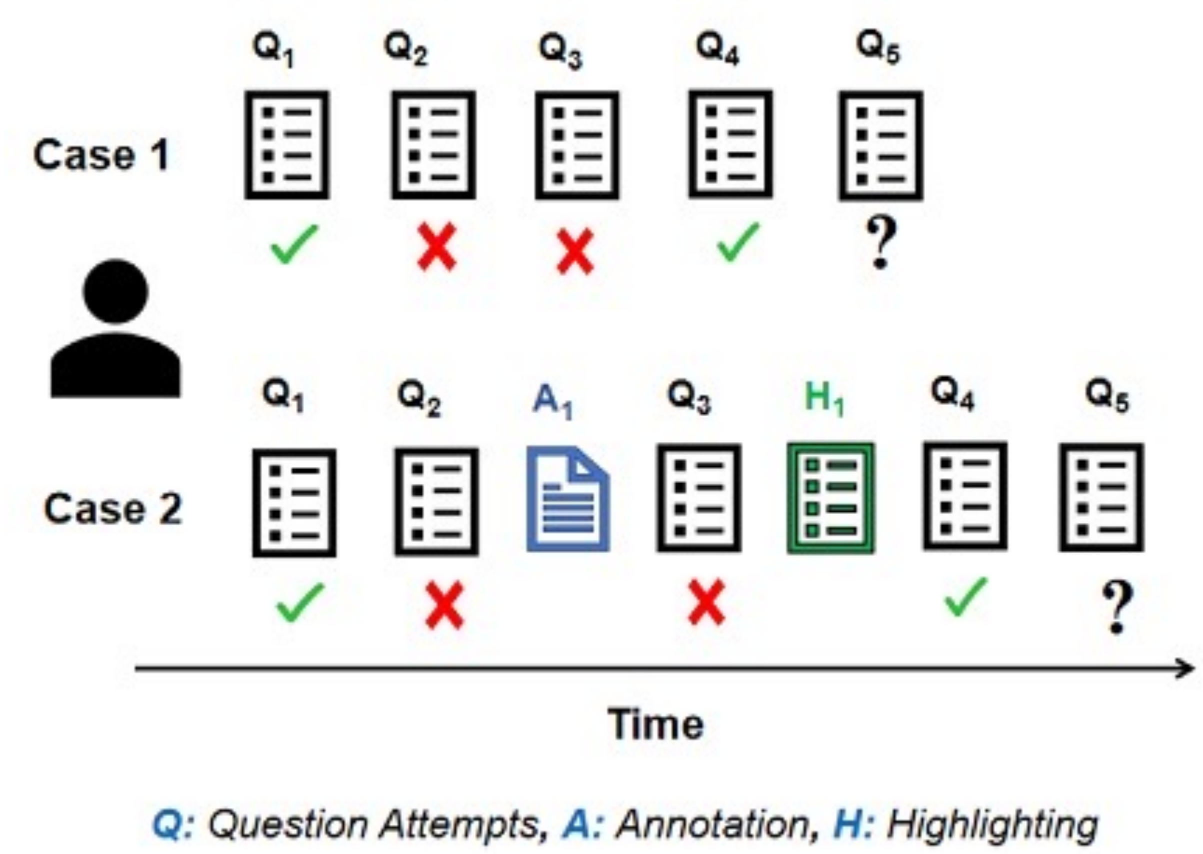


Image Credit: Alex Graves and Kevin Gimpel

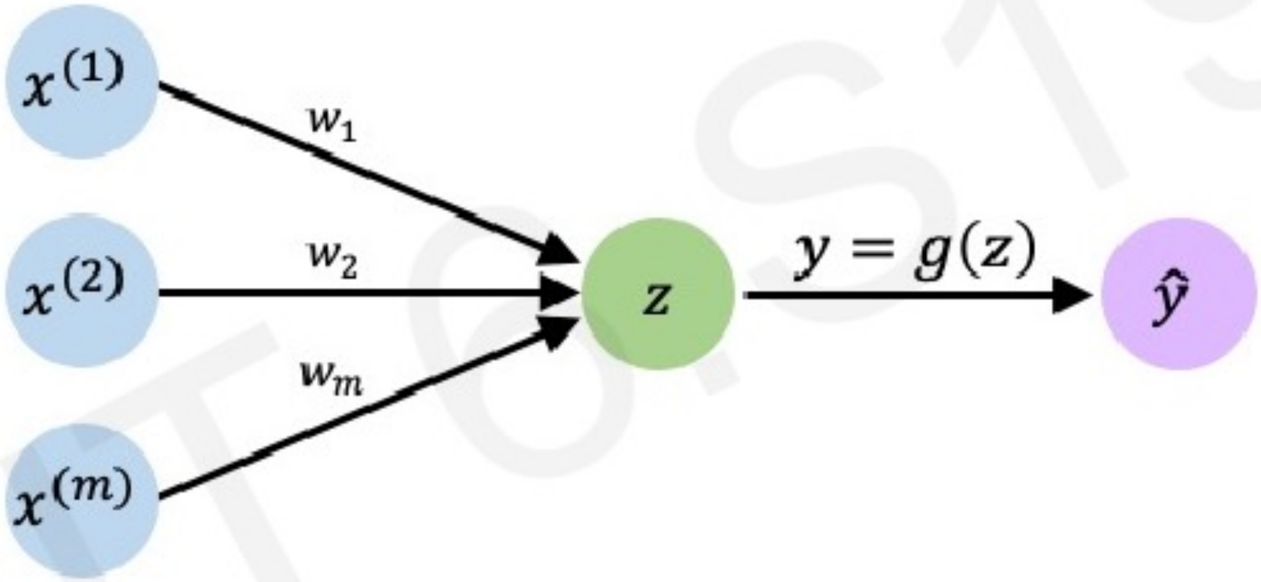
Sequences are everywhere...



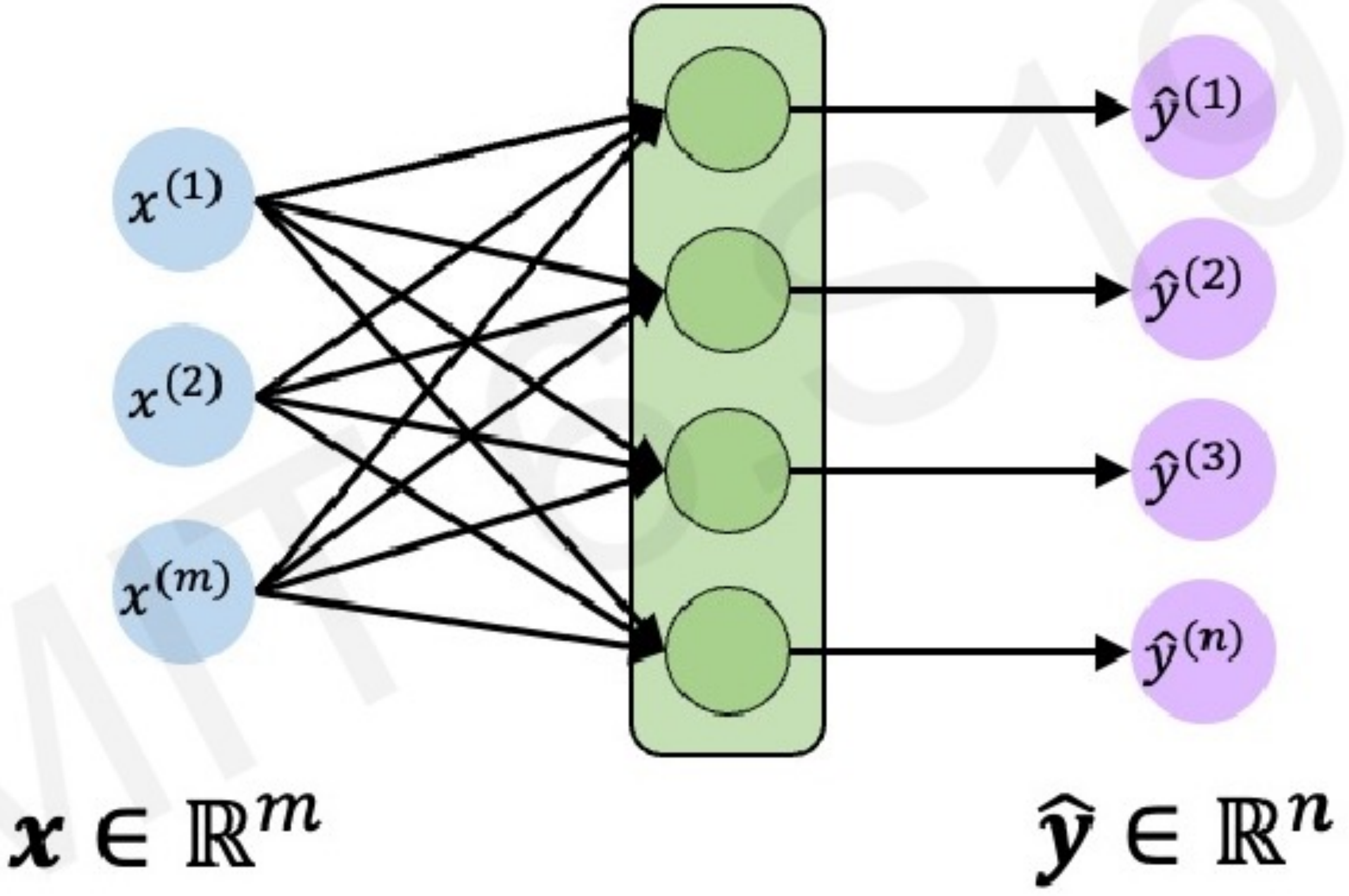
- Predict whether the student will answer correctly or incorrectly to the next question attempt
    - Given the students' prior question answer attempts

Neurons with Recurrence: Intution

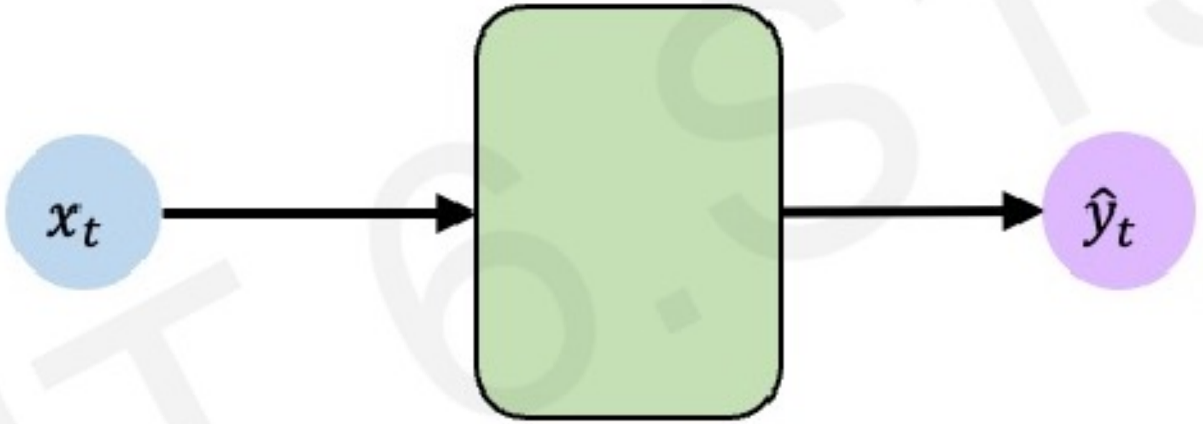
The Perceptron Revisited



Feed-Forward Networks Revisited



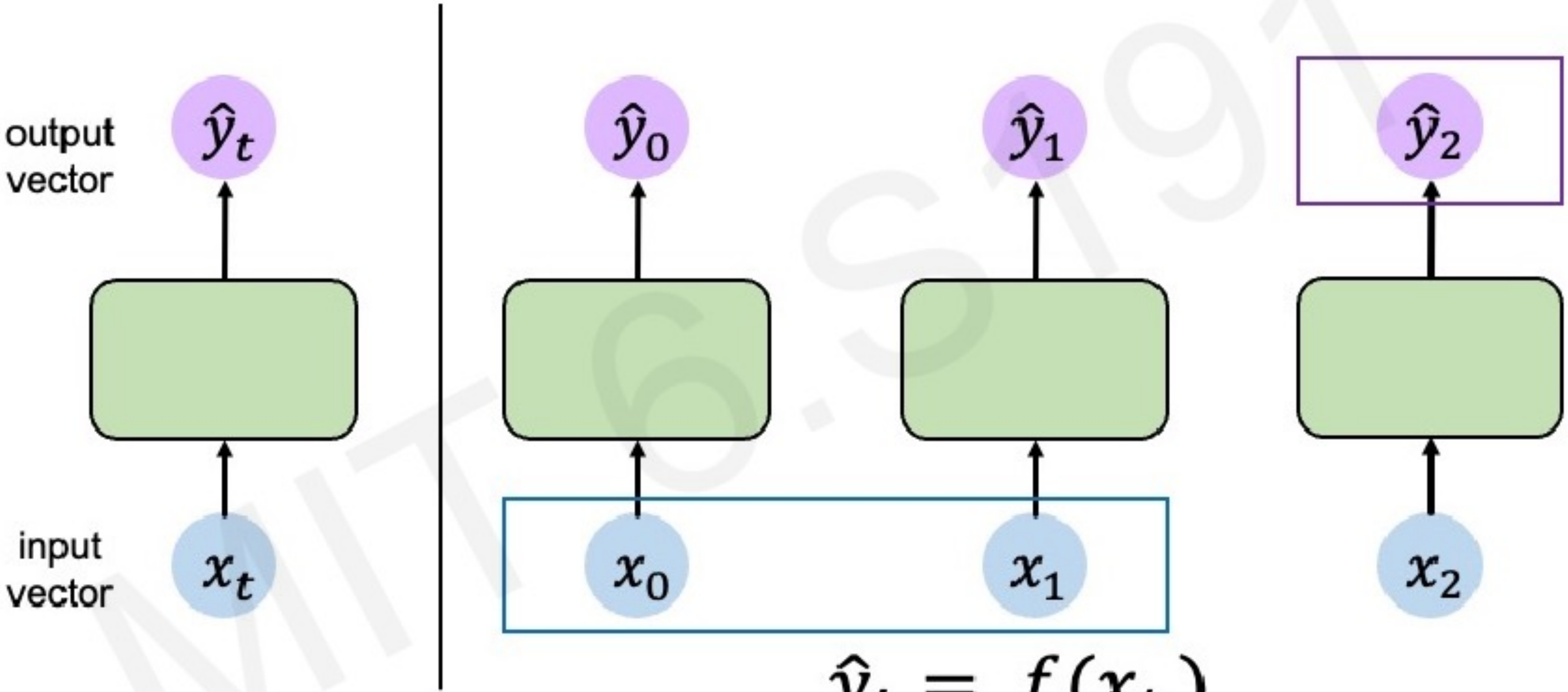
Feed-Forward Networks Revisited



$\boldsymbol{x}_t \in \mathbb{R}^m$

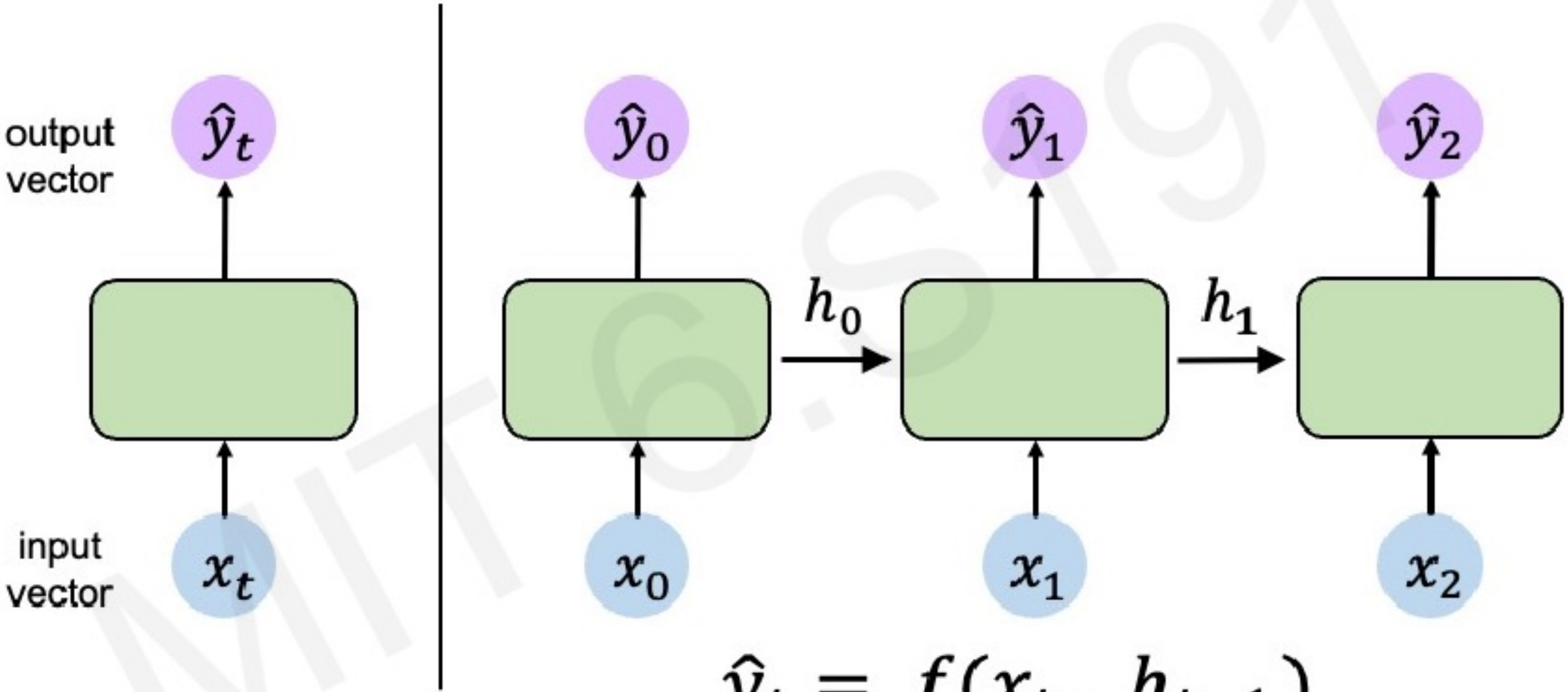
$\hat{\boldsymbol{y}}_t \in \mathbb{R}^n$

Handling Individual Time Steps



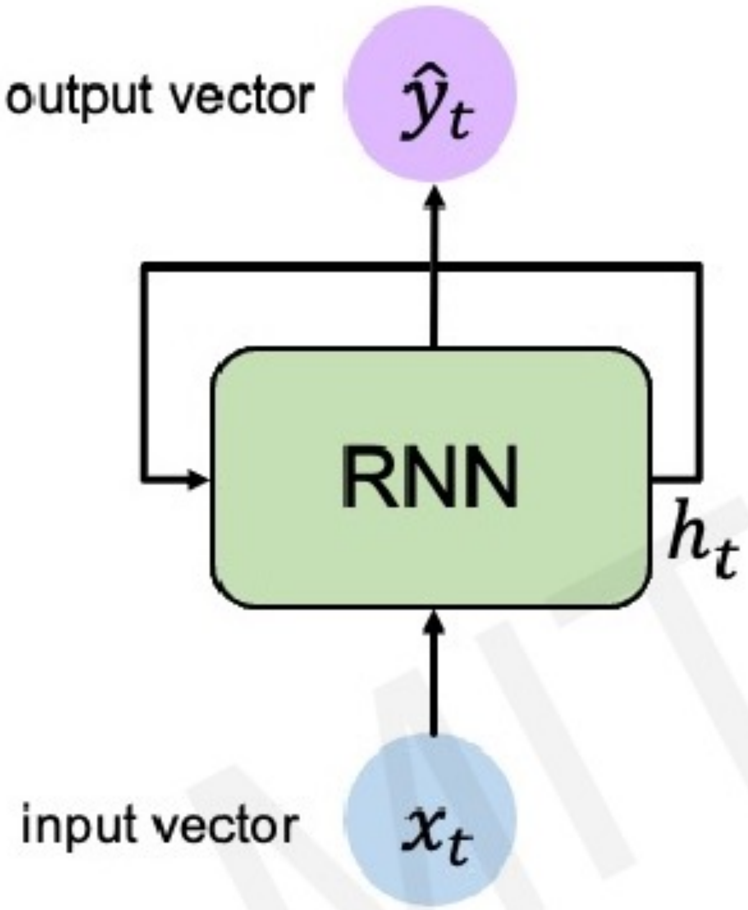
$$\hat{y}_t = f(x_t)$$

Neurons with Recurrence



$$\underbrace{\hat{y}_t}_{\text{output}} = f(\underbrace{x_t}_{\text{input}},\ \underbrace{h_{t-1}}_{\text{past memory}})$$

Recurrent Neural Networks (RNNs)



Apply a **recurrence relation** at every time step to process a sequence:

$$\boxed{h_t} = \boxed{f_W}(\boxed{x_t}, \boxed{h_{t-1}})$$

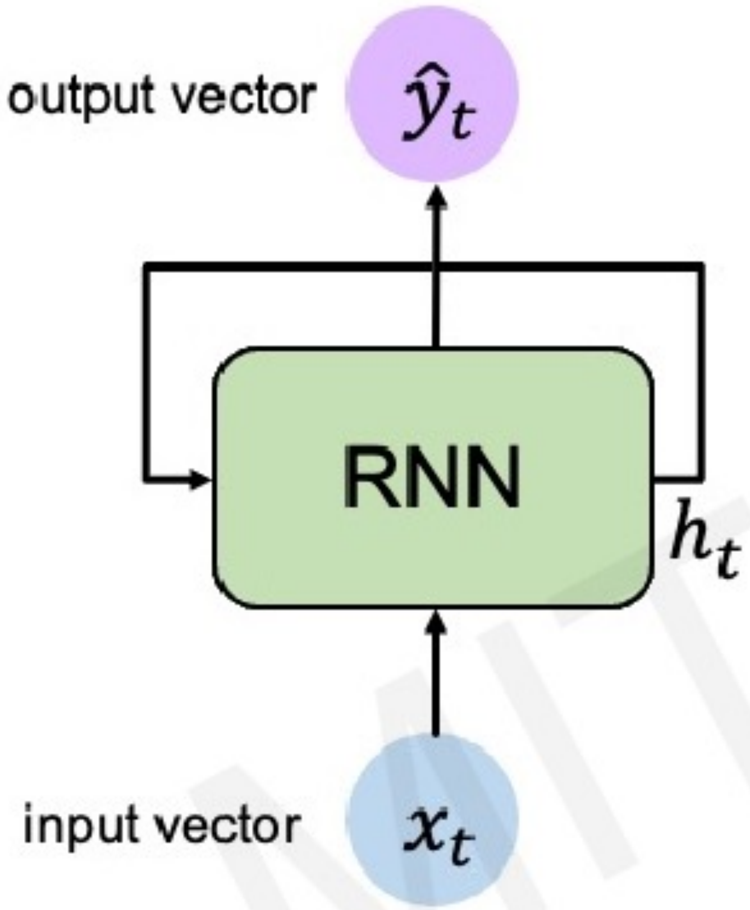
cell state — function with weights W — input — old state

Note: the same function and set of parameters are used at every time step

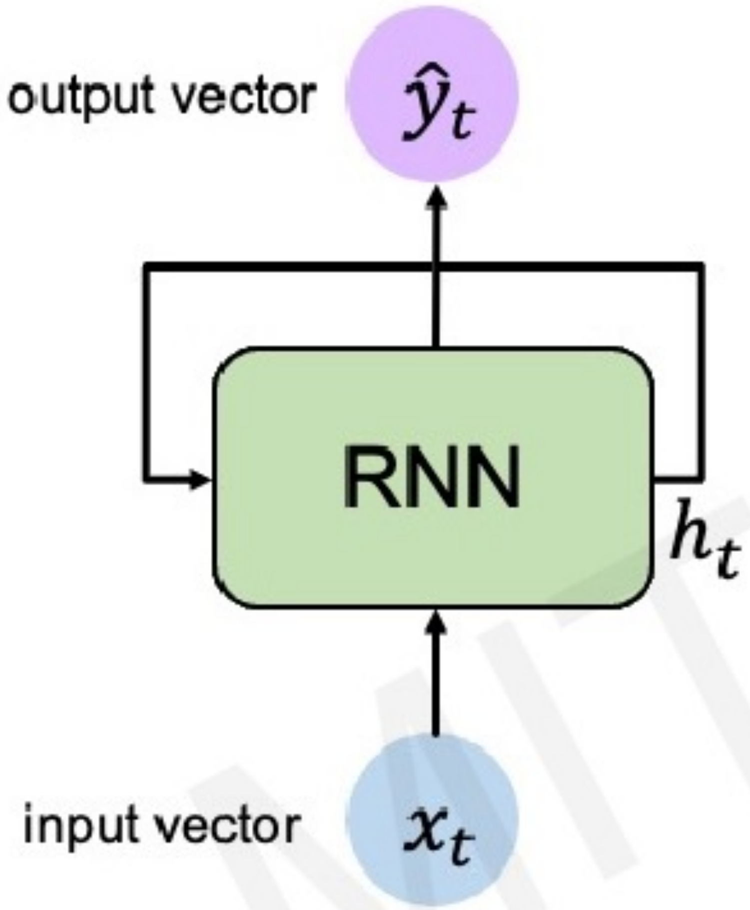
RNNs have a **state**, h_t, that is updated **at each time step** as a sequence is processed

Recurrent Neural Network (RNN)

RNN State Update and Output



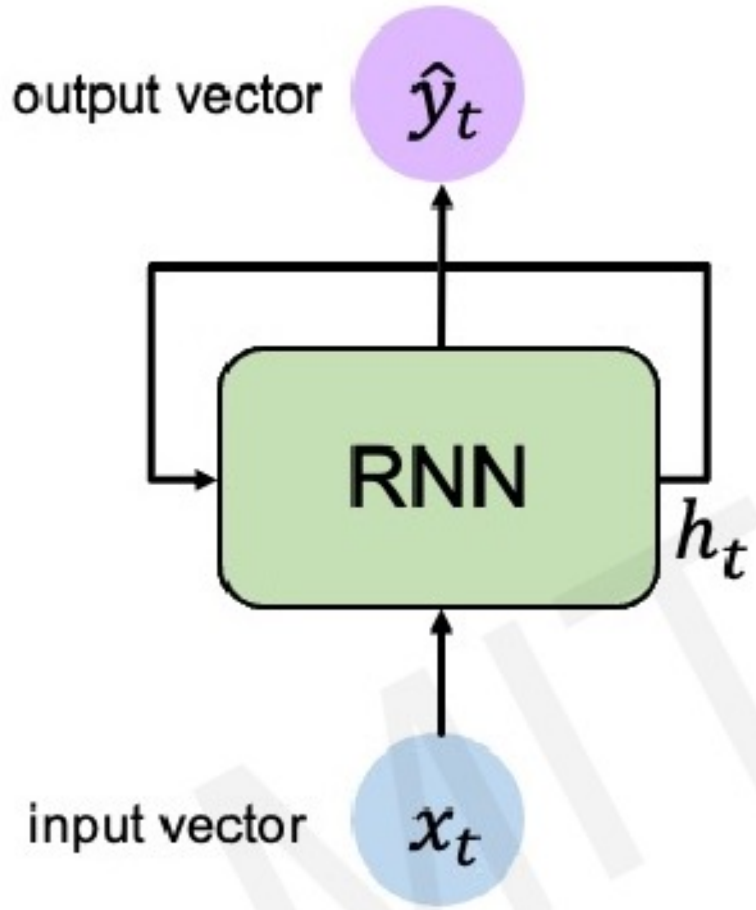
RNN State Update and Output



Input Vector

x_t

RNN State Update and Output



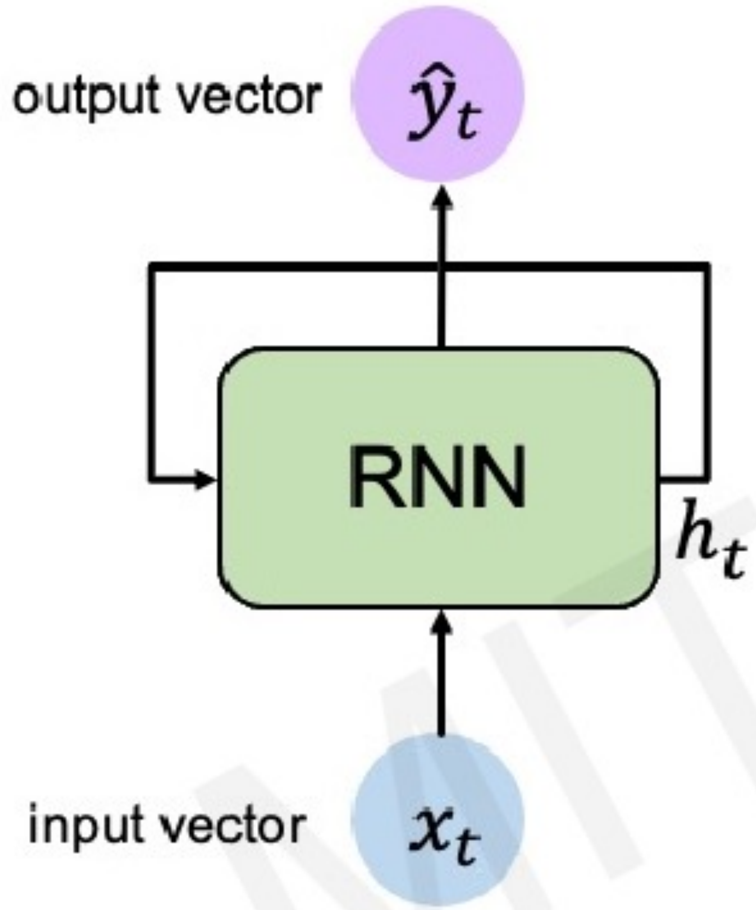
Update Hidden State

$$h_t = \tanh(\boldsymbol{W}_{hh}^T h_{t-1} + \boldsymbol{W}_{xh}^T x_t)$$

Input Vector

$$x_t$$

RNN State Update and Output



Output Vector

$$\hat{y}_t = \boldsymbol{W}_{hy}^T h_t$$

Update Hidden State

$$h_t = \tanh(\boldsymbol{W}_{hh}^T h_{t-1} + \boldsymbol{W}_{xh}^T x_t)$$

Input Vector

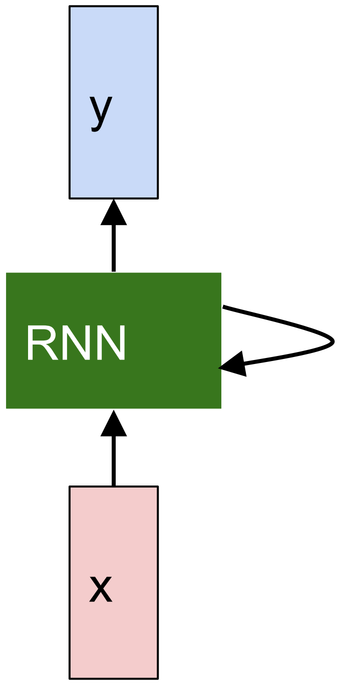
$$x_t$$

Recurrent Neural Network

We can process a sequence of vectors **x** by applying a **recurrence formula** at every time step:

$$h_t = f_W(h_{t-1}, x_t)$$

Notice: the same function and the same set of parameters are used at every time step.

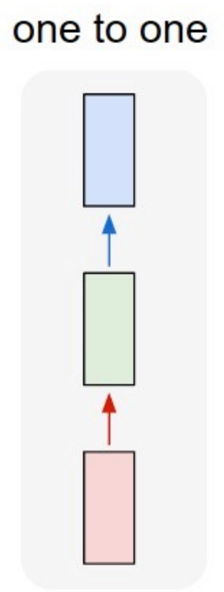


RNN and Sequence Processing

Sequences in Input or Output?

- It’s a spectrum…

Sequences in Input or Output?

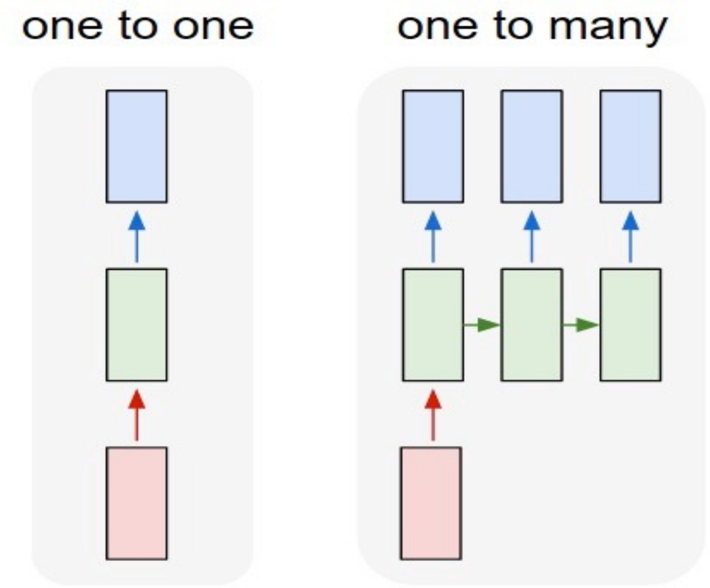


Input: No sequence
Output: No sequence

Example: “standard” classification / regression problems

Image Credit: Andrej Karpathy

Sequences in Input or Output?



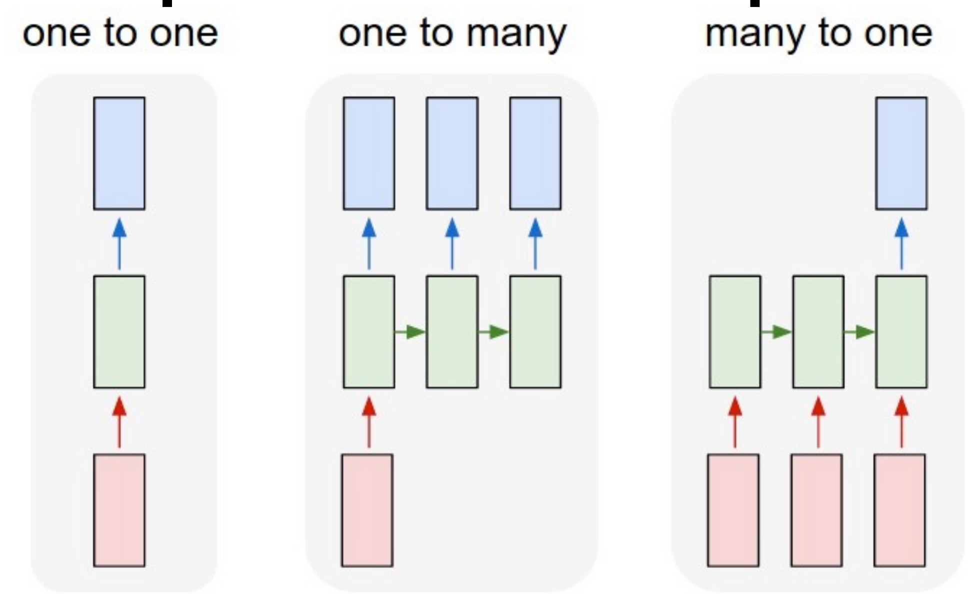
Input: No sequence
Output: No sequence
Example: “standard” classification / regression problems

Input: No sequence
Output: Sequence

Example: Im2Caption

Image Credit: Andrej Karpathy

Sequences in Input or Output?



Input: No sequence
Output: No sequence
Example: “standard” classification / regression problems

Input: No sequence
Output: Sequence

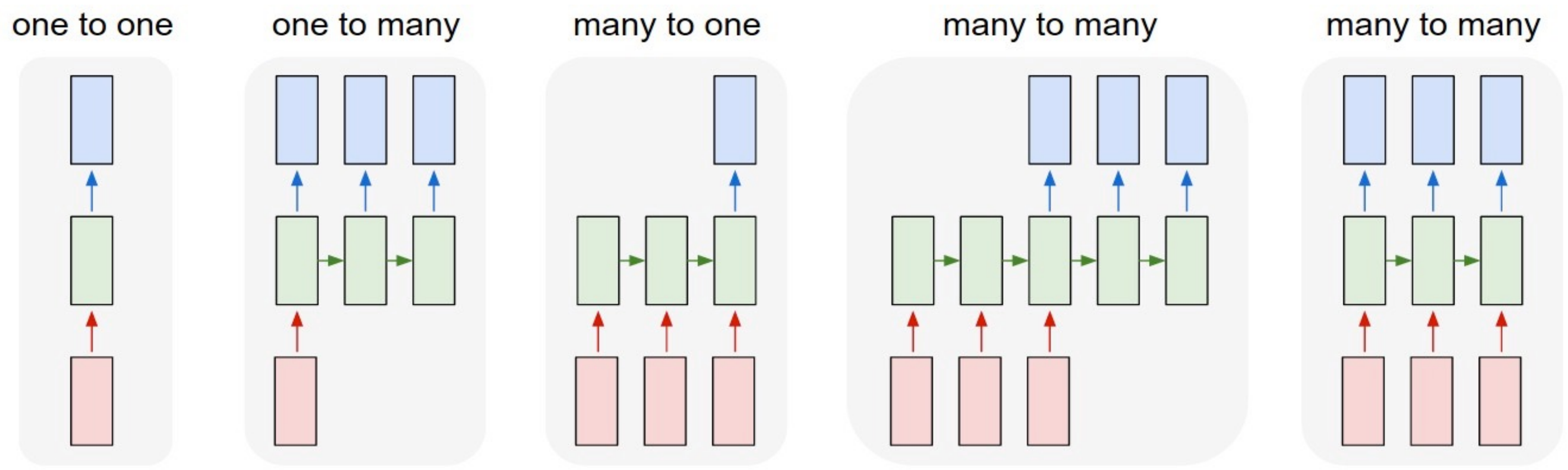
Example: Im2Caption

Input: Sequence
Output: No sequence

Example: sentence classification, MCQ answering

Image Credit: Andrej Karpathy

Sequences in Input or Output?



Input: No sequence
Output: No sequence
Example: "standard" classification / regression problems

Input: No sequence
Output: Sequence

Example: Im2Caption

Input: Sequence
Output: No sequence

Example: sentence classification, MCQ answering

Input: Sequence
Output: Sequence

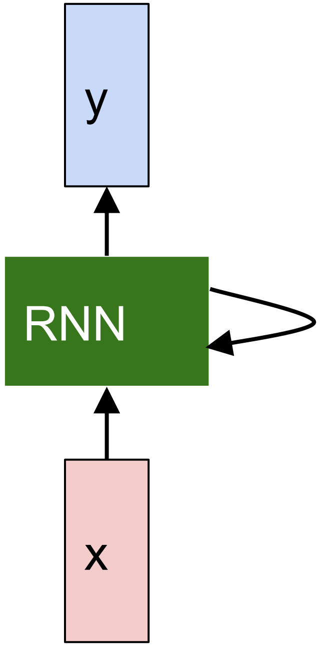
Example: machine translation, video classification,
video captioning, open-ended question answering

Image Credit: Andrej Karpathy

RNN: 2 Key Ideas

- Parameter Sharing
  - in computation graphs = adding gradients
- “Unrolling”
  - in computation graphs with parameter sharing
- Parameter sharing + Unrolling
  - Allows modeling arbitrary sequence lengths!
  - Keeps numbers of parameters in check

(Vanilla) Recurrent Neural Network



The state consists of a single *"hidden"* vector **h**:

$$y_t = W_{hy} h_t + b_y$$

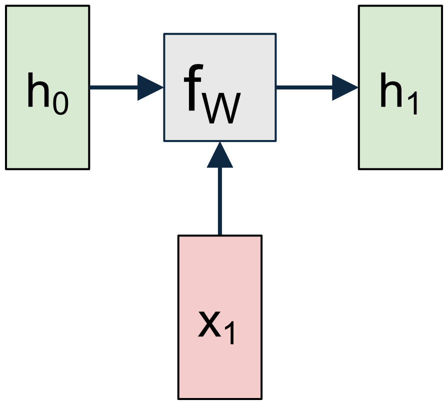
$$h_t = f_W(h_{t-1}, x_t)$$

$$\downarrow$$

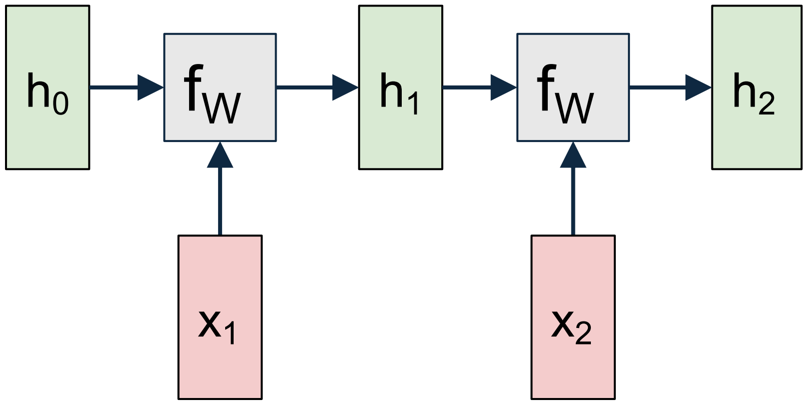
$$h_t = \tanh(W_{hh} h_{t-1} + W_{xh} x_t + b_h)$$

Sometimes called a "Vanilla RNN" or an "Elman RNN" after Prof. Jeffrey Elman

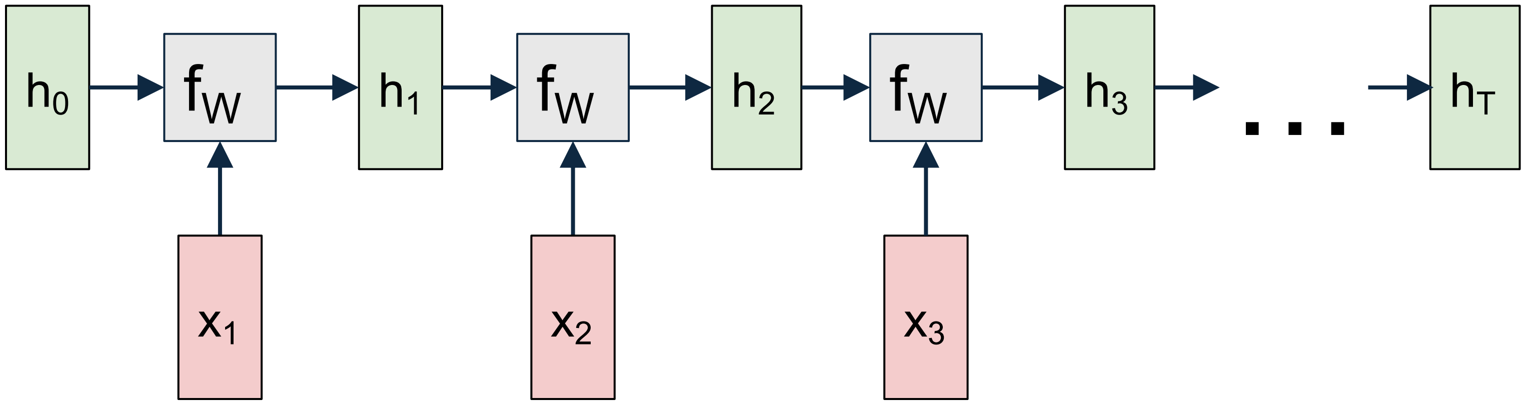
RNN: Computational Graph



RNN: Computational Graph

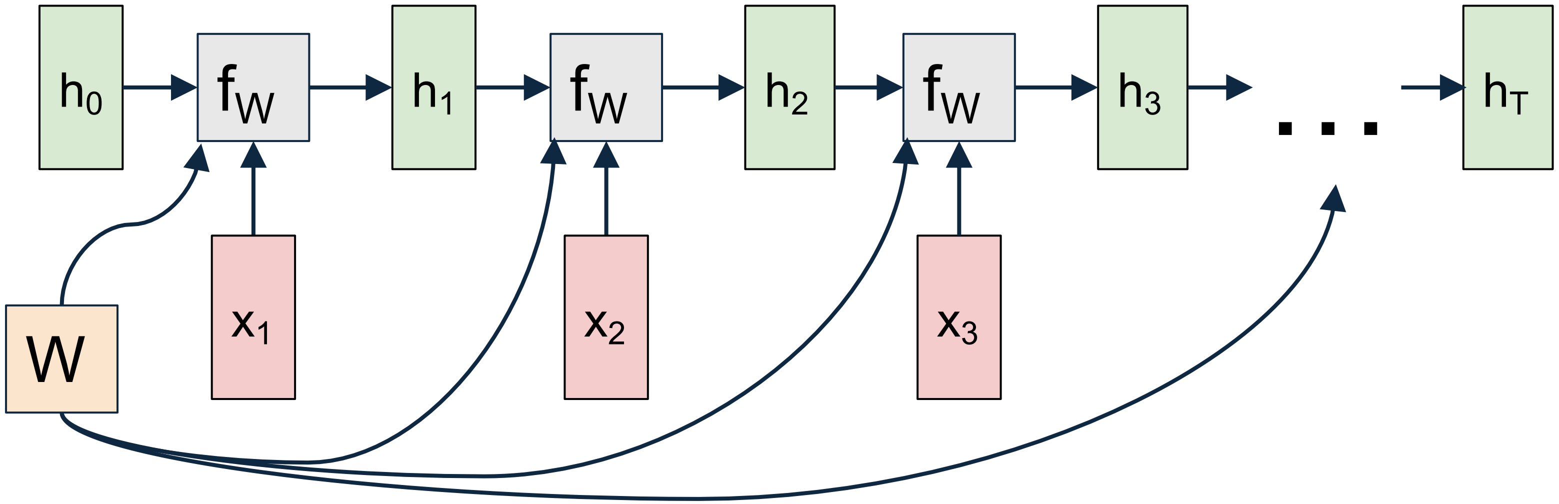


RNN: Computational Graph

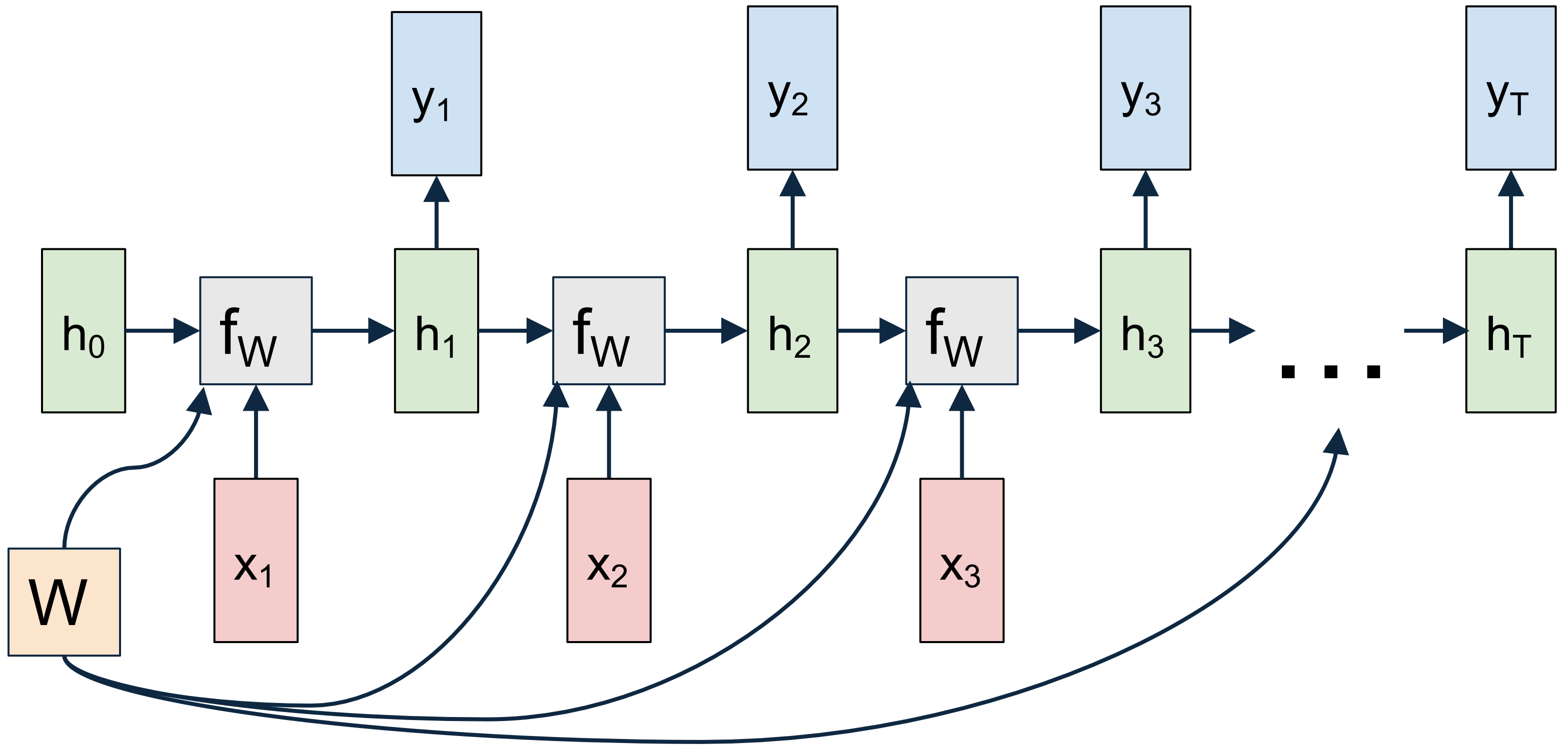


RNN: Computational Graph

Re-use the same weight matrix at every time-step

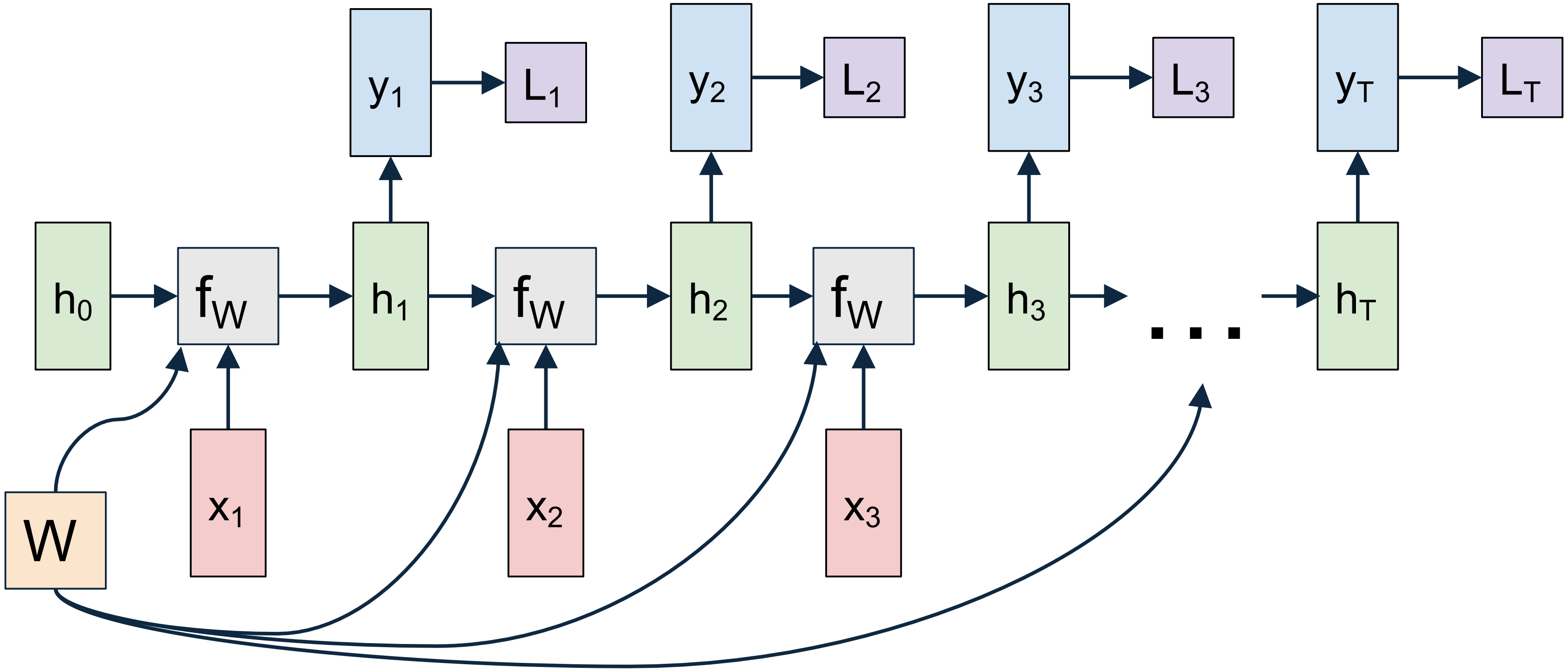


RNN: Computational Graph: Many to Many



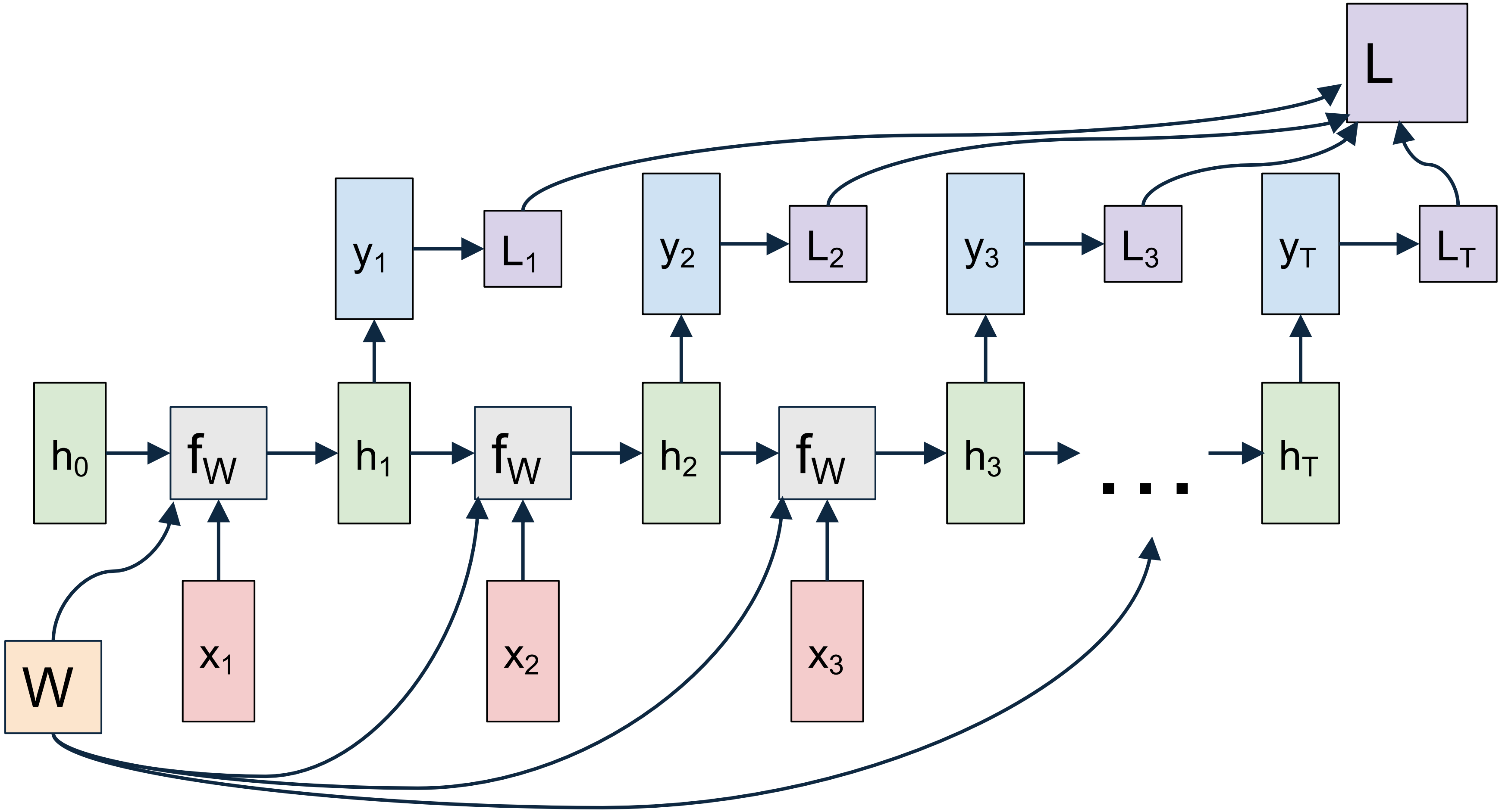
Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

RNN: Computational Graph: Many to Many



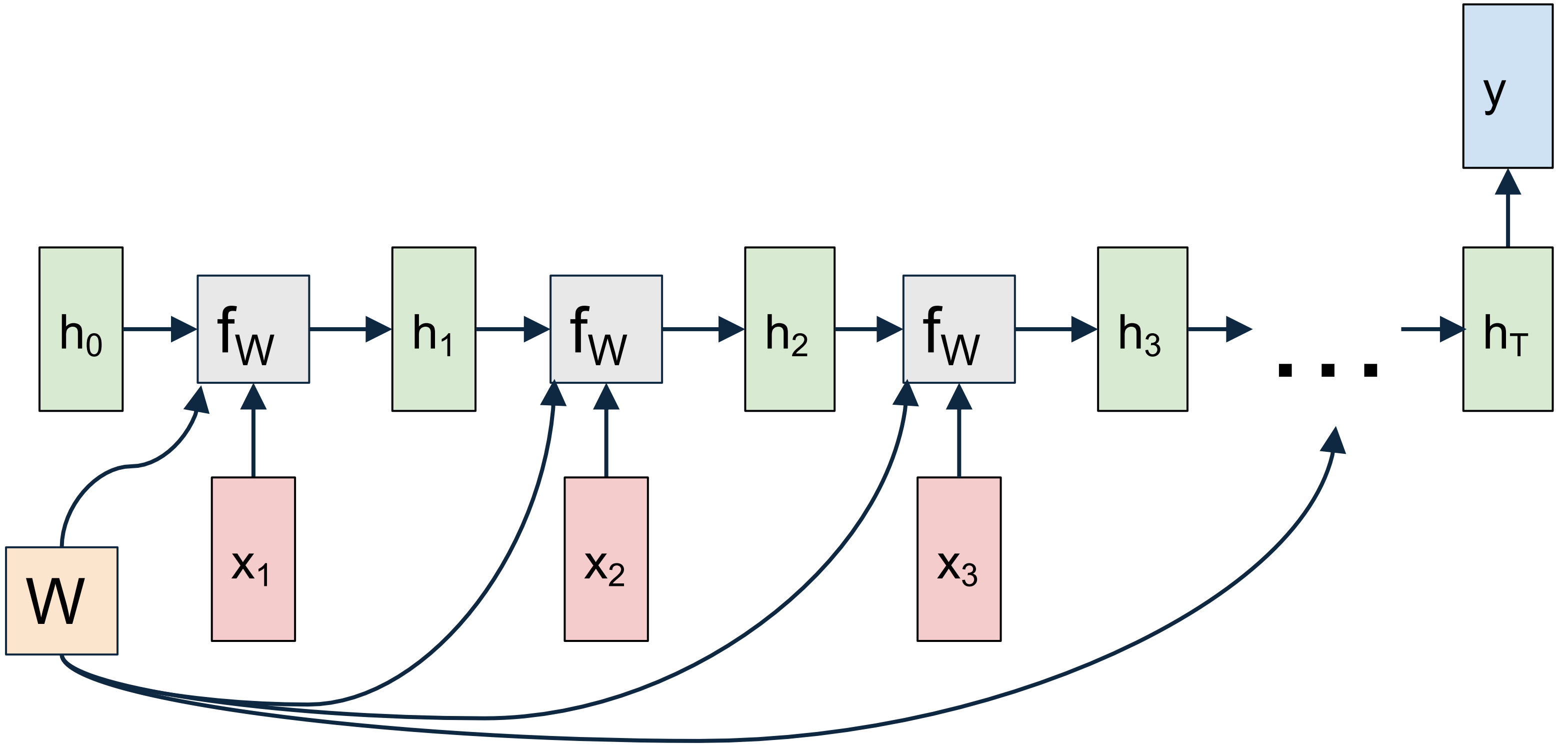
Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

RNN: Computational Graph: Many to Many

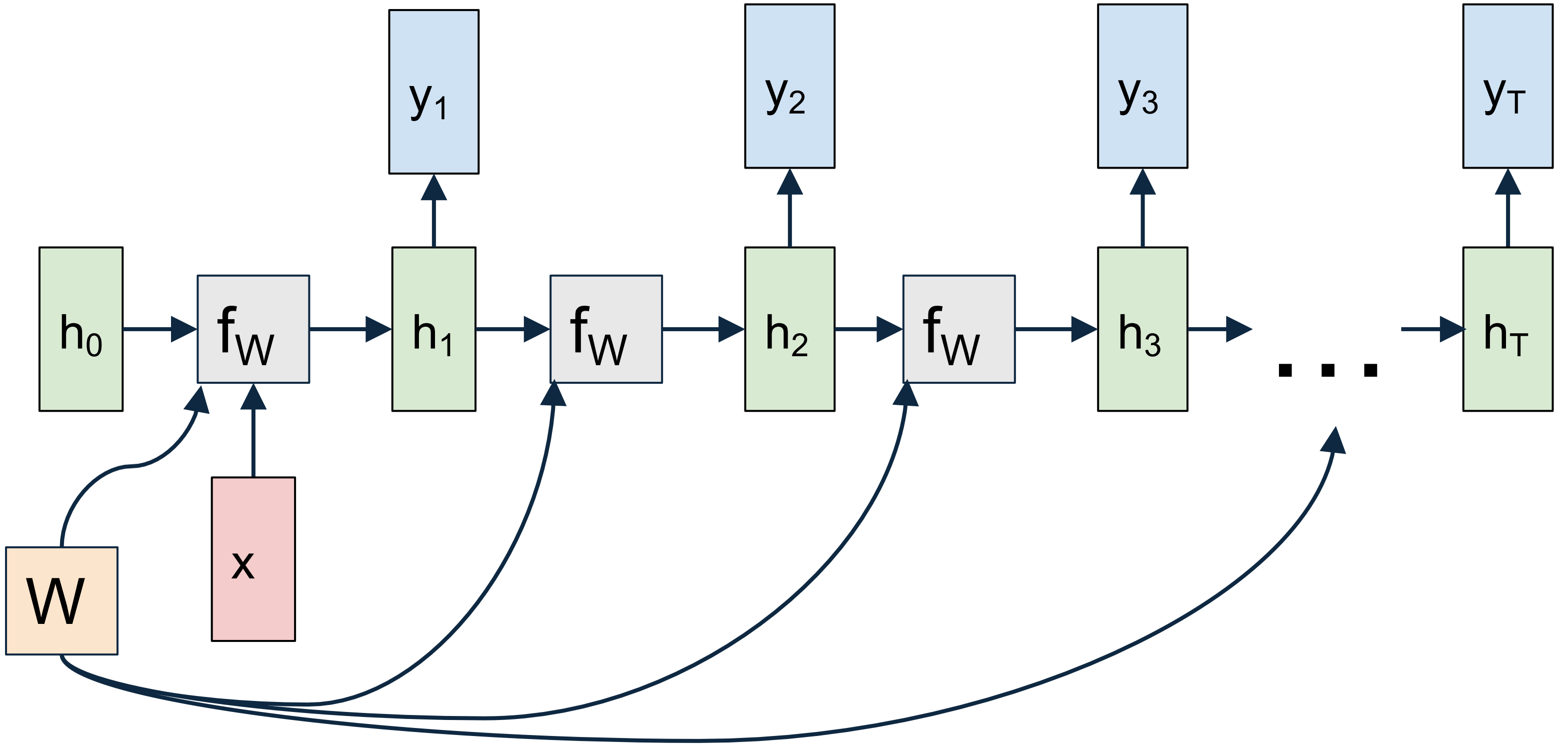


Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

RNN: Computational Graph: Many to One

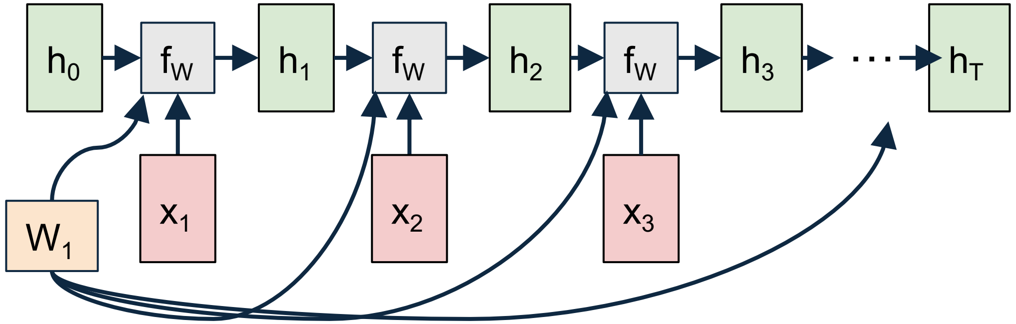


RNN: Computational Graph: One to Many



Sequence to Sequence: Many-to-one + one-to-many

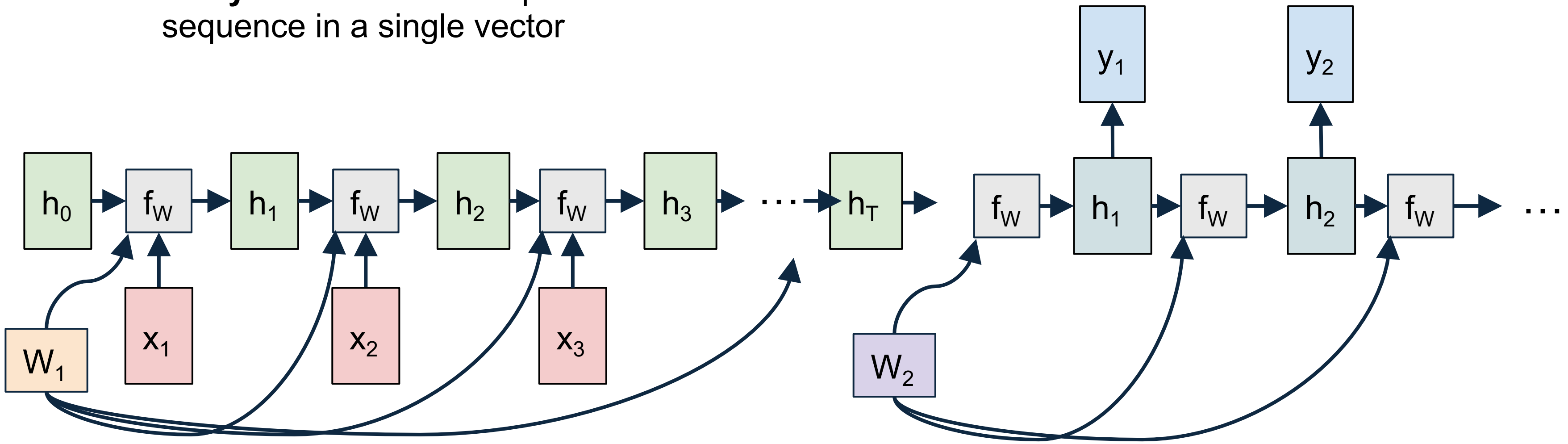
Many to one: Encode input sequence in a single vector



Sequence to Sequence: Many-to-one + one-to-many

One to many: Produce output sequence from single input vector

Many to one: Encode input sequence in a single vector



Advantages	Drawbacks
• Possibility of processing input of any length • Model size not increasing with size of input • Computation takes into account historical information • Weights are shared across time	• Computation being slow • Difficulty of accessing information from a long time ago • Cannot consider any future input for the current state

https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks

❒ **Loss function** — In the case of a recurrent neural network, the loss function $\mathcal{L}$ of all time steps is defined based on the loss at every time step as follows:

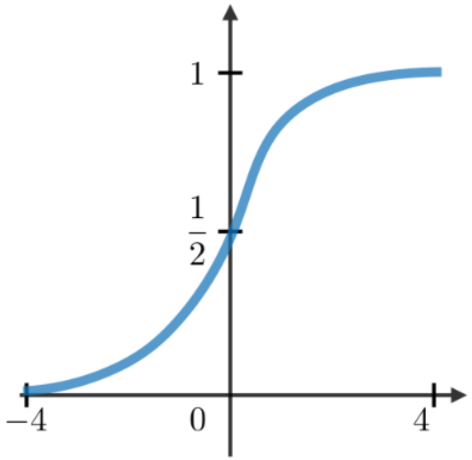
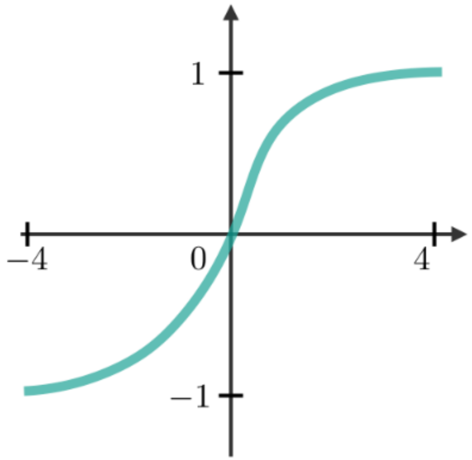
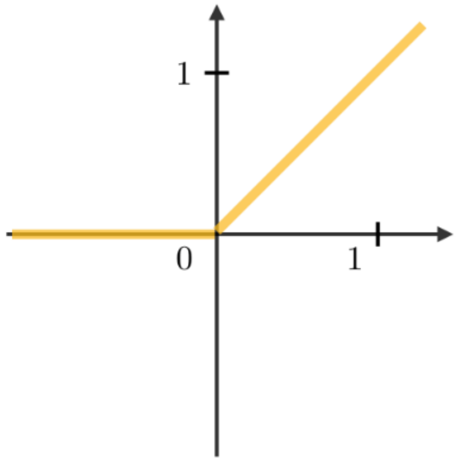
$$\boxed{\mathcal{L}(\widehat{y},y)=\sum_{t=1}^{T_y}\mathcal{L}(\widehat{y}^{<t>},y^{<t>})}$$

❒ **Backpropagation through time** — Backpropagation is done at each point in time. At timestep T, the derivative of the loss $\mathcal{L}$ with respect to weight matrix W is expressed as follows:

$$\boxed{\frac{\partial\mathcal{L}^{(T)}}{\partial W}=\sum_{t=1}^{T}\left.\frac{\partial\mathcal{L}^{(T)}}{\partial W}\right|_{(t)}}$$

https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks

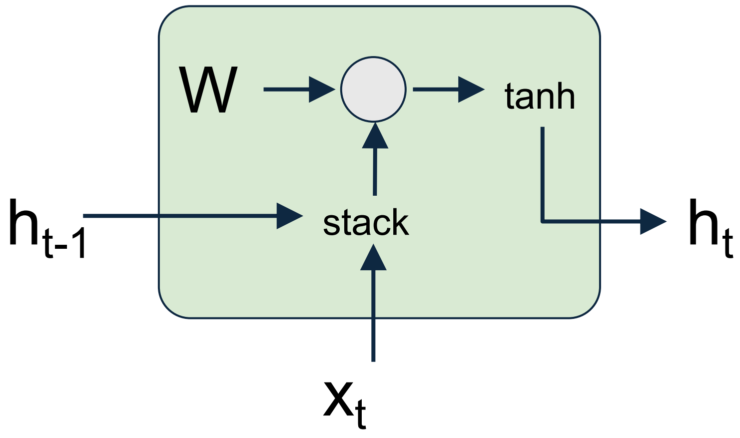
❐ **Commonly used activation functions** — The most common activation functions used in RNN modules are described below:

Sigmoid	Tanh	RELU
$g(z)=\dfrac{1}{1+e^{-z}}$	$g(z)=\dfrac{e^{z}-e^{-z}}{e^{z}+e^{-z}}$	$g(z)=\max(0,z)$
		

https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks

Vanilla RNN Gradient Flow

Bengio et al, “Learning long-term dependencies with gradient descent is difficult”, IEEE Transactions on Neural Networks, 1994
Pascanu et al, “On the difficulty of training recurrent neural networks”, ICML 2013

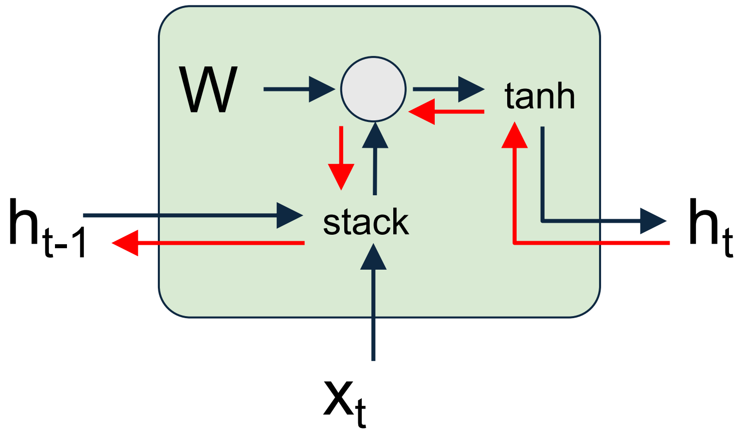


$$
\begin{aligned}
h_t &= \tanh(W_{hh}h_{t-1} + W_{xh}x_t) \\
&= \tanh\left(\begin{pmatrix} W_{hh} & W_{hx} \end{pmatrix} \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right) \\
&= \tanh\left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right)
\end{aligned}
$$

Vanilla RNN Gradient Flow

Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Pascanu et al, "On the difficulty of training recurrent neural networks", ICML 2013

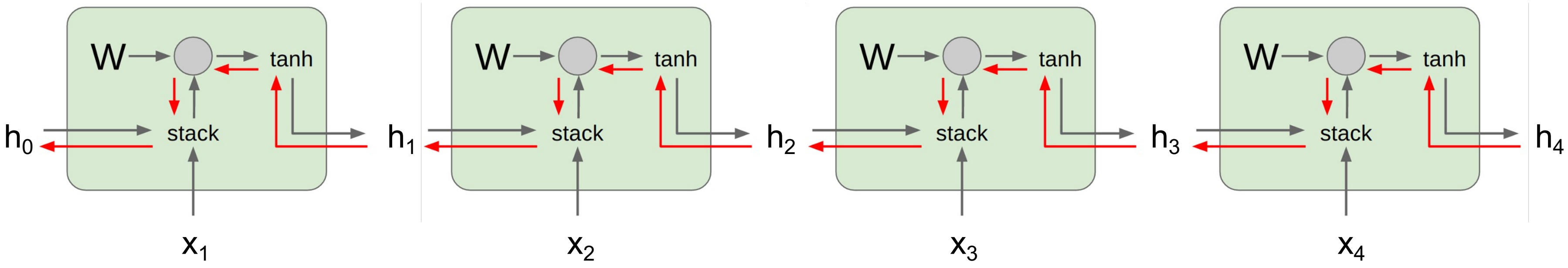
Backpropagation from h_t to h_{t-1} multiplies by W (actually W_{hh}^T)



$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$
$$= \tanh\left(\begin{pmatrix} W_{hh} & W_{hx} \end{pmatrix} \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right)$$
$$= \tanh\left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right)$$

Vanilla RNN Gradient Flow

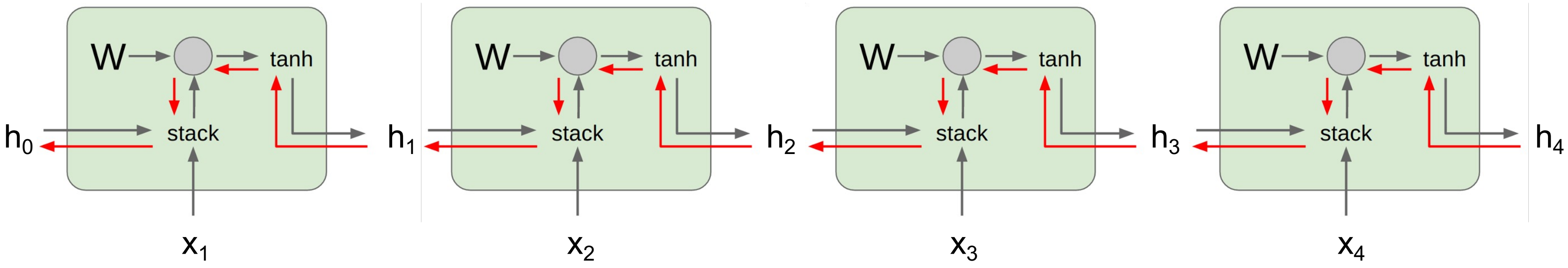
Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Pascanu et al, "On the difficulty of training recurrent neural networks", ICML 2013



Computing gradient of h_0 involves many factors of W (and repeated tanh)

Vanilla RNN Gradient Flow

Bengio et al, “Learning long-term dependencies with gradient descent is difficult”, IEEE Transactions on Neural Networks, 1994
Pascanu et al, “On the difficulty of training recurrent neural networks”, ICML 2013



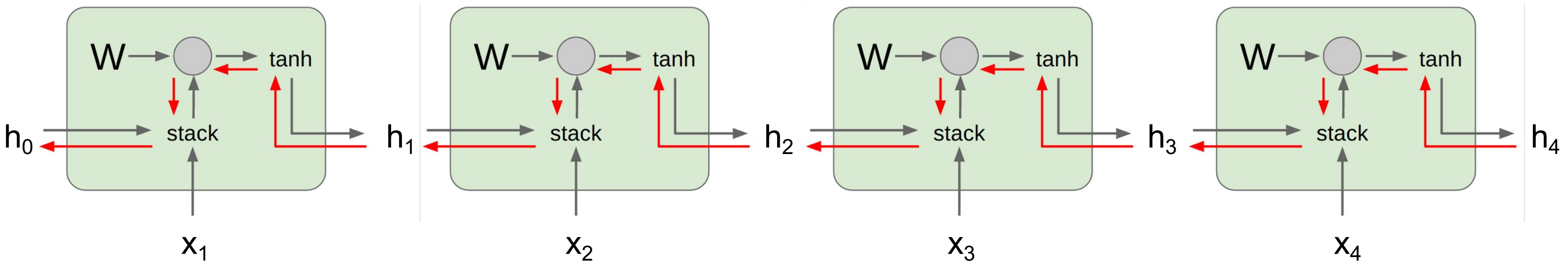
Computing gradient of h_0 involves many factors of W (and repeated tanh)

Largest singular value > 1:
Exploding gradients

Largest singular value < 1:
Vanishing gradients

Vanilla RNN Gradient Flow

Bengio et al, “Learning long-term dependencies with gradient descent is difficult”, IEEE Transactions on Neural Networks, 1994
Pascanu et al, “On the difficulty of training recurrent neural networks”, ICML 2013



Computing gradient of h_0 involves many factors of W (and repeated tanh)

Largest singular value > 1:
Exploding gradients

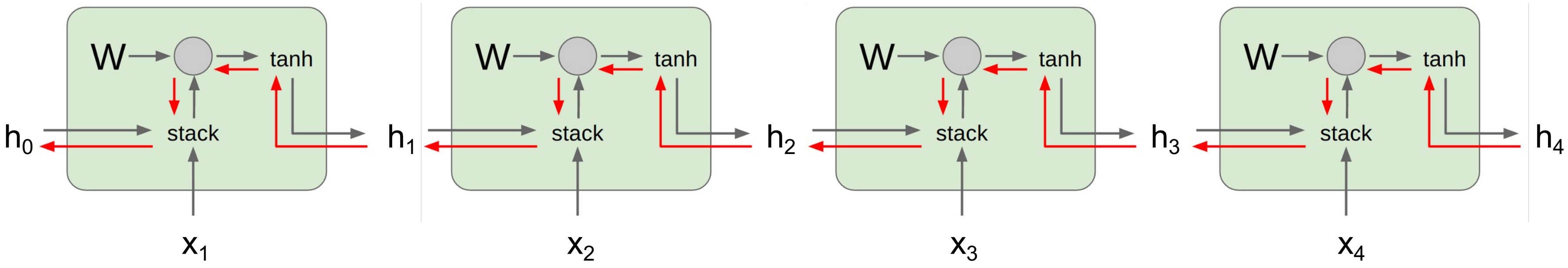
→ **Gradient clipping**: Scale gradient if its norm is too big

```
grad_norm = np.sum(grad * grad)
if grad_norm > threshold:
  grad *= (threshold / grad_norm)
```

Largest singular value < 1:
Vanishing gradients

Vanilla RNN Gradient Flow

Bengio et al, “Learning long-term dependencies with gradient descent is difficult”, IEEE Transactions on Neural Networks, 1994
Pascanu et al, “On the difficulty of training recurrent neural networks”, ICML 2013



Computing gradient of h_0 involves many factors of W (and repeated tanh)

Largest singular value > 1:
Exploding gradients

Largest singular value < 1:
Vanishing gradients

→ Change RNN architecture

Limitations of RNN

- RNNs can not handle long term dependencies in practice

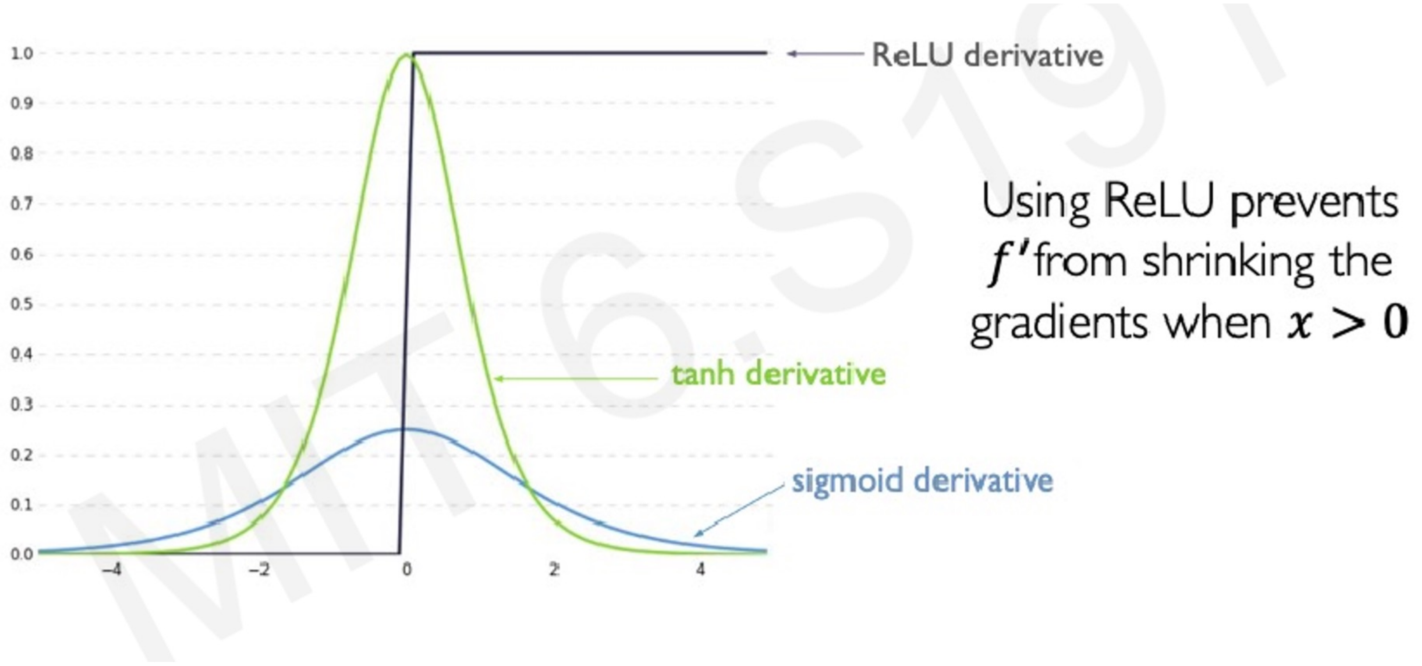
- Example task: next word prediction by RNN, vocabulary of English
  1. Bird fly in the _? [*sky*]
  2. I was born in Germany and I can speak fluently in_ ? [*German*]

  ➢ Sentence 1: sky is predicted from the previous words fly and bird
  ➢ Sentence 2: *German* needs to be predicted from the word Germany

- Gap between the relevant information and the sequence output place is large for Sentence 2
  - RNNs become unable to learn and connect the information

How to RNN Handle Limitations?

1. Choose a different activation function



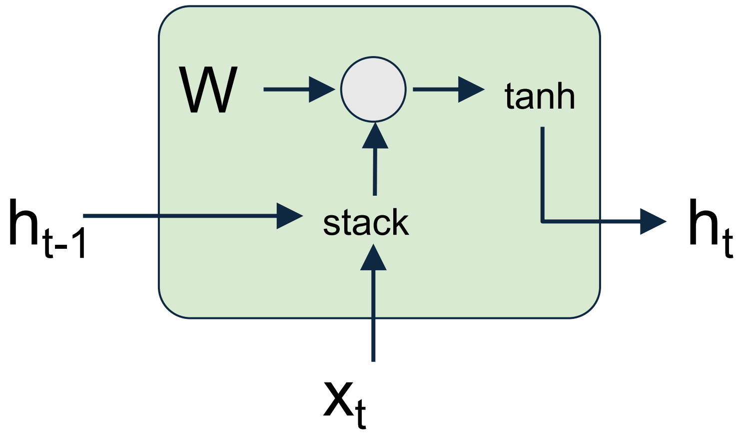
How to RNN Handle Limitations?

2. Initialize weight of the matrices to identity matrices

3. Change the RNN structure

Long Short Term Memory (LSTM)

- The recurrence block of an RNN has only one tanh activation



Long Short Term Memory (LSTM)

The recurrence block of an LSTM has four components

Vanilla RNN

$$h_t = \tanh\left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right)$$

LSTM

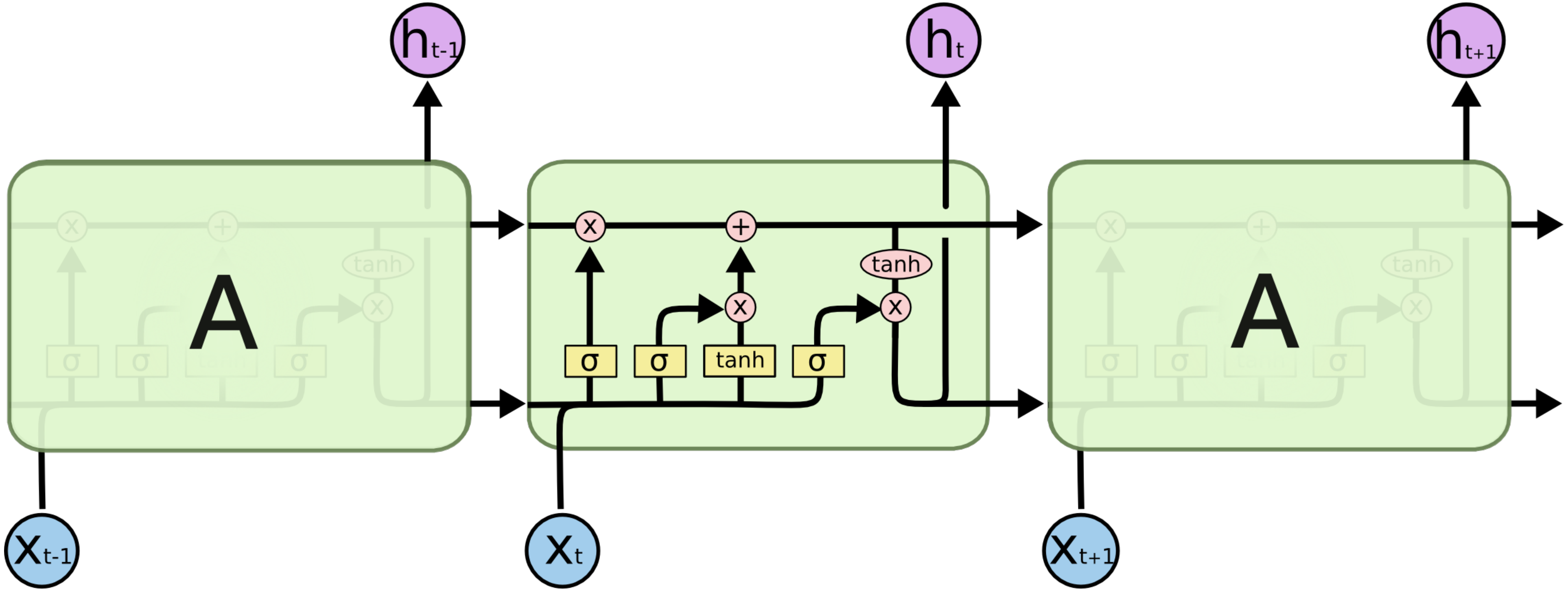
$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

Hochreiter and Schmidhuber, “Long Short Term Memory”, Neural Computation 1997

LSTM Structure



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

LSTMs: Key Concepts

1. Maintain a **cell state**
2. Use **gates** to control the **flow of information**
   - **Forget** gate gets rid of irrelevant information
   - **Store** relevant information from current input
   - Selectively **update** cell state
   - **Output** gate returns a filtered version of the cell state
3. Backpropagation through time with partially **uninterrupted gradient flow**

LSTM

- **Step 1:** The LSTM receives the input vector (xt) and the previous state (ht-1, ct-1).

- **Step 2:** The forget gate (ft) decides what information to discard from the cell state. It uses the input vector and the previous hidden state to generate a number between 0 and 1 for each number in the cell state ct-1.
  - A 1 represents "completely keep this“
  - A 0 represents "completely get rid of this".

- Forget Gate: ft = σ(Wf.[ht-1, xt] + bf)

https://www.linkedin.com/pulse/understanding-lstm-in-depth-look-its-architecture-pros-babu-thomas/

LSTM

- **Step 3:** The input gate (it) decides what new information to store in the cell state. It has two parts. A sigmoid layer called the "input gate layer" decides which values we'll update, and a tanh layer creates a vector of new candidate values (Ct~) that could be added to the state.
- Input Gate: $it = \sigma(Wi.[ht-1, xt] + bi)$
- Candidate Values(Cell State Update): $Ct\sim = \tanh(Wc.[ht-1, xt] + bc)$

LSTM

- **Step 4:** Update the old cell state (ct-1) to the new cell state (ct). The old cell state is multiplied by ft to forget the things we decided to forget earlier. Then we add the new candidate values, scaled by how much we decided to update each state value.

- Cell State(Final Cell State): ct = ft * ct-1 + it * Ct~

LSTM

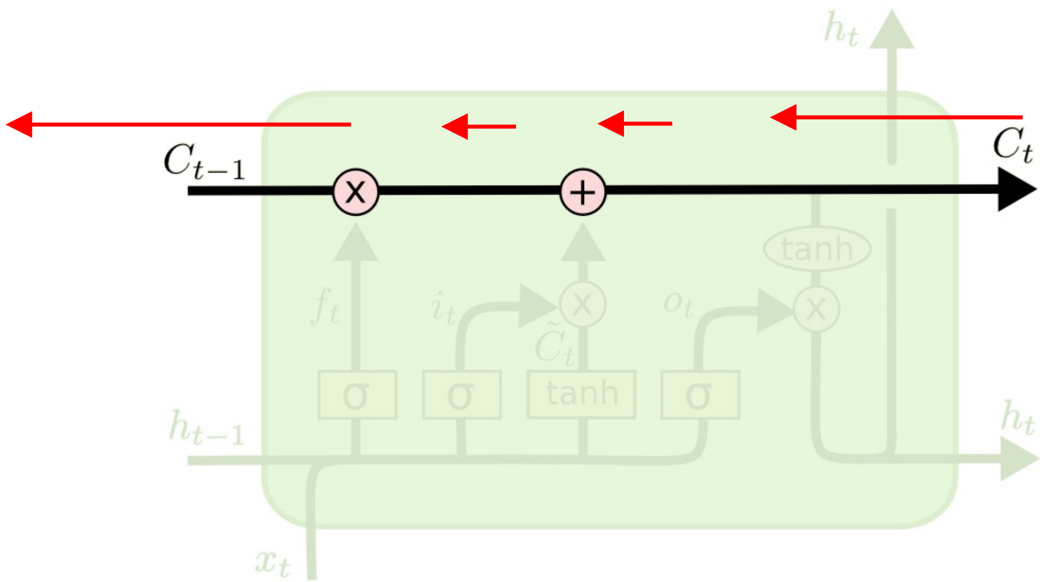
- **Step 5:** Decide the output. This output will be based on our cell state, but will be a filtered version. First, we run a sigmoid layer which decides what parts of the cell state we're going to output. Then, we put the cell state through tanh (to push the values to be between -1 and 1) and multiply it by the output of the sigmoid gate, so we only output the parts we decided to.
- Output Gate: $o_t = \sigma(W_o.[h_{t-1}, x_t] + b_o)$
- Hidden State: $h_t = o_t * \tanh(c_t)$

LSTM

- Why tanh activation in the cell state?
  - To overcome the vanishing gradient problem
  - The tanh activation function’s derivative can sustain for a long range before going to zero.

- Why sigmoid activation in the forget gate?
  - Sigmoid outputs 0 or 1, it can be used to forget or remember the information.

LSTMs Intuition: Additive Updates



Backpropagation from c_t to c_{t-1} only elementwise multiplication by f, no matrix multiply by W

Image Credit: Christopher Olah (http://colah.github.io/posts/2015-08-Understanding-LSTN

LSTMs Intuition: Additive Updates

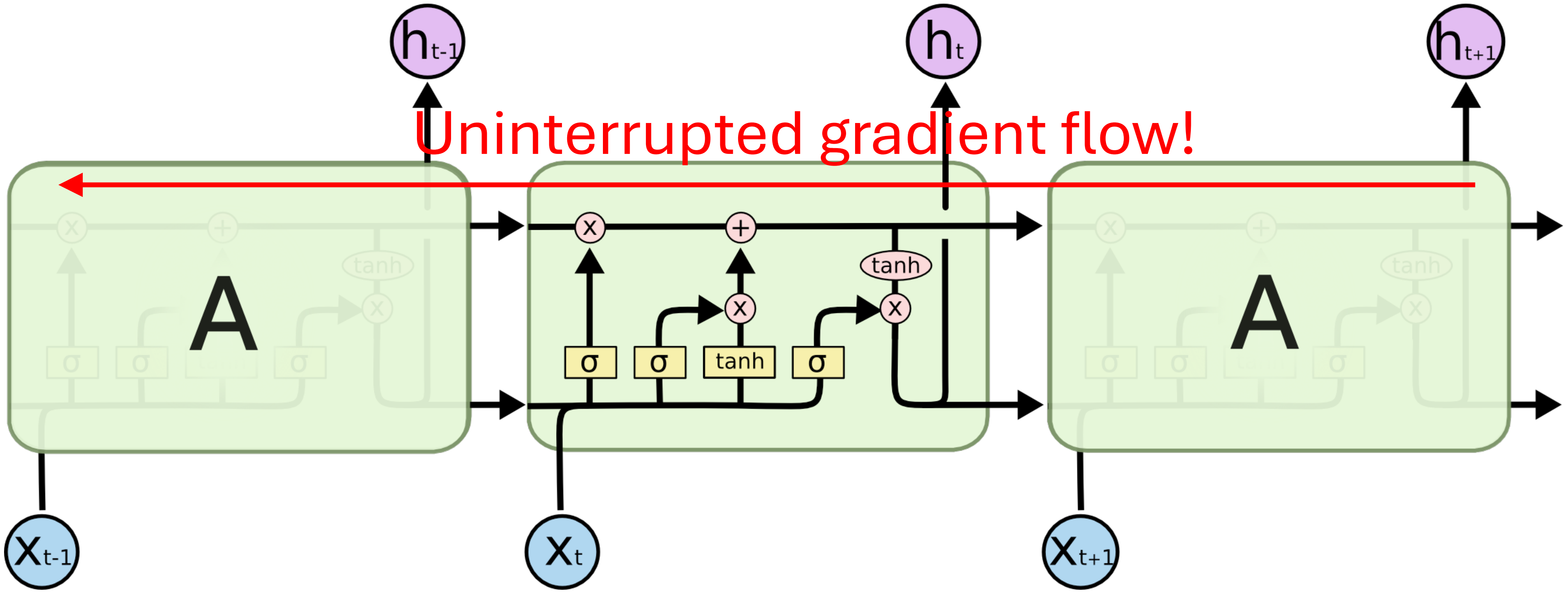
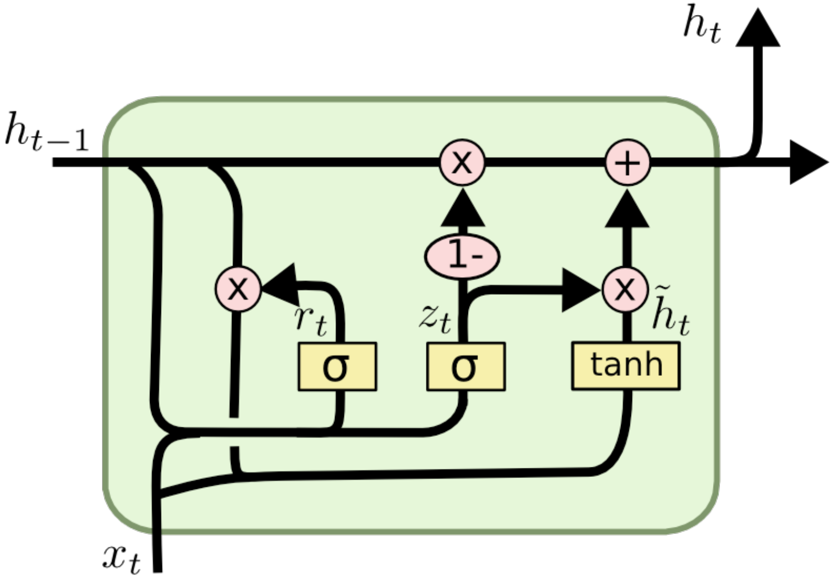


Image Credit: Christopher Olah (http://colah.github.io/posts/2015-08-Understanding-LSTN

LSTM Variants: Gated Recurrent Units (GRU)

- Changes:
  - No explicit memory; memory = hidden output
  - Z = memorize new and forget old



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

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

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

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

Image Credit: Christopher Olah (http://colah.github.io/posts/2015-08-Understanding-LSTM