SC 4001 CE/CZ 4042 Neural Network and Deep Learning

Last update: 13 Mar 2024

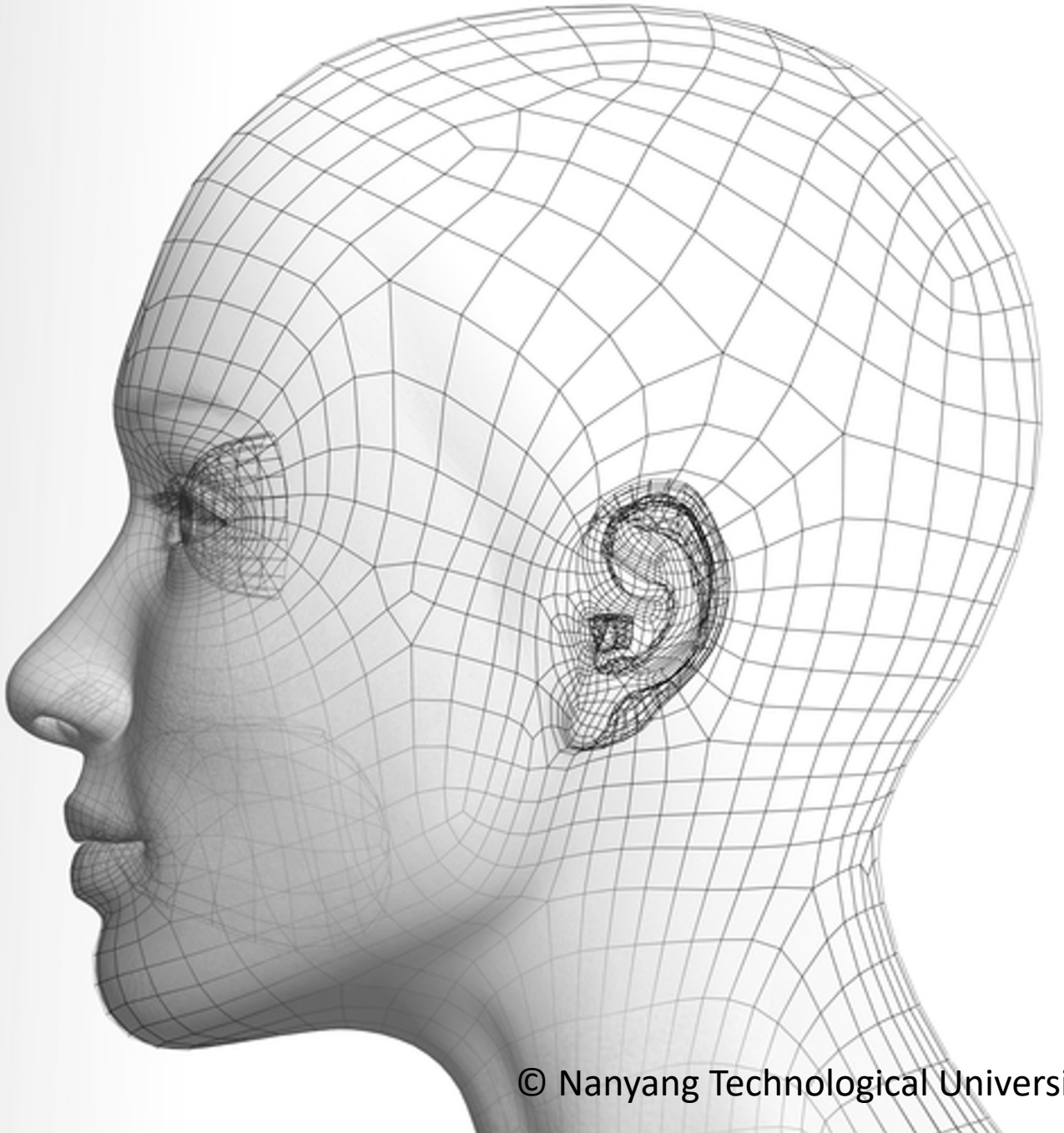
Recurrent Neural Networks (RNN)

Xingang Pan

潘新钢

https://xingangpan.github.io/

https://twitter.com/XingangP



Last week

- CNN Architectures

  You learn some classic architectures

- More on convolution
  - How to calculate FLOPs
  - Pointwise convolution
  - Depthwise convolution
  - Depthwise convolution + Pointwise convolution

  You learn how to calculate computation complexity of convolutional layer and how to design a lightweight network

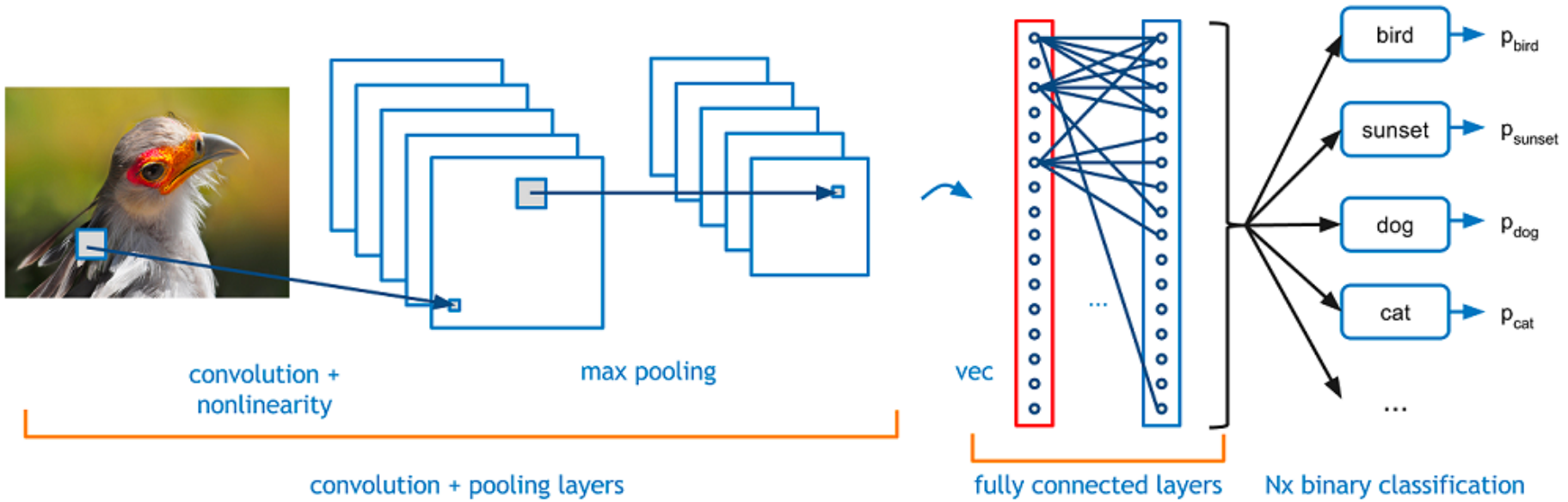
- Batch normalization

  You learn an important technique to improve the training of modern neural networks

- Prevent overfitting
  - Transfer learning
  - Data augmentation

  You learn two important techniques to prevent overfitting in neural networks

Previous: Convolutional Neural Networks (CNN)



How about sequential information?

- Turn a sequence of sound pressures into a sequence of word identities?
- Speech recognition?
- Video prediction?

Outline

- Recurrent Neural Network (RNN)
  - Hidden recurrence
  - Top-down recurrence
- Long Short-Term Memory (LSTM)
  - Long-term dependency
  - Structure of LSTM
- Example Applications

Recurrent Neural Network (RNN)

Recurrent Neural Networks (RNN)

Recurrent neural networks (RNN) are designed to process sequential information. That is, the data presented in a sequence.

The next data point in the sequence is usually dependent on the current data point.

Examples:

- Natural language processing (spoken words and written sentences). The next word in a sentence depends on the word which comes before it.
- Genomic sequences: a nucleotide in a DNA sequence is dependent on its neighbors.

RNN attempts to capture dependency among the data points in the sequence.

Text Generation with an RNN

QUEENE:
I had thought thou hadst a Roman; for the oracle,
Thus by All bids the man against the word,
Which are so weak of care, by old care done;
Your children were in your holy love,
And the precipitation through the bleeding throne.

BISHOP OF ELY:
Marry, and will, my lord, to weep in such a one were prettiest;
Yet now I was adopted heir
Of the world's lamentable day,
To watch the next way with his father with his face?

Text Generation with an RNN

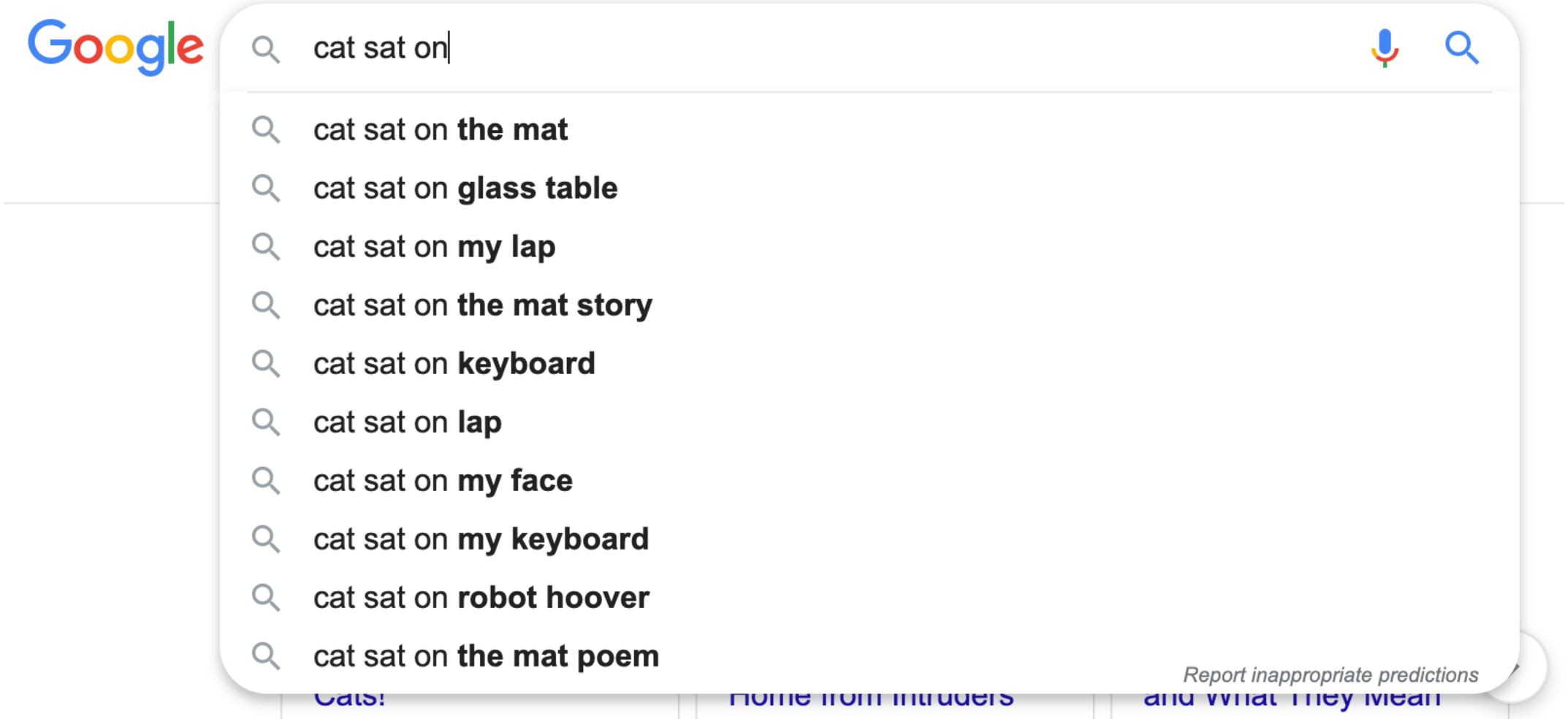
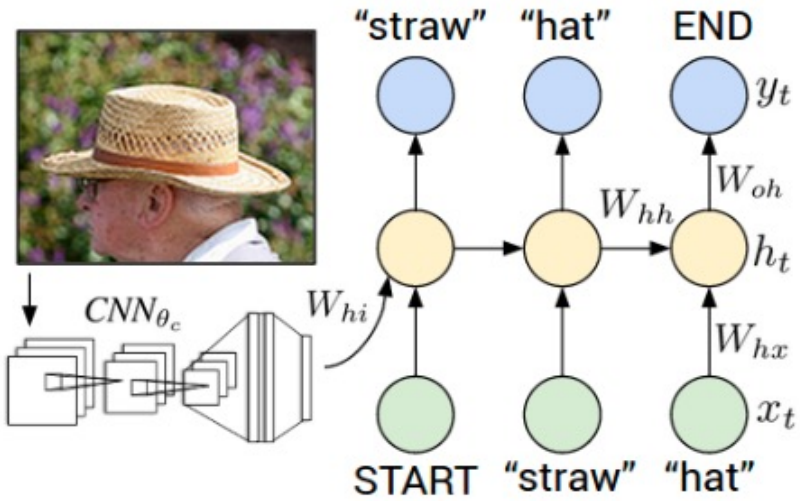


Image Captioning

"man in black shirt is playing guitar."

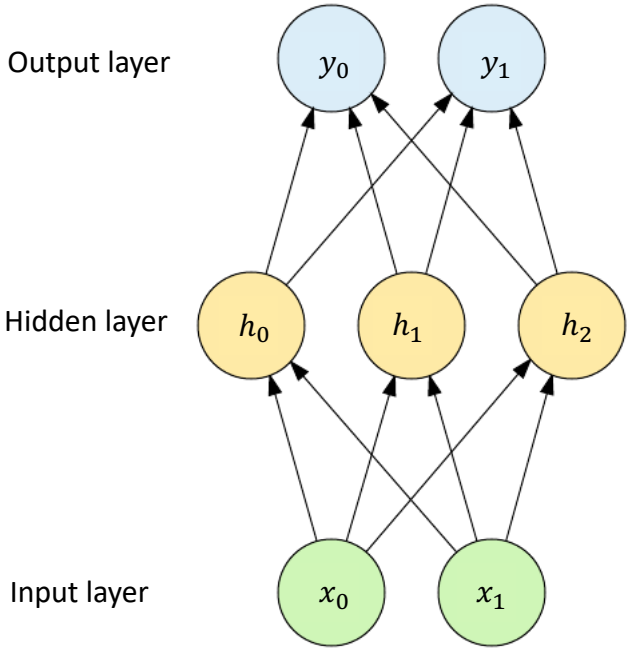
"construction worker in orange safety vest is working on road."

"two young girls are playing with lego toy."

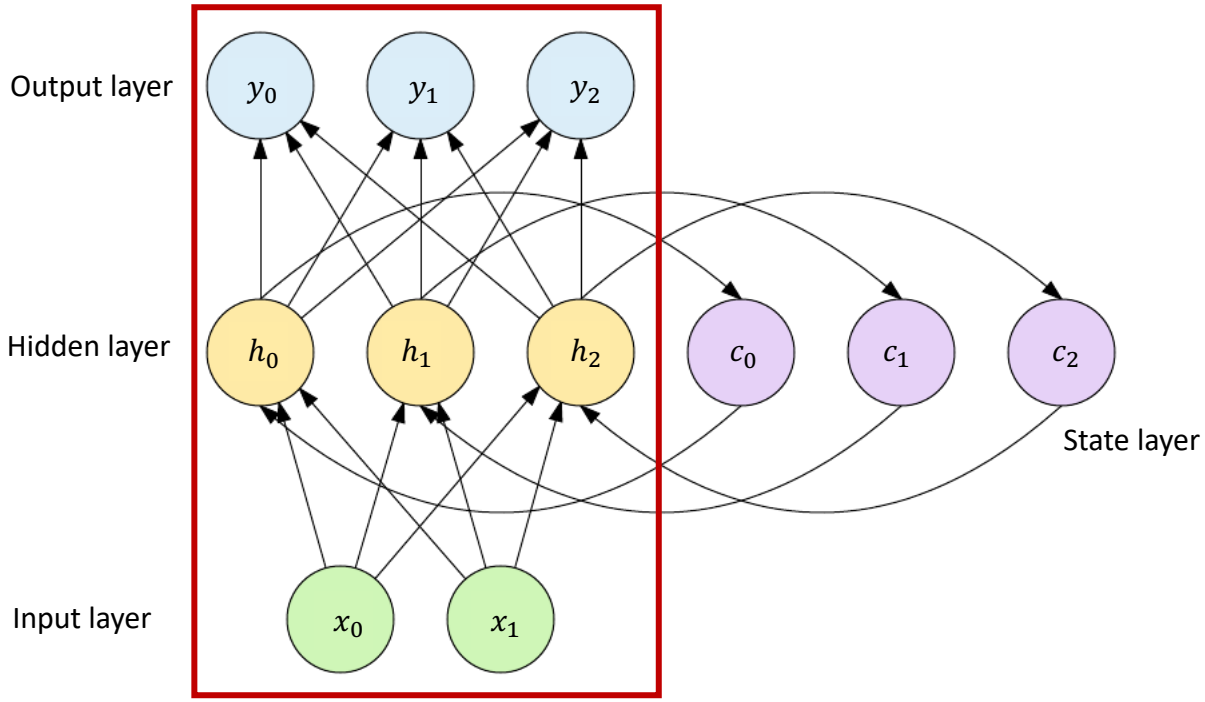


Recurrent Neural Networks (RNN)

Feedforward NN



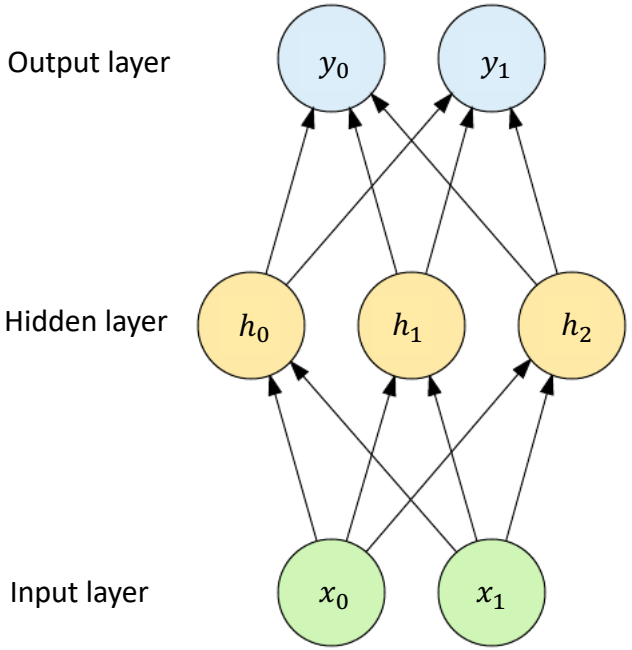
RNN with hidden recurrence



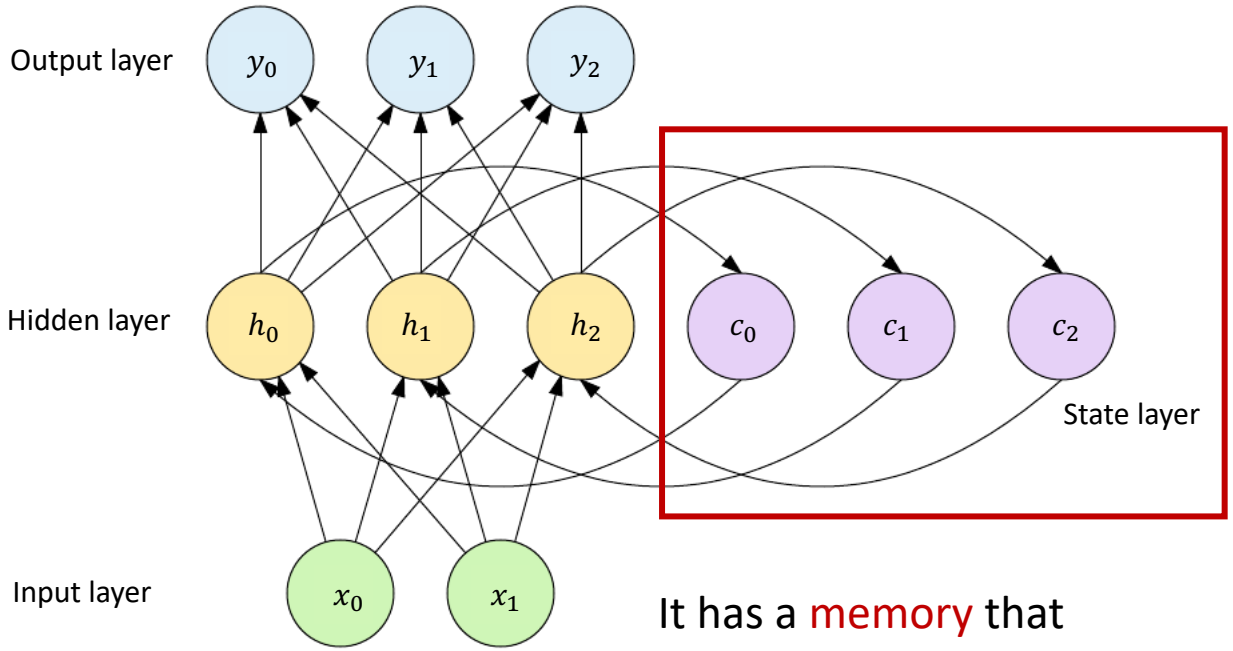
RNNs are called recurrent because they perform the same task for every data element (frame) of a sequence, with the output depending on the previous computations.

Recurrent Neural Networks (RNN)

Feedforward NN



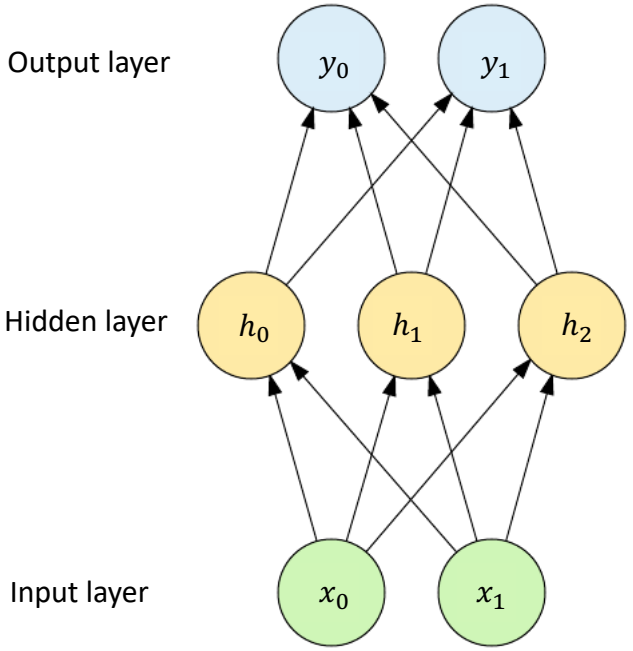
RNN with hidden recurrence



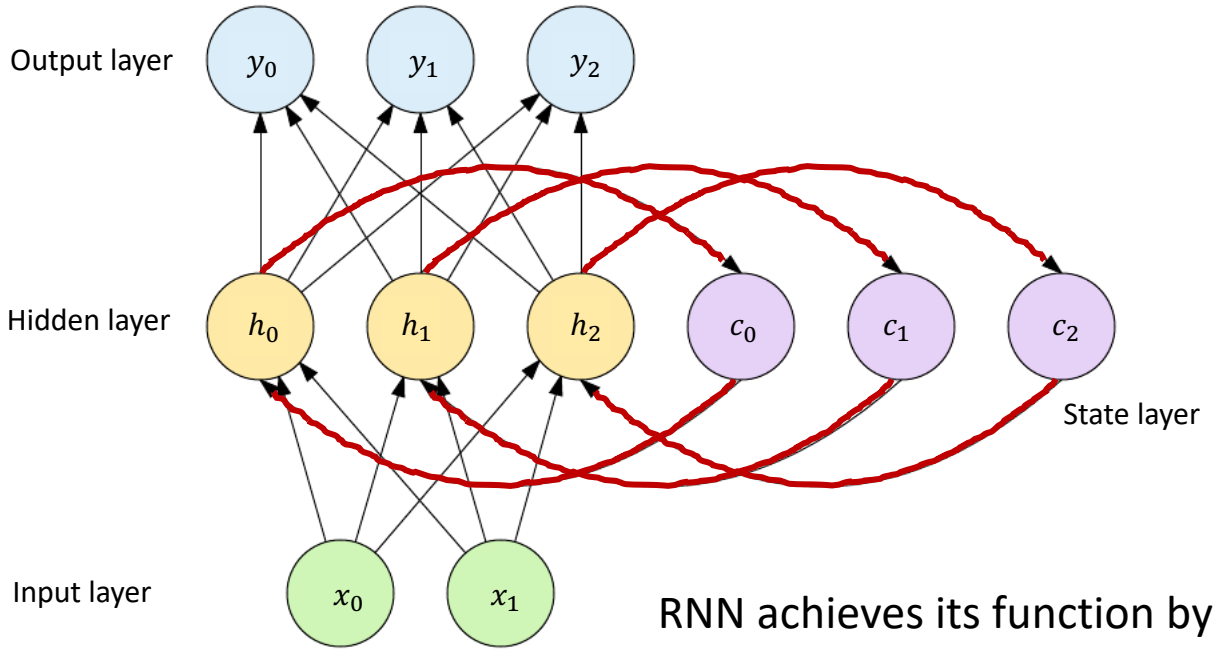
It has a memory that captures information about what has been processed so far.

Recurrent Neural Networks (RNN)

Feedforward NN



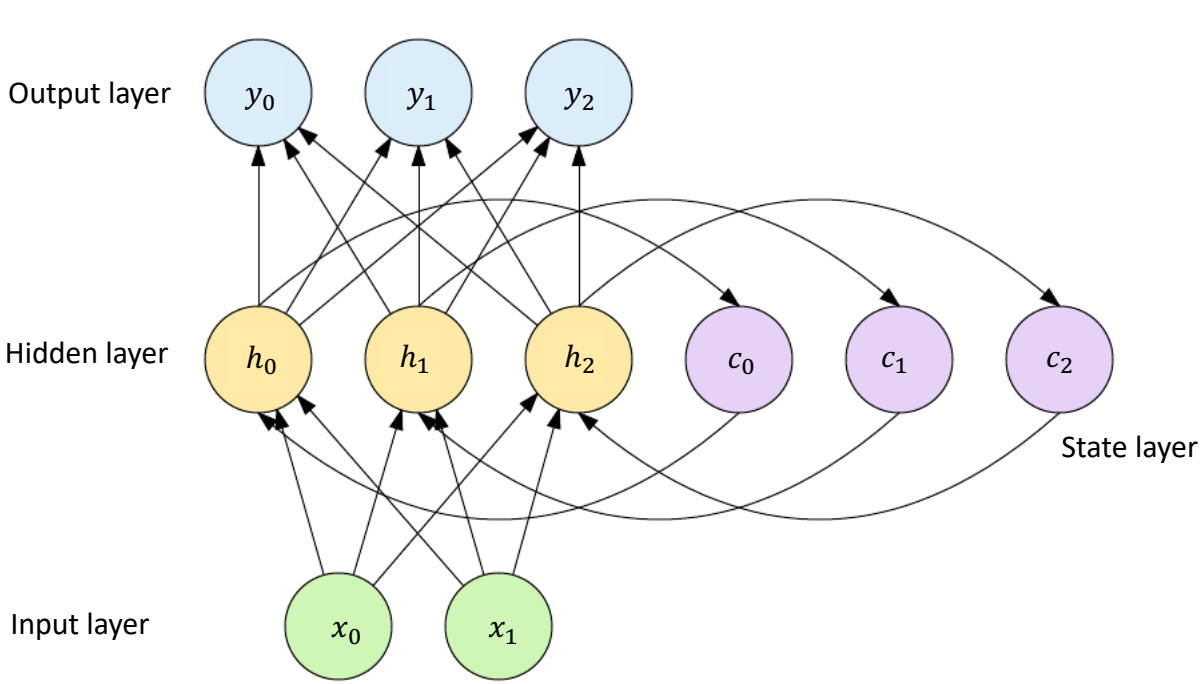
RNN with hidden recurrence



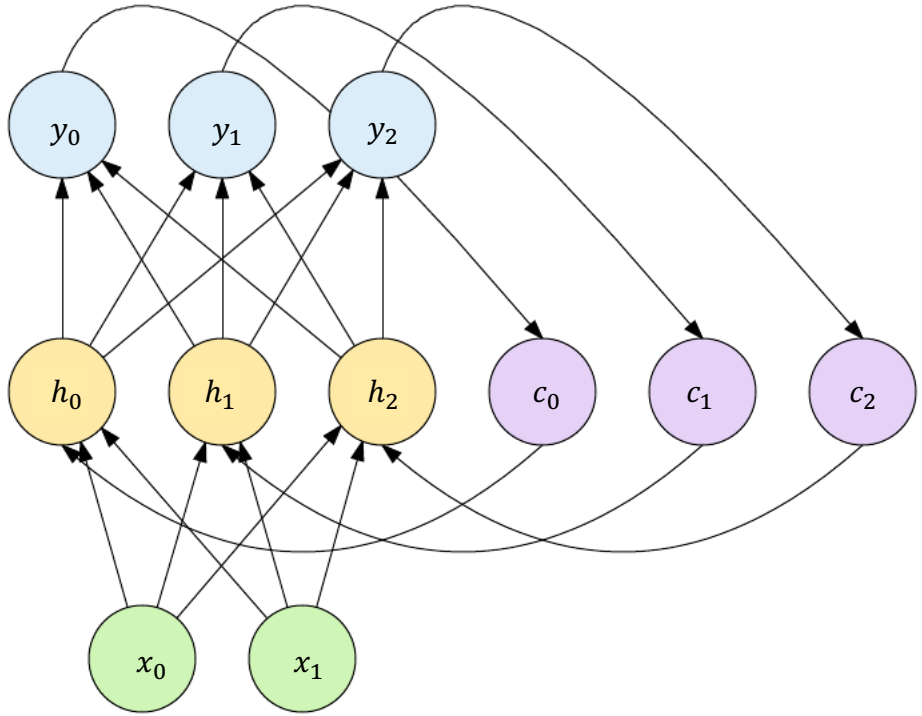
RNN achieves its function by using feedback connections that enables learning of sequential (temporal) information of sequences.

Types of RNN

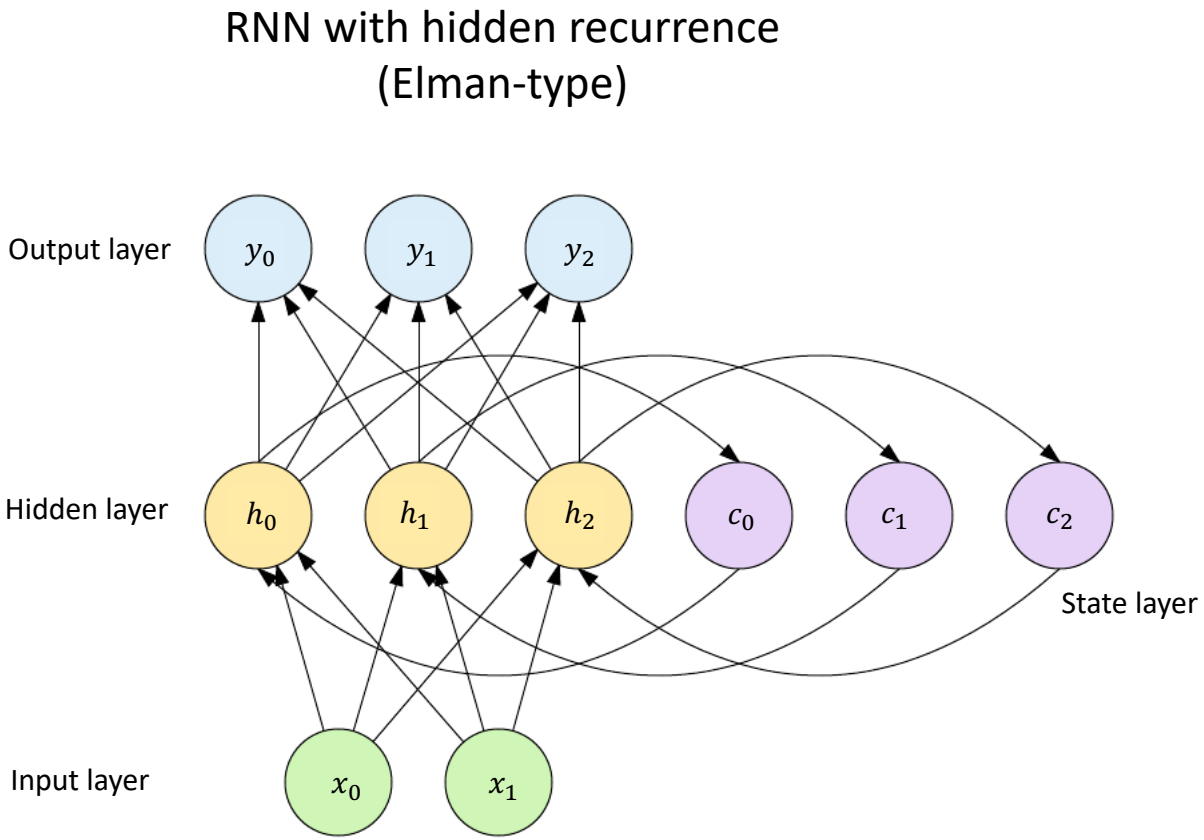
RNN with hidden recurrence
(Elman-type)

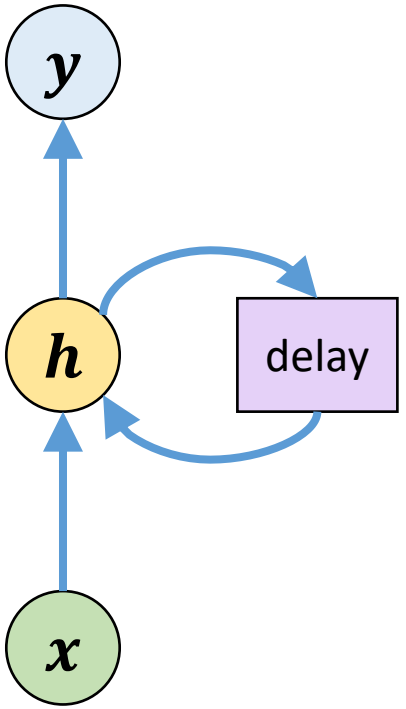


RNN with top-down recurrence
(Jordan-type)



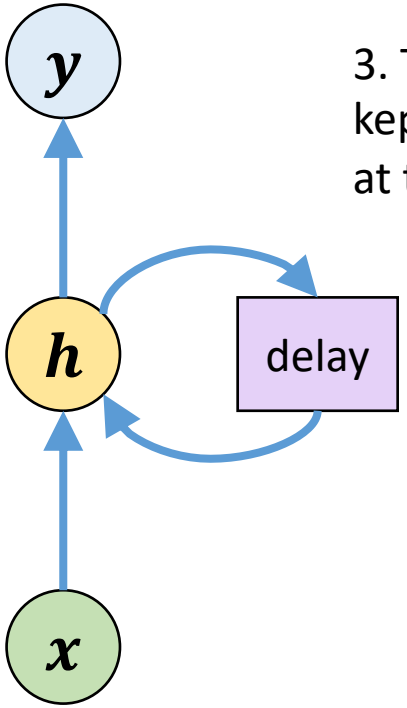
RNN with hidden recurrence (Elman type)





Elman type networks (sometimes called a "Vanilla RNN") that produce an output at each time step and have recurrent connections between hidden units.

RNN with hidden recurrence (Elman type)



3. The hidden-layer activation at time $t - 1$ is kept by the **delay unit** and fed to the hidden layer at time t together with the raw input $\boldsymbol{x}(t)$.

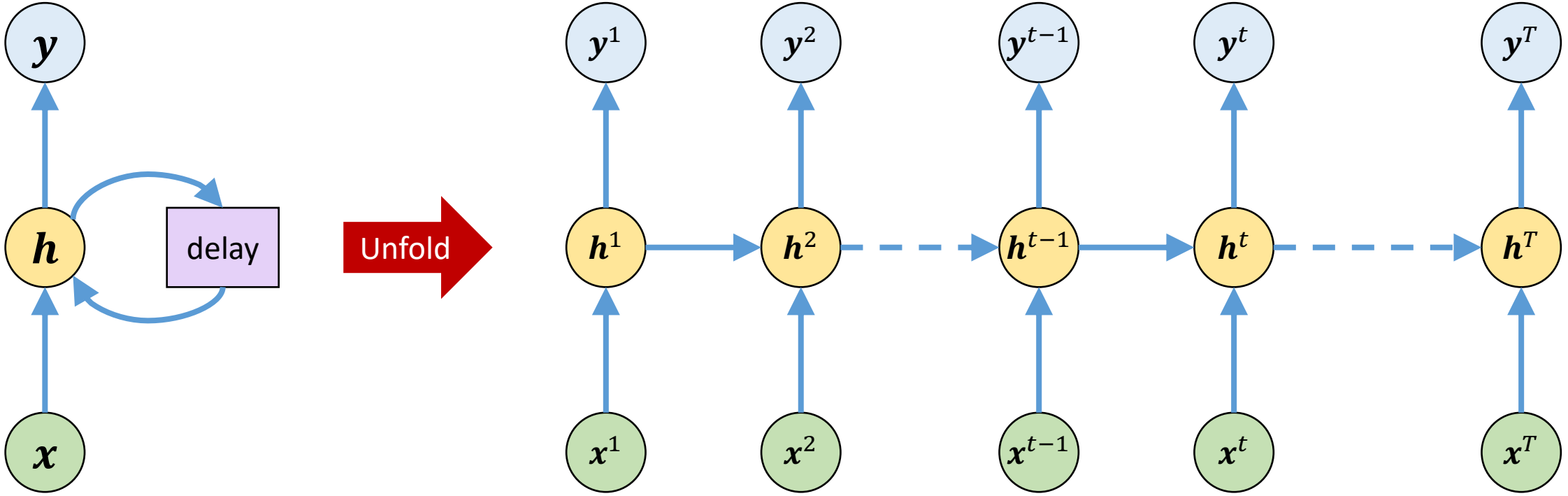
4. The **delay unit** represents that the activation is held for one time unit until the next time instance.

Here, one unit represents the time between two adjacent data points in the sequence .

2. Activations are updated at each time instant t.

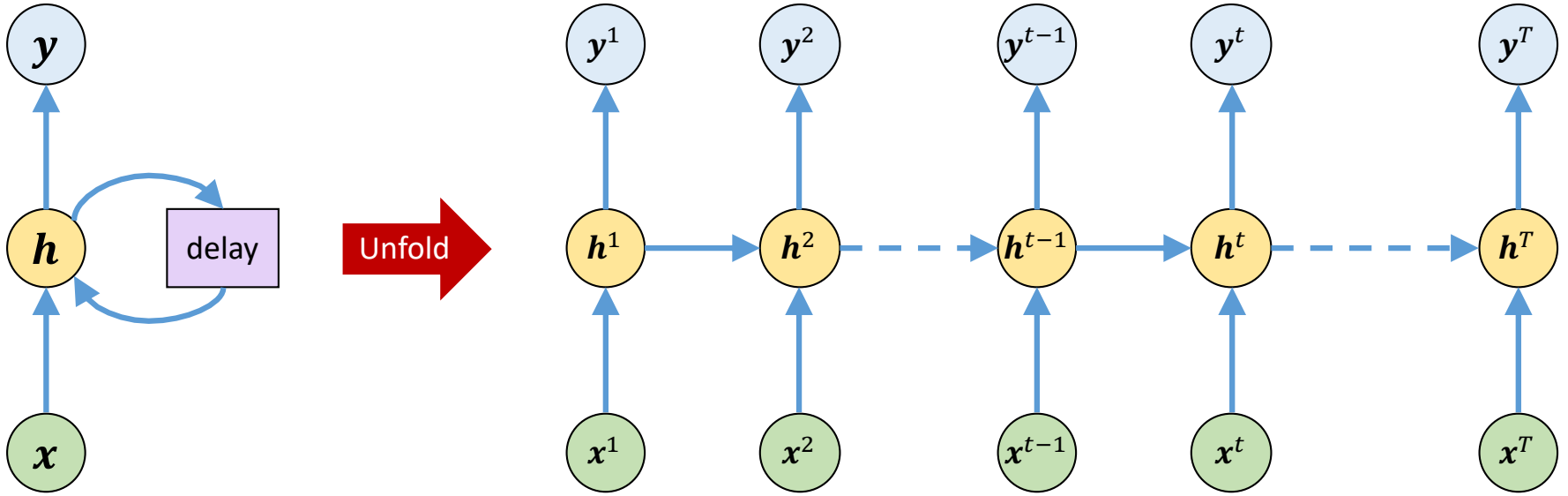
1. Data is presented as a sequence of instance t (time) that is discretized

RNN with hidden recurrence



The recurrent connections in the hidden-layer can be unfolded to process sequences of arbitrary length.

RNN with hidden recurrence



By considering the unfolded structure, $\boldsymbol{h}(t)$ is dependent on all the inputs at time t and before time t:

$$\boldsymbol{h}(t) = f^t(\boldsymbol{x}(t), \boldsymbol{x}(t-1), \dots, \boldsymbol{x}(2), \boldsymbol{x}(1))$$

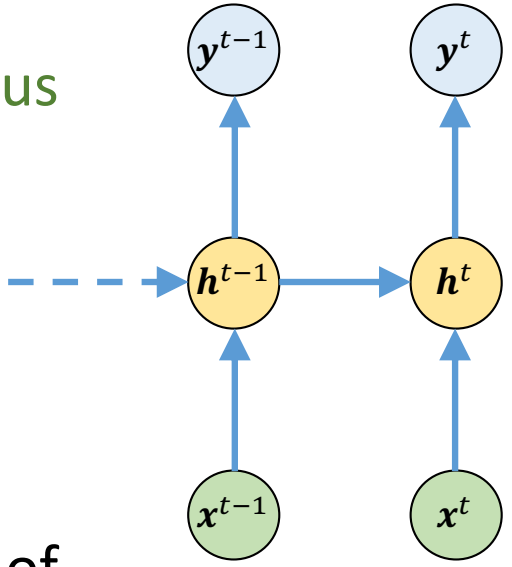
The function f^t takes the whole past sequence ,$(\boldsymbol{x}(t), \boldsymbol{x}(t-1), \dots, \boldsymbol{x}(2), \boldsymbol{x}(1))$ as input and produce the hidden layer activation.

RNN with hidden recurrence

Not an efficient way! The function f^t is dependent to the sequence length.

Let's represent the past inputs by the hidden-layer activations in the previous instant.

$$\boldsymbol{h}(t) = f^t(\boldsymbol{x}(t), \underbrace{\boldsymbol{x}(t-1), \ldots, \boldsymbol{x}(2), \boldsymbol{x}(1)}_{\boldsymbol{h}(t-1)})$$



Initial hidden state
Either set it to all zero or learn it

The recurrent structure allows us to factorize f^t into repeated applications of a function f. We can write:

new state $\boldsymbol{h}(t) = f(\boldsymbol{h}(t-1), \boldsymbol{x}(t))$

some function — old state — input vector at some time step

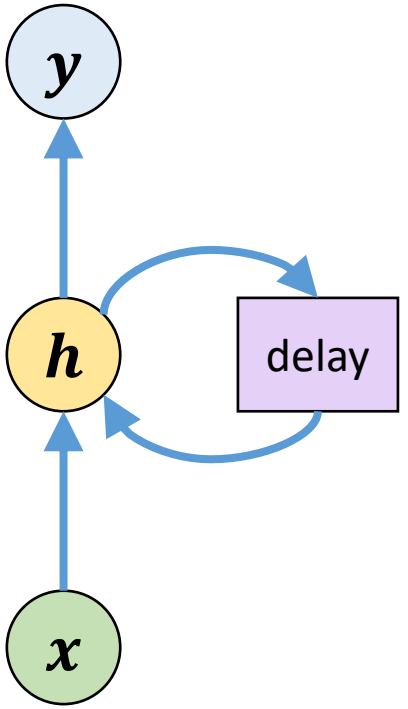
RNN with hidden recurrence

The folded structure introduces two major advantages:

$$\text{new state} \quad \boldsymbol{h}(t) = f(\boldsymbol{h}(t-1), \boldsymbol{x}(t))$$

some function — old state — input vector at some time step

1. Regardless of the sequence length, the learned model always has the same size, rather than specified in terms of a variable-length history of states.
2. It is possible to use same transition function f with the same parameters at every time step.



RNN with hidden recurrence

$\boldsymbol{U}$: weight vector that transforms raw inputs to the hidden-layer

$\boldsymbol{W}$: recurrent weight vector connecting previous hidden-layer output to hidden input

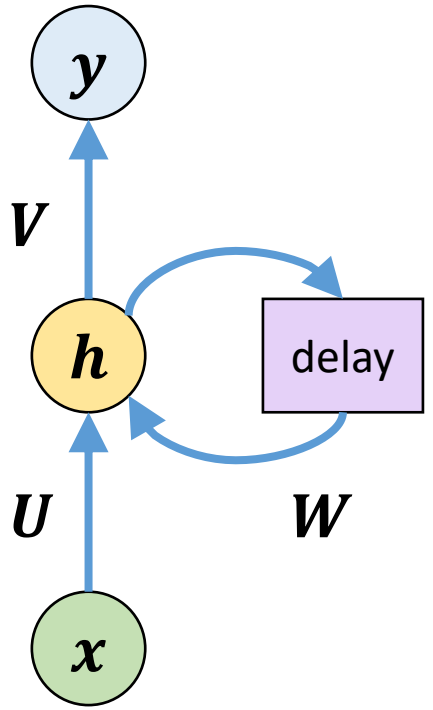
$\boldsymbol{V}$: weight vector of the output layer

$\boldsymbol{b}$: bias connected to hidden layer

$\boldsymbol{c}$: bias connected to the output layer

ϕ: the tanh hidden-layer activation function

σ: the linear/softmax output-layer activation function

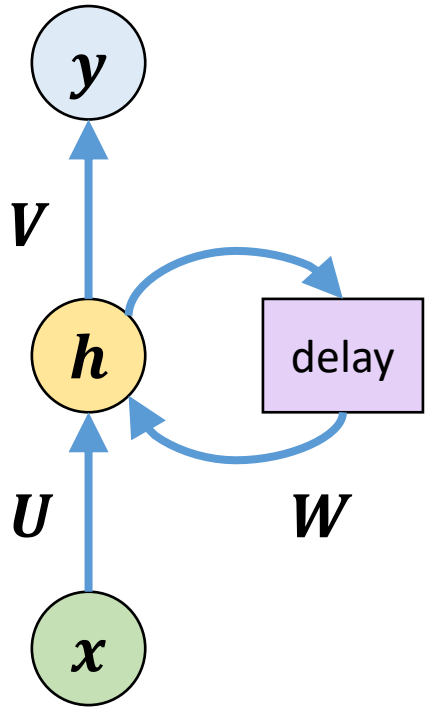


RNN with hidden recurrence

Let $\boldsymbol{x}(t)$, $\boldsymbol{y}(t)$, and $\boldsymbol{h}(t)$ be the input, output, and hidden output of the network at time t.

Activation of the Elman-type RNN with one hidden-layer is given by:

$$\boldsymbol{h}(t) = \phi\left(\boldsymbol{U}^{\mathsf{T}}\boldsymbol{x}(t) + \boldsymbol{W}^{\mathsf{T}}\boldsymbol{h}(t-1) + \boldsymbol{b}\right)$$
$$\boldsymbol{y}(t) = \sigma\left(\boldsymbol{V}^{\mathsf{T}}\boldsymbol{h}(t) + \boldsymbol{c}\right)$$

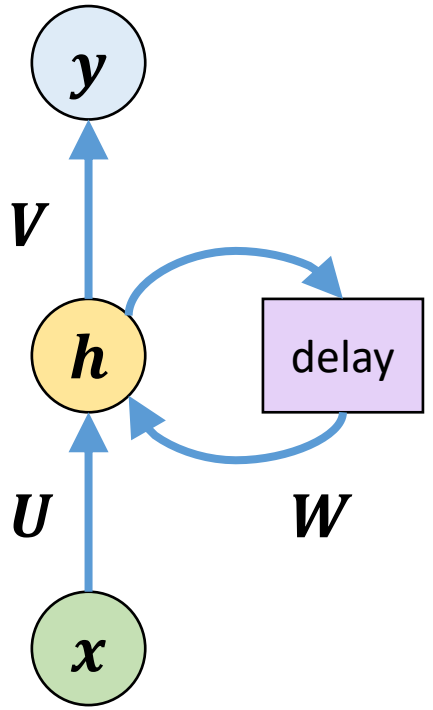


σ is a *softmax* function for classification and a *linear* function for regression.

RNN with hidden recurrence: batch processing

Given P patterns $\{\boldsymbol{x}_p\}_{p=1}^{P}$ where $\boldsymbol{x}_p = \left(\boldsymbol{x}_p(t)\right)_{t=1}^{T}$,

$$\boldsymbol{X}(t) = \begin{pmatrix} \boldsymbol{x}_1(\boldsymbol{t})^{\mathsf{T}} \\ \boldsymbol{x}_2(\boldsymbol{t})^{\mathsf{T}} \\ \vdots \\ \boldsymbol{x}_P(\boldsymbol{t})^{\mathsf{T}} \end{pmatrix}$$



Let $\boldsymbol{X}(t)$, $\boldsymbol{Y}(t)$, and $\boldsymbol{H}(t)$ be batch input, output, and hidden output of the network at time t

Activation of the three-layer Elman-type RNN is given by:

$$\boldsymbol{H}(t) = \phi(\boldsymbol{X}(t)\boldsymbol{U} + \boldsymbol{H}(t-1)\boldsymbol{W} + \boldsymbol{B})$$
$$\boldsymbol{Y}(t) = \sigma(\boldsymbol{H}(t)\boldsymbol{V} + \boldsymbol{C})$$

Example 1

A recurrent neural network with hidden recurrence has two input neurons, three hidden neurons, and two output neurons. The parameters of the network are initialized as $\boldsymbol{U} = \begin{pmatrix} -1.0 & 0.5 & 0.2 \\ 0.5 & 0.1 & -2.0 \end{pmatrix}$, $\boldsymbol{W} = \begin{pmatrix} 2.0 & 1.3 & -1.0 \\ 1.5 & 0.0 & -0.5 \\ -0.2 & 1.5 & 0.4 \end{pmatrix}$ and $\boldsymbol{V} = \begin{pmatrix} 2.0 & -1.0 \\ -1.5 & 0.5 \\ 0.2 & 0.8 \end{pmatrix}$.

Bias to the hidden layer $\boldsymbol{b} = \begin{pmatrix} 0.2 \\ 0.2 \\ 0.2 \end{pmatrix}$ and to the output layer $\boldsymbol{c} = \begin{pmatrix} 0.5 \\ 0.1 \end{pmatrix}$.

For a sequence of inputs $\big(\boldsymbol{x}(1), \boldsymbol{x}(2), \boldsymbol{x}(3), \boldsymbol{x}(4)\big)$ where

$$\boldsymbol{x}(1) = \begin{pmatrix} 1 \\ 2 \end{pmatrix}, \boldsymbol{x}(2) = \begin{pmatrix} -1 \\ 1 \end{pmatrix}, \boldsymbol{x}(3) = \begin{pmatrix} 0 \\ 3 \end{pmatrix}, \text{ and } \boldsymbol{x}(4) = \begin{pmatrix} 2 \\ -1 \end{pmatrix}$$

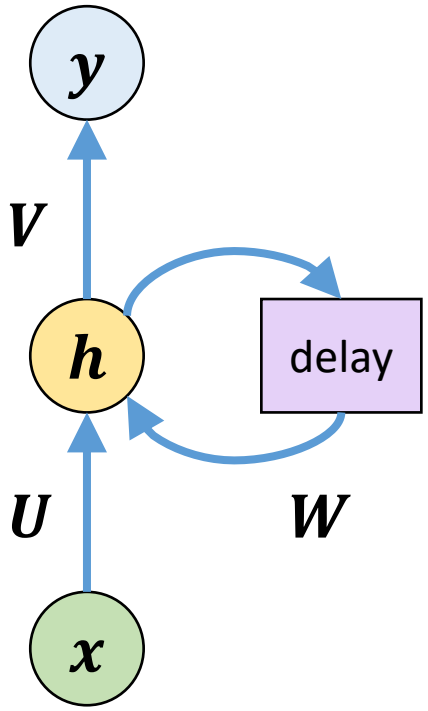
find the output of RNN.

Assume that hidden layer activations are initialized to zero and *tanh* and sigmoid functions for the hidden and output layer activation functions, respectively.

eg8.1.ipynb

Example 1 (con't)

$$\boldsymbol{U} = \begin{pmatrix} -1.0 & 0.5 & 0.2 \\ 0.5 & 0.1 & -2.0 \end{pmatrix}, \quad \boldsymbol{W} = \begin{pmatrix} 2.0 & 1.3 & -1.0 \\ 1.5 & 0.0 & -0.5 \\ -0.2 & 1.5 & 0.4 \end{pmatrix} \text{ and } \boldsymbol{V} = \begin{pmatrix} 2.0 & -1.0 \\ -1.5 & 0.5 \\ 0.2 & 0.8 \end{pmatrix}$$



$$\boldsymbol{b} = \begin{pmatrix} 0.2 \\ 0.2 \\ 0.2 \end{pmatrix} \text{ and } \boldsymbol{c} = \begin{pmatrix} 0.5 \\ 0.1 \end{pmatrix}$$

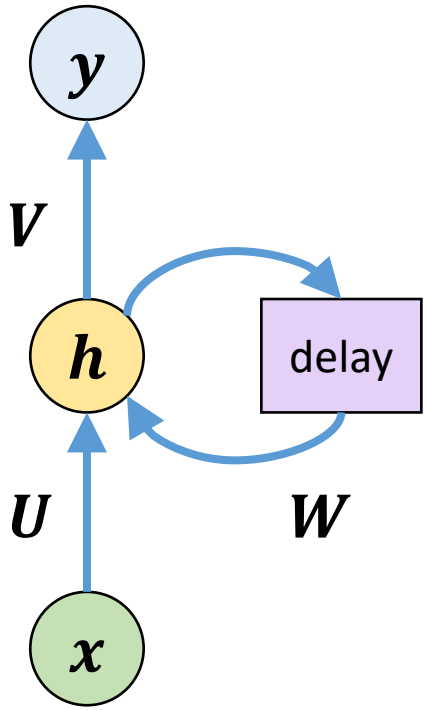
Three hidden neurons and two output neurons

$$\boldsymbol{h}(t) = \phi\left(\boldsymbol{U}^{\boldsymbol{T}}\boldsymbol{x}(t) + \boldsymbol{W}^{\boldsymbol{T}}\boldsymbol{h}(t-1) + \boldsymbol{b}\right)$$
$$\boldsymbol{y}(t) = \sigma\left(\boldsymbol{V}^{\boldsymbol{T}}\boldsymbol{h}(t) + \boldsymbol{c}\right)$$

$$\phi(u) = \tanh(u) = \frac{e^{u} - e^{-u}}{e^{u} + e^{-u}}$$
$$\sigma(u) = \text{sigmoid}(u) = \frac{1}{1 + e^{-u}}.$$

Assume $\boldsymbol{h}(0) = (0 \quad 0 \quad 0)^T$.

Example 1 (con't)



At t=1, $\boldsymbol{x}(1) = \begin{pmatrix}1\\2\end{pmatrix}$:

$$\boldsymbol{h}(1) = \phi\left(\boldsymbol{U}^{\boldsymbol{T}}\boldsymbol{x}(1) + \boldsymbol{W}^{\boldsymbol{T}}\boldsymbol{h}(0) + \boldsymbol{b}\right)$$

$$= \tanh\left(\begin{pmatrix}-1.0 & 0.5\\ 0.5 & 0.1\\ 0.2 & -2.0\end{pmatrix}\begin{pmatrix}1\\2\end{pmatrix} + \begin{pmatrix}2.0 & 1.5 & -0.2\\ 1.3 & 0.0 & 1.5\\ -1.0 & -0.5 & 0.4\end{pmatrix}\begin{pmatrix}0\\0\\0\end{pmatrix} + \begin{pmatrix}0.2\\0.2\\0.2\end{pmatrix}\right)$$

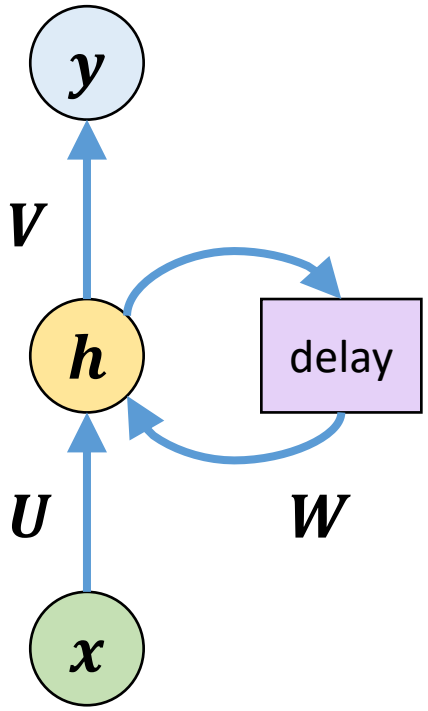
$$= \begin{pmatrix}0.2\\0.72\\-1.0\end{pmatrix}$$

$$\boldsymbol{y}(1) = \sigma\left(\boldsymbol{V}^{\boldsymbol{T}}\boldsymbol{h}(1) + \boldsymbol{c}\right)$$

$$= \text{sigmoid}\left(\begin{pmatrix}2.0 & -1.5 & 0.2\\ -1.0 & 0.5 & 0.8\end{pmatrix}\begin{pmatrix}0.2\\0.72\\-1.0\end{pmatrix} + \begin{pmatrix}0.5\\0.1\end{pmatrix}\right)$$

$$= \begin{pmatrix}0.41\\0.37\end{pmatrix}$$

Example 1 (con't)



At t=2, $\boldsymbol{x}(2) = \begin{pmatrix} -1 \\ 1 \end{pmatrix}$:

$$\boldsymbol{h}(2) = \phi\left(\boldsymbol{U}^T\boldsymbol{x}(2) + \boldsymbol{W}^T\boldsymbol{h}(1) + \boldsymbol{b}\right)$$

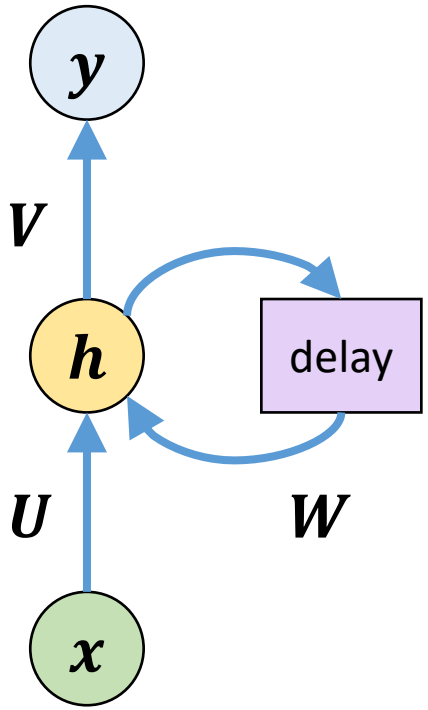
$$= \tanh\left(\begin{pmatrix} -1.0 & 0.5 \\ 0.5 & 0.1 \\ 0.2 & -2.0 \end{pmatrix}\begin{pmatrix} -1 \\ 1 \end{pmatrix} + \begin{pmatrix} 2.0 & 1.5 & -0.2 \\ 1.3 & 0.0 & 1.5 \\ -1.0 & -0.5 & 0.4 \end{pmatrix}\begin{pmatrix} 0.2 \\ 0.72 \\ -1.0 \end{pmatrix} + \begin{pmatrix} 0.2 \\ 0.2 \\ 0.2 \end{pmatrix}\right)$$

$$= \begin{pmatrix} 1.0 \\ -0.89 \\ -0.99 \end{pmatrix}$$

$$\boldsymbol{y}(2) = \sigma\left(\boldsymbol{V}^T\boldsymbol{h}(2) + \boldsymbol{c}\right)$$

$$= \text{sigmoid}\left(\begin{pmatrix} 2.0 & -1.5 & 0.2 \\ -1.0 & 0.5 & 0.8 \end{pmatrix}\begin{pmatrix} 1.0 \\ -0.89 \\ -0.99 \end{pmatrix} + \begin{pmatrix} 0.5 \\ 0.1 \end{pmatrix}\right) = \begin{pmatrix} 0.97 \\ 0.11 \end{pmatrix}$$

Example 1 (con't)



Similarly,

at t=3, $\boldsymbol{x}(3) = \begin{pmatrix} 0 \\ 3 \end{pmatrix}$; $\boldsymbol{h}(3) \begin{pmatrix} 0.99 \\ 0.3 \\ -1.0 \end{pmatrix}$ and $\boldsymbol{y}(3) = \begin{pmatrix} 0.86 \\ 0.18 \end{pmatrix}$

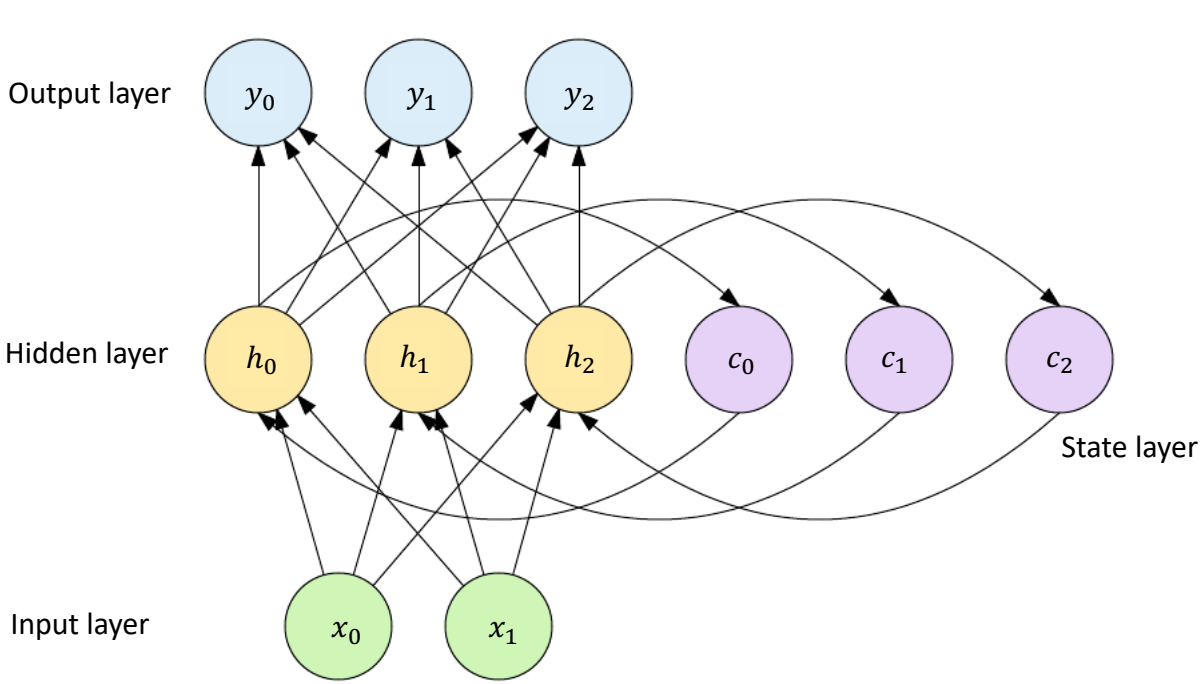
at t=4, $\boldsymbol{x}(4) = \begin{pmatrix} 2 \\ -1 \end{pmatrix}$; $\boldsymbol{h}(4) \begin{pmatrix} 0.31 \\ 0.71 \\ 0.79 \end{pmatrix}$ and $\boldsymbol{y}(4) = \begin{pmatrix} 0.55 \\ 0.68 \end{pmatrix}$

The output is $\big(\boldsymbol{y}(1), \boldsymbol{y}(2), \boldsymbol{y}(3), \boldsymbol{y}(4)\big)$ where

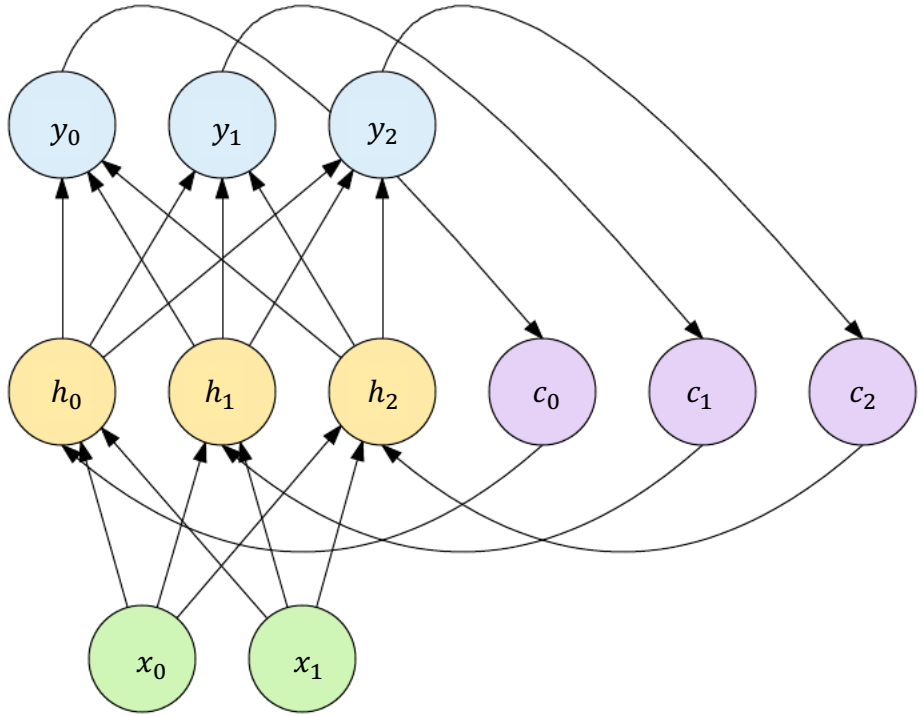
$$\boldsymbol{y}(1) = \begin{pmatrix} 0.41 \\ 0.37 \end{pmatrix}, \boldsymbol{y}(2) = \begin{pmatrix} 0.97 \\ 0.11 \end{pmatrix}, \boldsymbol{y}(3) = \begin{pmatrix} 0.86 \\ 0.18 \end{pmatrix} \boldsymbol{y}(4) = \begin{pmatrix} 0.55 \\ 0.68 \end{pmatrix}$$

Types of RNN

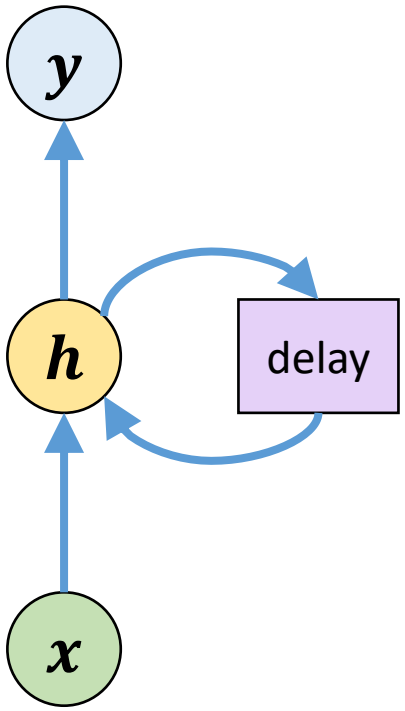
RNN with hidden recurrence
(Elman-type)



RNN with top-down recurrence
(Jordan-type)

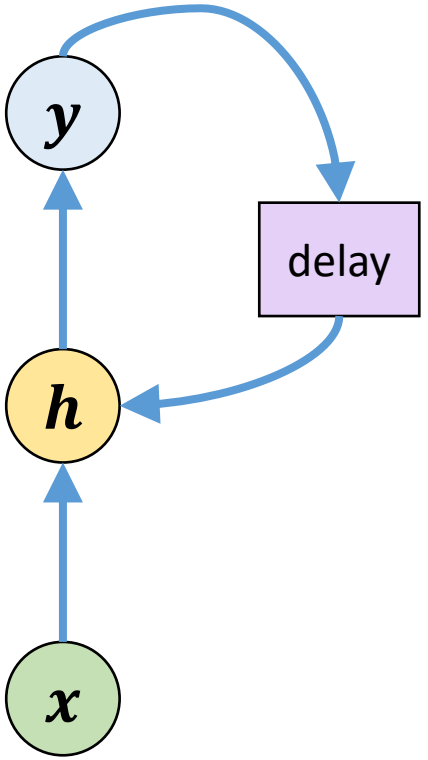


RNN with top-down recurrence (Jordan type)



The **hidden-layer activation** at time $t - 1$ is kept by the **delay unit** and fed to the hidden layer at time t together with the raw input $\boldsymbol{x}(t)$.

- $\boldsymbol{h}(t) = \phi\left(\boldsymbol{U}^{\mathsf{T}}\boldsymbol{x}(t) + \boldsymbol{W}^{\mathsf{T}}\boldsymbol{h}(t-1) + \boldsymbol{b}\right)$
- $\boldsymbol{y}(t) = \sigma\left(\boldsymbol{V}^{\mathsf{T}}\boldsymbol{h}(t) + \boldsymbol{c}\right)$



The **output of the output-layer** at time $t - 1$ is kept by the **delay unit** and fed to the hidden layer at time t together with the raw input $x(t)$ for time t

- $\boldsymbol{h}(t) = \phi\left(\boldsymbol{U}^{\mathsf{T}}\boldsymbol{x}(t) + \boldsymbol{W}^{\mathsf{T}}\boldsymbol{y}(t-1) + \boldsymbol{b}\right)$
- $\boldsymbol{y}(t) = \sigma\left(\boldsymbol{V}^{\mathsf{T}}\boldsymbol{h}(t) + \boldsymbol{c}\right)$

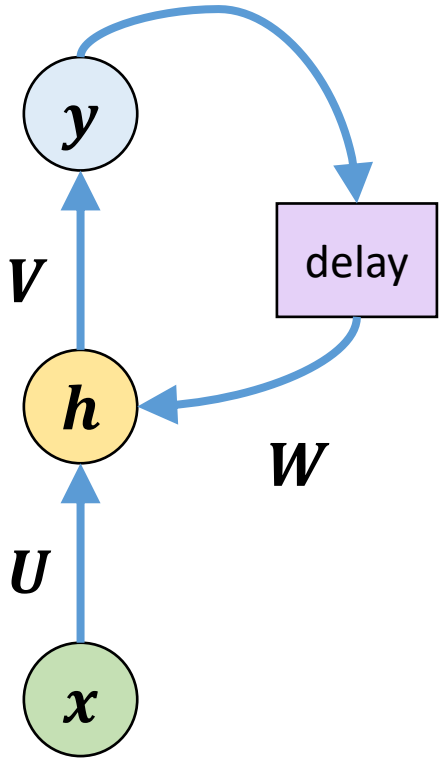
RNN with top-down recurrence

Let $\boldsymbol{x}(t)$, $\boldsymbol{y}(t)$, and $\boldsymbol{h}(t)$ be the input, output, and hidden output of the network at time t.

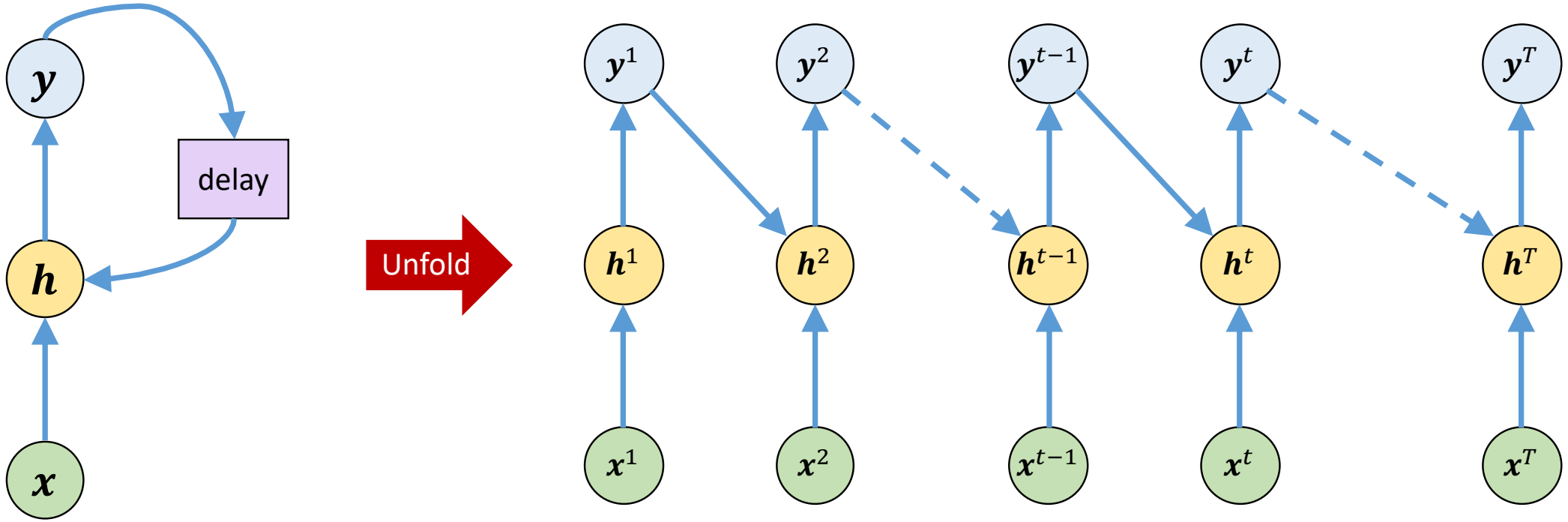
Activation of the Jordan-type RNN with one hidden-layer is given by:

$$\boldsymbol{h}(t) = \phi\left(\boldsymbol{U}^{\mathsf{T}}\boldsymbol{x}(t) + \boldsymbol{W}^{\mathsf{T}}\boldsymbol{y}(t-1) + \boldsymbol{b}\right)$$
$$\boldsymbol{y}(t) = \sigma\left(\boldsymbol{V}^{\mathsf{T}}\boldsymbol{h}(t) + \boldsymbol{c}\right)$$

Note that output of the previous time instant is fed back to the hidden layer and $\boldsymbol{W}$ represents the recurrent weight matrix connecting previous output to the current hidden input

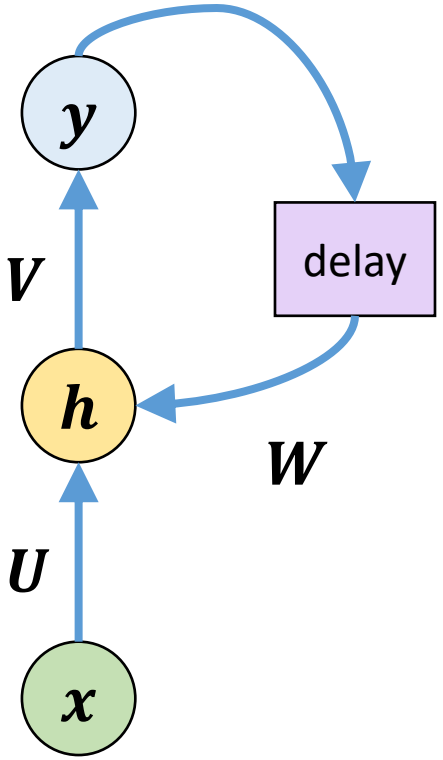


RNN with top-down recurrence



The recurrent connections in the hidden-layer is unfolded and represented in time for processing a time series $\left(x(t)\right)_{t=1}^{T}$

RNN with top-down recurrence: batch processing



Given P patterns $\{\boldsymbol{x}_p\}_{p=1}^{P}$ where $\boldsymbol{x}_p = \left(\boldsymbol{x}_p(t)\right)_{t=1}^{T}$,

$$\boldsymbol{X}(t) = \begin{pmatrix} \boldsymbol{x}_1(\boldsymbol{t})^{\mathsf{T}} \\ \boldsymbol{x}_2(\boldsymbol{t})^{\mathsf{T}} \\ \vdots \\ \boldsymbol{x}_P(\boldsymbol{t})^{\mathsf{T}} \end{pmatrix}$$

Let $\boldsymbol{X}(t)$, $\boldsymbol{Y}(t)$, and $\boldsymbol{H}(t)$ be batch input, output, and hidden output of the network at time t

Activation of the three-layer Jordan-type RNN is given by:

$$\boldsymbol{H}(t) = \phi(\boldsymbol{X}(t)\boldsymbol{U} + \boldsymbol{Y}(t-1)\boldsymbol{W} + \boldsymbol{B})$$
$$\boldsymbol{Y}(t) = \sigma(\boldsymbol{H}(t)\boldsymbol{V} + \boldsymbol{C})$$

Example 2

A recurrent neural network with top-down recurrence receives 2-dimensional input sequences and produce 1-dimensional output sequences. It has three hidden neurons and following weight matrices and biases:

Weight matrix connecting input the hidden layer $\boldsymbol{U} = \begin{pmatrix} -1.0 & 0.5 & 0.2 \\ 0.5 & 0.1 & -2.0 \end{pmatrix}$

Recurrence weight matrix connecting previous output to the hidden layer $\boldsymbol{W} = \begin{pmatrix} 2.0 & 1.3 & -1.0 \end{pmatrix}$

Weight matrix connecting hidden layer to the output $\boldsymbol{V} = \begin{pmatrix} 2.0 \\ -1.5 \\ 0.2 \end{pmatrix}$

Bias to the hidden layer $\boldsymbol{b} = \begin{pmatrix} 0.2 \\ 0.2 \\ 0.2 \end{pmatrix}$ and bias to the output layer $\boldsymbol{c} = 0.1$.

Given the following two input sequences:

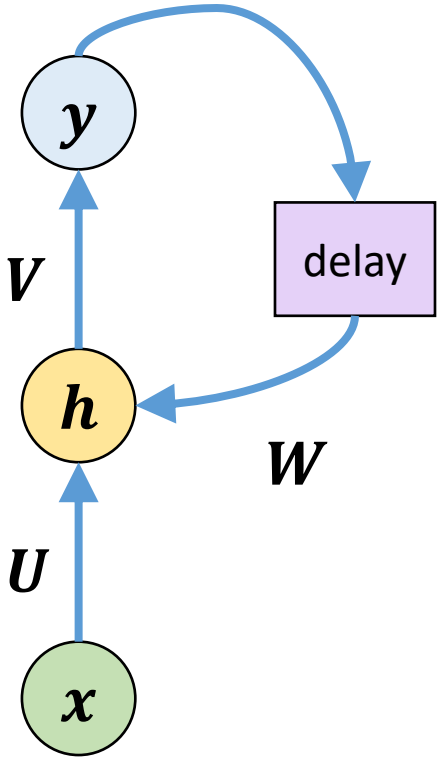
$$\boldsymbol{x}_1 = \left(\begin{pmatrix} 1 \\ 2 \end{pmatrix} \quad \begin{pmatrix} -1 \\ 1 \end{pmatrix} \quad \begin{pmatrix} 0 \\ 3 \end{pmatrix} \right)$$

$$\boldsymbol{x}_2 = \left(\begin{pmatrix} -1 \\ 0 \end{pmatrix} \quad \begin{pmatrix} 2 \\ -1 \end{pmatrix} \quad \begin{pmatrix} 3 \\ -1 \end{pmatrix} \right)$$

Using batch processing, find output sequences. Assume outputs are initialized to zero at the beginning. Assume tanh and sigmoid activations for hidden and output layer, respectlvely.

eg8.2.ipynb

Example 2 (con't)



$$\boldsymbol{x}_1 = \left(\begin{pmatrix}1\\2\end{pmatrix} \quad \begin{pmatrix}-1\\1\end{pmatrix} \quad \begin{pmatrix}0\\3\end{pmatrix}\right)$$
$$\boldsymbol{x}_2 = \left(\begin{pmatrix}-1\\0\end{pmatrix} \quad \begin{pmatrix}2\\-1\end{pmatrix} \quad \begin{pmatrix}3\\-1\end{pmatrix}\right)$$

Two sequences as batch of sequences:

$$\boldsymbol{X} = \left(\begin{pmatrix}1 & 2\\-1 & 0\end{pmatrix} \quad \begin{pmatrix}-1 & 1\\2 & -1\end{pmatrix} \quad \begin{pmatrix}0 & 3\\3 & -1\end{pmatrix}\right)$$

Three hidden neurons and one output neuron.

$$\boldsymbol{H}(t) = \phi(\boldsymbol{X}(t)\boldsymbol{U} + \boldsymbol{Y}(t-1)\boldsymbol{W} + \boldsymbol{B})$$
$$\boldsymbol{Y}(t) = \sigma(\boldsymbol{H}(t)\boldsymbol{V} + \boldsymbol{C})$$

Initially,

$$\boldsymbol{Y}(0) = \begin{pmatrix}0.0\\0.0\end{pmatrix}$$

Example 2 (con't)

$$\text{At } t = 1 : \boldsymbol{X}(1) = \begin{pmatrix} 1 & 2 \\ -1 & 0 \end{pmatrix}$$

$$\begin{aligned} \boldsymbol{H}(1) &= tanh(\boldsymbol{X}(1)\boldsymbol{U} + \boldsymbol{Y}(0)\boldsymbol{W} + \boldsymbol{B}) \\ &= tanh\left(\begin{pmatrix} 1 & 2 \\ -1 & 0 \end{pmatrix}\begin{pmatrix} -1.0 & 0.5 & 0.2 \\ 0.5 & 0.1 & -2.0 \end{pmatrix} + \begin{pmatrix} 0.0 \\ 0.0 \end{pmatrix}\begin{pmatrix} 2.0 & 1.3 & -1.0 \end{pmatrix} + \begin{pmatrix} 0.2 & 0.2 & 0.2 \\ 0.2 & 0.2 & 0.2 \end{pmatrix}\right) \\ &= \begin{pmatrix} 0.2 & 0.72 & -1.0 \\ 0.83 & -0.29 & 0.0 \end{pmatrix} \end{aligned}$$

$$\begin{aligned} \boldsymbol{Y}(1) &= sigmoid(\boldsymbol{H}(1)\boldsymbol{V} + \boldsymbol{C}) \\ &= sigmoid\left(\begin{pmatrix} 0.2 & 0.72 & -1.0 \\ 0.83 & -0.29 & 0.0 \end{pmatrix}\begin{pmatrix} 2.0 \\ -1.5 \\ 0.2 \end{pmatrix} + \begin{pmatrix} 0.1 \\ 0.1 \end{pmatrix}\right) \\ &= \begin{pmatrix} 0.31 \\ 0.90 \end{pmatrix} \end{aligned}$$

Example 2 (con't)

$$\text{At } t = 2 : \boldsymbol{X}(2) = \begin{pmatrix} -1 & 1 \\ 2 & -1 \end{pmatrix}$$

$$\boldsymbol{H}(2) = tanh(\boldsymbol{X}(2)\boldsymbol{U} + \boldsymbol{Y}(1)\boldsymbol{W} + \boldsymbol{B})$$

$$= tanh\left(\begin{pmatrix} -1 & 1 \\ 2 & -1 \end{pmatrix}\begin{pmatrix} -1.0 & 0.5 & 0.2 \\ 0.5 & 0.1 & -2.0 \end{pmatrix} + \begin{pmatrix} 0.31 \\ 0.9 \end{pmatrix}\begin{pmatrix} 2.0 & 1.3 & -1.0 \end{pmatrix} + \begin{pmatrix} 0.2 & 0.2 & 0.2 \\ 0.2 & 0.2 & 0.2 \end{pmatrix}\right)$$

$$= \begin{pmatrix} 0.98 & 0.21 & -0.98 \\ -0.46 & 0.98 & 0.94 \end{pmatrix}$$

$$\boldsymbol{Y}(2) = sigmoid(\boldsymbol{H}(2)\boldsymbol{V} + \boldsymbol{C})$$

$$= sigmoid\left(\begin{pmatrix} 0.98 & 0.21 & -0.98 \\ -0.46 & 0.98 & 0.94 \end{pmatrix}\begin{pmatrix} 2.0 \\ -1.5 \\ 0.2 \end{pmatrix} + \begin{pmatrix} 0.1 \\ 0.1 \end{pmatrix}\right)$$

$$= \begin{pmatrix} 0.83 \\ 0.11 \end{pmatrix}$$

Example 2 (con't)

$$\text{At } t = 3 : \boldsymbol{X}(3) = \begin{pmatrix} 0 & 3 \\ 3 & -1 \end{pmatrix}$$

$$\boldsymbol{H}(3) = tanh(\boldsymbol{X}(3)\boldsymbol{U} + \boldsymbol{Y}(1)\boldsymbol{W} + \boldsymbol{B})$$

$$= tanh\left(\begin{pmatrix} 0 & 3 \\ 3 & -1 \end{pmatrix}\begin{pmatrix} -1.0 & 0.5 & 0.2 \\ 0.5 & 0.1 & -2.0 \end{pmatrix} + \begin{pmatrix} 0.83 \\ 0.11 \end{pmatrix}\begin{pmatrix} 2.0 & 1.3 & -1.0 \end{pmatrix} + \begin{pmatrix} 0.2 & 0.2 & 0.2 \\ 0.2 & 0.2 & 0.2 \end{pmatrix}\right)$$

$$= \begin{pmatrix} 1.0 & 0.92 & -1.0 \\ -1.00 & 0.94 & 0.99 \end{pmatrix}$$

$$\boldsymbol{Y}(3) = sigmoid(\boldsymbol{H}(3)\boldsymbol{V} + \boldsymbol{C})$$

$$= sigmoid\left(\begin{pmatrix} 1.0 & 0.92 & -1.0 \\ -1.0 & 0.94 & 0.99 \end{pmatrix}\begin{pmatrix} 2.0 \\ -1.5 \\ 0.2 \end{pmatrix} + \begin{pmatrix} 0.1 \\ 0.1 \end{pmatrix}\right)$$

$$= \begin{pmatrix} 0.63 \\ 0.04 \end{pmatrix}$$

Example 2 (con't)

Output (batch):

$$\boldsymbol{Y} = \begin{pmatrix} \boldsymbol{Y}(1) & \boldsymbol{Y}(2) & \boldsymbol{Y}(3) \end{pmatrix}$$
$$= \left(\begin{pmatrix} 0.31 \\ 0.9 \end{pmatrix} \quad \begin{pmatrix} 0.83 \\ 0.11 \end{pmatrix} \quad \begin{pmatrix} 0.63 \\ 0.04 \end{pmatrix} \right)$$

Outputs for inputs

$$x_1 = \left(\begin{pmatrix} 1 \\ 2 \end{pmatrix} \quad \begin{pmatrix} -1 \\ 1 \end{pmatrix} \quad \begin{pmatrix} 0 \\ 3 \end{pmatrix} \right)$$
$$x_2 = \left(\begin{pmatrix} -1 \\ 0 \end{pmatrix} \quad \begin{pmatrix} 2 \\ -1 \end{pmatrix} \quad \begin{pmatrix} 3 \\ -1 \end{pmatrix} \right)$$

are

$$y_1 = (0.31, 0.83, 0.63)$$
$$y_2 = (0.9, 0.11, 0.04)$$

Backpropagation through time (BPTT)

Let's consider vanilla-RNN with hidden recurrence.
Forward propagation equations:

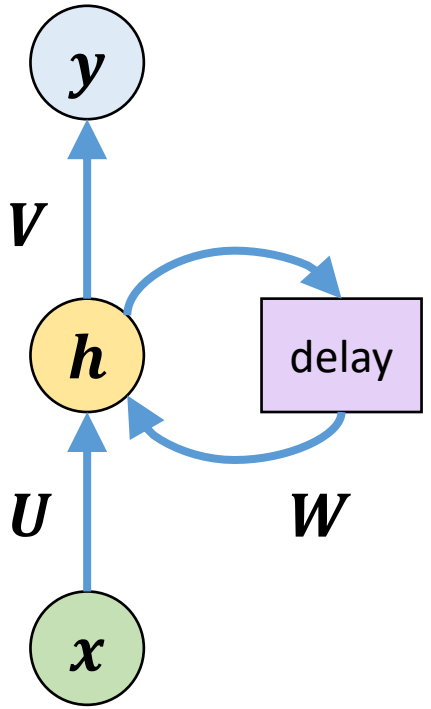
$$\boldsymbol{h}(t) = tanh\left(\boldsymbol{U}^{\mathsf{T}}\boldsymbol{x}(t) + \boldsymbol{W}^{\mathsf{T}}\boldsymbol{h}(t-1) + \boldsymbol{b}\right)$$
$$\boldsymbol{u}(t) = \boldsymbol{V}^{\mathsf{T}}\boldsymbol{h}(t) + \boldsymbol{c}$$

For classification,

$$\boldsymbol{y}(t) = softmax\left(\boldsymbol{u}(t)\right)$$

For regression,

$$\boldsymbol{y}(t) = \boldsymbol{u}(t)$$



Learnable parameters

- $\boldsymbol{U}$: weight vector that transforms raw inputs to the hidden-layer
- $\boldsymbol{W}$: recurrent weight vector connecting previous hidden-layer output to hidden input
- $\boldsymbol{V}$: weight vector of the output layer
- $\boldsymbol{b}$: bias connected to hidden layer
- $\boldsymbol{c}$: bias connected to the output layer

Backpropagation through time (BPTT)

Let's consider vanilla-RNN with hidden recurrence.
Forward propagation equations:

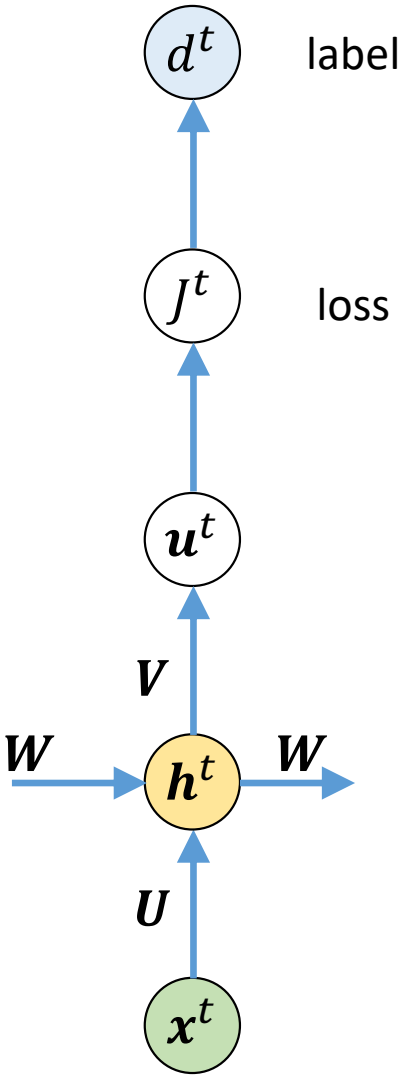
$$\boldsymbol{h}(t) = tanh\left(\boldsymbol{U}^{\mathsf{T}}\boldsymbol{x}(t) + \boldsymbol{W}^{\mathsf{T}}\boldsymbol{h}(t-1) + \boldsymbol{b}\right)$$
$$\boldsymbol{u}(t) = \boldsymbol{V}^{\mathsf{T}}\boldsymbol{h}(t) + \boldsymbol{c}$$

For classification,

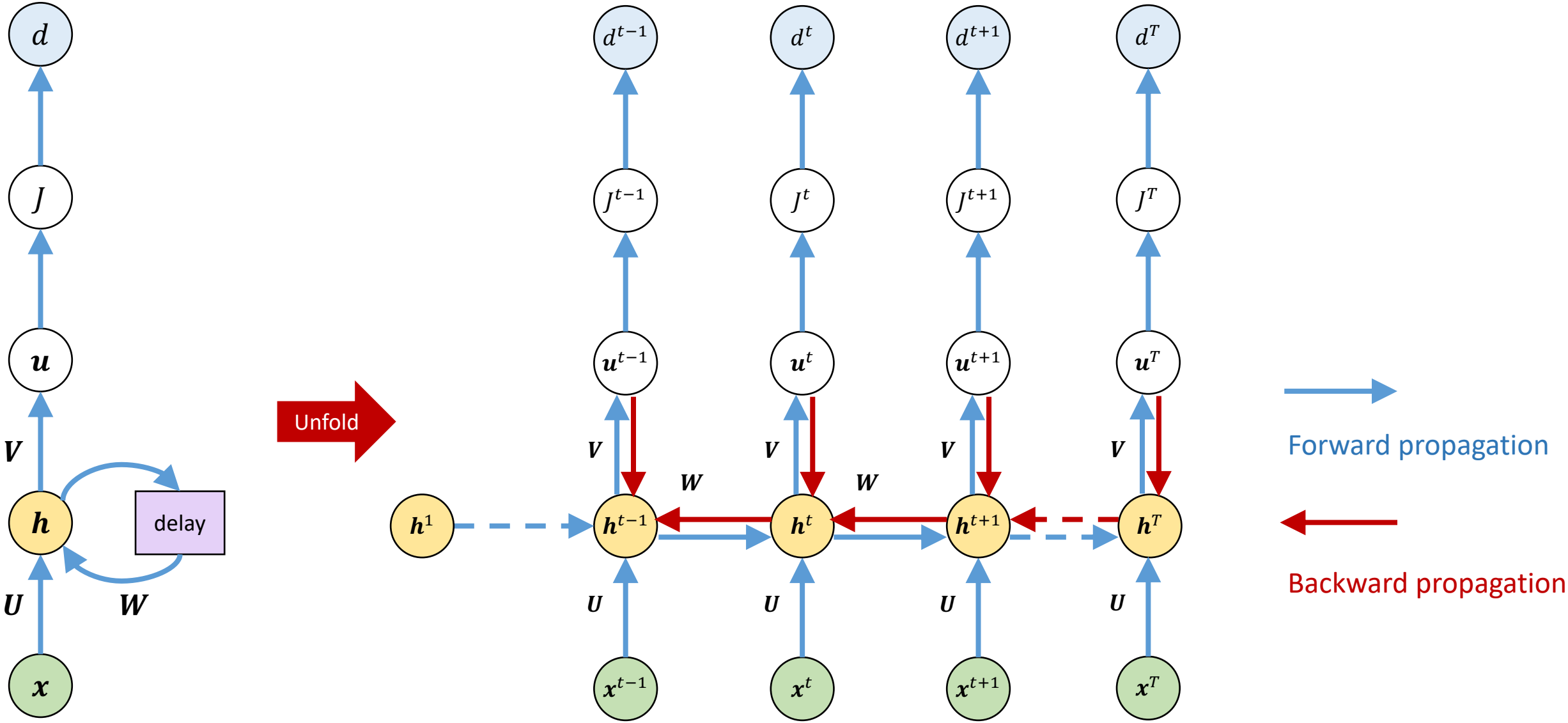
$$\boldsymbol{y}(t) = softmax\left(\boldsymbol{u}(t)\right)$$

For regression,

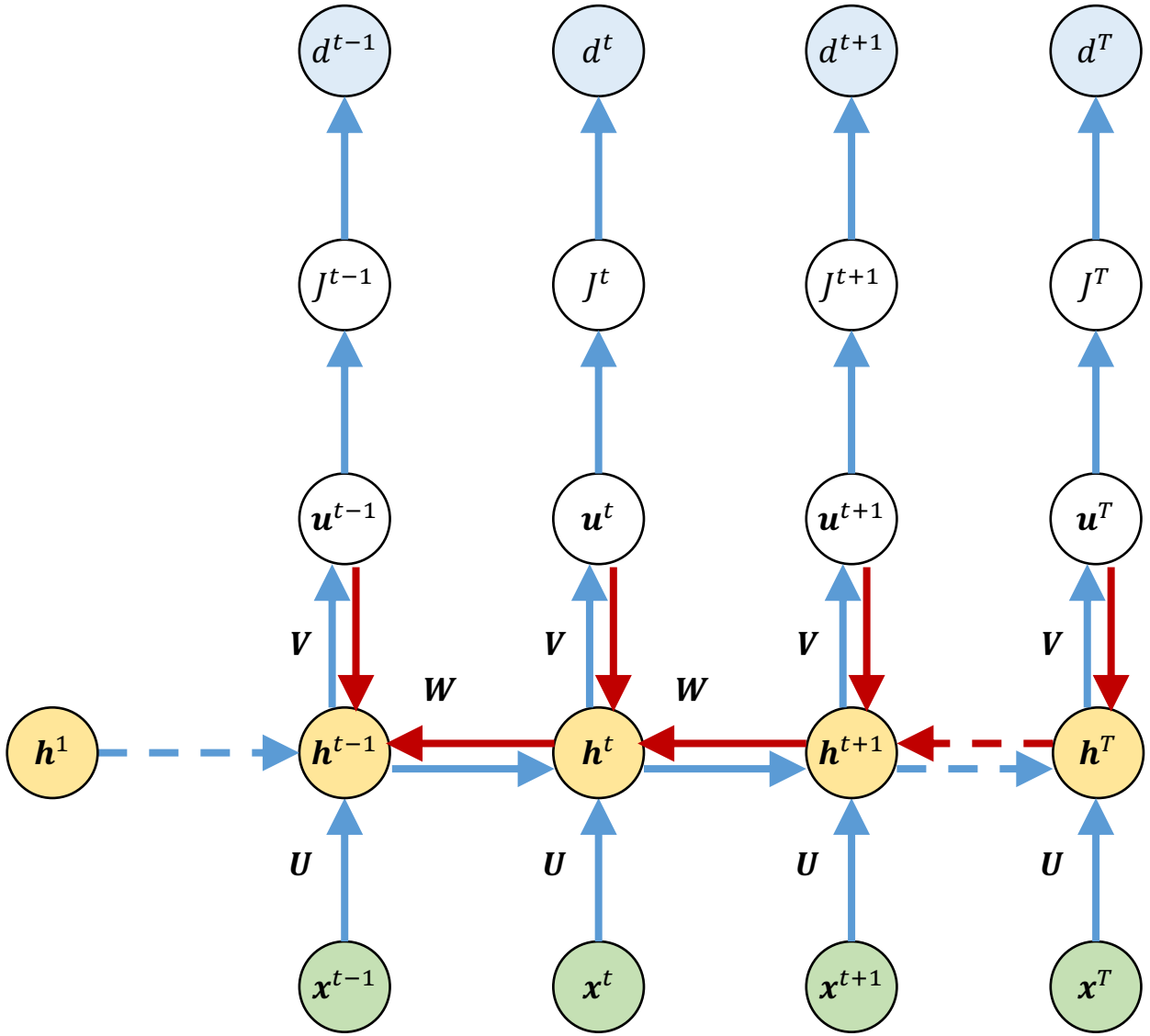
$$\boldsymbol{y}(t) = \boldsymbol{u}(t)$$



Backpropagation through time (BPTT)



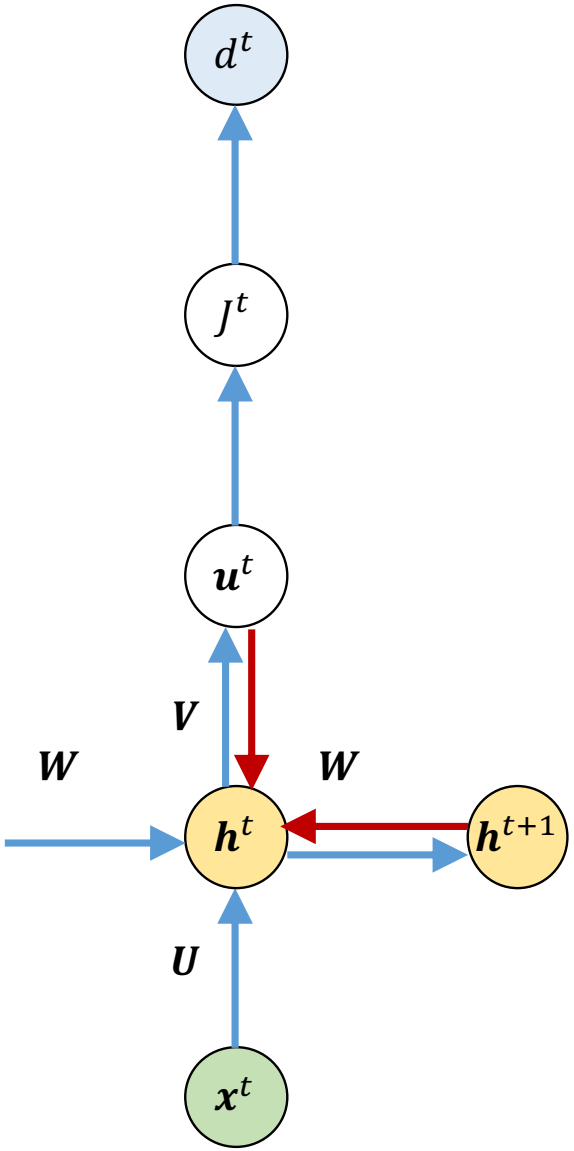
Backpropagation through time (BPTT)



The gradients propagate backward, starting from the end of the sequence from two directions:

- Top-down direction
- Reverse sequence direction

Backpropagation through time (BPTT)



The gradient computation involves performing a forward propagation pass moving left to right through the unfolded graph, followed by a backpropagation pass moving right to left through the graph. The gradients are propagated from the final time point to the initial time points .

The runtime is $O(T)$ where T is the length of input sequence and cannot be reduced by parallelization because the forward propagation is inherently sequential. The back-propagation algorithm applied to the unrolled graph with $O(T)$ cost is called *back-propagation through time (BPTT).*

The network with recurrence between hidden units is thus very powerful but also expensive to train.

Example 3

Generate 8-dimensional 16 input sequences of 64 time-steps with each input is a random number between [0.0, 1.0].

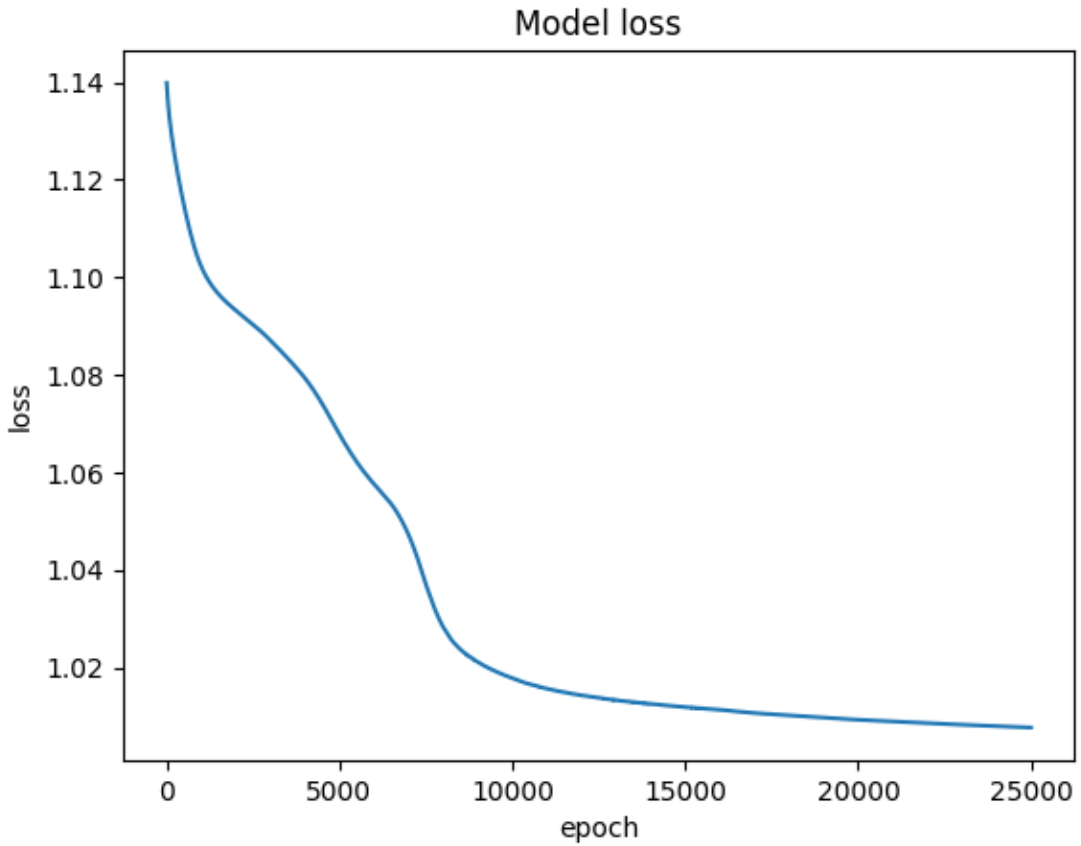
Generate the corresponding 1-dimensional labels by randomly generating a number from [0, 1, 2].

Create an RNN with one hidden layer of 5 neurons.

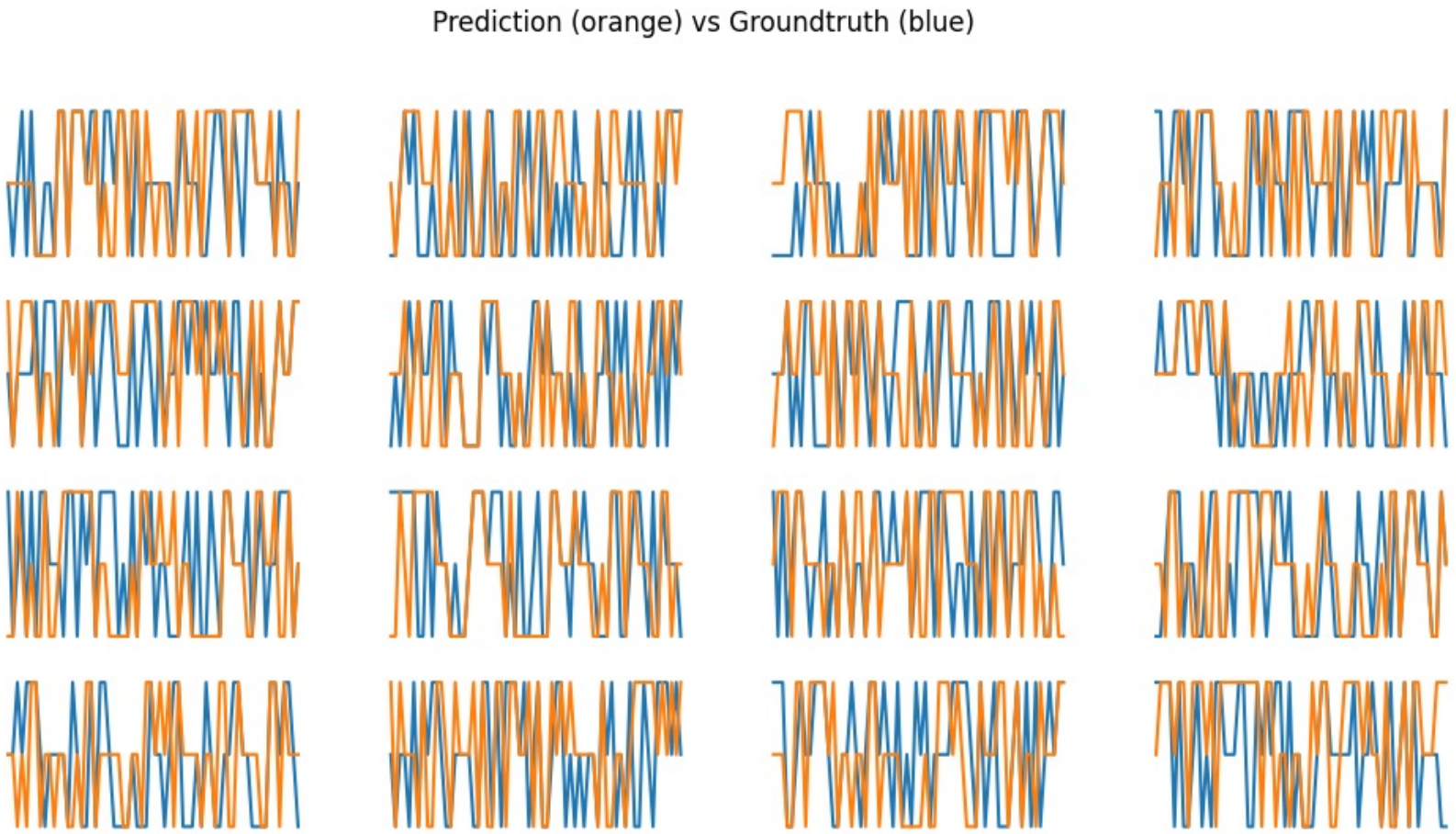
Plot the learning curves and predicted labels.

eg8.3.ipynb

Example 3 (con’t)



Example 3 (con’t)



Long Short-Term Memory (LSTM)

Long-term dependency

"The man who ate my pizza has purple hair"

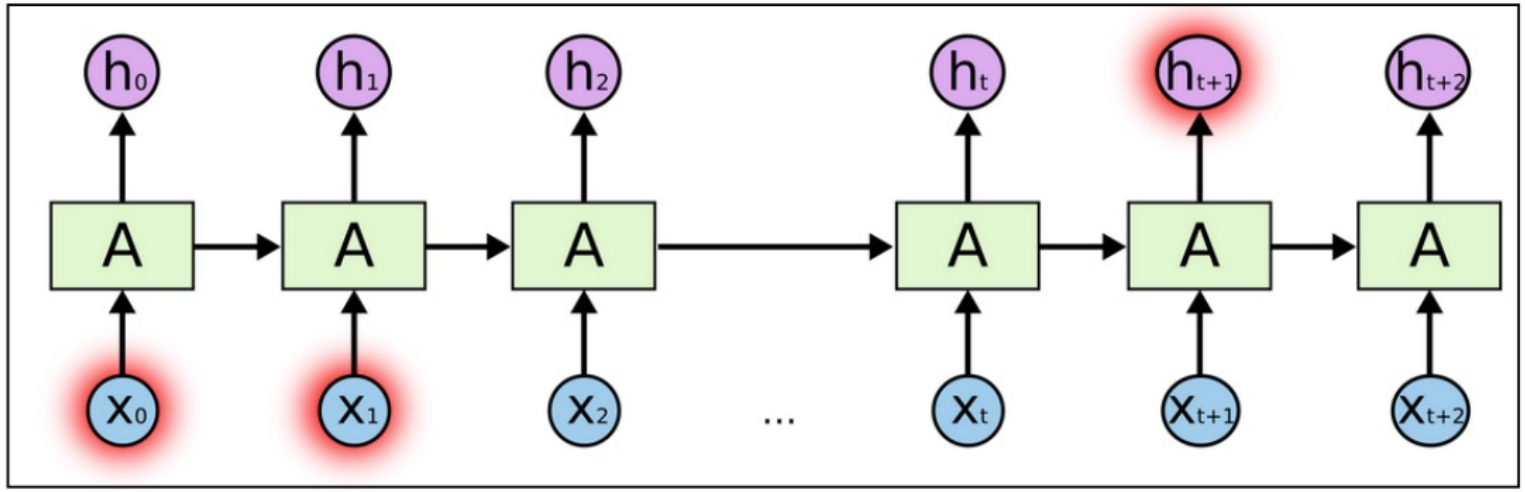
Purple hair is for the man and not the pizza!
This is a long term dependency.

There are cases where we need even more context

- To predict last word in "I grew up in France…<long paragraph>…I speak *French*"
- Using only recent information suggests that the last word is the name of a language. But more distant past indicates that it is *French*

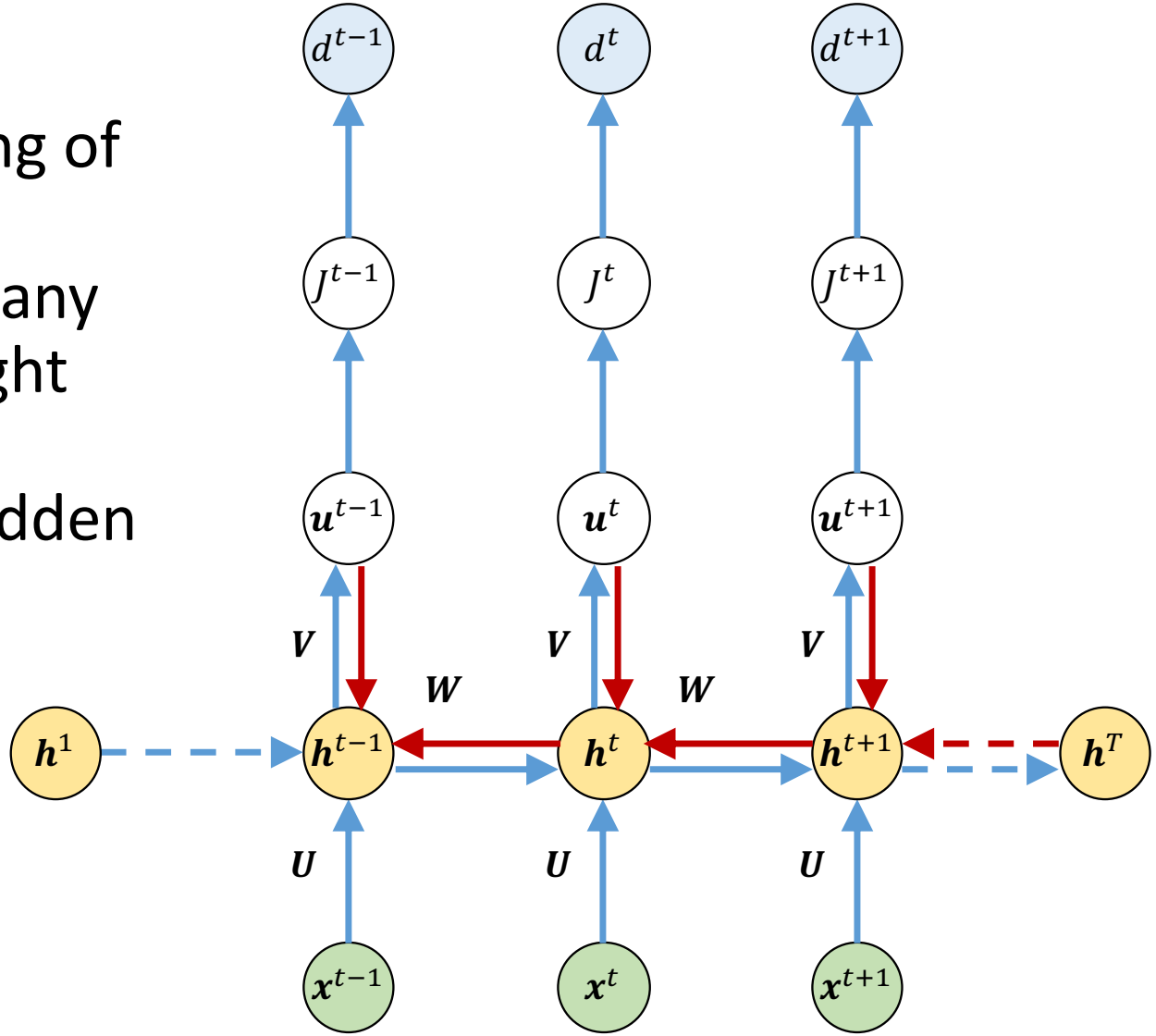
Long-term dependency

- RNNs work upon the fact that the result of an information is dependent on its previous state or previous n time steps
- RNNs have difficulty in learning long range dependencies
- Gap between relevant information and where it is needed is large



Exploding and vanishing gradients in RNN

During gradient back-propagation learning of RNN, the gradient can end up being multiplied a large number of times (as many as the number of time steps) by the weight matrix associated with the connections between the neurons of the recurrent hidden layer.

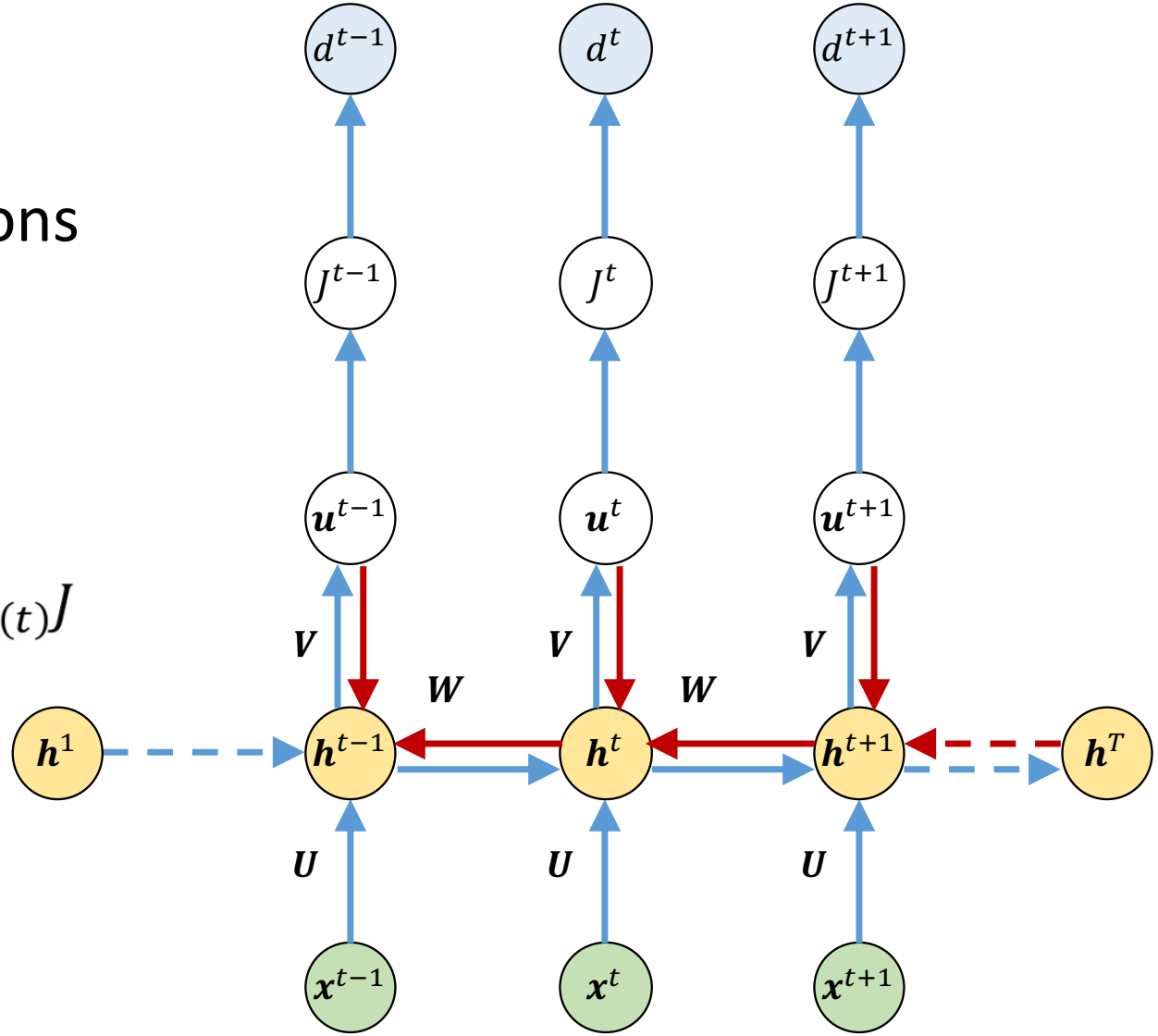


Exploding and vanishing gradients in RNN

Note that each time the activations are forward propagated in time, the activations are multiplied by $\boldsymbol{W}$ and each time the gradients are back propagated, the gradients are multiplied by $\boldsymbol{W}^{\top}$.

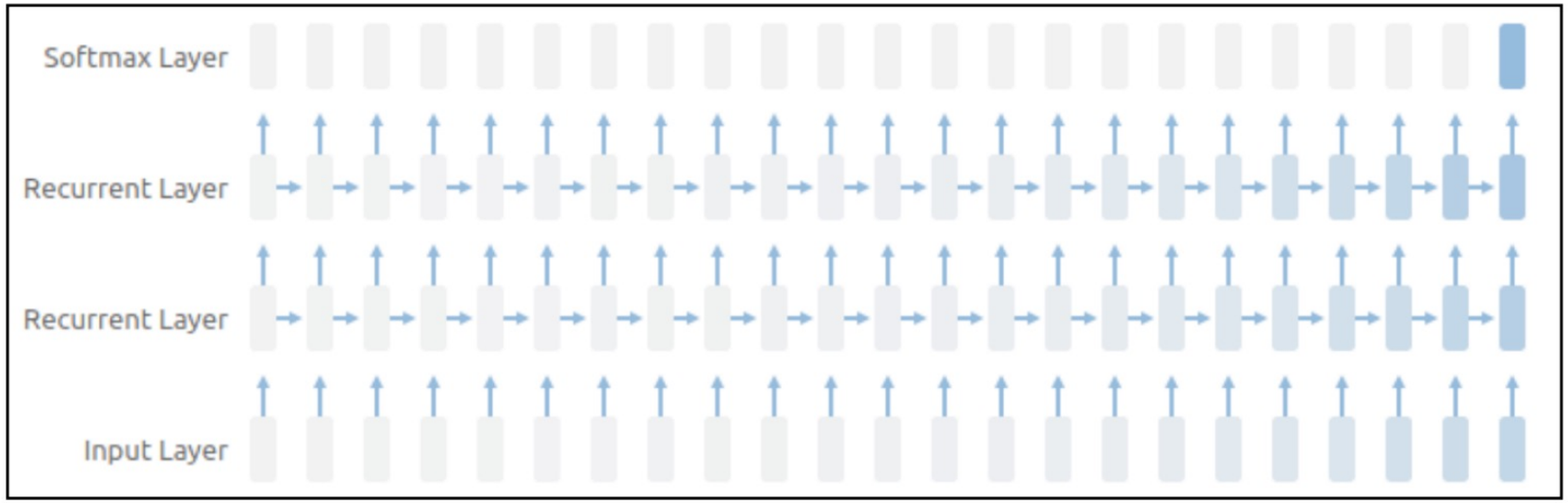
$$\nabla_{\boldsymbol{h}(t)} J = \boldsymbol{W}\,\mathrm{diag}\left(1 - \boldsymbol{h}^2(t+1)\right) \nabla_{\boldsymbol{h}(t+1)} J + \boldsymbol{V}\nabla_{u(t)} J$$

Gradient from reverse direction



Exploding and vanishing gradients in RNN

If the weights in this matrix are small, the recursive derivative can lead to a situation called **vanishing gradients** where the gradient signal gets so small that learning either becomes very slow or stops working altogether



Contribution from the earlier steps becomes insignificant in the gradient

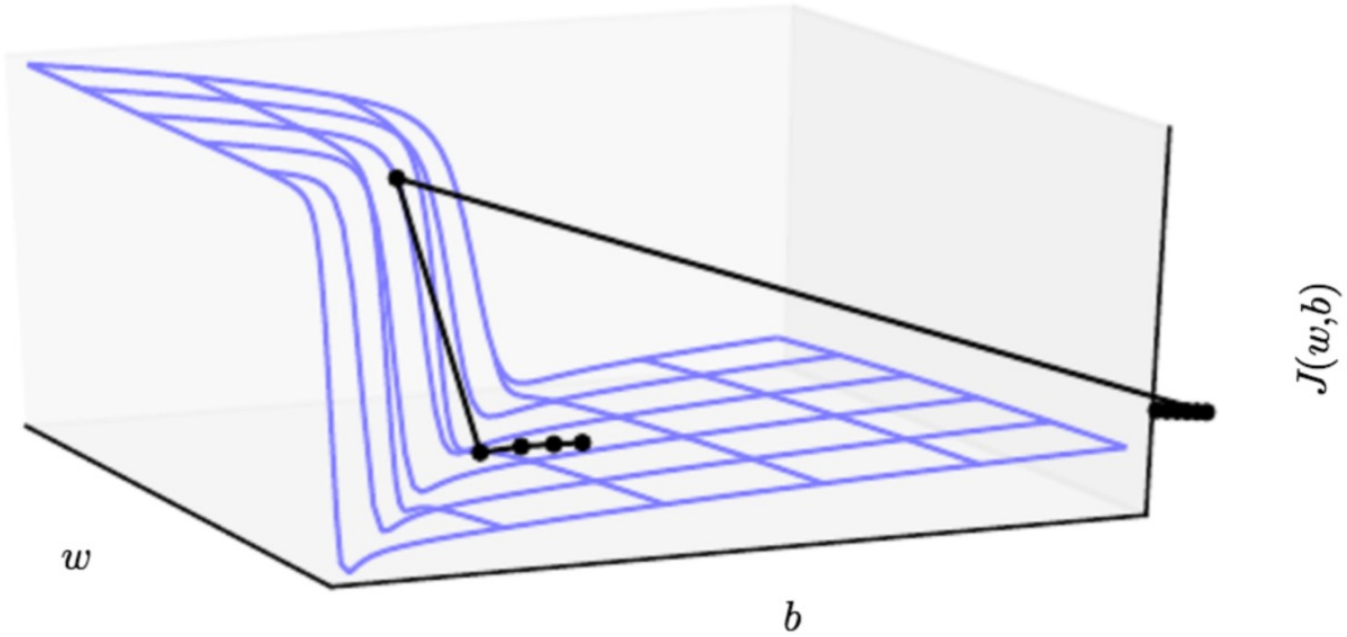
Exploding and vanishing gradients in RNN

Conversely, if the weights in this matrix are large, it can lead to a situation where the gradient signal is so large that it can cause learning to diverge. This is often referred to as **exploding gradients**.

Exploding gradient easily solved by clipping the gradients at a predefined threshold value.

Vanishing gradient is more concerning

Gradient Clipping



Due to long term dependencies, RNN tend to have gradients having very large or very small magnitudes. The large gradients resemble cliffs in the error landscape and when the gradient descent encounters the gradient updates can move the parameters away from true minimum.

A gradient clipping is employed to avoid the gradients to become too large.

Gradient Clipping

Commonly, two methods are used:

1. Clip the gradient g when it exceeds a threshold:

$$\text{if} \|g\| > \nu\text{:}$$
$$\|g\| \leftarrow \nu$$

2. Normalize the gradient when it exceeds a threshold:

$$\text{if} \|g\| > \nu\text{:}$$
$$\|g\| \leftarrow \frac{g}{\|g\|} \nu$$

Exploding and vanishing gradients in RNN

Vanishing and exploding gradients in gradient backpropagation learning makes it difficulty train RNN to learn long-term dependencies.

Solution: Gated RNNs

- Long short-term memory (LSTM), Gated Recurrent Units (GRU)

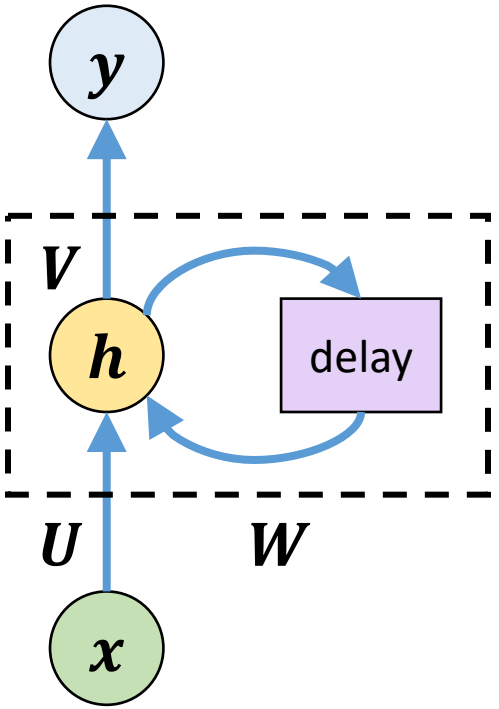
Basic RNN unit

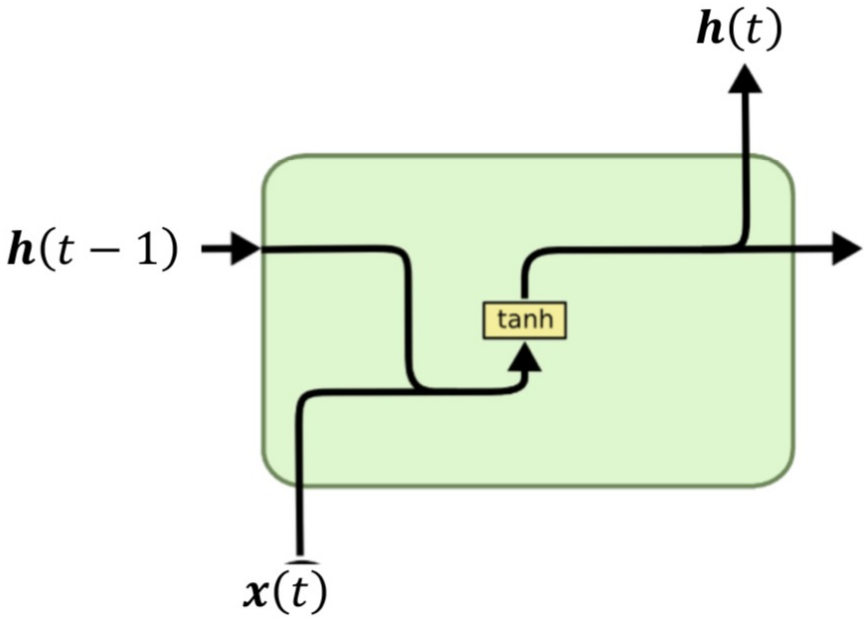
The RNN cell (memory unit) is characterized by

$$\boldsymbol{h}(t) = \phi\left(\boldsymbol{U}^{\mathsf{T}}\boldsymbol{x}(t) + \boldsymbol{W}^{\mathsf{T}}\boldsymbol{h}(t-1) + \boldsymbol{b}\right)$$

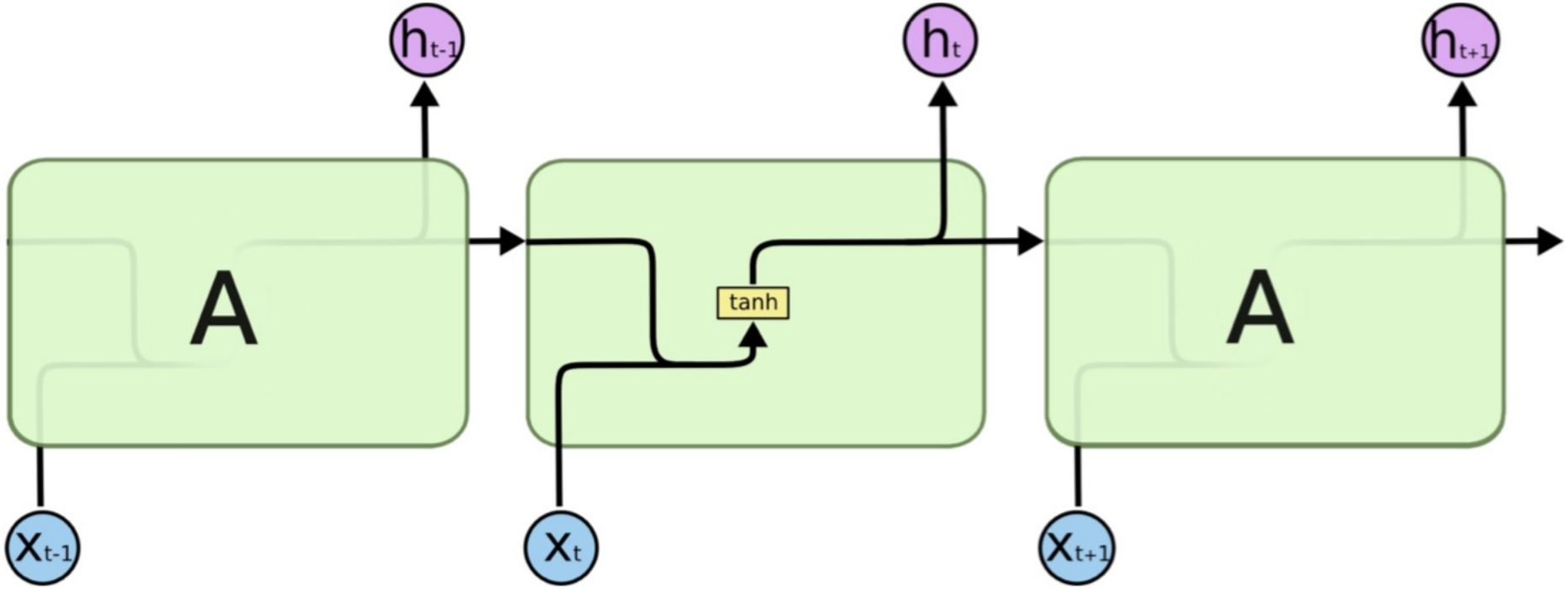
where ϕ is the *tanh* activation function and RNN cell is referred to as *tanh* units.

RNN build with simple *tanh* units are also referred to as vanilla RNN.





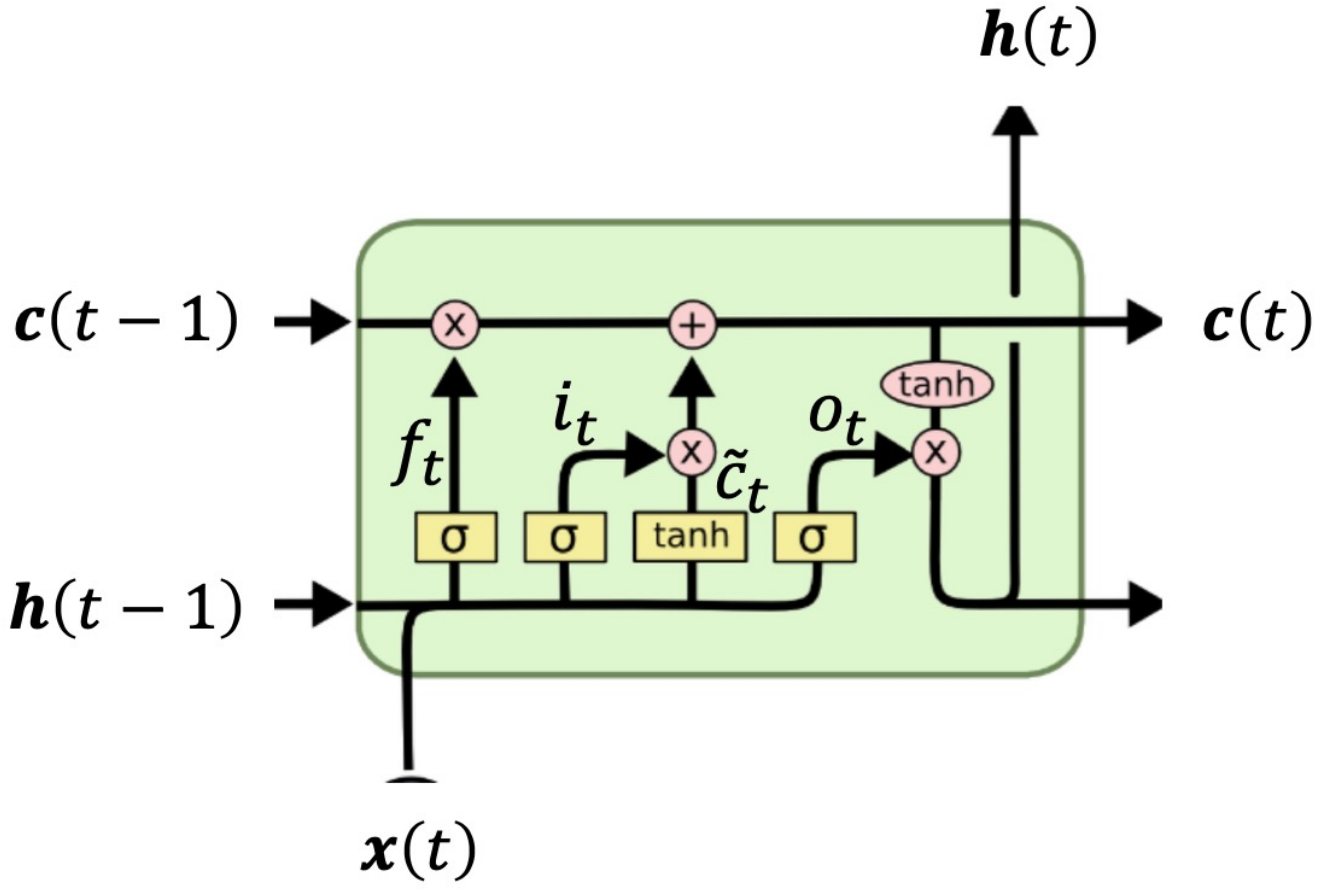
Basic RNN layer



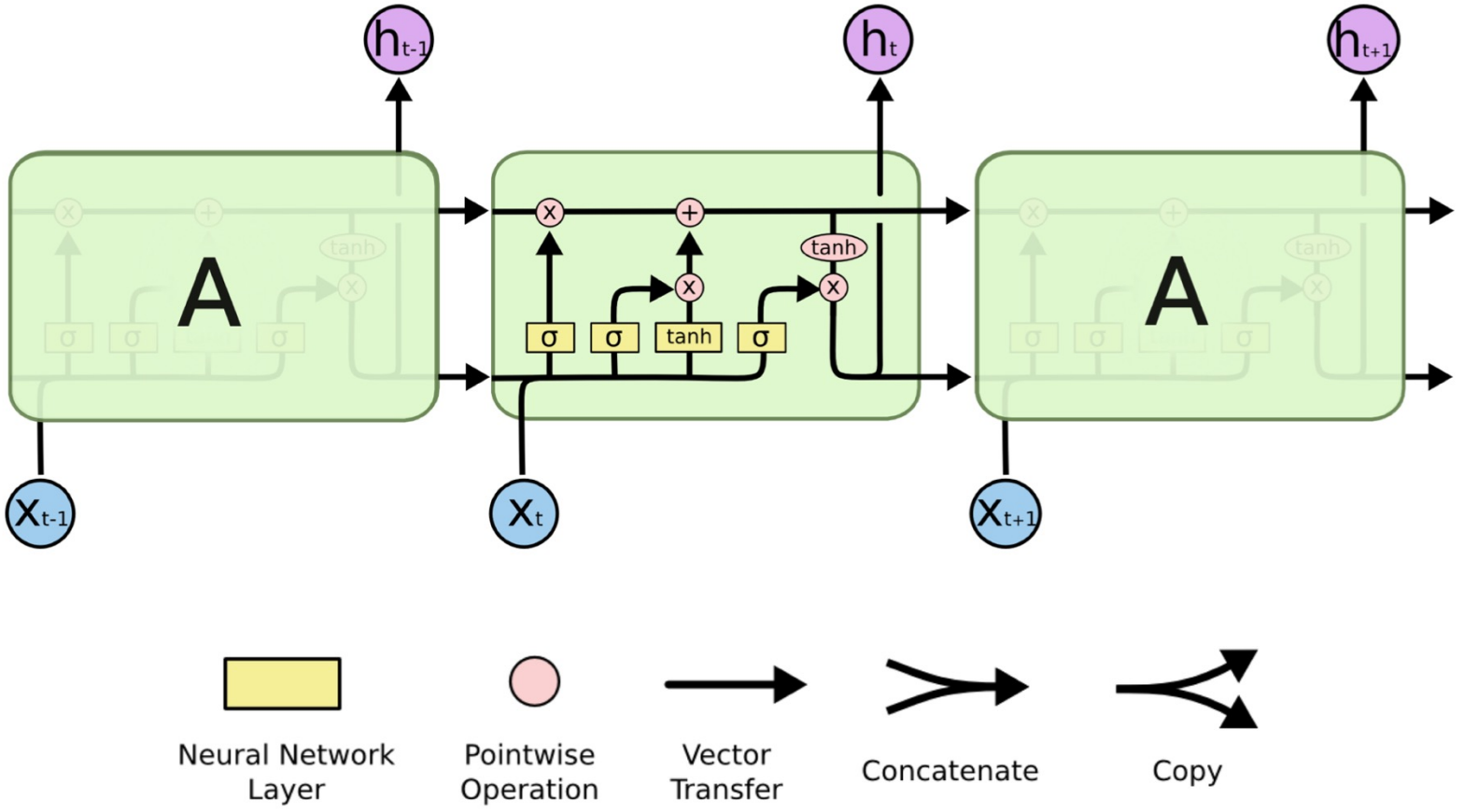
Long short-term memory (LSTM) unit

Key of LSTM is ”state”

- A persistent module called the cell-state
- “State” is a representation of past history
- It comprises a common thread through time



Long short-term memory (LSTM) unit

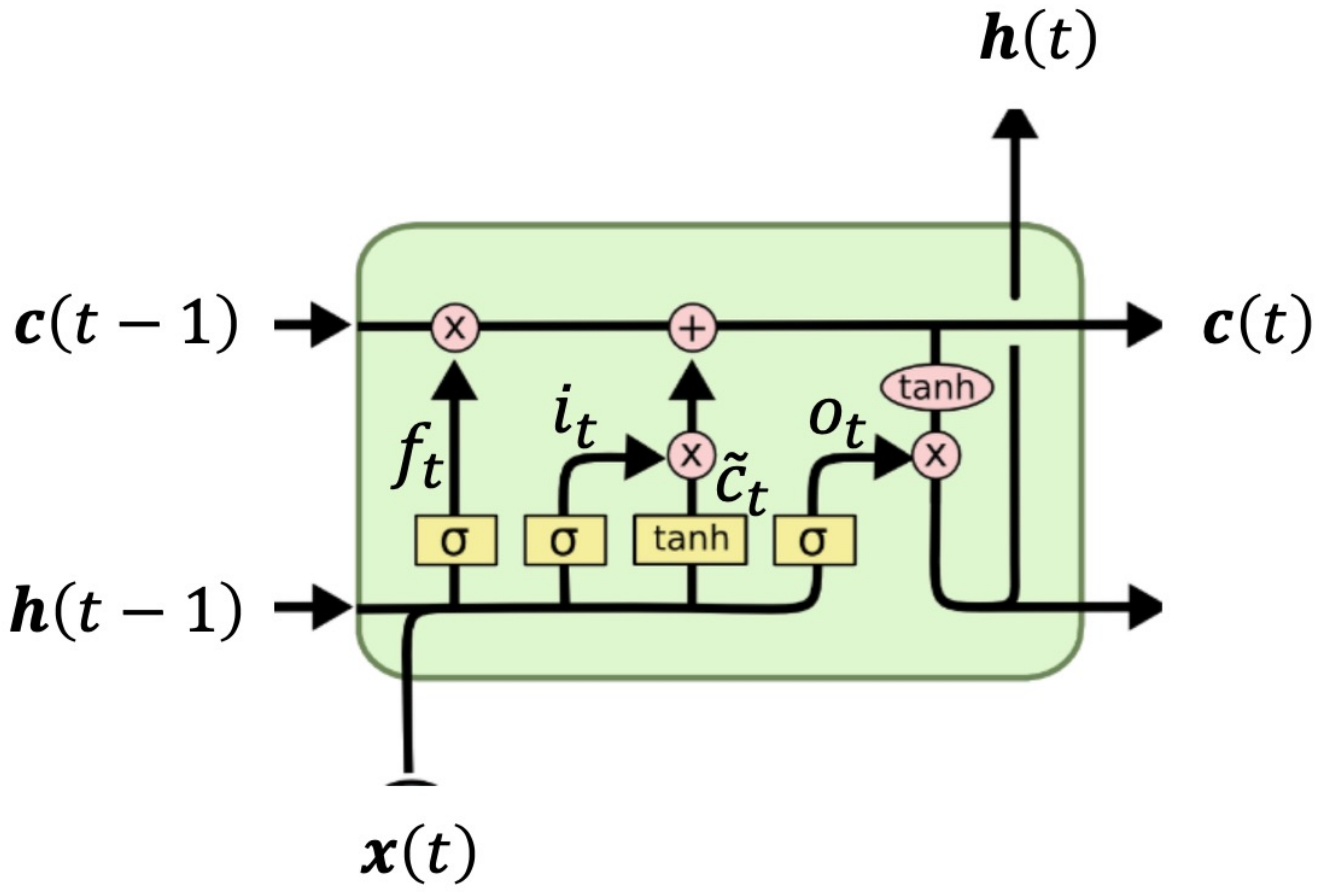


Cells are connected recurrently to each other - Replacing hidden units of ordinary recurrent networks

Long short-term memory (LSTM) unit

LSTMs provide a solution by incorporating memory units that allow the network to learn when to forget previous hidden states and when to update hidden states given new information.

Instead of having a single neural network layer, there are four, interacting in a very special way.



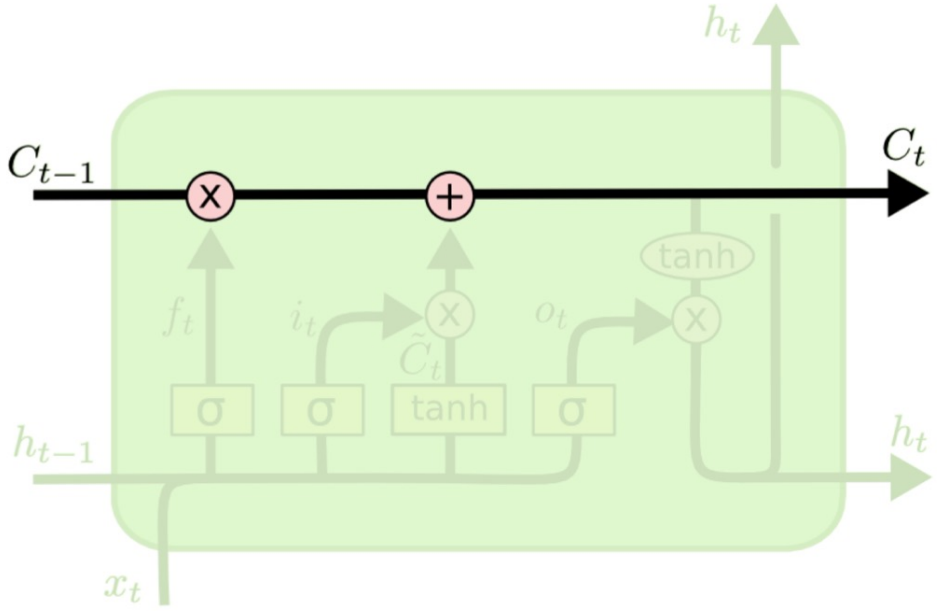
Hochreiter & Schmidhuber (1997)

Long short-term memory (LSTM) unit

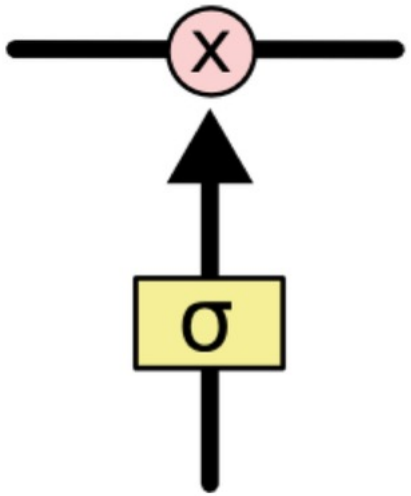
The key to LSTM is the cell state $c(t)$ - the horizontal line through the top of the diagram.

The long-term memory.

Like a conveyor belt that runs through entire chain with minor interactions



Gates



The LSTM has the ability to add and remove information to the cell states through gates.

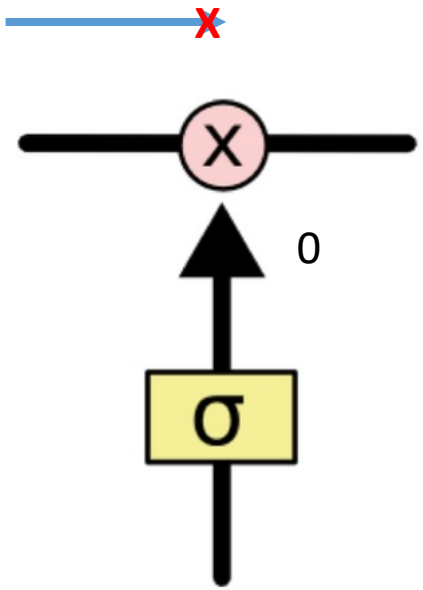
Gates are a way to optionally let information through.

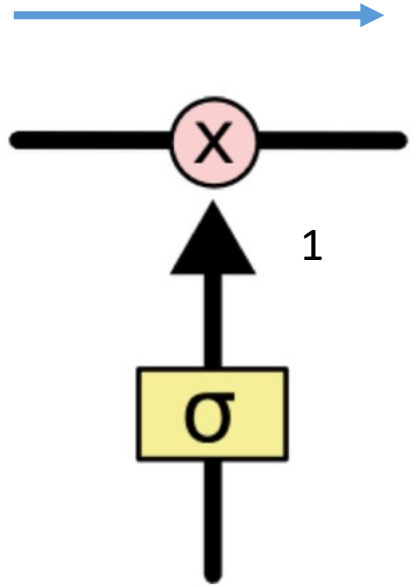
They are composed of sigmoid neural net layer and pointwise multiplication operations.

Gates

The sigmoid layer outputs numbers between zero and one, describing how much of each component should be let through.

A value of zero means “let nothing through,” while a value of one means “let everything through”





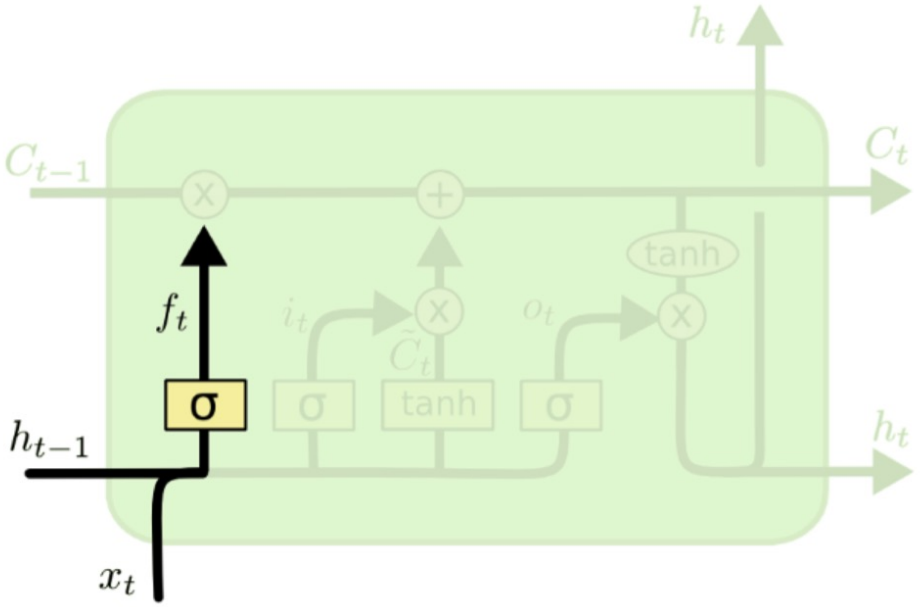
An LSTM has three of these gates to control cell state.

Forget gate

The forget gate can modulate the memory cell's self-recurrent connection, allowing the cell to remember or forget its previous state, as needed.

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

Value of $\boldsymbol{f}(t)$ determines if $c(t-1)$ is to be remembered or not.



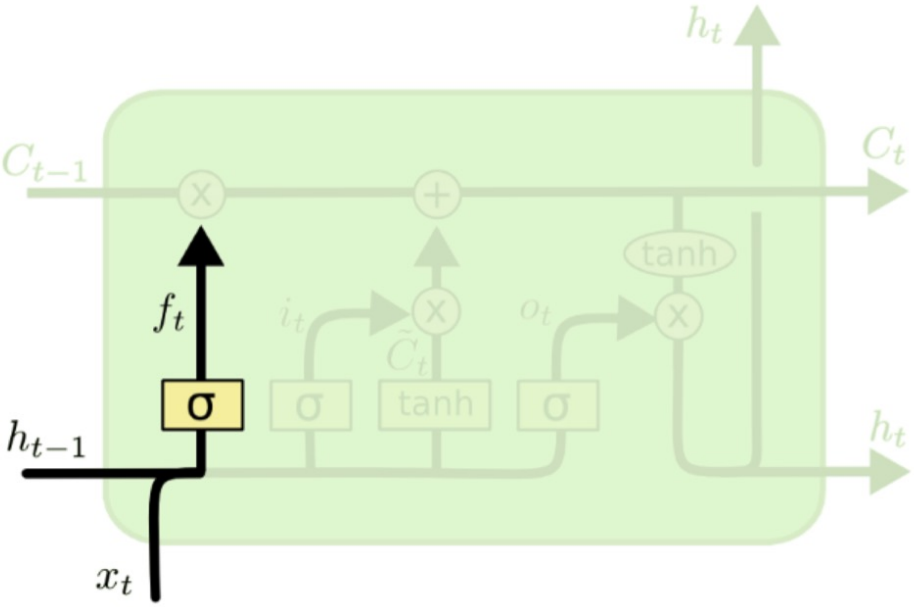
Forget gate

Example: Language modelling

Consider trying to predict the next word based on all previous ones

The cell state may include the gender of the present subject so that the proper pronouns can be used

When we see a new subject we want to forget old subject



Input gate

The input gate can allow incoming signal to alter the state of the memory cell or block it. It decides what new information to store in the cell stage.

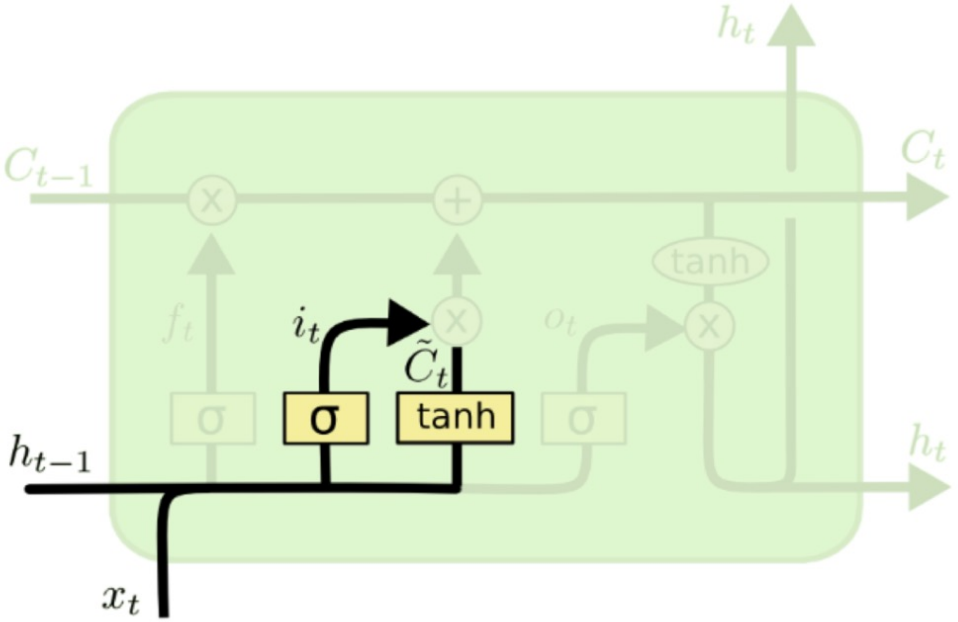
This has two parts:

A sigmoid input gate layer decides which values to update;

A tanh layer creates a vector of new candidate values $\tilde{\boldsymbol{c}}(t)$ that could be added to the state.

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

$$\tilde{\boldsymbol{c}}(t) = \phi\left(\boldsymbol{U}_{\boldsymbol{c}}^{\top}\boldsymbol{x}(t) + \boldsymbol{W}_{\boldsymbol{c}}^{\top}\boldsymbol{h}(t-1) + \boldsymbol{b}_{\boldsymbol{c}}\right)$$



Example: Language modeling
We'd want to add the gender of the new subject to the cell state, to replace the old one we are forgetting

Cell state

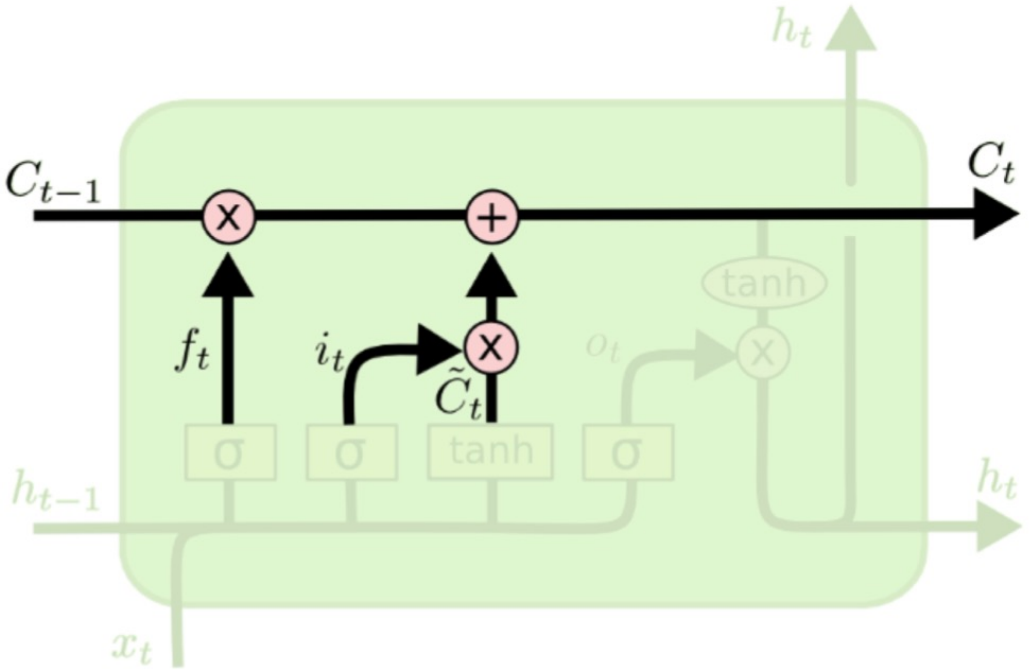
a vector of new candidate values

which values to update

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

input gate

forget gate



Example: Language modeling

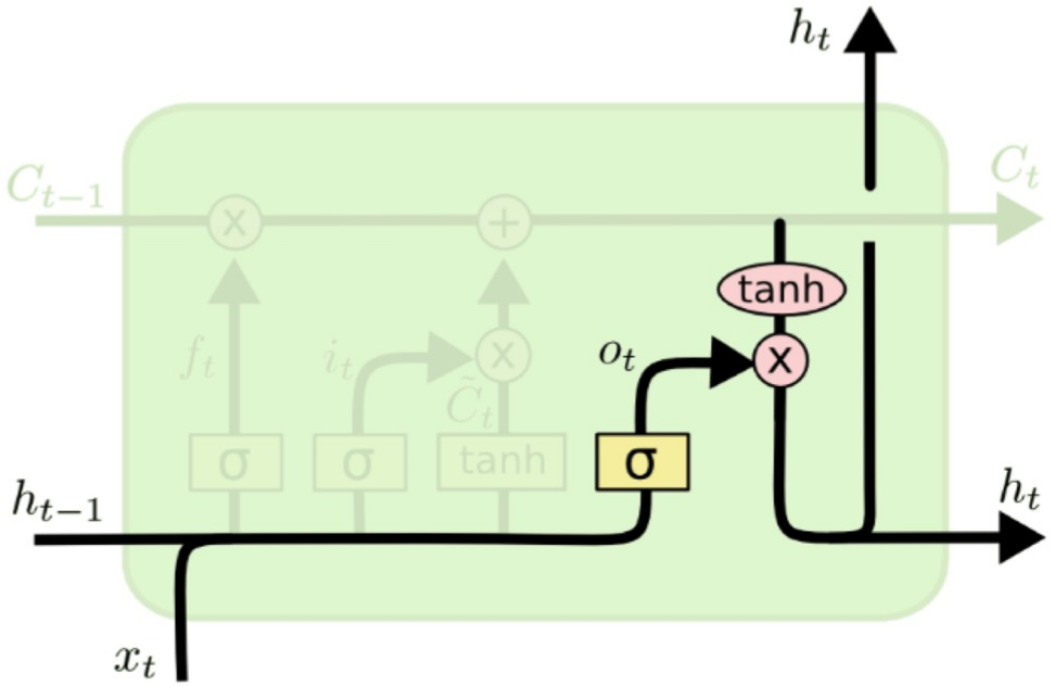
In the Language model, this is where we'd actually drop the information about the old subject's gender and add the new information, as we decided in previous steps

Output gate

The output gate can allow the state of the memory cell to have an effect on other neurons or prevent it.

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

$$\boldsymbol{h}(t) = \phi\left(\boldsymbol{c}(t)\right) \odot \boldsymbol{o}(t)$$

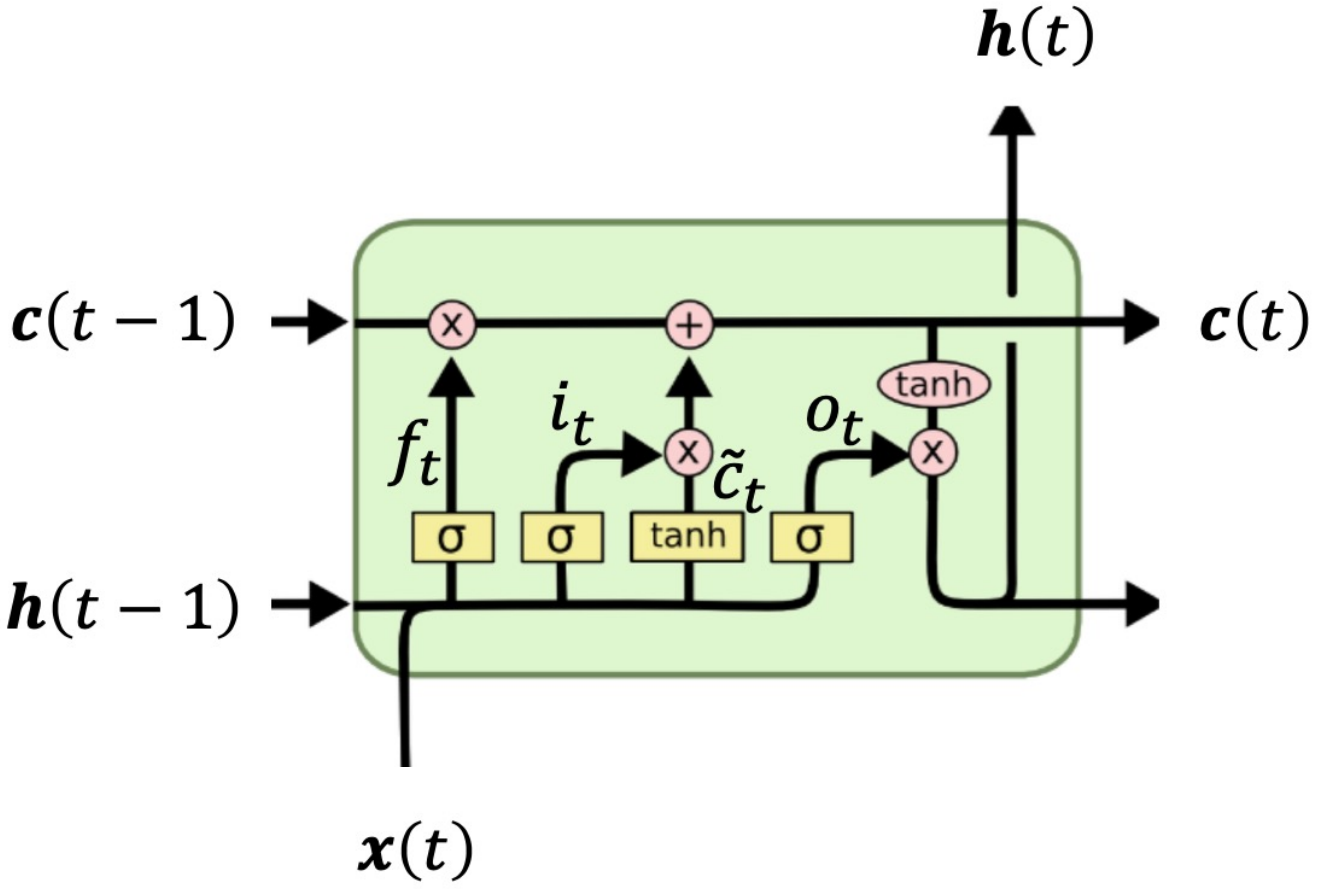


LSTM unit

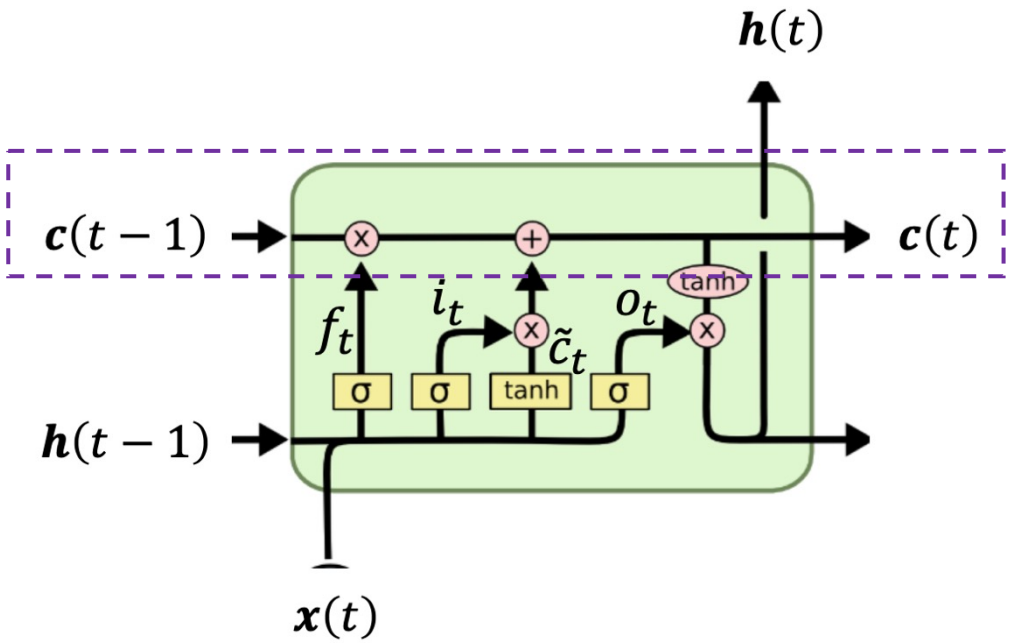
$$\boldsymbol{i}(t) = \sigma\left(\boldsymbol{U}_{\boldsymbol{i}}^{\top}\boldsymbol{x}(t) + \boldsymbol{W}_{\boldsymbol{i}}^{\top}\boldsymbol{h}(t-1) + \boldsymbol{b}_{\boldsymbol{i}}\right)$$
$$\boldsymbol{f}(t) = \sigma\left(\boldsymbol{U}_{\boldsymbol{f}}^{\top}\boldsymbol{x}(t) + \boldsymbol{W}_{\boldsymbol{f}}^{\top}\boldsymbol{h}(t-1) + \boldsymbol{b}_{\boldsymbol{f}}\right)$$
$$\boldsymbol{o}(t) = \sigma\left(\boldsymbol{U}_{\boldsymbol{o}}^{\top}\boldsymbol{x}(t) + \boldsymbol{W}_{\boldsymbol{o}}^{\top}\boldsymbol{h}(t-1) + \boldsymbol{b}_{\boldsymbol{o}}\right)$$

$$\tilde{\boldsymbol{c}}(t) = \phi\left(\boldsymbol{U}_{\boldsymbol{c}}^{\top}\boldsymbol{x}(t) + \boldsymbol{W}_{\boldsymbol{c}}^{\top}\boldsymbol{h}(t-1) + \boldsymbol{b}_{\boldsymbol{c}}\right)$$

$$\boldsymbol{c}(t) = \tilde{\boldsymbol{c}}(t) \odot \boldsymbol{i}(t) + \boldsymbol{c}(t-1) \odot \boldsymbol{f}(t)$$
$$\boldsymbol{h}(t) = \phi\left(\boldsymbol{c}(t)\right) \odot \boldsymbol{o}(t)$$



LSTM unit

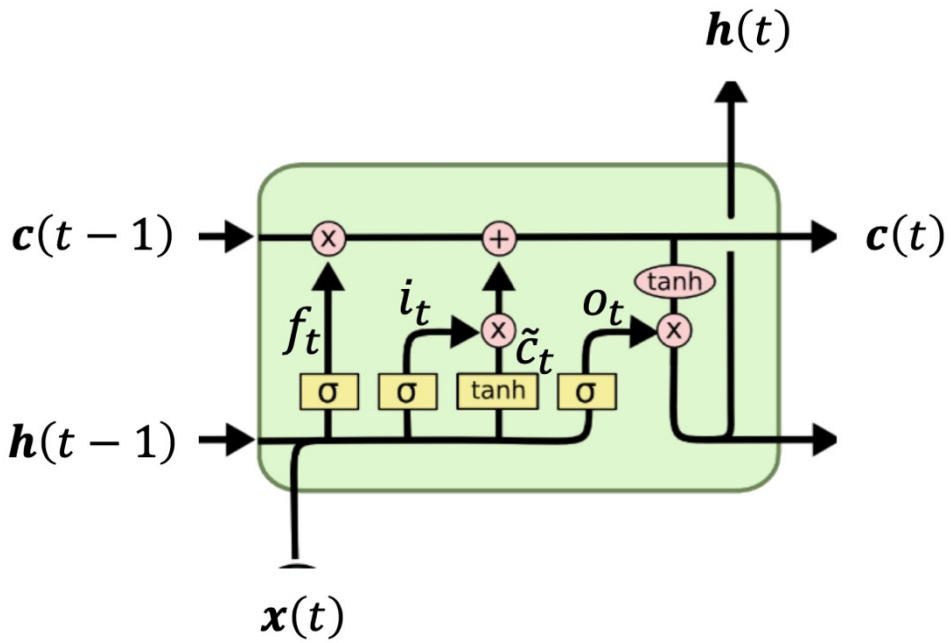


LSTM introduces a gated cell where the information flow can be controlled.

The most important component is the state unit $c_i(t)$ (for time step t and cell i) which has a **linear self- loop**.

The self-loop weight is controlled by a forget gate unit, which sets this weight to a value between 0 and 1 via a sigmoid unit.

LSTM unit



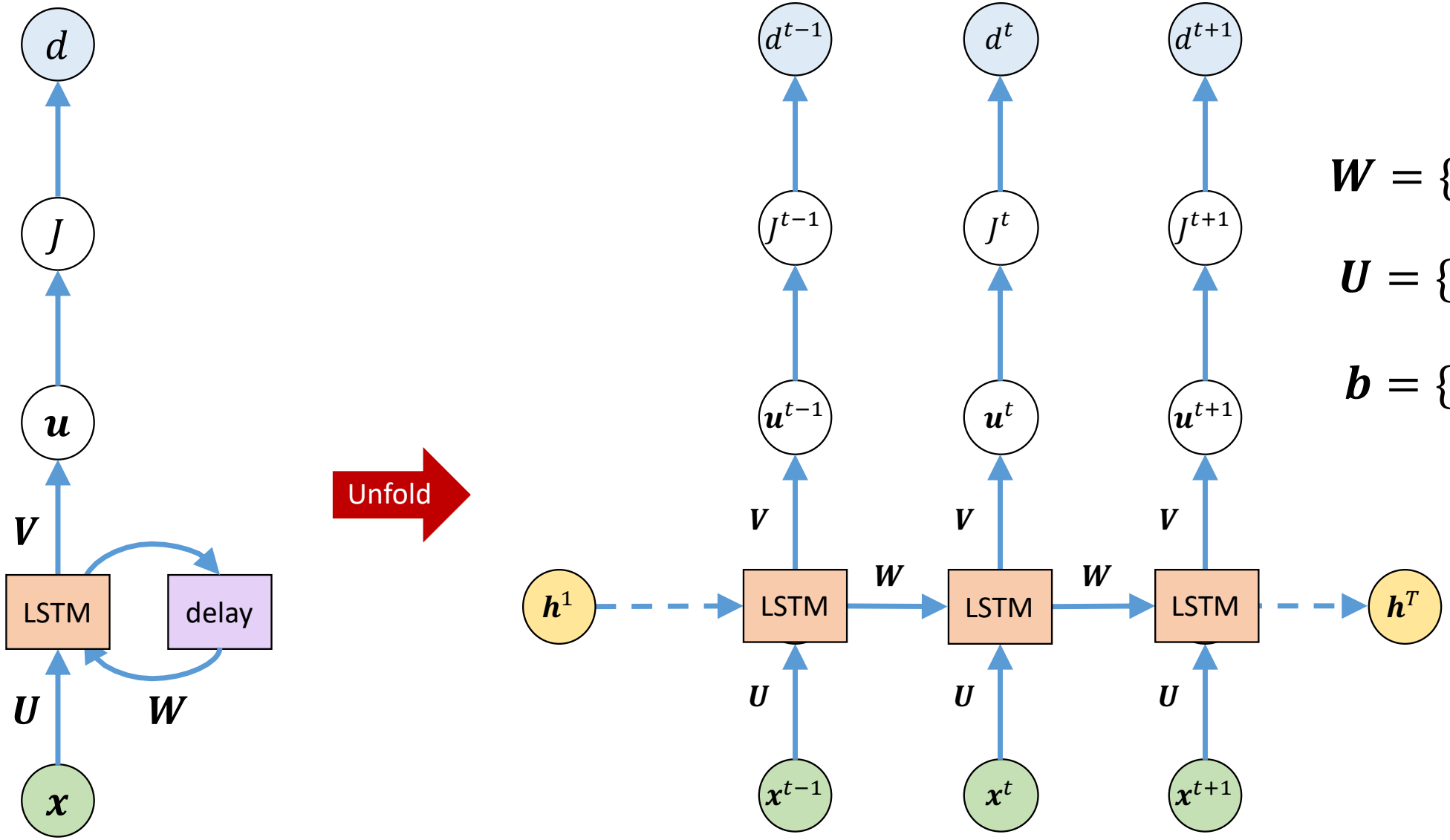
Clever idea!

The self-loop inside the cell is able to produces paths where the gradient can flow for long duration and to make the weight on this self-loop conditioned on the context rather than fixed.

By making the weight of this self-loop gated (controlled by another hidden unit), the time scale of integration can be changed dynamically based on the input sequence.

In this way, LSTM help preserve error terms that can be propagated through many layers and time steps.

Training LSTM networks



$$\boldsymbol{W} = \{\boldsymbol{W}_i, \boldsymbol{W}_f, \boldsymbol{W}_o, \boldsymbol{W}_c\}$$

$$\boldsymbol{U} = \{\boldsymbol{U}_i, \boldsymbol{U}_f, \boldsymbol{U}_o, \boldsymbol{U}_c\}$$

$$\boldsymbol{b} = \{\boldsymbol{b}_i, \boldsymbol{b}_f, \boldsymbol{b}_o, \boldsymbol{b}_c\}$$

Example 4

Generate 64 2-dimensional input sequences $\left(\boldsymbol{x}(t)\right)_{t=1}^{16}$ where $\left(x_1(t), x_2(t)\right) \in [0,1]^2$ by randomly generating numbers uniformly.

Generate 1-dimensional output sequences $\left(y(t)\right)_{t=1}^{16}$ where $y(t) \in \boldsymbol{R}$ by following the following recurrent relation:

$$y(t) = 5x_1(t-1)x_2(t-2) - 2x_1(t-7) + 3.5{x_2}^2(t-5) + 0.1\epsilon$$

where $\epsilon \sim N(0,1)$.

Train a LSTM layer to learn the mapping between input and output sequences.

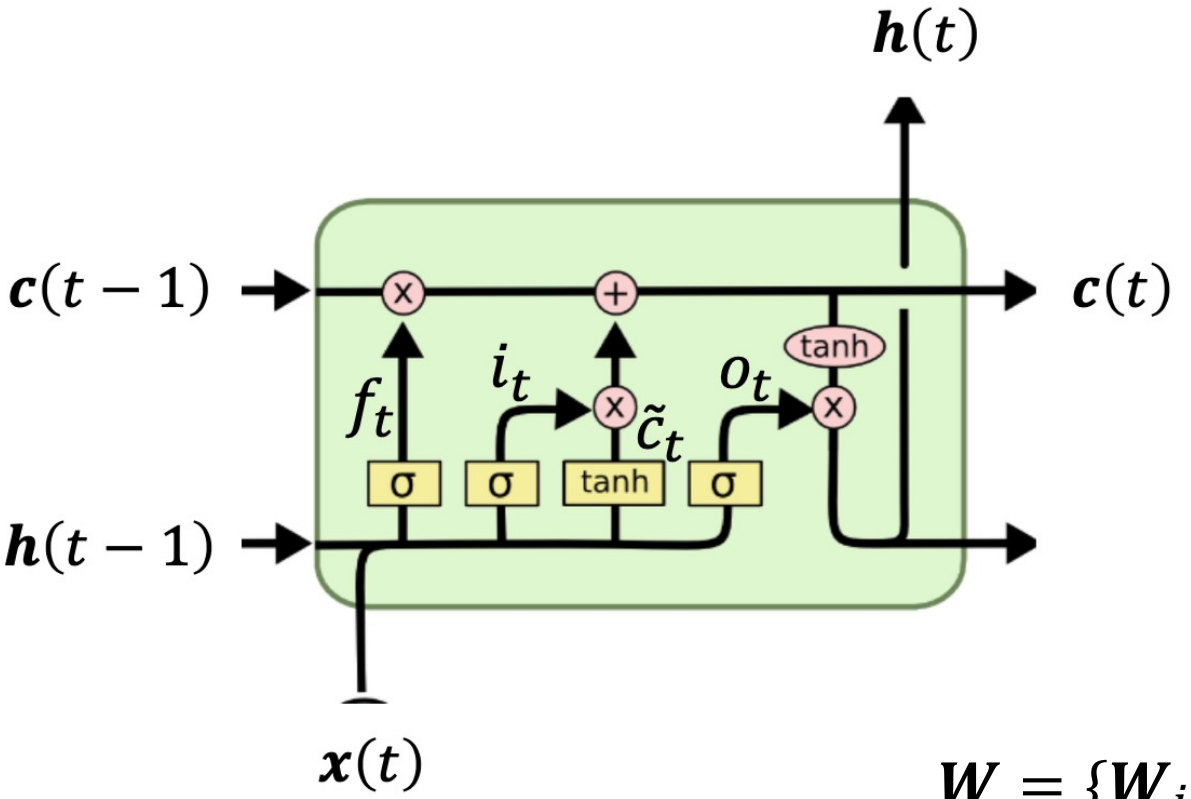
Use a learning factor $\alpha = 0.001$.

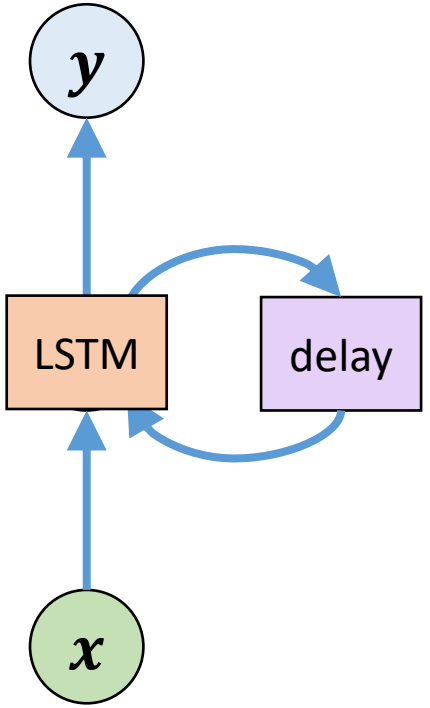
eg8.4a.ipynb

Repeat the above using RNN and GRU and compare the performances.

eg8.4b.ipynb

Example 4





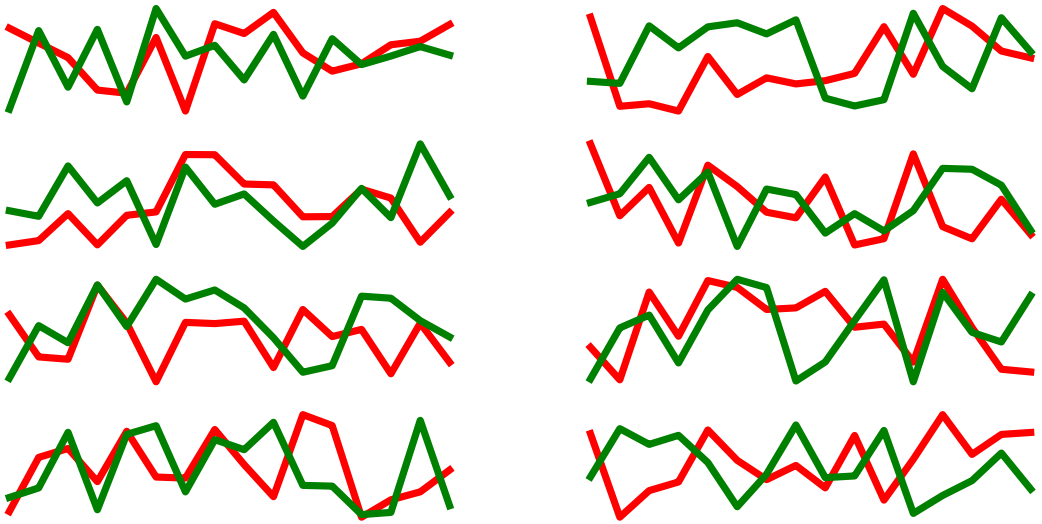
$$\boldsymbol{W} = \{\boldsymbol{W}_i, \boldsymbol{W}_f, \boldsymbol{W}_o, \boldsymbol{W}_c\}$$

$$\boldsymbol{U} = \{\boldsymbol{U}_i, \boldsymbol{U}_f, \boldsymbol{U}_o, \boldsymbol{U}_c\}$$

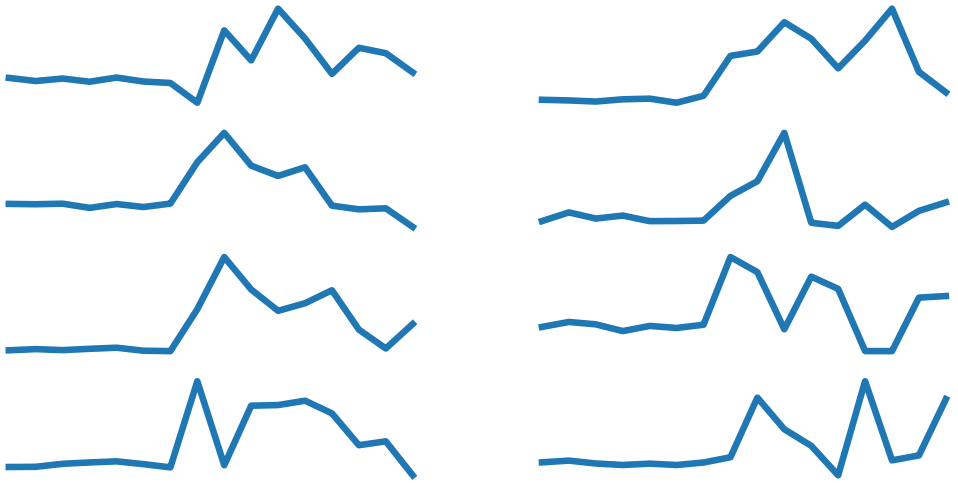
$$\boldsymbol{b} = \{\boldsymbol{b}_i, \boldsymbol{b}_f, \boldsymbol{b}_o, \boldsymbol{b}_c\}$$

Example 4

$x_1(t)$ and $x_2(t)$

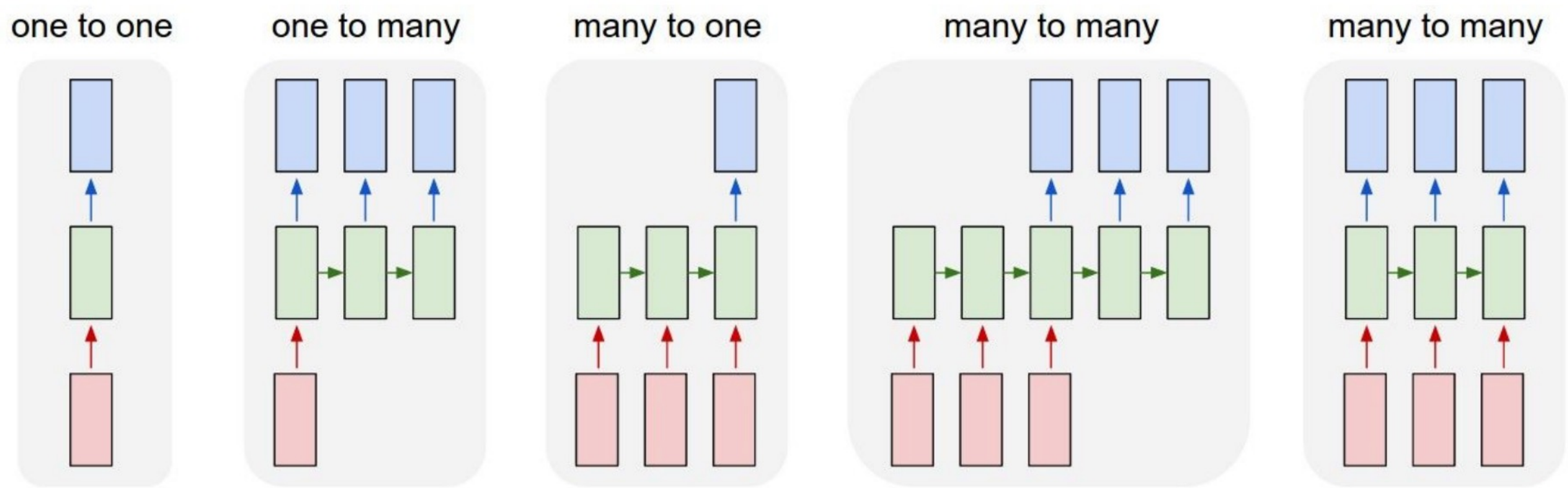


$$y(t) = 5x_1(t-1)x_2(t-1) - 2x_1(t-7) + 3.5{x_2}^2(t-5)$$



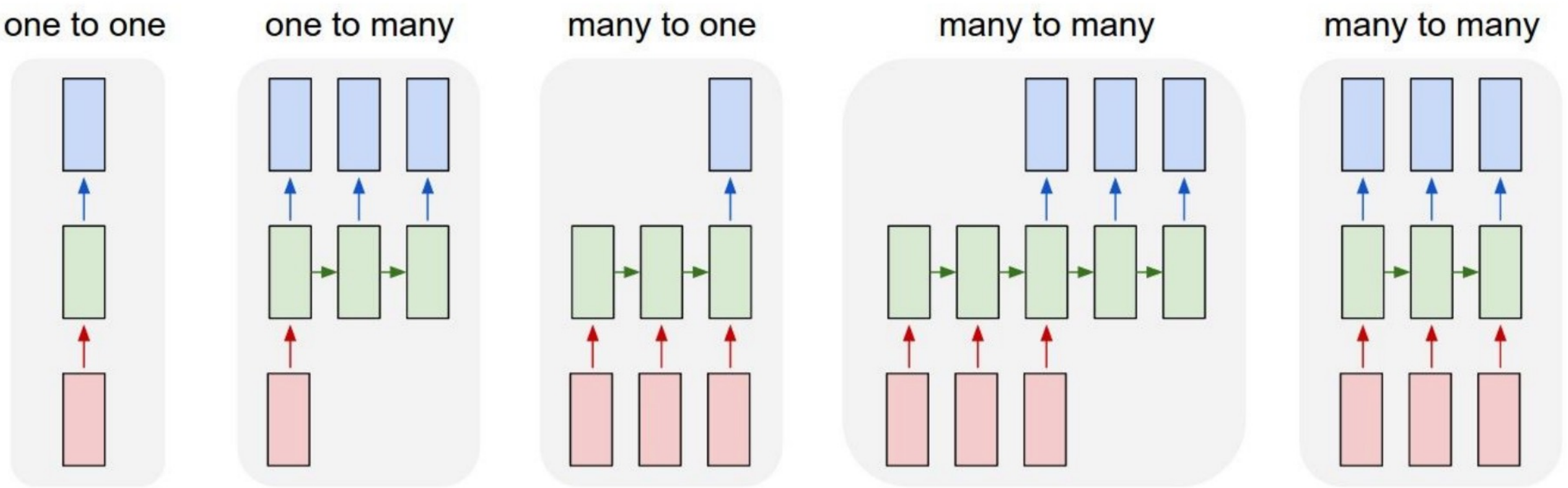
Example Applications

Input-output scenarios



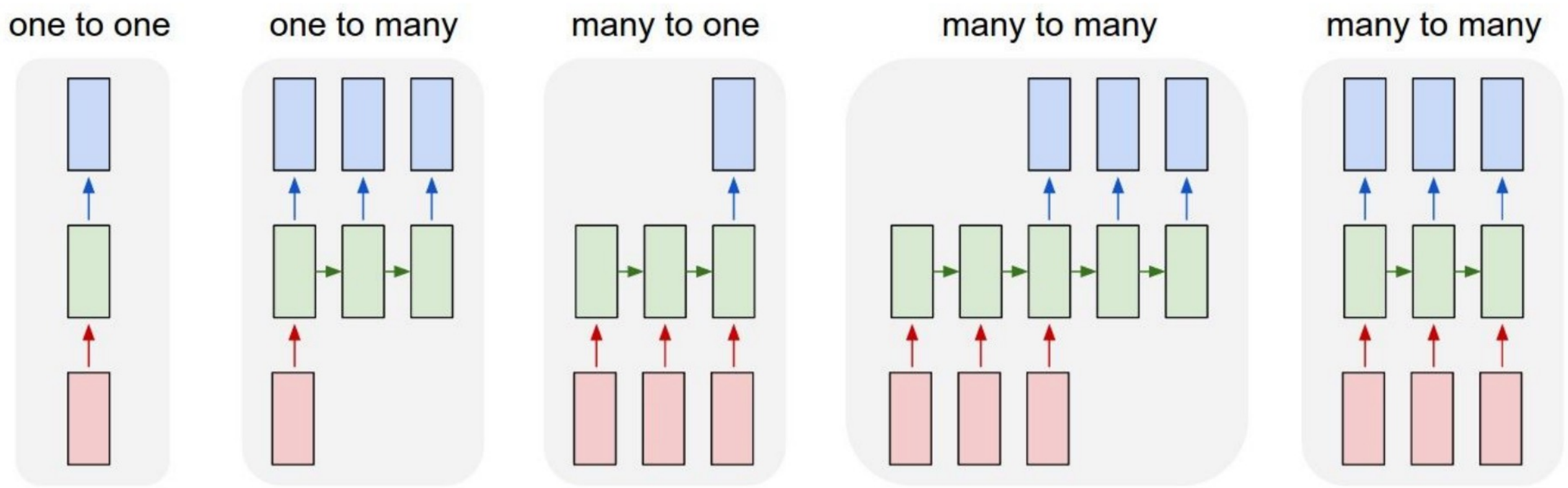
e.g. **Image Captioning**
image -> sequence of words

Input-output scenarios



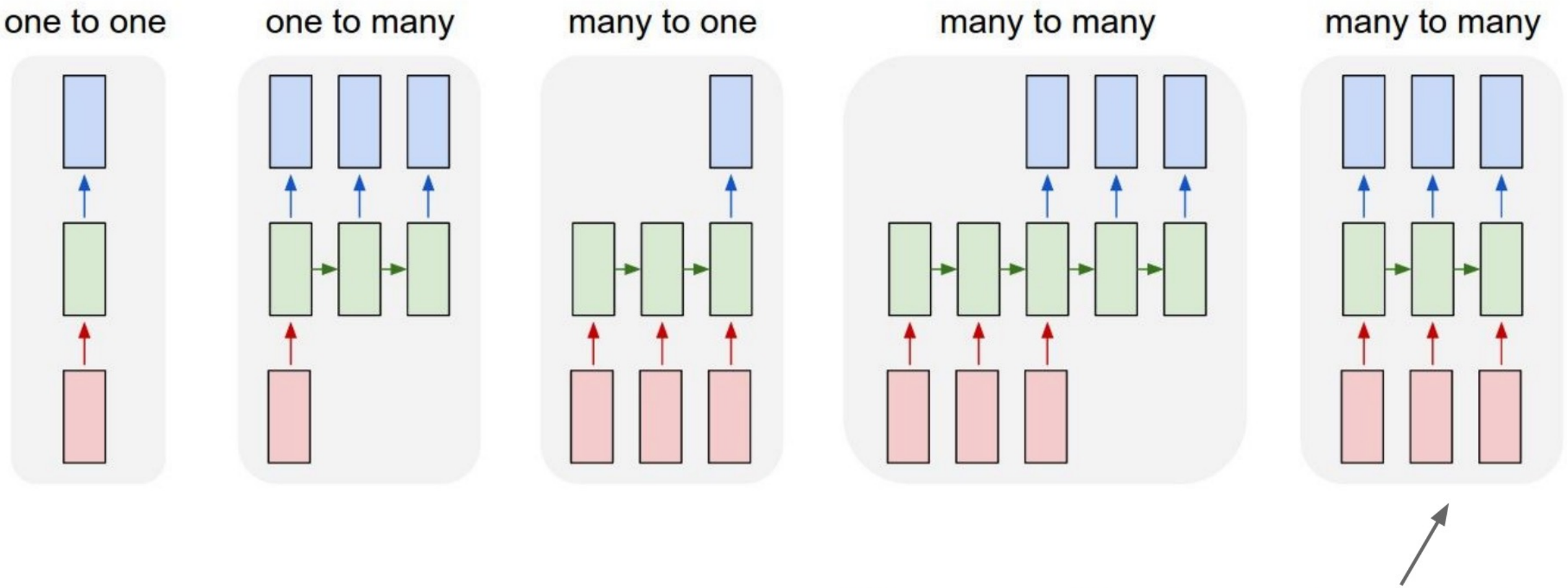
e.g. **Sentiment Classification**
sequence of words -> sentiment

Input-output scenarios



e.g. **Machine Translation**
seq of words -> seq of words

Input-output scenarios



Text classification: Dbpedia dataset

This dataset contains first paragraph of Wikipedia page of 56000 entities and label them as one of 15 categories like people, company, schools, etc.

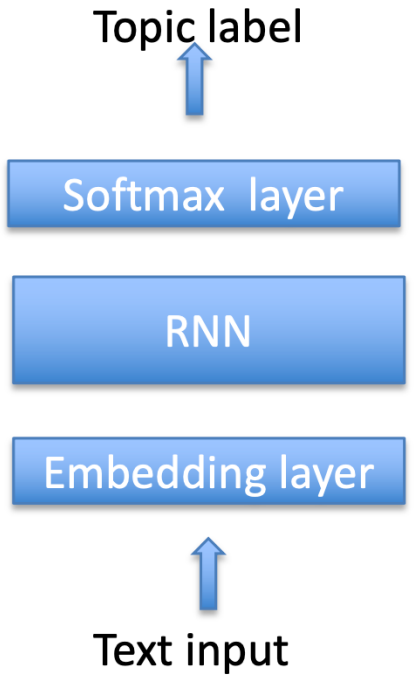
1	D. C. Thomson & Co.	D. C. Thomson & Company Limited is a Scottish publishing company based in Dundee best known for
1	Pyro Spectaculars	Pyro Spectaculars is headquartered in Rialto California USA and occupies a portion of a former World
1	Admiral Insurance	Admiral a trading name of EUI Limited is a car insurance specialist which launched in January 1993. Its
1	Pax Softnica	Pax Softnica (パックスソフトニカ) is a Japanese video game developer founded in 1983 under the nam
1	The ACME Laboratories Ltd	The ACME Laboratories Ltd is a major pharmaceutical company based in Bangladesh. It is part of the
2	Dubai Gem Private School & Nursery	Dubai Gem Private School (DGPS) is a British school located in the Oud Metha area of Dubai United Ara
2	Michael Wallace Elementary School	Michael Wallace Elementary School is a Canadian public school in Dartmouth Nova Scotia. It is operate
2	Alma Heights Christian Schools	Alma Heights Christian Schools (AHC) formerly Alma Heights Christian Academy is a private elementar

An input is a text
Requires to find all words in the text and remap them into word IDs, a number per each unit word (word-level inference)

my work is cool! → [23, 500, 5, 1402, 17]

We need to make sure that each sentence is of same length (no_words). The maximum document length is fixed and longer sentences will be truncated and shorter once are padded with zeros.

Text classification



$$\begin{bmatrix} w_{11} & w_{12} & w_{13} & w_{14} \\ w_{21} & w_{22} & w_{23} & w_{14} \\ w_{31} & w_{32} & w_{33} & w_{14} \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} w_{11} \\ w_{21} \\ w_{31} \end{bmatrix}$$

Matrix multiply with a one-hot vector just extracts a column from the weight matrix.

The embedding layer converts one-hot vector of word representations (of size of the vocabulary) to a vector of fixed length (embedding_size) vectors.

Embedding layer learns a weight matrix of [embedding size, vocab size]
And then maps word indexes of the sequences into
[batch_size, sequence_length, embedding_size] input to the RNN.

Sentiment classification

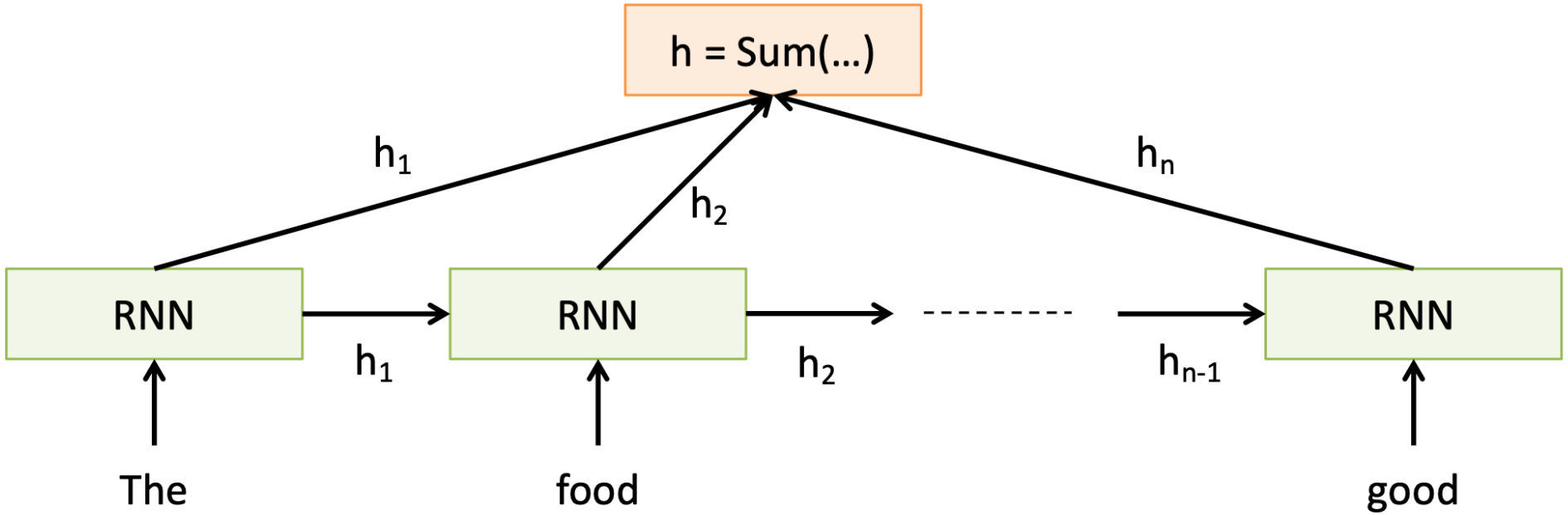
- “The food was really good”

“The chicken crossed the road because it was uncooked”

- Classify a
  restaurant review from Yelp! OR
  movie review from IMDB OR
  …
  as positive or negative

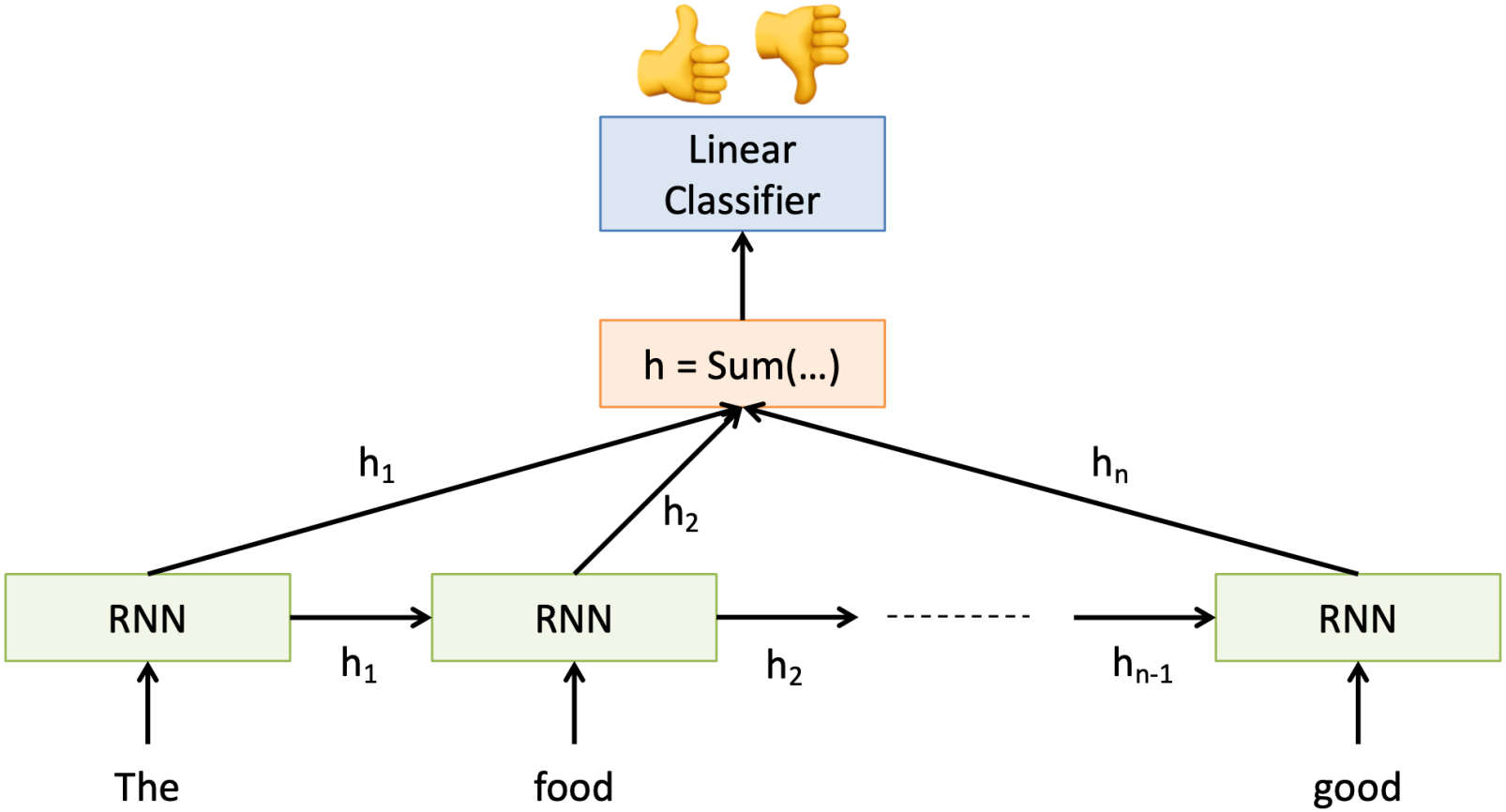
- Inputs: Multiple words, one or more sentences
- Outputs: Positive / Negative classification

Sentiment classification



http://deeplearning.net/tutorial/lstm.html

Sentiment classification



http://deeplearning.net/tutorial/lstm.html

Image captioning

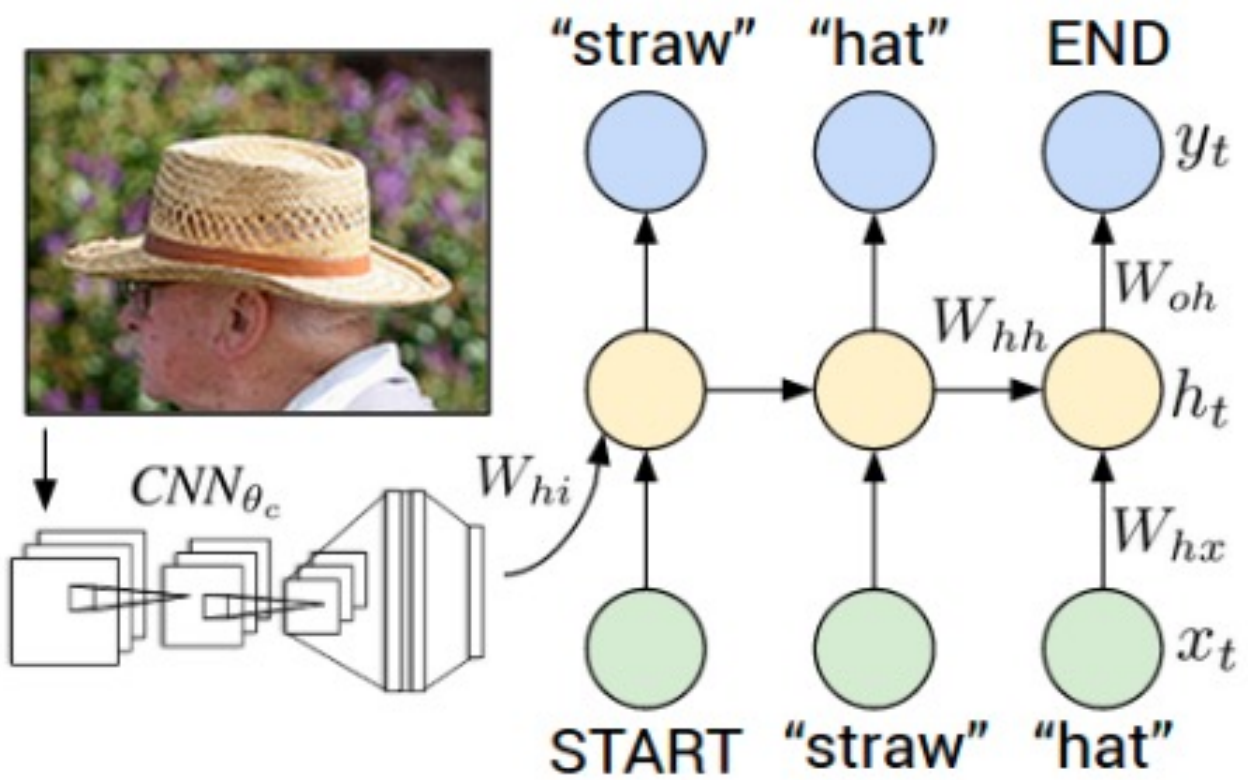
- Given an image, produce a sentence describing its contents
- Inputs: Image feature (from a CNN)
- Outputs: Multiple words (let's consider one sentence)

"man in black shirt is playing guitar."

"construction worker in orange safety vest is working on road."

"two young girls are playing with lego toy."

Image captioning



https://cs.stanford.edu/people/karpathy/sfmltalk.pdf

Image captioning

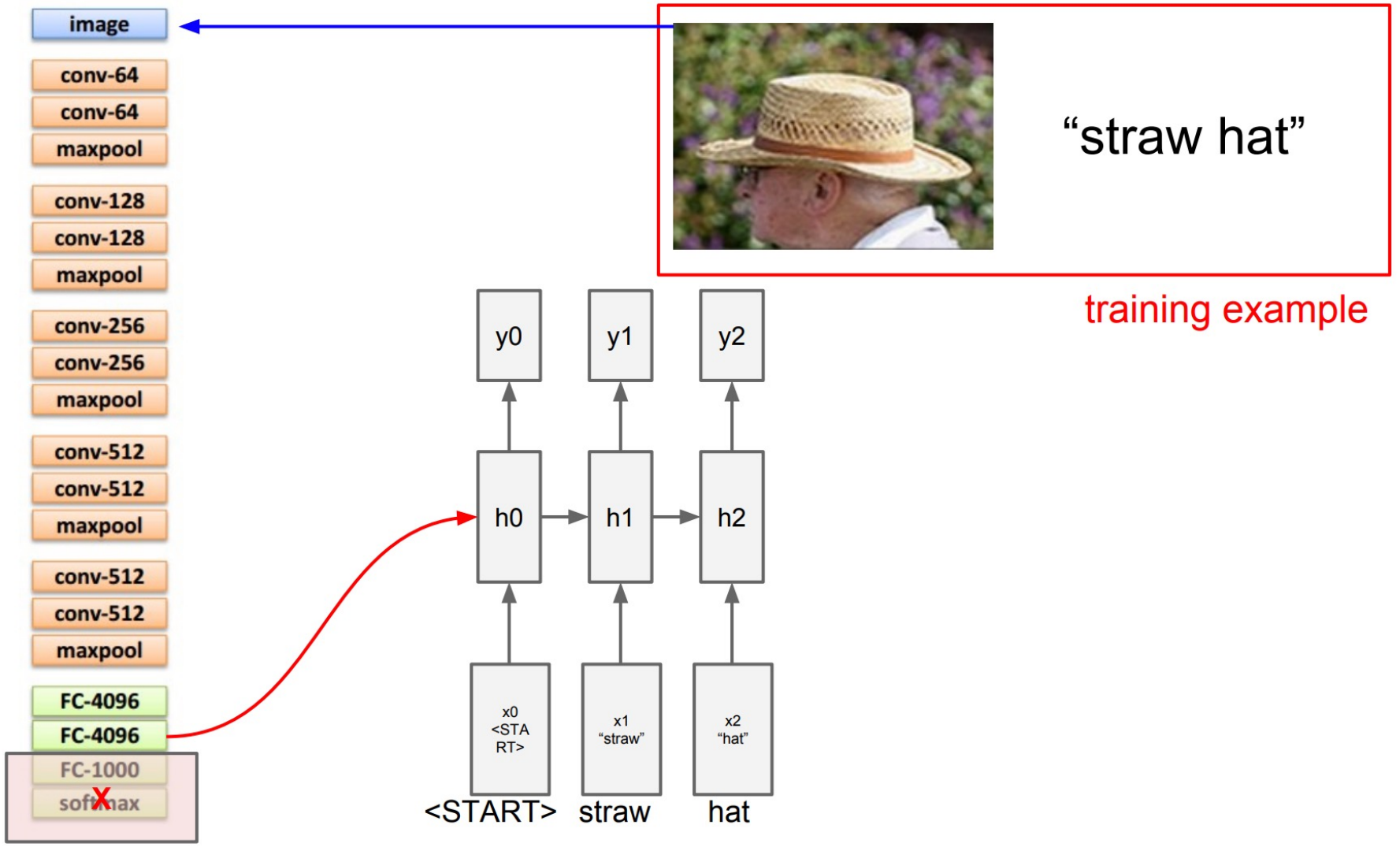


Image captioning

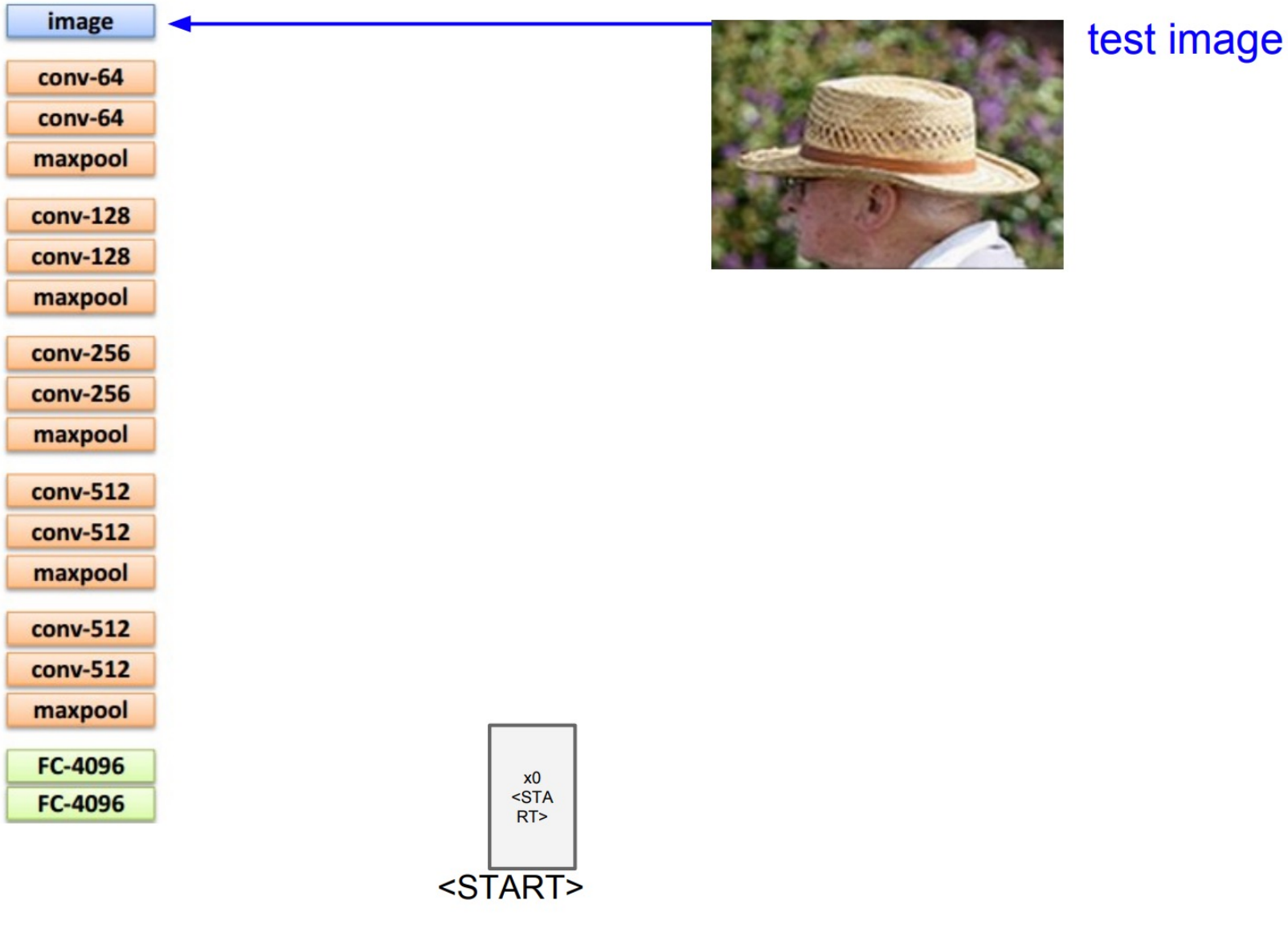


Image captioning

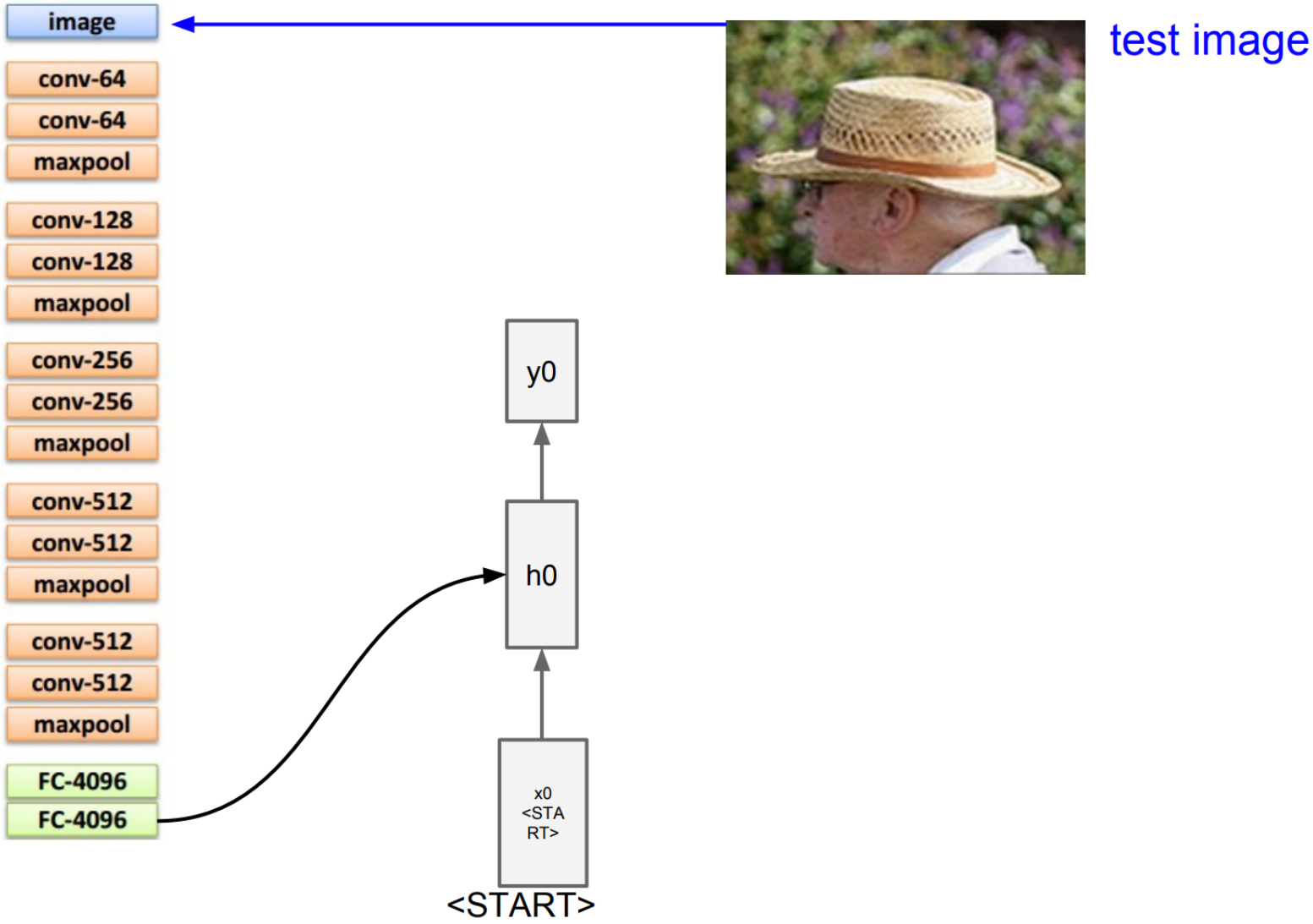


Image captioning

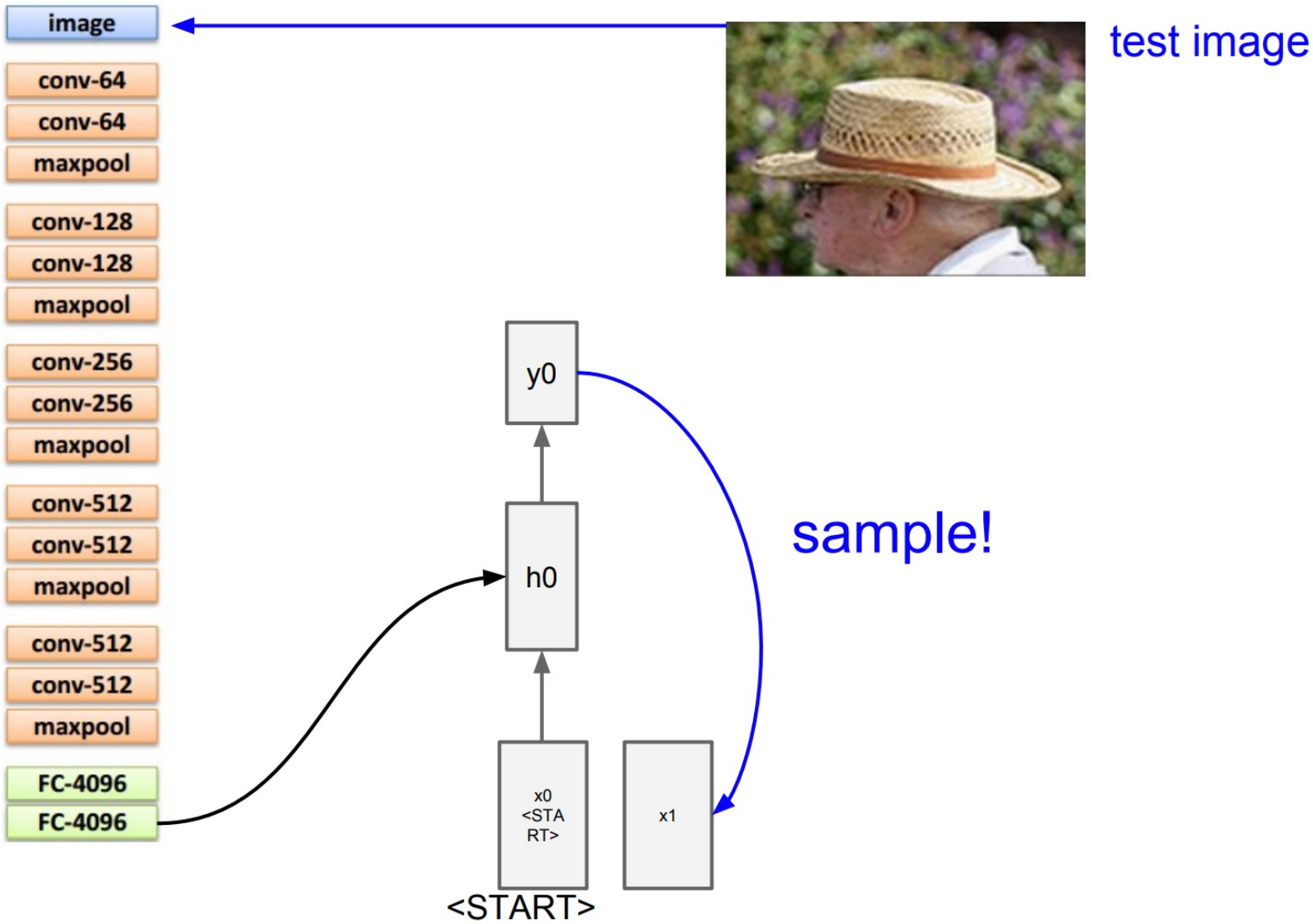


Image captioning

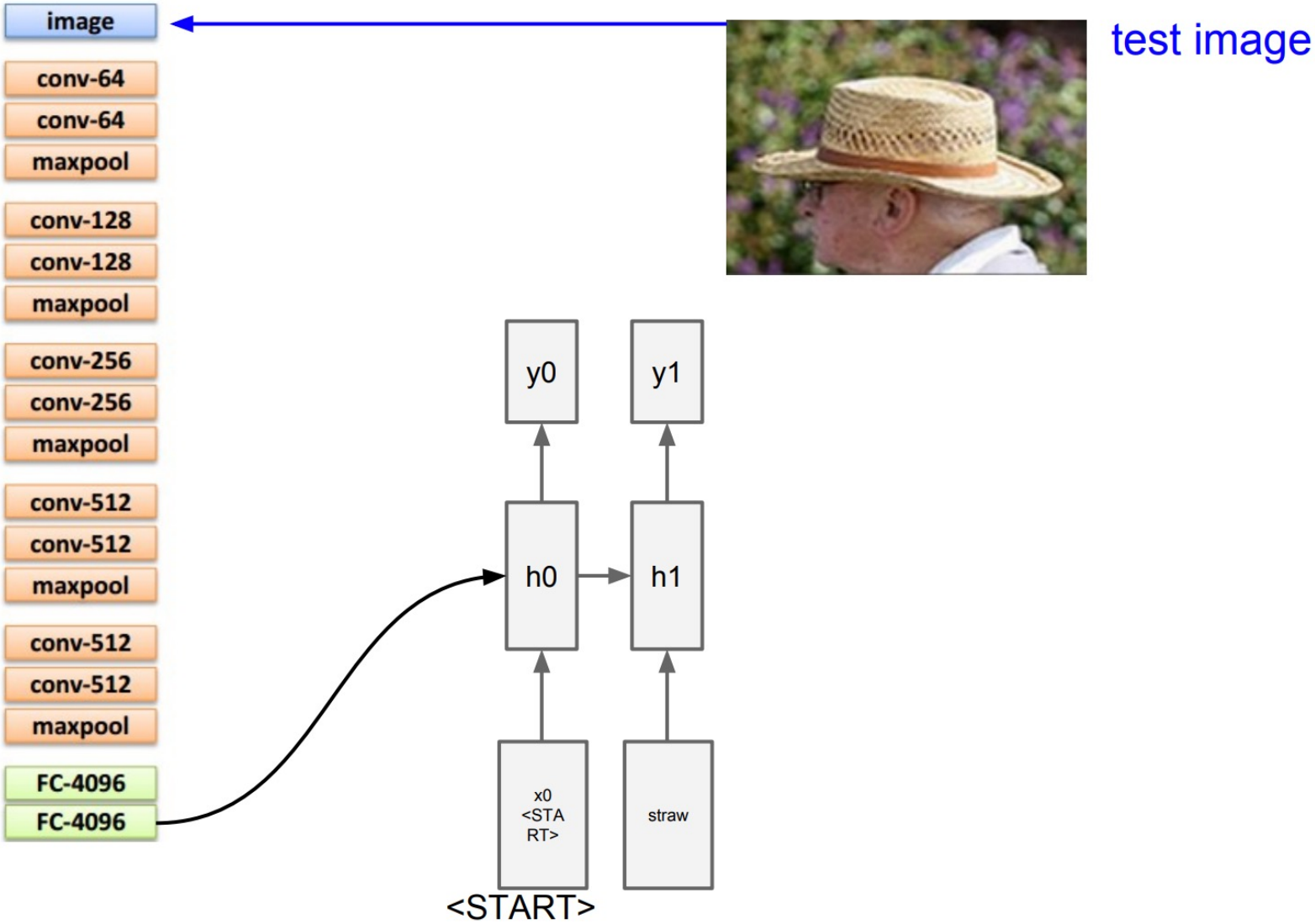


Image captioning

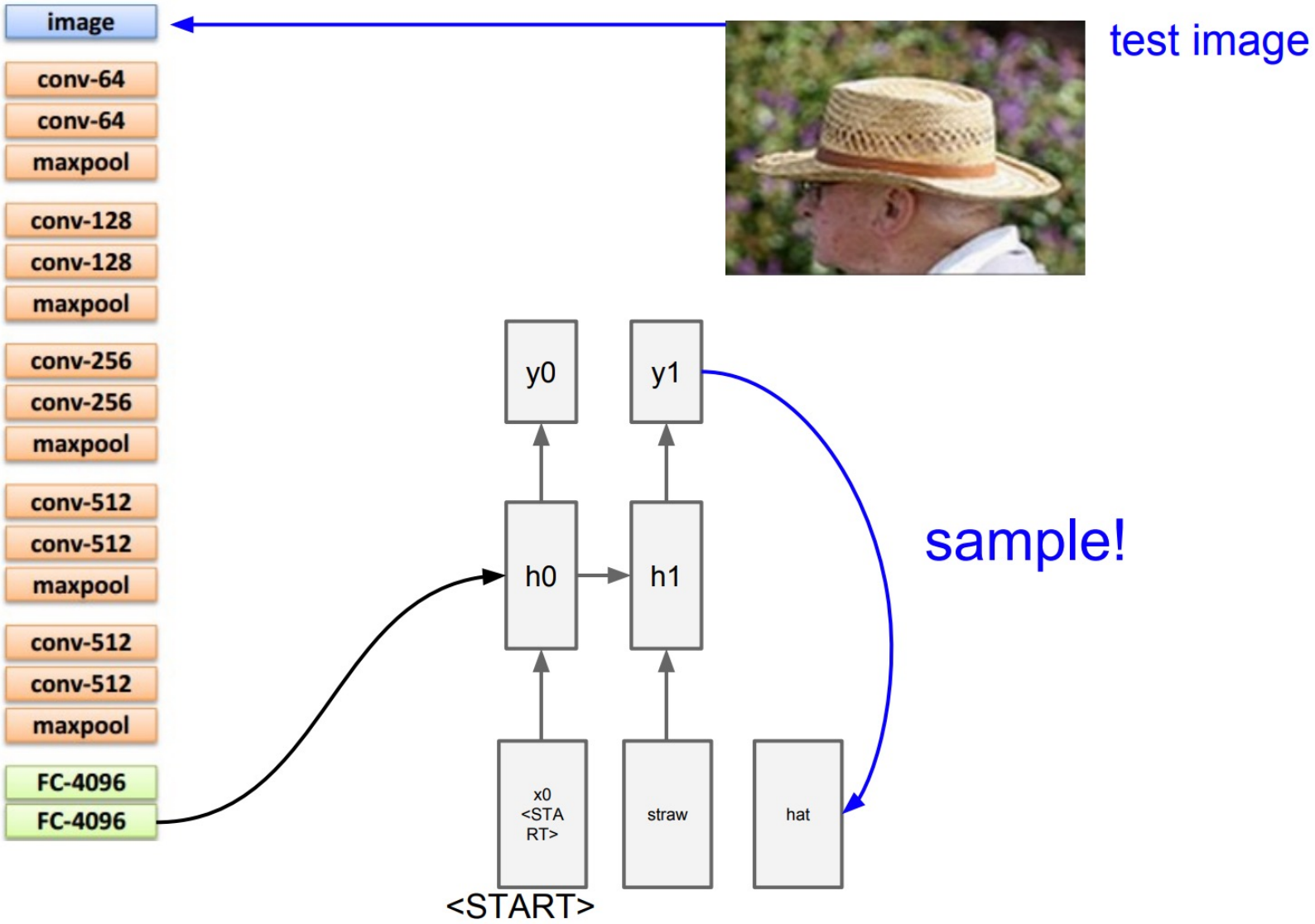


Image captioning

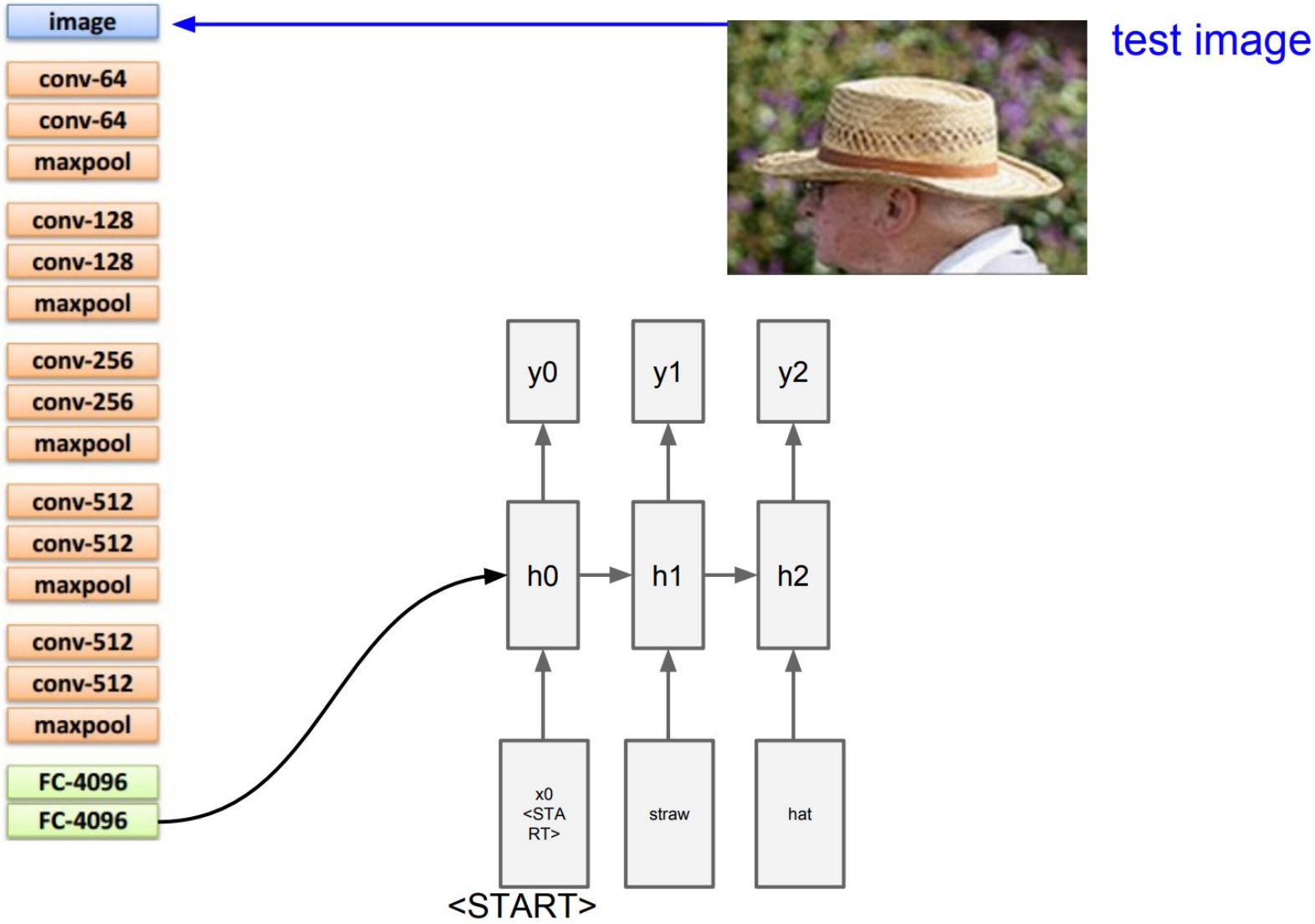


Image captioning

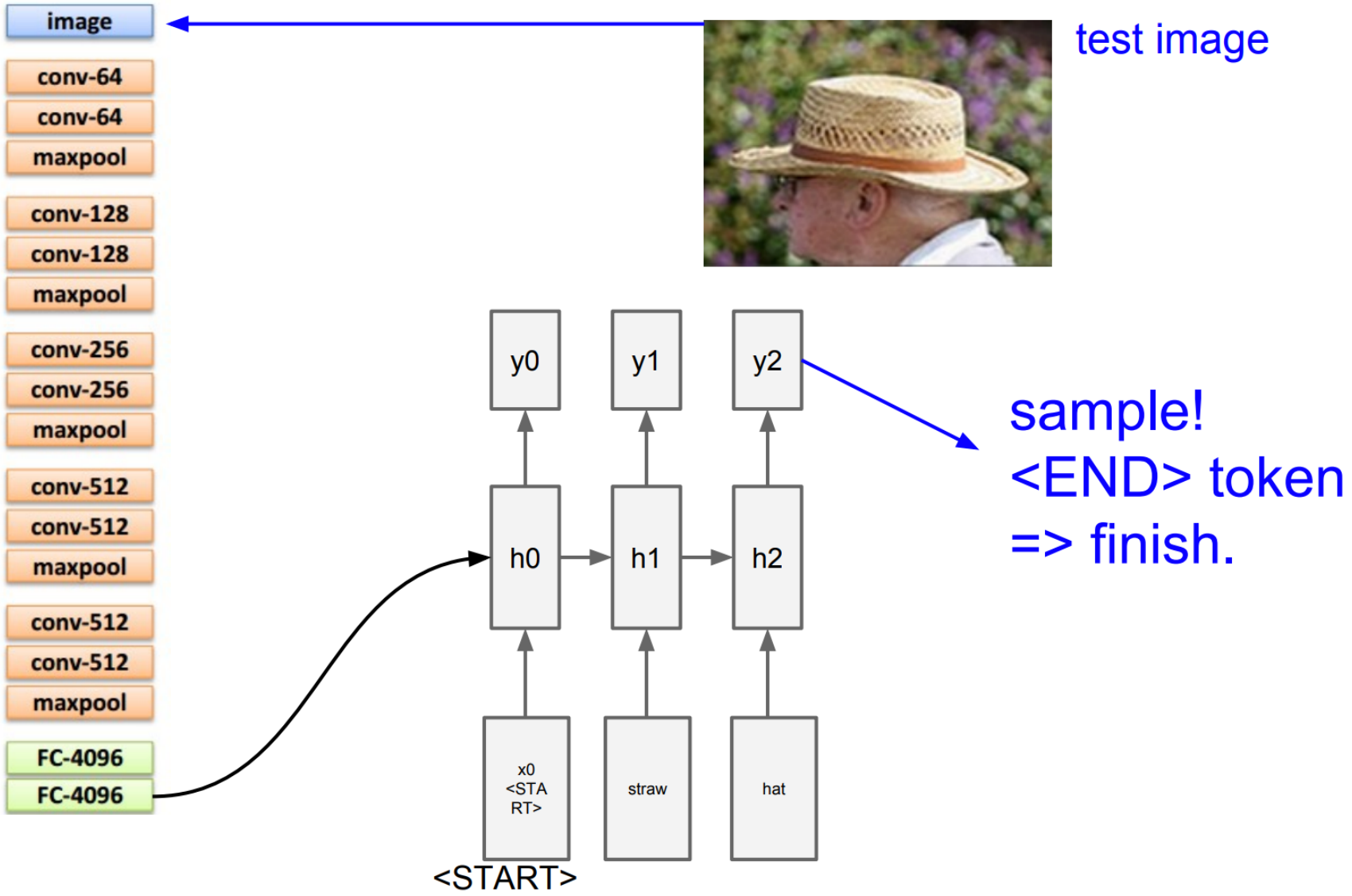
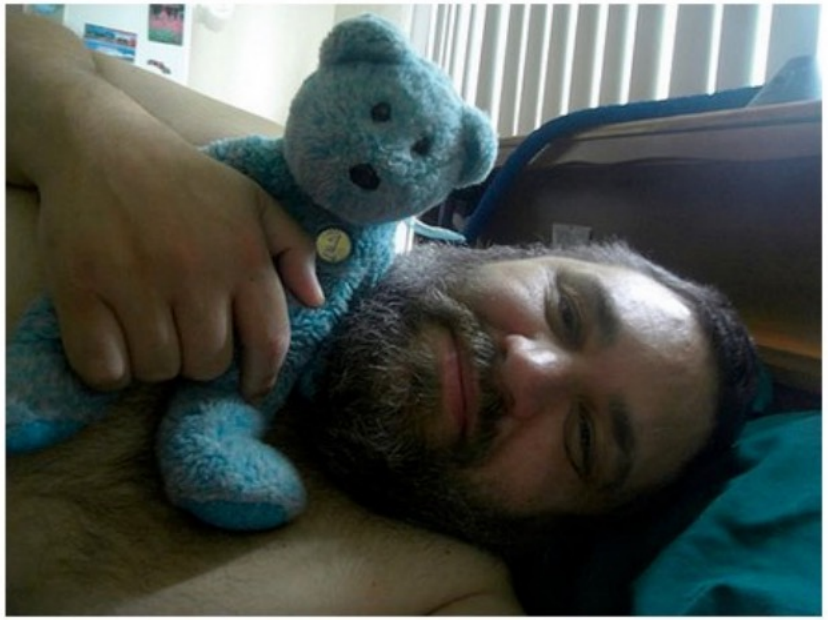


Image captioning

a young boy is holding a baseball bat
logprob: -7.65



a baby laying on a bed with a stuffed bear
logprob: -8.66

Image Credits

- https://cs.stanford.edu/people/karpathy/sfmltalk.pdf
- https://developer.ibm.com/articles/cc-cognitive-recurrent-neural-networks/