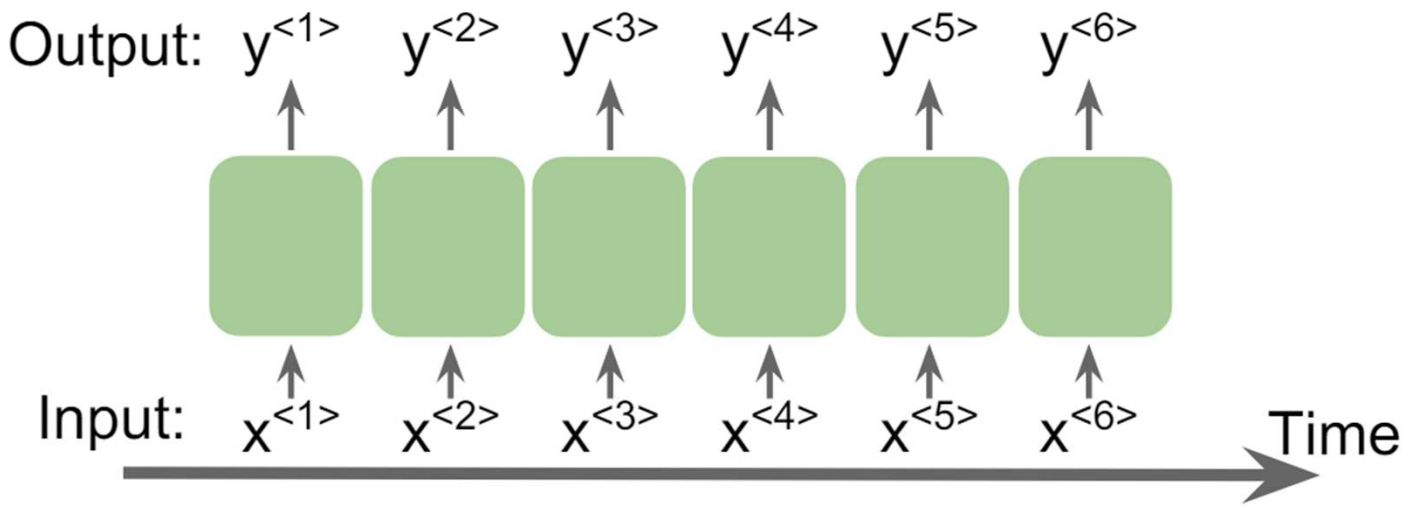
RNN

Fatemeh Mansoori

Sequence data: order matters

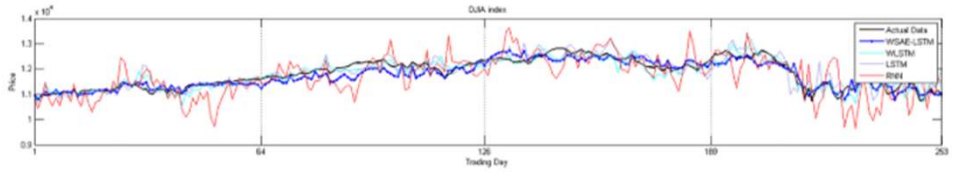
The movie my friend has **not** seen is good
The movie my friend has seen is **not** good

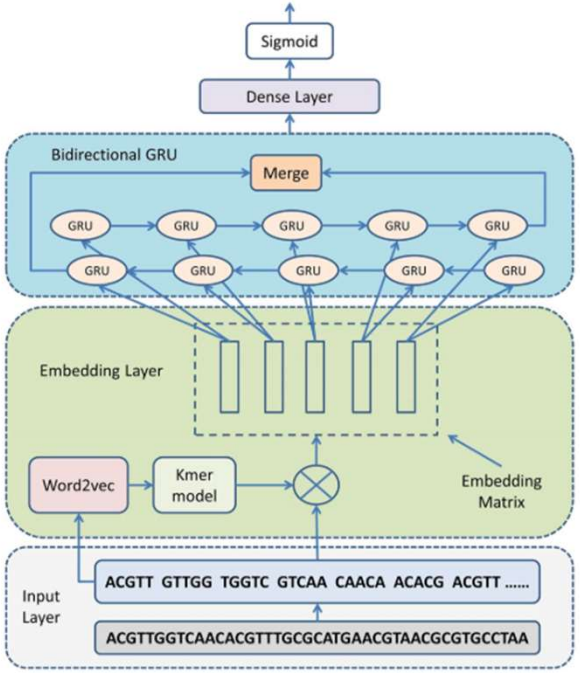


Applications: working with Sequential Data

- Text classification
- Speech recognition (acoustic modeling)
- language translation
- ...

Stock market predictions





Shen, Zhen, Wenzheng Bao, and De-Shuang Huang. "Recurrent Neural Network for Predicting Transcription Factor Binding Sites." *Scientific reports* 8, no. 1 (2018): 15270.

DNA or (amino acid/protein) sequence modeling

Applications: Speech Recognition

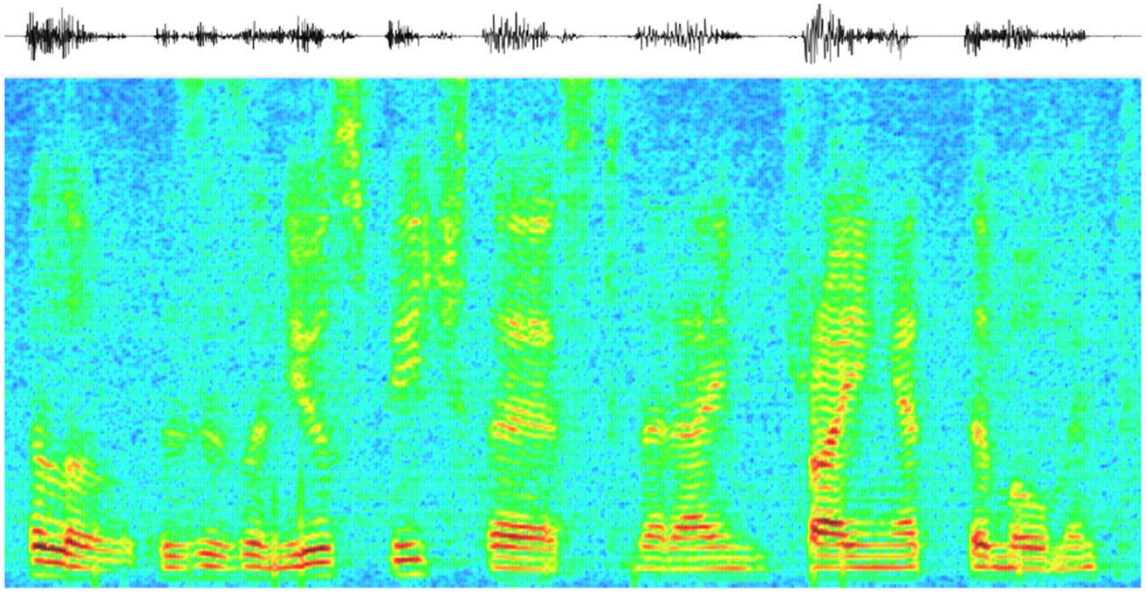


Figure: source

- Speech Recognition
  - Analyze a series of spectral vectors, determine what was said.
- Note: Inputs are sequences of vectors. Output is a classification result.

Application : Text analysis

Stephen Curry scored 34 points and was named the NBA Finals MVP as the Warriors claimed the franchise's seventh championship overall. And this one completed a journey like none other, after a run of five consecutive finals, then a plummet to the bottom of the NBA, and now a return to greatness just two seasons after having the league's worst record.

- Football or Basketball?
- Text Analysis
  - E.g. analyze document, identify topic
    - Input series of words, output classification output
  - E.g. read English, output Persian
    - Input series of words, output series of words

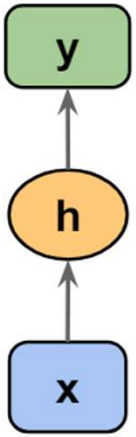
Application: Stock Market prediction

- Stock Market Prediction
  - Should I invest, vs. should I not invest in X?
  - Decision must be taken considering how things have fared over time.
- Note: Inputs are sequences of vectors. Output may be scalar or vector.

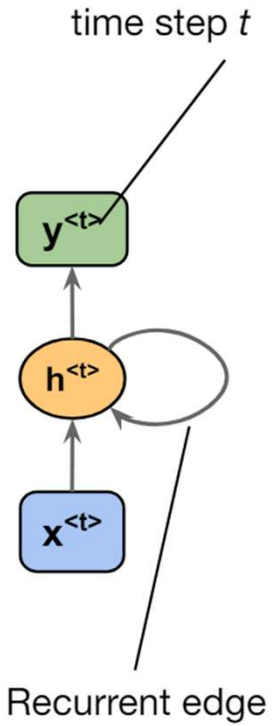
Recurrent Neural Network

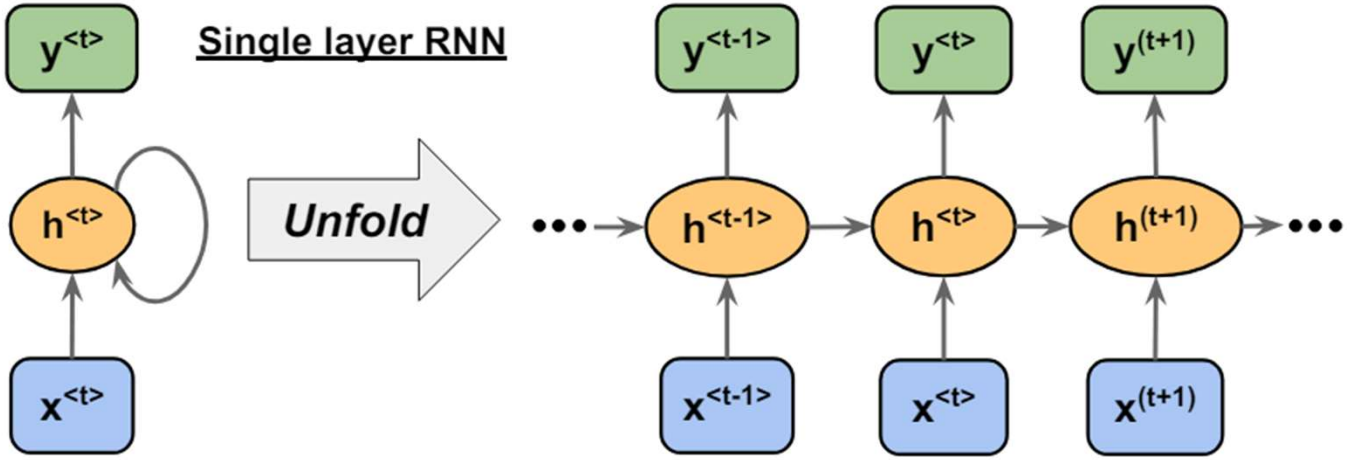
- A variant of the conventional feed-forward artificial neural networks to deal with sequential data
- Hold the knowledge about the past (Have memory!)

Networks we used previously: also called feedforward neural networks



Recurrent Neural Network (RNN)





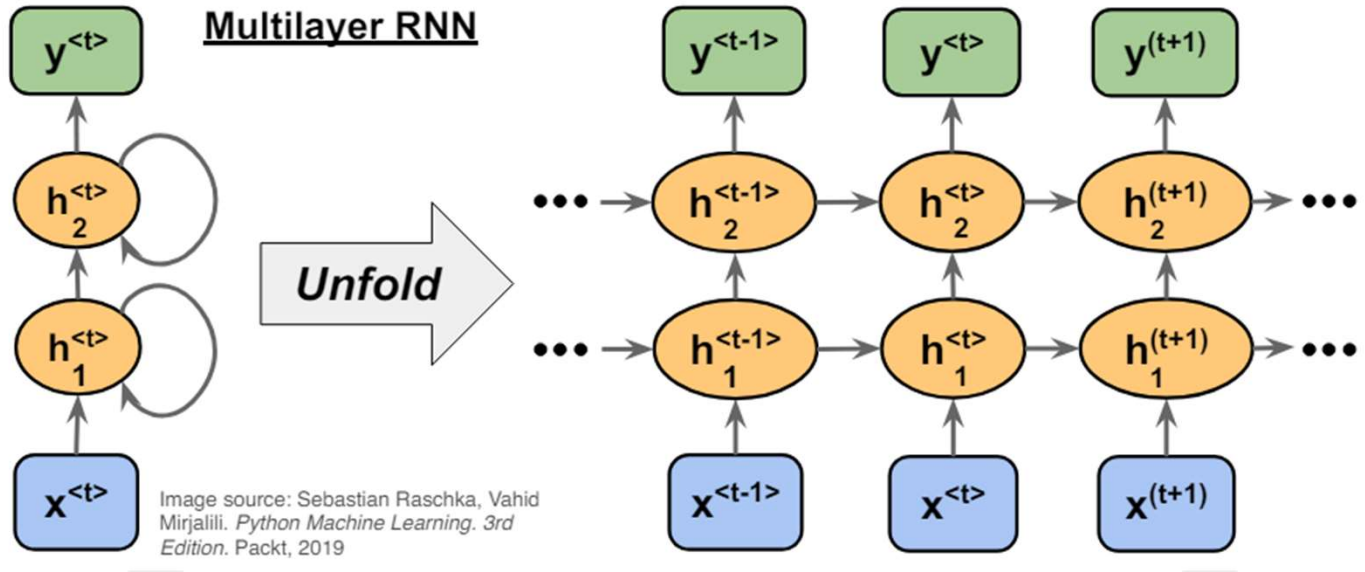
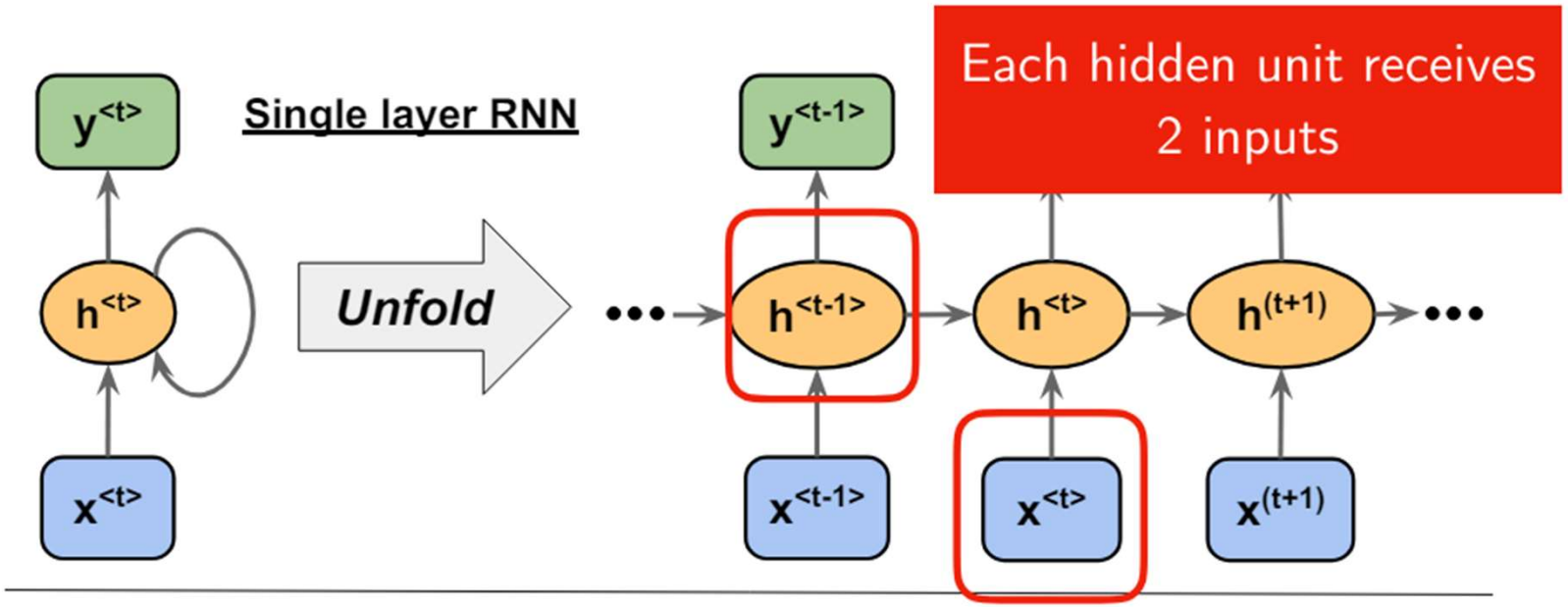
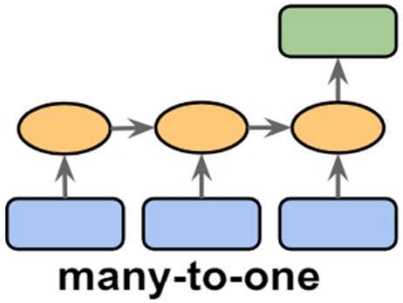


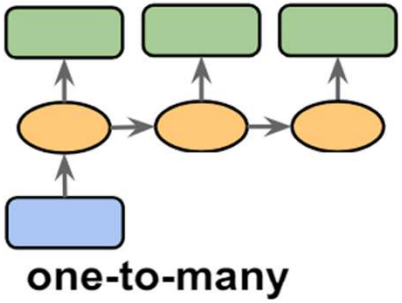
Image source: Sebastian Raschka, Vahid Mirjalili. *Python Machine Learning. 3rd Edition*. Packt, 2019


Single layer RNN
Unfold
Each hidden unit receives 2 inputs
y<t>
h<t>
x<t>
y<t-1>
h<t-1>
x<t-1>
h<t>
x<t>
h(t+1)
x(t+1)

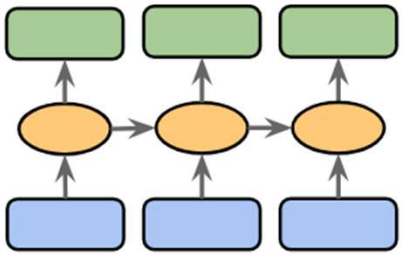
Different type of sequence modeling tasks



many-to-one



one-to-many



many-to-many

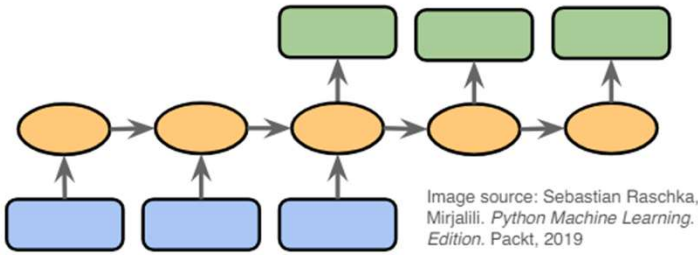
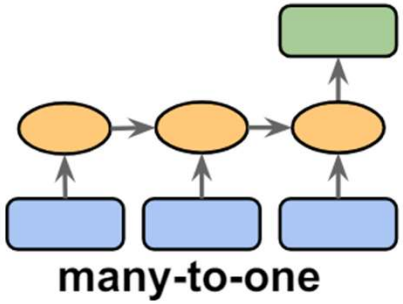


Image source: Sebastian Raschka, Vahid Mirjalili. *Python Machine Learning. 3rd Edition.* Packt, 2019

many-to-many

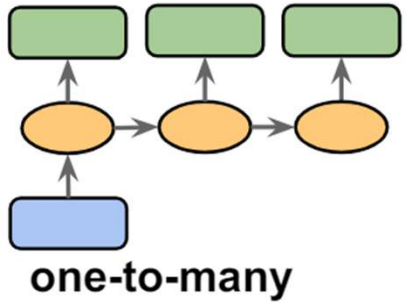
Different type of sequence modeling tasks



- Many-to-one: The input data is a sequence, but the output is a fixedsize vector, not a sequence.

- Ex.: sentiment analysis, the input is some text, and the output is a class label.

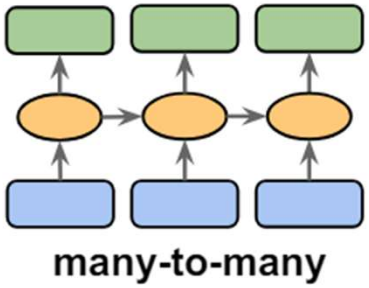
Different type of sequence modeling tasks

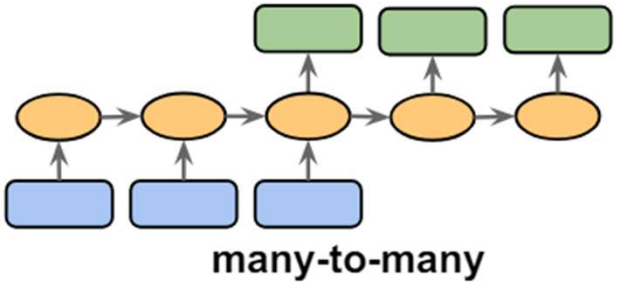


- One-to-many: Input data is in a standard format (not a sequence), the output is a sequence.
- Ex.: Image captioning, where the input is an image, the output is a text description of that image

Different type of sequence modeling tasks

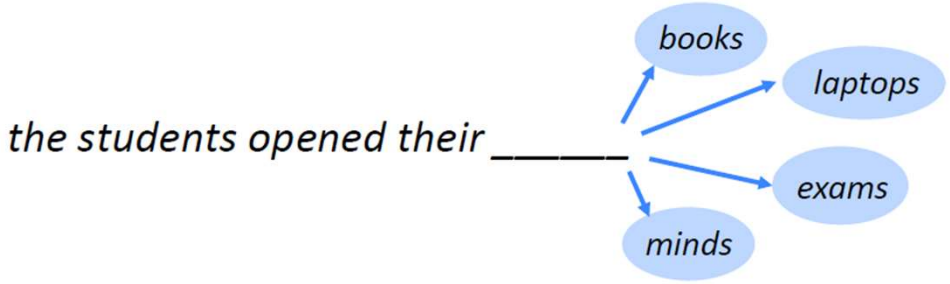
- Many-to-many: Both inputs and outputs are sequences. Can be direct or delayed.
- Ex.: Video-captioning, i.e., describing a sequence of images via text (direct).
- Translating one language into another (delayed)





Language Modeling

Language Modeling is the task of predicting what word comes next.



More formally: given a sequence of words $\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots, \boldsymbol{x}^{(t)}$, compute the probability distribution of the next word $\boldsymbol{x}^{(t+1)}$:

$$P(\boldsymbol{x}^{(t+1)} | \ \boldsymbol{x}^{(t)}, \ldots, \boldsymbol{x}^{(1)})$$

where $\boldsymbol{x}^{(t+1)}$ can be any word in the vocabulary $V = \{\boldsymbol{w}_1, \ldots, \boldsymbol{w}_{|V|}\}$

A system that does this is called a **Language Model**.

Language Modeling

You can also think of a Language Model as a system that assigns probability to a piece of text.

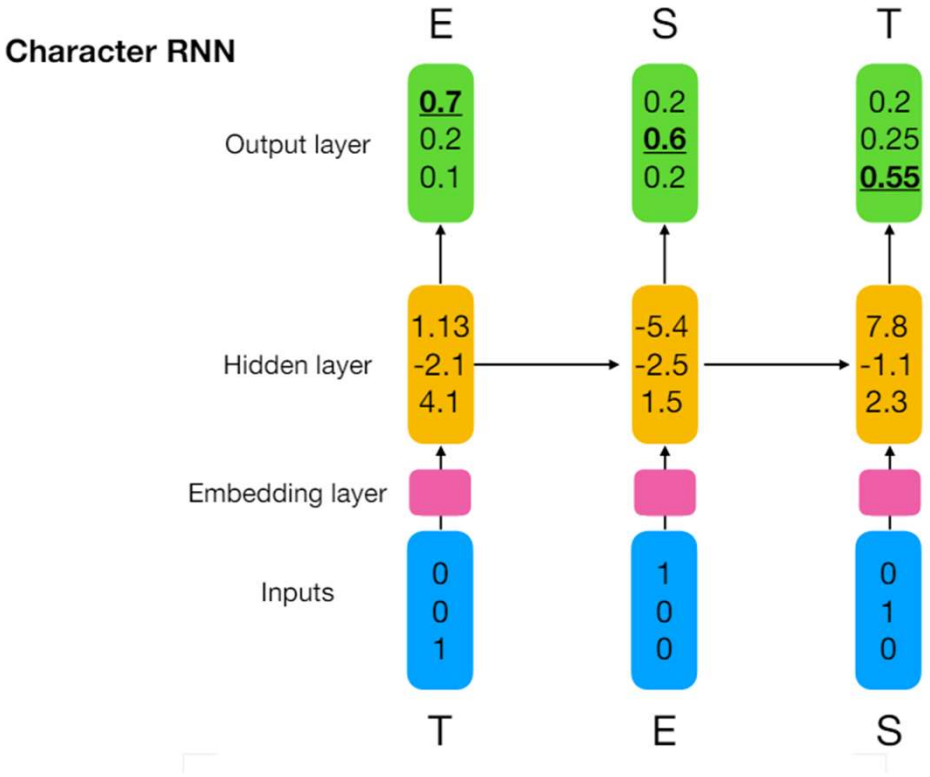
For example, if we have some text $\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(T)}$, then the probability of this text (according to the Language Model) is:

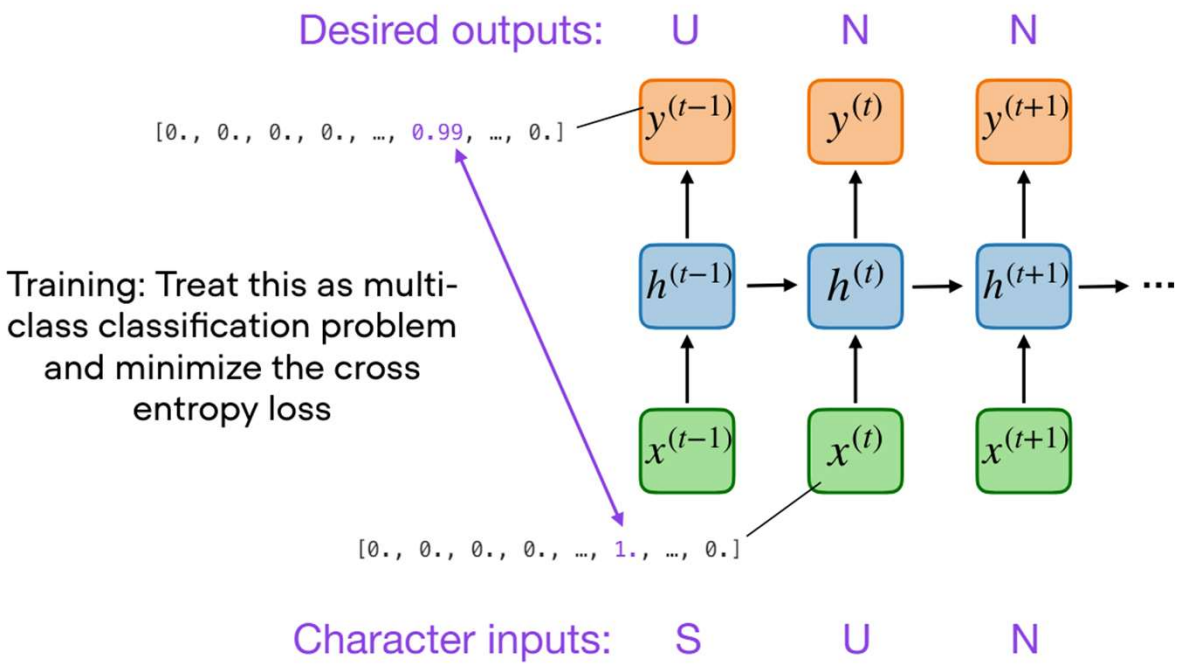
$$P(\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(T)}) = P(\boldsymbol{x}^{(1)}) \times P(\boldsymbol{x}^{(2)} | \, \boldsymbol{x}^{(1)}) \times \cdots \times P(\boldsymbol{x}^{(T)} | \, \boldsymbol{x}^{(T-1)}, \ldots, \boldsymbol{x}^{(1)})$$

$$= \prod_{t=1}^{T} P(\boldsymbol{x}^{(t)} | \, \boldsymbol{x}^{(t-1)}, \ldots, \boldsymbol{x}^{(1)})$$

This is what our LM provides

Language Model with RNN




Desired outputs:
U
N
N
[0., 0., 0., 0., …, 0.99, …, 0.]
$y^{(t-1)}$
$y^{(t)}$
$y^{(t+1)}$
Training: Treat this as multi-class classification problem and minimize the cross entropy loss
$h^{(t-1)}$
$h^{(t)}$
$h^{(t+1)}$
…
$x^{(t-1)}$
$x^{(t)}$
$x^{(t+1)}$
[0., 0., 0., 0., …, 1., …, 0.]
Character inputs:
S
U
N

RNN Step 1): Building the Vocabulary

"Raw" training dataset

$\mathbf{x}^{[1]} =$ "The sun is shining"

$\mathbf{x}^{[2]} =$ "The weather is sweet"

$\mathbf{x}^{[3]} =$ "The sun is shining,
the weather is sweet, and
one and one is two"

$\mathbf{y} = [0, 1, 0]$

class labels

```
vocabulary = {
    '<unk>': 0,
    'and': 1,
    'is': 2
    'one': 3,
    'shining': 4,
    'sun': 5,
    'sweet': 6,
    'the': 7,
    'two': 8,
    'weather': 9,
    '<pad>': 10
}
```

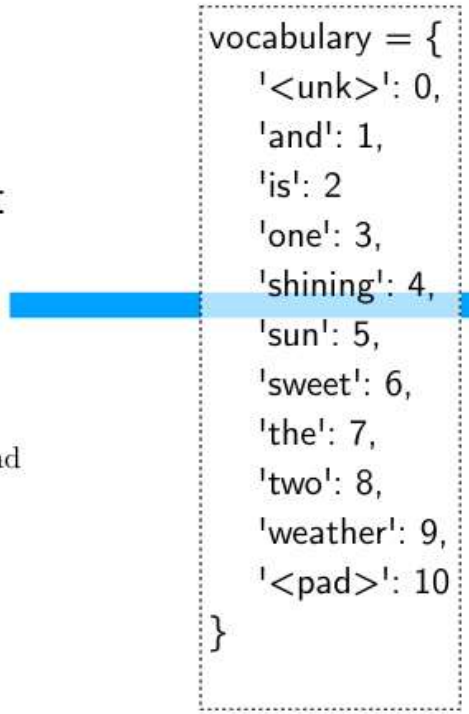
RNN Step 2): Training Example Texts to Indices

"Raw" training dataset

$\mathbf{x}^{[1]}$ = "The sun is shining"

$\mathbf{x}^{[2]}$ = "The weather is sweet"

$\mathbf{x}^{[3]}$ = "The sun is shining, the weather is sweet, and one and one is two"



$\mathbf{x}^{[1]}$ = "The sun is shining"

[7 5 2 4 ... 10 10 10]

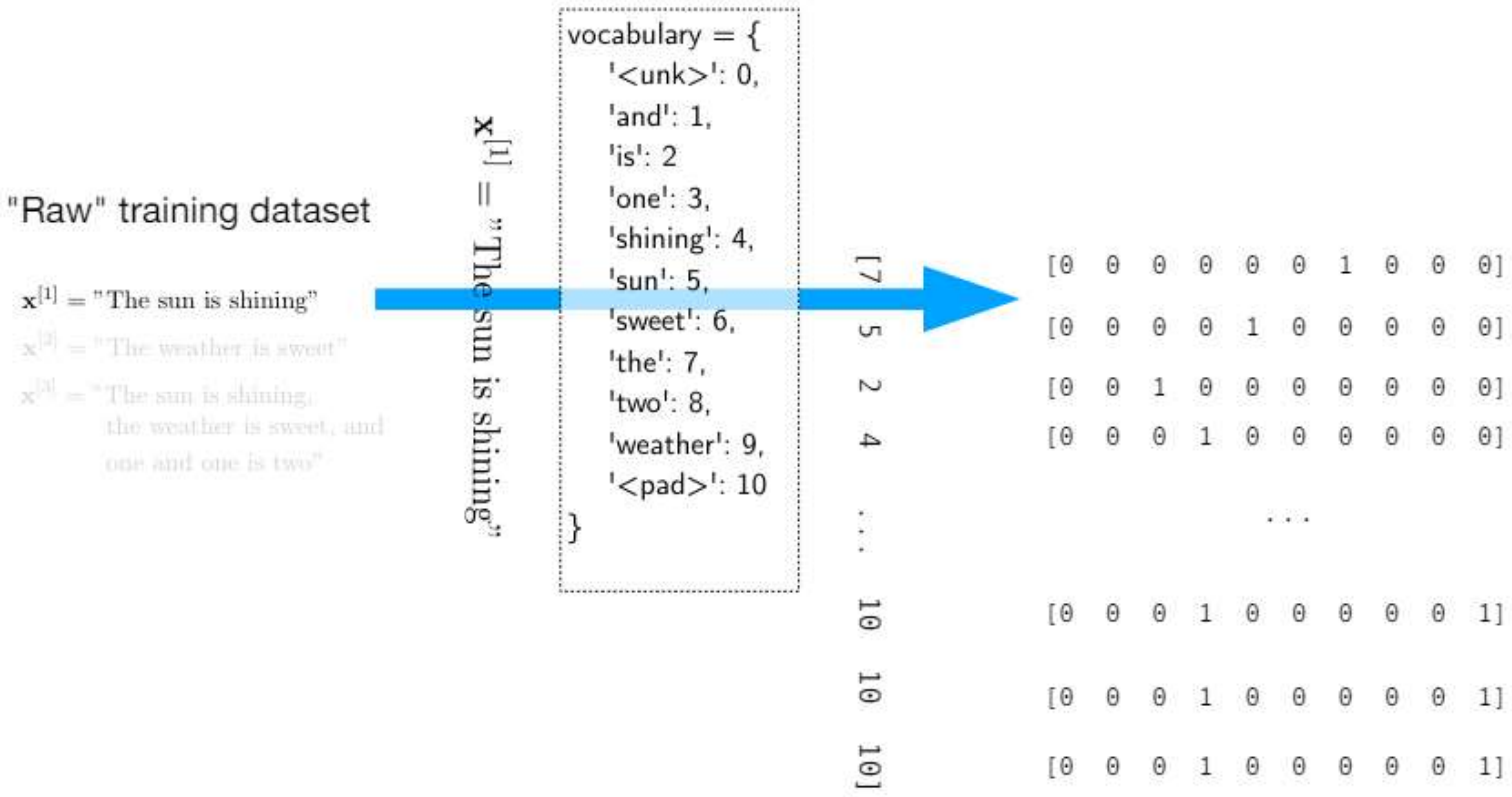
$\mathbf{x}^{[2]}$ = "The weather is sweet"

[7 9 2 6 ... 10 10 10]

$\mathbf{x}^{[3]}$ = "The sun is shining, the weather is sweet, and one and one is two"

[7 5 2 4 ... 3 2 8]

RNN Step 3): Indices to One-Hot Representation



RNN Step 4): One-Hot to Real via Embedding Matrix

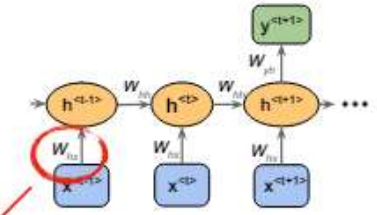
$\mathbf{x}^{[1]}$ = "The sun is shining"

```
vocabulary = {
    '<unk>': 0,
    'and': 1,
    'is': 2
    'one': 3,
    'shining': 4,
    'sun': 5,
    'sweet': 6,
    'the': 7,
    'two': 8,
    'weather': 9,
    '<pad>': 10
}
```

[7 5 2 4 ... 10 10 10]

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

Embedding is a linear layer



Embedding matrix

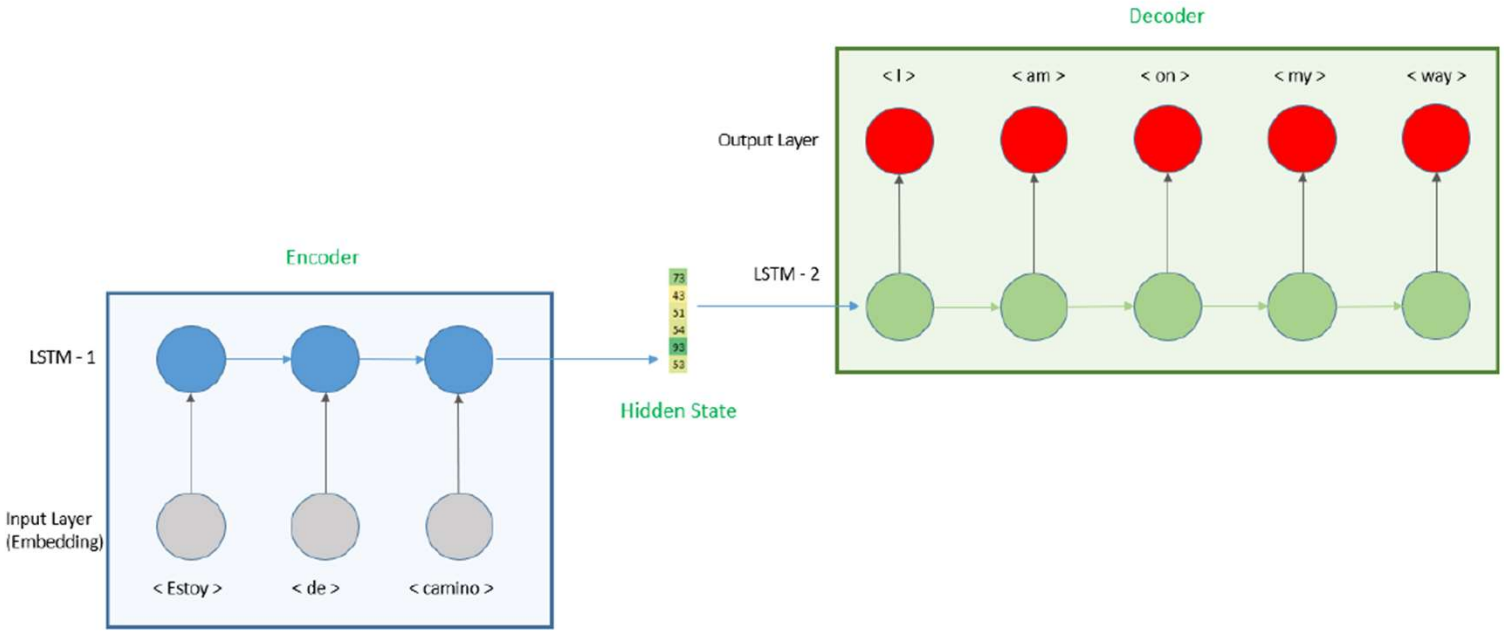
One-hot vector

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

$$\times \begin{bmatrix} 0.1 & 4.2 & 2.1 & 1.9 \\ 1.1 & 1.2 & 1.3 & 1.4 \\ 2.1 & 2.2 & 2.3 & 2.4 \\ 3.1 & 2.6 & 1.5 & 9.1 \\ 4.1 & 2.6 & 2.2 & 8.8 \\ 5.1 & 3.6 & 1.5 & 9.1 \\ 6.1 & 9.1 & 7.4 & 9.0 \\ 7.1 & 2.5 & 1.5 & 1.5 \\ 8.1 & 6.1 & 1.5 & 6.2 \\ 9.1 & 5.5 & 1.1 & 9.1 \\ 1.1 & 5.3 & 4.8 & 9.1 \end{bmatrix} = \begin{bmatrix} 7.1 & 2.5 & 1.5 & 1.5 \end{bmatrix}$$

Hidden layer output

Encoder decoder



Encoder Decoder

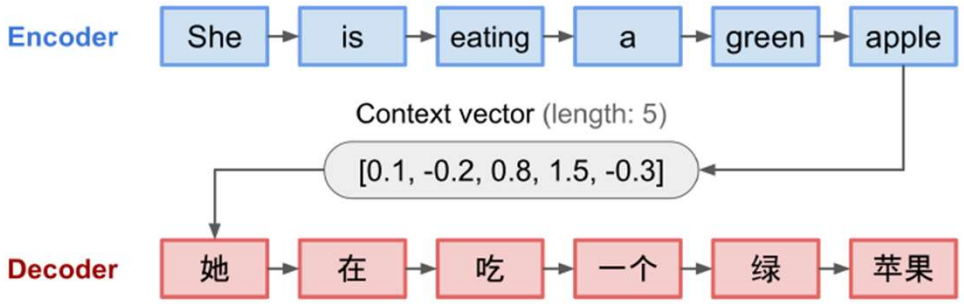


Figure: Context Vector, Source

- **Context Vector (Hidden State):** This vector is expected to convey the meaning of the whole source sequence from the encoder to the decoder.
  - The early work considered the last state of the encoder network as the context vector.

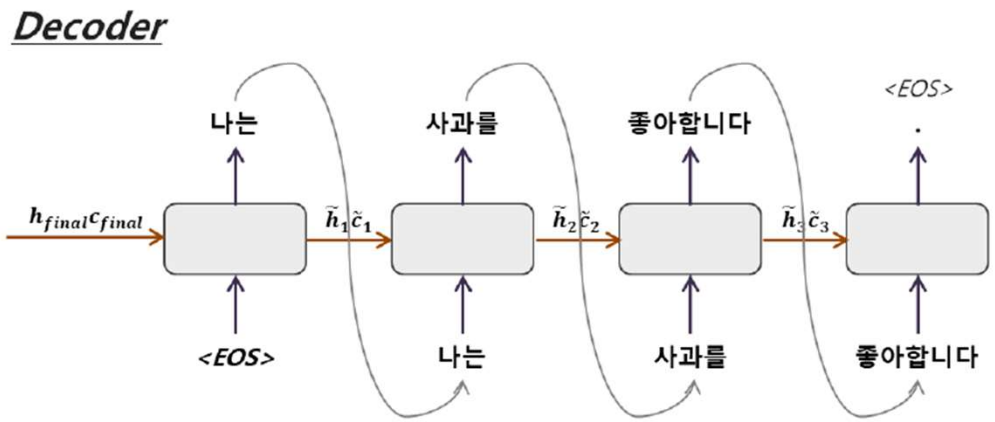
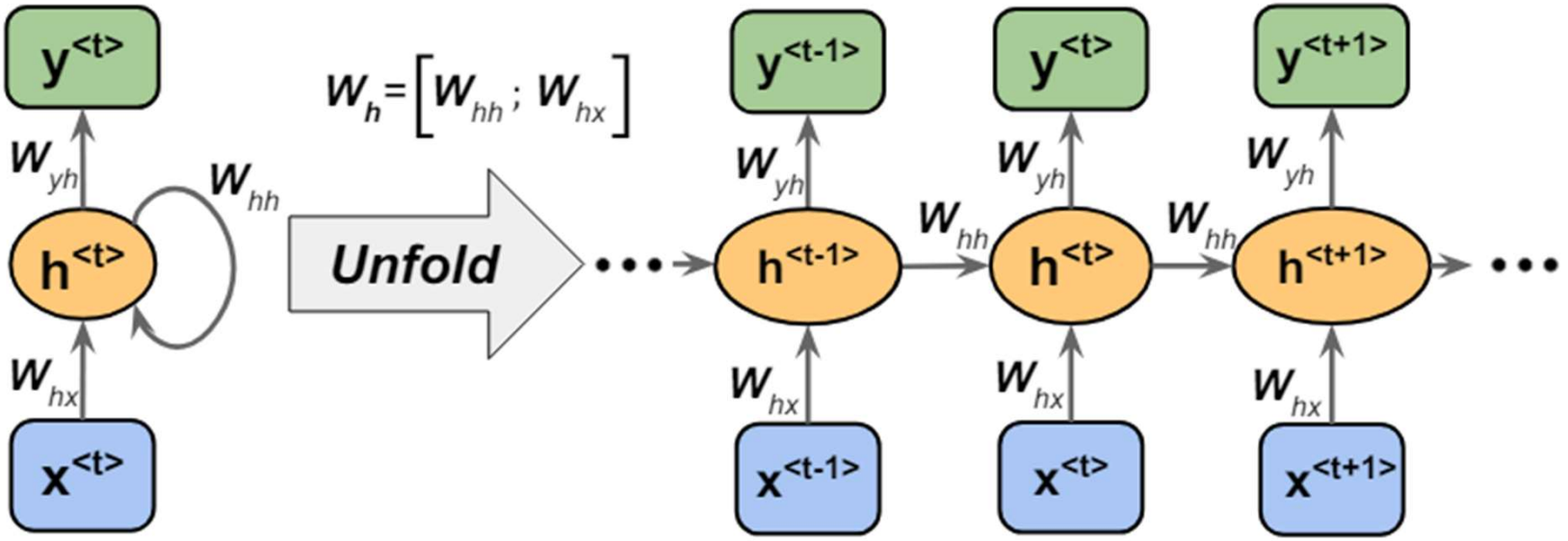


Figure: Decoder, Source

Decoder: The decoder uses the context vector to output the desired target sequence.

Weight matrices in a single-hidden layer RNN



Weight matrices in a single-hidden layer RNN

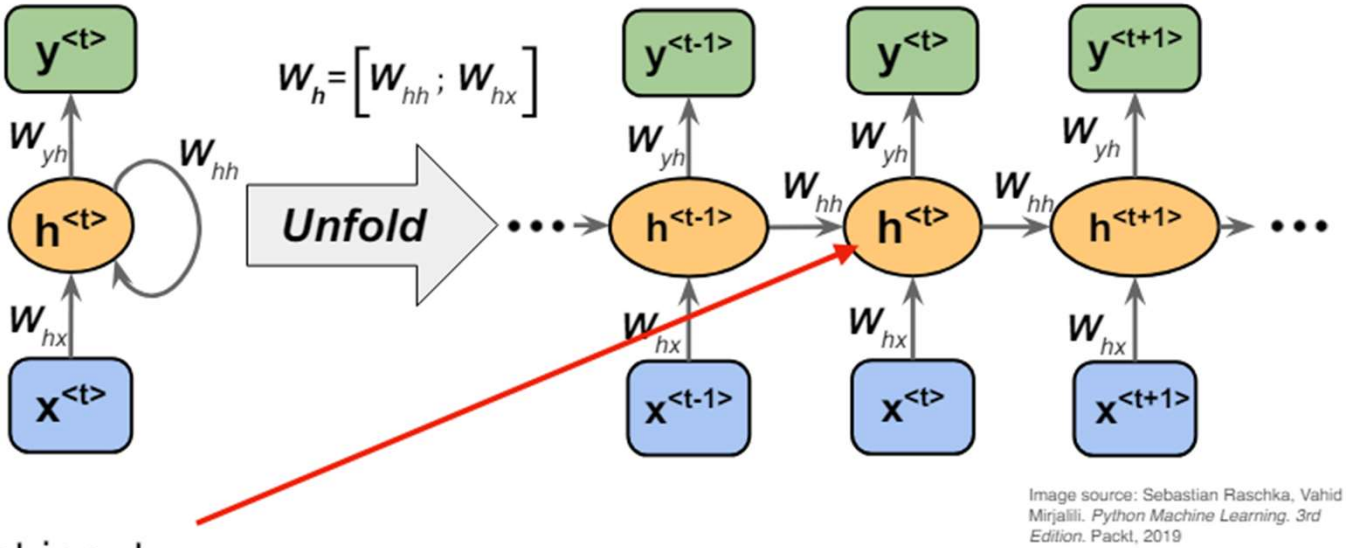


Image source: Sebastian Raschka, Vahid Mirjalili. *Python Machine Learning. 3rd Edition.* Packt, 2019

Net input:

$$\mathbf{z}_h^{\langle t \rangle} = \mathbf{W}_{hx}\mathbf{x}^{\langle t \rangle} + \mathbf{W}_{hh}\mathbf{h}^{\langle t-1 \rangle} + \mathbf{b}_h$$

Activation:

$$\mathbf{h}^{\langle t \rangle} = \sigma_h\left(\mathbf{z}_h^{\langle t \rangle}\right)$$

Weight matrices in a single-hidden layer RNN

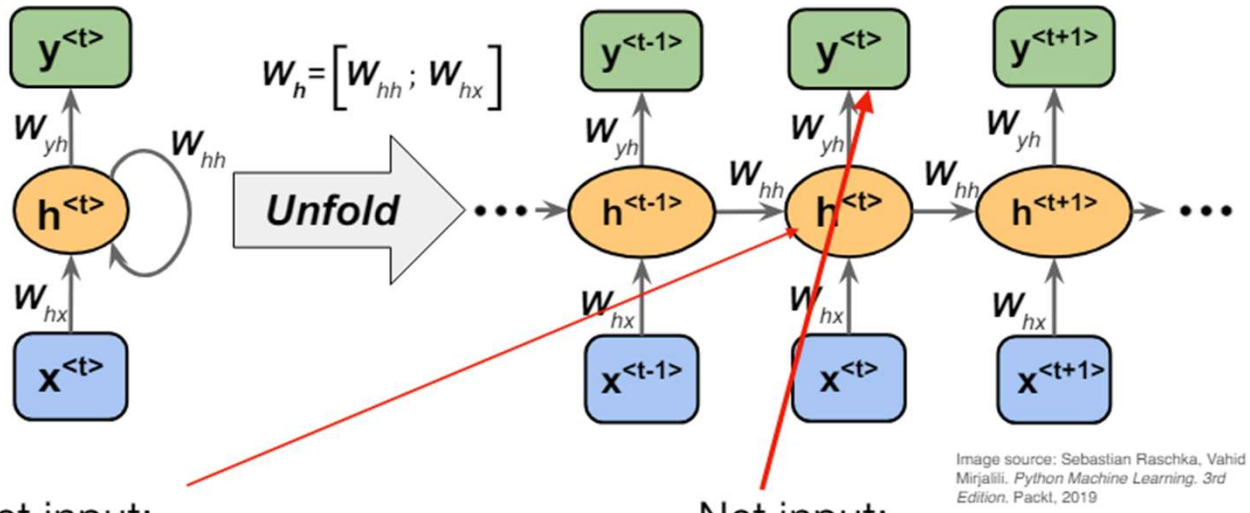


Image source: Sebastian Raschka, Vahid Mirjalili. *Python Machine Learning. 3rd Edition.* Packt, 2019

Net input:

$$\mathbf{z}_h^{\langle t \rangle} = \mathbf{W}_{hx}\mathbf{x}^{\langle t \rangle} + \mathbf{W}_{hh}\mathbf{h}^{\langle t-1 \rangle} + \mathbf{b}_h$$

Activation:

$$\mathbf{h}^{\langle t \rangle} = \sigma_h\left(\mathbf{z}_h^{\langle t \rangle}\right)$$

Net input:

$$\mathbf{z}_y^{\langle t \rangle} = \mathbf{W}_{yh}\mathbf{h}^{\langle t \rangle} + \mathbf{b}_y$$

Output:

$$\mathbf{y}^{\langle t \rangle} = \sigma_y\left(\mathbf{z}_y^{\langle t \rangle}\right)$$

RNN

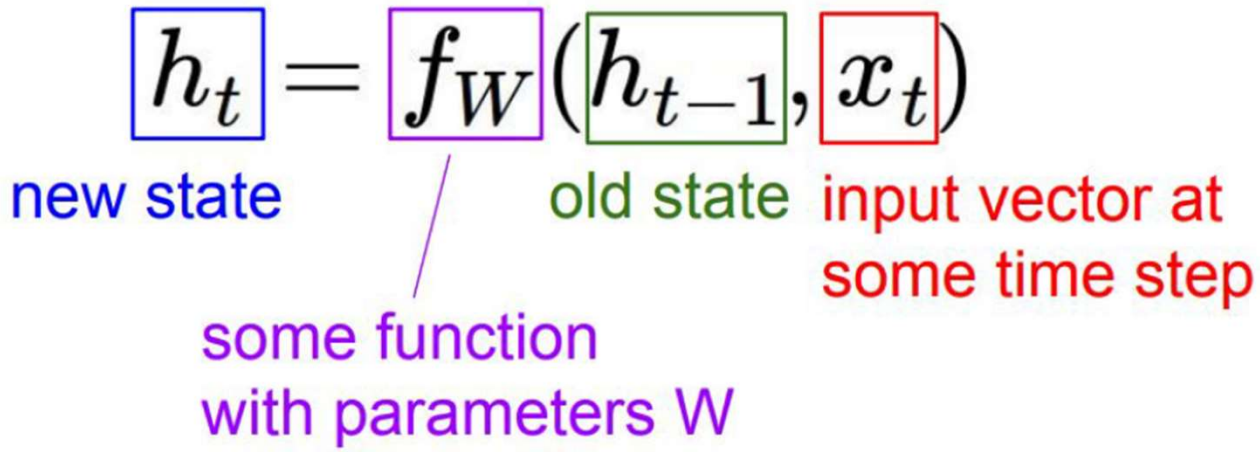


Figure: RNN formula, source

- We can process a sequence of vectors x by applying a recurrence formula at every time step
- The same function and the same set of parameters are used at every time step.

RNN: forward pass

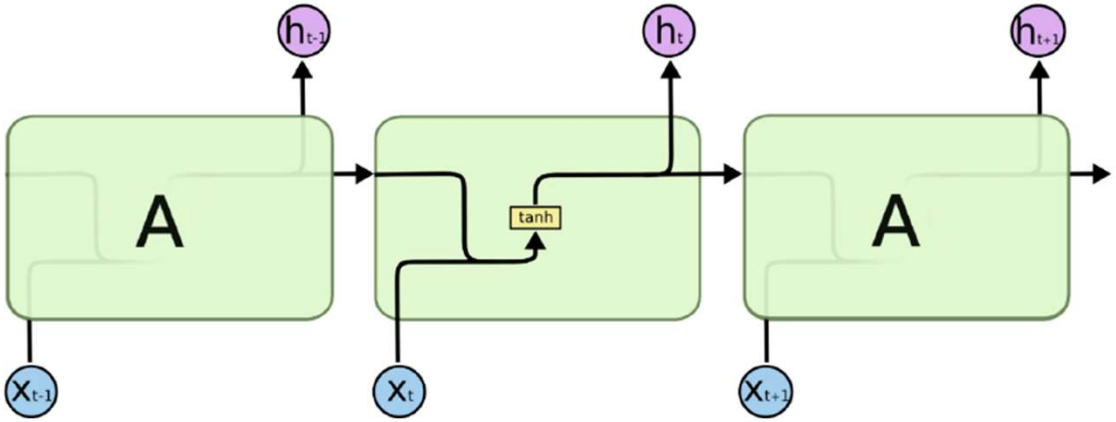
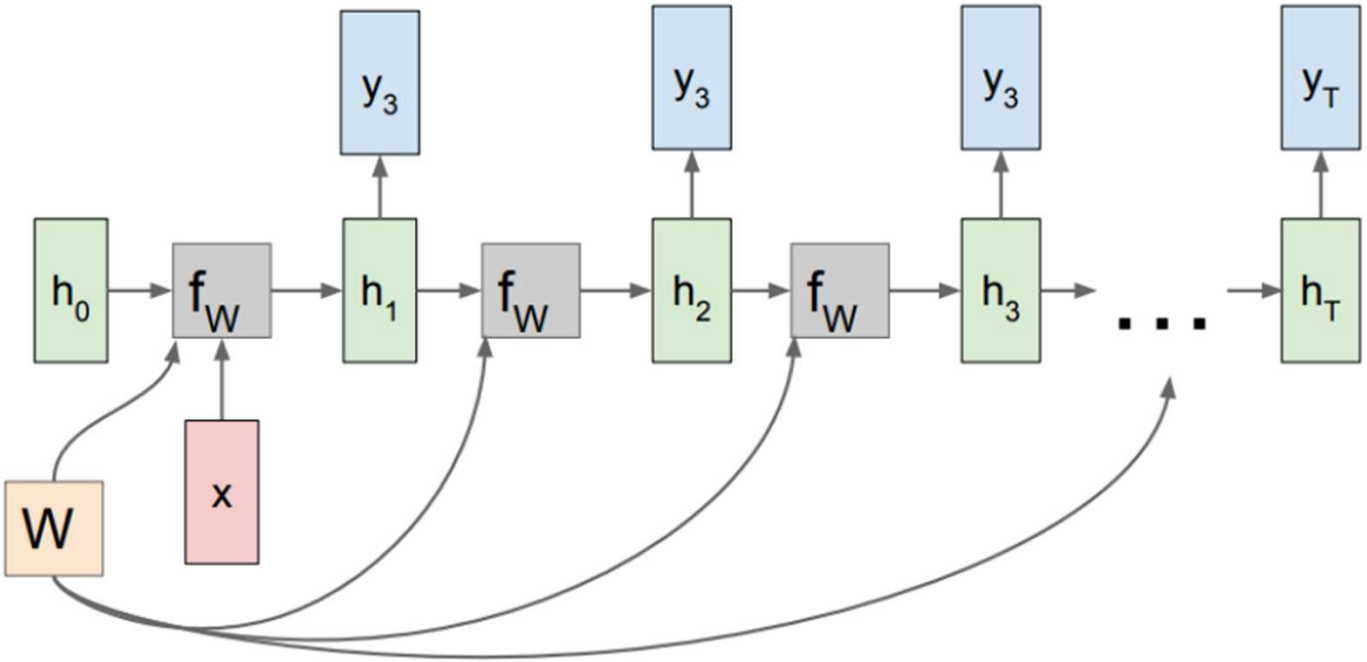


Figure: The repeating module in a standard RNN contains a single layer, source

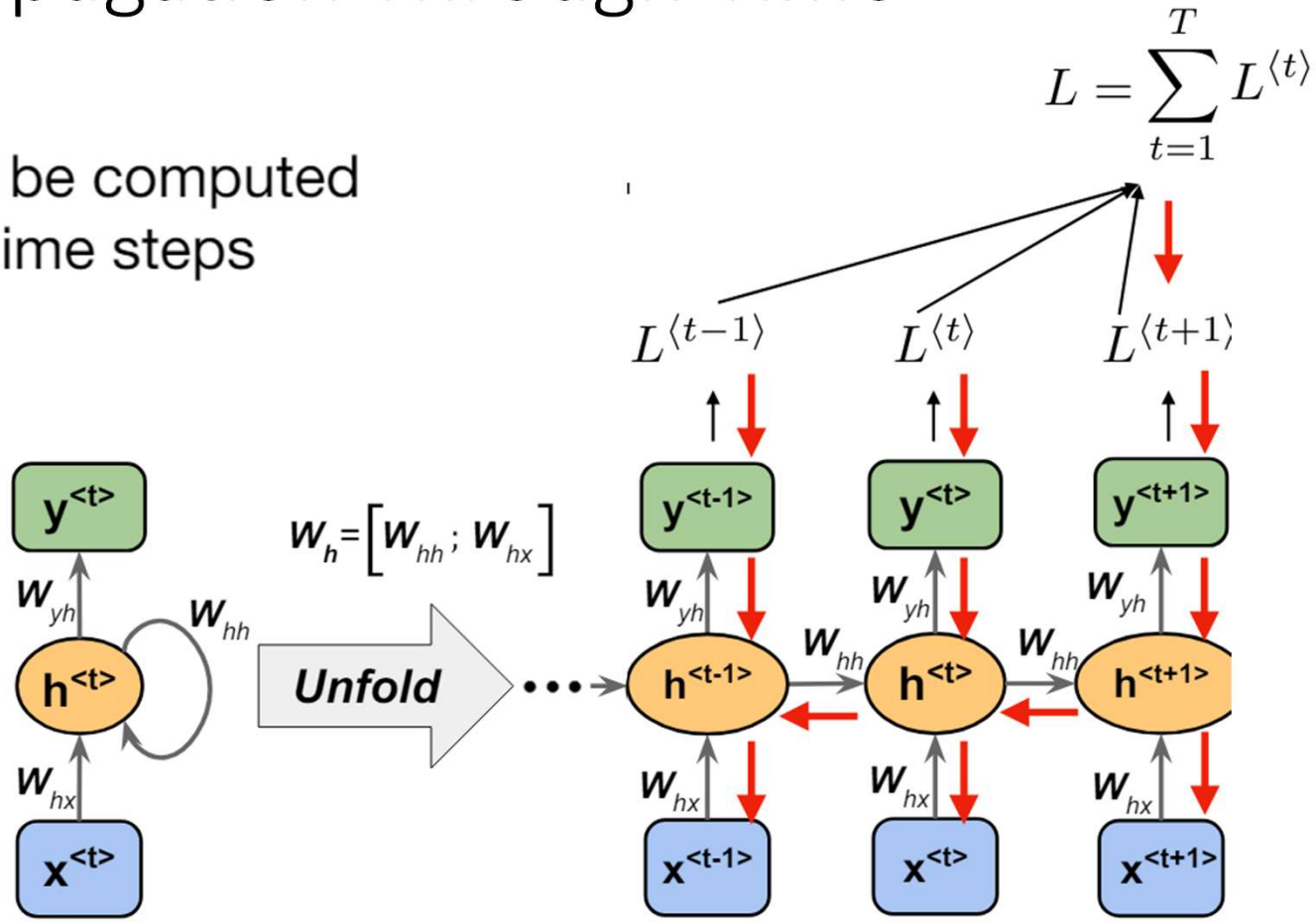
$$
\begin{aligned}
h_t &= \tanh(W_{hh}h_{t-1} + W_{hx}x_t + b_h) \\
&= \tanh((W_{hh}W_{hx}) \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} + b_h) \\
&= \tanh(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} + b_h)
\end{aligned}
$$

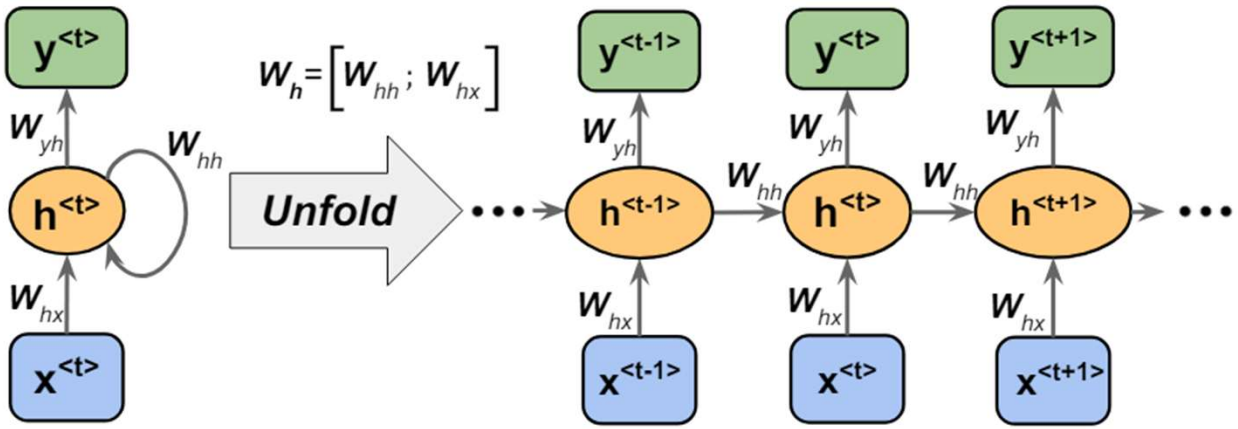
RNN: computational Graph



RNN: Backpropagation Through Time

The overall loss can be computed as the sum over all time steps



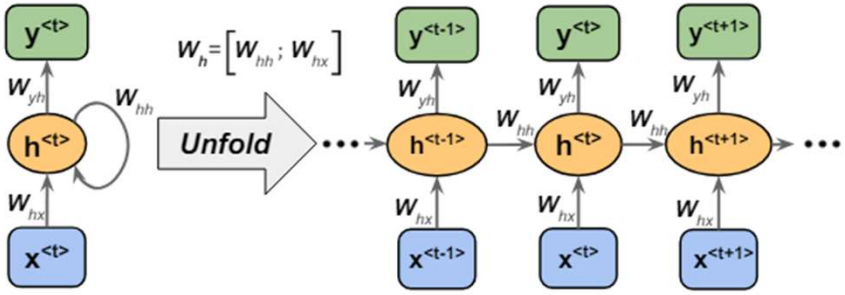


Werbos, Paul J. "Backpropagation through time: what it does and how to do it." *Proceedings of the IEEE* 78, no. 10 (1990): 1550-1560.

$$L = \sum_{t=1}^{T} L^{(t)}$$

$$\frac{\partial L^{(t)}}{\partial \mathbf{W}_{hh}} = \frac{\partial L^{(t)}}{\partial y^{(t)}} \cdot \frac{\partial y^{(t)}}{\partial \mathbf{h}^{(t)}} \cdot \left(\sum_{k=1}^{t} \frac{\partial \mathbf{h}^{(t)}}{\partial \mathbf{h}^{(k)}} \cdot \frac{\partial \mathbf{h}^{(k)}}{\partial \mathbf{W}_{hh}} \right)$$

Backpropagation through time



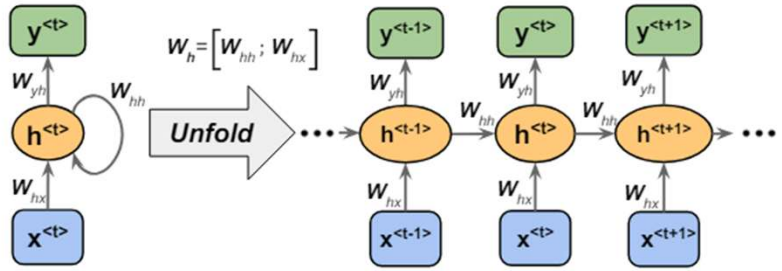
Werbos, Paul J. "Backpropagation through time: what it does and how to do it." *Proceedings of the IEEE* 78, no. 10 (1990): 1550-1560.

$$\frac{\partial L^{(t)}}{\partial \mathbf{W}_{hh}} = \frac{\partial L^{(t)}}{\partial y^{(t)}} \cdot \frac{\partial y^{(t)}}{\partial \mathbf{h}^{(t)}} \cdot \left(\sum_{k=1}^{t} \frac{\partial \mathbf{h}^{(t)}}{\partial \mathbf{h}^{(k)}} \cdot \frac{\partial \mathbf{h}^{(k)}}{\partial \mathbf{W}_{hh}} \right)$$

computed as a multiplication of adjacent time steps:

$$\frac{\partial \mathbf{h}^{(t)}}{\partial \mathbf{h}^{(k)}} = \prod_{i=k+1}^{t} \frac{\partial \mathbf{h}^{(i)}}{\partial \mathbf{h}^{(i-1)}}$$

Backpropagation through time



Werbos, Paul J. "Backpropagation through time: what it does and how to do it." *Proceedings of the IEEE* 78, no. 10 (1990): 1550-1560.

$$L = \sum_{t=1}^{T} L^{(t)} \qquad \frac{\partial L^{(t)}}{\partial \mathbf{W}_{hh}} = \frac{\partial L^{(t)}}{\partial y^{(t)}} \cdot \frac{\partial y^{(t)}}{\partial \mathbf{h}^{(t)}} \cdot \left(\sum_{k=1}^{t} \frac{\partial \mathbf{h}^{(t)}}{\partial \mathbf{h}^{(k)}} \cdot \frac{\partial \mathbf{h}^{(k)}}{\partial \mathbf{W}_{hh}} \right)$$

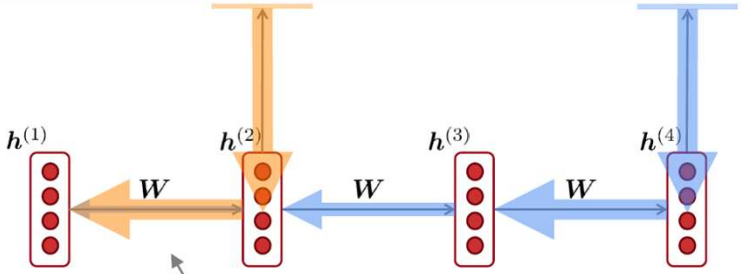
computed as a multiplication of adjacent time steps:

This is very problematic:
Vanishing/Exploding gradient problem!

$$\frac{\partial \mathbf{h}^{(t)}}{\partial \mathbf{h}^{(k)}} = \prod_{i=k+1}^{t} \frac{\partial \mathbf{h}^{(i)}}{\partial \mathbf{h}^{(i-1)}}$$

Why vanishing gradient a problem?

The derivative $\frac{\partial h_k}{\partial h_1}$ is essentially telling us how much our hidden state at time k will change when we change the hidden state at time 1 by a little bit. According to the above math, if the gradient vanishes it means the earlier hidden states have no real effect on the later hidden states, meaning no long term dependencies are learned! This can be formally proved, and has been in many papers, including the original LSTM paper.



Gradient signal from faraway is lost because it's much smaller than gradient signal from close-by.

So model weights are only updated only with respect to near effects, not long-term effects.

Effect of vanishing gradient on RNN-LM

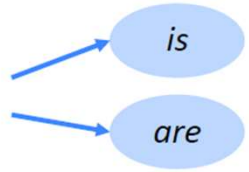
- **LM task:** *When she tried to print her tickets, she found that the printer was out of toner. She went to the stationery store to buy more toner. It was very overpriced. After installing the toner into the printer, she finally printed her _____*

- To learn from this training example, the RNN-LM needs to model the dependency between *"tickets"* on the 7th step and the target word *"tickets"* at the end.

- But if gradient is small, the model can't learn this dependency
  - So the model is unable to predict similar long-distance dependencies at test time

Effect of vanishing gradient on RNN-LM

- **LM task:** *The writer of the books ___*



- **Correct answer**: *The writer of the books <u>is</u> planning a sequel*
- **Syntactic recency:** *The <u>writer</u> of the books <u>is</u>* (correct)
- **Sequential recency:** *The writer of the <u>books</u> <u>are</u>* (incorrect)
- Due to vanishing gradient, RNN-LMs are better at learning from sequential recency than syntactic recency, so they make this type of error more often than we'd like [Linzen et al 2016]

Solutions to the vanishing/exploding gradient problems

1) Gradient Clipping: set a max value for gradients if they grow to large (solves only exploding gradient problem)

2) Truncated backpropagation through time (TBPTT)
simply limits the number of time steps the signal can backpropagate after each forward pass. E.g., even if the
sequence has 100 elements/steps, we may only backpropagate through 20 or so

3) Long short-term memory (LSTM) -- uses a memory cell for modeling long-range dependencies and avoid vanishing gradient problems

Hochreiter, Sepp, and Jürgen Schmidhuber. "Long short-term memory."
Neural computation 9, no. 8 (1997): 1735-1780.

Long-short term memory (LSTM)

LSTM cell:

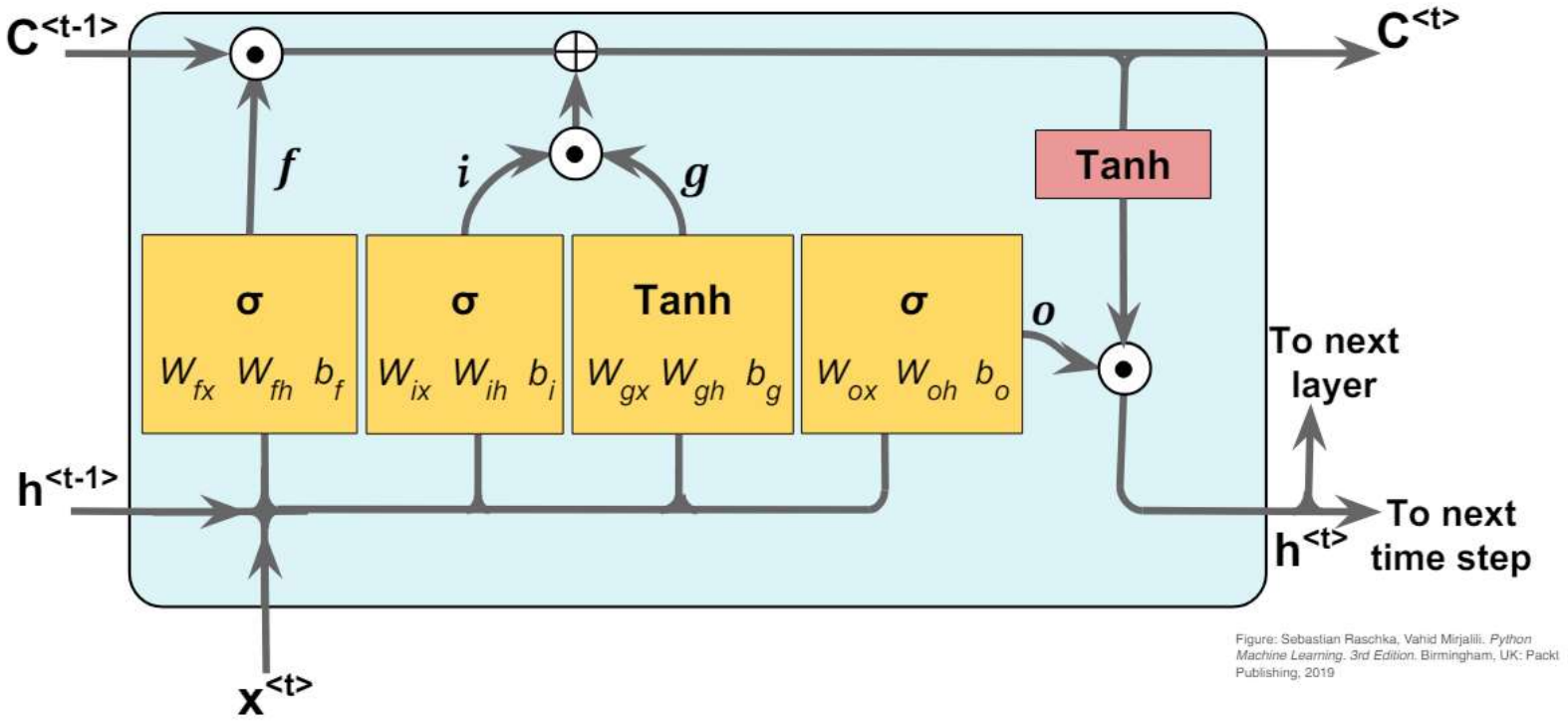
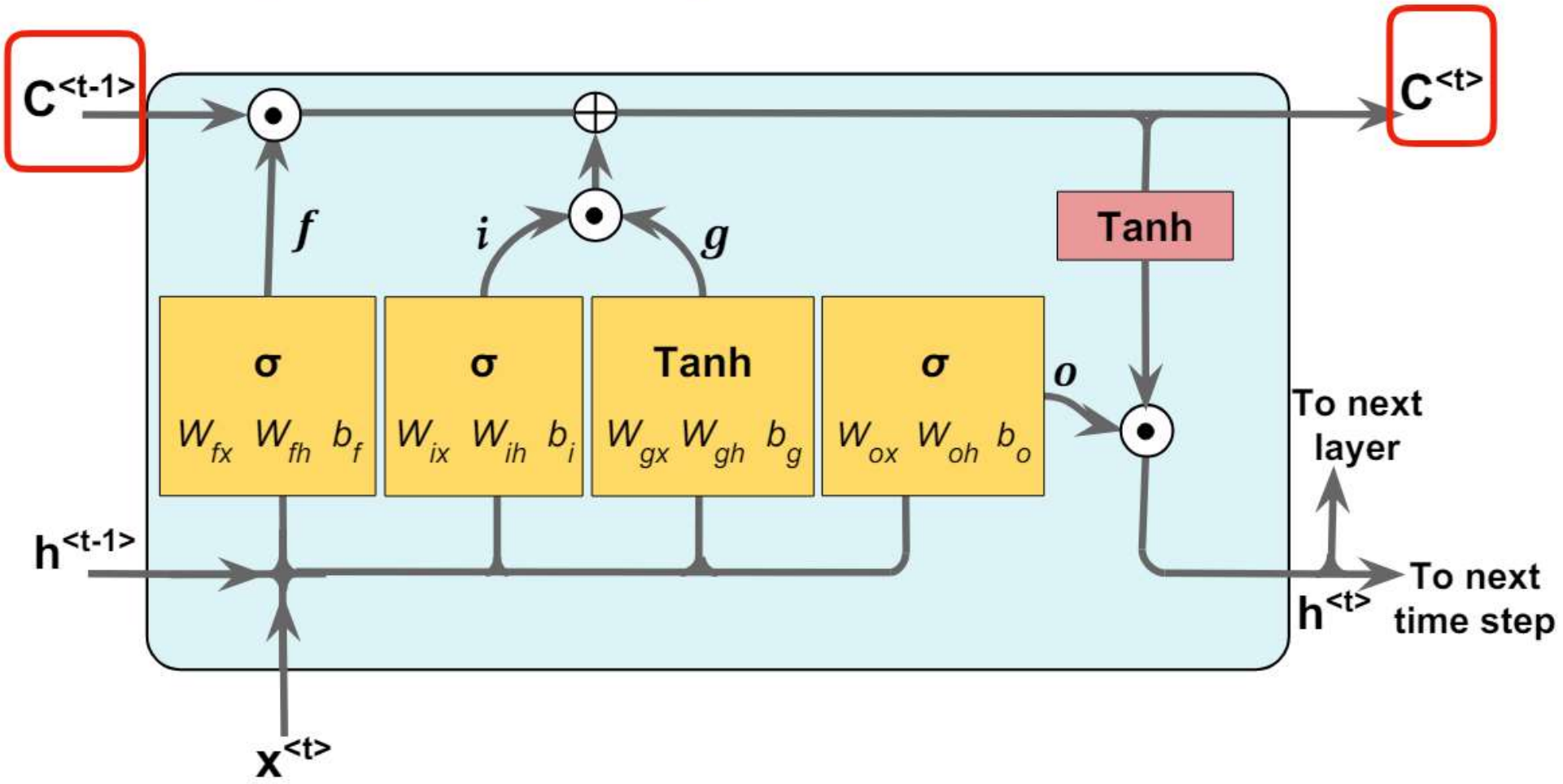
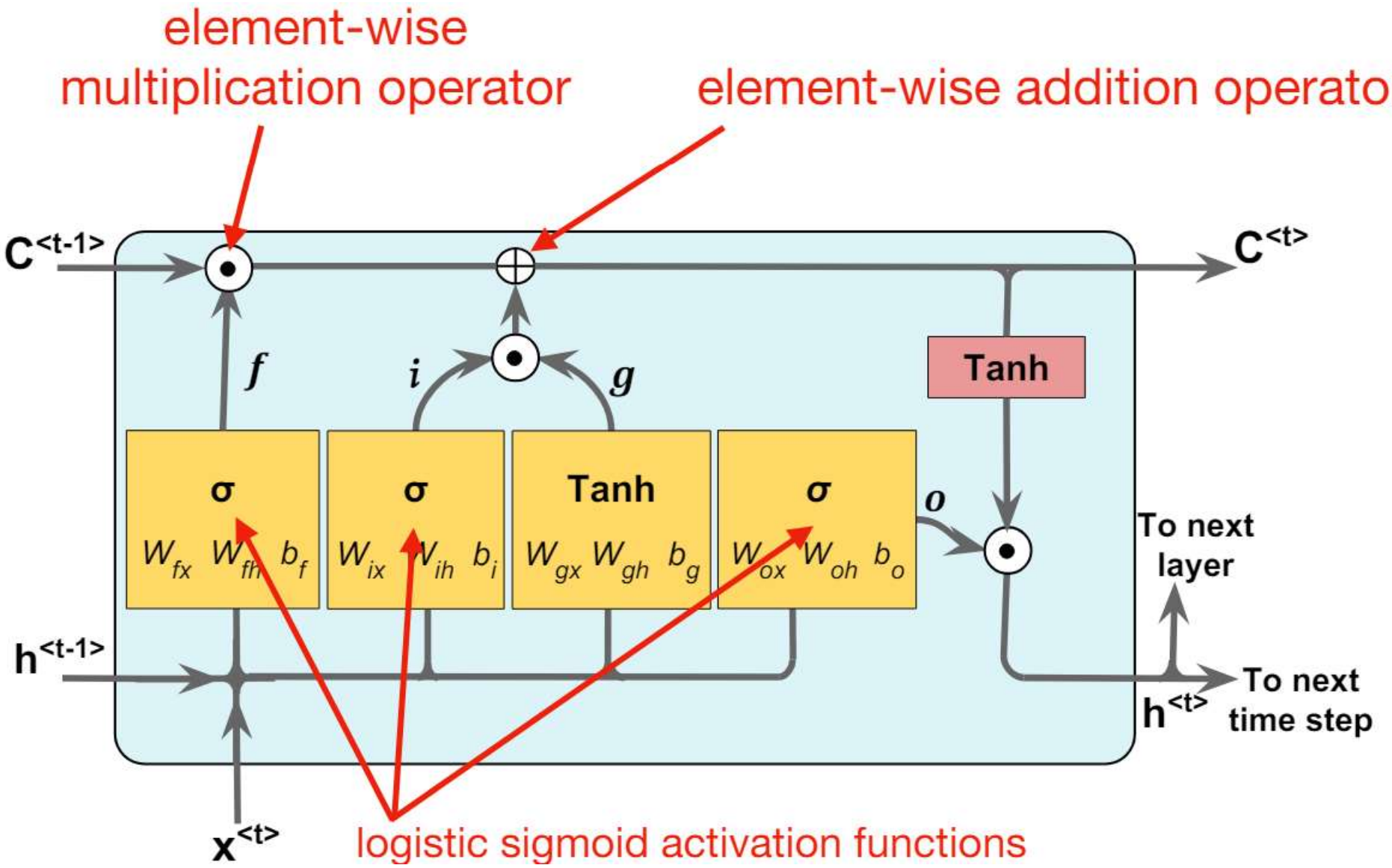


Figure: Sebastian Raschka, Vahid Mirjalili. *Python Machine Learning. 3rd Edition*. Birmingham, UK: Packt Publishing, 2019

Cell state at previous time step

Cell state at current time step

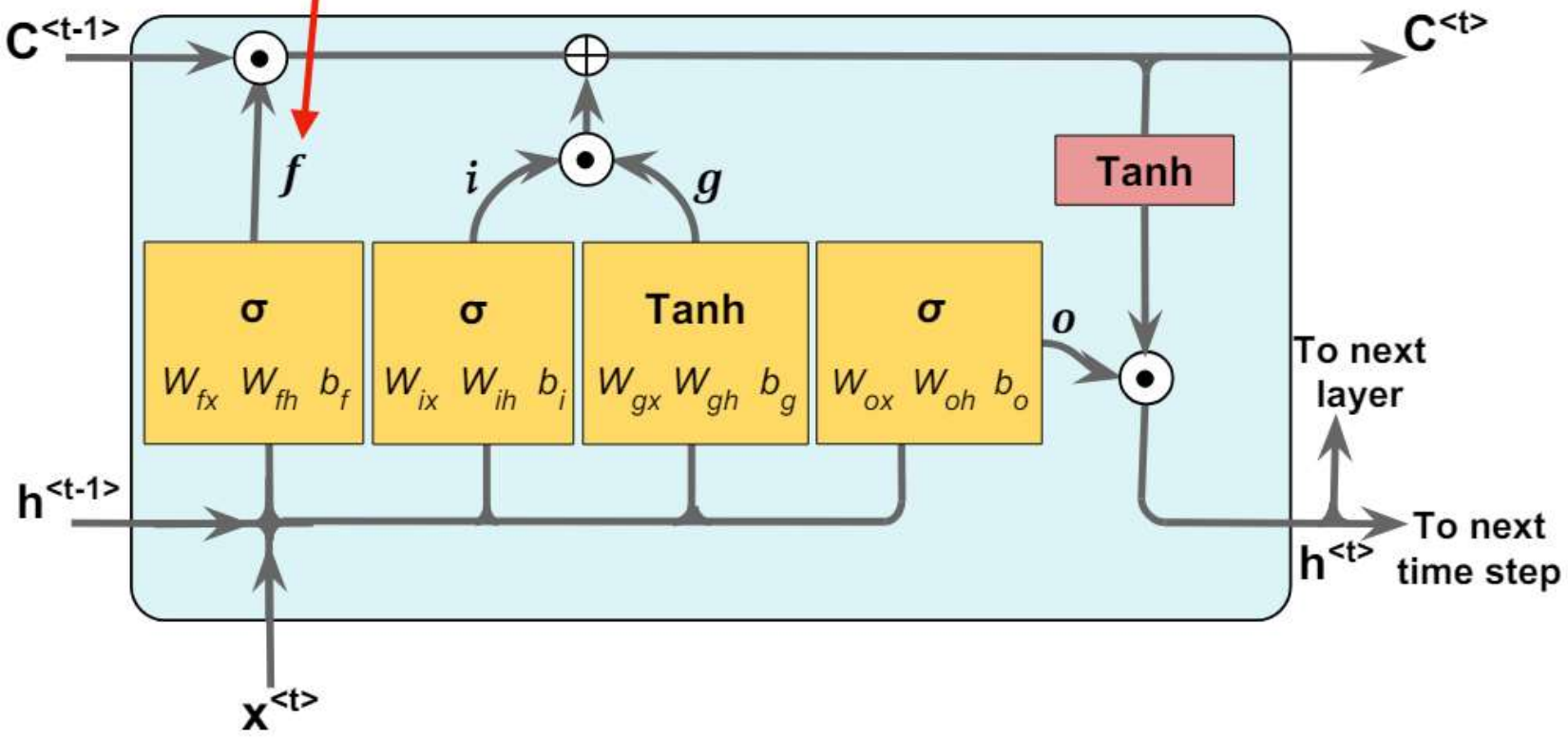



element-wise
multiplication operator
element-wise addition operator
C<t-1>
C<t>
f
i
g
Tanh
σ
σ
Tanh
σ
o
W_fx W_fh b_f
W_ix W_ih b_i
W_gx W_gh b_g
W_ox W_oh b_o
To next
layer
h<t-1>
h<t>
To next
time step
x<t>
logistic sigmoid activation functions

Gers, Felix A., Jürgen Schmidhuber, and Fred Cummins. "Learning to forget: Continual prediction with LSTM." (1999): 850-855.

"Forget Gate": controls which information is remembered, and which is forgotten; can reset the cell state

$$f_t = \sigma\left(\mathbf{W}_{fx}\mathbf{x}^{\langle t\rangle} + \mathbf{W}_{fh}\mathbf{h}^{\langle t-1\rangle} + \mathbf{b}_f\right)$$



The first step in our LSTM is to decide what information we're going to throw away from the cell state.

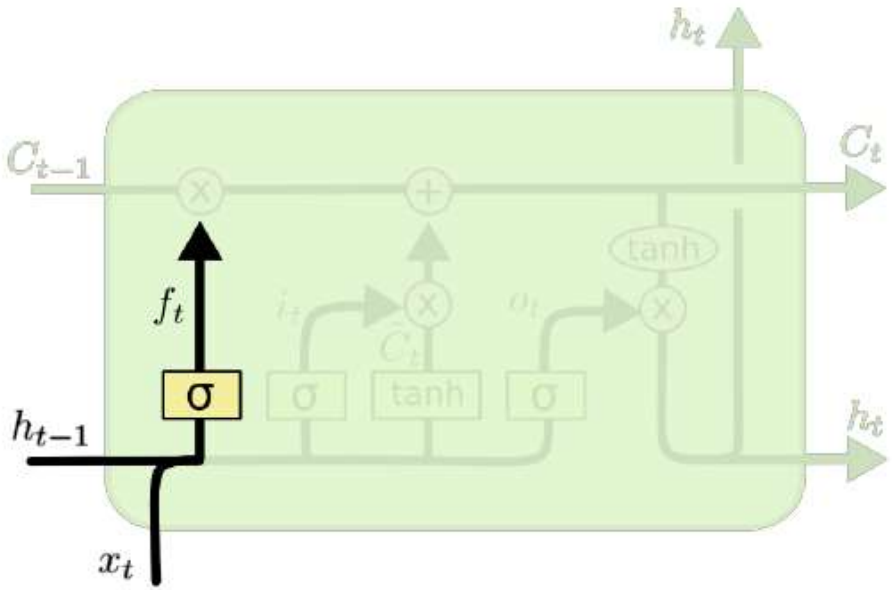
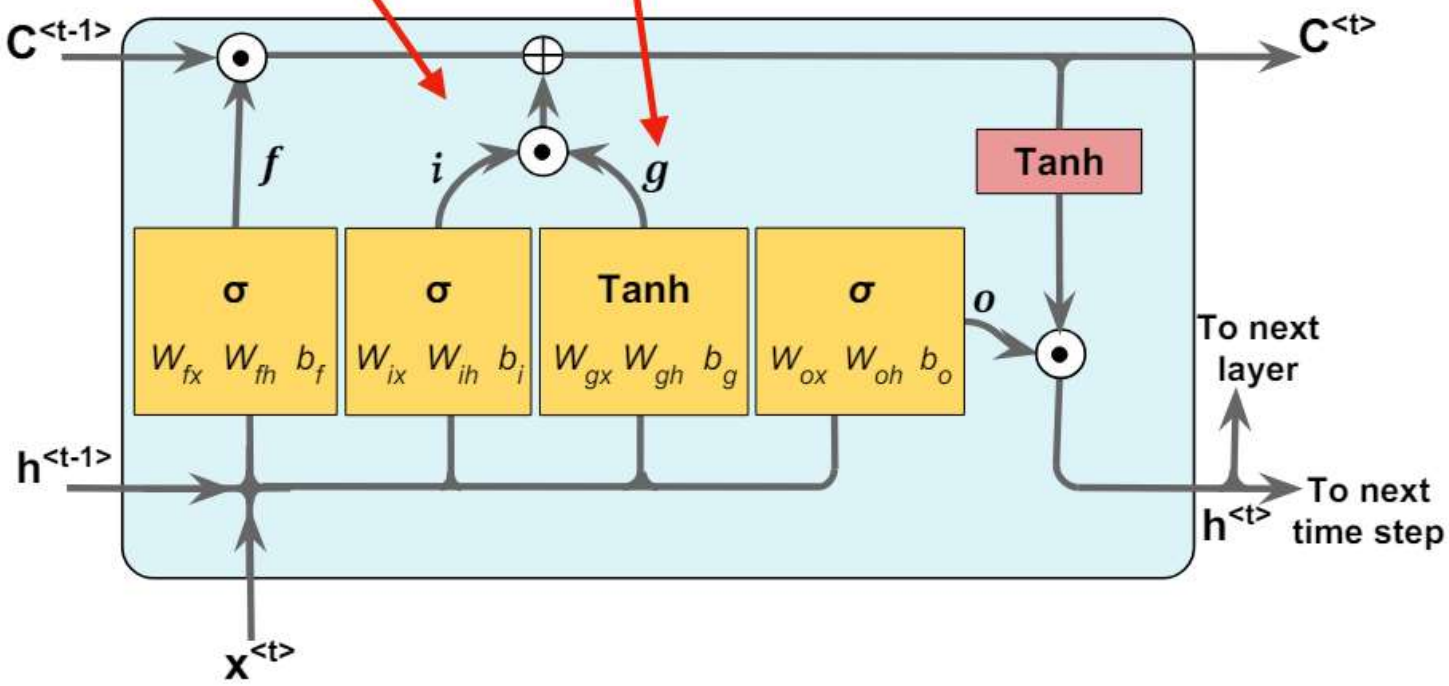


Figure: Forget Gate Layer

"Input Gate": $\mathbf{i}_t = \sigma\left(\mathbf{W}_{ix}\mathbf{x}^{\langle t\rangle} + \mathbf{W}_{ih}\mathbf{h}^{\langle t-1\rangle} + \mathbf{b}_i\right)$

"Input Node":

$$\mathbf{g}_t = \tanh\left(\mathbf{W}_{gx}\mathbf{x}^{\langle t\rangle} + \mathbf{W}_{gh}\mathbf{h}^{\langle t-1\rangle} + \mathbf{b}_g\right)$$



The next step is to decide what new information we're going to store in the cell state.

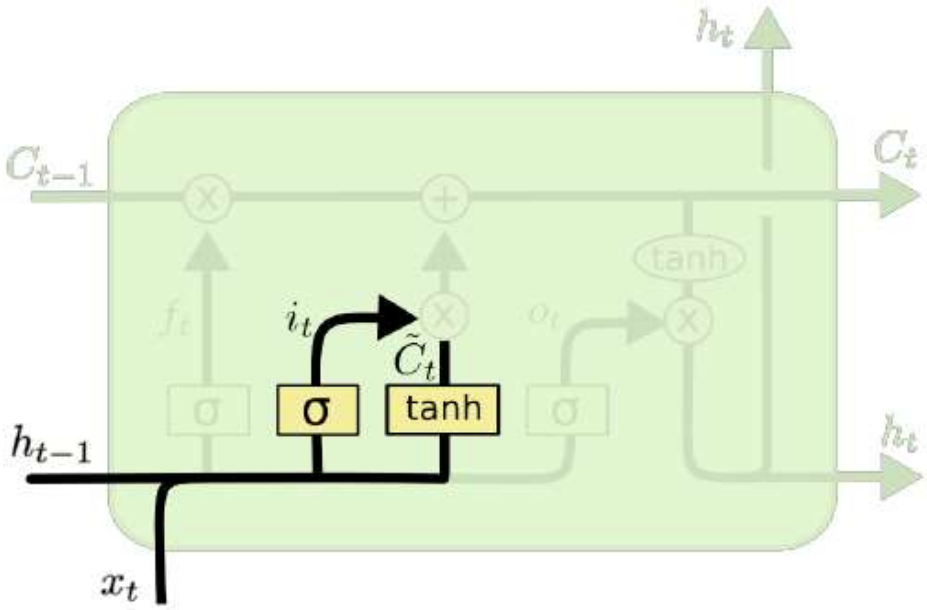
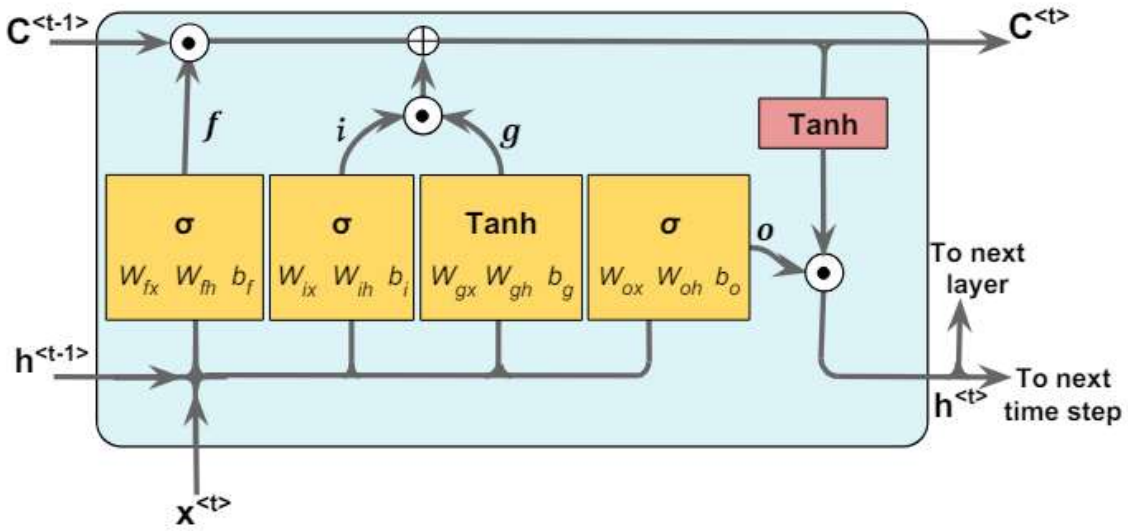


Figure: Input Gate Layer

Forget Gate Input Node Input Gate

$$C^{\langle t \rangle} = \left(C^{\langle t-1 \rangle} \odot f_t \right) \oplus \left(i_t \odot g_t \right)$$

For updating the cell state



It's now time to update the old cell state, Ct−1, into the new cell state Ct.

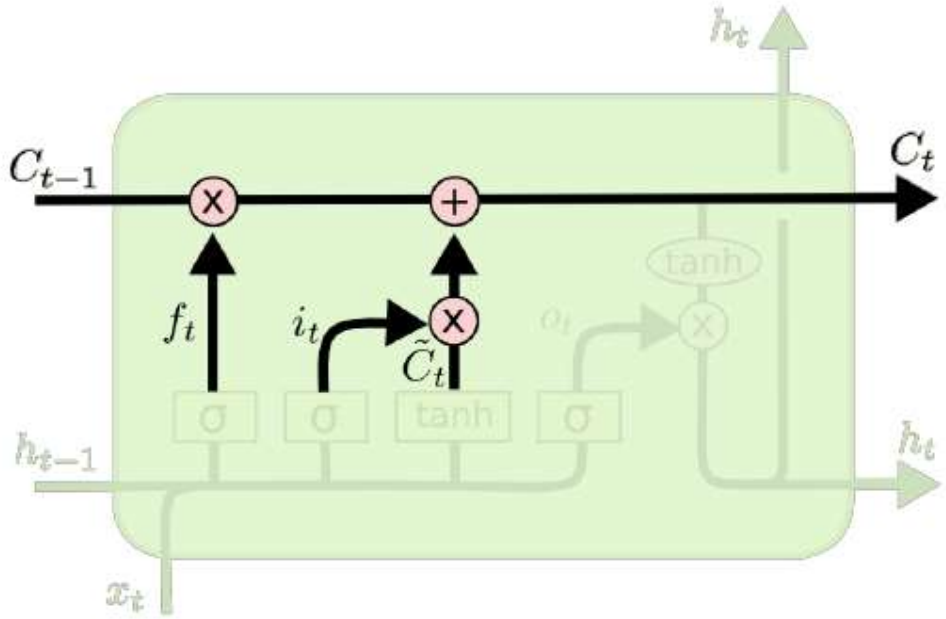
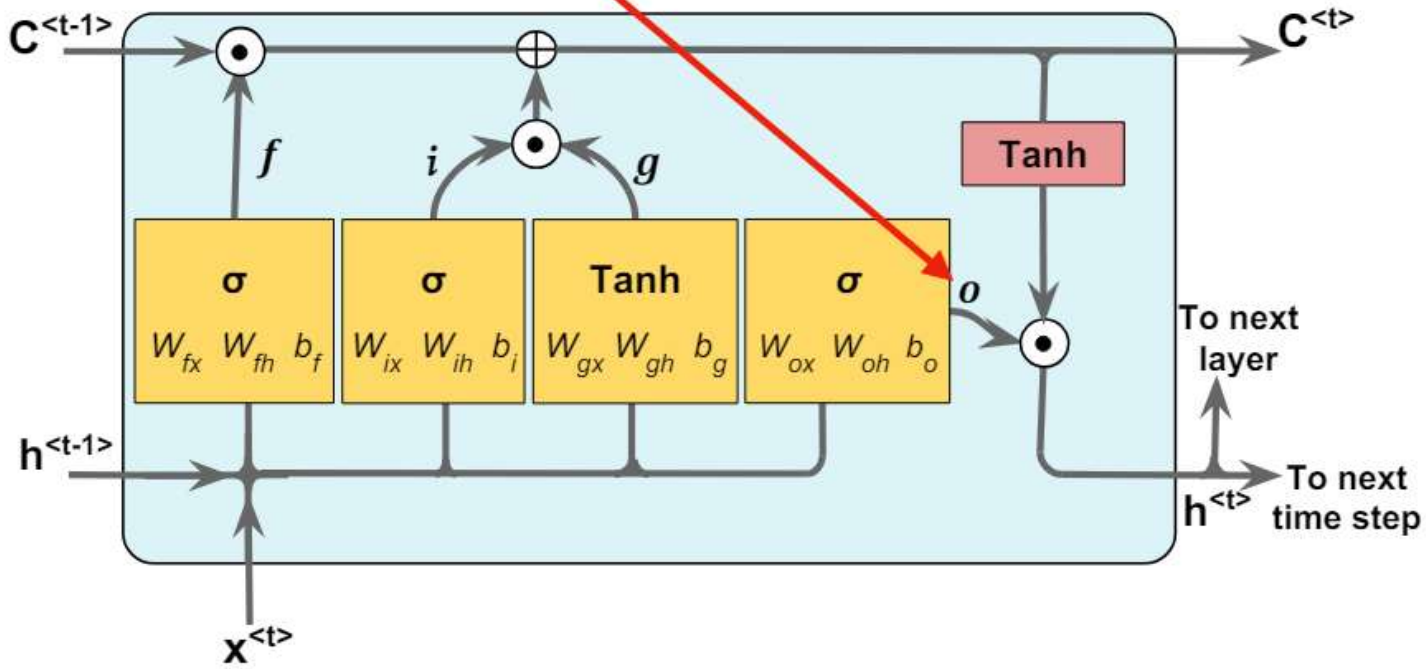


Figure: Update Cell State

Output gate for updating the values of hidden units:

$$\mathbf{o}_t = \sigma\left(\mathbf{W}_{ox}\mathbf{x}^{\langle t\rangle} + \mathbf{W}_{oh}\mathbf{h}^{\langle t-1\rangle} + \mathbf{b}_o\right)$$



Finally, we need to decide what we're going to output. This output will be based on our cell state

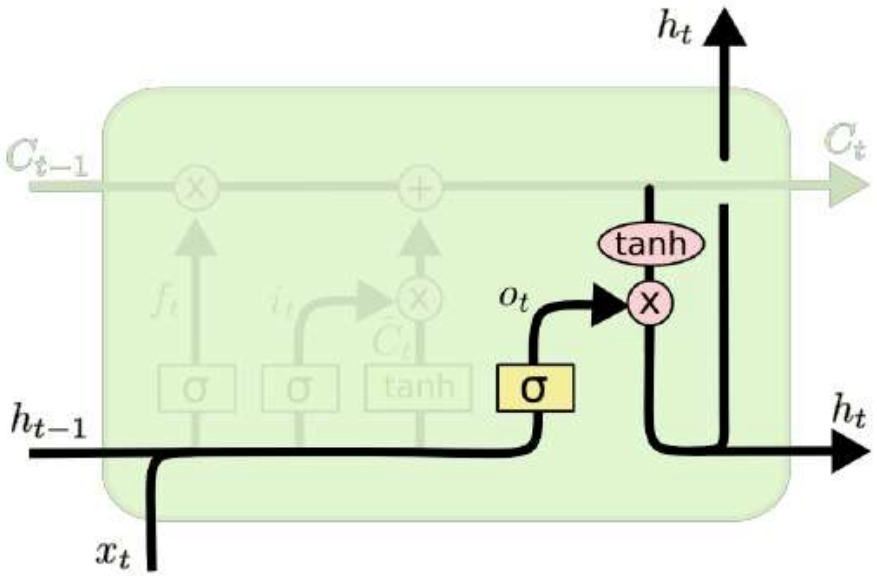
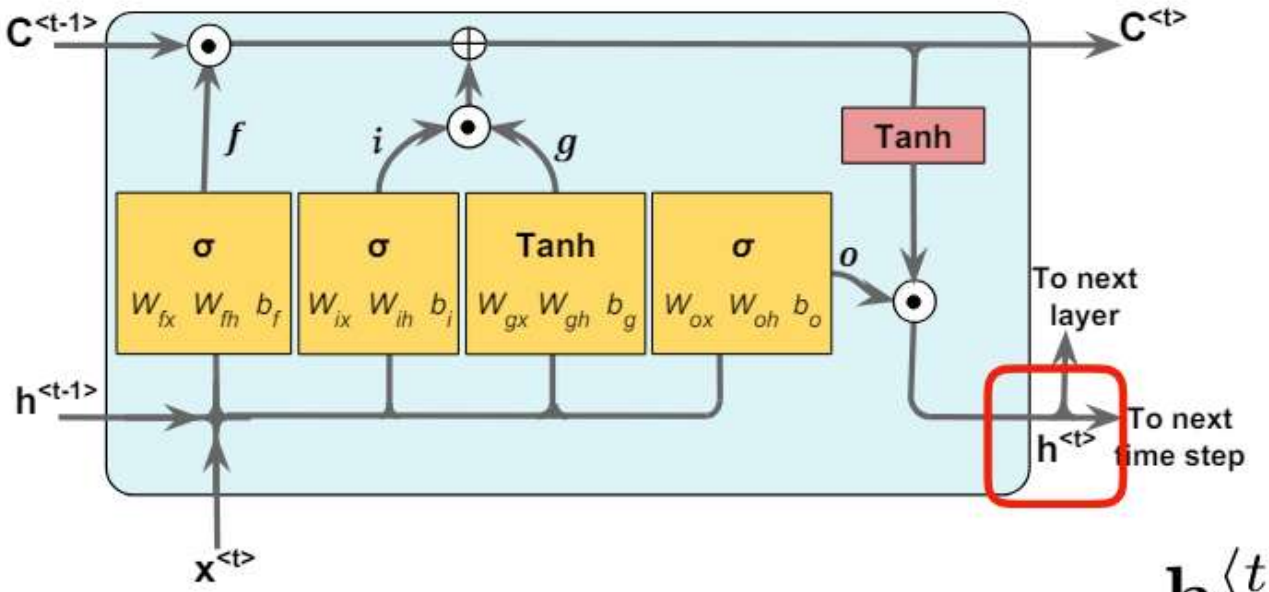


Figure: Output Gate Layer



$$\mathbf{h}^{\langle t \rangle} = \mathbf{o}_t \odot \tanh\left(\mathbf{C}^{\langle t \rangle}\right)$$

LSTM

We have a sequence of inputs $\boldsymbol{x}^{(t)}$, and we will compute a sequence of hidden states $\boldsymbol{h}^{(t)}$ and cell states $\boldsymbol{c}^{(t)}$. On timestep t:

Forget gate: controls what is kept vs forgotten, from previous cell state

Input gate: controls what parts of the new cell content are written to cell

Output gate: controls what parts of cell are output to hidden state

New cell content: this is the new content to be written to the cell

Cell state: erase ("forget") some content from last cell state, and write ("input") some new cell content

Hidden state: read ("output") some content from the cell

Sigmoid function: all gate values are between 0 and 1

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

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

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

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

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

$$\boldsymbol{h}^{(t)} = \boldsymbol{o}^{(t)} \circ \tanh \boldsymbol{c}^{(t)}$$

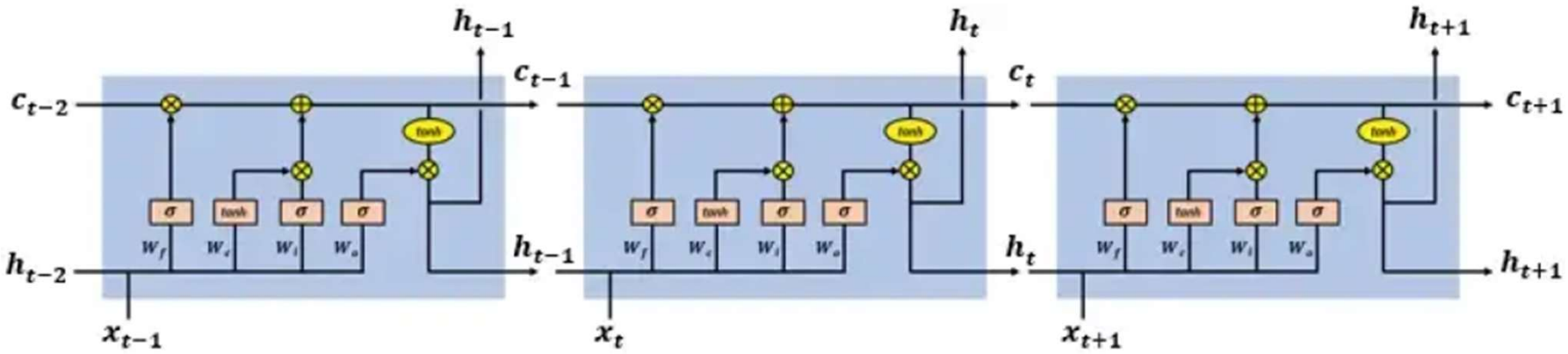
All these are vectors of same length n

Gates are applied using element-wise product

Why LSTM ?

- The LSTM does have the ability to remove and add information to the cell state.
- The gates in the previous slide let the information to pass through the units.
- The gates value are between zero and one and specify how much information should be let through .
- They also somehow solve the vanishing gradient .
- We can use more blocks of them so there will be more information to remember.

How LSTM solve vanishing gradients?



How LSTM solve vanishing gradients?

In LSTM we also have

$$\frac{\partial L_k}{\partial W} = \frac{\partial L_k}{\partial h_k}\frac{\partial h_k}{\partial c_k}\left(\prod_{t=2}^{k}\frac{\partial c_t}{\partial c_{t-1}}\right)\frac{\partial c_1}{\partial W}$$

But

$$c^t = \Gamma_u * \tilde{c}^t + (\Gamma_f) * c^{t-1}$$

$$\frac{\partial c_t}{\partial c_{t-1}} = \frac{\partial \Gamma_f}{\partial c_{t-1}}.c_{t-1} + \Gamma_f + \frac{\partial \Gamma_u}{\partial c_{t-1}}.\tilde{c_t} + \frac{\partial \tilde{c_t}}{\partial c_{t-1}}.\Gamma_u$$

Consider that the *forget gate* is added to other terms and allows better control of gradient values. But it doesn't guarantee that there is no vanishing or exploding in gradient.

Bidirectional RNN

- How to get information from future?