یادگیری ماشین

شبکه‌های بازگشتی
پردازش متن

پاییز۹۹


بسم
الله
الرحمن
الرحيم

کلمات کلیدی این بخش

- Tokenization
- Word Embedding
- recurrent neural networks (RNNs)
- Long Short-Term Memory (LSTM)

فهرست مطالب

کلیات پردازش متن

- تلاش برای فهم زبان طبیعی(NLU)
- در زبان طبیعی ترتیب کلمات در دنباله‌ها اهمیت بسیاری دارد

با این که کیفیت صداش افتزاح بود ولی در کل خوب بود.(نظر مثبت)

با این که کیفیت صداش خوب بود ولی در کل افتزاح بود.(نظر منفی)

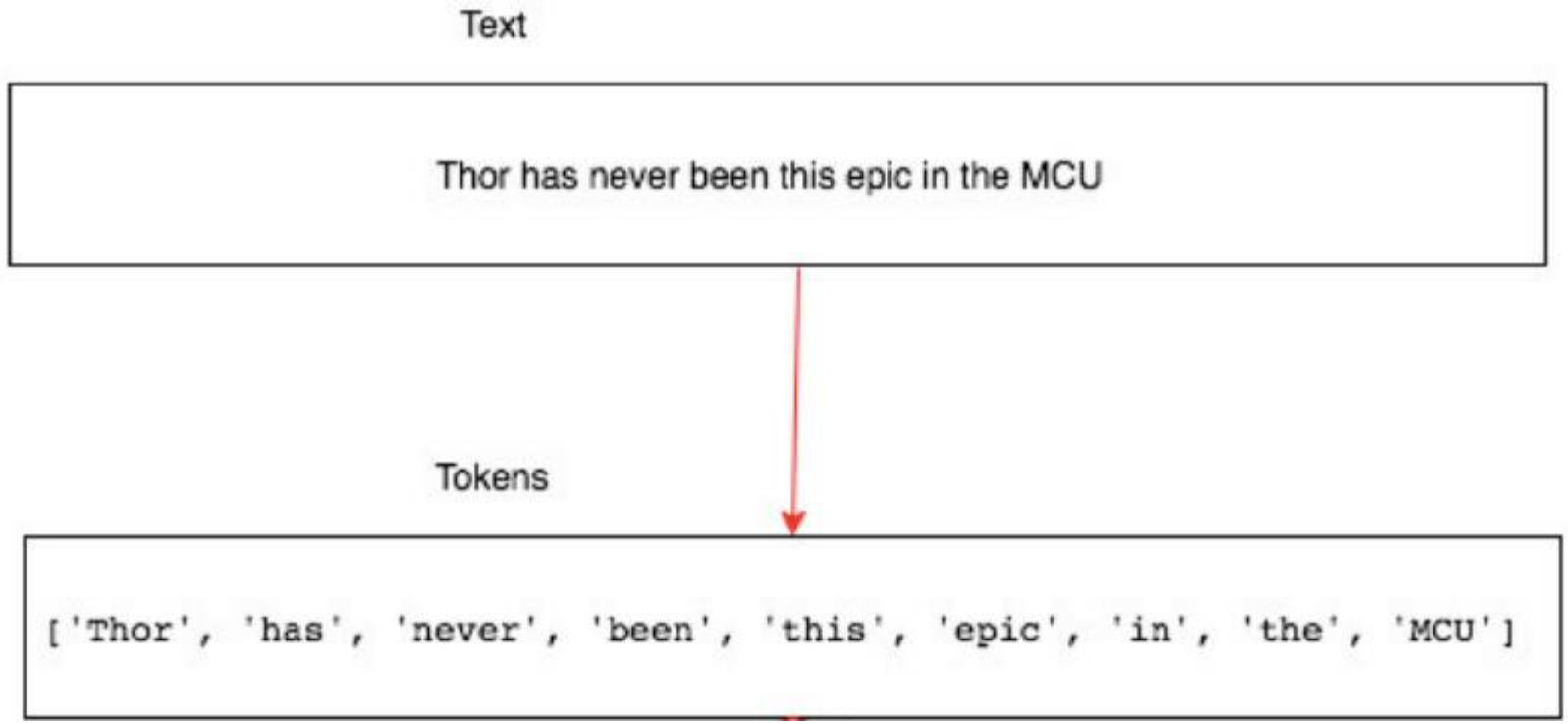
کلیات پردازش متن-تبدیل متن به ...؟

- برخلاف **تصویر** که داده‌ای بسیار با معنا برای ماشین است **متن** یک داده‌ی سخت‌فهم است.

Tokenization

- بردار کلمات
- بردار کاراکترها
- بردار ngram ها (کم کاربرد)

Tokenization(word)



Tokenization(word)

```
print(Thor_review.split())

#Results

['the', 'action', 'scenes', 'were', 'top', 'notch', 'in', 'this', 'movie.',
'Thor', 'has', 'never', 'been', 'this', 'epic', 'in', 'the', 'MCU.', 'He',
'does', 'some', 'pretty', 'epic', 'sh*t', 'in', 'this', 'movie', 'and',
'he', 'is', 'definitely', 'not', 'under-powered', 'anymore.', 'Thor', 'in',
'unleashed', 'in', 'this,', 'I', 'love', 'that.']
```

Tokenization(character)

```
thor_review = "the action scenes were top notch in this movie. Thor has
never been this epic in the MCU. He does some pretty epic sh*t in this
movie and he is definitely not under-powered anymore. Thor in unleashed in
this, I love that."

print(list(thor_review))
```

```
#Results
['t', 'h', 'e', ' ', 'a', 'c', 't', 'i', 'o', 'n', ' ', 's', 'c', 'e', 'n',
'e', 's', ' ', 'w', 'e', 'r', 'e', ' ', 't', 'o', 'p', ' ', 'n', 'o', 't',
'c', 'h', ' ', 'i', 'n', ' ', 't', 'h', 'i', 's', ' ', 'm', 'o', 'v', 'i',
'e', '.', ' ', 'T', 'h', 'o', 'r', ' ', 'h', 'a', 's', ' ', 'n', 'e', 'v',
'e', 'r', ' ', 'b', 'e', 'e', 'n', ' ', 't', 'h', 'i', 's', ' ', 'e', 'p',
'i', 'c', ' ', 'i', 'n', ' ', 't', 'h', 'e', ' ', 'M', 'C', 'U', '.', ' ',
'H', 'e', ' ', 'd', 'o', 'e', 's', ' ', 's', 'o', 'm', 'e', ' ', 'p', 'r',
'e', 't', 't', 'y', ' ', 'e', 'p', 'i', 'c', ' ', 's', 'h', '*', 't', ' ',
'i', 'n', ' ', 't', 'h', 'i', 's', ' ', 'm', 'o', 'v', 'i', 'e', ' ', 'a',
'n', 'd', ' ', 'h', 'e', ' ', 'i', 's', ' ', 'd', 'e', 'f', 'i', 'n', 'i',
't', 'e', 'l', 'y', ' ', 'n', 'o', 't', ' ', 'u', 'n', 'd', 'e', 'r', '-',
'p', 'o', 'w', 'e', 'r', 'e', 'd', ' ', 'a', 'n', 'y', 'm', 'o', 'r', 'e',
'.', ' ', 'T', 'h', 'o', 'r', ' ', 'i', 'n', ' ', 'u', 'n', 'l', 'e', 'a',
's', 'h', 'e', 'd', ' ', 'i', 'n', ' ', 't', 'h', 'i', 's', ',', ' ', 'I',
' ', 'l', 'o', 'v', 'e', ' ', 't', 'h', 'a', 't', '.']
```

hazm

```
>>> from __future__ import unicode_literals
>>> from hazm import *

>>> normalizer = Normalizer()
>>> normalizer.normalize('اصلاح نویسه ها و استفاده از نیم‌فاصله پردازش را آسان مي كند')
'اصلاح نویسه‌ها و استفاده از نیم‌فاصله پردازش را آسان می‌کند'

>>> sent_tokenize('ما هم برای وصل کردن آمدیم! ولی برای پردازش، جدا بهتر نیست؟')
['ما هم برای وصل کردن آمدیم!', 'ولی برای پردازش، جدا بهتر نیست؟']
>>> word_tokenize('ولی برای پردازش، جدا بهتر نیست؟')
['ولی', 'برای', 'پردازش', '،', 'جدا', 'بهتر', 'نیست', '؟']
```

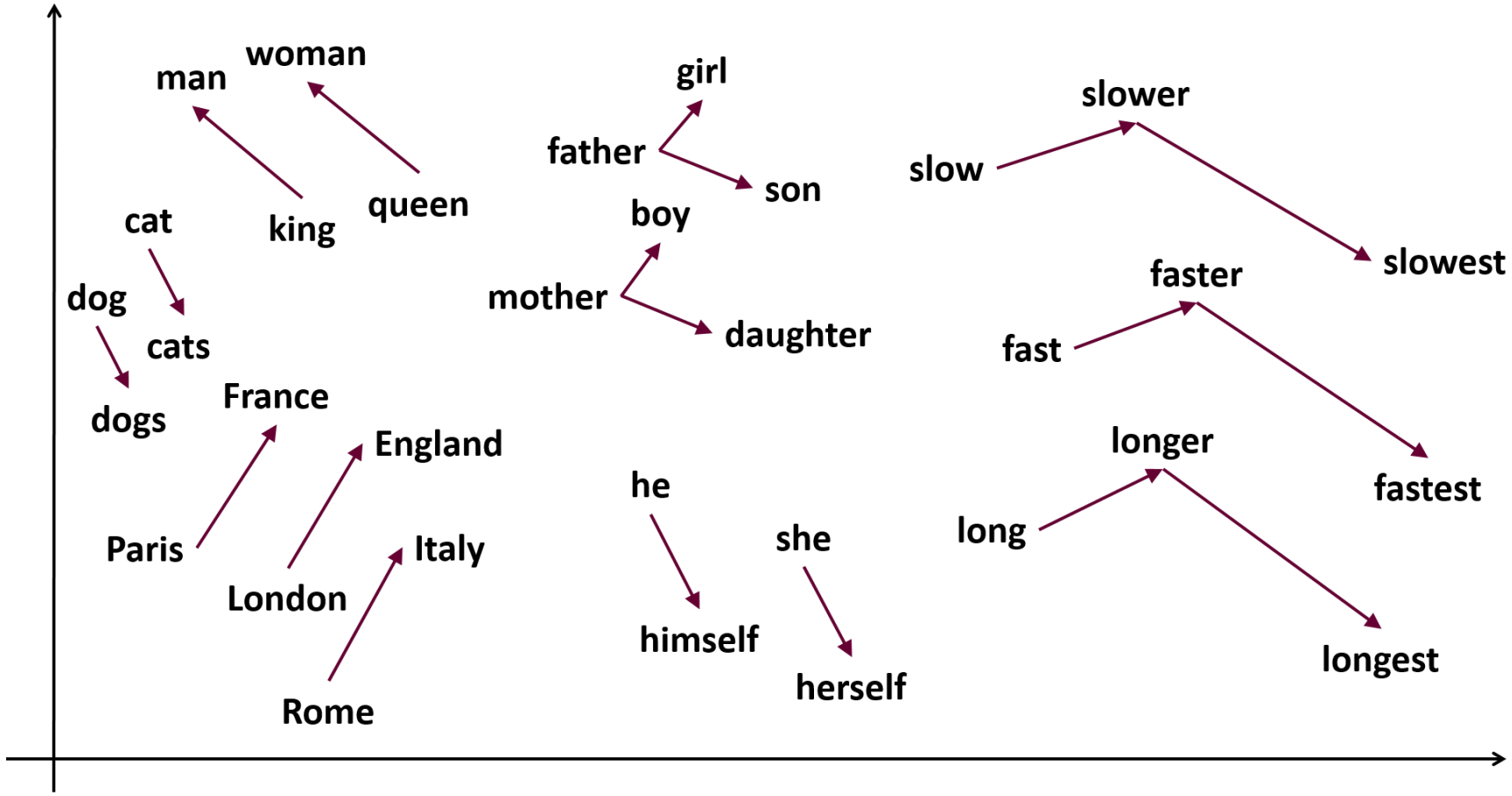
ngram

```
from nltk import ngrams

print(list(ngrams(thor_review.split(),2)))

#Results
[('the', 'action'), ('action', 'scenes'), ('scenes', 'were'), ('were',
'top'), ('top', 'notch'), ('notch', 'in'), ('in', 'this'), ('this',
'movie.'), ('movie.', 'Thor'), ('Thor', 'has'), ('has', 'never'), ('never',
'been'), ('been', 'this'), ('this', 'epic'), ('epic', 'in'), ('in', 'the'),
('the', 'MCU.'), ('MCU.', 'He'), ('He', 'does'), ('does', 'some'), ('some',
'pretty'), ('pretty', 'epic'), ('epic', 'sh*t'), ('sh*t', 'in'), ('in',
'this'), ('this', 'movie'), ('movie', 'and'), ('and', 'he'), ('he', 'is'),
('is', 'definitely'), ('definitely', 'not'), ('not', 'under-powered'),
('under-powered', 'anymore.'), ('anymore.', 'Thor'), ('Thor', 'in'), ('in',
'unleashed'), ('unleashed', 'in'), ('in', 'this,'), ('this,', 'I'), ('I',
'love'), ('love', 'that.')]
```

Vectorization(One-hot encoding)

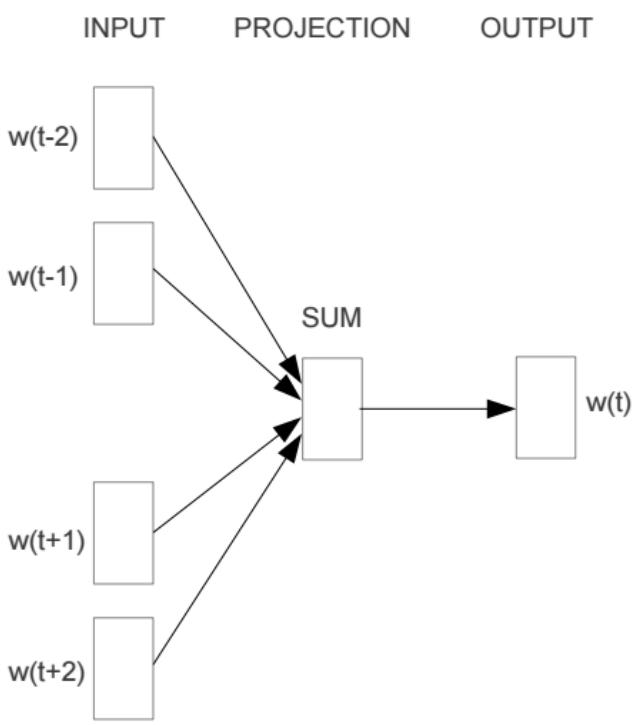
An	100000000
apple	010000000
a	001000000
day	000100000
keeps	000010000
doctor	000001000
away	000000100
said	000000010
the	000000001

Vectorization(word-embedding)

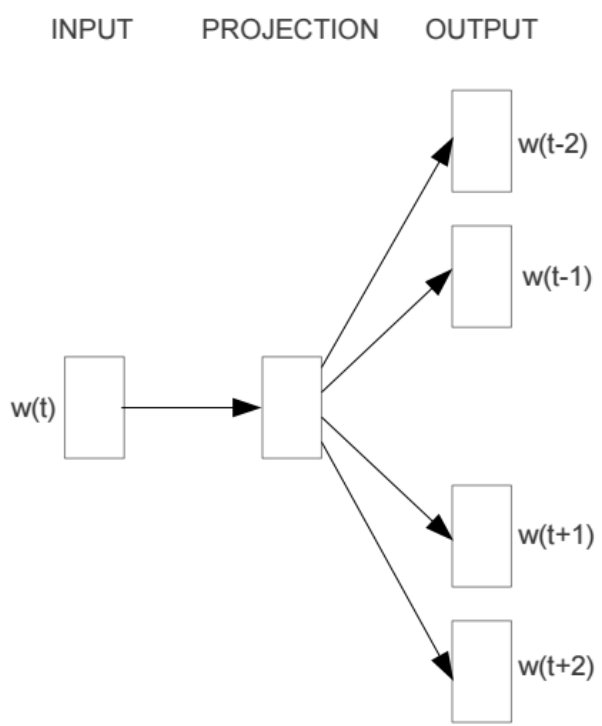


Vectorization(pretrain)

- Word2vec

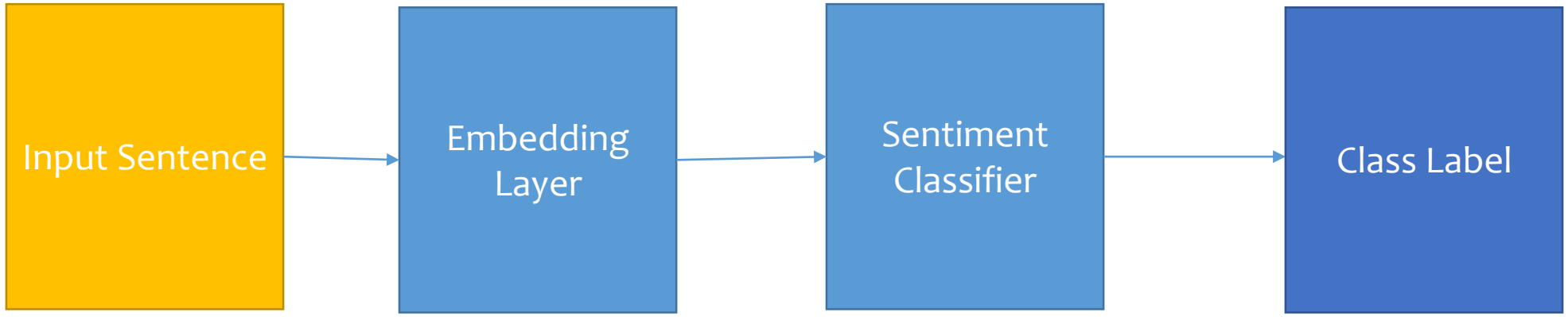


CBOW

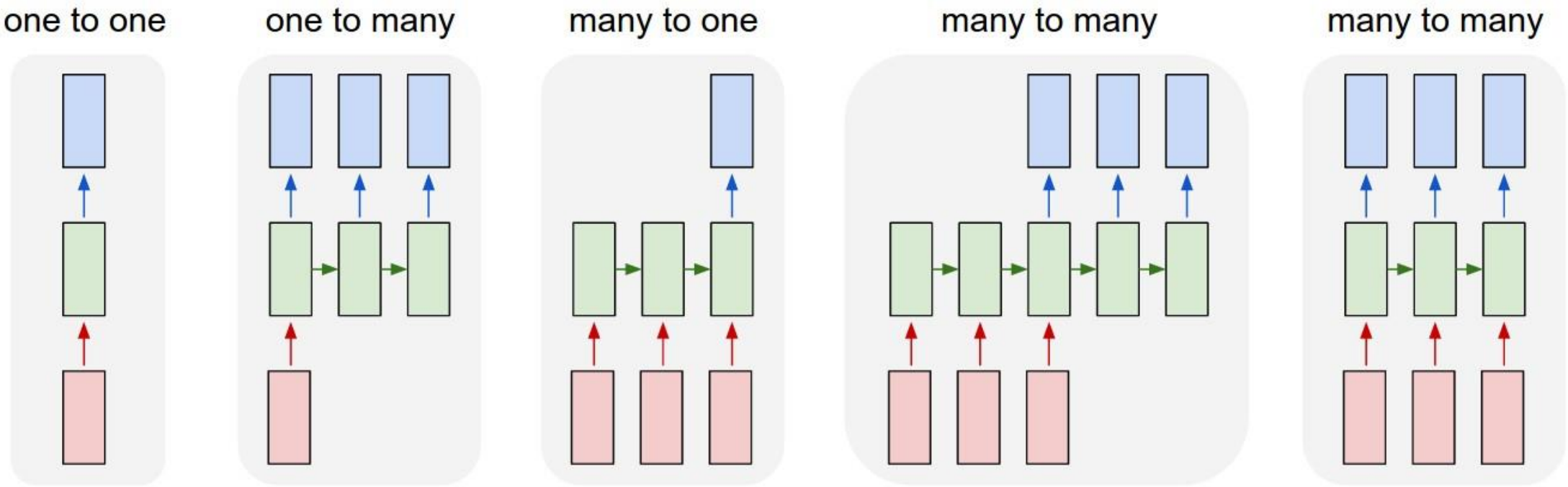


Skip-gram

Training word embedding by building a sentiment classifier

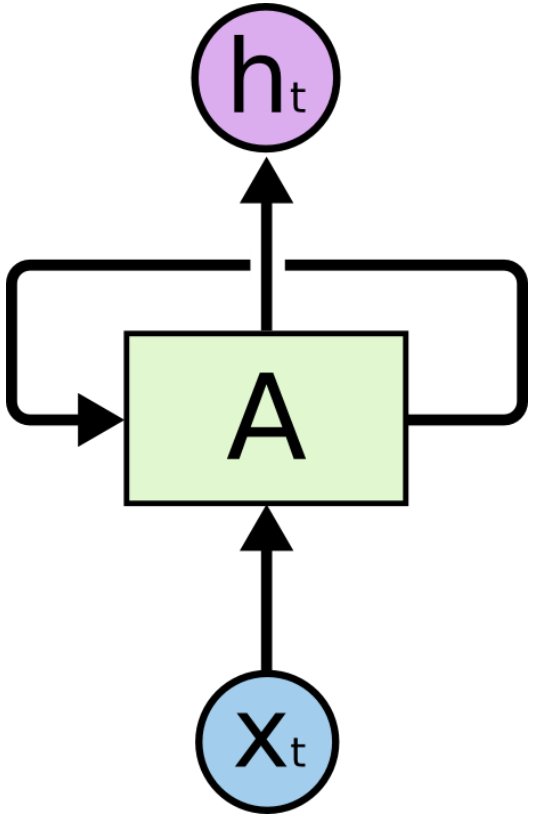


Recurrent Neural Networks

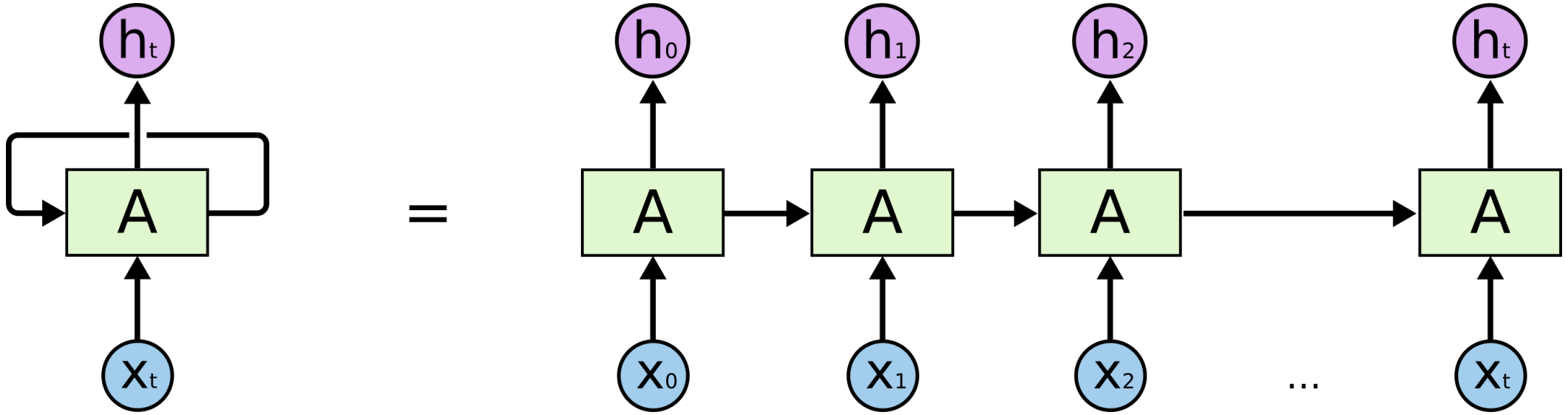


Each rectangle is a vector and arrows represent functions (e.g. matrix multiply). Input vectors are in red, output vectors are in blue and green vectors hold the RNN's state (more on this soon). From left to right: **(1)** Vanilla mode of processing without RNN, from fixed-sized input to fixed-sized output (e.g. image classification). **(2)** Sequence output (e.g. image captioning takes an image and outputs a sentence of words). **(3)** Sequence input (e.g. sentiment analysis where a given sentence is classified as expressing positive or negative sentiment). **(4)** Sequence input and sequence output (e.g. Machine Translation: an RNN reads a sentence in English and then outputs a sentence in French). **(5)** Synced sequence input and output (e.g. video classification where we wish to label each frame of the video). Notice that in every case are no pre-specified constraints on the lengths sequences because the recurrent transformation (green) is fixed and can be applied as many times as we like.

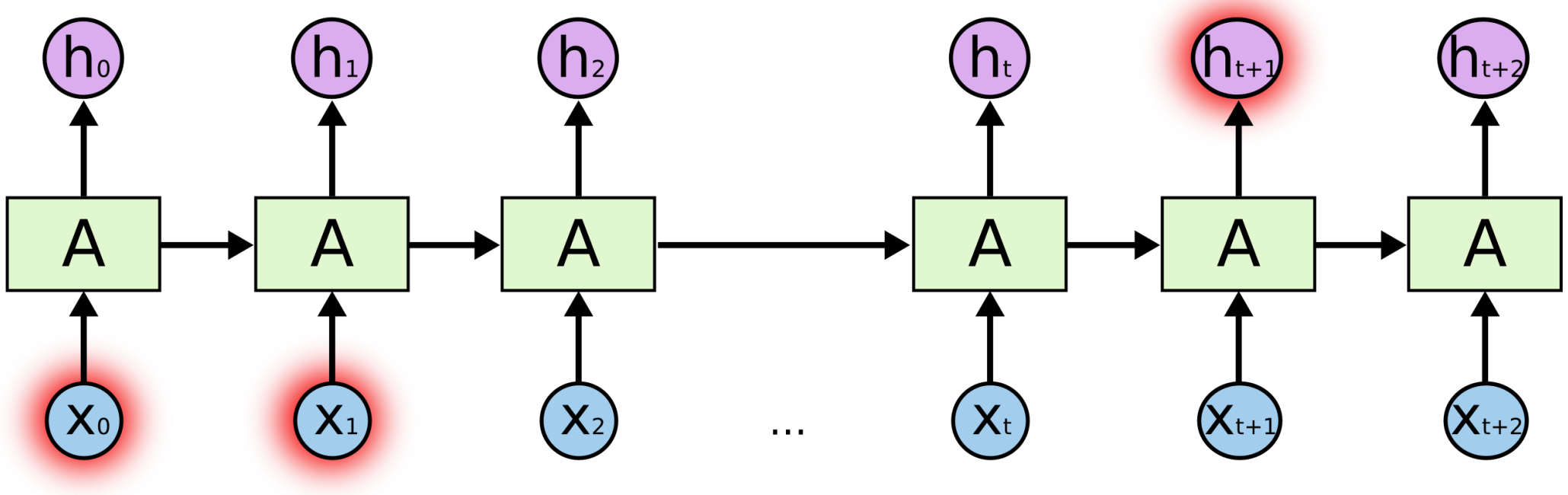
Unfolding RNN



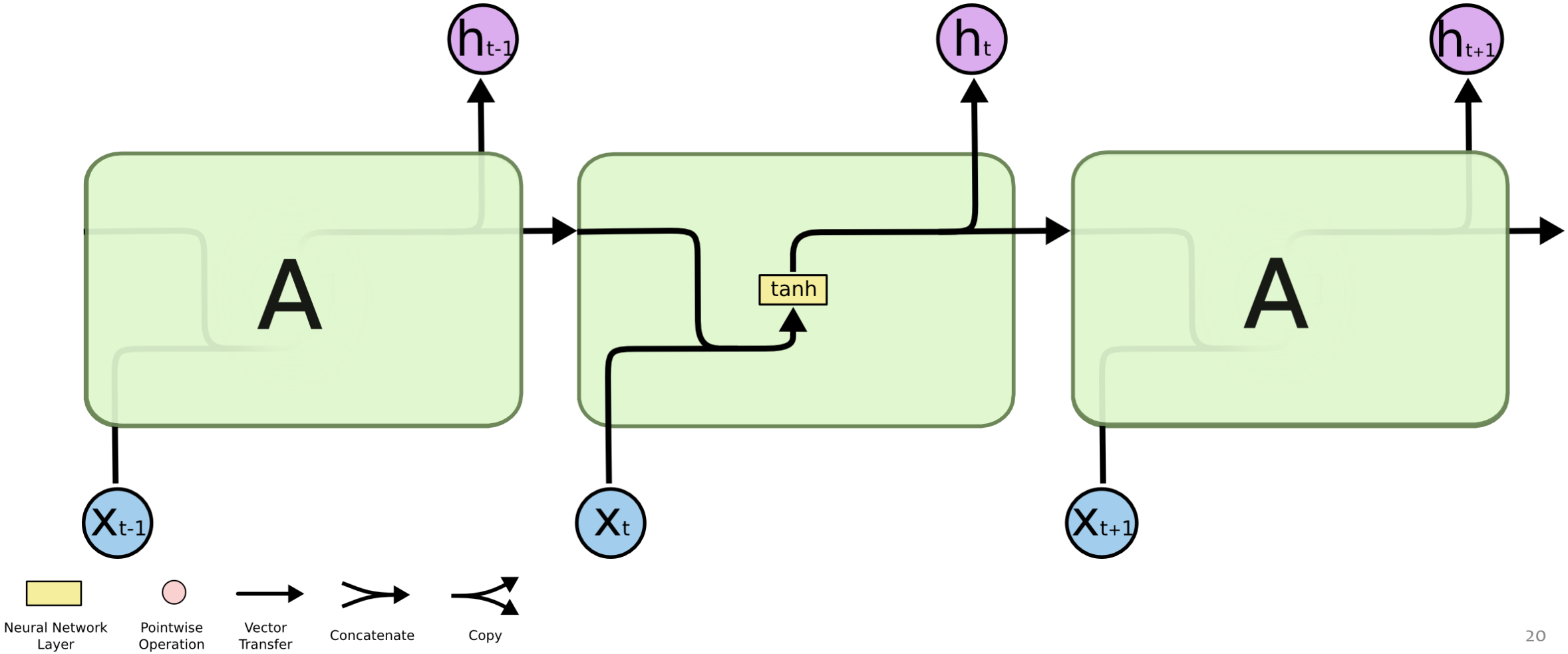
Unfolding RNN



Long-Term Dependency



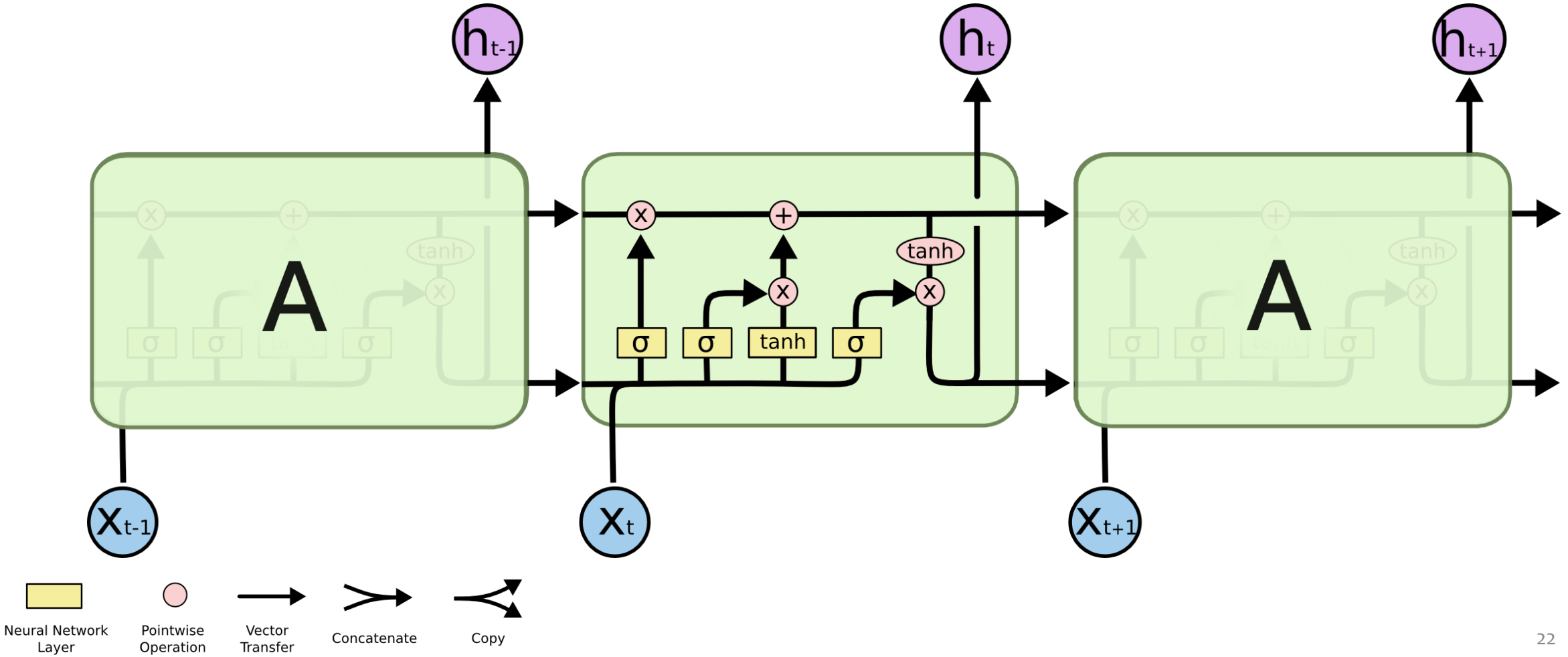
Vanilla RNN


h_{t-1}
h_t
h_{t+1}
A
tanh
A
X_{t-1}
X_t
X_{t+1}
Neural Network Layer
Pointwise Operation
Vector Transfer
Concatenate
Copy

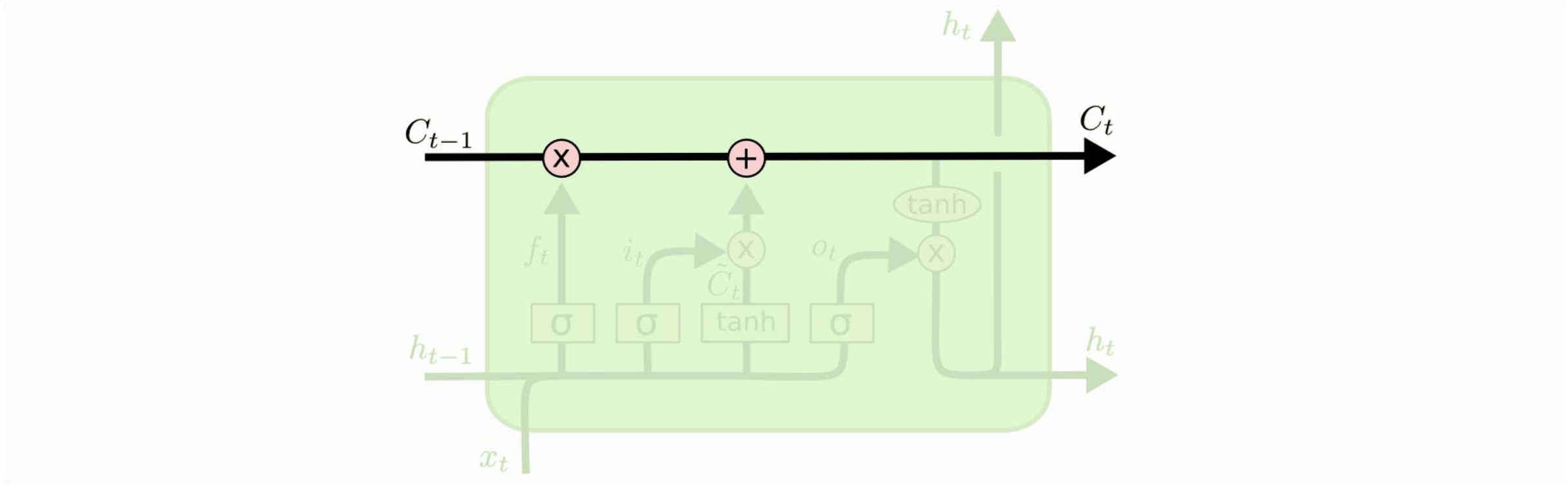
Vanilla RNN

```
import torch.nn as nn
from torch.autograd import Variable

class RNN(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super(RNN, self).__init__()
        self.hidden_size = hidden_size
        self.i2h = nn.Linear(input_size + hidden_size, hidden_size)
        self.i2o = nn.Linear(input_size + hidden_size, output_size)
        self.softmax = nn.LogSoftmax(dim=1)

    def forward(self, input, hidden):
        combined = torch.cat((input, hidden), 1)
        hidden = self.i2h(combined)
        output = self.i2o(combined)
        output = self.softmax(output)
        return output, hidden

    def initHidden(self):
        return Variable(torch.zeros(1, self.hidden_size))
```

LSTM (Long-Short Term Memory)



LSTM (Cell State)



Neural Network Layer

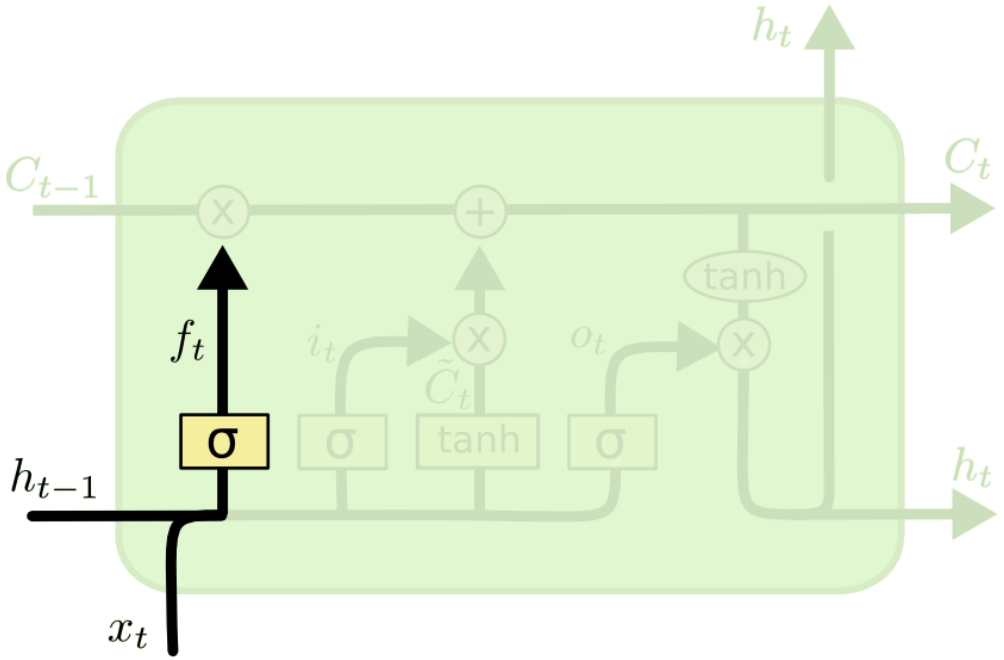
Pointwise Operation

Vector Transfer

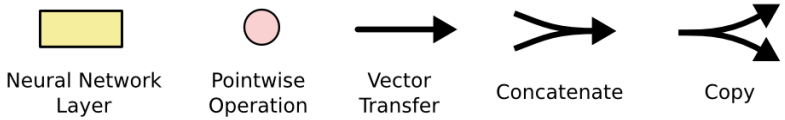
Concatenate

Copy

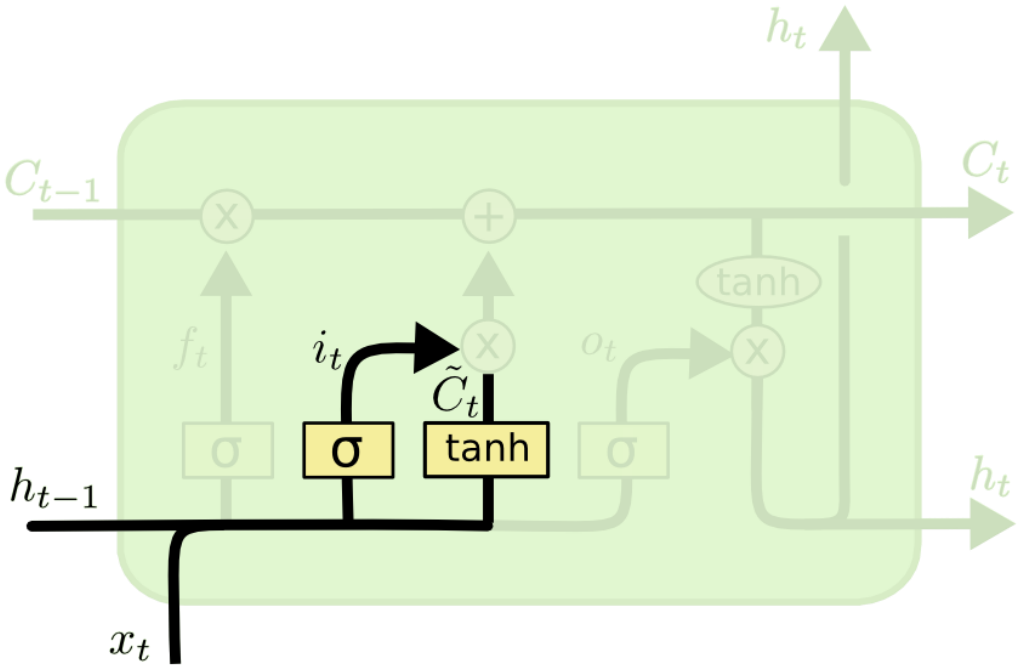
LSTM (forget gate layer)



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$



LSTM (input gate layer)



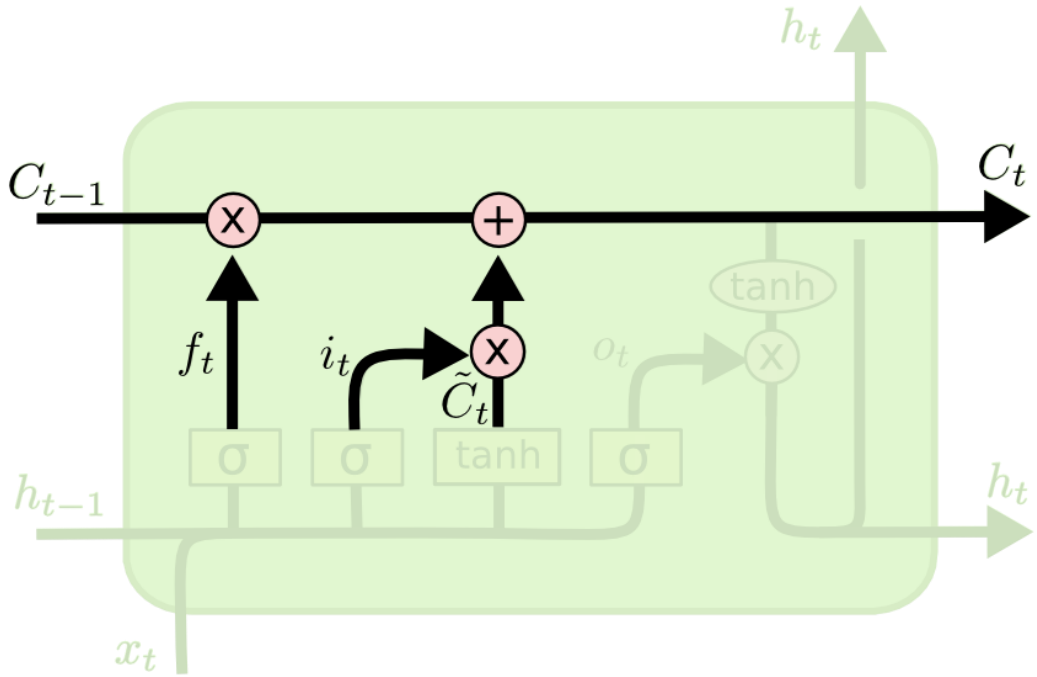
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

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

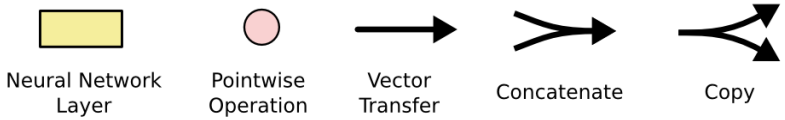
Neural Network Layer | Pointwise Operation | Vector Transfer | Concatenate | Copy

a tanh layer creates a vector of new candidate values

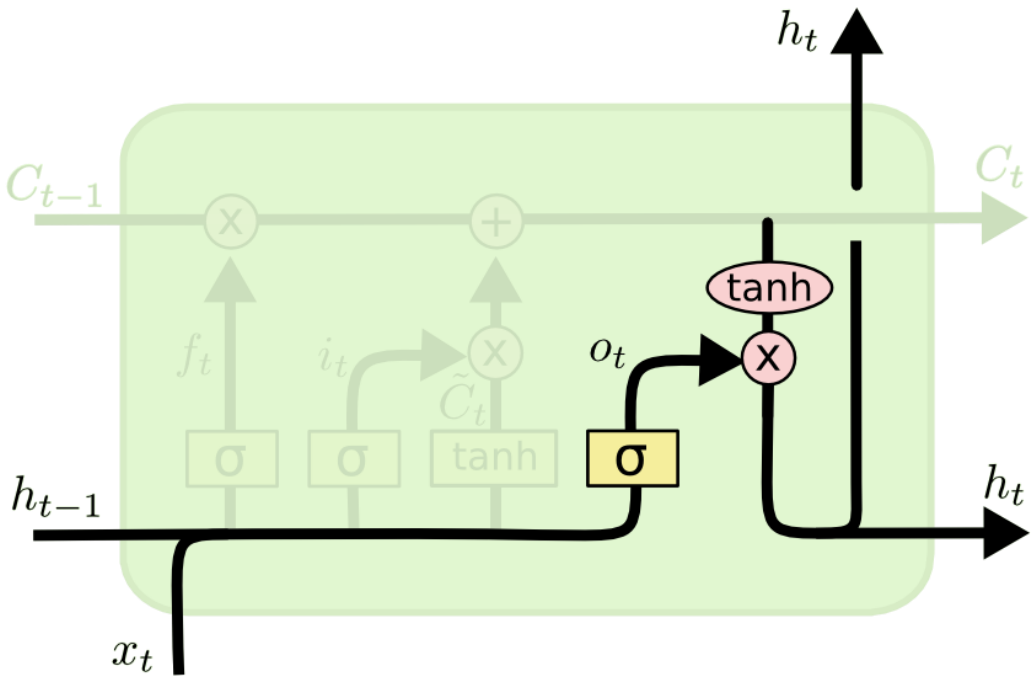
Cell State Final Value



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$



Output Value



$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$