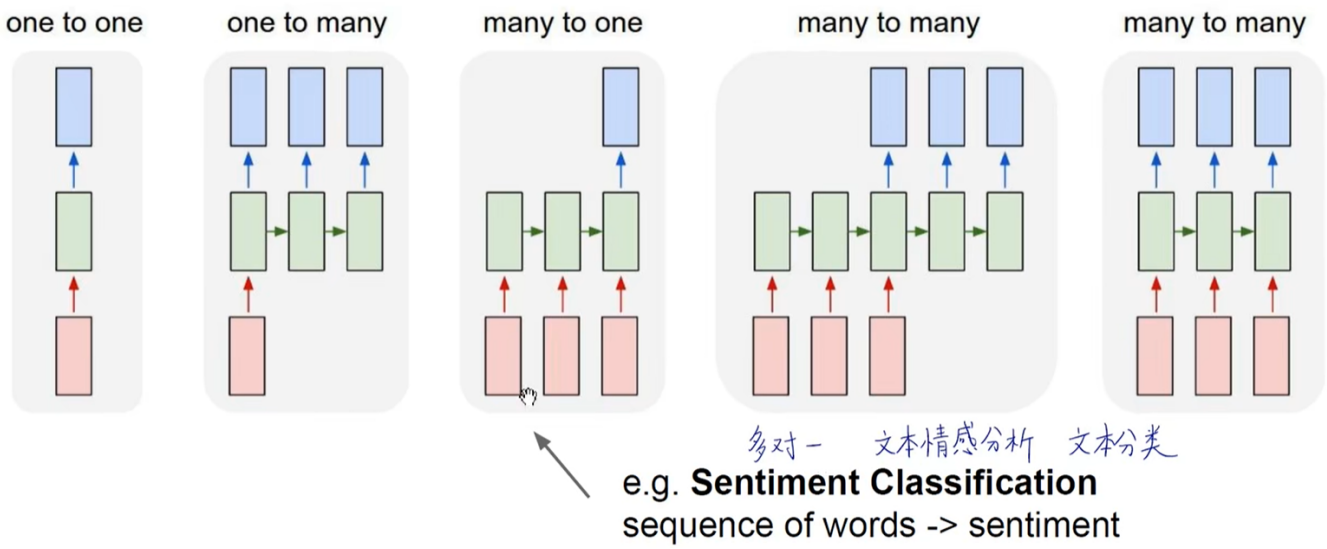


一对多：图像描述

多对一：文本情感分析 文本分类

多对多不对齐：机器翻译

多对多对齐：以帧为粒度的视频分类

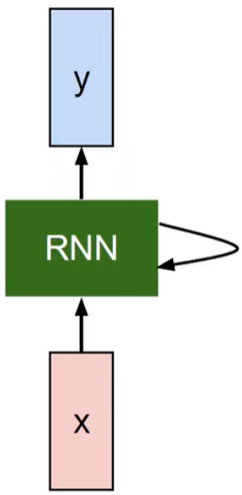
We can process a sequence of vectors **x** by applying a **recurrence formula** at every time step:

本时刻的隐含状态 上一时刻的隐含状态 本时刻的输入

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

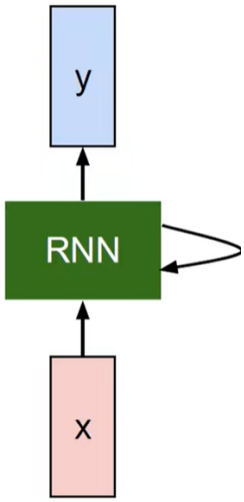
new state　old state　input vector at some time step

some function with parameters W



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

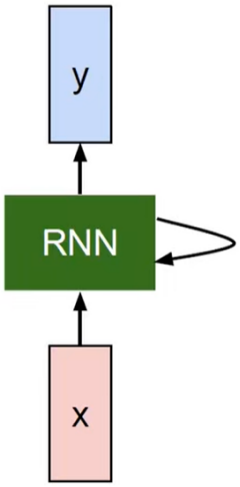
$$h_t = f_W(h_{t-1}, x_t)$$



沿时间维度 权值共享 每个时刻的 f_W 都相同

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

CNN：沿空间维度权值共享
RNN：沿时间维度权值共享



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

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

$$y_t = W_{hy} h_t$$

三套权重 沿时间维度权值共享

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

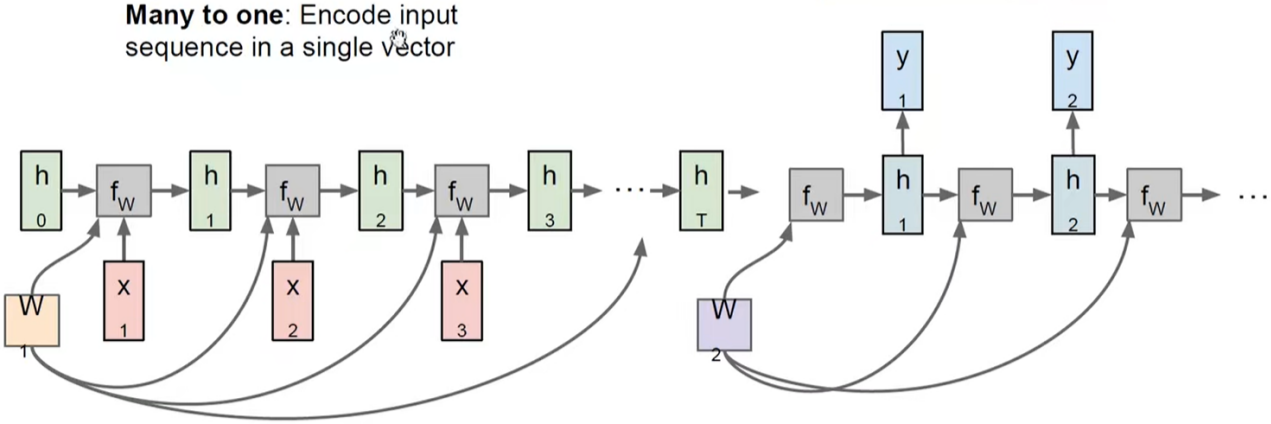
one-to-many

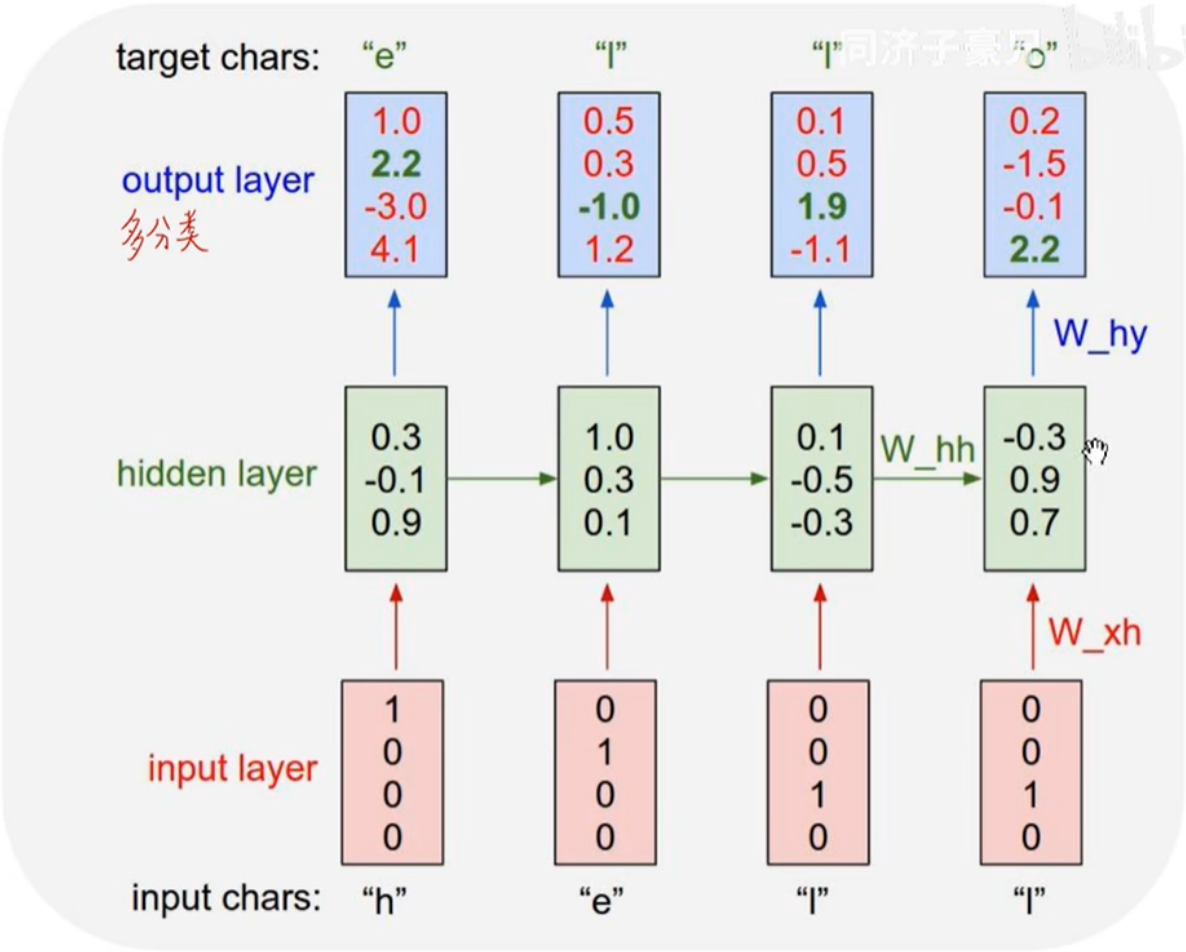
多对一 编码器 Encoder

Many to one: Encode input sequence in a single vector

一对多 解码器 Decoder

One to many: Produce output sequence from single input vector




target chars:
“e”
“l”
“l”
“o”
output layer
多分类
1.0
2.2
-3.0
4.1
0.5
0.3
-1.0
1.2
0.1
0.5
1.9
-1.1
0.2
-1.5
-0.1
2.2
W_hy
hidden layer
0.3
-0.1
0.9
1.0
0.3
0.1
0.1
-0.5
-0.3
W_hh
-0.3
0.9
0.7
W_xh
input layer
1
0
0
0
0
1
0
0
0
0
1
0
0
0
1
0
input chars:
“h”
“e”
“l”
“l”

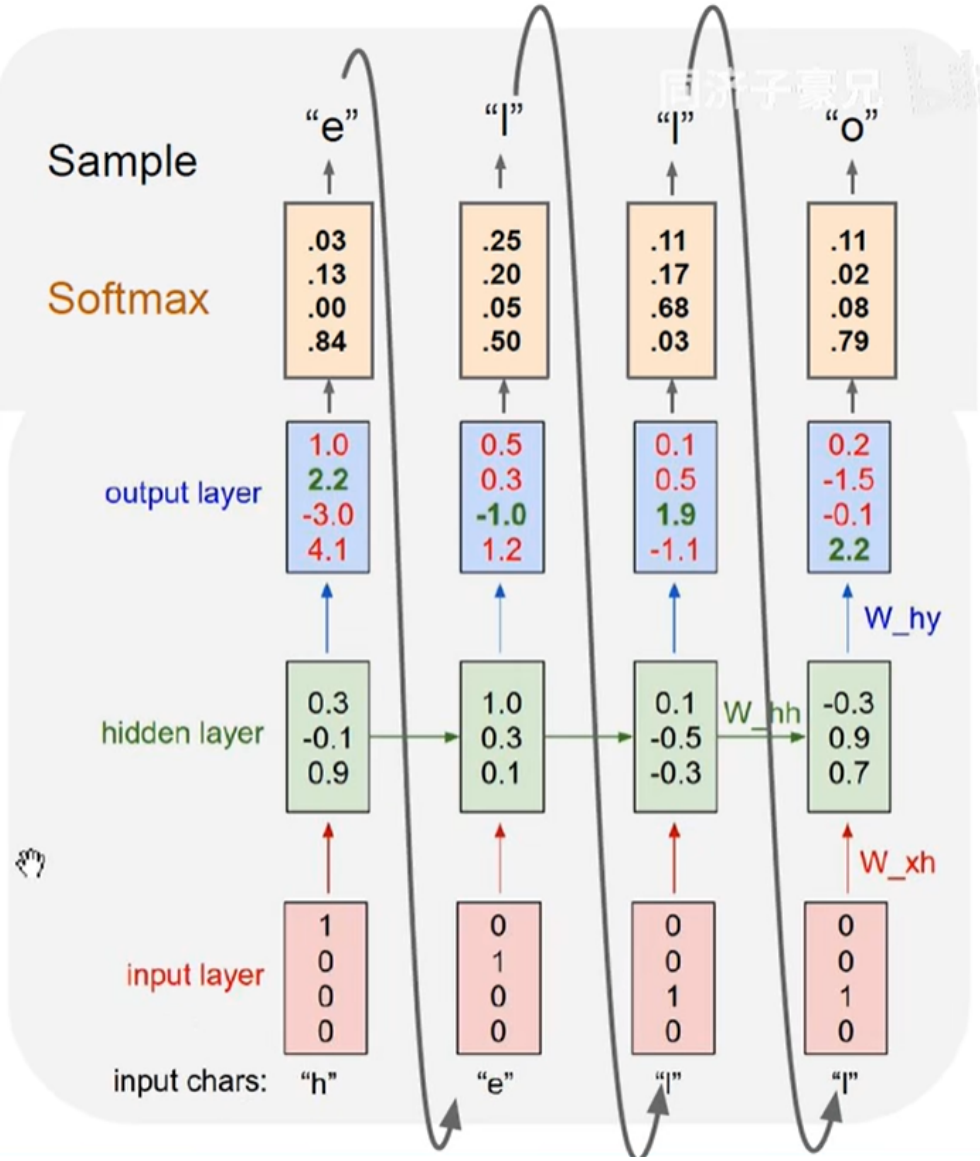
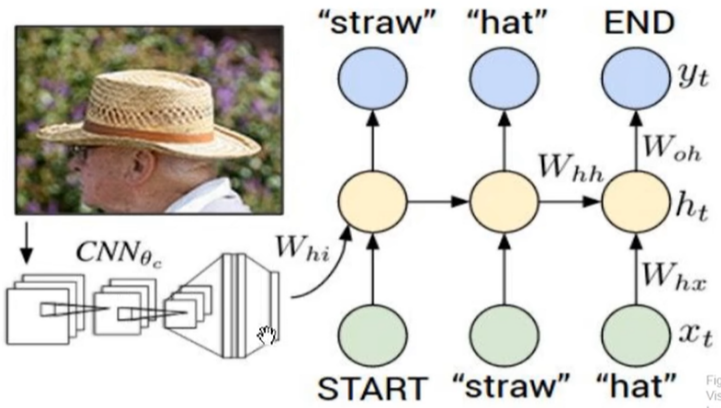


Image Captioning



Explain Images with Multimodal Recurrent Neural Networks, Mao et al.
Deep Visual-Semantic Alignments for Generating Image Descriptions, Karpathy and Fei-Fei
Show and Tell: A Neural Image Caption Generator, Vinyals et al.
Long-term Recurrent Convolutional Networks for Visual Recognition and Description, Donahue et al.
Learning a Recurrent Visual Representation for Image Caption Generation, Chen and Zitnick

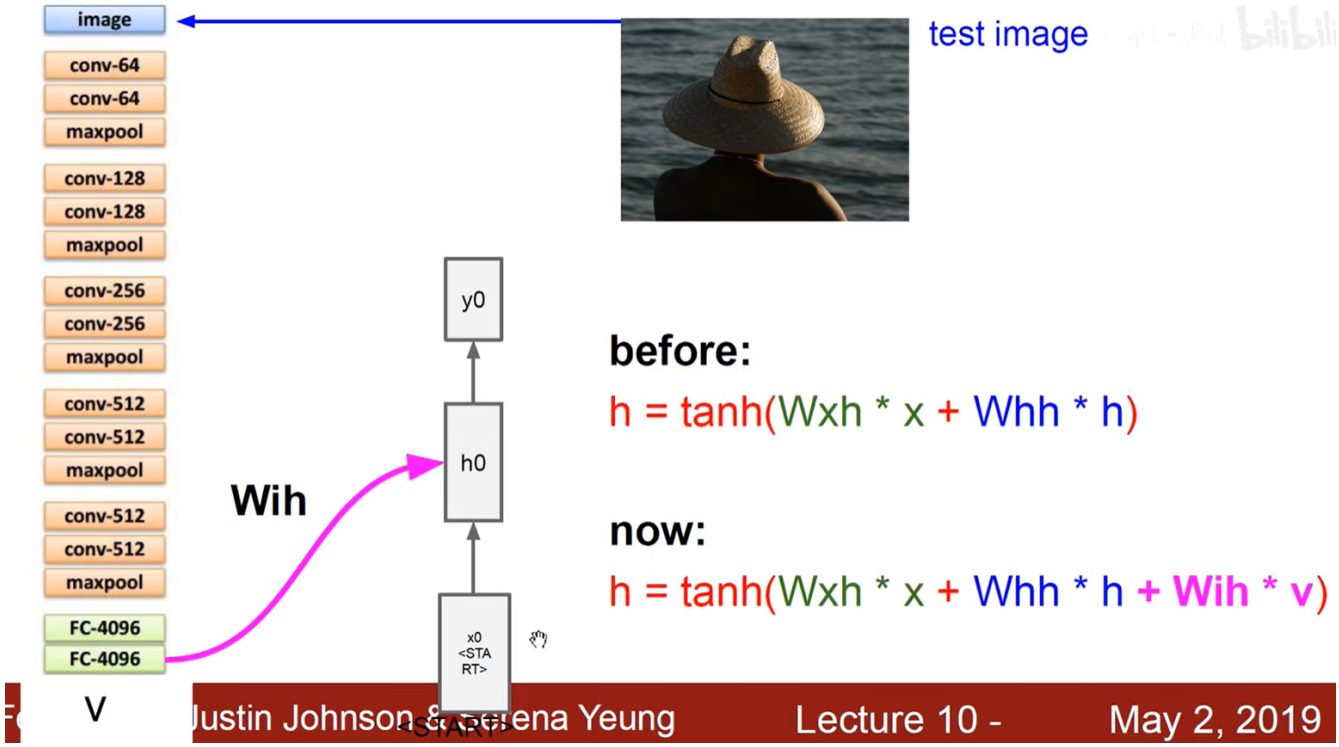


image
conv-64
conv-64
maxpool
conv-128
conv-128
maxpool
conv-256
conv-256
maxpool
conv-512
conv-512
maxpool
conv-512
conv-512
maxpool
FC-4096
FC-4096
V
Wih
y0
h0
x0
<STA
RT>
<START>
test image
before:
h = tanh(Wxh * x + Whh * h)
now:
h = tanh(Wxh * x + Whh * h + Wih * v)

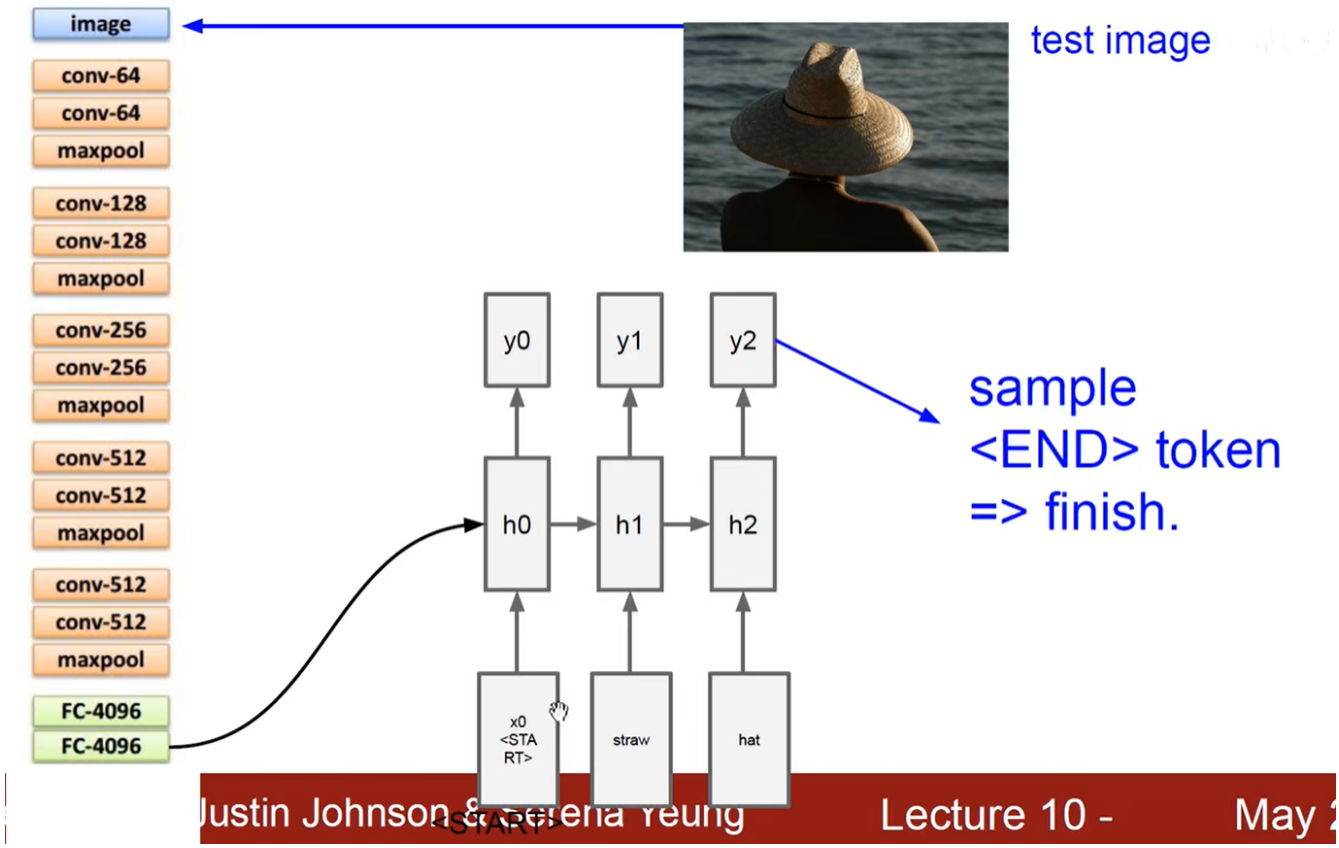
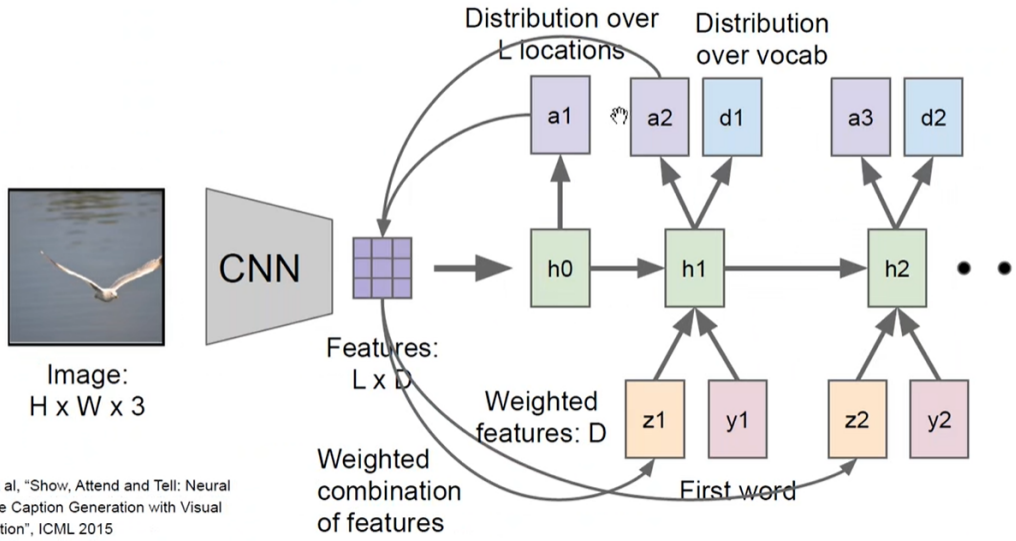


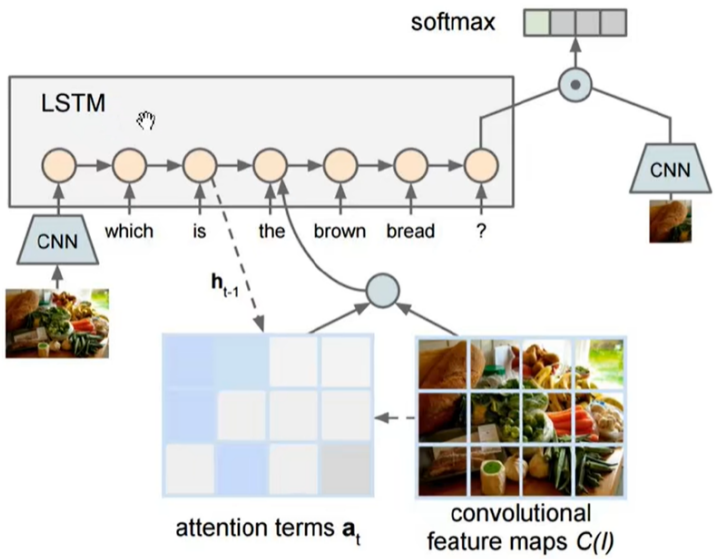
image
conv-64
conv-64
maxpool
conv-128
conv-128
maxpool
conv-256
conv-256
maxpool
conv-512
conv-512
maxpool
conv-512
conv-512
maxpool
FC-4096
FC-4096
y0
y1
y2
h0
h1
h2
x0
<STA
RT>
straw
hat
<START>
test image
sample
<END> token
=> finish.

Image Captioning with Attention



Xu et al, "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

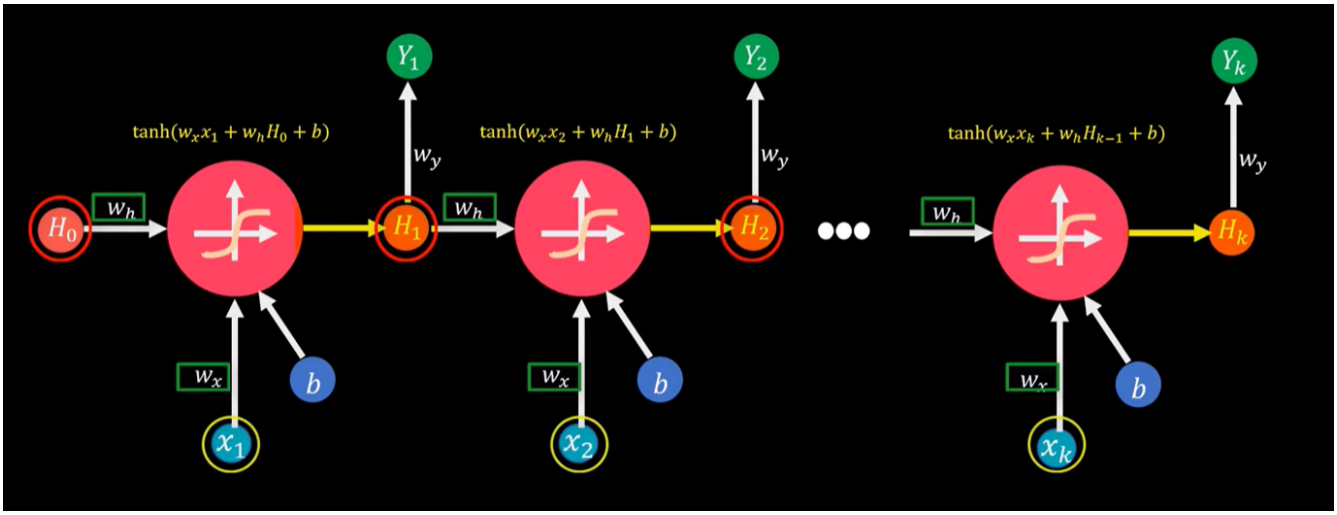
Visual Question Answering: RNNs with Attention



u et al, "Visual 7W: Grounded Question Answering in Images", CVPR 2016
ures from Zhu et al, copyright IEEE 2016. Reproduced for educational purposes.

What kind of animal is in the photo?
A **cat**.

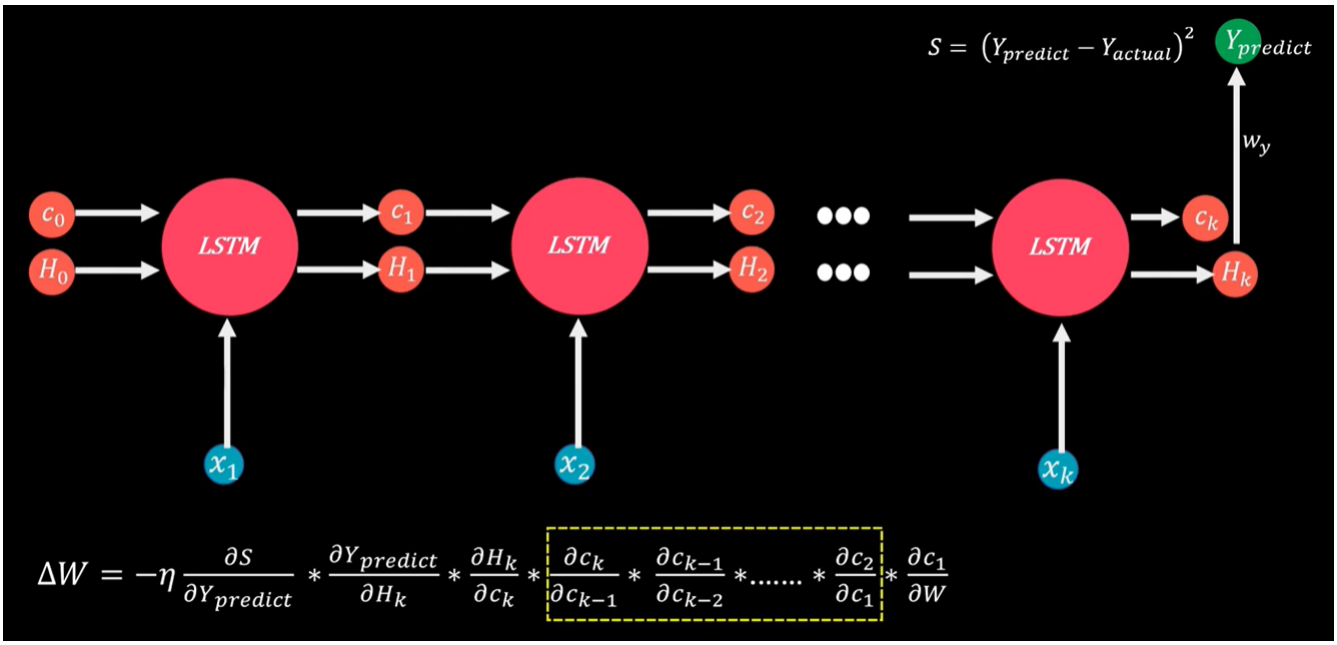
Why is the person holding a knife?
To cut the **cake** with.

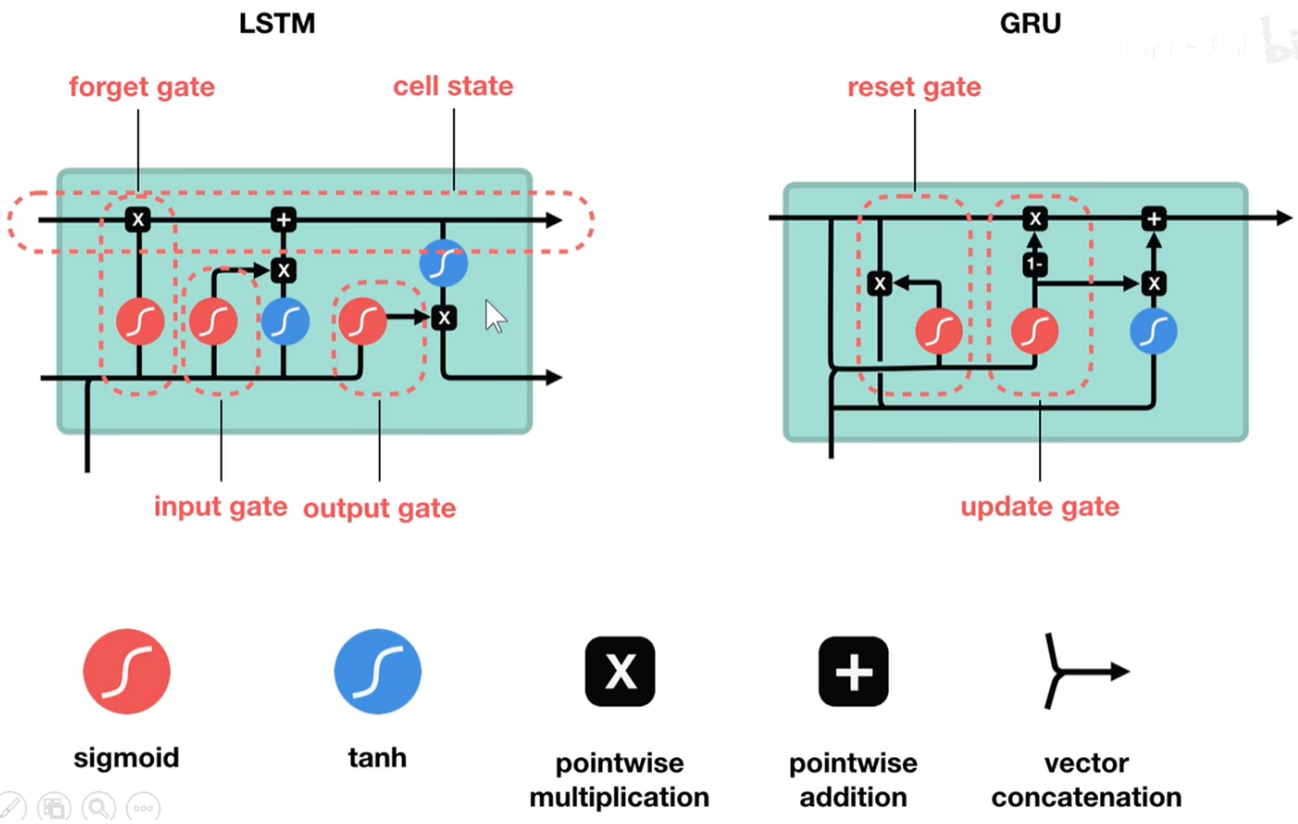


LSTM

遗忘门 输入门 输出门

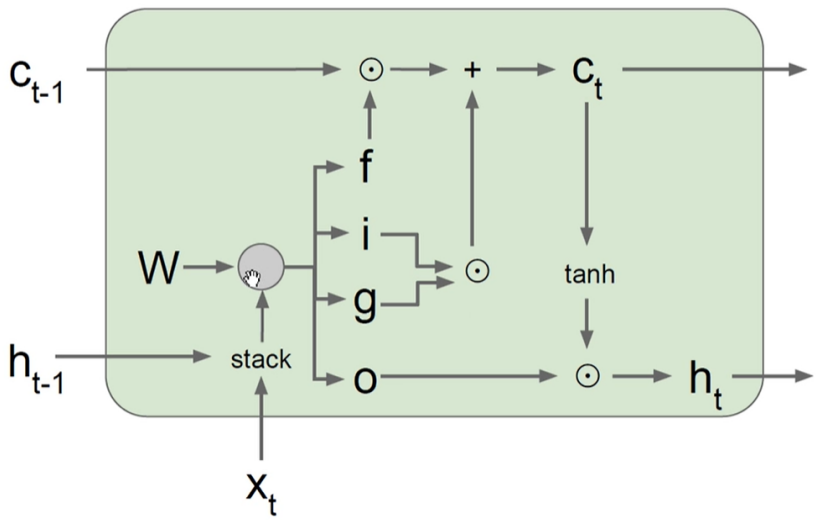
缓解梯度消失与爆炸





Long Short Term Memory (LSTM)

[Hochreiter et al., 1997]



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

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

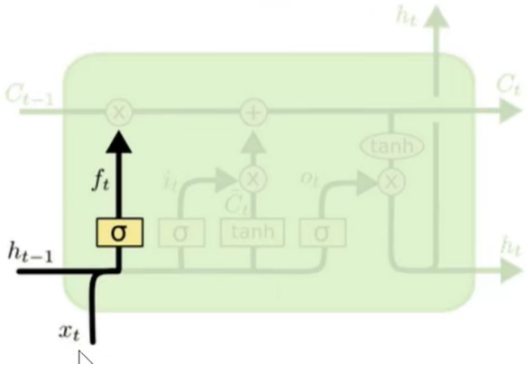
$$h_t = o \odot \tanh(c_t)$$

LSTM算法详解

第一步 选择被遗忘信息

① 通过“遗忘门层”实现

② 上一个时间的状态变量需要通过一个sigmoid激活函数。这个函数根据输入的当前数据和上一期状态，输出一个0到1之间的数值与Ct相乘



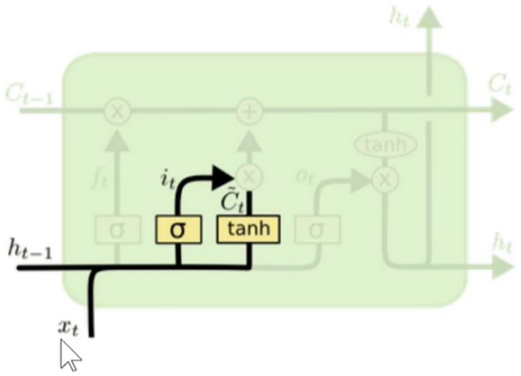
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)$$

LSTM算法详解

第二步 选择被保留信息

① “输入门层”： 使用sigmoid函数决定哪些信息要更新

② 使用tanh函数来生成更新Ct的信息



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

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

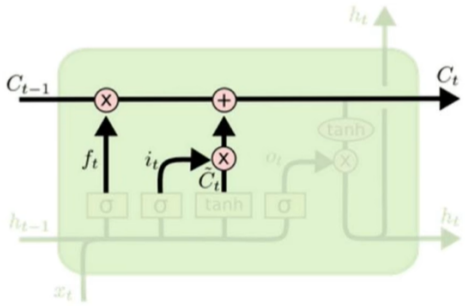
LSTM算法详解

第三步 更新cell中信息

① “遗忘门”层输出的信息 × 上一期的cell信息Ct-1，确定保留多少上一期cell信息的占比

② 使用tanh函数来生成更新Ct的信息

③ 在保留的状态信息里，添加第二步决定要保留的信息



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

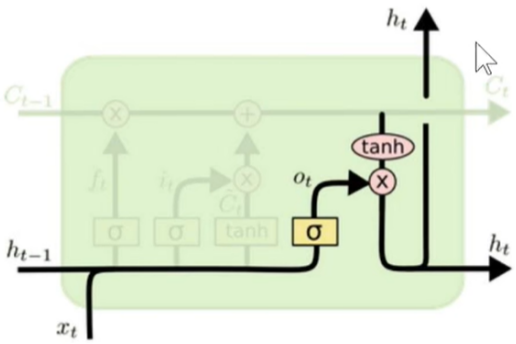
LSTM算法详解

第四步 决定输出

① 使用一个sigmoid函数决定哪些状态信息需要输出

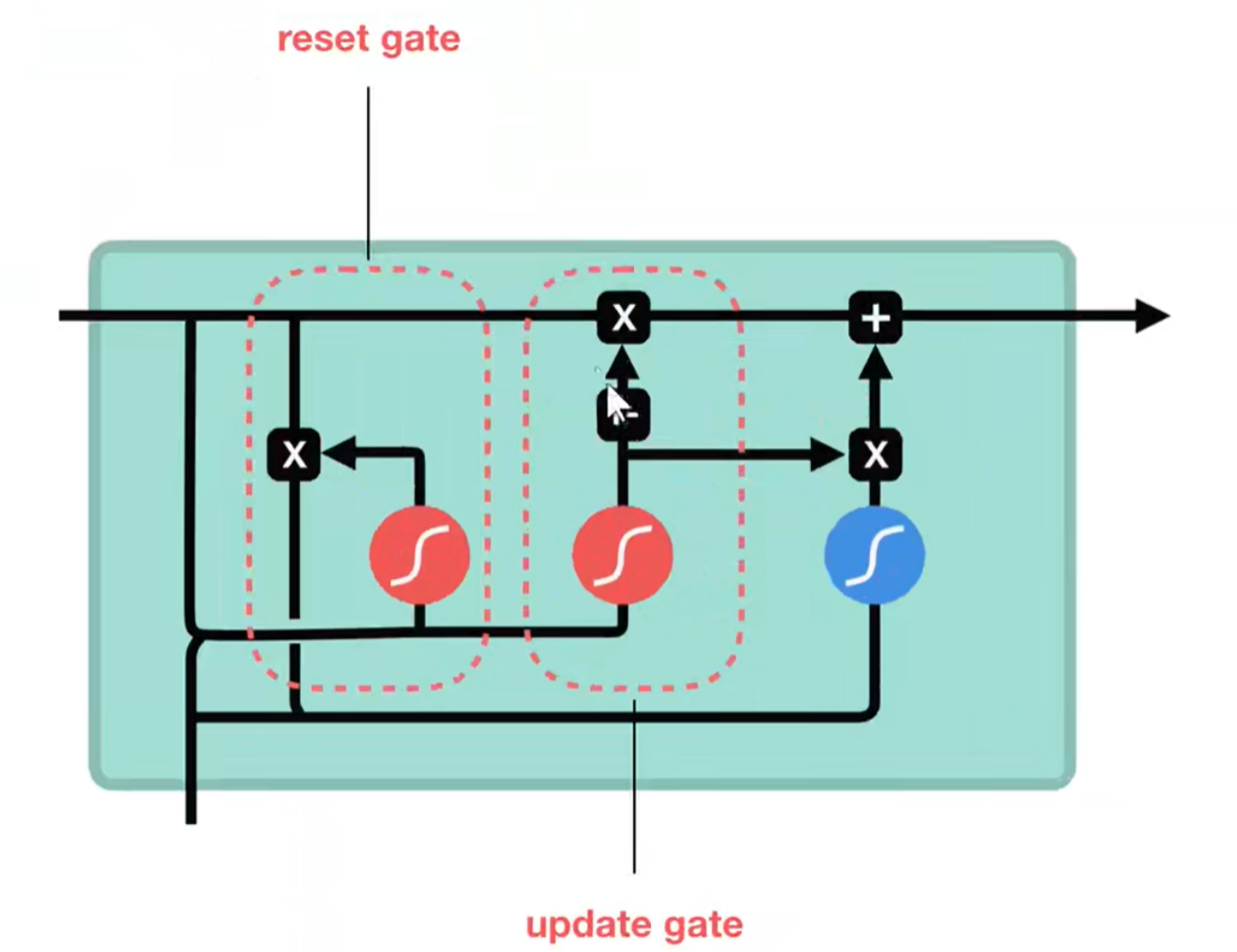
② 将状态信息通过tanh函数压缩到（-1， 1）之间

③ 乘以上一步得到的决策，获得需要输出的信息



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

GRU



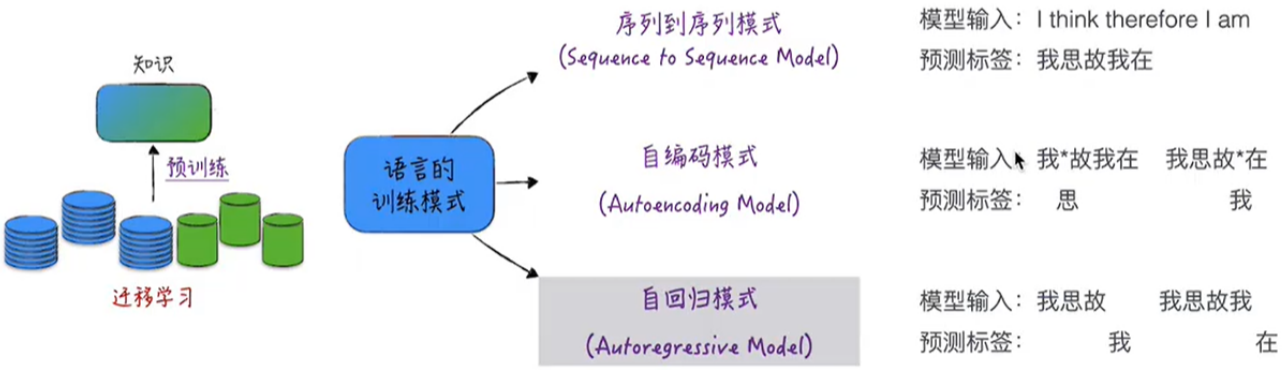
词的向量表示

one-hot编码

为了能够让整个网络能够理解我们的输入（英文/中文等），需要将词进行用向量表示。

- 建立一个包含所有序列词的词典包含（**开始和标志的两个特殊词，以及没有出现过的词用等**），每个词在词典里面有一个唯一的编号。
- 任意一个词都可以用一个N维的one-hot向量来表示。其中，N是词典中包含的词的个数

训练模式：如何使用数据



双流网络（Two-Stream Networks）和I3D网络（Inflated 3D ConvNets）都是深度学习领域中用于视频理解任务的网络架构。

双流网络（Two-Stream Networks）

双流网络是一种用于视频分析的深度学习架构，它结合了空间和时间信息来提高对视频内容的理解。这种网络通常由两个分支组成：

1. **空间流（Spatial Stream）**：这个分支专注于处理单帧图像，使用传统的2D卷积神经网络（如AlexNet、VGG等）来提取空间特征。空间流可以捕捉到视频中的静态特征，如物体的形状和纹理。
2. **时间流（Temporal Stream）**：这个分支处理视频序列，通常使用3D卷积神经网络来提取时间特征。时间流可以捕捉到视频中的运动信息，如物体的运动轨迹和速度。

双流网络的优势在于它能够同时利用空间和时间信息，这使得网络在处理复杂的视频任务时更加有效，如动作识别、视频分类等。

I3D网络（Inflated 3D ConvNets）

I3D网络是一种3D卷积神经网络的变体，它在3D卷积层中使用了膨胀卷积（Dilated Convolutions）。I3D网络的设计灵感来自于Inception网络架构，它通过在3D卷积层中使用不同比例的膨胀率来增加感受野，从而提高对视频内容的理解能力。

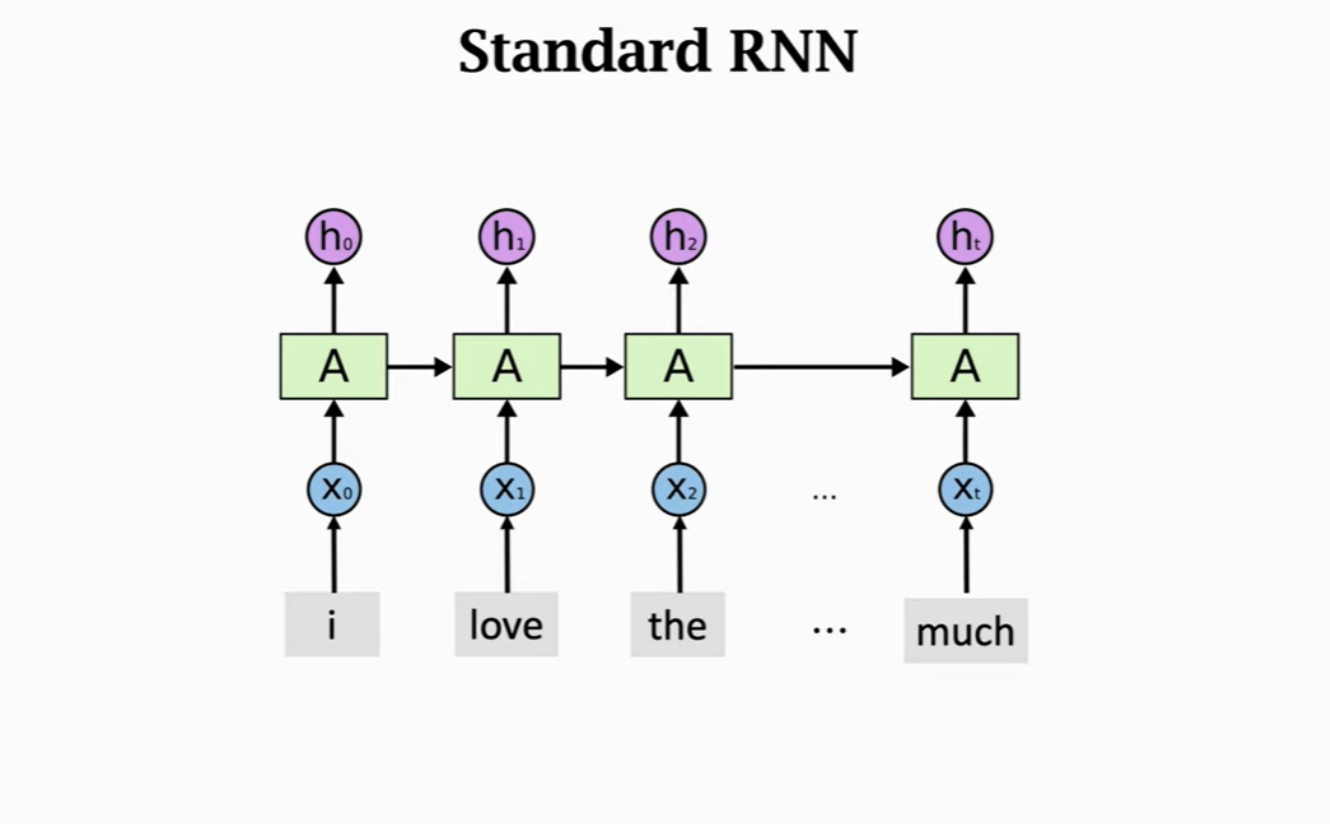
I3D网络的主要特点包括：

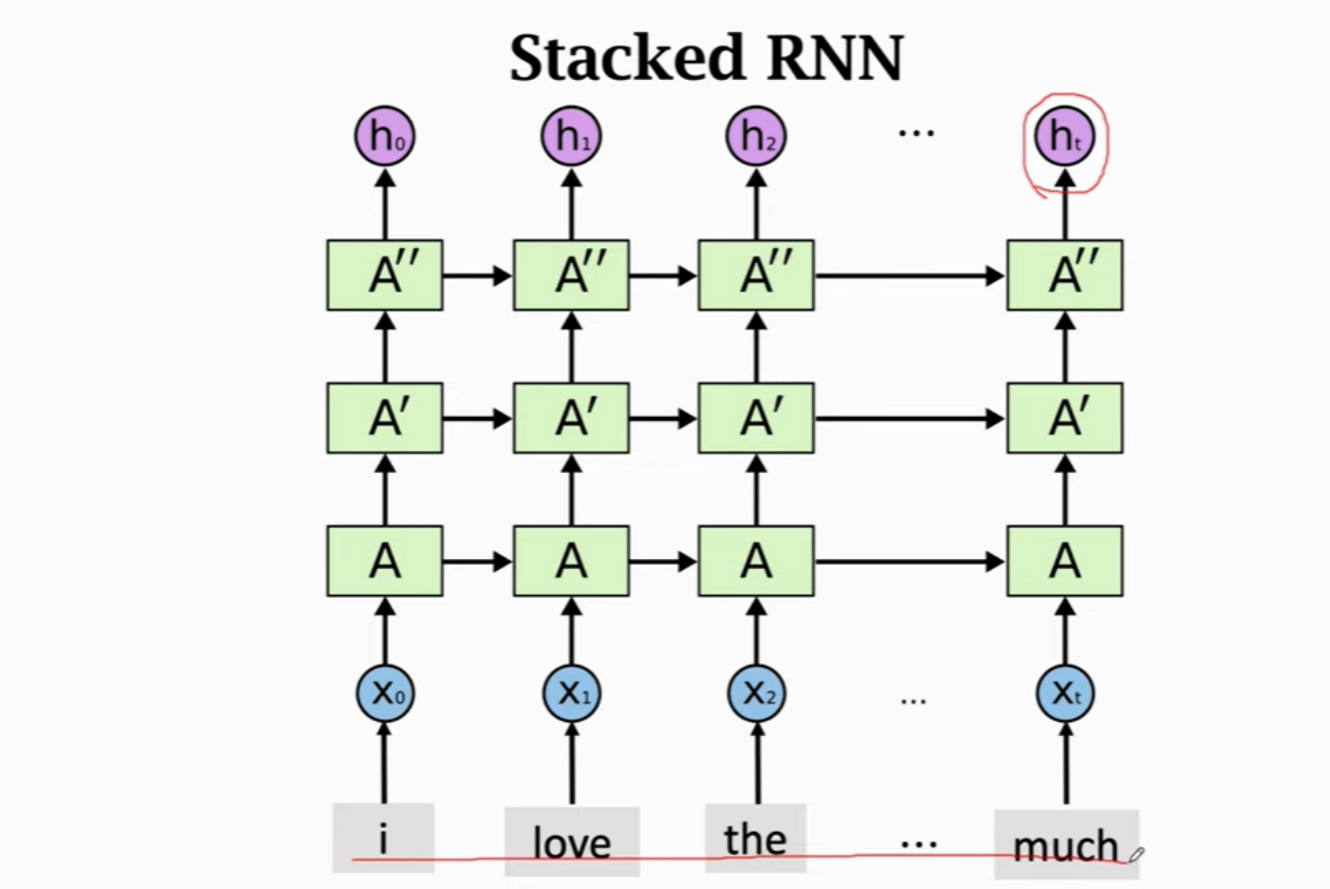
- **膨胀卷积**：通过增加卷积核的间距来扩大感受野，这有助于捕捉更广泛的时空特征。
- **多尺度特征融合**：I3D网络在不同层次上融合特征，这有助于捕捉不同尺度的特征，提高模型的泛化能力。
- **高效的计算**：尽管I3D网络使用了膨胀卷积，但其计算效率仍然相对较高，这使得它适用于大规模的视频分析任务。

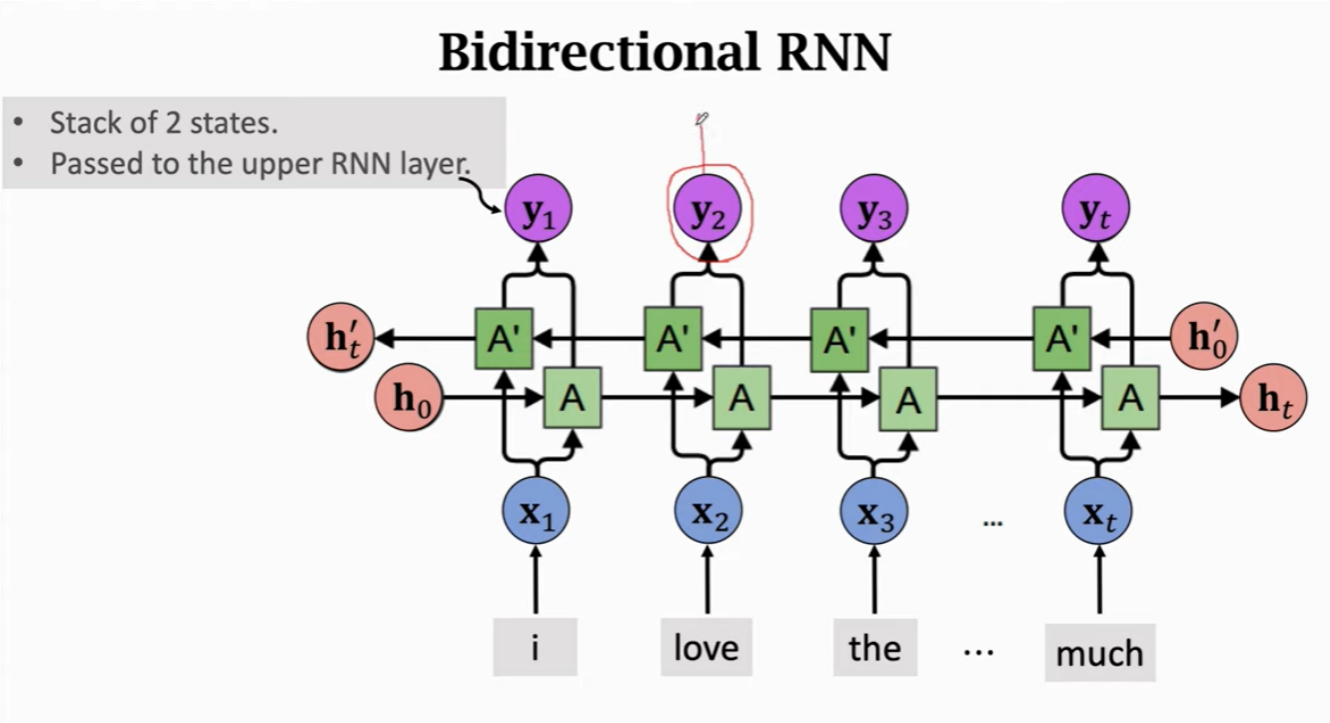
I3D网络在视频分类、动作识别等任务上表现出色，是当前视频理解领域的一个重要研究方向。

这两种网络架构都是为了更好地理解和分析视频内容，它们各有优势，可以根据具体的应用场景和需求进行选择。

多层RNN、双向RNN





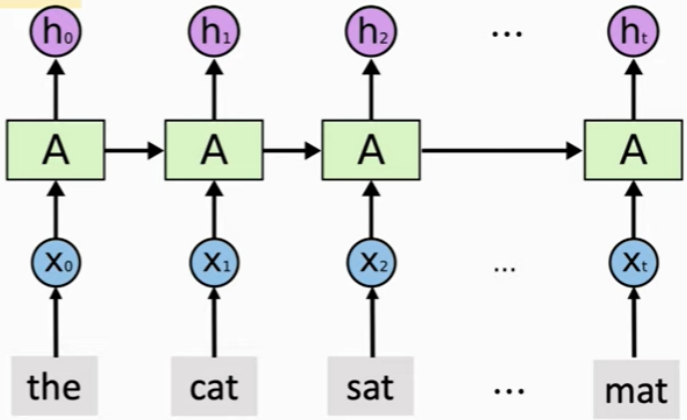


Embedding 预训练

Pretrain the Embedding Layer

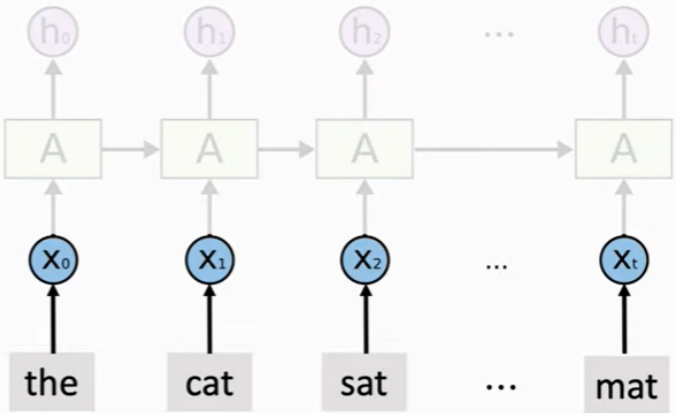
Step 1: Train a model on large dataset.

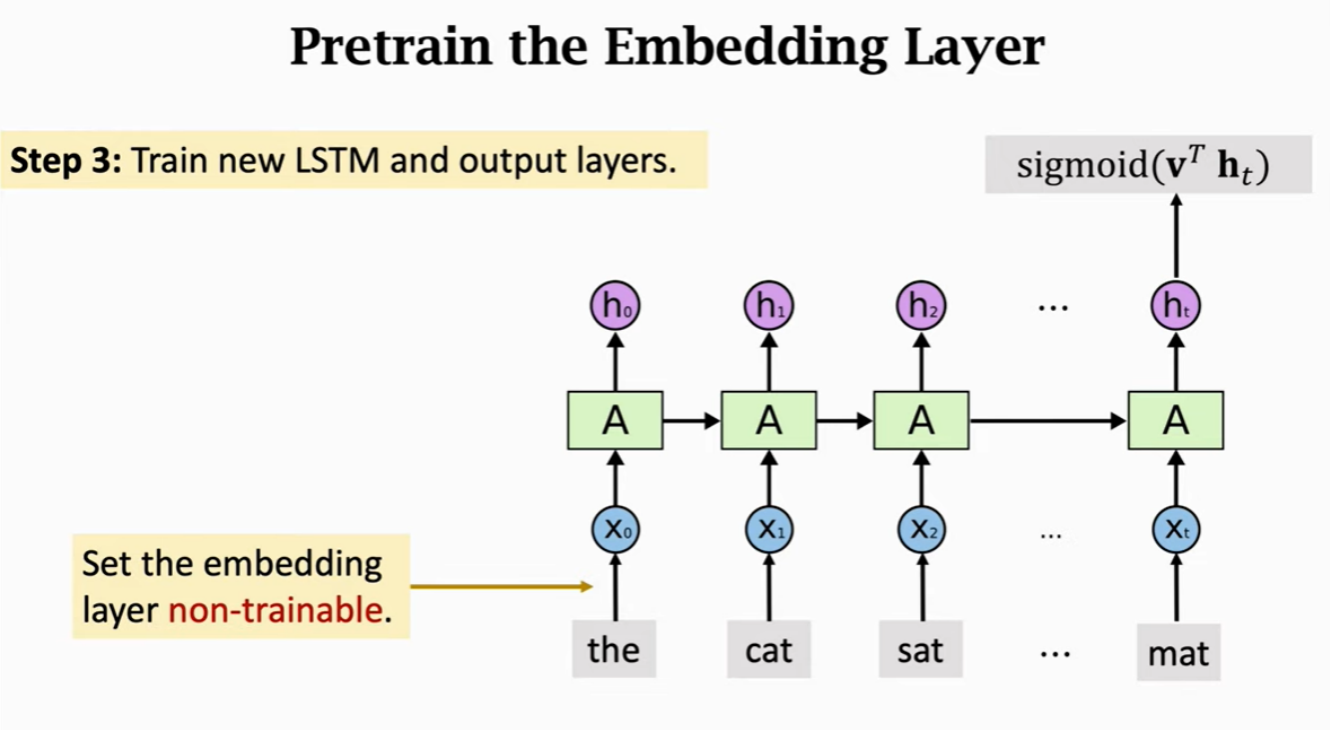
- Perhaps different problem.
- Perhaps different model.



Pretrain the Embedding Layer

Step 2: Keep only the embedding layer.





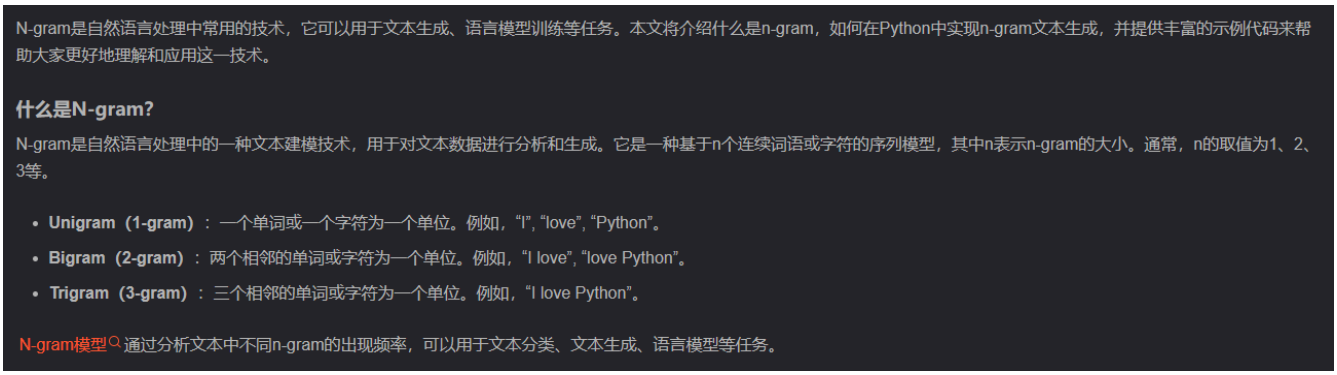
代码示例

```
import torch
import torch.nn as nn
import numpy as np
import os

# 假设 GloVe 词向量文件的路径
glove_file_path = 'glove.6B.100d.txt'  # 请替换为实际文件路径

# 读取 GloVe 词向量文件并创建字典
word_to_vec_map = {}
with open(glove_file_path, 'r', encoding='utf-8') as f:
    for line in f:
        values = line.split()
        word = values[0]
        vector = np.asarray(values[1:], dtype='float32')
        word_to_vec_map[word] = vector

# 假设你的词汇表大小和词向量的维度
vocab_size = len(word_to_vec_map)
embedding_dim = 100

# 创建一个随机初始化的嵌入矩阵
pretrained_embeddings = np.random.uniform(low=-0.25, high=0.25, size=(vocab_size,
embedding_dim))

# 用预训练的词向量更新嵌入矩阵
for word, vector in word_to_vec_map.items():
    if word in pretrained_embeddings:  # 确保词在词汇表中
```

```
        pretrained_embeddings[word] = vector

# 将 numpy 矩阵转换为 torch 张量
pretrained_embeddings = torch.from_numpy(pretrained_embeddings)

# 创建 Embedding 层
embedding_layer = nn.Embedding.from_pretrained(pretrained_embeddings)

# 测试 Embedding 层
input_indices = torch.tensor([1, 2, 988], dtype=torch.long)  # 假设这些是词汇表中的索引
output = embedding_layer(input_indices)
print(output)
```

自然语言处理

词向量编码



- 语言模型定义：对于语言序列 $w_1, w_2, \ldots\ldots, w_n$ ，语言模型就是计算该序列的概率，即 $\mathrm{P}(w_1, w_2, \ldots\ldots, w_n)$。

举例：1）我是个好人。2）好人我是个。

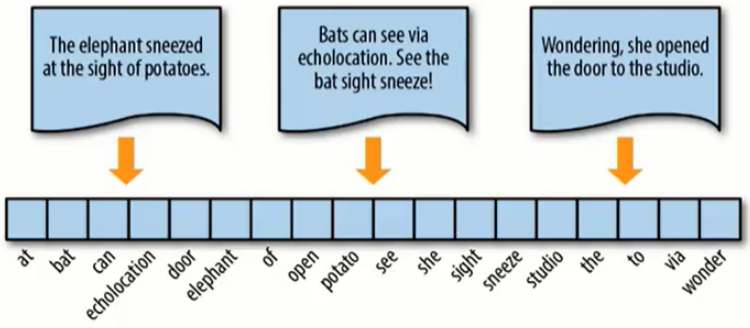
- n-gram语言模型：是一种以n-1阶马尔可夫模型的形式预测词序列下一项的概率语言模型。

n-gram语言模型指建立一个长度为n的窗口在文本上滑动，假定第n个词出现的概率只与前面n-1个词相关，与其他词不相关，整句的概率就是各个词出现概率的乘积。

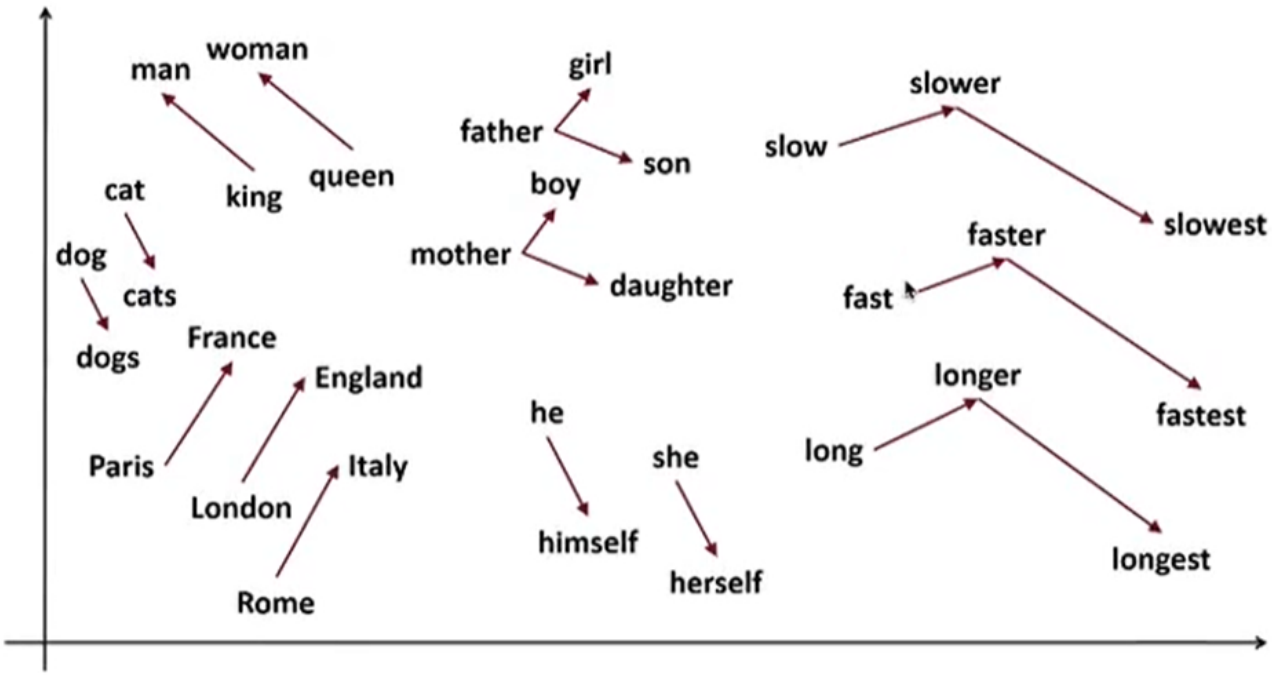
实验：词袋模型 (BoW)

实验概要

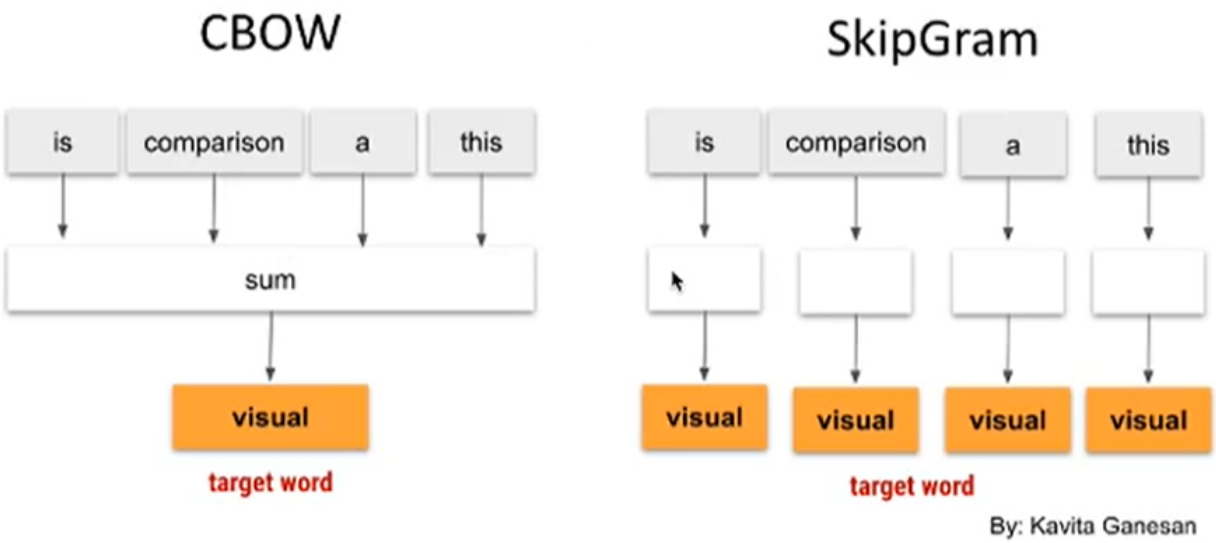
词袋 (Bag of Words) 模型是信息检索领域常用的文档表示方法。在信息检索中，BoW 模型假定对于一个文档，忽略它的单词顺序和语法、句法等要素，将其仅仅看作是若干个词汇的集合，文档中每个单词的出现都是独立的，不依赖于其它单词是否出现。也就是说，文档中任意一个位置出现的任何单词，都不受该文档语意影响而独立选择的。**词袋** (Bag of Words) 模型是基于一个假设：只需要确定了单词在语句中的是否出现，无论最终单词的排列顺序是怎么样的，最终表达的意思都将保持大致接近。



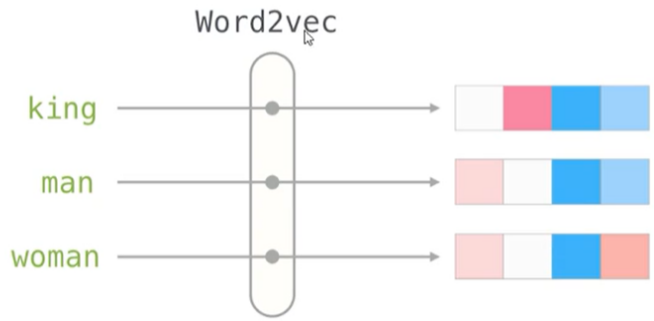
word2vec

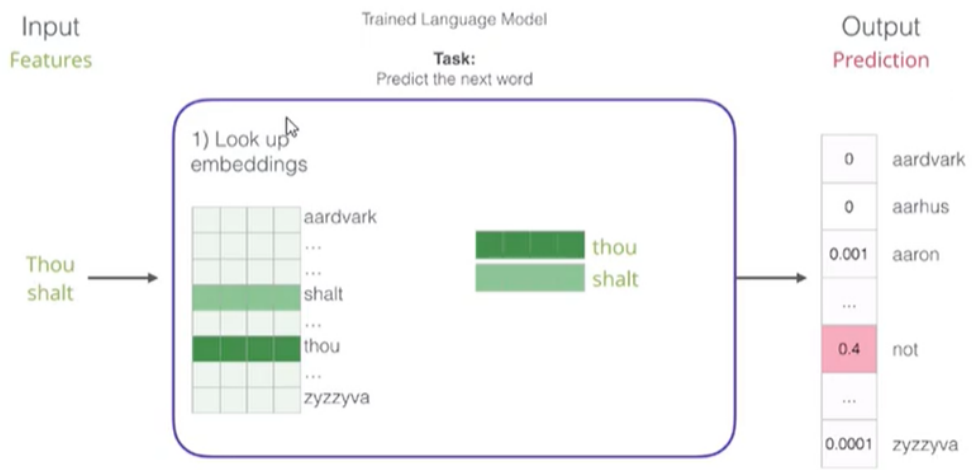


word2vec 不是模型而是**工具**，2013 Google 团队发表了 word2vec 工具。包含两个模型一个**跳字模型**(skip-gram)和**连续词袋模型**(CBow),word2vec 词向量可以比较好表达不同词之间的相似度和类比关系。



Word2vec 是一种有效创建词嵌入的方法，它自 2013 年以来就一直存在。但除了作为词嵌入的方法之外，它的一些概念已经被证明可以有效地创建推荐引擎和理解时序数据。在商业的、非语言的任务中。像 Airbnb、阿里巴巴、Spotify 这样的公司都从 NLP 领域中提取灵感并用于产品中，从而为新型推荐引擎提供支持。





CBOW 与 Skip-gram

我们不仅要考虑目标单词的前两个单词，还要考虑其后两个单词。

如果这么做，我们实际上构建并训练的模型就如下所示：

input 1	input 2	input 3	input 4	output
by	a	bus	in	red

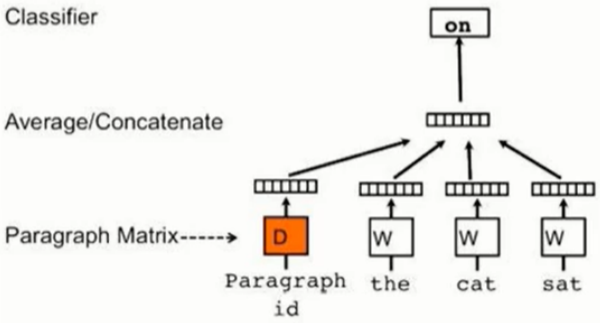
上述的这种架构被称为连续词袋 (**CBOW**)。

Dov2Vec基本原理

训练句向量的方法和词向量的方法非常类似。训练词向量的核心思想就是说可以根据每个单词 w_i 的上下文预测 w_i，也就是说上下文的单词对 w_i 是有影响的。那么同理，可以用同样的方法训练 doc2vec。例如对于一个句子 S

> i want to drink water

如果要去预测句子中的单词 `want`，那么不仅可以根据其他单词生成特征， 也可以根据其他单词和句子 S 来生成特征进行预测。因此 doc2vec 的框架如下所示：



每个段落/句子都被映射到向量空间中，可以用矩阵 D 的一列来表示。每个单词同样被映射到向量空间，可以用矩阵 W 的一列来表示。然后将段落向量和词向量级联或者求平均得到特征，预测句子中的下一个单词。这个段落向量/句向量也可以认为是一个单词，它的作用相当于是上下文的记忆单元或者是这个段落的主题，所以我们一般叫这种训练方法为：Distributed Memory Model of Paragraph Vectors (**PV-DM**)

在训练的时候我们固定上下文的长度，用滑动窗口的方法产生训练集。段落向量/句向量在该上下文中共享。

```
[  1.38192   1.82215  -1.25060     ...      0.19055
  -0.67227  -0.28646   0.47104     ...     -0.01183
  -1.51263  -0.31815  -1.21959     ...     -1.67615
     ...       ...       ...       ...        ...
   0.30669   1.5996    0.21065     ...     -1.85404 ]
```

CBOW和skip-gram的目标都是迭代出词向量字典(嵌入矩阵)

We are about to study the idea of deep learning.

We are about to study the idea of deep learning.

设置好窗口的长度后，需要通过窗口内的词语，预测目标词:

We、are、to、study -> about

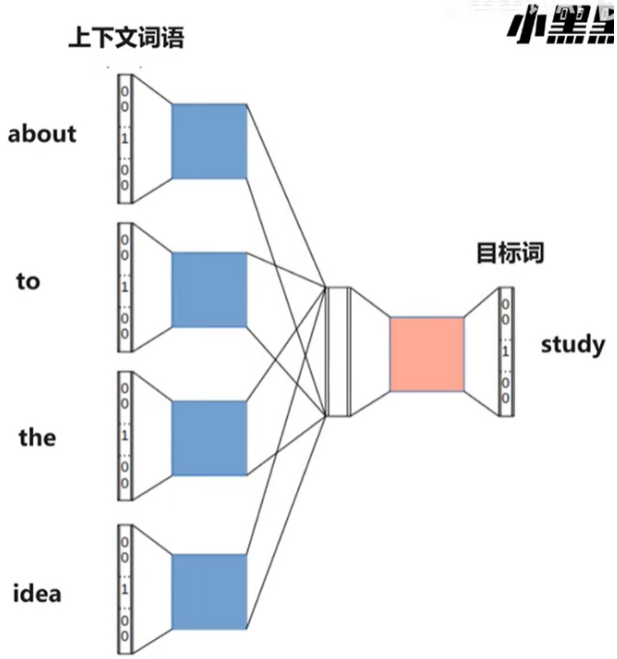
are、about、study、the -> to

about、to、the、idea -> study

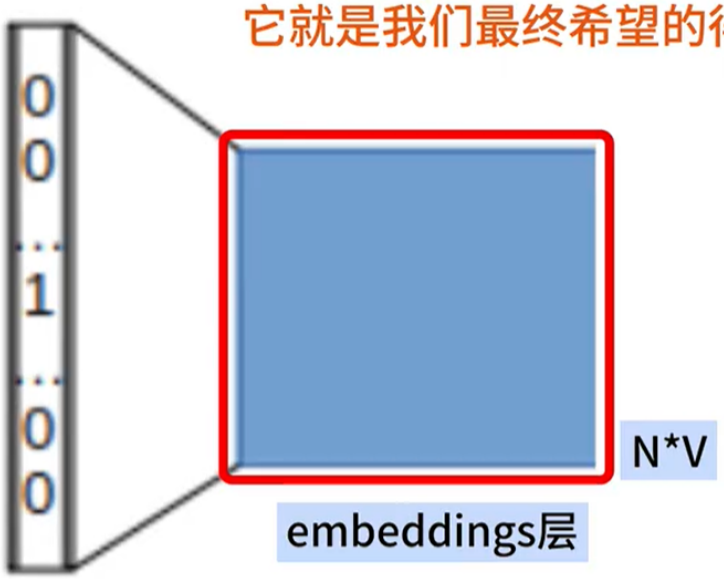
CBOW模型是一个神经网络:

该神经网络会接收上下文词语

将上下文词语转换为最有可能得目标词

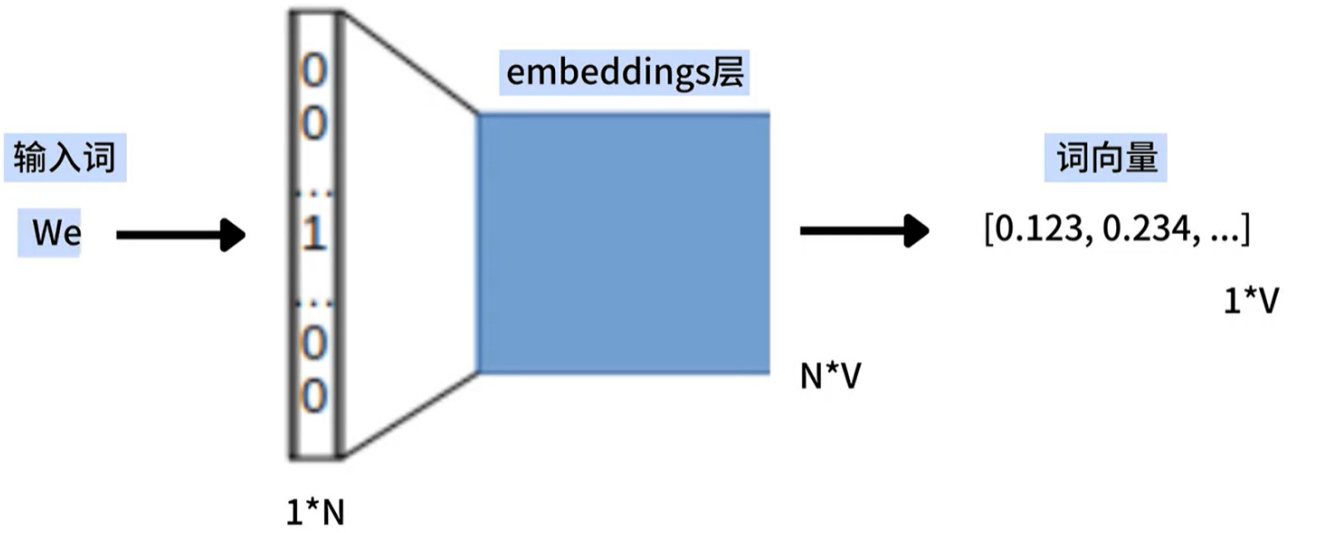


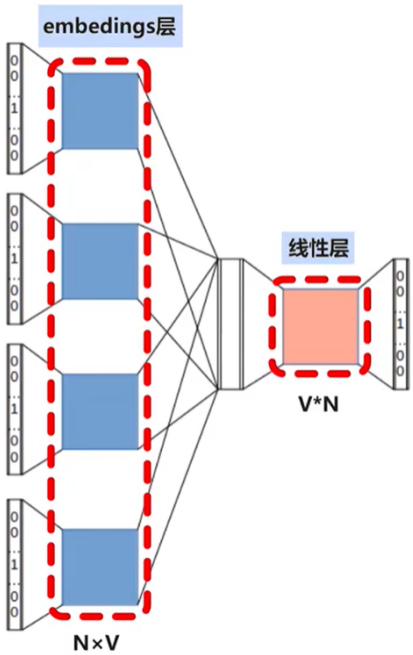
它就是我们最终希望的得到的嵌入矩阵



N：词表中的词语个数
V：词向量的维度

One-Hot



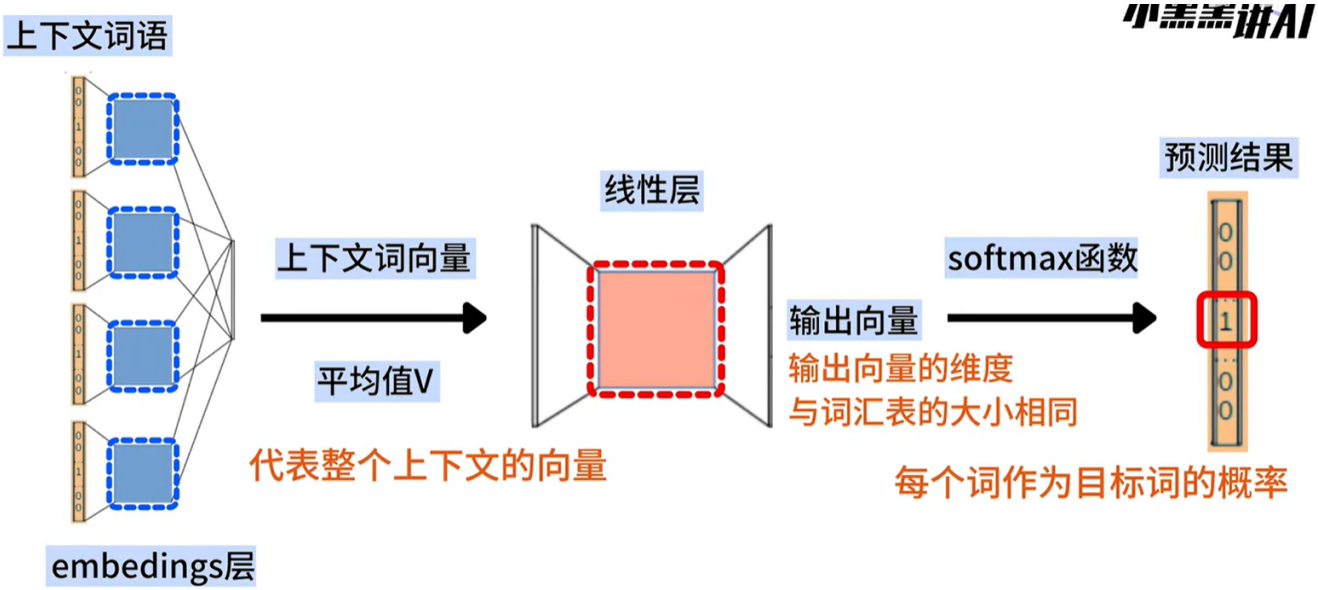


橙色标记的线性层:

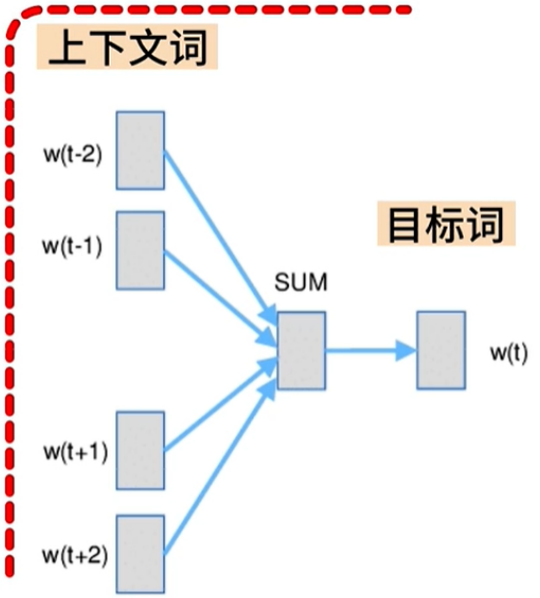
不设置激活函数

权重矩阵的维度: V*N
词向量维度: V
词表中词语的个数: N

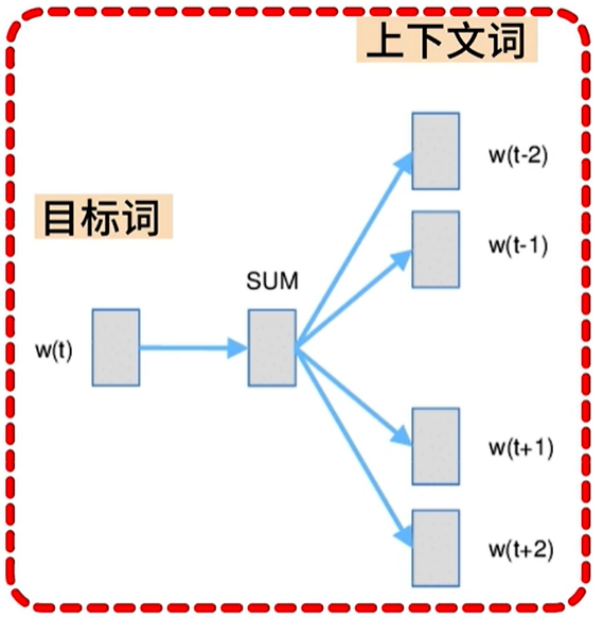
V个隐藏神经元、N个输出神经元的神经网络



Skip-gram



CBOW连续词袋模型



skip-gram跳字模型

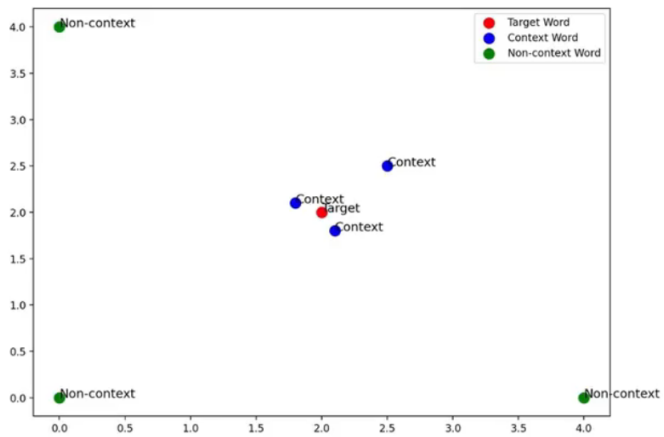
使用一个词，预测另一个词

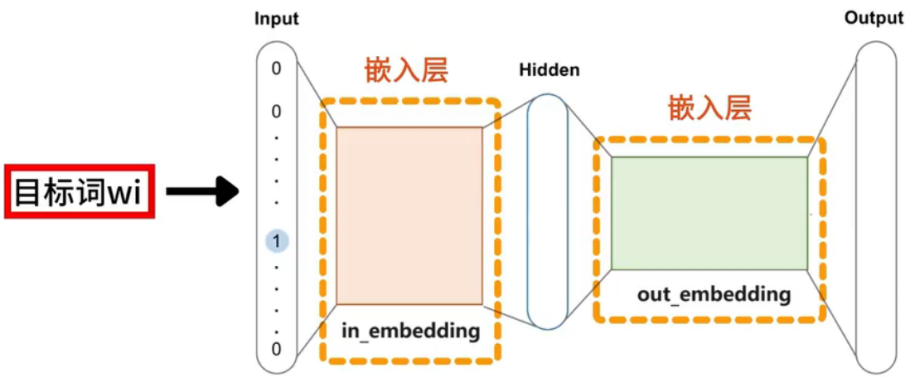
尽量使这两个词的词向量接近

Skip-gram在迭代时，调整词向量:

使目标词的词向量与其上下文的词向量尽可能的接近

使目标词的词向量与非上下文词的词向量尽可能的远离



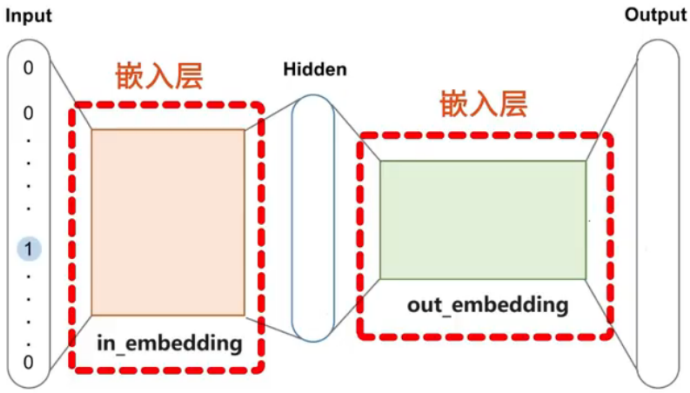


skip-gram模型

目标词的概率分布

词汇表中的每个词是目标词的上下文的可能性

词表中的词，与目标词有两种关系:
上下文词: 正样本，标记为1
非上下文词: 负样本，标记为0



作为skip-gram模型的输出

上下文　　不相关的词

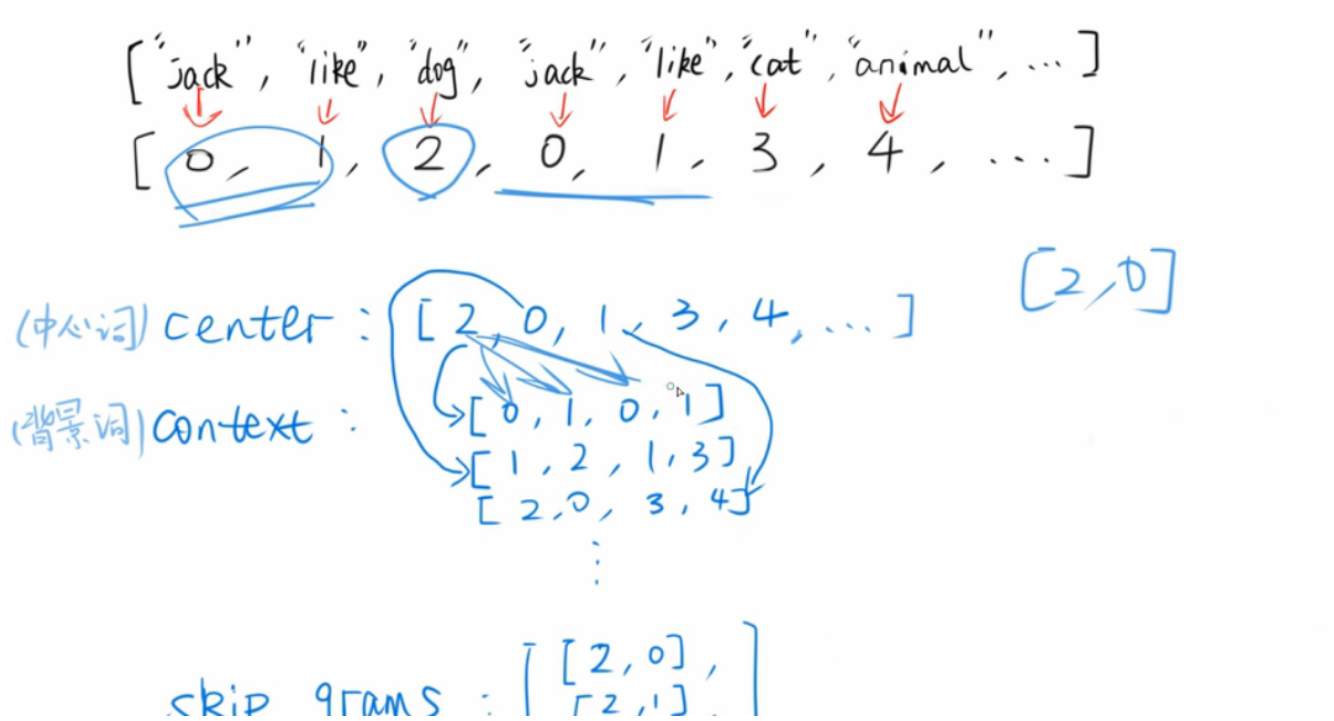
总词表: [study, about, to, the, idea, w1, w2, w3]

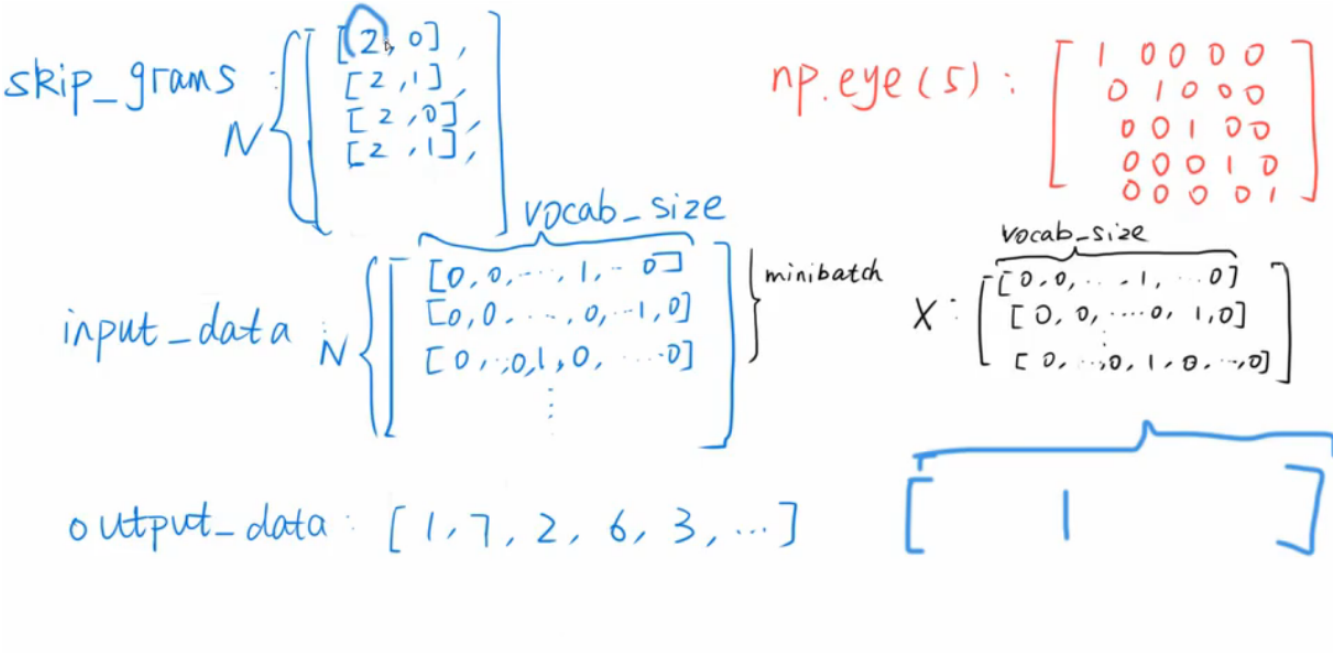
第1组:
1 正例(上下文词): about
0 负例(非上下文): to, the, idea, w1, w2, w3

第2组:
正例(上下文词): to
负例(非上下文): about, the, idea, w1, w2, w3

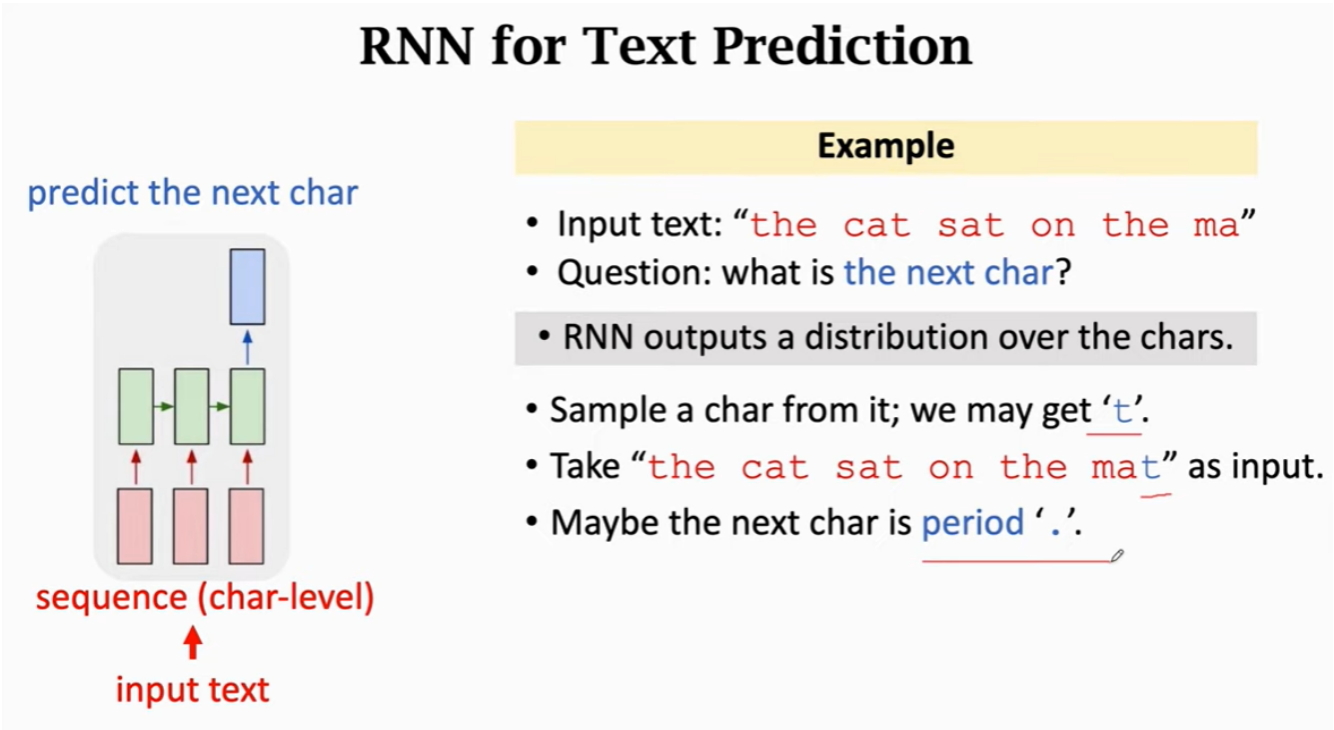
第3组:
正例(上下文词): the
负例(非上下文): about, to, idea, w1, w2, w3

第4组:
正例(上下文词): idea
负例(非上下文): about, to, the, w1, w2, w3



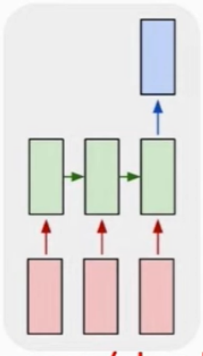

skip_grams
N
[2, 0],
[2, 1],
[2, 0],
[2, 1],
vocab_size
np.eye(5):
input_data
N
[0, 0, ..., 1, ... 0]
[0, 0, ..., 0, 1, 0]
[0, 0, 1, 0, ... 0]
minibatch
X:
vocab_size
[0, 0, ... 1, ... 0]
[0, 0, ... 0, 1, 0]
[0, ... 0, 1, 0, ... 0]
output_data: [1, 7, 2, 6, 3, ...]
[1]

文本生成


RNN for Text Prediction
predict the next char
sequence (char-level)
input text
Example
• Input text: "the cat sat on the ma"
• Question: what is the next char?
• RNN outputs a distribution over the chars.
• Sample a char from it; we may get 't'.
• Take "the cat sat on the mat" as input.
• Maybe the next char is period '.'.

Train an RNN for Text Prediction

predict the next char



sequence (char-level)

input text

How do we train such an RNN?

- Cut text to segments (with overlap).
- A segment is used as input text.
- Its next char is used as label.
- Training data: `(segment, next_char)` pairs
- It is a multi-class classification problem.
  - #class = #unique chars.

If the RNN is trained on Shakespeare's books, then the generated text is Shakespeare's style.

1. Prepare Training Data

```
print('Text length:', len(text))
```

```
Text length: 600893
```

```
text[0:1000]
```

```
'preface\n\n\nsupposing that truth is a woman--what then? is there
not ground\nfor suspecting that all philosophers, in so far as they
have been\ndogmatists, have failed to understand women--that the te
rrible\nseriousness and clumsy importunity with which they have usu
ally paid\ntheir addresses to truth, have been unskilled and unseem
ly methods for\nwinning a woman? certainly she has never allowed he
rself to be won; and\nat present every kind of dogma stands with sa
d and discouraged mien--if,\nindeed, it stands at all! for there ar
e scoffers who maintain that it\nhas fallen, that all dogma lies on
the ground--nay more, that it is at\nits last gasp. but to speak se
riously, there are good grounds for hoping\nthat all dogmatizing in
philosophy, whatever solemn, whatever conclusive\nand decided airs
it has assumed, may have been only a noble puerilism\nand tyronism;
and probably the time is at hand when it will be once\nand again un
```

1. Prepare Training Data

segment:

'preface\n\n\nsupposing that truth is a woman--what then? is there not ground\nfor suspecting that all philosophers, in so far as they have been\ndogmatists, have failed to understand women--that the terrible\nseriousness and clumsy importunity with which they have usually paid\ntheir addresses to truth, have been unskilled and unseem

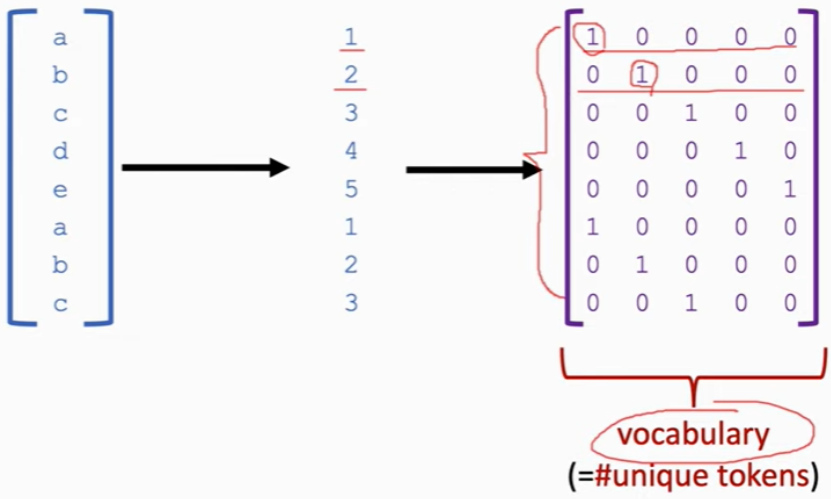
segments[0]:	preface\n\n\nsupposing that truth	**next_chars[0]:**	**i**
segments[1]:	face\n\n\nsupposing that truth is	**next_chars[1]:**	**a**
segments[2]:	e\n\n\nsupposing that truth is a w	**next_chars[2]:**	**o**
segments[3]:	\nsupposing that truth is a woma	**next_chars[3]:**	**n**
⋮		⋮	

2. Character to Vector

Dictionary

Token	Index
a	1
b	2
c	3
d	4
e	5

One-hot encoding (token to vector)



2. Character to Vector

- Vocabulary is $v = 57$ (including letter, blank, punctuation, etc.)
- Segment length is $l = 60$. (A segment has 60 chars.)
- Number of segments is $n = 200{,}278$. (Number of training samples.)

`segments[i]:` `\nsupposing that truth is a woma` → **X**: 60×57 matrix

`next_chars[i]:` `n` → **y**: 57×1 vector

There are $n = 200{,}278$ such pairs.

3. Build a Neural Network

```
from keras import layers

model = keras.models.Sequential()
model.add(layers.LSTM(128, input_shape=(seg_len, vocabulary)))
model.add(layers.Dense(vocabulary, activation='softmax'))
model.summary()
```

$l = 60$ $v = 57$ $v = 57$

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_1 (LSTM)                (None, 128)               95232
_________________________________________________________________
dense_1 (Dense)              (None, 57)                7353
=================================================================
Total params: 102,585
Trainable params: 102,585
Non-trainable params: 0
```

Predict the Next Char

```
pred = model.predict(x_input, verbose=0)[0]
```

- A segment of 60 chars (represented by a 60×57 matrix).

The probability distribution over the 57 unique chars.

Predict the Next Char

```
pred = model.predict(x_input, verbose=0)[0]
```

Question: Given the distribution, what will be the next char?

- Option 1: greedy selection.
- Option 2: sampling from the multinomial distribution.
- Option 3: adjusting the multinomial distribution.
  - `pred = pred ** (1 / temperature)`
  - `pred = pred / np.sum(pred)`
  - Sample according to `pred`.
  - Between greedy and multinomial (controlled `temperature`).

Text Generation: An Example

Example: seg_len=18

initial_input (seed): "the cat sat on the"

next_char: '_' (blank)

input: "he cat sat on the "

next_char: 'm'

input: "e cat sat on the m"

next_char: 'a'

Generate Text Using the Trained LSTM

Initial input (seed)

"immediately disclose what it really is--namely, a will to the"

Generated text after 20 epochs

"immediately disclose what it really is--namely, a will to the self-religious superitial self-partial religion himself of the superstition of the contrast in the accomant in the person of the assertion, at the valuations and consequences and things has no longer and how only an man of the soul and manifest of the disciplides and an whom all to the constance of its own power, in the constraining in the truth of the strength of life of the see to remain in the"

Train a Neural Network

1. Partition text to (segment, next_char) pairs.
2. One-hot encode the characters.
   - Character → $v \times 1$ vector.
   - Segment → $l \times v$ matrix.
3. Build and train a neural network.
   - $l \times v$ matrix ==> LSTM ==> Dense ==> $v \times 1$ vector.

Text Generation

1. Propose a seed segment.
2. Repeat the followings:
   a) Feed the segment (with one-hot) to the neural network.
   b) The neural network outputs probabilities.
   c) next_char ← Sample from the probabilities.
   d) Append next_char to the segment.

机器翻译与Seq2Seq

Sequence-to-Sequence Model (Seq2Seq)

English		German
"do you agree"	=> **[Seq2Seq]** =>	"bist du einverstanden"
"go to sleep"	=> **[Seq2Seq]** =>	"gehen Sie schlafen"
"We will fight"	=> **[Seq2Seq]** =>	"Wir werden kämpfen"

1. Tokenization & Build Dictionary

- `input_texts => [Eng_Tokenizer] => input_tokens`
- `target_texts => [Deu_Tokenizer] => target_tokens`

Tokenization in the char-level.

Tokenization in the word-level.

1. Tokenization & Build Dictionary

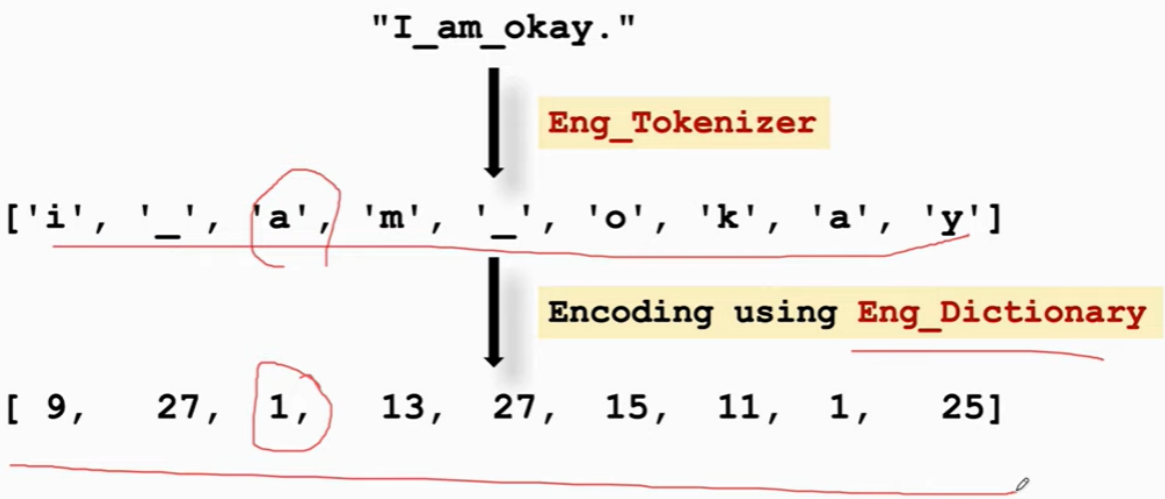
Eng_Dictionary

- 'a' => 1
- 'b' => 2
- 'c' => 3
- 'd' => 4
- …
- 'z' => 26
- '_' => 27

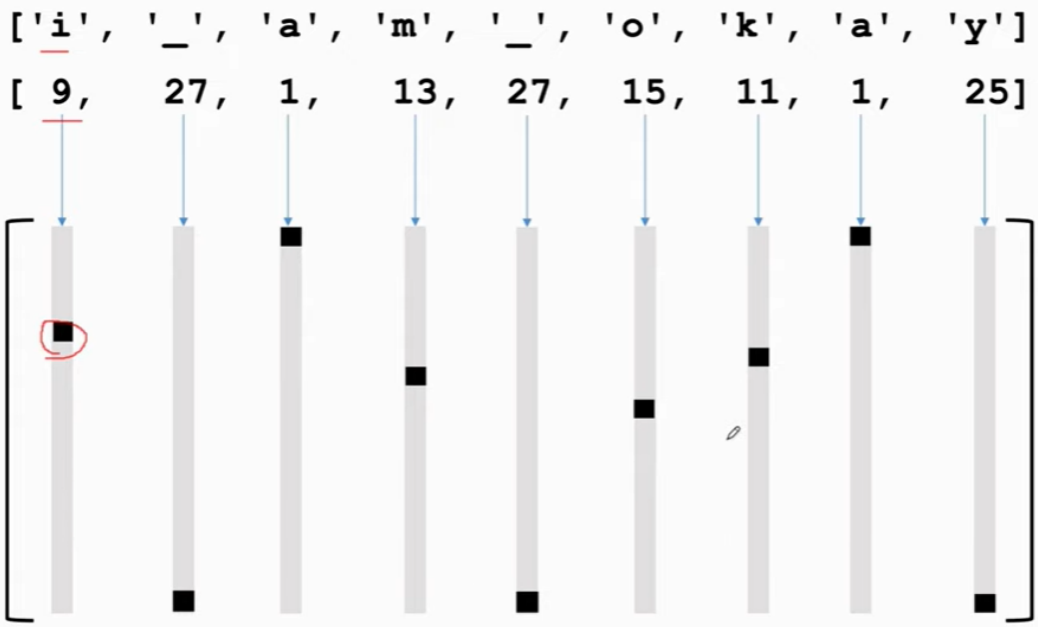
Deu_Dictionary

- '\t' => 1 ← start sign
- '\n' => 2 ← stop sign
- 'a' => 3
- 'b' => 4
- 'c' => 5
- 'd' => 6
- …
- 'z' => 28
- '_' => 29

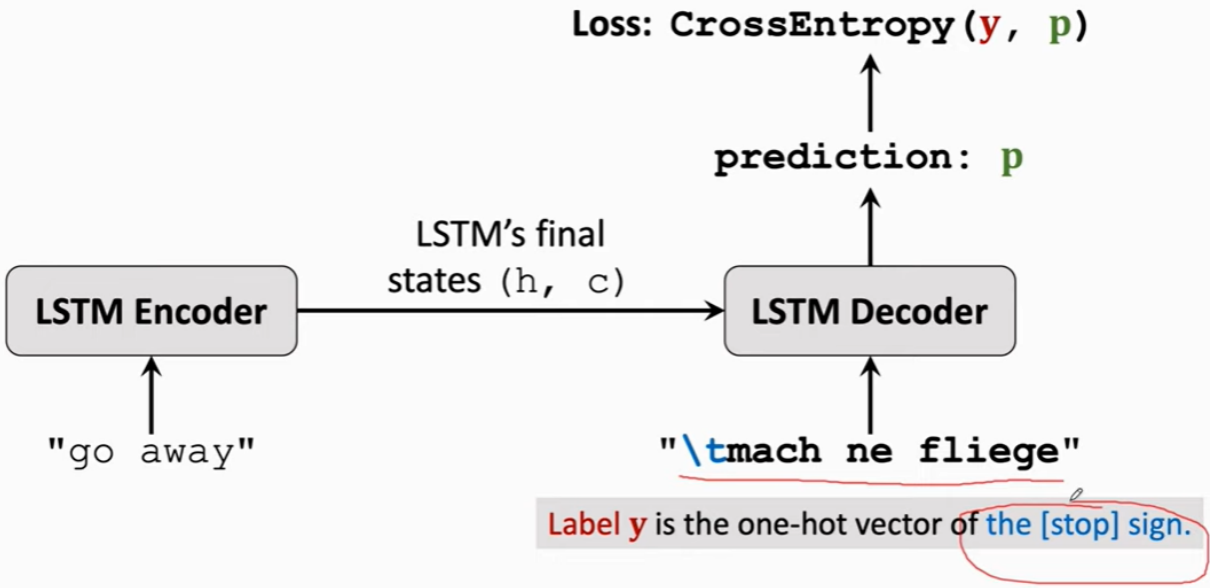
2. One-Hot Encoding



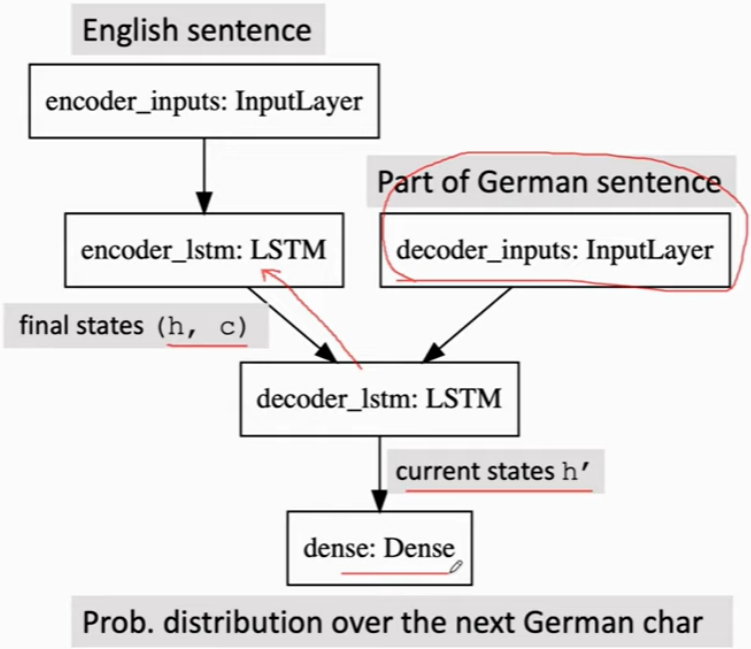
2. One-Hot Encoding



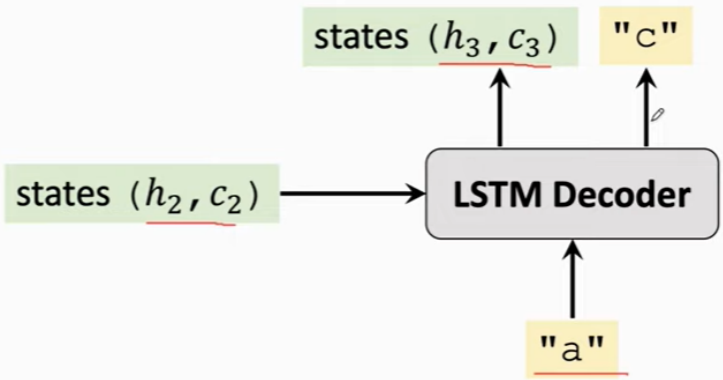
Training Seq2Seq Model



Seq2Seq Model in Keras

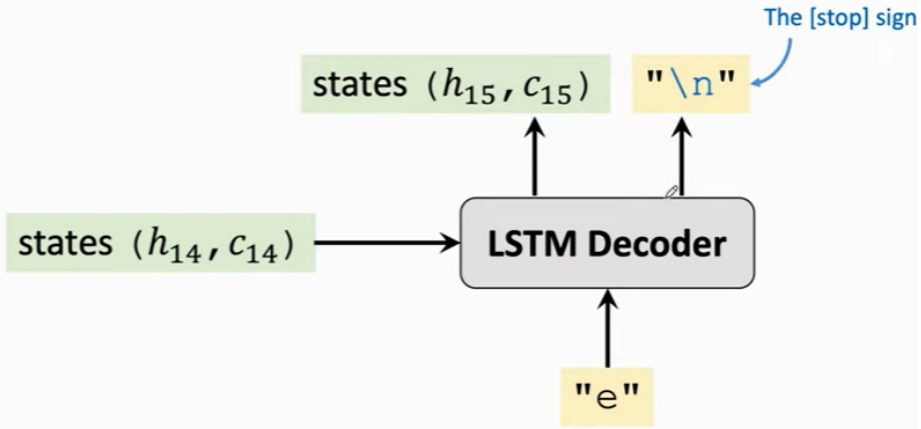


Inference



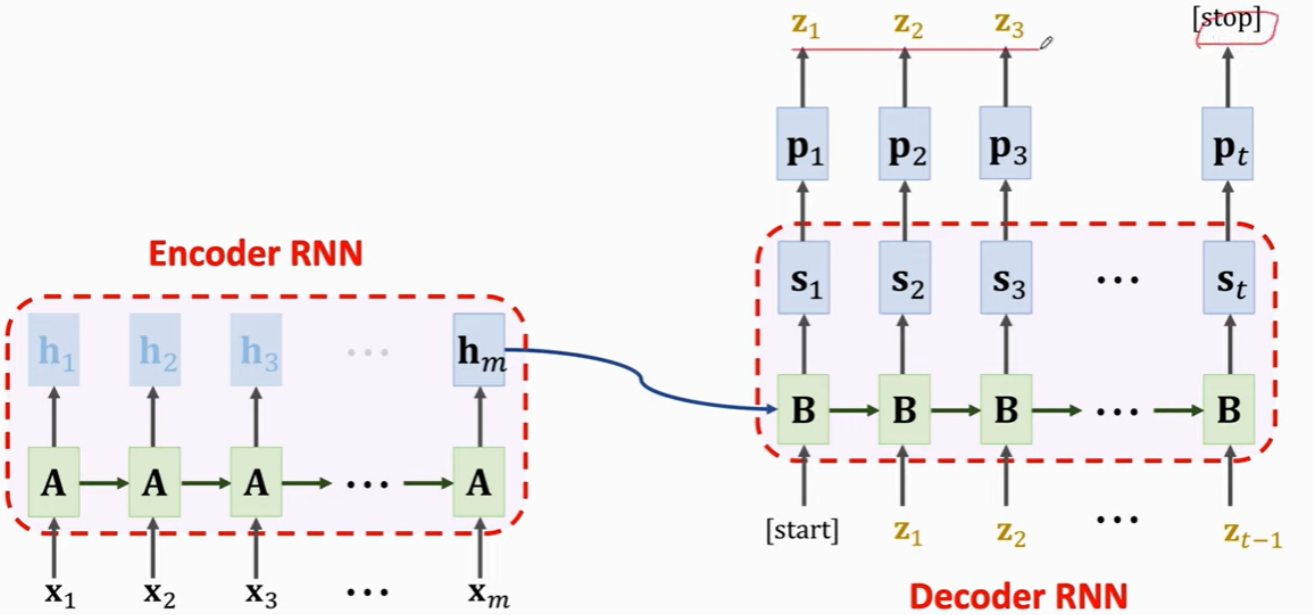
Record: "mac"

Inference



Record: "mach ne fliege"

Seq2Seq Model



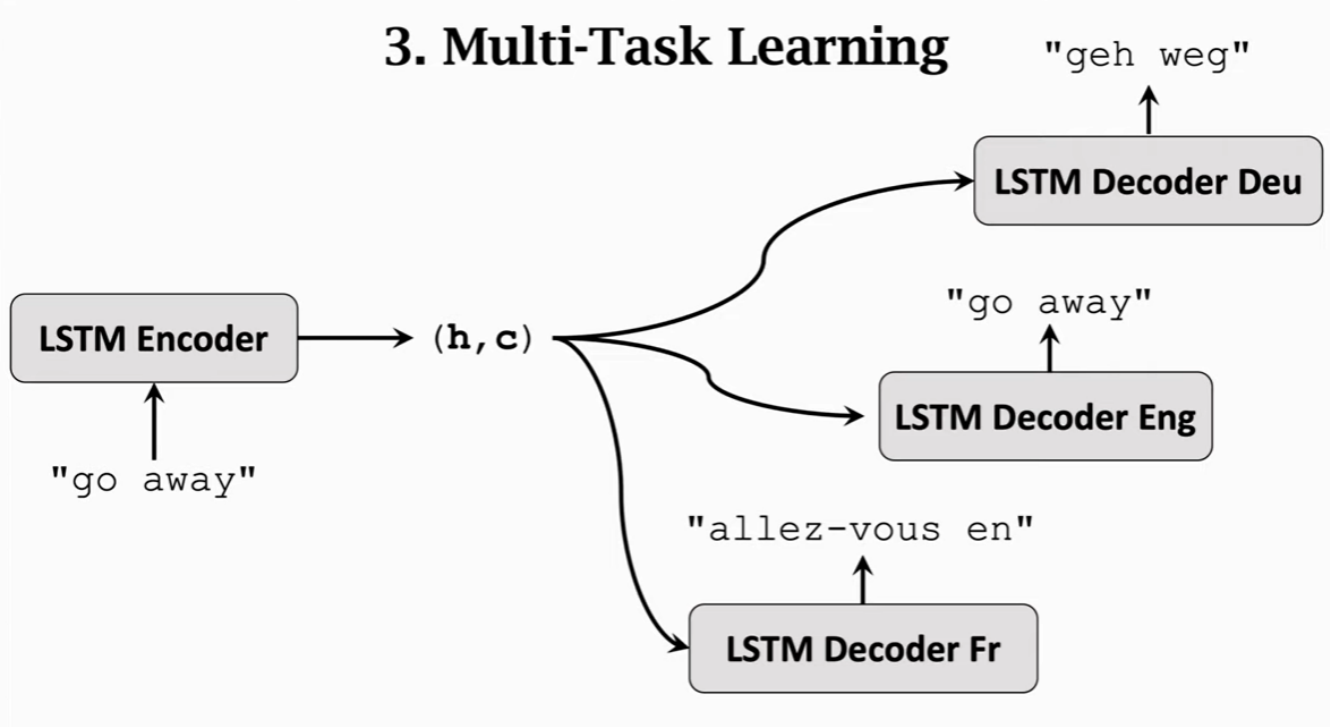
1. Bi-LSTM instead of LSTM (Encoder only!)

- Encoder's final states ($\mathbf{h}_t$ and $\mathbf{c}_t$) have all the information of the English sentence.
- If the sentence is long, the final states have forgotten early inputs.
- Bi-LSTM (left-to-right and right-to-left) has longer memory.
- Use Bi-LSTM in the encoder; use unidirectional LSTM in the decoder.

2. Word-Level Tokenization

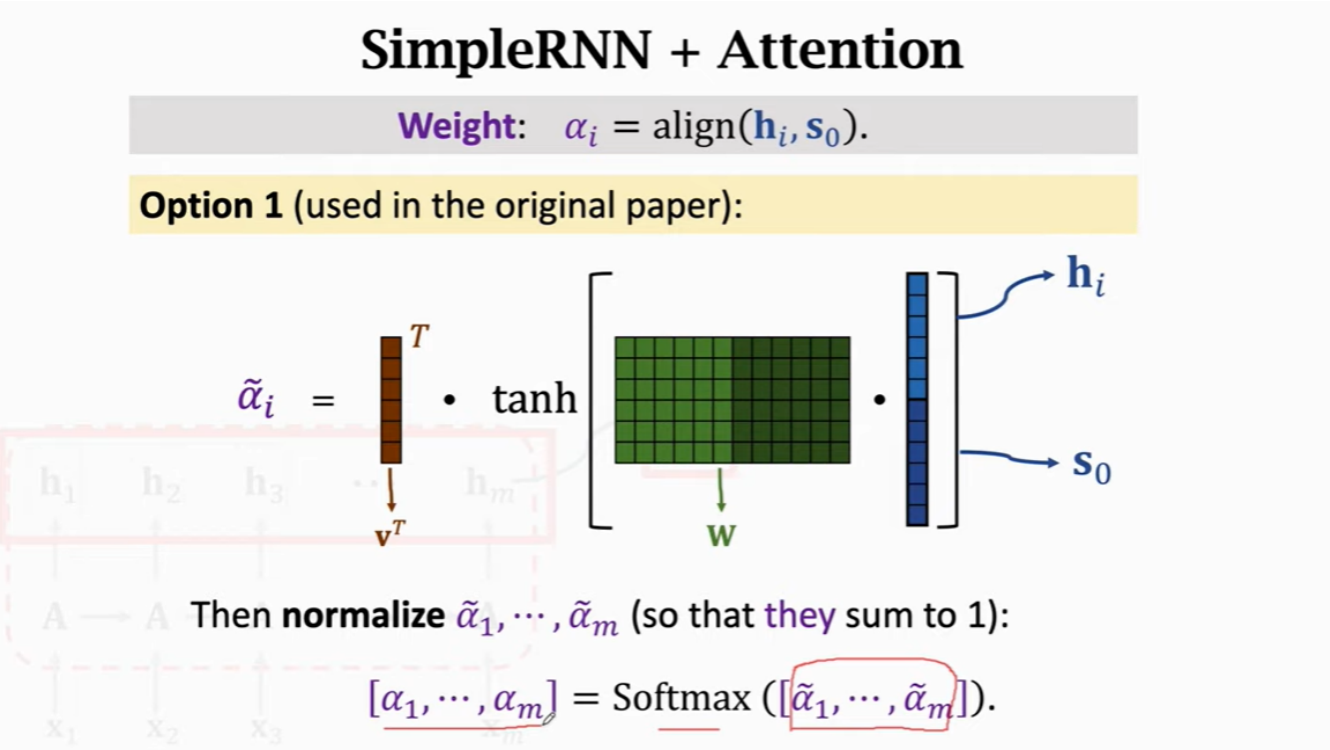
- Word-level tokenization instead of char-level.
  - The average length of English words is 4.5 letters.
  - The sequences will be 4.5x shorter.
  - Shorter sequence ➔ less likely to forget.

- But you will need a large dataset!
  - # of (frequently used) chars is ~10^2 ➔ one-hot suffices.
  - # of (frequently used) words is ~10^4 ➔ must use embedding.
  - Embedding Layer has many parameters ➔ overfitting!



机器翻译评价标准—BLEU

Attention



SimpleRNN + Attention

Weight: $\alpha_i = \text{align}(\mathbf{h}_i, \mathbf{s}_0)$.

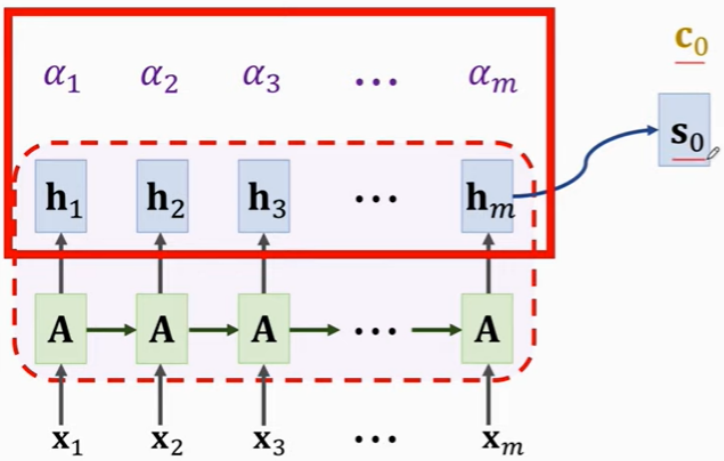
Option 2 (more popular; the same to Transformer):

1. Linear maps:
   - $\mathbf{k}_i = \mathbf{W}_K \cdot \mathbf{h}_i$, for $i = 1$ to m.
   - $\mathbf{q}_0 = \mathbf{W}_Q \cdot \mathbf{s}_0$.
2. Inner product:
   - $\tilde{\alpha}_i = \mathbf{k}_i^T \mathbf{q}_0$, for $i = 1$ to m.
3. Normalization:
   - $[\alpha_1, \cdots, \alpha_m] = \text{Softmax}([\tilde{\alpha}_1, \cdots, \tilde{\alpha}_m])$.

SimpleRNN + Attention

Weight: $\alpha_i = \text{align}(\mathbf{h}_i, \mathbf{s}_0)$.

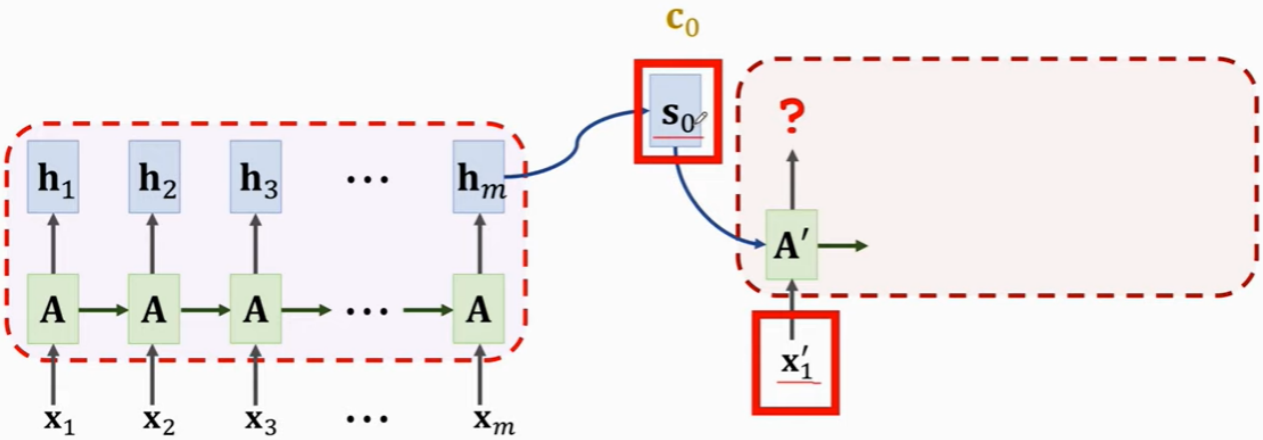
Context vector: $\mathbf{c}_0 = \alpha_1 \mathbf{h}_1 + \cdots + \alpha_m \mathbf{h}_m$.



SimpleRNN

SimpleRNN:

$$\mathbf{s}_1 = \tanh\left(\mathbf{A}' \cdot \begin{bmatrix}\mathbf{x}'_1 \\ \mathbf{s}_0\end{bmatrix} + \mathbf{b}\right)$$



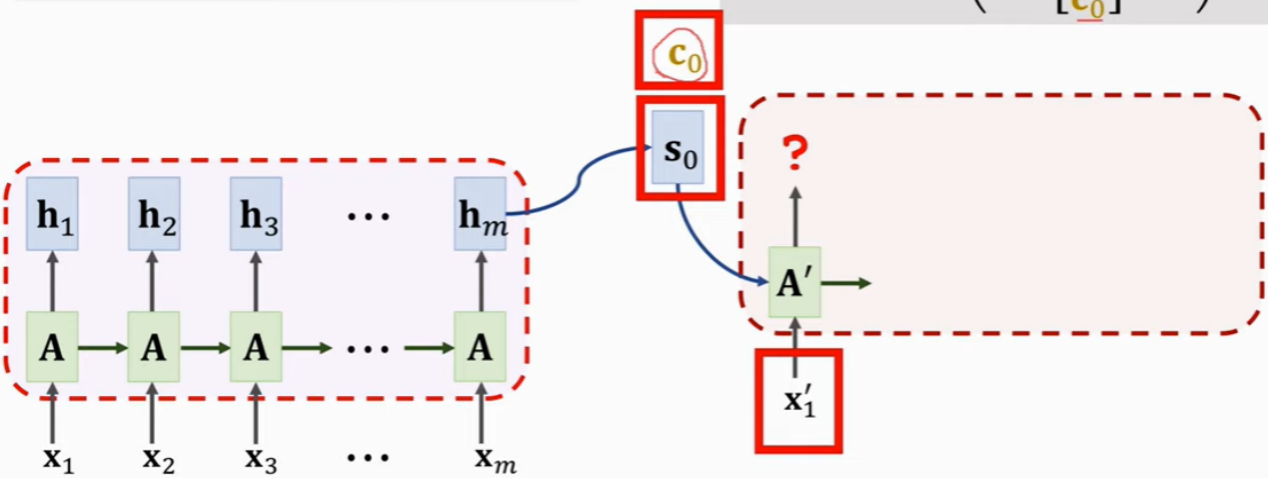
SimpleRNN + Attention

SimpleRNN:

$$\mathbf{s}_1 = \tanh\left(\mathbf{A}' \cdot \begin{bmatrix}\mathbf{x}'_1 \\ \mathbf{s}_0\end{bmatrix} + \mathbf{b}\right)$$

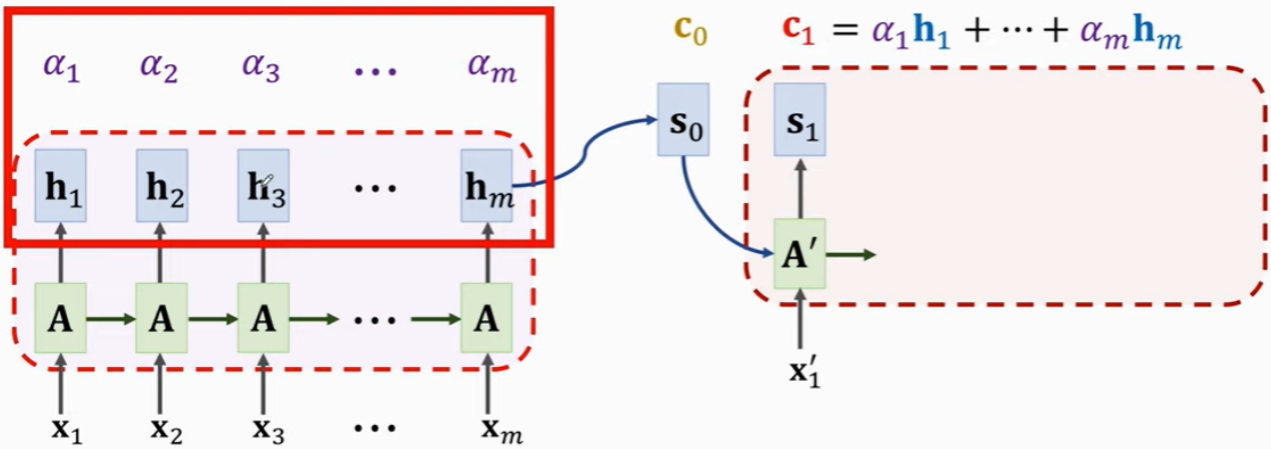
SimpleRNN + Attention:

$$\mathbf{s}_1 = \tanh\left(\mathbf{A}' \cdot \begin{bmatrix}\mathbf{x}'_1 \\ \mathbf{s}_0 \\ \mathbf{c}_0\end{bmatrix} + \mathbf{b}\right)$$

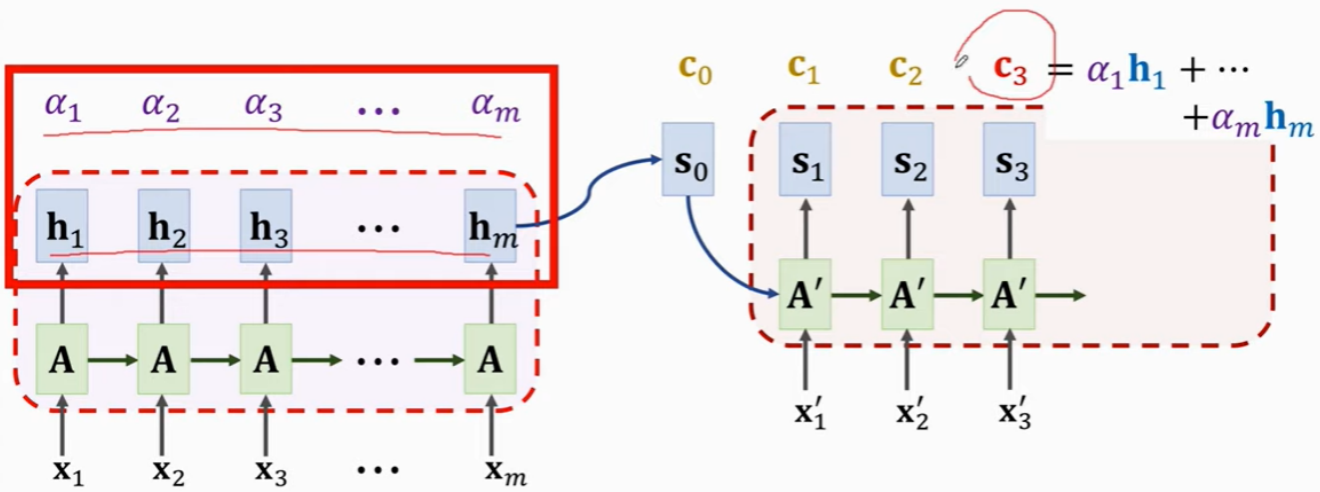


SimpleRNN + Attention

Weight: $\alpha_i = \text{align}(\mathbf{h}_i, \mathbf{s}_1)$.



SimpleRNN + Attention



Summary

- Standard Seq2Seq model: the decoder looks at only its current state.
- Attention: decoder additionally looks at all the states of the encoder.
- Attention: decoder knows where to focus.

- Downside: higher time complexity.
  - m: source sequence length
  - t: target sequence length
  - Standard Seq2Seq: $O(m + t)$ time complexity
  - Seq2Seq + attention: $O(mt)$ time complexity

self-Attention

SimpleRNN + Self-Attention

SimpleRNN:

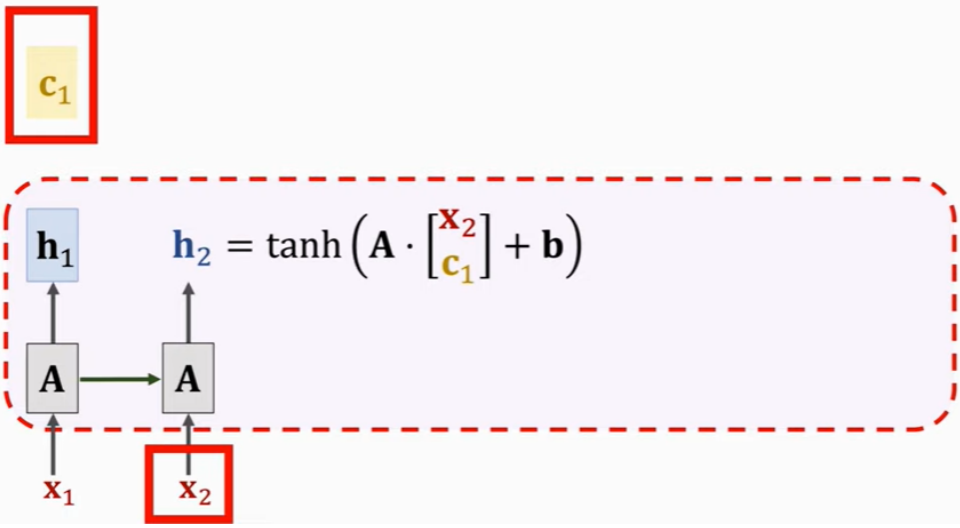
$$\mathbf{h}_1 = \tanh\left(\mathbf{A} \cdot \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{h}_0 \end{bmatrix} + \mathbf{b}\right)$$

SimpleRNN + Self-Attention:

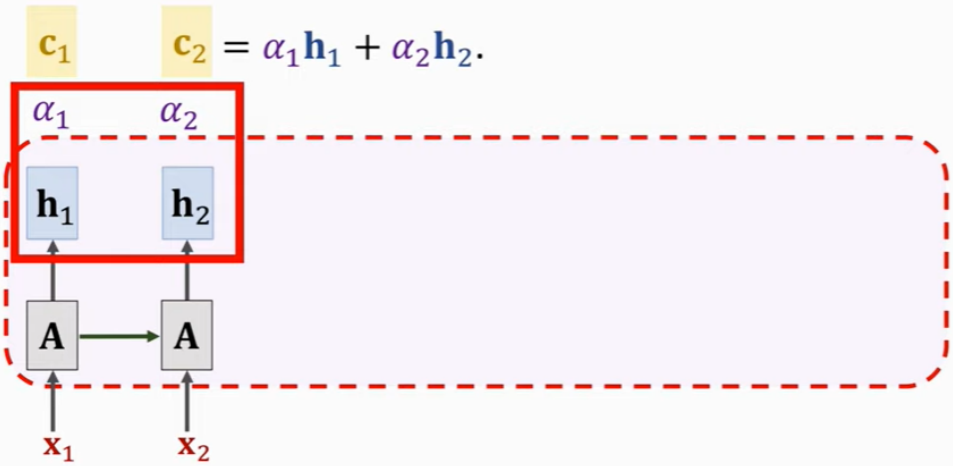
$$\mathbf{h}_1 = \tanh\left(\mathbf{A} \cdot \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{c}_0 \end{bmatrix} + \mathbf{b}\right)$$

$\mathbf{h}_0$

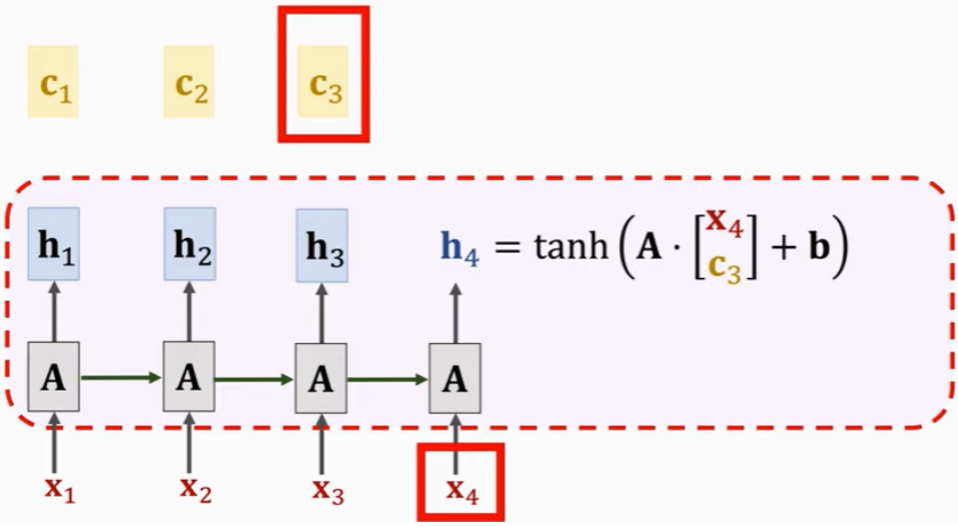
SimpleRNN + Self-Attention



SimpleRNN + Self-Attention

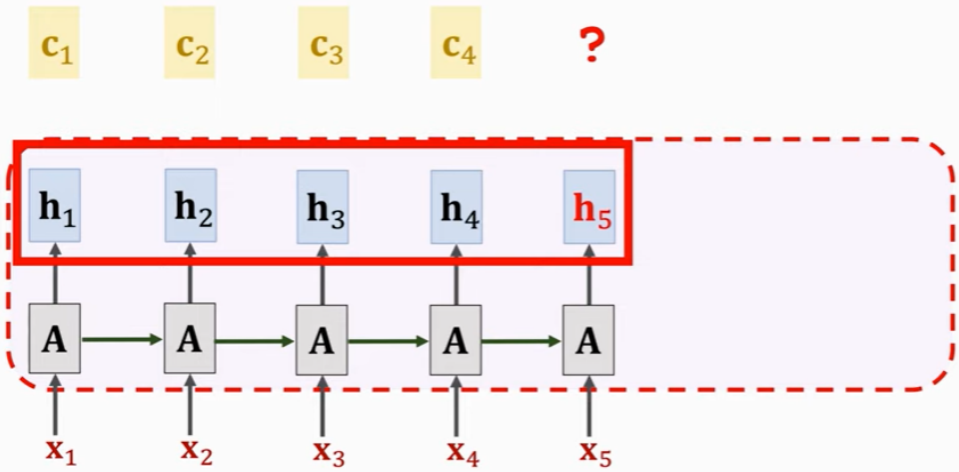


SimpleRNN + Self-Attention



SimpleRNN + Self-Attention

Weights: $\alpha_i = \text{align}(h_i, h_5)$.



Summary

- With self-attention, RNN is less likely to forget.
- Pay attention to the context relevant to the new input.

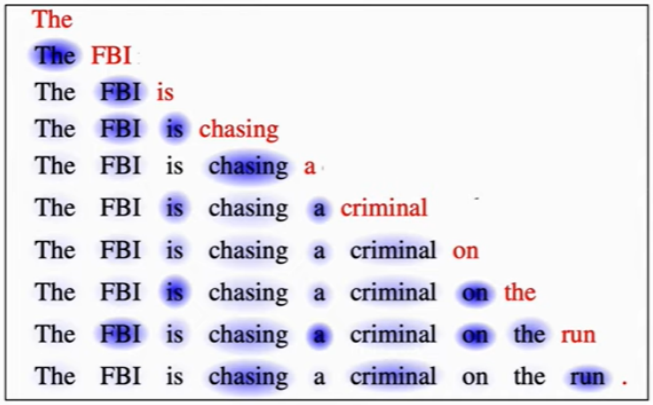


Figure is from the paper " Long Short-Term Memory-Networks for Machine Reading."