揚州大學
YANGZHOU UNIVERSITY

自 然 语 言 处 理

Natural language Processing

基于深度学习框架的序列处理

第6章

CONTENTS

语言的本质

语言本身是人类进化发展伴随而来的一种时间现象。

口语生成和变化是随着随着时间的推移的一系列声学时间。

口语和人类文字语言可以理解为一个连续输出流。

语言的时间性体现在：人类语言是随着时间展开的序列。

- flow of conversations,
- news feeds, and
- twitter streams

语言处理

语言的时间性质反映在用于处理语言的一些算法中。

- 例如：可用于隐式马尔科夫模型(HMM)词性标记的Viterbi算法
  - 一次输入一个单词，并且携带之前文本的信息

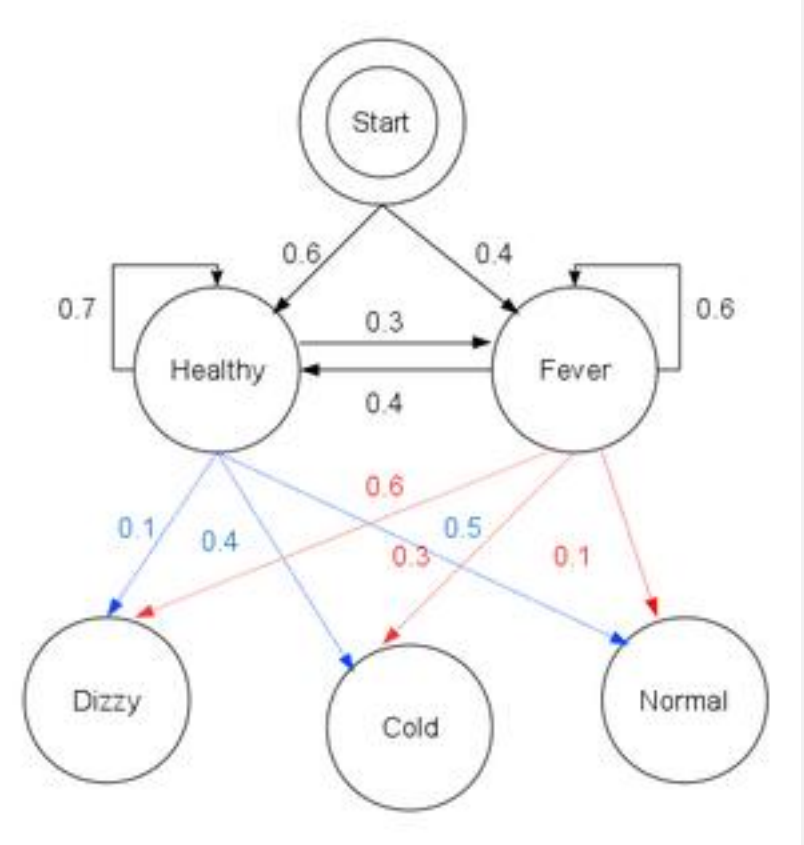
不具有时间性质的机器学习模型：

- 例如之前学习过的情感分析或者其他的文本分类任务
  - 因为这些模型把单词序列当做一个整体对待，
  - 即同时访问并处理输入的所有信息。

initial probabilities

transition probabilities

emission probabilities



Part.01

语言模型回顾

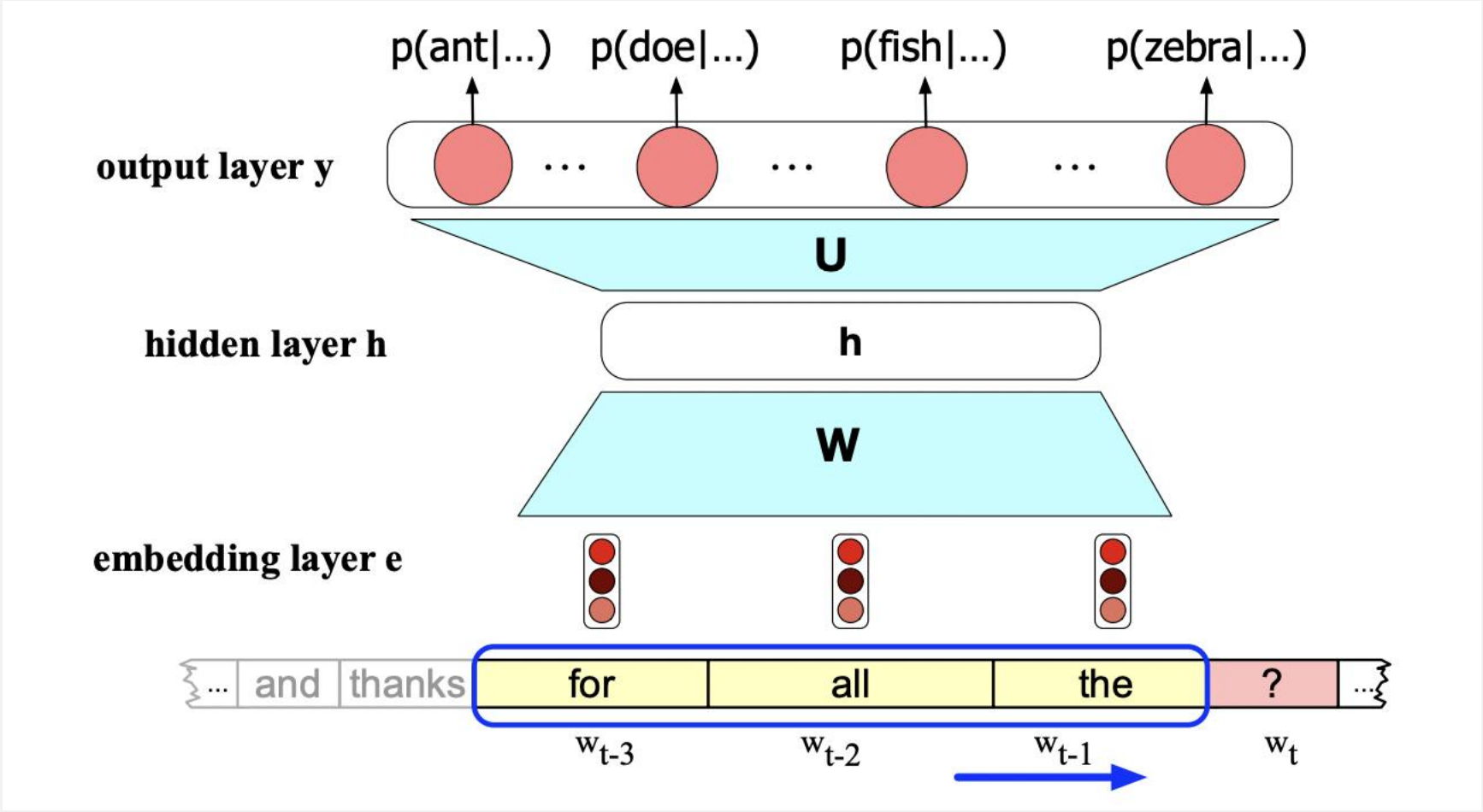
回顾：前馈神经网络

上一章中讲到的前馈神经网络

- 假设对数据进行同时访问

- 尽管也具有一个有关时间的简单模型
  - 有固定大小的单词窗口，
  - 在输入上滑动窗口，
  - 在该过程中完成独立预测。

回顾：前馈神经网络

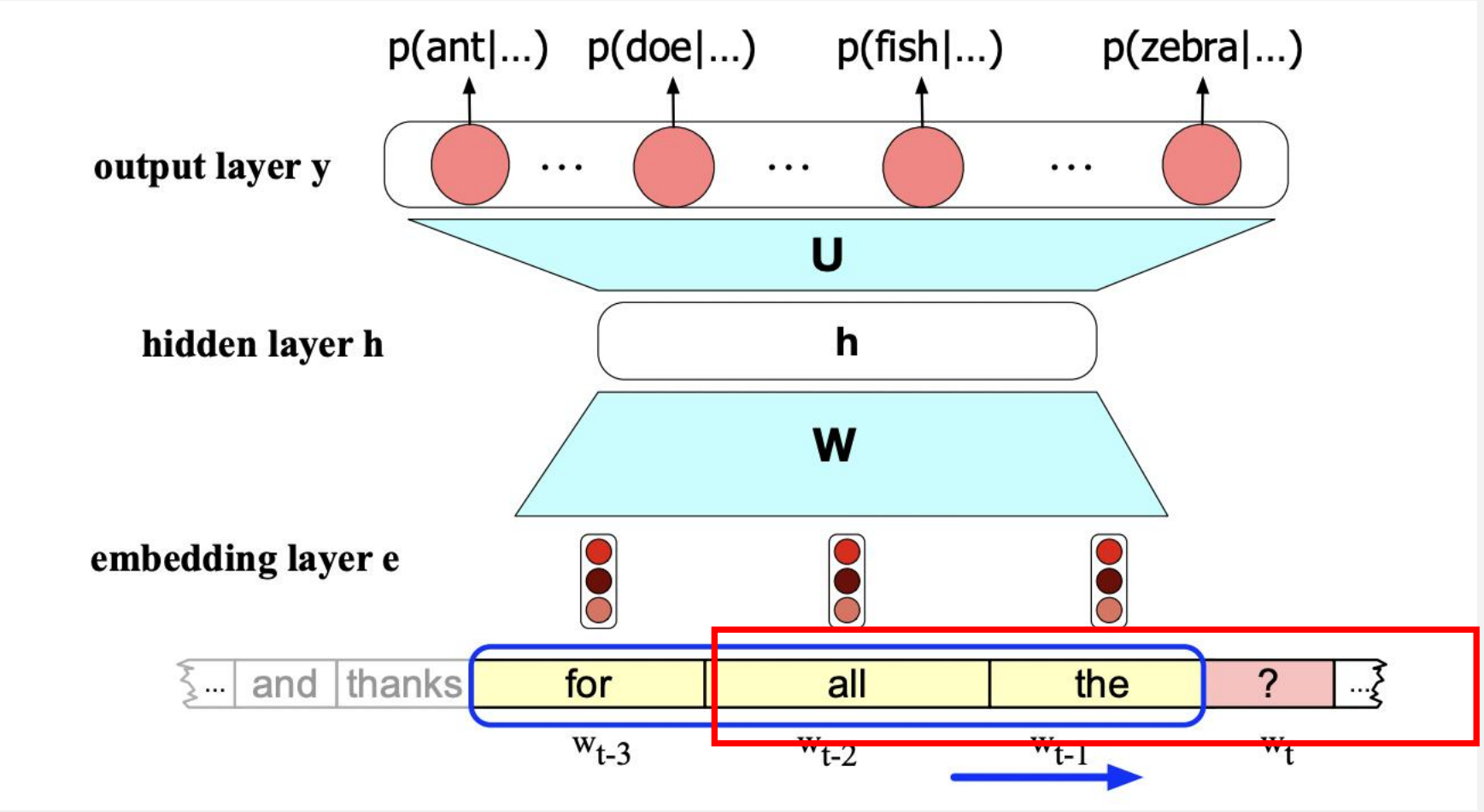

p(ant|...)
p(doe|...)
p(fish|...)
p(zebra|...)
output layer y
U
hidden layer h
h
W
embedding layer e
... and thanks
for
all
the
?
...
w_{t-3}
w_{t-2}
w_{t-1}
w_t

简单的前馈神经网络适用于序列处理？

尽管先前介绍到的前馈神经网络，带有滑动窗口

- 可以处理该窗口内的单词预测任务。
- 但是并不完全令人满意，为什么？
  - 前馈网络相对于N—grams
    - 是从基于单词的计数统计转化为使用词向量(词嵌入)
      - 可以解决相似单词之间的泛化问题
    - 但两者有共同的劣势—有限的上下文
      - 前馈为滑动窗口
      - N—grams为n
  - 滑动窗口的使用使得网络很难学习到系统性模式
    - 比如语法成分，或者有含义的单词组合(短语)

滑动窗口的劣势


p(ant|...)
p(doe|...)
p(fish|...)
p(zebra|...)
output layer y
U
hidden layer h
h
W
embedding layer e
... and thanks for all the ? ...
w_{t-3}
w_{t-2}
w_{t-1}
w_t

本章内容概述

为了应对文本序列的时间属性，并挑战先前的语言模型劣势。

介绍两种深度学习架构：

- RNN循环神经网络(recurrent neural networks)
  - 提出一种新的方式来表示先前文本内容的信息
  - 当前的决策基于先前序列的信息，甚至几百个单词
- Transformer网络
  - 自注意力机制
  - 位置编码
  - 有助于时间表示和长距离单词之间的关联

本章内容概述

这两种架构都提供了处理语言序列属性的功能机制，

这两种模型将应用于

- 语言建模任务、
- 序列建模任务（如词性标记）
- 文本分类任务（如情感分析）。

概率语言模型用于预测序列中的下一个单词

举例：想要预测“Thanks for all the”之后，‘fish’作为下一个单词出现的概率，

需要对下面的概率进行计算

$$P(fish|Thanks\ for\ all\ the)$$

语言模型可以把这样的条件概率分配给每一个可能的单词，

- 即对整个词汇表提供一个概率分布

概率语言模型用于预测序列中的下一个单词

所有单词的条件概率可以与链式法则结合

- 为整个序列分配概率

$$P(w_{1:n}) = \prod_{i=1}^{n} P(w_i|w_{<i})$$

怎样评估一个好的语言模型

模型预测未曾出现在训练集中的单词：

- 可以分配更高的概率给新词(对新单词的出现并不惊讶)

使用困惑度(perplexity)来评估模型好坏：

- 模型 θ 分配给测试数据的逆概率
- 根据测试数据的长度做归一化处理
- 对于一个测试数据$w_{1:n}$，其困惑度为

$$\begin{aligned} \mathrm{PP}_\theta(w_{1:n}) &= P_\theta(w_{1:n})^{-\frac{1}{n}} \\ &= \sqrt[n]{\frac{1}{P_\theta(w_{1:n})}} \end{aligned}$$

困惑度 + 链式法则

困惑度计算

- 衡量语言模型在计算每个新单词概率的优劣
- 结合使用链式法则，扩展测试集概率的计算

$$\mathrm{PP}_\theta(w_{1:n}) = \sqrt[n]{\prod_{i=1}^{n} \frac{1}{P_\theta(w_i|w_{1:n-1})}}$$

Part.02

RNN神经网络

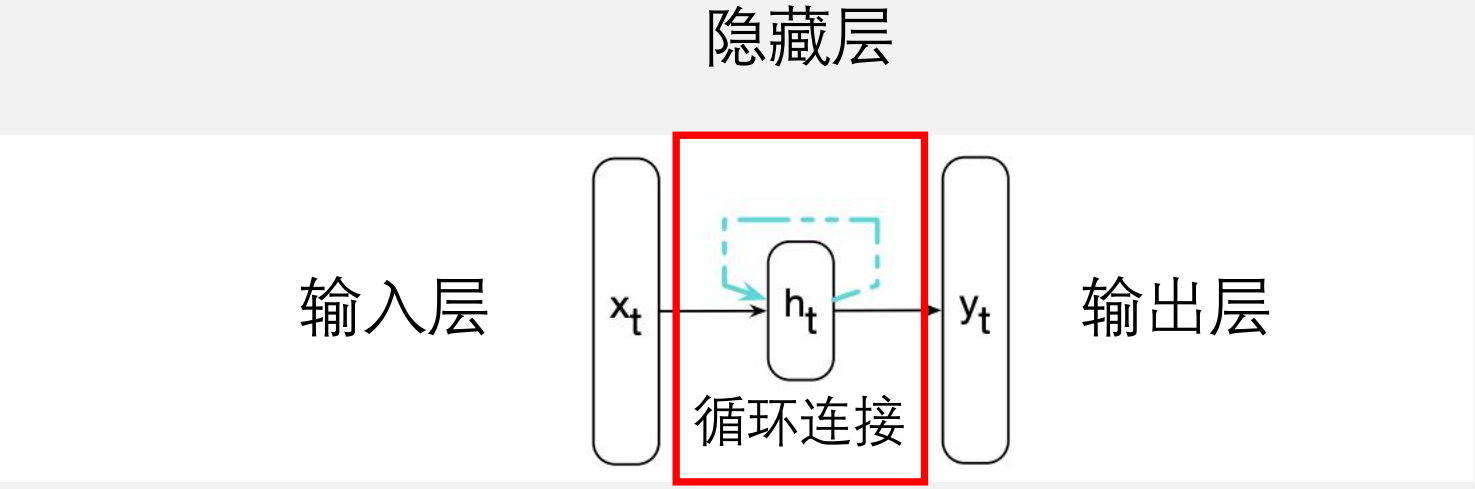
什么是RNN循环神经网络

Recurrent Neural Networks 循环神经网络是指：

- 任何在其网络连接中包含循环的神经网络模型
  - 某一个输出值直接或间接的依赖于先前的输入
  - 功能强大
  - 难以推理和理解
  - 已经被证明其处理语言序列时的有效性。

- 基于RNN的模型包括Elman，LSTM等

简单的RNN神经网络模型



RNN循环神经网络

来自于前一个时间戳的隐藏层以一种上下文或者记忆的形式

- 对先前的序列处理进行编码
- 并将编码的信息提供给后续的决策制定。

注意！

- RNN网络没有对先前上下文的长度进行限制。
- 在前一个隐藏层的编码信息中包含着整个序列的第一个输入的信息。

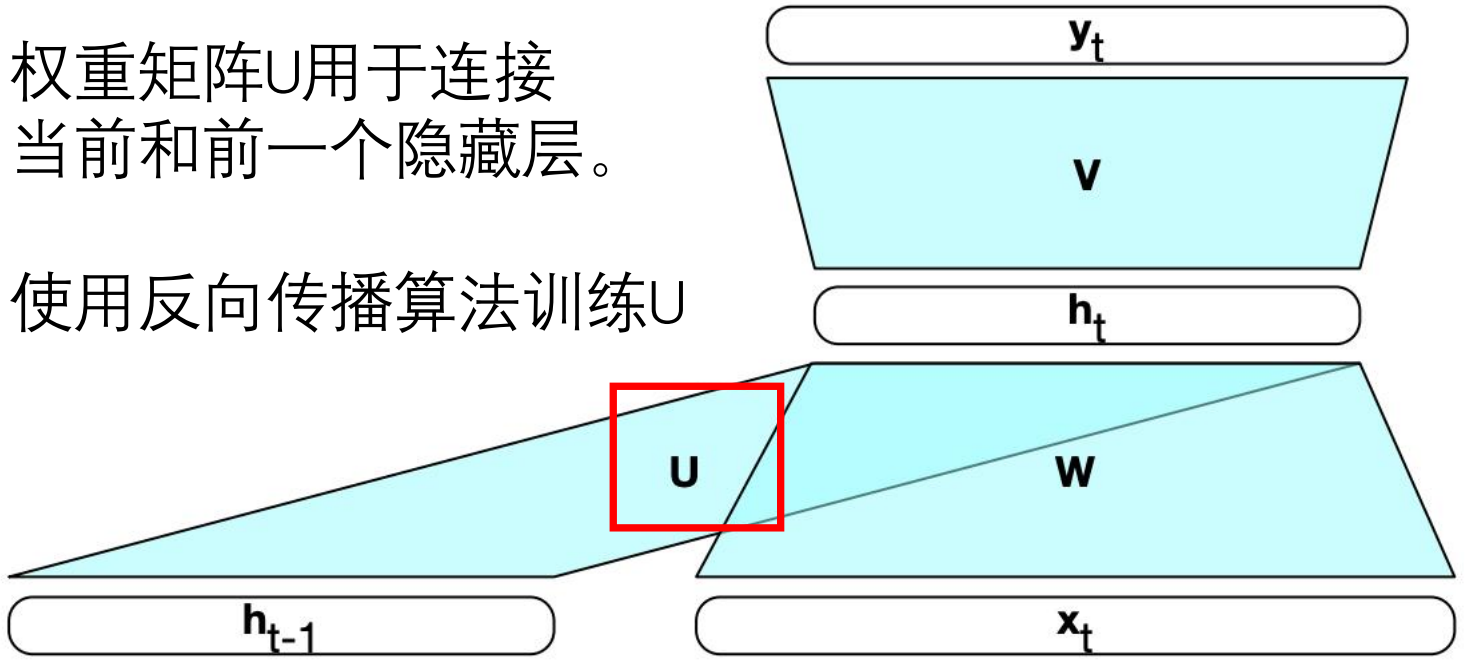
RNN循环神经网络

RNN的时间性质，使得RNN比非循环结构模型更加复杂

但RNN也利用了前馈的机制。

权重矩阵U用于连接
当前和前一个隐藏层。

使用反向传播算法训练U



RNN分析

RNN的前向推理与前馈网络几乎一样

- 将输入序列映射到输出序列

RNN输入到输出的映射过程需要隐藏层的h_t的激活值。

- 首先将输入x_t
  1. **乘以**权重矩阵W
  2. 将结果**加上**第t−1步隐藏层h_{t-1}的加权向量$U*x_{t-1}$
- 然后使用激活函数g对上一步计算的结果进行非线性转换
- 在将结果送入当前隐藏层h_t
- 最后可以使用h_t来正常计算当前的输出向量。

RNN分析

输入层，隐藏层，输出层的维度分别为d_{in}，d_h，d_{out}

$$\mathbf{h}_t = g(\mathbf{U}\mathbf{h}_{t-1} + \mathbf{W}\mathbf{x}_t)$$
$$\mathbf{y}_t = f(\mathbf{V}\mathbf{h}_t)$$

$W \in \mathbb{R}^{d_h \times d_{in}}$　　$\mathbf{U} \in \mathbb{R}^{d_h \times d_h}$　　$\mathbf{V} \in \mathbb{R}^{d_{out} \times d_h}$

$$\mathbf{y}_t = \text{softmax}(\mathbf{V}\mathbf{h}_t)$$

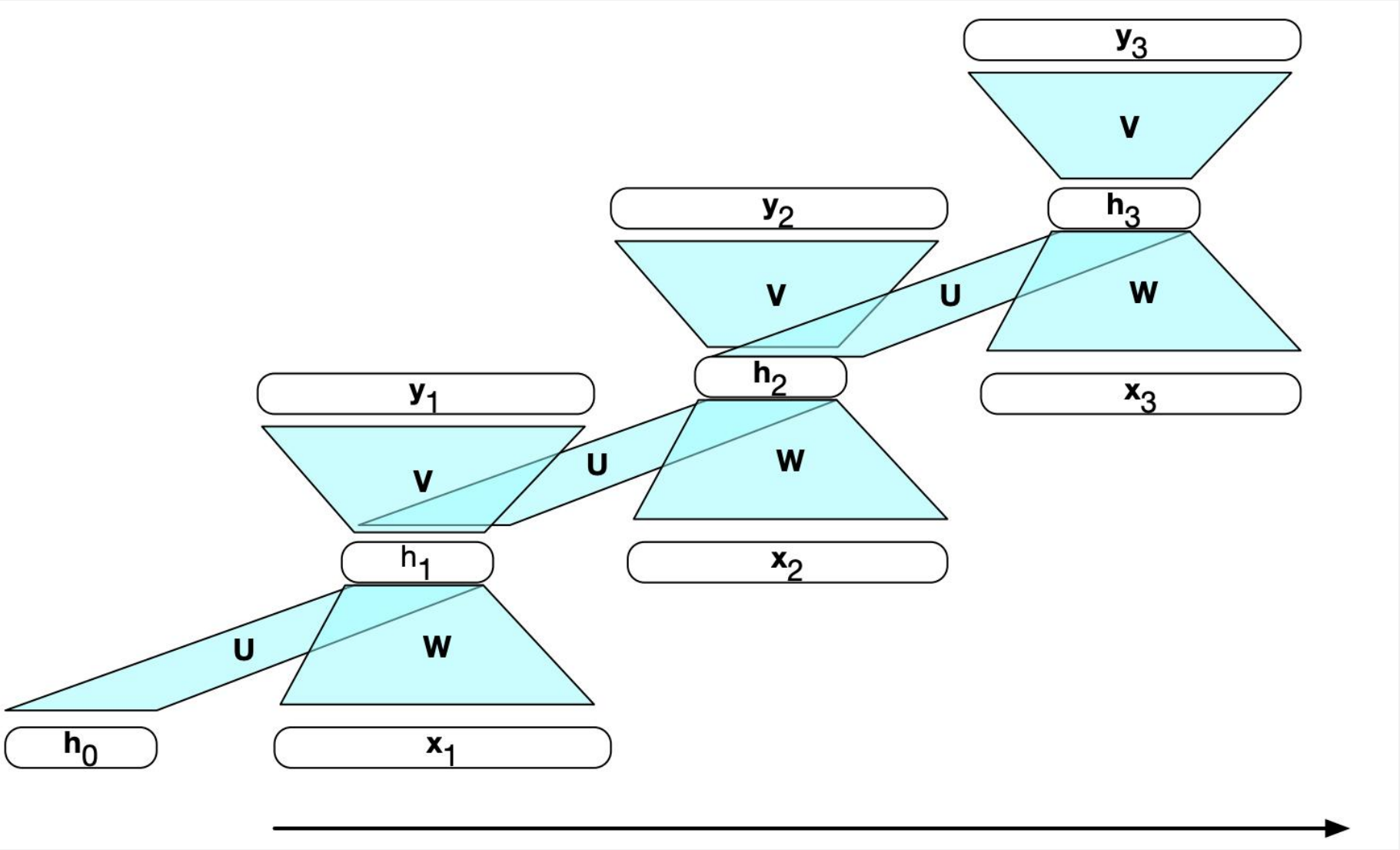
RNN分析

在时间t上的计算需要时间t−1时间上隐藏层的值
需要一个增量式推理算法来处理序列的开始到结束。

function FORWARDRNN(**x**, *network*) **returns** output sequence **y**

$\mathbf{h}^0 \leftarrow 0$
for $i \leftarrow 1$ **to** LENGTH(**x**) **do**
  $\mathbf{h}_i \leftarrow g(\mathbf{U}\mathbf{h}_{i-1} + \mathbf{W}\mathbf{x}_i)$
  $\mathbf{y}_i \leftarrow f(\mathbf{V}\mathbf{h}_i)$
return y

RNN网络展开



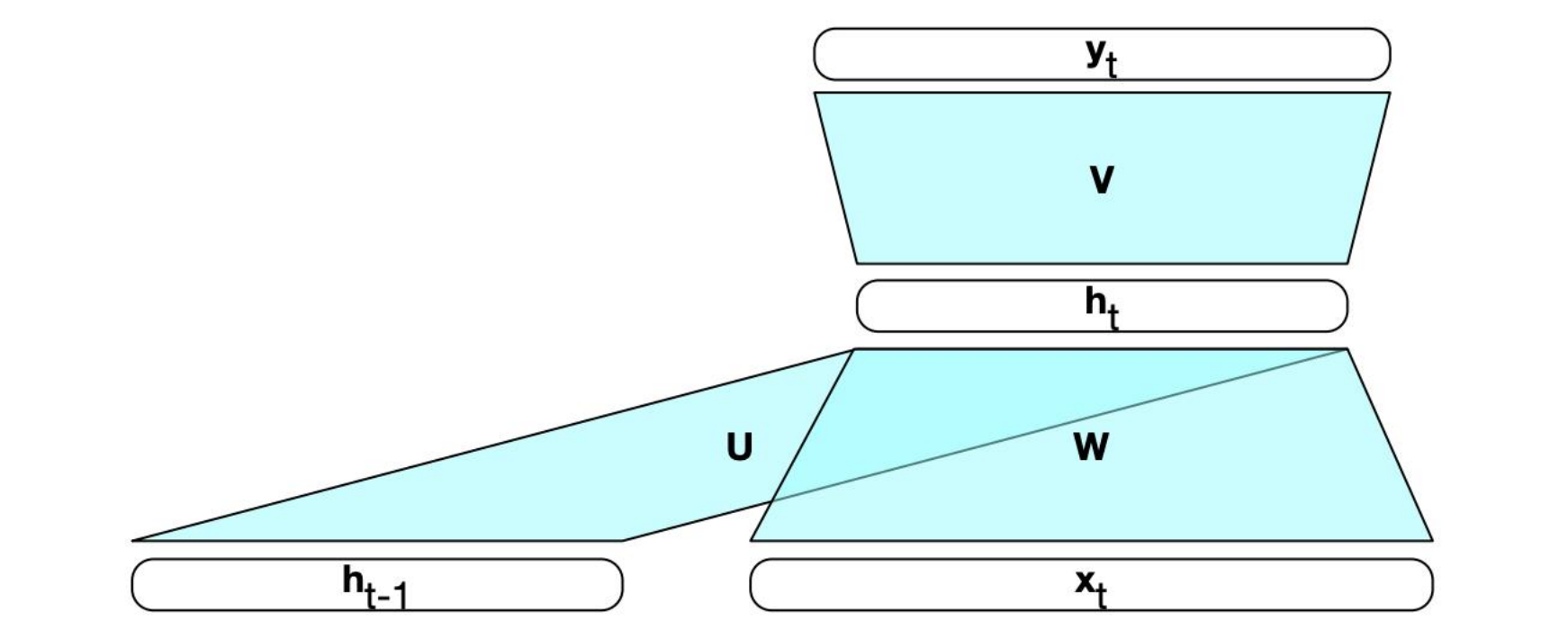
U，V，W跨时间共享

每一步都会计算h_i， y

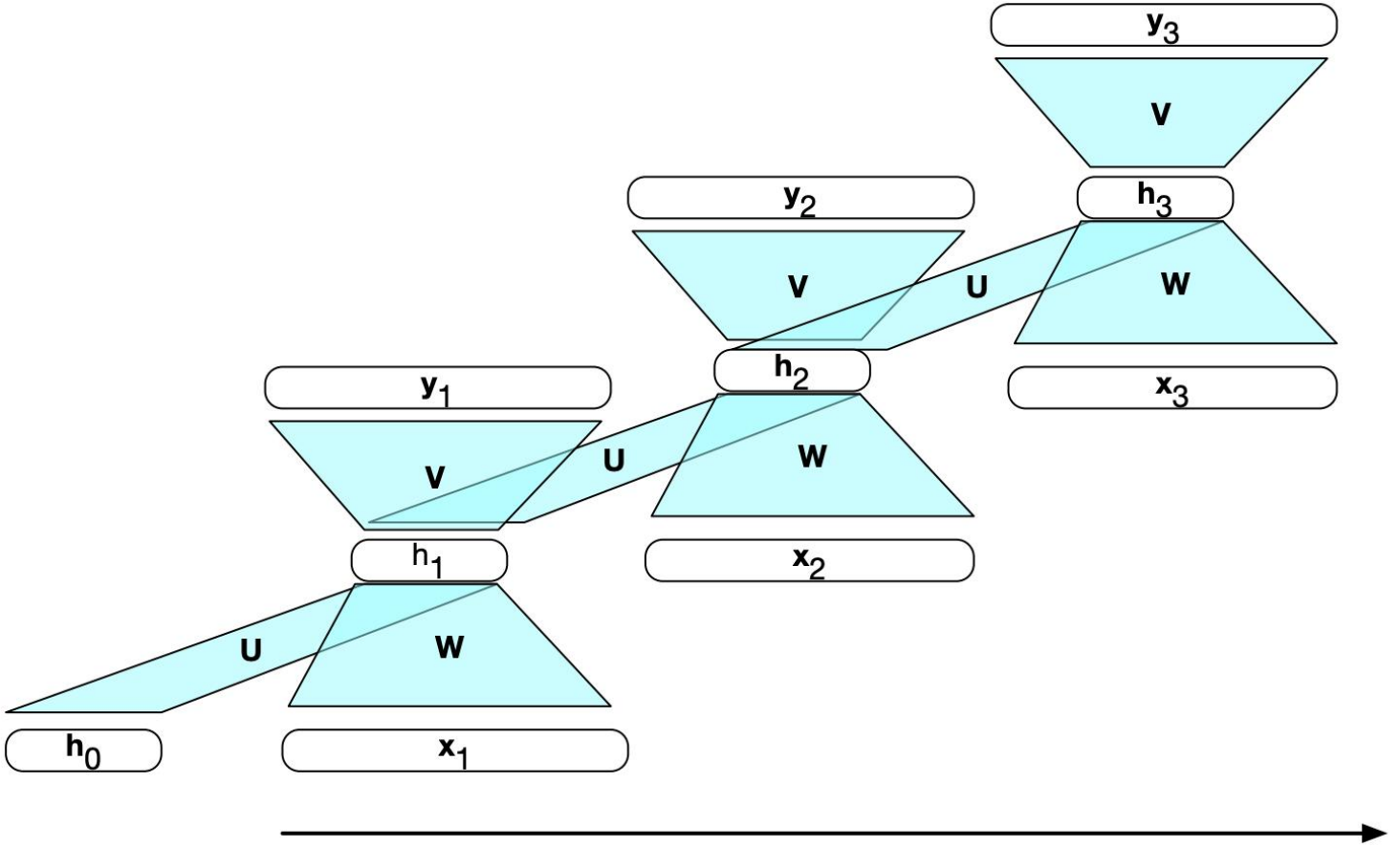
RNN模型训练

与前馈网络训练一样，需要使用训练集，损失函数和反向传播算法来计算梯度

- 梯度可以用于权重矩阵的调整和更新
- 有三个权重矩阵(U,V,W)



RNN模型训练



计算时间t时输出的损失函数需要时间t−1的隐藏层信息。

时间t的隐藏层会影响

- 当前输出
- 时间t+1的隐藏层、输出和损失
- 因此需要评估h_t产生的误差

RNN模型训练

需要对反向传播进行调整来应对这种情况，需要分两步：

1．向前推理，计算h_t，yt，在每一时间步上累积损失，
并且保存隐藏层的值以供下一时间步上使用

2．对序列反向处理，计算所需的梯度，
并且保存误差项，以便在每次后退时用于隐藏层

这种方法称为随时间反向传播(Backpropagation Through Time)

RNN模型训练—形式化

W 是输入层到当前隐藏层权重
U是从前一隐藏层到当前隐藏层权重
V是从当前隐藏层到输出层权重
x_i 当前输入向量
g 激活函数

$z_i = Wx_i$ 加权向量
$a_i = g(z_i)$ 激活后加权向量，作为隐藏层输入
$z_{i+1} = Ua_i$
$a_{i+1} = g(z_{i+1})$
$y_i = \text{softmax}(Va_{i+1})$

RNN语言模型

有一个文本序列X包含N个单词向量。 $\mathbf{X} = [\mathbf{x}_1; ...; \mathbf{x}_t; ...; \mathbf{x}_N]$
每一个单词x_t用one-hot encoding去表示。向量维度为|V|，V是词汇表大小。
通过词向量矩阵，可以获取单词x_t对应的词向量。

输出预测 y 是一个|V|维向量，表示所有词汇表上单词出现在当前位置的概率分布。

在时间t上：

$$\begin{aligned} \mathbf{e}_t &= \mathbf{E}\mathbf{x}_t \\ \mathbf{h}_t &= g(\mathbf{U}\mathbf{h}_{t-1} + \mathbf{W}\mathbf{e}_t) \\ \mathbf{y}_t &= \text{softmax}(\mathbf{V}\mathbf{h}_t) \end{aligned}$$

W 是输入层到当前隐藏层权重
U是从前一隐藏层到当前隐藏层权重
V是从当前隐藏层到输出层权重
E是词嵌入矩阵

RNN语言模型

在当前预测中，词汇表中的单词i出现在下一个位置的概率可以表达为：

$$P(w_{t+1} = i | w_1, \ldots, w_t) = \mathbf{y}_t[i]$$

$y_t[i]$是输出向量y_t的第i个元素。

那么整个序列的概率，可以表达为每一位置单词出现概率的乘积。

$$\begin{aligned} P(w_{1:n}) &= \prod_{i=1}^{n} P(w_i | w_{1:i-1}) \\ &= \prod_{i=1}^{n} \mathbf{y}_i[w_i] \end{aligned}$$

多分类损失函数

二分类损失函数

$$L_{CE}(\hat{y}, y) = -\log p(y|x) = -[y \log \hat{y} + (1-y)\log(1-\hat{y})]$$

多分类损失函数

$$L_{CE}(\hat{\mathbf{y}}, \mathbf{y}) = -\sum_{k=1}^{K} \mathbf{y}_k \log \hat{\mathbf{y}}_k$$

使用损失函数计算预测分布与真实分布的误差

交叉熵(cross entopy)

$$L_{CE} = -\sum_{w \in V} \mathbf{y}_t[w] \log \hat{\mathbf{y}}_t[w]$$

交叉熵损失是由预测模型分配给正确下一个单词的概率决定的。
那么在时间t，交叉熵损失可以表示为：

$$L_{CE}(\hat{\mathbf{y}}_t, \mathbf{y}_t) = -\log \hat{\mathbf{y}}_t[w_{t+1}]$$

为什么？

教师强制

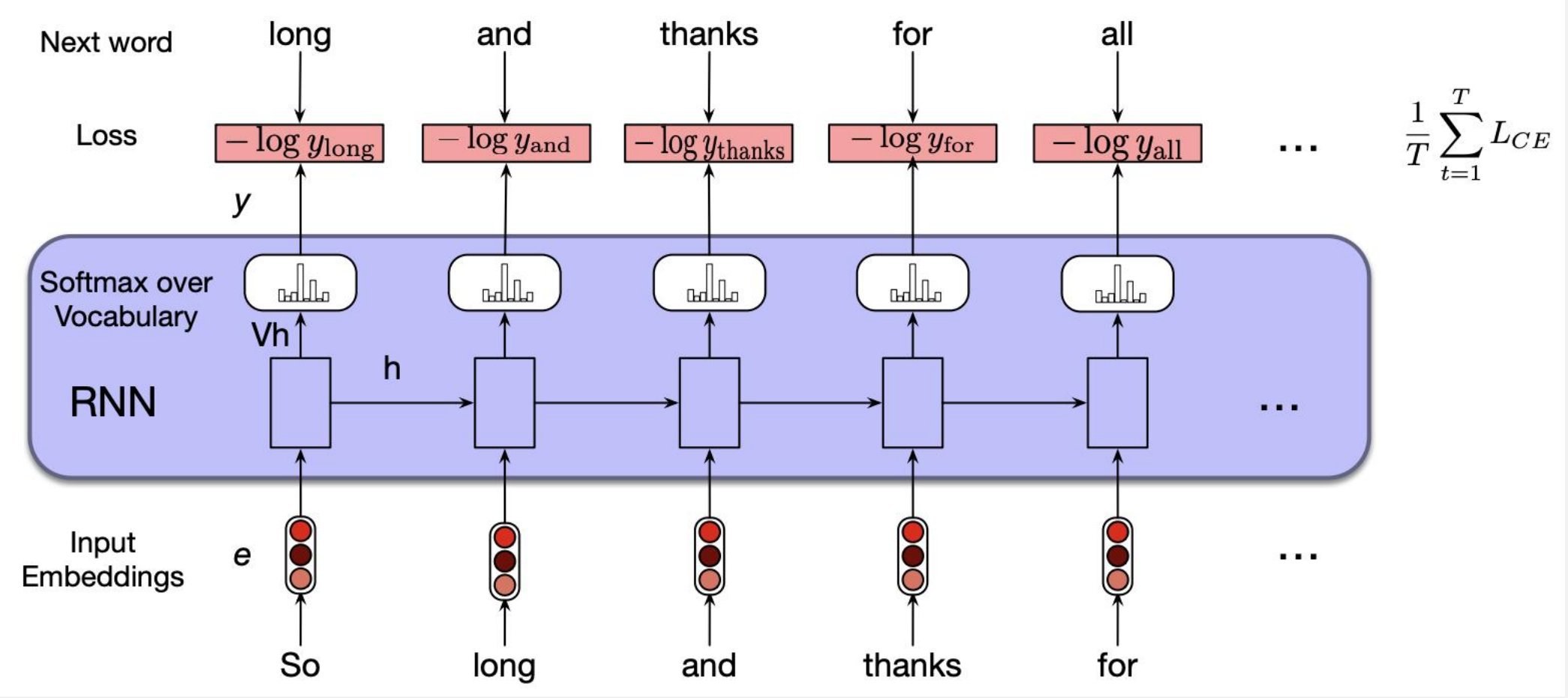
然而在每一步预测的单词都有可能是错误的该如何解决？

在预测x_{t+1}时，使用真实单词序列$w_{1:t}$，而不使用预测出来的单词序列

这种方法称为教师强制

使用RNN作为语言模型

所有项交叉熵的平均值用以估计损失
(预测值与目标值的差值)



使用RNN作为语言模型

输入向量矩阵E 与 可以作为softmax函数输入的最终层矩阵V很相似

- E的每一行对于的词汇表相应单词的词向量，
- 这些词向量是在训练过程中计算得到的。

矩阵E的shape为|V| x d_h，

- 因为这些向量的维度等于隐藏层神经元的个数(d_h)

$$\begin{aligned} \mathbf{e}_t &= \mathbf{E}\mathbf{x}_t \\ \mathbf{h}_t &= g(\mathbf{U}\mathbf{h}_{t-1} + \mathbf{W}\mathbf{e}_t) \\ \mathbf{y}_t &= \text{softmax}(\mathbf{E}^{intercal}\mathbf{h}_t) \end{aligned}$$

通过计算Vh，最终层矩阵V提供了一种对词汇表中

- 每一个单词的可能性进行评分的方法
- 也就是说计算得到的词向量具有计算得到的词义和单词功能的性质。

权重共享

E中包含了v个词向量

V中学习到了v个词向量

是否需要同时保留这两种权重矩阵？

$$
\begin{aligned}
\mathbf{e}_t &= \mathbf{E}\mathbf{x}_t \\
\mathbf{h}_t &= g(\mathbf{U}\mathbf{h}_{t-1} + \mathbf{W}\mathbf{e}_t) \\
\mathbf{y}_t &= \text{softmax}(\mathbf{E}^{intercal}\mathbf{h}_t)
\end{aligned}
$$

可以提升模型的预测性能
也可以减少需要训练的参数量

RNN用于其他NLP任务

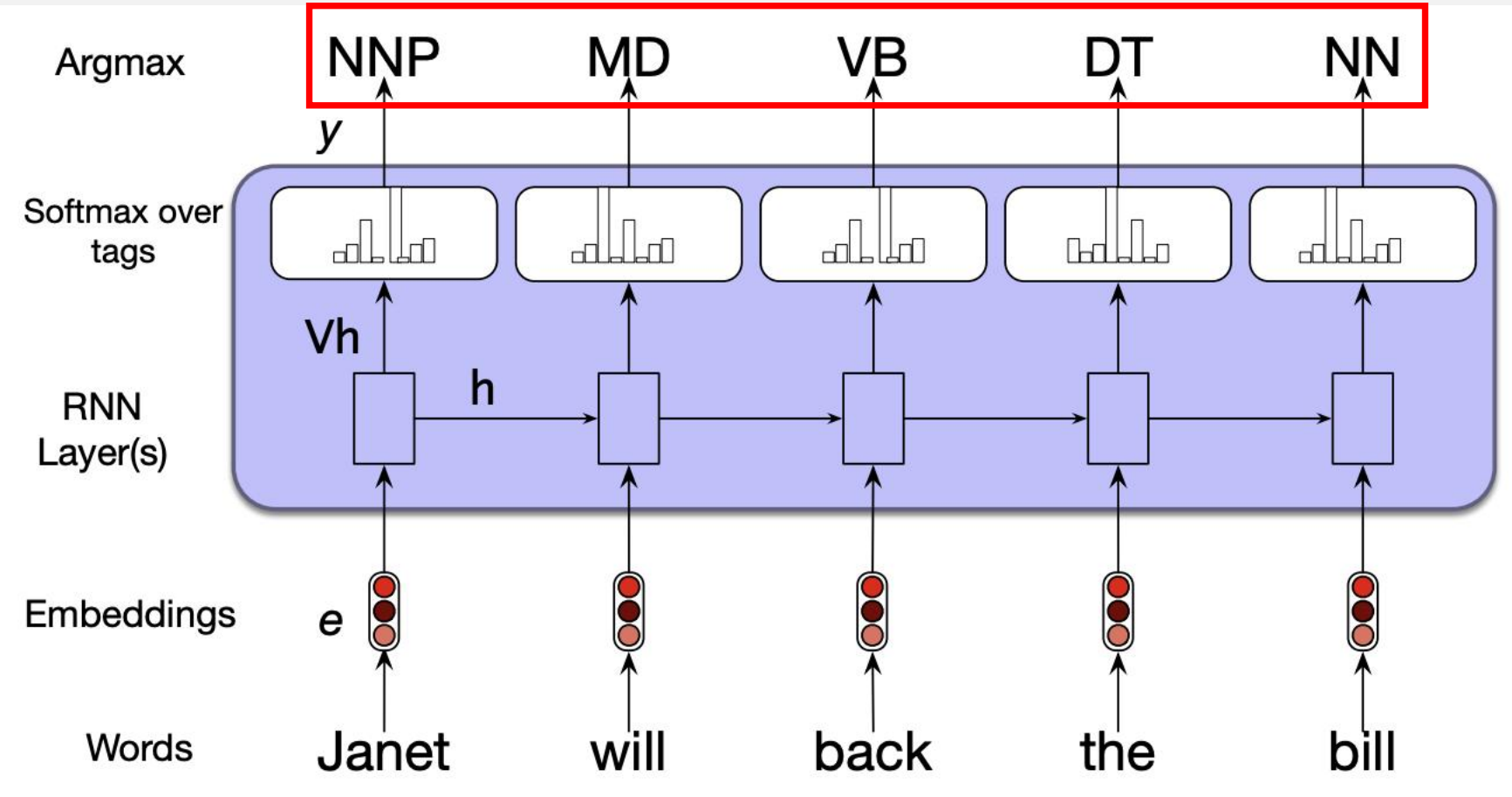
序列标注(sequence labelling)

序列分类(sequence classification)

文本生成(text generation)

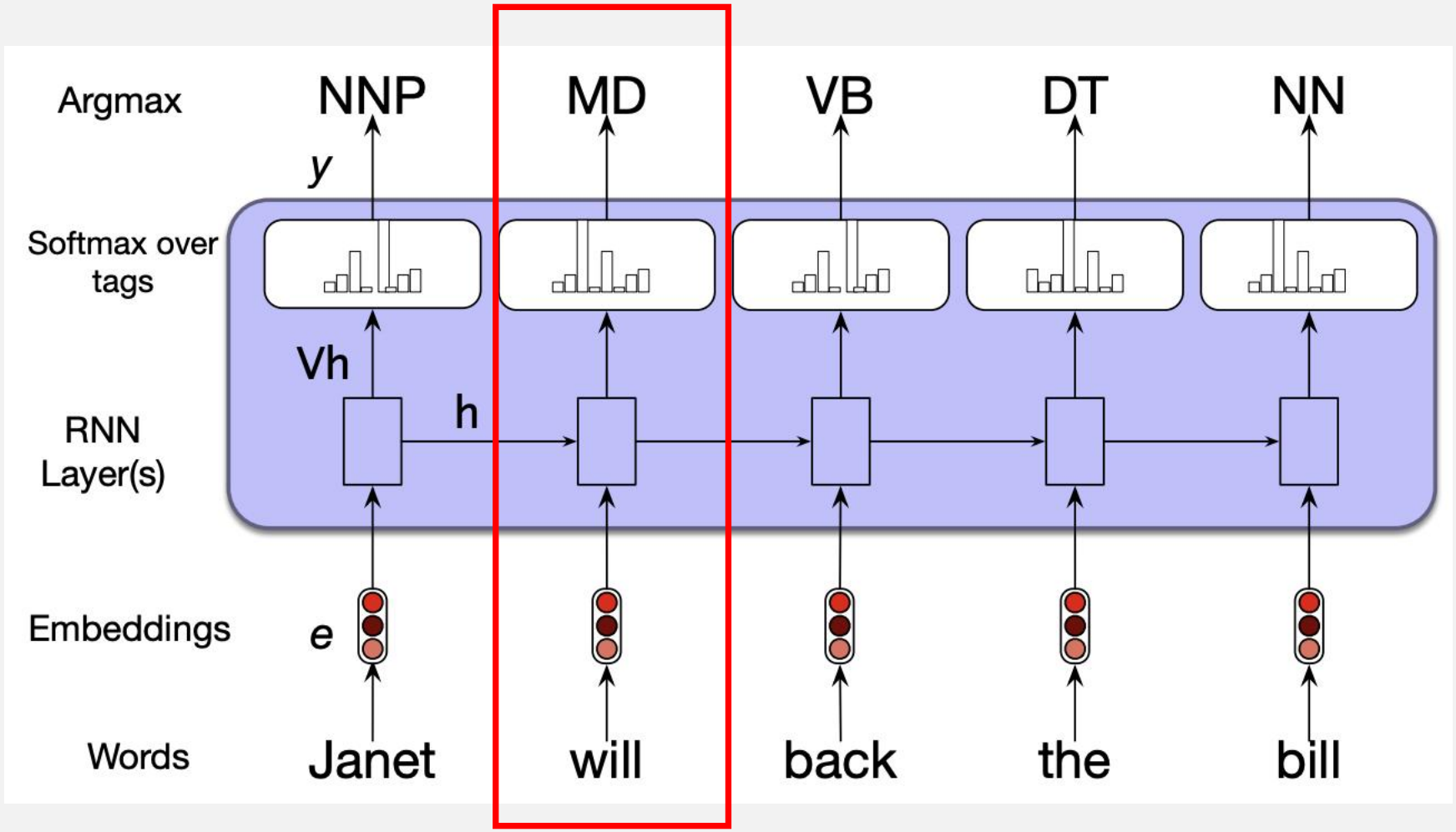
序列标注(sequence labelling)

带上下文信息的多分类任务



分类项个数为标签集的大小

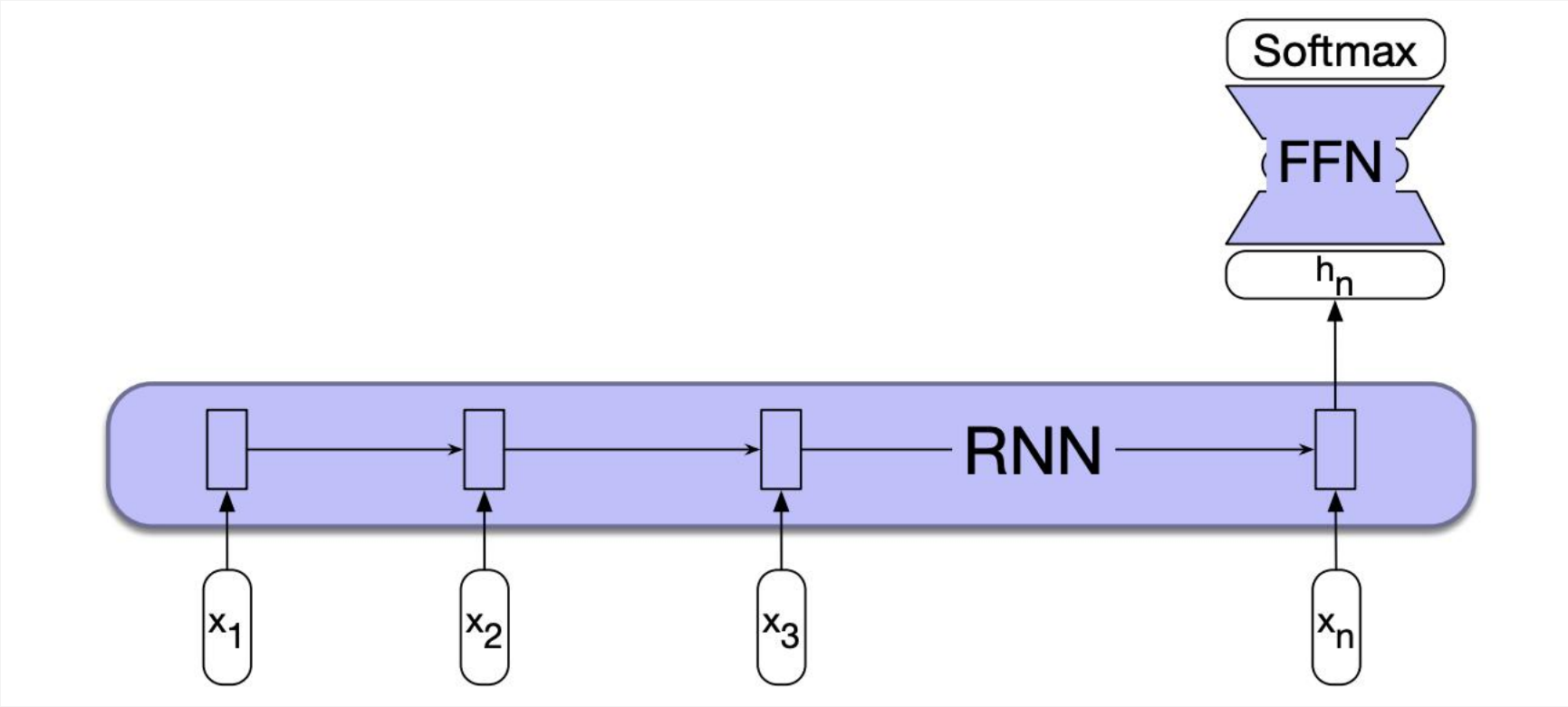
序列标记(sequence labelling)



$$\frac{1}{T}\sum_{t=1}^{T} L_{CE}$$

T个时间步

序列分类



情感分类(正面/负面)

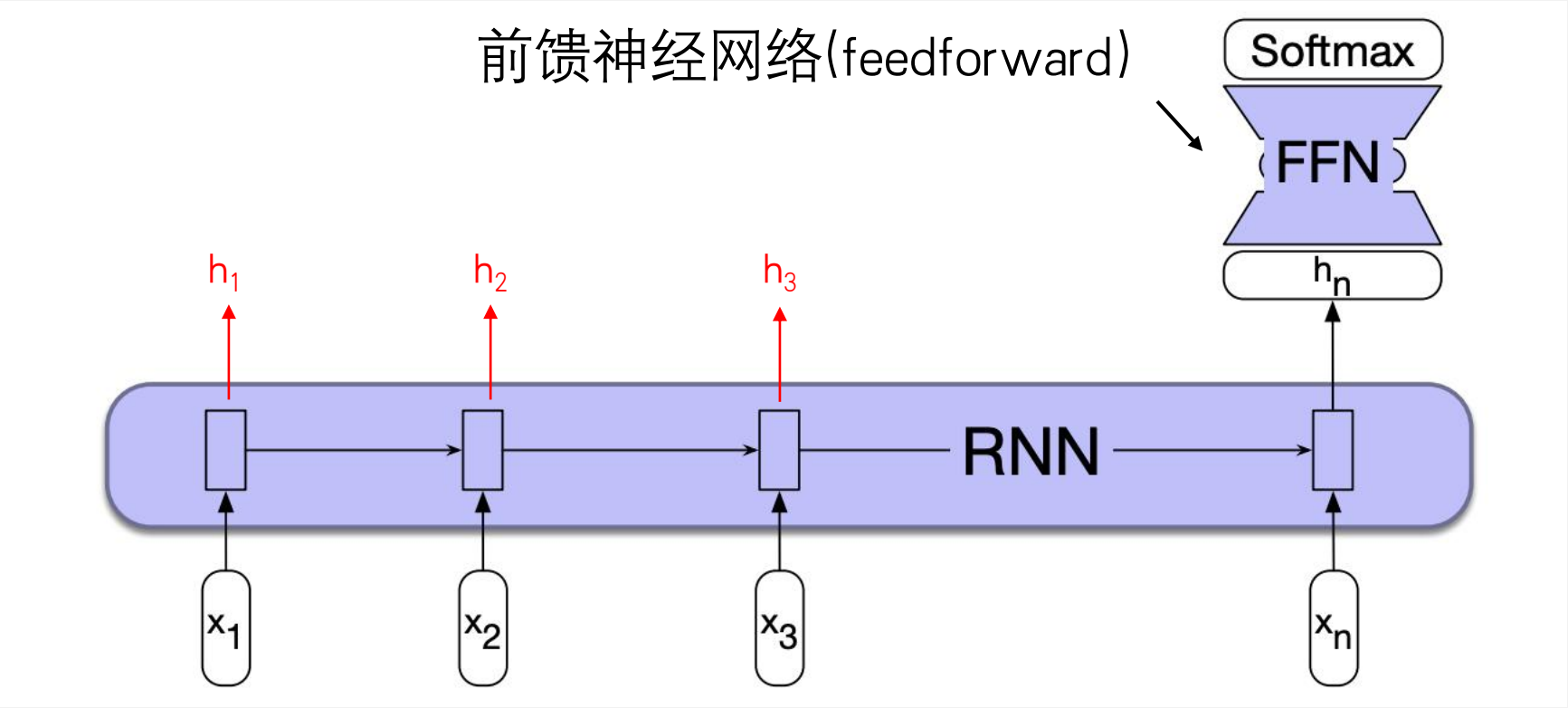
主题分类

垃圾邮件检测

等等

序列分类

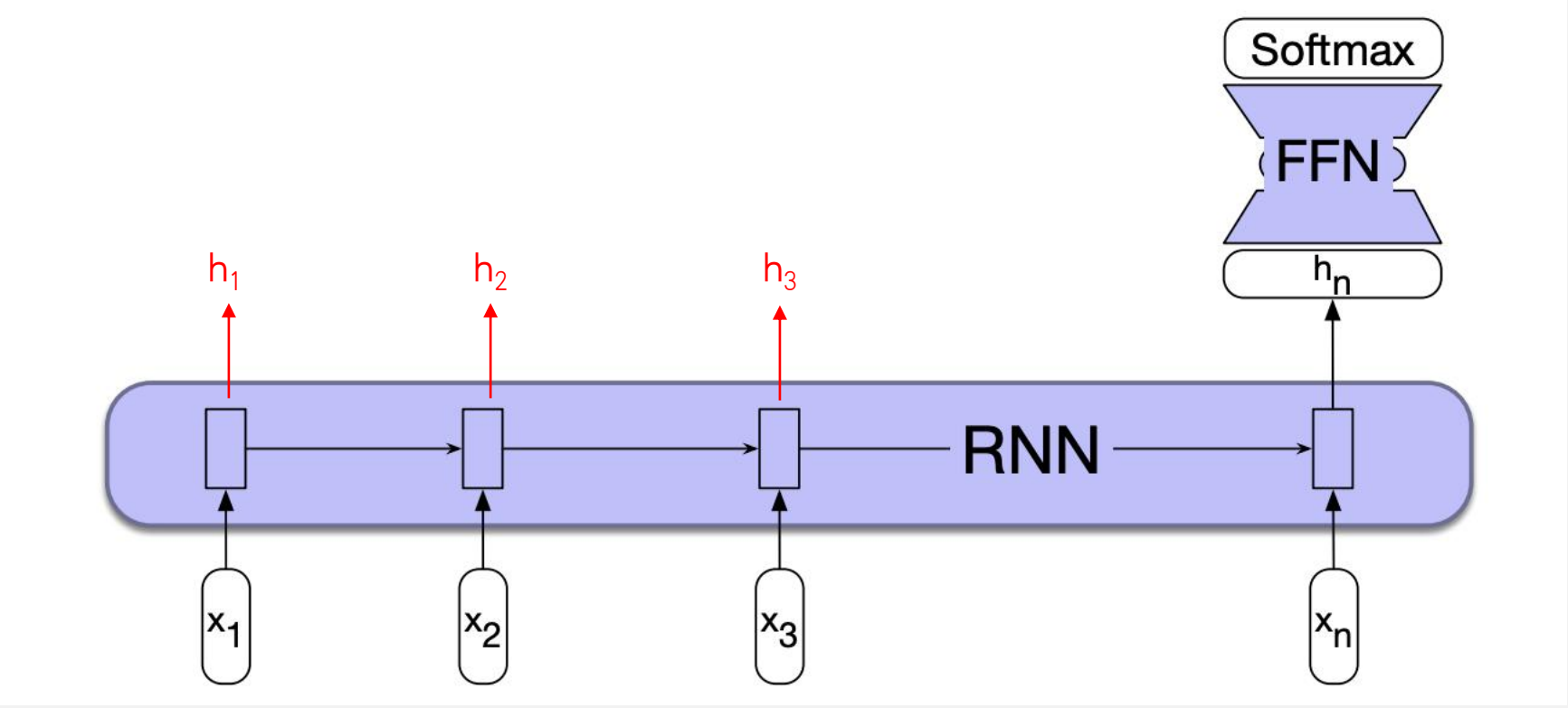
通过softmax计算所有分类项的可能性



预测值与真实值做交叉熵计算损失

将计算得到的损失逐层反向传播到所有权重矩阵（W,U,V），并将其元素进行逐个计算修改。

序列分类

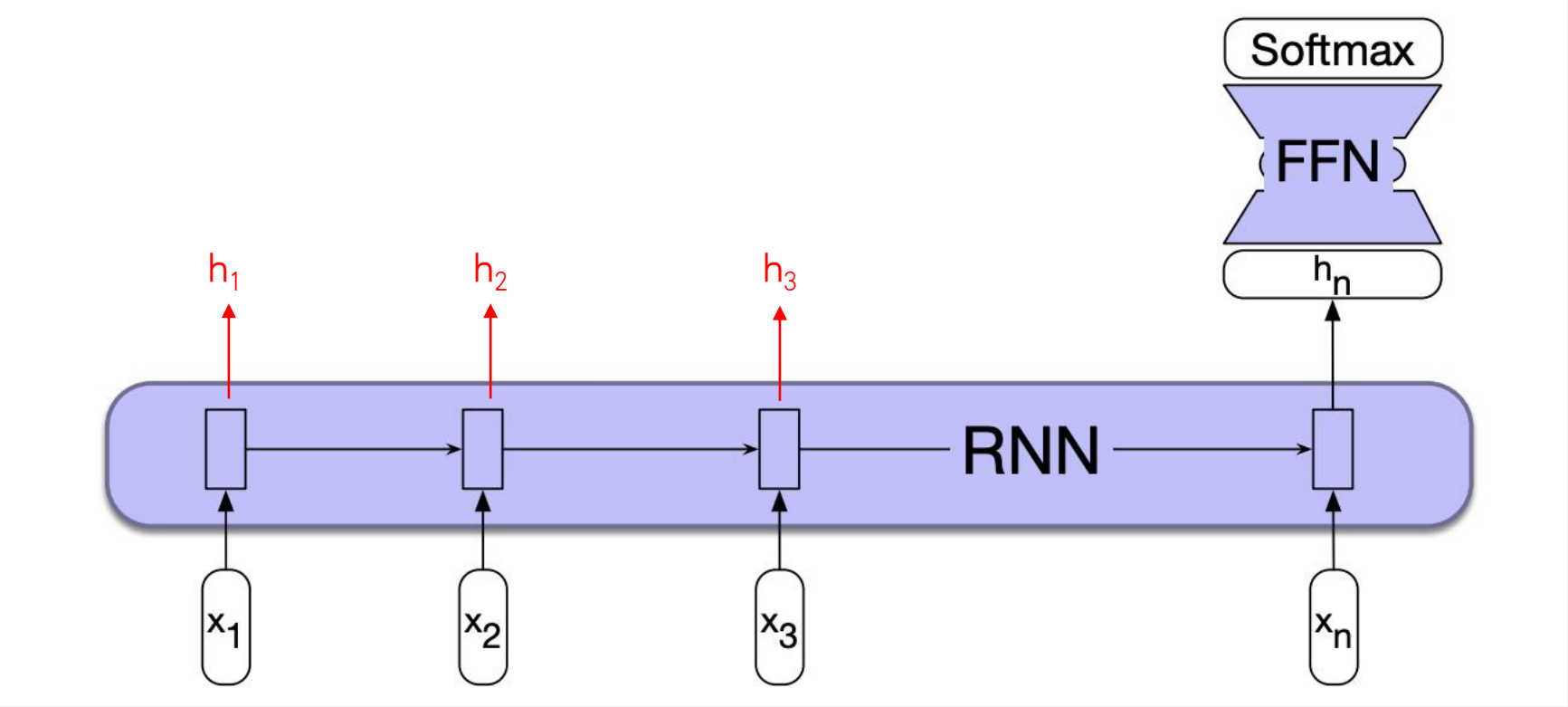


池化函数

$$\mathbf{h}_{mean} = \frac{1}{n}\sum_{i=1}^{n}\mathbf{h}_i$$

对所有时间步产生的隐藏层状态，求平均

序列分类



在所有时间步上的隐藏层向量中h_i的第K个元素最大值

基于RNN文本生成

问答系统(question answering)

机器翻译(machine translation)

文本摘要(text summarization)

对话系统(conversational dialogue)

…

等等需要自动生成文本的任务。

回顾N-grams 文本生成

1. 首先根据单词的合适程度，随机选择一个单词作为文本开头，

2. 然后根据已选择的单词，以及N-grams语法生成后续的单词，

3. 直到达到预设的文本长度，或者直到生成预设的序列结束词(token)。

自回归生成(autoregressive generation)

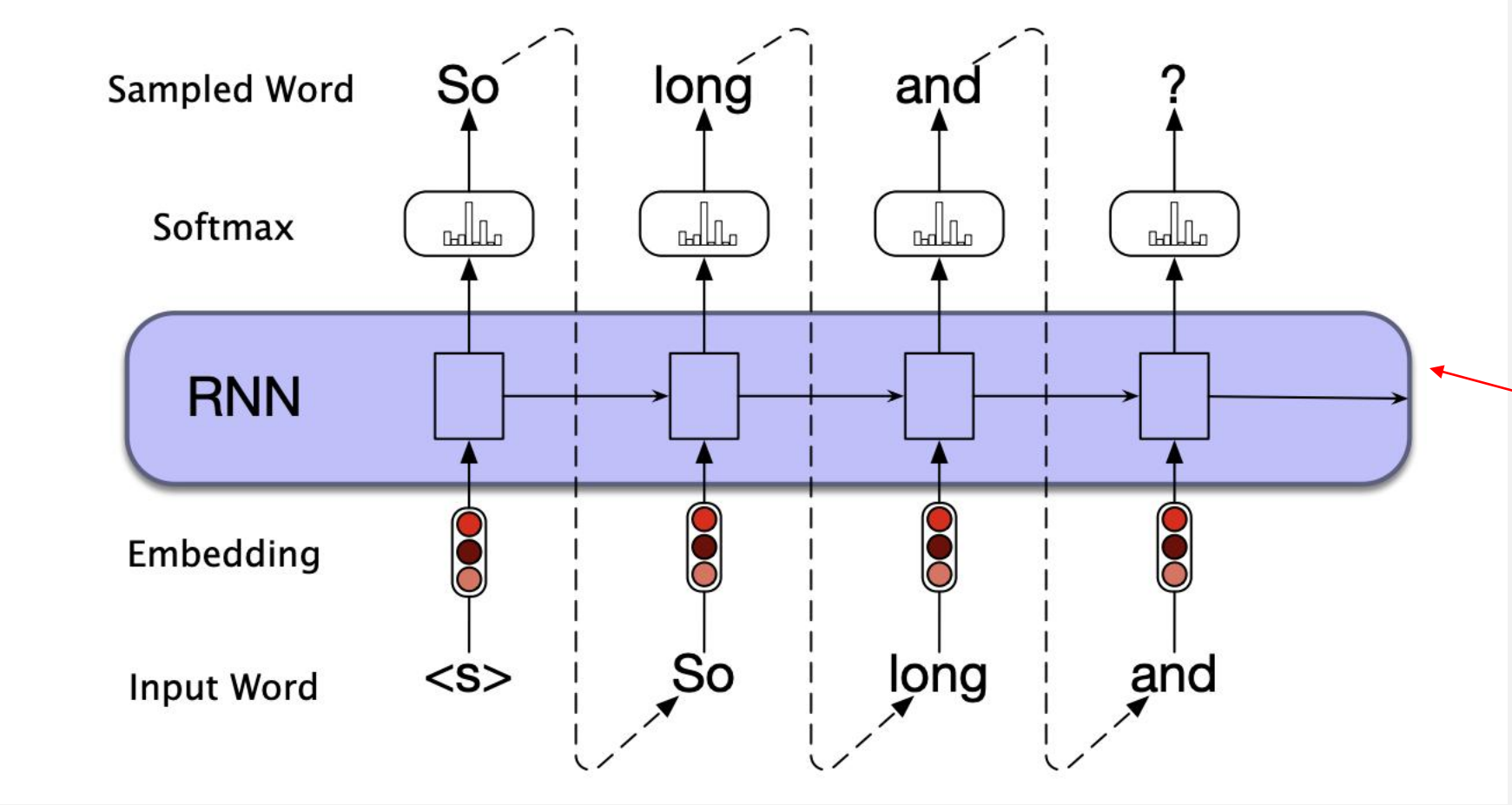
1. 使用<s>作为句子第一个输入的token，

2. 基于当前的输入<s>，从t_0时刻的softmax分布中采样一个单词，

3. 使用第一个单词向量作为下一个时间步神经网络的输入，
   - 并且以同样的方式生成下一个单词，

4. 继续生成，直到生成句尾标注</s>，或者达到预设长度。

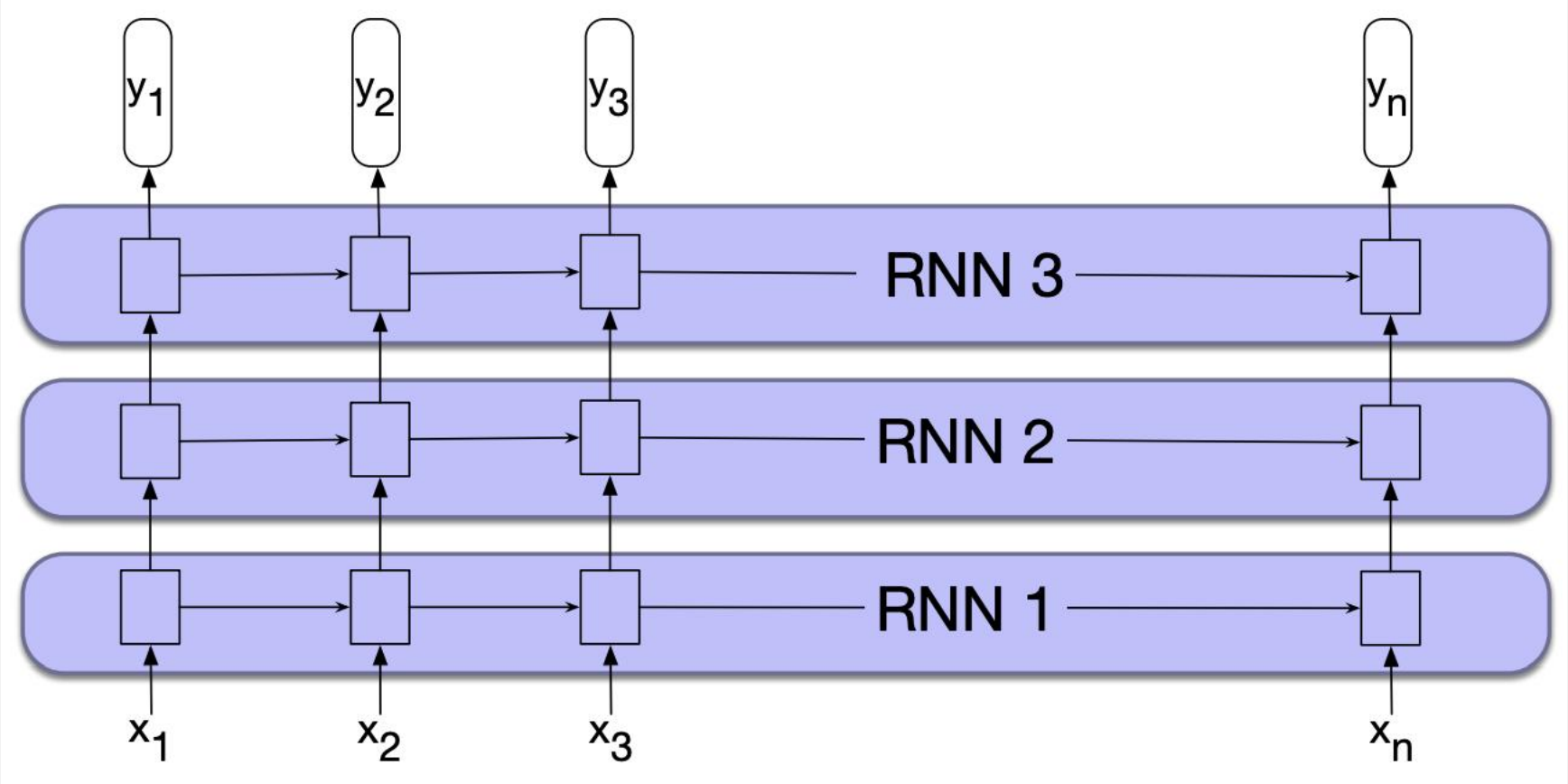
自回归生成(autoregressive generation)



隐藏层和循环连接

t_0 t_1 t_2 t_3 ……

堆叠(Stacked) RNNs



双向(Bidirectional）RNNs

序列是有方向的，之前介绍的RNN中，ht携带的是xt左侧的信息

$$\mathbf{h}_t^f = RNN_{forward}(\mathbf{x}_1, \ldots, \mathbf{x}_t)$$

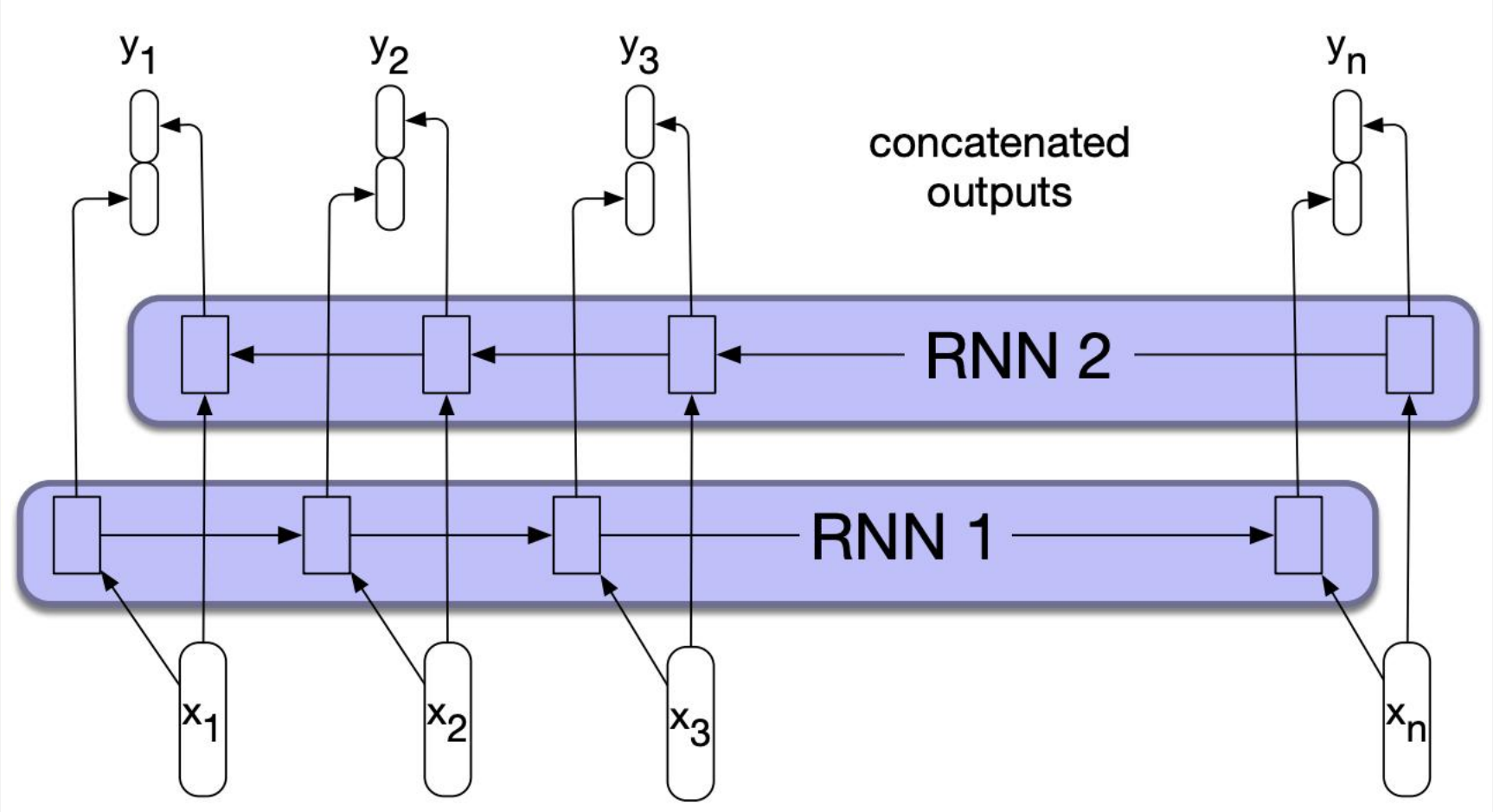
为了获取当前输入右侧的信息，可以将序列反转，再次训练一个RNN模型

$$\mathbf{h}_t^b = RNN_{backward}(\mathbf{x}_t, \ldots \mathbf{x}_n)$$

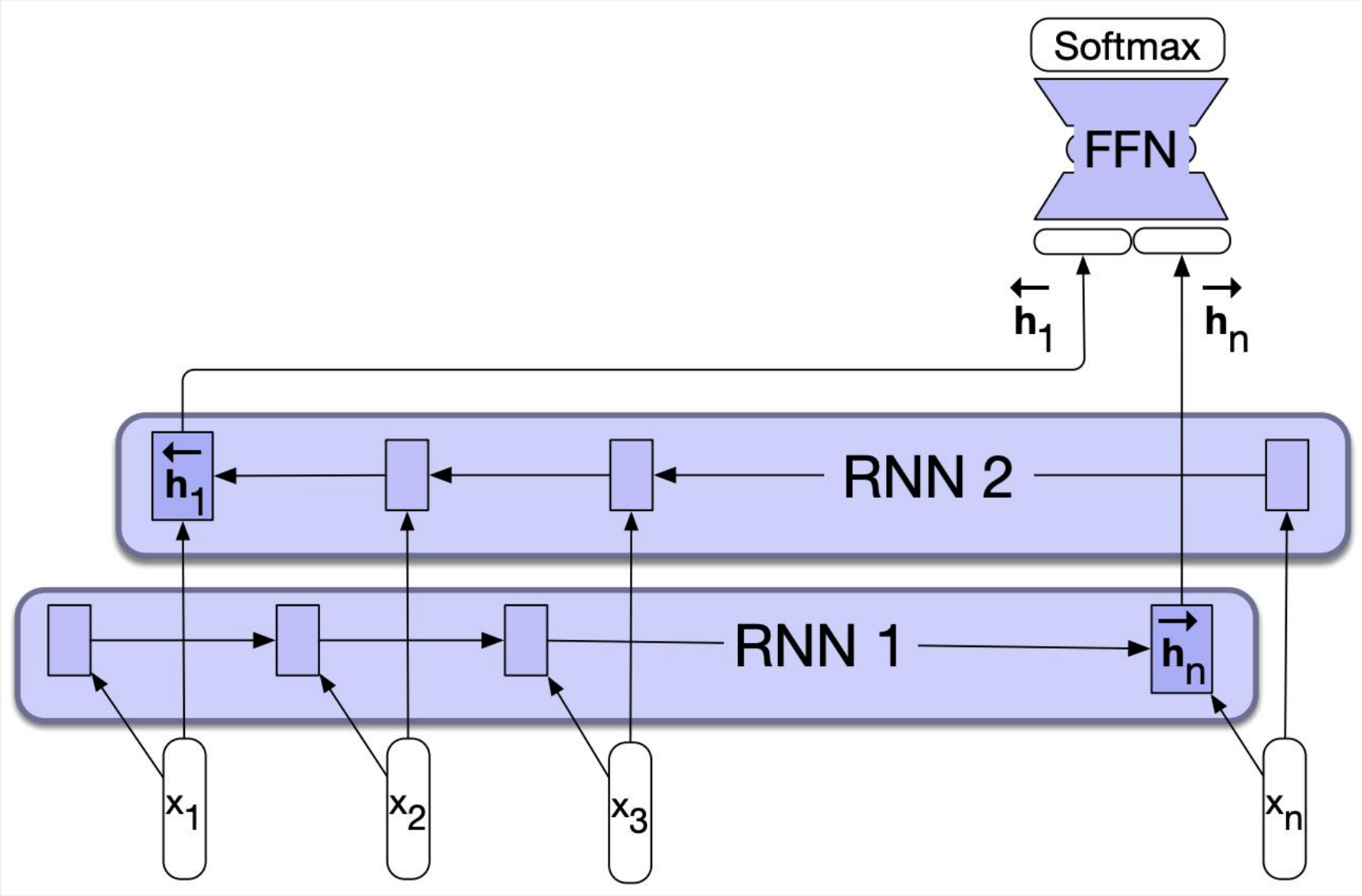
将上述两种训练的隐藏层向量结合为一个新的向量，使用 $\oplus$ 表示连接关系

$$\begin{aligned} \mathbf{h}_t &= [\mathbf{h}_t^f ; \mathbf{h}_t^b] \\ &= \mathbf{h}_t^f \oplus \mathbf{h}_t^b \end{aligned}$$

双向(Bidirectional）RNNs



双向(Bidirectional）RNNs



Simple RNN for IMDB Review

```
from keras.datasets import imdb
import numpy as np


(train_data, train_labels), (test_data, test_labels) = imdb.load_data(num_words=500)
x_train = vectorize_sequences(train_data)
x_test = vectorize_sequences(test_data)
y_train = np.asarray(train_labels).astype('float32')
y_test = np.asarray(test_labels).astype('float32')

x_val = x_train[:3000]
x_train = x_train[3000:]
y_val = y_train[:3000]
y_train = y_train[3000:]
```

Simple RNN for IMDB Review

```
def vectorize_sequences(sequences, dimension=500):
    results = np.zeros((len(sequences), dimension))
    for i, sequence in enumerate(sequences):
        results[i, sequence] = 1.
    return results
```

Simple RNN for IMDB Review

```
from keras.models import Sequential
from keras.layers import  SimpleRNN, Embedding, Dense

vocabulary = 10000
embedding_dim = 32
word_num = 500
state_dim = 32

model = Sequential()
model.add(Embedding(vocabulary,embedding_dim,input_length=word_num))
model.add(SimpleRNN(state_dim,return_sequences=False))
model.add(Dense(1,activation='sigmoid’))

model.summary()
```

Simple RNN for IMDB Review

```
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 embedding (Embedding)        (None, 500, 32)           320000

 simple_rnn (SimpleRNN)       (None, 32)                2080

 dense (Dense)                (None, 1)                 33

=================================================================
Total params: 322,113
Trainable params: 322,113
Non-trainable params: 0
```

Simple RNN for IMDB Review

```
from keras import optimizers

epochs = 3
model.compile(optimizer=optimizers.RMSprop(learning_rate=0.001),
              loss='binary_crossentropy', metrics=['acc'])
history = model.fit(x_train, y_train, epochs=epochs, batch_size=512, validation_data=(x_val,
y_val))
```

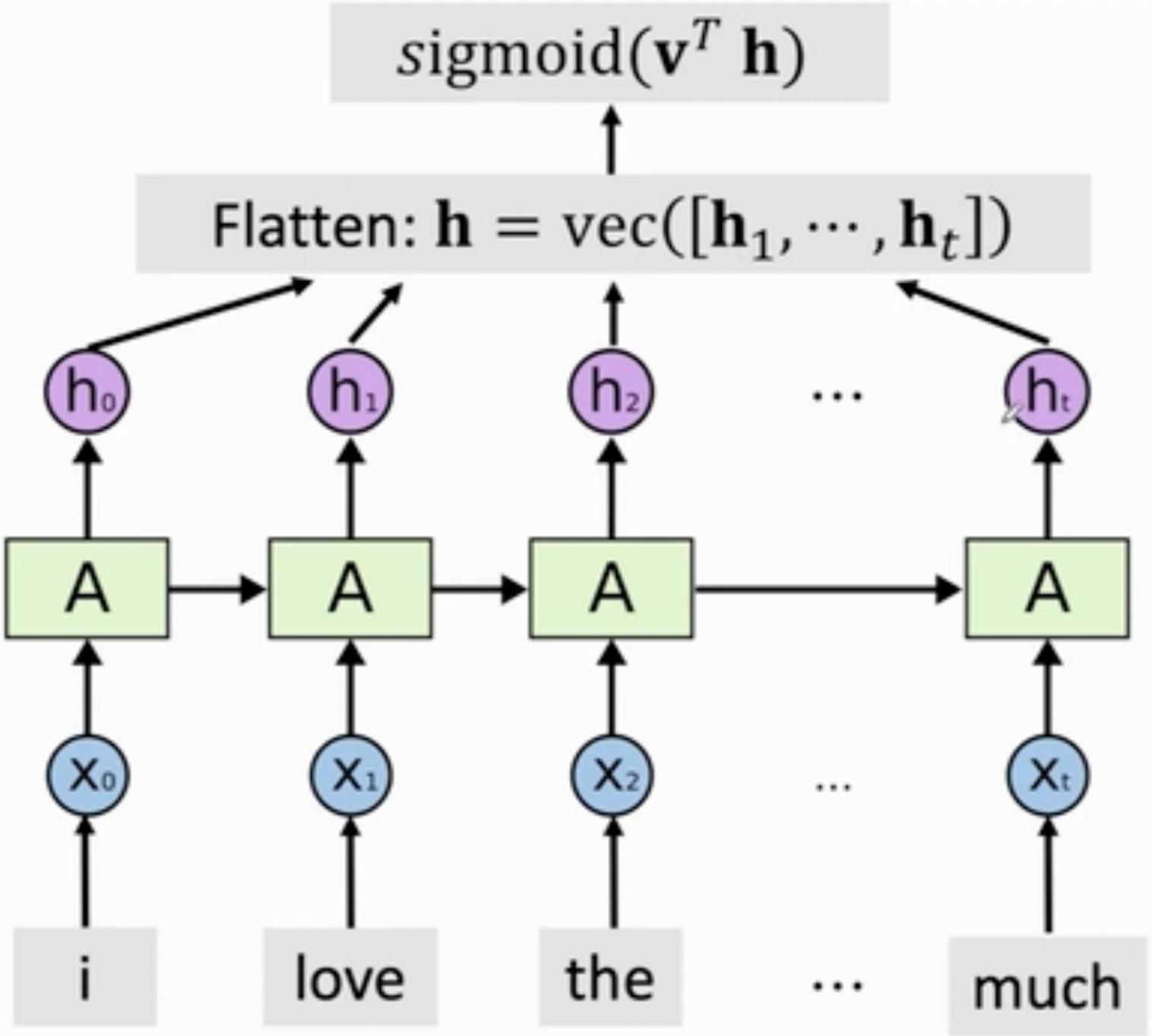
```
Epoch 1/3
43/43 [==============================] - 6s 124ms/step - loss: 0.6807 -
acc: 0.5627 - val_loss: 0.6713 - val_acc: 0.5813
Epoch 2/3
43/43 [==============================] - 5s 123ms/step - loss: 0.6706 -
acc: 0.5846 - val_loss: 0.6821 - val_acc: 0.5497
Epoch 3/3
43/43 [==============================] - 5s 124ms/step - loss: 0.6686 -
acc: 0.5919 - val_loss: 0.6638 - val_acc: 0.6013
782/782 [==============================] - 5s 7ms/step - loss: 0.6623 -
acc: 0.6057

_________________________________________________________________
_________________________
Test Loss and Accuracy
results  [0.6623082756996155, 0.605679988861084]
```

Simple RNN for IMDB Review

```
results = model.evaluate(x_test, y_test)
print("Test Loss and Accuracy")
print("results ", results)
```

```
Test Loss and Accuracy
results   [0.6623082756996155, 0.605679988861084]
```


$sigmoid(\mathbf{v}^T\ \mathbf{h})$
Flatten: $\mathbf{h} = \mathrm{vec}([\mathbf{h}_1, \cdots, \mathbf{h}_t])$
h_0
h_1
h_2
...
h_t
A
A
A
A
x_0
x_1
x_2
...
x_t
i
love
the
...
much

Simple RNN for IMDB Review

```
from keras.layers import Flatten

model = Sequential()
model.add(Embedding(vocabulary,embedding_dim,input_length=word_num))
model.add(SimpleRNN(state_dim,return_sequences=True))
model.add(Flatten())
model.add(Dense(1,activation='sigmoid'))
```

Simple RNN for IMDB Review

```
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 embedding (Embedding)        (None, 500, 32)           320000

 simple_rnn (SimpleRNN)       (None, 500, 32)           2080

 flatten (Flatten)            (None, 16000)             0

 dense (Dense)                (None, 1)                 16001

=================================================================
Total params: 338,081
Trainable params: 338,081
```

```
_____________________________________
Test Loss and Accuracy
results  [0.39525976777707672, 0.82264000177383342]
```

Stacked RNN for IMDB Review

```
model = Sequential()
model.add(Embedding(vocabulary,embedding_dim,input_length=word_num))
model.add(SimpleRNN(state_dim,return_sequences=True))
model.add(SimpleRNN(state_dim,return_sequences=True))
model.add(SimpleRNN(state_dim,return_sequences=True))
model.add(Flatten())
model.add(Dense(1,activation='sigmoid'))
```

双向 RNN for IMDB Review

from keras.layers import Bidirectional

```
model = Sequential()
model.add(Embedding(vocabulary,embedding_dim,input_length=word_num))
model.add(Bidirectional(SimpleRNN(state_dim,return_sequences=True)))
model.add(Flatten())
model.add(Dense(1,activation='sigmoid'))
```

```
_________________________________________________________________
Test Loss and Accuracy
results   [0.39559492468833923, 0.8216800093650818]
```

Part.03

LSTM

RNN缺点

RNN劣势：不适合长序列文本处理

- 尽管可以访问整个序列，但隐藏状态中的编码信息是非常局部的
- 当前状态与最近的前序输入更相关
- 长距离信息在许多NLP任务重很重要

The flights the airline was cancelling were full.

原因：

- 隐藏层权重用于 1)为当前决策提供信息 2) 为未来决策更新信息。

- 梯度消失(vanishing gradients)

LSTM长短期记忆（Long short-term memory）

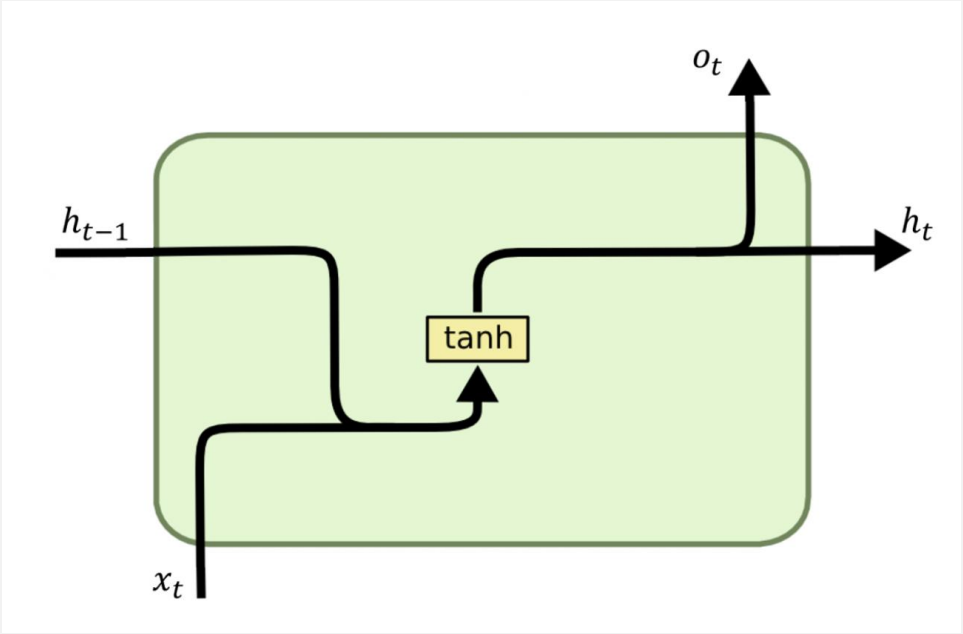
Hochreiter and Schmidhuber, 1997

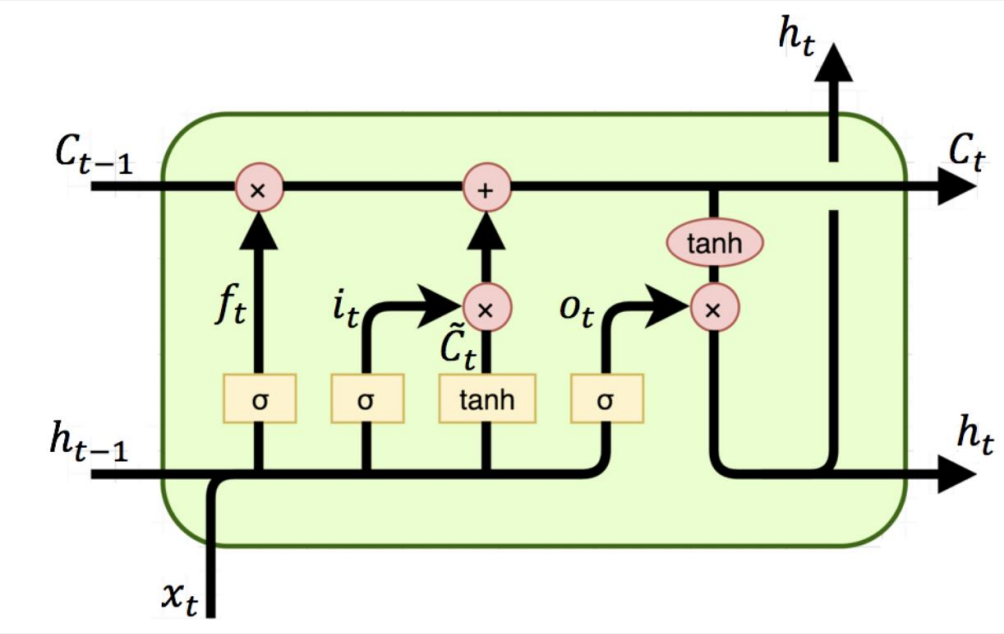
通过设计gate门机制实现，

从上下文中删除不需要的信息

添加对以后决策有帮助的信息

LSTM长短期记忆（Long short-term memory）



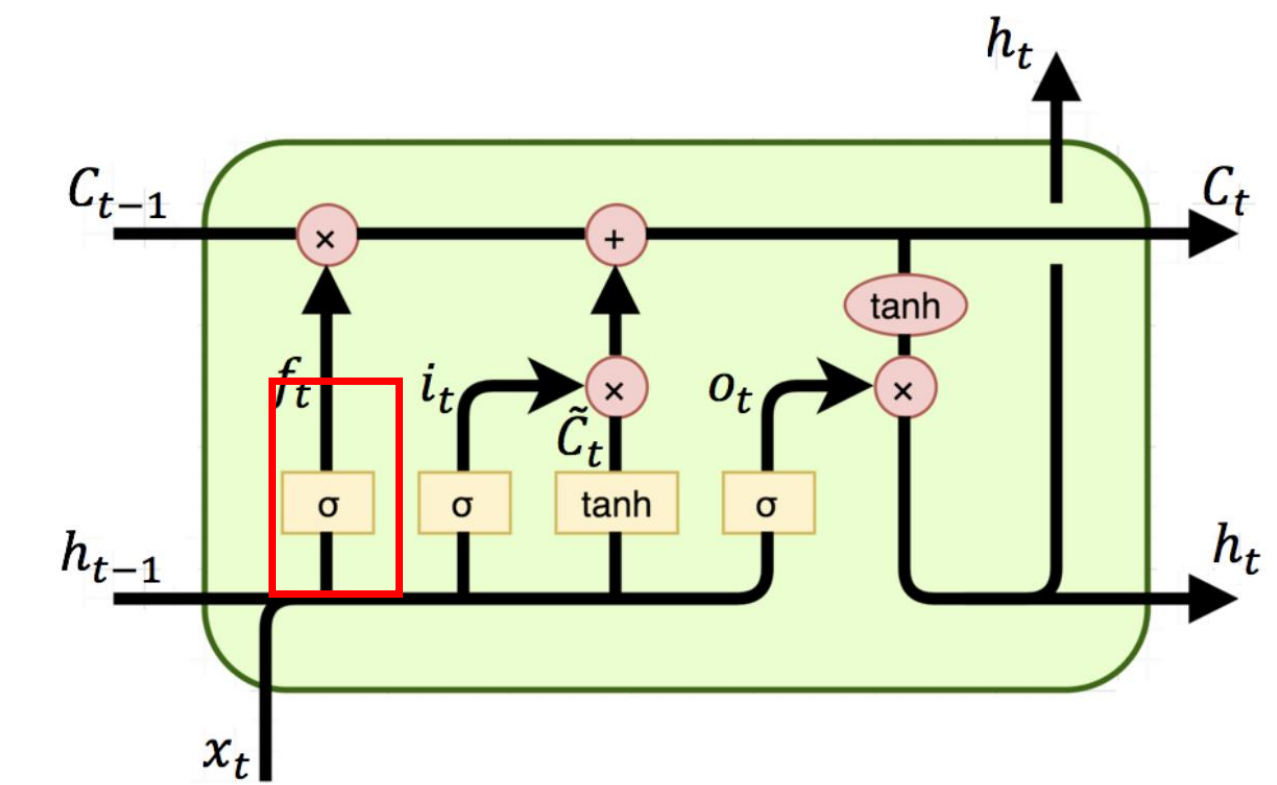


有那些门？

遗忘门(forget gate)，输入门(input gate)，输出门(output gate)

遗忘门(forget gate)

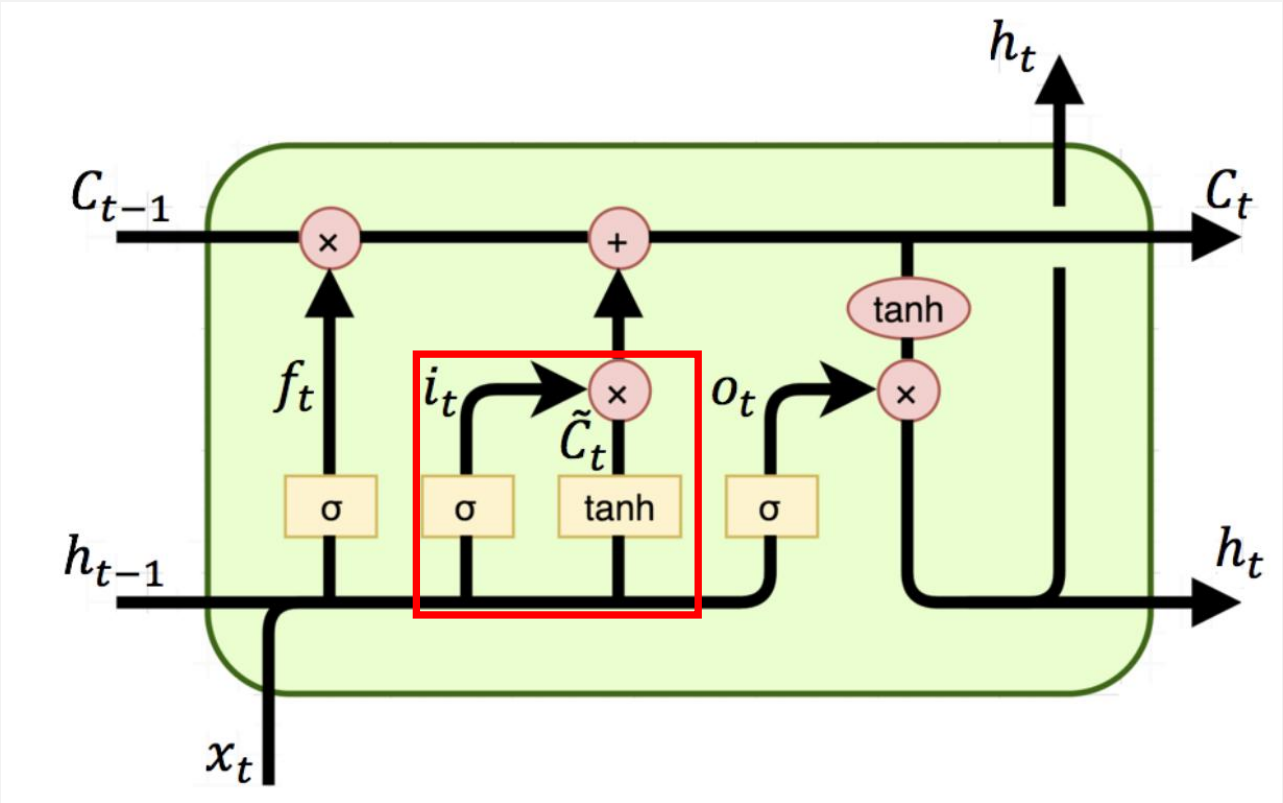
作用是为了从上下文中删除不需要的信息



1. 计算前一个隐藏层与当前输入的加权和
2. 用激活函数sigmoid非线性变换
3. 将结果与C_{t-1}做阿达玛乘积(Hadamard product)
4. 删除上下文中不需要的信息

$$\mathbf{f}_t = \sigma(\mathbf{U}_f \mathbf{h}_{t-1} + \mathbf{W}_f \mathbf{x}_t)$$
$$\mathbf{k}_t = \mathbf{c}_{t-1} \odot \mathbf{f}_t$$

输入门(input gate)



g_t携带之前状态信息和当前输入信息

$$\mathbf{g}_t = \tanh(\mathbf{U}_g\mathbf{h}_{t-1} + \mathbf{W}_g\mathbf{x}_t)$$

$$\mathbf{i}_t = \sigma(\mathbf{U}_i\mathbf{h}_{t-1} + \mathbf{W}_i\mathbf{x}_t)$$

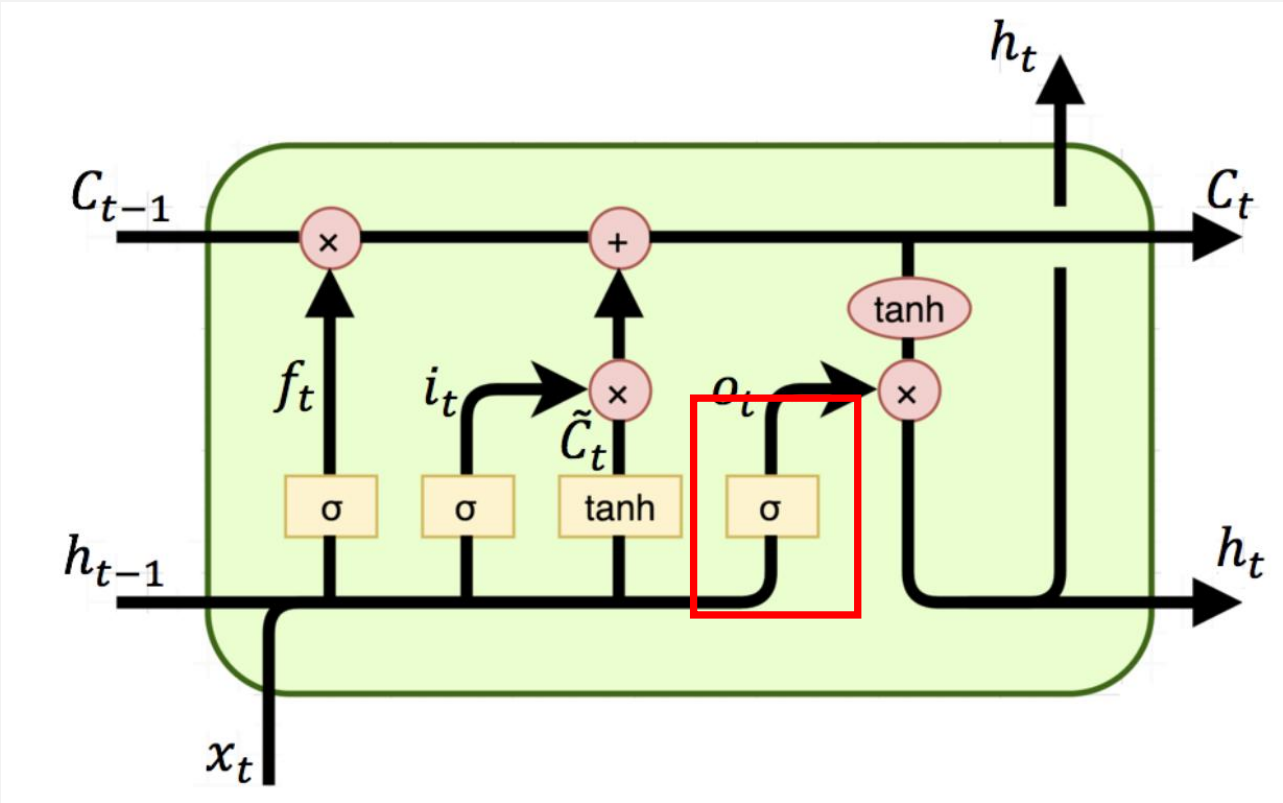
$$\mathbf{j}_t = \mathbf{g}_t \odot \mathbf{i}_t$$

C_t成为下一时间步的上下文向量

$$\mathbf{c}_t = \mathbf{j}_t + \mathbf{k}_t$$

输出门(output gate)

决定当前隐藏层需要那些信息(不考虑以后)



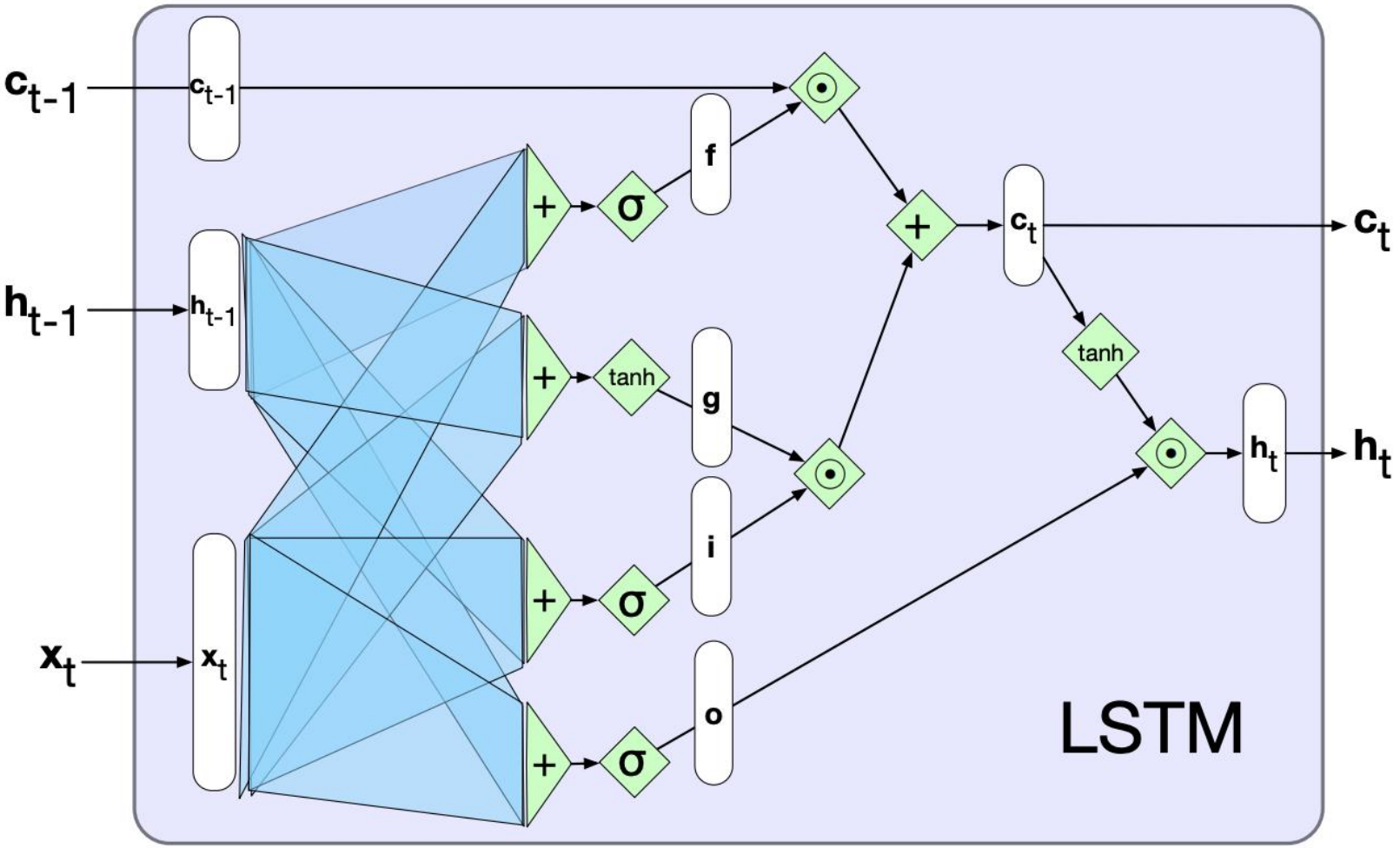
$$\mathbf{o}_t = \sigma(\mathbf{U}_o\mathbf{h}_{t-1} + \mathbf{W}_o\mathbf{x}_t)$$
$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t)$$

LSTM 结构图

t−1时上下文向量

t−1时隐藏层状态

当前输入



更新后的

t时上下文向量

t时隐藏层状态

σ Sigmoid 函数

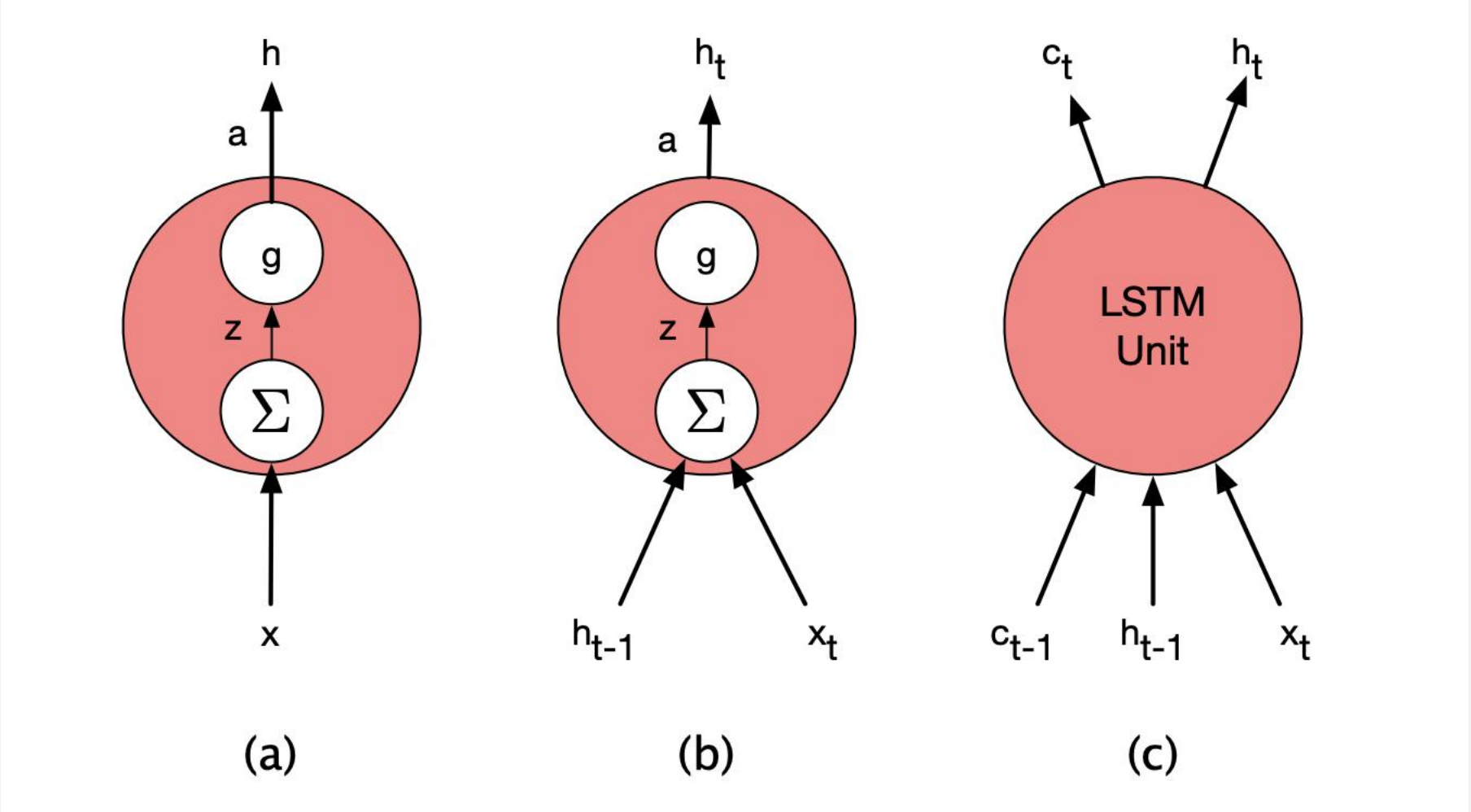
tanh 双曲正切函数

⊙ 阿达玛乘积

蓝色区域代表不同权重矩阵

对比 – 神经元，SRNN，LSTM

前馈网络机制



LSTM for IMDB Review

```
from keras.layers import LSTM

model = Sequential()
model.add(Embedding(vocabulary,embedding_dim,input_length=word_num))
model.add(LSTM(state_dim, return_sequences=True))
model.add(Flatten())
model.add(Dense(1,activation='sigmoid'))
```

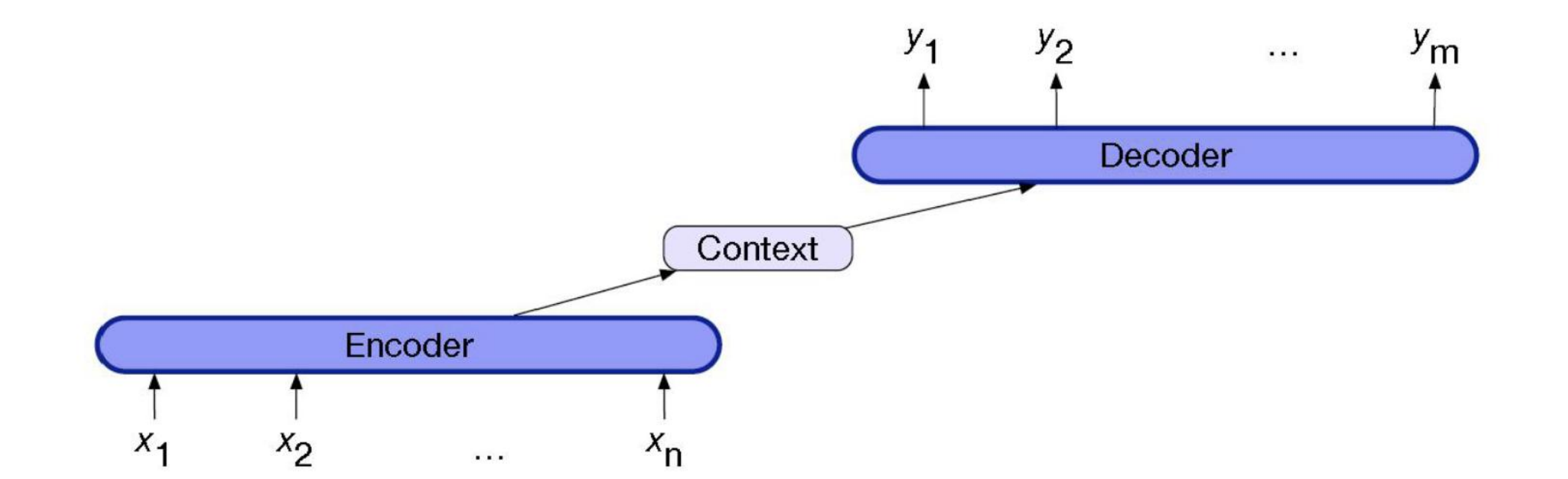
```
_________________________________________________________________
Test Loss and Accuracy
results   [0.38315340876579285, 0.8312399983406067]
```

Part.02

RNN编码器解码器模型

编码器—解码器模型

可生成上下文相关的任意长度的输出序列。
可用于机器翻译、摘要提取、自动问答或对话系统等前沿应用。



编码器：接受输入 x ，生成上下文表示 h（LSTM、RNN）
上下文：向量c，是h的函数，将输入的信息表征传递给解码器
解码器：接受输入 c ，根据表示h生成任意长度的输出序列y

基于RNN的编码器—解码器

回顾RNN中如何计算序列y的概率p(y)

$$p(y) = p(y_1)p(y_2|y_1)p(y_3|y_1,y_2)...P(y_m|y_1,...,y_{m-1})$$

在时间t，序列模型通过传递前缀t−1个tokens，并使用向前推理生成由多个隐藏状态组成的一个序列，最后一个隐藏状态h_{t-1}对应前缀的最后一个单词

之后，使用h_{t-1}作为生成下一token和下一隐藏状态h_t的起点

$$\mathbf{h}_t = g(\mathbf{h}_{t-1}, \mathbf{x}_t)$$

g 是激活函数

$$\mathbf{y}_t = f(\mathbf{h}_t)$$

f 是softmax函数

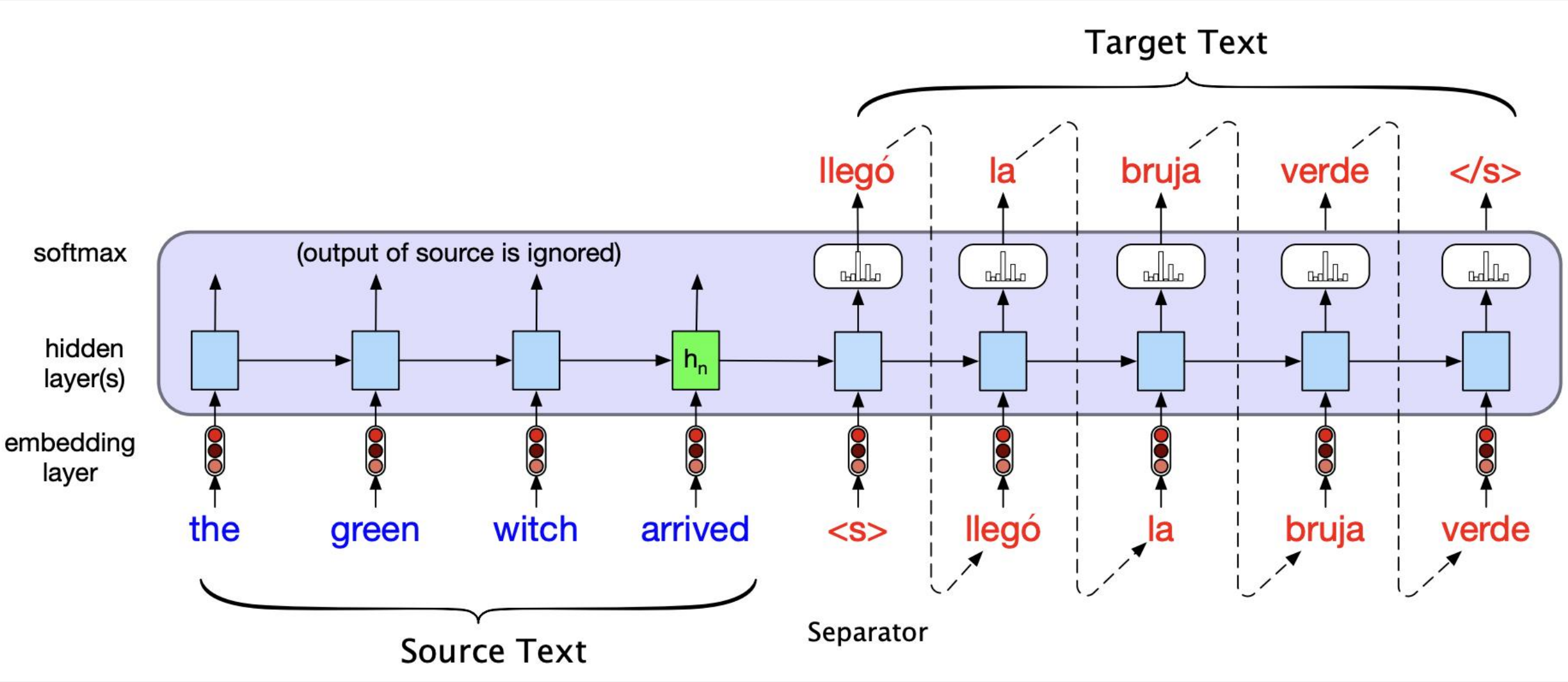
基于RNN的编码器—解码器

- 将该自回归生成语言模型转换为翻译模型
- 使用一个分隔符，将源句子(source)与目标句子(target)进行拼接

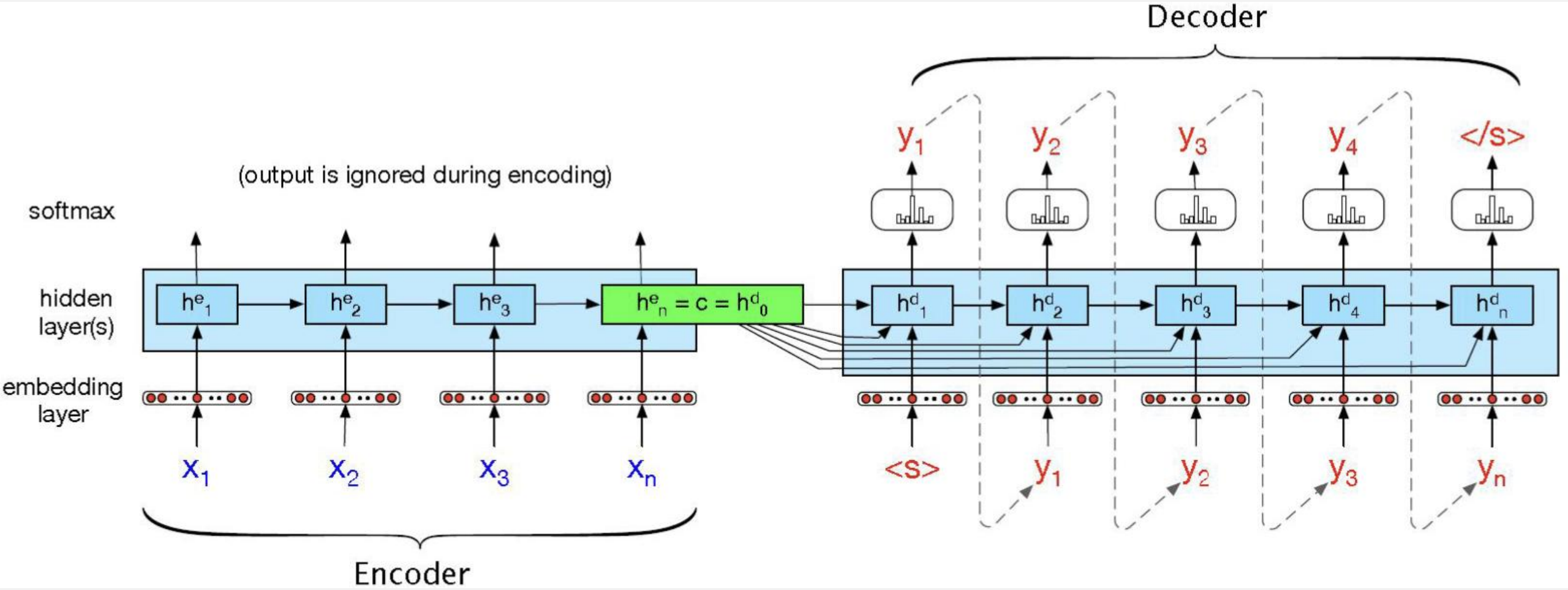
假设源句子为x，目标句子为y，则条件概率 $p(y|x)$ 可以表达为：

$$p(y|x) = p(y_1|x)p(y_2|y_1,x)p(y_3|y_1,y_2,x)...P(y_m|y_1,...,y_{m-1},x)$$

基于RNN的编码器—解码器



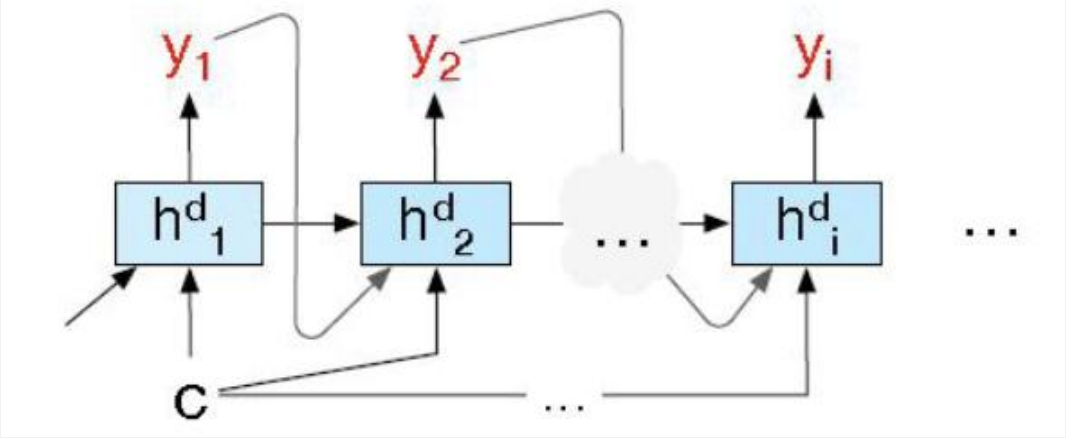
正式的基于RNN编码器–解码器的机器翻译



上下文向量C

在上述方法中，上下文向量C的影响随着序列生成而不断地减弱。

解决方案



$$
\begin{aligned}
\mathbf{c} &= \mathbf{h}_n^e \\
\mathbf{h}_0^d &= \mathbf{c} \\
\mathbf{h}_t^d &= g(\hat{y}_{t-1}, \mathbf{h}_{t-1}^d, \mathbf{c}) \\
\mathbf{z}_t &= f(\mathbf{h}_t^d) \\
y_t &= \text{softmax}(\mathbf{z}_t)
\end{aligned}
$$

$$
\hat{y}_t = \text{argmax}_{\text{w} \in \text{V}} P(w|x, y_1 \ldots y_{t-1})
$$

Part.03

训练编码解码器

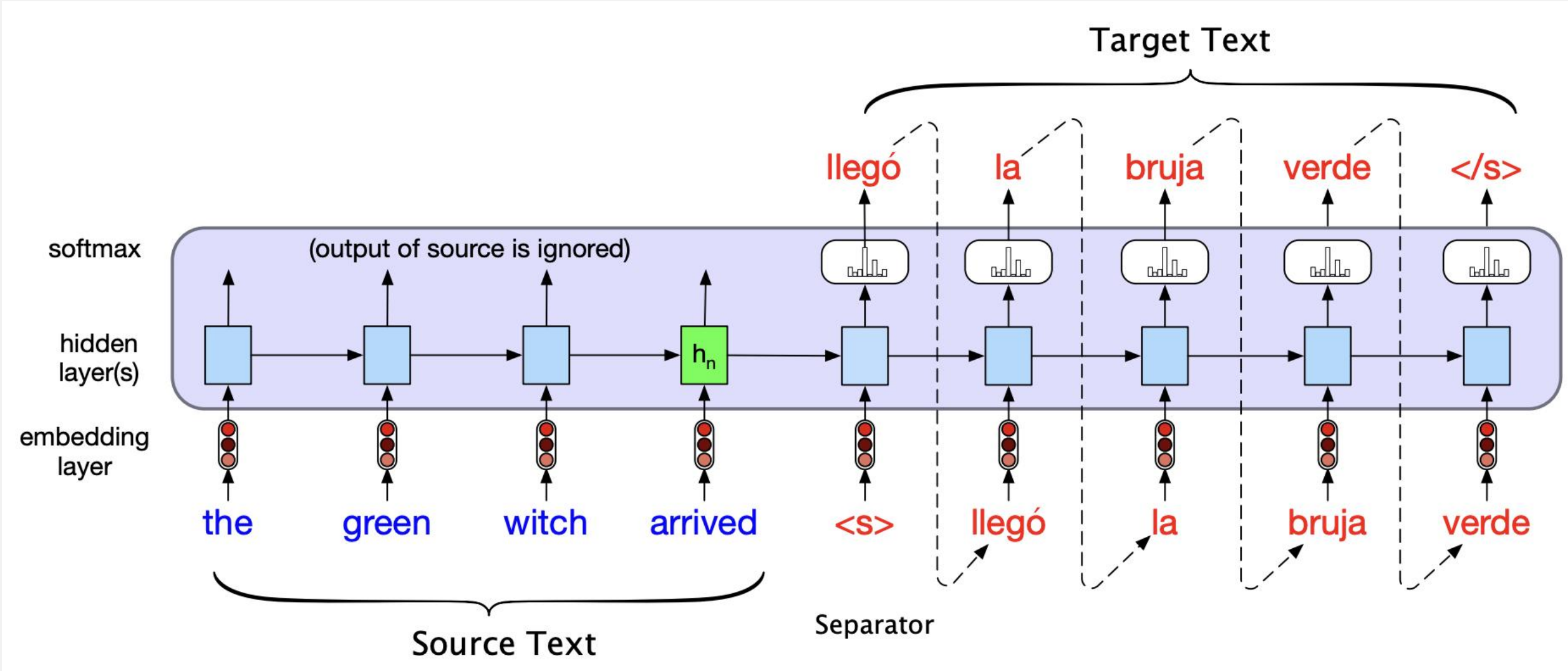
编解码器模型训练

准备训练数据：

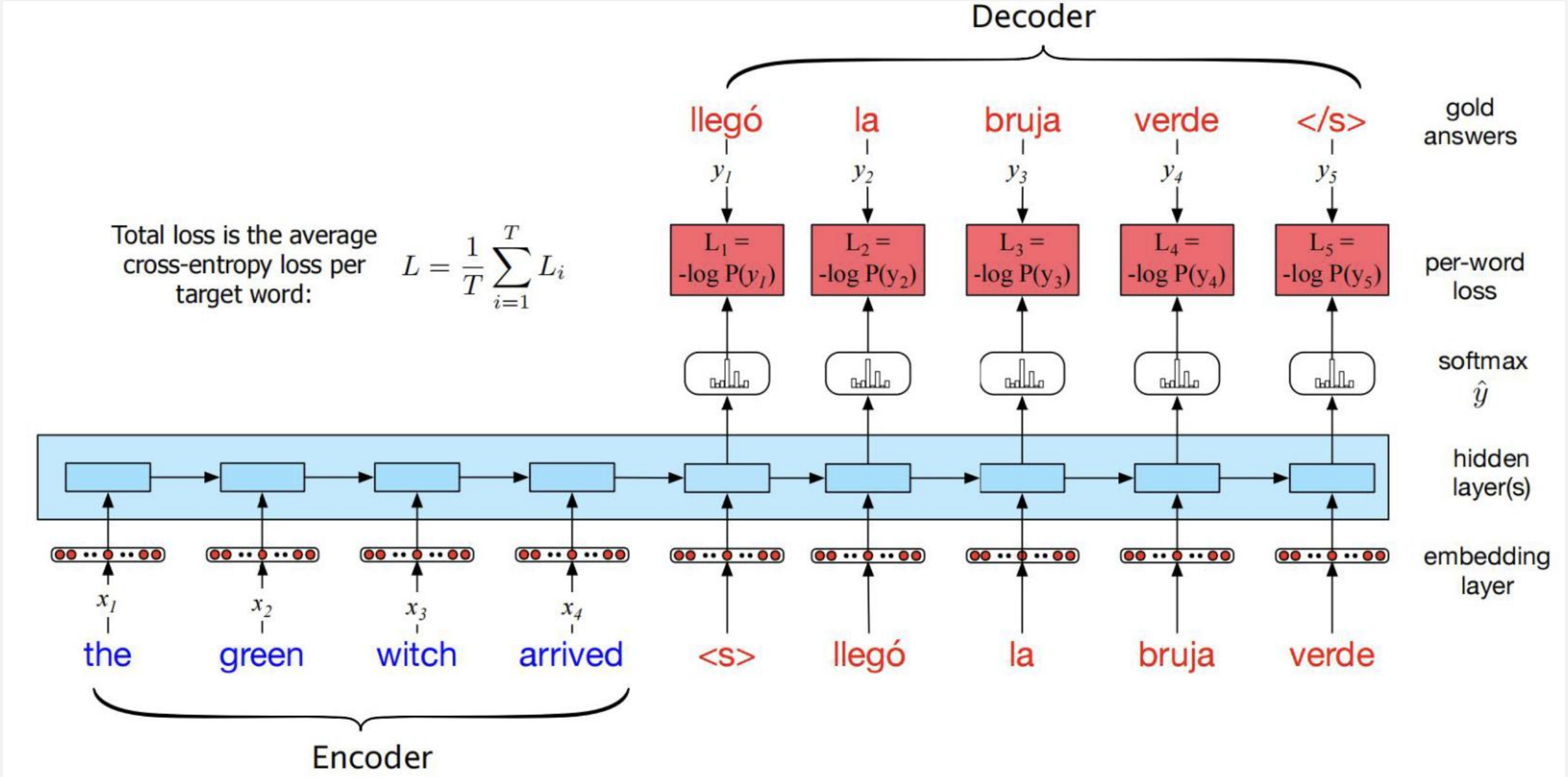
- 每一个训练样本是由一个源文本和一个目标文本组成的二元组
- 将二元组的两个元素通过一个分隔符进行拼接

网络模型使用源文本和分隔符，自回归地训练生成下一单词

回顾 – 模型推理


Target Text
llegó
la
bruja
verde
</s>
softmax
(output of source is ignored)
hidden layer(s)
h_n
embedding layer
the
green
witch
arrived
<s>
llegó
la
bruja
verde
Source Text
Separator

使用教师强制的解码器


Decoder
llegó
la
bruja
verde
</s>
gold answers
y_1
y_2
y_3
y_4
y_5
Total loss is the average cross-entropy loss per target word:
$L = \frac{1}{T}\sum_{i=1}^{T} L_i$
$L_1 = -\log P(y_1)$
$L_2 = -\log P(y_2)$
$L_3 = -\log P(y_3)$
$L_4 = -\log P(y_4)$
$L_5 = -\log P(y_5)$
per-word loss
softmax $\hat{y}$
hidden layer(s)
embedding layer
x_1
x_2
x_3
x_4
the
green
witch
arrived
<s>
llegó
la
bruja
verde
Encoder

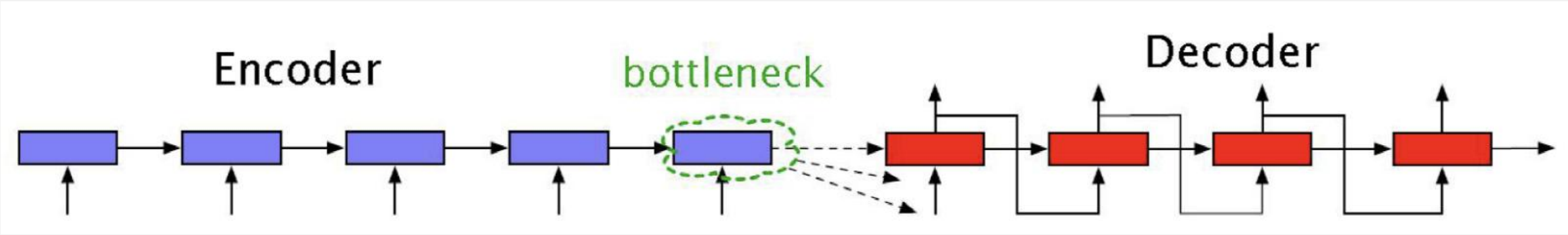
Part.04

注意力机制

隐藏状态是一个瓶颈

解码器自回归生成的唯一依据是隐藏信息－上下文向量c

- 上下文向量c需携带源文本的全部信息
  - 但隐藏状态对长序列文本不能很好地进行表示。



解决方案：注意力机制

- 允许 解码器从编码器获取每一时间步的隐藏信息，而不仅仅是最后一步。

注意力机制

在注意力机制中，上下文向量c是用于表示所有编码器隐藏状态的一个函数，即：

$$\mathbf{c} = f(\mathbf{h}_1^e \ldots \mathbf{h}_n^e)$$

为什么不可以直接使用，由所有隐藏状态组成的张量，作为上下文？

- 因为隐藏状态的数量，随着序列长度变化而不断地变化

注意力机制中，通过对编码器中的所有隐藏状态进行加权求和，

- 生成一个固定长度的上下文向量c(即上述函数)

动态生成上下文向量

这些权重用于关注源文本的“特殊”部分，

- 这一特殊部分与解码器当前生成的token是相关的。

- 该特殊部分是会变化的(因为不同时间步下的tokens也不同)
  - 因此，注意力机制生成的上下文向量是动态的

注意力机制根据解码器当前生成的token，
使用编码器的所有隐藏状态，
动态生成上下文向量c

计算解码器当前隐藏状态

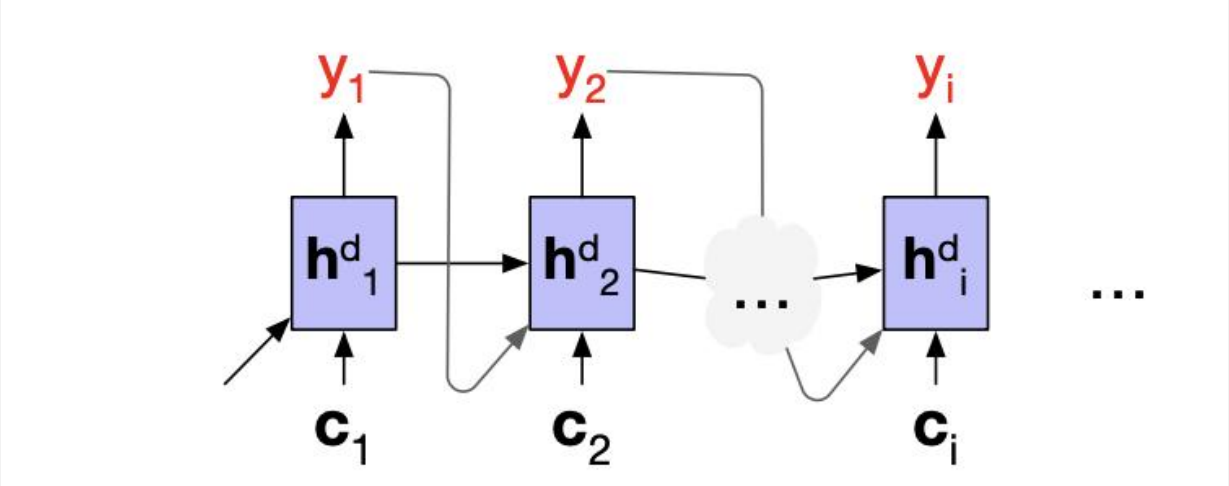
在解码阶段的时间步i上，向量c_i由编码器的所有隐藏状态推导得出。

计算当前隐藏状态(h^d_i)，需要考虑的条件包括：

- 当前上下文向量c_i
- 解码器的先前隐藏状态(h^d_{i-1})
- 解码器先前生成的输出($\hat{y}_{i-1}$)

$$\mathbf{h}_i^d = g(\hat{y}_{i-1}, \mathbf{h}_{i-1}^d, \mathbf{c}_i)$$

计算解码器当前隐藏状态



解码器的先前隐藏状态(h^d_{i-1}) 已知的

解码器先前生成的输出($\hat{y}_{i-1}$) 已知的

当前上下文向量c_i 未知的

$$\mathbf{h}_i^d = g(\hat{y}_{i-1}, \mathbf{h}_{i-1}^d, \mathbf{c}_i)$$

如何动态计算上下文向量c

首先需要计算：

- 编码器中的每一个隐藏状态与解码器中的前一个隐藏状态之间的相关性

如何计算？

使用 点积注意力

$$score(\mathbf{h}_{i-1}^{d}, \mathbf{h}_{j}^{e}) = \mathbf{h}_{i-1}^{d} \cdot \mathbf{h}_{j}^{e}$$

有了相关性后该如何做？

由于点积的结果是一个标量，无法直接作用在每一个编码器的隐藏状态上(向量)。

因此需要用softmax对所有计算获得的点积结果进行归一化处理，
从而获取权重向量α_{ij}

$$\alpha_{ij} = \text{softmax}(score(\mathbf{h}_{i-1}^{d}, \mathbf{h}_{j}^{e}) \ \forall j \in e)$$
$$= \frac{\exp(score(\mathbf{h}_{i-1}^{d}, \mathbf{h}_{j}^{e}))}{\sum_{k} \exp(score(\mathbf{h}_{i-1}^{d}, \mathbf{h}_{k}^{e}))}$$

动态计算：加权平均上下文向量c

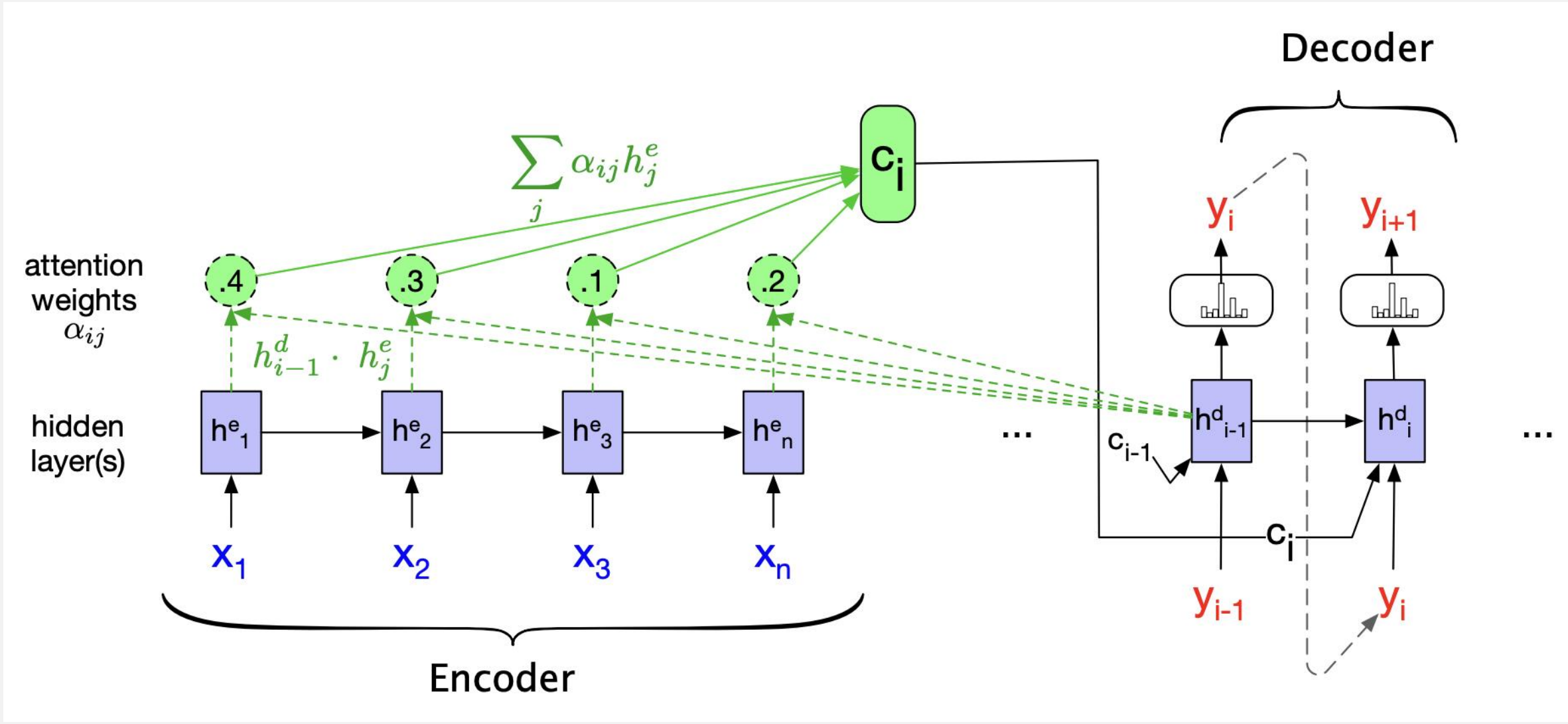
计算所需的条件：

1. 权重向量α_{ij}：　用以表示前一个解码器隐藏状态与每一个编码器隐藏状态的相关性
2. 每一个编码器隐藏状态：提供源文本的上下文信息

$$\mathbf{c}_i = \sum_j \alpha_{ij} \mathbf{h}_j^e$$

每一次生成新的解码器隐藏状态时，都会重新计算一次权重向量α_{ij}
从而生成新的上下文向量c

带注意力机制的编解码器模型



另一种注意力计算方法

在前一个解码器隐藏状态h_{i-1}^{d}与每一个解码器隐藏状态h_{j}^{e}之间添加一个参数矩阵

$$score(\mathbf{h}_{i-1}^{d}, \mathbf{h}_{j}^{e}) = \mathbf{h}_{t-1}^{d} \mathbf{W}_{s} \mathbf{h}_{j}^{e}$$

该矩阵是在正常的端到端训练过程中训练获得

- 可以帮助模型学习并捕获编码器和解码器之间的相关性
- 是一种双线性模型，允许编解码器之间的维度不同

本章总结

RNN	隐藏层，时间反向传播
LSTMs	遗忘门，输入门，输出门
Encoder—Decoder	多头自注意力机制