



首都师范大学

為學為師求實求新

深度学习应用与工程实践

10. 循环神经网络和语言模型

10. RNN and Language Models

李冰

Bing Li

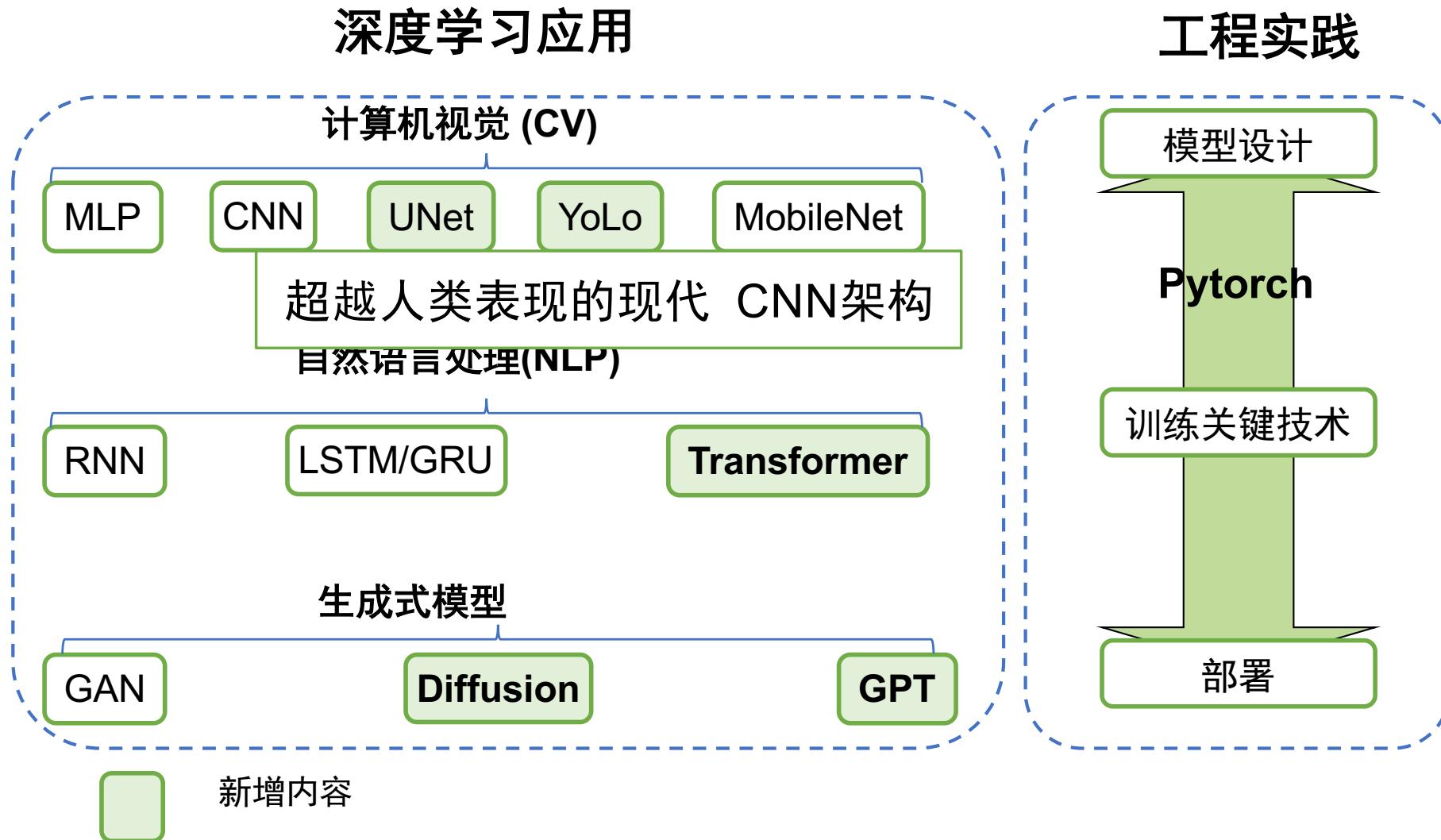
Associate Professor

Academy of Multidisciplinary Studies

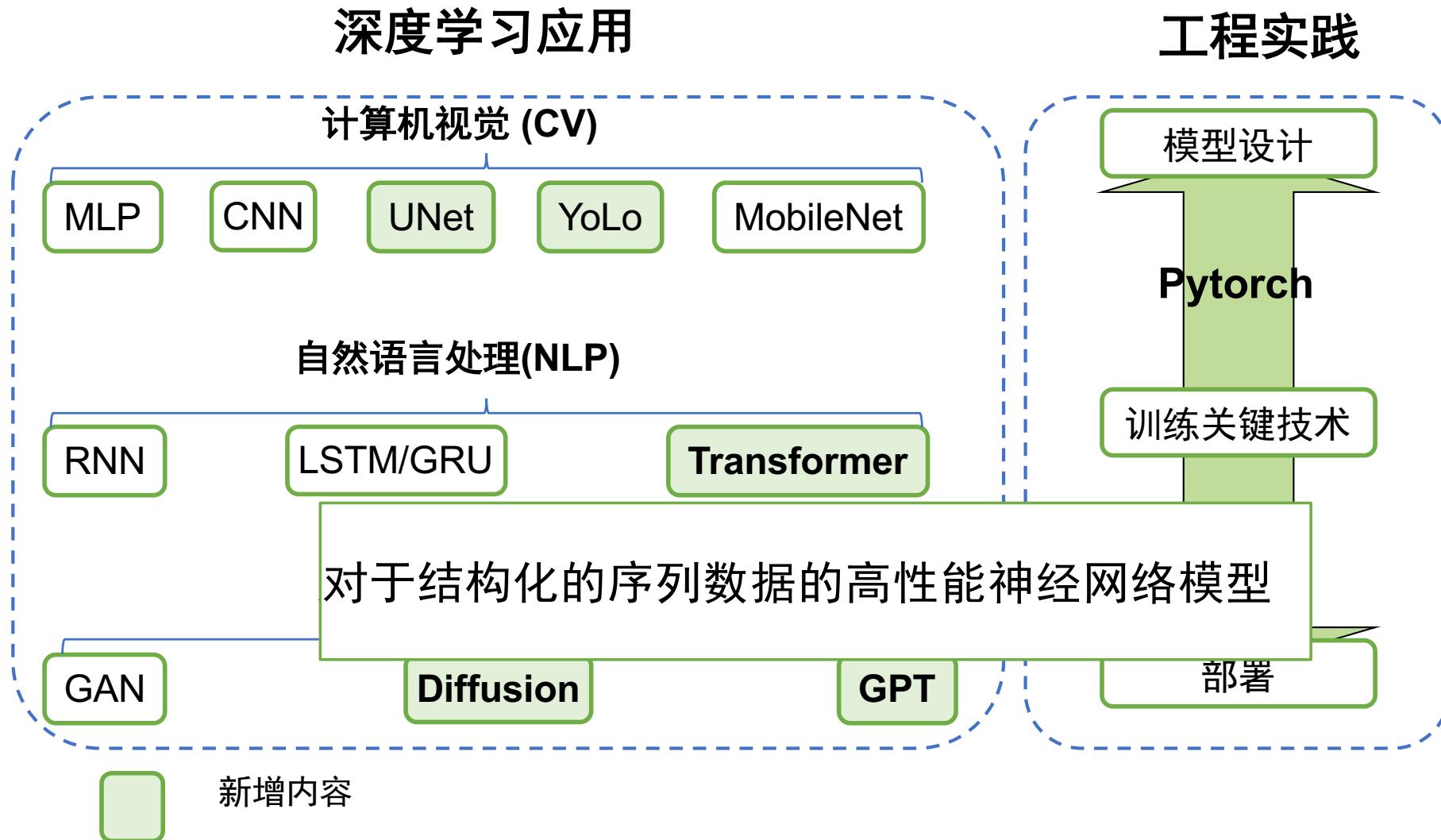
Capital Normal University



本门课的内容



本门课的内容

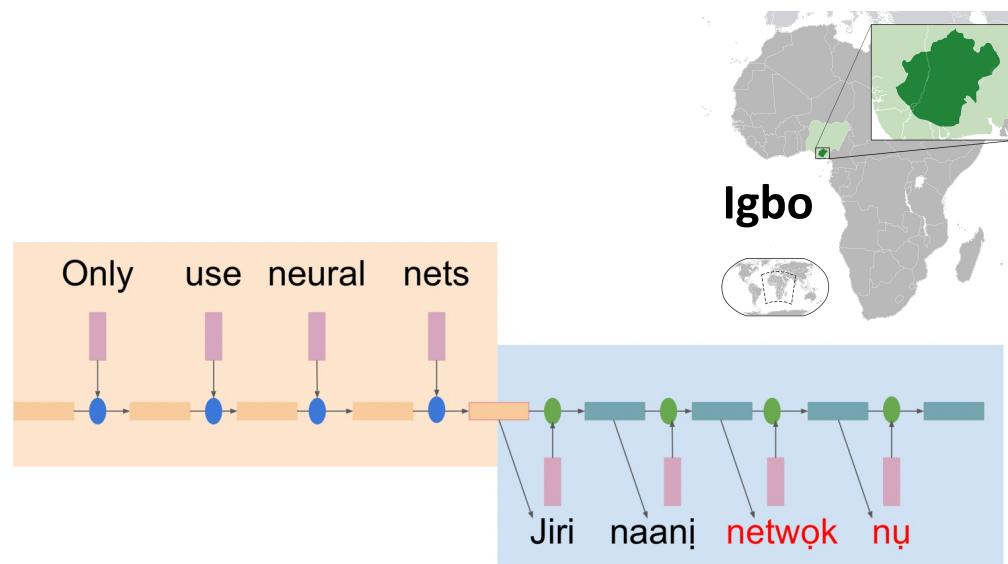


序列数据

机器翻译

序列数据

- 数据有时间维度. 数据会随时间变化.
- 大部分是句子和音频.
- 需要**序列化模型**来提取时间维度上的信息



图片分类

非序列数据（网格数据）

- 随着时间， 输入不会变化.
- 大部分是图片.



这节课

- 自然语言处理 (NLP)
- 词向量
- 循环神经网络
- 循环神经网络语言模型



自然语言处理

什么是自然语言处理？

自然语言处理时一个交叉学科：

- 计算机科学
- 人工智能
- 数学
- 语言学
- 是计算机和人类语言之间的交互



自然语言处理

NLP 应用

- 语义分析 & 问答
- 神经网络机器翻译
- 语音识别/字符识别
- 文本/情感分析
 - 文本分类(Text Classification)是指依据文本的内容,由计算机根据某种分类算法,把文本判分为预先定义好的一个或多个类别的过程。
 - 文本情感分析(也称为意见挖掘)是指用自然语言处理、文本挖掘以及计算机语言学等方法来识别和提取原素材中的主观信息。
- 词性标注
 - 给句子中每个词一个词性类别的任务。这里的词性类别可能是名词、动词、形容词或其他。
- ...



文本情感分析

($w_0, w_1 \dots w_{t-1}$) → 一个词

¥2099-2996

购买得积分

查看 >

享6期免息,可免94.5元,每期349.8元(每日11.7元)

天猫 Nintendo Switch 任天堂家用游戏机
续航版增强版 掌机NS体感游戏机 国行S...

推荐 3510

送给TA

帮我选

性价比高(3)

全部

有图/视频(1237)

追加(596)

按时间排序 | 默认排序



博**哥

3天前 | 套餐:单机标配 颜色分类:灰色手柄主机 版本类...

有句话说“不要把你的手放在switch上”。因为一放上去就根本停不下来, 太好玩了! 一台Switch可以满足在不同场景下的游玩体验。在家里可以连上电视, 分享Joy-Con手柄, 与家人体验游戏的乐趣。出门在外可以随身携带轻便掌机, 让游戏的快乐尽在掌中; 也可支在桌面, 与朋友对战或分享快乐。一举三得, 谁不爱



新势力周价

¥1277.42起

新势力周

3月24日
00:00开卖

跨店每500减5 店铺红包 抵5元

领券 >

¥1188起

任天堂 switch NS 主机 Lite 掌机 续航加强
版主机 动物之森限定粉色 专属优惠

物流快(88)

正品(36)

全部

有图/视频(360)

追加(38)

好评(1万)

中/差评

按时间排序 | 默认排序



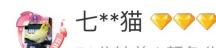
t**1

8分钟前

评价方未及时做出评价,系统默认好评!

浏览 0 次

评论 有用 ...



七**猫

51分钟前 | 颜色分类:新版续航+赠品+塞尔达荒野之息 [...]

不多说, 支持自由人, 把朋友一起拉入ns坑, 对自由人很信任, 金手指稍微带点划痕, 现在好玩的都是合作款游戏, 要人手一台机子联机, 还会继续推荐朋友来买的



¥1188起

任天堂 switch NS 主机 Lite 掌机 续航加强
版主机 动物之森限定粉色 专属优惠

推荐 978

送给TA

帮我选

发货 地 北京 | 快递: 免运费

月销 1580

服务 付款后 48 小时内发货 · 7 天无理由

选择 请选择 颜色分类 套餐 版本类型



共 24 种颜色分类可选

参数 品牌 任天堂型号...

宝贝评价 (10709)

查看全部 >

物流快(88) 正品(36)

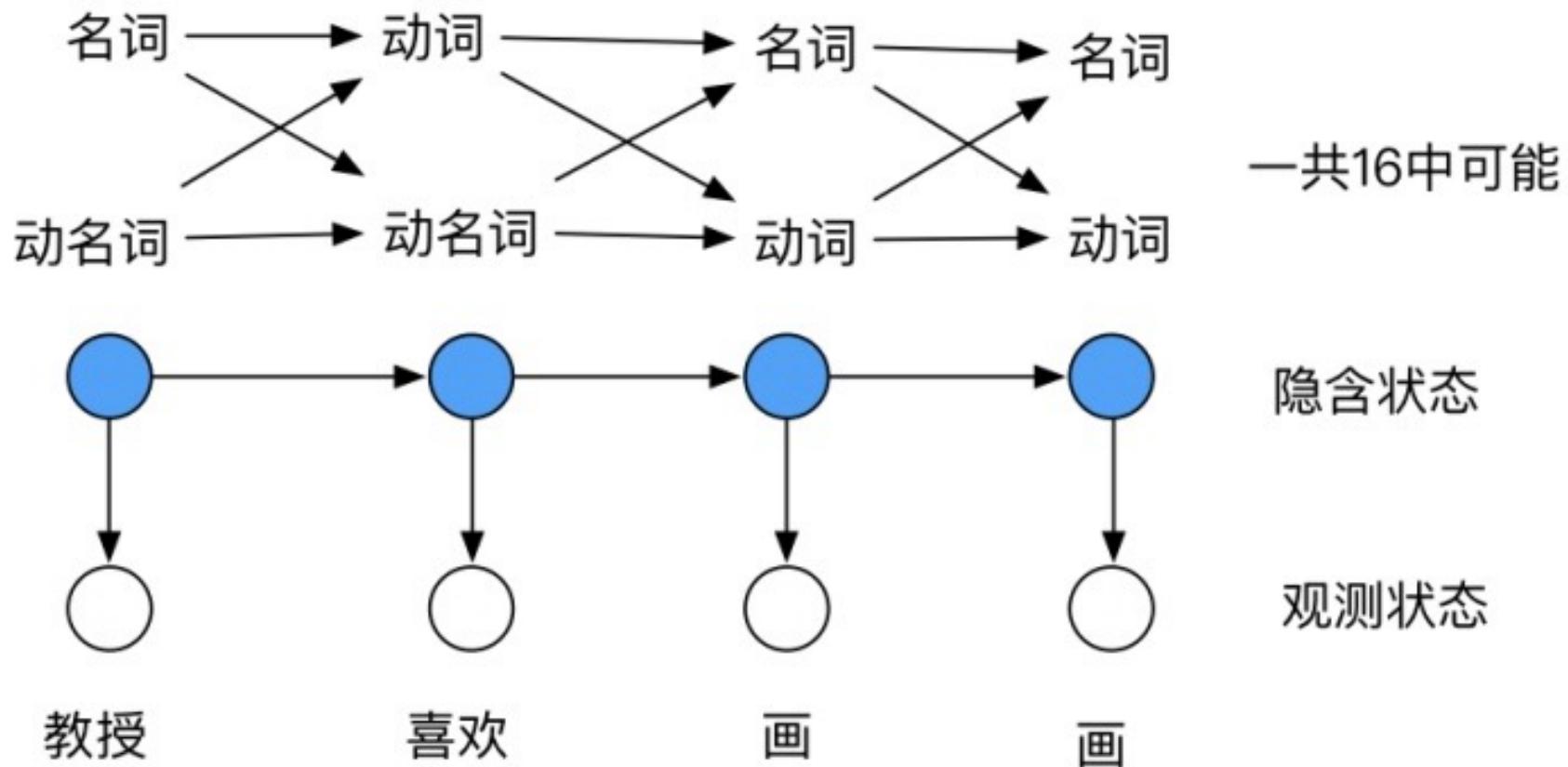


t**7

三码合一开机红, 没有使用过的痕迹, 新机沒错了, 游戏也是全新无划痕的, 没翻车好评 😎

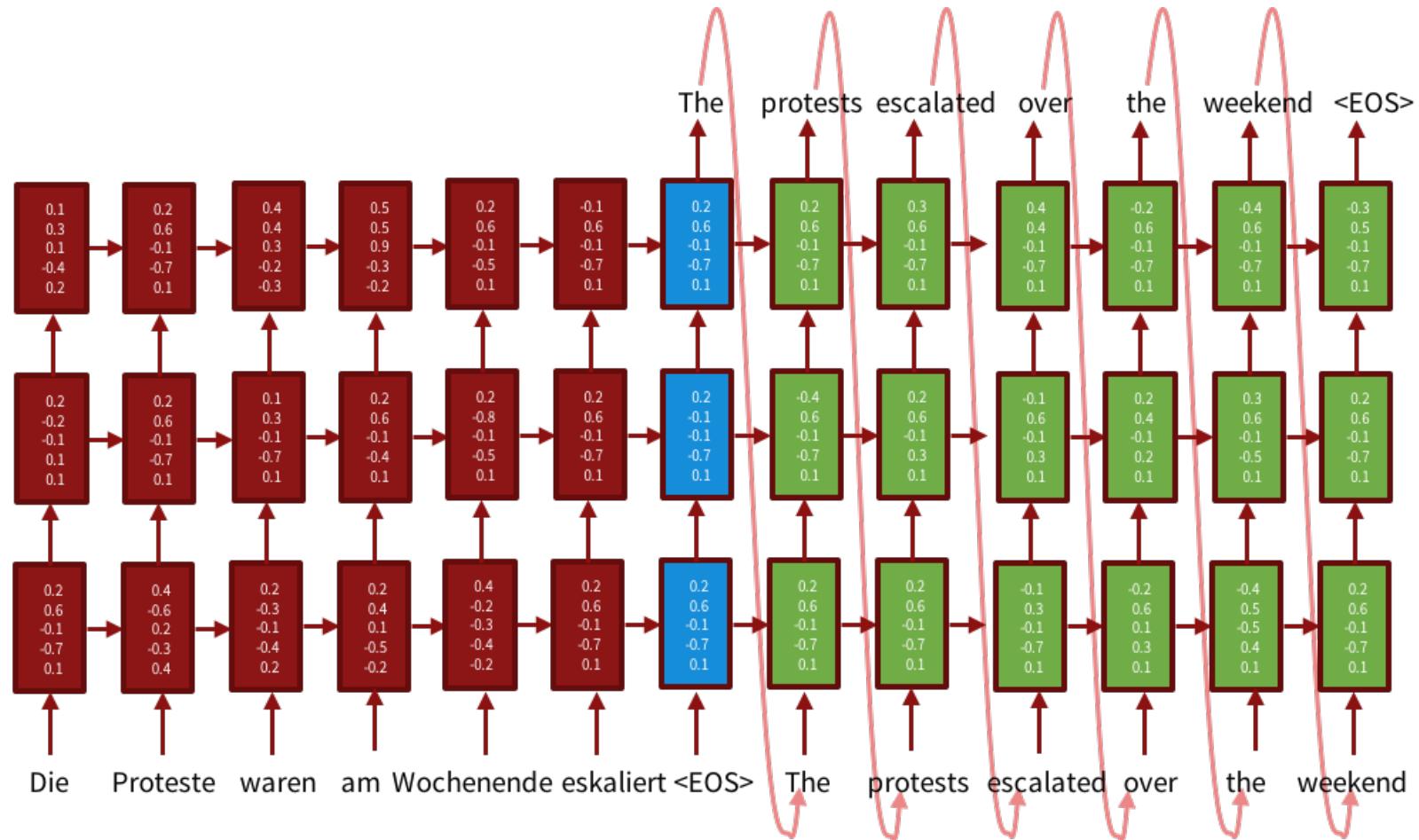
词性标注

- 为每个词标注它们的词性 $(w_0, w_1 \dots w_{t-1}) \rightarrow (t_0, \dots t_{t-1})$



神经网络机器翻译

- 为每个词找到对应的翻译 $(w_0, w_1 \dots w_{t-1}) \rightarrow (w_0^*, \dots w_{l-1}^*)$



这节课

- 自然语言处理 (NLP)
- 词嵌入
 - Word2Vec
- 循环神经网络
- 循环神经网络语言模型



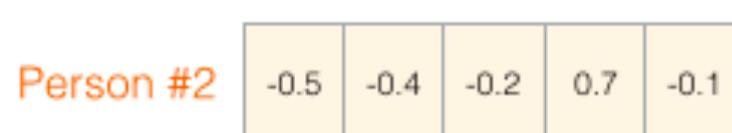
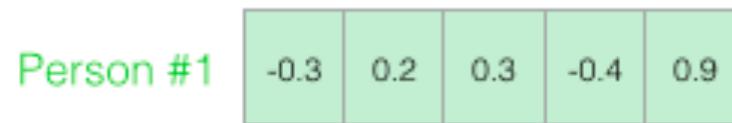
词嵌入Word Embedding

- 将自然语言中的词表示为向量。
- 五大人格特质测试OCEAN

O代表Openness to experience(开放性) ,
C代表Conscientiousness(严谨性) ,
E代表Extraversion(外向性) ,
A代表Agreeableness (宜人性) ,
N代表Negative emotionality(神经质)



Openness to experience	79	out of	100
Agreeableness	75	out of	100
Conscientiousness	42	out of	100
Negative emotionality	50	out of	100
Extraversion	58	out of	100



很容易地计算出相似的向量之间的相互关系。

词嵌入

- 每个单词或词组被映射为实数域上的向量。

word——>vector 映射

- 用稀疏的one-hot向量表示单词：

- 以字典建立向量，词所处的位置用1表示，其余为0。
- 只含一个 1、其他都是 0 的向量来唯一表示词语

“话筒” 表示为 [0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 ...]

“麦克” 表示为 [0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 ...]



词向量

但是...

- 随着词汇表的增大，向量会越来越长
 - 词汇表通常包含超过50k个单词，这意味着存储单词向量会占用大量内存。
 - 任意两个向量是正交的，两个词是孤立的。
 - 如何探索“话筒”和“麦克”之间的相关性和相似度？
- 所以
- 将相似性编码为单词向量。
 - 减少单词向量的维数



词嵌入 Embedding

一个维数为所有词的数量的高维空间嵌入到一个维数低得多的连续向量空间中。

The **quick brown fox jumps over the lazy dog.** fox =

$$\begin{bmatrix} 0.142 \\ -0.920 \\ -0.011 \\ 0.024 \\ 0.815 \\ -0.912 \end{bmatrix}$$

核心思想：一个单词的含义可以用周围的其他单词（上下文）来表示。

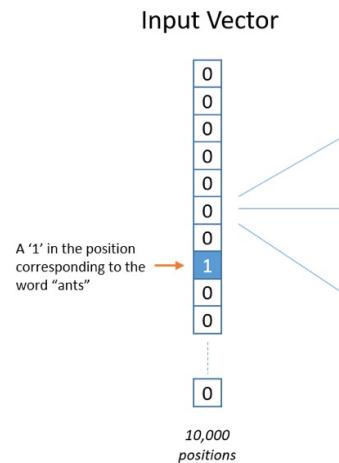
我们使用上下文来表示某个单词的含义，并将其编码在词向量中。

- 降低了维度
- 保持一定相关关系



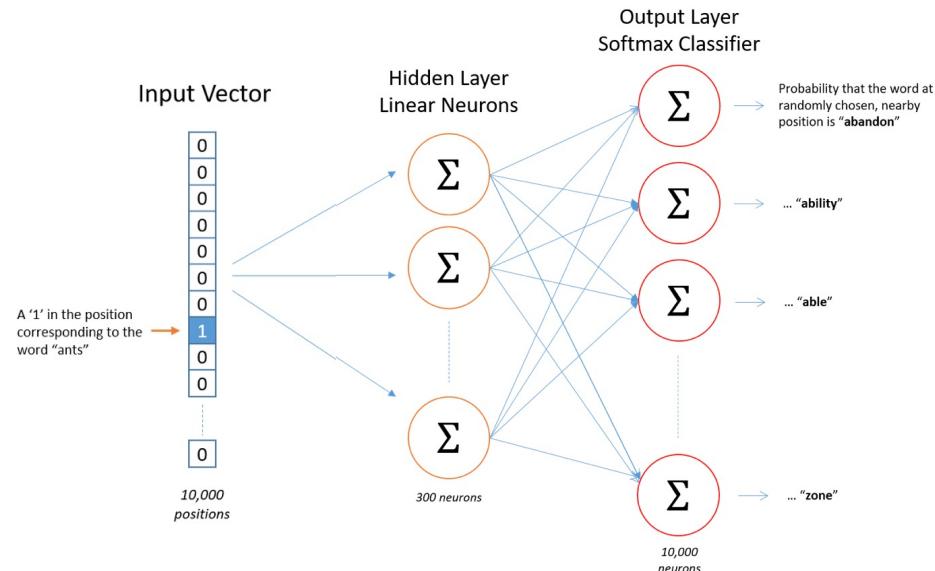
Word2Vec模型

- 为了得到 word——>vector 这个映射关系神经网络。
- 构建数据：用原始数据构建单词对，单词形式如下 [input word, out word]，训练时[data x, label y]。
- 输入层
 - 将所有词进行one-hot编码作为输入， 输入的是n维向量(n是词表单词个数)



Word2Vec模型

- 隐藏层，实际上存储了词汇表中所有单词的词向量。
 - 中间是只有一个隐藏层 (没有激活函数，只是线性的单元)。
 - 隐藏层这是一个尺寸为 [词表大小, 词向量大小] 的矩阵。
 - 矩阵的每一行对应了某一个单词的词向量。
- 输出层，维度跟输入层的维度一样，各维的值相加为1。
 - 用Softmax回归。用softmax来衡量神经网络的输出。

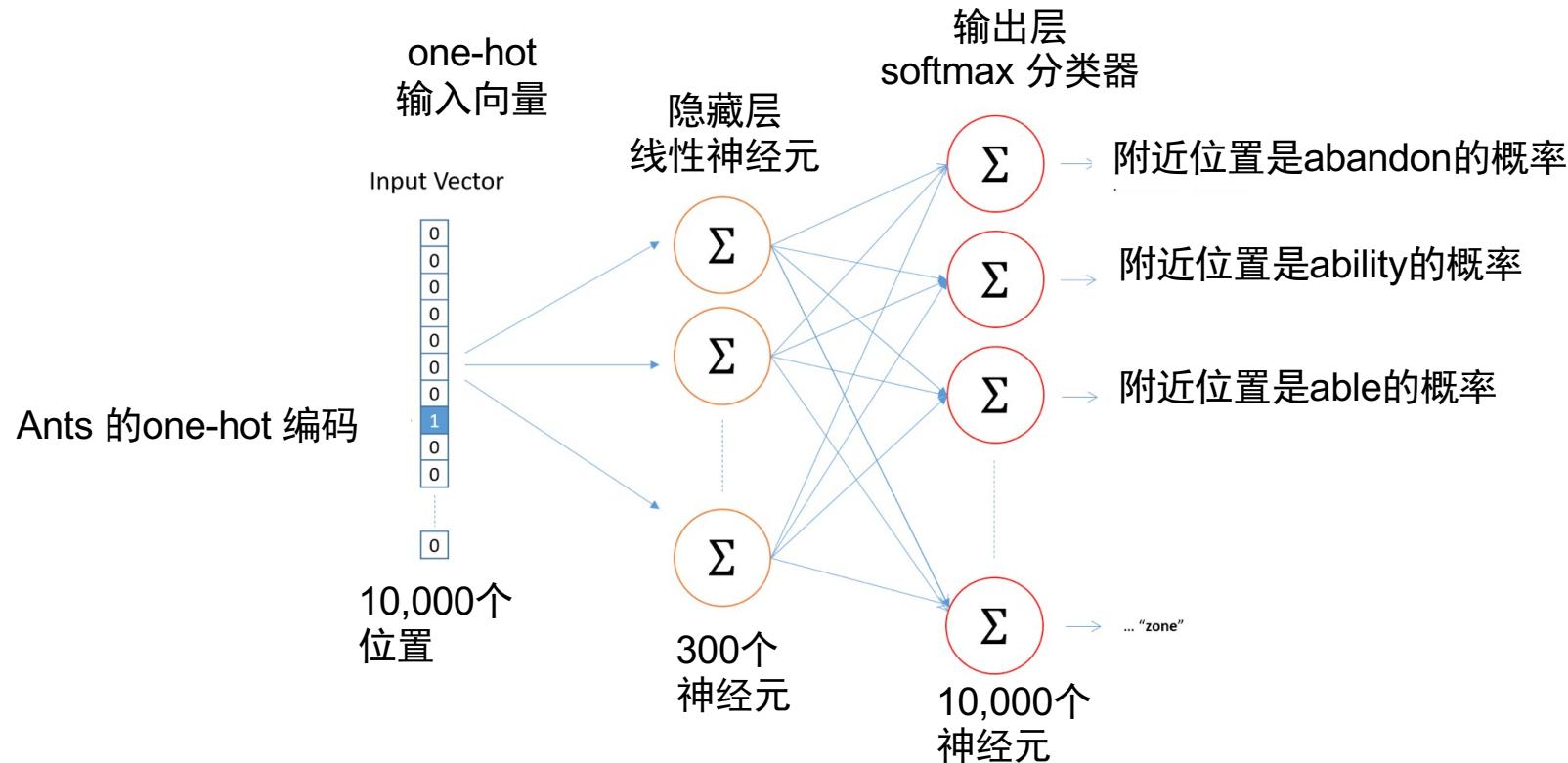


模型隐层权重矩阵，构成词嵌入表，也称为查找表Look up table。

Word2Vec

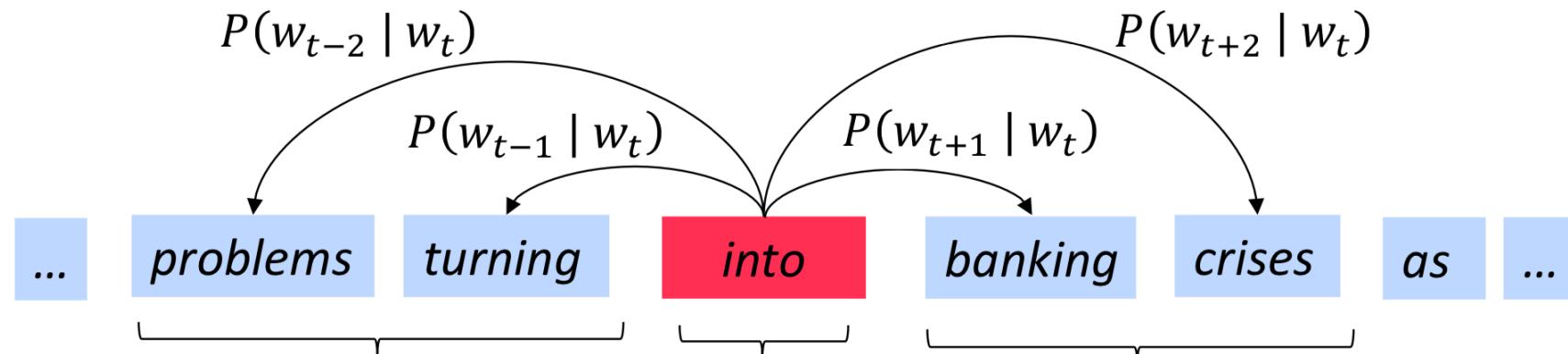
用原始数据构建单词对，单词形式如下 [input word, out word]，即 [data x, label y]。

Source Text	Training Samples
The quick brown fox jumps over the lazy dog.	(the, quick) (the, brown)
The quick brown fox jumps over the lazy dog.	(quick, the) (quick, brown) (quick, fox)
The quick brown fox jumps over the lazy dog.	(brown, the) (brown, quick) (brown, fox) (brown, jumps)
The quick brown fox jumps over the lazy dog.	(fox, quick) (fox, brown) (fox, jumps) (fox, over)



Word2Vec

定义一个单词的概率

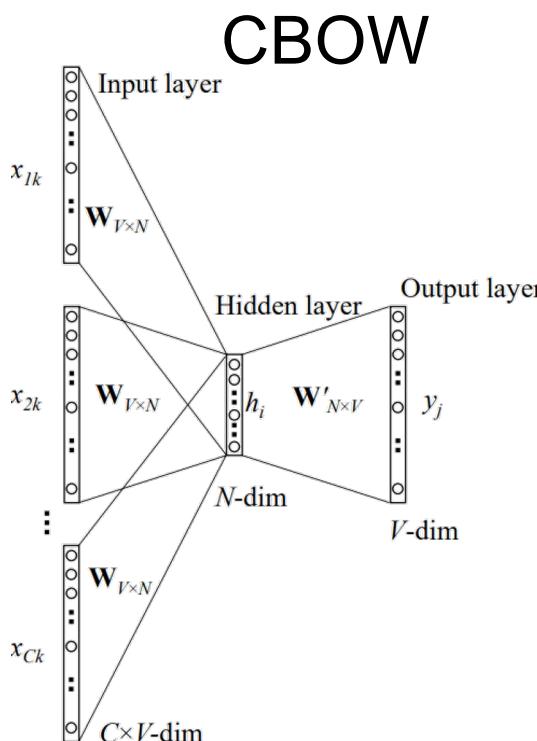


$P(w_{t+1} | w_t)$: 在位置t的中心词 w_{t+1} 的出现概率，后验概率。

给定的词称为“中心词”，临近词称为“背景词”。比如，into为中心词，turning等词就称为背景词。

Word2Vec

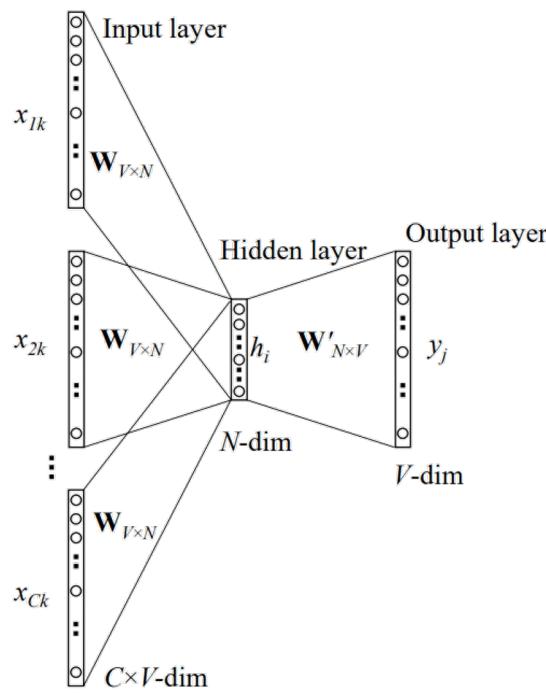
- 连续词袋模型 Continuous bag of words (CBOW): 从上下文单词预测中心词。
 - $P(\text{fox}|\text{quick}, \text{brown}, \text{jumps}, \text{over})$



- 1、输入层：上下文单词的One-Hot编码词向量， V 为词汇表单词个数， C 为上下文单词个数。
- 2、初始化一个权重矩阵 $W_{V \times N}$ ，然后用所有输入的One-Hot编码词向量左乘该矩阵,得到维数为 N 的向量，这里的 N 由自己根据任务需要设置。
- 3、将所得的向量 相加求平均 作为隐藏层向量 h 。
- 4、初始化另一个权重矩阵 $W_{N \times V}$,用隐藏层向量 h 左乘, 再经激活函数处理得到 V 维的向量 y , y 的每一个元素代表相对应的每个单词的概率分布。
- 5、编码词向量做比较, 误差越小越好 (根据误差更新两个权重 y 中概率最大的元素所指示的单词为预测出的中间词 (target word) 与 true label 的 one-hot 矩阵)

Word2Vec

- 连续词袋模型 Continuous bag of words (CBOW): 从上下文单词预测中心词.
 - $P(\text{fox}|\text{quick}, \text{brown}, \text{jumps}, \text{over})$



首先对背景词都进行one-hot处理，每个背景词都得到对应的one-hot向量 x ，然后每个背景词的向量 x 都和矩阵 W 相乘，然后求和取平均，公式如下：

$$\bar{\mathbf{v}} = \frac{1}{C} \sum_{i=1}^C \mathbf{W}^T \mathbf{x}^{(i)} = \frac{1}{C} \sum_{i=1}^C \mathbf{v}_i$$

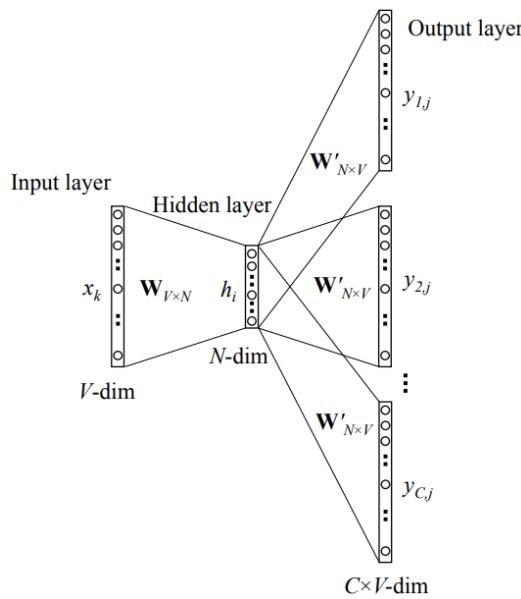
从隐层到输出层和之前的类似，和另一个矩阵 W' 相乘，这里假设中心词下标为 j ，有 $t_j = v^T u_j$ ，其中 u_j 为 W' 的第 j 列 ($N \times 1$)，结果得到一个数 u_k 。然后放入 Softmax 进行概率归一化：

$$p(w_j | w_1, \dots, w_C) = y_j = \frac{\exp(\bar{\mathbf{v}}^T \mathbf{u}_j)}{\sum_{k=1}^V \exp(\bar{\mathbf{v}}^T \mathbf{u}_k)}$$

Word2Vec

- Skip-grams (SG): 给定中心单词预测上下文单词.
 - $P(\text{quick}, \text{brown}, \text{jumps}, \text{over} | \text{fox}) = P(\text{quick} | \text{fox}) \cdot P(\text{brown} | \text{fox}) \cdot P(\text{jumps} | \text{fox}) \cdot P(\text{over} | \text{fox})$

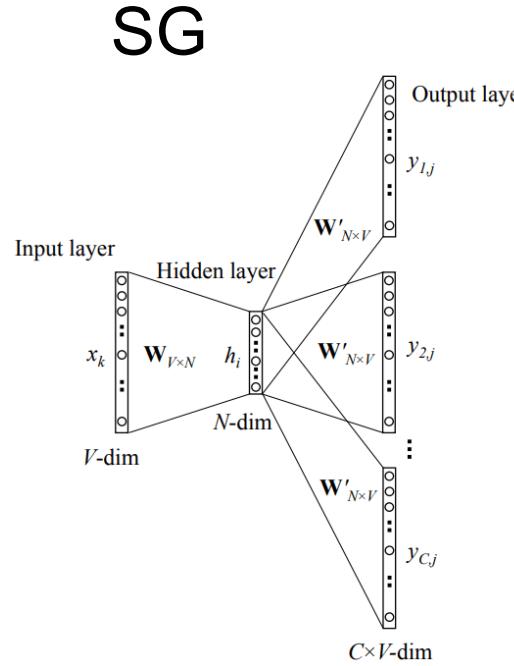
SG



1. 选输入词，选输出词，我们将会得到两组 (input word, output word) 形式的训练数据，即 ('apple', 'an') , ('apple', 'one')。
3. 假如我们先拿一组数据 ('dog', 'barked') 来训练神经网络，那么模型通过学习这个训练样本，会告诉我们词汇表中每个单词是 “barked”的概率大小。

Word2Vec

- Skip-grams (SG): 给定中心单词预测上下文单词.
 - $P(\text{quick}, \text{brown}, \text{jumps}, \text{over} | \text{fox}) = P(\text{quick} | \text{fox}) \cdot P(\text{brown} | \text{fox}) \cdot P(\text{jumps} | \text{fox}) \cdot P(\text{over} | \text{fox})$



1. 输入是one-hot后的向量 x , 维度为 $V \times 1$ 。然后是矩阵 W , 维度为 $V \times N$, 则输入到隐层的操作其实是矩阵的相乘, $v_k = W^T x = W^T_{k,:}$, 即相乘后其实得到的是矩阵 W^T 中的第k列 (称为 v_k), 其维度为 $N \times 1$ 。

2. 从隐层到输出层则是另一个矩阵 W' , 这个矩阵维度为 $N \times V$, 设 $t_{c,j} = v_k^T u_j$, 其中 u_j 为 W' 的第j列($N \times 1$), 结果得到一个数 $u_{c,j}$ 。这里 $u_{c,j}$ 代表输出的第c个面板的第j个单词的值 (因为要输出C个结果)。放入Softmax进行概率归一化:

$$p(w_{c,j} = w_{O,c} | w_I) = \hat{y}_{c,j} = \frac{\exp(t_{c,j})}{\sum_{a=1}^V \exp(t_{c,a})}$$

$w_{c,j}$ 为第c个背景词输出向量的第j个单词的值, 而 $w_{O,c}$ 为背景词中的第c个单词, w_I 为输入的单词。

合起来就是说给定单词 w_I , 第c个背景词为第j个词的概率。

Word2Vec

损失函数

- 但如何计算 $P(w_{t+j}|w_t; \theta)$?

每个单词使用两个向量: 当w是中心单词时用 v_w ; 当w是上下文单词时用 u_w . 对中心单词w=c 和上下文单词 w=o,

$$P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)}$$

- 通过将序列中的所有单词概率用极大似然估计表示为:

$$L(\theta) = \prod_{t=1}^T \prod_{j=-m, j \neq 0}^m P(w_{t+j}|w_t; \theta)$$

- 与最小化负对数似然函数等价

$$J(\theta) = -\frac{1}{T} \sum_{t=1}^T \sum_{j=-m, j \neq 0}^m \log P(w_{t+j}|w_t; \theta)$$

θ 是所有要优化的变量.
 m 是上下文单词窗口大小



Pytorch实现：词向量

```
# Author: Robert Guthrie

import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim

torch.manual_seed(1)

<torch._C.Generator object at 0x7f1bbf974030>

word_to_ix = {"hello": 0, "world": 1}
embeds = nn.Embedding(2, 5) # 2 words in vocab, 5 dimensional embeddings
lookup_tensor = torch.tensor([word_to_ix["hello"]], dtype=torch.long)
hello_embed = embeds(lookup_tensor)
print(hello_embed)

tensor([[ 0.6614,  0.2669,  0.0617,  0.6213, -0.4519]], grad_fn=<EmbeddingBackward0>)
```



Pytorch实现：SG模型

```
CONTEXT_SIZE = 2
EMBEDDING_DIM = 10
# We will use Shakespeare Sonnet 2
test_sentence = """When forty winters shall besiege thy brow, And dig deep trenches in thy beauty's field,
Thy youth's proud livery so gazed on now,
Will be a totter'd weed of small worth held:
Then being asked, where all thy beauty lies,
Where all the treasure of thy lusty days;
To say, within thine own deep sunken eyes,
Were an all-eating shame, and thriftless praise.
How much more praise deserv'd thy beauty's use,
If thou couldst answer 'This fair child of mine Shall sum my count, and make my old excuse,'  

Proving his beauty by succession thine!
This were to be new made when thou art old,  

And see thy blood warm when thou feel'st it cold.""".split()
# we should tokenize the input, but we will ignore that for now build a List of tuples.
# Each tuple is ([ word_i-CONTEXT_SIZE, ..., word_i-1 ], target word)
ngrams = [
    (
        [test_sentence[i - j - 1] for j in range(CONTEXT_SIZE)],
        test_sentence[i]
    )
    for i in range(CONTEXT_SIZE, len(test_sentence))
]
# Print the first 3, just so you can see what they look like.
print(ngrams[:3])
vocab = set(test_sentence)
word_to_ix = {word: i for i, word in enumerate(vocab)}
```

```
[(['forty', 'When'], 'winters'),
(['winters', 'forty'], 'shall'),
(['shall', 'winters'], 'besiege')]
```



Pytorch实现：SG模型

```
class NGramLanguageModeler(nn.Module):

    def __init__(self, vocab_size, embedding_dim, context_size):
        super(NGramLanguageModeler, self).__init__()
        self.embeddings = nn.Embedding(vocab_size, embedding_dim)
        self.linear1 = nn.Linear(context_size * embedding_dim, 128)
        self.linear2 = nn.Linear(128, vocab_size)

    def forward(self, inputs):
        embeds = self.embeddings(inputs).view((1, -1))
        out = F.relu(self.linear1(embeds))
        out = self.linear2(out)
        log_probs = F.log_softmax(out, dim=1)
        return log_probs

losses = []
loss_function = nn.NLLLoss()
model = NGramLanguageModeler(len(vocab),
                             EMBEDDING_DIM, CONTEXT_SIZE)
optimizer = optim.SGD(model.parameters(), lr=0.001)

# 2D Loss example (used, for example, with image
# inputs)
N, C = 5, 4
loss = nn.NLLLoss()
# input is of size N x C x height x width
data = torch.randn(N, 16, 10, 10)
conv = nn.Conv2d(16, C, (3, 3))
m = nn.LogSoftmax(dim=1)
# each element in target has to have 0 <= value < C
target = torch.empty(N, 8, 8,
                     dtype=torch.long).random_(0, C)
output = loss(m(conv(data)), target)
output.backward()
```



Pytorch实现：SG模型

```
for epoch in range(10):
    total_loss = 0
    for context, target in ngrams:

        # Step 1. Prepare the inputs to be passed to the model (i.e., turn the words
        # into integer indices and wrap them in tensors)
        context_idxs = torch.tensor([word_to_ix[w] for w in context], dtype=torch.long)

        # Step 2. Recall that torch *accumulates* gradients. Before passing in a
        # new instance, you need to zero out the gradients from the old instance
        model.zero_grad()

        # Step 3. Run the forward pass, getting log probabilities over next words
        log_probs = model(context_idxs)

        # Step 4. Compute your Loss function. (Again, Torch wants the target word wrapped in a tensor)
        loss = loss_function(log_probs, torch.tensor([word_to_ix[target]]], dtype=torch.long))

        # Step 5. Do the backward pass and update the gradient
        loss.backward()
        optimizer.step()

        # Get the Python number from a 1-element Tensor by calling tensor.item()
        total_loss += loss.item()
        losses.append(total_loss)
    print(losses) # The loss decreased every iteration over the training data

    # To get the embedding of a particular word, e.g. "beauty"
    print(model.embeddings.weight[word_to_ix["beauty"]])
```

[523.3628234863281, 520.9973816871643,
518.6471247673035, 516.3107891082764,
513.9880723953247, 511.67914295196533,
509.38300490379333, 507.09720492362976,
504.82066893577576, 502.5510606765747]



Pytorch实现： CBOW模型

- 请大家思考一下如何实现？



词向量

- 包括但不限于：
 - 计算相似度
 - 寻找相似词
 - 信息检索
 - 作为 SVM/LSTM 等模型的输入
 - 中文分词
 - 命名体识别
 - 句子表示
 - 情感分析
 - 文档表示
 - 文档主题判别

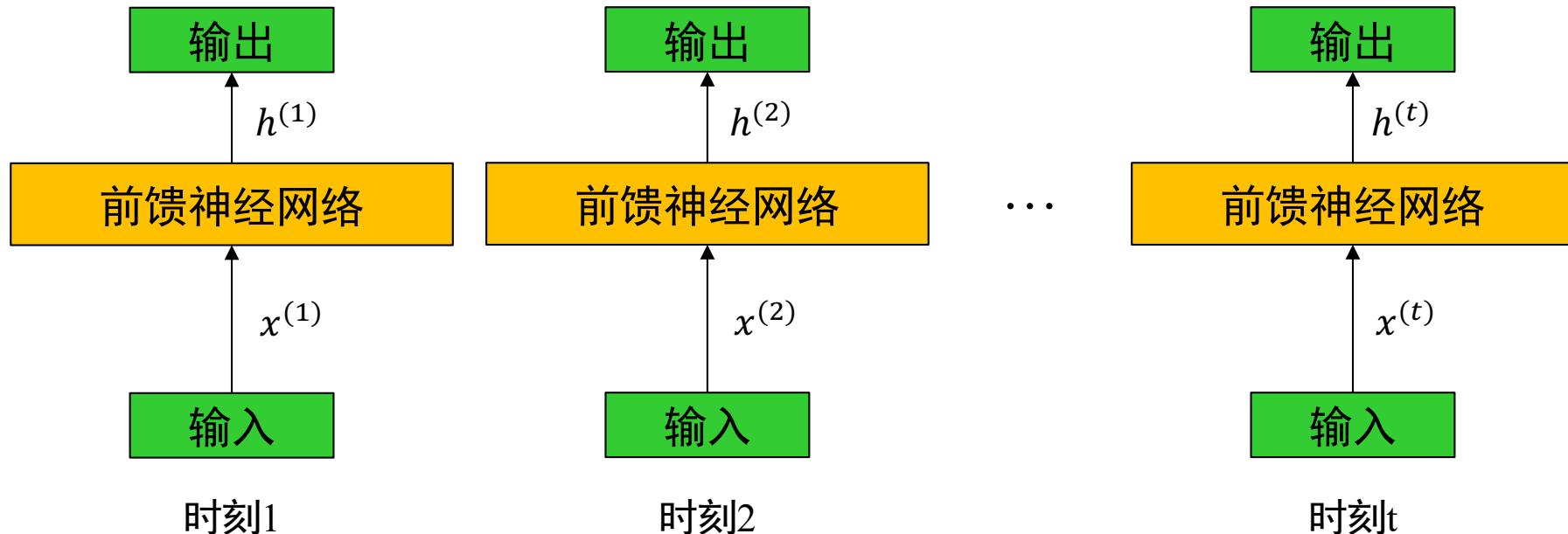
这节课

- 自然语言处理 (NLP)
- 词向量
- 循环神经网络
- 循环神经网络语言模型



原始神经网络模型

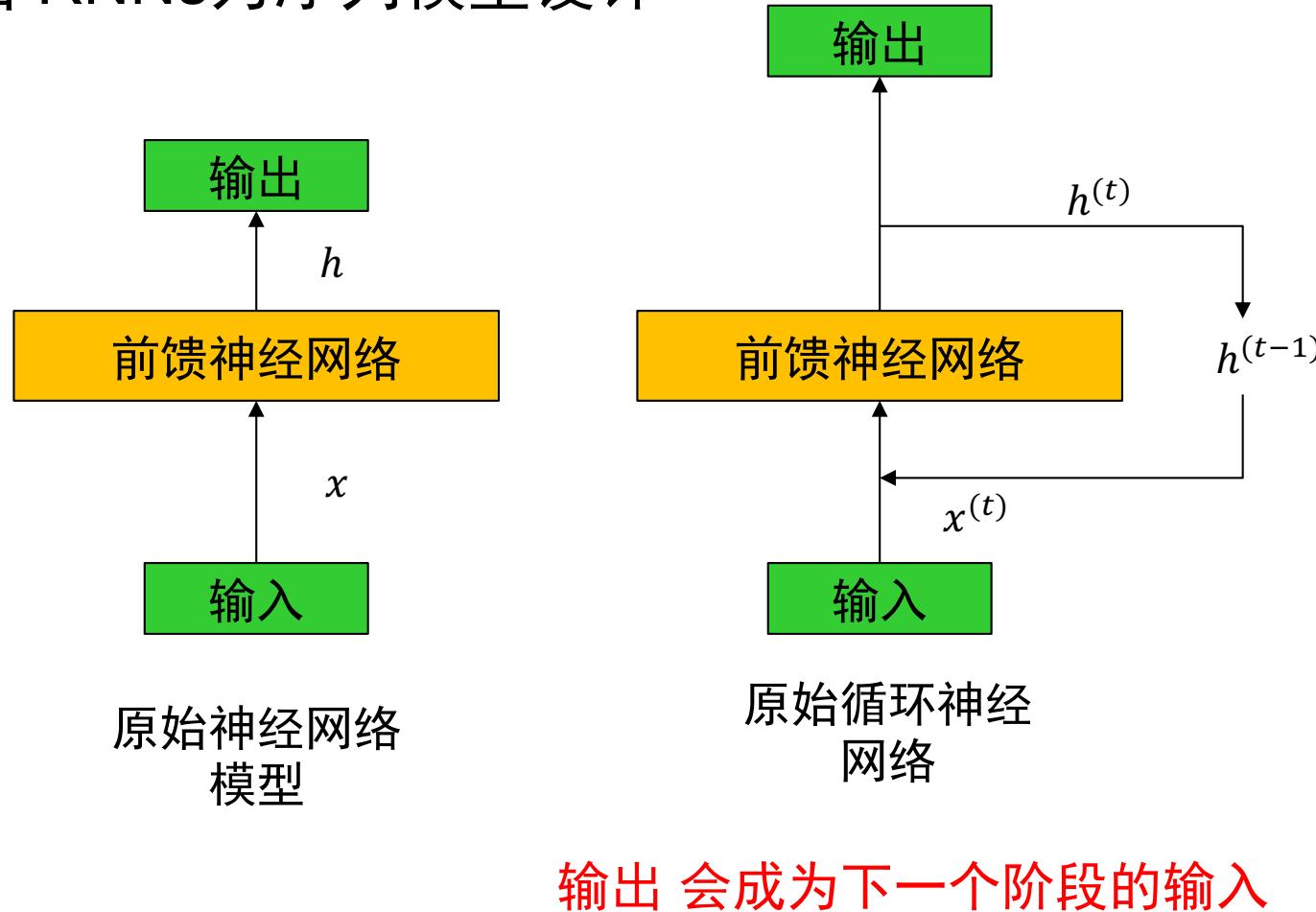
- 原始 NN不能很好的处理序列数据.
- 他们必须在时间维度对每一帧的内容进行处理。



原始神经网络模型无法在时间维度之间共享信息. 所以在处理序列数据时容易过拟合。

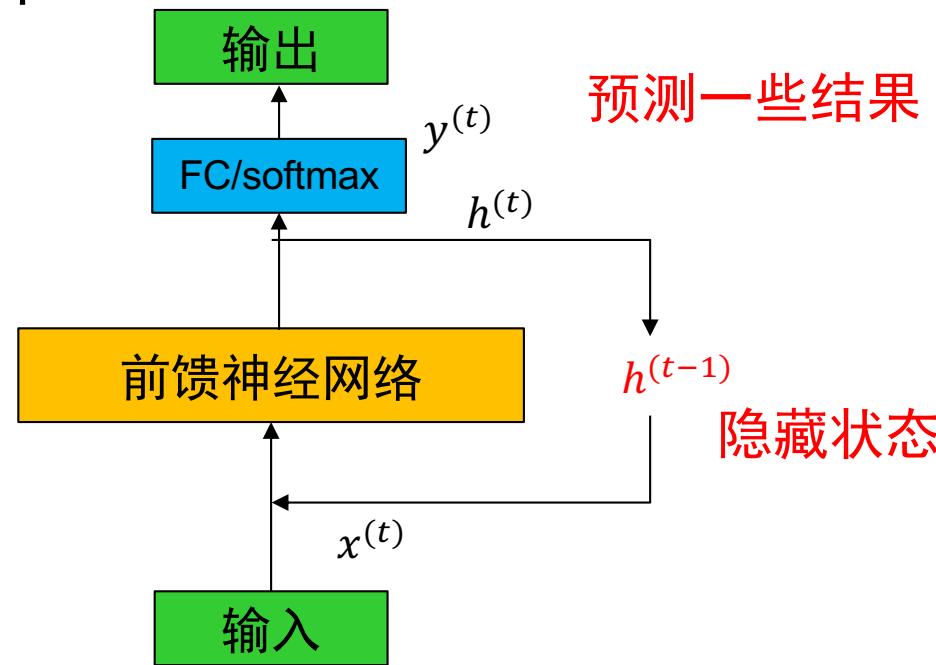
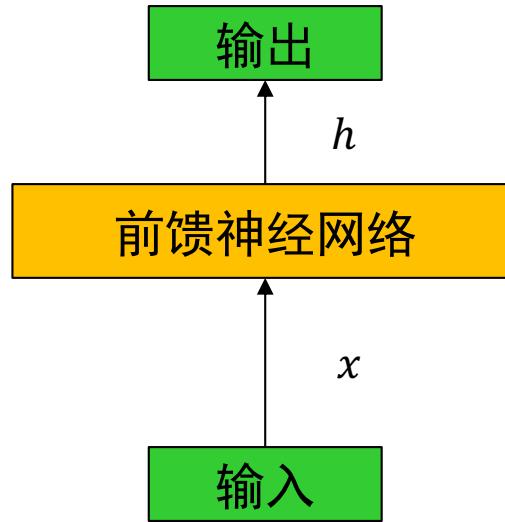
原始的循环神经网络 (RNN)

- 原始 RNNs 为序列模型设计



原始的循环神经网络 (RNN)

- 原始 RNNs 为序列模型设计

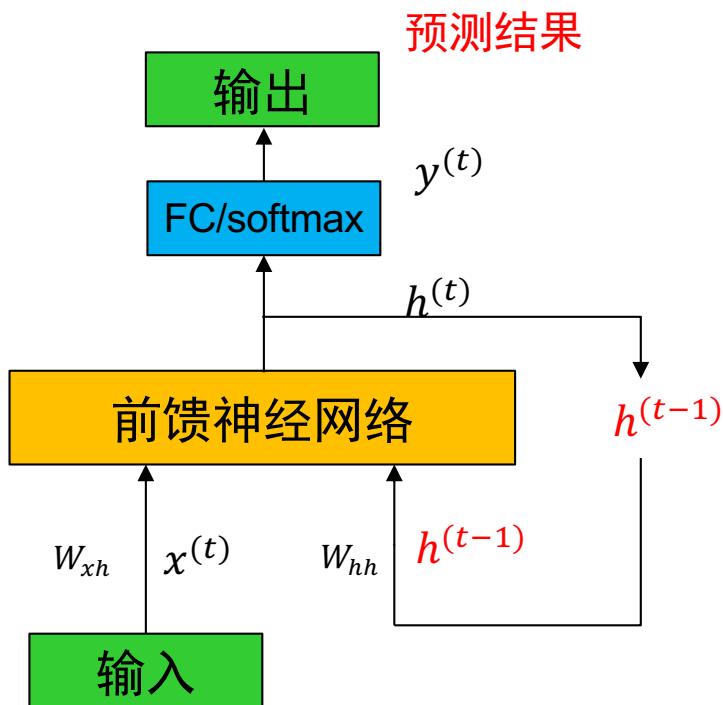


输出会成为下一个阶段的输入

原始的循环神经网络 (RNN)

RNN 前向传播

前一个时间戳的隐藏状态 $h^{(t-1)}$ 用来生成当前时刻的隐藏状态 $h^{(t)}$.



前向传播

$$h^{(t)} = f_W(h^{(t-1)}, x)$$

随着时间序列，计算出激活值
 $h^{(1)}, h^{(2)}, \dots, h^{(t)}$.

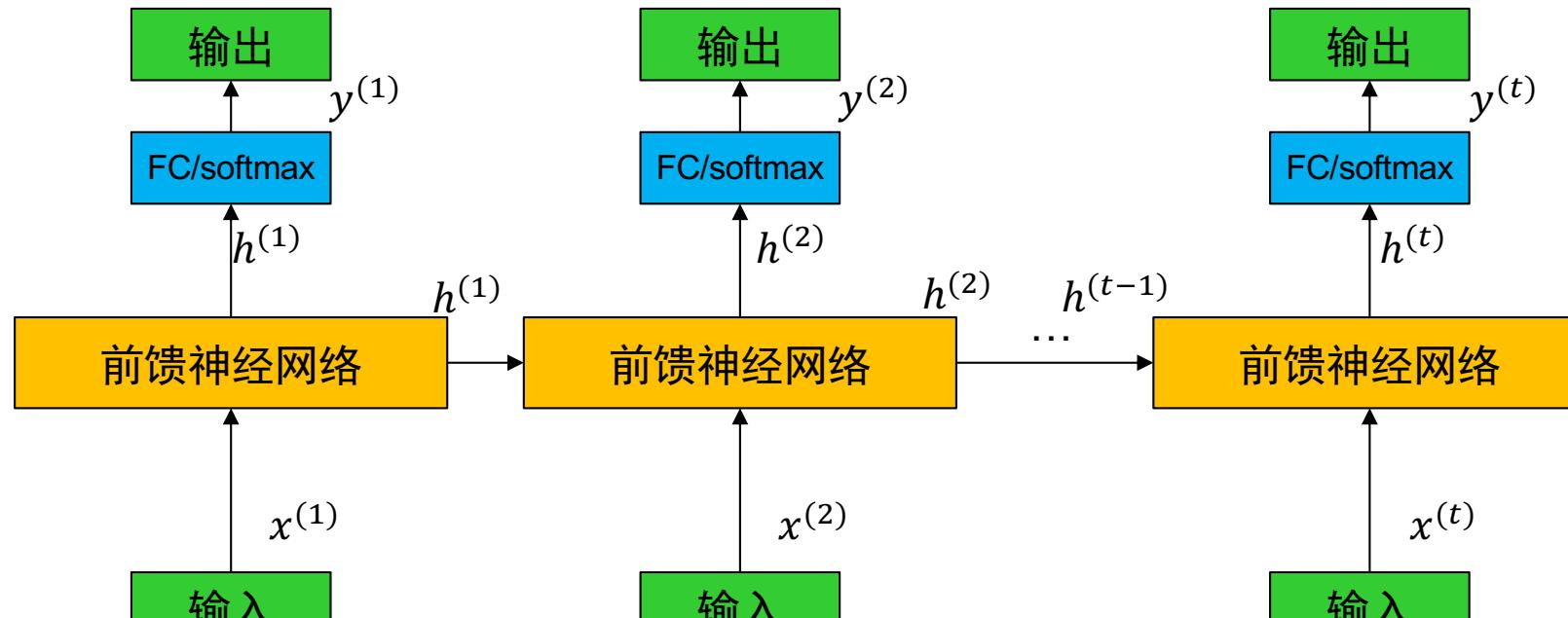
$$h^{(t)} = \tanh(W_{xh}x^{(t)} + W_{hh}h^{(t-1)} + b_h)$$

$$y^{(t)} = \sigma(W_{hy}h^{(t)} + b_y)$$

Vanilla RNN

展开 RNN的计算过程

- 在时间维度上展开RNN.

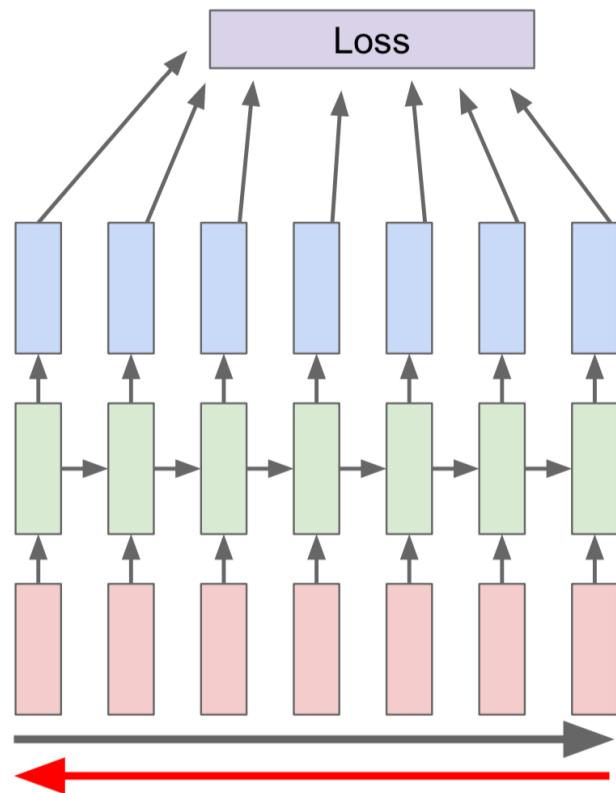


$$h^t = f_W(h_{t-1}, x)$$

Vanilla RNN: BPTT

原始RNN的反向传播

- 基于时间的反向传播BPTT



序列的前向来计算loss，整个序列的后向来计算梯度

```
Back_Propagation_Through_Time(a, y)    // a[t] is the input at time t. y[t] is the output
  Unfold the network to contain k instances of f
  do until stopping criteria is met:
    x = the zero-magnitude vector; // x is the current context
    for t from 0 to n - k          // t is time. n is the length of the training sequence
      Set the network inputs to x, a[t], a[t+1], ..., a[t+k-1]
      p = forward-propagate the inputs over the whole unfolded network
      e = y[t+k] - p;             // error = target - prediction
      Back-propagate the error, e, back across the whole unfolded network
      Sum the weight changes in the k instances of f together.
      Update all the weights in f and g.
    x = f(x, a[t]);              // compute the context for the next time-step
```

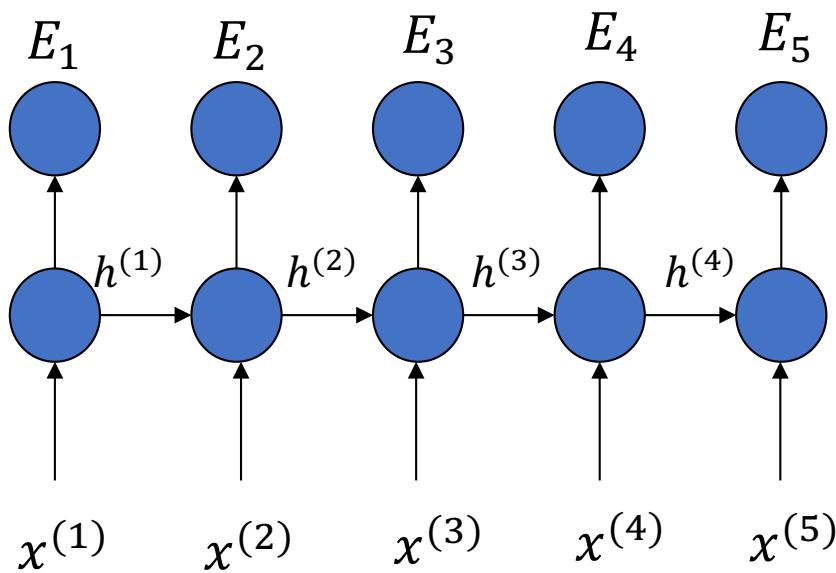


Source: https://en.wikipedia.org/wiki/Backpropagation_through_time

Credit to Stanford CS224N

梯度爆炸和消失

$t = 5$ 时间步长的RNN



$$\frac{\partial E_t}{\partial W} = \sum_{k=0}^t \frac{\partial E_t}{\partial h^{(t)}} \prod_{j=k+1}^t \left(\frac{\partial h^{(j)}}{\partial h^{(j-1)}} \right) \frac{\partial h^{(k)}}{\partial W}$$

避免梯度消失?

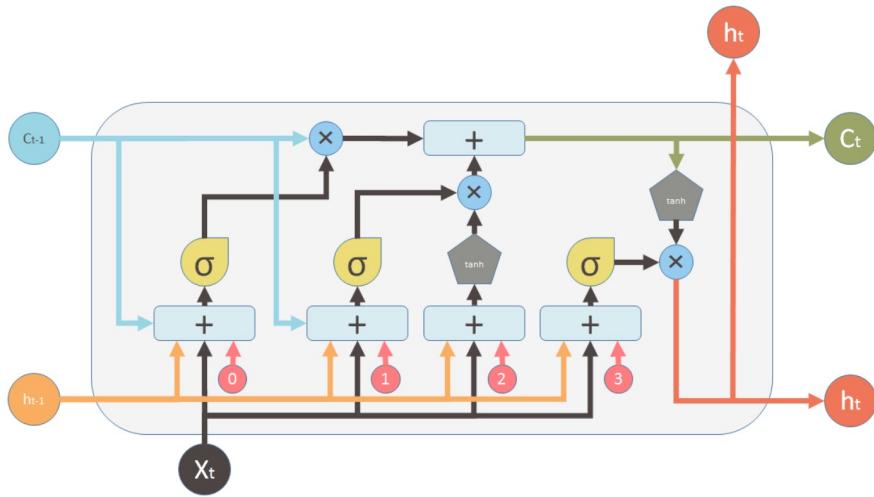
- 用 ReLU
- 用 LSTM

由于RNN的复杂特性，用 RMSprop 优化器来训练RNN模型。

由于激活函数 (sigmoid, tanh) 的特性，很容易遇到梯度爆炸 / 消失问题。

长短期记忆网络(LSTM)

- LSTM 用来解决“梯度消失”的问题。记忆单元(memory cell) 和输入/遗忘/输出门 让LSTM有历史信息并能更准确的预测。



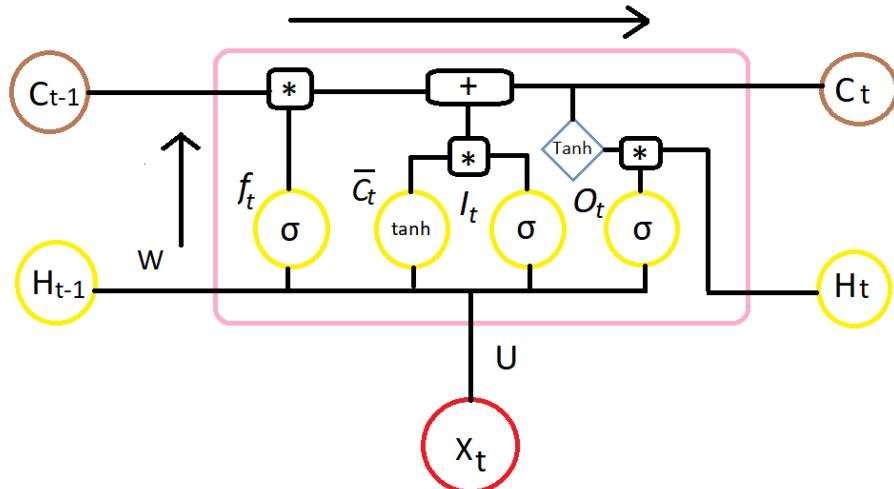
Inputs:	outputs:	Nonlinearities:	Vector operations:
X_t Input vector	C_t Memory from current block	σ Sigmoid	\times Element-wise multiplication
C_{t-1} Memory from previous block	h_t Output of current block	\tanh Hyperbolic tangent	$+$ Element-wise Summation / Concatenation
h_{t-1} Output of previous block		Bias: 0	

但是...

- LSTM 需要大量的训练数据.
- LSTM 花费大量时间训练，而且模型规模很大.



长短期记忆网络(LSTM)



LSTM 会搞清楚多少历史信息需要保留.

遗忘门

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

输入门

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

记忆单元

$$\begin{aligned}\bar{C}_t &= \tanh(W_c[h_{t-1}, x_t] + b_c) \\ C_t &= f_t \circ C_{t-1} + i_t \circ \bar{C}_t\end{aligned}$$

输出门/预测

$$\begin{aligned}o_t &= \sigma(W_o[h_{t-1}, x_t] + b_o) \\ h_t &= o_t \circ \tanh(C_t)\end{aligned}$$

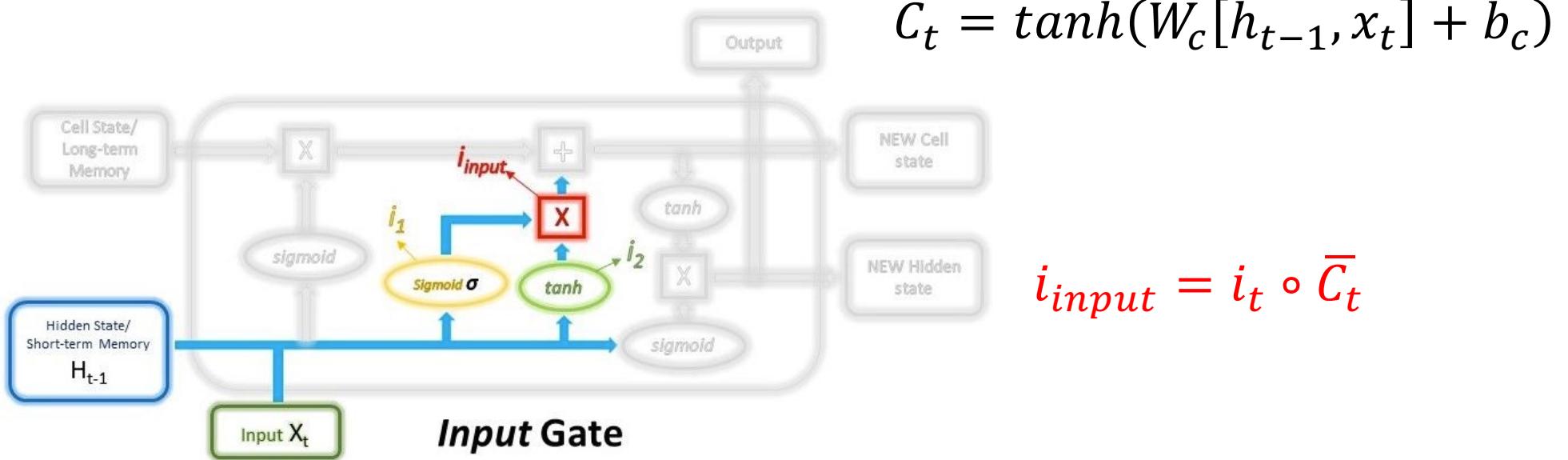


输入入门

- 哪些信息需要存储在长期记忆中
- 两个输入：一个是当前输入，一个是上一个time step 的状态。
- 操作：把不必要的信息过滤掉

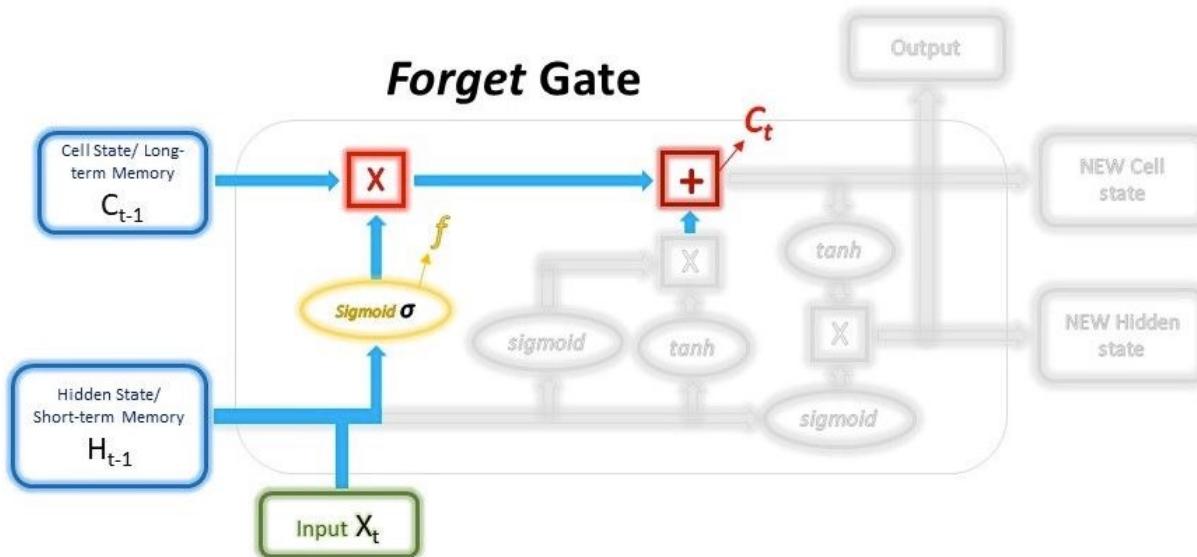
$$i_t = \sigma(W_i[h_{t-1}, x_t] + b_i)$$

$$\bar{C}_t = \tanh(W_c[h_{t-1}, x_t] + b_c)$$



遗忘门

- 来自长期记忆的哪些信息需要保留哪些需要丢弃。
- 两个输入：一个是长期记忆 C_{t-1} ，一个是forget vector；
- 其中forget vector 的得到与input gate 类似。



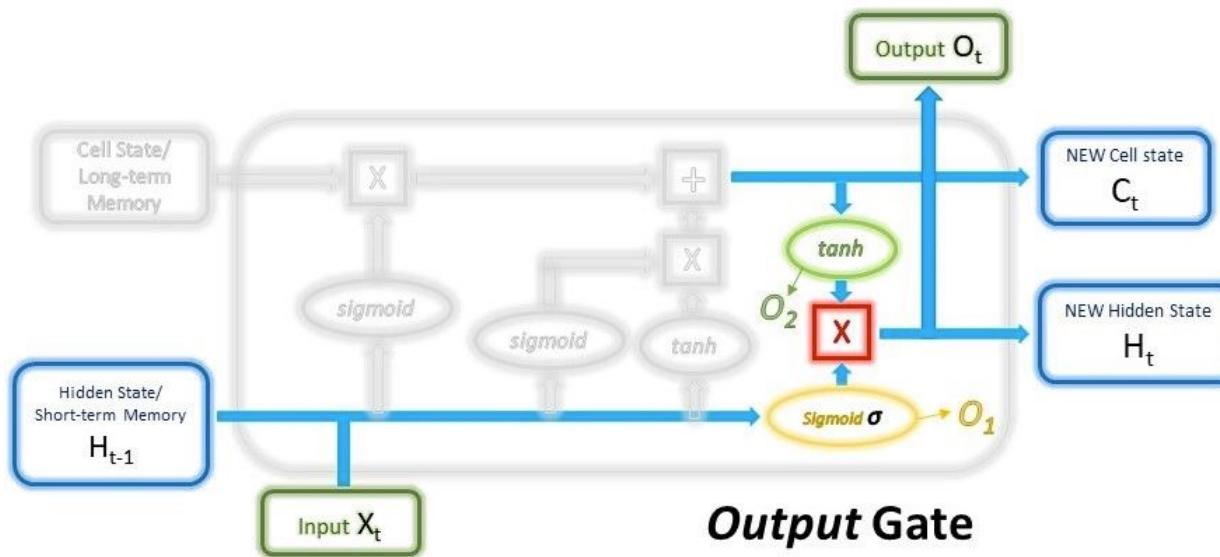
$$f_t = \sigma(W_f [h_{t-1}, x_t] + b_f)$$

记忆单元

$$C_t = f_t \circ C_{t-1} + i_{input}$$

输出入门

- 哪些内部状态作为输出；
- 产生新的短期记忆，即隐藏状态。
- 输入：新生成的长期记忆 C_t ，这次的输出。

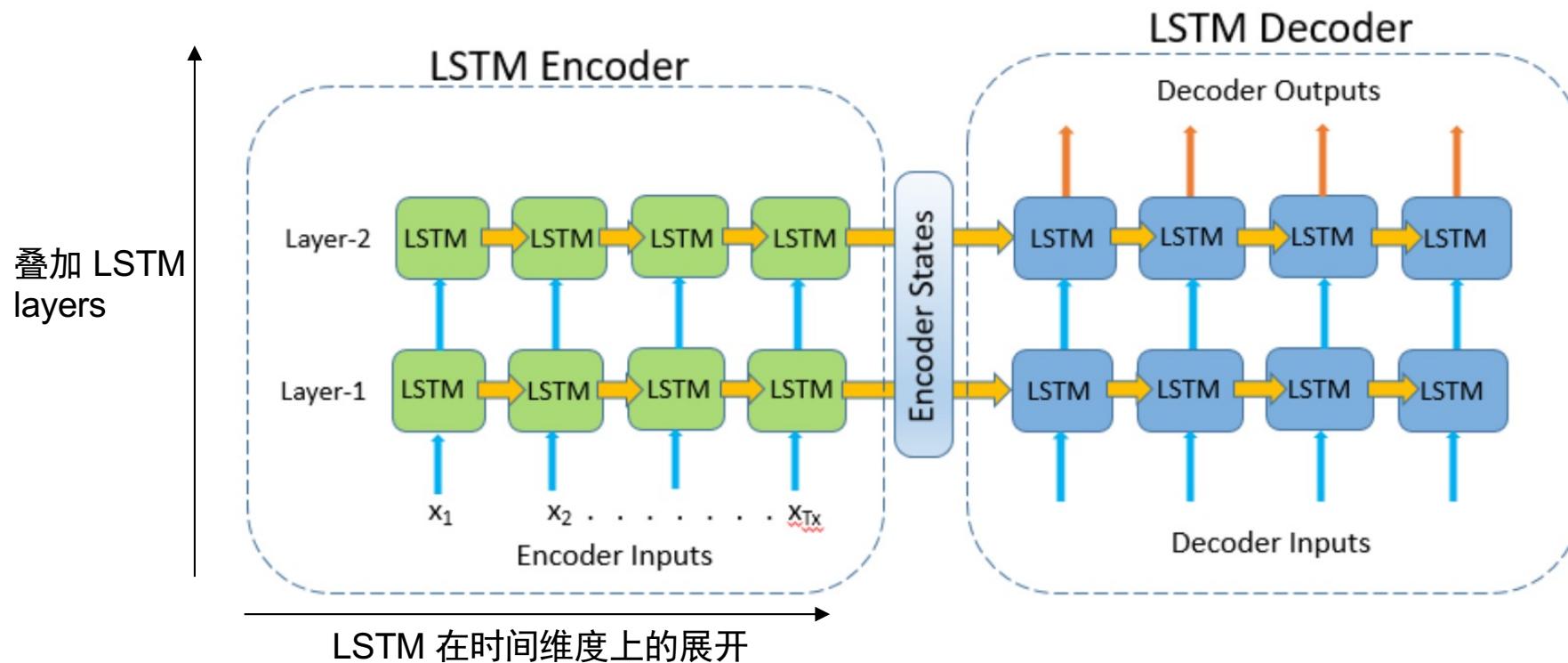


$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o)$$

$$h_t = o_t \circ \tanh(C_t)$$

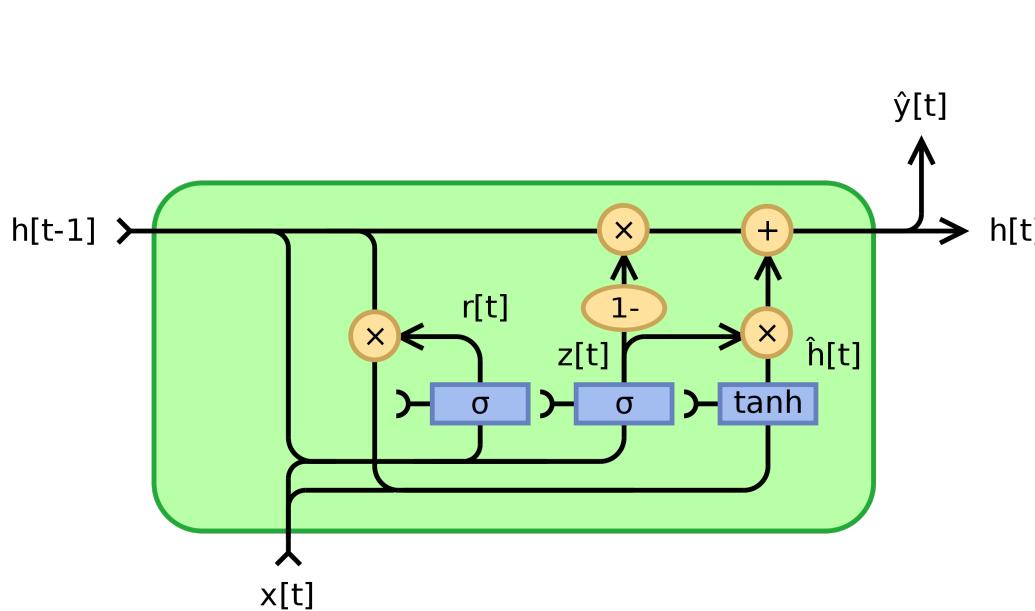
长短期记忆网络(LSTM)

- 对于语言任务，通常是多个LSTM模型合并构建成更深的LSTM网络。



Gated recurrent unit (GRU)

- LSTM单元的简单版本，能学到更有意义的单词表示.



更新门

$$z_t = \sigma(W^{(z)}x_t + U^{(z)}h_{t-1})$$

重置门

$$r_t = \sigma(W^{(r)}x_t + U^{(r)}h_{t-1})$$

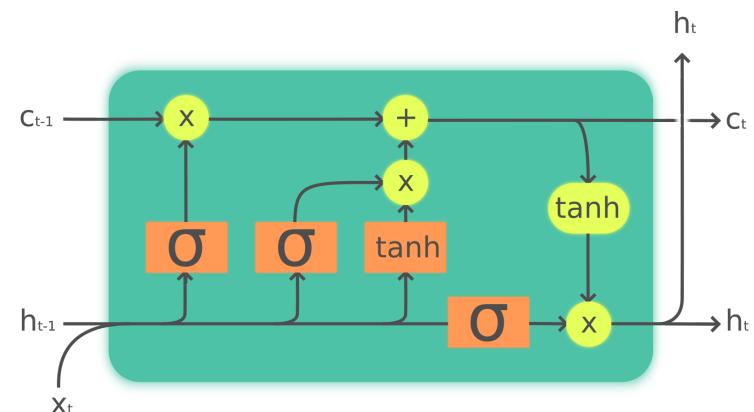
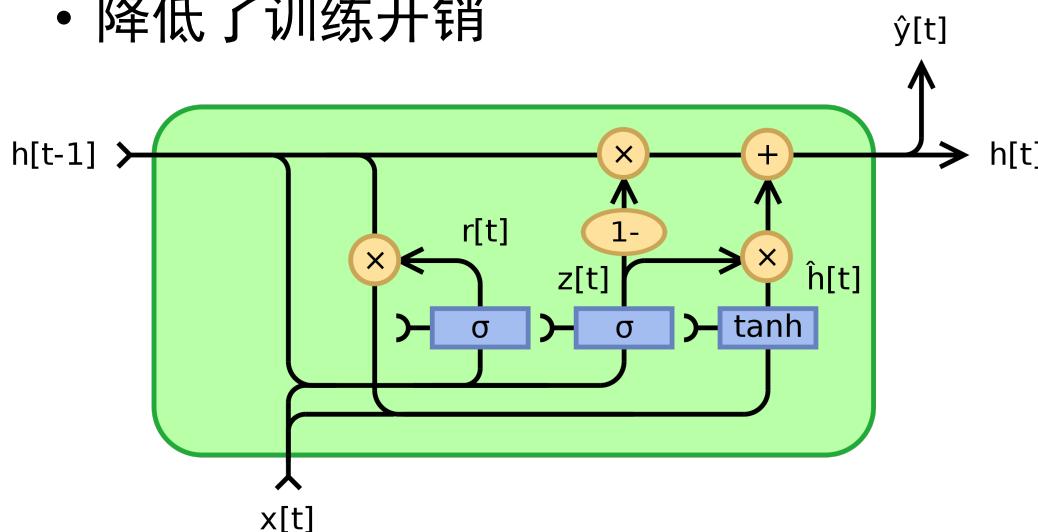
记忆内容

$$\begin{aligned}\hat{h}_t &= \sigma(W^{(h)}x_t + r_t U^{(h)}h_{t-1}) \\ h_t &= (1 - z_t) \circ h_{t-1} + z_t \circ \hat{h}_t\end{aligned}$$

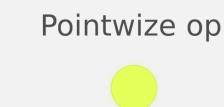
Gated recurrent unit (GRU)

- GRU 是LSTM的改进:

- 输入/遗忘门合并成一个更新门
- 没有额外的记忆单元 (c_t)
- 简化了控制依赖, 用更少的门操作
- 降低了训练开销

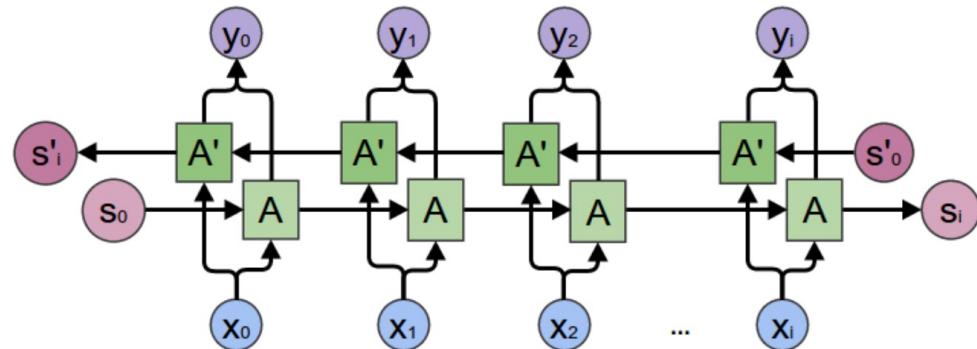


Legend:

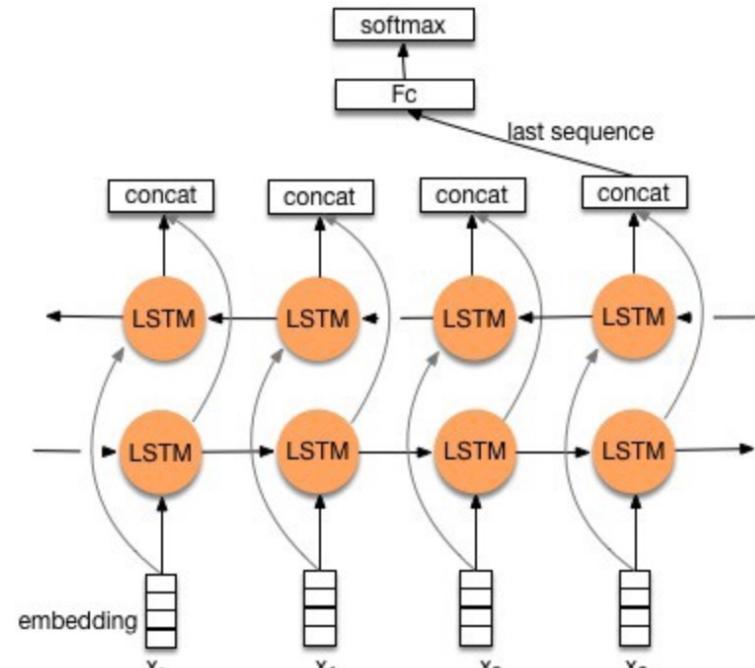


双向 RNNs

- Bidirectional RNN 在预测过程中同时连接前向隐藏状态和后向隐藏状态.
- 它将两个独立的循环模型放在一起，并在相反的方向执行两个正向传递



Bidirectional vanilla RNN



Bidirectional LSTM



这节课

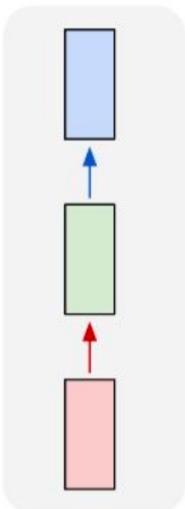
- 自然语言处理 (NLP)
- 词向量
- 循环神经网络
- 循环神经网络语言模型



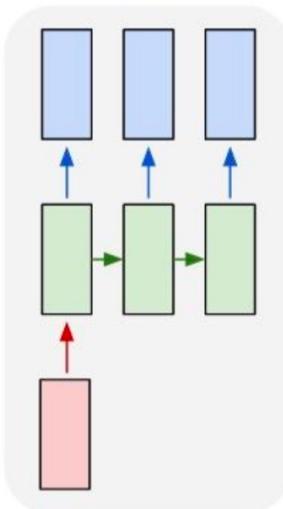
RNN 模型

RNN 的任务类型

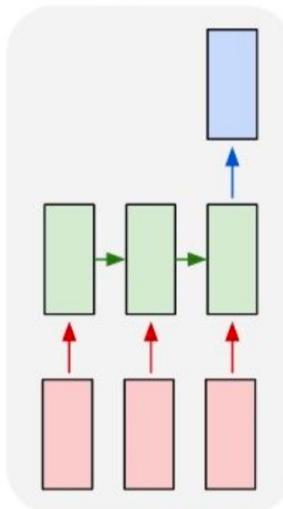
one to one



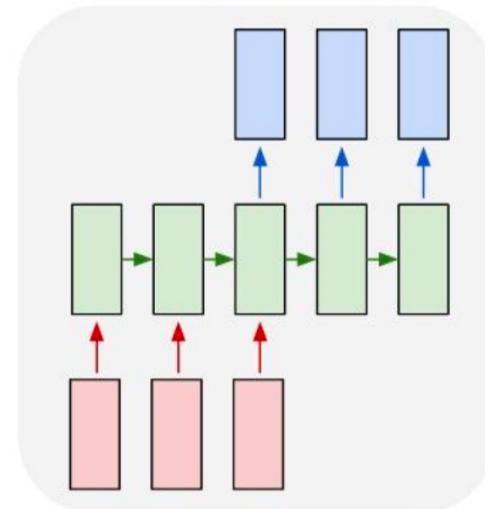
one to many



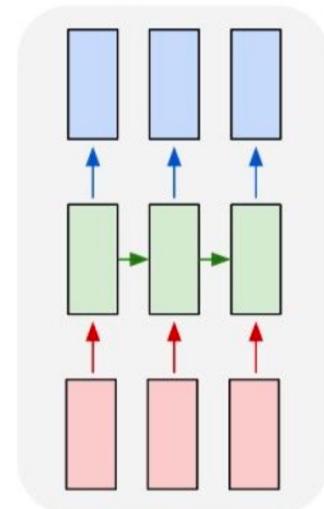
many to one



many to many



many to many



无序列模型

序列生成
图片加标题

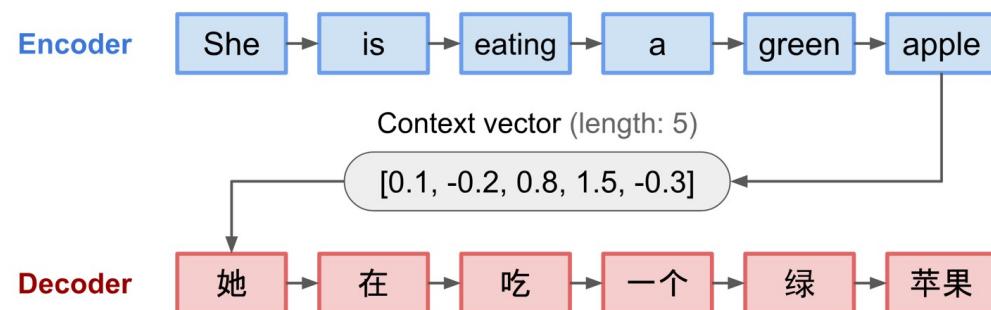
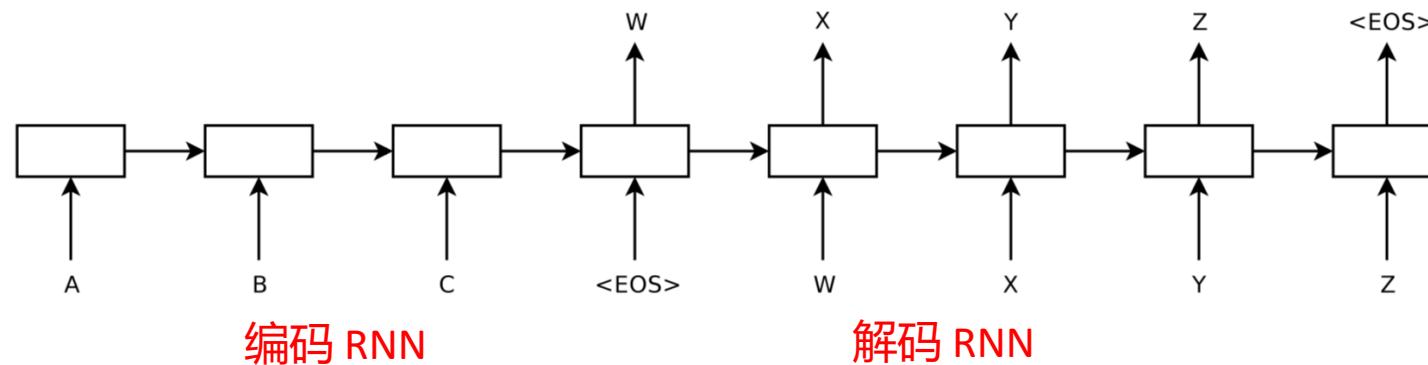
情感分析

神经网络机器翻译



RNN 模型

• 编码解码模型(Seq2Seq)



核心思想:

1. 用RNN把输入序列映射为固定大小的向量.
2. 把向量用另外的RNN映射为目标序列



RNN 模型

编码-解码模型 结果

Method	test BLEU score (ntst14)
Bahdanau et al. [2]	28.45
Baseline System [29]	33.30
Single forward LSTM, beam size 12	26.17
Single reversed LSTM, beam size 12	30.59
Ensemble of 5 reversed LSTMs, beam size 1	33.00
Ensemble of 2 reversed LSTMs, beam size 12	33.27
Ensemble of 5 reversed LSTMs, beam size 2	34.50
Ensemble of 5 reversed LSTMs, beam size 12	34.81

- 但编码-解码模型在很长的句子上效果不好



NLP 评价指标

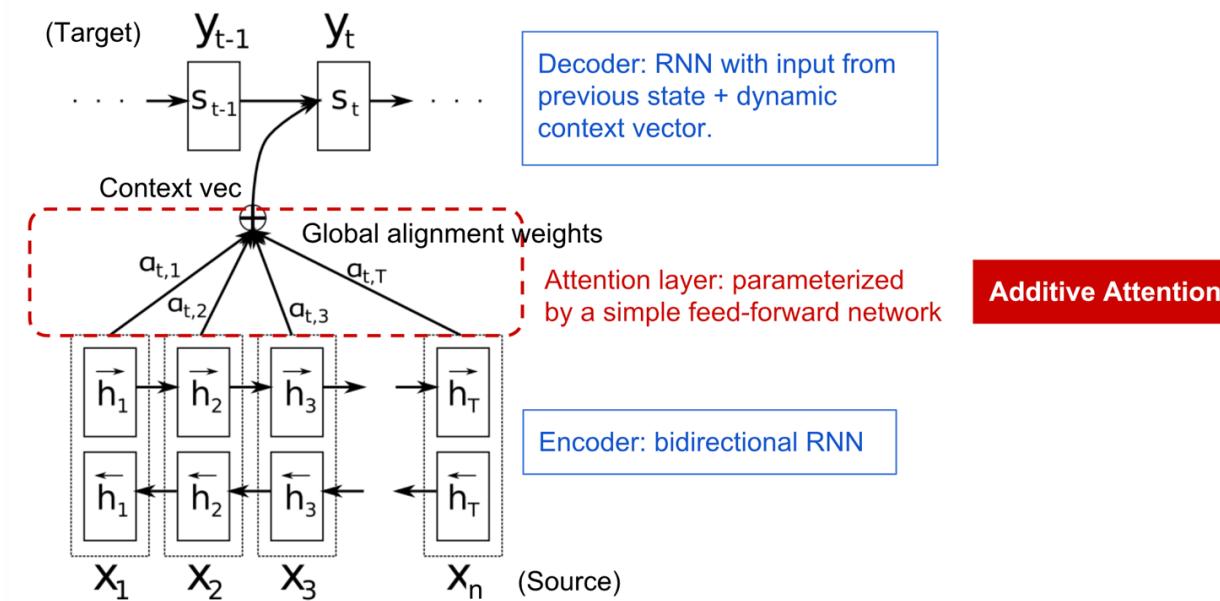
- **BLEU (Bilingual evaluation understudy)**

- BLEU 用于评估经过机器翻译的文本的质量。
- 比较候选译文和参考译文里的重合程度，重合程度越高就认为译文质量越高。
- 考虑文本的长度，会偏向于较短的翻译结果。
- See <https://en.wikipedia.org/wiki/BLEU> for details.
- 用于翻译，语言生成、图片标题生成、文本摘要、语音识别。



注意力模型

- 传统的编码器/解码器系统会因长句子而遭受性能损失。我们通常对句子的不同部分给予不同的关注。
- 在注意力机制中，我们在编码和解码层之间增加一个对齐层，来连接源和目标的含义。



She is eating a green apple.
high attention
low attention

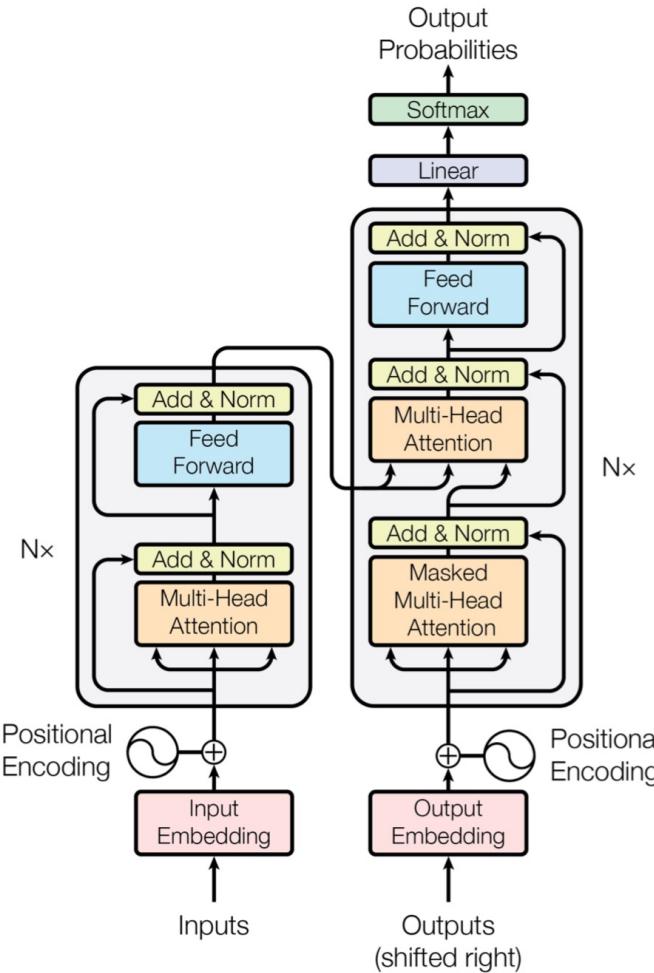
‘对齐’层为句子的每一部分提供了一个可以训练的变换因子 α .

会对从编码器出来的向量乘上权重再输入给解码器。

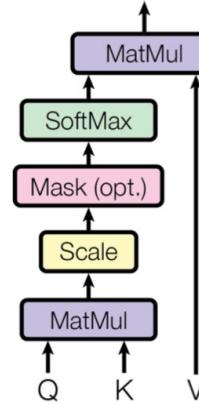


注意力模型

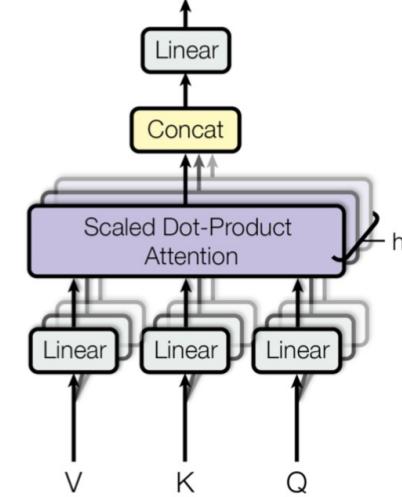
变换, Transformer



Scaled Dot-Product Attention



Multi-Head Attention

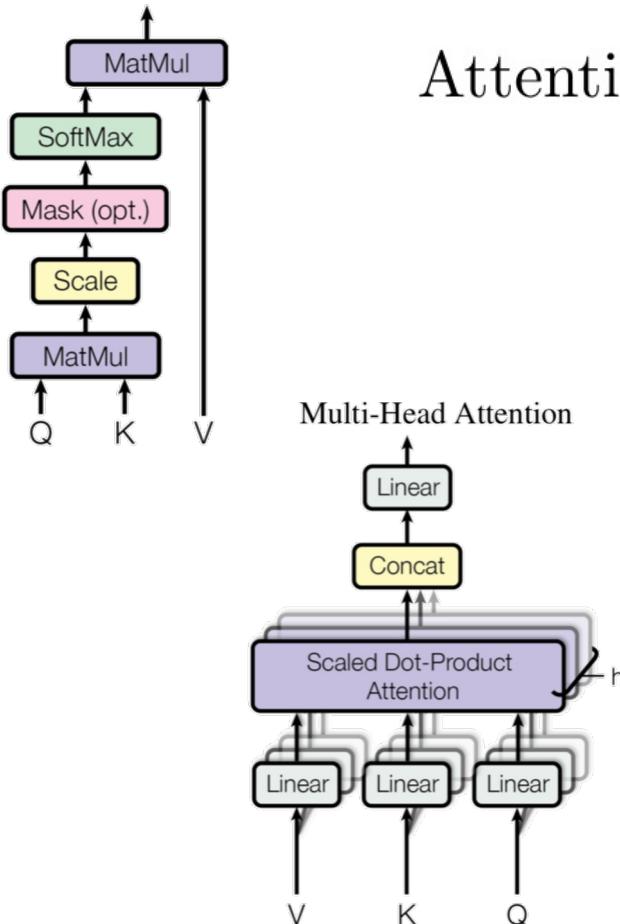


- RNNs 不在对序列模型建模.
- 相反, CNNs 用作特征提取.
- 语言模型有更深的结构.
- 模块化和可扩展的设计.



注意力模型

Scaled Dot-Product Attention



$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

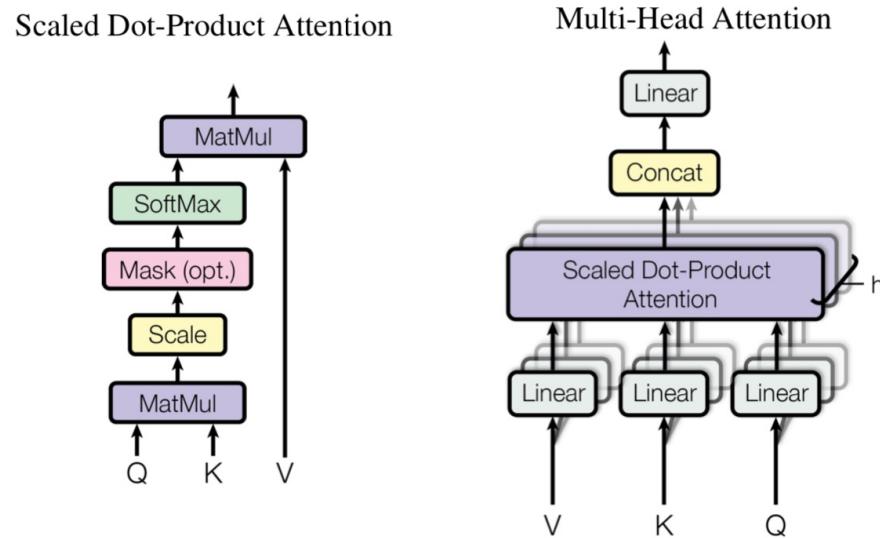
$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \dots, \text{head}_h)W^O$$

where $\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$

可以允许模型在不同的表示子空间里学习到相关的信息。

注意力模型-自注意力

变换，Transformer



- 例如输入一个句子，那么里面的每个词都要和该句子中的所有词进行attention计算。
- 目的是学习句子内部的词依赖关系，捕获句子的内部结构。



阅读资料

- 语言模型和RNN上一些经典的文章

- Graves, Alex. "Generating sequences with recurrent neural networks." (2013)
- Le, Quoc, and Tomas Mikolov. "Distributed representations of sentences and documents." (2014)
- Weston, Jason, Sumit Chopra, and Antoine Bordes. "Memory networks." (2014)
- Amodei, Dario, et al. "Deep speech 2: End-to-end speech recognition in English and Mandarin." (2016)
- Devlin, Jacob, et al. "Bert: Pre-training of deep bidirectional transformers for language understanding." (2018)
- So, David R., Chen Liang, and Quoc V. Le. "The evolved transformer." (2019)



这节课，我们学习了

- 自然语言处理概要
- 词向量
 - Word2Vec
- 循环神经网络(RNN)
 - 原始RNN
 - LSTM, GRU
 - 双向 RNN
- RNN语言模型
 - 编码解码模型 (Seq2Seq)
 - 注意力模型: 变换

