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

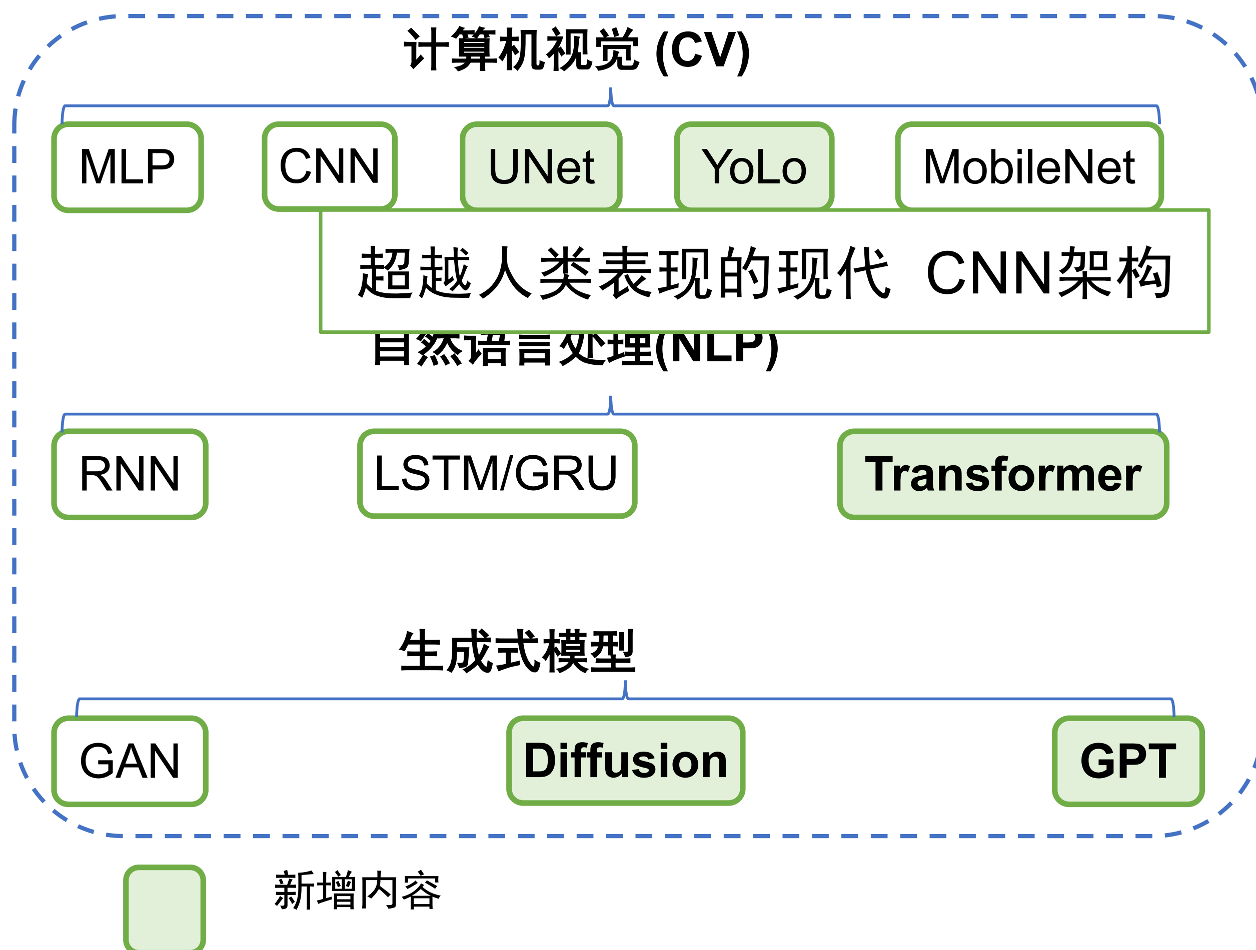
Bing Li

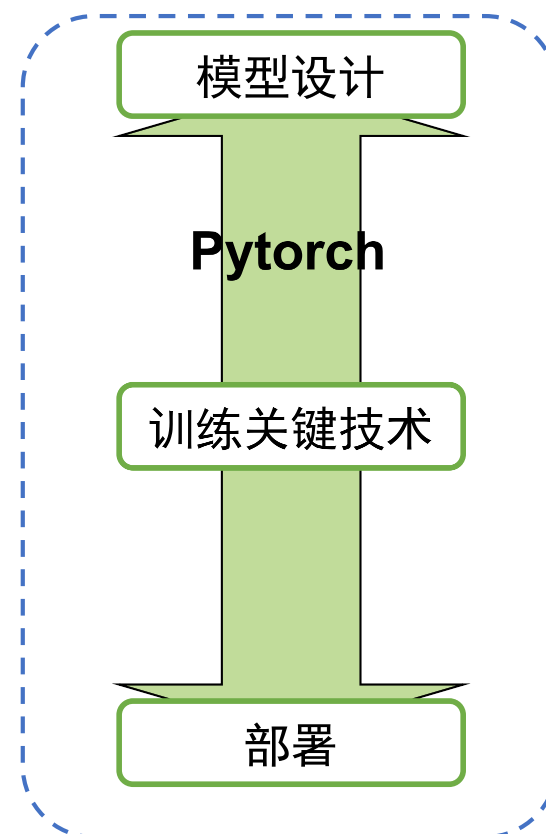
Associate Professor

Academy of Multidisciplinary Studies

Capital Normal University

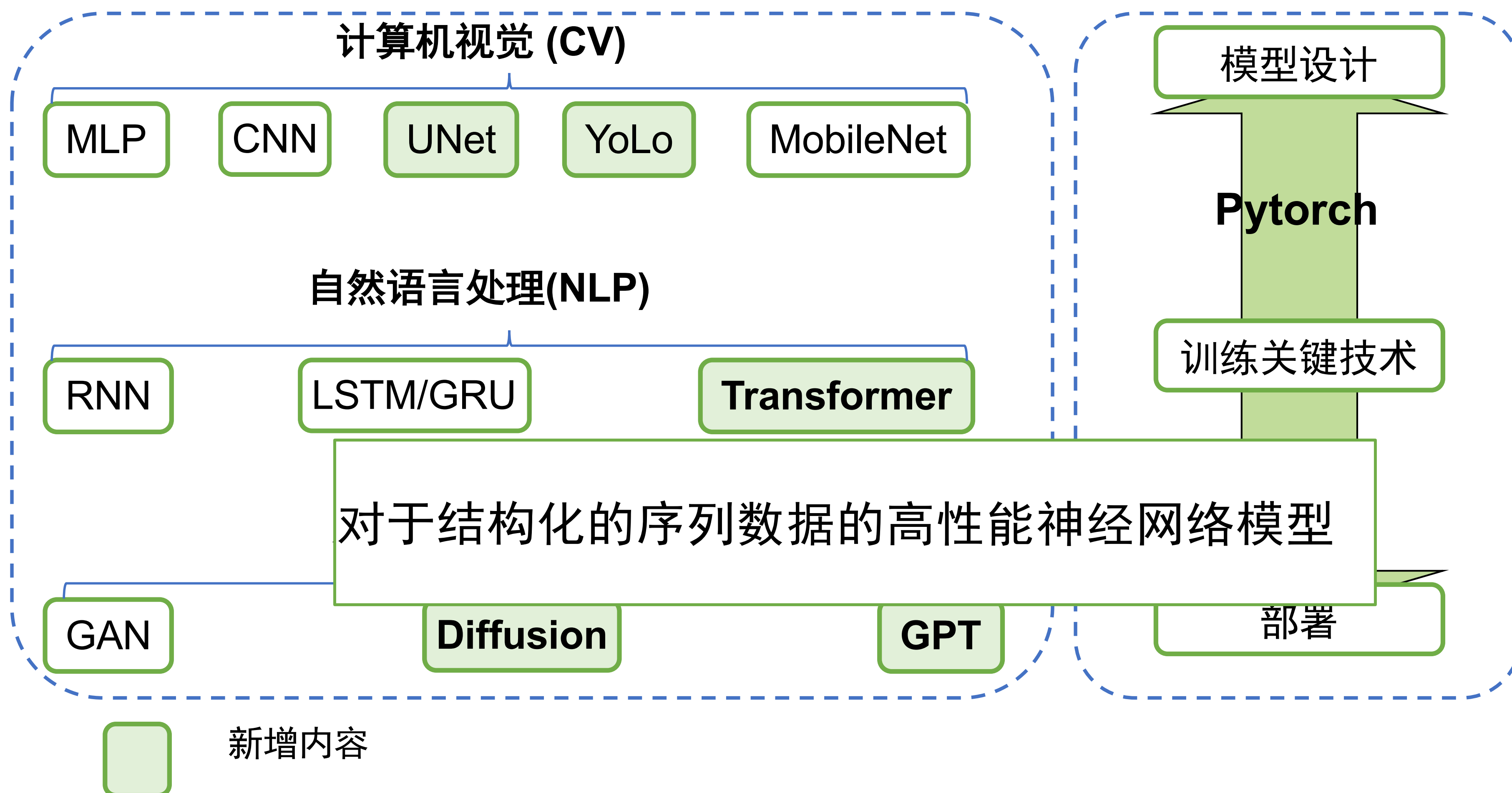
本门课的内容





新增内容

本门课的内容



序列数据

机器翻译

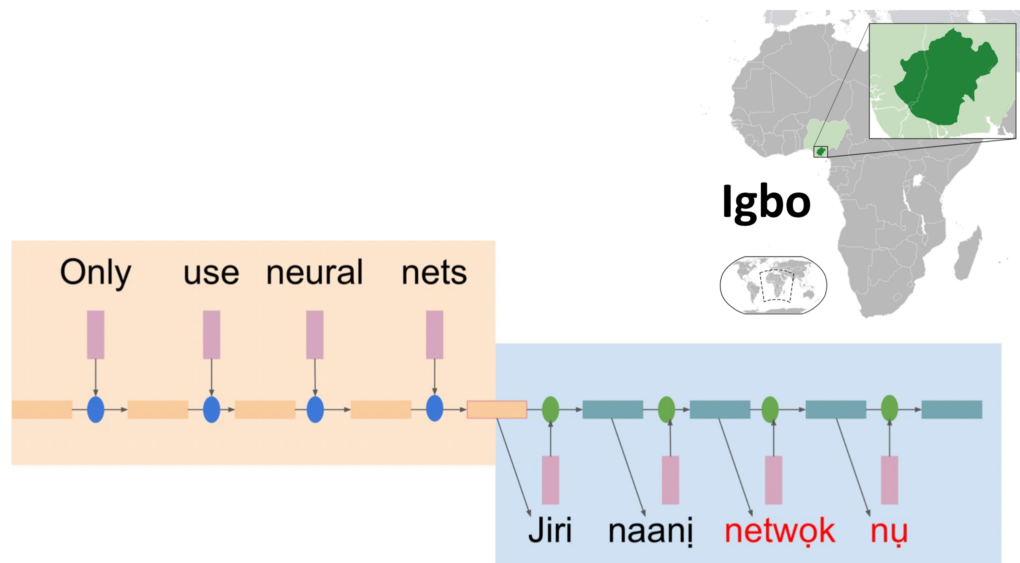
序列数据

- 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.
- CNN/DNN完成一次性的特征提取



https://blog.acolyer.org/2016/04/20/imagenet-classification-with-deep-convolutional-neural-networks/
https://nlp.stanford.edu/~johnhew/public/14-seq2seq.pdf

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

$$(w_0, w_1 \ldots w_{t-1}) \rightarrow 一个词$$

¥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... 分享

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戏体验。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 分享

物流快(88) 正品(36)

全部 有图/视频(360) 追加(38) 好评(1万) 中/差评

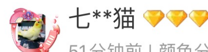
按时间排序 | 默认排序

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 分享

推荐 978 送给 TA 帮我选

发货 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 >

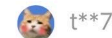
选择 请选择 颜色分类 套餐 版本类型 >

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)

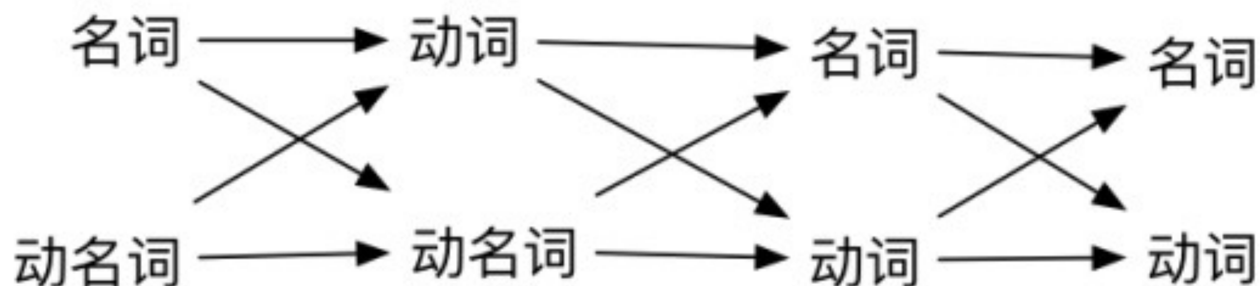


t**7

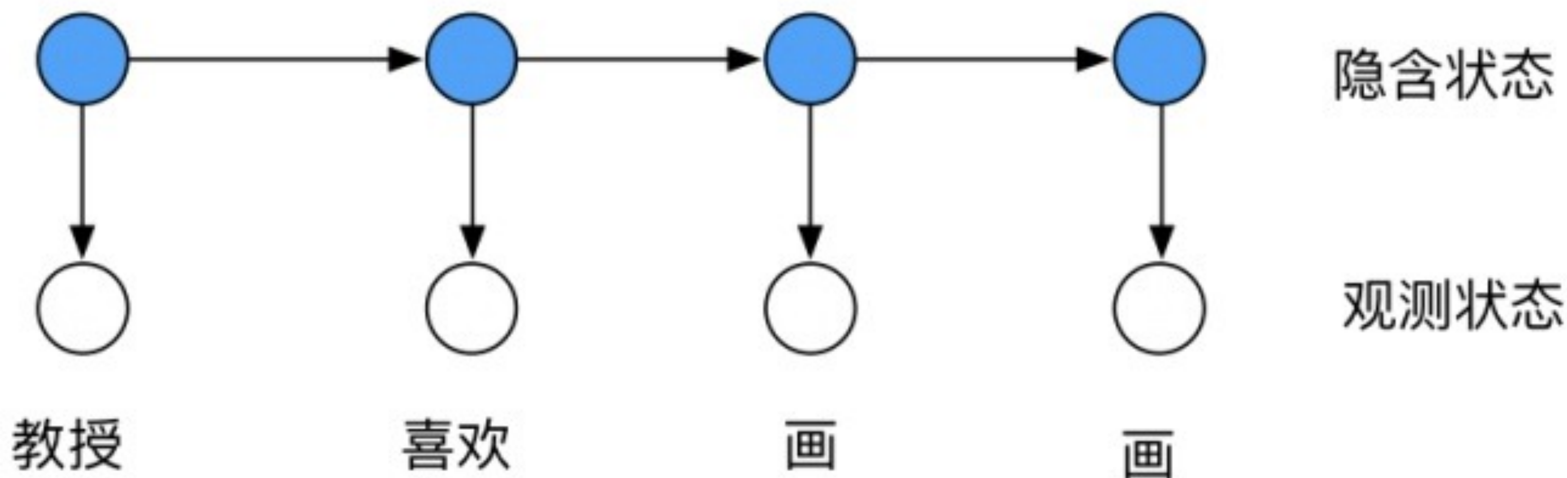
三码合一开机红，没有使用过的痕迹，新机没错了，游戏也是全新无划痕的，没翻车好评😎

词性标注

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



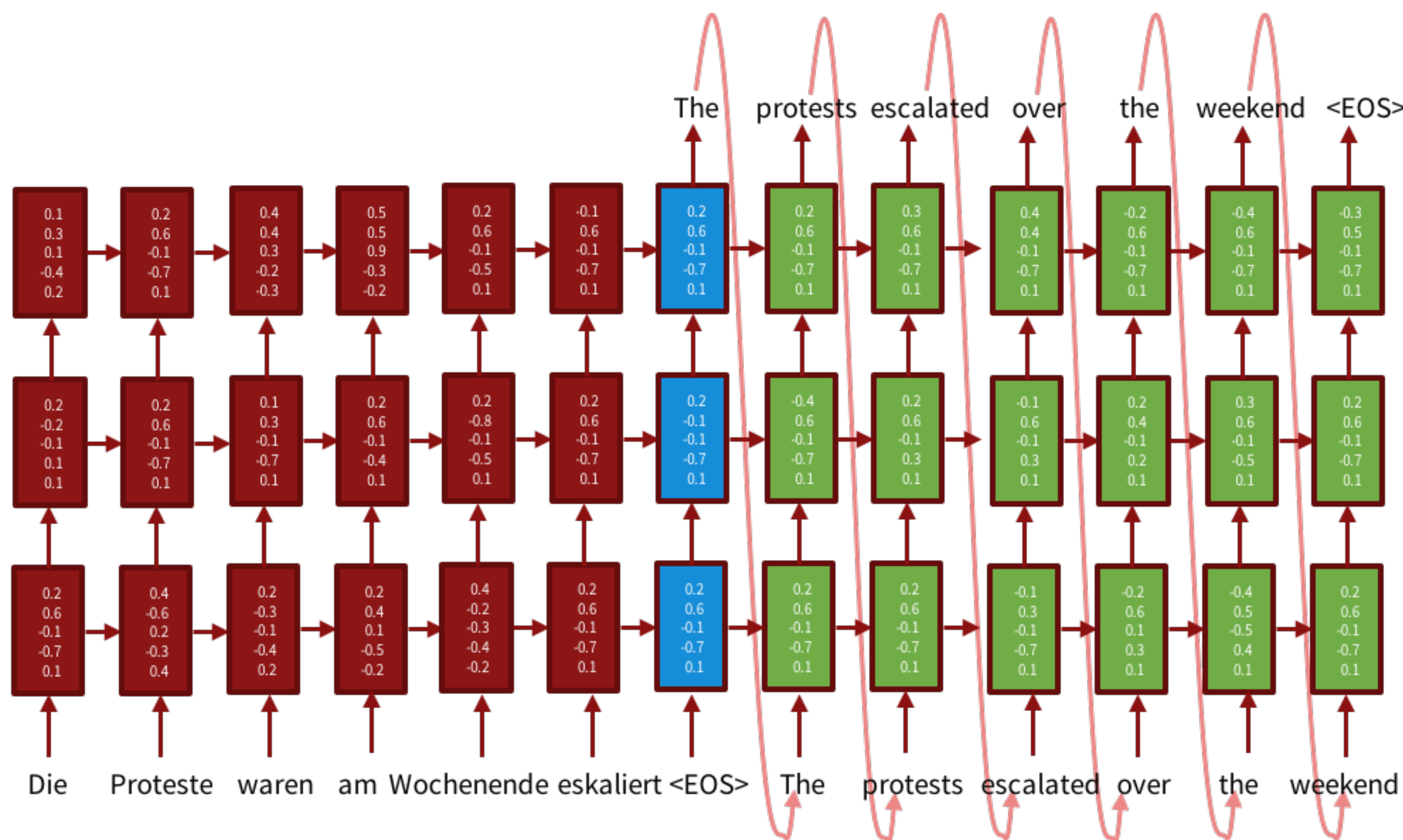
一共16中可能



Source: https://www.freecodecamp.org/news/an-introduction-to-part-of-speech-tagging-and-the-hidden-markov-model-953d45338f24/

神经网络机器翻译

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



Source: https://f417b5b8-a-62cb3a1a-s-sites.googlegroups.com/site/acl16nmt/home/nmt.png

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）

Trait	Score
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

	Trait #1	Trait #2	Trait #3	Trait #4	Trait #5
Jay	-0.4	0.8	0.5	-0.2	0.3

Person #1	-0.3	0.2	0.3	-0.4	0.9
Person #2	-0.5	-0.4	-0.2	0.7	-0.1

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 0 1 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.

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

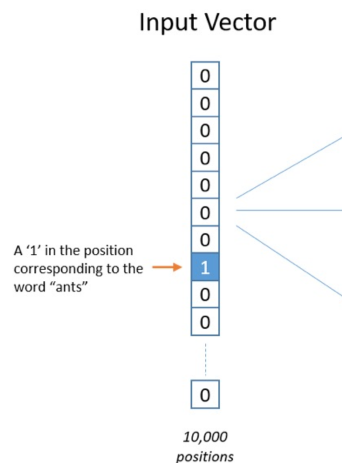
*核心思想：*一个单词的含义可以用周围的其他单词（上下文）来表示。

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

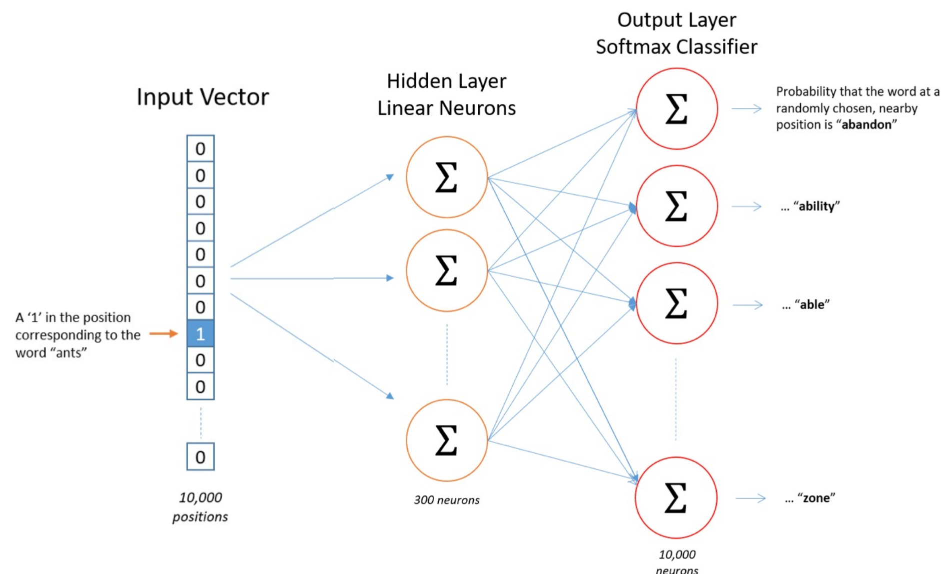
Word2Vec模型

- 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，用交叉熵来衡量神经网络的输出。



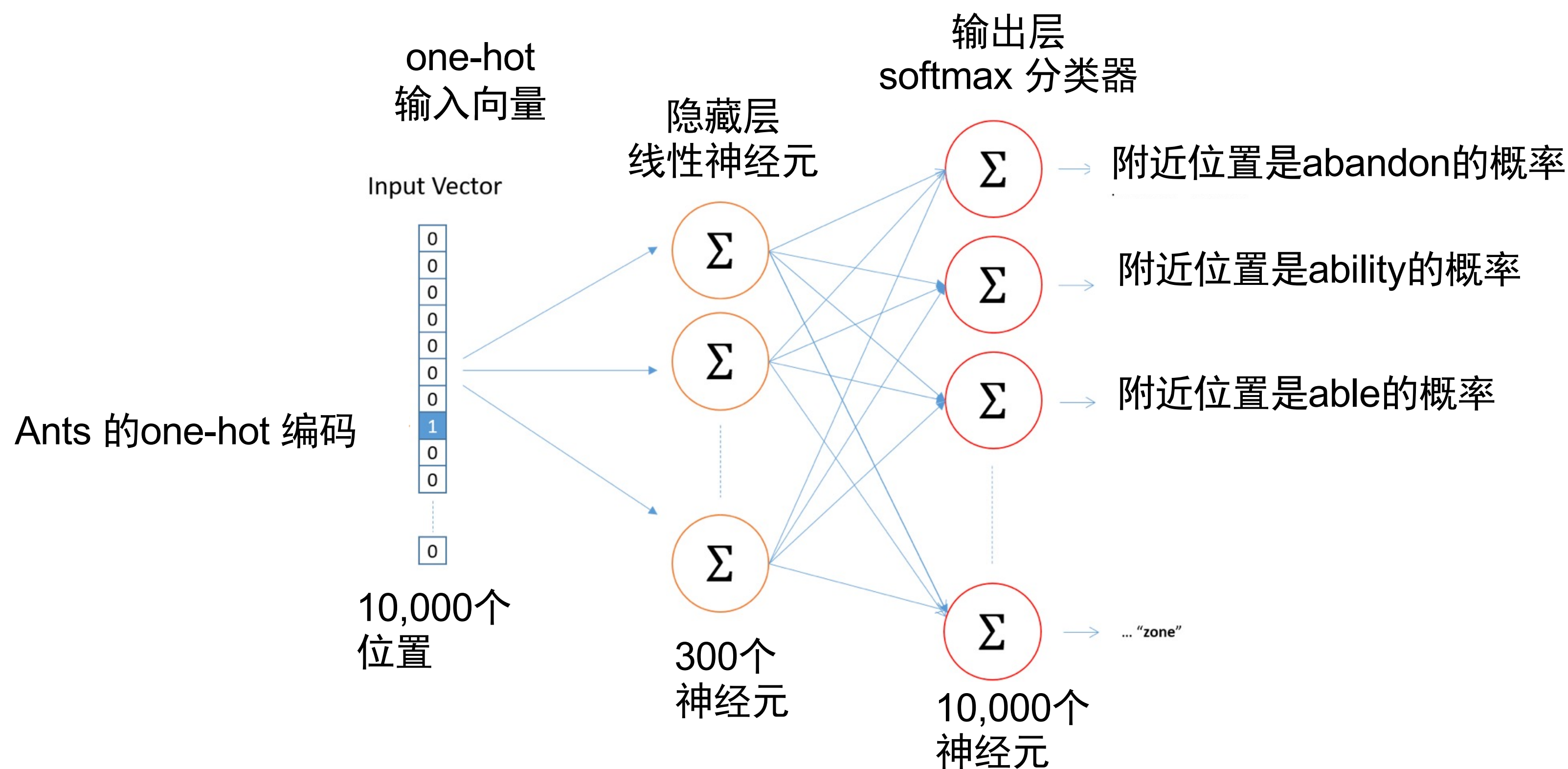
模型隐层权重矩阵，构成词嵌入表，也称为查找表Look up table。

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]。

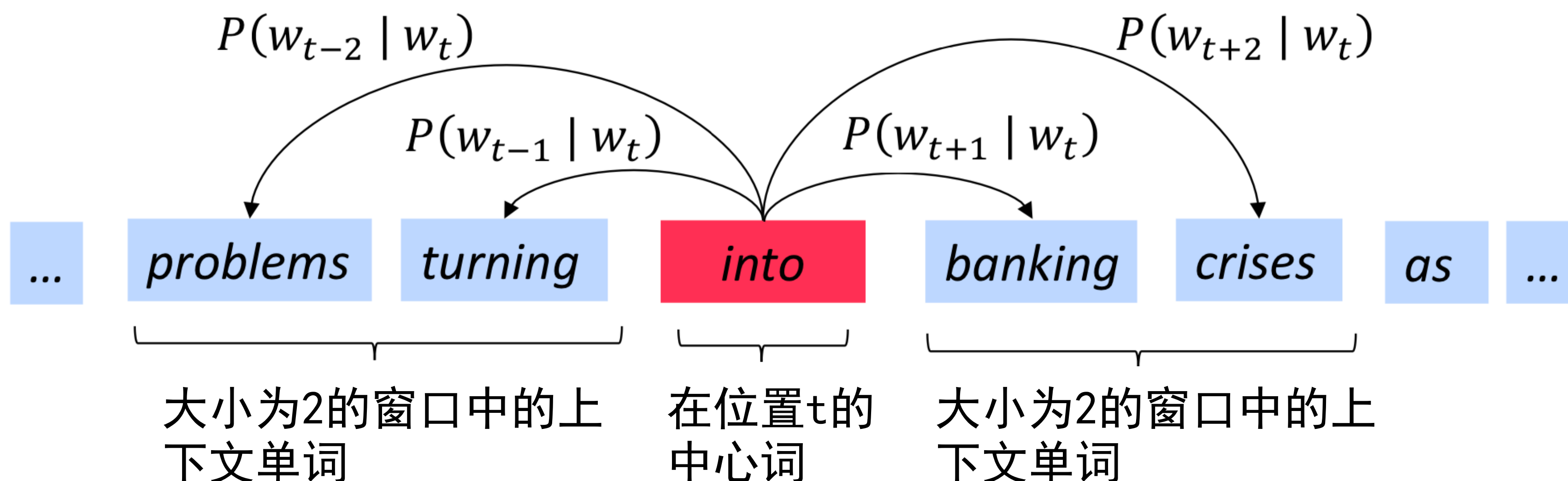
Source Text | Training Samples

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

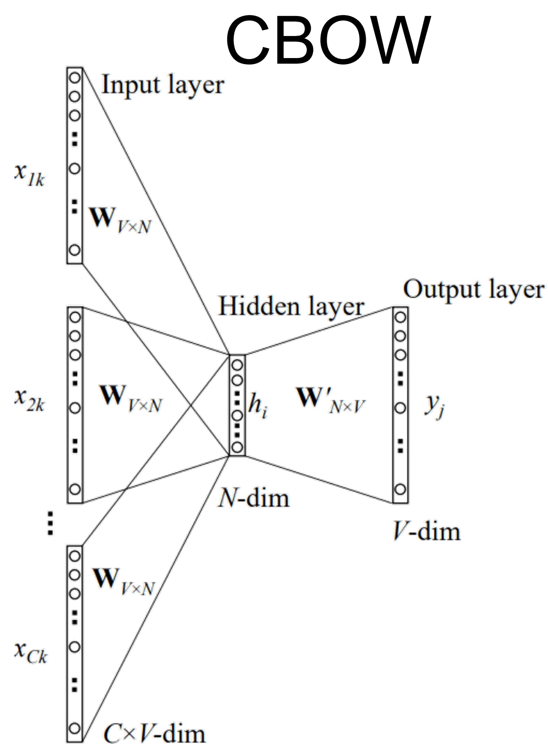


$P(w_{t+1}|w_t)$: 在已知单词w_t的条件下w_{t+1}出现的概率，后验概率。

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

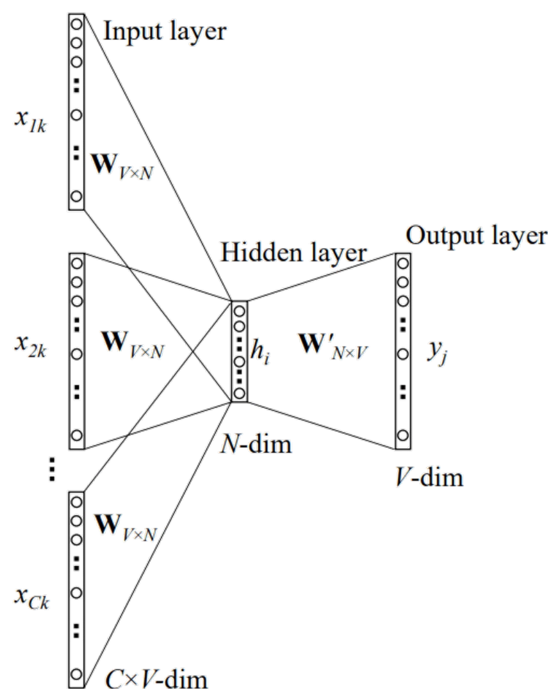
- 连续词袋模型 Continuous bag of words (CBOW): 从上下文单词预测中心词.
  - P(fox|quick,brown,jumps,over)



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_{VxN}，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_{NxV},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(fox|quick,brown,jumps,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：

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

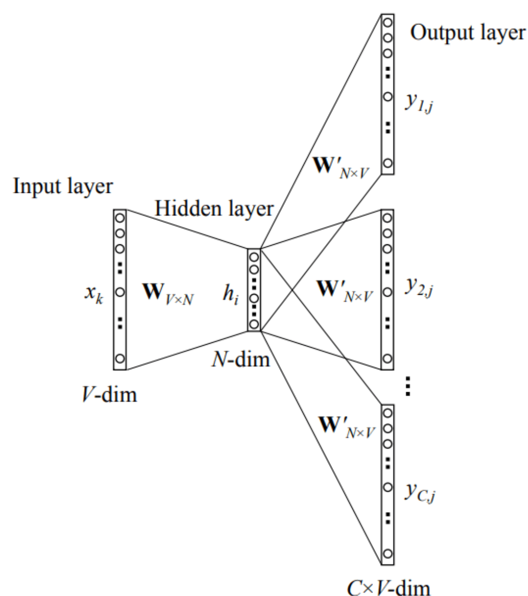
从隐层到输出层和之前的类似，和另一个矩阵W' 相乘，这里假设中心词下标为j，有$t_j = v^T u_j$，其中u_j为W'的第j列(N × 1)，结果得到一个数u_k。然后放入Softmax进行**概率归一化**：

$$p(w_j|w_1,\ldots,w_C) = y_j = \frac{\exp(\bar{\mathbf{v}}^{\mathrm{T}}\mathbf{u}_{\mathrm{j}})}{\sum_{k=1}^{V}\exp(\bar{\mathbf{v}}^{\mathrm{T}}\mathbf{u}_{\mathrm{k}})}$$

Word2Vec

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

SG

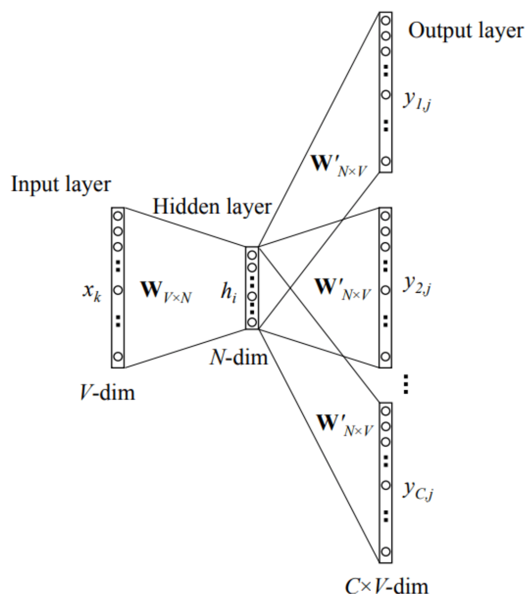


1. 选输入词，选输出词，我们将会得到两组 (input word, output word) 形式的训练数据，即 (‘apple’, ‘an’)，(‘apple’, ‘one’)。
3. 假如我们先拿一组数据 ('dog', 'barked') 来训练神经网络，那么模型通过学习这个训练样本，会告诉我们词汇表中每个单词是 “barked”的概率大小。

Word2Vec

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

SG



1. 输入是one-hot后的向量x,维度为V×1。然后是矩阵W，维度为V×N，则输入到隐层的操作其实是矩阵的相乘，$\mathbf{v}_k = \mathbf{W}^\mathbf{T}\mathbf{x} = \mathbf{W}^\mathbf{T}{}_\mathbf{k}$，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×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×V，设$t_{c\ j} = \mathbf{v}^T{}_k\mathbf{u}_j$，其中$\mathbf{u}_j$为W′的第j列(N×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

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

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

```python
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 tr

# 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

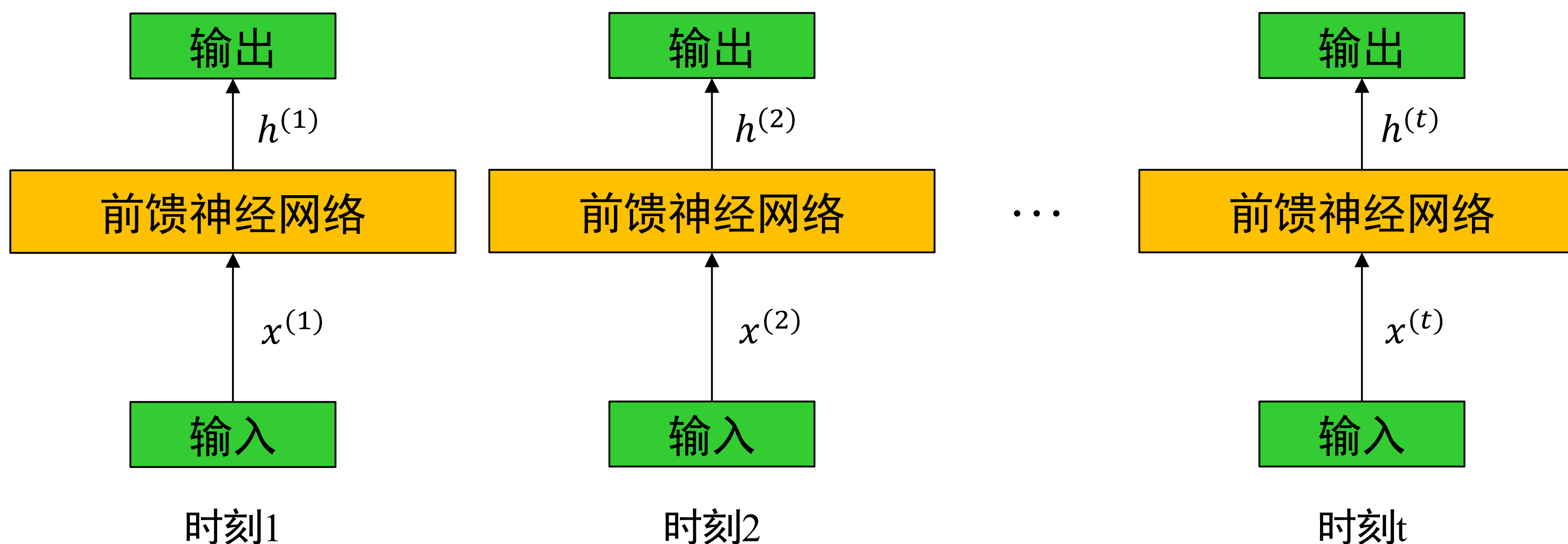
- 包括但不限于：
  - 计算相似度
    - 寻找相似词
    - 信息检索
  - 作为 SVM/LSTM 等模型的输入
    - 中文分词
    - 命名体识别
  - 句子表示
    - 情感分析
  - 文档表示
    - 文档主题判别

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

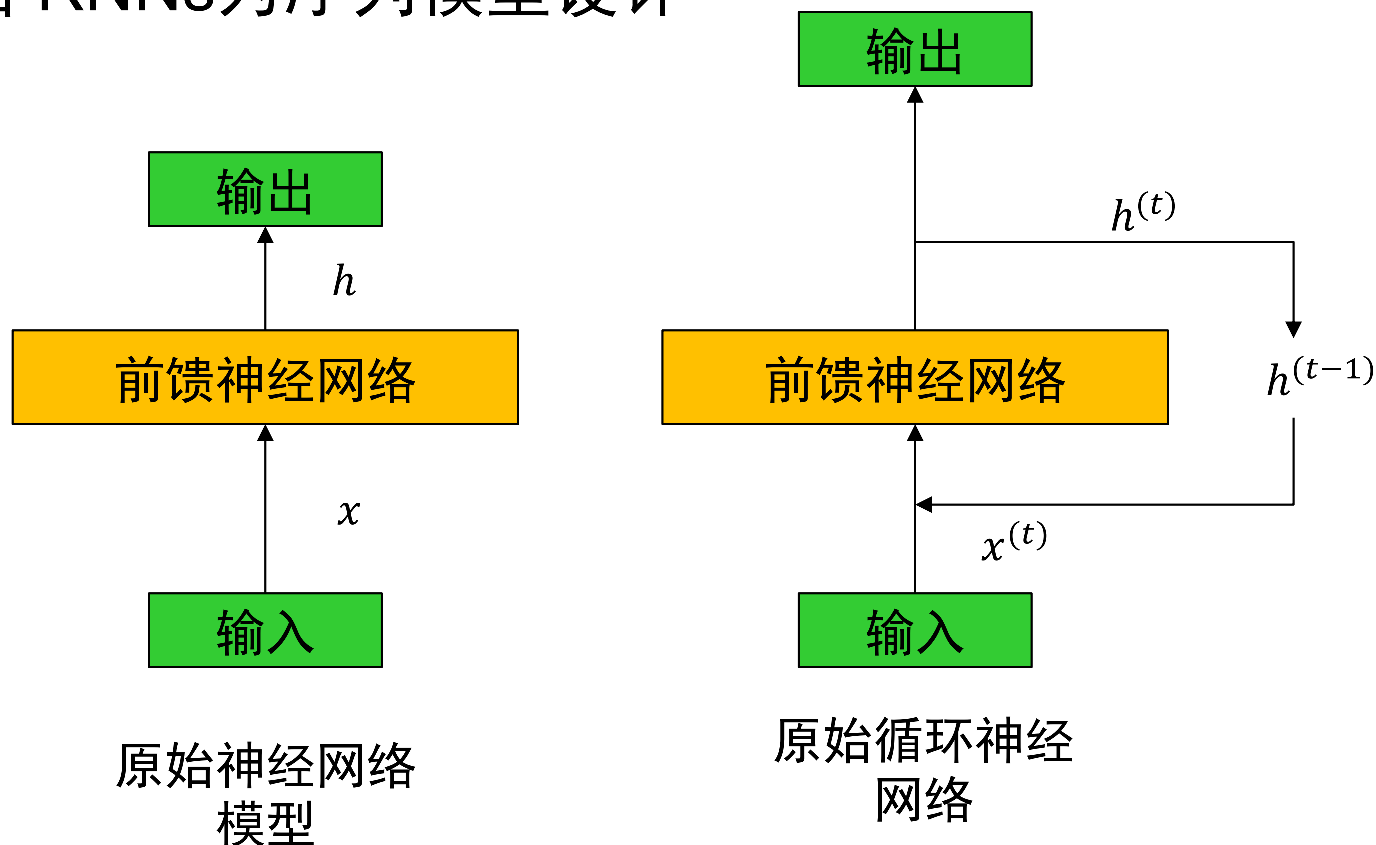
- 原始 NN不能很好的处理序列数据.
- 他们必须在时间维度对每一帧的内容进行处理。



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）

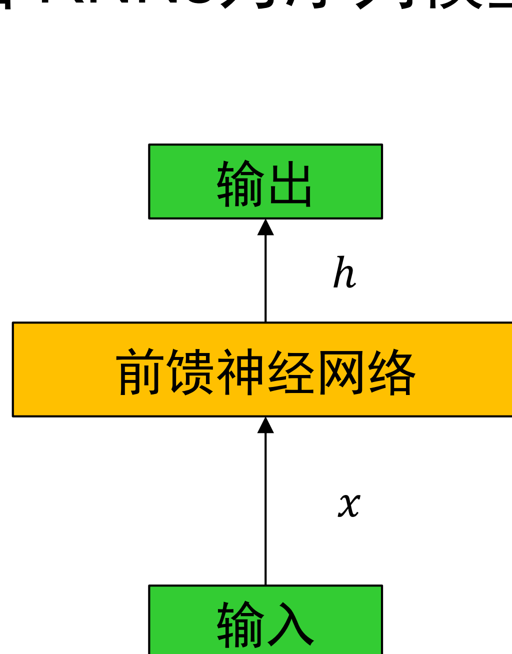
- 原始 RNNs为序列模型设计



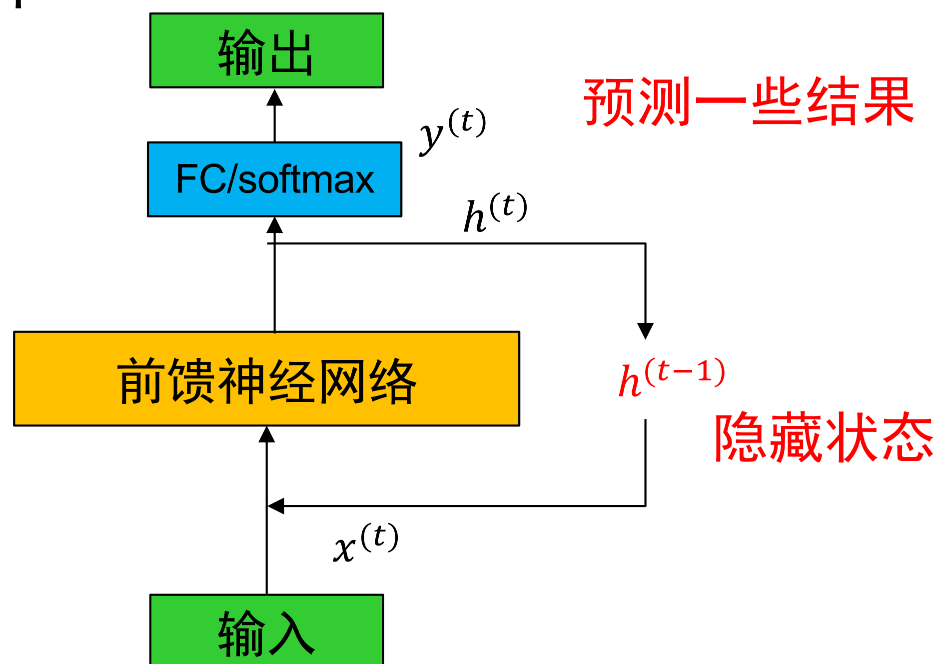
输出 会成为下一个阶段的输入

原始的循环神经网络（RNN）

- 原始 RNNs为序列模型设计



原始NN



预测一些结果

隐藏状态

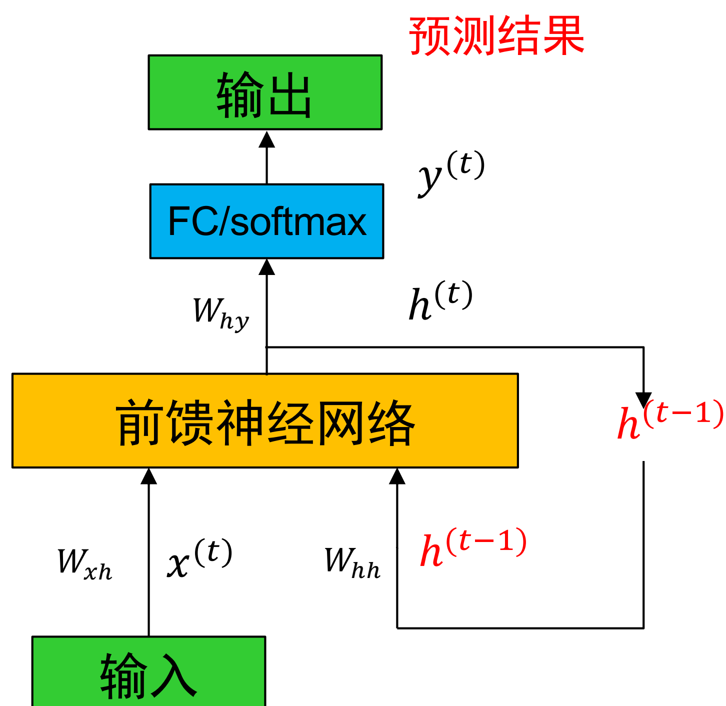
原始RNN

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 $h^{(1)}, h^{(2)}, \ldots, h^{(t)}$.

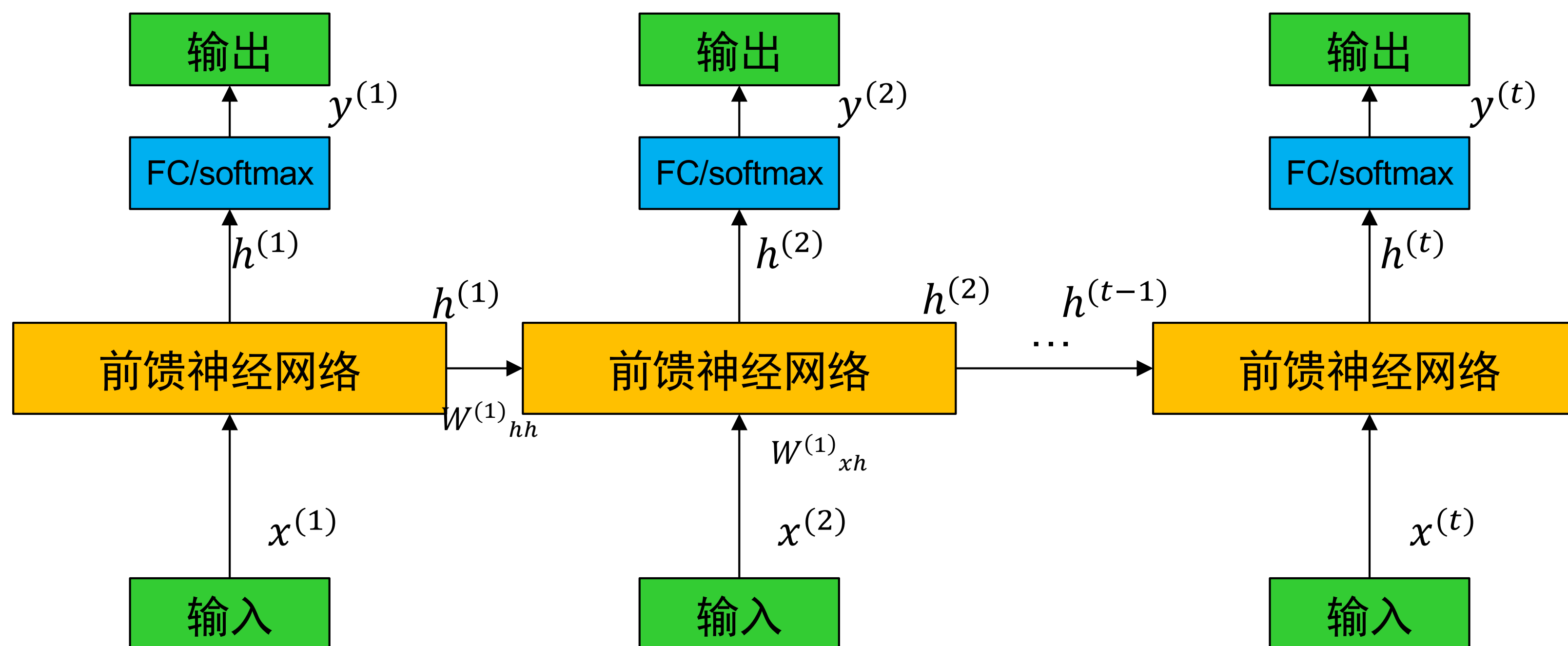
$$h^{(t)} = \tanh(W_{xh}x^{(t)} + W_{hh}h^{(t-1)} + b_h)$$

$$y^{(t)} = \sigma(W_{hy}h^{(t)} + b_y)$$
$$t \in [1, 2, 3, \ldots, L]$$

Vanilla RNN

展开 RNN的计算过程

- 在时间维度上展开RNN.



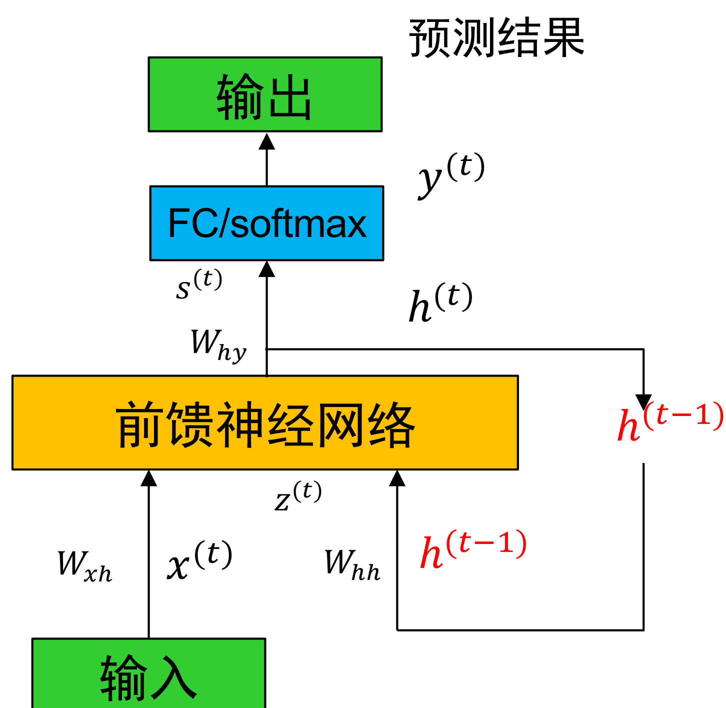
$$h^t = f_W(h_{t-1}, x)$$

原始的循环神经网络（RNN）

RNN 后向传播

- 基于时间的反向传播BPTT

前向传播



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

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

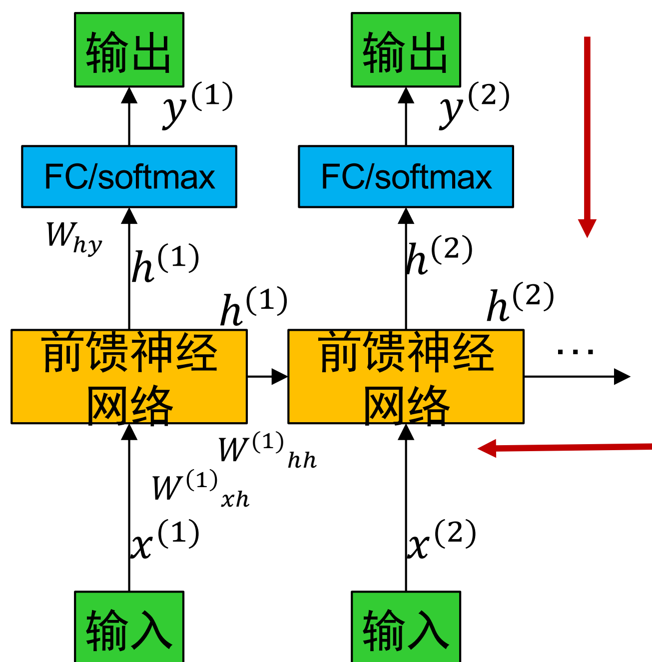
$$z^{(t)} = W_{xh}x^{(t)} + W_{hh}h^{(t-1)} + b_h$$
$$s^{(t)} = W_{hy}h^{(t)} + b_y$$
$$h^{(t)} = \mathrm{f}(z^{(t)}) = \tanh(z^{(t)})$$
$$y^{(t)} = \sigma(s^{(t)})$$
$$t \in [1, 2, 3, \dots, L]$$

Vanilla RNN: BPTT

原始RNN的反向传播

- 基于时间的反向传播BPTT

$$z^{(t)} = W_{xh}x^{(t)} + W_{hh}h^{(t-1)} + b_h$$
$$s^{(t)} = W_{hy}h^{(t)} + b_y$$
$$h^{(t)} = f(z^{(t)}) = tanh(z^{(t)})$$
$$y^{(t)} = \sigma(s^{(t)})$$
$$t \in [1, 2, 3, \dots, L]$$



后向传播

$$E^{(t)} = \frac{1}{2}\sum_{i=1}^{M}(y_i^{(t)} - d_i^{(t)}), E = \sum_{i=1}^{t} E^t$$

$$\frac{\partial E}{\partial W_{hy}} = \frac{\partial E}{\partial s}\frac{\partial s}{\partial W_{hy}} = \left(y^{(t)} - d^{(t)}\right)\odot\sigma'\left(s^{(t)}\right)\ \left(h^{(t)}\right)^T$$

$$\frac{\partial E}{\partial W_{xh}} = \frac{\partial E}{\partial z^t}\frac{\partial z^t}{\partial W_{xh}} = \delta_h^t\left(x^{(t)}\right)^T$$

$$\delta_h^t = \frac{\partial E^t}{\partial\ \ {}^{(t)}}\left(W_{hy}^T\delta_y^t + W_{hh}^T\delta_h^{t+1}\right)\odot f'(z^{(t)})$$

$$\delta_y^t = \frac{\partial E^t}{\partial z^{(t)}} = \left(y^{(t)} - d^{(t)}\right)\odot\sigma'\left(s^{(t)}\right)^T$$

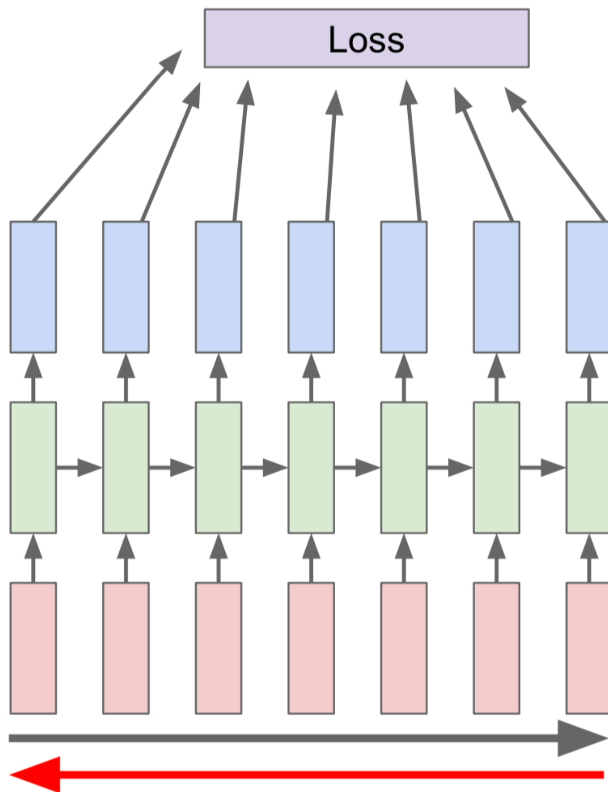
$$\frac{\partial E}{\partial W_{hh}} = \delta_h^t\left(h^{(t)}\right)^T$$

$$\frac{\partial E}{\partial W_{hy}} = \delta_y^t\left(h^{(t)}\right)^T$$

Vanilla RNN: BPTT

原始RNN的反向传播

- 基于时间的反向传播BPTT



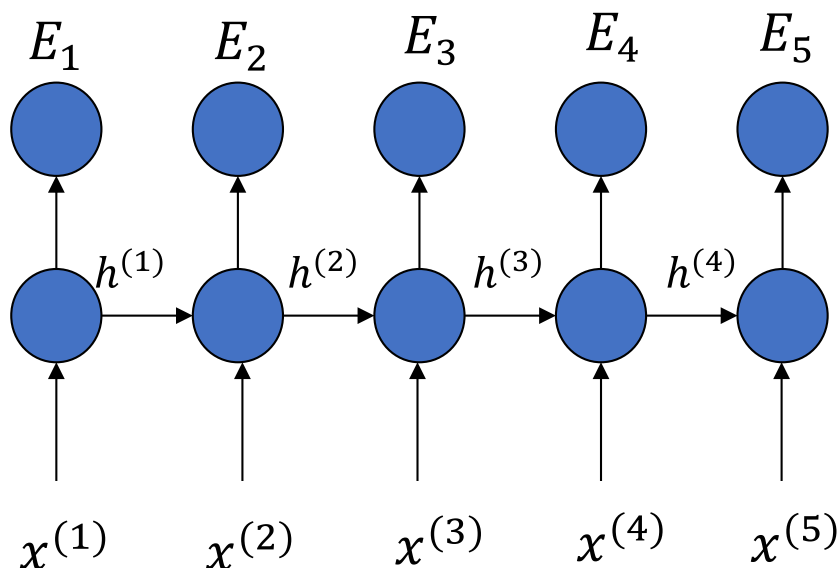
序列的前向来计算loss，整个序列的后向来计算梯度

```
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?

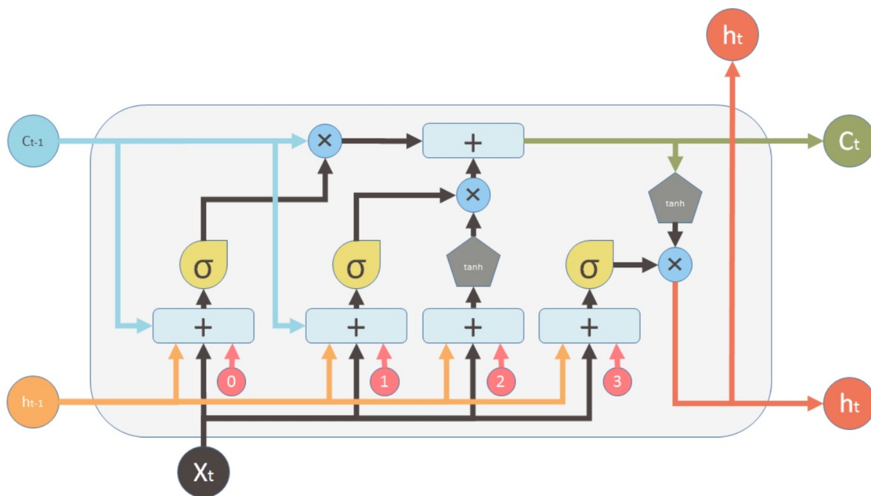
- 用 ReLU
- 用 LSTM

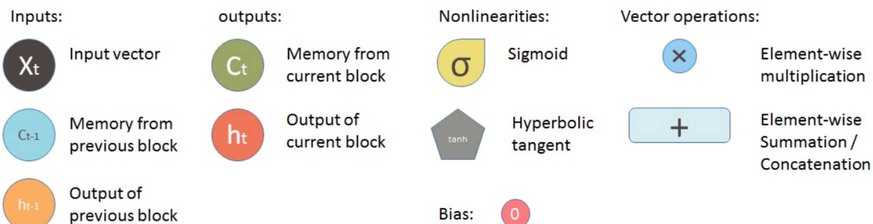
由于RNN的复杂特性，用 RMSprop 优化器来训练RNN模型.

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。



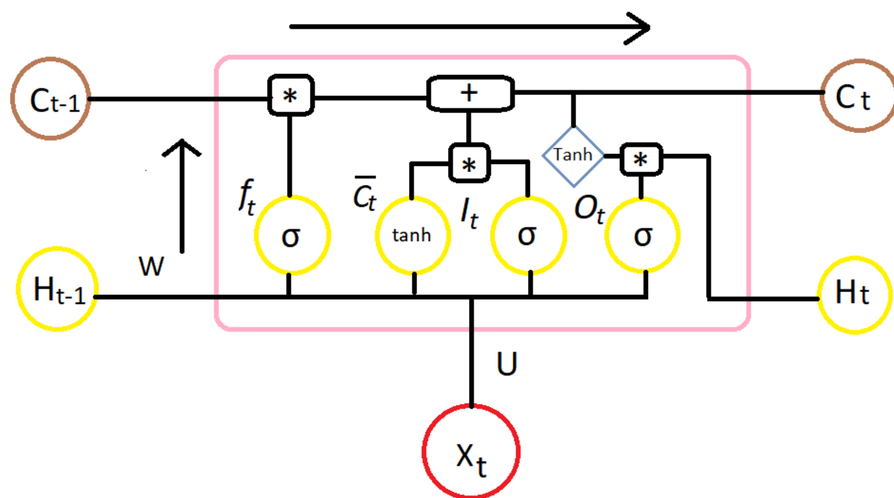


但是…

- 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.

Hochreiter, Sepp, and Jürgen Schmidhuber. "Long short-term memory." (1997)

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

$$\overline{C_t} = tanh(W_c[h_{t-1}, x_t] + b_c)$$

$$C_t = f_t \circ C_{t-1} + i_t \circ \overline{C_t}$$

输出门/预测

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

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

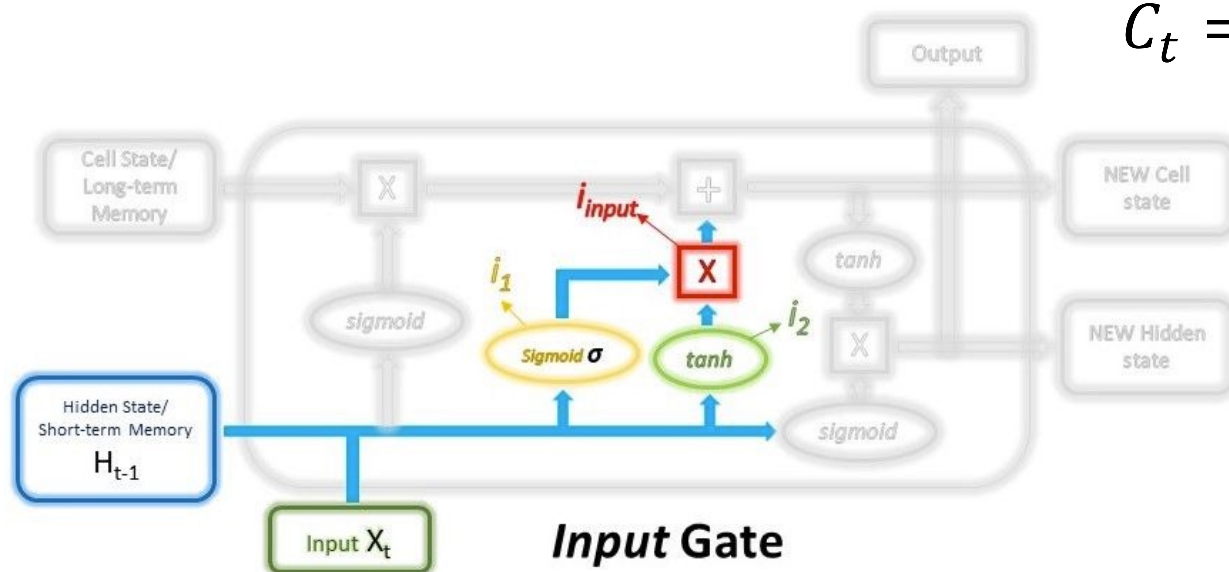
Hochreiter, Sepp, and Jürgen Schmidhuber. "Long short-term memory." (1997)

输入门

- 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

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

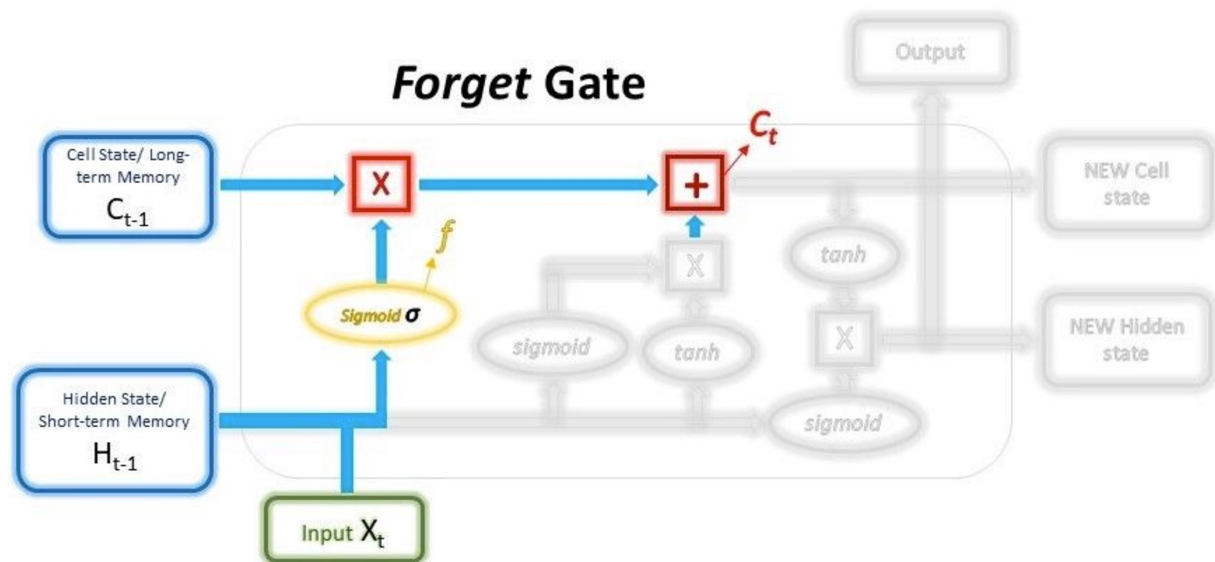
$$\overline{C}_t = tanh(W_c[h_{t-1}, x_t] + b_c)$$



$$i_{input} = i_t \circ \overline{C}_t$$

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。



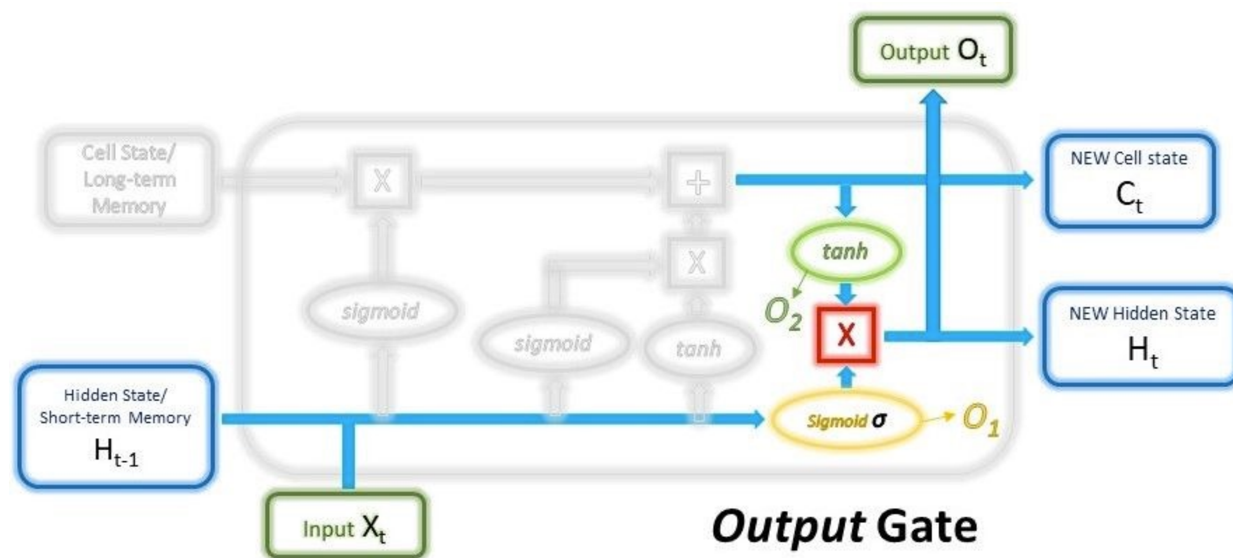
$$f_t = \sigma(W_f[h_{t-1}, x_t] + b_f)$$

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 。

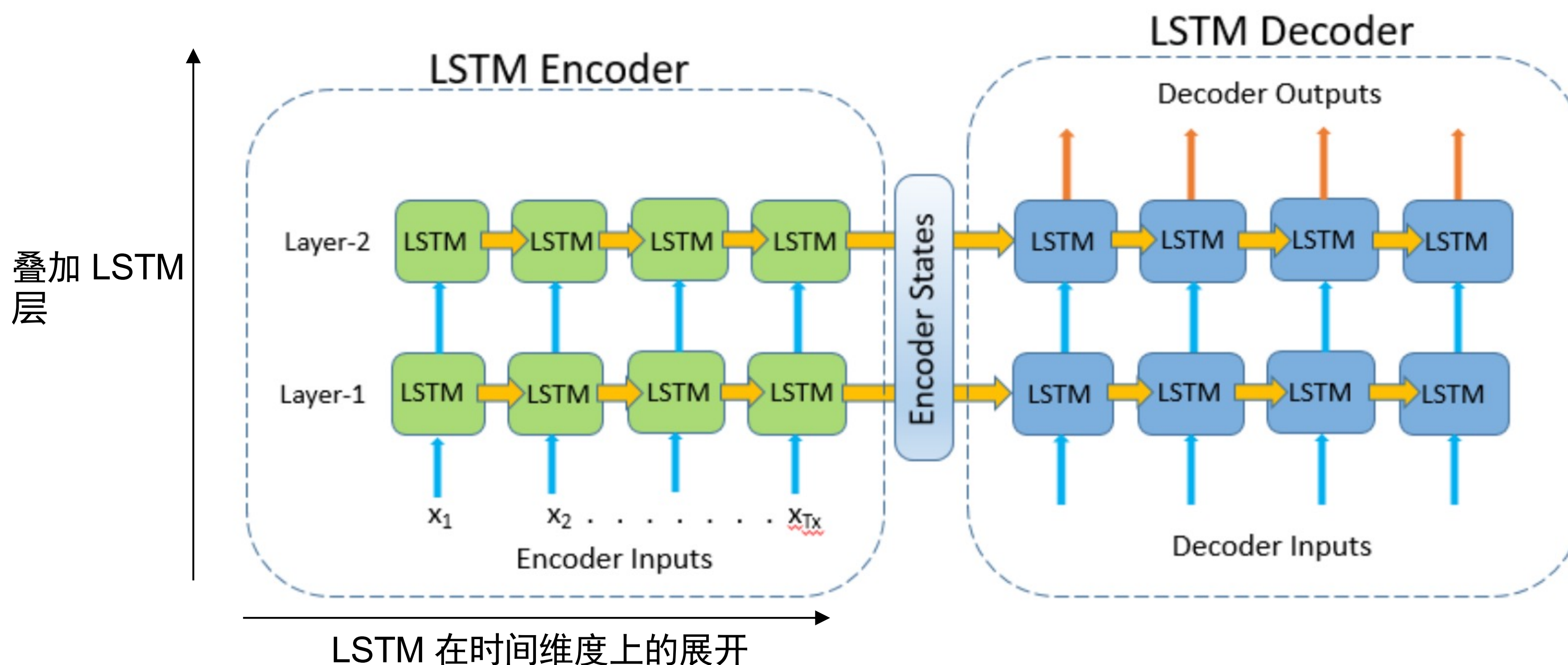


$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o)$$

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

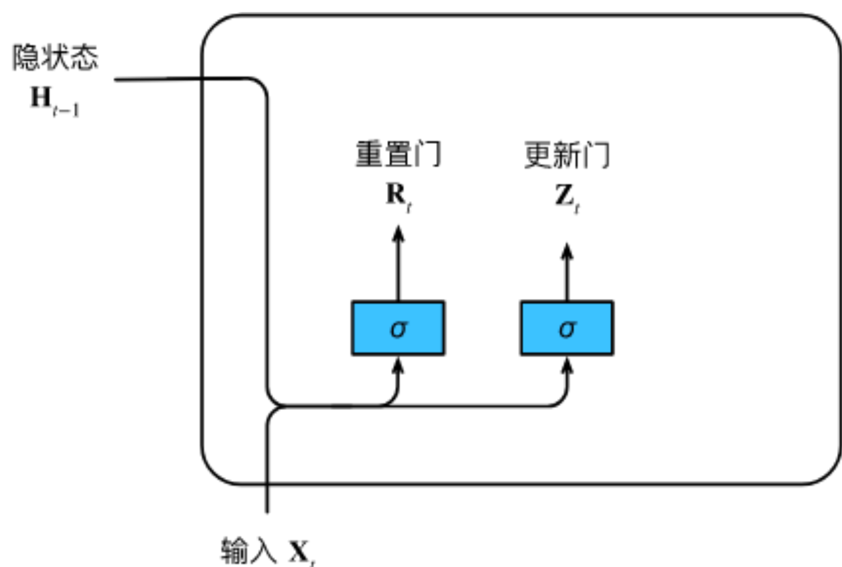
长短期记忆网络(LSTM)

- 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.



带激活函数的全连接层

复制

连结

重置门

$$\mathbf{R}_t = \sigma(\mathbf{X}_t \mathbf{W}_{xr} + \mathbf{H}_{t-1} \mathbf{W}_{hr} + \mathbf{b}_r),$$

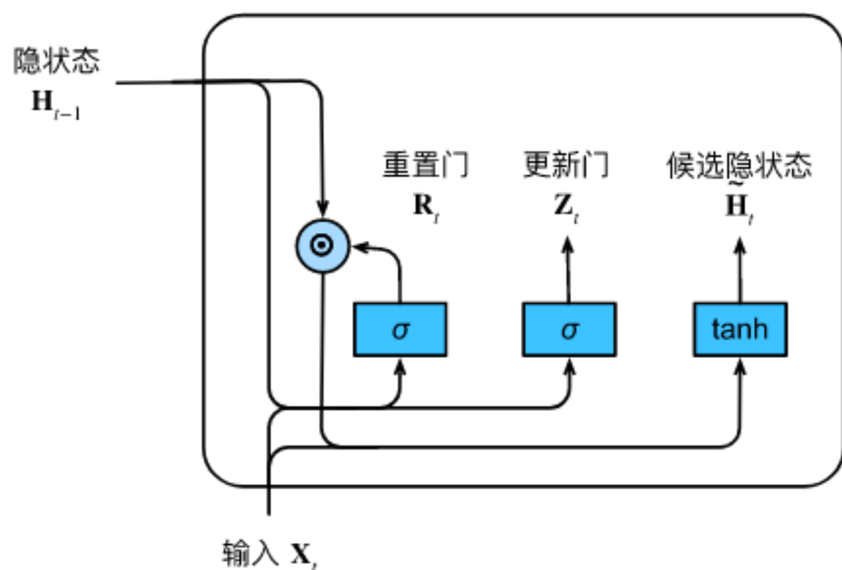
更新门

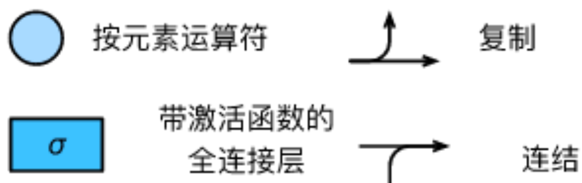
$$\mathbf{Z}_t = \sigma(\mathbf{X}_t \mathbf{W}_{xz} + \mathbf{H}_{t-1} \mathbf{W}_{hz} + \mathbf{b}_z),$$

Chung, Junyoung, et al. "Empirical evaluation of gated recurrent neural networks on sequence modeling." (2014)

Gated recurrent unit (GRU)

- LSTM单元的简单版本，能学到更有意义的单词表示.





重置门

$$\mathbf{R}_t = \sigma(\mathbf{X}_t\mathbf{W}_{xr} + \mathbf{H}_{t-1}\mathbf{W}_{hr} + \mathbf{b}_r),$$

更新门

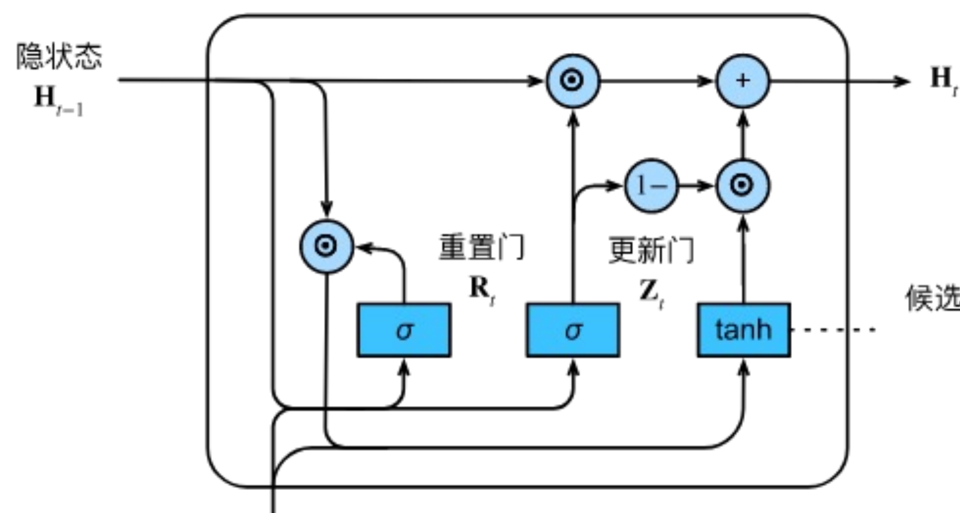
$$\mathbf{Z}_t = \sigma(\mathbf{X}_t\mathbf{W}_{xz} + \mathbf{H}_{t-1}\mathbf{W}_{hz} + \mathbf{b}_z),$$

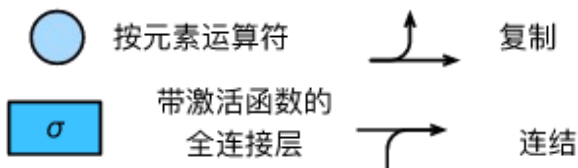
记忆内容

$$\tilde{\mathbf{H}}_t = \tanh(\mathbf{X}_t\mathbf{W}_{xh} + (\mathbf{R}_t \odot \mathbf{H}_{t-1})\mathbf{W}_{hh} + \mathbf{b}_h),$$

Gated recurrent unit (GRU)

- LSTM单元的简单版本，能学到更有意义的单词表示.





重置门

$$\mathbf{R}_t = \sigma(\mathbf{X}_t \mathbf{W}_{xr} + \mathbf{H}_{t-1} \mathbf{W}_{hr} + \mathbf{b}_r),$$

更新门

$$\mathbf{Z}_t = \sigma(\mathbf{X}_t \mathbf{W}_{xz} + \mathbf{H}_{t-1} \mathbf{W}_{hz} + \mathbf{b}_z),$$

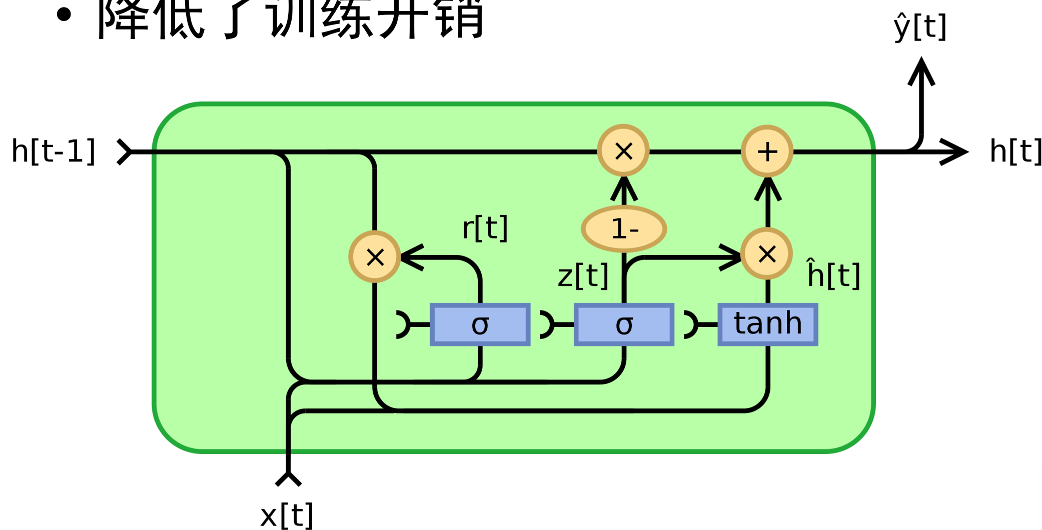
记忆内容

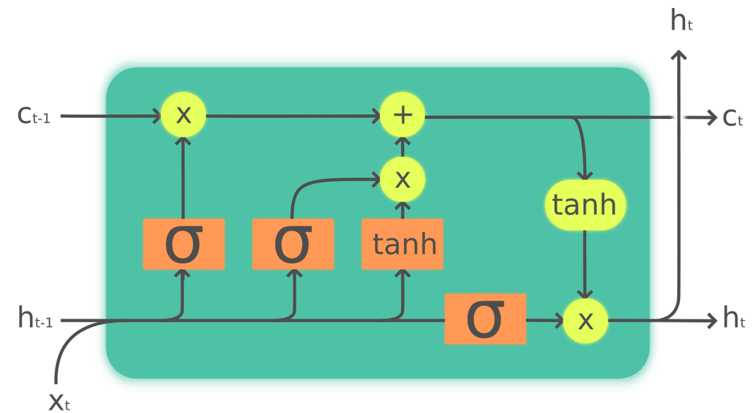
$$\tilde{\mathbf{H}}_t = \tanh(\mathbf{X}_t \mathbf{W}_{xh} + (\mathbf{R}_t \odot \mathbf{H}_{t-1}) \mathbf{W}_{hh} + \mathbf{b}_h),$$

$$\mathbf{H}_t = \mathbf{Z}_t \odot \mathbf{H}_{t-1} + (1 - \mathbf{Z}_t) \odot \tilde{\mathbf{H}}_t.$$

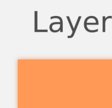
Gated recurrent unit (GRU)

- 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



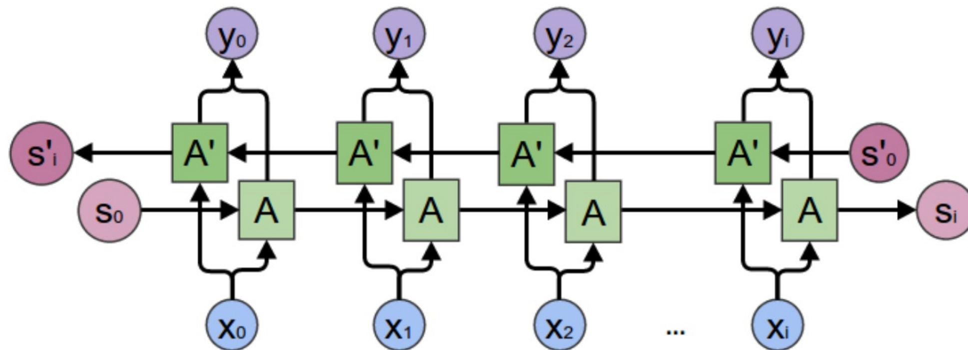


Legend: Layer Pointwize op Copy

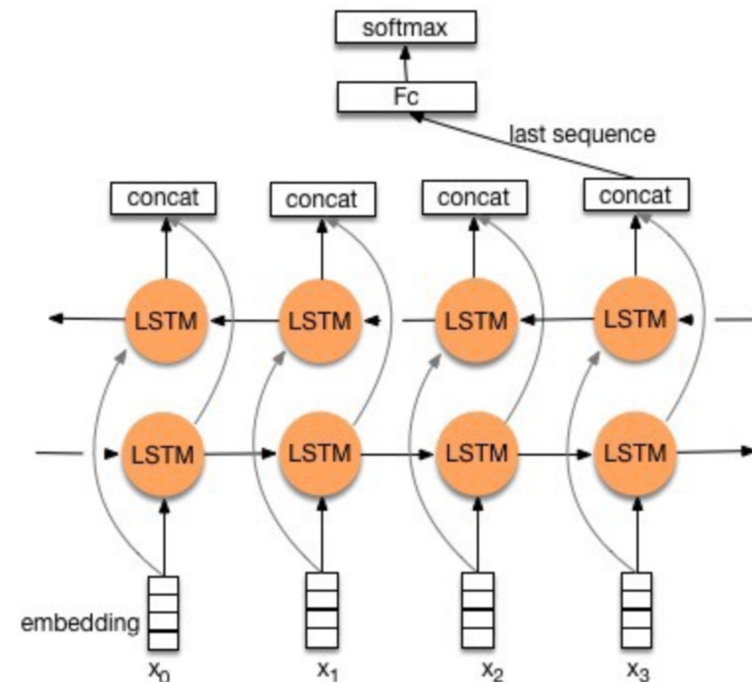


双向 RNNs

- 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

Schuster, Mike, and Kuldip K. Paliwal. "Bidirectional recurrent neural networks.” (1997)

Pytorch实现: RNN

- torch.nn.RNN： 多层RNN，tanh或者Relu做激活函数

$$h_t = \tanh(x_t W_{ih}^T + b_{ih} + h_{t-1} W_{hh}^T + b_{hh})$$

- **参数(input, h_0)**
  - input：输入序列，形状有（L，H_{in}）,(L,N,H_{in}),
  - h_0, 初始隐层状态，形状是（D*num_layer, H_{out}）

```
rnn = nn.RNN(10, 20, 2)
input = torch.randn(5, 3, 10)
h0 = torch.randn(2, 3, 20)
output, hn = rnn(input, h0)
```

Pytorch 实现: LSTM

- torch.nn.LSTM： 多层LSTM

$$
\begin{aligned}
i_t &= \sigma(W_{ii}x_t + b_{ii} + W_{hi}h_{t-1} + b_{hi}) \\
f_t &= \sigma(W_{if}x_t + b_{if} + W_{hf}h_{t-1} + b_{hf}) \\
g_t &= \tanh(W_{ig}x_t + b_{ig} + W_{hg}h_{t-1} + b_{hg}) \\
o_t &= \sigma(W_{io}x_t + b_{io} + W_{ho}h_{t-1} + b_{ho}) \\
c_t &= f_t \odot c_{t-1} + i_t \odot g_t \\
h_t &= o_t \odot \tanh(c_t)
\end{aligned}
$$

- **参数(input, h_0，c_0)**
  - input：输入序列，形状有（L，H_{in}）,(L,N,H_{in}),
  - h_0, 初始隐层状态，形状是（D*num_layer, H_{out}）
  - c_0, 初始记忆单元状态，形状是（D*num_layer, H_{cell}）

```
#Example of LSTM
rnn = nn.LSTM(10, 20, 2)
input = torch.randn(5, 3, 10)
h0 = torch.randn(2, 3, 20)
c0 = torch.randn(2, 3, 20)
output, (hn, cn) = rnn(input, (h0, c0))
```

Pytorch 实现: GRU

- torch.nn.GRU： 多层GRU

$$
\begin{aligned}
r_t &= \sigma(W_{ir}x_t + b_{ir} + W_{hr}h_{(t-1)} + b_{hr}) \\
z_t &= \sigma(W_{iz}x_t + b_{iz} + W_{hz}h_{(t-1)} + b_{hz}) \\
n_t &= \tanh(W_{in}x_t + b_{in} + r_t * (W_{hn}h_{(t-1)} + b_{hn})) \\
h_t &= (1 - z_t) * n_t + z_t * h_{(t-1)}
\end{aligned}
$$

- **参数(input, h_0)**
  - input：输入序列，形状有（L，H_{in}）,(L,N,H_{in}),
  - h_0, 初始隐层状态，形状是（D*num_layer, H_{out}）
  - c_0, 初始记忆单元状态，形状是（D*num_layer, H_{cell}）

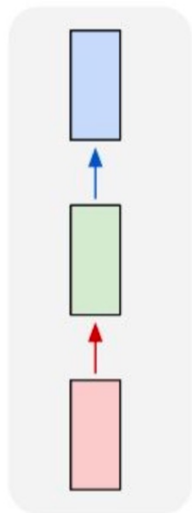
```
rnn = nn.GRU(10, 20, 2)
in_gru = torch.randn(5, 3, 10)
h0 = torch.randn(2, 3, 20)
out_gru, hn = rnn(in_gru, h0)
```

这节课

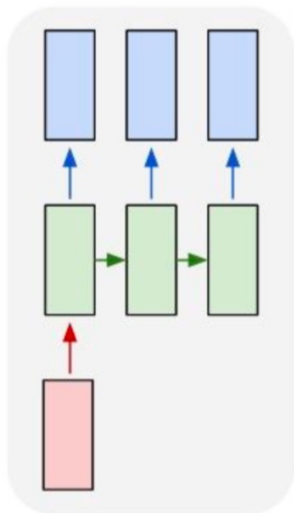
- 自然语言处理（NLP）
- 词向量
- 循环神经网络
- 循环神经网络语言模型

RNN 模型

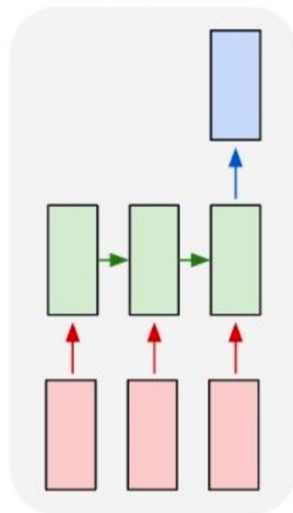
RNN 的任务类型



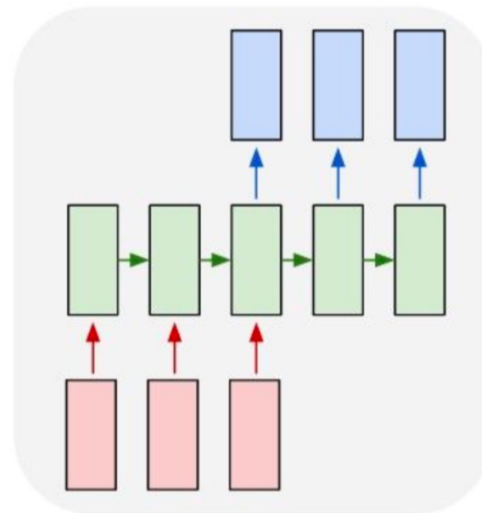
无序列模型

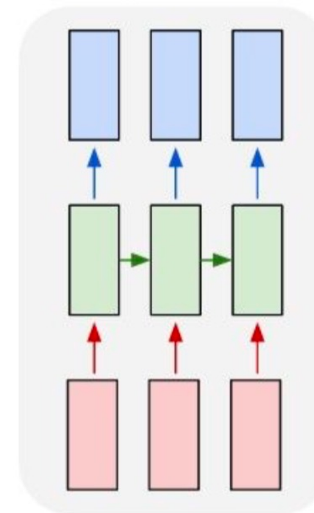


序列生成
图片加标题



情感分析

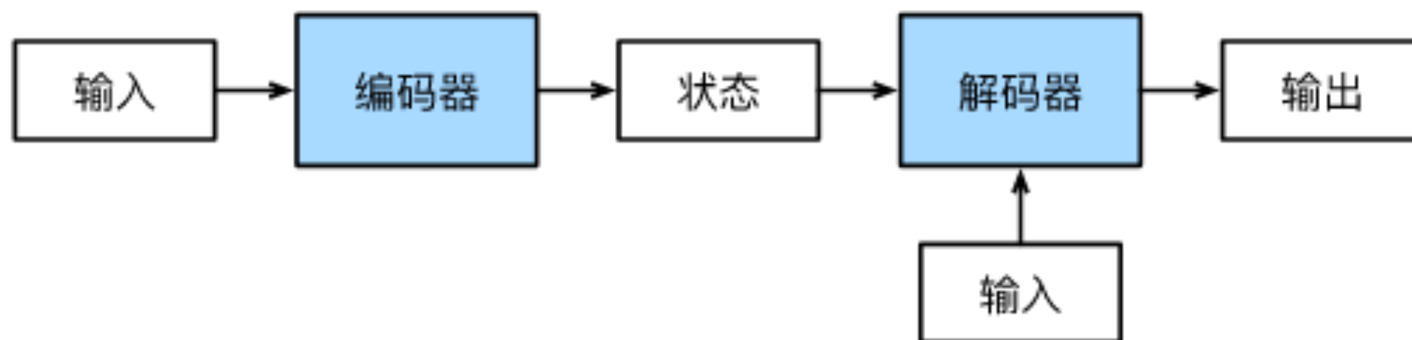




神经网络机器翻译

RNN 模型

- **编码解码模型(Seq2Seq)**
  - 机器翻译：输入（hello） -> 输出 （你好）。输入是1个英文单词，输出为2个汉字。
  - 对话机器中：我们提（输入）一个问题，机器会自动生成（输出）回答。这里的输入和输出显然是长度没有确定的序列（sequences）

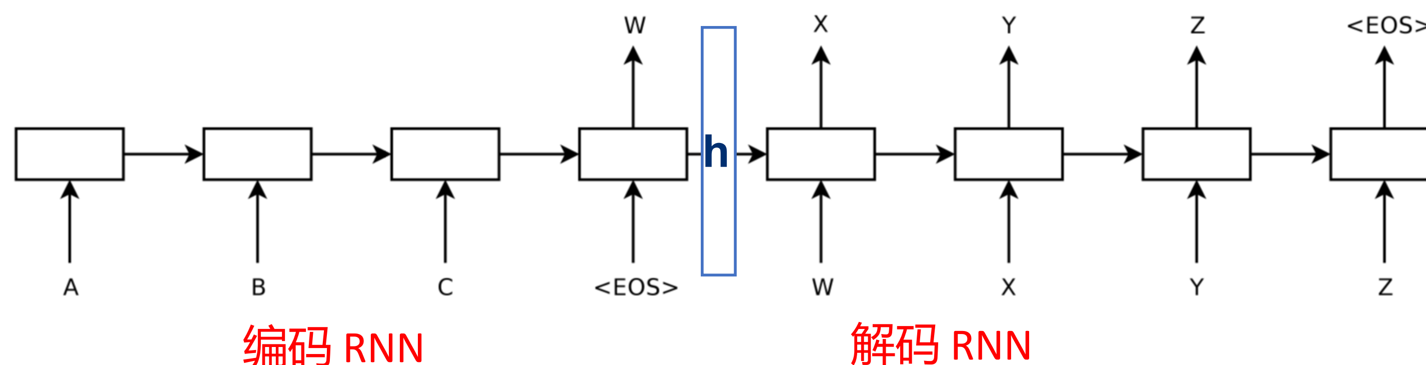


  - 编码器Encoder是将输入序列转化成一个固定长度的向量
  - 解码器Decoder是将输入的固定长度向量解码成输出序列
  - 其中编码解码的方式可以是RNN,CNN等

Sutskever, Ilya, Oriol Vinyals, and Quoc V. Le. "Sequence to sequence learning with neural networks." (2014)

RNN 模型

- **编码解码模型(Seq2Seq)**



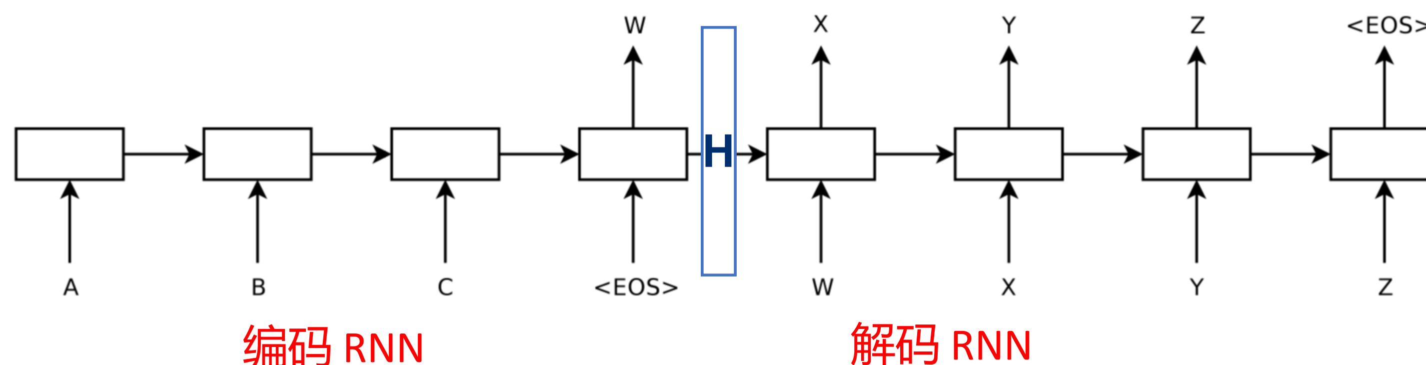
H是encoder输出的最终状态，向量H通常为RNN中的最后一个隐节点，或是多个隐节点的加权总和，作为decoder的初始状态；
W是encoder的最终输出，作为decoder的初始输入。

$$h_i = \mathrm{EncoderRNN}(x_i, h_{i-1})$$

x_i 是输入序列的第 i 个元素，h_{i-1} 是前一个时间步的隐藏状态。

RNN 模型

- 编码解码模型(Seq2Seq)



一个目标序列 $Y = [y_1, y_2, ..., y_m]$，其中 y_j 表示目标序列的第 j 个元素。
解码器在每个时间步生成一个输出 $\hat{y}_j$，表示模型生成的序列的第 j 个元素。

$$\text{Loss} = -\sum[\log(P(y_j|y_1, y_2, ..., y_{j-1}, h))]$$
$$P(y_j|y_1, y_2, ..., y_{j-1}, h) = \text{softmax}(o_j)$$

为了处理序列长度不同的情况，可以采用填充（padding）和掩码（masking）技术来处理填充位置上的输出，使其不参与误差计算。这样可以确保只计算有效部分的误差。

Sutskever, Ilya, Oriol Vinyals, and Quoc V. Le. "Sequence to sequence learning with neural networks." (2014)

Seq2Seq的应用

文字 – 文字	音频 – 文字	图片 – 文字
机器翻译 对话机器人 文章摘要 代码不全	语音识别	图像描述生成

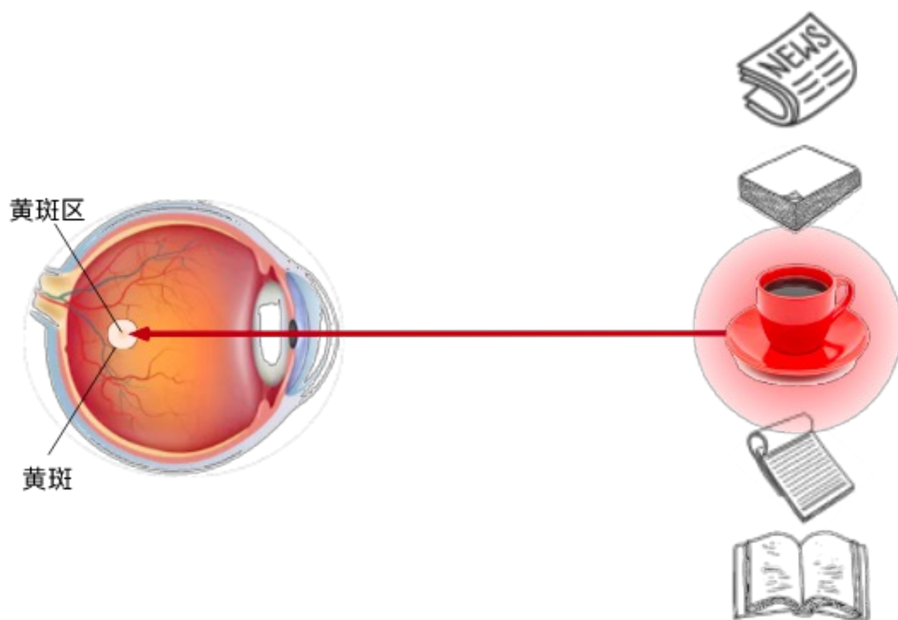
这节课

- 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
  - 注意力模型

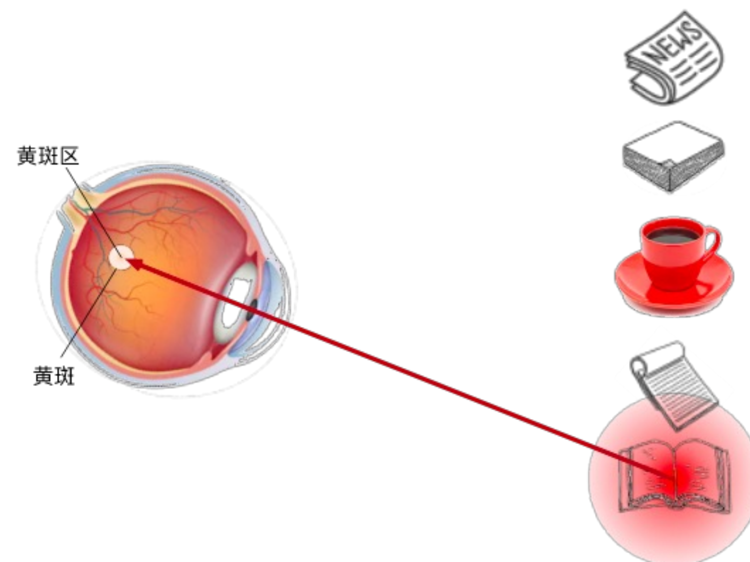
注意力模型

- 生物学的启发

非自主性(物体显眼)



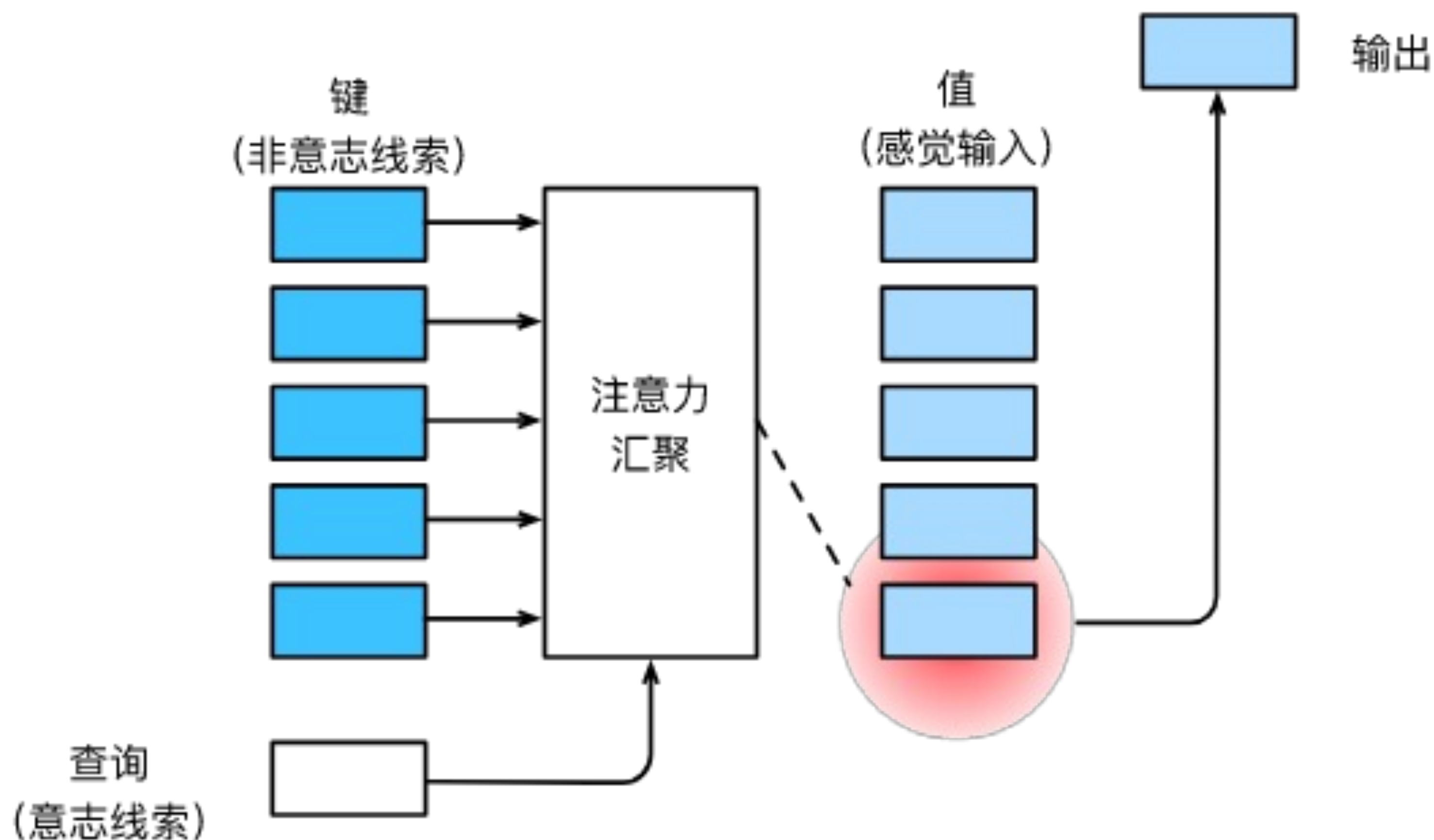
自主性(受认知控制)



“是否包含自主性提示”将注意力机制与全连接层或汇聚层区别开

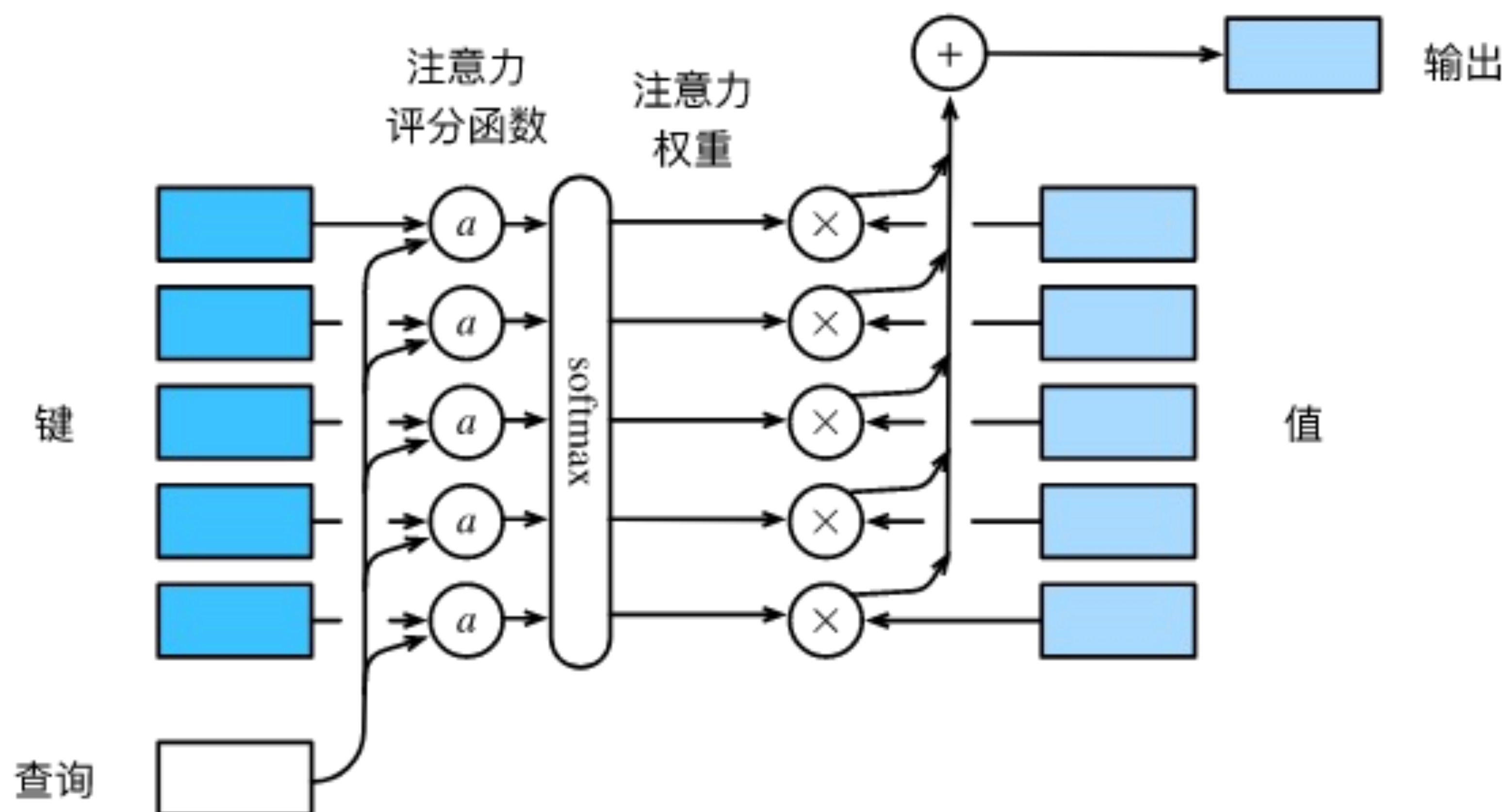
注意力模型

- 通过设计注意力汇聚的方式， 便于给定的查询（自主性提示）与键（非自主性提示）进行匹配， 这将引导得出最匹配的值（感官输入）。



注意力模型

- 意力汇聚的输出计算成为值的加权和， 其中a表示注意力评分函数



注意力模型

- 注意力汇聚的输出计算成为值的加权和， 其中a表示注意力评分函数

查询q和 m个“键一值”对 (k1,v1),…,(km,vm)， 注意力汇聚函数f表示为：

$$f(\mathbf{q}, (\mathbf{k}_1, \mathbf{v}_1), \ldots, (\mathbf{k}_m, \mathbf{v}_m)) = \sum_{i=1}^{m} \alpha(\mathbf{q}, \mathbf{k}_i)\mathbf{v}_i \in \mathbb{R}^v,$$

$$\alpha(\mathbf{q}, \mathbf{k}_i) = \mathrm{softmax}(a(\mathbf{q}, \mathbf{k}_i)) = \frac{\exp(a(\mathbf{q}, \mathbf{k}_i))}{\sum_{j=1}^{m} \exp(a(\mathbf{q}, \mathbf{k}_j))} \in \mathbb{R}.$$

注意力评分函数a不同会导致不同的注意力汇聚操作

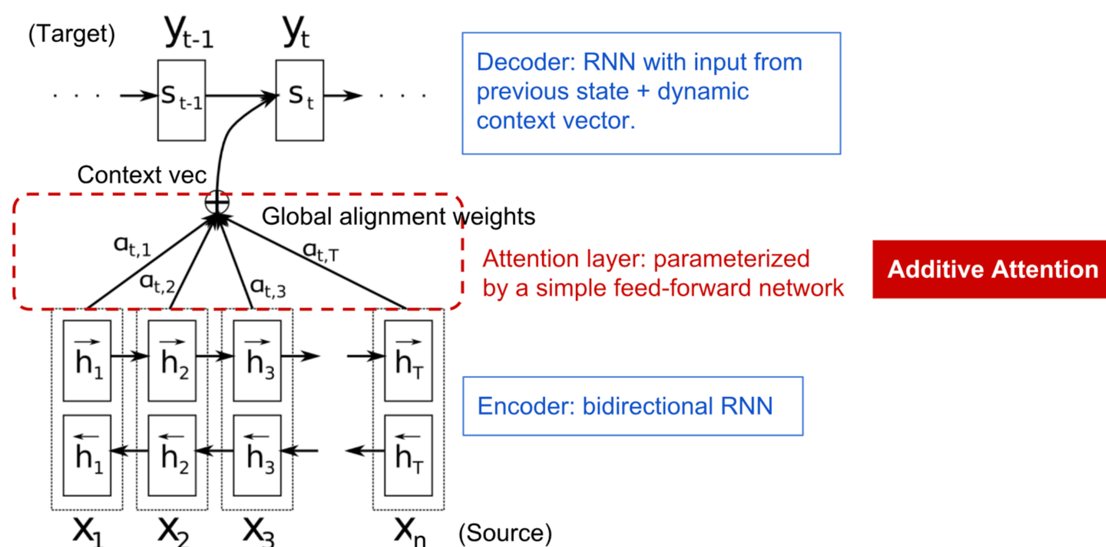
D. Bahdanau, K. Cho, and Y. Bengio. "Neural machine translation by jointly learning to align and translate.” (2014)

注意力模型

- 加性注意力

$$\alpha(\mathbf{q}, \mathbf{k}_i) = \mathrm{softmax}(a(\mathbf{q}, \mathbf{k}_i)) = \frac{\exp(a(\mathbf{q}, \mathbf{k}_i))}{\sum_{j=1}^{m} \exp(a(\mathbf{q}, \mathbf{k}_j))} \in \mathbb{R}.$$

$$a(\mathbf{q}, \mathbf{k}) = \mathbf{w}_v^\top \tanh(\mathbf{W}_q \mathbf{q} + \mathbf{W}_k \mathbf{k}) \in \mathbb{R},$$



Additive Attention

注意力模型

- 缩放点积注意力

$$\alpha(\mathbf{q}, \mathbf{k}_i) = \mathrm{softmax}(a(\mathbf{q}, \mathbf{k}_i)) = \frac{\exp(a(\mathbf{q}, \mathbf{k}_i))}{\sum_{j=1}^{m} \exp(a(\mathbf{q}, \mathbf{k}_j))} \in \mathbb{R}.$$

$$a(\mathbf{q}, \mathbf{k}) = \mathbf{q}^\top \mathbf{k} / \sqrt{d}.$$

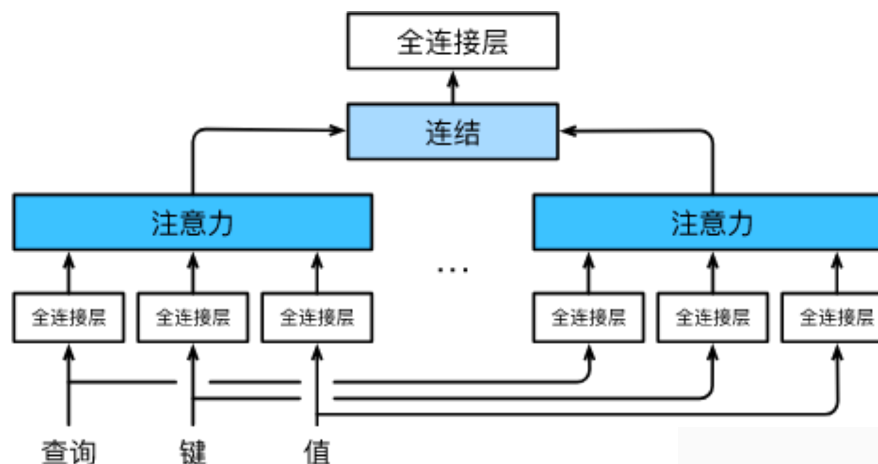
基于n个查询和m个键一值对计算注意力， 其中查询和键的长度为d，值的长度为v。 查询***Q***、 键***K***和 值***V***的缩放点积注意力是：

$$\mathrm{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d}}\right)\mathbf{V} \in \mathbb{R}^{n \times v}.$$

D. Bahdanau, K. Cho, and Y. Bengio. "Neural machine translation by jointly learning to align and translate." (2014)

注意力模型

- 多头注意力



$$\mathbf{h}_i = f(\mathbf{W}_i^{(q)}\mathbf{q}, \mathbf{W}_i^{(k)}\mathbf{k}, \mathbf{W}_i^{(v)}\mathbf{v}) \in \mathbb{R}^{p_v},$$

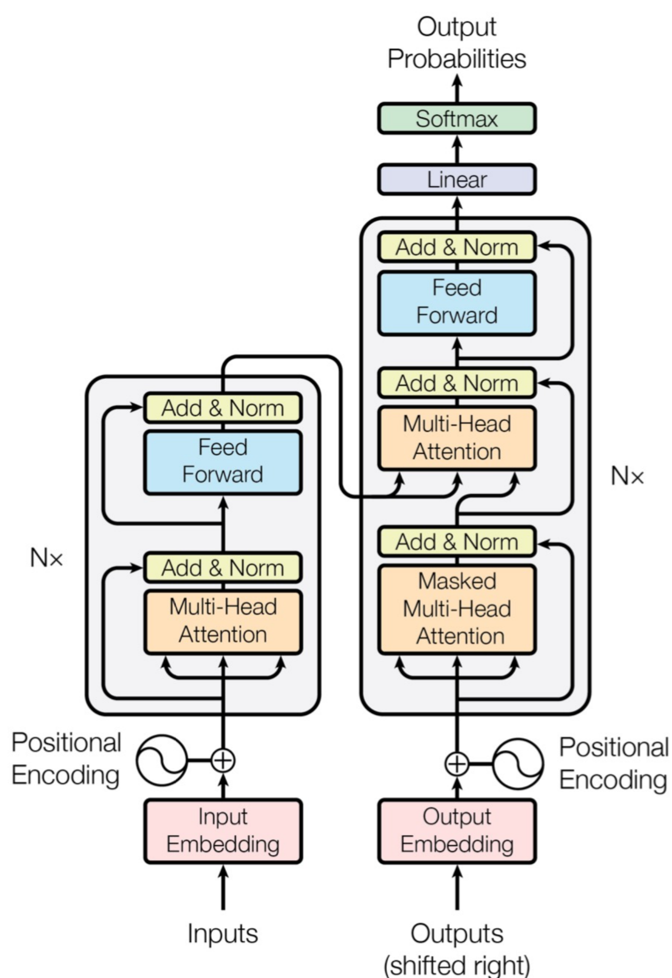
$$\mathbf{W}_o \begin{bmatrix} \mathbf{h}_1 \\ \vdots \\ \mathbf{h}_h \end{bmatrix} \in \mathbb{R}^{p_o}.$$

注意力汇聚函数f，可以是加性注意力或者缩放点积注意力

D. Bahdanau, K. Cho, and Y. Bengio. "Neural machine translation by jointly learning to align and translate." (2014)

注意力模型

变换，Transformer



Transformer的编码器和解码器是基于自注意力的模块叠加而成的，源（输入）序列和目标（输出）序列的*嵌入*（embedding）表示将加上*位置编码*（positional encoding），再分别输入到编码器和解码器中。

- 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.

Vaswani, Ashish, et al. "Attention is all you need." (2017)

注意力模型

- 位置编码（positional encoding）
  - 为了使用序列的顺序信息，通过在输入表示中添加 *位置编码*（positional encoding）来注入绝对的或相对的位置信息。

$X \in R^{n\times d}$ 表示一个序列中n个词元的d维嵌入表示， 位置编码使用相同形状的位置嵌入矩阵 $P \in R^{n\times d}$输出$X + P$， 矩阵第i行、第2j列和2j+1列上的元素为：

$$p_{i,2j} = \sin\left(\frac{i}{10000^{2j/d}}\right),$$
$$p_{i,2j+1} = \cos\left(\frac{i}{10000^{2j/d}}\right).$$

P中行代表词元在序列中的位置，列代表位置编码的不同维度

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)

阅读资料

- 《数学之美》,作者：吴军

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