

Language Modeling

Language Modeling是预测序列的下一个元素的任务。比如输入联想。

n-gram

n-gram指n个词一组，作为一个样本。idea是从数据库里找到大量n词一组的样本，直接通过计数计算条件概率。

首先，我们简单假设第i个词出现的概率只取决于(i-n,...,i-1)这n个词(顺序必须也固定):

n-gram Language Models

- First we make a **simplifying assumption**: $x^{(t+1)}$ depends only on the preceding $n-1$ words.

$$P(x^{(t+1)} | x^{(t)}, \dots, x^{(1)}) = P(x^{(t+1)} | \overbrace{x^{(t)}, \dots, x^{(t-n+2)}}^{n-1 \text{ words}}) \quad (\text{assumption})$$

prob of a n-gram \rightarrow $P(x^{(t+1)}, x^{(t)}, \dots, x^{(t-n+2)})$
= $P(x^{(t)}, \dots, x^{(t-n+2)})$ (definition of conditional prob)
prob of a (n-1)-gram \rightarrow

- Question:** How do we get these n -gram and $(n-1)$ -gram probabilities?
- Answer:** By **counting** them in some large corpus of text!

$$\approx \frac{\text{count}(x^{(t+1)}, x^{(t)}, \dots, x^{(t-n+2)})}{\text{count}(x^{(t)}, \dots, x^{(t-n+2)})} \quad (\text{statistical approximation})$$

8

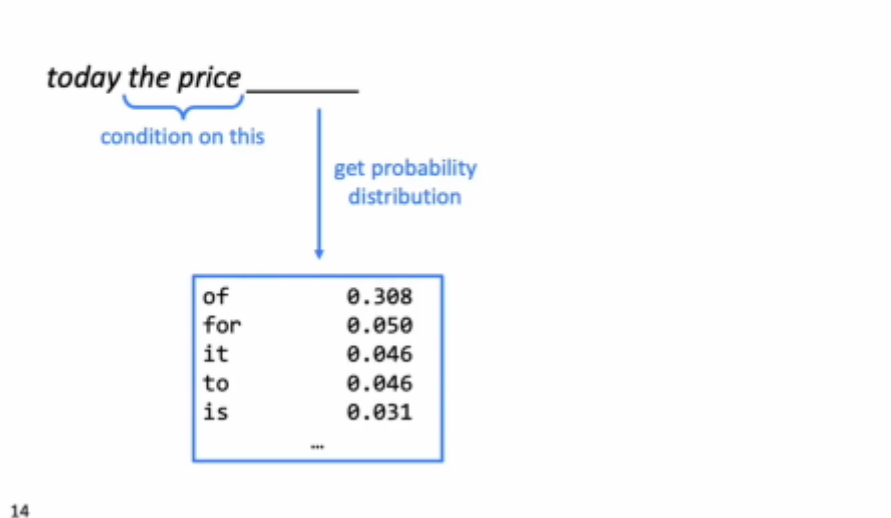
n-gram方法的缺点

- 可能取决于更之前的context，不在n-gram中
- sparsity problem: 分子可能为0，需要增加假样本(数据平滑)。分母可能为0，需要从n-gram退化到(n-1)-gram
- storage problem: 需要存储语料库中所有n-gram。

可以用language model来生成text，即逐一生成单词:

Generating text with a n-gram Language Model

- You can also use a Language Model to **generate text**.



14

Generating text with a n-gram Language Model

- You can also use a Language Model to **generate text**.

today the price of gold per ton , while production of shoe lasts and shoe industry , the bank intervened just after it considered and rejected an imf demand to rebuild depleted european stocks , sept 30 end primary 76 cts a share .

Surprisingly grammatical!

...but incoherent. We need to consider more than three words at a time if we want to model language well.

But increasing n worsens sparsity problem, and increases model size...

17

Build a neural language model

输入($50 \times \text{window size}$)维的向量, 经过 $[50, 50 \times \text{window size}]$ 的 W , 再乘以 U , 得到10000个单词各自生成的概率。

A fixed-window neural Language Model

output distribution

$$\hat{y} = \text{softmax}(Uh + b_2) \in \mathbb{R}^{|V|}$$

hidden layer

$$h = f(We + b_1)$$

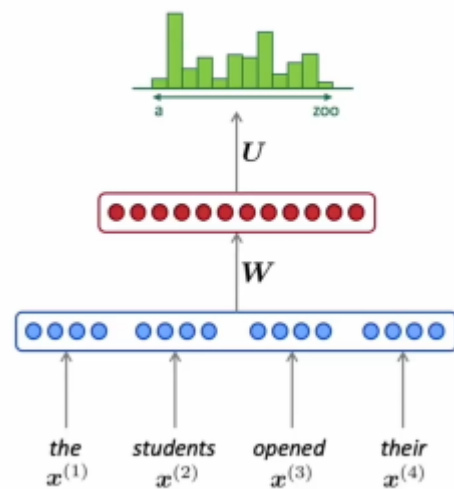
concatenated word embeddings

$$e = [e^{(1)}; e^{(2)}; e^{(3)}; e^{(4)}]$$

words / one-hot vectors

$$x^{(1)}, x^{(2)}, x^{(3)}, x^{(4)}$$

20



优点

- 没有sparsity problem
- 没有storage problem, 不用存n-gram

缺点

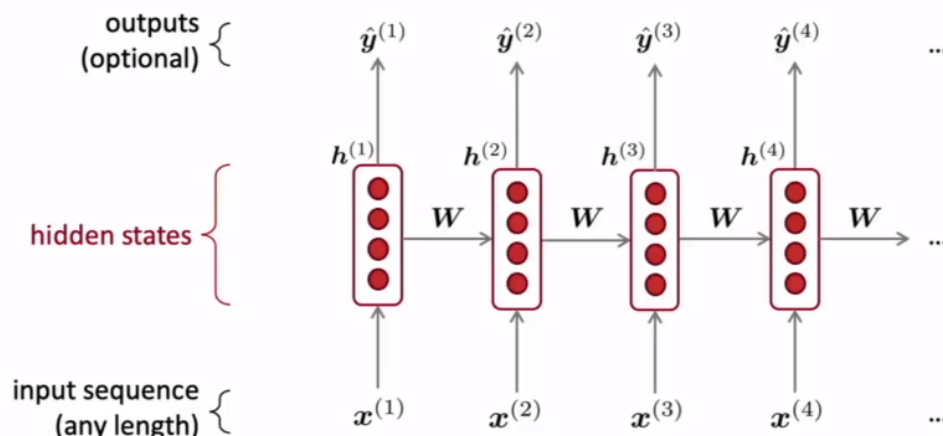
- window太小, 可能失去上下文
- window太大, W 也要增大
- $x^{(1)}$ 和 $x^{(2)}$ 在权重上没有联系

Recurrent Neural Networks (RNN)

Recurrent Neural Networks (RNN)

A family of neural architectures

Core idea: Apply the same weights W repeatedly



22

上图是RNN的简图。

注:

- h 是和时序有关的隐藏层
- y 可以输出, 也可以不输出
- W 变换是每个隐藏层共享的

A RNN Language Model

output distribution

$$\hat{y}^{(t)} = \text{softmax}(Uh^{(t)} + b_2) \in \mathbb{R}^{|V|}$$

hidden states

$$h^{(t)} = \sigma(W_h h^{(t-1)} + W_e e^{(t)} + b_1)$$

$h^{(0)}$ is the initial hidden state

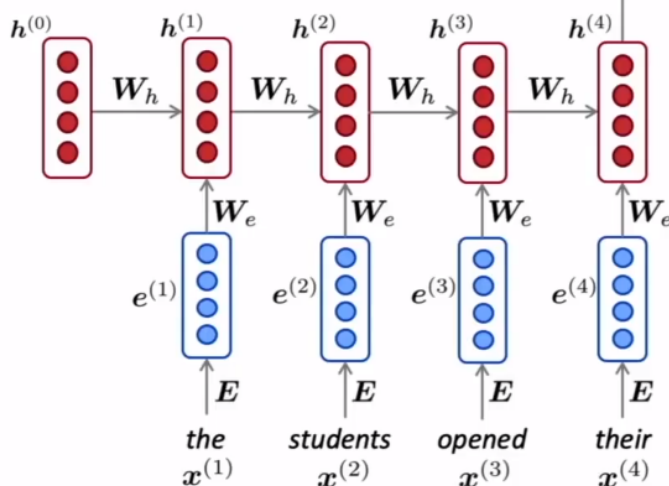
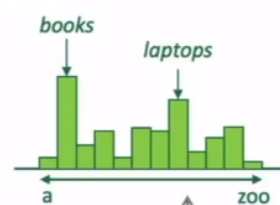
word embeddings

$$e^{(t)} = Ex^{(t)}$$

words / one-hot vectors

$$x^{(t)} \in \mathbb{R}^{|V|}$$

$$\hat{y}^{(4)} = P(x^{(5)} | \text{the students opened their})$$



23

Note: this input sequence could be much longer, but this slide doesn't have space!

上图是RNN的细节图和推导。wordvec可以训练, 也可以不训练。

优点

- 可以处理任意长度输入，不用固定window size
- 每一步都包含前面每一步的信息
- 因为共享权重，模型大小不会随输入变大
- 因为共享权重，不同输入会经过相同的权重变换
- 因为共享权重，学到的是更general的权重，而不是针对某些样本

缺点

- 计算很慢，必须串行计算
- 很难得到很多step之前的信息

注：

- 假设hidden unit有n个units，那么 W_h 维度是(n, n)， W_e 维度是(n, d)

Train a RNN language model

需手推：

Training a RNN Language Model

- Get a **big corpus of text** which is a sequence of words $x^{(1)}, \dots, x^{(T)}$
- Feed into RNN-LM; compute output distribution $\hat{y}^{(t)}$ **for every step t**.
 - i.e. predict probability dist of *every word*, given words so far
- **Loss function** on step t is **cross-entropy** between predicted probability distribution $\hat{y}^{(t)}$, and the true next word $y^{(t)}$ (one-hot for $x^{(t+1)}$):

$$J^{(t)}(\theta) = CE(y^{(t)}, \hat{y}^{(t)}) = - \sum_{w \in V} y_w^{(t)} \log \hat{y}_w^{(t)} = - \log \hat{y}_{x_{t+1}}^{(t)}$$

- Average this to get **overall loss** for entire training set:

$$J(\theta) = \frac{1}{T} \sum_{t=1}^T J^{(t)}(\theta) = \frac{1}{T} \sum_{t=1}^T - \log \hat{y}_{x_{t+1}}^{(t)}$$

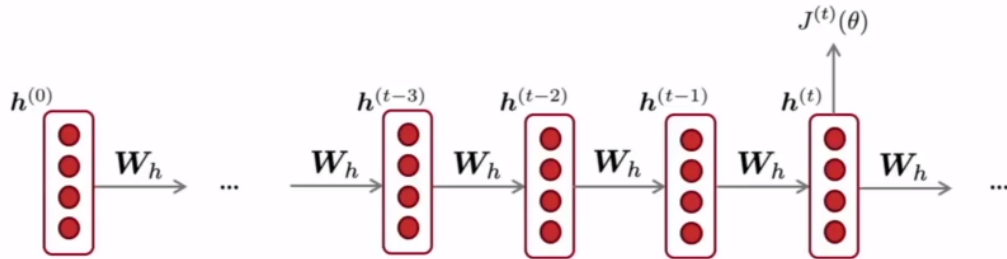
25

每个hidden layer都输出y计算loss消耗太大，一般训练时以sentence为单位，只计算最后一层的y的loss

BP for RNN

W_h 的梯度是每一层loss产生梯度的和。#这里下标是i不是t，即 $J^{(t)}$ 对每一层 W_h 的累加，具体到 $\frac{\partial J^{(t)}}{\partial W_h}$ 还要展开用链式法则(下一章有讲，梯度消失问题)

Backpropagation for RNNs



Question: What's the derivative of $J^{(t)}(\theta)$ w.r.t. the **repeated** weight matrix W_h ?

Answer:
$$\frac{\partial J^{(t)}}{\partial W_h} = \sum_{i=1}^t \frac{\partial J^{(t)}}{\partial W_h} \Big|_{(i)}$$

"The gradient w.r.t. a repeated weight is the sum of the gradient w.r.t. each time it appears"

Why?

32

Backpropagation for RNNs: Proof sketch

- Given a multivariable function $f(x, y)$, and two single variable functions $x(t)$ and $y(t)$, here's what the multivariable chain rule says:

$$\underbrace{\frac{d}{dt} f(x(t), y(t))}_{\text{Derivative of composition function}} = \frac{\partial f}{\partial x} \frac{dx}{dt} + \frac{\partial f}{\partial y} \frac{dy}{dt}$$

In our example:

Apply the multivariable chain rule:

$$\frac{\partial J^{(t)}}{\partial W_h} = \sum_{i=1}^t \frac{\partial J^{(t)}}{\partial W_h} \Big|_{(i)} \boxed{\frac{\partial W_h|_{(i)}}{\partial W_h}} = 1$$

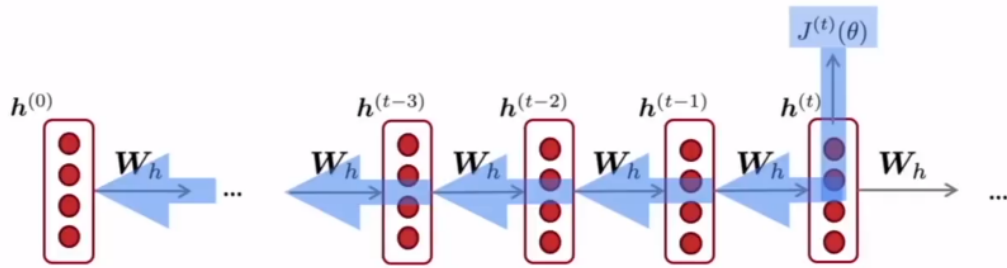
$$= \sum_{i=1}^t \frac{\partial J^{(t)}}{\partial W_h} \Big|_{(i)}$$

Source:

<https://www.khanacademy.org/math/multivariable-calculus/multivariable-derivatives/differentiating-vector-valued-functions/a/multivariable-chain-rule-simple-version>

34

Backpropagation for RNNs



$$\frac{\partial J^{(t)}}{\partial W_h} = \sum_{i=1}^t \frac{\partial J^{(t)}}{\partial W_h} \Big|_{(i)}$$

Question: How do we calculate this?

Answer: Backpropagate over timesteps $i=t, \dots, 0$, summing gradients as you go. This algorithm is called **"backpropagation through time"**

35

注:

- 前向传播全部结束后，会计算 W_h 的总梯度，最后只梯度下降一次

Evaluation Language Models

perplexity是预测概率的倒数的乘积，越低越好。

Evaluating Language Models

- The standard **evaluation metric** for Language Models is **perplexity**.

$$\text{perplexity} = \prod_{t=1}^T \left(\frac{1}{P_{\text{LM}}(\mathbf{x}^{(t+1)} | \mathbf{x}^{(t)}, \dots, \mathbf{x}^{(1)})} \right)^{1/T}$$

← Normalized by number of words

Inverse probability of corpus, according to Language Model

- This is equal to the exponential of the cross-entropy loss $J(\theta)$:

$$= \prod_{t=1}^T \left(\frac{1}{\hat{\mathbf{y}}_{\mathbf{x}_{t+1}}^{(t)}} \right)^{1/T} = \exp \left(\frac{1}{T} \sum_{t=1}^T -\log \hat{\mathbf{y}}_{\mathbf{x}_{t+1}}^{(t)} \right) = \exp(J(\theta))$$

Lower perplexity is better!

41

Why Language Modeling

Why should we care about Language Modeling?

- Language Modeling is a **benchmark task** that helps us **measure our progress** on understanding language
- Language Modeling is a **subcomponent** of many NLP tasks, especially those involving **generating text** or **estimating the probability of text**:
 - Predictive typing
 - Speech recognition
 - Handwriting recognition
 - Spelling/grammar correction
 - Authorship identification
 - Machine translation
 - Summarization
 - Dialogue
 - etc.

often it's kind of noisy and hard to make out what they're say

总结

Recap

- Language Model: A system that predicts the next word
- Recurrent Neural Network: A family of neural networks that:
 - Take sequential input of any length
 - Apply the same weights on each step
 - Can optionally produce output on each step
- Recurrent Neural Network \neq Language Model
- We've shown that RNNs are a great way to build a LM.
- But RNNs are useful for much more!

44

注：RNN不是language model，但是可以用来构建一个language model。RNN还可以做很多别的事情。

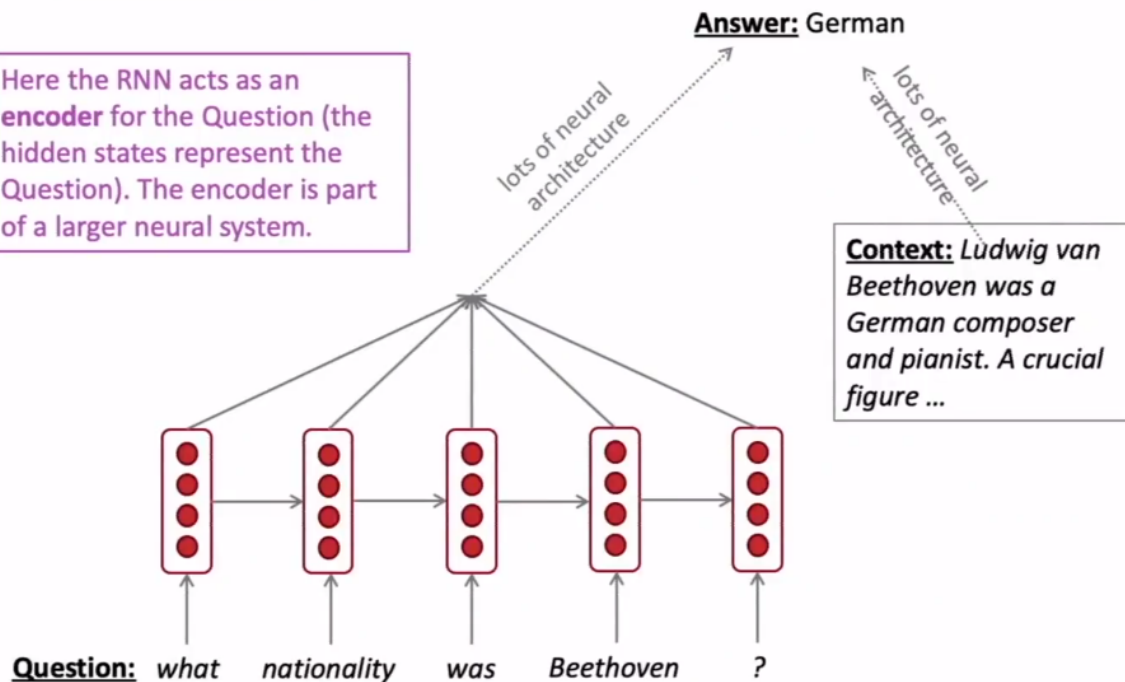
RNN还可以做：

- part of speech tagging
- sentence classification(like sentiment classification)
- general purpose encoder module(like question answering, machine translation)
- generate text(like speech recognition)

RNNs can be used as an encoder module

e.g. question answering, machine translation, *many other tasks!*

Here the RNN acts as an **encoder** for the Question (the hidden states represent the Question). The encoder is part of a larger neural system.



49

A note on terminology

RNN described in this lecture = "vanilla RNN"



Next lecture: You will learn about other RNN flavors

like **GRU**



and **LSTM**



and multi-layer RNNs



By the end of the course: You will understand phrases like
"stacked bidirectional LSTM with residual connections and self-attention"



51