

some concepts

machine learning and deep learning

[机器学习和深度学习区别的简要概述 - 知乎 \(zhihu.com\)](#)

深度学习 基于神经网络

神经网络

首先把人类眼中的特征转换成计算机的向量

把特征进行变化，特征：x身高+y体重等

神经网络生成很多特征，最终进行测算

特征的系数由神经网络学习得到

综上所述，深度学习是一种机器学习方法，神经网络是其基础模型，而大模型则是指具有大规模参数和复杂结构的神经网络模型。深度学习利用神经网络来学习数据的表示，而大模型则是神经网络的一种特例，具有更大的规模和复杂性。

NLP

pytorch

LECTURE 1

basic acknowledge

representing words as discrete symbols

Representing words as discrete symbols

In traditional NLP, we regard words as discrete symbols:
`hotel, conference, motel` – a `localist` representation

Means one 1, the rest 0s

Such symbols for words can be represented by `one-hot` vectors:

```
motel = [0 0 0 0 0 0 0 0 0 1 0 0 0]  
hotel = [0 0 0 0 0 0 1 0 0 0 0 0 0]
```

Vector dimension = number of words in vocabulary (e.g., 500,000)

one-hot向量

one-hot向量将类别变量转换为机器学习算法易于利用的一种形式的过程，这个向量的表示为一项属性的特征向量，也就是同一时间只有一个激活点（不为0），这个向量只有一个特征是不为0的，其他都是0，特别稀疏。

problem with words as discrete symbols

Problem with words as discrete symbols

Example: in web search, if user searches for “Seattle motel”, we would like to match documents containing “Seattle hotel”

But:

```
motel = [0 0 0 0 0 0 0 0 0 1 0 0 0]  
hotel = [0 0 0 0 0 0 1 0 0 0 0 0 0]
```

These two vectors are orthogonal

There is no natural notion of **similarity** for one-hot vectors!

Solution:

- Could try to rely on WordNet’s list of synonyms to get similarity?
 - But it is well-known to fail badly: incompleteness, etc.
- **Instead: learn to encode similarity in the vectors themselves**

¹⁶

虽然单词可以被表达为向量，但是在数学逻辑中，这两个向量是正交的，没有任何的自然联系

representing words by their context

Representing words by their context

- **Distributional semantics: A word’s meaning is given by the words that frequently appear close-by**
 - “*You shall know a word by the company it keeps*” (J. R. Firth 1957: 11)
 - One of the most successful ideas of modern statistical NLP!
- When a word w appears in a text, its **context** is the set of words that appear nearby (within a fixed-size window).
- Use the many contexts of w to build up a representation of w

```
...government debt problems turning into banking crises as happened in 2009...  
...saying that Europe needs unified banking regulation to replace the hodgepodge...  
...India has just given its banking system a shot in the arm...
```

These context words will represent **banking**

为了解决离散的词组无法被计算机认知的问题

人们提出分布式语义让计算机去了解单词

其中最成功的idea就是NLP

分布式语义：代表的是一个词的含义其实是由和伴随这个词经常出现的上下文给出的

word can be sorted refer to token and type

1. **Token**: Token指的是文本中的最小单位，通常是指由空格或标点符号分隔开的单词或符号。在某些情况下，token也可以是一个更大的单位，比如一个词组或短语。在NLP任务中，文本通常会被分解成一系列token，每个token代表文本中的一个基本单位。
2. **Type**: Type指的是不同的token种类或类型。换句话说，type是指所有不同token的集合。例如，在句子"I love love love NLP."中，有6个token (I, love, love, love, NLP, .)，但是只有4个不同的type (I, love, NLP, .)。

文中第一行的banking是这段话的token

这三句话banking都是一种类型，所以在这三句话中，我们用banking代表一种类型

word vectors

Word vectors

We will build a dense vector for each word, chosen so that it is similar to vectors of words that appear in similar contexts

$$\text{banking} = \begin{pmatrix} 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 0.109 \\ -0.542 \\ 0.349 \\ 0.271 \end{pmatrix}$$

Note: word vectors are also called word embeddings or (neural) word representations
They are a distributed representation

18

我们想使用vector去表示每个单词，让它不仅能够表达含义，还可以预测上下文

Word2vec:overview

overview

Word2Vec是一种广泛使用的自然语言处理（NLP）技术，用于学习词语的向量表示。它是由Google的研究团队在2013年提出的。Word2Vec的核心思想是将词语映射到一个高维空间中的向量，使得语义相近的词语在这个空间中的向量也相近。这种向量表示可以捕捉词语之间的语义关系，如同义、反义、上下位等。

3. Word2vec: Overview

Word2vec (Mikolov et al. 2013) is a framework for learning word vectors

Idea:

- We have a large corpus (“body”) of text
- Every word in a fixed vocabulary is represented by a vector
- Go through each position t in the text, which has a center word c and context (“outside”) words o
- Use the similarity of the word vectors for c and o to calculate the probability of o given c (or vice versa)
- Keep adjusting the word vectors to maximize this probability

首先我要有一个巨大的语料库

然后把每个单词用vector去表示

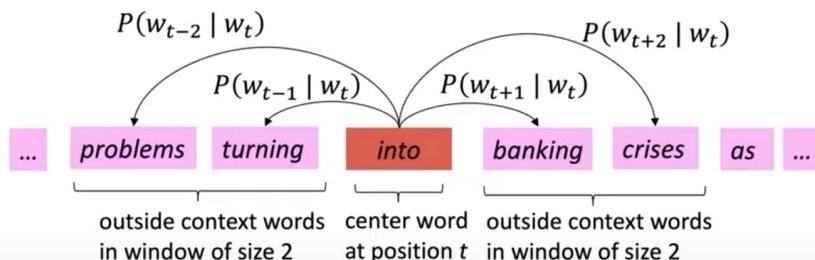
在一段文本中选择一个中心词，和上下文单词 o

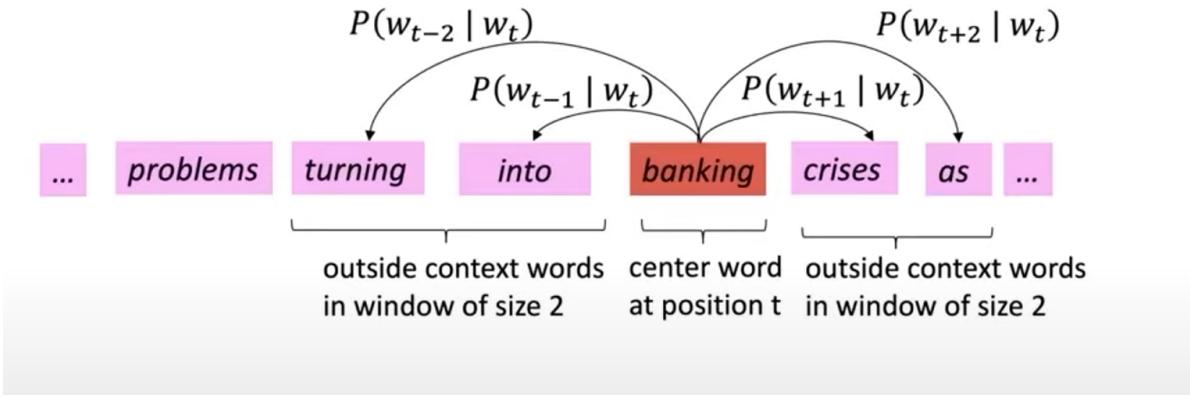
用 c 和 o 的向量去计算给中心词 c 出现 o 的概率

不断调整单词的向量 让他的可能性最大

Word2Vec Overview

Example windows and process for computing $P(w_{t+j} | w_t)$





为了提高出现上下文的概率

我们不断改变中心词 然后预测

objective function

Word2vec: objective function

For each position $t = 1, \dots, T$, predict context words within a window of fixed size m , given center word w_t . Data likelihood:

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

θ is all variables
to be optimized

sometimes called a *cost* or *loss* function

The **objective function** $J(\theta)$ is the (average) negative log likelihood:

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

Minimizing objective function \Leftrightarrow Maximizing predictive accuracy

??

1. **Likelihood**: 在Word2Vec的上下文中，似然 (likelihood) 通常指的是给定一个词和其上下文时，模型正确预测这个上下文出现的概率。例如，在Skip-gram模型中，给定一个中心词，似然就是模型正确预测其周围上下文词的概率。似然越高，表示模型对训练数据的拟合程度越好。
2. **Objective Function**: 目标函数 (objective function) 是在训练过程中需要优化的函数。在 Word2Vec中，通常使用对数似然函数 (log-likelihood function) 作为目标函数。对于Skip-gram模型，目标函数是最大化所有词及其上下文的对数似然之和。对于CBOW模型，目标函数则是最大化给定上下文词预测中心词的对数似然之和。

目标函数的用途是指导模型的训练过程。通过调整模型参数 (即词向量) 来最大化目标函数，Word2Vec模型能够学习到能够反映词之间语义关系的词向量。

Word2vec: objective function

- We want to minimize the objective function:
$$J(\theta) = -\frac{1}{T} \sum_{t=1}^T \sum_{\substack{-m \leq j \leq m \\ j \neq 0}} \log P(w_{t+j} | w_t; \theta)$$
- **Question:** How to calculate $P(w_{t+j} | w_t; \theta)$?
- **Answer:** We will use two vectors per word w :
 - v_w when w is a center word
 - u_w when w is a context word
- Then for a center word c and a context word o :

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

24

prediction function

Word2vec: prediction function

② Exponentiation makes anything positive
① Dot product compares similarity of o and c .
 $u^T v = u \cdot v = \sum_{i=1}^n u_i v_i$
Larger dot product = larger probability

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

③ Normalize over entire vocabulary to give probability distribution

- This is an example of the **softmax function** $\mathbb{R}^n \rightarrow (0,1)^n$
- $$\text{softmax}(x_i) = \frac{\exp(x_i)}{\sum_{j=1}^n \exp(x_j)} = p_i$$

- Open region
- The softmax function maps arbitrary values x_i to a probability distribution p_i
 - “max” because amplifies probability of largest x_i
 - “soft” because still assigns some probability to smaller x_i
 - Frequently used in Deep Learning

But sort of a weird name because it returns a distribution!

26

用于预测上下文

我们通过dot product代表两个向量，这里是中心词和上下文向量的相似度

softmax函数被用来计算给定中心词的上下文词的概率分布

其中， $p(c|w)$ 表示给定中心词 w 时，上下文词 c 出现的概率。

总而言之，首先通过向量表示文字，表示要与上下文紧密联系，使得具有相似上下文的单词向量相近，在大量学习之后，我们可以通过似然函数推理出上下文单词出现概率，从而预测上下文

to train the model: optimize value if parameters to minimize loss

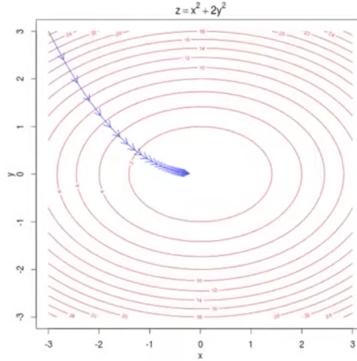
在生成词向量的过程，如何减少word->vec的损失

To train the model: Optimize value of parameters to minimize loss

To train a model, we gradually adjust parameters to minimize a loss

- Recall: θ represents **all** the model parameters, in one long vector
- In our case, with d -dimensional vectors and V -many words, we have:
- Remember: every word has two vectors

$$\theta = \begin{bmatrix} v_{aardvark} \\ v_a \\ \vdots \\ v_{zebra} \\ u_{aardvark} \\ u_a \\ \vdots \\ u_{zebra} \end{bmatrix} \in \mathbb{R}^{2dV}$$



- We optimize these parameters by walking down the gradient (see right figure)
- We compute **all** vector gradients!

这张图片展示了一个参数向量 θ ，它包含了一系列的词向量。在这里，“ R ”表示实数集，“ $2dV$ ”表示参数向量的维数，其中(V)是词汇表的大小，(d)是每个词向量的维数。因此，(\mathbb{R}^{2dV})表示参数向量 θ 属于一个维数为($2dV$)的实数空间，这意味着每个词有两个向量表示：一个用作中心词(通常用(v)表示)，另一个用作上下文词(通常用(u)表示)，每个向量都有(d)维。所以，如果你有一个词汇表大小为(V)的语料库，并且选择了一个(d)维的向量空间来表示每个词，那么整个模型的参数空间的维数将是($2dV$)，包括所有词的中心词向量和上下文词向量。

在Word2Vec模型中提到的计算向量梯度是指在训练过程中，为了最大化似然函数，我们需要计算目标函数(通常是对数似然函数)相对于每个词向量参数的导数，即梯度。

梯度计算公式的细节取决于所用的目标函数，但基本过程如下：

- 正向传播**: 给定一个中心词，模型使用当前的词向量计算上下文词出现的概率。
- 计算损失**: 使用目标函数计算预测概率和真实概率之间的差异。
- 反向传播**: 计算损失函数关于每个词向量的梯度。这涉及到对目标函数求导，并将这些导数(梯度)反向传播回网络，以便更新词向量。
- 更新词向量**: 使用梯度和学习率来更新词向量，这个过程称为梯度下降(或在最大化问题中的梯度上升)

whole process

1. 初始化：

- **选择模型**：确定使用CBOW (Continuous Bag of Words) 模型还是Skip-gram模型。
- **准备数据**：从文本语料库中收集训练数据，通常是单词及其上下文。
- **参数初始化**：随机初始化词向量。每个词有两个向量表示：作为中心词的向量和作为上下文词的向量。

2. 训练：

- **滑动窗口**：在语料库中滑动窗口，选取中心词和上下文词。
- **生成训练样本**：在Skip-gram模型中，训练样本是中心词与其周围上下文词的配对。在CBOW模型中，是上下文词的集合和目标中心词的配对。
- **计算概率**：使用softmax函数或其近似（比如负采样或层次softmax）来计算给定中心词下上下文词出现的概率。
- **计算损失**：通过比较预测的概率分布和实际的分布，计算损失函数。
- **计算梯度**：通过反向传播算法计算损失函数相对于每个词向量的偏导数，即梯度。
- **更新词向量**：使用计算出的梯度和学习率来更新每个词的向量。

3. 重复训练过程：

- **迭代优化**：重复上述训练过程多次，直到损失函数的值趋于稳定或达到预设的迭代次数。

4. 预测（使用模型）：

- **获取词向量**：一旦训练完成，每个词的向量表示可以从模型中提取出来。
- **词语相似度**：通过计算词向量之间的余弦相似度，可以找到语义上相似的词。
- **类比推理**：可以使用向量运算（如王后 - 女性 + 男性 = 国王）来进行词语间的类比推理。
- **新词预测**：虽然Word2Vec不是一个传统意义上的预测模型，但可以通过向量的加减运算预测和某些词语语义相近的词。

5. 结果使用：

- **NLP任务**：训练得到的词嵌入可以作为特征输入到各种自然语言处理任务中，如文本分类、情感分析、机器翻译等。

1. **实际概率**：实际概率（也称为观测概率或真实概率）是指在训练数据中，一个词的上下文词实际出现的概率分布。这是从训练集中直接得到的统计信息。例如，在一个大的文本语料库中，单词“bank”后面跟着“river”和“money”的频率可以决定“river”和“money”作为“bank”上下文的实际概率。
2. **预测概率**：预测概率是模型基于当前参数（词向量）计算得到的，给定一个词时其上下文词出现的概率。在Word2Vec模型中，这是通过softmax函数或其近似算法（如负采样或层次softmax）计算得到的。

在Word2Vec的训练过程中，我们希望模型预测的概率分布尽可能接近实际的概率分布。这就需要一个衡量标准，通常是损失函数，比如交叉熵损失函数，它可以衡量两个概率分布之间的差异。

关于偏导数和梯度的问题：

1. **偏导数**：在多变量函数中，偏导数是指函数对单一变量的导数，而保持其他变量恒定。在Word2Vec中，每个词都由一个向量表示，这些向量的每个元素都是多变量损失函数的变量。
2. **梯度**：梯度是由一个函数的所有偏导数组成的向量。在最优化问题中，梯度指向函数在给定点上增长最快的方向。因此，在Word2Vec中，我们计算损失函数相对于每个词向量的偏导数，所有这些偏导数组合起来形成了损失函数的梯度。

在梯度下降优化算法中，梯度的负方向指示了减少损失函数值的方向，我们通过在梯度的负方向上更新词向量来训练模型，这样做可以使预测概率分布更接近实际概率分布。

lecture2

Gradient Descent

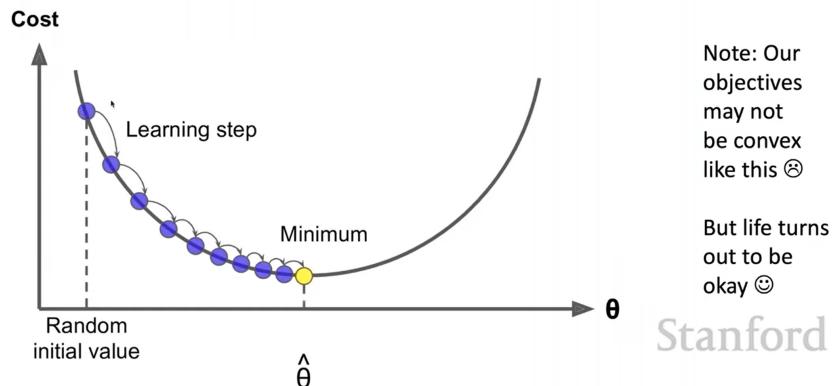
梯度下降算法，这里我们需要用梯度下降算法最小化目标函数 J 通过改边delta

方法就是朝梯度相反方向做出改变，直达到到最小值

3. Optimization: Gradient Descent



- To learn good word vectors: We have a cost function $J(\theta)$ we want to minimize
- **Gradient Descent** is an algorithm to minimize $J(\theta)$ by changing θ
- **Idea:** from current value of θ , calculate gradient of $J(\theta)$, then take **small step in the direction of negative gradient**. Repeat.



这张图片展示的是机器学习中常用的两个数学更新方程，特别是在梯度下降这样的优化算法中。

梯度并不是越大越好，在梯度下降算法中，如果梯度为0，说明达到最大值

Stochastic Gradient Descent

随机的梯度下降，用于解决梯度下降在问题中成本太过昂贵的问题，它只需要采样一小部分的中心词进行梯度计算估计就可以了

Stochastic Gradient Descent



- **Problem:** $J(\theta)$ is a function of **all** windows in the corpus (often, billions!)
 - So $\nabla_{\theta} J(\theta)$ is **very expensive to compute**
- You would wait a very long time before making a single update!
- Very bad idea for pretty much all neural nets!
- **Solution: Stochastic gradient descent (SGD)**
 - Repeatedly sample windows, and update after each one, or each small batch
- Algorithm:

```
while True:  
    window = sample_window(corpus)  
    theta_grad = evaluate_gradient(J, window, theta)  
    theta = theta - alpha * theta_grad
```

Stanford

在实际应用中，由于计算完整softmax代价很大（需要对整个词汇表的单词进行计算），通常使用一些近似方法，如负采样（Negative Sampling）或层次softmax（Hierarchical Softmax），以提高训练的效率。这些方法减少了需要更新的权重数量，从而加快了训练过程。负采样通过随机选择一小部分“负样本”（即上下文中没有出现的单词），而层次softmax通过构建一个二叉树来减少计算量，每个单词都是树的一个叶节点。这样，在更新权重时，只需要沿着从根节点到目标单词的叶节点的路径进行计算，而不是对整个词汇表进行计算。

The skip-gram model with negative sampling (hw2)

The skip-gram model with negative sampling (HW2)



- The normalization term is computationally expensive
- $$P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)}$$
- Hence, in standard word2vec and HW2 you implement the skip-gram model with **negative sampling**
- Main idea: train binary logistic regressions for a true pair (center word and a word in its context window) versus several noise pairs (the center word paired with a random word)

CS224N

对于softmax方法计算概率并进行采样，我们需要遍历每个单词，所以这个过程消耗很大

所以在skipgram model中

- 我们将在作业2中实现使用 negative sampling/负例采样方法的 skip-gram 模型。
- 使用一个 true pair (中心词及其上下文窗口中的词)与几个 noise pair (中心词与随机词搭配) 形成的样本，训练二元逻辑回归。

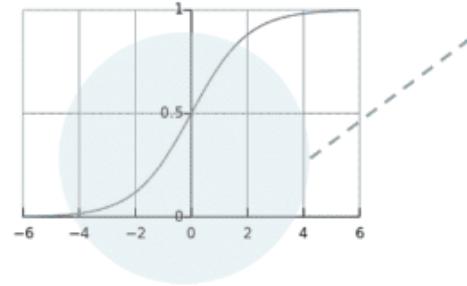
在Word2Vec模型的负采样中， J 表示特定于负采样策略的损失函数。这个损失函数衡量的是模型当前参数下，生成观察到的正样本（正确的单词上下文对）的概率，以及同时抑制模型生成一组随机选择的负样本（错误的单词上下文对）的概率。

The skip-gram model with negative sampling (HW2)

- From paper: “Distributed Representations of Words and Phrases and their Compositionality” (Mikolov et al. 2013)
- Overall objective function (they maximize): $J(\theta) = \frac{1}{T} \sum_{t=1}^T J_t(\theta)$

$$J_t(\theta) = \log \sigma(u_o^T v_c) + \sum_{i=1}^k \mathbb{E}_{j \sim P(w)} [\log \sigma(-u_j^T v_c)]$$

- The sigmoid function: $\sigma(x) = \frac{1}{1+e^{-x}}$ (we'll become good friends soon)
- So we maximize the probability of two words co-occurring in first log →



→ <http://www.showmeai.tech/>

本课以及作业中的目标函数是

$$J_{neg_sample}(\mathbf{o}, \mathbf{v}_c, \mathbf{U}) = -\log(\sigma(\mathbf{u}_o^\top \mathbf{v}_c)) - \sum_{k=1}^K \log(\sigma(-\mathbf{u}_k^\top \mathbf{v}_c))$$

- 我们取 k 个负例采样
- 最大化窗口中包围「中心词」的这些词语出现的概率，而最小化其他没有出现的随机词的概率

$$P(w) = U(w)^{3/4} / Z$$

- 我们用左侧的公式进行抽样，其中 $U(w)$ 是 unigram 分布
- 通过 $3/4$ 次方，相对减少常见单词的频率，增大稀有词的概率
- Z 用于生成概率分布

我们想让损失函数越小，我们需要第一项中的点积最大，接近1，这就导致第一项越小，也代表了上下文本相关性概率最大。我们希望第二项我们正确地预测负样本与当前词不相关时（即负样本的向量与上下文向量的点积应该是负数），代表着模型能识别不属于上下文的单词，所以点积是负数，加符号变成正数接近1，加对数变成接近0我们需要最大化目标函数

- 左侧为sigmoid函数(大家会在后续的内容里经常见到它)
- 我们要最大化2个词共现的概率

这个公式是在word2vec中使用负采样 (Negative Sampling) 时，用来选择负样本的概率分布。这里的($P(w)$)表示选择一个单词(w)作为负样本的概率。其中，($U(w)$)是单词(w)在训练语料中的不平衡(unnormalized)频率，即该单词出现的次数。Z是规范化常数 (normalization constant)，确保所有单词的($P(w)$)加起来等于1。

指数 $4/3$ 是一个经验值，它的作用是重新调整每个单词被选为负样本的概率，使得频率非常高的单词(如停用词)不会被过度采样，同时保证低频词有足够的机会被选中。这个调整能够改善训练的效果，因为它减少了常见词汇对负样本选择的影响，使得模型学习到的向量表示更加丰富和平衡。

co-occurrence matrix

我们能最简单想到的方法去联系中心词和上下文词就是，实现一个co-occurrence matrix矩阵，以此来计数，每个单词在上下文出现的概率，而不是通过算法去将word->vector

Example: Window based co-occurrence matrix

- Window length 1 (more common: 5–10)
- Symmetric (irrelevant whether left or right context)
- Example corpus:
 - I like deep learning
 - I like NLP
 - I enjoy flying

counts	I	like	enjoy	deep	learning	NLP	flying	.
I	0	2	1	0	0	0	0	0
like	2	0	0	1	0	1	0	0
enjoy	1	0	0	0	0	0	1	0
deep	0	1	0	0	1	0	0	0
learning	0	0	0	1	0	0	0	1
NLP	0	1	0	0	0	0	0	1
flying	0	0	1	0	0	0	0	1
.	0	0	0	0	1	1	1	0

这个就是窗口大小为1，也就是我们只统计前后1个单词，出现的概率，并形成所有单词作为行列的对称矩阵

co-occurrence vectors

svd中如何从matrix中提取vector

12

You

svd中我们如何把co occurrence 矩阵中提取一个单词的向量？

13

ChatGPT

在SVD（奇异值分解）用于词向量表示的情境中，通常是通过以下步骤来从共现矩阵中提取单词的向量：

1. **创建共现矩阵**: 首先，你需要创建一个共现矩阵。在这个矩阵中，每一行和每一列代表语料库中的一个单词，矩阵的每一个元素 X_{ij} 表示单词 i 和单词 j 在特定窗口大小内同时出现的次数。
2. **应用权重**: 有时，人们会对共现矩阵应用某种权重函数，如点信息增益（Pointwise Mutual Information, PMI），以减少高频单词的影响，并增强有意义的共现。
3. **进行奇异值分解**: 然后对这个矩阵进行SVD。SVD是一种分解方法，它将原始的共现矩阵 X 分解为三个矩阵的乘积 $U \cdot \Sigma \cdot V^T$ ，其中 U 和 V 是正交矩阵，它们的列向量称为左奇异向量和右奇异向量， Σ 是对角矩阵，其对角线上的元素称为奇异值。
4. **提取词向量**: 单词的向量可以通过取左奇异向量矩阵 U 中对该单词的行来获得。在某些情况下，人们可能会选择使用 $U \cdot \sqrt{\Sigma}$ 或者 $\sqrt{\Sigma} \cdot V^T$ 中的行来作为单词的向量，因为这样可以在向量中保留更多关于词频的信息。
5. **降维**: 最后，通常会选择保留前 k 个最大的奇异值以及与它们对应的奇异向量，这样做可以去除噪声并减少维度。这意味着矩阵 U 和 V 会被截断，只保留前 k 列。

举个例子，如果你有一个共现矩阵 X 并且对其进行SVD，得到 U , Σ , 和 V^T ，然后你想得到单词“apple”的向量，你会找到“apple”在共现矩阵的行索引 i ，然后取 U 中的第 i 行作为“apple”的词向量。如果你在降维到 k 维，那么你只取 U 的前 k 列的第 i 行。



那么对于这种co-occurrence vectors来说有一下问题

- Simple count co-occurrence vectors
 - Vectors increase in size with vocabulary
 - Very high dimensional: require a lot of storage (though sparse)
 - Subsequent classification models have sparsity issues → Models are less robust
- Low-dimensional vectors
 - Idea: store “most” of the important information in a fixed, small number of dimensions: a dense vector
 - Usually 25–1000 dimensions, similar to word2vec
 - How to reduce the dimensionality?

所以人们想到利用降维的方法。这样做可以去除噪声（去掉一些无关紧要的数据）并减少维度。

Classic Method: Dimensionality Reduction on X

降维并不是二维到一维，而是分割行或者列

Method 1: Dimensionality Reduction on X (HW1)

Singular Value Decomposition of co-occurrence matrix X

Factorizes X into $U\Sigma V^T$, where U and V are orthonormal

$$\begin{bmatrix} * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \end{bmatrix} = \begin{bmatrix} * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \end{bmatrix} \underbrace{\begin{bmatrix} \bullet & & & & \\ & \bullet & & & \\ & & \bullet & & \\ & & & \bullet & \\ & & & & \bullet \end{bmatrix}}_{\Sigma} \underbrace{\begin{bmatrix} * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \end{bmatrix}}_{V^T}$$

Retain only k singular values, in order to generalize.
 \hat{X} is the best rank k approximation to X , in terms of least squares.
 Classic linear algebra result. Expensive to compute for large matrices.

<http://www.showmeai.tech/>

方法一：对X进行降维（作业！）

使用SVD方法将共现矩阵 X 分解为 $U\Sigma V^T$

- Σ 是对角线矩阵，对角线上的值是矩阵的奇异值
- U, V 是对应于行和列的正交基

ShowMeAI

为了减少尺度同时尽量保存有效信息，可保留对角矩阵的最大的 k 个值，并将矩阵 U, V 的相应的行列保留

- 这是经典的线性代数算法，对于大型矩阵而言，计算代价昂贵

搜索 | 微信 ShowMeAI 研究中心

可以使用SVD方法将共现矩阵 分解为 $U\Sigma V^T$ ，其中：

- Σ 是对角线矩阵，对角线上的值是矩阵的奇异值（奇异值就是 AAT 的特征值开平方）
- U, V 是对应于行和列的正交基

为了减少尺度同时尽量保存有效信息，可保留对角矩阵的最大的 k 个值，并将矩阵 U, V 的相应的行列保留

Hacks to X

running an svd on raw counts doesn't work well

Hacks to X (several used in Rohde et al. 2005)

Scaling the counts in the cells can help *a lot*

- Problem: function words (*the, he, has*) are too frequent → syntax has too much impact. Some fixes:
 - $\min(X, t)$, with $t \approx 100$
 - Ignore them all
 - Ramped windows that count closer words more
 - Use Pearson correlations instead of counts, then set negative values to 0
 - Etc.

<http://www.showmeai.tech/>

#论文讲解#

按比例调整 counts 会很有效

- 对高频词进行缩放（语法有太多的影响）
 - 使用 \log 进行缩放
 - $\min(X, t), t \approx 100$
 - 直接全部忽视

ShowMeAI

- 在基于 window 的计数中，提高更加接近的单词的计数
- 使用 Pearson 相关系数
- Conclusion: 对计数进行处理是可以得到有效的词向量

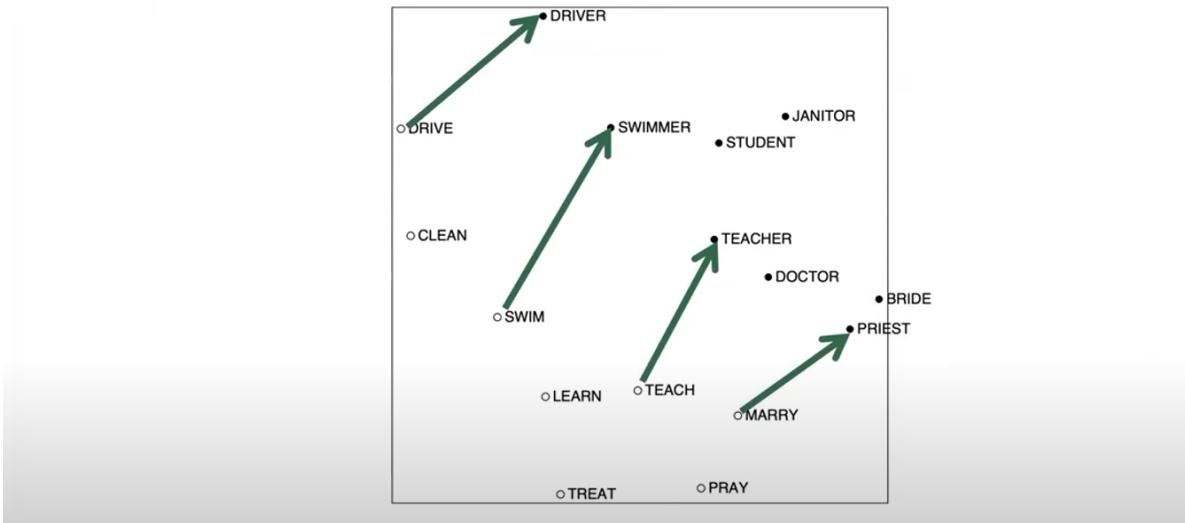
搜索 | 微信 ShowMeAI 研究中心

按比例调整 counts 会很有效

- 对高频词进行缩放（语法有太多的影响）like: a the an
 - 使用 \log 进行缩放
 - t 应该是单词出现频次的阈值
 - 忽略 function words
- 在基于 window 的计数中，提高更加接近的单词的计数
- 使用 Pearson 相关系数

Interesting semantic patterns emerge in the scaled vectors

Interesting semantic patterns emerge in the scaled vectors



verb to the person who does the verb, 这些向量大致平行

就好像drive to driver 就类似于 swim to swimmer

Count based vs. direct prediction

基于基数和基于预测的两种获得词向量的模型

Count based vs. direct prediction

- LSA, HAL (Lund & Burgess),
- COALS, Hellinger-PCA (Rohde et al, Lebret & Collobert)

- Fast training
- Efficient usage of statistics
- Primarily used to capture word similarity
- Disproportionate importance given to large counts

- Skip-gram/CBOW (Mikolov et al)
- NNLM, HLBL, RNN (Bengio et al; Collobert & Weston; Huang et al; Mnih & Hinton)

- Scales with corpus size
- Inefficient usage of statistics
- Generate improved performance on other tasks
- Can capture complex patterns beyond word similarity

基于计数：使用整个矩阵的全局统计数据来直接估计

- 优点：训练快速；统计数据高效利用
- 缺点：主要用于捕捉单词相似性；对大量数据给予比例失调的重视

基于预估模型：定义概率分布并试图预测单词

- 优点：提高其他任务的性能；能捕获除了单词相似性以外的复杂的模式

- 缺点：随语料库增大会增大规模；统计数据的低效使用（采样是对统计数据的低效使用）

Encoding meaning components in vector difference

meaning components 可以被代表为co-occurrence概率的比值

Encoding meaning components in vector differences

[Pennington, Socher, and Manning, EMNLP 2014]

Crucial insight: Ratios of co-occurrence probabilities can encode meaning components

	$x = \text{solid}$	$x = \text{gas}$	$x = \text{water}$	$x = \text{random}$
$P(x \text{ice})$	large	small	large	small
$P(x \text{steam})$	small	large	large	small
$\frac{P(x \text{ice})}{P(x \text{steam})}$	large	small	~ 1	~ 1

重点不是单一的概率大小，**重点是他们之间的比值**，其中蕴含着重要的信息成分。

- 例如我们想区分热力学上两种不同状态ice冰与蒸汽steam，它们之间的关系可通过与不同的单词 x 的共现概率的比值来描述
- 例如对于solid固态，虽然 $P(\text{solid} | \text{ice})$ 与 $P(\text{solid} | \text{steam})$ 本身很小，不能透露有效的信息，但是它们的比值 $\frac{P(\text{solid} | \text{ice})}{P(\text{solid} | \text{steam})}$ 却较大，因为solid更常用来描述ice的状态而不是steam的状态，所以在ice的上下文中出现几率较大
- 对于gas则恰恰相反，而对于water这种描述ice与steam均可或者fashion这种与两者都没什么联系的单词，则比值接近于 **1**。所以相较于单纯的共现概率，实际上共现概率的相对比值更有意义

Encoding meaning in vector differences

Q: How can we capture ratios of co-occurrence probabilities as linear meaning components in a word vector space?

A: Log-bilinear model: $w_i \cdot w_j = \log P(i|j)$

with vector differences $w_x \cdot (w_a - w_b) = \log \frac{P(x|a)}{P(x|b)}$

Glove combining the best of both worlds



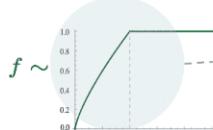
Combining the best of both worlds
GloVe [Pennington et al., EMNLP 2014]



$$w_i \cdot w_j = \log P(i|j)$$

$$J = \sum_{i,j=1}^V f(X_{ij}) (w_i^T \tilde{w}_j + b_i + \tilde{b}_j - \log X_{ij})^2$$

- Fast training
- Scalable to huge corpora
- Good performance even with small corpus and small vectors



<http://www.showmeai.tech/>

#论文讲解#

补充讲解

- 如果使向量点积等于共现概率的对数，那么向量差异变成了共现概率的比率
- 使用平方误差促使点积尽可能得接近共现概率的对数
- 使用 $f(x)$ 对常见单词进行限制

ShowMeAI

优点

- 训练快速
- 可以扩展到大型语料库
- 即使是小语料库和小向量，性能也很好

搜索 | 微信 ShowMeAI 研究中心

- $f(X_{ij})$ 是一个函数，应用于 X_{ij} 上， X_{ij} 可能表示某种类型的输入数据，比如单词对的共现频率。
- \vec{w}_i 和 \vec{w}_j 是向量，可能代表词 i 和词 j 在嵌入空间中的位置。
- $\vec{w}_i^T \vec{w}_j$ 是向量 \vec{w}_i 和 \vec{w}_j 的点积，通常用于计算两个向量之间的相似度或关联度。
- b_i 和 b_j 是偏差项，用于每个词向量模型中。
- $\log X_{ij}$ 可能代表了 X_{ij} 的自然对数，这在处理概率或信息论中很常见。
- 整个 $(\vec{w}_i^T \vec{w}_j + b_i + b_j - \log X_{ij})^2$ 表达式是实际预测（通过点积和偏差项计算得出）和目标值 $(\log X_{ij})$ 之间差异的平方，这种差异的平方通常用于计算平方误差。

logxij代表着单词两个单词出现的次数的比值（这里不是单词对的比值，这是两个单词的比值，应该和之前单词对差不多，只是消掉了分母）

How to evaluate word vectors

How to evaluate word vectors?

- Related to general evaluation in NLP: Intrinsic vs extrinsic
- Intrinsic:
 - Evaluation on a specific/intermediate subtask
 - Fast to compute
 - Helps to understand that system
 - Not clear if really helpful unless correlation to real task is established
- Extrinsic:
 - Evaluation on a real task
 - Can take a long time to compute accuracy
 - Unclear if the subsystem is the problem or its interaction or other subsystems
 - If replacing exactly one subsystem with another improves accuracy → Winning!

<http://www.showmeai.tech/>

如何评估词向量？

与NLP的一般评估相关: 内在与外在

- 内在
 - 对特定/中间子任务进行评估
 - 计算速度快
 - 有助于理解这个系统
 - 不清楚是否真的有用，除非与实际任务建立了相关性

ShowMeAI

- 外在
 - 对真实任务的评估
 - 计算精确度可能需要很长时间
 - 不清楚子系统问题所在，是交互还是其他子系统问题
 - 如果用另一个子系统替换一个子系统可以提高精确度

Intrinsic word vector evaluation

Intrinsic word vector evaluation

- Word Vector Analogies

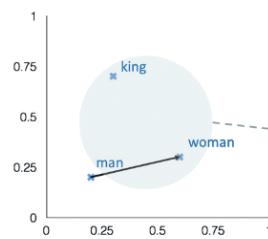
a:b :: c:?

man:woman :: king:?



$$d = \arg \max_i \frac{(x_b - x_a + x_c)^T x_i}{\|x_b - x_a + x_c\|}$$

- Evaluate word vectors by how well their cosine distance after addition captures intuitive semantic and syntactic analogy questions
- Discarding the input words from the search!
- Problem: What if the information is there but not linear?



<http://www.showmeai.tech/>

内在词向量评估

- 词向量类比

$a:b :: c: ? \rightarrow d = \arg \max_i \frac{(x_b - x_a + x_c)^T x_i}{\|x_b - x_a + x_c\|}$

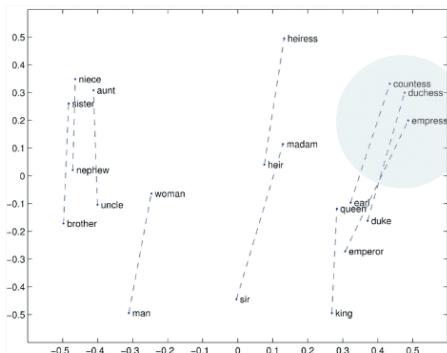
ShowMeAI

- 通过加法后的余弦距离是否能很好地捕捉到直观的语义和句法类比问题来评估单词向量
- 从搜索中丢弃输入的单词
- 问题：如果存在信息关联但不是线性的怎么办？

搜索 | 微信 ShowMeAI 研究中心

Glove Visualizations

Glove Visualizations



<http://www.showmeai.tech/>

Glove可视化效果

ShowMeAI

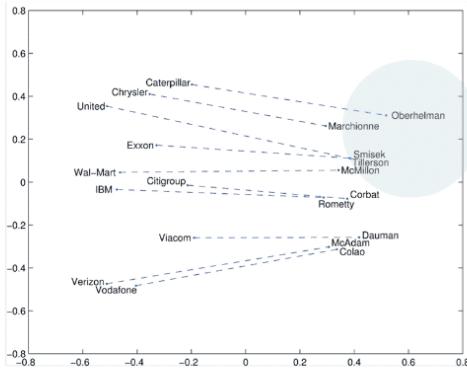
补充讲解

- 通过GloVe得到的词向量，我们对词向量进行减法计算，可以发现类比的词对有相似的距离。
- brother - sister, man - woman, king - queen

ShowMeAI

搜索 | 微信 ShowMeAI 研究中心

Glove Visualizations: Company - CEO



<http://www.showmeai.tech/>

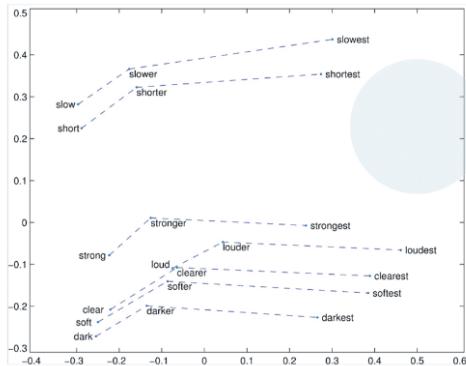
Glove可视化效果

ShowMeAI

- 公司与CEO词汇

搜索 | 微信 ShowMeAI 研究中心

Glove Visualizations: Superlatives



http://www.showmeai.tech/

Glove可视化效果



ShowMeAI
• 词汇比较级与最高级

ShowMeAI

搜索 | 微信 ShowMeAI 研究中心

Analogy evaluation and hyperparameters

Analogy evaluation and hyperparameters

Glove word vectors evaluation

Model	Dim.	Size	Sem.	Syn.	Tot.
SVD	300	6B	6.3	8.1	7.3
SVD-S	300	6B	36.7	46.6	42.1
SVD-L	300	6B	56.6	63.0	60.1
CBOW [†]	300	6B	63.6	<u>67.4</u>	65.7
SG [†]	300	6B	73.0	66.0	69.1
GloVe	300	6B	<u>77.4</u>	67.0	<u>71.7</u>

- **Sem.:** 表示在语义测试中的表现。这通常涉及到词义的相似性测试，如比较单词对是否在语义上接近。
- **Syn.:** 表示在语法测试中的表现。这涉及到词语的语法属性，比如词性标注或句法依赖关系。
- **Tot.:** 表示总体表现，可能是语义 (Sem.) 和语法 (Syn.) 分数的平均或加权总和

问题

这个随机的矩阵 是谁规定的 每个词对应一个向量？

在训练神经网络，特别是进行词嵌入训练时，随机初始化嵌入矩阵是一个常见的做法。这个过程并不是由某个人规定的，而是一种标准的机器学习技术，通常在模型的实现代码中自动完成。

随机初始化的目的是为了打破对称性，确保模型训练时可以正常学习。如果所有的向量初始化为同样的值，那么模型在训练过程中可能无法学习到有用的特征，因为每个单词的表示将会以同样的方式更新，从而无法分辨它们。

在训练开始时，每个单词被随机分配一个向量，这个向量的维度是提前设定的。例如，我们可能选择使用100维或300维的向量来表示每个单词。向量的具体数值是随机生成的，但通常会遵循一定的统计分布（如均匀分布或正态分布），且数值通常很小。

随着训练的进行，这些向量会根据模型的学习目标（如预测上下文单词）逐渐调整，以至于最终反映出单词的语义和语法特性。这样，模型最终得到的词向量就不再是随机的了，而是有意义的，能捕捉到单词之间的关系，如它们的相似度和共现频率等。

skip-gram 和glove的区别

Skip-gram 和 GloVe 都是用于学习词嵌入的模型，但它们在原理和实现上有一些关键的区别：

1. 模型原理：

- **Skip-gram (Word2Vec 的一种架构)**：基于局部上下文信息。给定一个中心词，模型的目标是预测它在文本中周围的词语。Skip-gram 通过优化这种局部上下文预测来学习词嵌入。
- **GloVe**：基于全局统计信息。GloVe 使用整个语料库中的词共现（co-occurrence）统计信息来学习词嵌入。它构建一个全局共现矩阵，并通过矩阵分解来得到词嵌入。

2. 优化目标：

- **Skip-gram**：优化的是局部上下文预测的对数似然函数。
- **GloVe**：优化的是词对之间的共现概率的对数与词嵌入内积之间的差异的平方和。

3. 处理大规模数据：

- **Skip-gram**：通常需要对大量的局部上下文窗口进行采样和训练，可能在处理非常大的语料库时效率较低。
- **GloVe**：由于是基于全局统计信息，可以在构建共现矩阵时一次性处理整个语料库，可能在处理大规模数据时更加高效。

4. 捕捉语义关系：

- **Skip-gram**：能够捕捉词语之间的相似性和语义关系，特别擅长捕捉罕见词的表示。
- **GloVe**：通过全局共现统计，能够捕捉更丰富的语义关系，特别是能够较好地保持词与词之间的线性关系（如“king” - “man” + “woman” ≈ “queen”）。

GloVe 用于什么时候： GloVe 适用于需要从全局统计信息中学习词嵌入的场景，特别是当你希望捕捉词与词之间的共现关系并保持词嵌入中的线性语义关系时。GloVe 在处理大规模语料库时也可能更高效，因为它避免了对大量局部上下文的重复采样和训练。

总的来说，Skip-gram 和 GloVe 都是强大的词嵌入学习模型，选择使用哪个模型取决于具体的任务需求和语料库特点。在实践中，尝试不同的模型并比较它们在特定任务上的表现是一个常见的做法。

Skip-gram的整体过程

Word2Vec 的 Skip-gram 模型整体过程包括以下步骤：

1. 初始化：

- 随机初始化词向量，为每个单词生成两个向量：中心词向量和外部词（上下文）向量。

2. 训练循环：

- 对于语料库中的每个单词，将其作为中心词，然后根据窗口大小选取周围的上下文词。
- 对于每个中心词和上下文词对，执行以下步骤：

3. 前向传播：

- 通过计算中心词向量和每个上下文词向量的点积得到一个分数。

- 应用 softmax 函数将分数转换成概率分布。

4. 计算损失:

- 使用交叉熵损失函数来计算当前的预测概率分布与实际的一热编码分布之间的差异。

5. 反向传播 (梯度计算) :

- 通过损失函数相对于词向量的梯度来更新中心词向量和上下文词向量。

6. 参数更新:

- 使用梯度下降 (或其他优化算法) 更新词向量。这通常包括计算学习率和梯度的乘积，并从当前向量中减去该乘积。

7. 重复步骤:

- 重复步骤2到6，直到达到一定的迭代次数，或者模型损失不再显著减少。

8. 后处理 (可选) :

- 训练完成后，可能会对词向量进行归一化处理。
- 有时还会使用技巧如词向量维数的降维 (例如使用 PCA) 以提高其实用性。

lecture3 Backprop and Neural Networks (向后传播和神经网络)

1.named entity recognition (NER) 命名实体识别

Named Entity Recognition (NER)

- The task: **find** and **classify** names in text, for example:

Last night , Paris Hilton wowed in a sequin gown .

PER PER

Samuel Quinn was arrested in the Hilton Hotel in Paris in April 1989 .

PER PER LOC LOC LOC DATE DATE

- Possible uses:

- Tracking mentions of particular entities in documents
- For question answering, answers are usually named entities
- Often followed by Named Entity Linking/Canonicalization into Knowledge Base

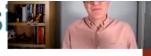
给文字添加标签，例如per人，loc位置

- 可能的用途

- 跟踪文档中提到的特定实体(组织、个人、地点、歌曲名、电影名等)
- 对于问题回答，答案通常是命名实体
- 许多需要的信息实际上是命名实体之间的关联
- 同样的技术可以扩展到其他 slot-filling 槽填充分类

- 通常后面是命名实体链接/规范化到知识库

Simple NER: Window classification using binary logistic class



- **Idea:** classify each word in its context window of neighboring words
- Train logistic classifier on hand-labeled data to classify center word {yes/no} for each class based on a concatenation of word vectors in a window
 - Really, we usually use multi-class softmax, but trying to keep it simple 😊
- **Example:** Classify “Paris” as +/– location in context of sentence with window length 2:

the museums in Paris are amazing to see .

$$X_{\text{window}} = [x_{\text{museums}} \quad x_{\text{in}} \quad x_{\text{Paris}} \quad x_{\text{are}} \quad x_{\text{amazing}}]^T$$

- Resulting vector $x_{\text{window}} = x \in \mathbb{R}^{5d}$, a column vector!
- To classify all words: run classifier for each class on the vector centered on each word in the sentence



1. 基本思想：

- 为每个单词分类，根据其邻近词（上下文窗口）来确定该单词是否属于某个特定类别。

2. 训练分类器：

- 在手标注数据上训练逻辑回归分类器，以确定中心词是否属于某个类别（是或否）。虽然通常会使用多类 softmax 分类器，但为了简化，这里使用了二元分类器。

3. 举例：

- 分类“Paris”（巴黎）这个单词，判断它在句子中是否是一个位置（地点名称）。这里使用的窗口长度是 2，即考虑中心词前后各两个词作为上下文。
- 给出的例子句子是“The museums in Paris are amazing to see.”（博物馆在巴黎是惊人的看点）。在这个句子中，“Paris”前后各有两个词，形成了一个上下文窗口。

4. 特征向量构建：

- 构建**特征向量** x_{window} 通过拼接（concatenation）中心词“Paris”和它的上下文词“museums”，“in”，“are”，“amazing”的词向量。这里 x_{museums} 、 x_{in} 、 x_{Paris} 、 x_{are} 、 x_{amazing} 分别是这些词的词向量。
- 拼接后的结果向量 x_{window} 属于 \mathbb{R}^{5d} 空间，是一个列向量，其中 d 是单个词向量的维度。

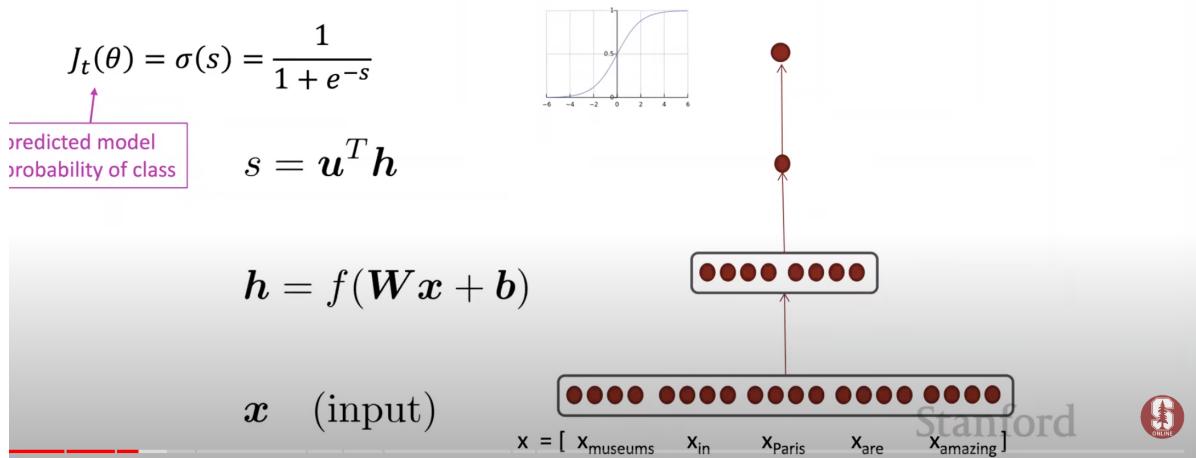
5. 分类操作：

- 为了对整个句子中的每个单词进行分类，针对每个单词中心的向量运行分类器，判断其所属类别。

NER: Binary classification for center word being location



- We do supervised training and want high score if it's a location



我们现在有了输入的特征向量 X , 其中由五个词向量组成

- 隐藏层计算方式: 通过特征矩阵W和偏置项 (偏置项允许模型输出在没有输入 (即输入为零) 时不等于零, 从而更灵活地拟合数据。例如, 在二维空间中, 偏置项允许直线不必经过原点。这提供了额外的自由度, 让模型能够更好地适应数据。) b 和非线性激活函数得到 h
- 得分计算公式: u 是权重向量, h 是隐藏层表示 (文中叫Neural Networks, 虽然这是机器学习, 但是他利用的是神经网络提取特征值的方法进行计算)
- costfunction, 将得分 s 映射到 $(0, 1)$ 内表示概率

SGD

Remember: Stochastic Gradient Descent



Update equation:

$$\theta^{new} = \theta^{old} - \alpha \nabla_{\theta} J(\theta)$$

$\alpha = \text{step size or learning rate}$

i.e., for each parameter: $\theta_j^{new} = \theta_j^{old} - \alpha \frac{\partial J(\theta)}{\partial \theta_j^{old}}$

In deep learning, we update the data representation (e.g., word vectors) too!

How can we compute $\nabla_{\theta} J(\theta)$?

1. By hand
2. Algorithmically: the backpropagation algorithm

Stanford

gradients 梯度概念

Gradients

- Given a function with 1 output and 1 input

$$f(x) = x^3$$

- Its gradient (slope) is its derivative

$$\frac{df}{dx} = 3x^2$$

“How much will the output change if we change the input a bit?”

At $x = 1$ it changes about 3 times as much: $1.01^3 = 1.03$

At $x = 4$ it changes about 48 times as much: $4.01^3 = 64.48$

Gradients

- Given a function with 1 output and n inputs

$$f(\mathbf{x}) = f(x_1, x_2, \dots, x_n)$$

- Its gradient is a vector of partial derivatives with respect to each input

$$\frac{\partial f}{\partial \mathbf{x}} = \left[\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \dots, \frac{\partial f}{\partial x_n} \right]$$

梯度是一个向量，包含了 f 对每个 x_i 的偏导数 ∇f

梯度向量中的每个分量是衡量函数 f 在相应变量 x_i 方向上的变化率

所以表示为下面这个式子

Jacobian Matrix: Generalization of the Gradient

- Given a function with m outputs and n inputs

$$\mathbf{f}(\mathbf{x}) = [f_1(x_1, x_2, \dots, x_n), \dots, f_m(x_1, x_2, \dots, x_n)]$$

- It's Jacobian is an $m \times n$ matrix of partial derivatives

$$\frac{\partial \mathbf{f}}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}$$
$$\left(\frac{\partial \mathbf{f}}{\partial \mathbf{x}} \right)_{ij} = \frac{\partial f_i}{\partial x_j}$$

雅可比矩阵 (Jacobian matrix)



Example Jacobian: Elementwise activation Function



$$\mathbf{h} = f(\mathbf{z}), \text{ what is } \frac{\partial \mathbf{h}}{\partial \mathbf{z}}? \quad \mathbf{h}, \mathbf{z} \in \mathbb{R}^n$$
$$h_i = f(z_i)$$

$$\left(\frac{\partial \mathbf{h}}{\partial \mathbf{z}} \right)_{ij} = \frac{\partial h_i}{\partial z_j} = \frac{\partial}{\partial z_j} f(z_i) \quad \text{definition of Jacobian}$$
$$= \begin{cases} f'(z_i) & \text{if } i = j \\ 0 & \text{if otherwise} \end{cases} \quad \text{regular 1-variable derivative}$$

$$\frac{\partial \mathbf{h}}{\partial \mathbf{z}} = \begin{pmatrix} f'(z_1) & & 0 \\ & \ddots & \\ 0 & & f'(z_n) \end{pmatrix} = \text{diag}(f'(\mathbf{z}))$$

这里是 n 个input n 个output的函数

那么对这个函数求偏导，只有 $i=j$ 才对函数有影响

所以这个函数的jacobian矩阵只有在对角线上有元素，可以类比 n input m output 的jacobian矩阵去理解

这个矩阵就被表示成 $\text{diag}(f'(\mathbf{z}))$ ，表示一个对角线上有 f' 导数，其他元素为 0 的对角矩阵。

Other Jacobians

$$\frac{\partial}{\partial \mathbf{x}}(\mathbf{Wx} + \mathbf{b}) = \mathbf{W}$$

$$\frac{\partial}{\partial \mathbf{b}}(\mathbf{Wx} + \mathbf{b}) = \mathbf{I} \text{ (Identity matrix)}$$

$$\frac{\partial}{\partial \mathbf{u}}(\mathbf{u}^T \mathbf{h}) = \mathbf{h}^T$$

$$\frac{\partial}{\partial \mathbf{u}}(\mathbf{u}^T \mathbf{h}) = \begin{bmatrix} \frac{\partial}{\partial u_1}(\mathbf{u}^T \mathbf{h}) \\ \frac{\partial}{\partial u_2}(\mathbf{u}^T \mathbf{h}) \\ \frac{\partial}{\partial u_3}(\mathbf{u}^T \mathbf{h}) \end{bmatrix}$$

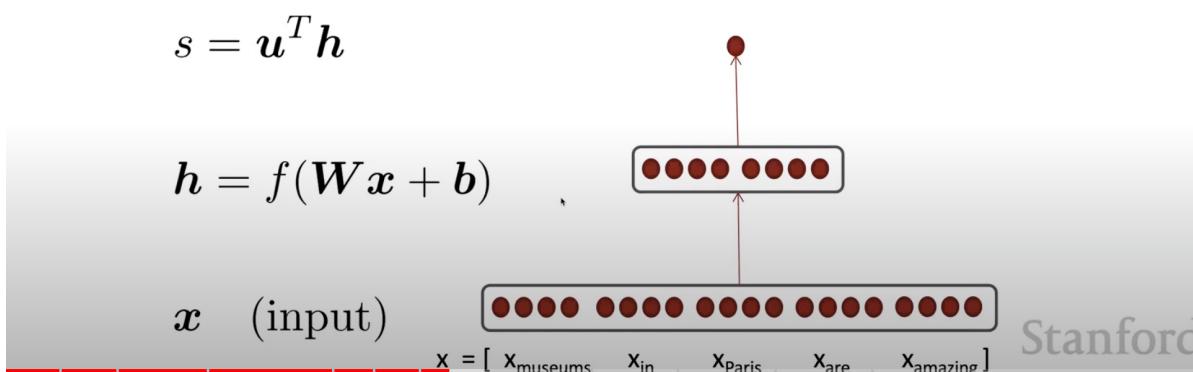
形如 $\mathbf{u}^T \mathbf{h}$ 也是向量，所以最终形式也要进行转置

Back to our Neural Net

Back to our Neural Net!



- Let's find $\frac{\partial s}{\partial b}$
 - Really, we care about the gradient of the loss J_t but we will compute the gradient of the score for simplicity



虽然我们对cost function J 感兴趣，但是这里为了简化，我们就求score 对b的偏导数

之前我们求 $wx+b$ 对 b 偏导，矩阵是单位矩阵，因为 b 和 x 相对独立。但是这里并不独立，所以需要通过链式法则来求解。

1. Break up equations into simple pieces



$$s = \mathbf{u}^T \mathbf{h}$$

$$s = \mathbf{u}^T \mathbf{h}$$

$$\mathbf{h} = f(\mathbf{W}\mathbf{x} + \mathbf{b}) \quad \rightarrow \quad \mathbf{h} = f(\mathbf{z})$$
$$\mathbf{z} = \mathbf{W}\mathbf{x} + \mathbf{b}$$

\mathbf{x} (input)

\mathbf{x} (input)

Carefully define your variables and keep track of their dimensionality! 

进一步分解链式法则

2. Apply the chain rule

$$s = \mathbf{u}^T \mathbf{h}$$
$$\mathbf{h} = f(\mathbf{z})$$
$$\mathbf{z} = \mathbf{W}\mathbf{x} + \mathbf{b}$$
$$\mathbf{x}$$
 (input)

$$\frac{\partial s}{\partial \mathbf{b}} = \frac{\partial s}{\partial \mathbf{h}} \frac{\partial \mathbf{h}}{\partial \mathbf{z}} \frac{\partial \mathbf{z}}{\partial \mathbf{b}}$$

3. Write out the Jacobians

$$\begin{aligned}
 s &= \mathbf{u}^T \mathbf{h} \\
 \mathbf{h} &= f(\mathbf{z}) \\
 \mathbf{z} &= \mathbf{W}\mathbf{x} + \mathbf{b} \\
 \mathbf{x} &\quad (\text{input})
 \end{aligned}
 \quad
 \begin{aligned}
 \frac{\partial s}{\partial \mathbf{b}} &= \frac{\partial s}{\partial \mathbf{h}} \frac{\partial \mathbf{h}}{\partial \mathbf{z}} \frac{\partial \mathbf{z}}{\partial \mathbf{b}} \\
 &\quad \downarrow \quad \downarrow \quad \downarrow \\
 &= \mathbf{u}^T \text{diag}(f'(\mathbf{z})) \mathbf{I} \\
 &= \mathbf{u}^T \circ f'(\mathbf{z})
 \end{aligned}$$

Useful Jacobians from previous slide

$$\frac{\partial}{\partial \mathbf{u}}(\mathbf{u}^T \mathbf{h}) = \mathbf{h}^T$$

$$\frac{\partial}{\partial \mathbf{z}}(f(\mathbf{z})) = \text{diag}(f'(\mathbf{z}))$$

$$\frac{\partial}{\partial \mathbf{b}}(\mathbf{W}\mathbf{x} + \mathbf{b}) = \mathbf{I}$$

Stanfo

22

这个小圆点 (◦) 表示的是哈达玛积 (Hadamard product)，也称作逐元素乘法。这是一个二元操作，它接受两个尺寸相同的矩阵，并产生另一个同样尺寸的矩阵，其各个元素是原两个矩阵对应元素的乘积。

Re-using Computation

- Suppose we now want to compute $\frac{\partial s}{\partial \mathbf{W}}$
 - Using the chain rule again:

$$\begin{aligned}
 \frac{\partial s}{\partial \mathbf{W}} &= \frac{\partial s}{\partial \mathbf{h}} \frac{\partial \mathbf{h}}{\partial \mathbf{z}} \frac{\partial \mathbf{z}}{\partial \mathbf{W}} \\
 \frac{\partial s}{\partial \mathbf{b}} &= \frac{\partial s}{\partial \mathbf{h}} \frac{\partial \mathbf{h}}{\partial \mathbf{z}} \frac{\partial \mathbf{z}}{\partial \mathbf{b}}
 \end{aligned}$$

The same! Let's avoid duplicated computation ...

我们发现s对W偏导的时候，前两项是完全相同的，所以我们有必要使用一种减少重复运算的算法

Re-using Computation

- Suppose we now want to compute $\frac{\partial s}{\partial W}$
 - Using the chain rule again:

$$\begin{aligned}\frac{\partial s}{\partial W} &= \delta \frac{\partial z}{\partial W} \\ \frac{\partial s}{\partial b} &= \delta \frac{\partial z}{\partial b} = \delta \\ \delta &= \frac{\partial s}{\partial h} \frac{\partial h}{\partial z} = u^T \circ f'(z)\end{aligned}$$

δ is the local error signal

我们把前两项替换为 δ (局部误差信号)

Derivative with respect to Matrix: Output shape



- What does $\frac{\partial s}{\partial W}$ look like? $W \in \mathbb{R}^{n \times m}$
- 1 output, nm inputs: 1 by nm Jacobian?
 - Inconvenient to then do $\theta^{new} = \theta^{old} - \alpha \nabla_{\theta} J(\theta)$
- Instead, we leave pure math and use the **shape convention**: the shape of the gradient is the shape of the parameters!

So $\frac{\partial s}{\partial W}$ is n by m :

$$\begin{bmatrix} \frac{\partial s}{\partial W_{11}} & \cdots & \frac{\partial s}{\partial W_{1m}} \\ \vdots & \ddots & \vdots \\ \frac{\partial s}{\partial W_{n1}} & \cdots & \frac{\partial s}{\partial W_{nm}} \end{bmatrix}$$

- 参数形状和jacobian矩阵形状不同，w有nxm个输入，1个输出，所以jacobian矩阵式一个1xnxm的形状（由于这是二元分类问题所以output只有一个值0/1）
- 梯度形状的重要性：我们通常希望梯度的形状与参数矩阵 W 的形状相匹配。这样做可以使更新参数更加直观和容易实现，尤其是在编写代码来更新神经网络参数时。
- 遵循**形状约定** (shape convention)：在数学上，尽管可以计算出不同形状的梯度（例如，标量对矩阵的雅可比矩阵），但在实践中，我们遵循一个“形状约定”，即梯度的形状应该与参数的形状相同。这意味着如果 W 是一个 $n \times m$ 矩阵，那么 s 对 w 的偏导也是一个 $n \times m$ 的矩阵。
- jacobian矩阵梯度元素： W_{ij} 是 W 矩阵中第*i*行第*j*列的元素，这种表示方法展示了如何计算每个权重对输出s的影响

Derivative with respect to Matrix

- What is $\frac{\partial s}{\partial \mathbf{W}} = \delta \frac{\partial z}{\partial \mathbf{W}}$
 - δ is going to be in our answer
 - The other term should be \mathbf{x} because $\mathbf{z} = \mathbf{W}\mathbf{x} + \mathbf{b}$
- Answer is: $\frac{\partial s}{\partial \mathbf{W}} = \delta^T \mathbf{x}^T$

δ is local error signal at z
 x is local input signal

Deriving local input gradient in backprop



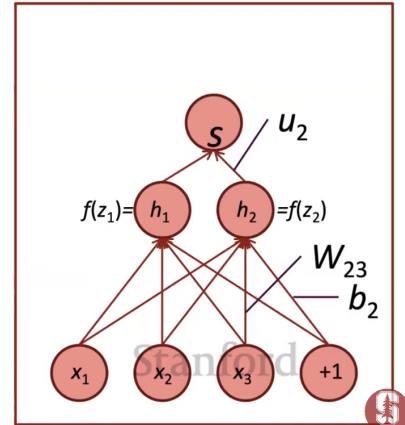
- For $\frac{\partial z}{\partial \mathbf{W}}$ in our equation:

$$\frac{\partial s}{\partial \mathbf{W}} = \delta \frac{\partial \mathbf{z}}{\partial \mathbf{W}} = \delta \frac{\partial}{\partial \mathbf{W}} (\mathbf{W}\mathbf{x} + \mathbf{b})$$

- Let's consider the derivative of a single weight W_{ij}
- W_{ij} only contributes to z_i
 - For example: W_{23} is only used to compute z_2 not z_1

$$\begin{aligned} \frac{\partial z_i}{\partial W_{ij}} &= \frac{\partial}{\partial W_{ij}} \mathbf{W}_i \cdot \mathbf{x} + b_i \\ &= \frac{\partial}{\partial W_{ij}} \sum_{k=1}^d W_{ik} x_k = x_j \end{aligned}$$

40



Why the Transposes?



$$\frac{\partial s}{\partial \mathbf{W}} = \delta^T \mathbf{x}^T$$

$$[n \times m] \quad [n \times 1][1 \times m]$$

$$= \begin{bmatrix} \delta_1 \\ \vdots \\ \delta_n \end{bmatrix} [x_1, \dots, x_m] = \begin{bmatrix} \delta_1 x_1 & \dots & \delta_1 x_m \\ \vdots & \ddots & \vdots \\ \delta_n x_1 & \dots & \delta_n x_m \end{bmatrix}$$

- Hacky answer: this makes the dimensions work out!
 - Useful trick for checking your work!
- Full explanation in the lecture notes
 - Each input goes to each output – you want to get outer product

Stanford

What shape should derivatives be?



- Similarly, $\frac{\partial s}{\partial b} = h^T \circ f'(z)$ is a row vector
 - But shape convention says our gradient should be a column vector because b is a column vector ...
- Disagreement between Jacobian form (which makes the chain rule easy) and the shape convention (which makes implementing SGD easy)
 - We expect answers in the assignment to follow the **shape convention**
 - But Jacobian form is useful for computing the answers

然而，按照形状惯例，因为 b 是一个列向量，我们的梯度应该也是一个列向量。这样的形状惯例可以让数学表达式在形式上保持一致性，特别是在编写代码实现算法时，这能使得矩阵运算更加直观和一致。

在机器学习和线性代数中，通常将变量（特别是在优化问题中的参数）表示为列向量。这是因为在矩阵乘法中，一个矩阵 A 乘以一个列向量 b ，结果是另一个列向量，计算过程遵循了线性映射的规则。这个习惯使得表达式和计算在形式上保持一致。

What shape should derivatives be?

Two options:

1. Use Jacobian form as much as possible, reshape to follow the shape convention at the end:
 - What we just did. But at the end transpose $\frac{\partial s}{\partial b}$ to make the derivative a column vector, resulting in δ^T
2. Always follow the shape convention
 - Look at dimensions to figure out when to transpose and/or reorder terms
 - The error message δ that arrives at a hidden layer has the same dimensionality as that hidden layer

1. 尽可能使用雅各比形式 (Jacobian form)，但在计算的最后阶段，根据形状约定进行转换：这意味着在计算过程中可以使用行向量，但在最终结果时，需要将导数 $\partial s / \partial b$ 转置成列向量，得到 δ^T 。
2. 始终遵循形状约定：这建议在进行导数计算时始终保持形状一致性。在操作中需要查看维度，以确定何时需要转置或重新排序项，确保每个计算步骤的输出与期望的形状一致。

幻灯片还指出，错误信息 δ ，即反向传播中到达隐藏层的误差梯度，应该与该隐藏层的维度一致。这是为了确保在神经网络中反向传播算法可以正确执行，因为每一层的权重更新依赖于与该层输出相同维度的梯度。

Backpropagation 反向传播

反向传播（Backpropagation）是神经网络训练中的一种算法，用于根据损失函数（表示网络预测错误程度的函数）来调整网络中的权重和偏置。**其核心目标是最小化损失函数的值，即减少神经网络输出与实际值之间的差异。**

1. 前向传播（Forward Propagation）：

- 输入数据在网络中前向传播，通过层间的权重和偏置，以及非线性激活函数，直到得到输出。
- 网络的输出与真实的标签值比较，计算损失函数的值。

2. 计算梯度（Gradient Calculation）：

- 损失函数相对于网络最后一层输出的梯度首先被计算出来。
- 这个梯度反映了损失函数对网络输出的敏感度。

3. 反向传播（Backpropagation）：

- 通过链式法则，损失函数的梯度被反向传播回网络中，计算中间层输出和最终损失之间的关系。
- 在每一层，计算损失函数对该层每个权重的偏导数，这些偏导数构成了梯度。
- 这一步是反向传播算法的核心，它允许我们理解如何通过调整每个权重来减少最终的损失。

4. 权重更新（Weight Update）：

- 一旦计算出损失函数对所有权重的梯度，就可以使用这些梯度来更新权重。
- 更新通常是通过梯度下降或其变种进行，权重按照其对损失的贡献程度进行调整，以期望在下一次前向传播时降低损失。

3. Backpropagation



We've almost shown you backpropagation

It's taking derivatives and using the (generalized, multivariate, or matrix) chain rule

Other trick:

We **re-use** derivatives computed for higher layers in computing derivatives for lower layers to minimize computation

优化技巧：幻灯片上的“other trick”提到了一个重要的优化技巧——在计算时重复利用上层计算出的导数来计算下层的导数，以减少计算量。这意味着在神经网络的某一层计算出的导数可以在计算之前的层时重复使用，这样可以避免重复的计算工作，使得算法更加高效。

Computation Graphs and Backpropagation

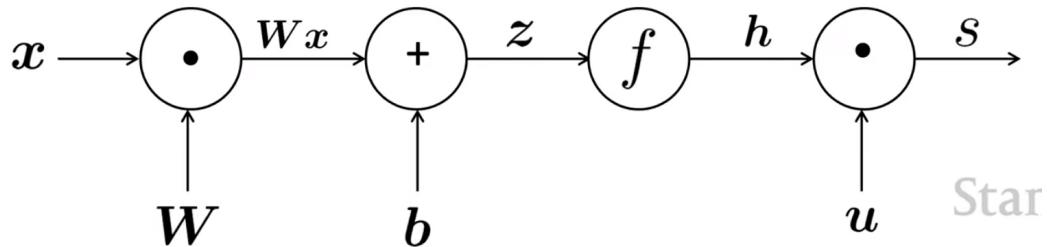
- Software represents our neural net equations as a graph

$$s = \mathbf{u}^T \mathbf{h}$$

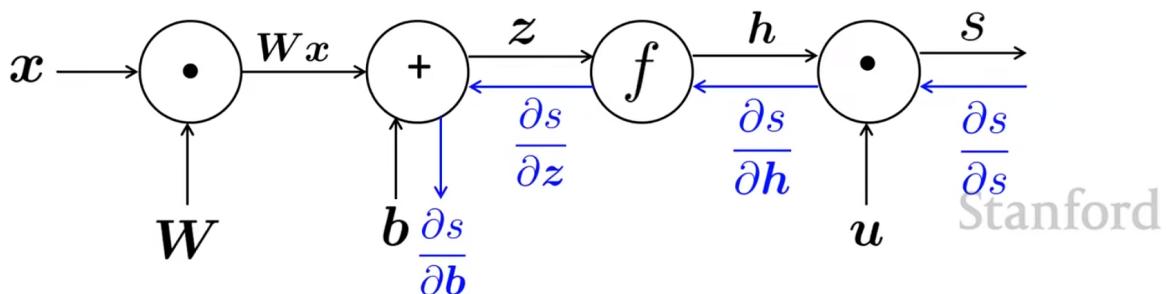
$$\mathbf{h} = f(\mathbf{z})$$

$$\mathbf{z} = \mathbf{W}\mathbf{x} + \mathbf{b}$$

$$\mathbf{x} \quad (\text{input})$$
- Source nodes: inputs
- Interior nodes: operations
- Edges pass along result of the operation



用计算图来代表神经网络中如何进行向前传播



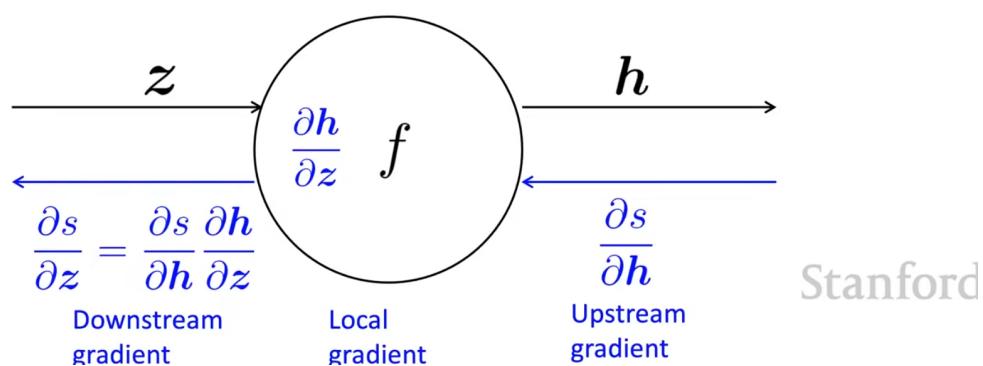
向后传播更新权重

Backpropagation: Single Node



- Each node has a **local gradient**
- The gradient of its output with respect to its input
- [downstream gradient] = [upstream gradient] x [local gradient]

$$h = f(z)$$



局部梯度就是这个节点输出对输入的偏导数

上游梯度就是损失函数对当前节点输出的偏导数

下游梯度就是损失函数对于当前节点输入的偏导数，是局部梯度和上有梯度的乘积

这三个概念都是相对于节点而言，这里为了简化只有一个节点

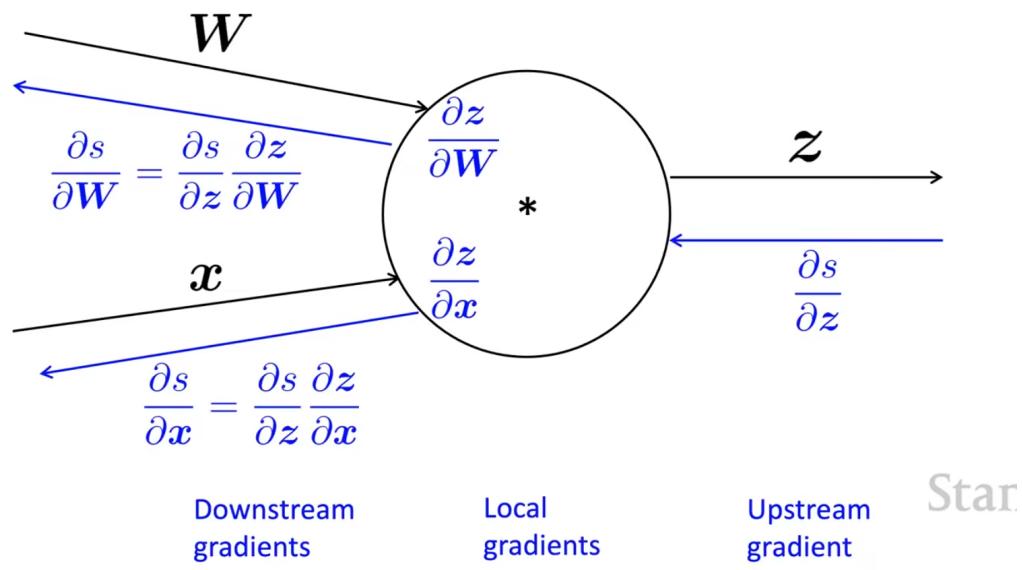
我们所谓的梯度是针对整个神经网络

在反向传播中，下游梯度的计算对于更新网络中的权重和偏置至关重要。每个节点的下游梯度都是基于其上游梯度和当前节点操作的局部梯度。通过这种方式，梯度信息可以从输出层回传到输入层，允许我们对每个参数的影响进行量化并相应地更新它们以减少损失。

Backpropagation: Single Node

- Multiple inputs → multiple local gradients

$$z = Wx$$



An Example

$$f(x, y, z) = (x + y) \max(y, z)$$

$$x = 1, y = 2, z = 0$$

Forward prop steps

$$a = x + y$$

$$b = \max(y, z)$$

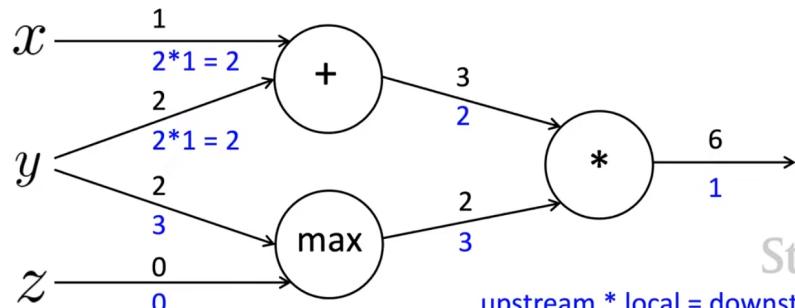
$$f = ab$$

Local gradients

$$\frac{\partial a}{\partial x} = 1 \quad \frac{\partial a}{\partial y} = 1$$

$$\frac{\partial b}{\partial y} = \mathbf{1}(y > z) = 1 \quad \frac{\partial b}{\partial z} = \mathbf{1}(z > y) = 0$$

$$\frac{\partial f}{\partial a} = b = 2 \quad \frac{\partial f}{\partial b} = a = 3$$



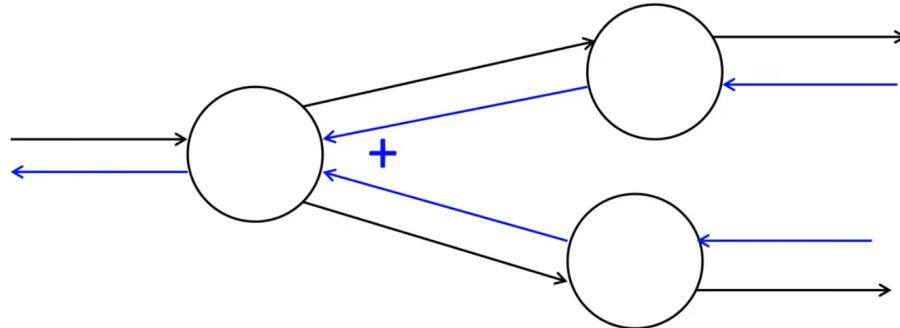
Stanfo

upstream * local = downstream

64

最后输入对应的下游梯度就是改变他们对应的权重，比如改变x那么输出结果就会变成两倍的x值

Gradients sum at outward branches



$$a = x + y$$

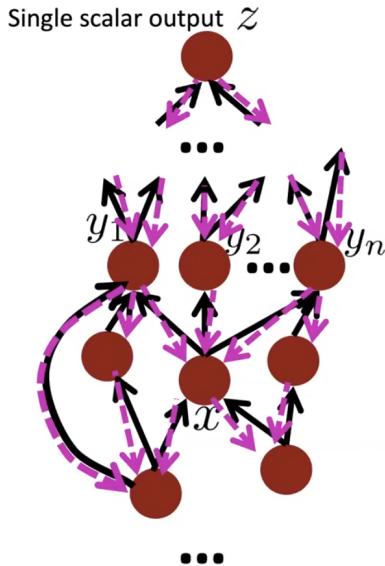
$$b = \max(y, z)$$

$$f = ab$$

$$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial a} \frac{\partial a}{\partial y} + \frac{\partial f}{\partial b} \frac{\partial b}{\partial y}$$

多个输出的时候只需要相加

Back-Prop in General Computation Graph



1. Fprop: visit nodes in topological sort order
 - Compute value of node given predecessors
2. Bprop:

- initialize output gradient = 1
- visit nodes in reverse order:
Compute gradient wrt each node using
gradient wrt successors
 $\{y_1, y_2, \dots, y_n\}$ = successors of x

$$\frac{\partial z}{\partial x} = \sum_{i=1}^n \frac{\partial z}{\partial y_i} \frac{\partial y_i}{\partial x}$$

Done correctly, big O() complexity of fprop and bprop is **the same**

In general, our nets have regular layer-structure
and so we can use matrices and Jacobians...

74

Stanfo

question

12

You

为什么我们都是在求 ds/dw 或者 ds/db 输入是 x 为什么不求 ds/dx



ChatGPT

在训练神经网络时，我们通常关注的是如何调整权重 w 和偏置 b 来最小化损失函数 s 。这是因为 w 和 b 是模型中的可学习参数，我们通过改变它们来改善模型的性能。所以，我们计算的是对权重的偏导数 $\frac{ds}{dw}$ 和对偏置的偏导数 $\frac{ds}{db}$ ，这些偏导数告诉我们如果稍微改变权重或偏置，损失函数会如何变化。

而 x 通常是固定不变的输入数据，比如训练集中的特征。在训练过程中，我们不会去调整输入数据来适应模型；相反，我们调整模型来适应输入数据。因此，我们不寻求 $\frac{ds}{dx}$ 作为训练的一部分，因为即使我们知道如何改变 x 来减少损失，我们在实际应用模型时也不能改变新的输入数据。

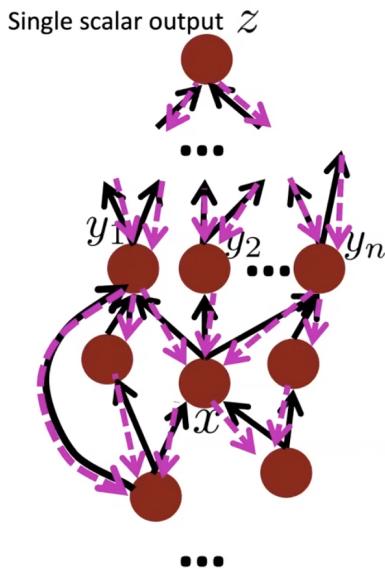
然而，在某些情况下， $\frac{ds}{dx}$ 仍然是有用的：

- **模型解释性**: 了解输入特征如何影响损失可以帮助我们解释模型的行为。
- **对抗性样本**: 在对抗性学习中， $\frac{ds}{dx}$ 用于创建对抗性样本，目的是故意扰乱模型的预测。
- **输入优化**: 如果输入是可调整的，例如在生成模型中，我们可能会希望调整输入以达到某种目标。

所以，尽管在标准的监督学习任务中，我们主要关注 $\frac{ds}{dw}$ 和 $\frac{ds}{db}$ ，但在特定的上下文或高级任务中， $\frac{ds}{dx}$ 也是重要的。



Back-Prop in General Computation Graph



1. Fprop: visit nodes in topological sort order
 - Compute value of node given predecessors
2. Bprop:
 - initialize output gradient = 1
 - visit nodes in reverse order:
 - Compute gradient wrt each node using gradient wrt successors
 - $\{y_1, y_2, \dots, y_n\}$ = successors of x

$$\frac{\partial z}{\partial x} = \sum_{i=1}^n \frac{\partial z}{\partial y_i} \frac{\partial y_i}{\partial x}$$

Done correctly, big O() complexity of fprop and bprop is **the same**

In general, our nets have regular layer-structure and so we can use matrices and Jacobians...

74

1. 前向传播 (Fprop) :

- 按照拓扑排序访问节点，即从输入层开始，按顺序向后访问，直到输出层。
- 对于每个节点，基于其前驱节点的值计算该节点的值。

2. 反向传播 (Bprop) :

- 初始化输出的梯度为 1，因为这是损失函数相对于自身的导数。
- 逆序访问节点，即从输出层开始，向前逐节点进行，直到输入层。
- 对每个节点，利用该节点对其后继节点的梯度来计算相对于该节点的梯度。

Manual Gradient checking: Numeric Gradient



For small h ($\approx 1e-4$),

$$f'(x) \approx \frac{f(x + h) - f(x - h)}{2h}$$

Easy to implement correctly

But approximate and **very slow**:

- You have to recompute f for **every parameter** of our model

Useful for checking your implementation

- In the old days, we hand-wrote everything, doing this everywhere was the key test
- Now much less needed; you can use it to check layers are correctly implemented

梯度检验的核心思想是用数值方法近似地计算函数的导数，并将其与你的解析梯度（即通过算法直接计算得到的梯度）进行比较。数值梯度的计算公式是：

$$f'(x) \approx \frac{f(x+h) - f(x-h)}{2h}$$

这里的点 x 是你想要计算梯度的地方， h 是一个很小的数，使得你可以计算 $f(x)$ 在 x 处的斜率的近似值。通常 h 选取为 $1e-4$ 左右，因为这个值在计算机中可以取得良好的精度，既不会太大以致于无法逼近真实的导数，也不会太小以致于受到浮点数舍入误差的影响。

幻灯片上指出了使用数值梯度的一些注意事项：

- 这种方法实现起来相对容易，并且当正确实现时可以为你的解析梯度提供一个很好的检验。
- 然而，数值梯度是近似值，并且计算速度非常慢，因为你需要为模型中的每个参数重新计算函数 f 。
- 尽管现在不像以前那样经常需要手动计算和检查梯度，但这个方法在检查新实现的层或复杂函数的梯度是否正确时仍然非常有用。

在现代深度学习框架中，通常使用自动微分来计算梯度，而梯度检验则用作调试工具，以确保自动微分的实现是正确的。如果数值梯度和解析梯度之间的差异非常小（在某个容忍阈值之内），我们可以相信我们的梯度计算是正确的。如果差异较大，则可能意味着解析梯度的计算存在错误。

We've mastered the core technology of neural nets!

- Backpropagation: recursively (and hence efficiently) apply the chain rule along computation graph
 - [downstream gradient] = [upstream gradient] \times [local gradient]
 - Forward pass: compute results of operations and save intermediate values
 - Backward pass: apply chain rule to compute gradients 
- 反向传播 (Backpropagation)**：这是一种递归（因而高效）的方式沿着计算图应用链式法则。它的基本原则是“下游梯度 = 上游梯度 \times 局部梯度”，这意味着梯度是如何通过网络反向传递的。
 - 前向传播 (Forward pass)**：这一步计算操作的结果，并保存中间值。在神经网络中，这通常涉及对输入数据应用权重，加上偏置，并通过激活函数，最终获得输出。
 - 后向传播 (Backward pass)**：在这一步，我们应用链式法则计算梯度。从输出开始，我们逆向工作，使用保存的中间值（从前向传播）和链式法则来计算每个参数相对于损失函数的梯度

反向传播是完整的循环算法，包含了前向传播和后向传播

反向传播 (Backpropagation) 和后向传播 (Backward pass) 这两个术语经常在深度学习的文献和讨论中互换使用，但它们可以从不同的角度描述同一过程：

1. 反向传播 (Backpropagation) :

- 这个术语通常用来描述整个用于训练神经网络的算法过程，它包括了前向传播（在网络中计算输出），计算损失，以及后向传播（计算梯度并传递回输入层的过程）。

- 反向传播强调的是算法的完整循环，包括了前向计算和后向梯度传播的整个序列。

2. 后向传播 (Backward pass) :

- 这个术语更具体地指反向传播过程中的一部分，即从输出层到输入层传递梯度的过程。
- 它仅指梯度计算和反向流动的部分，而不包括前向传播或梯度下降步骤（实际更新权重）。

Lecture4 Syntactic Structure and Dependency Parsing句法结构和依赖关系解析

在语言学和自然语言处理 (NLP) 中，句法结构分析是理解句子是如何构建的一个关键部分，它包括研究词语如何组合成短语和子句，以及这些短语和子句如何进一步组合成完整的句子。

“一致性”通常指的是句子成分在语法功能和形式上的一致匹配，如主谓一致，确保主语和谓语在人称和数上的一致。

“依赖性”通常描述的是句子中各个词语之间的依赖关系，即一个词（依赖词）在句法功能上依赖于另一个词（主导词）。依赖关系可以用来描绘句子的句法结构，是很多句法分析和句法解析算法的基础。

two views of linguistic structure

Constituency=phrase structure grammar = context-free grammars(CFGs) 无上下文语法

在自然语言处理中，CFGs 可以用来定义如何将句子分解为其组成部分，例如短语或单词。举个简单的例子，一个简单的CFG可能包含以下规则：

1. 句子 -> 名词短语 动词短语
2. 名词短语 -> 冠词 名词
3. 动词短语 -> 动词 名词短语
4. 冠词 -> 'the' | 'a'
5. 名词 -> 'cat' | 'dog'
6. 动词 -> 'chases' | 'sees'

这个CFG定义了如何生成简单句子，比如 "the cat chases the dog"。CFG中的每条规则描述了一个结构组件如何被分解或组合。

Phrase structure organizes words into nested constituents

Starting unit: words

the, cat, cuddly, by, door
Det N Adj P N

the
this
that
every

Words combine into phrases

NP [the cuddly cat] [by [the door] NP] PP
Prep

Phrases can combine into bigger phrases

the cuddly cat by the door

这是介绍句法结构的基础，显示了从单词到短语，再到更大短语的结构层次。

Dependency structure 依赖结构

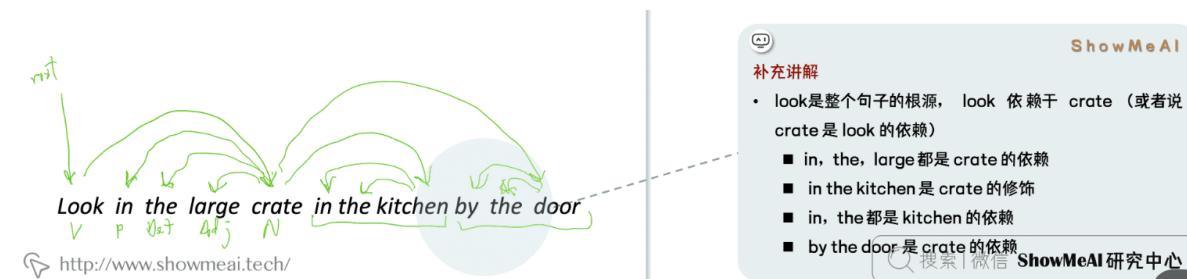
这张幻灯片讨论的是语言结构的另一种观点：依存结构（Dependency Structure）。依存结构揭示了词语之间的依赖关系，显示出哪些词语依赖于（修饰、附加到或作为参数的）其他词语。

在自然语言处理（NLP）中，依存结构是一种表现句子内部各个词之间关系的模型。依赖性分析是NLP中的一个任务，它识别出词之间的依存关系，通常是通过依存句法分析来完成的。

举例来说，在句子 "The cat sat on the mat" 中：

- "sat" 是主要动词，是句子的中心。
- "The cat" 是 "sat" 的主语，因此 "cat" 依赖于 "sat"。
- "on the mat" 是 "sat" 的状语，显示出位置，其中 "on" 依赖于 "sat"， "the mat" 依赖于 "on"。

依存结构通常通过一棵树来表示，树中的节点代表词语，而边代表词语之间的依存关系。这种结构对于许多NLP应用至关重要，例如句法分析、信息提取、问答系统和机器翻译，因为它有助于理解词语在句子中如何相互作用。



补充讲解

- `look` 是整个句子的根源, `look` 依赖于 `crate` (或者说 `crate` 是 `look` 的依赖)
 - `in`, `the`, `large` 都是 `crate` 的依赖
 - `in the kitchen` 是 `crate` 的修饰
 - `in`, `the` 都是 `kitchen` 的依赖
 - `by the door` 是 `crate` 的依赖

why do we need sentence structure为什么我们需要句子结构

- 为了能够正确地解释语言，我们需要理解句子结构
 - 人类通过将单词组合成更大的单元来传达复杂的意思，从而交流复杂的思想
 - 我们需要知道什么与什么相关联
 - 除非我们知道哪些词是其他词的参数或修饰词，否则我们无法弄清楚句子是什么意思

prepositional phrase attachment ambiguity 介词短语增加了歧义

San Jose cops kill man with knife

- 警察用刀杀了那个男子
 - cops 是 kill 的 subject (subject 指 主语)
 - man 是 kill 的 object (object 指 宾语)
 - knife 是 kill 的 modifier (modifier 指 修饰符)
 - 警察杀了那个有刀的男子
 - knife 是 man 的 modifier (名词修饰符, 简称为

人类大脑可能很好理解这句话，但是对于计算机来说可能会有两个含义，这就引起了歧义

The board approved [its acquisition] [by Royal Trustco Ltd.]
[of Toronto]
[for \$27 a share]
[at its monthly meeting].

歧义成倍增加

Coordination scope ambiguity 协调范围模糊

Shuttle veteran and longtime NASA executive Fred Gregory appointed to board

- 一个人: [[Shuttle veteran and longtime NASA executive] Fred Gregory] appointed to board
- 两个人: [Shuttle veteran] and [longtime NASA executive Fred Gregory] appointed to board

Adjectival/Adverbial Modifier Ambiguity 形容词修饰模糊

补充讲解

Students get first hand job experience

- `first hand` 表示第一手的，直接的，即学生获得了直接的工作经验
 - `first` 是 `hand` 的形容词修饰语(amod)
- `first` 修饰 `experience` , `hand` 修饰 `job`

Verb Phrase attachment ambiguity 动词短语(VP)依存歧义

Mutilated body washes up on Rio beach to be used for Olympic beach volleyball

- `to be used for Olympic beach volleyball` 是动词短语 (VP)
- 修饰的是 `body` 还是 `beach`

Dependency paths 依赖语法与树库

Dependency grammar and dependency structure 依赖语法和 依赖结构

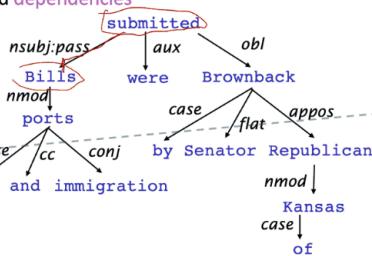


Dependency Grammar and Dependency Structure

Dependency syntax postulates that syntactic structure consists of relations between lexical items, normally binary asymmetric relations ("arrows") called **dependencies**

The arrow connects a **head** (governor, superior, regent) with a **dependent** (modifier, inferior, subordinate)

Usually, dependencies form a tree (connected, acyclic, single-head)



<http://www.showmeai.tech/>



依存文法和依存结构

ShowMeAI

- 关联语法假设句法结构包括词汇项之间的关系，通常是二元不对称关系（“箭头”），称为**依赖关系**



ShowMeAI

- 箭头连接头部(head)(调速器,上级,regent)和一个依赖(修饰词,下级,下属)
 - A → 依赖于 A 的事情
- 通常, 依赖关系形成一棵树(单头, 无环, 连接图)

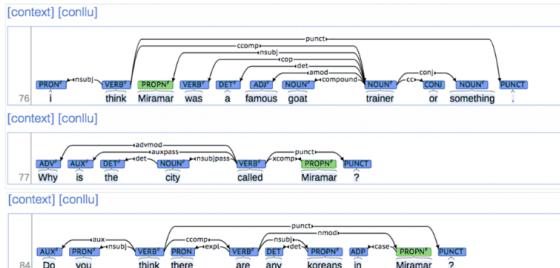
搜索 | 微信 ShowMeAI 研究中心

The rise of annotated data & Universal Dependencies treebanks 带注释数据的兴起：通用依存句法树库



The rise of annotated data: Universal Dependencies treebanks

[Universal Dependencies: <http://universaldependencies.org/>;
cf. Marcus et al. 1993, The Penn Treebank, *Computational Linguistics*]



<http://www.showmeai.tech/>

CS224n | Natural Language Processing with Deep Learning

Lecture 5: Dependency Parsing



带注释数据的兴起：通用依存句法树库

ShowMeAI

补充讲解

Universal Dependencies: 我们想要拥有一个统一的、并行的依赖描述, 可用于任何人类语言



ShowMeAI

- 从前手工编写语法然后训练得到可以解析句子的解析器
- 用一条规则捕捉很多东西真的很有效, 但是事实证明这在实践中不是一个好主意
 - 语法规则符号越来越复杂, 并且没有共享和重用人类所做的工作
- 句子结构上的treebanks 支持结构更有效

搜索 | 微信 ShowMeAI 研究中心

Stanford University X ShowMeAI

Treebank 是一个语料库, 其中的每个句子都经过了句法分析, 并以树状结构表示出来, 展示了句子中各个词语之间的语法关系。这些树状结构可以是依存关系树, 也可以是短语结构树。

the rise of annotated data



The rise of annotated data

Starting off, building a treebank seems a lot slower and less useful than building a grammar

But a treebank gives us many things

- Reusability of the labor
 - Many parsers, part-of-speech taggers, etc. can be built on it
 - Valuable resource for linguistics
- Broad coverage, not just a few intuitions
- Frequencies and distributional information
- A way to evaluate systems

<http://www.showmeai.tech/>

CS224n | Natural Language Processing with Deep Learning

Lecture 5: Dependency Parsing



带注释数据的兴起

ShowMeAI

从一开始, 构建 treebank 似乎比构建语法慢得多, 也没有那么有用



ShowMeAI

但是 treebank 给我们提供了许多东西

- 劳动力的可重用性
 - 许多解析器、词性标记器等可以构建在它之上
 - 语言学的宝贵资源
- 广泛的覆盖面, 而不仅仅是一些直觉
- 频率和分布信息
- 一种评估系统的办法

搜索 | 微信 ShowMeAI 研究中心

Stanford University X ShowMeAI

dependency conditioning preferences 依赖条件首选项

Christopher Manning
Stanford University
Natural Language Processing

Dependency Conditioning Preferences

What are the sources of information for dependency parsing?

1. Bilexical affinities [discussion → issues] is plausible
2. Dependency distance mostly with nearby words
3. Intervening material Dependencies rarely span intervening verbs or punctuation
4. Valency of heads How many dependents on which side are usual for a head?

ROOT Discussion of the outstanding issues was completed .

<http://www.showmeai.tech/>

CS224n | Natural Language Processing with Deep Learning

Lecture 5: Dependency Parsing

依赖条件首选项

依赖项解析的信息来源是什么？



ShowMeAI

1. Bilexical affinities (两个单词间的密切关系)
 - [discussion → issues] 是看上去有道理的
2. Dependency distance 依赖距离
 - 主要是与相邻词
3. Intervening material 介于中间的物质
 - 依赖很少跨越介于中间的动词或标点符号
4. Valency of heads
 - How many dependents on which side are usual for a head?

搜索 | 微信 ShowMeAI 研究中心

Stanford University X ShowMeAI

1. 双词亲和性 (Bilexical affinities): 指的是某些词对自然倾向于形成依存关系。例如，“discussion”和“issues”这两个词之间形成依存关系是合理的。
2. 依存距离 (Dependency distance): 依存关系通常存在于相邻的词之间，距离较近的词更可能形成依存。
3. 介入材料 (Intervening material): 依存关系很少跨越介入的动词或标点。
4. 核心词的价 (Valency of heads): 核心词 (head word) 通常与特定数量的依赖项 (dependents) 相关联，这个信息可以帮助解析器确定哪些词作为核心词是常见的。

Dependency parsing

Christopher Manning
Stanford University
Natural Language Processing

Dependency Parsing

- A sentence is parsed by choosing for each word what other word (including ROOT) is it a dependent of
- Usually some constraints:
 - Only one word is a dependent of ROOT
 - Don't want cycles A → B, B → A
- This makes the dependencies a tree
- Final issue is whether arrows can cross (non-projective) or not

ROOT I 'll give a talk tomorrow on bootstrapping

<http://www.showmeai.tech/>

CS224n | Natural Language Processing with Deep Learning

Lecture 5: Dependency Parsing

依赖关系分析

- 通过为每个单词选择它所依赖的其他单词(包括根)来解析一个句子



ShowMeAI

- 通常有一些限制
 - 只有一个单词是依赖干根的
 - 不存在循环 A→B, B→A
- 这使得依赖项成为树
- 最后一个问题是否箭头是否可以交叉(非投影的 non-projective)
 - 没有交叉的就是non-projective

搜索 | 微信 ShowMeAI 研究中心

Stanford University X ShowMeAI

**Projectivity**

- Defn: There are no crossing dependency arcs when the words are laid out in their linear order, with all arcs above the words
- Dependencies parallel to a CFG tree must be **projective**
 - Forming dependencies by taking 1 child of each category as head
- But dependency theory normally does allow non-projective structures to account for displaced constituents
 - You can't easily get the semantics of certain constructions right without these nonprojective dependencies



<http://www.showmeai.tech/>

CS224n | Natural Language Processing with Deep Learning

射影性

- 定义: 当单词按线性顺序排列时, 没有交叉的依赖弧, 所有的弧都在单词的上方

ShowMeAI

- 与CFG树并行的依赖关系必须是投影的
 - 通过将每个类别的一个子类别作为头来形成依赖关系
- 但是依赖理论通常允许非投射结构来解释移位的成分
 - 如果没有这些非投射依赖关系, 就不可能很容易获得某些结构的语义

搜索 | 微信 ShowMeAI 研究中心

Stanford University X ShowMeAI

基本的基于转换的依存关系解析器

- 解析器如下:

- 栈 σ 以 ROOT 符号开始, 由若干 w_i 组成
- 缓存 β 以输入序列开始, 由若干 w_i 组成
- 一个依存弧的集合 A , 一开始为空。每条边的形式是 (w_i, r, w_j) , 其中 r 描述了节点的依存关系
- 一组操作
- 最终目标是 $\sigma = [ROOT], \beta = \phi, A$ 包含了所有的依存弧

补充讲解

state之间的transition有三类:

1. SHIFT: 将buffer中的第一个词移出并放到stack上。
2. LEFT-ARC: 将 (w_j, r, w_i) 加入边的集合 A , 其中 w_i 是stack上的次顶层的词, w_j 是stack上的最顶层的词。
3. RIGHT-ARC: 将 (w_i, r, w_j) 加入边的集合 A , 其中 w_i 是stack上的次顶层的词, w_j 是stack上的最顶层的词。

我们不断的进行上述三类操作, 直到从初始态达到最终态。

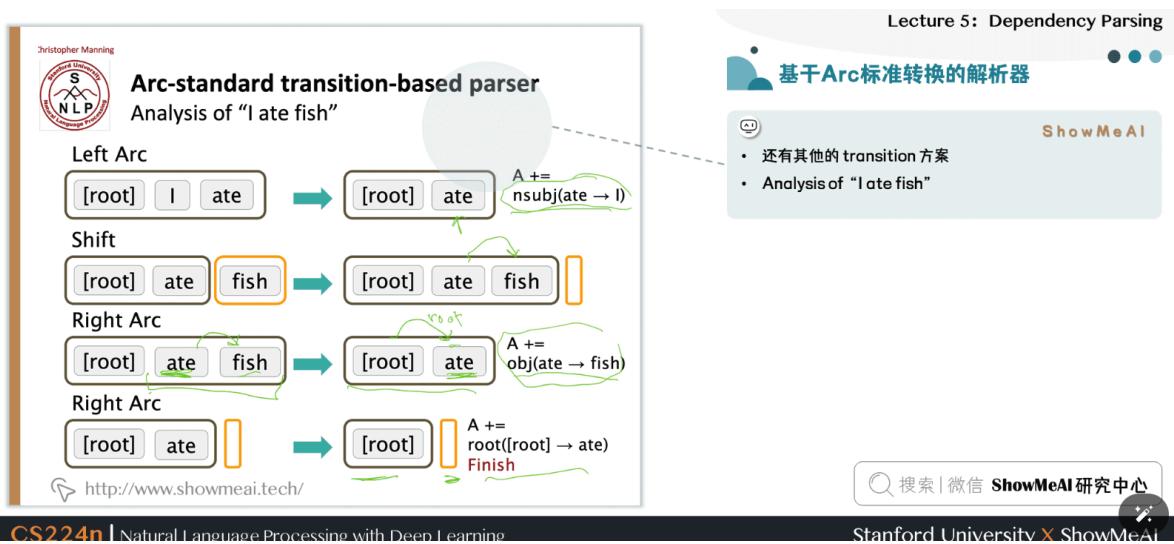
- 在每个状态下如何选择哪种操作呢?
- 当我们考虑到 LEFT-ARC 与 RIGHT-ARC 各有 $|R|(|R|$ 为 r 的类的个数)种类, 我们可以将其看做是class数为 $2|R| + 1$ 的分类问题, 可以用SVM等传统机器学习方法解决。

- **SHIFT**: 将下一个单词从输入移动到栈中。
- **LEFT-ARC**: 建立栈顶第二个单词到栈顶单词的依存关系，并且移除栈顶第二个单词。
- **RIGHT-ARC**: 建立栈顶单词到栈顶第二个单词的依存关系，并且移除栈顶单词。

在这个过程中，`σ` 表示当前栈的状态，`β` 表示输入队列的状态，`A` 表示到目前为止已经确定的依存关系。

举一个简单的例子，假设我们有句子 "I eat apples"，解析过程可能是这样的：

1. 初始状态，栈只有 ROOT，输入是 "I eat apples"。
2. SHIFT 操作，把 "I" 移动到栈中。
3. SHIFT 操作，把 "eat" 移动到栈中。
4. LEFT-ARC 操作，建立依存关系 "eat -> I"，并且 "I" 从栈中移除。
5. SHIFT 操作，把 "apples" 移动到栈中。
6. RIGHT-ARC 操作，建立依存关系 "eat -> apples"，并且 "apples" 从栈中移除。
7. 最后，栈中只剩下 ROOT 和 "eat"，并且所有单词都被处理完毕，依存树构建完成。



CS224n | Natural Language Processing with Deep Learning

Stanford University X ShowMeAI

如何选择下一步的行为：

- 我们需要解释如何选择下一步行动
 - Answer: 机器学习
- 每个动作都由一个有区别分类器(例如softmax classifier)对每个合法的移动进行预测
- 最多三种无类型的选择，当带有类型时，最多 $|R| \times 2 + 1$ 种
- Features: 栈顶单词，POS；buffer中的第一个单词，POS；等等
- 在最简单的形式中是没有搜索的
 - 但是，如果你愿意，你可以有效地执行一个 Beam search 束搜索(虽然速度较慢，但效果更好)：你可以在每个时间步骤中保留K个好的解析前缀
- 该模型的精度略低于依赖解析的最高水平，但它提供了非常快的线性时间解析，性能非常好

传统特征表示

这张幻灯片介绍了在依存句法分析中使用的常规特征表示方法。分析器如何选择它的下一步行动是通过机器学习的方法来实现的，其中使用了特征向量来代表解析器的当前状态。

3.5 传统特征表示

The slide illustrates the 'Conventional Feature Representation' used in dependency parsing. It shows a configuration with a Stack containing 'ROOT', 'has.VBZ', 'good.JJ', and 'He.PRP', and a Buffer containing 'control.NN'. A feature vector is generated as a sparse binary vector of length $10^6 \sim 10^7$, where each dimension corresponds to a feature template. Examples of feature templates include $s1.w = \text{good} \wedge s1.t = \text{JJ}$, $s2.w = \text{has} \wedge s2.t = \text{VBZ} \wedge s1.w = \text{good}$, and $lc(s2).t = \text{PRP} \wedge s2.t = \text{VBZ} \wedge s1.t = \text{JJ}$. The slide also defines 'Indicator features' as binary values indicating the presence of specific attribute-value pairs.

- 传统的特征表示使用二元的稀疏向量 $10^6 \sim 10^7$
- 特征模板：通常由配置中的1 ~ 3个元素组成
- Indicator features

这张幻灯片介绍了在依存句法分析中使用的常规特征表示方法。分析器如何选择它的下一步行动是通过机器学习的方法来实现的，其中使用了特征向量来代表解析器的当前状态。

幻灯片上展示了一个**特征向量**，它通常是高维的、稀疏的（这意味着大部分值为零），维度通常在 (10^6) 到 (10^7) 之间。这个向量包含了从解析配置中提取的不同特征，这些特征有助于机器学习模型预测下一步最佳的行动 (SHIFT, LEFT-ARC, 或 RIGHT-ARC)。

特征模板通常是配置中1到3个元素的组合。例如，幻灯片中展示的**特征模板**（在机器学习中，尤其是在自然语言处理任务中，特征模板用于从原始数据中提取出有助于学习和预测的信息。在依存句法分析中，特征模板定义了哪些属性和它们的组合会被用作输入，以便训练分类器来预测解析动作（比如 SHIFT, LEFT-ARC, 或 RIGHT-ARC）。）包括：

- 单个词的属性，如它的词性 (Part-of-Speech, POS)。
- 两个词之间的依存关系，如它们的词性和它们之间的特定关系。
- “指示器特征” (Indicator features)，它们是二进制的 (0或1)，指示某个特定的属性或属性组合是否存在。

在上面的幻灯片例子中，指示器特征表示了栈顶的词（如 "has"）、栈顶词的POS标签、缓冲区中的第一个词以及这些词的POS标签。这些特征将用于预测在给定的解析状态下执行哪个转移操作是最合适的。

这个过程是自然语言处理和机器学习中句法分析的关键组成部分，因为它允许计算模型学习如何识别和构建正确的语言结构，这是理解和处理自然语言的基础。

依赖分析的评估：(标记)依赖准确性

Evaluation of Dependency Parsing: (labeled) dependency accuracy

Acc = $\frac{\# \text{ correct deps}}{\# \text{ of deps}}$

UAS = $4 / 5 = 80\%$
LAS = $2 / 5 = 40\%$

Gold			Parsed		
1	2	She	1	2	She
2	0	saw	2	0	saw
3	5	the	3	4	the
4	5	video	4	5	video
5	2	lecture	5	2	lecture

Labels: nsubj, root, det, nn, obj, nsubj, ccomp

<http://www.showmeai.tech/>

ShowMeAI

• UAS (unlabeled attachment score) 指无标记依存正确率
• LAS (labeled attachment score) 指有标记依存正确率