cs 224n

Lecture 1 wordvecs1

Slides:

1.一些想法:

- 就像人是由周围的社会关系决定的，词是由上下文决定的。(PS：如果用马原来指导 这不是显然的嘛：） 把词当作人 人是一切社会关系的总和 词是一切上下文关系的总和)
- 一个词的表达有多了解他所处的周围坏境(及他的社会关系) 他就有多了解他自己 就有多接近他本身的意思。

2. 关于点积的疑惑点:

Dot product compares similarity of o and c.

[Mark] $u^T v = u.v = \sum_{i=1}^{n} u_i v_i$

Larger dot product = larger probability

为什么 点积越大与相似呢 比如说A,B 只有在A+B=C(定值) 时 A=B时点积最大 但是在这里 ui vi的和并不是一定的呀

为什么不直接用两个向量的差的向量的2-范数来表示？ 而且还是恒正的。 是因为计算量大嘛?

也许可以从向量内积的集合意义上入手理解:

> 内积的几何意义
>
> 点乘的几何意义是可以用来表征或计算两个向量之间的夹角，以及在b向量在a向量方向上的投影。
>
> 那么归一化后 就是在超球上的两个向量的夹角的cos值 该值越大 夹角越小 相似度越大
>
> Cosine Distance： 1 - Cosine Similarity 余弦距离

3.softmax理解摘要:

给定一组数据 或者可以看作一个向量 用softmax可以给出他们的一个概率分布:

This is an example of the **softmax function** $\mathbb{R}^n \to \mathbb{R}^n$

$$\text{softmax}(x_i) = \frac{\exp(x_i)}{\sum_{j=1}^{n} \exp(x_j)} = p_i$$

max:是因为最大的那个数据的概率(相比与直接平均)

soft:只因为仍然给那些小的数据一定的概率

这个挺漂亮的 但是最大的症结是为什么 ui*vi 越大 两个向量越相似。[**已解决(见2.)**]

Suggested Readings:

1.softax函数的一个性质：

对于softmax函数 如果所有的xi进行同样程度的平移(即减去一个向量) 函数输出不变。也就是说，它有一组“冗余”的参数(即可以将其中一个xi变为0)。

2. softmax 与 logistic regression:

softmax函数当其K=2时 依据1进行变形就会变为logistic regression:

$$h_\theta(x) = \frac{1}{\exp(\theta^{(1)\top}x) + \exp(\theta^{(2)\top}x^{(i)})} \begin{bmatrix} \exp(\theta^{(1)\top}x) \\ \exp(\theta^{(2)\top}x) \end{bmatrix}$$

Taking advantage of the fact that this hypothesis is overparameterized and setting $\psi = \theta^{(2)}$, we can subtract $\theta^{(2)}$ from each of the two parameters, giving us

$$\begin{aligned} h(x) &= \frac{1}{\exp((\theta^{(1)} - \theta^{(2)})^\top x^{(i)}) + \exp(\vec{0}^\top x)} \left[\exp((\theta^{(1)} - \theta^{(2)})^\top x) \exp(\vec{0}^\top x) \right] \\ &= \begin{bmatrix} \frac{1}{1+\exp((\theta^{(1)}-\theta^{(2)})^\top x^{(i)})} \\ \frac{\exp((\theta^{(1)}-\theta^{(2)})^\top x)}{1+\exp((\theta^{(1)}-\theta^{(2)})^\top x^{(i)})} \end{bmatrix} \\ &= \begin{bmatrix} \frac{1}{1+\exp((\theta^{(1)}-\theta^{(2)})^\top x^{(i)})} \\ 1 - \frac{1}{1+\exp((\theta^{(1)}-\theta^{(2)})^\top x^{(i)})} \end{bmatrix} \end{aligned}$$

3. 解决模型过大

运行梯度下降慢 过度拟合的问题 有两种既加快训练速度又提升词向量质量的办法：

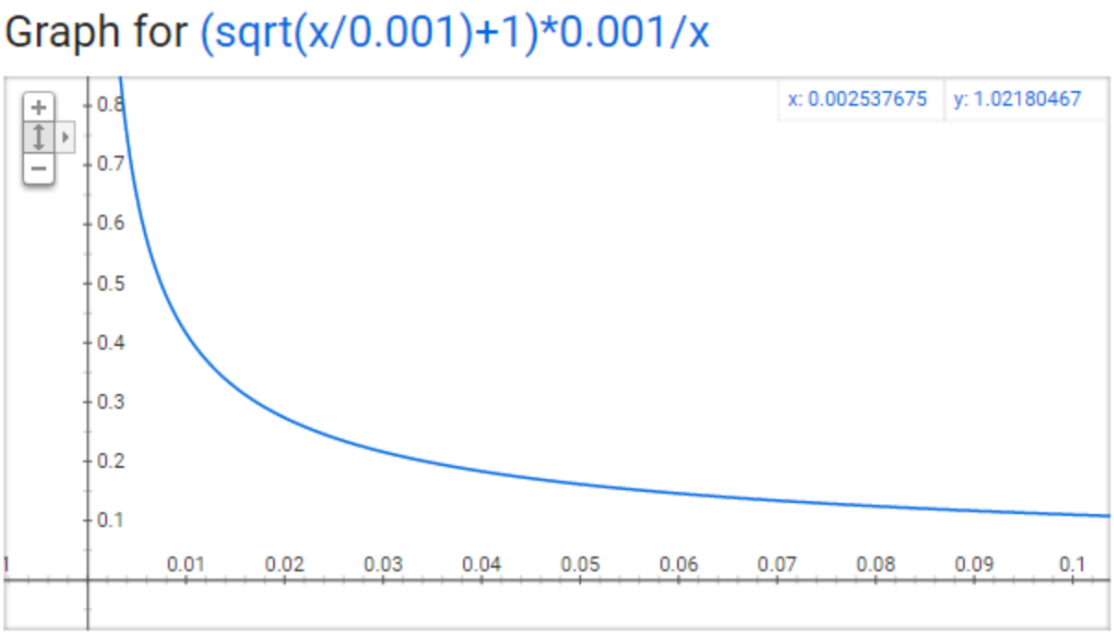
- 对频繁词进行二次采样以减少训练示例的数量
- 使用“负采样”的技术修改优化目标，每个训练样本仅更新模型权重的一小部分。

4. 二次采样：

word2vec C 代码实现了一个公式，用于计算在词汇表中保留给定单词的概率：

$$P(w_i) = (\sqrt{\frac{z(w_i)}{0.001}} + 1) \cdot \frac{0.001}{z(w_i)}$$

You can plot this quickly in Google to see the shape.



z(wi) 是语料库中属于该词的总词的概率分数

- 当 z(wi)<=0.0026z时，P(wi)=1.0P(wi)=1.0（被保留的机会为 100%）这意味着只有占总单词 0.26% 以上的单词才会被子采样。
- 当 z(wi)=0.00746时，P(wi)=0.5P(wi)=0.5（50% 的机会被保留）。
- 当 z(wi)=1.0，P(wi)=0.033P(wi)=0.033（3.3% 的机会被保留）。 也就是说，如果语料库完全由wiwi这个词组成，荒谬。

> [论文中定义的这个函数与 C 代码中实现的略有不同，但认为 C 实现是更权威的版本]

5.负采样:

- 思路：例如，在词对（“fox”、“quick”）上训练网络时，网络的“标签”或“正确输出”是一个单热向量。也就是说，对应于“quick”的输出神经元输出一个 1，而所有其他数千个输出神经元输出一个 0。 使用负采样，将随机选择少量“负”词（假设为 5 个）来更新权重。 （因此，只更新"正"词（“quick”）的权重，加上想要输出 0 的其他 5 个词的权重。总共有 6 个输出神经元，总共 1,800 个权重值。只是输出层中 3M 权重的 0.06%！
- 选择负样本： “负样本”（即我们将训练输出 0 的 5 个输出词）使用“一元分布”选择，其中更频繁的词更有可能被选为负样本。即利用单词出现的概率作为选择概率。作者在他们的论文中尝试了对该等式的多种变体，其中表现最好的是将字数提高到 3/4 次方：

$$P(w_i) = \frac{f(w_i)^{3/4}}{\sum_{j=0}^{n} \left(f(w_j)^{3/4} \right)}$$

  该等式倾向于增加出现频率较低的词的概率并降低出现频率较高的词的概率

> [论文称，选择 5-20 个单词对于较小的数据集效果很好，而对于大型数据集，只需选择 2-5 个单词即可。]

> [在隐藏层中，仅更新输入词的权重（无论是否使用负采样都是如此）]

Assignment 1:

1.奇异值分解(SVD):

[https://zhuanlan.zhihu.com/p/29846048]

[https://davetang.org/file/Singular_Value_Decomposition_Tutorial.pdf]

[lectures 7, 8, and 9 of CS168 课程笔记提供了对压缩通用算法(PCA/SVD)的高级处理]

SVD是对数据进行有效特征整理的过程。首先，对于一个m×n矩阵A，可以理解为其有m个数据，n个特征，（想象成一个n个特征组成的坐标系中的m个点），然而一般情况下，这n个特征并不是正交的，也就是说这n个特征并不能归纳这个数据集的特征。

SVD的作用就相当于是一个坐标系变换的过程，从一个不标准的n维坐标系，转换为一个标准的k维坐标系，并且使这个数据集中的点，到这个新坐标系的欧式距离为最小值（也就是这些点在这个新坐标系中的投影方差最大化），其实就是一个最小二乘的过程。

进一步，如何使数据在新坐标系中的投影最大化呢，就需要让这个新坐标系中的基尽可能的不相关，可以用协方差来衡量这种相关性。当对这个协方差矩阵进行特征分解之后，可以得到奇异值和右奇异矩阵，而右奇异矩阵则是一个新的坐标系，奇异值则对应这个新坐标系中每个基对于整体数据的影响大小，这时便可以提取奇异值最大的k个基作为新的坐标。

2.外积、内积和叉积：

英语语境里，外积(outer)和叉积(cross)是不一样的，外积是列向量乘行向量（内积相反），叉积是叉乘的结果。

3.matplotlib.pyplot.imshow 的一个小问题：

matplotlib.pyplot.imshow 的一个小问题是它可能会产生奇怪的结果(如果呈现的数据不是 uint8) 为了解决这个问题，应该在显示之前将图像显式转换为 uint8。

```
plt.imshow(np.uint8(img_tinted))
plt.show()
```

4. 高效SVD：

numpy、scipy 和 scikit-learn（sklearn）都提供了一些 SVD 的实现，但只有 scipy 和 sklearn 提供了 Truncated SVD 的实现，只有 sklearn 提供了计算大规模 Truncated SVD 的高效随机算法。所以应该使用 sklearn.decomposition.TruncatedSVD。

5. SVD实现的小细节：

SVD 存在"符号不确定性"的问题，这意味着组件的符号和变换的输出取决于算法和随机状态。要解决此问题，应将此类的实例与数据拟合一次，然后保留该实例以进行转换。

> https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.TruncatedSVD.html

```
SVD=TruncatedSVD(n_components=k, n_iter=n_iters, random_state=42)
    M_reduced=SVD.fit_transform(M)
```

6. numpy broadcasting :

> Computation on Arrays: Broadcasting by Jake VanderPlas.

7. 绘制图的参考 可以参考matplotlib库 简直应有尽有 :

> the Matplotlib gallery

8. Word2Vec使用小细节:

- 没法用gensim.downloader.load() 改成本地下载

```
from gensim import models

wv_from_bin = models.KeyedVectors.load_word2vec_format(
    '/home/wangxidong/gensim-data/word2vec-google-news-300/GoogleNews-
vectors-negative300.bin', binary=True)

-vocab = list(wv_from_bin.vocab.keys())
-print("Loaded vocab size %i" % len(vocab))
```

- 但上述方法的倒数第二行在gensim更新为4.0.0即以上后无法使用

> 参考https://github.com/RaRe-Technologies/gensim/wiki/Migrating-from-Gensim-3.x-to-4

使用如下方法代替:

```
vocab = list(wv_from_bin.index_to_key)
```

9. 查conda库:

> https://anaconda.org/

Lecture 2 wordvecs2

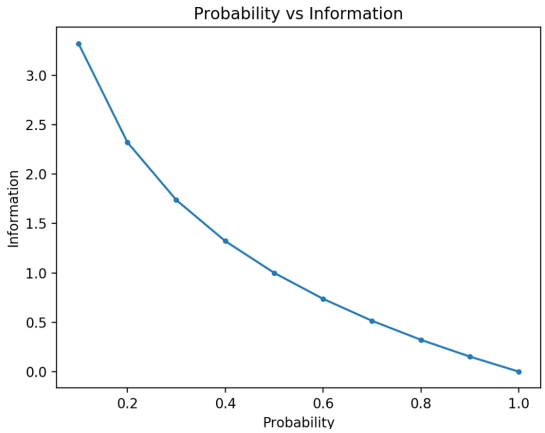
Slides:

1. entropy:

- 计算事件的信息:

  信息论背后的基本直觉是，了解不太可能发生的事件比了解可能发生的事件提供更多信息。

  > information(x) = h(x) = -log(p(x)) log以2为底 所以衡量信息信息度量的单位是比特

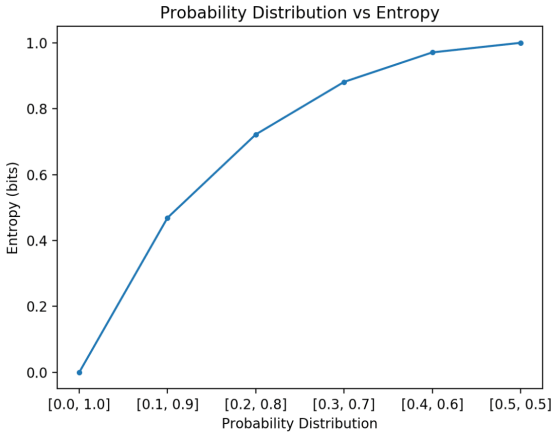


- 计算随机变量的熵：

  熵 表示或传输 从随机变量的概率分布中 提取事件 所需的平均比特数

  > H(X) = -sum(each k in K p(k) * log(p(k)))

  最低熵是针对具有概率为 1.0（确定性）的单个事件的随机变量计算的。 随机变量的最大熵是所有事件的可能性相等。



> 注意，在计算熵时，必须为概率添加一个很小的值，以避免计算零值的对数，这会导致无穷大而不是数字。

2. cross entropy：

交叉熵是信息论领域的一种度量，建立在熵的基础上，是对给定随机变量或事件集的两个概率分布之间差异的度量。

交叉熵是当使用模型 q 时，对来自分布为 p 的源的数据进行编码所需的平均位数

- 两个概率分布之间的交叉熵，例如来自 P 的 Q，可以正式表示为：

  > H(P, Q) = – sum x in X P(x) * log(Q(x))

  其中 P 可能是目标分布，Q 是目标分布的近似值。

  其中 P(x) 是事件 x 在 P 中的概率，Q(x) 是事件 x 在 Q 中的概率，log 是以 2 为底的对数，这意味着结果以位为单位。

  [如果改为使用 base-e 或自然对数，则结果的单位将称为 nats]

  如果两个概率分布相同，结果将是一个以比特为单位测量的正数，并且将等于分布的熵。

Suggested Readings:

Python review session:

Slides:

1. python:

python-review.ipynb

https://www.w3schools.com/python/

2. numpy:

https://cs231n.github.io/python-numpy-tutorial/

https://numpy.org/doc/stable/user/quickstart.html

3. MATPLOTLIB:

https://matplotlib.org/stable/gallery/index.html

Lecture 3 neural nets

Slides:

1. 一些基本知识

Lecture 4 backprop

Slides:

1.矩阵运算技巧:

训练尽量vectorized 而不是 for 循环

2. 非线性函数:

为了构建一个前馈深度网络，应该尝试的第一件事是 ReLU——由于良好的梯度回流，它可以快速训练并且表现良好

3. [Mark] 参数初始化：

- **通常必须将权重初始化为小的随机值 避免产生对称性**
- Xavier 初始化的方差与扇入 nin（前一层大小）和扇出 nout（下一层大小）成反比：

$$\mathrm{Var}(W_i) = \frac{2}{n_{\text{in}} + n_{\text{out}}}$$

4. 最优化：

对于复杂的网络和情况，通常使用一系列更复杂的“自适应”优化器中的一个会做得更好，这些优化器通过累积梯度来缩放参数调整。

- Adagrad
- RMSprop
- Adam [在许多情况下，相当好的、安全的]
- SparseAdam
- ...

5. 学习率

通常可以通过在训练时降低学习率来获得更好的结果

- 手动：每 k 个 epoch 将学习率减半
- 通过公式：$lr = lr_0 e^{-kt}$, for epoch t

  > epoch[对数据的一次传递(打乱或采样)]

- 还有更高级的方法，比如循环学习率 (q.v.) [帮助模型跳出大坑]

> 更高级的优化器仍然使用学习率，但它可能是优化器收缩的初始速率——因此可能能够从高开始(如0.1)

Suggested Readings:

1. backprop的一些细节：

注意梯度消失等一些问题

Assignment 2

1.矩阵求导参考：

Review of differential calculus

a2 的 gradient-notes

1.1 几点注意事项：

- 区分 微分 梯度 偏导数

$$
\begin{aligned}
f(x+h) &= f(x) + \mathrm{d}_x f(h) + o_{h\to 0}(h) && \textbf{differential} \\
&= f(x) + \langle \nabla_x f | h \rangle + o_{h\to 0}(h) && \textbf{gradient} \\
&= f(x) + \langle \frac{\partial f}{\partial x}(x) | h \rangle + o_{h\to 0} \\
&= f(x) + \langle \begin{pmatrix} \frac{\partial f}{\partial x_1}(x) \\ \vdots \\ \frac{\partial f}{\partial x_n}(x) \end{pmatrix} | h \rangle + o_{h\to 0} && \textbf{partial derivatives}
\end{aligned}
$$

- 注意在**Rn到R**的关系中(即只有一个输出函数/输出矩阵有一个维度为一维) Jacobian行列式与梯度之间有一个转置关系 $J(x) = \nabla_x f^T$

  因为定义的时候Jacobian就是个行向量 $\frac{\partial f_1}{\partial x_1}(x) \dots \frac{\partial f_1}{\partial x_n}(x)$

  但是梯度被定义为列向量 所以之间有个转置的关系

  > 当有多个输出函数时(大多数情况) 两者数值上相同

- 标记 $\frac{\partial}{\partial \cdot}$符号通常是模棱两可的，可以指梯度或Jacobian行列式。

1.2 几个有用的矩阵求导二级结论

(推导见a2 gradient-notes)

- 矩阵乘以列向量 并对列向量求导

  $$\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x} \quad \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{x}} = \boldsymbol{W}$$

- 行向量乘以矩阵 并对行向量求导

  $$\boldsymbol{z} = \boldsymbol{x}\boldsymbol{W} \quad \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{x}} = \boldsymbol{W}^T$$

- 向量对自己求导

  $$\boldsymbol{z} = \boldsymbol{x} \quad \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{x}} = \boldsymbol{I}$$

- 作用在矩阵元素上的函数 对矩阵求导

  $$\boldsymbol{z} = f(\boldsymbol{x}) \quad \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{x}} = \mathrm{diag}(f'(\boldsymbol{x}))$$

- 矩阵乘以列向量 并对矩阵求导

  $$\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x},\ \boldsymbol{\delta} = \frac{\partial J}{\partial \boldsymbol{z}} \quad \frac{\partial J}{\partial \boldsymbol{W}} = \boldsymbol{\delta}^T \boldsymbol{x}^T$$

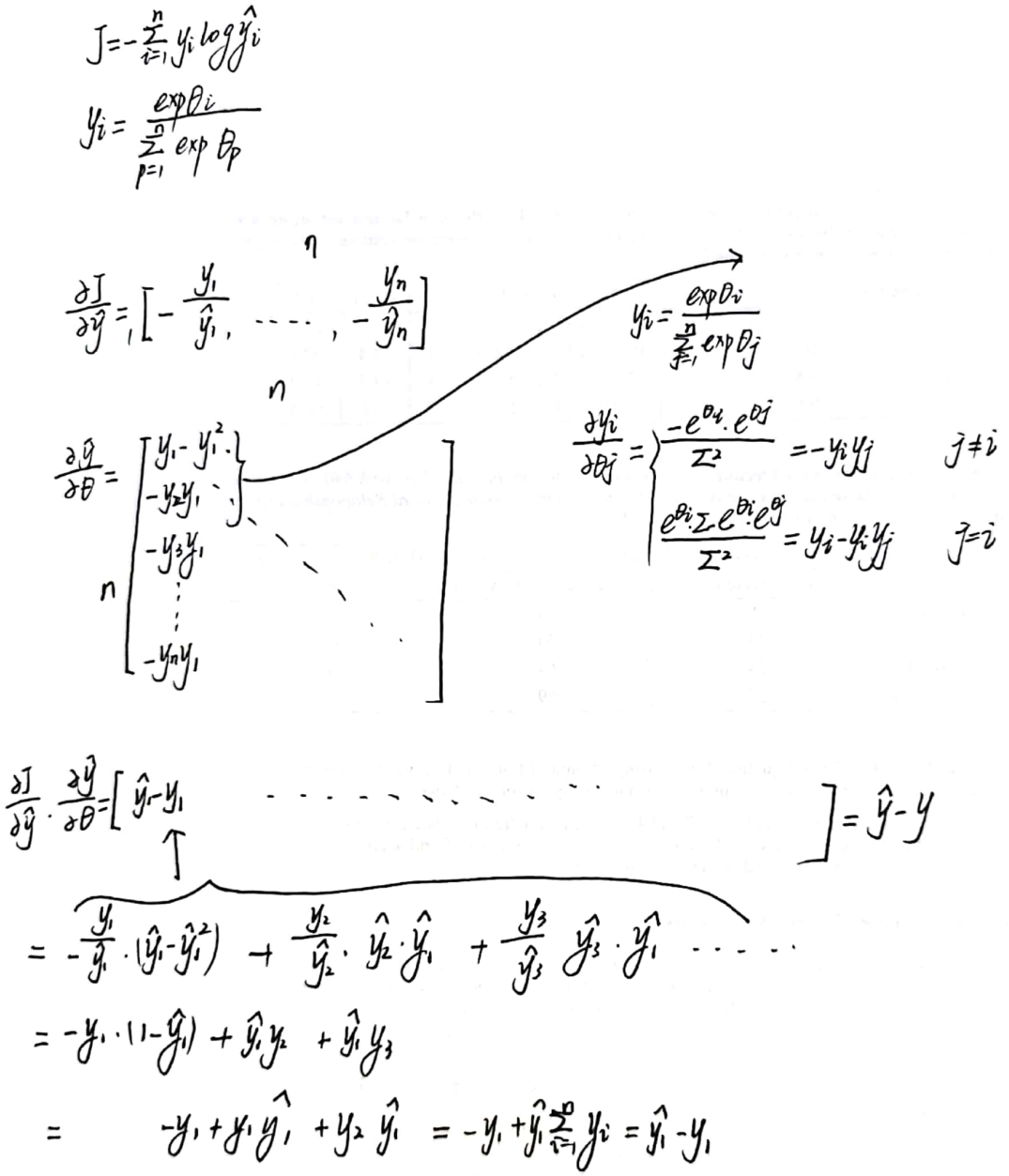
- 行向量乘以矩阵 并对矩阵求导

  $$\boldsymbol{z} = \boldsymbol{x}\boldsymbol{W},\ \boldsymbol{\delta} = \frac{\partial J}{\partial \boldsymbol{z}} \quad \frac{\partial J}{\partial \boldsymbol{W}} = \boldsymbol{x}^T \boldsymbol{\delta}$$

- 与 logits 相关的交叉熵损失

  $$\textbf{Cross-entropy loss with respect to logits } (\hat{\boldsymbol{y}} = \mathrm{softmax}(\boldsymbol{\theta}),\ J = CE(\boldsymbol{y}, \hat{\boldsymbol{y}}), \text{ what is } \frac{\partial J}{\partial \boldsymbol{\theta}}?)$$

  $$\frac{\partial J}{\partial \boldsymbol{\theta}} = \hat{\boldsymbol{y}} - \boldsymbol{y}$$ (如果为y是列向量的话 就转置一下)



- 矩阵求导如果没有square型的矩阵 则只有唯一的满足矩阵乘法性质的排列和转置 所以可以用这个性质去检验

2. 对偶空间

对于有限维矢量空间V而言，所有线性映射$V \to \mathbb{R}$构成一个矢量空间V^* V与V^*的一一对应

(只要构造 线性映射空间的几个基映射 即可)

3. word2vec 的实现

实现了 word2vec 模型并使用随机梯度训练下降 (SGD)训练词向量。

代码详见 https://github.com/wangxidong06/CS224N/tree/master

Lecture 5 dep-parsing

Slides:

1. 一些基本知识

Suggested Readings:

Lecture 6 rnn-lm

Slides:

1. 一些基本知识

Suggested Readings:

Assignment 3

1. pytorch

PyTorch Tutorial Session[colab notebook] [preview]

2. Neural Transition-Based Dependency Parsing 的实现

实现了一个基于神经网络的依赖解析器，目标是最大限度地提高 UAS（未标记附件分数）指标的性能。

代码详见https://github.com/wangxidong06/CS224N/tree/master

Lecture 7 fancy-rnn

Slides:

1. 一些基本知识

Suggested Reading:

Assignment 4