[白话解析] 带你一起梳理Word2vec相关概念

原创 罗西的思考 罗西的思考 2月6日

[白话解析] 带你一起梳理Word2vec相关概念

0x00 摘要

本文将尽量使用易懂的方式，尽可能不涉及数学公式，而是从整体的思路上来说，运用感性直觉的思考来帮大家梳理Word2vec相关概念。

0x01 导读

1. 原委

本来只是想写Word2vec，没想到一个个知识点梳理下来，反而Word2vec本身只占据了一小部分。所以干脆就把文章的重点放在梳理相关概念上，这样大家可以更好的理解Word2vec。

为了讨论Word2vec，我们需要掌握（或者暂且当做已知）的先决知识点有：

- 独热编码 / 分布式表示
- score function / loss function
- 自信息 / 熵 / 交叉熵 / KL散度 / 最大似然估计
- Softmax回归

我们知道，机器学习的4个关键部分：

- 输入的数字化表示 。
- 输出的数字化表示 。
- 输入到输出的映射方法。
- 对映射方法的评价标准。

所以我们的基本梳理流程是从输入到输出。就是

- 因为神经网络的输入，所以看到了独热编码 / 分布式表示。

- 因为神经网络的训练/损失函数，所以梳理自信息 / 熵 / 交叉熵 / KL散度 / 最大似然估计等一系列概念。
- 因为神经网络的输出，所以得理清Softmax回归。

机器学习里面概念艰深晦涩，和数学/物理都有关联，同一个概念往往会有各种解读。所以大家在各种解读中，找到一种自己能理解的，然后深入下去就行。

2. 简述Word2vec

Word2vec模型其实就是简单化的神经网络。但是这个神经网络的学习不是为了准确的预估正确的中心词/周围词，而是为了得到 `word—>vector` 这个映射关系。其具体分解如下：

- 构建数据：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]。
- 输入层: 将所有词语进行one-hot编码作为输入，输入的是n维向量(n是词表单词个数)
- 隐藏层：中间是只有一个隐藏层 (没有激活函数，只是线性的单元)。隐藏层实际上存储了词汇表中所有单词的word vectors。这是一个尺寸为 [vocabulary size x embedding size] 的矩阵。矩阵的每一行对应了某一个单词的word vector。
- 输出层：输出的也是独热向量。Output Layer维度跟Input Layer的维度一样，各维的值相加为1。用的是Softmax回归。softmax保证输出的向量是一个概率分布。一旦转换为概率之后，我们就可以用到最大似然估计（交叉熵）的方式来求得最大似然或者最小交叉熵。
- 定义loss损失函数：用来预测正确输出/优化模型。

  我们的 label y 值是一个概率分布，输出层经过softmax处理后，也是一个概率分布，这样就可以用交叉熵来衡量神经网络的输出与我们的 label y 的差异大小，也就可以定义出loss了。
- 迭代训练: 采用梯度下降算法，每次迭代比较 prediction 和 label y 之间的loss，然后相应优化，最终确保类似的单词有类似的向量。
- 最终矩阵：丢去output层，只用隐藏层的输出单元(就是Input Layer和Hidden Layer之间的权重)，构成了Look up table。

当这个模型训练好以后，我们并不会用这个训练好的模型处理新的任务，我们真正需要的是这个模型通过训练数据所学得的参数，例如隐层的权重矩阵。

下面就让我们一层层剖析相关概念。

0x01 数据/输入层相关概念

计算机系统只能识别数字，所以需要对观察到的内容进行数字表示，才能达到预测目标。

1. 独热编码

one-hot编码就是保证每个样本中的单个特征只有1位处于状态1，其他的都是0。具体编码举例如下，把语料库中，杭州、上海、宁波、北京每个都对应一个向量，向量中只有一个值为1，其余都为0。

```
杭州 [0,0,0,0,0,0,0,1,0,……, 0,0,0,0,0,0,0]
上海 [0,0,0,0,1,0,0,0,0,……, 0,0,0,0,0,0,0]
宁波 [0,0,0,1,0,0,0,0,0,……, 0,0,0,0,0,0,0]
北京 [0,0,0,0,0,0,0,0,0,……, 1,0,0,0,0,0,0]
```

其缺点是：

- 向量的维度会随着句子的词的数量类型增大而增大；如果将世界所有城市名称对应的向量合为一个矩阵的话，那这个矩阵过于稀疏，并且会造成维度灾难。
- 城市编码是随机的，向量之间相互独立，无法表示语义层面上词汇之间的相关信息。

所以，人们想对独热编码做如下改进：

- 将vector每一个元素由整形改为浮点型，变为整个实数范围的表示；
- 转化为低维度的连续值，也就是稠密向量。将原来稀疏的巨大维度压缩嵌入到一个更小维度的空间。并且其中意思相近的词将被映射到向量空间中相近的位置。

简单说，要寻找一个空间映射，把高维词向量嵌入到一个低维空间。然后就可以继续处理。

2. 分布式表示

分布式表示（Distributed Representation）其实Hinton 最早在1986年就提出了，基本思想是将每个词表达成 n 维稠密、连续的实数向量。而实数向量之间的关系可以代表词语之间的相似度，比如向量的夹角cosine或者欧氏距离。

有一个专门的术语来描述词向量的分布式表示技术——词嵌入【word embedding】。从名称上也可以看出来，独热编码相当于对词进行编码，而分布式表示则是将词从稀疏的大维度压缩嵌入到较低维度的向量空间中。

Distributed representation 最大的贡献就是让相关或者相似的词，在距离上更接近了。

Distributed representation 相较于One-hot方式另一个区别是维数下降极多，对于一个100W的词表，我们可以用100维的实数向量来表示一个词，而One-hot得要100W维。

为什么映射到向量空间当中的词向量就能表示是确定的哪个词并且还能知道它们之间的相似度呢？

- 关于为什么能表示词这个问题。分布式实际上就是求一个映射函数，这个映射函数将每个词原始的one-hot表示压缩投射到一个较低维度的空间并一一对应。所以分布式可以表示确定的词。

- 关于为什么分布式还能知道词之间的关系。就必须要了解分布式假设（distributional hypothesis）。其基于的分布式假设就是出现在相同上下文(context)下的词意思应该相近。所有学习word embedding的方法都是在用数学的方法建模词和context之间的关系。

词向量的分布式表示的核心思想由两部分组成：

- 选择一种方式描述上下文；
- 选择一种模型刻画目标词与其上下文之间的关系。

事实上，不管是神经网络的隐层，还是多个潜在变量的概率主题模型，都是应用分布式表示。

0x02 训练/隐藏层相关概念

这里的重点是交叉熵作为损失函数，从而扯出了一系列概念。

- KL散度（相对熵 ）可以衡量两个分布的接近程度，按说应该用KL散度来衡量损失计算代价。
- 而在特定情况下最小化KL散度等价于最小化交叉熵。且交叉熵的运算更简单，所以改用交叉熵来当做代价。
- 最小化交叉熵和最大似然函数的结果一样，所以两者都可以做损失函数。

1. score function

线性分类的方法包含2个重要的组成部分：score function和loss function。

假设训练样本的真实分类为K类。

线性分类器的score function就是以下线性函数：f(xi) = Wxi + b。其中W（[K x D]矩阵）和b（[K x 1]向量）统称为score function的参数（parameters）。

函数输出是一个K x 1的向量，其中第 i 项表示在第 i 类的得分。分数越高，表示 xi 更有可能是这一类。

W和b的**第i行**决定了**第i类**的分数，因此W和b的**第i行**被称为**第i类**的分类器（classifier）。

既然 xi 是一个D维向量（每一个维度是一个特征），那么我们可以将它看作是D维空间（样本空间）上的一个点。

对于每个classifier，都有一条直线（准确地说是决策边界Decision Boundary）。所以我们能看出来 xi 在这条直线的什么方位，就能做出分类了。

2. loss function

有时也被称为cost function或the objective。用于衡量模型(W和b)在输入集上的预测结果与真实标签的拟合程度。

“预测更准确”等价于“loss更小”。

3. 自信息

符合分布 P 的某一事件 x 出现，传达这条信息所需的最少信息长度为自信息，表达为

$$I(x) = log\frac{1}{P(x)}$$

解释：

关于编码。传输一个随机变量状态值需要一些状态位。编码越长，信息量越大。所以有这样一个等式： `p=1/a^n` 。

- p表示取到某个值的概率。
- a表示存储单元能够存储的数量，如果存储单元是bit，那 a 就是 2。
- n表示编码长度。

所以随机变量状态总取值的可能性应该是a^n，那取到单个值的概率就是 `p=1/a^n` 。从而编码长度就是如 **I(x)** 那个公式推导出来。

4. 熵

从分布 P 中随机抽选一个事件，传达这条信息（事件）所需的最优平均信息长度为香农熵，表达为

$$H(P) = \sum_x P(x) log \frac{1}{P(x)}$$

解释：

如果一个事件发生的概率是p(x)，则其自信息是 I(x)。

最优平均信息长度就是每种可能的信息熵的加权平均数(或者说是期望)。就是{信息长度}乘上{该信息发生的概率}, 然后求和。

$$\text{信息熵} = \sum_x (\text{信息 } x \text{ 发生的概率} \times \text{验证信息 } x \text{ 需要的信息量})$$

举例：

比如说：赌马比赛里，有4匹马 {A, B, C, D}，获胜概率分别为{1/2, 1/4, 1/8, 1/8}。则

$$H(P) = \frac{1}{2} log(2) + \frac{1}{4} log(4) + \frac{1}{8} log(8) + \frac{1}{8} log(8) = \frac{1}{2} + \frac{1}{2} + \frac{3}{8} + \frac{3}{8} = \frac{7}{4} bits$$

在二进制计算机中，一个比特为0或1，其实就代表了一个二元问题的回答。也就是说在计算机中，我们给哪一匹马夺冠这个事件进行编码，所需要的平均码长为1.75个比特。

5. 交叉熵

关于样本集的两个概率分布p(x)和q(x)，其中p(x)是未知的真实分布，q(x)是近似的非真实分布。如果用非真实分布q(x)来表示来自真实分布p(x)的平均编码长度，则称之为交叉熵。

$$H(p, q) = \sum_x p(x) log \frac{1}{q(x)} = -\sum_x p(x) \log q(x)$$

解释：

- 要传达的信息来自哪个分布，答案是 近似分布 q。
- 信息传递的方式由哪个分布决定，答案是 真实分布 p。
- 交叉熵就是用q(x)来对p(x)进行建模，用q(x)建立一个编码体系，把x的值传递给接收者。

其实**大家容易迷惑的**就是，在交叉熵公式中，p和q哪个放到 log() 里面。可以大致这么理解：

- 因为交叉熵公式中，**log() 这个是用来计算自信息，是编码长度**。

- 而**q(x)**目的就是要模拟逼近真实分布p(x)来进行编码传输信息。所以把q(x)放 log() 这里。

换这种思路理解也许会更清楚：

用分布 Q 的最佳信息传递方式来传达 真实分布 P 中随机抽选的一个事件，所需的平均信息长度为交叉熵，表达为

$$Hp(Q) = \sum_x P(x) log \frac{1}{Q(x)}$$

分布 Q 就是近似的非真实分布，用来编码P中随机抽选的某一个事件x，这个事件的编码结果就是Q(x)。这个事件x是用P中的概率P(x)来选取的。

举例子：

还是上面的赌马比赛里4匹马 {A, B, C, D}，Q分布预测的获胜概率分别为{1/4, 1/4, 1/4, 1/4}。则

$$H(P,Q) = \frac{1}{2} log(4) + \frac{1}{4} log(4) + \frac{1}{8} log(4) + \frac{1}{8} log(4) = \frac{2}{2} + \frac{2}{4} + \frac{2}{8} + \frac{2}{8} = 2bits$$

这样用Q来传送P，需要 2 bits。

6. KL 散度

有两个概率分布为：P , Q 。KL散度告诉了Q和P的接近程度，也就是相似度，利用交叉熵减去信息熵即可。

$$D_{KL}(P||Q) = H(P,Q) - H(P)$$

也可以这么理解：

用分布 Q 的最佳信息传递方式来传达分布 P，比 用分布 P 自己的最佳信息传递方式来传达分布 P，平均多耗费的信息长度为 KL 散度，KL 散度衡量了两个分布之间的差异。

这里可以理解为 P 是未知的真实分布，Q 是近似的非真实分布。

KL散度表达为：

$$D_{KL}(P||Q) = Dq(P) = H(P,Q) - H(P) = \sum_x P(x) log \frac{1}{Q(x)} - \sum_x P(x) log \frac{1}{P(x)} = \sum_x P(x) log$$

因为KL散度不具有交换性，所以不能理解为“距离”的概念，衡量的并不是两个分布在空间中的远近，更准确的理解还是衡量一个分布相比另一个分布的信息损失(information lost)。

在对数以2为底时， *log* 2 ，可以理解为“我们损失了多少位的信息”。

注意，

- 如果表达成 D_q(P) 形式，要传达的信息所属的分布在括号内；
- 如果表达成 D_KL(P||Q) 形式，要传达的信息所属的分布在前。

公式 D_Q(P) 里一共涉及了两个分布：

- 要传达的信息来自哪个分布，答案是 P
- 信息传递的方式由哪个分布决定，答案是 Q

按照之前的例子。

KL散度是 2 bits - 7/4 bits = 1/4 bit. 说明 在信息论中，用q去拟合p的产品的信息损耗是 1/4 bit。

KL散度可以被用于计算代价，而在机器学习评价两个分布之间的差异时，由于分布 P 会是给定的，所以此时 KL 散度和交叉熵的作用其实是一样的，而且因为交叉熵少算一项 (相对熵 = 交叉熵 - 信息熵)，更加简单，所以选择交叉熵会更好。

7. 交叉熵作为损失函数

交叉熵刻画的是实际输出（概率）与期望输出（概率）的距离，

交叉熵的值越小，两个概率分布就越接近，即拟合的更好。若P(x)为数据原始分布，则使交叉熵最小的分布Q(x)即是与P(x)最接近的分布。

在信息论中完整的交叉熵公式是：

$$CrossEntropy: H(p,q) = H(p) + D_{KL}(p \,||\, q)$$
$$Hp(q) = -\sum_{x} p(x) \log q(x)$$

可以看出**cross-entropy loss其实就是将p和q分布带入这个公式得到的**。

当p分布是已知，则真实数据的熵是常量；因为Q不影响H(P)。于是交叉熵和KL散度则是等价的。针对Q最小化交叉熵等价于最小化KL散度。

最小化交叉熵 实际上就是**最小化预测分布与实际分布之间的"距离"**。换句话说，交叉熵损失函数"想要"预测分布的所有概率密度都在正确分类上。

也就是说，在输出为概率分布的情况下，就可以使用交叉熵函数作为理想与现实的度量。这也就是为什么它可以作为有 Softmax 函数激活的神经网络的损失函数。

8. 最大似然估计

思想

以交叉熵函数作为损失函数的数学基础来自于统计学中的最大似然估计 Maximum Likelihood Estimation。

最大似然估计是一种参数估计，更确切的说，是一个点估计。

最大似然估计的思想: **使得观测数据（样本）发生概率最大的参数就是最好的参数**。通俗的说就是 —— 最像估计法（最可能估计法），即概率最大的事件，最可能发生。

最大似然估计常用于利用已知的样本结果，反推最有可能导致这一结果产生的参数值，往往模型结果已经确定，用于反推模型中的参数。确定参数值的过程，是找到能 "**最大化模型产生真实观察数据可能性**" 的那一组参数。因为结果已知，如果某一参数能使得结果产生的概率最大，则该参数为最优参数。

举例如下：假如说一个老猎人带着一个小徒弟出去打猎。目前我们知道他们其中一个人打死了一只兔子，假如让你猜这只兔子是谁打死的，大部分人会直接猜测是老猎人打死的，这其实就运用了最大似然的思想。样本就是死兔子。参数就是老猎人。

推导

在最大似然估计的语境中，由于总体分布参数未知，因此用 θ 表示总体的一个或多个未知参数。

假设分布率为 P=p(x;θ)，x是发生的样本，θ是代估计的参数，p(x;θ)表示估计参数为θ时，发生x的的概率。

我们要计算的是同时观察到所有这些数据的概率，也就是所有观测数据点的联合概率分布。由于我们假设样本之间彼此独立且服从参数为 θ 的分布，那么取得任意一个样本 xi 的概率可以表示为 P(xi|θ)，因而当前这个样本集所有数据的总概率就是单独观察到每个数据点的概率的乘积（即边缘概率的乘积）：

```
P(X) = P(x1|θ)P(x2|θ)P(x3|θ) ... P(xn|θ)
```

由于在抽样后会得到总体的一个观测，因此相应的概率在具体的样本集中都是已知的，上面这个函数则可以看作在不同的参数 θ 的取值下，取得当前这个样本集的概率，也即可能性 likelihood，因此将其称为参数 θ 相对于样本集 X 的似然函数 likelihood function，记为L(θ)，即有：

```
L(θ) = L(x1,x2,...,xn;θ) = p(x1|θ)...p(xn|θ)  连乘
```

对于抽样得到样本集 X 这个既成事实，我们可以认为这就是最该发生的一个结果，也即所有可能的结果中概率最大的那一个，这就是最大似然估计的命名由来。此时，我们需要寻找的就是那个使似然函数取得最大值的 θ 值：

```
argmax L(θ)
```

这里 argmax ƒ(x) 是使得函数 ƒ(x) 取得其最大值的所有自变量 x 的集合，而我们想要研究的问题最终再一次变成了一个求极值问题。

对于求一个函数的极值，最直接的设想是求导，然后让导数为0，那么解这个方程得到的θ就是了（当然，前提是函数L(θ)连续可微）。但如果θ是包含多个参数的向量那怎么处理呢？当然是求L(θ)对所有参数的偏导数，也就是梯度了。从而n个未知的参数，就有n个方程，方程组的解就是似然函数的极值点了，最终得到这n个参数的值。

使用

通过两个问题能够让我们知道如何使用极大似然。

- 问：如何体现出我们的估计或者模型在力求最好地拟合抽取的样本？

  答：求似然函数的最大值。

- 问：哪一套参数才是我们要找的最好的参数？

  答：似然函数最大的时候，对应的参数 θ

结合Word2vec的例子来简要通俗的说。

1. 训练模型，然后输出层经过Softmax得到了一个概率分布。
2. 生成一个似然函数 L(w)。这个似然函数一般就是从上面的概率分布(概率数值）构建出来。w就是隐藏矩阵的具体数值，作为似然函数 L 的参数。
3. 为了计算方便，会取对数似然函数，就是 logL(w)。
4. 求这个似然函数 logL(w) 的最大值。得到最大值的时候对应的w就是 L 的最佳参数。
5. 最终得到的w就是我们需要的隐藏矩阵。

9. 最小化交叉熵 vs 最大似然估计

最小化交叉熵和模型采用最大似然估计进行参数估计是一致的, 可以从数学公式做如下推导:。

通过最大似然估计方法使参数为Θ^的模型使预测值贴近真实数据的概率最大化：

$$\hat{\Theta} = arg\,max_{\theta}\Pi_{i=1}^{N} p(x_i|\Theta)$$

实际操作中，连乘很容易出现最大值或最小值溢出，造成计算不稳定，由于log函数的单调性，所以将上式进行取对数取负，最小化负对数似然(NLL)的结果与原始式子是一样的，从而最大化对数似然函数就等效于最小化负对数似然函数。

$$\hat{\Theta} = arg\,min_{\Theta} - \Sigma_{i=1}^{N} log(p(x_i|\Theta))$$

对模型的预测值进行最大似然估计，

$$\begin{aligned}\hat{\Theta} &= arg\,min_{\Theta} - \Sigma_{i=1}^{N} log(q(x_i|\Theta)) \\ &= arg\min_{\Theta} -\Sigma_{x\in X} p(x) log(q(x|\Theta)) \\ &= arg\,min_{\Theta} H(p,q)\end{aligned}$$

所以最小化NLL和最小化交叉熵最后达到的效果是一样的。这也是很多模型又采用最大似然估计作为损失函数的原因。

在机器学习的世界里面大概可以认为交叉熵和最大似然估计是一回事，如果看到这两个术语应该把他们联系在一起。

对于多类分类问题，似然函数就是衡量当前这个以predict为参数的单次观测下的多项式分布模型与样本值label之间的似然度。这是单个样本的似然函数。最后我们需要极大化的是整个样本集（或者一个batch）的似然函数。

可以说交叉熵是直接衡量两个分布，或者说两个model之间的差异。而似然函数则是解释以model的输出为参数的某分布模型对样本集的解释程度。因此，可以说这两者是“同貌不同源”，但是“殊途同归”。

10. 训练

概率模型的训练过程就是参数估计的过程。

机器学习的过程就是希望在训练数据上模型学到的分布 **P(model)** 和真实数据 **P(real)**的分布越接近越好，那么怎么最小化两个分布之间的不同呢？用默认的方法，使其KL散度最小。

但我们没有真实数据的分布，那么只能退而求其次，希望模型学到的分布和训练数据的分布 **P(training)** 尽量相同，也就是把训练数据当做模型和真实数据之间的代理人。

假设训练数据是从总体中独立同步分布采样(Independent and identically distributed sampled)而来，那么我们可以利用最小化训练数据的经验误差来降低模型的泛化误差。

简单说：

- 最终目的是希望学到的模型的分布和真实分布一致: **P(model) = P(real)**
- 但真实分布是不可知的，我们只好假设 训练数据 是从 真实数据 中独立同分布采样而来：**P(training) = P(real)**
- 退而求其次，我们希望学到的模型分布至少和训练数据的分布一致: **P(model) = P(training)**

由此非常理想化的看法是如果**模型(左)**能够学到**训练数据(中)**的分布，那么应该近似的学到了**真实数据(右)**的分布 **P(model) = P(training) = P(real)**。

0x03 输出层相关概念

输出层中主要概念就是Softmax。

Softmax函数一个重要的性质就是把全连接层的输出归一化转换到每一个对应分类的概率。一旦转换为概率之后，我们就可以用到最大似然估计（交叉熵）的方式来求得最大似然或者最小交叉熵。

1. 归一化

为什么要归一化？在每次迭代中把要预测的词相关权重增加，通过归一化，同时把其他的词相关权重减少。这个不难理解，总的预测概率和是1，把其中某一个词的概率增加就意味着把其他词的预测概率打压。能不能只增加其中某个词的概率呢？可以，但是收敛很慢。

2. Softmax回归

Softmax是一种回归模型，在机器学习中经常用Softmax来解决MECE原则的分类——每一个分类相互独立，所有的分类被完全穷尽。比如男人和女人就是负责MECE原则的。

Softmax 函数接收一个N维向量作为输入，然后把每一维的值转换成（0，1）之间的一个实数。我们训练的目标就是让属于第k类的样本经过 Softmax 以后，第 k 类的概率越大越好。这就使得分类问题能更好的用统计学方法去解释了。

max vs soft max

Max

顾名思义 max 取最大值就是绝对能取到最大值，比如有两个数，a和b，并且a > b，如果取max，那么就直接取a，没有第二种可能。

但有的时候我不想这样，因为这样会造成分值小的那个饥饿。所以我希望分值大的那一项经常取到，分值小的那一项也偶尔可以取到，那么我用soft max就可以了 。

Soft max

现在还是a和b，a>b，如果我们取按照softmax来计算取a和b的概率，因为a的softmax值大于b的，所以a会经常取到，而b也会偶尔取到，概率跟它们本来的大小有关。所以说不是max，而是 **Soft** max 。

正如它的名字一样，Softmax 函数是一个“软”的最大值函数，它不是直接取输出的最大值那一类作为分类结果，同时也会考虑到其它相对来说较小的一类的输出。

Softmax得到的概率

那a, b各自的概率（这里的概率是相对概率）究竟是多少呢，我们下面就来具体看一下定义：

假设我们有一个数组 V，我们想获取这个数组的最大值。Vi表示V中的第i个元素，那么这个元素 i 的Softmax值就是

$$Si = \frac{e^{Vi}}{\sum_j e^{Vj}}$$

也就是说，是该元素的指数，与所有元素指数和的比值(取指数是为了使差别更大)。

再简略说就是计算一组数值中每个值的占比。

再通俗的说：绝对max就是只选自己喜欢的男神，Soft max把所有备胎全部来评分，还做了个归一化。

比如：

加入原来的输出是 5, 1, -1， 经过Softmax函数起了作用，就都映射成为(0,1)的值：0.9817, 0.0180, 0.0003，而这些值的累和为1（满足概率的性质），那么我们就可以将它理解成概率，在最后选取输出结点的时候，我们就可以选取概率最大（也就是值对应最大的）结点，作为我们的预测目标！

Softmax的重要性质

总结一下，Softmax分类器会对线性运算输出的score向量进行进一步转化：通过**Softmax函数**将【当前样本对于各个类的分数】转化为【当前样本属于各个类的概率分布】，后者才是模型的真正输出。这也是Softmax分类器名字的由来。

注意：Softmax classifier将自己被输入的score向量看作是一种未归一化的对数概率分布（unnormalized log probabilities）。经过适当的转化以后把输出归一化转换到每一个对应分类的概率。一旦转换为概率之后，我们就可以用到最大似然估计（交叉熵）的方式来求得最大似然或者最小交叉熵。

将unnormalized log probabilities转化回normalized probabilities的方式是：先将每个log probabilities以e为底数做幂运算，转化为普通概率，再把所有类的普通概率做归一化，让它们加起来等于1。

0x04. Word2vec(通俗学习版本)

历经艰辛，终于来到了Word2vec.....

Word2vec通过训练，将独热向量稀疏空间中每个词都映射到一个较短的词向量上来。所有的这些词向量就构成了向量空间，进而可以用普通的统计学的方法来研究词与词之间的关系。即可以把对文本内容的处理简化为向量空间中的向量运算，计算出向量空间上的相似度，来表示文本语义上的相似度。

比如我们将King这个词用"Royalty","Masculinity", "Femininity"和"Age"4个维度来表示，则对应的词向量可能是(0.99,0.99,0.05,0.7)。当然在实际情况中，我们并不能对词向量的每个维度做一个很好的解释。

1. 训练方案

word2vec两个训练方案分别是CBOW和Skip-Gram。

- CBOW是从原始语句推测目标字词；训练输入是某一个特征词的上下文相关的词对应的词向量，而输出就是这特定的一个词的词向量。
- 而Skip-Gram正好相反，是从目标字词推测出原始语句。即输入是特定的一个词的词向量，而输出是特定词对应的上下文词向量。

用通俗的语言来说，就是

- CBOW : “周围词叠加起来预测当前词”（P(wt|Context)）
- Skip-Gram : “当前词分别来预测周围词”（P(wothers|wt)）

比如：对同样一个句子：Hangzhou is a nice city。我们要构造一个语境与目标词汇的映射关系，其实就是input与label的关系。这里假设滑窗尺寸为1。

- CBOW可以制造的映射关系为：[Hangzhou,a]—>is，[is,nice]—>a，[a,city]—>nice
- Skip-Gram可以制造的映射关系为: (is,Hangzhou)，(is,a)，(a,is)，(a,nice)，(nice,a)，(nice,city)

2. 基础架构

word2vec模型其实就是简单化的神经网络。但是这个学习不是为了准确的预估正确的中心词/周围词，而是为了得到 word——>vector 这个映射关系。再次介绍其具体分解如下：

- 构建数据：从原始数据出发构建单词对，单词形式如下 [input word, out word]，即[data x, label y]。

- 输入层: 将所有词语进行one-hot编码作为输入，输入的应该是n维向量(n是单词个数)

- 隐藏层：中间是只有一个隐藏层(没有激活函数，只是线性的单元)。隐藏层实际上存储了词汇表中所有单词的word vectors。这是一个尺寸为 [vocabulary size x embedding size] 的矩阵。矩阵的每一行对应了某一个单词的word vector。

- 输出层：输出的也是独热向量。Output Layer维度跟Input Layer的维度一样，各维的值相加为1。用的是Softmax回归。softmax保证输出的向量是一个概率分布。一旦转换为概率之后，我们就可以用到最大似然估计（交叉熵）的方式来求得最大似然或者最小交叉熵。

- 定义loss损失函数：用来预测正确输出/优化模型。

  我们的 label y 值是一个概率分布，输出层也是一个概率分布，这样就可以用交叉熵来衡量神经网络的输出与我们的 label y 的差异大小，也就可以定义出loss了。

- 迭代训练: 一般使用梯度下降优化算法来最小化代价函数，每次迭代比较 prediction 和 label y 之间的loss，然后相应优化，最终确保类似的单词有类似的向量。

- 最终矩阵：丢去output层，只用隐藏层的输出单元(就是Input Layer和Hidden Layer之间的权重)，构成了Look up table。

当这个模型训练好以后，我们并不会用这个训练好的模型处理新的任务，我们真正需要的是这个模型通过训练数据所学得的参数，例如隐层的权重矩阵。

3. 详细分解

3.1 构建数据

以The quick brown fox jumps over the lazy dog.为例。这句话中共有8个词（这里The与the算同一个词）。

（x,y）就是一个个的单词对。比如（the，quick）就是一个单词对，the就是样本数据，quick就是该条样本的标签。

取被扫描单词左右各2个词，这里的2被称为窗口尺寸，是可以调整的。这样左右各两个词共4个词拿出来，分别与被扫描的单词组成单词对，作为我们的训练数据。当句子头尾的单词被扫描时，其能取的单词对数要少几个。

最后得到训练数据：

```
(the, quick),(the, brown)
(quick,the),(quick,brown),(quick,fox)
(brown,the),(brown,quick),(brown,fox),(brown,jumps)
(fox,quick),(fox,brown),(fox,jumps),(fox,over)
......
```

以 (fox, jumps),(fox, brown) 这两个数据为例看看**这样取单词的目的**：jumps和brown可以理解为fox的**上下文**，我们希望当输入fox时候，神经网络能告诉我们哪个单词更可能出现在fox周围。如何能这样做到，具体就要看(fox, jumps),(fox, brown)两个单词对谁在训练集中出现的次数比较多，神经网络就会针对哪个单词对按照梯度下降进行更多的调整，从而就会倾向于预测谁将出现在fox周围。

3.2 输入层

将单词转为为独热编码的数字形式，这样才能作为神经网络的输入。比如(fox, jumps)可能是[(00000001),(00000010)]，具体取决于编码的结果。这个输入层是n维向量，n是词汇表中单词的个数。

神经网络的输入就是训练数据中的单词对(x,y)的独热编码，模型将会从每对单词出现的次数中习得统计结果。就像上文提到的，当得到更多类似(fox, jumps)这样的训练样本对，而对(fox, brown)这样的组合却看得很少，那么因此，当我们的模型完成训练后，给定一个单词”fox“作为输入，输出的结果中”jumps“要比”brown“被赋予更高的概率。

3.3 隐藏层

隐藏层实际上存储了词汇表中所有单词的word vectors。这是一个尺寸为 [vocabulary size x embedding size] 的矩阵。矩阵的每一行对应了某一个单词的word vector。

隐藏层的神经元应该是多少？这取决于我们希望得到的词向量是多少维，有多少个隐藏神经元词向量就是多少维。比如词汇表中一共有8个单词，那么每一个隐藏的神经元接收的输入都是一个8维向量，假设我们的隐藏神经元有3个，如此以来，隐藏层的权重就可以用一个8行3列的矩阵来表示。

3.4 交叉熵(为了学习原理，下面为参考代码)

实例参考

以TensorFlow的交叉熵函数 tf.nn.softmax_cross_entropy_with_logits(logits, labels, name=None)作为参考。

- 参数logits：就是神经网络最后一层的输出(W * X 矩阵)，如果有batch的话，它的大小就是[batchsize，num_classes]，单样本的话，大小就是num_classes。
- 参数labels：实际的标签，大小同上。

具体的执行流程大概分为两步：

- 先对网络最后一层的输出做一个softmax，这一步通常是求取输出属于某一类的概率，对于单样本而言，输出就是一个num_classes大小的向量（[Y1，Y2,Y3...]其中Y1，Y2，Y3...分别代表了是属于该类的概率）。
- softmax的输出向量[Y1，Y2,Y3...]和样本的实际标签做一个交叉熵。

关于**logits**的进一步说明：

说到Logits，首先要弄明白什么是**Odds**？在英文里，Odds的本意是指**几率、可能性**。

$$Odds\ （\ A\ ）= \text{发生事件 } A \text{ 次数} \ / \ \text{其他事件的次数（即不发生 } A \text{ 的次数）}$$

换句话说，**事件A的Odds** 等于 **事件A出现的次数** 和 **其它（非A）事件出现的次数** 之比；

相比之下，**事件A的概率** 等于 **事件A出现的次数** 与 **所有事件的次数** 之比。

很容易推导得知：概率 P（A）和Odds（A）的值域是不同的。前者被锁定在[0,1]之间，而后者则是[0, 正无穷大]

这说了半天，有何logit有什么关系呢？

请注意Logit一词的分解，对它（it）Log（取对数），这里“it”就是Odds。下面我们就可以给出Logit的定义了：

$$Logit(Odds) = log(\frac{P}{1-P})$$

这个公式实际上就是所谓**Logit变换**。

与概率不同的地方在于，Logit的一个很重要的特性，**就是它没有上下限**，这就给建模提供了方便。让回归拟合变得容易了！

通常，Logit对数的底是自然对象e，这里我们把Odds用**符号θ表示**，则有：

$$\Theta = ln(\frac{P}{1-P})$$

显然，知道θ后，我们也容易推e导出概率的值来：

$$p = \frac{e^{\Theta}}{1+e^{\Theta}}$$

于是，我们先借助Logit变换，让我们方便拟合数据（即逻辑回归），然后再变换回我们熟悉的概率。就是这么一个循环，为数据分析提供了便利。某种程度上，这类变换，非常类似于化学中的**催化剂**。

应用

经过softmax的加工，就变成“归一化”的概率（设为p1），这个新生成的概率p1，和labels所代表的概率分布（设为p2）一起作为参数，用来计算交叉熵。

这个差异信息，作为我们网络调参的依据，理想情况下，这两个分布尽量趋近最好。如果有差异（也可以理解为误差信号），我们就调整参数，让其变得更小，这就是损失（误差）函数的作用。

最终通过不断地调参，logit被锁定在一个最佳值（所谓最佳，是指让交叉熵最小，此时的网络参数也是最优的）。

3.5 训练

我们采用**最大似然**来构造出来了一个目标函数(损失函数)，在目标函数构建完成后还需要一种优化方法以便求出其中的参数。随机梯度下降（SGD）是用于极小化损失函数 f(W,b)（即最大化似然函数）的默认标准方法，可用于找到深度学习的权重和偏置。

梯度下降的目的，直白地说：就是减小真实值和预测值的距离，而损失函数用来度量真实值和预测值之间距离，所以梯度下降目的也就是减小损失函数的值。怎么减小损失函数的值呢？

我们要做的就是不断修改隐藏层的值以使损失函数越来越小。

SGD 在第 k 次迭代更新时只是简单地通过减去梯度f (W k, b k) 的估计值来极小化损失函数。

$$参数=参数 - 学习率 \times 损失函数对参数的偏导$$

3.6 输出层

神经网络基于这些训练数据将会输出一个概率分布，这个概率代表着我们的词典中的每个词是output word的可能性。输出层的神经元数量和语料库中的单词数量一样。每一个神经元可以认为对应一个单词的输出权重，词向量乘以该输出权重就得到一个数，该数字代表了输出神经元对应的单词出现在输入单词周围的可能性大小，通过对所有的输出层神经元的输出进行softmax操作，我们就把输出层的输出规整为一个概率分布了。

softmax直白来说就是将原来输出比如是3,1,-3通过softmax函数一作用，就映射成为(0,1)的值，而这些值的累和为1（满足概率的性质），那么我们就可以将它理解成概率。

这里有一点需要注意，我们说输出的是该单词出现在输入单词周围的概率大小，这个“周围”包含单词的前面，也包含单词的后面。

比如我们用水浒传来训练：

我们输入一个单词"宋公明"，那么那么最终模型的输出概率中，像“哥哥”， ”及时雨“这种相关词的概率将远高于像”这厮“，”武艺“非相关词的概率。因为”哥哥“，”及时雨“在文本中更大可能在”宋公明“的窗口中出现。

3.7 最终结果

最终我们需要的是训练出来的权重矩阵W，而不是那些输出层的概率数值。输入层的每个单词与矩阵W相乘得到的向量的就是我们想要的词向量（word embedding）。

这个矩阵（所有单词的word embedding）也叫做look up table，也就是说，任何一个单词的one hot乘以这个矩阵都将得到自己的词向量。有了look up table就可以免去训练过程直接查表得到单词的词向量了。

比如，假设原始数据是：

The brown fox jumps.

独热编码结果是：

```
1 The      [1,0,0,0]
```

```
brown    [0,1,0,0]
fox      [0,0,1,0]
jumps    [0,0,0,1]
```

最后得到的隐藏层参数矩阵是

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \\ 10 & 11 & 12 \end{bmatrix}$$

这个矩阵就存储了所有word vector，下面就是每一个单词实际对应的word vector

```
The      [1, 2, 3]
brown    [4, 5, 6]
fox      [7, 8, 9]
jumps    [10，11，12]
```

应用时，如果输入单词 fox

$$[0 \quad 0 \quad 1 \quad 0]$$

其最终的词向量是：

$$[7 \quad 8 \quad 9]$$

计算过程是：

$$[0 \quad 0 \quad 1 \quad 0] \times \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \\ 10 & 11 & 12 \end{bmatrix} = [7 \quad 8 \quad 9]$$

3.8 使用

得到了了稠密词向量之后，就可以做相关计算了。

比如用夹角的余弦值表示词意思之间的距离，距离范围为0-1之间，余弦值越大，两个词之间的意思越接近。

0x05 Word2vec(实际生产版本)

word2vec作为神经概率语言模型的输入，其本身其实是神经概率模型的副产品，是为了通过神经网络学习某个语言模型而产生的中间结果。

具体来说，“某个语言模型”指的是“CBOW”和“Skip-gram”。学习过程会用到两个降低复杂度的近似方法——Hierarchical Softmax或Negative Sampling。两个模型乘以两种方法，一共有四种实现。

1. 基本结构

按照原理来说，基本网络结构应该是四层：分别是**输入层，映射层，隐藏层 和 输出层**。

但是实际中，无论是哪种模型，其基本网络结构都是省略掉hidden layer。为什么要去掉这一层呢？据说是因为word2vec的作者嫌从hidden layer到output layer的矩阵运算太多了。

所以最终都是总共有三层，分别是**输入层，映射层 和 输出层**。

2. CBOW

CBOW是已知上下文，估算当前词语的语言模型。

2.1 似然函数

以huffman softmax为例，计算上下文向量到中心词的概率，是一连串的二分类问题，因为从根节点到中心词对应的叶子节点，需要多次决定沿左节点还是右节点到叶子节点。每个叶子节点代表语料库中的一个词语，于是每个词语都可以被01唯一地编码，并且其编码序列对应一个事件序列。假如w表示语料库C中任意一个词，于是我们可以计算 w 的条件概率：

$$p(w|Context(w))$$

然后，对于中心词w，从根节点到中心词节点的总概率就是这些节点概率连乘。

最终得到其学习目标，即最大化对数似然函数如下：

$$L = \sum_{w \in C} logp(w|Context(w))$$

2.2 基本结构

输入层是上下文的词语的词向量。

hs模式和negative模式中，输入层到映射层的处理是一样的，仅仅是映射层到输出层的处理不一致。输入层到映射层的具体操作是：**将上下文窗口中的每个词向量求和，然后再平均，得到一个和词向量一样维度的向量，假设叫上下文向量，这个向量就是映射层的向量。**

输出层输出最可能的w。

由于语料库中词汇量是固定的|C|个，所以上述过程其实可以看做一个多分类问题。给定特征，从|C|个分类中挑一个。

3. Skip-gram

Skip-gram只是逆转了CBOW的因果关系而已，即已知当前词语，预测上下文。

与CBOW的两个不同在于：

- 输入层不再是多个词向量，而是一个词向量
- 投影层其实什么事情都没干，直接将输入层的词向量传递给输出层

在Hierarchical Softmax思想下，u 表示w的上下文中的一个词语。每个u 都可以编码为一条01路径，把它们写到一起，得到目标函数。

于是语言模型的概率函数可以写作：

$$P(Context(w)|w) = \Pi_{u \in Context(w)} p(u|w)$$

注意这是一个词袋模型，所以每个u是无序的，或者说，互相独立的。

最终得到其学习目标，即最大化对数似然函数如下：

$$L = \sum_{u \in Context(w)} logp(u|w)$$

0x06 参考

通俗理解word2vec

机器学习必须熟悉的算法之word2vector（一）

Word2vec神经网络详细分析——TrainModelThread分析

word2vec模型训练保存加载及简单使用

贺完结！CS231n官方笔记授权翻译总集篇发布

https://www.zhihu.com/question/23765351

机器学习——softmax计算

softmax与多分类

交叉熵与最大似然估计

理解交叉熵和最大似然估计的关系

极大似然估计与交叉熵

最大似然估计 (Maximum Likelihood Estimation), 交叉熵 (Cross Entropy) 与深度神经网络

Why You Should Use Cross-Entropy Error Instead Of Classification Error Or Mean Squared Error For Neural Network Classifier Training

如何理解KL散度的不对称性？

（转）KL散度的理解

完美解释交叉熵

深度学习中交叉熵和KL散度和最大似然估计之间的关系

cs231n notesufldl tutorialcs231n 翻译

信息熵

https://github.com/thushv89/nlp_examples_thushv_dot_com/blob/master/kl_divergence.ipynb

https://www.countbayesie.com/blog/2017/5/9/kullback-leibler-divergence-explained

哈？你还认为似然函数跟交叉熵是一个意思呀？

Light on Math Machine Learning: Intuitive Guide to Understanding Word2vec

为什么是SoftMax？

原理就是这么简单 Softmax 分类

知乎：为什么交叉熵（cross-entropy）可以用于计算代价？

为什么用交叉熵做损失函数

为什么交叉熵（cross-entropy）可以用于计算代价？

DeepLearning学习笔记——极大似然估计

通过理解极大似然估计(MLE)提升对机器学习算法的认知

从最大似然估计开始，你需要打下的机器学习基石

TensorFlow四种Cross Entropy算法实现和应用

word2vec源码详解

word2vec原理推导与代码分析

机器学习算法实现解析——word2vec源码解析

神奇的Embedding