Lecture 2: Word Vectors and Word Senses

Optimization: Gradient Descent

在lecture 1中，目标函数已经给出，我们需要最小化目标函数，而gradient descent是一个能够最小化目标函数的算法。

Idea：对于现在的θ，计算$J(\theta)$的梯度（在lecture 1的笔记中有推导过程），然后在负梯度的方向上走一小步，重复这个步骤直至$J(\theta)$达到最小。

更新的公式：

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

算法：

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

Stochastic Gradient Descent

Problem:

- $J(\theta)$语料库中所有窗口的函数，而这个语料库可能是十亿级别的，所以$\nabla_\theta J(\theta)$的计算是十分expensive的。
- 每次更新θ都将耗费很长的时间。
- 对于几乎所有神经网络来说这都是很坏的情况。

Solution:

- 对窗口进行重复采样，然后对每一组样本进行更新（batch）
- 算法：

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

Stochastic gradients with word vectors

采用SGD的方式进行更新，会使得$\nabla_\theta J_t(\theta)$十分稀疏，我们可能只更新了确实出现了的那些词向量。

Solution:

- 需要稀疏矩阵更新操作来只更新矩阵U和V中的特定行。

- 需要保留单词向量的散列。

如果你有百万级别的词向量且使用并行计算，则不用到处发送巨大的更新是十分重要的。

Word2vec: More details

为什么在前面的推导中对每一个词都适用两个向量进行表示（中心词情况和上下文情况）？

Answer: 只是为了优化上更简单，最后会对二者做平均，当然也可以只使用一个向量表示一个词。

两个模型变种：

1. Skip-grams (SG):
   在给定中心词的情况下预测上下文。
2. Continuous Bag of Words (CBOW)
   在给定上下文的情况下预测中心词。

目前仅采用了naive softmax的方式去训练模型，虽然简单但也十分expensive，所以这里提供了更加高效的训练方式：**Negative Sampling（负采样）**

The skip-gram model with negative sampling

$P(o|c) = \frac{exp(u_o^T v_c)}{\sum_{w \in V} exo(u_w^T v_c)}$的分母计算十分expensive。因此，在标准的word2vec和HW2中都是通过**负采样**的方式实现skip-gram模型。

Main idea:

使用真实词对（中心词和窗口内的上下文词）和噪声词对（中心词与一个随机词进行配对）作为数据集训练一个二分类的逻辑回归模型。

此处插入skip-gram模型和negative sampling的内容

Lecture Notes: Part I Word Vectors I: Introduction, SVD and Word2Vec

> **4.3 Skip-Gram Model**
>
> 相比于CBOW方法，另一个方法是构建一个模型的输入是中心词，然后这个模型会预测或着生成上下文的词。
>
> Skip-gram模型的设定与CBOW大致相同，知识我们交换了x和y（输入和输出进行了交换）。输入是中心词的one-hot向量，表示为x，输出向量表示为$y^{(j)}$，矩阵V和U与CBOW一致。
>
> 6个步骤：
>
> 1. 生成中心词的one-hot输入向量，$x \in \mathbb{R}^{|V|}$
> 2. 获取中心词x的嵌入词向量（embedded word vector）$v_c = Vx$
> 3. 生成一个分值向量（score vector）$z = Uv_c$

4. 将分值向量转化成概率，$\hat{y} = softmax(z)$。注意$\hat{y}_{c-m}, \ldots, \hat{y}_{c-1}, \hat{y}_{c+1}, \ldots, \hat{y}_{c+m}$是观测到的每个上下文词的概率。
5. 我们希望生成的概率向量匹配真实的概率分布$y_{c-m}, \ldots, y_{c-1}, y_{c+1}, \ldots, y_{c+m}$（真实输出的one-hot表示）

Notation for Skip-Gram Model:

- w_i: Word i from vocabulary V
- $\mathcal{V} \in \mathbb{R}^{n \times |V|}$: Input word matrix
- v_i: i-th column of $\mathcal{V}$, the input vector representation of word w_i
- $\mathcal{U} \in \mathbb{R}^{n \times |V|}$: Output word matrix
- u_i: i-th row of $\mathcal{U}$, the output vector representation of word w_i

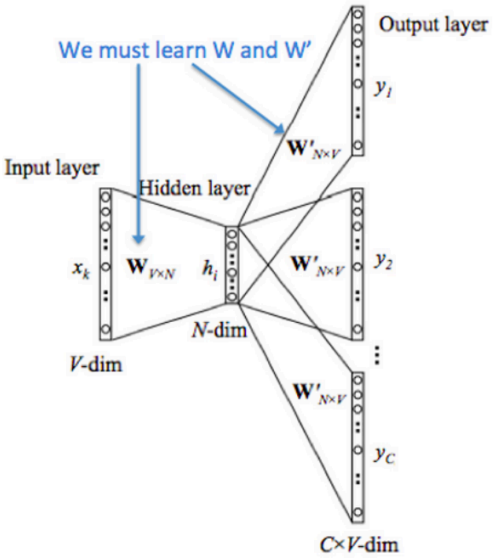


Figure 2: This image demonstrates how Skip-Gram works and how we must learn the transfer matrices

在CBOW中，我们需要生成一个目标函数去评估模型。这里一个关键的不同是我们使用朴素贝叶斯假设去摆脱概率。这是一个很强的条件独立假设，换言之，在给定中心词的情况下，所有输出词是完全独立的。

$$
\begin{aligned}
minimize J &= -\log P(w_{c-m}, \ldots, w_{c-1}, w_{c+1}, \ldots, w_{c+m} | w_c) \\
&= -\log \prod_{j=0, j \neq m}^{2m} P(w_{c-m+j} | w_c) \\
&= -\log \prod_{j=0, j \neq m}^{2m} P(u_{c-m+j} | v_c) \\
&= -\log \prod_{j=0, j \neq m}^{2m} \frac{exp(u_{c-m+j}^T v_c)}{\sum_{k=1}^{|V|} exp(u_k^T v_c)} \\
&= -\sum_{j=0, j \neq m}^{2m} u_{c-m+j}^T v_c + 2m \log \sum_{k=1}^{|V|} exp(u_k^T v_c)
\end{aligned}
$$

有了这个目标函数，我们可以对于未知参数求梯度并且在每个循环中使用SGD更新它们。

注意：

$$J = -\sum_{j=0,j\neq m}^{2m} \log P(u_{c-m+j}|v_c)$$
$$= \sum_{j=0,j\neq m}^{2m} H(\hat{y}, y_{c-m+j})$$

其中$H(\hat{y}, y_{c-m+j})$是概率向量$\hat{y}$与one-hot向量y_{c-m+j}的交叉熵（cross-entropy）

4.4 Negative Sampling

在目标函数中，我们注意到是$|V|$量级求和，计算量十分大。对于任何一次更新，我们计算目标函数都需要$O(|V|)$的时间复杂度。一个简单的想法就是我们可以去近似它，从而以更高的效率进行训练。

对于训练过程中的每一个step，我们可以只采样几个负样本而不是循环整个词汇表。我们从一个噪声分布($P_n(w)$)中采样，并且这个分布的概率与词汇表中的频率顺序相匹配（应该是按照词汇表中出现的频率排序与概率分布中每个词的概率排序一致）。我们需要对方程进行以下方面的更新，以融合负采样的部分：

- 目标函数
- 梯度
- 更新规则

Mikolov et al. present **Negative Sampling** in Distributed Representations of Words and Phrases and their Compositionality.

因为负采样是基于skip-gram模型的，所以它其实是在优化一个不同的目标函数。考虑一个中心词和上下文的词对(w, c)。这个词对是否来自于训练集？让我们用$P(D = 1|w, c)$来表示(w, c)来自于语料库数据。对应的$P(D = 0|w, c)$表示(w, c)不是自于语料库数据。首先，让我们用sigmoid函数对$P(D = 1|w, c)$建模：

$$P(D = 1|w, c, \theta) = \sigma(v_c^T v_w) = \frac{1}{1 + e^(- v_c^T v_w)}$$

现在我们构建一个新的目标函数，尝试最大化$P(D = 1|w, c)$当(w, c)真的在语料库中，和最大化$P(D = 0|w, c)$当(w, c)真的不在语料库中。我们采用一个简单的最大化似然函数的方法来实现。（这里我们采用θ作为模型参数，在我们的情形中是V和U。）

$$\begin{aligned}
\theta &= \arg\max_\theta \prod_{(w,c)\in D} P(D = 1|w, c, \theta) \prod_{(w,c)\in \tilde{D}} P(D = 0|w, c, \theta) \\
&= \arg\max_\theta \prod_{(w,c)\in D} P(D = 1|w, c, \theta) \prod_{(w,c)\in \tilde{D}} (1 - P(D = 1|w, c, \theta)) \\
&= \arg\max_\theta \sum_{(w,c)\in D} \log P(D = 1|w, c, \theta) + \sum_{(w,c)\in \tilde{D}} \log(1 - P(D = 1|w, c, \theta)) \\
&= \arg\max_\theta \sum_{(w,c)\in D} \log \frac{1}{1 + exp(-u_w^T v_c)} + \sum_{(w,c)\in \tilde{D}} \log(1 - \frac{1}{1 + exp(-u_w^T v_c)}) \\
&= \arg\max_\theta \sum_{(w,c)\in D} \log \frac{1}{1 + exp(-u_w^T v_c)} + \sum_{(w,c)\in \tilde{D}} \log(\frac{1}{1 + exp(u_w^T v_c)})
\end{aligned}$$

最大化似然函数等同于最小化负对数似然函数：

$$J = -\sum_{(w,c)\in D} \log \frac{1}{1+exp(-u_w^T v_c)} - \sum_{(w,c)\in \tilde{D}} \log(\frac{1}{1+exp(u_w^T v_c)})$$

$\tilde{D}$是一个“假的”或者“负的”语料库。在这个语料库中会有一些句子像 "stock boil fish is toy"。非自然的语句应该获得一个较小的发生概率。我们可以通过随机采样的方式生成$\tilde{D}$

在skip-gram，对于给定中心词c下，观测到的上下文词$c-m+j$ 新的目标函数为：

$$-\log \sigma(u_{c-m+j}^T v_c) - \sum_{k=1}^{K} \log \sigma(-\tilde{u}_k^T v_c)$$
$$\{\tilde{u}_k | k = 1, \ldots, K\} \backsim P_n(w)$$

目标函数的求和计算从$O(|V|)$ 下降为$O(K)$

且$P_n(w) = \frac{U(w)^{\frac{3}{4}}}{Z}$其中$U(w)$是unigram分布，$3/4$次方使得原本小概率的事件会增大很多，而原本大概率事件则增大很少，例如：

$$is : 0.9^{3/4} = 0.92$$
$$Constitution : 0.09^{3/4} = 0.16$$
$$bombastic : 0.01^{3/4} = 0.032$$

"bombastic"直接增大了三倍，而"is"只增大了一点点。

Why not capture co-occurrence counts directly?

Problems:

- size随着词汇表的增大而增大
- 维度非常高，需要很多存储空间
- 随后的分类模型会遇到“稀疏”的问题
- 模型不够鲁棒

Solution:

Idea: 把最重要的信息存储到一个固定的且较低维度的稠密向量中。

1. 采用SVD(奇异值分解)对矩阵X进行降维。
   奇异值分解：$X = U\Sigma V^T$，其中U, V是正交的。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,VU,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。
2. Hacks to X
   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）
     1）使用log进行缩放
     2）$min(X, t), t \approx 100$
     3）直接全部忽视
   - 在基于窗口的计数中，提高更加接近的单词的计数
   - 使用Person相关系数，然后设置负值为0

Count based vs. Direct prediction

Count based

代表：LSA, HAL, COALS, Hellinger-PCA

优点：

1. 训练速度快
2. 统计数据利用高效

缺点：

1. 主要用于捕捉单词的相似性
2. 对大量数据给予比例失调的重视

Direct prediction

代表：Skip-gram, CBOW, NNLM, HLBL, RNN

优点：

1. 提高其他任务的表现
2. 可以捕捉除了单词相似性意外的复杂模式

缺点：

1. 与语料库大小有关
2. 统计数据使用比较低效（采样是对统计数据的低效使用）

论文：Encoding meaning in vector differences（Glove）

将两个流派的想法结合起来，在神经网络中使用计数矩阵

此处根据Lecture Notes: Part II Word Vectors II: GloVe, Evaluation and Training

1. Comparison with Previous Methods（与其他方法的对比）
   目前来说，我们有两类主流的方式寻找单词嵌入（即词向量）。第一类是基于计数方式和矩阵分解的。虽然这类方法有效的利用了全局统计信息，但主要被用来捕捉单词相似性并且在单词类比、指示最优的字向量空间结构上效果比较差。另一类的方法是基于浅层窗口的（例如skip-gram和CBOW）。它们通过对局部上下文窗口进行预测从而学习到单词嵌入。这些方法展现了获取超越单词相似性的复杂语言模式的能力，但是无法利用上全局co-occurrence统计信息。
   对比来看，GloVe包含了一个加权最小二乘模型，用全局单词间的co-occurrence计数进行训练，因此更有效地利用统计量。模型生成一个具有意义的子结构的词向量空间。在单词类比任务上展现了超前的表现，并且在单词相似性任务上远超现有的所有方法。
2. Co-occurrence Matrix
   X表示word-word co-occurrence矩阵，X_{ij}表示单词j出现在单词i的上下文的次数。令$X_i = \sum_k X_{ik}$表示出现在单词i的上下文的单词总数。最后，令$P_{ij} = P(w_j|w_i) = \frac{X_{ij}}{X_i}$表示单词j出现在单词i的上下文的概率。
   计算这个剧震需要遍历一遍整个语料库并且收集统计量。对于庞大的语料库，一次遍历会十分昂贵，但是也只需要一次遍历。

3. Least Squares Objective（最小二乘目标）
   回忆skip-gram模型，我们使用softmax函数去计算单词j出现在单词i的上下文的概率：

$$Q_{ij} = \frac{exp(u_j^T v_i)}{\sum_{w=1}^{W} exp(u_w^T v_i)}$$

   训练时以在线随机的方式进行，但是隐含的全局交叉熵损失可以如下计算：

$$J = -\sum_{i \in corpus} \sum_{j \in context(i)} \log Q_{ij}$$

   因为同一个单词i和j可以在语料库中出现很多次，所以首先将i和j相同的值组合起来更有效：

$$J = -\sum_{i=1}^{W} \sum_{j=1}^{W} X_{ij} \log Q_{ij}$$

   其中co-occurring频率由矩阵X给出。交叉熵损失的一个显著缺点是它需要分布Q被正确归一化，归一化的过程中包含了十分昂贵的对整个词汇表进行求和的操作。作为替代，我们用最小二乘的目标，从而舍弃了对P和Q的归一化因子：

$$\hat{J} = \sum_{i=1}^{W} \sum_{j=1}^{W} X_i (\hat{P}_{ij} - \hat{Q}_{ij})^2$$

$$\hat{P}_{ij} = X_{ij} \; and \; \hat{Q}_{ij} = exp(u_j^T v_i)$$

   $\hat{P}$和$\hat{Q}$是未归一化分布。上面这个式子引出了一个新的问题，$-X_{ij}$经常在很大的值中选择，并且使得优化变得困难。一个有效的修改是最小化对数平方误差：

$$\begin{aligned} \hat{J} &= \sum_{i=1}^{W} \sum_{j=1}^{W} X_i (\log \hat{P}_{ij} - \log \hat{Q}_{ij})^2 \\ &= \sum_{i=1}^{W} \sum_{j=1}^{W} X_i (u_j^T v_i - \log X_{ij})^2 \end{aligned}$$

   另一个现象是X_i的权值因子不保证是最优的。所以我们引入一个更加一般的权值函数，使得我们能够自由地依赖于上下文单词：

$$\hat{J} = \sum_{i=1}^{W} \sum_{j=1}^{W} f(X_{ij}) (u_j^T v_i - \log X_{ij})^2$$

4.

4. Conclusion
   总结来说，GloVe 模型仅对单词共现矩阵中的非零元素训练，从而有效地利用全局统计信息，并生成具有有意义的子结构向量空间。给出相同的语料库，词汇，窗口大小和训练时间，它的表现都优于 word2vec，它可以更快地实现更好的效果，并且无论速度如何，都能获得最佳效果。

Evaluation of Word Vectors

此处根据Lecture Notes: Part II Word Vectors II: GloVe, Evaluation and Training

1. Intrinsic Evaluation
   词向量的Intrinsic evaluation是对由基于特定中间子任务进行嵌入操作生成的一组词向量的评估。这些子任务是简单且能快速计算的，所以我们可以借助这些任务更好地理解用于生成词向量的系统。一个intrinsic evaluation应该返回一个能够表示这些词向量在子任务表现水平的值。
   动机：让我们考虑一个例子，我们的最终目标是生成一个回答问题的系统，将词向量作为输入。一个实现的方式是训练一个机器学习系统能够满足：
   1）将单词作为输入
   2）将单词转化成词向量
   3）使用词向量作为机器学习系统的输入
   4）将系统输出的词向量映射回自然语言单词
   5）产生单词作为答案
   当然，在制造这个先进回答问题系统的过程中，我们需要创建一个最优的词向量表示，因为它们在下游任务中会被应用。为了在现实中做到，我们需要调整Word2vec子系统中许多的超参数（例如词向量表示的维度）。同时一个理想的方式是在Word2vec子系统由任何参数变化时都重新训练整个系统，这在工程角度时不现实的，因为机器学习系统是一个有百万个参数且需要很长时间进行训练的深度神经网络。这种情况下，我们希望想出一个简单的intrinsic evaluation技术，能够衡量单词转换成词向量好坏。显然一个要求就是这个intrinsic evaluation与最终任务的表现有一个正相关性。

2. Extrinsic Evaluation
   词向量的Extrinsic Evaluation是用真实任务去评估嵌入技术生成的一组词向量。这些任务是精细的且无法快速计算的。用上面的例子，一个系统允许使用回答作为评估是一种Extrinsic Evaluation系统。典型的，对一个在Extrinsic Evaluation中表现欠佳的系统进行优化，不会让我们决定哪个特定子系统出问题，所以这激发了我们对intrinsic Evaluation的需要。
   Extrinsic Evaluation：
   - 是否是在真实任务上进行评估
   - 计算性能的时候是否会很慢
   - 不清楚子系统是否存在问题，其他子系统或者内部互动
   - 如果更换子系统能够提高表现，那么这个替换像是好的

3. Intrinsic Evaluation Example: Word Vector Analogies
   一个被广泛选择作为词向量intrinsic evaluation的是在完成词向量类比的表现。在一个词向量类比任务重，我们使用下面这种形式提供一个不完整的类比：

   $$a : b :: c :?$$

   这时 intrinsic evaluation系统然后识别使得余弦相似度最大的词向量：

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

   这个衡量标准有一个很直观的解释。最理想的情况是$x_b - x_a = x_d - x_c$（例如，queen - king = actress - actor）。这告诉我们$x_b - x_a + x_c = x_d$。因此我们通过确定最大化两个词向量的标准化点乘来识别向量x_d（即余弦相似度）
   使用例如词向量类比的intrinsic evaluation技巧需要小心（记住多方向的语料库作为预训练）。例如，考虑这种形式的类比：

   City 1: State containing City 1 :: City 2: State Containing City 2

   例如：

Chicago: Illinois :: Houston → Texas
Chicago : Illinois : : Philadelphia → Pennsylvania
Chicago : Illinois : : Phoenix → Arizona
Chicago : Illinois : : Dallas → Texas
Chicago : Illinois : : Jacksonville → Florida
Chicago : Illinois : : Indianapolis → Indiana
Chicago : Illinois : : Austin → Texas
Chicago : Illinois : : Detroit → Michigan
Chicago : Illinois : : Memphis → Tennessee
Chicago : Illinois : : Boston → Massachusetts

上面的例子中，有很多重复名字的城市、镇、村。因此许多州会被视为是正确答案。例如美国最少有10个地方被叫做Phoenix，所以Arizona不是唯一的正确答案。现在让我们考虑另一个种形式的类比：

Capital City 1: Country 1 :: Capital City 2: Country 2

Abuja : Nigeria : : Accra → Ghana
Abuja : Nigeria : : Algiers → Algeria
Abuja : Nigeria : : Amman → Jordan
Abuja : Nigeria : : Ankara → Turkey
Abuja : Nigeria : : Antananarivo → Madagascar
Abuja : Nigeria : : Apia → Samoa
Abuja : Nigeria : : Ashgabat → Turkmenistan
Abuja : Nigeria : : Asmara → Eritrea
Abuja : Nigeria : : Astana → Kazakhstan
这个例子中，由于语料库是滞后的，所以预测出来的首都可能只是过去的首都而不是现在的首都。
上面两个例子描述了用词向量所做的语义测试。我们也可以使用词向量类比测试语法。下面的intrinsic evaluation测试了词向量捕捉最高级形容词的概念的能力：
bad : worst : : big → biggest
bad : worst : : bright → brightest
bad : worst : : cold → coldest
bad : worst : : cool → coolest
bad : worst : : dark → darkest
bad : worst : : easy → easiest
bad : worst : : fast → fastest
bad : worst : : good → best
bad : worst : : great → greatest

类似的，我们做了过去式的类比：
dancing : danced : : decreasing → decreased
dancing : danced : : describing → described
dancing : danced : : enhancing → enhanced
dancing : danced : : falling → fell
dancing : danced : : feeding → fed
dancing : danced : : flying → flew
dancing : danced : : generating → generated
dancing : danced : : going → went
dancing : danced : : hiding → hid
dancing : danced : : hitting → hit

4. Intrinsic Evaluation Tuning Example: Analogy Evaluations

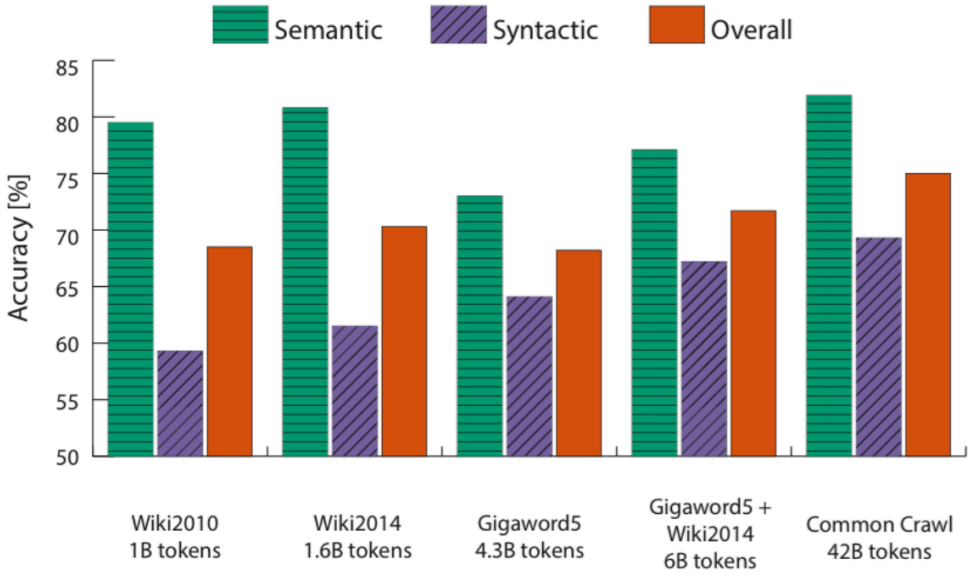
现在，我们探索一些能够使用intrinsic evaluation系统进行调整的词向量嵌入技巧中的超参数。首先让我们看一下不同的生成词向量的方法基于相同超参数在类比任务中的表现：

Model	Dimension	Size	Semantics	Syntax	Total
ivLBL	100	1.5B	55.9	50.1	53.2
HPCA	100	1.6B	4.2	16.4	10.8
GloVE	100	1.6B	67.5	54.3	60.3
SG	300	1B	61	61	61
CBOW	300	1.6B	16.1	52.6	36.1
vLBL	300	1.5B	54.2	64.8	60.0
ivLBL	300	1.5B	65.2	63.0	64.0
GloVe	300	1.6B	80.8	61.5	70.3
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	67.4	65.7
SG	300	6B	73.0	66.0	69.1
GloVe	300	6B	77.4	67.0	71.7
CBOW	1000	6B	57.3	68.9	63.7
SG	1000	6B	66.1	65.1	65.6
SVD-L	300	42B	38.4	58.2	49.2
GloVe	300	42B	81.9	69.3	75.0

Table 5: Here we compare the performance of different models under the use of different hyperparameters and datasets

基于上表，我们得到三个主要结论：

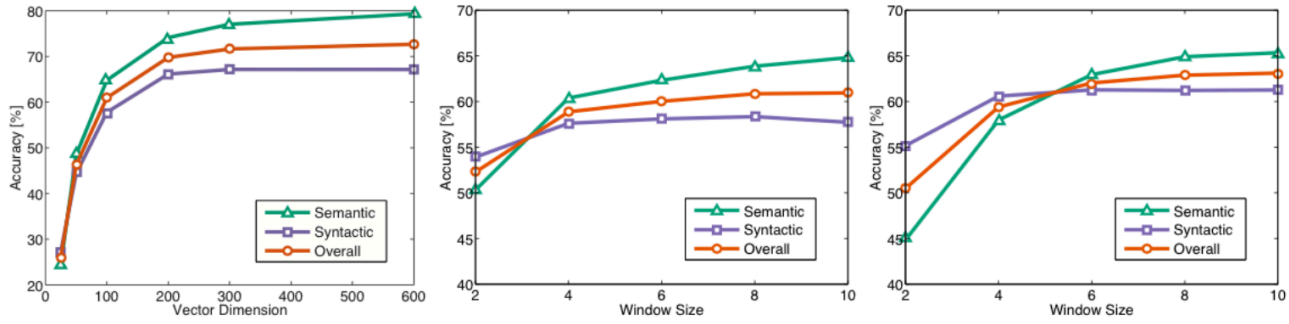
- **表现严重依赖于用于词嵌入的模型**：
  这是一个期望中的结果，因为不同的方法用完全不同的属性去尝试将单词转化成向量。（例如co-occurrence计数，奇异向量等）
- **表现随着语料库的增大而增强：**
  发生的原因是一个嵌入方式在观察更多样本时能够获取更多的经验。例如，如果这个模型之前没有遇到过类比的样本，那么该模型在类比任务中会产生错误的结果。



- **在极小的词向量维度情况下，表现会更差：**
  更低维度的词向量无法做到捕捉语料库中不同单词的不同含义。这可以看做是一个高偏差问题，我们模

型的复杂度很低。例如，让我们考虑单词“king”，“queen”，“man“，”women“。直觉上，我们会需要用两个维度，例如”性别“和”领导力“去将单词编码成2-bit的词向量。任何更小的维度会失去捕捉这四个单词语义差别的能力。

下图表示草参数对GloVe表现的影响：



(a) Symmetric context (b) Symmetric context (c) Asymmetric context

5. Intrinsic Evaluation Example: Correlation Evaluation
   另一个简单的衡量词向量质量的方式是让人用一个固定的尺度（0-10）给两个单词的相似度打分，然后跟余弦相似度对比。在许多个数据集上都进行了这个过程，包含了人类评价调查数据。

Model	Size	WS353	MC	RG	SCWS	RW
SVD	6B	35.3	35.1	42.5	38.3	25.6
SVD-S	6B	56.5	71.5	71.0	53.6	34.7
SVD-L	6B	65.7	72.7	75.1	56.5	37.0
CBOW	6B	57.2	65.6	68.2	57.0	32.5
SG	6B	62.8	65.2	69.7	58.1	37.2
GloVe	6B	65.8	72.7	77.8	53.9	38.1
SVD-L	42B	74.0	76.4	74.1	58.3	39.9
GloVe	42B	75.9	83.6	82.9	59.6	47.8
CBOW	100B	68.4	79.6	75.4	59.4	45.5

Table 6: Here we see the correlations between of word vector similarities using different embedding techniques with different human judgment datasets

Training for Extrinsic Tasks

目前我们关注了很多intrinsic任务并且强调他们在开发一个好的嵌入模型的重要性。当然，我们的最终目标是解决现实世界的问题，在extrinsic任务中使用生成的词向量。这里我们讨论处理extrinsic任务的一般方法。

1. **Problem Formulation**
   大多数NLP的外部任务可以公式化未一个分类任务。例如，给定一个句子，我嗯可以将这个句子划分为具有积极情感、消极情感或中立情感。相似的，在命名实体识别问题中，给定一段上下文和一个中心词，我们希望将中心词划分为多类别中的一类。对于输入"Jim bought 300 shares of Acme Corp. in 2006."，我们想得到一个分类的输出如："$[Jim]_{Person}$ bought 300 shares of $[AcmeCorp.]_{Organization}$ in $[2006]_{Time}$."
   对于这个问题，我们一般从训练集的形式入手：

$$\{x^{(i)}, y^{(i)}\}_1^N$$

   其中$x^{(i)}$是一个由一些词嵌入方法生成的d-维词向量，$y^{(i)}$是一个C-维的one-hot向量，表示我们最终想预测的类别。
   在典型的机器学习任务中，我们通常保持输入数据和目标标签不变，然后用最优化方法去训练模型权重。然而在NLP的应用中，我们引入一个概念：当我们为了一个外部任务训练模型是，我们会重新训练输入的词向量

（即生成不一样的词向量）。下面将讨论何时和为什么我们应该考虑使用这种方法。

2. **Retraining Word Vectors**
   如我们上面讨论的，我们用于外部任务的词向量是通过在一个较为简单的内部任务上进行优化而得到初始值的。在许多情况下，这些预训练的词向量是一个很好的代理，用于外部任务的最优词向量并且他们在外部任务中有好的表现。然而也存在这些预训练词向量可以使用外部任务进一步训练以获得更好表现的可能。然而重新训练词向量可能会有很高风险。
   如果我们使用外部任务重新训练词向量，我们需要保证训练集大到足够覆盖词汇表中的绝大多数词。这是因为Word2vec或着GloVe会将具有语义关联的单词映射到此空间中相同的区域。当我们在一个词汇表的一个小子集上重新训练这些单词，对于最后任务的表现可能是副作用的。下面用一个例子进一步阐述。考虑预训练的向量是在一个二维空间内（如下图6）。这里我们看到的词向量是被正确划分为外部分类的。现在如果我们只重新训练两个向量因为训练集大小的限制，那么我们会得到图7，其中一个单词背错误分类了，因为词向量更新导致边界位移。

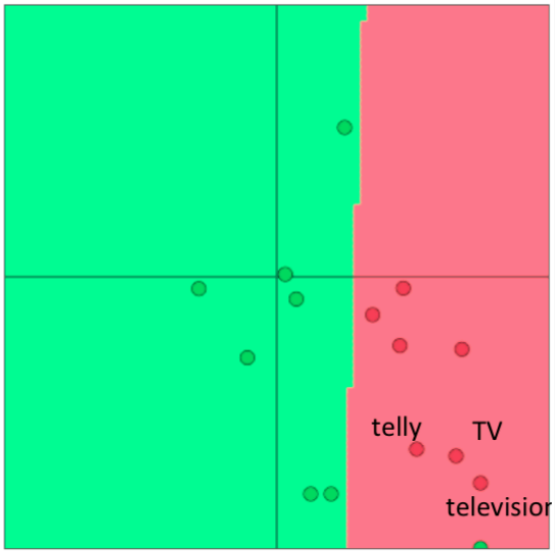


Figure 6: Here, we see that the words "Telly", "TV", and "Television" are classified correctly before retraining. "Telly" and "TV" are present in the extrinsic task training set while "Television" is only present in the test set.

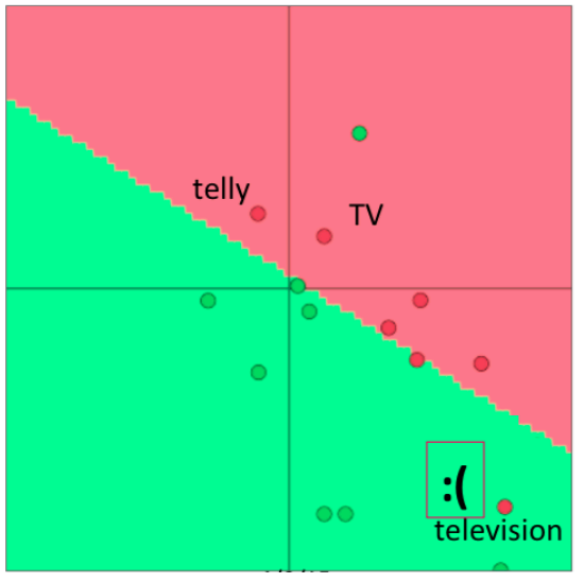


Figure 7: Here, we see that the words "Telly" and "TV" are classified correctly after training, but "Television" is not since it was not present in the training set.

   因此如果训练集小，则词向量不应该背重新训练。如果训练集足够大，重新训练会提升表现。

3. **Softmax Classification and Regularization**
   让我们考虑使用softmax分类函数：

$$p(y_j = 1|x) = \frac{exp(W_j x)}{\sum_{c=1}^{C} exp(W_c x)}$$

   这里我们计算词向量x被分到类别j的概率。使用Cross-entropy损失函数，我们可以计算训练样本的损失：

$$-\sum_{j=1}^{C} y_j \log(p(y_j = 1|x)) = -\sum_{j=1}^{C} y_j \log(\frac{exp(W_j x)}{\sum_{c=1}^{C} exp(W_c x)})$$

   当然，上式中最外的求和将有（C-1）个0，因为只有y_j是1在目前的情况中。因此让我们定义k为正确类别的index。那么我们可以将loss简化为：

$$-\log(\frac{exp(W_k x)}{\sum_{c=1}^{C} exp(W_c x)})$$

我们可以拓展上述loss到N个样本上：

$$-\sum_{i=1}^{N}\log(\frac{exp(W_{k(i)}x^{(i)})}{\sum_{c=1}^{C}exp(W_c x^{(i)})})$$

唯一的区别是$k(i)$现在是一个函数，用于返回样本$x^{(i)}$的正确index。
我们现在尝试估计需要被更新的参数数量（包括训练model weights和词向量）。我们知道简单的线性决策边界会需要一个模型输入最少一个d-维的词向量并且产生一个C类分布。因此为了更新模型的权重，我们会需要更新$C\cdot d$个参数。如果我们对词汇表V中的每一个词都做词向量更新，那么我们会更新$|V|$个词向量，每个词向量是d-维的。因此总共的参数个数将为$C\cdot d+|V|\cdot d$对于一个简单的线性分类器：

$$\nabla_{\theta}^{J(\theta)}=\begin{pmatrix}\nabla_{W_{\cdot 1}}\\ \vdots\\ \nabla_{W_{\cdot d}}\\ \nabla_{W_{x_{aardvark}}}\\ \vdots\\ \nabla_{W_{x_{zebra}}}\end{pmatrix}$$

考虑到这个模型的决策边界十分简单，这么大数量的参数很大可能会导致过拟合。
为了减少过拟合的风险，我们引入一个正则项：

$$-\sum_{i=1}^{N}\log(\frac{exp(W_{k(i)}x^{(i)})}{\sum_{c=1}^{C}exp(W_c x^{(i)})})+\lambda\sum_{k=1}^{C\cdot d+|V|\cdot d}\theta_k^2$$

最小化这个损失函数降低了参数取得十分大的值的可能，从而提升了模型的泛化能力（如果λ调到一个合适的值）。加入正则项的想法更像是一种要求，当我们探索一些更加复杂的模型（例如神经网络）。

4. **Window Classification**

目前我们首先探索了用一个单一的词向量x去完成外部任务的预测。在现实中，这是很难完成的，因为自然语言的天性。自然语言倾向于使用相同的词去表达非常不同的意思，并且我们通常需要通过上下文去决定这个单词的真正含义。例如，如果你被要求向某人解释"to sanction"的意思，你会马上意识到这取决于上下文，因为"to sanction"可以表达"to permit"或者"to punish"。在大多数情况下，我们倾向于使用一连串的单词输入到模型中。一连串是由一个中心词向量打头，然后后面跟着上下文词向量。上下文词选取的数量（context word window size）随着要解决的问题的不同而变化。一般来说，较窄的窗口大小会在语法测试上有更好的效果，而较宽的窗口大小会在语义测试上有更好的效果。
为了修改之前讨论过的softmax模型以使用上下文的词进行分类，我们简单地将$x^{(i)}$替换成$x_{window}^{(i)}$：

$$x_{window}^{(i)}=\begin{pmatrix}x^{(i-2)}\\ x^{(i-1)}\\ x^{(i)}\\ x^{(i+1)}\\ x^{(i+2)}\end{pmatrix}$$

因此，当我们计算损失的关于单词们的梯度时，我们会得到梯度向量：

$$\delta_{window}=\begin{pmatrix}\nabla_{x^{(i-2)}}\\ \nabla_{x^{(i-1)}}\\ \nabla_{x^{(i)}}\\ \nabla_{x^{(i+1)}}\\ \nabla_{x^{(i+2)}}\end{pmatrix}$$

5. **Non-linear Classifiers**

   我们现在引入非线性分类模型（例如神经网络）的需求。我们可以看到在图9中，一个线性分类器错分了很多数据点。使用一个非线性决策边界如图10所示，我们能够将所有数据点进行正确分类。即使过于简单化，这是一个典型的案例表现出对非线性决策边界的需求。在下一组notes中，我们将会学习神经网络，作为一类非线性模型并且已经在深度学习应用中取得很好效果。

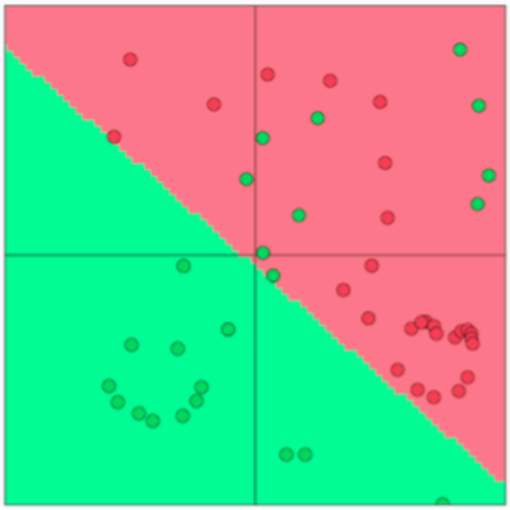


Figure 9: Here, we see that many examples are wrongly classified even though the best linear decision boundary is chosen. This is due linear decision boundaries have limited model capacity for this dataset.

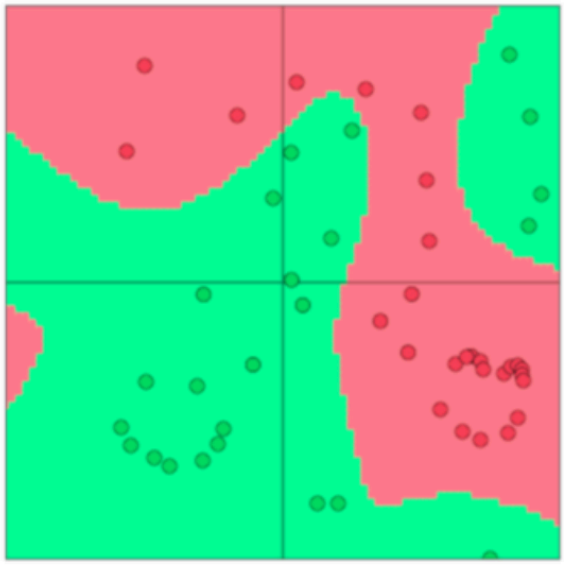


Figure 10: Here, we see that the non-linear decision boundary allows for much better classification of datapoints.