词的向量表示：word2vec

如何表达一个单词

过去人们使用分类词典(taxonomic resources，比如WordNet)来表达一个词的意义，这是人们在实践中建立起来的，很难再去获得更多有意义的价值。

分类词典的问题：

1、在分类层面上，失去了大量的细微差别(nuance)：很多同义词之间的微妙差别在分类词典中没有表示。比如adept, expert, good, practiced,proficient, skillful。相似但又有明显不同，不能混用，但在分类层面无法表示

2、不完整性:人类使用语言非常灵活，可能分类词典中没有相关的词汇

3、如何进行分类非常主观 ，不清晰，需要许多人多年的努力

4、无法度量单词之间的相似性（这是离散(discrete)表示的普遍问题,vector只是词表中的所有）

传统机器学习标记语言都是使用酒店，会议等原子符号，从神经网络的角度看，就是One-hot vector(高维向量中只有固定一维为1，其他为0)，维度可能非常高(维度与单词量相同)。

one hot vector的问题

它不能给出任何单词之间的内在关联，不能表示任何相似性，两个非常语义上非常相似的词，内积也为0。

事实上这个问题不仅存在于传统的基于逻辑的规则方法，基于概率的统计方法NLP方法，也存在这样的问题，虽然它们能表示一定程度上表示事情发生的概率，但是它们也是建立在符号表征的基础上，因此，它的每个单词都是独立的，不能表示任何两者之间的内在关系。

这里补充说明一些统计建模的存在的问题：

语言的统计模型可以用下一个词的条件概率表示:

$P(w_1^T) = \prod_{t=1}^{T} P(w_t|w_1^{t-1})$,其中$w_t$是第t个单词，$w_i^j = (w_i, w_{i+1}) \dots w_{j-1}, w_j$

语言建模的根本问题是**维度灾难**，如上述公式，假设有100000个单词，对连续10个单词的联合分布进行建模，那么就有100000*10-1个自由参数，另外，由于训练集有限，只能得到有限个连续词汇的组合，比如训练集中有The cat is walking in the bedroom ，根据语法和语义上的相似性，应该认为A dog was running in a room这样的句子有比较高的概率，对于这样的在训练集中出现次数为0的连续词汇组合，只能采用平滑给予很小的概率，显示是不合理。统计方法中通过减少连续词汇的个数的方法来尝试改善这个问题。

即**n-gram** 模型：$P(w_t|w_1^{t-1}) = P(w_t|w_{t-n-1}^{t-1})$，某一个词只与之前最接近的n-1个词相关。为了避免训练集中出现大量概率为0的项，只能取非常小的n，因此有两个明显的缺陷：**1、只能考虑很短的语境 2、没有考虑单词之间的相似性**，因此很难泛化[1]。

因此需要探索直接编码词汇的意义，使其能够衡量单词的相似性（相似性可以很好的解决许多概率为0的问题，更好的泛化）。

因此，我们需要一种表示单词之间相似性的表示方法。其idea就是**distributional similarity**

这是由英国语言学家JR Firth 提出的 you shall know a word by the company it keeps.**通过单词出现的上下文才能理解单词的意义 ，该词承载了句子中邻近词语大概是什么样的信息**

Wittgenstein（维特根斯坦）也提到:**a use theory of meaning** the right way to think about the meaning of words is understanding their uses in text，通过单词的在文旦中的使用来理解单词的意思。本质上说，如果你能够预测与某个单词相关的文本内容，那么你就可以理解这个单词的意义。

总结来说：我们要为每个单词创建一个密集向量，该向量能够很好地预测出现在这个单词的上下文中的其他单词,（ for each word we're going to come up for it a vector and that dense vector is gonna be chosen so that it'll be good at predicting other words that appear in the context of this word ）

对于每个单词，赋予一个向量，然后我们使用向量内积的方法作为相似度的衡量方式，然后我们改变单词的向量表示，使它能够很好的预测 。

关于相似度和可预测性之间老师并没有太多说明，我尝试在这里说明一下自己的理解：

如果两个单词出现在相同的语境中（上下文相同），那么它们的意思很可能非常相近：

假设word1,word2两个单词的上下文为context，如果我们认为所有的单词服从某个分布P，在给定context情况下，word1与word2 出现的概率相近，即P(word1|context) 与 P(word2|context)相近 ,分布P固定，context固定，那么word1与word2也应该是相近的。

word2vec

word2vec 是2003年Bengio 在他的论文《A neural probabilistic language model》中提出的，其

基本思想 ：

意义的使用理论(use theory of meaning)，预测每一个单词的上下文单词。

基本思路 ：

1、定义一个预测某个单词上下文的模型 ： $p(context|w_t)$

2、定义损失函数

3、利用大型语料库中资料进行训练，调整向量 ，训练后单词的量就可以很好的表示单词的意义。

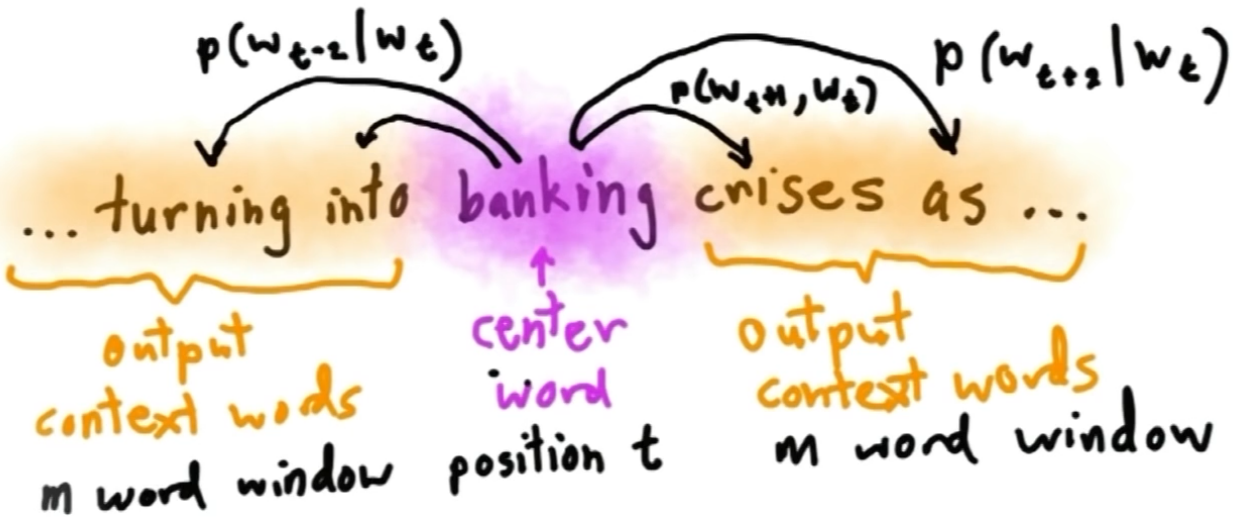
word2vec是一个家族：

Bengio 在2003年最早提出**NNLM** ，网络结构包含4层：one-hot 编码的输入层，投影层，隐藏层和softmax的输出层。

另外就是本文将要介绍的 **skip-gram（ SG ）** 和 **Continuous Bag of Words（ CBOW ）** ，其实原理与NNLM相同，为了提升学习效率，去掉了隐藏层，简化了网络结构，从而可以在更大的数据集上进行训练。

有两个快速的训练方法：**Hoerachical softmax** 和 **Negative sampling** ，后面也会介绍到。

skip-gram



skip-gram model如下：每次估测时，将一个词作为中心词(banking)，然后再选定的预测时窗内，定义某个单词出现的概率，即$P(w_{t-2}|w_t), P(w_{t-1}|w_t), P(w_{t+1}|w_t), P(w_{t+2}|w_t)$等，然后我们改变单词的向量表示，从而可以最大化上述的概率分布。

目标函数 ：

对于文档中出现的每个单词，最大化给定中心词出现其上下文的单词的概率：

$$J'(\theta) = \prod_{t=1}^{T} \prod_{\substack{-m \le j \le m \\ j \neq 0}} p(w_{t+j} | w_t ; \theta)$$

等价于最小化正规化负的对数似然函数：

Negative Log Likelihood

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

θ 为需要学习的参数，也就是单词的向量表示

m为定义的半径，超参数

T为文档长度

接下来定义概率分布模型：

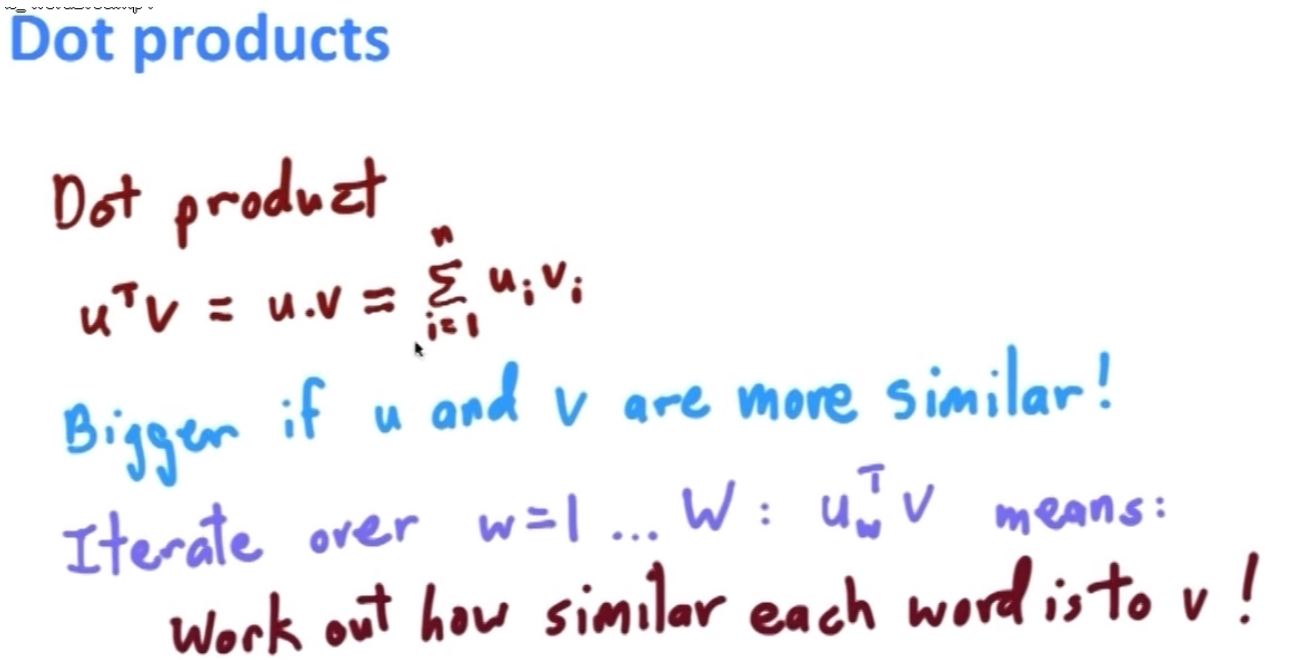
$$p(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w=1}^{V} \exp(u_w^T v_c)}$$

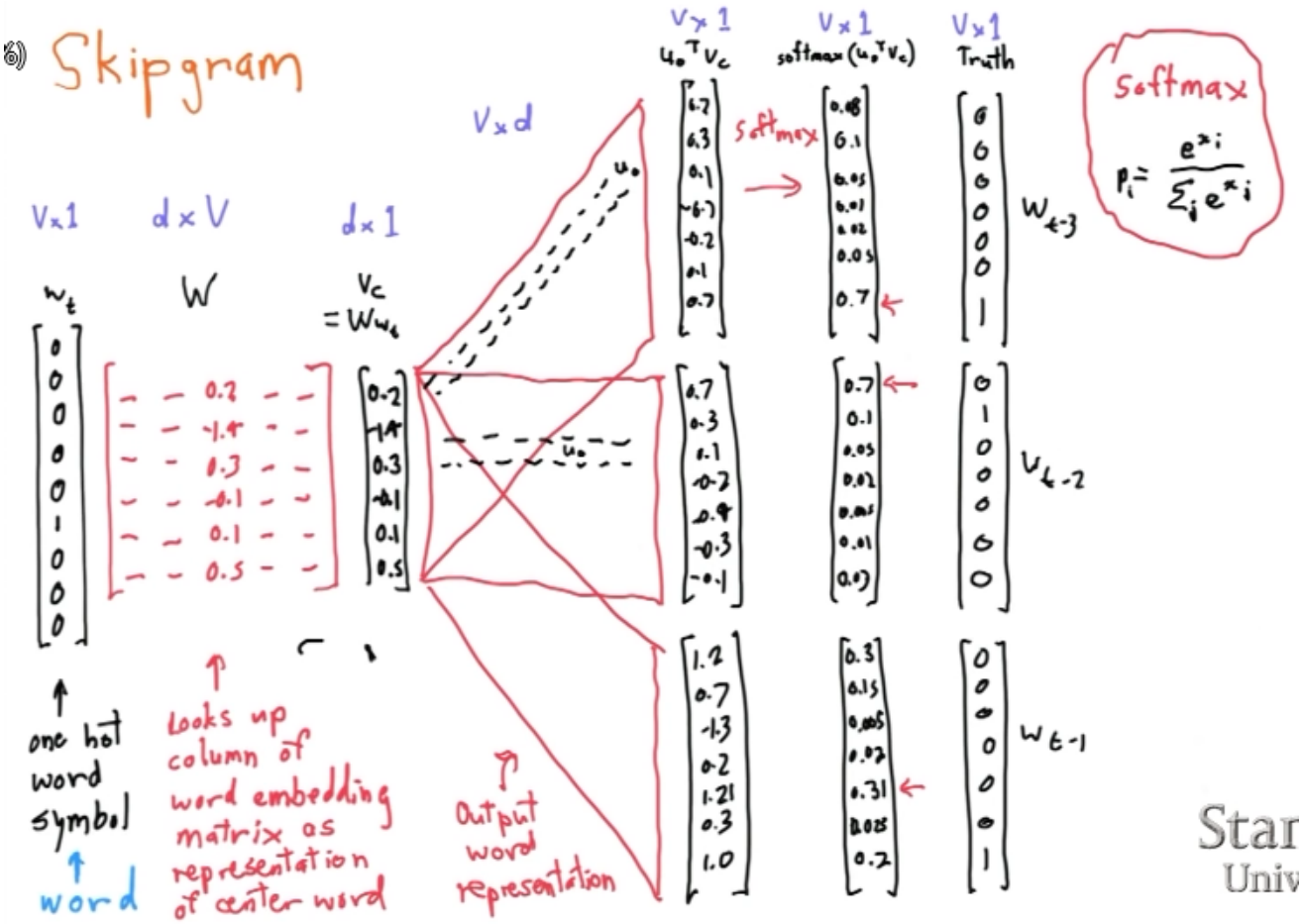
其中c,o为单词在词表中的索引，v_c为与中心词有关的向量表示，u_o为上下文的向量表示，上式中使用向量内积来计算概率，关于这个问题，事实上课程最后老师解释道，事实上没有什么必要的原因（只是因为这样做是最明显最简单，使用cos计算相似度，需要更多的数学的计算）

内积是相似度的衡量。

对于每个单词，为了数学上的简单，使用两个向量u,v来表示同一个单词，v表示center word vector，u 表示context word vector(如果将两个tie在一起训练，数学上会更难)，这种方式实践中也更好一些。

该模型中上下文单词的位置不重要，有其他模型关注单词的位置和距离，不过那主要是在句法(syntatic)上，而不是在语义(semantic)上,实践也说明，对于语义来说，位置和距离不重要





最左边是$w_t$center word的one-hot vector ，紧接着矩阵W是center word 的向量化表示，将W于w_t相乘得到W中对于的列即中心词的向量表示，然后乘上第二个矩阵(context word 的向量表示)，得到相似度，然后做softmax得到概率，与真实数据对比，计算损失。

训练模型

r

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

接下来就是使用梯度下降法进行训练

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

$P(u_o|v_c) = \frac{exp^{u_o^T v_c}}{\sum_{w=1}^{V}(exp^{u_w^T v_c})}$

$\frac{\alpha P(u_o|v_c)}{\alpha_{v_c}} = \frac{\alpha}{\alpha v_c}(log^{exp^{u_o^T v_c}} - log^{\sum_{w=1}^{V} exp^{u_w^T v_c}}) =\ldots= u_0 - \sum_{x=1}^{v} P(u_x|v_c)u_x$

定义$U = [u_1, u_2 \ldots u_W]$为输出向量。对上式化简可以得到（y只有在$u_0$处才为0）：

$\frac{\alpha J}{\alpha_{v_c}} = U(\hat{y} - y)$

上式意义非常明确，增大观察到的context word vector，并以概率减小其他向量。

另外还需要计算损失函数关于context word vector的导数：$\frac{\alpha P(u_o|v_c)}{\alpha_{u_i}}$

当i不是c的上下文中时：

$\frac{\alpha P(u_o|v_c)}{\alpha_{u_i}} = -P(u_i|v_c)v_c$

当i在c的上下文中时：

$\frac{\alpha P(u_o|v_c)}{\alpha_{u_o}} = v_c - P(u_o|v_c)v_c$

总结上式可以得到：

$\frac{\alpha J}{\alpha U} = v_c(\hat{y} - y)^T$

然后使用梯度下降法进行训练即可：

$\theta = \theta - \alpha\frac{\alpha J}{\alpha \theta}$

python 实现softmaxCostAndGradient

```
def softmaxCostAndGradient(predicted, target, outputVectors, dataset):
""" Softmax cost function for word2vec models
Implement the cost and gradients for one predicted word vector
and one target word vector as a building block for word2vec
models, assuming the softmax prediction function and cross
entropy loss.

Arguments:
predicted -- numpy ndarray, predicted word vector (\hat{v} in
             the written component)
target -- integer, the index of the target word
outputVectors -- "output" vectors (as rows) for all tokens
dataset -- needed for negative sampling, unused here.

Return:
cost -- cross entropy cost for the softmax word prediction
gradPred -- the gradient with respect to the predicted word
       vector
grad -- the gradient with respect to all the other word
       vectors

We will not provide starter code for this function, but feel
free to reference the code you previously wrote for this
assignment!
"""

### YOUR CODE HERE
## Gradient for $\hat{\bm{v}}$:

#  Calculate the predictions:
wxPlusb = np.dot(outputVectors, predicted)
yhat = softmax(wxPlusb)

#  Calculate the cost:
cost = -np.log(yhat[target])

#  Gradients
yhat[target] -= 1.0
```

```
grad = np.outer(yhat,predicted )
gradPred = np.dot(outputVectors.T, yhat)
### END YOUR CODE

return cost, gradPred, grad
```

pythnon 实现skip-gram

```
def skipgram(currentWord, C, contextWords, tokens, inputVectors, outputVectors,dataset,
word2vecCostAndGradient=softmaxCostAndGradient):
""" Skip-gram model in word2vec

Implement the skip-gram model in this function.

Arguments:
currrentWord -- a string of the current center word
C -- integer, context size
contextWords -- list of no more than 2*C strings, the context words
tokens -- a dictionary that maps words to their indices in
          the word vector list
inputVectors -- "input" word vectors (as rows) for all tokens
outputVectors -- "output" word vectors (as rows) for all tokens
word2vecCostAndGradient -- the cost and gradient function for
                           a prediction vector given the target
                           word vectors, could be one of the two
                           cost functions you implemented above.

Return:
cost -- the cost function value for the skip-gram model
grad -- the gradient with respect to the word vectors
"""

cost = 0.0
gradIn = np.zeros(inputVectors.shape)
gradOut = np.zeros(outputVectors.shape)

### YOUR CODE HERE
#获取当前词索引
index = tokens[currentWord]
#获取单前词Input向量
predicted = inputVectors[index]
#对于上下文中的每个单词根据公式计算cost和相应的梯度
for word in contextWords:
    target = tokens[word]
    cost_temp,gradIn_temp,gradOut_temp =
word2vecCostAndGradient(predicted,target,outputVectors,dataset)
    cost +=cost_temp
    gradIn[index] += gradIn_temp
    gradOut +=gradOut_temp


### END YOUR CODE
```

```
return cost, gradIn, gradOut
```

CBOW

与**skip-gram** 不同 ，**CBOW**(Continuous Bag-of-Words Model)是根据上下文来预测单前词的语言模型，

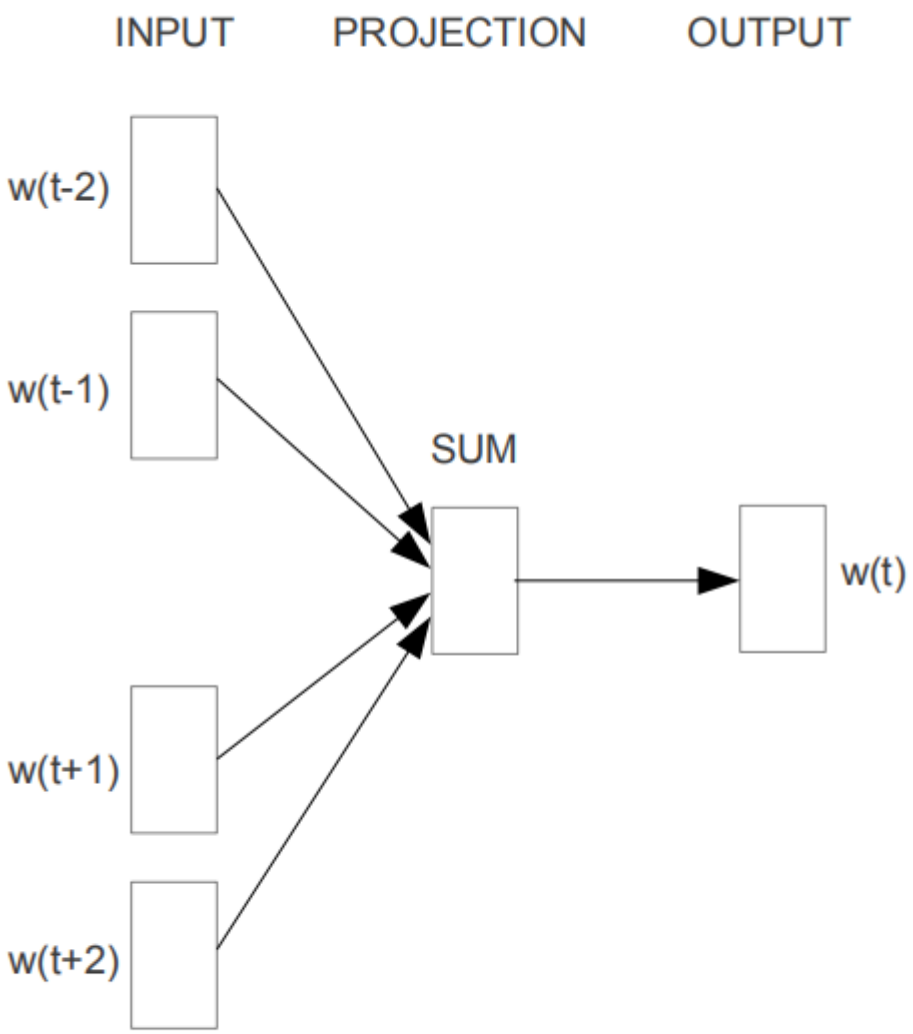
对数损失函数为：$J(\theta) = -\frac{1}{T}\sum_{t=1}^{T} log^{P(w|context(w))}$

其网络架构如下，包含三层：

输入层 ：上下文词语的词向量（可以理解为该网络前还有一个one-hot向量的输入层，其权重为表示所有词的词向量，两者相乘，就为上下文的词向量，由于与one-hot 向量相乘比较特殊，就是W对于的列，因此为了简单，直接去W中对应的列放到输入层）

投影层 ：对输入层向量进行求和，也是为了简化结构，加速训练。

输出层：与skip-gram一样，也为softmax(分类数为单词数)，输出概率，与w_t的one-hot编码对比，计算损失，更新权值。



CBOW

python 实现CBOW

```
def cbow(currentWord, C, contextWords, tokens, inputVectors, outputVectors,
         dataset, word2vecCostAndGradient=softmaxCostAndGradient):
    """CBOW model in word2vec

    Implement the continuous bag-of-words model in this function.

    Arguments/Return specifications: same as the skip-gram model

    Extra credit: Implementing CBOW is optional, but the gradient
    derivations are not. If you decide not to implement CBOW, remove
    the NotImplementedError.
    """


    cost = 0.0
```

```
    gradIn = np.zeros(inputVectors.shape)
    gradOut = np.zeros(outputVectors.shape)

    ### YOUR CODE HERE
    target = tokens[currentWord]
    predicted_indices =[tokens[word] for word in contextWords]
    predicted_vectors = inputVectors[predicted_indices]
    #将上下文向量求和，放入投影层
    predicted = np.sum(predicted_vectors,axis=0)
    cost,gradIn_sum, gradOut = word2vecCostAndGradient(predicted,target,outputVectors,dataset)
    for i in predicted_indices:
        gradIn[i] += gradIn_sum

    ### END YOUR CODE

    return cost, gradIn, gradOut
```

不同网络架构对比

传统的机器学习方法对词向量的表示方法主要是利用矩阵分解的方法，例如LSA(潜在语义分析)，LDA(潜在狄利克雷分配)等方法，YouHua Bengio等在《A Neural Probabilistic Language Model》一文中提出了一种用于估计神经网络语言模型（NNLM）模型结构，用来共同学习单词向量表示和统计语言模型。其网络架构如下图所示：

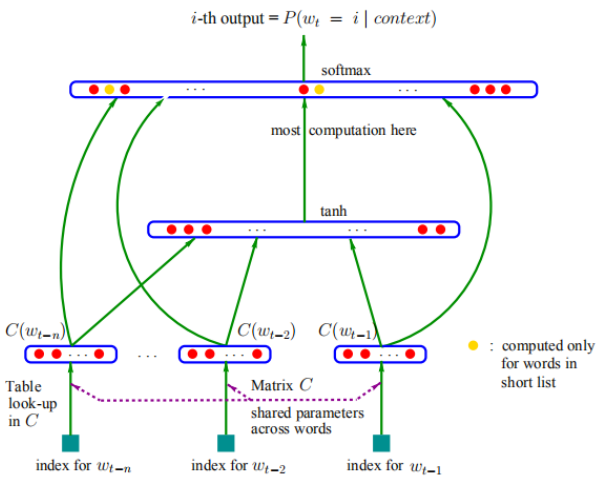
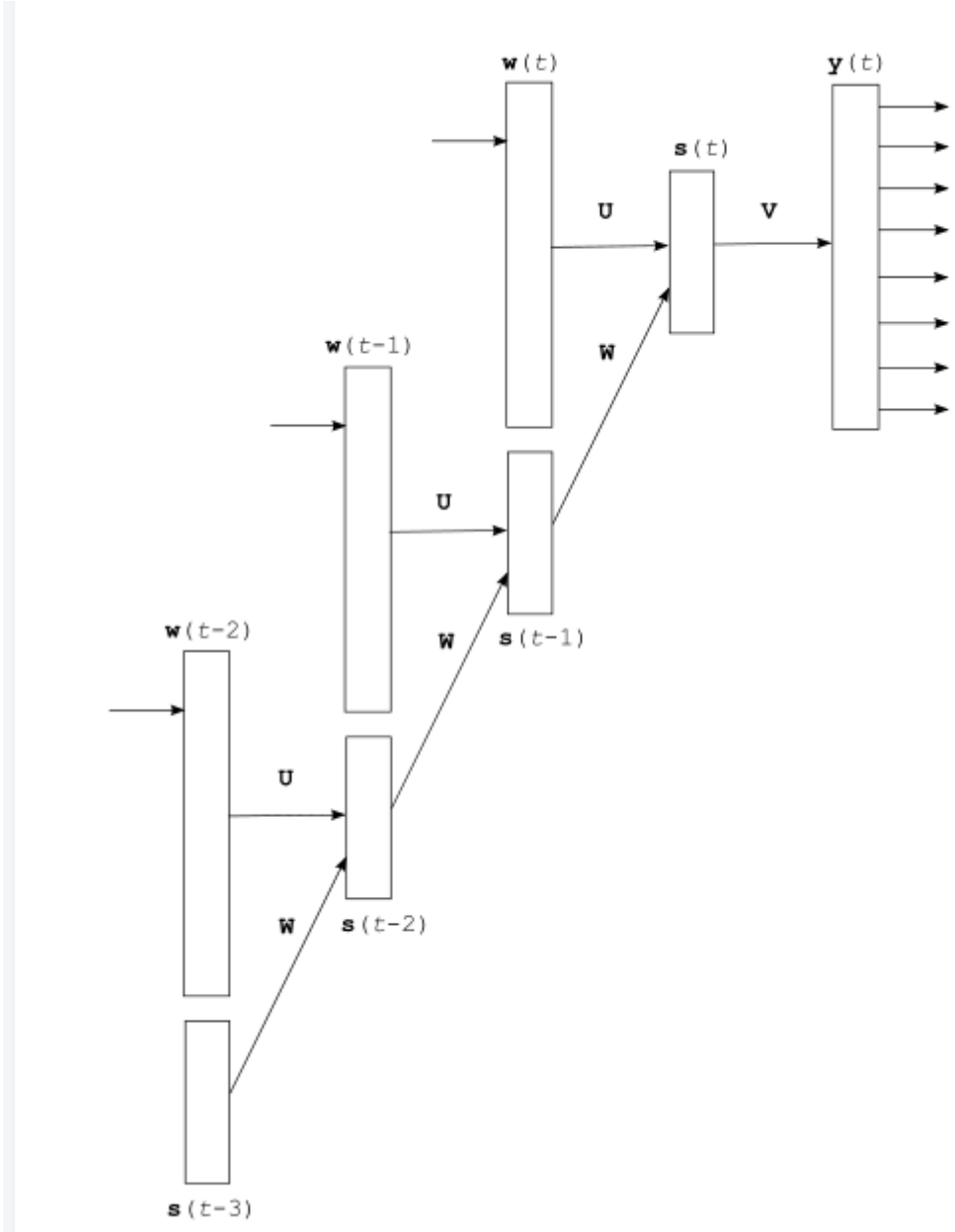


Figure 1: "Direct Architecture": $f(i, w_{t-1}, \cdots, w_{t-n}) = g(i, C(w_{t-1}), \cdots, C(w_{t-n}))$ where g is the neural network and $C(i)$ is the i-th word feature vector.

Training is achieved by looking for (θ, C) that maximize the training corpus penalized log-likelihood: $L = \frac{1}{T} \sum_t \log p_{w_t}(C(w_{t-n}), \cdots, C(w_{t-1}); \theta) + R(\theta, C)$, where $R(\theta, C)$ is a regularization term (e.g. a weight decay $\lambda||\theta||^2$, that penalizes slightly the norm of θ).

包含输入层，投影层，隐藏层，输出层，输入层是one-hot 向量，使用投影矩阵将输入层投影到投影层，投影层，隐藏层，输出层位一个简单的全连接网络，计算量最大的部分在隐藏层与输出层之间，使用 hierarchical softmax 和负采样等技术（后面详细说明），可以显著减少改成计算了，从而使计算瓶颈变为投影层与隐藏层之间计算。

T. Mikolov 等在《 Recurrent neural network based language model》一文中使用循环神经网络来共同学习单词向量表示和统计语言模型，包含输入层，隐藏层和输出层，其中输入层为词向量的表示，克服了NNLM需要制定上下文长度的限制，并且可以捕获一些短期记忆。其网络架构如下图，通过BPTT算法更新模型参数，学习词向量表示。



Tomas Mikolov 等在《Efficient Estimation of Word Representations in Vector Space》提出了skip-gram和CBOW模型（本文实现的方法），相比于NNLM和RNNLM，skip-gram和CBOW模型更加简单，从而可以在很低的计算成本下，从数十亿字的庞大数据集中学习高质量的单词向量。而且学习得到的单词向量不仅相似的单词彼此接近，而且使用简单的向量运算，可以捕获某些句法和语义上的信息。例如：

$$vector('smallest') = vector('biggest') - vector('big') + vector('small')$$

$$vector('France') = vector('Germany') - vector('Berlin') + vector('Paris')$$

Tomas Mikolov 等在《Efficient Estimation of Word Representations in Vector Space》中对上述方法做了对比：RNNLM语义表现不好，NNLM由于拥有隐藏层(事实上在文章中，也提到了隐藏层的作用)，显著好于RNNLM，CBOW在语义上与NNLM相当，在句法上明显好与NNLM，skip-gram在语义上明显好于其他所有结构，句法上比CBOW稍差但比NNLM要好。

Table 3: *Comparison of architectures using models trained on the same data, with 640-dimensional word vectors. The accuracies are reported on our Semantic-Syntactic Word Relationship test set, and on the syntactic relationship test set of [20]*

Model Architecture	Semantic-Syntactic Word Relationship test set		MSR Word Relatedness Test Set [20]
	Semantic Accuracy [%]	Syntactic Accuracy [%]	
RNNLM	9	36	35
NNLM	23	53	47
CBOW	24	64	61
Skip-gram	55	59	56

对于许多其他NLP任务，将词向量作为单词向量的初始值，对于数据量比较小的任务，甚至可以直接作为单词的表示（不参与训练），可以显著提升某些任务的表现。

softmax 计算问题

word2vec的推导中，我们可以看到最后要计算一个与单词量大小的softmax问题（分母项中需要计算与所有单词（百万级别以上）的内积并做指数计算，事实上这是深度学习在NLP中一个非常普遍的问题），对于大型的语料库，计算量非常大，下面介绍一些解决方案：1

1、**short list** ：使用最频繁的单词 ，其他单词都使用UNK表示。

这种方法实现最简单，但存在明显的问题，根据zipf定律，语料库中少部分高频词占很大部分，大部分词出现次数很少或者根本不出现，长尾效应很强，因此，对于绝大部分的低频次，都不能进行很好的处理。

2、**batch short list** :对每个batch进行处理时，仅仅使用该batch中存在的词汇，忽略不在该batch中的数据，在机器翻译中广泛应用，可以显著提高效率。但是这样得到的向量可能不准确（概率近似值可能不稳定）。

3、**逼近softmax 目标函数**

逼近softmax目标函数，主要包含两种方式：**因子分解softmax(Factorise the output vocabulary)** 和**随机采样近似法(stochastic approximation)**,下面就这两种途径的精神和典型方法做一些简单论述。

因子分解法 ：

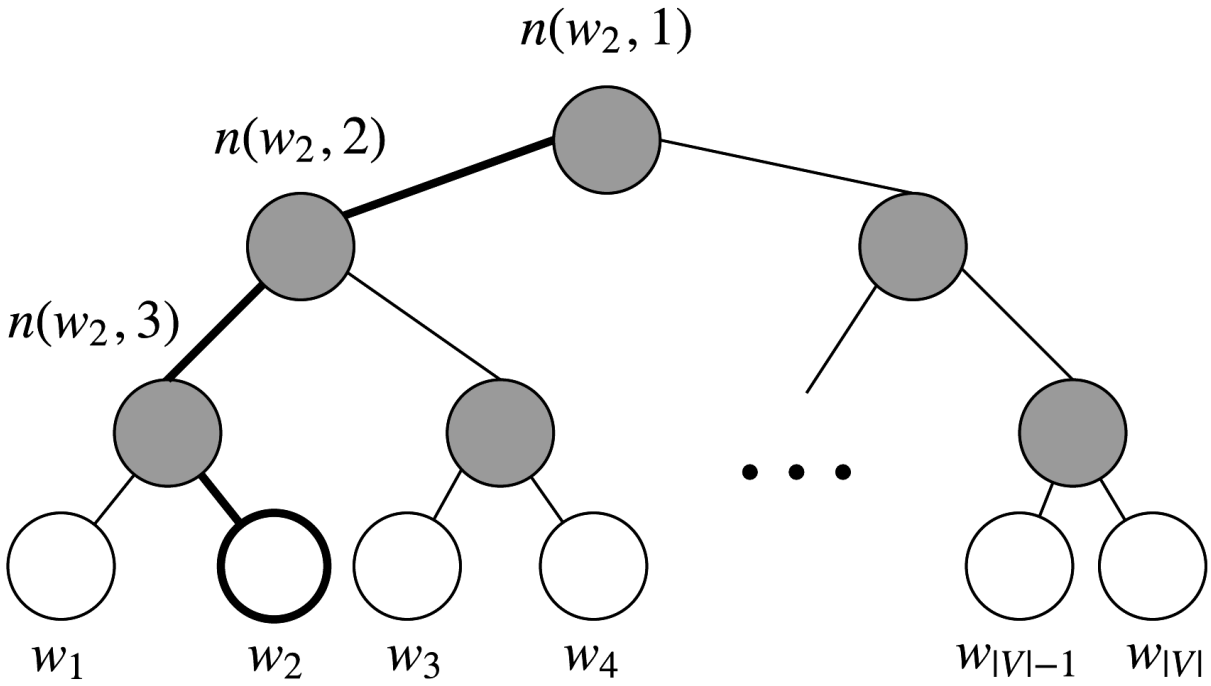
基本精神：使用链式法则，将一个大的softmax问题分解成一系列softmax。

class based softmax : 假设有一个函数将每个单词映射到一个类（类比词汇小的多），不在从完整的词汇中预测单词，先预测单词在哪个类中，然后再在所属类中预测我们要预测的单词，从而有两个softmax，但每个都很小。合理选择类，在测试和训练阶段都可以有$\sqrt{V}$的加速。（已经有很多算法将词汇组放到合适的类中）。该方法非常有效，而且提高合理的效率。

Hierarchical softmax: 将class based softmax 进行扩展，不仅仅分解一层，而是利用一个二叉树进行多层分类，二叉树中每个叶子节点代表词汇表中的单词，通过在树中的每个分支上来选择单词，为了获得更好的加速，二叉树采用霍夫曼树（以词频作为节点的权重进行构建），因此高频词编码更短，更靠近根节点，需要学习的词向量都在叶子节点上。

Hierarchical softmax 有两个主要问题：

1、无法获得很好的将所有单词进行分类 2、不容易在GPU上进行并行优化



定义：

p^w ：从root到w对应的节点路径

l^w ： root 带w路径长度

$p_1^w, p_2^w \dots p_{l^w}^w$ ： 路径p^w 上的各个节点

$d_1^w, d_2^w, \dots, d_{l^w}^w$ ：为节点在霍夫曼树中的编码（ 0,1 ）

$\theta_1^w, \theta_2^w, \dots, \theta_{l^w-1}^w$ ：路径p^w中非叶结点对于的参数向量

则$P(w|context(w)) = P(w|X_w) = \prod_{j=2}^{l^w} P(d_j^w|X_w, \theta_{j-1}^w)$,其中$X_w = \sum_{u \in context(w)} u$

（ $P(Y|X) = \sum_i P(Y, C=i|X) = \sum_i P(Y|C=i)P(C=i|X)$ ）,如果只有一个C与Y有关，

那么$P(Y|X) = P(Y|C=i)P(C=i|X)$ ））

$$P(d_j^w|X_w, \theta_{j-1}^w) = \begin{Bmatrix} \sigma(X_w^T\theta_{j-1}^w), d_j^w = 0 \\ 1-\sigma(X_w^T\theta_{j-1}^w), d_j^w = 1 \end{Bmatrix}$$

将上式合并有：$P(d_j^w|X_w, \theta_{j-1}^w) = \sigma(X_w^T\theta_{j-1}^w)^{1-d_j^w}(1-\sigma(X_w^T\theta_{j-1}^w))^{d_j^w}$

对数损失函数为：$J(\theta) = \frac{1}{|C|}\sum_{w \in C}^{T} log^{P(w|context(w))} = \frac{1}{|C|}\sum_{w \in C}^{T}\sum_{j=2}^{l^w}\sigma(X_w^T\theta_{j-1}^w)^{1-d_j^w}(1-\sigma(X_w^T\theta_{j-1}^w))^{d_j^w}$

每一项可以简记为：$l(w,j) = (1-d_j^w)log^{\sigma(X_w^T\theta_{j-1}^w)} + d_j^w log^{(1-\sigma(X_w^T\theta_{j-1}^w))^{d_j^w}}$

采用SGD策略，只计算一个w。

$\frac{\alpha J(\theta)}{\alpha\theta_{j-1}^w} = \frac{\alpha l(w,j)}{\alpha\theta_{j-1}^w} = \frac{\alpha}{\alpha\theta_{j-1}^w}((1-d_j^w)log^{\sigma(X_w^T\theta_{j-1}^w)} + d_j^w log^{(1-\sigma(X_w^T\theta_{j-1}^w))^{d_j^w}})$

$J(\theta)$中只有$l(w,j)$与θ_{j-1}^w有关

因为$\sigma'(x) = \sigma(x)(1-\sigma(x))$

从而有：$\frac{\alpha J(\theta)}{\alpha \theta_{j-1}^w} = (1 - d_j^w - \sigma(X_w^T \theta_{j-1}^w))X_w$

由于X_w与θ_{j-1}^w对称，$J(\theta)$中每个$l(w,j)$都与θ_{j-1}^w有关，$J(\theta) = \sum_{j=2}^{l^w} l(j,w)$

所以有$\frac{J(\theta)}{\alpha(u_{u \in context(w)})} = \frac{\alpha J(\theta)}{\alpha X_w} = \sum_{j=2}^{l^w}(1 - d_j^w - \sigma(X_w^T \theta_{j-1}^w))\theta_{j-1}^w$

从而可以利用梯度上升法进行更新

随机近似法（stochastic approximation）的采样来近似目标函数：

基本精神：采用机器学习算法中one-versus-All(OVA)策略：

对于多分类问题中每一种类型，$k \in y$,定义：重新定义训练集$D[k] = [(x_n, y' = (2[y_n = k] - 1)]$,然后对每个类别都进行二分类问题，可以证明OVA与softmax是等价的。

NCE（Noise Contrastive Estimation，噪声对比估计）：采用上述OVA思想，在对每个单词进行预测时，我们将所有单词分为两类，一类为在预测词上下文中的词，一类为不在预测词上下文中的词，我们的目标是试**最大限度地提高我们想要预测的单词的概率，并且最小化其他单词的概率**，然后执行一系列二分类问题，但是对对每个词都执行二分类问题，计算量太大，因此，我们尝试从词汇表采样K个噪声样本，对于预测的单词和噪声样本，做二分类问题，事实证明，当样本数量非常大时，可以作为softmax很好的近似。

需要说明的是，test时，仍然需要计算一个完整的softmax，只在训练时更快。

Importance Sampling（重要性采样）：与NCE相似，也尝试从词汇表采样K个噪声样本，但不去执行一系列二分类问题，而是执行一个包含目标样本和噪声样本的多分类问题，使用softmax，但这个softmax要小的多，事实证明，这是一个更稳健的方法。

NEG（Negative Sampling，负采样）：NCE 的目标是作为softmax的一种近似，skip-gram 模型仅仅需要学习高质量的词向量表示，所以可以将NCE进行简化，定义NEG目标函数为：

$J_{neg_sample(o,v_c,U)} = -log(\sigma(u_o^T v_c)) - \sum_{k=1}^{K} log(\sigma(-u_k^T v_c))$

根据$log'(x) = \frac{1}{x}$ 和 $\sigma'(x) = \sigma(x)(1 - \sigma(x))$

可以得到：

$\frac{\alpha J}{\alpha v_c} = (\sigma(u_o^T v_c) - 1)u_o - \sum_{k=1}^{K}(\sigma(u_k^T v_c))u_k$

$\frac{\alpha J}{\alpha u_o} = (\sigma(u_o^T v_c) - 1)v_c$

$\frac{\alpha J}{\alpha u_k} = (1 - \sigma(u_k^T v_c))v_c$ for all k = 1,2,....K

```
 def negSamplingCostAndGradient(predicted, target, outputVectors, dataset,
   K=10):
""" Negative sampling cost function for word2vec models

Implement the cost and gradients for one predicted word vector
and one target word vector as a building block for word2vec
models, using the negative sampling technique. K is the sample
size.

Note: See test_word2vec below for dataset's initialization.

Arguments/Return Specifications: same as softmaxCostAndGradient
```

```
    """

    # Sampling of indices is done for you. Do not modify this if you
    # wish to match the autograder and receive points!
    indices = [target]
    indices.extend(getNegativeSamples(target, dataset, K))

    ### YOUR CODE HERE
    #初始化
    cost = 0
    gradPred = np.zeros_like(predicted)
    grad = np.zeros_like(outputVectors)

    uovc = sigmoid(np.dot(outputVectors[target],predicted))
    cost = cost -np.log(uovc)
    gradPred = gradPred + np.multiply((uovc-1.0),outputVectors[target])
    grad[target] = grad[target] + np.multiply((uovc-1.0),predicted)

    for i in range(K):
        uk = outputVectors[indices[i+1]]
        ukvc = sigmoid(np.dot(uk,predicted))
        cost = cost - np.log(1.0-ukvc)
        gradPred = gradPred + np.multiply((ukvc),uk)
        grad[indices[i+1]] = grad[indices[i+1]] + np.multiply((ukvc),predicted)

    ### END YOUR CODE

    return cost, gradPred, grad
```

NEG采样 ：

关于采用，首先想到的可能是均匀采用或者根据词频进行采样，但NEG的文章说到，如果采用以词频的3/4次方的概率进行采样会更好。

具体操作如下：

定义c(w)表示词汇表中词w的词频

$sum_{weight} = \sum_{w \in 词汇表} c(w)^{0.75}$，那么采样到第j个单词$w_j$的概率为:$P(w_j) = c(w_j)^{0.75} / sum_{weight}$

有$\sum_{w \in 词汇表} P(w) = 1$,为了实现采样，将每个w的概率累加放到0-1之间

那么第j个单词w_j的区间为$\sum_{i=0}^{j-1} P(w_i)$到$\sum_{i=0}^{j} P(w_i)$，语料库很大时，可能有些单词出现非常小的概率，采样比较困难，因此可以每个区间乘上一个比较大的值N，使采样更加简单。那么那么第j个单词w_j的区间为$\sum_{i=0}^{j-1} P(w_i) * N$到$\sum_{i=0}^{j} P(w_i) * N$。

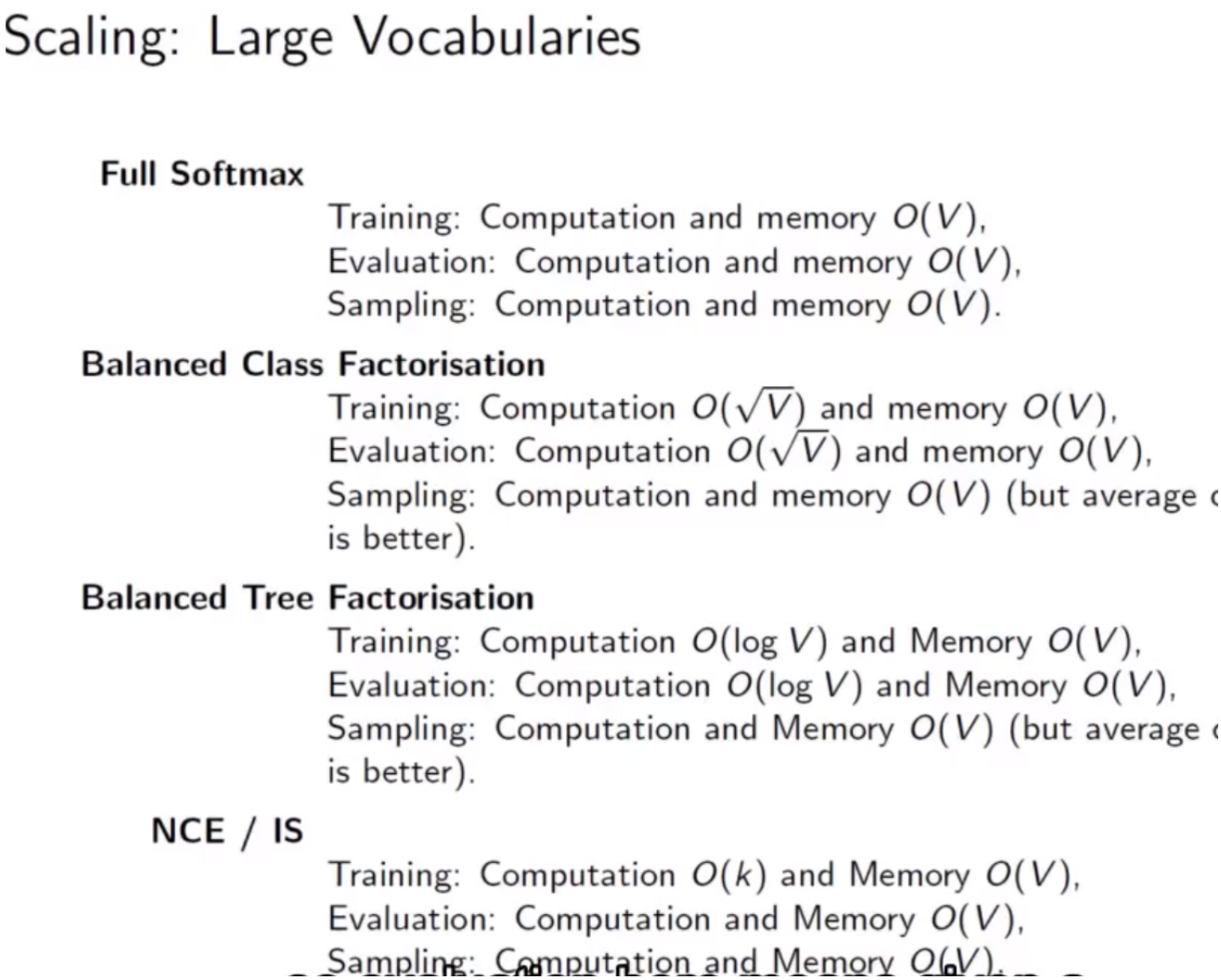
总结就是在0-N之间进行均匀采用，采样到的值如果在$\sum_{i=0}^{j-1} P(w_i) * N$与$\sum_{i=0}^{j} P(w_i) * N$之间，则表示采样第j个词$w_j$。

关于NEG采用中k的值，原文中建议：对于较小的训练集，k取5-20，对于比较大的数据集，k取2-5就可以。

文章中还提到对于过于频繁的词汇，比如'a','the','in'等词汇，不能表达太多的意义，因此，可以采用完成后可以按一定概率扔掉一些：

扔掉的概率为:$P(w_i) = 1 - \sqrt{t/f(w_i)}$,t一般取1e-3到1e-5。对于概率低于t的词语会全部保留，概率高于t的词汇可能会被甩掉很多，且概率越大，甩的可能越多。比如，如果$P(w_i) = 1e-3$，那么会有90%的概率会被甩掉，$P(w_i) = 1e-1$，那么会有99%的概率会被甩掉。

下图是不同softmax解决方案在训练，测试阶段需要的内存和计算量的总结。



相关课程：

斯坦福 CS224n:Natural Language Processing with Deep Learning ，Word Vector Representations word2vec

python 实现来自assigment1

牛津大学&DeepMind Deep Learning for Natural Language Processing ,Language Modelling

台湾大学（林轩田）机器学习基石 Linear Models for Classification Multiclass via Logistic Regression

相关文章：

YouSua Bengio《A neural probabilistic language model》

T. Mikolov《 Recurrent neural network based language model》

Tomas Mikolov《Efficient Estimation of Word Representations in Vector Space》

Tomas Mikolov《Distributed Representations of Words and Phrases and their Compositionality》

相关博客：

http://www.hankcs.com/nlp/word2vec.html