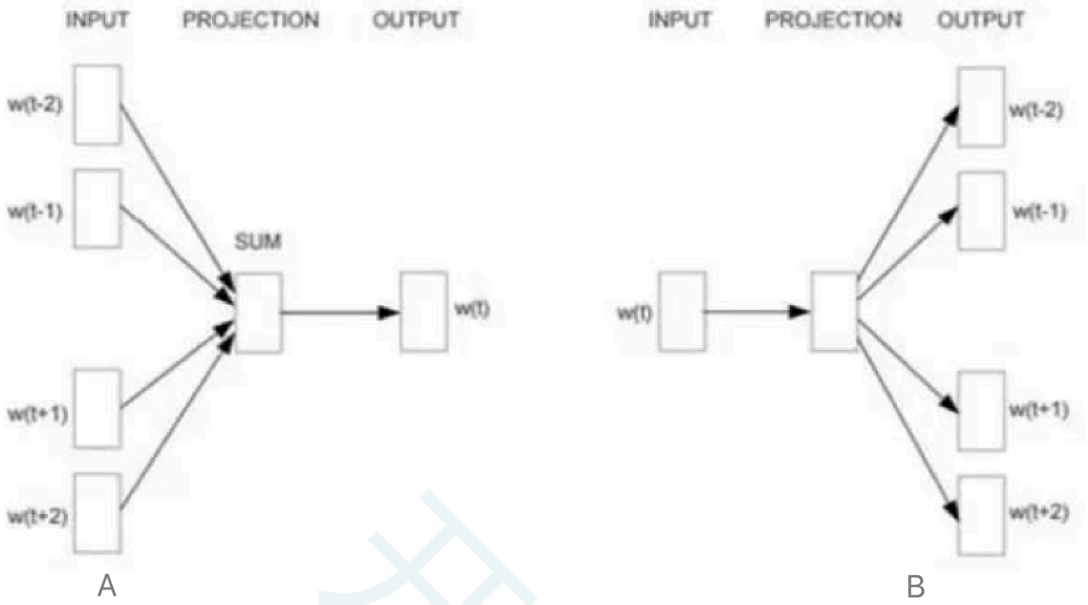
1. Skip gram 模型是在 Word2vec 算法中为词嵌入而设计的最优模型。以下哪一项描绘了 Skip gram 模型？



A、A
B、B
C、A 和 B
D、以上都不是

解析：

答案为（B）：这两个模型都是在 Word2vec 算法中所使用的。模型 A 代表着 CBOW，模型 B 代表着 Skip gram。

2. 中文同义词替换时，常用到 Word2Vec，以下说法错误的是

A、Word2Vec 基于概率统计
B、Word2Vec 结果符合当前预料环境
C、Word2Vec 得到的都是语义上的同义词
D、Word2Vec 受限于训练语料的数量和质量

解析：

正确答案是：C

Word2vec，为一群用来产生词向量的相关模型。这些模型为浅而双层的神经网络，用来训练以重新建构语言学之词文本。网络以词表现，并且需猜测相邻位置的输入词，在 word2vec 中词袋模型假设下，词的顺序是不重要的。

训练完成之后，word2vec 模型可用来映射每个词到一个向量，可用来表示词对词之间的关系。该向量为神经网络之隐藏。

Word2vec 依赖 skip-grams 或连续词袋（CBOW）来建立神经词嵌入。

3. 下列方法中，不可以用于特征降维的方法包括

A、主成分分析 PCA

B、线性判别分析 LDA

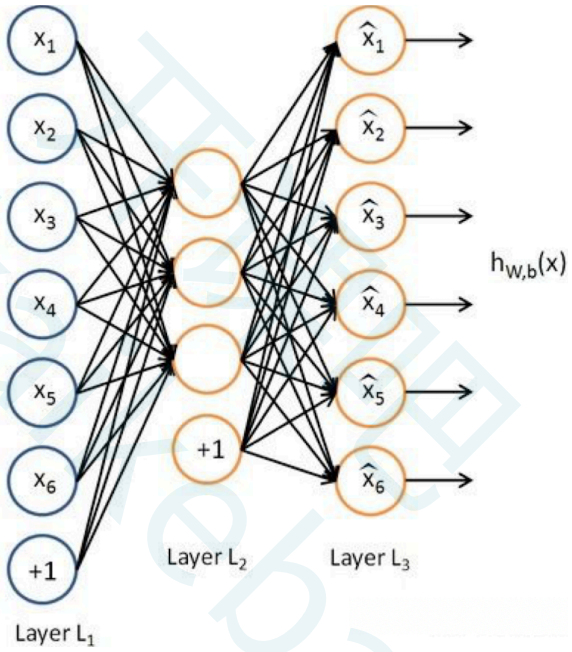
C、深度学习 SparseAutoEncoder

D、矩阵奇异值分解 SVD

解析：

正确答案是：C

特征降维方法主要有： PCA，LLE，Isomap SVD 和 PCA 类似，也可以看成一种降维方法 LDA:线性判别分析，可用于降维 AutoEncoder：AutoEncoder 的结构与神经网络的隐含层相同，由输入 L1,输出 L2 组成，中间则是权重连接。Autoencoder 通过 L2 得到输入的重构 L3，最小化 L3 与 L1 的差别 进行训练得到权重。在这样的权重参数下，得到的 L2 可以尽可能的保存 L1 的信息。 Autoencoder 的输出 L2 的维度由输出的神经元个数决定。当输出维度大于 L1 时，则需要在训练目标函数中加入 sparse 惩罚项，避免 L2 直接复制 L1（权重全为 1）。所以称为 sparseAutoencoder(Andrew Ng 提出的)。 结论：SparseAutoencoder 大多数情况下都是升维的，所以称之为特征降维的方法不准确。



4. 代码补全题：

对中文进行分词

"我爱北京天安门"————>"我 爱 北京 天安门"

设计如下函数，将对应输入转换为我们需要的分词之后的字符串。使用 jieba 补全如下代码：

```
def cut_word(text):
    # 用结巴对中文字符串进行分词
    text = _____
    return text
```

解析：text = " ".join(list(jieba.cut(text)))

5. （判断）TF-IDF 作用：用以评估一字词对于一个文件集或一个语料库中的其中一份文件的重要程度。

解析：正确

TF-IDF 的主要思想是：如果某个词或短语在一篇文章中出现的概率高，并且在其他文章中很少出现，则认为此词或者短语具有很好的类别区分能力，适合用来分类。

6. （填空题）假如一篇文件的总词语数是 100 个，而词语"非常"出现了 5 次，那么"非常"一词在该文件中的词频就是（）。而计算文件频率（IDF）的方法是以文件集的文件总数，除以出现"非常"一词的文件数。所以，如果"非常"一词在 1,000 份文件出现过，而文件总数是 10,000,000 份的话，其逆向文件频率就是（）。最后"非常"对于这篇文档的 tf-idf 的分数为（）

解析：

5/100=0.05

lg（10,000,000 / 1,0000）=3

0.05 * 3=0.15

词频（term frequency，tf）指的是某一个给定的词语在该文件中出现的频率

逆向文档频率（inverse document frequency，idf）是一个词语普遍重要性的度量。某一特定词语的 idf，可以**由总文件数目除以包含该词语之文件的数目，再将得到的商取以 10 为底的对数得到**

$$\mathrm{tfidf}_{i,j} = \mathrm{tf}_{i,j} \times \mathrm{idf}_i$$

最终得出结果可以理解为重要程度。

7. Word2Vec 案例练习

由于语料比较大，就提供了一个下载地址：http://www.sogou.com/labs/resource/cs.php

搜狗新闻中文语料(2.7G)

做中文分词处理之后的结果

保存加载模型，完成以下测试：

```
model.most_similar("警察")
model.similarity('男人','女人')
model.most_similar(positive=['女人', '丈夫'], negative=['男人'], topn=1)
```

解析：

步骤：

1、训练模型

2、测试模型结果

代码：

训练模型 API

```
from gensim import Word2Vec
Word2Vec(LineSentence(inp), size=400, window=5, min_count=5)
```

LineSentence(inp)：把 word2vec 训练模型的磁盘存储文件

转换成所需要的格式,如：[[“sentence1”],[”sentence1”]]

size：是每个词的向量维度

window：是词向量训练时的上下文扫描窗口大小，窗口为 5 就是考虑前 5 个词和后 5 个词

min-count：设置最低频率，默认是 5，如果一个词语在文档中出现的次数小于 5，那么就会丢弃

方法：

inp:分词后的文本

save(outp1):保存模型

训练的代码如下

```
    if len(sys.argv) < 3:
        sys.exit(1)
    # inp 表示语料库(分词)，outp：模型
    inp, outp = sys.argv[1:3]

    model = Word2Vec(LineSentence(inp), size=400, window=5, min_count=5, workers=multiprocessing.cpu_count())

    model.save(outp)
```

```
import sys
import multiprocessing

from gensim.models import Word2Vec
from gensim.models.word2vec import LineSentence

if __name__ == '__main__':

    if len(sys.argv) < 3:
        sys.exit(1)

    # inp 表示语料库(分词)，outp：模型
    inp, outp = sys.argv[1:3]

    model = Word2Vec(LineSentence(inp), size=400, window=5, min_count=5, workers=multiprocessing.cpu_count())

    model.save(outp)
```

运行命令

python trainword2vec.py ./corpus_seg.txt ./model/*

指定好分词的文件以及，保存模型的文件

加载模型测试代码

```
model = gensim.models.Word2Vec.load("*.model")
model.most_similar('警察')
```

```
model.similarity('男人','女人')
most_similar(positive=['女人', '丈夫'], negative=['男人'], topn=1)
improt gensim
gensim.models.Word2Vec.load("./model/corpus.model")

model.most_similar("警察")

Out:
[('警员', 0.6961891651153564),
 ('保安人员', 0.6414757370948792),
 ('警官', 0.6149201989173889),
 ('消防员', 0.6082159876823425),
 ('宪兵', 0.6013336181640625),
 ('保安', 0.5982533693313599),
 ('武警战士', 0.5962344408035278),
 ('公安人员', 0.5880240201950073),
 ('民警', 0.5878666639328003),
 ('刑警', 0.5800305604934692)]


model.similarity('男人','女人')
Out: 0.8909852730435042


model.most_similar(positive=['女人','丈夫'], negative=['男人'], topn=1)
Out: [('妻子', 0.7788498997688293)]
```