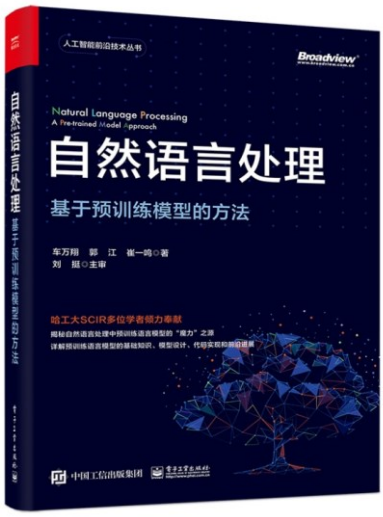


作者：车万翔，郭江，崔一鸣　著；刘挺主审
书号：ISBN 978-7-121-41512-8
出版社：电子工业出版社
定价：118.00 元
出版时间：2021.7

说明：本文档为《自然语言处理：基于预训练模型的方法》一书的章后习题参考答案。习题答案仅供参考。如对其中的答案有疑问，请联系本书责任编辑宋亚东，联系邮箱为 syd@phei.com.cn。

第 2 章

2.1.基于规则与基于机器学习的自然语言处理方法分别有哪些优缺点？

基于规则的方法：

优点：1.在特定的领域可以取得比较精确的结果。2.比较容易人工干预修改。

缺点：1.需要专业的语言学家进行大量规则设置。2.一个领域编写的规则难以迁移到新的领域。

基于机器学习的方法：

优点：1.不需要手工编写复杂规则。2.充分利用大量数据的语义信息。3.拥有较强的泛化能力和鲁棒性。

缺点：1.需要大量的数据进行参数估计。

2.2.如何在词的独热表示中引入词性、词义等特征？请举例说明。

利用 WordNet 提供的词性特征以及 synsets 上位词信息。

对于词性特征，我们可以单独对词性构建一个独热表示，随后将词的独热表示与词性独热表示进行拼接以融合词性特征。

对于词义特征，我们可以利用 synsets 的上位词信息。对所有的上位词构建一个独热表示与原单词表示进行拼接。

2.3.奇异值分解方法是如何反映词之间的高阶关系的？

利用奇异值分解方法，我们得到了低维、稠密的词向量。且词向量各列正交，每个维度表达着独立的“潜在语义”。原本具备高阶关系的词经过浓缩，其对应特征也会存在较高的相似度，从而反映了高阶关系。

2.4.在使用式 (2-18) 计算困惑度时，如果其中的某一项概率为 0，如何处理？

可以用一个较小值对含零项进行代替以避免出现正无穷的结果。

2.5.若使用逆向最大匹配算法对句子“研究生命的起源”进行分词，结果是什么？是否可以说明逆向最大匹配算法要优于正向最大匹配算法？

结果：研究 生命 的 起源

单从一个例子并不能说明逆向最大匹配算法优于正向最大匹配算法。

2.6.2.2.2.2 节介绍的子词切分算法是否可以用于中文？若能应用，则与中文分词相比有哪些缺点？

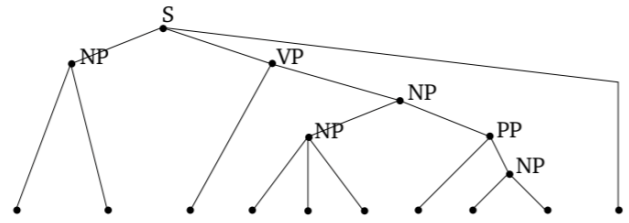
可以，基于词表的方法是从整体进行拆分，而基于 BPE 的算法则是从字到词进行聚合。

优点：不需要人工设计词表，充分利用语料中的共现信息。

缺点：在某些罕见的专有名词的识别精度上不如词表方法准确。需要大量的语料才能获得好的效果。

2.7.是否可以使用序列标注方法解决句法分析（短语结构和依存两种）问题？若能使用，则如何进行？

对于短语结构树，可以将树结构转化成序列结构，例如下图所示，论文《Constituent Parsing as Sequence Labeling》（https://aclanthology.org/D18-1162v2.pdf）中，每一个序列位置输出相邻两个节点的共同祖先个数以及最近公共祖先。



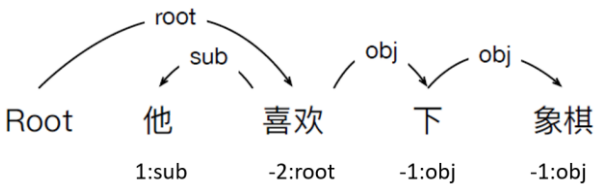
Linearized tree (absolute scale):

2NP 1S 2VP 4NP 4NP 3NP 4PP 5NP 1S

Linearized tree (relative scale):

2NP -1S 1VP 2NP 0NP -1NP 1PP 1NP -4S

对于依存句法树，如下图所示我们可以将每个序列位置的输出设置成分析树中的父节点相对位置以及依存边的种类。



2.8.使用何种评价方法评价一个中文分词系统？并请编程实现该评价方法。

使用 F 值进行判断。其中精确率表示预测分词结果中正确划分的比例。召回率表示标准分词结果中被成功切分的比例。

```
1. def to_region(split):
2.     '''''
3.     将划分的单词修改为长度区间
4.     Args:
5.         split: list of list of strings
6.
7.     Returns:
8.         regions: list of tuple(代表区间)
9.
10.    '''
11.    regions = []
12.    start = 0
13.    for word in split:
14.        regions.append((start, start+len(word)))
15.        start += len(word)
16.    return regions
17.
18.def compute_F1(pred_split, gold_split):
19.    """
20.    计算预测结果 pred_split 的 F1 值
21.    Args:
22.        pred_split: list of list of string
23.        gold_split: list of list of string
24.
25.    Returns:
26.        F1: F1 score
27.
28.    """
29.    pred_idx = set(to_region(pred_split))
30.    gold_idx = set(to_region(gold_split))
31.    correct_idx = pred_idx & gold_idx
32.    P = len(correct_idx) / len(pred_idx)
33.    R = len(correct_idx) / len(gold_idx)
34.    F1 = 2 * P * R / (P + R)
35.    return F1
36.
37.print(compute_F1(['我', '喜', '欢', '你'], ["我", "喜欢", "你"]))
```

第 3 章

注：nltk 可能遇到数据无法下载的情况，可以访问 http://www.nltk.org/nltk_data/直接下载 Project Gutenberg Selections、Stopwords Corpus、WordNet、Averaged Perceptron Tagger 和 SentiWordNet。

3.1.使用 NLTK 工具下载简·奥斯汀所著的 Emma 小说原文，并去掉其中的停用词。

```
1. import nltk
2. emma = nltk.corpus.gutenberg.words('austen-emma.txt')
3. stopwords = nltk.corpus.stopwords.words('english')
4. print(stopwords)
5.
6. emma = [w.lower() for w in emma]
7. emma_without_stopwords = [w for w in emma if w not in stopwords]
8. print(emma_without_stopwords)
```

3.2.使用 NLTK 提供的 WordNet 计算两个词（不是词义）的相似度，计算方法为两词各种词义之间的最大相似度。

```
1. from nltk.corpus import wordnet
2. word1 = 'dog'
3. word2 = 'cat'
4. word1_synsets = wordnet.synsets(word1)
5. word2_synsets = wordnet.synsets(word2)
6.
7. result = max([w1.path_similarity(w2) for w1 in word1_synsets for w2 in word2_synset
s])
8. print(result)
9. # result: 0.2
```

3.3.使用 NLTK 提供的 SentiWordNet 工具计算一个句子的情感倾向性，计算方法为每个词所处词性下的每个词义情感倾向性之和。

```
1. import nltk
2. sentence = ['welcome', 'to', 'harbin', 'institute', 'of', 'technology']
3.
4. sentence_tag = nltk.pos_tag(sentence)
5. tag_map = {'NN': 'n', 'NNP': 'n', 'NNPS': 'n', 'NNS': 'n', 'UH': 'n',\
6.            'VB': 'v','VBD': 'v','VBG': 'v','VBN': 'v','VBP': 'v','VBZ': 'v',\
7.            'JJ': 'a','JJR': 'a','JJS': 'a',\
8.            'RB': 'r', 'RBR': 'r', 'RBS': 'r', 'RP': 'r', 'WRB': 'r'}
```

```
9. sentence_tag = [(t[0], tag_map[t[1]]) if t[1] in tag_map else (t[0], '') for t in s
entence_tag]
10.
11.sentiment_synsets = [list(nltk.corpus.sentiwordnet.senti_synsets(t[0], t[1])) for t
in sentence_tag]
12.score = sum(sum([x.pos_score() - x.neg_score() for x in s]) / len(s) for s in senti
ment_synsets if len(s) != 0)
13.
14.print(score)
15.# result: 0.3125
```

3.4.使用真实文本对比 LTP 与正向最大匹配分词的结果，并人工分析哪些结果 LTP 正确，正向最大匹配错误；哪些结果 LTP 错误，正向最大匹配正确；以及哪些结果两个结果都错误。

```
1. import ltp
2. sentence = ['南京市长江大桥', '行行行', '结婚的和尚未结婚的确实在干扰分词啊']
3. ltp_model = ltp.LTP()
4. segment, _ = ltp_model.seg(sentence)
5. print(segment)
```

前向最大匹配算法（FFM）代码略。

结果：

以下蓝色表示正确，红色表示错误。

LTP：['南京市', '长江大桥'], ['行行行'], ['结婚', '的和尚未', '结婚', '的确实在', '干扰分词', '啊']

FFM：['南京市长', '江', '大桥'], ['行行', '行'], ['结婚', '的', '和尚', '未', '结婚', '的确', '实在', '干扰', '分词', '啊']

3.5.分析 view、reshape、transpose 和 permute 四种调整张量形状方法各自擅长处理的问题？

view 与 reshape 用于设置新的张量形状；但是 view 只能操作连续存储的张量，对于非连续的张量（如在 view 之前进行 transpose、permute 等操作），使用 reshape 进行调整，或是通过 contiguous 返回一个连续的 copy 后再进行 view；torch.reshape()的功能大致相当于 tensor.contiguous().view()。

```
1. a = torch.randn(2,3,4)
2. print(a.shape)
3. # torch.Size([2, 3, 4])
4. a = a.permute(1,0,2)
```

```
print(a.shape)
# torch.Size([3, 2, 4])

# 不连续无法直接使用 view
print(a.view(2,3,4))
# Traceback (most recent call last):
#   File "<stdin>", line 1, in <module>
#   RuntimeError: view size is not compatible with input tensor's size and stride ( at least one dimension spans across two contiguous subspaces). Use .reshape(...) instead.


# 使用 reshape 解决问题
print(a.reshape(2,3,4).shape)
    # torch.Size([2, 3, 4])
```

transpose 与 permute 用于交换张量的维度顺序，包含转置的概念；transpose 只能交换张量中的两个维度，若想要更多维度的交换，需要多次调用 transpose 或直接调用 permute。

```
a = torch.rand(3,4,5)

# 使用多次 transpose
b = a.transpose(0, 1).transpose(1, 2)
print(b.shape)
# torch.Size([4, 5, 3])

# 使用 permute 一次解决
c = a.permute(1, 2, 0)
    print(c.shape)
    # torch.Size([4, 5, 3])
```

综上所述，对于连续张量改变形状使用 view，对于不连续张量使用 reshape；对于张量做两个维度的交换使用 transpose，更多维度的交换使用 permute。

3.6.安装 PyTorch 并实际对比使用和不使用 GPU 时，三个大张量相乘时的效率？

参考本章 3.3.2 中例子使用 timeit 模块进行测时，测试环境为 GTX 1070 Ti 8G 显存，在使用和不使用 GPU 加速时，三个大张量相乘 500 次所需平均时间不在同一数量级，使用 CPU 计算耗时 9.4581ms，而 GPU 加速下计算仅耗时 0.0218ms；可见 GPU 加速大大提升了计算的效率。

```
import timeit
it1 = timeit.Timer('M.mm(M).mm(M)', 'import torch\nM = torch.rand(1000, 1000)')
t1 = it1.timeit(500)/500
print('{:.4f}ms'.format(t1*1000))
# CPU 计算耗时 9.4581ms
it2 = timeit.Timer('M.mm(M).mm(M)', 'import torch\nM = torch.rand(1000, 1000).cuda()')
```

```
t2 = it2.timeit(500)/500
print('{:.4f}ms'.format(t2*1000))
# GPU 计算耗时 0.0218ms
```

3.7.下载最新的 Common Crawl 数据，并实现抽取中文、去重、简繁转换、数据清洗等功能。

数据集介绍：Common Crawl 包含了超过 7 年的网络爬虫数据集，包含原始网页数据、元数据提取和文本提取。常见的爬行数据存储在 Amazon Web 服务的公共数据集和遍布全球的多个学术云平台上，拥有 PB 级规模，常用于学习词嵌入。

数据集格式：Common Crawl 数据集官网提供多种格式数据集的免费下载，包含原始数据的 WARC 格式、WARC 格式记录的重要元数据 WAT 格式、从网页中提取出的纯文本 WET 格式等等；对于自然语言处理任务选用包含文本信息的 WET 格式数据，其中数据集中爬取的每个网页信息格式为数据集头信息+\n+网页纯文本信息；每个网页信息间隔两个\n。

抽取中文：根据 Unicode 码实现抽取中文，常用中文编码的范围是：\u4e00 到\u9fff；基本拉丁语的范围是：\u0020'到\u007f；一般标点符号的范围是：\u2000 到\u206f；CJK 符号和标点符号的范围是：\u3000 到\u303f；半角和全角形式符号的范围是：\uff00 到\uffef；因此根据这一条件使用正则表达式完成中文的抽取。

```
def translate(str):
    pattern = re.compile('[^\u4e00-\u9fa50-9\s]')
    # 中文的编码范围是：\u4e00 到\u9fa5
    str = re.sub(pattern, '', str)
    return str
```

去重：对于数据集的文本去重，要根据应用任务具体分析来决定去重粒度，将原始数据集按行或按各个网页的文本进行去重；去重可以采用 set 或 list 数据结构维护当前信息来实现，为了提高效率，对每行文本做 hash 映射，在 hash 冲突后再进行详细比较。

```
import sys
f_in = open(sys.argv[1], "r") # 输入文件
lines_dic = {}
for line in f_in.readlines():
    line = line.strip()
    hashcode = hash(line)
    if hashcode not in lines_dic.keys():
        lines_dic[hashcode] = [line]
        print(line)
    elif line not in lines_dic[hashcode]:
        lines_dic[hashcode].append(line)
        print(line)
f_in.close()
```

繁简转换：采用书中提及的中文繁简体转换工具 OpenCC 进行数据的繁简转换，中文繁简转换同参考本书中 3.4.3 节脚本 convert_t2s.py。

```
import sys
import opencc
converter = opencc.OpenCC("t2s.json") # 载入繁简转换配置文件
f_in = open(sys.argv[1], "r") # 输入文件
for line in f_in.readlines():
    line = line.strip()
    line_t2s = converter.convert(line)
    print(line_t2s)
```

数据清洗：同样作为网页中提取的纯文本数据，其中也会因为文本编码、损坏 html 标签等问题导致乱码或机器字符，因此采取的数据清洗方法参考本书中 3.4.3 节对于维基百科的脚本 wikidata_cleaning.py。

```
import sys
import re
def remove_empty_paired_punc(in_str):
    return in_str.replace('()', '').replace('《》', '').replace('【】', '').replace('[]', '')

def remove_html_tags(in_str):
    html_pattern = re.compile(r'<[^>]+>', re.S)
    return html_pattern.sub('', in_str)

def remove_control_chars(in_str):
    control_chars = ''.join(map(chr, list(range(0, 32)) + list(range(127, 160))))
    control_chars = re.compile('[%s]' % re.escape(control_chars))
    return control_chars.sub('', in_str)
```

最终数据预处理脚本如下，对数据逐行做中文提取、简繁转换、数据清洗以及去重工作；得到预处理数据集。

```
import sys

f_in = open(sys.argv[1], 'r') # 输入文件
lines_dic = {}
for line in f_in.readlines():
    line = line.strip()
    # 数据清洗
    line = remove_empty_paired_punc(line)
    line = remove_html_tags(line)
    line = remove_control_chars(line)
    # 中文抽取
    line = translate(line)
    # 简繁转换
    line = converter.convert(line)
```

```
15.     # 去重
16.     hashcode = hash(line)
17.     if hashcode not in lines_dic.keys():
18.         lines_dic[hashcode] = [line]
19.         print(line)
20.     elif line not in lines_dic[hashcode]:
21.         lines_dic[hashcode].append(line)
22.         print(line)
```

第 4 章

4.1.试证明 Sigmoid 函数 y = $\frac{1}{1+e^{-x}}$的导数为 y′ = y(1 -y)。

$$y = \frac{1}{1+e^{-x}} = \frac{e^{x}}{e^{x}+1} = 1-(e^{x}+1)^{-1}$$

$$y' = -(-1)(e^{x}+1)^{-2}[e^{x}]' = (e^{x}+1)^{-2}e^{x} = \frac{e^{x}}{(e^{x}+1)^{2}} = \frac{e^{x}}{e^{x}+1}\frac{1}{e^{x}+1} = \frac{1}{1+e^{-x}}\frac{e^{-x}}{1+e^{-x}}$$

$$= \frac{1}{1+e^{-x}}\frac{(1+e^{-x})-1}{1+e^{-x}} = \frac{1}{1+e^{-x}}(1-\frac{1}{1+e^{-x}}) = y(1-y)$$

4.2.式 (4-5) 中，如何解决z_i过大，导致e^{z^i}数值溢出的问题？

根据如下公式：

$$\mathbf{Softmax}_z(z^i) = \frac{e^{z^i}}{\sum_{j=1}^{m} e^{z_j}} = \frac{\frac{e^{z^i}}{e^{M}}}{\sum_{j=1}^{m}\frac{e^{z_j}}{e^{M}}} = \frac{e^{(z^i-M)}}{\sum_{j=1}^{m} e^{(z_j-M)}} = \mathbf{Softmax}_{z-M}(z^i-\mathrm{M})$$

令 M=max(z_j),j=1,2,…,m，即 M 为所有z_j 中最大的值，我们只计算Softmax$_{z-M}(z^i-\mathrm{M})$的值.通过这样的变换，对任何一个z_j，减去 M 之后，e 的指数z_j-M 的最大值为 0，所以不会发生溢出。

4.3.若去掉式 (4-11) 中的 ReLU 激活函数，该多层感知器是否还能处理异或问题？为什么？

不能，因为异或问题不是线性可分的，无法使用线性分类器恰当地将输入划分到正确的类别。如果去掉了非线性的 ReLU 激活函数，原多层感知器将会变成两个线性变换的叠加，得到的还是一个线性变 x 换，依然无法对异或问题的非线性可分空间进行分割。但引入非线性激活函数（ReLU），通过设置恰当的参数值，其将原非线性可分的空间映射映射到新的隐含层空间，可使其在该空间内线性可分。

4.4.在使用卷积神经网络时，如何解决有效特征长度大于卷积核宽度的问题？

可以通过增加卷积层来扩大视野。比如有效特征长度为 6，卷积和宽度为 3，我们就可以设计 3 层卷积。

也可以在卷积之后用池化操作来扩大感受野。

4.5.循环神经网络中，各时刻共享权重的机制有何优缺点？

优点：一方面，权重共享使得模型泛化，能够处理不定长的输入序列。在用 RNN 做 NLP 相关任务时，输入是一个连续的序列，长度是不确定的，基于时刻分配权值显然是不行的，那我们就可以通过权值共享来解决这个问题。具体而言，比如，“我喜欢踢足球”和“我喜欢和朋友们一起踢足球”，句子长度不一样，但表达的中心思想实际上差不多。另一方面，权值共享的参数量少。

缺点：表达能力偏弱。展开后相当于多层权重共享的 MLP，表达能力有所降低。

4.6.在处理长距离依赖关系时，原始的循环神经网络与长短时记忆网络（LSTM）在机

制上有何本质的区别？

原始的 RNN 的隐藏层公式为：

（序列长度为 n ）

$$\boldsymbol{z}_t = \boldsymbol{W}^{\mathrm{xh}}\boldsymbol{x}_t + \boldsymbol{b}^{\mathrm{xh}} + \boldsymbol{W}^{hh}\boldsymbol{h}_{t-1} + \boldsymbol{b}^{hh}$$

$$\boldsymbol{h}_t = \tanh(\boldsymbol{z}_t)$$

$$\boldsymbol{o} = \boldsymbol{W}^{\mathrm{hy}}\boldsymbol{h}_n + \boldsymbol{b}^{\mathrm{hy}}$$

$$\boldsymbol{y} = \mathrm{Softmax}(\boldsymbol{o})$$

设最终定义某种损失函数为 $\boldsymbol{L}$ 。

在反向传播的时候，对于 $\boldsymbol{W}^{\mathrm{hy}}, \boldsymbol{b}^{\mathrm{hy}}$ 的求导根据链式法则很容易得到，不再赘述，考虑 $\boldsymbol{W}^{\mathrm{xh}}, \boldsymbol{W}^{hh}$ ，因为共享权重，这两个矩阵的链式求导后很复杂，以 $\boldsymbol{W}^{\mathrm{xh}}$ 为例：

$$\frac{\partial \boldsymbol{L}}{\partial \boldsymbol{W}^{\mathrm{xh}}} = \frac{\partial \boldsymbol{L}}{\partial \boldsymbol{o}}\frac{\partial \boldsymbol{o}}{\partial \boldsymbol{h}_n}\frac{\partial \boldsymbol{h}_n}{\partial \boldsymbol{z}_n}\frac{\partial \boldsymbol{z}_n}{\partial \boldsymbol{W}^{\mathrm{xh}}} + \frac{\partial \boldsymbol{L}}{\partial \boldsymbol{o}}\frac{\partial \boldsymbol{o}}{\partial \boldsymbol{h}_n}\frac{\partial \boldsymbol{h}_n}{\partial \boldsymbol{z}_n}\frac{\partial \boldsymbol{z}_n}{\partial \boldsymbol{h}_{n-1}}\frac{\partial \boldsymbol{h}_{n-1}}{\partial \boldsymbol{z}_{n-1}}\frac{\partial \boldsymbol{z}_{n-1}}{\partial \boldsymbol{W}^{\mathrm{xh}}}$$
$$+ \frac{\partial \boldsymbol{L}}{\partial \boldsymbol{o}}\frac{\partial \boldsymbol{o}}{\partial \boldsymbol{h}_n}\frac{\partial \boldsymbol{h}_n}{\partial \boldsymbol{z}_n}\frac{\partial \boldsymbol{z}_n}{\partial \boldsymbol{h}_{n-1}}\frac{\partial \boldsymbol{h}_{n-1}}{\partial \boldsymbol{z}_{n-1}}\frac{\partial \boldsymbol{z}_{n-1}}{\partial \boldsymbol{h}_{n-2}}\frac{\partial \boldsymbol{h}_{n-2}}{\partial \boldsymbol{z}_{n-2}}\frac{\partial \boldsymbol{z}_{n-2}}{\partial \boldsymbol{W}^{\mathrm{xh}}} + \cdots$$

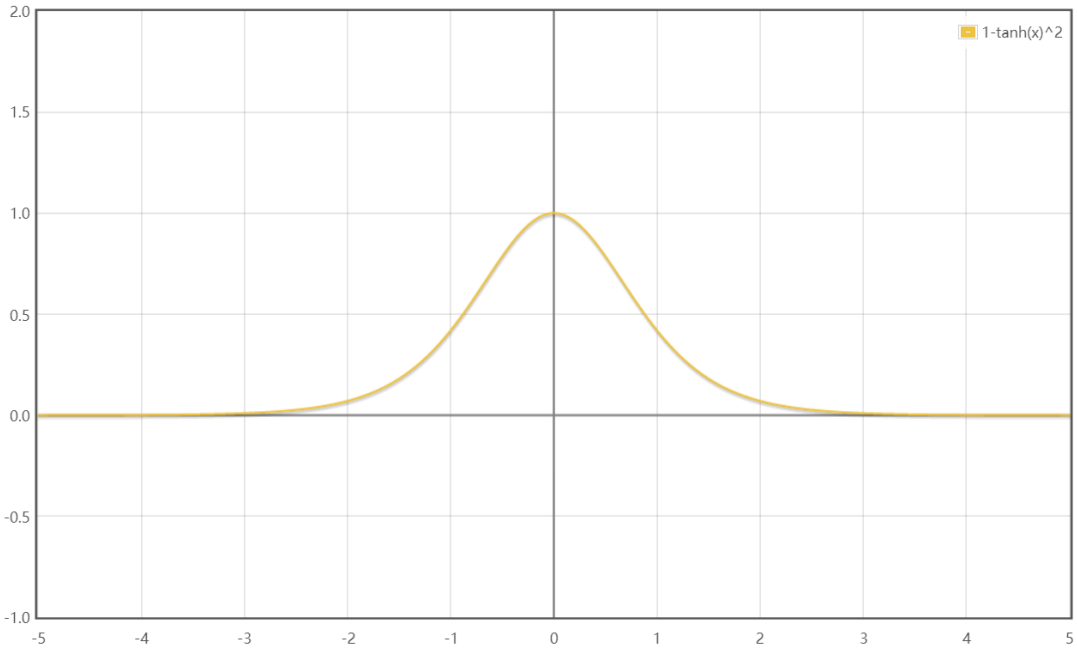
可以看到，$\boldsymbol{W}^{\mathrm{xh}}$ 的偏导随着时间序列会产生长期依赖。因为 $\boldsymbol{h}_t$ 是随着时间序列向前传播，而 $\boldsymbol{h}_t$ 又是 $\boldsymbol{W}^{\mathrm{xh}}$ 的函数。根据上述推导过程，

$$\frac{\partial \boldsymbol{L}}{\partial \boldsymbol{W}^{\mathrm{xh}}} = \sum_{\mathrm{k}=0}^{n}\frac{\partial \boldsymbol{L}}{\partial \boldsymbol{o}}\frac{\partial \boldsymbol{o}}{\partial \boldsymbol{h}_n}\left(\prod_{\mathrm{j}=\mathrm{k}+1}^{\mathrm{n}}\frac{\partial \boldsymbol{h}_j}{\partial \boldsymbol{z}_j}\frac{\partial \boldsymbol{z}_j}{\partial \boldsymbol{h}_{j-1}}\right)\frac{\partial \boldsymbol{h}_k}{\partial \boldsymbol{z}_k}\frac{\partial \boldsymbol{z}_k}{\partial \boldsymbol{W}^{\mathrm{xh}}}$$

进一步转化

$$\prod_{\mathrm{j}=\mathrm{k}+1}^{\mathrm{n}}\frac{\partial \boldsymbol{h}_j}{\partial \boldsymbol{z}_j}\frac{\partial \boldsymbol{z}_j}{\partial \boldsymbol{h}_{j-1}} = \prod_{j=k+1}^{\mathrm{n}}\tanh'\ \boldsymbol{W}^{hh} = \prod_{\mathrm{j}=\mathrm{k}+1}^{\mathrm{n}}\frac{\partial \boldsymbol{h}_j}{\partial \boldsymbol{h}_{j-1}}$$

观察 $\tanh' = 1 - tanh^2$ 的图像:



发现 $\tanh' \leq 1$ 。在训练过程中，$\tanh'$ 绝大多数情况下都是小于 1 的。

当 $0 \leq \boldsymbol{W}^{hh} \leq 1$ 时，$\prod_{j=k+1}^{\mathrm{n}}\tanh'\ \boldsymbol{W}^{hh}$ 就会趋近于 0，发生梯度消失；

当 $\boldsymbol{W}^{hh}$ 很大时时，$\prod_{j=k+1}^{\mathrm{n}} \tanh' \boldsymbol{W}^{hh}$ 就会趋近于无穷，发生梯度爆炸。

在发生梯度消失的时候，实际意义就是长距离的依赖被忽略了，并没有体现在权值矩阵的修改上。

在正向传播的过程中，也可以对长距离依赖丢失有所理解。在算 $\boldsymbol{h}_{\mathrm{t}}$ 时，$\boldsymbol{h}_{\mathrm{t}-1}$ 不仅代表着前一时刻，$\boldsymbol{h}_{\mathrm{t}-2}, \boldsymbol{h}_{\mathrm{t}-3,} \cdots$ 也是通过 $\boldsymbol{h}_{\mathrm{t}-1}$ 传递到当前时刻的，展开后可以得到 $\boldsymbol{h}_{t-2}, \boldsymbol{h}_{t-3,} \cdots$ 前面是累乘的 $\boldsymbol{W}^{hh}$，那么当 $\boldsymbol{W}^{hh} < 1$ 时，多次累乘之后乘积逼近于 0，意味着较远时刻损失了大量信息，传递过来的信息很少。

而 LSTM 首先将隐藏层更新公式修改为：

$$\boldsymbol{u}_t = \tanh(\boldsymbol{W}^{\mathrm{xh}} \boldsymbol{x}_t + \boldsymbol{b}^{\mathrm{xh}} + \boldsymbol{W}^{\mathrm{hh}} \boldsymbol{h}_{t-1} + \boldsymbol{b}^{\mathrm{hh}})$$

$$\boldsymbol{h}_t = \boldsymbol{h}_{t-1} + \boldsymbol{u}_t$$

在 RNN 中，长距离依赖的消失在于 $\frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{h}_{t-1}}$ 的大小。同理，我们来观察 LSTM 中 $\frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{h}_{t-1}}$ 的大小。

$$\frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{h}_{t-1}} = 1 + \frac{\partial \boldsymbol{u}_t}{\partial \boldsymbol{h}_{t-1}}$$

无论 $\frac{\partial \boldsymbol{u}_t}{\partial \boldsymbol{h}_{t-1}}$ 多少，可以保证 $1 \leq \frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{h}_{t-1}}$。那么即便是累乘之后，长距离的依赖的导数与当前依然是大于等于 1 的，也就不会出现梯度消失的情况。

也可以展开累加项来理解 $\boldsymbol{k} < \boldsymbol{t}$

$$\boldsymbol{h}_t = \boldsymbol{h}_{t-1} + \boldsymbol{u}_t = \boldsymbol{h}_{t-2} + \boldsymbol{u}_{t-1} + \boldsymbol{u}_t = \boldsymbol{h}_k + \boldsymbol{u}_{k+1} + \boldsymbol{u}_{k+2} + \cdots + \boldsymbol{u}_{t-1} + \boldsymbol{u}_t$$

也可以看出，$\boldsymbol{h}_t$ 与 $\boldsymbol{h}_k$ 有直接相连，跨过了中间的 $(\boldsymbol{t} - \boldsymbol{k})$ 层，使长期依赖依然有效。

4.7.在 Transformer 中，使用绝对位置的嵌入或编码有何缺点？针对该缺点有何解决方案？

绝对位置的嵌入或编码是比较朴素的融入位置信息的方法。它的缺点有：

第一，绝对位置编码相比相对位置编码对句子语义的影响不大，单词的相对位置才对语义有着关键性影响。

第二，位置嵌入为序列中每个绝对位置赋予一个连续、低维、稠密的向量表示，其位置向量只依赖于位置编号，直接将位置编码当做可训练参数。但其缺点为外推性不足，即如果预训练的最大长度为 N，那么该方法最多只能处理长度为 N 的句子，再长便无法处理了。

第三，本书的位置编码方法则将位置索引值映射到一个 d 维向量上（利用三角函数），一般称为 Sinusoidal 位置编码。提出该方法的作者希望借助形似三角函数的绝对位置的编码公式，让模型能够学习到相对位置信息。通过三角变换公式（$\sin(\alpha+\beta)$与$\cos(\alpha+\beta)$），作者证明了$PosEnc(p+k,i)$可以表示为$PosEnc(p,i)$与$PosEnc(p,i-1)$的线性表示，借此提供了表达相对位置信息的可能性。此外，作者证明了两个不同位置的位置编码之间的点积可以反映相对距离，但缺乏方向性，并且这种特性（相对距离）会被原始 Transformer 的注意力机制破坏（可参考论文《Attention is All You Need》）。

解决方案是改用真正的相对位置表示方法，即不完整建模每个输入的位置信息，在计算注意力机制时考虑当前位置与被 Attention 的位置的相对距离。相对位置编码的解决方案可参考 Google 论文《Self-Attention with Relative Position Representations》。此外，除了绝对位置表示与相对位置表示，目前有着许多其他的位置编码方法，如复数式位置编码（可参考 ICLR 2020 论文《Encoding Word Order in Complex Embeddings》）等。

4.8.实际运行本章处理情感分类和词性标注问题的代码，并对比各种模型的准确率，然后尝试使用多种方法提高每种模型的准确率。

提高模型准确率的方法可能有：修改学习率，更换优化器，增加 epoch 训练轮次等。

而针对具体的模型，可以尝试调节模型的具体超参数，如卷积核大小与个数等。

第 5 章

5.1-5.3

请读者自行实验。

5.4.分别从词表示以及实际应用两个角度分析静态词向量的优缺点。并针对其缺点，思考并提出一种合理的解决方案。

从词表示的角度，优点：1）分布式表示，能够较好表示单词的语义；2）具有一定的语义关系表达能力；3）易于扩展，通过语料选择、模型及参数设计等方式可以获得具有不同性质的词表示；4）通过语义组合，可以获得更大粒度语义单元（如短语、句子）的表示。缺点：1）无法表达词的多义性；2）可解释性差，各维度值的含义无从得知；3）对于低频词的表达能力弱。

从实际应用的角度，优点：1）维度低，易于存储和计算；2）可学习，可作为参数在下游任务中进行学习，从而获得与任务更相关的词表示；3）提供了一种简单有效的半监督学习手段。缺点：上下文无关，表达能力有限。

5.5.提出一种针对低频词的词向量学习改进方案。

引入子词信息，如字符或字（中文）。

引入外部知识，如词典。

5.6.将预训练词向量用于目标任务时，什么情形下，“冻结”词向量比精调词向量更合理？在情感分类任务上进行验证。

低资源场景（目标任务训练数据小）或者少样本学习场景。

预训练词向量的词表空间与目标任务训练数据词表空间重叠较小的场景。

第 6 章

6.1.分别从词表示和语义组合的角度阐述动态词向量的特点，以及其相比于静态词向量的优势。

特点：1）动态：词表示向量由其当前上下文决定；2）鲁棒（或稳健性）：对于未登录词（OOV）或低频词具有较强表达能力；3）层次（或粒度）：通过对不同层次的向量表示进行组合，可以适配不同的下游任务。

6.2–6.4

请读者自行实验。

6.5.为了训练中文上的 ELMo 模型，需要对模型结构进行哪些调整？

中文属于形态学不丰富的语种而且单词长度往往较小，因此，中文 ELMo 模型的输入表示层结构（如：是否使用 CNN，卷积核的大小）需要进行适当的调整。另外，不同的分词策略、粒度也会影响模型的表现，在模型设计时需要纳入考量。

6.6.除了以特征形式应用于下游任务，动态词向量还有哪些潜在的应用场景？

可用于语义相关的研究，例如词义消歧、探索词义随时间的延伸及变化（如“小米”）、新词义发现等。

第 7 章

7.1.从模型的角度对比分析 GPT 和 BERT 各自的优缺点是什么？

GPT 和 BERT 都是以 Transformer 为主体架构的预训练语言模型，都是通过“预训练+精调”的模式完成下游任务模型的搭建。

两者的主要区别和优缺点：

1）GPT 是单向模型，无法利用双向上下文，而 BERT 是双向模型；

2）GPT 是基于自回归的语言模型，能够应用在 NLU（自然语言理解）和 NLG（自然语言生成）类任务，而原生 BERT 是基于自编码的语言模型，只能完成 NLU 类任务，不能直接应用在文本生成类任务；

3）同等参数规模下，BERT 在 NLU 类效果优于 GPT。

7.2.阐述 BERT 的输入表示中为什么要包含位置向量？如果没有位置向量将有何影响？

传统的循环神经网络（RNN）中，每个时刻的隐层编码依赖于上一个时刻，即编码是有先后顺序的。因此模型能够了解哪些词是先出现的，哪些词是后出现的，从而了解整个序列中每个 token 的先后关系。

对于卷积神经网络来说，在每一个卷积窗内的单词之间是没有先后顺序的。然而，通过多层卷积以及池化层的操作后，高层网络中包含了其底层卷积窗内的信息（类似树状结构），因此能够获取一定的位置信息。

如果 BERT 的输入表示中不包含位置向量，那么 BERT 无法判别输入文本中每个 token 的位置。BERT 中的主干，即 Transformer 模型，从时间的维度上看，是一个与位置无关的设计。例如，如果不包含位置向量，句子“我 喜 欢 吃 苹 果”和“苹 果 喜 欢 吃 我”是完全等价的，然而这显然不满足编码需求，因为两个句子的含义是完全不同的。通过引入位置编码，对每个 token 的先后顺序进行编码，并通过位置向量矩阵查询得到对应的向量表示，以此来区分不同 token 的绝对位置关系。

需要注意的是 BERT 中 Transformer 的位置编码方法和 Transformer 原版的编码方法是不同的。Transformer 原版的位置编码方法是基于一种正弦函数的编码方式。感兴趣的读者可以阅读 Transformer 的原论文《Attention is all you need》进一步了解。

7.3.阐述应用三种不同掩码策略（MLM、WWM 和 NM）的 BERT，在预训练阶段和下游任务精调中的异同点。

下面通过一组例子来说明三者的异同点。

在预训练阶段，

原始文本	2022 冬季奥运会将在中国北京举行。
经过分词后	2022 冬 季 奥 运 会 将 在 中 国 北 京 举 行 。
MLM 输入	2022 [M] 季 奥 [M] 会 将 在 [M] 国 北 [M] 举 行 。
WWM 输入	2022 [M] [M] 奥 运 会 将 在 [M] [M] 北 京 举 行 。
NM 输入	2022 冬 季 奥 运 会 将 在 [M] [M] [M] [M] 举 行 。

可以看到，

1）MLM 的输入中，每个 token 是否被 mask 是完全独立的。

2）WWM 的输入中，如果一个 token 被 mask，那么同属于一个词的其他 token 也会被 mask。例如，“冬季”中的“冬”和“季”均被 mask；“中国”中的“中”和“国”均被 mask；

3）NM 的输入中，“中国”和“北京”均被 mask（此处是一个 bigram 的掩码）。

注意：以上仅考虑了替换成[M]的情况（80%概率），还有 10%可能会替换为词表中的一个随机词，另外还有 10%可能会保留原词。请结合本书 7.3.4 节中的（1）正确理解整词掩码”做进一步学习。

在下游任务精调阶段，不论预训练模型是由 MLM、WWM 还是 NM 训练而来的，其输入形式都是经过 WordPiece 分词所得。此处需要注意的是，

WWM 训练所得 BERT 并不要求在下游任务精调过程中将输入切分成词；

NM 训练所得 BERT 并不要求在下游任务精调过程中将输入组成为 N-gram。

综上所述，不同的掩码策略

只会影响预训练阶段的掩码 token 的选择方式；

不会影响预训练阶段和下游任务精调阶段的输入粒度，即仍然使用 WordPiece 分词后的序列作为输入。

7.4.BERT 中的 MLM 预训练任务采用了 15%的掩码概率，请阐述增大或减小掩码概率对预训练语言模型效果可能产生的影响。

如果增加掩码概率，输入文本中缺失部分增多，MLM 的文本恢复难度进一步提升。在一定范围内，增加掩码概率可能提高预训练效果，但掩码概率过高可能会影响预训练模型的学习。反之，如果减小掩码概率，文本中的缺失信息较少，也就是说可利用的上下文信息更丰富，使得模型更容易预测缺失位置的单词，不能很好地挖掘语义信息，导致预训练模型效果下降。

然而在 T5 论文中描述，掩码比例对下游任务的影响比较有限。论文中详细描述了 span-corruption 预训练任务的掩码比例对下游任务的影响。感兴趣的读者可阅读 T5 模型论文（https://arxiv.org/abs/1910.10683）中的 3.3.3 节和表格 6 做进一步了解。

7.5.以情感分类数据集 SST-2 为例 ,通过实验论证特征提取和模型精调两种 BERT 的典型应用方式对下游任务效果的影响。

略

7.6.在抽取式阅读理解任务中，篇章与问题的拼接顺序会对模型效果产生何种影响？请以具体的抽取式阅读理解任务 CMRC 2018 为例进行实验，并给出相应的实验结论。

略

第 8 章

8.1.阐述自回归语言模型和自编码语言模型的优缺点。

以下对比自回归语言模型和自编码语言模型的优缺点。此处只对比最原始的自回归和自编码语言模型。

自回归语言模型是最典型的语言模型类型，是一种单向建模的类型。其优点是建模方式自然，无需添加额外设置的预训练任务。缺点是如果不经过改造，原生自回归语言模型只能进行单向的建模。

自编码语言模型是一种双向建模的语言模型。其优点是可以利用合理的预训练任务的设计，使得模型天然地可以利用文本的上下文信息。但如果不经过改造，原生自编码语言模型（以 BERT 为代表）存在“预训练-精调”不一致的问题，同时设计一种高效的预训练任务也相对复杂。

8.2.阐述词向量因式分解与跨层参数共享对 ALBERT 模型解码时间的影响。

获取一个词的词向量是一个查表操作，与词向量维度的大小基本无关。而引入词向量因式分解后，需要利用一个额外全连接层将词向量维度还原为隐含层维度，因此相比不使用词向量因式分解的模型解码时间稍慢。

跨层参数共享虽然降低了模型的存储空间，即 Transformer 中的每层权重是一样，只需保存一层权重。但在解码时，输入文本仍然要经过 N 层 Transformer 的计算，因此不会节省解码时间。

8.3.相比传统通过文本切分的方式处理长文本，阐述长文本处理模型处理阅读理解和命名实体识别任务的优势。

在阅读理解或命名实体识别类任务中，当待处理文本的长度大于预训练模型可处理的最大长度时，需要将待处理文本进行切分，并将每一片输入到预训练模型得到预测结果，并对预测结果进行合并。

以抽取型阅读理解为例，如果预测答案的线索分散在多个文本块中，那么这种分片方式会导致多个线索之间无法直接在模型内部得到充分交互，因此会降低模型的性能。如果应用长文本处理模型，单次可处理的文本长度加长，这种情况就可以得到显著缓解。同时，长文本模型相比简单加长版本的普通预训练模型，在计算效率上要更高一些。

8.4.仿照 8.4.4 节中的介绍，尝试构造 GPT-3 在问答任务上的输入形式。

在小样本学习的情况下，以 SQuAD 2.0 为例

Context →	Title: The_Blitz Background: From the German point of view, March 1941 saw an improvement. The Luftwaffe flew 4,000 sorties that month, including 12 major and three heavy attacks. The electronic war intensified but the Luftwaffe flew major inland missions only on moonlit nights. Ports were easier to find and made better targets. To confuse the British, radio silence was observed until the bombs fell. X- and Y-Gerät beams were placed over false targets and switched only at the last minute. Rapid frequency changes were introduced for X-Gerät, whose wider band of frequencies and greater tactical flexibility ensured it remained effective at a time when British selective jamming was degrading the effectiveness of Y-Gerät. Q: How many sorties were flown in March 1941? A: 4,000 Q: When did the Luftwaffe fly inland missions? A:
Target Completion →	only on moonlit nights

更多样例请阅读 GPT-3 论文：https://arxiv.org/abs/2005.14165

8.5.仿照 7.4.2 节中的介绍，在 SST-2 数据集上，使用 RoBERTa-base 和 ELECTRA-base 模型训练单句文本分类模型，并对比两者的实验效果。

略

8.6.在 MNLI 数据集上，利用 TextBrewer 工具包实现 12 层 BERT-base-cased 模型蒸馏至 3 层的 BERT 模型，要求准确率不低于 81%。

略

第 9 章

9.1.阐述多语言预训练模型研究的主要意义与应用场景。

对于英语、中文等语言，我们往往拥有十分丰富的有标注的训练数据，然而对于一些低资源语言，往往很难获取大量的标注数据。通过多语言预训练模型，我们希望在同一个空间中表示不同的语言，建立起不同语言之间的关系，加强互相的表示能力。同时对于低资源语言，我们希望将高资源语言的知识进行迁移，提升模型表现。

9.2.使用 HuggingFace 提供的 transformers 库，分别实现基于 mBERT 模型与 XLM-R 模型的跨语言自然语言推理，并在相应的基准数据集 XNLI 上进行实验。

请同学们自行实现代码并进行实验。

9.3.试分析多媒体融合的预训练模型目前存在哪些主要挑战或瓶颈？结合最新的相关文献进行说明。

设计合适的训练任务：当前模型预训练任务主要包括掩蔽文本预测，掩蔽帧预测，视频-文本对齐等等。如何设计更加合适的训练任务捕获视频信息和文本信息之间的联系是一个重要的问题。

如何充分利用大量无标注的文本和图像数据：很多工作如 Li L, Chen Y C, Cheng Y, et al. Hero: Hierarchical encoder for video+ language omni-representation pre-training 会采用带字幕的视频来构建起文本和图像之间的联系。如何利用其他容易获取的具有相互联系的文本和图像提升预训练数据量是一个重要的问题。

如何设计更加高效的模型结构综合视频和文本信息是一个重要挑战。

9.4.融合了知识库的预训练模型（如 ERNIE、KnowBERT）存在哪些潜在的缺点？

ERNIE、KnowBERT 都需要利用命名实体识别的方式链接知识库里面的实体。因此命名实体识别的准确率会影响预训练模型融合知识库的效果。

随着知识库的不断更新，新的知识库难以融合进入之前的预训练模型。

9.5.除了实体、关系等结构化知识，还有哪些知识是目前预训练模型所缺少的或者难以从文本中直接学习获得的？

常识性知识：“我的身高不太适合打篮球。” 推测出说话人身高比较矮。理解这句话需要打篮球往往需要较高的身高这种常识性信息。

语用背景知识：情侣之间的“你真讨厌”和仇人之间的“你真讨厌”是两种不同的含义。文字有时并不会记录说话人、说话环境的信息，模型难以识别这种差别。

9.6.在多任务或多语言学习的过程中，考虑到不同任务（语言）的数据量、难度往往不一致，在训练时应当注意哪些问题，以及采用哪些策略？结合 MT-DNN，ERNIE 2.0，以及第 8 章介绍的 T5 等模型进行分析。

MT-DNN 首先对模型进行 BERT 的两个预训练。随后在四类下游任务中再次进行预训练，训练的过程中将所有四种下游任务的数据混合在一起组成 mini-batch，在更新参数时同时计算 loss 进行更新。

针对不同难度的任务，ERNIE 2.0 采用了持续学习的策略，对任务进行排序，按照难度每轮新增加一个训练任务。符合人类学习从简入难的习惯。

T5 对各种任务进行了形式上的统一，统一定义为 seq2seq 任务，通过给定不同的 Prompt 来指导不同的生成答案。