10.6 求近义词和类比词

在10.3节（word2vec的实现）中，我们在小规模数据集上训练了一个word2vec词嵌入模型，并通过词向量的余弦相似度搜索近义词。实际中，在大规模语料上预训练的词向量常常可以应用到下游自然语言处理任务中。本节将演示如何用这些预训练的词向量来求近义词和类比词。我们还将在后面两节中继续应用预训练的词向量。

10.6.1 使用预训练的词向量

基于PyTorch的关于自然语言处理的常用包有官方的**torchtext**以及第三方的**pytorch-nlp**等等。你可以使用 `pip` 很方便地按照它们，例如命令行执行

```
pip install torchtext
```

详情请参见其README。

本节我们使用torchtext进行练习。下面查看它目前提供的预训练词嵌入的名称。

```
import torch
import torchtext.vocab as vocab

vocab.pretrained_aliases.keys()
```

输出：

```
dict_keys(['charngram.100d', 'fasttext.en.300d', 'fasttext.simple.300d', 'glove.42B.3
```

下面查看查看该 `glove` 词嵌入提供了哪些预训练的模型。每个模型的词向量维度可能不同，或是在不同数据集上预训练得到的。

```
[key for key in vocab.pretrained_aliases.keys()
        if "glove" in key]
```

输出：

```
['glove.42B.300d',
 'glove.840B.300d',
 'glove.twitter.27B.25d',
 'glove.twitter.27B.50d',
 'glove.twitter.27B.100d',
 'glove.twitter.27B.200d',
 'glove.6B.50d',
 'glove.6B.100d',
 'glove.6B.200d',
 'glove.6B.300d']
```

预训练的GloVe模型的命名规范大致是“模型.（数据集.）数据集词数.词向量维度”。更多信息可以参考GloVe和fastText的项目网站[1,2]。下面我们使用基于维基百科子集预训练的50维GloVe词向量。第一次创建预训练词向量实例时会自动下载相应的词向量到 `cache` 指定文件夹（默认为 `.vector_cache` ），因此需要联网。

```
cache_dir = "/Users/tangshusen/Datasets/glove"
# glove = vocab.pretrained_aliases["glove.6B.50d"](cache=cache_dir)
glove = vocab.GloVe(name='6B', dim=50, cache=cache_dir) # 与上面等价
```

返回的实例主要有以下三个属性：

- `stoi` ：词到索引的字典；
- `itos` ：一个列表，索引到词的映射；
- `vectors` ：词向量。

打印词典大小。其中含有40万个词。

```
print("一共包含%d个词。" % len(glove.stoi))
```

输出：

```
一共包含400000个词。
```

我们可以通过词来获取它在词典中的索引，也可以通过索引获取词。

```
glove.stoi['beautiful'], glove.itos[3366] # (3366, 'beautiful')
```

10.6.2 应用预训练词向量

下面我们以GloVe模型为例，展示预训练词向量的应用。

10.6.2.1 求近义词

这里重新实现10.3节（word2vec的实现）中介绍过的使用余弦相似度来搜索近义词的算法。为了在求类比词时重用其中的求k近邻（k-nearest neighbors）的逻辑，我们将这部分逻辑单独封装在 `knn` 函数中。

```
def knn(W, x, k):
    # 添加的1e-9是为了数值稳定性
    cos = torch.matmul(W, x.view((-1,))) / (
        (torch.sum(W * W, dim=1) + 1e-9).sqrt() * torch.sum(x * x).sqrt())
    _, topk = torch.topk(cos, k=k)
    topk = topk.cpu().numpy()
    return topk, [cos[i].item() for i in topk]
```

然后，我们通过预训练词向量实例 `embed` 来搜索近义词。

```
def get_similar_tokens(query_token, k, embed):
    topk, cos = knn(embed.vectors,
                    embed.vectors[embed.stoi[query_token]], k+1)
    for i, c in zip(topk[1:], cos[1:]):  # 除去输入词
        print('cosine sim=%.3f: %s' % (c, (embed.itos[i])))
```

已创建的预训练词向量实例 `glove_6b50d` 的词典中含40万个词和1个特殊的未知词。除去输入词和未知词，我们从中搜索与“chip”语义最相近的3个词。

```
get_similar_tokens('chip', 3, glove)
```

输出：

```
cosine sim=0.856: chips
cosine sim=0.749: intel
cosine sim=0.749: electronics
```

接下来查找“baby”和“beautiful”的近义词。

```
get_similar_tokens('baby', 3, glove)
```

输出：

```
cosine sim=0.839: babies
cosine sim=0.800: boy
cosine sim=0.792: girl
```

```
get_similar_tokens('beautiful', 3, glove)
```

输出：

```
cosine sim=0.921: lovely
cosine sim=0.893: gorgeous
cosine sim=0.830: wonderful
```

10.6.2.2 求类比词

除了求近义词以外，我们还可以使用预训练词向量求词与词之间的类比关系。例如，“man”（男人）：“woman”（女人）:: “son”（儿子）：“daughter”（女儿）是一个类比例子：“man”之于“woman”相当于“son”之于“daughter”。求类比词问题可以定义为：对于类比关系中的4个词 $a : b :: c : d$，给定前3个词a、b和c，求d。设词w的词向量为$\text{vec}(w)$。求类比词的思路是，搜索与$\text{vec}(c) + \text{vec}(b) - \text{vec}(a)$的结果向量最相似的词向量。

```
def get_analogy(token_a, token_b, token_c, embed):
    vecs = [embed.vectors[embed.stoi[t]]
            for t in [token_a, token_b, token_c]]
    x = vecs[1] - vecs[0] + vecs[2]
    topk, cos = knn(embed.vectors, x, 1)
    return embed.itos[topk[0]]
```

验证一下“男-女”类比。

```
get_analogy('man', 'woman', 'son', glove) # 'daughter'
```

“首都-国家”类比：“beijing”（北京）之于“china”（中国）相当于“tokyo”（东京）之于什么？答案应该是“japan”（日本）。

```
get_analogy('beijing', 'china', 'tokyo', glove) # 'japan'
```

“形容词-形容词最高级”类比：“bad”（坏的）之于“worst”（最坏的）相当于“big”（大的）之于什么？答案应该是“biggest”（最大的）。

```
get_analogy('bad', 'worst', 'big', glove) # 'biggest'
```

“动词一般时-动词过去时”类比：“do”（做）之于“did”（做过）相当于“go”（去）之于什么？答案应该是“went”（去过）。

```
get_analogy('do', 'did', 'go', glove) # 'went'
```

小结

- 在大规模语料上预训练的词向量常常可以应用于下游自然语言处理任务中。
- 可以应用预训练的词向量求近义词和类比词。