词嵌入---处理文本数据（IMDB）

one-hot 编码或 one-hot 散列得到的词表示是稀疏的、高维的、硬编码的;而词嵌入是密集的、相对低维的，而且是从数据中学习得到的，如图 1。

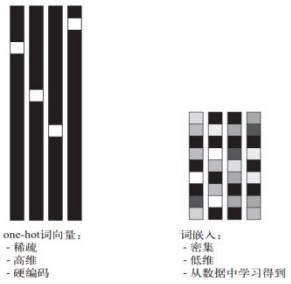


图 1

词嵌入的简单介绍：

词向量之间的几何关系应该表示这些词之间的语义关系。词嵌入的作用应该是将人类的语言映射到几何空间中。例如，在一个合理的嵌入空间中，同义词应该被嵌入到相似的词向量中，一般来说，任意两个词向量之间的几何距离（比如 L2 距离）应该和这两个词的语义距离有关（表示不同事物的词被嵌入到相隔很远的点，而相关的词则更加靠近）。除了距离，你可能还希望嵌入空间中的特定方向也是有意义的。为了更清楚地说明这一点，我们来看一个具体示例。

在图 2 中，四个词被嵌入在二维平面上，这四个词分别是 cat（猫）、 dog（狗）、 wolf（狼）和 tiger（虎）。对于我们这里选择的向量表示，这些词之间的某些语义关系可以被编码为几何变换。例如，从 cat 到 tiger 的向量与从 dog 到 wolf 的向量相等，这个向量可以被解释为“从宠物到野生动物”向量。同样，从 dog 到 cat 的向量与从 wolf 到 tiger 的向量也相等，它可以被解释为“从犬科到猫科”向量。

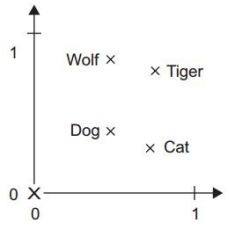


图 2

获取词嵌入的两种方法：

① 在完成任务的同时学习词嵌入：在这种情况下，一开始是随机的词向量，然后对这些词向量通过 **Embedding 层进行学习**得到词嵌入，其学习方式与学习神经网络的权重相同。

② 在不同于待解决问题的学习任务上预计算好词嵌入，然后将其加载到模型中。这种词嵌入叫作**预训练词嵌入**（pretrained word embedding）。

下面分别介绍上面两种情况。

1、 利用 Embedding 层学习词嵌入

目前还没有一个理想的词嵌入空间，可以完美地映射人类语言，并可用于所有自然语言处理任，更实际的角度来说，一个好的词嵌入空间在很大程度上取决于你的任务。英语电影评论情感分析模型的完美词嵌入空间，可能不同于英语法律文档分类模型的完美词嵌入空间，因为某些语义关系的重要性因任务而异。、

因此，合理的做法是对每个新任务都学习一个新的嵌入空间。幸运的是，反向传播

让这种学习变得很简单，而 Keras 使其变得更简单。我们要做的就是学习一个层的权重，这个层就是 Embedding 层。

```
from keras.layers import Embedding
embedding_layer = Embedding(1000, 64)
```

Embedding 层至少需要两个参数：标记的个数（这里是 1000，即最大单词索引 +1）和嵌入的维度（这里是 64）

2、使用预训练的词嵌入

有时可用的训练数据很少，以至于只用手头数据无法学习适合特定任务的词嵌入。那么应该怎么办？

在自然语言处理中使用预训练的词嵌入，其背后的原理与在图像分类中使用预训练的卷积神经网络是一样的：没有足够的数据来自己学习真正强大的特征，但你需要的特征应该是非常通用的，比如常见的视觉特征或语义特征。

有许多预计算的词嵌入数据库，你都可以下载并在 Keras 的 Embedding 层中使用。word2vec 就是其中之一。另一个常用的是 GloVe（global vectors for word representation，词表示全局向量），由斯坦福大学的研究人员于 2014 年开发。

接下的的与训练的词嵌入使用的是 GloVe，其他的词嵌入用法一致

3、实战环节：原始的 IMDB 数据到词嵌入

3.1 下载并处理 IMDB 原始数据

下载：打开 http://mng.bz/0tIo，下载原始 IMDB 数据集并解压

```
def dataprocess():
    """
    处理 IMDB 原始数据的标签,将训练评论转换成字符串列表，每个字符串对应一条评论,评论标签（正面 / 负面）
转换成 labels 列表
    :return: texts：训练集列表 labels：标签列表
    """
    labels = []
    texts = []
    imdb_dir = 'F:\\IMDB 数据集\\aclImdb'
    train_dir = os.path.join(imdb_dir, 'train')
    for label_type in ['neg', 'pos']:
        dir_name = os.path.join(train_dir, label_type)
        for fname in os.listdir(dir_name):
            if fname[-4:] == '.txt':
                f = open(os.path.join(dir_name, fname), 'r', encoding='utf-8')
                texts.append(f.read())
                f.close()
                if label_type == 'neg':
                    labels.append(0)
                else:
```

```
                labels.append(1)
    return texts, labels
```

3.2 对数据进行分词

预训练的词嵌入对训练数据很少的问题特别有用，所以我们又添加了以下限制：将训练数据限定为前 200 个样本。因此，你需要在读取 200 个样本之后学习对电影评论进行分类，（当然你也可以改变训练集的大小，这里取 200 只是为了更好的说明预训练词嵌入对训练数据很小的情况有用。）

```
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
maxlen = 100  # 在 100 个单词后截断评论
training_samples = 200  # 在 200 个样本上训练
validation_samples = 10000  # 在 10000 个样本上验证
max_words = 10000  # 只考虑数据集中前 10000 个最常用见的单词

tokenizer = Tokenizer(num_words=max_words)#创建一个分词器，设置只考虑前 10000 个最常见的单词
tokenizer.fit_on_texts(texts) #构建单词索引

sequences = tokenizer.texts_to_sequences(texts) #将字符串转换成整数索引的组成的列表

word_index = tokenizer.word_index
# Found 88582 unique tokens.
print("Found %s unique tokens." % len(word_index))
data = pad_sequences(sequences, maxlen=maxlen)
labels = np.asarray(labels)
# Shape of data tensor: (25000, 100)
print('Shape of data tensor:', data.shape)
# Shape of label tensor: (25000,)
print('Shape of label tensor:', labels.shape)
"""将数据划分为训练集和验证集，但首先要打乱数据，因为一开始数据中的样本是拍好序的（所有负面评价都
在前面，然后是所有正面的评论）"""
indices = np.arange(data.shape[0])
np.random.shuffle(indices)
data = data[indices]
labels = labels[indices]
x_train = data[: training_samples]
y_train = labels[: training_samples]
x_val = data[training_samples:training_samples + validation_samples]
y_val = labels[training_samples:training_samples + validation_samples]
```

3.3 下载 GloVe 词嵌入

打开 https://nlp.stanford.edu/projects/glove，下载 2014 年英文维基百科的预计算嵌入。这是一个 822 MB 的压缩文件，文件名是 glove.6B.zip，里面包含 400 000

个单词（或非单词的标记）的 100 维嵌入向量。解压文件。

3.4 对嵌入进行预处理

我们对解压后的文件（一个 .txt 文件）进行解析，构建一个将单词（字符串）映射为其向量表示（数值向量）的索引。

```
"""解析 GloVe 词嵌入文件"""
glove_dir = 'F:\\IMDB 数据集\\glove.6B'
embedding_index = {}
f = open(os.path.join(glove_dir,'glove.6B.100d.txt'),'r',encoding='utf-8')
for line in f:
    values = line.split()
    word = values[0]
    coefs = np.asarray(values[1:],dtype='float32')
    embedding_index[word]=coefs
f.close()
print('Found %s word vectors.'%len(embedding_index))
```

接下来，需要构建一个可以加载到 Embedding 层中的嵌入矩阵。它必须是一个形状为(max_words, embedding_dim) 的矩阵。

```
"""准备 GloVe 词嵌入矩阵"""
embedding_dim = 100
embedding_matrix = np.zeros(shape=(max_words, embedding_dim))
for word,i in word_index.items():
    if i < max_words:
        embedding_vector = embedding_index.get(word)
        if embedding_vector is not None: # 嵌入索引（embedding_index）中找不到词，其嵌入向
量全为 0
            embedding_matrix[i] = embedding_vector
```

3.5 定义模型

这里定义一个比较的简单的模型，不涉及 RNN 和 CNN，因为这里只是为了介绍词嵌入 Embedding 层

```
"""定义模型"""
model = Sequential()
model.add(Embedding(max_words,embedding_dim,input_length=maxlen))
model.add(Flatten())
model.add(Dense(32, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.summary()
```

3.6 在模型中加载 Glove 嵌入

Embedding 层只有一个权重矩阵，是一个二维的浮点数矩阵，其中每个元素 i 是与索　　引 i 相关联的词向量。很简单，将准备好的 GloVe 矩阵加载到 Embedding 层中，即模型的第一层。（如果不加载与训练词嵌入，这里可以注释掉）

```
"""在模型中加载 GloVe 嵌入(或不适用预训练)"""
model.layers[0].set_weights([embedding_matrix])# 与训练第一层（Embedding 层）
model.layers[0].trainable = False # 冻结预训练层
```

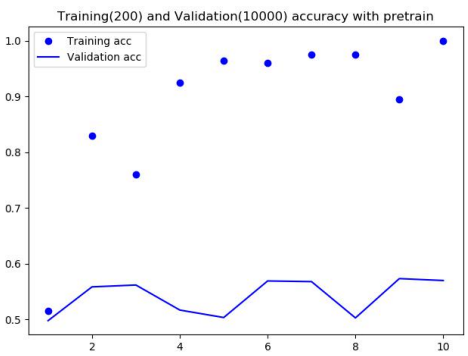
3.7 训练模型和评估模型

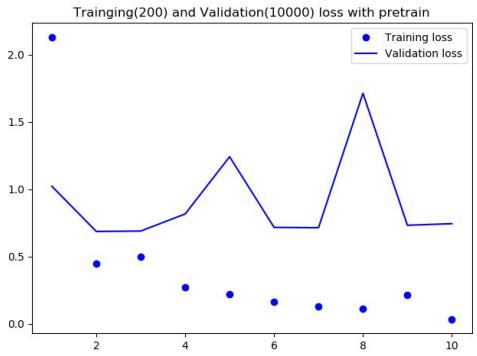
```
"""编译并训练模型"""
model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['acc'])
history = model.fit(x_train, y_train, epochs=10, batch_size=32, validation_data=(x_val,
y_val))
model.save('pre_trained_glove_model.h5')

"""绘制结果"""
acc = history.history['acc']
val_acc = history.history['val_acc']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(acc)+1)
plt.figure()
plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training(200) and Validation(10000) accuracy with pretrain')
plt.legend()

plt.figure()
plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Trainging(200) and Validation(10000) loss with pretrain')
plt.legend()

plt.show()
```





模型很快就开始过拟合，考虑到训练样本很少（200 个训练样本），这一点也不奇怪。出于同样的原因，验证精度的波动很大，但似乎达到了接近 60%。

训练集上进行测试

```
import os
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
import numpy as np
from keras.models import load_model
from keras.models import Sequential
def dataprocess():
    """
    处理 IMDB 原始数据的标签
    :return: 测试集集及其标签
    """
    labels = []
    texts = []
    imdb_dir = 'F:\\IMDB 数据集\\aclImdb'
    train_dir = os.path.join(imdb_dir, 'test')

    for label_type in ['neg', 'pos']:
        dir_name = os.path.join(train_dir, label_type)
        for fname in os.listdir(dir_name):
            if fname[-4:] == '.txt':
                f = open(os.path.join(dir_name, fname), 'r', encoding='utf-8')
                texts.append(f.read())
                f.close()
                if label_type == 'neg':
                    labels.append(0)
                else:
                    labels.append(1)
    return texts, labels
texts, labels = dataprocess()
tokenizer = Tokenizer(num_words=10000)
tokenizer.fit_on_texts(texts)
sequences = tokenizer.texts_to_sequences(texts)
```

```
x_test = pad_sequences(sequences, maxlen=100)
y_test = np.asarray(labels)

model = load_model('pre_trained_glove_model.h5')
loss, acc = model.evaluate(x_test, y_test)
print(loss, acc)
```

能到达 50%以上的正确率，还是不错的，因为我们只用了 200 个样本进行训练。