# 神经网络

## CNN

### 基本概念

卷积神经网络并不是一个新的概念，甚至在20世纪90年代就已经被广泛使用，但深度学习卷土重来的第一功臣非卷积神经网络莫属，原因之一就是卷积神经网络非常适合计算机视觉应用的模型。

卷积在工程和数学上都有很多应用——在统计学中，加权的滑动平均是一种卷积；在概率论中，两个统计独立的变量x和y求和的概率密度函数是x和y的概率密度函数的卷积；在信号系统中，任意一个线性系统的输出都可以通过将输入信号与系统函数（系统的冲击响应）做卷积获得。

在卷积网络的术语中，卷积的第一个参数通常表示输入（Input），第二个参数表示核（Kernel），输出被称作特征映射（Feature Map）。离散时间系统的响应y(n)可以由激励x(n)和单位样值响应h(n)的卷积得到，公式为：

许多机器学习库将互相关的实现称为卷积，公式为：

### 文本分类示例

#### 数据准备

使用Stanford的Glove进行文本向量化表示，词向量维度为100维，单词数量为40万。数据集使用news20groups，该数据共有20个类别，数据信息如下：

训练集：11314 篇文本。

|  |  |  |
| --- | --- | --- |
| 序号 | 类别 | 数量 |
| 1 | alt.atheism | 480 |
| 2 | comp.graphics | 584 |
| 3 | comp.os.ms-windows.misc | 591 |
| 4 | comp.sys.ibm.pc.hardware | 590 |
| 5 | comp.sys.mac.hardware | 578 |
| 6 | comp.windows.x | 593 |
| 7 | misc.forsale | 585 |
| 8 | rec.autos | 594 |
| 9 | rec.motorcycles | 598 |
| 10 | rec.sport.baseball | 597 |
| 11 | rec.sport.hockey | 600 |
| 12 | sci.crypt | 595 |
| 13 | sci.electronics | 591 |
| 14 | sci.med | 594 |
| 15 | sci.space | 593 |
| 16 | soc.religion.christian | 599 |
| 17 | talk.politics.guns | 546 |
| 18 | talk.politics.mideast | 564 |
| 19 | talk.politics.misc | 465 |
| 20 | talk.religion.misc | 377 |

测试集：7532篇文本。

|  |  |  |
| --- | --- | --- |
| 序号 | 类别 | 数量 |
| 1 | alt.atheism | 319 |
| 2 | comp.graphics | 389 |
| 3 | comp.os.ms-windows.misc | 394 |
| 4 | comp.sys.ibm.pc.hardware | 392 |
| 5 | comp.sys.mac.hardware | 385 |
| 6 | comp.windows.x | 395 |
| 7 | misc.forsale | 390 |
| 8 | rec.autos | 396 |
| 9 | rec.motorcycles | 398 |
| 10 | rec.sport.baseball | 397 |
| 11 | rec.sport.hockey | 399 |
| 12 | sci.crypt | 396 |
| 13 | sci.electronics | 393 |
| 14 | sci.med | 396 |
| 15 | sci.space | 394 |
| 16 | soc.religion.christian | 398 |
| 17 | talk.politics.guns | 364 |
| 18 | talk.politics.mideast | 376 |
| 19 | talk.politics.misc | 310 |
| 20 | talk.religion.misc | 251 |

统计代码的github地址为

#### 实验步骤

1. 将所有的新闻样本转化为词索引序列，即为每一个词依次分配一个整数ID。遍历所有的新闻文本，我们只保留最常见的20000个词，而且，每个新闻文本最多只保留1000个单词。

2. 生成一个词向量矩阵。第i列表示词索引为i的单词的词向量。

3. 将词向量矩阵载入Keras **Embedding层**，设置该层的权重不可再训练（即在之后的网络训练过程中，使用GloVe预训练词向量，词向量不再改变）。

4. Keras Embedding层之后使用**1D的卷积层**进行预测模型的构建，并用一个**softmax全连接层**输出新闻类别判断。

#### 实验代码

##### 数据准备

###### （1）导入GloVe词向量词典

我们使用了GloVe向量作为词嵌入层的初始集合。因此，这里导入GloVe词向量词典。

|  |
| --- |
| import numpy as np  # GloVe向量词典  embeddings\_index = dict()  # 读入向量  with open('../data/glove.6B/glove.6B.100d.txt', encoding='utf-8', errors='ignore') as infile:  for line in infile:  line = line.strip()  if line:  items = line.split()  word = items[0]  coefs = np.array(items[1:], dtype='float32')  embeddings\_index[word] = coefs  print('readed %d word vectors' % len(embeddings\_index)) |

###### （2）导入训练和测试语料

这里我们使用sklearn的load\_files方法来加载分类数据。

|  |
| --- |
| from sklearn.datasets import load\_files  train\_data = load\_files(container\_path='../data/20news-bydate/20news-bydate-train', # 分类语料根目录  categories=None, # 需要加载的类目名称列表  encoding="gbk", decode\_error="ignore") # 文件编码  print('readed %d train texts' % len(train\_data.data))  test\_data = load\_files(container\_path='../data/20news-bydate/20news-bydate-test', # 分类语料根目录  categories=None, # 需要加载的类目名称列表  encoding="gbk", decode\_error="ignore") # 文件编码  print('readed %d test texts' % len(test\_data.data)) |

###### （3）将文本数据转换称神经网络所需的2维张量

由于是多分类问题，这里我们对类别做了one-hot编码。

|  |
| --- |
| from keras.preprocessing.text import Tokenizer  from keras.preprocessing.sequence import pad\_sequences  from keras.utils import to\_categorical  # 构建“分词器”,设置在从语料获得词典时收录的最大词语数量  MAX\_NB\_WORDS = 20000  tokenizer = Tokenizer(num\_words=MAX\_NB\_WORDS)  # 构建词典  tokenizer.fit\_on\_texts(train\_data.data)  # 获得词典中单词索引  word\_index = tokenizer.word\_index  # 打印词典收录词语数量  print('dictionary len = %d' % len(word\_index))  # 将训练文本集中的每篇文本转换为词索引序列  train\_sequences = tokenizer.texts\_to\_sequences(train\_data.data)  # 将测试文本集中的每篇文本转换为词索引序列  test\_sequences = tokenizer.texts\_to\_sequences(test\_data.data)  # 使每篇文本的词索引序列长度一致  MAX\_SEQUENCE\_LENGTH=1000  x\_train = pad\_sequences(train\_sequences, maxlen=MAX\_SEQUENCE\_LENGTH)  x\_test = pad\_sequences(test\_sequences, maxlen=MAX\_SEQUENCE\_LENGTH)  # 获得每篇文本对应的类别  y\_train = to\_categorical(train\_data.target)  y\_test = to\_categorical(test\_data.target)  # 求类别索引对应类别名称  labels\_index = dict(enumerate(train\_data.target\_names))  # 输出数据形状  print("x\_train shape = ", x\_train.shape)  print("y\_train shape = ", y\_train.shape)  print("x\_val shape=", x\_test.shape)  print("y\_val shape=", y\_test.shape) |

###### （4）用GloVe词向量词典将构建的词典转换称2维张量矩阵

|  |
| --- |
| # 当前词典中的词语数量,之所以这样恐怕是因为Tokenizer类旧版中参数num\_words不起作用  num\_words = min(MAX\_NB\_WORDS, len(word\_index))  # 初始化嵌入层矩阵  EMBEDDING\_DIM = 100 # 嵌入层输出尺寸  embedding\_matrix = np.zeros((num\_words, EMBEDDING\_DIM))  # 遍历由训练集构建的词典中的每个词语  for word, i in word\_index.items():  # 如果词语的索引也就是位置超过了设置的词典收录词语数，则丢弃  if i >= num\_words:  continue  # 读取词语向量,如果词语未被GloVe收录，则其向量为0向量  if word in embeddings\_index:  embedding\_matrix[i] = embeddings\_index[word] |

##### 设计网络结构

###### （1）用词典向量矩阵初始化Embedding层

|  |
| --- |
| from keras.models import Sequential  from keras.layers.embeddings import Embedding  model = Sequential()  model.add(Embedding(input\_dim=num\_words, # 输入词向量矩阵的最大词语数量，也就是词典最大索引数+1，词典索引从0开始  output\_dim=EMBEDDING\_DIM, # 将输入向量矩阵转换为3维矩阵后，第3维的元素个数  weights=[embedding\_matrix], # 用词典向量矩阵初始化嵌入层参数  input\_length=MAX\_SEQUENCE\_LENGTH, # 嵌入层接受的输入文本集的每一行文本的单词序列的长度  trainable=False)) # 嵌入层不再可训练 |

###### （2）卷积抽取特征

该段结构可以重复多次。

|  |
| --- |
| # 卷积层  from keras.layers import Conv1D  model.add(Conv1D(filters=256, # 卷积滤子输出的维度，要求整数  kernel\_size=5, # 卷积核的空域或时域窗长度。要求是整数或整数的列表，或者是元组。如果是单一整数，则应用于所有适用的维度  activation='relu',# 激活函数  ))  # 池化层  from keras.layers import MaxPooling1D  model.add(MaxPooling1D(pool\_size=5))  # 丢弃层  from keras.layers import Dropout  model.add(Dropout(0.4)) |

###### （3）平坦层

|  |
| --- |
| # 平坦层  from keras.layers import Flatten  model.add(Flatten()) |

###### （4）全连接层

预测输出

|  |
| --- |
| from keras.layers import Dense  model.add(Dense(128, activation='relu'))  model.add(Dense(len(labels\_index), activation='softmax')) |

##### 设置模型

|  |
| --- |
| model.compile(loss='binary\_crossentropy',  optimizer='Adagrad',  metrics=['accuracy'])  model.summary() |

##### 训练

|  |
| --- |
| train\_history = model.fit(x\_train, y\_train,  batch\_size=25, # 1个训练周期中，mini-batch gradient descent中每一批次100项数据  epochs=10, # 整个数据集将被周期地训练10次，取最优的一次  verbose=2, # 显示训练过程  validation\_split=0.2) # 20%验证数据 |

##### 评测

|  |
| --- |
| model.evaluate(x\_test, y\_test, verbose=1) |

##### 保存模型

|  |
| --- |
| model.save\_weights('./CNN20\_model') |

## RNN

## LSTM

## Bi-LSTM

## 激活函数

## 初始化方法

### Uniform

## 损失函数

### 交叉熵

### KL散度

### 对称KL散度

## 正则化

### L1正则化

### L2正则化

### L2/L1正则原理

### Dropout

## 优化算法

### SGD

### Momentum

### Adagrad

# Keras库

**张量（tensor）**的概念。0阶张量即标量，1阶张量即向量，2阶张量即矩阵，3阶张量即立方体，4阶张量即多个立方体。维度概念的说明

|  |
| --- |
| In [6]: import numpy as np  In [7]: a = np.array([[1,2],  ...: [3,4]])  In [8]: np.sum(a, axis=0)  Out[8]: array([4, 6])  In [9]: np.sum(a, axis=1)  Out[9]: array([3, 7]) |

**数据格式（Data Format）**的概念，它是指对训练或测试数据的表达方法。例如100张RGC三通道彩色图片高度为16，宽读为32，在Keras中表示为(100, 3, 16, 32)，但是在TensorFlow中表示为(100, 16, 32, 3)，Keras默认的数据组织方式在~/.keras/keras.json中定义，可以查看此文件的image\_data\_format项，也可以通过K.image\_data\_format()函数返回得到数据组织方式。

**函数式模型**（Functional Model），在Keras0.x中有两种模型分别为Sequential序列化模型也就是层与层之间只有相邻关系，没有跨层连接。Graph图模型，层与层之间想怎么连都可以。Keras1和Keras2中，不再支持Graph模型，但是增加了“functional model API”，在使用时利用的是“函数式编程”的风格，因此称之为函数式模型。

**批数据（batch）**Keras的模块中经常会出现batch\_size，指的是Mini-batch的大小。需要注意的是Keras中用的优化器SGD是随机梯度下将方法，不代表是一个样本就更新一回模型参数，SGD仍然是基于Mini-batch的。Mini-batch Gradient Descent小批量梯度下降算法，这种方法把数据分为若干个批次，按批来更新参数，这样，一个批中的一组数据共同决定了本次梯度的方向，下降起来就不容易跑偏，减少了随机性。另一方面，因为批的样本数与整个数据集相比小了很多，计算量也减少了很多。

**训练周期数(epochs)**指的是训练过程中整个数据集将被循环训练多少次。

Keras的核心数据结构是“**模型**”，模型是一种组织网络层的方式。Keras中主要的模型是Sequential模型。Sequential是一系列网络层按顺序构成的栈。一般首先通过实例化一个Sequential栈。然后通过栈的add(网络层)方法添加网络层。常见的核心网络层有全连接层Dense，丢弃层Dropout，激活层activation（注意一般通过网络层类的激活函数参数来设置，而不是单独再添加一个激活层）。完成模型的搭建后，需要使用栈的compile(…)方法来编译模型，该方法的常用参数有损失函数设置loss，优化器设置optimizer等。完成编译后，调用栈的fit方法来训练模型，该方法可设置的参数有在整个训练集上执行训练的周期数epochs，每个周期都会得到一个模型，最终选择一个最优的模型。小批量随机梯度下降训练过程中每个批次抽取的数据量batch\_size。最后，通过栈的evaluate方法来评估模型在测试集上的准确率。

Sequential对象的方法

|  |  |  |
| --- | --- | --- |
| 方法名 | 用途 | 示例 |
| **model.summary()** | 打印模型概况 |  |
| **model.get\_config()** | 返回模型配置信息的字典。模型可以从配置信息重构回去，其中config是一个Json格式的字典对象 | config = model.get\_config()  model = Sequential.from\_config(config) |
| model.get\_layer() | 依据层名或下标获得层对象 |  |
| **model.get\_weights()** | 返回模型权重张量的列表，类型为Numpy array |  |
| model.set\_weights() | 表示从Numpy array里将权重载入给定模型，要求数组具有与model.get\_weights()相同的形状 |  |
| model.to\_json() | 表示返回代表模型的Json字符串，仅包含网络结构，不包含权值。用户可以从Json字符串中重构原模型 | json\_str = model.to\_json()  model = model\_from\_json(json\_string) |
| model.save\_weights(filepath) | 表示将模型权重保存到指定路径，文件类型是HDF5 |  |
| model.load\_weights(filepath, by\_name=False) | 表示从HDF5文件中加载权重到当前模型中，默认情况下模型的结构保持不变。如果想将权重载入不同的模型（有些层相同）中，则设置by\_name=True，只有名字匹配的层才会载入权重 |  |

compile方法的参数

|  |  |  |
| --- | --- | --- |
| 参数名 | 用途 | 示例 |
| optimizer | 优化器名或优化器对象，设置网络训练时的优化方法 | 可用的优化器详见官方文档 |
| loss | 损失函数名或损失函数对象，设置网络训练时的损失函数 | 可用的损失函数详见官方文档 |
| metrics | 网络训练时评估网络性能的指标。 | metrics=[‘accuracy’] |