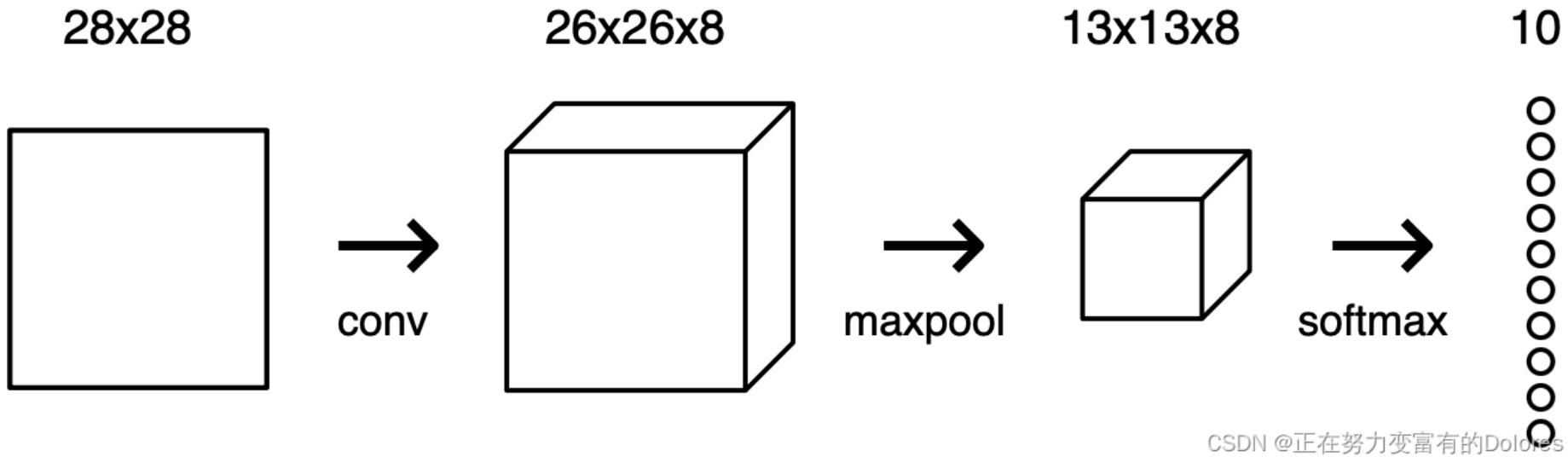
基于卷积神经网络的手写体识别

2023年10月

手写体数字0-9

MNIST数据集中的每个图像为28x28，并包含一个居中的灰度数字。我们的CNN将获取一幅图像并输出10个可能的类中的一个(每个数字对应一个类)。

网络架构



conv: 使用一个带有8个过滤器的卷积层作为CNN网络中的初始层。这意味着它将把28x28的输入图像变成26x26x8的输出体积，输出是26x26x8而不是28x28x8，因为我们使用了有效的填充，这会使输入的宽度和高度减少2。
maxpool: 图像中的相邻像素往往有相似的值，所以conv层通常也会在输出中为相邻像素产生相似的值。因此，在转换层的输出中包含的大部分信息是冗余的。所以pooling的作用就是过滤掉相似并且冗余的信息，增强训练效果。这里使用的是一个最大池化层的例子，池化大小为2。池化层将把一个26x26x8的输入转换成一个13x13x8的输出。
softmax: 为了完成我们的CNN，我们需要赋予它进行实际预测的能力。我们将通过使用用于多类分类问题的标准最后一层来实现Softmax层，这是一个使用Softmax函数作为其激活函数的全连接层。我们将使用具有10个节点的softmax层，每个节点代表每个数字，作为CNN中的最后一层。层中的每个节点都将连接到每个输入。在进行softmax变换后，概率最高的节点所代表的数字就是CNN的输出

手写体数字0-9

Keras: 是一个用Python编写的深度学习API，运行在机器学习平台TensorFlow之上。它的开发重点是实现快速实验，类似于把tensorflow各个模块都集成为一个软件工具包，直接调用就行，简单方便。

1. 训练前
安装工具库：tensorflow, numpy, mnist
下载mnist数据集：数据集
地址http://yann.lecun.com/exdb/mnist/

2. 重构数据

在开始之前，我们将图像像素值从[0,255]归一化到[-0.5,0.5]，以使我们的网络更容易训练(使用较小的、居中的值通常会得到更好的结果)。我们还将把每个图像从(28,28)重塑为(28,28,1)，因为Keras需要三维数据。

```
mndata = MNIST(r'xxxxxxxx', return_type='numpy', gz=True)
#xxxxxxxx is your mnist data file directory
#if your data already be decompressed, gz=Flase, if not, gz=True
train_images, train_labels = mndata.load_training()
test_images, test_labels = mndata.load_testing()

# Normalize the images.
# we'll normalize the image pixel values from [0, 255] to [-0.5, 0.5] to make our network easier to train
# (using smaller, centered values usually leads to better results).
train_images = (train_images / 255) - 0.5
test_images = (test_images / 255) - 0.5

# Reshape the images.
# We need reshape each image from (28, 28) to (28, 28, 1) because Keras requires the third dimension
train_images=train_images.reshape(-1, 28, 28, 1)
test_images=test_images.reshape(-1, 28, 28, 1)
```

3. 构建神经网络模型

每个Keras模型都是使用Sequential类(表示层的线性堆栈)或functional model类(更易定制的类)构建的。我们将使用更简单的顺序模型，因为我们的CNN将是一个线性层堆栈。num_filters、filter_size和pool_size是自解释变量，用于设置CNN的超参数。任何Sequential模型中的第一层都必须指定input_shape，所以我们在Conv2D上指定(28,28,1)输入。一旦指定了这个输入形状，Keras将自动推断出后面图层的输入形状。输出Softmax层有10个节点，每个节点对应一个类。

```
num_filters = 8
filter_size = 2
pool_size = 2
# Build the model.
model = Sequential([  Conv2D(num_filters, filter_size, input_shape=(28, 28, 1), activation='relu'),
                      MaxPooling2D(pool_size=pool_size),
                      Flatten(),
                      Dense(10, activation='softmax'),
])
```

4.编译模型

有三个关键的步骤优化器。我们将坚持使用一个相当不错的默认值:基于梯度的Adam优化器。Keras还有许多其他的优化器，可以根据项目需求进行替换。损失函数。由于我们使用Softmax输出层，我们将使用交叉熵损失(Cross-Entropy loss)。Keras区分了binary_crossentropy和categorical_crossentropy，所以我们将使用后者。点击此处查看keras的所有损失函数。指标列表。由于这是一个分类问题，我们将只让Keras报告精度指标。

```
# Compile the model.
model.compile( 'adam', loss='categorical_crossentropy', metrics=['accuracy'],
)
```

5. 训练模型

在Keras中训练模型的只需要调用 fit()函数

- The training data (images and labels): 训练数据(图像和标签)，通常分别称为X和Y。
- epochs: 在整个数据集上的迭代的数量。
- validation_data: 验证数据，不包含在训练数据里面，这是在训练期间用来定期测量网络性能的数据，用它和预测的数据进行比较，得出精确度。

```
# Train the model.
model.fit(
    train_images,
    to_categorical(train_labels),
    epochs=5,
    validation_data=(test_images,
    to_categorical(test_labels)),
)
```

6. 保存并使用模型

样我们可以在下一次使用模型来预测数据时直接调用，而省去重新训练模型的时间。

```
# Save the model to disk.
model.save_weights('cnn.mnist')

# Load the model from disk later using:
model.load_weights('cnn.mnist')
```

7. 输出

如果你使用的是我在github上传的完整代码来调试，当得到类似于以下的输出，则代表调试成功。

```
models predictions

[7 2 1 0 4 1 4 9 5 9]

checker

[7 2 1 0 4 1 4 9 5 9]
```

山东大学
SHANDONG UNIVERSITY

谢 谢