

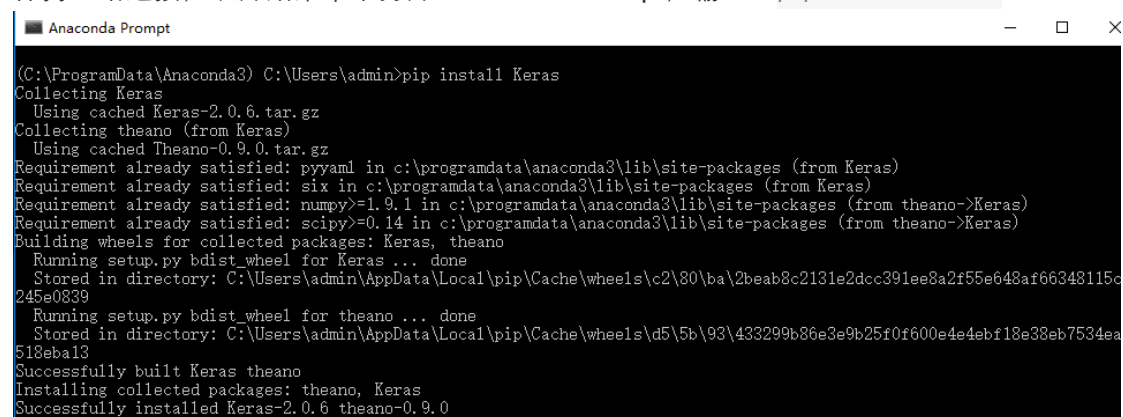
学习笔记 2-CNN 实践-2017-07-20

Keras:基于 Theano 和 TensorFlow 的深度学习库

Keras 安装

前提：安装 Anaconda，的版本是 Anaconda3 (64-bit)

保持网络连接，从开始菜单中打开 Anaconda Prompt，输入：`pip install Keras`



```

Anaconda Prompt
(C:\ProgramData\Anaconda3) C:\Users\admin>pip install Keras
Collecting Keras
  Using cached Keras-2.0.6.tar.gz
Collecting theano (from Keras)
  Using cached Theano-0.9.0.tar.gz
Requirement already satisfied: pyyaml in c:\programdata\anaconda3\lib\site-packages (from Keras)
Requirement already satisfied: six in c:\programdata\anaconda3\lib\site-packages (from Keras)
Requirement already satisfied: numpy>=1.9.1 in c:\programdata\anaconda3\lib\site-packages (from theano->Keras)
Requirement already satisfied: scipy>=0.14 in c:\programdata\anaconda3\lib\site-packages (from theano->Keras)
Building wheels for collected packages: Keras, theano
  Running setup.py bdist_wheel for Keras ... done
  Stored in directory: C:\Users\admin\AppData\Local\pip\Cache\wheels\c2\80\ba\2beab8c2131e2dcc391ee8a2f55e648af66348115d245e0839
  Running setup.py bdist_wheel for theano ... done
  Stored in directory: C:\Users\admin\AppData\Local\pip\Cache\wheels\d5\5b\93\433299b86e3e9b25f0f600e4e4ebf18e38eb7534ea518eba13
Successfully built Keras theano
Installing collected packages: theano, Keras
Successfully installed Keras-2.0.6 theano-0.9.0

```

目前 TensorFlow 在 Windows 下只支持 Python 3.5 版本，所以我之前的 2.7 果断换 3.5
以管理员身份打开 Anaconda Prompt，

输入：`pip install --upgrade --ignore-installed tensorflow`

Theano 是什么？

Theano 是一个 Python 库，可以在 CPU 或 GPU 上运行快速数值计算。这是 Python 深度学习中的一个关键基础库，你可以直接用它来创建深度学习模型或包装库，大大简化了程序。

Python 的核心 Theano 是一个数学表达式的编译器。它知道如何获取你的结构，并使之成为一个使用 numpy、高效本地库的非常高效的代码，如 BLAS 和本地代码 (C++)，在 CPU 或 GPU 上尽可能快地运行。它巧妙的采用一系列代码优化从硬件中攫取尽可能多的性能。

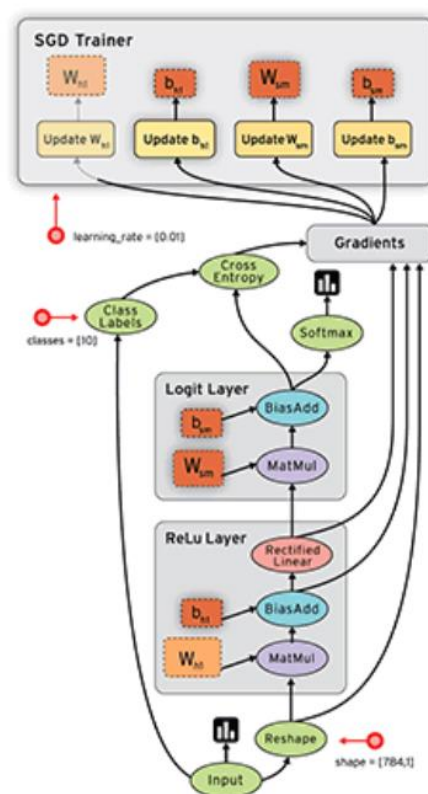
Theano 表达式的实际语法是象征性的，可以推送给初学者用于一般软件开发。具体来说，**表达式是在抽象的意义上定义，编译和后期是用来进行计算**。它是为深度学习中处理大型神经网络算法所需的计算而专门设计的。它是这类库的首创之一（发展始于 2007 年），被认为是深度学习研究和开发的行业标准。

TensorFlow 是什么？

TensorFlow™ 是一个采用数据流图 (data flow graphs)，用于数值计算的开源软件库。节点 (Nodes) 在图中表示数学操作，图中的线 (edges) 则表示在节点间相互联系的多维数据数组，即张量 (tensor)。它灵活的架构让你可以在多种平台上展开计算，例如台式计算机中的一个或多个 CPU (或 GPU)，服务器，移动设备等等。TensorFlow 最初由 Google 大脑小组 (隶属于 Google 机器智能研究机构) 的研究员和工程师们开发出来，用于机器学习和深度神经网络方面的研究，但这个系统的通用性使其也可广泛用于其他计算领域。

什么是数据流图 (Data Flow Graph) ？

数据流图用“结点” (nodes) 和“线”(edges)的有向图来描述数学计算。“节点”一般用来表示施加的数学操作，但也可以表示数据输入 (feed in) 的起点/输出 (push out) 的终点，或者是读取/写入持久变量 (persistent variable) 的终点。“线”表示“节点”之间的输入/输出关系。这些数据“线”可以输运“size 可动态调整”的多维数据数组，即“张量” (tensor)。张量从图中流过的直观图像是这个工具取名为“Tensorflow”的原因。一旦输入端的所有张量准备好，节点将被分配到各种计算设备完成异步并行地执行运算。



Keras 基本概念

符号主义

符号主义的计算首先定义各种变量，然后建立一个“**计算图**”，**计算图**规定了各个变量之间的计算关系。建立好的**计算图**需要编译以确定其内部细节，然而，此时的计算图还是一个“空壳子”，里面没有任何实际的数据，只有当你把需要运算的输入放进去后，才能在整个模型中形成数据流，从而形成输出值。

张量(tensor)

使用这个词汇的目的是为了表述统一，张量可以看作是向量、矩阵的自然推广，我们用张量来表示广泛的数据类型。

规模最小的张量是 0 阶张量，即标量，也就是一个数。

当我们把一些数有序的排列起来，就形成了 1 阶张量，也就是一个向量

如果我们继续把一组向量有序的排列起来，就形成了 2 阶张量，也就是一个矩阵

张量的阶数有时候也称为维度，或者轴，轴这个词翻译自英文 axis。譬如一个矩阵[[1,2],[3,4]]，是一个 2 阶张量，有两个维度或轴，沿着第 0 个轴（为了与 python 的计数方式一致，本文档维度和轴从 0 算起）你看到的是[1,2], [3,4]两个向量，沿着第 1 个轴你看到的是[1,3], [2,4]两个向量。

要理解“沿着某个轴”是什么意思，不妨试着运行一下下面的代码：

```
import numpy as np
a = np.array([[1,2],[3,4]])
sum0 = np.sum(a, axis=0)
sum1 = np.sum(a, axis=1)
print sum0
print sum1
```

data_format

这是一个无可奈何的问题，在如何表示一组彩色图片的问题上，Theano 和 TensorFlow 发生了分歧，'th'模式，也即 **Theano 模式**会把 100 张 RGB 三通道的 16×32（高为 16 宽为 32）彩色图表示为下面这种形式（100,3,16,32），**Caffe 采取的也是这种方式**。第 0 个维度是样本维，代表样本的数目，第 1 个维度是通道维，代表颜色通道数。后面两个就是高和宽了。这种 theano 风格的数据组织方法，称为“channels_first”，即通道维靠前。

而 TensorFlow，的表达形式是 (100,16,32,3)，即把通道维放在了最后，这种数据组织方式称为“channels_last”。

Keras 默认的数据组织形式在 ~/.keras/keras.json 中规定，可查看该文件的 image_data_format 一项查看，也可在代码中通过 K.image_data_format() 函数返回，**请在网络的训练和测试中保持维度顺序一致。**

Batch

深度学习的优化算法，说白了就是梯度下降。每次的参数更新有两种方式。

第一种，遍历全部数据集算一次损失函数，然后算函数对各个参数的梯度，更新梯度。这种方法每更新一次参数都要把数据集里的所有样本都看一遍，计算量开销大，计算速度慢，不支持在线学习，这称为 **Batch gradient descent，批梯度下降**。

另一种，每看一个数据就算一下损失函数，然后求梯度更新参数，这个称为 **随机梯度下降，stochastic gradient descent**。这个方法速度比较快，但是收敛性能不太好，可能在最优点附近晃来晃去，hit 不到最优点。两次参数的更新也有可能互相抵消掉，造成目标函数震荡的比较剧烈。

为了克服两种方法的缺点，现在一般采用的是一种折中手段，**mini-batch gradient decent，小批的梯度下降**，这种方法把数据分为若干个批，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，计算量也不是很大。基本上现在的梯度下降都是基于 mini-batch 的，所以 Keras 的模块中经常会出现 batch_size，就是指这个。

顺便说一句，Keras 中用的优化器 SGD 是 stochastic gradient descent 的缩写，但不代表是一个样本就更新一回，还是基于 mini-batch 的。

epochs

epochs 指的就是训练过程中数据将被“轮”多少次，就这样。

序贯 (Sequential) 模型

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**。

输入指定数据的 shape

模型需要知道输入数据的shape，因此，`Sequential` 的第一层需要接受一个关于输入数据shape的参数，后面的各个层则可以自动的推导出中间数据的shape，因此不需要为每个层都指定这个参数。有几种方法来为第一层指定输入数据的shape

- 传递一个 `input_shape` 的关键字参数给第一层，`input_shape` 是一个tuple类型的数据，其中也可以填入 `None`，如果填入 `None` 则表示此位置可能是任何正整数。数据的batch大小不应包含在其中。
- 有些2D层，如 `Dense`，支持通过指定其输入维度 `input_dim` 来隐含的指定输入数据shape。一些3D的时域层支持通过参数 `input_dim` 和 `input_length` 来指定输入shape。
- 如果你需要为输入指定一个固定大小的batch_size（常用于stateful RNN网络），可以传递 `batch_size` 参数到一个层中，例如你想指定输入张量的batch大小是32，数据shape是（6，8），则你需要传递 `batch_size=32` 和 `input_shape=(6,8)`。

```
model = Sequential()
model.add(Dense(32, input_dim=784))
```

```
model = Sequential()
model.add(Dense(32, input_shape=784))
```

编译

在训练模型之前，我们需要通过 `compile` 来对学习过程进行配置。`compile` 接收三个参数：

- 优化器optimizer：该参数可指定为已预定义的优化器名，如 `rmsprop`、`adagrad`，或一个 `Optimizer` 类的对象，详情见[optimizers](#)
- 损失函数loss：该参数为模型试图最小化的目标函数，它可为预定义的损失函数名，如 `categorical_crossentropy`、`mse`，也可以为一个损失函数。详情见[losses](#)
- 指标列表metrics：对分类问题，我们一般将该列表设置为 `metrics=['accuracy']`。指标可以是一个预定义指标的名字,也可以是一个用户定制的函数.指标函数应该返回单个张量,或一个完成 `metric_name -> metric_value` 映射的字典.请参考[性能评估](#)

例子：类似 VGG 的卷积神经网络

```
import numpy as np
import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras.optimizers import SGD

# Generate dummy data
x_train = np.random.random((100, 100, 100, 3))
y_train = keras.utils.to_categorical(np.random.randint(10, size=(100, 1)), num_classes=10)
x_test = np.random.random((20, 100, 100, 3))
y_test = keras.utils.to_categorical(np.random.randint(10, size=(20, 1)), num_classes=10)

model = Sequential()
# input: 100x100 images with 3 channels -> (100, 100, 3) tensors.
# this applies 32 convolution filters of size 3x3 each.
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(100, 100, 3)))
model.add(Conv2D(32, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(10, activation='softmax'))

sgd = SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)
model.compile(loss='categorical_crossentropy', optimizer=sgd)

model.fit(x_train, y_train, batch_size=32, epochs=10)
score = model.evaluate(x_test, y_test, batch_size=32)
```

函数式（Functional）模型

函数式模型应用更为广泛，序贯模型是函数式模型的一种特殊情况。

```
from keras.models import Model
```

图片预处理

图片生成器ImageDataGenerator

```
keras.preprocessing.image.ImageDataGenerator(featurewise_center=False,
        samplewise_center=False,
        featurewise_std_normalization=False,
        samplewise_std_normalization=False,
        zca_whitening=False,
        rotation_range=0.,
        width_shift_range=0.,
        height_shift_range=0.,
        shear_range=0.,
        zoom_range=0.,
        channel_shift_range=0.,
        fill_mode='nearest',
        cval=0.,
        horizontal_flip=False,
        vertical_flip=False,
        rescale=None,
        preprocessing_function=None,
        data_format=K.image_data_format())
```

用以生成一个batch的图像数据，支持实时数据提升。训练时该函数会无限生成数据，直到达到规定的epoch次数为止。

基于 Region Proposal 的深度学习目标检测算法

| 算法 | 优缺点 |
|---------------------|--|
| R-CNN | <p>存在问题：</p> <p>(1) 训练分为多个阶段，步骤繁琐：微调网络+训练 SVM+训练边框回归器</p> <p>(2) 训练耗时，占用磁盘空间大：5000 张图像产生几百 G 的特征文件</p> <p>(3) 速度慢：使用 GPU, VGG16 模型处理一张图像需要 47s。</p> |
| SPP-NET | <p>SPP-NET 相比于 R-CNN 加快目标检测的速度,但是依然存在着很多问题：</p> <p>(1) 训练分为多个阶段，步骤繁琐：微调网络+训练 SVM+训练训练边框回归器</p> <p>(2) SPP-NET 在微调网络的时候固定了卷积层，只对全连接层进行微调，而对于一个新的任务，有必要对卷积层也进行微调。(分类的模型提取的特征更注重高层语义，而目标检测任务除了语义信息还需要目标的位置信息)</p> |
| Fast R-CNN | <p>Fast R-CNN 融合了 R-CNN 和 SPP-NET 的精髓，并且引入多任务损失函数，使整个网络的训练和测试变得十分方便。在 Pascal VOC2007 训练集上训练,在 VOC2007 测试的结果为 66.9%(mAP), 如果使用 VOC2007+2012 训练集训练，在 VOC2007 上测试结果为 70%（数据集的扩充能大幅提高目标检测性能）。使用 VGG16 每张图像总共需要 3s 左右。</p> <p>缺点：region proposal 的提取使用 selective search，目标检测时间大多消耗在这上面（提 region proposal 2~3s，而提特征分类只需 0.32s），无法满足实时应用，而且并没有实现真正意义上的端到端训练测试（region proposal 使用 selective search 先提取出来）。那么有没有可能直接使用 CNN 直接产生 region proposal 并对其分类？Faster R-CNN 框架就是符合这样需要的目标检测框架。</p> |
| Faster R-CNN | <p>Faster R-CNN 将一直以来分离的 region proposal 和 CNN 分类融合到了一起，使用端到端的网络进行目标检测，无论在速度上还是精度上都得到了不错的提高。然而 Faster R-CNN 还是达不到实时的目标检测，预先获取 region proposal，然后在对每个 proposal 分类计算量还是比较大。比较幸运的是 YOLO 这类目标检测方法的出现让实时性也变的成为可能。</p> |

除了基于 Region Proposal 的深度学习目标检测算法，还有基于回归方法的深度学习目标检测算法。<https://zhuanlan.zhihu.com/p/21412911>)

CNN 实践一

```
Model loaded...
Epoch 1/10
3777/3777 [=====] - 717s - loss: 2.1216 - acc: 0.4024
Epoch 2/10
3777/3777 [=====] - 721s - loss: 1.6569 - acc: 0.4543
Epoch 3/10
3777/3777 [=====] - 706s - loss: 1.5980 - acc: 0.4631
Epoch 4/10
3777/3777 [=====] - 705s - loss: 1.5124 - acc: 0.4874
Epoch 5/10
3777/3777 [=====] - 704s - loss: 1.4417 - acc: 0.5036
Epoch 6/10
3777/3777 [=====] - 705s - loss: 1.2986 - acc: 0.5443
Epoch 7/10
3777/3777 [=====] - 705s - loss: 1.1778 - acc: 0.5854
Epoch 8/10
3777/3777 [=====] - 704s - loss: 1.1185 - acc: 0.5994
Epoch 9/10
3777/3777 [=====] - 703s - loss: 1.0512 - acc: 0.6277
Epoch 10/10
3777/3777 [=====] - 704s - loss: 1.0186 - acc: 0.6352
Number of images 1000
Image size: (100, 100, 3)
Done!
```

Reference:

- [1] <http://keras-cn.readthedocs.io/en/latest/>
- [2] <http://dataunion.org/24014.html>
- [3] <https://zhuanlan.zhihu.com/p/21412911>