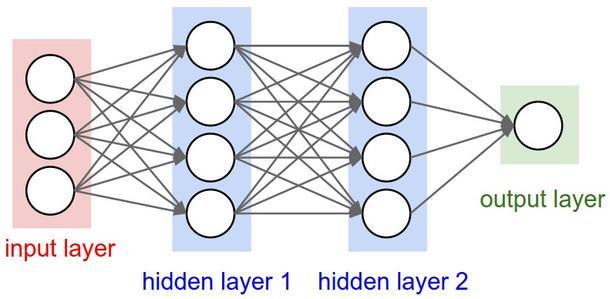
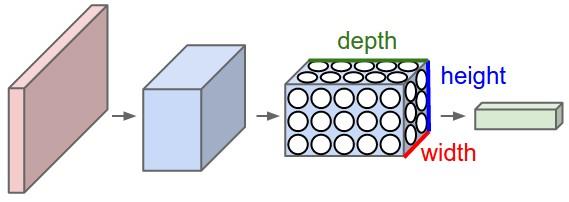
# 基于Python的卷积神经网络和特征提取

[卷积神经网络](http://www.yidianzixun.com/m/channel/keyword/%E5%8D%B7%E7%A7%AF%E7%A5%9E%E7%BB%8F%E7%BD%91%E7%BB%9C)（ConvNets）是受生物启发的MLPs（多层感知器），它们有着不同类别的层，并且每层的工作方式与普通的MLP层也有所差异。如果你对ConvNets感兴趣，这里有个很好的教程CS231n – Convolutional Neural Newtorks for Visual Recognition。CNNs的体系结构如下所示：



常规的[神经网络](http://www.yidianzixun.com/m/channel/keyword/%E7%A5%9E%E7%BB%8F%E7%BD%91%E7%BB%9C)（来自CS231n网站）



ConvNet网络体系结构（来自CS231n网站）

如你所见，ConvNets工作时伴随着3D卷积并且在不断转变着这些3D卷积。我在这篇文章中不会再重复整个CS231n的教程，所以如果你真的感兴趣，请在继续阅读之前先花点时间去学习一下。

**Lasagne 和 nolearn**

Lasagne和nolearn是我最喜欢使用的[深度学习](http://www.yidianzixun.com/m/channel/keyword/%E6%B7%B1%E5%BA%A6%E5%AD%A6%E4%B9%A0)[Python](http://www.yidianzixun.com/m/channel/keyword/python)包。Lasagne是基于Theano的，所以[GPU](http://www.yidianzixun.com/m/channel/keyword/gpu)的加速将大有不同，并且其对神经网络创建的声明方法也很有帮助。nolearn库是一个神经网络软件包实用程序集（包含Lasagne），它在神经网络体系结构的创建过程上、各层的检验等都能够给我们很大的帮助。

在这篇文章中我要展示的是，如何使用一些卷积层和池化层来建立一个简单的ConvNet体系结构。我还将向你展示如何使用ConvNet去训练一个[特征提取](http://www.yidianzixun.com/m/channel/keyword/%E7%89%B9%E5%BE%81%E6%8F%90%E5%8F%96)器，在使用如SVM、Logistic回归等不同的模型之前使用它来进行特征提取。大多数人使用的是预训练ConvNet模型，然后删除最后一个输出层，接着从ImageNets数据集上训练的ConvNets网络提取特征。这通常被称为是迁移学习，因为对于不同的问题你可以使用来自其它的ConvNets层，由于ConvNets的第一层过滤器被当做是一个边缘探测器，所以它们可以用来作为其它问题的普通特征探测器。

**加载MNIST数据集**

MNIST数据集是用于数字识别最传统的数据集之一。我们使用的是一个面向Python的版本，但先让我们导入需要使用的包：

import matplotlib import matplotlib.pyplot as plt import matplotlib.cm as cm from urllib import urlretrieve import cPickle as pickle import [os](http://www.yidianzixun.com/m/channel/keyword/os) import gzip import numpy as np import theano import lasagne from lasagne import layers from lasagne.updates import nesterov\_momentum from nolearn.lasagne import NeuralNet from nolearn.lasagne import visualize from sklearn.metrics import classification\_report from sklearn.metrics import confusion\_matrix

正如你所看到的，我们导入了用于绘图的matplotlib包，一些用于下载MNIST数据集的原生Python模块，numpy， theano，lasagne，nolearn 以及 scikit-learn库中用于模型评估的一些函数。

然后，我们定义一个加载MNIST数据集的函数（这个功能与Lasagne教程上使用的非常相似）

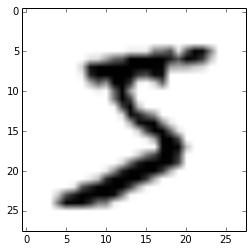
def load\_dataset: url = 'http://deeplearning.net/data/mnist/mnist.pkl.gz' [filename](http://www.yidianzixun.com/m/channel/keyword/filename) = 'mnist.pkl.gz' if not os.path.exists(filename): print("Downloading MNIST dataset...") urlretrieve(url, filename) with gzip.open(filename, 'rb') as f: data = pickle.load(f) X\_train, y\_train = data[0] X\_val, y\_val = data[1] X\_test, y\_test = data[2] X\_train = X\_train.reshape((-1, 1, 28, 28)) X\_val = X\_val.reshape((-1, 1, 28, 28)) X\_test = X\_test.reshape((-1, 1, 28, 28)) y\_train = y\_train.astype(np.uint8) y\_val = y\_val.astype(np.uint8) y\_test = y\_test.astype(np.uint8) return X\_train, y\_train, X\_val, y\_val, X\_test, y\_test

正如你看到的，我们正在下载处理过的MNIST数据集，接着把它拆分为三个不同的数据集，分别是：训练集、验证集和测试集。然后重置图像内容，为之后的Lasagne输入层做准备，与此同时，由于GPU/theano数据类型的限制，我们还把numpy的数据类型转换成了uint8。

随后，我们准备加载MNIST数据集并检验它：

X\_train, y\_train, X\_val, y\_val, X\_test, y\_test = load\_dataset plt.imshow(X\_train[0][0], cmap=cm.binary)

这个代码将输出下面的图像（我用的是IPython Notebook）



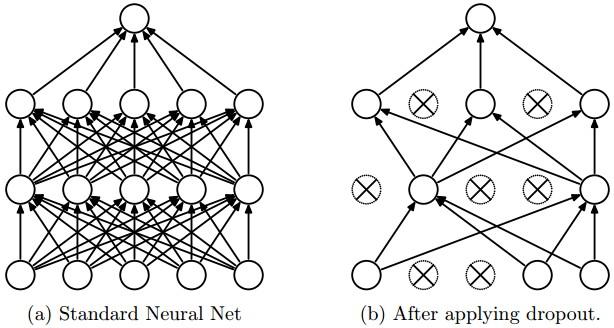
一个MNIST数据集的数字实例（该实例是5）

**ConvNet体系结构与训练**

现在，定义我们的ConvNet体系结构，然后使用单GPU/[CPU](http://www.yidianzixun.com/m/channel/keyword/cpu)来训练它（我有一个非常廉价的GPU，但它很有用）

net1 = NeuralNet( layers=[('input', layers.InputLayer), ('conv2d1', layers.Conv2DLayer), ('maxpool1', layers.MaxPool2DLayer), ('conv2d2', layers.Conv2DLayer), ('maxpool2', layers.MaxPool2DLayer), ('dropout1', layers.DropoutLayer), ('dense', layers.DenseLayer), ('dropout2', layers.DropoutLayer), ('output', layers.DenseLayer), ], # input layer input\_shape=(None, 1, 28, 28), # layer conv2d1 conv2d1\_num\_filters=32, conv2d1\_filter\_size=(5, 5), conv2d1\_nonlinearity=lasagne.nonlinearities.rectify, conv2d1\_W=lasagne.init.GlorotUniform, # layer maxpool1 maxpool1\_pool\_size=(2, 2), # layer conv2d2 conv2d2\_num\_filters=32, conv2d2\_filter\_size=(5, 5), conv2d2\_nonlinearity=lasagne.nonlinearities.rectify, # layer maxpool2 maxpool2\_pool\_size=(2, 2), # dropout1 dropout1\_p=0.5, # dense dense\_num\_units=256, dense\_nonlinearity=lasagne.nonlinearities.rectify, # dropout2 dropout2\_p=0.5, # output output\_nonlinearity=lasagne.nonlinearities.softmax, output\_num\_units=10, # optimization method params update=nesterov\_momentum, update\_learning\_rate=0.01, update\_momentum=0.9, max\_epochs=10, verbose=1, ) # Train the network nn = net1.fit(X\_train, y\_train)

如你所视，在layers的参数中，我们定义了一个有层名称/类型的元组字典，然后定义了这些层的参数。在这里，我们的体系结构使用的是两个卷积层，两个池化层，一个全连接层（稠密层，dense layer）和一个输出层。在一些层之间也会有dropout层，dropout层是一个正则化矩阵，随机的设置输入值为零来避免过拟合（见下图）。



Dropout层效果（来自CS231n网站）

调用训练方法后，nolearn包将会显示学习过程的状态，我的机器使用的是低端的的GPU，得到的结果如下：

# Neural Network with 160362 learnable parameters ## Layer information # name size --- -------- -------- 0 input 1x28x28 1 conv2d1 32x24x24 2 maxpool1 32x12x12 3 conv2d2 32x8x8 4 maxpool2 32x4x4 5 dropout1 32x4x4 6 dense 256 7 dropout2 256 8 output 10 epoch train loss valid loss train/val valid acc dur ------- --------- --- 1 0.85204 0.16707 5.09977 0.95174 33.71s 2 0.27571 0.10732 2.56896 0.96825 33.34s 3 0.20262 0.08567 2.36524 0.97488 33.51s 4 0.16551 0.07695 2.15081 0.97705 33.50s 5 0.14173 0.06803 2.08322 0.98061 34.38s 6 0.12519 0.06067 2.06352 0.98239 34.02s 7 0.11077 0.05532 2.00254 0.98427 33.78s 8 0.10497 0.05771 1.81898 0.98248 34.17s 9 0.09881 0.05159 1.91509 0.98407 33.80s 10 0.09264 0.04958 1.86864 0.98526 33.40s

正如你看到的，最后一次的精度可以达到0.98526，是这10个单元训练中的一个相当不错的性能。

**预测和混淆矩阵**

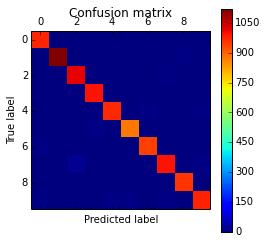
现在，我们使用这个模型来预测整个测试集：

preds = net1.predict(X\_test)

我们还可以绘制一个混淆矩阵来检查神经网络的分类性能：

cm = confusion\_matrix(y\_test, preds) plt.matshow(cm) plt.title('Confusion matrix') plt.colorbar plt.ylabel('True label') plt.xlabel('Predicted label') plt.show

上面的代码将绘制下面的混淆矩阵：



混淆矩阵

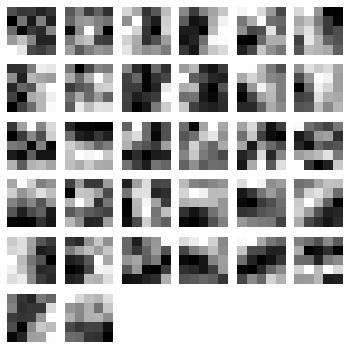
如你所视，对角线上的分类更密集，表明我们的分类器有一个良好的性能。

**过滤器的可视化**

我们还可以从第一个卷积层中可视化32个过滤器：

visualize.plot\_conv\_weights(net1.layers\_['conv2d1'])

上面的代码将绘制下面的过滤器：



第一层的5x5x32过滤器

如你所视，nolearn的plot\_conv\_weights函数在我们指定的层中绘制出了所有的过滤器。

**Theano层的功能和特征提取**

现在可以创建theano编译的函数了，它将前馈输入数据输送到结构体系中，甚至是你感兴趣的某一层中。接着，我会得到输出层的函数和输出层前面的稠密层函数。

dense\_layer = layers.get\_output(net1.layers\_['dense'], deterministic=True) output\_layer = layers.get\_output(net1.layers\_['output'], deterministic=True) input\_var = net1.layers\_['input'].input\_var f\_output = theano.function([input\_var], output\_layer) f\_dense = theano.function([input\_var], dense\_layer)

如你所视，我们现在有两个theano函数，分别是f\_output和f\_dense（用于输出层和稠密层）。请注意，在这里为了得到这些层，我们使用了一个额外的叫做“deterministic”的参数，这是为了避免dropout层影响我们的前馈操作。

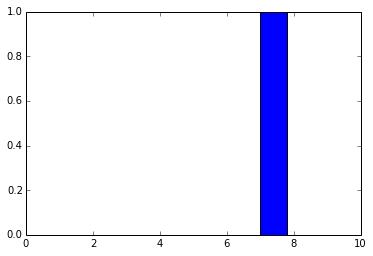
现在，我们可以把实例转换为输入格式，然后输入到theano函数输出层中：

[instance](http://www.yidianzixun.com/m/channel/keyword/instance) = X\_test[0][None, :, :] %timeit -n 500 f\_output(instance) 500 loops, best of 3: 858 µs per loop

如你所视，f\_output函数平均需要858µs。我们同样可以为这个实例绘制输出层激活值结果：

pred = f\_output(instance) N = pred.shape[1] plt.bar(range(N), pred.ravel)

上面的代码将绘制出下面的图：



输出层激活值

正如你所看到的，数字被认为是7。事实是为任何网络层创建theano函数都是非常有用的，因为你可以创建一个函数（像我们以前一样）得到稠密层（输出层前一个）的激活值，然后你可以使用这些激活值作为特征，并且使用你的神经网络作为特征提取器而不是分类器。现在，让我们为稠密层绘制256个激活单元：

pred = f\_dense(instance) N = pred.shape[1] plt.bar(range(N), pred.ravel)

上面的代码将绘制下面的图：

稠密层激活值

现在，你可以使用输出的这256个激活值作为线性分类器如Logistic回归或支持向量机的特征了。

最后，我希望你会喜欢这个教程。

**原文链接：**Convolutional neural networks and feature extraction with Python（译者/刘帝伟 审校/刘翔宇、朱正贵 责编/周建丁）