

E16 Deep Learning (C++/Python)

17341111 Xuehai Liu

2019 年 12 月 26 日

目录

1	The CIFAR-10 dataset	2
2	Convolutional Neural Networks (CNNs / ConvNets)	2
2.1	Architecture Overview	2
2.2	Layers used to build ConvNets	3
2.2.1	Convolutional Layer	3
2.2.2	Pooling Layer	4
3	Deep Learning Softwares	5
4	Tasks	5
5	Codes and Results	5
6	实验心得	7

1 The CIFAR-10 dataset

The CIFAR-10 dataset (<http://www.cs.toronto.edu/~kriz/cifar.html>) consists of 60000 32×32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.

The dataset is divided into five training batches and one test batch, each with 10000 images. The test batch contains exactly 1000 randomly-selected images from each class. The training batches contain the remaining images in random order, but some training batches may contain more images from one class than another. Between them, the training batches contain exactly 5000 images from each class. Here are the classes in the dataset, as well as 10 random images from each:

The classes are completely mutually exclusive. There is no overlap between automobiles and trucks. "Automobile" includes sedans, SUVs, things of that sort. "Truck" includes only big trucks. Neither includes pickup trucks.

2 Convolutional Neural Networks (CNNs / ConvNets)

Chinese version: <https://www.zybuluo.com/hanbingtao/note/485480>

English version: <http://cs231n.github.io/convolutional-networks/#layers>

2.1 Architecture Overview

Regular Neural Nets don't scale well to full images. In CIFAR-10, images are only of size $32 \times 32 \times 3$ (32 wide, 32 high, 3 color channels), so a single fully-connected neuron in a first hidden layer of a regular Neural Network would have $32 * 32 * 3 = 3072$ weights. This amount still seems manageable, but clearly this fully-connected structure does not scale to larger images. For example, an image of more respectable size, e.g. $200 \times 200 \times 3$, would lead to neurons that have $200 * 200 * 3 = 120,000$ weights. Moreover, we would almost certainly want to have several such neurons, so the parameters would add up quickly! Clearly, this full connectivity is wasteful and the huge number of parameters would quickly lead to overfitting.

Convolutional Neural Networks take advantage of the fact that the input consists of images and they constrain the architecture in a more sensible way. In particular, unlike a regular Neural Network, the layers of a ConvNet have neurons arranged in 3 dimensions: width, height, depth. (Note that the word depth here refers to the third dimension of an activation volume, not to the depth of a full Neural Network, which can refer to the total number of layers in a network.) For example, the input images in CIFAR-10 are an input volume of activations, and the volume has dimensions $32 \times 32 \times 3$ (width, height, depth respectively). As we will soon see, the neurons in a layer will only be connected to a small region of the layer before it, instead of all of the neurons in a fully-connected manner. Moreover, the final output layer would for CIFAR-10 have dimensions $1 \times 1 \times 10$, because by the

end of the ConvNet architecture we will reduce the full image into a single vector of class scores, arranged along the depth dimension. Here is a visualization:

Left: A regular 3-layer Neural Network. Right: A ConvNet arranges its neurons in three dimensions (width, height, depth), as visualized in one of the layers. Every layer of a ConvNet transforms the 3D input volume to a 3D output volume of neuron activations. In this example, the red input layer holds the image, so its width and height would be the dimensions of the image, and the depth would be 3 (Red, Green, Blue channels).

2.2 Layers used to build ConvNets

a simple ConvNet is a sequence of layers, and every layer of a ConvNet transforms one volume of activations to another through a differentiable function. We use three main types of layers to build ConvNet architectures: **Convolutional Layer**, **Pooling Layer**, and **Fully-Connected Layer** (exactly as seen in regular Neural Networks). We will stack these layers to form a full ConvNet architecture.

Example Architecture: Overview. We will go into more details below, but a simple ConvNet for CIFAR-10 classification could have the architecture **[INPUT - CONV - RELU - POOL - FC]**. In more detail:

- INPUT $[32 \times 32 \times 3]$ will hold the raw pixel values of the image, in this case an image of width 32, height 32, and with three color channels R,G,B.
- CONV layer will compute the output of neurons that are connected to local regions in the input, each computing a dot product between their weights and a small region they are connected to in the input volume. This may result in volume such as $[32 \times 32 \times 12]$ if we decided to use 12 filters.
- RELU layer will apply an elementwise activation function, such as the $\max(0, x)$ thresholding at zero. This leaves the size of the volume unchanged ($[32 \times 32 \times 12]$).
- POOL layer will perform a downsampling operation along the spatial dimensions (width, height), resulting in volume such as $[16 \times 16 \times 12]$.
- FC (i.e. fully-connected) layer will compute the class scores, resulting in volume of size $[1 \times 1 \times 10]$, where each of the 10 numbers correspond to a class score, such as among the 10 categories of CIFAR-10. As with ordinary Neural Networks and as the name implies, each neuron in this layer will be connected to all the numbers in the previous volume.

2.2.1 Convolutional Layer

To summarize, the Conv Layer:

- Accepts a volume of size $W_1 \times H_1 \times D_1$

- Requires four hyperparameters:
 - Number of filters K ,
 - their spatial extent F ,
 - the stride S ,
 - the amount of zero padding P .
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
 - $W_2 = (W_1 - F + 2P)/S + 1$
 - $H_2 = (H_1 - F + 2P)/S + 1$ (i.e. width and height are computed equally by symmetry)
 - $D_2 = K$
- With parameter sharing, it introduces $F \cdot F \cdot D_1$ weights per filter, for a total of $(F \cdot F \cdot D_1) \cdot K$ weights and K biases.
- In the output volume, the d -th depth slice (of size $W_2 \times H_2$) is the result of performing a valid convolution of the d -th filter over the input volume with a stride of S , and then offset by d -th bias.

A common setting of the hyperparameters is $F = 3, S = 1, P = 1$. However, there are common conventions and rules of thumb that motivate these hyperparameters.

2.2.2 Pooling Layer

It is common to periodically insert a Pooling layer in-between successive Conv layers in a ConvNet architecture. Its function is to progressively reduce the spatial size of the representation to reduce the amount of parameters and computation in the network, and hence to also control overfitting. The Pooling Layer operates independently on every depth slice of the input and resizes it spatially, using the **MAX** operation. The most common form is a pooling layer with filters of size 2×2 applied with a stride of 2 downsamples every depth slice in the input by 2 along both width and height, discarding 75% of the activations. Every MAX operation would in this case be taking a max over 4 numbers (little 2×2 region in some depth slice). The depth dimension remains unchanged. More generally, the pooling layer:

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires two hyperparameters:
 - their spatial extent F ,
 - the stride S ,

- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
 - $W_2 = (W_1 - F)/S + 1$
 - $H_2 = (H_1 - F)/S + 1$
 - $D_2 = D_1$
- Introduces zero parameters since it computes a fixed function of the input
- For Pooling layers, it is not common to pad the input using zero-padding.

3 Deep Learning Softwares

4 Tasks

1. Given the data set in the first section, please implement a convolutional neural network to calculate the accuracy rate. The major steps involved are as follows:
 - (a) Reading the input image.
 - (b) Preparing filters.
 - (c) Conv layer: Convolving each filter with the input image.
 - (d) ReLU layer: Applying ReLU activation function on the feature maps (output of conv layer).
 - (e) Max Pooling layer: Applying the pooling operation on the output of ReLU layer.
 - (f) Stacking conv, ReLU, and max pooling layers
2. You can refer to the codes in `cs231n`. Don't use Keras, TensorFlow, PyTorch, Theano, Caffe, and other deep learning softwares.
3. Please submit a file named `E16_YourNumber.rar`, which should includes the code files and the result pictures, and send it to `ai_201901@foxmail.com`

5 Codes and Results

由于本次实验中的模块都已经定义好，因此只需要实现 `main` 函数，调用可选用的包对 `cnn` 模型进行训练，测试即可。因此本次实验的代码相当简单，主要的时间放在了配置环境上面。下面是实现上述功能的主函数代码：

```
1 from cs231n.classifiers.cnn import *
2 from cs231n.data_utils import *
3 from cs231n.solver import *
```

```

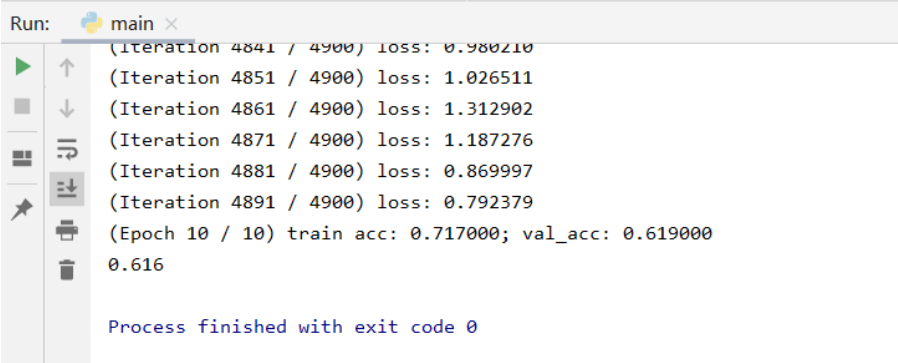
4  if __name__ == "__main__":
5
6      model = ThreeLayerConvNet()
7      data = get_CIFAR10_data()
8
9      solver = Solver(model,data)
10     X_test = data['X_test']
11     Y_test = data['y_test']
12     solver.train()
13     print (solver.check_accuracy(X_test,Y_test))

```

其中用到了一些模块。

- ThreeLayerConvNet 为一个三层的卷积神经网络。
- get_cifar10_data 为 data_utils 中实现的函数，用于从 cifar10 数据库中 load 数据并返回一个字典。字典中分别包含训练集、验证集和测试集的数据和标注
{ 'X_train': X_train, 'y_train': y_train, 'X_val': X_val, 'y_val': y_val, 'X_test': X_test, 'y_test': y_test, }
- Solver 为 solver.py 中定义的类。solver 定义了一个 model 的训练过程和测试过程，输入参数为 model 和 data。此处主要使用了 Solver 的两个 method: train 和 check_accuracy。两个函数的作用分别是使用 data 的测试集部分对 model 进行训练，而 check_accuracy 的作用是对 model 使用测试集的数据进行测试并返回测试集上的 acc。

测试结果如下. 训练了 10 个 epoch 之后得到的测试集准确率为 0.616.



```

Run: main x
(iteration 4841 / 4900) loss: 0.980210
(iteration 4851 / 4900) loss: 1.026511
(iteration 4861 / 4900) loss: 1.312902
(iteration 4871 / 4900) loss: 1.187276
(iteration 4881 / 4900) loss: 0.869997
(iteration 4891 / 4900) loss: 0.792379
(Epoch 10 / 10) train acc: 0.717000; val_acc: 0.619000
0.616

Process finished with exit code 0

```

图 1: result

6 实验心得

这次的实验主要考察了 cnn 神经网络的实现和在 cifar10 数据集上的测试。下面将分步骤分别对整个训练和测试的过程进行说明：

- 第一步，准备数据集

众所周知，训练深度学习神经网络的最关键的步骤就是数据集的准备。本次实验中，数据集是由官方提供的标准的数据集，包含训练、验证、测试集三个部分。运行 `cs231n/datasets/get-datasets.sh` 文件，获取 cifar10 数据集。dataset 如下，共有五个训练、验证文件和一个测试文件。



名称	修改日期	类型	大小
batches.meta	2009/3/31 12:45	META 文件	1 KB
data_batch_1	2009/3/31 12:32	文件	30,309 KB
data_batch_2	2009/3/31 12:32	文件	30,308 KB
data_batch_3	2009/3/31 12:32	文件	30,309 KB
data_batch_4	2009/3/31 12:32	文件	30,309 KB
data_batch_5	2009/3/31 12:32	文件	30,309 KB
readme.html	2009/6/5 4:47	Chrome HTML D...	1 KB
test_batch	2009/3/31 12:32	文件	30,309 KB

图 2: dataset

- 第二步，构造模型

然后，使用 `cnn` 构造模型。`cnn` 模块使用了 `cs231n/classifiers/cnn.py`,

网络所构建的结构如下：conv - relu - 2x2 max pool - affine - relu - affine - softmax

网络对应的层的作用分别如下：conv 层计算输入层的卷积输出，relu 层为线性激活函数，pool 池化层进行降维，affine 层进行仿射变换，最后 softmax 对整个神经网络的输入进行激活输出。

构造网络使用的超参数如下：

```
input_dim=(3, 32, 32), num_filters=32, filter_size=7, hidden_dim=100, num_classes=10,
weight_scale=1e-3, reg=0.0, dtype=np.float32
```

- 第三步，训练模型

其次，训练模型。训练模型采用常见的深度学习学习过程：先前向传播计算网络输出，然后计算损失，将损失通过反向传播到整个神经网络的参数网，对参数进行优化。本次实验在训练集上一共训练了 10 个 epoch，4900 轮次，在训练集上获得了 0.717 的准确率，在验证集上取得了 0.619 的准确率。最后的 loss 已经小于一，可见模型已经趋于拟合。

```
(Iteration 4821 / 4900) loss: 1.049451
(Iteration 4831 / 4900) loss: 0.691572
(Iteration 4841 / 4900) loss: 0.980210
(Iteration 4851 / 4900) loss: 1.026511
(Iteration 4861 / 4900) loss: 1.312902
(Iteration 4871 / 4900) loss: 1.187276
(Iteration 4881 / 4900) loss: 0.869997
(Iteration 4891 / 4900) loss: 0.792379
(Epoch 10 / 10) train acc: 0.717000; val_acc: 0.619000
```

图 3: train

- 第四步，测试模型

测试模型的过程就比较简单了。通过将测试集的测试数据 X feed 到网络中获取输出，将 output 和 ground truth 的 labels 进行比较，相同则增加 acc。计算得到本次实验在 cifar10 的测试集上的测试 acc 为 0.616。