

Tensorflow 利用 CNN 实现 Cifar-10 图像识别

刘浚哲

北京大学物理学院 1500011370

April 19, 2018

1. 数据预处理¹

由于 CIFAR 训练数据集只有 50000 张, 而测试数据集就有 10000 张. 参考前面已有的研究, 我们选择进行 Data Augmentation: 即对训练数据集进行扩充, 例如简单地对图片进行随机裁剪, 翻转, 亮度, 对比度等各方面进行调节之后, 这样可使训练样本增大, 同时也可以使分类器增强对图片畸变的适应能力, 从而避免过拟合问题. 我采用的数据读取程序 loader 与 data augmentation 程序都是采用的是 Tensorflow 官网上给出的样例程序 `cifar_input.py` 其中已经写好了数据的提取程序以及对图片进行随机处理的过程.

2. 网络模型及参数选择

我在程序中所使用的网络模型是 Benjamin Graham 在 2014 年发表的论文 Spatially-sparse convolutional neural networks². 文章中提出了一种适用于识别图片像素大多数为稀疏矩阵的网络模型.

文章指出: 慢池化过程可以保留更多的空白信息 (例如距离信息等), 对于手写数字识别等图片识别问题都将会比较有利. 因此在每一层卷积层之后便加一层 2×2 的池化层进行最大池化. 文章指出了两种图片处理的网络:

2.1. DeepCNet(l, k)

该种网络主体结构由 $l + 1$ 层卷积层中夹杂着 l 层池化层所构成, 而第 i 层卷积层的输出通道个数为 $n \times k$. 且第一层的卷积核为 3×3 形状, 随后的卷积核均为 2×2 的形状.

DeepCNet(l, k) 所满足的输入形状为: $N = 3 \times 2^k$, 因此一个 $l = 5, k = 100$ 的网络结构大致为:

$$\begin{aligned} input - 100C3 - MP2 - 200C2 - MP2 - \\ - 300C2 - MP2 - 400C2 - MP2 - \\ - 500C2 - MP2 - 600C2 - output \end{aligned}$$

2.2. DeepCNiN(l, k)

该网络结构实际上是在 DeepCNet 当中在每一个池化层后面都加上了一个 1×1 的小卷积核, 但并不改变输出的通道数, 因此也被称为“网络当中的网络”. 我们在 DeepCNet(l, k) 的每一个池化层及最后一个卷积层后面都加上一个小卷积核, 即可得到 DeepCNiN(l, k):

¹ Email Address: 1500011370@pku.edu.cn

² Spatially-sparse convolutional neural networks, Benjamin Graham, Dept of Statistics, University of Warwick, UK, September 23, 2014

$input - 100C3 - MP2 - 100C1 -$
 $- 200C2 - MP2 - 200C1 -$
 $- 300C2 - MP2 - 300C1 -$
 $- 400C2 - MP2 - 400C1 -$
 $- 500C2 - MP2 - 500C1 -$
 $- 600C2 - 600C1 - output$

2.3. 训练超参数的选择

文章中指出: 对于尺寸为 $n \approx N/3$ 的图片, 选择 l 的参数为:

$$l = \log_2(n)$$

因此对 CIFAR-10: $n = 32$, 故选择 $l = 5$. 而对于 k , 我们考虑到要控制参数个数, 因此选择为 $k = 100$.

在每一层卷积层和两层全连接层后面, 我们都加了 dropout 函数来随机丢弃节点, 以避免过拟合的产生. 经过调试显示: 在底层神经网络施加 dropout 的效果并没有在高层施加的效果显著, 因此我们对 6 层卷积层以及 2 层全连接层的 *keep_proportion* 分别为: 1.0, 1.0, 0.9, 0.8, 0.7, 0.6, 0.5, 0.5.

而在调整参数过程中, 我们发现选择 *learning_rate* = 0.001 恒定不变的学习率在 10000 次循环次数过后, 其 *loss_function* 的数值不能显著的减少了, 因此我们选择逐渐递减的 *learning_rate*.

3. 模型调参过程

首先作为比较, 我采用了一个较为简单的网络模型作为效果的参考, 网络主体为十层, 其中七层为卷积, 池化与规范层的合并层, 第八, 九层为全连接层, 最后一层为 softmax 层. 网络结构如下:

$input - 64C5 - MP3 - LRN -$
 $- 64C5 - MP3 - LRN -$
 $- 64C5 - MP3 - LRN -$
 $- 64C5 - MP3 - LRN -$
 $- 64C5 - MP3 - LRN -$
 $- 64C5 - MP3 - LRN -$
 $- 64C5 - MP3 - LRN -$
 $- 64C5 - LRN - MP3 -$
 $- FC - FC - softmax - output$

其中卷积核的大小为 5, 池化层的核大小为 3. 在池化层之后, 我们加上了一层 LRN 层, 为”局部响应归一化 (Local Response Normalization)”层, 它的作用是对局部神经元的活动创建竞争机制, 使得其中响应比较大的值变得相对更大, 并抑制其他反馈较小的神经元, 从而增强模型的泛化能力.

对该模型进行 10000 次循环训练, 学习率固定为 0.001, 训练后得到测试集上的训练结果为:

```

step 9730, loss=0.64 (145.4 examples/sec; 0.880 sec/batch)
step 9740, loss=0.80 (146.0 examples/sec; 0.877 sec/batch)
step 9750, loss=0.75 (145.1 examples/sec; 0.882 sec/batch)
step 9760, loss=0.76 (143.6 examples/sec; 0.892 sec/batch)
step 9770, loss=0.73 (140.9 examples/sec; 0.908 sec/batch)
step 9780, loss=0.76 (141.3 examples/sec; 0.906 sec/batch)
step 9790, loss=0.87 (135.6 examples/sec; 0.944 sec/batch)
step 9800, loss=0.69 (144.8 examples/sec; 0.884 sec/batch)
step 9810, loss=0.68 (146.5 examples/sec; 0.874 sec/batch)
step 9820, loss=0.71 (137.6 examples/sec; 0.930 sec/batch)
step 9830, loss=0.68 (144.2 examples/sec; 0.888 sec/batch)
step 9840, loss=0.86 (143.5 examples/sec; 0.892 sec/batch)
step 9850, loss=0.80 (142.7 examples/sec; 0.897 sec/batch)
step 9860, loss=0.73 (144.3 examples/sec; 0.887 sec/batch)
step 9870, loss=0.73 (144.9 examples/sec; 0.883 sec/batch)
step 9880, loss=0.78 (145.9 examples/sec; 0.877 sec/batch)
step 9890, loss=0.89 (145.1 examples/sec; 0.882 sec/batch)
step 9900, loss=0.61 (145.5 examples/sec; 0.879 sec/batch)
step 9910, loss=0.69 (144.4 examples/sec; 0.886 sec/batch)
step 9920, loss=0.85 (137.3 examples/sec; 0.933 sec/batch)
step 9930, loss=0.72 (138.3 examples/sec; 0.926 sec/batch)
step 9940, loss=0.72 (137.1 examples/sec; 0.933 sec/batch)
step 9950, loss=0.71 (134.8 examples/sec; 0.950 sec/batch)
step 9960, loss=0.51 (145.8 examples/sec; 0.878 sec/batch)
step 9970, loss=0.72 (145.9 examples/sec; 0.878 sec/batch)
step 9980, loss=1.09 (144.6 examples/sec; 0.885 sec/batch)
step 9990, loss=0.71 (145.1 examples/sec; 0.882 sec/batch)
precision @ 1=0.778

Process finished with exit code 0

```

图 1: 普通模型, 学习率固定为 0.001, 无 dropout, test accuracy 为 77.8%

分析问题: 可能是固定的学习率使得在后面的循环中无法收敛到局部最优, 由此考虑采用递减的学习率, 并在每一层池化过后加上 dropout 层, 每一层的 keep 比率从底层网络向高层网络递减, 即在底层网络中几乎不丢弃神经元, 而在高层网络中则随机丢弃一半的节点. 经过如上操作之后, 经过 10000 次模型训练, 得到测试集准确率为 57.3%... 这个结果比较出人意料, 我也忘记对结果进行截图了.

接下来就是对 DeepCNI 进行测试, 固定学习率为 0.001, 我们按照 2.2 当中的网络模型进行 10000 次训练, 在 3000 步左右时, 得到如下结果:

```

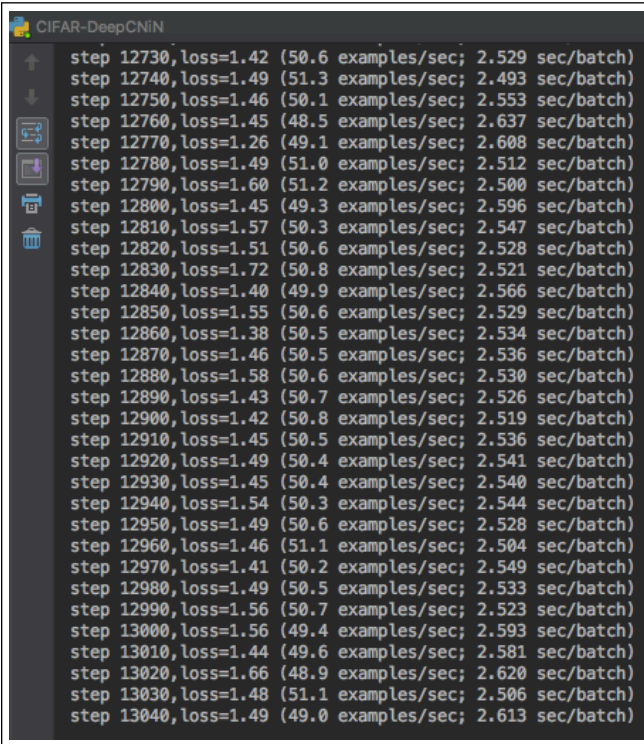
step 2740, loss=1.60 (53.8 examples/sec; 2.380 sec/batch)
step 2750, loss=1.59 (53.4 examples/sec; 2.395 sec/batch)
step 2760, loss=1.43 (54.3 examples/sec; 2.356 sec/batch)
step 2770, loss=1.86 (54.9 examples/sec; 2.330 sec/batch)
step 2780, loss=1.67 (53.0 examples/sec; 2.417 sec/batch)
step 2790, loss=1.51 (53.8 examples/sec; 2.381 sec/batch)
step 2800, loss=1.63 (53.6 examples/sec; 2.389 sec/batch)
step 2810, loss=1.62 (53.8 examples/sec; 2.381 sec/batch)
step 2820, loss=1.57 (53.4 examples/sec; 2.398 sec/batch)
step 2830, loss=1.51 (55.1 examples/sec; 2.325 sec/batch)
step 2840, loss=1.58 (53.6 examples/sec; 2.386 sec/batch)
step 2850, loss=1.63 (53.6 examples/sec; 2.387 sec/batch)
step 2860, loss=1.62 (53.6 examples/sec; 2.390 sec/batch)
step 2870, loss=1.63 (53.7 examples/sec; 2.384 sec/batch)
step 2880, loss=1.52 (53.7 examples/sec; 2.384 sec/batch)
step 2890, loss=1.65 (53.3 examples/sec; 2.402 sec/batch)
step 2900, loss=1.60 (54.2 examples/sec; 2.362 sec/batch)
step 2910, loss=1.62 (48.9 examples/sec; 2.615 sec/batch)
step 2920, loss=1.68 (53.0 examples/sec; 2.413 sec/batch)
step 2930, loss=1.70 (54.3 examples/sec; 2.356 sec/batch)
step 2940, loss=1.75 (53.9 examples/sec; 2.373 sec/batch)
step 2950, loss=1.58 (53.0 examples/sec; 2.415 sec/batch)
step 2960, loss=1.62 (54.3 examples/sec; 2.359 sec/batch)
step 2970, loss=1.64 (54.8 examples/sec; 2.338 sec/batch)
step 2980, loss=1.61 (54.0 examples/sec; 2.371 sec/batch)
step 2990, loss=1.58 (54.1 examples/sec; 2.367 sec/batch)
step 3000, loss=1.64 (54.2 examples/sec; 2.361 sec/batch)
step 3010, loss=1.61 (53.8 examples/sec; 2.380 sec/batch)
step 3020, loss=1.72 (52.2 examples/sec; 2.452 sec/batch)

```

图 2: DeepCNI 模型, 学习率固定为 0.001, 无 dropout, 效果显著地不如普通模型

该模型训练起来比普通模型慢了一倍左右... 而且横向比较在 3000 步时, 普通模型的 loss 已经降到了 1.2 左右, 而 DeepCNI 的 loss 还在 1.7 左右反复震荡. 我怀疑仍然是学习率过大的原因, 因此改而采用

decay 的 learning rate, 并在每一个卷积层 (不包括 1×1 的小网络) 之后加入 dropout, 重新训练后得到结果如下:



step	loss	examples/sec	sec/batch
step 12730	loss=1.42	(50.6 examples/sec)	2.529 sec/batch)
step 12740	loss=1.49	(51.3 examples/sec)	2.493 sec/batch)
step 12750	loss=1.46	(50.1 examples/sec)	2.553 sec/batch)
step 12760	loss=1.45	(48.5 examples/sec)	2.637 sec/batch)
step 12770	loss=1.26	(49.1 examples/sec)	2.608 sec/batch)
step 12780	loss=1.49	(51.0 examples/sec)	2.512 sec/batch)
step 12790	loss=1.60	(51.2 examples/sec)	2.500 sec/batch)
step 12800	loss=1.45	(49.3 examples/sec)	2.596 sec/batch)
step 12810	loss=1.57	(50.3 examples/sec)	2.547 sec/batch)
step 12820	loss=1.51	(50.6 examples/sec)	2.528 sec/batch)
step 12830	loss=1.72	(50.8 examples/sec)	2.521 sec/batch)
step 12840	loss=1.40	(49.9 examples/sec)	2.566 sec/batch)
step 12850	loss=1.55	(50.6 examples/sec)	2.529 sec/batch)
step 12860	loss=1.38	(50.5 examples/sec)	2.534 sec/batch)
step 12870	loss=1.46	(50.5 examples/sec)	2.536 sec/batch)
step 12880	loss=1.58	(50.6 examples/sec)	2.530 sec/batch)
step 12890	loss=1.43	(50.7 examples/sec)	2.526 sec/batch)
step 12900	loss=1.42	(50.8 examples/sec)	2.519 sec/batch)
step 12910	loss=1.45	(50.5 examples/sec)	2.536 sec/batch)
step 12920	loss=1.49	(50.4 examples/sec)	2.541 sec/batch)
step 12930	loss=1.45	(50.4 examples/sec)	2.540 sec/batch)
step 12940	loss=1.54	(50.3 examples/sec)	2.544 sec/batch)
step 12950	loss=1.49	(50.6 examples/sec)	2.528 sec/batch)
step 12960	loss=1.46	(51.1 examples/sec)	2.504 sec/batch)
step 12970	loss=1.41	(50.2 examples/sec)	2.549 sec/batch)
step 12980	loss=1.49	(50.5 examples/sec)	2.533 sec/batch)
step 12990	loss=1.56	(50.7 examples/sec)	2.523 sec/batch)
step 13000	loss=1.56	(49.4 examples/sec)	2.593 sec/batch)
step 13010	loss=1.44	(49.6 examples/sec)	2.581 sec/batch)
step 13020	loss=1.66	(48.9 examples/sec)	2.620 sec/batch)
step 13030	loss=1.48	(51.1 examples/sec)	2.506 sec/batch)
step 13040	loss=1.49	(49.0 examples/sec)	2.613 sec/batch)

图 3:DeepCNiN 模型, 学习率递减, 有 dropout, 效果显著地不如普通模型

可以看见在 13000 步左右还是处于 1.4 左右的 loss... 因此我最终还是放弃了该网络模型. 转而从普通模型上进行修改.

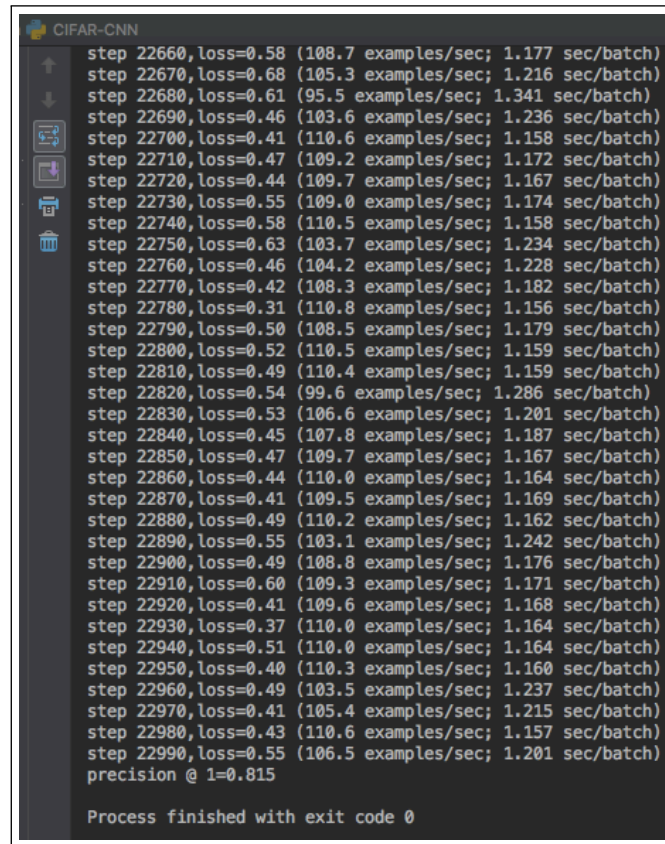
受课上同学的启发, 规范层与池化层的位置不同, 所带来的训练结果是完全不同的, 因此我对普通模型当中的规范层与池化层的位置进行了调换, 并在最后两层全连接层当中加入了 0.5 的 dropout, 得到模型如下:

$input - 64C5 - LRN - MP3 -$
 $- 64C5 - LRN - MP3 -$
 $- 64C5 - LRN - MP3 -$
 $- 64C5 - LRN - MP3 -$
 $- 64C5 - LRN - MP3 -$
 $- 64C5 - LRN - MP3 -$
 $- 64C5 - LRN - MP3 -$
 $- 64C5 - LRN - MP3 -$
 $- FC - dropout - FC - dropout -$
 $- softmax - output$

同时, 学习率固定为 0.00007, 确保在循环训练很多次后仍能学到东西. 并且受到 DeepCNiN 文献的启发, 我将激活函数改为 leaky relu, 定义如下:

$$leaky_relu(x) = \max(0, x) + negative_slope \times \min(0, x)$$

其中,negative_slope 是一个小于 1 的非零数, 我们取为 0.37. 对网络模型进行 23000 次训练, 训练结果最终如下图:



```
CIFAR-CNN
step 22660,loss=0.58 (108.7 examples/sec; 1.177 sec/batch)
step 22670,loss=0.68 (105.3 examples/sec; 1.216 sec/batch)
step 22680,loss=0.61 (95.5 examples/sec; 1.341 sec/batch)
step 22690,loss=0.46 (103.6 examples/sec; 1.236 sec/batch)
step 22700,loss=0.41 (110.6 examples/sec; 1.158 sec/batch)
step 22710,loss=0.47 (109.2 examples/sec; 1.172 sec/batch)
step 22720,loss=0.44 (109.7 examples/sec; 1.167 sec/batch)
step 22730,loss=0.55 (109.0 examples/sec; 1.174 sec/batch)
step 22740,loss=0.58 (110.5 examples/sec; 1.158 sec/batch)
step 22750,loss=0.63 (103.7 examples/sec; 1.234 sec/batch)
step 22760,loss=0.46 (104.2 examples/sec; 1.228 sec/batch)
step 22770,loss=0.42 (108.3 examples/sec; 1.182 sec/batch)
step 22780,loss=0.31 (110.8 examples/sec; 1.156 sec/batch)
step 22790,loss=0.50 (108.5 examples/sec; 1.179 sec/batch)
step 22800,loss=0.52 (110.5 examples/sec; 1.159 sec/batch)
step 22810,loss=0.49 (110.4 examples/sec; 1.159 sec/batch)
step 22820,loss=0.54 (99.6 examples/sec; 1.286 sec/batch)
step 22830,loss=0.53 (106.6 examples/sec; 1.201 sec/batch)
step 22840,loss=0.45 (107.8 examples/sec; 1.187 sec/batch)
step 22850,loss=0.47 (109.7 examples/sec; 1.167 sec/batch)
step 22860,loss=0.44 (110.0 examples/sec; 1.164 sec/batch)
step 22870,loss=0.41 (109.5 examples/sec; 1.169 sec/batch)
step 22880,loss=0.49 (110.2 examples/sec; 1.162 sec/batch)
step 22890,loss=0.55 (103.1 examples/sec; 1.242 sec/batch)
step 22900,loss=0.49 (108.8 examples/sec; 1.176 sec/batch)
step 22910,loss=0.60 (109.3 examples/sec; 1.171 sec/batch)
step 22920,loss=0.41 (109.6 examples/sec; 1.168 sec/batch)
step 22930,loss=0.37 (110.0 examples/sec; 1.164 sec/batch)
step 22940,loss=0.51 (110.0 examples/sec; 1.164 sec/batch)
step 22950,loss=0.40 (110.3 examples/sec; 1.160 sec/batch)
step 22960,loss=0.49 (103.5 examples/sec; 1.237 sec/batch)
step 22970,loss=0.41 (105.4 examples/sec; 1.215 sec/batch)
step 22980,loss=0.43 (110.6 examples/sec; 1.157 sec/batch)
step 22990,loss=0.55 (106.5 examples/sec; 1.201 sec/batch)
precision @ 1=0.815

Process finished with exit code 0
```

图 4: 普通模型, 学习率固定为 0.00007, 有 dropout, test accuracy 为 81.5%

可见交换过后, 确实对于准确率的提升有较大的改善.

4. 源代码

见附录.

References:

- [1] Learning Multiple Layers of Features from Tiny Images Alex Krizhevsky April 8, 2009
- [2] Spatially-sparse convolutional neural networks Benjamin Graham September 23, 2014
- [3] FAST AND ACCURATE DEEP NETWORK LEARNING BY EXPONENTIAL LINEAR UNITS (ELUS) Djork-Arne' Clevert, Thomas Unterthiner Sepp Hochreiter Johannes Kepler University, Linz, Austria Feb 22, 2016
- [4] Deep Residual Learning for Image Recognition Kaiming He Xiangyu Zhang Shaoqing Ren Jian Sun Microsoft Research Dec 10, 2015