

第二周机器学习

本文公式显示需要使用Mathjax，然后令人悲伤的是github不支持Mathjax 您可以将这篇md文件pull下来，使用您本地的markdown解析器解析 没有必要在公示显示上浪费时间，您也可以下载我本地生成的html用浏览器打开即可 或者您也可以下载我上传到github上的pdf [Mathjax开源项目地址](#)

关于机器学习的一些概念补充

深度学习与机器学习的关系



- 机器学习是实现人工智能的方法
- 深度学习是实现机器学习算法的技术

深度学习算法集合

- 卷积神经网络
- 循环神经网络
- 自动编码器
- 稀疏编码
- 深度信念网络
- 限制玻尔兹曼机
- 深度学习+强化学习 = 深度强化学习(AlphaGo)

深度学习进展

- 图像分类
- 机器翻译
- 图像生成
- AlphaGo

Test and Debug Your ML System

Debug the ML System

待补充

Machine learning diagnostic

待补充

Evaluating your hypothesis

待补充

决策树

这一章没有看吴恩达老师的视频，看的是中科院某博士讲的

Code

[参考博客链接](#)

```
1  #!/usr/bin/python
2  # -*- coding: UTF-8 -*-
3
4  import pandas as pd
5  import matplotlib.pyplot as plt
6  # 这里我们使用scikit-learn(sklearn)作为我们机器学习的模块
7  from sklearn.datasets.california_housing import fetch_california_housing
8  from sklearn import tree
9  import pydotplus
10 from io import BytesIO
11 from sklearn.model_selection import train_test_split
12 from sklearn.model_selection import GridSearchCV
13 from sklearn.ensemble import RandomForestRegressor
14
15 house_data = fetch_california_housing()
16 # print(house_data.DESCR)
17
18 # 定义树的最大深度等于2
19 dtr = tree.DecisionTreeRegressor(max_depth=2)
20 dtr.fit(house_data.data[:, [6, 7]], house_data.target)
21
22
23 dot_data = \
24     tree.export_graphviz(dtr,
25                           out_file=None,
26                           feature_names=house_data.feature_names[6:8],
27                           filled=True,
28                           impurity=False,
29                           rounded=True
30                           )
31 #
32 graph = pydotplus.graph_from_dot_data(dot_data)
33 graph.get_nodes()[7].set_fillcolor('#FFF2DD')
34 img_data = graph.create_png()
35 plt.imshow(plt.imread(BytesIO(img_data)))
36 # plt.show()
37 # graph.write_png('dtr_white_background.png')
38
39 # 拆分训练集 测试集
40 # 取0.1作为测试集
41 data_train, data_test, target_train, target_test = \
42     train_test_split(house_data.data, house_data.target, test_size = 0.1, random_state =
43                       42)
44 dtr = tree.DecisionTreeRegressor(random_state = 42)
```

```

44 dtr.fit(data_train, target_train)
45 print(dtr.score(data_test, target_test))
46
47
48 # 使用GridSearchCV帮助我们选择合适参数
49 tree_param_grid = {'min_samples_split': list((3, 6, 9)), 'n_estimators': list((10, 50,
100))}
50 grid = GridSearchCV(RandomForestRegressor(), param_grid=tree_param_grid, cv=5)
51 grid.fit(data_train, target_train)
52 print(grid.grid_scores_, grid.best_score_, grid.best_params_)
53

```

运行结果

```

1 0.637355881715626
2 (0.8074196516933743, {'min_samples_split': 6, 'n_estimators': 100})

```

(Ensemble learning)集成算法

Note: 集成算法并不是机器学习算法的一种, 而相当于把很多个机器学习算法拢在一块。

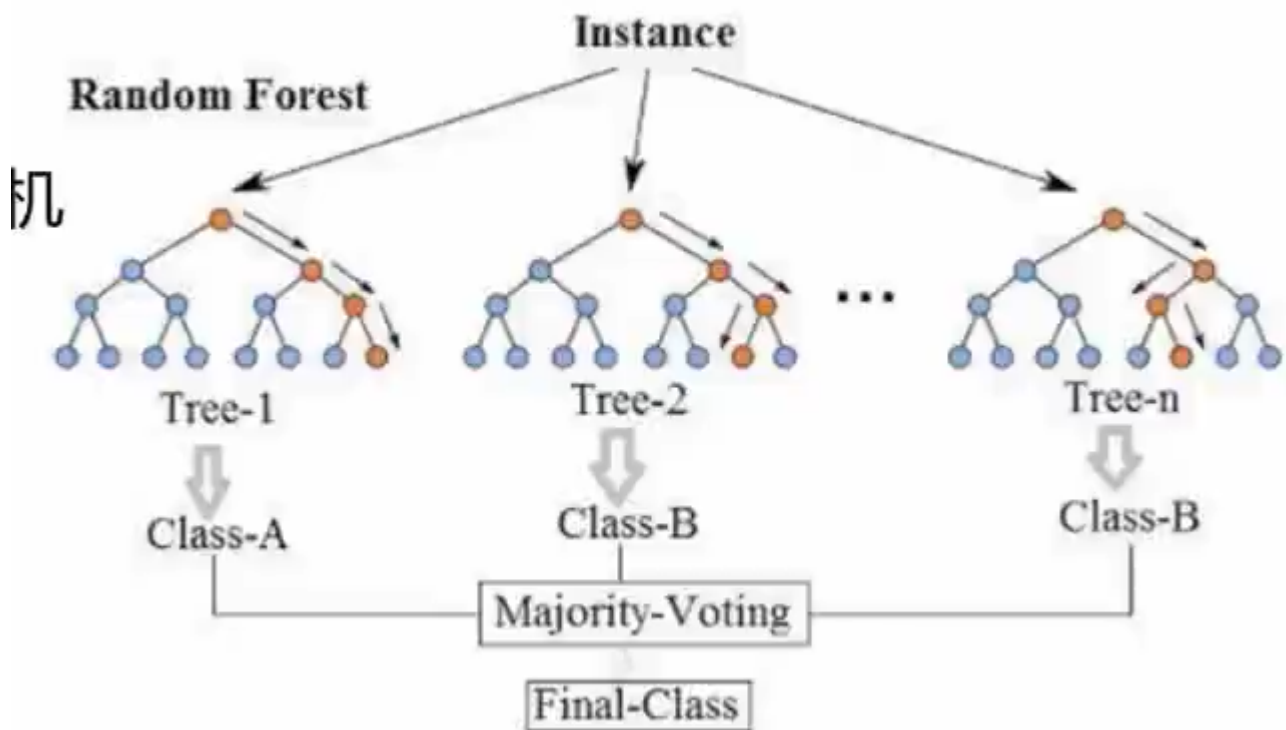
Ensemble learning介绍

- Ensemble learning
 - 目的: 让机器学习效果更好, 单个不行, 那就一群
 - Bagging: 训练多个分类器取平均: $f(x) = \frac{1}{M} \sum_{m=1}^M f_m(x)$
 - Boosting: 从弱学习器开始加强, 通过加权来进行训练
 - $F_m(x) = F_{m-1}(x) + \operatorname{argmin}_h \sum_{i=1}^n L(y_i, F_{m-1}(x_i) + h(x_i))$ 加入一个数, 要比原来强
 - Stacking: 聚合多个分类和回归模型 (可以分阶段来做)

Bagging模型

- 全称: bootstrap aggregation(说白了就是并行训练一堆分类器)
- 最典型的代表就是随机森林
- 随机: 数据采样随机(一般取60%-80%, 有放回), 特征选择随机 (获得一系列随机的树以后对当中特征也按照60%-80%进行采样)
- 之所以要进行随机, 是要保证泛化能力
- 森林: 很多决策树并行放在一起
- 理论上越多的树效果会越好, 但实际上基本超过一定数量就差不多上下浮动了

Random Forest Simplified



随机森林优势

- 可以处理很多维度(feature很多)的数据，并且不用做特征选择
- 在训练完后，它能够给出哪些feature比较重要
- 容易做成并行化方法，速度比较快
- 可以进行可视化展示，便于分析

Boosting模型

- 典型代表：AdaBoost, Xgboost
- Adaboost会根据前一次的分类效果调整数据权重
- 解释：如果某一个数据在这次分错了，那么下一次就会给他更大的权重
- 结果：每个分类器根据自身准确性来确定各自的权重，再合体

Stacking模型

- 堆叠：很暴力，拿来一堆直接上
- 可以堆叠各种各样的分类器（KNN、SVM、RF等）
- 分阶段：第一阶段得出各自结果，第二阶段再用前一阶段结果训练
- 为了刷结果，不择手段

堆叠在一起确实能使得准确率得到提升，但是速度是个问题

集成算法是竞赛与论文神奇，当我们更关注与结果时不妨来试试！

卷及神经网络入门

传统神经网络问题

- 参数过多
 - 举例
 - 假设图像大小 1000×1000
 - 假设下一层神经元为 10^6
 - 全连接参数为 $1000 \times 1000 \times 10^6 = 10^{12}$
 - 容易过拟合, 需要更多训练数据
 - 解决方案-局部连接
 - 原理: 图像的区域性
 - 举例 - 图像大小 1000×1000 - 下一层神经元为 10^6 - 局部连接范围为 10×10 - 全连接参数为 $10 \times 10 \times 10^6 = 10^8$
 - 解决方案 参数共享
 - 原理: 图像特征与位置无关
 - 举例
 - 图像大小 1000×1000
 - 下一层神经元为 10^6
 - 局部连接范围为 10×10
 - 全链接参数为 $10 \times 10 = 10^2$

卷积

设图像 $f(x)$, 模板是 $g(x)$, 然后将模板 $g(x)$ 在模板中移动, 每到一个位置, 就把 $f(x)$ 与 $g(x)$ 的定义域相交的元素进行乘积并且求和, 得出新的图像一点, 就是被卷积后的图像. 模版又称为卷积核. 卷积核做一个矩阵的形状. 输出size = 输入size - 卷积核size + (1,1)

步长的概念: 模板在图像上每一次移动的个数。

定义几个参数:

- 输入图片大小 $W \times W$
- Filter大小 $F \times F$
- 步长 S
- padding的像素数 P padding是在图像的周围补0
padding的大小应该是使输出size不变 输出图片大小为

对于多通道图像, 卷积Filter维度大小也跟着通道数扩大即可, 这样我们就可以得到一个单通道卷积和。那该如何获得多通道的图像卷积呢? 此时我们应该是使用多张卷积核。

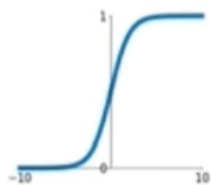
多个卷积核的物理含义是提取图像中的多种特征

Q: 卷基层, 输入三通道, 输出192通道, 卷积核大小是33, 问该卷基层有多少参数? A: $(3 \times 3 \times 3) \times 192 = 5184$

激活函数

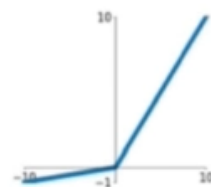
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



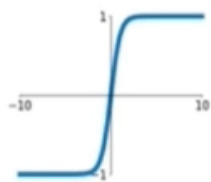
Leaky ReLU

$$\max(0.1x, x)$$



tanh

$$\tanh(x)$$

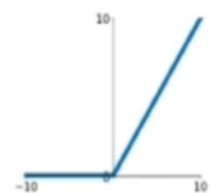


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

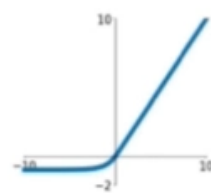
ReLU

$$\max(0, x)$$



ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



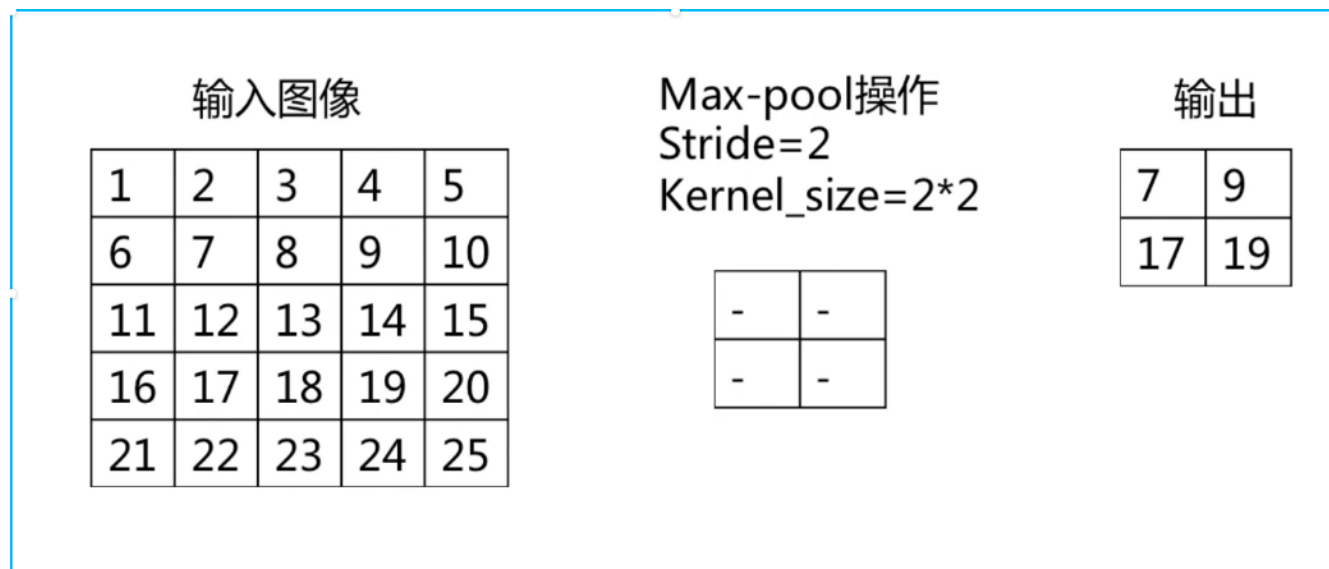
5	6	-2
-10	0	4
9	-1	7

ReLU激活

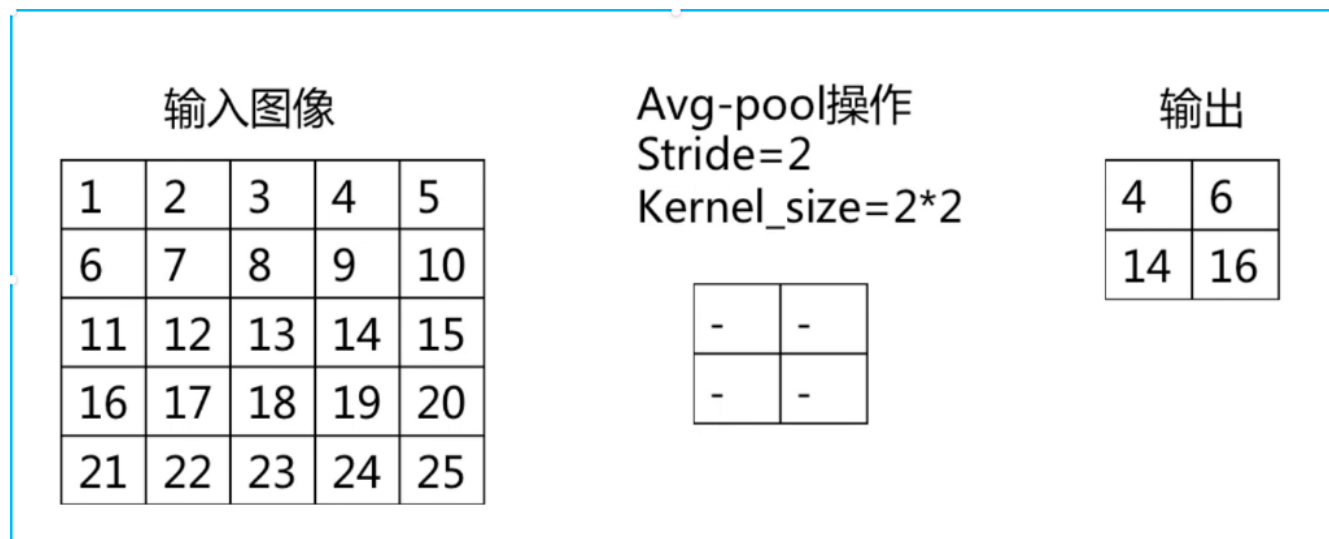
5	6	0
0	0	4
9	0	7

池化

最大化池化



平均值池化



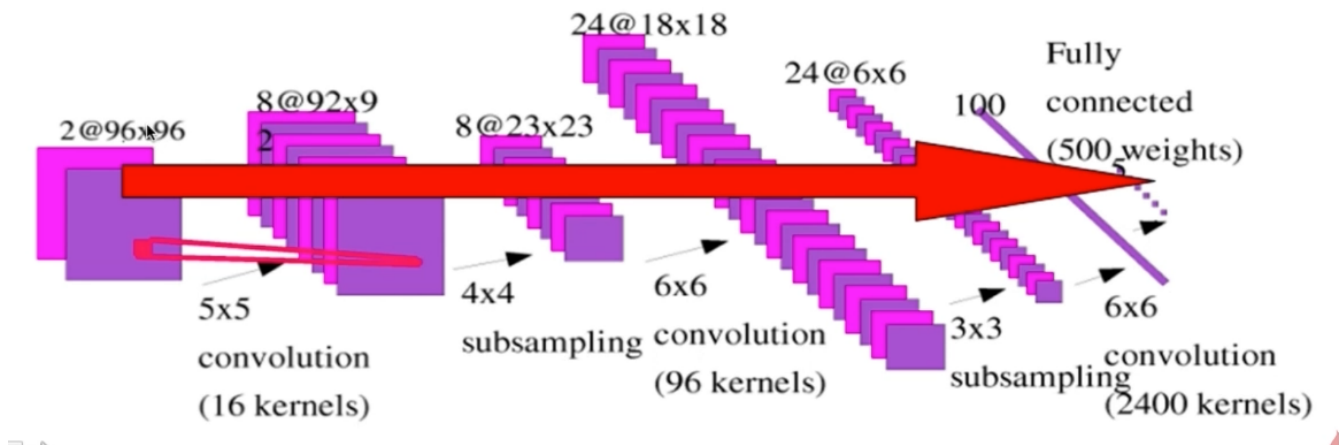
- 使用时不重叠、不补零
- 没有用于求导的参数
- 池化层的参数为步长和池化核大小
- 用于减小图像尺寸，从而减少计算量
- 一定程度解决平移鲁棒性

全连接

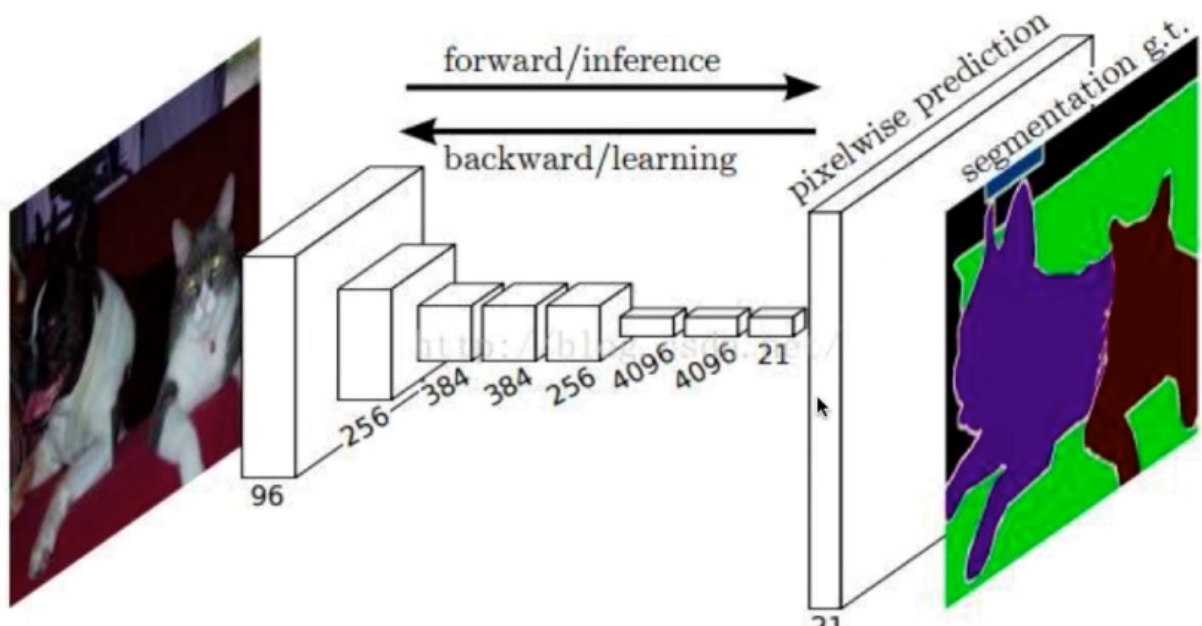
- 将上一层输出展开并连接到每一个神经元上
- 全连接层之后可以加全连接层，但是不可以加卷积层、池化层
- 全连接层即标准神经网络的层
- 相比卷积层，参数数目较大，占比比较大
- 参数数目 = 输入通道数目 * 输出通道数目

卷积神经网络结构

- 卷积神经网络=卷积层+池化层+全连接层



- 全卷积神经网络=卷积层+池化层



Code

最刺激的事情来了，如果之前没用过卷积神经网络的话，这里我们使用卷积神经网络对cifar-10图像数据进行分类
先加上三层隐藏层实现一个标准多层神经网络

```

1  #!/usr/bin/env python
2  # coding: utf-8
3
4
5  import tensorflow as tf
6  import os
7  import pickle
8  import numpy as np
9
10

```



```

11 CIFAT_DIR = '../cifar-10-batches-py'
12 print(os.listdir(CIFAT_DIR))
13
14
15 def load_data(filename):
16     """read data from data file"""
17     with open(os.path.join(filename), 'rb') as f:
18         # data = pickle.load(f, encoding='bytes')
19
20         # Python2.7代码
21         data = pickle.load(f)
22         return data['data'], data['labels']
23
24
25 class CifarData:
26     def __init__(self, filenames, need_shuffle):
27         all_data = []
28         all_labels = []
29         # 关于zip函数 具体看
30         # http://www.cnblogs.com/frydsh/archive/2012/07/10/2585370.html
31         for filename in filenames:
32             data, labels = load_data(filename)
33             for item, label in zip(data, labels):
34                 all_data.append(item)
35                 all_labels.append(label)
36
37         # 关于 vstack函数
38         # https://www.cnblogs.com/nkh222/p/8932369.html
39         self._data = np.vstack(all_data)
40         # 归一化处理
41         self._data = self._data / 127.5 - 1;
42         self._labels = np.hstack(all_labels)
43         print(self._data.shape)
44         print(self._labels.shape)
45         self._num_examples = self._data.shape[0]
46         self._need_shuffle = need_shuffle
47         self._indicator = 0
48         if self._need_shuffle:
49             self._shuffle_data()
50
51     def _shuffle_data(self):
52         # [0,1,2,3,4] => [2,1,3,4,0]
53         p = np.random.permutation(self._num_examples)
54         self._data = self._data[p]
55         self._labels = self._labels[p]
56
57     def next_batch(self, batch_size):
58         """return batch_size examples as a batch """
59         end_indicator = self._indicator + batch_size
60         if end_indicator > self._num_examples:
61             if self._need_shuffle:
62                 self._shuffle_data()
63                 self._indicator = 0
64             end_indicator = batch_size

```

```

64         else:
65             raise Exception("have no more examples")
66         if end_indicator > self._num_examples:
67             raise Exception('batch size is larger than all examles')
68         batch_data = self._data[self._indicator: end_indicator]
69         batch_labels = self._labels[self._indicator: end_indicator]
70         self._indicator = end_indicator
71         return batch_data, batch_labels
72
73
74 train_filenames = [os.path.join(CIFAT_DIR, 'data_batch_%d' % i) for i in range(1, 6)]
75 test_filenames = [os.path.join(CIFAT_DIR, 'test_batch')]
76
77 train_data = CifarData(train_filenames, True)
78 test_data = CifarData(test_filenames, False)
79 # batch_data, batch_labels = train_data.next_batch(10)
80 # print(batch_data, batch_labels)
81
82
83 # None 代表输入样本数是不确定的
84 x = tf.placeholder(tf.float32, [None, 3072])
85 # None
86 y = tf.placeholder(tf.int64, [None])
87
88 '''
89 # (3072,10)
90 w = tf.get_variable('w', [x.get_shape()[-1], 10],
91                     initializer=tf.random_normal_initializer(0, 1))
92 # (10, )
93 b = tf.get_variable('b', [10], initializer=tf.constant_initializer(0.0))
94 # [None,3072] * [3072,10] = [None,10]
95 y_ = tf.matmul(x, w) + b
96 '''
97
98 hidden1 = tf.layers.dense(x, 100, activation=tf.nn.relu)
99 hidden2 = tf.layers.dense(hidden1, 50, activation=tf.nn.relu)
100 hidden3 = tf.layers.dense(hidden2, 10, activation=tf.nn.relu)
101
102 # 这里10表示神经元个数 这个函数就等价于上面写的那么多代码
103 y_ = tf.layers.dense(hidden3, 10)
104
105 # y_ -> softmax
106 # y -> one_hot
107 # loss = ylogy_
108 loss = tf.losses.sparse_softmax_cross_entropy(labels=y, logits=y_)
109
110
111
112 '''
113 # [None,10]
114 p_y_1 = tf.nn.sigmoid(y_)
115 # 这里-1参数表示缺省值 保证为1列即可

```

```

116 y_resaped = tf.reshape(y, (-1, 1))
117 y_resaped_float = tf.cast(y_resaped, tf.float32)
118 # 计算loss
119 loss = tf.reduce_mean(tf.square(y_resaped_float - p_y_1))
120 '''
121
122 # indices
123 predict = tf.argmax(y_, 1)
124 correct_prediction = tf.equal(predict, y)
125 accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float64))
126
127 with tf.name_scope('train_op'):
128     # 这里1e-3是学习率 learning rate AdamOptimizer是梯度下降的一个变种
129     train_op = tf.train.AdamOptimizer(1e-3).minimize(loss)
130
131 '''
132 到此为止我们的计算图搭建完成
133 '''
134
135 init = tf.global_variables_initializer()
136 batch_size = 20
137 train_steps = 10000
138 test_steps = 100
139
140 with tf.Session() as sess:
141     sess.run(init)
142     for i in range(train_steps):
143         batch_data, batch_labels = train_data.next_batch(batch_size)
144         loss_val, accu_val, _ = sess.run(
145             [loss, accuracy, train_op],
146             feed_dict={x: batch_data, y: batch_labels})
147         if (i+1) % 500 == 0:
148             print('[Train] Step: %d, loss: %4.5f, acc: %4.5f' % (i+1, loss_val,
accu_val))
149         if (i+1) % 5000 == 0:
150             test_data = CifarData(test_filenames, False)
151             all_test_acc_val = []
152             for j in xrange(test_steps):
153                 test_batch_data, test_batch_labels \
154                     = test_data.next_batch(batch_size)
155                 test_acc_val = sess.run(
156                     [accuracy],
157                     feed_dict={
158                         x: test_batch_data,
159                         y: test_batch_labels
160                     }
161                 )
162                 all_test_acc_val.append(test_acc_val)
163             test_acc = np.mean(all_test_acc_val)
164             print('[Test] Step: %d, acc: %4.5f ' % (i+1, test_acc))

```

然后使用卷积和池化进行图像数据处理，搭建一个简单卷积神经网络

```
1  #!/usr/bin/env python
2  # coding: utf-8
3
4
5  import tensorflow as tf
6  import os
7  import pickle
8  import numpy as np
9
10
11  CIFAT_DIR = '../cifar-10-batches-py'
12  print(os.listdir(CIFAT_DIR))
13
14
15  def load_data(filename):
16      """read data from data file"""
17      with open(os.path.join(filename), 'rb') as f:
18          # data = pickle.load(f, encoding='bytes')
19
20          # Python2.7代码
21          data = pickle.load(f)
22          return data['data'], data['labels']
23
24
25  class CifarData:
26      def __init__(self, filenames, need_shuffle):
27          all_data = []
28          all_labels = []
29          # 关于zip函数 具体看
30          # http://www.cnblogs.com/frydsh/archive/2012/07/10/2585370.html
31          for filename in filenames:
32              data, labels = load_data(filename)
33              for item, label in zip(data, labels):
34                  all_data.append(item)
35                  all_labels.append(label)
36
37          # 关于 vstack函数
38          # https://www.cnblogs.com/nkh222/p/8932369.html
39          self._data = np.vstack(all_data)
40          # 归一化处理
41          self._data = self._data / 127.5 - 1;
42          self._labels = np.hstack(all_labels)
43          print(self._data.shape)
44          print(self._labels.shape)
45          self._num_examples = self._data.shape[0]
46          self._need_shuffle = need_shuffle
47          self._indicator = 0
48          if self._need_shuffle:
49              self._shuffle_data()
50
51      def _shuffle_data(self):
52          # [0,1,2,3,4] => [2,1,3,4,0]
53          p = np.random.permutation(self._num_examples)
54          self._data = self._data[p]
```

```

54     self._labels = self._labels[p]
55
56     def next_batch(self, batch_size):
57         """return batch_size examples as a batch """
58         end_indicator = self._indicator + batch_size
59         if end_indicator > self._num_examples:
60             if self._need_shuffle:
61                 self._shuffle_data()
62                 self._indicator = 0
63                 end_indicator = batch_size
64             else:
65                 raise Exception("have no more examples")
66         if end_indicator > self._num_examples:
67             raise Exception('batch size is larger than all examles')
68         batch_data = self._data[self._indicator: end_indicator]
69         batch_labels = self._labels[self._indicator: end_indicator]
70         self._indicator = end_indicator
71         return batch_data, batch_labels
72
73
74 train_filenames = [os.path.join(CIFAT_DIR, 'data_batch_%d' % i) for i in range(1, 6)]
75 test_filenames = [os.path.join(CIFAT_DIR, 'test_batch')]
76
77 train_data = CifarData(train_filenames, True)
78 test_data = CifarData(test_filenames, False)
79 # batch_data, batch_labels = train_data.next_batch(10)
80 # print(batch_data, batch_labels)
81
82
83 # None 代表输入样本数是不确定的
84 x = tf.placeholder(tf.float32, [None, 3072])
85 # None
86 y = tf.placeholder(tf.int64, [None])
87
88 # -1代表缺省 实际上x的维度应该理解为4维
89 x_image = tf.reshape(x, [-1, 3, 32, 32])
90 x_image = tf.transpose(x_image, perm = [0, 2, 3, 1])
91
92 # 卷积层
93 # conv1 : 神经元图、feature_map、输出图像
94 # padding = 'same|valid' same即使用padding
95 # 32*32
96 conv1 = tf.layers.conv2d(x_image,
97                           32, # 表示输出空间的维数（即卷积过滤器的数量）
98                           (3, 3),
99                           padding = 'same',
100                          activation = tf.nn.relu,
101                          name = 'conv1')
102 # 16 * 16
103 pooling1 = tf.layers.max_pooling2d(conv1,
104                                     (2, 2), # kernel size
105                                     (2, 2), # stride 表示卷积的纵向和横向的步长
106                                     name = 'pool1')

```

```

107
108 conv2 = tf.layers.conv2d(pooling1,
109                           32,
110                           (3,3),
111                           padding = 'same',
112                           activation = tf.nn.relu,
113                           name = 'conv2')
114 # 8 * 8
115 pooling2 = tf.layers.max_pooling2d(conv2,
116                                     (2,2), # kernel size
117                                     (2,2), # stride
118                                     name = 'pool2')
119
120 conv3 = tf.layers.conv2d(pooling2,
121                           32,
122                           (3,3),
123                           padding = 'same',
124                           activation = tf.nn.relu,
125                           name = 'conv3')
126
127 # 4 * 4 * 32
128 pooling3 = tf.layers.max_pooling2d(conv3,
129                                     (2,2), # kernel size
130                                     (2,2), # stride
131                                     name = 'pool3')
132 # [None, 4*4*32]
133 flatten = tf.layers.flatten(pooling3)
134
135
136
137 # 这里10表示神经元个数 这个函数就等价于上面写的那么多代码
138 y_ = tf.layers.dense(flatten, 10)
139
140
141 # y_->softmax
142 # y -> one_hot
143 # loss = ylogy_
144 loss = tf.losses.sparse_softmax_cross_entropy(labels=y, logits=y_)
145
146
147
148 '''
149 # [None,10]
150 p_y_1 = tf.nn.sigmoid(y_)
151 # 这里-1参数表示缺省值 保证为1列即可
152 y_resaped = tf.reshape(y, (-1, 1))
153 y_resaped_float = tf.cast(y_resaped, tf.float32)
154 # 计算loss
155 loss = tf.reduce_mean(tf.square(y_resaped_float - p_y_1))
156 '''
157
158 # indices
159 predict = tf.argmax(y_, 1)

```

```

160 correct_prediction = tf.equal(predict, y)
161 accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float64))
162
163 with tf.name_scope('train_op'):
164     # 这里1e-3是学习率 learning rate AdamOptimizer是梯度下降的一个变种
165     train_op = tf.train.AdamOptimizer(1e-3).minimize(loss)
166
167     '''
168     到此为止我们的计算图搭建完成
169     '''
170
171 init = tf.global_variables_initializer()
172 batch_size = 20
173 train_steps = 1000
174 test_steps = 100
175
176 with tf.Session() as sess:
177     sess.run(init)
178     for i in range(train_steps):
179         batch_data, batch_labels = train_data.next_batch(batch_size)
180         loss_val, accu_val, _ = sess.run(
181             [loss, accuracy, train_op],
182             feed_dict={x: batch_data, y: batch_labels})
183         if (i+1) % 500 == 0:
184             print('[Train] Step: %d, loss: %4.5f, acc: %4.5f' % (i+1, loss_val,
accu_val))
185         if (i+1) % 5000 == 0:
186             test_data = CifarData(test_filenames, False)
187             all_test_acc_val = []
188             for j in xrange(test_steps):
189                 test_batch_data, test_batch_labels \
190                     = test_data.next_batch(batch_size)
191                 test_acc_val = sess.run(
192                     [accuracy],
193                     feed_dict={
194                         x: test_batch_data,
195                         y: test_batch_labels
196                     })
197                 all_test_acc_val.append(test_acc_val)
198             test_acc = np.mean(all_test_acc_val)
199             print('[Test] Step: %d, acc: %4.5f ' % (i+1, test_acc))
200

```

卷积神经网络进阶

不同网络结构

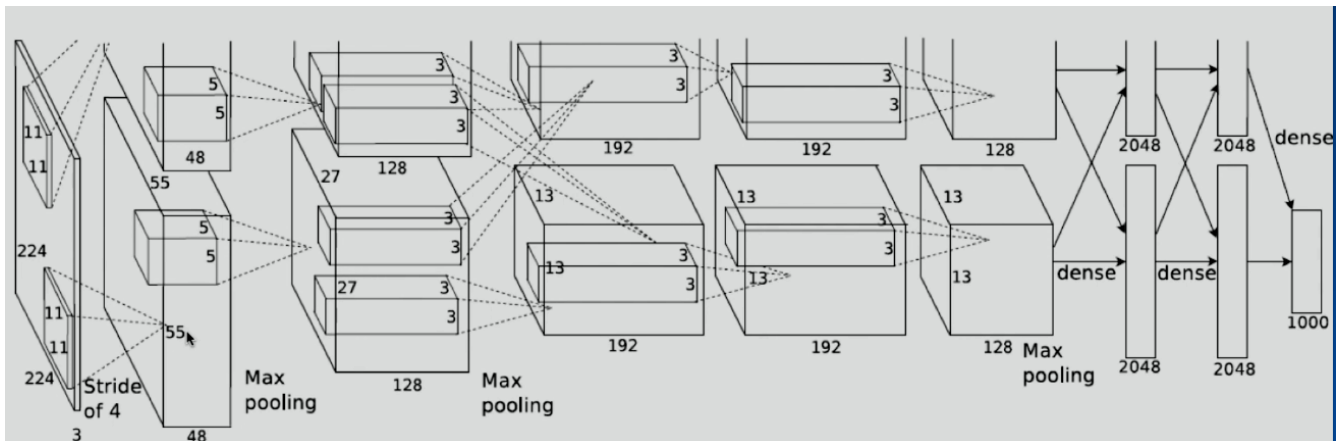
- 不同的网络结构解决的问题不同
- 不同的网络结构使用的技巧不同
- 不同的网络结构应用的场景不同

模型的进化

- 更深更宽 - AlexNet到VGGNet
- 不同的模型结构 - VGG到Inception/ResNet
- 优势组合-Inception+Res = InceptionResNet
- 自我学习-NASNet
- 实用-MobileNet

AlexNet

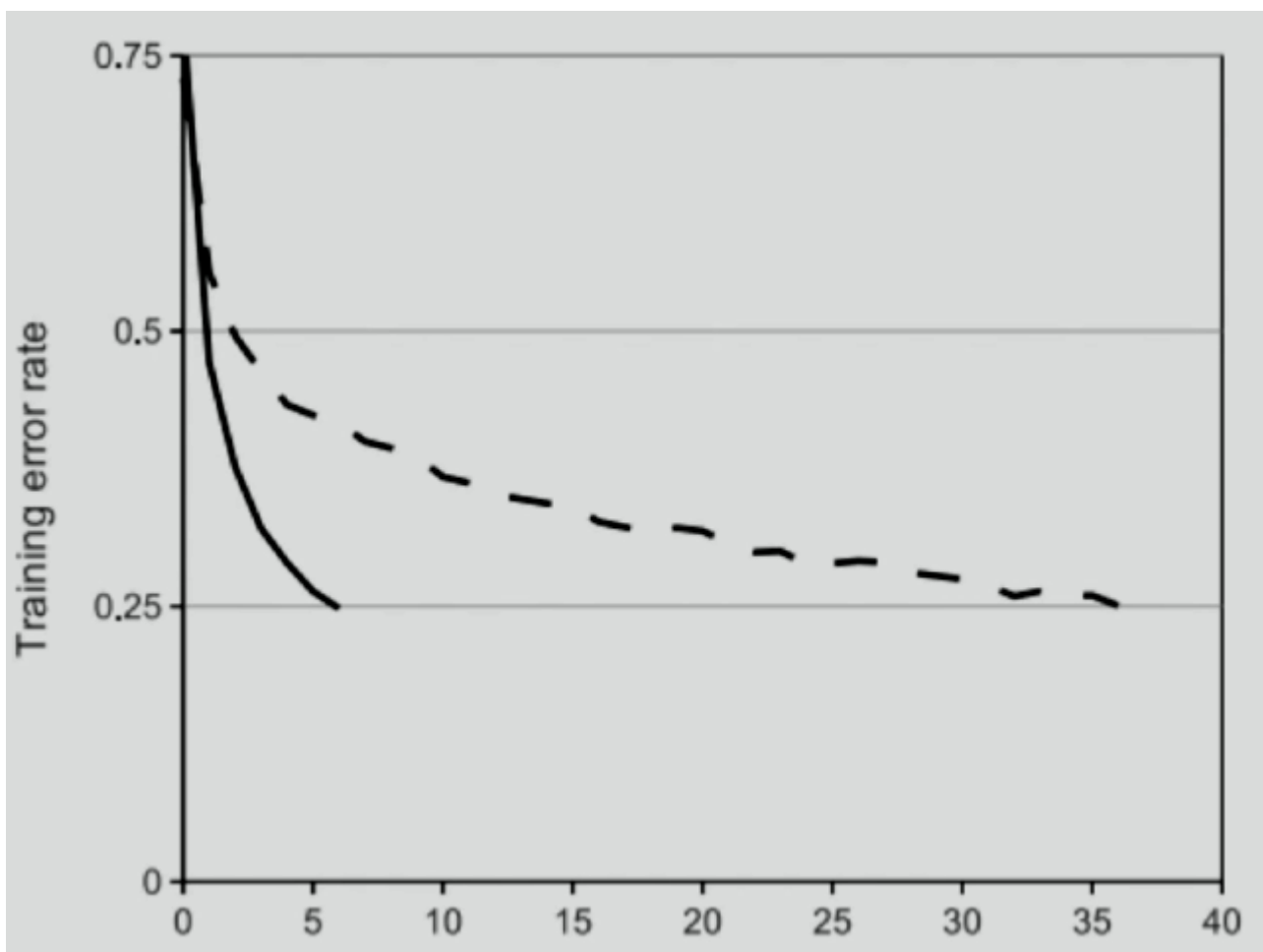
网络结构



第一个卷积层

- 输入224*224
- Stride = 4,卷积核 11*11
- 输出大小=(输出大小-卷积核+padding)/stride + 1 = 55
- 参数数目 = 3(1111)*96 = 35K

首次使用Relu



特点

- 2-GPU并行结构
- 1, 2, 5卷积层后跟随max-pooling层
- 两个全连接层上使用了dropout技术
 - dropout: 计算当前层某个神经元时, 随机把上层某个神经元输出的值屏蔽为0
 - 全连接层参数占全部参数数目的大部分, 容易过拟合
- Batch size = 128
- SGD momentum = 0.9
- Learning rate = 0.01, 过一定次数后下降为原来的1/10
- 7个CNN做ensemble: 18.2%→15.4%

VGGNET

网络结构

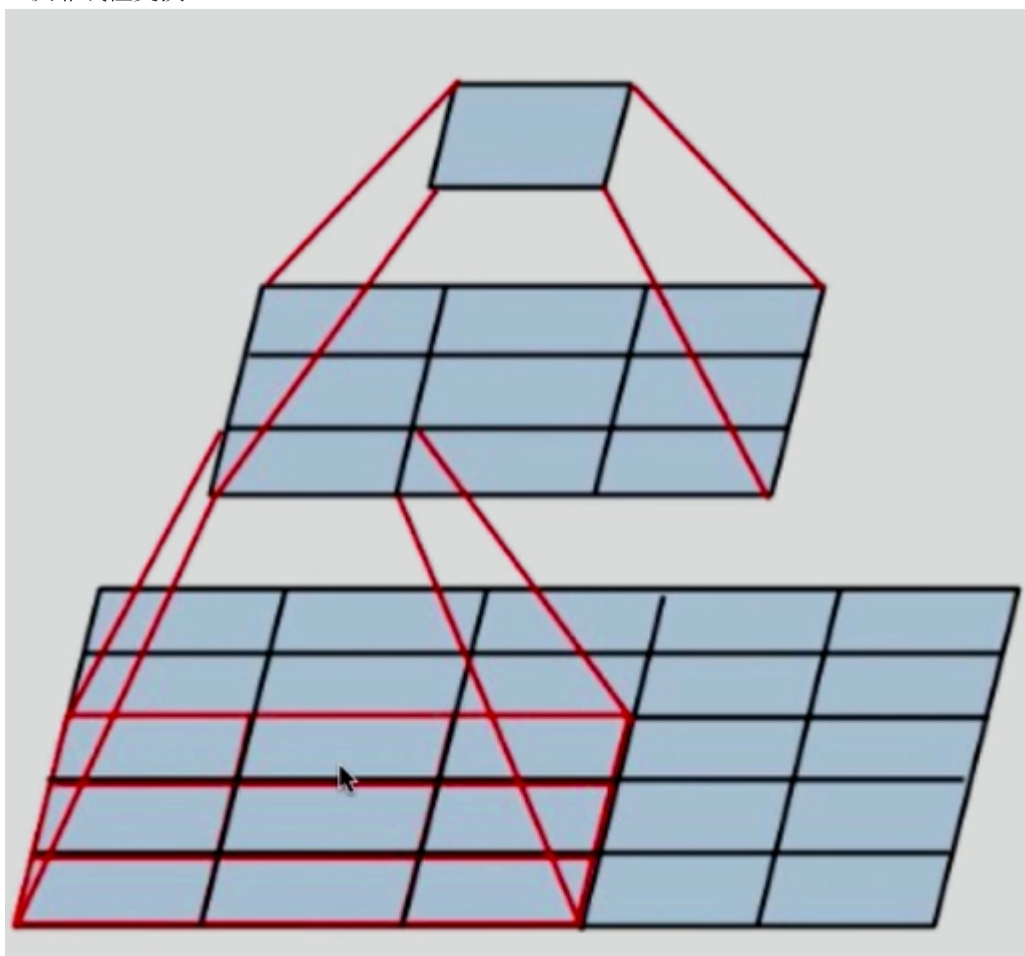
- 更深
- 多使用3x3的卷积核
 - 2个3x3的卷积层可以看做一层5x5的卷积层
 - 3个3x3的卷积层可以看做一层7x7的卷积层
- 1x1的卷积层可以看做是非线性变化

- 每次经过一个pooling层，通道数目翻倍

视野域

- 视野域：2个 $3 \times 3 = 1$ 个 5×5
- 2层比1层更多一次非线性变换

- 参数降低28%

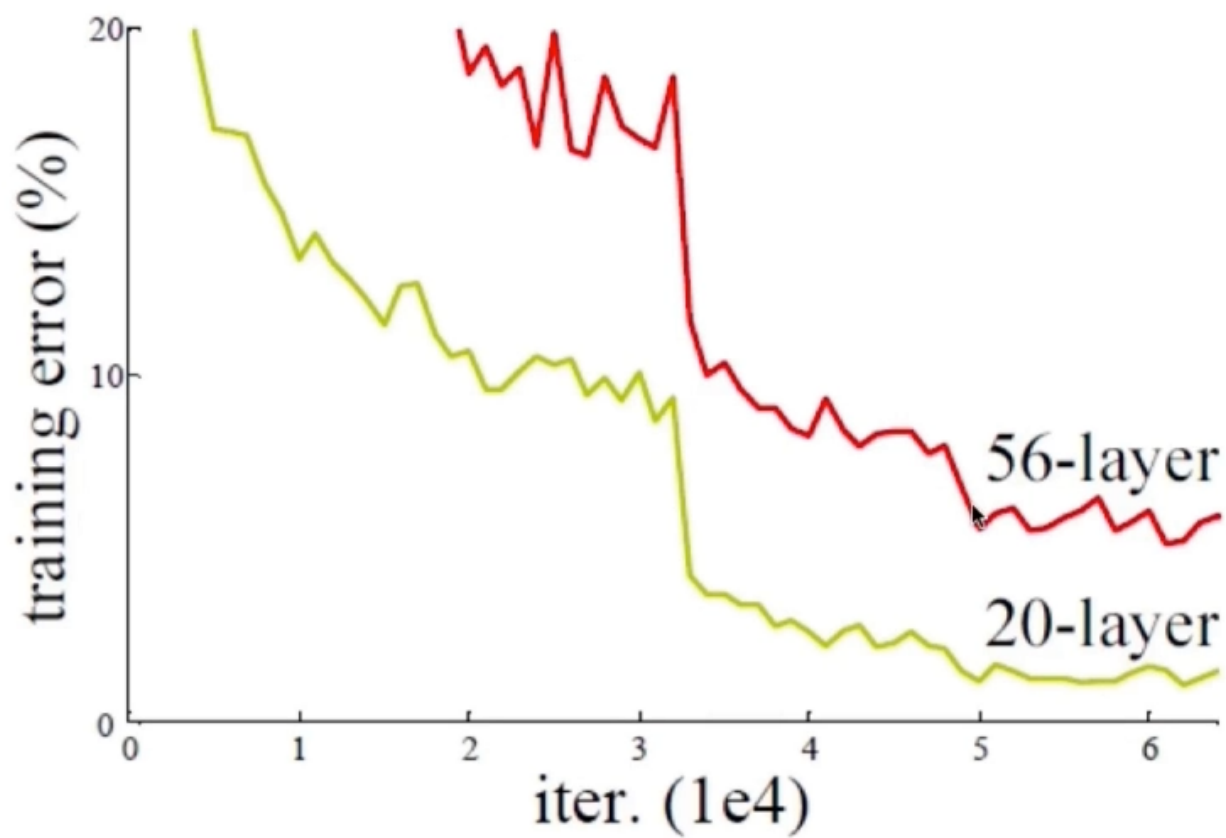


技巧

- 先训练浅层网络，再利用浅层网络参数去训练深层网络
- 多尺度输入
 - 不同的尺度训练多个分类器，然后做ensemble
 - 随机使用不同的Feature Scaling然后输入进分类器进行训练

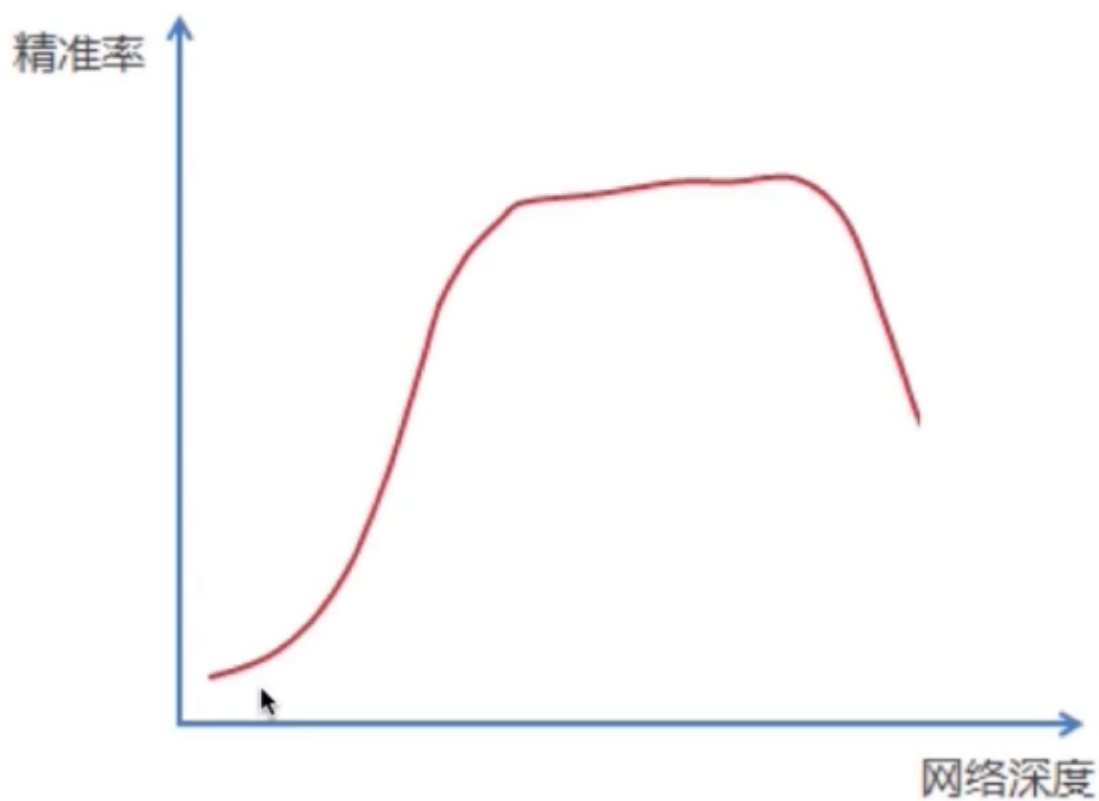
Resnet

问题抛出 :模型深度达到某个程度后继续加深会导致训练集准确率下降



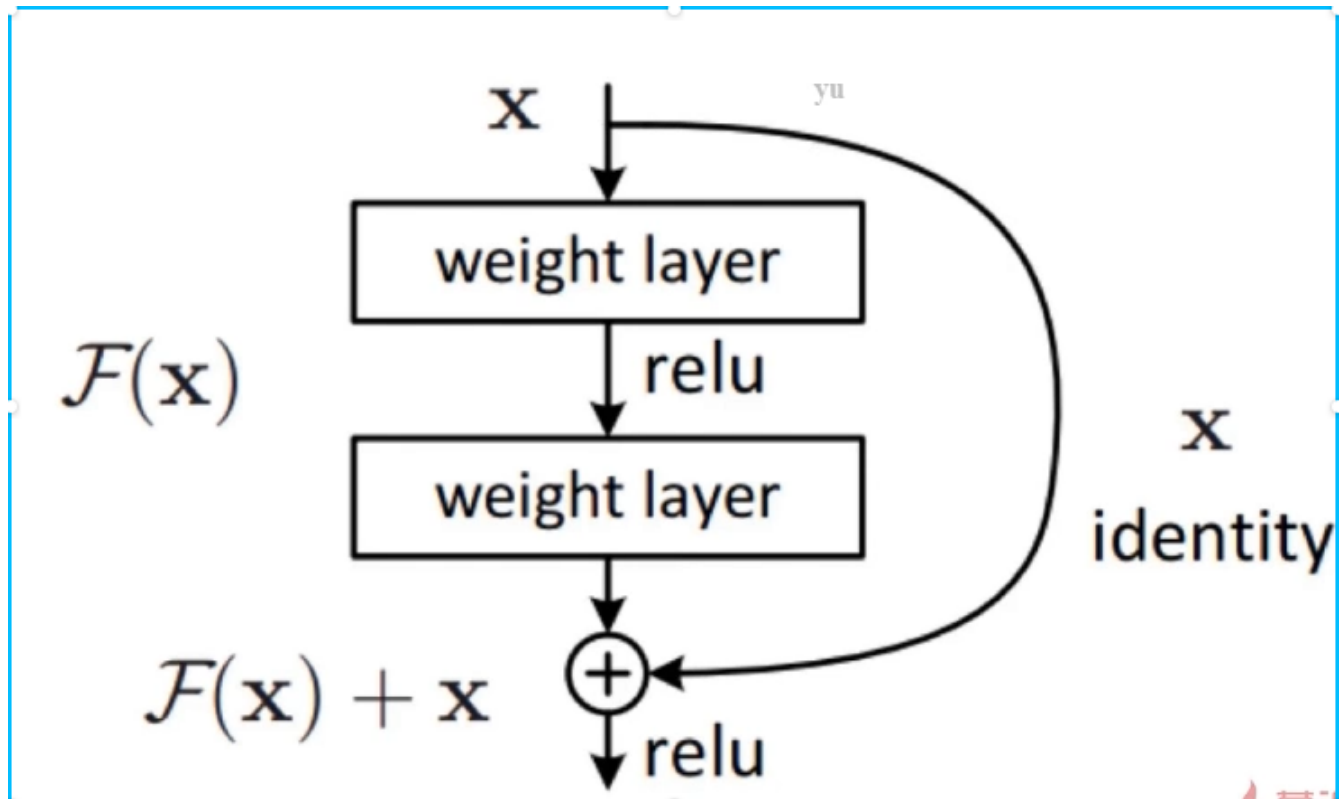
训练集误差图

慕课网



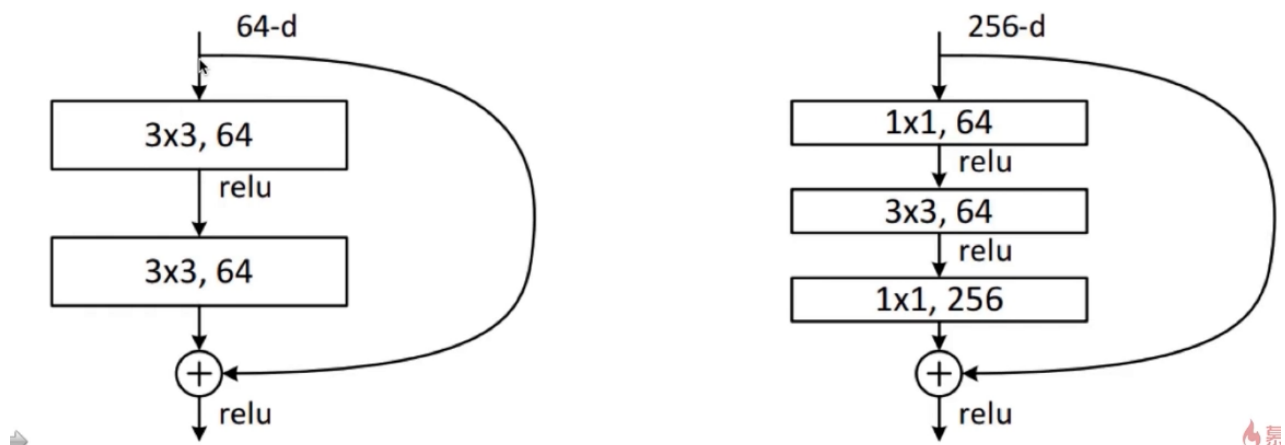
加深层次的问题解决

- 假设：深层网络更难优化而非深层网络学不到东西
 - 深层网络至少可以和浅层网络持平
 - $y=x$, 虽然增加了深度，但是误差不会增加 因此，ResNet采用了这样一种结构
- Identity部分是恒等变换
- $F(x)$ 是残差学习



模型结构

- Resnet-34 与 Resnet-101使用的子结构



- 先使用一个普通的卷积层，stride = 2
- 再经过一个3x3的max_pooling
- 再经过残差结构
- 没有中间的全连接层，直接到输出
- 残差结构使得网络需要学习的知识变少，容易学习

- 残差结构使得每一层的数据分布接近，容易学习