西北工業大學

Python 机器学习

及大数据

题 目 应用 tensorflow 实现简单的机器学习

学生姓名 LiAsNH3

学生学号 xxxxxxxx

学院名称 xxxxxxxx

专业名称 xxxxxxxx

说　明

随着机器学习的发展，诞生了众多的机器学习（深度学习）框架，如 TensorFlow、PyTorch、Caffe 等，本文选择了社区较为丰富的 TensorFlow 架构来实现机器学习领域一些简单的算法。

计算平台

本文所有的算法均用 Python 实现，其版本为 3.5.6；计算机性能如下：

- 处理器：Intel(R)Core(TM)i7-7700 CPU @3.60GHZ
- 内存大小：8G
- 操作系统：Windows10 家庭版
- GPU：计算能力 3.5，GeForce GT 730

数据集

本文所使用的训练图像集来源于网上，共两类，猫有，狗有测试图像集为百度图片里的，共采集了，其中猫，狗

使用到的 Python 工具包

下面列出用到的 Python 工具包以及他们的功能：

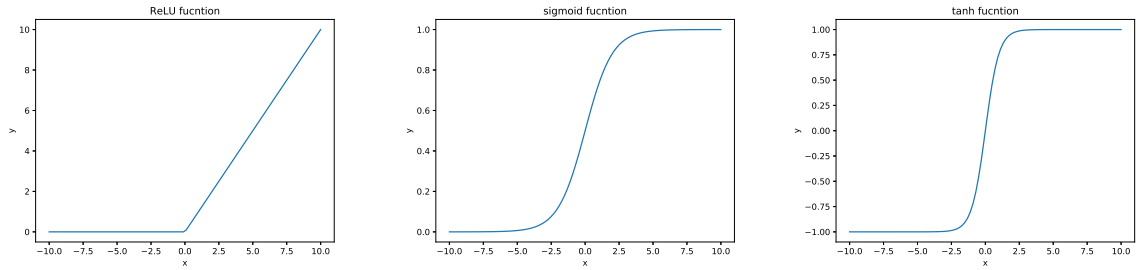
- tensorflow-gpu：机器学习架构
- matplotlib：绘图相关
- numpy：数值计算相关（主要是矩阵相关的操作）
- PIL：图像处理相关
- random：产生随机数，返回类型为 numpy 中的数组
- requests：网络爬虫相关
- 以及上述的依赖包（具体见提供的 Anaconda 包列表文件 TensorFlow-env.txt）

目 录

一、小试牛刀 — tensorflow 的简单尝试

1.1 应用机器学习的方法实现数据拟合

数据拟合是一种在数据处理中常用的方法，它能够反映数据的变化趋势，对于线性拟合的研究已经比较多了，有众多的实现方法；而对于非线性的拟合较为复杂，下面考虑一个二维区域上的非线性数据拟合，通过机器学习的方法来实现。由于所考虑的问题为非线性的，因此需要考虑应用特殊激活函数来实现去线性化，在图中给出了几种常用的非线性激活函数的函数图像。从图1-1上可以看出，这



(a) ReLU 函数 $f(x) = max(x, 0)$　(b) sigmoind 函数 $f(x) = \frac{1}{1+e^{-x}}$　(c) tanh 函数 $f(x) = \frac{1-e^{-2x}}{1+e^{-2x}}$

图 1-1: 常用的神经网络非线性激活函数

些激活函数都不是一条直线，所以通过这些激活函数，每一个节点都不是线性变换，于是整个神经网络也不是线性的了。为简单起见，此处选取激活函数 ReLU，构造了一个只有一层隐藏层的简单神经网络。

1.1.1 数据点集

通过产生随机数，在二维区域得到了一些带有噪声的数据点，如图1-2所示。事实上，这些数据是

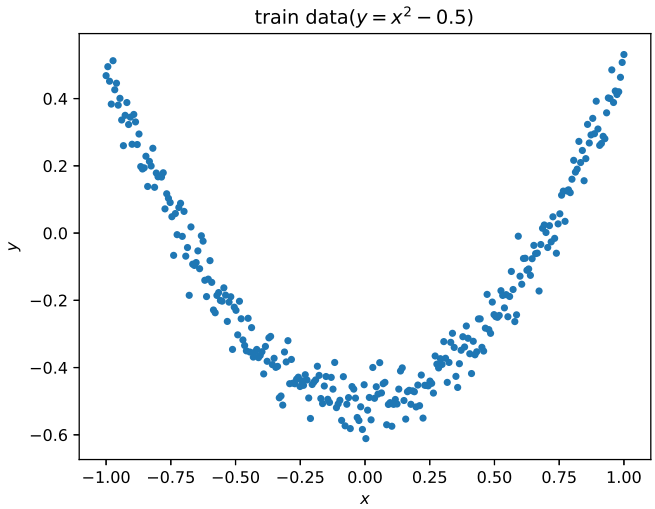


图 1-2: 需要拟合的二维数据

由 $y = x^2 - 0.5 + noise$ 得到的。接下来的任务就是通过机器学习得到这些数据点的最好近似。

1.1.2 tensorflow 实现

下面给出 tensorflow 实现机器学习的步骤（根据具体的代码来说明）：

- 应用 tensorflow 实现数据拟合首先需要导入 Python 的工具包[1]

```
'''小试牛刀
应用机器学习的方法来简单拟合数据
'''
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
```

- 训练数据准备

  此处给定一个非线性函数，随机在图像上取点并加上一个噪声项，此处还通过 matplotlib 中 scatter 函数将其绘制出来，结果见图1-2。

```
# 生成模拟训练数据
x_data = np.linspace(-1, 1, 300)[:, np.newaxis]
noise = np.random.normal(0, 0.05, x_data.shape)
y_data = np.square(x_data) - 0.5 + noise

plt.figure()  # 绘制生产的训练点集
plt.scatter(x_data, y_data, s=12.0)
plt.title("train data($y=x^2-0.5$)")
plt.xlabel("$x$")
plt.ylabel("$y$")
plt.savefig("doc/result_figure/fiting_trian.eps", dpi=300, pad_inches=0)
plt.show()
```

- 设置变量

  通过 tensorflow 的内置函数 placeholder 定义输入和输出信息，事实上 tensorflow 通过 placeholder 对数据进行了封装，为后续训练数据导入模型提供了方便。

```
# 定义网络输入信息
xs = tf.placeholder(tf.float32, [None, 1])
ys = tf.placeholder(tf.float32, [None, 1])
```

- 创建损失函数和计算模型

  定义整个神经网络的各层设置，首先自定义了一个网络层函数

```
def add_layer(inputs, in_size, out_size, activation_function=None):
    """定义神经网络中的网络层
    将每层的信息定义好，后续直接调用

    Args:
```

[1]代码见文件 small_try.py

```
33          inputs: 输入层的数据或者是前一层网络的输出
34          in_size: 与权重相关的参数，与前一层的输出确定
35          out_size: 与权重相关的参数，与当前层的输出相关
36          activation_function: 激活函数，默认没有
37
38
39      Returns:
40          返回一个tensorflow张量
41      """
42      Weights = tf.Variable(tf.random_normal([in_size, out_size]))  # 隐藏层的权重
43      biases = tf.Variable(tf.zeros([1, out_size]) + 0.1)  # 隐藏层的偏移量
44      Wx_plus_b = tf.matmul(inputs, Weights) + biases  # 向前传播计算
45      if activation_function is None:  # 设置激活函数，默认没有
46          outputs = Wx_plus_b
47      else:
48          outputs = activation_function(Wx_plus_b)
49      return outputs
```

接下来利用网络层函数定义输入层、隐藏层、输出层，并且隐藏层选用的激活函数为 ReLU 激活函数。

```
52  # 添加隐藏层
53  l1 = add_layer(xs, 1, 10, activation_function=tf.nn.relu)
54  # 添加输出层
55  prediction = add_layer(l1, 10, 1, activation_function=None)
```

损失函数是刻画当前预测值与真实答案之间的差距，在机器学习中一般通过反向传播算法来调整神经网络参数的取值使得差距缩小。下面通过预测值与真实值之间的平均欧式距离来定义的损失函数，定义了学习虑为 0.1 的以梯度下降法为优化方法反向传播。

```
57  # 计算真值与预层值之间的误差，设置损失函数
58  loss = tf.reduce_mean(tf.reduce_sum(tf.square(ys-prediction),
59                        reduction_indices=[1]))
60  train_step = tf.train.GradientDescentOptimizer(0.1).minimize(loss)
```

- 将数据导入模型，创建会话，训练模型
  此处所考虑的问题较为简单，并且是空间二维的，所以通过 matplotlib 包中的绘图函数实现了模型在训练过程中预测值随着迭代进行的动态变化情况。

```
62  # 启动tensorflow会话
63  sess = tf.Session()
64  init = tf.global_variables_initializer()
65  sess.run(init)
66
67  fig = plt.figure()
68  ax = fig.add_subplot(1, 1, 1)
69  ax.scatter(x_data, y_data, label="trian data", s=12.0)  # 画出真实数据的点状图
```

```
70  ax.set_xlabel("$x$")
71  ax.set_ylabel("$y$")
72  plt.ion()
73  # 如果不加这个，每画完一条线程序会暂停，加了后会一直画，如果一直画不是会很
74  # 密密麻麻看不清吗，后面 有语句会移除当前线条防止出现该情况
75  plt.show()
76
77
78  for i in range(1000):
79      # training
80      sess.run(train_step, feed_dict={xs: x_data, ys: y_data})
81      if i % 10 == 0:
82          # 改进绘画的效果
83          try:  # 尝试以下语句
84              ax.lines.remove(lines[0])  # 抹除画出来的第一条线
85          except Exception:
86              # 如果没有的话，忽略第一次的错误，因为这时还没画第一条线
87              pass
88          prediction_value = sess.run(prediction, feed_dict={xs: x_data})  # 预测值
89          # plot the prediction
90          lines = ax.plot(x_data, prediction_value, '-.', lw=3, c='orange', label="
                                                        prediction line")
91          ax.set_title("Prediction function and test data iterated {:d} times".format(
                                                        i))
92          # 画出预测线，r-表红色，lw表粗度为5
93          plt.pause(1)  # 每画完一条线暂停一秒
```

- 结果处理（可视化等）

  将迭代最终的预测结果、真实结果、训练数据绘制在一张图中，可视化训练结果。

```
95  y_data_real = np.square(x_data)-0.5
96  ax.plot(x_data, y_data_real, "r-", label="real function")
97  ax.legend()
98  ax.set_ylim(-0.7, 0.8)
99  plt.savefig("doc/result_figure/fiting_result.eps", dpi=300, pad_inches=0)
```

1.1.3 结果相关

运行上述程序，得到训练的预测结果、真实函数形状、训练点三者的图示，见图1-3。从图上来看，使用测试数据训练得到的结果可以很好近似原函形状。但实际上，机器学习方法很难显示的得到一个显示表达式，只能将其看做一个黑箱，输入一个数据，得到一个结果，数据与结果之间的这种抽象关系近似满足原函数。

1.2 应用机器学习方法实现二维区域上的简单数据二分类

下面来考虑机器学习中经典问题，对二区域上的点集进行分类，为简单起见这儿仅考虑对数据进行二分类。考虑单位正方形上的随机点，并且通过直线 $x+y-1=0$ 将其分为两类，应用 tensorflow

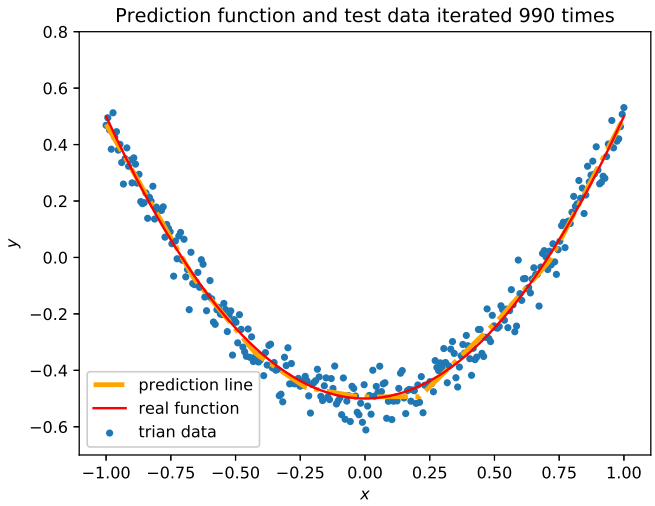


图 1-3: 需要拟合的二维数据

编程实现。此处所考虑的神经网络有一层输入层，一层隐藏层，一层输出层，并且考虑了偏置项[2]。

1.2.1 tensorflow 实现

- 导入必须的 Python 工具包 tensorflow、matplotlib、numpy

```
"""小试牛刀
利用tensorflow对二维区域上的数据进行二分类
"""
import tensorflow as tf
from numpy.random import RandomState
import matplotlib.pyplot as plt
import numpy as np
```

- 定义模型相关的参数权重、偏置项，以及输入、输出量

```
# 定义参数和输入节点
w = tf.Variable(tf.random_normal([2, 1], stddev=1, seed=1))
b = tf.Variable(tf.random_normal([1], stddev=1, seed=1))

x = tf.placeholder(tf.float32, shape=(None, 2), name="x-input")
y_ = tf.placeholder(tf.float32, shape=(None, 1), name='y-input')
```

- 定义前向传播过程及其损失函数

```
y = tf.nn.sigmoid(tf.matmul(x, w) + b)
cross_entropy = -tf.reduce_mean(y_ * tf.log(tf.clip_by_value(y, 1e-10, 1.0))
                                + (1 - y_) * tf.log(tf.clip_by_value(1 - y, 1e-10, 1
                                                                     .0)))
train_step = tf.train.AdamOptimizer(0.001).minimize(cross_entropy)
```

[2]代码实现见文件 sample_tensorflow.py

使用 sigmoid 函数将隐藏层输出转化为 0 到 1 之间的数值，转化后 y 代表预层是正样本的概率，$1-y$ 代表预层是负样本的概率。在上述代码中 cross_entropy 定义了真实值与预测值之间的交叉熵，这是分类中常用的损失函数；为了保证结果的合理性，防止出现计算机无法表示的小数，将 sigmoid 函数转化的结果限制到 10^{-10}~1 之间。train_step 定义了反向传播算法[3]。

- 生成模拟训练数据集和测试数据集

```
24 # 生成模拟训练数据
25 rdm = RandomState(1)
26 X = rdm.rand(256, 2)  # 模拟数据集
27 Y = [[int(x1+x2 < 1)] for (x1, x2) in X]  # 类标
28
29
30 # 生成模拟测试数据
31 rdm = RandomState(2)
32 X_test = rdm.rand(100, 2)
33 Y_test = [[int(x1+x2 < 1)] for (x1, x2) in X_test]  # 类标
34 test_feed = {x: X_test, y_: Y_test}
```

- 定义正确率计算

```
37 # 定义正确率计算
38 correct_prediction = tf.equal(tf.argmax(y, 1), tf.argmax(y_, 1))
39 accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
```

通过判断预测值与真实值之间是否相等，再统计逻辑真的结果即可得到预测正确的数目。

- 创建 tensorflow 会话，训练模型，可视化结果

```
42 # 创建一个tenor flow会话，训练结果
43 with tf.Session() as sess:
44     init_op = tf.global_variables_initializer()
45     sess.run(init_op)
46
47     # 训练模型
48     STEPS = 500
49     for i in range(STEPS):
50         # 从测试集中选取适当数量的样本进行训练
51         for input_x, input_y in zip(X, Y):
52             input_x = np.reshape(input_x, (1, 2))
53             input_y = np.reshape(input_y, (1, 1))
54             sess.run(train_step, feed_dict={x: input_x, y_: input_y})
55         if i % 100 == 0:
56             total_cross_entropy = sess.run(cross_entropy, feed_dict={x: X, y_: Y})
57             print("After %d training step(s), cross entropy on all data is %g" % (i,
                                                                    total_cross_entropy))
```

[3]tensorflow 中目前常用的优化方法有三种：tf.train.GradientDescentOptimizer、tf.train.AdamOptimizer 和 tf.train.MomentumOptimizer

```
58
59      # 输出训练后的参数取值。
60      print("\n")
61      print(sess.run(w))
62      print("\n"+"---"*50)
63
64      # 绘制测试数据
65      plt.figure()
66      t = np.linspace(0, 1, 100)
67      lin = -t + 1
68      plt.plot(t, lin, ls="-", c="r", label="classification line")
69
70      # 下面开始进行测试
71      pred_Y = sess.run(y, feed_dict={x: X_test})
72      pred_A = []
73      pred_B = []
74      index = np.zeros((100, 1), np.int)
75      for i in range(100):
76          if pred_Y[i] > 0.5:
77              pred_A.append(i)
78          else:
79              pred_B.append(i)
80      plt.scatter(X_test[pred_A, 0], X_test[pred_A, 1], c="b", label="predict A")
81      plt.scatter(X_test[pred_B, 0], X_test[pred_B, 1], c="g", label="predict B")
82      plt.legend()
83      plt.xlabel("x")
84      plt.ylabel("y")
85      plt.title("test data classification results")
86      plt.savefig("doc/result_figure/sample_class.eps", dpi=300)
87      plt.show()
```

1.2.2 结果相关

运行上述程序可以得到测试数据的分类结果，如图1-4所示。

二、大展身手 — 应用 tensorflow 实现对猫狗图集的分类

2.1 数据集说明

本文所用的猫狗数据集来源于 CSDN 社区，共有训练图像 25000 张，其中猫 12500 张，狗 12500 张，其部分图像如图2-1。

2.2 卷积神经网络简介

卷积神经网络（Convolutional Neural Network, CNN）是一种前馈神经网络，它的人工神经元可以响应一部分覆盖范围内的周围单元，对于大型图像处理有出色表现。卷积神经网络由一个或多个卷积层和顶端的全连通层（对应经典的神经网络）组成，同时也包括关联权重和池化层。这一结构使得卷积神经网络能够利用输入数据的二维结构。与其他深度学习结构相比，卷积神经网络在图像和语音识别方面能够给出更好的结果。这一模型也可以使用反向传播算法进行训练。相比较其他深度、前馈神

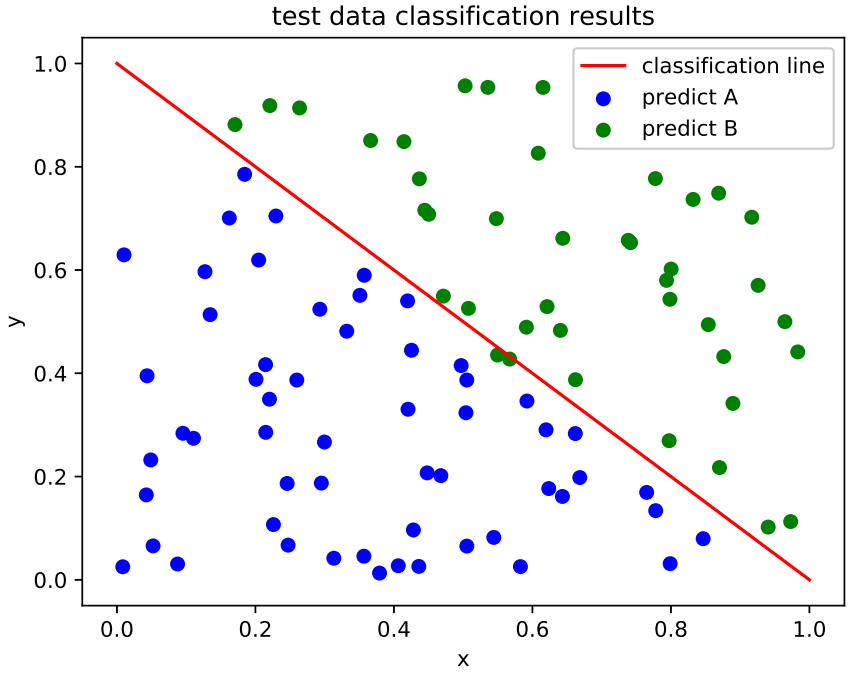


图 1-4: 简单二分类结果

经网络，卷积神经网络需要考量的参数更少，使之成为一种颇具吸引力的深度学习结构。常用的卷积神经网络架构，由输入层、卷积层 1、池化层 1、卷积层 2、池化层 2、全连接层 1、全连接层 2、softmax 层、输出层这样几层组成。这种结构模型称为 LeNet-5 模型。

2.3 tensorflow 实现卷积神经网络结构

本文考虑应用 LeNet-5 模型实现对猫狗图集的二分类。按上述模型的分层，逐层进行解释。

- 输入层

  输入层是整个神经网络的输入，在处理图像的卷积网格中，输入层一般输入的为矩阵，在本文所考虑的图像为彩色的，考虑到通道，输入的为 4 维矩阵 $208 \times 208 \times 3$。

- 卷积层 1

  这一层输入的为原始的图像像素，按批次传入，本文每批次传入 16 张图像，卷积层过滤器的尺寸为 3x3，深度为 16，步长为 1，共有 16 个偏置项。

```
7  def inference(images, batch_size, n_classes):
8
9      '''Build the model: 2conv  2pooling  2fc  1softmax
10     Args:
11         images: image batch, 4D tensor, tf.float32, [batch_size, width, height,
                                                        channels]
12     Returns:
13         output tensor with the computed logits, float, [batch_size, n_classes]
14     '''
15
```

图 2-1: 训练数据集数据样例

```
16      #conv1, shape = [kernel size, kernel size, channels, kernel numbers]
17      with tf.variable_scope('conv1') as scope:
18          weights = tf.get_variable('weights',
19                                    shape = [3,3,3,16],
20                                    dtype = tf.float32,
21                                    initializer=tf.contrib.layers.xavier_initializer()
                                                                                        )
22
23          biases = tf.get_variable('biases',
24                                   shape=[16],
25                                   dtype=tf.float32,
26                                   initializer=tf.constant_initializer(0.1))
27
28          conv = tf.nn.conv2d(images, weights, strides=[1,1,1,1], padding='SAME')
29          pre_activation = tf.nn.bias_add(conv, biases)
30          conv1 = tf.nn.relu(pre_activation, name= scope.name)
```

- 池化层 1

  这一层是卷积层 1 的输出，是一个 $208 \times 208 \times 16$ 本层采用的过滤器大小为 3×3，长和宽的步长均为 2

```
32      #pool1
33      with tf.variable_scope('pooling1') as scope:
34          pool1 = tf.nn.max_pool(conv1, ksize=[1,3,3,1],strides=[1,2,2,1],
35                                 padding='SAME', name='pooling1')
```

- 卷积层 2

  本层的输入矩阵大小为 $104 \times 104 \times 16$，使用的过滤器大小为 3×3，深度为 16。

```
37      #conv2
38      with tf.variable_scope('conv2') as scope:
39          weights = tf.get_variable('weights',
40                                    shape=[3,3,16,16],
41                                    dtype=tf.float32,
```

```
42                                  initializer=tf.contrib.layers.xavier_initializer()
                                                                                    )
43
44          biases = tf.get_variable('biases',
45                                   shape=[16],
46                                   dtype=tf.float32,
47                                   initializer=tf.constant_initializer(0.1))
48          conv = tf.nn.conv2d(pool1, weights, strides=[1,1,1,1],padding='SAME')
49          pre_activation = tf.nn.bias_add(conv, biases)
50          conv2 = tf.nn.relu(pre_activation, name='conv2')
```

- 池化层 2

  本层的输入数据大小为 $104 \times 104 \times 16$，采用过滤器大小为 3×3，步长为 1。

```
52      #pool2
53      with tf.variable_scope('pooling2') as scope:
54          pool2 = tf.nn.max_pool(conv2, ksize=[1,3,3,1], strides=[1,1,1,1],
55                                 padding='SAME',name='pooling2')
```

- 全连接层 1

  本层输入矩阵大小为 $104 \times 104 \times 16$。

```
57      #fc3
58      with tf.variable_scope('fc3') as scope:
59          reshape = tf.reshape(pool2, shape=[batch_size, -1])
60          dim = reshape.get_shape()[1].value
61          weights = tf.get_variable('weights',
62                                    shape=[dim,128],
63                                    dtype=tf.float32,
64                                    initializer=tf.contrib.layers.xavier_initializer()
                                                                                    )
65          biases = tf.get_variable('biases',
66                                   shape=[128],
67                                   dtype=tf.float32,
68                                   initializer=tf.constant_initializer(0.1))
69          fc3 = tf.nn.relu(tf.matmul(reshape, weights) + biases, name=scope.name)
```

- 全连接层 2

  本层的输入节点个数为 128，输出节点个数为 128。

```
71      #fc4
72      with tf.variable_scope('fc4') as scope:
73          weights = tf.get_variable('weights',
74                                    shape=[128,128],
75                                    dtype=tf.float32,
76                                    initializer=tf.contrib.layers.xavier_initializer()
                                                                                    )
```

```
77
78          biases = tf.get_variable('biases',
79                                   shape=[128],
80                                   dtype=tf.float32,
81                                   initializer=tf.constant_initializer(0.1))
82          fc4 = tf.nn.relu(tf.matmul(fc3, weights) + biases, name='fc4')
```

- softmax 层

  softmax 层主要用于分类问题，运算得到术语不同种类的概率分布情况。

```
84      # softmax
85      with tf.variable_scope('softmax_linear') as scope:
86          weights = tf.get_variable('softmax_linear',
87                                    shape=[128, n_classes],
88                                    dtype=tf.float32,
89                                    initializer=tf.contrib.layers.xavier_initializer()
                                                                                        )
90          biases = tf.get_variable('biases',
91                                   shape=[n_classes],
92                                   dtype=tf.float32,
93                                   initializer=tf.constant_initializer(0.1))
94          softmax_linear = tf.add(tf.matmul(fc4, weights), biases, name='
                                                              softmax_linear')
95
96      return softmax_linear
```

2.4 tensorflow 实现

由于训练数据较多，共有 25000 张图片，不能直接将所有数据代入模型中进行训练，需要分批次代入，编写了一个批次获取函数，循环迭代，训练模型。训练完成后，从训练图片集中分别随机挑取 20 张猫和狗的图片，代入模型中得到预测结果，验证模型的正确性；然后尝试应用该模型取识别任意一张猫和狗的图片，应用网络爬虫技术从网上爬取了猫和狗的图片集，验证模型的通用性。tensorflow 实现的流程图见图2-5。通过 tensorflow 工具包自带的可视化工具 tensorboard，可以得到随着训练进行，正确率的变化趋势见图2-2。

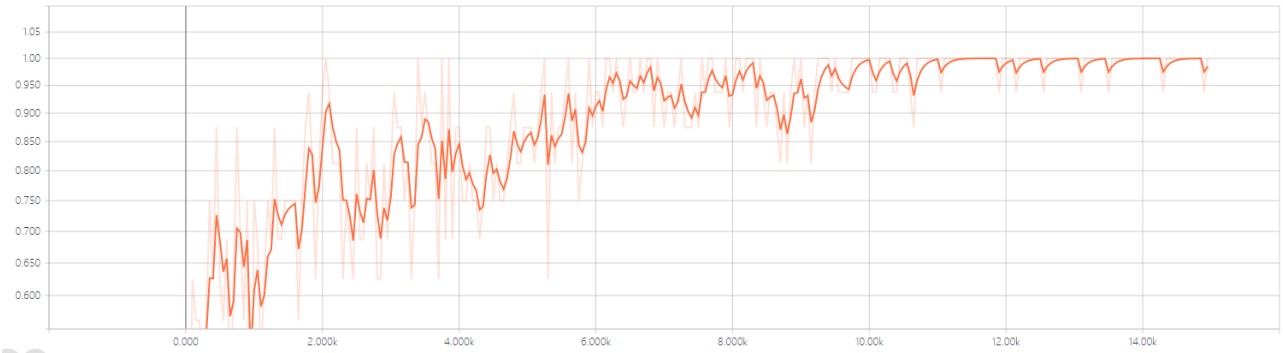


图 2-2: 随着训练中迭代进行模型识别准确率变化图

2.4.1 训练模型的测试

测试中从训练集中随机挑取了 61 张猫和狗的图片，代入模型，验证模型的准确性，结果见图2-3正确率为 100%。从结果上来看，该模型可很好的区分训练集中猫和狗。

```
C:\Users\lsa>cd c:\Users\lsa\Desktop\deep_learning\cats_and_dogs && cmd /C "set "PYTHONIOENCODING=UTF-8" && set "PYTHONUNBUFFERED=1" && D:\ProgramData\Anaconda3\envs\tensorflow_gpu\python.exe c:\Users\lsa\.vscode\extensions\ms-python.python-2019.6.24221\pythonFiles\ptvsd_launcher.py --default --client --host localhost --port 1224 c:\Users\lsa\Desktop\deep_learning\cats_and_dogs\my_test.py "
----------------------------------------------------
There are 31 cats
There are 30 dogs
There are 61 test images totally.....
----------------------------------------------------
2019-07-13 08:29:00.349838: I tensorflow/core/platform/cpu_feature_guard.cc:141] Your CPU supports instructions that this TensorFlow binary was not compiled to use: AVX AVX2
2019-07-13 08:29:00.501098: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1405] Found device 0 with properties:
name: GeForce GT 730 major: 3 minor: 5 memoryClockRate(GHz): 0.9015
pciBusID: 0000:01:00.0
totalMemory: 2.00GiB freeMemory: 1.65GiB
2019-07-13 08:29:00.508649: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1484] Adding visible gpu devices: 0
2019-07-13 08:29:00.892523: I tensorflow/core/common_runtime/gpu/gpu_device.cc:965] Device interconnect StreamExecutor with strength 1 edge matrix:
2019-07-13 08:29:00.897054: I tensorflow/core/common_runtime/gpu/gpu_device.cc:971]      0
2019-07-13 08:29:00.899242: I tensorflow/core/common_runtime/gpu/gpu_device.cc:984] 0:   N
2019-07-13 08:29:00.902335: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1097] Created TensorFlow device (/job:localhost/replica:0/task:0/device:GPU:0 with 1421 MB memory) -> physical GPU (device: 0, name: GeForce GT 730, pci bus id: 0000:01:00.0, compute capability: 3.5)
Reading chckpoint.....
Loading success, global steps is 14999
----------------------------------------------------
The model 's loss is 0.02
Correct: 61
Wrong: 0
The accuracy in test images are 100.00%
```

图 2-3: 模型测试结果

2.4.2 训练数据的获取—网络爬虫简单实现

应用 python 工具包 requests 实现从百度图片中按猫或狗关键词抓取猫和狗的图片并保存到本地，收集好数据代入前模型中进行训练，结果见图2-4，一共抓取了 180 张, 猫狗各 90 张，正确率为 57.76%。从结果来看，通过训练集训练得到的模型不能较好识别网上获得的图片，主要原因是在网上这些图片中含有较多的干扰信息，如在一张图中既有猫又有狗，对分类造成影响。

感谢

首先感谢胡海蔚同学提供的卷积神经网络模型工具，这给我的编程提供了很大帮助；其次感谢 CSDN 上额参考过的那些博主们，他们所提供的的解决问题的方法帮助我解决了许多环境配置或编写程序中的许多问题。

```
C:\Users\lsa>cd c:\Users\lsa\Desktop\deep_learning\cats_and_dogs && cmd /C "set "PYTHONIOENCODING=UTF-8" && set "PYTHONUNBUFFERED=1" && D:\ProgramData\Anaconda3\envs\tensorflow_gpu\python.exe c:\Users\lsa\.vscode\extensions\ms-python.python-2019.6.24221\pythonFiles\ptvsd_launcher.py --default --client --host localhost --port 1687 c:\Users\lsa\Desktop\deep_learning\cats_and_dogs\my_test.py "
----------------------------------------------------
There are 90 cats
There are 90 dogs
There are 180 test images totally.....
----------------------------------------------------
2019-07-13 08:49:27.327076: I tensorflow/core/platform/cpu_feature_guard.cc:141] Your CPU supports instructions that this TensorFlow binary was not compiled to use: AVX AVX2
2019-07-13 08:49:27.466735: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1405] Found device 0 with properties:
name: GeForce GT 730 major: 3 minor: 5 memoryClockRate(GHz): 0.9015
pciBusID: 0000:01:00.0
totalMemory: 2.00GiB freeMemory: 1.65GiB
2019-07-13 08:49:27.473487: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1484] Adding visible gpu devices: 0
2019-07-13 08:49:27.940748: I tensorflow/core/common_runtime/gpu/gpu_device.cc:965] Device interconnect StreamExecutor with strength 1 edge matrix:
2019-07-13 08:49:27.946977: I tensorflow/core/common_runtime/gpu/gpu_device.cc:971]      0
2019-07-13 08:49:27.949603: I tensorflow/core/common_runtime/gpu/gpu_device.cc:984] 0:   N
2019-07-13 08:49:27.951889: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1097] Created TensorFlow device (/job:localhost/replica:0/task:0/device:GPU:0 with 1421 MB memory) -> physical GPU (device: 0, name: GeForce GT 730, pci bus id: 0000:01:00.0, compute capability: 3.5)
Reading chckpoint.....
Loading success, global steps is 14999
----------------------------------------------------
2019-07-13 08:49:30.077288: W tensorflow/core/common_runtime/bfc_allocator.cc:219] Allocator (GPU_0_bfc) ran out of memory trying to allocate 802.09MiB. The caller indicates that this is not a failure, but may mean that there could be performance gains if more memory were available.
2019-07-13 08:49:30.093833: W tensorflow/core/common_runtime/bfc_allocator.cc:219] Allocator (GPU_0_bfc) ran out of memory trying to allocate 1.42GiB. The caller indicates that this is not a failure, but may mean that there could be performance gains if more memory were available.
2019-07-13 08:49:30.711640: W tensorflow/core/common_runtime/bfc_allocator.cc:219] Allocator (GPU_0_bfc) ran out of memory trying to allocate 1.32GiB. The caller indicates that this is not a failure, but may mean that there could be performance gains if more memory were available.
2019-07-13 08:49:31.184212: W tensorflow/core/common_runtime/bfc_allocator.cc:219] Allocator (GPU_0_bfc) ran out of memory trying to allocate 1.04GiB. The caller indicates that this is not a failure, but may mean that there could be performance gains if more memory were available.
The model 's loss is 2.73
Correct: 103
Wrong: 77
The accuracy in test images are 57.76%
```

图 2-4: 网络爬虫数据分类结果

参 考 文 献

[1] 郑泽宇，梁博文等，TensorFlow：实战 Google 深度学习框架，电子工业出版社，2018.2.

[2] 刘大成，Python 数据可视化之 matplotlib 实践，电子工业出版社，2018.9.

[3] 张若愚，Python 科学计算（第二版），清华大学出版社，2016.

[4] CSDN 博客，https://blog.csdn.net/mbx8x9u/article/details/79124840

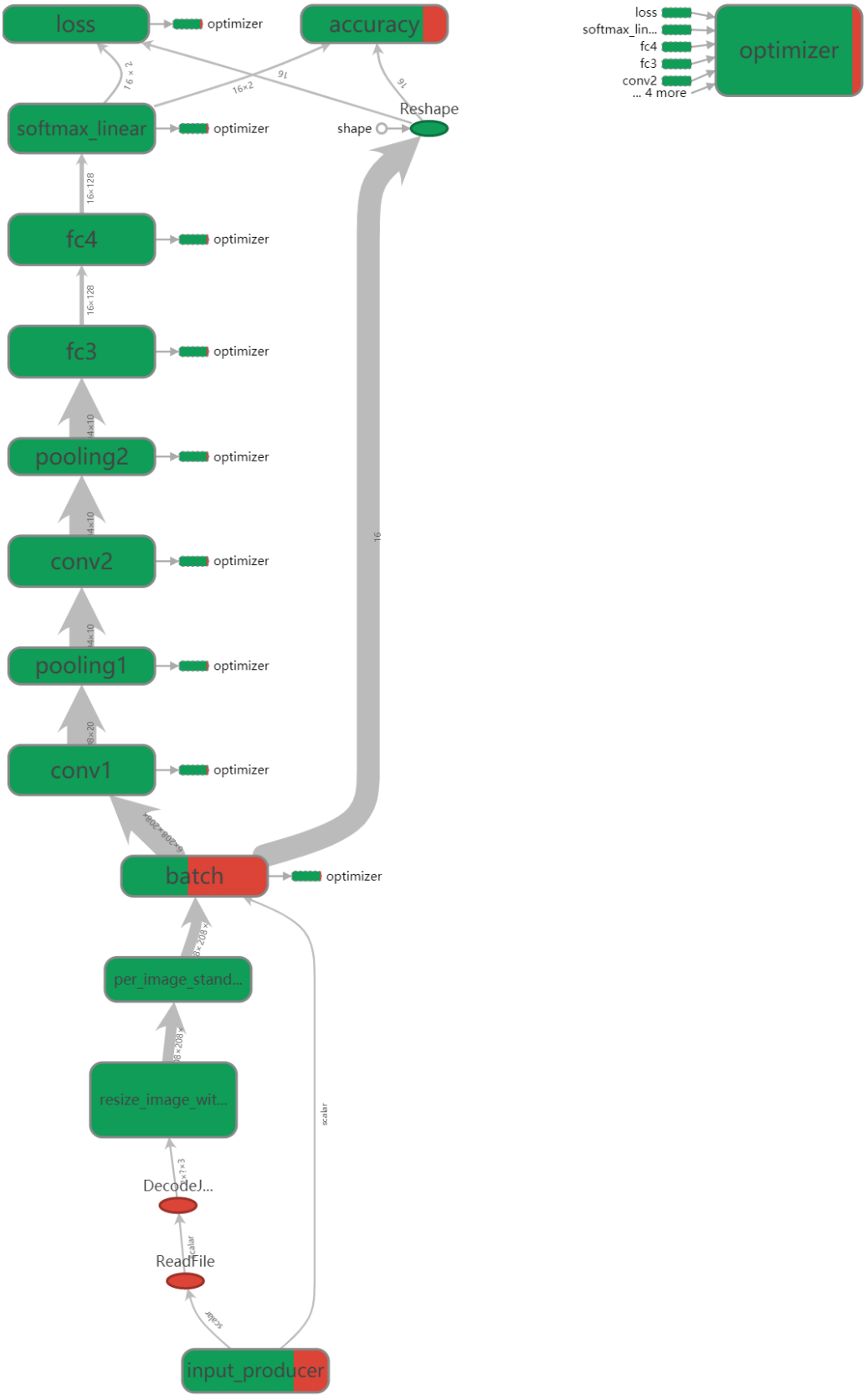


图 2-5: 猫狗图像集分类流程