WTF Daily Blog

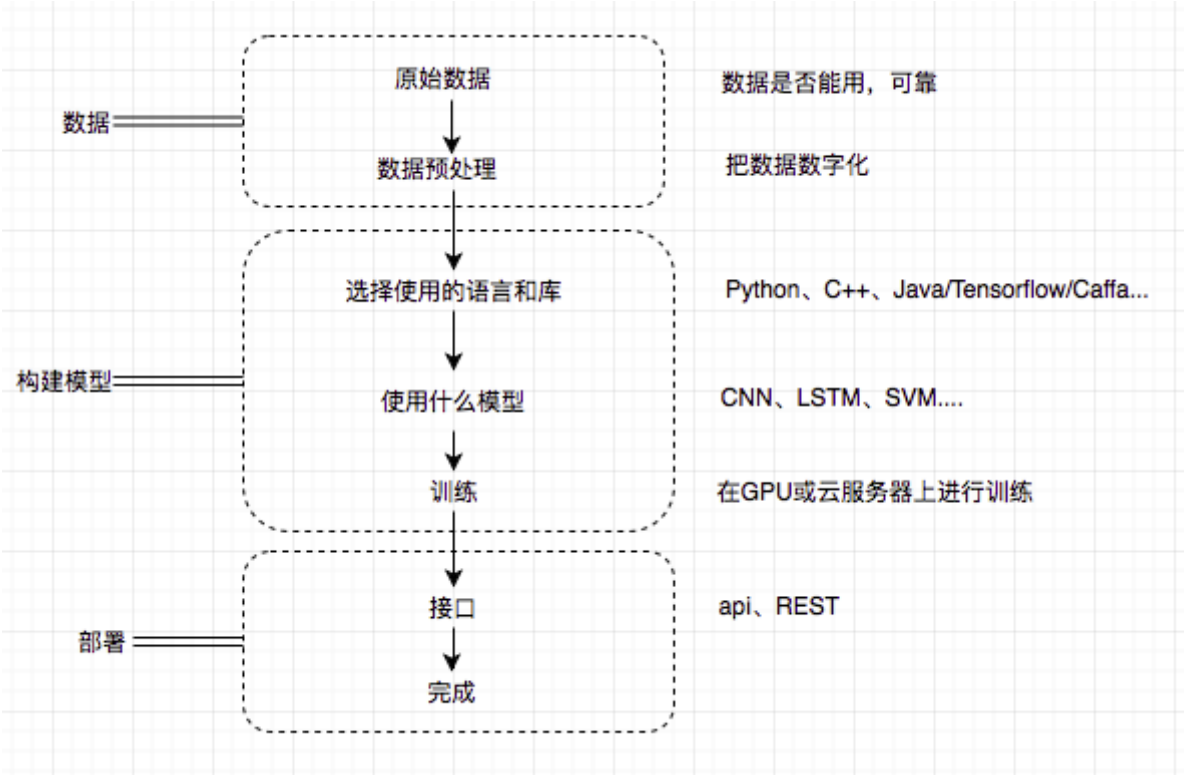
斗大的熊猫

TensorFlow练习2: 对评论进行分类

本帖是前一贴的补充：

1. 使用大数据，了解怎么处理数据不能一次全部加载到内存的情况。如果你内存充足，当我没说
2. 训练好的模型的保存和使用
3. 使用的模型没变，还是简单的feedforward神经网络（update：添加CNN模型）
4. 如果你要运行本帖代码，推荐使用GPU版本或强大的VPS，我使用小笔记本差点等吐血
5. 后续有关于中文的练习《TensorFlow练习13: 制作一个简单的聊天机器人》《TensorFlow练习7: 基于RNN生成古诗词》《TensorFlow练习18: 根据姓名判断性别》

在正文开始之前，我画了一个机器学习模型的基本开发流程图：



使用的数据集

使用的数据集：http://help.sentiment140.com/for-students/ (情绪分析)

数据集包含1百60万条推特，包含消极、中性和积极tweet。不知道有没有现成的微博数据集。

数据格式：移除表情符号的CSV文件，字段如下：

- 0 – the polarity of the tweet (0 = negative, 2 = neutral, 4 = positive)
- 1 – the id of the tweet (2087)
- 2 – the date of the tweet (Sat May 16 23:58:44 UTC 2009)
- 3 – the query (lyx). If there is no query, then this value is NO_QUERY.

- 4 – the user that tweeted (robotickilldozr)
- 5 – the text of the tweet (Lyx is cool)

training.1600000.processed.noemoticon.csv（238M）
testdata.manual.2009.06.14.csv（74K）

数据预处理

```
import nltk
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer

import pickle
import numpy as np
import pandas as pd
from collections import OrderedDict

org_train_file = 'training.1600000.processed.noemoticon.csv'
org_test_file = 'testdata.manual.2009.06.14.csv'

# 提取文件中有用的字段
def usefull_filed(org_file, output_file):
    output = open(output_file, 'w')
    with open(org_file, buffering=10000, encoding='latin-1') as f:
        try:
            for line in f:                # "4","2193601966","Tue Jun 16 08:40:49 PDT 2009","NO_QUERY","Amanda
                line = line.replace('"', '')
                clf = line.split(',')[0]   # 4
                if clf == '0':
                    clf = [0, 0, 1]  # 消极评论
                elif clf == '2':
                    clf = [0, 1, 0]  # 中性评论
                elif clf == '4':
                    clf = [1, 0, 0]  # 积极评论

                tweet = line.split(',')[-1]
                outputline = str(clf) + ':%:%:%:' + tweet
```

```
                output.write(outputline)  # [0, 0, 1]:%:%:%: that's a bummer.  You shoulda got David Carr of
        except Exception as e:
            print(e)
    output.close()  # 处理完成，处理后文件大小127.5M

usefull_filed(org_train_file, 'training.csv')
usefull_filed(org_test_file, 'tesing.csv')

# 创建词汇表
def create_lexicon(train_file):
    lex = []
    lemmatizer = WordNetLemmatizer()
    with open(train_file, buffering=10000, encoding='latin-1') as f:
        try:
            count_word = {}  # 统计单词出现次数
            for line in f:
                tweet = line.split(':%:%:%:')[1]
                words = word_tokenize(line.lower())
                for word in words:
                    word = lemmatizer.lemmatize(word)
                    if word not in count_word:
                        count_word[word] = 1
                    else:
                        count_word[word] += 1

            count_word = OrderedDict(sorted(count_word.items(), key=lambda t: t[1]))
            for word in count_word:
                if count_word[word] < 100000 and count_word[word] > 100:  # 过滤掉一些词
                    lex.append(word)
        except Exception as e:
            print(e)
    return lex

lex = create_lexicon('training.csv')

with open('lexcion.pickle', 'wb') as f:
    pickle.dump(lex, f)


"""
# 把字符串转为向量
def string_to_vector(input_file, output_file, lex):
```

```
    output_f = open(output_file, 'w')
    lemmatizer = WordNetLemmatizer()
    with open(input_file, buffering=10000, encoding='latin-1') as f:
        for line in f:
            label = line.split(':%:%:%:')[0]
            tweet = line.split(':%:%:%:')[1]
            words = word_tokenize(tweet.lower())
            words = [lemmatizer.lemmatize(word) for word in words]

            features = np.zeros(len(lex))
            for word in words:
                if word in lex:
                    features[lex.index(word)] = 1  # 一个句子中某个词可能出现两次,可以用+=1，其实区别不大

            features = list(features)
            output_f.write(str(label) + ":" + str(features) + '\n')
    output_f.close()


f = open('lexcion.pickle', 'rb')
lex = pickle.load(f)
f.close()

# lexcion词汇表大小112k,training.vec大约112k*1600000  170G  太大，只能边转边训练了
# string_to_vector('training.csv', 'training.vec', lex)
# string_to_vector('tesing.csv', 'tesing.vec', lex)
"""
```

上面代码把原始数据转为training.csv、和tesing.csv，里面只包含label和tweet。lexcion.pickle文件保存了词汇表。

> *如果数据文件太大，不能一次加载到内存，可以把数据导入数据库*
> *Dask可处理大csv文件*

开始漫长的训练

```
import os
import random
```

```
import tensorflow as tf
import pickle
import numpy as np
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer

f = open('lexcion.pickle', 'rb')
lex = pickle.load(f)
f.close()


def get_random_line(file, point):
    file.seek(point)
    file.readline()
    return file.readline()
# 从文件中随机选择n条记录
def get_n_random_line(file_name, n=150):
    lines = []
    file = open(file_name, encoding='latin-1')
    total_bytes = os.stat(file_name).st_size
    for i in range(n):
        random_point = random.randint(0, total_bytes)
        lines.append(get_random_line(file, random_point))
    file.close()
    return lines


def get_test_dataset(test_file):
    with open(test_file, encoding='latin-1') as f:
        test_x = []
        test_y = []
        lemmatizer = WordNetLemmatizer()
        for line in f:
            label = line.split(':%:%:%:')[0]
            tweet = line.split(':%:%:%:')[1]
            words = word_tokenize(tweet.lower())
            words = [lemmatizer.lemmatize(word) for word in words]
            features = np.zeros(len(lex))
            for word in words:
                if word in lex:
                    features[lex.index(word)] = 1

```

```
            test_x.append(list(features))
            test_y.append(eval(label))
    return test_x, test_y

test_x, test_y = get_test_dataset('tesing.csv')


#######################################################################

n_input_layer = len(lex)  # 输入层

n_layer_1 = 2000     # hide layer
n_layer_2 = 2000    # hide layer(隐藏层)听着很神秘，其实就是除输入输出层外的中间层

n_output_layer = 3       # 输出层


def neural_network(data):
    # 定义第一层"神经元"的权重和biases
    layer_1_w_b = {'w_':tf.Variable(tf.random_normal([n_input_layer, n_layer_1])), 'b_':tf.Variable(tf.rando
    # 定义第二层"神经元"的权重和biases
    layer_2_w_b = {'w_':tf.Variable(tf.random_normal([n_layer_1, n_layer_2])), 'b_':tf.Variable(tf.random_no
    # 定义输出层"神经元"的权重和biases
    layer_output_w_b = {'w_':tf.Variable(tf.random_normal([n_layer_2, n_output_layer])), 'b_':tf.Variable(tf

    # w·x+b
    layer_1 = tf.add(tf.matmul(data, layer_1_w_b['w_']), layer_1_w_b['b_'])
    layer_1 = tf.nn.relu(layer_1)  # 激活函数
    layer_2 = tf.add(tf.matmul(layer_1, layer_2_w_b['w_']), layer_2_w_b['b_'])
    layer_2 = tf.nn.relu(layer_2 ) # 激活函数
    layer_output = tf.add(tf.matmul(layer_2, layer_output_w_b['w_']), layer_output_w_b['b_'])

    return layer_output


X = tf.placeholder('float')
Y = tf.placeholder('float')
batch_size = 90

def train_neural_network(X, Y):
    predict = neural_network(X)
    cost_func = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(predict, Y))
```

```
    optimizer = tf.train.AdamOptimizer().minimize(cost_func)

    with tf.Session() as session:
        session.run(tf.initialize_all_variables())

        lemmatizer = WordNetLemmatizer()
        saver = tf.train.Saver()
        i = 0
        pre_accuracy = 0
        while True:   # 一直训练
            batch_x = []
            batch_y = []

            #if model.ckpt文件已存在:
            #   saver.restore(session, 'model.ckpt')  恢复保存的session

            try:
                lines = get_n_random_line('training.csv', batch_size)
                for line in lines:
                    label = line.split(':%:%:%:')[0]
                    tweet = line.split(':%:%:%:')[1]
                    words = word_tokenize(tweet.lower())
                    words = [lemmatizer.lemmatize(word) for word in words]

                    features = np.zeros(len(lex))
                    for word in words:
                        if word in lex:
                            features[lex.index(word)] = 1  # 一个句子中某个词可能出现两次,可以用+=1，其实区别不大

                    batch_x.append(list(features))
                    batch_y.append(eval(label))

                session.run([optimizer, cost_func], feed_dict={X:batch_x,Y:batch_y})
            except Exception as e:
                print(e)

            # 准确率
            if i > 100:
                correct = tf.equal(tf.argmax(predict,1), tf.argmax(Y,1))
                accuracy = tf.reduce_mean(tf.cast(correct,'float'))
                accuracy = accuracy.eval({X:test_x, Y:test_y})
                if accuracy > pre_accuracy:  # 保存准确率最高的训练模型
```

```
                    print('准确率: ', accuracy)
                    pre_accuracy = accuracy
                    saver.save(session, 'model.ckpt')  # 保存session
                i = 0
            i += 1


train_neural_network(X,Y)
```

上面程序占用内存600M，峰值1G。

运行：

```
(env) tianshuais-MacBook-Air:tf tianshuai$ python training.py
准确率:  0.459839
准确率:  0.477912
准确率:  0.493976
准确率:  0.495984
准确率:  0.508032
准确率:  0.51004
准确率:  0.516064
准确率:  0.522088
准确率:  0.526104
准确率:  0.528112
准确率:  0.532129
```

训练模型保存为model.ckpt。

使用训练好的模型

```
import tensorflow as tf
import pickle
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
import numpy as np

f = open('lexcion.pickle', 'rb')
lex = pickle.load(f)
```

```
f.close()

n_input_layer = len(lex)  # 输入层

n_layer_1 = 2000     # hide layer
n_layer_2 = 2000    # hide layer(隐藏层)听着很神秘，其实就是除输入输出层外的中间层

n_output_layer = 3       # 输出层
def neural_network(data):
    # 定义第一层"神经元"的权重和biases
    layer_1_w_b = {'w_':tf.Variable(tf.random_normal([n_input_layer, n_layer_1])), 'b_':tf.Variable(tf.random_
    # 定义第二层"神经元"的权重和biases
    layer_2_w_b = {'w_':tf.Variable(tf.random_normal([n_layer_1, n_layer_2])), 'b_':tf.Variable(tf.random_norr
    # 定义输出层"神经元"的权重和biases
    layer_output_w_b = {'w_':tf.Variable(tf.random_normal([n_layer_2, n_output_layer])), 'b_':tf.Variable(tf.r

    # w·x+b
    layer_1 = tf.add(tf.matmul(data, layer_1_w_b['w_']), layer_1_w_b['b_'])
    layer_1 = tf.nn.relu(layer_1)  # 激活函数
    layer_2 = tf.add(tf.matmul(layer_1, layer_2_w_b['w_']), layer_2_w_b['b_'])
    layer_2 = tf.nn.relu(layer_2 ) # 激活函数
    layer_output = tf.add(tf.matmul(layer_2, layer_output_w_b['w_']), layer_output_w_b['b_'])

    return layer_output

X = tf.placeholder('float')
def prediction(tweet_text):
    predict = neural_network(X)

    with tf.Session() as session:
        session.run(tf.initialize_all_variables())
        saver = tf.train.Saver()
        saver.restore(session, 'model.ckpt')

        lemmatizer = WordNetLemmatizer()
        words = word_tokenize(tweet_text.lower())
        words = [lemmatizer.lemmatize(word) for word in words]

        features = np.zeros(len(lex))
        for word in words:
            if word in lex:
                features[lex.index(word)] = 1
```

```

        #print(predict.eval(feed_dict={X:[features]})) [[val1,val2,val3]]
        res = session.run(tf.argmax(predict.eval(feed_dict={X:[features]}),1 ))
        return res


prediction("I am very happe")
```

上面使用简单的feedfroward模型，下面使用CNN模型

```
# https://github.com/Lab41/sunny-side-up
import os
import random
import tensorflow as tf
import pickle
import numpy as np
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer

f = open('lexcion.pickle', 'rb')
lex = pickle.load(f)
f.close()

def get_random_line(file, point):
    file.seek(point)
    file.readline()
    return file.readline()
# 从文件中随机选择n条记录
def get_n_random_line(file_name, n=150):
    lines = []
    file = open(file_name, encoding='latin-1')
    total_bytes = os.stat(file_name).st_size
    for i in range(n):
        random_point = random.randint(0, total_bytes)
        lines.append(get_random_line(file, random_point))
    file.close()
    return lines

def get_test_dataset(test_file):
```

```
    with open(test_file, encoding='latin-1') as f:
        test_x = []
        test_y = []
        lemmatizer = WordNetLemmatizer()
        for line in f:
            label = line.split(':%:%:%:')[0]
            tweet = line.split(':%:%:%:')[1]
            words = word_tokenize(tweet.lower())
            words = [lemmatizer.lemmatize(word) for word in words]
            features = np.zeros(len(lex))
            for word in words:
                if word in lex:
                    features[lex.index(word)] = 1

            test_x.append(list(features))
            test_y.append(eval(label))
    return test_x, test_y

test_x, test_y = get_test_dataset('tesing.csv')
##############################################################################
input_size = len(lex)
num_classes = 3

X = tf.placeholder(tf.int32, [None, input_size])
Y = tf.placeholder(tf.float32, [None, num_classes])

dropout_keep_prob = tf.placeholder(tf.float32)

batch_size = 90

def neural_network():
    # embedding layer
    with tf.device('/cpu:0'), tf.name_scope("embedding"):
        embedding_size = 128
        W = tf.Variable(tf.random_uniform([input_size, embedding_size], -1.0, 1.0))
        embedded_chars = tf.nn.embedding_lookup(W, X)
        embedded_chars_expanded = tf.expand_dims(embedded_chars, -1)
    # convolution + maxpool layer
    num_filters = 128
    filter_sizes = [3,4,5]
    pooled_outputs = []
    for i, filter_size in enumerate(filter_sizes):
```

```
        with tf.name_scope("conv-maxpool-%s" % filter_size):
            filter_shape = [filter_size, embedding_size, 1, num_filters]
            W = tf.Variable(tf.truncated_normal(filter_shape, stddev=0.1))
            b = tf.Variable(tf.constant(0.1, shape=[num_filters]))
            conv = tf.nn.conv2d(embedded_chars_expanded, W, strides=[1, 1, 1, 1], padding="VALID")
            h = tf.nn.relu(tf.nn.bias_add(conv, b))
            pooled = tf.nn.max_pool(h, ksize=[1, input_size - filter_size + 1, 1, 1], strides=[1, 1, 1, 1], p
            pooled_outputs.append(pooled)

    num_filters_total = num_filters * len(filter_sizes)
    h_pool = tf.concat(3, pooled_outputs)
    h_pool_flat = tf.reshape(h_pool, [-1, num_filters_total])
    # dropout
    with tf.name_scope("dropout"):
        h_drop = tf.nn.dropout(h_pool_flat, dropout_keep_prob)
    # output
    with tf.name_scope("output"):
        W = tf.get_variable("W", shape=[num_filters_total, num_classes], initializer=tf.contrib.layers.xavie
        b = tf.Variable(tf.constant(0.1, shape=[num_classes]))
        output = tf.nn.xw_plus_b(h_drop, W, b)

    return output

def train_neural_network():
    output = neural_network()

    optimizer = tf.train.AdamOptimizer(1e-3)
    loss = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(output, Y))
    grads_and_vars = optimizer.compute_gradients(loss)
    train_op = optimizer.apply_gradients(grads_and_vars)

    saver = tf.train.Saver(tf.global_variables())
    with tf.Session() as sess:
        sess.run(tf.global_variables_initializer())

        lemmatizer = WordNetLemmatizer()
        i = 0
        while True:
            batch_x = []
            batch_y = []

            #if model.ckpt文件已存在:
```

```
            #   saver.restore(session, 'model.ckpt')  恢复保存的session
            try:
                lines = get_n_random_line('training.csv', batch_size)
                for line in lines:
                    label = line.split(':%:%:%:')[0]
                    tweet = line.split(':%:%:%:')[1]
                    words = word_tokenize(tweet.lower())
                    words = [lemmatizer.lemmatize(word) for word in words]

                    features = np.zeros(len(lex))
                    for word in words:
                        if word in lex:
                            features[lex.index(word)] = 1  # 一个句子中某个词可能出现两次,可以用+=1，其实区别不大

                    batch_x.append(list(features))
                    batch_y.append(eval(label))

                _, loss_ = sess.run([train_op, loss], feed_dict={X:batch_x, Y:batch_y, dropout_keep_prob:0.5
                print(loss_)
            except Exception as e:
                print(e)

            if i % 10 == 0:
                predictions = tf.argmax(output, 1)
                correct_predictions = tf.equal(predictions, tf.argmax(Y, 1))
                accuracy = tf.reduce_mean(tf.cast(correct_predictions, "float"))
                accur = sess.run(accuracy, feed_dict={X:test_x[0:50], Y:test_y[0:50], dropout_keep_prob:1.0}
                print('准确率:', accur)

            i += 1

train_neural_network()
```

使用了CNN模型之后，准确率有了显著提升。

Facebook Google+ Twitter Weibo Email

相关文章

Ubuntu 16.04 安装 Tensorflow(GPU支持)

TensorFlow练习1: 对评论进行分类

TensorFlow练习5: 训练一个简单的游戏AI（Deep Q Network）...

TensorFlow练习8: 生成音乐

TensorFlow练习15: 中文语音识别

2016年11月16日 wtf ML、coding TensorFlow、教程

《TensorFlow练习2: 对评论进行分类》有5个想法

liang

2017年3月2日 下午5:20

你好，我在运行fnn时有一些错误，你写的这行 label = line.split(‘:%:%:%:’)[0]返回的是字符串吧，是不是不能直接当做标签呢

骆炜

2017年2月28日 下午2:44

你好， 我在训练的时候遇到两个错误，不知道是不是TF版本的问题（1.0）。能帮我看一下么

cost_func = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(predict, Y))

ValueError: Only call softmax_cross_entropy_with_logits with named arguments (labels=…, logits=…, …)

optimizer = tf.train.AdamOptimizer().minimize(cost_func)

ValueError: No gradients provided for any variable, check your graph for ops that do not support gradients, between variables

wtf

2017年2月28日 下午4:04

API变了 看一下文档

Neo

2016年12月28日 下午4:35

我按照你给的地址下载了数据，然后发现数据里面有很多乱码，你是怎么解决的啊？

billchen

2017年2月4日 下午4:44

with open(org_file, buffering=10000, errors='ignore') as f: