Tesoroflow学习笔记及经典代码收藏

# 一、学习笔记

## 一、selfCar

### 1.Introduction to tensorflow第7节

36,有个notobook，初次使用tensorflow的，可以做一下；

把这节课的概念完整串一下，包括 reduce train；

### 2.Deep neural Network s 第8节

选用多层线性模型来构成神经网络：1、稳定，不会因为输入值的小变化引起输出的大变化，2、求导很容易；

深度网络：1、跟宽度相比，通过让网络变深，可以只增加少量参数，就让网络有更好的表现；2、网络会出现结构化的效果，浅层网络会学习到简单的结构，比如线；深层次的结构会学习到更复杂的东西，比如几何形状；再深的层次就会学习到物件等等；

正则化需要再复习一下吴恩达的课程，

#### 关于范数

在深度学习中，监督类学习问题其实就是在规则化参数同时最小化误差。最小化误差目的是让模型拟合训练数据，而规则化参数的目的是防止模型过分拟合训练数据。

参数太多，会导致模型复杂度上升，容易过拟合，也就是训练误差小，测试误差大。因此，我们需要保证模型足够简单，并在此基础上训练误差小，这样训练得到的参数才能保证测试误差也小，而模型简单就是通过规则函数来实现的。

规则化项可以是模型参数向量的范数。如：L0、L1、L2等。

一、L0范数与L1范数

L0范数是指向量中非0的元素的个数。如果我们用L0范数来规则化一个参数矩阵W的话，就是希望W的大部分元素都是0。换句话说，让参数W是稀疏的。

L1范数是指向量中各个元素绝对值之和。L1范数是L0范数的最优凸近似。任何的规则化算子，如果他在Wi=0的地方不可微，并且可以分解为一个“求和”的形式，那么这个规则化算子就可以实现稀疏。W的L1范数是绝对值，|w|在w=0处是不可微。

虽然L0可以实现稀疏，但是实际中会使用L1取代L0。因为L0范数很难优化求解，L1范数是L0范数的最优凸近似，它比L0范数要容易优化求解。

二、L2范数

L2范数，又叫“岭回归”（Ridge Regression）、“权值衰减”（weight decay）。这用的很多吧，它的作用是改善过拟合。过拟合是：模型训练时候的误差很小，但是测试误差很大，也就是说模型复杂到可以拟合到所有训练数据，但在预测新的数据的时候，结果很差。

L2范数是指向量中各元素的平方和然后开根。我们让L2范数的规则项||W||2最小，可以使得W的每个元素都很小，都接近于0。而越小的参数说明模型越简单，越简单的模型则越不容易产生过拟合现象。

### 3、CNN

Translation Invariance平移不变性

一般计算下层的尺寸公式：

W\_out =[ (W−F+2P)/S] + 1

Tensorflow的计算方法：

**SAME Padding**,

out\_height = ceil(float(in\_height) / float(strides[1]))

**VALID Padding**,

out\_height = ceil(float(in\_height - filter\_height + 1) / float(strides[1]))

# 二、代码收藏

## 1、完整线性模型代码：

**from** tensorflow.examples.tutorials.mnist **import** input\_data

**import** tensorflow **as** tf

**import** numpy **as** np

**from** helper **import** batches *# Helper function created in Mini-batching section*

**def** **print\_epoch\_stats**(epoch\_i, sess, last\_features, last\_labels):

"""

Print cost and validation accuracy of an epoch

"""

current\_cost = sess.run(

cost,

feed\_dict={features: last\_features, labels: last\_labels})

valid\_accuracy = sess.run(

accuracy,

feed\_dict={features: valid\_features, labels: valid\_labels})

print('Epoch: {:<4} - Cost: {:<8.3} Valid Accuracy: {:<5.3}'.format(

epoch\_i,

current\_cost,

valid\_accuracy))

n\_input = 784 *# MNIST data input (img shape: 28\*28)*

n\_classes = 10 *# MNIST total classes (0-9 digits)*

*# Import MNIST data*

mnist = input\_data.read\_data\_sets('/datasets/ud730/mnist', one\_hot=**True**)

*# The features are already scaled and the data is shuffled*

train\_features = mnist.train.images

valid\_features = mnist.validation.images

test\_features = mnist.test.images

train\_labels = mnist.train.labels.astype(np.float32)

valid\_labels = mnist.validation.labels.astype(np.float32)

test\_labels = mnist.test.labels.astype(np.float32)

*# Features and Labels*

features = tf.placeholder(tf.float32, [**None**, n\_input])

labels = tf.placeholder(tf.float32, [**None**, n\_classes])

*# Weights & bias*

weights = tf.Variable(tf.random\_normal([n\_input, n\_classes]))

bias = tf.Variable(tf.random\_normal([n\_classes]))

*# Logits - xW + b*

logits = tf.add(tf.matmul(features, weights), bias)

*# Define loss and optimizer*

learning\_rate = tf.placeholder(tf.float32)

cost = tf.reduce\_mean(tf.nn.softmax\_cross\_entropy\_with\_logits(logits=logits, labels=labels))

optimizer = tf.train.GradientDescentOptimizer(learning\_rate=learning\_rate).minimize(cost)

*# Calculate accuracy*

correct\_prediction = tf.equal(tf.argmax(logits, 1), tf.argmax(labels, 1))

accuracy = tf.reduce\_mean(tf.cast(correct\_prediction, tf.float32))

init = tf.global\_variables\_initializer()

batch\_size = 128

epochs = 10

learn\_rate = 0.001

train\_batches = batches(batch\_size, train\_features, train\_labels)

**with** tf.Session() **as** sess:

sess.run(init)

*# Training cycle*

**for** epoch\_i **in** range(epochs):

*# Loop over all batches*

**for** batch\_features, batch\_labels **in** train\_batches:

train\_feed\_dict = {

features: batch\_features,

labels: batch\_labels,

learning\_rate: learn\_rate}

sess.run(optimizer, feed\_dict=train\_feed\_dict)

*# Print cost and validation accuracy of an epoch*

print\_epoch\_stats(epoch\_i, sess, batch\_features, batch\_labels)

*# Calculate accuracy for test dataset*

test\_accuracy = sess.run(

accuracy,

feed\_dict={features: test\_features, labels: test\_labels})

print('Test Accuracy: {}'.format(test\_accuracy))

## 2、保存及加载模型

### 1、保存示例

**import** tensorflow **as** tf

*# The file path to save the data*

save\_file = './model.ckpt'

*# Two Tensor Variables: weights and bias*

weights = tf.Variable(tf.truncated\_normal([2, 3]))

bias = tf.Variable(tf.truncated\_normal([3]))

*# Class used to save and/or restore Tensor Variables*

saver = tf.train.Saver()

**with** tf.Session() **as** sess:

*# Initialize all the Variables*

sess.run(tf.global\_variables\_initializer())

*# Show the values of weights and bias*

print('Weights:')

print(sess.run(weights))

print('Bias:')

print(sess.run(bias))

*# Save the model*

saver.save(sess, save\_file)

### 2、加载示例

注意，这里用了restore来加载参数，就不需要初始化了。

*# Remove the previous weights and bias*

tf.reset\_default\_graph()

*# Two Variables: weights and bias*

weights = tf.Variable(tf.truncated\_normal([2, 3]))

bias = tf.Variable(tf.truncated\_normal([3]))

*# Class used to save and/or restore Tensor Variables*

saver = tf.train.Saver()

**with** tf.Session() **as** sess:

*# Load the weights and bias*

saver.restore(sess, save\_file)

*# Show the values of weights and bias*

print('Weight:')

print(sess.run(weights))

print('Bias:')

print(sess.run(bias))

### 3、训练模型保存示例

*# Remove previous Tensors and Operations*

tf.reset\_default\_graph()

**from** tensorflow.examples.tutorials.mnist **import** input\_data

**import** numpy **as** np

learning\_rate = 0.001

n\_input = 784 *# MNIST data input (img shape: 28\*28)*

n\_classes = 10 *# MNIST total classes (0-9 digits)*

*# Import MNIST data*

mnist = input\_data.read\_data\_sets('.', one\_hot=**True**)

*# Features and Labels*

features = tf.placeholder(tf.float32, [**None**, n\_input])

labels = tf.placeholder(tf.float32, [**None**, n\_classes])

*# Weights & bias*

weights = tf.Variable(tf.random\_normal([n\_input, n\_classes]))

bias = tf.Variable(tf.random\_normal([n\_classes]))

*# Logits - xW + b*

logits = tf.add(tf.matmul(features, weights), bias)

*# Define loss and optimizer*

cost = tf.reduce\_mean(\

tf.nn.softmax\_cross\_entropy\_with\_logits(logits=logits, labels=labels))

optimizer = tf.train.GradientDescentOptimizer(learning\_rate=learning\_rate)\

.minimize(cost)

*# Calculate accuracy*

correct\_prediction = tf.equal(tf.argmax(logits, 1), tf.argmax(labels, 1))

accuracy = tf.reduce\_mean(tf.cast(correct\_prediction, tf.float32))

**import** math

save\_file = './train\_model.ckpt'

batch\_size = 128

n\_epochs = 100

saver = tf.train.Saver()

*# Launch the graph*

**with** tf.Session() **as** sess:

sess.run(tf.global\_variables\_initializer())

*# Training cycle*

**for** epoch **in** range(n\_epochs):

total\_batch = math.ceil(mnist.train.num\_examples / batch\_size)

*# Loop over all batches*

**for** i **in** range(total\_batch):

batch\_features, batch\_labels = mnist.train.next\_batch(batch\_size)

sess.run(

optimizer,

feed\_dict={features: batch\_features, labels: batch\_labels})

*# Print status for every 10 epochs*

**if** epoch % 10 == 0:

valid\_accuracy = sess.run(

accuracy,

feed\_dict={

features: mnist.validation.images,

labels: mnist.validation.labels})

print('Epoch {:<3} - Validation Accuracy: {}'.format(

epoch,

valid\_accuracy))

*# Save the model*

saver.save(sess, save\_file)

print('Trained Model Saved.')

4、训练模型加载参数示例

saver = tf.train.Saver()

*# Launch the graph*

**with** tf.Session() **as** sess:

saver.restore(sess, save\_file)

test\_accuracy = sess.run(

accuracy,

feed\_dict={features: mnist.test.images, labels: mnist.test.labels})

print('Test Accuracy: {}'.format(test\_accuracy))

## 3、预处理数据，训练模型进度条、训练结果画图示例

见IntroTensorflow

## 4、卷积网络代码示例

### 1、简单

*# Output depth*

k\_output = 64

*# Image Properties*

image\_width = 10

image\_height = 10

color\_channels = 3

*# Convolution filter*

filter\_size\_width = 5

filter\_size\_height = 5

*# Input/Image*

input = tf.placeholder(

tf.float32,

shape=[**None**, image\_height, image\_width, color\_channels])

*# Weight and bias*

weight = tf.Variable(tf.truncated\_normal(

[filter\_size\_height, filter\_size\_width, color\_channels, k\_output]))

bias = tf.Variable(tf.zeros(k\_output))

*# Apply Convolution*

conv\_layer = tf.nn.conv2d(input, weight, strides=[1, 2, 2, 1], padding='SAME')

*# Add bias*

conv\_layer = tf.nn.bias\_add(conv\_layer, bias)

*# Apply activation function*

conv\_layer = tf.nn.relu(conv\_layer)

2、完整

**from** tensorflow.examples.tutorials.mnist **import** input\_data

mnist = input\_data.read\_data\_sets(".", one\_hot=**True**, reshape=**False**)

**import** tensorflow **as** tf

*# Parameters*

learning\_rate = 0.00001

epochs = 10

batch\_size = 128

*# Number of samples to calculate validation and accuracy*

*# Decrease this if you're running out of memory to calculate accuracy*

test\_valid\_size = 256

*# Network Parameters*

n\_classes = 10 *# MNIST total classes (0-9 digits)*

dropout = 0.75 *# Dropout, probability to keep units*

*# Store layers weight & bias*

weights = {

'wc1': tf.Variable(tf.random\_normal([5, 5, 1, 32])),

'wc2': tf.Variable(tf.random\_normal([5, 5, 32, 64])),

'wd1': tf.Variable(tf.random\_normal([7\*7\*64, 1024])),

'out': tf.Variable(tf.random\_normal([1024, n\_classes]))}

biases = {

'bc1': tf.Variable(tf.random\_normal([32])),

'bc2': tf.Variable(tf.random\_normal([64])),

'bd1': tf.Variable(tf.random\_normal([1024])),

'out': tf.Variable(tf.random\_normal([n\_classes]))}

**def** **conv2d**(x, W, b, strides=1):

x = tf.nn.conv2d(x, W, strides=[1, strides, strides, 1], padding='SAME')

x = tf.nn.bias\_add(x, b)

**return** tf.nn.relu(x)

**def** **maxpool2d**(x, k=2):

**return** tf.nn.max\_pool(

x,

ksize=[1, k, k, 1],

strides=[1, k, k, 1],

padding='SAME')

**def** **conv\_net**(x, weights, biases, dropout):

*# Layer 1 - 28\*28\*1 to 14\*14\*32*

conv1 = conv2d(x, weights['wc1'], biases['bc1'])

conv1 = maxpool2d(conv1, k=2)

*# Layer 2 - 14\*14\*32 to 7\*7\*64*

conv2 = conv2d(conv1, weights['wc2'], biases['bc2'])

conv2 = maxpool2d(conv2, k=2)

*# Fully connected layer - 7\*7\*64 to 1024*

fc1 = tf.reshape(conv2, [-1, weights['wd1'].get\_shape().as\_list()[0]])

fc1 = tf.add(tf.matmul(fc1, weights['wd1']), biases['bd1'])

fc1 = tf.nn.relu(fc1)

fc1 = tf.nn.dropout(fc1, dropout)

*# Output Layer - class prediction - 1024 to 10*

out = tf.add(tf.matmul(fc1, weights['out']), biases['out'])

**return** out

*# tf Graph input*

x = tf.placeholder(tf.float32, [**None**, 28, 28, 1])

y = tf.placeholder(tf.float32, [**None**, n\_classes])

keep\_prob = tf.placeholder(tf.float32)

*# Model*

logits = conv\_net(x, weights, biases, keep\_prob)

*# Define loss and optimizer*

cost = tf.reduce\_mean(\

tf.nn.softmax\_cross\_entropy\_with\_logits(logits=logits, labels=y))

optimizer = tf.train.GradientDescentOptimizer(learning\_rate=learning\_rate)\

.minimize(cost)

*# Accuracy*

correct\_pred = tf.equal(tf.argmax(logits, 1), tf.argmax(y, 1))

accuracy = tf.reduce\_mean(tf.cast(correct\_pred, tf.float32))

*# Initializing the variables*

init = tf. global\_variables\_initializer()

*# Launch the graph*

**with** tf.Session() **as** sess:

sess.run(init)

**for** epoch **in** range(epochs):

**for** batch **in** range(mnist.train.num\_examples//batch\_size):

batch\_x, batch\_y = mnist.train.next\_batch(batch\_size)

sess.run(optimizer, feed\_dict={

x: batch\_x,

y: batch\_y,

keep\_prob: dropout})

*# Calculate batch loss and accuracy*

loss = sess.run(cost, feed\_dict={

x: batch\_x,

y: batch\_y,

keep\_prob: 1.})

valid\_acc = sess.run(accuracy, feed\_dict={

x: mnist.validation.images[:test\_valid\_size],

y: mnist.validation.labels[:test\_valid\_size],

keep\_prob: 1.})

print('Epoch {:>2}, Batch {:>3} -'

'Loss: {:>10.4f} Validation Accuracy: {:.6f}'.format(

epoch + 1,

batch + 1,

loss,

valid\_acc))

*# Calculate Test Accuracy*

test\_acc = sess.run(accuracy, feed\_dict={

x: mnist.test.images[:test\_valid\_size],

y: mnist.test.labels[:test\_valid\_size],

keep\_prob: 1.})

print('Testing Accuracy: {}'.format(test\_acc))

### 2、分离tensor。 Tf.split()

tf.split(  
    value,  
    num\_or\_size\_splits,  
    axis=0,  
    num=None,  
    name='split'  
)

Defined in [tensorflow/python/ops/array\_ops.py](https://www.github.com/tensorflow/tensorflow/blob/r1.8/tensorflow/python/ops/array_ops.py).

See the guide: [Tensor Transformations > Slicing and Joining](https://www.tensorflow.org/api_guides/python/array_ops#Slicing_and_Joining)

Splits a tensor into sub tensors.

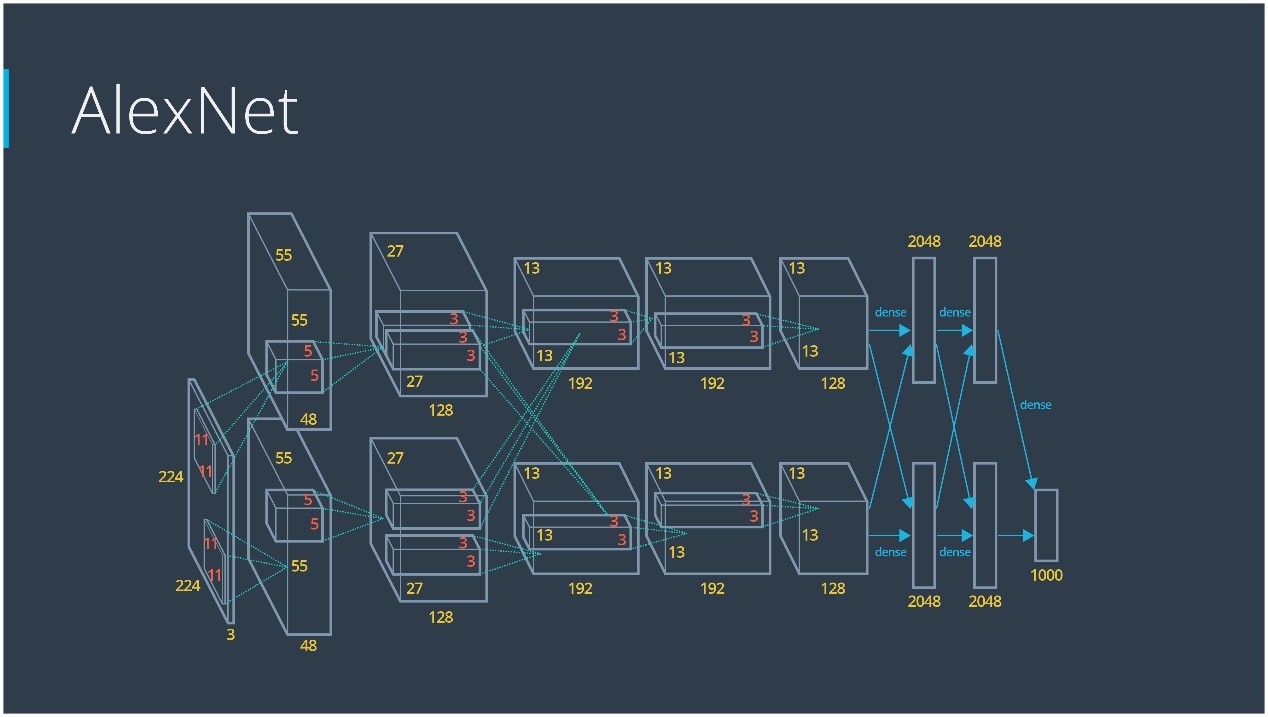
If num\_or\_size\_splits is an integer type, num\_split, then splits value along dimension axis into num\_splitsmaller tensors. Requires that num\_split evenly divides value.shape[axis].

If num\_or\_size\_splits is not an integer type, it is presumed to be a Tensor size\_splits, then splits value into len(size\_splits) pieces. The shape of the i-th piece has the same size as the value except along dimensionaxis where the size is size\_splits[i].

For example:

# 'value' is a tensor with shape [5, 30]  
# Split 'value' into 3 tensors with sizes [4, 15, 11] along dimension 1  
split0, split1, split2 = tf.split(value, [4, 15, 11], 1)  
tf.shape(split0)  # [5, 4]  
tf.shape(split1)  # [5, 15]  
tf.shape(split2)  # [5, 11]  
# Split 'value' into 3 tensors along dimension 1  
split0, split1, split2 = tf.split(value, num\_or\_size\_splits=3, axis=1)  
tf.shape(split0)  # [5, 10]

### 3、Alexnet



Alexnet实现代码

#### Alexnet.py

import numpy as np

import tensorflow as tf

net\_data = np.load("bvlc-alexnet.npy", encoding="latin1").item()

# param input-input kernel-weights biases-bias k\_h/kw 滤波器尺寸 sh/sw strides c\_o输出深度

def conv(input, kernel, biases, k\_h, k\_w, c\_o, s\_h, s\_w, padding="VALID", group=1):

'''

From https://github.com/ethereon/caffe-tensorflow

'''

# 图像通道数量

c\_i = input.get\_shape()[-1]

# 通道数量是整数？

assert c\_i % group == 0

assert c\_o % group == 0

# Param： i-input k-weights []-strides

convolve = lambda i, k: tf.nn.conv2d(i, k, [1, s\_h, s\_w, 1], padding=padding)

if tf.\_\_version\_\_ < "1.0.0":

if group == 1:

conv = convolve(input, kernel)

else:

input\_groups = tf.split(3, group, input)

kernel\_groups = tf.split(3, group, kernel)

output\_groups = [convolve(i, k) for i, k in zip(input\_groups, kernel\_groups)]

conv = tf.concat(3, output\_groups)

else:

if group == 1:

conv = convolve(input, kernel)

else:

# 把input这个tensor，在axis=3上（input最后一个维度是通道数量），分成group的形状，这里就是分成一维的

input\_groups = tf.split(input, group, 3)

kernel\_groups = tf.split(kernel, group, 3)

output\_groups = [convolve(i, k) for i, k in zip(input\_groups, kernel\_groups)]

conv = tf.concat(output\_groups, 3)

return tf.reshape(tf.nn.bias\_add(conv, biases), [-1] + conv.get\_shape().as\_list()[1:])

def AlexNet(features, feature\_extract=False):

"""

Builds an AlexNet model, loads pretrained weights

"""

# conv1

# conv(11, 11, 96, 4, 4, padding='VALID', name='conv1')

k\_h = 11

k\_w = 11

c\_o = 96

s\_h = 4

s\_w = 4

conv1W = tf.Variable(net\_data["conv1"][0])

conv1b = tf.Variable(net\_data["conv1"][1])

conv1\_in = conv(features, conv1W, conv1b, k\_h, k\_w, c\_o, s\_h, s\_w, padding="SAME", group=1)

conv1 = tf.nn.relu(conv1\_in)

# lrn1

# lrn(2, 2e-05, 0.75, name='norm1')

radius = 2

alpha = 2e-05

beta = 0.75

bias = 1.0

lrn1 = tf.nn.local\_response\_normalization(conv1, depth\_radius=radius, alpha=alpha, beta=beta, bias=bias)

# maxpool1

# max\_pool(3, 3, 2, 2, padding='VALID', name='pool1')

k\_h = 3

k\_w = 3

s\_h = 2

s\_w = 2

padding = 'VALID'

maxpool1 = tf.nn.max\_pool(lrn1, ksize=[1, k\_h, k\_w, 1], strides=[1, s\_h, s\_w, 1], padding=padding)

# conv2

# conv(5, 5, 256, 1, 1, group=2, name='conv2')

k\_h = 5

k\_w = 5

c\_o = 256

s\_h = 1

s\_w = 1

group = 2

conv2W = tf.Variable(net\_data["conv2"][0])

conv2b = tf.Variable(net\_data["conv2"][1])

conv2\_in = conv(maxpool1, conv2W, conv2b, k\_h, k\_w, c\_o, s\_h, s\_w, padding="SAME", group=group)

conv2 = tf.nn.relu(conv2\_in)

# lrn2

# lrn(2, 2e-05, 0.75, name='norm2')

# 局部响应正则化

radius = 2

alpha = 2e-05

beta = 0.75

bias = 1.0

lrn2 = tf.nn.local\_response\_normalization(conv2, depth\_radius=radius, alpha=alpha, beta=beta, bias=bias)

# maxpool2

# max\_pool(3, 3, 2, 2, padding='VALID', name='pool2')

k\_h = 3

k\_w = 3

s\_h = 2

s\_w = 2

padding = 'VALID'

maxpool2 = tf.nn.max\_pool(lrn2, ksize=[1, k\_h, k\_w, 1], strides=[1, s\_h, s\_w, 1], padding=padding)

# conv3

# conv(3, 3, 384, 1, 1, name='conv3')

k\_h = 3

k\_w = 3

c\_o = 384

s\_h = 1

s\_w = 1

group = 1

conv3W = tf.Variable(net\_data["conv3"][0])

conv3b = tf.Variable(net\_data["conv3"][1])

conv3\_in = conv(maxpool2, conv3W, conv3b, k\_h, k\_w, c\_o, s\_h, s\_w, padding="SAME", group=group)

conv3 = tf.nn.relu(conv3\_in)

# conv4

# conv(3, 3, 384, 1, 1, group=2, name='conv4')

k\_h = 3

k\_w = 3

c\_o = 384

s\_h = 1

s\_w = 1

group = 2

conv4W = tf.Variable(net\_data["conv4"][0])

conv4b = tf.Variable(net\_data["conv4"][1])

conv4\_in = conv(conv3, conv4W, conv4b, k\_h, k\_w, c\_o, s\_h, s\_w, padding="SAME", group=group)

conv4 = tf.nn.relu(conv4\_in)

# conv5

# conv(3, 3, 256, 1, 1, group=2, name='conv5')

k\_h = 3

k\_w = 3

c\_o = 256

s\_h = 1

s\_w = 1

group = 2

conv5W = tf.Variable(net\_data["conv5"][0])

conv5b = tf.Variable(net\_data["conv5"][1])

conv5\_in = conv(conv4, conv5W, conv5b, k\_h, k\_w, c\_o, s\_h, s\_w, padding="SAME", group=group)

conv5 = tf.nn.relu(conv5\_in)

# maxpool5

# max\_pool(3, 3, 2, 2, padding='VALID', name='pool5')

k\_h = 3

k\_w = 3

s\_h = 2

s\_w = 2

padding = 'VALID'

maxpool5 = tf.nn.max\_pool(conv5, ksize=[1, k\_h, k\_w, 1], strides=[1, s\_h, s\_w, 1], padding=padding)

# fc6, 4096

fc6W = tf.Variable(net\_data["fc6"][0])

fc6b = tf.Variable(net\_data["fc6"][1])

flat5 = tf.reshape(maxpool5, [-1, int(np.prod(maxpool5.get\_shape()[1:]))])

fc6 = tf.nn.relu(tf.matmul(flat5, fc6W) + fc6b)

# fc7, 4096

fc7W = tf.Variable(net\_data["fc7"][0])

fc7b = tf.Variable(net\_data["fc7"][1])

fc7 = tf.nn.relu(tf.matmul(fc6, fc7W) + fc7b)

if feature\_extract:

return fc7

# fc8, 1000

fc8W = tf.Variable(net\_data["fc8"][0])

fc8b = tf.Variable(net\_data["fc8"][1])

logits = tf.matmul(fc7, fc8W) + fc8b

probabilities = tf.nn.softmax(logits)

return probabilities

#### Alexnet迁移学习交通标志分类

'''

Alexnet 的迁移学习例子

'''

import pickle

import tensorflow as tf

from sklearn.model\_selection import train\_test\_split

from alexnet import AlexNet

from sklearn.utils import shuffle

import time

nb\_classes = 43

epochs = 10

batch\_size = 128

# TODO: Load traffic signs data.

with open('./train.p','rb') as f:

data\_read = pickle.load(f)

# TODO: Split data into training and validation sets.

X\_train, X\_val, y\_train, y\_val = train\_test\_split(data\_read['features'], data\_read['labels'], test\_size=0.33,

random\_state=0)

# TODO: Define placeholders and resize operation.

features = tf.placeholder(tf.float32, [None, 32, 32, 3])

labels = tf.placeholder(tf.int64, None)

resized = tf.image.resize\_images(features, (227,227))

# TODO: pass placeholder as first argument to `AlexNet`.

fc7 = AlexNet(resized, feature\_extract=True)

# NOTE: `tf.stop\_gradient` prevents the gradient from flowing backwards

# past this point, keeping the weights before and up to `fc7` frozen.

# This also makes training faster, less work to do!

fc7 = tf.stop\_gradient(fc7)

# TODO: Add the final layer for traffic sign classification.

shape = (fc7.get\_shape().as\_list()[-1], nb\_classes )

fc8w = tf.Variable(tf.truncated\_normal(shape, stddev=1e-2))

fc8b = tf.Variable(tf.zeros(nb\_classes))

logits = tf.nn.xw\_plus\_b(fc7,fc8w,fc8b)

# TODO: Define loss, training, accuracy operations.

# HINT: Look back at your traffic signs project solution, you may

# be able to reuse some the code.

cross\_entropy = tf.nn.sparse\_softmax\_cross\_entropy\_with\_logits(labels=labels, logits=logits)

loss\_op = tf.reduce\_mean(cross\_entropy)

opt = tf.train.AdamOptimizer()

train\_op = opt.minimize(loss\_op, var\_list=[fc8w, fc8b])

init\_op = tf.initialize\_all\_variables()

preds = tf.arg\_max(logits, 1)

accuracy\_op = tf.reduce\_mean(tf.cast(tf.equal(preds, labels), tf.float32))

def eval\_on\_data(X, y, sess):

total\_acc = 0

total\_loss = 0

for offset in range(0, X.shape[0], batch\_size):

end = offset + batch\_size

X\_batch = X[offset:end]

y\_batch = y[offset:end]

loss, acc = sess.run([loss\_op, accuracy\_op], feed\_dict={features: X\_batch, labels:y\_batch})

total\_loss += X\_batch.shape[0] \* loss

total\_acc += X\_batch.shape[0] \* acc

return total\_loss/X.shape[0], total\_acc/X.shape[0]

# TODO: Train and evaluate the feature extraction model.

with tf.Session() as sess:

sess.run(init\_op)

for i in range(epochs):

X\_train, y\_train = shuffle(X\_train, y\_train)

t0 = time.time()

train\_acc = 0

for offset in range(0, X\_train.shape[0], batch\_size):

end = offset + batch\_size

X\_batch = X\_train[offset:end]

y\_batch = y\_train[offset:end]

\_, train\_acc\_ = sess.run([train\_op, accuracy\_op], feed\_dict={features:X\_batch, labels:y\_batch})

train\_acc += train\_acc\_\*batch\_size

val\_loss, val\_acc = eval\_on\_data(X\_val, y\_val, sess)

print('Eopch:', i+1)

print("Time: %.3f" %(time.time() - t0))

print("Train acc:", train\_acc/X\_train.shape[0])

print("Validation loss:", val\_loss)

print("Validation\_acc", val\_acc)

print("")

5、数据分离

**from** sklearn.model\_selection **import** train\_test\_split

X\_train, X\_valid, y\_train, y\_valid = train\_test\_split(X\_train, y\_train, test\_size=0.3, random\_state=42, stratify = y\_train)

## 5、查看数据分布代码收藏

def get\_freq(labels):

count\_labels = Counter(labels)

count\_labels = dict([(k,count\_labels[k]) for k in sorted(count\_labels.keys())])

num\_labels = np.array(list(count\_labels.values()))

freq\_labels = num\_labels/num\_labels.sum().astype('float')

return freq\_labels

## 6、加载cifar数据代码

def load\_batch(filename):

with open(filename,'rb') as f:

dict\_file = pk.load(f,encoding='bytes')

X = dict\_file[b'data']

y = dict\_file[b'labels']

X = X.reshape(X.shape[0],3,32,32).transpose(0,2,3,1)

y = np.array(y)

return X,y

def load\_cifar(root):

train\_X = []

train\_y = []

for i in range(1,6):

f\_name = os.path.join(root, 'data\_batch\_%d' % i)

X, y = load\_batch(f\_name)

train\_X.append(X)

train\_y.append(y)

train\_X = np.concatenate(train\_X,axis=0)

train\_y = np.concatenate(train\_y,axis=0)

test\_X,test\_y = load\_batch(os.path.join(root,'test\_batch'))

return train\_X, train\_y, test\_X, test\_y

train\_X, train\_y, test\_X, test\_y = load\_cifar('cifar-10-batches-py')

## 7、Keras 网络示例

一个github

<https://github.com/keras-team/keras/blob/master/examples/mnist_cnn.py>

一个自己写的代码

# !/user/bin/env python

# -\*- coding:utf-8 -\*-

# author:Parker time: 2018/5/20

import csv

import cv2

import numpy as np

from keras.models import Sequential

from keras.layers import Flatten, Dense, Lambda, Cropping2D, Dropout

from keras.layers.convolutional import Conv2D

from sklearn.model\_selection import train\_test\_split

from sklearn.utils import shuffle

lines = []

# get all lines

with open(r'../newData/driving\_log.csv') as csvfile:

reader = csv.reader(csvfile)

for line in reader:

lines.append(line)

train\_lines, validation\_lines = train\_test\_split(lines, test\_size=0.2)

# get all three images and the steering

def generator(data, batch\_size\_=128):

num\_data = len(data)

while 1:

shuffle(data)

for offset in range(0, num\_data, batch\_size\_):

batch\_data = data[offset: offset+batch\_size\_]

images, angles = [], []

for single\_data in batch\_data:

for i in range(3):

source\_path = single\_data[i]

index = source\_path.find('IMG')

filename = '..\\newData\\' + source\_path[index:].strip()

image = cv2.imread(filename)

image = cv2.cvtColor(image, cv2.COLOR\_BGR2RGB)

angle = float(single\_data[3])

if i == 1:

angle += 0.2

elif i == 2:

angle -= 0.2

images.append(image)

angles.append(angle)

images.append(cv2.flip(image, 1))

angles.append(angle \* -1.0)

X\_ = np.array(images)

y\_ = np.array(angles)

yield shuffle(X\_, y\_)

batch\_size = 64

train\_generator = generator(train\_lines,batch\_size\_=batch\_size)

validation\_generator = generator(validation\_lines, batch\_size\_=batch\_size)

model = Sequential()

model.add(Lambda(lambda x: x/127.5 - 1, input\_shape=(160, 320, 3)))

model.add(Cropping2D(cropping=((70, 25), (0, 0))))

model.add(Conv2D(filters=24, kernel\_size=(5,5), strides=(2,2), activation='relu'))

model.add(Dropout(0.2))

model.add(Conv2D(filters=36, kernel\_size=(5,5), strides=(2,2), activation='relu'))

model.add(Conv2D(filters=48, kernel\_size=(5,5), strides=(2,2), activation='relu'))

model.add(Conv2D(filters=64, kernel\_size=(3,3), activation='relu'))

model.add(Conv2D(filters=64, kernel\_size=(3,3), activation='relu'))

model.add(Flatten())

model.add(Dropout(0.2))

model.add(Dense(100))

model.add(Dense(50))

model.add(Dense(10))

model.add(Dense(1))

model.compile(loss='mse', optimizer='adam')

model.fit\_generator(train\_generator, steps\_per\_epoch=len(train\_lines)/batch\_size,

validation\_data=validation\_generator, validation\_steps=len(validation\_lines)/batch\_size,

epochs=3, verbose=1)

model.save('model.h5')

8、Keras画出loss

from keras.models import Model

import matplotlib.pyplot as plt

history\_object = model.fit\_generator(train\_generator, samples\_per\_epoch =

len(train\_samples), validation\_data =

validation\_generator,

nb\_val\_samples = len(validation\_samples),

nb\_epoch=5, verbose=1)

### print the keys contained in the history object

print(history\_object.history.keys())

### plot the training and validation loss for each epoch

plt.plot(history\_object.history['loss'])

plt.plot(history\_object.history['val\_loss'])

plt.title('model mean squared error loss')

plt.ylabel('mean squared error loss')

plt.xlabel('epoch')

plt.legend(['training set', 'validation set'], loc='upper right')

plt.show()