
西南交通大學
Southwest Jiaotong University
计算机视觉及深度学习
卷积神经网络实验
--基于Tensorflow
教 师：龚 勋
Email：xgong@swjtu.edu.cn

卷积函数

tf.nn.conv2d(input, filter, strides, padding,
use_cudnn_on_gpu=None,name=None)

tf.nn.depthwise_conv2d(input, filter, strides, padding,
name=None)

tf.nn.separable_conv2d(input, depthwise_filter, pointwise_filter,
strides,padding, name=None)

tf.nn.conv2d

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

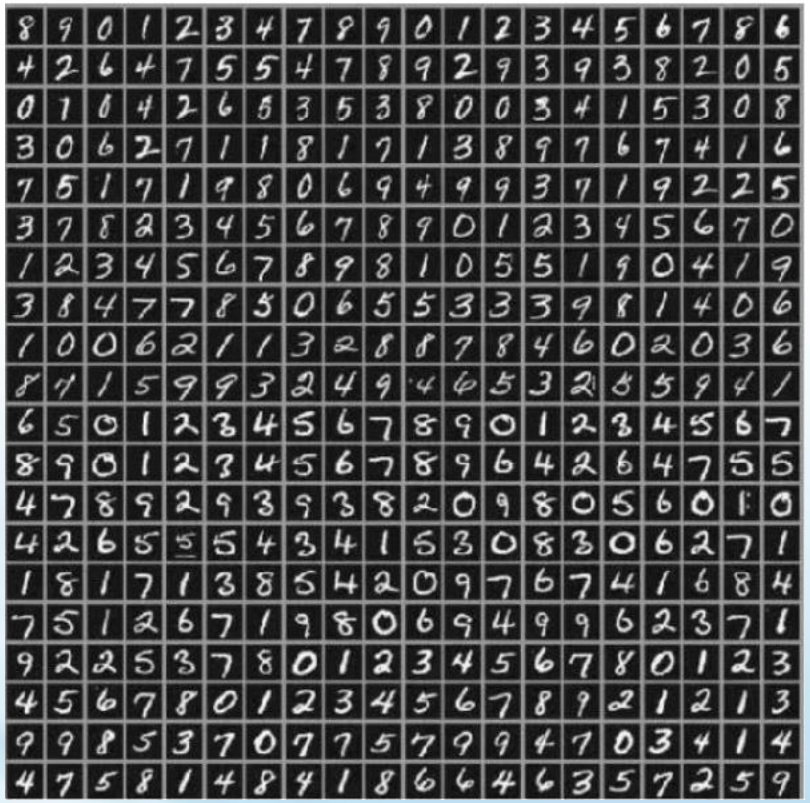
- **input**：需要做卷积的输入数据。注意：这是一个4维的张量（[batch, in_height, in_width, in_channels]），要求类型为float32或float64其中之一。
- **filter**：卷积核。[filter_height, filter_width, in_channels, out_channels]
- **strides**：图像每一维的步长，是一个一维向量，长度为4
- **padding**：定义元素边框与元素内容之间的空间。"SAME"或"VALID"，这个值决定了不同的卷积方式。当为"SAME"时，表示边缘填充，适用于全尺寸操作；当为"VALID"时，表示边缘不填充。
- **use_cudnn_on_gpu**：bool类型，是否使用cudnn加速
- **name**：该操作的名称

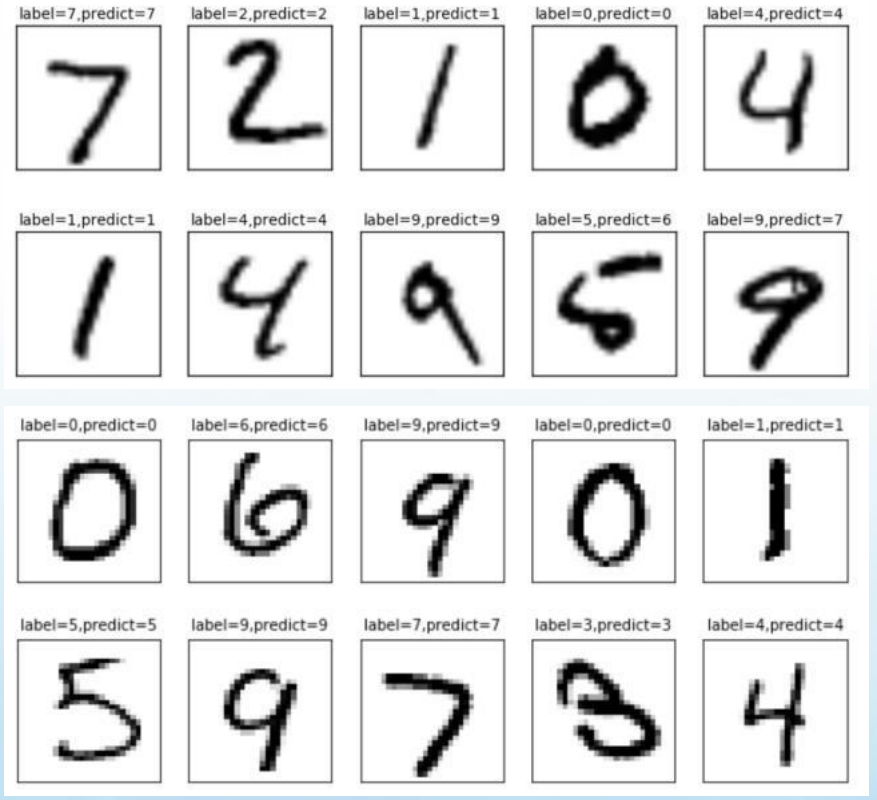
池化函数

最大池化：tf.nn.max_pool(value, ksize, strides, padding, name=None)

平均池化：tf.nn.avg_pool(value, ksize, strides, padding, name=None)

基础实验1：MNIST手写数字识别





MNIST 数据集来自美国国家标准与技术研究所, National Institute of Standards and Technology (NIST).

数据集由来自 250 个不同人手写的数字构成, 其中 50% 是高中学生, 50% 来自人口普查局 (the Census Bureau) 的工作人员

训练集：55000，验证集：5000，测试集：10000

http://yann.lecun.com/exdb/mnist

数据库读取

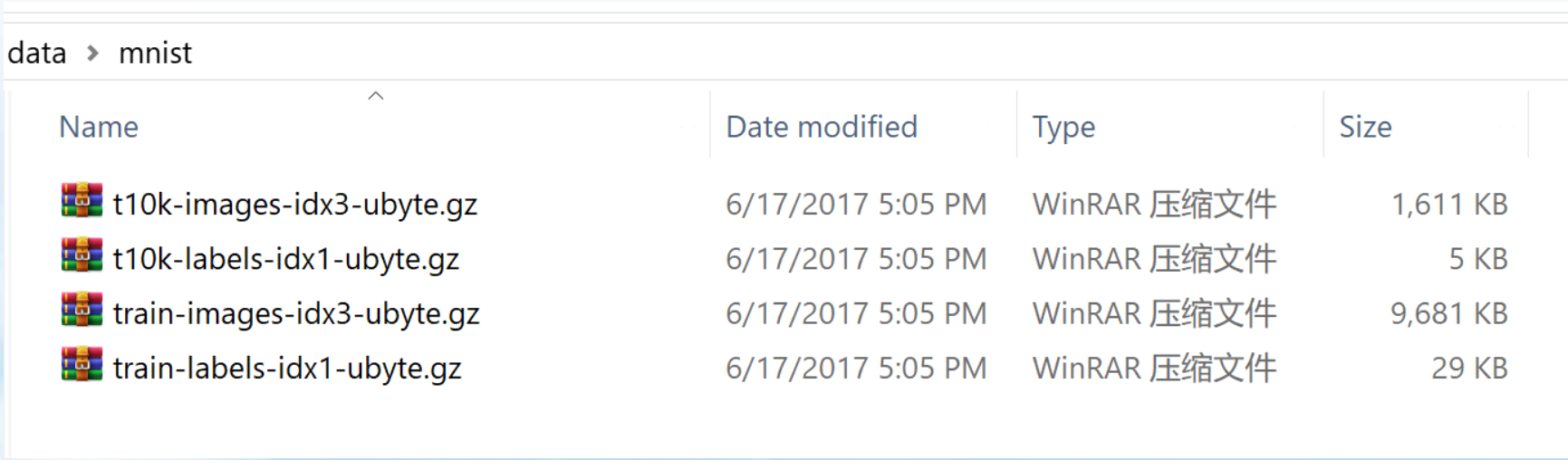
- **MNIST 数据集可在网上获取** http://yann.lecun.com/exdb/mnist/
- **TensorFlow提供了数据集读取方法**

```
import numpy as np
import tensorflow as tf
import tensorflow.examples.tutorials.mnist.input_data as
input_data
mnist = input_data.read_data_sets('./data', one_hot=True)
```

is deprecated and will be removed in a future version.

MNIST手写数字识别数据集

MNIST数据集文件在读取时如果指定目录下不存在，则会自动去下载，需等待一定时间如果已经存在了，则直接读取



MNIST手写数字识别数据集

In [3]:
```
print(mnist.train.num_examples,mnist.validation.num_examples,
      mnist.test.num_examples)
print(mnist.train.images.shape,mnist.train.labels.shape)
print(mnist.validation.images.shape,mnist.validation.labels.shape)
print(mnist.test.images.shape,mnist.test.labels.shape)
```

```
55000 5000 10000
(55000, 784) (55000, 10)
(5000, 784) (5000, 10)
(10000, 784) (10000, 10)
```

训练、验证、测试集数据量

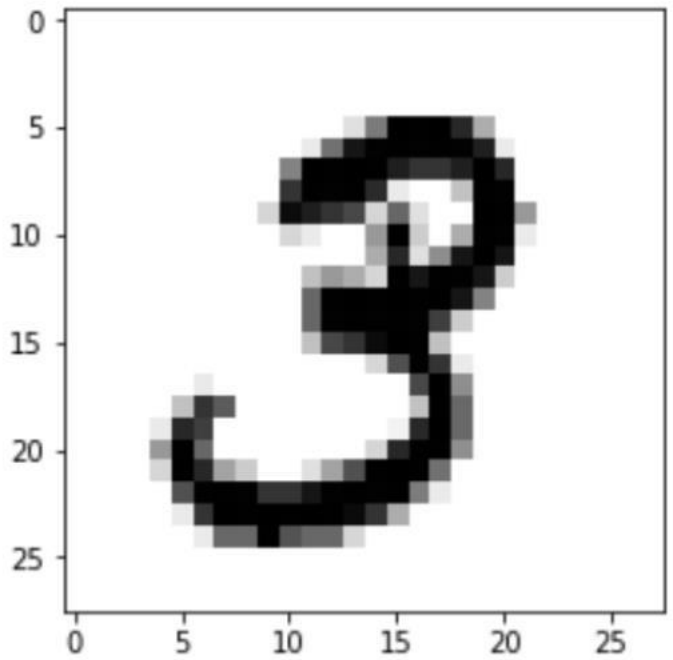
784=28*28，数据维数

10：标签维数

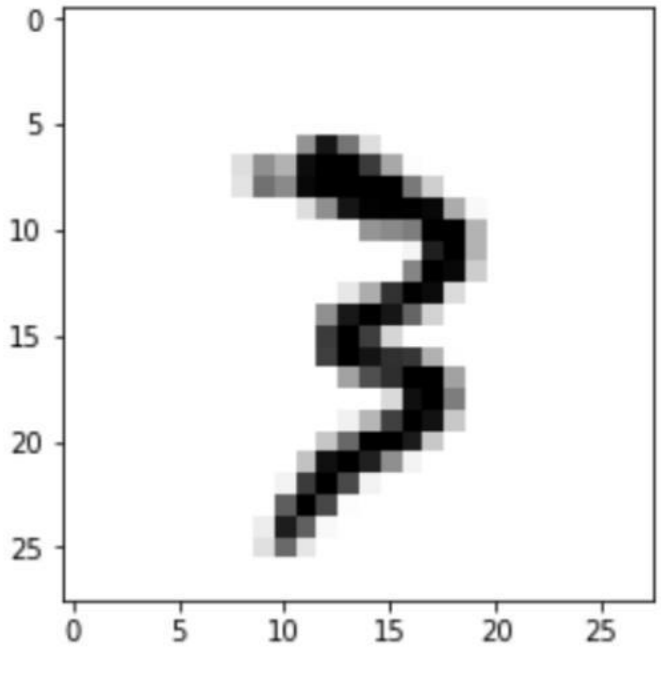
显示图像

```
%matplotlib inline
import matplotlib.pyplot as plt
def plot_image(image):
    plt.imshow(image.reshape(28,28), cmap='binary')
    plt.show()
```

```
plot_image(mnist.train.images[1])
```



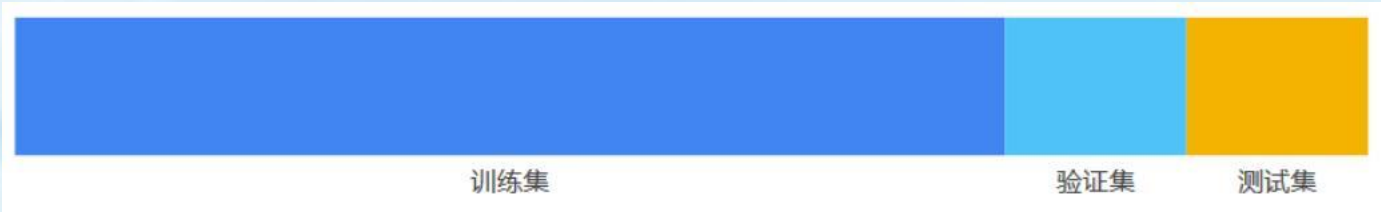

```
plot_image(mnist.train.images[20000])
```



数据集划分

将单个数据集拆分为一个训练集和一个测试集

通过将数据集划分为三个子集，可以大幅降低过拟合的发生几率：



使用验证集评估训练集的效果。

在模型“通过”验证集之后，使用测试集再次检查评估结果

回到MNIST

- 验证集

```
print('image: ' , mnist.validation.images.shape)
print('labels: ' , mnist.validation.labels.shape)
```

```
image:  (5000, 784)
labels:  (5000, 10)
```

- 测试集

```
print('image: ' , mnist.test.images.shape)
print('labels: ' , mnist.test.labels.shape)
```

```
image:  (10000, 784)
labels:  (10000, 10)
```

采用线性模型构建优化方程

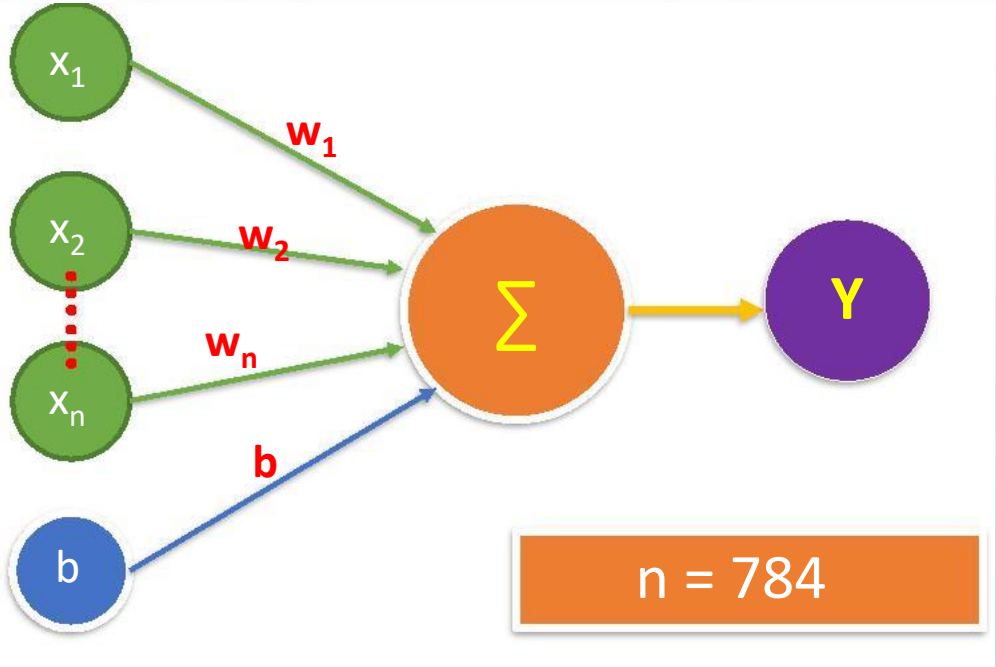
```
# data
x = tf.placeholder(tf.float32, [None, 784], name="X")
y = tf.placeholder(tf.float32, [None, 10], name="Y")

# parameters
W = tf.Variable(tf.random_normal([784, 10]), name="W")
b = tf.Variable(tf.zeros([10]), name="b")
```

模型 softmax分类

#model

forward = tf.matmul(x,W) + b

pred = tf.nn.softmax(forward)



```
sess = tf.Session()
init = tf.global_variables_initializer()
sess.run(init)
```

```
for epoch in range(train_epochs):
    for batch in range(total_batch):
        xs, ys = mnist.train.next_batch(batch_size)
        sess.run(optimizer, feed_dict={x:xs, y:ys})
    loss, acc = sess.run([loss_fun, accuracy],
                         feed_dict={x: mnist.validation.images,
                                    y: mnist.validation.labels})
    if epoch % display_step == 0:
        print("Epoch:%2d" % epoch, "loss:%.7f" % loss, "acc:%.7f" % acc)
print("Done! final acc = %f" % acc)
```

```
Epoch: 0 loss:5.3242607 acc:0.2898000
Epoch: 3 loss:2.1194503 acc:0.6140000
Epoch: 6 loss:1.5092000 acc:0.7084000
Epoch: 9 loss:1.2429380 acc:0.7502000
Epoch:12 loss:1.0902200 acc:0.7758000
Epoch:15 loss:0.9887187 acc:0.7976000
```

训练结果

……

Epoch:129 loss:0.4691718 acc:0.8956000

Epoch:132 loss:0.4666167 acc:0.8966000

Epoch:135 loss:0.4632896 acc:0.8972000

Epoch:138 loss:0.4608540 acc:0.8984000

Epoch:141 loss:0.4579205 acc:0.8984000

Epoch:144 loss:0.4551692 acc:0.8986000

Epoch:147 loss:0.4529768 acc:0.8990000

Done! final acc = 0.898200

修改Mnist识别--卷积神经网络

```
# hiden layer
H1_NN = 6
H2_NN = 12
H3_NN = 24
FC1_NN = 200  # fully connected layer

x_image = tf.reshape(x, [-1, 28, 28, 1])
h_conv1 = conv2d_layer(x_image, [6,6,1,H1_NN], [H1_NN], tf.nn.relu)
h_pool1 = max_pool_2x2(h_conv1)#output is 14*14
h_conv2 = conv2d_layer(h_pool1, [5,5,H1_NN,H2_NN], [H2_NN], tf.nn.relu)
h_pool2 = max_pool_2x2(h_conv2)#output is 7*7
h_conv3 = conv2d_layer(h_pool2, [4,4,H2_NN,H3_NN], [H3_NN], tf.nn.relu)
h_pool3 = max_pool_2x2(h_conv3)#output is 4*4

h_pool3_flat = tf.reshape(h_pool3, [-1, 4 * 4 * H3_NN])
fc1 = fcn_layer(h_pool3_flat, 4 * 4 * H3_NN, FC1_NN, tf.nn.relu)
```

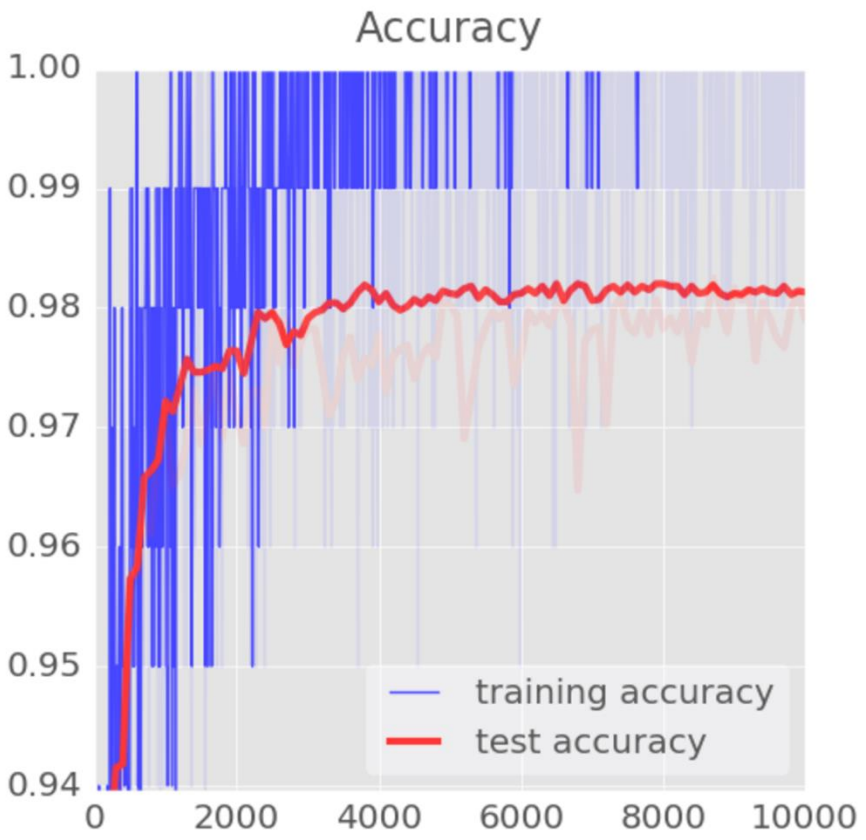
```python
def weight_variable(shape):
  initial = tf.truncated_normal(shape, stddev=0.1)
  return tf.Variable(initial)

def bias_variable(shape):
  initial = tf.constant(0.1, shape=shape)
  return tf.Variable(initial)

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

def max_pool_2x2(x):
  return tf.nn.max_pool(x, ksize=[1, 2, 2, 1],
                        strides=[1, 2, 2, 1], padding='SAME')

def conv2d_layer(inputs,
                 kernel_shape,
                 output_dim,
                 activation=None):
    W = weight_variable(kernel_shape)
    b = bias_variable(output_dim)
    h_conv = conv2d(inputs, W) + b
    if activation is None:
        outputs = h_conv
    else:
        outputs = activation(h_conv)
    return outputs
```

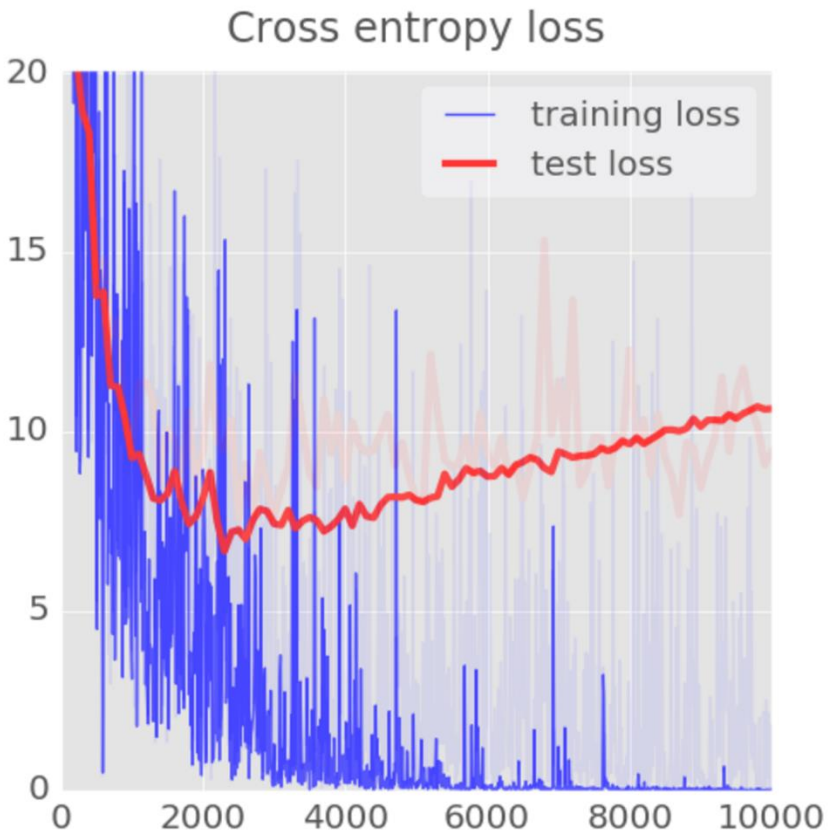
```
Epoch:26 loss:0.1022041 acc:0.9830000
Epoch:29 loss:0.1176529 acc:0.9816000
click to expand output; double click to hide output :0.9800000
Epoch:35 loss:0.1898626 acc:0.9756000
Epoch:38 loss:0.1107501 acc:0.9822000
Epoch:41 loss:0.1163032 acc:0.9820000
Epoch:44 loss:0.1304544 acc:0.9818000
Epoch:47 loss:0.1031366 acc:0.9822000
d:/ckpt_tf/mnist\mnistModel
Done! final acc = 0.980200, time=131.12s
```

```
test_acc = sess.run(accuracy, feed_dict={x: mnis
                                         y: mnis
print("Test Acc=%.5f" % test_acc)
```

```
Test Acc=0.98090
```

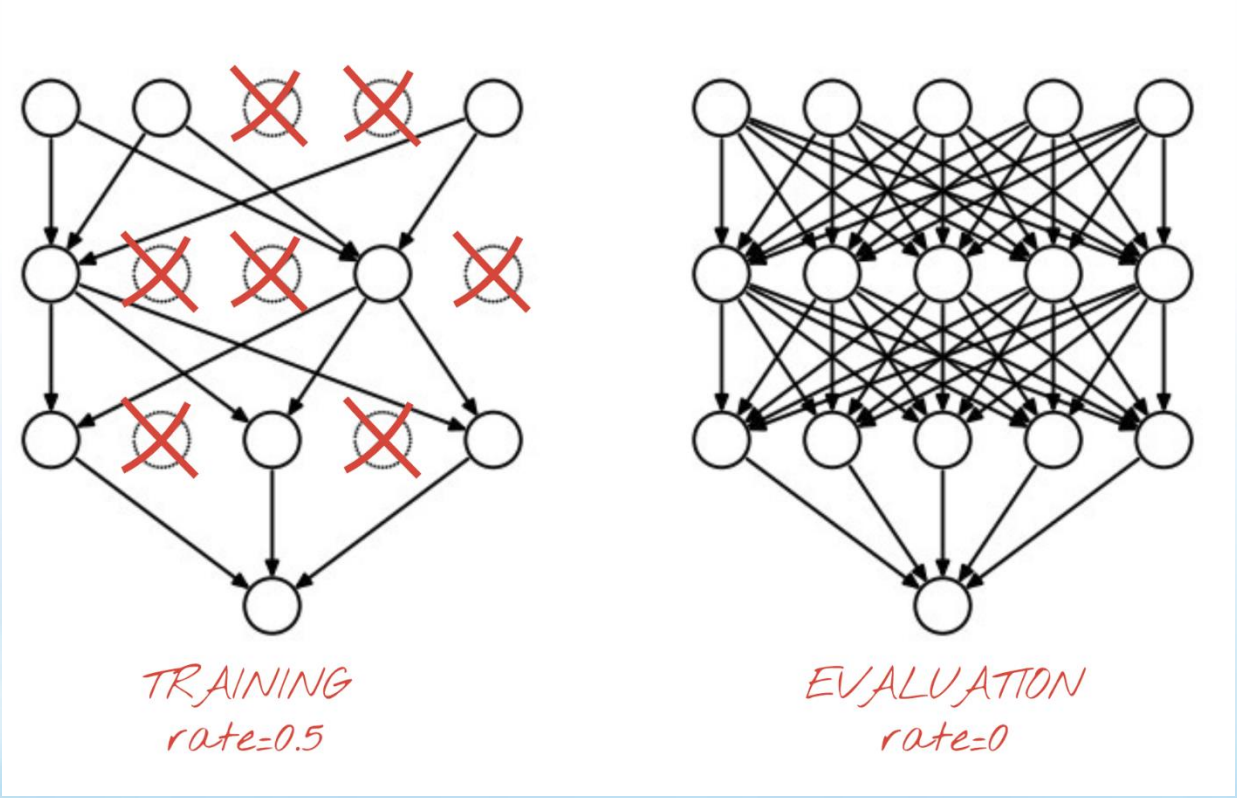
Overfitting





Reference: *https://github.com/GoogleCloudPlatform/tensorflow-without-a-phd*

Dropout




```
pkeep = tf.placeholder(tf.float32)

fc1_dp = tf.nn.dropout(fc1, pkeep)
```

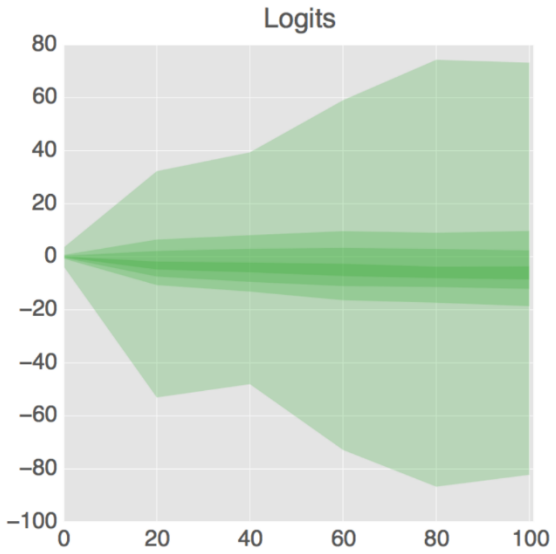
```
Epoch: 2 loss:0.0425114 acc:0.9870000 lr:0.0014147
Epoch: 5 loss:0.0361637 acc:0.9908000 lr:0.0006761
Epoch: 8 loss:0.0350688 acc:0.9920000 lr:0.0003525
Epoch:11 loss:0.0370960 acc:0.9920000 lr:0.0002106
Epoch:14 loss:0.0397607 acc:0.9922000 lr:0.0001485
.
Epoch:35 loss:0.0461600 acc:0.9922000 lr:0.0001002
Epoch:38 loss:0.0485143 acc:0.9918000 lr:0.0001001
Epoch:41 loss:0.0495552 acc:0.9922000 lr:0.0001000
Epoch:44 loss:0.0477311 acc:0.9918000 lr:0.0001000
Epoch:47 loss:0.0508552 acc:0.9916000 lr:0.0001000
d:/ckpt_tf/mnist\mnistModel
Done! final acc = 0.992000, time=137.62s
```

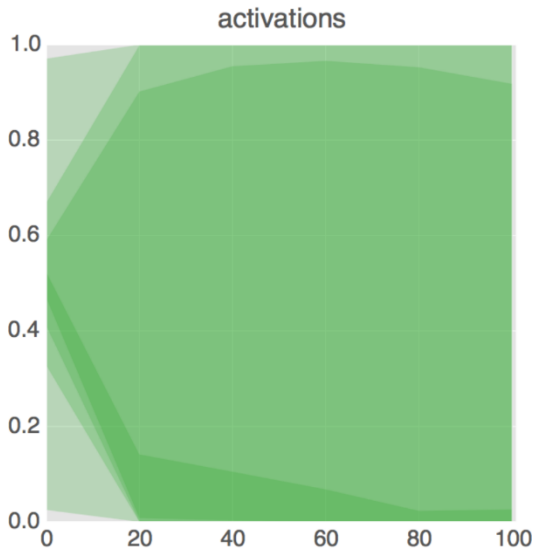
```
test_acc = sess.run(accuracy, feed_dict={x: mnist.test.images,
                                         y: mnist.test.labels,
                                         pkeep: 1.00})
print("Test Acc=%.5f" % test_acc)
```

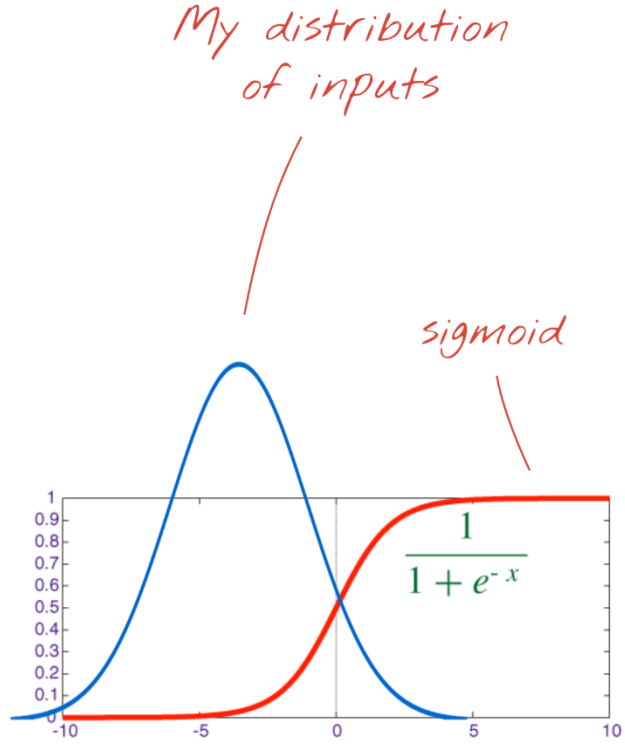
```
Test Acc=0.99170
```

Batch Normal

Without batch normalisation

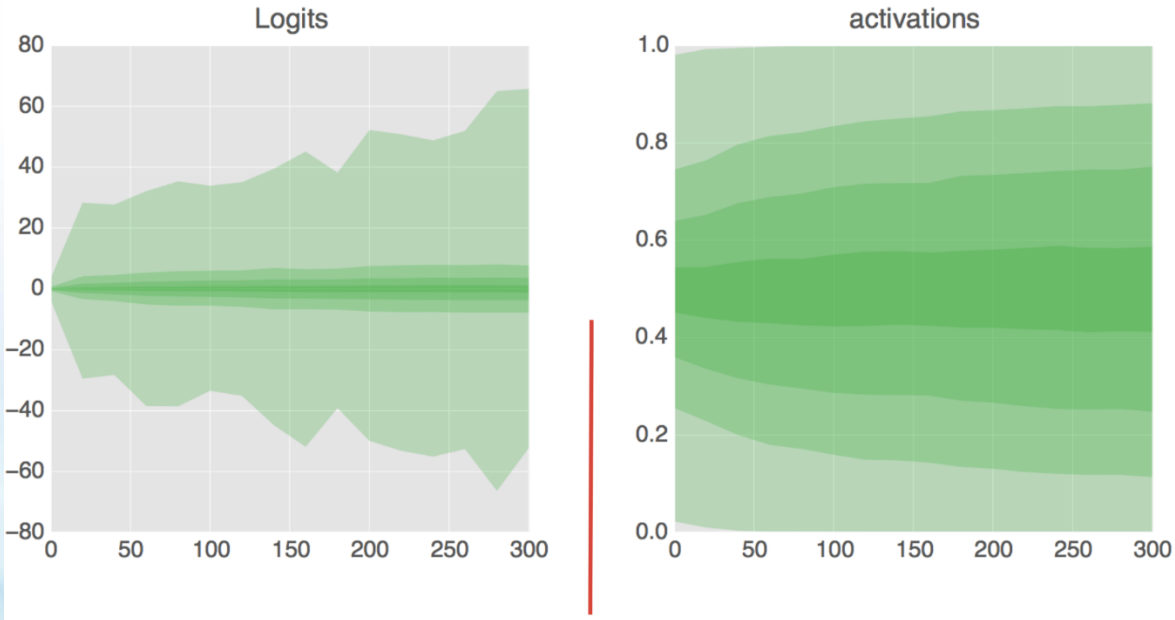


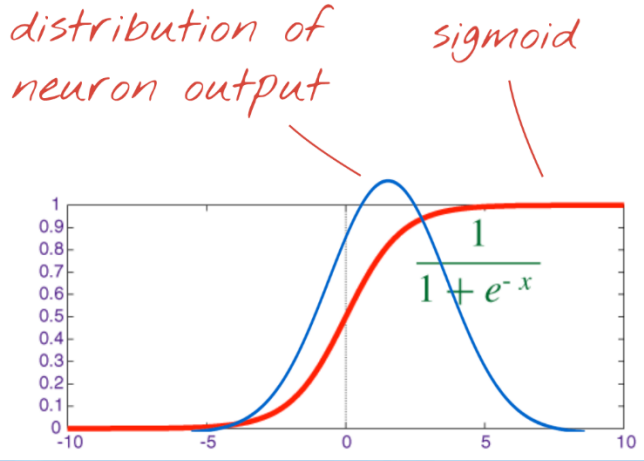




Reference: *https://github.com/GoogleCloudPlatform/tensorflow-without-a-phd*

With batch normalisation (sigmoid)





Reference: *https://github.com/GoogleCloudPlatform/tensorflow-without-a-phd*

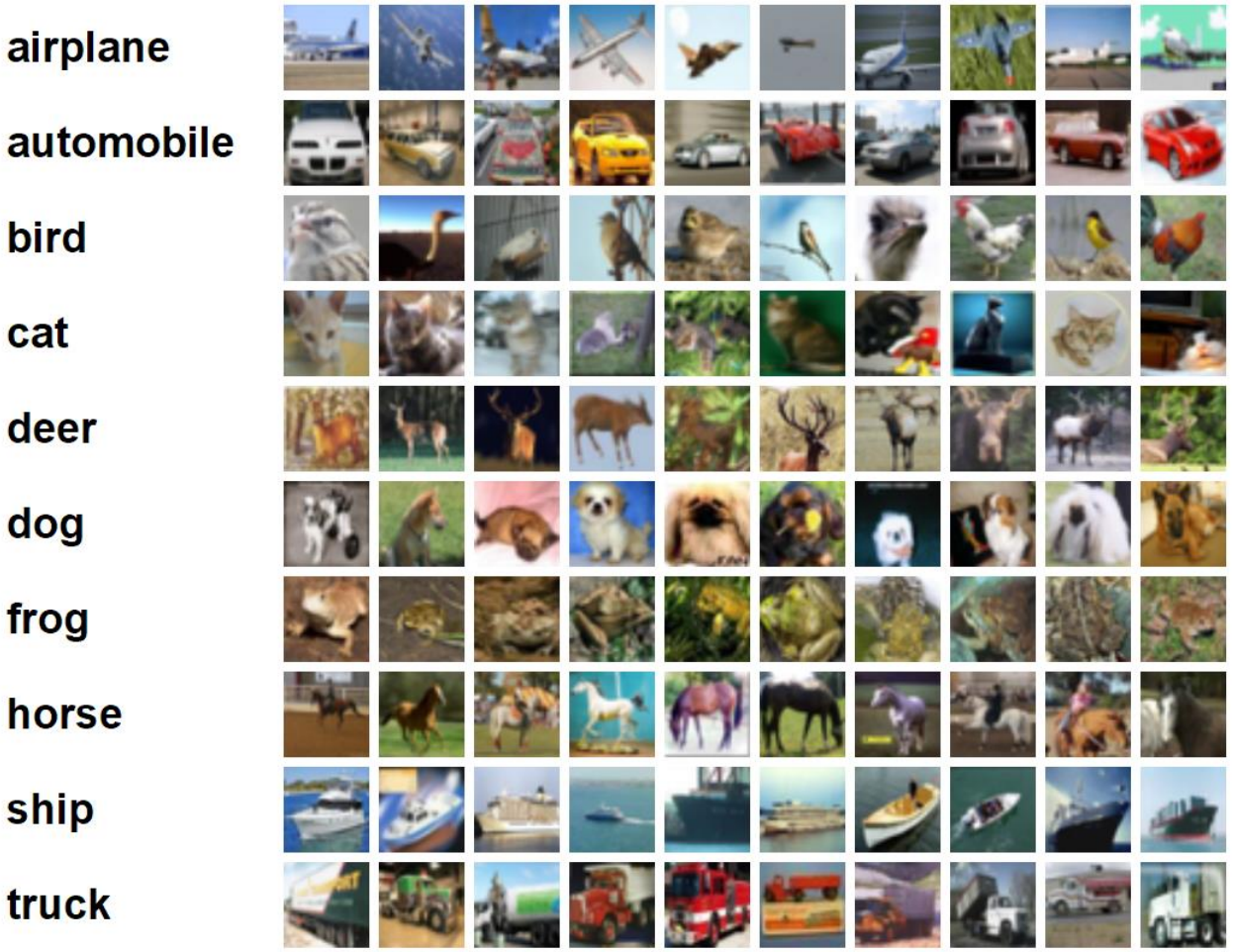
```
def batchnorm(Ylogits, is_test, iteration, offset, convolutional=False):
    exp_moving_avg = tf.train.ExponentialMovingAverage(0.999, iteration) # adding the iter
    bnepsilon = 1e-5
    if convolutional:
        mean, variance = tf.nn.moments(Ylogits, [0, 1, 2])
    else:
        mean, variance = tf.nn.moments(Ylogits, [0])
    update_moving_averages = exp_moving_avg.apply([mean, variance])
    m = tf.cond(is_test, lambda: exp_moving_avg.average(mean), lambda: mean)
    v = tf.cond(is_test, lambda: exp_moving_avg.average(variance), lambda: variance)
    Ybn = tf.nn.batch_normalization(Ylogits, m, v, offset, None, bnepsilon)
    return Ybn, update_moving_averages
```

```
def conv2d_layer(inputs,
                 kernel_shape,
                 output_dim,
                 activation=None,
                 bn=True,
                 iteration=0,
                 if_test=False):
    W = weight_variable(kernel_shape)
    b = bias_variable(output_dim)
    B1 = bias_variable(output_dim)
    h_conv = conv2d(inputs, W) + b
    update_ema = 1.0
    if bn == True:
        h_convBN, update_ema = batchnorm(h_conv, if_test, iteration, B1, convolutional=True)
    else:
        h_convBN = h_conv

    if activation is None:
        outputs = h_convBN
    else:
        outputs = activation(h_convBN)
    return outputs, update_ema
```

基础实验2： Cifar10分

The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.

http://www.cs.toronto.edu/~kriz/cifar.html



Name	Size
batches.meta	1 KB
data_batch_1	30,309 KB
data_batch_2	30,308 KB
data_batch_3	30,309 KB
data_batch_4	30,309 KB
data_batch_5	30,309 KB
readme.html	1 KB
test_batch	30,309 KB

cifar-10-batches-py

cifar-10-binary.tar.gz

cifar-10-python.tar.gz

数据处理

```
# 定义全局变量
tf.app.flags.DEFINE_string('dbpath',‘.\cifar-10-batches-py','')
tf.app.flags.DEFINE_string('ckp_log_path', ‘./cifar10', '')
FLAGS = tf.app.flags.FLAGS

def checkDb():
    if not os.path.exists(FLAGS.dbpath):
        print("DB does not exist!")
        exit(0)
    else:
        print(FLAGS.dbpath)
```

读取数据

```
def load_CIFAR_data(dbpath):
    images_train = []
    labels_train = []
    for i in range(5):
        fn = os.path.join(dbpath, 'data_batch_%d' % (i+1))
        print(fn)
        cur_img, cur_label = load_CIFAR_batch(fn)
        images_train.append(cur_img)
        labels_train.append(cur_label)
        Xtrain = np.concatenate(images_train)
        Ytrain = np.concatenate(labels_train)
    Xtest, Ytest = load_CIFAR_batch(os.path.join(dbpath, 'test_batch'))
    return Xtrain,Ytrain, Xtest, Ytest
```

读取一个文件

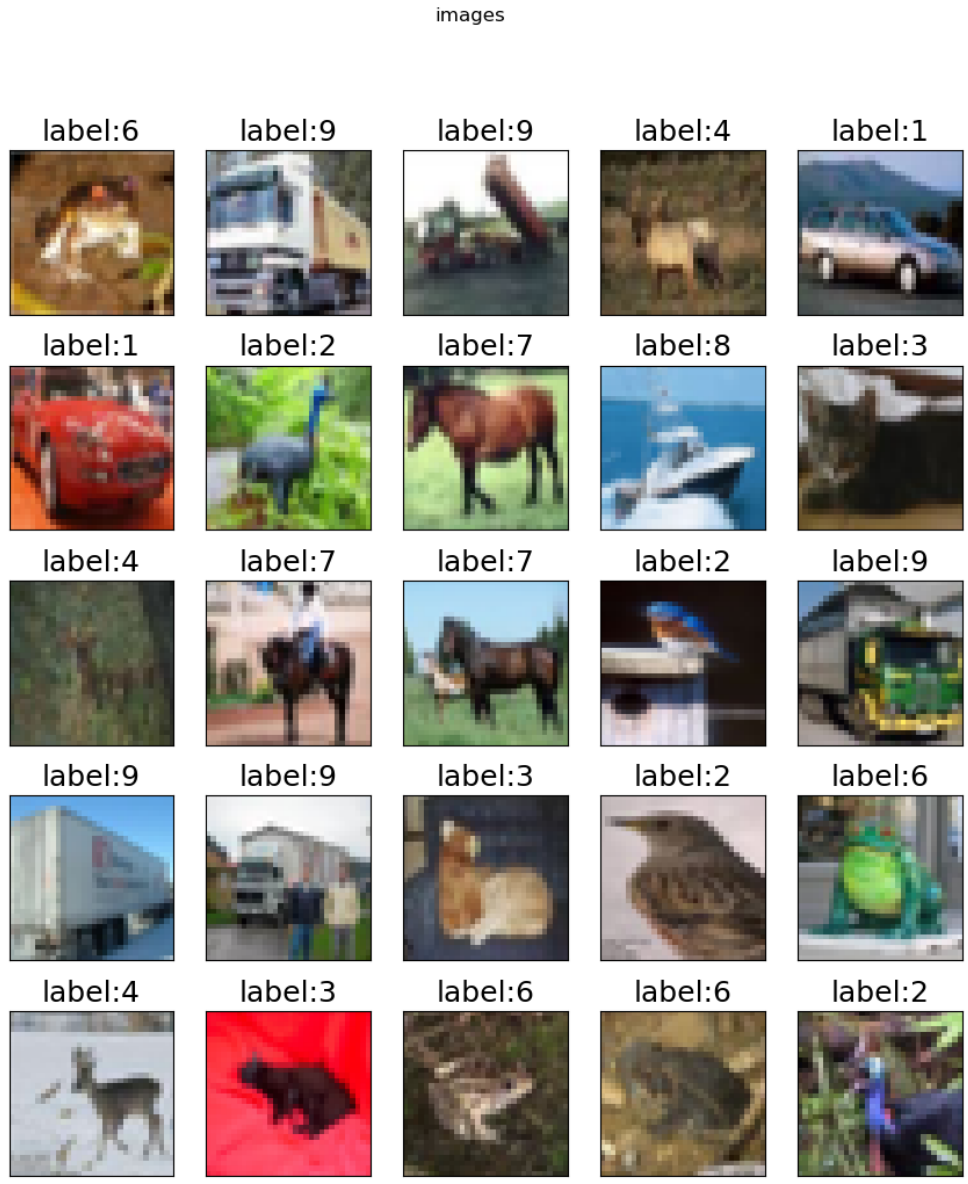
```
import pickle as p

def load_CIFAR_batch(filename):
    with open(filename,'rb') as f:
        # image size 32*32*3
        data_dict = p.load(f, encoding='bytes')
        images = data_dict[b'data']
        labels = data_dict[b'labels']
        # original shape order: BCWH
        images = images.reshape(10000, 3, 32, 32)
        # covert to BWHC for TF
        images = images.transpose(0,2,3,1)
        labels = np.array(labels)
        return images, labels
```

主流程

```
if __name__ == '__main__':
    # 1 load data
    checkDb()
    Xtrain,Ytrain, Xtest, Ytest = load_CIFAR_data(FLAGS.dbpath)
    print(Xtrain.shape, Ytrain.shape)
    print(Xtest.shape, Ytest.shape)
    plot_images_labels(Xtrain, Ytrain, 0, 25, "images")
    # 2 lable to OneHot
    Ytrain_onehot = toOneHotCode(Ytrain)
    Ytest_onehot = toOneHotCode(Ytest)
    # 3 normalization
    XtrainNorm = Xtrain.astype("float32") / 255.0
    XtestNorm = Xtest.astype("float32") / 255.0
    # 4 train
    train(XtrainNorm, Ytrain_onehot, XtestNorm, Ytest_onehot)
```

显示图像

```
def plot_images_labels(images,labels,
                       start, num=10,
                       caption=""):
    fig = plt.gcf()
    fig.set_size_inches(20,22)
    if num > 25:
        num = 25
    for i in range(0, num):
        ax = plt.subplot(5,5,i+1)
        ax.imshow(images[start], cmap='bi
        title = 'label:' + str(labels[sta
        ax.set_title(title, fontsize=18)
        ax.set_xticks([])
        ax.set_yticks([])
        start += 1

    plt.suptitle(caption)



    plt.show()
```



数据预处理

```
# One-Hot-Code
def toOneHotCode(ori):
    encoder = OneHotEncoder(sparse=False)
    sample = [[0],[1],[2],[3],[4],[5],[6],[7],[8],[9]]
    encoder.fit(sample)
    ori_re = ori.reshape(-1,1)
    one = encoder.transform(ori_re)
    return one
```

```
# Normalization
XtrainNorm = Xtrain.astype("float32") / 255.0
XtestNorm = Xtest.astype("float32") / 255.0
```

```python
def net(input, pkeep):
    ##############################model##############################
    # hiden layer
    H1_NN = 16;H2_NN = 32;H3_NN = 64;FC1_NN = 128

    h_conv1 = conv2d_layer(input, [3, 3, 3, H1_NN], [H1_NN], tf.nn.relu)
    h_pool1 = max_pool_2x2(h_conv1)  # output is 14*14
    h_conv2 = conv2d_layer(h_pool1, [3, 3, H1_NN, H2_NN], [H2_NN], tf.nn.relu)
    h_pool2 = max_pool_2x2(h_conv2)  # output is 7*7
    h_conv3 = conv2d_layer(h_pool2, [3, 3, H2_NN, H3_NN], [H3_NN], tf.nn.relu)
    h_pool3 = max_pool_2x2(h_conv3)  # output is 4*4

    h_pool3_flat = tf.reshape(h_pool3, [-1, 4 * 4 * H3_NN])
    fc1 = fcn_layer(h_pool3_flat, 4 * 4 * H3_NN, FC1_NN, tf.nn.relu)
    fc1_dp = tf.nn.dropout(fc1, pkeep)  ###

    forward = fcn_layer(fc1_dp, FC1_NN, 10, None)
    pred = tf.nn.softmax(forward)
    return  forward, pred
```

训练

```
for epoch in range(train_epochs):
    # shuffle at the beginning of each epoch
    Xtrain, Ytrain = shuffle(Xtrain, Ytrain)
    for batch in range(total_batch):
        xs, ys, idx_in_epoch = next_batch(Xtrain, Ytrain, batch_size, idx_in_epoch)
        _, loss, lr = sess.run([optimizer,loss_fun, learning_rate],
                               feed_dict={x: xs, y: ys, pkeep: 0.75, step: iteration})
        iteration += 1
    # validation
    acc = sess.run(accuracy, feed_dict={x: Xtest,
                                        y: Ytest,
                                        pkeep: 1.0, step: iteration})
    if (epoch + 1) % display_step == 0:
        print("Epoch:%2d" % epoch, "loss:{:.7f}".format(loss),
        "lr:%.7f" % lr, "acc:{:.7f}".format(acc))
    if (epoch + 1) % save_step == 0:
        fn = os.path.join(FLAGS.ckp_log_path, 'cifar10Model')
        print(fn)
        saver.save(sess, fn, global_step=(epoch + 1))
print("Done! final acc = %f, time=%.2fs" % (acc, time() - start))
```

next_batch

参考mnist.py

```
def next_batch(X, Y, batch_size, _index_in_epoch=0, shuffle=True):
    start = _index_in_epoch;    _num_examples = X.shape[0]
    if start + batch_size > _num_examples:
        …
    else:
        _index_in_epoch += batch_size
        end = _index_in_epoch
        return X[start:end], Y[start:end], _index_in_epoch
```

```
Epoch: 9 loss:0.5913961 lr:0.0007292 acc:0.7131000
Epoch:10 loss:0.7697335 lr:0.0006382 acc:0.7150000
Epoch:11 loss:0.6140158 lr:0.0005604 acc:0.7122000
Epoch:12 loss:0.3961719 lr:0.0004938 acc:0.7173000
Epoch:13 loss:0.7391727 lr:0.0004369 acc:0.7189000
Epoch:14 loss:0.3909289 lr:0.0003881 acc:0.7154000
Epoch:15 loss:0.4020708 lr:0.0003465 acc:0.7185000
Epoch:16 loss:0.3959954 lr:0.0003108 acc:0.7217000
Epoch:17 loss:0.4575052 lr:0.0002803 acc:0.7191000
Epoch:18 loss:0.2468506 lr:0.0002543 acc:0.7202000
Epoch:19 loss:0.6702175 lr:0.0002320 acc:0.7199000
Epoch:20 loss:0.1293726 lr:0.0002129 acc:0.7217000
Epoch:21 loss:0.3751268 lr:0.0001966 acc:0.7232000
```

Go Deep

```
def net(x, y, pkeep):
    ################################model#################################
    # hiden layer
    H1_NN = 16
    H2_NN = 32
    H3_NN = 64
    H4_NN = 128
    FC1_NN = 256  # fully connected layer

    h_conv1 = conv2d_layer(x, [3, 3, 3, H1_NN], [H1_NN], tf.nn.relu)
    h_pool1 = max_pool_2x2(h_conv1)  # output is 14*14
    h_conv2 = conv2d_layer(h_pool1, [3, 3, H1_NN, H2_NN], [H2_NN], tf.nn.relu)
    h_pool2 = max_pool_2x2(h_conv2)  # output is 7*7
    h_conv3 = conv2d_layer(h_pool2, [3, 3, H2_NN, H3_NN], [H3_NN], tf.nn.relu)
    h_pool3 = max_pool_2x2(h_conv3)  # output is 4*4
    print('h_pool3 shape: ' + str(h_pool3.shape))
    h_conv4 = conv2d_layer(h_pool3, [3, 3, H3_NN, H4_NN], [H4_NN], tf.nn.relu)

    h_pool4_flat = tf.reshape(h_conv4, [-1, 4 * 4 * H4_NN])
    fc1 = fcn_layer(h_pool4_flat, 4 * 4 * H4_NN, FC1_NN, tf.nn.relu)
    fc1_dp = tf.nn.dropout(fc1, pkeep)  ###

    forward = fcn_layer(fc1_dp, FC1_NN, 10, None)
    pred = tf.nn.softmax(forward)
    return  forward, pred
```

练习

- 将Cifar10数据读取及操作（next_batch）封装成一个类进行调用
- 增加网络深度，采用ResNet50进行网络修改，将准确率提升（>85%）

西南交通大學
Southwest Jiaotong University

Thank You !