最基本的神经网络代码

sess = tf.InteractiveSession()

//定义输入，以及输出占位符

x = tf.placeholder(tf.float32,shape=[**None**,784],name=**'x'**)  
y\_ = tf.placeholder(tf.int64,shape=[**None**,],name=**'y'**)

//定义神经网络层

network = tl.layers.InputLayer(x)

network = tl.layers.Denselayer(network, n\_units=800, act

=tf.nn.relu,name='relu1')

network = tl.layers.Denselayer(network,n\_units=10,act=tf.identity

,name='output')

//定义cost,以及优化方法

y=network.outputs

cost = tl.cost. cross\_entropy(y,y\_)

train\_op=tf.train.AdaOptimizer().minimize(cost)

//模型初始化

tl.layers.initialize\_global\_variables(sess)

//开始训练模型

tl.utils.fit(sess,network, train\_op,cost,X\_train,y\_train,x,y\_)

//测试模型

tl.utils.test(sess,network,X\_test,y\_test,x,y\_)

//关闭图

sess.close()

//CNN模型

network = tl.layers.InputLayer(x)

network = tl.layers.Conv2d(network, n\_filter=32, filter\_size=(5, 5), strides=(1, 1),act=tf.nn.relu, padding=**'SAME'**, name=**'cnn1'**)  
network = tl.layers.MaxPool2d(network, filter\_size=(2, 2), strides=(2, 2), padding=**'SAME'**, name=**'pool1'**)

network = tl.layers.FlattenLayer(network, name=**'flatten'**)

network = tl.layers.DenseLayer(network, n\_units=10,  
 act = tf.identity,  
 name=**'output'**)

// Denoising Autoencoder

network = tl.layers.InputLayer(x, name=**'input'**)  
network = tl.layers.DropoutLayer(network, keep=0.5, name=**'denoising1'**) *# if drop some inputs, it is denoise AE*network = tl.layers.DenseLayer(network, n\_units=196,  
 act = tf.nn.relu, name=**'relu1'**)  
recon\_layer1 = tl.layers.ReconLayer(network, x\_recon=x, n\_units=784,  
 act = tf.nn.softplus, name=**'recon\_layer1'**)

#Reconlayer里已经定义了cost以及AdamOptimizer().minimize()

recon\_layer1.pretrain(sess, x=x, X\_train=X\_train, X\_val=X\_val,  
 denoise\_name=**'denoising1'**, n\_epoch=200,  
 batch\_size=128, print\_freq=10, save=**True**,  
 save\_name=**'w1pre\_'**)

tfrecord 序列化（将图片转化为string）

TensorFlow-Slim（类似于tensorlayer）

TF-Slim is a lightweight library for defining, training and evaluating complex models in TensorFlow

import tensorflow.contrib.slim as slim

//arg\_scope界定了参数范围以及作用域

with slim.arg\_scope([slim.conv2d, slim.fully\_connected],

activation\_fn=tf.nn.relu,

weights\_initializer=tf.truncated\_normal\_initializer(stddev=0.01),

weights\_regularizer=slim.l2\_regularizer(0.0005)):

with slim.arg\_scope([slim.conv2d], stride=1, padding='SAME'):

net = slim.conv2d(inputs, 64, [11, 11], 4, padding='VALID', scope='conv1')

net = slim.conv2d(net, 256, [5, 5],

weights\_initializer=tf.truncated\_normal\_initializer(stddev=0.03),

scope='conv2')

net = slim.fully\_connected(net, 1000, activation\_fn=None, scope='fc')

Training Tensorflow models requires a model, a loss function, the gradient computation and a training routine that iteratively computes the gradients of the model weights relative to the loss and updates the weights accordingly. TF-Slim provides both common loss functions and a set of helper functions that run the training and evaluation routines.

Consider the case where we have a pre-trained VGG16 model. The model was trained on the ImageNet dataset, which has 1000 classes. However, we would like to apply it to the Pascal VOC dataset which has only 20 classes. To do so, we can initialize our new model using the values of the pre-trained model excluding the final layer:

# Load the Pascal VOC data

image, label = MyPascalVocDataLoader(...)

images, labels = tf.train.batch([image, label], batch\_size=32)

# Create the model

predictions = vgg.vgg\_16(images)

train\_op = slim.learning.create\_train\_op(...)

# Specify where the Model, trained on ImageNet, was saved.

model\_path = '/path/to/pre\_trained\_on\_imagenet.checkpoint'

# Specify where the new model will live:

log\_dir = '/path/to/my\_pascal\_model\_dir/'

# Restore only the convolutional layers:

variables\_to\_restore = slim.get\_variables\_to\_restore(exclude=['fc6', 'fc7', 'fc8'])

init\_fn = assign\_from\_checkpoint\_fn(model\_path, variables\_to\_restore)

# Start training.

slim.learning.train(train\_op, log\_dir, init\_fn=init\_fn)