原 TensorFlow人工智能引擎入门教程之六 训练的模型Model 保存 文件 并使用

基陆伯 图书馆
11495 馆藏 33889

上面好几章讲的是 其实CNN 训练不仅仅是图片 声音 文字 其他都是可以的，我从头到尾都没说过CNN只能用来训练图像。只是CNN在图像识别上效果非常显著 非常好。所以我就把CNN讲了好几章，后面讲讲DNN RNN/LSTM RBM等等 看空闲时间，这一章我们讲讲tensorflow训练的模型怎么系列化参数网络到模型文件里面，并使用模型数据来预测 分类数据。

在tensorflow中保存 模型 恢复模型的 类是tf.train.Saver() 默认 是所有的变量

当不传参数 默认就是所有的变量variable

```
tf.train.Saver.__init__(var_list=None, reshape=False, sharded=False,
max_to_keep=5, keep_checkpoint_every_n_hours=10000.0, name=None,
restore_sequentially=False, saver_def=None, builder=None)
```

Creates a Saver.

The constructor adds ops to save and restore variables.

`var_list` specifies the variables that will be saved and restored. It can be passed as a `dict` or a list:

- A `dict` of names to variables: The keys are the names that will be used to save or restore the variables in the checkpoint files.
- A list of variables: The variables will be keyed with their op name in the checkpoint files.

For example:

```
v1 = tf.Variable(..., name='v1')
v2 = tf.Variable(..., name='v2')

# Pass the variables as a dict:
saver = tf.train.Saver({'v1': v1, 'v2': v2})

# Or pass them as a list.
saver = tf.train.Saver([v1, v2])
# Passing a list is equivalent to passing a dict with the variable op names
# as keys:
saver = tf.train.Saver({v.op.name: v for v in [v1, v2]})
```

The optional `reshape` argument, if `True`, allows restoring a variable from a save file where the variab[...] different shape, but the same number of elements and type. This is useful if you have reshaped a variable and want to reload it from an older checkpoint.

保存模型

save(sess,save_path,...)

从文件中恢复模型

restore(sess,save_path,...)

```
save_path = saver.save(sess, "/root/alexnet.tfmodel")
```

保存

```
saver.restore(sess, "/root/alexnet.tfmodel")
```

恢复

此时 restore 恢复的是sess 这个 状态 ，你可以吧他看做时光机，把sess 的参数 恢复到训练结束时的参数 这时候的sess

就已经可以用来进行预测

```
input_x = .... predictions = sess.run(model, feed_dict={x: input_x})
```

下面来修改alexnet 网络

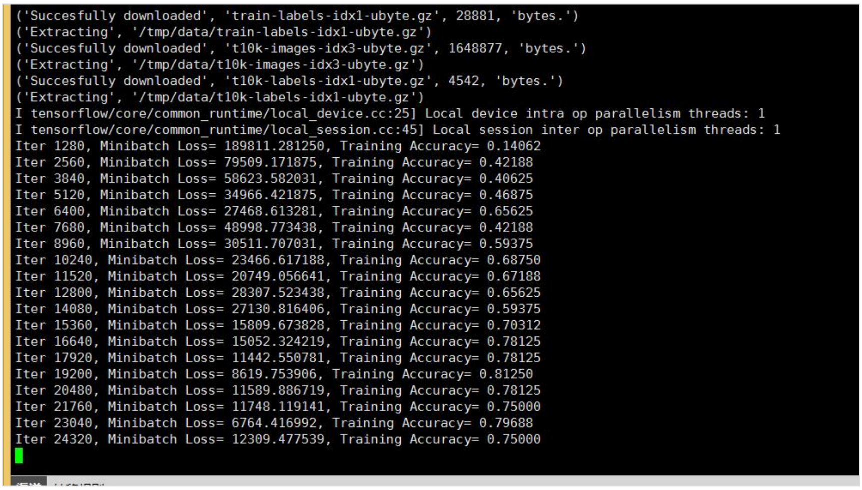
```
# Import MINST data import input_data mnist = input_data.read_data_sets("/tmp/data/", one_h
ot=True) import tensorflow as tf # Parameters learning_rate = 0.001 training_iters = 200000
batch_size = 64 display_step = 20 # Network Parameters n_input = 784 # MNIST data input (im
g shape: 28*28) n_classes = 10 # MNIST total classes (0-9 digits) dropout = 0.8 # Dropou
t, probability to keep units # tf Graph input x = tf.placeholder(tf.float32, [None, n_inpu
t]) y = tf.placeholder(tf.float32, [None, n_classes]) keep_prob = tf.placeholder(tf.float3
2) # dropout (keep probability) # Create custom model def conv2d(name, l_input, w, b):
    return tf.nn.relu(tf.nn.bias_add(tf.nn.conv2d(l_input, w, strides=[1, 1, 1, 1], paddin
g='SAME'),b), name=name) def max_pool(name, l_input, k):     return tf.nn.max_pool(l_inpu
t, ksize=[1, k, k, 1], strides=[1, k, k, 1], padding='SAME', name=name) def norm(name, l_in
put, lsize=4):     return tf.nn.lrn(l_input, lsize, bias=1.0, alpha=0.001 / 9.0, beta=0.7
5, name=name) def customnet(_X, _weights, _biases, _dropout):     # Reshape input picture
    _X = tf.reshape(_X, shape=[-1, 28, 28, 1])     # Convolution Layer     conv1 = conv2
d('conv1', _X, _weights['wc1'], _biases['bc1'])     # Max Pooling (down-sampling)     pool
1 = max_pool('pool1', conv1, k=2)     # Apply Normalization     norm1 = norm('norm1', pool
1, lsize=4)     # Apply Dropout     norm1 = tf.nn.dropout(norm1, _dropout)     #conv1 imag
e show     tf.image_summary("conv1", conv1)     # Convolution Layer     conv2 = conv2d('con
v2', norm1, _weights['wc2'], _biases['bc2'])     # Max Pooling (down-sampling)     pool
2 = max_pool('pool2', conv2, k=2)     # Apply Normalization     norm2 = norm('norm2', pool
2, lsize=4)     # Apply Dropout     norm2 = tf.nn.dropout(norm2, _dropout)     # Convolutio
n Layer     conv3 = conv2d('conv3', norm2, _weights['wc3'], _biases['bc3'])     # Max Pooli
ng (down-sampling)     pool3 = max_pool('pool3', conv3, k=2)     # Apply Normalization
    norm3 = norm('norm3', pool3, lsize=4)     # Apply Dropout     norm3 = tf.nn.dropout(nor
m3, _dropout)     #conv4     conv4 = conv2d('conv4', norm3, _weights['wc4'], _biases['bc
4'])     # Max Pooling (down-sampling)     pool4 = max_pool('pool4', conv4, k=2)     # Appl
y Normalization     norm4 = norm('norm4', pool4, lsize=4)     # Apply Dropout     norm4 = t
f.nn.dropout(norm4, _dropout)     # Fully connected layer     dense1 = tf.reshape(norm
4, [-1, _weights['wd1'].get_shape().as_list()[0]]) # Reshape conv3 output to fit dense laye
r input     dense1 = tf.nn.relu(tf.matmul(dense1, _weights['wd1']) + _biases['bd1'], nam
e='fc1') # Relu activation     dense2 = tf.nn.relu(tf.matmul(dense1, _weights['wd2']) + _bi
ases['bd2'], name='fc2') # Relu activation     # Output, class prediction     out = tf.matm
ul(dense2, _weights['out']) + _biases['out']     return out # Store layers weight & bias we
ights = {     'wc1': tf.Variable(tf.random_normal([3, 3, 1, 64])),     'wc2': tf.Variable(t
f.random_normal([3, 3, 64, 128])),     'wc3': tf.Variable(tf.random_normal([3, 3, 128, 25
6])),     'wc4': tf.Variable(tf.random_normal([2, 2, 256, 512])),     'wd1': tf.Variable(t
f.random_normal([2*2*512, 1024])),     'wd2': tf.Variable(tf.random_normal([1024, 1024])),
    'out': tf.Variable(tf.random_normal([1024, 10])) } biases = {     'bc1': tf.Variable(t
f.random_normal([64])),     'bc2': tf.Variable(tf.random_normal([128])),     'bc3': tf.Vari
able(tf.random_normal([256])),     'bc4': tf.Variable(tf.random_normal([512])),     'bd
1': tf.Variable(tf.random_normal([1024])),     'bd2': tf.Variable(tf.random_normal([102
4])),     'out': tf.Variable(tf.random_normal([n_classes])) } # Construct model pred = cust
omnet(x, weights, biases, keep_prob) # Define loss and optimizer cost = tf.reduce_mean(tf.n
n.softmax_cross_entropy_with_logits(pred, y)) optimizer = tf.train.AdamOptimizer(learning_r
ate=learning_rate).minimize(cost) # Evaluate model correct_pred = tf.equal(tf.argmax(pre
d,1), tf.argmax(y,1)) accuracy = tf.reduce_mean(tf.cast(correct_pred, tf.float32)) # Initia
lizing the variables init = tf.initialize_all_variables() #  tf.scalar_summary("loss", cos
t) tf.scalar_summary("accuracy", accuracy) # Merge all summaries to a single operator merge
d_summary_op = tf.merge_all_summaries() saver = tf.train.Saver() # Launch the graph with t
f.Session() as sess:     sess.run(init)     summary_writer = tf.train.SummaryWriter('/tmp/l
ogs', graph_def=sess.graph_def)     step = 1     # Keep training until reach max iterations
    while step * batch_size < training_iters:         batch_xs, batch_ys = mnist.train.nex
t_batch(batch_size)         # Fit training using batch data         sess.run(optimizer, fee
d_dict={x: batch_xs, y: batch_ys, keep_prob: dropout})         if step % display_step == 0:
            # Calculate batch accuracy             acc = sess.run(accuracy, feed_dict={x: b
atch_xs, y: batch_ys, keep_prob: 1.})             # Calculate batch loss             los
s = sess.run(cost, feed_dict={x: batch_xs, y: batch_ys, keep_prob: 1.})             prin
t "Iter " + str(step*batch_size) + ", Minibatch Loss= " + "{:.6f}".format(loss) + ", Traini
```

```
ng Accuracy= " + "{:.5f}".format(acc)              summary_str = sess.run(merged_summary_o
p, feed_dict={x: batch_xs, y: batch_ys, keep_prob: 1.})            summary_writer.add_summ
ary(summary_str, step)                  saver.save(sess, '/root/alexnet.tfmodel',
                                  step);          step += 1      print "Optimization Finished!"
     # Calculate accuracy for 256 mnist test images      print "Testing Accuracy:", sess.ru
n(accuracy, feed_dict={x: mnist.test.images[:256], y: mnist.test.labels[:256], keep_pro
b: 1.})
```

运行一下就可以了

在一个新的类 或者 其他里面使用

```
saver = tf.train.Saver() with tf.Session() as sess:      saver.restore(sess, '/root/alexne
t.tfmodel')    sess.run(....)
```



```
('Succesfully downloaded', 'train-labels-idx1-ubyte.gz', 28881, 'bytes.')
('Extracting', '/tmp/data/train-labels-idx1-ubyte.gz')
('Succesfully downloaded', 't10k-images-idx3-ubyte.gz', 1648877, 'bytes.')
('Extracting', '/tmp/data/t10k-images-idx3-ubyte.gz')
('Succesfully downloaded', 't10k-labels-idx1-ubyte.gz', 4542, 'bytes.')
('Extracting', '/tmp/data/t10k-labels-idx1-ubyte.gz')
I tensorflow/core/common_runtime/local_device.cc:25] Local device intra op parallelism threads: 1
I tensorflow/core/common_runtime/local_session.cc:45] Local session inter op parallelism threads: 1
Iter 1280, Minibatch Loss= 189811.281250, Training Accuracy= 0.14062
Iter 2560, Minibatch Loss= 79509.171875, Training Accuracy= 0.42188
Iter 3840, Minibatch Loss= 58623.582031, Training Accuracy= 0.40625
Iter 5120, Minibatch Loss= 34966.421875, Training Accuracy= 0.46875
Iter 6400, Minibatch Loss= 27468.613281, Training Accuracy= 0.65625
Iter 7680, Minibatch Loss= 48998.773438, Training Accuracy= 0.42188
Iter 8960, Minibatch Loss= 30511.707031, Training Accuracy= 0.59375
Iter 10240, Minibatch Loss= 23466.617188, Training Accuracy= 0.68750
Iter 11520, Minibatch Loss= 20749.056641, Training Accuracy= 0.67188
Iter 12800, Minibatch Loss= 28307.523438, Training Accuracy= 0.65625
Iter 14080, Minibatch Loss= 27130.816406, Training Accuracy= 0.59375
Iter 15360, Minibatch Loss= 15809.673828, Training Accuracy= 0.70312
Iter 16640, Minibatch Loss= 15052.324219, Training Accuracy= 0.78125
Iter 17920, Minibatch Loss= 11442.550781, Training Accuracy= 0.78125
Iter 19200, Minibatch Loss= 8619.753906, Training Accuracy= 0.81250
Iter 20480, Minibatch Loss= 11589.886719, Training Accuracy= 0.78125
Iter 21760, Minibatch Loss= 11748.119141, Training Accuracy= 0.75000
Iter 23040, Minibatch Loss= 6764.416992, Training Accuracy= 0.79688
Iter 24320, Minibatch Loss= 12309.477539, Training Accuracy= 0.75000
```


上一篇：TensorFlow人工智能引擎入门教程之五 AlphaGo 的策略网络（CNN）简单的实现

下一篇：TensorFlow人工智能引擎入门教程之七 DNN深度神经网络 的原理 以及 使用

发表评论：

请 登录 或者 注册 后再进行评论　　社交帐号登录：