单线程版本

```
import tensorflow as tf
import time
import os
from tensorflow.examples.tutorials.mnist import input_data

INPUT_NODE = 784     # 输入节点
OUTPUT_NODE = 10     # 输出节点
LAYER1_NODE = 500    # 隐藏层节点数  ，这里使用一个隐藏层，有500节点

BATCH_SIZE = 100     # 每次batch打包的样本个数

# 模型相关的参数
LEARNING_RATE_BASE = 0.8    #基础的学习率
LEARNING_RATE_DECAY = 0.99    #学习的衰减率
REGULARAZTION_RATE = 0.0001   #正则化项在损失函数中的系数
TRAINING_STEPS = 5000         #训练轮数
MOVING_AVERAGE_DECAY = 0.99    #滑动平均衰减率

def inference(input_tensor, avg_class, weights1, biases1, weights2, biases2):
    # 不使用滑动平均类
    if avg_class == None:
        layer1 = tf.nn.relu(tf.matmul(input_tensor, weights1) + biases1)
        return tf.matmul(layer1, weights2) + biases2

    else:
        # 使用滑动平均类
        layer1 = tf.nn.relu(tf.matmul(input_tensor, avg_class.average(weights1)) +
 avg_class.average(biases1))
        return tf.matmul(layer1, avg_class.average(weights2)) + avg_class.average(
biases2)

def train(mnist):
    x = tf.placeholder(tf.float32, [None, INPUT_NODE], name='x-input')
    y_ = tf.placeholder(tf.float32, [None, OUTPUT_NODE], name='y-input')
    # 生成隐藏层的参数。
    weights1 = tf.Variable(tf.truncated_normal([INPUT_NODE, LAYER1_NODE], stddev=0
.1))
    biases1 = tf.Variable(tf.constant(0.1, shape=[LAYER1_NODE]))
    # 生成输出层的参数。
    weights2 = tf.Variable(tf.truncated_normal([LAYER1_NODE, OUTPUT_NODE], stddev=
0.1))
    biases2 = tf.Variable(tf.constant(0.1, shape=[OUTPUT_NODE]))
```

```
    # 计算不含滑动平均类的前向传播结果
    y = inference(x, None, weights1, biases1, weights2, biases2)

    # 定义训练轮数及相关的滑动平均类
    global_step = tf.Variable(0, trainable=False)
    variable_averages = tf.train.ExponentialMovingAverage(MOVING_AVERAGE_DECAY, gl
obal_step)
    variables_averages_op = variable_averages.apply(tf.trainable_variables())
    average_y = inference(x, variable_averages, weights1, biases1, weights2, biase
s2)

    # 计算交叉熵及其平均值
    cross_entropy = tf.nn.sparse_softmax_cross_entropy_with_logits(logits=y, label
s=tf.argmax(y_, 1))
    cross_entropy_mean = tf.reduce_mean(cross_entropy)

    # 正则化损失函数的计算
    regularizer = tf.contrib.layers.l2_regularizer(REGULARAZTION_RATE)
    regularaztion = regularizer(weights1) + regularizer(weights2)
    loss = cross_entropy_mean + regularaztion

    # 设置指数衰减的学习率。
    learning_rate = tf.train.exponential_decay(
        LEARNING_RATE_BASE,
        global_step,
        mnist.train.num_examples / BATCH_SIZE,
        LEARNING_RATE_DECAY,
        staircase=True)

    # 优化损失函数
    train_step = tf.train.GradientDescentOptimizer(learning_rate).minimize(loss, g
lobal_step=global_step)

    # 反向传播更新参数和更新每一个参数的滑动平均值
    with tf.control_dependencies([train_step, variables_averages_op]):
        train_op = tf.no_op(name='train')

    # 计算正确率
    correct_prediction = tf.equal(tf.argmax(average_y, 1), tf.argmax(y_, 1))
    accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))

    # 初始化会话，并开始训练过程。
    cpu_num = 1

    config = tf.ConfigProto(device_count={"CPU": cpu_num,'GPU':0},
                inter_op_parallelism_threads = cpu_num,
                intra_op_parallelism_threads = cpu_num,
                log_device_placement=True)
```

```
    with tf.Session(config = config) as sess:
        tf.global_variables_initializer().run()
        validate_feed = {x: mnist.validation.images, y_: mnist.validation.labels}
        test_feed = {x: mnist.test.images, y_: mnist.test.labels}

        # 循环的训练神经网络。
        for i in range(TRAINING_STEPS):
            if i % 1000 == 0:
                validate_acc = sess.run(accuracy, feed_dict=validate_feed)
                print("After %d training step(s), validation accuracy using averag
e model is %g " % (i, validate_acc))

            xs,ys=mnist.train.next_batch(BATCH_SIZE)
            sess.run(train_op,feed_dict={x:xs,y_:ys})

        test_acc=sess.run(accuracy,feed_dict=test_feed)
        print(("After %d training step(s), test accuracy using average model is %g
" %(TRAINING_STEPS, test_acc)))


def main(argv=None):
    mnist = input_data.read_data_sets("./MNIST_data", one_hot=True)
    train(mnist)

if __name__=='__main__':
    time1 = time.time()
    main()
    time2 = time.time()
    print("单线程用时：", time2-time1)
```

```
Extracting ./MNIST_data/train-images-idx3-ubyte.gz
Extracting ./MNIST_data/train-labels-idx1-ubyte.gz
Extracting ./MNIST_data/t10k-images-idx3-ubyte.gz
Extracting ./MNIST_data/t10k-labels-idx1-ubyte.gz
1
After 0 training step(s), validation accuracy using average model is 0.1168
After 1000 training step(s), validation accuracy using average model is 0.9778
After 2000 training step(s), validation accuracy using average model is 0.979
After 3000 training step(s), validation accuracy using average model is 0.9832
After 4000 training step(s), validation accuracy using average model is 0.9832
After 5000 training step(s), test accuracy using average model is 0.9822
单线程用时： 408.33879709243774
```

多线程版本

```
import tensorflow as tf
```

```
import time
from tensorflow.examples.tutorials.mnist import input_data

INPUT_NODE = 784     # 输入节点
OUTPUT_NODE = 10     # 输出节点
LAYER1_NODE = 500    # 隐藏层节点数  ，这里使用一个隐藏层，有500节点

BATCH_SIZE = 100     # 每次batch打包的样本个数

# 模型相关的参数
LEARNING_RATE_BASE = 0.8    #基础的学习率
LEARNING_RATE_DECAY = 0.99    #学习的衰减率
REGULARAZTION_RATE = 0.0001   #正则化项在损失函数中的系数
TRAINING_STEPS = 5000         #训练轮数
MOVING_AVERAGE_DECAY = 0.99    #滑动平均衰减率

def inference(input_tensor, avg_class, weights1, biases1, weights2, biases2):
    # 不使用滑动平均类
    if avg_class == None:
        layer1 = tf.nn.relu(tf.matmul(input_tensor, weights1) + biases1)
        return tf.matmul(layer1, weights2) + biases2

    else:
        # 使用滑动平均类
        layer1 = tf.nn.relu(tf.matmul(input_tensor, avg_class.average(weights1)) +
 avg_class.average(biases1))
        return tf.matmul(layer1, avg_class.average(weights2)) + avg_class.average(
biases2)

def train(mnist):
    x = tf.placeholder(tf.float32, [None, INPUT_NODE], name='x-input')
    y_ = tf.placeholder(tf.float32, [None, OUTPUT_NODE], name='y-input')
    # 生成隐藏层的参数。
    weights1 = tf.Variable(tf.truncated_normal([INPUT_NODE, LAYER1_NODE], stddev=0
.1))
    biases1 = tf.Variable(tf.constant(0.1, shape=[LAYER1_NODE]))
    # 生成输出层的参数。
    weights2 = tf.Variable(tf.truncated_normal([LAYER1_NODE, OUTPUT_NODE], stddev=
0.1))
    biases2 = tf.Variable(tf.constant(0.1, shape=[OUTPUT_NODE]))

    # 计算不含滑动平均类的前向传播结果
    y = inference(x, None, weights1, biases1, weights2, biases2)

    # 定义训练轮数及相关的滑动平均类
    global_step = tf.Variable(0, trainable=False)
    variable_averages = tf.train.ExponentialMovingAverage(MOVING_AVERAGE_DECAY, gl
obal_step)
```

```
    variables_averages_op = variable_averages.apply(tf.trainable_variables())
    average_y = inference(x, variable_averages, weights1, biases1, weights2, biase
s2)

    # 计算交叉熵及其平均值
    cross_entropy = tf.nn.sparse_softmax_cross_entropy_with_logits(logits=y, label
s=tf.argmax(y_, 1))
    cross_entropy_mean = tf.reduce_mean(cross_entropy)

    # 正则化损失函数的计算
    regularizer = tf.contrib.layers.l2_regularizer(REGULARAZTION_RATE)
    regularaztion = regularizer(weights1) + regularizer(weights2)
    loss = cross_entropy_mean + regularaztion

    # 设置指数衰减的学习率。
    learning_rate = tf.train.exponential_decay(
        LEARNING_RATE_BASE,
        global_step,
        mnist.train.num_examples / BATCH_SIZE,
        LEARNING_RATE_DECAY,
        staircase=True)

    # 优化损失函数
    train_step = tf.train.GradientDescentOptimizer(learning_rate).minimize(loss, g
lobal_step=global_step)

    # 反向传播更新参数和更新每一个参数的滑动平均值
    with tf.control_dependencies([train_step, variables_averages_op]):
        train_op = tf.no_op(name='train')

    # 计算正确率
    correct_prediction = tf.equal(tf.argmax(average_y, 1), tf.argmax(y_, 1))
    accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))

    # 初始化会话，并开始训练过程。
    config = tf.ConfigProto(device_count={"CPU": 12,'GPU':0},
                inter_op_parallelism_threads = 1,
                intra_op_parallelism_threads = 12,
                log_device_placement=True)
    #time1 = time.time()
    with tf.Session(config = config) as sess:
        tf.global_variables_initializer().run()
        validate_feed = {x: mnist.validation.images, y_: mnist.validation.labels}
        test_feed = {x: mnist.test.images, y_: mnist.test.labels}

        # 循环的训练神经网络。
        for i in range(TRAINING_STEPS):
            if i % 1000 == 0:
```

```
                validate_acc = sess.run(accuracy, feed_dict=validate_feed)
                print("After %d training step(s), validation accuracy using averag
e model is %g " % (i, validate_acc))

            xs,ys=mnist.train.next_batch(BATCH_SIZE)
            sess.run(train_op,feed_dict={x:xs,y_:ys})

        test_acc=sess.run(accuracy,feed_dict=test_feed)
        print(("After %d training step(s), test accuracy using average model is %g
" %(TRAINING_STEPS, test_acc)))
    #time2 = time.time()
    #print("多线程用时：", time2-time1)

def main(argv=None):
    mnist = input_data.read_data_sets("./MNIST_data", one_hot=True)
    train(mnist)

if __name__=='__main__':
    time1 = time.time()
    main()
    time2 = time.time()
    print("多线程用时：", time2-time1)
```

```
Extracting ./MNIST_data/train-images-idx3-ubyte.gz
Extracting ./MNIST_data/train-labels-idx1-ubyte.gz
Extracting ./MNIST_data/t10k-images-idx3-ubyte.gz
Extracting ./MNIST_data/t10k-labels-idx1-ubyte.gz
After 0 training step(s), validation accuracy using average model is 0.1244
After 1000 training step(s), validation accuracy using average model is 0.9782
After 2000 training step(s), validation accuracy using average model is 0.9812
After 3000 training step(s), validation accuracy using average model is 0.9838
After 4000 training step(s), validation accuracy using average model is 0.984
After 5000 training step(s), test accuracy using average model is 0.9828
多线程用时： 43.9536898136139
```

对比

使用单线程时间是408ms，准确率为0.9822；使用12线程时间是43ms，准确率为0.9828