计算机学院实验报告

|  |  |  |
| --- | --- | --- |
| 实验题目：用Tensorflow做手写字识别 | | 学号： 201600181073 |
| 日期： | 班级： 智能16 | 姓名： 唐超 |
| **Email：1984386166@qq.com** | | |
| 实验目的：  通过本实验掌握如何使用Tensorflow创建神经网络，如何将创建的神经网络应用在MNIST数据集中，如何使用Tensorboard对训练过程进行可视化。 | | |
| 实验环境介绍：  python 3.6 & numpy 1.14.5 & matplotlib 2.2.2 & sklearn 0.19.1 & pandas 0.23.1 | | |
| **解决问题的主要思路：**  当使用Tensorflow训练大量深层的神经网络时，我们希望去跟踪神经网络的整个训练过程中的信息，比如迭代的过程中每一层参数是如何变化与分布的，比如每次循环参数更新后模型在测试集与训练集上的准确率是如何的，比如损失值的变化情况，等等。Tensorflow官方推出了可视化工具Tensorboard，可以帮助我们实现以上功能，它可以将模型训练过程中的各种数据汇总起来存在自定义的路径与日志文件中，然后在指定的web端可视化地展现这些信息。TensorBoard 通过读取 TensorFlow 的事件文件来运行。TensorFlow 的事件文件包括了你会在 TensorFlow 运行中涉及到的主要数据。下面是 TensorBoard 中汇总数据（Summary data）的大体生命周期。  首先，创建你想汇总数据的 TensorFlow 图，然后再选择你想在哪个节点进行汇总(summary)操作。比如，假设你正在训练一个卷积神经网络，用于识别 MNISt 标签。你可能希望记录学习速度(learning rate)的如何变化，以及目标函数如何变化。通过向节点附加scalar\_summary操作来分别输出学习速度和期望误差。然后你可以给每个 scalary\_summary 分配一个有意义的 标签，比如 'learning rate' 和 'loss function'。  或者你还希望显示一个特殊层中激活的分布，或者梯度权重的分布。可以通过分别附加histogram\_summary 运算来收集权重变量和梯度输出。所有可用的 summary 操作详细信息，可以查看summary\_operation文档。在TensorFlow中，所有的操作只有当你执行，或者另一个操作依赖于它的输出时才会运行。我们刚才创建的这些节点（summary nodes）都围绕着你的图像：没有任何操作依赖于它们的结果。因此，为了生成汇总信息，我们需要运行所有这些节点。这样的手动工作是很乏味的，因此可以使用tf.merge\_all\_summaries来将他们合并为一个操作。然后你可以执行合并命令，它会依据特点步骤将所有数据生成一个序列化的Summary protobuf对象。最后，为了将汇总数据写入磁盘，需要将汇总的protobuf对象传递给tf.train.Summarywriter。  SummaryWriter 的构造函数中包含了参数 logdir。这个 logdir 非常重要，所有事件都会写到它所指的目录下。此外，SummaryWriter 中还包含了一个可选择的参数 GraphDef。如果输入了该参数，那么 TensorBoard 也会显示你的图像。现在已经修改了你的图，也有了 SummaryWriter，现在就可以运行你的神经网络了！如果你愿意的话，你可以每一步执行一次合并汇总，这样你会得到一大堆训练数据。这很有可能超过了你想要的数据量。 | | |
| 实验步骤：  **1. 准备数据集、定义超参**  （1）首先是导入需要使用的包：  import tensorflow as tf from tensorflow.examples.tutorials.mnist import input\_data import os  （2）定义超参数  如果你问，这个超参数为啥要这样设定，如何选择最优的超参数？这个问题此处先不讨论，超参数的选择在机器学习建模中最常用的方法就是“交叉验证法”。而现在假设我们已经获得了最优的超参数，设置学利率为0.001，dropout的保留节点比例为0.9，最大循环次数为1000。  另外，还要设置两个路径，第一个是数据下载下来存放的地方，一个是summary输出保存的地方。  max\_steps = 1000  # 最大迭代次数 learning\_rate = 0.001   # 学习率 dropout = 0.9   # dropout时随机保留神经元的比例 data\_dir = '/usr/local/AI\_Trial\_Data/tf\_hand\_writing/MNIST\_DATA'   # 样本数据存储的路径 log\_dir = './MNIST\_LOG'    # 输出日志保存的路径  （3）GPU设置  这里使用GPU进行训练，如果使用cpu，可以略过此步。如果使用GPU建议进行设置。  os.environ["CUDA\_VISIBLE\_DEVICES"] = "0" config = tf.ConfigProto(allow\_soft\_placement = True) gpu\_options = tf.GPUOptions(per\_process\_gpu\_memory\_fraction = 0.33) config.gpu\_options.allow\_growth = True  上述代码的意思是使用GPU设备0，最多给GPU分配总共内存的百分之33，并且允许GPU按需申请内存。也就是说，假设一个程序使用一块GPU内存百分之10就够了，如果我们没有指定allow\_growth=True，那么程序会直接占用GPU内存的百分之33，因为这个是我们给它分配的。如果我们连0.33，也就是GPU内存的百分之33都没有指定，那么程序会直接占用整个GPU设备0。为了充分利用资源，特别是大家共享同一个服务器时，指定这些参数就很有必要了。  （4）下载数据下载数据是直接调用了[tensorflow](http://cuijiahua.com/blog/tag/tensorflow/" \t "_blank" \o "查看与 tensorflow 相关的文章)提供的函数read\_data\_sets，输入两个参数，第一个是下载到数据存储的路径，第二个one\_hot表示是否要将类别标签进行独热编码。它首先回去找制定目录下有没有这个数据文件，没有的话才去下载，有的话就直接读取。所以第一次执行这个命令，速度会比较慢，因为没有数据集，需要进行下载。在实验目录中，已经存在了该数据集，所以不用再去互联网上下载。  # 获取数据集，并采用采用one\_hot编码 mnist = input\_data.read\_data\_sets(data\_dir,one\_hot = True)  下载数据下载数据是直接调用了[tensorflow](http://cuijiahua.com/blog/tag/tensorflow/" \t "_blank" \o "查看与 tensorflow 相关的文章)提供的函数read\_data\_sets，输入两个参数，第一个是下载到数据存储的路径，第二个one\_hot表示是否要将类别标签进行独热编码。它首先回去找制定目录下有没有这个数据文件，没有的话才去下载，有的话就直接读取。所以第一次执行这个命令，速度会比较慢，因为没有数据集，需要进行下载。在实验目录中，已经存在了该数据集，所以不用再去互联网上下载。  # 获取数据集，并采用采用one\_hot编码 mnist = input\_data.read\_data\_sets(data\_dir,one\_hot = True)  **2.数据处理**  （1）创建tensorflow默认会话：  sess = tf.InteractiveSession(config = config)  为了使设置的GPU参数生效，我们需要在创建会话的时候传入这个config参数。  （2）创建输入数据的占位符，分别创建特征数据x，标签数据y\_  在tf.placeholder()函数中传入了3个参数，第一个是定义数据类型为float32；第二个是数据的大小，特征数据是大小784的向量，标签数据是大小为10的向量，None表示不确定大小，后续可以传入任何数量的样本；第3个参数是这个占位符的名称。  with tf.name\_scope('input'):      x = tf.placeholder(tf.float32, [None, 784], name='x-input')      y\_ = tf.placeholder(tf.float32, [None, 10], name='y-input')  mnist下载好的数据集是很多个大小为1\*784的向量，就是已经对28\*28的图片进行了向量化处理。  （3）使用tf.summary.image保存图像信息  前面也说了，特征数据其实就是图像的像素数据拉升成一个1\*784的向量，现在如果想在tensorboard上还原出输入的特征数据对应的图片，就需要将拉升的向量转变成28 \* 28 \* 1的原始像素了，于是可以用tf.reshape()直接重新调整特征数据的维度：  将输入的数据转换成[28 \* 28 \* 1]的shape，存储成另一个tensor，命名为image\_shaped\_input。  为了能使图片在tensorbord上展示出来，使用tf.summary.image将图片数据汇总给tensorbord。  tf.summary.image（）中传入的第一个参数是命名，第二个是图片数据，第三个是最多展示的张数，此处为10张。  # 保存图像信息  with tf.name\_scope('input\_reshape'):      image\_shaped\_input = tf.reshape(x, [-1, 28, 28, 1])      tf.summary.image('input', image\_shaped\_input, 10)  **3.初始化参数并保存到summary**  （1）初始化参数w和b  在构建神经网络模型中，每一层中都需要去初始化参数w,b,为了使代码简介美观，最好将初始化参数的过程封装成方法function。 创建初始化权重w的方法，生成大小等于传入的shape参数，标准差为0.1，遵循正态分布的随机数，并且将它转换成tensorflow中的variable返回。  # 初始化权重参数 def weight\_variable(shape):     initial = tf.truncated\_normal(shape, stddev = 0.1)     return tf.Variable(initial)  创建初始换偏执项b的方法，生成大小为传入参数shape的常数0.1，并将其转换成tensorflow的variable并返回。  # 初始化偏执参数 def bias\_variable(shape):     initial = tf.constant(0.1, shape = shape)     return tf.Variable(initial)  （2）记录训练过程参数变化  我们知道，在训练的过程在参数是不断地在改变和优化的，我们往往想知道每次迭代后参数都做了哪些变化，可以将参数的信息展现在tenorbord上，因此我们专门写一个方法来收录每次的参数信息。  # 绘制参数变化 def variable\_summaries(var):     with tf.name\_scope('summaries'):         # 计算参数的均值，并使用tf.summary.scaler记录         mean = tf.reduce\_mean(var)         tf.summary.scalar('mean', mean)          # 计算参数的标准差         with tf.name\_scope('stddev'):             stddev = tf.sqrt(tf.reduce\_mean(tf.square(var - mean)))         # 使用tf.summary.scaler记录记录下标准差，最大值，最小值         tf.summary.scalar('stddev', stddev)         tf.summary.scalar('max', tf.reduce\_max(var))         tf.summary.scalar('min', tf.reduce\_min(var))         # 用直方图记录参数的分布         tf.summary.histogram('histogram', var)  **4.创造损失函数**    使用tf.nn.softmax\_cross\_entropy\_with\_logits来计算softmax并计算交叉熵损失,并且求均值作为最终的损失值。  # 创建损失函数  with tf.name\_scope('loss'):      # 计算交叉熵损失（每个样本都会有一个损失）      diff = tf.nn.softmax\_cross\_entropy\_with\_logits(labels=y\_, logits=y)      with tf.name\_scope('total'):          # 计算所有样本交叉熵损失的均值          cross\_entropy = tf.reduce\_mean(diff)      tf.summary.scalar('loss', cross\_entropy)  首先，使用AdamOptimizer优化器训练模型，最小化交叉熵损失。  # 使用AdamOptimizer优化器训练模型，最小化交叉熵损失  with tf.name\_scope('train'):      train\_step = tf.train.AdamOptimizer(learning\_rate).minimize(cross\_entropy)  然后，计算准确率,并用tf.summary.scalar记录准确率。  # 计算准确率  with tf.name\_scope('accuracy'):      with tf.name\_scope('correct\_prediction'):          # 分别将预测和真实的标签中取出最大值的索引，弱相同则返回1(true),不同则返回0(false)          correct\_prediction = tf.equal(tf.argmax(y, 1), tf.argmax(y\_, 1))      with tf.name\_scope('accuracy'):          # 求均值即为准确率          accuracy = tf.reduce\_mean(tf.cast(correct\_prediction, tf.float32)) | | |
| **实验结果展示及分析：**  利用tensorboard进行结果展示如下  ①accuracy:  ②droupout  ③layer1的一些列属性  A.biases的max,mean,min,stddev  B.weight的max，mean,min,stddev  loss的变化： | | |