第1章 人工智能概述

只要计算可以表示为一个数据流图（data flow graph），就可以使用TensorFlow，只需要构建图，书写计算的内部循环即可。

第3章 可视化TensorFlow

有了神经网络，我们的系统自己就能学习到哪些特征是有效的、哪些是无效的，通过自己学习的这些特征，就可以做到自己分类，这就大大提高了我们解决语音、图像这种复杂抽象问题的能力。

TensorBoard

可视化的主要功能如下：

1. SCALARS：展示训练过程中的准确率、损失值、权重/偏置的变化情况；
2. IMAGES：展示训练过程中记录的图像；
3. AUDIO：展示训练过程中记录的音频；
4. GRAPHS：展示模型的数据流图，以及训练在各个设备上消耗的内存和时间；
5. DISTRIBUTIONS：展示训练过程中记录的数据的分布图；
6. HISTOGRAMS：展示训练过程中记录的数据的柱状图；
7. EMBEDDINGS：展示词向量后的投影分布。

第4章 TensorFlow基础知识

TensorFlow完全采用符号式编程。符号式计算一般是先定义各种变量，然后建立一个数据流图，在数据流图中规定各个变量之间的计算关系，最后需要对数据流图进行编译，但此时的数据流图还是一个空壳儿，里面没有任何实际数据，只有把需要运算的输入放进去后，才能在整个模型中形成数据流，从而形成输出值。

开启会话后，就可以用数据去填充（feed）节点，进行运算；关闭会话后，就不能进行计算了。

|  |
| --- |
| 1. with tf.Session() as sess: 2. # 指定在gpu上运行 3. with tf.device('/gpu:0'): 4. matrix1 = tf.constant([[3., 3.]]) 5. matrix2 = tf.constant([[2.], [2.]]) 6. product = tf.matmul(matrix1, matrix2) 7. **print**(sess.run(product))  # 输出为[[12.]] |

变量作用域

tf.variable\_scope()

|  |
| --- |
| 1. # 给Tensor加作用域前缀 2. with tf.variable\_scope('foo'): 3. v = tf.get\_variable('v', [1]) 4. # ValueError，变量'v'已经被定义过了  v.name为’foo/v:0’ 5. # v2 = tf.get\_variable('v', [1]) 6. with tf.variable\_scope('foo', reuse=True): 7. # 实现共享变量 8. v1 = tf.get\_variable('v', [1]) 9. # 给op加作用域前缀 10. with tf.variable\_scope('coo'): 11. x = 1.0 + tf.get\_variable('v', [1]) # x.name为'coo/add:0' |

tf.name\_scope()

其为变量划分范围，在可视化中，表示在计算图中的一个层级。其会影响op，及通过tf.Variable()创建的变量，但不会影响tf.get\_variable()创建的变量。

|  |
| --- |
| 1. with tf.variable\_scope('doo'): 2. with tf.name\_scope('bar'): 3. v = tf.get\_variable('v', [1]) # v.name为doo/v:0 4. b = tf.Variable(tf.zeros([1]), name='b') # b.name为doo/bar/b:0 5. x = 1.0 + v # x.name为doo/bar/add:0 |

批标准化（Batch Normalization，BN）

为了克服神经网络层数加深导致难以训练而诞生的。

**Covariate Shift**：指训练集的样本数据和目标样本集分布不一致时，训练得到的模型无法很好地泛化。

对于神经网络的各层输出，在经过了层内操作后，各层输出分布就会与对应的输入信号分布不同，而且差异会随着网络深度增大而加大，但是每一层所指向的样本标记仍然是不变的。因此，通过引入批标准化来规范化某些层或者所有层的输入，从而固定每层输入信号的均值和方差。

批标准化一般用在激活函数之前，对做规范化。

激活函数

tf.nn.dropout()：一个神经元以概率keep\_prob决定是否被抑制。如果被抑制，其输出为0；否则，其输出值放大为原来的1/keep\_prob倍。

|  |
| --- |
| 1. a = tf.constant([[-1.0, 2.0, 3.0, 4.0]]) 2. with tf.Session() as sess: 3. b = tf.nn.dropout(a, keep\_prob=0.9) |

激活函数的选择：

当输入数据特征相差明显时，用tanh的效果会很好，且在循环过程中会不断扩大特征效果并显示出来。当特征相差不明显时，sigmoid效果比较好。同时，用sigmoid和tanh作为激活函数时，需要对输入进行规范化，否则激活后的值全部都进入平坦区，隐层的输出会全部趋同，丧失原有的特征表达。而relu会好很多，有时输入可以不需要规范化来避免上述情况。

卷积函数

池化函数

池化函数一般跟在卷积函数的下一层，其利用一个矩阵窗口在张量上进行扫描，将每个矩阵窗口中的值通过最大值或平均值来减少元素个数。

|  |
| --- |
| 1. input\_data = tf.Variable(np.random.rand(10,9,9,3), dtype=np.float32) 2. filter\_data = tf.Variable(np.random.rand(2,2,3,2), dtype=np.float32) 3. y = tf.nn.conv2d(input\_data, filter\_data, strides=[1,1,1,1], padding='SAME') 4. # output的shape为[10,9,9,2] 5. output = tf.nn.max\_pool(value=y, ksize=[1,2,2,1], strides=[1,1,1,1], padding='SAME') |

参数padding：

1）’SAME’：输入数据维度和输出数据维度相同。

2）’VALID’：输入数据维度和输出数据维度不同。

分类函数

优化方法

BGD法（batch gradient descent）

SGD法（Stochastic gradient descent）

主要思想：将数据集拆分成一个个批次（batch），随机抽取一个批次来计算并更新参数，也称为MBGD（Minibatch Gradient Descent）。

缺点：

1. 由于抽取不可避免地梯度会有误差，需要手动调整学习率。且在参数更新时，对所有参数采用一样地学习率。
2. 容易收敛到局部最优，并且在某些情况下可能被困在鞍点。

Momentum法

能够更新学习率，在下降初期，前后梯度方向一致时，能够加速学习；在下降的中后期，在局部最小值的附近来回震荡时，能够抑制震荡，加快收敛。

Nesterov Momentum法

Adagrad法

能够自适应地为各个参数分配不同地学习率。

缺点：其学习率单调递减，在训练的后期学习率非常小，且需手动设置一个全局的初始学习率。

Adadelta法

解决了Adagrad存在的缺点。

RMSProp法

Adam法

模型的存储与加载

|  |
| --- |
| 1. (tr\_x, tr\_y), (te\_x, te\_y) = tf.keras.datasets.mnist.load\_data(path='mnist.npz') 2. train\_x = tr\_x.reshape([-1, 28 \* 28]) 3. test\_x = te\_x.reshape([-1, 28 \* 28]) 5. one\_hot = OneHotEncoder() 6. tr\_y = tr\_y.reshape([60000,-1]) 7. te\_y = te\_y.reshape([10000,-1]) 8. one\_hot.fit(tr\_y) 9. train\_y = one\_hot.transform(tr\_y).toarray() 10. test\_y = one\_hot.transform(te\_y).toarray() 12. **def** init\_weights(shape): 13. """定义权重函数""" 14. **return** tf.Variable(tf.random\_normal(shape, stddev=0.01)) 16. **def** model(X, w\_h, w\_h2, w\_o, p\_keep\_input, p\_keep\_hidden): 17. # 定义模型 18. # 第一个全连接层Wx+b 19. X = tf.nn.dropout(X, p\_keep\_input) 20. h = tf.nn.relu(tf.matmul(X, w\_h)) 22. h = tf.nn.dropout(h, p\_keep\_hidden) 23. # 第二个全连接层 24. h2 = tf.nn.relu(tf.matmul(h, w\_h2)) 25. h2 = tf.nn.dropout(h2, p\_keep\_hidden) 26. **return** tf.matmul(h2, w\_o) # 输出预测值 28. X = tf.placeholder(tf.float32, [None, 28\*28]) 29. Y = tf.placeholder(tf.float32, [None, 10]) 31. w\_h = init\_weights([784, 625]) 32. w\_h2 = init\_weights([625, 625]) 33. w\_o = init\_weights([625, 10]) 35. p\_keep\_input = tf.placeholder(tf.float32) 36. p\_keep\_hidden = tf.placeholder(tf.float32) 37. py\_x = model(X, w\_h, w\_h2, w\_o, p\_keep\_input, p\_keep\_hidden) 39. # 定义损失函数 40. cost = tf.reduce\_mean(tf.nn.softmax\_cross\_entropy\_with\_logits(logits=py\_x, labels=Y)) 41. train\_op = tf.train.RMSPropOptimizer(0.001, 0.9).minimize(cost) 42. predict\_op = tf.arg\_max(py\_x, 1) 44. # 定义存储路径 45. ckpt\_dir = './ckpt\_dir' 46. **if** **not** os.path.exists(ckpt\_dir): 47. os.makedirs(ckpt\_dir) 49. # 定义计数器，为训练轮次计数 50. global\_step = tf.Variable(0, name='global\_step', trainable=False) 52. saver = tf.train.Saver() 54. # 训练模型并存储 55. with tf.Session() as sess: 56. tf.initialize\_all\_variables().run() 57. start = global\_step.eval() 58. **print**('Start from:', start) 60. **for** i **in** range(start, 100): 61. **for** start, end **in** zip(range(0, len(train\_x), 128), range(128, len(train\_x)+1, 128)): 62. sess.run(train\_op, feed\_dict={X:train\_x[start:end], Y:train\_y[start:end], p\_keep\_input:0.8, p\_keep\_hidden:0.5}) 63. global\_step.assign(i).eval() 64. saver.save(sess, ckpt\_dir + './model.ckpt', global\_step=global\_step) |

其中，在chpt\_dir目录下，model.ckpt-{n}.data-00000-of-00001文件是训练过程中保存的模型。model.ckpt-{n}.meta文件是训练过程中保存的元数据，{n}表示迭代次数。以及1个checkpoint文本文件，里面保存中当前模型和最近的5个模型。

每个固定的轮数在检查点保存一个模型（.ckpt文件），就可在出现意外状况时接着上一次的地方开始训练，也有利于随时将模型拿出来进行预测。

队列

tf.FIFOQueue()

tf.FIFOQueue(capacity, dtypes)

其为先入先出队列，在训练一些语音、文字样本时，使用循环神经网络的网络结构，希望输入的训练样本是有序的，就要用FIFOQueue。

|  |
| --- |
| 1. q = tf.FIFOQueue(3, tf.float32) 2. init = q.enqueue\_many(([0.1, 0.2, 0.3],)) # 0.1先入 3. x = q.dequeue() 4. y = x + 1 5. q\_inc = q.enqueue([y]) 7. with tf.Session() as sess: 8. sess.run(init) 9. quelen = sess.run(q.size()) 10. **for** i **in** range(2): 11. sess.run(q\_inc) 13. quelen = sess.run(q.size()) 14. **for** i **in** range(quelen): 15. # 依次输出：0.3, 1.1, 1.2 16. **print**(sess.run(q.dequeue())) |

tf.RandomShuffleQueue()

tf.RandomShuffleQueue(capacity, min\_after\_dequeue, dtypes)

其为随机队列，在出队列时，是以随机的顺序产生元素的。

当1）队列长度等于最小值，执行出队操作；或2）队列长度大于最大值，执行入队操作时，会发生阻断现象。如下，设置在运行时的等待时间来解除阻断。

|  |
| --- |
| 1. with tf.Session() as sess: 2. run\_options = tf.RunOptions(timeout\_in\_ms=10000) 3. **try**: 4. sess.run(q.dequeue(), options=run\_options) 5. **except** tf.errors.DeadlineExceededError: 6. **print**('out of range') |

tf.train.QueueRunner()

tf.train.QueueRunner(queue=, enqueue\_ops=)

使用线程管理器QueueRunner创建一系列的新线程进行入队操作，让主线程继续使用数据，即训练网络和读取数据是异步的。

|  |
| --- |
| 1. q = tf.FIFOQueue(1000, tf.float32) 2. counter = tf.Variable(0.0) 3. increment\_op = tf.assign\_add(counter, tf.constant(1.0)) 4. enqueue\_op = q.enqueue(counter) 6. # 创建队列管理器 7. qr = tf.train.QueueRunner(q, enqueue\_ops=[increment\_op, enqueue\_op] \* 1) 9. with tf.Session() as sess: 10. tf.global\_variables\_initializer().run() 11. enqueue\_threads = qr.create\_threads(sess, start=True) # 启动线程 12. # 主线程 13. **for** i **in** range(10): 14. **print**(sess.run(q.dequeue())) |

存在的问题：入队操作自顾自地执行，在需要的出队操作完成之后，程序没法结束。此时，需要使用tf.train.Coordinator来完成线程间的同步，终止其他线程。

tf.train.Coordinator()

|  |
| --- |
| 1. with tf.Session() as sess: 2. tf.global\_variables\_initializer().run() 3. coord = tf.train.Coordinator() 4. enqueue\_threads = qr.create\_threads(sess, start=True, coord=coord) # 启动线程 5. # 主线程 6. **for** i **in** range(10): 7. **print**(sess.run(q.dequeue())) 8. coord.request\_stop()  # 通知其他线程关闭 9. coord.join(enqueue\_threads)  # 等待其他线程结束 |

若先request\_stop()再执行主线程，会抛出tf.errors.OutOfRangeError错误。

加载数据

1）预加载数据

在TensorFlow图中定义常量或变量来保存所有数据。缺点在于，将数据直接嵌入数据流图中，当训练数据较大时，很消耗内存。

2）填充数据

使用sess.run()中的feed\_dict参数。

缺点：数据量大时，较消耗内存、且数据转换等中间环节增加了不少开销。

3）从文件中读取数据

第5章 TensorFlow源代码解析

分析\tensorflow-master\models里面包含的基本模型。

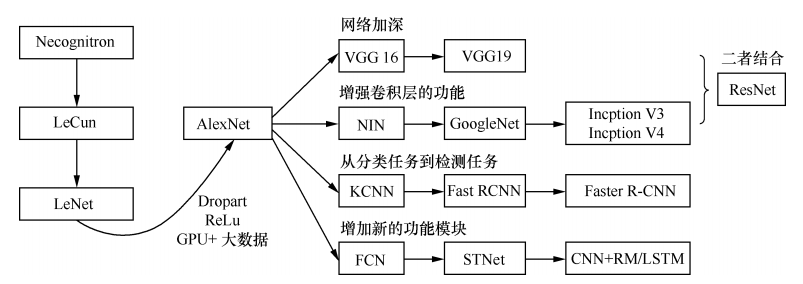
\tensorflow-master\tensorflow\contrib\slim\python\slim\nets\

第6章 神经网络的发展及其TensorFlow实现

卷积神经网络（CNN）

其为权值共享的网络结构，显著降低了模型的复杂度，减少了权值的数量。

其特点为：隐藏层分为卷积层和池化层。卷积层通过一块块卷积核在原始图像上平移来提取特征，每一个特征就是一个特征映射；而池化层通过汇聚特征后稀疏参数来减少要学习的参数，降低网络的复杂度。



LeNet：《GradientBased Learning Applied to Document Recognition》

AlexNet：《ImageNet Classification with Deep Convolutional Neural Networks》

VGGNet：《Very Deep Convolutional Networks for Large-Scale Visual Recognition》

NIN：《Network In Network》

GoogLeNet：《Going Deeper with Convolutions》

ResNet：《Deep Residual Learning for Image Recognition》

Faster R-CNN：《Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks》

防止过拟合：

1）**数据增强**（data augmentation）：增加训练数据是避免过拟合的好方法，并且能提升算法的准确率。当训练数据有限时，可通过一些变换从已有的训练数据集中生成一些新数据。

* 水平翻转图像
* 从原始图像随机地平移变换出一些图像
* 给图像增加一些随机的光照（又称颜色抖动）

1. Dropout

NIN的思想：1）将原来的线性卷积层（linear convolution layer）变为多层感知卷积层（multilayer perceptron）；2）将全连接层改进为全局平均池化。