如何基于TensorFlow使用LSTM和CNN实现时序分类任务

机器之心 · 4 个月前

> 时序数据经常出现在很多领域中，如金融、信号处理、语音识别和医药。传统的时序问题

> 通常首先需要人力进行特征工程，才能将预处理的数据输入到机器学习算法中。并且这种特征工程通常需要一些特定领域内的专业知识，因此也就更进一步加大了预处理成本。例如信号处理（即 EEG 信号分类），特征工程可能就涉及到各种频带的功率谱（power spectra）、Hjorth 参数和其他一些特定的统计学特征。本文简要地介绍了使用 CNN 和 LSTM 实现序列分类的方法，详细代码请查看 Github。

选自burakhimmetoglu

作者：Tom Brander

机器之心编译

参与：蒋思源

Github 项目地址：github.com/healthDataSc...

传统图像分类中也是采用的手动特征工程，然而随着深度学习的出现，卷积神经网络已经可以较为完美地处理计算机视觉任务。使用 CNN 处理图像不需要任何手动特征工程，网络会一层层自动从最基本的特征组合成更加高级和抽象的特征，从而完成计算机视觉任务。

在本文中，我们将讨论如何使用深度学习方法对时序数据进行分类。我们使用的案例是 UCI 项目中的人体活动识别（HAR）数据集。该数据集包含原始的时序数据和经预处理的数据（包含 561 个特征）。本文将对比用特征工程的机器学习算法和两种深度学习方法（卷积神经网络和循环神经网络），试验最后表明深度学习方法超越了传统使用特征工程的方法。

作者使用 TensorFlow 和实现并训练模型，文中只展示了部分代码，更详细的代码请查看 Github。

卷积神经网络（CNN）

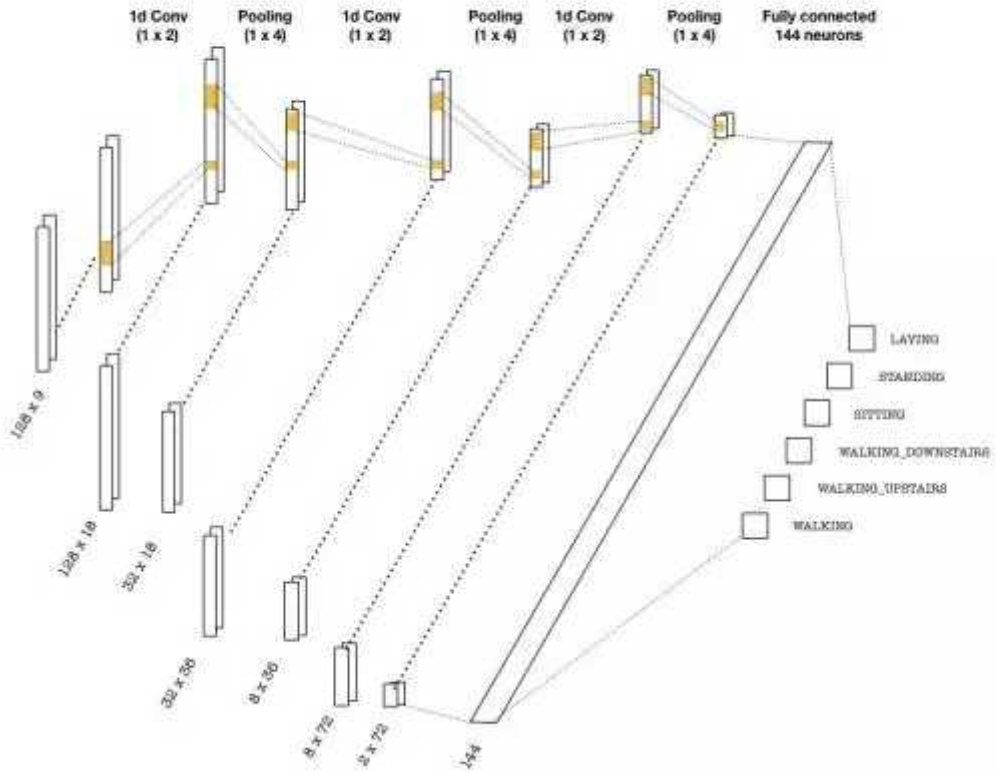
首先第一步就是将数据馈送到 Numpy 中的数组，且数组的维度为 (batch_size, seq_len, n_channels)，其中 batch_size 为模型在执行 SGD 时每一次迭代需要的数据量，seq_len 为时序序列的长度（本文中为 128），n_channels 为执行检测（measurement）的通道数。本文案例中通道数为 9，即 3 个坐标轴每一个有 3 个不同的加速检测（acceleration measurement）。我们有六个活动标签，即每一个样本属于 LAYING、STANDING、SITTING、WALKING_DOWNSTAIRS、WALKING_UPSTAIRS 或 WALKING。

下面，我们首先构建计算图，其中我们使用占位符为输入数据做准备：

```
graph = tf.Graph()
with graph.as_default():
inputs_ = tf.placeholder(tf.float32, [None, seq_len, n_channels],
name = 'inputs')
labels_ = tf.placeholder(tf.float32, [None, n_classes], name = 'labels')
keep_prob_ = tf.placeholder(tf.float32, name = 'keep')
learning_rate_ = tf.placeholder(tf.float32, name = 'learning_rate')
```

其中 inputs_是馈送到计算图中的输入张量，第一个参数设置为「None」可以确保占位符第一个维度可以根据不同的批量大小而适当调整。labels_是需要预测的 one-hot 编码标签，keep_prob_为用于 dropout 正则化的保持概率，learning_rate_ 为用于 Adam 优化器的学习率。

我们使用在序列上移动的 1 维卷积核构建卷积层，图像一般使用的是 2 维卷积核。序列任务中的卷积核可以充当为训练中的滤波器。在许多 CNN 架构中，层级的深度越大，滤波器的数量就越多。每一个卷积操作后面都跟随着池化层以减少序列的长度。下面是我们可以使用的简单 CNN 架构。



上图描述的卷积层可用以下代码实现：

```
with graph.as_default():
# (batch, 128, 9) -> (batch, 32, 18)
conv1 = tf.layers.conv1d(inputs=inputs_, filters=18, kernel_size=2, strides=1,
padding='same', activation = tf.nn.relu)
max_pool_1 = tf.layers.max_pooling1d(inputs=conv1, pool_size=4, strides=4,
padding='same')

# (batch, 32, 18) -> (batch, 8, 36)
conv2 = tf.layers.conv1d(inputs=max_pool_1, filters=36, kernel_size=2, strides=1,
padding='same', activation = tf.nn.relu)
max_pool_2 = tf.layers.max_pooling1d(inputs=conv2, pool_size=4, strides=4,
padding='same')

# (batch, 8, 36) -> (batch, 2, 72)
conv3 = tf.layers.conv1d(inputs=max_pool_2, filters=72, kernel_size=2, strides=1,
padding='same', activation = tf.nn.relu)
max_pool_3 = tf.layers.max_pooling1d(inputs=conv3, pool_size=4, strides=4,
padding='same')
```

一旦到达了最后一层，我们需要 flatten 张量并投入到有适当神经元数的分类器中，在上图中为 144 个神经元。随后分类器输出 logits，并用于以下两种案例：

- 计算 softmax 交叉熵函数，该损失函数在多类别问题中是标准的损失度量。
- 在最大化概率和准确度的情况下预测类别标签。

下面是上述过程的实现：

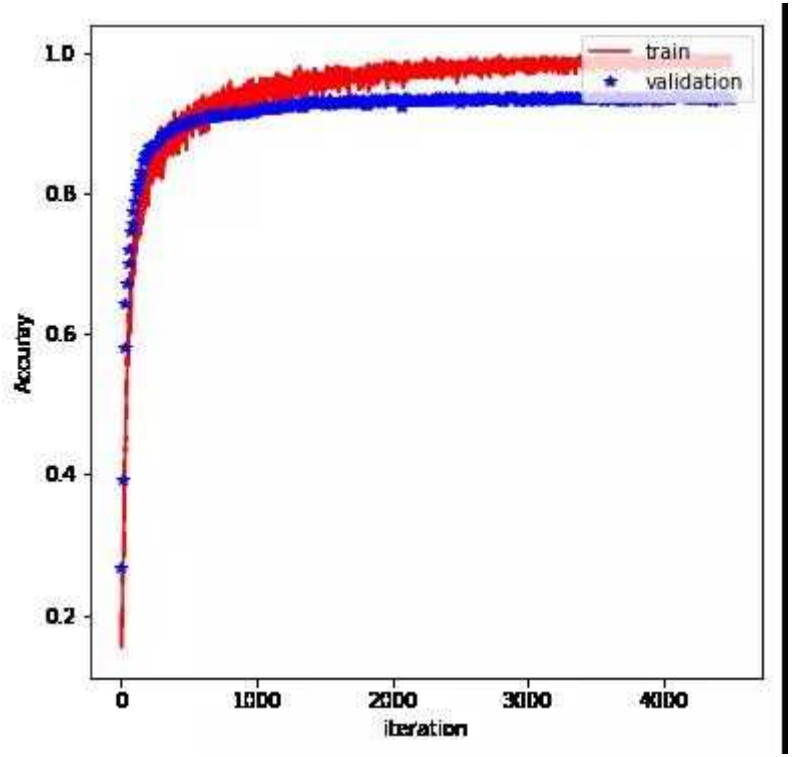
```
with graph.as_default():
# Flatten and add dropout
flat = tf.reshape(max_pool_3, (-1, 2*72))
flat = tf.nn.dropout(flat, keep_prob=keep_prob_)

# Predictions
logits = tf.layers.dense(flat, n_classes)

# Cost function and optimizer
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=logits,
labels=labels_)) optimizer = tf.train.AdamOptimizer(learning_rate_).minimize(cost)

# Accuracy
correct_pred = tf.equal(tf.argmax(logits, 1), tf.argmax(labels_, 1)) accuracy =
tf.reduce_mean(tf.cast(correct_pred, tf.float32), name='accuracy')
```

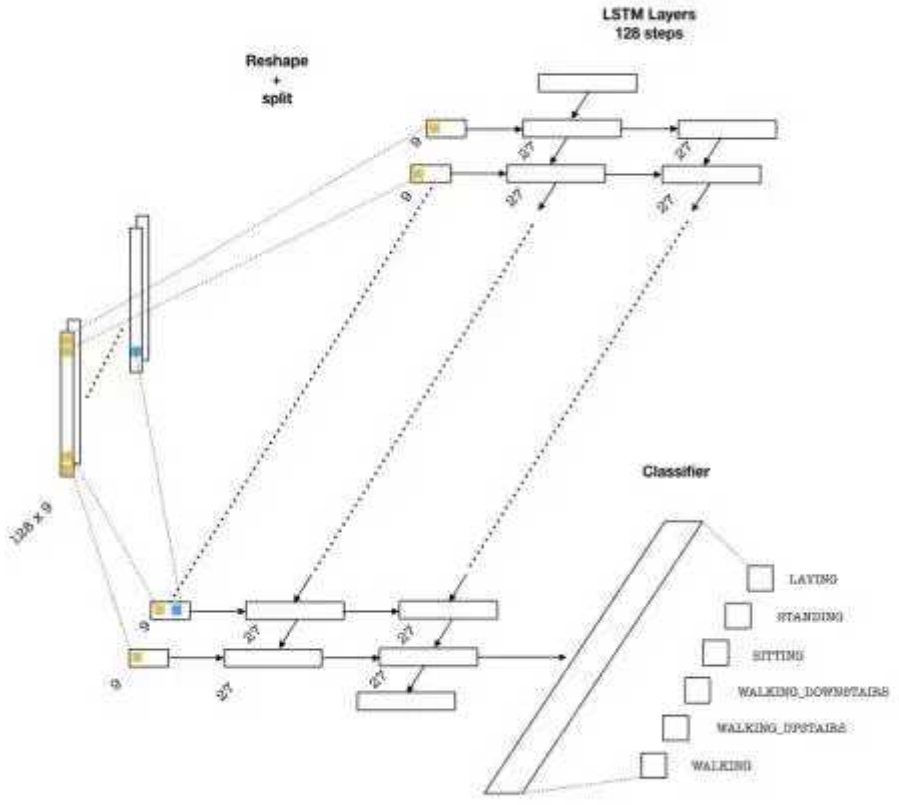
剩下的实现部分就比较典型了，读者可查看 GitHub 中的完整代码和过程。前面我们已经构建了计算图，后面就需要将批量训练数据馈送到计算图进行训练，同时我们还要使用验证集来评估训练结果。最后，完成训练的模型将在测试集上进行评估。我们在该实验中 batch_siza 使用的是 600、learning_rate 使用的是 0.001、keep_prob 为 0.5。在 500 个 epoch 后，我们得到的测试精度为 98%。下图显示了训练准确度和验证准确度随 epoch 的增加而显示的变化：



长短期记忆网络（LSTM）

LSTM 在处理文本数据上十分流行，它在情感分析、机器翻译、和文本生成等方面取得了十分显著的成果。因为本问题涉及相似分类的序列，所以 LSTM 是比较优秀的方法。

下面是能用于该问题的神经网络架构：



为了将数据馈送到网络中，我们需要将数组分割为 128 块（序列中的每一块都会进入一个 LSTM 单元），每一块的维度为（batch_size, n_channels）。随后单层神经元将转换这些输入并馈送到 LSTM 单元中，每一个 LSTM 单元的维度为 lstm_size，一般该参数需要选定为大于通道数量。这种方式很像文本应用中的嵌入层，其中词汇从给定的词汇表中嵌入为一个向量。后面我们需要选择 LSTM 层的数量（lstm_layers），我们可以设定为 2。

对于这一个实现，占位符的设定可以和上面一样。下面的代码段实现了 LSTM 层级：

```
with graph.as_default():
# Construct the LSTM inputs and LSTM cells
lstm_in = tf.transpose(inputs_, [1,0,2]) # reshape into (seq_len, N, channels)
lstm_in = tf.reshape(lstm_in, [-1, n_channels]) # Now (seq_len*N, n_channels)

# To cells
lstm_in = tf.layers.dense(lstm_in, lstm_size, activation=None)

# Open up the tensor into a list of seq_len pieces
lstm_in = tf.split(lstm_in, seq_len, 0)

# Add LSTM layers
lstm = tf.contrib.rnn.BasicLSTMCell(lstm_size)
drop = tf.contrib.rnn.DropoutWrapper(lstm, output_keep_prob=keep_prob_)
cell = tf.contrib.rnn.MultiRNNCell([drop] * lstm_layers)
initial_state = cell.zero_state(batch_size, tf.float32)
```

上面的代码段是十分重要的技术细节。我们首先需要将数组从 (batch_size, seq_len, n_channels) 重建维度为 (seq_len, batch_size, n_channels)，因此 tf.split 将在每一步适当地分割数据（根据第 0 个索引）为一系列 (batch_size, lstm_size) 数组。剩下的部分就是标准的 LSTM 实现了，包括构建层级和初始状态。

下一步就是实现网络的前向传播和成本函数。比较重要的技术点是我们引入了梯度截断，因为梯度截断可以在反向传播中防止梯度爆炸而提升训练效果。

下面是我们定义前向传播和成本函数的代码：

```
with graph.as_default():
outputs, final_state = tf.contrib.rnn.static_rnn(cell, lstm_in, dtype=tf.float32,
initial_state = initial_state)

# We only need the last output tensor to pass into a classifier
logits = tf.layers.dense(outputs[-1], n_classes, name='logits')

# Cost function and optimizer
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=logits, labels=labels_))

# Grad clipping
train_op = tf.train.AdamOptimizer(learning_rate_)
gradients = train_op.compute_gradients(cost) capped_gradients = [(tf.clip_by_value(grad,
-1., 1.), var) for grad, var in gradients] optimizer =
train_op.apply_gradients(capped_gradients)

# Accuracy
correct_pred = tf.equal(tf.argmax(logits, 1), tf.argmax(labels_, 1)) accuracy =
tf.reduce_mean(tf.cast(correct_pred, tf.float32), name='accuracy')
```

注意我们只使用了 LSTM 顶层输出序列的最后一个元素，因为我们每个序列只是尝试预测一个分类概率。剩下的部分和前面我们训练 CNN 的过程相似，我们只需要将数据馈送到计算图中进行训练。其中超参数可选择为 lstm_size=27、lstm_layers=2、batch_size=600、learning_rate=0.0005 和 keep_prob=0.5，我们在测试集中可获得大约 95% 的准确度。这一结果要比 CNN 还差一些，但仍然十分优秀。可能选择其它超参数能产生更好的结果，读者朋友也可以在 Github 中获取源代码并进一步调试。

对比传统方法

前面作者已经使用带 561 个特征的数据集测试了一些机器学习方法，性能最好的方法是梯度提升树，如下梯度提升树的准确度能到达 96%。虽然 CNN、LSTM 架构与经过特征工程的梯度提升树的精度差不多，但 CNN 和 LSTM 的人工工作量要少得多。

HAR 任务经典机器学习方法：github.com/bhimmetoglu/...

梯度提升树：rpubs.com/burakh/har_xg...

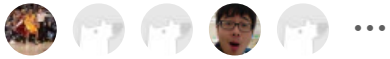
结语

在本文中，我们试验了使用 CNN 和 LSTM 进行时序数据的分类，这两种方法在性能上都有十分优秀的表现，并且最重要的是它们在训练中会一层层学习独特的特征，它们不需要成本昂贵的特征工程。

本文所使用的序列还是比较小的，只有 128 步。可能会有读者怀疑如果序列变得更长（甚至大于 1000），是不是训练就会变得十分困难。其实我们可以结合 LSTM 和 CNN 在这种长序列任务中表现得更好。总的来说，深度学习方法相对于传统方法有非常明显的优势。

LSTM　　卷积神经网络（CNN）

TensorFlow



文章被以下专栏收录

机器之心

关注人工智能学术和技术实现

进入专栏

4 条评论

写下你的评论...

墨墨无语

虽然看不懂，但是感觉很厉害的样子，先点个赞！

4 个月前

风雨人生路

虽然看不懂，但是觉得很厉害的样子，先点个赞

4 个月前

陈某人

如果CNN用Resnet，最终效果会不会有一定的提升呢

4 个月前

鬼马

你好，可以分享一下你的数据集吗

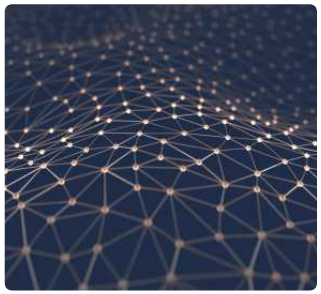
3 个月前

推荐阅读

每周工作80小时惹热议，吴恩达deeplearning.ai成为创业公司

在 8 月 8 日吴恩达正式宣布教育项目 deeplearning.ai 启动后（参见《机器之心专访吴恩达，… 查看全文 ›

机器之心 · 4 个月前 · 发表于 机器之心



爆款论文提出简单循环单元SRU：像CNN一样快速训练RNN（附开源代码）

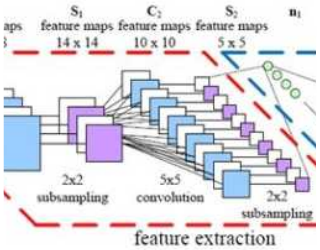
近日，一篇题为《Training RNNs as Fast as CNNs》的 arXiv 论文通过有意简化状态计算并展现… 查看全文 ›

机器之心 · 4 个月前 · 发表于 机器之心

TensorFlow搭建CNN卷积神经网络

该教程采用TernsorFlow搭建CNN卷积神经网络，并利用MNIST数据集进行数字的手写识别数据结构mnist原始图片输入，原始图片的尺寸为28×28，导入后会自动展开为28×28=784的listtensor : shape… 查看全文 >

duanzhengyi · 5 个月前



卷积神经网络(CNN)的参数优化方法

著名：本文是从 Michael Nielsen的电子书Neural Network and Deep Learning的深度学习那一章… 查看全文 >

元峰 · 1 年前 · 发表于 机器学习极简主义