

教程 | Kaggle网站流量预测任务第一名解决方案：从模型到代码详解时序预测

2017-12-05 机器之心

选自GitHub

作者：Artur Suilin

机器之心编译

参与：蒋思源、路雪、黄小天

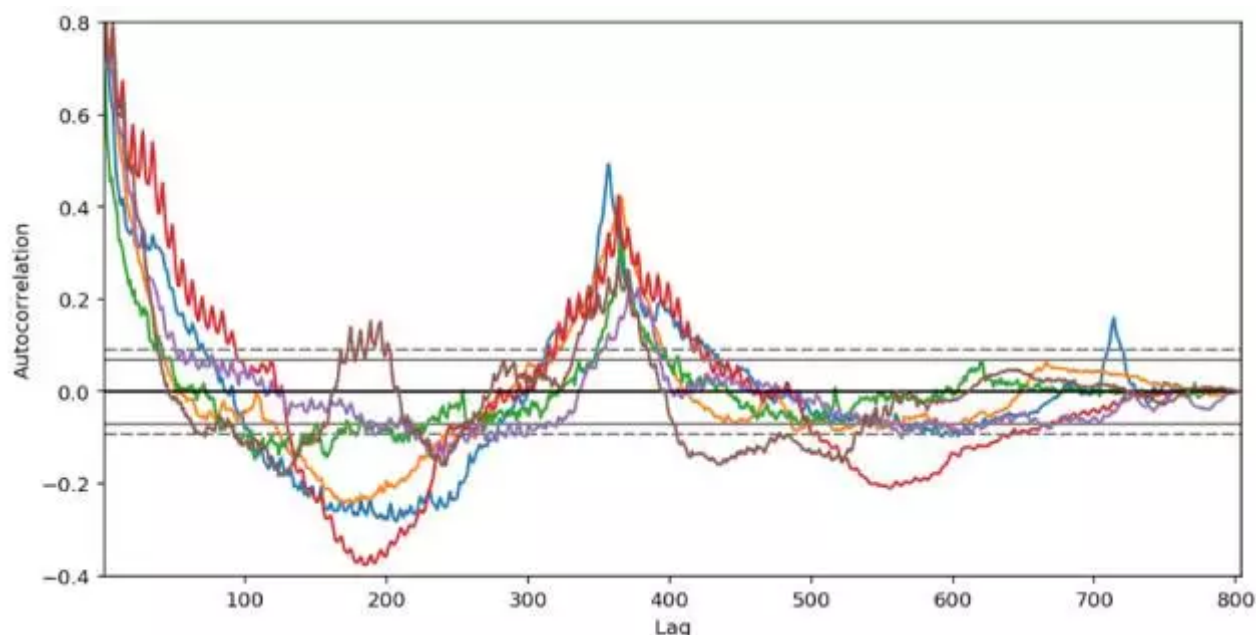
近日，Artur Suilin 等人发布了 Kaggle 网站流量时序预测竞赛第一名的详细解决方案。他们不仅公开了所有的实现代码，同时还详细解释了实现的模型与经验。机器之心简要介绍了他们所实现的模型与经验，更详细的代码请查看 GitHub 项目。

GitHub 项目地址：<https://github.com/Arturus/kaggle-web-traffic>

下面我们将简要介绍 Artur Suilin 如何修正 GRU 以完成网站流量时序预测竞赛。

预测有两个主要的信息源：

1. 局部特征。我们看到一个趋势时，希望它会继续（自回归模型）朝这个趋势发展；看到流量峰值时，知道它将逐渐衰减（滑动平均模型）；看到假期交通流量增加，就知道以后的假期也会出现流量增加（季节模型）。
2. 全局特征。如果我们查看自相关（autocorrelation）函数图，就会注意到年与年之间强大的自相关和季节间的自相关。



我决定使用 RNN seq2seq 模型进行预测，原因如下：

1. RNN 可以作为 ARIMA 模型的自然扩展，但是比 ARIMA 更灵活，更具表达性。
2. RNN 是非参数的，大大简化了学习。想象一下对 145K 时序使用不同的 ARIMA 参数。
3. 任何外源性的特征（数值或类别、时间依赖或序列依赖）都可以轻松注入该模型。
4. seq2seq 天然适合该任务：我们根据先前值（包括先前预测）的联合概率（joint probability）预测下一个值。使用先前预测可保持模型稳定，因为误差会在每一步累积，如果某一步出现极端预测，则有可能毁了所有后续步的预测质量。
5. 现在的深度学习出现了太多的炒作。

特征工程

RNN 足够强大来发现和学习自身特征。模型的特征列表如下：

- pageviews：原始值经过 $\log_{1p}()$ 的转换得到几乎正态的时序内值分布，而不是偏态分布。
- agent, country, site：这些特征从网页 url 中提取，然后经过 One-Hot 编码。
- day of week：捕捉每周的季节效应。
- year-to-year autocorrelation, quarter-to-quarter autocorrelation：捕捉各年、各季度的季节效应。
- page popularity：高流量和低流量页面具有不同的流量变化模式，该特征（pageviews 的中间值）帮助捕捉流量规模。pageviews 特征丢失了规模信息，因为每个 pageviews 序列被独立归一化至零均值和单位方差。

- lagged pageviews : 之后将具体介绍。

特征预处理

所有特征（包括 One-Hot 编码的特征）被归一化至零均值和单位方差。每个 pageviews 序列被独立归一化。

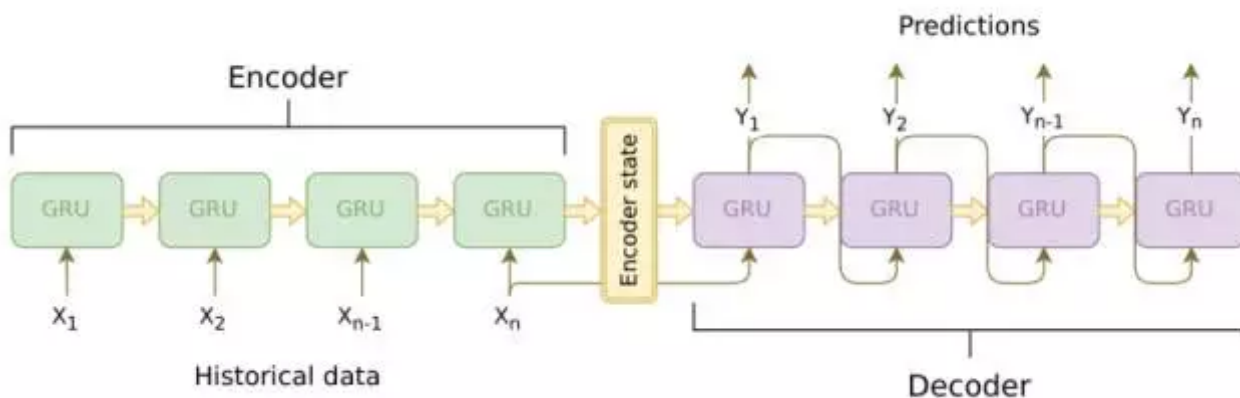
时间依赖特征（自相关性、国家等）被「拉伸」至时序长度，即每天重复使用 `tf.tile()` 命令。

模型在来自初始时序的随机固定长度样本上进行训练。例如，如果初始时序长度是 600 天，我们使用 200 天的样本进行训练，那么我们可以在前 400 天中随意选择开始采样的样本。

该采样工作是一种有效的数据增强机制：训练代码在每一步随机选择每次时序的开始点，生成无限量的几乎不重复的数据。

模型的核心技术

模型主要由两部分组成，即编码器和解码器。



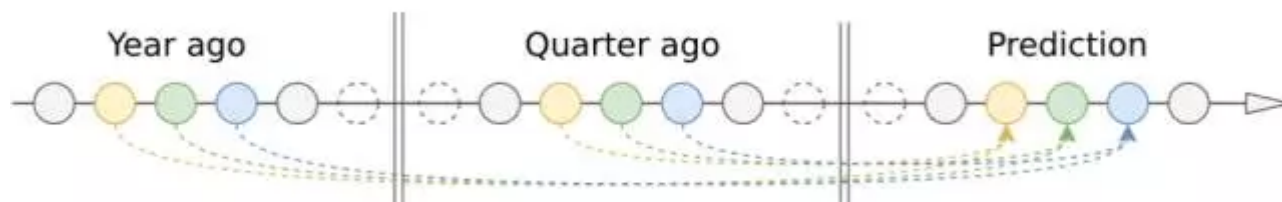
编码器为 cuDNN GRU，cuDNN 要比 TensorFlow 的 RNNCells 快大约 5 到 10 倍，但代价就是使用起来不太方便，且文档也不够完善。

解码器为 TF GRUBlockCell，该 API 封装在 `tf.nn.nn_cell.LSTMCell` 中。循环体内的代码从上一步获得预测，并加入到当前时间步的输入特征中。

处理长时间序列

LSTM/GRU 对于相对较短的序列（100-300 项以内）来说是非常好的解决方案。但对于较长的序列来说，LSTM/GRU 仍然有效，只不过会逐渐遗忘较早时间步所包含的信息。Kaggle 竞赛的时间序列长达 700 多天，所以我们需要找一些方法来「加强」GRU 的记忆力。

我们第一个方法先是考虑使用一些注意力机制。注意力机制可以将过去较长距离的有用信息保留到当前 RNN 单元中。对于我们的问题，最简单高效的注意力方法是使用固定权重的滑动窗口注意力机制。它在较长距离的过去时间步上有两个重要的点（考虑长期的季节性），即 1 年前和 1 个季度前。



我们可以采用 `current_day - 365` 和 `current_day - 90` 这两个时间点的编码器输出，并将它们馈送到全连接层以降低维度，并将结果加入到解码器的输入特征中。这个解决方案虽然简单却大大降低了预测误差。

随后我们将重要的点与它们的近邻求均值，并借此减少噪声和补偿不均匀的间隔（闰年和不同长度的月份）：

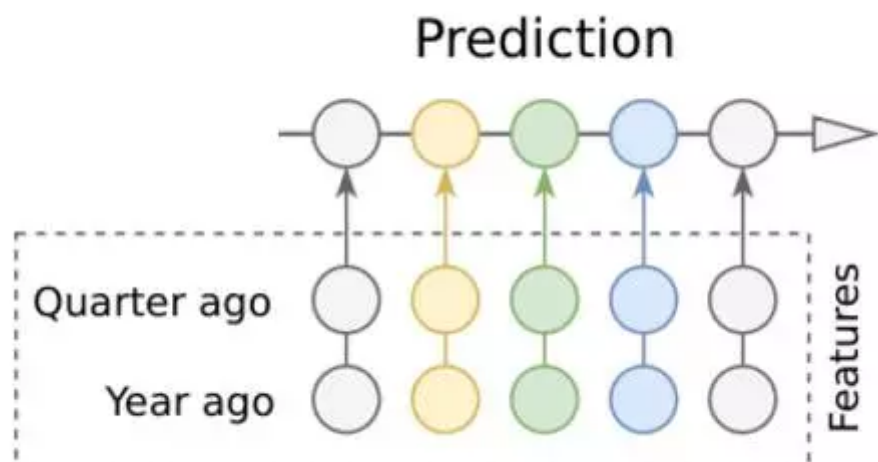
$$\text{attn_365} = 0.25 * \text{day_364} + 0.5 * \text{day_365} + 0.25 * \text{day_366}.$$

但随后我们意识到 0.25、0.5、0.25 是一个一维卷积核（`length=3`），我们可以自动学习更大的卷积核以检测过去重要的点。

最后，我们构建了一个非常大的注意力机制，它会查看每一个时间序列的「指纹」（指纹由较小的卷积网络产生），并决定应该注意哪些点和为较大卷积核生成权重。这个应用于解码器输出的较大卷积核会为每一个预测的日期生成一个注意力特征。虽然最后没有使用这种方法，但这个注意力机制仍然保留在代码中，读者可以在模型代码中找到它。

注意，我们并没有使用经典的注意力方案（Bahdanau 或 Luong 注意力机制），因为经典的注意力机制应该在每个预测步上使用所有的历史数据点从头开始计算，因此这种方法对于较长时间序列（约两天的天数）来说太耗时间了。所以我们的方案将会对所有数据点进行一次卷积，对所有预测时间步使用相同的注意力权重（这也是缺点），这样的方案计算起来要快很多。

因为我们对注意力机制的复杂度感到不太满意，因此我们试图完全移除注意力机制，并将一年前、半年前、一季前重要的数据点作为编码器和解码器的附加特征。这样的结果是非常令人惊讶的，甚至在预测质量方面都要比带注意力机制的模型略胜一筹。因此我们最好的公开分数都是仅使用滞后（lagged）数据点实现的，它们都没有使用注意力机制。



滞后数据点另一个重要的优势是，模型可以使用更短的编码器而不需要担心损失过去的信息，因为这些信息现在明确地包含在特征中。在采用这种方法后，即使我们编码器的长度是 60 到 90 天，结果也是完全可以接受的，而以前需要 300-400 天的长度才能获得相同的性能。此外，更短的编码器就等于更快速的训练和更少的信息损失。

损失和正则化

SMAPE（竞赛用的目标损失函数）因其在零值周围不稳定的行为而无法直接使用（当真值为零的时候，损失函数是阶跃函数；预测值也为零的时候，则损失函数不确定）。

我使用经平滑处理的可微 SMAPE 变体，它在所有实数上都表现良好：

```
epsilon = 0.1
summ = tf.maximum(tf.abs(true) + tf.abs(predicted) + epsilon, 0.5 + epsilon)
smape = tf.abs(predicted - true) / summ * 2.0
```

另一个选择是在 $\log_{1p}(\text{data})$ 上的 MAE 损失函数，它很平滑，且训练目标与 SMAPE 非常接近。

最终预测取最接近的整数，负面预测取零。

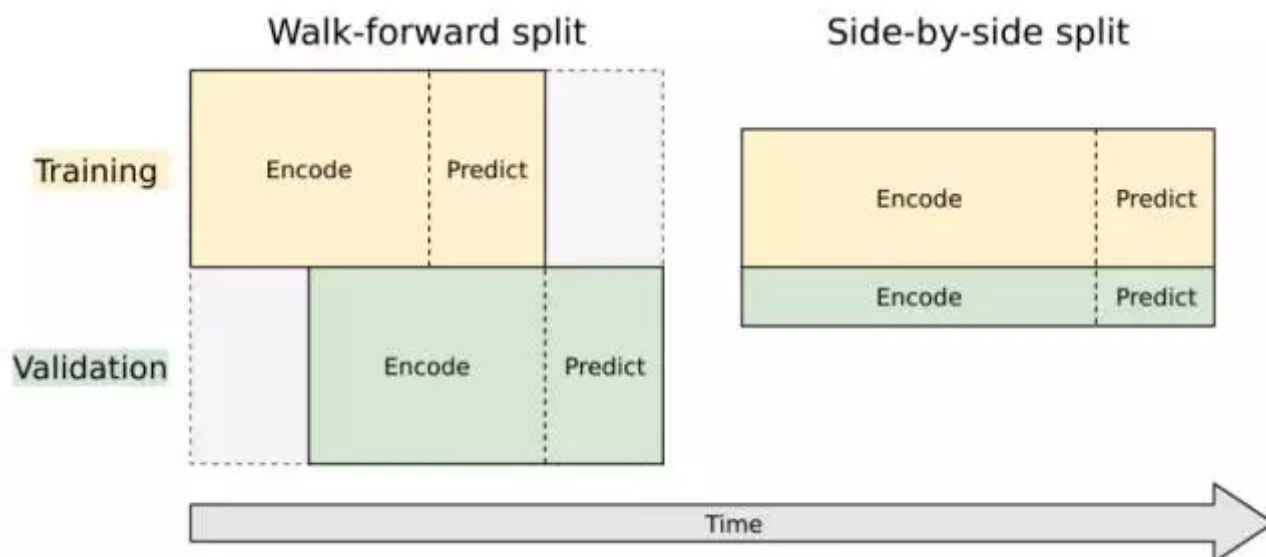
我尝试使用论文《Regularizing RNNs by Stabilizing Activations》中经正则化的 RNN 激活值，因为 cuDNN GRU 的内部权重无法直接正则化（也可能是我没有找到正确的方法）。稳性损失（Stability loss）不起作用，激活损失可以为较小损失权重（ $1e-06..1e-05$ ）带来少许改进。

训练和验证

我使用 COCOB 优化器（详见论文《Training Deep Networks without Learning Rates Through Coin Betting》）结合梯度截断进行训练。COCOB 尝试预测每个训练步的最优学习率，因此我完全不必调整学习率。它的收敛速度也比传统的基于动量的优化器快得多，尤其是在第一个 epoch 上，可以让我及早停止不成功的实验。

有两种方式可以将时序分割为训练和验证数据集：

1. Walk-forward 分割。这实际上不是分割：我们在完整数据集上训练和验证，但使用不同的时间跨度。验证用的时间跨度比训练用时间跨度前移一个预测间隔。
2. Side-by-side 分割。这是主流机器学习传统的分割模型。数据集被分割成两个独立的部分，一个用于训练，另一个用于验证。



两种方式我都试了，但对于这个任务来说 Walk-forward 更好，因为它与竞赛目标直接相关：使用历史值预测未来值。但是该分割破坏了时序结尾的数据点，使得训练准确预测未来的模型变得困难。

具体来说：比如，我们有 300 天的历史数据，想预测接下来 100 天的数据。如果我们选择 walk-forward 分割，我们必须使用前 100 天的数据用于真实训练，后面 100 天的数据用于训练模式的预测（运行解码器、计算损失），再后面 100 天的数据用于验证，最后 100 天用于对未来值真正进行预测。因此，我们实际上可以使用 1/3 的数据点来训练，最后一个训练数据点和第一个预测数据点之间隔了 200 天。间隔太大了，因为一旦我们离开某个训练数据，预测质量将出现指数级下降（不确定性增加）。使用 100 天差距训练的模型预测质量相对较好。

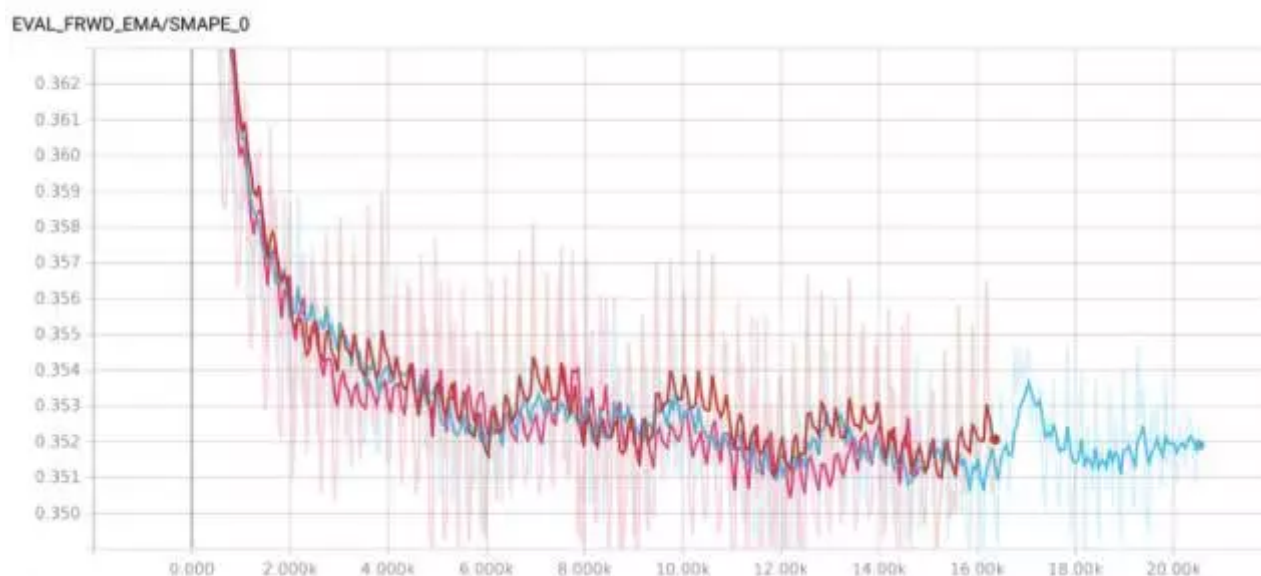
Side-by-side 分割所需要的计算力更少，因为它在端点时并不会消耗数据点。但是对于我们的数据，模型在验证集上的性能与在训练集上的性能是强相关的，并且与将来的实际模型性能几乎不具有相关性。换言之，并行分割对于我们的问题基本上是没有作用的，它只是复制了在训练数据集上观察到的模型损失。

我仅使用验证集（带有前向分步分割）进行模型调优，预测未来数值的最终模型只是在盲目的模式中进行训练，没有使用任何验证集。

降低模型方差

优于强噪音数据的输入，模型不可避免地具有高方差。坦白讲，我很惊讶 RNN 居然从噪音数据中学习到了东西。

在不同 seed 上训练的相同模型具有不同的表现，有时模型甚至在「不幸」的 seed 上变得发散。训练期间，表现也会逐步地发生很大波动。依靠纯粹的运气很难赢得比赛，因此我决定采取行动降低方差。



1. 我不知道哪个训练步骤最适合预测未来（但前数据的验证结果与未来数据的结果只有弱相关关系），所以我不能使用提前停止。但是我知道近似区域，其中模型（可能）进行了充分训练，但（可能）没有开始过拟合。我决定把这个最佳区域设置为 10500 到 11500 次迭代区间内，并且从这个区域的每第 10000 个步骤保存 10 个检查点。
2. 相似地，我决定在不同的 seed 上训练 3 个模型，并从每个模型中保存检查点。因此我一共有 30 个检查点。
3. 降低方差、提升模型性能的一个众所周知的方法是 ASGD（SGD 平均）。它很简单，并在 TensorFlow 中得到很好的支持。我们必须在训练期间保持网络权重的移动平均值，并在推断中使用这些平均权重，而不是原来的权重。


三个模型的结合表现不错（在每个检查点上使用平均模型权重的 30 个检查点的平均预测）。我在排行榜上（针对未来数据）获得了相较于历史数据上的验证大致相同的 SMAPE 误差。

理论上讲，你也可以把前两种方法用作集成学习，但我主要用其降低方差。

超参数调节

很多模型参数（层的数量、深度，激活函数，dropout 系数等）能够（并且应该）被调节从而获得更优的模型表现。手动调节乏味且费时，所以我决定自动化该过程，并使用 SMAC3 搜索超参数。下面是 SMAC3 的一些优势：

- 支持条件参数（例如，为每层联合调节层数和 dropout；如果 $n_layers > 1$ ，第二层上的 dropout 将被调节）
- 明确处理模型方差。SMAC 在不同种子上训练每个模型的若干个实例，如果实例在相同种子上训练还要对比模型。如果它在所有相同种子上优于另一个模型，则该模型获胜。

与我的期望相反，超参数搜索并没有建立定义明确的全局最小。所有的最佳模型大致具有相同的性能，但参数不同。可能 RNN 模型对于这个任务来说太具有表现力了，并且最好的模型得分更多依赖于模型架构上的数据信噪比。不管怎样，最好的参数设置依然可以在 `hparams.py` 文件中找到。  SYNCED

原文链接：https://github.com/Arturus/kaggle-web-traffic/blob/master/how_it_works.md

本文为机器之心编译，转载请联系本公众号获得授权。

✂️-----

加入机器之心（全职记者/实习生）：hr@jiqizhixin.com

投稿或寻求报道：content@jiqizhixin.com

广告&商务合作：bd@jiqizhixin.com