炼丹日志

这是哈赛第一次在结构化数据中炼丹，如果下面表述有不正确的地方还望各位大佬们海涵

Version 1

模型主体为 LSTM+DNN：其中 LSTM 的输入为每个 event 的 hit 的前 x 个与后 x 个的时间 gap 的序列，将 LSTM 的输出加上 self attention 之后与数据的其他原始特征 concrete 之后作为 DNN 的输入最后输出结果。效果十分不理想，最终线上 12+。

Version2

将输入的时间 gap 序列改为原始的时间序列之后，线上有不小的提升，但效果任然不好，线上 30+。

Version3

在 booster 模型的搭建过程中发现最强的序列特征并不是 t 序列特征而是 q 序列特征，故此时多加入了一个 q 序列的输入，即现在有两个 LSTM 输入和一个 DNN 输入，其余的模型构造方法同之前的 Version，但线上并没有提升，最终线上 30+。

Version4

在 booster 模型的搭建中发现如果按照原始数据集给的顺序做序列特征的时候其结果往往不是最佳的，故开始考虑按

一定排序规则排序后在进行序列特征的构造。经过在 booster 模型的测试中发现了四个特征性比较强的序列构造方式

1. 按 q 排序以后，取 t 列。
2. 按 terror q 排序还是取 t 列。
3. 按 hit 排序取 t 列。
4. 按 eventid q 排序取 q 列。

此时我们构造了四个序列，但此时按以前的思路来说需要构造四个 LSTM 来分别对四个序列建模，此时大佬队友说如果 LSTM 的 dim 太小的话学到的东西会比较有限，于是参考他的建议之后将四个序列合并之后送入一个 LSTM 当中，然后将其输出做 self attention 之后与 DNN 输入合并之后送入 DNN 最终输出结果，此时模型效果得到了显著的提高，最终线上 57+。

Version5

由于序列数的增加导致了 LSTM 的待拟合参数量急剧上升，故考虑了使用一维的 conv 来代替 LSTM，这个版本迭代的时间最长，主要尝试了如下模型：

1. 一维浅层 ResNet：线上 58+
2. 一维浅层 DenseNet：线上 60+
3. 一维浅层 ResNet+FocalLoss：线下训练效果不佳未训练完成
4. 一维浅层 DenseNet+FocalLoss：线下训练效果不佳未

训练完成

由此发现 DenseNet 效果比 SeResNet 效果更好，且均高于 LSTM，虽然本题数据分布不是很均衡，但是训练集测试集分布是一致的使用 FocalLoss 会打破这种微妙的平衡，在训练时就发现效果不如不加 FocalLoss。这总情况下使用 binary_crossentropy 即可保证其分布的一致性。

Version6

由于现在我们把模型的构建的主要精力放在了构造序列上，此时考虑将序列做一些预处理，最自然的想法自然是对序列去噪，所以对 t 序列与 q 序列进行去噪，以 t 序列举例，其具体实现方法为将序列按 t 排序后对 t 序列做 rolling 后取 mean，这样便使每个 t 的值都考虑到了其周围值的影响。对 q 也是类似的处理。

此时在构建模型上就犯难了，是直接使用处理后的序列，还是都使用呢，最后考虑都试一遍。

首先是两套序列都使用的情况，这时我并没有将其合并之后直接送入模型训练，我将这里分为两个输入，每套序列都送入一个 DenseNet 训练最后将两个模型的结果与 DNN 输入的结果合并送入 DNN，此时由于多训练了一个 DenseNet 导致其训练时长大大加长，并且在较早期就发现 valid 的 loss 有连续多个 epoch 回升的情况，至于为啥出现这个原因我还没想明白，但这个现象已经直接否认了这套方案。

其次是只是用降噪后的序列进行训练，在训练近乎平稳之后发现其 valid 的 loss 不如原始数据的 loss，故也摒弃了此套方案，分析原因后发现其噪声出现的范围较为集中，在 rolling 之后弱化了其噪声的序列特征，故在使用 rolling 之后其结果不如原始序列特征。

Version7

增加了三个新的序列：

1. 按 terror，q 排序后取 t 列。
2. 按 event_id，q 排序后取 dis 列
3. 按 terror，q 排序后取 dis 列

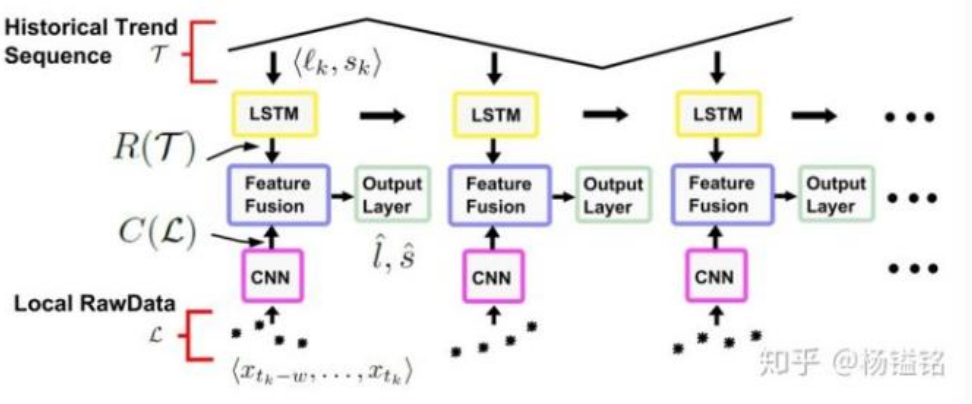
其中 dis 为每个 event 的每个 hit 所对应的(x,y)与每个 event 独有的(xcmc,ycmc)之间的距离。即构造了新的距离序列。模型仍采用 DenseNet，线上有了进一步的提升，最终线上 61.2+。

此时该模型线下的 AUC 已经达到 0.99951，对比于 booster 模型其线下 AUC 为 0.99978 所对应线上的分数为 68+，AUC 为 0.9995 左右的时候线上已经是达到 65+。这说明了我们的网络有过拟合的倾向，后续不管是在设计网络还是构造新序列的时候需要考虑到过拟合的问题。

Version8

二、TreNet模型

TreNet的主要是想就是结合LSTM和CNN，发挥两者对数据不同方面的表示能力，然后学习综合的表示用来进行预测。形式化表达即为 $< \tilde{l}, \tilde{s} >= f(R(T), C(L))$ 。R(T)通过针对趋势序列T训练一个LSTM来捕捉数据中存在的长期依赖。C(L)通过对L中局部点的集合使用CNN抽取特征。



feature fusion层结合LSTM和CNN的表示得到一个联合的特征。通过将R(T)和C(L)映射到同一个特征空间，然后相加之后经过激活函数。最后output层就是一个全连接层。如下式：

$$\begin{aligned}\langle \hat{l}, \hat{s} \rangle &= f(R(\mathcal{T}),\, C(\mathcal{L})) \\ &= \boldsymbol{W^o} \cdot \underbrace{\boldsymbol{\phi}(\boldsymbol{W^r} \cdot R(\mathcal{T}) + \boldsymbol{W^c} \cdot C(\mathcal{L}))}_{feature\ fusion} + \boldsymbol{b^o}\end{aligned}$$

模型的损失函数使用均方误差，外加一个正则项，即为：

$$J(\boldsymbol{W}, \boldsymbol{b}; \mathcal{T}, \mathcal{X}) = \frac{1}{|\mathcal{T}|} \sum_{k=1}^{|\mathcal{T}|} \left[(\hat{l}_k - l_k)^2 + (\hat{s}_k - s_k)^2 \right] + \lambda \|\boldsymbol{W}\|_2$$

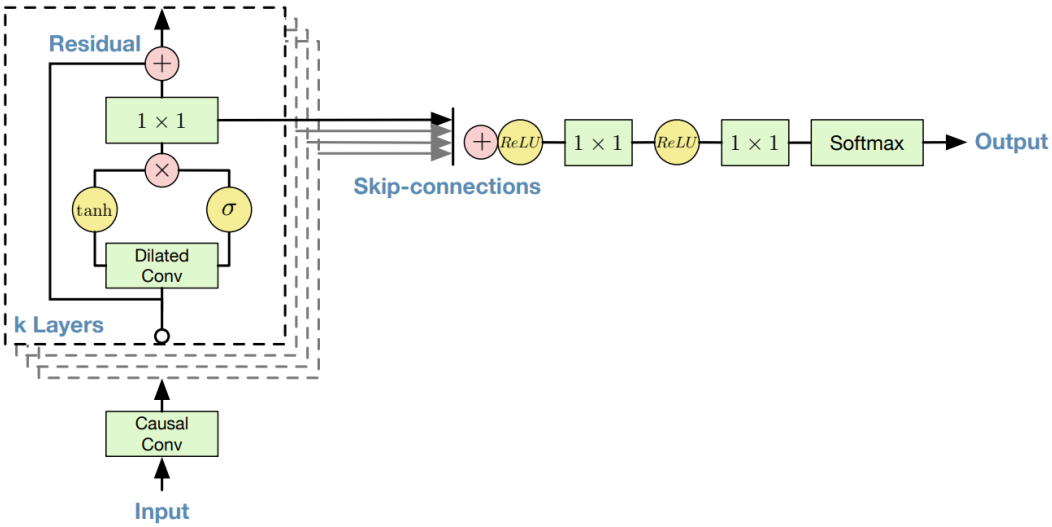
参考上述结构将输入的序列同时送入 cnn 与 lstm 当中然后将两个输出 concrete 之后和 nninput 一起输入 dnn 中，这个结构综合了 cnn 提取的空间特征与 lstm 所提取的时间特征，在特征输入不变的前提下线上有将近 2 分的提升，最终线上 62.9+

Version9

仔细排查代码发现在构造 nn_input 时忘记去掉 index，这会极大的影响其在测试集上的表现，现在去掉了 index 并丰富了 nn_input 的特征主要添加了两大类特征，一类是对 event groupby 之后的 t 特征，另一类是对 event groupby 之后的 dis 特征。最终线上 63.8+

Version10

引入 wavenet 来处理序列数据，wavenet 的引入增大了其在时域上的感知能力。其核心结构如下：



wavenet 主要通过多尺度的带孔卷积对序列进行多尺度的特征提取，最终线上达到了 65 分。