时间序列分析（5） TCN

随风
大数据、人工智能

关注他

228 人赞同了该文章

1 前言

在上一章中，我们介绍了深度学习模型 RNN/LSTM，通过神经网络对时间序列建模。在目前的 RNN 结构中， LSTM 与 GRU 是主要的模型，tensorflow 已经提供了对 GRU 的支持，可以自行修改上一章的代码。本章介绍另一个深度学习模型，时间卷积网络（Temporal Convolutional Network），利用卷积网络对时间序列建模。时间序列分析（4）RNN/LSTM：

随风：时间序列分析（4）RNN/LSTM
zhuanlan.zhihu.com

语言：python3

数据集：余额宝在2014-03-01~2014-08-31期间每日申购的总金额（数据来自天池大赛）

数据下载地址：tianchi.aliyun.com/comp

实验表明，RNN 在几乎所有的序列问题上都有良好表现，包括语音/文本识别、机器翻译、手写体识别、序列数据分析（预测）等。

在实际应用中，RNN 在内部设计上存在一个严重的问题：由于网络一次只能处理一个时间步长，后一步必须等前一步处理完才能进行运算。这意味着 RNN 不能像 CNN 那样进行大规模并行处理，特别是在 RNN/LSTM 对文本进行双向处理时。这也意味着 RNN 极度地计算密集，因为在整个任务运行完成之前，必须保存所有的中间结果。

CNN 在处理图像时，将图像看作一个二维的“块”（m*n 的矩阵）。迁移到时间序列上，就可以将序列看作一个一维对象（1*n 的向量）。通过多层网络结构，可以获得足够大的感受野。这种做法会让 CNN 非常深，但是得益于大规模并行处理的优势，无论网络多深，都可以进行并行处理，节省大量时间。这就是 TCN 的基本思想。

2017年 Google、Facebook 相继发表了研究成果，其中一篇叙述比较全面的论文是 "An Empirical Evaluation of Generic Convolutional and Recurrent Networks"。业界将这一新架构命名为时间卷积网络（TCN）。

论文地址：arxiv.org/pdf/1803.0127

2 CNN 扩展技术

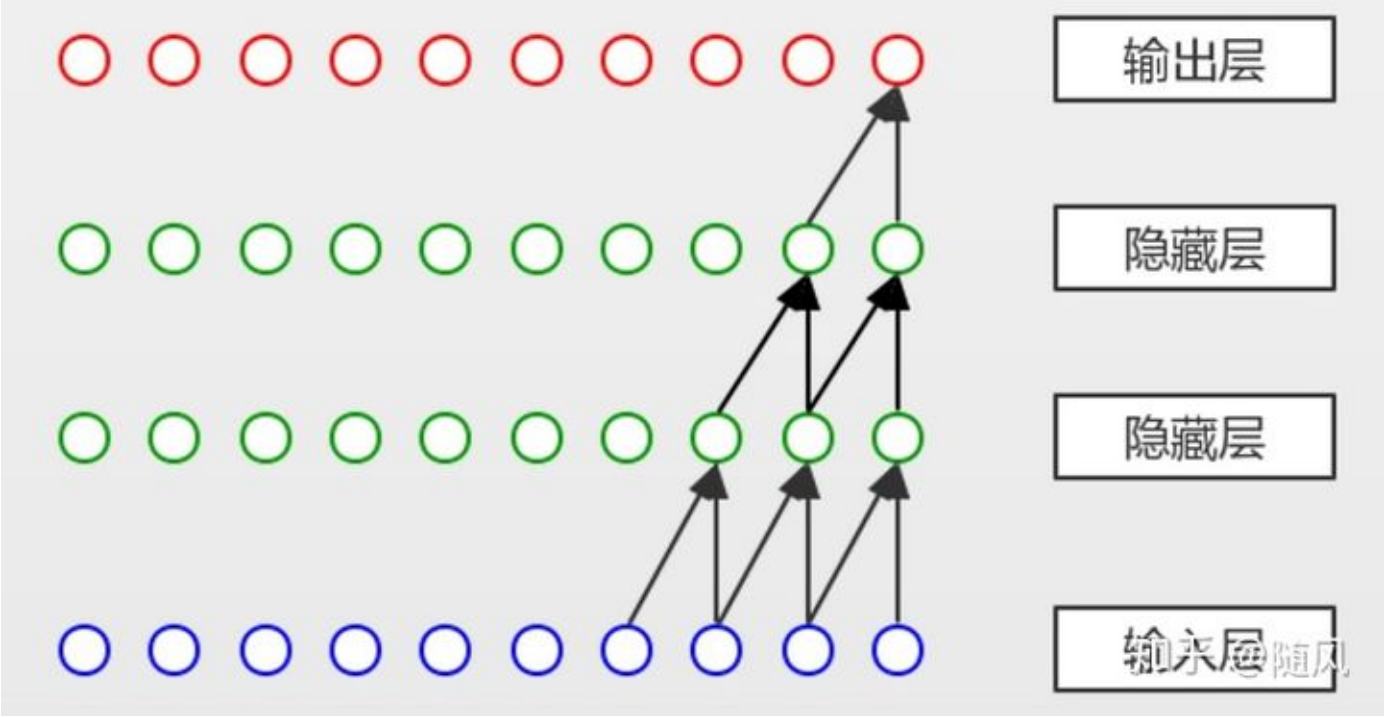
TCN 模型以 CNN 模型为基础，并做了如下改进：

适用序列模型：因果卷积（Causal Convolution）
记忆历史：空洞卷积/膨胀卷积（Dilated Convolution），残差模块（Residual block）

下面将分别介绍 CNN 的扩展技术。

2.1 因果卷积（Causal Convolution）

因为要处理序列问题（时序性），就必须使用新的 CNN 模型，这就是因果卷积。序列问题可以转化为：根据 $x_1, x_2, \ldots, x_t$ 去预测 $y_1, y_2, \ldots, y_t$ 。下面给出因果卷积的定义，滤波器 $F = (f_1, f_2, \ldots, f_K)$ ，序列 $X = (x_1, x_2, \ldots, x_T)$ ，在 x_t 处的因果卷积为：$(F * X)_{(x_t)} = \sum_{k=1}^{K} f_k x_{t-K+k}$ 。下图为一个因果卷积的实例，假设输入层最后两个节点分别为 x_{t-1}, x_t ，第一层隐藏层的最后一个节点为 y_t ，滤波器 $F = (f_1, f_2)$ ，根据公式有 $y_t = f_1 x_{t-1} + f_2 x_t$ 。



因果卷积

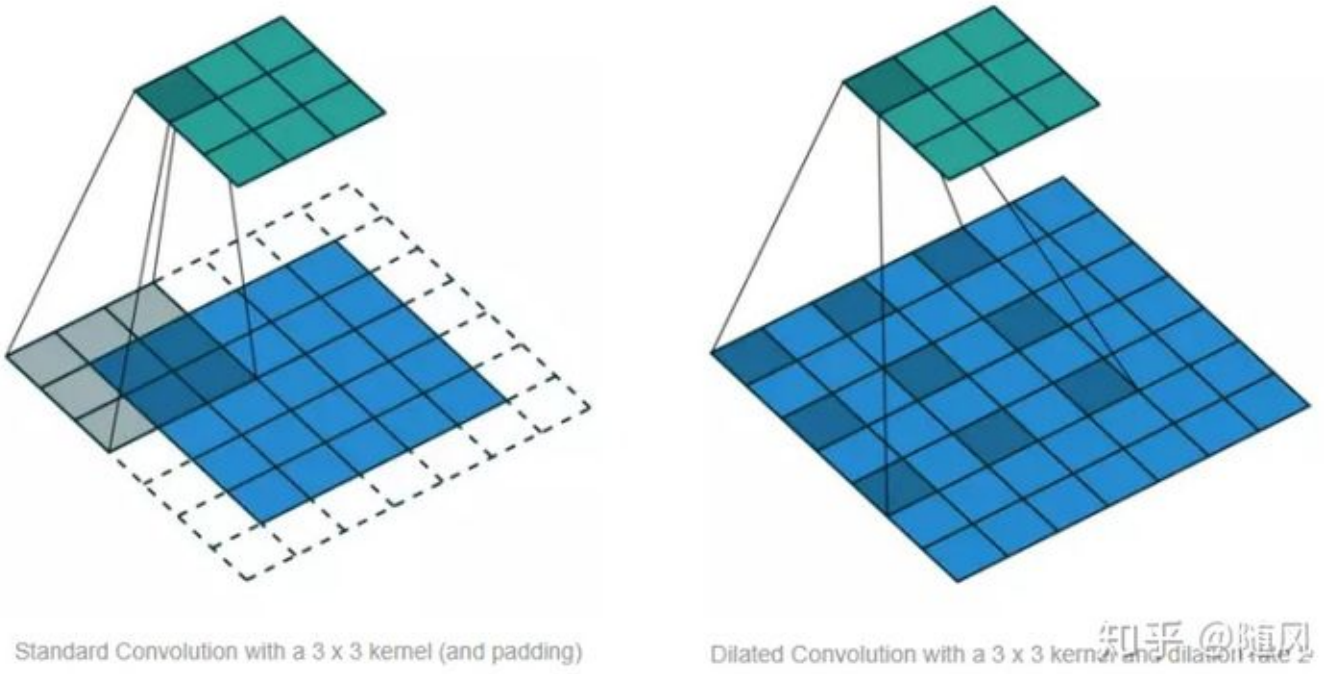
因果卷积有两个特点：

不考虑未来的信息。给定输入序列 $x_1, \cdots, x_T$ ，预测 $y_1, \cdots, y_T$ 。但是在预测 y_t 时，只能使用已经观测到的序列 $x_1, \cdots, x_t$ ，而不能使用 $x_{t+1}, x_{t+2}, \cdots$ 。
追溯历史信息越久远，隐藏层越多。上图中，假设我们以第二层隐藏层作为输出，它的最后一个节点关联了输入的三个节点，即 x_{t-2}, x_{t-1}, x_t ；假设以输出层作为输出，它的最后一个节点关

联了输入的四个节点，即 $x_{t-3}, x_{t-2}, x_{t-1}, x_t$ 。

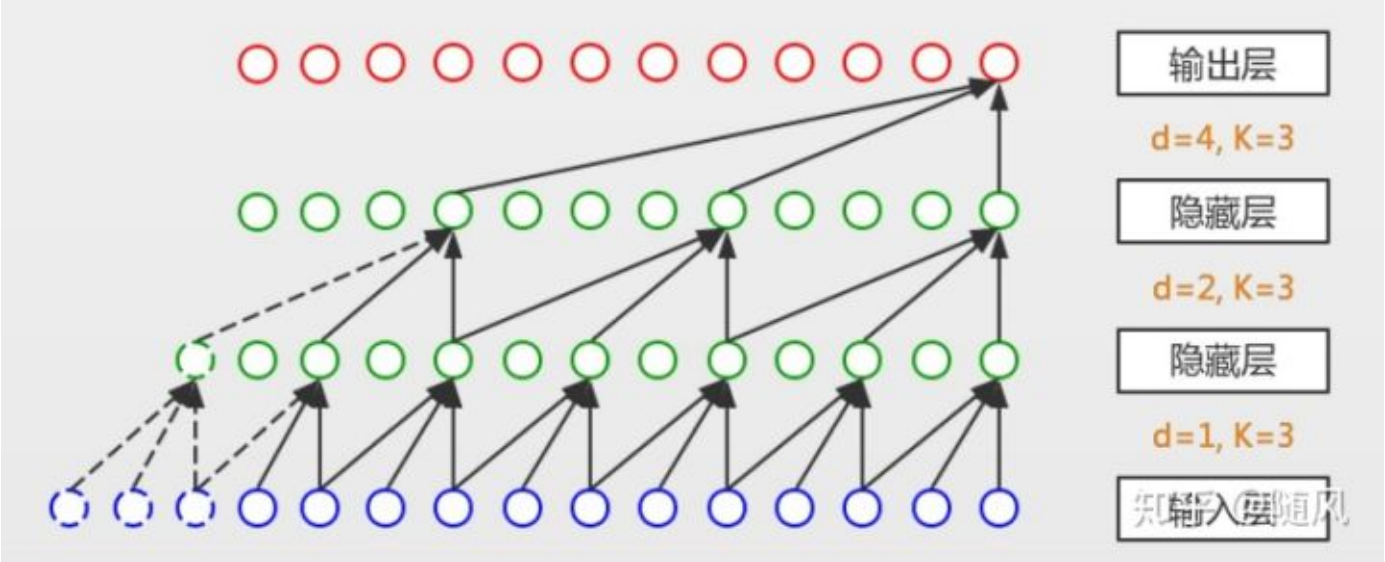
2.2 空洞卷积/膨胀卷积（Dilated Convolution）

标准的 CNN 通过增加 pooling 层来获得更大的感受野，而经过 pooling 层后肯定存在信息损失的问题。空洞卷积是在标准的卷积里注入空洞，以此来增加感受野。空洞卷积多了一个超参数 dilation rate，指的是 kernel 的间隔数量（标准的 CNN 中 dilatation rate 等于 1）。空洞的好处是不做 pooling 损失信息的情况下，增加了感受野，让每个卷积输出都包含较大范围的信息。下图展示了标准 CNN （左）和 Dilated Convolution （右），右图中的 dilatation rate 等于 2 。



下面给出空洞卷积的定义，滤波器 $F = (f_1, f_2, \ldots, f_K)$ ，序列 $X = (x_1, x_2, \ldots, x_T)$ ，在 x_t 处的 dilatation rate 等于 d 的空洞卷积为：

$(F *_d X)_{(x_t)} = \sum_{k=1}^{K} f_k x_{t-(K-k)d}$ ，下图为一个空洞卷积的实例，假设第一层隐藏层最后五个节点分别为 $x_{t-4}, x_{t-3}, x_{t-2}, x_{t-1}, x_t$ ，第二层隐藏层的最后一个节点为 y_t ，滤波器 $F = (f_1, f_2, f_3)$ ，根据公式有 $y_t = f_1 x_{t-2d} + f_2 x_{t-d} + f_3 x_t, (d = 2)$ 。



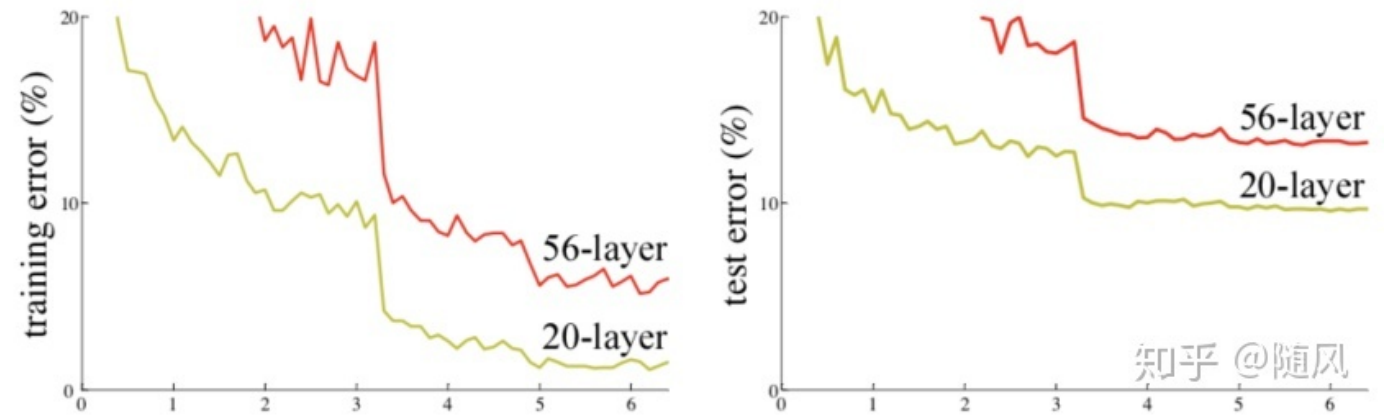
空洞卷积

空洞卷积的感受野大小为 $(K-1)d+1$ ，所以增大 K 或 d 都可以增加感受野。在实践中，通常随网络层数增加， d 以 2 的指数增长，例如上图中 d 依次为 $1, 2, 4$ 。

2.3 残差模块（Residual block）

CNN 能够提取 low/mid/high-level 的特征，网络的层数越多，意味着能够提取到不同 level的特征越丰富。并且，越深的网络提取的特征越抽象，越具有语义信息。

如果简单地增加深度，会导致梯度消失或梯度爆炸。对于该问题的解决方法是权重参数初始化和采用正则化层（Batch Normalization），这样可以训练几十层的网络。解决了梯度问题，还会出现另一个问题：网络退化问题。随着网络层数的增加，在训练集上的准确率趋于饱和甚至下降了。注意这不是过拟合问题，因为过拟合会在训练集上表现的更好。下图是一个网络退化的例子，20 层的网络比 56 层的网络表现更好。



网络退化

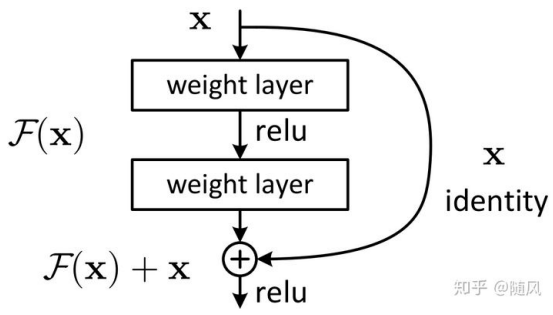
理论上 56 层网络的解空间包括了 20 层网络的解空间，因此 56 层网络的表现应该大于等于20 层网络。但是从训练结果来看，56 层网络无论是训练误差还是测试误差都大于 20 层网络（这也说明了为什么不是过拟合现象，因为 56 层网络本身的训练误差都没有降下去）。**这是因为虽然 56**

层网络的解空间包含了 20 层网络的解空间，但是我们在训练中用的是随机梯度下降策略，往往得到的不是全局最优解，而是局部最优解。显然 56 层网络的解空间更加的复杂，所以导致使用随机梯度下降无法得到最优解。

假设已经有了一个最优的网络结构，是 18 层。当我们设计网络结构时，我们并不知道具体多少层的网络拥有最优的网络结构，假设设计了 34 层的网络结构。那么多出来的 16 层其实是冗余的，我们希望训练网络的过程中，模型能够自己训练这 16 层为恒等映射，也就是经过这16 层时的输入与输出完全一样。但是往往模型很难将这 16 层恒等映射的参数学习正确，这样的网络一定比最优的 18 层网络表现差，这就是随着网络加深，模型退化的原因。

因此解决网络退化的问题，就是解决如何让网络的冗余层产生恒等映射（深层网络等价于一个浅层网络）。通常情况下，让网络的某一层学习恒等映射函数 $\boldsymbol{H(x)=x}$ 比较困难，但是如果我们把网络设计为 $\boldsymbol{H(x)=F(x)+x}$ ，我们就可以将学习恒等映射函数转换为学习一个残差函数 $\boldsymbol{F(x)=H(x)-x}$ ，只要 $\boldsymbol{F(x)=0}$ ，就构成了一个恒等映射 $\boldsymbol{H(x)=x}$ 。在参数初始化的时候，一般权重参数都比较小，非常适合学习 $\boldsymbol{F(x)=0}$ ，因此拟合残差会更加容易，这就是残差网络的思想。

下图为残差模块的结构，该模块提供了两种选择方式，也就是 identity mapping（即 $\boldsymbol{x}$ ，右侧“弯弯的线"，称为 shortcut 连接） 和 residual mapping（即 $\boldsymbol{F(x)}$ ），如果网络已经到达最优，继续加深网络，residual mapping 将被 push 为 0，只剩下 identity mapping，这样理论上网络一直处于最优状态了，网络的性能也就不会随着深度增加而降低了。



残差模块

这种残差模块结构可以通过前向神经网络 + shortcut 连接实现。而且 shortcut 连接相当于简单执行了同等映射，不会产生额外的参数，也不会增加计算复杂度，整个网络依旧可以通过端到端的反向传播训练。

上图中残差模块包含两层网络。实验证明，残差模块往往需要两层以上，单单一层的残差模块 并不能起到提升作用。shortcut 有两种连接方式：

（1）identity mapping 同等维度的映射（ $\boldsymbol{F(x)}$ 与 $\boldsymbol{x}$ 维度相同）：

$$\boldsymbol{F(x)=W_2\sigma(W_1x+b_1)+b_2}\,,\quad \boldsymbol{H(x)=F(x)+x}$$

（2）identity mapping 不同维度的映射（ $F(x)$ 与 x 维度不同）：

$$F(x) = W_2\sigma(W_1x + b_1) + b_2，\ H(x) = F(x) + W_sx$$

以上是基于全连接层的表示，实际上残差模块可以用于卷积层。加法变为对应 channel 间的两个 feature map 逐元素相加。

设计 CNN 网络的规则：

（1）对于输出 feature map 大小相同的层，有相同数量的 filters，即 channel 数相同；

（2）当 feature map 大小减半时（池化），filters 数量翻倍；

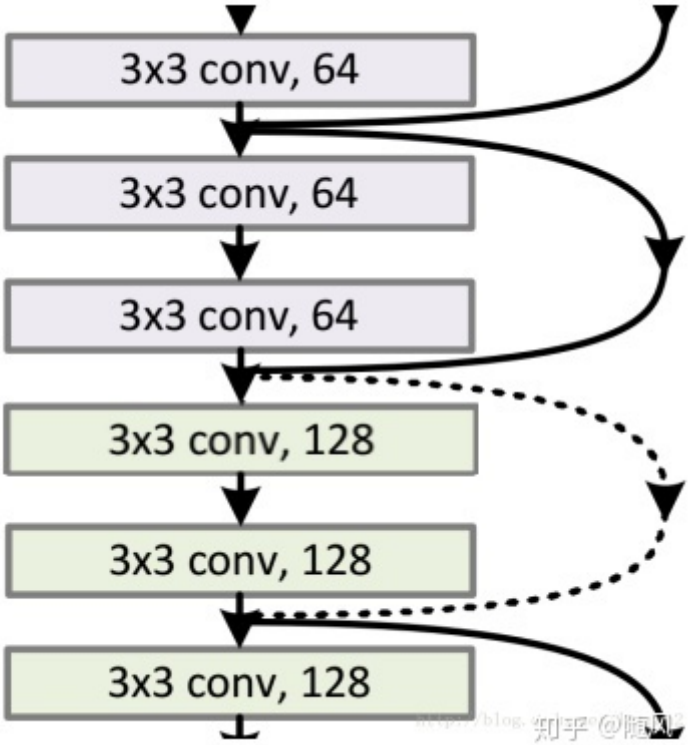
对于残差网络，维度匹配的 shortcut 连接为实线，反之为虚线。维度不匹配时，同等映射（identity mapping）有两种可选方案：

（1）直接通过 zero padding 来增加 channels（采用 zero feature map 补充）。

（2）增加 filters，直接改变 1x1 卷积的 filters 数目，这样会增加参数。

在实际中更多采用 zero feature map 补充的方式。

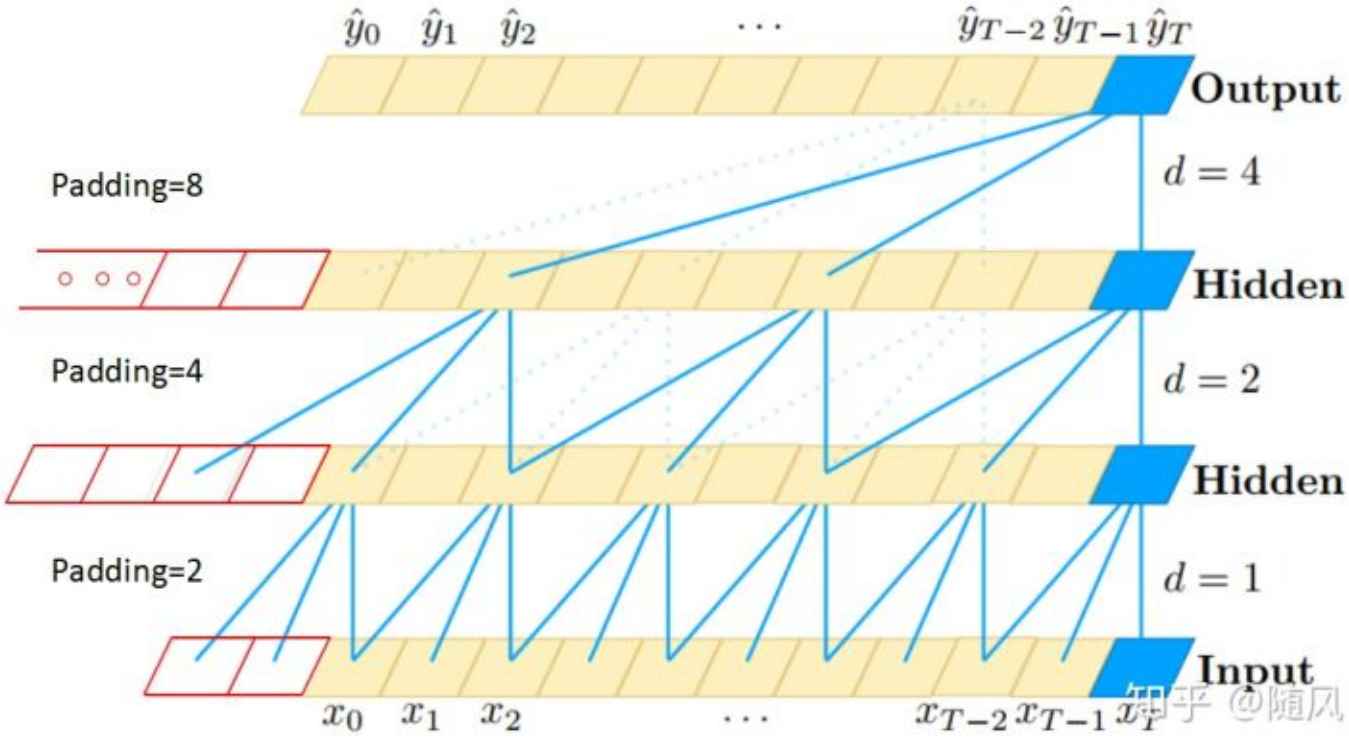
在残差网络中，有很多残差模块，下图是一个残差网络。每个残差模块包含两层，相同维度残差模块之间采用实线连接，不同维度残差模块之间采用虚线连接。网络的 2、3 层执行 3x3x64 的卷积，他们的 channel 个数相同，所以采用计算： $H(x) = F(x) + x$ ；网络的 4、5 层执行 3x3x128 的卷积，与第 3 层的 channel 个数不同 (64 和 128)，所以采用计算方式： $H(x) = F(x) + W_sx$ 。其中 W_s 是卷积操作（用 128 个 3x3x64 的 filter），用来调整 x 的 channel 个数。



残差网络

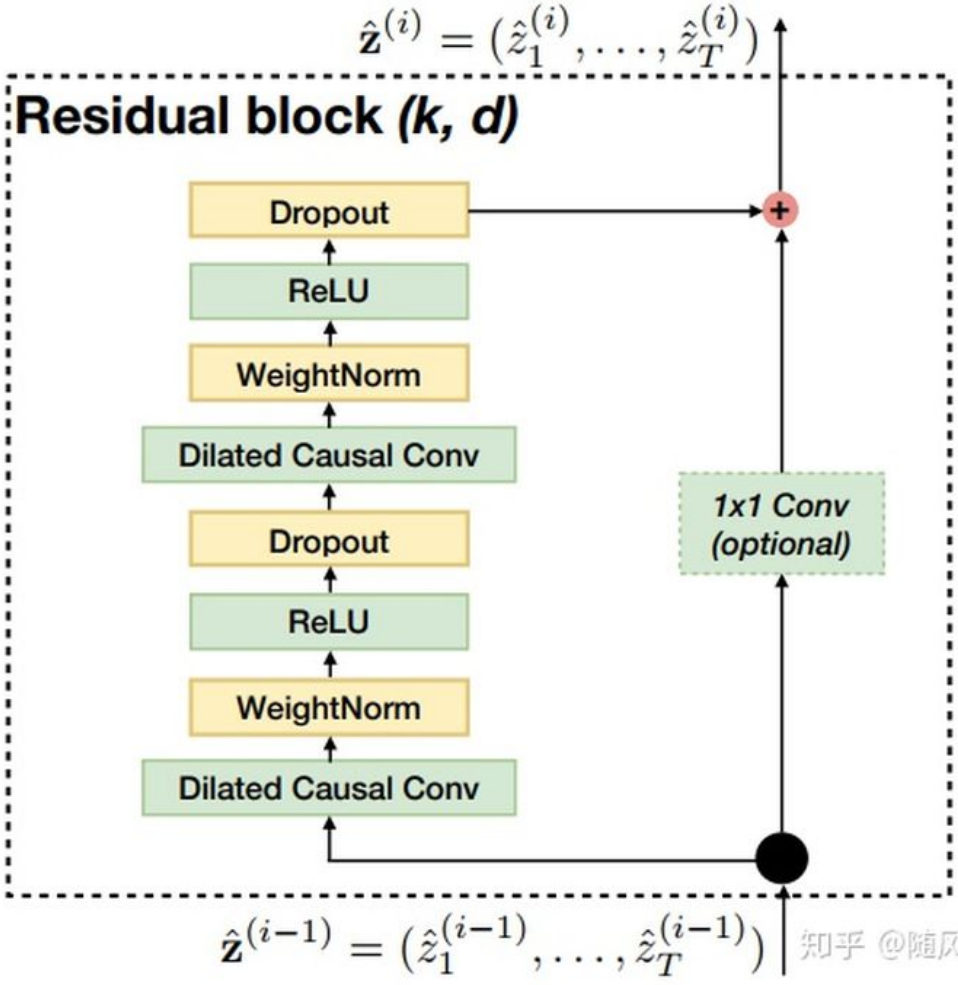
3 TCN（Temporal Convolutional Network）时间卷积网络

因为研究对象是时间序列，TCN 采用一维的卷积网络。下图是 TCN 架构中的因果卷积与空洞卷积，可以看到每一层 t 时刻的值只依赖于上一层 $t, t-1, ..$ 时刻的值，体现了因果卷积的特性；而每一层对上一层信息的提取，都是跳跃式的，且逐层 dilated rate 以 2 的指数增长，体现了空洞卷积的特性。由于采用了空洞卷积，因此每一层都要做 padding（通常情况下补 0），padding 的大小为 $(k-1)d$ 。



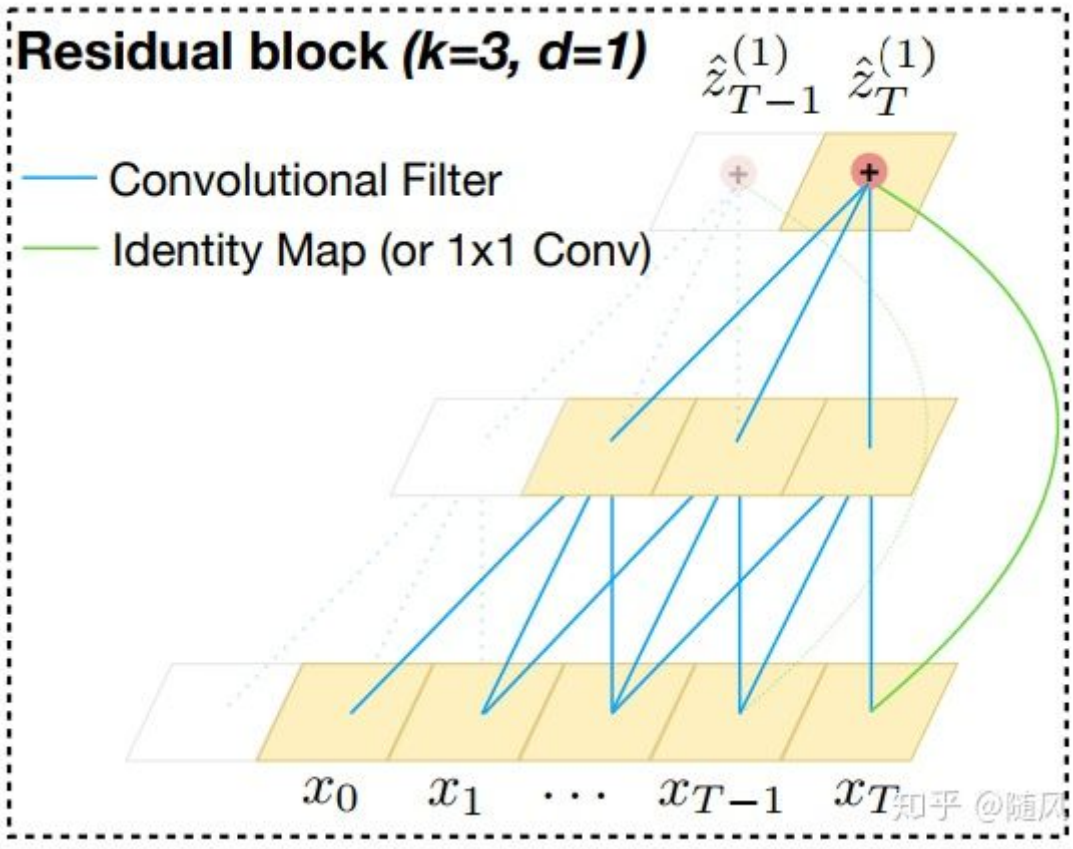
TCN 中的因果卷积与空洞卷积

下图是 TCN 架构中的残差模块，输入经历空洞卷积、权重归一化、激活函数、Dropout（两轮），作为残差函数 $\boldsymbol{F(x)}$ ；输入经历 1x1 卷积 filters，作为 shortcut 连接的 $\boldsymbol{x}$ 。



残差模块

下图是 TCN 的一个例子，当 $d = 1$ 时，空洞卷积退化为普通卷积。



TCN 实例

4 TCN 实战

这次我们采用 tensorflow 2.0 ，自从今年 3 月份发布以来，众多实用的新特性使得编写网络更加方便。推荐一个快速入门的教程：

czy36mengfei/tensorflow2_tutorials_chinese
github.com

由于 tensorflow 并没有提供 TCN 网络的 api，这里需要自己编写。

```
try:
    import tensorflow.python.keras as keras
except:
    import tensorflow.keras as keras

import tensorflow as tf
from tensorflow.python.framework import tensor_shape
from tensorflow.python.keras.engine.base_layer import InputSpec
```

```
from tensorflow.python.keras.utils import conv_utils
from tensorflow.python.ops import nn_impl
from tensorflow.python.ops import nn_ops


class WeightNormConv1D(keras.layers.Conv1D):
    def __init__(self, *args, **kwargs):
        self.weight_norm = kwargs.pop('weight_norm')
        super(WeightNormConv1D, self).__init__(*args, **kwargs)

    def build(self, input_shape):
        input_shape = tensor_shape.TensorShape(input_shape)
        if self.data_format == 'channels_first':
            channel_axis = 1
        else:
            channel_axis = -1
        if input_shape.dims[channel_axis].value is None:
            raise ValueError('The channel dimension of the inputs should be defined. F
        input_dim = int(input_shape[channel_axis])
        kernel_shape = self.kernel_size + (input_dim, self.filters)

        kernel = self.add_weight(
            name='kernel',
            shape=kernel_shape,
            initializer=self.kernel_initializer,
            regularizer=self.kernel_regularizer,
            constraint=self.kernel_constraint,
            trainable=True,
            dtype=self.dtype)

        # weight normalization
        if self.weight_norm:
            self.g = self.add_weight(name='wn/g',
                                     shape=(self.filters,),
                                     initializer=tf.ones_initializer(),
                                     trainable=True,
                                     dtype=kernel.dtype)

            self.kernel = tf.reshape(self.g, [1, 1, self.filters]) * nn_impl.l2_normal
        else:
            self.kernel = kernel

        if self.use_bias:
            self.bias = self.add_weight(
                name='bias',
```

```
                shape=(self.filters,),
                initializer=self.bias_initializer,
                regularizer=self.bias_regularizer,
                constraint=self.bias_constraint,
                trainable=True,
                dtype=self.dtype)
        else:
            self.bias = None

        self.input_spec = InputSpec(ndim=self.rank + 2, axes={channel_axis: input_dim}
        if self.padding == 'causal':
            op_padding = 'valid'
        else:
            op_padding = self.padding
        if not isinstance(op_padding, (list, tuple)):
            op_padding = op_padding.upper()
        self._convolution_op = nn_ops.Convolution(
            input_shape,
            filter_shape=self.kernel.get_shape(),
            dilation_rate=self.dilation_rate,
            strides=self.strides,
            padding=op_padding,
            data_format=conv_utils.convert_data_format(self.data_format, self.rank + 2

        self.built = True


class TemporalLayer(keras.layers.Layer):
    def __init__(self, input_channels, output_channels, kernel_size, strides, dilation
                 weight_norm=True):
        self.input_channels = input_channels
        self.output_channels = output_channels
        self.kernel_size = kernel_size
        self.strides = strides
        self.dilation_rate = dilation_rate
        self.padding = padding
        self.keep_pro = keep_pro
        self.weight_norm = weight_norm

        self.h1 = WeightNormConv1D(filters=self.output_channels, kernel_size=self.kern
                                   data_format='channels_last', dilation_rate=self.dil
                                   kernel_initializer=tf.random_normal_initializer(0,
                                   bias_initializer=tf.zeros_initializer(), weight_nor
        self.h2 = WeightNormConv1D(filters=self.output_channels, kernel_size=self.kern
                                   data_format='channels_last', dilation_rate=self.dil
```

```
                                    kernel_initializer=tf.random_normal_initializer(0,
                                    bias_initializer=tf.zeros_initializer(), weight_nor

        if self.input_channels != self.output_channels:
            self.shou_cut = keras.layers.Conv1D(filters=self.output_channels, kernel_s
                                                kernel_initializer=tf.random_normal_in
                                                bias_initializer=tf.zeros_initializer(
        else:
            self.shou_cut = None

        super(TemporalLayer, self).__init__()

    def call(self, inputs):
        inputs_padding = tf.pad(inputs, [[0, 0], [self.padding, 0], [0, 0]])
        h1_outputs = self.h1(inputs_padding)
        h1_outputs = keras.layers.Dropout(rate=self.keep_pro)(h1_outputs)

        h1_padding = tf.pad(h1_outputs, [[0, 0], [self.padding, 0], [0, 0]])
        h2_outputs = self.h2(h1_padding)
        h2_outputs = keras.layers.Dropout(rate=self.keep_pro)(h2_outputs)

        if self.input_channels != self.output_channels:
            res_x = self.shou_cut(inputs)
        else:
            res_x = inputs

        return keras.activations.relu(keras.layers.add([res_x, h2_outputs]))


class TemporalConvNet(keras.Model):
    def __init__(self, input_channels, layers_channels, strides=1, kernel_size=3, keep
        super(TemporalConvNet, self).__init__(name='TemporalConvNet')
        self.input_channels = input_channels
        self.layers_channels = layers_channels
        self.strides = strides
        self.kernel_size = kernel_size
        self.keep_pro = keep_pro
        self.temporal_layers = []
        num_layers = len(self.layers_channels)
        for i in range(num_layers):
            dilation_rate = 2 ** i
            tuple_padding = (self.kernel_size - 1) * dilation_rate,
            padding = tuple_padding[0]
            input_channels = self.input_channels if i == 0 else self.layers_channels[i
            output_channels = self.layers_channels[i]
```

```
            temporal_layer = TemporalLayer(input_channels, output_channels, self.kerne
                                           dilation_rate, padding, self.keep_pro, True
            self.temporal_layers.append(temporal_layer)

    def call(self, inputs):
        for i in range(len(self.temporal_layers)):
            if i == 0:
                outputs = self.temporal_layers[i](inputs)
            else:
                outputs = self.temporal_layers[i](outputs)
        return outputs
```

下面就可以训练网络了。其实利用深度学习训练时间序列的另一大好处是，一个模型可以**多入多出**，即我们可以输入 m 个序列，输出 n 个序列。这次我们采用一个模型同时预测 total_redeem_amt 与 total_purchase_amt 两个序列，并在输入端加入周一到周日的特征序列。

```
import matplotlib.pyplot as plt
import pandas as pd
import tensorflow as tf
from sklearn.preprocessing import MinMaxScaler
import numpy as np
from ts.tcn import TemporalConvNet

try:
    import tensorflow.python.keras as keras
except:
    import tensorflow.keras as keras


def generate_data():
    dateparse = lambda dates: pd.datetime.strptime(dates, '%Y%m%d')
    user_balance = pd.read_csv('../origion_data/user_balance_table.csv', parse_dates=[
                               date_parser=dateparse)

    user_balance = user_balance.groupby(['report_date'])['total_redeem_amt', 'total_pu
    user_balance.reset_index(inplace=True)
    user_balance.index = user_balance['report_date']

    user_balance = user_balance['2014-03-01':'2014-08-31']
    user_balance[['total_redeem_amt', 'total_purchase_amt']].to_csv(path_or_buf='../mi


# 数据集归一化
```

```
def get_normal_data(purchase_redeem_seq):
    scaler = MinMaxScaler(feature_range=(0, 1))
    scaled_data = scaler.fit_transform(purchase_redeem_seq[['total_redeem_amt', 'total
    return scaled_data, scaler


# 添加特征序列
def add_feature_seq(purchase_redeem_seq, scaled_data):
    report_date = purchase_redeem_seq.index
    weekday = []
    for i in range(len(report_date)):
        weekday.append(report_date[i].weekday())

    purchase_redeem_seq['weekday'] = weekday
    purchase_redeem_seq['workday1'] = 0
    purchase_redeem_seq['workday2'] = 0
    purchase_redeem_seq['workday3'] = 0
    purchase_redeem_seq['workday4'] = 0
    purchase_redeem_seq['workday5'] = 0
    purchase_redeem_seq['weekend1'] = 0
    purchase_redeem_seq['weekend2'] = 0

    for index, row in purchase_redeem_seq.iterrows():
        if row['weekday'] == 0:
            row['workday1'] = 1
        elif row['weekday'] == 1:
            row['workday2'] = 1
        elif row['weekday'] == 2:
            row['workday3'] = 1
        elif row['weekday'] == 3:
            row['workday4'] = 1
        elif row['weekday'] == 4:
            row['workday5'] = 1
        elif row['weekday'] == 5:
            row['weekend1'] = 1
        elif row['weekday'] == 6:
            row['weekend2'] = 1

    scaled_data = np.insert(scaled_data, 2, values=purchase_redeem_seq.workday1, axis=
    scaled_data = np.insert(scaled_data, 3, values=purchase_redeem_seq.workday2, axis=
    scaled_data = np.insert(scaled_data, 4, values=purchase_redeem_seq.workday3, axis=
    scaled_data = np.insert(scaled_data, 5, values=purchase_redeem_seq.workday4, axis=
    scaled_data = np.insert(scaled_data, 6, values=purchase_redeem_seq.workday5, axis=
    scaled_data = np.insert(scaled_data, 7, values=purchase_redeem_seq.weekend1, axis=
    scaled_data = np.insert(scaled_data, 8, values=purchase_redeem_seq.weekend2, axis=
```

```
    return scaled_data


# 构造训练集
def get_train_data(origion_scaled_data, feature_scaled_data, divide_train_valid_index,
    train_x, train_y = [], []
    normalized_train_feature = feature_scaled_data[0: divide_train_valid_index]
    normalized_train_label = origion_scaled_data[1: divide_train_valid_index + 1]
    for i in range(len(normalized_train_label) - seq_len + 1):
        train_x.append(normalized_train_feature[i: i + seq_len])
        train_y.append(normalized_train_label[i: i + seq_len])

    return train_x, train_y


# 构造验证集
def get_valid_data(origion_scaled_data, feature_scaled_data, divide_train_valid_index,
                   seq_len):
    valid_x, valid_y = [], []
    normalized_valid_feature = feature_scaled_data[divide_train_valid_index: divide_va
    normalized_valid_label = origion_scaled_data[divide_train_valid_index + 1: divide_
    for i in range(len(normalized_valid_label) - seq_len + 1):
        valid_x.append(normalized_valid_feature[i: i + seq_len])
        valid_y.append(normalized_valid_label[i: i + seq_len])

    return valid_x, valid_y


# 构造测试集
def get_test_data(origion_scaled_data, feature_scaled_data, divide_valid_test_index, s
    test_x, test_y = [], []
    normalized_test_feature = feature_scaled_data[divide_valid_test_index - seq_len +
    normalized_test_label = origion_scaled_data[divide_valid_test_index + 1:]
    for i in range(len(normalized_test_label)):
        test_x.append(normalized_test_feature[i: i + seq_len])
    test_y = normalized_test_label

    return test_x, test_y


generate_data()


divide_train_valid_index = 152
```

```
divide_valid_test_index = 173
seq_len = 7
input_channels = 9

dateparse = lambda dates: pd.datetime.strptime(dates, '%Y-%m-%d')
purchase_redeem_seq = pd.read_csv('../mid_data/purchase_redeem_seq.csv', parse_dates=[
                                  index_col='report_date', date_parser=dateparse)

origion_scaled_data, scaler = get_normal_data(purchase_redeem_seq)

feature_scaled_data = add_feature_seq(purchase_redeem_seq, origion_scaled_data)

train_x, train_y = get_train_data(origion_scaled_data, feature_scaled_data, divide_tra
valid_x, valid_y = get_valid_data(origion_scaled_data, feature_scaled_data, divide_tra
                                  divide_valid_test_index, seq_len)
test_x, test_y = get_test_data(origion_scaled_data, feature_scaled_data, divide_valid_

train_dataset = tf.data.Dataset.from_tensor_slices((train_x, train_y))
train_dataset = train_dataset.shuffle(buffer_size=1024).batch(10)

valid_dataset = tf.data.Dataset.from_tensor_slices((valid_x, valid_y))
valid_dataset = valid_dataset.shuffle(buffer_size=1024).batch(5)

model = TemporalConvNet(input_channels=input_channels, layers_channels=[32, 16, 8, 4,
model.compile(optimizer=keras.optimizers.Adam(0.001), loss=keras.losses.mean_squared_e

callbacks = [keras.callbacks.EarlyStopping(
    monitor='val_loss',
    min_delta=1e-3,
    patience=100,
    mode='min',
    verbose=2
)]
model.fit(train_dataset, validation_data=valid_dataset, callbacks=callbacks, epochs=10

test_x = tf.reshape(test_x, [len(test_x), seq_len, input_channels])
test_x_pred = model.predict(test_x)
pred_y = []

for i in test_x_pred:
    pred_y.append(i[-1])

inverse_pred_y = scaler.inverse_transform(pred_y)
inverse_test_y = scaler.inverse_transform(test_y)
total_redeem_amt_pred = inverse_pred_y[:, 0]
```

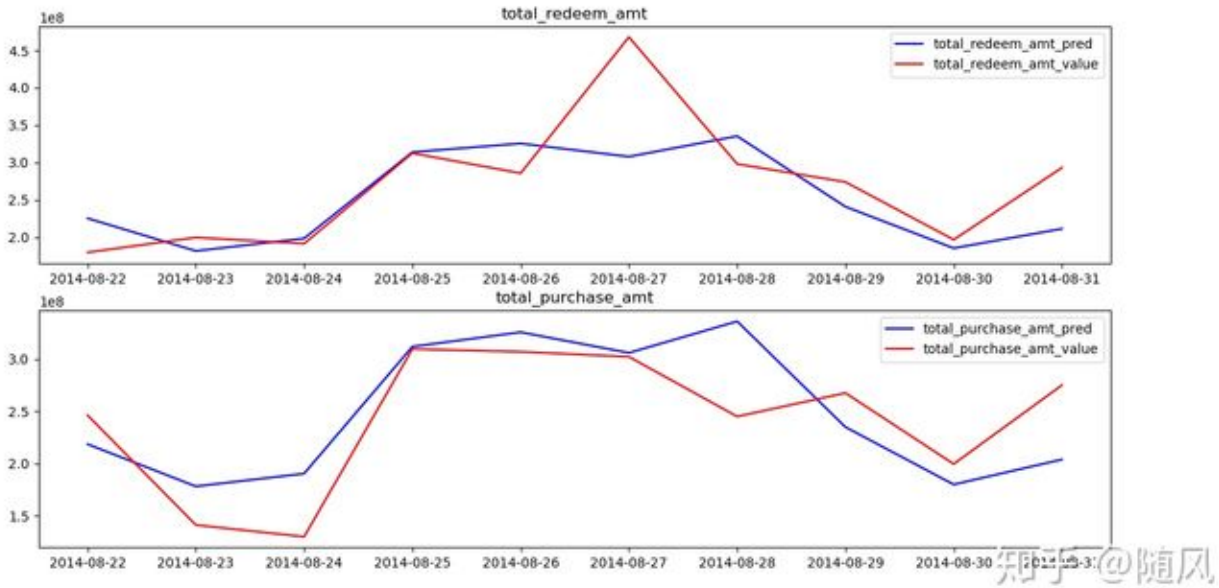
```
total_purchase_amt_pred = inverse_pred_y[:, 1]
total_redeem_amt_value = inverse_test_y[:, 0]
total_purchase_amt_value = inverse_test_y[:, 1]


report_date = ['2014-08-22', '2014-08-23', '2014-08-24', '2014-08-25', '2014-08-26', '
               '2014-08-29', '2014-08-30', '2014-08-31']
df = pd.DataFrame(
    data={'total_redeem_amt_pred': total_redeem_amt_pred, 'total_redeem_amt_value': to
          'total_purchase_amt_pred': total_purchase_amt_pred,
          'total_purchase_amt_value': total_purchase_amt_value}, index=report_date)

plt.figure(figsize=(18, 12))
plt.subplot(211)
plt.title('total_redeem_amt')
plt.plot(df['total_redeem_amt_pred'], label='total_redeem_amt_pred', color='blue')
plt.plot(df['total_redeem_amt_value'], label='total_redeem_amt_value', color='red')
plt.legend(loc='best')

plt.subplot(212)
plt.title('total_purchase_amt')
plt.plot(df['total_purchase_amt_pred'], label='total_purchase_amt_pred', color='blue')
plt.plot(df['total_purchase_amt_value'], label='total_purchase_amt_value', color='red'
plt.legend(loc='best')

plt.show()
```



tcn 预测效果

```
15/15 - 0s - loss: 0.0218 - mse: 0.0207 - val_loss: 0.0140 - val_mse: 0.0140
Epoch 271/1000
15/15 - 0s - loss: 0.0218 - mse: 0.0208 - val_loss: 0.0139 - val_mse: 0.0139
Epoch 272/1000
15/15 - 0s - loss: 0.0219 - mse: 0.0208 - val_loss: 0.0139 - val_mse: 0.0139
Epoch 273/1000
15/15 - 0s - loss: 0.0219 - mse: 0.0209 - val_loss: 0.0141 - val_mse: 0.0141
Epoch 274/1000
15/15 - 0s - loss: 0.0219 - mse: 0.0208 - val_loss: 0.0140 - val_mse: 0.0140
Epoch 275/1000
15/15 - 0s - loss: 0.0218 - mse: 0.0208 - val_loss: 0.0141 - val_mse: 0.0141
Epoch 276/1000
15/15 - 0s - loss: 0.0218 - mse: 0.0208 - val_loss: 0.0139 - val_mse: 0.0139
Epoch 277/1000
15/15 - 0s - loss: 0.0218 - mse: 0.0208 - val_loss: 0.0139 - val_mse: 0.0139
Epoch 00277: early stopping
```


最后 7 轮的训练误差与验证误差

深度学习的准确度依赖于数据量，这里所能给出的数据量实在太小。相比于 lstm，tcn 的速度可谓有了极大的提升，且准确度不弱于 lstm，更适合生产环境使用。