西安交通大學
XI'AN JIAOTONG UNIVERSITY

An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling

动机

时序数据模型的研究历程

1. 自回归模型，线性动力系统，隐马尔可夫模型等 --> 无法描述复杂高维时间序列
2. 传统CNN --> 感受野过小，只能记忆较短的时间片段
3. RNN、LSTM、GRU --> 无法并行处理，训练困难，占用内存，损失精度
4. TCN --> 表现更优

序列建模

假设给定一个序列 $x_0, x_1, \ldots, x_T$，我们希望预测每一时刻对应的输出 $y_0, y_1, \ldots, y_T$。序列模型网络是任意函数 f：$x^{T+1} -> y^{T+1}$ 产生的映射。$\hat{y_0}, \ldots, \hat{y_0} = f(x_0, \ldots, x_T)$，其中 $\hat{y}$ 是真实的对应值。它应该满足因果约束：y_t 仅仅依赖于 $x_0, x_1, \ldots, x_t$，而不依赖于 $x_{t+1}, \ldots, x_T$。在序列模型中，学习的目标是找到一个网络 f，最小化实际输出和预测之间的损失函数

$$L(y_0, y_1, \ldots, y_T, f(x_0, \ldots, x_T))$$

其中序列和输出是根据分布描绘的。

通用卷积神经网络架构

1. 不存在“信息泄露”，即只用1-N之间的数据预测N，而不会使用N+1及之后的数据
2. 能将任意长度的序列如同RNN那样映射为相同长度的输出序列

↓

1. 采用因果卷积
2. 采用全卷积FCN结构，并使用padding保持序列长度

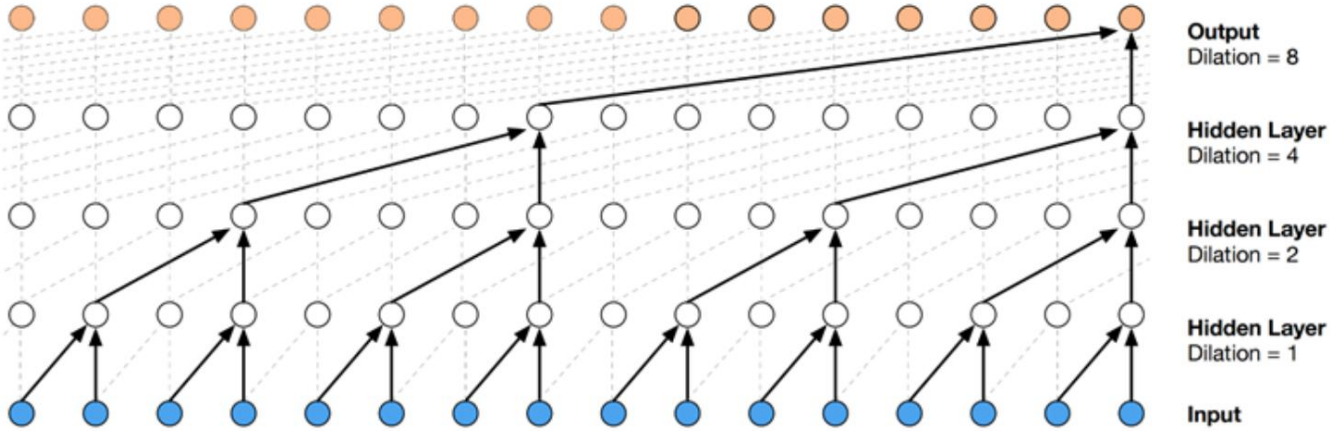
↓

TCN = 1D FCN + causal convolutions

Causal Convolution

所谓 causal convolution,
就是计算 t 时刻的输出时, 仅对前一层 t 时刻及之前的状态进行卷积.

Dilated Convolution



TCN给出了两种扩大感受野的方法

1. 选择更大的过滤器大小k
2. 增大扩展因子d，这样得到的有效感受野大小为(k-1)*d

Dilated Convolution

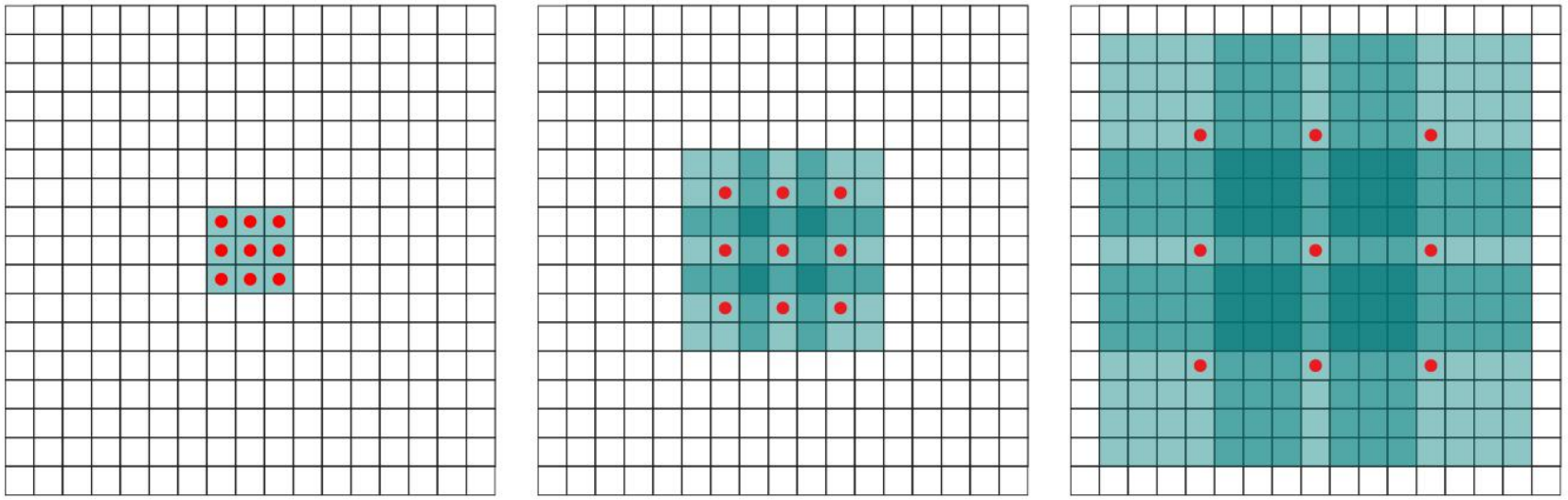
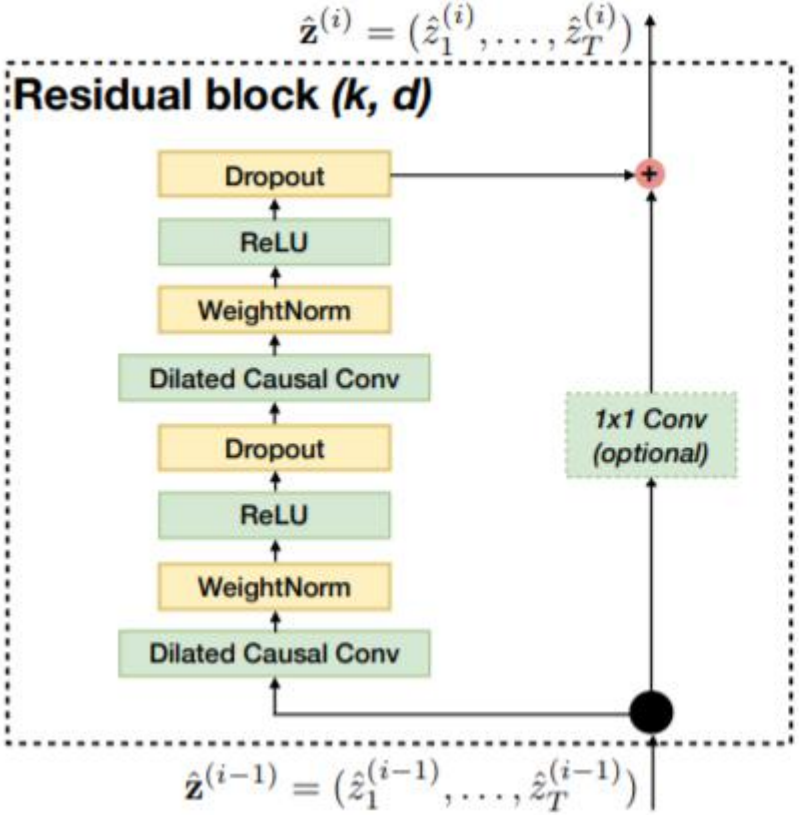
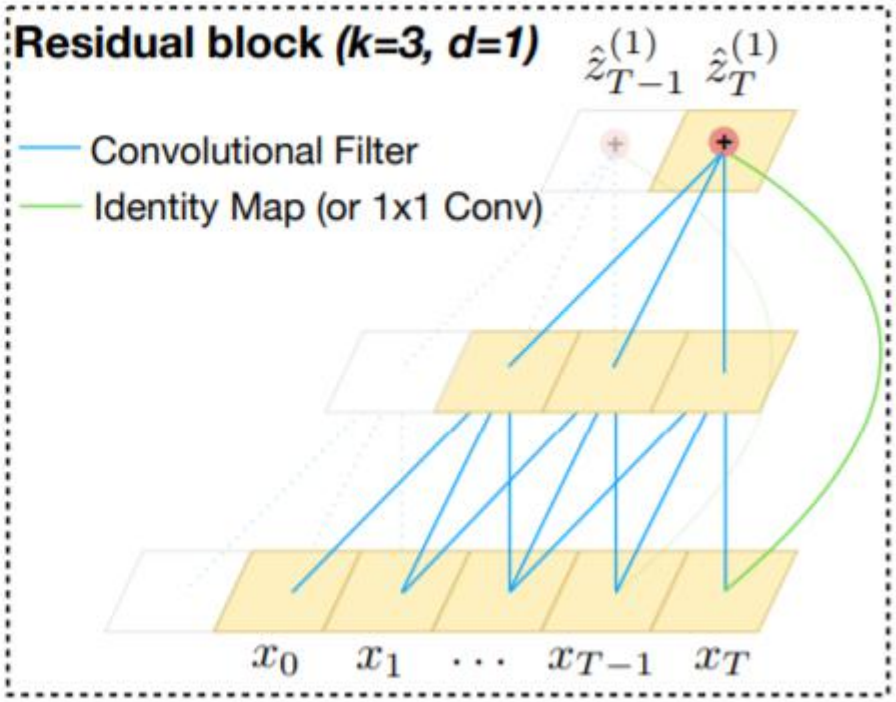


Figure 1: Systematic dilation supports exponential expansion of the receptive field without loss of resolution or coverage. (a) F_1 is produced from F_0 by a 1-dilated convolution; each element in F_1 has a receptive field of 3×3. (b) F_2 is produced from F_1 by a 2-dilated convolution; each element in F_2 has a receptive field of 7×7. (c) F_3 is produced from F_2 by a 4-dilated convolution; each element in F_3 has a receptive field of 15×15. The number of parameters associated with each layer is identical. The receptive field grows exponentially while the number of parameters grows linearly.

可以将dilated看成是kernel稀疏化的一种模式。而stride只是dilated的一种特例

Residual Connections





weight normalization

spatial dropout

结论

Sequence Modeling Task	Model Size (≈)	Models			
		LSTM	GRU	RNN	**TCN**
Seq. MNIST (accuracy^h)	70K	87.2	96.2	21.5	**99.0**
Permuted MNIST (accuracy)	70K	85.7	87.3	25.3	**97.2**
Adding problem T=600 (loss$^\ell$)	70K	0.164	**5.3e-5**	0.177	**5.8e-5**
Copy memory T=1000 (loss)	16K	0.0204	0.0197	0.0202	**3.5e-5**
Music JSB Chorales (loss)	300K	8.45	8.43	8.91	**8.10**
Music Nottingham (loss)	1M	3.29	3.46	4.05	**3.07**
Word-level PTB (perplexity$^\ell$)	13M	**78.93**	92.48	114.50	89.21
Word-level Wiki-103 (perplexity)	-	48.4	-	-	**45.19**
Word-level LAMBADA (perplexity)	-	4186	-	14725	**1279**
Char-level PTB (bpc$^\ell$)	3M	1.41	1.42	1.52	**1.35**
Char-level text8 (bpc)	5M	1.52	1.56	1.69	**1.45**

实验结果表明，TCN模型的性能明显优于LSTMs和GRUs等一般的递归体系结构。并且针对于卷积神经网络与循环神经网络的长期记忆能力，在实际应用中，RNN的无限记忆的优势并不存在。与之相比，相同大小的TCN模型表现出更加出色的长期记忆。

优缺点

TCN 在序列建模方面的优势是:

1. 可并行性 (只要抛弃了 RNN, 神经网络基本都具有了这一优点);
2. 通过调整 n, k, d, 可灵活地控制感受野, 能适应不同任务 (有些任务要求解决超长期依赖, 有些任务更依赖短期依赖);
3. 稳定的梯度 (同样地, 只要抛弃了 RNN, 时间传播方向上的梯度爆炸/消失问题就自然解决了);
4. 训练时的低内存占用，特别是面对长输入序列 (参数共享, 以及只存在沿网络方向的反向传播带来的裨益).

TCN 的缺点:

1. 推断时, 需要更多的内存 (此时 RNN 只需要维护一个 hidden state, 每次接受一个输入; 而 TCN 要保持一个足够长的序列, 以保留历史状态);
2. 迁移的困难性 (不同领域任务对感受野的大小不同, 使用小 k 和小 d 学好的模型难以应用于需要大 k 和大 d 的任务).

代码实现


```python
class Chomp1d(nn.Module):
    def __init__(self, chomp_size):
        super(Chomp1d, self).__init__()
        self.chomp_size = chomp_size

    def forward(self, x):
        return x[:, :, :-self.chomp_size].contiguous()
```

因果卷积实现

代码实现


```
class TemporalBlock(nn.Module):
    def __init__(self, n_inputs, n_outputs, kernel_size, stride, dilation, padding, dropout=0.2):
        super(TemporalBlock, self).__init__()
        self.conv1 = weight_norm(nn.Conv1d(n_inputs, n_outputs, kernel_size,
                                           stride=stride, padding=padding, dilation=dilation))
        self.chomp1 = Chomp1d(padding)
        self.relu1 = nn.ReLU()
        self.dropout1 = nn.Dropout(dropout)

        self.conv2 = weight_norm(nn.Conv1d(n_outputs, n_outputs, kernel_size,
                                           stride=stride, padding=padding, dilation=dilation))
        self.chomp2 = Chomp1d(padding)
        self.relu2 = nn.ReLU()
        self.dropout2 = nn.Dropout(dropout)

        self.net = nn.Sequential(self.conv1, self.chomp1, self.relu1, self.dropout1,
                                 self.conv2, self.chomp2, self.relu2, self.dropout2)
        self.downsample = nn.Conv1d(n_inputs, n_outputs, 1) if n_inputs != n_outputs else None
        self.relu = nn.ReLU()
        self.init_weights()

    def init_weights(self):
        self.conv1.weight.data.normal_(0, 0.01)
        self.conv2.weight.data.normal_(0, 0.01)
        if self.downsample is not None:
            self.downsample.weight.data.normal_(0, 0.01)

    def forward(self, x):
        out = self.net(x)
        res = x if self.downsample is None else self.downsample(x)
        return self.relu(out + res)
```

残差块实现

代码实现


```
class TemporalConvNet(nn.Module):
    def __init__(self, num_inputs, num_channels, kernel_size=2, dropout=0.2):
        super(TemporalConvNet, self).__init__()
        layers = []
        num_levels = len(num_channels)
        for i in range(num_levels):
            dilation_size = 2 ** i
            in_channels = num_inputs if i == 0 else num_channels[i-1]
            out_channels = num_channels[i]
            layers += [TemporalBlock(in_channels, out_channels, kernel_size, stride=1, dilation=dilation_size,
                                     padding=(kernel_size-1) * dilation_size, dropout=dropout)]

        self.network = nn.Sequential(*layers)

    def forward(self, x):
        return self.network(x)
```

模型整体实现

反思

1. 创新点比较少，但是提供了一种解决序列问题的思路
2. 实验大多为合成任务，说服力比较弱
3. 对于当前我们所做的任务可能不太适合
4. 1D卷积是否可以扩展到2D，如何处理
5. 与transformer的对比

相关工作

1. WaveNet，处理音频数据
2. TrellisNet，a) 在所有层之间进行权值共享；b) 输入序列作为每层输入的一部分