One Model To Learn Them All

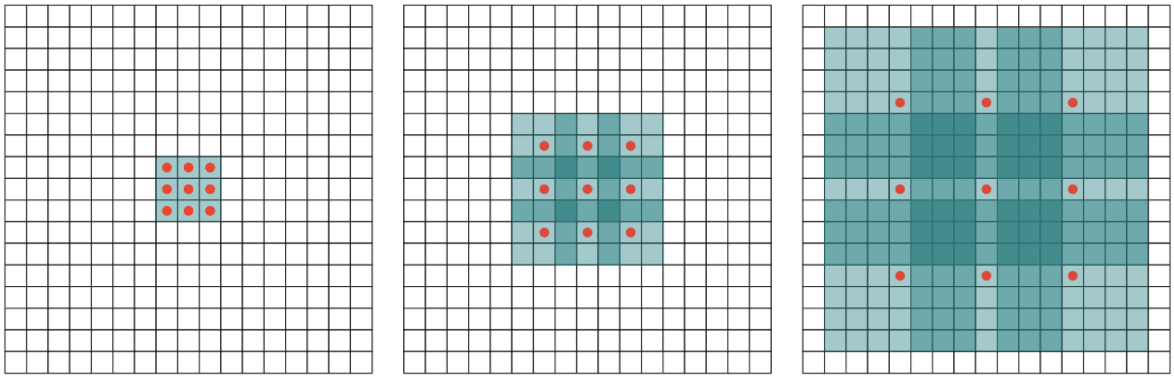
MultiModel，实现一个multi modal multi tasks model

工程结构

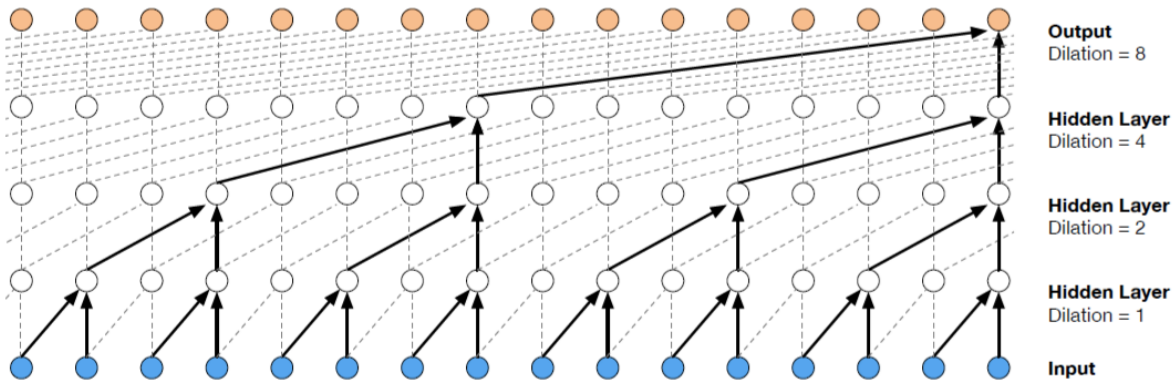
一、convolutional blocks

1. dilated convolutional layer：空洞卷积/扩张卷积，目的是为了扩大filter对于图像的respective field（感受到的图像的范围大小）

每个红点对应的weight不为0，其余为0。第一个的为1-dilated，就是一般的卷积，感受野为3*3；第二个为2-dilated，它前面接一层1-dilated后，感受野为7*7；第三个为4-dilated，它前面接1-dilated、2-dilated后，感受野为15*15。



看一维的wavelet中使用的dilated conv：

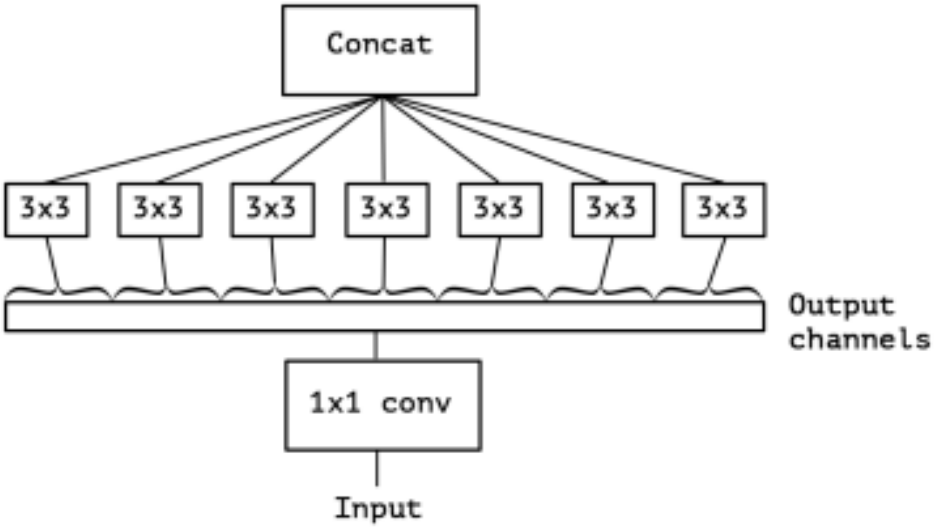


2. depthwise separable convolution

可以认为就是从将每个channel区分开来，然后分别独立计算。可以参考下图的extreme Inception，它先通过一个1*1的con（可以认为是pointwise的卷积，是对某个点的所有channel加权平均，目的是将输入channel映射到新的channel），再进行

depthwise separable convolution。而separable convolution是先depthwise再pointwise的。

Figure 4. An "extreme" version of our Inception module, with one spatial convolution per output channel of the 1x1 convolution.



3. 模型结构

convstep用的是separable dilated convolution，然后在hidden2和hidden4中加入residual部分，之后dropout一下。

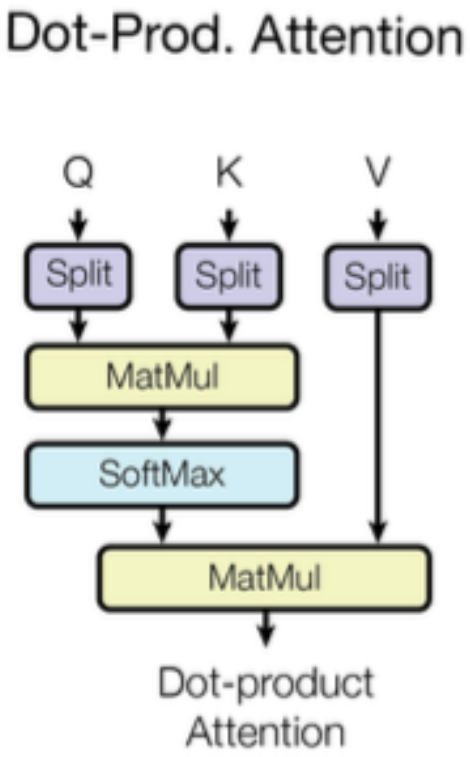
$$ConvStep_{d,s,f}(W,x) = LN(SepConv_{d,s,f}(W, ReLU(x))).$$

$$\begin{aligned}
hidden1(x) &= ConvStep(W_{h1}^{3\times1}, x) \\
hidden2(x) &= x + ConvStep(W_{h2}^{3\times1}, hidden1(x)) \\
hidden3(x) &= ConvStep(W_{h3}^{15\times1}, hidden2(x)) \\
hidden4(x) &= x + ConvStep_{d=8}(W_{h4}^{15\times1}, hidden3(x)) \\
ConvBlock(x) &= \begin{cases} Dropout(hidden4(x), 0.4) & \text{during training} \\ hidden4(x) & \text{otherwise} \end{cases}
\end{aligned}$$

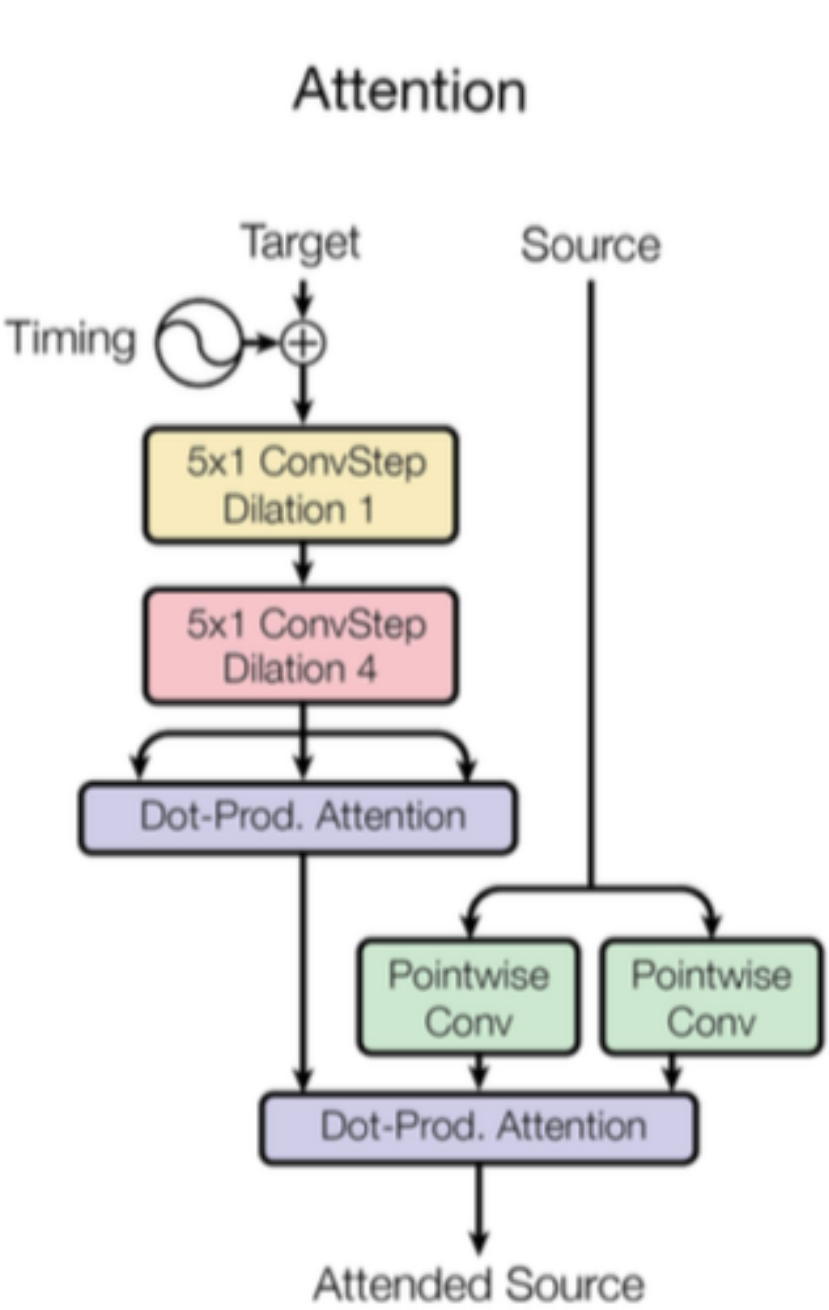
二、attention blocks

1. multi-head dot-product attention

使用multi-head dot-product attention mechanism来计算attention



左边是将target data（embedding）和timing embedding直接相加，然后通过两层dilation conv，再做multi-head self-attention；右边则是source data（embedding）分别传入两个pointwise conv，之后同左边输出到一个multi-head attention，得到一个target（target是query）关于source（source是key and value）的attention矩阵。



2. timing

timing的输出应该和target data的形式一样，都是[batch_size, time_step, hidden_size]。固定某个sample，对应time step为t，那么对应的hidden部分的第2d维和第2d+1维分别由sin和cos确定，可以认为对hidden部分的奇数位和偶数位分别用sin和cos处理。其中，d决定了正弦曲线的周期，t决定了正弦曲线的相位。

$$\Delta(2d) = 1e4^{-\frac{2d}{depth}}$$
$$timing(t, [2d, 2d+1]) = [\sin(t\Delta(2d)) \|_2 \cos(t\Delta(2d))]$$

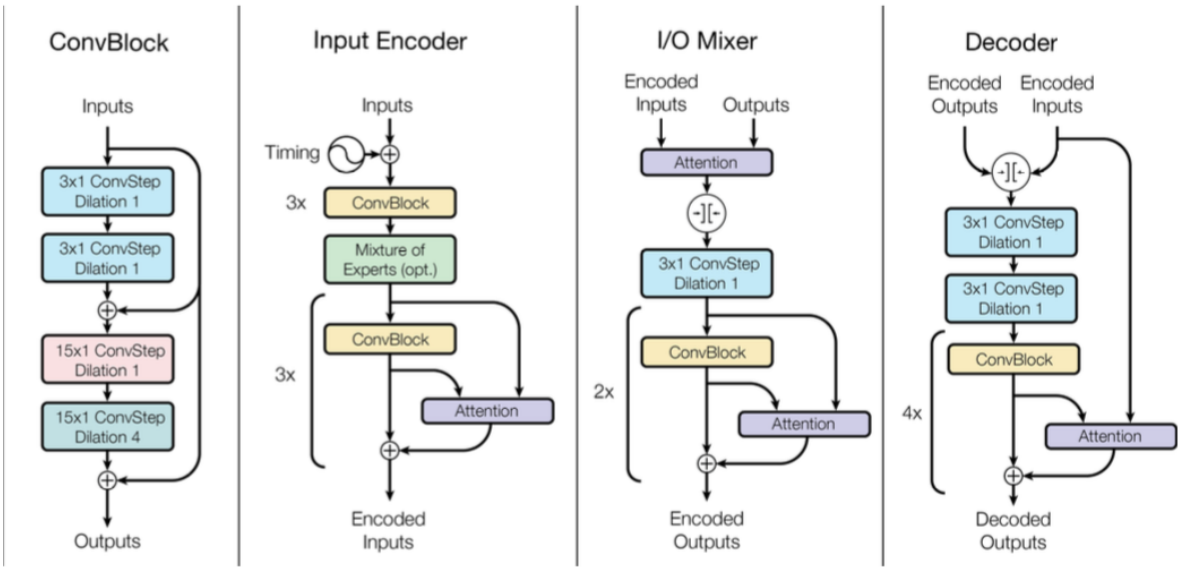
where $[a||_d b]$ represent concatenation of a and b along the d^{th} dimension. The *source* tensor is finally passed through two different pointwise convolutions to generate the memory keys K and values V and the query keys, memory keys and memory values are used to apply the attention mechanism between the self-attended *target* and the *source* (see Figure 3).

3. Mixture-of-Experts Blocks

各种expert blocks的组合。没有详述具体的内容。

4. Encoder and Mixer and Decoder

下图的input encoder输出encoded inputs以及I/O Mixer输出encoded outputs，两者作为Decoder的输入。图中白色的左右对称的圆表示concat（猜的）。attention的左侧输入是作为target的，因此左侧的conv层无法访问到未来的信息（具体原因不是很明白，文中没有详述）。下文的long term dependencies可以理解为输入与输出序列之间任意两个元素对应的距离（The shorter these paths between any combination of positions in the input and output sequences, the easier it is to learn long-range dependencies.）



The body of the MultiModel consists of 3 parts: the encoder that only processes the inputs, the mixer that mixes the encoded inputs with previous outputs (autoregressive part), and a decoder that processes the inputs and the mixture to generate new outputs.

The encoder, mixer and decoder are structured similarly to previous fully convolutional sequence to sequence models such as ByteNet [11] or WaveNet [29], but differ in the computational blocks that are used. We depict their architecture in Figure 3. As can be seen there, the encoder consists of 6 repeated convolutional blocks (described before) with a mixture-of-experts layer in the middle. The mixer consists of an attention block and 2 convolutional blocks. The decoder consists of 4 blocks of convolutions and attention, with a mixture-of-experts layer in the middle. Crucially, the convolutions in the mixer and decoder are padded on the left, so they can never access any information in the future. This allows the model to be autoregressive, and this convolutional autoregressive generation scheme offers large receptive fields over the inputs and past outputs, which are capable of establishing long term dependencies.

To allow the decoder to produce outputs for different tasks even with the same modality, we always start decoding with a command-token, such as *To-English* or *To-Parse-Tree*. We learn an embedding vector corresponding to each of the tokens during training.