实践教程 | PyTorch中相对位置编码的理解

原创 CV开发者都爱看的 极市平台 2021-08-25 22:00:00 手机阅读

作者 | 有为少年

编辑 | 极市平台

极市导读

本文重点讨论BotNet中的2D相对位置编码的实现中的一些细节。注意，这里的相对位置编码方式和Swin Transformer中的不太一样，读者可以自行比较。 >>加入极市CV技术交流群，走在计算机视觉的最前沿

前言

这里讨论的相对位置编码的实现策略实际上原始来自于：https://arxiv.org/pdf/1809.04281.pdf。

这里有一篇介绍性的文章：https://gudgud96.github.io/2020/04/01/annotated-music-transformer/， 图例非常清晰。

首先理解下相对位置自注意力中关于位置嵌入的一些细节。

3.3 RELATIVE POSITIONAL SELF-ATTENTION

As the Transformer model relies solely on positional sinusoids to represent timing information, Shaw et al. (2018) introduced relative position representations to allow attention to be informed by how far two positions are apart in a sequence. This involves learning a separate relative position embedding E^r of shape (H, L, D_h), which has an embedding for each possible pairwise distance $r = j_k - i_q$ between a query and key in position i_q and j_k respectively. The embeddings are ordered from distance $-L + 1$ to 0, and are learned separately for each head. In Shaw et al. (2018), the relative embeddings interact with queries and give rise to a S^{rel}, an $L \times L$ dimensional logits matrix which modulates the attention probabilities for each head as:

$$\text{RelativeAttention} = \text{Softmax}\left(\frac{QK^\top + S^{rel}}{\sqrt{D_h}}\right)V. \quad (3)$$

We dropped head indices for clarity. Our work uses the same approach to infuse relative distance information in the attention computation, while significantly improving upon the memory footprint for computing S^{rel}. For each head, Shaw et al. (2018) instantiate an intermediate tensor R of shape (L, L, D_h), containing the embeddings that correspond to the relative distances between all keys and queries. Q is then reshaped to an $(L, 1, D_h)$ tensor, and $S^{rel} = QR^\top$.[2] This incurs a total space complexity of $O(L^2D)$, restricting its application to long sequences.

相对注意力的一些相关概念。摘自Music Transformer。在不考虑head维度时：

- E^r：相对位置嵌入，大小为(L, D_h)
- R：来自Shaw论文中引入的相对位置嵌入的中间表示，大小为$(L\ ,\ L\ ,\ D_h)$
- $S^{rel} = QR^T$：表示相对位置编码与query的交互结果，大小为$(L\ ,\ L)$，即在D_h维度上进行了累加
- Music Transformer的一点工作就是将这个会占用较大存储空间的中间表示R去掉，直接得到S^{rel}，如下图所示：

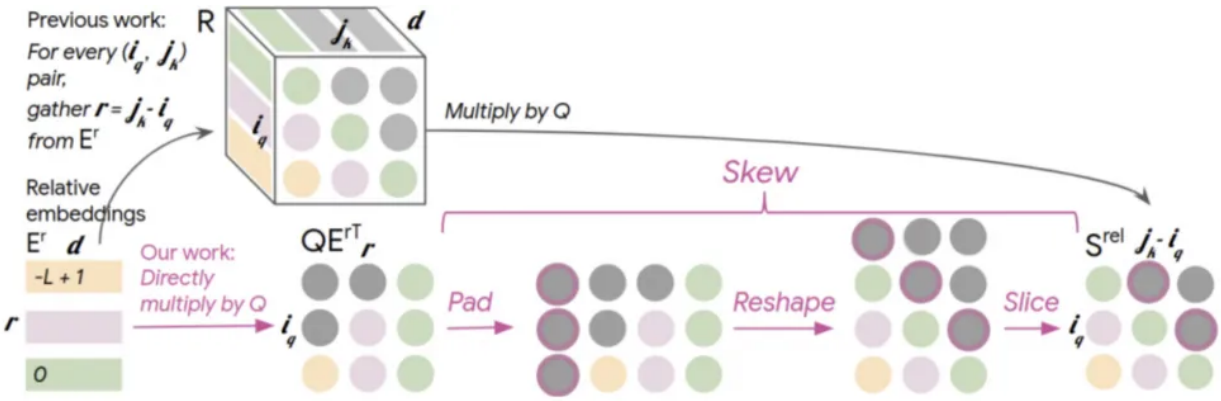


Figure 1: Relative global attention: the bottom row describes our memory-efficient "skewing" algorithm, which does not require instantiating R (top row, which is $O(L^2D)$). Gray indicates masked or padded positions. Each color corresponds to a different relative distance.

要注意这里的E^r表示的是针对相对位置$-L+1 \rightarrow 0$的嵌入，最小相对位置为$-L+1$，最大为0(因为需要考虑因果关系，前面的i看不到后面的j)，所以有L个位置。

而对于我们这里将要讨论的不考虑因果关系的情况，最小相对位置为$-N+1$，最大为$N-1$。所以我们的位置嵌入E^r形状为$(2N-1) \times d$。

代码分析

首先找份代码来看看， https://github.com/lucidrains/bottleneck-transformer-pytorch/blob/main/bottleneck_transformer_pytorch/bottleneck_transformer_pytorch.py 实现的相对位置编码涉及到几个关键的组件：

```
import torch
import torch.nn as nn
from einops import rearrange

def relative_to_absolute(q):
    """
    Converts the dimension that is specified from the axis
    from relative distances (with length 2*tokens-1) to absolute distance (length tokens)

    borrowed from lucidrains:
    https://github.com/lucidrains/bottleneck-transformer-pytorch/blob/main/bottleneck_transformer_pytorch/bottleneck_transf

    Input: [bs, heads, length, 2*length - 1]
    Output: [bs, heads, length, length]
    """
    b, h, l, _, device, dtype = *q.shape, q.device, q.dtype
    dd = {'device': device, 'dtype': dtype}
    col_pad = torch.zeros((b, h, l, 1), **dd)
    x = torch.cat((q, col_pad), dim=3)  # zero pad 2l-1 to 2l
    flat_x = rearrange(x, 'b h l c -> b h (l c)')
```

```
    flat_pad = torch.zeros((b, h, l - 1), **dd)
    flat_x_padded = torch.cat((flat_x, flat_pad), dim=2)
    final_x = flat_x_padded.reshape(b, h, l + 1, 2 * l - 1)
    final_x = final_x[:, :, :l, (l - 1):]
    return final_x

def rel_pos_emb_1d(q, rel_emb, shared_heads):
    """
    Same functionality as RelPosEmb1D
    Args:
        q: a 4d tensor of shape [batch, heads, tokens, dim]
        rel_emb: a 2D or 3D tensor
        of shape [ 2*tokens-1 , dim] or [ heads, 2*tokens-1 , dim]
    """
    if shared_heads:
        emb = torch.einsum('b h t d, r d -> b h t r', q, rel_emb)
    else:
        emb = torch.einsum('b h t d, h r d -> b h t r', q, rel_emb)
    return relative_to_absolute(emb)

class RelPosEmb1DAISummer(nn.Module):
    def __init__(self, tokens, dim_head, heads=None):
        """
        Output: [batch head tokens tokens]
        Args:
            tokens: the number of the tokens of the seq
            dim_head: the size of the last dimension of q
            heads: if None representation is shared across heads.
            else the number of heads must be provided
        """
        super().__init__()
        scale = dim_head ** -0.5
        self.shared_heads = heads if heads isnotNoneelseTrue
        if self.shared_heads:
            self.rel_pos_emb = nn.Parameter(torch.randn(2 * tokens - 1, dim_head) * scale)
        else:
            self.rel_pos_emb = nn.Parameter(torch.randn(heads, 2 * tokens - 1, dim_head) * scale)
    def forward(self, q):
        return rel_pos_emb_1d(q, self.rel_pos_emb, self.shared_heads)
```

可以看到：

- `RelPosEmb1DAISummer` 初始化了E^r
- `rel_pos_emb_1d` 为 `relative_to_absolute` 提供$QE^{r\top}$(为了便于书写，我们将其设为S)，通过在 `relative_to_absolute` 中各种形变和padding，从而得到了理解的难点在 `relative_to_absolute` 中的实现过程。

这里会把S从一个$(N，\ 2N-1)$tensor转化为一个$(N，\ N)$的tensor。这个过程实际上就是一个从表中查找的过程。

这里的实现其实有些晦涩，直接阅读代码是很难明白其中的意义。接下来会重点说这个。

需要注意的是，下面的分析都是按照1D的token序列来解释的，实际上2D的也是将H和W分别基于1D的策略处理的。也就是将H或者W合并到头索引那一维度，即这里的 `heads` ，结果就和1D的一致了，只是还会多一个额外的广播的过程。如下代码：

```
import torch.nn as nn
from einops import rearrange
from self_attention_cv.pos_embeddings.relative_embeddings_1D import RelPosEmb1D

class RelPosEmb2DAISummer(nn.Module):
    def __init__(self, feat_map_size, dim_head, heads=None):
        """
        Based on Bottleneck transformer paper
        paper: https://arxiv.org/abs/2101.11605 . Figure 4
        Output: qr^T [batch head tokens tokens]
        Args:
            tokens: the number of the tokens of the seq
            dim_head: the size of the last dimension of q
            heads: if None representation is shared across heads.
            else the number of heads must be provided
        """
        super().__init__()
        self.h, self.w = feat_map_size  # height , width
        self.total_tokens = self.h * self.w
        self.shared_heads = heads if heads isnotNoneelseTrue
        self.emb_w = RelPosEmb1D(self.h, dim_head, heads)
        self.emb_h = RelPosEmb1D(self.w, dim_head, heads)

    def expand_emb(self, r, dim_size):
        # Decompose and unsqueeze dimension
        r = rearrange(r, 'b (h x) i j -> b h x () i j', x=dim_size)
        expand_index = [-1, -1, -1, dim_size, -1, -1]  # -1 indicates no expansion
        r = r.expand(expand_index)
        return rearrange(r, 'b h x1 x2 y1 y2 -> b h (x1 y1) (x2 y2)')

    def forward(self, q):
        """
        Args:
            q: [batch, heads, tokens, dim_head]
        Returns: [ batch, heads, tokens, tokens]
        """
        assert self.total_tokens == q.shape[2], f'Tokens {q.shape[2]} of q must \
        be equal to the product of the feat map size {self.total_tokens} '
        # out: [batch head*w h h]
        r_h = self.emb_w(rearrange(q, 'b h (x y) d -> b (h x) y d', x=self.h, y=self.w))
        r_w = self.emb_h(rearrange(q, 'b h (x y) d -> b (h y) x d', x=self.h, y=self.w))
        q_r = self.expand_emb(r_h, self.h) + self.expand_emb(r_w, self.w)
        return q_r
```

提前的思考

首先我们要明确，为什么对于每个维度为d的token T_i，其对应的整体S会有$2N-1 \rightarrow N$这样一个缩减的过程？

因为对于长为N的序列中的每一个元素T_i，实际上与之可能有关的元素T_j最多只有N个(虽说是废话，但是在直接理解时可能确实容易忽略这一点。)。

所以对于每个元素，实际上这里的S并不会都用到。这里的S只是所有可能会用到的情形(分别对应于各种相对距离$j-i \in \{-N+1,\ -N+2,\ \cdots,\ -1,\ 0,\ 1,\ \cdots,\ N-2,\ N-1\}$)。

> 这里需要说明的一点是，有些相对注意力的策略中，会使用固定的窗口。

> 即对于窗口之外的j，和窗口边界上的j的相对距离认为是一样的，即$clip(j-i,-k,k)$，我们这里介绍的可以看做是$k=N-1$。

> 例如这个实现：https://github.com/TensorUI/relative-position-pytorch/blob/master/relative_position.py

所以这里前面展示的这个函数 `relative_to_absolute` 实际上就是在做这样一件事：从S中抽取对应于各个tokenT_i真实存在的相对距离$j-i$的位置嵌入集合$\{S_{i,rel_to_abs(j-i)}\}_j^{clip(j-i,-N+1,N-1)}$来得到最终的$S^{rel}$.

背后的动机

为了便于展示这个代码描述的过程的动机，我们首先构造一个简单的序列$\{a,\ b,\ c,\ d,\ e\}$，包含5个元素，则$N=5$。这里嵌入的维度为d。则位置$i\&j$对应的相对距离矩阵可以表示为：

Token ID		T_j 0	1	2	3	4
T_i	0	0	1	2	3	4
	1	-1	0	1	2	3
	2	-2	-1	0	1	1
	3	-3	-2	-1	0	1
	4	-4	-3	-2	-1	0

图1

这里红色标记表示各个位置上的相对距离。我们再看下假定已经得到的$S \in \mathbb{R}^{N\times(2N-1)}$：

$(j-i)_{abs}$		0	1	2	3	4	5	6	7	8
$(j-i)_{rel}$		-4	-3	-2	-1	0	1	2	3	4
T_i	0	S								
	1									
	2									
	3									
	4									

图2

这里对各个T_i都提供了独立的一套嵌入$S_i \in \mathbb{R}^{2N-1}$。为了直观的展示，这里我们也展示了对于这$2N-1$个相对位置的相对距离，同时也标注了对应于嵌入矩阵各列的绝对索引。

接下来我们就需要提取想要的那部分嵌入的tensor了。这个时候，我们需要明白，我们要获取的是哪部分结果：

$(j-i)_{abs}$		0	1	2	3	4	5	6	7	8
$(j-i)_{rel}$		-4	-3	-2	-1	0	1	2	3	4
T_i	0					■	■	■	■	■
	1				■	■	■	■	■	
	2			■	■	■	■	■		
	3		■	■	■	■	■			
	4	■	■	■	■	■				

图3

这里实际上就是结合了图1中已经得到的相对距离和图2中的$(j-i)_{rel}$，从而就可以明白，红色的这部分区域正是我们想要的那部分合理索引对应的位置编码。

稍微整理下，也就是如下的绝对索引对应的嵌入信息$S^{rel} \in \mathbb{R}^{N \times N}$(形状与$QK^{\top}$一致，可以直接元素级相加)：

Srel						
T_i	0	4	5	6	7	8
	1	3	4	5	6	7
	2	2	3	4	5	6
	3	1	2	3	4	5
	4	0	1	2	3	4

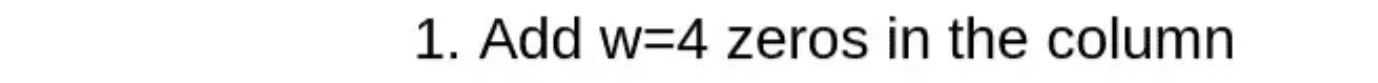
图4

而前面的代码 `relative_to_absolute` 正是在做这样一件事。就是通过不断的 `padding` 和 `reshape` 来从图3中获得图4中这些绝对索引对应的嵌入。

对应的流程

关于代码的流程，参考链接中的图例非常直观：

```
col_pad = torch.zeros((b, h, l, 1), **dd)
x = torch.cat((q, col_pad), dim=3)  # zero pad 2l-1 to 2l
```



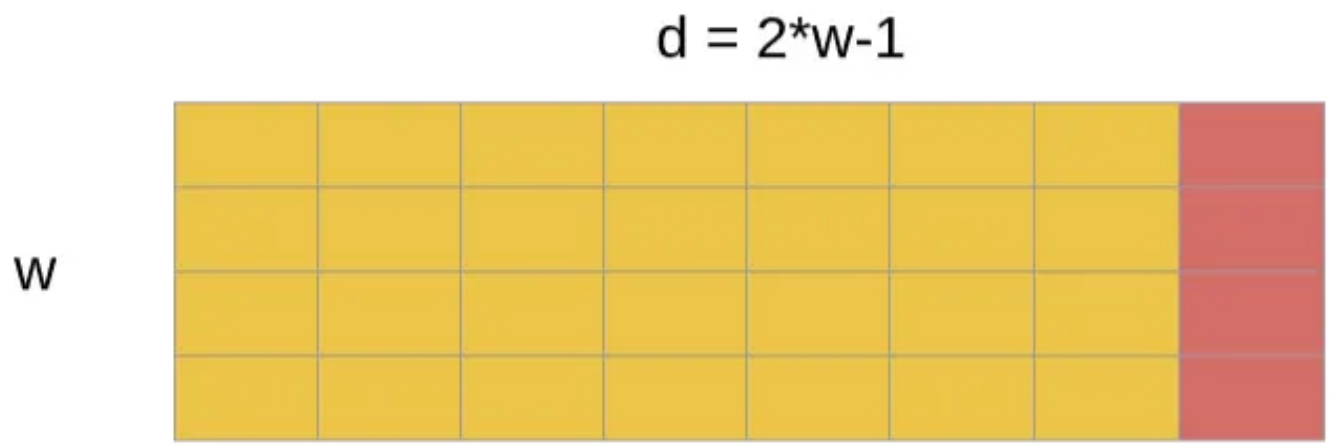


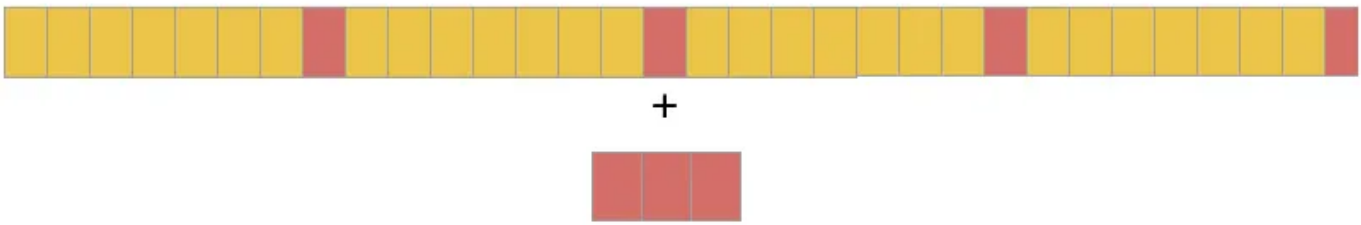
image.png

```
flat_x = rearrange(x, 'b h l c -> b h (l c)')
```

2. Flatten

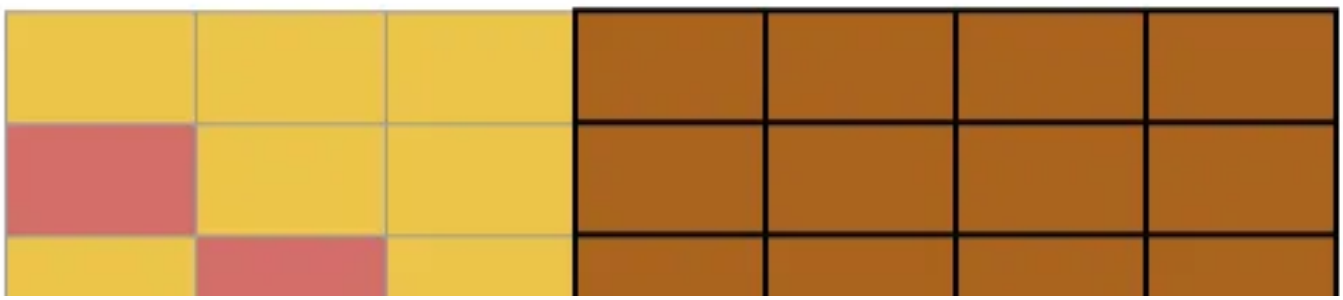
```
flat_pad = torch.zeros((b, h, l - 1), **dd)
flat_x_padded = torch.cat((flat_x, flat_pad), dim=2)
```

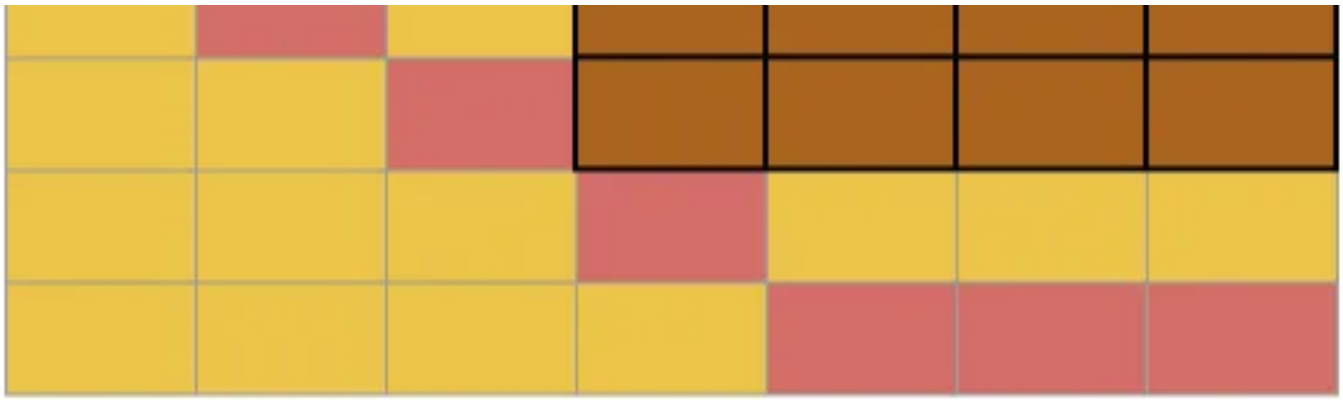
3. Add (w-1) zeros in the end



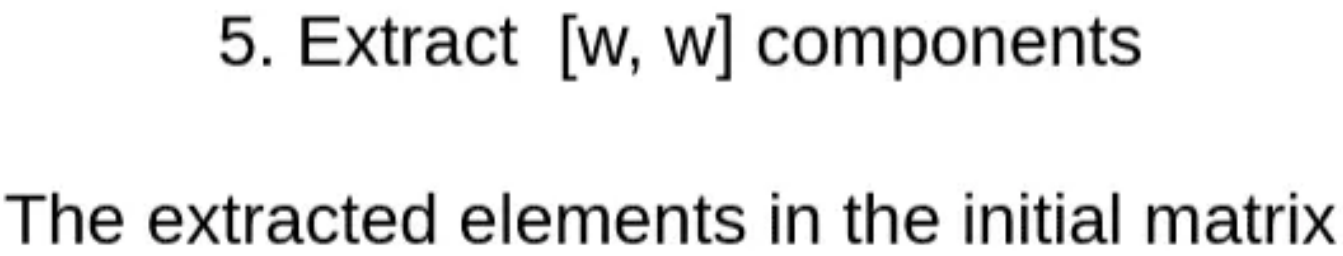

```
final_x = flat_x_padded.reshape(b, h, l + 1, 2 * l - 1)
final_x = final_x[:, :, :l, (l - 1):]
```

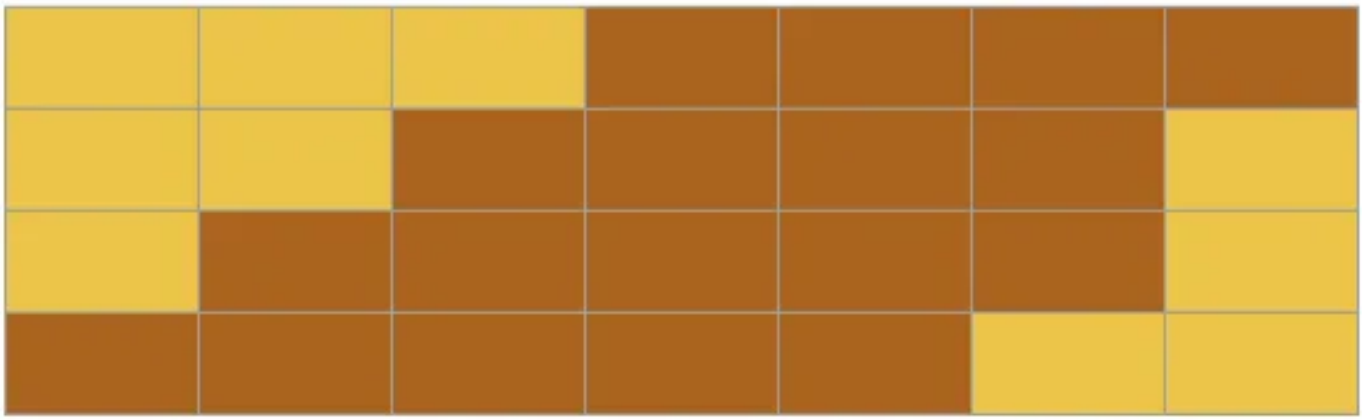
4. Reshape to w+1, d





将提取的内容对应于原始的R中，可以看到是如下区域，正如前面的分析所示。





参考

- AI SUMMER这篇文章写的很好，很直观，很清晰：https://theaisummer.com/positional-embeddings/

实操教程：OpenVINO2021.4+YOLOX目标检测模型测试部署 | 为什么你的显卡利用率总是0%?

算法技巧（trick）： 图像分类算法优化技巧 | 21个深度学习调参的实用技巧

觉得有用麻烦给个在看啦~

阅读原文