Transformer

Transformer仅使用注意力机制而不适用卷积和循环

- 注意力机制的好处：并行度更好
- 用于$Sequence\ transduction$（序列转换）
- RNN并行度不够，早期的时序信息会丢失
- 单个卷积层难以查看整个时序序列，但是可以输出多个通道->多头的来源

模型架构

（参数：输入维度，多头的个数）

$RESNET$的简要介绍

- 在传统的深度神经网络中，随着网络深度的增加，梯度消失或梯度爆炸的问题变得更加严重，这会导致训练变得困难。$ResNet$通过引入“残差连接”或“跳跃连接”来缓解这些问题。具体来说，假设网络的某一层的输入为x，如果这层网络的输出为$F(x)$，那么这层的残差块的输出则为：

$$y = F(x) + x$$

- 其中，$F(x)$表示通过若干层（如两层卷积）的变换，而x则是输入直接传递到输出的跳跃连接。残差块的基本思想是，网络的每一层都去学习一个与输入的残差（即$F(x)$部分）而不是直接学习输出。这样做的一个好处是，如果理想的映射函数$H(x)$就是输入x本身，那么网络只需要将$F(x)$学习为零即可，这样残差块的输出就会直接等于输入，从而避免退化问题。

RNN的简要介绍

RNN（循环神经网络）是一种处理序列数据的神经网络，其计算过程如下：

1. **输入表示**：
   假设输入序列为 $\{x_1, x_2, \ldots, x_T\}$，其中 $x_t \in \mathbb{R}^n$ 表示时间步 t 的输入向量。
2. **隐藏状态更新**：
   在每个时间步 t，RNN维护一个隐藏状态 $h_t \in \mathbb{R}^m$，它是当前输入 x_t 和前一时间步的隐藏状态 h_{t-1} 的函数。隐藏状态的更新公式为：

   $$h_t = f(W_h h_{t-1} + W_x x_t + b_h)$$

   其中：

- $W_h \in \mathbb{R}^{m \times m}$ 是隐藏状态的权重矩阵
- $W_x \in \mathbb{R}^{m \times n}$ 是输入的权重矩阵
- $b_h \in \mathbb{R}^m$ 是偏置向量
- $f(\cdot)$ 是激活函数（例如tanh或ReLU）

3. **输出计算**：
   RNN的输出 $y_t \in \mathbb{R}^k$ 可以在每个时间步计算，它是隐藏状态的函数，公式为：

$$y_t = g(W_y h_t + b_y)$$

   其中：
   - $W_y \in \mathbb{R}^{k \times m}$ 是输出的权重矩阵
   - $b_y \in \mathbb{R}^k$ 是输出的偏置向量
   - $g(\cdot)$ 是激活函数（例如softmax或线性激活）
4. **初始状态**：
   通常，RNN的初始隐藏状态 h_0 可以被初始化为零向量或作为可学习的参数。
5. **递归计算**：
   从时间步 $t = 1$ 到 T，重复上述的隐藏状态更新和输出计算过程。
6. **损失函数**：
   RNN的训练通过最小化预测输出 y_t 和实际标签之间的损失函数来完成。常用的损失函数包括交叉熵损失或均方误差。
7. **反向传播通过时间**（BPTT）：
   为了更新网络参数，RNN使用一种称为通过时间的反向传播（Backpropagation Through Time, BPTT）的算法，将误差反向传播到每个时间步，并计算每个参数的梯度。

ENCODER

- 输入为序列序列信息，输出为序列的向量信息。（结合GPT中嵌入矩阵的概念），词的嵌入只是表示词义信息，而embedding也会将位置信息编入向量

DECODER

- $auo-regressive$（自回归）：后面的输出结果可以利用前面的输出结果。

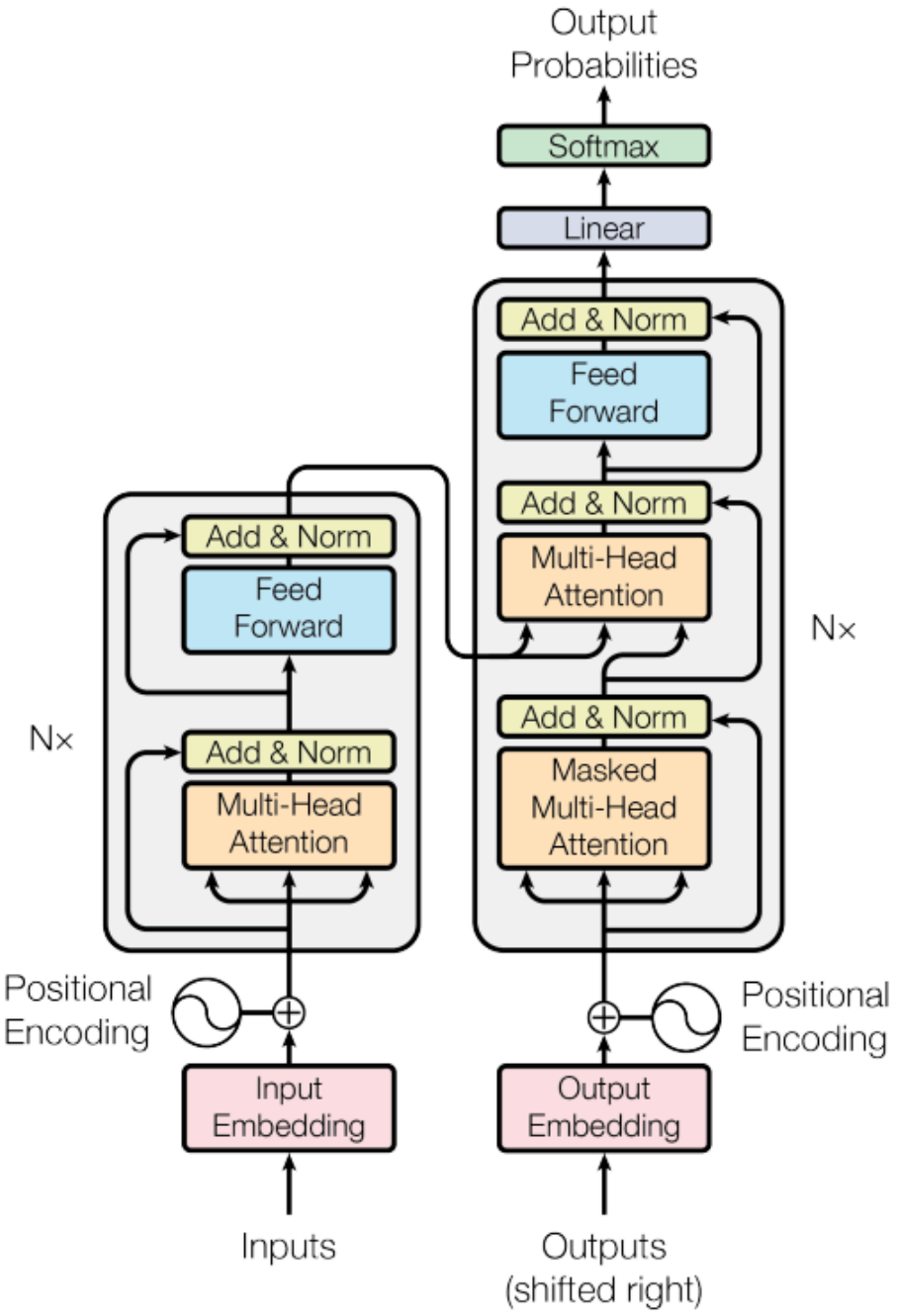


Figure 1: The Transformer - model architecture.

- input-embedding：嵌入层（将词转换为向量）
- BatchNorm(针对每一个特征数据标准化):

数据标准化的过程

数据标准化通常是将数据缩放到均值为0，标准差为1的分布。标准化的公式如下：

$$z = \frac{x - \mu}{\sigma}$$

其中：

- x 是原始数据点
- μ 是数据的均值
- σ 是数据的标准差

标准化的步骤如下：

1. **计算均值** μ：

$$\mu = \frac{1}{N}\sum_{i=1}^{N} x_i$$

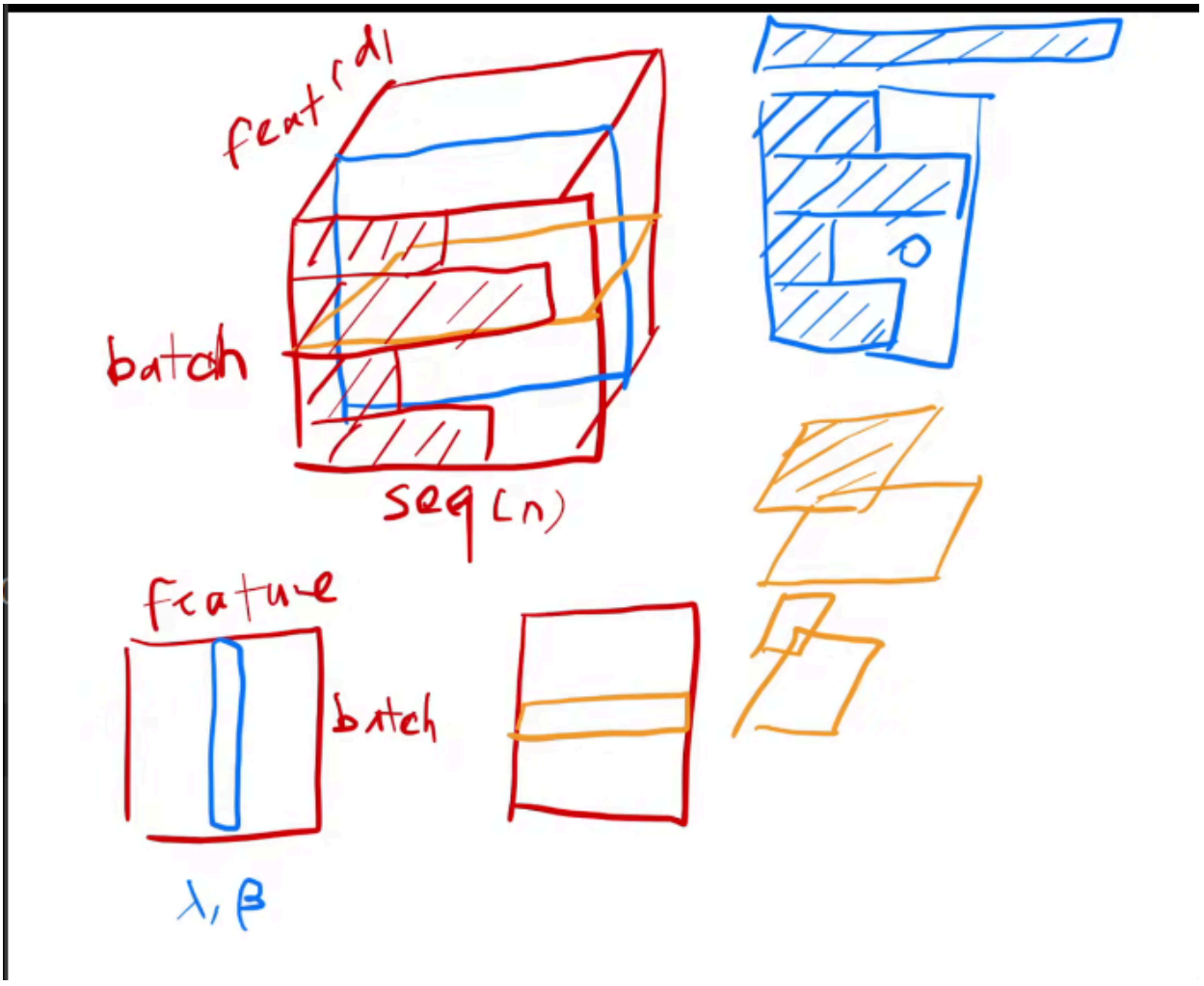
2. **计算标准差** σ：

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2}$$

3. **应用标准化公式**：
   对每个数据点 x_i，计算对应的标准化值 z_i：

$$z_i = \frac{x_i - \mu}{\sigma}$$

- LayerNorm(针对每一个样本数据标准化)：主要目的为适应不同的sequence的输入和输出



注意力层

Transformer中的注意力机制公式计算过程

在Transformer中，注意力机制的核心是**Scaled Dot-Product Attention**，其计算过程如下：

1. **输入表示**：
   假设输入的序列长度为 n，每个输入向量的维度为 d_k。我们有三个矩阵：
   - **查询矩阵** (Query): $Q \in \mathbb{R}^{n \times d_k}$
   - **键矩阵** (Key): $K \in \mathbb{R}^{n \times d_k}$
   - **值矩阵** (Value): $V \in \mathbb{R}^{n \times d_v}$
2. **计算注意力得分**：
   首先，计算查询矩阵 Q 与键矩阵 K 的点积，并除以 $\sqrt{d_k}$ 进行缩放，得到注意力得分矩阵：

$$\text{Attention Scores} = \frac{QK^T}{\sqrt{d_k}}$$

3. **应用Softmax函数**：
   对注意力得分矩阵的每一行应用Softmax函数，以获得注意力权重：

$$\text{Attention Weights} = \text{Softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)$$

4. **计算注意力输出**：
   最后，将注意力权重与值矩阵 V 相乘，得到最终的注意力输出：

$$\text{Output} = \text{Attention Weights} \times V$$

5. **多头注意力** (Multi-Head Attention):
   在多头注意力机制中，我们重复上述步骤 h 次，每次使用不同的 Q, K, V 矩阵（通常通过线性变换得到）。然后将每个头的输出连接起来并应用线性变换：

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \text{head}_2, \ldots, \text{head}_h)W^O$$

   其中 $\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$，$W_i^Q$, W_i^K, W_i^V 和 W^O 为可学习的权重矩阵。

补充：

- $\sqrt{d_k}$用于减小数字的向0-1两端分布的特性（防止梯度减小）。

  While for small values of d_k the two mechanisms perform similarly, additive attention outperforms dot product attention without scaling for larger values of d_k [3]. We suspect that for large values of d_k, the dot products grow large in magnitude, pushing the softmax function into regions where it has extremely small gradients [4]. To counteract this effect, we scale the dot products by $\frac{1}{\sqrt{d_k}}$.

- W_i^Q, W_i^K, W_i^V 将矩阵投影到低维，W^O将矩阵投影到高维，体现为图中的linear层。

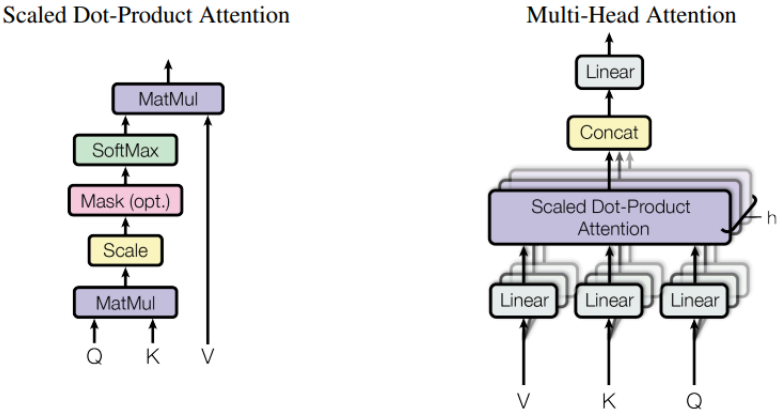
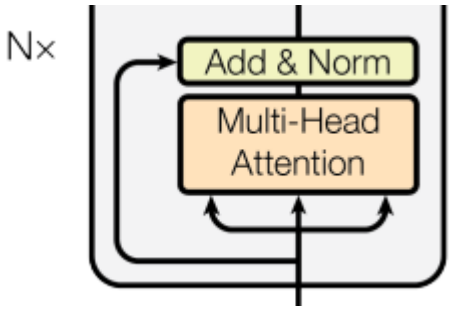


Figure 2: (left) Scaled Dot-Product Attention. (right) Multi-Head Attention consists of several attention layers running in parallel.

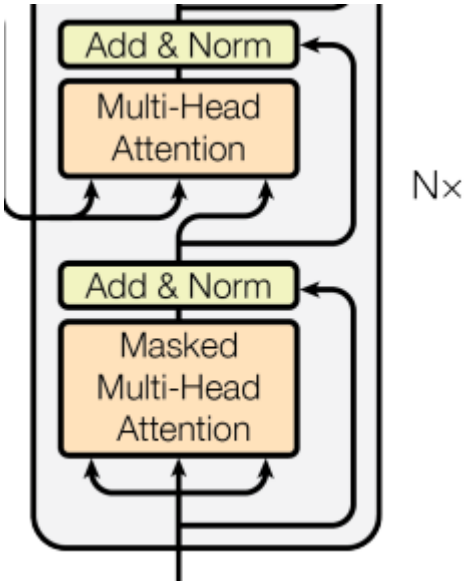
- [mask]部分的操作和3d是基本一致的，-∞导致softmax出来的值为0。
- 多头注意力机制的好处：具有可调参数的优点

注意力机制的具体使用

自注意力层：



- 此时的Q,K,V均为输入的矩阵，表示为：输出是输入的加权和
- 这种自注意力机制相当于将一个词序列进行语义的融合，使得所有的嵌入向量形成适合语境的词向量。



- [mask]自注意力块的输出作为下一个块的query
- 而输入的自注意层作为key和value
- 结合机器翻译：汉语作为输入，英文作为输出，英文的向量query来决定汉语向量的权重，并且形成汉语的权重相加向量，最终输出汉语
- 线性层主要是做语义空间的转换

$$\mathrm{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2 \tag{2}$$

While the linear transformations are the same across different positions, they use different parameters from layer to layer. Another way of describing this is as two convolutions with kernel size 1. The dimensionality of input and output is $d_{\text{model}} = 512$, and the inner-layer has dimensionality $d_{ff} = 2048$.

- 生成一个512的向量与嵌入向量相加完成位置信息的编码

In this work, we use sine and cosine functions of different frequencies:

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$

卷积在NLP中的应用

$training$

- ## BPE的介绍
- BPE是一种数据压缩算法，最初用于压缩算法的字符替换。其核心思想是通过迭代地合并最频繁出现的字符或字符序列对，来减少表示长度。在自然语言处理中，BPE的作用是将频繁出现的子词或字符组合合并，从而形成更紧凑的词汇表。
- **算法步骤**：

1. **初始化词汇表**：从训练数据中的所有字符开始，构建初始词汇表（每个字符作为一个单独的符号）。
2. **统计字符对频率**：计算在文本数据中相邻字符对的出现频率。
3. **合并频率最高的字符对**：将频率最高的字符对合并为一个新的符号，并更新词汇表。
4. **重复步骤2和3**：迭代执行，直到达到预定的词汇表大小或其他停止条件。
5. **最终输出**：一个词汇表，其中包含了单个字符和常见子词的组合。

Regularization

- dropout->label smoothing
- ## dropout
- **Dropout** 是一种常用的正则化技术，广泛应用于神经网络（包括大语言模型）的训练中。它的主要目的是减少模型的过拟合，并提高泛化能力。Dropout通过随机“丢弃”部分神经元来打破神经元之间的依赖性，从而增强模型的鲁棒性。
- **操作步骤**：

1. **训练时**：
   - 在训练过程中，对于每一层的每个神经元，以概率 p （通常在0.5左右）决定是否将其丢弃。
   - 被丢弃的神经元不会参与该次迭代的计算，这意味着它们的输出被暂时置为0。
2. **测试时**：
   - 在测试时，所有神经元都被保留，但为了保持输出的一致性，将神经元的输出乘以一个缩放因子（通常是训练时的保留概率 $1-p$），以补偿训练时丢弃神经元的影响。

- ## label smoothing

Label Smoothing 在大模型领域的介绍

Label Smoothing 是一种用于分类问题的正则化技术，旨在提高模型的泛化能力，减少过拟合现象。它通过对训练数据的标签进行平滑处理，防止模型过度自信地预测特定类别，从而增强模型的鲁棒性。

1. Label Smoothing 的基本原理

在传统的分类任务中，标签通常是 one-hot 编码的，即某个样本属于一个类别，其对应的标签为 1，其他类别的标签为 0。例如，在一个三分类问题中，如果一个样本的真实类别是类 A，则它的标签表示为 `[1, 0, 0]`。

Label Smoothing 对此进行了一些调整，将标签分布从完全确定的 one-hot 分布变为一个平滑的分布。具体来说，它将目标类别的概率值从 1 减少为一个稍小的值，而对其他类别分配一些非零的概率。这种调整有助于减少模型过度自信，从而提升模型的泛化性能。

公式:
假设有 K 个类别，真实标签是 one-hot 编码的，那么使用 label smoothing 后的标签分布为：

$$y_{\text{smooth}} = (1 - \epsilon) \cdot y_{\text{one-hot}} + \frac{\epsilon}{K}$$

其中：

- y_{smooth} 是经过平滑处理后的标签分布。
- ϵ 是平滑参数，通常是一个小值，如 0.1。
- K 是类别的数量。

这意味着对于目标类别的概率被调整为 $1 - \epsilon + \frac{\epsilon}{K}$，而其他类别的概率被调整为 $\frac{\epsilon}{K}$。

2. Label Smoothing 的优点

- **防止过拟合**：通过避免模型对训练数据中的某些标签过于自信，Label Smoothing 有助于减少过拟合。
- **提高泛化能力**：在验证集或测试集上，模型在面对噪声或数据分布略有不同的情况下表现得更加鲁棒。
- **处理不确定性**：Label Smoothing 对可能存在标签噪声的数据也更加友好，因为它不会假设所有标签都是完全正确的。