大模型之模型架构

语言模型的一开始就可以被看做是一个黑箱，当前大规模语言模型的能力在于给定一个基于自身需求的prompt就可以生成符合需求的结果。形式可以表达为：

$$prompt \rightsquigarrow completion$$

从数学的角度考虑就对训练数据 (traing data: （$x_1, \ldots, x_L$）)的概率分布：

$$trainingData \Rightarrow p(x_1, \ldots, x_L).$$

在学习内容中中，我们将彻底揭开面纱，讨论大型语言模型是如何构建的。今天的内容将着重讨论两个主题，分别是分词和模型架构：

- 分词：即如何将一个字符串拆分成多个标记。
- 模型架构：我们将主要讨论Transformer架构，这是真正实现大型语言模型的建模创新。

分词

回顾刚才的内容，语言模型 p是一个对标记（token）序列的概率分布，其中每个标记来自某个词汇表V：

$[the, mouse, ate, the, cheese]$

然而，自然语言并不是以标记序列的形式出现，而是以字符串的形式存在（具体来说，是Unicode字符的序列），比如上面的序列的自然语言为“**the mouse ate the cheese**”。

分词器将任意字符串转换为标记序列：the mouse ate the cheese $\Rightarrow [the, mouse, ate, the, cheese]$

这并不一定是语言建模中最引人注目的部分，但在确定模型的工作效果方面起着非常重要的作用。我们也可以将这个方式理解为自然语言和机器语言的一种显式的对齐。下面我对分词的一些细节进一步的讨论。

基于空格的分词

最简单的解决方案是使用`text.split(' ')`方式进行分词，这种分词方式对于英文这种按照空格，且每个分词后的单词有语义关系的文本是简单而直接的分词方式。然而，对于一些语言，如中文，句子中的单词之间没有空格：

”我今天去了商店。”

还有一些语言，比如德语，存在着长的复合词（例如Abwasserbehandlungsanlange）。即使在英语中，也有连字符词（例如father-in-law）和缩略词（例如don't），它们需要被正确拆分。例如，Penn Treebank将don't拆分为do和n't，这是一个在语言上基于信息的选择，但不太明显。因此，仅仅通过空格来划分单词会带来很多问题。

那么，什么样的分词才是好的呢？目前从直觉和工程实践的角度来说：

- 首先我们不希望有太多的标记（极端情况：字符或字节），否则

序列会变得难以建模。
- 其次我们也不希望标记过少，否则单词之间就无法共享参数（例如，mother-in-law和father-in-law应该完全不同吗？），这对于形态丰富的语言尤其是个问题（例如，阿拉伯语、土耳其语等）。
- 每个标记应该是一个在语言或统计上有意义的单位。

Byte pair encoding

将字节对编码（BPE）算法应用于数据压缩领域，用于生成其中一个最常用的分词器。BPE分词器需要通过模型训练数据进行学习，获得需要分词文本的一些频率特征。

学习分词器的过程，直觉上，我们将每个字符作为自己的标记，并组合那些经常共同出现的标记。整个过程可以表示为：

- 输入：训练语料库（字符序列）。
- 初始化词汇表V为字符的集合。
- 当我们仍然希望V继续增长时：
  找到V中共同出现次数最多的元素对x, x'。
- 用一个新的符号xx'替换所有x, x'的出现。将
- xx'添加到V中。

这里举一个例子：

1 $[t, h, e, c, c, a, r]$, $[t, h, e, c, c, a, t]$, $[t, h, e, c, r, a, t]$
2 [th, e, , $c, a, r]$, $[th, e, u, c, a, t]$, $[th, e, c, r, a, t]$ (th 出现了 3次)
3 [the, $\leq, c, a, r]$, [the, $\sqcup, c, a, t]$, [the, $u, r, a, t]$ (the 出现了 3次)
4 [the, $\sqcup, ca, r]$, [the, $u, ca, t]$, [the, $\sqcup, r, a, t]$ (ca 出现了 2次)

Unicode的问题

Unicode（统一码）是当前主流的一种编码方式。其中这种编码方式对BPE分词产生了一个问题（尤其是在多语言环境中），Unicode字符非常多（共144,697个字符）。在训练数据中我们不可能见到所有的字符。
为了进一步减少数据的稀疏性，我们可以对字节而不是Unicode字符运行BPE算法（Wang等人，2019年）。
以中文为例：

$$FeedForwardSequenceModel_{今天} \Rightarrow FeedForwardSequenceModel[x62, x11, 4e, ca]$$

BPE算法在这里的作用是为了进一步减少数据的稀疏性。通过对字节级别进行分词，可以在多语言环境中更好地处理Unicode字符的多样性，并减少数据中出现的低频词汇，提高模型的泛化能力。通过使用字节编码，可以将不同语言中的词汇统一表示为字节序列，从而更好地处理多语言数据。

Unigram model (SentencePiece)

与仅仅根据频率进行拆分不同，一个更“有原则”的方法是定义一个目标函数来捕捉一个好的分词的特征，这种基于目标函数的分词模型可以适应更好分词场景，Unigram model就是基于这种动机提出的。我们现在描述一下unigram模型（Kudo，2018年）。

这是SentencePiece工具（Kudo & Richardson，2018年）所支持的一种分词方法，与BPE一起使用。
它被用来训练T5和Gopher模型。给定一个序列$x_{1:L}$，一个分词器T是$p\left(x_{1:L}\right)=\prod_{(i,j)\in T} p\left(x_{i:j}\right)$的一个集合。这边给出一个实例：

- 训练数据（字符串）：ababc
- 分词结果 $T=(1,2),(3,4),(5,5)$ （其中$V=\{\text{ab}, \text{c}\}$）
- 似然值：$p(x_{1:L})=2/3\cdot 2/3\cdot 1/3=4/9$

在这个例子中，训练数据是字符串"ababc"。分词结果 $T = (1,2),(3,4),(5,5)$ 表示将字符串拆分成三个子序列：(a,b)，(a,b)，(c)。词汇表$V = \{\text{ab},\text{c}\}$表示了训练数据中出现的所有词汇。

似然值 $p(x_{1:L})$是根据 unigram 模型计算得出的概率，表示训练数据的似然度。在这个例子中，概率的计算为 $2/3 \cdot 2/3 \cdot 1/3 = 4/9$ 。这个值代表了根据 unigram 模型，将训练数据分词为所给的分词结果 T的概率。

unigram 模型通过统计每个词汇在训练数据中的出现次数来估计其概率。在这个例子中，ab 在训练数据中出现了两次，c出现了一次。因此，根据 unigram 模型的估计，$p(\text{ab}) = 2/3$，$p(\text{c}) = 1/3$ 。通过将各个词汇的概率相乘，我们可以得到整个训练数据的似然值为 $4/9$。

似然值的计算是 unigram 模型中重要的一部分，它用于评估分词结果的质量。较高的似然值表示训练数据与分词结果之间的匹配程度较高，这意味着该分词结果较为准确或合理。

算法流程

- 从一个“相当大”的种子词汇表 V 开始。
- 重复以下步骤：
  - 给定 V ，使用EM算法优化 $p(x)$ 和 T 。
  - 计算每个词汇 $x \in V$ 的 $loss(x)$ ，衡量如果将 x 从 V 中移除，似然值会减少多少。
  - 按照 $loss$ 进行排序，并保留 V 中排名靠前的80%的词汇。

这个过程旨在优化词汇表，剔除对似然值贡献较小的词汇，以减少数据的稀疏性，并提高模型的效果。通过迭代优化和剪枝，词汇表会逐渐演化，保留那些对于似然值有较大贡献的词汇，提升模型的性能。

模型架构

到目前为止，我们已经将语言模型定义为对标记序列的概率分布 $p(x_1, \ldots, x_L)$，我们已经看到这种定义非常优雅且强大（通过提示，语言模型原则上可以完成任何任务，正如GPT-3所示）。然而，在实践中，对于专门的任务来说，避免生成整个序列的生成模型可能更高效。

上下文向量表征 (Contextual Embedding): 作为先决条件，主要的关键发展是将标记序列与相应的上下文的向量表征：

$$[the, mouse, ate, the, cheese] \overset{\phi}{\Rightarrow} \left[\begin{pmatrix}1\\0.1\end{pmatrix}, \begin{pmatrix}0\\1\end{pmatrix}, \begin{pmatrix}1\\1\end{pmatrix}, \begin{pmatrix}1\\-0.1\end{pmatrix}, \begin{pmatrix}0\\-1\end{pmatrix}\right].$$

正如名称所示，标记的上下文向量表征取决于其上下文（周围的单词）；例如，考虑mouse的向量表示需要关注到周围某个窗口大小的其他单词。

- 符号表示：我们将 $\phi : V^L \to \mathbb{R}^{d \times L}$ 定义为嵌入函数（类似于序列的特征映射，映射为对应的向量表示）。
- 对于标记序列 $x1 : L = [x_1, \ldots, x_L]$，$\phi$ 生成上下文向量表征 $\phi(x_{1:L})$。

语言模型分类

对于语言模型来说，最初的起源来自于Transformer模型，这个模型是编码-解码端 （Encoder-Decoder）的架构。但是当前对于语言模型的分类，将语言模型分为三个类型：编码端（Encoder-Only），解码端（Decoder-Only）和编码-解码端（Encoder-Decoder）。因此我们的架构展示以当前的分类展开。

编码端（Encoder-Only）架构

编码端架构的著名的模型如BERT、RoBERTa等。这些语言模型生成上下文向量表征，但不能直接用于生成文本。可以表示为，$x_{1:L} \Rightarrow \phi(x_{1:L})$。这些上下文向量表征通常用于分类任务（也呗称为自然语言理解任务）。任务形式比较简单，下面以情感分类/自然语言推理任务举例：

$$\text{情感分析输入与输出形式：} [[CLS], \text{他们}, \text{移动}, \text{而}, \text{强大}] \Rightarrow \text{正面情绪}$$

$$\text{自然语言处理输入与输出形式：} [[CLS], \text{所有}, \text{动物}, \text{都}, \text{喜欢}, \text{吃}, \text{饼干}, \text{哦}] \Rightarrow \text{蕴涵}$$

该架构的优势是对于文本的上下文信息有更好的理解，因此该模型架构才会多用于理解任务。该架构的有点是对于每个 xi，上下文向量表征可以双向地依赖于左侧上下文 $(x_{1:i-1})$ 和右侧上下文 $(x_{i+1:L})$。但是缺点在于不能自然地生成完成文本，且需要更多的特定训练目标（如掩码语言建模）。

解码器（Decoder-Only）架构

解码器架构的著名模型就是大名鼎鼎的GPT系列模型。这些是我们常见的自回归语言模型，给定一个提示 $x_{1:i}$，它们可以生成上下文向量表征，并对下一个标记 x_{i+1}（以及递归地，整个完成 $x_{i+1:L}$）生成一个概率分布。$x_{1:i} \Rightarrow \phi(x_{1:i}), p(x_{i+1} \mid x_{1:i})$。我们以自动补全任务来说，输入与输出的形式为，$[[CLS], \text{他们}, \text{移动}, \text{而}] \Rightarrow \text{强大}$。

与编码端架构比，其优点为能够自然地生成完成文本，有简单的训练目标（最大似然）。缺点也很明显，对于每个 xi，上下文向量表征只能单向地依赖于左侧上下文 ($x_{1:i-1}$)。

编码-解码端（Encoder-Decoder）架构

编码-解码端架构就是最初的Transformer模型，其他的还有如BART、T5等模型。这些模型在某种程度上结合了两者的优点：它们可以使用双向上下文向量表征来处理输入 $x_{1:L}$，并且可以生成输出 $y_{1:L}$。可以公式化为：

$$x1:L \Rightarrow \phi(x1:L), p(y1:L \mid \phi(x1:L))。$$

以表格到文本生成任务为例，其输入和输出的可以表示为：

$$\left[\text{名称 :, 植物, |, 类型 :, 花卉, 商店}\right] \Rightarrow \left[\text{花卉, 是, 一, 个, 商店}\right]。$$

该模型的具有编码端，解码端两个架构的共同的优点，对于每个 x_i，上下文向量表征可以双向地依赖于左侧上下文 ($x_{1:i-1}$) 和右侧上下文 ($x_{i+1:L}$)，可以自由的生成文本数据。缺点就说需要更多的特定训练目标。

语言模型理论

下一步，我们会介绍语言模型的模型架构，重点介绍Transformer架构机器延伸的内容。另外我们对于架构还会对于之前RNN网络的核心知识进行阐述，其目的是对于代表性的模型架构进行学习，为未来的内容增加知识储备。

深度学习的美妙之处在于能够创建构建模块，就像我们用函数构建整个程序一样。因此，在下面的模型架构的讲述中，我们能够像下面的函数一样封装，以函数的的方法进行理解:

$$TransformerBlock(x_{1:L})$$

为了简单起见，我们将在函数主体中包含参数，接下来，我们将定义一个构建模块库，直到构建完整的Transformer模型。

基础架构

首先，我们需要将标记序列转换为序列的向量形式。$EmbedToken$函数通过在嵌入矩阵$E \in \mathbb{R}^{|v| \times d}$中查找每个标记所对应的向量，该向量的具体值这是从数据中学习的参数：

def $EmbedToken(x_{1:L} : V^L) \rightarrow \mathbb{R}^{d \times L}$:

- 将序列$x_{1:L}$中的每个标记xi转换为向量。
- 返回[Ex1,...,ExL]。

以上的词嵌入是传统的词嵌入，向量内容与上下文无关。这里我们定义一个抽象的$SequenceModel$函数，它接受这些上下文无关的嵌入，并将它们映射为上下文相关的嵌入。

$defSequenceModel(x_{1:L} : \mathbb{R}^{d \times L}) \rightarrow \mathbb{R}^{d \times L}$:

- 针对序列$x_{1:L}$中的每个元素xi进行处理，考虑其他元素。
- [抽象实现（例如，$FeedForwardSequenceModel$，$SequenceRNN$，$TransformerBlock$）]

最简单类型的序列模型基于前馈网络（Bengio等人，2003），应用于固定长度的上下文，就像n-gram模型一样，函数的实现如下：

def $FeedForwardSequenceModel(x_{1:L} : \mathbb{R}^{d \times L}) \rightarrow \mathbb{R}^{d \times L}$:

- 通过查看最后n个元素处理序列$x_{1:L}$中的每个元素xi。
- 对于每个$i = 1, \ldots, L$:

    - 计算h_i=$FeedForward(x_{i-n+1},\ldots,x_i)$。
- 返回[$h_1,\ldots,h_L$]。

递归神经网络

第一个真正的序列模型是递归神经网络（RNN），它是一类模型，包括简单的RNN、LSTM和GRU。基本形式的RNN通过递归地计算一系列隐藏状态来进行计算。

def $SequenceRNN(x:\mathbb{R}^{d\times L})\to\mathbb{R}^{d\times L}$:

- 从左到右处理序列$x_1,\ldots,x_L$，并递归计算向量$h_1,\ldots,h_L$。
- 对于$i=1,\ldots,L$:
    - 计算$h_i=RNN(h_{i-1},x_i)$。
    - 返回$[h_1,\ldots,h_L]$。

实际完成工作的模块是RNN，类似于有限状态机，它接收当前状态h、新观测值x，并返回更新后的状态：

def $RNN(h:\mathbb{R}^d,x:\mathbb{R}^d)\to\mathbb{R}^d$:

- 根据新的观测值x更新隐藏状态h。
- [抽象实现（例如，SimpleRNN，LSTM，GRU）]

有三种方法可以实现RNN。最早的RNN是简单RNN（Elman, 1990），它将h和x的线性组合通过逐元素非线性函数σ（例如，逻辑函数$\sigma(z)=(1+e-z)-1$或更现代的$ReLU$函数$\sigma(z)=max(0,z)$）进行处理。

def $SimpleRNN(h:\mathbb{R}d,x:\mathbb{R}d)\to\mathbb{R}d$:

- 通过简单的线性变换和非线性函数根据新的观测值x更新隐藏状态h。

- 返回$\sigma(Uh + Vx + b)$。

正如定义的RNN只依赖于过去，但我们可以通过向后运行另一个RNN来使其依赖于未来两个。这些模型被ELMo和ULMFiT使用。

def $BidirectionalSequenceRNN(x_{1:L} : \mathbb{R}^{d \times L}) \rightarrow \mathbb{R}^{2d \times L}$\:

- 同时从左到右和从右到左处理序列。
- 计算从左到右：
  $[h \rightarrow_1, \ldots, h \rightarrow_L] \leftarrow SequenceRNN(x_1, \ldots, x_L)$。
- 计算从右到左：
  $[h \leftarrow_L, \ldots, h \leftarrow_1] \leftarrow SequenceRNN(x_L, \ldots, x_1)$。
- 返回$[h \rightarrow_1 h \leftarrow_1, \ldots, h \rightarrow_L h \leftarrow_L]$。

注：

- 简单RNN由于梯度消失的问题很难训练。
- 为了解决这个问题，发展了长短期记忆（LSTM）和门控循环单元（GRU）（都属于RNN）。
- 然而，即使嵌入h200可以依赖于任意远的过去（例如，x1），它不太可能以“精确”的方式依赖于它（更多讨论，请参见Khandelwal等人，2018）。
- 从某种意义上说，LSTM真正地将深度学习引入了NLP领域。

Transformer

现在，我们将讨论Transformer（Vaswani等人，2017），这是真正推动大型语言模型发展的序列模型。正如之前所提到的，Transformer模型将其分解为Encoder-Only（GPT-2，GPT-3）、Decoder-Only（BERT，RoBERTa）和Encoder-Decoder（BART，T5）模型的构建模块。

关于Transformer的学习资源有很多：

- Illustrated Transformer和Illustrated GPT-2：对Transformer的视觉描述非常好。
- Annotated Transformer：Transformer的Pytorch实现。

强烈建议您阅读这些参考资料。该课程主要依据代码函数和接口进行讲解。

注意力机制

Transformer的关键是注意机制，这个机制早在机器翻译中就被开发出来了（Bahdananu等人，2017）。可以将注意力视为一个“软”查找表，其中有一个查询y，我们希望将其与序列$x_{1:L} = [x_1, \ldots, x_L]$的每个元素进行匹配。我们可以通过线性变换将每个$x_i$视为表示键值对：

$$(W_{key} x_i) : (W_{value} x_i)$$

并通过另一个线性变换形成查询：

$$W_{query} y$$

可以将键和查询进行比较，得到一个分数：

$$score_i = x_i^\top W_{key}^\top W_{query} y$$

这些分数可以进行指数化和归一化，形成关于标记位置$1, \ldots, L$的概率分布：

$$[\alpha_1, \ldots, \alpha_L] = softmax([score_1, \ldots, score_L])$$

然后最终的输出是基于值的加权组合：

$$\sum_{i=1}^{L} \alpha_i \, (W_{value} x_i)$$

我们可以用矩阵形式简洁地表示所有这些内容：

def $Attention(x_{1:L} : \mathbb{R}^{d \times L}, y : \mathbb{R}^d) \to \mathbb{R}^d$:

- 通过将其与每个x_i进行比较来处理y。
- 返回$W_{value} x_{1:L} \operatorname{softmax}\left(x_{1:L}^\top W_{key}^\top W_{query} y / \sqrt{d}\right)$

我们可以将注意力看作是具有多个方面（例如，句法、语义）的匹配。为了适应这一点，我们可以同时使用多个注意力头，并简单地组合它们的输出。

def $MultiHeadedAttention(x_{1:L} : \mathbb{R}^{d \times L}, y : \mathbb{R}^d) \to \mathbb{R}^d$:

- 通过将其与每个xi与nheads个方面进行比较，处理y。
- 返回$W_{output} [\underbrace{[\text{Attention}(x_{1:L}, y), \ldots, \text{Attention}(x_{1:L}, y)]}_{n_{heads} times}$

对于**自注意层**，我们将用x_i替换y作为查询参数来产生，其本质上就是将自身的x_i对句子的其他上下文内容进行$Attention$的运算：

def $SelfAttention(x_{1:L} : \mathbb{R}_{d \times L}) \to \mathbb{R}_{d \times L})$:

- 将每个元素xi与其他元素进行比较。
- 返回$[Attention(x_{1:L}, x_1), \ldots, Attention(x_{1:L}, x_L)]$。

自注意力使得所有的标记都可以“相互通信”，而**前馈层**提供进一步的连接：

def $FeedForward(x_{1:L} : \mathbb{R}^{d \times L}) \to \mathbb{R}^{d \times L}$:

- 独立处理每个标记。
- 对于$i = 1, \ldots, L$:
  - 计算$y_i = W_2 max(W_1 x_i + b_1, 0) + b_2$。
- 返回$[y_1, \ldots, y_L]$。

对于Transformer的主要的组件，我们差不多进行介绍。原则上，我们可以只需将$FeedForward \circ SelfAttention$序列模型迭代96次以构建GPT-3，但是那样的网络很难优化（同样受到沿深度方向的梯度消失问题的困扰）。因此，我们必须进行两个手段，以确保网络可训练。

残差连接和归一化

残差连接：计算机视觉中的一个技巧是残差连接（ResNet）。我们不仅应用某个函数f：

$$f(x1:L),$$

而是添加一个残差（跳跃）连接，以便如果f的梯度消失，梯度仍然可以通过$x_{1:L}进行计算：

$$x_{1:L} + f(x_{1:L})。$$

层归一化:另一个技巧是层归一化，它接收一个向量并确保其元素不会太大：

def $LayerNorm(x_{1:L} : \mathbb{R}^{d \times L}) \to \mathbb{R}^{d \times L}$:

- 使得每个x_i既不太大也不太小。

我们首先定义一个适配器函数，该函数接受一个序列模型f并使其“鲁棒”：

def $AddNorm(f : (\mathbb{R}d^{\times L} \to \mathbb{R}^{d \times L}), x_{1:L} : \mathbb{R}_{d \times L}) \to \mathbb{R}^{d \times L}$:

- 安全地将f应用于$x_{1:L}$。
- 返回$LayerNorm(x_{1:L} + f(x_{1:L}))$。

最后，我们可以简洁地定义Transformer块如下：

def $Transformer Block(x_{1:L}: \mathbb{R}^{d \times L}) \rightarrow \mathbb{R}^{d \times L}$:

- 处理上下文中的每个元素x_i。
- 返回$AddNorm(FeedForward, AddNorm(Self Attention, x_{1:L}))$。

位置嵌入

最后我们对目前语言模型的**位置嵌入**进行讨论。您可能已经注意到，根据定义，标记的嵌入不依赖于其在序列中的位置，因此两个句子中的mouse将具有相同的嵌入，从而在句子位置的角度忽略了上下文的信息，这是不合理的。

$$[\text{the, mouse, ate, the, cheese}][\text{the, cheese, ate, the, mouse}]$$

为了解决这个问题，我们将位置信息添加到嵌入中：

def $EmbedTokenWithPosition(x_{1:L}: \mathbb{R}^{d \times L})$:

- 添加位置信息。
- 定义位置嵌入：
  - 偶数维度：$P_{i,2j} = sin(i/10000^{2j/dmodel})$
  - 奇数维度：$P_{i,2j+1} = cos(i/10000^{2j/dmodel})$
- 返回$[x_1 + P_1, \ldots, x_L + P_L]$。

上面的函数中，i表示句子中标记的位置，j表示该标记的向量表示维度位置。

最后我们来聊一下GPT-3。在所有组件就位后，我们现在可以简要地定义GPT-3架构，只需将Transformer块堆叠96次即可：

$$GPT-3(x_{1:L}) = TransformerBlock^{96}(EmbedTokenWithPosition(x_{1:L}))$$

架构的形状（如何分配1750亿个参数）：

- 隐藏状态的维度：dmodel=12288
- 中间前馈层的维度：dff=4dmodel
- 注意头的数量：nheads=96
- 上下文长度：L=2048

这些决策未必是最优的。Levine等人（2020）提供了一些理论上的证明，表明GPT-3的深度太深，这促使了更深但更宽的Jurassic架构的训练。

不同版本的Transformer之间存在重要但详细的差异：

- 层归一化“后归一化”（原始Transformer论文）与“先归一化”（GPT-2），这影响了训练的稳定性（Davis等人，2021）。
- 应用了丢弃（Dropout）以防止过拟合。
- GPT-3使用了sparse Transformer（稀释 Transformer）来减少参数数量，并与稠密层交错使用。
- 根据Transformer的类型（Encdoer-Only, Decoder-Only, Encdoer-Decoder），使用不同的掩码操作。

延伸阅读

分词:

- Between words and characters: A Brief History of Open-Vocabulary Modeling and Tokenization in NLP. *Sabrina J. Mielke, Zaid Alyafeai, Elizabeth Salesky, Colin Raffel, Manan Dey, Matthias Gallé, Arun Raja, Chenglei Si, Wilson Y. Lee, Benoît Sagot, Samson Tan*. 2021. Comprehensive survey of tokenization.
- Neural Machine Translation of Rare Words with Subword

Units. *Rico Sennrich, B. Haddow, Alexandra Birch*. ACL 2015. Introduces **byte pair encoding** into NLP. Used by GPT-2, GPT-3.

- Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation. *Yonghui Wu, M. Schuster, Z. Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, M. Krikun, Yuan Cao, Qin Gao, Klaus Macherey, J. Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Lukasz Kaiser, Stephan Gouws, Y. Kato, Taku Kudo, H. Kazawa, K. Stevens, George Kurian, Nishant Patil, W. Wang, C. Young, Jason R. Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, G. Corrado, Macduff Hughes, J. Dean*. 2016. Introduces **WordPiece**. Used by BERT.
- SentencePiece: A simple and language independent subword tokenizer and detokenizer for Neural Text Processing. *Taku Kudo, John Richardson*. EMNLP 2018. Introduces **SentencePiece**.

模型架构:

- Language Models are Unsupervised Multitask Learners. Introduces GPT-2.
- Attention is All you Need. *Ashish Vaswani, Noam M. Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin*. NIPS 2017.
- Illustrated Transformer
- CS224N slides on RNNs
- CS224N slides on Transformers
- Train Short, Test Long: Attention with Linear Biases Enables Input Length Extrapolation. *Ofir Press, Noah A. Smith, M.*

Lewis. 2021. Introduces **Alibi embeddings**.

- Transformer-XL: Attentive Language Models beyond a Fixed-Length Context. *Zihang Dai, Zhilin Yang, Yiming Yang, J. Carbonell, Quoc V. Le, R. Salakhutdinov*. ACL 2019. Introduces recurrence on Transformers, relative position encoding scheme.
- Generating Long Sequences with Sparse Transformers. *R. Child, Scott Gray, Alec Radford, Ilya Sutskever*. 2019. Introduces **Sparse Transformers**.
- Linformer: Self-Attention with Linear Complexity. *Sinong Wang, Belinda Z. Li, Madian Khabsa, Han Fang, Hao Ma*. 2020. Introduces **Linformers**.
- Rethinking Attention with Performers. *K. Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamás Sarlós, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Lukasz Kaiser, David Belanger, Lucy J. Colwell, Adrian Weller*. ICLR 2020. Introduces **Performers**.
- Efficient Transformers: A Survey. *Yi Tay, M. Dehghani, Dara Bahri, Donald Metzler*. 2020.

Decoder-only 架构:

- Language Models are Unsupervised Multitask Learners. *Alec Radford, Jeff Wu, R. Child, D. Luan, Dario Amodei, Ilya Sutskever*. 2019. Introduces **GPT-2** from OpenAI.
- Language Models are Few-Shot Learners. *Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, J. Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, T. Henighan, R. Child, A. Ramesh, Daniel M.*

Ziegler, Jeff Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, Dario Amodei. NeurIPS 2020. Introduces **GPT-3** from OpenAI.

- Scaling Language Models: Methods, Analysis&Insights from Training Gopher. *Jack W. Rae, Sebastian Borgeaud, Trevor Cai, Katie Millican, Jordan Hoffmann, Francis Song, J. Aslanides, Sarah Henderson, Roman Ring, Susannah Young, Eliza Rutherford, Tom Hennigan, Jacob Menick, Albin Cassirer, Richard Powell, G. V. D. Driessche, Lisa Anne Hendricks, Maribeth Rauh, Po-Sen Huang, Amelia Glaese, Johannes Welbl, Sumanth Dathathri, Saffron Huang, Jonathan Uesato, John F. J. Mellor, I. Higgins, Antonia Creswell, Nathan McAleese, Amy Wu, Erich Elsen, Siddhant M. Jayakumar, Elena Buchatskaya, D. Budden, Esme Sutherland, K. Simonyan, Michela Paganini, L. Sifre, Lena Martens, Xiang Lorraine Li, A. Kuncoro, Aida Nematzadeh, E. Gribovskaya, Domenic Donato, Angeliki Lazaridou, A. Mensch, J. Lespiau, Maria Tsimpoukelli, N. Grigorev, Doug Fritz, Thibault Sottiaux, Mantas Pajarskas, Tobias Pohlen, Zhitao Gong, Daniel Toyama, Cyprien de Masson d'Autume, Yujia Li, Tayfun Terzi, Vladimir Mikulik, I. Babuschkin, Aidan Clark, Diego de Las Casas, Aurelia Guy, Chris Jones, James Bradbury, Matthew Johnson, Blake A. Hechtman, Laura Weidinger, Iason Gabriel, William S. Isaac, Edward Lockhart, Simon Osindero, Laura Rimell, Chris Dyer, Oriol Vinyals, Kareem W. Ayoub, Jeff Stanway, L. Bennett, D. Hassabis, K. Kavukcuoglu, Geoffrey Irving*. 2021. Introduces **Gopher** from DeepMind.
- Jurassic-1: Technical details and evaluation. *Opher Lieber, Or*

Sharir, Barak Lenz, Yoav Shoham. 2021. Introduces **Jurassic** from AI21 Labs.

Encoder-only 架构:

- BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *Jacob Devlin, Ming-Wei Chang, Kenton Lee, Kristina Toutanova*. NAACL 2019. Introduces **BERT** from Google.
- RoBERTa: A Robustly Optimized BERT Pretraining Approach. *Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, M. Lewis, Luke Zettlemoyer, Veselin Stoyanov*. 2019. Introduces **RoBERTa** from Facebook.

Encoder-decoder 架构:

- BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension. *M. Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, Luke Zettlemoyer*. ACL 2019. Introduces **BART** from Facebook.
- Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. *Colin Raffel, Noam M. Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, W. Li, Peter J. Liu*. J. Mach. Learn. Res. 2019. Introduces **T5** from Google.