BERT(Bidirectional Encoder Representations from Transformers)

BERT(Bidirectional Encoder Representations from Transformers) 是由 Google 在 2018 年提出的基于 Transformer 的预训练语言模型，

开创了自然语言处理（NLP）领域 `预训练-微调` 范式的先河。

1. 它通过大规模**无监督预训练**捕捉文本的深层语义，
2. 再通过**微调适配下游任务**，显著提升了多项 NLP 任务的性能。

（1）创新点解释

- 大规模无监督预训练

这里我偏向于解释为 **自监督** ，也就是**一开始训练的样本是没有label标注的，但是我们通过创建一些任务，使得model能够完成这些·任务，进而使得我们认为，模型具备了这些功能**

- 微调适配下游任务

这里就暗藏一个知识点，要是适配不同的任务，那么他最后的输出就有所不同，但是**主干的模块相同**

实际上，在无监督学习的阶段，我们分配给他一些任务，最后对主干模块输出的处理，例如全连接层等，与最终我们要将模型应用的时候最后输出的处理可能存在不同

然而模型的主干模块在预训练之后具备了一定的能力，使得这些输出后模块的改变来完成不同的任务成为可能

（2）模型结构

2.1 主干网络

模型的主干网络就是一个**编码器** ，注意这里只需要编码器，因为模型的能力重在**提取信息**

下面是两种模型的基本架构：

- **两种规格**（原论文）：
  - **BERT-Base**：12 层，768 隐藏维度，12 注意力头（110M 参数）。
  - **BERT-Large**：24 层，1024 隐藏维度，16 注意力头（340M 参数）。
- **注意力头**：
  -
    - **多头自注意力机制**（12 个头，隐藏维度 768）。
    - **前馈网络（FFN）**：两层全连接（中间维度 3072，GELU 激活）。
    - **残差连接 + 层归一化**：应用于自注意力和 FFN 后。
- 双向实现：
  - 每个注意力头允许关注上下文所有的信息，也就是不需要**掩码**

2.2 对于输出的处理

1. 输入表示

每个输入 Token 的嵌入由三部分组成：

- **Token Embeddings**：词嵌入向量，也就是对照到单词表
  - **分词方式**：使用 **WordPiece** 分词（将词拆分为子词，如 "unwanted" → "un", "##want", "##ed"）。
  - **特殊标记**：
    - `[CLS]`：分类任务的全局标记（位于序列开头）。
    - `[SEP]`：分隔句子对的标记。
    - `[MASK]`：预训练时用于遮盖词的标记。（对于MLM预训练阶段，在下文讲解）。
    - `[UNK]`：未知词标记。
- **Segment Embeddings**：区分句子对的嵌入（如问答中的问题和答案）。
  - 值为 0 或 1，标记当前 Token 属于第一个句子（Segment 0）还是第二个句子（Segment 1）。
- **Position Embeddings**：学习的位置编码。
  - 与 Transformer 不同，BERT 使用 **可学习的一维位置编码（非固定正弦函数）**。

$$\text{公式：} INPUT = Emd_{Token} + Emd_{Seg} + Emd_{Pos}$$

```
SegEmbedding: [sentense 1]:I love apple.      [sentense 2]:I love banana.
After Embedding: I[1] love[1] apple[1].       I[2] love[2] banana[2].
```

```
PosEmbedding:[sentense]:I love apple.
After Embedding:  I[1]  love[2]  apple[3] .
```

```
TokenEmbedding: [sentense 1]:I love apple.      [sentense 2]:I love banana.
After Embedding: [CLS]  I love apple  [Sep]  I love banana [SEP].
```

2.3 归一化的位置

以 **BERT** 为例（纯编码器结构），归一化出现在以下位置：

(1) 自注意力子层后

- **结构**：
  `输入 → 自注意力 → 残差连接 → LayerNorm → FFN → 残差连接 → LayerNorm`
- **作用**：
  - 稳定自注意力输出的分布，防止梯度消失或爆炸。
  - 经典Transformer论文（Vaswani et al., 2017）在残差连接 **之后** 使用LayerNorm（Post-LN）。

(2) 前馈神经网络（FFN）子层后

- 在FFN的输出后同样会经过LayerNorm，与自注意力子层一致。

(3) 为什么用LayerNorm而非BatchNorm?

- **序列数据特性**：BatchNorm依赖Batch内统计量，而文本序列长度可变，**BatchNorm不稳定**。
- **训练与推理一致性**：**LayerNorm对单样本独立**计算，不受Batch大小影响。

2.4 激活函数的位置

（1）在FFN的激活函数

也就是全连接层使用的激活函数

一般使用**GELU激活函数**

（2）注意力模块之中的激活函数

在注意力模块内部，使用**sigmoid激活函数**来得到注意力分数

（3）在注意力模块之后不需要激活函数

对于激活函数为什么不需要使用，参考**CNN之中，mobileNet V2 之中线性瓶颈之中对于激活函数的讲解**

并且，自注意力机制以及通过**KQV的交互来学习对应的内容，本身以及具备足够的非线性**

这一个细节也是保持性能的关键原因之一

（3）预训练-MLM

MLM就是 **掩码语言建模（Masked Language Model, MLM）**，更形象的说，就是大家所说的**完形填空**

为什么要分配MLM任务在作为预训练呢?

实际上我们在这里要给模型的能力就是提取文本信息，总结文本信息的能力

同时，我们要在没有标签的样本之中来实现这个操作，操作步骤如下：

1. **随机掩码**:

对于源文本，采取掩码的**策略：对于源文本的15% token 进行掩码处理**：对于15%掩码之中还有区别

- 80% 直接替换成为 `[mask]`
- 10%替换成为随机token
- 10%保持原来的token，防止模型过度依赖mask

所以现在的文本belike

```
I want to learn machine learning. ⟶ I [mask] to learn [mask] learning.
这就是掩码之后的效果。
```

2. **模型预测：**

- 经过输入处理之后输入model
- 经过自注意力模块，输出和输出**形状不变的向量（batch , seq_len , word_dim）**
- **取出掩码位置的向量，作为预测的对象，形成最终的输出（batch , masked_word , word_dim）**（这里类似于**ViT**的分类思想）
- **经过全连接层的处理，得到Logit**
- **logit经过softmax函数，最终映射到词汇表**
- **交叉熵函数计算损失**

（4）预训练-NSP

下一句预测（Next Sentence Prediction, NSP），更形象的说，就是七选五

更具体来说，就是选出两个句子，利用模型来判断是否属于连续的上下文句子

- **输入构造**：
  - **正样本**：从**文档**中连续选取两个句子（B 是 A 的下一句）。
  - **负样本**：随机**从其他文档**选取一个句子作为 B。
  - 输入格式：`[CLS] A [SEP] B [SEP]`。（添加了TokenEmb）
- **模型预测**：取 `[CLS]` 标记的输出向量，通过全连接层**二分类**（是/否下一句）。**（这里和MLM的思想类似）**

‘

这里要涉及到构造的**缺点**：

1. 对于**输入构造**：负样本是从其他文档之中获取，所以模型很可能学习的不是上下文连接，而是类似于**主题相似度**的一个东西，所以说，这里的采样有可能使得模型学习的侧重点不同，对于我们输入的同一个文档的不连续的句子的时候，他有可能会出现比较大的差错
2. 对于**CLS**：由于CLS最终是用于二分类，我们一开始把他解释为成保留了两个句子的向量，但是在实际操作之中，他被验证为不包含该信息。**他包含的信息，只有两个句子是不是，仅有是不是同一个句子的信息**

（5）微调下游任务

预训练后，BERT 通过简单调整适配具体任务：

5.1 文本分类（情感分析）

输入结构

- 格式：`[CLS] + 文本 + [SEP]`，例如 `[CLS] This movie is great! [SEP]`。
- **Segment Embedding**：全为 0（单句任务）。
- 这里的CLS可以看作休止符

’

注意力模块后的结构

1. **提取 [CLS] 标记的输出向量：**
   - **取最后一层 Transformer 编码器中 `[CLS]` 位置**的隐藏状态 $h_{cls} \in {}^{d}$（d 为隐藏维度，如 768）。
2. **分类头（Classifier Head）：**
   - **层归一化（可选）：** $h_{cls} = LayerNorm(h_{cls})$。
   - **全连接层：** $logits = W \cdot h_{cls} +$ ，其中 $W \in {}^{d \times num_classes}$。
   - **Softmax：**生成类别概率分布 $p = Softmax(logits)$。

5.2 命名实体识别（NER）

输入结构

- 格式：`[CLS] + Token1 + Token2 + ... + [SEP]`，例如 `[CLS] John lives in New York. [SEP]`。
- **Segment Embedding：**全为 0（单句任务）。

‘

注意力模块后的结构

1. **提取每个 Token 的输出向量：**
   - 对最后一层编码器的输出 $\in {}^{atch \times seqlen \times d}$ 取所有非特殊标记（如 `[CLS]`、`[SEP]`）的向量 $h_i \in {}^{d}$。
2. **序列标注头（NER Head）：**
   - **全连接层：**对每个 h_i 计算 $logits_i = W \cdot h_i +$ ，其中 $W \in {}^{d \times num_{tags}}$（如标签数为 5：PER, LOC, ORG, MISC, O）。
   - **Softmax/CRF：**
     - **Softmax：**独立预测每个 Token 的标签（**忽略标签间依赖**）。
     - **条件随机场（CRF）：**建模标签间转移概率（如 BIO 约束），提升序列一致性。

CRF解析

条件随机场是命名实体识别（NER）等序列标注任务中广泛使用的概率图模型，其核心思想是通过**建模标签间的转移约束，提升序列预测的合理性。**

- **1. 目的：**使得**标签转移**合法化。

例子：I（Person） love（0） China（Loc）.正确标签

I（Person） love（Loc） China（Loc）.错误标签

这之中包含了一些分类错误的情况。那就是标签转移需要解决的地方

正确标签：Person->0 , 0->Loc，这个逻辑符合我们句子的语义逻辑

错误标签：Person->Loc , Loc->Loc，通常情况下，我们在句子之中 不会把人名和地点一起连着说，这就是一种预测错误的情况

- **2. 策略**：评判每个标签偏移的合法性

训练一个标签转移矩阵，利用这个矩阵来**直接检索计算**标签偏移的合理性

- **总得分计算**：

$$Score = \sum_{i=1}^{n} (y_i, x, i) + \sum_{i=1}^{n} T_{y_i, y_{i+1}}$$

- **3. 例子解析**

输入句子

`"John lives in Paris"`

标签集

`{B-PER, I-PER, B-LOC, I-LOC, O}`

步骤 1：BERT 输出发射得分

假设 BERT 对每个 Token 的 `logits` 如下（已简化）：

Token	B-PER	I-PER	B-LOC	I-LOC	O
John	2.0	1.0	-1.0	-1.0	0.0
lives	-1.0	-1.0	-1.0	-1.0	1.0
in	-1.0	-1.0	-1.0	-1.0	1.0
Paris	-1.0	-1.0	2.0	1.0	0.0

步骤 2：定义转移矩阵

假设学习到的转移矩阵 T：

	B-PER	I-PER	B-LOC	I-LOC	O
B-PER	-1	1.0	-1	-1	0.0
I-PER	-1	0.5	-1	-1	0.5
B-LOC	-1	-1	-1	1.0	0.0

	B-PER	I-PER	B-LOC	I-LOC	O
I-LOC	-1	-1	-1	0.5	0.5
O	0.0	-1	0.0	-1	0.1

步骤 3：计算序列得分

假设候选序列 `[B-PER, O, O, B-LOC]`：

发射得分：

$2.0(ohn) + 1.0(lives) + 1.0(in) + 2.0(Paris) = 6.0$

转移得分：
$T_{B-PER,} + T_{,} + T_{,B-L} = 0.0 + 0.1 + 0.0 = 0.1$

总得分：

$6.0 + 0.1 = 6.1$

- **4. 解码：维特比算法（Viterbi Algorithm）**

寻找最优标签序列 $y^* = argmax_y P(y \ \ x)$：

1. **初始化**：
   - $_1() = (y, x, 1)$
   - $_1() = 0$
2. **递推**（t=2 到 n）：

$$t() = max_{1im}[_{t-1}(i) + T_{i,}] + (y, x, t)$$

$$_t() = argmax_{1im}[_{t-1}(i) + T_{i,}]$$

3. **终止与回溯**：
   - 最优路径得分：$max_n()$
   - 回溯路径：$y^*_n = argmax_n() \qquad y^*_{t-1} = _t(y^*_t)$

5.3 问答任务（如 SQuAD）

输入结构

- 格式：`[CLS] Question: ... [SEP] Context: ... [SEP]`，例如 `[CLS] Where was Einstein born? [SEP] Einstein was born in Ulm. [SEP]`。

- **Segment Embedding**：问题部分为 0，上下文部分为 1，在这里，上下文很可能不立马承接答案。

注意力模块后的结构

1. **提取上下文 Token 的输出向量**：
   - 取最后一层编码器的输出 $\in {}^{atch\times seqlen\times d}$，筛选出属于上下文部分的向量 $h \in {}^{d}$（即 **Segment ID = 1 的 Token**）。
2. **答案跨度预测头（Span Prediction Head）**：
   - **起始位置预测**：
     - 全连接层：$logits_{start} = W_{start} \cdot + {}_{start}$，$W_{start} \in {}^{d}$。
     - Softmax：生成每个 Token 作为答案起始位置的概率 $p_{start} \in {}^{seqlen}$。
   - **结束位置预测**：
     - 全连接层：$logits_{end} = W_{end} \cdot + {}_{end}$，$W_{end} \in {}^{d}$。
     - Softmax：生成每个 Token 作为答案结束位置的概率 $p_{end} \in {}^{seqlen}$。
3. **答案解码**：
   - 选择 `(i, j)` 使得 $p_{start}(i) \times p_{end}()$ 最大，且 `i ≤ j`。
   - 提取上下文中的 Token `i` 到 `j` 作为答案。

5.4 关键差异总结

任务	输入处理	输出头结构	输出目标
文本分类	取 `[CLS]` 标记的输出	单层全连接 + Softmax	类别概率分布
NER	取每个 Token 的输出	全连接 + Softmax/CRF	每个 Token 的标签概率
问答	取上下文 Token 的输出	两个全连接层分别预测起止位置	答案跨度的起止位置概率