NLP 基础

吴建军

wujianjunml@outlook.com

鞍山村出版社

目录

第一章　基于 DL 的 NLP

1.1　深度学习 NLP 技术概述

神经网络语言模型 (Neural Network Language Model, NNLM)。

预训练模型 (PTM,Pre-train Model)。第一代预训练模型专注于 word embedding 的学习 (word2vec)。其特点是 context-free，比如“苹果”这个词在分别表示水果和公司时，对应的 word embedding 是同样的。第二代预训练模型以 context-aware 为核心特征，也就是说“苹果”这个词在分别表示水果和公司时，对应 embedding 是不一样的，其中具有代表性的有 ELMo 等。[16] 是一个 PTM 技术做全面综述。使用 pre-trained 模型有几种方法：

- feature-based: 用预训练模型输出额外的特征，比如 embedding，然后重新使用全新的模型架构并从头训练。
- fine-tuning: 又称微调，可以
  - 在预训练模型的后面接几个简单的 layer，预训练模型的参数作为初始化参数，然后重新训练整个模型参数。
  - 在预训练模型的后面接几个简单的 layer，冻结预训练模型的参数，然后重新训练整个模型参数。

1.2 word2vec

语言模型 (Language Model) 就是：

$$P(w_i|w_{i-n+1}, w_{i-n}, \ldots, w_{i-1})$$

也就是给定前面 $n-1$ 个词后，预测接下来第 n 个词的概率分布。Neural Probabilistic Language Model (NPLM) [17] 提出了用神经实现语言模型。

$$\begin{aligned} \boldsymbol{x} &= concatenate(e_{i-n+1}, e_{i-n}, \ldots, e_{i-1}) \\ \boldsymbol{h} &= tanh(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}) \\ \boldsymbol{y} &= \boldsymbol{W}'\boldsymbol{h} \\ \boldsymbol{p} &= softmax(\boldsymbol{y}) \end{aligned} \tag{1.1}$$

其中，e 是一个 embedding 函数。$\boldsymbol{p}_i$ 表示词 i 的概率，通常选择概率最大的词作为预测时输出。一般用前面两个词去预测下一个词。样本预处理如图1.1。

图 1.1: NPLM 预处理示意图

下面便是他的代码实现。

```
class TrigramNNmodel(nn.Module):
    def ___init___(self, vocab_size, embedding_dim, context_size, h):
        super(TrigramNNmodel, self).___init___()
        self.context_size = context_size  # 上下文长度，这里就是2
        self.embedding_dim = embedding_dim
        # 这是一个带参数的 embedding 层，也就是一开始每个词的 embedding 随机给，随着训练进行，这个值会越来与越恰当
        self.embeddings = nn.Embedding(vocab_size, embedding_dim)
        self.linear1 = nn.Linear(context_size * embedding_dim, h)
        self.linear2 = nn.Linear(h, vocab_size, bias=False) # 输出确实是词典大小

    def forward(self, inputs):
        # 计算输出词的embedding,并拼接起来，每个词都编号了，这里的input是词的编号
        embeds = self.embeddings(inputs).view((-1, self.context_size * self.embedding_dim))
        # 过网络
        out = torch.tanh(self.linear1(embeds))
        out = self.linear2(out)
        log_probs = F.log_softmax(out, dim=1)
        return log_probs

loss_function = nn.NLLLoss() # negative log likelihood loss!!!!!!
model = TrigramNNmodel(len(vocab), EMBEDDING_DIM, CONTEXT_SIZE, H)
```

```
model.cuda(gpu) # load it to gpu!!!!!!
optimizer = optim.Adam(model.parameters(), lr = 2e-3)
for epoch in range(5):
    for it, data_tensor in enumerate(train_loader):
        context_tensor = data_tensor[:,0:2]
        target_tensor = data_tensor[:,2]
        context_tensor, target_tensor = context_tensor.cuda(gpu), target_tensor.cuda(gpu)
        model.zero_grad() # zero out the gradients from the old instance
        log_probs = model(context_tensor) # get log probabilities over next words
        loss = loss_function(log_probs, target_tensor) # compute loss function
        # backward pass and update gradient
        loss.backward()
        optimizer.step()
```

训练完成后，我们可以用 self.embeddings 作为每个词的 embedding。更多细节见这里。

当用当前词 x 预测它的下一个或者上一个词 y，x 用 one-hot 表示，词汇总个数为 V，那么网络可以表示为图1.2。

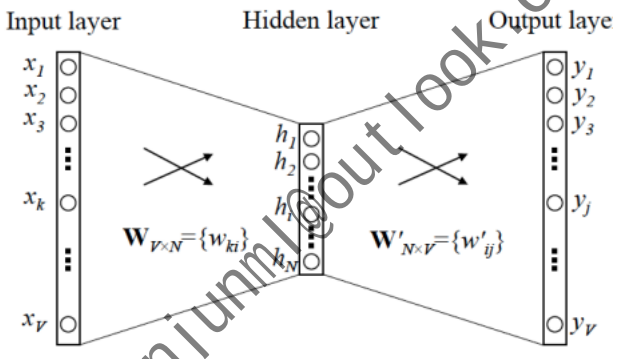

图 1.2: 当前词 x 预测下一个词 y

网络的公式为：

$$\begin{aligned} \boldsymbol{h} &= \boldsymbol{W}\boldsymbol{x} && \text{,其中,} \boldsymbol{x} \in \{0,1\}^V, \boldsymbol{W} \in R^{N\times V} \\ \boldsymbol{y} &= \boldsymbol{W'}\boldsymbol{h} && \text{,其中,} \boldsymbol{y} \in R^V, \boldsymbol{W'} \in R^{V\times N} \\ \boldsymbol{p} &= softmax(\boldsymbol{y}) \end{aligned} \tag{1.2}$$

隐层的激活函数其实是线性的，这也是 Word2vec 的独到之处。且 N 远小于 V。 当模型训练完后，比如现在输入一个 x 的 one-hot 表示，在输入层到隐含层的权重里，只有对应 1 这个位置 (比如位置 k) 的权重被激活，从而用向量 $\boldsymbol{W}_{:,k}$ 来表示 x。

CBOW(Continuous Bag Of Words) 将中间词作为目标词 (前后各 c 个词)，且隐层 $\boldsymbol{h}$ 取

累加值 (也可以是取均值)。

$$
\begin{aligned}
\boldsymbol{e_1} &= \boldsymbol{W}\boldsymbol{x_1} \\
\boldsymbol{e_2} &= \boldsymbol{W}\boldsymbol{x_2} \\
&\vdots \\
\boldsymbol{e_K} &= \boldsymbol{W}\boldsymbol{x_k} \\
\boldsymbol{h} &= \textstyle\sum_{k=1}^{K} \boldsymbol{e_k} \\
\boldsymbol{y} &= \boldsymbol{W'}\boldsymbol{h} \\
\boldsymbol{p} &= softmax(\boldsymbol{y})
\end{aligned}
$$

注意，多个词对应都是从同一个 $\boldsymbol{W}$ 中拿到自己的 embedding 的。

Skip-gram 利用中心词预测上下文，似然函数为:

$$\log p(context_w|w) \quad = \log \textstyle\prod_{u \in context_w} p(u|w)$$

实际操作中，我们会将上下文 $context_w$ 中每个词与 w 分别形成一个训练样本 (图1.3)。

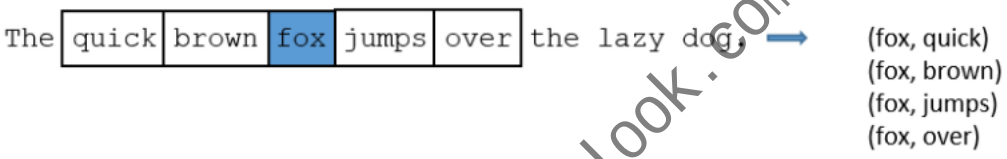

图 1.3: Skip-gram 的样本生成示意图，fox 为中心词

设中心词的 one-hot 表示为 $\boldsymbol{x}$，那么 Skip-gram 的前向传播过程为：

$$
\begin{aligned}
\boldsymbol{e} &= \boldsymbol{W}\boldsymbol{x} \\
\boldsymbol{y} &= \boldsymbol{W'}\boldsymbol{e} \\
\boldsymbol{p} &= softmax(\boldsymbol{y})
\end{aligned}
$$

Skip-gram 的效果比 CBOW 好一点。

接下来我们介绍几个优化技巧：

- Hierarchical softmax。因为语料库往往很大，$|V|$ 很大，所以式1.2中 softmax 计算很慢，因为需要遍历整个词典来计算分母中的归一项：

  $$\frac{\exp(y_j)}{\sum_{i=1}^{|V|} \exp(y_i)}$$

  然后找概率最大的项。为此提出 Hierarchical softmax，我们建立一颗二叉树来替换隐层 ($\boldsymbol{y} = \boldsymbol{W'}\boldsymbol{h}$) 和 softmax 层 ($\boldsymbol{p} = softmax(\boldsymbol{y})$)，二叉树每个叶子节点对应词典中一个词，每个中间节点 c 对应一个 logistic 回归:

  $$\sigma_c(0) = \frac{1}{1 + \exp(-\boldsymbol{\theta}_c^T \boldsymbol{h})}$$

其中 $\boldsymbol{\theta}_c$ 是参数。从每个中间节点往左走的概率为 $\sigma_c(0)$，往右走的概率是 $1-\sigma_c(0)$，计算哪个概率大，就往那边走。如图1.4，如果一个样本的目标词是 *soccer*，那么我们希望有：

$$\sigma_0(0) > 1-\sigma_c(0), \sigma_1(0) < 1-\sigma_1(0), \sigma_3(0) < 1-\sigma_3(0)$$

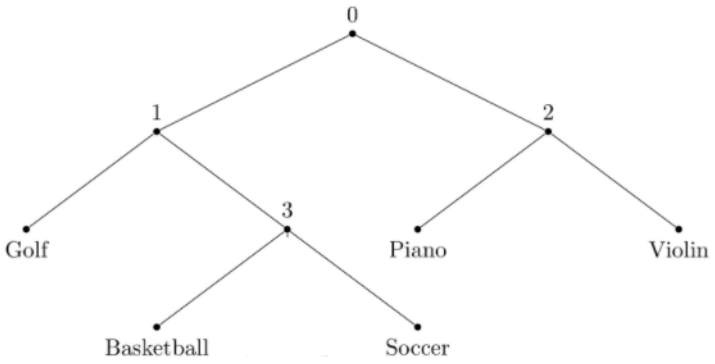

图 1.4: 层次化 softmax 示意图

我们根据词频构建一颗 Huffman 树，一旦构建完毕，CBOW 中对于每个训练样本 $(context_w, w)$ ($context_w$ 表示词 w 的上下文)，我们知道从根节点到词 w 对应节点的路径 p_w。然后我们可以导出似然函数：

$$\log p(w|context_w) = \log \prod_{c\in p_w} \sigma_c(0)^{1-d_c^w}(1-\sigma_c(0))^{d_c^w}$$

其中 d_c^w 表示训练样 $(context_w, w)$ 在节点 c 往左走 $d_c^w = 1$ 还是往右走 $d_c^w = 0$，通过进一步数学推导我们就可以最大化这个似然函数，由此求出每个参数的梯度，从而可以更新其中的参数 (当然，我们此时只能更新路径 p_w 上的参数。而且我们每次只能输入一个训练样本，不能输入一批，也不能用 BP 而是得手动求导然后编写迭代更新代码)。同理对于 Skip-gram 模型：

$$\begin{aligned}\log p(context_w|w) &= \log \prod_{u\in context_w} p(u|w) \\ &= \log \prod_{u\in context_w} \prod_{c\in p_u} \sigma_c(0)^{1-d_c^w}(1-\sigma_c(0))^{d_c^w} \\ &= \sum_{u\in context_w} \sum_{c\in p_u} (1-d_c^w)\log\sigma_c(0) + d_c^w \log(1-\sigma_c(0))\end{aligned}$$

p_u 表示从根节点到词 u 对应节点的路径。

- negative sampling。遇到生僻词候，在 Huffman 树中也要走很久，计算量也很大。在 CBOW 中，正确词只有一个，其他都是错误的。比如句子：” 我/永远/爱/中国/共产党”，中心词为’ 爱’，我们在选择噪声词的时候，选择了 K 个，但是实际上，在词汇表中，排除掉’ 我’，’ 永远’，’ 中国’，’ 共产党’ 这四个词汇的其他词都可以算做’ 爱’ 的噪声词，然而为了减少复杂度，我们只选择了其中的 K 个。对于 (input_word,postive_output_word) 我们最大化这个样本的似然函数，对于 (input_word,negtive_output_word) 的我们则

是最小化这个样本的似然函数。我们可以分别求得最大化似然和最小化似然的梯度更新公式，然后对于每个样本 $(context_w, w)$ 如果 $context_w$ 是正样本采用最大化似然导出的梯度更新公式，如果是负样本采用最小化似然导出的梯度更新公式。当然参数二者是一样的，只是更新方法不同。

- subsampling。用高频词去预测目标词时往往是没有意义的，即训练样本 (input_word, output_word) 中 input_word 是高频词，比如：(" 我"− >" 永远")，("the"− >"fox") 这些训练样本并不会改进" 永远" 或者"fox" 两个词的 embedding。我们以一定的概率删除这样的样本 (也就是不喂入网络)，删除概率与 input_word 的频率成正比。对高频词做降采样可以防止最后算出来语义接近的都是热门词。

得到每个词的 embedding 后，我们可以：

- 计算两个词之间的相似度。
- 对于短文本分类，可以直接把文档里面所有的 word 的 embedding 线性相加，作为文本的特征去训练分类器。
- word2vec 提供一种思路：可以根据 item 之间的邻近-非邻近 (共现-非共现) 关系来学习 item 的 embedding。这个思想可以运用到更多场景中，比如可以根据关系网络上节点之间的邻居-非邻居关系形成正负样本喂入 word2vec 来学习每个顶点的 embedding，另外把用户 APP 下载/使用的邻居-非邻居关系形成正负样本喂入 word2vec 来学习每个 app 的 embedding。

word2vec 的缺点有：

- 生词不能输出其 embedding。
- 一词多义搞不定。

另外 word2vec 的论文不止一篇 ([18] 提出 word2vec 的框架，[19] 提出 hierarchical softmax 和 negative sampling 两个训练技巧)，而且写得不是很清晰，然后很多博客各种含混的解读，导致理解起来充满困难，加上目前 (2020 年) 已经有更好的算法，可以不必多么细致地去了解它。GloVe release 出来的 pre-trained embedding 比 word2vec 的在细节处理更好一些，而且前者使用的语料库也更大，所以开源的 Glove 向量效果是好于 word2vec 向量的。

skip thought vectors [20] 是 word2vec skip-gram 在句子上的推广，它把 skip-gram 里每个词换成一个句子，每个句子都用 RNN 来编码。不过注意，使用 word2vec 得到每个词的 embedding，然后取平均作为句子的 embedding，得到的效果也不差，甚至更好。

1.3 seq2seq

[21] 和 [22] 同时提出了 Seq2Seq 架构，他们最重要的贡献在于通过采用诸如 LSTM/GRU 的 RNN 首次实现了将可变长度的输入序列映射为一个可变长度的输出序列。注意，[22] 还首次提出了 GRU(Gated Recurrent Unit) 这一重要的 RNN cell。在 Seq2Seq 架构中，Encoder 把输入序列编码成一个固定长度的向量，这个向量传给 Decoder，Decoder 再根据这个向量输出可变长度的输出序列。Seq2Seq 架构被用在了很多领域，如机器翻译 (文本-> 文本) [21]，QA 模型 (文本-> 文本) [23]，语音识别 (音频-> 文本) [24]，image caption(图片-> 文本) [25]，video caption(视频-> 文本) [26]。图1.5展示了 seq2seq 例子 (图源于 [27] 的第 10 章)：

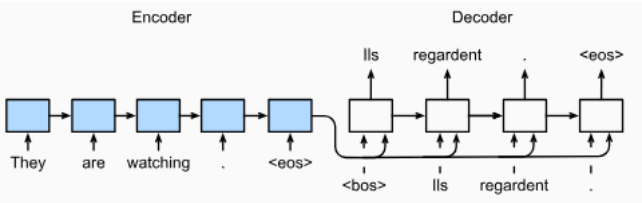

图 1.5: seq2seq 用于英语到法语的翻译示例

令输入序列是 $S = [s^1, s^2, \ldots, s^N]$，目标输出序列是 $O = [o^1, o^2, \ldots, o^M]$，每个 s^t 或 o^t 是输入或输出的第 t 个 word/token 的 one-hot 向量。机器翻译模型训练时目标函数为：

$$\frac{1}{|O|}\sum_{(S,O)\in\mathcal{O}} \log P(O|S)$$

在 [22] 中，Encoder 的计算过程为：经过 embedding 层每个输入 word/token 对应的 s^t 被转成向量 x^t，接着依次把每个 x^t 输入到 RNN 中 (具体的 RNN cell 可以是 LSTM 或者 GRU)，得到第 t 步的输出为：

$$h^t = GRU(x^t, h^{t-1}), \quad h^0\text{为全 } 0$$

等整个序列依次处理完毕 (输入序列会有个特殊的结束符号 eos(end-of-sequence))，我们可以得到 h^N，再进行如下运算：

$$c = tanh(Vh^N)$$

向量 c 就是 encoder 的输出。Decoder 也是一个 RNN，每个 o^t 也是先经过 embedding 被转成向量 y^t(输出序列有个特殊的开始符号 bos(beginning-of-sequence))，Decoder 的第 t 步的输出为：

$$\bar{h}^t = GRU([y^{t-1}\ c], \bar{h}^{t-1}), \bar{h}^0 = tanh(\bar{V}c), \quad y^0\text{为全 } 0$$

注意，这里把 y^{t-1} 和 c 两个向量 concatenate起来了。除了输出 $\bar{h}^t$，第 t 步还会输出下面的概率分布 (也就是第 t 步输出字典中第 j 个的词的概率，假设词典大小为 K)：

$$p(\bar{y}_j^t = 1) = \frac{exp(g_j s^t)}{\sum_{\bar{j}=1}^{K} exp(g_{\bar{j}} s^t)}$$

其中 g_j 是需要学习的权重向量，并且 $s_i^t = \max\{\bar{s}_{2i-1}^t, \bar{s}_{2i}^t\}$, $\bar{s}^t = O_h \bar{h}^t + O_y o^{t-1} + O_c c$ 。Decoder RNN 在某个时刻预测应该输出 eos 则结束。训练完成后，在预测推理时寻找使得下面取值最大的 O 作为输出序列：

$$\arg\max_O P(O|S)$$

此时，Encoder 计算过程不变，但是 Decoder 计算时需要 o^t，就是词典中第 $\hat{j}$ 个 word/token：

$$\hat{j} = \arg\max_j p(\bar{y}_j^{t-1} = 1)$$

$t = 0$ 时 $o^t = bos$，其实就是前一步输出概率最大的 token。

Teacher Forcing 是 RNN 架构在训练中常用的一个技巧，[28] 对此有个易读的介绍。假

图 1.6: image captioning 示例

设，我们要训练一个 image captioning 模型。图1.6的 ground truth 是“Two people reading a book”。训练中某个时刻，我们的模型预测错了第二个 token，其输出的第一个和第二个 token 分别是”Tow birds”. 如果不用 Teacher Forcing, 我们会直接把“birds”作为第三个时间步的输入喂给 RNN，如果用 Teacher Forcing，我们则把“people”这个正确的 token 喂给 RNN, 同时我们会记录”birds” 和”people”之间的 loss。图1.7展示了这个过程。Teacher Forcing 可以

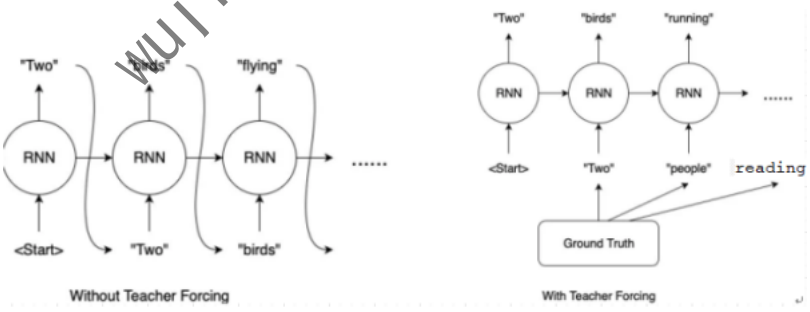

图 1.7: Teacher Forcing 示例

加速训练的收敛，特别是训练初始阶段，模型的预测精度很低，不使用 Teacher Forcing，则模型一直在错误的道路上不断前进。但是在推理阶段，没有 Teacher 校正，只能把前一步的输入作为下一步的输入，这个训练和推理之间的差异被称为 Exposure Bias。

[21] 使用了多层 LSTM。单层 LSTM 如下：

$$h^t, c^t = LSTM(x^t, h^{t-1}, c^{t-1}), t = 1, 2, \ldots, T$$

多层 LSTM 中, the input of the l-th layer ($l \geq 2$) is the hidden state h_{l-1}^t of the previous layer multiplied by dropout δ_{l-1}^t，如下：

$$
\begin{aligned}
&h_1^t, c_1^t = LSTM_1(x^t, h_1^{t-1}, c_1^{t-1}), t = 1, 2, \ldots, T \\
&h_2^t, c_2^t = LSTM_2(dropout(h_1^t), h_2^{t-1}, c_2^{t-1}), t = 1, 2, \ldots, T \\
&\ldots \\
&h_l^t, c_l^t = LSTM_l(dropout(h_{l-1}^t), h_l^{t-1}, c_l^{t-1}), t = 1, 2, \ldots, T
\end{aligned}
$$

[21] 还有个技巧: 将源 (source) 句子顺序颠倒后再输入 Encoder 中，比如源句子为"A B C"，那么输入 Encoder 的顺序为"C B A"，经过这样的处理后，取得了很大的提升，而且这样的处理使得模型能够很好地处理长句子。但是，注意 target 句子依然是按原始顺序输入。如果在 RNN 中每一步都选择概率最大的词，最后得到的序列不见得是概率最大的。[21] 使用 beam search 来生成输出序列。也就是第一个时间步长，选取当前条件概率最大的 k 个词 (设 $w_1^{(1)}, w_2^{(1)}, \ldots, w_k^{(1)}$)，之后的每个时间步 t，基于上个步长的输出序列，挑选出所有组合中条件概率最大的的 k 个，也就是使得下式最大的 k 个词 v：

$$p(v|w_i^{(t-1)}), i = 1, 2, \ldots, k$$

注意，完全可能取定某些 $w_i^{(t-1)}$ 后选择任何一个词得到的概率都不是 top k。最后一步从 k 个候选中挑出概率最大的。

怎么评价预测的 sequence 与目标 sequence 之间的差异呢？BLEU(Bilingual Evaluation Understudy) 这个指标被大量采用。令 $O = [o_1, o_2, \ldots, o_M]$ 是 target sequence，len_o 是 O 的 token 数，$P = [p_1, p_2, \ldots, p_N]$ 是 target sequence，len_p 是 P 的 token 数。设 P 中的 $n-gram$ 总共有 T_n 个，P 的 T_n 个 $n-gram$ 中有 R_n 个在 O 中出现，k 表示 P 的最长 gram 的长度 (注意，符号不算 token)，则

$$p_n = \frac{R_n}{T_n}, n = 1, 2, \ldots, k$$

$$\Omega = \prod_{n=1}^{k} p_n^{\frac{1}{2^n}}$$

$$BLEU = \exp(\min(0, 1 - \frac{len_o}{len_p}))\Omega$$

1.4 Attention 与 Transformer

seq2seq 将整个输入序列的信息编码成一个固定大小的状态向量 c，这样做有几个问题：

- 无论输入序列长度如何，向量 c 长度固定。可以想象，输入序列长度越长，向量 c 长度固定导致的问题会越严重。
- 不同位置的 word 对向量 c 贡献都是一样的，而实际中一句话中不同 word 的重要性是不一样的。

[29] 首次提出 Attention。其中，每个 target token 都对应一个 context vector c_t，然后 Decoder 在每个时间步的计算方式发生了变化，比如对 GRU 变化如下：

$$\bar{h}^t = GRU([y^{t-1}\ c], \bar{h}^{t-1}) \Rightarrow \bar{h}^t = GRU([y^{t-1}\ c^t], \bar{h}^{t-1})$$

注意，不再只有一个 c 向量了，而是有多个。每个 c^i 计算方式如下：

$$\begin{aligned} c^i &= \sum_{j=1}^{N} \alpha^{ij} h^j \\ \alpha^{ij} &= \frac{\exp(e^{ij})}{\sum_{j=1}^{N} \exp(e^{ij})} \\ e^{ij} &= f(\bar{h}^{i-1}, h^j) \end{aligned} \tag{1.3}$$

其中，h^j 是 encoder 中 RNN cell 的 hidden state，可以认为 h^j 是输入序列的一个表示，但是格外关注第 j 个 token。e^{ij} 表示输入序列第 j 个 token 和输出序列第 i 和 token 之间的关联度，$\bar{h}_i$ 表示 decoder 中第 i 个时间步的 hidden state。这里的 f 函数被称为 alignment mode，是一个简单的 MLP，比如：

$$f(\bar{h}^{i-1}, h^j) = v^T \tanh(W_1 \bar{h}^{i-1} + W_2 h^j) = v^T \tanh(W[\bar{h}^{i-1}\ h^j])$$

其中，v, W_1, W_2, W 是参数。总结一下，模型自动学习每个输入 token 和每个输出 token 之间的关联度，然后用这个关联度对输入加权组合得到多个 context vector，每个 context vectorc^i 蕴含着预测第 i 个输出时整个输入应该呈现的信息。e^{ij} 正是预测第 i 个输出时第 j 个输入的重要性得分，预测第 i 个输出时，每个输入的信息都有考虑，但是因为重要性不一样所以考虑程度也不一样。为了让每个 h^j 不但包含之前 token 的信息，也包括之后的 token 的信息，所以 [29] 使用了双向 RNN。先用一个 RNN 从前往后读输入 sequence，得到每个 $\overrightarrow{h}_j$，再用另外一个 RNN 从后往前读输入 sequence 得到每个 $\overleftarrow{h}_j$，最后 concatenate 二者得到 $h_j = [\overrightarrow{h}_j \quad \overleftarrow{h}_j]$。

f 是有好几种不同的实现，比如：

$$f(\bar{h}^{i-1}, h^j) = \begin{cases} (\bar{h}^{i-1})^T h^j & \text{dot} \\ (\bar{h}^{i-1})^T W_a h^j & \text{general} \\ v_a^T \tanh(W_a[\bar{h}^{i-1}\ h^j]) & \text{concat} \end{cases}$$

这三种实现被 [30] 称为 Global Attention，因为计算重要性得分时考虑了输入序列全部位置，当输入很长时 (比如文章摘要提取) 这种方法计算代价就很大，于是 [30] 提出 Local Attention。Local Attention 首先要计算输出第 t 个 token 时需要与输入序列的哪个位置对齐，设这个位置是 p_t，找到对齐位置后往前和往后各选择 D 个输入 token 组成的 window，然后计算重要性得分时只考虑这个 window 内的输入 hidden state。最简单的方法是 $p_t = t$，另外可以先按如下计算 p_t：

$$p_t = N\sigma(v_p^T \tanh(W_p h^t))$$

接着以位置 p_t 为中心向两边递减重要性得分：

$$\bar{a}^{ts} = a^{ts} \exp(-\frac{(s-p_t)^2}{D^2/2})$$

其中，N 是输入序列的长度，D 是 window 的宽度 (超参数)。到此，我们可以把 Attention 的作用概括为：将输入序列的每个位置和输出序列的每个位置进行关联，得到一个重要性得分。每次输出时，以根据重要性得分加权每个输入位置上的信息为基础，更好地给到输出。

我们现在换个方式看待 attention(参考 [27] 的第 11 章)。设有 m 个 key-value pair $D = \{(k_i, v_i), i = 1, 2, \ldots, m\}$，给定一个 query q，attention 可以概括为：

$$attention(q, D) = \sum_{i=1}^{m} a(q, k_i) v_i$$

a 表示 attention 权重，它常常会通过 softmax 来做归一化：

$$a(q, k_i) = \frac{\exp(a(q, k_i))}{\sum_j \exp(a(q, k_j))}$$

其中 $a(q, k_i)$ 表示 attention scoring function。可以看出当 $q = \bar{h}^{i-1}, k_i = v_i = h^i$ 时，式1.3中的 attention 与这个形式完全等价。 [31] 提出了 scaled dot product attention：假设 q 和 k_i 都是 d 维向量，为了保持点积前后方差不变，我们采用如下 attention scoring function：

$$a(q, k_i) = q^T k_i / \sqrt{d}$$

当有多个 query 需要计算 attention 时，

$$\begin{aligned}
&softmax(\tfrac{QK^T}{\sqrt{d}})V = \\
&= softmax\left(\begin{bmatrix} q_1 \\ q_2 \\ \vdots \\ q_n \end{bmatrix} \begin{bmatrix} k_1^T & k_2^T & \ldots & k_m \end{bmatrix} \tfrac{1}{\sqrt{d}}\right) \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_m \end{bmatrix} \\
&= softmax\left(\begin{bmatrix} q_1 k_1^T/\sqrt{d} & q_1 k_2^T \sqrt{d} & \ldots & q_1 k_m^T \sqrt{d} \\ q_2 k_1^T \sqrt{d} & q_2 k_2^T \sqrt{d} & \ldots & q_2 k_m^T \sqrt{d} \\ \vdots & \vdots & \ddots & \vdots \\ q_n k_1^T \sqrt{d} & q_n k_2^T \sqrt{d} & \ldots & q_n k_m^T \sqrt{d} \end{bmatrix}\right) \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_m \end{bmatrix}
\end{aligned}$$

下面的称为 Additive Attention：

$$a(q, k_i) = w_v^T \tanh(W_q q + W_k k_i)$$

additive attention 和 dot-producte attention 二者理论上是相似的，但是后者计算更快，因为矩阵乘法有高度优化的代码来执行。

Multi-Head Attention [31] 首先对 query,key,value 做 H(多个 head) 个变换：

$$\begin{aligned}
\hat{q}_h &= W_h^q q \\
\hat{k}_{ih} &= W_h^k k_i \\
\hat{v}_{ih} &= W_h^v v_i \\
h &= 1, 2, \ldots, H
\end{aligned}$$

这里 H 个矩阵 W_j^q, W_j^k, W_j^v 都是需要学习的参数，然后再执行下面的运算:

$$\begin{aligned}
&\hat{a}_{ih} = a(\hat{q}_h, \hat{k}_{ih}) && \text{1. 计算重要性权重} \\
&\hat{a}_{ih} = softmax(\hat{a}_{ih}), i = 1, 2, \ldots && \text{2. softmax 归一化} \\
&\hat{c}^h = \textstyle\sum_i \hat{a}_{ih} v_i && \text{3. 更新上下文变量} \\
&\hat{c} = W \begin{bmatrix} \hat{c}^1 \\ \hat{c}^2 \\ \vdots \\ \hat{c}^H \end{bmatrix} && \text{4. 拼接多个 head 做线性变换}
\end{aligned} \tag{1.4}$$

最后一步的 $\hat{c}$ 便是 Multi-Head Attention 的输出。用 scaled dot product attention 的话，那么整个过程可以表示为：

$$\begin{aligned}
&Q_h = W_h^q Q^T, K_h = W_h^k K^T, V_h = W_h^v V^T && \text{1. 变换 } q, k, v \\
&head_h = softmax(\tfrac{Q_h K_h^T}{\sqrt{d_h}}) V_h && \text{2. scaled dot product} \\
&\hat{V} = W_o \begin{bmatrix} head_1^T \\ head_2^T \\ \vdots \\ head_H^T \end{bmatrix} && \text{3. 多头拼接}
\end{aligned} \tag{1.5}$$

注意，这里最后一步是将多个 $head_h$ 向量先转置然后在列方向上 concatenate。设输入一个 $x_i, i = 1, 2, \ldots$ 序列，当 query=key=value=x_i 时则被称为 self-attention，此时 scaled dot product attention 变成

$$softmax(\frac{QK^T}{\sqrt{d}})V \Rightarrow softmax(\frac{XX^T}{\sqrt{d}})X$$

。在 encoder-decoder 架构中使用之前的 attention 时，source 和 target 是不同的，而使用 self attention 时两个是一样的，即 target=source。我们可以看到 scaled dot product attention 没

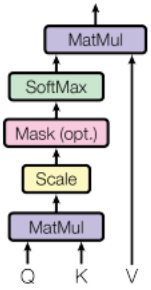

图 1.8: Scaled Dot-Product Attention

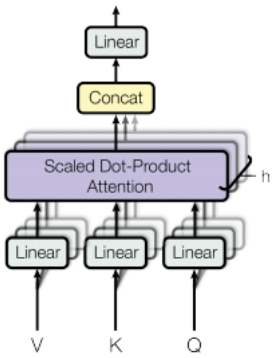

图 1.9: Multi-Head Attention

有可以学习的参数，所以需要 Multi-Head Attention，特别是使用 self-attention 来提取输入序列不同特征时，scaled dot product attention 相当于单次卷积，Multi-Head Attention 相当于多个独立的卷积核。同时使用三者的话，我们仍然称这个新的 layer 为 Multi-Head Attention，它操作为：

$$\hat{X} = MHA(X)$$

展开则为：

$$\begin{aligned} Q_h &= W_h^q X^T, \quad K_h = W_h^k X^T, \quad V_h = W_h^v X^T \\ \hat{V}_h &= softmax(\frac{Q_h K_h^T}{\sqrt{d_h}}) V_h \\ \hat{X} &= W_o \begin{bmatrix} \hat{V}_1 \\ \vdots \\ \hat{V}_H \end{bmatrix} \end{aligned} \tag{1.6}$$

Multi-Head Attention 的架构示意在图1.9中。可以看到这样的 Multi-Head Attention 完全脱离了 RNN，不存在一个一个地做序列计算，进而可以并行，尽管此时 X 是一个 sequence(每个 token 的 embedding 向量组成的矩阵)，而 $\hat{X}$ 就是输入句子一个新的 embedding 表示了。

设 n 表示最大长度，d_{model} 是 embedding size，则 $X \in R^{n \times d_{model}}$, 我们设

$$W_h^q \in R^{d_h \times d_{model}}, W_h^k \in R^{d_h \times d_{model}}, W_h^v \in R^{d_h \times d_{model}}$$

那么我们可以得到

$$Q_h \in R^{d_h \times n}, K_h \in R^{d_h \times n}, V_h \in R^{d_h \times n}$$

也就是 Q_h, Q_h, V_h 的每一列分别是一个 token 的 q, k, v 向量。从而，$\hat{V}_h \in R^{d_h \times n}$。设 $W_o \in R^{d_{model} \times H d_h}$，最后 $\hat{X} \in R^{d_{model} \times n}$。可以看到，几个参数矩阵都跟长度 n 无关，也就是无论序列有多少 token 都可以运算。

目前这个 Multi-Head Attention 的输出丢掉了输入序列的位置信息，而在序列学习中位置信息往往是重要的，所以我们要做 Positional Encoding。对于一个输入 token 序列的 embedding 矩阵 $X \in R^{n \times d}$，对应的 positional embedding 矩阵 $P \in R^{n \times d}$ 为：

$$\begin{aligned} p_{i,2k} &= \sin(w_k * i) \\ p_{i,2k+1} &= \cos(w_k * i) \end{aligned}$$

其中 $w_k = \frac{1}{10000^{2k/d}}$。positional embedding 矩阵 P 的每一行为：

$$p_{i,:} = \left[\sin(w_0 * i) \ \cos(w_0 * i) \ \sin(w_1 * i) \ \cos(w_1 * i) \ \ldots \ \sin(w_{d/2} * i) \ \cos(w_{d/2} * i) \right] \quad (1.7)$$

k 越大，对应的三角函数波的频率越小 (频率为 w_k)。每一个偶数列和奇数列分别如下：

$$p_{:,2k} = \begin{bmatrix} \sin(w_k * 0) \\ \sin(w_k * 1) \\ \vdots \\ \sin(w_k * n) \end{bmatrix} \quad p_{:,2k+1} = \begin{bmatrix} \cos(w_k * 0) \\ \cos(w_k * 1) \\ \vdots \\ \cos(w_k * n) \end{bmatrix},$$

i 越大，token 的位置越靠后，对应三角函数波的频率越低。图1.10 [32] 展示了对一个最大长度为 50 的句子做维度为 128 的 embedding 时，对应的 positional embedding：Positional

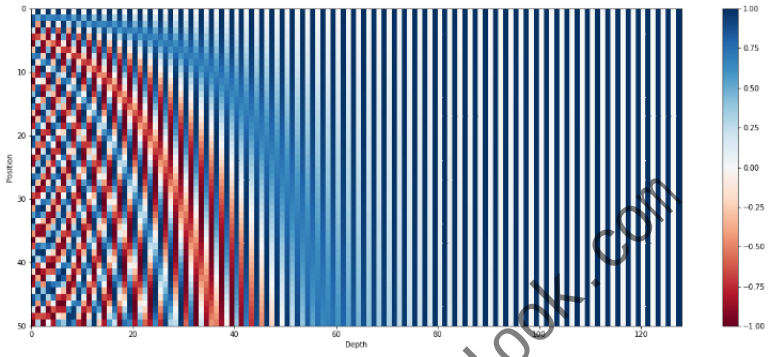

图 1.10: transformer 架构

Encoding 的输出便是 $X + P$，这里我们将位置信息 inject 到 X 中了。从图1.10中可以看到，其实只有前面几列存储了位置信息 (不同行的取值不一样)，可以猜想 token 的 embedding 应该是被训练成前面几个维度取值很小或者没有语义信息，所以 $X + P$ 的前面几个维度一直是位置信息。注意，无论 i 是多大，$p_{i,:}$ 都可以算，也就是输入序列长度无论多长，每个 token 都能计算一个位置编码。

[31] 首次提出 transformer 架构。不像 CV 领域 (其核心网络架构 CNN 是 80 年代被发明了) 和之前 NLP 领域 (核心架构 LSTM 也是 90 年代被发明了)，基于 Transformer 的网络模型和预训练模型是当前 NLP 最重要基石，是这一轮 deep learning 在网络结构上的重大创新，基于 Transformer 的网络模型也在 CV 领域得到广泛应用。transformer 架构完全抛弃了 RNN，也不采用 CNN。

Transformer 的整体架构如图1.11。Transformer 也是 encoder-decoder 架构。encoder 是 N 个结构一样的 layer 的堆叠，每个 layer 的输入是前一个 layer 的输出，第一个 layer 的输入是 $X + P$，也就是 source sequence 的 word embedding + positional encoding。encoder 的每个 layer 中有两个 sublayer。第一个 sublayer 是 Multi-Head Attention，图中的三个分叉箭头正是 self attention 的体现，也就是 Q, K, V 都是 $X + P$。另外一个 sublayer 是 position-wise FFN，其操作为：

$$FFN(x) = relu(xW_1 + b_1)W_2 + b_2$$

。$W_1 \in R^{d_{model} \times d_{ff}}, W_2 \in R^{d_{ff} \times d_{model}}$。注意，这两个参数矩阵也跟输入长度无关。 另外 Add & norm 执行的运算为：

$$LayerNorm(X + SubLayer(X))$$

可以看到，这里使用了 Layer Normalization 和 residual connection。[32] 认为，之所以采用 residual connection 是因为 Multi-Head Attention 处理后会丢掉位置信息，所以需要重新 inject。decoder 也是多个结构一样的 layer 的堆叠。decoder 的每个 layer 则有 3 个 sublayer

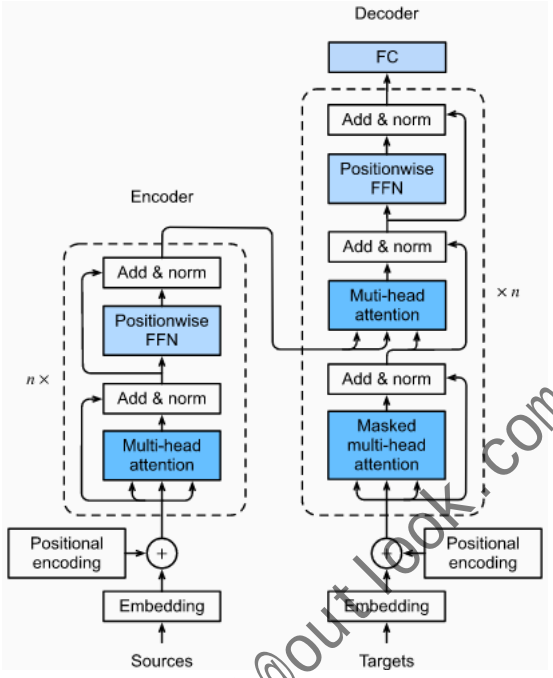

图 1.11: transformer 架构

第一个 sublayer 是 Masked Multi-Head Attention。其中 Masked 是指

$$softmax(\frac{QK^T}{\sqrt{d}})V \Rightarrow softmax(mask(\frac{QK^T}{\sqrt{d}}))V \tag{1.8}$$

，mask 将矩阵 $\frac{QK^T}{\sqrt{d}}$ 的上三角元素置为负无穷，之所以这样做是因为 Transformer 推理时是一个一个词预测，预测第 i 个 token 时只能根据前面的 $i-1$ 个 token 的信息来预测，无法使用 $i+1$ 及其以后 token 的信息，但是我们训练时却是一次性把 target 的所有 token 给到了，所以这里必须 mask，否则 attention 会看到 $i+1$ 及其以后的 token 的信息从而导致训练和推理使用到的信息不一样。 训练时，Masked Multi-Head Attention 的输入是 $Y+P$，也就是 target sequence 的 word embedding + positional encoding。第一个 sublayer 的输出是第二个 Multi-Head Attention 的 Q, 第二个 Multi-Head Attention 的 V 和 K 则是 encoder 的输出。当然啦，只是 decoder 的第一层 layer 是这样的，decoder 第二层 layer 的输入则是 decoder 前一层 layer 的输出。 第三个 sublayer 是一个 position-wise FFN。Decoder 的输出最后喂入一个 FC 层，这个 FC 层是 linear + softmax，从而我们可以得到概率最大的一个token。

[33] 基于 pytorch 实现了一个有 attention 机制的 seq2seq 模型。[34] 详细展示了如何使用 pytorch 实现完整的 Transformer。

Transformer 在训练时仍然采用了 Teacher Forcing，训练过程的伪代码如下：

```
def train_model(model, src, target):
   # 输入序列首先经过encoder得到memory
   memory = model.encode(src)
   predict = torch.zeros(1, 1).type_as(src)
   # 逐个token预测
   for i in range(target.size()[0]):
      #当前的target经过decoder
      out = model.decode(memory, predict, get_mask(predict.size(1)))
      # 经过fc层得到一个预测token
      _, next_word = torch.max(model.fc(out[:, -1]), dim=1)
      next_word = next_word.data[0]
      # 计算loss
      loss = loss(next_word, target[i])
      # 更新权重
      loss.backward()
      # 抛弃错误的预测token,把正确的token加入
      predict = torch.cat([predict, torch.empty(1, 1).fill_(target[i])], dim=1)
```

，预测过程的伪代码如下，注意，可以看到，是一个一个 token 的输出：

```
def predict_model(model, src, max_token_size, stop):
   # 输入序列首先经过encoder得到memory
   memory = model.encode(src)
   predict = torch.zeros(1, 1).type_as(src) # predict序列初始为空(bos)
   # 不断调用decoder逐个token预测
   for i in range(max_token_size):
      # 当前predict序列过decoder，注意计算掩码矩阵
      out = model.decode(memory, predict, get_mask(predict.size(1)))
      # 经过fc层得到一个预测token
      _, next_word = torch.max(model.fc(out[:, -1]), dim=1)
      next_word = next_word.data[0]
      # 判断是否提前结束
      if next_word == stop:
         break
      # 把刚刚预测的token追加到predict序列中，再次走decoder
      predict = torch.cat([predict, torch.empty(1, 1).fill_(next_word)], dim=1)
   # 返回整个预测序列
   return predict
```

有很多工作着力于 Transformer 计算复杂度高进而导致长序列建模低效的问题，如下：

- Transformer-XL [35]：一般用 Transformer 时输入序列的最大长度是固定的。粗略说来，为建模更长的序列，Transformer-XL 把输入序列先分段，记录每个段经过每层 Encoder 后的输出，第 τ 个段在第 $n-1$ 层 encoder 的输出为 h_{τ}^{n-1}，接着 $\bar{h}_{\tau+1}^{n-1}=[h_{\tau}^{n-1}\quad h_{\tau+1}^{n-1}]$，然后 $\bar{h}_{\tau+1}^{n-1}$ 被喂入第 n 层 Encoder 得到为 $h_{\tau+1}^{n}$。注意，这里 $\bar{h}_{\tau+1}^{n-1}$ 的长度总是 $h_{:}^{n-1}$ 的两倍。

- Longformer [36]：Transformer 让每个 token 与其他所有 token 做 Attention。Longformer 让每个 token 只和附近的 k 个 token 做 Attention(window attention)，也就是 w_i 和下面的 token 做 attention

$$(w_{i-k},\ldots,w_{i-2},w_{i-1},w_{i+1},w_{i+2},\ldots,w_{i+k})$$

  。Longformer 还提出做膨胀 (dilated)Attention，比如 w_i 和下面的 token 做 attention

$$(w_{i-k},\ldots,w_{i-3},w_{i-1},w_{i+1},w_{i+3},\ldots,w_{i+k})$$

- Reformer [37]：在 Attention 的计算中，其实每个 query 不需要与每个 key 都做计算，只需要与最相似 (Attention score 最大) 的一些 key 计算，Reformer 利用 Locality sensitive hashing 快速找到最相似的一批 key 做 Attention 计算。具体说来，假设我们已经有一个 LSH 函数 $hash$，对于位置 i 的 query(token)，计算 $hash(key_j), j=1,2,\ldots,n$，找到与 $hash(query_i)$ 落在同一个 hash bucket 中的 key，只与这些 key 计算 attention。Reformer 还使用了 Reversible Residual Network 和 Reversible Transformer 的技术从而大幅减小显占用。另外 Reformer 在做 Multi Head Attention 时，$Q=K$，也就是 W^q 和 W^k 是一样的。最后 Reformer 显存占用大大减少，计算速度大大加快，同时效果持平。

- BigBird [38]：BigBird 可以对长达 4096 的序列建模。BigBird 中，某些位置的词 (全局预定义的一些 token) 做 global attention，某些词 (也是全局预定义的一些 token) 与随机选择几个位置做 attention，剩下的词做 window attention。BigBird 最关键的贡献是高性能地实现了这三种 attention 的计算，可以参考 [39] 来帮助理解。

- Linformer [40]

ExT5 [41] 是一个在有监督的 EXMIX 数据集 (107 个有监督的 NLP 数据集) 和自监督的 C4 数据集上进行预训练的模型，证明了 multi-task learning 可以极大地提升 LLM。

1.5 BERT

BERT(Bidirectional Encoder Representations from Transformers) [42] 是一个里程碑的工作。他使用 WordPiece 将一个序列分成一个个 token，其输入有几个特点，见图1.12 [42]：

- 输入是一个 token 序列。可以是一个句子，也可以是一个句子对 (A,B)。输入总是用 [CLS] 这个特殊 token 开始。输入句子对时，两个句子之间用 [SEP] 隔开。[UNK] 表示一个不在词表中的 token，输入的序列长度固定为 512，长度不够时用 [PAD] 填充。
- 输入会做三个 embedding(每个都是 768 维)，然后 3 个 embedding 向量相加。不同于 Transformer，Position Embeddings 是训练得到的。Segment Embeddings 表示每个 token 是属于 A 句 (对应 0) 还是 B 句 (对应 1)，当只有一句话时全为 0，然后从 2*768 的参数矩阵 look up 得到向量，两句话时则两个向量相加，所以得到的仍然是 768 的向量 [43]。

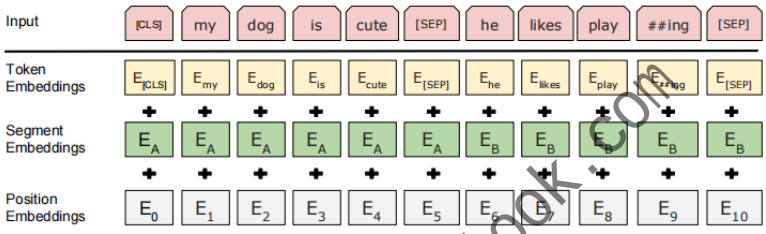

图 1.12: BERT 输入

BERT 只使用了 Transformer 的 Encoder 部分，它把许多 Encoder 堆叠起来，见图1.13 [44]。输出则是输入各 token 融合全文语义后的向量表示，注意输入和输出的 token 个数都是 512。

BERT 使用了 masked language model(MLM) 和 Next Sentence Prediction (NSP) 两大类任务来训练模型。所谓的 Bidirectional 具体指的就是 MLM 这样训练方法 (当然需要模型架构支持)，因为这种训练方法模型可以同时看到一个 token 的左边和右边。MLM randomly masks some of the tokens from the input, and the objective is to predict the original vocabulary id of the masked word based only on its context. MLM 其实就是完形填空，图1.14 [44] 是 MLM 的一个例子。再比如, in the sentence "I accessed the bank account," a unidirectional model would represent "bank" based on "I accessed the" but not "account." However, BERT represents "bank" using both its previous and next context — "I accessed the ... account" —starting from the very bottom of a deep neural network, making it deeply bidirectional [44]。Whole Word Masking(WWM) 是 masked LM 的改进。原有基于 WordPiece 的分词方式会把一个完整的词切分成若干个子词，在生成训练样本时，这些被分开的子词会随机被 mask。在 WWM 中，如果一个完整的词的部分 WordPiece 子词被 mask，则同属该词的其他部分也会被 mask。WWM 示例见图1.15 [45]。注意，masked LM 有两个问题：掩码预测不适用序列到序列的文本生成任务，并且掩码预测难以直接扩展到多语语料中。[46] 使用复述/释义 (paraphrasing)

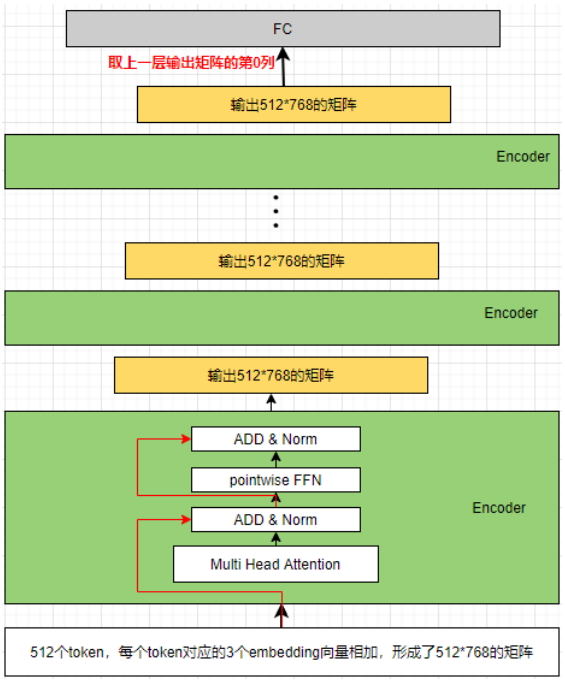

图 1.13: BERT 架构

Input: The man went to the [MASK]$_1$. He bought a [MASK]$_2$ of milk .
Labels: [MASK]$_1$ = store; [MASK]$_2$ = gallon

图 1.14: MLM 示例

来训练模型，就是一句话用另外一句话表达出相同的意思。Next Sentence Prediction (NSP) 是指给出两句话 (A,B)，预测 B 是否 A 的下一句，例子如图1.16。因为 BERT 缺乏 decoder，他无法实现 seq2seq，无法用问答数据训练，所以不适合做文本生成类任务。

[47] 发布了两个模型：$BERT_{base}$ 和 $BERT_{large}$。前者参数量 110M，包括 12 层 encoder；后者参数量 340M，包括 24 层 encoder。注意，参数中有 23M 是 token embedding(30K 个 token*768)。[47] 还发布了 bert-base-chinese(110M) 来支持中文。[45]([48]) 发布了多个基于 wwm 的 BERT 中文模型，如 BERT-wwm-ext、RoBERTa-wwm-ext-large 等。对 BERT 预训练好的模型做简单的 fine tuning 就可以在很多 NLP 任务上到达 SOTA。fine tuning 时，BERT 部分参数用预训练参数初始化，任务层参数随机初始化，然后所有参数都一起训练更新。

- 对于分类任务，无论是单句分类 (如情感分类) 任务，还是两个句子是否存在蕴含关系的任务，都是取最后一层 encoder 输出的 512*768 矩阵的第一列 (对应 [CLS])，喂给一

说明	样例
原始文本	使用语言模型来预测下一个词的probability。
分词文本	使用 语言 模型 来 预测 下 一个 词 的 probability 。
原始Mask输入	使 用 语 言 [MASK] 型 来 [MASK] 测 下 一 个 词 的 pro [MASK] ##lity 。
全词Mask输入	使 用 语 言 [MASK] [MASK] 来 [MASK] [MASK] 下 一 个 词 的 [MASK] [MASK] [MASK] 。

图 1.15: WWM 示例

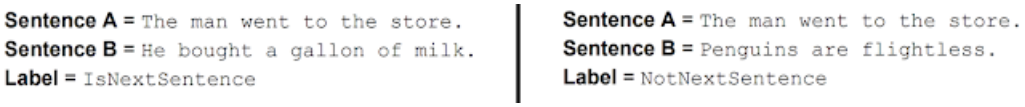

图 1.16: NSP 示例

个 Linear Classifier(比如 softmax) 即可。

- 对于单句的序列标注任务，取最后一层 encoder 输出的 512*768 全部列分别送入不同的 Linear Classifier，预测出每个 token 的标签 (类别)。

BERT 有很多发展，如：

- RoBERTa [49] 没有更改 BERT 的架构，只是对 BERT 进行了更强的训练，比如增大 batch、采用更多参数和更大的训练集、采用 Byte-Pair Encoding(BPE) [6] 这种 tokenization 方法，一句话只做一次 mask 变成做多次 mask(每次 mask 不同的 token) 等方法，从而达到了更好的效果。

- ALBERT(A Lite BERT) [50] 通过对 embedding 参数矩阵做分解 (比如，原来 30K*786 的 embedding 矩阵变成 30K*128(E_1) 和 128*768 (E_2) 两个矩阵，先通过 E_1 做 embedding，得到结果后与 E_2 做个矩向量乘法)，所有 encoder 层共享参数等方法使得参数量大大减小，只有 BERT 的十几分之一，同时效果没有下降。

- ERINE [51] 首先识别 token 序列中的命名实体，然后在知识图谱找到对应的实体，接着根据实体在图谱中的结构导出该实体的语义向量，将实体的知识图谱语义向量与 BERT 原来的 3 个 embedding 一起喂入模型。[52] 除了同时使用不同强度的 mask(字，实体，短语) 外，还使用多轮问答数据 Dialogue Language Model(DLM) 来训练模型，比如 QRQ、QRR、QQR，Q 表示 query，R 表示 foresponse)。DLM 训练任务随机 mask 对话中的任意 token，要求模型预测出 mask 掉的 token，见图1.17，注意 Segment Embedding 换成了 Dialogue Embedding。另外还随机替换 Q 或者 A，要求模型判断一段会话是否为真。

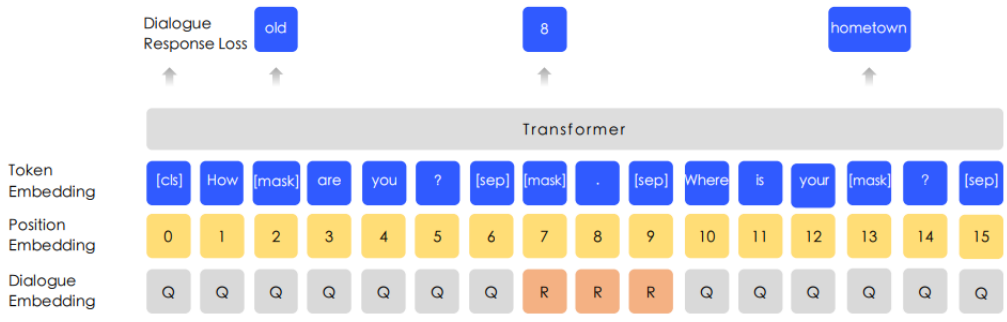

图 1.17: DLM 示例

DistilBERT [53] 采用知识蒸馏来压缩 BERT 的大小，DeBERTa [54] 使用 Relative Position Representations [55]、virtual adversarial 训练方法等技术达到当年的 SuperGLUE 榜首。[56] 对 BERT 做了深入的探讨，比如 BERT 到底学到了什么，怎么学的，效果如何，怎么改善。还有其他基于 Transformer 或者架构类似 BERT 的预训练语言模型，如 XLNet [57](引入了 Permutation Language Model)，ELECTRA [58]，UniLM [59]。[60] 和 [61] 都是不错的 BERT 系列工作的介绍博客。总结起来，BERT 后续很多工作都是提出难度更大、更多样化的预训练任务，从而增加模型的学习难度，同时增加多种来源的信息给到模型，让模型学习得更好。还有一些是训练/推理加速和模型轻量化的工作。

还有一些基于 Encoder-Decoder 架构的预训练模型，比如 BART [62]，T5 [63]。T5 是 Text-to-Text Transfer Transformer 的缩写，它把所有 NLP 任务统一成看做 Text-to-Text 问题，比如 machine translation、question answering、abstractive summarization 和 text classification 四个任务全部当做 Text-to-Text，然后试图用一个模型完成所有 NLP 任务。如图1.18，可以看到每种任务都加了一个 task-specific 的 prefix(冒号前面)，这里需要特别注意的是，两句话的相似度计算也被当做 Text-to-Text 问题 (其实是把相似度离散化，然后当做多分类问题)。T5 采用了与 Transformer 高度类似的架构 (有些许改动)。另外还构建了一个高达 800G 的超大语料集 C4(Colossal Clean Crawled Corpus) 来训练模型。[63] 做了极为丰富的实验，找到最优的训练策略，验证模型效果。最后 T5 模型的参数量高达 11B，打破了很多 NLP 任务的历史记录，当时大幅领先第二名。mT5 [64] 是 T5 的多语言版本，将包括中文在内的多个语种的 NLP 任务推向了新的高度。

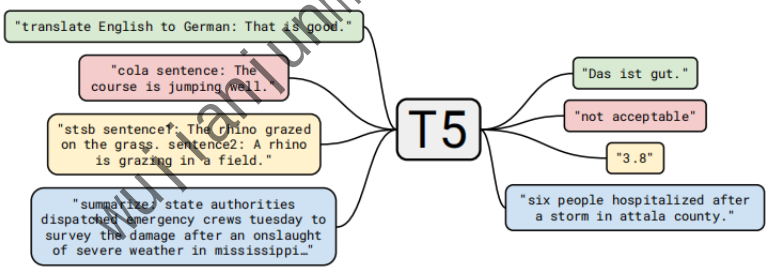

图 1.18: T5 模型输入输出示例

基于 BERT 模型家族做应用的工作也有不少，如

- Sentence-BERT [65]: 要计算两句话的相似度，BERT 要把两句话同时输入模型，最后输出一个相似度分，如果有 10000 句话，找出最相似的句子会导致大量计算，所以说 BERT 的构造使得它不适用于语义相似性搜索以及聚类等无监督任务。通过 SBERT(Sentence-BERT) 模型获取到的句子 embedding，可以直接通过 cos 相似度计算两个句子的相似度，这样无疑会快很多。不同于 BERT 本身使用 [CLS] 作为整句话的 embedding，SBERT 对一句话中所有 token 的 BERT 输出向量求均值作为整句话的 embedding。Sentence-BERT 的团队还发布了 SentenceTransformers 这个 python 包，可以用于计算句子或者文本的 embeddings。基于得到 embedding 向量，可以：

- Semantic Search：基于余弦相似度做相似语义搜索，可以做 symmetric semantic search(如句子搜句子)，也可以做 asymmetric semantic search(如关键词搜句子，句子搜文章，关键词搜文章等)，注意，这两种搜索使用的具体 embedding 模型是有重大差别的。对于一些复杂的搜索，第一步使用向量召回 100 个语料 (retrieval)，因为这一步要从海量语料中快速找到语义相关的，计算不是非常准确，所以需要再次精细计算召回的语料和 query 的相关度 (Re-Rank)。Sentence-BERT 提供了 CrossEncoder 模块 (retrieval 使用的称为 BiEncoder) 加载合适的精排模型 (如 ms-marco-MiniLM-L-12-v2) 进而输出两个句子的相关性得分 (注意，不是两个句子 embedding 之间的余弦相似度)。
- Clustering：先对句子做 embedding，然后基于 embedding 向量做聚类。可以使用多个聚类算法，诸如 k-Means(默认使用欧氏距离，但是规范化后二者等价 [66]，只是此时类中心 (距离均值) 这个概念缺乏基础)，Agglomerative Clustering(层次聚类，从下往上不断合并两个靠近的类，可以支持 cosine 等多种距离，速度很慢)，Fast Clustering(一种快速的社团挖掘算法)。
- 图文混合搜索：首先用模型把 text 和 Imag 转成同一个空间的 embedding 向量，然后就可以做搜索了，包括 Text-to-Image / Image-To-Text / Image-to-Image / Text-to-Text 四种搜索。这里是使用的模型是 OpenAI 的 CLIP([67], [68]，这是一个非常重要的图文匹配模型)。

- KeyBERT [69]：利用 BERT 提取关键词或者关键短语。首先，用 BERT 家族的某个模型 (比如 Sentence-Transformers 中的某个模型) 计算整个文档的 embedding，然后计算每个候选 N-gram 的 embedding(后选 N-gram 可以基于 TF-IDF 之类的方法选出，N 需要用户指定)，最后计算每个候选 N-gram 与整个文档的余弦相似度，把相似度最高者作为文档的关键词。注意中文要先分词，然后把分词后的序列输入 KeyBERT。

- BERTopic [70]：利用 BERT 抽取一组 doc 中的 topic。计算过程为：首先利用 BERT 类模型 (比如 Sentence-Transformers 中的某个模型) 把 doc 转为 embedding 向量，接着使用 UMAP 算法 (类似 t-SNE 算法) 对高维 embedding 降纬，然后使用 HDBSAN 聚类算法 (DBSCAN 算法的升级，与 UMAP 算法协同使用最好) 对低维 embedding 聚类，继续使用 c-TF-IDF(Class-based TF-IDF) 算法提取每个 cluster 中的关键词，这些关键词就是每个 cluster 的 topic。Top2Vec [71] 也是一个不错的 topic 挖掘工具包。

- NER-BERT [72]：有一些用 BERT 做 NER 的软件包，如 [73], [74]。[75] 提供了多种模型完成中文分词，实体提取，词性标注等任务。

参考文献

[1] Natural language processing. [EB/OL]. https://huggingface.co/tasks.

[2] Yoon Kim. Convolutional neural networks for sentence classification, 2014.

[3] Ye Zhang and Byron Wallace. A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification, 2016.

[4] Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. Bag of tricks for efficient text classification, 2016.

[5] Zhiheng Huang, Wei Xu, and Kai Yu. Bidirectional lstm-crf models for sequence tagging. 2015.

[6] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units, 2016.

[7] Changhan Wang, Kyunghyun Cho, and Jiatao Gu. Neural machine translation with byte-level subwords, 2019.

[8] Kaisuke Nakajima Mike Schuster. Japanese and korean voice search, 2012.

[9] Taku Kudo. Subword regularization: Improving neural network translation models with multiple subword candidates, 2018.

[10] Chen Y et al Zeng D, Liu K. Distant supervision for relation extraction via piecewise convolutional neural networks. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1753-–1762, 2015.

[11] nlp 中的实体关系抽取方法总结. [EB/OL]. http://www.uml.org.cn/ai/202009111.asp.

[12] 大话知识图谱–意图识别和槽位填充. [EB/OL]. https://zhuanlan.zhihu.com/p/165963264.

[13] 对话系统中自然语言理解 nlu——意图识别与槽位填充. [EB/OL]. https://blog.csdn.net/orangerfun/article/details/117821179.

[14] Houfeng Wang Xiaodong Zhang. A joint model of intent determination and slot filling for spoken language understanding. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence (IJCAI-16)*, 2016.

[15] Qian Chen, Zhu Zhuo, and Wen Wang. Bert for joint intent classification and slot filling. *arXiv preprint arXiv:1902.10909*, 2019.

[16] Xu Han, Zhengyan Zhang, Ning Ding, Yuxian Gu, Xiao Liu, Yuqi Huo, Jiezhong Qiu, Yuan Yao, Ao Zhang, Liang Zhang, Wentao Han, Minlie Huang, Qin Jin, Yanyan Lan, Yang Liu, Zhiyuan Liu, Zhiwu Lu, Xipeng Qiu, Ruihua Song, Jie Tang, Ji-Rong Wen, Jinhui Yuan, Wayne Xin Zhao, and Jun Zhu. Pre-trained models: Past, present and future, 2021.

[17] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. A neural probabilistic language model. *The journal of machine learning research*, 3:1137–1155, 2003.

[18] Quoc Le and Tomas Mikolov. Distributed representations of sentences and documents. In *International conference on machine learning*, pages 1188–1196. PMLR, 2014.

[19] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[20] Ryan Kiros, Yukun Zhu, Ruslan Salakhutdinov, Richard S Zemel, Antonio Torralba, Raquel Urtasun, and Sanja Fidler. Skip-thought vectors. *arXiv preprint arXiv:1506.06726*, 2015.

[21] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks, 2014.

[22] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation, 2014.

[23] Oriol Vinyals and Quoc Le. A neural conversational model. *arXiv preprint arXiv:1506.05869*, 2015.

[24] Rohit Prabhavalkar, Kanishka Rao, Tara N Sainath, Bo Li, Leif Johnson, and Navdeep Jaitly. A comparison of sequence-to-sequence models for speech recognition. In *Interspeech*, pages 939–943, 2017.

[25] Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. Show and tell: A neural image caption generator. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3156–3164, 2015.

[26] Subhashini Venugopalan, Marcus Rohrbach, Jeff Donahue, Raymond Mooney, Trevor Darrell, and Kate Saenko. Sequence to sequence – video to text, 2015.

[27] Aston Zhang, Zachary C. Lipton, Mu Li, and Alexander J. Smola. Dive into deep learning. *arXiv preprint arXiv:2106.11342*, 2021.

[28] Wanshun Wong. What is teacher forcing?a common technique in training recurrent neural networks. [EB/OL]. https://medium.com/p/3da6217fed1c.

[29] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate, 2016.

[30] Minh-Thang Luong, Hieu Pham, and Christopher D Manning. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*, 2015.

[31] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 6000–6010, Red Hook, NY, USA, 2017. Curran Associates Inc.

[32] Amirhossein Kazemnejad. Transformer architecture: The positional encoding. [EB/OL]. https://kazemnejad.com/blog/transformer_architecture_positional_encoding/.

[33] Sean Robertson. Nlp from scratch: Translation with a sequence to sequence network and attention. [EB/OL]. https://pytorch.org/tutorials/intermediate/seq2seq_translation_tutorial.html#training-the-model.

[34] Harvard. The annotated transformer. [EB/OL]. http://nlp.seas.harvard.edu/annotated-transformer/.

[35] Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V. Le, and Ruslan Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed-length context, 2019.

[36] Iz Beltagy, Matthew E. Peters, and Arman Cohan. Longformer: The long-document transformer, 2020.

[37] Nikita Kitaev, ?ukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer, 2020.

[38] Manzil Zaheer, Guru Guruganesh, Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, and Amr Ahmed. Big bird: Transformers for longer sequences, 2021.

[39] Vasudev Gupta. Understanding bigbird's block sparse attention. [EB/OL]. https://huggingface.co/blog/big-bird.

[40] Sinong Wang, Belinda Z. Li, Madian Khabsa, Han Fang, and Hao Ma. Linformer: Self-attention with linear complexity, 2020.

[41] Vamsi Aribandi, Yi Tay, Tal Schuster, Jinfeng Rao, Huaixiu Steven Zheng, Sanket Vaibhav Mehta, Honglei Zhuang, Vinh Q. Tran, Dara Bahri, Jianmo Ni, Jai Gupta, Kai Hui, Sebastian Ruder, and Donald Metzler. Ext5: Towards extreme multi-task scaling for transfer learning, 2022.

[42] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[43] Dileep Patchigolla. Understanding bert architecture. [EB/OL]. https://medium.com/analytics-vidhya/understanding-bert-architecture-3f35a264b187.

[44] Jacob Devlin and Ming-Wei Chang. Open sourcing bert: State-of-the-art pre-training for natural language processing. [EB/OL]. https://blog.research.google/2018/11/open-sourcing-bert-state-of-art-pre.html.

[45] bert-large-ner. [EB/OL]. https://github.com/ymcui/Chinese-BERT-wwm.

[46] Mike Lewis, Marjan Ghazvininejad, Gargi Ghosh, Armen Aghajanyan, Sida Wang, and Luke Zettlemoyer. Pre-training via paraphrasing, 2020.

[47] bert-base-ner. [EB/OL]. https://huggingface.co/bert-base-uncased.

[48] Yiming Cui, Wanxiang Che, Ting Liu, Bing Qin, and Ziqing Yang. Pre-training with whole word masking for chinese BERT. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 29:3504–3514, 2021.

[49] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach, 2019.

[50] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. Albert: A lite bert for self-supervised learning of language representations, 2020.

[51] Zhengyan Zhang, Xu Han, Zhiyuan Liu, Xin Jiang, Maosong Sun, and Qun Liu. Ernie: Enhanced language representation with informative entities, 2019.

[52] Yu Sun, Shuohuan Wang, Yukun Li, Shikun Feng, Xuyi Chen, Han Zhang, Xin Tian, Danxiang Zhu, Hao Tian, and Hua Wu. Ernie: Enhanced representation through knowledge integration, 2019.

[53] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter, 2020.

[54] Pengcheng He, Xiaodong Liu, Jianfeng Gao, and Weizhu Chen. Deberta: Decoding-enhanced bert with disentangled attention, 2021.

[55] Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. Self-attention with relative position representations, 2018.

[56] Anna Rogers, Olga Kovaleva, and Anna Rumshisky. A primer in bertology: What we know about how bert works, 2020.

[57] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Ruslan Salakhutdinov, and Quoc V. Le. Xlnet: Generalized autoregressive pretraining for language understanding, 2020.

[58] Kevin Clark, Minh-Thang Luong, Quoc V. Le, and Christopher D. Manning. Electra: Pre-training text encoders as discriminators rather than generators, 2020.

[59] Li Dong, Nan Yang, Wenhui Wang, Furu Wei, Xiaodong Liu, Yu Wang, Jianfeng Gao, Ming Zhou, and Hsiao-Wuen Hon. Unified language model pre-training for natural language understanding and generation, 2019.

[60] [EB/OL]. https://jkboy.com/archives/23359.html, title = BERT 系列 RoBERTa ALBERT ERINE 详解与使用学习笔记.

[61] 预训练自然语言模型. [EB/OL]. https://leovan.me/cn/2020/03/pre-trained-model-for-nlp/.

[62] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension, 2019.

[63] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *The Journal of Machine Learning Research*, 21(1):5485–5551, 2020.

[64] Linting Xue, Noah Constant, Adam Roberts, Mihir Kale, Rami Al-Rfou, Aditya Siddhant, Aditya Barua, and Colin Raffel. mt5: A massively multilingual pre-trained text-to-text transformer, 2021.

[65] Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks, 2019.

[66] Using k-means with cosine similarity - python. [EB/OL]. https://stackoverflow.com/questions/46409846/using-k-means-with-cosine-similarity-python/.

[67] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision, 2021.

[68] Clip. [EB/OL]. https://github.com/openai/CLIP.

[69] Maarten Grootendorst. Keybert: Minimal keyword extraction with bert., 2020.

[70] Maarten Grootendorst. Bertopic: Neural topic modeling with a class-based tf-idf procedure, 2022.

[71] Dimo Angelov. Top2vec: Distributed representations of topics. 2020.

[72] Zihan Liu, Feijun Jiang, Yuxiang Hu, Chen Shi, and Pascale Fung. Ner-bert: A pre-trained model for low-resource entity tagging, 2021.

[73] bert-base-chinese-ner. [EB/OL]. https://huggingface.co/ckiplab/bert-base-chinese-ner.

[74] bert-large-ner. [EB/OL]. https://huggingface.co/dslim/bert-large-NER.

[75] Ckip transformers. [EB/OL]. https://github.com/ckiplab/ckip-transformers.

[76] The atis (airline travel information system) dataset. [EB/OL]. https://github.com/howl-anderson/ATIS_dataset.

[77] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models, 2020.

[78] W Bradley Knox and Peter Stone. Tamer: Training an agent manually via evaluative reinforcement. In *2008 7th IEEE international conference on development and learning*, pages 292–297. IEEE, 2008.

[79] Garrett Warnell, Nicholas Waytowich, Vernon Lawhern, and Peter Stone. Deep tamer: Interactive agent shaping in high-dimensional state spaces. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

[80] Daniel M Ziegler, Nisan Stiennon, Jeffrey Wu, Tom B Brown, Alec Radford, Dario Amodei, Paul Christiano, and Geoffrey Irving. Fine-tuning language models from human preferences. *arXiv preprint arXiv:1909.08593*, 2019.

[81] Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. *Advances in neural information processing systems*, 30, 2017.

[82] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training. 2018.

[83] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[84] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[85] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35:27730–27744, 2022.

[86] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[87] Paul Christiano, Jan Leike, Tom B. Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences, 2023.

[88] Image gpt. [EB/OL]. https://openai.com/index/image-gpt/.

[89] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language models, 2023.

[90] Prajit Ramachandran, Barret Zoph, and Quoc V. Le. Searching for activation functions, 2017.

[91] Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding, 2023.

[92] Hugo Touvron, Louis Martin, and Stone. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

[93] Zhengxiao Du, Yujie Qian, Xiao Liu, Ming Ding, Jiezhong Qiu, Zhilin Yang, and Jie Tang. Glm: General language model pretraining with autoregressive blank infilling, 2022.

[94] Aohan Zeng, Xiao Liu, Zhengxiao Du, Zihan Wang, Hanyu Lai, Ming Ding, Zhuoyi Yang, Yifan Xu, Wendi Zheng, Xiao Xia, Weng Lam Tam, Zixuan Ma, Yufei Xue, Jidong Zhai, Wenguang Chen, Peng Zhang, Yuxiao Dong, and Jie Tang. Glm-130b: An open bilingual pre-trained model, 2023.

[95] Hongyu Wang, Shuming Ma, Li Dong, Shaohan Huang, Dongdong Zhang, and Furu Wei. Deepnet: Scaling transformers to 1,000 layers, 2022.

[96] Chatglm-6b. [EB/OL]. https://github.com/THUDM/ChatGLM-6B.

[97] Chatglm2-6b. [EB/OL]. https://github.com/THUDM/ChatGLM2-6B.

[98] Chatglm3. [EB/OL]. https://github.com/THUDM/ChatGLM3.

[99] Ming Ding, Zhuoyi Yang, Wenyi Hong, Wendi Zheng, Chang Zhou, Da Yin, Junyang Lin, Xu Zou, Zhou Shao, Hongxia Yang, et al. Cogview: Mastering text-to-image generation

via transformers. *Advances in Neural Information Processing Systems*, 34:19822–19835, 2021.

[100] Visualglm-6b. [EB/OL]. https://github.com/THUDM/VisualGLM-6B/tree/main.

[101] Weihan Wang, Qingsong Lv, Wenmeng Yu, Wenyi Hong, Ji Qi, Yan Wang, Junhui Ji, Zhuoyi Yang, Lei Zhao, Xixuan Song, Jiazheng Xu, Bin Xu, Juanzi Li, Yuxiao Dong, Ming Ding, and Jie Tang. Cogvlm: Visual expert for pretrained language models, 2024.

[102] Wenyi Hong, Weihan Wang, Qingsong Lv, Jiazheng Xu, Wenmeng Yu, Junhui Ji, Yan Wang, Zihan Wang, Yuxuan Zhang, Juanzi Li, Bin Xu, Yuxiao Dong, Ming Ding, and Jie Tang. Cogagent: A visual language model for gui agents, 2023.

[103] Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin de Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for nlp, 2019.

[104] Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation, 2021.

[105] Brian Lester, Rami Al-Rfou, and Noah Constant. The power of scale for parameter-efficient prompt tuning, 2021.

[106] Xiao Liu, Yanan Zheng, Zhengxiao Du, Ming Ding, Yujie Qian, Zhilin Yang, and Jie Tang. Gpt understands, too, 2023.

[107] Xiao Liu, Kaixuan Ji, Yicheng Fu, Weng Lam Tam, Zhengxiao Du, Zhilin Yang, and Jie Tang. P-tuning v2: Prompt tuning can be comparable to fine-tuning universally across scales and tasks, 2022.

[108] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models, 2021.

[109] Qingru Zhang, Minshuo Chen, Alexander Bukharin, Nikos Karampatziakis, Pengcheng He, Yu Cheng, Weizhu Chen, and Tuo Zhao. Adalora: Adaptive budget allocation for parameter-efficient fine-tuning, 2023.

[110] Haokun Liu, Derek Tam, Mohammed Muqeeth, Jay Mohta, Tenghao Huang, Mohit Bansal, and Colin Raffel. Few-shot parameter-efficient fine-tuning is better and cheaper than in-context learning, 2022.

[111] Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness, 2022.

[112] Noam Shazeer. Fast transformer decoding: One write-head is all you need, 2019.

[113] Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebrón, and Sumit Sanghai. Gqa: Training generalized multi-query transformer models from multi-head checkpoints, 2023.

[114] Guolin Ke, Di He, and Tie-Yan Liu. Rethinking positional encoding in language pre-training, 2021.

[115] 一文看懂 llama 中的旋转式位置编. [EB/OL]. https://zhuanlan.zhihu.com/p/642884818.

[116] Ofir Press, Noah A. Smith, and Mike Lewis. Train short, test long: Attention with linear biases enables input length extrapolation, 2022.

[117] Ta-Chung Chi, Ting-Han Fan, Peter J. Ramadge, and Alexander I. Rudnicky. Kerple: Kernelized relative positional embedding for length extrapolation, 2022.

[118] Ta-Chung Chi, Ting-Han Fan, Alexander I. Rudnicky, and Peter J. Ramadge. Dissecting transformer length extrapolation via the lens of receptive field analysis, 2023.

[119] Shouyuan Chen, Sherman Wong, Liangjian Chen, and Yuandong Tian. Extending context window of large language models via positional interpolation, 2023.

[120] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding, 2019.

[121] Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. Superglue: A stickier benchmark for general-purpose language understanding systems, 2020.