基于 BERT+CRF 的命名实体识别（NER）实验报告

一、 引言

命名实体识别（NER，Named Entity Recognition）是自然语言处理中的一项基础任务，旨在从文本中识别出具有特定意义的实体，如人名、地名、组织名等。

示例文本：我 爱 北 京 天 安 门
NER结果：O O B_LOC I_LOC O O O

1.1 数据集概览

这里实验数据的标签集包括'O', 'B_LOC', 'I_LOC', 'B_T', 'I_T', 'B_PER', 'I_PER', 'B_ORG', 'I_ORG' 等（由训练数据统计得到）。

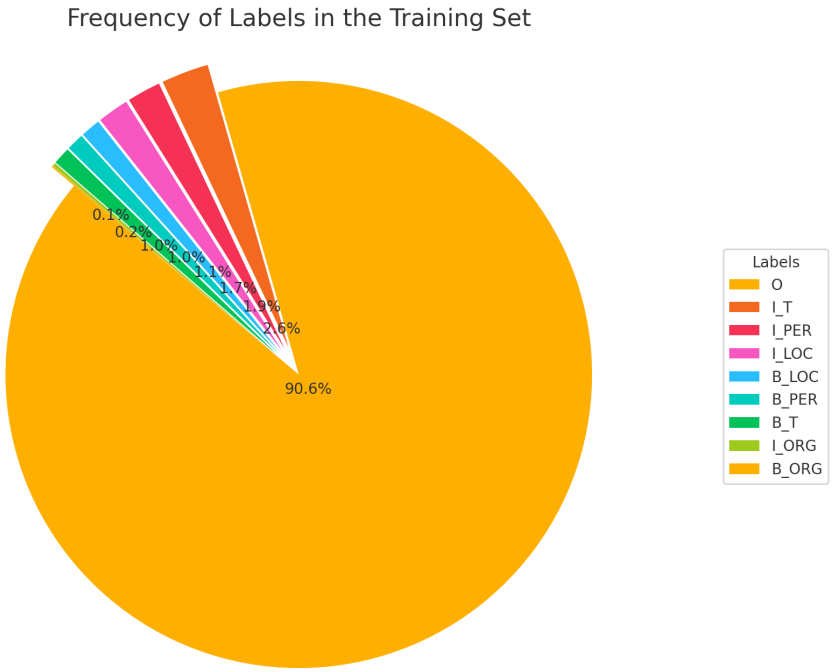


图 1　标签分布图

以下是这些标签的中文含义：

标签	含义	序号
O	其他	5
B_LOC	地点开始	2
I_LOC	地点内部	4
B_T	时间开始	3
I_T	时间内部	8
B_PER	人名开始	0
I_PER	人名内部	7
B_ORG	组织开始	6
I_ORG	组织内部	1

表 1　标签含义

二、 使用 CRF 分类的原因

在命名实体识别（NER）任务中，条件随机场（CRF，Conditional Random Field）相较于直接使用 Softmax 层有着显著的优势。CRF 是一种用于标注和分割序列数据的概率模型，通过考虑标签之间的转移关系，可以更准确地捕捉到序列数据中的上下文信息。

2.1 CRF 的工作原理

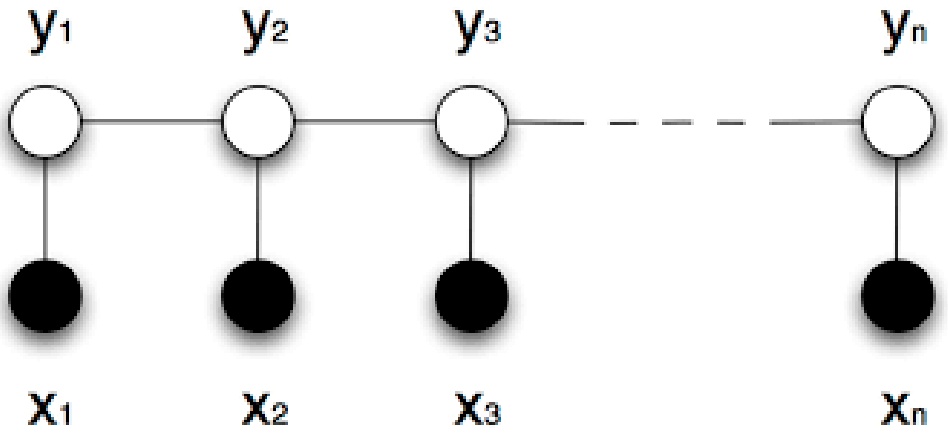


图 2　线性链条件随机场结构示意图

CRF 模型通过定义特征函数来表示观察序列和状态序列之间的关系。给定一个观

察序列 $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ 和对应的标签序列 $\mathbf{y} = (y_1, y_2, \ldots, y_n)$，CRF 的条件概率定义为：

$$P(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \exp\left(\sum_{t=1}^{n}\sum_{k} \lambda_k f_k(y_{t-1}, y_t, \mathbf{x}, t)\right) \tag{1}$$

其中，$Z(\mathbf{x})$ 是规范化因子，f_k 是特征函数，λ_k 是特征权重。

2.2 CRF 在 NER 任务中的应用

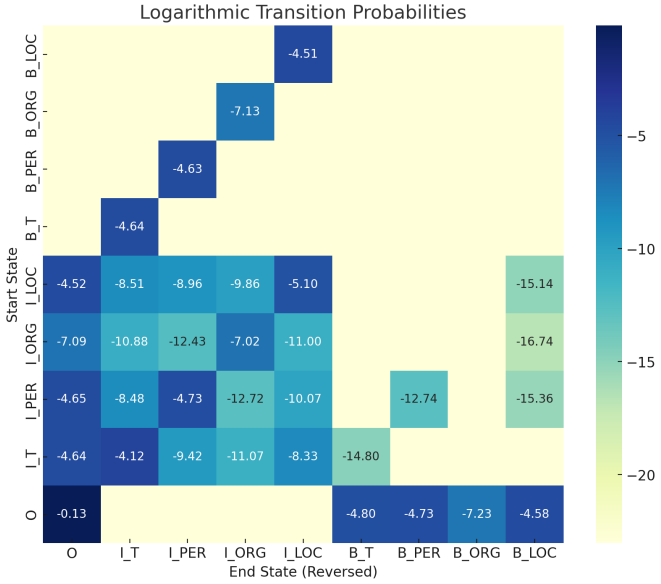


图 3　标签转移概率图

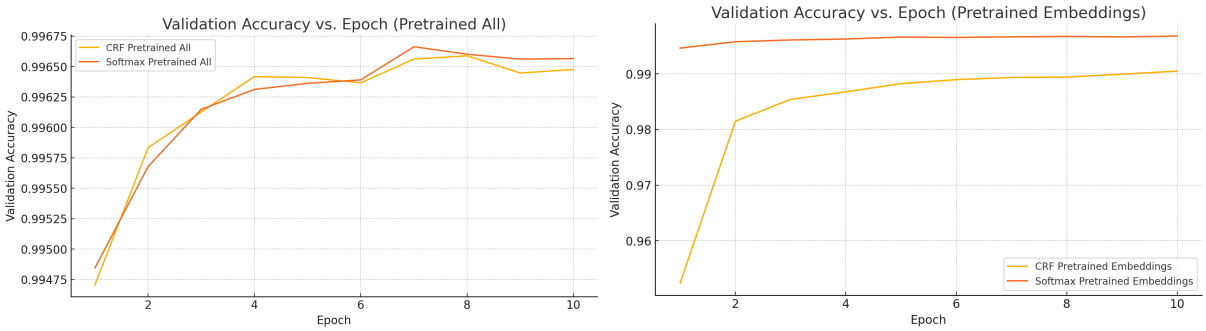
在 NER 任务中，标签之间存在明显的转移规律，例如一个”B_LOC” 标签后面通常会跟一个”I_LOC” 标签。CRF 通过学习这些转移概率，可以更好地预测标签序列。相比之下，直接使用 Softmax 层只考虑每个标签的独立概率，忽略了标签之间的依赖关系，容易导致不合理的标签序列。

2.3 BERT+Softmax 与 BERT+CRF 的对比

在 NER 任务中，BERT 模型已经能够捕捉到丰富的上下文信息，但在标签解码阶段，Softmax 层无法有效利用标签之间的转移信息。CRF 通过引入转移概率，进一步提升了模型的序列标注能力。

下图展示了两种模型在验证集上的准确率对比：

实验结果表明，在预训练词嵌入的情况下，BERT+CRF 模型在准确率上显著优于 BERT+Softmax 模型，特别是在复杂标签序列的预测上，BERT+CRF 模型表现更为出色，但是如果二者的基础模型都来自于大规模预料预训练的权重，二者差异不明显。



((a)) BERT+Softmax 与 BERT+CRF(预训练基础模型BERT) 的准确率对比图

((b)) BERT+Softmax 与 BERT+CRF(预训练词嵌入) 的准确率对比图

图 4　BERT+Softmax 与 BERT+CRF 的准确率对比

三、 训练流程

本次实验在单卡 L20（48GB 显存）上进行。神经网络层学习率设置为 5e-5，CRF 层学习率设置为神经网络层的 10 倍，即 3e-5。基础模型结构使用 bert-base-chinese，使用预训练权重，共 12 层，词表大小为 21128，嵌入维度为 768。在探究层数和是否预训练对训练过程的影响时，采取训练 10 个 epoch，batch size 设置为 420，pad 长度为 128；但是在最终精调模型时，pad 长度设置为 300,batch size 设置为 120，进行 5 个 epoch 训练。

3.1 训练设置

- 实验平台：AutoDL 算力云
- 设备：单卡 L20（48GB 显存）
- 训练轮数：10 个 epoch
- Batch size：420
- 优化算法：AdamW
- 学习率：5e-5(NN)，3e-5(CRF)
- 填充长度：128
- 基础模型：bert-base-chinese
- 模型层数：12 层
- 词表大小：21128
- 嵌入维度：768

3.2 模型结构展示

BERT+CRF 模型的结构如下图所示：

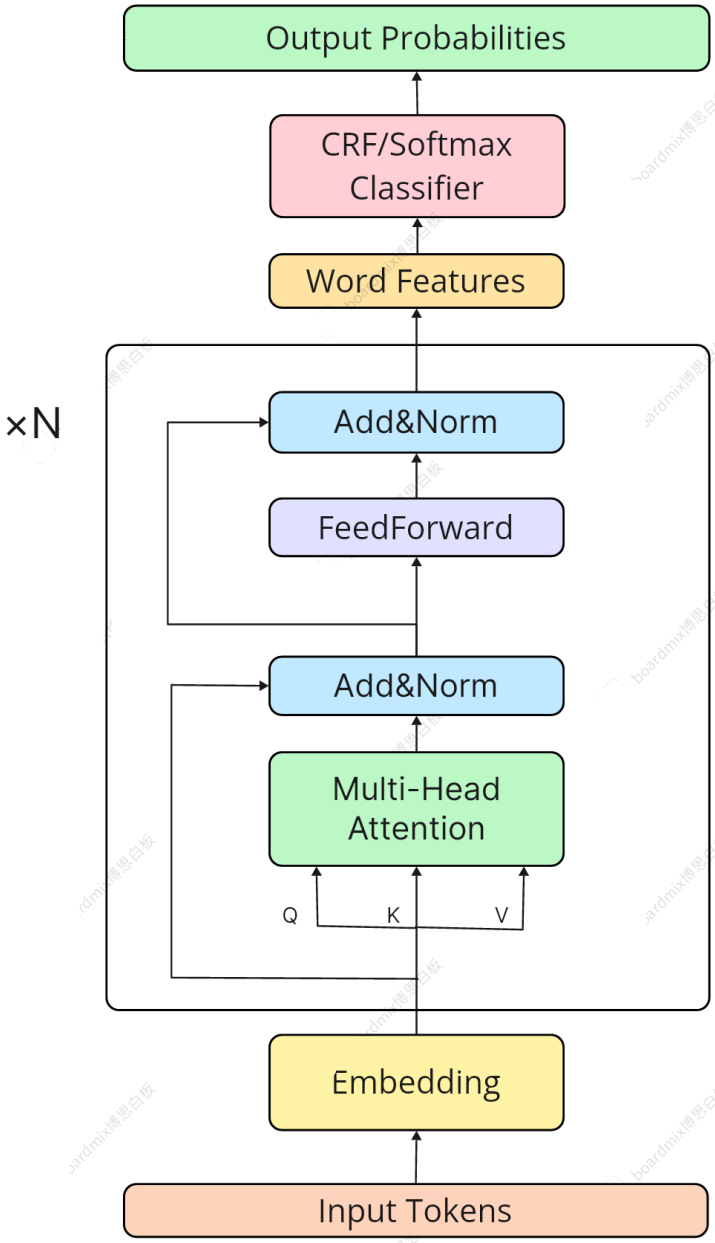


图 5　BERT+CRF 模型结构

3.3 模型效果

在训练到第 5 个 epoch 时，损失曲线已经非常平坦，验证集上的准确率改进不大，我们采取了 EarlyStop 策略，最佳的模型在发展集上的准确率是 99.65

图 6　训练过程

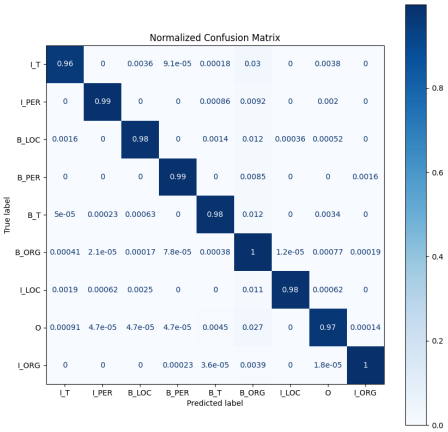


图 7　归一化混淆矩阵

3.4 输出样例

对于 *test.txt* 的测试文本：

测试文本的第一行：

记 者 1 月 1 5 日 从 上 海 铁 路 局 淮 南
西 站 获 悉 ， 淮 南 铁 路 预 计 春 运 期 间 发 送 旅 客 3 3 . 3 万 人 。
预 计 客 流 最 高 峰 日 为 2 月 6 日 ， 将 发 送 旅 客 1 . 8
万 人 ， 淮 南 东 开 往 广 州 南 及 北 京 方 向 的 高 铁 、
淮 南 站 开 往 沪 杭 甬 方 向 的 列 车 将 是 热 门 车 次 。

输出文件的第一行：

```
O O B_T I_T I_T I_T I_T O B_LOC I_LOC O O O B_LOC I_LOC
O O O O O B_LOC I_LOC O O O O O O O O O O O O O O O O O O
O O O O O O O O O B_T I_T I_T I_T O O O O O O O O O
O O O B_LOC I_LOC O O O B_LOC I_LOC O O B_LOC I_LOC O O O O O O
O O O O O O O O O O O O O O O O O O O O
```

其中，1 月 15 日、2 月 6 日作为时间实体，上海、淮南、广州、北京作为地点实体被模型正确识别出来，而“淮南站”的“淮南”没有被当作地点实体。

四、 详细分析

4.1 Padding 长度的选择

在 NER 任务中，句子长度的不同会对模型的训练和推理产生影响。为了选择合适的 padding 长度，我们统计了训练语料中所有句子的长度分布，找到了其上 3σ 分位点为 292。因此，我们选择将句子 pad 到长度 300，以覆盖绝大多数的句子长度，确保训练的有效性和模型的稳定性。在给对测试集进行预测时，考虑到测试集中存在超长句子已经超过了 bert 模型的输入长度上限（512 个 tokens），这里对其进行切分处理（如以逗号或句号为分割点），分别预测，然后将预测结果拼接起来。

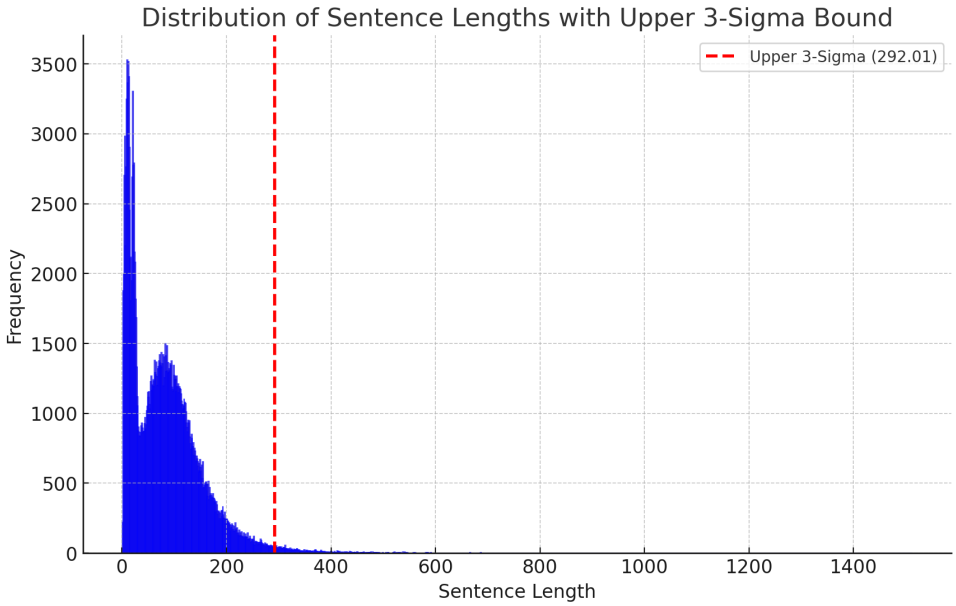


图 8　句子长度分布及 3σ 分位点

4.2 模型层数对训练过程的影响

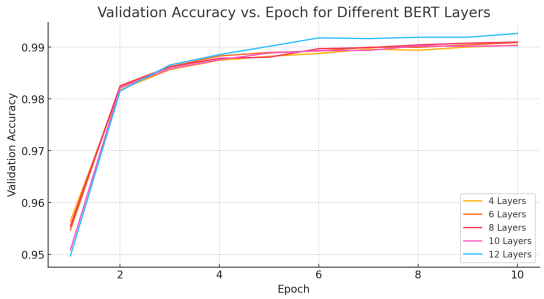


图 9　不同层数模型的准确率曲线

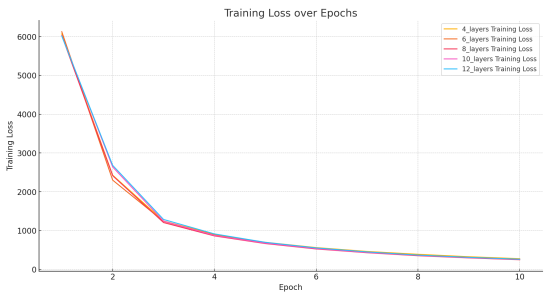


图 10　不同层数模型的损失曲线

为了探究模型层数对训练过程的影响，我们对不同层数的 BERT 模型进行了实验。

理论上，增加模型层数可以提升模型的表达能力，但同时也增加了计算复杂度和训练时间。

从图中可以看出，不同层数的 BERT 模型在准确率曲线上并没有体现出很大的差别，这与我们的预料是不符的。仅仅是更多的模型层数会消耗更多的算力，却没有达到更好的模型性能。这可能是训练 epoch 过少，模型没有达到完全收敛，理论上模型收敛时达到的损失仅仅与模型规模有关，与模型结构关系不大，但是从这 10 个 epoch 的训练中没有体现出这一点。

4.3 是否使用预训练权重对训练过程的影响

以下是使用与不使用预训练权重的模型在训练过程中的损失变化情况对比：

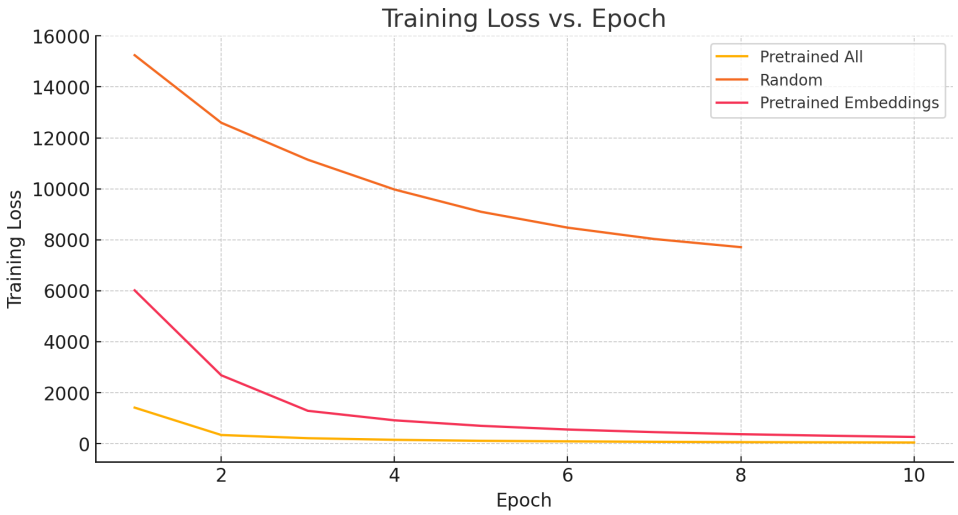


图 11　使用与不使用预训练权重的训练损失对比

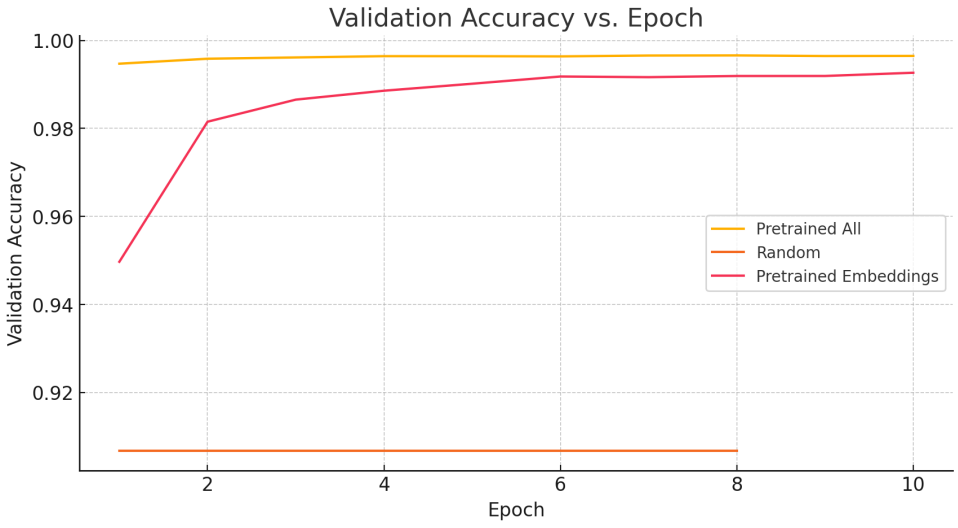


图 12　使用与不使用预训练权重的验证准确率对比

在本次实验中，我们分别测试了使用和不使用预训练权重对模型训练的影响，其中使用预训练权重又分为仅仅使用预训练词嵌入和使用完整的预训练 BERT 基础模型。实验结果显示，使用预训练权重可以显著加快模型的收敛速度，并提高最终的模型性能，完全随机初始化的模型几乎无法在段时间内达到收敛，其准确率曲线很低。

4.4 CRF 层学习率的设置

实验资料表明，CRF 层在 NER 任务中起到了重要的作用，经验上 CRF 的学习率常常取 NN 层的 100 倍，这样 CRF 权重才能及时地收敛，否则随着 NN 的收敛，即时模型达到较高的准确率，CRF 的转移概率也未必合理。因为 CRF 层的参数较少，较高的学习率可以加速其收敛。

五、 讨论

以 BERT 为代表的纯编码器架构模型，以 GPT 为代表的纯解码器架构模型，以 T5 为代表的完整的 Transformer 模型，理论上都可以完成命名实体识别任务，理论上纯编码器架构的 BERT 的性能最好，这是因为 BERT 能够高效地编码输入序列中的上下文信息。BERT 的双向编码机制使其能够同时捕捉到左侧和右侧的上下文信息，这对于精确定位和识别实体边界非常重要，要使得 GPT 和 T5 架构来完成 NER 任务成为可能，则需要采取合理的数据处理和训练方法。

5.1 GPT 架构在 NER 任务中的应用和问题

理论上 GPT 可以处理 NER 任务，以下是 GPT 用于处理 NER 的数据及其标签的示例：

GPT input:

根据之前的示例，请对以下汉字进行命名实体识别，你仅仅需要给出答案即可：涉嫌杀害流浪汉的“富二代”伍德黑德

GPT output:

O O O O O O O O O O O O O O B_PER I_PER I_PER I_PER

GPT（生成式预训练变换器）是一个纯解码器模型，主要用于生成任务。其单向编码机制（从左到右）在处理 NER 任务时存在一定的局限性。具体而言，GPT 在预测当前词的标签时，只能利用该词前面的上下文信息，而无法利用其后的信息。这会导致在某些情况下无法准确识别实体的边界。

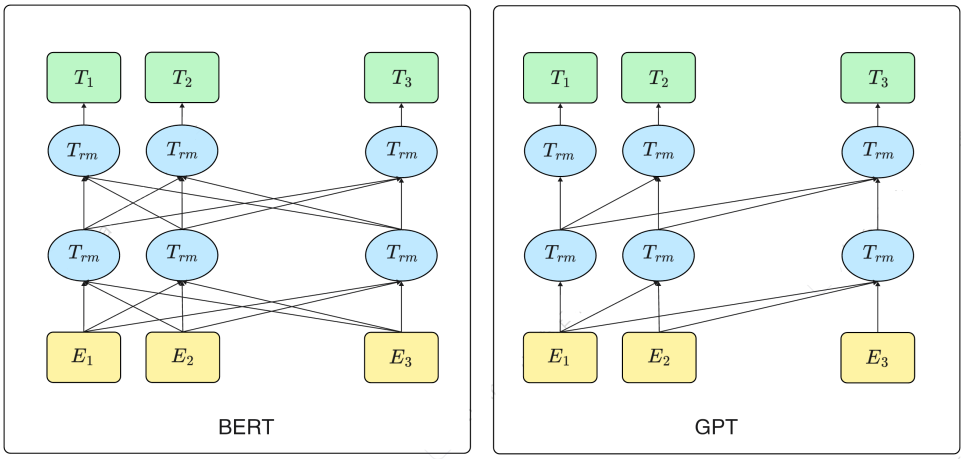


图 13　GPT 与 BERT 的区别

此外，GPT 需要将 NER 任务转化为一种生成任务，例如通过特定的格式将标签嵌入到生成的序列中。这种转化过程可能会引入额外的复杂性，并且对模型的训练和推理效率产生负面影响。

5.2 T5 架构在 NER 任务中的应用和问题

T5（文本到文本迁移变换器）是一种编码器-解码器架构，将所有 NLP 任务都统一成文本到文本的形式。尽管这种统一的框架使得 T5 在多任务学习中表现优异，但在处理 NER 任务时也存在一定挑战。

首先，T5 将 NER 任务视为一种序列到序列的转换任务，需要将输入文本和对应的标签序列进行适当的格式化。这可能会导致标签序列变得冗长，增加模型的训练复杂度和推理时间，比如，需要构造的训练数据应为：

T5 input:

涉嫌杀害流浪汉的“富二代”伍德黑德

T5 output:

涉 O 嫌 O 杀 O 害 O 流 O 浪 O 汉 O 的 O “O 富 O 二 O 代 O ” O 伍 B_PER 德 I_PER 黑 I_PER 德 I_PER

其次，T5 的解码器部分在生成标签序列时，可能会受到生成任务固有的问题的影响，例如标签的重复生成或遗漏。这会对 NER 任务的精度产生负面影响。

5.3 BERT 在 NER 任务中的优势

相比之下，纯编码器架构的 BERT 在处理 NER 任务比 GPT 和 T5 时具有明显的优势。其双向编码机制能够全面捕捉上下文信息，精确识别实体边界。同时，BERT 的输入格式相对简单，不需要将 NER 任务转化为生成任务或序列到序列转换任务，这使得模型的训练和推理更加高效和准确。尽管目前已经存在很多模型架构在多数据集评测上比 BERT+CRF 具有更高的准确率，但是基于 BERT+CRF 架构的模型性能与当前 SOTA 差别不大，在 NER 任务中仍然是目前的最佳选择之一。

Rank	Model Name	Accuracy	Paper Title	Year
1	ACE + document-context	94.6	Automated Concatenation of Embeddings for Structured Prediction	2021
2	LUKE 483M	94.3	LUKE: Deep Contextualized Entity Representations with Entity-aware Self-attention	2020
3	Co-regularized LUKE	94.22	Learning from Noisy Labels for Entity-Centric Information Extraction	2021
4	ASP+T5-3B	94.1	Autoregressive Structured Prediction with Language Models	2022
5	FLERT XLM-R	94.09	FLERT: Document-Level Features for Named Entity Recognition	2020
6	PL-Marker	94.0	Packed Levitated Marker for Entity and Relation Extraction	2021
7	CL-KL	93.85	Improving Named Entity Recognition by External Context Retrieving and Cooperative Learning	2021
8	XLNet-GCN	93.82	Named entity recognition architecture combining contextual and global features	2020

9	ASP+flan-T5-large	93.8	Autoregressive Structured Prediction with Language Models	2022
10	InferNER	93.76	InferNER: an attentive model leveraging the sentence-level information for Named Entity Recognition in Microblogs	2021
11	Cross-sentence context (First)	93.74	Exploring Cross-sentence Contexts for Named Entity Recognition with BERT	2020
12	Baseline + BS	93.65	Boundary Smoothing for Named Entity Recognition	2022
13	ACE	93.64	Automated Concatenation of Embeddings for Structured Prediction	2020
14	**BERT-CRF**	**93.6**	**Focusing on Potential Named Entities During Active Label Acquisition**	**2021**
15	CNN Large + fine-tune	93.5	Cloze-driven Pretraining of Self-attention Networks	2019
16	Biaffine-NER	93.5	Named Entity Recognition as Dependency Parsing	2020
17	GCDT + BERT-L	93.47	GCDT: A Global Context Enhanced Deep Transition Architecture for Sequence Labeling	2019
18	I-DARTS + Flair	93.47	Improved Differentiable Architecture Search for Language Modeling and Named Entity Recognition	2019
19	CrossWeigh + Pooled Flair	93.43	CrossWeigh: Training Named Entity Tagger from Imperfect Annotations	2019

20	LSTM-CRF+ELMo+BERT+Flair	93.38	Neural Architectures for Nested NER through Linearization	2019
21	Hierarchical + BERT	93.37	Hierarchical Contextualized Representation for Named Entity Recognition	2019
22	BERT-CRF (Replicated in AdaSeq)	93.35	Improving Named Entity Recognition by External Context Retrieving and Cooperative Learning	2021
23	BERT-MRC+DSC	93.33	Dice Loss for Data-imbalanced NLP Tasks	2019
24	XLNet	93.28	Named entity recognition architecture combining contextual and global features	2020
25	BARTNER	93.24	A Unified Generative Framework for Various NER Subtasks	2021
26	GoLLIE	93.1	GoLLIE: Annotation Guidelines improve Zero-Shot Information-Extraction	2023
27	Flair embeddings	93.09	Contextual String Embeddings for Sequence Labeling	2018
28	PromptNER [RoBERTa-large]	93.08	PromptNER: Prompt Locating and Typing for Named Entity Recognition	2023
29	W2NER	93.07	Unified Named Entity Recognition as Word-Word Relation Classification	2021
30	BERT-MRC	93.04	A Unified MRC Framework for Named Entity Recognition	2019

A 附录

1.1 参考文献

参考文献

[1] Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *NAACL-HLT*.

[2] Lafferty, J., McCallum, A., & Pereira, F. (2001). Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data. In *ICML*.

[3] Jing Li, Aixin Sun, Jianglei Han, Chenliang Li. A Survey on Deep Learning for Named Entity Recognition. *IEEE Transactions on Knowledge and Data Engineering*, 2020.

[4] Papers with Code. Named Entity Recognition (NER) on CoNLL-2003. `https://paperswithcode.com/sota/named-entity-recognition-ner-on-conll-2003`.

[5] `https://www.zhihu.com/question/316740909/answer/2383001471`

[6] `https://www.zhihu.com/market/pub/120231524/manuscript/1434957031799734272`

1.2 代码文件名和说明对照表

表 3　代码文件名和说明对照表

文件名	说明
datasets.py	包含用于加载和处理数据集的函数和类。
evaluation.py	包含用于评估模型性能的函数。
model.py	定义了深度学习模型的结构和相关函数。
requirements.txt	列出了项目所需的依赖库及其版本。
run_model_selection_num_layers.py	运行不同层数模型选择的脚本。
run_model_type_selection.py	运行不同模型类型选择的脚本。
train.py	主要的训练脚本，包含模型训练的主要逻辑。
test.py	读取测试文件，生成标签文件的脚本。
check.py	检查样本和标签是否字字对应。

1.3 大模型使用报告

在撰写本文档过程中，使用了大语言模型辅助，涉及到：

- 设计文档结构，帮助确定章节划分和内容组织。
- 辅助完成各章节的文字撰写和润色工作。
- 协助编写 LaTeX 格式的数学公式和代码段。
- 根据提示，辅助完成代码

User Input:
以下是我的模型训练代码，请补充其中标记 TODO 的内容【我的代码】
OpenAI GPT-4:
下面是对你的模型训练代码的补充，包括打印模型元数据、保存模型、将模型和数据移动到 CUDA 等操作：【模型输出代码】

User Input:
我训练过程中记录的日志包含了训练损失，验证损失，验证准确率。请根据我上传的日志文件，绘制训练损失和验证准确率曲线【日志文件】
OpenAI GPT-4:
【分析过程】
训练损失和验证准确率曲线如上图所示。蓝色曲线表示训练损失，橙色曲线表示验证损失，绿色曲线表示验证准确率。通过这些曲线可以观察到，随着训练轮次的增加，训练损失和验证损失总体上呈现下降趋势，而验证准确率保持在较高水平。

User Input:
使用 Latex 排版该表格，并着重强调 BERT+CRF 这一行【Benchmark 数据】
OpenAI GPT-4:
以下是使用 LaTeX 排版的表格，并着重强调了 BERT-CRF 这一行：【Latex 源码】

B 关键代码

模型结构有关代码：

```python
import torch
from transformers import BertTokenizer, BertModel
from torchcrf import CRF

class BERT_CRF_NER(torch.nn.Module):
    def __init__(self, bert_model_name, num_tags):
        super(BERT_CRF_NER, self).__init__()
        self.bert = BertModel.from_pretrained(bert_model_name)
        self.dropout = torch.nn.Dropout(0.1)
        self.classifier = torch.nn.Linear(self.bert.config.hidden_size, num_tags)
        self.crf = CRF(num_tags, batch_first=True)

    def forward(self, input_ids, attention_mask, labels=None):
        outputs = self.bert(input_ids, attention_mask=attention_mask)
        sequence_output = self.dropout(outputs[0])
        emissions = self.classifier(sequence_output)
        if labels is not None:
            loss = -self.crf(emissions, labels, mask=attention_mask.byte())
            return loss
        else:
            return self.crf.decode(emissions, mask=attention_mask.byte())

class BERT_Softmax(nn.Module):
    def __init__(self, bert_model_name, num_labels, cache_dir='./bert-base-chinese'):
        super(BERT_Softmax, self).__init__()
        self.bert = BertModel.from_pretrained(bert_model_name, cache_dir=cache_dir)
        self.dropout = nn.Dropout(0.1)
        self.fc = nn.Linear(self.bert.config.hidden_size, num_labels)
        self.softmax = nn.Softmax(dim=-1)

    def forward(self, input_ids, attention_mask, labels=None):
        outputs = self.bert(input_ids, attention_mask=attention_mask)
        sequence_output = self.dropout(outputs[0])
        logits = self.fc(sequence_output)
        predictions = self.softmax(logits)

        if labels is not None:
            loss_fn = nn.CrossEntropyLoss()
            # CrossEntropyLoss expects inputs of shape (N, C) and targets of shape (N)
            # Flatten the inputs and targets
            active_loss = attention_mask.view(-1) == 1
            active_logits = logits.view(-1, logits.shape[-1])[active_loss]
            active_labels = labels.view(-1)[active_loss]
            loss = loss_fn(active_logits, active_labels)
            return loss
        else:
            return predictions.max(dim=-1)[1]

# Model initialization and training procedures are omitted for brevity
```

Listing 1: model.py

模型训练关键代码：

```
model = BERT_CRF('bert-base-chinese', num_labels=num_labels, num_hidden_layers=num_hidden_layers, pretrained=pretrained)
    # Move model and tensors to CUDA (if available)
    device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
    model.to(device)
    save_dir = f'./models/type{args.model_type}_layers_{num_hidden_layers}_pretrained{pretrained}'
    os.makedirs(save_dir, exist_ok=True)

    label_map = {'I_T': 8, 'I_PER': 7, 'B_LOC': 2, 'B_PER': 0, 'B_T': 3, 'B_ORG': 6, 'I_LOC': 4, 'O': 5, 'I_ORG': 1}
    train_set = NERDataset('./data/train.txt', './data/train_TAG.txt', tokenizer, max_len=max_length, label_map=label_map)
    train_loader = DataLoader(train_set, batch_size=batch_size, shuffle=True, num_workers=18)
    val_set = NERDataset('./data/dev.txt', './data/dev_TAG.txt', tokenizer, label_map=label_map, max_len=512)
    val_loader = DataLoader(val_set, batch_size=batch_size, shuffle=False, num_workers=18)    # We MUST use the same label map with Train set!

    optimizer = AdamW([
        {'params': list(model.bert.parameters()) + list(model.fc.parameters()), 'lr': lr},
        {'params': list(model.crf.parameters()), 'lr': lr_crf}
    ]) if args.model_type == "bert_crf" else AdamW(model.parameters(), lr=lr)

    # TensorBoard and Logging Setup
    logging.basicConfig(filename=log_filename, level=logging.INFO)

    print('Model loaded successfully')
    for epoch in range(num_epochs):  # Number of epochs can be adjusted
        epoch_loss = train_epoch(model, train_loader, optimizer, device, epoch, num_epochs, save_dir, save_every)

        # Record Training Results
        logging.info(f'Epoch: {epoch + 1}, Training Loss: {epoch_loss}')  # Log the training loss

        val_loss, val_accuracy = validate(model, val_loader, device, epoch, num_epochs)

        # Record Validation Results
        logging.info(f'Epoch: {epoch + 1}, Validation Loss: {val_loss}, Validation Accuracy: {val_accuracy}')
```

Listing 2: train.py