# 实习报告-第三组

## 项目任务:

自然语言处理(NLP)是人工智能学科的一个重要领域，而信息抽取 (IE)又是 NLP 的一个重点任务。信息抽取包括命名实体识别，关系抽取，事件抽取，时间表达式和槽位填充，命名实体识别是文本高级语义理解的基础任务，也是重点任务。当前命名实体识别一般采用神经网络语言模型 + 循环神经网络 + 条件随机场等算法和技术来实现，我们的项目主要是采取神经网络语言模型+循环神经网络+条件随机场来完成的；

## 项目分工:

**韩明洋:对神经网络语言模型的学习和编程，使用特别是Bert模型，是通过在网上下载的语言模型来直接使用**

**李梓涵:对处理语言的网络和循环神经网络(特别是LSTM和TRANSFORMER)的学习，并且对相关代码编程**

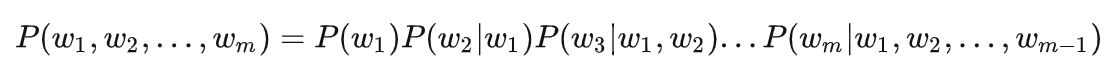
**邓嘉轩**:**学习条件随机场，预测任务的判别模式，条件随机场，对相关的代码的编程**

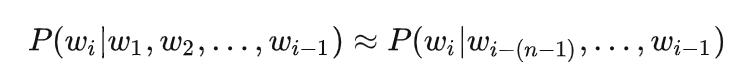
## 项目知识，实现：

## 神经网络语言模型

语言模型是一个单纯的、统一的、抽象的形式系统，语言客观事实经过语言模型的描述，比较适合于电子计算机进行自动处理，因而语言模型对于自然语言的信息处理具有重大的意义。换一句话说，语言模型其实就是看一句话是不是正常人说出来的正常的话。语言模型在信息检索、机器翻译、语音识别中担任着重要的任务。比如机器翻译、语音识别得到若干候选之后，可以利用语言模型挑一个尽量可靠的结果。

一般采用链式计算其概率值



由上式可知，当文本过长时，公式右部从第三项开始的每一项的计算难度都很大。为解决这个问题，有人提出n元模型(n-gram modeI) 以降低该计算难度。所谓n元模型就是在估算条件概率时，忽略距离大于n的上文词的影响，因此的计算可以化简为:

神经网络里面，我们一般情况下选择的是BERT模型

BERT（Bidirectional Encoder Representations from Transformers）是由Google于2018年提出的一种预训练语言模型，它基于Transformer架构，是自然语言处理领域的重大突破之一。BERT在处理自然语言任务时展现出了出色的性能，其先进的能力使得它在多项任务中成为了状态-of-the-art的模型。

BERT的核心思想在于“双向编码”，也就是它在预训练阶段能够同时考虑一个单词的左边和右边上下文信息，这使得它能够更好地理解句子中的语境。在介绍BERT之前，需要了解一下Transformer架构，因为BERT是在此基础上进行的改进，而我们主要就是transformer架构下，我们经过比较，我们选用了word\_to\_vec的代码库，Hugging face是全世界最大的NLP网站，我们里面有bert-base-chinese，我们使用的就是这个

Hugging Face Transformers库: Hugging Face提供了一个名为Transformers的库，其中包含了许多预训练模型，包括中文BERT的变体（例如bert-base-chinese）。您可以使用这个库来加载模型并生成语义向量。

from transformers import BertTokenizer, BertModel

import torch

# 加载中文BERT模型和分词器

tokenizer = BertTokenizer.from\_pretrained("bert-base-chinese")

model = BertModel.from\_pretrained("bert-base-chinese")

# 输入文本

text = "你好，这是一个示例句子。"

# 分词和生成语义向量

inputs = tokenizer(text, return\_tensors="pt")

outputs = model(\*\*inputs)

# 获取语义向量

semantic\_vector = outputs.last\_hidden\_state.mean(dim=1)

这个就是我们生成语义向量的形式

我们将我们的bert进行向量化，这种向量化有一定的意义，比如如果我们做男->女和老公->老婆的向量，我们会发现他们有一定的相似性

## 循环神经网络

循环神经网络 (RNN) 是一种使用序列数据或时序数据的人工神经网络。 这些深度学习算法常用于顺序或时间问题，如语言翻译、自然语言处理 (nlp)、语音识别、图像字幕等；它们包含在一些流行的应用中，比如 Siri、语音搜索和 Google Translate。 与前馈神经网络和卷积神经网络 (CNN) 一样，循环神经网络利用训练数据进行学习。 区别在于“记忆”，因为它从先前的输入中获取信息，以影响当前的输入和输出。 虽然传统的深度神经网络假设输入和输出相互独立的，但循环神经网络的输出依赖于序列中先前的元素。 尽管未来的活动也可能有助于确定特定序列的输出，但是单向循环神经网络无法在预测中说明这些事件。

循环神经网络的另一个显著特征是它们在每个网络层中共享参数。 虽然前馈网络的每个节点都有不同的权重，但循环神经网络在每个网络层都共享相同的权重参数。 尽管如此，这些权重仍可通过反向传播和梯度下降过程进行调整，以促进强化学习。

### 2. LSTM

LSTM（长短期记忆网络，Long Short-Term Memory）是一种特殊类型的循环神经网络（RNN），在处理序列数据时具有强大的建模能力。LSTM的设计目标是解决传统RNN中的长期依赖问题，即在处理长序列时，传统RNN模型的梯度消失或梯度爆炸问题。

LSTM的关键思想是引入了一种称为"记忆单元"（memory cell）的结构，该结构可以在网络的不同时间步保持和更新信息。记忆单元允许LSTM网络选择性地遗忘或存储信息，从而更好地捕捉序列中的长期依赖关系。

下面是LSTM中的关键组件：

* 输入门（Input Gate）：决定是否将当前输入纳入到记忆中。它使用一个sigmoid激活函数来生成一个0到1之间的值，表示保留多少当前输入的信息。
* 遗忘门（Forget Gate）：决定是否丢弃先前的记忆。它使用一个sigmoid激活函数来生成一个0到1之间的值，表示保留多少先前的记忆。
* 输出门（Output Gate）：决定当前记忆单元的输出。它使用一个sigmoid激活函数来生成一个0到1之间的值，表示输出多少记忆。
* 记忆单元（Memory Cell）：用于存储和更新记忆的信息。它通过输入门和遗忘门来决定保留和丢弃哪些信息，并通过输出门来生成输出。

LSTM的计算过程可以描述如下：

* 根据当前输入和前一个时刻的隐藏状态，计算输入门的激活值。
* 根据当前输入和前一个时刻的隐藏状态，计算遗忘门的激活值。
* 根据当前输入和前一个时刻的隐藏状态，计算输出门的激活值。
* 根据当前输入和前一个时刻的隐藏状态，计算新的记忆单元值。
* 根据输入门、遗忘门和记忆单元的值，计算当前时刻的隐藏状态。
* 根据输出门和当前时刻的隐藏状态，计算当前时刻的输出。

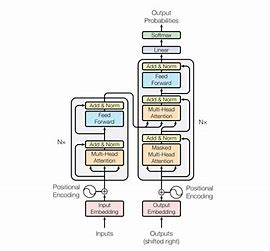
通过引入遗忘门、输入门、细胞状态和输出门，LSTM网络能够根据输入数据的特征和上下文信息，有选择地遗忘、记忆和更新信息。这种门控机制使得LSTM能够更好地捕捉长期依赖关系，避免了传统RNN中梯度消失和梯度爆炸的问题。它在许多序列建模任务中取得了很好的效果，如语言建模、机器翻译、语音识别等。

3. Transformer

Transformer是一种基于自注意力机制（self-attention）的神经网络结构，用于处理序列到序列（sequence-to-sequence）的任务，如机器翻译、文本摘要等。相对于传统的循环神经网络（RNN）和卷积神经网络（CNN），Transformer在处理长序列数据时具有更好的并行性和捕捉长期依赖关系的能力。

Transformer由以下几个主要组成部分组成：

* **编码器（Encoder）**：编码器是Transformer的一个核心组件，负责将输入序列转换为一系列高级表示。编码器由多个相同的层堆叠而成，每个层都包含两个子层：多头自注意力机制和前馈神经网络。自注意力机制用于捕捉输入序列中的上下文关系，前馈神经网络用于对每个位置的表示进行非线性变换。编码器的输出是一个经过多次处理后的输入序列的表示。
* **解码器（Decoder）**：解码器也是Transformer的核心组件，负责将编码器输出的表示转换为目标序列。解码器也由多个相同的层堆叠而成，每个层包含三个子层：多头自注意力机制、多头编码器-解码器注意力机制和前馈神经网络。解码器通过自注意力机制来关注自身的上下文信息，通过编码器-解码器注意力机制来关注输入序列的上下文信息。解码器的输出是目标序列的表示。
* **自注意力机制（Self-Attention）**：自注意力机制是Transformer的关键组成部分之一。它允许模型在生成输出的过程中，根据输入序列中不同位置的上下文信息进行加权聚合。自注意力机制通过计算每个位置与其他位置之间的注意力权重，从而捕捉到输入序列中的长距离依赖关系。
* **多头注意力机制（Multi-Head Attention）**：多头注意力机制进一步提升了模型的表示能力。它由多个平行的注意力头组成，每个头都可以关注输入序列不同位置的上下文信息。多头注意力机制能够捕捉到不同层次和不同方面的依赖关系，提供了更丰富的表达能力。
* **前馈神经网络（Feed-Forward Network）**：前馈神经网络是Transformer中的一个子层，用于对每个位置的表示进行非线性变换。它由两个全连接层组成，中间经过一个激活函数（通常是ReLU）进行处理。前馈神经网络能够引入非线性变换，增强模型的表达能力。
* **位置编码（Positional Encoding）**：由于Transformer中没有显式的循环连接，模型需要一种方式来处理序列的顺序信息。位置编码用于对输入序列中不同位置的标记进行编码。位置编码向输入的嵌入表示中添加了位置信息，使得模型能够区分不同位置的单词或符号，并为模型提供序列的顺序信息。
* **残差连接（Residual Connections）**和层归一化（Layer Normalization）：为了防止模型在处理深层网络时出现梯度消失或梯度爆炸的问题，Transformer采用了残差连接和层归一化的技术。残差连接允许信息在不同层之间直接传递，保持了信息的流动性。层归一化对每个子层的输入进行归一化处理，有助于稳定训练过程并加速收敛。

Transformer结构的引入在序列到序列任务中取得了重大突破，并在机器翻译等领域取得了显著的性能提升。它的并行计算性质使得可以更好地利用硬件资源，加速训练和推理过程。同时，Transformer还具有较好的泛化能力，可以适用于不同长度和领域的序列数据处理任务。通过自注意力机制、多头注意力机制、位置编码以及残差连接和层归一化等技术，Transformer能够更好地解决长期依赖问题。自注意力机制允许模型直接关注序列中不同位置的上下文信息，捕捉到长距离的依赖关系。多头注意力机制提供了更强大的表达能力，同时关注不同的信息层次。位置编码处理序列的顺序信息，确保模型能够区分不同位置的输入。残差连接和层归一化保证了模型的稳定性和有效性。

## 条件随机场

条件随机场应该和隐马尔可夫模型是一脉相承的

条件随机场通过引入自定义的特征函数，不仅可以表达观测之间的依赖，还可表示当前观测与前后多个状态之间的复杂依赖，可以有效克服HMM模型面临的问题。为了建立一个条件随机场，我们首先要定义一个特征函数集，该函数集内的每个特征函数都以标注序列作为输入，提取的特征作为输出。假设该函数集为：

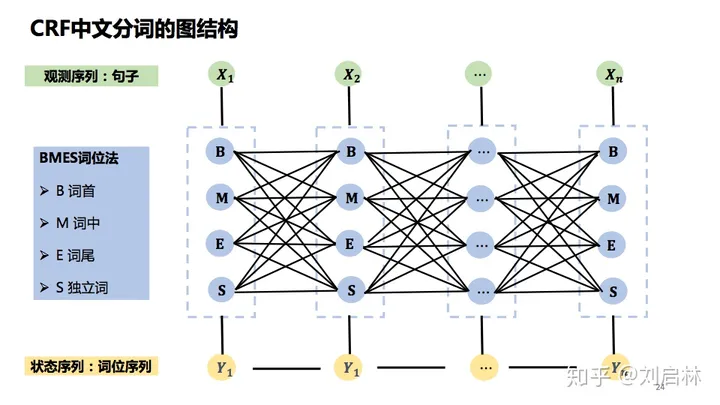


类似与HMM，我们的，马尔可夫模型就是一种序列标注，类似于我们的词性标注，我们所能够观测到的是字组成的序列，观测不到的是我们的每个字的标注，我们使用对数线性模型来计算给定观测序列的条件概率

### log_likehood_crf

这个是其对数似然模型的标志

log_linear_crf

这个是其对数线性模型的答案，我们使用的是skicitlearn\_crfsuite来进行我们的CRF场的编写

## 代码实现

我们根据这三部，我们的代码也基于三部分

### 项目数据

项目数据是基于我们的论文ACL 2018[Chinese NER using Lattice LSTM](https://github.com/jiesutd/LatticeLSTM)中收集的简历数据(https://github.com/jiesutd/LatticeLSTM)

## 2.LSTM

我们的代码主要是基于多模型的处理我们的神经网络，我们尝试了多种模型，最终得到我们的精确度

## Bert的使用

直接使用transformers的

import torch

from transformers import BertTokenizer, BertModel

# 加载预训练的BERT模型和Tokenizer

model\_name = "bert-base-uncased"  # 或者其他适用的BERT模型名称

tokenizer = BertTokenizer.from\_pretrained(model\_name)

model = BertModel.from\_pretrained(model\_name)

# 输入文本

text = "命名实体识别是一项重要的自然语言处理任务。"

# 使用Tokenizer将文本转化为BERT所需的输入格式

inputs = tokenizer(text, return\_tensors="pt", padding=True, truncation=True)

# 推断（前向传播）以获取BERT的输出，这将是语义向量

with torch.no\_grad():

    outputs = model(\*\*inputs)

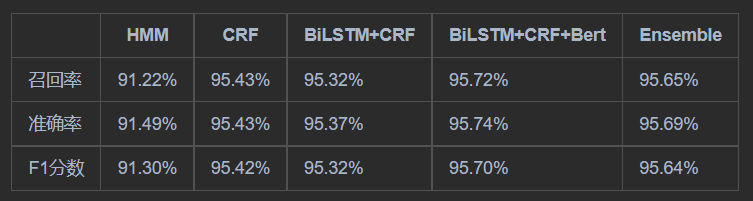
# 获取BERT的输出，即语义向量

semantic\_vector = outputs.last\_hidden\_state.mean(dim=1).squeeze(0)

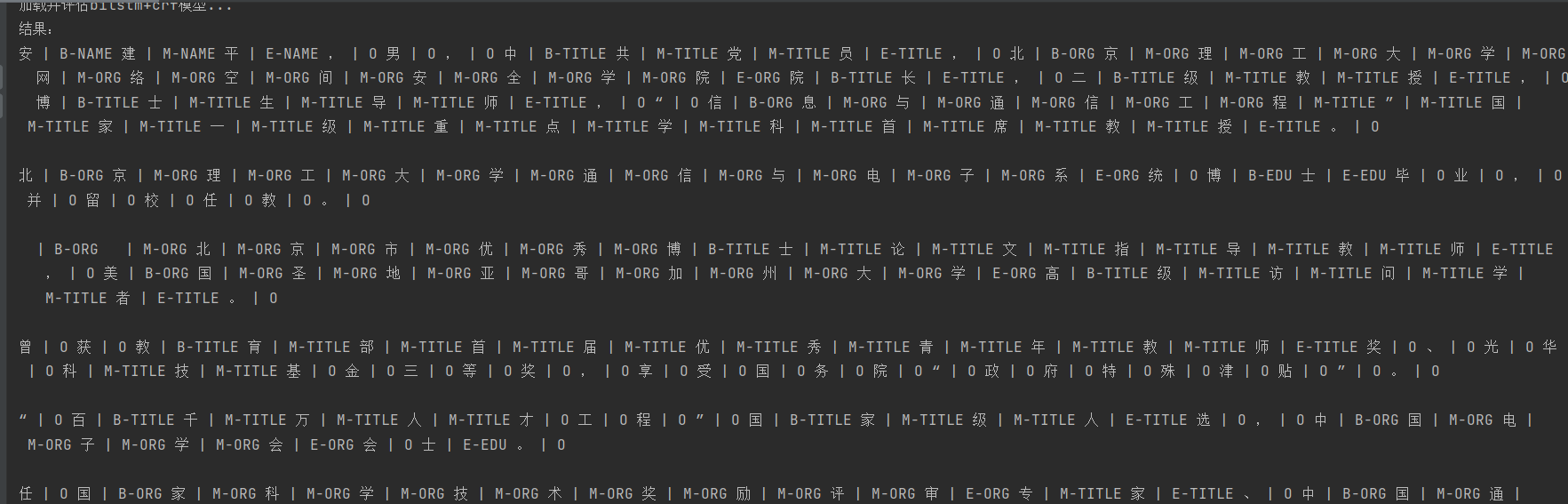
## LSTM && CRF

实际上这一块的编程比较简单，我们直接使用我们的

import torch  
import torch.nn as nn  
from torch.nn.utils.rnn import pad\_packed\_sequence, pack\_padded\_sequence  
class BiLSTM(nn.Module):  
 def \_\_init\_\_(self, vocab\_size, emb\_size, hidden\_size, out\_size):  
 *"""初始化参数：  
 vocab\_size:字典的大小  
 emb\_size:词向量的维数  
 hidden\_size：隐向量的维数  
 out\_size:标注的种类  
 """* super(BiLSTM, self).\_\_init\_\_()  
 self.embedding = nn.Embedding(vocab\_size, emb\_size)  
 self.bilstm = nn.LSTM(emb\_size, hidden\_size,  
 batch\_first=True,  
 bidirectional=True)  
 self.lin = nn.Linear(2\*hidden\_size, out\_size)  
 def forward(self, sents\_tensor, lengths):  
 emb = self.embedding(sents\_tensor) # [B, L, emb\_size]  
 packed = pack\_padded\_sequence(emb, lengths, batch\_first=True)  
 rnn\_out, \_ = self.bilstm(packed)  
 # rnn\_out:[B, L, hidden\_size\*2]  
 rnn\_out, \_ = pad\_packed\_sequence(rnn\_out, batch\_first=True)  
 scores = self.lin(rnn\_out) # [B, L, out\_size]  
 return scores  
 def test(self, sents\_tensor, lengths, \_):  
 *"""第三个参数不会用到，加它是为了与BiLSTM\_CRF保持同样的接口"""* logits = self.forward(sents\_tensor, lengths) # [B, L, out\_size]  
 \_, batch\_tagids = torch.max(logits, dim=2)  
 return batch\_tagids



我们可以看到，我们的模型实际上运行的结果非常的好，我们的模型BiLSTM+CRF+Bert的准确率是最高的



这个是我们的一个测试结果，看上去是很不错的s