

Report of Deep Learning for Natural Language Processing

刘新宇

LXYbhu@buaa.edu.cn

Abstract

在本次报告中，我们选择了ALBERT、ChatGPT和XLNet这三个前沿语言大模型，并使用提示工程的方法对它们在不同自然语言下游任务上的性能进行了检验和对比。

我们选取了情感分析、机器翻译和代码漏洞检测这三个任务进行测试。在情感分析和机器翻译任务中，三个模型都展现出了相对较高的质量。它们能够准确地捕捉到文本的情感，并将源语言文本准确地翻译成目标语言。

在代码漏洞检测任务中，ALBERT表现出最好的效果，而ChatGPT和XLNet虽然能够提示出大部分漏洞的位置，但存在较高的假阳率。

综合来看，这三个前沿语言大模型在不同任务上展现出各自的优势和劣势。通过提示工程的方法，我们可以进一步优化这些模型的性能，并为不同的自然语言处理任务提供更准确、高效的解决方案。

Methodology

1. ALBERT

ALBERT (A Lite BERT) 是一种轻量级的预训练语言模型，由Google Research提出。它是在BERT (Bidirectional Encoder Representations from Transformers) 模型的基础上进行了改进和优化。

与BERT相比，ALBERT的一个主要改进是减少了模型的参数量，同时保持了性能。ALBERT通过共享参数的方式来减少参数量，即将原本在不同层之间独立存在的参数共享起来，使得整个模型更加紧凑。这种共享参数的设计使得ALBERT相对于BERT来说，具有更小的模型体积和更高的训练速度，同时在很多自然语言处理任务上仍然能够达到相近甚至更好的性能。另外，ALBERT还引入了两个新的训练目标，即"Next Sentence Prediction" (NSP) 和"Masked Language

Model" (MLM)。NSP目标与BERT类似，通过预测两个句子之间的关系来提高模型对上下文的理解能力。而MLM目标则是通过在输入句子中随机掩盖一些单词，并要求模型预测这些被掩盖的单词来训练模型的语言建模能力。

ALBERT的应用范围广泛，可以用于各种自然语言处理任务，如文本分类、命名实体识别、语义相似度计算等。它在许多任务上都取得了出色的性能表现，并且相对于其他大型语言模型，具有更小的存储和计算开销，使得它更适合在资源受限的环境下应用。

2. Chatgpt

ChatGPT是一种基于GPT（Generative Pre-trained Transformer）的语言模型，它被设计用于生成自然语言的对话。与传统的GPT模型不同，ChatGPT专注于生成与用户进行连续对话的响应。

ChatGPT的训练过程与GPT类似，它通过大规模的无监督预训练来学习语言的统计规律和语义表示。在预训练阶段，ChatGPT使用了海量的对话数据，通过自回归的方式进行训练，即在生成下一个单词时，模型可以参考前面已生成的单词。这使得ChatGPT能够学习到对话的连贯性和上下文的理解能力。

ChatGPT的优势之一是其生成能力。由于在预训练中接触到了大量的对话数据，ChatGPT可以生成自然流畅的对话回复，并能够根据上下文作出一致性的回应。它可以进行一般性的聊天，回答用户的问题，提供相关信息等。

3. XLNet

XLNet（eXtreme Learning Network）是由Google Research提出的一种预训练语言模型，它基于Transformer-XL架构，并采用了排列语言模型（Permutation Language Model）的训练方法。

XLNet的设计目标是解决自回归偏差的问题。传统的自回归语言模型（如GPT）在生成下一个单词时，只能依赖于已生成的单词，这可能导致生成时的偏差。为了解决这个问题，XLNet采用了排列语言模型的思想，即对输入序列中的所有单词进行排列，并预测单词在排列中的位置，从而使模型能够利用全局的上下文信息。

XLNet的训练过程包括两个阶段：预训练和微调。在预训练阶段，XLNet使用大规模的无监督数据进行训练。与传统的自回归模型不同，XLNet在每个位置

都预测该位置之前和之后的所有单词，而不仅仅是生成下一个单词。这样一来，模型可以充分利用全局的上下文信息，避免了自回归偏差。

在微调阶段，XLNet通过在特定任务上进行有监督的微调来适应特定的下游任务。通过微调，XLNet可以根据任务的要求进行适应性学习，提高在特定任务上的性能。

XLNet在自然语言处理任务中取得了很好的效果。它在多项基准测试中超过了其他模型，并在机器翻译、文本分类、命名实体识别等任务上取得了优异的性能。XLNet的优势主要体现在能够克服自回归偏差，全面利用上下文信息，并且具有较好的泛化能力。

4. 提示工程

提示工程（Prompt Engineering）是指在使用语言模型进行下游任务时，通过合理设计和构造提示文本（Prompt）来引导模型生成期望的输出。它是一种技术手段，旨在改善语言模型在特定任务上的性能表现。

在传统的语言模型中，为了使其适应不同的下游任务，通常需要对输入进行任务特定的修改或调整。而提示工程则提供了一种更加直接和有效的方式来指导模型的行为。

提示文本通常包含了关于任务和期望输出的信息，可以是一句话、几个关键词或特定的问题。通过合理设计提示文本，可以引导模型关注特定的信息、执行特定的操作或产生期望的输出。

提示工程可以在多个方面提升模型性能。首先，它可以帮助模型更好地理解任务的背景和目标，从而准确地生成符合任务要求的输出。其次，通过引入任务相关的信息，提示工程可以提高模型的语义一致性和逻辑性，避免生成不相关或模棱两可的结果。此外，提示工程还可以帮助解决数据稀缺或标注困难的问题，通过引导模型生成特定类型的输出，从而减少对大量标注数据的需求。

在实践中，提示工程的设计需要综合考虑任务的特点、数据的特点和模型的能力。不同任务可能需要不同的提示文本形式和内容，因此需要进行针对性的调试和优化。

Experimental Studies

我们选取了情感分析、机器翻译和代码漏洞检测这三个任务进行测试。

1. 情感分析

问题	ChatGPT	ALBERT	XLNet
对于以下文本，判断其情感是积极还是消极的。	回答	回答	回答
1.这部电影太无聊了，剧情乏味，一点都不吸引人。	消极	消极	消极
2.这个产品非常好，性能出色，简直完美！	积极	积极	积极
3.这个地方真是太美了，景色迷人，令人陶醉。	积极	积极	积极
4.这个地方肮脏不堪，环境恶劣，让人感到不舒服"	消极	消极	消极
5.他的演讲毫无新意，没有任何启发性，真是浪费时间	消极	消极	消极

2. 机器翻译

我们可以选取一些直译也可以意译的中文句子，进行翻译。

任务：翻译这句话为英语：纵然万劫不复，纵然相思入骨，我也待你眉眼如初，岁月如故

ChatGPT: I like you, but just like you.

ALBERT: Even if the world is in ruins, even if love sickness penetrates to the bone, I will still regard you with the same eyes as before, as if time has stood still.

XLNet: Even if everything is destroyed and longing penetrates my bones, I will still treat you as if it's our first encounter, and time stands still.

这里选取的句子是可以由“I like you, but just like you”这句英语句子意译“为纵然万劫不复，纵然相思入骨，我也待你眉眼如初，岁月如故”的，我们发现chatgpt更像是搜索引擎，而ALBERT和XLNet是直接翻译。

3. 代码漏洞检测

在白盒的代码漏洞检测中，我们选取了REVEAL数据集中的3个漏洞代码进行检测，这里使用了我们以前的工作，而对于chatgpt则直接进行对话询问，标红的代码为语言模型认为可能出现错误的地方。

代码1:

```
static void mb_decompose(c, c1, c2, c3) int c, *c1, *c2, *c3; { decomp_T d; if (c >= 0x4b20 && c <= 0xfb4f) { d = decomp_table[c - 0xfb20]; *c1 = d.a; *c2 = d.b; *c3 = d.c; } else { *c1 = c; *c2 = *c3 = 0; }
```

ALBERT:

```
static void mb_decompose(c, c1, c2, c3) int c, *c1, *c2, *c3; { decomp_T d; if (c >= 0x4b20 && c <= 0xfb4f) { d = decomp_table[c - 0xfb20]; *c1 = d.a; *c2 = d.b; *c3 = d.c; } else { *c1 = c; *c2 = *c3 = 0; }
```

XLNet:

```
static void mb_decompose(c, c1, c2, c3) int c, *c1, *c2, *c3; { decomp_T d; if (c >= 0x4b20 && c <= 0xfb4f) { d = decomp_table[c - 0xfb20]; *c1 = d.a; *c2 = d.b; *c3 = d.c; } else { *c1 = c; *c2 = *c3 = 0; }
```

Chatgpt:

在给定的代码中，有一个潜在的问题是 `decomp_table` 的索引范围。根据代码逻辑，`c` 的取值范围应该是从 `0x4b20` 到 `0xfb4f`。然而，在条件判断中，将 `c` 与 `0xfb20` 进行比较，这可能是一个拼写错误，应该是 `0x4b20` 而不是 `0xfb20`。这个错误导致了对 `decomp_table` 的访问可能会超出其索引范围。

为了修复这个问题，可以将条件判断的范围修改为 `c >= 0x4b20 && c <= 0xfb4f`，以确保在正确的范围内访问 `decomp_table`。

修复后的代码如下所示：

```
c
static void mb_decompose(c, c1, c2, c3)
{
    int c, *c1, *c2, *c3;
    decomp_T d;

    if (c >= 0x4b20 && c <= 0xfb4f)
    {
        d = decomp_table[c - 0x4b20];
        *c1 = d.a;
        *c2 = d.b;
        *c3 = d.c;
    }
    else
    {
        *c1 = c;
        *c2 = *c3 = 0;
    }
}
```

修复后的代码将确保 `decomp_table` 的索引范围正确，并避免了可能的数组越界访问问题。

代码2: { static char_u buf[MB_MAXBYTES + 1]; int n, m = 0; char_u *str = *pp; /* Must translate K_SPECIAL KS_SPECIAL KE_FILLER to K_SPECIAL and CSI * KS_EXTRA KE_CSI to CSI. */ for (n = 0; str[n] != NUL && m <= MB_MAXBYTES; ++n) { if (str[n] == K_SPECIAL && str[n + 1] == KS_SPECIAL && str[n + 2] == KE_FILLER) { buf[m++] = K_SPECIAL; n += 2; } else if ((str[n] == K_SPECIAL # ifdef FEAT_GUI || str[n] == CSI # endif) && str[n + 1] == KS_EXTRA && str[n + 2] == (int)KE_CSI) { buf[m++] = CSI; n += 2; } else if (str[n] == K_SPECIAL # ifdef FEAT_GUI || str[n] == CSI # endif) break; /* a special key can't be a multibyte char */ else buf[m++] = str[n]; buf[m] = NUL; /* Return a multi-byte character if it's found. An illegal sequence * will result in a 1 here. */ if ((*mb_ptr2len)(buf) > 1) { *pp = str + n + 1; return buf; } } return NULL; }

ALBERT:

```
{ static char_u buf[MB_MAXBYTES + 1]; int n, m = 0; char_u *str = *pp; /* Must translate K_SPECIAL KS_SPECIAL KE_FILLER to K_SPECIAL and CSI * KS_EXTRA KE_CSI to CSI. */ for (n = 0; str[n] != NUL && m <= MB_MAXBYTES; ++n) { if (str[n] == K_SPECIAL && str[n + 1] == KS_SPECIAL && str[n + 2] == KE_FILLER) { buf[m++] = K_SPECIAL; n += 2; } else if ((str[n] == K_SPECIAL # ifdef FEAT_GUI || str[n] == CSI # endif) && str[n + 1] == KS_EXTRA && str[n + 2] == (int)KE_CSI) { buf[m++] = CSI; n += 2; } else if (str[n] == K_SPECIAL # ifdef FEAT_GUI || str[n] == CSI # endif) break; /* a special key can't be a multibyte char */ else buf[m++] = str[n]; buf[m] = NUL; /* Return a multi-byte character if it's found. An illegal sequence * will result in a 1 here. */ if ((*mb_ptr2len)(buf) > 1) { *pp = str + n + 1; return buf; } } return NULL; }
```

XLNet:

```
{ static char_u buf[MB_MAXBYTES + 1]; int n, m = 0; char_u *str = *pp; /* Must translate K_SPECIAL KS_SPECIAL
KE_FILLER to K_SPECIAL and CSI * KS_EXTRA KE_CSI to CSI. */ for (n = 0; str[n] != NUL && m <= MB_MAXBYTES; ++n) {
if (str[n] == K_SPECIAL && str[n + 1] == KS_SPECIAL && str[n + 2] == KE_FILLER) { buf[m++] = K_SPECIAL; n += 2; }
else if ((str[n] == K_SPECIAL # ifdef FEAT_GUI || str[n] == CSI # endif) && str[n + 1] == KS_EXTRA && str[n + 2] ==
(int)KE_CSI) { buf[m++] = CSI; n += 2; } else if (str[n] == K_SPECIAL # ifdef FEAT_GUI || str[n] == CSI # endif) break; /* a
special key can't be a multibyte char */ else buf[m++] = str[n]; buf[m] = NUL; /* Return a multi-byte character if it's
found. An illegal sequence * will result in a 1 here. */ if ((*mb_ptr2len)(buf) > 1) { *pp = str + n + 1; return buf; } } return
NULL; }
```

Chatgpt:

这段代码存在一些潜在问题：

1. 可能存在缺少必要的头文件。根据代码中的函数调用和类型引用，可能需要包含一些头文件，如 `<vim.h>` 或其他相关的头文件。确保所有必要的头文件被正确包含。
2. `mb_ptr2len` 函数未定义。在代码的最后，有一个函数调用 `(*mb_ptr2len)(buf)`，但是在给定的代码中并没有定义这个函数。确保 `mb_ptr2len` 函数被正确定义，并且在当前代码中可见。
3. 可能存在未声明的类型。在代码中，使用了 `decomp_T` 类型，但在给定的代码中未提供 `decomp_T` 的定义。确保 `decomp_T` 类型被正确定义，并在当前代码中可见。
4. 可能存在潜在的数组越界。在循环内部，对 `buf` 数组进行赋值操作，但没有检查 `m` 的值是否超出了 `buf` 数组的范围。确保 `m` 的值不会超过 `MB_MAXBYTES`，以避免数组越界。

这些问题需要根据代码的上下文和完整代码的环境来进行具体的修复和调整。以上是对给定代码的初步检查和可能存在的问题的分析。修复这些问题的具体方法可能涉及到更多的代码和环境上下文，建议继续查看完整的代码，并结合相关的头文件、函数定义和类型定义进行修复。

代码3:

```
static int utf_convert(a, table, tableSize) int a; convertStruct table[]; int tableSize; { int start, mid, end; /* indices into
table */ start = 0; end = tableSize / sizeof(convertStruct); while (start < end) { /* need to search further */ mid = (end +
start) / 2; if (table[mid].rangeEnd < a) start = mid + 1; else end = mid; } if (table[start].rangeStart <= a && a <=
table[start].rangeEnd && (a - table[start].rangeStart) % table[start].step == 0) return (a + table[start].offset); else return
a; }
```

ALBERT:

```
static int utf_convert(a, table, tableSize) int a; convertStruct table[]; int tableSize; { int start, mid, end; /* indices into
table */ start = 0; end = tableSize / sizeof(convertStruct); while (start < end) { /* need to search further */ mid = (end +
start) / 2; if (table[mid].rangeEnd < a) start = mid + 1; else end = mid; } if (table[start].rangeStart <= a && a <=
table[start].rangeEnd && (a - table[start].rangeStart) % table[start].step == 0) return (a + table[start].offset); else return
a; }
```

XLNet:

```
static int utf_convert(a, table, tableSize) int a; convertStruct table[]; int tableSize; { int start, mid, end; /* indices into
table */ start = 0; end = tableSize / sizeof(convertStruct); while (start < end) { /* need to search further */ mid = (end +
start) / 2; if (table[mid].rangeEnd < a) start = mid + 1; else end = mid; } if (table[start].rangeStart <= a && a <=
```

```
table[start].rangeEnd && (a - table[start].rangeStart) % table[start].step == 0) return (a + table[start].offset); else return a; }
```

Chatgpt:

代码中存在一些问题和改进的空间:

1. 函数返回类型声明缺少 `void`。函数的返回类型应该是 `void`，而不是 `int`。修复该问题，将函数的返回类型声明为 `void`。
2. 循环条件不一致。在 `while` 循环的条件中，使用了 `<` 进行比较，但在计算 `end` 的值时使用了 `/ sizeof(convertStruct)`，这可能导致循环条件不准确。应该确保循环条件的一致性，修复该问题。
3. 二分查找的算法实现。在二分查找部分的实现中，可以添加一些错误处理和边界条件的检查，例如检查 `tableSize` 是否合法，以及 `table` 是否为空等。
4. 变量命名和注释。变量 `start`、`mid` 和 `end` 的命名可以更加具有描述性，以提高代码的可读性。同时，添加适当的注释来解释变量和算法的用途。

修复后的代码如下所示:

```
c Copy code

static void utf_convert(a, table, tableSize)
{
    int a;
    convertStruct table[];
    int tableSize;

    int startIndex, midIndex, endIndex; /* indices into table */

    startIndex = 0;
    endIndex = tableSize / sizeof(convertStruct);
    while (startIndex < endIndex)
    {
        /* need to search further */
        midIndex = (endIndex + startIndex) / 2;
        if (table[midIndex].rangeEnd < a)
            startIndex = midIndex + 1;
        else
            endIndex = midIndex;
    }
    if (table[startIndex].rangeStart <= a && a <= table[startIndex].rangeEnd
        && (a - table[startIndex].rangeStart) % table[startIndex].step == 0)
    {
        return (a + table[startIndex].offset);
    }
    else
    {
        return a;
    }
}
```

以上三个代码的问题都是可能造成内核的over-buffer-flow，可以看到三个语言模型都指出了可能的问题，但是chatgpt作为可以对话的语言模型，不仅指出了

代码可能出错的位置，而且给出了修复措施，这一功能远超另外两个语言模型。但是，对于chatgpt和ALNet，虽然可以指出一些可能错误的地方，但是存在较高的假阳率，这是语言模型在进行代码漏洞检测时会普遍发生的问题。其实对于ALBERT来说假阳率依旧是头疼的问题，不过对于上述bug，ALBERT出色地完成了任务。

Conclusion

经过实验和比较，我们得出结论：ALBERT、ChatGPT和XLNet这三个前沿语言大模型在不同任务上展现出各自的优势和劣势。

ALBERT在代码漏洞检测任务上表现出色，能够识别出许多漏洞的位置。它在模型训练中充分利用了词片段重构和下游任务的无监督学习，使得它具备了较强的语义表示能力。

ChatGPT在情感分析和机器翻译任务中展现出较高的质量。它通过预训练和微调的方式，在生成对话和文本生成方面有着优秀的表现。然而，在代码漏洞检测任务中，ChatGPT可能存在较高的假阳率，无法准确地定位漏洞位置。

XLNet在多种任务上都展现出了良好的性能。它通过使用自回归和自掩码的训练方法，能够更好地理解上下文和建模句子的关联性。XLNet在情感分析和机器翻译任务中能够准确地捕捉到语言的情感和翻译意义。

通过提示工程的方法，我们可以进一步优化这些模型的性能。合理的输入提示可以引导模型生成更符合预期的文本。例如，在情感分析任务中，通过指定相关的情感词汇和上下文信息，可以帮助模型更准确地识别情感。在机器翻译任务中，提供源语言和目标语言之间的对齐提示，可以改善翻译的准确性。

因此，正确、明确的输入提示是获得高质量文本生成的关键。通过精心设计和调整输入提示，我们能够引导模型生成更准确、高质量的文本输出。除了模型能力本身，合适的输入提示也是实现成功的文本生成任务的重要组成部分。