人工智能导论项目报告

一、 项目介绍

1. 问题背景

在智能手机上，输入法的预测功能已经成为用户日常生活中的一部分，极大地提高了输入效率。然而，为何在电脑上这一功能相对较少被应用，是我们在项目中尝试解决的问题。通过深入研究和实践，我们希望不仅能够探索在电脑环境中实现输入预测的可能性，还能了解这一技术在不同平台上的应用差异。

2. 选择动机

我们选择这一课题的动机并非仅仅是出于好奇心，更是基于对人机交互的改善和技术创新的追求。通过分析手机输入法，我们希望挖掘其中的潜在机制，并将其应用于电脑环境中。同时，我们也了解到 GPT-2 可以一定程度上实现对于下一个单词预测这一功能。然而，gpt2 仅能预测一个最合适的单词，这不符合用户的实际用途。事实上，若要在电脑上进行输入法的完善，我们不仅需要最精确的单词，更需要多个预测结果，以方便用户自主地选择想输入的单词。此外，这也是对当前电脑输入方式的一种反思，探究是否有可能通过引入预测功能来提高用户体验。

3. 项目目标

我们的项目旨在开发一个能够在电脑上准确预测用户输入的系统，并通过比较手机和电脑输入方式的不同，探索为何输入预测在这两个平台上存在差异。此外，对于输入结果的记忆功能也是我们的，这将使我们的输入法符合用户的个性需求。通过实现这些目标，我们希望不仅能够提高电脑输入的效率，还能够为未来改进人机交互方式提供一些有价值的思考。

二、 相关准备工作

在选择输入预测作为项目主题之后，我们了解了相关领域的先前工作，以更好地了解当前研究的现状和前沿。

1. 分析输入预测原理

许多研究致力于深入分析输入预测的原理，特别是在手机输入法的成功案例中。这些工作涵盖了基于用户历史输入的统计模型、基于深度学习的神经网络模型等方面。通过深入挖掘这些原理，我们能够更好地理解为何手机输入法在准确预测用户下一个单词方面取得了成功。

2. 自然语言处理领域的研究

在自然语言处理领域，有很多研究关注文本生成和语言模型。其中，一些工作探讨了基于上下文的词汇预测，这对于我们的项目有一定的启示。例如，ChatGPT 等大型语言模型在生成文本时能够考虑上下文，这种思想可以被借鉴用于预测下一个单词。

3. 键盘输入方式的改进

一些研究致力于改进传统的键盘输入方式,包括使用布局优化、按键概率模型等方法，以提高用户的输入效率。虽然这些工作主要集中在改进传统的键盘输入，但它们提供了一些关于提高输入效率的启示，也能够为我们所用。

通过对这些相关工作的深入研究，我们能够从已有的知识中吸取经验教训，同时找到适合我们项目的方法和技术。这为我们的项目提供了坚实的理论基础和实践参考。

三、 数据集选取

由于我们的目的是创造一个适用于日常生活的预测输入系统，因此我们在选择数据集时也会注意挑选语言更加贴合生活的文本。最终经过筛选，除去我们在网上搜集到的中英文日常用语，我们还向中文数据集中添加了《追风筝的人》和《平凡的世界》这两本书，向英文数据集中添加了《权力的游戏》。

四、 模型构建

1. 英文部分（Markov Model）

首先我们定义一个 next_word 字典用于存储关键字和它对应的所有可能的下一个预测单词，很明显它的下一个预测单词可能有一个或者多个，故我们用一个新的字典来存储所有可能的预测单词，新字典的key值对应预测单词，value对应其出现频率，由此得到next_word结构：next_word = {}

next_word[key]={next_probability_word:value}，key 指的是我们预测下一个单词的依据，这里我们选取的是当前光标位置的前两个或前一个单词，next_probability_word 指的是可能的预测结果，value 指的是该结果在训练样例中出现的次数，用来代表它可能出现的概率。

然后根据训练样例初步扩展字典，初次扩展字典时我们采用的结构是next_word[key]=[next_probability_word]，即将所有可能的预测结果直接添加到一个列表里。

```
def expand_dictionary(dictionary,key,value):
    if key not in dictionary:
        dictionary[key]=[]
    dictionary[key].append(value)
```

这里我们不难看出一个 key 对应的预测列表中很可能会有重复的预测结果，也就是该预测结果在样例中出现次数不止一次，此时需要进行合并，并根据它的出现频率进行从大到小的排序，以便更加满足使用者预测的需求。故我们用 union(words)进行合并操作，words 指的是 key 对应的预测链表，合并后期望的结果是实现我们原先所设计的 next_word 结构，即 next_word[key]={next_probability_word:value}。

```
def union(words):
    probability_dict={}
    for item in words:
        probability_dict[item]=probability_dict.get(item,0)+1
    #根据出现概率进行从大到小的排序
    sorted_items = sorted(probability_dict.items(), key=lambda x: x[1], reverse=True)
    sorted_dict = OrderedDict(sorted_items)
    return sorted_dict
```

然后将上述过程整合成马尔可夫模型，通过传入的 filename 训练样例即可建立起数据库。

```
def train_Markov_Model(filename):
    for line in open(filename, 'r', encoding='utf-8'):
        tokens = line.lower().rstrip().split()
        tokens_length=len(tokens)
        for i in range(tokens_length):
```

```
            if i==0:
                continue
            elif i == 1:
                expand_dictionary(next_word, tokens[i-1], tokens[i])
            else:
                expand_dictionary(next_word,tokens[i-1],tokens[i])
                expand_dictionary(next_word,(tokens[i-2],tokens[i-1]),tokens[i])
```

最后我们用 next_words(x)函数来作为我们获取预测结果的外部接口，x 为预测依据，我们直接用 next_word.get(x)即可获得 x 对应的预测结果的字典，返回字典的 keys 即为所有预测可能。

```
def next_words(x):
    ans = next_word.get(x)
    if (ans is not None):
        return list(ans.keys())
    else:
        return ''
```

同时，为了避免每一次都需要重新构建模型，我们通过计算文件的哈希值可以用来快递判断文件是否发生改变，如果发生改变才会重新构建模型，否则就直接读取已经存储的模型数据，这样可以减少每次程序启动所需要的时间。

```
def calculate_file_hash(filename):
    # 使用 SHA-256 哈希算法计算文件内容的哈希值
    sha256 = hashlib.sha256()
    with open(filename, 'rb') as file:
        for block in iter(lambda: file.read(4096), b""):    # 逐块更新哈希值
            sha256.update(block)
    return sha256.hexdigest()
def has_data_changed(data_filename, model_filename):
    data_hash = calculate_file_hash(data_filename)
    # 检查模型文件的存在性
    if not os.path.exists(model_filename):
        return True  # 模型文件不存在，可能是第一次运行
    # 读取保存的哈希值
    if os.path.exists(model_filename + '.hash'):
        with open(model_filename + '.hash', 'r') as hash_file:
            saved_hash = hash_file.read()
        # 比较计算得到的哈希值和保存的哈希值
        return data_hash != saved_hash
    return True  # 没有保存的哈希值文件，认为数据已修改
```

2. 中文部分（LSTM）

在中文的下一个单词预测部分，我们小组采用了训练 LSTM 模型预测。LSTM 模型与英文部分的马尔科夫链相比，尽管耗时更长，但是由于其具有长时记忆（相比 RNN 的短时记忆）的优点，只需训练一次就可以运用于后续的单词预测当中。在中文部分我们分为以下的步骤来完成。

1.数据读取及数据预处理

（1）数据读取和处理

打开 Chinese_data.txt 文件并读取文本。先将 text 变量转换为列表，然后用 numpy 中的 unique 函数提取 words 中存在的不重复元素（即汉字）到新列表 existing_words 中，并为列表中每一个元素配一个键值组成字典。

```
import numpy as np
text = open('Chinese_data.txt', 'r', encoding='utf-8').read().lower()
words = list(text)
existing_words = np.unique(words)  # 获取 words 列表中的存在（不重复）元素
existing_word_index = dict((c, i) for i, c in enumerate(existing_words))  # 为列表中每个元素配一个值
```

设定模型目标是根据前五个字来预测下一个字。建立两个列表，用 append 函数从 words 列表中提取出一个子列表，该子列表包含当前单词的前 5 个字符，并添加到 prev_words 列表中。同理从 words 列表中提取出下一个单词（即当前 5 个单词后面的 1 个单词），并将这个单词添加到 next_words 列表中。

```
WORD_LENGTH = 5  # 获取预测的字符串长度
previous_word = [ ]
next_word = [ ]
for i in range(len(words) - WORD_LENGTH):
    previous_word.append(words[i:i + WORD_LENGTH])
    next_word.append(words[i + WORD_LENGTH])
```

（2）数据集划分和向量化

为了更好的帮助模型找到 X（输入特征）到 Y（标签）的映射关系，应使用多维数组来表示 X，Y，这里我们选择了以三维数组来处理获取的文本，并对其进行一系列数学转化，将文字转化为数字的形式存储。我们建立了三维数组 X，并通过先前字串列表长度、字串长度 5，现有所有字符长度这三个参数确定 X 在每一维的大小，并且设置为布尔类型，通过某个单词在以上三个变量中的位置明确其在 X 中的位置并将其值设置为 1。同理建立二维数组 Y，通过下一个字的列表长度、现有所有字符长度这两个参数确定其大小。

```
X = np.zeros((len(previous_word), WORD_LENGTH, len(existing_words)), dtype=bool)
Y = np.zeros((len(next_word), len(existing_words)), dtype=bool)
for i, each_words in enumerate(previous_word):
    for j, each_word in enumerate(each_words):
        X[i, j, existing_words_index[each_word]] = 1
    Y[i, existing_words_index[next_word[i]]] = 1
```

2. 模型训练

创建顺序模型（一种线性堆叠模型，模型的所有层按照顺序排列）。向模型中添加一个具有 128 个单元的 LSTM 层，input_shape 指定了 LSTM 层的输入形状。继续添加全连接层，将前一层输出的特征映射到新的空间。添加 softmax 激活函数将神经元的输出转换为一个概率分布，确保了所有输出加起来等于 1。添加学习率为 0.01 的 RMSprop 优化器。最后调用 compile 函数配置模型的训练，其中 loss 指定了损失函数，optimizer 指定了优化器，metrics 指定了要在训练过程中追踪的指标 accuracy。

```
model = Sequential()
model.add(LSTM(128, input_shape=(WORD_LENGTH, len(existing_words))))
model.add(Dense(len(existing_words)))
```

```
model.add(Activation('softmax'))
optimizer = RMSprop(learning_rate=0.01)
model.compile(loss='categorical_crossentropy', optimizer=optimizer, metrics=['accuracy'])
model.save('Chinese_LSTM_model.keras')    # 保存模型
```

3. 调用模型进行预测

（1）加载训练好的模型

```
model = load_model('Chinese_LSTM_model.keras')
```

（2）读入输入的句子至函数中并返回预测的汉字列表

```
def next_word(text_, n):
    if text_ == "":
        return "0"
    processed_text = process_input(text_)
    previous_words = model.predict(processed_text, verbose=0)[0]
    next_possible_word = get_words(previous_words, n)
    return [existing_words[idx] for idx in next_possible_word]
```

（3）处理输入的句子

创建一个三维数组，取句子中靠后的 5 个汉字（如果句子长度小于 5 就取所有汉字），根据它在字串中的位置和在现有所有字符列表中的位置来确定数组取 1 的位置。

```
def process_input(text_):
    processed_text = np.zeros((1, WORD_LENGTH, len(existing_words)))   # 三维数组
    temp_text = list(text_)
    text_len = len(temp_text)
    subtract = WORD_LENGTH - text_len
    for t, word in enumerate(temp_text):   # enumerate 函数为每个单词赋予一个索引（从 0 开始）
        processed_text[0, t + subtract, existing_word_index[word]] = 1
    return processed_text
```

（4）预测单词的函数

用 heapq.nlargest 函数找出 previous_words 数组中最大的 n_max 个元素

```
def get_words(previous_words, n_max):
    previous_words = np.asarray(previous_words).astype('float64')   # 转换为 NumPy
    previous_words = np.log(previous_words)   # 对数转换
    return heapq.nlargest(n_max, range(len(previous_words)), previous_words.take)
```

3. 记忆功能

我们进一步引入了模型的记忆功能，通过追踪每次用户的输入内容并将其纳入模型的训练数据中，以使得模型能够逐渐适应用户的个性化需求。这种记忆机制不仅丰富了模型的训练数据，也为模型提供了对用户输入模式的更深层次理解，从而提高了预测结果的个性化和准确性。

```
next_word = load _model(model_filename)    # 加载模型
train _Model(input_filename) # 添加输入数据集
save _model(model_filename)
```

五、 模型分析

为了发现模型可能存在的问题或瓶颈，了解模型在当前数据集上的表现，帮助评估模型的训练情况和表现，可视化地了解损失值和准确率的变化以判断模型是否有效地学习数据，并在训练集上取得进展，以及确定下一步改进模型或实验设计的方案，我们可视化了中文部分模型地训练过程，训练模型，并将模型保存在 history 文件之中。

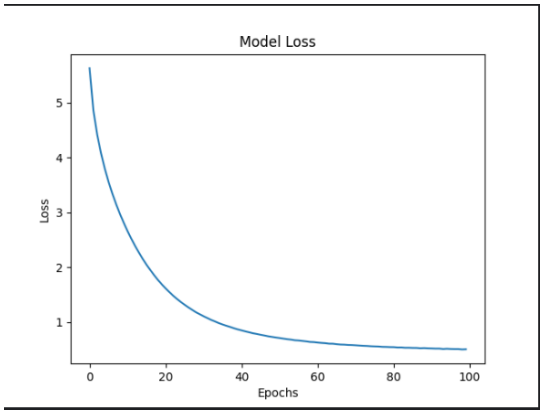
```
history = model.fit(X, Y, validation_split=0.05, batch_size=128, epochs=100,
shuffle=True).history
```

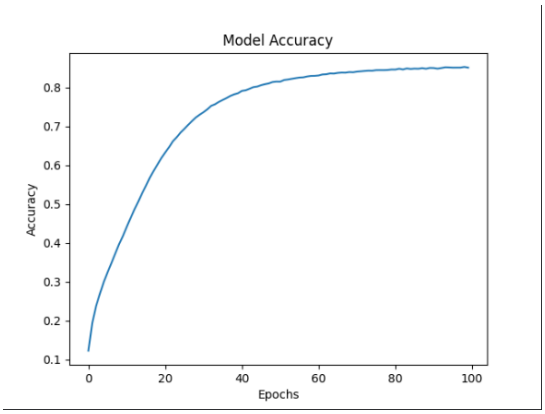
使用 Matplotlib 库绘制损失值随着训练周期变化的折线图。从 history.history['loss']中，取训练过程中记录的损失值数据，并将其在图表中展示。其中图表的 x 轴为训练周期（epochs），y 轴为损失值（Loss）。

```
plt.plot(history.history['loss'])
plt.title('Model Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.show()
```

使用 history.history['accuracy']获取训练过程中记录的准确率数据，利用 Matplotlib 库绘制准确率随着训练周期变化的折线图。其中图表的 x 轴为训练周期（Epochs），y 轴为准确率（Accuracy）

```
plt.plot(history.history['accuracy'])
plt.title('Model Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.show()
```





分析损失值图表我们可以发现：训练周期从 0 到 100，损失值是否随着训练周期（Epochs）的增加而逐渐下降并逐渐趋向于 0，这种下降趋势通常表示模型正在有效地学习数据特征；此外，当训练周期超过 70 之后，损失值在后续训练周期中保持稳定，说明暗示着模型已经接近收敛，这意味着模型可能已经学会了数据的表示并且参数已经收敛到一个较优的状态。

从准确度图表中我们可以分析出，在训练周期逐渐递增的过程中，模型的准确度随着 Epochs 的增加而先迅速陡增再缓慢增加，总体上而言模型的准确度呈现上升趋势，这种上升趋势通常表示表明模型正在从数据中学到更多信息，在学习过程中逐渐提高对数据的预测能力。

在 Epochs 大于 80 的训练后期，准确率逐渐会趋于稳定，表明模型的性能在测试集上达

到了接近收敛于稳定的水平，并且较难有有显著的改进。当 Epochs 训练接近尾声时，模型的准确度已经高达接近 90%，这充分表明了此时的训练模型已经获得了一个比较好的训练效果，但是训练后期验证准确率难以继续提高突破 90%，可能也表示模型可能在数据质量、数据量、模型复杂度、优化策略以及特征选择等方面存在一定的问题，尚有改进提升的空间。

六、 UI

为了使输入输出更加直观，同时也使项目完成度更高，我们在模型构建完成之后制作了相应的 UI 界面。在构建 UI 界面时，我们长时间受制于使用 Shift 键切换中英文的问题。由于 QT 仅能读取键盘事件，包括 Shift 键的按下和释放。然而，由于 Shift 键与其他按键绑定为组合键，这可能导致在使用组合键时误触发中英文转换。最后，我们通过判断字符串长度的方法解决了这一问题。具体而言，通过比较 Shift 键按下前和释放后输入字符串的长度变化，我们能够准确判断 Shift 键是否被按下，以避免误判中英文切换。

```
def keyPressEvent(self, event):
    # 检测 Shift 键是否被按下
    if event.key() == Qt.Key_Shift:
        self.shift_pressed_lens = len(self.text_edit.toPlainText())
    # 检测 Ctrl 键是否被按下
    if event.modifiers() & Qt.ControlModifier:
        self.suggest_on_off()
def keyReleaseEvent(self, event):
    # 检测同时按下 Shift 和其他键
    if event.key() == Qt.Key_Shift:
        if len(self.text_edit.toPlainText()) == self.shift_pressed_lens:
            self.switch_language()
```

我们预测的逻辑是根据目前光标位置之前的文本，所以在这里我们首先需要获取了光标的位置，通过以下函数即可实现。

```
def get_cursor_position(self):
    cursor = self.text_edit.textCursor()
    self.cursor_pos = cursor.position()
```

然后就需要处理光标前的文本，通过引入 re 库用来分别获取光标位置前面的中文和英文，借此也实现了中英文混输。其中，r'[A-Za-z]'代表英文字符，r'[\u4e00-\u9fa5]'代表中文字符。

```
def get_input_used_to_suggest(self):
    if self.current_language == 'ch':
        start = self.cursor_pos - 1
        my_re = re.compile(r'[A-Za-z,."?!]', re.S)
        while start >= 0:
            res = re.findall(my_re, self.current_input[start])
            if len(res):
                break
            start -= 1
    else:
        start = self.cursor_pos - 1
        my_re = re.compile(r'[\u4e00-\u9fa5，。！？、；：“”‘’（）【】]', re.S)
```

```
            while start >= 0:
                res = re.findall(my_re, self.current_input[start])
                if len(res):
                    break
                start -= 1
        return self.current_input[start + 1:self.cursor_pos]
```

同时，我们在点击按钮后发现因为鼠标点按了文本框以外的区域，导致光标在文本框中消失，所以我们在这边专门写了一个函数让光标回到原本在文本框中的位置，这样可以使得操作更加便捷。

```
# 设置光标位置
def set_cursor_position(self, lens):
    # 设置光标位置
    cursor = self.text_edit.textCursor()
    cursor.setPosition(self.cursor_pos + lens)
    self.text_edit.setTextCursor(cursor)

    # 将焦点设置回文本编辑框
    self.text_edit.setFocus()
```

以下则是一些美化的功能，包括按钮点击闪烁，无法点击时变为灰色等，由于篇幅有限就不全部展示。

```
def on_button_click_blink(self, sender):
    # 创建按钮闪烁效果
    animation = QPropertyAnimation(sender, b"styleSheet", self)
    animation.setDuration(20)  # 设置动画持续时间（毫秒）
    # 切换按钮样式表，实现闪烁效果
    animation.setStartValue(sender.styleSheet())
    animation.setEndValue("background-color: yellow;")
    # 在动画完成时，恢复按钮原始样式
    animation.finished.connect(lambda: sender.setStyleSheet(sender.styleSheet()))
    # 启动动画
    animation.start()
```

七、 未来改进方向

在未来的工作中，在对数据集的处理方面，我们希望能够找到更加高效精确的文本处理方法。在处理中文文本时，我们曾经尝试过使用 jieba 的分词器，可以按词语划分中文文本，但是训练后的效果并不是很理想，比如说“我们”，由于使用了 jieba 分词，输入了“我”之后就不可能出现“们”的预测。

同时，我们希望能够进一步优化和深化预测模型。采用更先进的自然语言处理方法，以提高模型对上下文的理解能力，从而更准确地预测用户的下一个输入。

当然，在记忆功能方面也不能止步于简单的将输入内容添加至训练集当中，需要引入更为智能和个性化的学习策略，以更好地适应不同用户的输入习惯和语言风格。通过进一步挖掘用户的输入历史，为每位用户提供更符合其需求的输入预测体验。

在未来，我们也划进一步优化交互界面，提供更直观、更智能的用户体验，以确保用户能够轻松而高效地与预测输入系统进行交互。

参考资料：

[1] 简述马尔可夫链【通俗易懂】
https://blog.csdn.net/a1097304791/article/details/122088595?ops_request_misc=%257B%2522request%255Fid%2522%253A%252217046952301680022898458%2522%252C%2522scm%2522%253A%252220140713.130102334..%2522%257D&request_id=170469523016800222898458&biz_id=0&utm_medium=distribute.pc_search_result.none-task-blog-2~all~top_positive~default-1-122088595-null-null.142^v99^control&utm_term=%E9%A9%AC%E5%B0%94%E7%A7%91%E5%A4%AB%E9%93%BE&spm=1018.2226.3001.4187

[2] （Python 数模）（预测模型一）马尔科夫链预测
https://blog.csdn.net/qq_43920838/article/details/132651461?ops_request_misc=%257B%2522request%255Fid%2522%253A%252217046950351680018651087 9%2522%252C%2522scm%2522%253A%252220140713.130102334..%2522%257D&request_id=170469503516800186510879&biz_id=0&utm_medium=distribute.pc_search_result.none-task-blog-2~all~baidu_landing_v2~default-5-132651461-null-null.142^v99^control&utm_term=python%20%E9%A9%AC%E5%B0%94%E7%A7%91%E5%A4%AB%E9%93%BE&spm=1018.2226.3001.4187

[3] 【全网最好】深度学习——LSTM 模型透彻讲解
https://www.bilibili.com/video/BV1wu411C7FW/?share_source=copy_web&vd_source=400979a8b115ef8a10d82b8f3bd02954

以上为在我们完成项目过程中对我们帮助比较大的几个参考资料，当然在完善过程中我们还通过百度等方式搜索了许多的资料，由于篇幅有限，这里就不一一列举了。