# 语言分析与机器翻译结课报告

**研究背景**

伴随着互联网的发展以及移动终端如手机的普及，人工智能算法研究和相关产业的落地，越来越多的公司逐渐将发展的目标投入到聊天机器人这一领域中，因为聊天机器人可以被定位为各种服务的入口，尤其是为移动端的APP应用以及为可穿戴设备提供各种服务入口。在未来，人们可以通过聊天机器人直接提供各种服务，而不是用使用诸如天气需求、航班预订、用餐预订等独立应用程序。目前已经存在一些落地具有代表性的产品，如苹果的Siri，微软的小冰，三星的Bixby等等。这些聊天机器人的产品都使得用户可以在移动设备上更加快捷高效的获得信息和进行相关的指令操作以此来简化了用户的操作，提供给用户良好的体验。在此基础上，研究者发现智能机器人可以应用到更加广泛的领域，例如问答系统、洽谈协商、电子商务、娱乐闲聊等等。

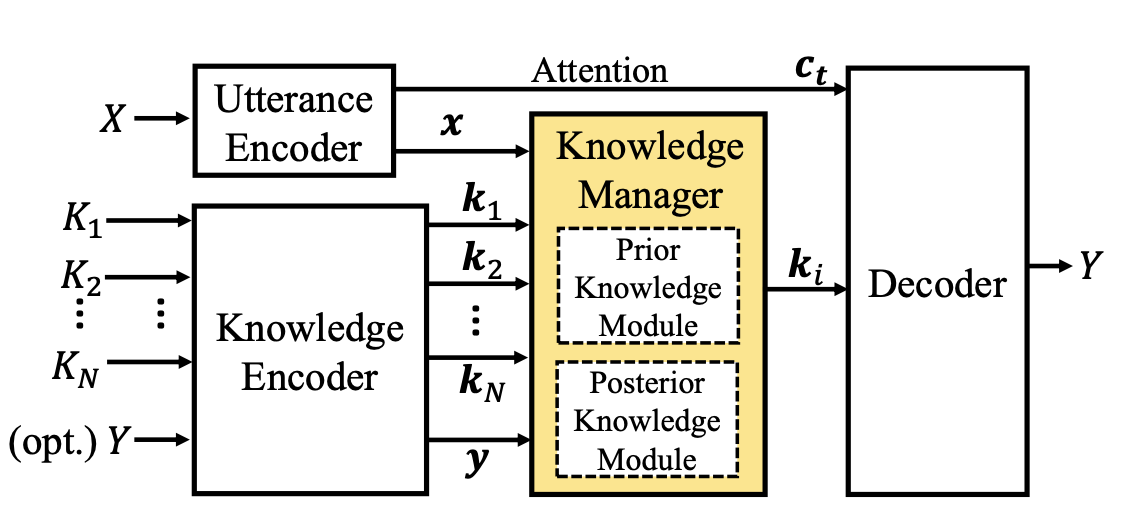
**报告内容**

本模块调用基于PostKS的生成模型，根据用户设定的机器人个性，选择出一个与当前提问相关度最高的个性信息，并将此个性融入到解码器中，用来生成一个具有个性化的回复。

**PostKS模型构建**

本文使用的模型的架构概述如下图所示，它由四个主要组件组成，其主要架构如下：

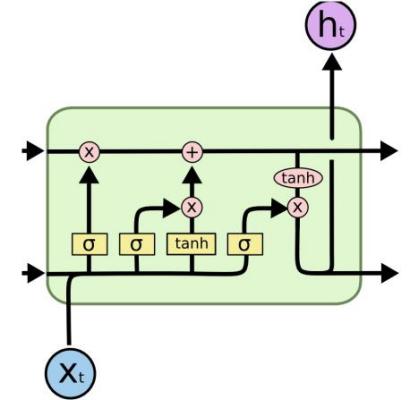
1. 对话编码器：将用户的询问编码成向量，并且将这个向量送入知识管理器。
2. 知识编码器：将每个知识作为输入并将其编码为知识向量。当回复可用时，它还将编码为向量。
3. 知识管理器：由两个子模块组成：先验知识模块与后验知识模块。给定先前编码的和（如果有的话也将作为输入）。知识管理器负责去选择合适的知识库并将其与经过Attention的上下文词向量进入解码器。
4. 解码器：根据选定的知识与基于注意的上下文向量产生回复。



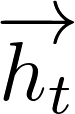
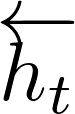
**PostKS**模型图

**编码层**

本文在编码层中，使用一个双向的有门控单元的RNN（GRU）其结构如图所示。



**GRU**模型图

它由两部分组成：前向RNN和后向RNN。对于给定话语，前向RNN从左到右读取话语，然后对于每一个获得一个从左到右的隐藏状态 ，反向RNN从右向左读取话语，并且对于每一个获得一个从右到左的隐藏状态。在时刻，这两个隐藏状态被合并成总的隐藏状态：

获得输入对话的编码向量，我们利用隐藏状态并且定义去获取编码器对于的输出编码向量。这个向量将会被送入知识管理器去优化知识的选择。同时，它将会被作为解码器的初始隐藏状态被解码。

本文的知识编码器与语句编码器拥有者相同的结构，但是他们之间不分享任何的参数。他们的主要区别在功能上，知识编码器编码使用双向RNN，编码每一条知识，并且如果回复已知的情况下也会编码回复，成为向量。这些被编码的向量将会被知识管理器使用。

在word embedding的时候，本文将输入语句中次的序号转化为具有固定大小的词向量，其中表示max\_vocab\_size词向量的最大特征数，即语句中词语的个数,embed\_size表示生成的词嵌入向量的纬度，在本文中设置为300维，padding\_idx表示被掩盖的位置，即这些位置的数据不参与到模型的训练当中。

**词嵌入模块**

|  |
| --- |
| **输入：**问题语句  **输出：**问题语句的词向量表示 |
| enc\_embedder = Embedder(self.src\_vocab\_size,self.embed\_size, self.padding\_idx)  dec\_embedder = Embedder(self.tgt\_vocab\_size,self.embed\_size, self.padding\_idx)  knowledge\_embedder = Embedder(self.tgt\_vocab\_size,self.embed\_size, self.padding\_idx) |

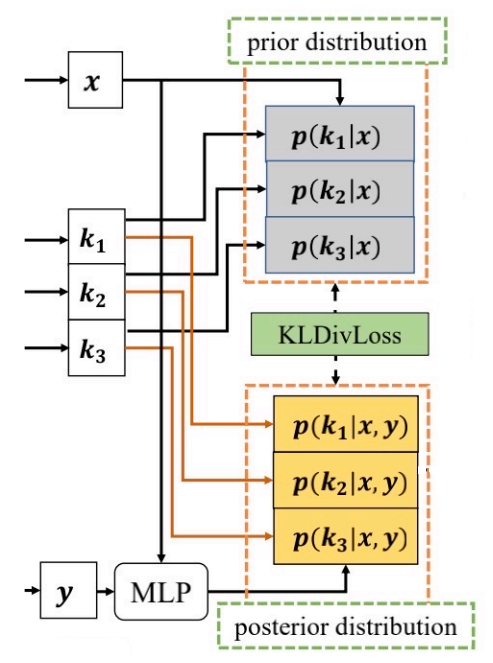
然后，本文构建双向的GRU单元对词向量再次编码，在这里需要对输入编码层和知识编码层的相关参数进行设置，其中hidden\_size表示隐含节点的个数这里的设置为800个隐含节点，enc\_embedder与knowledge\_embedder表示上文所建立的词嵌入模型，通过这两个模型获得单词的词向量。num\_layers表示有几个神经网络层，此项的本人设置为1层，dropout表示每个时间步中对隐含节点的屏蔽比率，在本文中本人设置为0.5，表示对每个时间步中50%的隐含节点进行屏蔽，防止整个训练出来的模型发生过拟合的现象。

**编码器模块**

|  |
| --- |
| **输入：**词向量  **输出：**编码过后的向量 |
| #话语编码器  self.encoder = RNNEncoder(self.embed\_size, self.hidden\_size,enc\_embedder,  self.num\_layers,self.bidirectional, self.dropout)  #知识管理器  self.knowledge\_encoder = RNNEncoder(self.embed\_size,self.hidden\_size,   knowledge\_embedder,self.num\_layers,self.bidirectional,self.dropout) |

**知识管理层**

知道了编码过后的和编码过后的集合，知识管理器的目标是选择一个合适的去和解码器融合生成带有这个知识的回复。当回复已知的时候，模型同样使用知识管理器去获得。知识管理器有两个子模块：先验知识模块与后验知识模块，其结构如下图。



知识管理层

在先验知识模块中，定义一个知识的条件概率分布:

我们使用点乘去计算与输入向量之间的关联关系。一个高的关联关系意味着与有较强的相关性，因此更应该被选择。其中，表示的是只有时候的概率，因为它是先验知识模块，其工作时并不知道之后回复是什么。然而，存在着很多不同的与输入语句相关的知识。所以，根据在训练中的仅仅使用先验分布去简单选择一个知识是十分困难的。

基于此，在后验知识选择模型中，我们定义一个同时考虑输入与回复的后验分布知识的分布：

其中是一个全连接层。与先验分布相比较，后验分布是相对准确的，因为后验分布在训练的过程中使用通过询问和回答共同选择出一个知识的，更能贴合回答的信息。

根据以上定义的分布，使用Gumbel-Softmax的方法去采样知识，因为Gumbel-Softmax方法可以使得不可积分的分布变得可以反向传播，使得模型可以被训练。在训练的过程中，知识是基于后验分布并且在知道真实回复的情况下被选择出来的，因此后验分布更可能通过这个分布去获得合适的知识。在预测阶段，因为期待的回答是未知的，所以后验分布是可不获得的，因此模型选择仅仅只能用过先验分布的数据去选择的个性知识。

知识管理模块

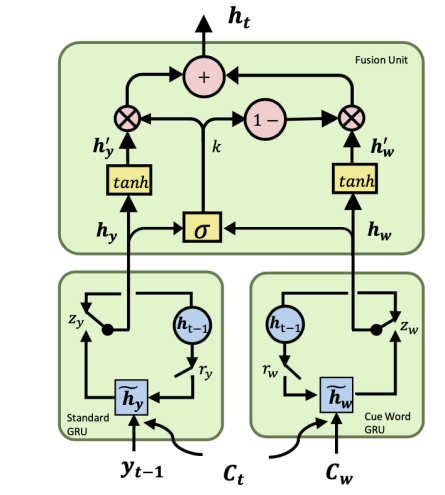
|  |
| --- |
| **输入：**所有机器人的个性  **输出：**选择出来的个性知识 |
| #先验分布  self.prior\_attention = Attention(query\_size=self.hidden\_size,                                     memory\_size=self.hidden\_size,                                     hidden\_size=self.hidden\_size,                                     mode="dot")  #后验分布  self.posterior\_attention = Attention(query\_size=self.hidden\_size,                                         memory\_size=self.hidden\_size,                                          hidden\_size=self.hidden\_size,                                          mode="dot")  #优化两种分布的loss  self.kl\_loss = torch.nn.KLDivLoss(size\_average=True) |

在本文中，为了模型可以在只有先验分布而没有后验分布的情况下，选择出一个适当的个性，使用先验分布去尽可能地估计后验分布。基于这种目的，引进了一种辅助损失函数KLDivLoss，其如下:

这里的表示的是模型的参数。当在减小先验分布与后验分布之间的KLDivLoss的时候，使用后验分布的信息去优化先验分布的信息。这样，在推理阶段，即使在后验分布不知道的情况下，先验分布也可以选择出一种与询问有关联的个性。

**解码层**

本文使用了带有分层选通融合单元的解码器HGFU。HGFU提供了一种将个性整合到响应生成中的方法，它由三个主要组成部分组成，即话语GRU、知识GRU和融合单元，其结构如图所示。



**HGFU**结构图

话语GRU、知识GRU遵循标准GRU结构，其中话语GRU的输入为最后产生的、上一个状态和上下文向量，其作用为记录当前对话的状态，上下文等内容；知识GRU的输入为当前对话选择的个性知识、上一个状态和上下文向量，其作用主要为记录当前个性的关键词信息。

融合单元融合两个GRU生成的隐藏状态去生成一个带有个性化的回复

其中,其中是参数。门控控制着和在最终隐藏状态的比例，以实现他们之间灵活的融合。

在获得隐藏状态之后，下一个单词根据下面的概率分布被产生：

**HGFU**解码器

|  |
| --- |
| **输入：**最后产生的单词; 解码起上一个状态; 上下文向量  **输出：**下一个产生的单词 |
| #话语GRU  self.rnn = nn.GRU(input\_size=self.rnn\_input\_size,hidden\_size=self.hidden\_size,              num\_layers=self.num\_layers,dropout=self.dropout if self.num\_layers > 1 else 0,              batch\_first=True)  #知识GRU  self.cue\_rnn = nn.GRU(input\_size=self.cue\_input\_size,hidden\_size=self.hidden\_size,               num\_layers=self.num\_layers,dropout=self.dropout if self.num\_layers > 1 else 0,               batch\_ first=True)  #获得话语GRU以及知识GRU的隐藏状态  rnn\_output, rnn\_hidden = self.rnn(rnn\_input, hidden)  cue\_output, cue\_hidden = self.cue\_rnn(cue\_input, hidden)  h\_y = self.tanh(self.fc1(rnn\_hidden))  h\_cue = self.tanh(self.fc2(cue\_hidden))  k = self.sigmoid(self.fc3(torch.cat([h\_y, h\_cue], dim=-1)))  new\_hidden = k \* h\_y + (1 - k) \* h\_cue  #获得输出  out\_input\_list.append(new\_hidden.transpose(0, 1))  out\_input = torch.cat(out\_input\_list, dim=-1) |

**模型训练**

在完成PostKS生成模型的构建后，接下来需要读入已经预先处理好的训练集和验证集，使用Corpus中定一个create\_batches函数根据batch\_size读取样本集合即一次训练所训练的样本数。因为数据集庞大外加训练设备的显存有限，在本文中的本人设置为32。

train\_iter = corpus.create\_batches(config.batch\_size, "train", shuffle=True,

device=device)

valid\_iter = corpus.create\_batches(config.batch\_size, "valid", shuffle=False, device=device)

完成PostKS生成模型的构建后，又需要分别使用已经经过分完的样本集合的数据集对模型进行训练，调参。在本文中使用Adam优化器对模型进行优化，设置Adam优化器的初始参数如下：

optimizer = getattr(torch.optim, config.optimizer)(model.parameters(),lr=config.lr)

torch.optim.lr\_scheduler.ReduceLROnPlateau(optimizer=optimizer,

                           factor=config.lr\_decay, patience=1, verbose=True, min\_lr=1e-5)

其中config.lr为Adam优化器的初始学习速率,在本报告中其本人值为0.00005，config,lr\_decay为Adam优化器的学习速度下降速率，优化器的学习速率即反传播算法中权重更新的幅度，学习速率太低，会导致模型训练速度很慢，学习速率太高，有可能在梯度下降过程中跳过了最小值，使训练过程变得发散, 不利于得到最优解。

完成了优化器的配置后，开始建立Trainer来进行模型的学习过程的配置，以及完成整个训练。

   trainer = Trainer(model, optimizer, train\_iter,valid\_iter, logger, generator,

  valid\_metric\_name="-loss",config.num\_epochs,config.save\_dir,config.log\_steps,config.valid\_steps, config.grad\_clip,lr\_scheduler, save\_summary=False)

其中model传入的是已经搭建好的PostKS生成模型用于训练，optimizer是上文中建立的优化器用来优化模型的各项参数。train\_iter与valid\_iter表示是经过处理的已经分好的数据集和验证集，logger是用来记录训练数据的。valid\_metric\_name表示用何种方法去评价模型的好坏，本文使用的是loss这种损失函数。num\_epochs表示模型训练的轮数，本文设定为20轮，其中5轮预先训练知识管理层用来优化个性知识层预测的准确性，15轮训练训练整体的参数即编码器、解码器等等。grad\_clip表示的是梯度剪裁，当梯度特别大的时候，将其投影到一个比较小的尺度上，防止在反向传播过程中的梯度爆炸，这里 的为5。

**实验结果**

完成对模型基本参数的配置后，本文将对PostKS生成模型进行训练和测试。在数据获取与预处理阶段，本文已清洗筛选出质量高且拥有高频个性标签的对话，并进行训练集、测试集、验证集的构建，具体的数据分布如表所示。

本文采用以下指标评价生成模型效果，主要包括：

Acc指的是正确预测的样本数占总预测样本数的比值。Acc的数值越高，模型的表现越好。

PPL是用在自然语言处理领域中，衡量语言模型好坏的指标。它主要是根据每个词来估计一句话出现的概率。PPL越小，模型的效果就越好。

BLEU表示机器翻译译文与参考译文之间的相似度。这些指标通常作为一些机器学习问题的评测标准。BLEU的数值越高，模型的效果越好。

每种数据集的数据量

|  |  |
| --- | --- |
|  | 数据量 |
| 训练集 | 166211 |
| 验证集 | 4383 |
| 测试集 | 4383 |

根据上述模型参数配置，本文将PostKS生成模型进行训练训练。，将模型的训练出来结果与传统的生成聊天机器人中使用的两层Encoder的LSTM+beam search模型进行对比，以此来对PostKS模型的效果来进行客观的对比。PostKS模型的训练结果如表所示。

**PostKS**训练结果

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | epochs | Acc | PPL | BLEU |
| PostKS（预训练） | 5 | 0 | 25906.16 | 00.03 |
| PostKS | 10 | 0.32 | 103.51 | 25.57 |
| PostKS | 15 | 0.36 | 68.39 | 27.62 |
| PostKS | 20 | 0.39 | 47.67 | 28.64 |

在对于PostKS的训练中，前五轮为预训练阶段，只要对于知识管理器以及编码器的参数进行训练，不对解码器进行训练。从结果上来看，在前五轮的训练当中，模型的精确度，困惑度以及BLEU值等评价指标的参数不好。在第6到20轮的训练中，本文对整个模型进行了全面的训练，在前几轮的训练中，准确度显著上升，困惑度大幅下降，BLEU值的参数也大幅上升。在后几轮的训练中，准确度，困惑度，BLEU值等指标趋于平稳，模型达到收敛，模型的性能趋近于稳定。其最终结果取第20轮训练的结果。

在相同的数据集条件下，对两层Encoder的LSTM+beam search模型进行训练，并将其结果与PostKS生成模型进行对比。在相同测试集中所得到的评估结果如表所示。

其中两个评判指标分别为PPL和BLEU值。其中PPL表示预测下一个词时有多少种选择，BLEU值表示机器翻译译文与参考译文之间的相似度。这些指标通常作为一些机器学习问题的评测标准。从两个模型的训练结果上可以看出，PostKS的模型性能强与传统的LSTM模型。对于本数据集，其在相同轮数下可以获得更低的困惑度以及更高的BLEU值。说明，先验分布被PostKS模型很好地利用，并且对选用个性知识产生了有效的指导作用，使得模型的效果更好。可见，PostKS模型相对于基线模型可以有效提高对话模型的性能。

不同模型训练结果对比

|  |  |  |  |
| --- | --- | --- | --- |
|  | epochs | PPL | BLEU |
| LSTM | 10 | 141.13 | 18.97 |
| LSTM | 20 | 84.25 | 21.60 |
| PostKS | 10 | 103.51 | 25.57 |
| PostKS | 20 | 47.67 | 28.64 |

同时，本文也对不同的解码器进行的对比实验，文本使用PostKS的解码方式为软解码（HGFU）的解码方式，将其与传统的硬解码方式,其解码方式如公式4.9进行对比，硬解码与软解码的对比结果如下表所示。

解码器效果对比

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | epochs | Acc | PPL | BLEU |
| PostKS（软解码） | 10 | 0.32 | 103.51 | 25.57 |
| PostKS（软解码） | 15 | 0.36 | 68.39 | 27.62 |
| PostKS（软解码） | 20 | 0.39 | 47.67 | 28.64 |
| PostKS（硬解码） | 10 | 0.25 | 131.61 | 12.29 |
| PostKS（硬解码） | 15 | 0.263 | 93.538 | 14.84 |
| PostKS（硬解码） | 20 | 0.279 | 70.254 | 15.45 |

从表中可以看出，解码器的选择对于模型最终的训练效果起到了至关重要的作用，软解码的准确度、困惑度以及BLEU值等各项指标均好于传统的硬解码方式，说明在个性化生成的解码器中使用HGFU解码的方法对于模型的效果提升明显。对于硬解码来说，其生成的回复大多是无意义的如“是的呢”，“我也是”等相关性不明显的回复。而使用软解码方法生成的回复，则相对多样与回复的相关性更高，语句更为流畅。可以看出，使用HGFU解码器之后，整个模型性能大幅提升。

此外，本文也加入了人工评价的环节，三位评估者给通过硬解码器和软解码器产生的回复来给这两个模型进行打分。评分的角度为回复相关性以及回复流畅性，其分数的范围为0到2，0是完全不相关或者完全不流畅，1是回复的相关性可以接受或者流畅性可以接受，2表示回复是特别相关或者回答是自然且信息完全的。本文随机从4383个回复中随机挑选200个回复在进行人工评分。本文根据三位评估者给的平均分来评估模型的性能，其结果如表。

人工评价数据表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 回复相关度 | 回复流畅性 | PPL | BLEU |
| PostKS（软解码） | 1.04 | 1.33 | 47.67 | 28.64 |
| PostKS（硬解码） | 0.33 | 1.07 | 70.254 | 15.45 |

从表中我们可以看出，对于两个模型来说人工评价的结果与机器评估的效果相一致，基于HGFU解码器生成的模型，更容易生成有意义、相关性强、信息量大的回复如“明天下午来北京的！”，“我的错，被人发现了”，而不像传统的硬解码器那样，趋向于生成单一、死板的万能回复如“我也想去”，“我也是”这一类的回复。可见，HGFU解码器在人工和机器评价上的性能表现都优于传统的硬解码方法。

综上所示，PostKS的表现出众，可以很好地生成带有个性化的、相关度高的、流畅的回复信息。