NLP 真实场景问题研究报告——迁移学习

姓名: 孙华山
学号: 1120192305
专业: 人工智能
指导老师：鉴萍
课程：机器学习实践

2022 年 5 月 3 日

目录

1 实验简介

在真实环境下的自然语言处理问题中，通常具有更多的困难和挑战，如：数据稀缺、类不平衡问题、推理速度要求等等，而这些问题既是实验理论研究的问题，也是实验理论转向工业应用成果所需要解决的问题。本实验基于《语义相似度计算》实验，进一步学习并探究以下两个问题 (1) 语义相似度计算和自然语言推理 (NLI) 间的迁移学习 (Transfer Learning) 问题；(2) 稀缺场景下，语义相似度计算的基于预训练模型的提示学习 (Prompt Learning)

2 实验目标

(1) 通过查阅文献，复习课堂知识，了解并学习真实场景下，神经网络需面对的问题，如：数据稀缺、语义歧义等；同时学习相应的解决方法，如数据增强方法、知识蒸馏、元学习、度量学习等。

(2) 通过在语义相似度计算模型上进行二次预训练用于 NLI 问题，进一步学习并理解迁移学习 (Transfer Learning) 的原理、优缺点和应用场景。

(3) 通过人为构造数据稀缺场景，在语义相似度计算问题上，进行提示学习，进一步学习并掌握提示学习 (Prompt Learning) 的基本原理，相应方法和优缺点以及应用场景。

3 任务描述

3.1 本实验任务

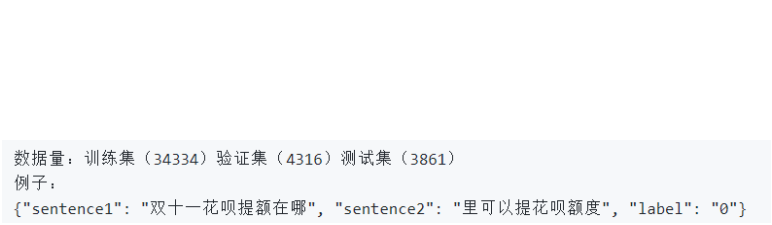
本实验主要有以下两个任务：

(1) ：基于训练好的语义相似度模型，在 NLI 数据集上进行二次预训练，实现迁移学习，并通过分析并总结实验结果（对比试验？迁移和完全自学）。

(2) ：人为构造语义相似度计算任务的数据稀缺问题，并设计，代码实现预训练模型的 prompt 模式，进行提示学习，分析实验结果。

3.2 数据和评价指标描述

数据描述：本实验主要使用了蚂蚁金融语义相似度数据集 (AFQMC) 和原生中文自然语言推理数据集 (OCNLI) 进行相关实验，两类问题均为文本分类问题，数据展示见下如图 ()：



(a) AFQMC 数据集展示

(b) OCNLI 数据集展示

图 1: 实验使用数据集展示

(1) **AFQMC**：‘sentence’ 表示两句话，‘label’ 为对应标签:0：不相似，1：相似；为二分类问题。

(2) **OCNLI**：‘sentence’ 表示两句话，‘label’ 为 neutral，entailment，contradiction 三种类别；为多分类问题。

评价指标：由于是分类问题，因此主要使用准确率和 (Macro-)F1 分数两种评价手段：

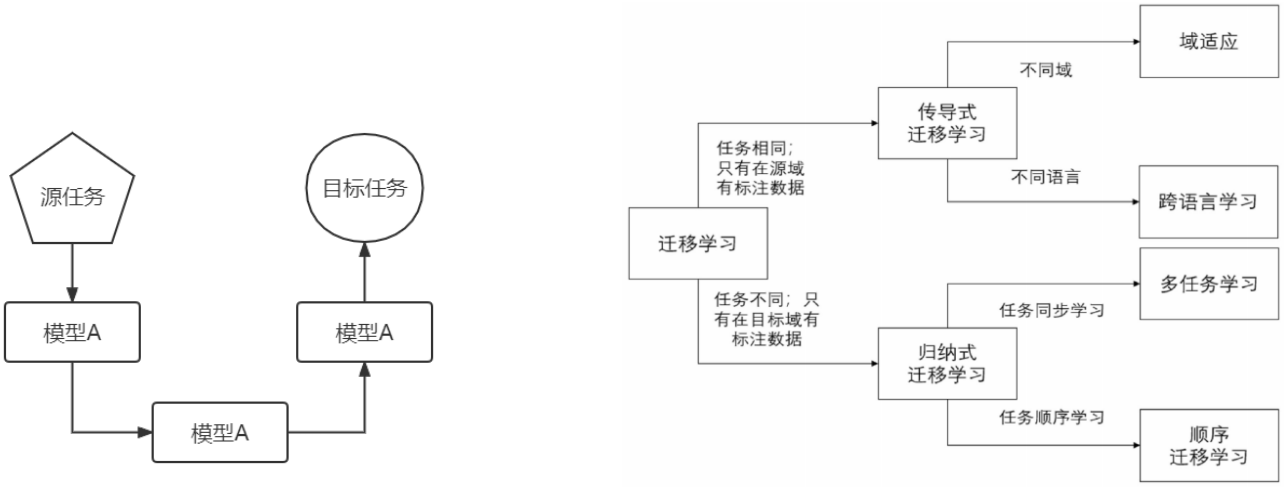
$$
\begin{aligned}
Accuracy &= \frac{TP+TN}{N} \\
Macro-F1 &= \frac{1}{S}\sum_{t\in S}\frac{2P_t\times R_t}{P_t+R_t} \\
where\ P &= \frac{TP_t}{TP_t+FP_t}, \quad R = \frac{TP_t}{TP_t+FN_t}
\end{aligned}
\tag{1}
$$

4 相关原理

4.1 迁移学习 (Transfer Learning)

4.1.1 迁移学习概述

迁移学习是一种将旧有的学习经验或者知识结构迁移到类似领域，从而改进目标领域或任务的方法 [5]; 是把为任务 A 开发的模型作为原始模型，将其重新应用在任务 B 中来开发出一套针对任务 B 的模型。其中任务 A 和任务 B 应该具有一定的相关性或者相似性, 基本流程如图 ()。



(a) 迁移学习一般流程　　(b) 迁移学习在 NLP 中的分类

图 2: 迁移学习

迁移学习的**关键**是找到目标任务与源任务之间的相似性，并顺利的实现知识/经验的迁移，使得计算机像人一样，通过之前的知识快速学习，举一反三，迅速适应另一个任务。

在自然语言处理中，根据一定的规则，将迁移学习的方法按图 () 分类 [5]，其中，顺序迁移学习的方法进展较大。

4.1.2 顺序迁移学习

顺序迁移学习的一般做法就是在大规模的无标签文本语料库上进行预训练表征，然后用有标签的数据将这些表征迁移到有监督的目标任务中，也就是 **"Pre-train" + "Fine-tune"** 的过程。首先通过大量的语料库在大模型上学习文本的通用表征，然后在通过修改结构等优化方案，使得模型适应另一个任务。

适应：通常使用的适应优化方案包括：(1) 不修改模型结构，在预训练模型的基础上下游添加一个线性层进行训练，如：Bert 通常的做法；(2) 修改预训练模型的内部结构。如 Houlsby[1] 尝试使用预训练的模型来尽可能地初始化结构不同的目标任务模型。

优化方案：需要确定是否改变，以及改变哪些参数，通常 (1) 不改变预训练模型参数，使用其输出作为表征，仅仅只训练下游的分类器；(2) 对模型整体进行微调：**需注意**，要避免覆盖有用的预训练信息，并最大限度地实现正向转移。

在进行模型参数更新时，需要注意：

1. 时间上：所有层在训练时均同时进行调整可能带来不好的性能。如：Hiton 等人提出对模型进行逐层训练去适应新的任务和数据；也有在共同训练的层之间利用不同的组合方法进行学习，最后再联合训练所有参数。

2. 强度上：使用较低的学习率训练模型避免模型遗忘以前的信息，低学习率在低网络层、训练早期和训练后期尤为重要。Howard[2] 提出在不同层中学习率衰减的微调方法；Liu[3] 提出，**预热 (Warm Up)** 可以减少训练早期的变异。

4.2 提示学习 (Prompt Learning)

4.2.1 提示学习概述

目前 NLP 邻域发展主要经历了四个范式：基于浅层学习的完全监督方法、基于深层学习的完全监督方法、预训练 + 微调的方法和预训练 + 提示 + 预测的方法 [4]。**提示学习**最大的区别就是加入了**提示**阶段，将下游任务进行重构，将输入输

出形式设计成预训练任务中的形式，在文本提示的帮助下解决的问题。(预训练 + 微调 (objective engineering)VS 提示学习 (textual prompt))。

如情感分类中，“I missed the bus today.”，我们可以接上一句提示：“I felt so ___ ”，让模型使用情绪表达词汇填空。以这种提示的方式来可控制模型的行为，**一定程度上使得语言模型本身就可以用来预测（predict）所期望的输出** [4]。

4.2.2 Prompt 基本流程

提示学习主要由 3 个步骤 [4]：

(1) **Prompt Addition**: 设计提示函数 (prompting function)，包括设计一个拥有输入槽和答案槽的模板，如：情感分析中：输入为：“I love this movie.”，则可设计 “[X],Overall, it was a [Z] movie.” X 为输入，Z 为预测答案。然后再使用待选择答案对 Z 进行填空。

(2) **Answer Search** : 此过程主要是在所有填空答案集中，选择出有最高打分的填充答案。可以使用 arg max 进行选择。

(3) **Answer Mapping**：将最高的分数的填充答案映射到最高概率的类别中，如情感分析中，填充可能是不同的情绪词汇 (e.g. “excellent”，“fabulous”，“wonderful”)，那么就需要通过映射，将其映射到正确的真实标签 (Positive) 中；部分任务填充就是答案，那么就不用进行映射。

部分模板如 ():

Type	Task	Input (`[X]`)	Template	Answer (`[Z]`)
Text CLS	Sentiment	I love this movie.	`[X]` The movie is `[Z]`.	great fantastic ...
	Topics	He prompted the LM.	`[X]` The text is about `[Z]`.	sports science ...
	Intention	What is taxi fare to Denver?	`[X]` The question is about `[Z]`.	quantity city ...
Text-span CLS	Aspect Sentiment	Poor service but good food.	`[X]` What about service? `[Z]`.	Bad Terrible ...
Text-pair CLS	NLI	`[X1]`: An old man with ... `[X2]`: A man walks ...	`[X1]? [Z], [X2]`	Yes No ...
Tagging	NER	`[X1]`: Mike went to Paris. `[X2]`: Paris	`[X1] [X2]` is a `[Z]` entity.	organization location ...
Text Generation	Summarization	Las Vegas police ...	`[X]` TL;DR: `[Z]`	The victim ... A woman ... ...
	Translation	Je vous aime.	French: `[X]` English: `[Z]`	I love you. I fancy you. ...

图 3: 不同任务模板实例 [4]

关于 Prompt Engineeri 和 Answer Engineering 可查看文献 [4] 进行更多学习，如何选择、设计模板、提示方法和答案集合等。

4.3 本实验设计

1. **迁移学习部分**：本实验的训练方案为：预训练 + 微调，在预训练好的语义相似度计算模型基础上，在 NLI 问题上进行二次预训练，同样使用较低学习率和 warm up 的方法，具体参数设置见 7.1 实验细节。

2. **提示学习部分**：本实验为 text pair classification，首先选择 Bert 预训练语言模型作为基础，同时选择 cloze prompts(完型填空式提示) 方式进行提示，具体模板构建和答案集合见 7.1 实验细节。

5 实验环境

本实验在百度 AI Studio 平台进行部署，使用 Paddle 的深度学习框架，以及部分 PaddleNLP 模型 API。

6 实验过程

本次实验基于《语义相似度计算》实验代码，因此本部分仅仅给出本实验修改或者添加的部分，其余部分仅进行文本描述。

部分 prompt 部分模型构建：

```
#构建槽
def nli_trans(text_a,text_b,tokenizer,max_seq_length):
    #模板:"a"？，__,"b"
    query='"'+text_a+'"'+'？'
    qid=','+'"'+text_b+'"'
    ids=tokenizer(query,qid,max_seq_len=max_seq_length)
    #加上mask的部分:
    seq_index=ids['input_ids'].index(102)
    ids['input_ids'].insert(seq_index+1,103)
    ids['token_type_ids'].insert(seq_index+1,1)
    ids['mask_index']=seq_index+1
    return ids
nli_trans_fun=partial(nli_trans,tokenizer=tokenizer_bert,max_seq_length=512)
```

函数作用：构建 Mask 输入

```
#输入:
ids=nli_trans_fun(text_a='今天外面有点冷',text_b='今天气温比较低')
#输出:
['[CLS]','"','今','天','外','面','有','点','冷','"','？','[SEP]',
'[MASK]',',','"','今','天','气','温','比','较','低','"','[SEP]']
```

基于 Bert 的 MaskLanguageModel：

```
from paddlenlp.transformers import BertPretrainedModel
class BertLMPredictionHead(nn.Layer):
    """
    Bert Model with a `language modeling` head on top for CLM fine-tuning.
    """

    def __init__(self,
                 hidden_size,
                 vocab_size,
                 activation,
                 embedding_weights=None):
        super(BertLMPredictionHead, self).__init__()
        self.transform = nn.Linear(hidden_size, hidden_size)
        self.activation = getattr(nn.functional, activation)
        self.layer_norm = nn.LayerNorm(hidden_size)
        self.decoder_weight = self.create_parameter(
            shape=[vocab_size, hidden_size],
            dtype=self.transform.weight.dtype,
            is_bias=False) if embedding_weights is None else embedding_weights
        self.decoder_bias = self.create_parameter(
            shape=[vocab_size], dtype=self.decoder_weight.dtype, is_bias=True)

    def forward(self, hidden_states, masked_positions=None):
        if masked_positions is not None:
```

```
            hidden_states = paddle.reshape(hidden_states,
                                           [-1, hidden_states.shape[-1]])
            hidden_states = paddle.tensor.gather(hidden_states,
                                                 masked_positions)
        # gather masked tokens might be more quick
        hidden_states = self.transform(hidden_states)
        hidden_states = self.activation(hidden_states)
        hidden_states = self.layer_norm(hidden_states)
        #print(hidden_states.shape)
        #print(self.decoder_weight.shape)
        hidden_states = paddle.tensor.matmul(
            hidden_states, self.decoder_weight,
            transpose_y=True) + self.decoder_bias
        return hidden_states

class BertOnlyMLMHead(nn.Layer):
    def __init__(self, hidden_size, vocab_size, activation, embedding_weights):
        super().__init__()
        self.predictions = BertLMPredictionHead(
            hidden_size=hidden_size,
            vocab_size=vocab_size,
            activation=activation,
            embedding_weights=embedding_weights)

    def forward(self, sequence_output, masked_positions=None):
        prediction_scores = self.predictions(sequence_output, masked_positions)
        return prediction_scores


class BertForMaskedLM(BertPretrainedModel):

    def __init__(self, bert,vocab_size,embedding_dim_weight=None):
        super(BertForMaskedLM, self).__init__()
        self.bert = bert
        self.cls = BertOnlyMLMHead(
            self.bert.config["hidden_size"],
            vocab_size,
            self.bert.config["hidden_act"],
            embedding_weights=embedding_dim_weight)

        self.apply(self.init_weights)

    def forward(self,
                input_ids,
                token_type_ids=None,
                masked_positions=None,
                position_ids=None,
                attention_mask=None):

        outputs = self.bert(
            input_ids,
            token_type_ids=token_type_ids,
            position_ids=position_ids,
```

```
            attention_mask=attention_mask)
        sequence_output = outputs[0]
        prediction_scores = self.cls(sequence_output, masked_positions=masked_positions)
        return prediction_scores
```

7 实验结果与分析

7.1 基于 Transfer Learning 的 NLI 实验

本次实验主要使用了 NLI 数据机上预训练好的 Bert 模型，进一步在文本相似度分类问题上进行二次与训练。

7.1.1 实验细节

实验首先在 OCNLI 数据集上进行首次预训练，然后分别在 AFQMC 数据集，抽取正负样本各 1k,5k,10k 用于二次预训练和直接预训练进行比对，实验参数如下：

最大学习率	Warmup	Batch size	max_length	max_epoch	优化器	学习准则
5e-5	线性	64	512	10	AdamW	交叉熵损失

表 1: 实验参数设置

7.1.2 实验结果

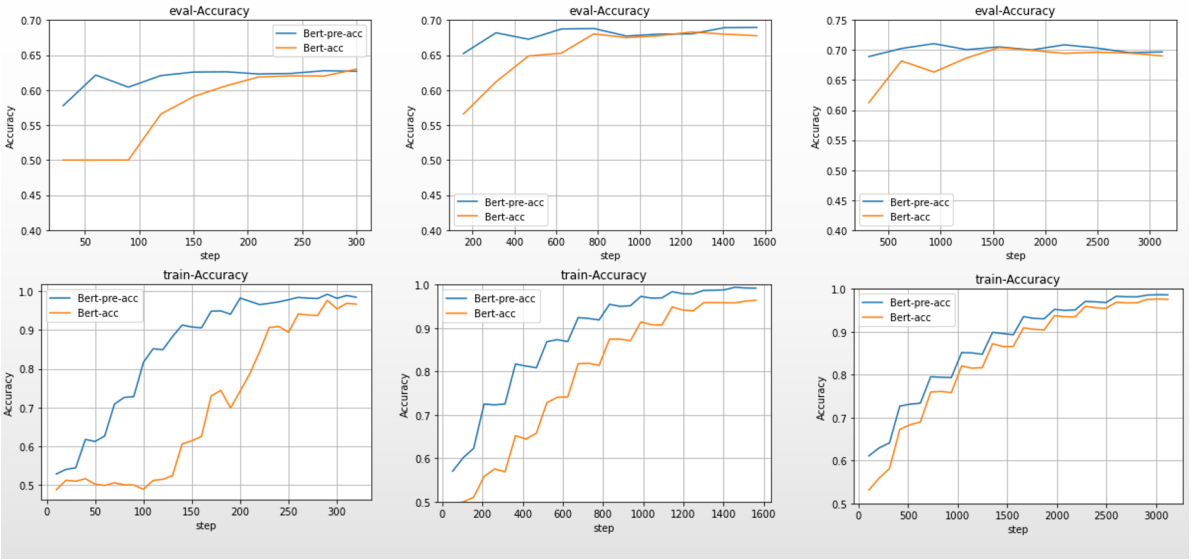


图 4: 测试集和训练集准确率变化：（左到右数据集分别为 1k,5k,10k）

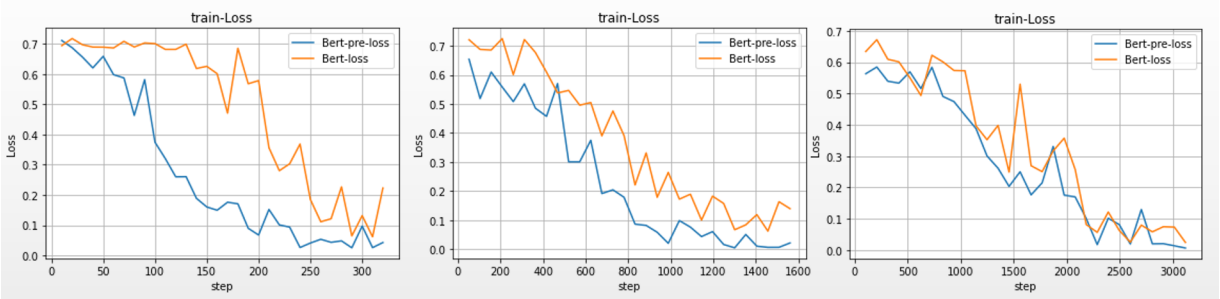
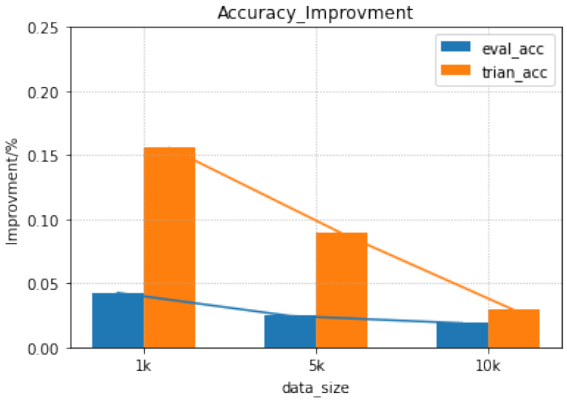
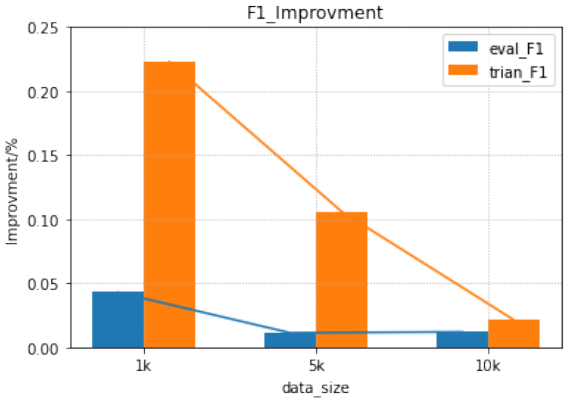


图 5: 训练集 Loss 变化：（左到右数据集分别为 1k,5k,10k）

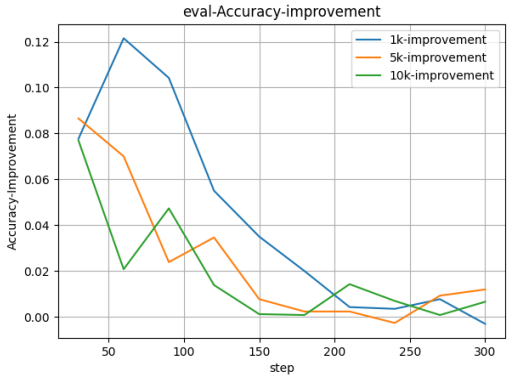


(a) 训练集测试集准确率平均提升对比

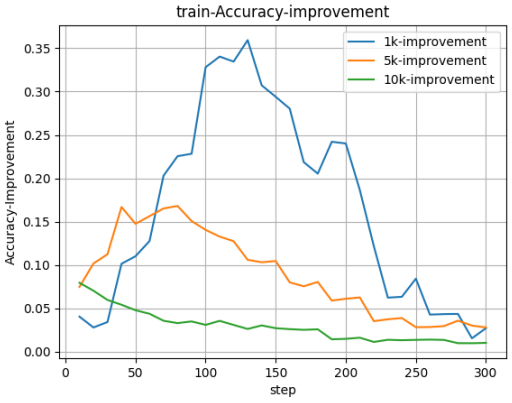


(b) 训练集测试集 F1 分数平均提升对比

图 6: 模型是否二次预训练性能对比



(a) 测试集准确率提升变化



(b) 训练集准确率提升变化

图 7: NLI 迁移到语义相似度学习各提升变化

7.1.3 实验分析

1. 观察图 4 图 5，无论在训练集还是测试集，二次预训练的模型 (蓝色) 的准确率基本始终比首次预训练的模型 (橙色) 准确率高，同时 Loss 较低；特别时在模型训练初期，二次预训练的模型能够较迅速降低 Loss，拟合训练集，同时快速提升在训练集和验证集上的准确率。这一定程度上**说明**了本次实验中，在 NLI 上训练好的模型保留了部分学习到的 NLI 知识，在进行二次预训练时，将这种知识运用在了语义相似度计算上，一定程度上，这些先验知识为语义相似度计算提供了部分知识基础，因为在推理中，一定程度上模型也学习到了两个句子的语义，使得模型在二次 fine-tuning 时，能够利用之前 NLI 的知识为基础，迅速拟合数据，提高模型的准确率。而直接 fine-tuning 的模型由于没有这种知识，则需要一定的时间拟合数据，模型性能提升较慢。

2. 观察图 6，在不同数据集大小上，是否进行二次预训练的模型性能提升平均值 (准确率和 F1) 比较。容易看到，随着数据集的规模变大，准确率和 F1 的平均提升逐渐降低；在较小的数据机上，NLI 预训练后对模型性能提升更大。**分析认为**，在数据集较小时，直接进行 fine-tuning 会由于数据集较小，而模型较大，在训练时无法学习到足够的任务相关的有效信息，而导致模型性能较差；而经过相似任务 (NLI 和语义相似度) 任务预训练的模型，可以提供部分相关的先验知识或者任务的共同特征，用于缓解这种信息特征的缺失，进而能够使得模型在性能上相对有较大提升。当数据集较为充足时，这种信息缺失问题就会减少很多，此时预训练得到的知识则帮助相对减弱。

3. 观察图 7 在训练集和验证集上的准确率提升变换，特别是图 7(a)，模型提升变化为以开始较高，然后在训练过程中逐渐降低，进一步佐证了：NLI 任务训练好的模型在语义相似度计算任务时，一定程度上为模型带入的先验知识或者相似的人物特征，使得模型在一开始就有一个较好的初始化参参数，具有相对较好的泛化能力，在验证集上性能有一定提升；同时，数据量越小，迁移学习带来的辅助作用相对更大。

总结：(1) 对于相似或者有共同特征的任务，迁移学习能够有效将一个任务上学习到的知识应用到另一个任务上，使得在此

任务上模型的性能有一定会程度的提升，并使得其能够利用之前学习到的知识，快速拟合数据，学习本任务的有效信息和特征。(2) 在数据集较小时，模型会由于无法提取到足够的有效任务特征而使得模型表现不好，数据集越大，这种有效信息确实问题会减轻，同时迁移学习在能够使用相似任务的共同特征或者知识对这种缺失进行补充，一定程度上可以缓解该问题。(3) 综上所述，迁移学习能够在相似任务之间迁移此前学习到的知识或者特征，能够有效帮助模型更快训练，或者在较小数据集上性能得到一定程度的提升。

注：10 中做了一个语义相似度到 NLI 的迁移学习，此时 NLI 数据集较大，因此在后期直接预训练的效果更好；单在训练之初，经过预训练的模型性能比直接训练的效果更佳，也一定程度上说明了迁移学习在初期为任务提供了一定基础，使得模型在初期有较好的表现；查阅过部分论文后，了解到：在迁移学习时，先固定一定轮数不改变迁移模型的参数，此后再解冻，模型性能提升会更快，后期将一步实验，观察结果。

8 收获与感想

1. 首先，通过阅读相关论文和查阅资料，对迁移学习和提示学习有了初步了解，基本掌握其中的原理和方法，拓宽了我的知识面。

2. 通过本次实验，我能够观察到，深度的方法确实捕获了相似任务之间的特征，同时迁移学习一定程度上也确实将不同任务之间的有效信息进行了共享，为模型再本任务上的性能有一定提升。一定程度上认为，虽然深度学习的方法可解释性较差，但是从宏观的角度上看，深度的方法也一定程度上模拟到了部分人类的思维过程，迁移学习仿佛也模拟了人类在不同任务之间，使用此前学习的经验用于提升当前任务的过程，这让我再一次觉得，虽然深度学习、NLp 或者人工智能，虽然要完全处理透彻很难，但也一定程度上有了模拟人类内在机制的效果。进一步加深了我对具有可解释性的方法、或者可解释性结构的模型 (如 AAAI2021 中科大提出的 RAM, 联想空间模型) 有了更多兴趣，更想去关注、学习和思考如何利用浅层、深层的方法作为工具，在宏观上构建可解释的方法和模型。

3. 最后，虽然预训练模型的 prompt 模式并没有成功进行实验，但是通过阅读论文，让我对这一部分有了一定了解，知道其也是 NLP 解决问题的又一范式，同时我认为提出的思想：对模型进行任务性的描述，将任务描述告知模型，使得模型性能提升，并且可以进行少样本学习，这种思想仿佛也是模拟了人类思维，启发意义很大。

参考文献

[1] Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin de Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for nlp. In *ICML*, 2019.

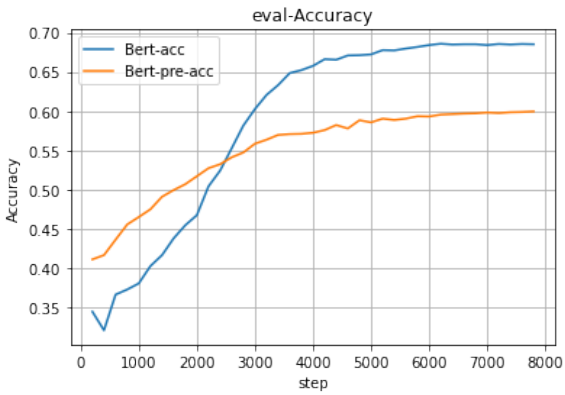
[2] Jeremy Howard and Sebastian Ruder. Universal language model fine-tuning for text classification. In *ACL*, 2018.

[3] Nelson F. Liu, Matt Gardner, Yonatan Belinkov, Matthew E. Peters, and Noah A. Smith. Linguistic knowledge and transferability of contextual representations. *ArXiv*, abs/1903.08855, 2019.

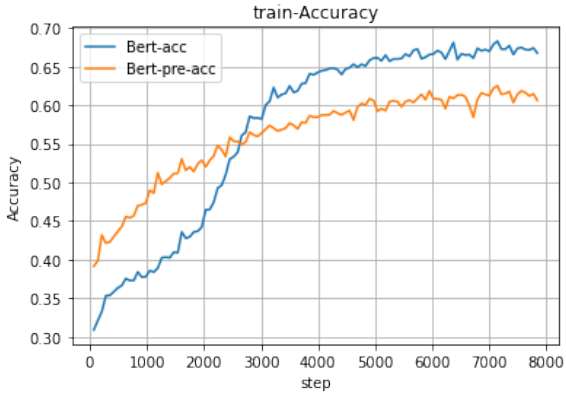
[4] Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *ArXiv*, abs/2107.13586, 2021.

[5] 袁君. 迁移学习在自然语言处理中的应用综述. *科教导刊：电子版*.

9 其他数据



(a) 测试集准确率变化



(b) 训练集准确率

图 8: 语义相似度到 NLI 迁移学习