Prefix-Tuning: Optimizing Continuous Prompts for Generation

Xiang Lisa Li,Percy Liang

2021

摘要

微调是利用大型通用预训练语言模型来应用于下游任务。然而，全量微调修改了所有语言模型参数，因此需要为每个任务存储一个完整的副本。在本文中提出了前缀微调，这是用于自然语言生成任务的一种轻量级替代方法，它保持语言模型参数冻结，并优化一系列连续的任务特定向量，称之为前缀。前缀微调使用提示为语言模型进行汲取灵感，允许后续标记参考此前缀，就好像它是“虚拟标记”一样。本文将前缀微调应用于 GPT-2 以进行 Table-to-Text 任务，以及应用于 BART 以进行摘要生成。通过仅修改 0.1% 的参数，前缀微调在完整数据设置中获得了可比较的性能，在低数据设置中优于微调。

关键词：前缀微调；大语言模型；

1 引言

微调是利用大型通用预训练语言模型 (LM) 来应用于下游任务。然而，目前需要为每个任务存储 LM 参数的修改副本。鉴于当前 LM 的规模，这可能会导致昂贵的开销；例如，GPT-2 具有 774M 个参数，而 GPT-3 具有 1750B 个参数。解决这一问题的方法是轻量级微调，即冻结大部分预训练参数，只调整较小的参数集。

在本文中，我们提出了前缀微调方法，这是自然语言生成任务中微调方法的一种轻量级替代方法，其灵感来源于提示法。考虑生成数据表文本描述的任务，如图 1 所示，其中任务输入是线性化表格（如” 名称：星巴克 | 类型：咖啡店”），输出是文本描述（如” 星巴克提供咖啡”）。前缀微调将一系列连续的任务特定向量预置到输入中，我们称之为前缀，如图 1（下）中红色块所示。为了生成每个标记，LM 可以像处理” 虚拟标记” 序列一样处理前缀，但与提示不同的是，前缀完全由自由参数组成，与真实标记并不对应。图 1（上图）中的微调更新了 LM 的所有参数，因此需要为每个任务存储一份经过调整的模型副本，而前缀微调只对前缀进行优化。因此，我们只需存储一份大型 LM 和针对特定任务学习的前缀，因此每个额外任务的开销非常小（例如，从表格到文本的 25 万个参数）。

与全量微调不同，前缀微调也是模块化的：我们会训练一个上游前缀，该前缀会引导一个未修改的 LM，因此一个 LM 可以同时支持多项任务。在任务与用户相对应的个性化背景下，我们将为每个用户设置一个单独的前缀，只对该用户的数据进行训练，从而避免数据交叉污染。此外，基于前缀的架构使我们甚至可以在一个批次中处理来自多个用户/任务的示例，这

是其他轻量级微调方法（如 Adapter 微调）无法做到的。我们利用 GPT-2 和 BART 进行前缀微调评估，在存储方面，前缀微调比全量微调少存储 1000 倍的参数。就在完整数据集上进行训练时的性能而言，前缀调优和微调优在 Table-to-Text 方面性能相当，而前缀微调在摘要生成方面性能略有下降。在低数据设置下，前缀微调在这两项任务上的表现都优于微调。因此本文对推动高效微调模型发展有较大的意义。

2 相关工作

轻量级微调。轻量级方法主要分为三大类，分别为 additive、selective 和 Reparametrization-based [13]，如图 1。它们可以通过基本方法或基本概念框架来区分：该方法是为模型引入新参数，还是对现有参数的一小部分进行微调？或者，也可以根据其主要目标进行分类：该方法的目标是最大限度地减少内存占用，还是仅仅提高存储效率？对于前者的区分方法更易于理解，因此这里以基本概念来进行区分。

Additive 方法。Additive 方法主要思想是用额外的参数或者层来增强与训练模型，接着只训练新添加的参数或者层。而 Additive 方法细分为两大子类别，主要分为适配器方法和提示方法。适配器方法是插入带有可训练参数的新模块，例如适配器微笑是在预训练 LM 的每一层之间插入任务特定层（适配器） [7]。提示微调是一种利用预训练 LM 的方法，即在任务输入中预先添加指令和一些示例，并通过 LM 生成任务输出。对于自回归 LM 而言，最成功的提示形式是 GPT-3，它使用人工设计的提示来调整其生成，以适应不同任务中的少量提示设置。对于像 BERT 这样的掩码式 LM，提示工程已被探索用于自然语言理解任务。例如，AutoPrompt [12] 搜索一连串离散的触发词，并将其与每个输入连接起来，从 BERT 中获取情感或事实知识。与 AutoPrompt 不同的是，我们的方法优化的是连续前缀，因为连续前缀更具表现力。相比之下，前缀微调优化的是适用于该任务所有实例的特定任务前缀。因此，与应用仅限于句子重构的前述工作不同，前缀微调可应用于 NLG 任务。前缀微调是属于提示方法的一种，这种方法冻结了大部分预训练参数，只对较小的一组参数进行调整。与这一方法相比，我们的方法在任务特定参数方面进一步减少了 30 倍，仅调整了 0.1%，同时在 Table-to-Text 任务中保持了相当的性能。

Selective 方法。选择方法是选择模型的某些子网络或者结构进行微调。选择性方法最早的方法只是对网络的几个顶层进行微调，现代方法通过基于层的类型或者内部结构，稀疏更新方法是选择方法的一个极端版本- [14]，它可以完全忽略模型结构，单独选择参数。

Reparametrization-based 方法。基于重构的参数高效微调方法利用低秩表示来最小化可训练参数的数量。神经网络具有低维表示这一概念已在深度学习的实证和理论分析中得到广泛探讨。通过证明，在低秩子空间中可以有效地进行微调- [1]。此外，他们还证明，对于较大的模型或预先训练较长时间的模型，需要调整的子空间的大小较小。通过重新参数化神经网络参数的更新公式，来减少预训练中所需的训练时间。

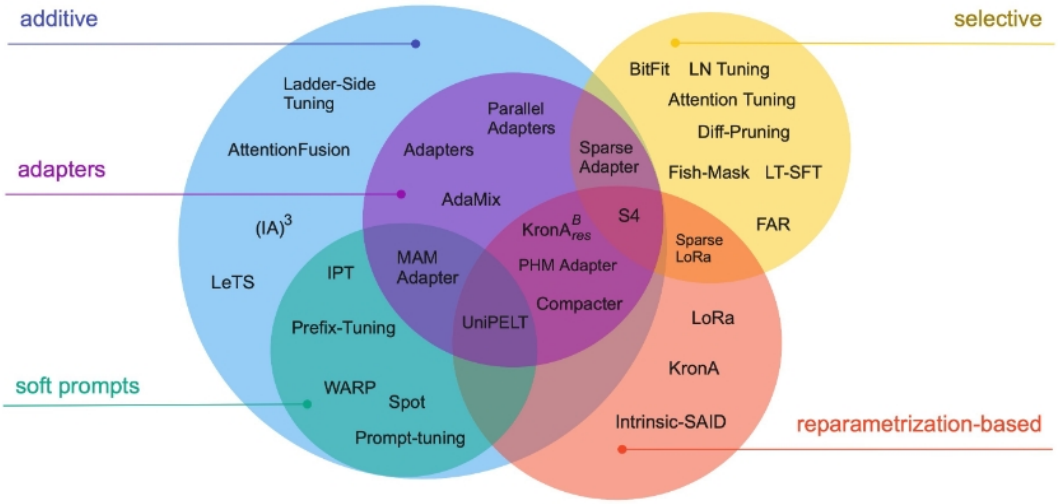

图 1. 轻量级微调方法总结

可控生成。可控生成的目的是引导预训练的语言模型与句子级属性（如积极情绪或体育）相匹配。这种控制可以在训练时进行：对语言模型进行了预训练 [10]，使其符合关键词或 URL 等元数据的条件。控制也可以在解码时进行。然而，目前还没有直接的方法来应用这些可控生成技术，对生成的内容实施细粒度控制，这也是表格到文本和摘要等任务的要求。

3 本文方法

3.1 初步想法

提示法证明，适当的语境条件可以在不改变 LM 参数的情况下引导 LM。例如，如果我们希望 LM 生成一个单词（如 Obama），我们可以将其常见搭配作为上下文（如 Barack）的前缀，这样 LM 就会为所需单词分配更高的概率。将这一直觉延伸到生成单个单词或句子之外，我们希望找到一种语境，引导 LM 解决 NLG 任务。直观地说，上下文可以通过指导从 x 中提取什么来影响任务输入 x 的编码，也可以通过指导下一个标记的分布来影响任务输出 y 的生成。然而，这样的语境是否存在并不明显。使用自然语言任务指令（如” 用一句话概括下表”）作为上下文可能会指导人类解决任务，但这对中等规模的预训练 LM 来说是失败的。

我们可以将指令优化为连续的单词嵌入，而不是对离散的标记进行优化，其效果将向上传播到所有转换器激活层，并向右传播到后续标记。严格来说，这比局限于实词嵌入的离散提示更具表现力。通过优化所有层的激活，而不仅仅是嵌入层的激活，前缀微调技术在提高表现力方面更进一步。另一个好处是，前缀微调可以直接修改网络中更深的表征，从而避免了在网络深度上的冗长计算路径。

3.2 本文方法概述

微调是在通用大语言模型的基础上，调整模型参数来生成 Table-to-Text 与摘要生成的模型。而本文提出的前缀微调是一种高效的微调方法，在输入前插入一段连续的任务特定向量，并且在每一个 transformers 层激活函数前插入一段连续的任务特定向量，称之为前缀，如图 2，

图中红色部分是两种微调方法中需要优化的模型参数。通过优化上述插入的前缀向量，冻结模型其他参数，只需要修改 0.1% 的参数，因此获得高效微调的效果。

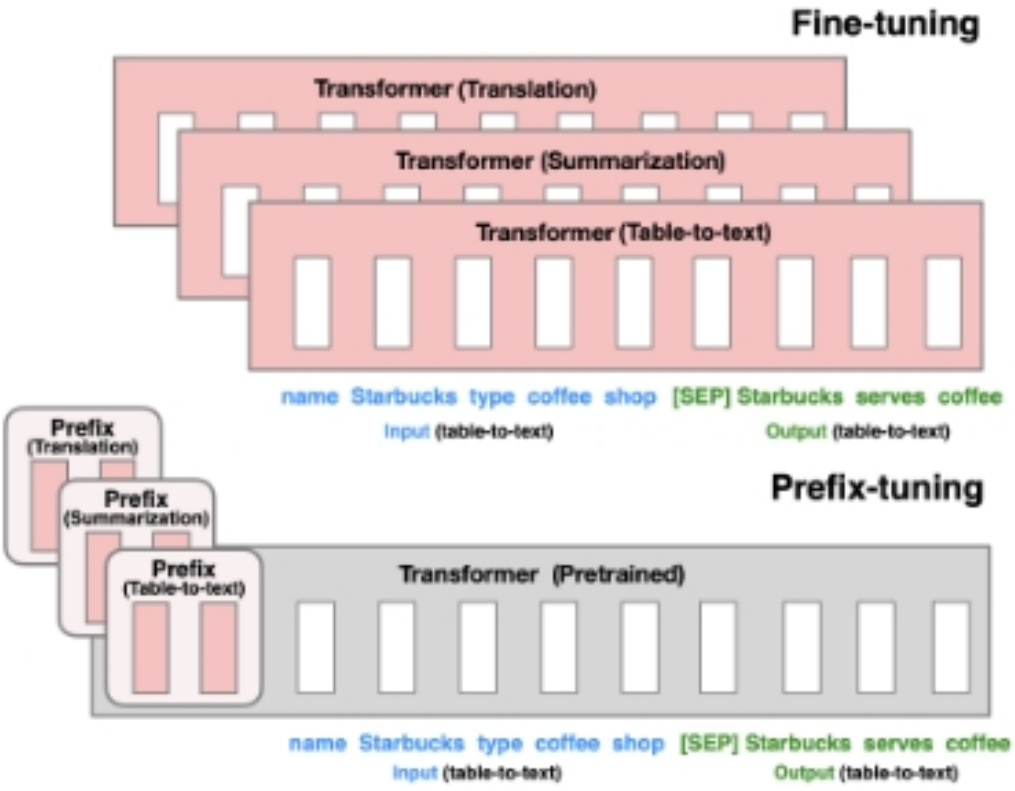

图 2. 全量微调与前缀微调的区别

3.3 自回归 LM 和编码器-解码器架构

假设我们有一个以 ϕ 为参数的自回归语言模型，设 $z = [x; y]$ 是 x 和 y 的并集，$x_i dx$ 表示与 x 对应的下标序列，y_{idx} 表示与 y 对应的下标序列，第 i 时间步的激活向量为 h_i，自回归神经网络计算 h_i 如下所示

$$h_i = \mathrm{LM}_\phi(z_i, h_{<i}) \tag{1}$$

我们也可以使用编码器-解码器架构, 如 BART，其中 x 由双向编码器编码，解码器自回归预测 y（以编码的 x 及其左语境为条件）。我们使用相同的索引和激活符号，每个 h_i 由双向编码器计算；$i \in y_{idx}$ 的每个 h_i 由自回归解码器使用上述相同方程计算。

3.4 前缀向量

如图 3所示，前缀微调为自回归 LM 插入前缀，得到 $z = [PERFIX; x; y]$，或者为编码器-解码器架构预置 $z = [PERFIX; x; PERFIX'; y]$。这里 P_{idx} 表示前缀索引序列，我们用 $|P_{idx}|$ 表示前缀索引序列的长度。我们遵循上述的递推公式(1)，但前缀索引的激活是自由参数，其维度为 $|P_{idx}| \times dim(h_i)$。

$$h_i = \begin{cases} P_\theta[i,:] & \text{if} i \in P_{idx} \\ LM_\theta(z_i, h_{<i}) & \text{otherwise} \end{cases} \tag{2}$$

训练目标与(2)相同，但可训练参数集有所变化：语言模型参数 ϕ 固定不变，前缀参数 θ 是唯一的可训练参数。在这里，每个 h_i 都是可训练 P_θ 的函数。当 $i \in P_{idx}$ 时，这一点很明显，因为 h_i 是直接从 P_θ 复制而来的。相反，h_i 仍然取决于 P_θ，因为前缀激活总是在左侧上下文中，因此会影响右侧的任何激活。

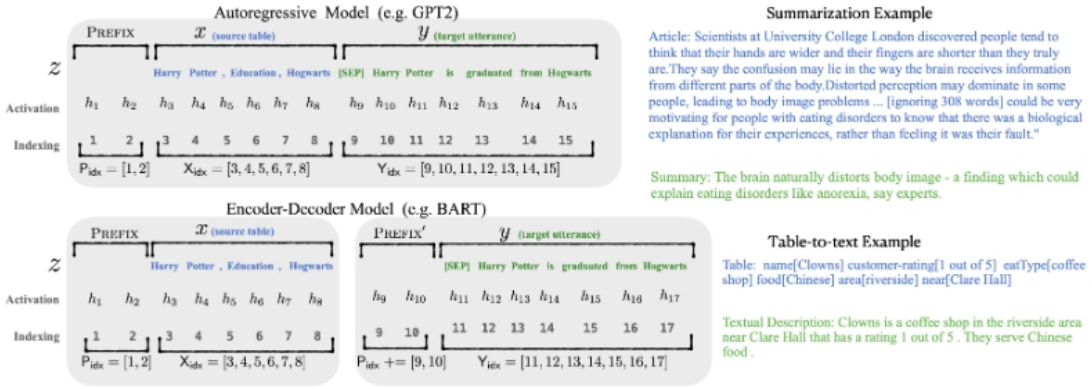

图 3. 前缀微调示例

3.5 参数更新

在完全微调框架中，我们使用预训练参数 ϕ 进行初始化。这里 p_ϕ 是一个可训练的语言模型分布，我们根据以下对数似然目标进行梯度更新：

$$\max_\phi \log p_\phi(y|x) = \max_\phi \sum_{i \in Y_{idx}} \log p_\phi(z_i|h_{<i}) \tag{3}$$

但是通过上述公式直接优化 P_θ 会导致优化不稳定，甚至性能有所下降。因此我们使用重参数化，将矩阵 $P_\theta[i,:] = MLP_\theta(P'_\theta[i,:])$ 重新参数化成一个由大型前馈神经网络 MLP 组成的较小矩阵 P'_θ。因此现在可训练参数包括 P'_θ 和 MLP_θ。同时训练这些参数，可以避免以上情况，需要注意的是 P_θ 和 P'_θ 的行数相同，即前缀长度相同，但是列数不同。

4 复现细节

4.1 与已有开源代码对比

本文的复现代码部分参考论文的开源代码 https://github.com/ XiangLi1999/PrefixTuning. 但是这份开源代码是无法直接运行的，里面有很多 bug 需要更正，同时也缺少了一些必要的文件，需要进行一些处理才能使用。

数据集。该代码只给出 E2E 数据集和对应的评测指标处理，而在 WebNLG 和 DART 数据集中是缺少了必要的文件的。在 WebNLG 数据集中，没有给出论文的指标的计算文件，因此需要找出对应指标的源代码，并更改代码中的一些细节问题，才能正确地得出对应指标分数，其源代码部分处理如图所示。

```bash
#! /bin/bash
cd ../evaluation
current_dir=$(pwd)
# 打印当前目录
echo "Current Directory: $current_dir"

OUTPUT_FILE=example/bart-large_webnlg-all.txt
export TEAMR=team

echo $OUTPUT_FILE
echo $TEAMR

cp $OUTPUT_FILE ./webnlg-automatic-evaluation/submissions/$TEAMR.txt

# BLEU
cd webnlg-automatic-evaluation/
python evaluation.py $TEAMR
. bleu_eval_3ref.sh
cd ..
echo "ALL:"; cat webnlg-automatic-evaluation/eval/bleu3ref-$TEAMR\_all-cat.txt > bleu_all.txt
# BLEU seen
echo "SEEN:"; cat webnlg-automatic-evaluation/eval/bleu3ref-$TEAMR\_old-cat.txt > bleu_seen.txt
# BLEU unseen
echo "UNSEEN:"; cat webnlg-automatic-evaluation/eval/bleu3ref-$TEAMR\_new-cat.txt > bleu_unseen.txt

# METEOR
cd meteor-1.5/
../webnlg-automatic-evaluation/meteor_eval.sh

cd ..
echo "ALL:"; cat webnlg-automatic-evaluation/eval/meteor-$TEAMR-all-cat.txt > meteor_all.txt
# METEOR seen
echo "SEEN:"; cat webnlg-automatic-evaluation/eval/meteor-$TEAMR-old-cat.txt >meteor_seen.txt
# METEOR unseen
echo "UNSEEN:"; cat webnlg-automatic-evaluation/eval/meteor-$TEAMR-new-cat.txt > meteor_unseen.txt
```

图 4. WebNLG 指标计算脚本

```python
def bleu_ref_files_gen(b_reduced, param):
    ids_refs = defaultdict(list)
    for entry in b_reduced.entries:
        ids_refs[entry.id] = entry.lexs
    # length of the value with max elements
    max_refs = sorted(ids_refs.values(), key=len)[-1]
    # write references files for BLEU
    for j in range(0, len(max_refs)):
        with open('references/gold-' + param + '-reference' + str(j) + '.lex', 'w+') as f:
            out = ''
            # extract values sorted by key (natural sorting)
            values = [ids_refs[key] for key in natsorted(ids_refs.keys(), reverse=False)]
            for iter, ref in enumerate(values):
                try:
                    # detokenise
                    lex_detokenised = ' '.join(re.split('(\W)', ref[j].lex))
                    # delete redundant white spaces
                    lex_detokenised = ' '.join(lex_detokenised.split())
                    # convert to ascii and lowercase
                    out += unidecode(lex_detokenised.lower()) + '\n'
                except IndexError:
                    out += '\n'
                    lex_detokenised = ''
                id_str = str(iter + 1)
                '''with open('references/bleu_per_block/gold-' + param + '-reference' + str(j) + '-' + id_str + '.lex', 'w+') as f_item:
                    f_item.write(unidecode(lex_detokenised.lower()))'''
            f.write(out)
```

图 5. WebNLG 指标计算

在 DART 数据集中，该代码中没有给出数据集、数据处理方法以及指标计算，因此我使用的数据集位于 https://github.com/Yale-LILY/dart，并更改其中的脚本以及对应的源代码，如图 6

```
#! /bin/bash
cd ../evaluation

OUTPUT_FILE=example/bart-large_dart.txt

TEST_TARGETS_REF0=dart_reference/all-delex-reference0.lex
TEST_TARGETS_REF1=dart_reference/all-delex-reference1.lex
TEST_TARGETS_REF2=dart_reference/all-delex-reference2.lex

# BLEU
./multi-bleu.perl ${TEST_TARGETS_REF0} ${TEST_TARGETS_REF1} ${TEST_TARGETS_REF2} < ${OUTPUT_FILE} > bleu.txt

python prepare_files.py ${OUTPUT_FILE} ${TEST_TARGETS_REF0} ${TEST_TARGETS_REF1} ${TEST_TARGETS_REF2}
# METEOR
cd meteor-1.5/
java -Xmx2G -jar meteor-1.5.jar ../${OUTPUT_FILE} ../all-notdelex-refs-meteor.txt -l en -norm -r 8 > ../meteor.txt
cd ..

# TER
cd tercom-0.7.25/
java -jar tercom.7.25.jar -h ../relexicalised_predictions-ter.txt -r ../all-notdelex-refs-ter.txt > ../ter.txt
cd ..
```

图 6. DART 指标计算

实现代码。开源代码中出现了很多 bug 需要修复。（1）代码中需要加载原始的 gpt2 的模型参数，由于在代码中函数接口下载非常慢，所以直接到 hugging-face 官网下载 gpt2 的模型，接着直接在代码中加载。（2）代码中对于已经微调好的模型，不能直接在微调好的模型上接着训练，而是从头训练，因此需要添加以下代码，如图 7。（3）对于论文的 low-data 实验，代码中没有给出分别长度为 50,100,200,500 长度的数据集，因此需要对数据集进行采集出 50,100,200,500 长度的数据集，才能进行 low-data 实验，添加的数据处理代码如图 8

```
    if model_args.prefixModel_name_or_path is not None:
        config2 = AutoConfig.from_pretrained(model_args.prefixModel_name_or_path, cache_dir=model_args.cache_dir)
        # print(config2)

        if model_args.prefix_mode == 'embedding':
            model = PrefixEmbTuning.from_pretrained(
                    model_args.prefixModel_name_or_path,
                    from_tf=bool(".ckpt" in model_args.prefixModel_name_or_path),
                    config=config2,
                    cache_dir=model_args.cache_dir,
                    model_gpt2=gpt2, optim_prefix=optim_prefix_bool, preseqlen=model_args.preseqlen,
                    use_infix=(data_args.format_mode == 'infix')
                )

        elif model_args.prefix_mode == 'activation':

            model = PrefixTuning.from_pretrained(
                model_args.prefixModel_name_or_path,
                from_tf=bool(".ckpt" in model_args.prefixModel_name_or_path),
                config=config2,
                cache_dir=model_args.cache_dir,
                model_gpt2=gpt2, optim_prefix=optim_prefix_bool, preseqlen=model_args.preseqlen,
                use_infix=(data_args.format_mode == 'infix')
            )
        else:
            assert False, "invalid prefix mode"
```

图 7. 添加代码-模型加载

```
import random
import argparse

def random_sample(src_file, output_file, size):
    with open(src_file, 'r', encoding='utf-8') as src:
        # 读取所有行
        lines = src.readlines()

        # 确保抽样大小不超过总行数
        size = min(size, len(lines))

        # 随机抽取指定行数的数据
        sampled_lines = random.sample(lines, size)

        # 将抽取的数据写入输出文件
        with open(output_file, 'w', encoding='utf-8') as output:
            output.writelines(sampled_lines)
parser = argparse.ArgumentParser()
parser.add_argument("--size",type=int,default=50)
args = parser.parse_args()
# 调用函数，将src1_train.txt中的100行数据随机抽取并写入lowdata_size.txt
random_sample('src1_train.txt', 'lowdata_{}.txt'.format(args.size), args.size)
```

图 8. 添加代码-数据处理

4.2 实验环境搭建

主要是安装 transformers 版本为 3.2.0，但是在安装过程中会出现很多版本冲突，需要将冲突的包版本卸载后，再安装不会冲突的版本，如图 9。

```
setup(
    name="transformers",
    version="3.2.0",
    author="Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Sam Shleifer, Patrick von Platen, Sylvain Gug
    author_email="thomas@huggingface.co",
    description="State-of-the-art Natural Language Processing for TensorFlow 2.0 and PyTorch",
    long_description=open("README.md", "r", encoding="utf-8").read(),
    long_description_content_type="text/markdown",
    keywords="NLP deep learning transformer pytorch tensorflow BERT GPT GPT-2 google openai CMU",
    license="Apache",
    url="https://github.com/huggingface/transformers",
    package_dir={"": "src"},
    packages=find_packages("src"),
    install_requires=[
        "numpy",
        "tokenizers == 0.8.1.rc2",
        # dataclasses for Python versions that don't have it
        "dataclasses;python_version<'3.7'",
        # utilities from PyPA to e.g. compare versions
        "packaging",
        # filesystem locks e.g. to prevent parallel downloads
        "filelock",
        # for downloading models over HTTPS
        "requests",
        # progress bars in model download and training scripts
        "tqdm >= 4.27",
        # for OpenAI GPT
        "regex != 2019.12.17",
        # for XLNet
        "sentencepiece != 0.1.92",
        # for XLM
        "sacremoses",
    ],
```

图 9. 安装环境脚本

5 实验结果分析

5.1 数据集以及评测指标

我们在三个标准神经生成数据集上对表格到文本任务进行了评估：E2E [6]、WebNLG [4] 和 DART [5]。E2E 数据集包含 8 个不同字段的约 50K 个示例；它包含一个源表的多个测试引用，平均输出长度为 22.9。我们使用了官方评估脚本，其中包括 BLEU [8]、NIST [2]、METEOR [9]、ROUGE-L [3] 和 CIDEr [11], 数据集示例如图 11。WebNLG 数据集包含 22K 个示例，输入 x 是（主体、属性、客体）三元组的序列。平均输出长度为 22.5。在训练和验证拆分中，输入描述的实体来自 9 个不同的 DBpedia 类别（如纪念碑）。测试部分由两部分组成：前半部分包含训练数据中出现过的 DB 类别，后半部分包含 5 个未出现过的类别。这些未见类别用于评估外推法。我们使用官方评估脚本，该脚本会报告 BLEU、METEOR 和 TER，数据集示例如图 12。DART 是一个开放领域的从表格到文本的数据集，其输入格式（实体-关系-实体三元组）与 WebNLG 相似。平均输出长度为 21.6。它由来自 WikiSQL、WikiTableQuestions、

E2E 和 WebNLG 的 82K 个示例组成，并应用了一些手动或自动转换。我们使用官方评估脚本并报告了 BLEU、METEOR、TER。在总结任务中，我们使用了 XSUM [?] 数据集，这是一个新闻文章的抽象总结数据集。该数据集有 225K 个例子。文章的平均长度为 431 个单词，摘要的平均长度为 23.3 个单词。我们报告了由 python 软件包 rouge-score 计算得出的 ROUGE-1、ROUGE2 和 ROUGE-L，数据集示例如图 13。三个数据集的一些信息如图 10。

	#examples	input length	output length
E2E	50K	28.5	27.8
WebNLG	22K	49.6	30.7
DART	82K	38.8	27.3
XSUM	225K	473.3	28.1

图 10. 数据集

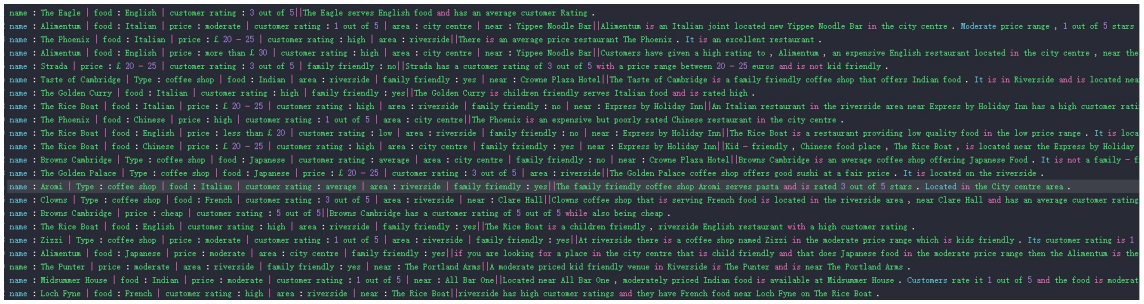

图 11. e2e 数据集

```
{
    "1": {
        "category": "Airport",
        "dbpedialinks": [],
        "lexicalisations": [
            {
                "comment": "good",
                "lang": "",
                "lex": "The Aarhus is the airport of Aarhus, Denmark.",
                "xml_id": "Id1"
            },
            {
                "comment": "good",
                "lang": "",
                "lex": "Aarhus Airport serves the city of Aarhus, Denmark.",
                "xml_id": "Id2"
            }
        ],
        "links": [],
        "modifiedtripleset": [
            {
                "object": "\"Aarhus, Denmark\"",
                "property": "cityServed",
                "subject": "Aarhus_Airport"
            }
        ],
        "originaltriplesets": {
            "originaltripleset": [
                [
                    {
                        "object": "\"Aarhus, Denmark\"@en",
                        "property": "cityServed",
                        "subject": "Aarhus_Airport"
                    }
                ]
            ]
```


图 12. WebNLG 数据集

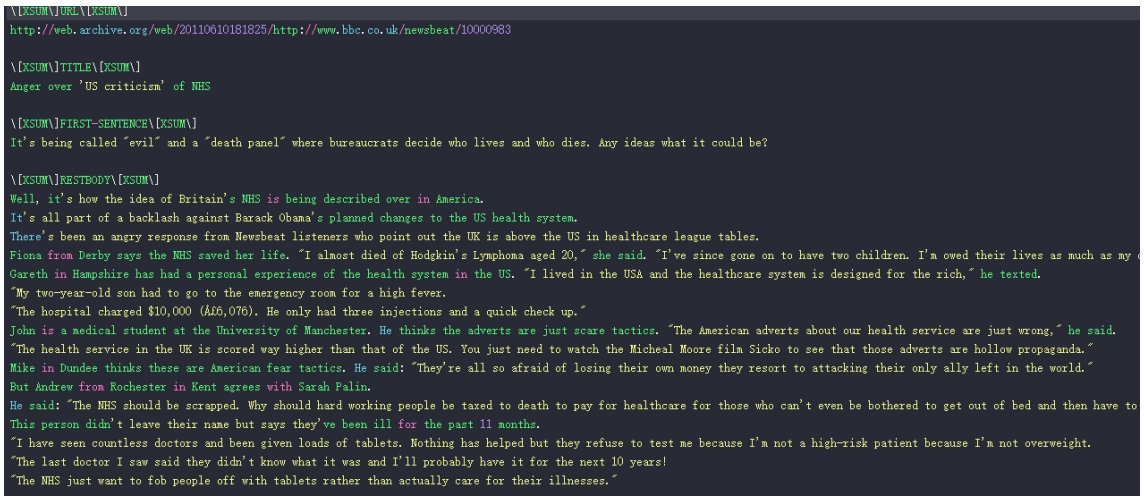

图 13. XSUM 数据集

指标计算。BLEU 即双语评估替补。所谓替补就是代替人类来评估机器翻译的每一个输出结果。BlEU 所做的，给定一个机器生成的翻译，自动计算一个分数，衡量机器翻译的好坏。取值范围是 [0, 1]，越接近 1，表明翻译质量越好，其计算公式为 $BLEU = BP * exp(\frac{1}{n}\sum_{i=1}^{N} P_n)$。METEOR 指标是基于 BLEU 进行了一些改进，其目的是解决一些 BLEU 标准中固有的缺陷。计算特定的序列匹配，同义词，词根和词缀，释义之间的匹配关系，改善了 BLEU 的效果，使其跟人工判别共更强的相关性。ROUGE 主要关注机器生成的文本中是否捕捉到了参考文本的信息，着重于涵盖参考文本的内容和信息的完整性。ROUGE 通过计算 N-gram 的共现情况，来评估机器生成内容的召回率。

数据预处理。对于" 表到文本"，我们将表 x 线性化，以适应语言模型语境。例如，在 E2E 数据集中，"（字段 A，值 A），（字段 B，值 B）" 被线性化为" 字段 A：值 A | 字段 B：值 B"。此外，在 WebNLG 和 DART 中，三重序列"（entity1.1, relation1, entity1.2），(entity2.1, relation2, entity2.2）" 被线性化为"entity1.1 ： relation1 ： entity1.2 | entity2.1 ： relation2 ： entity2.2"。对于摘要任务，我们将文章 x 截断为 512 个 BPE 标记。

5.2 Table-to-Text 生成任务

我们发现，只需更新 0.1% 的特定任务参数，前缀调整在表格到文本的生成中就很有效，即使更新的参数比其他轻量级基线（ADAPTER 和 FT-TOP2）少 30 倍，也能取得与（完全）微调相当的性能。这一趋势适用于所有数据集：E2E、WebNLG8 和 DART。如果我们将前缀调整和适配器调整的参数数匹配为 0.1%，表 1显示前缀调整明显优于 ADAPTER，平均每个数据集可提高 4.1BLEU。该实验中主要与 adapter、finetune、finetune-top 方法进行对比，主要是为了证明该方法在各种指标上比其他方法好。

表 1. Table-to-Text 任务

	E2E					WebNLG									DART		
	BLEU	NIST	METEOR	ROUGE_L	CIDEr	BLUE-S	BLUE-U	BLUE-A	MET-S	MET-U	MET-A	TER-S	TER-U	TER-A	BLUE	MET	TER
prefix	69.2	8.75	46	71.2	2.46	57.13	43.77	51.1	0.41	0.36	0.38	0.38	0.48	0.43	48.5	0.41	0.45
adaptertune	67.8	8.60	45.5	70.6	2.42	64.51	50.2	58	0.46	0.4	0.43	0.32	0.44	0.38	44.3	0.4	0.45
finetune	68.7	8.77	46	70.9	2.43	62.51	38.7	62	0.48	0.45	0.41	0.42	0.34	0.35	48.1	0.36	0.5
finetune-top	67.8	8.59	44.5	70.9	2.28	59.4	49.8	57.4	0.4	0.25	0.39	0.35	0.43	0.27	42.6	0.32	0.53

5.3 摘要生成任务

如表 2，前缀调整的性能略低于微调。XSUM 与三个表对表文本数据集之间存在一些差异，这也是为什么前缀调整在表对表文本中具有比较优势的原因：(1) XSUM 包含的示例数平均是三个表对表文本数据集的 4 倍；(2) 输入文章的长度平均是表对表文本数据集线性化表格输入长度的 17 倍；(3) 摘要比表对表文本更复杂，因为它需要从文章中选择关键内容。

表 2. 摘要生成任务

	R-1	R-2	R-L
prefix	41.3	18.6	35.1
finetune	44.2	20.1	38.2

5.4 低数据设置

当训练示例数量较少时，前缀微调具有相对优势。为了更系统地探索低数据设置，对 e2e 数据集进行了子采样，以获得大小为 50、100、200、500 的小型数据集。对于每种规模，我们抽取 5 个不同的数据集，并对 2 个训练随机种子进行平均。接着对不同小型数据集进行训练，并得到相应的指标分数。可以看到，在每一个指标中，前缀微调在低数据情况下，效果都比全量微调要好。但随着数据集规模的增大，差距也在缩小。虽然这两种方法在低数据量情况下都倾向于生成不足（缺失表格内容），但前缀调整往往比微调更忠实。例如，全量微调 FT（100，200）会错误地声称客户评分较低，而真实评分是平均值，而前缀调整（100，200）生成的描述则忠实于表格。

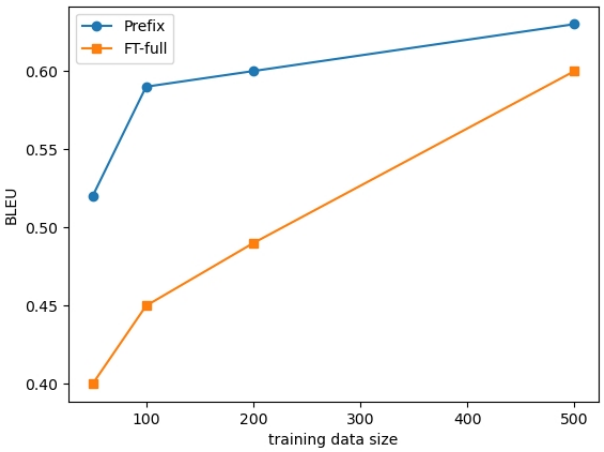

图 14. BLEU 指标

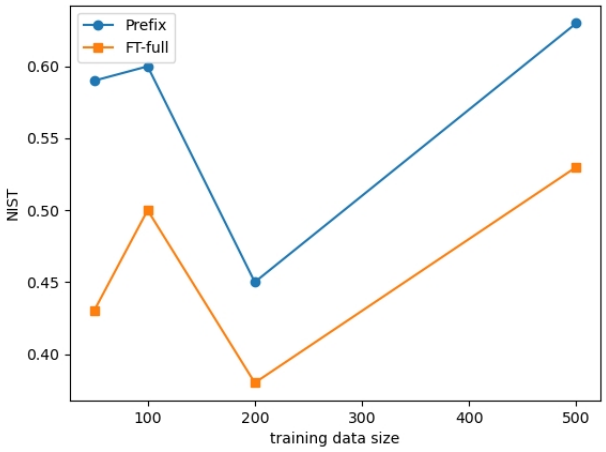

图 15. NIST 指标

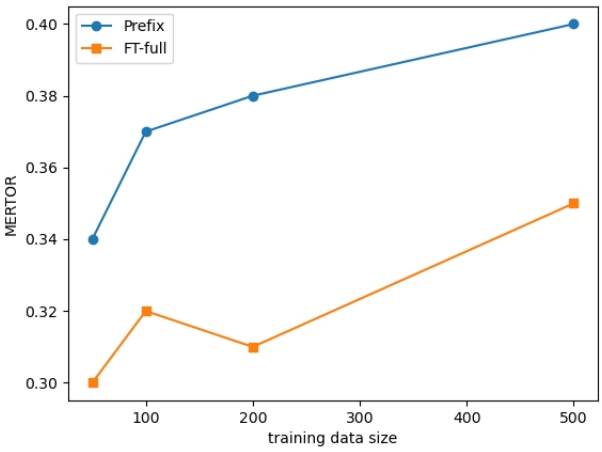

图 16. MERTOR 指标

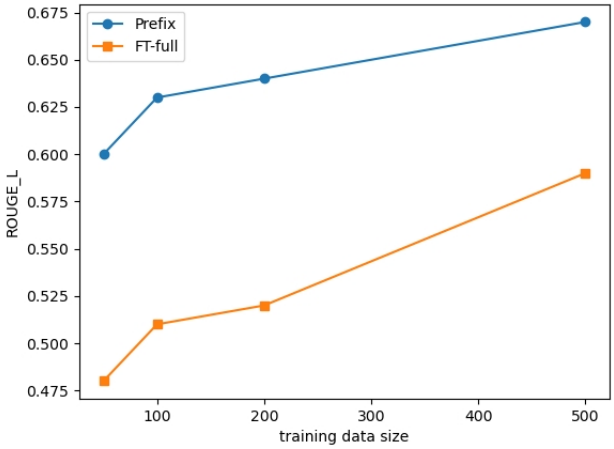

图 17. ROUGE 指标

Source	name : The Eagle | type : coffee shop | food : Chinese | price : cheap | customer rating : average | area : riverside | family friendly : no | near : Burger King
Prefix (50)	The Eagle is a cheap Chinese coffee shop located near Burger King.
Prefix (100)	The Eagle is a cheap coffee shop located in the riverside near Burger King. It has average customer ratings.
Prefix (200)	The Eagle is a cheap Chinese coffee shop located in the riverside area near Burger King. It has average customer ratings.
Prefix (500)	The Eagle is a coffee shop that serves Chinese food. It is located in the riverside area near Burger King. It has an average customer rating and is not family friendly.
FT (50)	The Eagle coffee shop is located in the riverside area near Burger King.
FT (100)	The Eagle is a cheap coffee shop near Burger King in the riverside area. It has a low customer rating and is not family friendly.
FT (200)	The Eagle is a cheap Chinese coffee shop with a low customer rating. It is located near Burger King in the riverside area.
FT (500)	The Eagle is a cheap Chinese coffee shop with average customer ratings. It is located in the riverside area near Burger King.

图 18. 低数据情况下生成的例子

5.5 Embedding-only and prefix

Embedding-only 表示只微调嵌入层，而不在每一个 transformers 层的激活函数前插入向量，而表 3显示，性能显著下降，表明只调整嵌入层的表现力不够，只调整嵌入层就限制了离散提示优化的性能上限。

同时研究了前缀向量在序列中的位置对性能的影响。在前缀调整中，我们将它们放在开头 $[PREFIX; x; y]$。我们也可以将可向量放在 x 和 y 之间（即 $[x; INFIX; y]$），并将其称为中缀调整。如表 3显示，infix-tuning 略逊于 prefix-tuning。我们认为这是因为前缀调整可以影响 x 和 y 的激活，而后缀调整只能影响 y 的激活。

表 3. Embedding and infix

	BLEU	NIST	MET	ROUGE	CIDEr
PERFIX	69.2	8.75	46	71.2	2.46
EMB-1	17.5	2.56	17.2	33.2	0.33
EMB-10	61.2	6.82	38.7	65.6	1.78
EMB-20	61.3	6.92	38.6	65.4	1.81
INFIX-1	65.6	8.34	45.3	69.1	2.38
INFIX-10	68.1	8.57	46.4	70.7	2.42
INFIX-20	66.8	8.48	45.9	70.1	2.43

6 总结与展望

本报告系统地阐述了前缀微调的运行机制、复现细节以及实验设置。在相关工作章节中总结了现有的微调方法，并进行分类，找出前缀微调所属的类别，易于理解原理。在本文方法章节中主要阐述了前缀微调的原理，并比较它与其他微调方法的本质区别，说明模型更新参数的方式。在复现细节章节中主要说明了论文开源代码中所遇到的问题，并说明如何解决的，同时也说明实验环境搭建。在实验结果分析章节主要阐述我复现代码中所得出的一些实验结果，覆盖论文的大部分实验。实验中的不足是超参数设置得不够合理，后续可以继续更改超参数进行实验。未来的研究方向：（1）本文是为每一个下游任务训练一个模型参数，那可以考虑是否可以为在模型参数中设置某一小块参数是属于用户的，以致于每一个用户相当于有一个属于自己的模型参数，用户能更加贴合地使用模型。（2）跨用户批处理，在相同的个性化设置下，即使不同用户的查询有不同的前缀支持，前缀调整也可以对这些查询进行批处理。当多个用户向云 GPU 设备查询其输入时，将这些用户归入同一批次会提高计算效率。

参考文献

[1] Luke Zettlemoyer Armen Aghajanyan and Sonal Gupta. Intrinsic dimensionality explains the effectiveness of language model fine-tuning. 2020.

[2] Anja Belz and Ehud Reiter. Comparing automatic and human evaluation of nlg systems. 2006.

[3] Chin-Yew. Rouge: A package for automatic evaluation of summaries. 2004.

[4] Shashi Narayan Claire Gardent, Anastasia Shimorina and Laura Perez-Beltrachini. The webnlg challenge: Generating text from rdf data. 2017.

[5] Amrit Rau Abhinand Sivaprasad Chiachun Hsieh Nazneen Fatema Rajani Xiangru Tang Aadit Vyas Neha Verma Pranav Krishna Yangxiaokang Liu Nadia Irwanto Jessica Pan Faiaz Rahman Ahmad Zaidi Murori Mutuma Yasin Tarabar Ankit Gupta Tao Yu Yi Chern Tan Xi Victoria Lin Caiming Xiong Dragomir Radev, Rui Zhang and Richard Socher. Dart: Open-domain structured data record to text generation. 2020.

[6] Ondrej Dusek Jekaterina Novikova and Verena Rieser. The e2e dataset: New challenges for end-toend generation. 2017.

[7] Andreas R¨ uckl´ e Kyunghyun Cho Jonas Pfeiffer, Aishwarya Kamath and Iryna Gurevych. Adapterfusion: Non-destructive task composition for transfer learning. 2020.

[8] Todd Ward Kishore Papineni, Salim Roukos and WeiJing Zhu. Bleu: A method for automatic evaluation of machine translation. 2002.

[9] Alon Lavie and Abhaya Agarwal. Meteor: An automatic metric for mt evaluation with high levels of correlation with human judgments. 2007.

[10] L. R. Varshney Caiming Xiong N. Keskar, B. McCann and R. Socher. A conditional transformer language model for controllable generation. 2019.

[11] C. Lawrence Zitnick Ramakrishna Vedantam and Devi Parikh. Cider: Consensus-based image description evaluation. 2015.

[12] Robert L. Logan IV au2 Eric Wallace Taylor Shin, Yasaman Razeghi and Sameer Singh. Autoprompt: Eliciting knowledge from language models with automatically generated prompts. 2020.

[13] Anna Rumshisky Vladislav Lialin, Vijeta Deshpande. Scaling down to scale up: A guide to parameter-efficient fine-tuning. 2023.

[14] Varun Nair Yi-Lin Sung and Colin A Raffel. Training neural networks with fixed sparse masks. 2021.