Prompt Tuning 微调实战

为了不影响阅读体验，详细的代码放置在GitHub：llm-action 项目中 peft_prompt_tuning_clm.ipynb文件，这里仅列出关键步骤。

第一步，引进必要的库，如：Prompt Tuning 配置类 PromptTuningConfig。

```
Plain Text
```

```
from peft import get_peft_config, get_peft_model, PromptTuningInit, PromptTuningConfig, TaskType, PeftType
```

第二步，创建 Prompt Tuning 微调方法对应的配置。

```
Plain Text
```

```
peft_config = PromptTuningConfig(
    task_type=TaskType.CAUSAL_LM,
    prompt_tuning_init=PromptTuningInit.TEXT,
    num_virtual_tokens=8,
    prompt_tuning_init_text="Classify if the tweet is a complaint or not:",
    tokenizer_name_or_path=model_name_or_path,
)
```

参数说明：

- prompt_tuning_init：提示嵌入的初始化方法。PEFT支持文本（TEXT）和随机（RANDOM）初始化。在原理篇中提到过 Prompt token 的初始化方法和长度对于模型性能有影响。与随机初始化和使用样本词汇表初始化相比，Prompt Tuning 采用类标签初始化模型的效果更好。不过随着模型参数规模的提升，这种gap最终会消失。因此，如果需要使用类标签和样本词汇表初始化需指定为TEXT。
- prompt_tuning_init_text：用于初始化提示嵌入的文本，在使用文本（TEXT）初始化方法时使用。
- task_type：指定任务类型。如：条件生成任务（SEQ_2_SEQ_LM），因果语言建模（CAUSAL_LM）等。
- num_virtual_tokens：指定虚拟Token数。在原理篇中，提到过提示虚拟 Token 的长度在20左右时的表现已经不错（超过20之后，提升Prompt token长度，对模型的性能提升不明显了）；同样的，这个gap也会随着模型参数规模的提升而减小（即对于超大规模模型而言，即使提示虚拟 Token 长度很短，对性能也不会有太大的影响）。

第三步，通过调用 get_peft_model 方法包装基础的 Transformer 模型。

```
model = AutoModelForCausalLM.from_pretrained(model_name_or_path)
model = get_peft_model(model, peft_config)
model.print_trainable_parameters()
```

通过 print_trainable_parameters 方法可以查看可训练参数的数量(仅为8,192)以及占比（仅为0.00146%）。

```
trainable params: 8,192 || all params: 559,222,784 || trainable%: 0.0014648902430985358
```

Prompt Tuning 模型类结构如下所示：

```
PeftModelForCausalLM(
  (base_model): BloomForCausalLM(
    (transformer): BloomModel(
      (word_embeddings): Embedding(250880, 1024)
      (word_embeddings_layernorm): LayerNorm((1024,), eps=1e-05, elementwise_affine=True)
      (h): ModuleList(
        ...
      )
      (ln_f): LayerNorm((1024,), eps=1e-05, elementwise_affine=True)
    )
    (lm_head): Linear(in_features=1024, out_features=250880, bias=False)
  )
  (prompt_encoder): ModuleDict(
    (default): PromptEmbedding(
      (embedding): Embedding(8, 1024)
    )
  )
  (word_embeddings): Embedding(250880, 1024)
)
```

从模型类结构可以看到，Prompt Tuning 只在输入层加入 prompt virtual tokens，其他地方均没有变化，具体可查看 PromptEmbedding 的源码。

```
class PromptEmbedding(torch.nn.Module):
    def __init__(self, config, word_embeddings):
        super().__init__()

        total_virtual_tokens = config.num_virtual_tokens * config.num_transformer_submodules
        # 初始化 embedding 层
        self.embedding = torch.nn.Embedding(total_virtual_tokens, config.token_dim)

        # 如果使用文本进行初始化，执行如下逻辑，PromptTuningConfig 配置类需要传入初始化文本。
        if config.prompt_tuning_init == PromptTuningInit.TEXT:
            from transformers import AutoTokenizer

            tokenizer = AutoTokenizer.from_pretrained(config.tokenizer_name_or_path)
            init_text = config.prompt_tuning_init_text
            init_token_ids = tokenizer(init_text)["input_ids"]
            # Trim or iterate until num_text_tokens matches total_virtual_tokens
            num_text_tokens = len(init_token_ids)
            if num_text_tokens > total_virtual_tokens:
                init_token_ids = init_token_ids[:total_virtual_tokens]
            elif num_text_tokens < total_virtual_tokens:
                num_reps = math.ceil(total_virtual_tokens / num_text_tokens)
                init_token_ids = init_token_ids * num_reps
            init_token_ids = init_token_ids[:total_virtual_tokens]

            word_embedding_weights = word_embeddings(torch.LongTensor(init_token_ids)).detach().clone()
            word_embedding_weights = word_embedding_weights.to(torch.float32)
            # 初始化embedding层的权重
            self.embedding.weight = torch.nn.Parameter(word_embedding_weights)

    def forward(self, indices):
        # Just get embeddings
        prompt_embeddings = self.embedding(indices)
        return prompt_embeddings
```

第四步，模型训练的其余部分均无需更改，当模型训练完成之后，保存高效微调部分的模型权重以供模型推理即可。

```
Plain Text
```

```
peft_model_id = f"{model_name_or_path}_{peft_config.peft_type}_{peft_config.task_type}"
model.save_pretrained(peft_model_id)
```

输出的模型权重文件如下所示：

```
Plain Text
```

```
/data/nfs/llm/model/bloomz-560m_PROMPT_TUNING_CAUSAL_LM
├── [ 500]  adapter_config.json
├── [ 33K]  adapter_model.bin
└── [ 111]  README.md

0 directories, 3 files
```

注意：这里只会保存经过训练的增量 PEFT 权重。其中，adapter_config.json 为 Prompt Tuning 配置文件；adapter_model.bin 为 Prompt Tuning 权重文件。

第五步，加载微调后的权重文件进行推理。

```
from peft import PeftModel, PeftConfig

peft_model_id = f"{model_name_or_path}_{peft_config.peft_type}_{peft_config.task_type}"

# 加载PEFT配置
config = PeftConfig.from_pretrained(peft_model_id)

# 加载基础模型
model = AutoModelForCausalLM.from_pretrained(config.base_model_name_or_path)
# 加载PEFT模型
model = PeftModel.from_pretrained(model, peft_model_id)

# Tokenizer编码
inputs = tokenizer(f'{text_column} : {dataset["test"][i]["Tweet text"]} Label : ', return_tensors="pt")

# 模型推理
outputs = model.generate(
        input_ids=inputs["input_ids"],
        attention_mask=inputs["attention_mask"],
        max_new_tokens=10,
        eos_token_id=3
    )

# Tokenizer 解码
print(tokenizer.batch_decode(outputs.detach().cpu().numpy(), skip_special_tokens=True))
```

至此，我们完成了Prompt Tuning的训练及推理。

Python

```python
# coding=utf-8
# Copyright 2023-present the HuggingFace Inc. team.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.

import math

import torch

from .config import PromptTuningInit


class PromptEmbedding(torch.nn.Module):
    """
    The model to encode virtual tokens into prompt embeddings.

    Args:
        config ([`PromptTuningConfig`]): The configuration of the prompt e
mbedding.
        word_embeddings (`torch.nn.Module`): The word embeddings of the ba
se transformer model.

    **Attributes**:
        - **embedding** (`torch.nn.Embedding`) -- The embedding layer of t
he prompt embedding.

    Example:

    ```py
    >>> from peft import PromptEmbedding, PromptTuningConfig

    >>> config = PromptTuningConfig(
    ...     peft_type="PROMPT_TUNING",
    ...     task_type="SEQ_2_SEQ_LM",
    ...     num_virtual_tokens=20,
```
```

```
    ...     token_dim=768,
    ...     num_transformer_submodules=1,
    ...     num_attention_heads=12,
    ...     num_layers=12,
    ...     prompt_tuning_init="TEXT",
    ...     prompt_tuning_init_text="Predict if sentiment of this review is positive, negative or neutral",
    ...     tokenizer_name_or_path="t5-base",
    ... )

    >>> # t5_model.shared is the word embeddings of the base model
    >>> prompt_embedding = PromptEmbedding(config, t5_model.shared)
    ```

    Input Shape: (`batch_size`, `total_virtual_tokens`)

    Output Shape: (`batch_size`, `total_virtual_tokens`, `token_dim`)
    """

    def __init__(self, config, word_embeddings):
        super().__init__()

        total_virtual_tokens = config.num_virtual_tokens * config.num_transformer_submodules
        self.embedding = torch.nn.Embedding(total_virtual_tokens, config.token_dim)
        if config.prompt_tuning_init == PromptTuningInit.TEXT and not config.inference_mode:
            from transformers import AutoTokenizer

            tokenizer_kwargs = config.tokenizer_kwargs or {}
            tokenizer = AutoTokenizer.from_pretrained(config.tokenizer_name_or_path, **tokenizer_kwargs)
            init_text = config.prompt_tuning_init_text
            init_token_ids = tokenizer(init_text)["input_ids"]
            # Trim or iterate until num_text_tokens matches total_virtual_tokens
            num_text_tokens = len(init_token_ids)
            if num_text_tokens > total_virtual_tokens:
                init_token_ids = init_token_ids[:total_virtual_tokens]
            elif num_text_tokens < total_virtual_tokens:
                num_reps = math.ceil(total_virtual_tokens / num_text_tokens)
                init_token_ids = init_token_ids * num_reps
            init_token_ids = init_token_ids[:total_virtual_tokens]
            init_token_ids = torch.LongTensor(init_token_ids).to(word_embeddings.weight.device)
```
```

```
            word_embedding_weights = word_embeddings(init_token_ids).detach().clone()
            word_embedding_weights = word_embedding_weights.to(torch.float32)
            self.embedding.weight = torch.nn.Parameter(word_embedding_weights)

    def forward(self, indices):
        # Just get embeddings
        prompt_embeddings = self.embedding(indices)
        return prompt_embeddings
```