实验 4：注意力机制与 Transformer

2024 年 4 月 7 日

1 实验简介

本实验介绍利用 PyTorch 构建 Transformer 模型的方法。在实验中，我们将利用 PyTorch 内置的 Transformer 模块搭建一个 Seq2Seq 网络，并利用该模型构建一个德语到英语的翻译器。通过本实验，你将掌握以下技能：

- 基于注意力机制和 Transformer 进行模型构建的方法
- 机器翻译模型的建模、训练方法
- 利用 `torchtext` 等工具进行文本数据处理的基本方法

2 实验环境准备

首先，你需要安装完成本实验所需的必要组件。

2.1 安装 Miniconda

从以下链接下载 Miniconda 安装包并完成安装[Link]，安装过程中需要选中 `Register Anaconda as my default Python 3.10`。若计算机上已经安装好 Miniconda 或 Anaconda，请跳过此步骤。

2.2 安装 Jupyter Notebook

在 Anaconda Prompt 中执行 `pip install notebook`，以安装 Jupyter Notebook。

安装完成后，在 Anaconda Prompt 执行 `jupyter notebook`，检查是否能够成功启动 Notebook。

若计算机上已经安装好 Jupyter Notebook，请跳过此步骤。

2.3 安装必要的 Package

你可以使用以下代码在 Jupyter Notebook 中检查并安装必要的包，也可事先在 Anaconda Prompt 中通过 `pip` 或 `conda` 安装好。

[1]:
```
import pkgutil

# 检查 PyTorch 是否已安装
if pkgutil.find_loader('torch'):
    print('PyTorch is available.')
else:
    !pip install torch -i https://pypi.tuna.tsinghua.edu.cn/simple/

# 检查 torchtext 是否已安装
if pkgutil.find_loader('torchtext'):
    print('torchtext is available.')
else:
    !pip install torchtext -i https://pypi.tuna.tsinghua.edu.cn/simple/

# 检查 torchdata 是否已安装
if pkgutil.find_loader('torchdata'):
    print('torchdata is available.')
else:
    !pip install torchdata -i https://pypi.tuna.tsinghua.edu.cn/simple/

# 检查 spaCy 是否已安装
if pkgutil.find_loader('spacy'):
    print('spacy is available.')
else:
    !pip install spacy -i https://pypi.tuna.tsinghua.edu.cn/simple/

# 检查 portalocker 是否已安装
if pkgutil.find_loader('portalocker'):
    print('portalocker is available.')
else:
```

```
!pip install portalocker -i https://pypi.tuna.tsinghua.edu.cn/simple/
```

```
PyTorch is available.
torchtext is available.
torchdata is available.
spacy is available.
portalocker is available.
```

[2]:
```
# 安装 spaCy 英语、德语处理工具
import spacy

try:
    spacy.load('en_core_web_sm')
    print('en_core_web_sm is available.')
except:
    print('Downloading en_core_web_sm ...')
    !python -m spacy download en_core_web_sm

try:
    spacy.load('de_core_news_sm')
    print('de_core_news_sm is available.')
except:
    print('Downloading de_core_news_sm ...')
    !python -m spacy download de_core_news_sm
```

```
en_core_web_sm is available.
de_core_news_sm is available.
```

[3]:
```
import warnings
warnings.filterwarnings('ignore')
```

3 数据源及数据预处理

本实验将使用 Multi30k 数据集。Multi30k 是一个大规模多语言多模态数据集，包含约 30000 张图片及平行的英语、德语和法语图片描述。在本实验中，我们仅使用英语、德语文本数据。

首先，指定数据源 URL 及源语言、目标语言。

[4]:
```
from torchtext.data.utils import get_tokenizer
from torchtext.vocab import build_vocab_from_iterator
from torchtext.datasets import multi30k, Multi30k
from typing import Iterable, List

multi30k.URL["train"] = "https://raw.githubusercontent.com/neychev/
 ↪small_DL_repo/master/datasets/Multi30k/training.tar.gz"
multi30k.URL["valid"] = "https://raw.githubusercontent.com/neychev/
 ↪small_DL_repo/master/datasets/Multi30k/validation.tar.gz"

SRC_LANGUAGE = 'de'
TGT_LANGUAGE = 'en'

token_transform = {}
vocab_transform = {}
```

接下来，创建德语、英语 Tokenizer。Tokenizer 是自然语言处理的基础组件，用于将句子切分成用于处理的基本单位——token。

[5]:
```
token_transform[SRC_LANGUAGE] = get_tokenizer('spacy',␣
 ↪language='de_core_news_sm')
token_transform[TGT_LANGUAGE] = get_tokenizer('spacy',␣
 ↪language='en_core_web_sm')
```

进行 Token 化，生成 `torchtext` 的 Vocab 对象。

[6]:
```
# 产生 token 列表的工具函数
def yield_tokens(data_iter: Iterable, language: str) -> List[str]:
    language_index = {SRC_LANGUAGE: 0, TGT_LANGUAGE: 1}

    for data_sample in data_iter:
        yield token_transform[language](data_sample[language_index[language]])

# 定义特殊 token
UNK_IDX, PAD_IDX, BOS_IDX, EOS_IDX = 0, 1, 2, 3
special_symbols = ['<unk>', '<pad>', '<bos>', '<eos>']

for ln in [SRC_LANGUAGE, TGT_LANGUAGE]:
```

```
    # 训练数据迭代器
    train_iter = Multi30k(split='train', language_pair=(SRC_LANGUAGE,␣
 ↪TGT_LANGUAGE))
    # 创建 Vocab 对象
    vocab_transform[ln] = build_vocab_from_iterator(yield_tokens(train_iter,␣
 ↪ln),
                                                    min_freq=1,
                                                    specials=special_symbols,
                                                    special_first=True)

for ln in [SRC_LANGUAGE, TGT_LANGUAGE]:
    vocab_transform[ln].set_default_index(UNK_IDX)
```

你可以通过以下代码段查看一些“源语言-目标语言”语句对的样例。

[21]:
```
from torch.utils.data import DataLoader

example_dataloader = DataLoader(train_iter, batch_size=5)
for batch in example_dataloader:
    print(batch)
    break  # 仅查看 1 个 batch
```

```
[('Zwei junge weiße Männer sind im Freien in der Nähe vieler Büsche.', 'Mehrere
Männer mit Schutzhelmen bedienen ein Antriebsradsystem.', 'Ein kleines Mädchen
klettert in ein Spielhaus aus Holz.', 'Ein Mann in einem blauen Hemd steht auf
einer Leiter und putzt ein Fenster.', 'Zwei Männer stehen am Herd und bereiten
Essen zu.'), ('Two young, White males are outside near many bushes.', 'Several
men in hard hats are operating a giant pulley system.', 'A little girl climbing
into a wooden playhouse.', 'A man in a blue shirt is standing on a ladder
cleaning a window.', 'Two men are at the stove preparing food.')]
```

4 创建模型

Transformer 是一种 Seq2Seq 模型，其最早是为了解决机器翻译问题而提出的。我们利用 PyTorch 自带的 Transformer 模块创建模型，该模型包含三个部分：

- Embedding 层：将输入的文本数据进行 Embedding，并加入位置编码以提供明确的位置信息；

- Transformer 模型：即 Transformer 编码器-解码器架构；
- 线性层：输出目标语言每个 token 的概率分布。

[6]:
```
from torch import Tensor
import torch
import torch.nn as nn
from torch.nn import Transformer
import math
DEVICE = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

# 用于创建位置编码，编码方式采用论文 Attention is all you need 中提出的方法
class PositionalEncoding(nn.Module):
    def __init__(self,
                 emb_size: int,
                 dropout: float,
                 maxlen: int = 5000):
        super(PositionalEncoding, self).__init__()
        den = torch.exp(- torch.arange(0, emb_size, 2)* math.log(10000) /␣
 ↪emb_size)
        pos = torch.arange(0, maxlen).reshape(maxlen, 1)
        pos_embedding = torch.zeros((maxlen, emb_size))
        pos_embedding[:, 0::2] = torch.sin(pos * den)
        pos_embedding[:, 1::2] = torch.cos(pos * den)
        pos_embedding = pos_embedding.unsqueeze(-2)

        self.dropout = nn.Dropout(dropout)
        self.register_buffer('pos_embedding', pos_embedding)

    def forward(self, token_embedding: Tensor):
        return self.dropout(token_embedding + self.pos_embedding[:
 ↪token_embedding.size(0), :])

# 进行 Embedding 操作
class TokenEmbedding(nn.Module):
    def __init__(self, vocab_size: int, emb_size):
        super(TokenEmbedding, self).__init__()
        self.embedding = nn.Embedding(vocab_size, emb_size)
```

```
        self.emb_size = emb_size

    def forward(self, tokens: Tensor):
        return self.embedding(tokens.long()) * math.sqrt(self.emb_size)

# 定义 Transformer 模型
class Seq2SeqTransformer(nn.Module):
    def __init__(self,
                 num_encoder_layers: int,
                 num_decoder_layers: int,
                 emb_size: int,
                 nhead: int,
                 src_vocab_size: int,
                 tgt_vocab_size: int,
                 dim_feedforward: int = 512,
                 dropout: float = 0.1):
        super(Seq2SeqTransformer, self).__init__()
        self.transformer = Transformer(d_model=emb_size,
                                       nhead=nhead,
                                       num_encoder_layers=num_encoder_layers,
                                       num_decoder_layers=num_decoder_layers,
                                       dim_feedforward=dim_feedforward,
                                       dropout=dropout)
        self.generator = nn.Linear(emb_size, tgt_vocab_size)
        self.src_tok_emb = TokenEmbedding(src_vocab_size, emb_size)
        self.tgt_tok_emb = TokenEmbedding(tgt_vocab_size, emb_size)
        self.positional_encoding = PositionalEncoding(
            emb_size, dropout=dropout)

    def forward(self,
                src: Tensor,
                trg: Tensor,
                src_mask: Tensor,
                tgt_mask: Tensor,
                src_padding_mask: Tensor,
                tgt_padding_mask: Tensor,
```

```
                memory_key_padding_mask: Tensor):
        src_emb = self.positional_encoding(self.src_tok_emb(src))
        tgt_emb = self.positional_encoding(self.tgt_tok_emb(trg))
        outs = self.transformer(src_emb, tgt_emb, src_mask, tgt_mask, None,
                                src_padding_mask, tgt_padding_mask,␣
↪memory_key_padding_mask)
        return self.generator(outs)

    def encode(self, src: Tensor, src_mask: Tensor):
        return self.transformer.encoder(self.positional_encoding(
                            self.src_tok_emb(src)), src_mask)

    def decode(self, tgt: Tensor, memory: Tensor, tgt_mask: Tensor):
        return self.transformer.decoder(self.positional_encoding(
                          self.tgt_tok_emb(tgt)), memory,
                          tgt_mask)
```

在训练过程中，我们需要对后续的单词进行遮盖，避免模型看到。同时，我们也需要遮盖住 Padding token。这里定义相关的掩膜函数用于实现上述功能。

[7]:
```
def generate_square_subsequent_mask(sz):
    mask = (torch.triu(torch.ones((sz, sz), device=DEVICE)) == 1).transpose(0,␣
↪1)
    mask = mask.float().masked_fill(mask == 0, float('-inf')).masked_fill(mask␣
↪== 1, float(0.0))
    return mask


def create_mask(src, tgt):
    src_seq_len = src.shape[0]
    tgt_seq_len = tgt.shape[0]

    tgt_mask = generate_square_subsequent_mask(tgt_seq_len)
    src_mask = torch.zeros((src_seq_len, src_seq_len),device=DEVICE).type(torch.
↪bool)

    src_padding_mask = (src == PAD_IDX).transpose(0, 1)
```

```
    tgt_padding_mask = (tgt == PAD_IDX).transpose(0, 1)
    return src_mask, tgt_mask, src_padding_mask, tgt_padding_mask
```

定义模型超参数。

[10]:
```
torch.manual_seed(0)

SRC_VOCAB_SIZE = len(vocab_transform[SRC_LANGUAGE])
TGT_VOCAB_SIZE = len(vocab_transform[TGT_LANGUAGE])
EMB_SIZE = 512
NHEAD = 8
FFN_HID_DIM = 512
BATCH_SIZE = 128
NUM_ENCODER_LAYERS = 3
NUM_DECODER_LAYERS = 3

transformer = Seq2SeqTransformer(NUM_ENCODER_LAYERS, NUM_DECODER_LAYERS,␣
 ↪EMB_SIZE,
                                 NHEAD, SRC_VOCAB_SIZE, TGT_VOCAB_SIZE,␣
 ↪FFN_HID_DIM)

for p in transformer.parameters():
    if p.dim() > 1:
        nn.init.xavier_uniform_(p)

transformer = transformer.to(DEVICE)

loss_fn = torch.nn.CrossEntropyLoss(ignore_index=PAD_IDX)

optimizer = torch.optim.Adam(transformer.parameters(), lr=0.0001, betas=(0.9, 0.
 ↪98), eps=1e-9)
```

5 数据规整

通过之前定义的数据迭代器，只能得到源语言、目标语言的原始文本。我们需要对数据迭代器产生的数据进行处理，得到以 Batch 为单位的 Tensor，以便数据能够直接输入模型中。

[12]:
```python
from torch.nn.utils.rnn import pad_sequence

# 用于组合多个数据变换的工具函数
def sequential_transforms(*transforms):
    def func(txt_input):
        for transform in transforms:
            txt_input = transform(txt_input)
        return txt_input
    return func

# 添加 BOS/EOS, 创建 Tensor
def tensor_transform(token_ids: List[int]):
    return torch.cat((torch.tensor([BOS_IDX]),
                      torch.tensor(token_ids),
                      torch.tensor([EOS_IDX])))

# 将原始文本转换为 Tensor
text_transform = {}
for ln in [SRC_LANGUAGE, TGT_LANGUAGE]:
    text_transform[ln] = sequential_transforms(token_transform[ln],
                                               vocab_transform[ln],
                                               tensor_transform)

# 将数据规整为 Batch Tensor
def collate_fn(batch):
    src_batch, tgt_batch = [], []
    for src_sample, tgt_sample in batch:
        src_batch.append(text_transform[SRC_LANGUAGE](src_sample.rstrip("\n")))
        tgt_batch.append(text_transform[TGT_LANGUAGE](tgt_sample.rstrip("\n")))

    src_batch = pad_sequence(src_batch, padding_value=PAD_IDX)
    tgt_batch = pad_sequence(tgt_batch, padding_value=PAD_IDX)
    return src_batch, tgt_batch
```

6 定义训练和评估函数

这里将训练及评估过程封装成函数，方便后续模型训练。在训练过程中，首先定义数据加载器，随后创建 mask 以构建模型输入，最后通过反向传播进行单步训练的更新。

[13]:
```
from torch.utils.data import DataLoader

def train_epoch(model, optimizer):
    model.train()
    losses = 0
    train_iter = Multi30k(split='train', language_pair=(SRC_LANGUAGE,␣
 ↪TGT_LANGUAGE))
    train_dataloader = DataLoader(train_iter, batch_size=BATCH_SIZE,␣
 ↪collate_fn=collate_fn)

    for src, tgt in train_dataloader:
        src = src.to(DEVICE)
        tgt = tgt.to(DEVICE)
        tgt_input = tgt[:-1, :]
        src_mask, tgt_mask, src_padding_mask, tgt_padding_mask =␣
 ↪create_mask(src, tgt_input)
        logits = model(src, tgt_input, src_mask, tgt_mask, src_padding_mask,␣
 ↪tgt_padding_mask, src_padding_mask)
        optimizer.zero_grad()
        tgt_out = tgt[1:, :]
        loss = loss_fn(logits.reshape(-1, logits.shape[-1]), tgt_out.
 ↪reshape(-1))
        loss.backward()
        optimizer.step()
        losses += loss.item()

    return losses / len(list(train_dataloader))


def evaluate(model):
    model.eval()
```

```
    losses = 0
    val_iter = Multi30k(split='valid', language_pair=(SRC_LANGUAGE,␣
↪TGT_LANGUAGE))
    val_dataloader = DataLoader(val_iter, batch_size=BATCH_SIZE,␣
↪collate_fn=collate_fn)

    for src, tgt in val_dataloader:
        src = src.to(DEVICE)
        tgt = tgt.to(DEVICE)
        tgt_input = tgt[:-1, :]
        src_mask, tgt_mask, src_padding_mask, tgt_padding_mask =␣
↪create_mask(src, tgt_input)
        logits = model(src, tgt_input, src_mask, tgt_mask,src_padding_mask,␣
↪tgt_padding_mask, src_padding_mask)
        tgt_out = tgt[1:, :]
        loss = loss_fn(logits.reshape(-1, logits.shape[-1]), tgt_out.
↪reshape(-1))
        losses += loss.item()

    return losses / len(list(val_dataloader))
```

利用以上定义的函数，训练模型。你可以修改 NUM_EPOCHS 以调整训练所需时间。在使用 CPU 训练的情况下，每个 epoch 约需 10 分钟。

[14]:
```
from timeit import default_timer as timer
NUM_EPOCHS = 10

for epoch in range(1, NUM_EPOCHS+1):
    start_time = timer()
    train_loss = train_epoch(transformer, optimizer)
    end_time = timer()
    val_loss = evaluate(transformer)
    print((f"Epoch: {epoch}, Train loss: {train_loss:.3f}, Val loss: {val_loss:.
↪3f}, "f"Epoch time = {(end_time - start_time):.3f}s"))
```

```
Epoch: 1, Train loss: 5.344, Val loss: 4.103, Epoch time = 490.643s
Epoch: 2, Train loss: 3.759, Val loss: 3.311, Epoch time = 470.126s
```

```
Epoch: 3, Train loss: 3.159, Val loss: 2.892, Epoch time = 457.676s
Epoch: 4, Train loss: 2.768, Val loss: 2.644, Epoch time = 461.359s
Epoch: 5, Train loss: 2.479, Val loss: 2.444, Epoch time = 460.422s
Epoch: 6, Train loss: 2.252, Val loss: 2.314, Epoch time = 459.568s
Epoch: 7, Train loss: 2.063, Val loss: 2.196, Epoch time = 458.628s
Epoch: 8, Train loss: 1.900, Val loss: 2.112, Epoch time = 457.921s
Epoch: 9, Train loss: 1.758, Val loss: 2.064, Epoch time = 459.461s
Epoch: 10, Train loss: 1.636, Val loss: 2.018, Epoch time = 459.066s
```

可通过以下代码保存模型权重。

[17]:
```python
# 保存模型权重
PATH = './transformer.pth'
torch.save(transformer.state_dict(), PATH)
```

7 模型测试

定义一些工具函数，以便进行机器翻译测试。

[15]:
```python
# 采用贪心算法生成输出序列
def greedy_decode(model, src, src_mask, max_len, start_symbol):
    src = src.to(DEVICE)
    src_mask = src_mask.to(DEVICE)

    memory = model.encode(src, src_mask)
    ys = torch.ones(1, 1).fill_(start_symbol).type(torch.long).to(DEVICE)
    for i in range(max_len-1):
        memory = memory.to(DEVICE)
        tgt_mask = (generate_square_subsequent_mask(ys.size(0))
                    .type(torch.bool)).to(DEVICE)
        out = model.decode(ys, memory, tgt_mask)
        out = out.transpose(0, 1)
        prob = model.generator(out[:, -1])
        _, next_word = torch.max(prob, dim=1)
        next_word = next_word.item()

        ys = torch.cat([ys,
```

```
                        torch.ones(1, 1).type_as(src.data).fill_(next_word)],␣
 ↪dim=0)
        if next_word == EOS_IDX:
            break
    return ys


# 翻译测试函数
def translate(model: torch.nn.Module, src_sentence: str):
    model.eval()
    src = text_transform[SRC_LANGUAGE](src_sentence).view(-1, 1)
    num_tokens = src.shape[0]
    src_mask = (torch.zeros(num_tokens, num_tokens)).type(torch.bool)
    tgt_tokens = greedy_decode(
        model,  src, src_mask, max_len=num_tokens + 5, start_symbol=BOS_IDX).
 ↪flatten()
    return " ".join(vocab_transform[TGT_LANGUAGE].lookup_tokens(list(tgt_tokens.
 ↪cpu().numpy()))).replace("<bos>", "").replace("<eos>", "")
```

你可以使用不同的德语句子，测试模型的翻译效果。

[16]:
```
print(translate(transformer, "Eine Gruppe von Menschen steht vor einem Iglu ."))
```

```
A group of people stand in front of an empty instrument .
```

思考：如何设计指标以合理评价翻译质量？