如何构建一个基本的Transformer模型(英中翻译）

发表评论 / ChatGPT, GPT, OpenAI

一个简化版的Transformer模型训练程序示例，使用PyTorch框架。这个示例展示了如何构建一个基本的Transformer模型，用于一个简单的序列到序列的任务（例如，机器翻译或文本生成）。注意，这个例子是为了演示目的而简化的，实际应用中可能需要更复杂的数据处理、模型架构调整和训练策略。

0. 准备环境

确保安装了PyTorch。可以通过pip install torch安装。

```
<code>pip install torch</code>
```

1.

1. 准备训练样本

我们将定义一小批简单的英文句子及其对应的中文翻译。这里使用的是极其简化的数据集，仅用于演示目的。

```
# 英文到中文的简单句子对
english_sentences = &#91;
    "Hello, how are you?",  # 你好，你怎么样？
    "I am learning translation.",  # 我在学习翻译。
    "This is a pen.",  # 这是一支笔。
    "What is your name?",  # 你叫什么名字？
    "I love programming."  # 我爱编程。
]

chinese_sentences = &#91;
    "你好，你怎么样？",
    "我在学习翻译。",
    "这是一支笔。",
    "你叫什么名字？",
    "我爱编程。"
]

# 假设我们已经有了英文和中文的词汇表和编码函数（在实际应用中，需要根据实际数据构建）
# 为了简化，我们这里不实现这一部分，而是直接使用英文和中文句子的索引表示


```

2. 训练代码

```
import torch
import torch.nn as nn
import torch.optim as optim
import math
import jieba
import pickle

class TransformerModel(nn.Module):
    def __init__(self, ntoken, ninp, nhead, nhid, nlayers, dropout=0.5):
```

```
        super(TransformerModel, self).__init__()
        self.model_type = 'Transformer'
        self.pos_encoder = PositionalEncoding(ninp, dropout)
        encoder_layers = nn.TransformerEncoderLayer(d_model=ninp, nhead=nhead, dim_feedforwa
        self.transformer_encoder = nn.TransformerEncoder(encoder_layers, num_layers=nlayers)
        self.encoder = nn.Embedding(ntoken, ninp)
        self.ninp = ninp
        self.decoder = nn.Linear(ninp, ntoken)
        self.init_weights()

    def init_weights(self):
        initrange = 0.1
        self.encoder.weight.data.uniform_(-initrange, initrange)
        self.decoder.bias.data.zero_()
        self.decoder.weight.data.uniform_(-initrange, initrange)

    def forward(self, src):
        src = self.encoder(src) * math.sqrt(self.ninp)
        src = self.pos_encoder(src)
        output = self.transformer_encoder(src)
        output = self.decoder(output)
        return output

class PositionalEncoding(nn.Module):
    def __init__(self, d_model, dropout=0.1, max_len=5000):
        super(PositionalEncoding, self).__init__()
        self.dropout = nn.Dropout(p=dropout)

        position = torch.arange(max_len).unsqueeze(1)
        div_term = torch.exp(torch.arange(0, d_model, 2) * -(math.log(10000.0) / d_model))
        pe = torch.zeros(max_len, d_model)
        pe&#91;:, 0::2] = torch.sin(position * div_term)
        pe&#91;:, 1::2] = torch.cos(position * div_term)
        self.register_buffer('pe', pe.unsqueeze(0))

    def forward(self, x):
        x = x + self.pe&#91;:, :x.size(1)]
        return self.dropout(x)

# 英文到中文的简单句子对
english_sentences = &#91;
    "Hello, how are you?",  # 你好，你怎么样？
    "I am learning translation.",  # 我在学习翻译。
    "This is a pen.",  # 这是一支笔。
    "What is your name?",  # 你叫什么名字？
    "I love programming."  # 我爱编程。
]

chinese_sentences = &#91;
    "你好，你怎么样？",
    "我在学习翻译。",
    "这是一支笔。",
    "你叫什么名字？",
    "我爱编程。"
]

# 定义结束标记
end_token = '&lt;eos>'

# 构建词汇表的函数
def build_english_vocab(sentences):
    vocab = set(word for sentence in sentences for word in sentence.split())
    # 添加结束标记到词汇表中
    vocab.add(end_token)
    return {word: i for i, word in enumerate(vocab)}
```

```
75
76  # 使用jieba进行分词
77  def build_chinese_vocab(sentences):
78      vocab = set(word for sentence in sentences for word in jieba.cut(sentence))
79      # 添加结束标记到词汇表中
80      vocab.add(end_token)
81      return {word: i for i, word in enumerate(vocab)}
82
83  def encode_english(sentence, vocab, max_len):
84      words = sentence.split()&#91;:max_len - 1]  # 保留一个位置给结束标记
85      words.append(end_token)  # 添加结束标记
86      return &#91;vocab.get(word, vocab&#91;'&lt;unk>']) for word in words]
87
88  def encode_chinese(sentence, vocab, max_len):
89      words = list(jieba.cut(sentence))&#91;:max_len - 1]  # 保留一个位置给结束标记
90      words.append(end_token)  # 添加结束标记
91      return &#91;vocab.get(word, vocab&#91;'&lt;unk>']) for word in words]
92
93  # 参数设置
94  ntokens = 1000  # 词汇表大小
95  emsize = 200  # 嵌入维度
96  nhid = 200  # 前馈网络的维度
97  nlayers = 2  # Transformer层的数量
98  nhead = 2  # 多头注意力的头数
99  dropout = 0.2  # dropout的比例
100
101 english_vocab = build_english_vocab(english_sentences)
102 chinese_vocab = build_chinese_vocab(chinese_sentences)
103
104 # 编码函数
105 # 假设词汇表中包含&lt;unk>
106 english_vocab&#91;'&lt;unk>'] = len(english_vocab)
107 chinese_vocab&#91;'&lt;unk>'] = len(chinese_vocab)
108
109 # 定义最大序列长度
110 max_seq_length = max(len(sentence.split()) for sentence in english_sentences)
111
112 # 编码英文和法文句子
113 encoded_english_sentences = &#91;encode_english(sentence, english_vocab, max_seq_length) for
114 encoded_chinese_sentences = &#91;encode_chinese(sentence, chinese_vocab, max_seq_length) for
115
116 # 词汇表大小
117 english_vocab_size = len(english_vocab)
118 chinese_vocab_size = len(chinese_vocab)
119
120 print(f"english_vocab_size {english_vocab_size}")
121 print(f"chinese_vocab_size {chinese_vocab_size}")
122
123 ntokens = chinese_vocab_size  # 将中文词汇表的大小用作 ntokens
124
125 model = TransformerModel(ntokens, emsize, nhead, nhid, nlayers, dropout)
126
127 # 训练代码 ( 伪代码 )
128 criterion = nn.CrossEntropyLoss()
129 optimizer = optim.SGD(model.parameters(), lr=0.01)
130
131
132 # 假设已经定义了模型、损失函数和优化器
133 # 注意：在实际应用中，你需要根据任务调整模型的输入输出维度以及其他参数
134
135 epochs = 2000
136
137 for epoch in range(epochs):
138     total_loss = 0
139     for eng, chi in zip(encoded_english_sentences, encoded_chinese_sentences):
```

```
        model.train()
        optimizer.zero_grad()

        # 调整输入序列的形状以匹配 batch_first=True
        # &#91;批次大小, 序列长度]
        src = torch.tensor(&#91;eng], dtype=torch.long)  # 添加额外的维度来表示批次大小
        tgt = torch.tensor(&#91;chi], dtype=torch.long)

        output = model(src)

        # 重塑输出和目标以适应交叉熵损失
        output_reshaped = output.view(-1, ntokens)
        tgt_reshaped = tgt.view(-1)

        loss = criterion(output_reshaped, tgt_reshaped)
        loss.backward()
        optimizer.step()

        total_loss += loss.item()

    if epoch % 100 == 0:
        print(f"Epoch {epoch+1}, Loss: {total_loss / len(encoded_english_sentences)}")

# 保存模型的状态字典
model_path = f"models/gpt_model_1_{epochs}.pth"
torch.save(model.state_dict(), model_path)
print(f"Model saved to {model_path}")

# 保存 ntokens
print(f"ntokens {ntokens}")
with open(f"models/ntokens_1_{epochs}.pkl", "wb") as f:
    pickle.dump(ntokens, f)

# 保存 max_seq_length
print(f"max_seq_length {max_seq_length}")
with open(f"models/max_seq_length_1_{epochs}.pkl", "wb") as f:
    pickle.dump(max_seq_length, f)

# 保存英文词汇表
print(f"english_vocab {english_vocab}")
with open(f"models/english_vocab_1_{epochs}.pkl", "wb") as f:
    pickle.dump(english_vocab, f)

# 保存中文词汇表
print(f"chinese_vocab {chinese_vocab}")
with open(f"models/chinese_vocab_1_{epochs}.pkl", "wb") as f:
    pickle.dump(chinese_vocab, f)


# 输出模型参数数量
print(sum(p.numel() for p in model.parameters())/1e6, 'M parameters')

```

3. 加载模型产生文本

```
import torch
import torch.nn.functional as F
import torch.nn as nn
import math
import pickle

class TransformerModel(nn.Module):
    def __init__(self, ntoken, ninp, nhead, nhid, nlayers, dropout=0.5):

```

```
        super(TransformerModel, self).__init__()
        self.model_type = 'Transformer'
        self.pos_encoder = PositionalEncoding(ninp, dropout)
        encoder_layers = nn.TransformerEncoderLayer(d_model=ninp, nhead=nhead, dim_feedforwa
        self.transformer_encoder = nn.TransformerEncoder(encoder_layers, num_layers=nlayers)
        self.encoder = nn.Embedding(ntoken, ninp)
        self.ninp = ninp
        self.decoder = nn.Linear(ninp, ntoken)
        self.init_weights()

    def init_weights(self):
        initrange = 0.1
        self.encoder.weight.data.uniform_(-initrange, initrange)
        self.decoder.bias.data.zero_()
        self.decoder.weight.data.uniform_(-initrange, initrange)

    def forward(self, src):
        src = self.encoder(src) * math.sqrt(self.ninp)
        src = self.pos_encoder(src)
        output = self.transformer_encoder(src)
        output = self.decoder(output)
        return output

class PositionalEncoding(nn.Module):
    def __init__(self, d_model, dropout=0.1, max_len=5000):
        super(PositionalEncoding, self).__init__()
        self.dropout = nn.Dropout(p=dropout)

        position = torch.arange(max_len).unsqueeze(1)
        div_term = torch.exp(torch.arange(0, d_model, 2) * -(math.log(10000.0) / d_model))
        pe = torch.zeros(max_len, d_model)
        pe&#91;:, 0::2] = torch.sin(position * div_term)
        pe&#91;:, 1::2] = torch.cos(position * div_term)
        self.register_buffer('pe', pe.unsqueeze(0))

    def forward(self, x):
        x = x + self.pe&#91;:, :x.size(1)]
        return self.dropout(x)

end_token = '&lt;eos>'

def encode_english(sentence, vocab, max_len):
    words = sentence.split()&#91;:max_len - 1]  # 保留一个位置给结束标记
    words.append(end_token)  # 添加结束标记
    return &#91;vocab.get(word, vocab&#91;'&lt;unk>']) for word in words]

def decode_chinese(indices, vocab):
    words = &#91;list(vocab.keys())&#91;list(vocab.values()).index(idx)] for idx in indices]
    # 去除结束标记之后的部分
    if end_token in words:
        words = words&#91;:words.index(end_token)]
    return ''.join(words)

# 加载模型
ntokens = 1000  # 假设词汇表大小为1000
emsize = 200  # 嵌入维度
nhid = 200  # 前馈网络的维度
nlayers = 2  # Transformer层的数量
nhead = 2  # 多头注意力的头数
dropout = 0.2  # dropout的比例

epochs = 2000

# 加载 ntokens
with open(f"models/ntokens_1_{epochs}.pkl", "rb") as f:
```

```
    ntokens = pickle.load(f)
print(f"ntokens {ntokens}")

# 加载 max_seq_length
with open(f"models/max_seq_length_1_{epochs}.pkl", "rb") as f:
    max_seq_length = pickle.load(f)
print(f"max_seq_length {max_seq_length}")

# 加载英文词汇表
with open(f"models/english_vocab_1_{epochs}.pkl", "rb") as f:
    english_vocab = pickle.load(f)
print(f"english_vocab {english_vocab}")

# 加载中文词汇表
with open(f"models/chinese_vocab_1_{epochs}.pkl", "rb") as f:
    chinese_vocab = pickle.load(f)
print(f"chinese_vocab {chinese_vocab}")

model = TransformerModel(ntokens, emsize, nhead, nhid, nlayers, dropout)

model_path = f"models/gpt_model_1_{epochs}.pth"
model.load_state_dict(torch.load(model_path))
model.eval()


# 定义生成文本的函数
def generate_text(model, start_text, max_len=50):
    with torch.no_grad():
        tokenized_start_text = encode_english(start_text, english_vocab, max_seq_length)
        input_seq = torch.tensor(&#91;tokenized_start_text], dtype=torch.long)
        for _ in range(max_len):
            output = model(input_seq)
            predicted_token = torch.argmax(output&#91;:, -1, :], dim=-1)  # 获取最后一个时间步
            input_seq = torch.cat(&#91;input_seq, predicted_token.unsqueeze(0)], dim=1)  #
            if predicted_token.item() == end_token:  # 如果生成了终止符号，停止生成
                break
        generated_text = decode_chinese(input_seq.squeeze().tolist(), chinese_vocab)
        return generated_text

# 使用生成文本的函数
start_text = "Hello, how are you?"  # 初始文本
generated_text = generate_text(model, start_text)
print("Generated text:", generated_text)
```

运行的结果：

```
max_seq_length 4
english_vocab {'how': 0, 'learning': 1, 'are': 2, 'am': 3, 'Hello,': 4, 'pen.': 5, 'a': 6, 'T
chinese_vocab {'，': 0, '&lt;eos>': 1, '在': 2, '名字': 3, '这是': 4, '我': 5, '笔': 6, '翻译': 
Generated text: 这是，在。
```