1 Sequence to Sequence Learning with Neural Networks

1. 论文地址: Sequence to Sequence Learning with Neural Networks
2. 作者: Ilya Sutskever, Oriol Vinyals, Quoc V. Le
3. 代码参考 1: bentrevett / pytorch-seq2seq 代码参考 2: farizrahman4u / seq2seq

1.1 Notes

Abstract

本文提出了 LSTM-based **Sequence-to-Sequence** 模型，解决了传统神经网络在处理变长输入-输出序列时的困难. 尽管使用了有限的词汇表，但通过反转输入句子顺序这一简单的数据预处理技巧，在机器翻译任务中使模型性能超越了传统的机器翻译系统，并展示了 LSTM 在处理复杂序列到序列任务中的巨大潜力.

Situation

Challenge: 传统的深度学习方法解决序列化问题要求输入和输出的维度固定，但很多任务(如机器翻译)涉及到可变长输入和输出序列，这导致传统神经网络无法直接进行建模.

Needs: 需要新方法处理不定长输入和输出的序列，且能够捕捉长期的时间依赖关系（因为翻译任务对上下文很敏感）。

Task – Translation

Task: 文章需要解决可变长的输入-输出序列问题，尤其是机器翻译任务，需要将可变长的输入语言字段翻译为可变长的目标语言字段.

Action – Seq2Seq Model

Idea: 使用两独立的 LSTM，分别作为编码器(encoder)和解码器(decoder)(如图1所示).

1. 第一个 LSTM 作为编码器(encoder): 每时间步读取一个 input，从而将全部 input 组成的 sequence 编码为固定维度的 vector.
2. 第二个 LSTM 作为解码器(decoder)，将 decoder 编码的 vector 解码成目标序列.

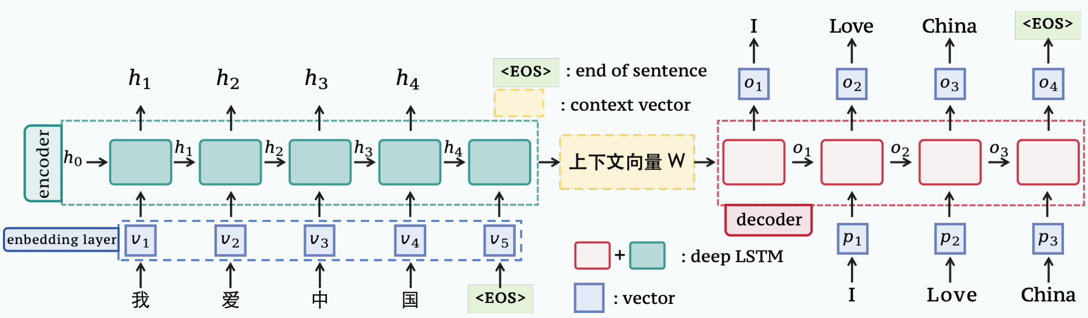

Figure 1: Sequence to sequence model example

Details and Tricks:

1. **Deep LSTM**: 文章发现使用深层 LSTM（四层LSTM），而非浅层 LSTM，可以提高模型的性能，特别是捕捉句子中长程依赖关系

2. **Reversing**: 一个重要的技巧是反转输入句子的顺序。具体来说，训练时将源语言句子反转顺序输入 LSTM，而目标语言句子保持不变. 例如正确对应关系是 $a \to \alpha, b \to \beta, c \to \gamma$，未反转的训练是使 $a, b, c \to \alpha, \beta, \gamma$，反转的训练是使 $c, b, a \to \alpha, \beta, \gamma$. [a]

3. **EOS**: end-of-sentence token，指自然语言句子结束的标志，以告知模型句子已结束.

4. **Training:** 采用最大化对数概率的方式进行训练，记源语言句子为 S，翻译结果为 T，训练集为 $\mathcal{S}$，那么

$$\text{Train:} \quad \max \frac{1}{|\mathcal{S}|} \sum_{(T,S)\in\mathcal{S}} \log p(T \mid S) \tag{1a}$$

$$\text{Test:} \quad \hat{T} = \arg\max_{T} p(T \mid S) \tag{1b}$$

5. **Decoding:** 使用 beam search 解码器生成目标序列[b].

[a] 文章猜测这样可以让输入句子的后面的词语与输出句子的前面的词语联系更紧密，使 LSTM 更容易捕捉输入和输出之间的映射关系.

[b] Beam Search 是一种启发式搜索算法，其核心思想是在解码过程中保持一个固定大小(即 beam width)的候选序列，并在每个时间步选择最有可能的序列扩展. 详见如何通俗理解beam search？seq2seq中的beam search算法

Result

该方法在实验中取得了显著的成果：

1. 机器翻译性能：在 WMT'14 英法翻译任务中，LSTM模型达到了 34.8 的 BLEU 分数，超越了传统的基于短语的统计机器翻译(SMT)系统（其BLEU分数为 33.3）.

2. LSTM 重评分：使用 LSTM 对 SMT 系统的 1000 个候选翻译进行重评分后(使用 LSTM 判断 SMT 生成的最好的翻译结果作为最终输出)，BLEU 分数提升至 36.5，接近该任务上的最佳结果.

3. Reversing 的影响：通过反转输入句子的顺序，LSTM 在长句子上的表现显著提升，测试困惑度(Perplexity)从 5.8 降至 4.7，BLEU 分数从 25.9 提高到 30.6.

4. 性能表现：LSTM 不仅在常规的短句翻译中表现良好，在长句子上也没有受到性能下降的影响，表现出了良好的鲁棒性

5. 与其他方法的比较：尽管 LSTM 模型在词汇表的大小上有限，但它依然能够在机器翻译任务中超越传统的统计机器翻译(SMT)方法，并且接近当时最佳的结果

Method	**test BLEU score (ntst14)**
Baseline System	33.30
Single forward LSTM, beam size 12	26.17
Single reversed LSTM, beam size 12	30.59
Ensemble of 5 reversed LSTMs, beam size 12	**34.81**

Table 1: The performance of the LSTM on WMT'14 English to French test set (ntst14).

Experiment details

LSTM architecture: 训练使用了 4 层深度 LSTM，每层有 1000 个单元，词嵌入维度为 1000. 总的来说，LSTM 模型有 384M 个参数，其中 64M 个是纯递归连接(32M 用于 decoder LSTM，32M 用于 encoder LSTM).

Vocabulary: 输入词汇表(英文)大小为 160,000，输出词汇表(法语)大小为 80,000，词汇表外的词语统一用 ”UNK” 表示.

Training:

1. initialization: 初始化 LSTM 的所有参数为 [-0.08, 0.08] 的均匀分布.
2. 梯度下降：使用随机梯度下降(SGD)来训练模型，固定学习率为 0.7，每训练 5 个 epoch 后将学习率减半
3. batch size：使用 128 个序列的批量进行训练，且每个批次的序列长度被调整为接近相同，以加快训练速度
4. 梯度裁剪：为了防止梯度爆炸(LSTM 一般不会梯度消失)，训练中对梯度进行裁剪，当梯度的 2-范数超过 5 时，将其缩放为 $\frac{5g}{\|g\|_2}$(g 为梯度).
5. parallelization: 使用了8个GPU并行化来加速训练

1.2 Codes

1.2.1 Introduction

Introduction

完整实现 Seq2Seq model 应用在机器翻译任务中的流程如下：

1. 准备工作：获取数据集→ 下载分词模型+ 分词→ 构建词汇表+ 用索引向量化
2. 数据处理：批处理(Batching) + 填充(Padding)
3. 构建模型：编码器encoder + 解码器decoder → seq2seq model
4. 训练+测试：定义模型→ 循环训练→ 评估模型+ 测试

1.2.2 Preparation

Preparation – module

需要用到的库及其作用如下：

1. torchtext: PyTorch 官方用于自然语言处理(NLP)的工具包, 本教程使用的版本为0.17.0. 不同的torch 与torchtext 版本适配可详见 pytorch text
2. random & numpy ｜｜spacy: 开源的自然语言处理(NLP)库
3. datasets: 由 Hugging Face 开发，提供了访问和处理大量 NLP 数据集的工具
4. tqdm: 用于显示进度条
5. evaluate: 简化模型评估过程，计算各种指标

```python
import torch
import torch.nn as nn
import torch.optim as optim
import random
import numpy as np
import spacy
import datasets
import tqdm
import evaluate
import torchtext
```

Preparation – random seed

为获得可重复性结果，需要固定随机种子.

```python
seed = 1234
random.seed(seed)
np.random.seed(seed)
torch.manual_seed(seed)
torch.cuda.manual_seed(seed)
torch.backends.cudnn.deterministic = True
```

Preparation – dataset

我们使用 **bentrevett/multi30k** 数据集，共 30k 条英德互译句，其中 29k 为train data，1k 为 test data. 因此还需要将 dataset 拆分成 train, validation, test 三部分.

```python
dataset = datasets.load_dataset("bentrevett/multi30k")
train_data, valid_data, test_data = (
    dataset["train"],
    dataset["validation"],
    dataset["test"],
)
================================== test ==================================
print(dataset)        # to check the dataset
print(train_data[1])  # to check the second data in the train dataset
================================== test ==================================
```

Preparation – tokenizer(sentence → tokens)

一句话(sentence)包含很多词(token)，将其分开需要构造分词器(tokenizer). 我们使用 spaCy model，针对不同语言需要下载不同模型：

1. 德语：”de_core_news_sm”. 终端中输入：python -m spacy download en_core_web_sm
2. 英语：”en_core_web_sm”. 终端中输入：python -m spacy download en_core_web_sm
3. 中文：”zh_core_web_sm”. 终端中输入：python -m spacy download zh_core_web_sm

```python
en_nlp = spacy.load("en_core_web_sm")   # English tokenizer
de_nlp = spacy.load("de_core_news_sm")  # German tokenizer
================================== test ==================================
string = "Hi, I am Ray!"
string_list = [token.text for token in en_nlp.tokenizer(string)]
print("string list:", string_list, "\n", "type of each element:", type(string_list[0]))
================================== test ==================================
def tokenize_example(example, en_nlp, de_nlp, max_length, lower, sos_token, eos_token):
    """
    Use the English and German NLP models to tokenize and obtain the English and German token lists respectively
    Return: new creature en_tokens, de_tokens features
    example: dataset
    en_nlp, de_nlp: English and German NLP models
    max_length: maximum number of tokens, usually 1000
    lower: whether to convert to lowercase
    sos_token, eos_token: start and end characters
    """
    en_tokens = [token.text for token in en_nlp.tokenizer(example["en"])][:max_length]
    de_tokens = [token.text for token in de_nlp.tokenizer(example["de"])][:max_length]
    if lower:
        en_tokens = [token.lower() for token in en_tokens]
        de_tokens = [token.lower() for token in de_tokens]
    # add <sos> and <eos>
    en_tokens = [sos_token] + en_tokens + [eos_token]
    de_tokens = [sos_token] + de_tokens + [eos_token]
    return {"en_tokens": en_tokens, "de_tokens": de_tokens}
```

下面对 dataset 中的所有的句子都分词，让 dataset 中出现两个新的特征：en_tokens, de_tokens

```
max_length = 1000
lower = True
sos_token = "<sos>"
eos_token = "<eos>"

fn_kwargs = {
    "en_nlp": en_nlp,
    "de_nlp": de_nlp,
    "max_length": max_length,
    "lower": lower,
    "sos_token": sos_token,
    "eos_token": eos_token,
}

train_data = train_data.map(tokenize_example, fn_kwargs=fn_kwargs)
valid_data = valid_data.map(tokenize_example, fn_kwargs=fn_kwargs)
test_data = test_data.map(tokenize_example, fn_kwargs=fn_kwargs)
==================================== test ====================================
print(valid_data[0])  # to check the new validation dataset
==================================== test ====================================
```

Preparation – vocabulary

为将 token 数值化，一个简单的想法是将全部出现过的 token 汇成词汇表(vocabulaty)，使用相应的序号(index)作为该 token 的数值编码，因此需要先创建词汇表. 注意

1. 要为两种语言各自创建 vocabulary
2. 为防数据泄露，即仅使用 train set 中的数据训练，构造 vocabulary 时也要仅使用 train set.
3. 需要添加特殊的 token：
   (a) $<$ unk $>$: unknown token, 在验证集和测试集中出现，但训练集中未出现.
   (b) $<$ sos $>$: start of sentence，句子开始
   (c) $<$ eos $>$: end of sentence，句子结束
   (d) $<$ pad $>$: padding，对需要填充位置补充的 token

```
min_freq = 2  # the least frequency, whose < min_freq will be taken by
    unk_token
unk_token = "<unk>"
pad_token = "<pad>"

special_tokens = [
    unk_token,
    pad_token,
    sos_token,
    eos_token,
]

en_vocab = torchtext.vocab.build_vocab_from_iterator(
    train_data["en_tokens"],  # remember use the train set
    min_freq=min_freq,
    specials=special_tokens,
```

```
)

de_vocab = torchtext.vocab.build_vocab_from_iterator(
    train_data["de_tokens"],
    min_freq=min_freq,
    specials=special_tokens,
)
================================= test 1  ==================================
print("top 15 tokens in English:", en_vocab.get_itos()[:15])
print("top 15 tokens in German:", de_vocab.get_itos()[:15])
print("the 'boy' in English tokens list is located at:", en_vocab.get_stoi
    ()["boy"], "the index of it is ", en_vocab["boy"])
print("the 'boy(junge in German)' in German tokens list is located at:",
    de_vocab.get_stoi()["junge"], "the index of it is ", de_vocab["junge"])
print("the length of Engilsh tokens list is", len(en_vocab))
print("the length of German tokens list is", len(de_vocab))
================================= test 1 =================================
assert en_vocab[unk_token] == de_vocab[unk_token]
assert en_vocab[pad_token] == de_vocab[pad_token]
# above is a double check

unk_index = en_vocab[unk_token]
pad_index = en_vocab[pad_token]
# set tokens that are not in the vocabulary as unk_index
en_vocab.set_default_index(unk_index)
de_vocab.set_default_index(unk_index)
================================= test 2 =================================
string = "Hi, I am Ray!"
string_list = [token.text for token in en_nlp.tokenizer(string)]
string_indexs = en_vocab.lookup_indices(string_list)
string_tokens = en_vocab.lookup_tokens(string_indexs)
print(" The string is: ", string, "\n", "The vector of this string is",
    string_indexs,  "\n", "The tokens of this string is", string_tokens)
================================= test 2 =================================
```

Preparation – numericalization(toekns → indexs)

下面将 tokens 数值化，仿照 tokenize_example 定义函数.

```
def numericalize_example(example, en_vocab, de_vocab):
    """
    Convert token list to numbers, toekns to indexes
    Return newly created en_ids, de_ids features
    example: dataset
    en_vocab: English vocabulary
    de_vocab: German vocabulary
    """
    en_ids = en_vocab.lookup_indices(example["en_tokens"])
    de_ids = de_vocab.lookup_indices(example["de_tokens"])
    return {"en_ids": en_ids, "de_ids": de_ids}

fn_kwargs = {"en_vocab": en_vocab, "de_vocab": de_vocab}

train_data = train_data.map(numericalize_example, fn_kwargs=fn_kwargs)
valid_data = valid_data.map(numericalize_example, fn_kwargs=fn_kwargs)
test_data = test_data.map(numericalize_example, fn_kwargs=fn_kwargs)
================================= test =================================
print(train_data[1])
print("The index list of the 2ed train_data is:", train_data[1]["en_ids"],
    "\n", "The token list of the 2ed train_data is:", en_vocab.lookup_tokens
```

```
    (train_data[1]["en_ids"]))
================================== test ==================================
```

Tensor

为方便后续输入 torch, 需要将 dataset 中的数值数据转化为张量/ tensor 形式.

```
data_type = "torch"  # tensor type
format_columns = ["en_ids", "de_ids"]

train_data = train_data.with_format(
    type=data_type,
    columns=format_columns,
    output_all_columns=True
)

valid_data = valid_data.with_format(
    type=data_type,
    columns=format_columns,
    output_all_columns=True,
)

test_data = test_data.with_format(
    type=data_type,
    columns=format_columns,
    output_all_columns=True,
)
================================== test ==================================
if type(train_data[1]["en_ids"]) == torch.Tensor:
    print("Already become tensor")
================================== test ==================================
```

1.2.3 Dataloader

Dataloader

在 Dataloader 中，我们将完成批处理(Batching)和数据填充(Padding). 在一个批次中，由于句子长度不一样因此需要将所有句子的数值化向量填充成相同长度.

```
def get_collate_fn(pad_index):  # Fill with pad_index (here is 1)
    def collate_fn(batch):
        batch_en_ids = [example["en_ids"] for example in batch]
        batch_de_ids = [example["de_ids"] for example in batch]
        batch_en_ids = nn.utils.rnn.pad_sequence(batch_en_ids,
            padding_value=pad_index)  # putput dim: (max_len, batch_size,
            feature_size)
        batch_de_ids = nn.utils.rnn.pad_sequence(batch_de_ids,
            padding_value=pad_index)
        batch = {
            "en_ids": batch_en_ids,
            "de_ids": batch_de_ids,
        }
        return batch

    return collate_fn

def get_data_loader(dataset, batch_size, pad_index, shuffle=False):
```

```
    collate_fn = get_collate_fn(pad_index)
    data_loader = torch.utils.data.DataLoader(
        dataset=dataset,
        batch_size=batch_size,
        collate_fn=collate_fn,
        shuffle=shuffle,
    )
    return data_loader

batch_size = 128
train_data_loader = get_data_loader(train_data, batch_size, pad_index,
    shuffle=True)
valid_data_loader = get_data_loader(valid_data, batch_size, pad_index)
test_data_loader = get_data_loader(test_data, batch_size, pad_index)
```

1.2.4 Model

Introduction

下面将通过三个部分构建模型:

编码器 encoder + 解码器 decoder → seq2seq model(将encoder 与decoder 组装)

在原文中，使用的 4 层的 LSTM，但是为了程序快速运行，此处只使用 2 层的 LSTM.

- 有关RNN 的介绍可以看视频【循环神经网络】5分钟搞懂RNN，3D动画深入浅出
- 有关LSTM 的介绍可以看博客Understanding LSTM Networks
- 关于各种PyTroch 的命令，详见官方文档torch.nn

Encoder

```
class Encoder(nn.Module):
    def __init__(self, input_dim, embedding_dim, hidden_dim, n_layers,
        dropout):
        super().__init__()
        self.hidden_dim = hidden_dim
        self.n_layers = n_layers
        self.embedding = nn.Embedding(input_dim, embedding_dim)   #
            input_dim to embedding_dim
        self.rnn = nn.LSTM(embedding_dim, hidden_dim, n_layers, dropout=
            dropout)
        self.dropout = nn.Dropout(dropout)

    def forward(self, src):
        embedded = self.dropout(self.embedding(src))  # embedded dim: [src
            length, batch size, embedding dim]
        outputs, (hidden, cell) = self.rnn(embedded)
        # outputs are always from the top hidden layer
        return hidden, cell
```

Decoder

```python
class Decoder(nn.Module):
    def __init__(self, output_dim, embedding_dim, hidden_dim, n_layers,
       dropout):
        super().__init__()
        self.output_dim = output_dim
        self.hidden_dim = hidden_dim
        self.n_layers = n_layers
        self.embedding = nn.Embedding(output_dim, embedding_dim)
        self.rnn = nn.LSTM(embedding_dim, hidden_dim, n_layers, dropout=
            dropout)
        self.fc_out = nn.Linear(hidden_dim, output_dim)
        self.dropout = nn.Dropout(dropout)

    def forward(self, input, hidden, cell):
        input = input.unsqueeze(0)
        embedded = self.dropout(self.embedding(input))
        output, (hidden, cell) = self.rnn(embedded, (hidden, cell))
        prediction = self.fc_out(output.squeeze(0))
        return prediction, hidden, cell
```

Seq2Seq model

在训练中, 需要使用到teacher forcing ratio(教师强制比例). teacher forcing 的核心思想是: 在训练过程中, 有 teacher_forcing_ratio 的概率将真实的目标(ground-truth)输出作为下一时间步的输入, 从而防止错误随着时间步的向前而不断积累.

```python
class Seq2Seq(nn.Module):
    def __init__(self, encoder, decoder, device):
        super().__init__()
        self.encoder = encoder
        self.decoder = decoder
        self.device = device
        assert (
            encoder.hidden_dim == decoder.hidden_dim
        ), "Hidden dimensions of encoder and decoder must be equal!"
        assert (
            encoder.n_layers == decoder.n_layers
        ), "Encoder and decoder must have equal number of layers!"

    def forward(self, src, trg, teacher_forcing_ratio):
        batch_size = trg.shape[1]
        trg_length = trg.shape[0]
        trg_vocab_size = self.decoder.output_dim
        outputs = torch.zeros(trg_length, batch_size, trg_vocab_size).to(
            self.device)
        hidden, cell = self.encoder(src)
        input = trg[0, :]
        for t in range(1, trg_length):
            output, hidden, cell = self.decoder(input, hidden, cell)
            # output = [batch size, output dim], hidden = [n layers, batch
                size, hidden dim], cell = [n layers, batch size, hidden dim]
            outputs[t] = output
            teacher_force = random.random() < teacher_forcing_ratio
            top1 = output.argmax(1)
            input = trg[t] if teacher_force else top1
        return outputs
```

1.2.5 Train

Train – model definition

```
input_dim = len(de_vocab)
output_dim = len(en_vocab)
encoder_embedding_dim = 256
decoder_embedding_dim = 256
hidden_dim = 512
n_layers = 2
encoder_dropout = 0.5
decoder_dropout = 0.5
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

encoder = Encoder(
    input_dim,
    encoder_embedding_dim,
    hidden_dim,
    n_layers,
    encoder_dropout,
)

decoder = Decoder(
    output_dim,
    decoder_embedding_dim,
    hidden_dim,
    n_layers,
    decoder_dropout,
)

model = Seq2Seq(encoder, decoder, device).to(device)
print("Our Sequence to sequence is:\n", model)

def init_weights(m):  # weight initialization
    for name, param in m.named_parameters():
        nn.init.uniform_(param.data, -0.08, 0.08)

model.apply(init_weights)

# calculate the parameters number
def count_parameters(model):
    return sum(p.numel() for p in model.parameters() if p.requires_grad)

print(f"There is {count_parameters(model)} trainable parameters in our
    model.")

optimizer = optim.Adam(model.parameters())             # optimizer
criterion = nn.CrossEntropyLoss(ignore_index=pad_index)  # loss function
```

Train – loop

```
# Train loop
def train_fn(model, data_loader, optimizer, criterion, clip,
    teacher_forcing_ratio, device):
    model.train()
    epoch_loss = 0
    for i, batch in enumerate(data_loader):
        src = batch["de_ids"].to(device)
```

```
        trg = batch["en_ids"].to(device)
        optimizer.zero_grad()
        output = model(src, trg, teacher_forcing_ratio)
        output_dim = output.shape[-1]
        output = output[1:].view(-1, output_dim)
        trg = trg[1:].view(-1)
        loss = criterion(output, trg)
        loss.backward()
        torch.nn.utils.clip_grad_norm_(model.parameters(), clip)
        optimizer.step()
        epoch_loss += loss.item()
    return epoch_loss / len(data_loader)
================================== test ==================================
# test .view(-1)
tensor = [[1, 2, 3],
          [4, 5, 6]]
tensor = torch.tensor(tensor)
flattened_tensor = tensor.view(-1, 6)
print(flattened_tensor)
================================== test ==================================

# Evaluation loop
def evaluate_fn(model, data_loader, criterion, device):
    model.eval()
    epoch_loss = 0
    with torch.no_grad():
        for i, batch in enumerate(data_loader):
            src = batch["de_ids"].to(device)  # src dim: [src length, batch
                 size]
            trg = batch["en_ids"].to(device)  # trg dim: [trg length, batch
                 size]
            output = model(src, trg, 0)       # no teacher forcing, output
                dim: [trg length, batch size, trg vocab size]
            output_dim = output.shape[-1]
            output = output[1:].view(-1, output_dim)
            trg = trg[1:].view(-1)
            loss = criterion(output, trg)
            epoch_loss += loss.item()
    return epoch_loss / len(data_loader)
```

Train – model train

```
n_epochs = 10
clip = 1.0
teacher_forcing_ratio = 0.5

best_valid_loss = float("inf")

print("Training on", device)

for epoch in tqdm.tqdm(range(n_epochs)):
    train_loss = train_fn(
        model,
        train_data_loader,
        optimizer,
        criterion,
        clip,
        teacher_forcing_ratio,
        device,
```

```
    )
    valid_loss = evaluate_fn(
        model,
        valid_data_loader,
        criterion,
        device,
    )
    if valid_loss < best_valid_loss:
        best_valid_loss = valid_loss
        torch.save(model.state_dict(), "tut1-model.pt")
    print(f"\tTrain Loss: {train_loss:7.3f} | Train PPL: {np.exp(train_loss):7.3f}")
    print(f"\tValid Loss: {valid_loss:7.3f} | Valid PPL: {np.exp(valid_loss):7.3f}")
```

1.2.6 Evaluation

Evaluation

```
model.load_state_dict(torch.load("tut1-model.pt"))
test_loss = evaluate_fn(model, test_data_loader, criterion, device)
print(f"| Test Loss: {test_loss:.3f} | Test PPL: {np.exp(test_loss):7.3f} |")

def translate_sentence(
    sentence,
    model,
    en_nlp,
    de_nlp,
    en_vocab,
    de_vocab,
    lower,
    sos_token,
    eos_token,
    device,
    max_output_length=25,
):
    model.eval()
    with torch.no_grad():
        if isinstance(sentence, str):
            tokens = [token.text for token in de_nlp.tokenizer(sentence)]
        else:
            tokens = [token for token in sentence]
        if lower:
            tokens = [token.lower() for token in tokens]
        tokens = [sos_token] + tokens + [eos_token]
        ids = de_vocab.lookup_indices(tokens)
        tensor = torch.LongTensor(ids).unsqueeze(-1).to(device)
        hidden, cell = model.encoder(tensor)
        inputs = en_vocab.lookup_indices([sos_token])
        for _ in range(max_output_length):
            inputs_tensor = torch.LongTensor([inputs[-1]]).to(device)
            output, hidden, cell = model.decoder(inputs_tensor, hidden, cell)
            predicted_token = output.argmax(-1).item()
            inputs.append(predicted_token)
            if predicted_token == en_vocab[eos_token]:
                break
        tokens = en_vocab.lookup_tokens(inputs)
```

```
    return tokens

================================= test 1 =================================
sentence = test_data[0]["de"]
expected_translation = test_data[0]["en"]

print("The German is: ", sentence)
print("The expected translation is: ", expected_translation)

translation = translate_sentence(
    sentence,
    model,
    en_nlp,
    de_nlp,
    en_vocab,
    de_vocab,
    lower,
    sos_token,
    eos_token,
    device,
)

print("The model output is: ", " ".join(translation[1:-1]))
================================= test 1 =================================

================================= test 2 =================================
sentence = "Ein Mann sitzt auf einer Bank."

translation = translate_sentence(
    sentence,
    model,
    en_nlp,
    de_nlp,
    en_vocab,
    de_vocab,
    lower,
    sos_token,
    eos_token,
    device,
)

print("The German is: ", sentence)
print("The model output is: ", " ".join(translation[1:-1]))
================================= test 2 =================================

================================= test 3 =================================
translations = [
    translate_sentence(
        example["de"],
        model,
        en_nlp,
        de_nlp,
        en_vocab,
        de_vocab,
        lower,
        sos_token,
        eos_token,
        device,
    )
    for example in tqdm.tqdm(test_data)
]
```

```
bleu = evaluate.load("bleu")

predictions = [" ".join(translation[1:-1]) for translation in translations]

references = [[example["en"]] for example in test_data]
predictions[0:2], references[0:2]
================================= test 3 =================================
```

Calculate the BLEU

```
def get_tokenizer_fn(nlp, lower):
    def tokenizer_fn(s):
        tokens = [token.text for token in nlp.tokenizer(s)]
        if lower:
            tokens = [token.lower() for token in tokens]
        return tokens

    return tokenizer_fn

tokenizer_fn = get_tokenizer_fn(en_nlp, lower)
================================== test ==================================
tokenizer_fn(predictions[0]), tokenizer_fn(references[0][0])
================================== test ==================================
results = bleu.compute(
    predictions=predictions, references=references, tokenizer=tokenizer_fn
)

print(results)
```

First updated: April 8, 2025