Minor in AI

Machine Translation

Hands-on Experience 01

April 02, 2025

1 Machine Translation

Machine translation (MT) involves automatically translating text from one language to another. Traditional methods relied on statistical approaches, but with advancements in deep learning, sequence-to-sequence (seq2seq) models have been widely adopted. However, seq2seq models suffer from limitations in handling long-range dependencies, which led to the development of the **attention mechanism** and later, **transformers**.

2 Code Implementation

2.1 Library Imports and Device Selection

The following libraries are imported to facilitate deep learning and natural language processing (NLP) tasks:

```
import torch
import torch.nn as nn
import torch.optim as optim
import numpy as np
import random
import matplotlib.pyplot as plt
import re
from collections import Counter
import pandas as pd

# Select device for computation
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(f"Using device: {device}")
```

```
Using device: cuda

A module that was compiled using NumPy 1.x cannot be run in
NumPy 2.0.2 as it may crash. To support both 1.x and 2.x
versions of NumPy, modules must be compiled with NumPy 2.0.
Some module may need to rebuild instead e.g. with 'pybind11>=2.12'.

If you are a user of the module, the easiest solution will be to
downgrade to 'numpy<2' or try to upgrade the affected module.
We expect that some modules will need time to support NumPy 2.
```

- `PyTorch` is used to build and train the Seq2Seq model.
- `SpaCy` is utilized for tokenization of both English and French text.
- `Pandas` helps with data handling from CSV files.
- `Matplotlib` (though not used yet) can be useful for visualization.

2.2 Downloading and Loading SpaCy Tokenizers

To process English and French text, SpaCy models are downloaded and loaded:

Loading spaCy Tokenizers

- **English Tokenizer**: `spacy.load("en_core_web_sm")` loads a pretrained English tokenizer.
- **French Tokenizer**: `spacy.load("fr_core_news_sm")` loads a pretrained French tokenizer.

```
!python -m spacy download fr_core_news_sm

import spacy

spacy_en = spacy.load("en_core_web_sm")
spacy_fr = spacy.load("fr_core_news_sm")
```

2.3 Tokenization Functions

Let's code some functions which will be responsible for tokenizing English and French text using the `spaCy` library and loading a dataset from a CSV file.

- `tokenize_en(text)`: Tokenizes an English sentence into a list of words.
- `tokenize_fr(text)`: Tokenizes a French sentence into a list of words.
- Both functions use `spaCy`'s tokenizer to handle text segmentation.

```
MAX_LEN = 100

def tokenize_en(text):
    return [tok.text for tok in spacy_en.tokenizer(text)]

def tokenize_fr(text):
    return [tok.text for tok in spacy_fr.tokenizer(text)]

csv_path = "eng-french.csv"
df = pd.read_csv(csv_path)
df.head()
```

	English words/sentences	French words/sentences
0	Hi.	Salut!
1	Run!	Cours !
2	Run!	Courez !
3	Who?	Qui ?
4	Wow!	Ça alors !

2.4 Reading and Preprocessing the Dataset

The dataset is read from a CSV file and column names are standardized:

```
df = df.rename(columns = {"English words/sentences":"english","French
    words/sentences":"french"})
df.head()
```

	english	french
0	Hi.	Salut!
1	Run!	Cours !
2	Run!	Courez !
3	Who?	Qui ?
4	Wow!	Ça alors !

2.5 Building Vocabulary

This vocabulary-building step is essential for converting words into numerical indices, which can then be fed into deep learning models. It ensures that only the most frequent words are considered, reducing unnecessary memory usage and improving model efficiency. The inclusion of special tokens helps with sequence modeling, making this an important preprocessing step in neural machine translation tasks.

2.5.1 Importing Required Modules

- **Counter** from `collections` is used to count the frequency of words.
- **Vocab** from `torchtext.vocab` is used to create the vocabulary.
- **OrderedDict** ensures that words are stored in order of frequency.

2.5.2 Defining Special Tokens

Four special tokens are defined:

- `<pad>`: Used for padding shorter sequences.
- `<sos>` (Start of Sentence): Indicates the beginning of a sentence.
- `<eos>` (End of Sentence): Marks the end of a sentence.
- `<unk>` (Unknown): Represents words not found in the vocabulary.

2.5.3 Building the Vocabulary (`build_vocab` function)

- A `Counter` object is initialized to count word occurrences.
- Each sentence in the dataset is tokenized using the given tokenizer.
- The `Counter` is updated with the tokenized words.
- Words are sorted by their frequency in descending order using `OrderedDict`.
- A vocabulary object is created with `torchtext.vocab.vocab`, and special tokens are included.
- The `<unk>` token is set as the default for any unknown words encountered.

2.5.4 Creating Source and Target Vocabularies

- **SRC_VOCAB** is built from the English sentences (`df["english"]`), using the `tokenize_en` function.
- **TRG_VOCAB** is built from the French sentences (`df["french"]`), using `tokenize_fr`.

```
from collections import Counter
from torchtext.vocab import Vocab
from collections import OrderedDict

special_tokens = ["<pad>","<sos>","<eos>","<unk>"]

def build_vocab(sentences, tokenizer):
    counter = Counter()
    for s in sentences:
        counter.update(tokenizer(s))
    sorted_by_freq = OrderedDict(sorted(counter.items(), key=lambda x: x[1], reverse=True))
    vocab_obj = vocab(sorted_by_freq, specials=special_tokens)
    vocab_obj.set_default_index(vocab_obj["<unk>"])
    return vocab_obj

SRC_VOCAB = build_vocab(df["english"], tokenize_en)
TRG_VOCAB = build_vocab(df["french"], tokenize_fr)
```

2.6 Dataset and DataLoader Preparation

This section explains how the dataset is prepared and structured for training a sequence-to-sequence (Seq2Seq) translation model. The process includes tokenizing text, converting words into numerical representations, and defining a custom PyTorch dataset.

2.6.1 Numericalization Function

The function `numericalize` converts a list of tokens (words) into numerical indices based on the given vocabulary.

- **Input**: A list of tokenized words and the corresponding vocabulary.
- **Processing**:
  - Adds a `<sos>` (Start of Sentence) token at the beginning.
  - Replaces each word with its corresponding index from the vocabulary.
  - Appends a `<eos>` (End of Sentence) token at the end.
- **Output**: A list of numerical indices representing the sentence.

2.6.2 Custom Dataset Class

The class `TranslationDataset` inherits from `torch.utils.data.Dataset` and defines how the dataset is structured for PyTorch.

2.6.3 Methods and Functionality

`__init__` Method

- Takes the dataset `df`, which contains English and French sentence pairs.
- Stores it in the class instance.

`__len__` Method

- Returns the total number of sentence pairs in the dataset.

`__getitem__` Method

- Retrieves a specific sentence pair based on an index.
- **Processing Steps**:
  - **Tokenization**: The English and French sentences are tokenized using `tokenize_en` and `tokenize_fr`, respectively.
  - **Truncation**: Sentences are truncated to `MAX_LEN` to prevent excessively long sequences.
  - **Numericalization**: The tokenized sentences are converted into numerical sequences using the `numericalize` function.
  - **Conversion to Tensor**: The numericalized sequences are converted into PyTorch tensors.
- **Returns**: A tuple containing:
  - The numericalized English sentence as a tensor.
  - The numericalized French sentence as a tensor.

Why is this step crucial?

1. **Prepares the dataset for training**: The `TranslationDataset` class ensures that each batch is formatted correctly for the model.
2. **Handles text-to-numerical conversion**: Since deep learning models work with numbers rather than raw text, this step is crucial.
3. **Facilitates efficient data loading**: The dataset can be easily passed to a `DataLoader` to enable mini-batch training, shuffling, and parallel data processing.

```
from torch.utils.data import Dataset, DataLoader

def numericalize(tokens, vocab):
    return [vocab['<sos>']] + [vocab[token] for token in tokens] + [
   vocab['<eos>']]

class TranslationDataset(Dataset):
    def __init__(self, df):
        self.df = df

```

```
    def __len__(self):
        return len(self.df)

    def __getitem__(self, idx):
        en = tokenize_en(self.df.iloc[idx]['english'])[:MAX_LEN]
        fr = tokenize_fr(self.df.iloc[idx]['french'])[:MAX_LEN]
        en_ids = numericalize(en, SRC_VOCAB)
        fr_ids = numericalize(fr, TRG_VOCAB)
        return torch.tensor(en_ids), torch.tensor(fr_ids)
```

Now let's see how the dataset is split into training and testing subsets and how it is loaded efficiently using PyTorch's `DataLoader`.

2.6.4 Dataset Splitting

- The dataset is first instantiated using the `TranslationDataset` class:

```
full_dataset = TranslationDataset(df)
```

- The total dataset size is determined, and 90% of it is allocated for training while 10% is reserved for testing:

```
train_size = int(0.9 * len(full_dataset))
```

- The dataset is then split into two parts:
  - **Training Set**: Contains 90% of the dataset.
  - **Testing Set**: Contains the remaining 10%.

```
train_dataset, test_dataset =
torch.utils.data.random_split(full_dataset, [train_size,
            len(full_dataset) - train_size])
```

2.6.5 DataLoader

PyTorch's `DataLoader` is used to efficiently load data in batches for training and testing.

Training DataLoader

- The training set is loaded in batches of 32.
- `shuffle=True` ensures that the training data is shuffled to improve generalization.
- The `collate_fn` function is used to pad sentences to the same length within a batch.

```
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True,
                          collate_fn=collate_fn)
```

Testing DataLoader

- The testing set is also loaded in batches of 32.
- `shuffle=False` ensures that the test data is not shuffled to maintain consistency during evaluation.
- The `collate_fn` function is again used for padding sequences.

```
test_loader = DataLoader(test_dataset, batch_size=32, shuffle=False,
                         collate_fn=collate_fn)
```

This dataset preparation step is crucial for training a Seq2Seq model efficiently.

```
from torch.nn.utils.rnn import pad_sequence

def collate_fn(batch):
    src_batch, tgt_batch = zip(*batch)
    src_batch = pad_sequence(src_batch, padding_value=SRC_VOCAB['<pad>'
    ], batch_first=True)
    tgt_batch = pad_sequence(tgt_batch, padding_value=TRG_VOCAB['<pad>'
    ], batch_first=True)
    return src_batch, tgt_batch
```

Finally, the dataset is split into training and testing sets, and DataLoaders are created:

```
full_dataset = TranslationDataset(df)
train_size = int(0.9 * len(full_dataset))
train_dataset, test_dataset = torch.utils.data.random_split(full_dataset
    , [train_size, len(full_dataset) - train_size])
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True,
    collate_fn=collate_fn)
test_loader = DataLoader(test_dataset, batch_size=32, shuffle=False,
    collate_fn=collate_fn)
```

3 Key Takeaways

- Machine translation has evolved from statistical models to deep learning-based Seq2Seq models with attention mechanisms.
- Attention mechanisms address the limitation of handling long-range dependencies in traditional Seq2Seq models.
- Tokenization and vocabulary building are crucial preprocessing steps for neural machine translation.
- PyTorch and SpaCy provide essential tools for implementing and training machine translation models efficiently.