Report

General Params

```
TRAIN_BATCH_SIZE = 256
VALID_BATCH_SIZE = 64
TEST_BATCH_SIZE = 32
```

Task 1: Bidirectional LSTM model

- **Model Architecture:**
  - BiLSTM model is instantiated with the following hyperparameters:
    - `vocab_size`: Size of the vocabulary (determined by the size of the word index ~8.2K).
    - `embedding_dim`: Dimension of the input features (set to 100).
    - `hidden_size`: Number of units in the hidden layers (set to 256).
    - `output_size`: Size of the output from the LSTM layer (set to 128).
    - `num_layers`: Number of recurrent layers (set to 1).
    - `dropout_val`: Dropout probability (set to 0.33).
    - `num_tags`: Number of output classes (NER tags is 9).
- **Loss Function and Optimizer:**
  - CrossEntropyLoss is used as the loss function, with special tokens ignored during computation.
  - AdamW optimizer is employed with a learning rate of 0.001.

1. What are the precision, recall, and F1 score on the validation data?

```
processed 51362 tokens with 5942 phrases; found: 5609 phrases; correct: 4468.
accuracy:  77.97%; (non-O)
accuracy:  95.55%; precision:  79.66%; recall:  75.19%; FB1:  77.36
           LOC: precision:  87.41%; recall:  83.94%; FB1:  85.64  1764
          MISC: precision:  74.81%; recall:  74.73%; FB1:  74.77  921
           ORG: precision:  73.18%; recall:  67.34%; FB1:  70.14  1234
           PER: precision:  78.93%; recall:  72.42%; FB1:  75.54  1690
Precision: 79.66, Recall: 75.19, F1 Score: 77.36
```

2. What are the precision, recall, and F1 score on the test data?

```
processed 46435 tokens with 5648 phrases; found: 5274 phrases; correct: 3732.
accuracy:  70.46%; (non-O)
accuracy:  93.39%; precision:  70.76%; recall:  66.08%; FB1:  68.34
           LOC: precision:  81.46%; recall:  76.68%; FB1:  79.00  1570
          MISC: precision:  61.89%; recall:  63.39%; FB1:  62.63  719
           ORG: precision:  65.31%; recall:  59.96%; FB1:  62.52  1525
           PER: precision:  69.32%; recall:  62.59%; FB1:  65.78  1460
Precision: 70.76, Recall: 66.08, F1 Score: 68.34
```

Task 2: Using GloVe word embeddings

- **Model Architecture:**
  - BiLSTM model is instantiated with the following hyperparameters:
    - `vocab_size`: Size of the vocabulary (determined by the size of the GloVe embeddings ~400K).
    - `embedding_dim`: Dimension of the input features (set to 100, matching the GloVe embeddings).
    - `hidden_size`: Number of units in the hidden layers (set to 256).
    - `output_size`: Size of the output from the LSTM layer (set to 128).
    - `num_layers`: Number of recurrent layers (set to 1).
    - `dropout_val`: Dropout probability (set to 0.33).
    - `num_tags`: Number of output classes (NER tags is 9).
- **GloVe Embeddings:**
  - The pre-trained GloVe embeddings (`glove_embedding_matrix`) are used for initializing the embedding layer of the model.
- **Loss Function and Optimizer:**
  - CrossEntropyLoss is used as the loss function, with special tokens ignored during computation.
  - AdamW optimizer is employed with a learning rate of 0.001.

1. What is the precision, recall, and F1 score on the validation data?

```
processed 51362 tokens with 5942 phrases; found: 5939 phrases; correct: 5201.
accuracy:  87.32%; (non-O)
accuracy:  97.41%; precision:  87.57%; recall:  87.53%; FB1:  87.55
           LOC: precision:  90.43%; recall:  93.14%; FB1:  91.77  1892
          MISC: precision:  81.57%; recall:  77.77%; FB1:  79.62  879
           ORG: precision:  79.33%; recall:  77.85%; FB1:  78.58  1316
           PER: precision:  93.36%; recall:  93.87%; FB1:  93.61  1852
Precision: 87.57, Recall: 87.53, F1 Score: 87.55
```

2. What are the precision, recall, and F1 score on the test data?

```
processed 46435 tokens with 5648 phrases; found: 5700 phrases; correct: 4727.
accuracy:  85.22%; (non-O)
accuracy:  96.53%; precision:  82.93%; recall:  83.69%; FB1:  83.31
           LOC: precision:  83.96%; recall:  89.75%; FB1:  86.76  1783
          MISC: precision:  70.70%; recall:  69.09%; FB1:  69.88  686
           ORG: precision:  78.51%; recall:  77.42%; FB1:  77.96  1638
           PER: precision:  91.59%; recall:  90.23%; FB1:  90.90  1593
Precision: 82.93, Recall: 83.69, F1 Score: 83.31
```

3. BiLSTM with Glove Embeddings outperforms the model without. Can you provide a rationale for this?
    - **Semantic Information:** GloVe embeddings capture semantic relationships, enhancing the model's understanding of word context and meaning.
    - **Generalization:** Pre-trained on diverse corpora, GloVe enables better generalization across various domains compared to models without embeddings.
    - **Reduced Dimensionality:** Lower-dimensional GloVe embeddings provide more expressive word representations, aiding generalization and relationship capture.
    - **Transfer Learning Effect:** GloVe acts as a form of transfer learning, leveraging pre-trained embeddings for downstream tasks.
    - **Improved Initialization:** GloVe embeddings serve as better initializations for model parameters, aiding convergence during training.
    - **Handling OOV Words:** GloVe effectively handles out-of-vocabulary words, contributing to improved model performance. It also has a lrage vocabulary.
    - **Knowledge Transfer:** GloVe embeddings transfer knowledge from a large corpus, benefiting tasks with limited labeled data.
    - **Overall Performance:** The combined effect results in superior precision, recall, and F1 scores in both validation and test datasets for the BiLSTM model with GloVe embeddings.

Bonus: The Transformer Encoder

- **Model Architecture:**
    - The model is an instance of the `TransformerNER` class, representing a Transformer encoder for sequence tagging tasks.
    - Hyperparameters:
        - `vocab_size`: Size of the vocabulary (determined by the size of the word index ~8.2K).
        - `embedding_dim`: Dimension of the input embeddings (set to 128).
        - `num_attention_heads`: Number of attention heads in the multi-head self-attention mechanism (set to 8).
        - `num_encoder_layers`: Number of transformer encoder layers (set to 6).
        - `num_classes`: Number of output classes (NER tags).
        - `max_len`: Maximum sequence length (set to 128).
- **Loss Function and Optimizer:**
    - CrossEntropyLoss is used as the loss function, with special tokens ignored during computation.
    - AdamW optimizer is employed with a learning rate of 0.0001, betas=(0.9, 0.98), and epsilon (eps) set to 1e-9.

1. What is the precision, recall, and F1 score on the validation data?

```
processed 51362 tokens with 5942 phrases; found: 4857 phrases; correct: 3132.
accuracy:  53.28%; (non-O)
accuracy:  92.05%; precision:  64.48%; recall:  52.71%; FB1:  58.01
           LOC: precision:  81.53%; recall:  73.27%; FB1:  77.18  1651
          MISC: precision:  71.82%; recall:  64.97%; FB1:  68.22  834
           ORG: precision:  55.45%; recall:  47.80%; FB1:  51.34  1156
           PER: precision:  44.90%; recall:  29.64%; FB1:  35.71  1216
Precision: 64.48, Recall: 52.71, F1 Score: 58.01
```

2. What are the precision, recall, and F1 score on the test data?

```
processed 46435 tokens with 5648 phrases; found: 4029 phrases; correct: 2242.
accuracy:  41.62%; (non-O)
accuracy:  89.46%; precision:  55.65%; recall:  39.70%; FB1:  46.34
           LOC: precision:  73.55%; recall:  65.53%; FB1:  69.31  1486
          MISC: precision:  61.25%; recall:  57.41%; FB1:  59.26  658
           ORG: precision:  49.25%; recall:  33.71%; FB1:  40.03  1137
           PER: precision:  24.87%; recall:  11.50%; FB1:  15.73  748
Precision: 55.65, Recall: 39.70, F1 Score: 46.34
```

3. What is the reason behind the poor performance of the transformer?
    - **Attention Mechanism Limitations:** The attention mechanism in transformers may struggle to capture long-range dependencies in sequential data, impacting its ability to understand context over extensive distances.
    - **Sequential Information Handling:** Transformers process sequences in parallel, potentially losing sequential information crucial for NER tasks.
    - **Fixed Context Window:** Despite attention mechanisms, transformers often rely on a fixed context window, limiting their ability to capture context beyond a certain range.
    - **Model Complexity:** The complexity of transformer models might lead to overfitting, especially when dealing with limited labeled data.
    - **Hyperparameter Tuning:** Transformers are sensitive to hyperparameter choices, and fine-tuning them for specific NER tasks is crucial.
    - **Lack of Pre-training:** Unlike GloVe embeddings, transformers may not have been pre-trained on a task-specific corpus, affecting their ability to capture domain-specific nuances.
    - **Token-Level Information:** Transformers treat tokens individually, potentially struggling with entity boundaries and relationships.
    - **Overall Complexity:** The transformer's intricate architecture and self-attention mechanism might not align optimally with the characteristics of Named Entity Recognition tasks.
    - **Need for Task-Specific Adaptation:** Transformers may require more task-specific adaptations or modifications to excel in Named Entity Recognition compared to simpler architectures like BiLSTM.

Dependencies

Installation

```
!pip install datasets accelerate
!pip install ipython-autotime
```

```
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 134.8/134.8 kB 12.0 MB/s eta 0:00:00
Requirement already satisfied: fsspec[http]<=2023.10.0,>=2023.1.0 in /usr/local/lib/python3.10/dist-packages (from datasets) (2023.6.0)
Requirement already satisfied: aiohttp in /usr/local/lib/python3.10/dist-packages (from datasets) (3.8.6)
Collecting huggingface-hub<1.0.0,>=0.14.0 (from datasets)
  Downloading huggingface_hub-0.19.0-py3-none-any.whl (311 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 311.2/311.2 kB 41.3 MB/s eta 0:00:00
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from datasets) (23.2)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.10/dist-packages (from datasets) (6.0.1)
Requirement already satisfied: psutil in /usr/local/lib/python3.10/dist-packages (from accelerate) (5.9.5)
Requirement already satisfied: torch>=1.10.0 in /usr/local/lib/python3.10/dist-packages (from accelerate) (2.1.0+cu118)
Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp->datasets) (23.1.0)
Requirement already satisfied: charset-normalizer<4.0,>=2.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp->datasets) (3.3.2)
Requirement already satisfied: multidict<7.0,>=4.5 in /usr/local/lib/python3.10/dist-packages (from aiohttp->datasets) (6.0.4)
Requirement already satisfied: async-timeout<5.0,>=4.0.0a3 in /usr/local/lib/python3.10/dist-packages (from aiohttp->datasets) (4.0.3)
Requirement already satisfied: yarl<2.0,>=1.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp->datasets) (1.9.2)
Requirement already satisfied: frozenlist>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from aiohttp->datasets) (1.4.0)
Requirement already satisfied: aiosignal>=1.1.2 in /usr/local/lib/python3.10/dist-packages (from aiohttp->datasets) (1.3.1)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from huggingface-hub<1.0.0,>=0.14.0->datasets) (3.13.1)
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.10/dist-packages (from huggingface-hub<1.0.0,>=0.14.0->datasets) (4.5.0)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests>=2.19.0->datasets) (3.4)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests>=2.19.0->datasets) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests>=2.19.0->datasets) (2023.7.22)
Requirement already satisfied: sympy in /usr/local/lib/python3.10/dist-packages (from torch>=1.10.0->accelerate) (1.12)
Requirement already satisfied: networkx in /usr/local/lib/python3.10/dist-packages (from torch>=1.10.0->accelerate) (3.2.1)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages (from torch>=1.10.0->accelerate) (3.1.2)
Requirement already satisfied: triton==2.1.0 in /usr/local/lib/python3.10/dist-packages (from torch>=1.10.0->accelerate) (2.1.0)
Requirement already satisfied: python-dateutil>=2.8.1 in /usr/local/lib/python3.10/dist-packages (from pandas->datasets) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas->datasets) (2023.3.post1)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.8.1->pandas->datasets) (1.16.0)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from jinja2->torch>=1.10.0->accelerate) (2.1.3)
Requirement already satisfied: mpmath>=0.19 in /usr/local/lib/python3.10/dist-packages (from sympy->torch>=1.10.0->accelerate) (1.3.0)
Installing collected packages: dill, multiprocess, huggingface-hub, accelerate, datasets
Successfully installed accelerate-0.24.1 datasets-2.14.6 dill-0.3.7 huggingface-hub-0.19.0 multiprocess-0.70.15
Collecting ipython-autotime
  Downloading ipython_autotime-0.3.2-py2.py3-none-any.whl (7.0 kB)
Requirement already satisfied: ipython in /usr/local/lib/python3.10/dist-packages (from ipython-autotime) (7.34.0)
Requirement already satisfied: setuptools>=18.5 in /usr/local/lib/python3.10/dist-packages (from ipython->ipython-autotime) (67.7.2)
Collecting jedi>=0.16 (from ipython->ipython-autotime)
  Downloading jedi-0.19.1-py2.py3-none-any.whl (1.6 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.6/1.6 MB 9.4 MB/s eta 0:00:00
Requirement already satisfied: decorator in /usr/local/lib/python3.10/dist-packages (from ipython->ipython-autotime) (4.4.2)
Requirement already satisfied: pickleshare in /usr/local/lib/python3.10/dist-packages (from ipython->ipython-autotime) (0.7.5)
Requirement already satisfied: traitlets>=4.2 in /usr/local/lib/python3.10/dist-packages (from ipython->ipython-autotime) (5.7.1)
Requirement already satisfied: prompt-toolkit!=3.0.0,!=3.0.1,<3.1.0,>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from ipython->ipython-autotime) (3.0.39)
Requirement already satisfied: pygments in /usr/local/lib/python3.10/dist-packages (from ipython->ipython-autotime) (2.16.1)
Requirement already satisfied: backcall in /usr/local/lib/python3.10/dist-packages (from ipython->ipython-autotime) (0.2.0)
Requirement already satisfied: matplotlib-inline in /usr/local/lib/python3.10/dist-packages (from ipython->ipython-autotime) (0.1.6)
Requirement already satisfied: pexpect>4.3 in /usr/local/lib/python3.10/dist-packages (from ipython->ipython-autotime) (4.8.0)
Requirement already satisfied: parso<0.9.0,>=0.8.3 in /usr/local/lib/python3.10/dist-packages (from jedi>=0.16->ipython->ipython-autotime) (0.8.3)
Requirement already satisfied: ptyprocess>=0.5 in /usr/local/lib/python3.10/dist-packages (from pexpect>4.3->ipython->ipython-autotime) (0.7.0)
Requirement already satisfied: wcwidth in /usr/local/lib/python3.10/dist-packages (from prompt-toolkit!=3.0.0,!=3.0.1,<3.1.0,>=2.0.0->ipython->ipython-autotime) (0.2.9)
Installing collected packages: jedi, ipython-autotime
Successfully installed ipython-autotime-0.3.2 jedi-0.19.1
```

Imports

```python
import os
import shutil
from typing import List, Tuple, Dict

import itertools
from collections import Counter
import math

from tqdm import tqdm

import warnings
warnings.filterwarnings("ignore")

import csv
import numpy as np
import pandas as pd

from sklearn.metrics import precision_score, recall_score, f1_score, accuracy_score

import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader

from datasets import load_dataset

from dataclasses import dataclass

%load_ext autotime
```

```
time: 366 µs (started: 2023-11-11 02:58:27 +00:00)
```

Config

This code snippet and configuration define several aspects related to the setup and configuration of a Named Entity Recognition (NER) task, likely using GloVe embeddings and the CoNLL 2003 dataset. Here's a brief overview:

1. **CUDA Configuration:**
   - `os.environ["CUDA_LAUNCH_BLOCKING"] = "1"`: This environment variable configuration forces CUDA to synchronize with the CPU, potentially useful for debugging CUDA-related issues.
2. **Working Directory:**
   - The code attempts to set the current working directory to a specific path within Google Drive.
3. **Path Configuration:**
   - `PathConfig` class: Defines paths, especially the path to the GloVe embedding file (`glove.6B.100d.txt`), using the `CURRENT_DIR`.
4. **Dataset Configuration:**
   - `DatasetConfig` class: Contains configuration parameters for dataset processing and preprocessing for the CoNLL 2003 dataset.
   - `name`: Specifies the dataset name.
   - `cols_to_drop`: Lists columns to be dropped during processing.
   - `rename_cols`: Specifies column renaming, especially renaming "ner_tags" to "labels."
   - `THRESHOLD`: Sets a threshold value for some preprocessing step.
   - `PAD_TOKEN` and `UNKNOWN_TOKEN`: Define special tokens ("" and "") for padding and unknown words.
   - `embedding_size`: Specifies the size of GloVe embeddings (100 in this case).
   - `ner_tag2idx` and `ner_idx2tag`: Dictionaries mapping NER tags to indices and vice versa.
   - `NUM_NER_TAGS`: Stores the number of NER tags.

- `SPECIAL_TOKEN_TAG`: A special token tag value.

```
os.environ["CUDA_LAUNCH_BLOCKING"] = "1"

# Set the current working directory
try:
    os.chdir("/content/drive/MyDrive/Colab Notebooks/CSCI544/HW4")
except:
    pass


class PathConfig:
    # Get the current dir
    CURRENT_DIR = os.getcwd()

    # Glove embedding path
    GLOVE_100d_File = os.path.join(CURRENT_DIR, "glove.6B.100d.txt")


class DatasetConfig:
    # General Info
    name = "conll2003"

    # Processing
    cols_to_drop =  ["id", "pos_tags", "chunk_tags"]
    rename_cols = {"ner_tags": "labels"}

    # Preprocessing
    THRESHOLD = 3
    PAD_TOKEN = "<pad>"
    UNKNOWN_TOKEN = "<unk>"
    embedding_size = 100

    # NER Tags list and converter dictionaries
    ner_tag2idx = {'O': 0, 'B-PER': 1, 'I-PER': 2, 'B-ORG': 3, 'I-ORG': 4, 'B-LOC': 5, 'I-LOC': 6, 'B-MISC': 7, 'I-MISC': 8}
    ner_idx2tag = {v: k for k, v in ner_tag2idx.items()}

    NUM_NER_TAGS = len(ner_tag2idx)
    SPECIAL_TOKEN_TAG = -100
```

```
time: 196 ms (started: 2023-11-11 02:58:27 +00:00)
```

Helper Functions & Support Scripts

Accelarator Configuration

```
def get_device():
    if torch.cuda.is_available():
        # Check if GPU is available
        return torch.device("cuda")
    else:
        # Use CPU if no GPU or TPU is available
        return torch.device("cpu")

device = get_device()
device
```

```
device(type='cpu')time: 17.9 ms (started: 2023-11-11 02:58:27 +00:00)
```

CoNLL evaluation functions

```
%%bash
if [ ! -f conlleval.py ]; then
    echo "Downloading conlleval.py ..."
    wget https://raw.githubusercontent.com/sighsmile/conlleval/master/conlleval.py
else
    echo "File conlleval.py already exists"
fi
```

```
File conlleval.py already exists
time: 484 ms (started: 2023-11-11 02:58:27 +00:00)
```

```
from conlleval import evaluate
```

```
time: 747 ms (started: 2023-11-11 02:58:28 +00:00)
```

Helper functions

1. `load_glove_embeddings`
    - It creates a word-to-index mapper (`word2idx`), assigning an index to each word.
    - Special token vectors for unknown words and padding are calculated and added to the embeddings dictionary.
    - Special tokens are added to the word-to-index mapper with specific indices (0 for padding, 1 for unknown).

```
# Load glove embeddings to dictionary
def load_glove_embeddings(path):
    """
    glove_emb_dict = load_glove_embeddings(PathConfig.GLOVE_100d_File)
    """
    embeddings = pd.read_csv(
        PathConfig.GLOVE_100d_File, sep=" ", quoting=csv.QUOTE_NONE, header=None, index_col=0
    )
    embeddings = {key: val.values for key, val in embeddings.T.items()}

    # Generate word-to-index mapper
    word2idx = {word: index for index, word in enumerate(embeddings.keys(), start=2)}

    # Add Special token vectors
    embeddings[DatasetConfig.UNKNOWN_TOKEN] = np.mean(np.vstack(list(embeddings.values())), axis=0)
    embeddings[DatasetConfig.PAD_TOKEN] = np.zeros(DatasetConfig.embedding_size, dtype="float64")

    # Add Special token keys to word-to-index mapper
    word2idx[DatasetConfig.PAD_TOKEN] = 0
    word2idx[DatasetConfig.UNKNOWN_TOKEN] = 1

    return word2idx, embeddings
```

```
time: 1.07 ms (started: 2023-11-11 02:58:28 +00:00)
```

Download Glove Embeddings

```
%%bash
if [ ! -f glove.6B.zip ]; then
    echo "Downloading glove.6B.zip..."
    wget http://nlp.stanford.edu/data/glove.6B.zip -y
    unzip -o glove.6B.zip
else
    echo "File glove.6B.zip already exists"
fi
```

```
File glove.6B.zip already exists
time: 14.8 ms (started: 2023-11-11 02:58:28 +00:00)
```

```
glove_word2idx, glove_emb_dict = load_glove_embeddings(PathConfig.GLOVE_100d_File)
```

```
time: 39.4 s (started: 2023-11-11 02:58:28 +00:00)
```

Dataset Preparation

Process Data

- Utilizes the `load_dataset` function to load a dataset named "conll2003."
- Removes specified columns from the dataset. The columns to be removed are defined in `DatasetConfig.cols_to_drop`.
- Renames columns based on the configuration specified in `DatasetConfig.rename_cols`. It renames the "ner_tags" column to "labels."

```
dataset = load_dataset("conll2003")
dataset = dataset.remove_columns(DatasetConfig.cols_to_drop)
for old_name, new_name in DatasetConfig.rename_cols.items():
    dataset = dataset.rename_column(old_name, new_name)
```

Downloading builder script: 100% 9.57k/9.57k [00:00<00:00, 322kB/s]

Downloading metadata: 100% 3.73k/3.73k [00:00<00:00, 144kB/s]

Downloading readme: 100% 12.3k/12.3k [00:00<00:00, 388kB/s]

Downloading data: 100% 983k/983k [00:00<00:00, 5.12MB/s]

Generating train split: 100% 14041/14041 [00:05<00:00, 1133.64 examples/s]

Generating validation split: 100% 3250/3250 [00:01<00:00, 2008.94 examples/s]

Generating test split: 100% 3453/3453 [00:01<00:00, 4236.00 examples/s]

```
time: 12.5 s (started: 2023-11-11 02:59:08 +00:00)
```

```
dataset
```

```
DatasetDict({
    train: Dataset({
        features: ['tokens', 'labels'],
        num_rows: 14041
    })
    validation: Dataset({
        features: ['tokens', 'labels'],
        num_rows: 3250
    })
    test: Dataset({
        features: ['tokens', 'labels'],
        num_rows: 3453
    })
})time: 3.94 ms (started: 2023-11-11 02:59:20 +00:00)
```

EDA

```
df = pd.DataFrame(dataset["train"])
df.head()
```

	tokens	labels
0	[EU, rejects, German, call, to, boycott, Briti...	[3, 0, 7, 0, 0, 0, 7, 0, 0]
1	[Peter, Blackburn]	[1, 2]
2	[BRUSSELS, 1996-08-22]	[5, 0]
3	[The, European, Commission, said, on, Thursday...	[0, 3, 4, 0, 0, 0, 0, 0, 0, 7, 0, 0, 0, 0, 0, ...
4	[Germany, 's, representative, to, the, Europea...	[5, 0, 0, 0, 0, 3, 4, 0, 0, 0, 1, 2, 0, 0, 0, ...

```
time: 910 ms (started: 2023-11-11 02:59:20 +00:00)
```

Word to index mapper

- Uses the `Counter` class from the `collections` module to count the occurrences of words in the dataset.
- Discards words with frequencies below a specified threshold (`DatasetConfig.THRESHOLD`).
- Generates indexes for the remaining words, starting from index 2.
- Adds special tokens (`PAD_TOKEN` and `UNKNOWN_TOKEN`) with their respective indexes (0 and 1) to the word-to-index mapping.
- The purpose is to create a mapping between words and their corresponding indexes.

```
def generate_word_indexing(dataset, threshold):
    # Count occurences of the words using itertools and Counter
    word_frequency = Counter(itertools.chain(*dataset))

    # Discard words with frequency below threshold
    word_frequency = {
        word: freq
        for word, freq in word_frequency.items()
        if freq >= threshold
    }

    # Generate indexes
    word2idx = {
        word: index
        for index, word in enumerate(word_frequency.keys(), start=2)
```

```
    }

    # Add special tokens
    word2idx[DatasetConfig.PAD_TOKEN] = 0
    word2idx[DatasetConfig.UNKNOWN_TOKEN] = 1

    return word2idx

word2idx = generate_word_indexing(dataset['train']['tokens'], threshold=DatasetConfig.THRESHOLD)
```

```
time: 330 ms (started: 2023-11-11 02:59:21 +00:00)
```

Create GloVe Embeddings Matrix

1. **Initialization**:
    - Initializes an embedding matrix with zeros, where the number of rows is the length of `word2idx` (vocabulary size) and the number of columns is the specified `embedding_dim` (e.g., `DatasetConfig.embedding_size`).
2. **Embedding Matrix Population**:
    - Iterates through each word in the `word2idx` mapping.
    - If the word is present in the pre-trained GloVe embeddings (`glove_emb_dict`), its corresponding vector is used in the embedding matrix.
    - If the word is not found, the vector for the `UNKNOWN_TOKEN` is used.

```
def create_glove_embedding_matrix(word2idx, glove_emb_dict, embedding_dim):
    embedding_matrix = np.zeros((len(word2idx), embedding_dim))

    for word, idx in word2idx.items():
        if word in glove_emb_dict:
            embedding_matrix[idx] = glove_emb_dict[word]
        else:
            embedding_matrix[idx] = glove_emb_dict[DatasetConfig.UNKNOWN_TOKEN]

    return embedding_matrix

glove_embedding_matrix = create_glove_embedding_matrix(glove_word2idx, glove_emb_dict, DatasetConfig.embedding_size)
```

```
time: 1.07 s (started: 2023-11-11 02:59:22 +00:00)
```

Create a Pytorch dataset

1. **Data Class**:
    - A `DatasetItem` data class is defined using the `@dataclass` decorator. It represents an item in the dataset and contains:
        - `embeddings`: Torch tensor representing token embeddings.
        - `targets`: Torch tensor representing target labels.
        - `original_length`: Integer representing the original length of the sequence.
2. **Tokenization Method (tokenize)**:
    - Converts tokens to their respective indexes using the provided tokenizer.
    - Handles different embedding types (custom, glove).
3. **getitem Method**:
    - Overrides the `__getitem__` method to get an item from the dataset at the specified index.
    - Tokenizes input tokens and converts them to indexes.
    - Returns a `DatasetItem` containing Torch tensors for embeddings, targets, and the original length of the sequence.

```
@dataclass
class DatasetItem:
    embeddings: torch.Tensor
    targets: torch.Tensor
    original_length: int


class NERDatasetCustom(Dataset):
    def __init__(self, dataset, split, tokenizer, embedding_type="custom"):
        self.name = DatasetConfig.name
        self.dataset = dataset[split]
        self.tokenizer = tokenizer

        # Options: [custom, glove, transformer]
        self.embedding_type = embedding_type

    def __len__(self):
        return self.dataset.num_rows

    def tokenize(self, tokens):
        """
        Code to convert all tokens to their respective indexes
        """
        if self.embedding_type == "glove":
            return [
                self.tokenizer.get(token.lower(), self.tokenizer[DatasetConfig.UNKNOWN_TOKEN])
                for token in tokens
            ]
        return [
            self.tokenizer.get(token, self.tokenizer[DatasetConfig.UNKNOWN_TOKEN])
            for token in tokens
        ]

    def __getitem__(self, idx):
        if idx >= self.__len__():
            raise IndexError

        item = self.dataset[idx]

        item["input_ids"] = self.tokenize(item["tokens"])

        embeddings = item["input_ids"]
        targets = item["labels"]
        seq_len = len(targets)

        return DatasetItem(
            torch.tensor(embeddings, dtype=torch.long),
            torch.tensor(targets, dtype=torch.long),
            seq_len
        )
```

```
time: 4.49 ms (started: 2023-11-11 02:59:23 +00:00)
```

`collate_fn`:

- Handles padding for sequences in the dataset (`embeddings` and `targets`).
- Iterates through each item in the dataset to extract `embeddings`, `targets`, and `original_length`.
- Uses `nn.utils.rnn.pad_sequence` to pad the `embeddings` and `targets` sequences to the maximum length in the batch.

`collate_fn_transformer`:

- Extends the functionality of `collate_fn` for transformer models by incorporating attention mask handling.
- Creates an key padding mask (`src_key_padding_mask`) based on the padded embeddings.
- Utilizes the attention mask to handle padding for the source sequence in transformers.

```
def collate_fn(data: DatasetItem, tokenizer: dict):
    """
    Collate function for handling padding
    """
    embeddings, targets, og_len = [], [], []

    for item in data:
        embeddings.append(item.embeddings)
        targets.append(item.targets)
        og_len.append(item.original_length)

    # Pad the embeddings sequence
    embeddings = nn.utils.rnn.pad_sequence(
        embeddings, batch_first=True, padding_value=tokenizer[DatasetConfig.PAD_TOKEN]
    )
    # Pad the targets sequence
    targets = nn.utils.rnn.pad_sequence(
        targets, batch_first=True, padding_value=DatasetConfig.SPECIAL_TOKEN_TAG
    )

    return {"embeddings": embeddings, "targets": targets, "original_length": og_len}


def collate_fn_transformer(data: DatasetItem, tokenizer: dict):
    """
    Collate function for handling padding and creating attention mask
    """
    embeddings, targets, og_len = [], [], []

    for item in data:
        embeddings.append(item.embeddings)
        targets.append(item.targets)
        og_len.append(item.original_length)


    # Pad the embeddings sequence
    embeddings = nn.utils.rnn.pad_sequence(
        embeddings, batch_first=True, padding_value=tokenizer[DatasetConfig.PAD_TOKEN]
    )
    # Create attention mask
    src_key_padding_mask = (embeddings != tokenizer[DatasetConfig.PAD_TOKEN]).float().t()
    # Pad the targets sequence
    targets = nn.utils.rnn.pad_sequence(
        targets, batch_first=True, padding_value=DatasetConfig.SPECIAL_TOKEN_TAG
    )

    return {
        "embeddings": embeddings,
        "targets": targets,
        "src_key_padding_mask": src_key_padding_mask,
        "original_length": og_len
    }
```

```
time: 1.08 ms (started: 2023-11-11 02:59:23 +00:00)
```

Training & Evaluation loop

`train_and_evaluate`:

- Train and evaluate a neural network model using the specified parameters.
- Iterates through epochs and batches for training.
- Applies gradient clipping to prevent exploding gradients.
- Evaluates the model on the validation set after each epoch.
- Saves model checkpoints and best model based on validation loss.
- Implements early stopping.
- Returns the best model based on validation loss.

`evaluate_model`:

- Evaluate a trained model on a given data loader.
- Unpads the sequences using the original length obtained from data loader.
- Computes precision, recall, and F1 score using the `evaluate` function.
- Transforms predictions and labels into human-readable format.
- Returns precision, recall, and F1 score.

```
def train_and_evaluate(
    model,
    train_data_loader, valid_data_loader,
    optimizer, loss_fn,
    device,
    num_epochs,
    checkpoint=False,
    path="model.pt",
    early_stopping_patience=5,
    model_type="lstm"
):
    """
    Trains and evaluates the model.

    Args:
        model (nn.Module): The neural network model.
        train_data_loader (DataLoader): The DataLoader for training data.
        valid_data_loader (DataLoader): The DataLoader for validation data.
        optimizer (torch.optim): The optimizer for updating model weights.
        loss_fn: The loss function.
        device (torch.device): The device to perform computations.
        num_epochs (int): The number of epochs.
```

```
        checkpoint (bool, optional): Whether to save model checkpoints.
        path (str, optional): The path to save the model.
        early_stopping_patience (int, optional): Number of epochs to wait before early stopping.
        model_type (str, optional): Type of the model ("lstm" or "transformer").

    Returns:
        nn.Module: The best model.

    """
    # Create directory for saving checkpoint model states
    if checkpoint:
        dirname = path.split(".")[0]
        checkpoint_path = os.path.join(dirname)
        if os.path.exists(checkpoint_path):
            shutil.rmtree(checkpoint_path)
        os.makedirs(dirname)

    best_loss = float('inf')
    no_improvement_count = 0
    best_model = None

    for epoch in range(num_epochs):
        # Train Step
        model.train()
        train_loss = 0.0

        progress_bar = tqdm(train_data_loader, desc=f'Epoch {epoch+1}/{num_epochs}')

        for batch in progress_bar:
            embeddings = batch['embeddings'].to(device, dtype=torch.long, non_blocking=True)
            labels = batch['targets'].to(device, dtype=torch.long, non_blocking=True)
            seq_lengths = batch["original_length"]

            optimizer.zero_grad()

            if model_type == "lstm":
                outputs = model(embeddings, seq_lengths)
            elif model_type == "transformer":
                src_key_padding_mask = batch["src_key_padding_mask"].to(
                    device, dtype=torch.float, non_blocking=True
                )
                outputs = model(embeddings, src_key_padding_mask)

            outputs = outputs.view(-1, outputs.shape[-1])
            labels = labels.view(-1)
            loss = loss_fn(outputs, labels)

            loss.backward()
            # Apply gradient clipping
            torch.nn.utils.clip_grad_norm_(model.parameters(), max_norm=1)

            optimizer.step()

            train_loss += loss.item() * embeddings.size(1)

        train_loss /= len(train_data_loader.dataset)

        # Validation Step
        model.eval()
        valid_loss = 0.0

        with torch.no_grad():
            for batch in valid_data_loader:
                embeddings = batch['embeddings'].to(device, dtype=torch.long, non_blocking=True)
                labels = batch['targets'].to(device, dtype=torch.long, non_blocking=True)
                seq_lengths = batch["original_length"]

                if model_type == "lstm":
                    outputs = model(embeddings, seq_lengths)
                elif model_type == "transformer":
                    src_key_padding_mask = batch["src_key_padding_mask"].to(
                    device, dtype=torch.float, non_blocking=True
                )
                    outputs = model(embeddings, src_key_padding_mask)

                outputs = outputs.view(-1, outputs.shape[-1])
                labels = labels.view(-1)
                loss = loss_fn(outputs, labels)

                valid_loss += loss.item() * embeddings.size(1)

            valid_loss /= len(valid_data_loader.dataset)

        epoch_log = (
            f"Train Loss : {round(train_loss, 4)},"
            f" Validation Loss: {round(valid_loss, 4)}"
        )
        print(epoch_log)

        # Check for improvement in validation loss
        if valid_loss < best_loss:
            # Save checkpoint if needed
            if checkpoint:
                cp = os.path.join(checkpoint_path, f"{dirname}_epoch{epoch+1}_loss{valid_loss:.4f}.pt")
                torch.save(model.state_dict(), cp)
                print(f"Validation loss improved from {best_loss:.4f}--->{valid_loss:.4f}")
                print(f"Saved Checkpoint to '{cp}'")

            best_loss = valid_loss
            best_model = model
            no_improvement_count = 0
        else:
            no_improvement_count += 1

            # Early stopping condition
            if no_improvement_count >= early_stopping_patience:
                print(f"No improvement for {early_stopping_patience} epochs. Stopping early.")
                break

    if checkpoint:
        # Save the best model
        best_model_path = os.path.join(checkpoint_path, f"{dirname}-best.pt")
        torch.save(best_model.state_dict(), best_model_path)
        print(f"Saved best model to '{os.path.relpath(best_model_path)}'")

    # Save current model
    torch.save(model.state_dict(), path)

    return best_model
```

```python


def evaluate_model(model, data_loader, device, model_type="lstm"):
    all_preds = []
    all_labels = []

    model.eval()

    with torch.no_grad():
        for batch in tqdm(data_loader):
            embeddings = batch['embeddings'].to(device, dtype=torch.long, non_blocking=True)
            labels = batch['targets'].to(device, dtype=torch.long, non_blocking=True)
            seq_lengths = batch["original_length"]

            if model_type == "lstm":
                outputs = model(embeddings, seq_lengths)
            elif model_type == "transformer":
                src_key_padding_mask = batch["src_key_padding_mask"].to(
                    device, dtype=torch.float, non_blocking=True
                )
                outputs = model(embeddings, src_key_padding_mask)

            preds = torch.argmax(outputs, dim=2)

            preds = preds.detach().cpu().numpy()
            labels = labels.detach().cpu().numpy()

            for pred, label, length in zip(preds, labels, seq_lengths):
                pred = [DatasetConfig.ner_idx2tag.get(p, 'O') for p in pred[:length]]
                label = [DatasetConfig.ner_idx2tag.get(l, 'O') for l in label[:length]]
                all_preds.append(pred)
                all_labels.append(label)

    # Evaluate using conlleval
    precision, recall, f1 = evaluate(
        itertools.chain(*all_labels), itertools.chain(*all_preds)
    )

    return precision, recall, f1
```

time: 3.42 ms (started: 2023-11-11 02:59:23 +00:00)

Training Config

```python
TRAIN_BATCH_SIZE = 256
VALID_BATCH_SIZE = 64
TEST_BATCH_SIZE = 32
NUM_EPOCHS = 5
```

time: 855 µs (started: 2023-11-11 02:59:23 +00:00)

Model Architectures

Bidirectional LSTM model

```python
class BiLSTM(nn.Module):
    def __init__(
        self, vocab_size, embedding_dim, num_tags,
        hidden_size, num_layers, lstm_output_size, dropout_val,
        embeddings_matrix = None

    ):
        """
        Recurrent Neural Network (RNN) model for sequence data processing.

        Args:
            vocab_size (int): Size of vocabulary
            embedding_dim (int): Dimension of the input features.
            num_tags (int): Number of output classes.
            hidden_size (int): Number of units in the hidden layers.
            num_layers (int): Number of recurrent layers.
            lstm_output_size (int): Size of the output from the LSTM layer.
            dropout_val (float): Dropout probability.
            embeddings_matrix (np.array): Pretrained embeddings matrix. Default is None

        """
        super(BiLSTM, self).__init__()

        # Model Attributes
        self.hidden_size = hidden_size
        self.num_layers = num_layers

        # Model Layer Definition
        if embeddings_matrix is not None:
            self.embedding = nn.Embedding.from_pretrained(
                torch.from_numpy(embeddings_matrix).float(),
                freeze=True
            )
        else:
            self.embedding = nn.Embedding(vocab_size, embedding_dim)

        self.lstm = nn.LSTM(
            embedding_dim, hidden_size, num_layers, batch_first=True, bidirectional=True
        )

        self.fc = nn.Linear(hidden_size * 2, lstm_output_size)
        self.dropout = nn.Dropout(dropout_val)
        self.elu = nn.ELU(alpha=0.01)
        self.classifier = nn.Linear(lstm_output_size, num_tags)

    def init_hidden(self, batch_size):
        hidden = (
            torch.zeros(self.num_layers * 2, batch_size, self.hidden_size).to(device),
            torch.zeros(self.num_layers * 2, batch_size, self.hidden_size).to(device)
        )
        return hidden

    def forward(self, x, seq_len):
        batch_size = x.size(0)
        hidden = self.init_hidden(batch_size)

        # Embedding Layer
```

```
        embeds = self.embedding(x).float()

        # LSTM layer
        packed_embeds = nn.utils.rnn.pack_padded_sequence(
            embeds, seq_len, batch_first=True, enforce_sorted=False
        )
        out, _ = self.lstm(packed_embeds, hidden)
        out, _ = nn.utils.rnn.pad_packed_sequence(out, batch_first=True)

        # Apply fully connected layer for final prediction
        out = self.dropout(out)
        out = self.fc(out)
        out = self.elu(out)
        out = self.classifier(out)

        return out
```

```
time: 12.7 ms (started: 2023-11-10 22:51:08 +00:00)
```

Transformer Encoder model

```
class PositionalEncoding(nn.Module):
    def __init__(self, d_model, dropout: float, max_len=512):
        """
        Positional encoding module for transformer input.

        Args:
            d_model (int): Dimension of the model.
            max_len (int): Maximum length of the input sequence.
        """
        super(PositionalEncoding, self).__init__()

        # Create positional encoding matrix
        self.encoding = torch.zeros(max_len, d_model)
        position = torch.arange(0, max_len).unsqueeze(1).float()
        div_term = torch.exp(torch.arange(0, d_model, 2).float() * -(math.log(10000.0) / d_model))

        # Compute sine and cosine components of positional encoding
        self.encoding[:, 0::2] = torch.sin(position * div_term)
        self.encoding[:, 1::2] = torch.cos(position * div_term)

        # Add batch dimension
        self.encoding = self.encoding.unsqueeze(0)

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

    def forward(self, x):
        device = x.device
        encoding = self.encoding[:, :x.size(1)].detach().to(device)

        # Apply dropout to positional embeddings
        encoding = self.dropout(encoding)

        return x + encoding

class TokenEmbedding(nn.Module):
    def __init__(self, vocab_size, d_model):
        """
        Token embedding module for transformer input.

        Args:
            vocab_size (int): Size of the vocabulary.
            d_model (int): Dimension of the model.
        """
        super(TokenEmbedding, self).__init__()
        self.embedding = nn.Embedding(vocab_size, d_model)

    def forward(self, x):
        return self.embedding(x)

class TransformerNER(nn.Module):
    def __init__(self, vocab_size, d_model, nhead, num_encoder_layers, num_classes, max_len=512):
        """
        Transformer-based NER model.

        Args:
            vocab_size (int): Size of the vocabulary.
            d_model (int): Dimension of the model.
            nhead (int): Number of attention heads in the transformer.
            num_encoder_layers (int): Number of transformer encoder layers.
            num_classes (int): Number of output classes.
            max_len (int): Maximum length of the input sequence.
        """
        super(TransformerNER, self).__init__()
        self.embedding = TokenEmbedding(vocab_size, d_model)
        self.positional_encoding = PositionalEncoding(d_model, 0.25, max_len)
        self.transformer_encoder = nn.TransformerEncoder(
            nn.TransformerEncoderLayer(d_model, nhead),
            num_layers=num_encoder_layers
        )
        self.fc = nn.Linear(d_model, num_classes)

    def forward(self, src, src_key_padding_mask=None):
        """
        Forward pass of the model.

        Args:
            src (torch.Tensor): Input tensor containing token indices.
            src_mask (torch.Tensor, optional): Mask to indicate padding in the input sequence.

        Returns:
            torch.Tensor: Output tensor.
        """
        x = self.embedding(src)
        x = self.positional_encoding(x)
        x = self.transformer_encoder(x, src_key_padding_mask=src_key_padding_mask)
        x = self.fc(x)
        return x
```

```
time: 2.37 ms (started: 2023-11-11 01:15:20 +00:00)
```

Training & Evaluation of Models

- Data Preprocessing Pipeline

- Prepares the data for training, validation, and testing by creating custom datasets (`NERDatasetCustom`) and corresponding data loaders (`DataLoader`).
  - Batch size is provided
  - Passed a valid tokenizer for converting words to tokens.
  - Data Loader is passed a collator function to perform padding and create masks for respective tasks
- Training:
  - The train_and_evaluate function is called to train and evaluate the Transformer model.
  - Training is performed for `N` epochs with early stopping patience set to `P`.
  - Model checkpoints are saved during training.
- Evaluation:
  - Every model is evaluated on validation and test dataset

Bidirectional LSTM model + Custom Embeddings

```
train_dataset = NERDatasetCustom(
    dataset = dataset,
    split='train',
    tokenizer = word2idx,
    embedding_type="default",
)

valid_dataset = NERDatasetCustom(
    dataset = dataset,
    split='validation',
    tokenizer = word2idx,
    embedding_type="default",
)

test_dataset = NERDatasetCustom(
    dataset = dataset,
    split='test',
    tokenizer = word2idx,
    embedding_type="default",
)

train_data_loader = DataLoader(
    train_dataset,
    batch_size=TRAIN_BATCH_SIZE,
    drop_last=True,
    shuffle=True,
    collate_fn=lambda x: collate_fn(x, word2idx)
)

valid_data_loader = DataLoader(
    valid_dataset,
    batch_size=VALID_BATCH_SIZE,
    drop_last=False,
    shuffle=True,
    collate_fn=lambda x: collate_fn(x, word2idx)
)

test_data_loader = DataLoader(
    test_dataset,
    batch_size=TEST_BATCH_SIZE,
    drop_last=False,
    shuffle=False,
    collate_fn=lambda x: collate_fn(x, word2idx)
)
```

```
time: 1.47 ms (started: 2023-11-10 11:17:06 +00:00)
```

```
vocab_size = len(word2idx)
embedding_dim = 100
hidden_size = 256
output_size = 128
num_layers = 1
dropout_val = 0.33
num_tags = DatasetConfig.NUM_NER_TAGS

net = BiLSTM(
    vocab_size, embedding_dim, num_tags,
    hidden_size, num_layers, output_size, dropout_val
).to(device)

criterion = nn.CrossEntropyLoss(ignore_index=DatasetConfig.SPECIAL_TOKEN_TAG).to(device)
# optimizer = optim.Adam(net.parameters(), lr=0.001)
# optimizer = optim.SGD(net.parameters(), lr=0.001)
optimizer = optim.AdamW(net.parameters(), lr=0.001)
# scheduler = torch.optim.lr_scheduler.()

best_model = train_and_evaluate(
    model=net,
    train_data_loader=train_data_loader,
    valid_data_loader=valid_data_loader,
    optimizer=optimizer,
    loss_fn=criterion,
    device=device,
    num_epochs=50,
    checkpoint=True,
    path="bilstm_custom_embeddings_v7.pt",
    early_stopping_patience=15
)
```

```
Saved Checkpoint to 'bilstm_custom_embeddings_v7/bilstm_custom_embeddings_v7_epoch3_loss0.2354.pt'
Epoch 5/50: 100%|██████████| 54/54 [00:04<00:00, 11.79it/s]
Train Loss : 0.0479, Validation Loss: 0.1927
Validation loss improved from 0.2354--->0.1927
```

```
Validation loss improved from 0.1517--->0.1475
Saved Checkpoint to 'bilstm_custom_embeddings_v7/bilstm_custom_embeddings_v7_epoch8_loss0.1475.pt'
Epoch 10/50: 100%|██████████| 54/54 [00:04<00:00, 11.64it/s]
Train Loss : 0.0145, Validation Loss: 0.1437
Validation loss improved from 0.1475--->0.1437
Saved Checkpoint to 'bilstm_custom_embeddings_v7/bilstm_custom_embeddings_v7_epoch9_loss0.1437.pt'
Epoch 11/50: 100%|██████████| 54/54 [00:06<00:00,  8.76it/s]
Train Loss : 0.0122, Validation Loss: 0.1497
Epoch 12/50: 100%|██████████| 54/54 [00:04<00:00, 11.31it/s]
Train Loss : 0.0097, Validation Loss: 0.155
Epoch 13/50: 100%|██████████| 54/54 [00:04<00:00, 11.15it/s]
Train Loss : 0.0082, Validation Loss: 0.1596
Epoch 14/50: 100%|██████████| 54/54 [00:05<00:00,  9.49it/s]
Train Loss : 0.0063, Validation Loss: 0.1621
Epoch 15/50: 100%|██████████| 54/54 [00:04<00:00, 11.74it/s]
Train Loss : 0.0055, Validation Loss: 0.1753
Epoch 16/50: 100%|██████████| 54/54 [00:05<00:00,  9.25it/s]
Train Loss : 0.0048, Validation Loss: 0.1847
Epoch 17/50: 100%|██████████| 54/54 [00:04<00:00, 11.54it/s]
Train Loss : 0.004, Validation Loss: 0.1963
Epoch 18/50: 100%|██████████| 54/54 [00:04<00:00, 11.94it/s]
Train Loss : 0.0034, Validation Loss: 0.2025
Epoch 19/50: 100%|██████████| 54/54 [00:05<00:00,  9.23it/s]
Train Loss : 0.0029, Validation Loss: 0.2086
Epoch 20/50: 100%|██████████| 54/54 [00:04<00:00, 11.94it/s]
Train Loss : 0.0024, Validation Loss: 0.2274
Epoch 21/50: 100%|██████████| 54/54 [00:05<00:00,  9.65it/s]
Train Loss : 0.0022, Validation Loss: 0.2195
Epoch 22/50: 100%|██████████| 54/54 [00:04<00:00, 11.08it/s]
Train Loss : 0.0021, Validation Loss: 0.2279
Epoch 23/50: 100%|██████████| 54/54 [00:04<00:00, 11.91it/s]
Train Loss : 0.0019, Validation Loss: 0.2432
Epoch 24/50: 100%|██████████| 54/54 [00:05<00:00,  9.10it/s]
Train Loss : 0.0018, Validation Loss: 0.2382
Epoch 25/50: 100%|██████████| 54/54 [00:04<00:00, 11.20it/s]
Train Loss : 0.0016, Validation Loss: 0.2442
No improvement for 15 epochs. Stopping early.
Saved best model to 'bilstm_custom_embeddings_v7/bilstm_custom_embeddings_v7-best.pt'
time: 2min 43s (started: 2023-11-10 11:17:06 +00:00)
```

Evaluate model on Validation set

```
precision, recall, f1 = evaluate_model(best_model, valid_data_loader, device)
print(f'Precision: {precision:.2f}, Recall: {recall:.2f}, F1 Score: {f1:.2f}')
```

```
100%|██████████| 51/51 [00:00<00:00, 55.44it/s]
processed 51362 tokens with 5942 phrases; found: 5609 phrases; correct: 4468.
accuracy:  77.97%; (non-O)
accuracy:  95.55%; precision:  79.66%; recall:  75.19%; FB1:  77.36
              LOC: precision:  87.41%; recall:  83.94%; FB1:  85.64  1764
             MISC: precision:  74.81%; recall:  74.73%; FB1:  74.77  921
              ORG: precision:  73.18%; recall:  67.34%; FB1:  70.14  1234
              PER: precision:  78.93%; recall:  72.42%; FB1:  75.54  1690
Precision: 79.66, Recall: 75.19, F1 Score: 77.36
time: 1.03 s (started: 2023-11-10 11:19:49 +00:00)
```

Evaluate model on Test set

```
precision, recall, f1 = evaluate_model(best_model, test_data_loader, device)
print(f'Precision: {precision:.2f}, Recall: {recall:.2f}, F1 Score: {f1:.2f}')
```

```
100%|██████████| 108/108 [00:01<00:00, 70.05it/s]
processed 46435 tokens with 5648 phrases; found: 5274 phrases; correct: 3732.
accuracy:  70.46%; (non-O)
accuracy:  93.39%; precision:  70.76%; recall:  66.08%; FB1:  68.34
              LOC: precision:  81.46%; recall:  76.68%; FB1:  79.00  1570
             MISC: precision:  61.89%; recall:  63.39%; FB1:  62.63  719
              ORG: precision:  65.31%; recall:  59.96%; FB1:  62.52  1525
              PER: precision:  69.32%; recall:  62.59%; FB1:  65.78  1460
Precision: 70.76, Recall: 66.08, F1 Score: 68.34
time: 1.71 s (started: 2023-11-10 11:19:50 +00:00)
```

Bidirectional LSTM Model + Glove Embeddings

```
train_dataset = NERDatasetCustom(
    dataset = dataset,
    split='train',
    tokenizer = glove_word2idx,
    embedding_type="glove",
)

valid_dataset = NERDatasetCustom(
    dataset = dataset,
    split='validation',
    tokenizer = glove_word2idx,
    embedding_type="glove",
)

test_dataset = NERDatasetCustom(
    dataset = dataset,
    split='test',
    tokenizer = glove_word2idx,
    embedding_type="glove",
)

train_data_loader = DataLoader(
    train_dataset,
    batch_size=TRAIN_BATCH_SIZE,
    drop_last=True,
    shuffle=True,
    collate_fn=lambda x: collate_fn(x, glove_word2idx)
)

valid_data_loader = DataLoader(
    valid_dataset,
    batch_size=VALID_BATCH_SIZE,
    drop_last=False,
    shuffle=True,
    collate_fn=lambda x: collate_fn(x, glove_word2idx)
)

test_data_loader = DataLoader(
    test_dataset,
    batch_size=TEST_BATCH_SIZE,
    drop_last=False,
    shuffle=False,
    collate_fn=lambda x: collate_fn(x, glove_word2idx)
)
```

```
time: 1.02 ms (started: 2023-11-11 01:31:10 +00:00)
```

```python
vocab_size = len(glove_word2idx)
embedding_dim = 100
hidden_size = 256
output_size = 128
num_layers = 1
dropout_val = 0.33
num_tags = DatasetConfig.NUM_NER_TAGS

net_with_glove = BiLSTM(
    vocab_size, embedding_dim, num_tags,
    hidden_size, num_layers, output_size, dropout_val,
    glove_embedding_matrix
).to(device)

criterion = nn.CrossEntropyLoss(ignore_index=DatasetConfig.SPECIAL_TOKEN_TAG).to(device)
# optimizer = optim.Adam(net_with_glove.parameters(), lr=0.001)
# optimizer = optim.SGD(net_with_glove.parameters(), lr=0.001)
optimizer = optim.AdamW(net_with_glove.parameters(), lr=0.001)

best_model_glove = train_and_evaluate(
    model=net_with_glove,
    train_data_loader=train_data_loader,
    valid_data_loader=valid_data_loader,
    optimizer=optimizer,
    loss_fn=criterion,
    device=device,
    num_epochs=50,
    checkpoint=True,
    path="bilstm_glove_embeddings_v2.pt",
    early_stopping_patience=15
)
```

```
Train Loss : 0.0221, Validation Loss: 0.0929
Validation loss improved from 0.1019--->0.0929
Saved Checkpoint to 'bilstm_glove_embeddings_v2/bilstm_glove_embeddings_v2_epoch6_loss0.0929.pt'
Epoch 8/50: 100%|██████████| 54/54 [00:07<00:00,  7.67it/s]
Train Loss : 0.0201, Validation Loss: 0.0911
Validation loss improved from 0.0929--->0.0911
Saved Checkpoint to 'bilstm_glove_embeddings_v2/bilstm_glove_embeddings_v2_epoch7_loss0.0911.pt'
Epoch 9/50: 100%|██████████| 54/54 [00:05<00:00,  9.50it/s]
Train Loss : 0.0181, Validation Loss: 0.0846
Validation loss improved from 0.0911--->0.0846
Saved Checkpoint to 'bilstm_glove_embeddings_v2/bilstm_glove_embeddings_v2_epoch8_loss0.0846.pt'
Epoch 10/50: 100%|██████████| 54/54 [00:07<00:00,  7.59it/s]
Train Loss : 0.0167, Validation Loss: 0.0824
Validation loss improved from 0.0846--->0.0824
Saved Checkpoint to 'bilstm_glove_embeddings_v2/bilstm_glove_embeddings_v2_epoch9_loss0.0824.pt'
Epoch 11/50: 100%|██████████| 54/54 [00:05<00:00,  9.61it/s]
Train Loss : 0.0152, Validation Loss: 0.0789
Validation loss improved from 0.0824--->0.0789
Saved Checkpoint to 'bilstm_glove_embeddings_v2/bilstm_glove_embeddings_v2_epoch10_loss0.0789.pt'
Epoch 12/50: 100%|██████████| 54/54 [00:06<00:00,  8.10it/s]
Train Loss : 0.0137, Validation Loss: 0.0799
Epoch 13/50: 100%|██████████| 54/54 [00:07<00:00,  6.80it/s]
Train Loss : 0.0124, Validation Loss: 0.0763
Validation loss improved from 0.0789--->0.0763
Saved Checkpoint to 'bilstm_glove_embeddings_v2/bilstm_glove_embeddings_v2_epoch12_loss0.0763.pt'
Epoch 14/50: 100%|██████████| 54/54 [00:06<00:00,  8.46it/s]
Train Loss : 0.0111, Validation Loss: 0.0793
Epoch 15/50: 100%|██████████| 54/54 [00:05<00:00,  9.05it/s]
Train Loss : 0.0104, Validation Loss: 0.0796
Epoch 16/50: 100%|██████████| 54/54 [00:05<00:00, 10.49it/s]
Train Loss : 0.0089, Validation Loss: 0.0839
Epoch 17/50: 100%|██████████| 54/54 [00:06<00:00,  8.39it/s]
Train Loss : 0.0082, Validation Loss: 0.0807
Epoch 18/50: 100%|██████████| 54/54 [00:04<00:00, 11.08it/s]
Train Loss : 0.0075, Validation Loss: 0.0887
Epoch 19/50: 100%|██████████| 54/54 [00:06<00:00,  8.87it/s]
Train Loss : 0.0065, Validation Loss: 0.0881
Epoch 20/50: 100%|██████████| 54/54 [00:05<00:00, 10.05it/s]
Train Loss : 0.0057, Validation Loss: 0.0902
Epoch 21/50: 100%|██████████| 54/54 [00:04<00:00, 11.49it/s]
Train Loss : 0.0051, Validation Loss: 0.0884
Epoch 22/50: 100%|██████████| 54/54 [00:06<00:00,  8.10it/s]
Train Loss : 0.0045, Validation Loss: 0.0926
Epoch 23/50: 100%|██████████| 54/54 [00:04<00:00, 11.22it/s]
Train Loss : 0.004, Validation Loss: 0.0991
Epoch 24/50: 100%|██████████| 54/54 [00:07<00:00,  7.70it/s]
Train Loss : 0.0038, Validation Loss: 0.1022
Epoch 25/50: 100%|██████████| 54/54 [00:06<00:00,  7.92it/s]
Train Loss : 0.0036, Validation Loss: 0.0987
Epoch 26/50: 100%|██████████| 54/54 [00:04<00:00, 11.25it/s]
Train Loss : 0.0033, Validation Loss: 0.1046
Epoch 27/50: 100%|██████████| 54/54 [00:06<00:00,  8.92it/s]
Train Loss : 0.0027, Validation Loss: 0.1069
Epoch 28/50: 100%|██████████| 54/54 [00:05<00:00, 10.06it/s]
Train Loss : 0.0023, Validation Loss: 0.1173
No improvement for 15 epochs. Stopping early.
Saved best model to 'bilstm_glove_embeddings_v2/bilstm_glove_embeddings_v2-best.pt'
time: 3min 28s (started: 2023-11-11 01:31:21 +00:00)
```

Evaluate model on Validation set

```python
precision, recall, f1 = evaluate_model(best_model_glove, valid_data_loader, device)
print(f'Precision: {precision:.2f}, Recall: {recall:.2f}, F1 Score: {f1:.2f}')
```

```
100%|██████████| 51/51 [00:01<00:00, 36.51it/s]
processed 51362 tokens with 5942 phrases; found: 5939 phrases; correct: 5201.
accuracy:  87.32%; (non-O)
accuracy:  97.41%; precision:  87.57%; recall:  87.53%; FB1:  87.55
              LOC: precision:  90.43%; recall:  93.14%; FB1:  91.77  1892
             MISC: precision:  81.57%; recall:  77.77%; FB1:  79.62  879
              ORG: precision:  79.33%; recall:  77.85%; FB1:  78.58  1316
              PER: precision:  93.36%; recall:  93.87%; FB1:  93.61  1852
Precision: 87.57, Recall: 87.53, F1 Score: 87.55
time: 1.59 s (started: 2023-11-11 01:36:07 +00:00)
```

Evaluate model on Test set

```python
precision, recall, f1 = evaluate_model(best_model_glove, test_data_loader, device)
print(f'Precision: {precision:.2f}, Recall: {recall:.2f}, F1 Score: {f1:.2f}')
```

```
100%|██████████| 108/108 [00:01<00:00, 91.99it/s]
processed 46435 tokens with 5648 phrases; found: 5700 phrases; correct: 4727.
accuracy:  85.22%; (non-O)
accuracy:  96.53%; precision:  82.93%; recall:  83.69%; FB1:  83.31
              LOC: precision:  83.96%; recall:  89.75%; FB1:  86.76  1783
             MISC: precision:  70.70%; recall:  69.09%; FB1:  69.88  686
              ORG: precision:  78.51%; recall:  77.42%; FB1:  77.96  1638
              PER: precision:  91.59%; recall:  90.23%; FB1:  90.90  1593
```

```
Precision: 82.93, Recall: 83.69, F1 Score: 83.31
time: 1.28 s (started: 2023-11-11 01:36:14 +00:00)
```

Transformer Encoder + Custom Embeddings

```python
train_dataset = NERDatasetCustom(
    dataset = dataset,
    split='train',
    tokenizer = word2idx,
    embedding_type="transformer",
)

valid_dataset = NERDatasetCustom(
    dataset = dataset,
    split='validation',
    tokenizer = word2idx,
    embedding_type="transformer",
)

test_dataset = NERDatasetCustom(
    dataset = dataset,
    split='test',
    tokenizer = word2idx,
    embedding_type="transformer",
)

train_data_loader = DataLoader(
    train_dataset,
    batch_size=TRAIN_BATCH_SIZE,
    drop_last=True,
    shuffle=True,
    collate_fn=lambda x: collate_fn_transformer(x, word2idx)
)

valid_data_loader = DataLoader(
    valid_dataset,
    batch_size=VALID_BATCH_SIZE,
    drop_last=False,
    shuffle=True,
    collate_fn=lambda x: collate_fn_transformer(x, word2idx)
)

test_data_loader = DataLoader(
    test_dataset,
    batch_size=TEST_BATCH_SIZE,
    drop_last=False,
    shuffle=False,
    collate_fn=lambda x: collate_fn_transformer(x, word2idx)
)
```

```
time: 1.4 ms (started: 2023-11-11 01:37:02 +00:00)
```

```python
vocab_size = len(word2idx)
embedding_dim = 128
num_attention_heads = 8
num_encoder_layers = 6
num_classes = DatasetConfig.NUM_NER_TAGS
max_len = 128  # Sequence max length

net_with_transformer = TransformerNER(
    vocab_size, embedding_dim, num_attention_heads,
    num_encoder_layers, num_classes, max_len
).to(device)

# Define the loss function and optimizer
criterion = nn.CrossEntropyLoss(ignore_index=DatasetConfig.SPECIAL_TOKEN_TAG).to(device)
optimizer = optim.AdamW(net_with_transformer.parameters(), lr=0.0001, betas=(0.9, 0.98), eps=1e-9)

# Train the Transformer model
best_model_transformer = train_and_evaluate(
    model=net_with_transformer,
    train_data_loader=train_data_loader,
    valid_data_loader=valid_data_loader,
    optimizer=optimizer,
    loss_fn=criterion,
    device=device,
    num_epochs=100,
    checkpoint=True,
    path="transformer_enc_model_v2.pt",
    early_stopping_patience=15,
    model_type="transformer"
)
```

```
Train Loss : 0.0394, Validation Loss: 0.2358
Epoch 40/100: 100%|██████████| 54/54 [00:12<00:00,  4.16it/s]
Train Loss : 0.039, Validation Loss: 0.2428
Epoch 41/100: 100%|██████████| 54/54 [00:12<00:00,  4.25it/s]
Train Loss : 0.0392, Validation Loss: 0.2422
Epoch 42/100: 100%|██████████| 54/54 [00:12<00:00,  4.28it/s]
Train Loss : 0.0384, Validation Loss: 0.2378
Epoch 43/100: 100%|██████████| 54/54 [00:12<00:00,  4.17it/s]
Train Loss : 0.0386, Validation Loss: 0.2403
Epoch 44/100: 100%|██████████| 54/54 [00:14<00:00,  3.72it/s]
Train Loss : 0.0382, Validation Loss: 0.2356
Epoch 45/100: 100%|██████████| 54/54 [00:12<00:00,  4.22it/s]
Train Loss : 0.0376, Validation Loss: 0.2382
Epoch 46/100: 100%|██████████| 54/54 [00:12<00:00,  4.26it/s]
Train Loss : 0.0381, Validation Loss: 0.2428
Epoch 47/100: 100%|██████████| 54/54 [00:12<00:00,  4.30it/s]
Train Loss : 0.0373, Validation Loss: 0.2379
Epoch 48/100: 100%|██████████| 54/54 [00:13<00:00,  4.12it/s]
Train Loss : 0.0377, Validation Loss: 0.2447
No improvement for 15 epochs. Stopping early.
Saved best model to 'transformer_enc_model_v2/transformer_enc_model_v2-best.pt'
time: 11min 30s (started: 2023-11-11 01:37:49 +00:00)
```

Evaluate model on Validation set

```
precision, recall, f1 = evaluate_model(
    best_model_transformer, valid_data_loader, device, model_type="transformer"
)
print(f'Precision: {precision:.2f}, Recall: {recall:.2f}, F1 Score: {f1:.2f}')
```

```
100%|██████████| 51/51 [00:02<00:00, 22.68it/s]
processed 51362 tokens with 5942 phrases; found: 4857 phrases; correct: 3132.
accuracy:  53.28%; (non-O)
accuracy:  92.05%; precision:  64.48%; recall:  52.71%; FB1:  58.01
              LOC: precision:  81.53%; recall:  73.27%; FB1:  77.18  1651
             MISC: precision:  71.82%; recall:  64.97%; FB1:  68.22  834
              ORG: precision:  55.45%; recall:  47.80%; FB1:  51.34  1156
              PER: precision:  44.90%; recall:  29.64%; FB1:  35.71  1216
Precision: 64.48, Recall: 52.71, F1 Score: 58.01
time: 2.45 s (started: 2023-11-11 01:50:22 +00:00)
```

Evaluate model on Test set

```
precision, recall, f1 = evaluate_model(
    best_model_transformer, test_data_loader, device, model_type="transformer"
)
print(f'Precision: {precision:.2f}, Recall: {recall:.2f}, F1 Score: {f1:.2f}')
```

```
100%|██████████| 108/108 [00:02<00:00, 50.06it/s]
processed 46435 tokens with 5648 phrases; found: 4029 phrases; correct: 2242.
accuracy:  41.62%; (non-O)
accuracy:  89.46%; precision:  55.65%; recall:  39.70%; FB1:  46.34
              LOC: precision:  73.55%; recall:  65.53%; FB1:  69.31  1486
             MISC: precision:  61.25%; recall:  57.41%; FB1:  59.26  658
              ORG: precision:  49.25%; recall:  33.71%; FB1:  40.03  1137
              PER: precision:  24.87%; recall:  11.50%; FB1:  15.73  748
Precision: 55.65, Recall: 39.70, F1 Score: 46.34
time: 2.34 s (started: 2023-11-11 01:50:32 +00:00)
```

References

1. https://huggingface.co/datasets/conll2003
2. https://huggingface.co/docs/datasets/installation
3. https://huggingface.co/docs/transformers/installation
4. https://stackoverflow.com/questions/37793118/load-pretrained-glove-vectors-in-python
5. https://stackoverflow.com/a/52070223/12639940
6. https://github.com/sighsmile/conlleval
7. https://nlp.stanford.edu/data/glove.6B.zip
8. https://stats.stackexchange.com/questions/248715/selection-of-values-for-padding-tokens-in-sentence-classification-with-word-embe
9. https://pytorch.org/tutorials/beginner/translation_transformer.html

THE END