# CSCI 544 Assignment 4

**Data:**

The CoNLL-2003 corpus for the tasks was downloaded from the given link utilizing the datasets library. Depending on the task, the varied embeddings required varied text data processing.

**Task 1: No specific embeddings**

Using the script shared by instructors on piazza, a vocabulary(word2idx) was created from the training dataset, keeping words that have a frequency >= threshold.
For this assignment, the threshold was set to 2.
In word2idx, as suggested, index for “PAD” was set to 0, and 1 for “UNK” or unknown words. Following assignment guidelines, the words in the datasets (train, test and val) were mapped to their corresponding index in the vocabulary, and columns such as ‘pos_tags’ and ‘chunk_tags’ were dropped from the dataset.

As labels were already replaced by corresponding indices in the data, a dictionary was created for mapping reference to evaluate predictions. The dictionary,

‘id2label’ had 10 entries: 9 labels, given values from 0 to 9, and a “PAD”, labelled -1 to easily distinguish labels from other NER tags.

Now that the data we need is numeric, it is ready to be passed into the model, save for one problem: input lengths. A model would need a defined input size in a batch, but different sentences have different lengths. Thus, to ensure every “batch” has the same length, the following procedure was followed:

- Calculate lengths of original sentences
- PAD inputs of the batch to match the length of the largest sentence in the batch, with 0 (as word2idx as PAD as 0)
- PAD labels of the batch to the same length, with -1 (as -1 is defined in id2label)
- Return inputs, labels and original lengths

We define train, val and test DataLoaders, assigning a self-defined collate function to the dataloaders. The collate function follows the procedure above.
With the DataLoaders prepared, the data was ready to be trained on.

**Task 2: Glove-100 embeddings**

Since Glove embeddings were used in this task, the vocabulary used would be all the words present in the embeddings.
The embeddings were loaded into a dictionary, which was used to calculate a word2idx similar to task 1.

A problem to solve here was the **case-insensitivity of the Glove embeddings**. Since NER tags treat capitalization as important information, having the same would be essential for good model performance.

To solve the following problem, an additional dimension to each embedding was taken, making the embedding dimensions to be 101, instead of 100. The extra bit was designed to handle the case of each word, defined as:

- -1.0 for lowercase
- 0 for titlecase
- 1.0 for uppercase

Also, in word2idx, there were 3 kinds of unknown tags used:

- [‘UNKL’]: lower case
- [‘UNKT’]: title case
- [‘UNKU’]: upper case

Thus, the size of word2idx and the word embeddings dictionary was nearly tripled, as the for every word, there were 3 versions of embeddings, for each case.
Examples of the word “happy”, “Happy”, and “HAPPY” are shown in notebook.

The words in the datasets were mapped to the the word2idx ids, and the DataLoaders were prepared following the same fashion as that of task1

An embedding matrix was created using the word embeddings dictionary. Each index was mapped back to its corresponding embedding, with the addition of 4 extra embedding vectors:

- ‘PAD’ embedding: an array of 101 zeros
- ‘UNKL’ embedding: an array of 101, with the first 100 being the average of all other word embeddings, and the last element being -1.0
- ‘UNKU’ embedding: an array of 101, with the first 100 being the average of all other word embeddings, and the last element being 1.0
- ‘UNKT’ embedding: an array of 101, with the first 100 being the average of all other word embeddings, and the last element being 0.0

This embedding matrix was converted into a tensor, frozen and fed to the embedding layer of the model

**Model:**

This model is a Bi LSTM, designed for both task 1 and task 2 in the homework. Here's the architecture for both models:

Task 1:

```
class BiLSTMNER(nn.Module):
    def __init__(self, vocab_size, tagset_size, embedding_dim=100, hidden_dim=256, linear_dim=128, dropout=0.33, num_layers=1):
        super(BiLSTMNER, self).__init__()
        self.embedding = nn.Embedding(vocab_size, embedding_dim)
        self.lstm = nn.LSTM(embedding_dim, hidden_dim,
                            num_layers=num_layers,
                            dropout=dropout if num_layers > 1 else 0,
                            batch_first = True,
                            bidirectional=True)
        self.linear = nn.Linear(2*hidden_dim, linear_dim)
        self.elu = nn.ELU(alpha = 0.75)
        self.dropout = nn.Dropout(p=dropout)
        self.classifier = nn.Linear(linear_dim, tagset_size)

    def forward(self, sentence, lengths):
        embedded = self.embedding(sentence)

        # Pack the embeddings
        packed_embedded = pack_padded_sequence(embedded, lengths.cpu(), batch_first=True, enforce_sorted=False)

        packed_lstm_out, _ = self.lstm(packed_embedded)

        # Unpack the sequence
        lstm_out, _ = pad_packed_sequence(packed_lstm_out, batch_first=True)
        lstm_out = self.dropout(lstm_out)
        linear_out = self.elu(self.linear(lstm_out))
        tag_space = self.classifier(linear_out)

        return tag_space.permute(0,2,1)
```

Task 2:

```python
class BiLSTMNER(nn.Module):
    def __init__(self, vocab_size, tagset_size, embedding_dim=101, hidden_dim=256, linear_dim=128, dropout=0.33, num_layers=1):
        super(BiLSTMNER, self).__init__()
        self.embedding = nn.Embedding.from_pretrained(embedding_matrix, freeze=True)
        self.lstm = nn.LSTM(embedding_dim,
                            hidden_dim,
                            num_layers=num_layers,
                            dropout=dropout if num_layers > 1 else 0,
                            batch_first = True,
                            bidirectional=True)
        self.linear = nn.Linear(2*hidden_dim, linear_dim)
        self.elu = nn.ELU(alpha = 0.5) #0.75
        self.dropout = nn.Dropout(p=dropout)
        self.classifier = nn.Linear(linear_dim, tagset_size)

    def forward(self, sentence, lengths):
        embedded = self.embedding(sentence)

        # Pack the embeddings
        packed_embedded = pack_padded_sequence(embedded, lengths.cpu(), batch_first=True, enforce_sorted=False)

        packed_lstm_out, _ = self.lstm(packed_embedded)

        # Unpack the sequence
        lstm_out, _ = pad_packed_sequence(packed_lstm_out, batch_first=True)
        lstm_out = self.dropout(lstm_out)
        linear_out = self.elu(self.linear(lstm_out))
        tag_space = self.classifier(linear_out)

        return tag_space.permute(0,2,1)
```

A peculiar feature to note in the forward propagations of each model is the use of torch.utils.rnn's pack_padded_sequence() and pad_packed_sequence(). These functions help the model focus only on the words of the sentence and not the padding, thus avoiding unnecessary information being passed to the model, which might confuse the model. Using these helped improve model accuracy significantly.

**Model Training:** (Mention hyperparameters & training clearly)
Similar architectures were used for the Bi-LSTM models in both tasks. As defined in the assignment guidelines, following was the structure of each model:

**Task 1:**

```
BiLSTMNER(
  (embedding): Embedding(1127234, 100)
  (lstm): LSTM(101, 256, batch_first=True, bidirectional=True)
  (linear): Linear(in_features=512, out_features=128, bias=True)
  (elu): ELU(alpha=0.75)
  (dropout): Dropout(p=0.33, inplace=False)
  (classifier): Linear(in_features=128, out_features=9, bias=True)
)
```

BATCH SIZE: 16

**Task 2:**

```
BiLSTMNER(
  (embedding): Embedding(1127234, 101)
  (lstm): LSTM(101, 256, batch_first=True, bidirectional=True)
  (linear): Linear(in_features=512, out_features=128, bias=True)
  (elu): ELU(alpha=0.5)
  (dropout): Dropout(p=0.33, inplace=False)
  (classifier): Linear(in_features=128, out_features=9, bias=True)
)
```

BATCH SIZE: 32

The following training procedure was followed for both models:

- Define Optimizer: torch.optim.SGD
- Define Loss: nn.CrossEntropyLoss(weighted)

- Model was moved to gpu(if exists)
- Setting learning rate to 0.75 or 75e-2, train the model for 100 epochs:
    - Fetch inputs, labels and lengths from DataLoader
    - Pass inputs, labels and lengths to gpu(if present)
    - Fetch model predictions passing inputs and labels
    - Calculate loss, gradient and update model weights
    - Calculate training and validation loss every epoch
- After 100 epochs, when training and validation losses do not improve by significant amount, reduce learning rate to 0.5 or 5e-1 and continue training (20 or more epochs)

Post training, model predictions were fetched using a self-defined helper function, and the predictions were evaluated following the script mentioned in the assignment guidelines.

**Model Performance Overview:**

(as reported by the evaluation function of conlleval.py)

**Task 1**

| Dataset | Precision(%) | Recall (%) | F1-Score(%) |
|---|---|---|---|
| Validation | 82.14 | 73.91 | 77.81 |

| | | | |
|---|---|---|---|
| Test | 70.40 | 64.89 | 67.53 |

**Task 2**

| Dataset | Precision(%) | Recall (%) | F1-Score(%) |
|---|---|---|---|
| Validation | 89.37 | 92.80 | 91.05 |
| Test | 83.29 | 88.7 | 85.91 |

Q) BiLSTM with Glove Embeddings outperforms the model without. Can you provide a rationale for this?

⇒ The BiLSTM with Glove Embeddings outperforms the model without due to the enhanced ability of Glove embeddings to capture semantic relationships and context in the data.

⇒ This results in a more nuanced understanding of the input, leading to improved model performance in tasks requiring contextual comprehension.

⇒ Also, Glove embeddings provide pre-trained word representations, leveraging a larger corpus for learning. This helps the model generalize better to a wide range of language patterns and nuances, contributing to its overall superior performance compared to a model without such embeddings.

The bonus task follows the same data processing as task 1. The data is loaded, and the instructor's recommended code is used to prepare word2idx.
To prepare the data for the model, a different collate function is used for the transformer model, as this model does not need the lengths of the original sentences (or sequences), but a source mask, that marks the indexes of the padding on the input sequences. Once the DataLoaders are prepared, they are passed to the model for prediction.

The model's architecture follows the guidelines mentioned in the assignment, as well as the Positional and Token Embedding Class as per the reference links in the assignment.
Following is the model architecture:

```
TransformerNER(
  (token_embedding): Embedding(23625, 128)
  (pos_encoder): PositionalEncoding()
  (dropout): Dropout(p=0.1, inplace=False)
  (transformer_encoder): TransformerEncoder(
    (layers): ModuleList(
      (0-5): 6 x TransformerEncoderLayer(
        (self_attn): MultiheadAttention(
```

```
        (out_proj):
NonDynamicallyQuantizableLinear(in_features=128,
out_features=128, bias=True)
      )
      (linear1): Linear(in_features=128, out_features=512,
bias=True)
      (dropout): Dropout(p=0.1, inplace=False)
      (linear2): Linear(in_features=512, out_features=128,
bias=True)
      (norm1): LayerNorm((128,), eps=1e-05,
elementwise_affine=True)
      (norm2): LayerNorm((128,), eps=1e-05,
elementwise_affine=True)
      (dropout1): Dropout(p=0.1, inplace=False)
      (dropout2): Dropout(p=0.1, inplace=False)
    )
   )
  )
  (classifier): Linear(in_features=128, out_features=9,
bias=True)
)
```

Using the AdamW optimizer and learning rate as 1e-4, the transformer is trained for 60 epochs.

Model predictions are generated using the same user defined function. The predictions are then evaluated using the conlleval.py's evaluation function.

| Dataset | Precision(%) | Recall (%) | F1-Score(%) |
|---|---|---|---|
| Validation | 67.21 | 67.03 | 67.12 |
| Test | 57.50 | 53.43 | 55.39 |

Q) What is the reason behind the poor performance of the transformer?

⇒ The poor performance of the transformer model may be due to factors such as limited training data or inadequate hyperparameter tuning. BiLSTM models, especially when enhanced with GloVe embeddings, could outperform transformers in scenarios with smaller datasets or less complex tasks.

⇒ Transformers trained from scratch without leveraging pre-trained models, it may not perform as well because it lacks this pre-trained contextual knowledge.

⇒ The CoNLL-2003 dataset might not require the full representational power of a Transformer model. Simple models can be quite competitive on tasks where the input data doesn't require complex reasoning or extensive context.

Following pages of the pdf are the jupyter notebooks for this assignment taskwise.

# SN_NLP_HW4_task1

November 10, 2023

[ ]:
```
!pip install datasets
```

```
Collecting datasets
  Downloading datasets-2.14.6-py3-none-any.whl (493 kB)
                             493.7/493.7
kB 10.0 MB/s eta 0:00:00
Requirement already satisfied: numpy>=1.17 in
/usr/local/lib/python3.10/dist-packages (from datasets) (1.23.5)
Requirement already satisfied: pyarrow>=8.0.0 in /usr/local/lib/python3.10/dist-
packages (from datasets) (9.0.0)
Collecting dill<0.3.8,>=0.3.0 (from datasets)
  Downloading dill-0.3.7-py3-none-any.whl (115 kB)
                             115.3/115.3
kB 13.4 MB/s eta 0:00:00
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-
packages (from datasets) (1.5.3)
Requirement already satisfied: requests>=2.19.0 in
/usr/local/lib/python3.10/dist-packages (from datasets) (2.31.0)
Requirement already satisfied: tqdm>=4.62.1 in /usr/local/lib/python3.10/dist-
packages (from datasets) (4.66.1)
Requirement already satisfied: xxhash in /usr/local/lib/python3.10/dist-packages
(from datasets) (3.4.1)
Collecting multiprocess (from datasets)
  Downloading multiprocess-0.70.15-py310-none-any.whl (134 kB)
                             134.8/134.8
kB 18.1 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-0.18.0-py3-none-any.whl (301 kB)
                             302.0/302.0
kB 15.5 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: 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: 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)
Installing collected packages: dill, multiprocess, huggingface-hub, datasets
Successfully installed datasets-2.14.6 dill-0.3.7 huggingface-hub-0.18.0
multiprocess-0.70.15
```

[ ]:
```
!wget https://raw.githubusercontent.com/sighsmile/conlleval/master/conlleval.py
```

```
--2023-11-08 04:26:22--
https://raw.githubusercontent.com/sighsmile/conlleval/master/conlleval.py
Resolving raw.githubusercontent.com (raw.githubusercontent.com)…
185.199.108.133, 185.199.109.133, 185.199.110.133, …
Connecting to raw.githubusercontent.com
(raw.githubusercontent.com)|185.199.108.133|:443… connected.
HTTP request sent, awaiting response… 200 OK
```

```
Length: 7502 (7.3K) [text/plain]
Saving to: ‘conlleval.py’

conlleval.py        100%[===================>]   7.33K  --.-KB/s    in 0s

2023-11-08 04:26:22 (93.7 MB/s) - ‘conlleval.py’ saved [7502/7502]
```

[ ]:
```
import torch
import torch.nn as nn
from torch.utils.data import DataLoader
from torch.optim import Adam,SGD
from torch.nn.functional import cross_entropy
from torch.nn.utils.rnn import pad_sequence
from tqdm import tqdm
from torch.nn.utils.rnn import pack_padded_sequence, pad_packed_sequence
```

[ ]:
```
import datasets

dataset = datasets.load_dataset("conll2003")
```

```
Downloading builder script:   0%|          | 0.00/9.57k [00:00<?, ?B/s]
Downloading metadata:   0%|          | 0.00/3.73k [00:00<?, ?B/s]
Downloading readme:   0%|          | 0.00/12.3k [00:00<?, ?B/s]
Downloading data:   0%|          | 0.00/983k [00:00<?, ?B/s]
Generating train split:   0%|          | 0/14041 [00:00<?, ? examples/s]
Generating validation split:   0%|          | 0/3250 [00:00<?, ? examples/s]
Generating test split:   0%|          | 0/3453 [00:00<?, ? examples/s]
```

[ ]:
```
import itertools
from collections import Counter
# REFERENCE CODE PROVIDED BY SHOUMIK
word_frequency = Counter(itertools.chain(*dataset['train']['tokens']))  # type:␣
 ↪ignore

# Remove words below threshold 2
word2idx = {
    word: frequency
    for word, frequency in word_frequency.items()
    if frequency >= 2
}

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

```
}

word2idx['[PAD]'] = 0
word2idx['[UNK]'] = 1
```

[ ]:
```
dataset = (
    dataset
    .map(lambda x: {
            'input_ids': [
                word2idx.get(word, word2idx['[UNK]'])
                for word in x['tokens']
            ]
        }
    )
)

dataset['train']['input_ids'][:3]
```

```
Map:   0%|          | 0/14041 [00:00<?, ? examples/s]
Map:   0%|          | 0/3250 [00:00<?, ? examples/s]
Map:   0%|          | 0/3453 [00:00<?, ? examples/s]
```

[ ]:
```
[[2, 3, 4, 5, 6, 7, 8, 9, 10], [11, 12], [13, 14]]
```

[ ]:
```
dataset
```

[ ]:
```
DatasetDict({
    train: Dataset({
        features: ['id', 'tokens', 'labels', 'input_ids'],
        num_rows: 14041
    })
    validation: Dataset({
        features: ['id', 'tokens', 'labels', 'input_ids'],
        num_rows: 3250
    })
    test: Dataset({
        features: ['id', 'tokens', 'labels', 'input_ids'],
        num_rows: 3453
    })
})
```

[ ]:
```
columns_to_remove = ['pos_tags', 'chunk_tags']
for split in dataset.keys():
    dataset[split] = dataset[split].remove_columns(columns_to_remove)

# Rename ner_tags to labels
for split in dataset.keys():
```

```
    dataset[split] = dataset[split].rename_column('ner_tags', 'labels')

print(dataset)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-23-d5fc103fe155> in <cell line: 2>()
      1 columns_to_remove = ['pos_tags', 'chunk_tags']
      2 for split in dataset.keys():
----> 3     dataset[split] = dataset[split].remove_columns(columns_to_remove)
      4
      5 # Rename ner_tags to labels

/usr/local/lib/python3.10/dist-packages/datasets/arrow_dataset.py in␣
 ↪wrapper(*args, **kwargs)
    590             self: "Dataset" = kwargs.pop("self")
    591         # apply actual function
--> 592         out: Union["Dataset", "DatasetDict"] = func(self, *args,␣
 ↪**kwargs)
    593         datasets: List["Dataset"] = list(out.values()) if␣
 ↪isinstance(out, dict) else [out]
    594         for dataset in datasets:

/usr/local/lib/python3.10/dist-packages/datasets/arrow_dataset.py in␣
 ↪wrapper(*args, **kwargs)
    555         }
    556         # apply actual function
--> 557         out: Union["Dataset", "DatasetDict"] = func(self, *args,␣
 ↪**kwargs)
    558         datasets: List["Dataset"] = list(out.values()) if␣
 ↪isinstance(out, dict) else [out]
    559         # re-apply format to the output

/usr/local/lib/python3.10/dist-packages/datasets/fingerprint.py in␣
 ↪wrapper(*args, **kwargs)
    509             # Call actual function
    510
--> 511             out = func(dataset, *args, **kwargs)
    512
    513             # Update fingerprint of in-place transforms + update␣
 ↪in-place history of transforms

/usr/local/lib/python3.10/dist-packages/datasets/arrow_dataset.py in␣
 ↪remove_columns(self, column_names, new_fingerprint)
   2153         for column_name in column_names:
   2154             if column_name not in dataset._data.column_names:
```

```
-> 2155                 raise ValueError(
   2156                     f"Column name {column_name} not in the dataset. "
   2157                     f"Current columns in the dataset: {dataset._data.
 ↪column_names}"

ValueError: Column name pos_tags not in the dataset. Current columns in the␣
 ↪dataset: ['id', 'tokens', 'labels', 'input_ids']
```

[ ]:
```
label2id = dataset["train"].features["labels"].feature
id2label = {id: label for label, id in enumerate(label2id.names)}
# label2id.names
id2label['PAD'] = -1
id2label
```

[ ]:
```
{'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,
 'PAD': -1}
```

[ ]:

Task 1: Bidirectional LSTM model

[ ]:
```
class BiLSTMNER(nn.Module):
    def __init__(self, vocab_size, tagset_size, embedding_dim=100,␣
 ↪hidden_dim=256, linear_dim=128, dropout=0.33, num_layers=1):
        super(BiLSTMNER, self).__init__()
        self.embedding = nn.Embedding(vocab_size, embedding_dim)
        self.lstm = nn.LSTM(embedding_dim, hidden_dim,
                            num_layers=num_layers,
                            dropout=dropout if num_layers > 1 else 0,
                            batch_first = True,
                            bidirectional=True)
        self.linear = nn.Linear(2*hidden_dim, linear_dim)
        self.elu = nn.ELU(alpha = 0.75)
        self.dropout = nn.Dropout(p=dropout)
        self.classifier = nn.Linear(linear_dim, tagset_size)

    def forward(self, sentence, lengths):
        embedded = self.embedding(sentence)
```

```
        # Pack the embeddings
        packed_embedded = pack_padded_sequence(embedded, lengths.cpu(),␣
 ↪batch_first=True, enforce_sorted=False)

        packed_lstm_out, _ = self.lstm(packed_embedded)

        # Unpack the sequence
        lstm_out, _ = pad_packed_sequence(packed_lstm_out, batch_first=True)
        lstm_out = self.dropout(lstm_out)
        linear_out = self.elu(self.linear(lstm_out))
        tag_space = self.classifier(linear_out)

        return tag_space.permute(0,2,1)
```

[ ]:
```
import torch
from torch.utils.data import DataLoader, TensorDataset
from torch.nn.utils.rnn import pad_sequence

def preprocess_data(data):
    input_ids = [torch.tensor(seq) for seq in data['input_ids']]
    labels = [torch.tensor(label) for label in data['labels']]
    return list(zip(input_ids, labels))

def dynamic_padding(batch):
    inputs = [item[0] for item in batch]
    labels = [item[1] for item in batch]
    lengths = torch.tensor([len(inp) for inp in inputs])

    # Dynamic padding in the batch
    inputs = pad_sequence(inputs, batch_first=True)
    labels = pad_sequence(labels, batch_first=True, padding_value=-1)

    return inputs, labels, lengths


# Hyperparameters
BATCH_SIZE = 32
# Preprocess the train, val, and test data
train_data = preprocess_data(dataset['train'])
val_data = preprocess_data(dataset['validation'])
test_data = preprocess_data(dataset['test'])

train_loader = DataLoader(train_data, batch_size=BATCH_SIZE, shuffle=True,␣
 ↪collate_fn=dynamic_padding, num_workers=2)
val_loader = DataLoader(val_data, batch_size=BATCH_SIZE,␣
 ↪collate_fn=dynamic_padding, num_workers=2)
```

```
test_loader = DataLoader(test_data, batch_size=BATCH_SIZE,␣
 ↪collate_fn=dynamic_padding, num_workers=2)
```

[ ]:
```
vocab_size = max([max(seq) for seq in dataset['train']['input_ids']]) + 1
tagset_size = max([max(seq) for seq in dataset['train']['labels']]) + 1
# Device definition
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
all_labels = [label for sublist in dataset['train']['labels'] for label in␣
 ↪sublist]
label_counts = {label: all_labels.count(label) for label in id2label.values()␣
 ↪if label != -1}
total_labels = len(all_labels) # We subtract the count of 'PAD' labels
weights = [total_labels / label_counts[id2label[key]] for key in id2label.
 ↪keys() if key != 'PAD']
# Normalization idea 1: dividing by max weight
# max_weight = max(weights)
# weights = [weight / max_weight for weight in weights]
# Normalization idea 2: dividing by sum of weights
weights = [weight / sum(weights) for weight in weights]

weights_tensor = torch.tensor(weights).to(device)
```

[ ]:
```
sample_batch = next(iter(train_loader))
len(sample_batch)
```

[ ]:
```
3
```

[ ]:
```
model = BiLSTMNER(vocab_size, tagset_size)
model
```

[ ]:
```
LEARNING_RATE = 5e-1 #75e-2 #3e-1#5e-1#75e-2#1.0 #5e-1#1e-1
loss_function = torch.nn.CrossEntropyLoss(weight=weights_tensor,␣
 ↪ignore_index=-1).to(device)
# Optimizer
# optimizer = Adam(model.parameters(), lr=LEARNING_RATE)
optimizer = SGD(model.parameters(), lr=LEARNING_RATE)
# Move model to the device
model = model.to(device)
```

[ ]:
```
# Loss function
# loss_function = torch.nn.CrossEntropyLoss(ignore_index=-1).to(device)
#120 so far
EPOCHS = 20#100
for epoch in range(EPOCHS):
    model.train()
    total_loss = 0
```

```
    # Wrap your training loader with tqdm for progress bar
    for inputs, targets, lengths in train_loader: #tqdm(train_loader,␣
↪desc=f"Epoch {epoch + 1}/{EPOCHS}"):
        optimizer.zero_grad()

        # Fetch inputs and targets and move them to the current device
        inputs = inputs.to(device)
        targets = targets.to(device)
        lengths = lengths.to(device)

        # Forward pass
        outputs = model(inputs, lengths)

        # Compute loss and backpropagate
        loss = loss_function(outputs,targets) #(outputs.view(-1, tagset_size),␣
↪targets.view(-1))
        loss.backward()
        optimizer.step()

        total_loss += loss.item()
    # Validation
    model.eval()
    val_loss = 0
    with torch.no_grad():
        for inputs, targets, lengths in val_loader:
            inputs = inputs.to(device)
            targets = targets.to(device)
            lengths = lengths.to(device)

            outputs = model(inputs, lengths)

            loss = loss_function(outputs,targets) #(outputs.view(-1,␣
↪tagset_size), targets.view(-1))
            val_loss += loss.item()

    print(f"Epoch {epoch + 1}/{EPOCHS}, Training Loss: {total_loss /␣
↪len(train_loader)} Validation Loss: {val_loss / len(val_loader)}")
```

```
Epoch 1/20, Training Loss: 0.005469360117599525 Validation Loss:
2.287004227712331
Epoch 2/20, Training Loss: 0.005165160213441191 Validation Loss:
2.2760457327584573
Epoch 3/20, Training Loss: 0.0043842436359324945 Validation Loss:
2.337395059049817
Epoch 4/20, Training Loss: 0.004425346395035289 Validation Loss:
2.3204613802518588
Epoch 5/20, Training Loss: 0.003768849798078025 Validation Loss:
```

```
2.412500930118569
Epoch 6/20, Training Loss: 0.003826590034966718 Validation Loss:
2.3090612552867547
Epoch 7/20, Training Loss: 0.003562867568888286 Validation Loss:
2.3641352345741447
Epoch 8/20, Training Loss: 0.0035704300531498017 Validation Loss:
2.3746119245032515
Epoch 9/20, Training Loss: 0.003481599730786874 Validation Loss:
2.4231946685686325
Epoch 10/20, Training Loss: 0.0029924438989804504 Validation Loss:
2.418409530529567
Epoch 11/20, Training Loss: 0.0033467773290150556 Validation Loss:
2.3444884500752265
Epoch 12/20, Training Loss: 0.0027910100784341555 Validation Loss:
2.4335305455708767
Epoch 13/20, Training Loss: 0.0029307371941180308 Validation Loss:
2.419531327196672
Epoch 14/20, Training Loss: 0.0031870355251199323 Validation Loss:
2.419259003286586
Epoch 15/20, Training Loss: 0.0024465062133976496 Validation Loss:
2.413897556894314
Epoch 16/20, Training Loss: 0.0026728114813162067 Validation Loss:
2.455604896179907
Epoch 17/20, Training Loss: 0.002937689349575049 Validation Loss:
2.426296499837752
Epoch 18/20, Training Loss: 0.0023684401665187062 Validation Loss:
2.466683931882209
Epoch 19/20, Training Loss: 0.002733392068534291 Validation Loss:
2.410260948969567
Epoch 20/20, Training Loss: 0.002615849651615183 Validation Loss:
2.407557534061146
```

[ ]:
```python
def get_predictions(model, loader, device):
    model.eval()
    all_predictions = []
    with torch.no_grad():
        for inputs, _, lengths in loader: # We don't need targets now
            inputs = inputs.to(device)

            outputs = model(inputs, lengths)
            # Get predictions
            predictions = torch.argmax(outputs, dim=1)

            # Truncate predictions to their original lengths
            truncated_predictions = [pred[:len_].tolist() for pred, len_ in␣
 ↪zip(predictions, lengths)]
```

```
            all_predictions.extend(truncated_predictions)

    return all_predictions
```

[ ]:
```
val_predictions = get_predictions(model, val_loader, device)
test_predictions = get_predictions(model, test_loader, device)
```

[ ]:
```
from conlleval import evaluate
import itertools
# labels = ner_tags
# Map the labels back to their corresponding tag strings
idx2tag  = {id:tag for (tag,id) in id2label.items()}
labels = [
list(map(idx2tag.get, labels))
for labels in dataset['validation']['labels']
]
# This is the prediction by your model
preds = [
list(map(idx2tag.get, labels))
for labels in val_predictions
]
precision, recall, f1 = evaluate(itertools.chain(*labels),itertools.
  ↪chain(*preds))
```

```
processed 51362 tokens with 5942 phrases; found: 5347 phrases; correct: 4392.
accuracy:  75.56%; (non-O)
accuracy:  95.06%; precision:  82.14%; recall:  73.91%; FB1:  77.81
              LOC: precision:  82.14%; recall:  85.85%; FB1:  83.95  1920
             MISC: precision:  89.00%; recall:  76.36%; FB1:  82.19  791
              ORG: precision:  79.58%; recall:  67.11%; FB1:  72.82  1131
              PER: precision:  80.47%; recall:  65.74%; FB1:  72.36  1505
```

[ ]:
```
from conlleval import evaluate
import itertools
# labels = ner_tags
# Map the labels back to their corresponding tag strings
idx2tag  = {id:tag for (tag,id) in id2label.items()}
labels = [
list(map(idx2tag.get, labels))
for labels in dataset['test']['labels']
]
# This is the prediction by your model
preds = [
list(map(idx2tag.get, labels))
for labels in test_predictions
]
precision, recall, f1 = evaluate(itertools.chain(*labels),itertools.
  ↪chain(*preds))
```

```
processed 46435 tokens with 5648 phrases; found: 5206 phrases; correct: 3665.
accuracy:  68.69%; (non-O)
accuracy:  92.81%; precision:  70.40%; recall:  64.89%; FB1:  67.53
              LOC: precision:  71.74%; recall:  79.14%; FB1:  75.26  1840
             MISC: precision:  74.51%; recall:  64.53%; FB1:  69.16  608
              ORG: precision:  69.66%; recall:  60.69%; FB1:  64.86  1447
              PER: precision:  67.43%; recall:  54.67%; FB1:  60.38  1311
```

[ ]:
```python
# code to save pytorch model
# Move the model to CPU
model.to('cpu')

# Save the model's state_dict
torch.save(model.state_dict(), 'model_hw4_task1.pth')
```

[ ]:
```
```

# SN_NLP_HW4_task2

November 10, 2023

[ ]:
```
!pip install datasets
```

```
Collecting datasets
  Downloading datasets-2.14.6-py3-none-any.whl (493 kB)
                             493.7/493.7
kB 7.5 MB/s eta 0:00:00
Requirement already satisfied: numpy>=1.17 in
/usr/local/lib/python3.10/dist-packages (from datasets) (1.23.5)
Requirement already satisfied: pyarrow>=8.0.0 in /usr/local/lib/python3.10/dist-
packages (from datasets) (9.0.0)
Collecting dill<0.3.8,>=0.3.0 (from datasets)
  Downloading dill-0.3.7-py3-none-any.whl (115 kB)
                             115.3/115.3
kB 5.2 MB/s eta 0:00:00
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-
packages (from datasets) (1.5.3)
Requirement already satisfied: requests>=2.19.0 in
/usr/local/lib/python3.10/dist-packages (from datasets) (2.31.0)
Requirement already satisfied: tqdm>=4.62.1 in /usr/local/lib/python3.10/dist-
packages (from datasets) (4.66.1)
Requirement already satisfied: xxhash in /usr/local/lib/python3.10/dist-packages
(from datasets) (3.4.1)
Collecting multiprocess (from datasets)
  Downloading multiprocess-0.70.15-py310-none-any.whl (134 kB)
                             134.8/134.8
kB 10.0 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-0.18.0-py3-none-any.whl (301 kB)
                             302.0/302.0
kB 13.1 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: 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: 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)
Installing collected packages: dill, multiprocess, huggingface-hub, datasets
Successfully installed datasets-2.14.6 dill-0.3.7 huggingface-hub-0.18.0
multiprocess-0.70.15
```

[ ]:
```
!wget http://nlp.stanford.edu/data/glove.6B.zip
!unzip glove.6B.zip
```

```
--2023-11-08 04:34:06--  http://nlp.stanford.edu/data/glove.6B.zip
Resolving nlp.stanford.edu (nlp.stanford.edu)… 171.64.67.140
Connecting to nlp.stanford.edu (nlp.stanford.edu)|171.64.67.140|:80…
connected.
HTTP request sent, awaiting response… 302 Found
Location: https://nlp.stanford.edu/data/glove.6B.zip [following]
```

```
--2023-11-08 04:34:06--  https://nlp.stanford.edu/data/glove.6B.zip
Connecting to nlp.stanford.edu (nlp.stanford.edu)|171.64.67.140|:443…
connected.
HTTP request sent, awaiting response… 301 Moved Permanently
Location: https://downloads.cs.stanford.edu/nlp/data/glove.6B.zip [following]
--2023-11-08 04:34:07--  https://downloads.cs.stanford.edu/nlp/data/glove.6B.zip
Resolving downloads.cs.stanford.edu (downloads.cs.stanford.edu)… 171.64.64.22
Connecting to downloads.cs.stanford.edu
(downloads.cs.stanford.edu)|171.64.64.22|:443… connected.
HTTP request sent, awaiting response… 200 OK
Length: 862182613 (822M) [application/zip]
Saving to: ‘glove.6B.zip’

glove.6B.zip        100%[===================>] 822.24M  5.00MB/s    in 2m 39s

2023-11-08 04:36:46 (5.16 MB/s) - ‘glove.6B.zip’ saved [862182613/862182613]

Archive:  glove.6B.zip
  inflating: glove.6B.50d.txt
  inflating: glove.6B.100d.txt
  inflating: glove.6B.200d.txt
  inflating: glove.6B.300d.txt
```

[ ]:
```
!wget https://raw.githubusercontent.com/sighsmile/conlleval/master/conlleval.py
```

```
--2023-11-08 04:37:09--
https://raw.githubusercontent.com/sighsmile/conlleval/master/conlleval.py
Resolving raw.githubusercontent.com (raw.githubusercontent.com)…
185.199.109.133, 185.199.110.133, 185.199.108.133, …
Connecting to raw.githubusercontent.com
(raw.githubusercontent.com)|185.199.109.133|:443… connected.
HTTP request sent, awaiting response… 200 OK
Length: 7502 (7.3K) [text/plain]
Saving to: ‘conlleval.py’

conlleval.py        100%[===================>]   7.33K  --.-KB/s    in 0s

2023-11-08 04:37:09 (104 MB/s) - ‘conlleval.py’ saved [7502/7502]
```

[ ]:
```
import torch
import torch.nn as nn
from torch.utils.data import DataLoader
from torch.optim import Adam,SGD
from torch.nn.functional import cross_entropy
from torch.nn.utils.rnn import pad_sequence
from tqdm import tqdm
from torch.nn.utils.rnn import pack_padded_sequence, pad_packed_sequence
```

```
import numpy as np
import pandas as pd
```

[ ]:
```
import datasets

dataset = datasets.load_dataset("conll2003")
```

```
Downloading builder script:   0%|          | 0.00/9.57k [00:00<?, ?B/s]
Downloading metadata:   0%|          | 0.00/3.73k [00:00<?, ?B/s]
Downloading readme:   0%|          | 0.00/12.3k [00:00<?, ?B/s]
Downloading data:   0%|          | 0.00/983k [00:00<?, ?B/s]
Generating train split:   0%|          | 0/14041 [00:00<?, ? examples/s]
Generating validation split:   0%|          | 0/3250 [00:00<?, ? examples/s]
Generating test split:   0%|          | 0/3453 [00:00<?, ? examples/s]
```

[ ]:
```
def load_embeddings(filename):
    word2vec = {}
    with open(filename, 'r', encoding='utf-8') as f:
        for line in f:
            line = line.strip().split()
            word = line[0]
            embedding = [float(x) for x in line[1:]]
            word2vec[word] = np.array(embedding)
    return word2vec

word2vec = load_embeddings('glove.6B.100d.txt')
keys = list(word2vec.keys())
for i in keys:
    word2vec[i] = np.append(word2vec[i],-1.0)#non-capitalized
    word2vec[i.title()] = np.append(word2vec[i][:-1],0.0) # title
    word2vec[i.upper()] = np.append(word2vec[i][:-1],1.0) # fully capitalized
```

[ ]:
```
word2vec['happy']
```

[ ]:
```
array([-0.090436 ,  0.19636  ,  0.29474  , -0.47706  , -0.80436  ,
        0.3078   , -0.55205  ,  0.58453  , -0.17056  , -0.84846  ,
        0.19528  ,  0.23671  ,  0.46827  , -0.58977  , -0.12163  ,
       -0.24697  , -0.072944 ,  0.17259  , -0.0485   ,  0.9527   ,
        0.50629  ,  0.58497  , -0.19367  , -0.45459  , -0.031095 ,
        0.51633  , -0.24052  , -0.1007   ,  0.53627  ,  0.024225 ,
       -0.50162  ,  0.73692  ,  0.49468  , -0.34744  ,  0.89337  ,
        0.057439 , -0.19127  ,  0.39333  ,  0.21182  , -0.89837  ,
        0.078704 , -0.16344  ,  0.45261  , -0.41096  , -0.19499  ,
       -0.13489  , -0.016313 , -0.021849 ,  0.17136  , -1.2413   ,
        0.079503 , -0.91144  ,  0.35699  ,  0.36289  , -0.24934  ,
```

```
       -2.1196   ,  0.14534  ,  0.52964  ,  0.90134  ,  0.033603 ,
        0.022809 ,  0.70625  , -1.0362   , -0.59809  ,  0.70592  ,
       -0.072793 ,  0.67033  ,  0.52763  , -0.47807  , -0.67374  ,
        0.36632  , -0.38284  , -0.10349  , -0.6402   ,  0.18104  ,
        0.82568  ,  0.066403 , -0.40791  , -0.083813 , -0.36487  ,
        0.045362 , -0.073527 , -0.20117  ,  0.37441  , -1.4024   ,
       -0.25605  , -0.4708   , -0.16145  , -0.87921  , -0.36325  ,
       -0.17357  , -0.077983 ,  0.43273  ,  0.0089295, -1.0316   ,
       -0.11589  , -0.34524  ,  0.11514  , -0.40812  ,  0.20203  ,
       -1.       ])
```

[ ]:
```
word2vec['Happy']
```

[ ]:
```
array([-0.090436 ,  0.19636  ,  0.29474  , -0.47706  , -0.80436  ,
        0.3078   , -0.55205  ,  0.58453  , -0.17056  , -0.84846  ,
        0.19528  ,  0.23671  ,  0.46827  , -0.58977  , -0.12163  ,
       -0.24697  , -0.072944 ,  0.17259  , -0.0485   ,  0.9527   ,
        0.50629  ,  0.58497  , -0.19367  , -0.45459  , -0.031095 ,
        0.51633  , -0.24052  , -0.1007   ,  0.53627  ,  0.024225 ,
       -0.50162  ,  0.73692  ,  0.49468  , -0.34744  ,  0.89337  ,
        0.057439 , -0.19127  ,  0.39333  ,  0.21182  , -0.89837  ,
        0.078704 , -0.16344  ,  0.45261  , -0.41096  , -0.19499  ,
       -0.13489  , -0.016313 , -0.021849 ,  0.17136  , -1.2413   ,
        0.079503 , -0.91144  ,  0.35699  ,  0.36289  , -0.24934  ,
       -2.1196   ,  0.14534  ,  0.52964  ,  0.90134  ,  0.033603 ,
        0.022809 ,  0.70625  , -1.0362   , -0.59809  ,  0.70592  ,
       -0.072793 ,  0.67033  ,  0.52763  , -0.47807  , -0.67374  ,
        0.36632  , -0.38284  , -0.10349  , -0.6402   ,  0.18104  ,
        0.82568  ,  0.066403 , -0.40791  , -0.083813 , -0.36487  ,
        0.045362 , -0.073527 , -0.20117  ,  0.37441  , -1.4024   ,
       -0.25605  , -0.4708   , -0.16145  , -0.87921  , -0.36325  ,
       -0.17357  , -0.077983 ,  0.43273  ,  0.0089295, -1.0316   ,
       -0.11589  , -0.34524  ,  0.11514  , -0.40812  ,  0.20203  ,
        0.       ])
```

[ ]:
```
word2vec['HAPPY']
```

[ ]:
```
array([-0.090436 ,  0.19636  ,  0.29474  , -0.47706  , -0.80436  ,
        0.3078   , -0.55205  ,  0.58453  , -0.17056  , -0.84846  ,
        0.19528  ,  0.23671  ,  0.46827  , -0.58977  , -0.12163  ,
       -0.24697  , -0.072944 ,  0.17259  , -0.0485   ,  0.9527   ,
        0.50629  ,  0.58497  , -0.19367  , -0.45459  , -0.031095 ,
        0.51633  , -0.24052  , -0.1007   ,  0.53627  ,  0.024225 ,
       -0.50162  ,  0.73692  ,  0.49468  , -0.34744  ,  0.89337  ,
        0.057439 , -0.19127  ,  0.39333  ,  0.21182  , -0.89837  ,
        0.078704 , -0.16344  ,  0.45261  , -0.41096  , -0.19499  ,
       -0.13489  , -0.016313 , -0.021849 ,  0.17136  , -1.2413   ,
```

```
 0.079503 , -0.91144  ,  0.35699  ,  0.36289  , -0.24934  ,
-2.1196   ,  0.14534  ,  0.52964  ,  0.90134  ,  0.033603 ,
 0.022809 ,  0.70625  , -1.0362   , -0.59809  ,  0.70592  ,
-0.072793 ,  0.67033  ,  0.52763  , -0.47807  , -0.67374  ,
 0.36632  , -0.38284  , -0.10349  , -0.6402   ,  0.18104  ,
 0.82568  ,  0.066403 , -0.40791  , -0.083813 , -0.36487  ,
 0.045362 , -0.073527 , -0.20117  ,  0.37441  , -1.4024   ,
-0.25605  , -0.4708   , -0.16145  , -0.87921  , -0.36325  ,
-0.17357  , -0.077983 ,  0.43273  ,  0.0089295, -1.0316   ,
-0.11589  , -0.34524  ,  0.11514  , -0.40812  ,  0.20203  ,
 1.       ])
```

[ ]:
```python
import itertools
from collections import Counter
# REFERENCE CODE PROVIDED BY SHOUMIK
# word_frequency = Counter(itertools.chain(*dataset['train']['tokens']))  #␣
 ↪type: ignore

# # Remove words below threshold 2
# word2idx = {
#     word: frequency
#     for word, frequency in word_frequency.items()
#     if frequency >= 2
# }

# word2idx = {
#     word: index
#     for index, word in enumerate(word_frequency.keys(), start=2)
# }
word2idx = {word: i+4 for i, word in enumerate(word2vec)} # + 4 for padding,␣
 ↪unknown lower, unknown title, unknown upper
word2idx['[PAD]'] = 0 #padding
word2idx['[UNKL]'] = 1 #unkown lowercase word
word2idx['[UNKT]'] = 2 #unkown titlecase word
word2idx['[UNKU]'] = 3 #unkown uppercase word
```

[ ]:
```python
def which_unknown(word2idx,word):
    """
    decides which unkown case to assign a particular word
    """
    if word.istitle():
        return word2idx['[UNKT]']
    if word.isupper():
        return word2idx['[UNKU]']
    #if nothing else, lets just assume its a lower case:
    return word2idx['[UNKL]']
```

```
dataset = (
    dataset
    .map(lambda x: {
            'input_ids': [
                word2idx.get(word,which_unknown(word2idx,word))␣
 ↪#word2idx['[UNK]']
                for word in x['tokens']
            ]
        }
    )
)

dataset['train']['input_ids'][:3]
```

```
Map:   0%|          | 0/14041 [00:00<?, ? examples/s]
Map:   0%|          | 0/3250 [00:00<?, ? examples/s]
Map:   0%|          | 0/3453 [00:00<?, ? examples/s]
```

[ ]:
```
[[401213, 7582, 400966, 584, 8, 5264, 400551, 10242, 6],
 [402419, 417195],
 [407382, 1]]
```

[ ]:
```
dataset['train']['tokens'][:3]
```

[ ]:
```
[['EU', 'rejects', 'German', 'call', 'to', 'boycott', 'British', 'lamb', '.'],
 ['Peter', 'Blackburn'],
 ['BRUSSELS', '1996-08-22']]
```

[ ]:
```
dataset
```

[ ]:
```
DatasetDict({
    train: Dataset({
        features: ['id', 'tokens', 'pos_tags', 'chunk_tags', 'ner_tags',
'input_ids'],
        num_rows: 14041
    })
    validation: Dataset({
        features: ['id', 'tokens', 'pos_tags', 'chunk_tags', 'ner_tags',
'input_ids'],
        num_rows: 3250
    })
    test: Dataset({
        features: ['id', 'tokens', 'pos_tags', 'chunk_tags', 'ner_tags',
'input_ids'],
        num_rows: 3453
    })
})
```

[ ]:
```
columns_to_remove = ['pos_tags', 'chunk_tags']
for split in dataset.keys():
    dataset[split] = dataset[split].remove_columns(columns_to_remove)

# Rename ner_tags to labels
for split in dataset.keys():
    dataset[split] = dataset[split].rename_column('ner_tags', 'labels')

print(dataset)
```

```
DatasetDict({
    train: Dataset({
        features: ['id', 'tokens', 'labels', 'input_ids'],
        num_rows: 14041
    })
    validation: Dataset({
        features: ['id', 'tokens', 'labels', 'input_ids'],
        num_rows: 3250
    })
    test: Dataset({
        features: ['id', 'tokens', 'labels', 'input_ids'],
        num_rows: 3453
    })
})
```

[ ]:
```
label2id = dataset["train"].features["labels"].feature
id2label = {id: label for label, id in enumerate(label2id.names)}
id2label['PAD'] = -1
id2label
```

[ ]:
```
{'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,
 'PAD': -1}
```

[ ]:
```
# creating embedding matrix for model
embedding_matrix = np.zeros((len(word2idx), 101))
random_embeddings = np.random.randn(101)
avg_vector = np.mean(np.array(list(word2vec.values())), axis=0) # Average␣
 ↪vector for unknown words

# Fill in embedding matrix with GloVe vectors
```

```
embedding_matrix[word2idx['[PAD]']] = np.zeros(101)#np.random.randn(101)
# embedding_matrix[word2idx['[UNK]']] = random_embeddings
embedding_matrix[word2idx['[UNKL]']] = np.append(avg_vector[:-1],-1.0)
embedding_matrix[word2idx['[UNKT]']] = np.append(avg_vector[:-1],0.0)
embedding_matrix[word2idx['[UNKU]']] = np.append(avg_vector[:-1],1.0)
for i, word in enumerate(word2vec):
    embedding_matrix[i+4] = word2vec[word]

# converting into torch tensor and Freezeing the embeddings:
embedding_matrix = torch.from_numpy(embedding_matrix.astype('float32'))
embedding_matrix.requires_grad = False
```

Task 1: Bidirectional LSTM model

[ ]:
```
class BiLSTMNER(nn.Module):
    def __init__(self, vocab_size, tagset_size, embedding_dim=101,␣
 ↪hidden_dim=256, linear_dim=128, dropout=0.33, num_layers=1):
        super(BiLSTMNER, self).__init__()
        self.embedding = nn.Embedding.from_pretrained(embedding_matrix,␣
 ↪freeze=True)
        self.lstm = nn.LSTM(embedding_dim,
                            hidden_dim,
                            num_layers=num_layers,
                            dropout=dropout if num_layers > 1 else 0,
                            batch_first = True,
                            bidirectional=True)
        self.linear = nn.Linear(2*hidden_dim, linear_dim)
        self.elu = nn.ELU(alpha = 0.5) #0.75
        self.dropout = nn.Dropout(p=dropout)
        self.classifier = nn.Linear(linear_dim, tagset_size)

    def forward(self, sentence, lengths):
        embedded = self.embedding(sentence)

        # Pack the embeddings
        packed_embedded = pack_padded_sequence(embedded, lengths.cpu(),␣
 ↪batch_first=True, enforce_sorted=False)

        packed_lstm_out, _ = self.lstm(packed_embedded)

        # Unpack the sequence
        lstm_out, _ = pad_packed_sequence(packed_lstm_out, batch_first=True)
        lstm_out = self.dropout(lstm_out)
        linear_out = self.elu(self.linear(lstm_out))
        tag_space = self.classifier(linear_out)

        return tag_space.permute(0,2,1)
```

[ ]:
```
import torch
from torch.utils.data import DataLoader, TensorDataset
from torch.nn.utils.rnn import pad_sequence

def preprocess_data(data):
    input_ids = [torch.tensor(seq) for seq in data['input_ids']]
    labels = [torch.tensor(label) for label in data['labels']]
    return list(zip(input_ids, labels))

def dynamic_padding(batch):
    inputs = [item[0] for item in batch]
    labels = [item[1] for item in batch]
    lengths = torch.tensor([len(inp) for inp in inputs])

    # Dynamic padding in the batch
    inputs = pad_sequence(inputs, batch_first=True)
    labels = pad_sequence(labels, batch_first=True, padding_value=-1)

    return inputs, labels, lengths


# Hyperparameters
BATCH_SIZE = 16#32
# Preprocess the train, val, and test data
train_data = preprocess_data(dataset['train'])
val_data = preprocess_data(dataset['validation'])
test_data = preprocess_data(dataset['test'])

train_loader = DataLoader(train_data, batch_size=BATCH_SIZE, shuffle=True,␣
 ↪collate_fn=dynamic_padding, num_workers=2)
val_loader = DataLoader(val_data, batch_size=BATCH_SIZE,␣
 ↪collate_fn=dynamic_padding, num_workers=2)
test_loader = DataLoader(test_data, batch_size=BATCH_SIZE,␣
 ↪collate_fn=dynamic_padding, num_workers=2)
```

[ ]:
```
from collections import Counter

vocab_size = max([max(seq) for seq in dataset['train']['input_ids']]) + 1
tagset_size = max([max(seq) for seq in dataset['train']['labels']]) + 1
# Device definition
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
all_labels = [label for sublist in dataset['train']['labels'] for label in␣
 ↪sublist]
label_counts = {label: all_labels.count(label) for label in id2label.values()␣
 ↪if label != -1}
total_labels = len(all_labels) # We subtract the count of 'PAD' labels
```

```
weights = [total_labels / label_counts[id2label[key]] for key in id2label.
 ↪keys() if key != 'PAD']
weights = [weight / sum(weights) for weight in weights]

weights_tensor = torch.tensor(weights).to(device)
```

[ ]:
```
label_counts
```

[ ]:
```
{0: 169578,
 1: 6600,
 2: 4528,
 3: 6321,
 4: 3704,
 5: 7140,
 6: 1157,
 7: 3438,
 8: 1155}
```

[ ]:
```
weights
```

[ ]:
```
[0.0019872122378459785,
 0.05105870861658262,
 0.07442302934395877,
 0.05331236780089311,
 0.09097934040751764,
 0.04719712561196713,
 0.2912597034308084,
 0.09801846331281132,
 0.291764049237615]
```

[ ]:
```
sample_batch = next(iter(train_loader))
len(sample_batch)
```

[ ]:
```
3
```

[ ]:
```
model = BiLSTMNER(vocab_size, tagset_size)
model
```

[ ]:
```
BiLSTMNER(
  (embedding): Embedding(1127234, 101)
  (lstm): LSTM(101, 256, batch_first=True, bidirectional=True)
  (linear): Linear(in_features=512, out_features=128, bias=True)
  (elu): ELU(alpha=0.5)
  (dropout): Dropout(p=0.33, inplace=False)
  (classifier): Linear(in_features=128, out_features=9, bias=True)
)
```

[ ]:
```
LEARNING_RATE = 75e-2#75e-3#1e-1 #75e-2 #3e-1 #75e-2#1.0 #5e-1#1e-1
loss_function = torch.nn.CrossEntropyLoss(weight=weights_tensor,␣
 ↪ignore_index=-1).to(device)
# Optimizer
# optimizer = Adam(model.parameters(), lr=LEARNING_RATE)
optimizer = SGD(model.parameters(), lr=LEARNING_RATE)
# Move model to the device
model = model.to(device)
```

[ ]:
```
# Loss function
# loss_function = torch.nn.CrossEntropyLoss(ignore_index=-1).to(device)
#300 so far
EPOCHS = 50#100
for epoch in range(EPOCHS):
    model.train()
    total_loss = 0

    # Wrap your training loader with tqdm for progress bar
    for inputs, targets, lengths in train_loader: #tqdm(train_loader,␣
 ↪desc=f"Epoch {epoch + 1}/{EPOCHS}"):
        optimizer.zero_grad()

        # Fetch inputs and targets and move them to the current device
        inputs = inputs.to(device)
        targets = targets.to(device)
        lengths = lengths.to(device)

        # Forward pass
        outputs = model(inputs, lengths)

        # Compute loss and backpropagate
        loss = loss_function(outputs,targets) #(outputs.view(-1, tagset_size),␣
 ↪targets.view(-1))
        loss.backward()
        optimizer.step()

        total_loss += loss.item()
    # Validation
    model.eval()
    val_loss = 0
    with torch.no_grad():
        for inputs, targets, lengths in val_loader:
            inputs = inputs.to(device)
            targets = targets.to(device)
            lengths = lengths.to(device)

            outputs = model(inputs, lengths)
```

```
            loss = loss_function(outputs,targets) #(outputs.view(-1,␣
↪tagset_size), targets.view(-1))
            val_loss += loss.item()

    print(f"Epoch {epoch + 1}/{EPOCHS}, Training Loss: {total_loss /␣
↪len(train_loader)} Validation Loss: {val_loss / len(val_loader)}")
```

```
Epoch 1/50, Training Loss: 0.8141673816143109 Validation Loss:
0.46101467019202663
Epoch 2/50, Training Loss: 0.4019707834861477 Validation Loss:
0.38149524649020794
Epoch 3/50, Training Loss: 0.29123258200348917 Validation Loss:
0.3048748771897426
Epoch 4/50, Training Loss: 0.23722692021298178 Validation Loss:
0.2586067534033574
Epoch 5/50, Training Loss: 0.18413354061726386 Validation Loss:
0.2921877385244942
Epoch 6/50, Training Loss: 0.1530364083634278 Validation Loss:
0.2612298799258675
Epoch 7/50, Training Loss: 0.12721821191139385 Validation Loss:
0.267946543714988
Epoch 8/50, Training Loss: 0.10603927182943766 Validation Loss:
0.2779435542590149
Epoch 9/50, Training Loss: 0.09258149781106316 Validation Loss:
0.3695681261089311
Epoch 10/50, Training Loss: 0.07892728415593624 Validation Loss:
0.27078670984152337
Epoch 11/50, Training Loss: 0.06462572047042454 Validation Loss:
0.3118343375122005
Epoch 12/50, Training Loss: 0.05058781890124867 Validation Loss:
0.33446648336122925
Epoch 13/50, Training Loss: 0.04687063341786881 Validation Loss:
0.34744265180305256
Epoch 14/50, Training Loss: 0.04462885138413504 Validation Loss:
0.3374287801859692
Epoch 15/50, Training Loss: 0.03914838839500797 Validation Loss:
0.3360361746297124
Epoch 16/50, Training Loss: 0.04255056708458051 Validation Loss:
0.3500289000573556
Epoch 17/50, Training Loss: 0.0359404510317226 Validation Loss:
0.37614886597923175
Epoch 18/50, Training Loss: 0.031041965541221118 Validation Loss:
0.3702047535824682
Epoch 19/50, Training Loss: 0.02865660242067548 Validation Loss:
0.4041899945571405
Epoch 20/50, Training Loss: 0.026185777423116282 Validation Loss:
0.43221006305045606
```

```
Epoch 21/50, Training Loss: 0.020296966618690418 Validation Loss:
0.4745804030681564
Epoch 22/50, Training Loss: 0.028469396430846824 Validation Loss:
0.3870915525129012
Epoch 23/50, Training Loss: 0.019723668958767555 Validation Loss:
0.4122595866356991
Epoch 24/50, Training Loss: 0.01999404195104658 Validation Loss:
0.43159717892895594
Epoch 25/50, Training Loss: 0.015126698944503478 Validation Loss:
0.47038530968925263
Epoch 26/50, Training Loss: 0.013885648200870293 Validation Loss:
0.4845913674039522
Epoch 27/50, Training Loss: 0.023258937037658176 Validation Loss:
0.3996040560874712
Epoch 28/50, Training Loss: 0.02842222355540305 Validation Loss:
0.44186423833103866
Epoch 29/50, Training Loss: 0.02141465075057613 Validation Loss:
0.4581854166695529
Epoch 30/50, Training Loss: 0.018800236306552403 Validation Loss:
0.4494286697375655
Epoch 31/50, Training Loss: 0.026931648098797764 Validation Loss:
0.4473968314780762
Epoch 32/50, Training Loss: 0.014011399006965073 Validation Loss:
0.4508027569437891
Epoch 33/50, Training Loss: 0.02777551077592461 Validation Loss:
0.40967439293604446
Epoch 34/50, Training Loss: 0.01773252678398678 Validation Loss:
0.4547111651679877
Epoch 35/50, Training Loss: 0.016209725656501802 Validation Loss:
0.4406877006162785
Epoch 36/50, Training Loss: 0.009660987915795846 Validation Loss:
0.48731793246432903
Epoch 37/50, Training Loss: 0.012933923710199559 Validation Loss:
0.4917783123912692
Epoch 38/50, Training Loss: 0.011264054662558231 Validation Loss:
0.5075230035728461
Epoch 39/50, Training Loss: 0.010057648031365643 Validation Loss:
0.5474503829383255
Epoch 40/50, Training Loss: 0.009002702014070052 Validation Loss:
0.5137960492004172
Epoch 41/50, Training Loss: 0.009090074769285122 Validation Loss:
0.5007213809676241
Epoch 42/50, Training Loss: 0.008009474877618399 Validation Loss:
0.530473636971857
Epoch 43/50, Training Loss: 0.01098611742911839 Validation Loss:
0.5216467962269484
Epoch 44/50, Training Loss: 0.00958631974299351 Validation Loss:
0.5133806593931243
```

```
Epoch 45/50, Training Loss: 0.007986280597817627 Validation Loss:
0.5080674441700852
Epoch 46/50, Training Loss: 0.007965048375486537 Validation Loss:
0.5433329121643771
Epoch 47/50, Training Loss: 0.006377094976759755 Validation Loss:
0.5660444770146441
Epoch 48/50, Training Loss: 0.007802823373501201 Validation Loss:
0.552705097078289
Epoch 49/50, Training Loss: 0.006507759757714004 Validation Loss:
0.5455380512419252
Epoch 50/50, Training Loss: 0.006306102752050989 Validation Loss:
0.560062765672952
```

[ ]:
```python
def get_predictions(model, loader, device):
    model.eval()
    all_predictions = []
    with torch.no_grad():
        for inputs, _, lengths in loader: # We don't need targets now
            inputs = inputs.to(device)

            outputs = model(inputs, lengths)
            # Get predictions
            predictions = torch.argmax(outputs, dim=1)

            # Truncate predictions to their original lengths
            truncated_predictions = [pred[:len_].tolist() for pred, len_ in␣
 ↪zip(predictions, lengths)]

            all_predictions.extend(truncated_predictions)

    return all_predictions
```

[ ]:
```python
val_predictions = get_predictions(model, val_loader, device)
test_predictions = get_predictions(model, test_loader, device)
```

[ ]:
```python
from conlleval import evaluate
import itertools
# labels = ner_tags
# Map the labels back to their corresponding tag strings
idx2tag  = {id:tag for (tag,id) in id2label.items()}
labels = [
list(map(idx2tag.get, labels))
for labels in dataset['validation']['labels']
]
# This is the prediction by your model
preds = [
list(map(idx2tag.get, labels))
```

```
for labels in val_predictions
]
precision, recall, f1 = evaluate(itertools.chain(*labels),itertools.
  ↪chain(*preds))
```

```
processed 51362 tokens with 5942 phrases; found: 6170 phrases; correct: 5514.
accuracy:  93.78%; (non-O)
accuracy:  98.38%; precision:  89.37%; recall:  92.80%; FB1:  91.05
              LOC: precision:  93.48%; recall:  96.84%; FB1:  95.13  1903
             MISC: precision:  78.54%; recall:  86.12%; FB1:  82.15  1011
              ORG: precision:  85.95%; recall:  87.62%; FB1:  86.78  1367
              PER: precision:  93.49%; recall:  95.87%; FB1:  94.67  1889
```

[ ]:
```
from conlleval import evaluate
import itertools
# labels = ner_tags
# Map the labels back to their corresponding tag strings
idx2tag  = {id:tag for (tag,id) in id2label.items()}
labels = [
list(map(idx2tag.get, labels))
for labels in dataset['test']['labels']
]
# This is the prediction by your model
preds = [
list(map(idx2tag.get, labels))
for labels in test_predictions
]
precision, recall, f1 = evaluate(itertools.chain(*labels),itertools.
  ↪chain(*preds))
```

```
processed 46435 tokens with 5648 phrases; found: 6015 phrases; correct: 5010.
accuracy:  90.84%; (non-O)
accuracy:  97.26%; precision:  83.29%; recall:  88.70%; FB1:  85.91
              LOC: precision:  86.04%; recall:  93.82%; FB1:  89.76  1819
             MISC: precision:  63.13%; recall:  75.36%; FB1:  68.70  838
              ORG: precision:  81.09%; recall:  84.17%; FB1:  82.60  1724
              PER: precision:  92.90%; recall:  93.88%; FB1:  93.39  1634
```

[ ]:
```
# code to save pytorch model
# Move the model to CPU
model.to('cpu')

# Save the model's state_dict
torch.save(model.state_dict(), 'model_hw4_task2.pth')
```

[ ]:
```

```

# SN_NLP_HW4_task3

November 10, 2023

[1]:
```
!pip install datasets
```

```
Collecting datasets
  Downloading datasets-2.14.6-py3-none-any.whl (493 kB)
                             493.7/493.7
kB 9.0 MB/s eta 0:00:00
Requirement already satisfied: numpy>=1.17 in
/usr/local/lib/python3.10/dist-packages (from datasets) (1.23.5)
Requirement already satisfied: pyarrow>=8.0.0 in /usr/local/lib/python3.10/dist-
packages (from datasets) (9.0.0)
Collecting dill<0.3.8,>=0.3.0 (from datasets)
  Downloading dill-0.3.7-py3-none-any.whl (115 kB)
                             115.3/115.3
kB 17.6 MB/s eta 0:00:00
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-
packages (from datasets) (1.5.3)
Requirement already satisfied: requests>=2.19.0 in
/usr/local/lib/python3.10/dist-packages (from datasets) (2.31.0)
Requirement already satisfied: tqdm>=4.62.1 in /usr/local/lib/python3.10/dist-
packages (from datasets) (4.66.1)
Requirement already satisfied: xxhash in /usr/local/lib/python3.10/dist-packages
(from datasets) (3.4.1)
Collecting multiprocess (from datasets)
  Downloading multiprocess-0.70.15-py310-none-any.whl (134 kB)
                             134.8/134.8
kB 19.6 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 41.6 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: 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: 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)
Installing collected packages: dill, multiprocess, huggingface-hub, datasets
Successfully installed datasets-2.14.6 dill-0.3.7 huggingface-hub-0.19.0
multiprocess-0.70.15
```

[2]:
```
!wget https://raw.githubusercontent.com/sighsmile/conlleval/master/conlleval.py
```

```
--2023-11-09 23:59:07--
https://raw.githubusercontent.com/sighsmile/conlleval/master/conlleval.py
Resolving raw.githubusercontent.com (raw.githubusercontent.com)…
185.199.108.133, 185.199.109.133, 185.199.110.133, …
Connecting to raw.githubusercontent.com
(raw.githubusercontent.com)|185.199.108.133|:443… connected.
HTTP request sent, awaiting response… 200 OK
```

```
Length: 7502 (7.3K) [text/plain]
Saving to: ‘conlleval.py’

conlleval.py        100%[===================>]   7.33K  --.-KB/s    in 0s

2023-11-09 23:59:07 (87.3 MB/s) - ‘conlleval.py’ saved [7502/7502]
```

[3]:
```
import torch
import torch.nn as nn
from torch.utils.data import DataLoader
from torch.optim import Adam,SGD,AdamW
from torch.nn.functional import cross_entropy
from torch.nn.utils.rnn import pad_sequence
from tqdm import tqdm
from torch.nn.utils.rnn import pack_padded_sequence, pad_packed_sequence
import numpy as np
import pandas as pd
```

[4]:
```
import datasets

dataset = datasets.load_dataset("conll2003")
```

```
Downloading builder script:   0%|          | 0.00/9.57k [00:00<?, ?B/s]
Downloading metadata:   0%|          | 0.00/3.73k [00:00<?, ?B/s]
Downloading readme:   0%|          | 0.00/12.3k [00:00<?, ?B/s]
Downloading data:   0%|          | 0.00/983k [00:00<?, ?B/s]
Generating train split:   0%|          | 0/14041 [00:00<?, ? examples/s]
Generating validation split:   0%|          | 0/3250 [00:00<?, ? examples/s]
Generating test split:   0%|          | 0/3453 [00:00<?, ? examples/s]
```

[5]:
```
import itertools
from collections import Counter
# REFERENCE CODE PROVIDED BY SHOUMIK
word_frequency = Counter(itertools.chain(*dataset['train']['tokens']))  # type:␣
 ↪ignore

# Remove words below threshold 2
word2idx = {
    word: frequency
    for word, frequency in word_frequency.items()
    if frequency >= 2
}

word2idx = {
```

```
    word: index
    for index, word in enumerate(word_frequency.keys(), start=2)
}

word2idx['[PAD]'] = 0
word2idx['[UNK]'] = 1
```

[6]:
```
dataset = (
    dataset
    .map(lambda x: {
            'input_ids': [
                word2idx.get(word, word2idx['[UNK]'])
                for word in x['tokens']
            ]
        }
    )
)

dataset['train']['input_ids'][:3]
```

```
Map:   0%|          | 0/14041 [00:00<?, ? examples/s]
Map:   0%|          | 0/3250 [00:00<?, ? examples/s]
Map:   0%|          | 0/3453 [00:00<?, ? examples/s]
```

[6]:
```
[[2, 3, 4, 5, 6, 7, 8, 9, 10], [11, 12], [13, 14]]
```

[7]:
```
dataset['train']['tokens'][:3]
```

[7]:
```
[['EU', 'rejects', 'German', 'call', 'to', 'boycott', 'British', 'lamb', '.'],
 ['Peter', 'Blackburn'],
 ['BRUSSELS', '1996-08-22']]
```

[8]:
```
dataset
```

[8]:
```
DatasetDict({
    train: Dataset({
        features: ['id', 'tokens', 'pos_tags', 'chunk_tags', 'ner_tags',
'input_ids'],
        num_rows: 14041
    })
    validation: Dataset({
        features: ['id', 'tokens', 'pos_tags', 'chunk_tags', 'ner_tags',
'input_ids'],
        num_rows: 3250
    })
    test: Dataset({
        features: ['id', 'tokens', 'pos_tags', 'chunk_tags', 'ner_tags',
```

```
'input_ids'],
        num_rows: 3453
    })
})
```

[9]:
```
columns_to_remove = ['pos_tags', 'chunk_tags']
for split in dataset.keys():
    dataset[split] = dataset[split].remove_columns(columns_to_remove)

# Rename ner_tags to labels
for split in dataset.keys():
    dataset[split] = dataset[split].rename_column('ner_tags', 'labels')

print(dataset)
```

```
DatasetDict({
    train: Dataset({
        features: ['id', 'tokens', 'labels', 'input_ids'],
        num_rows: 14041
    })
    validation: Dataset({
        features: ['id', 'tokens', 'labels', 'input_ids'],
        num_rows: 3250
    })
    test: Dataset({
        features: ['id', 'tokens', 'labels', 'input_ids'],
        num_rows: 3453
    })
})
```

[10]:
```
label2id = dataset["train"].features["labels"].feature
id2label = {id: label for label, id in enumerate(label2id.names)}
id2label['PAD'] = -1
id2label
```

[10]:
```
{'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,
 'PAD': -1}
```

[ ]:

Task 3: Transformer model

[38]:
```
import math
class PositionalEncoding(nn.Module):
    def __init__(self, d_model, max_len=5000):
        super(PositionalEncoding, self).__init__()
        pe = torch.zeros(max_len, d_model)
        position = torch.arange(0, max_len, dtype=torch.float).unsqueeze(1)
        div_term = torch.exp(torch.arange(0, d_model, 2).float() * (-math.
 ↪log(10000.0) / d_model))
        pe[:, 0::2] = torch.sin(position * div_term)
        pe[:, 1::2] = torch.cos(position * div_term)
        pe = pe.unsqueeze(0).transpose(0, 1)
        self.register_buffer('pe', pe)

    def forward(self, x):
        x = x + self.pe[:x.size(0), :]
        return x

class TransformerNER(nn.Module):
    def __init__(self, vocab_size, tagset_size, emb_size=128, nhead=8,␣
 ↪ff_size=512, num_layers=6, dropout=0.1, max_seq_length=128):
        super(TransformerNER, self).__init__()
        self.token_embedding = nn.Embedding(vocab_size, emb_size)
        self.pos_encoder = PositionalEncoding(emb_size, max_seq_length)
        self.dropout = nn.Dropout(dropout)
        encoder_layers = nn.TransformerEncoderLayer(d_model=emb_size,␣
 ↪nhead=nhead, dim_feedforward=ff_size, dropout=dropout)
        self.transformer_encoder = nn.TransformerEncoder(encoder_layers,␣
 ↪num_layers=num_layers)
        self.classifier = nn.Linear(emb_size, tagset_size)
        self.emb_size = emb_size
    def forward(self, src, src_mask):
        src = self.token_embedding(src) * math.sqrt(self.emb_size)
        src = self.pos_encoder(src)
        src = self.dropout(src)
        output = self.transformer_encoder(src, src_key_padding_mask=src_mask)
        logits = self.classifier(output)
        return logits
```

[14]:
```
import torch
from torch.utils.data import DataLoader, TensorDataset
from torch.nn.utils.rnn import pad_sequence

def preprocess_data(data):
    input_ids = [torch.tensor(seq) for seq in data['input_ids']]
    labels = [torch.tensor(label) for label in data['labels']]
```

```
    return list(zip(input_ids, labels))

def dynamic_padding(batch):
    inputs = [item[0] for item in batch]
    labels = [item[1] for item in batch]

    # Dynamic padding in the batch
    inputs_padded = pad_sequence(inputs, batch_first=False)  # Transformer␣
 ↪expects (seq_len, batch, feature)
    labels_padded = pad_sequence(labels, batch_first=False, padding_value=-1) ␣
 ↪# padding as -1
    # Create the source mask for the transformer
    # `True` values are where the attention should NOT focus (i.e., padding)
    src_mask = (inputs_padded == 0).transpose(0, 1)

    return inputs_padded, labels_padded, src_mask


# Hyperparameters
BATCH_SIZE = 4#32
# Preprocess the train, val, and test data
train_data = preprocess_data(dataset['train'])
val_data = preprocess_data(dataset['validation'])
test_data = preprocess_data(dataset['test'])

train_loader = DataLoader(train_data, batch_size=BATCH_SIZE, shuffle=True,␣
 ↪collate_fn=dynamic_padding, num_workers=2)
val_loader = DataLoader(val_data, batch_size=BATCH_SIZE,␣
 ↪collate_fn=dynamic_padding, num_workers=2)
test_loader = DataLoader(test_data, batch_size=BATCH_SIZE,␣
 ↪collate_fn=dynamic_padding, num_workers=2)
```

[15]:
```
from collections import Counter

vocab_size = max([max(seq) for seq in dataset['train']['input_ids']]) + 1
tagset_size = max([max(seq) for seq in dataset['train']['labels']]) + 1
# Device definition
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
all_labels = [label for sublist in dataset['train']['labels'] for label in␣
 ↪sublist]
label_counts = {label: all_labels.count(label) for label in id2label.values()␣
 ↪if label != -1}
total_labels = len(all_labels) # We subtract the count of 'PAD' labels
weights = [total_labels / label_counts[id2label[key]] for key in id2label.
 ↪keys() if key != 'PAD']
weights = [weight / sum(weights) for weight in weights]
```

```
weights_tensor = torch.tensor(weights).to(device)
```

[16]:
```
label_counts
```

[16]:
```
{0: 169578,
 1: 6600,
 2: 4528,
 3: 6321,
 4: 3704,
 5: 7140,
 6: 1157,
 7: 3438,
 8: 1155}
```

[17]:
```
weights
```

[17]:
```
[0.0019872122378459785,
 0.05105870861658262,
 0.07442302934395877,
 0.05331236780089311,
 0.09097934040751764,
 0.04719712561196713,
 0.2912597034308084,
 0.09801846331281132,
 0.291764049237615]
```

[18]:
```
sample_batch = next(iter(train_loader))
len(sample_batch)
```

[18]:
```
3
```

[45]:
```
model = TransformerNER(vocab_size, tagset_size)
model
```

[45]:
```
TransformerNER(
  (token_embedding): Embedding(23625, 128)
  (pos_encoder): PositionalEncoding()
  (dropout): Dropout(p=0.1, inplace=False)
  (transformer_encoder): TransformerEncoder(
    (layers): ModuleList(
      (0-5): 6 x TransformerEncoderLayer(
        (self_attn): MultiheadAttention(
          (out_proj): NonDynamicallyQuantizableLinear(in_features=128,
out_features=128, bias=True)
        )
        (linear1): Linear(in_features=128, out_features=512, bias=True)
        (dropout): Dropout(p=0.1, inplace=False)
```

```
        (linear2): Linear(in_features=512, out_features=128, bias=True)
        (norm1): LayerNorm((128,), eps=1e-05, elementwise_affine=True)
        (norm2): LayerNorm((128,), eps=1e-05, elementwise_affine=True)
        (dropout1): Dropout(p=0.1, inplace=False)
        (dropout2): Dropout(p=0.1, inplace=False)
      )
    )
  )
  (classifier): Linear(in_features=128, out_features=9, bias=True)
)
```

[46]:
```
LEARNING_RATE = 1e-4 #75e-2 #3e-1 #75e-2#1.0 #5e-1#1e-1
loss_function = torch.nn.CrossEntropyLoss(weight=weights_tensor,␣
 ↪ignore_index=-1).to(device)
# Optimizer
optimizer = AdamW(model.parameters(), lr=LEARNING_RATE)
# optimizer = SGD(model.parameters(), lr=LEARNING_RATE)
# Move model to the device
model = model.to(device)
```

[26]:
```
for param_group in optimizer.param_groups:
        param_group['lr'] = 75e-5
```

[51]:
```
# # Loss function
# # loss_function = torch.nn.CrossEntropyLoss(ignore_index=-1).to(device)
# #120 so far
# EPOCHS = 20#100
# for epoch in range(EPOCHS):
#     model.train()
#     total_loss = 0

#     # Wrap your training loader with tqdm for progress bar
#     for inputs, targets, src_masks in train_loader: #tqdm(train_loader,␣
 ↪desc=f"Epoch {epoch + 1}/{EPOCHS}"):
#         optimizer.zero_grad()

#         # Fetch inputs and targets and move them to the current device
#         inputs = inputs.to(device)
#         targets = targets.to(device)
#         src_masks = src_masks.to(device)

#         # Forward pass
#         outputs = model(inputs, src_masks)

#         # Compute loss and backpropagate
#         loss = loss_function(outputs.view(-1, tagset_size), targets.view(-1))
#         loss.backward()
```

```
#         optimizer.step()

#         total_loss += loss.item()
#     # Validation
#     model.eval()
#     val_loss = 0
#     with torch.no_grad():
#         for inputs, targets, src_masks in val_loader:
#             inputs = inputs.to(device)
#             targets = targets.to(device)
#             src_masks = src_masks.to(device)

#             outputs = model(inputs, src_masks)

#             loss = loss_function(outputs.view(-1, tagset_size), targets.
 ↪view(-1))
#             val_loss += loss.item()

#     print(f"Epoch {epoch + 1}/{EPOCHS}, Training Loss: {total_loss /␣
 ↪len(train_loader)} Validation Loss: {val_loss / len(val_loader)}")
from torch.nn.utils import clip_grad_norm_
EPOCHS = 20
max_grad_norm = 1.0  # Gradient clipping to avoid exploding gradients
# best_val_loss = float('inf')  # Initialize best validation loss to infinity

for epoch in range(EPOCHS):
    model.train()
    total_loss = 0
    for inputs, targets, src_masks in train_loader:
        optimizer.zero_grad()
        inputs = inputs.to(device)
        targets = targets.to(device)
        src_masks = src_masks.to(device)

        # Forward pass
        outputs = model(inputs, src_masks)

        # Compute loss and backpropagate
        loss = loss_function(outputs.view(-1, tagset_size), targets.view(-1))
        loss.backward()

        # Clip gradients to prevent exploding gradient issues
        clip_grad_norm_(model.parameters(), max_grad_norm)

        optimizer.step()

        total_loss += loss.item()
```

```
    # Validation step
    model.eval()
    val_loss = 0
    with torch.no_grad():
        for inputs, targets, src_masks in val_loader:
            inputs = inputs.to(device)
            targets = targets.to(device)
            src_masks = src_masks.to(device)

            outputs = model(inputs, src_masks)

            loss = loss_function(outputs.view(-1, tagset_size), targets.
↪view(-1))
            val_loss += loss.item()

    avg_val_loss = val_loss / len(val_loader)

    # Save model if validation loss improved
    if avg_val_loss < best_val_loss:
        best_val_loss = avg_val_loss
        torch.save(model.state_dict(), 'best_model.pth')
        print(f"Saved best model at epoch {epoch+1}")

    print(f"Epoch {epoch + 1}/{EPOCHS}, Training Loss: {total_loss /␣
↪len(train_loader)}, Validation Loss: {avg_val_loss}")
```

```
Epoch 1/20, Training Loss: 0.25672982032419006, Validation Loss:
1.3076018528075748
Epoch 2/20, Training Loss: 0.2380272679884781, Validation Loss:
1.3175036643977136
Epoch 3/20, Training Loss: 0.21996602557405748, Validation Loss:
1.507679769863923
Epoch 4/20, Training Loss: 0.21389504836245954, Validation Loss:
1.3976437898990324
Epoch 5/20, Training Loss: 0.20527116575259186, Validation Loss:
1.625894182066957
Epoch 6/20, Training Loss: 0.18477508292132552, Validation Loss:
1.6557512455895216
Epoch 7/20, Training Loss: 0.18166712979951888, Validation Loss:
1.5729816806742916
Epoch 8/20, Training Loss: 0.17552787759987484, Validation Loss:
1.641558542251261
Epoch 9/20, Training Loss: 0.16980772537564712, Validation Loss:
1.570450601972129
Epoch 10/20, Training Loss: 0.17381892107437868, Validation Loss:
1.9158893123859742
Epoch 11/20, Training Loss: 0.15373687123350555, Validation Loss:
```

```
1.8380141960133847
Epoch 12/20, Training Loss: 0.1462394629470486, Validation Loss:
1.678482179925424
Epoch 13/20, Training Loss: 0.14419726657668736, Validation Loss:
2.0013715906194514
Epoch 14/20, Training Loss: 0.1361331452026204, Validation Loss:
2.096280108555531
Epoch 15/20, Training Loss: 0.1348539833754832, Validation Loss:
2.0400649859925712
Epoch 16/20, Training Loss: 0.12350502015039513, Validation Loss:
1.9163316486066426
Epoch 17/20, Training Loss: 0.12344464670080027, Validation Loss:
2.1319513438056195
Epoch 18/20, Training Loss: 0.11872022898458283, Validation Loss:
2.1043459351551155
Epoch 19/20, Training Loss: 0.11868443912102856, Validation Loss:
2.030022319200344
Epoch 20/20, Training Loss: 0.10864171985373432, Validation Loss:
2.1652188491868105
```

[34]:
```python
def get_predictions(model, loader, device):
    model.eval()
    all_predictions = []
    with torch.no_grad():
        for inputs, _, src_mask in loader: # We don't need targets, but we do␣
↪need the mask
            inputs = inputs.to(device)
            src_mask = src_mask.to(device)

            # Forward pass, get logits for each token in the sequence
            outputs = model(inputs, src_mask)

            # Get predictions
            predictions = torch.argmax(outputs, dim=2)  # dim=2 because outputs␣
↪are (seq_length, batch, num_tags)

            # Transpose predictions to match inputs shape
            predictions = predictions.transpose(0, 1)  # Now predictions are␣
↪(batch, seq_length)

            # Remove padding (convert masks to indices and select non-padded␣
↪elements)
            for batch_idx, batch in enumerate(predictions):
                # Get the indices where src_mask is False (meaning valid␣
↪tokens, not padding)
                valid_indices = ~src_mask[batch_idx]
                valid_predictions = batch[valid_indices]
```

```
                all_predictions.append(valid_predictions.tolist())

    return all_predictions
```

[52]:
```
val_predictions = get_predictions(model, val_loader, device)
test_predictions = get_predictions(model, test_loader, device)
```

[53]:
```
from conlleval import evaluate
import itertools
# labels = ner_tags
# Map the labels back to their corresponding tag strings
idx2tag  = {id:tag for (tag,id) in id2label.items()}
labels = [
list(map(idx2tag.get, labels))
for labels in dataset['validation']['labels']
]
# This is the prediction by your model
preds = [
list(map(idx2tag.get, labels))
for labels in val_predictions
]
precision, recall, f1 = evaluate(itertools.chain(*labels),itertools.
 ↪chain(*preds))
```

```
processed 51362 tokens with 5942 phrases; found: 5926 phrases; correct: 3983.
accuracy:  67.22%; (non-O)
accuracy:  92.97%; precision:  67.21%; recall:  67.03%; FB1:  67.12
              LOC: precision:  79.77%; recall:  78.77%; FB1:  79.27  1814
             MISC: precision:  74.73%; recall:  76.36%; FB1:  75.54  942
              ORG: precision:  54.21%; recall:  63.39%; FB1:  58.44  1568
              PER: precision:  61.30%; recall:  53.31%; FB1:  57.03  1602
```

[54]:
```
from conlleval import evaluate
import itertools
# labels = ner_tags
# Map the labels back to their corresponding tag strings
idx2tag  = {id:tag for (tag,id) in id2label.items()}
labels = [
list(map(idx2tag.get, labels))
for labels in dataset['test']['labels']
]
# This is the prediction by your model
preds = [
list(map(idx2tag.get, labels))
for labels in test_predictions
]
precision, recall, f1 = evaluate(itertools.chain(*labels),itertools.
 ↪chain(*preds))
```

```
processed 46435 tokens with 5648 phrases; found: 5249 phrases; correct: 3018.
accuracy:  54.23%; (non-O)
accuracy:  90.06%; precision:  57.50%; recall:  53.43%; FB1:  55.39
              LOC: precision:  74.43%; recall:  71.04%; FB1:  72.70  1592
             MISC: precision:  63.24%; recall:  65.67%; FB1:  64.43  729
              ORG: precision:  46.68%; recall:  51.17%; FB1:  48.82  1821
              PER: precision:  47.15%; recall:  32.28%; FB1:  38.33  1107
```

[ ]: