REPORT HW4 CSCI544

Task 1: Bidirectional LSTM model

a) What are the precision, recall, and F1 score on the validation data?

Ans) The precision, recall and F1 Score for the best model on the validation data in Task1 are : precision: 83.52%; recall: 78.21%; FB1: 80.78

b) What are the precision, recall, and F1 score on the test data?

Ans) The precision, recall and F1 Score for the best model on the test data in Task1 are : precision: 73.56%; recall: 67.53%; FB1: 70.41

Explanation of my solution :

The approach involves preprocessing the CoNLL 2003 dataset, defining a BiLSTM model architecture, and training the model using a DataLoader with customized collate functions. The training process incorporates early stopping to prevent overfitting.
Below is a breakdown of the solution, covering important points such as data preprocessing, model architecture, hyperparameters, and the training loop:

Data Preprocessing:

- Loading Dataset: The code uses the Hugging Face datasets library to load the CoNLL-2003 NER dataset.
- Word Frequency Filtering: The word frequency is calculated, and words occurring less than twice are removed.
- Word-to-Index Mapping: Words are mapped to indices, and special tokens like [PAD] and [UNK] are added.
- Token Indexing: The training dataset is preprocessed using the provided word-to-index mapping.
- Dataset Split: The dataset is split into training, testing, and validation sets.
- Labeling: The 'ner_tags' column is renamed to 'labels' for consistency.

Model Architecture:

- BiLSTM Model: The model is defined as a subclass of nn.Module with an embedding layer, a BiLSTM layer, a linear layer, and dropout.
- Embedding Layer: Converts word indices to dense vectors.
- BiLSTM Layer: Bidirectional LSTM layer processes input sequences.

- Linear Layer: Produces output scores for each tag.
- Dropout: Applied to the LSTM output for regularization.

Hyperparameters:
- vocab_size(Determined dynamically based on the unique words in the training set.): Vocabulary size, determined by the number of unique words in the training set.
- Tagset_size (9): Number of unique tags in the dataset.
- Embedding_dim (100): Dimensionality of word embeddings (100 for GloVe).
- Num_lstm_layers (1): Number of LSTM layers.
- Lstm_hidden_dim (256): Number of hidden units in each LSTM layer.
- Linear_output_dim (128): Dimensionality of the linear layer output.
- Learning_rate (0.01): Learning rate for the optimizer (Adam).
- Num_epochs (100): Maximum number of training epochs.
- Batch_size (32): Number of samples in each mini-batch.

Model Training:
- Loss Function (nn.CrossEntropyLoss): Cross-entropy loss is used for multi-class classification, ignoring the padding token.
- Optimizer (optim.Adam): Adam optimizer is used for parameter optimization.
- Early Stopping: Training includes early stopping based on the F1 score on the validation set.
- Model Saving: The best model is saved to a file ('task1_model.pt').

Evaluation:
- Evaluation Metrics: The F1 score is used as the evaluation metric.
- Test Set Evaluation: The model is evaluated on the test set after training.
- Prediction: Model predictions are converted back from indices to tag labels for analysis.
- The code uses the tqdm library for progress bars during evaluation.

GPU Usage:
- The code checks for GPU availability and moves the model to the GPU if available.

Task 2: Using GloVe word embeddings

a) What are the precision, recall, and F1 score on the validation data?

Ans) The precision, recall and F1 Score for the best model on the validation data in Task2 are : precision: 89.15%; recall: 88.05%; FB1: 88.60

b) What are the precision, recall, and F1 score on the test data?

Ans) The precision, recall and F1 Score for the best model on the test data in Task2 are : precision: precision: 82.32%; recall: 83.57%; FB1: 82.94

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

Ans) BiLSTM with Glove embeddings outperforms the non-pretrained model because Glove embeddings capture richer semantic information from large external corpora. Pretrained embeddings bring contextual understanding of words, aiding the model in recognizing intricate patterns and improving generalization. The pretrained vectors, learned from extensive text data, provide a meaningful initialization point for the embedding layer, allowing the model to converge faster and better represent the intricacies of language, leading to enhanced performance in tasks like Named Entity Recognition where contextual understanding is crucial.

Explanation of my solution :

The solution provides a word embedding approach using pre-trained GloVe embeddings and integrates them into a BiLSTM model for Named Entity Recognition. The model training and evaluation procedures, including early stopping, are well-defined given below:

Data Preprocessing:

- Word Embedding Initialization: The code initializes a dictionary (word_dict) to map words to indices and a list (embedding_matrix) to store GloVe word embeddings.
- GloVe Embedding Loading: The code reads a pre-trained GloVe file (glove.6B.100d.txt) and extracts word embeddings, updating the dictionary and embedding matrix accordingly.
- Special Token Handling: It inserts zero vectors and average vectors at the beginning of the embedding matrix.
- Word Dictionary Expansion: The code adds capitalized and uppercase forms of words to the dictionary with their corresponding vectors.
- Dataset Preprocessing: The code defines a function (preprocess_sample_glove) to convert tokens to GloVe indices, renames columns, and removes unnecessary columns. It then applies this function to the dataset.

Model Architecture:

- BiLSTM Model with GloVe Embeddings: The model is defined with an embedding layer initialized with GloVe embeddings, a BiLSTM layer, a linear layer, ELU activation, a classification layer, and dropout.

- Embedding Layer: Uses pre-trained GloVe embeddings with the option to freeze or fine-tune (freeze=False) during training.
- GloVe Embedding Type Conversion: Converts the GloVe embeddings to the expected data type for the LSTM layer.
- Dropout: Applied to the LSTM output for regularization.

Hyperparameters:
- glove_embedding_matrix: Initialized with pre-trained GloVe embeddings.
- tagset_size: Number of unique tags in the dataset (9).
- embedding_dim: Dimensionality of word embeddings (100 for GloVe).
- num_lstm_layers: Number of BiLSTM layers in the mode (1).
- lstm_hidden_dim: Number of hidden units in each LSTM layer (256).
- linear_output_dim: Dimensionality of the output after the linear layer (128).
- learning_rate: Learning rate for the optimizer (Adam) (0.001).
- num_epochs: Maximum number of training epochs (100).
- batch_size: Number of samples in each mini-batch (32).
- dropout: Probability of dropout in the Dropout layer (0.33).

Model Training:
- Loss Function (nn.CrossEntropyLoss): Cross-entropy loss is used for multi-class classification, ignoring the padding token.
- Optimizer (optim.Adam): Adam optimizer is used for parameter optimization.
- Early Stopping: Training includes early stopping based on the F1 score on the validation set.
- Model Saving: The best model is saved to a file ('task2_model.pt').

Evaluation:
- Evaluation Metrics: The F1 score is used as the evaluation metric.
- Test Set Evaluation: The model is evaluated on the test set after training.
- Prediction: Model predictions are converted back from indices to tag labels for analysis.
- The code uses the tqdm library for progress bars during evaluation.

GPU Usage:
- The code checks for GPU availability and moves the model to the GPU if available.

csci544hw4

November 11, 2023

[14]:
```
! wget https://raw.githubusercontent.com/sighsmile/conlleval/master/conlleval.py
```

```
--2023-11-11 02:08:20--
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.1'

conlleval.py.1      100%[===================>]   7.33K  --.-KB/s    in 0s

2023-11-11 02:08:20 (78.2 MB/s) - 'conlleval.py.1' saved [7502/7502]
```

[15]:
```
import torch
import torch.nn as nn
import torch.optim as optim
import numpy as np
import itertools
from collections import Counter
from torch.utils.data import DataLoader
from torch.nn.utils.rnn import pad_sequence
from torch.utils.data import TensorDataset
from conlleval import evaluate
from tqdm import tqdm
torch.manual_seed(1)
np.random.seed(1)
```

[16]:
```
import datasets

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

```
  0%|          | 0/3 [00:00<?, ?it/s]
```

[17]:
```
dataset
```

[17]:
```
DatasetDict({
    train: Dataset({
        features: ['id', 'tokens', 'pos_tags', 'chunk_tags', 'ner_tags'],
        num_rows: 14042
    })
    validation: Dataset({
        features: ['id', 'tokens', 'pos_tags', 'chunk_tags', 'ner_tags'],
        num_rows: 3251
    })
    test: Dataset({
        features: ['id', 'tokens', 'pos_tags', 'chunk_tags', 'ner_tags'],
        num_rows: 3454
    })
})
```

[18]:
```
word_freq = Counter(itertools.chain(*dataset['train']['tokens']))

word_freq = {
    word: frequency
    for word, frequency in word_freq.items()
    if frequency >= 2
}

w2ids = {
    word: index
    for index, word in enumerate(word_freq.keys(), start=2)
}

w2ids['[PAD]'] = 0
w2ids['[UNK]'] = 1

# Preprocess the dataset using the provided word2idx mapping
def preprocess_sample(sample):
    # Convert tokens to their respective indexes using w2ids
    input_ids = [w2ids.get(word, w2ids['[UNK]']) for word in sample['tokens']]

    # Update the sample with 'input_ids'
    sample['input_ids'] = input_ids

    # Remove 'pos tags' and 'chunk tags'
    sample.pop('pos_tags', None)
    sample.pop('chunk_tags', None)
    sample.pop('id', None)

    # Rename 'ner_tags' to 'labels'
    sample['labels'] = sample.pop('ner_tags')
```

```
    return sample

# Apply the preprocessing using .map() function
preprocessed_dataset = dataset.map(preprocess_sample)
```

```
0%|          | 0/14042 [00:00<?, ?ex/s]

0%|          | 0/3251 [00:00<?, ?ex/s]

0%|          | 0/3454 [00:00<?, ?ex/s]
```

[19]:

```
preprocessed_dataset
```

[19]:

```
DatasetDict({
    train: Dataset({
        features: ['tokens', 'input_ids', 'labels'],
        num_rows: 14042
    })
    validation: Dataset({
        features: ['tokens', 'input_ids', 'labels'],
        num_rows: 3251
    })
    test: Dataset({
        features: ['tokens', 'input_ids', 'labels'],
        num_rows: 3454
    })
})
```

[20]:

```
# Assuming you have a preprocessed train, test, and validation dataset
train_dataset = preprocessed_dataset['train']
test_dataset = preprocessed_dataset['test']
validation_dataset = preprocessed_dataset['validation']

# Define the special label for 'PAD'
PAD_LABEL = 9

# Create custom collate function for DataLoader
def custom_collate(batch):
    # Separate input_ids and labels
    input_ids = [torch.tensor(item['input_ids']) for item in batch]
    labels = [torch.tensor(item['labels']) for item in batch]
    input_id_orig = [len(terms) for terms in input_ids]

    # Pad input_ids and labels using pad_sequence
    input_ids = pad_sequence(input_ids, batch_first=True,␣
↪padding_value=w2ids['[PAD]'])
    labels = pad_sequence(labels, batch_first=True, padding_value=PAD_LABEL)
```

```
    return {'input_ids': input_ids, 'labels': labels, 'input_id_orig':␣
↪input_id_orig}

# Create DataLoader for train, test, and validation datasets
batch_size = 32  # You can adjust the batch size as needed
train_loader = DataLoader(train_dataset, batch_size=batch_size,␣
↪collate_fn=custom_collate, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=len(test_dataset),␣
↪collate_fn=custom_collate)
validation_loader = DataLoader(validation_dataset, batch_size=batch_size,␣
↪collate_fn=custom_collate)
```

[21]:
```
# Define the BiLSTM model
class BiLSTMModel(nn.Module):
    def __init__(self, vocab_size, embedding_dim, num_lstm_layers,␣
↪lstm_hidden_dim, linear_output_dim, tagset_size):
        super(BiLSTMModel, self).__init__()
        self.embedding = nn.Embedding(vocab_size, embedding_dim)
        self.bilstm = nn.LSTM(embedding_dim, lstm_hidden_dim,␣
↪num_layers=num_lstm_layers, bidirectional=True, batch_first=True)
        self.linear = nn.Linear(2 * lstm_hidden_dim, linear_output_dim)
        self.elu = nn.ELU()
        self.classifier = nn.Linear(linear_output_dim, tagset_size)
        self.dropout = nn.Dropout(p=0.33)  # Adjust the dropout rate as needed

    def forward(self, input_ids):
        embeddings = self.embedding(input_ids)
        lstm_out, _ = self.bilstm(embeddings)
        lstm_out = self.dropout(lstm_out)  # Apply dropout to the LSTM output
        linear_out = self.linear(lstm_out)
        elu_out = self.elu(linear_out)
        logits = self.classifier(elu_out)
        return logits

# Check for GPU availability
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# Define hyperparameters
vocab_size = len(w2ids)
print(vocab_size)
tagset_size = 9
embedding_dim = 100
num_lstm_layers = 1
lstm_hidden_dim = 256
linear_output_dim = 128
learning_rate = 0.01
num_epochs = 100  # You can adjust the number of epochs
```

```python
# Create BiLSTM model
model = BiLSTMModel(vocab_size, embedding_dim, num_lstm_layers,␣
 ↪lstm_hidden_dim, linear_output_dim, tagset_size)
model.to(device)

# Define loss function and optimizer
criterion = nn.CrossEntropyLoss(ignore_index=9)  # Ignore the pad token
optimizer = optim.Adam(model.parameters(), lr=learning_rate)

# Early stopping parameters
patience = 5  # Number of epochs with no improvement before stopping
best_validation_loss = float('inf')
counter = 0
best_validation_loss = 0

idx_to_tag = {0:'O', 1:'B-PER', 2:'I-PER', 3:'B-ORG', 4:'I-ORG', 5:'B-LOC', 6:
 ↪'I-LOC', 7:'B-MISC', 8:'I-MISC'}

def train(model, train_loader, optimizer, criterion, idx_to_tag):
    model.train()

    for batch in train_loader:
        input_ids, labels = batch['input_ids'].to(device, dtype=torch.long),␣
 ↪batch['labels'].to(device, dtype=torch.long)
        optimizer.zero_grad()
        logits = model(input_ids)
        loss = criterion(logits.view(-1, tagset_size), labels.view(-1))
        loss.backward()
        optimizer.step()

def eval_model(model, loader, idx_to_tag):
    model.eval()

    with torch.no_grad():
        preds = []
        real_labels = []
        for batch in tqdm(loader):
            val_input_ids, val_labels = batch['input_ids'].to(device,␣
 ↪dtype=torch.long), batch['labels'].to(device, dtype=torch.long)
            logits = model(val_input_ids)

            predictions = torch.argmax(logits, dim=-1).cpu().numpy().tolist()
            real_val_labels = val_labels.cpu().numpy().tolist()

            for temp in range(len(batch['input_id_orig'])):
                preds.append(predictions[temp][:batch['input_id_orig'][temp]])
```

```
                real_labels.append(real_val_labels[temp][:
↪batch['input_id_orig'][temp]])

    preds = list(itertools.chain(*preds))
    real_labels = list(itertools.chain(*real_labels))

    preds = [idx_to_tag[prediction] for prediction in preds]
    real_labels = [idx_to_tag[label] for label in real_labels]

    # Evaluate on validation data and print the results
    metrics = evaluate(real_labels, preds)

    return metrics

# Training loop
for epoch in range(num_epochs):
    train(model, train_loader, optimizer, criterion, idx_to_tag)

    print(f"Epoch {epoch+1}:")
    # Set the model to evaluation mode for validation
    val_loss = eval_model(model, validation_loader, idx_to_tag)

    # Early stopping check
    if val_loss[2] > best_validation_loss:
        best_validation_loss = val_loss[2]
        counter = 0
        # Save the model as .pt
        torch.save(model.state_dict(), 'task1_model.pt')
    else:
        counter += 1
        if counter >= patience:
            print(f'Early stopping at epoch {epoch+1}')
            print(f'Best F1 score: {best_validation_loss}')
            break
```

```
11984
Epoch 1:

100%|      | 102/102 [00:00<00:00, 150.08it/s]

processed 51362 tokens with 5942 phrases; found: 5459 phrases; correct: 4249.
accuracy:  74.11%; (non-O)
accuracy:  95.20%; precision:  77.83%; recall:  71.51%; FB1:  74.54
              LOC: precision:  86.46%; recall:  75.07%; FB1:  80.36  1595
             MISC: precision:  82.49%; recall:  70.50%; FB1:  76.02  788
              ORG: precision:  64.90%; recall:  63.83%; FB1:  64.36  1319
              PER: precision:  77.63%; recall:  74.05%; FB1:  75.80  1757
Epoch 2:
```

```
100%|      | 102/102 [00:00<00:00, 149.51it/s]

processed 51362 tokens with 5942 phrases; found: 5661 phrases; correct: 4561.
accuracy:  79.93%; (non-O)
accuracy:  95.87%; precision:  80.57%; recall:  76.76%; FB1:  78.62
              LOC: precision:  86.68%; recall:  85.36%; FB1:  86.01  1809
             MISC: precision:  86.06%; recall:  73.64%; FB1:  79.37  789
              ORG: precision:  71.97%; recall:  66.82%; FB1:  69.30  1245
              PER: precision:  78.00%; recall:  76.98%; FB1:  77.49  1818
Epoch 3:

100%|      | 102/102 [00:00<00:00, 144.93it/s]

processed 51362 tokens with 5942 phrases; found: 5116 phrases; correct: 4329.
accuracy:  75.74%; (non-O)
accuracy:  95.76%; precision:  84.62%; recall:  72.85%; FB1:  78.30
              LOC: precision:  89.49%; recall:  80.68%; FB1:  84.86  1656
             MISC: precision:  87.25%; recall:  76.46%; FB1:  81.50  808
              ORG: precision:  77.44%; recall:  65.77%; FB1:  71.13  1139
              PER: precision:  83.28%; recall:  68.40%; FB1:  75.11  1513
Epoch 4:

100%|      | 102/102 [00:00<00:00, 145.90it/s]

processed 51362 tokens with 5942 phrases; found: 5520 phrases; correct: 4517.
accuracy:  79.22%; (non-O)
accuracy:  95.92%; precision:  81.83%; recall:  76.02%; FB1:  78.82
              LOC: precision:  90.83%; recall:  80.84%; FB1:  85.54  1635
             MISC: precision:  73.96%; recall:  77.33%; FB1:  75.61  964
              ORG: precision:  71.36%; recall:  73.01%; FB1:  72.17  1372
              PER: precision:  86.51%; recall:  72.75%; FB1:  79.03  1549
Epoch 5:

100%|      | 102/102 [00:00<00:00, 143.87it/s]

processed 51362 tokens with 5942 phrases; found: 5415 phrases; correct: 4490.
accuracy:  78.09%; (non-O)
accuracy:  95.92%; precision:  82.92%; recall:  75.56%; FB1:  79.07
              LOC: precision:  86.35%; recall:  85.03%; FB1:  85.68  1809
             MISC: precision:  84.63%; recall:  72.23%; FB1:  77.94  787
              ORG: precision:  76.64%; recall:  70.69%; FB1:  73.55  1237
              PER: precision:  83.06%; recall:  71.34%; FB1:  76.75  1582
Epoch 6:

100%|      | 102/102 [00:00<00:00, 144.38it/s]

processed 51362 tokens with 5942 phrases; found: 5347 phrases; correct: 4455.
accuracy:  78.17%; (non-O)
accuracy:  95.92%; precision:  83.32%; recall:  74.97%; FB1:  78.93
              LOC: precision:  92.93%; recall:  77.95%; FB1:  84.78  1541
             MISC: precision:  82.29%; recall:  75.60%; FB1:  78.80  847
              ORG: precision:  70.49%; recall:  73.75%; FB1:  72.08  1403
```

```
             PER: precision:  85.93%; recall:  72.58%; FB1:  78.69  1556
Epoch 7:

100%|     | 102/102 [00:00<00:00, 142.77it/s]

processed 51362 tokens with 5942 phrases; found: 5777 phrases; correct: 4651.
accuracy:  81.83%; (non-O)
accuracy:  96.02%; precision:  80.51%; recall:  78.27%; FB1:  79.38
             LOC: precision:  87.77%; recall:  84.81%; FB1:  86.27  1775
            MISC: precision:  82.38%; recall:  76.57%; FB1:  79.37  857
             ORG: precision:  68.94%; recall:  71.51%; FB1:  70.20  1391
             PER: precision:  81.41%; recall:  77.52%; FB1:  79.42  1754
Epoch 8:

100%|     | 102/102 [00:00<00:00, 145.71it/s]

processed 51362 tokens with 5942 phrases; found: 5881 phrases; correct: 4675.
accuracy:  81.18%; (non-O)
accuracy:  96.05%; precision:  79.49%; recall:  78.68%; FB1:  79.08
             LOC: precision:  81.68%; recall:  88.08%; FB1:  84.76  1981
            MISC: precision:  78.88%; recall:  76.14%; FB1:  77.48  890
             ORG: precision:  73.79%; recall:  70.54%; FB1:  72.13  1282
             PER: precision:  81.54%; recall:  76.49%; FB1:  78.94  1728
Epoch 9:

100%|     | 102/102 [00:00<00:00, 146.59it/s]

processed 51362 tokens with 5942 phrases; found: 5821 phrases; correct: 4696.
accuracy:  81.03%; (non-O)
accuracy:  95.95%; precision:  80.67%; recall:  79.03%; FB1:  79.84
             LOC: precision:  87.14%; recall:  86.66%; FB1:  86.90  1827
            MISC: precision:  84.98%; recall:  77.33%; FB1:  80.98  839
             ORG: precision:  82.03%; recall:  67.04%; FB1:  73.78  1096
             PER: precision:  72.46%; recall:  81.00%; FB1:  76.49  2059
Epoch 10:

100%|     | 102/102 [00:00<00:00, 134.19it/s]

processed 51362 tokens with 5942 phrases; found: 5086 phrases; correct: 4376.
accuracy:  76.08%; (non-O)
accuracy:  95.77%; precision:  86.04%; recall:  73.65%; FB1:  79.36
             LOC: precision:  93.82%; recall:  80.13%; FB1:  86.44  1569
            MISC: precision:  85.71%; recall:  78.09%; FB1:  81.73  840
             ORG: precision:  78.60%; recall:  69.57%; FB1:  73.81  1187
             PER: precision:  83.96%; recall:  67.92%; FB1:  75.09  1490
Epoch 11:

100%|     | 102/102 [00:00<00:00, 146.24it/s]

processed 51362 tokens with 5942 phrases; found: 5605 phrases; correct: 4615.
accuracy:  80.59%; (non-O)
accuracy:  96.20%; precision:  82.34%; recall:  77.67%; FB1:  79.93
             LOC: precision:  89.59%; recall:  83.40%; FB1:  86.38  1710
```

```
            MISC: precision:  81.92%; recall:  76.68%; FB1:  79.22  863
             ORG: precision:  73.80%; recall:  69.95%; FB1:  71.82  1271
             PER: precision:  81.66%; recall:  78.07%; FB1:  79.82  1761
Epoch 12:

100%|     | 102/102 [00:00<00:00, 145.17it/s]

processed 51362 tokens with 5942 phrases; found: 5662 phrases; correct: 4595.
accuracy:  80.08%; (non-O)
accuracy:  96.02%; precision:  81.16%; recall:  77.33%; FB1:  79.20
             LOC: precision:  85.86%; recall:  83.61%; FB1:  84.72  1789
            MISC: precision:  88.69%; recall:  75.70%; FB1:  81.69  787
             ORG: precision:  66.01%; recall:  73.01%; FB1:  69.33  1483
             PER: precision:  86.21%; recall:  75.03%; FB1:  80.23  1603
Epoch 13:

100%|     | 102/102 [00:00<00:00, 148.67it/s]

processed 51362 tokens with 5942 phrases; found: 5173 phrases; correct: 4449.
accuracy:  77.07%; (non-O)
accuracy:  95.88%; precision:  86.00%; recall:  74.87%; FB1:  80.05
             LOC: precision:  89.98%; recall:  85.52%; FB1:  87.69  1746
            MISC: precision:  84.10%; recall:  76.90%; FB1:  80.34  843
             ORG: precision:  80.32%; recall:  64.21%; FB1:  71.36  1072
             PER: precision:  86.51%; recall:  71.01%; FB1:  78.00  1512
Epoch 14:

100%|     | 102/102 [00:00<00:00, 146.90it/s]

processed 51362 tokens with 5942 phrases; found: 5707 phrases; correct: 4537.
accuracy:  78.77%; (non-O)
accuracy:  95.85%; precision:  79.50%; recall:  76.35%; FB1:  77.90
             LOC: precision:  85.54%; recall:  86.01%; FB1:  85.78  1847
            MISC: precision:  86.57%; recall:  75.49%; FB1:  80.65  804
             ORG: precision:  64.19%; recall:  71.36%; FB1:  67.58  1491
             PER: precision:  83.32%; recall:  70.79%; FB1:  76.55  1565
Epoch 15:

100%|     | 102/102 [00:00<00:00, 147.21it/s]

processed 51362 tokens with 5942 phrases; found: 5692 phrases; correct: 4599.
accuracy:  80.17%; (non-O)
accuracy:  95.98%; precision:  80.80%; recall:  77.40%; FB1:  79.06
             LOC: precision:  81.34%; recall:  87.53%; FB1:  84.32  1977
            MISC: precision:  83.69%; recall:  75.70%; FB1:  79.50  834
             ORG: precision:  74.01%; recall:  68.16%; FB1:  70.96  1235
             PER: precision:  83.78%; recall:  74.86%; FB1:  79.07  1646
Epoch 16:

100%|     | 102/102 [00:00<00:00, 147.43it/s]

processed 51362 tokens with 5942 phrases; found: 5564 phrases; correct: 4647.
accuracy:  80.37%; (non-O)
```

```
accuracy:  96.15%; precision:  83.52%; recall:  78.21%; FB1:  80.78
              LOC: precision:  89.28%; recall:  85.68%; FB1:  87.44  1763
             MISC: precision:  87.12%; recall:  77.01%; FB1:  81.75  815
              ORG: precision:  74.26%; recall:  72.48%; FB1:  73.36  1309
              PER: precision:  82.95%; recall:  75.52%; FB1:  79.06  1677
Epoch 17:

100%|     | 102/102 [00:00<00:00, 143.49it/s]

processed 51362 tokens with 5942 phrases; found: 5534 phrases; correct: 4614.
accuracy:  80.43%; (non-O)
accuracy:  96.11%; precision:  83.38%; recall:  77.65%; FB1:  80.41
              LOC: precision:  91.81%; recall:  82.96%; FB1:  87.16  1660
             MISC: precision:  86.64%; recall:  76.68%; FB1:  81.36  816
              ORG: precision:  70.05%; recall:  74.12%; FB1:  72.03  1419
              PER: precision:  84.75%; recall:  75.41%; FB1:  79.80  1639
Epoch 18:

100%|     | 102/102 [00:00<00:00, 148.47it/s]

processed 51362 tokens with 5942 phrases; found: 5679 phrases; correct: 4525.
accuracy:  79.03%; (non-O)
accuracy:  95.86%; precision:  79.68%; recall:  76.15%; FB1:  77.88
              LOC: precision:  92.04%; recall:  79.26%; FB1:  85.17  1582
             MISC: precision:  84.28%; recall:  75.60%; FB1:  79.70  827
              ORG: precision:  61.13%; recall:  74.35%; FB1:  67.09  1631
              PER: precision:  83.89%; recall:  74.65%; FB1:  79.00  1639
Epoch 19:

100%|     | 102/102 [00:00<00:00, 141.30it/s]

processed 51362 tokens with 5942 phrases; found: 5586 phrases; correct: 4549.
accuracy:  79.91%; (non-O)
accuracy:  96.04%; precision:  81.44%; recall:  76.56%; FB1:  78.92
              LOC: precision:  88.91%; recall:  82.47%; FB1:  85.57  1704
             MISC: precision:  82.48%; recall:  76.57%; FB1:  79.42  856
              ORG: precision:  69.58%; recall:  70.62%; FB1:  70.10  1361
              PER: precision:  82.94%; recall:  74.97%; FB1:  78.76  1665
Epoch 20:

100%|     | 102/102 [00:00<00:00, 131.05it/s]

processed 51362 tokens with 5942 phrases; found: 5358 phrases; correct: 4524.
accuracy:  78.94%; (non-O)
accuracy:  96.07%; precision:  84.43%; recall:  76.14%; FB1:  80.07
              LOC: precision:  91.22%; recall:  83.18%; FB1:  87.02  1675
             MISC: precision:  84.32%; recall:  77.01%; FB1:  80.50  842
              ORG: precision:  75.22%; recall:  71.07%; FB1:  73.08  1267
              PER: precision:  84.69%; recall:  72.37%; FB1:  78.04  1574
Epoch 21:

100%|     | 102/102 [00:00<00:00, 146.61it/s]
```

```
processed 51362 tokens with 5942 phrases; found: 5364 phrases; correct: 4419.
accuracy:  77.36%; (non-O)
accuracy:  95.74%; precision:  82.38%; recall:  74.37%; FB1:  78.17
              LOC: precision:  91.72%; recall:  80.19%; FB1:  85.56  1606
             MISC: precision:  87.77%; recall:  74.73%; FB1:  80.73  785
              ORG: precision:  67.68%; recall:  73.38%; FB1:  70.41  1454
              PER: precision:  83.81%; recall:  69.11%; FB1:  75.75  1519
Early stopping at epoch 21
Best F1 score: 80.77524769685381
```

1 Validation Results Task 1

[22]:
```
# Load the state dictionary
model.load_state_dict(torch.load('task1_model.pt'))
model.eval()
# Move the model to the same device as the input data (cuda or cpu)
model.to(device)

with torch.no_grad():
    preds = []
    real_labels = []
    for batch in tqdm(validation_loader):
        val_input_ids, val_labels = batch['input_ids'].to(device, dtype=torch.
 ↪long), batch['labels'].to(device, dtype=torch.long)
        logits = model(val_input_ids)

        predictions = torch.argmax(logits, dim=-1).cpu().numpy().tolist()
        real_val_labels = val_labels.cpu().numpy().tolist()

        for temp in range(len(batch['input_id_orig'])):
            preds.append(predictions[temp][:batch['input_id_orig'][temp]])
            real_labels.append(real_val_labels[temp][:
 ↪batch['input_id_orig'][temp]])

preds = list(itertools.chain(*preds))
real_labels = list(itertools.chain(*real_labels))

preds = [idx_to_tag[prediction] for prediction in preds]
real_labels = [idx_to_tag[label] for label in real_labels]

# Evaluate on validation data and print the results
metrics = evaluate(real_labels, preds)
```

```
100%|      | 102/102 [00:00<00:00, 144.04it/s]
```

```
processed 51362 tokens with 5942 phrases; found: 5564 phrases; correct: 4647.
accuracy:  80.37%; (non-O)
```

```
accuracy:  96.15%; precision:  83.52%; recall:  78.21%; FB1:  80.78
              LOC: precision:  89.28%; recall:  85.68%; FB1:  87.44  1763
             MISC: precision:  87.12%; recall:  77.01%; FB1:  81.75  815
              ORG: precision:  74.26%; recall:  72.48%; FB1:  73.36  1309
              PER: precision:  82.95%; recall:  75.52%; FB1:  79.06  1677
```

2 Test Results Task 1

[23]:
```
# Load the state dictionary
model.load_state_dict(torch.load('task1_model.pt'))
model.eval()
# Move the model to the same device as the input data (cuda or cpu)
model.to(device)

with torch.no_grad():
    preds = []
    real_labels = []
    for batch in tqdm(test_loader):
        test_input_ids, test_labels = batch['input_ids'].to(device, dtype=torch.
 ↪long), batch['labels'].to(device, dtype=torch.long)
        logits = model(test_input_ids)

        predictions = torch.argmax(logits, dim=-1).cpu().numpy().tolist()
        real_val_labels = test_labels.cpu().numpy().tolist()

        for temp in range(len(batch['input_id_orig'])):
            preds.append(predictions[temp][:batch['input_id_orig'][temp]])
            real_labels.append(real_val_labels[temp][:
 ↪batch['input_id_orig'][temp]])

preds = list(itertools.chain(*preds))
real_labels = list(itertools.chain(*real_labels))

preds = [idx_to_tag[prediction] for prediction in preds]
real_labels = [idx_to_tag[label] for label in real_labels]

# Evaluate on validation data and print the results
metrics = evaluate(real_labels, preds)
```

```
100%|          | 1/1 [00:00<00:00,  1.05it/s]

processed 46435 tokens with 5648 phrases; found: 5185 phrases; correct: 3814.
accuracy:  72.40%; (non-O)
accuracy:  94.03%; precision:  73.56%; recall:  67.53%; FB1:  70.41
              LOC: precision:  81.21%; recall:  77.76%; FB1:  79.45  1597
             MISC: precision:  73.30%; recall:  64.53%; FB1:  68.64  618
              ORG: precision:  67.07%; recall:  63.52%; FB1:  65.24  1573
```

```
PER: precision:  72.23%; recall:  62.40%; FB1:  66.95  1397
```

[28]:
```
!wget http://nlp.stanford.edu/data/glove.6B.zip
```

```
--2023-11-11 02:15:05--  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]
--2023-11-11 02:15:05--  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]
--2023-11-11 02:15:06--  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.2’

glove.6B.zip.2      100%[===================>] 822.24M  5.01MB/s    in 2m 39s

2023-11-11 02:17:45 (5.17 MB/s) - ‘glove.6B.zip.2’ saved [862182613/862182613]
```

[31]:
```
!unzip glove.6B.zip
```

```
Archive:  glove.6B.zip.1
  inflating: glove.6B.50d.txt
  inflating: glove.6B.100d.txt
  inflating: glove.6B.200d.txt
  inflating: glove.6B.300d.txt
```

[32]:
```
# Initialize the word_dict dictionary
word_dict = {'[PAD]': 0, '[UNK]': 1}

# Initialize the embedding_matrix list
embedding_matrix = []

# Open and read the glove file
with open('glove.6B.100d.txt', 'r') as file:
    for line in file:
        # Split the line into words
```

```
        line_words = line.split()

        # Add the word to the dictionary and its corresponding embedding to the␣
 ↪list
        word_dict[line_words[0]] = len(word_dict)
        embedding_matrix.append([float(x) for x in line_words[1:]])

embedding_dimension = 100

# Insert zero vector at the beginning of embedding_matrix
embedding_matrix.insert(0, np.zeros(embedding_dimension))

# Insert the average vector at the beginning of embedding_matrix
embedding_matrix.insert(1, np.average(np.asarray(embedding_matrix), axis=0))

# Iterate through the keys in word_dict
for key in list(word_dict.keys()):
    # Check if the key is alphabetic
    if key.isalpha():
        # Check if the capitalized form is not in the dictionary
        if key.capitalize() not in word_dict.keys():
            # Add the capitalized form to the dictionary and its corresponding␣
 ↪vector to embedding_matrix
            word_dict[key.capitalize()] = len(word_dict)
            embedding_matrix.append(embedding_matrix[word_dict[key]])

        # Check if the uppercase form is not in the dictionary
        if key.upper() not in word_dict.keys():
            # Add the uppercase form to the dictionary and its corresponding␣
 ↪vector to embedding_matrix
            word_dict[key.upper()] = len(word_dict)
            embedding_matrix.append(embedding_matrix[word_dict[key]])

# Convert embedding_matrix to a NumPy array
embedding_matrix = np.asarray(embedding_matrix)
```

[33]:

```
# Preprocess the dataset using the provided word2idx mapping
def preprocess_sample_glove(sample):
    # Convert tokens to their respective indexes using w2ids
    glove_input_ids = [word_dict.get(word, word_dict['[UNK]']) for word in␣
 ↪sample['tokens']]

    # Update the sample with 'input_ids'
    sample['glove_input_ids'] = glove_input_ids

    # Remove 'pos tags' and 'chunk tags'
    sample.pop('pos_tags', None)
```

```
    sample.pop('chunk_tags', None)
    sample.pop('id', None)

    # Rename 'ner_tags' to 'labels'
    sample['labels'] = sample.pop('ner_tags')

    return sample

# Apply the preprocessing using .map() function
preprocessed_glove_dataset = dataset.map(preprocess_sample_glove)
```

```
0%|          | 0/14042 [00:00<?, ?ex/s]

0%|          | 0/3251 [00:00<?, ?ex/s]

0%|          | 0/3454 [00:00<?, ?ex/s]
```

[34]:
```
preprocessed_glove_dataset
```

[34]:
```
DatasetDict({
    train: Dataset({
        features: ['tokens', 'glove_input_ids', 'labels'],
        num_rows: 14042
    })
    validation: Dataset({
        features: ['tokens', 'glove_input_ids', 'labels'],
        num_rows: 3251
    })
    test: Dataset({
        features: ['tokens', 'glove_input_ids', 'labels'],
        num_rows: 3454
    })
})
```

[35]:
```
# Assuming you have a preprocessed train, test, and validation dataset
train_dataset = preprocessed_glove_dataset['train']
test_dataset = preprocessed_glove_dataset['test']
validation_dataset = preprocessed_glove_dataset['validation']

# Define the special label for 'PAD'
PAD_LABEL = 9

# Create custom collate function for DataLoader
def custom_collate(batch):
    # Separate input_ids and labels
    glove_input_ids = [torch.tensor(item['glove_input_ids']) for item in batch]
    labels = [torch.tensor(item['labels']) for item in batch]
    input_id_orig = [len(terms) for terms in glove_input_ids]
```

```
    # Pad input_ids and labels using pad_sequence
    glove_input_ids = pad_sequence(glove_input_ids, batch_first=True,␣
 ↪padding_value=word_dict['[PAD]'])
    labels = pad_sequence(labels, batch_first=True, padding_value=PAD_LABEL)

    return {'glove_input_ids': glove_input_ids, 'labels': labels,␣
 ↪'input_id_orig': input_id_orig}

# Create DataLoader for train, test, and validation datasets
batch_size = 32  # You can adjust the batch size as needed
train_loader = DataLoader(train_dataset, batch_size=batch_size,␣
 ↪collate_fn=custom_collate, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=len(test_dataset),␣
 ↪collate_fn=custom_collate)
validation_loader = DataLoader(validation_dataset, batch_size=batch_size,␣
 ↪collate_fn=custom_collate)
```

[36]:
```
# Define the BiLSTM model
class BiLSTMModel(nn.Module):
    def __init__(self, glove_embedding_matrix, embedding_dim, num_lstm_layers,␣
 ↪lstm_hidden_dim, linear_output_dim, tagset_size):
        super(BiLSTMModel, self).__init__()
        self.embedding = nn.Embedding.from_pretrained(torch.
 ↪from_numpy(glove_embedding_matrix), freeze=False)
        self.bilstm = nn.LSTM(embedding_dim, lstm_hidden_dim,␣
 ↪num_layers=num_lstm_layers, bidirectional=True, batch_first=True)
        self.linear = nn.Linear(2 * lstm_hidden_dim, linear_output_dim)
        self.elu = nn.ELU()
        self.classifier = nn.Linear(linear_output_dim, tagset_size)
        self.dropout = nn.Dropout(p=0.33)  # Adjust the dropout rate as needed

    def forward(self, glove_input_ids):
        embeddings = self.embedding(glove_input_ids)
        # Ensure the data type of the embeddings matches the expected data type␣
 ↪for the LSTM layer
        embeddings = embeddings.to(torch.float32)  # Change torch.float32 to␣
 ↪the correct data type

        lstm_out, _ = self.bilstm(embeddings)
        lstm_out = self.dropout(lstm_out)  # Apply dropout to the LSTM output
        linear_out = self.linear(lstm_out)
        elu_out = self.elu(linear_out)
        logits = self.classifier(elu_out)
        return logits
```

```
# Check for GPU availability
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# Define hyperparameters
glove_embedding_matrix = embedding_matrix
tagset_size = 9
embedding_dim = 100
num_lstm_layers = 1
lstm_hidden_dim = 256
linear_output_dim = 128
learning_rate = 0.001
num_epochs = 100  # You can adjust the number of epochs

# Create BiLSTM model
model = BiLSTMModel(glove_embedding_matrix, embedding_dim, num_lstm_layers,␣
↪lstm_hidden_dim, linear_output_dim, tagset_size)
model.to(device)

# Define loss function and optimizer
criterion = nn.CrossEntropyLoss(ignore_index=9)  # Ignore the pad token
optimizer = optim.Adam(model.parameters(), lr=learning_rate)

# Early stopping parameters
patience = 5  # Number of epochs with no improvement before stopping
best_validation_loss = float('inf')
counter = 0
best_validation_loss = 0

idx_to_tag = {0:'O', 1:'B-PER', 2:'I-PER', 3:'B-ORG', 4:'I-ORG', 5:'B-LOC', 6:
↪'I-LOC', 7:'B-MISC', 8:'I-MISC'}

def train(model, train_loader, optimizer, criterion, idx_to_tag):
    model.train()

    for batch in train_loader:
        glove_input_ids, labels = batch['glove_input_ids'].to(device,␣
↪dtype=torch.long), batch['labels'].to(device, dtype=torch.long)
        optimizer.zero_grad()
        logits = model(glove_input_ids)
        loss = criterion(logits.view(-1, tagset_size), labels.view(-1))
        loss.backward()
        optimizer.step()

def eval_model(model, loader, idx_to_tag):
    model.eval()

    with torch.no_grad():
```

```
        preds = []
        real_labels = []
        for batch in tqdm(loader):
            glove_val_input_ids, val_labels = batch['glove_input_ids'].
↪to(device, dtype=torch.long), batch['labels'].to(device, dtype=torch.long)
            logits = model(glove_val_input_ids)

            predictions = torch.argmax(logits, dim=-1).cpu().numpy().tolist()
            real_val_labels = val_labels.cpu().numpy().tolist()

            for temp in range(len(batch['input_id_orig'])):
                preds.append(predictions[temp][:batch['input_id_orig'][temp]])
                real_labels.append(real_val_labels[temp][:
↪batch['input_id_orig'][temp]])

    preds = list(itertools.chain(*preds))
    real_labels = list(itertools.chain(*real_labels))

    preds = [idx_to_tag[prediction] for prediction in preds]
    real_labels = [idx_to_tag[label] for label in real_labels]

    # Evaluate on validation data and print the results
    metrics = evaluate(real_labels, preds)

    return metrics

# Training loop
for epoch in range(num_epochs):
    train(model, train_loader, optimizer, criterion, idx_to_tag)

    print(f"Epoch {epoch+1}:")
    # Set the model to evaluation mode for validation
    val_loss = eval_model(model, validation_loader, idx_to_tag)

    # Early stopping check
    if val_loss[2] > best_validation_loss:
        best_validation_loss = val_loss[2]
        counter = 0
        # Save the model as .pt
        torch.save(model.state_dict(), 'task2_model.pt')
    else:
        counter += 1
        if counter >= patience:
            print(f'Early stopping at epoch {epoch+1}')
            print(f'Best F1 score: {best_validation_loss}')
            break
```

```
Epoch 1:

100%|     | 102/102 [00:00<00:00, 126.41it/s]

processed 51362 tokens with 5942 phrases; found: 5703 phrases; correct: 4794.
accuracy:  81.83%; (non-O)
accuracy:  96.63%; precision:  84.06%; recall:  80.68%; FB1:  82.34
              LOC: precision:  87.57%; recall:  85.52%; FB1:  86.53  1794
             MISC: precision:  82.30%; recall:  67.57%; FB1:  74.21  757
              ORG: precision:  72.30%; recall:  73.97%; FB1:  73.13  1372
              PER: precision:  90.34%; recall:  87.30%; FB1:  88.79  1780
Epoch 2:

100%|     | 102/102 [00:00<00:00, 127.53it/s]

processed 51362 tokens with 5942 phrases; found: 6003 phrases; correct: 5165.
accuracy:  88.00%; (non-O)
accuracy:  97.52%; precision:  86.04%; recall:  86.92%; FB1:  86.48
              LOC: precision:  90.04%; recall:  90.58%; FB1:  90.31  1848
             MISC: precision:  75.15%; recall:  79.07%; FB1:  77.06  970
              ORG: precision:  79.79%; recall:  80.69%; FB1:  80.24  1356
              PER: precision:  92.40%; recall:  91.75%; FB1:  92.07  1829
Epoch 3:

100%|     | 102/102 [00:00<00:00, 126.92it/s]

processed 51362 tokens with 5942 phrases; found: 6049 phrases; correct: 5257.
accuracy:  89.93%; (non-O)
accuracy:  97.75%; precision:  86.91%; recall:  88.47%; FB1:  87.68
              LOC: precision:  92.16%; recall:  90.91%; FB1:  91.53  1812
             MISC: precision:  75.45%; recall:  81.02%; FB1:  78.14  990
              ORG: precision:  81.30%; recall:  83.67%; FB1:  82.47  1380
              PER: precision:  92.02%; recall:  93.27%; FB1:  92.64  1867
Epoch 4:

100%|     | 102/102 [00:00<00:00, 128.32it/s]

processed 51362 tokens with 5942 phrases; found: 5869 phrases; correct: 5232.
accuracy:  88.69%; (non-O)
accuracy:  97.81%; precision:  89.15%; recall:  88.05%; FB1:  88.60
              LOC: precision:  94.17%; recall:  89.66%; FB1:  91.86  1749
             MISC: precision:  83.22%; recall:  81.24%; FB1:  82.22  900
              ORG: precision:  81.43%; recall:  83.37%; FB1:  82.39  1373
              PER: precision:  93.02%; recall:  93.27%; FB1:  93.14  1847
Epoch 5:

100%|     | 102/102 [00:00<00:00, 123.39it/s]

processed 51362 tokens with 5942 phrases; found: 5845 phrases; correct: 5204.
accuracy:  88.49%; (non-O)
accuracy:  97.77%; precision:  89.03%; recall:  87.58%; FB1:  88.30
              LOC: precision:  92.75%; recall:  90.53%; FB1:  91.63  1793
             MISC: precision:  81.40%; recall:  82.10%; FB1:  81.75  930
```

```
              ORG: precision:  83.04%; recall:  83.22%; FB1:  83.13  1344
              PER: precision:  93.81%; recall:  90.55%; FB1:  92.15  1778
Epoch 6:

100%|     | 102/102 [00:00<00:00, 129.83it/s]

processed 51362 tokens with 5942 phrases; found: 5918 phrases; correct: 5248.
accuracy:  88.95%; (non-O)
accuracy:  97.81%; precision:  88.68%; recall:  88.32%; FB1:  88.50
              LOC: precision:  93.06%; recall:  90.58%; FB1:  91.81  1788
             MISC: precision:  79.39%; recall:  82.32%; FB1:  80.83  956
              ORG: precision:  83.86%; recall:  83.30%; FB1:  83.58  1332
              PER: precision:  92.73%; recall:  92.73%; FB1:  92.73  1842
Epoch 7:

100%|     | 102/102 [00:00<00:00, 124.84it/s]

processed 51362 tokens with 5942 phrases; found: 5965 phrases; correct: 5264.
accuracy:  89.74%; (non-O)
accuracy:  97.85%; precision:  88.25%; recall:  88.59%; FB1:  88.42
              LOC: precision:  94.31%; recall:  90.20%; FB1:  92.21  1757
             MISC: precision:  84.12%; recall:  81.56%; FB1:  82.82  894
              ORG: precision:  79.10%; recall:  85.23%; FB1:  82.05  1445
              PER: precision:  91.60%; recall:  92.94%; FB1:  92.27  1869
Epoch 8:

100%|     | 102/102 [00:00<00:00, 127.33it/s]

processed 51362 tokens with 5942 phrases; found: 6039 phrases; correct: 5277.
accuracy:  89.74%; (non-O)
accuracy:  97.77%; precision:  87.38%; recall:  88.81%; FB1:  88.09
              LOC: precision:  91.79%; recall:  91.34%; FB1:  91.57  1828
             MISC: precision:  81.28%; recall:  81.02%; FB1:  81.15  919
              ORG: precision:  79.89%; recall:  84.41%; FB1:  82.09  1417
              PER: precision:  91.73%; recall:  93.38%; FB1:  92.55  1875
Epoch 9:

100%|     | 102/102 [00:00<00:00, 130.07it/s]

processed 51362 tokens with 5942 phrases; found: 6045 phrases; correct: 5277.
accuracy:  89.50%; (non-O)
accuracy:  97.76%; precision:  87.30%; recall:  88.81%; FB1:  88.05
              LOC: precision:  92.37%; recall:  89.60%; FB1:  90.96  1782
             MISC: precision:  79.33%; recall:  82.00%; FB1:  80.64  953
              ORG: precision:  80.90%; recall:  85.61%; FB1:  83.19  1419
              PER: precision:  91.33%; recall:  93.76%; FB1:  92.53  1891
Early stopping at epoch 9
Best F1 score: 88.59537719075439
```

3 Validation Results Task 2

[37]:
```python
# Load the state dictionary
model.load_state_dict(torch.load('task2_model.pt'))
model.eval()
# Move the model to the same device as the input data (cuda or cpu)
model.to(device)

with torch.no_grad():
    preds = []
    real_labels = []
    for batch in tqdm(validation_loader):
        val_glove_input_ids, val_labels = batch['glove_input_ids'].to(device,␣
↪dtype=torch.long), batch['labels'].to(device, dtype=torch.long)
        logits = model(val_glove_input_ids)

        predictions = torch.argmax(logits, dim=-1).cpu().numpy().tolist()
        real_val_labels = val_labels.cpu().numpy().tolist()

        for temp in range(len(batch['input_id_orig'])):
            preds.append(predictions[temp][:batch['input_id_orig'][temp]])
            real_labels.append(real_val_labels[temp][:
↪batch['input_id_orig'][temp]])

preds = list(itertools.chain(*preds))
real_labels = list(itertools.chain(*real_labels))

preds = [idx_to_tag[prediction] for prediction in preds]
real_labels = [idx_to_tag[label] for label in real_labels]

# Evaluate on validation data and print the results
metrics = evaluate(real_labels, preds)
```

```
100%|          | 102/102 [00:00<00:00, 142.27it/s]

processed 51362 tokens with 5942 phrases; found: 5869 phrases; correct: 5232.
accuracy:  88.69%; (non-O)
accuracy:  97.81%; precision:  89.15%; recall:  88.05%; FB1:  88.60
              LOC: precision:  94.17%; recall:  89.66%; FB1:  91.86  1749
             MISC: precision:  83.22%; recall:  81.24%; FB1:  82.22  900
              ORG: precision:  81.43%; recall:  83.37%; FB1:  82.39  1373
              PER: precision:  93.02%; recall:  93.27%; FB1:  93.14  1847
```

4 Test Results Task 2

[38]:
```
# Load the state dictionary
model.load_state_dict(torch.load('task2_model.pt'))
model.eval()
# Move the model to the same device as the input data (cuda or cpu)
model.to(device)

with torch.no_grad():
    preds = []
    real_labels = []
    for batch in tqdm(test_loader):
        test_glove_input_ids, test_labels = batch['glove_input_ids'].to(device,␣
 ↪dtype=torch.long), batch['labels'].to(device, dtype=torch.long)
        logits = model(test_glove_input_ids)

        predictions = torch.argmax(logits, dim=-1).cpu().numpy().tolist()
        real_val_labels = test_labels.cpu().numpy().tolist()

        for temp in range(len(batch['input_id_orig'])):
            preds.append(predictions[temp][:batch['input_id_orig'][temp]])
            real_labels.append(real_val_labels[temp][:
 ↪batch['input_id_orig'][temp]])

preds = list(itertools.chain(*preds))
real_labels = list(itertools.chain(*real_labels))

preds = [idx_to_tag[prediction] for prediction in preds]
real_labels = [idx_to_tag[label] for label in real_labels]

# Evaluate on validation data and print the results
metrics = evaluate(real_labels, preds)
```

```
100%|     | 1/1 [00:00<00:00,  1.26it/s]

processed 46435 tokens with 5648 phrases; found: 5734 phrases; correct: 4720.
accuracy:  85.92%; (non-O)
accuracy:  96.64%; precision:  82.32%; recall:  83.57%; FB1:  82.94
              LOC: precision:  88.55%; recall:  88.07%; FB1:  88.31  1659
             MISC: precision:  69.69%; recall:  73.36%; FB1:  71.48  739
              ORG: precision:  75.07%; recall:  78.87%; FB1:  76.92  1745
              PER: precision:  89.63%; recall:  88.19%; FB1:  88.90  1591
```