hindi-ner

June 23, 2024

1 Hindi NER

NER(Named Entity Recognition) is a token classification task, Where we classify the given token into several given categories. Here I have used 2-Layer transformer architecture to generate the contextual embeddings the we simply apply softmax for multiclass classification.

1.1 Import Required libraries

[2]:
```
from datasets import load_dataset
from tokenizers import␣
 ↪decoders,models,pre_tokenizers,processors,trainers,Tokenizer,normalizers
from torch.utils.data import Dataset,DataLoader
import torch
import torch.nn as nn
import torch.nn.functional as F
```

1.2 Load dataset

Here we are using cfilt/HiNER-collapsed. It was released by CFILT lab IIT Bombay in 2022. It has text and their NER tags as following - 0-B-loc - 3-I-loc - 2-B-per
- 5-I-per - 1-B-org
- 4-I-org - 6-O
the above given ordering is correct and the ordering given in the dataset library is incorrect. This dataset has 3 splits. - train-75.8k training exaples - test-21.7k training examples - validation -10.9k training examples
we also have full dataset cfilt/HiNER. It has 23 categories.

[3]:
```
ds = load_dataset("cfilt/HiNER-collapsed")
```

```
Downloading data:   0%|          | 0.00/7.10M [00:00<?, ?B/s]

Downloading data:   0%|          | 0.00/1.02M [00:00<?, ?B/s]

Downloading data:   0%|          | 0.00/2.02M [00:00<?, ?B/s]

Generating train split:   0%|          | 0/75827 [00:00<?, ? examples/s]

Generating validation split:   0%|          | 0/10851 [00:00<?, ? examples/s]

Generating test split:   0%|          | 0/21657 [00:00<?, ? examples/s]
```

[4]:
```
print(ds)
```

```
DatasetDict({
    train: Dataset({
        features: ['id', 'tokens', 'ner_tags'],
        num_rows: 75827
    })
    validation: Dataset({
        features: ['id', 'tokens', 'ner_tags'],
        num_rows: 10851
    })
    test: Dataset({
        features: ['id', 'tokens', 'ner_tags'],
        num_rows: 21657
    })
})
```

[5]:
```
train=ds['train'].data.to_pandas()
```

[6]:
```
print (train)
```

```
          id                                                tokens  \
0          0  [ ,   ,  ,  ,   ,  ,   ,  ,…
1          1  [ ,  ,  ,  ,   ,  ,  ,   ,  …
2          2  [   ,   , ,,   , ,,   ,   ,  …
3          3  [   ,    ,   ,   - ,  ,  …
4          4  [ ,  ,  ,  ,   ,  ,  , …
…        …                                                   …
75822  75822                                [   ,  ,  , .]
75823  75823  [    ,  ,   ,  ,  ,   ,  , …
75824  75824  [   ,,   ,   ,  ,  ,  ,  …
75825  75825  [   , (,   :, Life), 1976,  ,  …
75826  75826  [ ,  ,  ,  ,  ,  ,  ,  ,  ,…

                                               ner_tags
0      [6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, …
1                      [6, 6, 6, 0, 3, 6, 6, 6, 6, 6, 6]
2                      [0, 0, 6, 0, 6, 0, 3, 6, 6, 6, 6]
3                 [6, 6, 6, 6, 6, 6, 6, 0, 6, 6, 6, 6, 6]
4           [6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6]
…                                                     …
75822                                      [6, 6, 6, 6]
75823  [0, 6, 6, 6, 6, 6, 6, 6, 6, 2, 6, 2, 6, 6, 6, …
75824  [0, 0, 6, 6, 0, 6, 0, 6, 6, 6, 0, 6, 6, 0, 6, …
75825                [6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6]
75826  [6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, …

[75827 rows x 3 columns]
```

[7]:
```
train.head()
```

[7]:

	id	tokens	ner_tags
0	0	[, , , , , , , ,…	[6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, …
1	1	[, , , , , , , , …	[6, 6, 6, 0, 3, 6, 6, 6, 6, 6, 6]
2	2	[, , ,, , ,, , , …	[0, 0, 6, 0, 6, 0, 3, 6, 6, 6, 6]
3	3	[, , , - , , …	[6, 6, 6, 6, 6, 6, 6, 0, 6, 6, 6, 6, 6]
4	4	[, , , , , , , …	[6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6]

[8]:
```
features=ds['train'].features['ner_tags'].feature.names
```

[9]:
```
len(features)
```

[9]: 7

[10]:
```
print(features)
```

```
['O', 'B-PER', 'I-PER', 'B-LOC', 'I-LOC', 'B-ORG', 'I-ORG']
```

[11]:
```
validation=ds['validation'].data.to_pandas()
validation.head()
```

[11]:

	id	tokens	ner_tags
0	0	[, , , , , , …	[1, 4, 6, 6, 6, 2, 5, 6, 1, 6, 6, 6, 6, 6, 6, …
1	1	[, N.Z.A., ,, , , , , …	[0, 3, 6, 0, 6, 6, 0, 6, 0, 6, 6, 6, 0, 6, 6, …
2	2	[, , , , , , , , …	[6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, …
3	3	[- - ,, , , , , …	[0, 0, 6, 6, 0, 6, 0, 6, 6, 6, 0, 6, 6, 0, 6, …
4	4	[, , , , , , , , ,…	[1, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, …

[12]:
```
test=ds['test'].data.to_pandas()
test.head()
```

[12]:
```
  id                                                tokens  \
0  0                        [   ,  :,   -  , .]
1  1       [  ,     ,  ,   ,   ,   ,   , .]
2  2  [   ,    ,  ,    ,   ,   ,  , …
3  3  [ ,    ,   ,    ,   ,      , …
4  4                                  [   ,  , .]

                                            ner_tags
0                                        [6, 6, 6, 6]
1                              [6, 2, 6, 6, 6, 6, 6, 6]
2  [0, 0, 6, 0, 3, 6, 6, 0, 6, 6, 0, 6, 6, 6, 6, …
3  [6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, …
4                                           [6, 6, 6]
```

[13]:
```
text=""
for txt in train['tokens']:
    text+=" ".join(txt)+"\n"
```

[14]:
```
for txt in validation['tokens']:
    text+=" ".join(txt)+"\n"
```

[15]:
```
for txt in test['tokens']:
    text+=" ".join(txt)+"\n"
```

[16]:
```
len(text)
```

[16]:
```
11077784
```

[17]:
```
with open("./data.txt", "w",encoding='utf-8') as f:
    f.write(text)
```

1.3 Tokenizer

Here we are using Byte-Pair-Encoding Tokenizer for tokeninzing our text. I have used tokenizers library to make the tokenizer with vocab size 30000.

[19]:
```
bpe_tokenizer=Tokenizer(model=models.BPE(unk_token="[UNK]"))
```

[20]:
```
bpe_tokenizer.normalizer=normalizers.NFC()
```

[21]:
```
bpe_tokenizer.pre_tokenizer=pre_tokenizers.Sequence([pre_tokenizers.
↪Whitespace(),pre_tokenizers.Punctuation()])
```

[22]:
```
bpe_trainer=trainers.
↪BpeTrainer(vocab_size=30000,min_frequency=2,special_tokens=["[UNK]","[PAD]","[CLS]","[SEP]"
```

[23]:
```
bpe_tokenizer.train(files=['./data.txt'],trainer=bpe_trainer)
```

[24]:
```
bpe_tokenizer.save("bpe_tokenizer.json")
```

[25]:
```
bpe_tokenizer.encode("     :   -  ").ids
```

[25]:
```
[3684, 7762, 26, 4413, 13, 4413]
```

[26]:
```
bpe_tokenizer.encode(["   ", " :", "  -  ", "."],is_pretokenized=True).tokens
```

[26]:
```
['   ', ' ', ':', '  ', '-', '  ', '.']
```

[27]:
```
bpe_tokenizer.encode(["   ", " :", "  -  ", "."],is_pretokenized=True).word_ids
```

[27]:
```
[0, 1, 1, 2, 2, 2, 3]
```

[28]:
```
bpe_tokenizer.decode([3684, 7762, 26, 4413, 13, 4413])
```

[28]:
```
'      :    -   '
```

[29]:
```
tags=ds['train'].features['ner_tags'].feature
```

[30]:
```
tags.int2str(1)
```

[30]:
```
'B-PER'
```

[31]:
```
def tokenize_words(example):
    return bpe_tokenizer.encode(example,is_pretokenized=True).ids
```

[32]:
```
train['token_ids']=train['tokens'].apply(tokenize_words)
test['token_ids']=test['tokens'].apply(tokenize_words)
validation['token_ids']=validation['tokens'].apply(tokenize_words)
```

[33]:
```
train.head()
```

[33]:

	id	tokens	ner_tags
0	0	[, , , , , , , ,…	[6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, …
1	1	[, , , , , , , , …	[6, 6, 6, 0, 3, 6, 6, 6, 6, 6, 6]
2	2	[, , ,, , ,, , , …	[0, 0, 6, 0, 6, 0, 3, 6, 6, 6, 6]
3	3	[, , , - , , …	[6, 6, 6, 6, 6, 6, 6, 0, 6, 6, 6, 6, 6]
4	4	[, , , , , , , …	[6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6]

```
                                                token_ids
0  [995, 2312, 935, 1285, 4190, 937, 2126, 1072, …
1  [3140, 1738, 1149, 19612, 18475, 934, 1103, 73…
2  [4290, 3791, 12, 1595, 12, 1086, 1033, 1021, 9…
3  [2472, 2191, 7071, 4250, 4695, 16342, 13, 8897…
4  [1725, 1455, 15391, 4025, 4172, 934, 1021, 972…
```

[34]:
```
def get_word_ids(example):
    return bpe_tokenizer.encode(example,is_pretokenized=True).word_ids
```

[35]:
```
train['word_ids']=train['tokens'].apply(get_word_ids)
test['word_ids']=test['tokens'].apply(get_word_ids)
validation['word_ids']=validation['tokens'].apply(get_word_ids)
```

[36]:
```
train.head()
```

[36]:
```
  id                                             tokens  \
0  0  [ ,    ,   ,   ,    ,  ,    ,   ,…
1  1  [ ,   ,   ,   ,    ,  ,   ,    ,  …
2  2  [   ,    , ,,    , ,,   ,   ,  …
3  3  [    ,     ,    ,   - ,  ,   …
4  4  [ ,   ,   ,   ,    ,  ,   , …

                                              ner_tags  \
0  [6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, …
1                  [6, 6, 6, 0, 3, 6, 6, 6, 6, 6, 6]
2                  [0, 0, 6, 0, 6, 0, 3, 6, 6, 6, 6]
3            [6, 6, 6, 6, 6, 6, 6, 0, 6, 6, 6, 6, 6]
4      [6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6]

                                              token_ids  \
0  [995, 2312, 935, 1285, 4190, 937, 2126, 1072, …
1  [3140, 1738, 1149, 19612, 18475, 934, 1103, 73…
2  [4290, 3791, 12, 1595, 12, 1086, 1033, 1021, 9…
3  [2472, 2191, 7071, 4250, 4695, 16342, 13, 8897…
4  [1725, 1455, 15391, 4025, 4172, 934, 1021, 972…

                                              word_ids
0  [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,…
1                  [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
2              [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 10]
3  [0, 0, 1, 2, 3, 3, 3, 3, 4, 5, 6, 7, 8, 9, 10,…
4  [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 11, 12,…
```

[37]:
```
print(train.iloc[10,4])
print(train.iloc[10,2])
print(len(train.iloc[10,4]))
```

```
print(len(train.iloc[10,2]))
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 11, 11, 12, 13, 14, 15, 16, 17, 18, 19,
20, 21, 22, 23, 24, 25, 26, 27, 28, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38,
39, 40, 40]
[6 6 6 6 1 6 2 5 0 3 6 6 1 6 2 5 5 0 6 0 6 6 6 2 5 0 3 6 6 6 6 6 2 5 0 3 6
 0 6 6 6]
45
41
```

since we have used byte pair encoding tokenizer to encode the text which can split the words in several different subwords. So our NER tags cab be mismatched in this case. we will use same tag for these subwords.

[38]:
```
def align_ner_tags(ner_tags,word_ids):
  adjusted_tag=[]
  for i in word_ids:
    adjusted_tag.append(ner_tags[i])
  return adjusted_tag
```

[39]:
```
adj=align_ner_tags(train.iloc[10,2],train.iloc[10,4])
print(adj)
```

```
[6, 6, 6, 6, 1, 6, 2, 5, 0, 3, 6, 6, 6, 6, 1, 6, 2, 5, 5, 0, 6, 0, 6, 6, 6, 2,
5, 0, 3, 6, 6, 6, 6, 6, 6, 2, 5, 0, 3, 6, 0, 6, 6, 6, 6]
```

[40]:
```
print(len(adj))
```

```
45
```

[41]:
```
adjusted_tags=[]
for i in range(len(train)):
  adjusted_tags.append(align_ner_tags(train.iloc[i,2],train.iloc[i,4]))
```

[42]:
```
train['adjusted_tags']=adjusted_tags
```

[43]:
```
train.head()
```

[43]:

	id	tokens	ner_tags
0	0	[, , , , , , , ,…	[6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, …
1	1	[, , , , , , , , …	[6, 6, 6, 0, 3, 6, 6, 6, 6, 6, 6]
2	2	[, , ,, , ,, , , …	
3	3	[, , , - , , …	
4	4	[, , , , , , , …	

```
2                  [0, 0, 6, 0, 6, 0, 3, 6, 6, 6, 6]
3            [6, 6, 6, 6, 6, 6, 6, 0, 6, 6, 6, 6, 6]
4      [6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6]

                                              token_ids  \
0  [995, 2312, 935, 1285, 4190, 937, 2126, 1072, …
1  [3140, 1738, 1149, 19612, 18475, 934, 1103, 73…
2  [4290, 3791, 12, 1595, 12, 1086, 1033, 1021, 9…
3  [2472, 2191, 7071, 4250, 4695, 16342, 13, 8897…
4  [1725, 1455, 15391, 4025, 4172, 934, 1021, 972…

                                               word_ids  \
0  [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,…
1                [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
2             [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 10]
3  [0, 0, 1, 2, 3, 3, 3, 3, 4, 5, 6, 7, 8, 9, 10,…
4  [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 11, 12,…

                                         adjusted_tags
0  [6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, …
1                 [6, 6, 6, 0, 3, 6, 6, 6, 6, 6, 6]
2              [0, 0, 6, 0, 6, 0, 3, 6, 6, 6, 6, 6]
3  [6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 0, 6, 6, 6, …
4   [6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6]
```

[44]:
```
adjusted_tags=[]
for i in range(len(test)):
  adjusted_tags.append(align_ner_tags(test.iloc[i,2],test.iloc[i,4]))
```

[45]:
```
test['adjusted_tags']=adjusted_tags
```

[46]:
```
adjusted_tags=[]
for i in range(len(validation)):
  adjusted_tags.append(align_ner_tags(validation.iloc[i,2],validation.
 ↪iloc[i,4]))
```

[47]:
```
validation['adjusted_tags']=adjusted_tags
```

[48]:
```
train.head()
```

[48]:
```
   id                                             tokens  \
0   0  [ ,    ,   ,   ,    ,  ,    ,   ,…
1   1  [ ,   ,   ,   ,    ,  ,   ,    ,  …
2   2  [   ,    , ,,    , ,,   ,   ,  …
3   3  [    ,     ,    ,   - ,  ,   …
4   4  [  ,   ,   ,   ,    ,  ,   , …
```

```
                                                 ner_tags  \
0  [6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, …
1                  [6, 6, 6, 0, 3, 6, 6, 6, 6, 6, 6]
2                  [0, 0, 6, 0, 6, 0, 3, 6, 6, 6, 6]
3            [6, 6, 6, 6, 6, 6, 6, 0, 6, 6, 6, 6, 6]
4      [6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6]

                                               token_ids  \
0  [995, 2312, 935, 1285, 4190, 937, 2126, 1072, …
1  [3140, 1738, 1149, 19612, 18475, 934, 1103, 73…
2  [4290, 3791, 12, 1595, 12, 1086, 1033, 1021, 9…
3  [2472, 2191, 7071, 4250, 4695, 16342, 13, 8897…
4  [1725, 1455, 15391, 4025, 4172, 934, 1021, 972…

                                                word_ids  \
0  [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,…
1                [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
2             [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 10]
3  [0, 0, 1, 2, 3, 3, 3, 3, 4, 5, 6, 7, 8, 9, 10,…
4  [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 11, 12,…

                                           adjusted_tags
0  [6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, …
1                  [6, 6, 6, 0, 3, 6, 6, 6, 6, 6, 6]
2               [0, 0, 6, 0, 6, 0, 3, 6, 6, 6, 6, 6]
3  [6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 0, 6, 6, 6, …
4   [6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6]
```

1.4 Define the dataset

Here I am using max sequence length 512 so will pad the text with padding token which has index 1 in the tokenizer.

[50]:
```
class NERDataset(Dataset):
  def __init__(self,word_ids,token_tags):
    super(NERDataset,self).__init__()
    self.ids=word_ids
    self.tags=token_tags
  def __len__(self):
    return len(self.ids)
  def __getitem__(self,idx):
    if len(self.ids[idx])<512:
      padding=[1]*(512-len(self.ids[idx]))
      padding2=[-1]*(512-len(self.tags[idx]))
      self.ids[idx]+=padding
      self.tags[idx]+=padding2
    return torch.tensor(self.ids[idx]),torch.tensor(self.tags[idx])
```

[51]:
```python
train_dataset=NERDataset(train['token_ids'].to_list(),train['adjusted_tags'].
 ↪to_list())
```

[52]:
```python
train_dataset[0]
```

[52]:
```
(tensor([  995,  2312,   935,  1285,  4190,   937,  2126,  1072,  1031,   961,
          1058,  1031,   944,   995,  2312,   935,  8589,  2788,  5012,   930,
           944,  1365,  2509,  1334,  1000,  2109,   950,   422, 18429,   989,
          1494,   930,    14,     1,     1,     1,     1,     1,     1,     1,
             1,     1,     1,     1,     1,     1,     1,     1,     1,     1,
             1,     1,     1,     1,     1,     1,     1,     1,     1,     1,
             1,     1,     1,     1,     1,     1,     1,     1,     1,     1,
             1,     1,     1,     1,     1,     1,     1,     1,     1,     1,
             1,     1,     1,     1,     1,     1,     1,     1,     1,     1,
             1,     1,     1,     1,     1,     1,     1,     1,     1,     1,
             1,     1,     1,     1,     1,     1,     1,     1,     1,     1,
             1,     1,     1,     1,     1,     1,     1,     1,     1,     1,
             1,     1,     1,     1,     1,     1,     1,     1,     1,     1,
             1,     1,     1,     1,     1,     1,     1,     1,     1,     1,
             1,     1,     1,     1,     1,     1,     1,     1,     1,     1,
             1,     1,     1,     1,     1,     1,     1,     1,     1,     1,
             1,     1,     1,     1,     1,     1,     1,     1,     1,     1,
             1,     1,     1,     1,     1,     1,     1,     1,     1,     1,
             1,     1,     1,     1,     1,     1,     1,     1,     1,     1,
             1,     1,     1,     1,     1,     1,     1,     1,     1,     1,
             1,     1,     1,     1,     1,     1,     1,     1,     1,     1,
             1,     1,     1,     1,     1,     1,     1,     1,     1,     1,
             1,     1,     1,     1,     1,     1,     1,     1,     1,     1,
             1,     1,     1,     1,     1,     1,     1,     1,     1,     1,
             1,     1,     1,     1,     1,     1,     1,     1,     1,     1,
             1,     1,     1,     1,     1,     1,     1,     1,     1,     1,
             1,     1,     1,     1,     1,     1,     1,     1,     1,     1,
             1,     1,     1,     1,     1,     1,     1,     1,     1,     1,
             1,     1,     1,     1,     1,     1,     1,     1,     1,     1,
             1,     1,     1,     1,     1,     1,     1,     1,     1,     1,
             1,     1,     1,     1,     1,     1,     1,     1,     1,     1,
             1,     1,     1,     1,     1,     1,     1,     1,     1,     1,
             1,     1,     1,     1,     1,     1,     1,     1,     1,     1,
             1,     1,     1,     1,     1,     1,     1,     1,     1,     1,
             1,     1,     1,     1,     1,     1,     1,     1,     1,     1,
             1,     1,     1,     1,     1,     1,     1,     1,     1,     1,
             1,     1,     1,     1,     1,     1,     1,     1,     1,     1,
             1,     1,     1,     1,     1,     1,     1,     1,     1,     1,
             1,     1,     1,     1,     1,     1,     1,     1,     1,     1,
             1,     1,     1,     1,     1,     1,     1,     1,     1,     1,
             1,     1,     1,     1,     1,     1,     1,     1,     1,     1,
```

```
          1,      1,      1,      1,      1,      1,      1,      1,      1,      1,
          1,      1,      1,      1,      1,      1,      1,      1,      1,      1,
          1,      1,      1,      1,      1,      1,      1,      1,      1,      1,
          1,      1,      1,      1,      1,      1,      1,      1,      1,      1,
          1,      1,      1,      1,      1,      1,      1,      1,      1,      1,
          1,      1,      1,      1,      1,      1,      1,      1,      1,      1,
          1,      1,      1,      1,      1,      1,      1,      1,      1,      1,
          1,      1,      1,      1,      1,      1,      1,      1,      1,      1,
          1,      1,      1,      1,      1,      1,      1,      1,      1,      1,
          1,      1]),
 tensor([ 6,  6,  6,  6,  6,  6,  6,  6,  6,  6,  6,  6,  6,  6,  6,  6,  6,  6,
          6,  6,  6,  6,  6,  6,  6,  6,  6,  6,  6,  6,  6,  6,  6, -1, -1, -1,
         -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
         -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
         -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
         -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
         -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
         -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
         -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
         -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
         -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
         -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
         -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
         -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
         -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
         -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
         -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
         -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
         -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
         -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
         -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
         -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
         -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
         -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
         -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
         -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
         -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
         -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
         -1, -1, -1, -1, -1, -1, -1, -1]))
```

[ ]:

```
# this function generates the padding mask for single example.
def generate_padding_mask(batch):
  batch_size,seq_length=batch.shape
  mask=torch.full((batch_size,seq_length),False)
  indices=batch.argmin(dim=-1)
  for i in range(len(indices)):
    mask[i,0:indices[i]]=True
```

```
  return mask
```

1.5 Define the Model

In the model we are using 2 transformer encoder layer and d_model=256, feedforward dim=512 with 8 attension heads, Then finally a single fully connected layer which is an output layer

[67]:
```
class PositionalEncoding(nn.Module):
    def __init__(self, d_model, max_sequence_length):
        super().__init__()
        self.max_sequence_length = max_sequence_length
        self.d_model = d_model

    def forward(self):
        even_i = torch.arange(0, self.d_model, 2).float()
        denominator = torch.pow(10000, even_i/self.d_model)
        position = (torch.arange(self.max_sequence_length)
                            .reshape(self.max_sequence_length, 1))
        even_PE = torch.sin(position / denominator)
        odd_PE = torch.cos(position / denominator)
        stacked = torch.stack([even_PE, odd_PE], dim=2)
        PE = torch.flatten(stacked, start_dim=1, end_dim=2)
        return PE

class NERModel(nn.Module):
  def __init__(self,vocab_size,tagset_size,max_seq_length,d_model,num_layers):
    super(NERModel,self).__init__()
    self.embedding=nn.Embedding(vocab_size,d_model)
    self.transformer_encoder=nn.TransformerEncoder(nn.
 ↪TransformerEncoderLayer(d_model=256,nhead=8,dim_feedforward=max_seq_length,batch_first=True
    self.fc=nn.Linear(d_model,tagset_size)
    self.pos=PositionalEncoding(d_model,max_seq_length)
  def forward(self, word_ids):
        mask = generate_padding_mask(word_ids)
        word_ids = word_ids.long()
        word_ids = self.embedding(word_ids)
        word_ids = word_ids + self.pos().to(word_ids.device)
        word_ids = self.transformer_encoder(word_ids, src_key_padding_mask=mask.
 ↪to(word_ids.device))
        word_ids = self.fc(word_ids)
        return word_ids
```

[ ]:
```
vocab_size=30000
target_size=7
max_seq_length=512
d_model=256
num_layers=2
```

```
batch_size=32
learning_rate=0.001
num_epochs=10
device=torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

[68]:
```
model=NERModel(vocab_size,target_size,max_seq_length,d_model,num_layers)
```

[58]:
```
dataloader=DataLoader(train_dataset,batch_size=batch_size,shuffle=True)
```

[71]:
```
criterion=nn.CrossEntropyLoss()
optimizer=torch.optim.Adam(model.parameters(),lr=learning_rate)
```

[69]:
```
model=model.to(device)
```

[61]:
```
device
```

[61]:
```
device(type='cuda')
```

[72]:
```
for i in range(num_epochs):
  model.train()
  for x,y in dataloader:
    x=x.to(device)
    y=y.to(device)
    out=model(x)
    indices=y.argmin(dim=-1)
    loss=0
    for j in range(len(indices)):
      loss+=criterion(out[j,0:indices[j]],y[j,0:indices[j]])
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
    print(f"epoch={i} loss = {loss.item()}")
```

```
Streaming output truncated to the last 5000 lines.
epoch=7 loss = 7.416934013366699
epoch=7 loss = 3.2502119541168213
epoch=7 loss = 8.193572998046875
epoch=7 loss = 2.7964730262756348
epoch=7 loss = 4.981128692626953
epoch=7 loss = 5.639580726623535
epoch=7 loss = 3.72403883934021
epoch=7 loss = 4.152966499328613
epoch=7 loss = 7.028319358825684
epoch=7 loss = 4.692986488342285
epoch=7 loss = 7.036635875701904
epoch=7 loss = 3.992140293121338
epoch=7 loss = 4.548179626464844
```

```
epoch=9 loss = 8.880252838134766
epoch=9 loss = 4.683468341827393
epoch=9 loss = 4.857133865356445
epoch=9 loss = 5.131529808044434
epoch=9 loss = 6.338727951049805
epoch=9 loss = 4.59323263168335
epoch=9 loss = 5.106088638305664
epoch=9 loss = 6.462996959686279
epoch=9 loss = 5.359297752380371
epoch=9 loss = 5.319521903991699
epoch=9 loss = 7.794879913330078
epoch=9 loss = 4.387789249420166
epoch=9 loss = 4.304499626159668
epoch=9 loss = 5.434600830078125
epoch=9 loss = 8.792350769042969
epoch=9 loss = 5.3236308097839355
epoch=9 loss = 4.524515151977539
epoch=9 loss = 4.815862655639648
epoch=9 loss = 4.342594623565674
epoch=9 loss = 5.737475872039795
epoch=9 loss = 5.666189670562744
epoch=9 loss = 5.104007244110107
epoch=9 loss = 4.474134922027588
epoch=9 loss = 8.019667625427246
epoch=9 loss = 5.391564846038818
epoch=9 loss = 4.250690937042236
epoch=9 loss = 5.205535411834717
epoch=9 loss = 6.9742584228515625
epoch=9 loss = 4.910806179046631
epoch=9 loss = 5.168911933898926
epoch=9 loss = 4.688228130340576
epoch=9 loss = 5.7520647048950195
epoch=9 loss = 5.374390602111816
epoch=9 loss = 5.114196300506592
epoch=9 loss = 5.577239513397217
epoch=9 loss = 8.540487289428711
epoch=9 loss = 5.302555084228516
epoch=9 loss = 3.46586275100708
epoch=9 loss = 4.407724380493164
epoch=9 loss = 6.237401008605957
epoch=9 loss = 5.690214157104492
epoch=9 loss = 3.746105432510376
epoch=9 loss = 2.949219226837158
```

[73]:
```
torch.save(model, "model.pt")
```

1.6 test the model

[74]:
```
test_dataset=NERDataset(test['token_ids'].to_list(),test['adjusted_tags'].
 ↪to_list())
test_dataloader=DataLoader(test_dataset,batch_size=32,shuffle=True)
```

[75]:
```
# evaluate the model on test data
model.eval()
with torch.no_grad():
  correct = 0
  total = 0
  for x, y in test_dataloader:
    x=x.to(device)
    y=y.to(device)
    out = model(x)
    indices=y.argmin(dim=-1)
    for i in range(len(indices)):
      correct += (out[i,0:indices[i]].argmax(dim=-1) == y[i,0:indices[i]]).
 ↪sum().item()
      total += indices[i].sum().item()
  accuracy = correct / total
  print("Test Accuracy:", accuracy)
```

```
Test Accuracy: 0.9426961824503889
```

Although we have a good test accuracy but it is overfitted. We have a small dataset and we have trained it only for 10 epochs. Let's test it with unseen data.

[76]:
```
example="            "
```

[79]:
```
tokens=bpe_tokenizer.encode(example)
print(tokens)
```

```
Encoding(num_tokens=5, attributes=[ids, type_ids, tokens, offsets,
attention_mask, special_tokens_mask, overflowing])
```

[82]:
```
ids=tokens.ids
print(ids)
```

```
[2449, 1377, 19501, 2959, 930]
```

[81]:
```
tokens.tokens
```

[81]:
```
[' ', ' ', '  ', ' ', ' ']
```

[83]:
```
padding=[1]*(512-len(ids))
ids+=padding
print(ids)
```

```
[2449, 1377, 19501, 2959, 930, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1]
```

[84]:
```
out=model(torch.tensor(ids).unsqueeze(0).to(device))
```

[86]:
```
out=out.squeeze()
```

[87]:
```
probs=F.softmax(out,dim=-1)
```

[88]:
```
probs.argmax(dim=-1)
```

[88]:
```
tensor([6, 6, 3, 5, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
        6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
        6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
        6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
        6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
        6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
        6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
        6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
        6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
        6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
        6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
        6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
        6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
        6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
        6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
        6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
        6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
        6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
```

```
6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
6, 6, 6, 6, 6, 6, 6, 6], device='cuda:0')
```

It has correctly identified “khan” as person name but “rahim” is incorrectly identified.

[ ]: