Intro to NLP

Assignment - 2

Soumodipta Bose
2021201086

Files Structure:

Configurable Script:

The scripts parameters are configurable using config.json:

Exploring Data and Pre-processing

Preparing the dataset from the conllu form:

```
IGNORE_TAG_INDEX = 0
def create_data(sentences,vocab,tags,max_seq_len=50):
    sents_idx=[]
    sent_tags=[]
    #present=0
    #not_present=0
    for sent in sentences:
        sent_idx=[]
        sent_tag=[]
        for token in sent:
            if (token["form"].lower() in vocab):
                sent_idx.append(vocab[token["form"].lower()])
            else:
                sent_idx.append(vocab["<UNK>"])
            sent_tag.append(tags[token["upos"]])
        sents_idx.append(sent_idx)
        sent_tags.append(sent_tag)
    for i in range(len(sents_idx)):
        if len(sents_idx[i]) < max_seq_len:
            sents_idx[i]=sents_idx[i]+[vocab["<PAD>"] for _ in range(max_seq_len - len(sents_idx[i]))]
            sent_tags[i]=sent_tags[i]+[tags["PAD"] for _ in range(max_seq_len - len(sent_tags[i]))]
#     print(present)
#     print(not_present)
    return sents_idx,sent_tags
```

Making the dataset compatible for data loading

```
class POSDataSet(Dataset):
    def __init__(self, x, y):
        self.sent = torch.LongTensor(x)
        self.sent_tags = torch.LongTensor(y)

    def __getitem__(self, idx):
        return self.sent[idx], self.sent_tags[idx]

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

Using Pytorch's Data loader to create mini-batches

```
train_dataset = POSDataSet(x_train,y_train)
dev_dataset = POSDataSet(x_dev,y_dev)
test_dataset = POSDataSet(x_test,y_test)
```

```
BATCH_SIZE =32
train_dataloader = DataLoader(train_dataset, batch_size=BATCH_SIZE, shuffle=True)
dev_dataloader = DataLoader(dev_dataset, batch_size=BATCH_SIZE, shuffle=False)
test_dataloader = DataLoader(test_dataset, batch_size=BATCH_SIZE, shuffle=False)
```

Model Creation

The model was created using Pytorch. The model consisted of an initial embedding layer that used pre-trained embedding embeddings using Glove. Then was fed into an LSTM and finally a Linear Fully connected layer with output dimensions of the tag set size.

```
class POSTagger(nn.Module):
    def __init__(self,max_seq_len,embeddings,hidden_dim,n_layers,tagset_size):
        super().__init__()
        self.max_seq_len = max_seq_len
        self.num_labels = tagset_size
        self.hidden_dim = hidden_dim
        self.embeddings = nn.Embedding.from_pretrained(embeddings,padding_idx=0)
        self.lstm = nn.LSTM(input_size=embeddings.size()[1], hidden_size= self.hidden_dim , num_layers=n_layers)
        self.hidden2tag = nn.Linear(self.hidden_dim,self.num_labels)

    def forward(self,input_seq):
        device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
        input_seq = input_seq.to(device)
        embed_out =self.embeddings(input_seq)
        lstm_out,_ = self.lstm(embed_out)
        logits = self.hidden2tag(lstm_out)
        return logits
```

Hyper Parameter Tuning

I experimented with different learning rates, hidden dimensions, embedding sizes and hidden layers to find the best fit for the model.

00%| | 50/50 [00:21<00:00, 2.38it/s]
Model : Training Accuracy : 0.9444250333984174 Validation Accuracy: 0.9430091185410334

Model Parameters | Learning Rate : 0.0001 Layers : 1 Hidden Dim : 32
Analysis | : Test Accuracy : 0.9430091185410334

Model : Training Accuracy : 0.9526872880485048 Validation Accuracy: 0.952887537993921

Model Parameters | Learning Rate : 0.0001 Layers : 1 Hidden Dim : 64
Analysis | : Test Accuracy : 0.952887537993921

100%| | 50/50 [00:23<00:00, 2.16it/s]
Model : Training Accuracy : 0.9576816360086322 Validation Accuracy: 0.9592705167173252
…
…
100%| | 50/50 [00:51<00:00, 1.02s/it]
Model : Training Accuracy : 0.9641968965162881 Validation Accuracy: 0.9630699088145896

Model Parameters | Learning Rate : 0.0001 Layers : 2 Hidden Dim : 256
Analysis | : Test Accuracy : 0.9630699088145896

100%| | 50/50 [01:12<00:00, 1.45s/it]
Model : Training Accuracy : 0.9637652861987462 Validation Accuracy: 0.9629179331306991

Model Parameters | Learning Rate : 0.0001 Layers : 2 Hidden Dim : 512
Analysis | : Test Accuracy : 0.9629179331306991

Finally the best configuration was chosen to be :

Learning rate = 0.0005

Hidden Dimensions = 128

Hidden layers = 2

Embedding Size = 100

Analyzing the scores of the tags

Calculating the precision, recall and f1-score of all the different tags. As we can see pad has a f1-score of zero because we ignore the pad tag during gradient calculation in the model, this greatly saves us from miss-classification errors where words were being incorrectly tagged as pads.

	precision	recall	f1-score	support
PAD	0.00	0.00	0.00	0
ADJ	0.93	0.97	0.95	220
ADP	0.98	0.99	0.98	1434
ADV	0.95	0.72	0.82	76
AUX	0.96	1.00	0.98	256
CCONJ	1.00	1.00	1.00	109
DET	0.98	0.87	0.92	512
INTJ	0.97	1.00	0.99	36
NOUN	0.99	0.97	0.98	1166
NUM	0.95	0.96	0.95	127
PART	0.72	0.52	0.60	56
PRON	0.85	0.98	0.91	392
PROPN	0.98	0.99	0.99	1567
SYM	0.00	0.00	0.00	0
VERB	0.98	0.97	0.98	629
micro avg	0.97	0.97	0.97	6580
macro avg	0.82	0.80	0.80	6580
weighted avg	0.97	0.97	0.97	6580

Test dataset

Output

Following is the output of the pos tagger:

```
❯ python3 pos_tagger.py
Mary has a little lamb
Mary    PROPN
has     VERB
a       DET
little  ADJ
lamb    VERB
```

Model Exploration

Improving the version of the model using Dropout and Bi-directional LSTM.

```
class ImprovedPOSTagger(POSTagger):
    def __init__(self,max_seq_len,embeddings,hidden_dim,n_layers,tagset_size,dropout=0.8,bidirectional=True):
        super().__init__(max_seq_len,embeddings,hidden_dim,n_layers,tagset_size)
        self.lstm = nn.LSTM(input_size=embeddings.size()[1], hidden_size=hidden_dim, dropout=dropout, num_layers=n_layers, bidirectional=bidirectional)
        self.hidden2tag = nn.Linear(self.hidden_dim*2,self.num_labels)
```

By improving the model two more parameters were added:

Dropout = 0.3

Bidirectional = True

Final Accuracy:

Dataset	Accuracy
Training	0.9687801870311376
Validation	0.9614689945815773
Test	0.9667173252279635