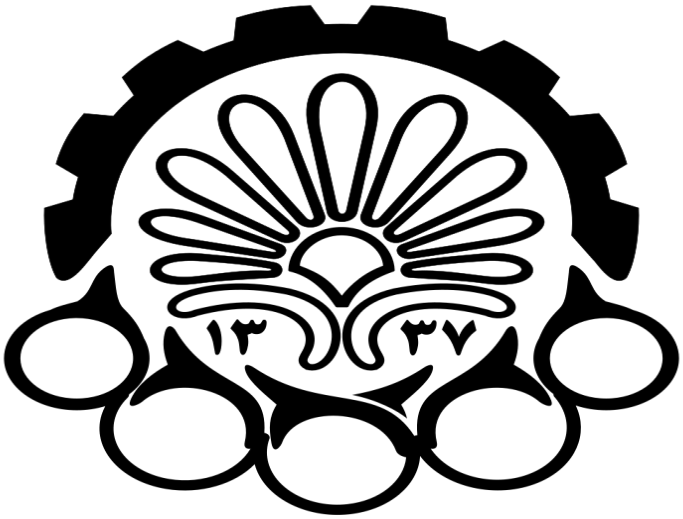


دانشگاه صنعتی امیرکبیر

(پلی تکنیک تهران)

درس پردازش زبان طبیعی

استاد ممتازی

نیما پری فرد

۴۰۲۱۳۱۰۱۷

فهرست تمرین اول پردازش زبان طبیعی

توجه

شاید شکل ظاهری بعضی screenshot ها گرفته‌شده متفاوت باشد، دلیلش این این است بعضی ها از colab برداشتم و بعضی ها را از pycharm برداشتم.

بخش ۱

مشاهده داده و اعمال پیش پردازش های لازم

پیش‌پردازش هایی که انجام دادم:

- نیم فاصله ها را حذف کردم چون نمی شناخت آن ها را و بجاش \u200c می‌گذاشت و کلمه جدید به اشتباه تولید می‌کرد.
- token بندی کردم

```
def preprocess_text(text):
    text = text.replace('u200c', '')
    text = text.replace('\\', '')
    text = text.replace('\n', '')
    words = word_tokenize(text)
    return words
# Apply preprocessing to the 'sentences' column
train['sentences'] = train['sentences'].apply(preprocess_text)
validation['sentences'] = validation['sentences'].apply(preprocess_text)
test['sentences'] = test['sentences'].apply(preprocess_text)
Executed at 2024.05.25 22:57:50 in 12s 83ms
```

- بعضی داده ها طول جمله شان با طول تگ ها یکسان نبود آن ها را حذف کردم.

```
def get_mismatched_rows(dataset):
    mismatched_indices = []
    for i, (sentence, tag) in enumerate(zip(dataset.sentences, dataset.pos_tags)):
        if len(sentence) != len(tag):
            mismatched_indices.append(i)
    return mismatched_indices

# Get the indices of the mismatched rows in the train dataset
mismatched_indices_train = get_mismatched_rows(train)
mismatched_indices_test = get_mismatched_rows(test)
mismatched_indices_valid = get_mismatched_rows(validation)

# Drop the mismatched rows from the train dataset
train = train.drop(mismatched_indices_train)
test = test.drop(mismatched_indices_test)
validation = validation.drop(mismatched_indices_valid)
Executed at 2024.05.25 22:57:51 in 122ms
```

- دیتاست مخصوص داده ها به صورت ارث‌بری از تابع عمومی دیتاست در پایتورچ درست کردم در آن پدینگ انجام می‌شود تا طول جملات یکسان شوند. همچنین وقتی می خواد خروجی بده اندیکس لغات و تگ ها را به جای خود لغات و تگ ها قرار می‌دهد.

```python
class POSDataset(Dataset):
    def __init__(self, df, words_dict=None, pos_dict=None, max_length=None):
        self.df = df
        self.words_dict = words_dict
        self.pos_dict = pos_dict
        self.max_length = max_length

    def __len__(self):
        return len(self.df)

    def __getitem__(self, idx):
        sentence = self.df.iloc[idx]['sentences']
        pos_tags = self.df.iloc[idx]['pos_tags']

        # Pad the sequences to the maximum length
        if len(sentence) < self.max_length:
            sentence += ['PAD'] * (self.max_length - len(sentence))
            pos_tags += ['PAD'] * (self.max_length - len(pos_tags))

        return sentence, pos_tags

    def collate_fn(self, batch):
        sentences, pos_tags = zip(*batch)
        sentences = [[self.words_dict[word] if self.words_dict.get(word) else 0 for word in sentence] for sentence in sentences]
        pos_tags = [[self.pos_dict[tag] for tag in pos_tag] for pos_tag in pos_tags]
        return torch.LongTensor(sentences), torch.tensor(pos_tags)
```

Executed at 2024.05.25 22:57:51 in 161ms

```python
train_dataset = POSDataset(train,words_dict=words_dict,pos_dict=tags_dict,max_length=max_length)
valid_dataset = POSDataset(validation, words_dict=words_dict,pos_dict=tags_dict,max_length=max_length)
test_dataset = POSDataset(test,  words_dict=words_dict,pos_dict=tags_dict,max_length=max_length)

# Create DataLoaders
train_dataloader = DataLoader(train_dataset, batch_size=512, shuffle=True, collate_fn=train_dataset.collate_fn)
valid_dataloader = DataLoader(valid_dataset, batch_size=1000, shuffle=False, collate_fn=train_dataset.collate_fn)
test_dataloader = DataLoader(test_dataset, batch_size=1000, shuffle=False, collate_fn=train_dataset.collate_fn)
```

Executed at 2024.05.25 22:58:01 in 72ms

- داده ها بسیار کثیف است چون داده های توییت هست برای همین توقع درصد بالا نباید داشت.

بخش ۲

بخش الف

کد مدل استفاده شده برای ران کردن قسمت بعدی:

نیما پری فرد ۴۰۲۱۱۳۱۰۷

```python
# Define the model
class RNNPOSTagger(nn.Module):
    def __init__(self, vocab_size=5000, embedding_dim=300, hidden_dim=300, output_dim=300, n_layers=2, bidirectional=False, dropout=0.5):
        super().__init__()
        self.embedding = nn.Embedding(vocab_size, embedding_dim)
        self.rnn = nn.LSTM(embedding_dim, hidden_dim, num_layers = n_layers, bidirectional = bidirectional)
        self.fc = nn.Linear(hidden_dim * 2 if bidirectional else hidden_dim, output_dim)
        self.dropout = nn.Dropout(dropout)

    def forward(self, text):
        embedded = self.dropout(self.embedding(text))
        outputs, (hidden, cell) = self.rnn(embedded)
        predictions = self.fc(self.dropout(outputs))
        return predictions
```

در مدل بالا یک جمله به آن می‌دهیم و در نهایت تگ های آن را خروجی می‌دهد. البته این قبل آموزش است یادم رفت خروجی بگیرم و بعد در صورت سوال متوجه شدم که می‌خواد اینو خروجی بگیرم با وزن های رندم یک مدل ساختم فقط خروجی گرفتم.

```python
def predict_sentence(sentence, model, words_dict, tags_reverse_dict):
    # Preprocess the sentence
    sentence = preprocess_text(sentence)
    # Convert the sentence to the input format required by the model
    sentence_input = [words_dict[word] if words_dict.get(word) else 0 for word in sentence]
    sentence_input = torch.LongTensor(sentence_input).unsqueeze(0).to(device)
    # Pass the sentence through the model
    model.eval()
    with torch.no_grad():
        outputs = model(sentence_input)
        _, preds = torch.max(outputs, 2)
    # Convert the output to tags
    predicted_tags = [tags_reverse_dict[tag] for tag in preds[0].tolist()]
    return predicted_tags

# Use the function
sentence = "دکتر اصغری رف اب بخوره"
model = RNNPOSTagger(vocab_size=num_words, embedding_dim=300, hidden_dim=300, output_dim=num_classes, n_layers=2)
predicted_tags = predict_sentence(sentence, model, words_dict, tags_reverese_dict)
print(predicted_tags)
```

Executed at 2024.05.27 23:41:45 in 184ms

```
['NUM', 'NUM', 'N', 'ADV', 'N']
```

معماری شبکه را با استفاده کد زیر نمایش دادم:

```python
summary(
    RNNPOSTagger(vocab_size=num_words, embedding_dim=200, hidden_dim=200, output_dim=num_classes, n_layers=4),
    (23,),
    dtypes=[torch.long],
    branching=False,
    verbose=2,
    col_width=16,
    col_names=["kernel_size", "output_size", "num_params", "mult_adds"],
)
```

Executed at 2024.05.25 18:40:35 in 188ms

```
==========================================================================================
Layer (type:depth-idx)                   Kernel Shape     Output Shape     Param #         Mult-Adds
==========================================================================================
Embedding: 1-1                           [200, 30739]     [-1, 23, 200]    6,147,800       6,147,800
Dropout: 1-2                             --               [-1, 23, 200]    --              --
LSTM: 1-3                                --               [-1, 23, 200]    1,286,400       1,280,000
  weight_ih_l0                           [800, 200]
  weight_hh_l0                           [800, 200]
  weight_ih_l1                           [800, 200]
  weight_hh_l1                           [800, 200]
  weight_ih_l2                           [800, 200]
  weight_hh_l2                           [800, 200]
  weight_ih_l3                           [800, 200]
  weight_hh_l3                           [800, 200]
Dropout: 1-4                             --               [-1, 23, 200]    --              --
Linear: 1-5                              [200, 24]        [-1, 23, 24]     4,824           4,800
==========================================================================================
Total params: 7,439,024
Trainable params: 7,439,024
Non-trainable params: 0
Total mult-adds (M): 7.43
==========================================================================================
Input size (MB): 0.00
Forward/backward pass size (MB): 0.07
Params size (MB): 28.38
Estimated Total Size (MB): 28.45
==========================================================================================
```

با استفاده از کلاس ModelOptimization مدل را آموزش دادم. تقریبا براین این کلاس از کلاس برای آموزش مدل‌ها استفاده کردم.

درون وقتی مدل را load می‌کنم یک عمل مقداردهی اولیه با استفاده از xavier_normal مقدار دهی کردم.

```python
def xavier_normal_init(self):
    for module in self.model.modules():
        if isinstance(module, nn.Linear):
            nn.init.xavier_normal_(module.weight)
            if module.bias is not None:
                nn.init.constant_(module.bias, 0)
```

```python
class ModelOptimization:
    def __init__(self, model, train_dataloader, val_dataloader, device, vocab_dict, num_epochs=1, lr=0.0001, step_size=50, gamma=0.5, weight_decay=0.0, runs_name='final_train'):
        self.model = model
        self.xavier_normal_init()
        self.train_dataloader = train_dataloader
        self.val_dataloader = val_dataloader
        self.device = device
        self.num_epochs = num_epochs
        # self.criterion = nn.CrossEntropyLoss(ignore_index=0)
        self.criterion = nn.CrossEntropyLoss()
        self.optimizer = optim.Adam(self.model.parameters(), lr=lr, weight_decay=weight_decay)
        self.scheduler = lr_scheduler.StepLR(self.optimizer, step_size=step_size, gamma=gamma)
        self.writer = SummaryWriter(f'runs/{runs_name}')
        self.training_losses = []
        self.validation_losses = []
        self.training_accuracies = []
        self.validation_accuracies = []
        self.best_model = model
        self.best_accuracy = 0.0
        self.vocab_dict = vocab_dict
```

با استفاده از تابع مدل را یک epoch آموزش می‌دادم.

```python
def train_one_epoch(self, epoch):
    self.model.train()
    running_loss = 0.0
    y_test = []
    y_preds = []
    with tqdm(enumerate(self.train_dataloader),unit='batch',total=len(self.train_dataloader)) as progress_bar:
        for i, (inputs, labels) in progress_bar:
            inputs = inputs.to(self.device)
            labels = labels.to(self.device)
            outputs = self.model(inputs)
            outputs_copy = outputs
            outputs = outputs.view(-1, outputs.shape[-1])
            labels_copy = labels
            labels = labels.view(-1)
            loss = self.criterion(outputs, labels)
            _, preds = torch.max(outputs_copy, 2)
            self.optimizer.zero_grad()
            loss.backward()
            self.optimizer.step()
            running_loss += loss.item() * inputs.size(0)
            y_test.extend(labels_copy.tolist())
            y_preds.extend(preds.tolist())
            progress_bar.set_postfix(Epoch=epoch + 1, Loss=running_loss / len(self.train_dataloader.dataset))
        predicted_tags, actual_tags = self.flat_tags(y_preds, y_test)
        epoch_acc = accuracy_score(predicted_tags, actual_tags) * 100
        epoch_loss = running_loss / len(self.train_dataloader.dataset)
    return epoch_loss, epoch_acc
```

با استفاده از تابع زیر مدل را به حالت eval می‌بردم تا نتایج روی داده های validation ارزیابی کنم.

```python
def evaluate(self):
    self.model.eval()
    running_val_loss = 0.0
    y_test = []
    y_preds = []
    with torch.no_grad():
        for inputs, labels in self.val_dataloader:
            inputs = inputs.to(self.device)
            labels = labels.to(self.device)
            # inputs.requires_grad =True
            outputs = self.model(inputs)
            _, preds = torch.max(outputs, 2)
            outputs = outputs.view(-1, outputs.shape[-1])
            labels_copy = labels
            labels = labels.view(-1)
            loss = self.criterion(outputs, labels)
            running_val_loss += loss.item() * inputs.size(0)
            y_test.extend(labels_copy.tolist())
            y_preds.extend(preds.tolist())
    predicted_tags, actual_tags = self.flat_tags(y_preds, y_test)
    val_accuracy = accuracy_score(predicted_tags, actual_tags) * 100
    epoch_val_loss = running_val_loss / len(self.val_dataloader.dataset)
    return epoch_val_loss, val_accuracy
```

با استفاده از تابع زیر خروجی مدل را به حالت flat در می‌آورم تا در نهایت بتوانم دقت را خروجی را با لیبل ها مقایسه کنم.

```python
def flat_tags(self, preds, labels):
    flat_predicted_tags = []
    flat_actual_tags = []
    for pred, label in zip(preds, labels):
        for p, l in zip(pred, label):
            if l > 0:
                flat_actual_tags.append(l)
                flat_predicted_tags.append(p)
    return flat_predicted_tags, flat_actual_tags
```

با استفاده از تابع مدل را با تعداد epoch دریافت شده آموزش می‌دهم تا نتایج را برای هر epoch خروجی می‌دادم.

```python
def train(self, num_epochs=0):
    if num_epochs != 0:
        self.num_epochs = num_epochs
    for epoch in range(self.num_epochs):
        train_loss, train_acc = self.train_one_epoch(epoch)
        val_loss, val_acc = self.evaluate()
        self.training_losses.append(train_loss)
        self.validation_losses.append(val_loss)
        self.training_accuracies.append(train_acc)
        self.validation_accuracies.append(val_acc)
        print(f"Epoch {epoch + 1}/{self.num_epochs} Training Loss: {train_loss:.4f} Training Accuracy: {train_acc:.4f} Validation Loss: {val_loss:.4f}
         Validation Accuracy: {val_acc:.4f}")
        self.writer.add_scalar('RNNPos/Loss/train', train_loss, epoch)
        self.writer.add_scalar('RNNPos/Accuracy/train', train_acc, epoch)

        self.writer.add_scalar('RNNPos/Loss/validation', val_loss, epoch)
        if val_acc > self.best_accuracy:
            self.best_accuracy = val_acc
            self.best_model = self.model.state_dict()

        # Step the scheduler
        self.scheduler.step()

    self.writer.close()
    print(f"Best validation accuracy: {self.best_accuracy}")
```

در نهایت مدل را سیو می‌کنم.

```python
def save_model(self, path='pos_model_weights.pth'):
    torch.save(self.best_model, path)
```

uted at 2024.05.25 23:22:50 in 151ms

برای آموزش مدل که تهش CRF داره، یک کلاس جدید ساختم برای آموزش چون خروجی آن در نهایت از لایهCrf می‌اید کمی بعضی از خروجی گرفتن ها متفاوت است.

```python
class CRFModelOptimization:
    def __init__(self, model, train_dataloader, val_dataloader, device, vocab_dict, num_epochs=1, lr=0.0001, step_size=50, gamma=0.5, weight_decay=0.0,
     runs_name='final_train'):
        self.model = model
        self.xavier_normal_init()
        self.train_dataloader = train_dataloader
        self.val_dataloader = val_dataloader
        self.device = device
        self.num_epochs = num_epochs
        self.criterion = nn.CrossEntropyLoss()
        self.optimizer = optim.Adam(self.model.parameters(), lr=lr, weight_decay=weight_decay)
        self.scheduler = lr_scheduler.StepLR(self.optimizer, step_size=step_size, gamma=gamma)
        self.writer = SummaryWriter(f'runs/{runs_name}')
        self.training_losses = []
        self.validation_losses = []
        self.training_accuracies = []
        self.validation_accuracies = []
        self.best_model = model
        self.best_accuracy = 0.0
        self.vocab_dict = vocab_dict
```

دیکشنری تگ ها ساختم برای متوجه شدن خروجی ها این دیکشنری قرار دادم.

```
{'AJ': 1,
 'PAD': 0,
 'NUM': 2,
 'DETe': 3,
 'POSTP': 4,
 'NUMe': 5,
 'PRO': 6,
 'PROe': 7,
 'CL': 8,
 'Pe': 9,
 'ADV': 10,
 'CONJe': 11,
 'N': 12,
 'CONJ': 13,
 'P': 14,
 'AJe': 15,
 'INT': 16,
 'Ne': 17,
 'RESe': 18,
 'PUNC': 19,
 'RES': 20,
 'ADVe': 21,
 'V': 22,
 'DET': 23}
```

بخش ب

با استفاده از داده های validation هایپرپارامترها را tune کردم.

```python
best_val_loss = float('inf')
best_model_optimization = None
best_model = None
best_hyperparams = None
learning_rates = [0.01]
weight_decays = [0,1e-5]
num_epochs = 5
for lr in learning_rates:
    for wd in weight_decays:
        print(f"Training with learning rate: {lr} and weight decay: {wd}")
        model = RNNPOSTagger(vocab_size=num_words+1, embedding_dim=300, hidden_dim=300, output_dim=num_classes, n_layers=2)
        model = model.to(device)
        for param in model.parameters():
            param.requires_grad = True

        model_optimization = ModelOptimization(model, train_dataloader, valid_dataloader, device,vocab_dict=words_dict, num_epochs=num_epochs, lr=lr, weight_decay=wd, runs_name='hyperparameter_tuning')

        model_optimization.train()
        val_loss = model_optimization.validation_losses[-1]

        if val_loss < best_val_loss:
            best_val_loss = val_loss
            best_hyperparams = (lr, wd)
            best_model_optimization = model_optimization
            best_model = model
print(f"Best hyperparameter: Learning rate: {best_hyperparams[0]}, Weight decay: {best_hyperparams[1]}")
```

```
Epoch 5/5 Training Loss: 0.2055  Validation Loss: 0.2356 Validation Accuracy: 66.6817
Best validation accuracy: 66.68174416500814
Training with learning rate: 0.01 and weight decay: 1e-05

100%|██████████| 19/19 [00:22<00:00,  1.17s/batch, Epoch=1, Loss=0.96]

Epoch 1/5 Training Loss: 0.9598  Validation Loss: 3.3780 Validation Accuracy: 0.7629

100%|██████████| 19/19 [00:22<00:00,  1.17s/batch, Epoch=2, Loss=0.411]

Epoch 2/5 Training Loss: 0.4107  Validation Loss: 0.4739 Validation Accuracy: 38.9814

100%|██████████| 19/19 [00:22<00:00,  1.17s/batch, Epoch=3, Loss=0.33]

Epoch 3/5 Training Loss: 0.3298  Validation Loss: 0.3561 Validation Accuracy: 49.8703

100%|██████████| 19/19 [00:22<00:00,  1.19s/batch, Epoch=4, Loss=0.28]

Epoch 4/5 Training Loss: 0.2801  Validation Loss: 0.3063 Validation Accuracy: 54.6861

100%|██████████| 19/19 [00:22<00:00,  1.17s/batch, Epoch=5, Loss=0.23]

Epoch 5/5 Training Loss: 0.2297  Validation Loss: 0.2512 Validation Accuracy: 64.4985
Best validation accuracy: 64.49852240516253
Best hyperparameter: Learning rate: 0.01, Weight decay: 0
```

اموزش مدل در نهایت:

```
best_model_optimization.train(30)
```

```
100%|██████████| 19/19 [00:03<00:00,  4.83batch/s, Epoch=1, Loss=0.186]
Epoch 1/30 Training Loss: 0.1857 Training Accuracy: 72.0912 Validation Loss: 0.1766 Validation Accuracy: 72.7127
100%|██████████| 19/19 [00:03<00:00,  5.03batch/s, Epoch=2, Loss=0.164]
Epoch 2/30 Training Loss: 0.1640 Training Accuracy: 74.6051 Validation Loss: 0.1674 Validation Accuracy: 73.6083
100%|██████████| 19/19 [00:04<00:00,  4.18batch/s, Epoch=3, Loss=0.15]
Epoch 3/30 Training Loss: 0.1498 Training Accuracy: 76.3319 Validation Loss: 0.1605 Validation Accuracy: 74.8055
100%|██████████| 19/19 [00:03<00:00,  4.98batch/s, Epoch=4, Loss=0.138]
Epoch 4/30 Training Loss: 0.1382 Training Accuracy: 78.1859 Validation Loss: 0.1560 Validation Accuracy: 76.7445
100%|██████████| 19/19 [00:03<00:00,  4.90batch/s, Epoch=5, Loss=0.127]
Epoch 5/30 Training Loss: 0.1267 Training Accuracy: 80.4223 Validation Loss: 0.1506 Validation Accuracy: 77.9869
100%|██████████| 19/19 [00:04<00:00,  3.85batch/s, Epoch=6, Loss=0.118]
Epoch 6/30 Training Loss: 0.1182 Training Accuracy: 82.0616 Validation Loss: 0.1489 Validation Accuracy: 78.2582
100%|██████████| 19/19 [00:03<00:00,  5.01batch/s, Epoch=7, Loss=0.112]
Epoch 7/30 Training Loss: 0.1117 Training Accuracy: 82.9647 Validation Loss: 0.1477 Validation Accuracy: 78.7136
100%|██████████| 19/19 [00:03<00:00,  4.99batch/s, Epoch=8, Loss=0.106]
Epoch 8/30 Training Loss: 0.1059 Training Accuracy: 83.9678 Validation Loss: 0.1454 Validation Accuracy: 78.9066
100%|██████████| 19/19 [00:04<00:00,  4.14batch/s, Epoch=9, Loss=0.101]
```

Precision, recall, f1 score, accuracy برای داده‌های تست:

```
best_model.eval()
y_test = []
y_score = []
y_preds = []
with torch.no_grad():
    for inputs, labels in test_dataloader:
        inputs = inputs.to(device)
        labels = labels.to(device)
        outputs = best_model(inputs)
        _, preds = torch.max(outputs, 2)
        y_test.extend(labels.tolist())
        y_score.extend(outputs.tolist())
        y_preds.extend(preds.tolist())
tag_dict = train_dataset.pos_dict
# Flatten the predicted tags and the actual tags
flat_predicted_tags, flat_actual_tags = flat_tags(y_preds, y_test)

# Calculate the confusion matrix
cm = confusion_matrix(flat_actual_tags, flat_predicted_tags)
plt.figure(figsize=(20, 10))
sns.heatmap(cm, annot=True, fmt='d', annot_kws={"size": 8})
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()
# Generate classification report
report = classification_report(flat_actual_tags, flat_predicted_tags, labels=list(tag_dict.values()), target_names=list(tag_dict.keys()))
print(report)
```

	precision	recall	f1-score	support
AJ	0.60	0.76	0.67	1674
PAD	0.00	0.00	0.00	0
NUM	0.96	0.94	0.95	511
DETe	0.57	0.91	0.70	88
POSTP	0.88	1.00	0.93	403
NUMe	0.41	0.47	0.44	19
PRO	0.93	0.86	0.89	1081
PROe	0.00	0.00	0.00	9
CL	0.49	0.50	0.49	42
Pe	0.84	0.80	0.82	351
ADV	0.90	0.85	0.87	1346
CONJe	1.00	1.00	1.00	1
N	0.68	0.78	0.72	7785
CONJ	0.98	0.96	0.97	2260
P	0.96	0.97	0.96	2256
AJe	0.44	0.28	0.34	511
INT	0.97	1.00	0.99	39
Ne	0.66	0.48	0.56	4141
RESe	0.00	0.00	0.00	4
PUNC	0.98	0.94	0.96	3283
RES	0.57	0.48	0.52	249
ADVe	0.93	0.82	0.87	34
V	0.88	0.87	0.88	4143
DET	0.85	0.91	0.88	684
micro avg	0.80	0.80	0.80	30914
macro avg	0.69	0.69	0.68	30914
weighted avg	0.80	0.80	0.79	30914

بخش ج

با استفاده از کتابخانه نصب شده یک لایه crf بالای مدل قرار دادم. در نهایت خروجی که دریافت می‌کنیم با استفاده از لایه crf که در بالای مدل قرار دادم. برای crf از کتاب‌خانه pytorch-crf استفاده کردم.

```python
class RNNPOSTaggerWithCRF(nn.Module):
    def __init__(self, vocab_size, embedding_dim, hidden_dim, output_dim, n_layers, bidirectional=True, dropout=0.2):
        super().__init__()
        self.embedding = nn.Embedding(vocab_size, embedding_dim)
        self.rnn = nn.LSTM(embedding_dim, hidden_dim, num_layers = n_layers, bidirectional = bidirectional)
        self.fc = nn.Linear(hidden_dim * 2 if bidirectional else hidden_dim, output_dim)
        self.crf = CRF(output_dim, batch_first=True)
        self.dropout = nn.Dropout(dropout)

    def forward(self, text, is_training=True):
        embedded = self.dropout(self.embedding(text))
        outputs, (hidden, cell) = self.rnn(embedded)
        outputs = self.fc(self.dropout(outputs))
        if is_training:
            return outputs
        else:
            best_tags_list = torch.tensor(self.crf.decode(outputs))
            return best_tags_list
```

Executed at 2024.05.23.19:24:17 in 8ms

هایپرپارامتر ها را تیون کردم:

```python
best_val_loss = float('inf')
best_hyperparams = None
best_model_optimization = None
best_model = None
learning_rates = [0.01]
weight_decays = [0,1e-5]
num_epochs = 5
for lr in learning_rates:
    for wd in weight_decays:
        print(f"Training with learning rate: {lr} and weight decay: {wd}")
        model = RNNPOSTaggerWithCRF(vocab_size=num_words + 5000, embedding_dim=300, hidden_dim=300, output_dim=num_classes, n_layers=2)
        model = model.to(device)
        for param in model.parameters():
            param.requires_grad = True

        model_optimization = CRFModelOptimization(model, train_dataloader, valid_dataloader, device,vocab_dict=words_dict, num_epochs=num_epochs, lr=lr, weight_decay=wd, runs_name='hyperparameter_tuning')

        model_optimization.train()
        val_loss = model_optimization.validation_losses[-1]

        if val_loss < best_val_loss:
            best_val_loss = val_loss
            best_hyperparams = (lr, wd)
            best_model_optimization = model_optimization
            best_model = model
print(f"Best hyperparameter: Learning rate: {best_hyperparams[0]}, Weight decay: {best_hyperparams[1]}")
```

```
Epoch 2/5 Training Loss: 0.4209  Validation Loss: 0.4838 Validation Accuracy: 37.5490
100%|██████████| 19/19 [00:22<00:00,  1.17s/batch, Epoch=3, Loss=0.328]
Epoch 3/5 Training Loss: 0.3281  Validation Loss: 0.3600 Validation Accuracy: 48.6400
100%|██████████| 19/19 [00:22<00:00,  1.17s/batch, Epoch=4, Loss=0.262]
Epoch 4/5 Training Loss: 0.2622  Validation Loss: 0.2946 Validation Accuracy: 57.6865
100%|██████████| 19/19 [00:22<00:00,  1.16s/batch, Epoch=5, Loss=0.206]
Epoch 5/5 Training Loss: 0.2055  Validation Loss: 0.2356 Validation Accuracy: 66.6817
Best validation accuracy: 66.68174416500814
Training with learning rate: 0.01 and weight decay: 1e-05
100%|██████████| 19/19 [00:22<00:00,  1.17s/batch, Epoch=1, Loss=0.96]
Epoch 1/5 Training Loss: 0.9598  Validation Loss: 3.3780 Validation Accuracy: 0.7629
100%|██████████| 19/19 [00:22<00:00,  1.17s/batch, Epoch=2, Loss=0.411]
Epoch 2/5 Training Loss: 0.4107  Validation Loss: 0.4739 Validation Accuracy: 38.9814
100%|██████████| 19/19 [00:22<00:00,  1.17s/batch, Epoch=3, Loss=0.33]
Epoch 3/5 Training Loss: 0.3298  Validation Loss: 0.3561 Validation Accuracy: 49.8703
100%|██████████| 19/19 [00:22<00:00,  1.19s/batch, Epoch=4, Loss=0.28]
Epoch 4/5 Training Loss: 0.2801  Validation Loss: 0.3063 Validation Accuracy: 54.6861
100%|██████████| 19/19 [00:22<00:00,  1.17s/batch, Epoch=5, Loss=0.23]
Epoch 5/5 Training Loss: 0.2297  Validation Loss: 0.2512 Validation Accuracy: 64.4985
Best validation accuracy: 64.49852240516253
Best hyperparameter: Learning rate: 0.01, Weight decay: 0
```

مدل را در نهایت آموزش دادم:

```
100%|██████████| 19/19 [00:22<00:00,  1.18s/batch, Epoch=22, Loss=0.0644]

Epoch 22/30 Training Loss: 0.0644  Validation Loss: 0.1663 Validation Accuracy: 79.0182

100%|██████████| 19/19 [00:22<00:00,  1.18s/batch, Epoch=23, Loss=0.0642]

Epoch 23/30 Training Loss: 0.0642  Validation Loss: 0.1684 Validation Accuracy: 79.0815

100%|██████████| 19/19 [00:22<00:00,  1.19s/batch, Epoch=24, Loss=0.0636]

Epoch 24/30 Training Loss: 0.0636  Validation Loss: 0.1678 Validation Accuracy: 79.1478

100%|██████████| 19/19 [00:22<00:00,  1.19s/batch, Epoch=25, Loss=0.0627]

Epoch 25/30 Training Loss: 0.0627  Validation Loss: 0.1679 Validation Accuracy: 79.0875

100%|██████████| 19/19 [00:22<00:00,  1.20s/batch, Epoch=26, Loss=0.0625]

Epoch 26/30 Training Loss: 0.0625  Validation Loss: 0.1738 Validation Accuracy: 79.3318

100%|██████████| 19/19 [00:22<00:00,  1.20s/batch, Epoch=27, Loss=0.0619]

Epoch 27/30 Training Loss: 0.0619  Validation Loss: 0.1708 Validation Accuracy: 79.2865

100%|██████████| 19/19 [00:22<00:00,  1.19s/batch, Epoch=28, Loss=0.0613]

Epoch 28/30 Training Loss: 0.0613  Validation Loss: 0.1700 Validation Accuracy: 79.2473

100%|██████████| 19/19 [00:22<00:00,  1.19s/batch, Epoch=29, Loss=0.061]

Epoch 29/30 Training Loss: 0.0610  Validation Loss: 0.1699 Validation Accuracy: 79.1116

100%|██████████| 19/19 [00:22<00:00,  1.19s/batch, Epoch=30, Loss=0.0607]

Epoch 30/30 Training Loss: 0.0607  Validation Loss: 0.1719 Validation Accuracy: 79.1237
Best validation accuracy: 79.33176527350582
```

Precision, recall, f1 score, accuracy برای داده‌های تست:

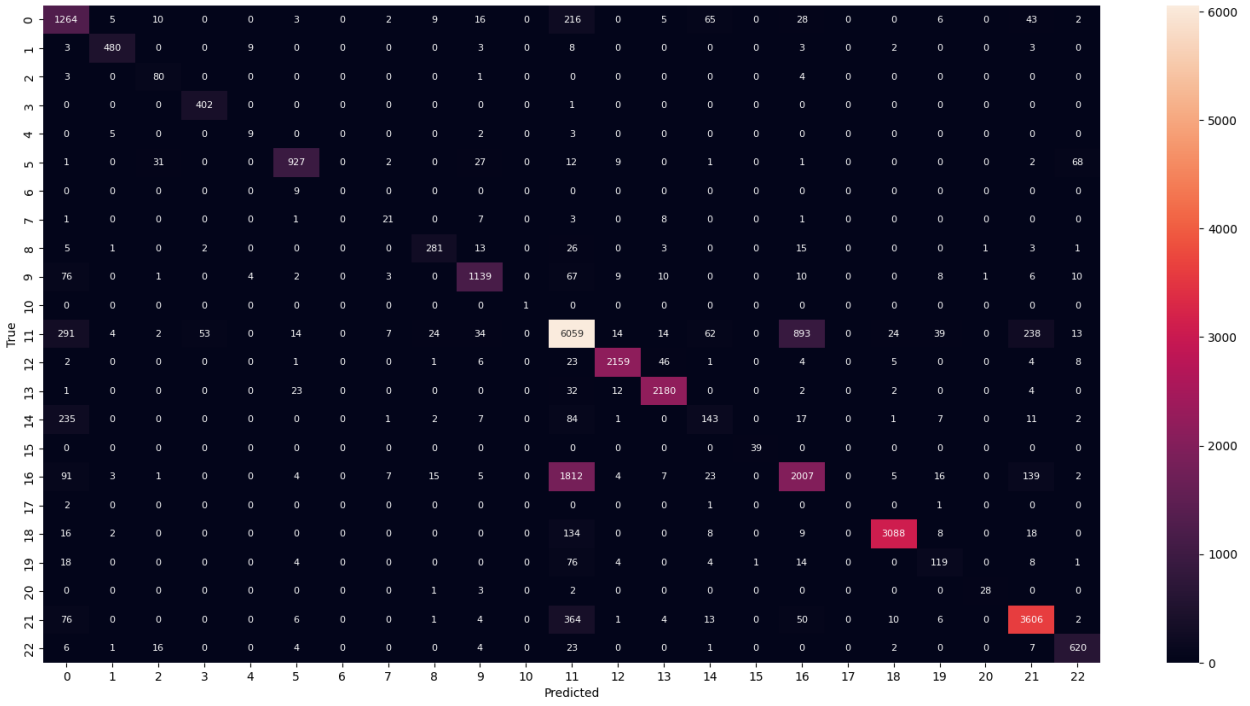
	precision	recall	f1-score	support
AJ	0.60	0.70	0.65	1768
PAD	0.00	0.00	0.00	0
NUM	0.95	0.94	0.95	589
DETe	0.61	0.96	0.75	76
POSTP	0.86	1.00	0.92	411
NUMe	0.43	0.60	0.50	20
PRO	0.91	0.85	0.88	1129
PROe	0.00	0.00	0.00	9
CL	0.44	0.38	0.41	42
Pe	0.87	0.83	0.85	441
ADV	0.81	0.87	0.84	1389
CONJe	1.00	1.00	1.00	1
N	0.67	0.75	0.71	8210
CONJ	0.96	0.96	0.96	2414
P	0.96	0.96	0.96	2465
AJe	0.46	0.31	0.37	523
INT	1.00	1.00	1.00	26
Ne	0.65	0.51	0.57	4601
RESe	0.00	0.00	0.00	1
PUNC	0.99	0.95	0.97	3659
RES	0.68	0.50	0.58	245
ADVe	0.80	0.85	0.82	41
V	0.87	0.85	0.86	4409
DET	0.83	0.88	0.85	693
micro avg	0.79	0.79	0.79	33162
macro avg	0.68	0.69	0.68	33162
weighted avg	0.79	0.79	0.79	33162

بخش د

```python
def report(best_model, tags_reverse_dict):
    best_model.eval()
    y_test = []
    y_score = []
    y_preds = []
    with torch.no_grad():
        for inputs, labels in test_dataloader:
            inputs = inputs.to(device)
            labels = labels.to(device)
            outputs = best_model(inputs)
            _, preds = torch.max(outputs, 2)
            y_test.extend(labels.tolist())
            y_score.extend(outputs.tolist())
            y_preds.extend(preds.tolist())
    ner_dict = train_dataset.pos_dict
    flat_predicted_tags, flat_actual_tags = flat_tags(y_preds, y_test)
    cm = confusion_matrix(flat_actual_tags, flat_predicted_tags)
    plt.figure(figsize=(20, 10))
    sns.heatmap(cm, annot=True, fmt='d', annot_kws={"size": 8})
    plt.xlabel('Predicted')
    plt.ylabel('True')
    plt.show()
    np.fill_diagonal(cm, 0)
    worst_errors = find_worst_errors(flat_predicted_tags, flat_actual_tags, n=5)
    print("-------- top 5 worst index in confusion matrix -----")
    for (actual, predicted), count in worst_errors:
        print(f"Actual: {tags_reverese_dict[actual]}, Predicted: {tags_reverese_dict[predicted]}, Count: {count}")
    print("-----------------------------------------------------")
    report_classification = classification_report(flat_actual_tags, flat_predicted_tags, labels=list(ner_dict.values()), target_names=list(ner_dict.keys()))
    print(report_classification)
    print(f"Accuracy: {accuracy_score(flat_predicted_tags, flat_actual_tags)}")
```

از کد بالا برای رسم بهترین ماتریس درهم‌ریختگی بهترین مدل های بخش ب و ج رسم می‌کند همچنین ۵ تا از بدترین نتیجه های هر دو مدل را نمایش می‌دهد.

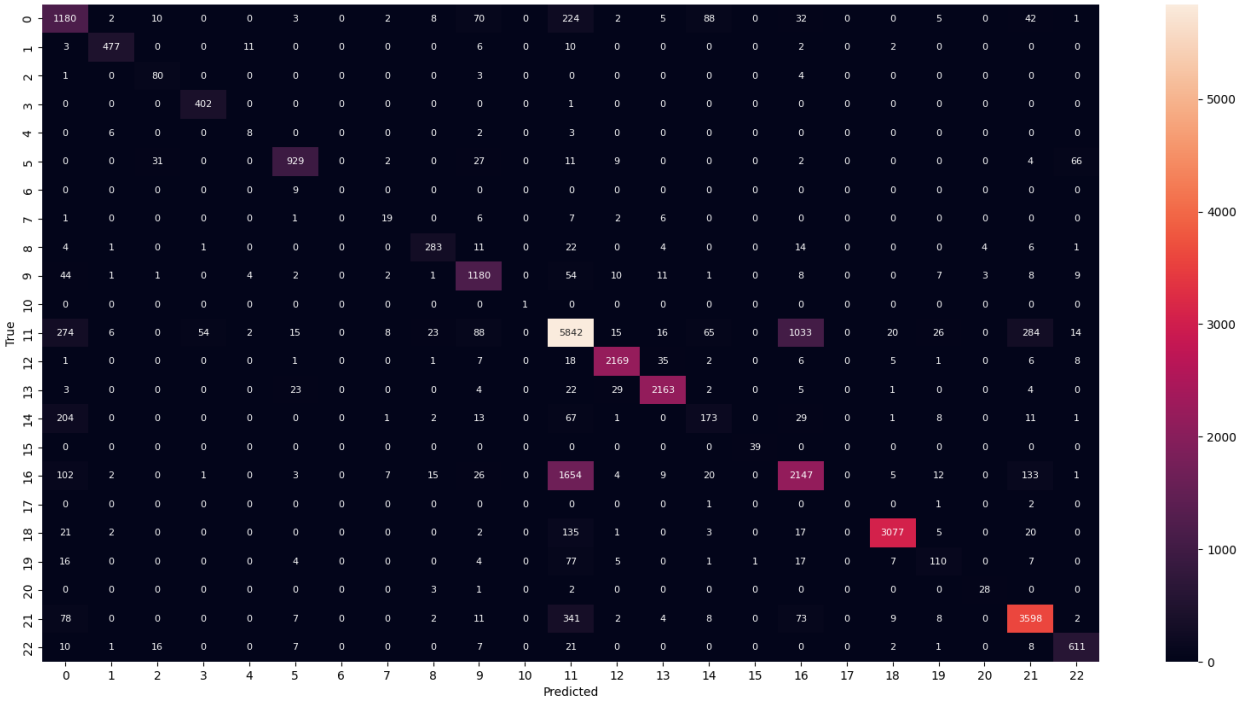
مدل بخش ب:



در لیست پراشتباه ترین موارد در ماتریس درهم‌ریختگی برای بهترین مدل بخش ب:

```
-------- top 5 worst index in confusion matrix -----
Actual: N, Predicted: Ne, Count: 1812
Actual: Ne, Predicted: N, Count: 893
Actual: N, Predicted: V, Count: 364
Actual: AJ, Predicted: N, Count: 291
Actual: V, Predicted: N, Count: 238
----------------------------------------------------
```

مدل بخش ج:



در لیست پر اشتباه ترین موارد در ماتریس درهم‌ریختگی برای مدل بخش ج:

```
-------- top 5 worst index in confusion matrix -----
Actual: N, Predicted: Ne, Count: 1654
Actual: Ne, Predicted: N, Count: 1033
Actual: N, Predicted: V, Count: 341
Actual: V, Predicted: N, Count: 284
Actual: AJ, Predicted: N, Count: 274
----------------------------------------------------
              precision    recall  f1-score   support
```

انالیز نتایج:

- هر دو مدل تقریبا عملکرد یکسانی از خود نشان دادند، نمی‌شه گفت کدوم داره بهتر عمل می‌کند چوت تفاوت ها واقعا جزئی است.

- هر دو مدل در برچسب‌های خاصی اشتباهات رایج یکسانی دارند.

	precision	recall	f1-score	support
AJ	0.60	0.76	0.67	1674
PAD	0.00	0.00	0.00	0
NUM	0.96	0.94	0.95	511
DETe	0.57	0.91	0.70	88
POSTP	0.88	1.00	0.93	403
NUMe	0.41	0.47	0.44	19
PRO	0.93	0.86	0.89	1081
PROe	0.00	0.00	0.00	9
CL	0.49	0.50	0.49	42
Pe	0.84	0.80	0.82	351
ADV	0.90	0.85	0.87	1346
CONJe	1.00	1.00	1.00	1
N	0.68	0.78	0.72	7785
CONJ	0.98	0.96	0.97	2260
P	0.96	0.97	0.96	2256
AJe	0.44	0.28	0.34	511
INT	0.97	1.00	0.99	39
Ne	0.66	0.48	0.56	4141
RESe	0.00	0.00	0.00	4
PUNC	0.98	0.94	0.96	3283
RES	0.57	0.48	0.52	249
ADVe	0.93	0.82	0.87	34
V	0.88	0.87	0.88	4143
DET	0.85	0.91	0.88	684
micro avg	0.80	0.80	0.80	30914
macro avg	0.69	0.69	0.68	30914
weighted avg	0.80	0.80	0.79	30914

	precision	recall	f1-score	support
AJ	0.60	0.70	0.65	1768
PAD	0.00	0.00	0.00	0
NUM	0.95	0.94	0.95	589
DETe	0.61	0.96	0.75	76
POSTP	0.86	1.00	0.92	411
NUMe	0.43	0.60	0.50	20
PRO	0.91	0.85	0.88	1129
PROe	0.00	0.00	0.00	9
CL	0.44	0.38	0.41	42
Pe	0.87	0.83	0.85	441
ADV	0.81	0.87	0.84	1389
CONJe	1.00	1.00	1.00	1
N	0.67	0.75	0.71	8210
CONJ	0.96	0.96	0.96	2414
P	0.96	0.96	0.96	2465
AJe	0.46	0.31	0.37	523
INT	1.00	1.00	1.00	26
Ne	0.65	0.51	0.57	4601
RESe	0.00	0.00	0.00	1
PUNC	0.99	0.95	0.97	3659
RES	0.68	0.50	0.58	245
ADVe	0.80	0.85	0.82	41
V	0.87	0.85	0.86	4409
DET	0.83	0.88	0.85	693
micro avg	0.79	0.79	0.79	33162
macro avg	0.68	0.69	0.68	33162
weighted avg	0.79	0.79	0.79	33162

- برچسب‌هایی مانند 'ADJ'و PRO و CON وPUNC و Pو V و DETبه طور مداوم توسط هر دو مدل به خوبی تشخیص داده می‌شوند، که نشان می‌دهد این دسته‌ها برای مدل‌ها آسان‌تر هستند.

```
-------- top 5 worst index in confusion matrix -----
Actual: N, Predicted: Ne, Count: 1654
Actual: Ne, Predicted: N, Count: 1033
Actual: N, Predicted: V, Count: 341
Actual: V, Predicted: N, Count: 284
Actual: AJ, Predicted: N, Count: 274
--------------------------------------------------
              precision    recall  f1-score   support
```

```
-------- top 5 worst index in confusion matrix -----
Actual: N, Predicted: Ne, Count: 1654
Actual: Ne, Predicted: N, Count: 1033
Actual: N, Predicted: V, Count: 341
Actual: V, Predicted: N, Count: 284
Actual: AJ, Predicted: N, Count: 274
--------------------------------------------------
              precision    recall  f1-score   support
```

- برچسب‌های مثلNe، N ،AJ که تعداد بالاتری دارند و خطای آن ها در عملکرد مدل بسیار حساس است نرخ خطای بالاتری دارند. چیزی که مشخص است بیش ترین موردی که مدل با آن مشکل دارد این تگ هاست درصد اشتباه آن به نسبت تعداد آن بالاست یک ایده ای می‌توان استفاده کرد، این است در تابع خطا اولویت بیش تری به این کلاس بدهیم این کار را انجام دادم اما نتایج بدتر شد و حدود ده درصد افت نتایج داشتم.

بخش ۳

بخش الف

کد های این قسمت خیلی جاهایش مثل سوال قبلیست در واقع بیش ترش از قسمت قبل برداشتم فقط اسمش را تغییر دادم برای همین اون کد ها را نمی‌ارم برای مثال کد دیتاست که قسمت گزاشتم و با کمک اون پد با داده اضافه می‌کردم.

```python
class NERDataset(Dataset):
    def __init__(self, df, words_dict=None, ner_dict=None, max_length=None):
        self.df = df
        self.words_dict = words_dict
        self.ner_dict = ner_dict
        self.max_length = max_length

    def __len__(self):
        return len(self.df)

    def __getitem__(self, idx):
        sentence = self.df.iloc[idx]['sentences']
        ner_tags = self.df.iloc[idx]['ner_labels']

        # Pad the sequences to the maximum lengths of the
        if len(sentence) < self.max_length:
            sentence += ['PAD'] * (self.max_length - len(sentence))
            ner_tags += ['PAD'] * (self.max_length - len(ner_tags))

        return sentence, ner_tags

    def collate_fn(self, batch):
        sentences, ner_tags = zip(*batch)
        sentences = [[self.words_dict[word] if self.words_dict.get(word) else 0 for word in sentence] for sentence in sentences]
        ner_tags = [[self.ner_dict[tag] for tag in ner_tag] for ner_tag in ner_tags]
        return torch.LongTensor(sentences), torch.tensor(ner_tags)
Executed at 2024.05.27 20:15:44 in 95ms
```

کد مدل استفاده شده برای ران کردن قسمت بعدی:

```python
# Define the model
class RNNNERTagger(nn.Module):
    def __init__(self, vocab_size=5000, embedding_dim=300, hidden_dim=300, output_dim=300, n_layers=2, bidirectional=False, dropout=0.5):
        super().__init__()
        self.embedding = nn.Embedding(vocab_size, embedding_dim)
        self.rnn = nn.LSTM(embedding_dim, hidden_dim, num_layers = n_layers, bidirectional = bidirectional)
        self.fc = nn.Linear(hidden_dim * 2 if bidirectional else hidden_dim, output_dim)
        self.dropout = nn.Dropout(dropout)

    def forward(self, text):
        embedded = self.dropout(self.embedding(text))
        outputs, (hidden, cell) = self.rnn(embedded)
        predictions = self.fc(self.dropout(outputs))
        return predictions
Executed at 2024.05.27 20:13:57 in 96ms
```

در مدل بالا یک جمله به آن می‌دهیم و در نهایت تگ های آن را خروجی می‌دهد. البته این قبل آموزش است یادم رفت خروجی بگیرم و بعد در صورت سوال متوجه شدم که می‌خواد اینو خروجی بگیرم با وزن های رندم یک مدل ساختم فقط خروجی گرفتم.

```python
def predict_sentence(sentence, model, words_dict, tags_reverse_dict):
    sentence = preprocess_text(sentence)
    sentence_input = [words_dict[word] if words_dict.get(word) else 0 for word in sentence]
    sentence_input = torch.LongTensor(sentence_input).unsqueeze(0).to(device)
    model.eval()
    with torch.no_grad():
        outputs = model(sentence_input)
        _, preds = torch.max(outputs, 2)
    predicted_tags = [tags_reverse_dict[tag] for tag in preds[0].tolist()]
    return predicted_tags

sentence = "اصغری دکتر"
predicted_tags = predict_sentence(sentence, best_model, words_dict, tags_reverese_dict)
print(predicted_tags)
```

```
['PAD', 'PAD']
```

معماری شبکه را با استفاده کد زیر نمایش دادم:

```python
summary(
    RNNNERTagger(vocab_size=num_words, embedding_dim=200, hidden_dim=200, output_dim=num_classes, n_layers=4),
    (23,),
    dtypes=[torch.long],
    branching=False,
    verbose=2,
    col_width=16,
    col_names=["kernel_size", "output_size", "num_params", "mult_adds"],
)
```

```
==========================================================================================================================
Layer (type:depth-idx)                   Kernel Shape     Output Shape     Param #          Mult-Adds
==========================================================================================================================
Embedding: 1-1                           [200, 15014]     [-1, 23, 200]    3,002,800        3,002,800
Dropout: 1-2                             --               [-1, 23, 200]    --               --
LSTM: 1-3                                --               [-1, 23, 200]    1,286,400        1,280,000
  weight_ih_l0                           [800, 200]
  weight_hh_l0                           [800, 200]
  weight_ih_l1                           [800, 200]
  weight_hh_l1                           [800, 200]
  weight_ih_l2                           [800, 200]
  weight_hh_l2                           [800, 200]
  weight_ih_l3                           [800, 200]
  weight_hh_l3                           [800, 200]
Dropout: 1-4                             --               [-1, 23, 200]    --               --
Linear: 1-5                              [200, 14]        [-1, 23, 14]     2,814            2,800
==========================================================================================================================
Total params: 4,292,014
Trainable params: 4,292,014
Non-trainable params: 0
Total mult-adds (M): 4.29
==========================================================================================================================
Input size (MB): 0.00
Forward/backward pass size (MB): 0.07
Params size (MB): 16.37
Estimated Total Size (MB): 16.45
==========================================================================================================================
```

بخش ب

با استفاده از داده های validation هایپرپارامترها را tune کردم.

```python
best_val_loss = float('inf')
best_model_optimization = None
best_model = None
best_hyperparams = None
learning_rates = [0.01]
weight_decays = [0,1e-5]
num_epochs = 5
for lr in learning_rates:
    for wd in weight_decays:
        print(f"Training with learning rate: {lr} and weight decay: {wd}")
        model = RNNNERTagger(vocab_size=num_words+1, embedding_dim=300, hidden_dim=300, output_dim=num_classes, n_layers=2)
        model = model.to(device)
        for param in model.parameters():
            param.requires_grad = True

        model_optimization = ModelOptimization(model, train_dataloader, valid_dataloader, device,vocab_dict=words_dict, num_epochs=num_epochs, lr=lr, weight_decay=wd, runs_name='hyperparameter_tuning')

        model_optimization.train()
        val_loss = model_optimization.validation_losses[-1]

        if val_loss < best_val_loss:
            best_val_loss = val_loss
            best_hyperparams = (lr, wd)
            best_model_optimization = model_optimization
            best_model = model
print(f"Best hyperparameter: Learning rate: {best_hyperparams[0]}, Weight decay: {best_hyperparams[1]}")
```

Executed at 2024.05.27 20:31:43 in 43s 940ms

```
Epoch 5/5 Training Loss: 0.0613 Training Accuracy: 89.1832 Validation Loss: 0.0502 Validation Accuracy: 89.7120
Best validation accuracy: 89.71201688696789
Training with learning rate: 0.01 and weight decay: 1e-05

100%|██████████| 14/14 [00:07<00:00,  1.76batch/s, Epoch=1, Loss=0.346]

Epoch 1/5 Training Loss: 0.3460 Training Accuracy: 71.4492 Validation Loss: 0.0727 Validation Accuracy: 89.2798

100%|██████████| 14/14 [00:07<00:00,  1.90batch/s, Epoch=2, Loss=0.0772]

Epoch 2/5 Training Loss: 0.0772 Training Accuracy: 89.0080 Validation Loss: 0.0682 Validation Accuracy: 89.3049

100%|██████████| 14/14 [00:07<00:00,  1.75batch/s, Epoch=3, Loss=0.0737]

Epoch 3/5 Training Loss: 0.0737 Training Accuracy: 88.9949 Validation Loss: 0.0656 Validation Accuracy: 89.3275

100%|██████████| 14/14 [00:07<00:00,  1.75batch/s, Epoch=4, Loss=0.0702]

Epoch 4/5 Training Loss: 0.0702 Training Accuracy: 89.0028 Validation Loss: 0.0625 Validation Accuracy: 89.3451

100%|██████████| 14/14 [00:07<00:00,  1.79batch/s, Epoch=5, Loss=0.0666]

Epoch 5/5 Training Loss: 0.0666 Training Accuracy: 89.0326 Validation Loss: 0.0589 Validation Accuracy: 89.3527
Best validation accuracy: 89.35266623109011
Best hyperparameter: Learning rate: 0.01, Weight decay: 0
```

اموزش مدل در نهایت:

```
100%|██████████| 14/14 [00:08<00:00,  1.62batch/s, Epoch=5, Loss=0.0429]

Epoch 5/10 Training Loss: 0.0429 Training Accuracy: 90.4538 Validation Loss: 0.0371 Validation Accuracy: 91.2726

100%|██████████| 14/14 [00:08<00:00,  1.67batch/s, Epoch=6, Loss=0.0414]

Epoch 6/10 Training Loss: 0.0414 Training Accuracy: 90.7648 Validation Loss: 0.0358 Validation Accuracy: 91.4057

100%|██████████| 14/14 [00:08<00:00,  1.70batch/s, Epoch=7, Loss=0.04]

Epoch 7/10 Training Loss: 0.0400 Training Accuracy: 90.9926 Validation Loss: 0.0343 Validation Accuracy: 92.0214

100%|██████████| 14/14 [00:08<00:00,  1.62batch/s, Epoch=8, Loss=0.0388]

Epoch 8/10 Training Loss: 0.0388 Training Accuracy: 91.2116 Validation Loss: 0.0332 Validation Accuracy: 92.3782

100%|██████████| 14/14 [00:08<00:00,  1.65batch/s, Epoch=9, Loss=0.0375]

Epoch 9/10 Training Loss: 0.0375 Training Accuracy: 91.4886 Validation Loss: 0.0322 Validation Accuracy: 92.6019

100%|██████████| 14/14 [00:08<00:00,  1.68batch/s, Epoch=10, Loss=0.0364]

Epoch 10/10 Training Loss: 0.0364 Training Accuracy: 91.7555 Validation Loss: 0.0311 Validation Accuracy: 92.6723
Best validation accuracy: 92.67226215007287
```

Precision, recall, f1 score, accuracy برای داده‌های تست:

```python
best_model.eval()
y_test = []
y_score = []
y_preds = []
with torch.no_grad():
    for inputs, labels in test_dataloader:
        inputs = inputs.to(device)
        labels = labels.to(device)
        outputs = best_model(inputs)
        _, preds = torch.max(outputs, 2)
        y_test.extend(labels.tolist())
        y_score.extend(outputs.tolist())
        y_preds.extend(preds.tolist())
ner_dict = train_dataset.ner_dict
# Flatten the predicted tags and the actual tags
flat_predicted_tags, flat_actual_tags = flat_tags(y_preds, y_test)
flat_actual_tags_ = [tags_reverese_dict[i] for i in flat_actual_tags]
flat_predicted_tags_ = [tags_reverese_dict[i] for i in flat_predicted_tags]
# Calculate the confusion matrix
cm = confusion_matrix(flat_actual_tags, flat_predicted_tags)
plt.figure(figsize=(20, 10))
sns.heatmap(cm, annot=True, fmt='d', annot_kws={"size": 8})
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()
# Generate classification report
report = classification_report(flat_actual_tags, flat_predicted_tags, labels=list(ner_dict.values()), target_names=list(ner_dict.keys()))
print(report)
print(f"Accuracy: {accuracy_score(flat_predicted_tags, flat_actual_tags)}")
```

	precision	recall	f1-score	support
I-event	0.00	0.00	0.00	327
PAD	0.00	0.00	0.00	0
B-loc	0.47	0.64	0.54	613
I-loc	0.00	0.00	0.00	177
B-pro	0.00	0.00	0.00	144
I-pers	0.23	0.04	0.06	406
B-fac	0.00	0.00	0.00	67
I-pro	0.00	0.00	0.00	102
B-pers	0.55	0.89	0.68	635
B-org	0.62	0.49	0.55	898
B-event	0.00	0.00	0.00	87
I-fac	0.00	0.00	0.00	150
O	0.96	0.99	0.98	37218
I-org	0.50	0.32	0.39	1042
accuracy			0.92	41866
macro avg	0.24	0.24	0.23	41866
weighted avg	0.90	0.92	0.91	41866

Accuracy: 0.9239717192948932

بخش ج

با استفاده از کتابخانه نصب شده یک لایه crf بالای مدل قرار دادم.

```python
class RNNNERTaggerWithCRF(nn.Module):
    def __init__(self, vocab_size, embedding_dim, hidden_dim, output_dim, n_layers, bidirectional=True, dropout=0.2):
        super().__init__()
        self.embedding = nn.Embedding(vocab_size, embedding_dim,)
        self.rnn = nn.LSTM(embedding_dim, hidden_dim, num_layers = n_layers, bidirectional = bidirectional)
        self.fc = nn.Linear(hidden_dim * 2 if bidirectional else hidden_dim, output_dim)
        self.crf = CRF(output_dim, batch_first=True)
        self.dropout = nn.Dropout(dropout)

    def forward(self, text, is_training=True):
        embedded = self.dropout(self.embedding(text))
        outputs, (hidden, cell) = self.rnn(embedded)
        outputs = self.fc(self.dropout(outputs))
        if is_training:
            return outputs
        else:
            best_tags_list = torch.tensor(self.crf.decode(outputs))
            return best_tags_list
```

Executed at 2024.05.27 20:52:23 in 100ms

```
==========================================================================================================================
Layer (type:depth-idx)                   Kernel Shape     Output Shape     Param #          Mult-Adds
==========================================================================================================================
Embedding: 1-1                           [300, 15014]     [-1, 24, 300]    4,504,200        4,504,200
Dropout: 1-2                             --               [-1, 24, 300]    --               --
LSTM: 1-3                                --               [-1, 24, 600]    3,609,600        3,600,000
  weight_ih_l0                           [1200, 300]
  weight_hh_l0                           [1200, 300]
  weight_ih_l0_reverse                   [1200, 300]
  weight_hh_l0_reverse                   [1200, 300]
  weight_ih_l1                           [1200, 600]
  weight_hh_l1                           [1200, 300]
  weight_ih_l1_reverse                   [1200, 600]
  weight_hh_l1_reverse                   [1200, 300]
Dropout: 1-4                             --               [-1, 24, 600]    --               --
Linear: 1-5                              [600, 14]        [-1, 24, 14]     8,414            8,400
==========================================================================================================================
Total params: 8,122,214
Trainable params: 8,122,214
Non-trainable params: 0
Total mult-adds (M): 8.11
==========================================================================================================================
Input size (MB): 0.00
Forward/backward pass size (MB): 0.17
Params size (MB): 30.98
Estimated Total Size (MB): 31.15
```

هایپرپارامتر ها را تیون کردم:

```
Training with learning rate: 0.001 and weight decay: 1e-05

100%|██████████| 14/14 [00:51<00:00,  3.69s/batch, Epoch=1, Loss=0.467]

Epoch 1/5 Training Loss: 0.4673  Validation Loss: 2.6879 Validation Accuracy: 12.9969

100%|██████████| 14/14 [00:52<00:00,  3.73s/batch, Epoch=2, Loss=0.0989]

Epoch 2/5 Training Loss: 0.0989  Validation Loss: 0.1102 Validation Accuracy: 85.7265

100%|██████████| 14/14 [00:52<00:00,  3.77s/batch, Epoch=3, Loss=0.0767]

Epoch 3/5 Training Loss: 0.0767  Validation Loss: 0.0754 Validation Accuracy: 89.4180

100%|██████████| 14/14 [00:53<00:00,  3.82s/batch, Epoch=4, Loss=0.0726]

Epoch 4/5 Training Loss: 0.0726  Validation Loss: 0.0695 Validation Accuracy: 89.4306

100%|██████████| 14/14 [00:51<00:00,  3.70s/batch, Epoch=5, Loss=0.0711]

Epoch 5/5 Training Loss: 0.0711  Validation Loss: 0.0679 Validation Accuracy: 89.4306
Best validation accuracy: 89.43056742222446
Best hyperparameter: Learning rate: 0.01, Weight decay: 0
```

مدل را در نهایت آموزش دادم:

```
Epoch 4/10 Training Loss: 0.0416  Validation Loss: 0.0392 Validation Accuracy: 91.2977

100%|██████████| 14/14 [00:51<00:00,  3.70s/batch, Epoch=5, Loss=0.0393]

Epoch 5/10 Training Loss: 0.0393  Validation Loss: 0.0367 Validation Accuracy: 91.7274

100%|██████████| 14/14 [00:51<00:00,  3.71s/batch, Epoch=6, Loss=0.0369]

Epoch 6/10 Training Loss: 0.0369  Validation Loss: 0.0345 Validation Accuracy: 92.3531

100%|██████████| 14/14 [00:52<00:00,  3.74s/batch, Epoch=7, Loss=0.0348]

Epoch 7/10 Training Loss: 0.0348  Validation Loss: 0.0322 Validation Accuracy: 92.6748

100%|██████████| 14/14 [00:51<00:00,  3.71s/batch, Epoch=8, Loss=0.0334]

Epoch 8/10 Training Loss: 0.0334  Validation Loss: 0.0308 Validation Accuracy: 92.8708

100%|██████████| 14/14 [00:51<00:00,  3.67s/batch, Epoch=9, Loss=0.0317]

Epoch 9/10 Training Loss: 0.0317  Validation Loss: 0.0296 Validation Accuracy: 93.1849

100%|██████████| 14/14 [00:52<00:00,  3.72s/batch, Epoch=10, Loss=0.0304]

Epoch 10/10 Training Loss: 0.0304  Validation Loss: 0.0284 Validation Accuracy: 93.3935
Best validation accuracy: 93.39347640347792
```

Precision, recall, f1 score, accuracy برای داده‌های تست:

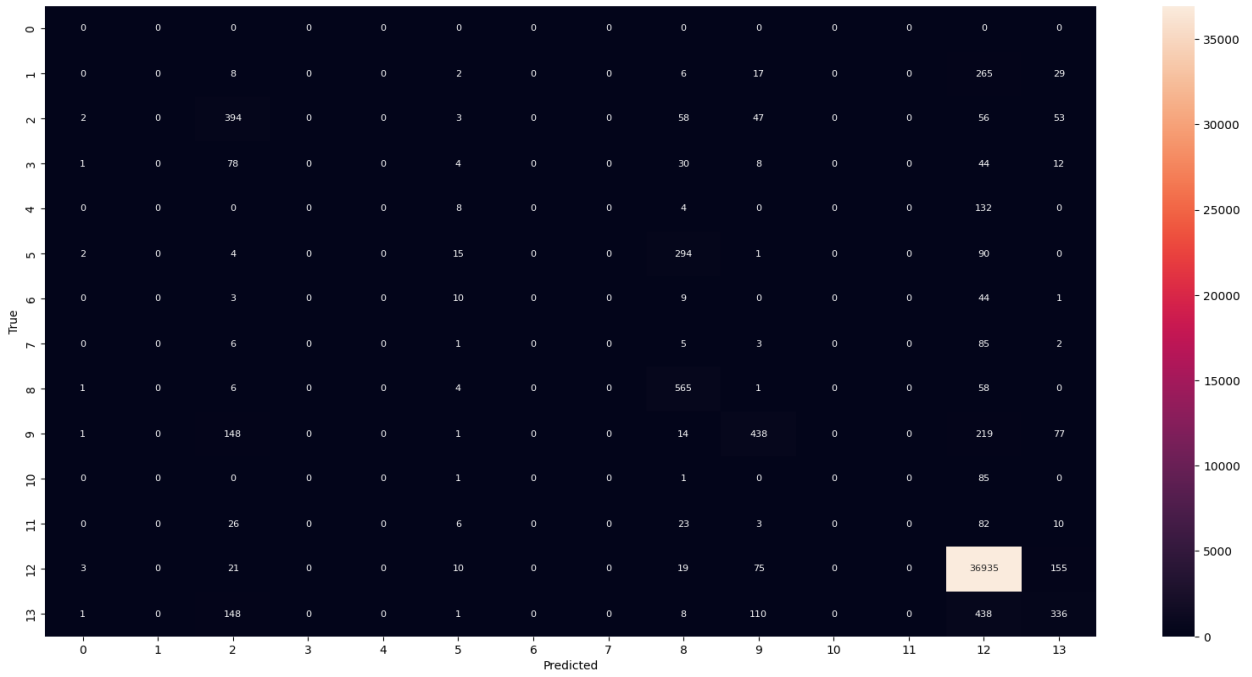
	precision	recall	f1-score	support
I-event	0.57	0.06	0.12	327
PAD	0.00	0.00	0.00	0
B-loc	0.46	0.81	0.59	613
I-loc	0.50	0.01	0.01	177
B-pro	0.86	0.04	0.08	144
I-pers	0.62	0.63	0.63	406
B-fac	0.00	0.00	0.00	67
I-pro	0.00	0.00	0.00	102
B-pers	0.82	0.81	0.81	635
B-org	0.71	0.54	0.61	898
B-event	0.25	0.01	0.02	87
I-fac	0.00	0.00	0.00	150
O	0.96	0.99	0.98	37218
I-org	0.63	0.33	0.43	1042
accuracy			0.93	41866
macro avg	0.46	0.30	0.31	41866
weighted avg	0.92	0.93	0.92	41866

بخش د

```python
def report(best_model, tags_reverse_dict):
    best_model.eval()
    y_test = []
    y_score = []
    y_preds = []
    with torch.no_grad():
        for inputs, labels in test_dataloader:
            inputs = inputs.to(device)
            labels = labels.to(device)
            outputs = best_model(inputs)
            _, preds = torch.max(outputs, 2)
            y_test.extend(labels.tolist())
            y_score.extend(outputs.tolist())
            y_preds.extend(preds.tolist())
    ner_dict = train_dataset.ner_dict
    flat_predicted_tags, flat_actual_tags = flat_tags(y_preds, y_test)
    cm = confusion_matrix(flat_actual_tags, flat_predicted_tags)
    plt.figure(figsize=(20, 10))
    sns.heatmap(cm, annot=True, fmt='d', annot_kws={"size": 8})
    plt.xlabel('Predicted')
    plt.ylabel('True')
    plt.show()
    np.fill_diagonal(cm, 0)
    worst_errors = find_worst_errors(flat_predicted_tags, flat_actual_tags, n=5)
    print("-------- top 5 worst index in confusion matrix -----")
    for (actual, predicted), count in worst_errors:
        print(f"Actual: {tags_reverese_dict[actual]}, Predicted: {tags_reverese_dict[predicted]}, Count: {count}")
    print("-------------------------------------------------")
    report_classification = classification_report(flat_actual_tags, flat_predicted_tags, labels=list(ner_dict.values()), target_names=list(ner_dict.keys()))
    print(report_classification)
    print(f"Accuracy: {accuracy_score(flat_predicted_tags, flat_actual_tags)}")
```

از کد بالا برای رسم بهترین ماتریس درهمریختگی بهترین مدل های بخش ب و ج رسم می‌کند همچنین ۵ تا از بدترین نتیجه های هر دو مدل را نمایش می‌دهد.

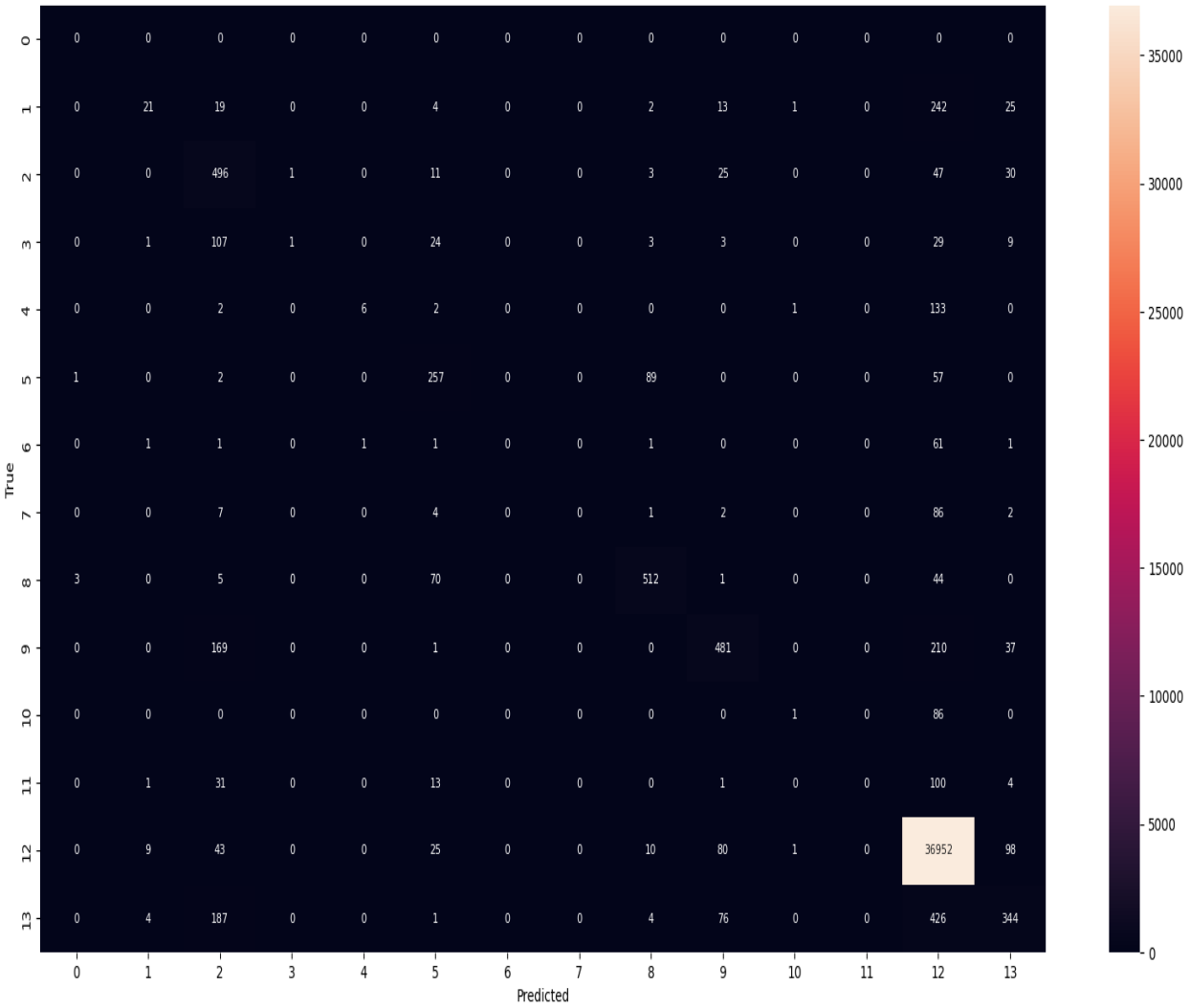
مدل بخش ب:



در لیست پراشتباه ترین موارد در ماتریس درهم‌ریختگی برای بهترین مدل بخش ب:

```
-------- top 5 worst index in confusion matrix -----
Actual: O, Predicted: I-org, Count: 438
Actual: B-pers, Predicted: I-pers, Count: 294
Actual: O, Predicted: I-event, Count: 265
Actual: O, Predicted: B-org, Count: 219
Actual: I-org, Predicted: O, Count: 155
----------------------------------------------------
```

مدل بخش ج:



در لیست پراشتباه ترین موارد در ماتریس درهم‌ریختگی برای بهترین مدل بخش ج:

```
-------- top 5 worst index in confusion matrix -----
Actual: O, Predicted: I-org, Count: 426
Actual: O, Predicted: I-event, Count: 242
Actual: O, Predicted: B-org, Count: 210
Actual: B-loc, Predicted: I-org, Count: 187
Actual: B-loc, Predicted: B-org, Count: 169
-----------------------------------------------------
```

انالیز نتایج:

- مدل CRF-LSTM عملکرد بهتری نشان می‌دهد، با اشتباهات کمتر و نرخ تشخیص درست بیشتر این را با مقایسه نتیجه به دست آمده از نتایج در سطح entity متوجه شدم.

خروجی در سطح emtity برای مدل بخش ب:

```
Entity-level metrics:
Precision: 0.3237
Recall: 0.3433
F1-score: 0.3332
Accuracy: 0.1999
```

خروجی در سطح emtity برای مدل بخش ج:

```
Entity-level metrics:
Precision: 0.4373
Recall: 0.4325
F1-score: 0.4349
Accuracy: 0.2779
```

-

هر دو مدل در برچسب‌های خاصی اشتباهات رایج یکسانی دارند.

	precision	recall	f1-score	support
I-event	0.00	0.00	0.00	327
PAD	0.00	0.00	0.00	0
B-loc	0.47	0.64	0.54	613
I-loc	0.00	0.00	0.00	177
B-pro	0.00	0.00	0.00	144
I-pers	0.23	0.04	0.06	406
B-fac	0.00	0.00	0.00	67
I-pro	0.00	0.00	0.00	102
B-pers	0.55	0.89	0.68	635
B-org	0.62	0.49	0.55	898
B-event	0.00	0.00	0.00	87
I-fac	0.00	0.00	0.00	150
O	0.96	0.99	0.98	37218
I-org	0.50	0.32	0.39	1042
accuracy			0.92	41866
macro avg	0.24	0.24	0.23	41866
weighted avg	0.90	0.92	0.91	41866

Accuracy: 0.9239717192948932

	precision	recall	f1-score	support
I-event	0.00	0.00	0.00	327
PAD	0.00	0.00	0.00	0
B-loc	0.47	0.64	0.54	613
I-loc	0.00	0.00	0.00	177
B-pro	0.00	0.00	0.00	144
I-pers	0.23	0.04	0.06	406
B-fac	0.00	0.00	0.00	67
I-pro	0.00	0.00	0.00	102
B-pers	0.55	0.89	0.68	635
B-org	0.62	0.49	0.55	898
B-event	0.00	0.00	0.00	87
I-fac	0.00	0.00	0.00	150
O	0.96	0.99	0.98	37218
I-org	0.50	0.32	0.39	1042
accuracy			0.92	41866
macro avg	0.24	0.24	0.23	41866
weighted avg	0.90	0.92	0.91	41866

Accuracy: 0.9239717192948932

- برچسب‌هایی مانند O , B-Pers , B-org به طور مداوم توسط هر دو مدل به طور متوسط به نظر بهتر تشخیص داده می‌شوند، که نشان می‌دهد این دسته‌ها برای مدل‌ها آسان‌تر هستند.

```
-------- top 5 worst index in confusion matrix -----
Actual: O, Predicted: I-org, Count: 426
Actual: O, Predicted: I-event, Count: 242
Actual: O, Predicted: B-org, Count: 210
Actual: B-loc, Predicted: I-org, Count: 187
Actual: B-loc, Predicted: B-org, Count: 169
--------------------------------------------------
```


```
-------- top 5 worst index in confusion matrix -----
Actual: O, Predicted: I-org, Count: 438
Actual: B-pers, Predicted: I-pers, Count: 294
Actual: O, Predicted: I-event, Count: 265
Actual: O, Predicted: B-org, Count: 219
Actual: I-org, Predicted: O, Count: 155
--------------------------------------------------
```


- برچسب‌های که تعداد بالاتری دارند و خطای آن ها در عملکرد مدل بسیار حساس است نرخ خطای بالاتری دارند. چیزی که مشخص است بیش ترین موردی که مدل با آن مشکل دارد این تگ هاست درصد اشتباه آن به نسبت تعداد آن بالاست.

- چیزی که مشخص است تگ O به علت تعداد زیاد باعث می‌شود عملکردی دروغین دریافت کنیم که شاید دقت آن بالا باشد اما نباید به آن توجه کرد. شاید بهتر باشد از ارزیابی سطح entity برای خروجی اخر این سیستم استفاده کرد.

بخش ۴

در تسک NER، ارزیابی می‌تواند در دو سطح مختلف انجام شود: سطح token و سطح level. هر دو سطح دیدگاه‌های متفاوتی از عملکرد مدل NER ارائه می‌دهند.

سطح token

در سطح token، ارزیابی هر token کلمه را به صورت مجزا در نظر می‌گیرد. این به این معنی است که برای هر کلمه در متن، برچسب پیش‌بینی‌شده با برچسب واقعی مقایسه می‌شود.

معیارها:

- دقت Accuracy: نسبت tokenهای درست پیش‌بینی‌شده به کل تعداد tokenها.

- دقت، بازخوانی، امتیاز F۱: بر اساس تعداد tokenهای درست پیش‌بینی‌شده برای هر برچسب محاسبه می‌شود.

با استفاده تابع پایین در ارزیابی در سطح توکن انجام دادم.

```python
def calculate_token_metrics(actual_tags, predicted_tags):
    assert len(actual_tags) == len(predicted_tags), "Length of actual and predicted tags must be the same"

    correct = 0
    total = len(actual_tags)

    # Initialize counts for precision, recall, and F1-score
    true_positive = 0
    false_positive = 0
    false_negative = 0

    for actual, predicted in zip(actual_tags, predicted_tags):
        if actual == predicted:
            correct += 1
            if actual != 'O':
                true_positive += 1
        else:
            if predicted != 'O':
                false_positive += 1
            if actual != 'O':
                false_negative += 1

    # Calculate metrics
    accuracy = correct / total
    precision = true_positive / (true_positive + false_positive) if (true_positive + false_positive) > 0 else 0
    recall = true_positive / (true_positive + false_negative) if (true_positive + false_negative) > 0 else 0
    f1_score = 2 * precision * recall / (precision + recall) if (precision + recall) > 0 else 0

    return accuracy, precision, recall, f1_score
```

نتایج برای مدل قسمت ب:

```
Token-level metrics:
Accuracy: 0.9240
Precision: 0.5245
Recall: 0.3761
F1-score: 0.4380
```

نتایج برای قسمت ج:

```
Token-level metrics:
Accuracy: 0.9332
Precision: 0.6245
Recall: 0.4559
F1-score: 0.5270
```

سطح entity

در سطح entity، ارزیابی کل entityها را به صورت یک واحد در نظر می‌گیرد. یک entity به درستی تشخیص داده می‌شود فقط اگر همه tokenهای تشکیل‌دهنده آن به درستی برچسب‌گذاری شده باشند. این روش سخت‌گیرانه‌تر است زیرا یک پیش‌بینی ناقص که فقط برخی tokenهای entity به درستی تشخیص داده شده‌اند به عنوان موفقیت در نظر گرفته نمی‌شود.

معیارها:

- دقت، بازخوانی، امتیاز F۱: بر اساس تعداد entityهای درست پیش‌بینی‌شده محاسبه می‌شود. یک entity به درستی پیش‌بینی‌شده در نظر گرفته می‌شود اگر و فقط اگر همه tokenهای آن به درستی برچسب‌گذاری شده باشند.

با استفاده از توابع زیر این بخش پیاده‌سازی کردم:

تابع زیر یک entity را به صورت کامل برای ما استخراج می‌کند.

```python

def extract_entities(tags):
    entities = []
    current_entity = []
    for idx, tag in enumerate(tags):
        if tag.startswith('B-'):
            if current_entity:
                entities.append(tuple(current_entity))
                current_entity = []
            current_entity = [tag[2:], idx, idx]
        elif tag.startswith('I-') and current_entity:
            if tag[2:] == current_entity[0]:
                current_entity[2] = idx
            else:
                entities.append(tuple(current_entity))
                current_entity = [tag[2:], idx, idx]
        else:
            if current_entity:
                entities.append(tuple(current_entity))
                current_entity = []
    if current_entity:
        entities.append(tuple(current_entity))
    return entities
```

در نهایت با تابع زیر تمام entity ها را استخراج می‌کنیم و عملکرد را در این سطح بیان می‌کنیم.

```python
def calculate_entity_metrics(actual_tags, predicted_tags):
    actual_entities = extract_entities(actual_tags)
    predicted_entities = extract_entities(predicted_tags)

    true_positive = 0
    false_positive = 0
    false_negative = 0

    matched_predicted = set()

    for entity in actual_entities:
        if entity in predicted_entities:
            true_positive += 1
            matched_predicted.add(entity)
        else:
            false_negative += 1

    for entity in predicted_entities:
        if entity not in matched_predicted:
            false_positive += 1

    precision = true_positive / (true_positive + false_positive) if (true_positive + false_positive) > 0 else 0
    recall = true_positive / (true_positive + false_negative) if (true_positive + false_negative) > 0 else 0
    f1_score = 2 * precision * recall / (precision + recall) if (precision + recall) > 0 else 0

    # Calculate accuracy
    total_entities = len(actual_entities) + len(predicted_entities) - true_positive
    accuracy = true_positive / total_entities if total_entities > 0 else 0

    return precision, recall, f1_score, accuracy
```

خروجی در سطح emtity برای مدل بخش ب:

```
Entity-level metrics:
Precision: 0.3237
Recall: 0.3433
F1-score: 0.3332
Accuracy: 0.1999
```

خروجی در سطح emtity برای مدل بخش ج:

```
Entity-level metrics:
Precision: 0.4373
Recall: 0.4325
F1-score: 0.4349
Accuracy: 0.2779
```

که مشاهده می‌شود با crf نتایج به طور قابل توجهی بهتر شده.

- سطح token: tokenهای جداگانه را ارزیابی می‌کند، منجر به امتیازات بالاتر می‌شود اگر اکثر tokenها به درستی برچسب‌گذاری شده باشند، حتی اگر entityها به طور کامل تشخیص داده نشوند.

- سطح entity: کل entityها را ارزیابی می‌کند، معیار سخت‌گیرانه‌تر و اغلب واقع‌بینانه‌تری از عملکرد مدل در کاربردهای دنیای واقعی ارائه می‌دهد.

- موارد استفاده:

- سطح token: مفید برای درک عملکرد مدل در مقیاس دقیق‌تر، به ویژه در طول فرایند آموزش.

- سطح entity: برای ارزیابی انتها به انتهای سیستم‌های NER معنی‌دارتر است، زیرا توانایی مدل در تشخیص کامل entityها را منعکس می‌کند.