به نام خداوند جان و خرد

دانشگاه تهران

دانشکدگان فنی

دانشکده مهندسی برق و کامپیوتر

پردازش زبان طبیعی

تمرین شماره ۳

نام و نام خانوادگی: علی خرم فر

شماره دانشجویی: ۸۱۰۱۰۲۱۲۹

اردیبهشت‌ماه ۱۴۰۲

فهرست مطالب

۱_ مقدمه

در این تمرین از محیط کولب استفاده شده است. متاسفانه باتوجه به محدودیت GPU دقت برخی مدل‌ها خصوصا سوال ۴ بالا نیست و این مورد باتوجه به اینکه دقت بالا مورد نیاز این تمرین نیست مد نظر قرار گرفت.

۲_پاسخ سوال اول

ابتدا داده‌های ارائه شده را بارگذاری می‌کنیم:

```
file_path = '/content/gdrive/MyDrive/Colab Notebooks/NLP CA3/train.json'

with open(file_path, 'r') as file:
    train_data = json.load(file)

file_path = '/content/gdrive/MyDrive/Colab Notebooks/NLP CA3/valid.json'

with open(file_path, 'r') as file:
    valid_data = json.load(file)

file_path = '/content/gdrive/MyDrive/Colab Notebooks/NLP CA3/test.json'

with open(file_path, 'r') as file:
    test_data = json.load(file)
```

در این مسئله ۳ فایل داده برای آموزش، ارزیابی و آزمون وجود دارد. همانطور که در فایل تمرین هم اشاره شد هر داده ۴ فیلد دارد:

۱.text : شامل جملات است که برای هر جمله لیستی از توکنهای آن را دارد.

۲.verb_index : مکان قرارگیری فعل در جمله است. توجه کنید که اولین کلمه در مکان صفرم در نظر گرفته میشود.

۳. srl_label: نشان دهنده برچسب هر توکن در مسئله SRL است.

۴. word_indices : ایندکس توکنها را نشان میدهد.

در این مسئله قصد داریم که لیبل‌های مورد نظر را به کمک روش‌های مبتنی بر شبکه‌های RNN تشخیص دهیم.

۱-۲_ پیش پردازش و آماده‌سازی داده‌ها

تبدیل برچسب به عدد

برای حل این مشکل یک تابع نوشته شد. تمام برچسب‌ها به مقدار عددی متناظرشان تبدیل شدند.

```
labels = {
    'O': 0, 'B-ARG0': 1, 'I-ARG0': 2, 'B-ARG1': 3, 'I-ARG1': 4, 'B-ARG2': 5, 'I-ARG2': 6, 'B-ARGM-LOC': 7, 'I-ARGM-LOC': 8, 'B-ARGM-TMP': 9, 'I-ARGM-TMP': 10
}

def convert_labels(srl_frames):
    converted_srl_frames = []
    for frame in srl_frames:
        converted_frame = [labels[label] for label in frame]
        converted_srl_frames.append(converted_frame)
    return converted_srl_frames


train_data['srl_frames'] = convert_labels(train_data['srl_frames'])
valid_data['srl_frames'] = convert_labels(valid_data['srl_frames'])

print(train_data['srl_frames'][1])
```

```
[0, 0, 0, 0, 0, 3, 0, 0, 0, 0, 0, 0]
```

برای نمونه در شکل بالا دومین عنصر از داده‌های آموزشی را خروجی کردیم که مشاهده می‌شود کلمه ششم آن لیبل ۳ را دارد که معادل B-ARG1 است.

یکسان‌سازی طول جملات

برای اینکه طول همه جملات در دیتاست یکسان شود، کافی است که طول بزرگترین جمله را به دست آورده و تمامی جملات را به آن اندازه برسانیم. کلمات اضافی با توکن <pad> پر می‌شود. در تابعی که تعریف شد تعداد pad مورد نیاز که برابر با تفاضل طول بزرگترین جمله و جمله فعلی است بدست آمده و به جمله فعلی اضافه می‌شود.

این مورد برای فیلد برچسب‌ها هم انجام می‌شود تا طول آن‌ها هم یکسان شود. کافی است که برای هر توکن پد اضافه شده مقدار ۰ به آن اضافه شود.

نتیجه به صورت زیر است که نشان می‌دهد این عمل به خوبی انجام شده است:

```
Before padding (text): ['The', 'progress', 'of', 'this', 'coordinated', 'offensive', 'was', 'already', 'very', 'entrenched', 'by', 'then', '.']
Before padding (labels): [0, 0, 0, 0, 0, 3, 0, 0, 0, 0, 0, 0, 0]
After padding (text): ['The', 'progress', 'of', 'this', 'coordinated', 'offensive', 'was', 'already', 'very', 'entrenched', 'by', 'then', '.', '
After padding (labels): [0, 0, 0, 0, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
```

در نهایت هم کلاس Vocab پیاده‌سازی شد. این کلاس دقیقا مطابق سوال پیاده‌سازی شده و نیازی به تکرار توضیحات آن وجود ندارد.

۲-۲_ مدل مبتنی بر LSTM ENCODER

ابتدا مطابق خواسته سوال یک Vocab میسازیم تا بتوانیم کلمات دیتاست را مدیریت کنیم.

مدل LSTM به صورت زیر است :

یک لایه LSTM تعریف شده است. در قسمت Forward hidden state را برای هر فعل باتوجه به شماره جایگاه کلمه فعل در هر نمونه از هر Batch را بدست آورده و در ادامه آن را با hidden state هر توکن میچسبانیم. عمل طبقه‌بندی پس از این عمل انجام می‌شود:

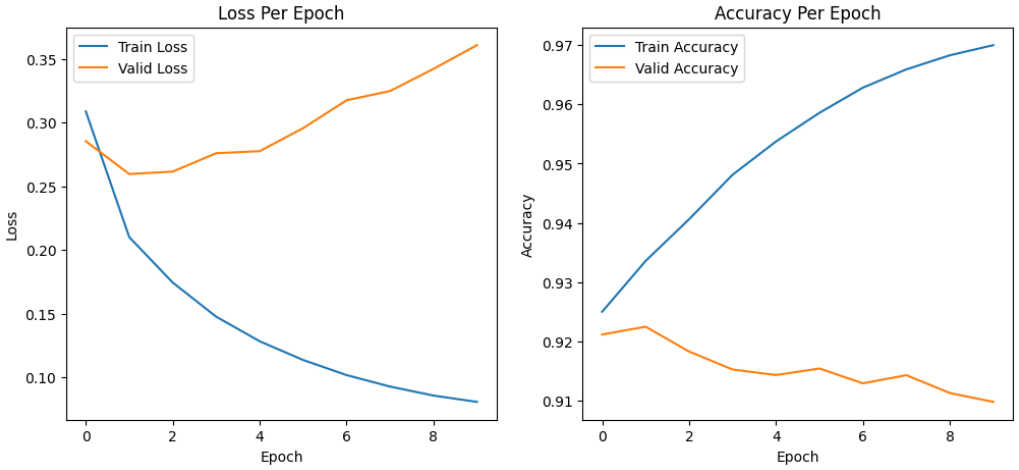
```python
class SRLModel_LSTM(nn.Module):
    def __init__(self, vocab_size, embedding_dim, hidden_dim, num_tags):
        super(SRLModel_LSTM, self).__init__()
        self.embedding = nn.Embedding(vocab_size, embedding_dim)
        self.lstm = nn.LSTM(embedding_dim, hidden_dim, batch_first=True)
        self.classifier = nn.Linear(hidden_dim * 2, num_tags)

    def forward(self, x, verb_indices):
        x = self.embedding(x)
        lstm_out, _ = self.lstm(x)
        verb_hidden_states = lstm_out[range(len(verb_indices)), verb_indices]
        verb_hidden_states = verb_hidden_states.unsqueeze(1).expand(-1, lstm_out.size(1), -1)
        lstm_concat = torch.cat((lstm_out, verb_hidden_states), dim=-1)
        srl_tags = self.classifier(lstm_concat)
        return srl_tags
```

برای مرحله یادگیری از ابرپارامترهای پیشنهادی استفاده شد.

```python
# New Train code
model = SRLModel_LSTM(len(vocab), embedding_dim=64, hidden_dim=64, num_tags=len(labels))
optimizer = torch.optim.Adam(model.parameters(), lr=0.1)
loss_function = nn.CrossEntropyLoss()

num_epochs = 10
history = train_model(model, train_loader, valid_loader, optimizer, loss_function, num_epochs)
```

```
Epoch 1/10: Train Loss = 0.3086, Train Acc = 0.9250, Valid Loss = 0.2853, Valid Acc = 0.9212
Epoch 2/10: Train Loss = 0.2100, Train Acc = 0.9336, Valid Loss = 0.2595, Valid Acc = 0.9225
Epoch 3/10: Train Loss = 0.1743, Train Acc = 0.9406, Valid Loss = 0.2614, Valid Acc = 0.9183
Epoch 4/10: Train Loss = 0.1476, Train Acc = 0.9481, Valid Loss = 0.2759, Valid Acc = 0.9153
Epoch 5/10: Train Loss = 0.1283, Train Acc = 0.9537, Valid Loss = 0.2775, Valid Acc = 0.9144
Epoch 6/10: Train Loss = 0.1136, Train Acc = 0.9586, Valid Loss = 0.2955, Valid Acc = 0.9155
Epoch 7/10: Train Loss = 0.1017, Train Acc = 0.9628, Valid Loss = 0.3174, Valid Acc = 0.9130
Epoch 8/10: Train Loss = 0.0927, Train Acc = 0.9659, Valid Loss = 0.3247, Valid Acc = 0.9144
Epoch 9/10: Train Loss = 0.0856, Train Acc = 0.9683, Valid Loss = 0.3421, Valid Acc = 0.9113
Epoch 10/10: Train Loss = 0.0807, Train Acc = 0.9700, Valid Loss = 0.3607, Valid Acc = 0.9099
```

نمودار خطا و دقت به صورت زیر است :



در نتیجه فرایند آموزش مقدار خطای داده‌های آموزشی به طور کلی کاهش یافته که نشان می‌دهند مدل به خوبی در حال یادگیری از روی داده‌های train است. مقدار خطا در داده‌های ارزیابی ولی پس از ۲ ایپاک که کاهش می‌یابد مجددا افزایش می‌یابد. این مورد نشان می‌دهد که پس از ۲ ایپاک اگرچه خطای داده آموزشی کاهش می‌یابد ولی مدل تعمیم خود را از دست داده و بر روی داده‌های دیده‌نشده به خوبی کار نمی‌کند که در اصل بیش‌برازش اتفاق می‌افتد. همین مورد برای دقت نیز برقرار است.

به طور کلی دقت مدل بالاست. البته این مورد را باید در نظر داشته باشیم که تعداد زیادی لیبل صفر نیز داریم که فکر می‌کنیم این مورد باعث می‌شود دقت افزایش یابد.

یکی از راه های ارزیابی داده‌ها استفاده از F1 score است. در این معیار، همانگونه که خواهیم دید، علاوه بر خطای مطلق (مثل Accuracy)، نوع خطا را نیز تاثیر می‌دهد.

$$F1\ score = 2 * \frac{Precision\ * Recall}{Precision + Recall}$$

فرمول فوق بیان می‌کند زمانی F1 score بیشتر است که داده های هر دسته به درستی تشخیص داده شده باشند.

مقدار F1 اگر لیبل صفر را حساب نکنیم برابر است با:

Validation F1 Score : 0.4491

که به نظر استفاده از این پارامتر میتواند بهتر باشد باتوجه به هدف این تمرین.

حدود ۱۰ بار مدل آموزش دیده‌شد در یکی از اجراها مقدار F1 اگر صفر را هم کلاس کنیم برابر زیر شد:

Validation F1 Score : 0.9071

خروجی نوت بوک که آخرین اجرا نیست:

Validation F1 Score : 0.9100

باتوجه به اینکه گفته شد مهم نحوه پیاده‌سازی است تلاش بیشتری برای افزایش آن انجام نشد.

آموزش بدون خطای ایندکس صفر

باتوجه به اینکه هدف یافتن لیبل‌های دارای ارتباط معنایی است و pad ها جز لیبل ها نیستند، اینبار همان مدل قبلی را تابع loss با پارامتر ignore_index = 0 آموزش می‌دهیم.

Train Model with Ignore Index 0 for loss

```
lstm_model = SRLModel_LSTM(len(vocab), embedding_dim=64, hidden_dim=64, num_tags=len(labels))
optimizer = torch.optim.AdamW(model.parameters(), lr=0.01)
loss_function = nn.CrossEntropyLoss(ignore_index=0)  # '0' is the label for padding

num_epochs = 10
history = train_model(model, train_loader, valid_loader, optimizer, loss_function, num_epochs)
```

در این صورت نتیجه بسیار ضعیف خواهد بود:

Train Model with Ignore Index 0 for loss

```
lstm_model = SRLModel_LSTM(len(vocab), embedding_dim=64, hidden_dim=64, num_tags=len(labels))
optimizer = torch.optim.Adam(lstm_model.parameters(), lr=0.001)
loss_function = nn.CrossEntropyLoss(ignore_index=0)  # '0' is the label for padding

num_epochs = 10
history = train_model(lstm_model, train_loader, valid_loader, optimizer, loss_function, num_epochs)
```

```
Epoch 1/10: Train Loss = 1.6074, Train Acc = 0.0306, Valid Loss = 1.3752, Valid Acc = 0.0380
Epoch 2/10: Train Loss = 1.1538, Train Acc = 0.0415, Valid Loss = 1.1605, Valid Acc = 0.0450
Epoch 3/10: Train Loss = 0.9464, Train Acc = 0.0473, Valid Loss = 1.0330, Valid Acc = 0.0496
Epoch 4/10: Train Loss = 0.8099, Train Acc = 0.0511, Valid Loss = 0.9867, Valid Acc = 0.0505
Epoch 5/10: Train Loss = 0.7041, Train Acc = 0.0541, Valid Loss = 0.9369, Valid Acc = 0.0524
Epoch 6/10: Train Loss = 0.6152, Train Acc = 0.0564, Valid Loss = 0.9177, Valid Acc = 0.0531
Epoch 7/10: Train Loss = 0.5401, Train Acc = 0.0585, Valid Loss = 0.9120, Valid Acc = 0.0539
Epoch 8/10: Train Loss = 0.4721, Train Acc = 0.0603, Valid Loss = 0.9044, Valid Acc = 0.0542
Epoch 9/10: Train Loss = 0.4111, Train Acc = 0.0620, Valid Loss = 0.9298, Valid Acc = 0.0542
Epoch 10/10: Train Loss = 0.3583, Train Acc = 0.0634, Valid Loss = 0.9361, Valid Acc = 0.0553
```

تلاش زیادی برای بهبود مدل و آموزش با پارامترهای مختلف انجام شد ولی نتیجه بسیار متفاوتی در پی نداشت. البته این آموزش روی همان حلقه آموزش قبلی انجام شد.

آموزش و محاسبه خطا و دقت بدون در نظر گرفتن ایندکس صفر

یک حلقه آموزشی جدید نوشته شد که در آن لیبل padding ها را mask میکنیم تا فقط دقت لیبل‌ها محاسبه شود.

نتیجه آموزش با این مفروضات بسیار امیدوارکننده بود و مدل پیشرفت خوبی داشت. به نظر می‌رسد که این راه بهترین راه برای آموزش همچین مدلی برای همچین هدفی باشد.

همچنین برای محاسبه f1 نیز این مورد در نظر گرفته شد و لیبل pad ها را در نظر نگرفتیم.

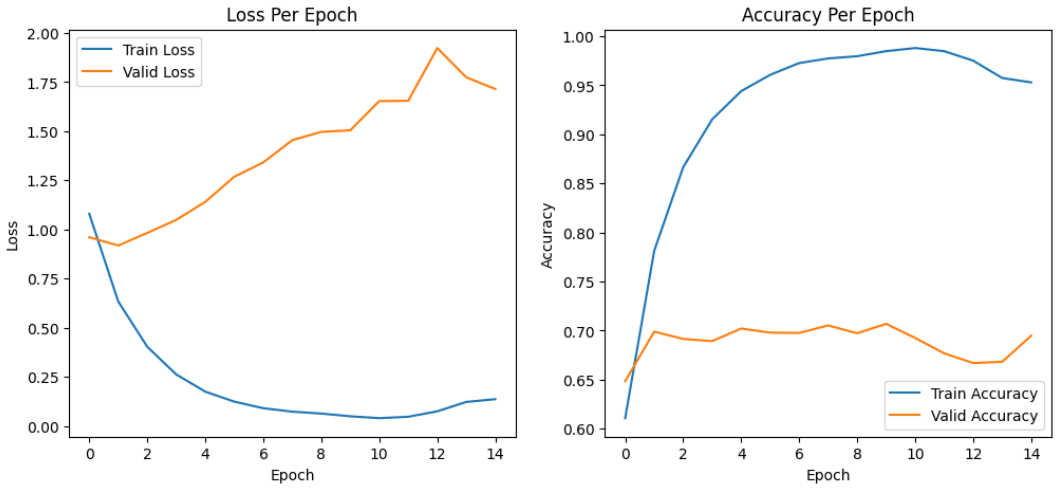
نتیجه آموزش به صورت زیر است:

Train Model with Ignore Index 0 for loss

```
lstm_model = SRLModel_LSTM(len(vocab), embedding_dim=64, hidden_dim=64, num_tags=len(labels))
optimizer = torch.optim.Adam(lstm_model.parameters(), lr=0.01)
loss_function = nn.CrossEntropyLoss(ignore_index=0)  # '0' is the label for padding

num_epochs = 15
history = train_model_mask(lstm_model, train_loader, valid_loader, optimizer, loss_function, num_epochs, device)
```

```
Epoch 1/15: Train Loss = 1.0802, Train Acc = 0.6108, Valid Loss = 0.9613, Valid Acc = 0.6484
Epoch 2/15: Train Loss = 0.6339, Train Acc = 0.7812, Valid Loss = 0.9189, Valid Acc = 0.6989
Epoch 3/15: Train Loss = 0.4039, Train Acc = 0.8665, Valid Loss = 0.9830, Valid Acc = 0.6916
Epoch 4/15: Train Loss = 0.2632, Train Acc = 0.9153, Valid Loss = 1.0492, Valid Acc = 0.6894
Epoch 5/15: Train Loss = 0.1759, Train Acc = 0.9440, Valid Loss = 1.1409, Valid Acc = 0.7021
Epoch 6/15: Train Loss = 0.1250, Train Acc = 0.9607, Valid Loss = 1.2685, Valid Acc = 0.6980
Epoch 7/15: Train Loss = 0.0916, Train Acc = 0.9726, Valid Loss = 1.3414, Valid Acc = 0.6977
Epoch 8/15: Train Loss = 0.0740, Train Acc = 0.9774, Valid Loss = 1.4546, Valid Acc = 0.7053
Epoch 9/15: Train Loss = 0.0643, Train Acc = 0.9796, Valid Loss = 1.4968, Valid Acc = 0.6973
Epoch 10/15: Train Loss = 0.0505, Train Acc = 0.9848, Valid Loss = 1.5047, Valid Acc = 0.7069
Epoch 11/15: Train Loss = 0.0410, Train Acc = 0.9879, Valid Loss = 1.6534, Valid Acc = 0.6926
Epoch 12/15: Train Loss = 0.0481, Train Acc = 0.9848, Valid Loss = 1.6551, Valid Acc = 0.6769
Epoch 13/15: Train Loss = 0.0757, Train Acc = 0.9751, Valid Loss = 1.9231, Valid Acc = 0.6670
Epoch 14/15: Train Loss = 0.1235, Train Acc = 0.9574, Valid Loss = 1.7741, Valid Acc = 0.6683
Epoch 15/15: Train Loss = 0.1374, Train Acc = 0.9530, Valid Loss = 1.7150, Valid Acc = 0.6948
```

پس نتیجه نهایی این سوال با این پارامترها به دست آمد. از بهینه ساز AdamW استفاده شد و نرخ یادگیری را روی ۰.۰۱ قرار دادیم. نرخ کمتر و بیشتر دقت کمتری نتیجه میداد. بچ سایز هم ۶۴ بود. نمودار خطا و دقت هم به صورت زیر است :



مشاهده می‌شود که دقت مدل روی داده‌های آموزشی بسیار عالی است. در داده‌های ارزیابی نیز تا ایپلاک ۷ پیشرفت خوبی حاصل می‌شود و پس از آن بیش‌برازش خواهیم داشت.

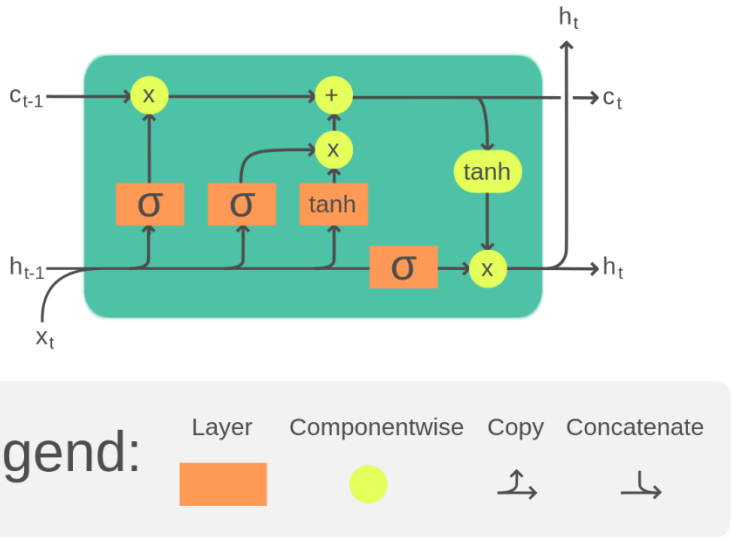
در این مدل نهایی پارامتر F1 به صورت زیر است :

Validation F1 Score: 0.6942

اگرچه این مورد از ۰.۹ قبلی کمتر است ولی در این مدل ما لیبل پدها را در نظر نگرفتیم.

۲-۳_ چرا LSTM برای SRL

مشاهده شد که دقت این مدل برای تشخیص لیبل‌ها مناسب است. در اینجا این مورد را بررسی میکنیم که چرا از LSTM برای SRL استفاده کرده‌ایم. همانطور که در نمونه‌ها نیز مشاهده شد برای تشخیص بعضی لیبل‌ها به دلیل اینکه فاصله بین کلمه با فعل زیاد است، مشکلی وجود دارد که باعث می‌شود یک شبکه عصبی عادی نتواند این ارتباط را به خوبی تشخیص دهد. ولی در LSTM ما اطلاعات مورد نیاز برای تشخیص آن را در حافظه ذخیره کرده و این مورد کمک می‌کند که در جمله کلماتی که دور از هم هستند ولی روی تشخیص لیبل تاثیر دارند را حفظ کنیم.



۲-۴_ مدل مبتنی بر GRU Encoder Model

معماری این مدل نیز همانند معماری ارائه شده برای LSTM است و در قسمت Forward، hidden state را برای هر فعل باتوجه به شماره جایگاه کلمه فعل در هر نمونه از هر Batch را بدست آورده و در ادامه آن را با hidden state هر توکن میچسبانیم. عمل طبقه‌بندی پس از این عمل انجام می‌شود:

Part3 : GRU Encoder Model

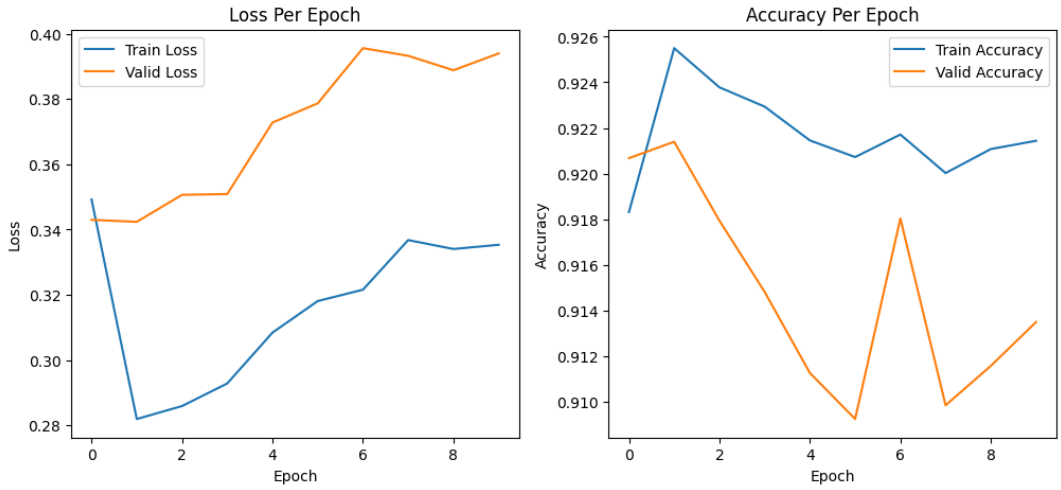
```python
class SRLModel_GRU(nn.Module):
    def __init__(self, vocab_size, embedding_dim, hidden_dim, num_tags):
        super(SRLModel_GRU, self).__init__()
        self.embedding = nn.Embedding(vocab_size, embedding_dim)
        self.gru = nn.GRU(embedding_dim, hidden_dim, batch_first=True)
        self.classifier = nn.Linear(hidden_dim * 2, num_tags)

    def forward(self, x, verb_indices):
        x = self.embedding(x)
        gru_out, _ = self.gru(x)
        verb_hidden_states = gru_out[range(len(verb_indices)), verb_indices]
        verb_hidden_states = verb_hidden_states.unsqueeze(1).expand(-1, gru_out.size(1), -1)
        gru_concat = torch.cat((gru_out, verb_hidden_states), dim=-1)
        srl_tags = self.classifier(gru_concat)
        return srl_tags
```

این مدل نیز باتوجه به هایپرپارامترهای پیش‌نهادی آموزش داده شد و نتیجه زیر حاصل شد:

```python
model = SRLModel_GRU(len(vocab), embedding_dim=64, hidden_dim=64, num_tags=len(labels))
optimizer = torch.optim.AdamW(model.parameters(), lr=0.1)
loss_function = nn.CrossEntropyLoss()
num_epochs = 10
history = train_model(model, train_loader, valid_loader, optimizer, loss_function, num_epochs)
```

```
Epoch 1/10: Train Loss = 0.3377, Train Acc = 0.9192, Valid Loss = 0.3284, Valid Acc = 0.9207
Epoch 2/10: Train Loss = 0.2754, Train Acc = 0.9267, Valid Loss = 0.3309, Valid Acc = 0.9125
Epoch 3/10: Train Loss = 0.2834, Train Acc = 0.9244, Valid Loss = 0.3721, Valid Acc = 0.9110
Epoch 4/10: Train Loss = 0.2989, Train Acc = 0.9227, Valid Loss = 0.3668, Valid Acc = 0.9129
Epoch 5/10: Train Loss = 0.3426, Train Acc = 0.9180, Valid Loss = 0.4262, Valid Acc = 0.8953
Epoch 6/10: Train Loss = 0.3553, Train Acc = 0.9200, Valid Loss = 0.4153, Valid Acc = 0.9147
Epoch 7/10: Train Loss = 0.3500, Train Acc = 0.9209, Valid Loss = 0.3924, Valid Acc = 0.9203
Epoch 8/10: Train Loss = 0.3530, Train Acc = 0.9202, Valid Loss = 0.3934, Valid Acc = 0.9107
Epoch 9/10: Train Loss = 0.3548, Train Acc = 0.9203, Valid Loss = 0.4220, Valid Acc = 0.9178
Epoch 10/10: Train Loss = 0.3628, Train Acc = 0.9214, Valid Loss = 0.3964, Valid Acc = 0.9192
```

نمودار خطا و دقت آن به صورت زیر است :



همانطور که مشاهده می‌شود در این مدل هم خطا پس از ۲ ایپاک افزایش یافته و هم دقت کاهش می‌یابد. متوجه می‌شویم که پس از ایپاک دوم فرآیند یادگیری به خوبی انجام نمی‌شود. به طور کلی این مدل از مدل قبلی مرحله آموزش سریع‌تری دارد ولی دقت آن کمی پایین تر است.

مقدار F1 آن نیز پس از چند اجرا برابر مقدار زیر شد:

Validation F1 Score : 0.8847

همانطور که مشاهده می‌شود از مقدار ارائه شده در قسمت LSTM کمتر است.

آموزش و محاسبه خطا و دقت بدون در نظر گرفتن ایندکس صفر

یک حلقه آموزشی جدید نوشته شد که در آن لیبل padding ها را mask میکنیم تا فقط دقت لیبل‌ها محاسبه شود.

همچنین برای محاسبه f1 نیز این مورد در نظر گرفته شد و لیبل pad ها را در نظر نگرفتیم.

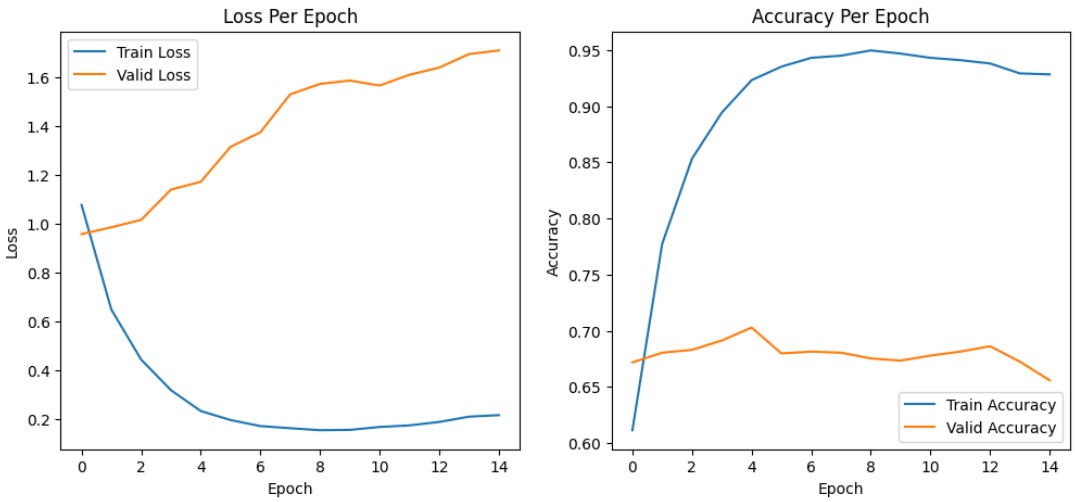
نتیجه آموزش به صورت زیر است:

Train Model with Ignore Index 0 for loss

```
gru_model = SRLModel_GRU(len(vocab), embedding_dim=64, hidden_dim=64, num_tags=len(labels))
optimizer = torch.optim.Adam(gru_model.parameters(), lr=0.01)
loss_function = nn.CrossEntropyLoss(ignore_index=0)  # '0' is the label for padding

num_epochs = 15
history = train_model_mask(gru_model, train_loader, valid_loader, optimizer, loss_function, num_epochs, device)
```

```
Epoch 1/15: Train Loss = 1.0771, Train Acc = 0.6114, Valid Loss = 0.9573, Valid Acc = 0.6718
Epoch 2/15: Train Loss = 0.6483, Train Acc = 0.7774, Valid Loss = 0.9851, Valid Acc = 0.6804
Epoch 3/15: Train Loss = 0.4437, Train Acc = 0.8532, Valid Loss = 1.0158, Valid Acc = 0.6830
Epoch 4/15: Train Loss = 0.3184, Train Acc = 0.8943, Valid Loss = 1.1396, Valid Acc = 0.6913
Epoch 5/15: Train Loss = 0.2330, Train Acc = 0.9232, Valid Loss = 1.1716, Valid Acc = 0.7028
Epoch 6/15: Train Loss = 0.1961, Train Acc = 0.9352, Valid Loss = 1.3147, Valid Acc = 0.6798
Epoch 7/15: Train Loss = 0.1714, Train Acc = 0.9431, Valid Loss = 1.3741, Valid Acc = 0.6814
Epoch 8/15: Train Loss = 0.1626, Train Acc = 0.9450, Valid Loss = 1.5295, Valid Acc = 0.6804
Epoch 9/15: Train Loss = 0.1545, Train Acc = 0.9497, Valid Loss = 1.5721, Valid Acc = 0.6753
Epoch 10/15: Train Loss = 0.1558, Train Acc = 0.9468, Valid Loss = 1.5860, Valid Acc = 0.6734
Epoch 11/15: Train Loss = 0.1677, Train Acc = 0.9431, Valid Loss = 1.5656, Valid Acc = 0.6779
Epoch 12/15: Train Loss = 0.1742, Train Acc = 0.9410, Valid Loss = 1.6097, Valid Acc = 0.6814
Epoch 13/15: Train Loss = 0.1883, Train Acc = 0.9380, Valid Loss = 1.6389, Valid Acc = 0.6862
Epoch 14/15: Train Loss = 0.2099, Train Acc = 0.9292, Valid Loss = 1.6944, Valid Acc = 0.6724
Epoch 15/15: Train Loss = 0.2162, Train Acc = 0.9283, Valid Loss = 1.7094, Valid Acc = 0.6558
```

پس نتیجه نهایی این سوال با این پارامترها به دست آمد. از بهینه ساز AdamW استفاده شد و نرخ یادگیری را روی ۰.۰۱ قرار دادیم. نرخ کمتر و بیشتر دقت کمتری نتیجه میداد. بچ سایز هم ۶۴ بود. نمودار خطا و دقت هم به صورت زیر است :



مشاهده می‌شود که دقت مدل روی داده‌های آموزشی بسیار عالی است. در داده‌های ارزیابی نیز تا ایپلاک ۴ پیشرفت خوبی حاصل می‌شود و پس از آن بیش‌برازش خواهیم داشت.

در این مدل نهایی پارامتر F1 به صورت زیر است :

Validation F1 Score: 0.6568

اگرچه این مورد از ۰.۸۸ قبلی کمتر است ولی در این مدل ما لیبل پدها را در نظر نگرفتیم.

۵-۲_ پاسخ به سوالات بخش چهارم

مزیت LSTM به RNN

همانطور که در بخش قبلی نیز دلیل استفاده از LSTM برای SRL را به صورت کلی شرح دادیم، برای تشخیص بعضی لیبل‌ها به دلیل اینکه فاصله بین کلمه با فعل زیاد است، مشکلی وجود دارد که باعث می‌شود یک شبکه عصبی عادی نتواند این ارتباط را به خوبی تشخیص دهد. ولی در LSTM ما اطلاعات مورد نیاز برای تشخیص آن را در حافظه ذخیره کرده و این مورد کمک می‌کند که در جمله کلماتی که دور از هم هستند ولی روی تشخیص لیبل تاثیر دارند را حفظ کنیم.

پس یکی از مزایای آن این است که با کمک LSTM میتوانیم اطلاعات را برای مدت طولانی تری حفظ کنیم و همچنین از مشکل محوشدگی گرادیان یا Vanishing جلوگیری کنیم.

همچنین در LSTM که شکل آن در قسمت قبل مشاهده شد این مکانیزم وجود دارد که اطلاعات مهم‌تر حفظ و اطلاعاتی که نیاز نیست از حافظه حذف شوند.

به طور کلی این مورد به ما کمک می‌کند که در تسک‌های مختلف NLP که دنباله‌ای از کلمات وابسته به هم داریم بتوانیم انعطاف‌پذیری بیشتری داشته باشیم.

تفاوت LSTM و GRU

هردوی آنها برای حل مشکل ناپدید شدن گرادیان در شبکه‌های RNN طراحی شده‌اند. هر دو ساختار از گیت‌های مختلفی برای کنترل جریان اطلاعات حافظه استفاده می‌کنند، اما در ساختار و تعداد گیت‌های مورد استفاده تفاوت‌هایی دارد.

LSTM دارای سه گیت است: ورودی، فراموشی یا forget، و خروجی. همچنین از یک cell state برای حفظ اطلاعات در طول زمان استفاده می‌کند که به آن اجازه می‌دهد اطلاعات را برای مدت طولانی‌تری حفظ کرده و مدیریت کند.

GRU تنها دو گیت دارد: دروازه Update و دروازه RESET. این معماری از حالت داخلی جداگانه‌ای مانند LSTM استفاده نمی‌کند، بلکه ترکیبی از hidden state و cell state را در یک ساختار واحد به کار می‌برد که می‌تواند به ساده‌سازی مدل کمک کند. این مورد کمک می‌کند که سرعت آموزش افزایش یابد زیرا که پارامترهای کمتری داریم. ولی در نهایت LSTM معمولا بهتر جواب می‌دهد. هرچند که این مورد بستگی به هر تسک و داده دارد.

چسباندن hidden state فعل

در تسک ارائه شده در این تمرین یعنی SRL اصلی ترین کلمه که بقیه کلمات از آن تاثیر میپذیرند فعل است. در طراحی این مدل که در قسمت قبل بررسی شد، Hidden Sate هر فعل به هر توکن چسبیده شد که هدف اصلی آن افزایش دقت مدل بود. باتوجه به اینکه برخی کلمات از فعل دور هستند وقتی که این عمل انجام می‌شود، مدل تشخیص بهتری از نقش معنایی کلمه داشته باشد و لیبل آن به خوبی تشخیص داده شود. همچنین جمله ممکن است فعل‌های مختلفی داشته باشد، ولی آنچه مد نظر ما در مرحله آموزشی است مورد تاکید قرار می‌گیرد.

محوشدگی گرادیان

همانطور که در قسمت‌های قبلی هم اشاره کردیم، مشکل محو شدگی گرادیان‌ها یکی از چالش‌های اصلی در شبکه‌های عصبی بازگشتی (RNN) است، به خصوص وقتی با داده‌هایی که دارای وابستگی‌های

طولانی هستند سروکار داریم. این مشکل باعث می‌شود که گرادیان‌ها در طی بازگشت، به سرعت کاهش یابند و به صفر نزدیک شوند، که این امر موجب می‌شود مدل نتواند از داده‌های ابتدایی برای یادگیری استفاده کند.

تابع ReLU به عنوان activation function می‌توانند به کاهش محو شدگی گرادیان کمک کنند زیرا گرادیان آنها در مقایسه با sigmoid یا tanh بهتر حفظ می‌شود. همچنین نرخ یادگیری بالا می‌تواند موجب تشدید محو شدگی گرادیان شود. کاهش نرخ یادگیری می‌تواند به کنترل بهتر گرادیان ها و جلوگیری از ناپدید شدن آن ها کمک کند. استفاده از روش‌های نرمال سازیمانند Batch Normalization یا Layer Normalization که می‌توانند به استاندارد کردن توزیع ورودی هر لایه کمک کرده و از محو شدگی گرادیان جلوگیری کنند.

۶-۲_ مدل مبتنی بر Encoder-Decoder

در این قسمت میخواهیم این مسئله را به صورت پرسش پاسخ یا QAبررسی کنیم. باتوجه به اینکه در قسمت قبل این سوال تمامی لیبل‌ها به عدد تبدیل شد، در این قسمت مجددا دیتاست را از ورودی میخوانیم.

تبدیل داده‌ها به جفت پرسش‌پاسخ

ابتدا یک تابع نوشته که لیبل‌های زیر را مد نظر قرار می‌دهیم:

['ARG0', 'ARG1', 'ARG2', 'ARGM-TMP', 'ARGM-LOC']

در این تابع در یک حلقه تمامی جملات بررسی می‌شود و برای تمامی لیبل‌های بالا، پرسش و پاسخ تولید می‌کنیم. سپس در یک حلقه داخلی تر تمامی لیبل‌ها بررسی می‌شوند و اگر آن جمله لیبل مورد نظر را داشت کلمات را به فرم نمونه ارائه شده بین توکن آغازین و پایان جمله قرار می‌دهیم. در نهایت نیز در یک حلقه جداگانه تمامی فعل‌ها را به ابتدای پرسش می‌چسبانیم.

برای نمونه المنت ۱ ام از داده‌های آموزشی پس از اعمال این تابع به صورت زیر خواهد بود:

```
qa_train[1]
```

```
('inscribed [SEPT] A primary stele , three secondary steles , and two inscribed steles . ARG1',
 '<s> steles </s>')
```

امبدینگ کلمات برای مقداردهی اولیه

برای بدست آوردن امبدینگ کلمات از یک مدل pretrained استفاده می‌کنیم. برای اینکار Glove نسخه 6B را دانلود کرده و پس از استخراج در درایو آن را ایمپورت کرده و برای تمامی کلمات امبینگ را در یک ماتریس ذخیره کرده و این ماتریس را در ادامه به مدل به عنوان مقداردهی اولیه می‌دهیم.

```
embedding_dim = 300
embedding_matrix = np.zeros((len(vocab.word2id) + 1, embedding_dim))  # +1 for zero indexing
for word, i in vocab.word2id.items():
    embedding_vector = embeddings_dict.get(word)
    if embedding_vector is not None:
        embedding_matrix[i] = embedding_vector
    else:
        # Words not found in embedding index will be all-zeros.
        embedding_matrix[i] = np.random.normal(size=(embedding_dim,))  # Or other initializatio
```

مدل seq2seq

این نوع معماری برای بسیاری از تسک‌های NLP ایده‌آل است. حال به بررسی قسمت‌های مختلف پیاده‌سازی‌شده می‌پردازیم:

انکودر در این مدل یک لایه امبدینگ با مقداردهی اولیه برای تبدیل اندیس کلمات به بردار دارد. همچنین یک لایه LSTM به صورت bidirectionalپیاده سازی شده که اطلاعات هم از ابتدا به انتها و هم از انتها به ابتدا پردازش شده و نتیجه بهتری را خروجی می‌دهد. خروجی hidden state و cell state برای محاسبه attention مرحله بعد نیز استفاده می‌شود.

متد توجه یا attention وظیفه دارد تا وزن توجه را برای هر توکن در خروجی انکودر محاسبه کند. این وزن‌ها بر اساس خروجی انکودر و hidden state دیکودر محاسبه شده و اهمیت هر توکن را برای تولید کلمه بعدی در دیکودر نشان می‌دهد.

دیکورد نیز مانند انکورد یک لایه امبدینگ داشته و همچنین از مکانیزم توجه استفاده می‌کند. ورودی لایه LSTM در این دیکودر ترکیبی از بردارهای ویژگی و بردارهای مکانیزم توجه است. در نهایت یک لایه FULLY CONNECTED برای تبدیلhiddenstate به احتمال هر توکن خروجی استفاده می‌شود.

کلاس Seq2Seq نیز ساختار کلی مدل را تعریف می‌کند، که شامل انکودر و دیکودر است. این مدل داده‌های ورودی را از طریق انکودر می‌گیرد و خروجی‌های آن را به دیکودر منتقل می‌کند. در هر مرحله از تولید دنباله خروجی، دیکودر بر اساس حالت فعلی و خروجی‌های انکودر، کلمه بعدی را تولید می‌کند.

آماده‌سازی داده‌ها برای seq2seq

در این مرحله باید دیتاست مورد نیاز این مسئله را تولید کنیم. هر جمله پرسش و پاسخ باید به یک سری عدد تبدیل شوند که برای اینکار یک کلاس تعریف کرده و همچنین از متدهای کلاس Vocab استفاده می‌کنیم. قبل از تبدیل هر کلمه به ایندکس مورد نظر باید عمل توکنایزیشن نیز انجام شود. برای نمونه المنت ۱ام از داده‌های آموزشی پس از اعمال این توابع و استفاده از کلاس دیتاست به صورت زیر خواهد بود:

```
qa_train[1]
```

```
('inscribed [SEPT] A primary stele , three secondary steles , and two inscribed steles . ARG1',
 '<s> steles </s>')
```

```
train_dataset[1]
```

```
([11660,
  3,
  129,
  3401,
  11659,
  10,
  185,
  8394,
  8395,
  10,
  14,
  100,
  11660,
  8395,
  12,
  6],
 [1, 8395, 2])
```

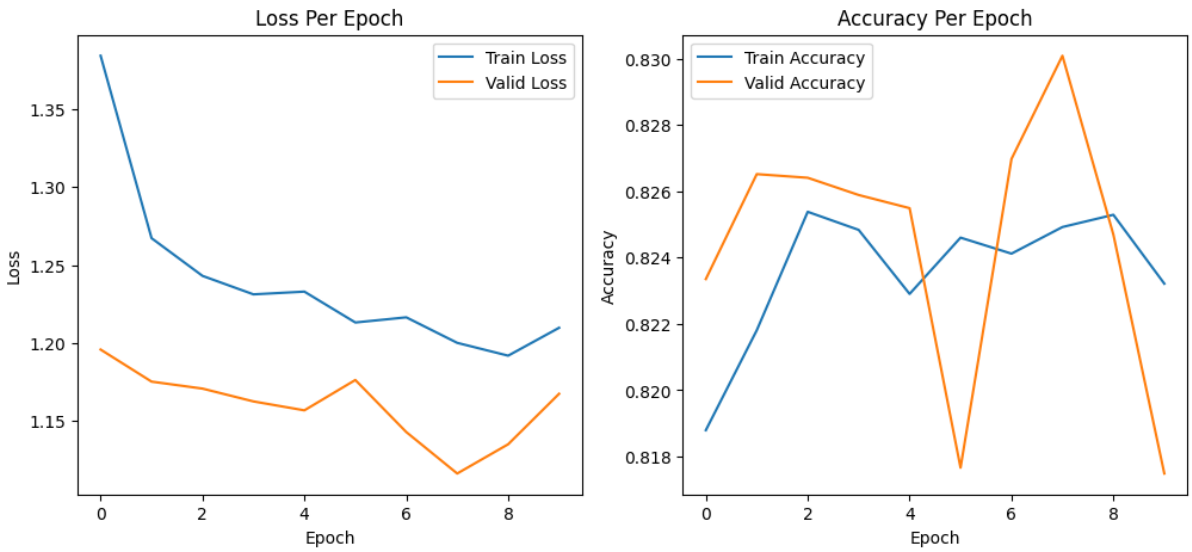
که به صورت یک جفت پرسش و پاسخ تفکیک شده ارائه می‌شود. مثلا در مثال بالا ۱ و ۲ همان توکن‌های آغازین و پایان جمله هستند.

پس از آموزش این مدل نتایج زیر حاصل شد:

```
encoder = Encoder(vocab_size=len(vocab), embedding_dim=300, hidden_dim=64, embedding_matrix=embedding_matrix, bidirectional=True).to(device)
decoder = Decoder(embedding_dim=300, hidden_dim=64, output_dim=len(vocab), embedding_matrix=embedding_matrix, bidirectional_encoder=True).to(device)
model = Seq2Seq(encoder, decoder, device).to(device)

criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.01)

num_epochs = 10
history = train_model(model, train_loader, valid_loader, optimizer, criterion, num_epochs)
```

```
Epoch 1/10: Train Loss = 1.3840, Train Acc = 0.8188, Valid Loss = 1.1959, Valid Acc = 0.8234
Epoch 2/10: Train Loss = 1.2672, Train Acc = 0.8218, Valid Loss = 1.1753, Valid Acc = 0.8265
Epoch 3/10: Train Loss = 1.2431, Train Acc = 0.8254, Valid Loss = 1.1709, Valid Acc = 0.8264
Epoch 4/10: Train Loss = 1.2312, Train Acc = 0.8248, Valid Loss = 1.1627, Valid Acc = 0.8259
Epoch 5/10: Train Loss = 1.2330, Train Acc = 0.8229, Valid Loss = 1.1569, Valid Acc = 0.8255
Epoch 6/10: Train Loss = 1.2132, Train Acc = 0.8246, Valid Loss = 1.1764, Valid Acc = 0.8177
Epoch 7/10: Train Loss = 1.2165, Train Acc = 0.8241, Valid Loss = 1.1431, Valid Acc = 0.8270
Epoch 8/10: Train Loss = 1.2002, Train Acc = 0.8249, Valid Loss = 1.1165, Valid Acc = 0.8301
Epoch 9/10: Train Loss = 1.1919, Train Acc = 0.8253, Valid Loss = 1.1352, Valid Acc = 0.8247
Epoch 10/10: Train Loss = 1.2098, Train Acc = 0.8232, Valid Loss = 1.1676, Valid Acc = 0.8175
```

نمودار دقت و خطا به صورت زیر است:



مشخص اصت که به طور کلی خطا کاهش یافته است و مدل به روی داده‌های آموزشی جواب می‌دهد. ولی در ایپاک ۵ خطا افزایش یافته و پس از آن نوسانی می‌شود. احتمالا مدل دچار بیش‌برازش می‌شود. دقت نیز یک حالت نوسانی دارد.

به طور کلی مقدار f1 نیز به صورت زیر بدست آمد:

F1 Score on validation set: 0.7849972284224219

این مقدار از دو مدل قبلی کمتر است.

نظر شخصی: من انتظار داشتم این مدل دقت بالاتری داشته باشد. شاید دلیل دقت پایین تر این باشد که این مدل پیچیده‌تر است و نیاز به منابع بیشتری دارد.

۷-۲_ استفاده از BeamSearch

برای تولید خروجی در این مدل از BeamSearch استفاده می‌شود. در این رویکرد برای تولید خروجی چند Inference به صورت جداگانه از مدل انجام شده و در هر مرحله kبهترین احتمالات حفظ می‌شوند. این عمل تا یک حد خاصی تکرار شده و در نهایت Beam k برتر انتخاب شده و ماکزیمم گیری انجام می‌شود. از این روش برای تولید خروجی مدل‌های Seq2seq استفاده می‌شود.

ابتدا یک کلاس Beam پیاده‌سازی کرده سپس در متد مربوط به BeamSearch برای Inference از مدل آموزش داده‌شده ابتدا ورودی را توکنایز کرده و به عدد تبدیل می‌کنیم(مانند مرحله پیش‌پردازش داده‌ها) سپس جمله ورودی را به انکورد می‌دهیم. سپس Beam ها را مقداردهی اولیه می‌کنیم(مثلا احتمال ۰ و اینکه توکن آغازین جمله را در آن قرار می‌دهیم) سپس در یک حلقه به تعداد max_len عمل توضیح داده‌شده تکرار می‌شود و از بهترین Beam ها که بر اساس احتمال مرتب شده انتخاب می‌شود.

همچنین در پایان حلقه یک شرط برای مشاهده توکن پایانی جمله قرار می‌دهیم. در نهایت بهترین Beam انتخاب شده و اعداد ایندکس به کلمه تبدیل می‌شوند تا در خروجی چاپ شود.

یک نمونه خروجی از داده‌های آموزشی به صورت زیر است :

```
# Test Beam Search with trained data
qa_train[1]

('inscribed [SEPT] A primary stele , three secondary steles , and two inscribed steles . ARG1',
 '<s> steles </s>')

output_text = beam_search(model, "inscribed [SEPT] A primary stele , three secondary steles , and two inscribed steles . ARG1", vocab, beam_size=16, max_len=10)
print(output_text)

<s> steles </s>
```

۸-۲_ پاسخ سوالات قسمت چهارم

محدودیت‌های تبدیل به مسئله QA

یکی از مشکلات تبدیل جملات به پرسش‌پاسخ است که ممکن است بار اضافی داشته باشد. همچنین کیفیت این مدل به صورت مستقیم به کیفیت پرسش‌ها بستگی دارد. اگر که داده‌های آموزشی کیفیت مناسب و تعداد مناسب را نداشته باشند دقت کاهش می‌یابد. این مورد نسبی است یعنی منظور این است که این روش نسبت به روش‌های قبلی بررسی شده تعداد دادگان بیشتری می‌طلبد.

اینکه مسئله دقیقا کلمه مد نظر را تولید کند بسیار سخت است و باید مدل به خوبی تنظیم شود وگرنه ممکن است کلمات تولید شوند که ارتباطی با تسک ندارد. این مورد در این تمرین مشاهده شد.

همچنین محاسبه خطا و تحلیل آن با روش قبلی کاملا متفاوت است. در روش قبلی کافی بود لیبل ها مقایسه شوند. اینجا ممکن است بخشی از پاسخ تولید شود و یا هیچ قسمتی از پاسخ تولید نشود. همچنین پاسخ بسیاری از جملات ممکن است خالی باشد در جمله دیگر خالی نباشد.

استفاده از توکن‌های <s> و</s>

از این توکن‌ها برای مشخص کردن ابتدا و انتهای دنباله خروجی استفاده می‌شود تا مدل به این شکل از فرمت جمله آموزش ببیند تا در مرحله Inference از مدل بتوانیم به خوبی از مدل خروجی بگیریم. در مدل‌هایی که با جملات و دنباله‌های متغیر سروکار دارند، تعیین شروع و پایان به طور موثر به مدل کمک می‌کند تا یاد بگیرد چگونه با دنباله‌های مختلفی از طول‌ها کار کند. یکی دیگر از کاربردهای آن را در استفاده از روش BeamSearch برای تولید خروجی نیز مشاهده کردیم که تا زمان مشاهده توکن خروجی، توکن جدید تولید می‌شود. همچنین با مقداردهی اولیه با توکن آغاز جمله نشان دادیم که مدل باید کلمه جدید تولید کند.

آموزش و محاسبه خطا و دقت بدون در نظر گرفتن ایندکس صفر

اینبار مدل Seq2Seq را بدون توجه به loss لیبل پدها آموزش می‌دهیم. در مدل قبلی بسیاری از پاسخ‌ها خالی هستند و هچنین مدل جواب مناسبی به بسیاری از سوالات نمیدهد که ما انتظار داریم. یکی از دلایل میتواند محاسبه پاسخ خالی در خطا باشد. یکی دیگر از موارد کم بودن تعداد داده‌های آموزشی برای این نوع تسک است.

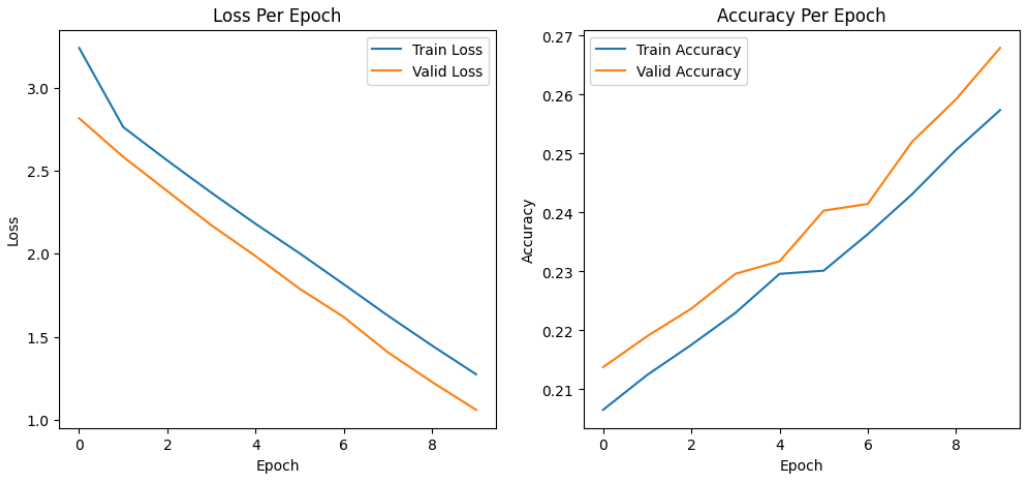
پارامترهای مختلفی برای آموزش استفاده شد ولی دقت در این حالت کمتر از ۰.۳۰ بود. مدت زمان آموزش در این روش بسیار طولانی تر از روش‌های قبلی بود. در این آموزش از امبدینگ ۳۰۰ بعدی استفاده شد. نرخ آموزشی هم ۰.۰۰۱ بود.

نتیجه آموزش داده‌ها به صورت زیر است :

```
encoder = Encoder(vocab_size=len(vocab), embedding_dim=300, hidden_dim=64, embedding_matrix=embedding_matrix, bidirectional=True).to(devi
decoder = Decoder(embedding_dim=300, hidden_dim=64, output_dim=len(vocab), embedding_matrix=embedding_matrix, bidirectional_encoder=True)
model = Seq2Seq(encoder, decoder, device).to(device)

loss_function = nn.CrossEntropyLoss(ignore_index=0)
optimizer = optim.AdamW(model.parameters(), lr=0.001)

num_epochs = 10
history = train_model(model, train_loader, valid_loader, optimizer, loss_function, num_epochs)
```

```
Epoch 1/10: Train Loss = 3.2395, Train Acc = 0.2065, Valid Loss = 2.8160, Valid Acc = 0.2138
Epoch 2/10: Train Loss = 2.7628, Train Acc = 0.2124, Valid Loss = 2.5835, Valid Acc = 0.2190
Epoch 3/10: Train Loss = 2.5616, Train Acc = 0.2176, Valid Loss = 2.3774, Valid Acc = 0.2237
Epoch 4/10: Train Loss = 2.3675, Train Acc = 0.2230, Valid Loss = 2.1717, Valid Acc = 0.2296
Epoch 5/10: Train Loss = 2.1805, Train Acc = 0.2296, Valid Loss = 1.9863, Valid Acc = 0.2317
Epoch 6/10: Train Loss = 2.0029, Train Acc = 0.2301, Valid Loss = 1.7895, Valid Acc = 0.2403
Epoch 7/10: Train Loss = 1.8169, Train Acc = 0.2363, Valid Loss = 1.6197, Valid Acc = 0.2415
Epoch 8/10: Train Loss = 1.6282, Train Acc = 0.2431, Valid Loss = 1.4071, Valid Acc = 0.2520
Epoch 9/10: Train Loss = 1.4479, Train Acc = 0.2506, Valid Loss = 1.2287, Valid Acc = 0.2592
Epoch 10/10: Train Loss = 1.2751, Train Acc = 0.2574, Valid Loss = 1.0612, Valid Acc = 0.2679
```

نمودار خطا دقت به صورت زیر است:



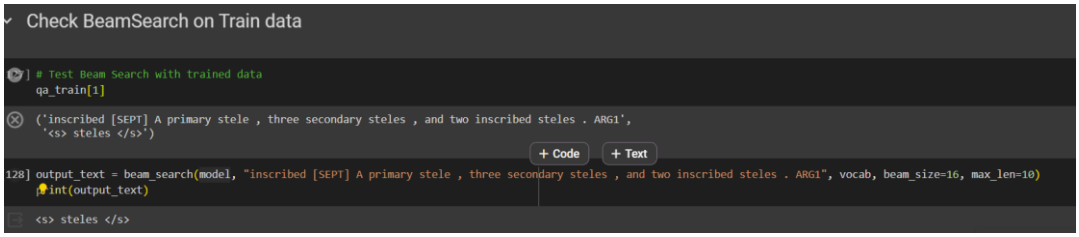
احتمالا اگر منابع پردازشی بیشتری در اختیار بود دقت بالاتری حاصل می‌شد.

مقدار پارامتر f1 در این حالت برابر است با :

F1 Score on validation set: 0.109332693451754

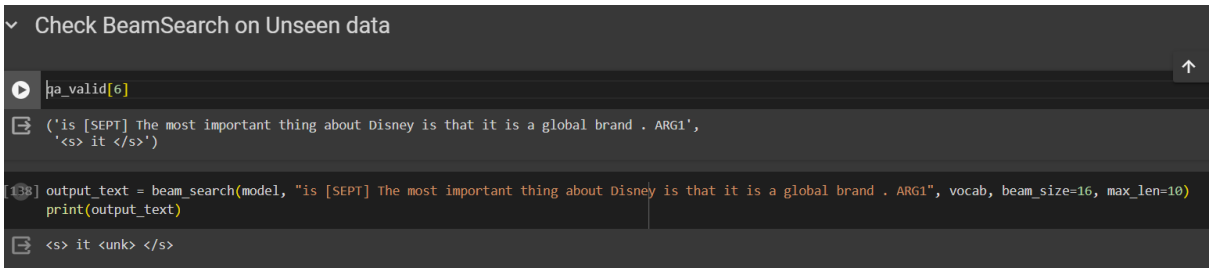
به طور کلی ماهیت این نوع از شبکه‌ها که بر اساس پرسش و پاسخ کار میکنند بسیار متفاوت است و بدیهی است که هم زمان آموزش بیشتر بوده و هم نسبت به آن دقت پایین تر است. ولی هر کدام مزیت‌ها و معایب خود را دارند که در بخش بعد آن‌ها را بررسی می‌کنیم.

در این مدل هم یک نمونه از پرسش پاسخ‌های داده‌های آموزشی را برای اینکه چک کنیم مدل کار میکند یا خیر را بررسی میکنم:



باتوجه به اینکه `<s> steles </s>` داده شد متوجه میشویم که مدل به خوبی طبق سوال پیاده‌سازی شده است.

این مورد بر روی داده‌های دیده‌نشده هم تست شد:



اینکه کلمه it را به خوبی تشخیص داده نشان‌دهنده این است که مدل بر روی داده‌های دیده نشده هم کار می‌کند. فقط دقت آن کم است. اگر منابع پردازشی بیشتر بود احتمالا دقت بالاتر میرفت. زیرا برخی موارد پاسخ مورد نظر دریافت نمیشد.

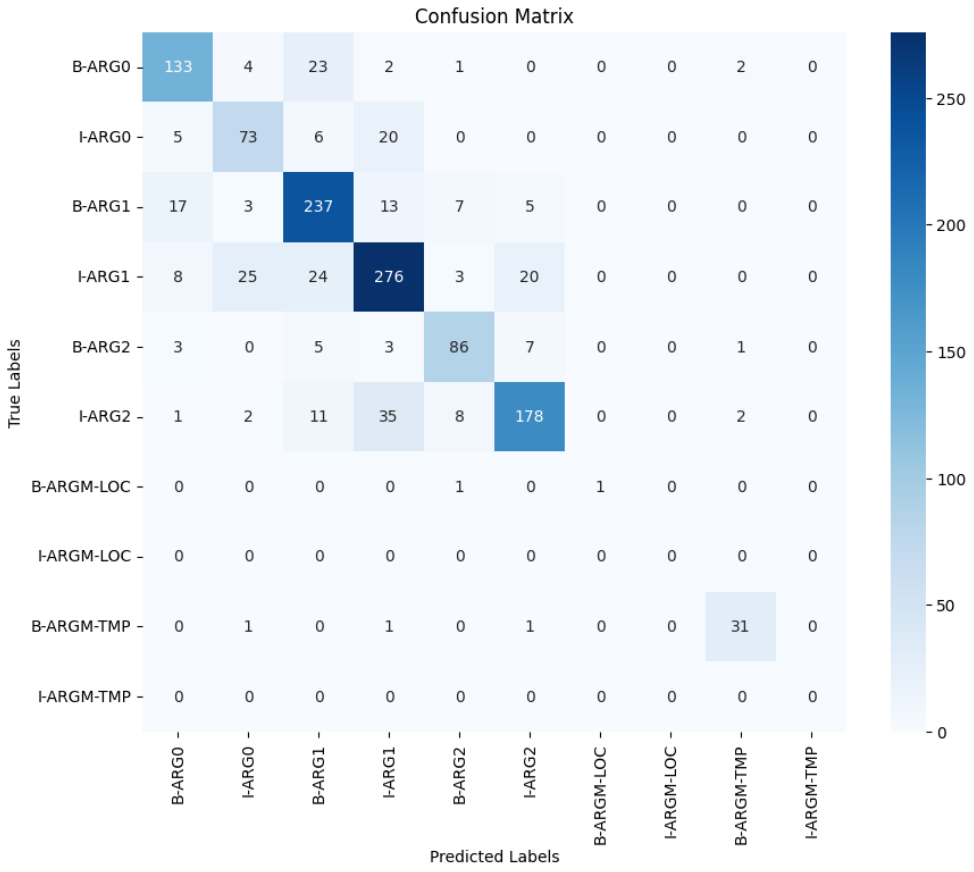
۹-۲_ تحلیل نتایج

مقایسه مدل بخش دوم و چهارم

ابتدا گزارش طبقه‌بندی و ماتریس کانفیوژن را خروجی می‌گیریم.

برای مدل بخش دوم:

```
Classification Report:
                precision    recall  f1-score   support

       B-ARG0       0.80      0.37      0.51       357
       I-ARG0       0.68      0.25      0.37       287
       B-ARG1       0.77      0.36      0.49       662
       I-ARG1       0.79      0.27      0.40      1032
       B-ARG2       0.81      0.39      0.53       219
       I-ARG2       0.84      0.36      0.51       493
    B-ARGM-LOC      1.00      0.10      0.18        10
    I-ARGM-LOC      1.00      1.00      1.00         0
    B-ARGM-TMP      0.86      0.45      0.59        69
    I-ARGM-TMP      1.00      1.00      1.00         0

     micro avg      0.79      0.32      0.46      3129
     macro avg      0.86      0.46      0.56      3129
  weighted avg      0.79      0.32      0.46      3129
```



برای مدل بخش چهارم:

متاسفانه هنگام پردازش این مورد منابع پردازشی کولب باز هم تمام شد. ولی به طور کلی در چند اجرا برای گرفتن خروجی ریپورت مشخص بود که کلماتی که ARGM-TMP بودند نسبت به مدل بخش دوم درصد بیشتری تشخیص داده شده بود.

در مدل بخش دوم `B-ARGM-LOC` دقت بسیار پایین است. ولی این موارد را مدل چهارم بهتر تشخیص می‌دهد.

به طور کلی باتوجه به نتایج این تمرین مدل بخش دوم نتایج کمی بهتری داشت.

به نظر می‌رسد که مدل LSTM Encoder بخش دوم در تشخیص نقش‌های معنایی که ارتباط مستقیم با فعل دارند بهتر عمل می‌کند. ARG0 و ARG1 از این نوع هستند که همان کنشگر و کنش‌پذیر هستند. این نقش‌های معنایی ساده‌تر هستند و روابط نزدیک‌تری با فعل دارند.

دقیقا این مورد در ConfusionMatrix قابل مشاهده بود و این مورد یکبار خروجی گرفته شد. متاسفانه به دلیل کمبود زمان گریس و همچنین نیازمند GPU برای ترید مجدد امکان اجرای مجدد وجود نداشت. ولی به طور کلی تعداد لیبل‌های این کلاس‌ها هم بیشتر از کلاس‌های دیگر هستند که این مورد هم مزید بر علت است که دقت مدل بالاتر بود.

باتوجه به مشکلات بالا تعداد لیبل‌ها به روش غیرحرفه‌ای و با سرچ در دیتاست انجام شد.

برای نقش‌های معنای پیچیده تر مدل بخش چهارم یا Encoder-Decoder باتوجه به اینکه درک عمیق‌تری از جمله دارد بهتر عمل می‌کند. به طور کلی معمولا نقش‌های معنایی مثل زمان و مکان فاصله بیشتری از فعل دارند.

ولی به طور کلی مدل LSTM بسیار سبک‌تر است و پردازش سبک‌تر داشته و کم‌هزینه‌تر است. ولی اگر نیاز به بررسی دقیق‌تر و جامع‌تر باید می‌توان از مدل Encoder-Decoder استفاده کرد.

مقایسه دو نمونه

دو مثال زیر برای درک بهتر این تفاوت قابل بررسی است:

جمله : "The company granted him the permission to access the confidential files."

مدل LSTM Encoder:

[Arg0 The company] granted [Arg1 him] the permission [Arg2 to access the confidential files]

مدل Encoder-Decoder :

[Arg0 The company] granted [Arg1 him] the permission [Arg2 to access the confidential files]

برای جمله ساده هر دو مدل می‌توانند به خوبی تشخیص دهند.

جمله دوم:

" Despite the heavy rains, the festival was attended by thousands of people, who enjoyed the various performances immensely".

مدل LSTM Encoder:

[ArgM-TMP Despite the heavy rains], the festival [Arg0 was attended by thousands of people] who enjoyed the various performances immensely

مدل Encoder-Decoder :

[ArgM-TMP Despite the heavy rains], the festival [Arg0 was attended by thousands of people, who enjoyed the various performances immensely]

جملات پیچیده‌تر نیاز به درک معنایی بهتری دارند.

برای درک بهتر این تمرین از مقالات مختلفی از جمله (Marcheggiani & Titov, 2017) ، (Daza & Frank, 2018) و (Chung & Park, 2017) استفاده شد.

۳_ مراجع

Chung, E., & Park, J. G. (2017). Sentence-Chain Based Seq2seq Model for Corpus Expansion. *Etri Journal*, *39*(4), 455-466.

Daza, A., & Frank, A. (2018). A sequence-to-sequence model for semantic role labeling. *arXiv preprint arXiv:1807.03006*.

Marcheggiani, D., & Titov, I. (2017). Encoding sentences with graph convolutional networks for semantic role labeling. *arXiv preprint arXiv:1703.04826*.