به نام خدا

دانشگاه تهران

دانشکدگان فنی

دانشکده مهندسی برق و کامپیوتر

درس پردازش زبان طبیعی

پاسخ تمرین ۳

نام و نام خانودگی: سامان اسلامی نظری

شماره دانشجویی: ۸۱۰۱۹۹۳۷۵

اردیبهشت ماه ۱۴۰۳

فهرست

بخش اول: آماده کردن مجموعه داده ۳

بخش دوم: LSTM Encoder Model ٤

پیاده‌سازی مدل ٤

مقدار f1 score برای مجموعه داده Validation ٥

بخش سوم: Gru Encoder Model ٥

پیاده‌سازی مدل ٥

مزیت LSTM به RNN چیست؟ ٦

تفاوت LSTM و GRU ٧

دلیل چسباندن hidden state فعل به بقیه توکن‌ها ٨

روش حل مشکل محو شدگی گرادیان ٨

بخش چهارم: Encoder-Decoder Model ٩

محدودیت‌های روش QA ٩

دلیل استفاده از تگ‌های <s> و </s> ١٠

بخش اول: آماده کردن مجموعه داده

در این بخش کلاس Vocab پیاده‌سازی شد که در فایل نوت‌بوک قابل مشاهده می‌باشد. این کلاس وظیفه دریافت فایل داده و ساخت Vocabulary از روی آن است. همچنین یک سری تابع برای دریافت داده و تبدیل آن‌ها به لیستی از ایندکس‌هایشان در Vocab پیاده‌سازی شد.

بخش دوم: LSTM Encoder Model

پیاده‌سازی مدل

معماری گفته شده در PyTorch به صورت زیر پیاده‌سازی شد:

```
class LSTMSemanticRoleLabeler(nn.Module):
    def __init__(self, embedding_dim, hidden_dim, vocab_size, labels_count):
        super(LSTMSemanticRoleLabeler, self).__init__()

        self.hidden_dim = hidden_dim
        self.word_embeddings = nn.Embedding(vocab_size, embedding_dim)
        self.lstm = nn.LSTM(embedding_dim, hidden_dim, batch_first=True)
        self.hidden2label = nn.Linear(hidden_dim * 2, labels_count)

    def forward(self, sentence, verb_indices):
        embeds = self.word_embeddings(sentence)
        lstm_out, _ = self.lstm(embeds)

        verb_hidden_states = lstm_out[torch.arange(lstm_out.size(0)),
verb_indices]
        verb_hidden_states_expanded =
verb_hidden_states.unsqueeze(1).expand(-1, lstm_out.size(1), -1)
        concatenated_states = torch.cat((lstm_out,
verb_hidden_states_expanded), dim=2)

        label_space = self.hidden2label(concatenated_states)
        label_scores = F.log_softmax(label_space, dim=1)
        return label_scores
```

پس از بررسی چند مقدار مختلف، Epoch = 50 انتخاب شد؛ باقی پارامترها مشابه مقادیر پیشنهاد شده در صورت پروژه تعریف شدند.

مقدار f1 score برای مجموعه داده Validation

ابتدا مدل بدون هیچگونه وزن‌دهی در تابع loss آموزش داده شد. همانطور که در نوت‌بوک نیز قابل مشاهده است، مقادیر accuracy برای این مدل بسیار بالا است؛ دلیل این موضوع این است که تعداد زیادی کلاس از نوع ‘O’ داریم که تا حد خوبی به درستی قابل پیش‌بینی توسط مدل می‌باشند. با این حال این نوع آموش در معیاری مثل F1 Score که دقت در تمام کلاس‌ها را مد نظر قرار می‌دهد بد عمل می‌کند. این مقدار در حدود ۰.۱ می‌باشد.

یک راه حل این است که فرکانس حضور هر کلاس در طول داده‌های آموزش را بدست آورده و سپس معکوس این فرکانس را به عنوان وزن در تابع loss بدهیم. در این صورت کلاس‌هایی که بیش‌تر در داده‌ها حضور دارند وزن کم‌تری داشته و اهمیت کم‌تری در آموزش پیدا می‌کنند. با این روش مقدار F1 Score حدودا دو برابر و نزدیک ۰.۲ می‌شود.

بخش سوم: Gru Encoder Model

پیاده‌سازی مدل

این مدل به صورت زیر در PyTorch پیاده‌سازی شد:

```
class GRUSemanticRoleLabeler(nn.Module):
    def __init__(self, embedding_dim, hidden_dim, vocab_size, labels_count):
        super(GRUSemanticRoleLabeler, self).__init__()

        self.hidden_dim = hidden_dim
        self.word_embeddings = nn.Embedding(vocab_size, embedding_dim)
        self.gru = nn.GRU(embedding_dim, hidden_dim, batch_first=True)
        self.hidden2label = nn.Linear(hidden_dim * 2, labels_count)

    def forward(self, sentence, verb_indices):
        embeds = self.word_embeddings(sentence)
        gru_out, _ = self.gru(embeds)

        verb_hidden_states = gru_out[torch.arange(gru_out.size(0)),
verb_indices]
        verb_hidden_states_expanded =
verb_hidden_states.unsqueeze(1).expand(-1, gru_out.size(1), -1)
        concatenated_states = torch.cat((gru_out,
verb_hidden_states_expanded), dim=2)

        label_space = self.hidden2label(concatenated_states)
        label_scores = F.log_softmax(label_space, dim=1)
        return label_scores
```

مزیت LSTM به RNN چیست؟

یکی از مزیت‌های اصلی LSTM نسبت به RNN-ها داشتن قدرت بیش‌تر در به حافظه سپردن کلمات در مکان‌های بسیار عقب‌تر در دنباله می‌باشد. شبکه‌های LSTM مشکل Vanishing Gradient و ارتباط بلند مدت دنباله‌ها را تا حدی بهتر از RNN-ها هندل می‌کند. Gradient Vanishing به این مشکل اشاره می‌کند که در مواردی که یک کلمه

به کلمه دیگر در دنبال وابسته است که بسیار دورتر از آن ظاهر می‌شود، شبکه به خوبی نمی‌تواند این ارتباط را تشخیص دهد و در طول زمان این ارتباط از state آن حذف می‌شود.

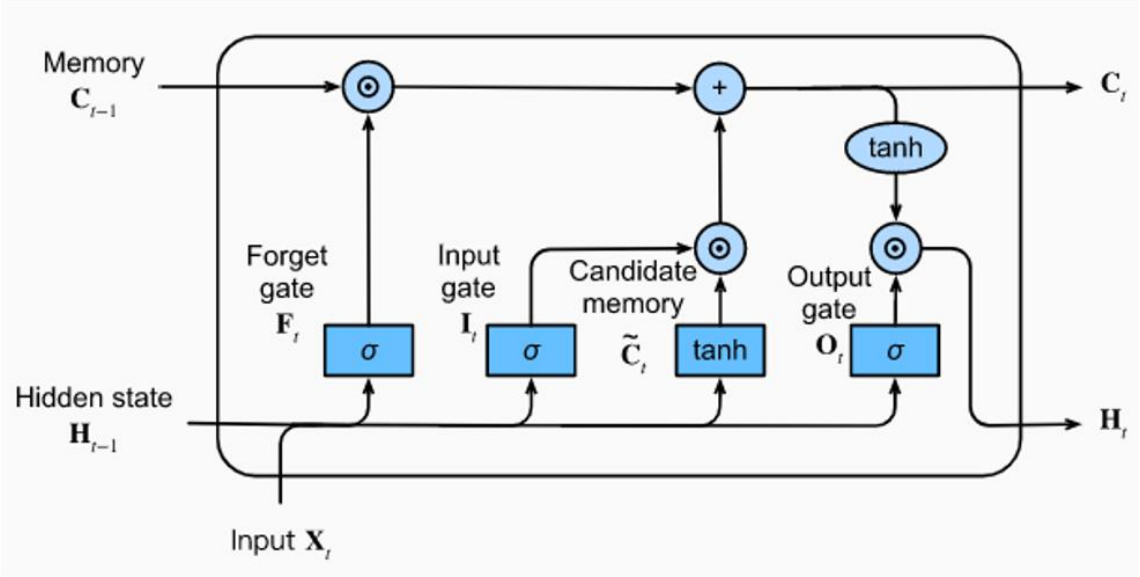
تفاوت LSTM و GRU

اولین و مهم‌ترین تفاوت این دو مدل در نحوه پیاده‌سازی معماریشان می‌باشد. مدل LSTM از یک سری گیت‌های ورودی، فراموشی و خروجی برای به خاطر سپردن دنباله‌های بلند مدت استفاده می‌کند؛ این در حالی است که مدل GRU از یک معماری ساده‌تری استفاده می‌کند؛ معماری این مدل شامل گیت‌های آپدیت (تعیین می‌کند چه مقدار از دنباله قدیمی را به خاطر بسپرد) و ریست (تعیین می‌کند چه مقدار از دنباله قدیمی را فراموش کند) می‌باشد.

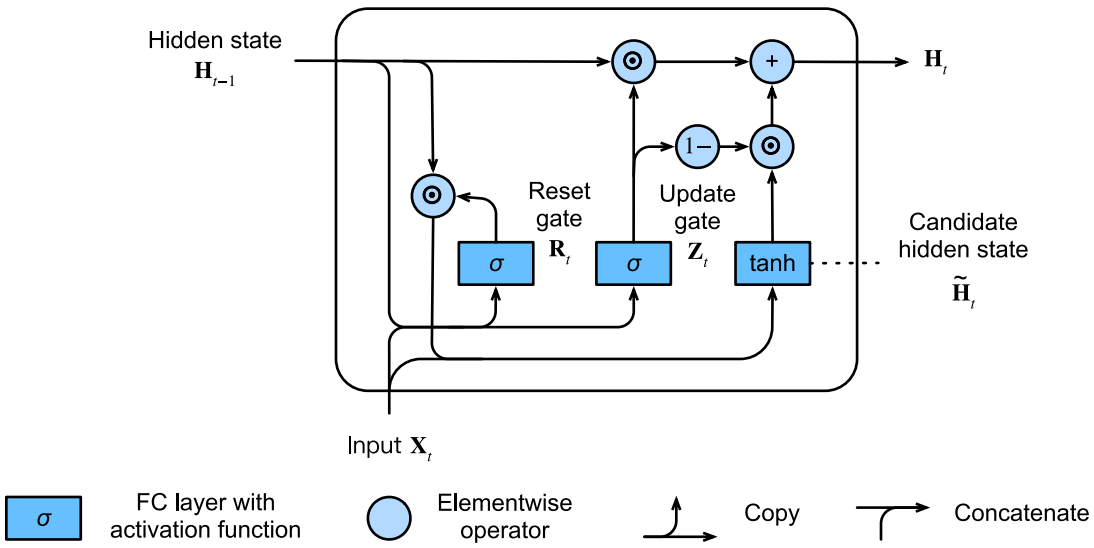
یکی دیگر از تفاوت‌های این دو مدل در تعداد پارامترها می‌باشد؛ مدل LSTM معمولا پارامترهای زیادتری نسبت به GRU دارد. این باعث می‌شود مدل GRU پیاده‌سازی ساده‌تری داشته و نسبت به overfitting مقاوم‌تر باشد.

مدل LSTM به خاطر طراحی پیچیده‌تری که دارد می‌تواند روابط پیچیده‌تری در متن را تشخیص و پیشبینی کند. با این حال مدل GRU نیز می‌تواند تا حد خوبی روابط را تشخیص داده و همچنان مدل محبوبی در تسک‌ها NLP می‌باشد.

در نتیجه پیچیده‌تر بودن مدل LSTM باعث شده تا آموزش آن سخت‌تر و زمان‌برتر باشد.



عکس ۱ معماری مدل LSTM که در آن گیت‌های ورودی، فراموشی و خروجی مشخص شده‌اند.



عکس ۲ معماری مدل GRU که در آن گیت‌های ریست و آپدیت مشخص شده‌اند.

دلیل چسباندن hidden state فعل به بقیه توکن‌ها

در این تسک ما قصد داریم تا تگ‌هایی را تشخیص دهیم که تا حد خوبی به خود فعل مربوط هستند. Hidden State فعل نیز به ما می‌تواند کمک کند تا این تگ‌ها را بهتر تشخیص دهیم. در اصل این تسک نه مثل یک دنباله در زمان بلکه باید به صورتی با آن برخورد شود که کل جمله را یک جا داریم و از اطلاعات مکان‌های دیگر دنباله (و نه فقط کلمات قبلی) باید استفاده کنیم تا کیفیت مدل را افزایش دهیم. این کار تا حدی به مثابه پیاده‌سازی دو لایه LSTM می‌باشد که در آن ما اطلاعات کل جمله را به لایه بعدی می‌دهیم.

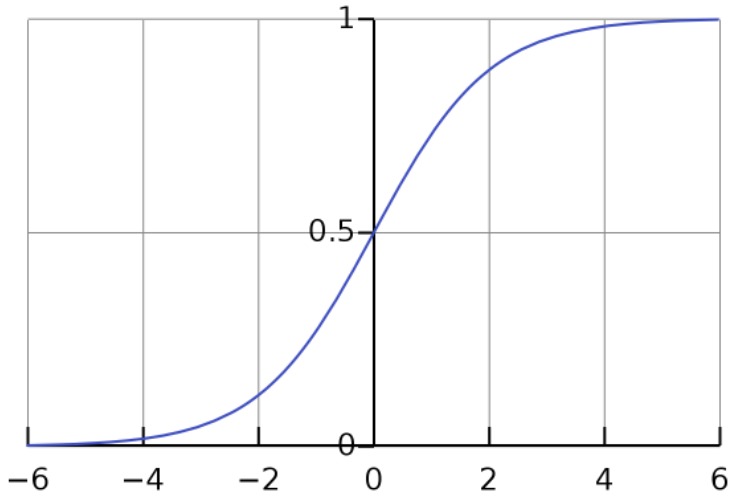
بنابراین داشتن اطلاعات راجع به خود فعل و انتقال آن به بخش‌های دیگر جمله می‌تواند به ما کمک کند تا پیش‌بینی‌های بهتری در تگ‌ها بدست آوریم.

روش حل مشکل محو شدگی گرادیان

یکی از عملیات‌های مهم در شبکه‌های عصبی back propagation می‌باشد که در حین آموزش دادن شبکه استفاده می‌کنیم. هنگامی که در این عملیات ما سعی می‌کنیم تا گرادیان تابع loss را نسبت به وزن‌های شبکه محاسبه کنیم، ممکن است مقادیر آن بسیار کوچک و یا بسیار بزرگ شوند؛ در صورتی که این مقادیر خیلی بزرگ باشند، exploding gradients رخ می‌دهد و هنگامی که این مقادیر خیلی کوچک شوند، vanishing gradient یا همان محوشدگی گرادیان رخ می‌دهد.

در حالت محوشدگی گرادیان، از آن‌جایی که هر چه عقب‌تر می‌رویم مقدار مشتق تابع نیز کم‌تر می‌شود، لایه‌های ابتدایی نرخ یادگیری پایین‌تری نسبت به لایه‌های اولیه پیدا می‌کنند؛ این مشکل اساسی محوشدگی گرادیان می‌باشد. برخی از راه حل‌های موجود به شرح زیر می‌باشند:

- استفاده از تابع ReLu: توابع ReLu در نواحی‌ای که توابعی مثل Sigmoid اشباع شده‌اند خطی بوده و منجر به مشتقی برابر با باقی نواحی تابع می‌شود. برای درک بهتر به عکس زیر توجه کنید:



عکس ۳ تابع Sigmoid

در شکل بالا، تابع Sigmoid را مشاهده می‌کنیم. در مقادیری که کوچک باشند، مثلا کم‌تر از پنج، مقدار گرادیان بسیار کوچک شده و در نهایت می‌تواند در گام‌های بعدیِ back propagation منجر به محوشوندگی گرادیان شود. اما در تابع ReLu و توابع مشابه آن، مقدار گرادیان در طول بازه ثابت بوده و تفاوتی نمی‌کند.

- نرمالایز کردن Batch: در صورتی که نمی‌خواهیم تابع‌های فعالسازی را تغییر دهیم، می‌توانیم از این روش برای بهوددهی مشکل استفاده کنیم.
- کم کردن مقدار learning rate: در صورتی که مقدار learning rate را بدون در نظر گرفتن موارد قبلی افزایش دهیم، مدل می‌تواند خیلی سریع‌تر به یک نقطه همگرا شود. با کم کردن این مقدار شانس رخداد محوشوندگی گرادیان را می‌توانیم کاهش دهیم؛ با این حال واضحا زمان بیش‌تری برای آموزش مدل صرف خواهد شد.
- انتخاب شیوه مقداردهی اولیه برای وزن‌های شبکه: با روش‌هایی مثل Xavier Initialization می‌توانیم شانس رخداد محوشوندگی گرادیان را کاهش دهیم ولی با این حال تضمینی در این مورد وجود ندارد.

بخش چهارم: Encoder-Decoder Model

محدودیت‌های روش QA

محدودیت اصلی در این روش نوع تولید سوالات است. از آنجایی که طبق یک سری پترن‌های از پیش تعیین‌شده سوالات را تولید می‌کنیم، تنوع سوالات بسیار کم بوده و ممکن است نتوانیم بسیاری از مفاهیم زبانی که در این

پترن‌ها نمی‌گنجند را تشخیص دهیم. بنابراین این روش ممکن است پترن‌هایی که در داده‌های آموزش دیده نشده‌اند را به خوبی پوشش ندهد. همچنین این روش ممکن است نتواند تنوع دامنه متون را به خوبی آموزش دهد که منجر به overfit شدن به یک نوع از متون خواهد شد.

دلیل استفاده از تگ‌های <s> و </s>

یک دلیل اصلی استفاده از این تگ‌ها این است که بتوانیم کلمه شروع و پایان را تشخیص دهیم. مدل می‌تواند کلماتی که احتمال بیش‌تری برای شروع پاسخ دارند را برابر احتمال این در نظر بگیرد که یک کلمه چقدر محتمل است بعد از تگ <s> بیاید. همچنین احتمال اینکه بعد یک کلمه تگ پایان یعنی </s> بیاید برابر این است که یک کلمه چقدر محتمل است که آخرین کلمه پاسخ باشد. به عبارتی دیگر این دو تگ به ما کمک می‌کنند تا شروع و پایان بازه پاسخ‌ها را بهتر تشخیص دهیم.