Problem 1:

(a

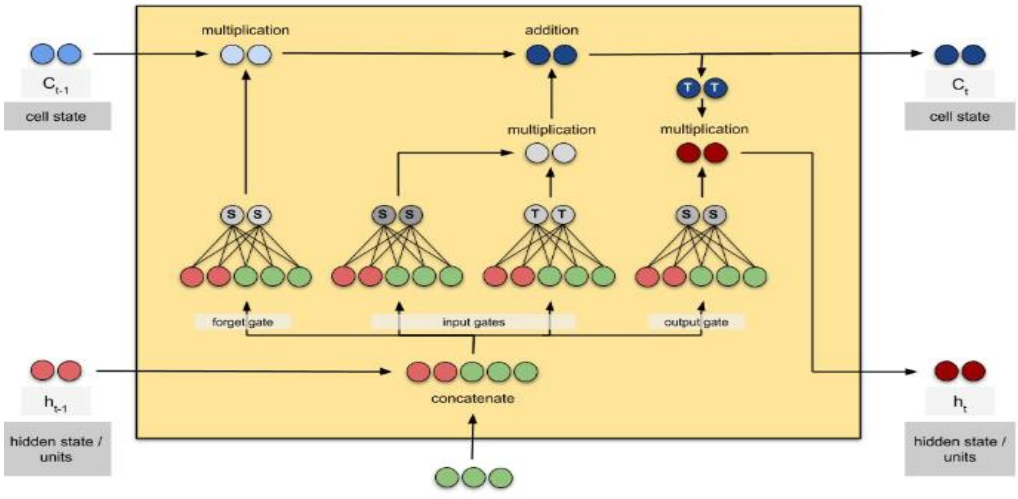
تمام شبکه های RNN یک حلقه فیدبک در لایه بازگشتی دارند که باعث حفظ اطلاعات در مموری در طول زمان میشود. اما برای vanilla RNN که شبکه های بازگشتی ساده هستند آموختن وابستگی های زمانی بلند مدت مشکل است و دلیل آن این است که گرادیان تابع ضرر در طول زمان به صورت نمایی کاهش میابد که به آن محوشدگی گرادیان هم میگویند. محوشدگی گرادیان به دلیل عمیق شدن شبکه اتفاق می افتد که برای مثال در پردازش دنباله یک جمله باعث میشود پارامتر های مربوط به کلمات ابتدایی یک جمله به درستی به روزرسانی نشوند و شبکه این کلمات را فراموش کند درحالیکه ممکن است کلمات ابتدایی یک جمله مهم باشند. شبکه های LSTM نوعی از شبکه های RNN هستند که از واحد های اضافی استفاده میکنند که به آن ها سلول های حافظه میگویند و این سلول ها اطلاعات را در حافظه برای دنباله های با طول بلند نگه میدارند. اینکه کدام اطلاعات در حافظه باید نگه داشته شوند و کدام یک فراموش شوند با مجموعه ای از گیت ها کنترل میشود:

$$\tilde{\boldsymbol{c}}^{(t)} = \tanh\left(\boldsymbol{W}_{ac}\boldsymbol{h}^{(t-1)} + \boldsymbol{W}_{xc}\,\boldsymbol{x}^{(t)} + \boldsymbol{b}_c\right)$$

$$\boldsymbol{c}^{(t)} = \boldsymbol{u}^{(t)}.\tilde{\boldsymbol{c}}^{(t)} + \boldsymbol{f}^{(t)}.\boldsymbol{c}^{(t-1)}$$

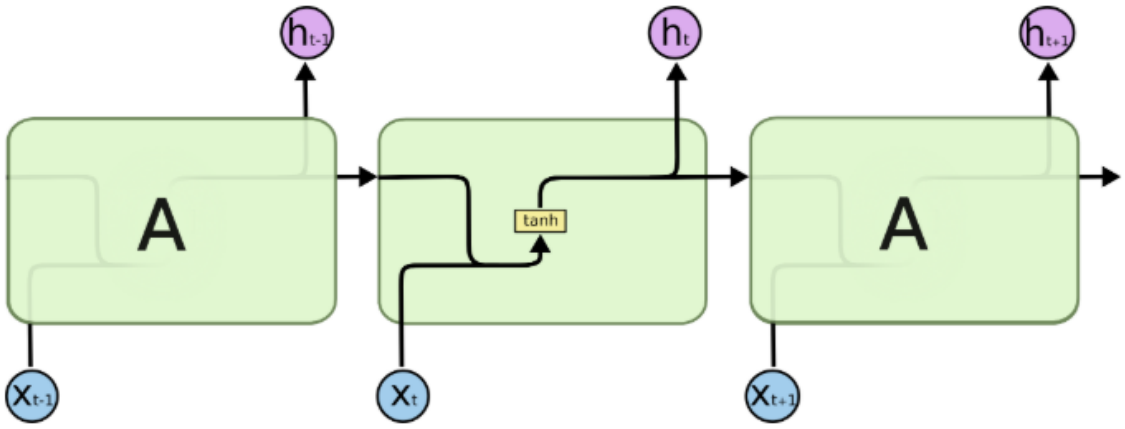
برای مثال بردار u و f تعیین میکنند به ترتیب چه مقدار از اطلاعات ورودی زمان فعلی و چه مقدار از اطلاعات گذشته حفظ شوند.

معماری شبکه LSTM و گیت هایی که در مورد آن صحبت کردیم به شکل زیر است:



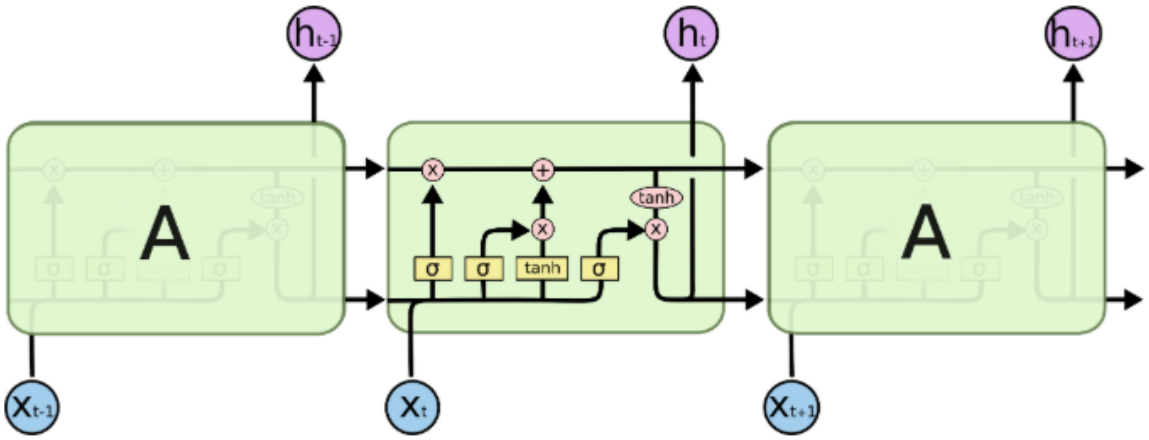
تفاوت معماری دو شبکه vallina RNN و LSTM در شکل زیر نشان داده شده است:

vallina RNN



The repeating module in a standard RNN contains a single layer.

LSTM:



The repeating module in an LSTM contains four interacting layers.

b)

در ابتدا از corpus ای که در اختیار داریم یک vocabulary میسازیم که کلمات یکتا را در خود نگه داری میکند. سپس باید متون را به واحد هایی در سطح کلمه یا کاراکتر شکست و در نهایت به این واحد ها به صورت ترتیبی یا افزایشی یک عدد تخصیص داده شود تا مدل بتواند آن ها را بفهمد. تا اینجای کار متن به

یکیسری عدد encode شده است که مدل بتواند با آن ها کار کند. اگر مثلا تسک مدل ما تولید جمله باشد جملاتی که تولید میشوند هم به صورت یکسری عدد هستند اما همانطور که اشاره کردیم تسک ما تولید جملات است پس باید اعداد را به کاراکتر تبدیل کنیم و سپس این کاراکتر ها را بهم متصل کنیم تا به جملات در سطح فهم و درک انسان تبدیل شود که به این کار decode کردن میگویند.

تا اینجای کار در مورد پیش پردازش و پس پردازش متون صحبت کردیم اما اگر بخواهیم در مورد مدل های sequence to sequence صحبت کنیم میتوان توضیحات زیر را اضافه کرد:

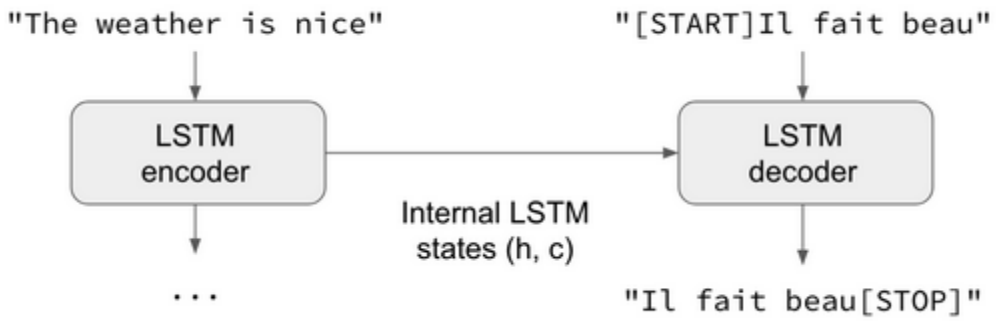
یادگیری Sequence-to-Sequence(Seq2Seq) آموزش مدل برای تبدیل دنباله از یک دامنه (برای مثال جملات انگلیسی) به دنباله در دامنه ای دیگر (برای مثال همان جمله که به فرانسوی ترجمه شده است) میباشد. مثال:

```
"the cat sat on the mat" -> [Seq2Seq model] -> "le chat etait assis sur le tapis"
```

یادگیری Seq2Seq میتواند در ترجمه ماشینی یا Question answering استفاده شود و در حالت کلی برای هر زمانی که نیاز به تولید جمله باشد قابل استفاده است. برای انجام این تسک راه های متعددی وجود دارد که یکی از آن ها استفاده از RNN ها است.

در حالت کلی دنباله ورودی و خروجی طول های متفاوتی دارد (برای مثال میتوان به ترجمه ماشین اشاره کرد) و تمام دنباله ورودی برای پیش بینی هدف نیاز است. معماری مدل به صورت کلی از دو قسمت تشکیل شده است:

۱) یک لایه RNN (یا استکی از این لایه ها) که به آن encoder گفته میشود. این قسمت دنباله ورودی را پردازش میکند و در نهایت internal state را برمیگرداند. ضمنا در encoder خروجی نادیده گرفته میشوند و فقط state بازیابی میشود و به آن context یا conditioning برای decoder در قدم بعدی گفته میشود.
۲) یک لایه RNN دیگر (یا استکی از لایه ها) که به آن decoder گفته میشود. این قسمت طوری آموزش میبیند که با استفاده از کاراکتر های قبلی، کاراکترهای بعدی از دنباله هدف را پیش بینی کند. در اینجا decoder از state vector های encoder به عنوان initial state (حالت اولیه) استفاده میکند تا بداند چه چیزهایی را باید تولید کند. به صورت موثر decoder یاد میگیر که target[t+1...] را تولید کند وقتی که target[...t] به آن داده شده باشد و روی دنباله ورودی مشروط باشد. شکل زیر معماری این مدل را نشان میدهد:



Problem 2:

a)

در static embedding ها مشکلات به شرح زیر هستند:

۱- در استاتیک امبدینگ ها ما با یک دیکشنری روبرو هستیم. یعنی وقتی به یک کلمه در متن برمیخوریم با استفاده از این دیکشنری بردار عدد متناسب با آن را برمیداریم. هر کلمه در این vocabulary ما یک بردار دارد. پس در اینجا ما با یک vocabulary روبرو هستیم که در آن کلمات یکتا قرار دارد. این کلمات یکتا از داده های train به دست آمده اند پس ممکن است ما در داده های test به کلمه ای بربخوریم که در train ندیده باشیم و در این حالت مدل نمیداند چه وکتوری از اعداد را به این کلمه ناشناخته تخصیص بدهد.

۲- در استاتیک امبدینگ ها کانتکست در نظر گرفته نمیشود. این در حالی است که کلمات براساس کانتکست میتواند معانی متفاوتی داشته باشد. بنابراین کار برای مدل سخت میشود و عملکرد مدل در تسک های معنایی بالا نیست.

۳- استاتیک امبدینگ cross domain نیست. برای مثال اگر corpus ما در دامنه مسائل مالی باشد، یک یک امبدینگ مثل word2vec بردار مربوط به کلمه bank را به بردار های مربوط به مسائل مالی و بانکی نزدیک میکند و معنی دیگر bank را که "حاشیه رودخانه" است در نظر نمیگیرد.

این در حالی است که برای مثال contextual embedding ها میتوانند از مدل هایی مبتنی بر ترنسفورمر به دست آیند. در این حالت امبدینگ ها با پاس دادن کل جمله به مدل پیش آموخته به دست می آیند. در اینجا هم یک vocabulary داریم ولی این vocabulary کانتکسچوال امبدینگ ها را در برنمیگیرد. امبدینگی که برای هر کلمه تولید میشود به بقیه کلمه ها در جمله داده شده وابستگی دارد. مدل های ترنسفورمر بر اساس

مکانیزم توجه کار میکنند و توجه یعنی به ارتباط یک کلمه با همسایگانش نگاه کنند. بنابراین برای یک کلمه داده شده امبدینگ استاتیک ندارد و این امبدینگ ها به صورت پویا از مدل پیش آموخته تولید میشوند.

برای مثال دو جمله زیر را در نظر بگیریم:

1. I will show you a valid point of reference and talk to the point.
2. Where have you placed the point.

امبدینگی که از مدل هایی مثل word2vec (static embedding) برای کلمه point به دست می آید در هر سه جایی که رخ داده است یکسان است در حالیکه با استفاده از مدل هایی مثل BERT (contextual embedding) سه امبدینگ متفاوت با توجه به جایی که رخ داده اند به دست می آید.

b)

BERT pre-training objectives:

۱- Masked Language Modeling:

در این تسک یک کلمه جا افتاده در جمله پیش بینی میشود. در این حالت نیازی به برچسب نیست و میتوان یک کلمه از جمله ورودی را mask کرد. برای مثال اگر یک نمونه مانند Machine Learning is Super Cool در دیتاست داشته باشیم، برت در طول آموزش چنین ورودی را میگیرد:

[CLS] Machine [MASK] is Super Cool [SEP] Machine Learning Is Super Cool [EOS]

و هدف مدل پیش بینی آن کلمه جا افتاده خواهد بود. برت حدود ۱۵ درصد از جملات ورودی را mask خواهد کرد و از مدل میخواهد که این جاهای خالی را پیش بینی کند.

۲- Next Sentence Prediction:

از آنجایی که بسیاری از تسک های downstream مهم، در بردارنده ارتباط بین دو جمله هستند، برت هم روی چیزی پیش آموزش میبینند که به آن پیش بینی جمله بعدی گفته میشود.
هدف در اینجا این است که اگر دو جمله A و B داده شده باشند، بتواند پیش بینی کند که آیا B بعد از A آمده است یا خیر. بعد از ساختن داده برای این تسک، ورودی به این شکل در می آید:

[CLS] Sentence A [SEP] Sentence B [EOS]

وقتی این جمله به برت داده میشود، هدف برای توکن [CLS] این است که بازنمایی ای را یاد بگیرد که به ما اجازه دهد تشخیص بدهیم که آیا جمله B باید بعد از جمله A بیاید یا خیر.

مراجع:

اسلاید های درسی و لینک های زیر:

1) https://stats.stackexchange.com/questions/222584/difference-between-feedback-rnn-and-lstm-gru
2) https://blog.keras.io/a-ten-minute-introduction-to-sequence-to-sequence-learning-in-keras.html
3) https://stackoverflow.com/questions/62272056/what-are-the-differences-between-contextual-embedding-and-word-embedding
4) https://medium.com/analytics-vidhya/an-overview-of-the-various-bert-pre-training-methods-c365512342d8