کتابچه راهنمای NLP : خلاصه کلاس آموزشی CS۲۲۴N استنفورد

پردازش زبان طبیعی (NLP) یکی از مهمترین فن آوری های عصر اطلاعات است. درک مفاهیم پیچیده زبان نیز بخش مهمی از هوش مصنوعی است. برنامه های NLP در همه‌جا یافت میشوند زیرا مردم از آن برای ارتباطات استفاده میکنند: جستجو در وب، تبلیغات، ایمیل، خدمات به مشتری، ترجمه زبان، گزارش های رادیولوژی، و غیره. به تازگی، روشهای یادگیری عمیق در عملکردهای مختلف NLP کارایی بسیار بالایی کسب کرده اند. در این کتابچه سعی شده مفاهیم اساسی NLP به صورت مختصر بیان شود. مفاهیم آورده شده در این کتابچه عمدتا از کلاس آموزشی CS۲۲۴N استنفورد استخراج شده است. کتابچه پیش رو نتیجه کنجکاوی و درک من به عنوان یک دانش آموز است که مطمئنا خالی از اشکال نیست.

"با واژه ها بر مردم حکمرانی میکنیم." (بنجامین دیسرائلی)

https://alisterta.github.io/

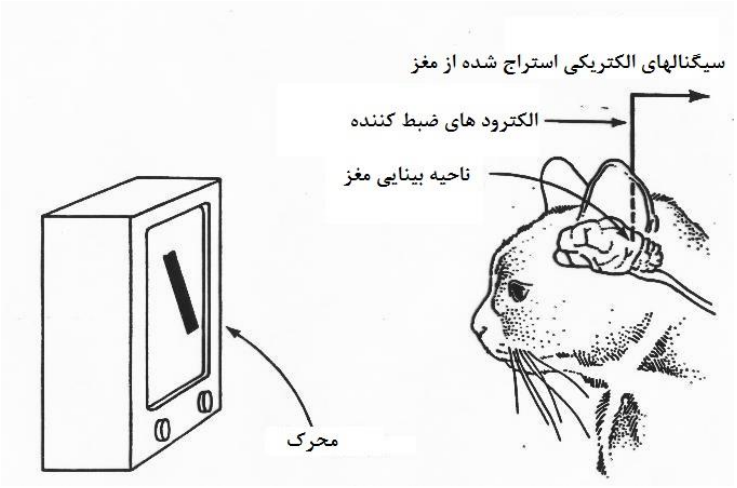
@Alisterta

فهرست مطالب

شبکه های عصبی پیچشی (کانولوشن)

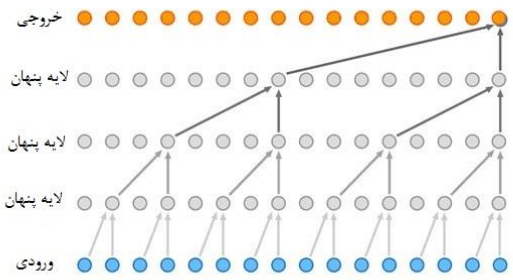
ناوردایی محلی[۱] و ترکیب پذیری[۲]

در ابتدا از مطالعات قشر بینایی گربه الهام گرفته شد [۱]، شبکه‌های عصبی کانولوشنی که از این به بعد آن را به اختصار CNNs می‌نامیم در بینایی ماشین برای کار برروی مشبک های باقاعده[۳] همانند عکس‌ها توسعه داده شدند [۲]. CNNs شبکه‌های عصبی پیش‌خوری[۴] هستند که هر نورون در یک لایه، ورودی را از نورون‌های مجاور لایه پیشین دریافت میکند. این همسایگی ها یا به عبارتی نواحی ارداک محلی[۵]، این امکان را به CNNs میدهند که الگوهای پیچیده‌تر را به یک روش سلسله‌مراتبی، با ترکیب ویژگی‌های سطح پایین ابتدایی و ویژگی‌های سطح بالا، بازشناسایی کنند.



شکل ۱: آزمایش بینایی هابل و ویزل بر روی گربه

این خاصیت، ترکیب‌پذیری نام دارد. به عنوان مثال، لبه‌ها میتوانند از پیکسل‌های خام تصویر استخراج شوند، در مقابل از لبه‌ها میتوان برای استخراج اشکال ساده استفاده کرد، و در نهایت از اشکال میتوان برای بازشناسایی اشیاء استفاده کرد. از سوی دیگر، موقعیت‌هایِ قطعی[۶] ویژگی‌ها در یک عکس اهمیت چندانی ندارند، فقط ضبط‌کردن موقعیت‌های نسبی برای ترکیب ویژگی‌های سطح بالا سودمند هستند. در نتیجه مدل بایستی بتواند یک ویژگی را صرفنظر از موقعیت آن در عکس، تشخیص دهد. این خاصیت ناوردایی محلی نامیده می‌شود. **ترکیب پذیری و ناوردایی محلی دو مفهوم کلیدی CNNs هستند.**



شکل ۲: معماری Wavenet دیپ مایند

---

۱ Local invariance

۲ Compositionality

۳ Regular grids

۴ Feedforward neural networks

۵ Local receptive fields

۶ Absolute positions

CNNs عملکرد بسیار خوبی در بینایی ماشین بدست آورده‌اند [۳]، اما از آنها همچنین میتوان در پردازش زبان طبیعی[۱] یا به اختصار NLP استفاده کرد. حقیقتا، در NLP، ویژگی های مرتبه بالا (ان-گرم[۲]) میتوانند از ویژگی‌های مرتبه پایین همانند بینایی ماشین، ساخته شوند که ترتیب نه در سطح متن، بلکه در سطح محلی ضروری است.(به عنوان مثال: "پیشنهاد نمیشود"، " خوب است"، "بد هست"). در واقع، در تلاش برای تعیین مثبت یا منفی بودن یک نقد فیلم، واقعا اهمیت نمی دهیم که آیا "خوب است" در ابتدا و یا در پایان متن آورده شده است، ما فقط باید به این حقیقت دست یابیم که " است" مقدم بر"خوب" است و غیره. توجه داشته باشید که CNNs امکان کدگذاری وابستگی‌های برد-بلند[۳] را ندارند، و در نتیجه، برای برخی از کارها مثل مدلسازی زبان، جایی که وابستگی راه-دور[۴] اهمیت خاصی دارند، معماری‌هایی بازگشتی مثل LSTMs ارجعیت دارند.(البته لازم به ذکر است این مشکل CNNs برای اولین بار در مقاله WaveNet به میزان زیادی برطرف گردید.)

کانولوشن و تجمیع[۵]

هر چند در [۴] پیشنهاد میشود که لایه‌های کانولوشنی میتوانند مستقیما برروی یک‌دیگر انباشته شوند، اما ساختار ابتدایی CNN، یک لایه کانولوشن است که به دنبال آن یک لایه تجمیع قرار گرفته است. در ادامه به صورت جزئی چگونگی فعل و انفعال این دولایه با استفاده از یک مثال در طبقه‌بندی متن ارائه خواهد شد.

ورودی

یک متن را می‌توان به عنوان یک ماتریس $A \in \mathbb{R}^{s \times d}$ نمایش داد، که s طول متن و d بُعد بردارهای تعبیه کلمه[۶] است. ازآنجا که متون مختلف اندازه هایی متغییر دارند و s بایستی ثابت باشد، ما متون بلند را به اولین s کلمه کوتاه کرده و متون کوتاه را با یک بردار خاص گسترش مرز[۷] میدهیم. بردارهای کلمه ممکن است به صورت تصادفی مقداردهی شوند یا به صورت ازپیش‌آموزش‌داده‌شده[۸] باشند. در مورد دوم

---

۱ Natural language processing

۲ N-gram

۳ Long-range dependencies

۴ Long-distance dependence

۵ Pooling

۶ Word embedding vectors

۷ Padding

۸ Pre-trained

(ازپیش‌آموزش‌داده‌شده)، بردارها می‌توانند در طول آموزش بروزرسانی شوند یا ثابت باقی بمانند [۵].در نظر گرفتن A به عنوان یک عکس می‌تواند گمراه کننده باشد، زیرا در اینجا فقط یک بعد مکانی[۱] وجود دارد.

در بینایی ماشین، عبارت کانال به عمقِ بعد اشاره دارد (با تعداد لایه‌های پنهان در شبکه اشتباه نشود). اگر با عکس‌ها سروکار داشته باشیم، دو بعد مکانی به اضافه عمیق داریم. ورودی یک تنسور به ابعاد (طول × عرض × تعداد کانال) خواهد بود، به عنوان مثال یک ماتریس دوبعدی، که هر درایه آن میتواند یک بردار با طول ۳ یا ۱ به ترتیب در یک عکس رنگی و عکس سطح خاکستری باشد.

توضیح بیشتر: فضای پنهان

کلمه latent به معنای پنهان است، و در اکثر مواقع در یادگیری ماشینی به همین معنا اشاره دارد. در مدل های رمزنگار که بعداً به آنها خواهیم پرداخت، رمزنگار یک داده با ابعاد بالای ورودی را به یک لایه گلوگاه تبدیل می‌کند، جایی که تعداد نورون ها بسیار کمتر است. سپس رمزگشا، ورودی رمزنگاری شده را به شکل اولیه ورودی تبدیل میکند. فضای پنهان فضاییست که دادها در لایه گلوگاه قرار میگیرند. فضای پنهان حاوی یک بازنمایی فشرده از ورودی است، و تنها اطلاعاتی است که رمزگشا اجازه دارد از آن برای بازتولید خروجی استفاده کند. برای اینکه خروجی بهینه تولید شود، شبکه بایستی فقط ویژگی های که بیشترین ارتباط با ورودی را دارند، یاد بگیرد. در تعبیه کلمه هدف ما نگاشت دادن کلمات به فضای پنهانیست که کلمات با معانی مشابه در کنار یکدیگر قرار میگیرند.

لایه کانولوشن

لایه کانولوشن، عملیات خطی است که به دنبال آن تبدیل غیرخطی می‌آید. عملیات خطی شامل ضرب (عنصربه‌عنصر) هر نمونه از یک پنجره یک بعدی که بر روی متن ورودی توسط یک فیلتر[۲] اعمال می‌شود، است، که به عنوان یک ماتریس از پارامترها نمایش داده می‌شود. فیلترها، همانند پنجره، فقط یک بعد مکانی دارند، اما به طور کامل در طول عمق ورودی بسط داده می‌شود (d). اگر h اندازه پنجره باشد، ماتریس پارامتر W درارتباط با فیلتر، از این رو به $\mathbb{R}^{h\times d}$ تعلق دارد. W به صورت تصادفی مقداردهی میشود و در طول آموزش یادگرفته می‌شود.

نمونه‌های پنجره اعمال‌شده بر روی ورودی، نواحی[۳] یا میدان‌های تاثیر[۴] نامیده می‌شوند. تعداد $\frac{(s-h)}{stride+1}$ از آنها وجود دارد، که stride به تعداد کلماتی که در هر گام پنجره را بر روی آنها می‌لغزانیم، اشاره دارد. با stride یک، تعداد $s-h+1$ میدان تاثیر وجود دارد. خروجی لایه کانولوشن برای یک فیلتر، یک بردار $o \in \mathbb{R}^{s-h+1}$ است که عناصر آن به صورت زیر محاسبه می‌شوند:

---

۱ Spatial dimension

۲ Filter

۳ Regions

۴ Receptive fields

$$o_i = W . A[i: i + h - ۱, :]$$

که $A[i: i + h - ۱, :] \in \mathbb{R}^{h \times d}$ ناحیه ماتریس i اُم و . عملگریست که جمع ضرب ردیفی دو ماتریس را برمی‌گرداند. توجه داشته باشید برای یک فیلتر، W مشابهی بر روی تمامی نمونه‌های پنجره، صرف‌نظر از موقعیت قرارگیری آنها در متن اعمال میشود. به عبارت دیگر، پارامترهای فیلتر در بین میدان‌های تاثیر به اشتراک گذاشته می‌شوند. **این دقیقا همان چیزیست که خصوصیت ناوردایی مکانی را به مدل می‌دهد، زیرا فیلتر آموزش دیده، الگوها را صرف‌نظر از موقعیت قرارگیری آنها در ورودی بازتشخیص میدهد. این خصوصیت تعداد کل پارامترهای مدل را به شکل شگفت آوری کاهش میدهد.**

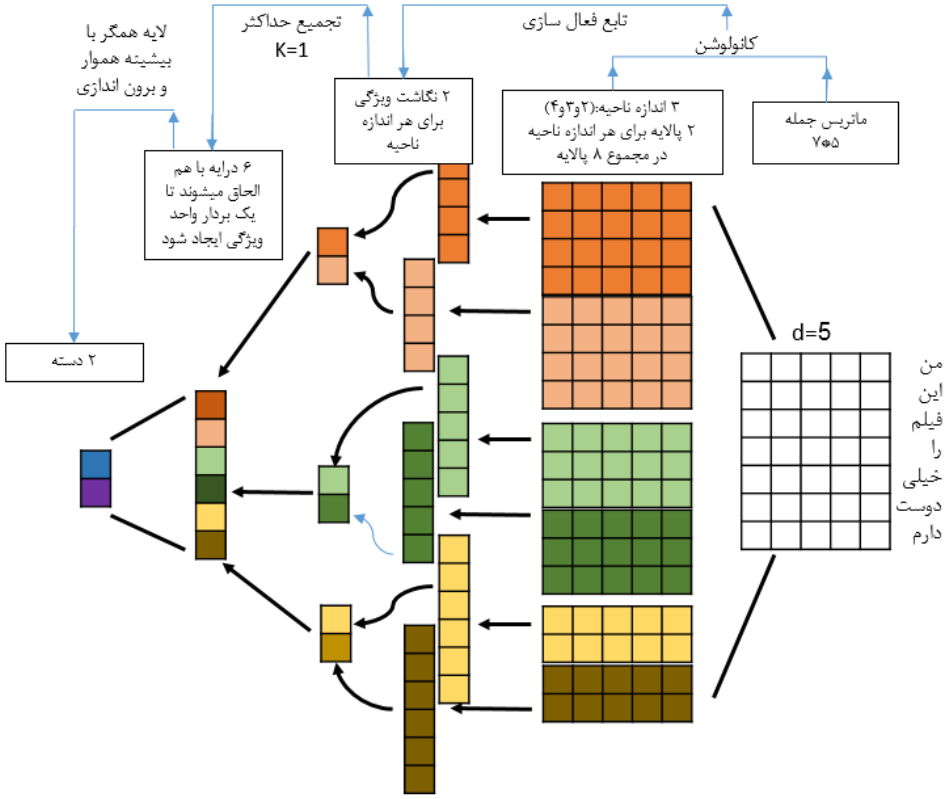
سپس یک تابع فعال سازی غیرخطی f مثل یکسوساز[۱] $(max(۰, x))$ یا تانژانت هذلودی[۲] $\frac{e^{۲x}-۱}{e^{۲x}+۱}$ به صورت عنصر به عنصر بر روی o اعمال می شود (یکسوساز نسبت به تانژانت هذلولی، بهتر با پدیده ناپدیدشدن گرادیان مقابله میکند، و در نواحی اعداد بیشتر از صفر، گرادیان همواره ثابت است، از سوی دیگر در هنگام استفاده از تانژانت هذلولی گرادیان کوچک و کوچکتر میشود)، که چیزی به نام نگاشت ویژگی[۳] $c \in \mathbb{R}^{s-h+۱}$ را برمی‌گرداند:

$$c_i = f(o_i) + b$$

که $b \in \mathbb{R}$ سوءگیری[۴] با قابلیت آموزش است.

برای طبقه‌بندی جمله کوتاه، بهترین اندازهای ناحیه معمولا بین ۱ و ۱۰ هستند، و در عمل، n_f فیلتر ($n_f \in [۱۰۰,۶۰۰]$) بر روی هر ناحیه اعمال می‌شوند تا مدل را قادر سازند ویژگی‌های متفاوت و مکمل برای هر ناحیه یاد بگیرد [۶].

از آنجا که هر فیلتر یک نگاشت ویژگی را تولید میکند، هر ناحیه، در یک فضای n_f بعدی تعبیه



شکل ۳: معماری CNN برای دسته بندی جملات کوتاه

---

[۱] ReLU

[۲] Tanh

[۳] Feature map

[۴] Bias

می‌شود. افزون بر این، استفاده از نواحی با اندازه متغیر کارایی مدل را افزایش میدهد [۶]. در این حالت شاخه های موازی متفاوتی تولید میشود (برای هر اندازه ناحیه، یکی)، و خروجی ها بعد از تجمیع، همانطور که در شکل ۳ نمایش داده شده، با هم الحاق می‌شوند. کارایی و هزینه مدل با n_f افزایش پیدا میکند و از یک نقطه مشخص به بعد، مدل شروع به بیش‌برارزش[۱] می‌کند.

رمزنگاری متن

همانطور که در شکل ۳ نمایش داده شده، از یک دیدگاه کلی، معماری CNN هر کدام از نسخه های فیلتر شده ورودی را، به یک نورون در بردار ویژگی نهایی متصل میکند. این بردار را میتوان به عنوان نسخه رمزنگاری شده متن ورودی دید. این درواقع درونمایه مدل است که ما به آن علاقه‌مند هستیم. مابقی معماری وابسته به کار مورد نظر است.

لایه بیشینه‌هموار[۲]

از آنجا که در اینجا هدف، دسته‌بندی کردن متون است، یک تابع بیشینه‌هموار برروی متن رمزنگاری شده اعمال، تا احتمال دسته خروجی بدست آید. هر چند، کارهای مختلف از معماری های متفاوت استفاده می‌کنند: به عنوان مثال برای تعیین اینکه دو جمله معادل یکدیگر هستند، ترجمه یا خلاصه سازی، میتوان از یک مدل رمزنگار CNN که بر روی ورودی اعمال می‌شود و یک رمز گشا LSTM که خروجی را تولید می‌کند، به صورت یکپارچه استفاده کرد [۷].

برگردیم به دسته‌بند خودمان، تابع بیشینه‌هموار یک بردار $x \in \mathbb{R}^k$ را به یک بردار از اعداد نقطه اعشار که مجموع‌شان یک است تبدیل می‌کند:

$$softmax(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{k} e^{x_i}}$$

در حالت دسته‌بندی دودویی، به جای داشتن دو نورون در لایه خروجی یا بیشینه‌هموار، جایی که هر نورون یکی از دسته ها را نمایش میدهد، میتوان از یک لایه خروجی با یک نورون به همرا تابع سیگموئید[۳] $\sigma(x) = \frac{1}{1+e^{-x}}$ استفاده کرد. در این حالت نورون، احتمال خروجی تعلق داشتن به یکی از دو دسته را تولید میکند و تصمیم گیری با توجه به اینکه خروجی تولید شده $\sigma(x)$ بزرگتر یا کوچکتر از ۰.۵ است، انجام میشود، این دو متد مشابه هستند ($\frac{1}{1+e^{-x}} = \frac{e^{x}}{e^{x}+e^{0}}$). بنابراین، یک لایه یک-نورون سیگموید را همچنین به عنوان یک لایه دو-نورون بیشینه‌هموار دید که یکی از نورون های آن هرگز فعال نمیشود و خروجی آن همواره صفر است.

---

[۱] Overfitting

[۲] Softmax

[۳] Sigmoid

تعداد پارامترها

تعداد پارامترهایِبا قابلیت آموزش برای CNN مان، جمع عبارات زیر است:

ماتریس تعبیه کلمه (اگر ثابت نباشند): $(v+1) \times d$، که v اندازه واژگان است. ما همچنین یک ردیف برای بردارِ گسترش‌مرز-صفر[1] اضافه میکنیم.

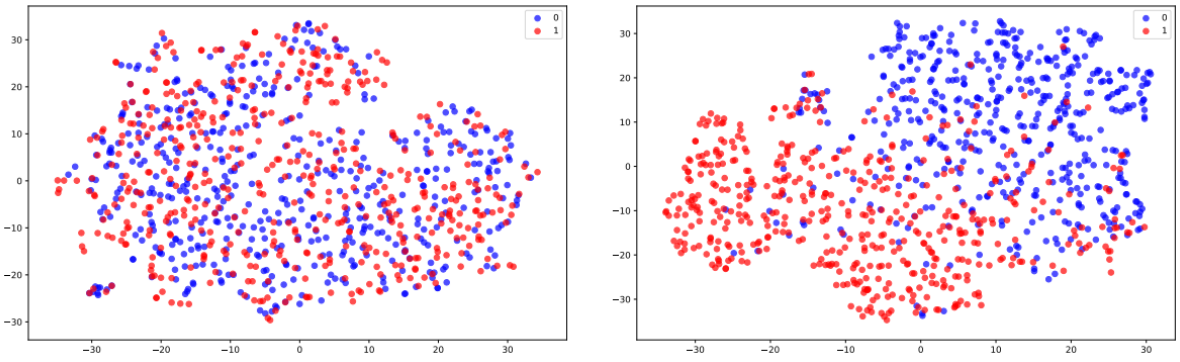
لایه کانولوشن: $h \times d \times n_f + n_f$ (تعداد درایه در هر فیلتر در تعداد فیلتر ها، به اضافه سوءگیری).

لایه بیشینه‌هموار: $n_f \times 1 + 1$ (لایه تماما متصل[2] با یک خروجی ۱-بعدی و ۱ سوءگیری).

تصویرسازی و درک بازنمایی درونی و پیش‌بینی

رمزنگاری متن

یک راه ساده و سریع برای تایید اینکه مدل‌مان به صورت موثر آموزش می‌بیند چک کردن این موضوع است که آیا بازنمایی درونی متن با عقل جور در میاید یا خیر. بردار ویژگی که به لایه بیشینه‌هموار تغذیه شد را به خاطر بیاورید، آن را میتوان به عنوان یک بردار رمزنگاری n_f بعدی از متن ورودی دید. با گرفتن خروجی این لایه برای یک زیرمجموعه از نمونه های آموزشی و تبدیل برداری های متناظر آنها به یک نگاشت با ابعاد کمتر، میتوان متوجه شد که آیا هیچ همبستگی بین این بردارهای رمزنگاری شده و برچسب ها وجود دارد یا خیر. شکل ۴ و ۵ اثبات میکند که مدلمان قطعا در حال آموزش بازنمایی معنی داری از متون است.



شکل ۴ سمت چپ (نمایش ۱۰۰۰ نمونه از بردارهای رمزگذاری‌شده قبل از آموزش) شکل ۵ سمت راست (نمایش ۱۰۰۰ نمونه از بردارهای رمزگذاری‌شده بعد از۲ گام آموزش)

[1] Zero-padding

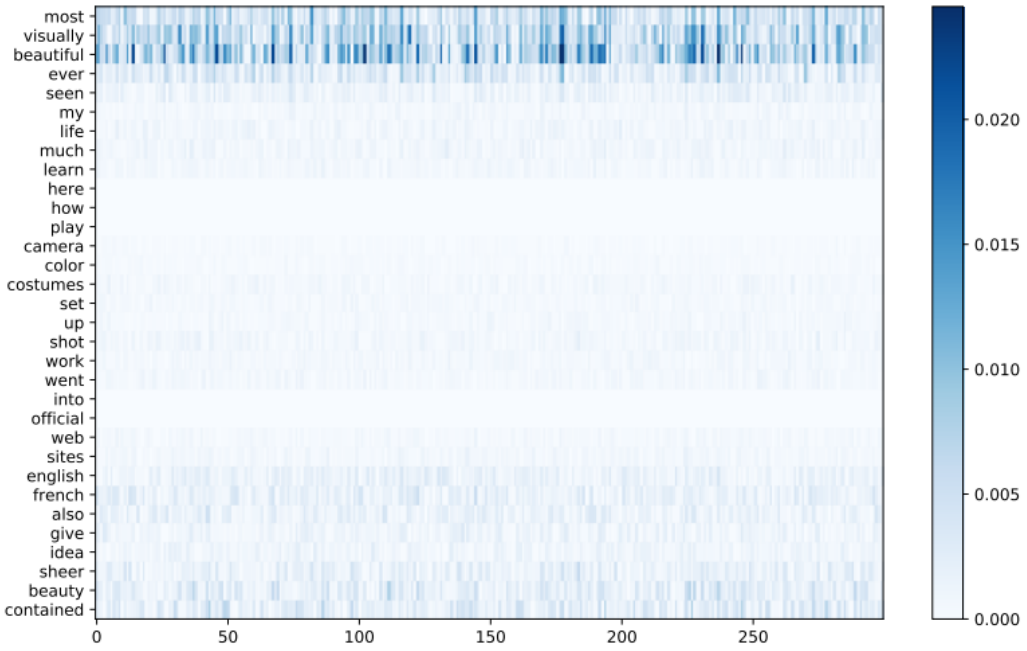
[2] Fully connected

شناسایی نواحی متضمن پیشگویی[1]

این متد در بخش ۳,۵ (جدول ۵ و ۶) مقاله نمایش داده شده است [۸]. به خاطر بیاورید که قبل از ازدست دادن اطلاعات با اعمال عملیات تجمیع، هر n_f فیلتر یک نگاشت ویژگی را ایجاد میکرد که درایه های آن خروجی کانولوشن فیلتر با ناحیه ادراک متناظر را نشان میداد. در نتیجه، هر ناحیه ادراک در یک فضای n_f بعدی جاسازی میشود. بدین ترتیب، بعد از آموزش، میتوان نواحی یک متن داده شده که محتملترین متضمن پیشگویی دسته خود هستند را با بررسی خروجی میانی مدل بدست آورد. به عنوان مثال، برخی از نقد های دیتابیس IMDB که بیشترین ناحیه متضمن پیشگویی برای نقد های منفی را دارند عبارتند از: "بدترین فیلم"، "پولت را هدر نده"، "بازی و فیلمنامه ضعیف"، "کیفیت تصویر افتضاح". از طرف دیگر، برخی از نواحی که دلالت بر مثبت بودن نقدها دارند عبارتند از: "موسیقی متن عالی"، " تصویربرداری زیبا"، "ماجراجویی جالب"، "پایانی کاملا رضایت بخش".

نگاشتهای برجستگی[2]

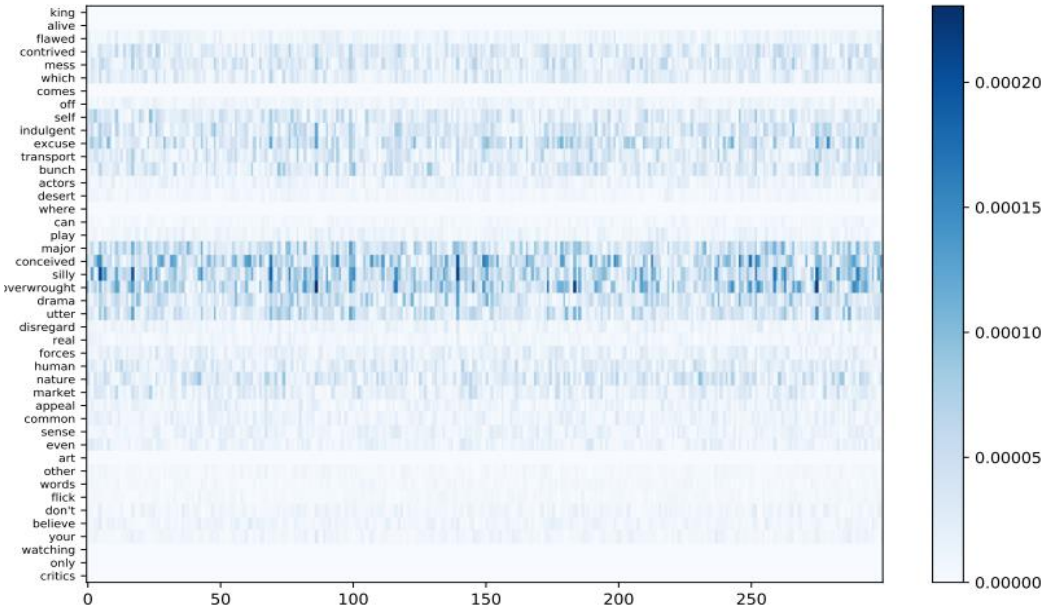
نگاشت برجستگی یک راه دیگر برای درک اینکه چگونه یک مدل پیش‌بینی را انجام میدهد، است. فرض کنید که تمام عکس های "دسته پرنده" شامل درخت و برگ است. چگونه میتوان مطمئن شد که CNN از پیکسل های مربوط به پرنده برای پیش‌بینی این دسته استفاده میکند و نه درخت و برگها؟ ایده بسیار ساده است، ما گرادیان دسته خروجی را نسبت به عکس ورودی محاسبه میکنیم. این متد به ما میگوید که چگونه تغییرات کوچک در پیکسل های عکس ورودی، خروجی دسته را تغییر میدهد. همه مقادیر مثبت در گرادیان ها به ما می‌گوید که یک تغییر کوچک در ورودی به آن پیکسل، ارزش



شکل ۶ : نگاشت برجستگی برای یکی از نمونه های آزمایش دیتابیس IMDB (برچسب مثبت)

[1] Predictive regions

[2] Saliency maps



شکل ۷ : نگاشت برجستگی برای یکی از نمونه های آزمایش دیتابیس IMDB (برچسب منفی)

خروجی را افزایش میدهد. این متد ابتدا در [۹] برای بینایی ماشین ارایه شد و توسط [۱۰] به NLP اعمال شد. ایده این است که عناصر ورودی متن $A \in \mathbb{R}^{s \times d}$ با توجه به تاثیر آنها بر روی پیش‌بینی درجه بندی شود. یک تخمین میتواند توسط طول مشتقات جزئی مرتبه اول خروجی مدل $CNN : A \rightarrow CNN(A)$ به نسبت ردیف a از A بدست آورده شود:

$$saliency(a) = |\frac{\partial(CNN)}{\partial a}|a$$

تفسیر این است، که ما شناسایی می کنیم که کدام کلمات در متن نیاز به تغییر حداقل دارند تا امتیاز دسته به شکل حداکثر تغییر کند. مشتق را میتوان با اجرای یک مرحله الگوریتم انتشار معکوس بدست آورد. دقت داشته باشید ما امتیاز دسته را انتشار معکوس میدهیم نه مقدار هزینه. در شکل ۶ و ۷ نمونه ای از نگاشت برجستگی دیتا بیس IMDB نمایش داده شده است.

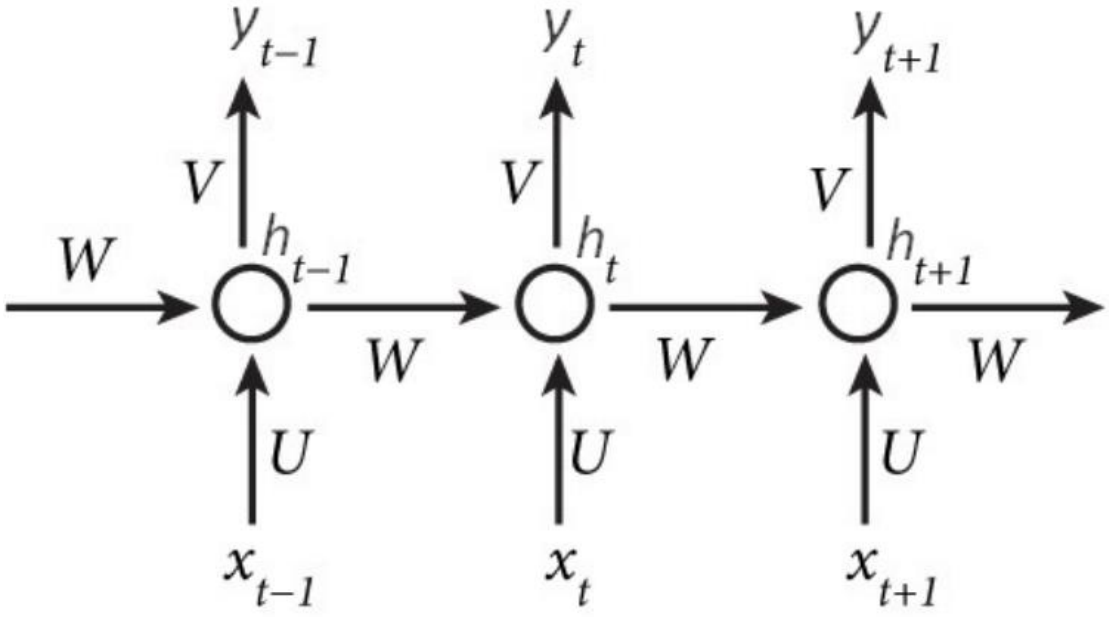
شبکه های عصبی بازگشتی [۱] یا به اختصار RNNs

در ابتدا یک تصویر کلی از چارچوب RNN ارائه خواهد شد و در ادامه به دو نمونه از این شبکه ها که به صورت فراگیر در عمل استفاده میشوند (LSTM و GRU) خواهیم پرداخت. در [۱۱] به صورت جزیی به شبکه های LSTM و RNN و کاربرد آنها پرداخته شده است. ما در اینجا به صورت کلی به آنها را بیان خواهیم کرد.

چهارچوب RNN

با آنکه CNNs مسلما برای کار با مشبک ها خوب هستند، RNNs به طور اختصاصی برای کار بر روی دنباله ها [۲] بوجود آمدند [۱۲]. به عنوان مثال سری زمانی، یا در NLP دنباله جملات. هر چند شبکه های CNN امکان ضبط برخی از اطلاعات ترتیبی را دارند اما آنها به الگوهای محلی محدود هستند، و وابستگی های بلند-برد نادیده گرفته میشوند [۱۳]. همانگونه که در شکل ۸ نمایش داده شده، یک RNN میتواند همانند یک زنجیره از لایه های عصبی ساده که برخی پارامتر ها را به اشتراک میگذارند، دید.

از یک دید بالا، یک RNN با یک لیست ترتیبی از بردار های ورودی $\{x_1, \dots, x_T\}$ به همراه یک وضعیت پنهان h. که با صفر مقدار دهی اولیه شده، تغذیه میشود، و یک لیست ترتیبی از وضعیت های پنهان $\{h_1, \dots, h_T\}$ را بر می‌گرداند. بردارهای خروجی ممکن است به عنوان ورودی برای واحد های RNN دیگر، زمانی که از معماری های عمیق استفاده میکنیم، عمل کنند (همانند شکل ۹). وضعیت های پنهان کم و بیش به عنوان حافظه "کوتاه مدت" مدل در نظر گرفته میشوند. توجه داشته باشید که هر نمونه آموزشی یک دنباله کامل $\{x_1, \dots, x_T\}$ برای خودش است، و ممکن است در ارتباط با یک برچسب مخصوص به خودش باشد. به عنوان



شکل ۸: سه گام از یک RNN باز شده، برگرفته شده از وبلاگ Denny Britz

[۱] Recurrent Neural Networks

[۲] Sequences

مثال در دسته‌بندی جملات کوتاه، دنباله ها فقط با یک برچسب در ارتباط هستند، در حالیکه در مدل سازی زبان، هر دنباله (کلمه) به دنبال پیش بینی دنباله بعدی(کلمه بعدی) است، هر برچسب متفاوت میباشد.

در هر گام T در دنباله، وضعیت پنهان h_t، به عنوان وضعیت پنهان پیشین h_{t-1} و بردار ورودی کنونی x_t به شکل بازگشتی زیر تعریف میشود:

$$h_t = f(Ux_t + Wh_{t-1} + b)$$

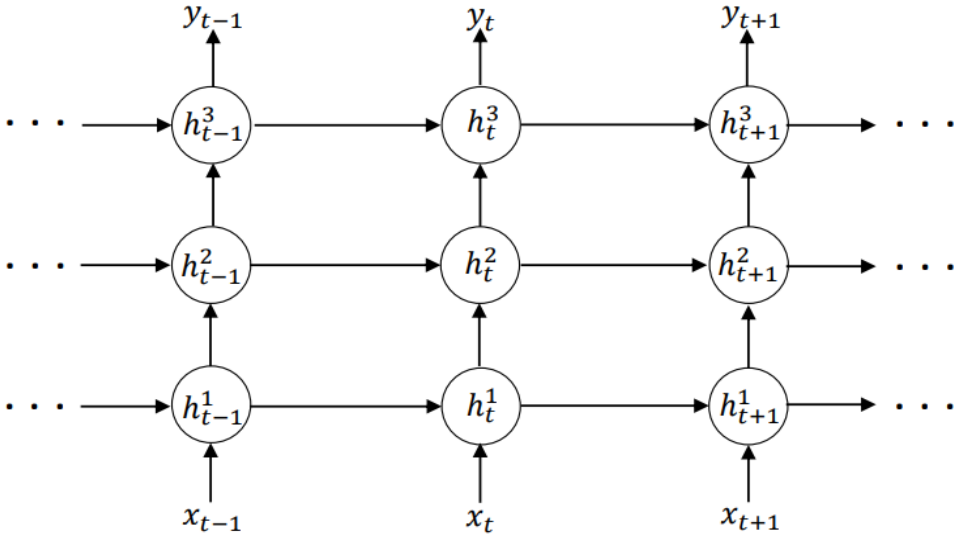
که f یک تابع غیرخطی مثل تانزانت هذلولی(که به صورت عنصر به عنصر اعمال میشود)، $x_t \in \mathbb{R}^{d_{in}}$ ، $U \in \mathbb{R}^{H \times d_{in}}$ و $W \in \mathbb{R}^{H \times H}$ ماتریس های پارامتر که در میان همه گام های زمانی به اشتراک گذاشته میشوند و h_t، h_{t-1} و b به $\mathbb{R}^H$ تعلق دارند. d_{in} میتواند اندازه واژگان باشد، اگر بردارهای one-hot یا تعبیه شده به عنوان ورودی به مدل داده شوند. H بُعد لایه پنهان است که معمولا $H \sim ۱۰۰$ است. هرچه این لایه بزرگتر باشد، میزان حافظه ای که مصرف میشود بیشتر خواهد بود و همچنین هزینه محاسبات نیز افزایش پیدا خواهد کرد.

بردار خروجی $y_t \in \mathbb{R}^{d_{out}}$، وضعیت پنهان کنونی $h_t \in \mathbb{R}^H$ با توجه به کاری که میخواهیم انجام دهیم، تبدیل میکند. برای دسته‌بندی، به صورت ذیل محاسبه میشود:

$$y_t = softmax(Vh_t)$$

که $V \in \mathbb{R}^{d_{out} \times H}$ ماتریس پارامتری است که بین همه گام های زمانی به اشتراک گذاشته میشود. d_{out} به تعداد دسته ها وابسته است. به عنوان مثال، برای یک دسته بند متن سه-دسته، $d_{out} = ۳$، برای مدل زبان در سطح کلمه $d_{out} = |V|$ است.

توجه کنید وقتی تعداد متعددی از لایه های RNN را به صورت عمودی بر روی یکدیگر سوار میکنیم(معماری RNN عمیق)، وضعیت های پنهان واحد های سطح پایین تر به صورت مستقیم به واحد های سطح بالاتر متصل می شوند و لایه خروجی در بالای این پشته قرار میگیرد (شکل ۷).



شکل ۹: سه گام از یک RNN عمیق باز شده. هر دایره یک واحد RNN را نمایش میدهد. وضعیت های پنهان در لایه‌های میانی(۱ و ۲) به عنوان ورودی به واحد مربوطه در لایه بالایی عمل میکنند.

مدل سازی زبان

مدل سازی زبان یک حالت خاص از کلاسه بندی است که مدل آموزش داده میشود تا کلمه یا کاراکتر بعدی در یک جمله را پیش بینی کند.در هر گام زمانی t، بردار خروجی، توزیع احتمال x_t بر روی همه کلمات/کارامترهای در واژگان را میدهد، که مشروط به کلمات/کاراکترهای پیشین در جمله است$P[x_{t-1}, \dots, x_t]$. در زمان آزمایش، احتمال یک دنباله کامل $\{x_1, \dots, x_T\}$ با ضرب همه احتمالات شرطی بدست می آید:

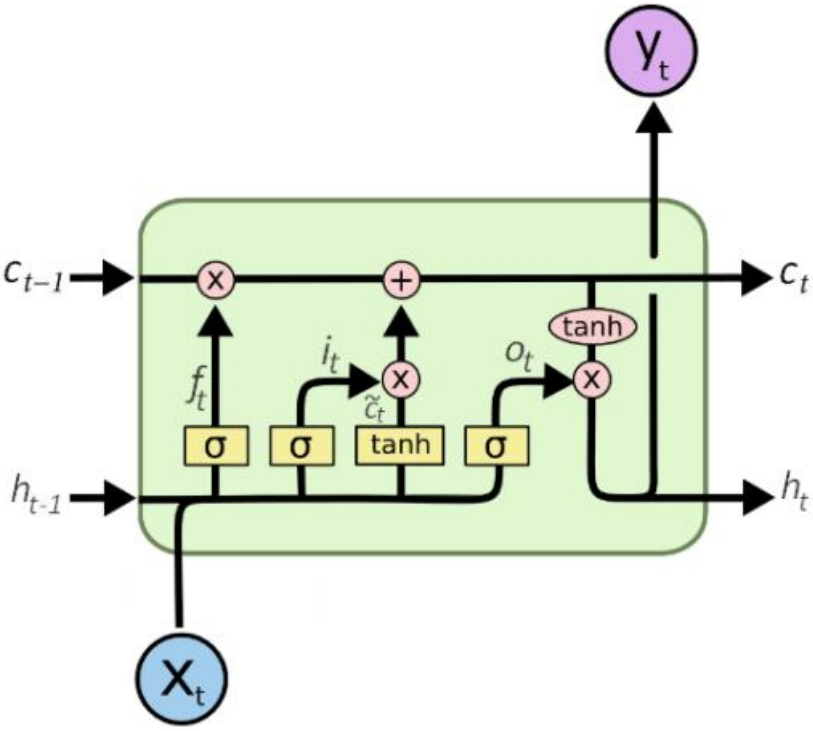
$$P[\{x_1, \dots, x_T\}| = P[x_1] \times \prod_{t=2}^{T} [x_t|x_{t-1}, \dots, x_1]$$

از مدل زبان همچنین میتوان برای متن با طول دلخواه، (تا رسیدن به یک توکن خاص (انتهای جمله) یا برای گام زمانی تعیین شده) استفاده کرد. برای یک مدل زبان بر مبنای کاراکتر، T میتواند به سادگی از ۲۰ با ۲۵ عبور کند. این امر به شدت تاثیرات نامطلوب که به مسئله ناپدید شدن و انفجار گرادیان[۱] معروف است را تقویت میکند، که مدل را از یادگیری وابستگی های بلند-برد باز میدارد. توجه کنید این مشکل همچنین میتواند با شبکه های عصبی پیشخور، مثل پرسپترون چند لایه[۲] تجربه شود، اما این مشکل با RNN به علت تمایل ریشه ای آنها به عمیق شدن، بدتر میشود.

یک واحد LSTM

در عمل، محققین در هنگام انتخاب $RNNs$ ، از واحد $LSTM$ یا GRU استفاده می‌کنند، زیرا این سلول ها به شکلی طراحی شده اند که مشکل انفجار/ناپدید شدن گرادیان را حل میکنند و اطلاعات را در بازه زمانی طولانی تری به خاطر میسپارند[۱۴].

همانگونه که در شکل ۱۰ نشان داده شده، دو چیزی که در واحد $LSTM$ نسبت به RNN تغییر یافته ۱) حضور وضعیت سلول[۳] c_t، که به عنوان حافظه آشکار[۴] عمل میکند،



شکل ۱۰: واحد LSTM برگرفته شده از وبلاگ Chris Colah

[۱] Vanishing and Exploding gradients

[۲] Multi-Layer Perceptron

[۳] Cell state

[۴] Explicit memory

و ۲) چگونگی محاسبه وضعیت های پنهان است. در $RNNs$ ساده، وضعیت پنهان با یک لایه $h_t = tanh(Ux_t + Wh_{t-1} + b)$ محاسبه میشود. اما در واحد LSTM، وضعیت پنهان با تعامل چهار لایه محاسبه میشود و به شبکه قابلیتی میدهد که اطلاعات خاصی در مورد عناصر پیشین در دنباله را فراموش یا به خاطر بسپارد.

لایه های درونی

این چهار لایه به شکل زیر هستند:

۱) لایه گیت فراموشی: $f_t = \sigma(U_f x_t + W_f h_{t-1} + b_f)$
۲) لایه گیت ورودی: $i_t = \sigma(U_i x_t + W_i h_{t-1} + b_i)$
۳) لایه محاسبه کاندید ارزش: $c_t = \sigma(U_c x_t + W_c h_{t-1} + b_c)$
۴) لایه گیت خروجی: $o_t = \sigma(U_o x_t + W_o h_{t-1} + b_o)$

به لطف تابع سیگموئید (σ) که به صورت عنصر به عنصر اعمال میشود، لایه های گیت ورودی، فراموشی و خروجی بردارهایی تولید میکنند که که تمام درایه های آنها بین ۰ و ۱ یا نزدیک به آن دو هستند. وقتی یکی از این لایه ها در بردار دیگری ضرب میشود، بدین شکل به عنوان یک فیلتر عمل میکند که فقط یک نسبت خاص از بردار را انتخاب میکند. دقیقا به این دلیل است که این لایه ها گیت نامیده میشوند. حالت غیر متعارف زمانیست که تمامی مقادیر یک بردار یک هستند که تمام بردار عبور خواهد کرد، یا تمام آن برابر با صفر است که هیچ کدام عبور نخواهند کرد. توجه کنید که گیت های فراموشی، ورودی و خروجی به شکل کاملا مشابه محاسبه میشوند و فقط تنها پارامتر ها متفاوت میباشند. هر چند این پارامتر ها در تمامی گام های زمانی به اشتراک گذاشته میشوند.

فراموش کردن/یادگرفتن

با به حساب آوردن یک نمونه آموزشی جدید x_t و وضعیت پنهان جاری h_{t-1}، لایه گیت فراموشی f_t تعیین میکند که چه میزان از وضعیت سلول قبلی c_{t-1} بایستی فراموش شود (چه میزان از حافظه بایستی آزاد شود)، از طرفی از ورودی مشابهی، لایه گیت ورودی i_t تصمیم میگیرد که چه میزان از ارزش کاندید c_t باید در حافظه نوشته شود، یا به عبارتی دیگر چه میزان از اطلاعات جدید بایستی یاد گرفته شود. ترکیب خروجی دو فیلتر، وضعیت سلول را بروزرسانی میکند:

$$c_t = f_t \circ c_{t-1} + i_t \circ c_t$$

که $\circ$ اشاره به ضرب عنصر به عنصر(ضرب درایه ای یا هادامار) دارد. با این روش، اطلاعات مهم با ورودی های جدید جایگزین نمیشوند و میتوان آنها را برای یک دوره زمانی بلند مدت نگاه داشت. در انتها، فعالساز h_t از حافظه بروزرسانی شده محاسبه میشود، و در لایه گیت خروجی o_t ضرب میشود:

$$h_t = \tanh(c_t) \circ o_t$$

گیت خروجی به واحد اجازه میدهد تا فقط زمانی فعال شود که اطلاعات موجود در حافظه برای گام زمانی جاری مرتبط باشند. در انتها همانند RNN ساده قبل، بردار خروجی به عنوان یک تابع از وضعیت پنهان جدید محاسبه میشود:

$$y_t = softmax(Vh_t)$$

قیاس RNN ساده

اگر تصمیم بگیریم که همه چیز درباره وضعیت پیشین را فراموش کنیم (همه عناصر f_t تهی هستند)، همه اطلاعات جدید را یاد بگیریم (همه عناصر i_t برابر یک هستند)، و تمام وضعیت سلول را که به گام زمانی بعدی میفرستیم را به حافظه بسپاریم (همه عناصر o_t برابر یک هستند)، داریم:

$$c_t = c_t = tanh(U_c x_t + W_c h_{t-1} + b_c)$$

و بدینگونه ما به واحد RNN ساده قبلی خودمان برمیگردیم که تنها تفاوت آن یک تابع تانژانت هذلولی اضافه است، در پایان ما به

$$h_t = \tanh(tanh(U_c x_t + W_c h_{t-1} + b_c))$$

به جای

$$h_t = \tanh(U_c x_t + W_c h_{t-1} + b_c)$$

که همانند RNN ساده هست میرسیم.

واحد بازگشتی گیت‌دار (GRU)

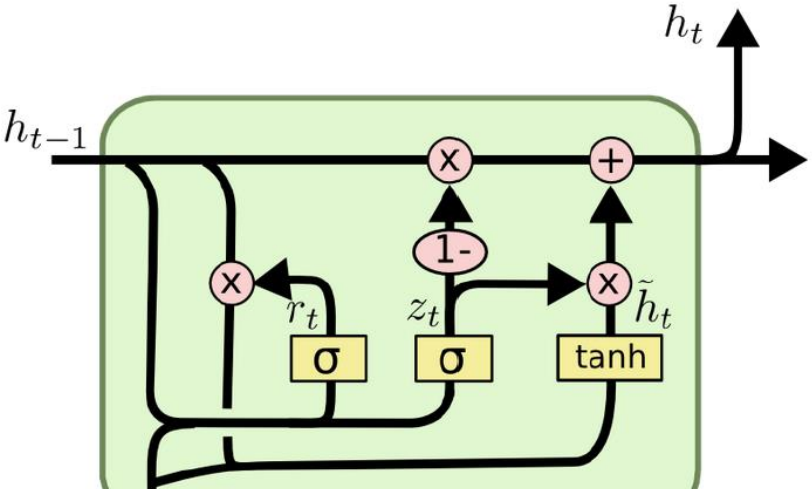
همانطور که در شکل ۱۱ نمایش داده شده، واحد GRU شکل ساده شده واحد LSTM است [۱۵] که فقط دو گیت (گیت بازنشانی[۱] و گیت بروزرسانی[۲]) دارد، و هیچ گونه حافظه آشکاری c_t وجود ندارد.

۱) لایه گیت بازنشانی: $r_t = \sigma(U_r x_t + W_r h_{t-1} + b_r)$

۲) لایه گیت بروزرسانی: $z_t = \sigma(U_z x_t + W_z h_{t-1} + b_z)$

وضعیت پنهان منتخب به شکل زیر محاسبه می‌شود:

$$\tilde{h}_t = tanh(U_h x_t + W_h(r_t \circ h_{t-1}) + b_h)$$



شکل ۱۱:واحد GRU برگرفته شده از بلاگ Chris Colah

[۱] Reset Gate

[۲] Update Gate

هنگامی که تمامی عناصر گیت بازنشانی به صفر نزدیک میشود، اطلاعات از گام‌های زمانی (ذخیره شده در h_{t-1}) دور ریخته شده، و وضعیت پنهان منتخب بنابراین فقط بر مبنای ورودی کنونی x_t است. وضعیت پنهان جدید در نهایت به روشی مشابه سلول وضعیت LSTM بدست می آید، با تعامل خطی بین وضعیت پنهان پیشین و وضعیت منتخب:

$$h_t = z_t \circ h_{t-1} + (1 - z_t) \circ \tilde{h}_t)$$

تنها تفاوت اینست که، گیت بروزرسانی z_t در اینجا به عنوان گیت فراموشی عمل میکند و میزان اطلاعاتی که از وضعیت پنهان پیشین باید فراموش شود را تعیین می‌کند، و گیت ورودی با گیت فراموشی ادغام میشوند.

تفاوت RNN و LSTM و GRU

واحد RNN ساده تمام وضعیت پنهانش در هر گام زمانی را به نمایش میگذارد(بیان میکند)، بنابراین با گذشت زمان، تاثیر ورودی های قبلی به سرعت با ورودی های جدیدتر جایگزین میشود. در نتیجه RNN قادر نیست که ویژگی های مهم را برای بیش از چند گام به خاطر بسپارد. قبلا نشان دادیم که RNN مشابه LSTM است وقتی که برای تمامی t ها، $f_{t=1}$، $i_{t=1}$ و $o_{t=1}$ هستند (همه چیز را درباره گذشته فراموش کرده و و همه چیز درباره حال را یاد میگیریم).

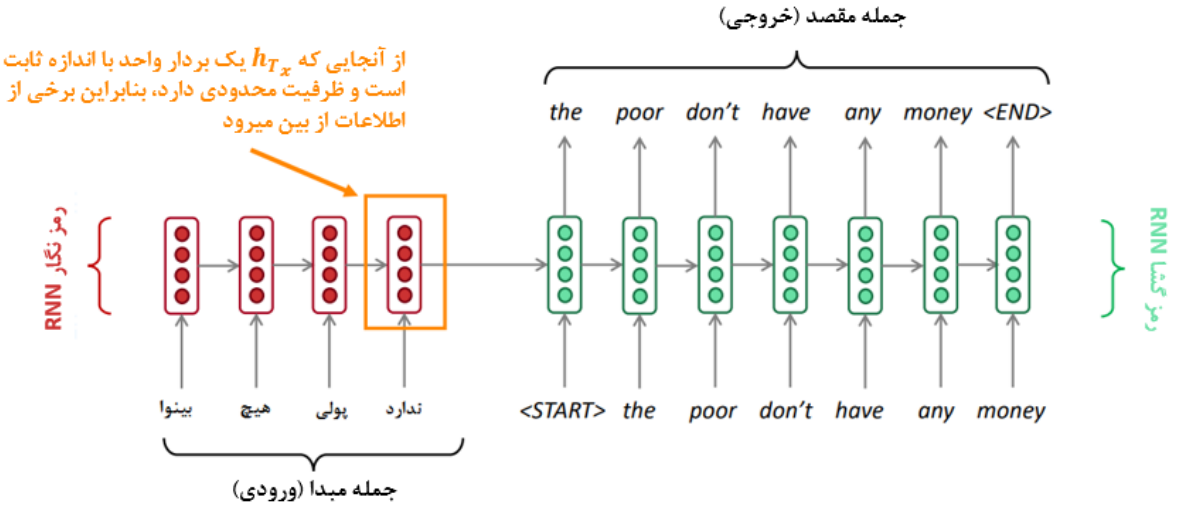
از سوی دیگر، به لطف استفاده از حافظه آشکار (سلول) و سازرکار گیت، واحد LSTM قادر به کنترل میزان اطلاعاتی که از گذشته بایستی به خاطر سپرده شوند (گیت فراموشی f_t)، میزان اطلاعاتی که از ورودی جاری بایستی در حافظه نوشته شود (گیت ورودی i_t)، و چه میزان اطلاعات بایستی به گام زمانی بعدی و لایه های بالاتر داده شود (گیت خروجی o_t)، است.

GRU همچنین از سازوکار گیت استفاده میکند، اما هیچ حافظه آشکاری (وضعیت سلول) ندارد. در نتیجه، سازوکار گیت GRU ساده تر است، بدون گیت خروجی: تعامل خطی بین اطلاعات پیشین و جدید مستقیما به وضعیت پنهان جدید، بدون فیلترینگ تزریق میشود. و در نهایت، در GRU بالانس بین اطلاعات پیشین و جدید فقط توسط گیت بروزرسانی z_t کنترل میشود، اما LSTM دو گیت مستقل از هم، فراموشی و ورودی دارد.

هرچند هر دو واحد LSTM و GRU نسبت به واحد RNN بهتر هستند [۱۶]، اما هیچ هیچ مدرکی وجود ندارد که کدام بهتر است [۱۶ ۱۷].از آنجا که GRU ساده تر است، پیاده سازی ساده تر، بازدهی بیشتر، و پارامتر های کمتری دارد در نتیجه به داده های آموزش کمتری برای آموزش نیاز دارد.

سازوکار توجه[1]

سازوکار توجه [۱۸] در معماری های رمزنگار-رمزگشا[2] برای ترجمه ماشینی عصبی[3] (NMT) توسعه داده شد [۷ ۱۵]، و به صورت مکرر در کارهای مشابه مثل image captioning (ترجمه یک عکس به جمله) [۷]، و خلاصه سازی [۱۹] استفاده شد. از یک دیدگاه سطح بالا، در حالت عادی در یک معماری رمزنگار-رمزگشا، رمزنگار بایستی خروجی را در یک بردار با طول ثابت جاسازی کنید.



شکل ۱۲:مشکل گلوگاه معماری دنباله به دنباله

اگر به رمزگشا این امکان را بدهیم که از میان چندین برداری که توسط رمزنگار تولید شده، یکی را برای خود برگزیند یا بر روی تعدادی از آنها تمرکز کند، مشکل از دست دادن اطلاعات منتفی میشود و میتوان اطلاعات بیشتری را ذخیره کرد. امروزه، سازوکار توجه در تمامی مدل های یادگیری عمیق حضور دارد، و کارکردشان فقط به رمزنگار-رمزگشا محدود نمیشود.

در ادامه، ابتدا سازوکار توجه در متن رمزنگار-رمزگشا برای ترجمه ماشینی عصبی ارایه خواهیم کرد [۷]، و سپس خودتوجه[4] را معرفی خواهیم کرد.

توجه در رمزنگار-رمزگشا

از یک دید سطح بالا، همانطور که در شکل ۱۳ نمایش داده شده، رمزنگار ورودی را در یک بردار جاسازی میکند، و رمزگشا خروجی هایی از این بردار تولید میکند.

---

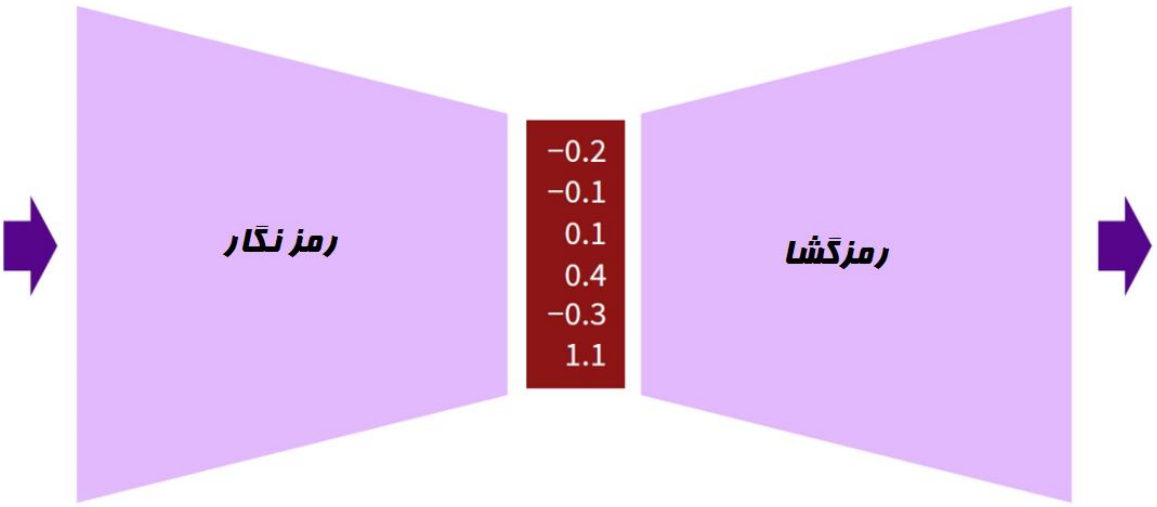
[1] Attention Mechanism

[2] Encoder-Decoder

[3] Neural Machine Translation

[4] Self-Attention

در ترجمه ماشینی عصبی (NMT)، ورودی و خروجی دنباله ای از کلمات به شکل متناظر $x = \{x_t, \dots, xT_t\}$ و $y = \{y_t, \dots, yT_t\}$ هستند. x و y معمولا به جملات مبدا و هدف اشاره دارند. زمانی که هر دوی ورودی و خروجی دنباله هستند، معماری رمزنگار-رمزگشا گاهی اوقات دنباله به دنباله[۱] نامیده میشود [۷].



شکل ۱۳: نمای کلی از معماری رمزنگار-رمزگشا

به لطف مشتق‌پذیر بودن معماری های رمزنگار-رمزگشا، پارامترهای θ آنها میتوانند توسط برآورد درست نمایی بیشینه[۲] بر روی یک مجموعه موازی متن، بهینه شوند. این روش آموزش همچنین یکپارچه[۳] نامیده میشود.

$$armax_\theta \left\{ \sum_{(x,y)\epsilon \text{مجموعه متن}} \log p(y|x;\theta) \right\}$$

در اینجا، تابعی که ما میخواهیم آن را بیشینه کنیم، لگاریتم احتمال ترجمه صحیح است.

رمزنگار[۴]

جمله مبداء را میتوان توسط هر مدلی (به عنوان مثال CNN، تماما متصل) در یک بردار تعبیه کرد. معمولا برای ترجمه ماشینی، رمزنگار یک شبکه RNN عمیق است. [۱۸] از یک شبکه عمیق RNN دوطرفه[۵] استفاده کرد. این گونه معماری ها از دو RNN با پارامتر های مجزا (به جز ماتریس تعبیه کلمه) استفاده میکنند، که RNN پیشخور اطلاعات را از سمت چپ به راست پردازش میکند و RNN معکوس اطلاعات را

---

۱ Sequence-To-Sequence

۲ *Maximum Likelihood Estimation*

۳ End-To-End

۴ Encoder

۵ bidirectional

از سمت راست به چپ پردازش میکند. دو جمله تعبیه شده در هر گام زمانی t برای بدست آمدن بازنمایی درونی RNN دوطرفه به یکدیگر الحاق میشوند:

$$h_t = [\vec{h}_t; \overleftarrow{h}_t]$$

RNN دو طرفه هنگام رمزنگاری کلمات مبداء نه فقط کمات پیشین بلکه تمامی متن را به حساب می‌آورد. در نتیجه، h_t برمبنای x_t و یک پنجره کوچک از کلمات اطراف آن است، در حالی که در یک RNN غیر دو طرفه، h_t بر مبنای x_t و گروه کوچکی از کلمات پیشین است. تمرکز بر روی پنجره کوچکی از کلمات در اطراف x_t شاید سودمند باشد، اما [۲۰] بهترین نتایج را توسط یک RNN رمزنگار عمیق غیر دوطرفه بدست آورد. در ادامه، وضعیت پنهان رمزنگار با علامت $\bar{h}_t$ نوشته خواهد شد. در مقالات معمولا Annotation نامیده میشوند که به صورت قرارداد از اینجا به بعد (با توجه به اینکه دقیقا به وضعیت پنهان رمزنگار اشاره دارد) آن را وضعیت پنهان رمزنگار مینامیم.

رمزگشا[۱]

اگرچه مدل های مختلف از رمزنگارهای مختلفی استفاده می‌کنند، در ترجمه ماشینی عصبی رمزگشا معمولا یک RNN غیر دوطرفه است زیرا این مدل به طور طبیعی برای تولید دنباله ها پذیرفته شده است، و معمولا عمیق است. رمزگشا در هر لحظه یک کلمه از جمله هدف را تولید می‌کند.

ایده کلیدی: اگر رمز گشا را مجبور کنیم که فقط از آخرین وضعیت پنهان رمزنگار h_{T_x} که توسط رمزنگار تولید میشود برای تولید خروجی استفاده کند، رمزنگار مجبور است تا جایی که امکان پذیر است اطلاعات در h_{T_x} ذخیره کند، از آنجایی که h_{T_x} یک بردار واحد با اندازه ثابت است و ظرفیت محدودی دارد، بنابراین برخی از اطلاعات از بین میرود (شکل ۱۲). از سوی دیگر، سازوکار توجه این اجازه را به رمزگشا میدهد که تمامی دنباله وضعیت های پنهان رمزنگار $(h_1, \dots, h_{T_x})$ در هر گام زمانی را در نظر بگیرد. در نتیجه، رمزنگار قادر است تا اطلاعات بیشتری با توزیع آن در بین همه وضعیت های پنهان رمزنگار ذخیره کند با این هدف که بعدا رمزگشا قادر خواهد بود که از بین این بردار ها توجه بیشتری به برخی از آنها اختصاص دهد.

به عبارت دقیقتر، جمله هدف $y = (y_1, \dots, y_{T_x})$ بر مبنای توزیع زیر در هر گام یک کلمه y_t را تولید میکند:

$$P[y_t|\{y_1, \dots, y_{t-1}\}, c_t] = softmax(W_s\tilde{h}_t)$$

که $\tilde{h}_t$، وضعیت پنهان توجه[۲]، به صورت زیر محاسبه میشود:

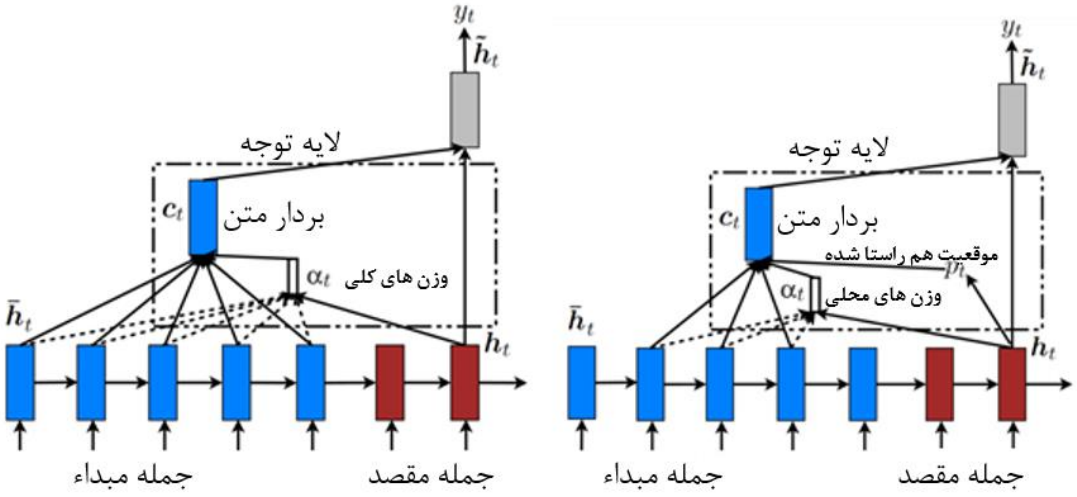
$$\tilde{h}_t = \tanh(w_c[c_t; h_t])$$

---

[۱] Decoder

[۲] Attentional Hidden State

h_t وضعیت پنهان رمزگشا است و اطلاعاتی درباره کلمات تولید شده قبلی $\{y_1, \dots, y_{t-1}\}$ فراهم میکند، c_t بردار متن[1] مبدا و [;] الحاق[2] است. W_c و W_s ماتریس هایی با پارامترهای قابل آموزش هستند. در این رابطه سوءگیریها[3] برای سادگی حذف شده اند. همانگونه که در شکل ۱۴ نشان داده شده، بردار متن c_t را میتوان به دو روش محاسبه کرد: محلی[4] و کلی[5]. هر کدام از این دو متد در بخش های بعد توضیح داده خواهد شد.



شکل ۱۴:توجه کلی در سمت چپ و توجه محلی در سمت راست

اطلاعات‌نگاشت سازوکار توجه

قبل از پرداختن به این جزئیات و ذکر علائم ریاضی در اطلاعات‌نگاشت[6] صفحات بعد یک تصویر کلی از سازوکار توجه آورده شده است.

---

[1] Context vector

[2] Concatenation

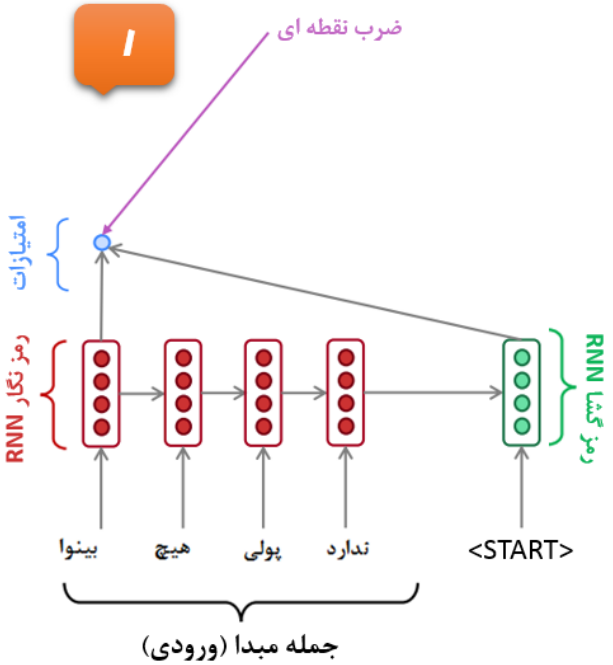
[3] Biases

[4] Locally

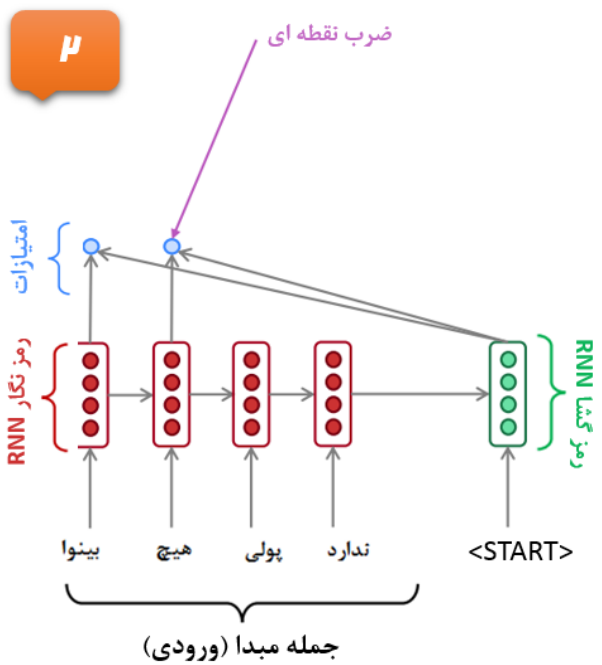
[5] Globally

[6] Infographic

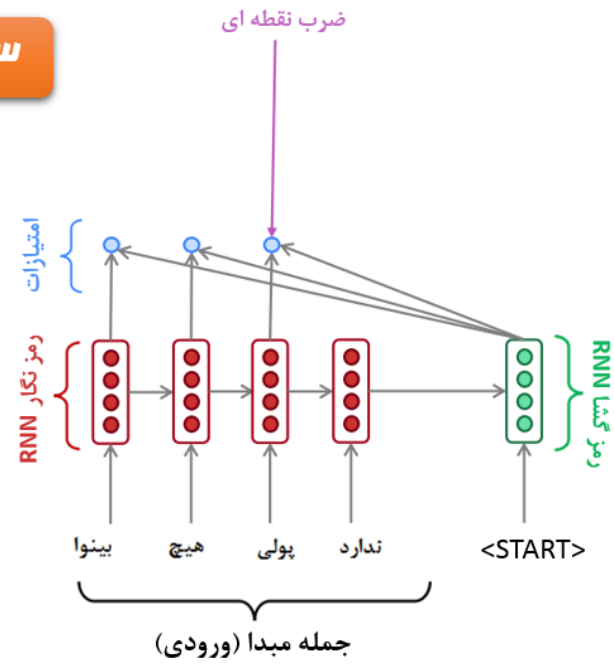
۱



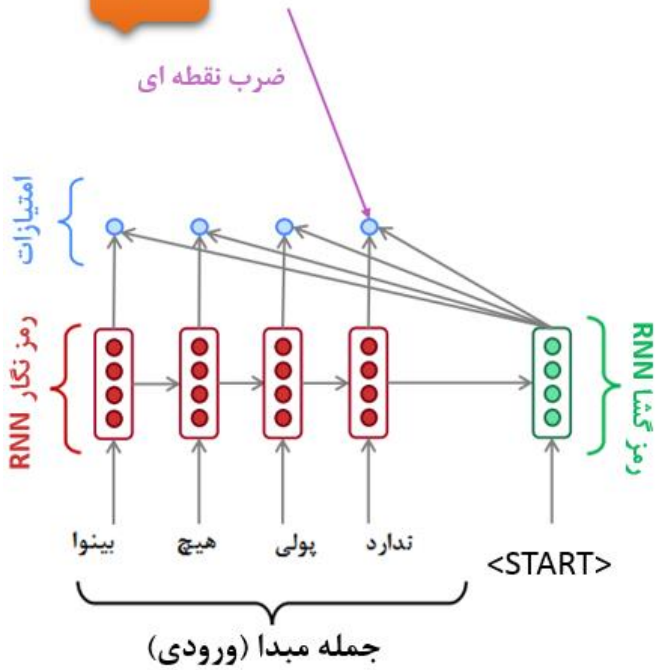
۲



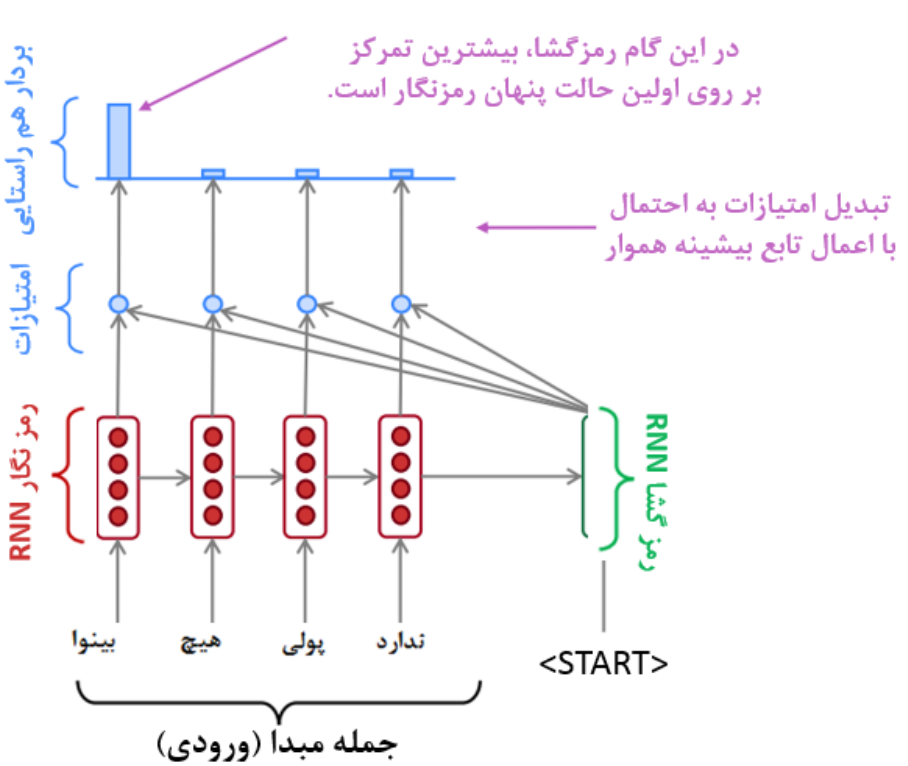
۳

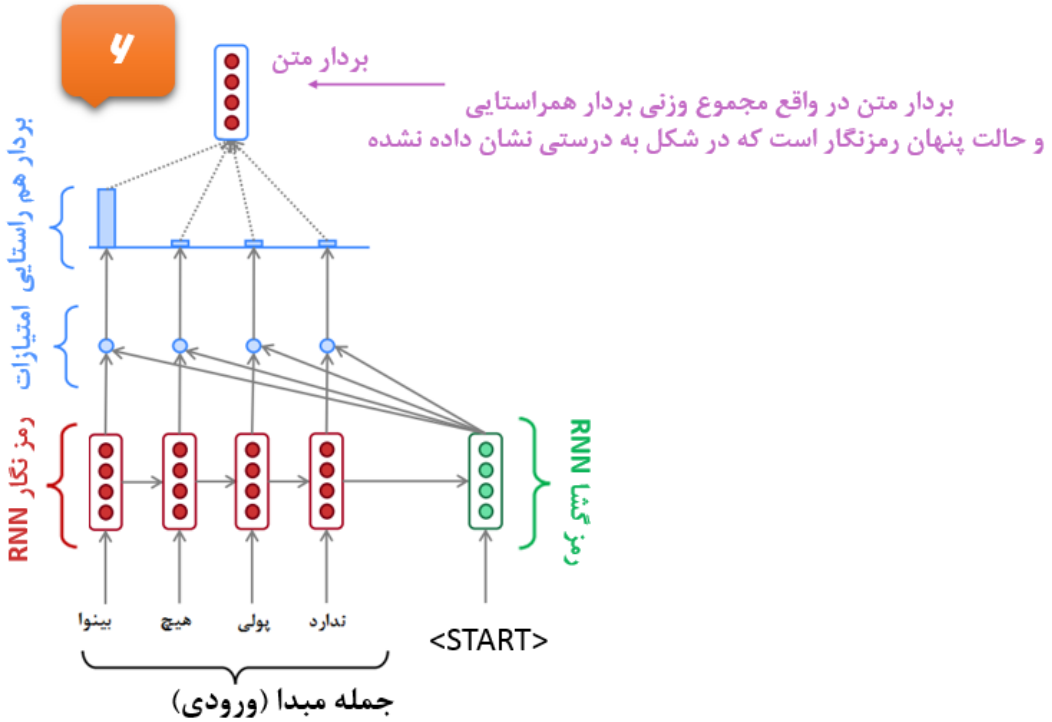


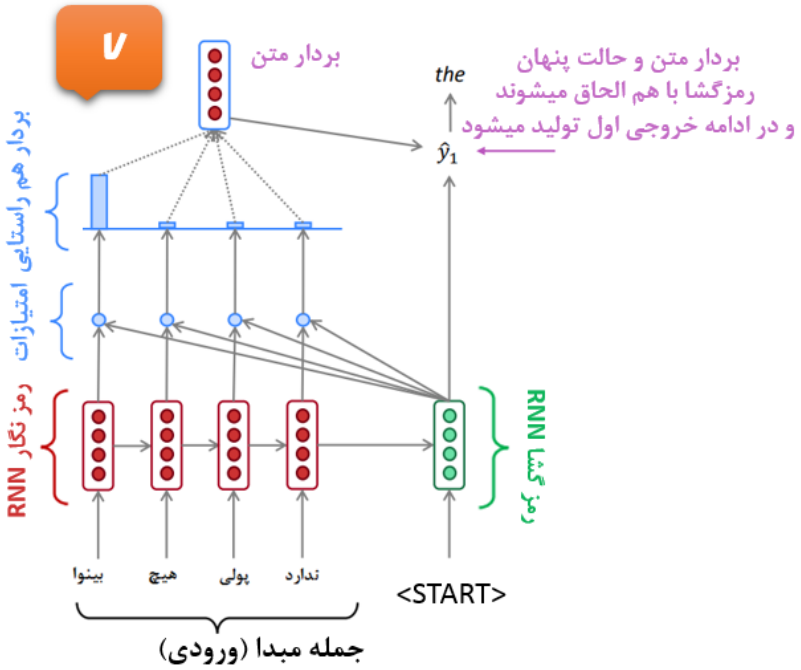
۴

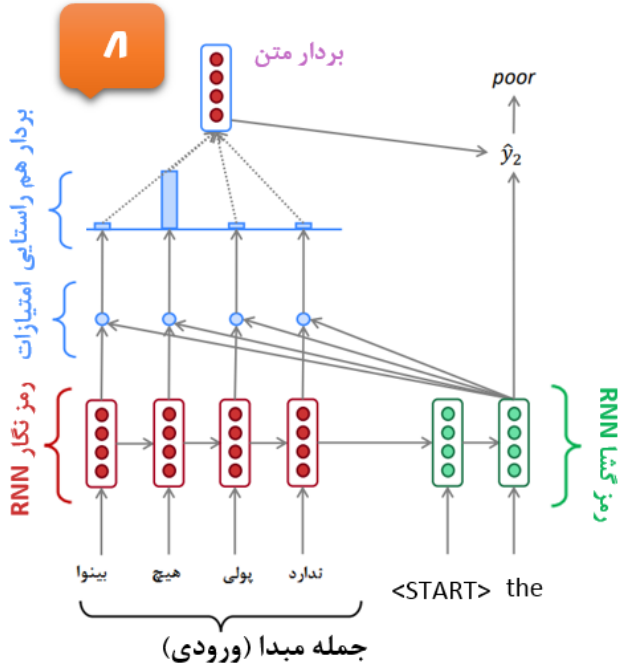


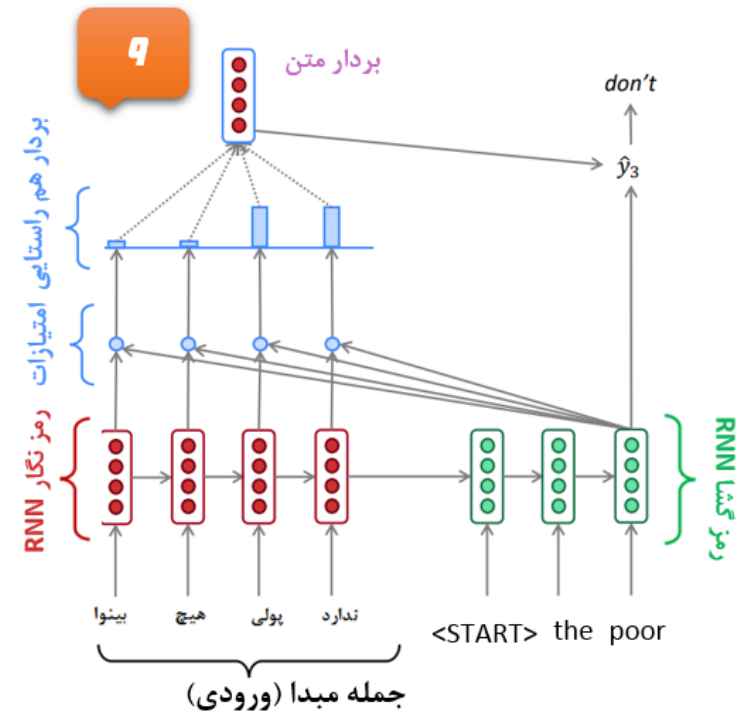
۵

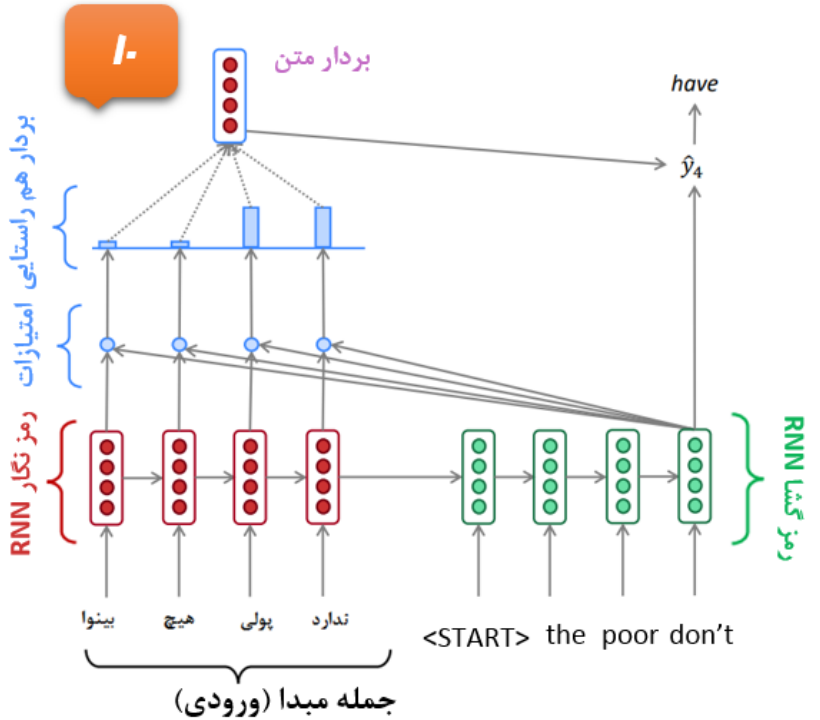


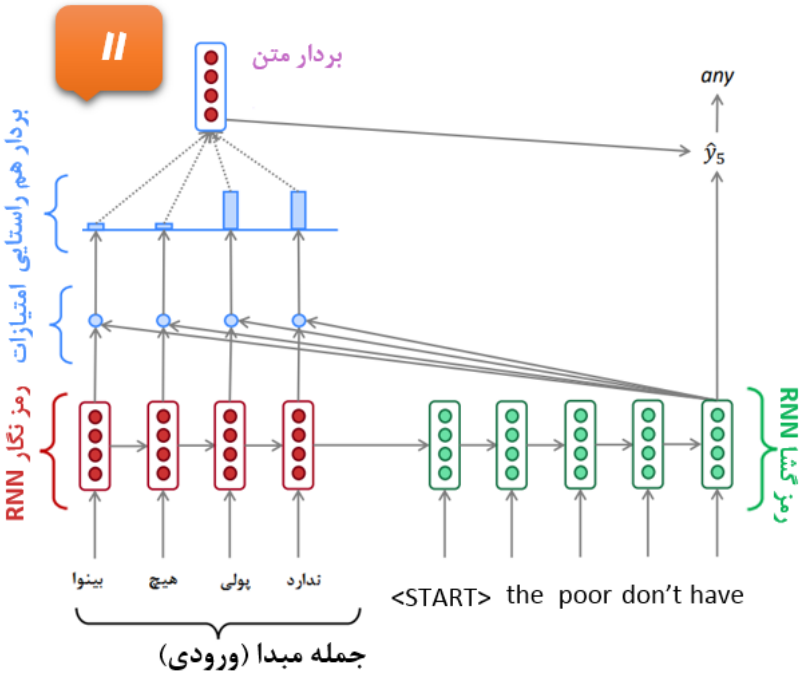

۶
بردار متن
بردار متن در واقع مجموع وزنی بردار همراستایی
و حالت پنهان رمزنگار است که در شکل به درستی نشان داده نشده
بردار هم راستایی امتیازات
رمز نگار RNN
رمز گشا RNN
بینوا
هیچ
پولی
ندارد
<START>
جمله مبدا (ورودی)

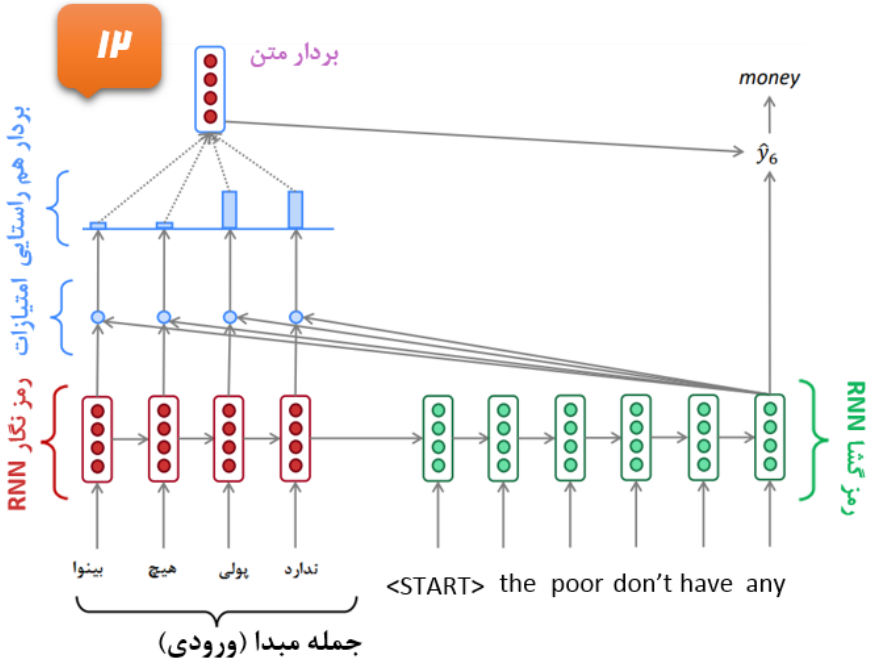

۷
بردار متن
the
بردار متن و حالت پنهان
رمزگشا با هم الحاق میشوند
و در ادامه خروجی اول تولید میشود
$\hat{y}_1$
بردار هم راستایی امتیازات
رمز نگار RNN
رمز گشا RNN
بینوا
هیچ
پولی
ندارد
<START>
جمله مبدا (ورودی)


۸
بردار متن
poor
$\hat{y}_2$
بردار هم راستایی امتیازات
رمز نگار RNN
رمز گشا RNN
بینوا
هیچ
پولی
ندارد
<START>
the
جمله مبدا (ورودی)


۹
بردار متن
don't
$\hat{y}_3$
بردار هم راستایی امتیازات
رمز نگار RNN
رمز گشا RNN
بینوا
هیچ
پولی
ندارد
<START> the poor
جمله مبدا (ورودی)


۱۰
بردار متن
have
$\hat{y}_4$
بردار هم راستایی امتیازات
رمز نگار RNN
رمز گشا RNN
بینوا
هیچ
پولی
ندارد
<START> the poor don't
جمله مبدا (ورودی)


۱۱
بردار متن
any
$\hat{y}_5$
بردار هم راستایی امتیازات
رمز نگار RNN
رمز گشا RNN
بینوا
هیچ
پولی
ندارد
<START> the poor don't have
جمله مبدا (ورودی)


۱۲
بردار متن
money
$\hat{y}_6$
بردار هم راستایی امتیازات
رمز نگار RNN
رمز گشا RNN
بینوا
هیچ
پولی
ندارد
<START> the poor don't have any
جمله مبدا (ورودی)

توجه کلی[۵۵]

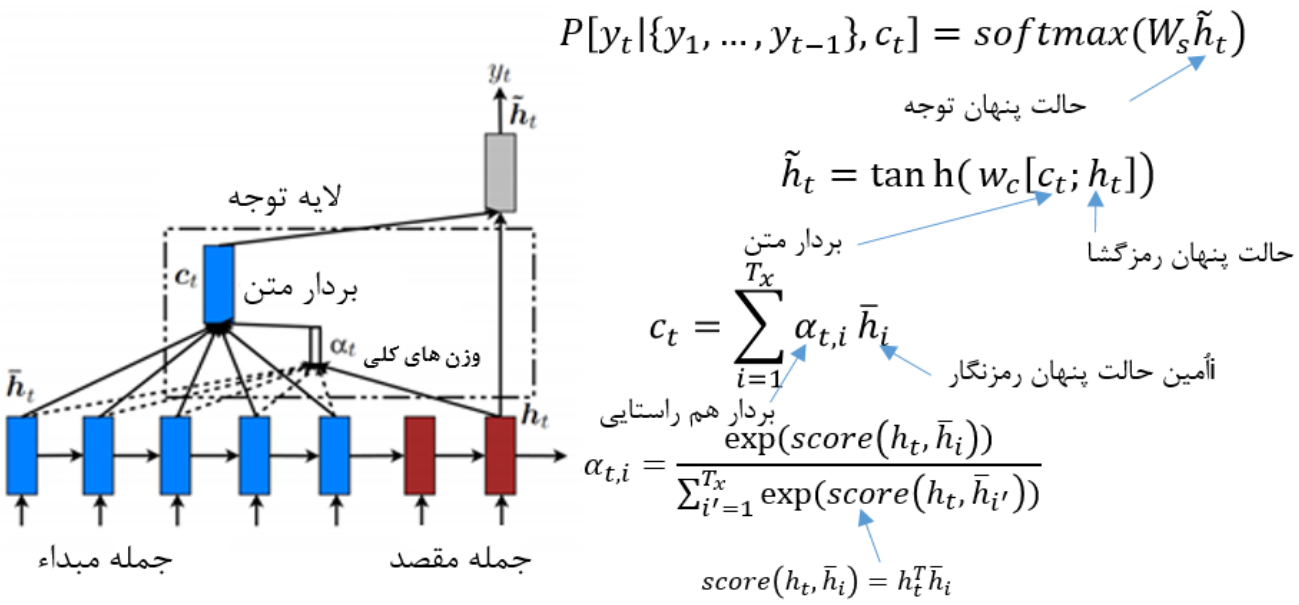
در اینجا، بردار متن c_t توسط مجموع وزنی لیست تمامی وضعیت های پنهان رمزنگار $\bar{h}_t$ جمله مبدا محاسبه میشود. T_x وضعیت پنهان رمزنگار وجود دارد که هر کدام یک بردار با اندازه تعداد نورون ها در لایه رمزگشا است. اندازه c_t برابر با وضعیت پنهان رمزنگار است. اندازه بردار هم راستایی[۵۶] α_t برابر با اندازه T_x جمله مبدا است، بنابر این میتواند متغیر باشد.

$$c_t = \sum_{i=1}^{T_x} \alpha_{t,i}\, \bar{h}_i$$

بردار هم راستایی با اعمال تابع بیشینه هموار بر روی خروجی عملیات هم راستایی ($score()$) بین وضعیت پنهان کنونی هدف h_t و تمامی وضعیت های پنهان مبداء $\bar{h}_t$ محاسبه میشود:

$$\alpha_{t,i} = \frac{\exp(score(h_t, \bar{h}_i))}{\sum_{i'=1}^{T_x} \exp(score(h_t, \bar{h}_{i'}))}$$

به عبارت دیگر، α_t توزیع احتمال بر روی همه وضعیت های پنهان مبداء است (ضرایب بین صفر و یک هستند که مجموع آنها یک است)، و بیان کننده این موضوع است که کدام کلمات در جمله مبداء به احتمال بیشتر میتوانند در پیش بینی کلمه بعدی کمک کنند.$score()$



شکل ۱۵:خلاصه سازوکار توجه کلی ارائه شده در [۲۰]

در تئوری میتواند هر تابع مقایسه[۵۷] باشد. [۲۰] سه روش ضرب نقطه ای[۵۸] ($score(h_t, \bar{h}_i) = h_t^T \bar{h}_i$)، یک فرمول جامع تر با ماتریسی

---

[۵۵] Global Attention

[۵۶] Alignment Vector

[۵۷] Comparison function

[۵۸] Dot product

از پارامتر ها $score(h_t, \bar{h}_i) = h_t^T W_\alpha \bar{h}_i$، و یک لایه تماما متصل را امتحان کردند و دریافتند که مورد اول برای توجه کلی بهتر عمل میکند و مورد دوم برای توجه محلی. خلاصه توجه کلی در شکل ۱۵ نمایش داده شده است.

توجه محلی

در نظر گرفتن تمام کلمات در جمله مبداء برای تولید هر کلمه هدف پرهزینه است، و از طرفی ممکن است ضروری نباشد. برای کم کردن این مشکل، [۲۰] پیشنهاد داد که فقط بر روی پنجره کوچکی از وضعیت های پنهان رمزنگار با اندازه $2D + 1$ تمرکز کنیم :

$$c_t = \sum_{i=p_t-D}^{p_t+D} \alpha_{t,i} \bar{h}_i$$

D توسط کاربر تعیین میشود، و موقعیت p_t جایی که مرکز پنجره باید قرار بگیرد، میتواند به t (هم راستایی یکنوا[۵۹]) یا با یک سازوکار تمایز پذیر (هم راستایی پیشگو[۶۰]) بر مبنای اطلاعات کلمات هدف تولید شده قبلی $\{y_1, \dots, y_{t-1}\}$ذخیره شده در h_t تنظیم گردد:

$$p_t = T_x . \sigma(\upsilon_p^\top \tanh(W_p h_t))$$

که T_x طول جمله مبداء، σ تابع سیگموئید، و υ_p و W_p پارامترهای قابل آموزش هستند. وزنهای هم راستایی همانند توجه کلی محاسبه میشوند، با این تفاوت که یک عبارت توزیع نرمال با مرکزیت p_t و انحراف معیار $\frac{D}{2}$ به آن اضافه میشود:

$$\alpha_{t,i} = \frac{\exp(score(h_t, \bar{h}_i))}{\sum_{i'=p_t-D}^{p_t+D} \exp(score(h_t, \bar{h}_{i'}))} \exp(-\frac{(i-p_t)^2}{2(\frac{D}{2})^2})$$

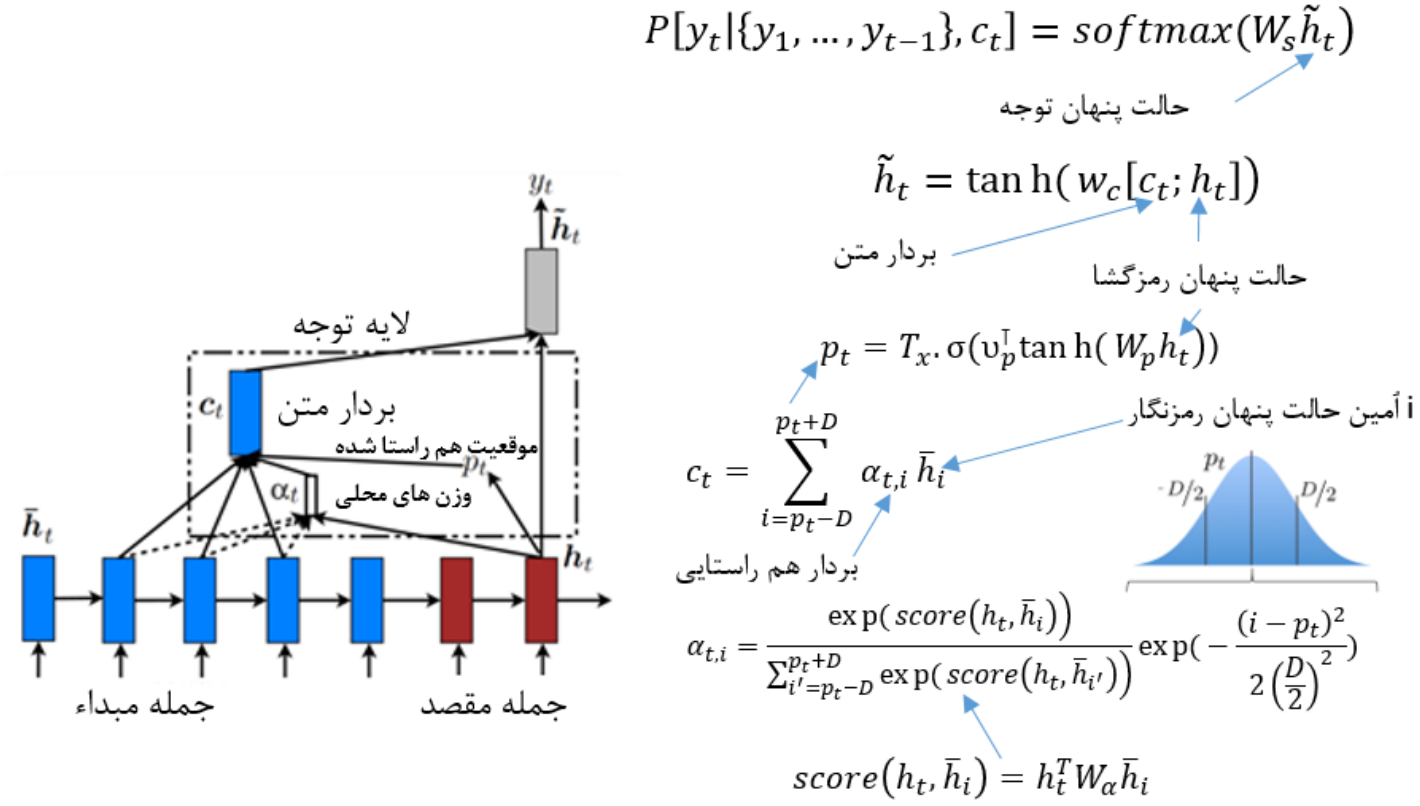
توجه داشته باشید که $p_t \in \mathbb{R} \cap [0, T_x]$ و $i \in \mathbb{N} \cap [p_t - D, p_t + D]$ هستند. اضافه شدن عبارت گوسی باعث میشود که ورزن های هم راستایی با دور شدن i از مرکز پنجره p_t رو به زوال روند، این امر باعث میشود که اهمیت بیشتری به وضعیت های پنهان رمزنگار نزدیک p_t داده شود. همچنین برخلاف توجه کلی، اندازه α_t ثابت و برابر با $2D + 1$ است (از آنجایی که فقط وضعیت های پنهان رمزنگار موجود در پنجره به حساب آورده میشوند.) توجه محلی را در حقیقت میتوان همانند توجه کلی دید که وزن های هم راستایی در یک توزیع نرمال بریده شده[۶۱] ضرب شده است. خلاصه توجه محلی در شکل ۱۶ نمایش داده شده است.

---

[۵۹] Monotonic Alignment

[۶۰] Predictive Alignment

[۶۱] Truncated Normal Distribution



شکل ۱۶: خلاصه attention موضعی با مکانیسم predictive alignment ارائه شده در [۲۰]

خودتوجه[۶۲]

همانگونه که قبلا توضیح داده شد، یک رمزنگار RNN یک دنباله $\{x_1,\dots,x_T\}$ به طول T را از ورودی دریافت میکند. به طور معمول، RNN دنباله ورودی را به دنباله وضعیت های پنهان رمزنگار$\{h_1,\dots,h_T\}$ نگاشت میدهد. هدف در اینجا دقیقا همانند سازوکار توجه در رمزنگار-رمزگشا است: به جای اینکه آخرین وضعیت پنهان رمزنگار h_T را به عنوان خلاصه فراگیر کل مجموعه در نظر بگیریم، که معمولا منجر به از دست دادن اطلاعات میشود، یک نمایش درونی با به حساب آوردن همه وضعیت های پنهان رمزنگار در تمامی گام های زمانی محاسبه میشود. برای رسیدن به این هدف در سال ۲۰۱۷/۲۰۱۶ خودتوجه در مقالات ظهور پیدا کرد (به عنوان مثال[۲۱] [۲۲]).

در این روش وضعیت پنهان رمزنگار h_T ابتدا از یک لایه متراکم[۶۳] عبور میکند. یک ضریب هم راستایی α_t با مقایسه خروجی u_t از لایه متراکم با بردار متن قابل آموزش u (به صورت تصادفی مقدار دهی میشود) بدست می آید و توسط یک تابع بیشینه هموار نرمال سازی میشود. بردار توجه s در نهایت از یک جمع وزنی وضعیت های پنهان رمزنگار بدست می آید.

$$u_t = \tanh(Wh_t)$$

$$\alpha_t = \frac{\exp(score(u_t,u))}{\sum_{t'=1}^{T}\exp(score(u_{t'},u))}$$

---

[۶۲] Self-Attention

[۶۳] Dense Layer

$$s = \sum_{t=1}^{T} \alpha_t h_t$$

score در تئوری میتواند هر تابع هم راستایی باشد. یک متد ساده استفاده از $score(u_t, u) = u_t{}^T u$ است.

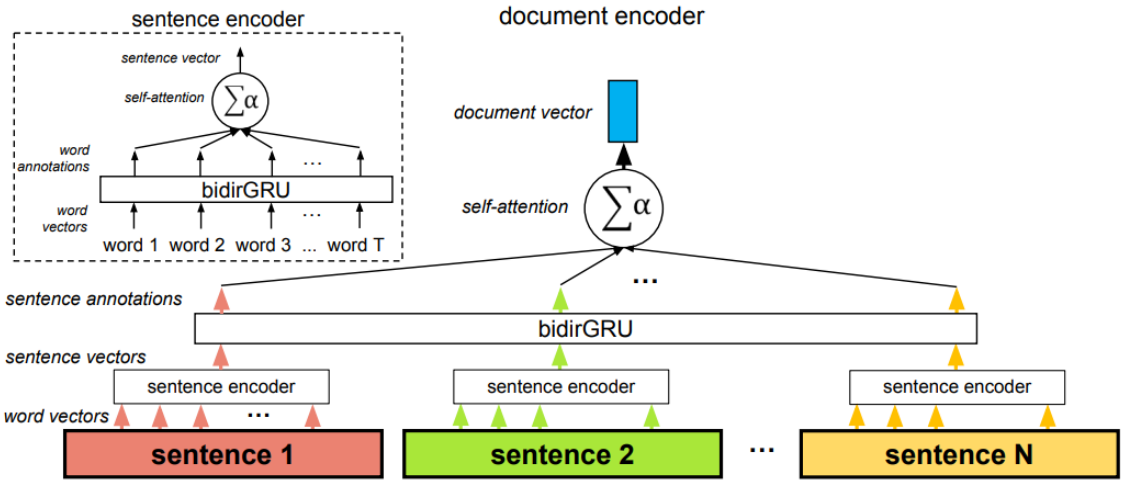
بردار متن را میتوان به عنوان یک بازنمایی از مساعدترین کلمه تفسیر کرد. هنگامی که با یک نمونه جدید روبرو میشود، مدل از این اطلاعات برای تصمیم اینکه توجه را به کدام کلمه اختصاص دهد استفاده میکند. در طول آموزش، با استفاده از الگوریتم انتشار معکوس مدل بردار متن را بروز رسانی میکند.

تفاوت خودتوجه و توجه بکار رفته در مدل دنباله به دنباله

بردار متن در تعریف خودتوجه بالا هیچ ربطی به بردار متنی که در توجه دنباله به دنباله استفاده شد، ندارد. در مدل دنباله به دنباله ، بردار متن c_t برابر با مجموع وزنی $c_t = \sum_{i=1}^{T_x} \alpha_{t,i} \bar{h}_i$ است، و برای محاسبه وضعیت پنهان توجه $\tilde{h}_t = \tanh(w_c[c_t; h_t])$ استفاده شد. در خودتوجه هرچند، بردار متن به جای وضعیت پنهان رمزگشا هنگام اجرای هم راستایی با score()، استفاده میشود. بنابراین در خودتوجه، بردار هم راستایی α بیانگر شباهت هر کدام از کلمات ورودی با مساعدترین کلمه است (در مجموع)، اما در توجه دنباله به دنباله، α بیانگر رابطه هر کدام از کلمات مبدا در تولید عنصر بعدی در جمله مقصد است.

توجه سلسله مراتبی[۶۴]

یک مثال ساده و خوب از اینکه چگونه خودتوجه میتواند در عمل مفید واقع شود در معماری شکل ۱۷ آورده شده است. در این معماری، سازوکار خودتوجه دو بار در بازی شرکت میکند: در سطح کلمه، و در سطح جمله. این متد به دو دلیل عمده با عقل جور در می آید: اول اینکه با ساختار سلسله مراتبی متن همخوانی دارد (کلمه ←جملات←متن). دوم، در فاز کدگذاری متن، این اجازه را به مدل میدهد که ابتدا در یابد که کدام کلمات در هر جمله مهم هستند، و سپس، کدام جملات مهم هستند.



شکل ۱۷: خلاصه attention موضعی با مکانیسم predictive alignment ارائه شده در [۲۰]

[۶۴] Hierarchical Attention

ویژگی های توجه

- ✓ توجه به صورت چشمگیری کارایی ترجمه ماشینی عصبی (NMT) را افزایش میدهد. (با اجازه دادن به رمزگشا که بر روی بخش های خاصی از ورودی تمرکز کند.)
- ✓ توجه مشکل گلوگاه را حل میکند. (با استفاده از توجه، رمزگشا دیگر فقط آخرین از وضعیت پنهان رمزنگار استفاده نمی‌کند و مستقیما به حالات پنهان رمزنگار دسترسی دارد.)
- ✓ توجه در حل مشکل ناپدید شدن گرادیان کمک بسزایی می‌کند. (همانند معماری Resnet، در اینجا توجه مسیر های میانبری را برای مدل فراهم میکند.)
- ✓ توجه کمک بسزایی به تفسیرپذیری مدل میکند
  - با تفسیر توزیع توجه میتوان مشاهده کرد رمزگشا بر روی چه کلماتی تمرکز میکند.
  - بدون هیچ زحمتی بردار هم‌راستایی محاسبه میشود و ما به آن دسترسی داریم. (شبکه خودش هم‌راستایی را یاد میگیرد.

Transformer

[۲۳] "توجه همه چیزیست که شما به آن نیاز دارید" بدون شک یکی از تاثیر گذارترین و جالب ترین مقالات سال ۲۰۱۷ بود. در این مقاله ایده های جالبی مطرح شد و مدل ارایه شده از هیچ RNN و CNN استفاده نکرد. به قول Eugenio Culurciello " بیاید RNN ها را دور بندازیم، توجه تنها چیزیست که شما به آن نیاز دارید". در این معماری از هیچ لایه بازگشتی RNN و CNN استفاده نشد و تماما بر مبنای سازوکار خودتوجه ساخته شد. یکی از معایب اصلی RNN پردازش کلمات به صورت ترتیبی است، این موضوع یکی از موانع موجود بر سر راه موازی سازی محاسبات است. Transformer برای حل این مشکل، لایه های بازگشتی را کاملا با سازوکار توجه جایگزین میکند از طرفی مشکل وابستگی بلند-برد در RNN ها و اشکال مختلف آن در دنباله های با طول زیاد هنوز یک مسئله مهم می‌باشد. Transformer با این ایده که سازوکار توجه مشکل وابستگی های بلند-برد را حل می‌کند، چه لزومی به استفاده از RNN است، به کلی آن را از مدل خود حذف می‌کند.

جز اصلی Transformer واحدی به نام سازوکار خودتوجه چندگانه[۶۵] است. Transformer بازنمایی رمزنگاری شده ورودی را به عنوان یک جفت کلید[۶۶] و ارزش[۶۷] n بُعدی میبیند (K, V)، در زمینه NMT هر دوی کلید و ارزش حالات پنهان رمزنگار هستند. در رمزگشا، خروجی پیشین در یک ماتریسی به نام query (Q) ذخیره میشود، و خروجی بعدی با نگاشت این query و مجموعه از کلید و

---

[۶۵] Multi-Head Self-Attention

[۶۶] Key

[۶۷] Value

ارزش ها تولید میشود. خروجی مجموع وزنی ارزش ها است، جایی که وزن تخضیص داده شده به هر کدام از ارزش ها با ضرب نقطه ای query و کلید ها بدست می آید.

$$Q, K, V = softmax\left(\frac{QK^{\top}}{\sqrt{n}}\right)V$$

منابع:

[۱] Hubel, David H., and Torsten N. Wiesel (۱۹۶۲). Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. The Journal of physiology ۱۶۰,۱:۱۰۶-۱۵۴.

[۲]LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (۱۹۹۸). Gradient-based learning applied to document recognition. Proceedings of the IEEE, ۸٦(۱۱), ۲۲۷۸-۲۳۲٤.

[۳]Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." Advances in neural information processing systems. ۲۰۱۲.

[٤]Springenberg, Jost Tobias, et al. "Striving for simplicity: The all convolutional net." arXiv preprint arXiv:۱٤۱۲,٦۸۰٦ (۲۰۱٤).

[٥] Kim, Y. (۲۰۱٤). Convolutional Neural Networks for Sentence Classification. Proceedings of the ۲۰۱٤ Conference on Empirical Methods in Natural Language Processing (EMNLP ۲۰۱٤), ۱۷٤٦–۱۷٥۱.

[٦] Zhang, Ye, and Byron Wallace. "A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification." arXiv preprint arXiv:۱٥۱۰,۰۳۸۲۰ (۲۰۱٥)

[۷] Sutskever, Ilya, Oriol Vinyals, and Quoc V. Le. "Sequence to sequence learning with neural networks." Advances in neural information processing systems. ۲۰۱٤.

[۸] Johnson, R., Zhang, T. (۲۰۱٥). Effective Use of Word Order for Text Categorization with Convolutional Neural Networks. To Appear: NAACL-۲۰۱٥, (۲۰۱۱)

[۹] Simonyan, K., Vedaldi, A., and Zisserman, A. (۲۰۱۳). Simonyan, Karen, Andrea Vedaldi, and Andrew Zisserman. "Deep inside convolutional networks: Visualising image classification models and saliency maps." arXiv preprint arXiv:۱۳۱۲,٦۰۳٤ (۲۰۱۳). arXiv preprint arXiv:۱۳۱۲,٦۰۳٤

[۱۰] Li, J., Chen, X., Hovy, E., and Jurafsky, D. (۲۰۱٥). Visualizing and understanding neural models in nlp. arXiv preprint arXiv:۱٥۰٦,۰۱۰٦٦

[۱۱] Lipton, Zachary C., John Berkowitz, and Charles Elkan. "A critical review of recurrent neural networks for sequence learning." arXiv preprint arXiv:۱٥۰٦,۰۰۰۱۹ (۲۰۱٥).

[۱۲] Elman, J. L. (۱۹۹۰). Finding structure in time. Cognitive Science, ۱٤:۲, ۱۷۹-۲۱۱.

[۱۳] Goldberg, Y. (۲۰۱٥). A primer on neural network models for natural language processing. Journal of Artificial Intelligence Research, ٥۷, ۳٤٥-٤۲۰.

[۱٤] Hochreiter, S., Schmidhuber, J. (۱۹۹۷). Long short-term memory. Neural computation, ۹(۸), ۱۷۳٥

[۱٥] Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., Bengio, Y. (۲۰۱٤). Learning phrase representations using RNN encoder-decoder for statistical machine translation. arXiv preprint arXiv:۱٤۰٦,۱۰۷۸.

[١٦] Chung, J., Gulcehre, C., Cho, K., & Bengio, Y. (٢٠١٤). Empirical evaluation of gated recurrent neural networks on sequence modeling. arXiv preprint arXiv:١٤١٢,٣٥٥٥.

[١٧] Greff, Klaus, et al. "LSTM: A search space odyssey." IEEE transactions on neural networks and learning systems ٢٨,١٠ (٢٠١٧): ٢٢٢٢-٢٢٣٢.

[١٨]Bahdanau, Dzmitry, Kyunghyun Cho, and Yoshua Bengio. "Neural machine translation by jointly learning to align and translate." arXiv preprint arXiv:١٤٠٩,٠٤٧٣ (٢٠١٤).

[١٩] Rush, Alexander M., Sumit Chopra, and Jason Weston. "A neural attention model for abstractive sentence summarization." arXiv preprint arXiv:١٥٠٩,٠٠٦٨٥ (٢٠١٥).

[٢٠] Luong, Minh-Thang, Hieu Pham, and Christopher D. Manning. "Effective approaches to attention-based neural machine translation." arXiv preprint arXiv:١٥٠٨,٠٤٠٢٥ (٢٠١٥).

[٢١] Lin, Zhouhan, et al. "A structured self-attentive sentence embedding." arXiv preprint arXiv:١٧٠٣,٠٣١٣٠ (٢٠١٧).

[٢٢] Yang, Zichao, et al. "Hierarchical attention networks for document classification." Proceedings of the ٢٠١٦ Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies. ٢٠١٦.

[٢٣] Vaswani, et al. “Attention is All you Need.” arXiv preprint arXiv:١٧٠٦,٠٣٧٦٢. ٢٠١٧.