

## در ک LLM از ابتدا با استفاده از ریاضی دبیرستان

گردآورنده: ایمان جوادی سیسی

درس: یادگیری ماشین

استاد: دکتر مهدی آژینی

مقطع: کارشناسی ارشد

دانشگاه آزاد اسلامی واحد علوم و تحقیقات

آذر ۱۴۰۲

لينك مقاله:

<https://towardsdatascience.com/understanding-langs-from-scratch-using-middle-school-math-e602d27ec876>

در این مقاله، ما درباره چگونگی کار مدل‌های زبانی بزرگ (LLM‌ها) از پایه صحبت می‌کنیم با این فرض که تنها می‌دانید چگونه دو عدد را جمع و ضرب کنید. این مقاله به گونه‌ای طراحی شده است که کاملاً خودبسته باشد. ما با ساختن یک هوش مصنوعی مولد ساده با قلم و کاغذ شروع می‌کنیم و سپس هر چیزی که برای درک عمیق مدل‌های زبانی بزرگ مدرن و معماری ترانسفرمر نیاز داریم را بررسی می‌کنیم. این مقاله تمام زبان پیچیده و اصطلاحات تخصصی در یادگیری ماشین را حذف می‌کند و همه چیز را به سادگی آنچه هستند، یعنی اعداد، ارائه می‌دهد. ما همچنان به نام‌های مفاهیم اشاره خواهیم کرد تا در هنگام مطالعه محتوای پر از اصطلاحات، افکار شما را متصل نگه داریم.

رفن از جمع و ضرب به پیشرفته ترین مدل‌های هوش مصنوعی امروزی، بدون فرض دانش دیگر یا ارجاع به منابع دیگر، به این معنی است که ما مطالب بسیار زیادی را پوشش می‌دهیم. این یک توضیح ساده یا ابتدایی از مدل‌های زبانی بزرگ (LLM) نیست یک فرد مصمم می‌تواند تثوارآ یک LLM مدرن را از تمام اطلاعات ارائه شده در اینجا بازسازی کند. من هر کلمه یا خط غیرضروری را حذف کرده‌ام و به همین دلیل این مقاله واقعاً برای مرور سطحی طراحی نشده است.

## چه مواردی را پوشش خواهیم داد؟

۱. یک شبکه عصبی ساده
۲. این مدل‌ها چگونه آموزش داده می‌شوند؟
۳. چگونه همه این‌ها زبان را تولید می‌کنند؟
۴. چه چیزی باعث می‌شود مدل‌های زبانی بزرگ (LLM‌ها) به این خوبی کار کنند؟
۵. تعبیه‌ها (Embeddings)
۶. توکنایزرهای زیرواژه‌ای (Sub-word tokenizers)
۷. توجه به خود (Self-attention)
۸. سافت‌مکس (Softmax)
۹. اتصالات باقی‌مانده (Residual connections)
۱۰. نرمال‌سازی لایه‌ای (Layer Normalization)
۱۱. دراپ‌اوت (Dropout)
۱۲. توجه چندسری (Multi-head attention)

۱۳. تعبیه‌های موقعیتی (Positional embeddings)

۱۴. معماری GPT

۱۵. معماری transformer

بیایید شروع کنیم:

اولین نکته‌ای که باید توجه کنیم این است که شبکه‌های عصبی تنها می‌توانند اعداد را به عنوان ورودی دریافت کنند و فقط اعداد را خروجی دهنند. بدون هیچ استثنایی. هنر در این است که بفهمیم چگونه ورودی‌های خود را به اعداد تبدیل کنیم و خروجی‌های عددی را به شکلی تفسیر کنیم که به اهدافمان برسد. و در نهایت، ساختن شبکه‌های عصبی که ورودی‌هایی را که شما اراده می‌دهید دریافت کنند و خروجی‌هایی را که می‌خواهید به شما بدهند (با توجه به تفسیری که برای این خروجی‌ها انتخاب کردید). بیایید پررسی کنیم چگونه از جمع و ضرب اعداد به مدل‌هایی مانند Llama 3.1 می‌رسیم.

## یک شبکه عصبی ساده

بیایید یک شبکه عصبی ساده را بررسی کنیم که می‌تواند یک شیء را طبقه‌بندی کند:

- داده‌های موجود برای شی: رنگ غالب (RGB) و حجم (بر حسب میلی‌لیتر)
- طبقه‌بندی به: برگ یا گل

در اینجا داده‌های یک برگ و یک گل آفتابگردان می‌تواند به این شکل باشد:

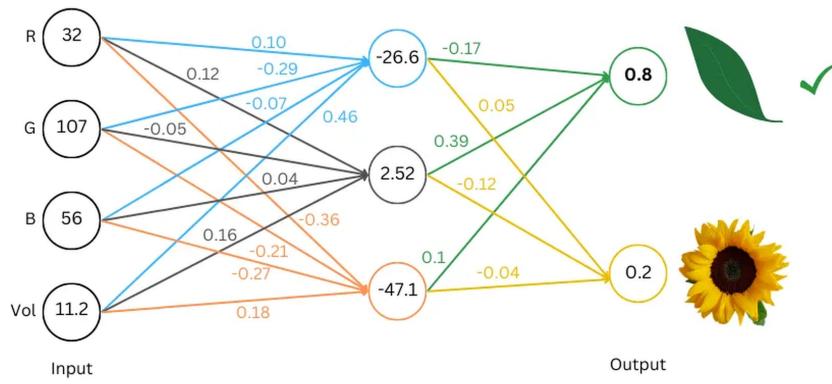
|     | Leaf | Flower |
|-----|------|--------|
| R   | 32   | 241    |
| G   | 107  | 200    |
| B   | 56   | 4      |
| Vol | 11.2 | 59.5   |

بیایید اکنون یک شبکه عصبی بسازیم که این طبقه‌بندی را انجام دهد. ما باید درباره تفسیر ورودی‌ها و خروجی‌ها تصمیم بگیریم. ورودی‌های ما از قبل اعداد هستند، بنابراین می‌توانیم آنها را مستقیماً به شبکه وارد کنیم. خروجی‌های ما دو شیء هستند: برگ و گل، که شبکه عصبی نمی‌تواند آنها را مستقیماً خروجی دهد. بیایید به چند روش که می‌توانیم در اینجا استفاده کنیم نگاه کنیم:

- ما می‌توانیم شبکه را طوری طراحی کنیم که یک عدد خروجی دهد. اگر عدد مثبت باشد، می‌گوییم که شیء یک برگ است، و اگر منفی باشد، می‌گوییم که یک گل است.

- یا، می‌توانیم شبکه را طوری طراحی کنیم که دو عدد خروجی دهد. ما اولین عدد را به عنوان عدد مربوط به برگ و دومی را به عنوان عدد مربوط به گل تفسیر می‌کنیم، و انتخاب ما عددی است که بزرگ‌تر باشد.

هر دو روش به شبکه اجازه می‌دهند که عدد یا اعدادی را خروجی دهد که می‌توانیم به عنوان برگ یا گل تفسیر کنیم. باید در اینجا روش دوم را انتخاب کنیم، زیرا به خوبی به سایر مواردی که بعداً بررسی خواهیم کرد تعمیم می‌یابد. و این یک شبکه عصبی است که با استفاده از این روش طبقه‌بندی را انجام می‌دهد. باید آن را بررسی کنیم:



## برخی اصطلاحات

نورون‌ها/گره‌ها: اعداد داخل دایره‌ها

وزن‌ها: اعداد رنگی روی خطوط

لایه‌ها: به مجموعه‌ای از نورون‌ها یک لایه گفته می‌شود. می‌توانید این شبکه را به عنوان شبکه‌ای با ۳

لایه در نظر بگیرید: لایه ورودی با ۴ نورون، لایه میانی با ۳ نورون، و لایه خروجی با ۲ نورون.

برای محاسبه پیش‌بینی یا خروجی از این شبکه (که به آن «عبور رو به جلو» یا forward pass می‌شود)، از سمت چپ شروع می‌کنید. ما داده‌های لازم برای نورون‌های لایه ورودی را در اختیار داریم. برای حرکت «رو به جلو» به لایه بعدی، عدد داخل دایره را در وزن مربوط به جفت نورون ضرب می‌کنید و همه آن‌ها را با هم جمع می‌کنید. ما محاسبات دایره‌های آبی و نارنجی را در بالا نشان داده‌ایم. با اجرای کل شبکه، می‌بینیم که عدد اول در لایه خروجی بزرگ‌تر است، بنابراین آن

را به صورت «شبکه این مقادیر (Vol، RGB) را به عنوان برگ طبقه‌بندی کرده است» تفسیر می‌کنیم. یک شبکه خوب آموزش دیده می‌تواند ورودی‌های مختلفی برای (Vol، RGB) بگیرد و شیء را به درستی طبقه‌بندی کند.

مدل هیچ در کی از اینکه برگ یا گل چیست، یا (Vol، RGB) چه هستند، ندارد. وظیفه آن این است که دقیقاً ۴ عدد را دریافت کند و دقیقاً ۲ عدد را خروجی دهد. این تفسیر ماست که ۴ عدد ورودی (Vol، RGB) هستند و همچنین تصمیم ماست که به اعداد خروجی نگاه کنیم و نتیجه بگیریم که اگر عدد اول بزرگ‌تر است، شیء یک برگ است و غیره. و در نهایت، این نیز به ما بستگی دارد که وزن‌های مناسب را انتخاب کنیم به‌طوری که مدل اعداد ورودی ما را بگیرد و دو عدد درست به ما بدهد، به‌طوری که وقتی آن‌ها را تفسیر می‌کنیم، به تفسیری که می‌خواهیم برسیم

یک اثر جانبی جالب این است که می‌توانید از همین شبکه استفاده کنید و به جای ورود (RGB)، Vol، ۴ عدد دیگر مانند پوشش ابر، رطوبت و غیره را وارد کنید و دو عدد خروجی را به عنوان «آفتای در یک ساعت» یا «بارانی در یک ساعت» تفسیر کنید. سپس، اگر وزن‌ها را به خوبی تنظیم کرده باشید، می‌توانید از همین شبکه برای انجام دو کار به‌طور همزمان استفاده کنید. طبقه‌بندی برگ/گل و پیش‌بینی باران در یک ساعت! شبکه فقط دو عدد به شما می‌دهد؛ اینکه آن را به عنوان طبقه‌بندی یا پیش‌بینی یا چیز دیگری تفسیر کنید، کاملاً به خودتان بستگی دارد.

موارد حذف شده برای ساده‌سازی (می‌توانید بدون کاهش قابلیت در ک، آن‌ها را نادیده بگیرید)

- لایه فعال‌سازی: یک نکته مهم که در این شبکه وجود ندارد، «لایه فعال‌سازی» است. این یک اصطلاح پیچیده برای این است که ما عدد داخل هر دایره را گرفته و یکتابع غیرخطی را بر روی آن اعمال می‌کنیم RELU یکتابع رایج است که در آن اگر عدد منفی باشد، آن را صفر می‌کنید و اگر مثبت باشد، آن را بدون تغییر می‌گذارید. بنابراین، در مثال ما در بالا، ما لایه میانی را گرفته و دو عدد (۶-۲۶ و ۱-۴۷) را قبل از حرکت به لایه بعدی با صفر جایگزین می‌کنیم. البته، ما باید وزن‌ها را دوباره آموزش دهیم تا شبکه دوباره مفید شود. بدون لایه فعال‌سازی، تمام جمع‌ها و ضرب‌ها در شبکه می‌توانند به یک لایه واحد کاهش یابند. در مورد ما، می‌توانید دایره سبز را به صورت جمع مستقیم RGB با برخی وزن‌ها بنویسید و به لایه میانی نیازی نخواهید داشت. این می‌تواند چیزی شبیه به:

$$(\dots \times -0.017 + 0.012 \times 0.039 - 0.036 \times 0.01) \times R + (-0.029 \times -0.017 - 0.005 \times 0.039 - 0.021 \times 0.01) \times G + \dots$$

باشد. این معمولاً ممکن نیست اگر یک غیرخطی بودن وجود داشته باشد. این به شبکه‌ها کمک می‌کند تا با موقعیت‌های پیچیده‌تر مقابله کنند.

- بایاس: به این عدد «بایاس» گفته می‌شود. بنابراین اگر بایاس برای گره آبی بالای ۰۰۲۵ باشد، مقدار در گره برابر خواهد بود با:

$$(۳۲ \times ۰۰۱۰) + (۱۰۷ \times -۰۰۲۹) + (۵۶ \times -۰۰۰۷) + (۱۱۰۲ \times ۰۰۴۶) + ۰۰۲۵ = -۲۶۰۳۵$$

کلمه «پارامترها» معمولاً برای اشاره به تمام این اعداد در مدل که نورون‌ها/گره‌ها نیستند، استفاده می‌شود.

- سافت‌مکس: اعداد را مثبت می‌کنیم و مجموع آن‌ها را برابر با ۱ می‌کنیم). اگر همه اعداد در لایه خروجی قبل مثبت بودند، یک راه برای انجام این کار این است که هر عدد را بر مجموع کل اعداد در لایه خروجی تقسیم کنیم. هرچند، معمولاً از یک تابع «سافت‌مکس» استفاده می‌شود که می‌تواند هم اعداد مثبت و هم منفی را مدیریت کند.

## این مدل‌ها چگونه آموزش داده می‌شوند؟

در مثال بالا، ما به طور جادویی وزن‌هایی داشتیم که به ما اجازه می‌داد داده‌ها را وارد مدل کنیم و خروجی خوبی بگیریم. اما این وزن‌ها چگونه تعیین می‌شوند؟ فرآیند تنظیم این وزن‌ها (یا «پارامترها») را «آموزش مدل» می‌نامند، و ما به مقداری داده آموزشی برای آموزش مدل نیاز داریم. فرض کنیم داده‌هایی داریم که در آن ورودی‌ها را داریم و می‌دانیم که هر ورودی به برگ یا گل مربوط می‌شود؛ این «داده آموزشی» ماست. از آنجا که برچسب برگ/گل را برای هر مجموعه از اعداد (Vol, R, G, B) داریم، این داده «برچسب گذاری شده» است.

نحوه کار به این صورت است:

- با اعداد تصادفی شروع کنید، یعنی هر پارامتر/وزن را به یک عدد تصادفی تنظیم کنید.
- اکنون می‌دانیم که وقتی داده‌های مربوط به برگ را وارد می‌کنیم ( $R = ۳۲, G = ۱۰۷, B = ۵۶$ , Vol = ۱۱۰۲) می‌خواهیم عدد بزرگ‌تری برای برگ در لایه خروجی داشته باشیم. فرض کنیم می‌خواهیم عدد مربوط به برگ ۰۰۸ و عدد مربوط به گل ۰۰۲ باشد (همان‌طور که در مثال بالا نشان داده شده، اما این اعداد فقط برای نشان‌دادن فرآیند آموزش هستند؛ در واقعیت ما نمی‌خواهیم ۰۰۸ و ۰۰۲ داشته باشیم. در واقعیت، این‌ها باید احتمالات باشند که در اینجا نیستند، و ما می‌خواهیم آن‌ها ۱ و ۰ باشند).
- ما اعدادی را که می‌خواهیم در لایه خروجی داشته باشیم، و اعدادی را که از پارامترهای تصادفی به دست می‌آوریم (که متفاوت از چیزی است که می‌خواهیم) می‌دانیم. بنابراین، برای همه نورون‌های لایه خروجی، تفاوت بین عدد مطلوب و عدد فعلی را می‌گیریم. سپس این

تفاوت‌ها را جمع می‌کنیم. به عنوان مثال، اگر لایه خروجی در دو نورون ۰۰۶ و ۰۰۴ باشد، داریم:  $002 = 004 - 008$  و  $002 = 004 - 006$ ، بنابراین مجموع تفاوت‌ها ۰۰۴ می‌شود (علامت منفی را قبل از جمع نادیده می‌گیریم). می‌توانیم این را «خطا (Loss)» بنامیم. ایده‌آل این است که خطا نزدیک به صفر باشد، یعنی می‌خواهیم «خطا را کمینه کنیم».

- پس از محاسبه خطا، می‌توانیم هر پارامتر را کمی تغییر دهیم تا بینیم افزایش یا کاهش آن خطرا را افزایش یا کاهش می‌دهد. این به «گرادیان» آن پارامتر معروف است. سپس می‌توانیم هر یک از پارامترهای میزان کمی در جهت حرکت دهیم که خطای کاهش یابد (جهت گرادیان). پس از اینکه همه پارامترها را کمی جایه‌جا کردیم، باید خطای کمتر شود.
- این فرآیند را تکرار کنید و خطای کاهش دهید، و در نهایت مجموعه‌ای از وزن‌ها/پارامترها را خواهید داشت که «آموزش دیده» هستند. این کل فرآیند «نزول گرادیان» نامیده می‌شود.

### چند نکته:

- شما اغلب چندین مثال آموزشی دارید، بنابراین وقتی وزن‌ها را کمی تغییر می‌دهید تا خطای را برای یک مثال کمینه کنید، ممکن است خطای برای مثال دیگری بدتر کنید. راه مقابله با این مسئله این است که خطای را به صورت میانگین خطای بر روی همه مثال‌ها تعریف کنید و سپس گرادیان را بر روی آن میانگین خطای بگیرید. این کار میانگین خطای را بر روی کل مجموعه داده آموزشی کاهش می‌دهد. هر چرخه چنین فرآیندی یک «دوره (Epoch)» نامیده می‌شود. سپس می‌توانید دوره‌ها را تکرار کنید تا وزن‌هایی را بیابید که میانگین خطای کاهش می‌دهند.
- ما در واقع نیازی به «جایه‌جا کردن وزن‌ها» برای محاسبه گرادیان هر وزن نداریم می‌توانیم آن را از فرمول استنتاج کنیم (مثالاً اگر وزن در مرحله آخر ۰۰۱۷ باشد و مقدار نورون مثبت باشد و ما یک عدد بزرگ‌تر در خروجی بخواهیم، می‌بینیم که افزایش این عدد به ۰۰۱۸ کمک می‌کند).

در عمل، آموزش شبکه‌های عمیق یک فرآیند سخت و پیچیده است زیرا گرادیان‌ها می‌توانند به راحتی از کنترل خارج شوند و در حین آموزش به صفر یا بی‌نهایت برسند (که به مشکلات «ناپدید شدن گرادیان» و «منفجر شدن گرادیان» معروف است). تعریف ساده خطایی که در اینجا درباره آن صحبت کردیم کاملاً معتبر است، اما به ندرت استفاده می‌شود زیرا فرم‌های تابعی بهتری وجود دارند که برای اهداف خاص به خوبی کار می‌کنند. با مدل‌های مدرن که حاوی میلیاردها پارامتر هستند، آموزش یک مدل به منابع محاسباتی عظیمی نیاز دارد که مشکلات خاص خود را دارد (محدودیت‌های حافظه، موازی‌سازی و غیره).

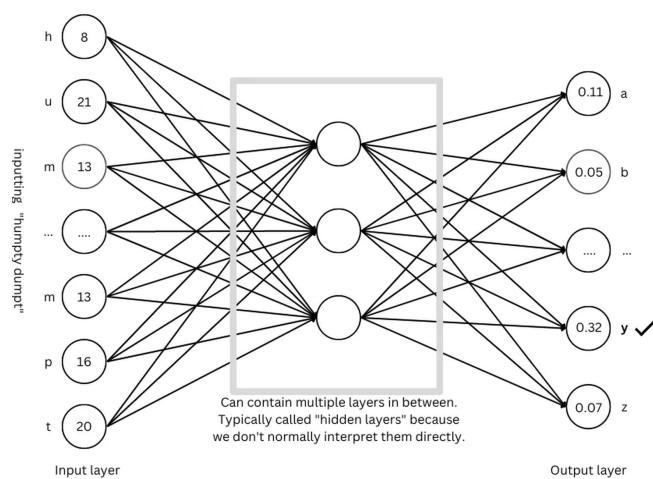
## چگونه همه این‌ها به تولید زبان کمک می‌کنند؟

به یاد داشته باشید، شبکه‌های عصبی تعدادی عدد را دریافت می‌کنند، بر اساس پارامترهای آموزش دیده محاسباتی انجام می‌دهند، و تعدادی عدد دیگر را خروجی می‌دهند. همه چیز درباره تفسیر و آموزش پارامترها (یعنی تنظیم آن‌ها به اعداد مشخص) است. اگر بتوانیم دو عدد را به عنوان «برگ/گل» یا «باران یا آفتاب در یک ساعت» تفسیر کنیم، می‌توانیم آن‌ها را به عنوان «حرف بعدی در یک جمله» نیز تفسیر کنیم.

اما در زبان انگلیسی بیش از ۲۶ حرف وجود دارد، بنابراین باید تعداد نورون‌ها در لایه خروجی را گسترش دهیم، مثلاً به ۴۶ حرف در زبان انگلیسی (باید چند نماد مانند فاصله، نقطه و غیره را نیز اضافه کنیم). هر نورون می‌تواند به یک کاراکتر مربوط باشد و ما به (حدود ۴۶) نورون در لایه خروجی نگاه می‌کنیم و می‌گوییم کاراکتری که به نورونی با بزرگ‌ترین عدد در لایه خروجی مربوط است، کاراکتر خروجی است. اکنون شبکه‌ای داریم که می‌تواند برشی ورودی‌ها را بگیرد و یک کاراکتر خروجی دهد.

چه می‌شود اگر ورودی در شبکه خود را با این کاراکترها جایگزین کنیم: «Humpty Dumpty» و از آن بخواهیم یک کاراکتر خروجی دهد و آن را به عنوان «پیشنهاد شبکه برای کاراکتر بعدی در دنباله‌ای که وارد کردۀ ایم» تفسیر کنیم. احتمالاً می‌توانیم وزن‌ها را بخوبی تنظیم کنیم تا «y» را خروجی دهد! به این ترتیب «Humpty Dumpty» کامل می‌شود. به جز یک مشکل، چگونه می‌توانیم این لیست از کاراکترها را در شبکه وارد کنیم؟ شبکه ما فقط اعداد را می‌پذیرد!!

یک راه حل ساده این است که به هر کاراکتر یک عدد اختصاص دهیم. فرض کنیم  $a = 1$ ,  $b = 2$ ,  $c = 3$  و همین‌طور ادامه دهیم. اکنون می‌توانیم «humpty dumpty» را وارد کنیم و آن را آموزش دهیم تا به ما «y» بدهد. شبکه ما به این صورت است:



خوب، اکنون می‌توانیم با ارائه یک لیست از کاراکترها به شبکه، یک کاراکتر جلوتر را پیش‌بینی کنیم. ما می‌توانیم از این واقعیت برای ساختن یک جمله کامل استفاده کنیم. برای مثال، وقتی «y» را پیش‌بینی کردیم، می‌توانیم آن «y» را به لیست کاراکترهایی که داریم اضافه کنیم و آن را به شبکه بدهیم و از آن بخواهیم کاراکتر بعدی را پیش‌بینی کنند. و اگر به خوبی آموزش دیده باشد، باید یک فاصله به ما بدهد، و همین طور ادامه باید. در نهایت، باید بتوانیم به صورت بازگشتی «Humpty Dumpty»  $\text{sat on a wall}$  را تولید کنیم. ما هوش مصنوعی مولد داریم. علاوه بر این، اکنون شبکه‌ای داریم که قادر به تولید زبان است! البته، هیچ‌کس واقعاً اعداد اختصاص داده شده به صورت تصادفی را وارد نمی‌کند و ما در ادامه به طرح‌های منطقی‌تری خواهیم رسید. اگر نمی‌توانید صبر کنید، می‌توانید به بخش one-hot encoding در پیوست مراجعه کنید.

خواندنگان هوشمند متوجه خواهند شد که ما در واقع نمی‌توانیم «Humpty Dumpty» را به شبکه وارد کنیم، زیرا طبق نمودار، لایه ورودی فقط ۱۲ نورون دارد، هر یک برای یک کاراکتر در «humpty dumpt» (شامل فاصله). پس چگونه می‌توانیم «y» را برای عبور بعدی وارد کنیم؟ قرار دادن یک نورون سیزدهم در آنجا مرا مجبور می‌کند که کل شبکه را تغییر دهیم، که عملی نیست. راه حل ساده است؛ بیانیه «I» را حذف کنیم و ۱۲ کاراکتر اخیر را ارسال کنیم. بنابراین ما «umpty»  $\text{umpty dumpty}$  را ارسال می‌کنیم و شبکه یک فاصله را پیش‌بینی خواهد کرد. سپس «dumpty»  $\text{dumpty dumpty}$  را وارد می‌کنیم و شبکه یک «S» تولید می‌کند و همین طور ادامه می‌باید. به این شکل به نظر می‌رسد:

| Input (underlined)                | Output    |
|-----------------------------------|-----------|
| <u>humpty_dumpt</u>               | → y       |
| <u>humpty_dumpty</u> .            | → <space> |
| <u>humpty_dumpty_</u>             | → s       |
| <u>humpty_dumpty_s</u>            | → a       |
| <u>humpty_dumpty_sa</u>           | → t       |
| ...                               | ..        |
| humpty dumpty <u>sat on a wal</u> | → l       |

ما در خط آخر با وارد کردن تنها «sat on the wal» به مدل، اطلاعات زیادی را دور می‌اندازیم. پس شبکه‌های پیشرفته امروزی چه می‌کنند؟ کم و بیش دقیقاً همین کار را. طول ورودی‌هایی که می‌توانیم به یک شبکه بدهیم ثابت است (که توسط اندازه لایه ورودی تعیین می‌شود). این به «طول زمینه» (context length) معروف است زمینه‌ای که به شبکه ارائه می‌شود تا پیش‌بینی‌های آینده را انجام دهد. شبکه‌های مدرن می‌توانند طول زمینه بسیار بزرگی داشته باشند (چندین هزار کلمه) و این کمک می‌کند. روش‌هایی برای وارد کردن دنباله‌های با طول بینهایت وجود دارد، اما عملکرد آن روش‌ها، هرچند چشمگیر، از آن زمان توسط مدل‌های دیگر با طول زمینه بزرگ (اما ثابت) پشت سر گذاشته شده است.

نکته دیگری که خوانندگان دقیق متوجه خواهند شد این است که ما برای ورودی‌ها و خروجی‌ها، تفاسیر متفاوتی برای حروف یکسان داریم! برای مثال، وقتی «h» را وارد می‌کنیم، به سادگی آن را با عدد ۸ نشان می‌دهیم، اما در لایه خروجی از مدل نمی‌خواهیم که یک عدد واحد را خروجی دهد (۸) برای «h»، (۹) برای «i» و غیره...)، بلکه از مدل می‌خواهیم که ۲۶ عدد را خروجی دهد و سپس می‌بینیم کدام یک بیشترین است و اگر هشتمنی عدد بیشترین باشد، خروجی را به عنوان «h» تفسیر می‌کنیم. چرا از یک تفسیر یکسان و سازگار در هر دو طرف استفاده نمی‌کنیم؟ می‌توانستیم این کار را بکنیم، اما در مورد زبان، آزادی در انتخاب بین تفاسیر مختلف به شما شانس بهتری برای ساخت مدل‌های بهتر می‌دهد. و اتفاقاً موثرترین تفاسیر شناخته شده کنونی برای ورودی و خروجی متفاوت هستند. در واقع، نحوه وارد کردن اعداد در این مدل بهترین روش برای انجام آن نیست؛ ما به زودی به روش‌های بهتری برای این کار خواهیم پرداخت.

## چه چیزی باعث می‌شود مدل‌های زبانی بزرگ به این خوبی کار کنند؟

تولید «Humpty Dumpty sat on a wall» کاراکتر به کاراکتر، فاصله زیادی با آنچه مدل‌های زبانی بزرگ مدرن می‌توانند انجام دهند دارد. تفاوت‌ها و نوآوری‌های زیادی وجود دارد که ما را از هوش مصنوعی مولد ساده‌ای که در بالا بحث کردیم به ربات‌های شبیه انسان می‌رساند. بیاید آن‌ها را مرور کنیم:

### تعییه‌ها (Embeddings)

به یاد دارید که گفتیم نحوه‌ای که ما کاراکترها را به مدل وارد می‌کنیم بهترین روش نیست. ما فقط به طور دلخواه یک عدد برای هر کاراکتر انتخاب کردیم. چه می‌شود اگر اعداد بهتری را اختصاص دهیم که به ما امکان دهد شبکه‌های بهتری را آموزش دهیم؟ چگونه این اعداد بهتر را پیدا کنیم؟ در اینجا یک ترفند هوشمندانه وجود دارد:

وقتی مدل‌های بالا را آموزش می‌دادیم، روش ما این بود که وزن‌ها را جایه‌جا می‌کردیم و می‌دیدیم که در نهایت به خطای کمتری می‌رسیم. سپس وزن‌های آرامی و به صورت بازگشتی تغییر می‌دادیم. در هر مرحله ما:

- ورودی‌ها را وارد می‌کردیم
- لایه خروجی را محاسبه می‌کردیم
- آن را با خروجی ایده‌آل خود مقایسه می‌کردیم و خطای میانگین را محاسبه می‌کردیم
- وزن‌ها را تنظیم می‌کردیم و دوباره شروع می‌کردیم

در این فرآیند، ورودی‌ها ثابت بودند. این در زمانی که ورودی‌ها (Vol, RGB) بودند منطقی بود. اما اعدادی که اکنون برای  $a, b, c$  و غیره قرار می‌دهیم، به طور دلخواه توسط ما انتخاب شده‌اند. چه می‌شود اگر در هر تکرار علاوه بر جایه‌جایی وزن‌ها، ورودی‌ها را نیز جایه‌جا کنیم و بیسیم آیا می‌توانیم با استفاده از یک عدد متفاوت برای نمایش « $a$ » و غیره، به خطای کمتری برسیم؟ ما قطعاً با کاهش خطای مدل را بهتر می‌کنیم (این همان جهتی است که ورودی « $a$ » را در آن حرکت دادیم، به صورت طراحی شده). اساساً، نزول گرادیان را نه تنها بر روی وزن‌ها، بلکه بر روی نمایش عددی ورودی‌ها نیز اعمال می‌کنیم، زیرا این اعداد به هر حال به طور دلخواه انتخاب شده‌اند. به این کار «تعییه» (embedding) گفته می‌شود. این یک نگاشت از ورودی‌ها به اعداد است و همان‌طور که دیدید، نیاز به آموزش دارد. فرآیند آموزش یک تعییه بسیار شیوه به آموزش یک پارامتر است. یک مزیت بزرگ این است که وقتی یک تعییه را آموزش دادیم، می‌توانید آن را در مدل دیگری نیز استفاده کنید، اگر بخواهید. به یاد داشته باشید که باید به طور مداوم از همان تعییه برای نمایش یک توکن/کاراکتر/کلمه استفاده کنید.

ما درباره تعییه‌هایی صحبت کردیم که فقط یک عدد به ازای هر کاراکتر دارند. با این حال، در واقعیت تعییه‌ها بیش از یک عدد دارند. این به این دلیل است که سخت است غایی یک مفهوم را با یک عدد تنها به تصویر کشید. اگر به مثال برگ و گل خود نگاه کنیم، ما چهار عدد برای هر شیء داریم (اندازه لایه ورودی). هر یک از این چهار عدد یک ویژگی را منتقل می‌کند و مدل توانست از همه آن‌ها برای حدس مؤثر شیء استفاده کند. اگر فقط یک عدد داشتیم، مثلاً کanal قرمز رنگ، ممکن بود برای مدل بسیار سخت‌تر باشد. ما در اینجا سعی داریم زبان انسانی را به تصویر بکشیم ما به بیش از یک عدد نیاز خواهیم داشت.

بنابراین به جای نمایش هر کاراکتر با یک عدد، شاید بتوانیم آن را با چندین عدد نمایش دهیم تا غنای آن را به تصویر بکشیم؟ بیایید به هر کاراکتر چندین عدد اختصاص دهیم. یک مجموعه مرتب از اعداد را «بردار» می‌نامیم (مرتب به این معنا که هر عدد دارای یک موقعیت است و اگر موقعیت دو عدد را جایجا کنیم، بردار متفاوتی به دست می‌آوریم. این در مورد داده‌های برگ/گل ما صادق بود؛ اگر اعداد  $R$  و  $G$  را برای برگ جایجا می‌کردیم، رنگ متفاوتی به دست می‌آوردیم و دیگر همان بردار قبلی نبود). طول یک بردار به سادگی تعداد اعدادی است که در آن قرار دارد. ما به هر کاراکتر یک بردار اختصاص خواهیم داد. دو سؤال مطرح می‌شود:

- اگر به هر کاراکتر به جای یک عدد، یک بردار اختصاص دهیم، چگونه «humpty dumpty» را به شبکه وارد می‌کنیم؟ پاسخ ساده است. فرض کنیم به هر کاراکتر یک بردار ۱۰ عددی اختصاص داده‌ایم. در این صورت، به جای اینکه لایه ورودی ۱۲ نورون داشته باشد، ما ۱۲۰ نورون خواهیم داشت، زیرا هر یک از ۱۲ کاراکتر در «humpty dumpty» دارای ۱۰ عدد برای ورودی است. اکنون نورون‌ها را در کنار هم قرار می‌دهیم و آمده‌ایم.

- چگونه این بردارها را پیدا کنیم؟ خوشبختانه، ما به تازگی یاد گرفتیم که چگونه اعداد تعییه را آموزش دهیم. آموزش یک بردار تعییه تفاوتی ندارد. اکنون ۱۲ ورودی به جای ۱۲ ورودی دارید، اما تمام کاری که می‌کنید این است که آن‌ها را جایه‌جا می‌کنید تا بیسند چگونه می‌توانید خطرا را کمینه کنید. سپس اولین ۱۰ عدد از آن‌ها را می‌گیرید و آن بردار مربوط به «*h*» است و به همین ترتیب.

البته تمام بردارهای تعییه باید طول یکسانی داشته باشند، در غیر این صورت ما راهی برای وارد کردن تمام ترکیب‌های کاراکترها به شبکه نخواهیم داشت. مثلاً «*humpty dumpty*» و در تکرار بعدی «*umpty dumpty*» در هر دو مورد ما ۱۲ کاراکتر را به شبکه وارد می‌کنیم و اگر هر یک از ۱۲ کاراکتر با بردارهای طول ۱۰ نمایش داده نمی‌شوند، نمی‌توانستیم به طور قابل اعتماد همه آن‌ها را به یک لایه ورودی با طول ۱۲ وارد کنیم. باید این بردارهای تعییه را بصری‌سازی کنیم:

|        | a    | b    | z    |
|--------|------|------|------|
| row 1  | 0.9  | 3.0  | -1.0 |
| row 2  | -1.2 | -0.4 | 0.7  |
| .      | .    | .    | .    |
| .      | .    | .    | .    |
| .      | .    | .    | .    |
| .      | .    | .    | .    |
| row 10 | 0.1  | 5.1  | -0.3 |

باید یک مجموعه مرتب از بردارهای هماندازه را ماتریس بنامیم این ماتریس بالا به عنوان یک ماتریس تعییه شناخته می‌شود. شما به آن یک شماره ستون که به حرف شما مربوط است می‌دهید، و با نگاه به آن ستون در ماتریس، برداری را دریافت می‌کنید که برای نمایش آن حرف استفاده می‌کنید. این می‌تواند به طور کلی تر برای تعییه هر مجموعه دلخواهی از اشیا اعمال شود شما فقط نیاز دارید که به تعداد اشیایی که دارید، ستون در این ماتریس داشته باشید.

## توکن‌سازهای زیرواژه

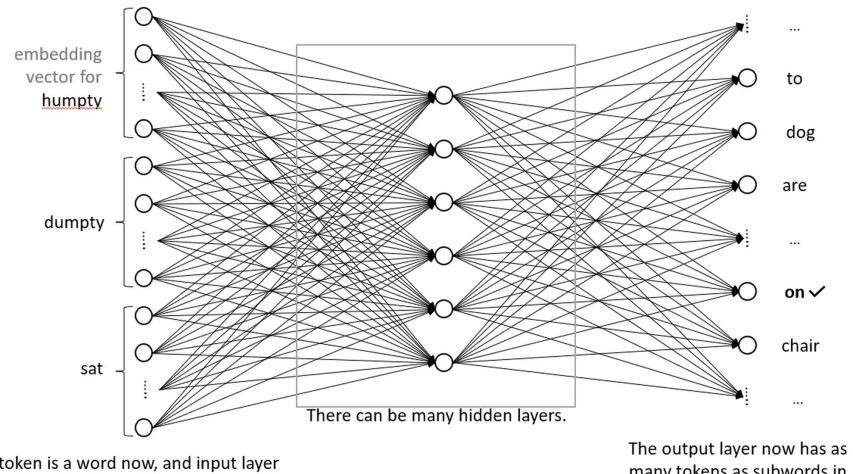
تاکنون، ما از کاراکترها به عنوان بلوک‌های ساختمانی پایه زبان استفاده می‌کردیم. این روش محدودیت‌هایی دارد. وزن‌های شبکه عصبی باید کار سنجی‌نی را انجام دهند، جایی که باید معنای توالی خاصی از کاراکترها (یعنی کلمات) را در کنار یکدیگر در ک کنند و سپس آن‌ها را در کنار کلمات دیگر قرار دهند. اگر به طور مستقیم به کلمات تعییه‌هایی اختصاص دهیم و شبکه را به پیش‌بینی کلمه بعدی و ادار کنیم، چه می‌شود؟ شبکه به هر حال چیزی بیشتر از اعداد را در ک کنند، بنابراین

می توانیم یک بردار ۱۰ بعدی به هر یک از کلمات "humpty"، "sat" و "on" و غیره اختصاص دهیم و سپس دو کلمه را به آن تغذیه کنیم تا کلمه بعدی را به ما بدهد. "توکن" اصطلاحی برای یک واحد منفرد است که ما آن را تعییه می کنیم و سپس به مدل تغذیه می کنیم. مدل های ما تاکنون از کاراکترها به عنوان توکن استفاده می کردند، اکنون پیشنهاد می کنیم از کل کلمات به عنوان یک توکن استفاده کنیم (البته می توانید از کل جملات یا عبارات به عنوان توکن استفاده کنید).

استفاده از توکن سازی کلمات یک اثر عمیق بر مدل ما دارد. بیش از ۱۸۰ هزار کلمه در زبان انگلیسی وجود دارد. با استفاده از طرح تفسیر خروجی ما که دارای یک نورون برای هر خروجی ممکن است، ما به صدها هزار نورون در لایه خروجی نیاز داریم، نه حدود ۲۶ نورون. با توجه به اندازه لایه های پنهان مورد نیاز برای دستیابی به نتایج معنadar برای شبکه های مدرن، این مسئله کمتر فوری می شود. با این حال، شایان ذکر است که از آنجایی که ما هر کلمه را جداگانه رفتار می کنیم و با تعییه های عددی تصادفی برای هر یک شروع می کنیم، کلمات بسیار مشابه (مانند "cat" و "cats") بدون هیچ ارتباطی شروع می شوند. انتظار می رود که تعییه های دو کلمه به یکدیگر نزدیک باشند - که بدون شک مدل یاد خواهد گرفت. اما، آیا می توانیم به نحوی از این شاهد آشکار برای شروع سریع تر و ساده سازی مسائل استفاده کنیم؟

بله، می توانیم. رایج ترین طرح تعییه در مدل های زبانی امروزی روشنی است که در آن کلمات را به زیر کلمات تقسیم می کنید و سپس آن ها را تعییه می کنید. در مثال "cat" کلمه "cats" را به دو توکن "cat" و "s" تقسیم می کنیم. اکنون برای مدل آسان تر است که مفهوم "s" را در کنار سایر کلمات آشنا و غیره در گ کنند. این همچنین تعداد توکن های مورد نیاز ما را کاهش می دهد (sentencpiece) یک توکن ساز رایج با گزینه های اندازه واژگان در دهها هزار تا صدها هزار کلمه در انگلیسی است). یک توکن ساز چیزی است که متن ورودی شما (مثلاً "Humpty Dumpty") را می گیرد و آن را به توکن ها تقسیم می کند و اعداد مربوطه را به شما می دهد که برای جستجوی بردار تعییه برای آن توکن در ماتریس تعییه نیاز دارد. به عنوان مثال، در مورد "humpty dumpty"، اگر از توکن ساز سطح کاراکتر استفاده می کنیم و ماتریس تعییه خود را مانند تصویر بالا مرتب کرده ایم، توکن ساز ابتدا "humpty dumpty" را به کاراکترها [h<sup>?</sup>, u<sup>?</sup>, ..., t<sup>?</sup>] تقسیم می کند و سپس اعداد [۲۱, ۸, ..., ۲۰] را به شما می دهد زیرا برای جستجوی بردار تعییه برای h<sup>?</sup> باید به ستون هشتم ماتریس تعییه مراجعه کنید (بردار تعییه چیزی است که به مدل تغذیه می کنید، نه عدد ۸، برخلاف قبل). ترتیب ستون ها در ماتریس کاملاً بی ربط است، می توانیم هر ستونی را به h<sup>?</sup> اختصاص دهیم و تا زمانی که هر بار که h<sup>?</sup> را وارد می کنیم، همان بردار را جستجو کنیم، باید خوب باشیم. توکن سازها فقط به ما یک عدد دلخواه (اما ثابت) می دهند تا جستجو را آسان کنند. وظیفه اصلی که ما واقعاً به آن ها نیاز داریم، تقسیم جمله به توکن ها است.

با استفاده از تعییه ها و توکن سازی زیرواژه، یک مدل می تواند به شکل زیر باشد:



بخش‌های بعدی با پیشرفت‌های اخیر در مدل‌سازی زبان و مواردی که باعث شده‌اند LLM‌ها به اندازه امروز قدرتمند شوند، سروکار دارند. با این حال، برای درک این موارد، چند مفهوم ریاضی پایه وجود دارد که باید بدانید. این مفاهیم عبارتند از:

- ماتریس‌ها و ضرب ماتریس‌ها
  - مفهوم کلی توابع در ریاضیات
  - توان رسانی اعداد (مثلاً  $aaa$ )
  - مانگن: نمونه، واریانس، و انحراف

خود توجیہ

تاکنون ما فقط یک ساختار ساده شبکه عصبی (به نام شبکه فیدفوروارد) را دیده‌ایم که شامل تعدادی لایه است و هر لایه کاملاً به لایه بعدی متصل است (یعنی خطی بین هر دو نورون در لایه‌های متوالی وجود دارد) و فقط به لایه بعدی متصل است (مثلاً هیچ خطی بین لایه ۱ و لایه ۲ وغیره وجود ندارد). با این حال، همانطور که می‌توانید تصور کنید، هیچ چیز مانع از حذف یا ایجاد اتصالات دیگر نمی‌شود. یا حتی ساختن ساختارهای پیچیده‌تر. یا باید یک ساختار خاص و مهم را بررسی کنیم: خودتوجی.

اگر به ساختار زبان انسان نگاه کنیم، کلمه بعدی که می‌خواهیم پیش‌بینی کنیم به همه کلمات قبلی بستگی دارد. با این حال، آن‌ها ممکن است تا حد زیادی به برخی از کلمات قبل از خود وابسته باشند. Damian had a secret child, a girl, and” به عنوان مثال، اگر سعی کنیم کلمه بعدی در جمله: he had written in his will that all his belongings, along with the magical orb,

خبر خوب این است که مدل فیدفوروارد ساده ما به همه کلمات در متن متصل است و بنابراین می‌تواند وزن‌های مناسب برای کلمات مهم را یاد بگیرد. اما مشکل اینجاست که وزن‌های اتصال موقعیت‌های خاص در مدل ما از طریق لایه‌های فیدفوروارد ثابت هستند (برای هر موقعیت). اگر کلمه مهم همیشه در یک موقعیت ثابت بود، وزن‌ها را به درستی یاد می‌گرفت و مشکلی نداشتم. با این حال، کلمه مرتبط با پیش‌بینی بعدی می‌تواند در هر نقطه از سیستم باشد. می‌توانیم جمله بالا را پارافریز کنیم و هنگام حدس زدن "her vs his"، یک کلمه بسیار مهم برای این پیش‌بینی "boy/girl" خواهد بود، صرف نظر از اینکه در کجا در آن جمله ظاهر شود. بنابراین، ما به وزن‌های نیاز داریم که نه تنها به موقعیت بلکه به محتوای آن موقعیت نیز بستگی داشته باشند. چگونه به این هدف دست می‌یابیم؟

خودتوجهی کاری شبیه جمع کردن بردارهای تعییه شده برای هر یک از کلمات انجام می‌دهد، اما به جای اینکه آنها را مستقیماً با هم جمع کند، به هر کدام وزنی اعمال می‌کند. بنابراین اگر بردارهای تعییه شده برای  $x_1$ ,  $x_2$ ,  $x_3$  باشند، آنگاه هر کدام را با یک وزن (عدد) ضرب می‌کند و سپس آنها را با هم جمع می‌کند. چیزی شبیه به  $output = u_1x_1 + u_2x_2 + u_3x_3$  که خروجی خروجی خود توجه است. اگر وزن‌ها را به صورت  $u_1$ ,  $u_2$ ,  $u_3$  بنویسیم به طوری که  $output = u_1x_1 + u_2x_2 + u_3x_3$  باشد، چگونه این وزن‌های  $u_1$ ,  $u_2$ ,  $u_3$  را پیدا کنیم؟

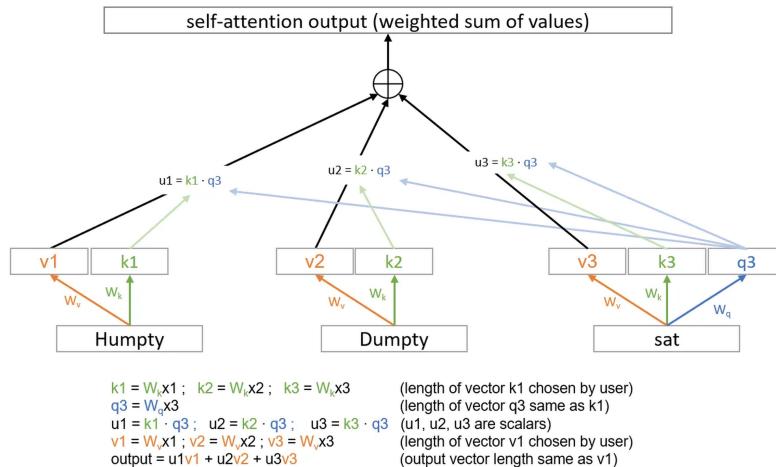
در حالت ایده آل، می‌خواهیم این وزن‌ها به برداری که می‌خواهیم اضافه کنیم وابسته باشند، همانطور که دیدیم برخی ممکن است مهم‌تر از برخی دیگر باشند. اما مهم برای چه کسی؟ برای کلمه‌ای که می‌خواهیم پیش‌بینی کنیم. بنابراین می‌خواهیم وزن‌ها نیز به کلمه‌ای که می‌خواهیم پیش‌بینی کنیم وابسته باشند. حالا این یک مسئله است، ما لبته کلمه‌ای را که می‌خواهیم پیش‌بینی کنیم قبل از اینکه آن را پیش‌بینی کنیم نمی‌دانیم. بنابراین، خود توجه از کلمه بلافصله قبل از کلمه‌ای که می‌خواهیم پیش‌بینی کنیم استفاده می‌کند، یعنی آخرین کلمه در جمله موجود (من واقعاً نمی‌دانم چرا این و چرا چیز دیگری)، اما بسیاری از چیزها در یادگیری عمیق آزمون و خطا هستند و من گمان می‌کنم این خوب کار می‌کند).

عالی، بنابراین ما به وزن‌هایی برای این بردارها نیاز داریم و می‌خواهیم هر وزن به کلمه‌ای که در حال جمع‌آوری آن هستیم و کلمه بلافصله قبل از کلمه‌ای که می‌خواهیم پیش‌بینی کنیم بستگی داشته باشد. اساساً، ما به تابعی نیاز داریم  $F(x_1, x_3) = u_1$  که در آن  $x_1$  کلمه‌ای است که می‌خواهیم وزن دهیم و  $x_3$  آخرین کلمه در دنباله‌ای است که داریم (فرض می‌کنیم فقط ۳ کلمه داریم). اکنون، یک راه مستقیم برای دستیابی به این هدف این است که یک بردار برای  $x_1$  داشته باشیم (آن را  $k_1$

بنامیم) و یک بردار جداگانه برای  $x_3$  (آن را  $q_3$  بنامیم) و سپس به سادگی حاصلضرب نقطه‌ای آنها را بگیریم. این به ما یک عدد می‌دهد و به هر دو  $x_1$  و  $x_3$  بستگی دارد. چگونه این بردارهای  $k_1$  و  $q_3$  را بدست می‌آوریم؟ ما یک شبکه عصبی تک لایه کوچک می‌سازیم تا از  $x_1$  به  $k_1$  برویم (یا از  $x_2$  به  $k_2$ ، از  $x_3$  به  $k_3$  وغیره). و ما شبکه دیگری می‌سازیم که از  $x_3$  به  $q_3$  وغیره می‌رود. با استفاده از نماد ماتریسی ما اساساً با ماتریس‌های وزنی  $W_k$  و  $W_q$  مواجه می‌شویم به طوری که  $k_1 = W_k x_1$  و  $q_1 = W_q x_1$  وغیره. اکنون می‌توانیم حاصلضرب نقطه‌ای  $k_1$  و  $q_3$  را برای به دست آوردن یک اسکالر بگیریم، بنابراین

$$u_1 = F(x_1, x_3) = W_k x_1 W_q x_3$$

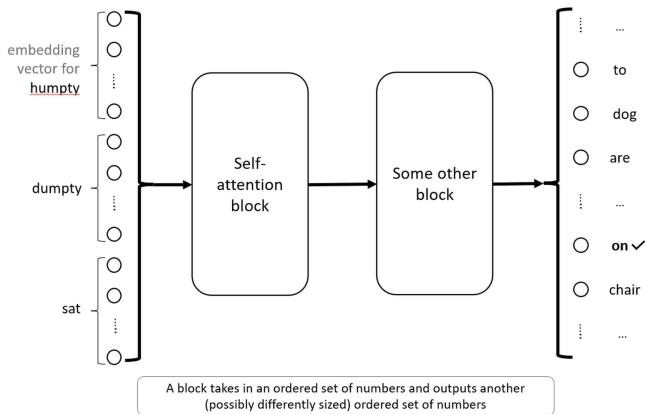
یک چیز اضافی که در خود توجه اتفاق می‌افتد این است که ما مستقیماً مجموع وزنی بردارهای تعییه شده را نمی‌گیریم. در عوض، مجموع وزنی برخی از "ارزش" آن بردار تعییه شده را می‌گیریم، که توسط یک شبکه تک لایه کوچک دیگر به دست می‌آید. معنای این کار شبیه به  $k_1$  و  $q_1$  است، ما اکنون  $v_1$  را نیز برای کلمه  $x_1$  داریم و آن را از طریق ماتریس  $W_v$  به دست می‌آوریم به طوری که  $v_1 = W_v x_1$ . سپس این  $v_1$  جمع می‌شود. بنابراین همه چیز به این شکل است اگر فقط ۳ کلمه داشته باشیم و سعی کنیم چهارمین کلمه را پیش‌بینی کنیم:



علاوه بر این، علامت جمع نشان دهنده جمع ساده بردارها است، به این معنی که آنها باید طول یکسانی داشته باشند. یک اصلاح نهایی که در اینجا نشان داده نشده است این است که مقادیر اسکالر  $u_1$ ,  $u_2$ ,  $u_3$  وغیره لزوماً به ۱ جمع نمی‌شوند. اگر نیاز داریم که آنها وزن باشند، باید آنها را به ۱ جمع کنیم. بنابراین ما از یک ترفند آشنا در اینجا استفاده خواهیم کرد و از تابع softmax استفاده خواهیم کرد.

این خود توجه است. همچنین توجه متقابل وجود دارد که در آن می‌توانید  $q_3$  را از آخرین کلمه بیاورید، اما  $k$  ها و  $v$  ها می‌توانند کاملاً از جمله دیگری بیایند. این به عنوان مثال در کارهای ترجمه ارزشمند است. حالا ما می‌دانیم توجه چیست.

کل این کار را می توان اکنون در یک جعبه قرار داد و آن را "بلوک خود توجه" نامید. اساساً، این بلوک خود توجه بردارهای تعییه شده را می گیرد و یک بردار خروجی واحد با هر طول دلخواه کاربر را خارج می کند. این بلوک دارای سه پارامتر،  $W_v$ ,  $W_q$ ,  $W_k$  است - نیازی نیست پیچیده تر از این باشد. بسیاری از این بلوک‌ها در ادبیات یادگیری ماشین وجود دارند و معمولاً با جعبه‌هایی در نمودارها با نام خودشان نشان داده می‌شوند. چیزی شبیه به این:



یکی از چیزهایی که در مورد خود توجه خواهد شد این است که موقعیت چیزها تاکنون مرتبط به نظر نمی‌رسد. ما از همان  $\mathbb{H}$  در سراسر هیئت مدیره استفاده می‌کیم و بنابراین تعویض  $\text{Humpty}$  و  $\text{Dumpty}$  در واقع تفاوتی ایجاد نخواهد کرد - همه اعداد یکسان خواهند بود. این بدان معناست که در حالی که توجه می‌تواند بهمراه چه چیزی توجه کند، این به موقعیت کلمه بستگی نخواهد داشت. با این حال، می‌دانیم که موقعیت کلمات در انگلیسی مهم است و احتمالاً می‌توانیم با دادن حس موقعیت یک کلمه به مدل، عملکرد را بهبود بخشیم.

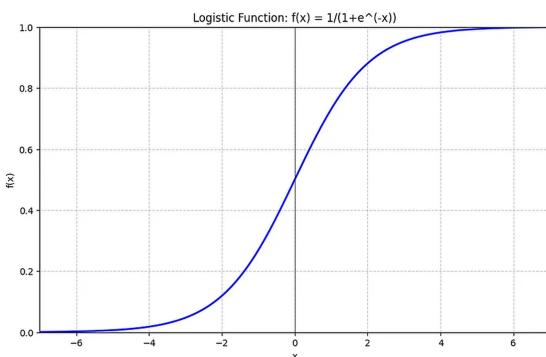
و بنابراین، هنگامی که از توجه استفاده می‌شود، اغلب بردارهای تعییه را مستقیماً به بلوک خود توجه تغذیه نمی‌کنیم. بعداً خواهیم دید که چگونه "رمزگذاری موقعیتی" قبل از تغذیه به بلوک‌های توجه به بردارهای تعییه اضافه می‌شود.

توجه برای افراد پیشرفته: کسانی که این اولین بار نیست که در مورد خود توجه می‌خواهند، متوجه خواهند شد که ما به هیچ ماتریس  $K$  و  $Q$  یا اعمال ماسک و غیره ارجاع نمی‌دهیم. این به این دلیل است که این چیزها جزئیات پیاده‌سازی هستند که از نحوه آموزش معمول این مدل‌ها ناشی می‌شوند. یک دسته از داده‌ها تغذیه می‌شود و مدل به طور همزمان آموزش می‌یابد تا  $\text{humpty}$  را از  $\text{dumpty}$  و  $\text{sat}$  مربوط می‌شود و بر تفسیر یا حتی خروجی‌های مدل تأثیری ندارد و ما تصمیم گرفته‌ایم هک‌های کارایی آموزش را در اینجا حذف کنیم.

## سافتمنکس

گفتیم که نرم افزار در اولین یادداشت به طور خلاصه مورد بحث قرار گرفت. در اینجا مشکلی وجود دارد که نرم افزار سعی در حل آن دارد: در تفسیر خروجی ما به تعداد گزینه هایی که می خواهیم شبکه از بین آنها انتخاب کند، نورون داریم. و گفتیم که قرار است انتخاب شبکه را به عنوان نورون با بالاترین مقدار تفسیر کنیم. سپس گفتیم که قصد داریم ضرر را به عنوان تفاوت بین مقداری که شبکه ارائه می کند و مقدار ایده آلی که می خواهیم محاسبه کنیم. اما آن مقدار ایده آلی که می خواهیم چیست؟ آن را در مثال برگ / گل روی .۸۰ تغییر کردیم. اما چرا نه ۵ یا ۱۰ یا ۱۰ میلیون؟ هر چه بالاتر باشد برای آن مثال آموزشی بهتر است. در حالت ایده آل می خواهیم بین نهایت آنجا باشد! حالا این مشکل را غیرقابل حل می کند - همه ضررها بین نهایت خواهد بود و برنامه ما برای به حداقل رساندن ضرر با حرکت دادن پارامترها (به یاد داشته باشید "نزول گرادیان") شکست می خورد. چگونه با این مشکل برخورد کنیم؟

یک کار ساده‌ای که می توانیم انجام دهیم این است که مقادیری را که می خواهیم محدود کنیم. مثلاً بین ۰ تا ۱ این همه ضرر را محدود می کند، اما اکنون با این مشکل مواجه هستیم که وقتی شبکه بیش از حد تخمین می زند چه اتفاقی می افتد. فرض کنید در یک مورد (۱،۵) برای (برگ، گل) و در مورد دیگر (۰،۰) خروجی می دهد. مورد اول انتخاب درست را انجام داد اما ضرر بدتر است! خوب، پس حالا به راهی نیاز داریم تا خروجی های لایه آخر را نیز در محدوده (۰،۱) تبدیل کنیم تا ترتیب حفظ شود. می توانیم از هر تابعی (یک "تابع" در ریاضیات به سادگی نگاشت یک عدد به عدد دیگر است - یک عدد وارد می شود، دیگری خارج می شود - بر اساس قاعده در مورد آنچه برای یک ورودی خاص خروجی خواهد بود) در اینجا برای انجام کار استفاده کنیم. یک گزینه ممکن تابع لجستیک است (به نمودار زیر مراجعه کنید) که همه اعداد را به اعداد بین (۰،۱) نگاشت می کند و ترتیب را حفظ می کند:



اکنون، ما برای هر یک از نورون های لایه آخر عددی بین ۰ تا ۱ داریم و می توانیم ضرر را با تنظیم نورون صحیح به ۰، بقیه به ۱، و گرفتن تفاوت آن با آنچه شبکه به ما ارائه می دهد محاسبه کنیم. این کار خواهد کرد، اما می توانیم بهتر عمل کنیم؟

برگردیم به مثال "Humpty dumpty" خود، فرض کنید سعی داریم کاراکتر به کاراکتر "dumpty" را تولید کنیم و مدل ما هنگام پیش‌بینی " $m$ " در dumpty اشتباه می‌کند. به جای اینکه آخرین لایه را با " $m$ " به عنوان بالاترین مقدار به ما بدهد، " $u$ " را به عنوان بالاترین مقدار به ما می‌دهد اما " $m$ " دوم نزدیک است.

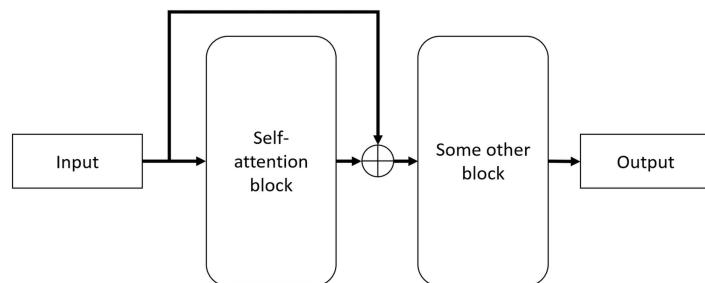
اکنون می‌توانیم با "ادامه دهیم و سعی کنیم کاراکتر بعدی را پیش‌بینی کنیم و غیره، اما اعتقاد مدل کم خواهد بود زیرا ادامه‌های خوبی از "humpty duu.." وجود ندارد. از سوی دیگر، " $m$ " دوم نزدیک بود، بنابراین می‌توانیم به " $m$ " نیز شанс بدیم، چند کاراکتر بعدی را پیش‌بینی کنیم و بینیم چه اتفاقی می‌افتد؟ شاید یک کلمه کلی بهتر به ما بدهد؟

پس آنچه در اینجا در مورد آن صحبت می‌کنیم نه تنها انتخاب کورکورانه حداکثر مقدار است، بلکه تلاش برای چند مورد است. راه خوب برای انجام این کار چیست؟ خوب، باید به هر کدام شناسی اختصاص دهیم - بگوییم که با  $50\%$ ، دومی با  $25\%$  و غیره انتخاب خواهیم کرد. این یک راه خوب برای انجام آن است. اما شاید بخواهیم شانس به پیش‌بینی‌های مدل زیربنایی وابسته باشد. اگر مدل مقادیر  $m$  و  $u$  را در اینجا (در مقایسه با سایر مقادیر) بسیار نزدیک به هم پیش‌بینی کند - پس شاید یک شанс نزدیک  $50\%-50\%$  برای کاوش دو مورد ایده خوبی باشد؟

پس ما به یک قانون خوب نیاز داریم که همه این اعداد را می‌گیرد و آنها را به شانس تبدیل می‌کند. این همان کاری است که softmax انجام می‌دهد. این یک تعمیم تابع لجستیک در بالا است اما با ویژگی‌های اضافی. اگر  $10$  عدد دلخواه به آن بدهید -  $10$  خروجی به شما می‌دهد، هر کدام بین  $0$  تا  $1$  و مهمتر از همه، همه  $10$  تا به  $1$  جمع می‌شوند تا بتوانیم آنها را به عنوان شانس تفسیر کنیم. شما softmax را به عنوان آخرین لایه در تقریباً هر مدل زبانی پیدا خواهید کرد.

## اتصالات باقیمانده

ما به آرامی نحوه تجسم شبکه‌های خود را با پیشرفت بخش‌ها تغییر داده‌ایم. اکنون از جعبه‌ها/بلوک‌ها برای نشان دادن مفاهیم خاصی استفاده می‌کنیم. این نماد برای نشان دادن مفهوم خاص اتصالات باقیمانده مفید است. بیایید به اتصال باقیمانده همراه با یک بلوک خود توجه نگاه کنیم:



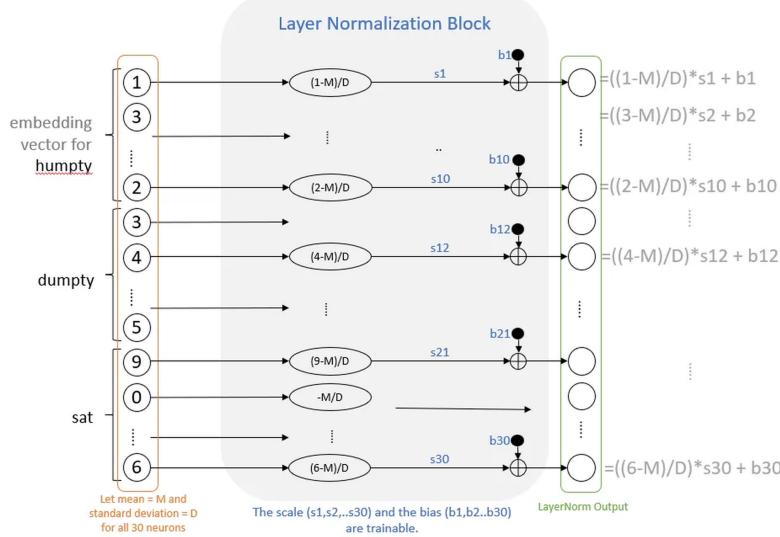
توجه داشته باشید که ما "ورودی" و "خروجی" را به عنوان جعبه قرار داده‌ایم تا کارها ساده‌تر شود، اما اینها اساساً فقط مجموعه‌ای از نورون‌ها/اعداد هستند که مانند بالا نشان داده شده‌اند.

پس اینجا چه خبره؟ اساساً خروجی بلوک خود توجه را می‌گیریم و قبل از انتقال آن به بلوک بعدی، آن را به ورودی اصلی اضافه می‌کنیم. اولین نکته قابل توجه این است که این مستلزم آن است که ابعاد خروجی بلوک خود توجه اکنون باید با ورودی یکسان باشد. این مشکلی نیست زیرا همانطور که اشاره کردیم خروجی خود توجه توسط کاربر تعیین می‌شود. اما چرا این کار را انجام دهیم؟ ما در اینجا به همه جزئیات نمی‌پردازیم اما نکته اصلی این است که با عمیق تر شدن شبکه‌ها (بیشتر لایه‌ها بین ورودی و خروجی) آموزش آن‌ها به طور فراینده‌ای سخت‌تر می‌شود. نشان داده شده است که اتصالات باقیمانده به چالش‌های آموزشی کمک می‌کنند.

## نرمال‌سازی لایه

نرمال‌سازی لایه یک لایه نسبتاً ساده است که داده‌های ورودی به لایه را می‌گیرد و با کم کردن میانگین و تقسیم بر انحراف استاندارد آن را نرمال می‌کند (شاید کمی بیشتر، همانطور که در زیر می‌بینیم). به عنوان مثال، اگر بخواهیم نرمال‌سازی لایه را بلافاصله پس از ورودی اعمال کنیم، تمام نورون‌های لایه ورودی را می‌گیرد و سپس دو آمار را محاسبه می‌کنند: میانگین و انحراف استاندارد آن‌ها. فرض کنید میانگین  $M$  و انحراف استاندارد  $D$  باشد، سپس کاری که نرمال‌سازی لایه انجام می‌دهد این است که هر یک از این نورون‌ها را می‌گیرد و با  $(x - M)/D$  جایگزین می‌کند که در آن  $x$  مقدار اصلی هر نورون را نشان می‌دهد.

حالا این چطور کمک می‌کند؟ اساساً بدراد ورودی را ثابت می‌کند و به آموزش شبکه‌های عمیق کمک می‌کند. یک نگرانی این است که با نرمال‌سازی ورودی‌ها، آیا اطلاعات مفیدی را از آن‌ها حذف می‌کنیم که ممکن است در یادگیری چیزی ارزشمند در مورد هدف ما مفید باشد؟ برای رسیدگی به این موضوع، لایه نرمال‌سازی لایه دارای یک پارامتر مقیاس و یک پارامتر بایاس است. اساساً، برای هر نورون، شما فقط آن را در یک اسکالر ضرب می‌کنید و سپس یک بایاس به آن اضافه می‌کنید. این مقادیر اسکالر و بایاس پارامترهایی هستند که می‌توانند آموزش بینند. این به شبکه اجازه می‌دهد تا برخی از تغییرات را که ممکن است برای پیش‌بینی‌ها ارزشمند باشد، بیاموزد. و از آنجایی که اینها تنها پارامترها هستند، بلوک LayerNorm پارامترهای زیادی برای آموزش ندارد. کل چیزی شبیه به این است:



مقیاس و بایاس پارامترهای قابل آموزش هستند. می توانید بینید که نرمال سازی لایه یک بلوک نسبتاً ساده است که هر عدد فقط به صورت نقطه ای (پس از محاسبه اولیه میانگین و انحراف استاندارد) روی آن عمل می شود. یادآور لایه فعال سازی (مانند **RELU**) است با این تفاوت کلیدی که در اینجا برخی پارامترهای قابل آموزش داریم (هرچند بسیار کمتر از لایه های دیگر به دلیل عملیات نقطه ای ساده).

انحراف استاندارد یک معیار آماری از میزان پراکندگی مقادیر است، به عنوان مثال، اگر همه مقادیر یکسان باشند، می گوییم انحراف استاندارد صفر است. اگر به طور کلی، هر مقدار واقعاً از میانگین این مقادیر بسیار دور باشد، آنگاه انحراف استاندارد بالایی خواهد داشت. فرمول محاسبه انحراف استاندارد برای مجموعه ای از اعداد،  $a_3 \dots a_2 \dots a_1$  (مثلاً  $N$  عدد) تقریباً به این صورت است: میانگین (این اعداد) را از هر یک از اعداد کم کنید، سپس پاسخ هر یک از  $N$  عدد را مربع کنید. همه این اعداد را با هم جمع کنید و سپس بر  $N$  تقسیم کنید. حالا جذر جواب را بگیرید.

توجه برای افراد پیشرفته: متخصصان ML با تجربه متوجه خواهند شد که هیچ بخشی در مورد نرمال سازی دسته وجود ندارد. در واقع، ما حتی مفهوم دسته ها را در این مقاله اصلاً معرفی نکرده ایم. تا حد زیادی، من معتقدم دسته ها شتاب دهنده های آموزشی دیگری هستند که به درک مفاهیم اصلی (به جز شاید نرمال سازی دسته که در اینجا به آن نیازی نداریم) مرتبط نیستند.

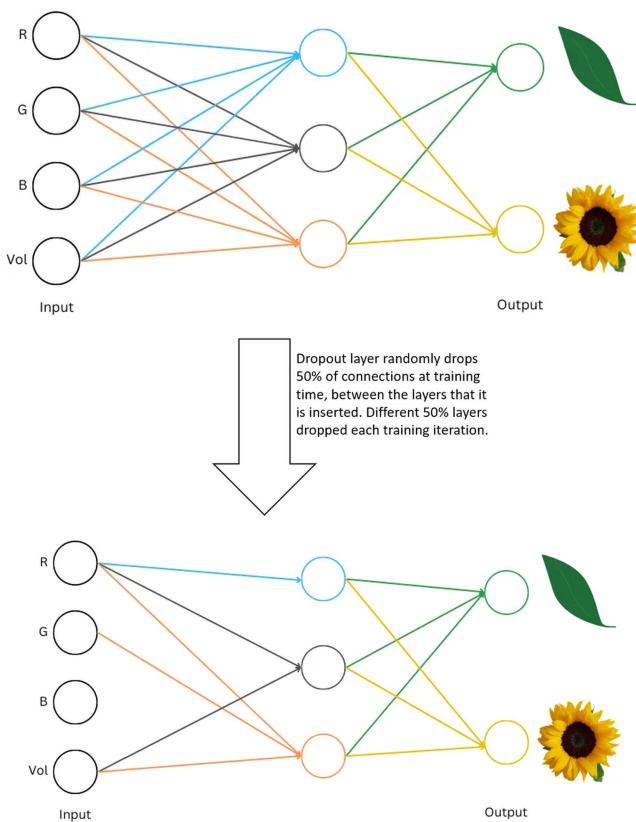
## dropout

یک روش ساده اما موثر برای جلوگیری از بیش برازش مدل است. بیش برازش اصطلاحی است برای زمانی که شما مدل را روی داده های آموزشی خود آموزش می دهید و روی آن مجموعه داده خوب عمل می کنید اما به خوبی به مثال هایی که مدل ندیده است تعیین نمی یابد. تکنیک هایی که

به ما کمک می‌کنند از بیش برآزش جلوگیری کنیم، "تکنیک‌های تنظیم بیش از حد" نامیده می‌شوند و dropout یکی از آنهاست.

اگر یک مدل را آموزش دهید، ممکن است در داده‌ها اشتباه کند و / یا به روشنی خاص بیش از حد برآزش کند. اگر مدل دیگری را آموزش دهید، ممکن است همین کار را انجام دهد، اما به روشنی متفاوت. اگر تعداد زیادی از این مدل‌ها را آموزش دهید و خروجی‌ها را میانگین بگیرید چه؟ اینها معمولاً "مدل‌های گروهی" نامیده می‌شوند زیرا خروجی‌هارا با ترکیب خروجی‌های گروهی از مدل‌ها پیش‌بینی می‌کنند و مدل‌های گروهی معمولاً عملکرد بهتری نسبت به هر یک از مدل‌های فردی دارند. در شبکه‌های عصبی، می‌توانید همین کار را انجام دهید. می‌توانید چندین مدل (کمی متفاوت) بسازید و سپس خروجی‌های آن‌ها را ترکیب کنید تا مدل بهتری داشته باشد. با این حال، این می‌تواند از نظر محاسباتی گران باشد. Dropout تکیکی است که کاملاً مدل‌های گروهی را نمی‌سازد اما برخی از جوهر مفهوم را به دست می‌آورد.

مفهوم ساده است، با درج یک لایه حذف در حین آموزش، کاری که انجام می‌دهید حذف تصادفی درصد مشخصی از اتصالات مستقیم نورون‌ین لایه‌هایی است که حذف درج شده است. با در نظر گرفتن شبکه اولیه ما و درج یک لایه Dropout بین ورودی و لایه میانی با نرخ حذف ۵۰٪ می‌تواند چیزی شبیه به این باشد:



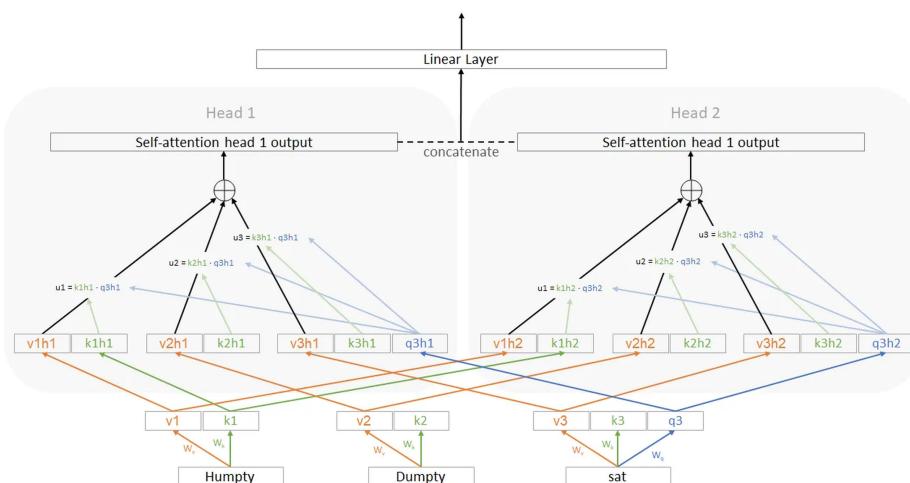
اکنون، این شبکه را مجبور می‌کند با افزونگی زیادی آموزش بینند. اساساً، شما در حال آموزش تعدادی مدل مختلف به طور همزمان هستید - اما آنها وزن‌ها را به اشتراک می‌گذارند.

اکنون برای استنباط، می‌توانیم رویکرد مشابهی را با یک مدل گروهی دنبال کنیم. می‌توانیم چندین پیش‌بینی را با استفاده از حذف انجام دهیم و سپس آن‌ها را ترکیب کنیم. با این حال، از آنجایی که این کار از نظر محاسباتی پرهزینه است - و از آنجایی که مدل‌های ما وزن‌های مشترکی دارند - چرا به جای استفاده از٪ ۵۰ از وزن‌ها در یک زمان، همه آن‌ها را همزمان استفاده نمی‌کنیم. این باید به ما تقریبی از آنچه یک گروه ارائه می‌دهد را بدهد.

یک مشکل این است: مدلی که با٪ ۵۰ از وزن‌ها آموزش دیده است، اعداد بسیار متفاوتی در نورون‌های میانی نسبت به مدلی که از همه وزن‌ها استفاده می‌کند خواهد داشت. آنچه ما می‌خواهیم میانگین‌گیری بیشتر به سبک گروهی در اینجا است. چگونه این کار را انجام دهیم؟ خوب، یک راه ساده این است که به سادگی همه وزن‌ها را در ۰.۵ ضرب کنیم زیرا اکنون از دو برابر وزن استفاده می‌کنیم. این همان کاری است که Dropout در حین استنباط انجام می‌دهد. از شبکه کامل با تمام وزن‌ها استفاده می‌کند و به سادگی وزن‌ها را با  $(1-p)$  ضرب می‌کند که در آن  $p$  احتمال حذف است. و این نشان داده شده است که به عنوان یک تکنیک تنظیم بیش از حد به خوبی عمل می‌کند.

## توجه چند سر

این بلوک کلیدی در معماری ترانسفورماتور است. ما قبلاً دیده‌ایم که یک بلوک توجه چیست. یادتان باشد که خروجی یک بلوک توجه توسط کاربر تعیین می‌شد و طول  $T$ ‌ها بود. آنچه یک سر توجه چندگانه است اساساً چندین سر توجه را به صورت موازی اجرا می‌کنید (همه ورودی‌های یکسانی را می‌گیرند). سپس همه خروجی‌های آن‌ها را می‌گیریم و به سادگی آن‌ها را به هم متصل می‌کنیم. چیزی شبیه به این است:



به خاطر داشته باشید که فلش‌هایی که از  $v1h1$  به  $v1$  می‌روند لایه‌های خطی هستند - یک ماتریس روی هر فلش وجود دارد که تبدیل می‌شود. من فقط آن‌ها را برای جلوگیری از شلوغی نشان ندادم.

اینجا اتفاقی که می‌افتد این است که ما در حال تولید کلید، پرسش و مقادیر یکسانی برای هر یک از سرها هستیم. اما سپس اساساً یک تبدیل خطی را بر روی آن اعمال می‌کنیم (به طور جداگانه برای هر  $k, q, v$  و به طور جداگانه برای هر سر) قبل از استفاده از آن مقادیر  $k, q, v$ . این لایه اضافی در خود توجه وجود نداشت.

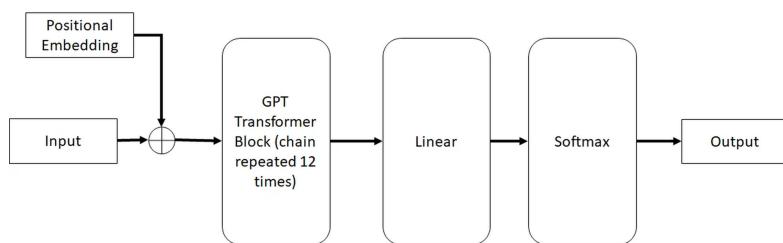
گفتنی است که به نظر من، این یک روش کمی تعجب‌آور برای ایجاد توجه چند سر است. به عنوان مثال، چرا به جای اضافه کردن یک لایه جدید و به اشتراک گذاشتن این وزن‌ها، ماتریس‌های جداگانه‌ای برای هر یک از سرها ایجاد نمی‌کنیم؟ اگر می‌دانید به من اطلاع دهید - من واقعاً نمی‌دانم.

## رمزگذاری و تعییه موقعیتی

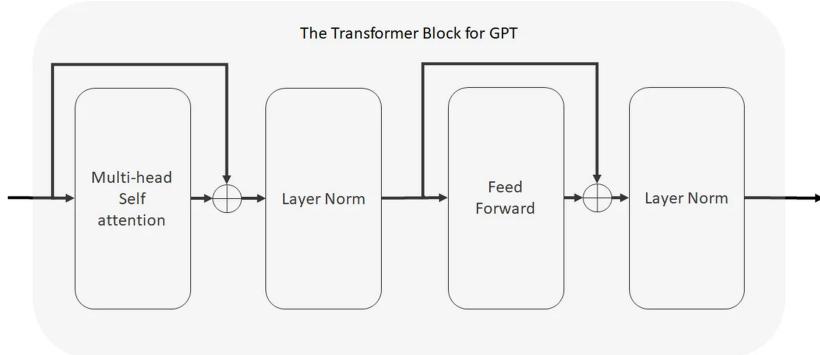
ما به طور خلاصه در مورد انگیزه استفاده از رمزگذاری موقعیتی در بخش خود توجه صحبت کردیم. اینها چیستند؟ در حالی که تصویر رمزگذاری موقعیتی را نشان می‌دهد، استفاده از تعییه موقعیتی رایج‌تر از استفاده از رمزگذاری است. به همین ترتیب، ما در اینجا در مورد یک تعییه موقعیتی مشترک صحبت می‌کنیم، اما ضمیمه همچنین رمزگذاری موقعیتی مورد استفاده در مقاله اصلی را پوشش می‌دهد. یک تعییه موقعیتی با هر تعییه دیگری متفاوت نیست، به جز اینکه به جای تعییه واژگان، اعداد ۱، ۲، ۳ و غیره را تعییه می‌کنیم. بنابراین این تعییه ماتریسی با همان طول تعییه کلمه است و هر ستون مربوط به یک عدد است. این همه ماجراست.

## GPT معماری

بیایید در مورد معماری GPT صحبت کنیم. این چیزی است که در اکثر مدل‌های GPT (با تغییرات) استفاده می‌شود. اگر تا اینجا مقاله را دنبال کرده باشید، در ک این موضوع باید کاملاً ساده باشد. با استفاده از نمادگذاری جعبه‌ای، این چیزی است که معماری در سطح بالا به نظر می‌رسد:



در این مرحله، به جز "بلوک ترانسفورماتور GPT"، همه بلوک‌های دیگر با جزئیات زیادی مورد بحث قرار گرفته‌اند. علامت + در اینجا به سادگی به این معنی است که دو بردار با هم جمع می‌شوند (که به این معنی است که دو تعبیه باید اندازه یکسانی داشته باشند). بیاید به این بلوک ترانسفورماتور GPT نگاه کنیم:



و این تقریباً همه چیز است. این در اینجا "ترانسفورماتور" نامیده می‌شود زیرا از ترانسفورماتور مشتق شده است و نوعی معماری است که در بخش بعدی به آن خواهیم پرداخت. این بر درک تأثیری ندارد زیرا قبل از همه بلوک‌های ساختمانی نشان داده شده در اینجا را پوشش داده‌ایم. بیاید همه چیزهایی را که تاکنون پوشش داده‌ایم برای ساخت این معماری GPT مرور کنیم:

- دیدیم که چگونه شبکه‌های عصبی اعداد را می‌گیرند و اعداد دیگری را خروجی می‌دهند و وزن‌هایی به عنوان پارامتر دارند که می‌توانند آموزش بیینند.
- می‌توانیم تفسیرهایی را به این اعداد ورودی/خروجی متصل کنیم و معنای دنیای واقعی به یک شبکه عصبی بدھیم.
- می‌توانیم شبکه‌های عصبی را زنجیره کنیم تا شبکه‌های بزرگتری ایجاد کنیم و می‌توانیم به هر کدام یک "بلوک" بگوییم و آن را با یک کادر برای ساده‌تر کردن نمودارها نشان دهیم. هر بلوک همچنان کار یکسانی انجام می‌دهد، یعنی یک دسته عدد را می‌گیرد و دسته دیگری از اعداد را خروجی می‌دهد.
- ما انواع مختلفی از بلوک‌ها را یاد گرفتیم که برای اهداف مختلف عمل می‌کنند.
- GPT فقط یک آرایش خاص از این بلوک‌ها است که در بالا نشان داده شده است و تفسیر آن را در قسمت ۱ مورد بحث قرار دادیم.

تغییراتی در طول زمان در این مورد ایجاد شده است زیرا شرکت‌ها به سمت LLM‌های مدرن قدرتمند پیشرفت کرده‌اند، اما اصول اولیه همچنان یکسان است.

اکنون، این ترانسفورماتور GPT در واقع همان چیزی است که در مقاله اصلی ترانسفورماتور که معماری ترانسفورماتور را معرفی کرد، "رمزگشا" نامیده می‌شود. بیایید به آن نگاهی بیندازیم.

## معماری ترانسفورماتور

این یکی از نوآوری‌های کلیدی است که اخیراً شتاب سریعی در قابلیت‌های مدل‌های زبانی ایجاد کرده است. ترانسفورماتورها نه تنها دقت پیش‌بینی را بهبود بخشیدند، بلکه آموزش آن‌ها نیز آسان‌تر / کارآمدتر از مدل‌های قبلی است و به مدل‌های بزرگ‌تر اجازه می‌دهد. این همان چیزی است که معماری GPT بالا بر اساس آن است.

اگر به معماری GPT نگاه کنید، می‌بینید که برای تولید کلمه بعدی در دنباله عالی است. اساساً منطق مشابهی را که در قسمت ۱ مورد بحث قرار دادیم دنبال می‌کند. با چند کلمه شروع کنید و سپس به صورت تک تک تولید کنید. اما اگر می‌خواستید ترجمه انجام دهید چه می‌شد؟ اگر جمله‌ای به زبان آلمانی داشتید (مثلًاً "Wo wohnst du" = "Where do you live") و می‌خواستید آن را به انگلیسی ترجمه کنید. چگونه مدل را برای انجام این کار آموزش می‌دادیم؟

خوب، اولین کاری که باید انجام دهیم این است که راهی برای وارد کردن کلمات آلمانی پیدا کنیم. یعنی باید تعییه خود را برای شامل کردن هم آلمانی و هم انگلیسی گسترش دهیم. حالا فکر می‌کنم اینجا یک راه ساده برای وارد کردن اطلاعات وجود دارد. چرا جمله آلمانی را در ابتدای هر چیزی که تاکنون به انگلیسی تولید شده است اضافه نمی‌کنیم و آن را به زمینه تغذیه نمی‌کنیم؟ برای اینکه کار برای مدل آسان‌تر شود، می‌توانیم یک جداکننده اضافه کنیم. این در هر مرحله به این شکل خواهد بود:

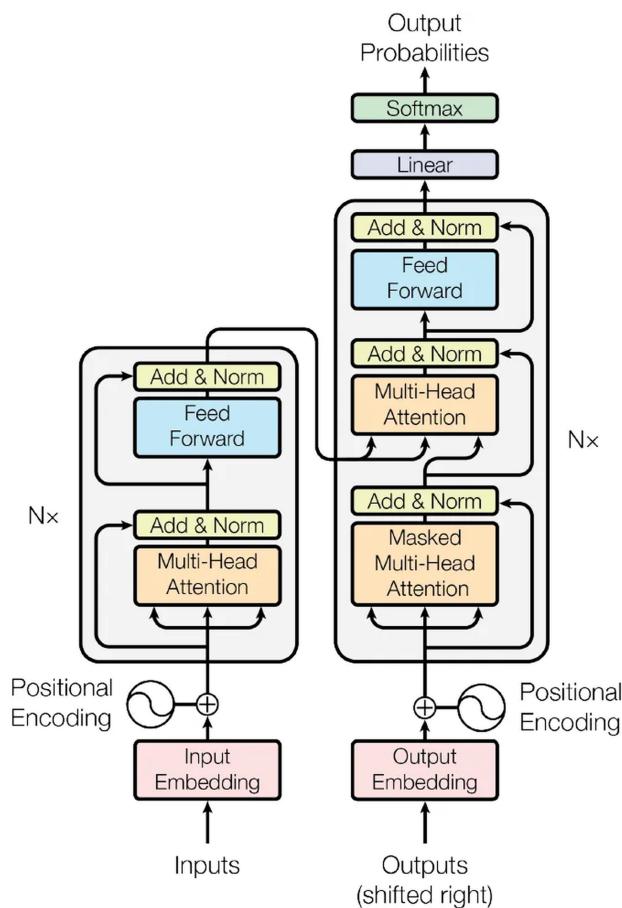
|        |    |        |        |        |       |       |       |   |       |
|--------|----|--------|--------|--------|-------|-------|-------|---|-------|
|        |    |        | Wo     | wohnst | du    | ?     | <SEP> | → | Where |
|        |    | Wo     | wohnst | du     | ?     | <SEP> | Where | → | do    |
|        |    | Wo     | wohnst | du     | ?     | <SEP> | Where | → | you   |
|        | Wo | wohnst | du     | ?      | <SEP> | Where | do    | → | live  |
| wohnst | du | ?      | <SEP>  | Where  | do    | you   | live  | → | ?     |
|        |    |        |        |        |       |       |       | → | <END> |

این کار خواهد کرد، اما جای بهبود دارد:

- اگر طول متن ثابت باشد، گاهی اوقات جمله اصلی از بین می‌رود
- مدل چیزهای زیادی برای یادگیری دارد. دو زبان به طور همزمان، اما همچنین باید بداند که <SEP> توکن جداکننده‌ای است که در آن باید شروع به ترجمه کند

- شما کل جمله آلمانی را با جابجایی‌های مختلف، برای هر تولید کلمه پردازش می‌کنید. این بدان معناست که نمایشن‌های داخلی متفاوتی از یک چیز وجود خواهد داشت و مدل باید بتواند برای ترجمه از همه آن‌ها عبور کند.

در اصل برای این کار ایجاد شد و از یک "رمزگذار" و یک "رمزگشا" تشکیل شده است - که اساساً دو بلوک جداگانه هستند. یک بلوک به سادگی جمله آلمانی را می‌گیرد و یک نمایش میانی (باز هم، دسته‌هایی از اعداد، اساساً) می‌دهد - این رمزگذار نامیده می‌شود. بلوک دوم کلمات را تولید می‌کند (تاکنون زیاد در مورد آن صحبت کرده‌ایم). تنها تفاوت این است که علاوه بر تغذیه آن با کلماتی که تاکنون تولید شده‌اند، جمله آلمانی رمزگذاری شده (از بلوک رمزگذار) را نیز به آن تغذیه می‌کنیم. بنابراین همانطور که زبان را تولید می‌کند، زمینه آن اساساً تمام کلمات تولید شده تاکنون به علاوه آلمانی است. این بلوک رمزگشا نامیده می‌شود. هر یک از این رمزگذارها و رمزگشاها از چند بلوک تشکیل شده‌اند، به ویژه بلوک توجه که بین لایه‌های دیگر قرار گرفته است. بیایید به تصویر یک ترانسفورماتور از مقاله "توجه همه چیزی است که نیاز دارید" نگاه کنیم و سعی کنیم آن را درک کنیم:



مجموعه عمودی بلوک‌ها در سمت چپ "رمزگذار" و بلوک‌های سمت راست "رمزگشا" نامیده می‌شود. باید مرور کنیم و هر چیزی را که قبل از پوشش نداده‌ایم در ک کنیم:

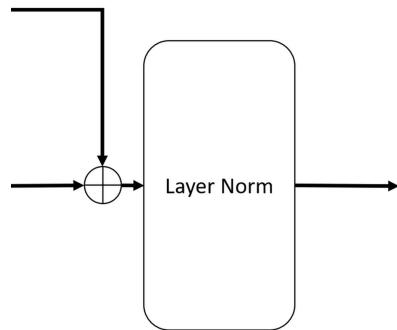
مرور نحوه خواندن نمودار: هر یک از این جعبه‌ها یک بلوک است که مجموعه‌ای از ورودی‌ها را به شکل نورون می‌گیرد و مجموعه‌ای از نورون‌ها را به عنوان خروجی خارج می‌کند که سپس می‌تواند توسط بلوک بعدی پردازش شود یا توسط ما تفسیر شود. فلش‌ها نشان می‌دهند که خروجی یک بلوک به کجا می‌رود. همانطور که می‌بینید، اغلب خروجی یک بلوک را می‌گیریم و آن را به عنوان ورودی به چندین بلوک تغذیه می‌کنیم. باید هر چیزی را که در اینجا وجود دارد مرور کنیم:

: یک شبکه فیدفوروارد شبکه‌ای است که حاوی چرخه نیست. شبکه اصلی ما در بخش ۱ یک فیدفوروارد است. در واقع، این بلوک از ساختار بسیار مشابه استفاده می‌کند. این شامل دو لایه خطی است که هر کدام به دنبال یک RELU (به یادداشت در مورد RELU در بخش اول مراجعه کنید) و یک لایه حذف هستند. به خاطر داشته باشید که این شبکه فیدفوروارد برای هر موقعیت به طور مستقل اعمال می‌شود. این بدان معناست که اطلاعات در موقعیت ۰، دارای یک شبکه فیدفوروارد است و در موقعیت ۱ یکی دارد و غیره. اما نورون‌های موقعیت  $x$  هیچ ارتباطی با شبکه فیدفوروارد موقعیت  $x$  ندارند. این مهم است زیرا اگر این کار را نمی‌کردیم، به شبکه اجازه می‌دادیم در زمان آموزش با نگاه کردن به جلو تقلب کند.

توجه متقاطع: متوجه خواهید شد که رمزگشا دارای یک توجه چند سر با فلش‌هایی است که از رمزگذار می‌آید. اینجا چه خبره؟ مقدار، کلید و پرسش در خود توجه و توجه چند سر را به خاطر دارید؟ همه آنها از یک دنباله آمده بودند. در واقع پرس و جو فقط از آخرین کلمه دنباله بود. پس اگر پرس و جو را نگه داریم اما مقدار و کلید را از یک دنباله کاملاً متفاوت بگیریم چه؟ این همان چیزی است که در اینجا اتفاق می‌افتد. مقدار و کلید از خروجی رمزگذار می‌آیند. از نظر ریاضی چیزی تغییر نکرده است، جز اینکه ورودی‌های کلید و مقدار اکنون از کجا می‌آیند.

$Nx$  به سادگی نشان می‌دهد که این بلوک  $N$  بار به صورت زنجیره‌ای تکرار می‌شود. بنابراین اساساً شما بلوک را پشت سر هم می‌چینید و ورودی را از بلوک قبلی به بعدی منتقل می‌کنید. این راهی برای عمیق‌تر کردن شبکه عصبی است. اکنون، با نگاهی به نمودار، در مورد نحوه تغذیه خروجی رمزگذار به رمزگشا جای برای سردرگمی وجود دارد. فرض کنید  $N=5$ . آیا خروجی هر لایه رمزگذار را به لایه رمزگذار متناظر تغذیه می‌کنیم؟ خیر. اساساً شما رمزگذار را یک بار و فقط یک بار اجرا می‌کنید. سپس فقط آن نمایش را می‌گیرید و همان چیز را به هر یک از ۵ لایه رمزگشا تغذیه می‌کنید.

بلوک Add & Norm: این اساساً همان چیزی است که در زیر آمده است (احتمالاً نویسنده‌گان سعی کرده‌اند در فضای صرف‌جویی کنند).



همه چیز دیگری قبلًا مورد بحث قرار گرفته است. اکنون شما توضیح کاملی در مورد معماری ترانسفورماتور دارید که از عملیات ساده جمع و ضرب ساخته شده و کاملاً مستقل است! شما می‌دانید هر خط، هر جمع، هر کادر و هر کلمه به چه معناست از نظر نحوه ساخت آن‌ها از ابتدا. از نظر تئوری، این یادداشت‌ها حاوی آنچه شما برای کدنویسی ترانسفورماتور از ابتدا نیاز دارید، است.