سوال پنجم

پاسخ یک)

تابع خودتوجه در مدل ترنسفورمر را می توان به عنوان یک شبکه کاملاً متصل در قالب یک ماتریس نشان داد. بیایید ابتدا مکانیسم خودتوجه را بشکنیم:

با توجه به یک دنباله ورودی از بردارهای کلمه $X = \{x_1, x_2, \ldots, x_N\}$ که N طول دنباله و D ابعاد هر بردار کلمه است. مکانیسم خودتوجه وزن توجه را برای هر کلمه در دنباله بر اساس روابط بین همه کلمات در دنباله محاسبه می کند.

عملیات خودتوجهی را می توان به صورت زیر فرموله کرد:

$$Attn(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

برای نمایش این عملیات به عنوان یک شبکه کاملاً متصل در قالب یک ماتریس، می‌توانیم کلمه بردارهای x_i را به هم متصل کنیم تا یک ماتریس X' به شکل $N \times D$ تشکیل دهیم که در آن هر ردیف یک بردار کلمه را نشان می دهد. سپس، می‌توانیم ماتریس توجه A را با ضرب X' در transpose آن محاسبه کنیم:

$$A = X'(X')^T$$

این ضرب ماتریس روابط بین تمام بردارهای کلمه در دنباله ورودی را نشان می دهد. در نهایت، برای نگاشت توالی کامل ورودی در یک بردار خروجی با همان بعد، می‌توانیم ضرب ماتریس دیگری را بین A و یک ماتریس وزن قابل یادگیری W بدست آوریم:

$$Output = softmax(AW)$$

پاسخ دو)

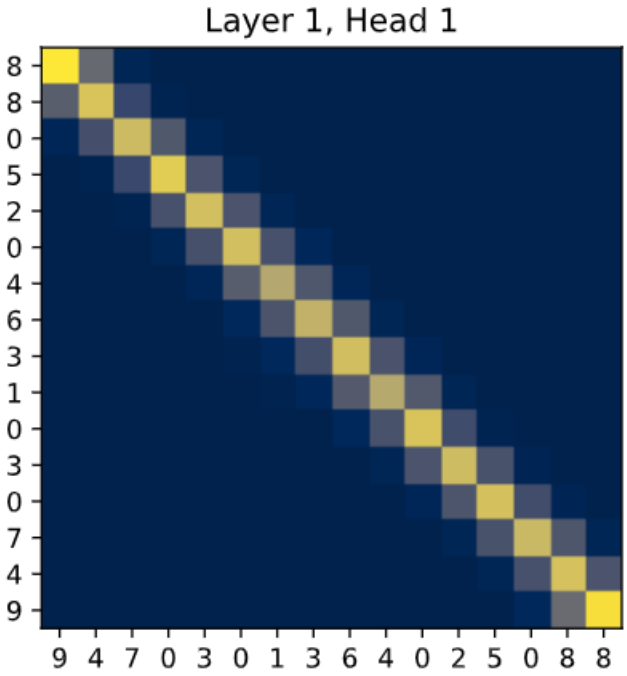
که در آن W به شکل $N \times N$ است (با فرض اینکه بخواهیم ابعاد یکسانی را حفظ کنیم) و شامل پارامترهای $O(N^2)$ است.

این نمایش شبکه کاملاً متصل در واقع دارای پارامترهای $O(N^2D^2)$ خواهد بود، همانطور که ورودی های N^2 در ماتریس توجه A داریم و هر درایه آن محصول ضرب بردارهای D بعدی است.

پاسخ سه)

در مکانیسم خودتوجه، هر کلمه در دنباله ورودی به همه کلمات دیگر توجه می کند، اما همه توجه ها به یک اندازه مهم نیستند. برای دستیابی به این هدف، وزن توجه با استفاده از یک تابع softmax محاسبه می‌شود، که منجر به توزیع توجه پراکنده می‌شود که در آن فقط چند کلمه توجه قابل توجهی را دریافت می‌کنند.

هر موقعیت در دنباله به همه موقعیت ها توجه می کند، اما قدرت توجه توسط ضرب نقطه ای تعیین می شود، که منجر به یک ماتریس پراکنده می شود که در آن بیشتر وزن های توجه نزدیک به صفر هستند. این پراکندگی از طریق تابع softmax به دست می‌آید که بر مرتبط‌ترین کلمات تأکید می‌کند و در عین حال کلمات کمتر مرتبط را سرکوب می‌کند.



در نمودار بالا:

- بلوک‌های سایه‌دار وزن‌های توجه غیر صفر را نشان می‌دهند، که نشان‌دهنده اهمیت توجه است.

- بلوک های مورب نشان دهنده توجه به یک کلمه (توجه به خود) هستند، که در آن وزن توجه ممکن است به دلیل اهمیت توجه به یک کلمه صفر نباشد.

- بلوک های خارج از مورب نشان دهنده توجه به کلمات دیگر در دنباله هستند، جایی که بیشتر وزن های توجه به دلیل پراکندگی نزدیک به صفر هستند.

این ساختار پراکنده با به اشتراک گذاری پارامترها به مکانیسم خودتوجه اجازه می دهد تا وابستگی های بین کلمات را در توالی ورودی به طور موثر ثبت کند و در عین حال قابلیت پردازش محاسباتی را حفظ کند.

پاسخ چهار)

برای درک اینکه چرا خروجی های یک لایه توجه چند سر با توجه به ترتیب مجدد ترتیب ورودی در صورت حذف کدگذاری موقعیتی معادل هستند، اجزای کلیدی درگیر را تجزیه می کنیم:

1. لایه توجه چند سر: این لایه چندین سر توجه را به صورت موازی محاسبه می‌کند و به مدل اجازه می‌دهد تا روی بخش‌های مختلف دنباله ورودی به طور همزمان تمرکز کند.

2. رمزگذاری موقعیتی: در مدل ترنسفورمر، رمزگذاری موقعیتی به جاسازی های ورودی اضافه می شود تا اطلاعاتی در مورد موقعیت هر کلمه در دنباله ارائه شود. این به مدل کمک می کند تا بین کلمات در موقعیت های مختلف تفاوت قائل شود.

هنگامی که رمزگذاری موقعیتی حذف می شود، مدل اطلاعات صریحی در مورد موقعیت هر کلمه در دنباله ندارد. در عوض، برای انجام محاسبات توجه صرفاً بر محتوای کلمات متکی است. در نتیجه، مکانیسم توجه مستقل از موقعیت می شود، به این معنی که با همه کلمات بدون توجه به موقعیت آنها در دنباله به طور یکسان برخورد می کند.

حال، خروجی لایه توجه چند سر را در نظر بگیریم که با $Y(X)$ مشخص می‌شود، که در آن X دنباله ورودی و Y دنباله خروجی است. خروجی به صورت زیر محاسبه می شود:

$$Y(X) = Concat[H_1; \dots; H_H].W(o)$$

جایی که:

- $H_1; \dots; H_H$ خروجی های سرهای توجه فردی هستند.

- $W(o)$ ماتریس وزن برای تبدیل خروجی است.

از آنجایی که مکانیسم توجه بدون رمزگذاری موقعیتی مستقل از موقعیت است، خروجی هر سر توجه H_i صرفاً بر اساس محتوای کلمات است نه موقعیت آنها. بنابراین، برای هر جایگشت σ دنباله ورودی، خروجی سرهای توجه ثابت می ماند. در نتیجه، هنگام به هم پیوستن خروجی‌های سرهای توجه و اعمال تبدیل خروجی $W(o)$، دنباله خروجی حاصل $Y(X)$ معادل هر مرتب‌سازی مجدد دنباله ورودی خواهد بود. این به این دلیل است که مدل یاد گرفته است بدون در نظر گرفتن موقعیت مطلق کلمات، به اطلاعات مربوطه در توالی ورودی توجه کند. به این ترتیب، حذف کدگذاری موقعیتی منجر به مکانیزم توجه مستقل از موقعیت می‌شود که منجر به خروجی‌های معادل با توجه به ترتیب مجدد توالی ورودی می‌شود.

سوال چهارم

پاسخ یک)

تفاوت بین رمزگذاری موقعیتی مطلق و رمزگذاری موقعیتی نسبی در این است که رمزگذاری موقعیتی نسبی اطلاعات بین موقعیت های زوجی را در نظر می گیرد، در حالی که رمزگذاری موقعیتی مطلق اینطور نیست. در حالی که رمزگذاری موقعیتی مطلق اطلاعات موقعیتی یک کلمه را ضبط می کند، اطلاعات موقعیتی را برای کل جمله (یا دنباله) نمی گیرد. رمزگذاری موقعیتی نسبی نیز از نظر محاسباتی ناکارآمد و در زمان استنتاج یا همان inference سریع نیست (اطلاعات بیشتر در لینک).

تفاوت بین رمزگذاری موقعیتی یادگیری شونده و رمزگذاری موقعیتی ثابت در این است که در رمزگذاری موقعیتی یادگیری شونده نوع جاسازی، مکان و موقعیت بر اساس داده‌های ورودی تعیین می‌شود ولی در رمزگذاری موقعیتی ثابت، مکان یا موقعیت مستقل از داده‌های ورودی است و به صورت ثابت تعیین می‌شود. رمزگذاری موقعیتی ثابت از داده ها بهره ای نمی برد ولی رمزگذاری موقعیتی یادگیری شونده وابسته به محدودیت داده ممکن است خوب یادگرفته نشود (اطلاعات بیشتر در لینک).

پاسخ دوم)

رمزگذاری موقعیتی چرخشی (RoPE) اطلاعات موقعیتی مطلق را با یک ماتریس چرخشی رمزگذاری می کند و به طور طبیعی وابستگی موقعیت نسبی صریح را در فرمول توجه خود گنجانده است. حال، برای توضیح اینکه چرا بر معایب غلبه می کند، اجازه دهید معایب هر دو روش را مرور کنیم:

رمزگذاری موقعیتی مطلق: اطلاعات موقعیتی نسبی را شامل نمی شود.

رمزگذاری موقعیت نسبی: محاسباتی ناکارآمد، برای استنتاج مناسب نیست.

بنابراین می دانیم که رمزگذاری موقعیتی چرخشی باید ویژگی های زیر را داشته باشد:

مورد 1: شامل اطلاعات موقعیتی نسبی است.

مورد 2: از نظر محاسباتی کارآمد باشد.

مورد 3: برای استنتاج مناسب باشد.

جهت مورد 1: رمزگذاری موقعیتی چرخشی شامل اطلاعات موقعیتی نسبی است که زوایای ماتریس چرخش را به موقعیت کلمه فعلی وابسته می‌کند و با ویژگی‌های ماتریس چرخش، به ما می‌گوید که دو کلمه چقدر از هم فاصله دارند.

جهت مورد 2: ممکن است به نظر برسد که استفاده از رمزگذاری موقعیتی چرخشی از نظر محاسباتی کارآمد نیست زیرا ما باید ماتریس چرخش را ایجاد کنیم، اما محققان آن یک فرمول محاسباتی کارآمد پیدا کرده اند که از نظر محاسباتی کارآمد است که در معادله 34 مقاله ارائه شده است.

جهت مورد 3: از آنجایی که ما قبلاً می دانیم که رمزگذاری موقعیتی چرخشی مانند رمزگذاری موقعیتی مطلق است زیرا رمزگذاری ها فقط به موقعیت فعلی کلمه بستگی دارند، رمزگذاری های موقعیتی برای کلماتی که قبلاً تولید شده بودند تغییر نمی کنند و باعث می شود حافظه پنهان (KV cache) دوباره در طول استنتاج امکان پذیر شود.

(اطلاعات بیشتر همراه با مثال در لینک)

پاسخ سوم:

همانطور که میدانیم بردار امبدینگ توسط ماتریس چرخش به بردار امبدینگ دیگری تبدیل میشود. ماتریس چرخش نباید خود امبدینگ را تغییر دهد زیرا بردار امبدینگ تاثیر گرفته از امبدینگ توکن هاست و نباید بر اساس جایگاه آنها دچار تغییر شود. حال

آنکه جایگاه توکن ها روی زاویه تتا تاثیرگذار بوده و به این ترتیب رمزگذاری موقعیتی دچار تغییر میشود. ایده اساسی استفاده از ماتریس چرخش دو بعدی D//2 برای چرخاندن یک ماتریس D-بعدی است که در آن زاویه چرخش هر ماتریس چرخش دو بعدی با m * Thetha تعیین می شود (که m موقعیت کلمه، و تتا، یک ترم از پیش محاسبه شده است که در همه کلمات به اشتراک گذاشته می شود)، و یک منحنی نمایی تولید می کند. هنگامی که m افزایش می یابد، منحنی نمایی به سمت بالا به مقدار بزرگتر تغییر می کند تا فاصله موقعیتی بالاتر را نشان دهد. و از آنجایی که تتا وابسته به ایندکس جفت کلمات است، پس هر جفت کلمه به مقدار متفاوتی دچار چرخش میشود. پس ویژگی درونی ماتریس چرخش شامل حفظ طول بردار اصلی و تغییر زاویه با محور افقی از نظر مفهومی خود را اینگونه نشان میدهد که بردار توکن امبدینگ را حفظ کرده، ولی چرخش آن باعث تغییر در بردار رمزگذاری موقعیتی آنها میشود.

پاسخ چهارم:

به طور خلاصه، در حالی که جاگذاری‌های موقعیتی سینوسی و کسینوسی اطلاعات موقعیتی ثابتی را بر اساس شاخص‌های موقعیت ارائه می‌کنند، RoPE اطلاعات موقعیتی نسبی را بین نشانه‌ها معرفی می‌کند و به مدل اجازه می‌دهد تا نمایش‌های تطبیقی بیشتری از موقعیت‌های دنباله را بیاموزد. این به طور بالقوه می تواند منجر به بهبود عملکرد در وظایفی شود که نیاز به درک روابط موقعیتی در توالی دارند.

برای انکودینگ سینوسی کسینوسی عادی داریم:

$$P(k, 2i) = \sin\left(\frac{k}{n^{2i/d}}\right)$$

$$P(k, 2i+1) = \cos\left(\frac{k}{n^{2i/d}}\right)$$

که در آن:

ترم k موقعیت یک شی در دنباله ورودی $0 \leq k < L/2$

ترم d ابعاد فضای تعبیه خروجی فرض برابر 4

ترم n یک ترم از پیش تعیین شده برابر 100

ترم i برای مپ کردن $0 \leq i < d/2$

sequence	index	$i = 0$	$i = 0$	$i = 1$	$i = 1$
دانشگاه	0	$\sin(0) = 0$	$\cos(0) = 1$	$\sin(0) = 0$	$\cos(0) = 1$
صنعتی	1	$\sin(1/1) = 0.84$	$\cos(1/1) = 0.54$	$\sin(1/10) = 0.10$	$\cos(1/10) = 1.0$

شریف	2	$\sin(2/1) = 0.91$	$\cos(2/1) = -0.42$	$\sin(2/10) = 0.20$	$\cos(2/10) = 0.98$

که در نهایت داریم:

$$output\ of\ positional\ embedding\ layer = word\ embedding + positional\ encoding\ matrix$$

اما برای انکودینگ موقعیت بوسیله RoPE داریم:

$$\theta_i = 100^{\frac{2(i-1)}{d}}$$

با فرض $d = 4$ داریم:

$$R^d_{\Theta,m} = \begin{bmatrix} \cos m\theta_1 & -\sin m\theta_1 & 0 & 0 \\ \sin m\theta_1 & \cos m\theta_1 & 0 & 0 \\ 0 & 0 & \cos m\theta_2 & -\sin m\theta_2 \\ 0 & 0 & \sin m\theta_2 & \cos m\theta_2 \end{bmatrix}$$

پس به ازای ***m=0,1,2*** داریم:

$$R^d_{\Theta,0} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R^d_{\Theta,0} = \begin{bmatrix} 0.54 & -0.84 & 0 & 0 \\ 0.84 & 0.54 & 0 & 0 \\ 0 & 0 & -0.83 & 0.54 \\ 0 & 0 & -0.54 & -0.83 \end{bmatrix}$$

$$R^d_{\Theta,0} = \begin{bmatrix} -0.41 & -0.9 & 0 & 0 \\ 0.9 & -0.41 & 0 & 0 \\ 0 & 0 & 0.4 & -0.91 \\ 0 & 0 & 0.91 & 0.4 \end{bmatrix}$$

که در نهایت طبق مقاله داریم:

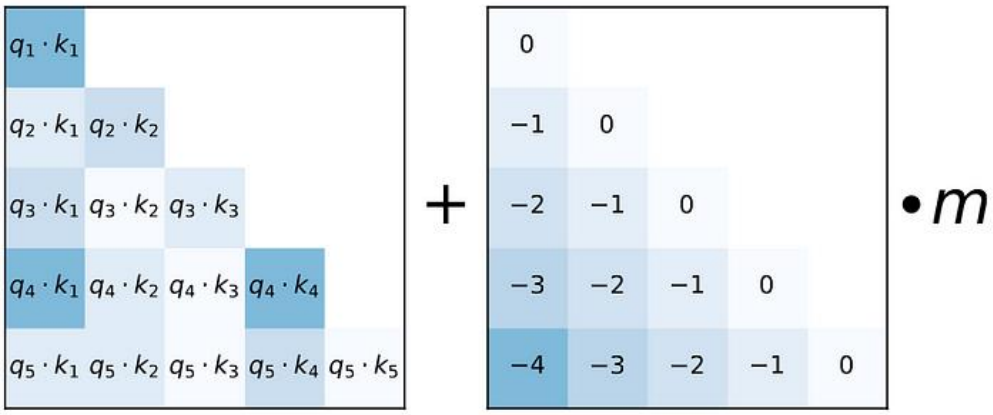
$$f_{\{q,k\}}(x_m, m) = R^d_{\Theta,m} W_{\{q,k\}} x_m$$

تفاوت: در رمزگذاری موقعیتی سینوسی، رمزگذاری‌های موقعیتی معمولاً به جاسازی‌های ورودی اضافه می‌شوند (که سپس به بردارهای ***Query***، ***Key*** و ***Value*** تبدیل می‌شوند) قبل از اینکه به مکانیسم توجه به خود وارد شوند.

اما همانطور که میبینید در ***RoPE*** ماتریس انکودینگ موقعیت دیگر با امبدینگ کلمات جمع نمیشود بلکه در خروجی ضرب امبدینگ کلمات در وزن های ***query*** و ***key*** (و نه ***value***)، ضرب میشود.

پاسخ پنجم:

طبق فرمول ارایه شده در شکل زیر شمایی از عملکرد ALiBi میبینید.



همانطور که میبینید تفاوت میان ایندکس key و query (ضربدر m) میتواند دربردارنده موقعیت نسبی میان توکن ها باشد. می‌توانیم ببینیم که هر چه کلید داده شده در گذشته بیشتر باشد - مقدار بالاتری از مقدار توجه کم می‌شود (اعداد در ماتریس سمت چپ مقادیر توجه هستند). اگر اعداد در ماتریس سمت چپ زیاد باشند، به این معنی است که کلید واقعاً با پرس و جو مرتبط است. هر مقداری که محاسبه می کنید، هر چقدر هم که مهم باشد، هر چه در گذشته بیشتر باشد، بیشتر از آن کم می کنیم و این کار را به صورت خطی انجام می دهیم. بنابراین به صورت خطی تنزل می یابد (از این رو نام آن توجه با بایاس خطی ALiBi است) و می تواند به یک مقدار منفی برای dropoff برود. سپس softmax را به محصولات نقطه query-key اعمال می کنیم که به ما توزیع می دهد.

نقش m: یک عدد از پیش تعریف شده ثابت است، به عنوان مثال، 0.4. اعداد موقعیت q*k می توانند خیلی سریع بزرگ شوند، بنابراین m آنها را عادی سازی می کند زیرا شناور بین 0-1 است. این ترم برای هر سر متفاوت است تا شیب سر توجه را کنترل کند و هر سر را کمی متفاوت از یکدیگر کند تا مدل فقط به نویز متکی نباشد و یک مجموعه بسازد. این می تواند با ایجاد تفاوت اندکی در نحوه کار سرها موثرتر باشد تا مدل بتواند انتخاب کند که از کدام یک بیشتر استفاده کند.

سوال دوم

پاسخ یک)

Shapes

In Encoder:

Query = Key = Value = (Batch, Sequence Length, Embedding) = (32, 2048, 1024)

Output of Attention = Input of Add_Norm 1 = (Batch, Sequence Length, hidden) = (32, 2048, 1024)

Output of Add_Norm 1 = Input of FF = (32, 2048, 1024)

Output of FF = Input of Add_Norm 2 = (32, 2048, 1024), where in the FF, we go to (32, 2048, 1024/2) in first layer and then get back to (32, 2048, 1024) in second layer.

Output of Encoder = Output of Add_Norm 2 = (32, 2048, 1024)

In Decoder:

Query = Key = Value = (Batch, Sequence Length, Embedding) = (32, 2048, 1024)

Output of Attention 1 = Input of Add_Norm 1 = (Batch, Sequence Length, hidden) = (32, 2048, 1024)

Output of Add_Norm 1 = Input of Attention 2 (Query per head) = (32, 2048, 1024)

Output of Attention 2 = Input of Add_Norm 2 = (32, 2048, 1024)

Output of Add_Norm 2 = Input of FF = (32, 2048, 1024)

Output of FF = Input of Add_Norm 3 = (32, 2048, 1024), where in the FF, we go to (32, 2048, 1024/2) in first layer and then get back to (32, 2048, 1024) in second layer.

Output of Decoder = Output of Add_Norm 2 = (32, 2048, 1024)

پاسخ دوم)

Num parameters

In Encoder

Attention: (768*1024+1024) + (1024*768/4+768/4) * 3 * 4 = 787456 + 2361600 = 3149056

Add_Norm 1: 1024*2 = 2048

FF: (1024*1024/2+1024/2) + (1024/2*1024+1024) = 524800 + 525312 = 1050112

Add_Norm 2: 1024*2 = 2048

Total Encoder = 12 * 4203264 = 50439168

In Decoder

Attention 1: (768*1024+1024) + (1024*768/4+768/4) * 3 * 4 = 787456 + 2361600 = 3149056

Add_Norm 1: 1024*2 = 2048

Attention 2: (768*1024+1024) + (1024*768/4+768/4) * 3 * 4 = 787456 + 2361600 = 3149056

Add_Norm 2: 1024*2 = 2048

FF: (1024*1024/2+1024/2) + (1024/2*1024+1024) = 524800 + 525312 = 1050112

Add_Norm 3: 1024*2 = 2048

Total Decoder = 8 * 7354368 = 58834944

Total Params = Total Encoder + Total Decoder = 50439168 + 50439168 = 109274112