نام و نام خانوادگی:

رضا قربانی پاجی

نام همفکران:

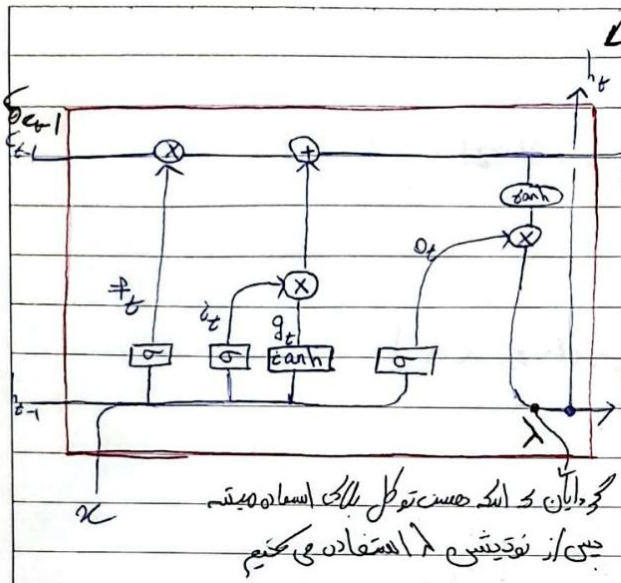
حسین شاه‌آبادی- عرفان یگانگی

شماره دانشجویی:

403206565

تمرین سوم درس یادگیری ژرف

سوال 1



$$i_t = \sigma\left(W_i \begin{bmatrix} h_{t-1} \\ x_t \end{bmatrix} + b_i\right)$$

$$f_t = \sigma\left(W_f \begin{bmatrix} h_{t-1} \\ x_t \end{bmatrix} + b_f\right)$$

$$o_t = \sigma\left(W_o \begin{bmatrix} h_{t-1} \\ x_t \end{bmatrix} + b_o\right)$$

$$g_t = \tanh\left(W_g \begin{bmatrix} h_{t-1} \\ x_t \end{bmatrix} + b_g\right)$$

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

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

Scanned with CamScanner

subject : date :

$$\frac{\partial \mathcal{L}}{\partial h_t} = \frac{\partial \mathcal{L}_t}{\partial h_t} + \delta_{h_t}$$

$$\frac{\partial \mathcal{L}}{\partial c_t} = \frac{\partial \mathcal{L}}{\partial h_t} \times \frac{\partial h_t}{\partial c_t} + \delta_{c_t} = \Phi$$

از اینجایی که در مرحله آپدیت‌های بعدی پارامترها از این استفاده میشه از این مقدارهای استفاده می‌کنیم

$$\frac{\partial \mathcal{L}}{\partial h_t} \odot ((1 - \tanh^2(c_t)) \odot o_t) + \delta_c = \Phi$$

$$\delta_{c_{t-1}} = \frac{\partial \mathcal{L}}{\partial c_t} \odot \frac{\partial c_t}{\partial c_{t-1}} \longrightarrow$$ گرادیان به سلول قبلی برون

$$= \Phi \odot f_t$$

$$\frac{\partial \mathcal{L}}{\partial f_t} = \Phi \times \frac{\partial c_t}{\partial f_t} = \Phi \odot c_{t-1}$$

$$\frac{\partial \mathcal{L}}{\partial w_f} = \sum_{t=1}^{T} \Phi \times \frac{\partial c_t}{\partial f_t} \times \frac{\partial f_t}{\partial w_f} = \sum \Phi \odot c_{t-1} \odot f_t(1-f_t) \otimes \underbrace{\begin{bmatrix} h_{t-1} \\ x_t \end{bmatrix}^T}_{\hat{x}^T}$$

$$\frac{\partial \mathcal{L}}{\partial b_f} = \sum_{t=1}^{T} \Phi \times \frac{\partial c_t}{\partial f_t} \times \frac{\partial f_t}{\partial b_f} = \sum \Phi \odot c_{t-1} \odot f_t(1-f_t) \otimes 1$$

$$\frac{\partial \mathcal{L}}{\partial i_t} = \Phi \times \frac{\partial c_t}{\partial i_t} = \Phi \odot g_t$$

$$\frac{\partial \mathcal{L}}{\partial w_i} = \sum_{t=1}^{T} \Phi \times \frac{\partial c_t}{\partial i_t} \times \frac{\partial i_t}{\partial w_i} = \sum_{t=1}^{T} \Phi \odot g_t \odot (i_t(1-i_t)) \times \underbrace{\begin{bmatrix} h_{t-1} \\ x_t \end{bmatrix}^T}_{\hat{x}^T}$$

$$\frac{\partial \mathcal{L}}{\partial b_i} = \sum_{t=1}^{T} \Phi \times \frac{\partial c_t}{\partial i_t} \times \frac{\partial i_t}{\partial b_i} = \sum_{t=1}^{T} \Phi \odot g_t \odot (i_t(1-i_t)) \times 1$$

$$\frac{\partial \mathcal{L}}{\partial g_t} = \Phi \times \frac{\partial c_t}{\partial g_t} = \Phi \odot i_t$$

$$\frac{\partial \mathcal{L}}{\partial w_g} = \sum_{t=1}^{T} \Phi \times \frac{\partial c_t}{\partial g_t} \times \frac{\partial g_t}{\partial w_g} = \sum_{t=1}^{T} \Phi \odot i_t \odot (1-g_t^2) \times \hat{x}^T$$

$$\frac{\partial \mathcal{L}}{\partial b_g} = \sum \Phi \times \frac{\partial c_t}{\partial g_t} \times \frac{\partial g_t}{\partial b_g} = \sum_{t=1}^{T} \Phi \odot i_t \odot (1-g_t^2) \times 1$$

Scanned with CamScanner

$$\frac{\partial L}{\partial o_t} = \frac{\partial L}{\partial h_t} \times \frac{\partial h_t}{\partial o_t} = \left(\frac{\partial L_t}{\partial h_t} + \delta_{h_t}\right) \odot \tanh(c_t)$$

$$\frac{\partial L}{\partial w_o} = \frac{\partial L}{\partial h_t} \times \frac{\partial h_t}{\partial o_t} \times \frac{\partial o_t}{\partial w_o} = \left(\frac{\partial L_t}{\partial h_t} + \delta_{h_t}\right) \odot \tanh(c_t) \odot o_t \odot (1-o_t) \times \hat{x}^T$$

$$\frac{\partial L}{\partial b_o} = \frac{\partial L}{\partial h_t} \times \frac{\partial h_t}{\partial o_t} \times \frac{\partial o_t}{\partial b_o} = \left(\frac{\partial L_t}{\partial h_t} + \delta_{h_t}\right) \odot \tanh(c_t) \odot o_t \odot (1-o_t) \times \hat{x}^T$$

برای حساب کردن $\delta_{h_{t-1}} = \frac{\partial L}{\partial h_{t-1}}$ باید وزن‌های مربوط به h_{t-1} را جدا کنیم. وزن‌های مثلا برای w_f

می‌نویسیم و همین کار برای بقیه w ها تکراری شود

$$\underset{w_f}{\left[\quad\quad\quad \Big| \quad\quad\right]} \begin{bmatrix} h_{t-1} \\ x_t \end{bmatrix}$$

$w_f[:, :d_h] = w_{fh}$ → این کار را برای همه پارامترها می‌کنیم

که میشه w_{oh} ، w_{ih} ، w_{gh}

$$\delta_{h_{t-1}} = \frac{\partial L}{\partial h_{t-1}} = \frac{\partial L}{\partial h_t} \times \left(\frac{\partial h_t}{\partial f_t} \times \frac{\partial f_t}{\partial h_{t-1}} + \frac{\partial h_t}{\partial i_t} \frac{\partial i_t}{\partial h_{t-1}} + \frac{\partial h_t}{\partial g_t} \frac{\partial g_t}{\partial h_{t-1}} + \frac{\partial h_t}{\partial o_t} \frac{\partial o_t}{\partial h_{t-1}}\right)$$

$$\delta_{h_{t-1}} = \frac{\partial L}{\partial h_{t-1}} = \frac{\partial L}{\partial f_t} \times \frac{\partial f_t}{\partial h_{t-1}} + \frac{\partial L}{\partial i_t} \times \frac{\partial i_t}{\partial h_{t-1}} + \frac{\partial L}{\partial g_t} \times \frac{\partial g_t}{\partial h_{t-1}} + \frac{\partial L}{\partial o_t} + \frac{\partial o_t}{\partial h_{t-1}}$$

$$= w_{fh}^T\left(\Phi \odot c_{t-1} \odot f_t \odot (1-f_t)\right) + w_{ih}^T\left(\Phi \odot g_t \odot i_t \odot (1-i_t)\right) + w_{gh}^T\left(\Phi \odot i_t \odot (1-g_t^2)\right)$$

$$+ w_{oh}^T\left(\left(\frac{\partial L_t}{\partial h_t} + \delta_{h_t}\right) \odot \tanh(c_t) \odot o_t \odot (1-o_t)\right)$$

دلیل درست شدن dim از اینجا فهمیدی

CS Scanned with CamScanner

سوال 2-بخش الف

می خواهیم نشان دهیم که (Self-Attention) را می توان با ضرب ماتریس(لایه تماما متصل) نشان داد

می توانیم معادله attention را به صورت زیر بازنویسی کرد:

$$Y = \text{Softmax}\left(\frac{(W^Q X)(W^K X)^T}{\sqrt{d_k}}\right) W^V X$$

X یک دنباله ورودی که ابعاد آن $N \times D$ ، ماتریس های W^Q، W^K و W^V با ابعاد $d_k \times D$

$$M = \text{Softmax}\left(\frac{(W^Q X)(W^K X)^T}{\sqrt{d_k}}\right) \equiv \text{Softmax}(Z)$$

بنابراین

$$M = \text{Softmax}(Z) \quad , \quad V = W^V X$$

$$Y = MV$$

این اثبات مانند یک شبکه fully connected ما را با ضرب ماتریسی از ورودی به خروجی میبرد یعنی میتوانیم به این صورت بازنویسی را داشته باشیم $Y = W\ X$

سوال 2-بخش ب

ضرب های شامل W^Q, W^K, W^V : هر کدام با اندازه های $d_k \times D$

از آنجایی که ما این سه ماتریس را داریم:

$$Total\ Parameters = 3 \times dk \times D$$

معمولا $dk = D$ و یا اینکه $d_k \approx D$ که رفرنس آن مقاله attention is all you need است

$$Total\ Parameters \approx 3 \times D \times D\ = O(D^2)$$

بنابراین با فرض داشتن N ورودی (N = sequence length):

- برای هر موقعیت i,j به یک ماتریس $D \times D$ نیاز داریم.
- ما به تعداد N^2 از چنین موقعیت هایی داریم. (ماتریس وزن توجه M دارای اندازه N×N است زیرا pairwise similarity ها را برای N توکن محاسبه می کند.)
- بنابراین تعداد کل پارامتر ها چنین می شود:

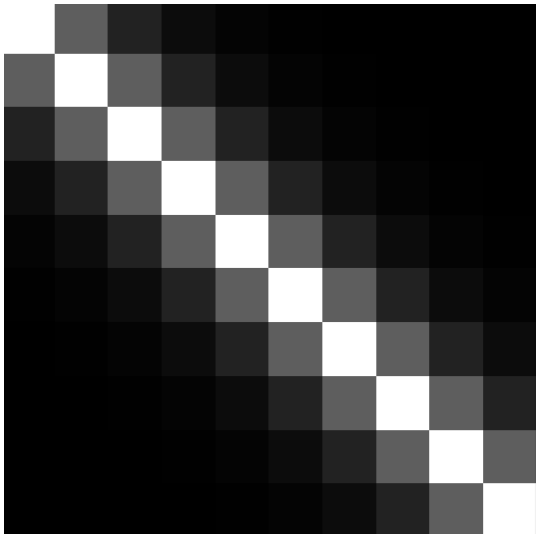
$$Final\ Parameters\ = O(N^2 D^2)$$

سوال 2-بخش ج

در سازوکار خودتوجهی (Self-Attention)، هر عنصر (مانند یک کلمه یا توکن) در دنباله ورودی قابلیت توجه به تمام عناصر دیگر موجود در همان دنباله را دارد. با این حال، شدت یا اهمیت این توجه برای تمام جفت‌های ممکن یکسان نیست.

برای تعیین میزان اهمیت توجه هر عنصر به سایر عناصر، وزن‌های توجه محاسبه می‌شوند. این محاسبه اغلب با استفاده از ضرب نقطه‌ای (dot product) میان بردارهای نمایش‌دهنده موقعیت‌های مختلف آغاز می‌شود که یک ماتریس از امتیازات خام توجه را تولید می‌کند.

سپس، برای تبدیل این امتیازات خام به یک توزیع احتمال معتبر که وزن‌های توجه نهایی را مشخص کند، از تابع softmax استفاده می‌شود. ویژگی کلیدی تابع softmax در این زمینه این است که تفاوت بین امتیازات را تشدید می‌کند؛ به این معنی که امتیازات بالاتر وزن‌های توجه به‌مراتب بیشتری دریافت می‌کنند، در حالی که امتیازات پایین‌تر به وزن‌هایی نزدیک به صفر تبدیل می‌شوند. این فرآیند منجر به یک توزیع توجه 'پراکنده' (sparse) می‌شود که در آن تنها تعداد محدودی از عناصر وزن توجه قابل ملاحظه‌ای از عنصر مبدأ دریافت می‌کنند. به عبارت دیگر، اعمال تابع softmax به طور مؤثری بر مرتبط‌ترین کلمات یا موقعیت‌ها تأکید کرده و در عین حال توجه به موارد کمتر مرتبط را کاهش می‌دهد و منجر به ماتریس وزن توجهی می‌شود که بیشتر درایه‌های آن مقادیر کوچکی نزدیک به صفر دارند.



1. **محورها (Axes):** هر دو محور افقی و عمودی نشان‌دهنده موقعیت‌های عناصر (مانند کلمات یا توکن‌ها) در دنباله ورودی هستند.
2. **مربعات (Cells):** هر عنصر در ماتریس در موقعیت (j,i) نشان‌دهنده میزان توجهی است که عنصر در موقعیت i به عنصر در موقعیت j می‌کند..
3. **تحلیل الگوهای مشاهده شده در نمودار:**

- **قطر اصلی:** روشن‌ترین و سفیدترین مربع‌ها بر روی قطر اصلی ماتریس قرار دارند (موقعیت‌هایی مانند (۰,۰)، (۱,۱)، (۲,۲) و غیره). این نشان می‌دهد که هر عنصر بیشترین میزان توجه را به خودش اختصاص می‌دهد. این یک ویژگی بسیار رایج در خودتوجهی است، زیرا معمولاً هویت و ویژگی‌های خود کلمه یا توکن برای پردازش آن بسیار مهم است.
- **عناصر نزدیک به قطر اصلی:** عناصر خاکستری روشن و تیره در مجاورت قطر اصلی (مانند (۰,۱)، (۱,۰)، (۱,۲)، (۲,۱) و...) نشان می‌دهند که عناصر علاوه بر خودشان، به همسایگان نزدیک خود نیز توجه قابل توجهی می‌کنند. شدت این توجه با فاصله گرفتن از قطر اصلی (یعنی فاصله گرفتن از خود عنصر اصلی) کاهش می‌یابد (خاکستری تیره‌تر می‌شود).
- **عناصر دور از قطر اصلی:** بیشتر عناصر دور از قطر اصلی کاملاً مشکی هستند. این بدان معناست که عناصر در این موقعیت‌ها توجه بسیار کمی به عناصر دیگر که در دنباله از آن‌ها دور هستند، اختصاص می‌دهند. وزن توجه برای این جفت‌ها نزدیک به صفر است.

بیشتر وزن‌های توجه بین عناصر دور از هم بسیار پایین (نزدیک به صفر) هستند. این ماهیت "پراکنده" توجه را نشان می‌دهد که در آن توجه به جای توزیع یکنواخت بر روی همه عناصر، بر روی زیرمجموعه‌ای خاص (خود عنصر و همسایگان نزدیک) متمرکز شده است.

سوال 2-بخش د

می توان E_t را به صورت زیر بازنویسی کرد

$$E_t^{(m)} = \begin{cases} \sin\left(\frac{t}{f_m}\right) & for\ odd\ m \\ \cos\left(\frac{t}{f_m}\right) & for\ even\ m \end{cases}$$

حال با استفاده از رابطه بالا E_{t+k} را بدست می آوریم.

$$\sin\left(\frac{t+\mathrm{k}}{f_m}\right) = \sin\left(\frac{t}{f_m}\right)\ \cos\left(\frac{\mathrm{k}}{f_m}\right) + cos\left(\frac{t}{f_m}\right)\sin\left(\frac{\mathrm{k}}{f_m}\right)$$

$$cos\left(\frac{t+\mathrm{k}}{f_m}\right) = cos\left(\frac{\mathrm{t}}{f_m}\right)cos\left(\frac{\mathrm{k}}{f_m}\right) - sin\left(\frac{\mathrm{t}}{f_m}\right)\ sin\left(\frac{\mathrm{k}}{f_m}\right)$$

همانطور که در ضرب ماتریسی زیر مشخص است توانستیم یک تبدیل خطی را بدست بیاوریم که در آن ماتریس اول $T^{(k)}$ و ماتریس دوم E_t همان است

$$\begin{bmatrix} cos\left(\frac{k}{f_m}\right) & sin\left(\frac{k}{f_m}\right) \\ -sin\left(\frac{k}{f_m}\right) & cos\left(\frac{k}{f_m}\right) \end{bmatrix}\begin{bmatrix} \sin\left(\frac{\mathrm{t}}{f_m}\right) \\ cos\left(\frac{\mathrm{t}}{f_m}\right) \end{bmatrix} = \begin{bmatrix} sin\left(\frac{\mathrm{t}+k}{f_m}\right) \\ cos\left(\frac{\mathrm{t}+k}{f_m}\right) \end{bmatrix}$$

سوال 3-بخش الف

(i)

بر اساس فرمول (2) ارائه شده، αi به صورت زیر تعریف می‌شود:

$$\alpha_i = \frac{\exp(k_i^T q)}{\sum_{j=1}^{n} \exp(k_j^T q)}$$

این فرمول همان تابع softmax است. تابع softmax ورودی‌ها را به یک توزیع احتمال تبدیل می‌کند، به این معنی که خروجی‌ها مقادیر غیرمنفی هستند و مجموع آنها برابر با یک است.

بنابراین، α را می‌توان به عنوان یک توزیع احتمال categorical تفسیر کرد زیرا:

- هر α_i 0 بزرگتر یا مساوی صفر است (چون تابع نمایی همیشه مثبت است و مخرج نیز مجموع مقادیر مثبت است).
- مجموع تمام α_i برای $i = 1, \dots, n$ برابر با یک است:

$$\sum_{i=1}^{n} \alpha_i = \sum_{i=1}^{n} \frac{\exp(k_i^T q)}{\sum_{j=1}^{n} \exp(k_j^T q)}$$

با توجه به اینکه مخرج برای تمام جملات در مجموع یکسان است، می‌توان آن را از مجموع فاکتور گرفت:

$$\sum_{i=1}^{n} \alpha_i = \frac{1}{\sum_{j=1}^{n} \exp(k_j^T q)} \sum_{i=1}^{n} \exp(k_i^T q)$$

صورت کسر و مخرج کسر اکنون یکسان هستند (فقط نمایه جمع متفاوت است، اما روی مجموعه یکسانی از مقادیر جمع بسته می‌شود). بنابراین:

$$\sum_{i=1}^{n} \alpha_i = \frac{\sum_{i=1}^{n} \exp(k_i^T q)}{\sum_{j=1}^{n} \exp(k_j^T q)} = 1$$

این ویژگی که مجموع α_i برابر با یک است، به ما اجازه می‌دهد تا α را به عنوان یک توزیع احتمال در نظر بگیریم که در آن α_i نشان‌دهنده احتمال انتخاب بردار مقدار v_i است.

(ii)

توزیع α زمانی تقریباً تمام وزن خود را روی مقدار α_j متمرکز می‌کند که عبارت ورودی به تابع نمایی برای آن نمایه خاص j، (یعنی $k_j^T q$) به طور قابل توجهی بزرگتر از عبارات مشابه برای سایر نمایه‌ها $i \neq j$ باشد. به عبارت دیگر، برای اینکه توزیع α بر روی یک α_j خاص متمرکز شود، بردار پرسش q باید شباهت بسیار بیشتری (بر اساس ضرب داخلی) با بردار کلید k_j داشته باشد تا با هر بردار کلید دیگر k_i. این اتفاق زمانی می‌افتد که ضرب داخلی $k_j^T q$ حداکثر مقدار را در میان تمام ضرب‌های داخلی $k_i^T q$ داشته باشد و این حداکثر مقدار به میزان قابل توجهی از مقادیر دیگر بزرگتر باشد.

(iii)

بر اساس توضیحاتی که در بخش (ii) بیان شد، اگر توزیع α "پراکنده" باشد، به این معنی است که مقادیرα_i به جای اینکه تقریباً تمام وزن خود را روی یک α_i خاص متمرکز کنند، بین چندین یا همه v_i توزیع شده‌اند (یعنی چندین α_i مقادیر قابل توجهی دارند و نزدیک به صفر نیستند). این وضعیت زمانی رخ می‌دهد که شباهت بردار پرسش q با چندین بردار کلید k_i (محاسبه شده از طریق ضرب داخلی $k_i^T q$) نسبتاً نزدیک به هم باشد، در نتیجه مقادیر $exp(k_i^T q)$ و به تبع آن α_i برای چندین i قابل مقایسه هستند.

با توجه به فرمول تعریف خروجی c:

$$c = \sum_{i=1}^{n} \alpha_i v_i$$

که نشان می‌دهد c یک مجموع وزن‌دار از بردارهای مقدار v_i است، اگر توزیع α پراکنده باشد، خروجی c ویژگی‌هایی خواهد داشت که ترکیبی از چندین بردار مقدار v_i هستند. به عبارت دیگر، c یک "میانگین" یا "ترکیب خطی" از چندین بردار v_i خواهد بود که وزن هر بردار v_i در این ترکیب توسط α_i متناظر آن تعیین می‌شود. این باعث می‌شود که c نماینده‌ای از اطلاعات جمع‌آوری شده از چندین منبع (بردارهای مقدار v_i) باشد که هر کدام به میزان مرتبط بودنشان با پرسش q (که توسط α_i نشان داده می‌شود) در خروجی نهایی سهم دارند. برخلاف حالت تمرکز که c تقریباً برابر با یک v_j خاص می‌شد، در حالت پراکندگی c می‌تواند اطلاعات متنوع‌تری را از چندین بردار مقدار در خود جای دهد.

(iv)

در راستای توضیح امکان شباهت بین بردار خروجی مکانیزم توجه و یکی از بردارهای مقدار، با توجه به آنچه در بخش‌های (ii) و (iii) بررسی شد، می‌توان نتیجه گرفت که:

خروجی c به صورت مجموع وزن‌دار بردارهای مقدار vi تعریف می‌شود:

$$c = \sum_{i=1}^{n} \alpha_i v_i$$

که در آن α_i وزن‌های توجه هستند و مجموعشان برابر با یک است، و این وزن‌ها بر اساس شباهت بردار پرسش q با بردارهای کلید k_i محاسبه می‌شوند.

بر اساس بخش (iii)، اگر توزیع وزن‌های α "پراکنده" باشد (یعنی چندین α_i مقادیر قابل توجهی داشته باشند)، آنگاه c ترکیبی از چندین بردار مقدار v_i خواهد بود و لزوماً به هیچ یک از آن‌ها بسیار شبیه نخواهد بود، بلکه میانگینی از آن‌ها خواهد بود.

اما، امکان شباهت زیاد بین خروجی c و یکی از بردارهای مقدار v_j زمانی رخ می‌دهد که توزیع وزن‌های α به جای پراکندگی، بر روی یک نمایه خاص j "متمرکز" شود. این تمرکز زمانی اتفاق می‌افتد که شباهت بردار پرسش q با بردار کلید متناظر k_j (که با عبارت:

$$k_j^T q$$

سنجیده می‌شود) به طور قابل توجهی بیشتر از شباهت q با سایر بردارهای کلید ki باشد. در این حالت، مقدار α_j به ۱ نزدیک می‌شود و مقادیر سایر α_i به صفر نزدیک می‌شوند.
وقتی $\alpha_j \approx 1$ و $\alpha_i \approx 0$ برای $i = j$ باشد، در فرمول c داریم:

$$c = \alpha_1 v_1 + \ + \alpha_j v_j + \ + \alpha_n v_n \approx 0 \cdot v_1 + \ + 1 \cdot v_j + \ + 0 \cdot v_n = v_j$$

بنابراین، خروجی c بسیار شبیه به بردار مقدار v_j می‌شود.
نتیجه کلی این است که مکانیزم توجه این توانایی را دارد که خروجی خود را به یکی از بردارهای مقدار که بیشترین ارتباط یا شباهت را با پرسش ورودی q دارد، نزدیک کند. این انتخاب و تمرکز بر روی یک بردار مقدار خاص از طریق سازوکار محاسبه وزن‌های توجه α و ماهیت تابع softmax امکان‌پذیر می‌شود.

سوال 3-بخش ب

(i)

فرض کنید A ماتریسی است که از الحاق بردارهای پایه $a1, \dots, am$ تشکیل شده است و B ماتریسی است که از الحاق بردارهای پایه $b1, \dots, bp$ تشکیل شده است. ترکیب‌های خطی بردارهای v_a و v_b را می‌توان به صورت زیر بیان کرد:

$$v_a = c_1 a_1 + c_2 a_2 + \cdots + c_m a_m = Ac$$
$$v_b = d_1 b_1 + d_2 b_2 + \cdots + d_m b_m = Bd$$

ما باید ماتریس M را طوری بسازیم که وقتی در v_b ضرب می‌شود، حاصل صفر شود و وقتی در va ضرب می‌شود، همان بردار را تولید کند (در فضای خودش):

$$Mv_b = 0$$

$$Mv_a = v_a$$

$$Ms = v_a$$

$$M(v_a + v_b) = v_a$$

$$Mv_a + Mv_b = v_a$$

به راحتی می‌توان دید که چون $a_j^T b_k = 0$ برای تمام k,j (دو زیرفضای A و B بر هم متعامد هستند)،

$$A^T B = 0$$

. همچنین، چون $a_i^T a_j = 0$ برای $i = j$ و $a_i^T a_i = 1$ هرگاه $i = j$ باشد (تمام بردارهای پایه نرم 1 دارند و بر هم متعامد هستند)،

$$A^T A = I$$

. اگر M را با A^T، v_a را با A_c و v_b را با B_d جایگزین کنیم:

$$a_j^T b_k = 0$$

$$a_i^T a_j = 0 \text{ for } i \neq j$$

$$a_i^T a_i = 1 \text{ whenever } i = j$$

$$A^T Ac + A^T Bd = Ic + 0d = c$$

ما می‌دانیم که در فضای R^d (نه بر حسب A یا B)، v_a فقط مجموعه‌ای از c است (یا می‌توانیم v_a را به عنوان c که به صورت یک بردار در R^dبیان شده است در نظر بگیریم). بنابراین، $M = A^T$.

(ii)

$c \approx 0.5v_a + 0.5v_b$این به معنای آن است که αa تقریباً برابر با 0.5 و αb نیز تقریباً برابر با 0.5 است. این وضعیت می‌تواند زمانی محقق شود که (برای هر i که نه برابر با a و نه برابر با b است):

$$k_a^T q \approx k_b^T q \gg k_i^T q$$

همانطور که در پرسش قبلی توضیح داده شد، اگر حاصل‌ضرب داخلی بزرگ باشد، جرم احتمال مربوط به آن نیز بزرگ خواهد بود. ما به دنبال توزیع متوازنی از جرم احتمال بین αa و αb هستیم. بردار q بیشترین همسویی را با k_a و k_b خواهد داشت وقتی که مضربی بزرگ از برداری باشد که در جهت k_aو در جهت k_b مولفه دارد:

$$q = \beta(k_a + k_b), \text{where } \beta \geq 0$$

اکنون، از آنجایی که کلیدها بر هم متعامد هستند، به راحتی می‌توان نشان داد که حاصل‌ضرب داخلی q با هر یک از کلیدها به صورت زیر خواهد بود:

$$k_a^T q = \beta k_a^T(k_a + k_b) = \beta(k_a^T k_a + k_a^T k_b) = \beta(1 + 0) =$$
$$k_b^T q = \beta k_b^T(k_a + k_b) = \beta(k_b^T k_a + k_b^T k_b) = \beta(0 + 1) =$$
$$k_i^T q = \beta k_i^T(k_a + k_b) = \beta(k_i^T k_a + k_i^T k_b) = \beta(0 + 0) = 0$$

بنابراین، وقتی از تابع نمایی استفاده می‌کنیم، فقط $exp(\beta)$ مقداری قابل توجه خواهد داشت، زیرا $exp(0) = 1$ نسبت به آن در محاسبه جرم احتمال مخرج ناچیز خواهد بود. به این ترتیب به دست می‌آوریم که:

$$\alpha_a = \alpha_b = \frac{\exp(\beta)}{n - 2 + 2\exp(\beta)} \approx \frac{\exp(\beta)}{2\exp(\beta)} \approx \frac{1}{2} \text{ for } \beta \gg 0$$

سوال 3-بخش ج

(i)

از آنجایی که واریانس‌ها (که با مقادیر روی قطر ماتریس کوواریانس مشخص می‌شوند) برای هر $i \in 1,2,\ldots,n$ بسیار ناچیز هستند، این واقعیت نشان‌دهنده آن است که نقاط داده مربوط به هر کلید بسیار نزدیک به میانگین خود قرار دارند. بنابراین، می‌توانیم با تقریب خوبی فرض کنیم که هر بردار کلید k_i بسیار به بردار میانگین متناظر خود نزدیک است:

$$k_i \approx \mu_i$$

از آنجایی که تمام بردارهای میانگین (μ_i) بر هم متعامد هستند (یعنی حاصل‌ضرب داخلی هر جفت متمایز از آن‌ها صفر است)، با توجه به تقریب $k_i \approx \mu_i$، مسئله‌ای مشابه با حالت قبل که در آن تمام

بردارهای کلید (ki) بر هم عمود بودند، مطرح می‌شود. این شباهت ساختاری به ما اجازه می‌دهد تا از تحلیل‌های قبلی برای بررسی تأثیر بردار پرسش q بر توزیع توجه استفاده کنیم. در این چارچوب، بردار پرسش q را می‌توان به گونه‌ای طراحی کرد که بیشترین همسویی را با دو بردار میانگین خاص، مثلاً μ_a و μ_b، داشته باشد تا توزیع توجه α بر روی مقادیر مربوط به آن‌ها متمرکز شود. این حالت با در نظر گرفتن q به صورت ترکیب خطی این دو بردار میانگین، با ضریبی بزرگ و مثبت، ایجاد می‌شود:

$$q = \beta(\mu_a + \mu_b), \text{where } \beta \gg 0$$

در اینجا، β یک اسکالر مثبت و بسیار بزرگ است. مقدار بزرگ β اهمیت دارد زیرا تضمین می‌کند که حاصل‌ضرب‌های داخلی $k_b^T q$و $k_a^T q$ (که به دلیل تقریب $k_i \approx \mu_i$ و تعامد میانگین‌ها، تقریباً برابر با β می‌شوند) به طور قابل توجهی بزرگتر از حاصل‌ضرب‌های داخلی $k_i^T q$ برای سایر i باشند (که تقریباً صفر می‌شوند). این اختلاف بزرگ در مقادیر ورودی تابع softmax باعث می‌شود که وزن‌های α_a و α_b به طور قابل ملاحظه‌ای بزرگتر از سایر وزن‌های α_i شوند و توزیع توجه بر روی این دو مولفه متمرکز گردد.

(ii)

در این سناریوی خاص، رفتار بردار کلید ka با سایر کلیدها تفاوت دارد. در حالی که سایر بردارهای کلید k_i برای هر i که برابر با a نیست، تقریباً ثابت و بدون تغییر قابل توجهی باقی می‌مانند (به دلیل واریانس‌های بسیار ناچیز)، بردار k_a مقداری حول میانگین خود μ_a با یک عامل مقیاس‌دهنده تصادفی تغییر می‌کند. این تغییر را می‌توان به صورت تقریبی زیر مدل‌سازی کرد:

$$k_a \approx \gamma\mu_a, \text{where } \gamma \sim N(1,0.5)$$

در این رابطه، γ یک متغیر تصادفی است که از توزیع نرمال با میانگین ۱ و واریانس ۰.۵ پیروی می‌کند. این بدان معناست که ضریب مقیاس‌دهنده برای μ_a در تشکیل k_a مقداری حول ۱ دارد، اما دقیقاً ۱ نیست. برای سایر کلیدها ($i = a$)، تقریب ساده‌تر برقرار است:

$$k_i \approx \mu_i, \text{whenever } i \neq a$$

از آنجایی که بردار پرسش q به گونه‌ای تنظیم شده است که بیشترین همسویی را در جهت‌های بردارهای کلید k_a و k_b داشته باشد (مشابه تحلیل‌های بخش‌های پیشین)، می‌توان فرض کرد که حاصل‌ضرب داخلی بین q و هر بردار کلید دیگر ki که متفاوت از k_a و k_b است، نزدیک به صفر خواهد بود. این فرض بر پایه تعامد بردارهای میانگین μ_i استوار است. بنابراین، برای درک نحوه تأثیر q بر وزن‌های توجه، باید حاصل‌ضرب داخلی q را با k_a و k_b بررسی کنیم. با استفاده از مدل تقریبی k_a و k_i و فرض اینکه q به صورت $q = \beta(\mu_a + \mu_b)$ با $\beta \gg 0$ است (همانند بخش قبلی)، حاصل‌ضرب‌های داخلی اصلی به صورت زیر به دست می‌آیند:

حاصل‌ضرب داخلی $k_a^T q$:

$$k_a^T q \approx (\gamma\mu_a)^T\big(\beta(\mu_a + \mu_b)\big) = \gamma\beta(\mu_a^T\mu_a + \mu_a^T\mu_b)$$

با فرض نرمال بودن بردارهای میانگین ($\mu_a^T\mu_a = 1$) و تعامد آن‌ها ($\mu_a^T\mu_b = 0$)، این عبارت ساده می‌شود به:

$$k_a^T q \approx \gamma\beta(1+0) = \gamma$$

حاصل‌ضرب داخلی $k_b^T q$:

$$k_b^T q \approx \mu_b^T\big(\beta(\mu_a + \mu_b)\big) = \beta(\mu_b^T\mu_a + \mu_b^T\mu_b)$$

با توجه به تعامد ($\mu_b^T\mu_a = 0$) و نرمال بودن ($\mu_b^T\mu_b = 1$):

$$k_b^T q \approx \beta(0+1) =$$

برای سایر کلیدها ($i = a, b$)، $k_i^T q \approx \mu_i^T\beta(\mu_a + \mu_b) = \beta(\mu_i^T\mu_a + \mu_i^T\mu_b) = \beta(0+0) = 0$.

حال می‌توانیم با استفاده از فرمول softmax، تقریب‌هایی برای وزن‌های توجه αa و αb به دست آوریم.

با یادآوری اینکه برای مقادیر بزرگ β، ترم‌های $exp(0) = 1$ مربوط به سایر کلیدها در مخرج سهم قابل توجهی دارند (در واقع $n-2$ ترم داریم که هر کدام به ۱ نزدیک می‌شوند):

$$\alpha_a = \frac{\exp(k_a^T q)}{\sum_{j=1}^{n}\exp\big(k_j^T q\big)} \approx \frac{\exp(\gamma\beta)}{\exp(\gamma\beta) + \exp(\beta) + (n-2)\exp(0)} = \frac{\exp(\gamma\beta)}{\exp(\gamma\beta) + \exp(\beta) + n - 2}$$

برای $\beta \gg 0$، ترم‌های $exp(\gamma\beta)$ و $exp(\beta)$ بسیار بزرگتر از $n-2$ می‌شوند، بنابراین می‌توان از $n-2$ صرف نظر کرد:

$$\alpha_a \approx \frac{\exp(\gamma\beta)}{\exp(\gamma\beta) + \exp(\beta)} = \frac{1}{1 + \exp(\beta - \gamma\beta)} = \frac{1}{1 + \exp\big(\beta(1-\gamma)\big)}$$

به طور مشابه برای α_b:

$$\alpha_b = \frac{\exp(k_b^T q)}{\sum_{j=1}^{n}\exp\big(k_j^T q\big)} \approx \frac{\exp(\beta)}{\exp(\gamma\beta) + \exp(\beta) + n - 2} \approx \frac{\exp(\beta)}{\exp(\gamma\beta) + \exp(\beta)} = \frac{1}{1 + \exp(\gamma\beta - \beta)}$$
$$= \frac{1}{1 + \exp\big(\beta(\gamma - 1)\big)}$$

از آنجایی که γ یک متغیر تصادفی در بازه تقریبی [0.5, 1.5] (با توجه به میانگین ۱ و واریانس ۰.۵ توزیع نرمال) است و β بسیار بزرگ است، می‌توانیم دو حالت حدی را برای γ بررسی کنیم:

حالت اول: γ کوچک است (مثلاً $\gamma \approx 0.5$). در این حالت $0 < \gamma - 1$ و $0 > \gamma - 1$.

$$\alpha_a \approx \frac{1}{1 + \exp\big(\beta \cdot \text{مثبت}\big)} \approx \frac{1}{1+\infty} \approx 0$$

$$\alpha_b \approx \frac{1}{1 + \exp\big(\beta \cdot \text{منفی}\big)} \approx \frac{1}{1+0} \approx 1$$

در این حالت، توزیع توجه عمدتاً روی αb متمرکز می‌شود.

حالت دوم: γ بزرگ است (مثلاً $\gamma \approx 1.5$). در این حالت $0 < \gamma - 1$و $0 < \gamma - 1$.

$$\alpha_a \approx \frac{1}{1+\exp\left(\beta \cdot \text{منفی}\right)} \approx \frac{1}{1+0} \approx 1$$
$$\alpha_b \approx \frac{1}{1+\exp\left(\beta \cdot \text{مثبت}\right)} \approx \frac{1}{1+\infty} \approx 0$$

در این حالت، توزیع توجه عمدتاً روی α_a متمرکز می‌شود.

از آنجایی که خروجی c تقریباً برابر با مجموع وزن‌دار v_a و v_b است ، زیرا سهم سایر بردارهای مقدار هنگامی که β بزرگ است ناچیز می‌شود، می‌توان مشاهده کرد که بردار خروجی c می‌تواند بسته به مقدار تصادفی ۷، به بردار مقدار v_a نزدیک شود یا به بردار مقدار v_b. به عبارت دیگر، واریانس در بردار کلید ka باعث می‌شود که خروجی مکانیزم توجه به طور تصادفی بین دو بردار مقدار v_a و v_b "نوسان" کند یا به یکی از آن‌ها متمایل شود، به جای اینکه همیشه یک ترکیب ثابت از آن‌ها باشد. این نشان‌دهنده تأثیر نویز در بردارهای کلید بر تصمیم‌گیری مکانیزم توجه است.

سوال 3-بخش د

(i)

با فرض‌های مشابه قبل، می‌توانیم پرسش‌های q_1 و q_2 را طوری طراحی کنیم که یکی از آن‌ها v_a و دیگری v_b را کپی کند. با توجه به اینکه کلیدها شبیه میانگین‌هایشان هستند و طبق توضیحات قبل، پرسش‌ها را به صورت زیر بیان می‌کنیم:

$$q_1 = \beta\mu_a, \quad q_2 = \beta\mu_b, \text{for } \beta \gg 0$$

با توجه به تعامد میانگین‌ها، این طراحی به ما می‌دهد که خروجی‌های توجه مربوطه تقریباً برابرند با:

$$c_1 \approx v_a; \quad c_2 \approx v_b$$

و از آنجایی که توجه چند-سر (multi-headed attention) در این حالت میانگینی از این دو مقدار است، مشاهده می‌کنیم که خروجی نهایی تقریباً برابر است با:

$$c \approx \frac{1}{2}(v_a + v_b)$$

نکات اضافی:

۱. همچنین می‌توان پرسش‌ها را جابجا تعریف کرد ($q_1 = \beta\mu_b$ و $q_2 = \beta\mu_a$). این کار جای v_a و v_b را در خروجی‌های c_1 و c_2 عوض می‌کند ($c_1 \approx v_b, c_2 \approx v_a$)، اما میانگین نهایی بدون تغییر باقی می‌ماند.

$$q_1 = \beta\mu_b, \quad q_2 = \beta\mu_a$$
$$c_1 \approx v_b, \quad c_2 \approx v_a$$

۲. حتی می‌توان از همان پرسش بخش قبلی $(q_1 = q_2 = \beta(\mu_a + \mu_b))$ برای هر دو پرسش استفاده کرد. در این صورت خروجی هر دو سر توجه یکسان خواهد بود و میانگین آن‌ها نیز همان مقدار تکراری خواهد بود ($c_1 = c_2 = c$).

$$q_1 = q_2 = \beta(\mu_a + \mu_b)$$
$$c1 = c2 = c$$

(ii)

در ارتباط با بخش (ج) دوم، اگر بردارهای پرسش را به صورت$q_1 = \beta\mu_a$ و$q_2 = \beta\mu_b$ انتخاب کنیم، در این صورت (با توجه به اینکه حاصل‌ضرب‌های داخلی سایر کلیدها در این پرسش‌ها ناچیز خواهند بود)، داریم که:

حاصل‌ضرب داخلی k_a با q_1:

$$k_a^T q \approx \gamma \mu_a^T \beta \mu_a \approx \gamma\beta, \text{where } \beta \gg 0$$

حاصل‌ضرب داخلی k_b با q_2:

$$k_b^T q \approx \mu_b^T \beta \mu_b \approx \beta, \text{where } \beta \gg 0$$

می‌توانیم برای مقادیر α حل کنیم (باز هم توجه کنید که حاصل‌ضرب‌های داخلی سایر کلیدها وقتی β بزرگ است ناچیز هستند):

برای پرسش q_1:

$$\alpha_{a1} \approx \frac{\exp(\gamma\beta)}{\exp(\gamma\beta)} \approx 1$$

برای پرسش q_2:

$$\alpha_{b2} \approx \frac{\exp(\beta)}{\exp(\beta)} \approx 1$$

از آنجایی که می‌توان گفت $\alpha_i 1 \approx 0$ برای تمام $i = a$ و $\alpha_i 2 \approx 0$ برای تمام $i = b$ است، به راحتی دیده می‌شود که خروجی‌های متناظر هر سر توجه تقریباً برابرند با:

$$c_1 \approx v_a, \quad c_2 \approx v_b$$

این بدان معناست که خروجی نهایی توجه چند-سر (که میانگین c_1 و c_2 است) همیشه تقریباً میانگینی از مقادیر v_a و v_b خواهد بود:

$$c \approx \frac{1}{2}(v_a + v_b)$$

سوال 4-بخش 1

در اینجا جزئیات محاسبات برای تولید توکن جدید (توکن n+1) آورده شده است(یعنی تا الان n توکن تولید کردیم و میخوایم توکن n+1 رو تولید کنیم:

1. **Projections برای Q, K, V:**
   - محاسبه بردار Q برای توکن جدید: ضرب جاسازی توکن جدید (اندازه 1×d) در ماتریس تبدیل W_Q (اندازه $d \times d_h$). هزینه برای هر سر:$\approx d \times d_h$.
   - محاسبه بردارهای K برای تمام n+1 توکن (n توکن قبلی + توکن جدید): ضرب داخلی این توکن‌ها (اندازه (n+1)×d) در ماتریس WK (اندازه $d \times d_h$). هزینه برای هر سر: $\approx d_h$ ×d×(n+1).
   - محاسبه بردارهای V برای تمام n+1 توکن: مشابه K. هزینه برای هر سر: $\approx d \times d_h \times (n+1)$.

   - **کل هزینه :(برای nh سر)**

$$n_h \times [d_h \times d + d_h \times d \times (1+n) + d_h \times d \times (1+n)] = n_h \times (2n+3) \times d_h \times d.$$

2. **Attention Scores:**
   - محاسبه ضرب داخلی بین بردار Q توکن جدید (1×dh) و بردارهای K تمام n+1 توکن .هزینه برای هر سر: $\approx (n+1) \times d_h$.
   - مقیاس‌بندی و اعمال Softmax هزینه‌ای کمتر و متناسب با تعداد امتیازات (O(n)) دارند.

   - **کل هزینه امتیازات:**

$$n_h \times d_h \times (1+n).$$

3. **Value Weighting:**
   - ضرب attention score ها در بردارهای V تمام n+1 توکن (اندازه d_h ×(n+1)).
   - هزینه برای هر head:$d_h \times (n+1)$.

   - **کل هزینه:**

$$n_h \times (n+1) \times d_h$$

4. **Output Projection:**
   - خروجی‌های همه n_h سر (هر کدام 1×dh) به هم متصل شده (Concatenate) و برداری به اندازه $1\times(n_h d_h)$ تشکیل می‌دهند.
   - این بردار حاصل در ماتریس تبدیل خروجی نهایی W_O (اندازه $d\times(n_h d_h)$) ضرب می‌شود.
   - **هزینه تبدیل خروجی:**

$$n_h \times d_h \times d.$$

جمع کل هزینه ها:

$$cost = n_h(2n+3)d_h d + 2n_h d_h(\mathrm{n}+1) + n_h d_h d$$

سوال 4-بخش2

1. توضیح KV-Cache:

- **هدف:** در طول فرآیند تولید توکن به توکن توسط مدل‌های رمزگشا، مکانیزم توجه (Attention) نیاز دارد که امتیازات شباهت Scores را بین بردار Query توکن فعلی و بردارهای Key تمامی توکن‌های قبلی محاسبه کند. همچنین برای محاسبه خروجی نهایی وزن‌دهی شده، به بردارهای Value تمامی توکن‌های قبلی نیاز دارد. بدون استفاده از حافظه پنهان Cache، بردارهای K و V برای تمامی توکن‌های پیشین باید در هر گام تولید توکن، مجدداً از روی جاسازی‌هایشان محاسبه شوند که همانطور که در تحلیل قبلی دیدیم، بسیار ناکارآمد و پرهزینه است.
- **مکانیزم:** KV-cache بردارهای K و V را که برای هر توکن در زمان پردازش محاسبه می‌شوند، ذخیره می‌کند. هنگامی که مدل توکن بعدی (مثلاً توکن n+1) را تولید می‌کند، تنها کافیست:
  - بردارهای Q، K و V را فقط برای توکن جدید (n+1) بر اساس جاسازی ورودی آن محاسبه کند.
  - بردارهای K و V جدید را به بردارهای K و V که قبلاً در حافظه پنهان برای توکن‌های ۱ تا n ذخیره شده بودند، اضافه کند.
  - از بردار Q جدید و تمامی تاریخچه ذخیره شده در حافظه پنهان برای K و V (که حالا شامل توکن n+1 هم می‌شود) برای محاسبه امتیازات توجه و خروجی نهایی در این گام استفاده کند.

2. تحلیل هزینه محاسباتی:

- **Projections برای Q, K, V:**
  - فقط Q، K، V برای توکن جدید (n+1) محاسبه می‌شود.
  - محاسبه Q جدید (اندازه dh×1): هزینه برای هر سر ≈d×dh.
  - محاسبه K جدید (اندازه dh×1): هزینه برای هر سر ≈d×dh.
  - محاسبه V جدید (اندازه dh×1): هزینه برای هر سر ≈d×dh.

  - **کل هزینه تبدیل‌ها:**

$$3 \times n_h \times d_h \times d$$

- **Update Cache:** اضافه کردن K جدید و V جدید به حافظه پنهان موجود. حافظه پنهان اکنون (n+1) بردار K و (n+1) بردار V در هر سر دارد. این عمدتاً یک عملیات حافظه‌ای است.
- **Attention Score:**
  - محاسبه ضرب نقطه‌ای بین Qجدید (اندازه d_h×1) و تمامی بردارهای K در حافظه پنهان به‌روز شده
  - هزینه برای هر سر: ≈d_h×(n+1).

  - **کل هزینه امتیازات:**

$$n_h \times d_h \times (\mathrm{n}+1)$$

- **Value Weighting:**
  - ضرب وزن‌های توجه در تمامی بردارهای V در حافظه پنهان به‌روز شده
  - هزینه برای هر سر: ≈(n+1)×d_h.
  -
  - **کل هزینه:**

$$n_h \times (n+1) \times d_h$$

- **Output Projection:**
  - مشابه قبل: اتصال خروجی سرها و ضرب در W_O.
  - **هزینه تبدیل خروجی:**

$$nh \times dh \times d$$

جمع کل و تقریبی هزینه‌ها:

$$cost = 3n_h d_h d + 2n_h(n+1)d_h + n_h d_h d$$

3. میزان حافظه ذخیره‌سازی مورد نیاز:

در گام n (هنگام تولید توکن n+1)، طول متن قبلی n است. حافظه پنهان بردارهای K و V را برای این n توکن ذخیره می‌کند.

- اندازه حافظه پنهان K : $n \times n_h \times d_h$ عنصر.
- اندازه حافظه پنهان V : $n \times n_h \times d_h$ عنصر.
- کل عناصر(میزان حافظه مورد نیاز): $2 \times n \times n_h \times d_h$.

سوال 4-بخش 3

1. توضیح MQA با KV-Cache:

- **مکانیزم:** در MQA، embedding ورودی به n_h سر Query نگاشت می‌شود، اما فقط یک سر Key و یک سر Value وجود دارد که بین تمام سرهای Q به اشتراک گذاشته می‌شوند. در محاسبات توجه، هر یک از n_h سر Q به همان تک سر K توجه می‌کند و از *همان* تک سر V استفاده می‌کند.
- **KV-Cache در MQA:** به جای ذخیره n_h مجموعه بردار K و V برای هر توکن (یک مجموعه برای هر سر)، تنها به ذخیره یک مجموعه بردار K و V برای هر توکن نیاز داریم، زیرا این مجموعه بین همه سرهای توجه به اشتراک گذاشته می‌شود. هنگام تولید توکن جدید، تک بردارهای K و V جدید محاسبه و به حافظه پنهان مشترک اضافه می‌شوند.:اصلی‌ترین مزیت MQA، به خصوص با KV-Cache در طول استنتاج، کاهش چشمگیر اندازه KV-Cache است. این امر باعث صرفه‌جویی در حافظه می‌شود و پهنای باند حافظه مورد نیاز برای خواندن بردارهای K و V در طول محاسبه توجه را کاهش می‌دهد.

2. تحلیل هزینه محاسباتی (MQA با KV-Cache):

مراحل تولید توکن n+1:

- **Projections برای Q, K, V:**
  - محاسبه Qجدید برای هر یک از سرهای n_h: هزینه برای تمام سرهای Q تقریباً برابر است با $n_h \times (d \times d_h)$
  - محاسبه تک Kجدید (اندازه $1 \times d_h$): هزینه تقریباً برابر است با $d \times d_h$.
  - محاسبه تک Vجدید (اندازه $1 \times d_h$): هزینه تقریباً برابر است با $d \times d_h$.
  - **کل هزینه ها:**

$$(n_h + 2) \times d \times d_h$$

- **Update Cache:**
  - تک Kجدید و Vجدید محاسبه شده، به حافظه پنهان مشترک اضافه می‌شوند. این حافظه پنهان اکنون (n+1) بردار K و (n+1) بردار V دارد.
- **Attention Score:**
  - برای هر یک از سرهای n_h، ضرب نقطه‌ای بین بردار Qجدید آن سر و بردارهای K ترانهاده شده از حافظه پنهان *مشترک* به‌روز شده محاسبه می‌شود.
  - هزینه برای هر سر Q: تقریباً برابر است با $(n + 1) \times d_h$
  - **کل هزینه:**

$$n_h \times (n + 1) \times d_h$$

- **Value Weighting:**
  - برای هر یک از سرهای n_h، وزن‌های توجه آن سر در بردارهای V از حافظه پنهان *مشترک* به‌روز شده ضرب می‌شود.
  - هزینه برای هر سر Q: تقریباً برابر است با $(n + 1) \times d_h$
  - **کل هزینه:**

$$n_h \times (n + 1) \times d_h$$

- **Output Projection:**
  - اتصال خروجی‌های تمام سرهای n_h و ضرب در ماتریس وزن خروجی W_O.
  - **کل هزینه:**

$$(n_h \times d_h) \times d.$$

جمع کل و تقریبی هزینه‌ها:

$$cost = (n_h + 2).d.d_h + 2.n_h.(n + 1).d_h + n_h.d_h.d$$

3. میزان حافظه ذخیره‌سازی مورد نیاز:

KV-Cache برای MQA فقط یک مجموعه بردار K و V را برای هر توکن در متن فعلی ذخیره می‌کند که بین تمام سرهای توجه به اشتراک گذاشته شده است.

در گام n, حافظه پنهان K و V را برای n توکن ذخیره می‌کند.

- اندازه حافظه پنهان تک K مشترک : $n \times 1 \times dh = n \times dh$
- اندازه حافظه پنهان تک V مشترک: $n \times 1 \times dh = n \times dh$
- حافظه موردنیاز برحسب تعداد عناصر:

$$2 \times n \times dh$$

میزان حافظه مورد نیاز برای KV-Cache در Multi-Query Attention n_h برابر کوچکتر از Multi-Head Attention استاندارد است.

سوال4-بخش 4

1. توضیح Grouped-Query Attention (GQA):

- MHA با ذخیره بردارهای K و V جداگانه برای هر سر، حافظه زیادی مصرف می‌کند. MQA تنها یک جفت K/V مشترک برای همه سرها ذخیره می‌کند که حافظه را به حداقل می‌رساند اما ممکن است ظرفیت مدل را کاهش دهد. GQA تلاش می‌کند تعادلی بین این دو برقرار کند.
در GQA، تعداد کل سرهای Query (n_h) به g گروه متمایز تقسیم می‌شود (به طوری که n_h بر g بخش‌پذیر باشد). تمامی سرهای Query در یک گروه، یک مجموعه مشترک از وزن‌های تبدیل K و V و بردارهای K/V ذخیره شده در حافظه پنهان مرتبط با آن گروه را به اشتراک می‌گذارند.
  - تعداد g مجموعه ماتریس تبدیل K و g مجموعه ماتریس تبدیل V وجود دارد.
  - KV-Cache برای هر لایه، g مجموعه بردار K و g مجموعه بردار V ذخیره می‌کند.
  - هر سر Query همچنان با استفاده از ماتریس تبدیل W_i^Q خاص خود، بردار Q منحصربه‌فردی را محاسبه می‌کند.
  - هنگام انجام توجه، هر سر Query از بردار Q منحصربه‌فرد خود استفاده می‌کند اما با بردارهای K و V مرتبط با گروه اختصاص داده شده به آن تعامل دارد.

2. تحلیل هزینه محاسباتی:

مراحل تولید توکن n+1:

- **Projections برای Q, K, V:**
  - محاسبه Q منحصربه‌فرد برای هر یک از سرهای nh: هزینه کل Q تقریباً برابر است با: $\mathrm{n_h} \times d \times \mathrm{d_h}$
  - محاسبه بردار K برای هر یک از گروه‌های g: هزینه کل K تقریباً برابر است با: $g \times d \times \mathrm{d_h}$
  - محاسبه بردار V برای هر یک از گروه‌های g: هزینه کل V تقریباً برابر است با: $g \times d \times \mathrm{d_h}$
  - **کل هزینه‌ها:**

$$(n_h + 2g) \times d_h \times d$$

- **Update Cache:**
  - اضافه کردن هر یک از g بردار K جدید و g بردار V جدید به ورودی‌های حافظه پنهان گروه مربوطه.
- **Attention Score:**
  - هر سر Q_i امتیازات خود را با بردارهای K متعلق به گروهش محاسبه می‌کند.
  - هزینه برای هر سر تقریباً برابر است با $(\mathrm{n} + 1) \times \mathrm{d_h}$
  - **کل هزینه:**

$$n_h \times (1 + n) \times d_h$$

- **Value Weighting:**
  - هر سر امتیازات خود را در بردارهای V متعلق به گروهش ضرب می‌کند.
  - هزینه برای هر سر تقریباً برابر است با $(n + 1) \times d_h$
  - **کل هزینه:**

$$n_h \times (n + 1) \times d_h$$

- **Output Projection:**
  - مشابه MHAوMQA: اتصال خروجی سرها و ضرب در ماتریس W_O.
  - **کل هزینه:**

$$(n_h \times d_h) \times d$$

جمع کل و تقریبی هزینه‌ها:

$$Cost = (n_h + 2_g).d_h.d + 2.n_h \times (n + 1).d_h + (n_h \times d_h).d$$

3. میزان حافظه ذخیره‌سازی مورد نیاز:

- اندازه حافظه پنهان K: $g \times n \times d_h$
- اندازه حافظه پنهان V: $g \times n \times dh$
- حافظه موردنیاز برحسب تعداد عناصر:

$$2 \times g \times n \times dh$$

میزان حافظه مورد نیاز GQA به صورت خطی با تعداد گروه‌ها (g) مقیاس می‌شود. این مقدار بین MQA (g=1) و MHA (g=n_h) قرار می‌گیرد و در مقایسه با مصرف حافظه MHA، فاکتور کاهش n_h/g را ارائه می‌دهد.

سوال4-بخش 5

حالت 1: Standard Multi Head Attention

- **Projections برای Q, K, V:**

$$n_h \times (2n + 3) \times d_h \times d$$
$$12 \times (2 \times 1024 + 3) \times 64 \times 768 = 1{,}209{,}729{,}024$$

- **Attention Score:**

$$n_h \times d_h \times (n + 1)$$
$$12 \times 64 \times (1024 + 1) = 787{,}200$$

- **Value Weighting:**

$$n_h \times (n + 1) \times d_h$$
$$12 \times (1024 + 1) \times 64 = 787{,}200$$

- **Output Projection:**

$$n_h \times d_h \times d$$
$$12 \times 64 \times 768 = 589{,}824$$

جمع کل و تقریبی هزینه‌ها:

$$1{,}209{,}729{,}024 + 787{,}200 + 787{,}200 + 589{,}824 = 1{,}211{,}893{,}248\ FLOPs$$

حالت 2: KV-Cache

- **Projections برای Q, K, V:**

$$3 \times n_h \times d_h \times d$$
$$3 \times 12 \times 64 \times 768 = 1{,}769{,}472$$

- **Attention Score**

$$n_h \times d_h \times (n + 1)$$
$$12 \times 64 \times (1024 + 1) = 787{,}200$$

- **Value Weighting:**

$$n_h \times (n + 1) \times d_h$$
$$12 \times (1024 + 1) \times 64 = 787{,}200$$

- **Output Projection:**

$$n_h \times d_h \times d$$
$$12 \times 64 \times 768 = 589{,}824$$

جمع کل و تقریبی هزینه‌ها:

$$cost = 3n_h d_h d + 2n_h(n+1)d_h + n_h d_h d$$
$$cost = 1{,}769{,}472 + 2 \times 787{,}200 + 589{,}824 = 3{,}145{,}728\ FLOPs$$

حافظه موردنیاز برحسب تعداد عناصر

$$2 \times n \times n_h \times d_h$$
$$2 \times 1024 \times 12 \times 64 = 1{,}572{,}864\ elements$$
$$1{,}572{,}864 \times 2 = 3{,}145{,}728\ Bytes$$

حالت 3: Multi Query Attention

- **Projections برای Q, K, V:**

$$(n_h + 2) \times d \times d_h$$
$$(12 + 2) \times 768 \times 64 = 688{,}128$$

- **Attention Score**

$$n_h \times (n+1) \times d_h$$
$$12 \times (1024+1) \times 64 = 787{,}200$$

- **Value Weighting:**

$$n_h \times (n+1) \times d_h$$
$$12 \times (1024+1) \times 64 = 787{,}200$$

- **Output Projection:**

$$(n_h \times d_h) \times d$$
$$(12 \times 64) \times 768 = 689{,}824$$

جمع کل و تقریبی هزینه‌ها:

$$cost = (n_h + 2).d.d_h + 2.n_h.(n+1).d_h + n_h.d_h.d$$
$$cost = (12+2) \times 768 \times 64 + 2 \times 12 \times (1024+1) \times 64 + 12 \times 64 \times 76$$
$$= 2{,}852{,}352\ FLOPs$$

حافظه موردنیاز برحسب تعداد عناصر

$$2 \times n \times dh$$
$$2 \times 1024 \times 64 = 131{,}072$$
$$131{,}072 \times 2 = 262{,}144\ Bytes$$

حالت 4: Grouped Query Attention

- **Projections برای Q, K, V:**

$$(n_h + 2g) \times d_h \times d$$
$$(12 + 2 \times 4) \times 64 \times 768 = 983{,}040$$

- **Attention Score**

$$n_h \times (\mathrm{n} + 1) \times d_h$$
$$12 \times (1024 + 1) \times 64 = 787{,}200$$

- **Value Weighting:**

$$n_h \times (n + 1) \times d_h$$
$$12 \times (1024 + 1) \times 64 = 787{,}200$$

- **Output Projection:**

$$(n_h \times d_h) \times d$$
$$(12 \times 64) \times 768 = 589{,}824$$

جمع کل و تقریبی هزینه‌ها:

$$Cost = \left(n_h + 2_g\right).d_h.d + 2.n_h \times (n + 1).d_h + (n_h \times d_h).d$$
$$Cost = 983{,}040 + 2 \times 787{,}200 + 589{,}824 = 3{,}147{,}264\ FLOPs$$

حافظه موردنیاز برحسب تعداد عناصر

$$2 \times g \times n \times dh$$
$$2 \times 4 \times 1024 \times 64 = 524{,}288$$
$$524{,}288 \times 2 = 1{,}048{,}576\ Bytes$$

Multi Head Attention Latent در DeepSeek-R1: نوآوری برای صرفه‌جویی در حافظه مکانیزم نوآورانه توجه چندسر پنهان (Multi-Head Latent Attention) در مدل DeepSeek-R1 با بهینه‌سازی ذخیره‌سازی KV cache در طول استنتاج، کاهش قابل توجهی در میزان استفاده از حافظه ایجاد می‌کند. برخلاف مکانیزم توجه چندسر سنتی (Multi-Head Attention) که ماتریس‌های Key و Value با اندازه کامل را برای هر توکن در دنباله ورودی و برای تمام سرها ذخیره می‌کند، MLA از یک تکنیک فشردگی آموخته‌شده استفاده می‌نماید.

۱. **چالش کش Key-Value:** در مدل‌های ترنسفورمر استاندارد، کش KV کلیدها و مقادیری را که برای هر توکن در هر لایه محاسبه می‌شوند، ذخیره می‌کند. در طول فرآیند تولید دنباله، اندازه این کش به طور خطی با طول دنباله رشد می‌کند. برای دنباله‌های طولانی و مدل‌های بزرگ، کش KV به یک bottleneck اصلی تبدیل شده و مقدار قابل توجهی از حافظه را مصرف می‌کند.

۲. **فشردگی آموخته‌شده در فضای پنهان:** MLA با معرفی تصویرهای خطی آموخته‌شده که بردارهای اصلی و با ابعاد بالای کلید و مقدار را فشرده کرده و به یک فضای «پنهان» با ابعاد پایین‌تر تبدیل می‌کنند، به این چالش پاسخ می‌دهد.

۳. **ذخیره‌سازی نمایش‌های فشرده‌شده:** مکانیزم توجه DeepSeek-R1 به جای ذخیره‌سازی جفت‌های Key و Value کامل و فشرده‌نشده، این نمایش‌های پنهان فشرده‌شده و به مراتب کوچک‌تر را در کش KV ذخیره می‌کند. این کار به طرز چشمگیری میزان حافظه مورد نیاز برای ذخیره زمینه تاریخی (زمینه قبلی دنباله) را کاهش می‌دهد.

۴. **بازسازی برای محاسبه توجه:** هنگامی که مدل نیاز به محاسبه توجه برای یک توکن جدید دارد، بردارهای پنهان فشرده‌شده با استفاده از ماتریس‌های آموخته‌شده دیگر به ابعاد مورد نیاز project داده می‌شوند. اگرچه این فرآیند شامل محاسبات است، اما صرفه‌جویی اصلی در حافظه ناشی از ذخیره‌سازی حالت فشرده‌شده به جای جفت‌های KV با اندازه کامل است.

با فشرده‌سازی کش KV در یک فضای پنهان با ابعاد پایین‌تر، توجه چندسر پنهان DeepSeek-R1 به طور موثری سربار حافظه مرتبط با پردازش دنباله‌های طولانی را کاهش می‌دهد و این امر امکان استنتاج سریع‌تر و توانایی مدیریت زمینه‌های (متن‌های) بزرگ‌تر را فراهم می‌آورد.

سوال 5-بخش 1

مدل BERT:

(الف) ترنسفورمر دو طرفه و Encoder-Only:

مدل BERT بر اساس معماری ترنسفورمر ساخته شده، اما تنها از بخش اِنکودر آن استفاده می‌کند. این اِنکودر به دلیل مکانیسم خودتوجهی (Self-Attention)، قادر است هنگام پردازش هر کلمه، به تمام کلمات دیگر در همان دنباله ورودی (چه قبل و چه بعد از آن کلمه) توجه کند. این قابلیت "دو طرفه" بودن، برخلاف مدل‌های قدیمی‌تر (مانند RNNها یا مدل‌های زبانی یک‌طرفه) که فقط به گذشته نگاه می‌کردند، به BERT اجازه می‌دهد تا معنای کلمات و عبارات را در یک زمینه کامل‌تر و غنی‌تر درک کند و نمایش‌های برداری (Embeddings) قدرتمندتری برای آن‌ها ایجاد کند. هدف اصلی BERT درک عمیق ورودی متنی است.

(ب)

BERT با استفاده از حجم عظیمی از داده‌های متنی بدون برچسب از طریق دو وظیفه نظارت نشده پیش‌آموزش داده می‌شود:

- **Masked Language Modeling (MLM):** در این روش، تعدادی از کلمات ورودی به طور تصادفی برای پوشانده شدن انتخاب می‌شوند. معمولاً ۱۵٪ از توکن‌ها برای این کار انتخاب می‌شوند، که از بین این ۱۵٪، ۸۰٪ با توکن ویژه [MASK] جایگزین می‌شوند، ۱۰٪ با یک کلمه تصادفی دیگر جایگزین می‌شوند، و ۱۰٪ باقی‌مانده بدون تغییر باقی می‌مانند. مدل باید با استفاده از زمینه موجود (هم کلمات قبل و هم بعد) سعی کند کلمات اصلی که پنهان یا تغییر داده شده‌اند را پیش‌بینی کند. این کار مدل را مجبور می‌کند تا نمایش‌های عمیقاً دو طرفه از متن یاد بگیرد، وابستگی‌های پیچیده بین کلمات را درک کند و در نتیجه درک عمیقی از زمینه پیدا کند
- **Next Sentence Prediction (NSP).:** در این وظیفه، مدل دو جمله A و B را به عنوان ورودی دریافت می‌کند و باید پیش‌بینی کند که آیا جمله B بلافاصله بعد از جمله A در متن اصلی آمده است یا خیر (به عنوان یک وظیفه طبقه‌بندی دوگانه). هدف از این وظیفه این بود که مدل یاد بگیرد روابط منطقی بین جملات و ساختار اسناد را درک کند، که می‌تواند برای وظایفی مانند پاسخ به سوالات یا استنتاج زبانی مفید باشد (هرچند اهمیت آن در مدل‌های بعدی کمتر شد).

(ج)

توکن [CLS] (مخفف Classification) یک توکن مخصوص است که همیشه در ابتدای هر دنباله ورودی به BERT اضافه می‌شود. پس از پردازش کامل دنباله توسط لایه‌های ترنسفورمر، بردار خروجی مربوط به این توکن در لایه نهایی، به عنوان یک نمایش فشرده و کلی از معنای کل دنباله در نظر گرفته می‌شود. هنگام تنظیم دقیق BERT برای وظایف طبقه‌بندی (مثل تعیین احساسات یک متن یا تشخیص اسپم)، این بردار [CLS] استخراج شده و به یک لایه ساده کاملاً متصل (که برای وظیفه خاص آموزش می‌بیند) داده می‌شود تا بر اساس آن، کلاس مورد نظر پیش‌بینی شود. این توکن نقطه تمرکز مدل برای تصمیم‌گیری‌های سطح جمله/سند است.

(د)

مدل استاندارد BERT، با توجه به معماری فقط اِنکودر و ماهیت دو طرفه آن، مستقیماً برای "تولید متن" (Text Generation) طراحی نشده و نمی‌تواند کلمات را به صورت ترتیبی و خودکار تولید کند. مدل‌های تولید متن معمولاً به معماری‌های دیکودر (Decoder) یا ترکیبی (Encoder-Decoder) نیاز دارند که بتوانند کلمات را گام به گام بر اساس آنچه قبلاً تولید شده پیش‌بینی کنند. BERT عمدتاً برای وظایم "درک زبان" (Language Understanding) مانند طبقه‌بندی، پاسخ به سوال یا استخراج اطلاعات کاربرد دارد.

(ه)

RoBERTa (Robustly optimized BERT approach) یک پیشرفت نسبت به BERT اصلی است که توسط فیس‌بوک معرفی شد. معماری اصلی حفظ شد، اما فرآیند پیش‌آموزش بهینه‌سازی شد. RoBERTa برای مدت طولانی‌تر، با داده‌های بیشتر (حدود ۱۰ برابر BERT) و اندازه‌های بچ بزرگتر آموزش داده شد. همچنین، وظیفه NSP از آن حذف شد و از پوشاندن دینامیک (Dynamic Masking) استفاده کرد که در آن الگوی کلمات پوشانده شده در هر مرحله از آموزش تغییر می‌کند، که منجر به یادگیری قوی‌تر می‌شود. این بهینه‌سازی‌ها باعث شد RoBERTa در بسیاری از معیارها عملکرد بهتری نسبت به BERT اصلی داشته باشد.

(و)

مدل ViT (Vision Transformer) نشان داد که معماری ترنسفورمر می‌تواند برای بینایی کامپیوتر نیز مؤثر باشد.

- **شباهت‌ها:** هر دو بر پایه معماری ترنسفورمر (با لایه‌های سلف-اَتنشن و فیدفوروارد) بنا شده‌اند. هر دو برای پردازش، ورودی را به دنباله‌ای از واحدهای کوچک ("توکن") تبدیل می‌کنند (کلمات برای BERT، تکه‌های تصویر برای ViT). هر دو از جایگذاری‌های موقعیتی برای اضافه کردن اطلاعات مکانی/ترتیبی استفاده می‌کنند و غالباً یک توکن ویژه (مشابه [CLS]) برای نمایش کلی ورودی دارند که برای وظایف طبقه‌بندی استفاده می‌شود. هر دو برای عملکرد بالا نیاز به پیش‌آموزش در مقیاس بزرگ دارند.
- **تفاوت‌ها:** اصلی‌ترین تفاوت در نوع داده ورودی است (متن در برابر تصویر) و به تبع آن، نحوه آماده‌سازی ورودی؛ BERT توکن‌های زبانی را پردازش می‌کند، در حالی که ViT تصویر را به تکه‌های کوچک تقسیم کرده و آن‌ها را مسطح کرده و جایگذاری می‌کند. وظایف پیش‌آموزش اصلی آن‌ها نیز متفاوت است (مختص زبان در برابر مختص تصویر). BERT برای وظایف NLP و ViT برای وظایف بینایی کامپیوتر کاربرد دارد.

سوال 5-بخش 2

(آ)

مدل‌های Decoder-Only مانند GPT با وظیفه پیش‌بینی توکن بعدی (NTP) آموزش می‌بینند. این وظیفه ذاتاً Autoregressive است؛ به این معنی که مدل در هر مرحله، توکن بعدی را بر اساس دنباله‌ای از توکن‌هایی که تا آن لحظه مشاهده یا تولید کرده است پیش‌بینی می‌کند. در طول آموزش، مدل با دیدن بخش‌هایی از متن و پیش‌بینی کلمه بعدی در آن دنباله، یاد می‌گیرد که چگونه دنباله‌های متنی محتمل و منسجم ایجاد کند. این فرآیند پیش‌بینی ترتیبی، همان روشی است که مدل برای تولید متن در زمان استنتاج (Inference) نیز از آن استفاده می‌کند.

(ب)

معماری Decoder-Only از یک نسخه خاص از خودتوجهی به نام "خودتوجهی ماسک شده" (Masked Self-Attention) استفاده می‌کند. این مکانیسم تضمین می‌کند که هنگام محاسبه نمایش برای یک توکن خاص، مدل فقط به توکن‌های قبلی در دنباله دسترسی دارد و از دیدن توکن‌های بعدی منع می‌شود. این "ماسک علّی" (Causal Mask) برای حفظ خاصیت Autoregressive ضروری است. زیرا در تولید متن، ما کلمات را یکی پس از دیگری تولید می‌کنیم و هر کلمه جدید فقط باید به کلمات تولید شده قبلی وابسته باشد، نه کلماتی که هنوز وجود ندارند. این مکانیسم ماسک شده به مدل اجازه می‌دهد تا برای پیش‌بینی توکن بعدی صرفاً بر اساس گذشته یاد بگیرد و تولید گام به گام را ممکن می‌سازد.

(ج)

بسیار خوب، متن شما را با اضافه کردن توضیحات مربوط به روش‌های نمونه‌برداری ویرایش می‌کنم:

تولید متن در مدل‌های Decoder-Only یک فرآیند Autoregressive و تکراری است. با دریافت یک پرامپت اولیه، مدل آن را پردازش کرده و با استفاده از مکانیسم خودتوجهی ماسک شده، احتمالات برای توکن بعدی را پیش‌بینی می‌کند. سپس، یکی از توکن‌ها (بر اساس توزیع احتمالات پیش‌بینی شده و با استفاده از روش‌های نمونه‌برداری رایج مانند Greedy Sampling که توکن با بیشترین احتمال انتخاب می‌شود، یا روش‌های متنوع‌تر مانند Top-k/Top-p Sampling که توکن‌ها را از میان مجموعه‌ای از محتمل‌ترین‌ها انتخاب می‌کنند) انتخاب شده و به انتهای دنباله فعلی اضافه می‌شود. دنباله جدید (ترکیب ورودی اولیه و توکن‌های تولید شده تا این مرحله) به عنوان ورودی در گام بعدی به مدل داده می‌شود تا توکن بعدی را پیش‌بینی کند. این چرخه تکرار می‌شود: پیش‌بینی توکن بعدی بر اساس تمام توکن‌های تولید شده قبلی، اضافه کردن آن به دنباله، و ادامه فرآیند، تا زمانی که متن کافی تولید شود یا مدل توکن پایان دنباله را پیش‌بینی کند.

(د)

همانطور که قبلاً ذکر شد، مقایسه دقیق می‌تواند پیچیده باشد. اما یکی از دلایل بالقوه این است که در تولید متن Autoregressive، در هر گام فقط یک توکن جدید تولید می‌شود و محاسبات توجه (به دلیل ماسک) به توکن‌های قبلی محدود است. در حالی که مدل‌های Encoder-Only معمولاً کل دنباله ورودی را به صورت یکجا پردازش می‌کنند تا یک نمایش متراکم تولید کنند. برای برخی وظایف خاص

تولید یا پردازش که در Decoder-Only بهینه‌تر پیاده‌سازی می‌شوند، ممکن است در زمان استنتاج کارایی بیشتری مشاهده شود، هرچند پیچیدگی کلی ترنسفورمر بالا باقی می‌ماند.

(ه)

طول کانتکست یا پنجره زمینه، حداکثر طول دنباله ایست که مدل می‌تواند در آن واحد پردازش کند. افزایش این طول به مدل اجازه می‌دهد تا اطلاعات بیشتری از ابتدای ورودی را در نظر بگیرد و در فرآیند تولید یا پیش‌بینی‌های بعدی از آن‌ها استفاده کند، که به بهبود انسجام و ارتباط متن تولید شده در مقیاس بزرگتر کمک می‌کند. با این حال، مکانیسم خودتوجهی در معماری ترنسفورمر، پیچیدگی محاسباتی و حافظه‌ای از مرتبه O(n2) دارد که در آن n طول دنباله ورودی است. این پیچیدگی درجه دوم (Quadratic) به این معنی است که با دو برابر شدن طول دنباله، محاسبات و نیاز به حافظه تقریباً چهار برابر می‌شود. به همین دلیل، مدیریت کانتکست‌های بسیار طولانی از نظر محاسباتی بسیار پرهزینه است و یکی از چالش‌های اصلی در مدل‌های زبانی بزرگ محسوب می‌شود.

(و)

مدل‌های GPT بر اساس الگوهای آماری که در داده‌های عظیم آموزشی یاد گرفته‌اند، عمل می‌کنند و در هر گام توکن بعدی را بر اساس محتمل‌ترین دنباله آماری پیش‌بینی می‌کنند. "Hallucination" یا توهم به تولید اطلاعاتی توسط مدل اشاره دارد که نادرست، ساختگی یا بی‌اساس هستند، حتی اگر به نظر منطقی برسند. دلایل این پدیده عبارتند از: یادگیری الگوهای نادرست یا سوگیری‌دار از داده‌های آموزشی، پیش‌بینی بر اساس احتمال آماری که گاهی با واقعیت منطبق نیست، تلاش برای خلق اطلاعات جدید و پر کردن جای خالی دانش با ساختن عباراتی محتمل، و محدودیت ذاتی مدل در "فهمیدن" حقیقت در مقابل یادگیری همبستگی‌های آماری صرف. مدل فاقد توانایی راستی‌آزمایی اطلاعات به معنای انسانی است و صرفاً محتمل‌ترین دنباله را بر اساس داده‌هایی که دیده، تولید می‌کند.

سوال 5-بخش3

(الف)

روش‌های PEFT برای حل مشکل تنظیم دقیق (Fine-Tuning) مدل‌های زبانی بزرگ معرفی شدند که نیاز به منابع محاسباتی و حافظه بسیار زیادی برای آموزش تمام پارامترهای مدل دارند. PEFT به جای آموزش کامل مدل، تنها بخش کوچک و افزایشی از پارامترها را آموزش می‌دهد یا پارامترهای ورودی/خروجی را تغییر می‌دهد. این کار باعث صرفه‌جویی قابل توجهی در زمان، حافظه و قدرت محاسباتی مورد نیاز برای آموزش می‌شود.

- **LoRA (Low-Rank Adaptation):** در این روش، به ماتریس‌های وزن اصلی مدل از پیش آموزش‌دیده (W_0)، یک ماتریس افزایشی low rank ($\Delta W = BA$) اضافه می‌شود که فقط ماتریس‌های کم‌ابعاد B و A (با ابعاد $d \times r$ و $r \times k$ که r، rank بسیار کمتری از k,d دارد) آموزش

داده می‌شوند. ماتریس وزن نهایی در زمان استفاده می‌شود ($W_0 + BA$). این روش تعداد پارامترهای آموزش‌دیدنی را به شدت کاهش می‌دهد.

- **Adapter-Tuning:** در این روش، لایه‌های کوچک و جدید (Adapter Modules) بین لایه‌های اصلی ترنسفورمر اضافه می‌شوند. تنها پارامترهای این لایه‌های جدید در طول تنظیم دقیق آموزش می‌بینند، در حالی که پارامترهای اصلی مدل فریز (ثابت) باقی می‌مانند.
- **Prompt-Tuning:** در این روش، پارامترهای مدل اصلی ثابت نگه داشته می‌شوند و تنها یک دنباله کوچک از بردارهای قابل آموزش (soft prompts) به ورودی مدل اضافه می‌شود. مدل یاد می‌گیرد چگونه این پرامپت‌های نرم را برای هدایت خروجی به سمت وظیفه مورد نظر استفاده کند.

(ب)

در روش LoRA، ماتریس A معمولاً با مقادیر تصادفی کوچک و ماتریس B با صفرهای کامل مقداردهی اولیه می‌شوند. دلیل مقداردهی اولیه صفر برای B این است که در ابتدای فرآیند تنظیم دقیق، ماتریس افزایشی $\Delta W = BA$ یک ماتریس صفر باشد. این کار تضمین می‌کند که در همان گام اول آموزش، تغییرات ایجاد شده در ماتریس وزن اصلی ($W_0 + BA$) صفر باشد و عملکرد مدل با شروع آموزش به صورت ناگهانی و شدید تغییر نکند. این کار باعث می‌شود آموزش با یک نقطه شروع پایدار آغاز شود و از بی‌ثباتی در مراحل اولیه جلوگیری می‌کند.

(ج)

فرض کنیم ابعاد embedding برابر d

و رتبه LoRA برابر r باشد. در یک بلوک Encoder، ما به وزن‌های لایه‌های Attention و Feed-Forward نگاه می‌کنیم:

لایه Attention:

شامل ماتریس‌های پروجکشن برای Q، K، V برای هر هد و ماتریس پروجکشن خروجی است. برای هر هد، ماتریس‌های پروجکشن Q، K، V از ابعاد d به $\frac{d}{h}$ می‌روند. اعمال LoRA با رتبه r به یک ماتریس با ابعاد $d \times (\frac{d}{h})$، پارامترهای آموزش‌دیدنی جدیدی به تعداد $dr + \frac{dr}{h}$اضافه می‌کند. در لایه Attention، تعداد کل پارامترهای LoRA (با در نظر گرفتن h هد برای Q, K, V و ماتریس خروجی نهایی با ابعاد $d\ x\ d$) می‌شود $3hdr + 5dr$.

لایه Feed-Forward:

شامل دو لایه خطی است. لایه اول از ابعاد d به $4d$

و لایه دوم از $4d$ به d می‌رود. اعمال LoRA با رتبه r:

برای لایه اول : پارامترهای LoRA می‌شود $5dr$.

برای لایه دوم (): پارامترهای LoRA می‌شود $5dr$. کل پارامترهای LoRA در Feed-Forward می‌شود $10dr$.

تعداد کل پارامترهای آموزش‌دیدنی با LoRA (بدون Bias) در این بلوک Encoder برابر است با:

$$\boldsymbol{3hdr + 15dr}$$

(د)

در اینجا چهار نمونه از روش‌هایی که LoRA را بهبود داده‌اند یا تغییر داده‌اند، آورده شده است:

1. **QLoRA:** این روش LoRA را با کوانتیزاسیون (کمی‌سازی) ترکیب می‌کند. وزن‌های اصلی مدل پیش‌آموزش‌دیده به دقت پایین‌تری (مثلاً 4 بیت) کوانتیزه می‌شوند و تنها ماتریس‌های کم‌رتبه LoRA آموزش داده می‌شوند. تفاوت اصلی آن در استفاده از کوانتیزاسیون برای کاهش شدید مصرف حافظه مدل پایه است.
2. **LoRA+:** این افزونه پیشنهاد می‌کند که نرخ یادگیری (Learning Rate) برای ماتریس‌های A و B در LoRA باید متفاوت باشد. معمولاً نرخ یادگیری بالاتری برای ماتریس B (که با صفر مقداردهی شده) نسبت به ماتریس A استفاده می‌شود تا به بهبود عملکرد در برخی وظایف کمک کند.
3. **DyLoRA (Dynamic LoRA):** در این روش، رتبه r برای ماتریس‌های LoRA در طول فرآیند آموزش به صورت پویا تنظیم می‌شود، به جای اینکه ثابت بماند. این کار به مدل اجازه می‌دهد با رتبه‌های کوچکتر شروع کرده و در صورت لزوم آن را افزایش دهد تا تعادلی بین کارایی و دقت برقرار شود.
4. **LongLoRA:** این روش LoRA را با تکنیک‌هایی برای مدیریت و گسترش پنجره زمینه (Context Window) ترکیب می‌کند، مانند استفاده از مکانیسم توجه پراکنده (Sparse Attention). تفاوت آن در تمرکز بر حفظ کارایی LoRA در حالی که مدل قادر به پردازش دنباله‌های ورودی بسیار طولانی‌تر باشد.

الف)

در یک لایه خود-توجه چندسر با N_h سر، پارامترهای قابل آموزش شامل وزن‌ها و بایاس‌های مربوط به تبدیلات خطی برای تولید بردارهای پرسش (Q)، کلید (K)، و مقدار (V) در هر سر توجه، و نیز وزن‌ها و بایاس‌های تبدیل خطی نهایی خروجی پس از الحاق سرها هستند. تعداد کل این پارامترها برابر است با:

$$2N_h D_{\mathrm{i}} \mathrm{D}_{\mathrm{hid}} + 2N_h \mathrm{D}_{\mathrm{hid}} + N_h D_o D_i + N_h d_v + D_{\mathrm{out}} N_h D_o + D_o ut$$

ب)

برای بازنویسی معادله ۱ (خود-توجه) و معادله ۳ (خود-توجه چندسر) برای یک تک پیکسل کوئری که آن را با بردار z_qنمایش می‌دهیم، مراحل به صورت زیر خواهد بود. فرض می‌کنیم که Z ماتریس کامل ورودی است که شامل z_q به عنوان یکی از ردیف‌های خود است.

معادله ۱ بازنویسی شده برای یک کوئری z_q (خود-توجه تک سر):

در این حالت، بردار پرسش از z_q و ماتریس‌های کلید و مقدار از کل ورودی Z مشتق می‌شوند. خروجی خود-توجه برای کوئری تکی z_q (که یک بردار با ابعاد $Di \times 1$ است) به صورت زیر محاسبه می‌شود:

ابتدا بردار پرسش q را برای z_q محاسبه می‌کنیم:

$$q = z_q W_Q$$

سپس ماتریس امتیازات توجه بین q و تمام کلیدها K (مشتق شده از Z) را محاسبه می‌کنیم:

$$S_q = qK^T = z_q W_Q (ZW_K)^T$$

با اعمال softmax پس از مقیاس‌بندی، وزن‌های توجه برای z_q نسبت به تمام ورودی‌ها به دست می‌آید:

$$\alpha_q = \mathrm{softmax}\left(\frac{S_q}{\sqrt{d_k}}\right) = \mathrm{softmax}\left(\frac{z_q W_Q (ZW_K)^T}{\sqrt{d_k}}\right)$$

در نهایت، خروجی خود-توجه برای کوئری z_q به صورت مجموع وزن‌دار بردارهای مقدار (مشتق شده از Z) محاسبه می‌شود:

$$o_q = \alpha_q V = \mathrm{softmax}\left(\frac{z_q W_Q (ZW_K)^T}{\sqrt{d_k}}\right) ZW_V$$

خروجی o_q برداری با ابعاد $Do \times 1$ است.

معادله ۳ بازنویسی شده برای یک کوئری z_q (خود-توجه چندسر):

برای خود-توجه چندسر، همین فرآیند برای هر سر توجه تکرار می‌شود. خروجی هر سر h برای کوئری تکی z_q به صورت زیر است:

$$\mathrm{head}_h(z_q, Z) = \mathrm{softmax}\left(\frac{z_q W_{Q_h} \left(ZW_{K_h}\right)^T}{\sqrt{d_k}}\right) ZW_{V_h}$$

در اینجا، $W_{Q\mathrm{h}}, W_{K\mathrm{h}}, W_{V\mathrm{h}}$ ماتریس‌های وزن مخصوص سر h هستند. سپس خروجی تمام N_h سر برای کوئری z_q به هم الحاق می‌شوند:

$$\text{Concat}\left(\text{head}_1(z_q, Z), \ldots, \text{head}_{N_h}(z_q, Z)\right)$$

این بردار الحاق شده ابعادی برابر با $(N_h D_o) \times 1$ دارد. در نهایت، یک تبدیل خطی نهایی شامل ماتریس وزن W_O و بردار بایاس b_Oبر این بردار الحاق شده اعمال می‌شود تا خروجی نهایی خود-توجه چندسر برای کوئری z_q به دست آید:

$$multihead\_o_q = \text{Concat}\left(\text{head}_1(z_q, Z), \ldots, \text{head}_{N_h}(z_q, Z)\right) W_o + b_o$$

ج)

برای ساده‌سازی عبارت $A_{[q,k]}^{[relative]}$ در معادله ۶، عبارت‌های داده شده در معادله ۷ را جایگزین می‌کنیم.

$$A_{q,k}^{relative} = v^T \widetilde{W_k} r_\delta$$

$$r_\delta = \begin{pmatrix} ||\delta|| \\ \delta_1 \\ \delta_2 \end{pmatrix}$$

سایر عبارت‌های داده شده در معادله ۷ که در این بخش معرفی شده‌اند عبارتند از تعریف بردار $v(h)$ و مقادیر مشخص برای WQ و WK:

$$v^{(h)} = -\alpha^{(h)} \begin{pmatrix} 1 \\ -2\Delta_1^{(h)} \\ -2\Delta_2^{(h)} \end{pmatrix}$$

$$A_{q,k}^{relative} = v^T \widetilde{W_k} r_\delta = -\alpha^{(h)} (||\delta|| - \delta_1 2\Delta_1^{(h)} - \delta_2 2\Delta_2^{(h)}$$

د)

در یک لایه خود-توجه تک‌سر استاندارد (با در نظر گرفتن رمزگذاری موقعیتی مطلق)، هزینه محاسباتی غالب مربوط به ضرب‌های ماتریسی اصلی برای محاسبه پرسش‌ها، کلیدها، مقادیر، امتیازات توجه و خروجی نهایی است. هزینه‌ها به ترتیب از $Q = ZW_Q, K = ZW_K, V = ZW_V, QK^T$ و α_V به دست می‌آیند. با استفاده از ابعاد جدید، هزینه محاسباتی غالب کلی به صورت زیر است:

$$O(ND_iD_{hid} + ND_iD_{hid} + ND_iD_o + N^2D_{hid} + N^2D_o)$$

هزینه محاسباتی یک لایه توجه تک‌سر با استفاده از رمزگذاری گاوسی:

برای رمزگذاری گاوسی، هزینه محاسباتی شامل محاسبه $V = ZW_V$، محاسبه ماتریس امتیاز توجه (که شامل ترم محتوا ZZ^Tو ترم‌های نسبی است)، اعمال softmax، و ضرب وزن‌های توجه در V است. با توجه به اینکه محاسبه ترم محتوا ZZ^T هزینه $O(N^2D_i)$ دارد و ترم‌های نسبی هزینه $O(N^2D_p)$ دارند (که با توجه به $D_i \gg D_p$ کمتر است)، قسمت غالب در محاسبه امتیاز توجه $O(N^2D_i)$ است. هزینه‌های سایر عملیات‌های غالب در این لایه نیز در نظر گرفته می‌شود. هزینه محاسباتی غالب کلی لایه به صورت زیر است:

$$O(ND_iD_o + N^2D_i + N^2D_o)$$

برای مقایسه هزینه‌های محاسباتی غالب بین دو حالت، عبارت‌های زیر را در نظر می‌گیریم:

برای رمزگذاری مطلق:

$$O(ND_iD_{hid} + ND_iD_{hid} + ND_iD_o + N^2D_{hid} + N^2D_o)$$

برای رمزگذاری گاوسی:

$$O(ND_iD_o + N^2D_i + N^2D_o)$$

با توجه به شرایط داده شده که $D_i = D_{\text{hid}} = D_o$ و $Di \gg Dp$است، هر دو عبارت غالب بالا به صورت تقریبی به یک شکل ساده می‌شوند. با جایگزینی $D_i = D_hid = D_o = D$:

هزینه مطلق ≈$O(ND^2 + ND^2 + ND^2 + N^2D + N^2D) = O(ND^2 + N^2D)$

هزینه گاوسی ≈$O\left(ND^2 + N^2D + N^2D_p + N^2 + N^2D\right) = O(ND^2 + N^2D)$ (با صرف نظر از ترم‌های کمتر غالب با توجه به$D \gg D_p$).

تحت شرایط داده شده، هزینه‌های محاسباتی غالب برای یک لایه توجه تک‌سر با استفاده از رمزگذاری مطلق و با استفاده از رمزگذاری گاوسی تقریباً یکسان و از مرتبه $O\left(ND_i^2 + N^2D_i\right)$ هستند.