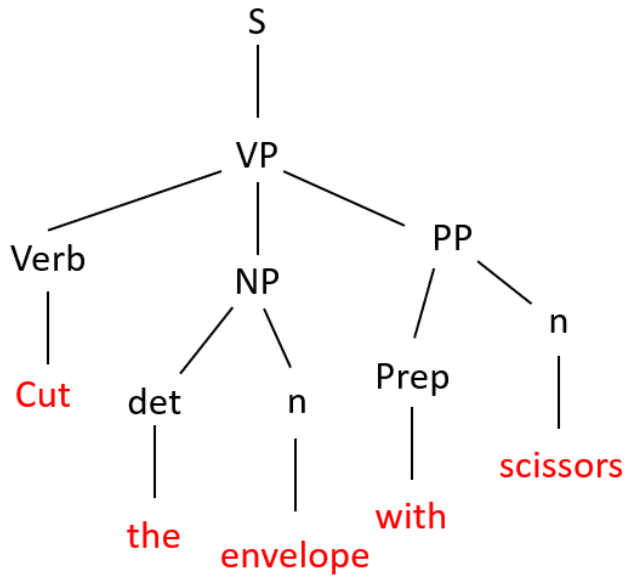
تمرین ششم NLP

سوال 1

A) پارسرهای PCFG (Probabilistic Context-Free Grammar) برای مدیریت ابهام در تجزیه جملات طراحی شده اند و با اختصاص احتمال به اجزای مختلف و درختان تجزیه بر اساس احتمال وقوع آن ها، این ابهام را مدیریت می کنند.

هنگام مواجهه با چندین تجزیه ممکن برای یک جمله، تجزیه گر هر گزینه را ارزیابی کرده و احتمال های تمام تفسیرهای ممکن را تعیین می کند. سپس گزینه ای را که بالاترین احتمال کلی را دارد به عنوان محتمل ترین نمایندگی از ساختار جمله انتخاب می کند.

B)



ساختار جمله با بالاترین احتمال در بالا نشان داده شده است.

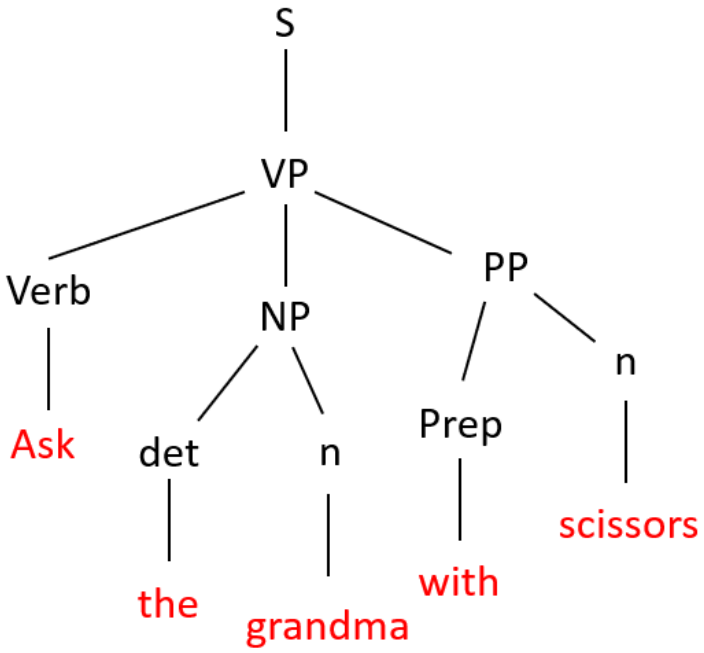
درخت تجزیه حاصل برابر است با :

$1.0 * 0.3 * 0.7 * 1.0 * (0.1)^5 = 0.0000021$

اگر از قاعده [VP → Verb NP] به جای [VP → Verb NP PP] استفاده کنیم یک درخت دیگر ایجاد می شود که احتمال آن کمتر است:

$1.0 * 0.7 * 0.3 * 0.7 * 1.0 * (0.1)^5 = 0.00000147$

C) درخت این جمله شبیه بخش B است که به شکل زیر است:



درخت تجزیه حاصل برابر است با :

$1.0 * 0.3 * 0.7 * 1.0 * (0.1)^5 = 0.0000021$

از نظر معنایی، «with scissors» باید به عبارت اسمی متصل شود، بنابراین درخت تجزیه به دست آمده معقول نیست.

D) به منظور کاهش مشکل داده های پراکنده ما با توجه به کلمه اصلی در سمت چپ هر قاعده تولید، به جای تمام غیرترمینال ها در هر قاعده تولید، واژه گزینی خواهیم کرد. به ویژه، قواعدی که از VPگسترش می یابند باید به صورت زیر اصلاح شوند:

production rule	probability
VP $(x) \rightarrow$ Verb NP	(p_x)
VP $(x) \rightarrow$ Verb NP PP	(q_x)

where
$x \in \{$ Cut, Ask, Find, ... $\}$,
$p_x \stackrel{\text{def}}{=}$ P(VP $(x) \rightarrow$ Verb NP | VP, x),
$q_x \stackrel{\text{def}}{=}$ P(VP $(x) \rightarrow$ Verb NP PP | VP, x),
and $p_x + q_x = 1$.

زیرا ما محدودیتی برای lexicalization به سر واژه های سمت راست قوانین قائل نشدیم، پیشنهاد lexicalized PCFGs به روش های مختلف اشکالی ندارد. به ویژه نیازی نیست که بر روی سر واژه شرط بگذارید، می توانید بر روی ترکیب کامل واژه ها برای تمام غیرترمینال ها شرط بگذارید، به شرطی که مشخص کنید بر روی چه چیزی شرط می گذارید، هرچند این کار بسیار کمتر عملی خواهد بود.

E) کلمه اصلی برای هر گره دیگری در درخت تجزیه به جز گره کلمه آخر در هر دو جمله یکسان است. بنابراین، احتمال شرطی هر گره برای گسترش قاعده خاص در هر دو جمله یکسان است به جز گره مربوط به آخرین کلمه. با این حال، آخرین کلمه در هر دو جمله داده شده کنترل نمی کند که کدام قانون گسترش بر روی گره های اجداد استفاده شود.

بنابراین، همان درخت تجزیه دقیقا توسط PCFG ها انتخاب خواهد شد، حتی اگر عبارت حرف اضافه در جمله اول باید به عبارت اسم وصل شود و عبارت حرف اضافه در جمله دوم باید به عبارت فعل وصل شود. (اگرچه انجام آن به روش دیگر غیرممکن نیست، اما کمتر منطقی به نظر می رسد)

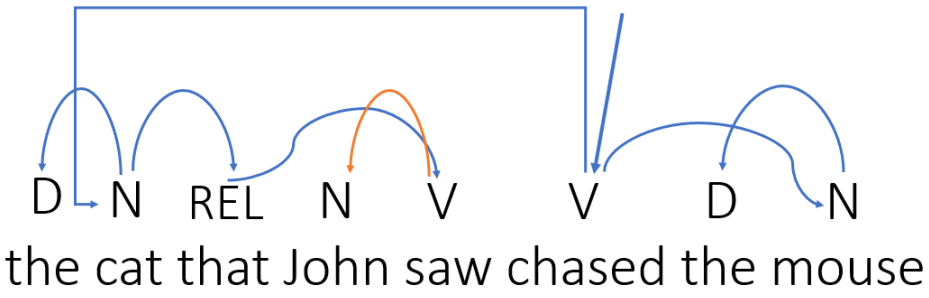
اینکه کدام مورد انتخاب شود بستگی به ارزش واقعی P(VP (Find) → Verb NP | VP, Find) و P(VP (Find) → Verb NP PP | VP, Find) دارد. به طور خلاصه، lexicalization کلمه اصلی تمام ابهامات را حل نمی کند، همانطور که در جملات داده شده نشان داده شده است.

سوال 2

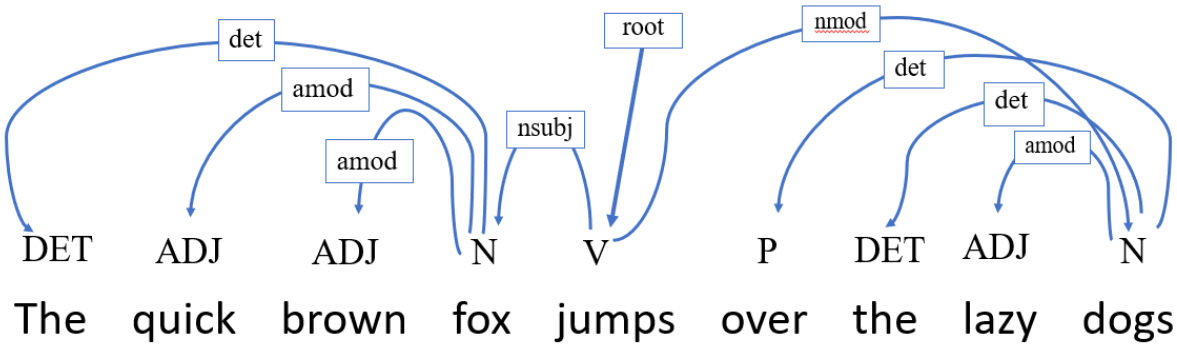
A) Head ها

- the SBAR → that
- the NP “The cat that John saw” → cat
- the topmost S → chased
- the VP “chased the mouse” → chased

B) درخت وابستگی حاصل:



سوال 3



سوال 4

جمله: مرد پسر را با دوربین دوچشمی دید.

The man saw the boy with the binoculars.

ابهام: عبارت "with the binoculars" می تواند به دو عنصر مختلف در جمله وابسته باشد و منجر به دو تفسیر متفاوت شود:

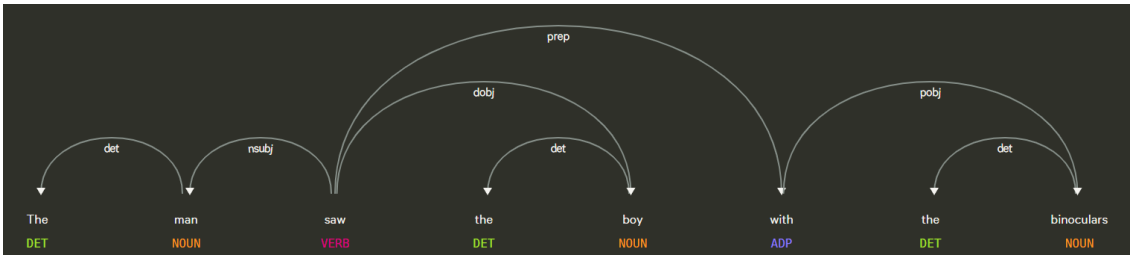
"with the binoculars" به فعل "saw" وابسته است. در این تفسیر، binoculars ابزاری است که مرد برای دیدن پسر از آن استفاده می کند.
معنی: مرد از دوربین دوچشمی برای دیدن پسر استفاده کرد.

The man used binoculars to see the boy.

"With the binoculars" به اسم "the boy" وابسته است. در این تفسیر، دوربین دوچشمی با پسر ارتباط دارد.
معنی: پسر دوربین دوچشمی داشت و مرد او را دید.

The boy had the binoculars, and the man saw him.

تحلیل وابستگی:
در تفسیر اول، عبارت حرف اضافه به فعل "saw" متصل می شود و رابطه verb-prepositional را تشکیل می دهد.
در تفسیر دوم، عبارت حرف اضافه به اسم "the boy" متصل می شود و رابطه noun-prepositional را تشکیل می دهد.



همانطور که مشخص است پارسر های آنلاین تفسیر اول را برای جمله ما انتخاب کرد.

سوال 5

1. Positional Encoding

1.1 Sine and Cosine Angles

1) هدف از محاسبه زوایا با استفاده از سینوس و کسینوس برای positional encoding در معماری ترنسفورمر فراهم کردن وسیله ای برای گنجاندن اطلاعات مربوط به موقعیت هر توکن در یک دنباله است. برخلاف شبکه های عصبی بازگشتی (RNNs) که داده ها را به صورت ترتیبی پردازش کرده و به طور ذاتی ترتیب را حفظ می کنند ترنسفورمرها همه توکن ها را به طور همزمان پردازش می کنند که می تواند منجر به از دست رفتن اطلاعات موقعیتی شود. با استفاده از توابع سینوس و کسینوس positional encoding نمایه های منحصر به فردی برای هر موقعیت ایجاد می کنند که می توانند به جاسازی های توکن اضافه شوند بدون اینکه ارزش های آنها به طور قابل توجهی تحریف شوند. این به مدل اجازه می دهد تا موقعیت های نسبی توکن ها را درک کند، که به بهبود مکانیزم های توجه و افزایش عملکرد کلی در وظایف مبتنی بر توالی کمک می کند.

2) نمایش اطلاعات موقعیتی با استفاده از توابع سینوس و کسینوس در ترنسفورمرها برای حفظ ترتیب توالی های ورودی بسیار مهم است. از آنجا که ترنسفورمرها همه توکن ها را به طور همزمان پردازش می کنند اطلاعات ترتیبی ذاتی را ندارند که برای درک روابط در داده ها حیاتی است. با استفاده از توابع سینوس و کسینوس برای positional encoding هر موقعیت در توالی به طور منحصر به فردی نمایان می شود و به مدل اجازه می دهد تا موقعیت های نسبی توکن ها را تشخیص دهد.در نتیجه، این کدگذاری به مکانیسم های توجه ترنسفورمر اجازه می دهد تا به طور موثری از موقعیت توکن ها استفاده کنند و توانایی آن را در مدل سازی وابستگی های توالی در وظایفی مانند ترجمه زبان و تولید متن افزایش می دهد.

2.1 Sine and Cosine Positional Encodings

1) تابع positional_encoding در معماری Transformer از زوایای محاسبه شده برای تولید کدگذاری های موقعیتی با پیروی از یک رویکرد سیستماتیک استفاده می کند. اولا زاویه ها را برای هر موقعیت با استفاده از تابع get_angles محاسبه می کند، که عبارت داخلی معادلات سینوس و کسینوس را بر اساس موقعیت و بعد محاسبه می کند. زاویه ها از فرمول زیر به دست می آیند:

$$PE_{(pos,2i)} = sin\left(\frac{pos}{10000^{\frac{2i}{d}}}\right)$$

$$PE_{(pos,2i+1)} = cos\left(\frac{pos}{10000^{\frac{2i}{d}}}\right)$$

جایی که pos موقعیت توکن و i ایندکس بعد و d اندازه کل بعد است. پس از به دست آوردن این زوایا، تابع سینوس را به ایندکس های زوج (2i) و تابع کسینوس را به ایندکس های فرد (2i+1) ماتریس زاویه اعمال می کند. این منجر به ایجاد یک ماتریس می شود که در آن هر موقعیت ترکیب منحصر به فردی از مقادیر سینوس و کسینوس دارد که موقعیت نسبی آن را در توالی ثبت می کند. در نهایت این ماتریس تغییر شکل داده شده و به عنوان یک تنسور که نمایانگر کدگذاری های موقعیتی است، بازگردانده می شود که می تواند به جاسازی های توکن اضافه شود تا مدل را از موقعیت های آن ها در یک توالی مطلع کند.

2) شکل ماتریس positional encoding در این معماری (1 , positions , d) است، جایی که "positions" حداکثر تعداد موقعیت هایی را که باید کدگذاری شوند نشان می دهد و d بعد کدگذاری ها است. این ماتریس با اعمال توابع سینوس و کسینوس به زوایای محاسبه شده برای هر موقعیت تولید می شود که منجر به کدگذاری های منحصر به فردی می شود که اطلاعات موقعیتی توکن ها در یک توالی را ضبط می کند.

ادغام positional encoding با ورودی های embed شده با افزودن ماتریس positional encoding به ورودی های embed شده انجام می شود. این جمع به مدل اجازه می دهد تا هم اطلاعات معنایی از embedding و هم موقعیت نسبی هر توکن لازم برای درک توالی را حفظ کند. از آنجا که مقادیر توابع سینوس و کسینوس بین -1 و 1 متغیر هستند، این جمع به طور قابل توجهی embedding های اصلی را تحریف نمی کند و یادگیری موثر روابط درون داده ها را در حالی که ترتیب آن ها حفظ می شود، امکان پذیر می سازد.

3) استفاده از positional encoding با استفاده از توابع سینوس و کسینوس به جای یادگرفتن positional encoding در معماری ترنسفورمر مزایای متعددی دارد که به شرح زیر است:

نمایش ثابت: کدگذاری های سینوسی یک نمایش قطعی و پیوسته از موقعیت ها فراهم می کنند که به مدل اجازه می دهد تا به توالی های طولانی تر از آنچه در طول آموزش دیده شده، تعمیم یابد ولی در embedding های یادگرفته شده خاص داده های آموزشی هستند و ممکن است به خوبی تعمیم نیابند.
Smoothness and Periodicity: توابع سینوس و کسینوس انتقالات نرمی بین کدگذاری های موقعیتی ایجاد می کنند که توانایی مدل را در درک فاصله های نسبی بین توکن ها آسان می کند. این تناوب به مدل کمک می کند تا الگوها را در دنباله ها به طور موثری شناسایی کند.

کاهش پیچیدگی: کدگذاری های سینوسی نیاز به پارامترهای اضافی مرتبط با embedding های یادگرفته شده را از بین می برند و پیچیدگی و نیازهای حافظه مدل را کاهش می دهند. این می تواند به زمان های آموزش سریع تر و کاهش خطر overfitting منجر شود.

قابلیت تفسیر بهبود یافته: ماهیت ریاضی توابع سینوسی تحلیل اینکه چگونه اطلاعات موقعیتی در مدل ادغام می شود را آسان تر می کند و بینش هایی در مورد نحوه پردازش دنباله ها توسط مدل ارائه می دهد.

2. Masking

Padding Mask 1.2

1) در معماری ترنسفورمر padding mask نقش حیاتی دارد که کارش این است که مدل به توکن های پدینگ که به توالی های ورودی برای یکنواختی در پردازش دسته ای اضافه می شوند، توجه نکند. توکن های پدینگ هیچ معنایی ندارند و اگر در محاسبات توجه در نظر گرفته شوند، می توانند مدل را گمراه کنند. padding mask یک ماتریس باینری است که نشان می دهد کدام توکن ها داده های واقعی هستند و کدام پدینگ هستند. هنگام محاسبه توجه ماسک برای بی اثر کردن تاثیر این موقعیت های پد شده اعمال می شود، به طوری که نمرات توجه آن ها به طور موثر به صفر تنظیم می شود.

این به مدل اجازه می دهد تا تنها بر روی توکن های معنادار تمرکز کند و توانایی آن در یادگیری و درک روابط درون داده ها را بهبود بخشد. با استفاده از padding mask ترنسفورمرها می توانند به طور کارآمد توالی های با طول متغیر را مدیریت کنند در حالی که دقت و عملکرد را در وظایف مختلفی مانند ترجمه زبان و تولید متن حفظ می کنند.

2) padding mask در معماری ترنسفورمر با شناسایی اینکه کدام توکن ها در توالی ورودی داده واقعی هستند و کدام توکن های پدینگ هستند، ساخته می شود. به طور معمول، این کار با ایجاد یک ماتریس باینری انجام می شود که هر ورودی آن به یک توکن در توالی ورودی مربوط می شود: مقدار ۱ نشان دهنده این است که توکن داده معتبر است، در حالی که مقدار ۰ نشان دهنده یک توکن پرکننده است. این ماتریس در مرحله پیش پردازش داده های ورودی قبل از اینکه به مدل تغذیه شود، تولید می شود.

padding mask بر محاسبات توجه درون ترنسفورمر تأثیر می گذارد. هنگام محاسبه امتیاز های توجه ماسک اطمینان می دهد که هر توجهی که به توکن های پدینگ داده می شود به طور موثری نادیده گرفته می شود و از تاثیرگذاری آن ها بر یادگیری مدل جلوگیری می کند. این بدان معناست که تنها توکن های معتبر در وزن های توجه مشارکت دارند، که به مدل اجازه می دهد بر روی ورودی های معنادار تمرکز کند و پدینگ غیرمفید را نادیده بگیرد. در نتیجه استفاده از ماسک پدینگ توانایی مدل را برای یادگیری از توالی های با طول های مختلف بدون فریب خوردن توسط توکن های پدینگ نامربوط افزایش می دهد.

Look-ahead Mask 2.2

1) ماسک look-ahead در دیکودر معماری ترنسفورمر در طول آموزش ضروری است، زیرا از نشت اطلاعات از توکن های آینده به توکن های فعلی جلوگیری می کند. این مکانیزم ماسک گذاری اطمینان می دهد که هنگام پیش بینی توکن بعدی در یک توالی، مدل فقط به توکن هایی که قبلا تولید شده اند دسترسی دارد و به طور مؤثری خاصیت autoregressive مدل را حفظ می کند. ماسک look-ahead معمولا به صورت یک ماتریس مثلثی پیاده سازی می شود که در آن موقعیت های مربوط به توکن های آینده به صفر تنظیم می شوند، در حالی که موقعیت های مربوط به توکن های جاری و گذشته به یک تنظیم می شوند. با اعمال این ماسک در حین محاسبه توجه، دیکودر از توجه به توکن های آینده محدود می شود و بدین ترتیب پیش بینی ها برای یک توکن خاص تنها به توکن های قبلا تولید شده وابسته خواهند بود. این رویکرد سناریوهای دنیای واقعی را تقلید می کند که در آن اطلاعات آینده در دسترس نیست، و به مدل اجازه می دهد تا یاد بگیرد که توالی ها را به صورت منسجم و متناسب با زمینه تولید کند.

2) ماسک look-ahead در معماری ترنسفورمر به صورت یک ماتریس مثلثی ایجاد شده است، جایی که هر موقعیت در توالی دارای یک ورودی متناظر است که نشان می دهد آیا می تواند به موقعیت های دیگر توجه کند یا خیر. به طور خاص، برای یک موقعیت مشخص i ماسک اجازه می دهد که توجه به موقعیت های 0 تا i انجام شود و از توجه به موقعیت های بزرگتر از i جلوگیری می کند. این منجر به ایجاد یک ماتریس می شود که در آن مقادیر بالای قطر اصلی به منفی بی نهایت یا صفر پس از اعمال تابع softmax تنظیم می شوند و به طور موثری توکن های آینده را ماسک می کند.

در حین محاسبات self-attention در دیکودر، این ماسک look-ahead به امتیاز های توجه قبل از عملیات سافت مکس اعمال می شود. هنگام محاسبه وزن های توجه، مدل نمرات توجه را در ماسک look-ahead ضرب می کند. این اطمینان می دهد که هر نمره ای که به یک توکن آینده مربوط می شود به طور موثری نادیده گرفته می شود و از نشت اطلاعات از توکن های آینده به پیش بینی توکن فعلی جلوگیری می کند. در نتیجه، پیش بینی هر توکن تنها بر اساس توکن های تولید شده قبلی استوار است.

3. Self-Attention

1) هدف از تابع scaled_dot_product_attention در معماری ترنسفورمر محاسبه نمرات توجه بین مجموعه ای از بردارهای ورودی است که به مدل اجازه می دهد هنگام تولید خروجی ها بر بخش های مرتبط ورودی تمرکز کند. این تابع عملیات های کلیدی زیر را انجام می دهد:

محاسبه ضرب نقطه ای: این محاسبه ضرب نقطه ای بردارهای query را با بردارهای key محاسبه می کند تا تعیین کند هر توکن باید بر اساس ارتباطش با سایر توکن ها چقدر توجه دریافت کند.

مقیاس بندی: حاصل ضرب های نقطه ای با تقسیم آن ها بر جذر ابعاد بردارهای key مقیاس بندی می شوند. این مقیاس بندی به کاهش مشکلات مربوط به مقادیر بزرگ ضرب نقطه ای کمک می کند که می تواند منجر به گرادیان های بسیار کوچک در طول آموزش شود.

ماسک گذاری: این تابع یک ماسک (در صورت ارائه) اعمال می کند تا اطمینان حاصل شود که برخی موقعیت ها مانند توکن های padding بر نمرات توجه تاثیر نگذارند. این برای حفظ یکپارچگی جریان اطلاعات درون مدل بسیار مهم است.

نرمال سازی سافت مکس: پس از اعمال ماسک، از تابع سافت مکس برای تبدیل نمرات مقیاس بندی

شده به وزن های توجه استفاده می شود که مجموع آن ها برابر با ۱ است. این نرمال سازی امکان تفسیر احتمالاتی از میزان توجهی که هر توکن باید دریافت کند را فراهم می کند.

مجموع وزنی: در نهایت، این وزن های توجه برای محاسبه یک مجموع وزنی از بردارهای ارزش استفاده می شوند که خروجی ای تولید می کند که اطلاعات مرتبط ترین را بر اساس پرسش های ورودی منعکس می کند.

2) مقیاس گذاری ضرب نقطه ای در مکانیزم self-attention ترنسفورمرها با حل مشکل مقادیر بزرگ ضرب نقطه ای، ثبات را در طول آموزش بهبود می بخشد. هنگام محاسبه نمرات توجه حاصل ضرب های نقطه ای بردارهای query و key می توانند بسیار بزرگ شوند به ویژه با افزایش ابعاد این بردارها. این می تواند منجر به گرادیان های بسیار کوچک در طول backpropagation شود، که یادگیری موثر مدل را دشوار می کند.

3) تابع scaled_dot_product_attention دو خروجی تولید می کند:

1- وزن های توجه (Attention Weights)

2- بردار زمینه (context vector)

1- وزن های توجه: این وزن ها نشان می دهند که هر توکن در توالی ورودی باید نسبت به دیگر توکن ها چقدر توجه کند. آن ها با محاسبه حاصل ضرب نقطه ای بین بردارهای query و key، مقیاس بندی نتایج و اعمال یک ماسک (در صورت لزوم) و سپس نرمال سازی آن ها با استفاده از تابع سافت مکس محاسبه می شوند. وزن های توجه کمک می کنند تا تعیین شود کدام توکن ها برای تولید خروجی بیشترین ارتباط را دارند.

بردار زمینه: این مجموع وزنی از بردارهای ارزش است که با استفاده از وزن های توجه محاسبه شده است. وکتور زمینه نمایانگر خلاصه ای از اطلاعات مرتبط از دنباله ورودی بر اساس مکانیزم توجه است.

4. Encoder

1.4 Encoder Layer

1)

- query , key , value matrix
- Multi-Head Self-Attention Mechanism
- Add & Norm Layer (after self-attention)
- Feed-Forward Neural Network
- Residual Connections
- Add & Norm Layer (after the feed-forward network)

2) مکانیسم multi-head attention در لایه Encoder به صورت زیر پیاده سازی شده است:

تبدیل ورودی: ورودی های embed شده به سه مجموعه از بردارها تبدیل می شوند:

پرسش ها(Q)، کلیدها(K) ، و مقادیر (V) با استفاده از پیش بینی های خطی آموخته شده.

Scaled Dot-Product Attention: هر head توجه به طور مستقل نمرات توجه را با استفاده از مکانیزم توجه نقطه ای مقیاس شده محاسبه می کند، که با مقایسه query ها و key ها، ارتباط توکن ها را محاسبه می کند، مقیاس بندی، ماسک گذاری (در صورت نیاز) و نرمال سازی از طریق تابع سافت مکس را اعمال می کند.

Parallel Attention Heads: چندین attention head به طور موازی عمل می کنند که به مدل اجازه می دهد به طور همزمان بر روی بخش های مختلف توالی تمرکز کند و روابط متنوعی را شناسایی کند.

ادغام: خروجی های تمام attention head ها به یک بردار واحد ادغام می شوند.

لایه خطی نهایی: خروجی متصل شده از طریق یک لایه خطی دیگر عبور می کند تا اطلاعات همه

head ها را در یک خروجی واحد برای لایه ترکیب کند.

3) شبکه feedforward در لایه رمزگذار نمایانگر هر توکن را به طور مستقل پردازش می کند تا اطلاعات به دست آمده توسط مکانیزم توجه خودی را تصحیح و تقویت کند. این شامل دو تبدیل خطی با یک فعال سازی ReLU در میان آن ها است که امکان تبدیل های غیرخطی و یادگیری ویژگی های غنی تر را فراهم می کند. این به مدل کمک می کند تا الگوها و روابط پیچیده تری را در داده ها شناسایی کند در حالی که کارایی را حفظ می کند.

5. Decoder

Decoder Layer 5.1

1) تفاوت های کلیدی بین لایه انکودر و دیکودر در معماری ترنسفورمر در عملکردها و ساختارهای آن ها نهفته است. لایه انکودر توالی ورودی را پردازش می کند تا مجموعه ای از نمایش های پیوسته ایجاد کند و از self-attention برای فهم روابط بین تمام توکن ها بدون هیچ گونه محدودیتی استفاده می کند. در مقابل لایه دیکودر نه تنها از self-attention استفاده می کند بلکه از multi-head attention ماسک شده نیز بهره می برد تا از تاثیر توکن های آینده بر پیش بینی های توکن های فعلی جلوگیری کند و خاصیت autoregressive را حفظ کند. علاوه بر این، لایه دیکودر شامل یک مکانیزم توجه اضافی است که به خروجی انکودر توجه می کند و به آن اجازه می دهد تا از اطلاعات کدگذاری شده هنگام تولید توالی خروجی استفاده کند.

2) در لایه دیکودر ترنسفورمر، ماسک look-ahead و ماسک padding نقش های متمایز اما مکملی ایفا می کنند. ماسک look-ahead برای جلوگیری از دسترسی دیکودر به توکن های آینده در حین آموزش اعمال می شود، به طوری که پیش بینی ها برای یک توکن خاص تنها به توکن های قبلا تولید شده وابسته باشند. در همین حال، ماسک padding برای نادیده گرفتن توکن های پدینگ در توالی های ورودی استفاده می شود، به طوری که این توکن های غیرمفید بر محاسبات توجه تاثیر نگذارند. در کنار هم، این ماسک ها به دیکودر کمک می کنند تا بر روی اطلاعات مرتبط تمرکز کند و در عین حال به محدودیت های تولید متوالی پایبند باشد، که در نهایت توانایی مدل را برای تولید خروجی های منسجم و متناسب با زمینه افزایش می دهد.

3) در لایه Decoder خروجی های لایه های multi-head attention از طریق یک سری مراحل پردازش می شوند. اولا خروجی از لایه multi-head self-attention ماسک شده به ورودی اصلی خود با استفاده از یک residual connection اضافه می شود، که سپس با نرمال سازی لایه برای تثبیت و بهبود آموزش دنبال می شود. سپس این خروجی ترکیبی به یک شبکه عصبی feedforward تغذیه می شود، که شامل دو تبدیل خطی با یک تابع فعال سازی غیرخطی در میان آن ها معمولا ReLU است. خروجی از شبکه feedforward دوباره به ورودی اصلی خود اضافه می شود (یک residual connection دیگر) و نرمال سازی می شود. این فرآیند به رمزگشا اجازه می دهد تا الگوها و روابط پیچیده را به طور موثری شناسایی کند در حالی که ثبات را در طول لایه حفظ می کند.

Full Decoder 5.2

1) full decoder در معماری ترنسفورمر با ترکیب چندین لایه هر کدام شامل چندین مولفه کلیدی، توالی ورودی را پردازش می کند. هر لایه دیکودر شامل یک مکانیزم multi-head self-attention ماسک شده است که به مدل اجازه می دهد بر روی توکن های تولید شده قبلی تمرکز کند و از تاثیر توکن های آینده بر پیش بینی فعلی جلوگیری کند. این با یک مکانیزم توجه دنبال می شود که به خروجی انکودر توجه می کند و به دیکودر این امکان را می دهد که از زمینه ای که توسط انکودر فراهم شده است بهره برداری کند. پس از این لایه های توجه، یک شبکه عصبی feedforward خروجی های ترکیبی را پردازش می کند، غیرخطی بودن را معرفی کرده و نمایندگی ویژگی ها را بهبود می بخشد. هر یک از این مراحل با residual connections و نرمال سازی لایه ای همراه است تا آموزش را تثبیت کرده و عملکرد را بهبود بخشد. با انباشتن لایه های متعدد دیکودر، مدل می تواند به طور موثری روابط پیچیده را یاد بگیرد و خروجی های منسجمی بر اساس توالی ورودی تولید کند.

2) در طول آموزش، دیکودر از طریق یک مکانیزم توجه که به طور خاص برای این منظور طراحی شده است، با خروجی های انکودر تعامل می کند. پس از پردازش توالی ورودی خود با استفاده از توجه چندسر خودی ماسک شده، دیکودر از لایه توجه دومی استفاده می کند که خروجی انکودر را به عنوان کلیدها و مقادیر در حالی که از پرسش های خود استفاده می کند، می گیرد. این به دیکودر اجازه می دهد تا بر روی اطلاعات مرتبط از خروجی انکودر تمرکز کند و به طور مؤثری از زمینه ای که توسط انکودر فراهم شده است بهره برداری کند. با توجه به خروجی های انکودر، دیکودر می تواند پیش بینی هایی تولید کند که توسط کل توالی ورودی آگاه شده اند، و این توانایی را برای تولید خروجی های منسجم و مناسب از نظر زمینه افزایش می دهد.

6. Transformer

1) کلاس ترنسفورمر اجزای انکودر و دیکودر را با قرار دادن لایه های متعدد انکودر و دیکودر به صورت متوالی یکپارچه می کند و به آن ها اجازه می دهد به طور هماهنگ با یکدیگر کار کنند. رمزگذار توالی ورودی را پردازش می کند تا مجموعه ای از embedding های متنی تولید کند، که سپس به رمزگشا منتقل می شوند. رمزگشا از این embedding ها به عنوان بخشی از مکانیزم توجه خود استفاده می کند، که به آن اجازه می دهد بر بخش های مرتبط ورودی رمزگذاری شده تمرکز کند و در عین حال خروجی های قبلی خود را از طریق self-attention ماسک شده در نظر بگیرد. این تعامل ساختاریافته اطمینان می دهد که دیکودر می تواند خروجی هایی تولید کند که هم از داده های ورودی و هم از زمینه ای که توسط انکودر ایجاد شده است، مطلع باشند و این امر وظایف تولید دنباله مؤثری مانند ترجمه یا تولید متن را آسان می کند.

2) لایه اشتراکی embedding در معماری ترنسفورمر نقش مهمی در نگاشت توکن های ورودی و خروجی به یک فضای برداری پیوسته ایفا می کند، که امکان نمایش موثر کلمات یا زیرکلمات را فراهم می آورد. این لایه دو عملکرد اصلی دارد:

نمایش ورودی: این فرآیند توکن های ورودی گسسته (مانند کلمات) را به embedding های برداری متراکم تبدیل می کند که اطلاعات معنایی و روابط بین توکن ها را در بر می گیرد. این فرآیند به مدل این امکان را می دهد که با داده های عددی کار کند به جای شناسه های توکن خام.

نمایش خروجی: لایه embedding مشابهی اغلب برای خروجی در طول رمزگشایی استفاده می شود، که به مدل اجازه می دهد خروجی رمزگشا را به فضای واژگان بازگرداند تا پیش بینی ها را تولید کند. با به اشتراک گذاشتن وزن های embedding بین لایه های ورودی و خروجی، معماری تعداد پارامترها را کاهش می دهد و ثبات در نحوه نمایش توکن ها در سراسر مدل را ترویج می کند.

3) فرایند forward pass برای ترنسفورمر با پردازش متوالی ورودی از طریق اجزای انکودر و دیکودر آن پیاده سازی می شود. در ابتدا، توکن های ورودی به امبدینگ ها تبدیل می شوند که سپس با positional encodings جمع می شوند تا ترتیب توکن ها را ثبت کنند. ورودی embed شده به انکودر داده می شود، جایی که از طریق چندین لایه انکودر که شامل multi-head self-attention و

شبکه های feedforward هستند، عبور کرده و نمایش های متنی تولید می کند. دیکودر توالی هدف (که برای آموزش به سمت راست شیفت داده شده است) را دریافت کرده و آن را از طریق لایه های خود پردازش می کند، که شامل multi-head self-attention ماسک شده و مکانیزم های توجه است که بر روی خروجی های انکودر تمرکز دارند. در نهایت، خروجی از آخرین لایه دیکودر به احتمال ها برای هر توکن در واژگان با استفاده از یک لایه خطی به همراه سافت مکس تبدیل می شود که به مدل این امکان را می دهد تا پیش بینی هایی بر اساس ورودی ها و زمینه پردازش شده تولید کند.

محمد حقیقت - 403722042