



دانشگاه تربیت مدرس

دانشکده علوم ریاضی

پایان نامه دوره کارشناسی ارشد علوم کامپیوتر

روش های عمیق مبتنی بر مبدل های بینایی در
تحلیل داده های تصویری

توسط

سید محمد بادزهره

استاد راهنما

آقای دکتر منصور رزقی

پاییز ۱۴۰۳

تقدیم به

پدر بزرگوار و مادر مهربانم و برادر عزیزم
آن‌ها که از خواسته‌هایشان گذشتند، سختی‌ها را به جان خریدند و خود را سپر بلای مشکلات و
ناملایمات کردند تا من به جایگاهی که اکنون در آن ایستاده‌ام برسم.

قدردانی

از استاد کرامت‌دور، جناب آقای دکتر رزقی که بارها بنامی‌های دلسوزانه و ارزشمند خود، همواره در مسیر تحقیق این پایان‌نامه یار و راهنمای من بودند، نهایت سپاس و قدردانی را دارم.

از خانواده عزیزم که با محبت بی‌پایان، صبوری و حمایت‌های بی‌دریغ‌شان، همواره پشتیبان من در طی این مسیر سخت و پرچالش بودند، صمیمانه سپاسگزارم.

سید محمد باذخره

پاییز ۱۴۰۳

چکیدہ

بيعلعذدقفللعقفلدخقفدللقفللقفا

فهرست مطالب

د	فهرست جداول
ه	فهرست تصاویر
۱	پیش‌گفتار
۲	۱ مفاهیم اولیه
۲	۱.۱ مقدمه
۲	۱.۱.۱ آغاز هوش مصنوعی و هدف اصلی
۳	۲.۱.۱ دوره طلایی و پیشرفت‌های اولیه
۳	۳.۱.۱ انتظارات بیش از حد و ظهور عصر تاریک
۳	۴.۱.۱ عوامل اصلی عصر تاریک هوش مصنوعی
۴	۵.۱.۱ پایان عصر تاریک و بازگشت هوش مصنوعی
۵	۲.۱ انواع مدل یادگیری ماشین و شبکه‌های عصبی
۵	۱.۲.۱ یادگیری ماشین: مروری کلی
۵	۲.۲.۱ تقسیم‌بندی‌های اصلی در یادگیری ماشین
۵	۳.۲.۱ یادگیری نظارت شده (Supervised-Learning)
۷	۴.۲.۱ معرفی چند مدل از الگوریتم یادگیری کلاسیک

۵.۲.۱	ماشین بردار پشتیبان: (Support-Vector-Machine)	۸
۶.۲.۱	بیز ساده: (Naive-Bayes)	۸
۷.۲.۱	معایب:	۹
۸.۲.۱	شبکه‌های عصبی بازگشتی (RNN) و شبکه‌های حافظه بلند مدت	
۹	کوتاه مدت (LSTM)	۹
۹.۲.۱	RNN:	۱۰
۱۰.۲.۱	مزایا و معایب Rnn:	۱۰
۱۱.۲.۱	شبکه‌های حافظه بلند مدت - کوتاه مدت: (LSTM)	۱۱
۱۲.۲.۱	Lstm: ظهور	۱۲
۳.۱	اختار LSTM: نوآوری در مقایسه با RNN	۱۳
۱.۳.۱	وضعیت سلولی (Cell): (State)	۱۳
۲.۳.۱	دروازه‌ها: (Gates)	۱۳
۳.۳.۱	به‌روزرسانی وضعیت سلولی:	۱۴
۴.۳.۱	مشکلات کلی rnn و lstm و ظهور ترانسفورمرها	۱۵
۲	پیشینه پژوهش	۲۰
۱.۲	مقدمه	۲۰
۲.۲	مشکلات ترجمه ماشینی و ترانسفورمرها:	۲۰
۳.۲	ظهور ترانسفورمرها:	۲۱
۴.۲	معماری ترانسفورمرها:	۲۱
۱.۴.۲	Embedding:	۲۲
۲.۴.۲	embedding: positional	۲۳
۳.۴.۲	attention:	۲۵

۲۹	(Add): Connection Residual	۴.۴.۲
۲۹	مزایای Connection Residual در ترانسفورمر	۵.۴.۲
۳۰	(Norm): Normalization Layer	۵.۲
۳۳	decoder:	۶.۲
۳۳	attention head multi masked	۷.۲
۳۴	attention: mask	۸.۲
۳۵	transformer: vision	۹.۲
۳۶	transformer: vision in embedding patch	۱.۹.۲
۳۶	شکل پیچ ها:	۲.۹.۲
۳۸	تعداد پیچ ها:	۳.۹.۲
۳۹	بردار کردن هر پیچ	۴.۹.۲
۳۹	اعمال لایه خطی: (Projection)	۱۰.۲
۴۱	Token: CLS	۱۰.۱۰.۲
۴۳	Transformer: Swin	۱۱.۲

۴۴	روش های پیشنهادی	۳
----	------------------	---

۴۵	آزمایشات و نتایج	۴
----	------------------	---

۴۶	کتاب نامه	
----	-----------	--

۴۷	جزئیات مدل ها و جدول پارامترها	آ
----	--------------------------------	---

فهرست جداول

فهرست تصاویر

۲۲	۲.۴.۱ معماری ترانسفورمرها
۲۳	embedding word ۲.۴.۲
۲۴	embedding word ۲.۴.۳
۲۵	embedding word ۲.۴.۴
۲۷	Attention ۲.۴.۵
۲۸	attention head multi ۲.۴.۶
۳۴	Decoder ۲.۶.۷
۳۶	patch to iamge ۲.۹.۸
۳۷	Image original ۲.۹.۹
۳۸	Image patch ۲.۹.۱۰
۴۰	Transformer Vision in Embedding ۱۰.۱۱
۴۲	Transformer Vision in Token Cls ۱۰.۱۲

پیش گفتار

قدشتمقدکنقصدبثقلدقفخدلqxفادخفادخ

فصل ۱

مفاهیم اولیه

در این فصل به معرفی مقدمات و مفاهیم مورد نیاز در این پایان نامه می پردازیم.

۱.۱ مقدمه

در این بخش به تاریخچه هوش مصنوعی، دستاورد های اولیه، چالش ها، دلایل رکود هوش مصنوعی و پایان عصر تاریک هوش مصنوعی صحبت میکنیم

۱.۱.۱ آغاز هوش مصنوعی و هدف اصلی

هوش مصنوعی به عنوان شاخه ای از علوم کامپیوتر، در دهه ۱۹۵۰ با هدف ساخت سیستم ها و ماشین هایی که توانایی تقلد از هوش انسانی را دارند، آغاز شد. نخستین بار مکاری در سال ۱۹۵۶ این اصطلاح را به کار گرفت. و هوش مصنوعی به عنوان علمی که در آن به مطالعه الگوریتم هایی برای تقلید رفتار انسانی می پردازد، شناخته شد. اهداف اولیه هوش مصنوعی شامل توانایی درک زبان، یادگیری، حل مسئله و تولید موجودات هوشمند بود. در این دوران پروژه های تحقیقاتی زیادی

به امید دستیابی به هوش مصنوعی عمومی (Artificial AGI) General (Intelligence) شروع به کار کردند

۲.۱.۱ دوره طلایی و پیشرفت‌های اولیه

در دهه ۵۰ و ۶۰ میلادی، هوش مصنوعی به عنوان یکی از پرچمداران پژوهش‌های نوین شناخته می‌شد. الگوریتم‌های اولیه به کمک روش‌های منطقی و ریاضیاتی برای حل مسئله و بازی‌های ساده توسعه یافتند مانند انواع الگوریتم جستجوی درختی که در این دوره به وجود آمدند و زمینه ساز اولین دستاوردهای هوش مصنوعی در بازی‌های تخته‌ای همچون شطرنج شدند. در این دوران پیشرفت‌های بیشتری در پردازش زبان طبیعی (NLP) و سیستم‌های خبره (Expert Systems) نیز صورت گرفت که این امید را در دانشمندان و محققان تقویت کرد که دستیابی به هوش مصنوعی عمومی به زودی ممکن خواهد بود.

۳.۱.۱ انتظارات بیش از حد و ظهور عصر تاریک

با وجود پیشرفت‌های هوش مصنوعی، محدودیت‌های تکنولوژی مثل gpu ها و محاسباتی در آن زمان و همچنین کمبود داده‌های کافی برای آموزش مدل‌های پیچیده‌تر، باعث شد که بسیاری از پروژه‌های تحقیقاتی نتوانند به نتایج پیش‌بینی شده دست یابند. و در نتیجه، هوش مصنوعی در دهه ۷۰ به مرحله‌ای از رکود وارد شد که به آن عصر تاریک هوش مصنوعی یا (AI Winter) می‌گویند. در این دوران بسیاری از پروژه‌ها تعطیل و سرمایه‌گذاری‌ها قطع شدند و دولت‌ها و سازمان‌های سرمایه‌گذار به دلیل عدم دستیابی به نتایج مطلوب از ادامه سرمایه‌گذاری منصرف شدند.

۴.۱.۱ عوامل اصلی عصر تاریک هوش مصنوعی

- محدودیت‌های سخت‌افزاری: در آن زمان، سیستم‌های اولیه هوش مصنوعی به محاسبات سنگینی نیاز داشتند که با توان پردازشی محدود آن زمان همخوانی نداشت.

● کمبود داده‌ها: در آن زمان، دسترسی به داده‌های کافی برای آموزش مدل‌های پیچیده ممکن نبود و الگوریتم‌های موجود به داده‌های بیشتری نیاز داشتند تا بتوانند به درستی آموزش ببینند و عملکرد مطلوبی داشته باشند.

● روش‌های محدود یادگیری: الگوریتم‌های اولیه به شدت به برنامه‌ریزی انسانی وابسته بودند و در بسیاری از موارد، مدل‌ها قادر به تعمیم به مسائل جدید نبودند و نمی‌توانستند تعمیم‌پذیری خیلی بالایی داشته باشند. [۲].

۵.۱.۱ پایان عصر تاریک و بازگشت هوش مصنوعی

پس از چندین سال رکود و عدم سرمایه‌گذاری در حوزه هوش مصنوعی، سرانجام در دهه ۱۹۸۰ و ۱۹۹۰ عصر تاریک هوش مصنوعی با تحولات تکنولوژی و از همه مهم‌تر ظهور سیستم‌های خبره (Expert-Systems) به پایان رسید. سیستم‌های خبره به عنوان یکی از اولین تلاش‌های موفق برای کاربردهای صنعتی در هوش مصنوعی به وجود آمدند. برخلاف الگوریتم‌های اولیه، این سیستم‌ها از پایگاه بزرگ قواعد و قوانین (Rule-Based-Systems) استفاده می‌کردند. در سیستم‌های خبره به جای تلاش برای شبیه‌سازی کلی هوش مصنوعی، بر حل مسائل تخصصی برای صنایع و سازمان‌ها تمرکز می‌کردند. برای مثال، سیستم‌های خبره در پزشکی برای تشخیص بیماری‌ها و پیشنهاد درمان، در صنعت برای مدیریت و پیش‌بینی خرابی ماشین‌آلات، و در امور مالی برای تحلیل و ارزیابی ریسک کاربرد داشتند. هرچند این سیستم‌ها نمی‌توانستند درک عمیق و هوشمندی عمومی را ایجاد کنند. اما برای رفع نیازهای پیچیده مناسب بودند. همزمان با موفقیت این سیستم‌ها، بهبودهای زیادی در سخت‌افزارها و کاهش هزینه‌های پردازش به وجود آمد. در دهه‌های ۱۹۸۰ و ۱۹۹۰ کامپیوترها به تدریج قوی‌تر و مقرون به‌صرفه‌تر شدند و امکان پردازش داده‌های بیشتر و اجرای الگوریتم‌های پیچیده‌تر فراهم شد. این افزایش توان محاسباتی، نیاز به پردازش داده‌های بزرگ و پیچیده را فراهم کرد و در نتیجه دسترسی به داده‌ها و انجام محاسبات سنگین برای توسعه الگوریتم‌های جدید فراهم شد. از طرف دیگر، پیشرفت‌های انجام شده در ذخیره‌سازی داده

و رشد اینترنت باعث دسترسی گسترده تر به داده ها و منابع اطلاعاتی شد. به این ترتیب، مجموعه ای از عوامل شامل ظهور سیستم های خبره، افزایش قدرت پردازش و دسترسی به داده های بیشتر، منجر به بازگشت هوش مصنوعی شد و این دوره نه تنها پایان عصر هوش مصنوعی بود، بلکه راه را برای الگوریتم های یادگیری ماشین و توسعه شبکه های عصبی هموار کرد. [؟، ؟، ؟]

۲.۱ انواع مدل یادگیری ماشین و شبکه های عصبی

۱.۲.۱ یادگیری ماشین: مروری کلی

یادگیری ماشین (Machine-Learning) شاخه ای از هوش مصنوعی است که به مدل های محاسباتی این امکان را می دهد الگو ها از داده ها را به طور خودکار یاد بگیرند و بتوانند تصمیم گیری کنند در واقع، هدف یادگیری ماشین این است که مدل ها بتوانند از داده ها الگو ها و روابط پنهان را استخراج کنند و به نتایج و رفتار و تصمیم های قابل اعتماد دست یابند.

۲.۲.۱ تقسیم بندی های اصلی در یادگیری ماشین

یادگیری ماشین به سه دسته اصلی تقسیم می شود: یادگیری با نظارت (Supervised-Learning) یادگیری بدون نظارت (Unsupervised-Learning) یادگیری تقویتی (Reinforcement-Learning)

۳.۲.۱ یادگیری نظارت شده (Supervised-Learning)

یادگیری نظارت شده یکی از رایج ترین روش ها در یادگیری ماشین شناخته می شود. که در آن از مجموعه داده های برچسب گذاری شده برای آموزش مدل استفاده می کنیم. هدف این الگوریتم تشخیص الگو ها در میان داده های ورودی است که این امکان را می دهد پیش بینی یا طبقه بندی هایی روی داده جدید انجام دهد. این نوع شامل دو الگوریتم Regression و classification می شود.

طبقه‌بندی (Classification)

طبقه‌بندی یکی از مهم‌ترین و اصلی‌ترین وظایف در یادگیری نظارت شده است که هدف آن تخصیص داده‌ها به یک لیبل مشخص است. در این روش مدل با داده‌های برچسب دار (label) آموزش می‌بیند و یاد می‌گیرد که داده‌های جدید را بر اساس الگوها و ویژگی‌هایی که در داده‌های آموزشی دیده است، به دسته مناسب اختصاص دهد. از کاربرد های طبقه‌بندی می‌توان به تشخیص اسپم (spam)، تشخیص بیماری که آیا یک فرد مبتلا به بیماری هست یا نه، تشخیص چهره و ... استفاده کرد.

رگرسیون (regression)

رگرسیون یکی از مهم‌ترین وظایف یادگیری ماشین است و هدف آن پیش‌بینی مقادیر پیوسته است. برخلاف طبقه‌بندی که خروجی آن دسته‌بندی مجزا است، در رگرسیون، خروجی یک مقدار پیوسته است و مدل یاد می‌گیرد روابط بین متغیرهای مستقل و متغیرهای هدف را شناسایی کند. از کاربرد های رگرسیون می‌توان به پیش‌بینی قیمت مسکن، پیش‌بینی آب و هوا و ... اشاره کرد.

یادگیری تقویتی (Reinforcement-Learning)

یادگیری تقویتی نوعی یادگیری بر پایه پاداش و تنبیه است که در آن، مدل با محیط تعامل می‌کند و بر اساس پاداش یا تنبیه یاد می‌گیرد. برخلاف یادگیری نظارت‌شده و بدون نظارت، یادگیری تقویتی به مدل این امکان را می‌دهد، که از طریق آزمون و خطا بهترین راهکارها را برای انجام یک عمل یاد بگیرد. در این روش، مدل به جای برچسب، از یک تابع پاداش استفاده می‌کند که مشخص می‌کند چه اقداماتی باعث نتیجه بهینه می‌شود. از کاربرد های یادگیری تقویتی می‌توان به بازی‌ها (games)، کنترل رباتیک (robotic control)، سیستم‌های توصیه‌گر (Recommender-Systems) نام برد.

۴.۲.۱ معرفی چند مدل از الگوریتم یادگیری کلاسیک

نزدیک ترین همسایه (k-nearest-Neighbors)

KNN یکی از الگوریتم های ساده و کار آمد در یادگیری نظارت شده است که هم در دسته بندی و هم در رگرسیون کاربرد دارد این الگوریتم برای پیش بینی دسته بندی های یک نمونه جدید، به k نزدیک ترین داده ها در فضای ویژگی نگاه میکند و بر اساس اکثریت نزدیک ترین همسایه دسته بندی را انجام میدهد

مزایا:

- سادگی و قابل فهم بودن: این الگوریتم به سادگی با اندازه گیری فاصله بین نقاط داده کار می کند و بدون نیاز به آموزش مدل پیچیده قابل استفاده است.
- عملکرد خوب در داده های با تعداد ویژگی کم: در مسائلی که تعداد ویژگی ها کم است این الگوریتم به خوبی کار میکند.

معایب:

- حساسیت به داده های پرت: نقاط پرت می توانند به طور قابل توجهی بر نتایج تاثیر بگذارند.
- کندی در داده های بزرگ: این الگوریتم نیاز به محاسبه فاصله برای هر نقطه جدید دارد که در داده های بزرگ بار محاسباتی سنگین می شود.
- عدم کارایی در داده های با ابعاد بالا: در داده های با تعداد ویژگی های زیاد، کارایی الگوریتم کاهش می یابد.

۵.۲.۱ ماشین بردار پشتیبان: (Support-Vector-Machine)

الگوریتمی است که با یافتن یک ابر صفحه بهینه، داده‌ها را به کلاس مختلف تقسیم میکند. این الگوریتم یک ابر صفحه به دست می‌آورد که هدف آن حداکثر کردن فاصله میان داده‌های دو کلاس است و به این ترتیب میتواند طبقه بندی دقیقی داشته باشد.

مزایا:

- توانایی مقابله با داده‌های پیچیده و ابعاد بالا: SVM می‌تواند به خوبی با داده‌های چندبعدی و پیچیده کار کند.
- مقاومت در برابر بیش‌براش (Overfitting): با استفاده از هسته‌ها (kernels) می‌توان داده‌های غیرخطی را نیز به فضای بالاتر برد و جداسازی بهتری انجام داد.

معایب:

- پیچیدگی محاسباتی: آموزش SVM به دلیل نیاز به حل مسائل بهینه‌سازی، در حجم‌های بالای داده محاسباتی زمان‌بر است.
- کارایی پایین در داده‌های پرت: در صورتی که داده‌ها شامل نقاط پرت زیادی باشند، دقت مدل کاهش می‌یابد.

۶.۲.۱ بیز ساده: (Naive-Bayes)

بیز ساده مبتنی بر قضیه بیز است و فرض میکند ویژگی‌ها به صورت شرطی مستقل از هم هستند. این مدل برای اولین بار در حوزه پردازش متن به کار رفت و هنوز هم در بسیاری از موارد مانند طبقه بندی ایمیل و تحلیل احساسات مورد استفاده قرار می‌گیرد. نایو بیز بر اساس احتمالات محاسبه میکند که یک نمونه جدید به کدام دسته تعلق دارد. این الگوریتم بر اساس قضیه بیز، احتمال تعلق

یک نمونه ب دسته را به ازای هر ویژگی محاسبه کرده و بیشترین احتمال را بر اساس جواب نهایی در نظر میگیرد.

مزایا:

- سرعت بالا: به دلیل محاسبات ساده و فرض استقلال ویژگی‌ها، Bayes Naive بسیار سریع و کم حجم است.
- کارایی در داده‌های کوچک: حتی با داده‌های کم، این الگوریتم عملکرد نسبتاً خوبی دارد.

۷.۲.۱ معایب:

- فرض استقلال ویژگی‌ها: فرض استقلال ویژگی‌ها ممکن است در بسیاری از مسائل واقعی صادق نباشد و این می‌تواند دقت مدل را کاهش دهد.
- حساسیت به داده‌های نادرست: در صورت داده‌های نادرست یا پرت، مدل ممکن است دقت کمتری داشته باشد.

۸.۲.۱ شبکه‌های عصبی بازگشتی (RNN) و شبکه‌های حافظه بلند مدت کوتاه مدت (LSTM)

شبکه عصبی بازگشتی با RNN ها و مدل هایی با حافظه بلند مدت تر مانند Lstm با هدف داده های ترتیبی و وابسته به زمان توسعه یافتند. این مدل ها به ویژه در تحلیل زبان طبیعی، صوت و پیش بینی سری های زمانی بسیار موفق عمل کرده اند. زیرا قادر به حفظ اطلاعات گذشته بودند و از این اطلاعات گذشته برای پیش بینی در لحظه حال و آینده استفاده میکنند.

RNN: ۹.۲.۱

مدل های اولیه شبکه های عصبی، مانند شبکه های چند لایه، (Mlp) قادر به پردازش داده های مستقل و ثابت بودند و نمیتوانستند وابستگی های زمانی را یاد بگیرند. در بسیاری از مباحث دنیای واقعی، مانند تحلیل متن و صدا، داده ها به ترتیب خاصی وابسته هستند. به همین دلیل شبکه های Rnn معرفی شدند تا از اطلاعات پیشین در پردازش داده های بعدی استفاده کنند.

RNN: ساختار و عملکرد

شبکه های Rnn دارای حلقه بازگشتی هستند که به مدل این امکان را می دهد اطلاعات را در توالی نگه دارند. و در هر گام زمانی ورودی فعلی x_t و وضعیت قبلی h_{t-1} به عنوان ورودی به نرون داده می شود.

$$h_t = \sigma(W \cdot x_t + U \cdot h_{t-1} + b) \quad (۱.۲.۱)$$

در اینجا:

h_t وضعیت مخفی یا حالت در گام زمانی t است.

W وزن هایی است که به ورودی x_t اعمال می شود.

U وزن های اعمال شده بر وضعیت قبلی h_{t-1} است.

b بایاس مدل است.

σ تابع فعال سازی، معمولاً تانژانت هیپربولیک یا سیگموید.

با استفاده از این فرایند، مدل این توانایی را دارد که اطلاعات گذشته را در خود ذخیره کرده و در پردازش های بعدی آن ها را به کار بگیرد.

Rnn: مزایا و معایب ۱۰.۲.۱

در این قسمت به مزایا و معایب RNN میپردازیم.

مزایا:

- حفظ وابستگی زمانی: RNN قادر به پردازش توالی‌های طولانی است و می‌تواند اطلاعات را در طول توالی به خاطر بسپارد.
- کاربردهای گسترده در داده‌های ترتیبی: این مدل در تحلیل زبان طبیعی، پیش‌بینی سری‌های زمانی و پردازش صوتی بسیار موفق عمل می‌کند.

معایب:

- مشکل ناپدید شدن و انفجار گرادیان (Vanishing and Exploding Gradient): در فرآیند آموزش با روش پسانتشار، اگر توالی داده طولانی باشد، گرادیان‌ها ممکن است به سرعت کوچک یا بزرگ شوند، که منجر به ناپایداری آموزش و کاهش دقت می‌شود.
- محدودیت در پردازش توالی‌های بسیار بلند: RNN در حفظ اطلاعات طولانی مدت دچار مشکل است و برای پردازش وابستگی‌های طولانی، عملکرد ضعیفی دارد.

۱۱.۲.۱ شبکه‌های حافظه بلندمدت - کوتاه‌مدت (LSTM)

علل پیدایش lstm:

شبکه‌های lstm به عنوان یک راه حل برای یکی از بزرگترین مشکلات شبکه‌های عصبی بازگشتی (RNN) ها معرفی شدند. یکی از بزرگترین مشکلاتی که در شبکه‌های بازگشتی وجود داشت مشکل ناپدید شدن گرادیان (Vanishing Gradient) بود. این مشکل باعث میشد RNN ها قادر به یادگیری وابستگی‌های بلند مدت نباشند. برای درک عمیق تر ابتدا به توضیح مشکل ناپدید شدن گرادیان و سپس حل آن توسط lstm می‌پردازیم.

Vanishing: Gradient

شبکه‌های RNN برای پردازش داده‌های ترتیبی، از حالت‌های بازگشتی استفاده می‌کنند. در فرآیند آموزش، RNN از الگوریتم پس‌انتشار خطا از طریق زمان (Backpropagation Through Time - BPTT) استفاده می‌شود. این الگوریتم گرادیان‌ها را برای به‌روزرسانی وزن‌ها محاسبه می‌کند. با این حال، به دلایل زیر، RNN‌ها وابستگی‌های بلندمدت ناکام می‌مانند:

- ضریب‌های بازگشتی کوچک‌تر از ۱: در فرآیند محاسبه گرادیان‌ها، اگر مقدار مشتقات یا ضرایب در هر مرحله کوچک‌تر از ۱ باشد، ضرب مکرر این مقادیر در طول توالی منجر به کوچک شدن گرادیان‌ها به سمت صفر می‌شود. این پدیده، ناپدید شدن گرادیان نام دارد.

فرمول کلی گرادیان در زمان t به صورت زیر است:

$$\frac{\partial L}{\partial W} = \prod_{k=1}^t \frac{\partial h_k}{\partial h_{k-1}} \cdot \frac{\partial h_t}{\partial L}$$

در اینجا، $\frac{\partial h_k}{\partial h_{k-1}}$ می‌تواند مقداری کوچک‌تر از ۱ باشد، و ضرب مکرر این مشتقات باعث کاهش شدید مقدار گرادیان می‌شود.

- تاثیر مستقیم بر وزن‌ها: زمانی که گرادیان‌ها نزدیک به صفر شوند، وزن‌های مدل به‌طور موثری به‌روزرسانی نمی‌شوند. این امر مانع از یادگیری وابستگی‌های طولانی مدت در داده‌ها می‌شود.

۱۲.۲.۱ Lstm: ظهور

در سال ۱۹۹۷، Hochreiter Sepp و Schmidhuber Jürgen شبکه‌های حافظه بلندمدت - کوتاه‌مدت (LSTM) را معرفی کردند. انگیزه اصلی توسعه LSTM مشکل ناپدید شدن گرادیان در شبکه‌های RNN بود. این مشکل در مسائل یادگیری داده‌های ترتیبی طولانی باعث می‌شد RNN نتواند وابستگی‌های بلندمدت را به درستی یاد بگیرد.

راه حل LSTM برای پایداری جریان گرادیان‌ها:

LSTM با معرفی یک معماری جدید در شبکه‌های بازگشتی، جریان گرادیان‌ها را در طول توالی پایدار نگه می‌دارد. این کار از طریق اضافه کردن وضعیت سلولی (Cell) (State) و دروازه‌ها (Gates) به ساختار RNN انجام می‌شود. این اجزا به LSTM امکان می‌دهند که:

۱. اطلاعات غیرضروری را فراموش کند.

۲. اطلاعات مهم جدید را اضافه کند.

۳. اطلاعات مهم قبلی را حفظ کند.

۳.۱ اختار LSTM: نوآوری در مقایسه با RNN

LSTM شامل اجزای جدیدی است که به آن امکان مدیریت بهتر اطلاعات را می‌دهد:

۱.۳.۱ وضعیت سلولی (Cell): (State)

مسیر اصلی ذخیره اطلاعات در LSTM است که می‌تواند اطلاعات مهم را در طول توالی حفظ کند. برخلاف RNN که وابسته به خروجی‌های بازگشتی h_t است، LSTM یک مسیر جداگانه برای عبور اطلاعات از وضعیت سلولی دارد که به حفظ گرادیان‌ها کمک می‌کند.

۲.۳.۱ دروازه‌ها: (Gates)

دروازه‌ها نقش فیلترهای اطلاعاتی را دارند که جریان اطلاعات را در طول فرآیند یادگیری کنترل می‌کنند:

- دروازه فراموشی (Forget Gate): تعیین می‌کند چه اطلاعاتی از وضعیت سلولی باید حذف شود.

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

مقدار فراموشی برای هر عنصر از وضعیت سلولی: f_t

تابع سیگموئید که خروجی آن بین ۰ و ۱ است: σ

وقتی $f_t = 0$ ، اطلاعات به طور کامل حذف می‌شود؛ وقتی $f_t = 1$ ، اطلاعات حفظ می‌شود.

• دروازه ورودی (Input Gate): تعیین می‌کند چه اطلاعات جدیدی باید به وضعیت سلولی اضافه شود.

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

میزان اطلاعات جدیدی که به وضعیت سلولی وارد می‌شود: i_t

مقدار جدید محاسبه شده برای اضافه شدن به وضعیت سلولی: \tilde{C}_t

• دروازه خروجی (Output Gate): تعیین می‌کند چه اطلاعاتی از وضعیت سلولی به خروجی منتقل شود.

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$

$$h_t = o_t \cdot \tanh(C_t)$$

۳.۳.۱ به‌روزرسانی وضعیت سلولی:

وضعیت سلولی C_t با استفاده از اطلاعات جدید و قدیمی به‌روزرسانی می‌شود:

$$C_t = f_t \cdot C_{t-1} + i_t \cdot \tilde{C}_t$$

این ساختار باعث می‌شود که اطلاعات قدیمی مهم حفظ شده و اطلاعات غیرضروری حذف شوند.

علت پایداری گرادیان در Lstm:

- حذف ضرب‌های مکرر: برخلاف RNN که به ضرب‌های مکرر وزن‌ها و گرادیان‌ها وابسته است، LSTM با مسیر جداگانه وضعیت سلولی، از کاهش نمایی گرادیان جلوگیری می‌کند.
- استفاده از توابع سیگموئید و تانژانت هیپربولیک: توابع سیگموئید در دروازه‌ها و تانژانت هیپربولیک در وضعیت سلولی باعث محدود کردن مقادیر و جلوگیری از انفجار گرادیان می‌شوند.
- مدیریت اطلاعات توسط دروازه‌ها: دروازه‌های فراموشی و ورودی به مدل اجازه می‌دهند تنها اطلاعات مهم حفظ شود و داده‌های غیرضروری حذف شوند، که این موضوع از پیچیدگی‌های محاسباتی غیرضروری جلوگیری می‌کند.

۴.۳.۱ مشکلات کلی rnn و lstm و ظهور ترانسفورمرها

شبکه‌های بازگشتی rnn، lstm ها توانستند بسیاری از مشکلات و محدودیت‌های مدل‌های اولیه را حل کنند اما همچنان با چالش‌ها و محدودیت‌هایی مواجهه بودند که در مسائل پیچیده‌تر، مانند ترجمه زبان یا تحلیل داده‌های بلند مدت و حجیم، مشکلاتی را ایجاد می‌کردند. این مشکلات در نهایت باعث پیدایش ترانسفورمرها شد. که در اینجا مشکلات آن‌ها را بررسی می‌کنیم

مشکل وابستگی ترتیبی در RNN و LSTM

RNN و LSTM داده‌ها را به ترتیب پردازش می‌کنند، به این معنی که برای پردازش داده‌های گام زمانی t ، باید تمامی داده‌های قبلی $(t - 1)$ را پردازش کرده باشند. این ویژگی مشکلات زیر را ایجاد می‌کند:

- غیرقابل موازی‌سازی: به دلیل وابستگی ترتیبی، پردازش داده‌ها به صورت موازی ممکن نیست، که باعث افزایش زمان محاسباتی می‌شود. این مشکل در داده‌های بلند، مانند متن‌های

طولانی یا سری‌های زمانی بزرگ، قابل توجه است.

- کندی آموزش و استنتاج: ترتیب خطی باعث می‌شود که زمان آموزش و پیش‌بینی مدل‌ها به شدت افزایش یابد، به ویژه زمانی که با حجم زیادی از داده‌ها سر و کار داریم.

محدودیت در یادگیری وابستگی‌های بسیار طولانی

با وجود پیشرفت LSTM در یادگیری وابستگی‌های بلندمدت نسبت به RNN این مدل‌ها همچنان در یادگیری وابستگی‌های بسیار بلند، مانند ارتباطات بین کلمات در دو جمله متفاوت یا درک ساختار کلی یک متن، محدودیت دارند:

- مشکل در داده‌های بسیار طولانی: حتی در LSTM ظرفیت حفظ اطلاعات محدود است و با افزایش طول توالی، دقت مدل کاهش می‌یابد.
- تاثیر تدریجی داده‌های اولیه: داده‌های ابتدایی توالی ممکن است با گذشت زمان اهمیت خود را از دست بدهند، زیرا گرادین‌ها به تدریج ضعیف‌تر می‌شوند.

پیچیدگی محاسباتی و حافظه

ها LSTM به دلیل ساختار پیچیده‌ای که شامل چندین ماتریس ضرب (برای دروازه‌های فراموشی، ورودی و خروجی) و به‌روزرسانی وضعیت سلول است، نیاز به حافظه و محاسبات زیادی دارند:

- نیاز به حافظه بیشتر: برای ذخیره وضعیت سلولی و گرادین‌ها، به حافظه بیشتری نسبت به مدل‌های ساده‌تر نیاز است.

- هزینه محاسباتی بالا: به خصوص در داده‌های بزرگ، محاسبات سنگین باعث کندی اجرای مدل‌ها می‌شود.

مشکل پردازش وابستگی‌های غیرمتوالی

ها RNN و LSTM به طور طبیعی برای یادگیری وابستگی‌های محلی و متوالی مناسب هستند. با این حال، در مسائل پیچیده مانند ترجمه زبان یا تحلیل متون، وابستگی‌های غیرمحلی و غیرمتوالی نیز وجود دارند. به عنوان مثال:

در یک جمله طولانی، ممکن است کلمه‌ای در ابتدای جمله با کلمه‌ای در انتهای جمله ارتباط معنایی داشته باشد. ها RNN و LSTM برای یادگیری این نوع وابستگی‌ها به شدت محدود هستند.

گرادیان‌های ناپایدار و مشکلات بهینه‌سازی:

با وجود بهبودهایی که LSTM نسبت به RNN در پایداری گرادیان‌ها ارائه داد، هنوز هم:

- مسائل گرادیان‌های ناپایدار: در توالی‌های بسیار بلند، گرادیان‌ها ممکن است همچنان کاهش یابند یا حتی در برخی موارد به حد انفجار برسند.

- مشکلات بهینه‌سازی: در مسائل پیچیده‌تر، یافتن مینیمم مناسب تابع هزینه با استفاده از ها RNN و LSTM دشوار است.

نیاز به مدلی با ظرفیت بیشتر و سرعت بالاتر:

- مدل‌های بزرگ‌تر: برای مسائل پیچیده‌تر، مدل‌هایی با تعداد پارامتر بیشتر نیاز است که RN-ها و LSTM به دلیل محدودیت در حافظه و پردازش، پاسخگوی این نیاز نیستند.

- کارایی در داده‌های چندوجهی: (Multimodal) برای داده‌هایی که شامل اطلاعات متنی، صوتی و تصویری هستند، ها RNN و LSTM توانایی لازم برای هم‌زمان پردازش این اطلاعات را ندارند.

وابستگی ترتیبی در RNN و LSTM یک مانع اساسی در برای استفاده از این مدل ها در مسئل پیچیده و بزرگ بود. که باعث به وجود آمدن ترانسفورمر ها شد. که با طراحی مبتنی بر موازی سازی و ماکانیزم توجه،، این محدودیت را حذف کردند و راه حلی کارآمد تر برای پردازش داده های ترتیبی ارائه دادند.

فصل ۲

پیشینه پژوهش

۱.۲ مقدمه

ظهور مدل‌های Transformer و انقلاب در یادگیری عمیق، یکی از تحولات اساسی در حوزه پردازش زبان طبیعی (NLP) و یادگیری ماشین به شمار می‌رود. این مدل‌ها باعث تغییرات عمده‌ای در نحوه ساخت و آموزش مدل‌های زبانی و همچنین در بسیاری از کاربردهای دیگر یادگیری ماشین شده‌اند. و توانستند بسیاری از مشکلات مدل‌های قبلی را حل کنند.

۲.۲ مشکلات ترجمه ماشینی و ترانسفورمرها:

در ابتدا، ترجمه ماشینی (MT) یک چالش اساسی در زمینه پردازش زبان طبیعی بود. مدل‌های اولیه‌ای مانند مدل‌های مبتنی بر قواعد (Rule-based models) برای ترجمه استفاده می‌شدند که در آن‌ها، ترجمه‌ها به صورت دستی با استفاده از قواعد زبانی مشخص تنظیم می‌شدند. این روش‌ها هرچند دقیق بودند، اما محدودیت‌های زیادی داشتند و نمی‌توانستند ویژگی‌های پیچیده‌تر زبان را مدل‌سازی کنند. سپس مدل‌های آماری (Statistical Models) معرفی شدند. این مدل‌ها

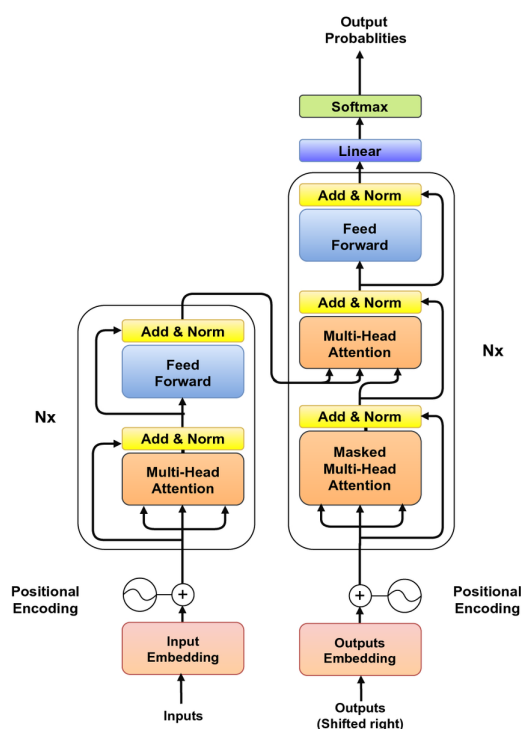
از داده‌های ترجمه‌شده برای آموزش مدل‌های آماری استفاده می‌کردند که احتمال ترجمه‌ای صحیح را براساس شواهد آماری محاسبه می‌کردند. مدل‌هایی مانند مدل‌های ترجمه آماری مبتنی بر جمله (Statistical (Phrase-based Models) از این نوع بودند، که قادر به ترجمه جملات بهتر از مدل‌های مبتنی بر قواعد بودند، اما هنوز هم در ترجمه‌های پیچیده با مشکلاتی روبه‌رو بودند. بعد از این مدل‌ها مدل‌ها بازگشتی به وجود آمدند که مشکلات آن را در فصل گذشته بیان کردیم و د نهایت این مشکلات باعث به وجود آمدن ترانسفورمرها شد.

۳.۲ ظهور ترانسفورمرها:

در سال ۲۰۱۷ مقاله ... توسط گوگل منتشر شد که مفهوم جدیدی به نام ترانسفورمرها را بیان کرد. این مقاله به موضوع ترجمه ماشینی پرداخت و نشان داده با استفاده از مفهوم مکانیزم توجه (Attention-Mechanism) می‌توان بسیاری از مشکلات مدل‌های قبلی را حل کرد. این مدل‌ها از پردازش موازی استفاده می‌کردند برخلاف مدل‌های قبلی که از پردازش‌های سریالی استفاده می‌کردند. و این مدل‌ها قادر هستند به طور همزمان به تمام بخش‌های ورودی توجه کنند. و این کار باعث شد که ترانسفورمرها در پردازش تصویر و متن بسیار سریع‌تر و دقیق‌تر از مدل‌های قبلی عمل کنند.

۴.۲ معماری ترانسفورمرها:

در تصویر ... معماری ترانسفورمر نمایش داده شده است و بخش‌ها و اجزای مختلف آن مشخص شده است که از دو بخش اصلی انکودر و دیکدر تشکیل شده است هدف انکودر این است که داده ورودی را بگیرد و ویژگی‌های آن را استخراج می‌کند و هدف دیکدر این است که ویژگی‌های استخراج شده را به زبان مقصد تبدیل می‌کند. که به طور مختصر به معماری و بخش‌های مختلف آن می‌پردازیم.



شکل ۲.۴.۱: معماری ترانسفورمر ها

۱.۴.۲ Embedding:

در زبان طبیعی کلمات به شکل رشته های متنی هستند مانند کتاب ، ماشین و ... کامپیوتر ها نمیتوانند به طور مستقیم به شکل رشته های متنی این کلمات را پردازش کنند. به همین دلیل در یادگیری ماشین ما این کلمات را به شکل یک بردار نمایش می دهیم. و آن بردار بیانگر آن کلمه در مدل است تا ماشین بتواند آن کلمه را پردازش کند.

این بردار ها ویژگی های کلمه را فضای عددی نمایش میدهد. روش های مختلفی برای تبدیل متن به بردار وجود دارند. از جمله این روشها میتوان به روش Word2Vec و GloVe اشاره کرد. همانطور که در تصویر ... نشان داده شده است، هر کلمه که به صورت توکن است ابتدا در دیکشنری تعریف شده پیدا می شود و پس از پیدا شدن در دیکشنری با استفاده از روش های embedding، هر کلمه به برداری از اعداد تبدیل می شوند. این embedding ها شباهت های معنایی بین کلمات را مدل سازی میکنند به طوری که کلماتی که از نظر معنایی شبیه به هم هستند، بردار آن ها نیز به

یک دیگر نزدیک تر است. و به این ترتیب کلمات برای مدل ها و شبکه های عصبی قابل فهم می شود.

YOUR	CAT	IS	A	LOVELY	CAT
105	6587	5475	3578	65	6587
952.207	171.411	621.659	776.562	6422.693	171.411
5450.840	3276.350	1304.051	5567.288	6315.080	3276.350
1853.448	9192.819	0.565	58.942	9358.778	9192.819
...
1.658	3633.421	7679.805	2716.194	2141.081	3633.421
2671.529	8390.473	4506.025	5119.949	735.147	8390.473

شکل ۲.۴.۲: word embedding

۲.۴.۲ positional embedding

ما تا الان هر کلمه را به برداری از اعداد که برای مدل قابل فهم باشد، تبدیل کرده ایم. اما مدل های ترانسفورمر نمیتوانند جایگاه هر کلمه را تشخیص دهند. در مدل های ترانسفورمر بر خلاف مدل های بازگشتی به دلیل این که کلمه های به صورت موازی وارد میشوند نیاز داریم تا جایگاه هر کلمه را بدانیم به طور مثال جمله من تو رو دوست داریم باید به طور دقیق بدانیم کلمه من، کلمه اول جمله است، کلمه تو کلمه دوم جمله است و ... حال ما باید به مدل توالی این کلمات را بفهمیم، پس ما نیاز داریم به مدل یک سری اطلاعات اضافی بدهیم به طوری که مدل توالی کلمات را یاد بگیرد. روش های مختلفی برای اضافه کردن positional embedding به مدل وجود دارد که در ترانسفورمر ها از روش Embedding Positional Sinusoidal استفاده میشود. این روش قابل یادگیری نیست و صرفاً از یک سری فرمول های ساده برای positional embedding استفاده میکند.

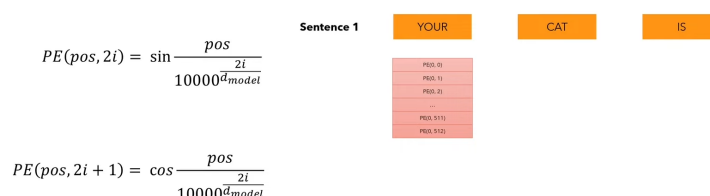
برای موقعیت pos در توالی و بعد i در فضای برداری، تعبیه موقعیتی به صورت زیر تعریف

می شود:

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

و برای مقادیر فرد:

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



شکل ۲.۴.۳: embedding word

pos - موقعیت کلمه در توالی است. برای مثال، اگر توالی ورودی شامل N کلمه باشد، موقعیت‌ها از 0 تا $N - 1$ تغییر می‌کنند. به عبارت دیگر، pos می‌تواند هر عددی از مجموعه $\{0, 1, 2, \dots, N - 1\}$ باشد که نشان‌دهنده موقعیت یک کلمه خاص در توالی است.

i - شاخص بعد در بردار موقعیتی است. این متغیر به اندیس بعدی که موقعیت کلمه در آن نمایش داده می‌شود اشاره دارد. برای مثال، اگر فضای برداری مدل دارای ابعاد d باشد، i از 0 تا $d - 1$ تغییر می‌کند.

d - ابعاد فضای برداری مدل است. این مقدار مشخص می‌کند که هر کلمه در توالی به چه تعداد ابعاد در فضای برداری نگاشت می‌شود. به عبارت دیگر، d نشان‌دهنده تعداد ویژگی‌ها (یا ابعاد) در بردار موقعیتی است.

10000 - یک مقدار ثابت است که برای تنظیم مقیاس توابع تناوبی استفاده می‌شود. این مقدار به گونه‌ای تنظیم شده است که از نوسانات زیاد جلوگیری کند و همچنین فرکانس‌های مختلفی را برای ابعاد مختلف به وجود آورد.

همانطور که در شکل زیر مشاهده میکنید بعد از embedding کلمات embed-positional ding به آن اضافه می شود. که در این روش از توابع سینوس و کسینوس استفاده میشود. این توابع موقعیت ها را در فضای برداری به گونه ای نگاشت میکنند که مدل بتواند از ترتیب کلمات در توالی آگاه باشد. این ویژگی به مدل کمک میکند تا مدل توالی زمانی را بتواند درک کند و الگوهای زمانی را شبیه سازی کند. از مزایای این مدل میتوان به عدم نیاز به آموزش و توزیع متوازن جایگاه هر کدام از کلمات اشاره کرد.

YOUR	CAT	IS	A	LOVELY	CAT
992.207	171.411	421.699	775.962	4422.493	171.411
5402.640	3276.350	1364.051	5567.258	4315.080	3276.350
1853.448	9192.819	0.345	56.942	9358.778	9192.819
...
1.458	3633.421	7679.805	2716.194	2141.081	3633.421
2671.529	8395.473	4556.025	5119.989	735.147	8395.473
+	+	+	+	+	+
...	1664.068	1281.458
...	8080.133	7902.890
...	2620.399	912.870
...	3821.102
...	9386.405	1659.217
...	2120.159	7018.620

شکل ۲.۴.۴: embedding word

۳.۴.۲ attention:

در روش های قدیمی (مانند RNN یا LSTM توالی ورودی (مثلاً یک جمله) معمولاً به صورت گام به گام پردازش می شد. اما در ترانسفورمر می خواهیم مدلی داشته باشیم که به هر موقعیت (مثلاً یک کلمه) در توالی نگاه کند و به همه موقعیت های دیگر نیز به صورت موازی دسترسی داشته باشد. به این مفهوم توجه می گوئیم. م به زبان ساده، وقتی توکن (کلمه) i به توکن های دیگر نگاه می کند، می خواهد بداند کدام توکن ها برای تفسیر معنای خودش مهم ترند.

به طور مثال در جمله یک گربه روی زمین نشسته است میخواد بداند کلمه گربه به واژه نشستن بیشتر توجه کند یا به زمین، مثلاً در این جا فعل نشستن ارتباط نزدیک تری به گربه دارد، و از نظر معنایی مرتبط تر است.

Q = Query (پرسش), K = Key (کلید), V = Value (مقدار / ارزش)

در Attention Dot-Product Scaled، ابتدا شباهت یا ارتباط بین Query و Key را با محاسبه ضرب داخلی (Dot Product) به دست می آوریم، سپس آن را نرمال می کنیم (با تقسیم بر d_k) و از تابع softmax استفاده می کنیم تا ضرایب توجه (Attention Weights) را به دست آوریم. در نهایت با همین ضرایب، ترکیبی خطی از بردارهای Value را می گیریم. فرمول به شکل زیر است:

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

که در آن:

$Q \in \mathbb{R}^{n \times d_k}$ توکن n ماتریس پرسش برای

$K \in \mathbb{R}^{n \times d_k}$ توکن n ماتریس کلید برای

$V \in \mathbb{R}^{n \times d_v}$ توکن n ماتریس مقدار برای

در حالت چندسری $d_k = d_{\text{model}}$ ابعاد بردارهای پرسش و کلید است (معمولاً d_k)

تقسیم بر d_k باعث می شود مقدار ضرب داخلی در ابعاد بالا خیلی بزرگ نشود و شیب ها (Gradients) پایدار بمانند.

$$\alpha = \text{softmax} \left(\frac{QK^T}{\sqrt{d_k}} \right)$$

α یک ماتریس با ابعاد $n \times n$ است که سطر i -ام آن ضرایب توجه برای توکن i را نشان می‌دهد. تفسیر ضرایب توجه: هر سطر از α نشان می‌دهد که توکن فعلی به چه توکن‌هایی در جمله، با چه شدتی توجه می‌کند.

	YOUR	CAT	IS	A	LOVELY	CAT
YOUR	0.268	0.119	0.134	0.148	0.179	0.152
CAT	0.124	0.278	0.201	0.128	0.154	0.115
IS	0.147	0.132	0.262	0.097	0.218	0.145
A	0.210	0.128	0.206	0.212	0.119	0.125
LOVELY	0.146	0.158	0.152	0.143	0.227	0.174
CAT	0.195	0.114	0.203	0.103	0.157	0.229

شکل ۲.۴.۵: Attention

ایدهٔ چندسری: به جای آنکه فقط یک بار Q, K, V بسازیم و عملیات توجه را انجام دهیم، چندین مجموعهٔ متفاوت Q_i, K_i, V_i می‌سازیم (هر کدام یک «Head» یا سر نام دارد) و به صورت موازی محاسبات Attention را انجام می‌دهیم. سپس خروجی همهٔ این Headها را کنار هم قرار داده (Concat) و در نهایت با یک ماتریس وزن دیگر ضرب می‌کنیم تا به بعد اصلی بازگردیم. فرمول مربوط به این ایده به شکل زیر است:

$$\text{head}_i = \text{Attention}(Q_i, K_i, V_i)$$

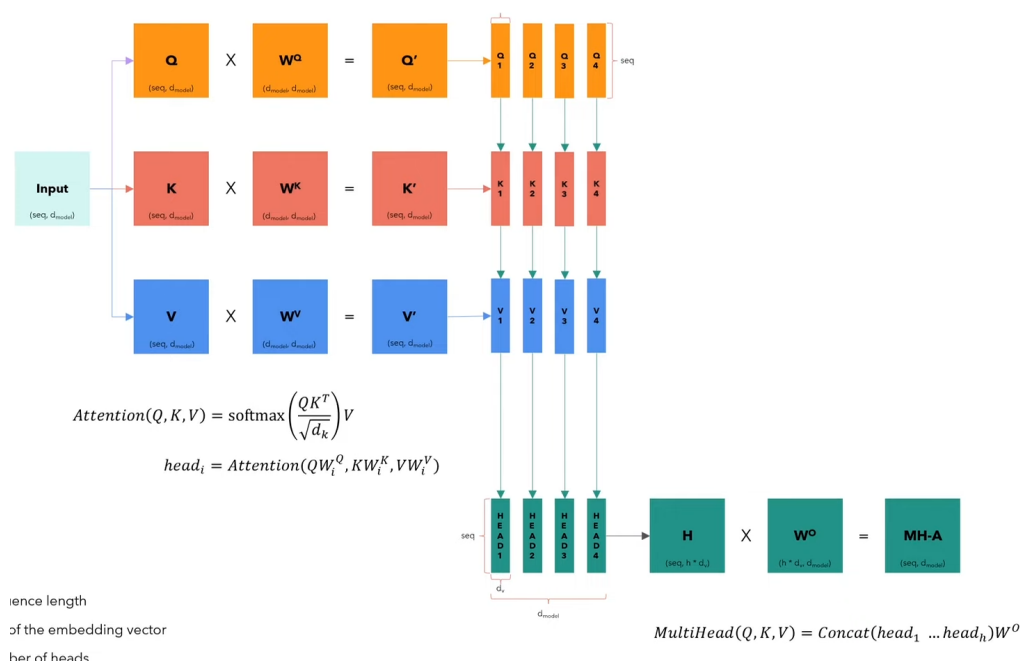
$$\text{MultiHead}(Q, K, V) = [\text{head}_1 \oplus \dots \oplus \text{head}_h] W_O$$

که در آن \oplus نشان‌دهنده عمل الحاق (Concatenation) است.

ماتریس وزن W_O به شکل زیر است:

$$W_O \in \mathbb{R}^{(h \cdot d_v) \times d_{\text{model}}}$$

که W_O ماتریسی است که خروجی الحاق شده را به بعد d_{model} برمی گرداند.



شکل ۲.۴.۶: attention head multi

چرا چندین سر؟

مشاهده چند منظر متفاوت: هر Head می‌تواند الگوهای گوناگونی از وابستگی‌ها را بیاموزد (مثلاً یک Head می‌تواند یاد بگیرد کلمه فعلی با کلمات همسایه نزدیک خود بیشتر مرتبط شود، یک Head دیگر روی ارتباط با کلماتی در فاصله دورتری متمرکز باشد، Head دیگر روی مطابقت جنس و تعداد در دستور زبان و ...).

افزایش ظرفیت مدل: با داشتن چند Head، مدل می‌تواند قدرت بیان بیشتری داشته باشد. ابعاد کمتر در هر Head: در عمل، اگر d_{model} مثلاً ۵۱۲ باشد، و تعداد هاها $h = 8$ ، آنگاه هر Head ابعادی در حدود $d_k = 64$ خواهد داشت؛ و این محاسبات ضرب داخلی را نیز مقیاس‌پذیر و قابل موازی‌سازی می‌کند.

۴.۴.۲ (Add): Connection Residual

در معماری‌های مختلف هنگامی که تعداد لایه‌ها زیاد می‌شود، اغلب دچار ناپایداری گرادیان (Vanishing/Exploding Gradients) می‌شوند. و باعث مشکل در آموزش مدل می‌شود. در مدل ترانسفورمر به جای این که خروجی attention را به صورت مستقیم به لایه بعدی بدهیم، ورودی آن را نیز حفظ کرده و به خروجی اضافه می‌کنیم. اگر x ورودی به زیرماژول و $\text{SubLayer}(x)$ خروجی آن زیرماژول باشد، در انتهای کار، ما عبارت زیر را محاسبه می‌کنیم:

$$x + \text{SubLayer}(x)$$

این جمع به صورت عنصر به عنصر (Element-wise Addition) انجام می‌شود.

۵.۴.۲ مزایای Connection Residual در ترانسفورمر

کمک به جریان یافتن گرادیان

وقتی ورودی مستقیماً به خروجی اضافه می‌شود، مسیری مستقیم برای عبور شیب (گرادیان) به عقب ایجاد می‌گردد. در صورت نبود این اتصال، اگر شبکه عمیق شود، گرادیان‌ها ممکن است در لایه‌های پایین محو شوند و عملاً vanishing gradient رخ می‌دهد.

حفظ اطلاعات اصلی (هویت ورودی)

حتی اگر زیرماژول تغییری در اطلاعات ورودی ایجاد کند، با وجود Connection، Residual ورودی اصلی همواره در خروجی نهایی حضور دارد. این ویژگی باعث می‌شود در صورت ناکافی بودن یادگیری زیرماژول یا در مراحل اولیه آموزش، دست‌کم بخشی از سیگنال/اطلاعات خام به لایه‌های بالاتر برسد.

کاهش ریسک تخریب ویژگی‌ها

در شبکه‌های عمیق، یکی از مشکلات این است که هر لایه ممکن است بخشی از اطلاعات مفید را تخریب کند. Connection Residual تضمین می‌کند که اگر لایه‌ای به هر دلیل نتوانست الگوی بهینه را یاد بگیرد، اطلاعات قبلی حداقل به صورت دست‌نخورده تا حدی منتقل می‌شود.

۵.۲ (Norm): Normalization Layer

در یادگیری عمیق، نرمال‌سازی (Normalization) داده‌های یک لایه یا فعال‌سازی‌ها، اغلب به سرعت بخشیدن به همگرایی و پایدار کردن آموزش کمک شایانی می‌کند. شاید معروف‌ترین نوع نرمال‌سازی، Normalization Batch باشد که پیش‌تر در کارهای بینایی (ها) (CNN) بسیار مورد استفاده قرار گرفت.

Normalization Layer روشی جایگزین است که در ترانسفورمر استفاده می‌شود. علت اصلی این انتخاب، ماهیت توالی‌محور (Sequence) بودن داده‌ها در NLP و عدم تمایل به وابستگی به آمار مینی‌بچ است.

تفاوت Norm Layer با Norm Batch

Normalization: Batch

در Norm، Batch برای نرمال سازی، میانگین و واریانس روی تمام نمونه های موجود در مینی بچ (و نیز در طول ابعاد ویژگی) محاسبه می شود. این موضوع در NLP کمی دردسرساز است؛ چون ترتیب (Order) توکن ها، طول جمله ها و حتی اندازه مینی بچ ممکن است نامنظم باشد. همچنین به خاطر تنوع طول توالی ها (Sequence)، (Length) پیاده سازی Norm Batch می تواند پیچیده شود.

Normalization: Layer

در Norm، Layer برای هر توکن به صورت جداگانه (در طول بُعد ویژگی)، میانگین و واریانس گرفته می شود. فرض کنید در یک لایه، بردار $h_i \in \mathbb{R}^{d_{\text{model}}}$ مربوط به توکن i باشد؛ یعنی ابعاد ویژگی آن d_{model} است. ما میانگین μ_i و واریانس σ_i^2 را از اجزای این بردار محاسبه می کنیم:

$$\mu_i = \frac{1}{d_{\text{model}}} \sum_{k=1}^{d_{\text{model}}} h_{i,k}, \quad \sigma_i^2 = \frac{1}{d_{\text{model}}} \sum_{k=1}^{d_{\text{model}}} (h_{i,k} - \mu_i)^2$$

سپس نرمال سازی برای هر مؤلفه k در بردار توکن i به شکل زیر انجام می شود:

$$\hat{h}_{i,k} = \frac{h_{i,k} - \mu_i}{\sqrt{\sigma_i^2 + \epsilon}}$$

در نهایت، برای این که مدل بتواند مقیاس و بایاس جدیدی یاد بگیرد، شبیه Norm، Batch دو

پارامتر γ (Scale) و β (Bias) نیز در طول بُعد ویژگی اعمال می شوند:

$$\text{LayerNorm}(h_i) = \gamma \odot \hat{h}_i + \beta$$

که در آن $\gamma, \beta \in \mathbb{R}^{d_{\text{model}}}$ هستند و \odot ضرب عنصر به عنصر است.

مزایای Normalization Layer در ترانسفورمر

- بی‌نیازی از وابستگی به ابعاد مینی‌بچ: با Norm، Layer می‌توان حتی با اندازه مینی‌بچ برابر ۱ نیز به‌خوبی آموزش دید، چراکه آمارها وابسته به ابعاد ویژگی‌اند و نه مینی‌بچ.
 - پایدارسازی توزیع فعال‌سازی‌ها: زمانی که مدل در حال یادگیری است، توزیع‌های داخلی لایه‌های میانی ممکن است تغییر کند (پدیده Covariate Internal Shift). Norm Layer Shift). با نرمال‌سازی این توزیع، آموزش را پایدارتر و سریع‌تر می‌کند.
 - سازگاری با داده‌های توالی‌محور: هر توکن را جداگانه نرمال می‌کند و نگرانی‌ای بابت ترتیب طول جمله‌ها، یا قرار گرفتن چند جمله کوتاه/بلند در یک مینی‌بچ نداریم.
- در معماری ترانسفورمر، پس از خروجی هر زیرماژول (مثل Attention یا MLP)، مراحل به‌شکل زیر است:
- Connection: Residual ابتدا ورودی همان زیرماژول (مثلاً بردار x) را با خروجی زیرماژول $(\text{SubLayer}(x))$ جمع می‌کنیم. حاصل این جمع را می‌توان چنین نوشت:

$$z = x + \text{SubLayer}(x)$$

این z حالا ترکیبی از اطلاعات اصلی ورودی و اطلاعات یادگرفته‌شده توسط SubLayer است. Normalization: Layer سپس این بردار z را وارد لایه LayerNorm می‌کنیم:

$$y = \text{LayerNorm}(z)$$

خروجی نهایی را می‌توان به لایه بعدی پاس داد یا به مرحله بعدی در همین لایه. به عبارتی اگر بخواهیم در یک فرمول واحد بیان کنیم:

$$\text{Norm \& Add} = \text{LayerNorm}(x + \text{SubLayer}(x))$$

۶.۲ decoder:

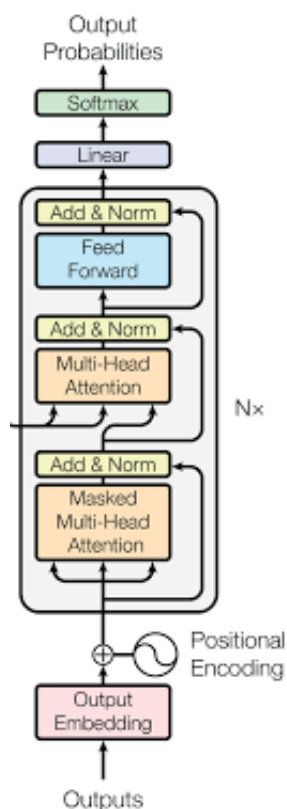
دیکودر در معماری ترانسفورمرها وظیفه تولید خروجی نهایی را بر عهده دارد. این خروجی معمولاً می‌تواند توالی هدف (Target) (Sequence) باشد، مثل ترجمه یک جمله یا پیش‌بینی توکن‌های بعدی در یک توالی. در این بخش دیکدر دو ورودی اصلی دارد: توالی هدف که معمولاً به صورت خودکار تولید می‌شود (مثلاً در ترجمه ماشینی یا تولید متن)، و نمایش (Representation) گذشته که توسط انکودر (Encoder) تولید شده است و شامل ویژگی‌های استخراج‌شده از توالی ورودی می‌باشد. دیکودر از این ورودی‌ها استفاده می‌کند تا به صورت گام‌به‌گام، خروجی نهایی خود را تولید کند.

همانطور که در تصویر مشاهده می‌کنید دیکدر دو ورودی دارد.

تمامی بخش‌های دیکدر مانند انکدر می‌باشند اما در دیکدر attention head multi masked وجود دارد.

۷.۲ attention head multi masked

در ترانسفورمر، مکانیزم Attention Multi-Head در بخش دیکودر به صورت Masked پیاده‌سازی می‌شود تا مدل نتواند توکن‌های آینده را ببیند و به صورت خودبازگشتی (Autoregressive) توکن بعدی را پیش‌بینی کند. در واقع ایده اصلی استفاده از mask جلوگیری از مشاهده آینده است. در معماری‌های خودبازگشتی (Autoregressive)، مدل در گام i از دیکودر تنها باید به توکن‌های قبلی $\{y_1, \dots, y_{i-1}\}$ دسترسی داشته باشد؛ اما نه به توکن‌های $\{y_{i+1}, y_{i+2}, \dots\}$. اگر مدل بتواند توکن‌های آینده را «نگاه» کند، پیش‌بینی توکن بعدی آسان و غیرواقعی می‌شود (مشکل نشت اطلاعات).



شکل ۲.۶.۷:

Decoder

به همین دلیل در Self-Attention Multi-Head Masked دیکودر، از یک ماتریس ماسک M استفاده می‌کنیم که اجازه نمی‌دهد هر توکن به توکن‌های آینده‌اش توجه کند.

۸.۲ مثال عددی mask attention:

فرض کنید دنباله ۴ توکنی داریم:

$$[y_1, y_2, y_3, y_4]$$

خروجی Dot-Product Scaled (قبل از softmax) یک ماتریس 4×4 خواهد بود:

$$S = \begin{bmatrix} s_{1,1} & s_{1,2} & s_{1,3} & s_{1,4} \\ s_{2,1} & s_{2,2} & s_{2,3} & s_{2,4} \\ s_{3,1} & s_{3,2} & s_{3,3} & s_{3,4} \\ s_{4,1} & s_{4,2} & s_{4,3} & s_{4,4} \end{bmatrix}$$

برای سطر ۱ (توکن اول): می‌تواند خودش (ستون ۱) را ببیند، اما ستون‌های ۲ تا ۴ را ماسک می‌کنیم. برای سطر ۲ (توکن دوم): می‌تواند به ستون‌های ۱ و ۲ نگاه کند، اما ۳ و ۴ ماسک می‌شوند. برای سطر ۳: ستون‌های ۱، ۲ و ۳ را ببیند، ستون ۴ ممنوع است. برای سطر ۴: ستون ۱، ۲، ۳، ۴ آزاد است. (چون چهارمین توکن می‌تواند توکن‌های قبلی را ببیند، و از طرفی این توکن «خودش» نیز موردی ندارد - بسته به پیاده‌سازی ممکن است تصمیم بگیریم توکن فعلی از خودش نیز استفاده کند یا نه؛ در معماری استاندارد، سطر i معمولاً به ستون i هم دسترسی دارد).

در عمل، ماتریس ماسک M به شکل زیر خواهد بود (اگر به شکل پایین‌مثلثی نشانه‌گذاری کنیم):

$$M = \begin{bmatrix} 0 & -\infty & -\infty & -\infty \\ 0 & 0 & -\infty & -\infty \\ 0 & 0 & 0 & -\infty \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

به این ترتیب، پس از جمع شدن با S و اجرای softmax در هر سطر، ضرایب توجه‌ی ستون‌های ماسک‌شده به صفر میل می‌کنند.

۹.۲ transformer: vision

ایده ترانسفورمرها در تصویر از تعمیم دادن ترانسفورمر متن به وجود آمده است.

ما در این بخش transformer vision را در کلاس بندی استفاده می‌کنیم.

در روش‌های متداول برای پردازش تصویر از convolution ها استفاده می‌کردند. اما در

ترانسفورمرها عکس‌ها به پیچ‌های مختلف شکسته می‌شوند. و این قسمت‌های شکسته شده

عکس به یک دیگر توجه میکنند که چقدر به یک دیگر شباهت دارند. در قسمت های بعد به طور مفصل به این کار ها میپردازیم.

۱.۹.۲ transformer: vision in embedding patch

در ترانسفورمر های مبتنی بر متن هر کلمه به توکن تبدیل می شود. و هر کدام از این کلمات به بردار هایی تبدیل میشود. و این بردار ها بعد از اضافه شده embedding positional وارد Attention در ترانسفورمر ها میرسید.

حال همین ایده در تصویر پیاده سازی شده است. همانطور که در تصویر ... مشاهده می کنید. در Transformer، Vision به جای عملیات کانولوشن، مستقیماً تصویر را به بلاک های غیرهم پوشان $(P \times P)$ قطعه بندی می کنیم تا موازی سازی بهتری داشته باشیم و به مدل اجازه دهیم از سازوکار Self-Attention (توجه سراسری) برای ارتباط بین این بلاک ها استفاده کند.



شکل ۲.۹.۸: patch to image

۲.۹.۲ شکل پیچ ها:

فرض کنید ابعاد تصویر ورودی $(H \times W \times C)$ است. به عنوان مثال: فرض کنیم اندازه تصویر ما $224 \times 224 \times 3$ باشد. یعنی طول و عرض تصویر به ترتیب ۲۲۴ و سه کانال رنگی داشته باشد.

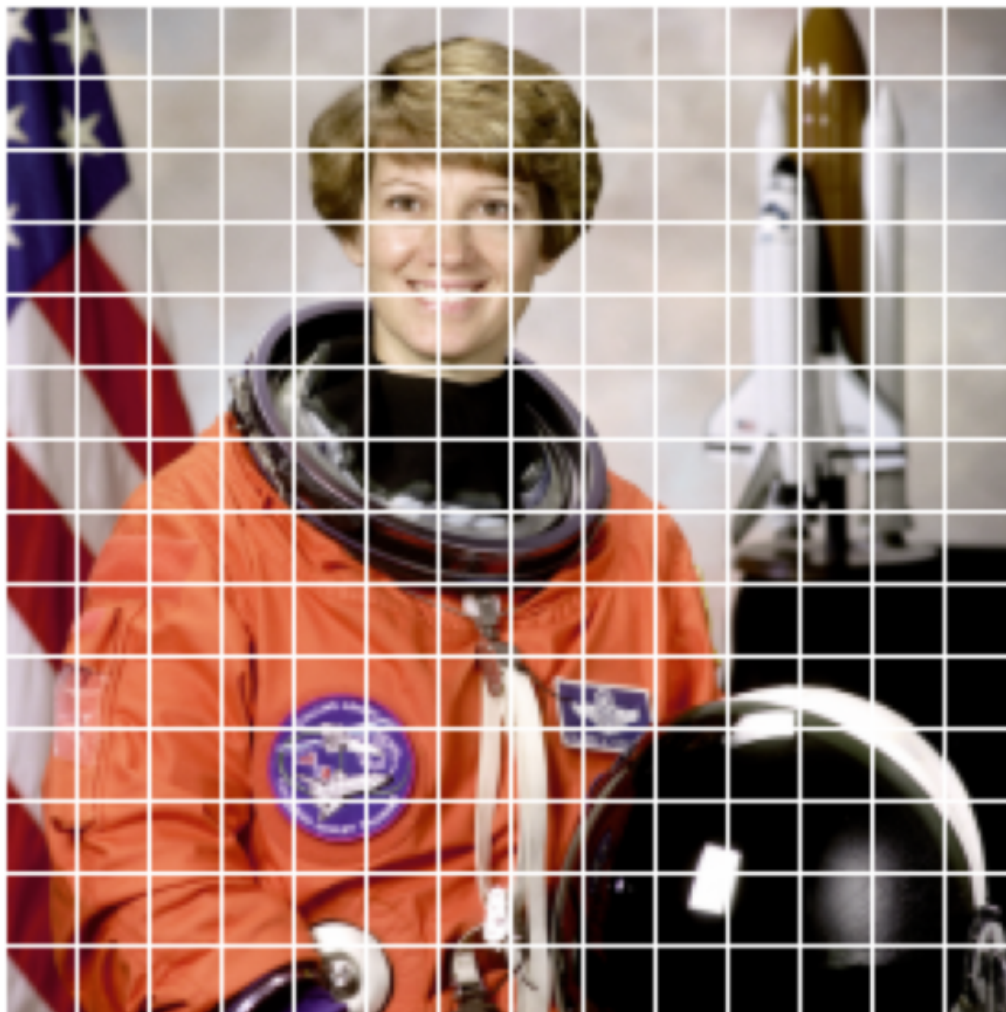
$$H = 224, \quad W = 224, \quad C = 3$$



شکل ۲.۹.۹: Image original

حال اگر اندازه هر پچ $(P \times P)$ باشد (مثلاً 16×16)، تصویر به صورت یک جدول مشبک از پچ‌های کوچک تقسیم می‌شود.

به هر پچ می‌توان مانند یک «کاشی» از تصویر نگاه کرد: پچ اول: مختصات (۰ تا ۱۵ در ارتفاع) و (۰ تا ۱۵ در عرض)، پچ دوم: مختصات (۰ تا ۱۵ در ارتفاع) و (۱۶ تا ۳۱ در عرض)، و به همین ترتیب تا کل تصویر پوشش داده شود.



شکل ۲.۹.۱۰: Image patch

۳.۹.۲ تعداد پچ ها:

اگر پچ های ما بدون هم پوشانی باشند، ابعاد پچ دقیقاً باید بر ابعاد تصویر بخش پذیر باشد.

تعداد پچ ها افقی:

$$\frac{W}{P}$$

تعداد پچ ها عمودی:

$$\frac{H}{P}$$

در مجموع:

$$\left(\frac{H}{P}\right) \times \left(\frac{W}{P}\right) = \frac{H}{P} \times \frac{W}{P}.$$

برای مثال اگر:

$$H = 224, \quad W = 224, \quad P = 16 :$$

$$\frac{224}{16} = 14 \Rightarrow 14 \times 14 = 196 \quad (\text{تعداد پچ‌ها}).$$

تر اکثر نسخ‌های مبدل‌های بینایی، چ‌ها بدون هم پوشانی هستند. (Non-overlapping) اندازه پچ‌های کوچک باعث می‌شود، تعداد پچ‌ها زیاد شوند. و با زیاد شدن تعداد پچ‌ها، هزینه attention زیاد می‌شود. و هم چنین اندازه پچ‌های بزرگ باعث می‌شود تعداد پچ‌ها کمتر شود، و هزینه‌های attention کمتر شود. اما در پچ‌های بزرگ باعث می‌شود که جزییات محلی (local) از بین برود.

۴.۹.۲ بردار کردن هر پچ

هر پچ دارای ابعاد $(P \times P \times C)$ است. برای مثال اگر $P = 16$ و $C = 3$ ، پچ ابعاد $16 \times 16 \times 3$ دارد. برای این‌که بتوانیم پچ‌ها را مانند «توکن»های NLP به مدل ترانسفورمر بدهیم، باید آن‌ها را به یک بردار یک‌بعدی تبدیل کنیم. اگر بخواهیم همه پیکسل‌های پچ را به صورت ردیفی (Row-major) دنبال هم بگذاریم، طول این بردار خواهد بود:

$$P \times P \times C = P^2 \times C.$$

در مثال $16 \times 16 \times 3$ ، طول بردار می‌شود 768.

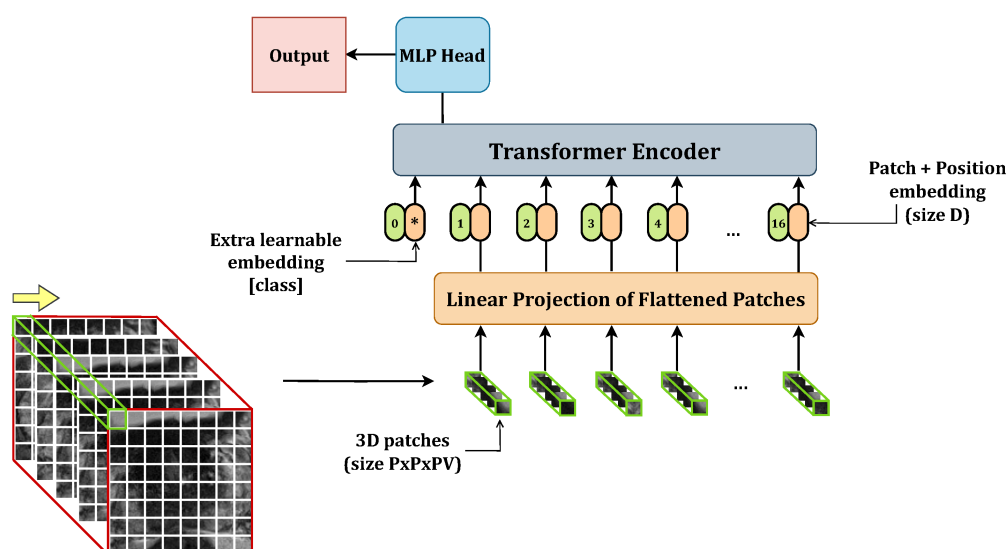
۱۰.۲ اعمال لایه خطی: (Projection)

بعد از Flatten کردن، معمولاً یک لایه خطی (Fully-Connected Layer) روی این بردار اعمال می‌شود تا آن را به بعد d_{model} (مثلاً ۷۶۸ یا ۱۰۲۴) برسد. در حقیقت، این لایه تبدیل Feature

(Transformation) را انجام می‌دهد تا همهٔ پچ‌ها یک نمایندگی (Embedding) با ابعاد یکنواخت

d_{model} پیدا کنند:

$$(P^2 \times C) \rightarrow d_{\text{model}}.$$



شکل ۲.۱۰.۱۱: Transformer Vision in Embedding

این مرحله شبیه ساخت توکن در NLP است؛ با این تفاوت که در NLP توکن «کلمه» یا «زیرکلمه» است و پیش‌تر دارای بردار تعبیه‌شده Embedding شده بوده است. در trans-vision former ما ابتدا باید تصاویر را پچ کنیم و سپس بردارهای embedding شده را از این پچ‌ها به دست آوریم.

ترانسفورمر نیاز دارد ورودی‌اش توالی توکن‌ها باشد. در NLP توالی کلمات داریم، در ViT توالی «پچ»‌ها:

$$\{x_{\text{patch}_1}, x_{\text{patch}_2}, \dots, x_{\text{patch}_N}\}.$$

هر پچ اکنون یک بردار d_{model} -بعدی است. پس یک مجموعه با طول N (تعداد پچ‌ها) و عرض d_{model} خواهیم داشت. اگر عدد پچ‌ها N باشد (مثلاً ۱۹۶)، ترانسفورمر می‌تواند خودش با استفاده از مکانیزم Self-Attention، وابستگی (Relations) میان پچ‌ها را یاد بگیرد: کدام بخش از

تصویر برای کدام بخش دیگر مهم‌تر است، چگونه ترکیب جهانی (Global Context) ساخته شود، و غیره.

معمولاً پچ‌ها را به صورت ردیفی (Row by Row) شماره گذاری می‌کنند (ابتدا پچ‌های ردیف بالایی از چپ به راست، سپس ردیف بعدی و ...)، تا مدل در صورت نیاز بتواند از پوزیشن‌ها اطلاعات مکانی تقریبی داشته باشد. در عمل، چون قصد داریم (در مراحل بعد) به هر پچ یک Embedding Positional هم اضافه کنیم، مکان دقیق هر پچ در بعد دوم (ویژگی) رمز می‌شود. در ویژن ترانسفورمر دیگر به کانولوشن وابسته نیست. در عوض از embedding استفاده می‌شود.

Split کردن تصویر به بلاک‌های $(P \times P)$ ، Flatten و Linear Projection، عملیات ریاضی ساده‌ای هستند و به راحتی قابل موازی‌سازی روی GPU/TPU هستند.

۱.۱.۲ Token: CLS

Token CLS یک بردار ویژه است که به ابتدای دنباله ورودی اضافه می‌شود و نقش آن این است که اطلاعات کلی و مرتبط با دنباله ورودی (چه متن، چه تصویر) را در خود خلاصه کند. token cls در به ابتدای پچ‌های تصویری قرار می‌گیرد.

این توکن یک بردار با ابعاد d_{model} است (همان ابعاد سایر توکن‌ها).

بردار CLS یک پارامتر یادگرفتنی است، یعنی مدل در طول آموزش مقادیر آن را برای ذخیره و جمع‌آوری اطلاعات بهینه می‌کند.

در وظایف دسته‌بندی، (Classification) هدف اصلی این است که یک پیش‌بینی کلی برای کل ورودی (مثلاً یک جمله یا یک تصویر) ارائه دهیم. Token CLS دقیقاً همین وظیفه را بر عهده دارد.

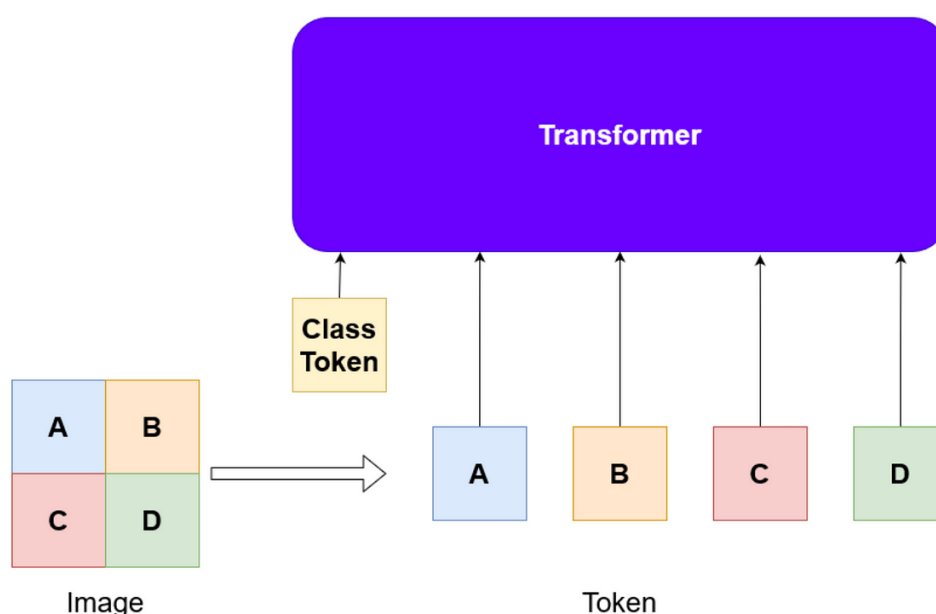
Token CLS از طریق مکانیزم Self-Attention در ترانسفورمر می‌تواند با همه توکن‌های دیگر (یعنی پچ‌های تصویر) ارتباط برقرار کند و اطلاعات مهم آن‌ها را در طول لایه‌های ترانسفورمر

به صورت تدریجی یاد بگیرد. و به عنوان نماینده ای از تمام تصویر یا متن ها در مدل حضور پیدا کند.

Token CLS از طریق ضرب داخلی در مکانیزم Attention می تواند به تمام پچ ها نگاه کند. ضرایب توجه (α) تعیین می کند که Token CLS چه مقدار اطلاعات از هر پچ دریافت کند. و چقدر در پروسه مدل دخیل باشد.

Token CLS به طور ضمنی یاد می گیرد که روی ویژگی هایی که برای دسته بندی مهم هستند (مانند الگوها، اشکال و بخش های کلیدی تصویر) تمرکز کند.

در طول لایه های ترانسفورمر، Token CLS نقش محوری در تنظیم بازنمایی کل تصویر ایفا می کند. به عبارت دیگر، این توکن به نوعی مرکز پردازش کل اطلاعات تصویر است.



شکل ۲.۱۰.۱۲: Transformer Vision in Token Cls

Token cls به عنوان پارامترهای یادگیرنده تعریف میشود و این پارامترها در طول فرایند یادگیری بروز رسانی میشوند.

فصل ۳

روش های پیشنهادی

در این پژوهش ما دو روش را بهبود داده ایم :

فصل ۴

آزمایشات و نتایج

کتاب نامه

پیوست آ

جزئیات مدل‌ها و جدول پارامترها

Abstract

Recently, graph neural networks (GNNs) have shown success at learning representations of functional brain graphs derived from functional magnetic resonance imaging (fMRI) data.

Key Words: Clustering , DBSCAN , Voronoi diagrams , Delaunay triangulation , Outlier detection .



Enhancing MDBSCAN and MOGA-DBSCAN

A Thesis Presented for the Degree of
Master in Computer Science

Faculty of Mathematical Sciences

Tarbiat Modares University

by

Seyed Mohammad Badzohreh

Supervisor

Dr. Mansoor Rezghi

2024