دانشگاه تربیت مدرس

دانشکده علوم ریاضی

پایان‌نامه دوره کارشناسی ارشد علوم کامپیوتر

روش های عمیق مبتنی برمبدل 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

آن ها که از خواسته هایشان گذشتند، سختی ها را به جان خریدند و خود را سپر بلای مشکلات و ناملایمات کردند تا من به جایگاهی که اکنون در آن ایستاده ام برسم .

قدردانی

از استاد گرانقدر، جناب آقای دکتر رزقی که با راهنمایی‌های دلسوزانه و ارزشمند خود، همواره در مسیر تحقیق این پایان نامه یار و راهنمای من بودند، نهایت سپاس و قدردانی را دارم.

از خانواده عزیزم که با محبت بی‌پایان، صبوری و حمایت های بی دریغ شان، همواره پشتیبان من در طی این مسیر سخت و پرچالش بودند، صمیمانه سپاسگزارم.

سید محمد بادزهره

پاییز ۱۴۰۳

چکیده

ببعلعذدقفلهعقفدلخقفدللقفلقفاقا

فهرست مطالب

فهرست جداول

فهرست تصاویر

پیش‌گفتار

قدثثمقدکنقصدبثقلدقفخدلقخفادخفادخ

فصل ۱

مفاهیم اولیه

در این فصل به معرفی مقدمات و مفاهیم مورد نیاز در این پایان‌نامه می‌پردازیم.

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 مکارتی در سال ۱۹۵۶ این اصطلاخ 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 AGI (Artificial General Intelligence) شروع به کار کردند

۲.۱.۱ دورهٔ طلایی و پیشرفت‌های اولیه

در دهه ۵۰ و۶۰ میلادی، هوش مصنوعی به عنوان یکی از پرچمداران پژوهش های نوین شناخته می شد. الگوریتم های اولیه به کمک روش های منطقی و ریاضیاتی برای حل مسئله و بازی های ساده توسعه یافتند مانند انواع الگوریتم جستوجوی درختی که در این دوره به وجود آمدند و زمینه ساز اولین دستاوردهای هوش مصنوعی در بازی های تخته ای همچون شطرنج شدند. در این دوران پیشرفت های بیشتری در پردازش زبان طبیعی (NLP) و سیستم های 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 پیشرفت های های هوش مصنوعی، محدودیت های تکنولوژی مثل gpu ها و محاسباتی در آن زمان و همچنین کمبود داده های کافی برای آموزش مدل های پیچیده تر، باعث شد که بسیاری از پروژه های تحقیقاتی نتوانند به نتایج پیش بینی شده دست یابند. و در نتیجه، هوش مصنوعی در دهه ۷۰ به مرحله ای از رکود وارد شد که به آن عصر تاریک هوش مصنوعی یا AI) (Winter می گویند. در این دوران بسیاری از پروژه ها تعطیل و سرمایه گذاری ها قطع شدند و دولت ها و سازمان های سرمایه گذار به دلیل عدم دستیابی به نتایج مطلوب از ادامه سرمایه گذاری 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 نمی توانستند تعمیم پذیری خیلی بالایی داشته باشند. [؟].

۵.۱.۱ پایان عصر تاریک و بازگشت هوش مصنوعی

پس از چندین سال رکود و عدم سرمایه گذاری در حوزه هوش مصنوعی، سرانجام در دهه ۱۹۸۰ و ۱۹۹۰ عصر تاریک هوش مصنوعی با تحولات تکنولوژی و از همه مهم تر ظهور سیستم های خبره (Expert-Systems) به پایان رسید. سیستم های خبره به عنوان یکی از اولین تلاش های موفق برای کاربردهای صنعتی در هوش مصنوعی به وجود آمدند.بر خلاف الگوریتم های اولیه، این سیستم هااز پایگاه بزرگ قواعد و قوانین (Rule-Based-Systems) استفاده میکردند. در سیستم های خبره به جای تلاش برای شبیه سازی کلی هوش مصنوعی، بر حل مسائل تخصصی برای صنایع و سازمان ها تمرکز میکردند. برای مثال، سیستم های خبره در پزشکی برای تشخیص بیماری ها و پیشنهاد درمان، در صنعت برای مدیریت و پیش بینی خرابی ماشین آلات، و در امور مالی برای تحلیل و ارزیابی ریسک کاربرد داشتند. هر چند این سیستم ها نمی توانستند درک عمیق و هوشمندی عمومی را ایجاد کنند. اما برای رفع نیاز های پیچیده مناسب بودند. همزمان با موفقیت این سیستم ها، بهبودهای زیادی در سخت افزارها و کاهش هزینه های پردازش به وجود آمد. در دهه های ۱۹۸۰ و ۱۹۹۰ کامپیوتر ها به تدریج قوی تر و مقرون به رفه تر شدند و امکان پردازش داده های بیشتر و اجرای الگوریتم های پیچیده تر فراهم شد. این افزایش توان محاسباتی، نیاز به پردازش داده های بزرگ و پیچیده را فراهم کرد و در نتیجه دسترسی به داده ها و انجام محاسبات سنگین برای توسعه الگوریتم های جدید فراهم شد. از طرف دیگر، پیشرفت های انجام شده در ذخیره سازی 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) شاخه‌ای 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 تقسیم‌بندی‌های اصلی در یادگیری ماشین

یادگیری ماشین به سه دستهٔ اصلی تقسیم می‌شود: 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)

طبقه بندی یکی از مهم ترین و اصلی ترین وظایف در یادگیری نظارت شده است که هدف آن تخصیص داده ها به یک لیبل مشخص است. در این روش مدل با داده های برچسب دار (label) آموزش میبیند و یاد میگیرد که داده های جدید را بر اساس الگو ها و ویژگی هایی که در داده های آموزشی دیده است، به دسته مناسب اختصاص دهد. از کاربرد های طبقه بندی می توان به تشخصیص اسپم ،(spam) تشخیص بیماری که آیا یک فرد مبتلا به بیماری هست یا نه، تشخصیص چهره و ... استفاده کرد.

رگرسیون (regression)

رگرسیون یکی از مهم ترین وظایف یادگیری ماشین است و هدف آن پیش بینی مقادیر پیوسته است. بر حلاف طبقه بندی که خروجی آن دسته بندی مجزا است، در رگرسیون، خروجی یک مقدار پیوسته است و مدل یاد میگیرد روابط بین متغیر های مستقل و متغیر های هدف را شناسایی کند. از کاربرد های رگرسیون میتوان به پیش بینی قیمت مسکن، پیش بینی اب 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 می دهد، 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 می کند چه اقداماتی باعث نتیجه بهینه میشود. از کاربرد های یادگیری تقویتی میتوان به بازی ها ،(games) کنترل رباتیک (robotic control-) ، سیستم های توصیه گر (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 اندازه‌گیری فاصله بین نقاط داده کار می‌کند و بدون نیاز به آموزش مدل پیچیده قابل استفاده است.
- عملکرد خوب در داده‌های با تعداد ویژگی کم: در مسائلی که تعداد ویژگی ها کم است این الگوریتم به خوبی کار میکند.

معایب:

- حساسیت به داده‌های پرت: نقاط پرت می‌توانند به‌طور قابل توجهی بر نتایج تاثیر بگذارند.
- کندی در داده‌های بزرگ: این الگوریتم نیاز به محاسبه فاصله برای هر نقطه جدید دارد که در داده‌های بزرگ بار محاسباتی سنگین می‌شود.
- عدم کارایی در داده‌های با ابعاد بالا: در داده‌های با تعداد ویژگی‌های زیاد، کارایی الگوریتم کاهش می‌یابد.

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

الگوریتمی است که با یافتن یک ابر صفحه بهینه، داده ها را به کلاس مختلف تقسیم میکند. این الگوریتم یک ابر صفحه به دست می آورد که هدف آن حداکثر کردن فاصله میان داده های 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 ویژگی ها 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 میگبرد.

مزایا:

- سرعت بالا: به دلیل محاسبات ساده و فرض استقلال ویژگی‌ها، Bayes Naive بسیار سریع و کم‌حجم 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 کوتاه‌مدت (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 نمتوانستند 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.

ساختار و عملکرد :RNN

شبکه های Rnn داری 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 نورون داده می‌شود.

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

در اینجا:

h_t وضعیت مخفی یا حالت در گام زمانی t است.

W وزن‌هایی است که به ورودی x_t اعمال می‌شود.

U وزن‌های اعمال‌شده بر وضعیت قبلی h_{t-1} است.

b بایاس مدل است.

σ تابع فعال‌سازی، 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 Gradient): Exploding and 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 طولانی‌مدت دچار مشکل است و برای پردازش وابستگی‌های طولانی، عملکرد ضعیفی دارد.

۱۱.۲.۱ شبکه‌های حافظه بلندمدت_ کوتاه‌مدت :(LSTM)

علل پیدایش :lstm

شبکه های lstm به عنوان یک راه حل برای یکی از بزرگترین مشکلات شبکه های عصبی بازگشتی (RNN) ها معرفی شدند. یکی از بزرگترین مشکلاتی که در شبکه های بازگشتی وجود داشت مشکل ناپدید شدن گرادیان (Vanishing Gradient) بود. این مشکل باعث میشد RNN ها قادر به یادگیری وابستگی های 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، هاRNN در یادگیری 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 طولانی‌مدت در داده‌ها می‌شود.

۱۲.۲.۱ ظهور Lstm:

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

راه‌حل 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.

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

مقدار فراموشی برای هر عنصر از وضعیت سلولی. : f_t

تابع سیگموید 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.

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

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

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.

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

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

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 سیگموید و تانژانت هیپربولیک: توابع سیگموید 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 ترانسفومرها

شبکه های بازگشتی rnn ، lstm ها توانستند بسیازی از مشکلات و محدودیت های مدل های اولیه را حل کنند اما همچنان با جالش ها و محدودیت هایی مواجهه بودند که در مسائل پیچیده تر، مانند ترجمه زبان یا تحلیل داده های بلند مدت و حجیم، مشکلاتی را ایجاد میکردند. این مشکلات در نهایت باعث پیدایش ترانسفورمر ها شد. که در اینجا مشکلات آن ها را بررسی میکنیم

مشکل وابستگی ترتیبی در هاRNN و هاLSTM

هاRNN و هاLSTM 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 RNNها، 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 گرادیان‌ها به تدریج ضعیف‌تر می‌شوند.

پیچیدگی محاسباتی و حافظه

LSTMها 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 گرادیان‌ها، 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

هاRNN و هاLSTM 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. هاRNN و هاLSTM 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 هاRNN و هاLSTM 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-Nها و هاLSTM 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، هاRNN و هاLSTM 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 باعصث به وحود 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 مدل های 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

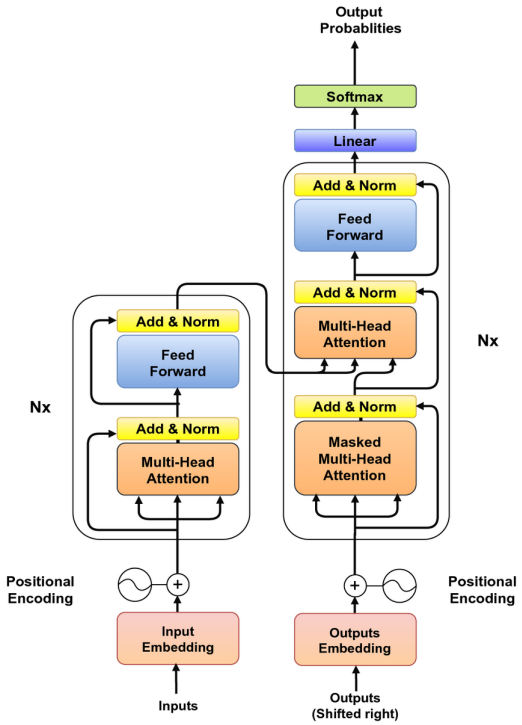
از داده‌های ترجمه‌شده برای آموزش مدل‌های آماری استفاده می‌کردند که احتمال ترجمه‌ای صحیح را براساس شواهد آماری محاسبه می‌کردند. مدل‌هایی مانند مدل‌های ترجمه آماری مبتنی بر جمله Models) Statistical (Phrase-based از این نوع بودند، که قادر به ترجمه جملات بهتر از مدل‌های مبتنی بر قواعد بودند، اما هنوز هم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

۳.۲ ظهور ترانسفورمر ها:

در سال ۲۰۱۷ مقاله ... توسط گوگل منتشر شد که مفهوم جدیدی به نام ترانسفورمر ها را بیان کرد. این مقاله به موضوع ترجمه ماشینی پرداخت و نشان داده با استفاده از مفهوم مکانیزم توجه (Attention-Mechanism) می توان بسیاری از مشکلات مدل های قبلی را حل کرد. این مدل ها از پردازش موازی استفاده میکردند بر خلاف مدل های قبلی که از پردازش های سریالی استفاده میکردند. و این مدل ها قادر هستند به طور همزمان به تمام بخش های ورودی توجه کند. و این کار باعث شد که ترانسفورمر ها در پردازش تصویر و متن بسیار سریع تر و دقیق تر از مدل های قبلی عمل کنند.

۴.۲ معماری ترانسفورمر ها:

در تصویر ... معماری ترانسفورمر نمایش داده شده است و بخش هاو 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 میگیرد و ویژگی های آن را استخراج میکند و هدف دیکدر این است که ویژگی های استخراج شده را به زبان مقصد تبدیل میکند. که به طور مختصر به معماری و بخش های مختلف آن میپردازیم.



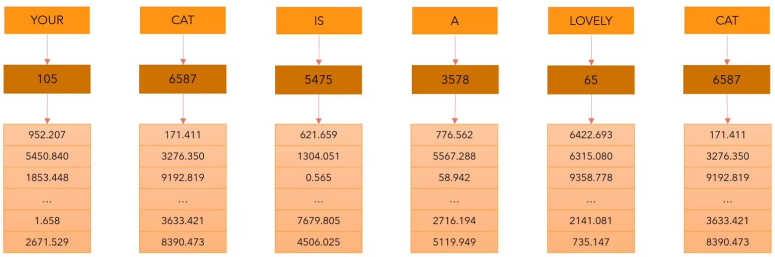
شکل ۲.۴.۱ : معماری ترانسفورمر ها

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 Word۲Vec 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.



شکل ۲.۴.۲: embedding word

۲.۴.۲ embedding: positional

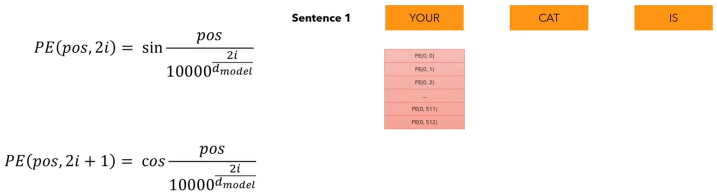
ما تا الان هر کلمه را به برداری از اعداد که برای مدل قابل فهم باشد، تبدیل کرده ایم. اما مدل های ترانسفورمر نمیتوانند جایگاه هر کلمه را تشخیص دهند. در مدل های ترانسفومر بر خلاف مدل های بازگشتی به دلیل این که کلمه های به صورت موازی وارد میشوند نیاز داریم تا جایگاه هر کلمه را بدانیم به طور مثال جمله من تو رو دوست داریم باید به طور دقیق بدانیم کلمه من، کلمه اول حمله 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 embedding positional به مدل وجود دارد که در ترانسفومرر ها از روش Embedding Positional Sinusoidal استفاده میشود. این روش قابل یادگیری نیست و صرفا از یک سری فرمول های ساده برای embedding positional استفاده میکند.

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

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

و برای مقادیر فرد:

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



شکل ۲.۴.۳: embedding word

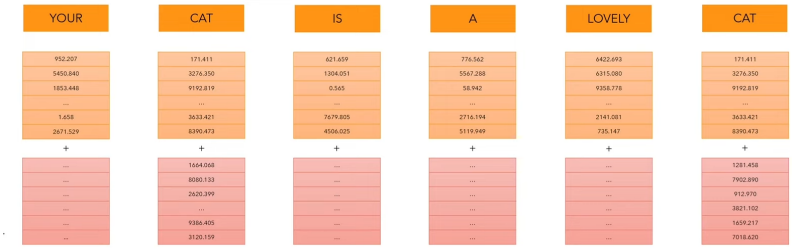
- pos: موقعیت کلمه در توالی است. برای مثال، اگر توالی ورودی شامل N کلمه باشد، موقعیت‌ها از 0 تا $N-1$ تغییر می‌کنند. به عبارت دیگر، pos می‌تواند هر عددی از مجموعه $\{0, 1, 2, \ldots, N-1\}$ باشد که نشان‌دهنده موقعیت یک کلمه خاص در توالی است.

- 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 به‌گونه‌ای 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 positional embed- 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 الگو های 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.



شکل ۲.۴.۴: embedding word

۳.۴.۲ attention:

در روش‌های قدیمی (مانند RNN یا lstm)، توالی ورودی (مثلاً یک جمله) معمولاً به‌صورت گام‌به‌گام پردازش می‌شد. اما در ترانسفورمر می‌خواهیم مدلی داشته باشیم که به هر موقعیت (مثلاً یک کلمه) در توالی نگاه کند و به همهٔ موقعیت‌های دیگر نیز به‌صورت موازی دسترسی داشته باشد. به این مفهوم توجه می‌گوییم. م به زبان ساده، وقتی توکن (کلمه) i به توکن‌های دیگر نگاه می‌کند، می‌خواهد بداند کدام توکن‌ها برای تفسیر معنای خودش مهم‌ترند.

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

$$Q = \text{(پرسش) Query}, \quad K = \text{(کلید) Key}, \quad V = \text{(مقدار / ارزش) Value}$$

در Attention Dot-Product Scaled، ابتدا شباهت یا ارتباط بین Query و Key را با محاسبهٔ ضرب داخلی (Dot Product) به‌دست می‌آوریم، سپس آن را نرمال می‌کنیم (با تقسیم بر d_k) و از تابع softmax استفاده می‌کنیم تا ضرایب توجه (Attention Weights) را به‌دست آوریم. در نهایت با همین ضرایب، ترکیبی خطی از بردارهای Value را می‌گیریم.

فرمول به‌شکل زیر است:

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

که در آن:

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

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

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

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

تقسیم بر d_k باعث می‌شود مقدار ضرب داخلی در ابعاد بالا خیلی بزرگ نشود و شیب‌ها (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 iـ‌ام 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: به‌جای آنکه فقط یک‌بار 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 به‌صورت موازی محاسبات Attention را انجام می‌دهیم. سپس خروجی همهٔ این هاHead 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 به‌شکل زیر است:

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

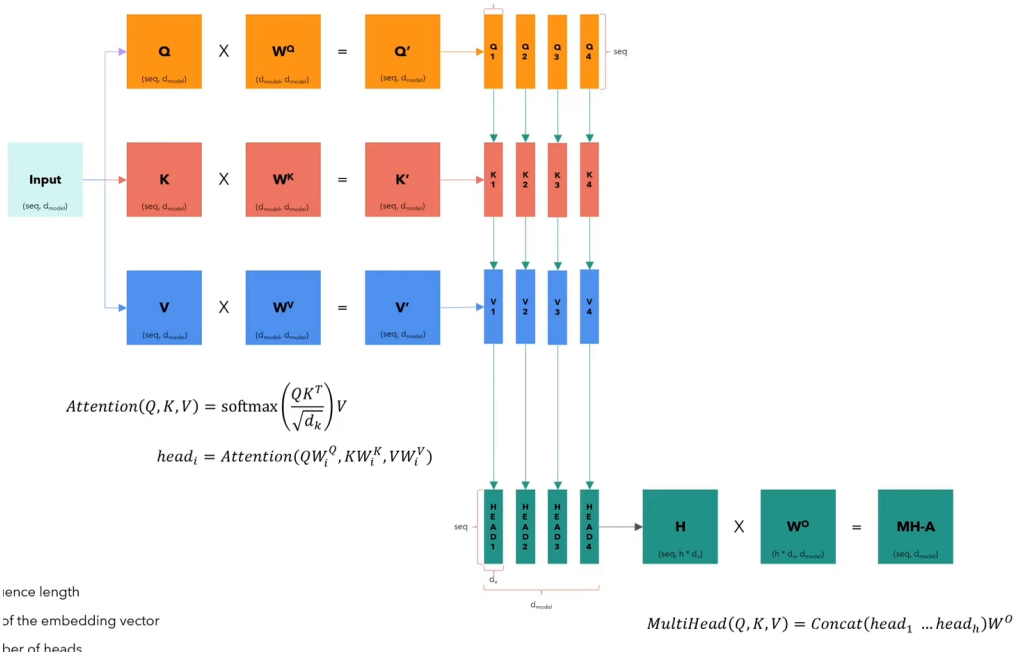
$$\text{MultiHead}(Q, K, V) = [\text{head}_1 \oplus \cdots \oplus \text{head}_h]W_O$$

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

ماتریس وزن W_O به‌شکل زیر است:

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

که W_O ماتریسی است که خروجی الحاق‌شده را به بعد d_{model} برمی‌گرداند.



شکل ۲.۴.۶: attention head multi

چرا چندین سر؟

مشاهدهٔ چند منظر متفاوت: هر Head می‌تواند الگوهای گوناگونی از وابستگی‌ها را بیاموزد (مثلاً یک Head می‌تواند یاد بگیرد کلمهٔ فعلی با کلمات همسایهٔ نزدیک خود بیشتر مرتبط شود، یک Head دیگر روی ارتباط با کلماتی در فاصلهٔ 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 هاHead $h = 8$، 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 Connection Residual (Add):

در معماری های های مختلف هنگامی که تعداد لایه ها 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 شوندو عملا vanishing gradient رخ میدهد.

حفظ اطلاعات اصلی (هویت ورودی)

حتی اگر زیرماژول تغییری در اطلاعات ورودی ایجاد کند، با وجود Residual Connection، ورودی اصلی همواره در خروجی نهایی حضور دارد. این ویژگی باعث می‌شود در صورت ناکافی بودن یادگیری زیرماژول یا در مراحل اولیهٔ 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. Residual Connection 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 (CNNها) بسیار مورداستفاده 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 نرمال‌سازی، میانگین و واریانس روی تمام نمونه‌های موجود در مینی‌بچ (و نیز در طول ابعاد ویژگی) محاسبه می‌شود. این موضوع در NLP کمی دردسرساز است؛ چون ترتیب (Order) توکن‌ها، طول جمله‌ها و حتی اندازهٔ مینی‌بچ ممکن است نامنظم باشد. همچنین به خاطر تنوع طول توالی‌ها (Sequence Length)، پیاده‌سازی Norm Batch می‌تواند پیچیده شود.

Normalization: Layer

در Norm، Layer برای هر توکن به‌صورت جداگانه (در طول بُعد ویژگی)، میانگین و واریانس گرفته می‌شود. 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 می‌کنیم:

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

سپس نرمال‌سازی برای هر مؤلفهٔ k در بردار توکن i به شکل زیر انجام می‌شود:

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

در نهایت، برای این‌که مدل بتواند مقیاس و بایاس جدیدی یاد بگیرد، شبیه Norm، Batch دو پارامتر γ (Scale) و β (Bias) نیز در طول بعد ویژگی اعمال می‌شوند:

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

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 Layer ،Norm می‌توان حتی با اندازهٔ 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 (پدیدهٔ Covariate Internal Shift). Norm Layer 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 جملهٔ کوتاه/بلند در یک مینی‌بچ نداریم.

در معماری ترانسفورمر، پس از خروجیِ هر زیرماژول (مثل Attention یا MLP)، مراحل به‌شکل زیر است:

Connection: Residual ابتدا ورودی همان زیرماژول (مثلاً بردار x) را با خروجی زیرماژول (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 لایهٔ LayerNorm می‌کنیم:

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

خروجی نهایی را می‌توان به لایهٔ بعدی پاس داد یا به مرحلهٔ بعدی در همین لایه.

به‌عبارتی اگر بخواهیم در یک فرمول واحد بیان کنیم:

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

decoder: ۶.۲

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 ابن 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) کدشده 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.

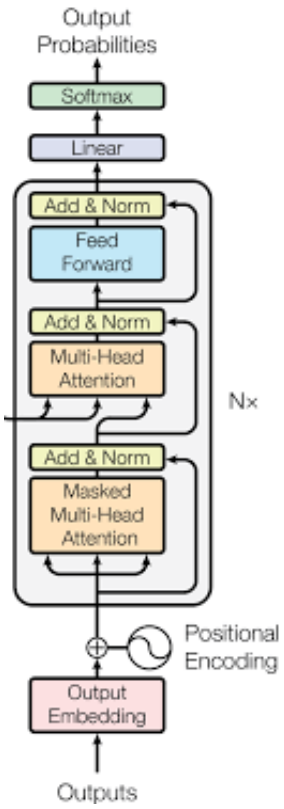
همانطور که در تصویر مشاهده میکنید دیکدر دو ورودی دارد.

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

attention head multi masked ۷.۲

در ترانسفورمر، مکانیزم Attention Multi-Head در بخش دیکودر به‌صورت Masked پیاده‌سازی می‌شود تا مدل نتواند توکن‌های آینده را ببیند و به‌صورت 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 $\{y_1, \ldots, y_{i-1}\}$ دسترسی داشته باشد؛ اما نه به توکن‌های $\{y_{i+1}, y_{i+2}, \ldots\}$. 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 attention: mask

فرض کنید دنبالهٔ 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 ترانسفورمر ها در تصویر از تعمیم دادن ترانسفومر متن به وجود آمده است.

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

در روش های متداول برای پردازش تصویر از convolution ها استفاده می کردند. اما در ترانسفورمر ها عکس ها به پچ های مختلف شکسته می شوند. و این قسمت های 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 Vision Transformer، به‌جای عملیات کانولوشن، مستقیماً تصویر را به بلاک‌های غیرهم‌پوشان ($P \times P$) قطعه‌بندی می‌کنیم تا موازی‌سازی بهتری داشته باشیم و به مدل اجازه دهیم از سازوکار Self-Attention (توجه سراسری) برای ارتباط بین این بلاک‌ها استفاده کند.

شکل ۲.۹.۸: patch to iamge

۲.۹.۲ شکل پچ ها:

فرض کنید ابعاد تصویر ورودی ($H \times W \times C$) است. به‌عنوان مثال:

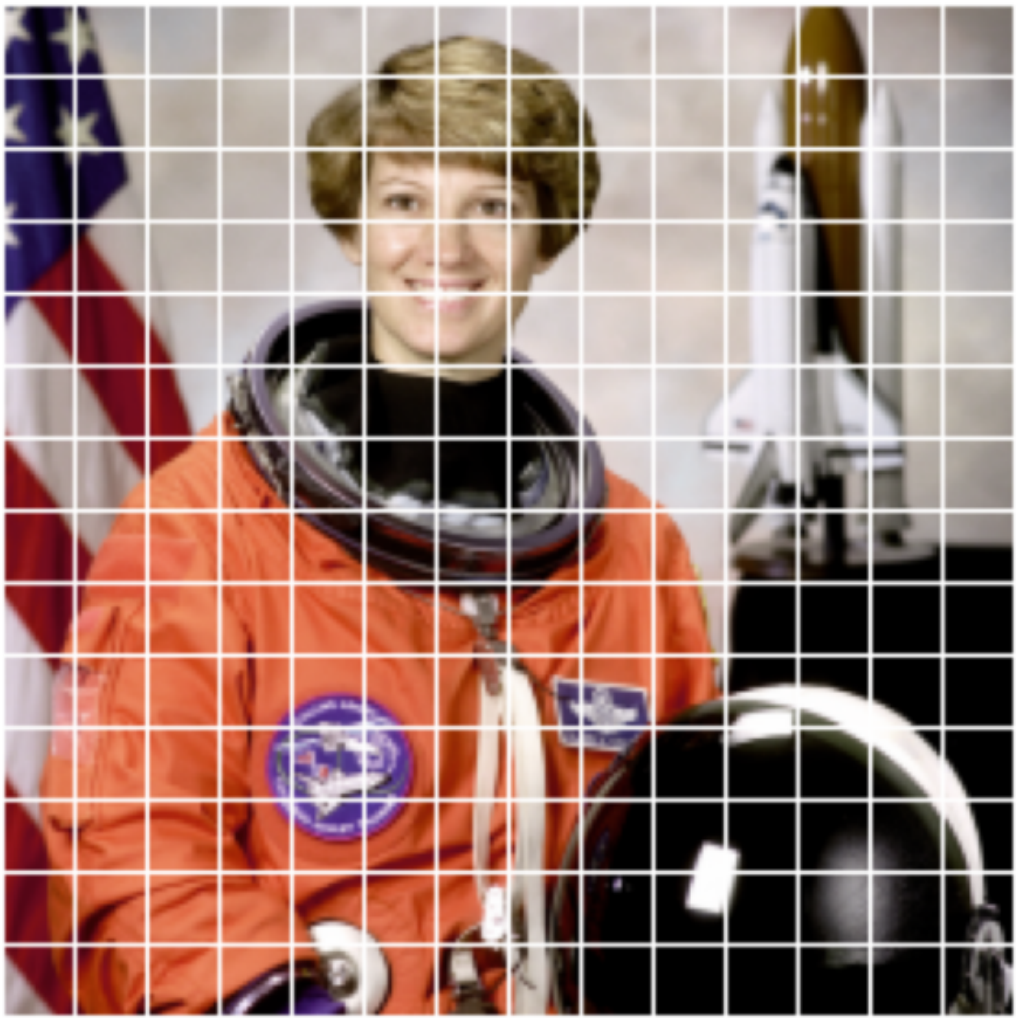
فرض کنیم اندازه تصویر ما $224 \times 224 \times 3$ باشد. یعنی طول و عرض تصویر به ترتیب ۲۲۴ و سه کانال رنگی داشته باشد.

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

شکل ۲.۹.۹: Image original

حال اگر اندازهٔ هر پچ ($P \times P$) باشد (مثلاً 16×16)، تصویر به‌صورت 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 پچ‌های ما بدون هم‌پوشانی باشند، ابعاد پچ دقیقاً باید بر ابعاد تصویر بخش‌پذیر باشد.

تعداد پچ‌ها افقی:

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

تعداد پچ‌ها 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:

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

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

ئر اکثر نسخ های مبدل های بینایی، چ ها بدون هم پوشانی هستند.(Non-overlapping) اندازه پچ های کوچک باعث می شود، تعداد پچ ها زیاد شوند. و با زیاد شدن تعداد پچ ها، هزینه attention زیاد میشود. و هم چنین اندازه پچ های بزرگ باعث میشود تعداد پچ ها کمتر شود، و هزینه های attention کمتر شود. اما در پچ های 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 همهٔ پیکسل‌های پچ را به‌صورت ردیفی (Row-major) دنبال هم بگذاریم، طول این بردار خواهد بود:

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

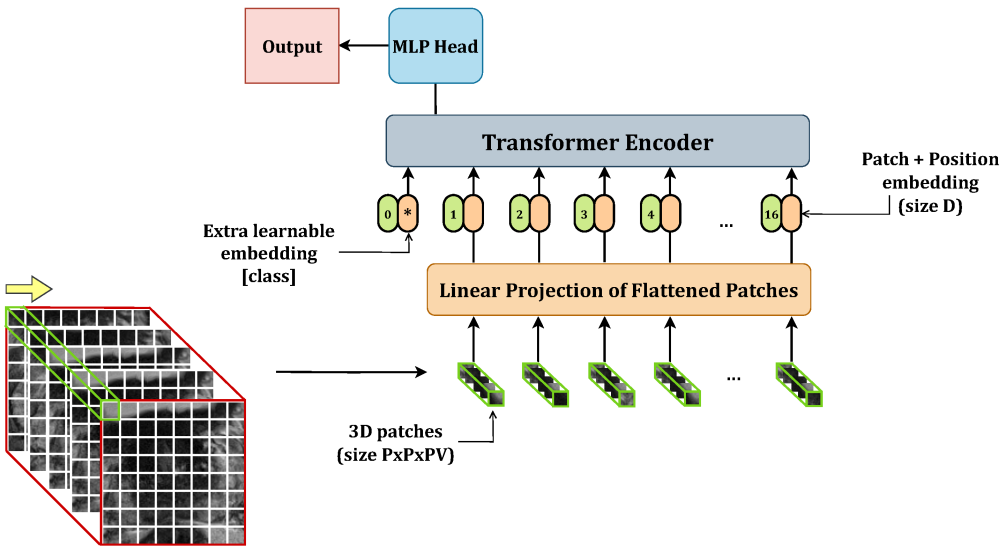
در مثال $3 \times 16 \times 16$، طول بردار می‌شود 768.

۱۰.۲ اعمال لایهٔ خطی :(Projection)

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

Transformation) را انجام می‌دهد تا همهٔ پچ‌ها یک نمایندگی (Embedding) با ابعاد یکنواخت d_{model} پیدا کنند:

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



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 بردار های embedding شده را از این پچ ها به دست آوریم.

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

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

هر پچ اکنون یک بردار d_{model}ـبعدی 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 به‌صورت ردیفی (Row by Row) شماره‌گذاری 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.

۱.۱۰.۲ Token: CLS

Token CLS یک بردار ویژه است که به ابتدای دنبالهٔ ورودی اضافه می‌شود و نقش آن این است که اطلاعات کلی و مرتبط با دنبالهٔ ورودی (چه متن، چه تصویر) را در خود خلاصه کند. token cls در به ابتدای پچ های 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 تجمیع 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 همهٔ توکن‌های دیگر (یعنی پچ‌های تصویر) ارتباط برقرار کند و اطلاعات مهم آن‌ها را در طول لایه‌های ترانسفورمر

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

Token CLS از طریق ضرب داخلی در مکانیزم Attention می‌تواند به تمام پچ‌ها نگاه کند. ضرایب توجه (α) تعیین می‌کند که Token CLS چه مقدار اطلاعات از هر پچ دریافت کند.

و چقدر در پروسه مدل دخیل باشد.

Token CLS به‌طور ضمنی یاد می‌گیرد که روی ویژگی‌هایی که برای دسته‌بندی مهم هستند (مانند الگوها، اشکال و بخش‌های کلیدی تصویر) تمرکز کند.

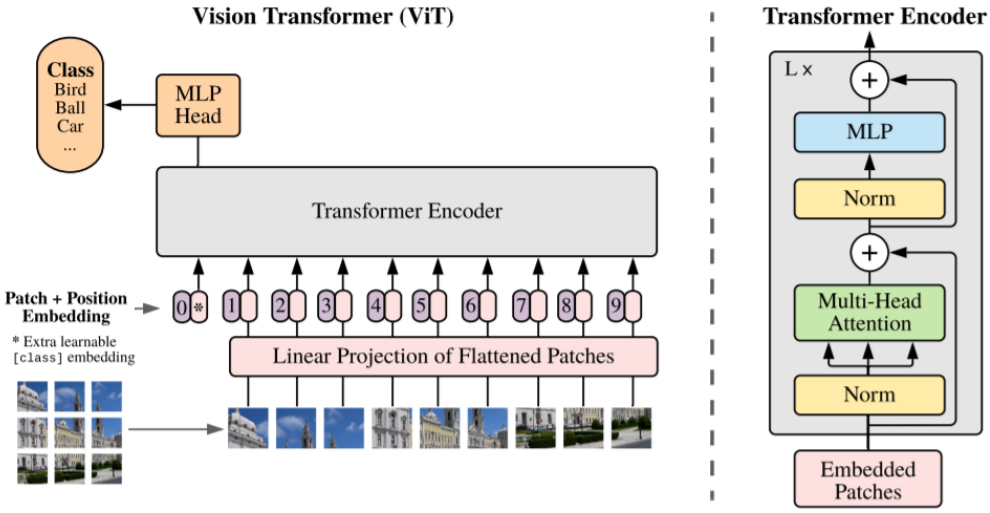
در طول لایه‌های ترانسفورمر، Token CLS نقش محوری در تنظیم بازنمایی کل تصویر ایفا می‌کند. به عبارت دیگر، این توکن به نوعی مرکز پردازش کل اطلاعات تصویر است.

Token cls به عنوان پارامتر های یادگیرنده تعریف میشود و این پارامتر ها در طول فرایند یادگیری بروز رسانی میشوند.

۲.۱۰.۲ encoder: transformer vision

انکودر در ترانسفورمر ها همانطور که مشاهده می کنید به مانند ترانسفورمر اصلی است در اخر با این تفاوت که دیگر به دیکدر نمیرویم و پس از عبور از بلاک های ترانسفورمر در ساده ترین حالت یک لایه خطی (Fully Connected) یا یک لایه Mlp (Multi-Layer Perceptron) روی بردار نهایی اعمال می شود و این لایه ها به تعداد کلاس ها خروچی میدهد. سپس خروجی هر لایه با گذر از تابع softmax به احتمال هر کلاس تبدیل می شود. و در نهایت مدل کلاس با بیشترین احتمال را به عنوان خروجی پیش بینی میکند.

در ترانسفورمر ها، هر لایه انکودر و دیکودر با پردازش عمیق تری روی توالی ورودی میتواند نمایش بهتری از ویژگی ها را به دست بیاورد. تکرار چندین باره encoder یا Decoder باعث میشود مدل بتواند ساختار های پیچیده ای را یاد بگیرد و کیفیت و دقت مدل در شناسایی توالی های طولانی و معنا های پنهان افزایش یاید . در نتیجه مدل با تعداد لایه های بیشتر اغلب عملکرد بهتری



شکل ۲.۱۰.۱۲: Transformer Vision in Token Cls

از خود نشان میدهد.

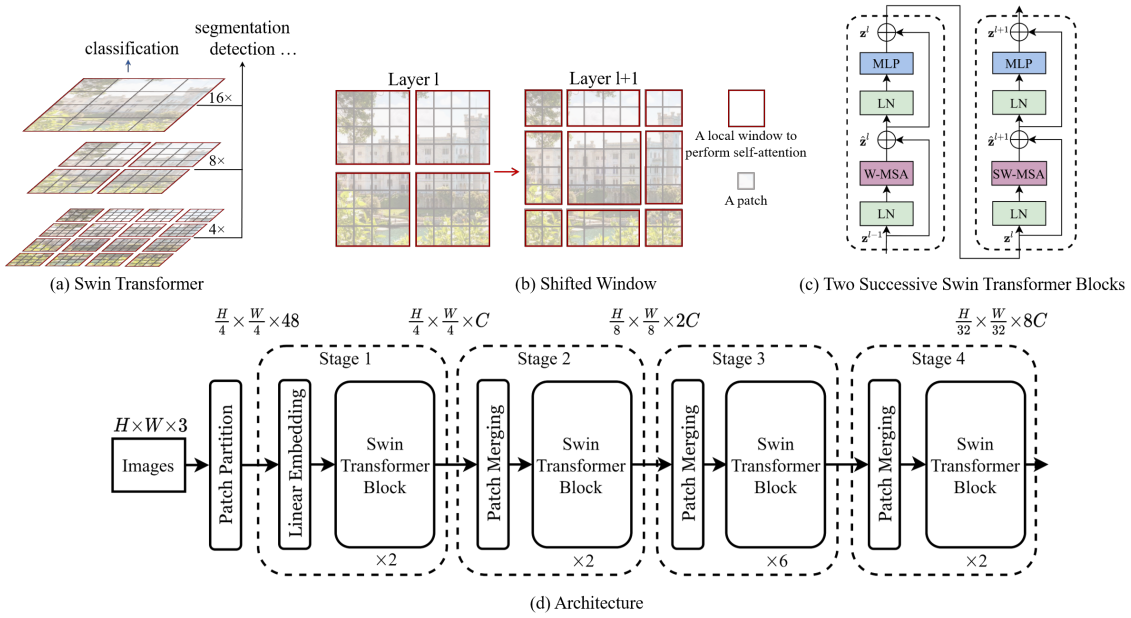
۱۱.۲ Transformer: Swin

ایده transformer swin از ترکیب چند مفهوم کلیدی در مدل های ترانسفورمر و شبکه های کانولوشنی شکل گرفت. یکی از بزرگترین مشکلات در ترانسفورمر های اولیه، نیاز به محاسبات بسیار زیاد در زمانی بود که تصویر ورودی تبعاد بسیار بزرگی داشت. در ترانسفورمر معمولی هر پچ به تمامی پچ های دیگر توجه (attention) میکرد.

و در مواقعی که تعداد پچ ها زیاد میشد، هزینه محاسباتی و حافظه به شدت افزایش پیدا میکرد.

در شبکه های کانولوشنی، معماری معمولا به صورت سلسه مراتبی پیش می رود. یعنی ابتدا ویژگی های محلی استخراج میشود، سپس با عمیق تر شدن لایه ها، این ویژگی در سطوح بالا با یک دیگر ترکیب میشوند. در transformer Swin با دانش بر این موضوع، میتوانند هم هزینه های محاسباتی را کاهش دهند و هم دقت مدل را افزایش دهند.

در Transformer Swin به جای آنکه مدل به تمام پَچ‌ها در سطح ویژگی نگاه کند، تصویر را



شکل ۲.۱۱.۱۳: Transformer Swin

به "پنجره‌های محلی" (Local Windows) تقسیم می‌کند و توجه خود را محدود به همان ناحیه می‌کند.

سپس با تکنیک جابه‌جایی (Shift) این پنجره‌ها در لایه‌های بعدی، توان مدل برای ترکیب اطلاعات از نواحی مختلف تصویر (و در نهایت دیدن کل تصویر) افزایش پیدا می‌کند. این رویکرد، ایدهٔ کلیدی بود که باعث شد مدل هم محاسبات سبک‌تر شود و هم همچنان ارتباط‌های جهانی (Global) را در طول لایه‌ها به‌دست آورد.

یکی دیگر از ایده‌های مهم در Swin، کوچک کردن تدریجی نقشهٔ ویژگی (Feature Map) در طول معماری است، شبیه به کاری که در ResNet یا سایر CNNها انجام می‌شود. این موضوع ضمن کاهش هزینهٔ محاسباتی، باعث می‌شود مدل بتواند با سطوح مختلفی از ویژگی‌ها کار کند و در نهایت خروجی نهایی باکیفیت‌تری ارائه دهد.

۱.۱۱.۲ قطعه‌بندی پچ (Patch Partition):

فرض میکنیم تصویر ورودی I دارای ابعاد $(H \times W \times 3)$ باشد. گام نخست، تقسیم تصویر به پچ‌های کوچک $(P \times P)$ است. اگر P اندازهٔ پچ (Patch size) باشد، آنگاه تعداد پچ‌ها در بعد افقی و عمودی، به‌ترتیب $\frac{H}{P}$ و $\frac{W}{P}$ خواهد بود. هر پچ را می‌توان به‌صورت یک بردار درآورد:

$$X_{\text{patch}} \in \mathbb{R}^{(P^2 \cdot 3)}.$$

سپس کل تصویر به $\frac{H}{P} \times \frac{W}{P}$ پچ تبدیل خواهد شد و در نتیجه، ماتریس X از کنار هم قرار گرفتن این پچ‌ها به صورت زیر به‌دست می‌آید:

$$X \in \mathbb{R}^{\left(\frac{H}{P} \cdot \frac{W}{P}\right) \times \left(P^2 \cdot 3\right)}.$$

۲.۱۱.۲ Embedding: Linear

در ادامه برای این‌که بتوانیم هر پچ را در یک فضای برداری با بعد C (ابعاد مدل) نمایش دهیم، یک لایهٔ خطی (Fully Connected Layer) روی هر پچ اعمال می‌شود:

$$Z = X \cdot W_{\text{embed}} + b_{\text{embed}}, \quad Z \in \mathbb{R}^{\left(\frac{H}{P} \cdot \frac{W}{P}\right) \times C}.$$

در عمل، این عملیات معادل یک تبدیل خطی ساده است:

$$W_{\text{embed}} \in \mathbb{R}^{\left(P^2 \cdot 3\right) \times C}, \quad b_{\text{embed}} \in \mathbb{R}^{C}.$$

پس از این مرحله، ما در هر موقعیت (h, w) (از شبکهٔ پچ‌ها) یک بردار $z_{h,w} \in \mathbb{R}^C$ داریم. این ماتریس Z ورودیِ اولین مرحله (Stage) از Transformer Swin است.

هر بلوک transformer swin از دو بخش اصلی تشکیل شده است.

- پنجره‌بندی تصویر (Partition Window) یا پنجره‌بندی جابه‌جاشده (Win- Shifted Partition dow)
- اعمال WMSA (Attention Self Multi-Head Window)
- لایهٔ Connection Skip و Norm Layer
- مسیر MLP:
  - یک لایهٔ MLP شامل دو لایهٔ Fully-Connected و تابع فعال‌ساز GeLU (یا تابع مشابه)
  - لایهٔ Connection Skip و Norm Layer

۳.۱۱.۲ Self-Attention: Multi-Head Window

تعریف پنجره‌های محلی:

در Swin، Transformer به‌جای آن‌که تمام پیکسل‌های یک نقشهٔ ویژگی بزرگ را یک‌جا در محاسبهٔ Attention درگیر کنیم، نقشهٔ ویژگی را به قطعه‌های کوچکی به‌اندازهٔ $(M \times M)$ تقسیم می‌کنیم. این قطعه‌های کوچک را «پنجره‌های محلی» می‌نامیم.

اگر اندازهٔ نقشهٔ ویژگی در یک لایه $(H' \times W')$ باشد، با تقسیم آن به پنجره‌های $(M \times M)$، در راستای طول تقریباً $\frac{H'}{M}$ پنجره خواهیم داشت و در راستای عرض هم $\frac{W'}{M}$ پنجره. (برای راحتی، فرض می‌کنیم H' و W' دقیقاً مضربی از M باشند تا تقسیم بدون باقی‌مانده انجام شود.)

هر کدام از این پنجره‌های $(M \times M)$ دارای M^2 پیکسل (یا موقعیت مکانی) است، و در هر پیکسل هم یک بردار ویژگی با بعد C قرار دارد.

به بیان ساده‌تر:

- نقشهٔ ویژگی مثل یک صفحهٔ بزرگ است.

- آن را مانند شطرنج به مربع‌های کوچکی $(M \times M)$ بخش می‌کنیم.
- در هر مربع (پنجره)، فقط به همان مربع نگاه می‌کنیم و محاسبات Attention را انجام می‌دهیم.
- این کار باعث می‌شود تعداد پیکسل‌هایی که درگیر محاسبهٔ Attention هستند، به‌مراتب کمتر شود و هزینهٔ محاسباتی کاهش یابد.

۴.۱۱.۲ Attention:

برای هر بلوک، ابتدا بردارهای Query، Key و Value ساخته می‌شوند. اگر $z_i \in \mathbb{R}^C$ بردار ورودی مربوط به موقعیت i باشد، آنگاه:

$$q_i = z_i W_Q, \quad k_i = z_i W_K, \quad v_i = z_i W_V,$$

که

$$W_Q, W_K, W_V \in \mathbb{R}^{C \times d}.$$

پارامتر d معمولاً $\frac{C}{h}$ در نظر گرفته می‌شود و h تعداد سربندی (Head)ها است. در Multi-Head Attention، خروجی نهایی با ترکیب h سر توجه محاسبه می‌شود.

در یک سر توجه، توجه به‌صورت زیر تعریف می‌شود:

$$\text{Attention}(Q, K, V) = \text{Softmax}\Big(\frac{QK^\top}{\sqrt{d}}\Big) V,$$

که در آن:

- Q, K, V به‌ترتیب ماتریس‌هایی هستند که از کنار هم قرار دادن q_i, k_i, v_i (برای تمام پیکسل‌های آن پنجره) ساخته می‌شوند.

- $\sqrt{d}$ عامل مقیاس‌کننده برای جلوگیری از بزرگ شدن بیش‌ازحد ضرب داخلی است.

در Transformer Swin، این محاسبات به‌صورت پنجره‌ای انجام می‌شوند؛ یعنی برای هر پنجره، تنها پیکسل‌های داخل همان پنجره در ماتریس‌های Q و K و V لحاظ می‌شوند. به این ترتیب، زمان محاسبه و مصرف حافظه به‌شدت کاهش می‌یابد (در مقایسه با ViT که همه‌چیز را با هم مقایسه می‌کند).

تعداد سربندی h معمولاً طوری انتخاب می‌شود که $C = h \times d$. خروجی هر سر پس از محاسبهٔ Attention به‌صورت زیر با هم ادغام می‌شوند:

$$\mathrm{MultiHead}(Q, K, V) = \left[\mathrm{head}_1,\ \mathrm{head}_2,\ \ldots,\ \mathrm{head}_h\right] W_O,$$

که

$$\mathrm{head}_j = \mathrm{Attention}\big(Q_j,\ K_j,\ V_j\big), \quad W_O \in \mathbb{R}^{C \times C}$$

ماتریس ترکیب نهایی است.

۵.۱۱.۲ Windows: shifted

در Transformer Swin، ایدهٔ «پنجره‌های جابه‌جاشده» (Windows Shifted) به این منظور ارائه شده است تا مدل، ارتباط پیکسل‌های واقع در پنجره‌های مجاور را هم یاد بگیرد. اگر فقط از پنجره‌های ثابت (بدون جابه‌جایی) استفاده کنیم، هر بلوک از تصویر تنها با پیکسل‌های همان پنجره در ارتباط خواهد بود و ممکن است اطلاعات نواحی مرزی با نواحی مجاور به‌خوبی تبادل نشود.

روش Swin برای رفع این محدودیت از یک تکنیک ساده اما مؤثر استفاده می‌کند:

- در یک لایه، محاسبات Attention در پنجره‌های محلی ثابت انجام می‌شود.
- در لایهٔ بعدی، پنجره‌ها به اندازه‌ای مشخص جابه‌جا می‌شوند (به‌صورت شیفت افقی و عمودی) تا نواحی مرزی نیز در محاسبات گنجانده شوند.

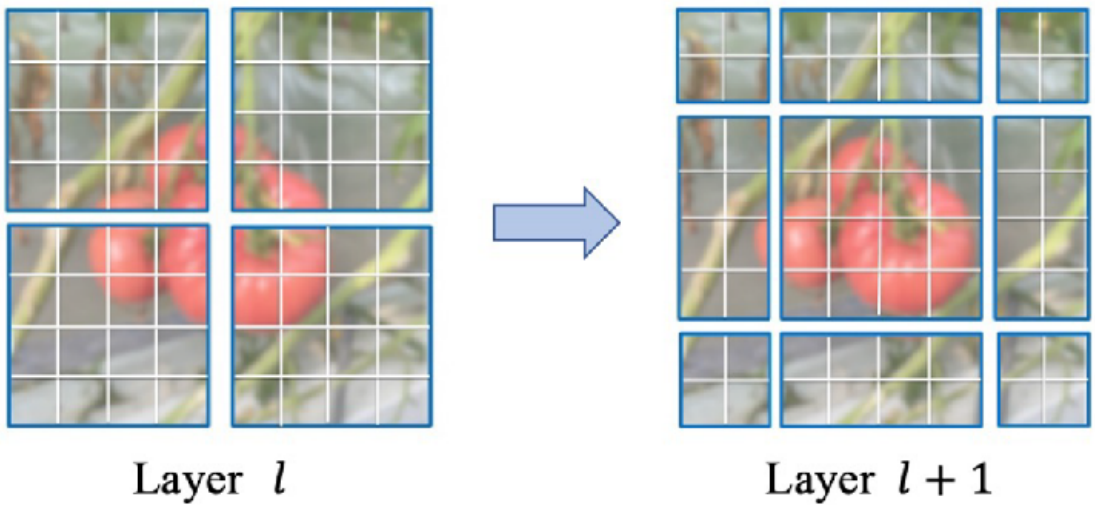
- این فرآیند باعث می‌شود که پیکسل‌ها در پنجره‌های مختلف (و در مرزهای مختلف) در محاسبات دخیل شوند و تبادل اطلاعات بهتری میان نواحی تصویر رخ دهد.

بلوک اول :(W-MSA)

در این بلوک، نقشهٔ ویژگی به پنجره‌های $(M \times M)$ تقسیم می‌شود. هیچ جابه‌جایی در این تقسیم‌بندی وجود ندارد؛ یعنی اگر نقشهٔ ویژگی را یک مستطیل بزرگ در نظر بگیریم، آن را شبیه کاشی‌کاری یا شطرنج‌بندی به بلوک‌های مربعی $(M \times M)$ برش می‌زنیم. در این حالت، پیکسل‌های هر پنجره فقط با همدیگر (درون همان پنجره) ارتباط برقرار می‌کنند

بلوک دوم :(SW-MSA)

مطابق تصویر ... بعد از اینکه بلوک اول کارش تمام شد، در بلوک دوم، قبل از تقسیم‌بندی به پنجره‌های $(M \times M)$، نقشهٔ ویژگی را جابه‌جا (Shift) می‌کنیم. در مقالهٔ اصلی، این مقدار جابه‌جایی معمولاً نیمِ اندازهٔ پنجره $\frac{M}{2}$

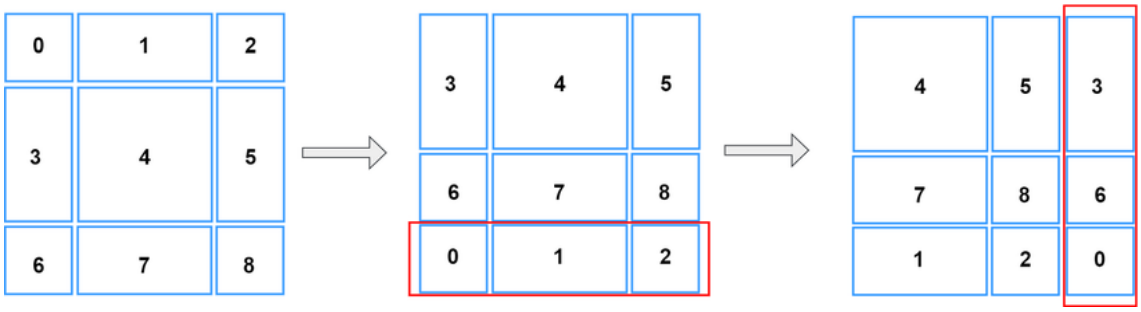


شکل ۲.۱۱.۱۴: Window Shifted vs Widow

در راستای افقی و عمودی است. به این ترتیب:

- پیکسل‌هایی که پیش از این در دو پنجرهٔ جداگانه قرار داشتند، ممکن است حالا به دلیل جابه‌جایی وارد یک پنجرهٔ مشترک شوند.
- مدل حالا می‌تواند بین این پیکسل‌های «مرزی» نیز Attention برقرار کند و اطلاعات را بهتر مبادله کند.

با این جابه‌جایی، بخشی از پیکسل‌ها در نقشهٔ ویژگی از یک طرف «خارج» می‌شوند. برای اینکه این پیکسل‌ها را از دست ندهیم، از ترفندی به نام Shift Cyclic استفاده می‌شود. در Cyclic Shift، پیکسل‌هایی که از سمت راست بیرون می‌روند دوباره از سمت چپ وارد می‌شوند و بالعکس؛ درست شبیه وقتی که یک تصویر را به‌صورت حلقه‌ای اسکرول می‌کنیم (Wrap around). مثالی از shift cycle در تصویر زیر



شکل ۲.۱۱.۱۵: Shift cycle

در بلوک اول (بدون جابه‌جایی)، پنجره‌ها ثابت‌اند و پیکسل‌های مرزی در هر پنجره ممکن است فرصت کافی برای تبادل اطلاعات با پیکسل‌های مرزیِ پنجرهٔ کناری را نداشته باشند.

در بلوک دوم (جابه‌جاشده)، مرزهای پنجره‌ها تغییر می‌کند و برخی پیکسل‌هایی که قبلاً در پنجره‌های جدا بودند، اکنون در یک پنجرهٔ مشترک‌اند؛ در نتیجه مدل می‌تواند رابطه و همبستگی بین آن‌ها را هم یاد بگیرد.

این جابه‌جایی و قرارگیری مجدد پیکسل‌ها کنار هم در نهایت کمک می‌کند تا مدل بتواند

اطلاعات کل تصویر را با هزینهٔ محاسباتی کمتر (نسبت به توجهِ سراسریِ کامل) در اختیار داشته باشد.

اگر بخواهم با مثال توضیح دهم فرض کنید در یک تابلوی شطرنجی، خانه‌های کناری همدیگر را «نمی‌بینند» چون در دو بلوک مختلف هستند. اما اگر کمی تابلوی شطرنجی را به سمت بالا ـ چپ یا پایین ـ راست جابه‌جا کنیم، حالا بخشی از آن خانه‌ها وارد یک بلوک واحد می‌شوند و اطلاعاتشان با هم ترکیب می‌شود. سپس به‌طور دوره‌ای (Cyclic)، گوشه‌های اضافی را به آن سمت دیگر تابلوی شطرنجی می‌آوریم تا هیچ چیز از دست نرود.

به این شکل، سِری اول و دوم بلوک‌های Transformer Swin (W-MSA و SW-MSA) تکمیل‌کنندهٔ یکدیگر می‌شوند:

- بلوک اول: محاسبهٔ Attention در چهارچوب پنجره‌های ثابت.
- بلوک دوم: محاسبهٔ Attention در پنجره‌های جابه‌جاشده که منجر به تعامل بیشتر بین مرزهای مختلف می‌شود.

۶.۱۱.۲ Mlp:

پس از انجام Window (یا Shifted Window) Multi-Head Self-Attention، خروجی به یک مسیر MLP می‌رود. ساختار این MLP به‌صورت زیر است:

$$X' = \mathrm{GELU}(XW_1 + b_1)W_2 + b_2,$$

که:

$$W_1 \in \mathbb{R}^{C\times(rC)}, \quad W_2 \in \mathbb{R}^{(rC)\times C}$$

هستند و r معمولاً ضریب افزایش بعد را نشان می‌دهد (مثلاً ۴).

GELU، ReLU یا سایر توابع فعال‌ساز نیز در اینجا قابل استفاده هستند.

۷.۱۱.۲ merging: patch

در مدل Swin، Transformer ساختار سلسله‌مراتبی به این معناست که ما در چند مرحله (Stage) مختلف، نقشهٔ ویژگی (Feature Map) را کوچک‌تر می‌کنیم و در عین حال، عمق (تعداد کانال‌های ویژگی) را افزایش می‌دهیم. هدف از این کار دو چیز است:

استخراج ویژگی‌های سطح بالاتر: وقتی نقشهٔ ویژگی کوچک‌تر می‌شود، هر واحد از نقشهٔ ویژگی بیانگر بخش گسترده‌تری از تصویر اصلی است؛ پس مدل به‌تدریج جزئیات محلی را با درک کلی‌تری از تصویر جایگزین می‌کند.

کاهش هزینهٔ محاسبات: در مراحل بعدی، چون ابعاد فضایی کمتر می‌شود، مدل راحت‌تر می‌تواند با ویژگی‌های جدید کار کند (چون حالا مثلاً به‌جای $(H \times W)$ پیکسل، تعداد کمتری پیکسل داریم).

این فرایند کوچک‌سازی در Transformer Swin با نام Merging Patch شناخته می‌شود که شبیه به Downsampling در شبکه‌های کانولوشنی (مثل Pooling یا Stride-Convolution) عمل می‌کند.

ابعادی به شکل $(\frac{H}{P}, \frac{W}{P})$ با تعداد کانال C دارد. این یعنی پس از برش‌دادن تصویر به پچ‌ها و یک یا چند بلوک پردازشی، اکنون یک نقشهٔ ویژگی داریم که کوچک‌تر از تصویر اصلی است، اما هنوز ممکن است خیلی بزرگ باشد.

در مرحلهٔ بعدی (Stage بعد)، می‌خواهیم این نقشه را نصف کنیم (یعنی طول و عرض را دو برابر کوچک کنیم) و در عوض عمق کانال را دو برابر کنیم (تا ظرفیت مدل در استخراج ویژگی‌های پیچیده‌تر بیشتر شود). برای انجام این کار از یک فرایند به نام Merging Patch استفاده می‌کنیم:

۱. انتخاب بلوک‌های (2×2)

ابتدا نقشهٔ ویژگی را به بلوک‌های (2×2) تقسیم می‌کنیم (در بُعد مکانی). اگر $Z_{i,j}$ ویژگیِ مکان (i, j) باشد، یک بلوک (2×2) شامل چهار پیکسل است:

$$Z_{2i,2j}, \quad Z_{2i,2j+1}, \quad Z_{2i+1,2j}, \quad Z_{2i+1,2j+1}.$$

۲. ادغام (Concat) ویژگی‌های چهار پیکسل

برای هر بلوک (2×2)، این چهار پیکسل را به هم می‌چسبانیم (Concat) در بُعد کانال. اگر هر پیکسل یک بردار از بعد C باشد، حالا بعدِ حاصل از این چهار پیکسل کنار هم می‌شود $4C$. نام این بردار ادغام‌شده Z' است.

۳. لایهٔ خطی برای تغییر بعد

وقتی چهار بردار Cـ بعدی را کنار هم گذاشته‌ایم، یک بردار $4C$ـ بعدی شکل گرفته است. حال با یک لایهٔ خطی (Connected Fully)، بعدِ $4C$ را به بعد جدیدی تبدیل می‌کنیم. معمولاً این بعد جدید برابر $2C$ در نظر گرفته می‌شود؛ یعنی می‌خواهیم دو برابر بزرگ‌تر از قبل باشد اما نه چهار برابر:

$$Z' \mapsto Z'' = Z'W_{\text{merge}} + b_{\text{merge}},$$

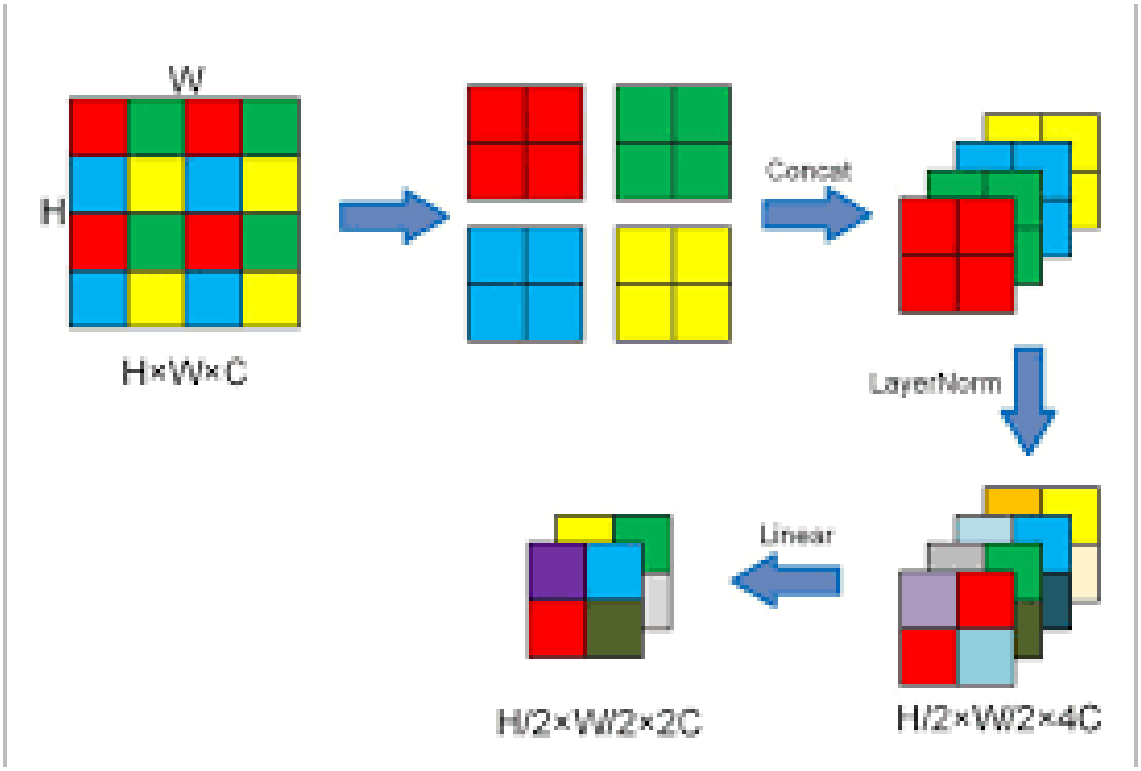
که باعث می‌شود بُعد ویژگی از $4C$ به $2C$ کاهش یابد.

۴. کاهش ابعاد مکانی

در عین حال، وقتی هر چهار پیکسل (2×2) را در یک ادغام می‌کنیم، یعنی نقشهٔ ویژگی فضای $(\frac{H}{2P} \times \frac{W}{2P})$ پیدا می‌کند (چون هر بلوک (2×2) تبدیل به یک بردار می‌شود).

به عبارت دیگر، تعداد نقاط مکانی نصف می‌شود (هم در طول و هم در عرض)، اما کانال از C به $2C$ افزایش می‌یابد.

در شبکه‌های کانولوشنی، ما مرتباً با لایه‌های Pooling یا Convolution با استراید ۲ روبه‌رو می‌شویم تا ابعاد فضایی را پایین بیاوریم و در عوض تعداد کانال‌ها را بالا ببریم. این کار کمک می‌کند



شکل ۲.۱۱.۱۶: Merging Patch

اطلاعات سطح بالاتر (مانند ساختار کلی اشیا) راحت‌تر استخراج شود. Transformer Swin هم از همین ایدهٔ سلسله‌مراتب الهام گرفته است. همچنین اگر ابعاد فضایی را پیوسته پایین نیاوریم، هزینهٔ Attention به‌شدت زیاد می‌شود (چون در هر لایه باید محاسبهٔ Attention برای همهٔ پیکسل‌ها انجام شود).

در معماری کلی Transformer Swin، در Stage ۱ ویژگی‌های اصلی گرفته می‌شود و بعد از گذر از W-MSA و SW-MSA، عملیات Merging Patch انجام می‌شود. حالا در Stage ۲ ویژگی‌های کوچک‌تری داریم، اما تعداد کانال‌ها افزایش یافته است.

مطابق آن‌چه در کانولوشن داریم، با افزایش عمق، تعداد ویژگی‌ها کمتر و تعداد کانال‌ها افزایش پیدا می‌کند.

فصل ۳

روش های پیشنهادی

در این پژوهش ما دو روش را بهبود داده ایم :

فصل ۴

آزمایشات و نتایج

کتاب‌نامه

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 .

T. M. U.

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