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

فصل ۲

مفاهیم اولیه

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

۱.۲ مقدمه

در این بخش به تاریخچه هوش مصنوعی، دستاوردهای اولیه، چالش‌ها، دلایل رکود هوش مصنوعی و پایان عصر تاریک هوش مصنوعی صحبت می‌کنیم.

۱.۱.۲ آغاز هوش مصنوعی و هدف اصلی

هوش مصنوعی به عنوان شاخه‌ای از علوم کامپیوتر، در دهه ۱۹۵۰ با هدف ساخت سیستم‌ها و ماشین‌هایی که توانایی تقلید از هوش انسانی را دارند، آغاز شد. نخستین بار، مکارتی در سال ۱۹۵۶ این اصطلاح را به کار گرفت [۳۲] و هوش مصنوعی به عنوان علمی که در آن به مطالعه الگوریتم‌هایی برای تقلید رفتار انسانی می‌پردازد، شناخته شد. اهداف اولیه هوش مصنوعی شامل توانایی درک زبان، یادگیری، حل مسئله و تولید موجودات هوشمند بود. در این دوران پروژه‌های تحقیقاتی زیادی به

امید دستیابی به هوش مصنوعی عمومی (AGI, Artificial General Intelligence) شروع به کار کردند [۸، ۴۰].

۲.۱.۲ دورهٔ طلایی و پیشرفت‌های اولیه

در دههٔ ۵۰ و ۶۰ میلادی، هوش مصنوعی به عنوان یکی از پرچمداران پژوهش‌های نوین شناخته می‌شد. الگوریتم‌های اولیه با تکیه بر روش‌های منطقی و ریاضیاتی برای حل مسئله و بازی‌های ساده توسعه یافتند؛ مانند انواع الگوریتم‌های جستجوی درختی که در این دوره به وجود آمدند و زمینه‌ساز اولین دستاوردهای هوش مصنوعی در بازی‌های تخته‌ای همچون شطرنج شدند [۳۹]. 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 به‌زودی ممکن خواهد بود [۱۴].

۳.۱.۲ انتظارات بیش از حد و ظهور عصر تاریک

با وجود پیشرفت‌های هوش مصنوعی، محدودیت‌های تکنولوژی (مثل عدم وجود GPUهای پرقدرت در آن زمان) و همچنین کمبود داده‌های کافی برای آموزش مدل‌های پیچیده‌تر، باعث شد که بسیاری از پروژه‌های تحقیقاتی نتوانند به نتایج پیش‌بینی‌شده دست یابند. در نتیجه، هوش مصنوعی در دههٔ ۷۰ به مرحله‌ای از رکود وارد شد که به آن عصر تاریک هوش مصنوعی یا AI Winter می‌گویند [۲۸، ۸]. در این دوران، بسیاری از پروژه‌ها تعطیل و سرمایه‌گذاری‌ها قطع شدند و دولت‌ها و سازمان‌های سرمایه‌گذار به دلیل عدم دستیابی به نتایج مطلوب از ادامهٔ سرمایه‌گذاری منصرف شدند.

۴.۱.۲ عوامل اصلی عصر تاریک هوش مصنوعی

- محدودیت‌های سخت‌افزاری: در آن زمان، سیستم‌های اولیهٔ هوش مصنوعی به محاسبات سنگینی نیاز داشتند که با توان پردازشی محدود آن دوره همخوانی نداشت [۴۰].

- 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 به‌درستی آموزش ببینند و عملکرد مطلوبی داشته باشند [۸].
- 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 [۴۴].

۵.۱.۲ پایان عصر تاریک و بازگشت هوش مصنوعی

پس از چندین سال رکود و عدم سرمایه‌گذاری در حوزهٔ هوش مصنوعی، سرانجام در دههٔ ۱۹۸۰ و ۱۹۹۰ عصر تاریک هوش مصنوعی با تحولات تکنولوژی و از همه مهم‌تر ظهور سیستم‌های خبره (Expert Systems) به پایان رسید [۱۴]. سیستم‌های خبره به عنوان یکی از اولین تلاش‌های موفق برای کاربردهای صنعتی در هوش مصنوعی به‌وجود آمدند. بر خلاف 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 می‌شد. 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 [۳۳].

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 به صرفه‌تر شدند و امکان پردازش داده‌های بیشتر و اجرای الگوریتم‌های پیچیده‌تر فراهم شد. این افزایش توان محاسباتی، نیاز به پردازش داده‌های بزرگ و پیچیده را برآورده کرد و در نتیجه دسترسی به داده‌ها و انجام محاسبات سنگین برای توسعهٔ الگوریتم‌های جدید تسهیل شد. از سوی

دیگر، پیشرفت‌های انجام‌شده در ذخیره‌سازی داده و رشد اینترنت باعث دسترسی گسترده‌تر به داده‌ها و منابع اطلاعاتی گردید [۴۰].

به این ترتیب، مجموعه‌ای از عوامل، شامل ظهور سیستم‌های خبره، افزایش قدرت پردازش و دسترسی به داده‌های بیشتر، منجر به بازگشت هوش مصنوعی شد. این دوره نه تنها پایان عصر تاریک هوش مصنوعی بود، بلکه راه را برای الگوریتم‌های یادگیری ماشین و توسعهٔ شبکه‌های عصبی هموار کرد [۴۴].

۲.۲ انواع مدل یادگیری ماشین و شبکه‌های عصبی

یادگیری ماشین و شبکه‌های عصبی در سال‌های اخیر مورد توجه بسیاری قرار گرفته‌اند و در حوزه‌های متنوعی از جمله پردازش تصویر، پردازش زبان طبیعی و داده‌کاوی استفاده می‌شوند [۴، ۳۵، ۳۷].

۱.۲.۲ یادگیری ماشین: مروری کلی

یادگیری ماشین (Machine Learning) شاخه‌ای از هوش مصنوعی است که به مدل‌های محاسباتی این امکان را می‌دهد الگوها را از داده‌ها به شکل خودکار یاد بگیرند و بتوانند تصمیم‌گیری کنند [۱۶، ۳۵]. در واقع، هدف یادگیری ماشین این است که مدل‌ها بتوانند از داده‌ها الگوها و روابط پنهان را استخراج کنند و به نتایج و تصمیم‌های قابل اعتماد دست یابند.

۲.۲.۲ تقسیم‌بندی‌های اصلی در یادگیری ماشین

به طور کلی، یادگیری ماشین به سه دستهٔ اصلی تقسیم می‌شود:

- یادگیری با نظارت[۱]
- یادگیری بدون نظارت[۲]

[۱] Learning Supervised
[۲] Learning Unsupervised

- یادگیری تقویتی[۳]

این طبقه‌بندی در بسیاری از کتاب‌ها و مراجع مهم یادگیری ماشین مطرح شده است [۴، ۳۷].

۳.۲.۲ یادگیری نظارت‌شده (Supervised Learning)

یادگیری نظارت‌شده یکی از رایج‌ترین روش‌ها در یادگیری ماشین شناخته می‌شود که در آن از مجموعه داده‌های برچسب‌گذاری‌شده برای آموزش مدل استفاده می‌کنیم [۲۳]. هدف این الگوریتم تشخیص الگوها در میان داده‌های ورودی است تا بتواند روی داده‌های جدید پیش‌بینی یا طبقه‌بندی انجام دهد. این نوع شامل دو دسته الگوریتمِ Regression و Classification می‌شود.

طبقه‌بندی (Classification)

طبقه‌بندی یکی از مهم‌ترین و اصلی‌ترین وظایف در یادگیری نظارت‌شده است که هدف آن تخصیص هر داده به یک لیبل مشخص است [۴]. در این روش، مدل با داده‌های برچسب‌دار (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 کاربردهای طبقه‌بندی می‌توان به تشخیص اسپم (Spam Detection)، تشخیص بیماری (مثلاً آیا یک فرد مبتلا به بیماری هست یا نه) و تشخیص چهره اشاره کرد [۳۷].

رگرسیون (Regression)

رگرسیون یکی از مهم‌ترین وظایف یادگیری ماشین است و هدف آن پیش‌بینی مقادیر پیوسته است [۳۶]. بر خلاف طبقه‌بندی که خروجی آن یک دسته‌بندی مجزا است، در رگرسیون خروجی یک مقدار پیوسته خواهد بود و مدل می‌آموزد روابط بین متغیرهای مستقل و متغیر هدف را شناسایی کند. از کاربردهای رگرسیون می‌توان به پیش‌بینی قیمت مسکن یا پیش‌بینی آب‌وهوا اشاره کرد.

---

[۳] Learning Reinforcement

۴.۲.۲ یادگیری تقویتی (Reinforcement Learning)

یادگیری تقویتی، نوعی یادگیری بر پایهٔ پاداش و تنبیه است که در آن مدل با محیط تعامل می‌کند و بر اساس پاداش یا تنبیه یاد می‌گیرد [۴۶]. برخلاف یادگیری نظارت‌شده و بدون نظارت، یادگیری تقویتی به مدل این امکان را می‌دهد تا 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 نتیجهٔ بهینه می‌شود. از کاربردهای یادگیری تقویتی می‌توان به بازی‌ها (Games)، کنترل رباتیک (Robotic Control) و سیستم‌های توصیه‌گر (Recommender Systems) اشاره کرد.

۵.۲.۲ معرفی چند مدل از الگوریتم یادگیری کلاسیک

نزدیک‌ترین همسایه (k-Nearest Neighbors, kNN)

الگوریتم kNN یکی از روش‌های ساده و درعین‌حال کارآمد در یادگیری نظارت‌شده است که هم در دسته‌بندی و هم در رگرسیون کاربرد دارد [۷، ۱۲، ۳۵]. این الگوریتم برای پیش‌بینی دسته‌بندی یک نمونهٔ جدید، به k نزدیک‌ترین داده‌ها در فضای ویژگی نگاه می‌کند و بر اساس اکثریت نزدیکی همسایه‌ها، آن را به یک دسته اختصاص 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 [۷].
- عملکرد خوب در داده‌های با تعداد ویژگی کم: در مسائلی که تعداد ویژگی‌ها کم است، این الگوریتم اغلب به خوبی عمل می‌کند [۲۳].

معایب:

- حساسیت به داده‌های پرت: نقاط پرت می‌توانند به‌طور قابل توجهی بر نتایج تأثیر بگذارند [۱۲].
- کندی در داده‌های بزرگ: این الگوریتم نیاز به محاسبه فاصله برای هر نقطهٔ جدید دارد و در داده‌های بزرگ بار محاسباتی بالایی خواهد داشت [۳۵].
- عدم کارایی در داده‌های با ابعاد بالا: در داده‌هایی با تعداد ویژگی‌های زیاد، کارایی الگوریتم کاهش می‌یابد [۳۷].

۶.۲.۲ ماشین بردار پشتیبان (Support Vector Machine, SVM)

الگوریتم SVM با یافتن یک ابرصفحهٔ بهینه، داده‌ها را به کلاس‌های مختلف تقسیم می‌کند [۶، ۴۷]. این الگوریتم یک ابرصفحه به دست می‌آورد که هدف آن حداکثر کردن فاصله میان داده‌های دو کلاس است و به این ترتیب می‌تواند طبقه‌بندی دقیقی داشته باشد.

مزایا:

- توانایی مقابله با داده‌های پیچیده و ابعاد بالا: SVM می‌تواند به خوبی با داده‌های چندبعدی و پیچیده کار کند [۴۷].
- مقاومت در برابر بیش‌برازش (**Overfitting**): با استفاده از هسته‌ها (kernels)، داده‌های غیرخطی نیز به فضای بالاتر برده می‌شوند و جداسازی بهتری انجام می‌شود [۶].

معایب:

- پیچیدگی محاسباتی: آموزش SVM به دلیل نیاز به حل مسائل بهینه‌سازی، در حجم‌های بالای داده محاسباتی زمان‌بر است [۳۷].

- کارایی پایین در داده‌های پرت: در صورتی که داده‌ها شامل نقاط پرت زیادی باشند، دقت مدل کاهش می‌یابد [۴].

۷.۲.۲ بیز ساده (Naive Bayes)

بیز ساده مبتنی بر قضیه بیز است و فرض می‌کند ویژگی‌ها به‌صورت شرطی مستقل از هم هستند [۱۰، ۳۵]. این مدل برای اولین بار در حوزهٔ پردازش متن به کار رفت و هنوز هم در بسیاری از کاربردها مانند طبقه‌بندی ایمیل و تحلیل احساسات مورد استفاده قرار می‌گیرد [۳۱]. در Naive Bayes بر اساس احتمالات محاسبه می‌شود که یک نمونهٔ جدید به کدام دسته تعلق دارد. این الگوریتم بر اساس قضیهٔ بیز، احتمال تعلق یک نمونه به هر دسته را به ازای هر ویژگی محاسبه کرده و در نهایت بالاترین احتمال را به‌عنوان جواب نهایی در نظر می‌گیرد [۴].

مزایا:

- سرعت بالا: به دلیل محاسبات ساده و فرض استقلال ویژگی‌ها، Naive Bayes بسیار سریع و کم‌حجم است [۳۱].
- کارایی در داده‌های کوچک: حتی با داده‌های کم، این الگوریتم عملکرد نسبتاً خوبی دارد [۳۷].

معایب:

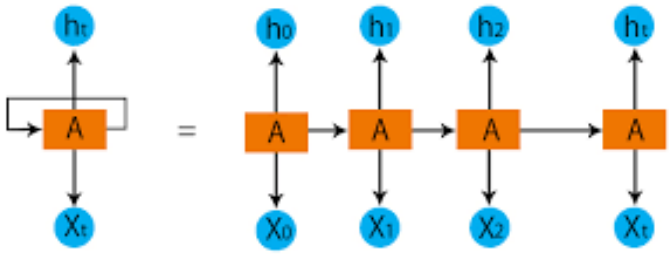
- فرض استقلال ویژگی‌ها: فرض استقلال ویژگی‌ها ممکن است در بسیاری از مسائل واقعی صادق نباشد و این می‌تواند دقت مدل را کاهش دهد [۱۰].
- حساسیت به داده‌های نادرست: در صورت وجود داده‌های نادرست یا پرت، مدل ممکن است دقت کمتری داشته باشد [۴].

۸.۲.۲ شبکه‌های عصبی بازگشتی (RNN) و شبکه‌های حافظه بلندمدت کوتاه‌مدت (LSTM)

شبکه‌های عصبی بازگشتی (RNN) و مدل‌هایی با حافظهٔ بلندمدت ـ کوتاه‌مدت (LSTM) با هدف پردازش داده‌های ترتیبی و وابسته به زمان توسعه یافتند [۴۳، ۲۰]. این مدل‌ها به‌ویژه در تحلیل زبان طبیعی، پردازش صوت و پیش‌بینی سری‌های زمانی بسیار موفق عمل کرده‌اند؛ زیرا قادر به حفظ اطلاعات گذشته هستند و از این اطلاعات برای پیش‌بینی در لحظهٔ حال و آینده استفاده می‌کنند [۱۵].

۹.۲.۲ RNN

مدل‌های اولیهٔ شبکه‌های عصبی، مانند شبکه‌های چندلایه (MLP)، قادر به پردازش داده‌های مستقل و ثابت بودند و نمی‌توانستند وابستگی‌های زمانی را یاد بگیرند [۴]. در بسیاری از مباحث دنیای واقعی مانند تحلیل متن و صدا، داده‌ها به توالی خاصی وابسته هستند. به همین دلیل، شبکه‌های RNN معرفی شدند تا بتوانند از اطلاعات پیشین در پردازش داده‌های بعدی استفاده کنند [۴۳].



شکل ۲.۲.۱: RNN

ساختار و عملکرد RNN

شبکه‌های RNN دارای حلقهٔ بازگشتی هستند که به مدل این امکان را می‌دهد اطلاعات را در توالی نگه دارد و در هر گام زمانی، ورودی فعلی x_t و وضعیت قبلی h_{t-1} را به‌عنوان ورودی دریافت کند

[۱۶]:

$$h_t = \sigma\big(W \cdot x_t + U \cdot h_{t-1} + b\big) \qquad (۲.۲.۱)$$

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 به 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 پردازش‌های بعدی از آن‌ها بهره ببرد.

۱۰.۲.۲ مزایا و معایب RNN

در این قسمت به مزایا و معایب شبکه‌های RNN می‌پردازیم.

مزایا:

- حفظ وابستگی زمانی: RNN قادر به پردازش توالی‌های طولانی است و می‌تواند اطلاعات را در طول توالی به خاطر بسپارد [۱۳].
- کاربردهای گسترده در داده‌های ترتیبی: این مدل در تحلیل زبان طبیعی، پیش‌بینی سری‌های زمانی و پردازش صوت بسیار موفق عمل می‌کند [۱۵].

معایب:

- مشکل ناپدید شدن و انفجار گرادیان (**Vanishing and Exploding Gradient**): در فرایند آموزش با روش پس‌انتشار، اگر توالی داده‌ها طولانی باشد، گرادیان‌ها ممکن است بسیار کوچک یا بزرگ شوند که منجر به ناپایداری در آموزش و کاهش دقت می‌شود [۱۹].
- محدودیت در پردازش توالی‌های بسیار بلند: RNN در حفظ اطلاعات طولانی‌مدت با مشکل مواجه است و برای پردازش وابستگی‌های طولانی، عملکرد ضعیفی دارد [۲۰، ۱۶].

۱۱.۲.۲ شبکه‌های حافظه بلندمدت ـ کوتاه‌مدت (LSTM)

علل پیدایش LSTM

شبکه‌های LSTM به عنوان یک راه‌حل برای یکی از بزرگ‌ترین مشکلات شبکه‌های عصبی بازگشتی (RNN) معرفی شدند [۲۰]. یکی از برجسته‌ترین مشکلات موجود در RNNها، معضل ناپدید شدن گرادیان (Vanishing Gradient) بود که مانع یادگیری وابستگی‌های بلندمدت می‌شد [۱۹، ۱۶]. برای درک عمیق‌تر این مسأله، ابتدا به توضیح مشکل ناپدید شدن گرادیان و سپس راهکار LSTM می‌پردازیم.

Gradient Vanishing

شبکه‌های RNN برای پردازش داده‌های ترتیبی از حلقه‌های بازگشتی بهره می‌برند. در فرایند آموزش RNN، از الگوریتم پس‌انتشار خطا از طریق زمان (Backpropagation Through Time, BPTT) استفاده می‌شود که گرادیان‌ها را جهت به‌روزرسانی وزن‌ها محاسبه می‌کند [۴۳]. با این حال، RNNها در یادگیری وابستگی‌های بلندمدت معمولاً ناکام می‌مانند. علت اصلی این امر شامل موارد زیر است:

- ضریب‌های بازگشتی کوچک‌تر از ۱: در فرایند محاسبهٔ گرادیان‌ها، اگر مقدار مشتقات یا

ضرایب در هر مرحله کوچک‌تر از ۱ باشد، ضرب مکرر این ضرایب در طول توالی منجر به کوچک‌شدن گرادیان‌ها به سمت صفر می‌شود؛ پدیده‌ای که به ناپدید شدن گرادیان معروف است [۱۹].

فرمول کلی گرادیان در زمان t به‌صورت 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 این فرمول، $\frac{\partial h_k}{\partial h_{k-1}}$ ممکن است مقداری کوچک‌تر از ۱ باشد، و ضرب مکرر آن در طول توالی باعث کاهش شدید مقدار گرادیان می‌گردد.

- تأثیر مستقیم بر وزن‌ها: زمانی که گرادیان‌ها به صفر نزدیک می‌شوند، وزن‌های مدل عملاً به‌روزرسانی نمی‌شوند و این امر مانع از یادگیری وابستگی‌های طولانی‌مدت در داده‌ها می‌شود [۱۶].

۱۲.۲.۲ ظهور LSTM

در سال ۱۹۹۷، Sepp Hochreiter و Jürgen Schmidhuber شبکه‌های حافظهٔ بلندمدت ـ کوتاه‌مدت (LSTM) را معرفی کردند [۲۰]. انگیزهٔ اصلی توسعهٔ LSTM حل مشکل ناپدید شدن گرادیان در شبکه‌های RNN بود. این مشکل در مسائل یادگیری داده‌های ترتیبی طولانی مانع می‌شد RNN وابستگی‌های بلندمدت را به‌درستی فراگیرد.

راه‌حل **LSTM** برای پایداری جریان گرادیان‌ها

LSTM با معرفی 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 [۱۵]. این اجزا به LSTM امکان می‌دهند:

۱. اطلاعات غیرضروری را فراموش کند،

۲. اطلاعات مهم جدید را اضافه کند،

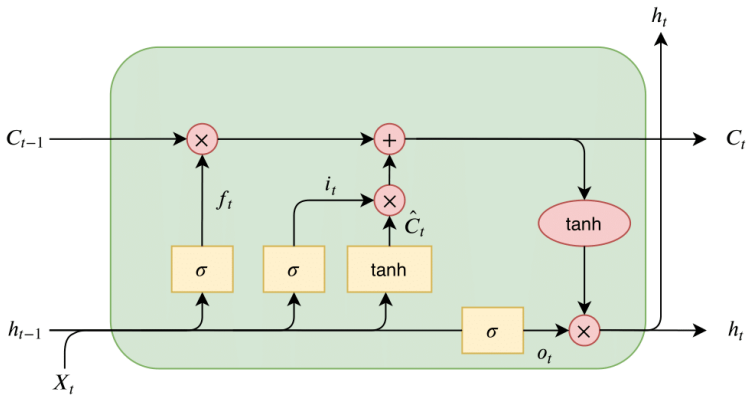
۳. اطلاعات مهم قبلی را حفظ کند.

۳.۲ اختار LSTM: نوآوری در مقایسه با RNN

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

۱.۳.۲ وضعیت سلولی (Cell State)

مسیر اصلی ذخیرهٔ اطلاعات در LSTM است که می‌تواند اطلاعات مهم را در طول توالی حفظ کند. برخلاف RNN که عمدتاً بر خروجی‌های بازگشتی h_t متکی است، LSTM یک مسیر جداگانه برای عبور اطلاعات از وضعیت سلولی دارد که به حفظ گرادیان‌ها کمک شایانی می‌کند [۲۰].



شکل ۲.۳.۲: LSTM

۲.۳.۲ دروازه‌ها (Gates)

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

- دروازهٔ فراموشی (**Forget Gate**): تعیین می‌کند چه اطلاعاتی از وضعیت سلولی باید حذف شود [۱۵]:

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

f_t : میزان فراموشی برای هر عنصر از وضعیت سلولی ,

σ : تابع سیگموید (خروجی بین ۰ و ۱) .

در صورت $f_t = 0$، اطلاعات حذف می‌شود و در صورت $f_t = 1$، حفظ می‌شود.

- دروازهٔ ورودی (**Input Gate**): تعیین می‌کند چه اطلاعات جدیدی باید به وضعیت سلولی اضافه شود:

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

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

که در آن i_t میزان اطلاعات جدید و $\tilde{C}_t$ مقدار جدید قابل اضافه‌شدن به وضعیت سلولی را نشان می‌دهد.

- دروازهٔ خروجی (**Output Gate**): تعیین می‌کند چه اطلاعاتی از وضعیت سلولی به خروجی منتقل شود:

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

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

۳.۳.۲ 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 اطلاعات قدیمی مهم حفظ و داده‌های 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 جداگانهٔ وضعیت سلولی، از کاهش نمایی گرادیان جلوگیری می‌کند [۱۹].
- استفاده از توابع سیگموید و تانژانت هیپربولیک: توابع سیگموید در دروازه‌ها و تانژانت هیپربولیک در وضعیت سلولی مقادیر را محدود می‌کنند و مانع از انفجار گرادیان می‌شوند [۱۵، ۱۶].
- مدیریت اطلاعات توسط دروازه‌ها: دروازه‌های فراموشی و ورودی به مدل اجازه می‌دهند تنها اطلاعات مهم حفظ شود و داده‌های غیرضروری حذف شوند؛ این موضوع از پیچیدگی محاسباتی غیرضروری جلوگیری می‌کند [۲۰].

ویژگی	RNN	LSTM
مشکل ناپدید شدن گرادیان	وجود دارد	برطرف شده
توانایی حفظ وابستگی‌های طولانی‌مدت	محدود به وابستگی کوتاه‌مدت	بسیار خوب
ساختار دروازه‌ها	ندارد	دارای دروازه‌های فراموشی، ورودی و خروجی
پایداری گرادیان	ضعیف	پایدار

جدول ۲.۳.۱: مقایسه ویژگی‌های RNN و LSTM

۴.۳.۲ مشکلات کلی **RNN** و **LSTM** و ظهور ترنسفورمرها

شبکه‌های بازگشتی RNN و LSTM توانستند بسیاری از مشکلات و محدودیت‌های مدل‌های اولیه را حل کنند؛ اما همچنان با چالش‌ها و محدودیت‌هایی مواجه بودند که در مسائل پیچیده‌تر، مانند ترجمهٔ زبان یا تحلیل داده‌های بلندمدت و حجیم، مشکلات جدی ایجاد می‌کردند [۲۰، ۱۶]. این مشکلات در نهایت به پیدایش ترنسفورمرها (Transformers) منجر شد [۴۸]. در ادامه، مهم‌ترین محدودیت‌های RNN و LSTM مورد بررسی قرار می‌گیرند.

مشکل وابستگی ترتیبی در **RNN**ها و **LSTM**ها

RNNها و LSTMها داده‌ها را به‌صورت ترتیبی پردازش می‌کنند؛ به این معنی که برای پردازش داده‌های گام زمانی t، باید تمامی داده‌های قبلی $(t-1)$ پردازش شده باشند [۴۳، ۲۰]. این ویژگی مشکلات زیر را ایجاد می‌کند:

- غیرقابل موازی‌سازی: به دلیل وابستگی ترتیبی، پردازش داده‌ها به‌صورت موازی ممکن نیست و همین امر باعث افزایش زمان محاسباتی می‌شود. در داده‌های بلند (مانند متن‌های طولانی یا سری‌های زمانی بزرگ)، این مشکل نمود بیشتری دارد.
- کندی آموزش و استنتاج: پردازش خطی داده‌ها موجب می‌شود زمان آموزش و پیش‌بینی مدل‌ها به‌شدت افزایش یابد، به‌ویژه زمانی که با حجم زیادی از داده مواجه هستیم.

محدودیت در یادگیری وابستگی‌های بسیار طولانی

با وجود پیشرفت LSTM در یادگیری وابستگی‌های بلندمدت نسبت به RNNهای معمولی، این مدل‌ها همچنان در یادگیری وابستگی‌های بسیار بلند، مانند ارتباط بین کلمات در جملات دور از هم یا درک ساختار کلی یک متن، محدودیت دارند [۱۹]:

- مشکل در داده‌های بسیار طولانی: حتی در LSTM نیز ظرفیت حفظ اطلاعات محدود است و با افزایش طول توالی، دقت مدل افت می‌کند.
- تأثیر تدریجی داده‌های اولیه: داده‌های ابتدایی توالی ممکن است با گذشت زمان اهمیت خود را از دست بدهند، چراکه گرادیان‌ها به‌تدریج ضعیف‌تر می‌شوند.

پیچیدگی محاسباتی و حافظه

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

[۱۶]:

- نیاز به حافظه بیشتر: برای ذخیرهٔ وضعیت سلولی و گرادیان‌ها، LSTMها به حافظهٔ بیشتری نسبت به مدل‌های ساده‌تر احتیاج دارند.
- هزینهٔ محاسباتی بالا: در داده‌های حجیم، انجام محاسبات سنگین می‌تواند اجرای مدل را بسیار کند سازد.

مشکل پردازش وابستگی‌های غیرمتوالی

RNNها و LSTMها به‌طور طبیعی برای یادگیری وابستگی‌های محلی و متوالی مناسب هستند. اما در مسائلی مانند ترجمهٔ زبان یا تحلیل متون، روابط غیرمحلی و غیرمتوالی نیز اهمیت دارند [۲]. به‌عنوان مثال، در جمله‌ای طولانی ممکن است کلمه‌ای در ابتدای جمله با کلمه‌ای در انتهای جمله ارتباط معنایی داشته باشد. RNNها و LSTMها برای یادگیری این‌گونه وابستگی‌ها محدودیت دارند.

گرادیان‌های ناپایدار و مشکلات بهینه‌سازی

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

- مسائل گرادیان‌های ناپایدار: در توالی‌های بسیار بلند، گرادیان‌ها ممکن است همچنان دچار کاهش یا حتی در مواردی انفجار شوند.
- مشکلات بهینه‌سازی: در مسائلی با ساختار پیچیده، یافتن مینیمم مناسب تابع هزینه برای RNNها و LSTMها دشوار است.

نیاز به مدلی با ظرفیت بیشتر و سرعت بالاتر

- مدل‌های بزرگ‌تر: برای مسائل پیچیده‌تر، به مدل‌هایی با تعداد پارامتر بالاتر نیاز است؛ اما RNNها و LSTMها به‌دلیل محدودیت در حافظه و پردازش، پاسخ‌گوی این نیاز نیستند.

- کارایی در داده‌های چندوجهی (**Multimodal**): برای داده‌هایی که ترکیبی از اطلاعات متنی، صوتی و تصویری هستند، RNNها و LSTMها توانایی لازم جهت پردازش موازی این اطلاعات را ندارند.

در مجموع، وابستگی ترتیبی در RNN و LSTM مانعی اساسی برای استفاده از این مدل‌ها در مسائل پیچیده و بزرگ بود که درنهایت به ظهور ترنسفورمرها منتهی شد [۴۸]. ترنسفورمرها با طراحی مبتنی بر موازی‌سازی و مکانیزم توجه (Attention Mechanism)، این محدودیت را برطرف کرده و راه‌حلی کارآمدتر برای پردازش داده‌های ترتیبی ارائه دادند.

فصل ۳

پیشینه پژوهش

۱.۳ مقدمه

ظهور مدل‌های Transformer و انقلاب در یادگیری عمیق، یکی از تحولات اساسی در حوزهٔ پردازش زبان طبیعی (NLP) و یادگیری ماشین به شمار می‌رود [۴۸، ۲]. این مدل‌ها باعث تغییرات عمده‌ای در نحوهٔ 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[۹، ۴۲].

۲.۳ مشکلات ترجمه ماشینی و ترانسفورمرها

در ابتدا، ترجمه ماشینی (MT) یک چالش اساسی در زمینهٔ پردازش زبان طبیعی بود. مدل‌های اولیه‌ای مانند مدل‌های مبتنی بر قواعد (Rule-based Models) برای ترجمه استفاده می‌شدند که در آن‌ها، ترجمه‌ها به‌صورت دستی با استفاده از قواعد زبانی مشخص تنظیم می‌شدند [۳۸، ۲۱]. این روش‌ها هرچند دقیق بودند، اما محدودیت‌های زیادی داشتند و نمی‌توانستند ویژگی‌های پیچیده‌تر زبان را مدل‌سازی کنند.

سپس مدل‌های آماری (Statistical Models) معرفی شدند [۵، ۲۴]. این مدل‌ها از داده‌های ترجمه‌شده برای آموزش مدل‌های آماری استفاده می‌کردند که احتمال ترجمهٔ صحیح را براساس شواهد آماری محاسبه می‌کردند. مدل‌هایی مانند مدل‌های ترجمهٔ آماری مبتنی بر جمله (Phrase-based Statistical Models) [۲۵] از این نوع بودند که قادر به ترجمهٔ جملات بهتر از مدل‌های مبتنی بر قواعد بودند، اما هنوز هم در ترجمه‌های پیچیده با مشکلاتی روبه‌رو بودند.

بعد از این مدل‌ها، مدل‌های بازگشتی (Recurrent Models) به وجود آمدند که مشکلات آن‌ها در فصل گذشته بیان شد [۱۳، ۴۵]. در نهایت، این مشکلات باعث به وجود آمدن ترانسفورمرها شد [۲].

۳.۳ ظهور ترانسفورمرها

در سال ۲۰۱۷، مقاله‌ای توسط گوگل منتشر شد که مفهوم جدیدی به نام ترانسفورمرها را معرفی کرد [۴۸]. این مقاله به موضوع ترجمهٔ ماشینی پرداخت و نشان داد که با استفاده از مکانیزم توجه (Attention Mechanism) می‌توان بسیاری از مشکلات مدل‌های قبلی را حل کرد [۳۰].

مدل‌های ترانسفورمر برخلاف مدل‌های قبلی که از پردازش سریالی استفاده می‌کردند، از پردازش موازی بهره می‌برند. این ویژگی به ترانسفورمرها اجازه می‌دهد که به‌طور همزمان به تمام بخش‌های ورودی توجه کنند. این قابلیت باعث شد که ترانسفورمرها در پردازش تصویر و متن بسیار سریع‌تر و دقیق‌تر از مدل‌های قبلی عمل کنند [۴۸].

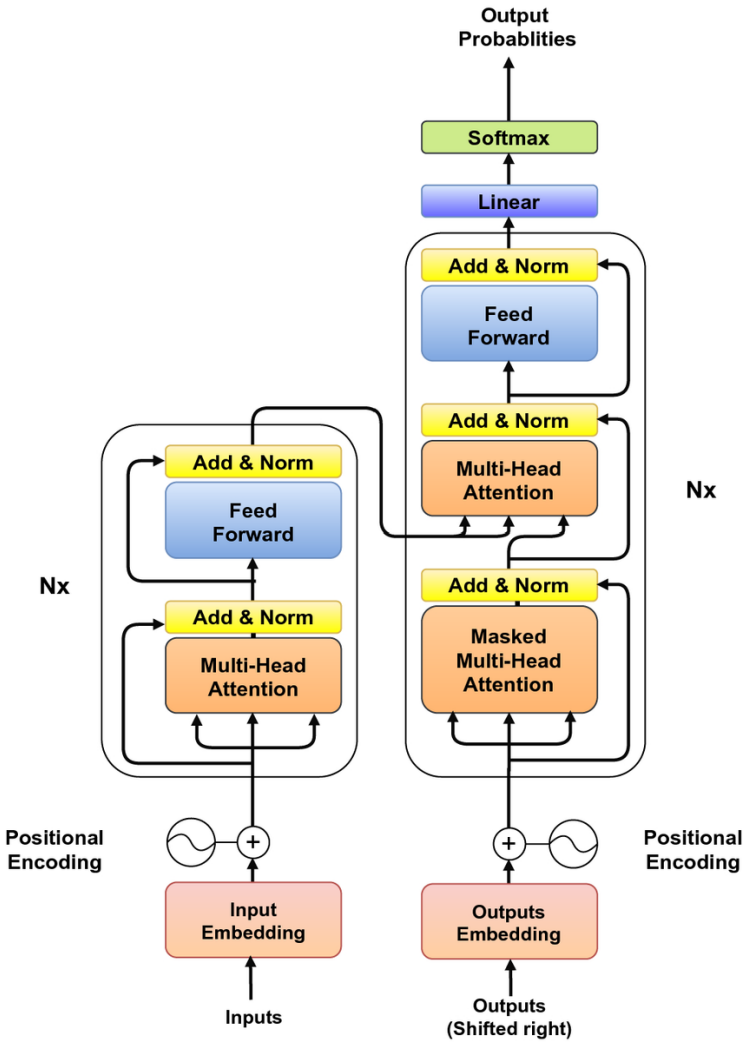
۴.۳ معماری ترانسفورمرها

در تصویر ۳.۴.۱، معماری ترانسفورمر نمایش داده شده است و بخش‌ها و اجزای مختلف آن مشخص شده است. معماری ترانسفورمر از دو بخش اصلی تشکیل شده است:

- انکودر (**Encoder**): وظیفهٔ انکودر این است که دادهٔ ورودی را دریافت کند و ویژگی‌های

آن را استخراج کند.

- دیکودر (Decoder): وظیفهٔ دیکودر این است که ویژگی‌های استخراج‌شده را به زبان مقصد تبدیل کند.



شکل ۳.۴.۱: معماری ترانسفورمرها

۱.۴.۳ Embedding

ددر زبان طبیعی، کلمات به شکل رشته‌های متنی هستند مانند ، و ... . کامپیوترها نمی‌توانند به‌طور مستقیم این کلمات را به شکل رشته‌های متنی پردازش کنند. به همین دلیل، در یادگیری

ماشین این کلمات را به شکل یک بردار نمایش می‌دهیم. این بردار بیانگر آن کلمه در مدل است تا ماشین بتواند آن کلمه را پردازش کند [؟].

این بردارها ویژگی‌های کلمه را در فضای عددی نمایش می‌دهند. روش‌های مختلفی برای تبدیل متن به بردار وجود دارند. از جمله این روش‌ها می‌توان به روش‌های Word2Vec [۳۴] و GloVe [۴۱] اشاره کرد.

همان‌طور که در شکل ۳.۴.۲ نشان داده شده است، هر کلمه که به صورت توکن است، ابتدا در دیکشنری تعریف‌شده پیدا می‌شود و پس از پیدا شدن در دیکشنری، با استفاده از روش‌های Embedding، هر کلمه به برداری از اعداد تبدیل می‌شود. این Embeddingها شباهت‌های معنایی بین کلمات را مدل‌سازی می‌کنند و کلماتی که از نظر معنایی شبیه به هم هستند، بردار آن‌ها نیز به یکدیگر نزدیک‌تر است. به این ترتیب، کلمات برای مدل‌ها و شبکه‌های عصبی قابل‌فهم می‌شوند [۳۴، ۴۱].

شکل ۳.۴.۲: word embedding

۲.۴.۳ positional embedding

ما تا الان هر کلمه را به برداری از اعداد که برای مدل قابل فهم باشد، تبدیل کرده‌ایم. اما مدل‌های ترانسفورمر نمی‌توانند جایگاه هر کلمه را تشخیص دهند. در مدل‌های ترانسفورمر، برخلاف مدل‌های بازگشتی، به دلیل اینکه کلمات به‌صورت موازی وارد می‌شوند، نیاز داریم تا جایگاه هر کلمه را بدانیم. به‌طور مثال، در جملهٔ «من تو را دوست دارم» باید به‌طور دقیق بدانیم که «من» کلمهٔ اول جمله است،

«تو» کلمهٔ دوم جمله است و ... .

حال ما باید به مدل توالی این کلمات را بفهمانیم. بنابراین، نیاز داریم به مدل یک سری اطلاعات اضافی بدهیم به‌طوری‌که مدل توالی کلمات را یاد بگیرد. روش‌های مختلفی برای اضافه‌کردن Positional Embedding به مدل وجود دارد. در ترانسفورمرها از روش Sinusoidal Positional Embedding استفاده می‌شود [۴۸].

این روش قابل یادگیری نیست و صرفاً از یک سری فرمول‌های ساده برای Positional Embedding استفاده می‌کند. برای موقعیت pos در توالی و بُعد i در فضای برداری، تعبیهٔ موقعیتی به‌صورت زیر تعریف می‌شود:

$$PE(pos, 2i) = \sin\left(\frac{pos}{10000^{\frac{2i}{d}}}\right) \tag{۳.۴.۱}$$

و برای مقادیر فرد:

$$PE(pos, 2i+1) = \cos\left(\frac{pos}{10000^{\frac{2i}{d}}}\right) \tag{۳.۴.۲}$$

- pos: موقعیت کلمه در توالی است (مثلاً از 0 تا $N-1$ برای یک توالی Nـتایی).
- i: شاخص بعد در بردار موقعیتی (از 0 تا $d-1$ برای بعد فضای برداری d).
- d: ابعاد فضای برداری مدل که نشان می‌دهد هر کلمه در چند بعد نمایش داده می‌شود.
- 10000: یک مقدار ثابت برای تنظیم مقیاس توابع تناوبی و ایجاد فرکانس‌های مختلف در ابعاد گوناگون.

همان‌طور که در شکل شکل ۳.۴.۳ مشاهده می‌کنید، بعد از embedding کلمات، به آن positional embedding اضافه می‌شود. در این روش از توابع سینوس و کسینوس استفاده می‌شود.

این توابع موقعیت‌ها را در فضای برداری به‌گونه‌ای نگاشت می‌کنند که مدل بتواند از ترتیب کلمات در توالی آگاه باشد [۴۸]. این ویژگی به مدل کمک می‌کند تا توالی زمانی را درک کرده و الگوهای زمانی را شبیه‌سازی کند. از مزایای این روش می‌توان به عدم نیاز به آموزش و توزیع متوازن جایگاه کلمات اشاره کرد.

شکل ۳.۴.۳: word embedding + positional embedding

۳.۴.۳ attention

در روش‌های قدیمی (مانند RNN یا LSTM)، توالی ورودی (مثلاً یک جمله) معمولاً به‌صورت گام‌به‌گام پردازش می‌شد [۱۳، ۲۰]. 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.

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 جمله‌ی «یک گربه روی زمین نشسته است» می‌خواهد بداند کلمه‌ی «گربه» به واژه‌ی «نشستن» بیشتر توجه کند یا به «زمین». در این‌جا فعل «نشستن» ارتباط نزدیک‌تری به «گربه» دارد و از نظر معنایی مرتبط‌تر است.

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

در Scaled Dot-Product Attention [۴۸]، 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:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \qquad (۳.۴.۳)$$

که در آن:

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

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

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

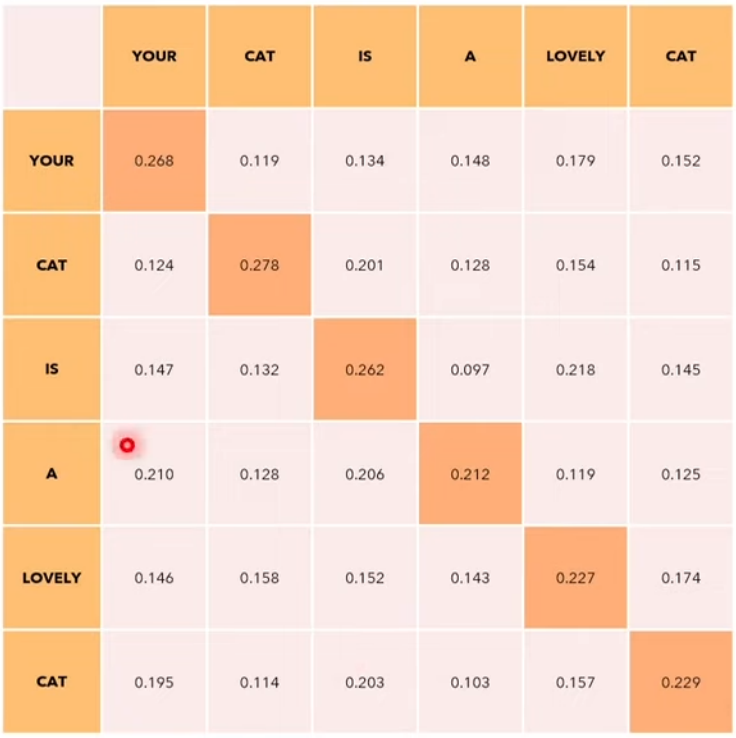
تقسیم بر d_k باعث می‌شود مقدار ضرب داخلی در ابعاد بالا خیلی بزرگ نشود و شیب‌ها (Gradients) پایدار بمانند.

$$\alpha = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) \qquad (۳.۴.۴)$$

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.

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



	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 ۳.۴.۴: Attention

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:

$$\text{head}_i = \text{Attention}(Q_i, K_i, V_i) \tag{۳.۴.۵}$$

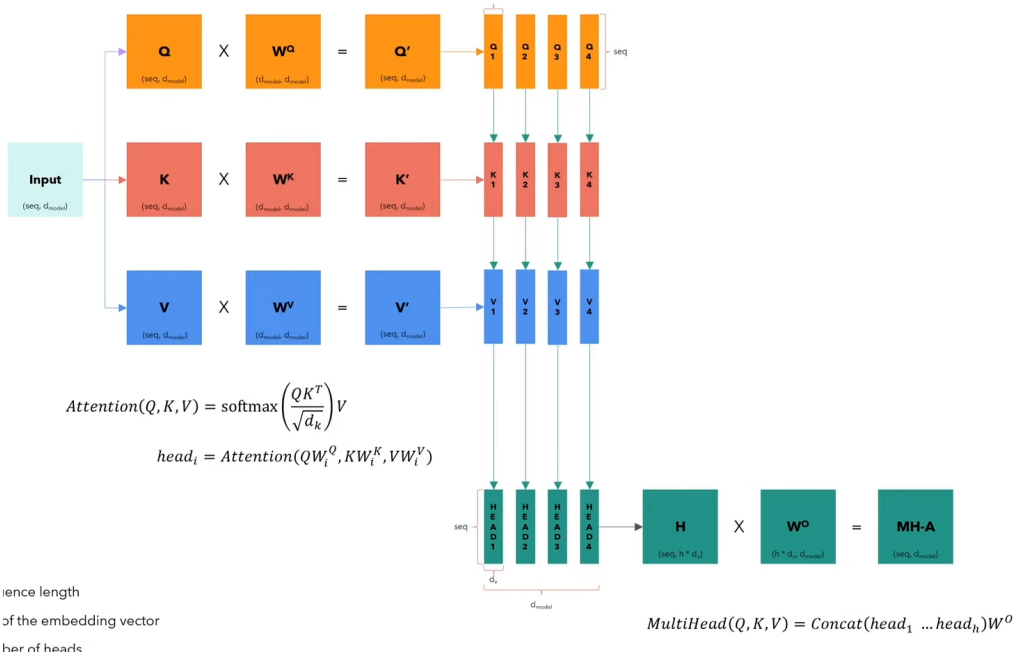
$$\text{MultiHead}(Q, K, V) = [\text{head}_1 \oplus \cdots \oplus \text{head}_h]W_O \tag{۳.۴.۶}$$

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 ۳.۴.۵: multi head attention

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.

۴.۴.۳ Residual Connection (Add)

در معماری‌های عمیق، هنگامی که تعداد لایه‌ها زیاد می‌شود، اغلب دچار ناپایداری گرادیان (Van-ishing/Exploding Gradients) می‌شوند و این مشکل باعث دشواری در آموزش مدل می‌گردد [۳، ۲۰].

در مدل ترانسفورمر [۴۸]، به جای این که خروجی attention را به‌صورت مستقیم به لایهٔ بعدی بدهیم، ورودی آن را نیز حفظ کرده و به خروجی اضافه می‌کنیم. ایدهٔ اصلی این روش از اتصالات باقی‌مانده (Residual Connections) در شبکه‌های عمیق الهام گرفته شده است [۱۷].

اگر x ورودی به زیرماژول و $\text{SubLayer}(x)$ خروجی آن زیرماژول باشد، در انتهای کار عبارت زیر را محاسبه می‌کنیم:

$$x + \text{SubLayer}(x) \tag{۳.۴.۷}$$

این جمع به‌صورت عنصر به‌عنصر (Element-wise Addition) انجام می‌شود.

۵.۴.۳ مزایای Residual Connection 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 gradient vanishing رخ دهد [۳، ۲۰].

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 ناکافی‌بودن یادگیری زیرماژول یا در مراحل اولیهٔ آموزش، دست‌کم بخشی از سیگنال (اطلاعات) خام به لایه‌های بالاتر برسد [۱۷، ۴۸].

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 به‌صورت دست‌نخورده تا حدی منتقل می‌شود.

۵.۳ (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، Batch Normalization باشد که پیش‌تر در کارهای بینایی (CNNها) بسیار مورداستفاده قرار گرفت [۲۲].

Layer Normalization روشی جایگزین است که در ترانسفورمر استفاده می‌شود [۱، ۴۸]. علت اصلی این انتخاب، ماهیت توالی‌محور (Sequence) بودن داده‌ها در NLP و عدم تمایل به وابستگی به آمار مینی‌بچ است.

تفاوت Layer Norm با Batch Norm

Batch Normalization

در Batch Norm، برای نرمال‌سازی، میانگین و واریانس روی تمام نمونه‌های موجود در مینی‌بچ (و نیز در طول ابعاد ویژگی) محاسبه می‌شود [۲۲]. 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 Batch Norm می‌تواند پیچیده شود.

Layer Normalization:

در Layer Norm، برای هر توکن به‌صورت جداگانه (در طول بُعد ویژگی)، میانگین و واریانس گرفته می‌شود [۱]. 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:

$$\mu_i = \frac{1}{d_{\text{model}}} \sum_{k=1}^{d_{\text{model}}} h_{i,k}, \quad \sigma_i^2 = \frac{1}{d_{\text{model}}} \sum_{k=1}^{d_{\text{model}}} (h_{i,k} - \mu_i)^2 \tag{۳.۵.۸}$$

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

$$\hat{h}_{i,k} = \frac{h_{i,k} - \mu_i}{\sqrt{\sigma_i^2 + \epsilon}} \tag{۳.۵.۹}$$

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

$$\text{LayerNorm}(h_i) = \gamma \odot \hat{h}_i + \beta \tag{۳.۵.۱۰}$$

که در آن $\gamma, \beta \in \mathbb{R}^{d_{\text{model}}}$ هستند و $\odot$ ضربِ عنصر به عنصر است [۱].

مزایای Layer Normalization 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 [۱].
- پایدارسازی توزیع فعال‌سازی‌ها: زمانی که مدل در حال یادگیری است، توزیع‌های داخلی لایه‌های میانی ممکن است تغییر کند (پدیدهٔ Internal Covariate Shift). Layer Norm با نرمال‌سازی این توزیع، آموزش را پایدارتر و سریع‌تر می‌کند [۲۲، ۱].
- 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 [۴۸].

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

Residual Connection: 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.

Layer Normalization: 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:

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

۶.۳ decoder

دیکودر در معماری ترانسفورمرها وظیفهٔ تولید خروجی نهایی را بر عهده دارد. این خروجی معمولاً می‌تواند توالی هدف (Target Sequence) باشد، مانند ترجمهٔ یک جمله یا پیش‌بینی توکن‌های بعدی در یک توالی [۴۸].

در این بخش، دیکودر دو ورودی اصلی دارد: ۱. توالی هدف که معمولاً به‌صورت خودکار تولید می‌شود (مثلاً در ترجمهٔ ماشینی یا تولید متن)، ۲. 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 به‌صورت گام‌به‌گام، خروجی نهایی خود را تولید کند [۲، ۴۵].

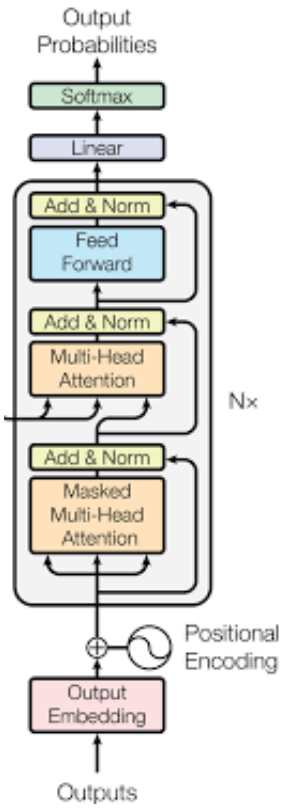
همان‌طور که در شکل ۳.۶.۶ مشاهده می‌کنید، دیکودر دو ورودی دارد.

تمامی بخش‌های دیکودر مانند انکودر هستند اما در دیکودر masked multi-head attention وجود دارد [۴۸].

۷.۳ attention head multi masked

در ترانسفورمر، مکانیزم Multi-Head Attention در بخش دیکودر به‌صورت Masked پیاده‌سازی می‌شود تا مدل نتواند توکن‌های آینده را ببیند و به‌صورت خودبازگشتی (Autoregressive) توکن بعدی را پیش‌بینی کند [۴۸]. در واقع ایدهٔ اصلی استفاده از mask جلوگیری از مشاهدهٔ 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



شکل ۳.۶.۶: Decoder

توکن‌های آینده را «نگاه» کند، پیش‌بینی توکن بعدی آسان و غیرواقعی می‌شود (مشکل نشت اطلاعات) [۲، ۴۵].

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

۸.۳ مثال عددی mask attention

فرض کنید دنبالهٔ ۴ توکنی داریم:

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

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

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

- سطر ۱ (توکن اول): تنها می‌تواند خودش (ستون ۱) را ببیند، اما ستون‌های ۲ تا ۴ ماسک می‌شوند.
- سطر ۲ (توکن دوم): می‌تواند به ستون‌های ۱ و ۲ نگاه کند، اما ستون‌های ۳ و ۴ ماسک می‌شوند.
- سطر ۳: می‌تواند ستون‌های ۱، ۲ و ۳ را ببیند، اما ستون ۴ ماسک می‌شود.
- سطر ۴: می‌تواند به ستون‌های ۱، ۲، ۳ و ۴ دسترسی داشته باشد (چهارمین توکن می‌تواند توکن‌های قبلی را ببیند. همچنین این توکن خودش نیز معمولاً در دسترس است —— بسته به پیاده‌سازی، ممکن است 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 (با نشانه‌گذاری پایین‌مثلثی):

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

به این ترتیب، پس از جمع شدن با S و اجرای `softmax` در هر سطر، ضرایب توجه ستون‌های ماسک‌شده به صفر میل می‌کنند [۴۸].

۹.۳ vision transformer

ایدهٔ ترانسفورمرها در حوزهٔ بینایی (Vision) از تعمیم ترانسفورمر متن به تصاویر به وجود آمده است [۱۱].

ما در این بخش از Vision Transformer برای وظیفهٔ Classification (کلاس‌بندی) استفاده می‌کنیم.

در روش‌های متداول برای پردازش تصویر، از convolutionهای متوالی استفاده می‌کردند؛ اما در ترانسفورمرها تصاویر به پچ‌های مختلف شکسته می‌شوند [۱۱]. هر پچِ شکسته‌شده از تصویر می‌تواند با سایر پچ‌ها به‌صورت موازی وارد مکانیزم توجه (Attention) شود و شباهت یا ارتباطشان با یکدیگر سنجیده شود. در بخش‌های بعد، به‌طور مفصل روند انجام این کار را توضیح خواهیم داد.

۱.۹.۳ patch embedding in vision transformer

در ترانسفورمرهای مبتنی بر متن، هر کلمه به توکن تبدیل می‌شود و سپس هر کلمه به برداری تبدیل می‌گردد. این بردارها پس از افزودن positional embedding وارد مکانیزم Attention می‌شوند [۴۸].

حال همین ایده در تصویر پیاده‌سازی شده است. همان‌طور که در شکل ۳.۹.۷ مشاهده می‌کنید، در Vision Transformer، به‌جای استفاده از عملیات کانولوشن‌های متوالی که در شبکه‌های CNN مرسوم است [۲۷، ۲۶، ۱۷]، تصویر را به بلاک‌های غیرهم‌پوشان ($P \times P$) تقسیم می‌کنیم. این کار علاوه بر ساده‌سازی موازی‌سازی، به مدل اجازه می‌دهد از سازوکار Self-Attention برای ارتباط بین این بلاک‌ها استفاده کند [۱۱].

شکل ۳.۹.۷: patch to image

۲.۹.۳ شکل پچ‌ها:

فرض کنید ابعاد تصویر ورودی ($H \times W \times C$) باشد. به‌عنوان مثال، اگر اندازهٔ تصویر $224 \times 224 \times 3$ باشد، طول و عرض تصویر به‌ترتیب 224 و تصویر دارای سه کانال رنگی است:

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

حال اگر اندازهٔ هر پچ ($P \times P$) باشد (برای نمونه 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 (در ارتفاع15 تا 0) و (در عرض15 تا 0)، - پچ دوم: مختصات (در ارتفاع15 تا 0) و (در عرض31 تا 16)، - و به همین ترتیب تا کل تصویر پوشش داده شود.

۳.۹.۳ تعداد پچ‌ها:

اگر پچ‌ها بدون هم‌پوشانی باشند، ابعاد پچ باید بر ابعاد تصویر بخش‌پذیر باشد.

- تعداد پچ‌های افقی: $\frac{W}{P}$ - تعداد پچ‌های عمودی: $\frac{H}{P}$

در مجموع:

$$\left(\frac{H}{P}\right) \times \left(\frac{W}{P}\right) = \frac{H}{P} \times \frac{W}{P}. \qquad (۳.۹.۱۱)$$

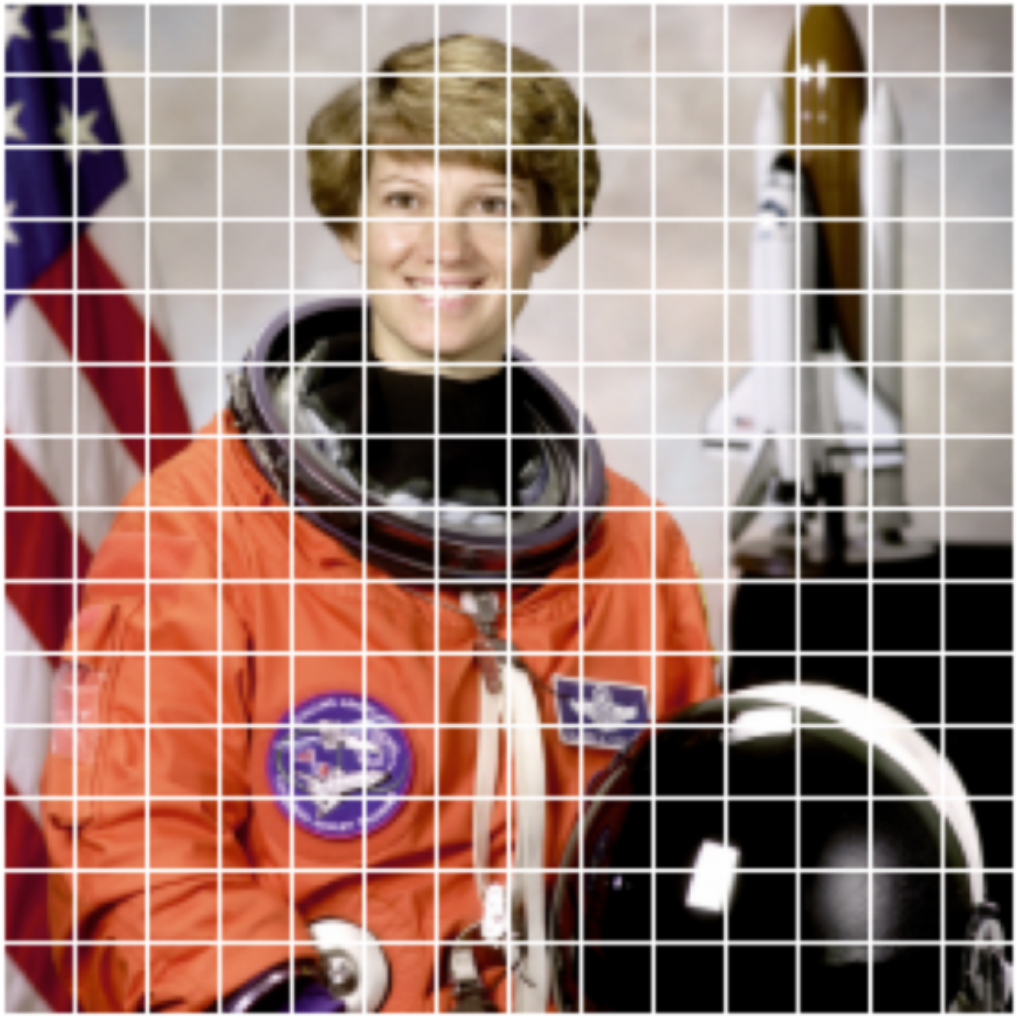
برای مثال اگر:

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

شکل ۳.۹.۸: Image original

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

در اکثر نسخه‌های مبدل‌های بینایی، پچ‌ها بدون هم‌پوشانی (Non-overlapping) هستند. اندازهٔ پچ‌های کوچک باعث می‌شود تعداد پچ‌ها زیاد شود و در نتیجه هزینهٔ Attention بالا رود. از طرفی، پچ‌های بزرگ هزینهٔ Attention را کاهش می‌دهند؛ اما ممکن است جزییات محلی (local details) را از دست بدهیم [۱۱].



شکل ۳.۹.۹: paches of image

۴.۹.۳ بردارکردن هر پچ

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

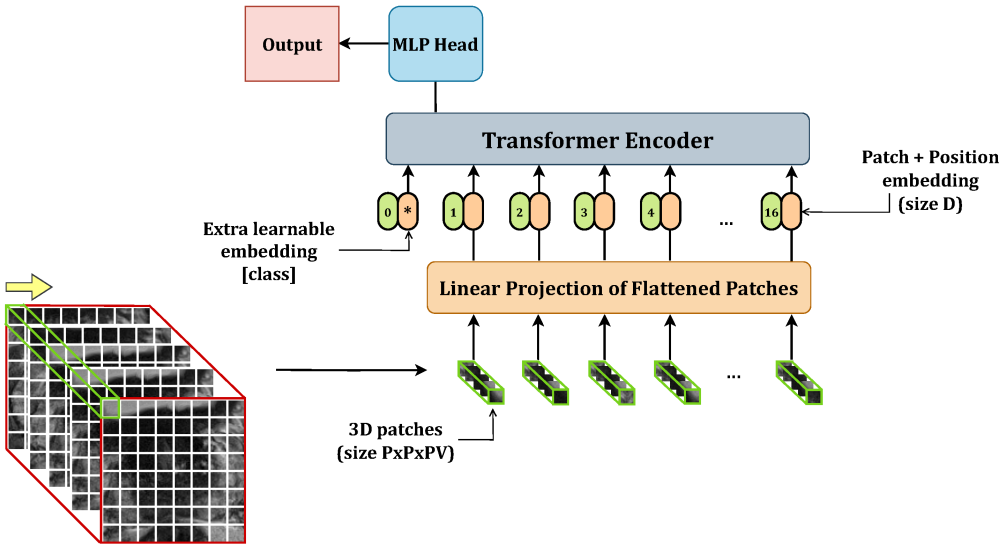
$$P \times P \times C = P^2 \times C. \qquad (۳.۹.۱۲)$$

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

۳.۱۰ 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 یک تبدیل ویژگی (Feature Transformation) 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 ۳.۱۰.۱۰: Embedding in Vision Transformer

این مرحله شبیه ساخت توکن در NLP است؛ با این تفاوت که در NLP، توکن «کلمه» یا «زیرکلمه» است و از قبل دارای بردار تعبیه‌شده (Embedding) بوده است [۴۸]. در Vision Transformer [۱۱]، ما ابتدا باید تصاویر را پچ کنیم و سپس بردارهای Embeddingشده را از این پچ‌ها به دست آوریم.

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

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

هر پچ اکنون یک بردار d_{model}ـ 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 با 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) شکل گیرد و غیره [۴۸، ۱۱].

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 موقعیت‌ها، اطلاعات مکانی تقریبی داشته باشد. در عمل، چون قصد داریم (در مراحل بعد) به هر پچ یک Positional Embedding هم اضافه کنیم، مکان دقیق هر پچ در بُعد دوم (ویژگی) کد می‌شود.

در ویژن ترانسفورمر [۱۱] دیگر به کانولوشن وابسته نیستیم. در عوض، از Embedding استفاده می‌شود. Split کردن تصویر به بلاک‌های $(P \times P)$، Flatten و Linear Projection همگی عملیات ریاضی ساده‌ای هستند که به‌راحتی روی TPU/GPU قابل موازی‌سازی‌اند.

۱.۱۰.۳ CLS Token

CLS Token یک بردار ویژه است که به ابتدای دنبالهٔ ورودی اضافه می‌شود و نقش آن، خلاصه‌کردن اطلاعات کل ورودی (چه متن، چه تصویر) است [۹، ۱۱].

در ویژن ترانسفورمر، این توکن در ابتدای پچ‌های تصویری قرار می‌گیرد. این توکن یک بردار با ابعاد d_{model} است (همان ابعاد سایر توکن‌ها) و پارامتری یادگرفتنی محسوب می‌شود؛ یعنی مدل طی آموزش، مقادیر آن را برای ذخیره و تجمیع اطلاعات بهینه می‌کند.

در وظایف دسته‌بندی (Classification)، هدف این است که یک پیش‌بینی کلی برای کل ورودی (مثلاً یک جمله یا یک تصویر) ارائه دهیم؛ CLS Token دقیقاً همین وظیفه را بر عهده دارد [۹]. این

توکن از طریق مکانیزم Self-Attention در ترانسفورمر با تمامی توکن‌های دیگر (پچ‌های تصویر) ارتباط می‌گیرد و اطلاعات مهم آن‌ها را در لایه‌های مختلف ترانسفورمر به‌صورت تجمعی یاد می‌گیرد. به عبارتی، CLS Token نقش نمایندهٔ کل تصویر یا متن را بر عهده دارد.

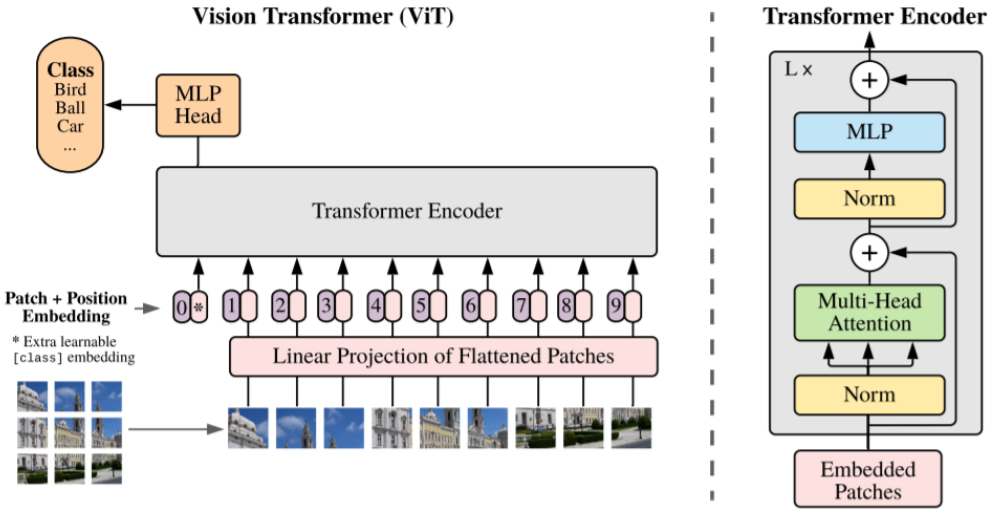
CLS Token از طریق ضرب داخلی در مکانیزم Attention، می‌تواند به تمام پچ‌ها نگاه کند و با ضرایب توجه (α) مشخص کند که از هر پچ چه مقدار اطلاعات بگیرد. بدین‌ترتیب، به‌طور ضمنی یاد می‌گیرد روی ویژگی‌هایی که برای دسته‌بندی مهم هستند (نظیر الگوها، اشکال و بخش‌های کلیدی تصویر) متمرکز شود.

در طول لایه‌های ترانسفورمر، CLS Token نقش محوری در خلاصه‌سازی بازنمایی کل تصویر ایفا می‌کند. این توکن به‌صورت پارامتر قابل یادگیری تعریف شده و در طول فرآیند آموزش به‌روزرسانی می‌شود [۹، ۱۱].

۲.۱۰.۳ Encoder in vision transformer

انکودر در ترانسفورمرها همانند ترانسفورمر اصلی است [۴۸]، با این تفاوت که در Vision Transformer [۱۱] دیگر به دیکودر نمی‌رویم. پس از عبور از بلاک‌های ترانسفورمر، در ساده‌ترین حالت یک لایهٔ خطی (Fully Connected) یا یک لایهٔ MLP (Multi-Layer Perceptron) بر روی بردار نهایی اعمال می‌شود و این لایه‌ها به تعداد کلاس‌ها خروجی می‌دهند. سپس خروجی هر لایه با گذر از تابع softmax به احتمال هر کلاس تبدیل می‌شود و در نهایت مدل کلاس با بیشترین احتمال را به‌عنوان خروجی پیش‌بینی می‌کند.

در ترانسفورمرها، هر لایهٔ انکودر و دیکودر با پردازش عمیق‌تر روی توالی ورودی، می‌تواند نمایش بهتری از ویژگی‌ها به‌دست بیاورد [۴۸]. تکرار چندین‌بارهٔ Encoder یا Decoder موجب می‌شود مدل بتواند ساختارهای پیچیده‌ای را یاد بگیرد و کیفیت و دقت آن در شناسایی توالی‌های طولانی و معانی پنهان افزایش یابد [۴۸، ۱۱]. در نتیجه، مدل با تعداد لایه‌های بیشتر اغلب عملکرد بهتری از خود نشان می‌دهد.



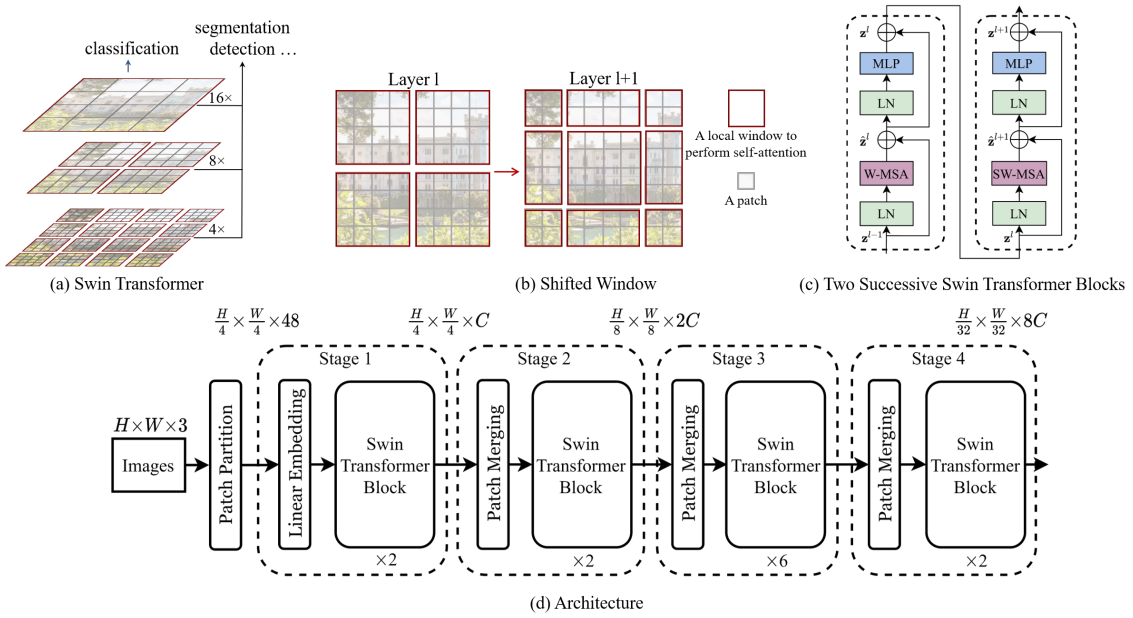
شکل ۳.۱۰.۱۱: Cls Token in Vision Transformer

۱۱.۳ Transformer: Swin

ایدهٔ Swin Transformer از ترکیب چند مفهوم کلیدی در مدل‌های ترانسفورمر و شبکه‌های کانولوشنی شکل گرفت [۴۸، ۱۷، ۲۹]. یکی از بزرگترین مشکلات در ترانسفورمرهای اولیه، نیاز به محاسبات بسیار زیاد در زمانی بود که تصویر ورودی ابعاد بسیار بزرگی داشت [۱۱]. در ترانسفورمر معمولی هر پچ به تمامی پچ‌های دیگر توجه (attention) می‌کرد و در مواقعی که تعداد پچ‌ها زیاد می‌شد، هزینهٔ محاسباتی و حافظه به‌شدت افزایش پیدا می‌کرد.

در شبکه‌های کانولوشنی، معماری معمولاً به‌صورت سلسله‌مراتبی پیش می‌رود [۱۷]؛ یعنی ابتدا ویژگی‌های محلی استخراج می‌شود، سپس با عمیق‌تر شدن لایه‌ها، این ویژگی‌ها در سطوح بالاتر با یکدیگر ترکیب می‌شوند. در Swin Transformer [۲۹]، با دانش بر این موضوع توانسته‌اند هم هزینه‌های محاسباتی را کاهش دهند و هم دقت مدل را افزایش دهند.

در Swin Transformer، به‌جای آنکه مدل به تمام پچ‌ها در یک سطح ویژگی نگاه کند، تصویر را به «پنجره‌های محلی» (Local Windows) تقسیم می‌کند و توجه را محدود به همان ناحیه می‌سازد [۲۹]. سپس با تکنیک جابه‌جایی (Shift) این پنجره‌ها در لایه‌های بعدی، توان مدل برای ترکیب



شکل ۳.۱۱.۱۲: Swin Transformer

اطلاعات از نواحی مختلف تصویر (و در نهایت دیدن کل تصویر) افزایش پیدا می‌کند. این رویکرد، ایدهٔ کلیدی‌ای بود که باعث شد مدل هم محاسبات سبک‌تری داشته باشد و هم بتواند ارتباط‌های جهانی (Global) را در طول لایه‌ها به‌دست آورد.

یکی دیگر از ایده‌های مهم در Swin، کوچک‌کردن تدریجی نقشهٔ ویژگی (Feature Map) در طول معماری است؛ مشابه کاری که در ResNet یا سایر CNNها انجام می‌شود [۱۷]. این امر ضمن کاهش هزینهٔ محاسباتی، باعث می‌شود مدل بتواند با سطوح مختلفی از ویژگی‌ها کار کند و در نهایت خروجی نهایی باکیفیت‌تری ارائه دهد.

۱.۱۱.۳ قطعه‌بندی پچ (Patch Partition)

فرض کنیم تصویر ورودی I دارای ابعاد $(H \times W \times 3)$ باشد. گام نخست، تقسیم تصویر به پچ‌های کوچک $(P \times P)$ است [۱۱]. 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:

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

۲.۱۱.۳ Linear Embedding

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 [۱۱، ۲۹]:

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

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 Swin Transformer خواهد بود [۲۹].

هر بلوک Swin Transformer از چند بخش اصلی تشکیل شده است [۲۹]:

- پنجره‌بندی تصویر (**Window Partition**) یا پنجره‌بندی جابه‌جاشده (**Shifted Window Partition**)
- اعمال WMSA (**Window Multi-Head Self Attention**)
- لایهٔ **Skip Connection** و **Layer Norm**

- مسیر **MLP**:

  - یک لایهٔ MLP شامل دو لایهٔ **Fully-Connected** و تابع فعال‌ساز **GeLU** (یا تابع مشابه)

  - لایهٔ **Skip Connection** و **Layer Norm**

۳.۱۱.۳ Window Multi-Head Self-Attention

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.

۴.۱۱.۳ 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 به‌صورت $\frac{C}{h}$ در نظر گرفته می‌شود که در آن 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 **Swin Transformer**، این محاسبات به‌صورت پنجره‌ای انجام می‌شوند؛ یعنی برای هر پنجره، تنها پیکسل‌های داخل همان پنجره در ماتریس‌های Q، 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.

۵.۱۱.۳ shifted Windows

در **Swin Transformer**، ایدهٔ «پنجره‌های جابه‌جاشده» (*Shifted Windows*) به این منظور ارائه شده است تا مدل، ارتباط پیکسل‌های واقع در پنجره‌های مجاور را هم یاد بگیرد [۲۹]. 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 [۲۹]:

- 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 [۲۹]. 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 شکل ؟؟، 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 [۲۹]. در مقالهٔ اصلی، این مقدار جابه‌جایی معمولاً نیمِ اندازهٔ پنجره $\frac{M}{2}$ 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 Cyclic Shift 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 Cycle Shift در شکل ؟؟ آمده است.

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 [۲۹].

اگر بخواهیم با مثال توضیح دهیم، 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 **Swin Transformer** (**W-MSA** و **SW-MSA**) تکمیل‌کنندهٔ یکدیگر می‌شوند [۲۹]:

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

۶.۱۱.۳ Mlp

پس از انجام *Shifted Window Multi-Head Self-Attention*، خروجی به یک مسیر **MLP** می‌رود [۲۹]. ساختار این MLP به‌صورت زیر است:

$$X' = \mathrm{GELU}(XW_1 + b_1)\, W_2 + b_2, \qquad (۳.۱۱.۱۳)$$

که در آن

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

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

تابع فعال‌ساز GELU (یا ReLU و سایر توابع) نیز در این‌جا قابل استفاده است [۱۸].

۷.۱۱.۳ patch merging

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 اصلی از این کار عبارت است از:

- 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 [۱۷].
- کاهش هزینهٔ محاسبات: در مراحل بعدی، چون ابعاد فضایی کمتر می‌شود، مدل راحت‌تر می‌تواند با ویژگی‌های جدید کار کند (چون مثلاً به‌جای $(H \times W)$ پیکسل، تعداد کمتری پیکسل داریم) [۲۹].

این فرایند کوچک‌سازی در Swin Transformer با نام Patch Merging شناخته می‌شود که شبیه به Downsampling در شبکه‌های کانولوشنی (مثل Pooling یا Stride-Convolution) عمل می‌کند [۲۹].

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

در مرحلهٔ بعد (Stage بعدی)، 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 فرایندی به نام Patch Merging استفاده می‌کنیم [۲۹]:

۱. انتخاب بلوک‌های (2×2)

ابتدا نقشهٔ ویژگی را در بُعد مکانی به بلوک‌های (2×2) تقسیم می‌کنیم. 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 در بُعد کانال به هم می‌چسبانیم (Concat). اگر هر پیکسل یک بردار از بعد C باشد، اکنون بعدِ حاصل از کنار هم گذاشتن این چهار پیکسل می‌شود $4C$. نام این بردار ادغام‌شده را Z' می‌گذاریم.

۳. لایهٔ خطی برای تغییر بعد

وقتی چهار بردار Cـ بعدی را کنار هم می‌گذاریم، یک بردار $4C$ـ بعدی شکل می‌گیرد. 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 دو برابر بزرگ‌تر از قبل اما نه چهار برابر:

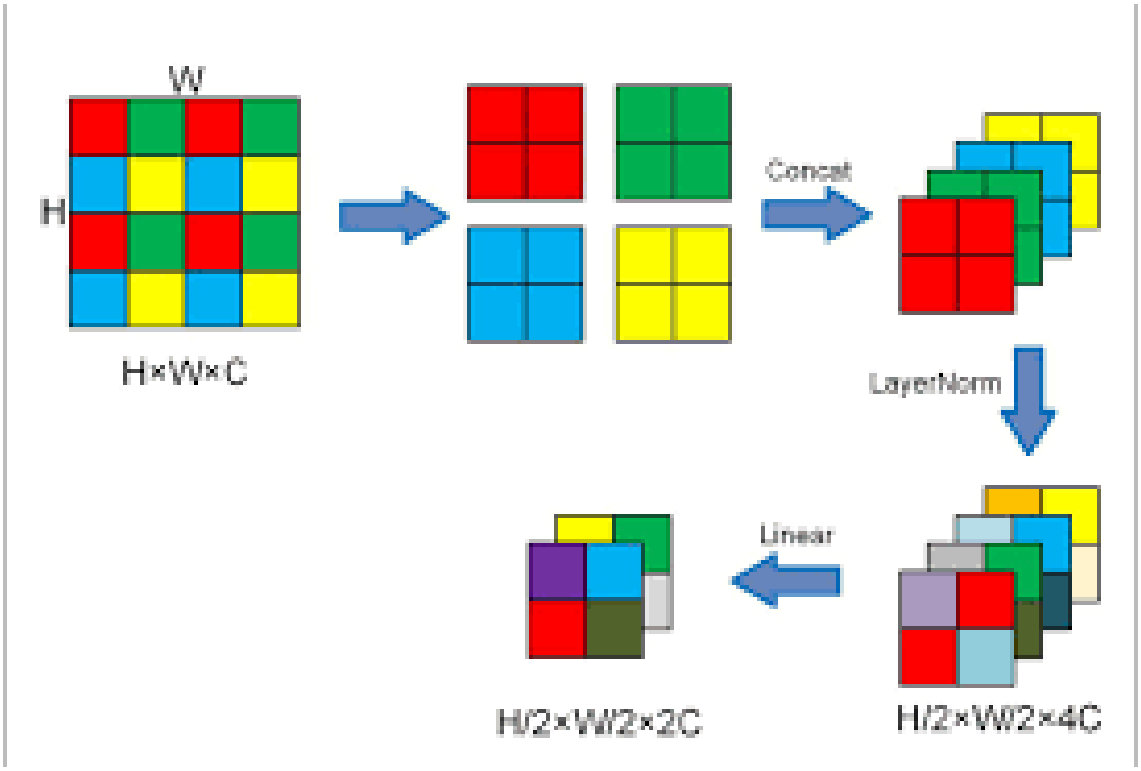
$$Z' \mapsto Z'' = Z' W_{\text{merge}} + b_{\text{merge}}, \qquad (۳.۱۱.۱۴)$$

که بعد ویژگی را از $4C$ به $2C$ کاهش می‌دهد.

۴. کاهش ابعاد مکانی

در عین حال، وقتی هر چهار پیکسل (2×2) را ادغام می‌کنیم، نقشهٔ ویژگی ما ابعاد فضایی $\left(\frac{H}{2P} \times \frac{W}{2P}\right)$ خواهد داشت (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.



شکل ۳.۱۱.۱۳: Merging Patch

در شبکه‌های کانولوشنی، مرتبا از لایه‌های Pooling یا Stride-Convolution برای کوچک‌کردن (Downsampling) ابعاد استفاده می‌شود تا اطلاعات سطح بالاتر (مثل ساختار کلی اشیا) راحت‌تر استخراج شود [۱۷]. Swin Transformer هم همین ایدهٔ سلسله‌مراتب را به دنیای ترانسفورمرها آورده است [۲۹]. همچنین اگر ابعاد فضایی را کم نکنیم، هزینهٔ Attention به‌شدت زیاد می‌شود (چون باید در هر لایه برای همهٔ پیکسل‌ها Attention محاسبه گردد).

در معماری کلی Swin Transformer، پس از Stage 1 و عبور از بلوک‌های W-MSA و SW-MSA، عملیات Patch Merging انجام می‌شود. سپس در Stage 2، ویژگی‌های کوچک‌تری داریم، اما تعداد کانال‌ها افزایش یافته است [۲۹]. مشابه معماری‌های کانولوشنی، با افزایش Depth، ابعاد

فضایی کاهش و تعداد کانال‌ها افزایش پیدا می‌کند.

در انتهای Stage آخر، خروجی به یک لایهٔ FC داده می‌شود تا تعداد کلاس‌ها (num_classes) را پیش‌بینی کند. پس از گذر از Softmax، احتمال هر کلاس به‌دست می‌آید و مدل در نهایت کلاس نهایی را برمی‌گزیند.

فصل ۴

پیشینه پژوهش

تحلیل شبکه‌های ترانسفورمر و CNN در پردازش تصاویر

در شبکه‌های ترانسفورمر، تصاویری که وارد شبکه می‌شوند، به پچ‌هایی با ابعاد مشخص تقسیم می‌شوند (مثل 8×8 یا 16×16 پیکسل). این پچ‌ها به عنوان ورودی به مدل داده می‌شوند، و در طی پردازش، مدل به طور عمومی این پچ‌ها را به صورت غیرمحلی (global) و مستقل از مکان‌هایشان در تصویر پردازش می‌کند.

با این حال، در مدل‌های CNN یا شبکه‌های کانولوشنی، ویژگی‌های محلی (local features) می‌توانند به راحتی شناسایی شوند چون شبکه به طور طبیعی در داخل تصویر حرکت کرده و ویژگی‌های اطراف یک نقطه خاص را تحلیل می‌کند. این ویژگی‌های محلی مثل لبه‌ها، بافت‌ها و اشیاء می‌توانند شناسایی شوند، زیرا هر فیلتر در یک ناحیه محلی از تصویر اعمال می‌شود و اطلاعات محلی را از آن ناحیه استخراج می‌کند.

اما در ترانسفورمرها، چون تصویر به پچ‌های ثابت تقسیم می‌شود و سپس این پچ‌ها به مدل وارد می‌شوند، دید محلی مدل محدود می‌شود. یعنی مدل نمی‌تواند به راحتی ویژگی‌های محلی تصویر را

مانند یک شبکه کانولوشنی شناسایی کند. به عبارت دیگر، مدل برای بررسی ارتباطات و ویژگی‌ها فقط با توجه به پچ‌های جداگانه و بدون آگاهی از ساختار کلی تصویر، عمل می‌کند.

در عین حال، ترانسفورمرها به دلیل ساختار توجه (self-attention) خود می‌توانند به روابط گلوبال (global relationships) نیز توجه داشته باشند. یعنی تمام پچ‌ها می‌توانند به هم متصل شوند و اطلاعاتی از نقاط دورتر تصویر را دریافت کنند. این ویژگی باعث می‌شود که مدل توانایی پردازش اطلاعات جهانی و تطبیق آن با سایر بخش‌های تصویر را داشته باشد.

اما این موضوع که ترانسفورمر نمی‌تواند به طور طبیعی دید محلی داشته باشد، به این معنی است که برخی از اطلاعات مفیدی که برای تحلیل دقیق تصاویر ضروری است، ممکن است از دست برود یا با مشکل مواجه شود. برای رفع این مشکل، معمولاً روش‌هایی مثل استفاده از لایه‌های کانولوشن در کنار ترانسفورمرها یا تقسیم‌بندی بهتر پچ‌ها به کار می‌رود تا شبکه قادر باشد هم دید محلی و هم دید گلوبال را به طور همزمان در اختیار داشته باشد.

۱.۰.۴ ویژگی‌های محلی (Local Features)

در شبکه‌های CNN، فیلترهای کانولوشنی (Convolutional Filters) برای استخراج ویژگی‌های محلی طراحی شده‌اند. این فیلترها معمولاً روی نواحی کوچک تصویر (مانند 3×3 یا 5×5 پیکسل) اعمال می‌شوند. فرض کنید تصویری از یک گربه دارید؛ در لایه‌های ابتدایی یک CNN، این فیلترها ممکن است لبه‌ها (Edges)، گوشه‌ها (Corners)، یا بافت‌های کوچک (Textures) در موهای گربه را شناسایی کنند. این پردازش محلی است زیرا هر فیلتر فقط روی ناحیه کوچکی از تصویر تمرکز می‌کند.

۲.۰.۴ ویژگی‌های جهانی (Global Features)

با عمیق‌تر شدن شبکه و افزایش تعداد لایه‌ها، خروجی لایه‌های ابتدایی (ویژگی‌های محلی) به ویژگی‌های بزرگ‌تر و پیچیده‌تر ترکیب می‌شوند. این فرآیند با استفاده از عملیات‌هایی مثل Pooling

(مانند MaxPooling یا AveragePooling) و فیلترهای بزرگ‌تر انجام می‌شود. برای مثال، پس از چند لایه، CNN ممکن است به جای گوشه‌های گربه، ساختار کل گوش گربه را شناسایی کند. در لایه‌های عمیق‌تر، CNN می‌تواند کل شکل گربه یا حتی دسته‌بندی نهایی (مانند اینکه این یک گربه است) را انجام دهد. این پردازش جهانی است زیرا کل تصویر را برای استنباط ویژگی‌های پیچیده در نظر می‌گیرد.

۳.۰.۴ ترانسفورمرها و محدودیت‌های دید محلی

در ترانسفورمرها، ورودی تصویر به پچ‌های ثابت (مانند 16×16) تقسیم می‌شود و هر پچ به طور مستقل پردازش می‌شود، بدون آنکه ارتباطات بین پیکسل‌های داخل پچ یا بین پچ‌ها به‌صورت محلی در نظر گرفته شود. به عنوان یک مثال مشکل، اگر یک چشم گربه در مرز دو پچ جدا شود، مدل ممکن است این ارتباط محلی بین دو پچ را درک نکند و ویژگی چشم گربه از دست برود.

در ترانسفورمرها، ارتباطات بین پچ‌ها با استفاده از مکانیزم Self-Attention محاسبه می‌شود که به مدل اجازه می‌دهد ارتباطات گلوبال بین تمام پچ‌ها را بررسی کند، اما اغلب ویژگی‌های محلی نهفته در هر پچ نادیده گرفته می‌شوند.

برای حل مشکل دید local ، global ترانسفورمر ها، چند روش را پیاده کرده ایم

۴.۰.۴ روش اول:

۵.۰.۴ تبدیل تصاویر به دو پچ مجزا:

فرض کنید تصویری با اندازهٔ 224×224 پیکسل داریم که اندازه‌ای متداول در دیتاست‌هایی نظیر ImageNet است. این تصویر به دو صورت مختلف به پچ‌هایی با اندازه‌های متفاوت تقسیم می‌شود.

در روش اول، تصویر به بلوک‌هایی با ابعاد 8×8 پیکسل تقسیم می‌شود. در این حالت، تعداد

پچ‌ها در هر ردیف و ستون به ترتیب برابر با 28 است و در مجموع

$$28 \times 28 \; = \; 784$$

پچ از تصویر استخراج می‌شود. هر پچ شامل 8×8 پیکسل است و اگر تصویر دارای سه کانال رنگی باشد (مانند تصاویر RGB)، هر پچ شامل

$$8 \times 8 \times 3 \; = \; 192$$

مقدار عددی خواهد بود. این پچ‌ها پس از تبدیل به بردار، به عنوان ورودی به یکی از مسیرهای پردازشی در ترانسفورمر وارد می‌شوند.

ویک بار دیگر همان تصویر به بلوک‌هایی با ابعاد 16×16 پیکسل تقسیم می‌شود. در این حالت، تعداد پچ‌ها در هر ردیف و ستون به ترتیب 14 است و در مجموع

$$14 \times 14 \; = \; 196$$

پچ ایجاد می‌شود. هر پچ 16×16 پیکسل را شامل می‌شود و در صورت RGB بودن تصویر، هر پچ دارای

$$16 \times 16 \times 3 \; = \; 768$$

مقدار عددی خواهد بود. این پچ‌ها نیز به بردار تبدیل شده و به مسیر پردازشی جداگانه‌ای در ترانسفورمر وارد می‌شوند.

بنابراین در همان ابتدا ما دو تا لایه موازی را در ترانسفورمر پیش میگیریم یکی با دید جزئی و یکی هم با دیدگاه جهانی و در ادامه این دید های جزئی و جهانی را با یک دیگر ترکیب میکنیم اما قبل آن باید یک سری کار ها برای انجام این کار صورت گیرد.

۶.۰.۴ هماهنگ سازی پچ ها:

در این مرحله، هدف آن است که پس از لایه‌های اولیهٔ ترانسفورمر (یا هر مرحله‌ای که پچ‌های 8×8 و 16×16 جاسازی اولیه شده‌اند)، تعداد و ترتیب پچ‌های هر دو مسیر را هماهنگ کنیم تا امکان

ادغام (ترکیب) آن‌ها در لایه‌های بعدی فراهم شود. دو عمل مهم در این بخش اتفاق می‌افتد:

۱. تکرار (Replication) پچ‌های 16×16 به تعداد ۴ برابر

۲. تغییر ترتیب (Re-Order) پچ‌های 8×8

این دو گام باعث می‌شوند در نهایت، هر دو مجموعهٔ پچ، دارای ۷۸۴ ردیف (پچ) باشند و ردیف‌های متقابل در هر دو مجموعه، به ناحیهٔ فضایی یکسانی از تصویر اصلی اشاره کنند. در ادامه، هر یک از این مراحل را با جزئیات بیشتری توضیح می‌دهیم.

تکرار ۴ برابری پچ‌های 16×16

چرا باید تعداد پچ‌های 16×16 را ۴ برابر کنیم؟

اگر تصویر ورودی 224×224 باشد، پچ‌های 16×16 در هر بعد

$$\frac{224}{16} = 14$$

قطعه تولید می‌کنند و بنابراین در کل،

$$14 \times 14 = 196$$

پچ خواهیم داشت. در مقابل، پچ‌های 8×8 به‌خاطر نصف‌بودن ضلع پچ (8 به‌جای 16)، تعداد قطعات در هر بعد دو برابر می‌شود:

$$\frac{224}{8} = 28.$$

پس تعداد کل پچ‌ها

$$28 \times 28 = 784$$

خواهد بود. واضح است که

$$784 = 4 \times 196.$$

یعنی پچ‌های 8×8 چهار برابر بیشتر از پچ‌های 16×16 هستند.

چون قصد داریم در گامی بعدی (مثلاً یک لایه انکودر مشترک) این دو مجموعهٔ پچ را ادغام یا مقایسه کنیم، باید تعداد پچ‌های هر دو مسیر یکسان باشد.

مکانیزم تکرار

برای هم‌اندازه‌کردن این دو مجموعه، هر پچ 16×16 را دقیقاً چهار بار کپی می‌کنیم. به صورت ریاضی، اگر

$$X^{(16\times16)} \in \mathbb{R}^{196\times D}$$

ماتریسی در ابعاد $196 \times D$ باشد (یعنی ۱۹۶ پچ، هر کدام برداری با بعد D)، عمل تکرار به شکل زیر نوشته می‌شود:

$$\tilde{X}^{(16\times16)} = \underbrace{\left[X^{(16\times16)}, X^{(16\times16)}, X^{(16\times16)}, X^{(16\times16)}\right]}_{\text{تکرار ۴ مرتبه}} \in \mathbb{R}^{784\times D}.$$

عملگر $[\cdot]$ در این‌جا به معنای الحاق (Concatenate) در راستای بُعد اول (تعداد پچ‌ها) است. در نتیجه، ۴ نسخهٔ یکسان از $X^{(16\times16)}$ پشت سر هم قرار می‌گیرند و ابعاد نهایی به $784 \times D$ می‌رسد.

از نظر مفهومی، چنین برداشتی وجود دارد که هریک از پچ‌های 16×16، وقتی روی تصویر اصلی نگاه کنیم، با چهار منطقهٔ کوچک‌تر 8×8 هم‌پوشانی دارد (چون 16×16 ازلحاظ مساحت ۴ برابر 8×8 است). اما فعلاً صرفاً از نظر تعداد، آن را ۴ مرتبه تکرار می‌کنیم؛ بعداً در مرحلهٔ «تغییر ترتیب» توضیح می‌دهیم که چگونه می‌توان این تکرار را به بخش‌های تصویر ربط داد.

تغییر ترتیب (Re-Order) پچ‌های 8×8

اکنون که پچ‌های 16×16 به‌صورت ۴برابر تکرار شده و به ۷۸۴ پچ رسیده‌اند، می‌خواهیم پچ‌های 8×8 را نیز به شکلی بازآرایی کنیم که هر گروه ۴تایی از پچ‌های 8×8 دقیقاً متناظر با یک پچ

16×16 باشد. این متناظر بودن از نظر موقعیت مکانی در تصویر اهمیت دارد.

چرا بازآرایی (Re-Order) لازم است؟

در استخراج اولیهٔ پچ‌های 8×8، معمولاً طبق یک ترتیب خطی (مانند Row-Major) از گوشهٔ بالا ـ چپ تصویر تا گوشهٔ پایین ـ راست حرکت می‌کنیم و پچ‌ها را شماره‌گذاری می‌کنیم (۱، ۲، ۳، ... ۷۸۴). در این شماره‌گذاری عادی، پچ‌های ۱، ۲، ۳ و ۴ لزوماً در کنار هم قرار دارند، اما این هم‌جواری ممکن است دقیقاً با پچ اولِ 16×16 منطبق نباشد.

برای مثال، ممکن است پچ ۱ در 8×8 با پیکسل‌های ردیف ۰ تا ۷ و ستون ۰ تا ۷ هم‌پوشانی داشته باشد، درحالی‌که پچ ۲ در 8×8 مربوط به ردیف ۰ تا ۷ و ستون ۸ تا ۱۵ است. اگر بگوییم پچ اولِ 16×16 (که کل ناحیهٔ صفر تا ۱۵ در سطر و صفر تا ۱۵ در ستون را می‌پوشاند) با ۴ پچ 8×8 متناظر است، لازم است به‌درستی تشخیص دهیم که آن ۴ پچ در کدام شماره‌های ۱ تا ۷۸۴ قرار گرفته‌اند. برای مثال (در یک چینش فرضی):

- پچ‌های (۱، ۲، ۲۹، ۳۰) از میان 8×8 احتمالاً چهار بخش کوچکی هستند که روی‌هم پیکسل‌های سطر ۰..۱۵ و ستون ۰..۱۵ را می‌پوشانند. پس این ۴ پچ باهم معادل پچ اولِ 16×16 هستند.
- پچ دومِ 16×16 ممکن است با پچ‌های (۳، ۴، ۳۱، ۳۲) در 8×8 هم‌پوشانی داشته باشد، و به همین شکل ادامه می‌یابد.

بنابراین برای اینکه «ردیف اول تکرارشدهٔ پچ 16×16» با «۴ ردیف درست از پچ‌های 8×8» روبه‌رو شود، باید ترتیب پچ‌های 8×8 دقیقاً طبق این نقشهٔ فضایی بازآرایی (Re-Order) شود.

تابع ReOrder

به‌صورت ریاضی، می‌توان این بازآرایی را به شکل یک تابع $\mathrm{ReOrder}(\cdot)$ نشان داد. اگر

$$X^{(8\times8)} \in \mathbb{R}^{784\times D}$$

ماتریسی با ابعاد $784 \times D$ باشد (شماره‌گذاری ردیفی عادی)، خروجی زیر را خواهیم داشت:

$$\hat{X}^{(8\times8)} \;=\; \mathrm{ReOrder}\big(X^{(8\times8)}\big) \;\in\; \mathbb{R}^{784\times D}.$$

وظیفهٔ ReOrder آن است که ردیف‌های $X^{(8\times8)}$ را طوری جابه‌جا کند که ۴ ردیف پشت‌سرهم در $\hat{X}^{(8\times8)}$ دقیقاً همان چهار بخشی از تصویر باشند که یک پچ خاص 16×16 (در حالت تکرارشده) روی آن قرار دارد. به‌عبارت دیگر، از ۱، ۲، ۳، ۴ در چینش عادی، ممکن است تبدیل به ۱، ۲، ۲۹، ۳۰ شود (اگر چنین ترتیبی در صفحهٔ تصویر باهم منطبق است).

Positional Embedding ۷.۰.۴

در این مرحله که هماهنگ‌سازی پچ‌ها (Alignment Patch) به اتمام رسیده و هر دو مجموعهٔ پچ (مسیر 8×8 و مسیر 16×16 تکرارشده) دارای ابعاد یکسان $(784 \times D)$ و ترتیب متناظر هستند، می‌توان جاسازی مکانی (Embedding Positional) را اعمال کرد. هدف از افزودن Positional Embedding آن است که مدل بتواند جایگاه هر پچ در تصویر اصلی را درک کند و صرفاً با بردارهای ویژگی انتزاعی مواجه نباشد.

اغلب در مدل‌های ترانسفورمر بینایی، برای هر پچ (صرف‌نظر از اندازه‌اش) یک بردار مکان (Embedding Position) پیش‌بینی می‌شود که در همان ابتدای مسیر با بردار ویژگی پچ جمع می‌گردد. اما در رویکرد فعلی، چون ما ابتدا لازم داشتیم پچ‌های 16×16 را تکرار کنیم و پچ‌های 8×8 را تغییر ترتیب بدهیم، بهتر است پس از این بازآرایی، Embedding Positional را به‌گونه‌ای اعمال کنیم که دقیقاً منعکس‌کنندهٔ جایگاه نهایی هر پچ در ترتیب هماهنگ‌شده باشد.

در غیر این صورت، اگر قبل از هماهنگی، Embedding Positional اعمال شده بود، تکرار و جابه‌جایی پچ‌ها ممکن است ساختار مکان‌یابی آن‌ها را به‌هم بریزد یا نیاز به به‌روزرسانی مجدد Embedding Position باشد.

از آنجا که هر دو مجموعهٔ پچ (8×8 و 16×16 تکرارشده) پس از هماهنگ‌سازی در ابعاد

$\mathbb{R}^{784\times D}$ هستند، ماتریس جاسازی مکانی (E_{pos}) نیز باید ۷۸۴ سطر داشته باشد. در نتیجه:

$$E_{\text{pos}} \in \mathbb{R}^{784\times D},$$

که در آن هر سطر از E_{pos} مختص یک پچ (ردیف) در خروجی مرحلهٔ هماهنگ‌سازی است.

در این روش، تنها از یک مجموعهٔ E_{pos} مشترک برای هر دو نوع پچ استفاده می‌شود. جون هر دو ابعاد یکسان هستند و embedding positional پارامتر یادگیرنده ندارد میتوان از یک positional embedding استفاده کرد.

۸.۰.۴ لایه های اول تا هشتم انکودر

پس از آنکه Embedding Positional به پچ‌های هر دو مسیر اعمال شد، عملاً هر دو مجموعهٔ خروجی دارای شکل و ابعاد یکسان $(784 \times D)$ هستند (در این مرحله فرض گرفته‌ایم token CLS وجود نداشته باشد یا در محاسبات فعلی نادیده گرفته شود). این امر باعث می‌شود که در هر دو مسیر، Query، Key و Value در مکانیزم Self-Attention نیز ابعاد یکسانی داشته باشند.

۹.۰.۴ لایه نهم انکودر

در لایهٔ نهم، ابتدا Query و Key را برای هر مسیر به‌صورت جداگانه محاسبه می‌کنیم. طبق روال استاندارد ترانسفورمر، هر ورودی با ماتریس‌های وزنیِ یادگیری‌پذیر (W_Q و W_K) ضرب می‌شود تا به فضاهای Q و K نگاشت شود. بسته به طراحی، می‌توان از همان وزن‌ها یا وزن‌های جداگانه استفاده کرد؛ اما برای سادگی، فرض کنیم وزن‌ها مشترک هستند:

$$Q^{(8)} = X^{(8)}W_Q, \quad K^{(8)} = X^{(8)}W_K,$$

$$Q^{(16)} = X^{(16)}W_Q, \quad K^{(16)} = X^{(16)}W_K.$$

هرکدام از $Q^{(8)}$ و $Q^{(16)}$ ابعادی معادل $\mathbb{R}^{784\times d_k}$ دارند (d_k معمولاً $\frac{D}{h}$ در صورت چندسری بودن Attention است، یا ممکن است با D برابر باشد در صورت تک‌سری).

به‌طور مشابه $K^{(8)}$ و $K^{(16)}$ نیز ابعادی معادل $\mathbb{R}^{784\times d_k}$ دارند.

۱۰.۰.۴ محاسبهٔ ماتریس شباهت (QK^T) و میانگین‌گیری

مکانیزم خودتوجهی (Self-Attention) معمولاً از ضرب Q در K^T برای محاسبهٔ میزان شباهت پچ‌ها استفاده می‌کند. شما می‌خواهید قبل از Softmax، میانگین شباهت‌های دو مسیر را بگیرید. بنابراین به این ترتیب عمل می‌کنیم:

شباهت مسیر 8×8:

$$S^{(8)} = Q^{(8)}K^{(8)^T} \in \mathbb{R}^{784\times 784}$$

شباهت مسیر 16×16:

$$S^{(16)} = Q^{(16)}K^{(16)^T} \in \mathbb{R}^{784\times 784}$$

ادغام شباهت‌ها:

سپس برای ادغام این دو شباهت، از میانگین‌گیری استفاده می‌کنیم:

$$S_{\text{merged}} = \frac{S^{(8)} + S^{(16)}}{2}.$$

در هر دوی $S^{(8)}$ و $S^{(16)}$ ابعاد $\mathbb{R}^{784\times 784}$ دارند و بنابراین جمع‌کردن و میانگین‌گیری آن‌ها بدون مشکل صورت می‌گیرد.

۱۱.۰.۴ اعمال مقیاس‌بندی $\frac{1}{\sqrt{d_k}}$ و Softmax

در نسخهٔ کلاسیک Attention، ماتریس شباهت QK^T معمولاً با ضریب $\frac{1}{\sqrt{d_k}}$ مقیاس (Scaling) می‌شود تا مقادیر بزرگ در ماتریس شباهت کنترل شوند و یادگیری پایدارتر شود:

$$\tilde{S}_{\text{merged}} = \frac{1}{\sqrt{d_k}} S_{\text{merged}} = \frac{1}{\sqrt{d_k}} \cdot \frac{S^{(8)} + S^{(16)}}{2} = \frac{S^{(8)} + S^{(16)}}{2\sqrt{d_k}}.$$

در گام بعدی، باید بر روی هر سطر این ماتریس $\tilde{S}_{\text{merged}}$ عمل Softmax انجام دهیم تا ضرایب توجه (A) به دست آید:

$$A = \text{softmax}(\tilde{S}_{\text{merged}}) \in \mathbb{R}^{784\times784}.$$

به‌عبارت دیگر، برای هر عنصر A_{ij}:

$$A_{ij} = \frac{\exp(\tilde{S}_{\text{merged},ij})}{\sum_{k=1}^{784} \exp(\tilde{S}_{\text{merged},ik})}$$

این ماتریس A نشان‌دهندهٔ وزن‌های توجه بین هر دو پچ است. در اینجا:

- سطرها نشان‌دهندهٔ پچ‌های Query هستند.
- ستون‌ها نشان‌دهندهٔ پچ‌های Key هستند.

در این مرحله، خروجی نهایی مکانیزم توجه (Attention) تنها بر اساس بردارهای V مربوط به پچ‌های 8×8 تولید می‌شود. در معماری استاندارد ترانسفورمر، پس از محاسبهٔ نقشهٔ توجه $\text{softmax}(QK^T)$، نتیجه در بردارهای ارزش (V) ضرب می‌گردد تا بردار نهایی توجه به دست آید:

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

اما ما در اینجا از بردارهای V صرفاً متعلق به مسیر پچ‌های 8×8 استفاده می‌شود. برای این منظور، ابتدا با استفاده از ماتریس وزنی W_V، بردار ارزش $V^{(8)}$ را از $X^{(8)}$ (بردار ویژگی پچ‌های 8×8) استخراج می‌کنیم:

$$V^{(8)} = X^{(8)} W_V \in \mathbb{R}^{784\times d_v},$$

که در آن W_V یک ماتریس یادگیری‌پذیر با ابعاد $D \times d_v$ است.

پس از آن‌که نقشهٔ توجه نهایی A (محاسبه‌شده بر پایهٔ ترکیب میانگین‌شده از شباهت‌های مربوط به مسیرهای 8×8 و 16×16) شکل گرفت، خروجی مکانیزم توجه ($O_{\text{Attention}}$) با ضرب A در $V^{(8)}$ حاصل می‌شود:

$$O_{\text{Attention}} = AV^{(8)} = \text{softmax}\left(\frac{S^{(8)} + S^{(16)}}{2\sqrt{d_k}}\right) V^{(8)},$$

که در آن:

$$A = \text{softmax}\left(\frac{S^{(8)} + S^{(16)}}{2\sqrt{d_k}}\right),$$

و:

$$S^{(8)} = Q^{(8)}K^{(8)^T}, \quad S^{(16)} = Q^{(16)}K^{(16)^T}.$$

با جای‌گذاری کامل، فرمول زیر به‌دست می‌آید:

$$O_{\text{Attention}} = \text{softmax}\left(\frac{1}{2\sqrt{d_k}}\left(Q^{(8)}K^{(8)^T} + Q^{(16)}K^{(16)^T}\right)\right) V^{(8)}.$$

مزیت این رویکرد

به این ترتیب:

- وزن‌های توجه (A) از ترکیب میانگین‌شدهٔ شباهت‌های دو مسیر (اطلاعات محلی از پچ‌های کوچک و اطلاعات کلی از پچ‌های بزرگ) به دست می‌آید.
- بردار ارزش (V) تنها از مسیر پچ‌های 8×8 استخراج می‌شود.

این روش باعث می‌شود ویژگی‌های محلی (که در $V^{(8)}$ متمرکز هستند) مستقیماً در خروجی نهایی منعکس شوند، اما وزن‌دهی به این ویژگی‌ها تحت تأثیر هر دو نما (محلی و کلی) انجام گیرد. به بیان دیگر، با وجود آن‌که اطلاعات ارزش از مسیر ریزدانه انتخاب می‌شود، مکانیزم توجه نقشهٔ

شباهت خود را از ادغام دو مقیاس به دست می‌آورد. این رویکرد می‌تواند توازنی مطلوب میان جزئی‌نگری و شناخت ساختار وسیع‌تر تصویر برقرار سازد.

۱۲.۰.۴ ادغام وزنی

به‌جای آنکه شباهت‌های مربوط به پچ‌های 8×8 و 16×16 را به شکل مساوی (ضریب $\frac{1}{2}$) با هم جمع کنیم، می‌توان از پارامتری یادگیری‌پذیر (parameter trainable) به‌نام α استفاده کرد که در بازهٔ $[0, 1]$ قرار دارد. فرمول ترکیب ماتریس شباهت‌ها ($S^{(8)}$ و $S^{(16)}$) به‌شکل زیر تغییر می‌کند:

$$S_{\text{merged}} = \alpha S^{(8)} + (1 - \alpha) S^{(16)}.$$

تأثیر مقدار α در مدل

- اگر α به‌سمت ۱ متمایل شود، نقش پچ‌های 8×8 در توجه مدل پررنگ‌تر خواهد شد.
- اگر α کوچک باشد، توجه بیشتری به پچ‌های 16×16 اختصاص داده می‌شود.

با استفاده از پارامتر α انعطاف پذیری مدل به صورت قابل توجه ای افزایش پیدا میکند.

آموزش پارامتر α

خود مدل در فرایند آموزش با پس‌انتشار خطا (Back-Propagation) می‌تواند مقدار بهینهٔ α را بیاموزد. این انعطاف‌پذیری به مدل اجازه می‌دهد تا به‌طور خودکار تعادلی میان اطلاعات جزئی (از پچ‌های کوچک‌تر) و اطلاعات کلی (از پچ‌های بزرگ‌تر) برقرار کند.

۱.۴ روش دوم sampling: down

در sampling down پس از چند مرحله پردازش (بلوک‌های ترنسفورمر)، حجم توکن‌های پچ را به شکل مؤثری کاهش دهیم تا هم هزینه‌ی محاسباتی (به‌ویژه در عملیات توجه چندسری) کمتر

شود و هم مدل به‌تدریج بتواند از حالت توجه به جزییات ریز (ویژگی‌های محلی) به نمایی کلی‌تر از تصویر (ویژگی‌های کلی) برسد.

در این مدل ما token cls را تا انتها دست نخورده باقی میگذاریم.

پس از عبور تصویر از مرحله‌ی پَچ‌گذاری Patch) (Embedding و الحاق توکن [CLS] ، تعداد توکن‌ها در ابتدا به‌صورت:

$$N = 1 + 784 = 785 \tag{۴.۱.۱}$$

در نظر گرفته می‌شود. در این رابطه، عدد 784 بیانگر تقسیم‌بندی یک تصویر 224×224 به پچ‌های 8×8 (یعنی $28 \times 28 = 784$ پچ) و عدد 1 توکن ویژه‌ی [CLS] است. همچنین بعدِ ویژگی هر توکن (شامل پچ‌ها و [CLS]) برابر با 768 خواهد بود. ازاین‌رو، شکل ورودی در شروع کار:

$$(B, 785, 768) \tag{۴.۱.۲}$$

است که B اندازه‌ی بچ Batch) (Size محسوب می‌شود.

در حالت نرمال در ترانسفورمرها اندازه ورودی اتنشن ها در بلوک های انکودر تغییری نمیکنند.

روش دانسمپلینگ جفتی بر این ایده استوار است که توکن [CLS] (ابتدای توالی) دست‌نخورده باقی بمانَد و سایر توکن‌ها (نماینده‌ی پچ‌ها) را دوتادوتا با هم میانگین بگیریم. اگر ورودی ما

$$(B, N, 768) \tag{۴.۱.۳}$$

باشد و در آن $N - 1$ توکن پچ وجود داشته باشد، آنگاه با جفت‌کردن و میانگین‌گیری پچ‌ها، تعداد نهایی از رابطه‌ی

$$N_{new} = 1 + \frac{N-1}{2} = \frac{N+1}{2} \tag{۴.۱.۴}$$

به‌دست می‌آید. توکن [CLS] همچنان در موقعیت اول باقی می‌مانَد و ابعاد ویژگی (یعنی 768) تغییری نمی‌کند.

بنابراین، پس از بلوک سوم که همچنان ورودی $(B, 785, 768)$ دارد و خروجی همان $(B, 785, 768)$ را تولید می‌کند، با اجرای دانسمپلینگ جفتی تعداد توکن‌ها از 785 به $393 = \frac{784}{2} + 1$ کاهش می‌یابد؛ درنتیجه ورودی بلوک بعدی $(B, 393, 768)$ خواهد بود.

و همین کار پس از بلوک انکودر ششم و نهم اعمال می شود. و همینطور ورودی انکودر بعدی کم و کم تر میشود.

به‌طور خلاصه، ابعاد ورودی هر بلوک پس از آنکه دانسمپلینگ در انتهای بلوک‌های ۳، ۶ و ۹ اعمال شود، بدین ترتیب تغییر می‌کند:

$$(B, 785, 768) \xrightarrow{\text{بلوک ۳ + دانسمپلینگ}} (B, 393, 768),$$

$$(B, 393, 768) \xrightarrow{\text{بلوک ۶ + دانسمپلینگ}} (B, 197, 768),$$

$$(B, 197, 768) \xrightarrow{\text{بلوک ۹ + دانسمپلینگ}} (B, 99, 768).$$

۱.۱.۴ حرکت تدریجی از جزئیات به کلیت

در لایه‌های اولیه، وقتی هنوز دانسمپلینگ انجام نشده، مدل همهٔ پچ‌های ریز و «اطلاعات محلی» را در اختیار دارد و مسسسی‌تواند ویژگی‌های ظریف را پردازش کند. اما وقتی چند لایه گذشت و وارد بلوک‌های بالاتر شدیم، با اعمال میانگین‌گیری جفتی، اطلاعات هر دو پچ مجاور در یک بردار ادغام می‌شود. این وضعیت را می‌توان شکل‌گیری نوعی «نمای کلی‌تر» از تصویر دانست؛ زیرا به‌جای ۲ پچ مجزا، حالا یک پچ ترکیبی داریم که اطلاعاتشان را در خود گنجانده است. به این ترتیب، مدل در سطوح بالاتر روی ویژگی‌های انتزاعی‌تر یا خلاصه‌تر متمرکز می‌شود و نیازی نیست همچنان هزینه‌ی نگه‌داشتن تمام جزئیات محلی را بپردازد.

۲.۱.۴ کم نشدن پارامتر ها در این مدل

در معماری‌های ترنسفورمر، پارامترهای قابل یادگیری تنها به شکل وزن‌ها و بایاس‌هایی تعریف می‌شوند که یا در مرحله‌ی تعبیه‌سازی ،(Embedding) یا در بخش توجه چندسری، یا در شبکه‌های MLP هر بلوک ترنسفورمر به‌کار می‌روند. نکته‌ی کلیدی این است که ابعاد اغلب این وزن‌ها و بایاس‌ها تنها به بُعد پنهان (Hidden Dimension) یا نرخ گسترش (Expansion Ratio) وابسته است و تغییری در آن‌ها به تناسب کم یا زیاد شدن تعداد توکن‌های ورودی به وجود نمی‌آید.

لایه‌های تعبیه‌ساز و موقعیتی (Positional Embedding) ابتدا در این بخش، پارامترها به‌صورت ماتریس‌ها یا بردارهایی تعریف می‌شوند که بُعد آن‌ها با بُعد پنهان (مثلاً ۷۶۸) تنظیم می‌شود و همچنین به طول حداکثری توالی وابستگی دارد (مثلاً اگر حداکثر تعداد توکن ۷۸۵ در نظر گرفته شود، در همان ابتدای تعریف پارامتر شکل می‌گیرد). در هر صورت، این پارامترها فارغ از آن که در عمل چند توکن مؤثر باقی بماند، ثابت خواهند ماند.

توجه چندسری (Multi-Head Attention) در این بخش، ماتریس‌های W_Q، W_K، W_V و W_O موجودند که ابعادشان همگی تابعی از بُعد ویژگی D هستند؛ برای مثال اگر $D = 768$ باشد، هر کدام از این ماتریس‌ها در ابعاد ثابتی (مانند 768×768) تعریف شده و یاد گرفته می‌شود. این پارامترها به «تعداد توکن» بستگی ندارند؛ بلکه تعیین می‌کنند چگونه هر توکن در فضای ویژگی (Feature Space) نگاشت یا پردازش شود.

شبکه‌های MLP داخل هر بلوک ترنسفورمر در این بخش، وزن‌ها و بایاس‌ها به‌صورت W_1, b_1 و W_2, b_2 تعریف می‌شوند. ابعاد این لایه‌ها عموماً از قانون

$$D \to 4D \to D$$

پیروی می‌کند (اگر نسبت گسترش ۴ باشد). در نتیجه، شکل W_1 و W_2 هم تنها وابسته به D است. تعداد کم یا زیاد شدن توکن‌ها (مثلاً نصف‌شدن توکن‌ها پس از دانسمپلینگ) صرفاً بر روی تعداد محاسبات اثر می‌گذارد، اما ساختار و تعداد این پارامترها را دگرگون نمی‌کند.

در یک لایهٔ ترنسفورمر، مهم‌ترین قسمت از نظر هزینه، محاسبهٔ Self-Attention است. اگر تعداد توکن‌ها را N_{tokens} بنامیم و فرض کنیم بعد بردار ویژگی‌ها D باشد، آنگاه مهم‌ترین بخش محاسباتی مربوط به ضرب ماتریسی $QK^\top$ است که پیچیدگی زمانی $O(N_{\text{tokens}}^2 \cdot D)$ دارد.

وقتی دانسمپلینگ جفتی انجام می‌دهیم و تعداد توکن‌ها را از N_{tokens} به حدود $\frac{N_{\text{tokens}}}{2}$ کاهش می‌دهیم، پیچیدگی زمانی در لایه‌های بعدی به شکل زیر تغییر می‌کند:

$$O(N_{\text{tokens}}^2 \cdot D) \quad \longrightarrow \quad O\left(\left(\frac{N_{\text{tokens}}}{2}\right)^2 \cdot D\right) \longrightarrow O\left(\frac{N_{\text{tokens}}^2}{4} \cdot D\right).$$

و به این ترتیب حجم محاسبات به شکل چشم گیری کم میشود.

فصل ۵

آزمایشات و نتایج

کتاب‌نامه

[1] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016. 31, 32, 33

[2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *Proceedings of the 2015 International Conference on Learning Representations (ICLR)*, San Diego, CA, 2015. 19, 21, 22, 34, 35

[3] Yoshua Bengio et al. *Learning long-term dependencies with gradient descent is difficult*. IEEE Transactions on Neural Networks, 1994. 30

[4] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, New York, 2006. 6, 7, 10, 11

[5] Peter F Brown, Vincent J. Della Pietra, Stephen A. Della Pietra, and Robert L Mercer. The mathematics of statistical machine translation: Parameter estimation. *Computational linguistics*, 19(2):263–311, 1993. 22

[6] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995. 9

[7] Thomas M. Cover and Peter E. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1):21–27, 1967. 8

[8] Daniel Crevier. *AI: The Tumultuous History of the Search for Artificial Intelligence*. Basic Books, New York, 1993. 4, 5

[9] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018. 21, 42, 43

[10] Pedro Domingos and Michael Pazzani. On the optimality of the simple bayesian classifier under zero-one loss. *Machine Learning*, 29(2–3):103–130, 1997. 10

[11] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, et al. An image is worth 16x16 words: Transformers for image recognition at scale, 2020. 37, 39, 41, 42, 43, 44, 45, 46

[12] Richard O. Duda and Peter E. Hart. *Pattern Classification and Scene Analysis*. John Wiley & Sons, New York, 1973. 8, 9

[13] Jeffrey L. Elman. Finding structure in time. *Cognitive Science*, 14(2):179–211, 1990. 12, 22, 26

[14] Edward A. Feigenbaum and Pamela McCorduck. *The Fifth Generation: Artificial Intelligence and Japan's Computer Challenge to the World*. Addison-Wesley, Reading, MA, 1983. 4, 5

[15] Felix A. Gers, Jürgen Schmidhuber, and Fred Cummins. Learning to forget: Continual prediction with lstm. In *Proceedings of the Ninth International Conference on Artificial Neural Networks (ICANN-99)*, pages 850–855, Edinburgh, UK, 1999. 11, 12, 14, 16, 17

[16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, Cambridge, MA, 2016. 6, 12, 13, 14, 17, 19

[17] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016. 30, 31, 37, 44, 45, 52, 54

[18] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016. 52

[19] Sepp Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6(2):107–116, 1998. 13, 14, 17, 18

[20] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997. 11, 13, 14, 15, 17, 18, 26, 30

[21] John Hutchins. *Machine translation: past, present, future*. Ellis Horwood Chichester, 1986. 21

[22] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456, 2015. 31, 32, 33

[23] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An Introduction to Statistical Learning: with Applications in R*. Springer, New York, 2013. 7, 8

[24] Philipp Koehn. *Statistical Machine Translation*. Cambridge University Press, 2010. 22

[25] Philipp Koehn, Franz Josef Och, and Daniel Marcu. Statistical phrase-based translation. In *Proc. of NAACL/HLT*, pages 48–54, 2003. 22

[26] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012. 37

[27] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. 37

[28] James Lighthill. *Artificial Intelligence: A General Survey*. HM Stationery Office, London, 1973. Science Research Council Report. 4

[29] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted

windows. In *Proc. of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 10012–10022, 2021. 44, 46, 49, 50, 51, 52, 53, 54

[30] Minh-Thang Luong, Hieu Pham, and Christopher D Manning. Effective approaches to attention-based neural machine translation. In *Proc. of EMNLP*, pages 1412–1421, 2015. 22

[31] Andrew McCallum and Kamal Nigam. A comparison of event models for naive bayes text classification. In *AAAI-98 Workshop on Learning for Text Categorization*, pages 41–48, Madison, WI, 1998. 10

[32] John McCarthy, Marvin Minsky, Nathaniel Rochester, and Claude E. Shannon. A proposal for the dartmouth summer research project on artificial intelligence. *Dartmouth College AI Archive*, 1956. 3

[33] Pamela McCorduck. *Machines Who Think: A Personal Inquiry into the History and Prospects of Artificial Intelligence*. A. K. Peters, Ltd., Natick, MA, 2nd edition, 2004. 5

[34] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013. 24

[35] Tom M. Mitchell. *Machine Learning*. McGraw-Hill, New York, 1997. 6, 8, 9, 10

[36] Douglas C. Montgomery, Elizabeth A. Peck, and Geoffrey G. Vining. *Introduction to Linear Regression Analysis*. John Wiley & Sons, Hoboken, NJ, 6th edition, 2021. 7

[37] Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective*. MIT Press, Cambridge, MA, 2012. 6, 7, 9, 10

[38] Makoto Nagao. A framework of a mechanical translation between japanese and english by analogy principle. In *Proc. of the international NATO symposium on artificial and human intelligence*, pages 173–180, 1984. 21

[39] Allen Newell, J. Clifford Shaw, and Herbert A. Simon. Report on a general problem-solving program. In *Proceedings of the International Conference on Information Processing*, pages 256–264, 1959. 4

[40] Nils J. Nilsson. *The Quest for Artificial Intelligence: A History of Ideas and Achievements*. Cambridge University Press, Cambridge, 2010. 4, 6

[41] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *Proc. of EMNLP*, pages 1532–1543, 2014. 24

[42] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. OpenAI report, 2018. 21

[43] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986. 11, 13, 18

[44] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson, London, 3rd edition, 2016. 5, 6

[45] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014. 22, 34, 35

[46] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 2nd edition, 2018. 8

[47] Vladimir Vapnik. *Statistical Learning Theory*. Wiley, New York, 1998. 9

[48] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008, 2017. 17, 20, 21, 22, 25, 26, 27, 30, 31, 33, 34, 36, 37, 41, 42, 43, 44

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.

Vision Trnasformer

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