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

نابستان ۱۴۰۴

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 پایان‌نامه یار و راهنمای من بودند، نهایت سپاس و قدردانی را دارم.

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

سید محمد بادزهره

تابستان ۱۴۰۴

# چکیده

ببعلعذدقفلهعقفدلخقفدللقفلقفاقا

# فهرست مطالب

# فهرست جداول

# فهرست تصاویر

# پیش‌گفتار

قدثثمقدکنقصدبثقلدقفخدلقخفادخفادخ

# فصل ۱

# مقدمه

در سال‌های اخیر، توسعه سریع فناوری‌های مبتنی بر داده و نیاز روزافزون به تحلیل هوشمند اطلاعات، موجب رشد چشم‌گیر الگوریتم‌های یادگیری ماشین و یادگیری عمیق شده است. در این میان، مدل‌های مبتنی بر معماری‌های شبکه‌های عصبی، به‌ویژه در حوزه‌هایی چون پردازش زبان طبیعی، بینایی ماشین، و تحلیل سری‌های زمانی، جایگاه ویژه‌ای یافته‌اند. یکی از تحولات بنیادی در این مسیر، ظهور معماری مبدل ها [۱] بوده است که با بهره‌گیری از مکانیزم توجه، رویکردی نوین و مؤثر برای مدل‌سازی وابستگی‌های طولانی در داده‌های ترتیبی ارائه می‌دهد. درک جایگاه و کارکرد این معماری، مستلزم شناخت دقیق‌تر از مفاهیم بنیادین در هوش مصنوعی، یادگیری ماشین، شبکه‌های عصبی و ساختارهای بازگشتی است. از این‌رو، فصل حاضر به‌منظور ارائه بستر نظری لازم، به بررسی سیر تاریخی هوش مصنوعی، معرفی روش‌های یادگیری ماشین، مروری بر الگوریتم‌های کلاسیک، و تحلیل ساختار شبکه‌های بازگشتی از جمله شبکه های حافظه کوتاه مدت طولانی [۲] اختصاص دارد.

[۱] Transformer

[۲] LSTM

### ۱.۰.۱ 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 [۳] شروع به کار کردند [۸، ۴۰].

### ۲.۰.۱ دورهٔ طلایی و پیشرفت‌های اولیه

در دههٔ ۵۰ و ۶۰ میلادی، هوش مصنوعی[۴] 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[۵] که در این دوره به وجود آمدند و زمینه‌ساز اولین دستاوردهای هوش مصنوعی در بازی‌های تخته‌ای[۶] همچون شطرنج[۷] شدند [۳۹].

در این دوران، پیشرفت‌های بیشتری در پردازش زبان طبیعی[۸] و سیستم‌های خبره[۹] نیز صورت گرفت که این امید را در دانشمندان و محققان تقویت کرد که دستیابی به هوش مصنوعی عمومی[۱۰] به‌زودی ممکن خواهد بود [۱۴].

---

[۳] AGI, Artificial General Intelligence
[۴] Artificial Intelligence (AI)
[۵] Tree Search Algorithms
[۶] Board Games
[۷] Chess
[۸] Natural Language Processing (NLP)
[۹] Expert Systems
[۱۰] Artificial General Intelligence (AGI)

### ۳.۰.۱ انتظارات بیش از حد و ظهور عصر تاریک

با وجود پیشرفت‌های هوش مصنوعی، محدودیت‌های تکنولوژی (مثل عدم وجود پردازنده های گرافیکی [۱۱] پرقدرت در آن زمان) و همچنین کمبود داده‌های کافی برای آموزش مدل‌های پیچیده‌تر، باعث شد که بسیاری از پروژه‌های تحقیقاتی نتوانند به نتایج پیش‌بینی‌شده قابل قبول دست یابند. در نتیجه، هوش مصنوعی در دهۀ ۷۰ به مرحله‌ای از رکود وارد شد که به آن عصر تاریک هوش مصنوعی یا زمستان هوش مصنوعی [۱۲] 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 ادامه سرمایه‌گذاری منصرف شدند.

### ۴.۰.۱ 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 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].

[۱۱] GPU
[۱۲] AI Winter

### ۵.۰.۱ 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 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 [۱۳] 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 توسعه 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].

---

[۱۳] Rule Based Systems

## ۱.۱ انواع مدل یادگیری ماشین و شبکه‌های عصبی

یادگیری ماشین و شبکه‌های عصبی به عنوان دو زیرشاخه مهم از هوش مصنوعی، در سال‌های اخیر به‌طور گسترده‌ای مورد توجه پژوهشگران و صنعت قرار گرفته‌اند. این مدل‌ها با هدف یادگیری الگوها و روابط موجود در داده‌ها توسعه یافته‌اند و امروزه در حوزه‌های مختلفی از جمله بینایی ماشین، پردازش زبان طبیعی، پیش‌بینی سری‌های زمانی و داده‌کاوی مورد استفاده قرار می‌گیرند [۴، ۳۵، ۳۷].

### ۱.۱.۱ یادگیری ماشین: مروری کلی

یادگیری ماشین [۱۴] 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.

### ۲.۱.۱ تقسیم‌بندی‌های اصلی در یادگیری ماشین

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

- یادگیری با نظارت[۱۵]
- یادگیری بدون نظارت[۱۶]
- یادگیری تقویتی[۱۷]

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

[۱۴] Machine Learning
[۱۵] Supervised Learning
[۱۶] Unsupervised Learning
[۱۷] Reinforcement Learning

### ۳.۱.۱ یادگیری نظارت‌شده

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 الگوریتم رگرسیون [۱۸] و کلاس بندی[۱۹] می‌شود.

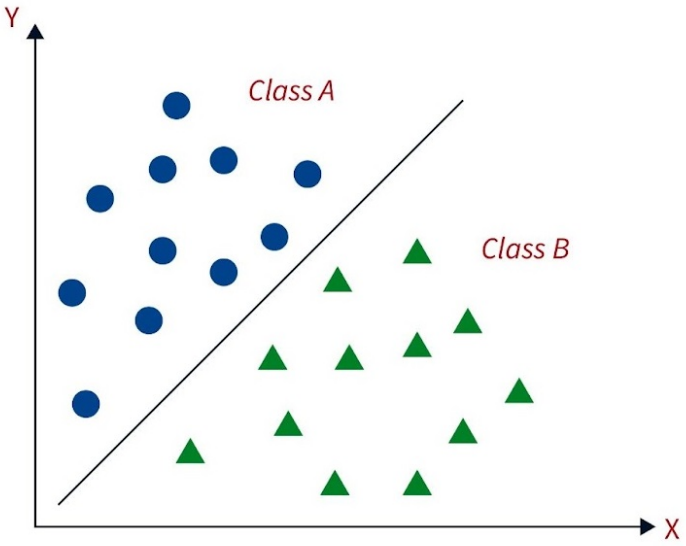
کلاس‌بندی

در میان روش‌های یادگیری با نظارت، کلاس‌بندی [۲۰] یکی از پرکاربردترین مسائل محسوب می‌شود. هدف در این مسئله، تخصیص هر نمونه ورودی به یکی از برچسب‌های گسسته از پیش تعریف‌شده است. مدل با استفاده از داده‌های آموزشی که شامل ویژگی‌ها و برچسب صحیح هستند، الگوهای حاکم بر داده را می‌آموزد و تلاش می‌کند بر اساس آن، داده‌های جدید را به دسته مناسب اختصاص دهد. این روش در کاربردهای متنوعی نظیر تشخیص هرزنامه[۲۱]، شناسایی بیماری‌ها، طبقه‌بندی تصاویر و تحلیل احساسات به‌طور گسترده مورد استفاده قرار می‌گیرد [۴، ۳۷].

رگرسیون

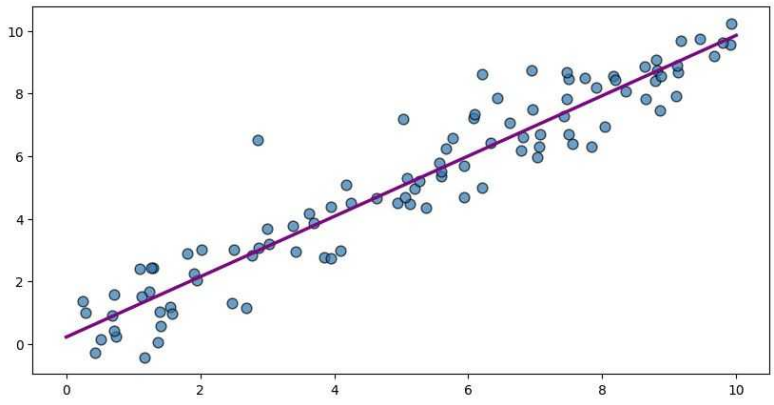
در مقابل، رگرسیون [۲۲] به مسئله پیش بینی مقادیر پیوسته می‌پردازد. در این نوع از یادگیری نظارت‌شده، مدل می‌کوشد رابطه‌ای میان متغیرهای ورودی و خروجی‌های عددی برقرار کند تا بر اساس آن، بتواند خروجی نمونه‌های جدید را تخمین بزند. تفاوت اصلی رگرسیون با کلاس‌بندی در ماهیت خروجی است؛ به‌طوری‌که در رگرسیون، خروجی به جای دسته‌بندی، یک مقدار عددی خواهد بود. از جمله کاربردهای رایج این نوع مدل‌ها می‌توان به پیش‌بینی قیمت مسکن، تخمین میزان فروش، و پیش‌بینی

[۱۸] Regression
[۱۹] Classification
[۲۰] Classification
[۲۱] Spam Detection
[۲۲] Regression




شکل ۱.۱.۱: کلاس بندی

وضعیت آب‌وهوا اشاره کرد [۳۶].




شکل ۱.۱.۲: رگرسیون

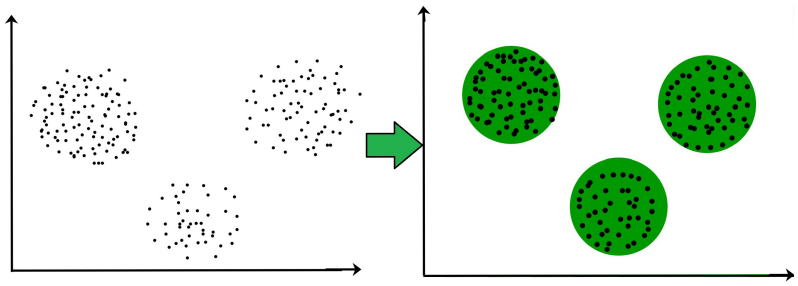
### ۴.۱.۱ یادگیری بدون نظارت

یادگیری بدون نظارت [۲۳] نوعی از روش‌های یادگیری ماشین است که در آن مدل بدون استفاده از برچسب‌های خروجی، سعی در کشف الگوها، ساختارها یا روابط پنهان میان داده‌ها دارد [۴]. در

[۲۳] Unsupervised Learning

این رویکرد، داده‌های ورودی تنها شامل ویژگی‌ها هستند و هدف، گروه‌بندی یا استخراج ساختار درونی آن‌ها بدون دانش قبلی از دسته‌بندی صحیح است. برخلاف یادگیری با نظارت که مدل از پاسخ صحیح برای آموزش استفاده می‌کند، در یادگیری بدون نظارت، مدل باید به‌تنهایی ساختارهای معنادار را در داده‌ها کشف کند.

از کاربردهای رایج یادگیری بدون نظارت می‌توان به خوشه‌بندی [۲۴]، کاهش ابعاد [۲۵]، آشکارسازی ناهنجاری‌ها [۲۶]، و استخراج ویژگی‌ها اشاره کرد.



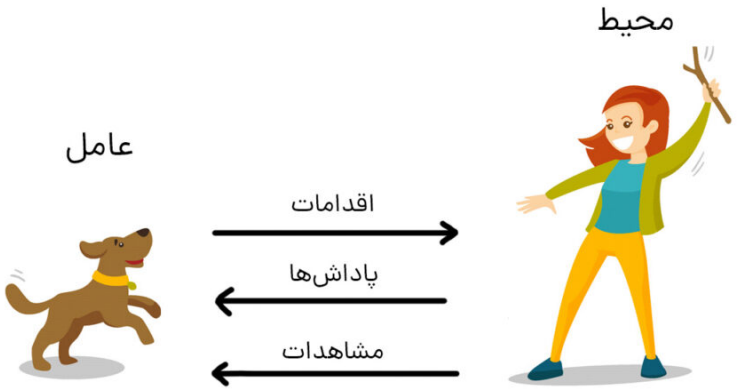
شکل ۱.۱.۳: خوشه بندی

### ۵.۱.۱ یادگیری تقویتی

یادگیری تقویتی، [۲۷] 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 نتیجه بهینه می‌شود. از کاربردهای یادگیری تقویتی می‌توان به بازی‌ها [۲۸]،

[۲۴] Clustering
[۲۵] Dimensionality Reduction
[۲۶] Anomaly Detection
[۲۷] reinforcement learning
[۲۸] Games

کنترل رباتیک [۲۹] و سیستم‌های توصیه‌گر [۳۰] اشاره کرد.




شکل ۱.۱.۴: یادگیری تقویتی

### ۱.۱.۶ معرفی چند مدل از الگوریتم یادگیری کلاسیک

الگوریتم‌های یادگیری کلاسیک، پایه و اساس بسیاری از پیشرفت‌های اولیه در یادگیری ماشین را شکل داده‌اند. این الگوریتم‌ها با وجود سادگی نسبی، در بسیاری از کاربردها همچنان عملکرد قابل قبولی از خود نشان می‌دهند و در بسیاری از سامانه‌های عملیاتی مورد استفاده قرار می‌گیرند.

الگوریتم‌هایی مانند نزدیک‌ترین همسایه، ماشین بردار پشتیبان، بیز ساده و درخت تصمیم از جمله مشهورترین روش‌های کلاسیک یادگیری هستند که هر یک بر اساس اصول ریاضی و آماری متفاوتی طراحی شده‌اند. این مدل‌ها معمولاً برای مسائل طبقه‌بندی یا رگرسیون به‌کار می‌روند و از مزایایی چون پیاده‌سازی آسان، قابلیت تفسیر بالا و نیاز کمتر به تنظیمات پیچیده برخوردارند.

در این بخش، به معرفی اجمالی چند مورد از این الگوریتم‌ها پرداخته می‌شود تا زمینهٔ درک عمیق‌تر روش‌های نوین‌تر یادگیری، مانند شبکه‌های عصبی و مدل‌های عمیق، فراهم گردد.

---

[۲۹] Robotic Control

[۳۰] Recommender Systems

### ۷.۱.۱ نزدیک‌ترین همسایه

الگوریتم نزدیک‌ترین همسایه [۳۱] 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 نمونه 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].

[۳۱] k-Nearest Neighbors

### ۸.۱.۱ ماشین بردار پشتیبان

الگوریتم ماشین بردار پشتیبان [۳۲] با یافتن یک ابرصفحه 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: ماشین بردار پشتیبان می‌تواند به خوبی با داده‌های چندبعدی و پیچیده کار کند [۴۷].
- مقاومت در برابر بیش‌برازش [۳۳]: با استفاده از هسته‌ها [۳۴]، داده‌های غیرخطی نیز به فضای بالاتر برده می‌شوند و جداسازی بهتری انجام می‌شود[۶].

معایب:

- پیچیدگی محاسباتی: آموزش ماشین بردار پشتیبان 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].

### ۹.۱.۱ بیز ساده

بیز ساده [۳۵] مبتنی بر قضیه بیز [۳۶]است و فرض می‌کند ویژگی‌ها به‌صورت شرطی مستقل از هم هستند[۱۰، ۳۵]. این مدل برای اولین بار در حوزه پردازش متن به کار رفت و هنوز هم در بسیاری

[۳۲] Support Vector Machine, SVM
[۳۳] Overfitting
[۳۴] kernels
[۳۵] Bayes Naive
[۳۶] Bayes' theorem

از کاربردها مانند طبقه‌بندی ایمیل و تحلیل احساسات مورد استفاده قرار می‌گیرد[۳۱]. در بیز ساده بر اساس احتمالات محاسبه می‌شود که یک نمونه جدید به کدام دسته تعلق دارد. این الگوریتم بر اساس قضیه 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، بیز ساده 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].

### ۱۰.۱.۱ شبکه‌های عصبی بازگشتی و شبکه‌های حافظه بلندمدت کوتاه‌مدت

شبکه‌های عصبی بازگشتی [۳۷] و مدل‌هایی با حافظهٔ بلندمدت_کوتاه‌مدت [۳۸] 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].

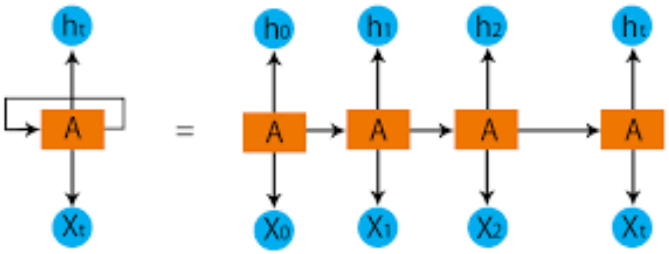
[۳۷] RNN
[۳۸] LSTM

### ۱۱.۱.۱ شبکه‌های عصبی بازگشتی

مدل‌های اولیهٔ شبکه‌های عصبی، مانند شبکه‌های چندلایه [۳۹]، 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، صدا و داده‌ها به توالی خاصی وابسته هستند. به همین دلیل، شبکه‌های عصبی بازگشتی معرفی شدند تا بتوانند از اطلاعات پیشین در پردازش داده‌های بعدی استفاده کنند[۴۳].

### ۱۲.۱.۱ ساختار و عملکرد شبکه‌های عصبی بازگشتی

شبکه‌های شبکه‌های عصبی بازگشتی [۴۰] 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].




شکل ۱.۱.۵: شبکه های عصبی بازگشتی

$$h_t = \sigma\big(W \cdot x_t + U \cdot h_{t-1} + b\big) \qquad (۱.۱.۱)$$

در اینجا:

- $h_t$ وضعیت مخفی یا حالت در گام زمانی $t$ است.

---
[۳۹] MLP
[۴۰] recurrent neural network

- $W$ وزن‌هایی است که به ورودی $x_t$ اعمال می‌شود.
- $U$ وزن‌های اعمال‌شده به وضعیت قبلی $h_{t-1}$ است.
- $b$ بایاس مدل است.
- $\sigma$ 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.

### ۱۳.۱.۱ مزایا و معایب شبکه‌های عصبی بازگشتی

در این قسمت به مزایا و معایب شبکه‌های شبکه‌های عصبی بازگشتی می‌پردازیم.

مزایا:

- حفظ وابستگی زمانی: شبکه‌های عصبی بازگشتی 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 [۴۱]: در فرایند آموزش با روش پس‌انتشار[۴۲] ، اگر توالی داده‌ها طولانی باشد، گرادیان‌ها ممکن است بسیار کوچک یا بزرگ شوند که منجر به ناپایداری در آموزش و کاهش دقت می‌شود [۱۹].

[۴۱] Vanishing and Exploding Gradient
[۴۲] back propagation

- محدودیت در پردازش توالی‌های بسیار بلند: شبکه‌های عصبی بازگشتی در حفظ اطلاعات طولانی‌مدت با مشکل مواجه است و برای پردازش وابستگی‌های طولانی، عملکرد ضعیفی دارد [۲۰، ۱۶].

### ۱۴.۱.۱ شبکه‌های حافظه بلندمدت_کوتاه‌مدت

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

### ۱۵.۱.۱ ناپدید شدن گرادیان در شبکه های بازگشتی

شبکه‌های عصبی بازگشتی برای پردازش داده‌های ترتیبی از حلقه‌های بازگشتی بهره می‌برند. در فرایند آموزش شبکه‌های عصبی بازگشتی، از الگوریتم پس‌انتشار خطا از طریق زمان [۴۳] استفاده می‌شود که گرادیان‌ها را جهت به‌روزرسانی وزن‌ها محاسبه می‌کند.[۴۳].

با این حال، شبکه های عصبی بازگشتی 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[۴۴] معروف است [۱۹].

[۴۳] Backpropagation Through Time
[۴۴] vanishing gradient

فرمول کلی گرادیان در زمان $t$ به‌صورت زیر است:

$$\frac{\partial L}{\partial W} = \prod_{k=1}^{t} \frac{\partial h_k}{\partial h_{k-1}} \cdot \frac{\partial h_t}{\partial L} \qquad (۱.۱.۲)$$

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].

### ۱۶.۱.۱ ظهور شبکه‌های حافظه بلندمدت_کوتاه‌مدت

در سال ۱۹۹۷ ،شبکه‌های حافظه بلندمدت_کوتاه‌مدت معرفی شد. [۲۰]. انگیزه اصلی توسعه این شبکه حل مشکل ناپدید شدن گرادیان در شبکه‌های عصبی بازگشتی بود. این مشکل در مسائل یادگیری داده‌های ترتیبی طولانی مانع می‌شد شبکه های عصبی بازگشتی وابستگی‌های بلندمدت را به‌درستی فراگیرد.

### ۱۷.۱.۱ راه‌حل شبکه‌های حافظه بلندمدت_کوتاه‌مدت برای پایداری جریان گرادیان‌ها

با معرفی شبکه حافظه بلند مدت، کوتاه مدت [۴۵] در شبکه‌های بازگشتی، جریان گرادیان‌ها را در طول توالی پایدار نگه می‌دارد. این کار از طریق اضافه کردن وضعیت سلولی [۴۶] و دروازه‌ها [۴۷] به ساختار شبکه‌های عصبی بازگشتی انجام می‌شود. [۱۵]. این اجزا به این شبکه این امکان را می‌دهند:

۱. اطلاعات غیرضروری را فراموش کند،

۲. اطلاعات مهم جدید را اضافه کند،

---

[۴۵] long short-term memory
[۴۶] Cell State
[۴۷] Gates

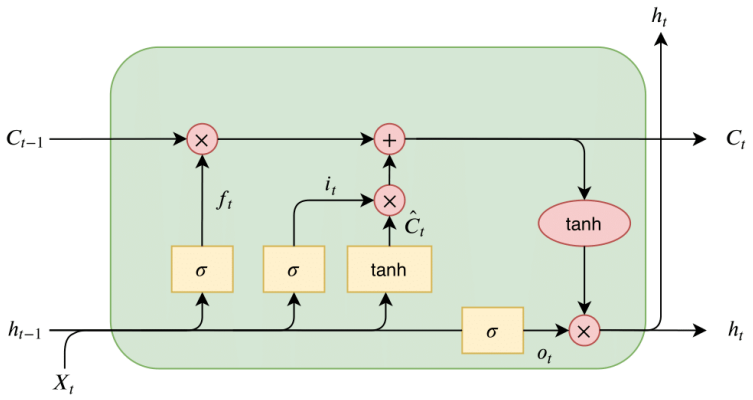
۳. اطلاعات مهم قبلی را حفظ کند.

## ۲.۱ ساختار شبکه های حافظه بلندـ مدت کوتاهـ مدت

شبکه های حافظه بلندـ مدت شامل اجزای جدیدی است که به آن امکان مدیریت بهتر اطلاعات را می‌دهد:

### ۱.۲.۱ وضعیت سلولی

مسیر اصلی ذخیره اطلاعات در شبکه های حافظه بلندـ مدت کوتاه مدت است که می‌تواند اطلاعات مهم را در طول توالی حفظ کند. برخلاف شبکه های عصبی بازگشتی که عمدتاً بر خروجی‌های بازگشتی $h_t$ متکی است، شبکه حافظه بلند مدت، کوتاه مدت یک مسیر جداگانه برای عبور اطلاعات از وضعیت سلولی دارد که به حفظ گرادیان‌ها کمک شایانی می‌کند [۲۰].




شکل ۱.۲.۶: LSTM

### ۲.۲.۱ دروازه‌ها

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

- دروازهٔ فراموشی [۴۸] تعیین می‌کند چه اطلاعاتی از وضعیت سلولی باید حذف شود [۱۵].

$$f_t = \sigma\big(W_f \cdot [h_{t-1}, x_t] + b_f\big) \tag{۱.۲.۳}$$

در این معادله، $f_t$ بیانگر میزان فراموشی برای هر مؤلفه از وضعیت سلولی در گام زمانی جاری است. این مقدار با استفاده از تابع فعال‌ساز سیگموید $\sigma$ محاسبه می‌شود که خروجی آن بین صفر تا یک قرار دارد. هرچه مقدار $f_t$ به صفر نزدیک‌تر باشد، آن مؤلفه از وضعیت سلولی بیشتر فراموش می‌شود؛ و بالعکس، مقدار نزدیک به یک نشان‌دهنده حفظ کامل آن اطلاعات در حافظه سلولی است.

- دروازهٔ ورودی [۴۹]: تعیین می‌کند چه اطلاعات جدیدی باید به وضعیت سلولی اضافه شود:

$$i_t = \sigma\big(W_i \cdot [h_{t-1}, x_t] + b_i\big) \tag{۱.۲.۴}$$

$$\tilde{C}_t = \tanh\big(W_C \cdot [h_{t-1}, x_t] + b_C\big) \tag{۱.۲.۵}$$

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

- دروازهٔ خروجی [۵۰]:

تعیین می‌کند چه اطلاعاتی از وضعیت سلولی به خروجی منتقل شود:

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

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

---

[۴۸] Forget Gate
[۴۹] Input Gate
[۵۰] Output Gate

### ۳.۲.۱ به‌روزرسانی وضعیت سلولی

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

$$C_t = f_t \cdot C_{t-1} + i_t \cdot \tilde{C}_t \qquad (۱.۲.۶)$$

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

علت پایداری گرادیان در شبکه های حافظه بلندــ مدت کوتاه مدت

- حذف ضرب‌های مکرر: برخلاف شبکه های بازگشتی که به ضرب‌های مکرر وزن‌ها و گرادیان‌ها وابسته است، شبکه های حافظه بلندــ مدت کوتاه مدت 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 ۱.۲.۱: مقایسه ویژگی‌های RNN و LSTM

### ۴.۲.۱ مشکلات کلی شبکه های بازگشتی و ظهور مبدل ها

شبکه‌های بازگشتی که به آن ها پرداخته شد 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 مبدل‌ها [۵۱] منجر شد[۴۸].

### ۵.۲.۱ مشکل وابستگی ترتیبی در شبکه های بازگشتی

شبکه های بازگشتی 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 شبکه های حافظه بلندــ مدت کوتاه مدت در یادگیری وابستگی‌های بلندمدت نسبت به شبکه های بازگشتی 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 شبکه های حافظه بلندــمدت کوتاه مدت نیز ظرفیت حفظ اطلاعات محدود است و با افزایش طول توالی، دقت مدل افت می‌کند.

[۵۱] Transformers

- تأثیر تدریجی داده‌های اولیه: داده‌های ابتدایی توالی ممکن است با گذشت زمان اهمیت خود را از دست بدهند، چراکه گرادیان‌ها به‌تدریج ضعیف‌تر می‌شوند.

### ۶.۲.۱ پیچیدگی محاسباتی و حافظه در شبکه های بلند مدت کوتاه مدت

شبکه های حافظه بلند_مدت کوتاه مدت 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 ذخیره وضعیت سلولی و گرادیان‌ها، شبکه های حافظه بلند_مدت کوتاه مدت به حافظه بیشتری نسبت به مدل‌های ساده‌تر احتیاج دارند.
- هزینهٔ محاسباتی بالا: در داده‌های حجیم، انجام محاسبات سنگین می‌تواند اجرای مدل را بسیار کند سازد.

### ۷.۲.۱ مشکل پردازش وابستگی‌های غیرمتوالی

شبکه های بازگشتی به‌طور طبیعی برای یادگیری وابستگی‌های محلی و متوالی مناسب هستند. اما در مسائلی مانند ترجمه 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. شبکه های بازگشتی برای یادگیری این‌گونه وابستگی‌ها محدودیت دارند.

### ۸.۲.۱ گرادیان‌های ناپایدار و مشکلات بهینه‌سازی

با وجود بهبودهایی که شبکه حافظه بلندمدت، کوتاه مدت نسبت به شبکه های بازگشتی معمولی در پایداری گرادیان ارائه داد، هنوز هم مشکلاتی در این شبکه ها وجود دارد که شامل موارد زیر است:

- 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 شبکه های بازگشتی دشوار است.

- نیاز به مدلی با ظرفیت بیشتر و سرعت بالاتر: برای مسائل پیچیده‌تر، به مدل‌هایی با تعداد پارامتر بالاتر نیاز است؛ اما این شبکه های بازگشتی به‌دلیل محدودیت در حافظه و پردازش، پاسخ‌گوی این نیاز نیستند.

- کارایی در داده‌های چندوجهی [۵۲]:برای داده‌هایی که ترکیبی از اطلاعات متنی، صوتی و تصویری هستند،شبکه های بازگشتی توانایی لازم جهت پردازش موازی این اطلاعات را ندارند.

در مجموع، وابستگی ترتیبی در شبکه های بازگشتی مانعی اساسی برای استفاده از این مدل‌ها در مسائل پیچیده و بزرگ بود که درنهایت به ظهور مبدل ها منتهی شد [۴۸]. مبدل‌ها با طراحی مبتنی بر موازی‌سازی و مکانیزم توجه [۵۳]، این محدودیت را برطرف کرده و راه‌حلی کارآمدتر برای پردازش داده‌های ترتیبی ارائه دادند.

---

[۵۲] Multimodal

[۵۳] Attention Mechanism

# فصل ۲

# پیشینه پژوهش

با توجه به مشکلات مطرح شده شبکه های بازگشتی، معماری جدیدی به نام مبدل ها [۱] مطرح شد ظهور مبدل‌ها یکی از تحولات اساسی در حوزه پردازش زبان طبیعی [۲] و یادگیری ماشین به شمار می‌رود.[۴۸، ۲]. این مدل‌ها باعث تغییرات عمده‌ای در نحوه ساخت و آموزش مدل‌های زبانی و همچنین در بسیاری از کاربردهای دیگر یادگیری ماشین شده‌اند و توانستند بسیاری از مشکلات مدل‌های قبلی را حل کنند[۹، ۴۲].

## ۱.۲ مشکلات ترجمه ماشینی و مبدل ها

در ابتدا، ترجمه ماشینی [۳] یک چالش اساسی در زمینهٔ پردازش زبان طبیعی بود. مدل‌های اولیه‌ای مانند مدل‌های مبتنی بر قواعد [۴] 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

[۱] Transformer
[۲] NLP
[۳] machine translation
[۴] Rule-based Models

محدودیت‌های زیادی داشتند و نمی‌توانستند ویژگی‌های پیچیده‌تر زبان را مدل‌سازی کنند.

سپس مدل‌های آماری [۵] معرفی شدند [۵، ۲۴]. این مدل‌ها از داده‌های ترجمه‌شده برای آموزش مدل‌های آماری استفاده می‌کردند که احتمال ترجمه صحیح را براساس شواهد آماری محاسبه می‌کردند. مدل‌هایی مانند مدل‌های ترجمه آماری مبتنی بر جمله [۶] [۲۵] از این نوع بودند که قادر به ترجمه جملات بهتر از مدل‌های مبتنی بر قواعد بودند، اما هنوز هم در ترجمه‌های پیچیده با مشکلاتی روبه‌رو بودند.

بعد از این مدل‌ها، مدل‌های بازگشتی [۷] به وجود آمدند که مشکلات آن‌ها در فصل گذشته بیان شد [۱۳، ۴۵]. در نهایت، این مشکلات باعث به وجود آمدن ترانسفورمرها شد [۲].

## ۲.۲ ظهور ترانسفورمرها

در سال ۲۰۱۷، مقاله‌ای توسط گوگل [۸] منتشر شد که مفهوم جدیدی به نام مبدل ها [۹] را معرفی کرد [۴۸]. این مقاله به موضوع ترجمه ماشینی پرداخت و نشان داد که با استفاده از مکانیزم توجه 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 مبدل ها در پردازش تصویر و متن بسیار سریع‌تر و دقیق‌تر از مدل‌های قبلی عمل کنند [۴۸].

---

[۵] Statistical Models
[۶] Phrase-based Statistical Models
[۷] Recurrent Models
[۸] google
[۹] Transformers

## ۳.۲ معماری مبدل ها

در تصویر ۲.۳.۱، معماری مبدل نمایش داده شده است و بخش‌ها و اجزای مختلف آن مشخص شده است. معماری ترانسفورمر از دو بخش اصلی تشکیل شده است:

- رمزگذار [۱۰]: وظیفهٔ رمزگذار این است که دادهٔ ورودی را دریافت کند و ویژگی‌های آن را استخراج کند.
- رمزگشا [۱۱]: وظیفهٔ رمزگشا این است که ویژگی‌های استخراج‌شده را به زبان مقصد تبدیل کند.

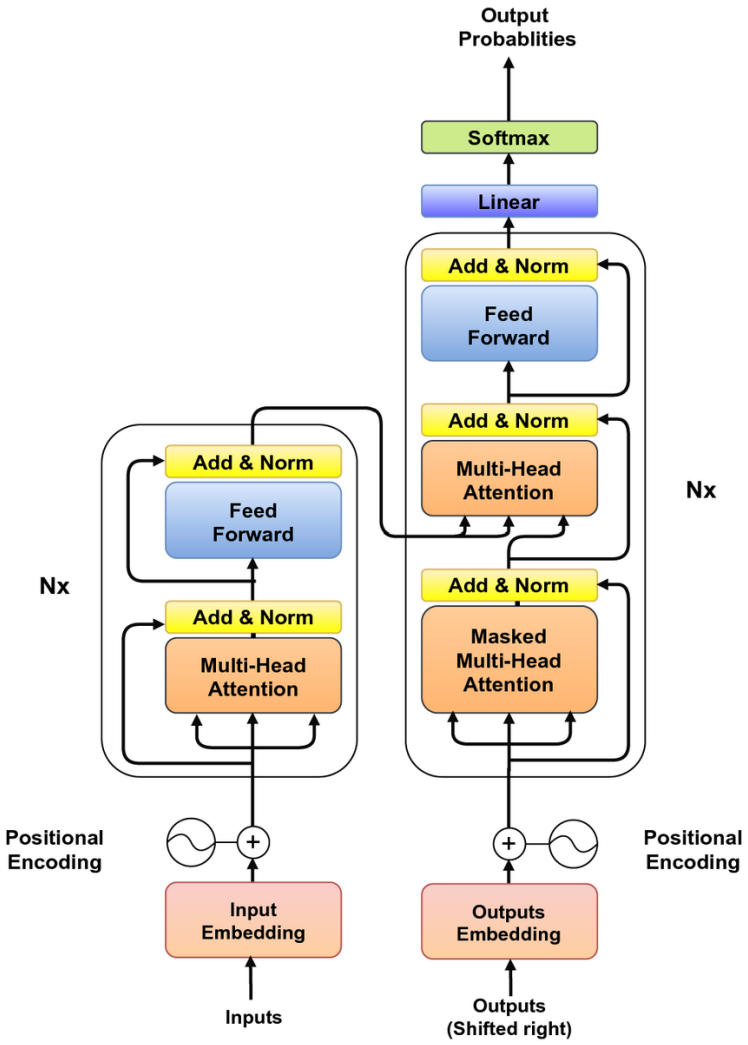
### ۱.۳.۲ جاسازی

در زبان طبیعی، کلمات به شکل رشته‌های متنی هستند مانند کتاب، ماشین 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 ۲.۳.۲ 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 تعبیه کردن [۱۲]، هر کلمه به برداری از اعداد تبدیل می‌شود. این جاسازی‌ها شباهت‌های معنایی بین کلمات

[۱۰] Encoder
[۱۱] Decoder
[۱۲] Embedding

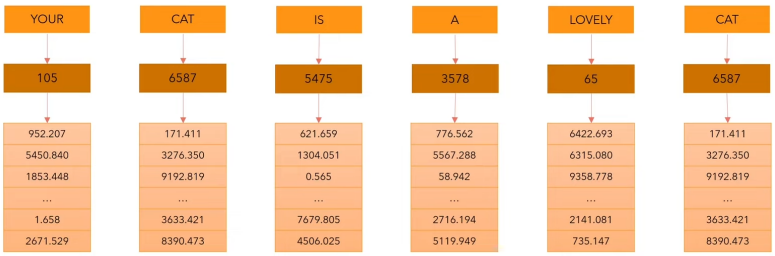



شکل ۲.۳.۱: معماری ترانسفورمرها

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].

### ۲.۳.۲ جاسازی موقعیتی

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




شکل ۲.۳.۲:

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 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 جاسازی موقعیتی [۱۳] به مدل وجود دارد. در ترانسفورمرها از روش جاسازی موقعیت سینوسی [۱۴] استفاده می‌شود [۴۸].

این روش قابل یادگیری نیست و صرفاً از یک سری فرمول‌های ساده برای جاسازی موقعیتی استفاده می‌کند. برای موقعیت مشخص [۱۵] در توالی و بُعد $i$ در فضای برداری، تعبیه موقعیتی به‌صورت زیر تعریف می‌شود.

و برای مقادیر زوج:

$$PE(pos, 2i) = \sin\left(\frac{pos}{10000^{\frac{2i}{d}}}\right) \qquad (۲.۳.۱)$$

و برای مقادیر فرد:

[۱۳] Positional Embedding
[۱۴] Sinusoidal Positional Embedding
[۱۵] pos

$$PE(pos, 2i+1) = \cos\left(\frac{pos}{10000^{\frac{2i}{d}}}\right) \qquad (۲.۳.۲)$$

- $pos$: موقعیت کلمه در توالی است (مثلاً از 0 تا $N-1$ برای یک توالی $N$ـتایی).
- $i$: 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 ۲.۳.۳ مشاهده می‌کنید، بعد از جاسازی کلمات، به آن جا سازی موقعیتی 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 ۲.۳.۳:

### ۳.۳.۲ توجه

در روش شبکه های بازگشتی، 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[۱۶] 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.

سه تا از اجزای اصلی یک توجه شامل موارد زیر است.

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

در ضرب شباهت های توجه [۱۷] [۴۸]، ابتدا شباهت یا ارتباط بین پرسش [۱۸] و کلید [۱۹] را با محاسبهٔ ضرب داخلی [۲۰] به‌دست می‌آوریم، سپس آن را نرمال می‌کنیم (با تقسیم بر $d_k$) و از تابع سافت مکس [۲۱] استفاده می‌کنیم تا ضرایب توجه [۲۲] را به‌دست آوریم. در نهایت با همین ضرایب، ترکیبی خطی از بردارهای مقدار [۲۳] را می‌گیریم.

فرمول به‌شکل زیر است:

---

[۱۶] attention
[۱۷] Scaled Dot-Product Attention
[۱۸] Query
[۱۹] Key
[۲۰] Dot Product
[۲۱] softmax
[۲۲] Attention Weights
[۲۳] value

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \qquad (۲.۳.۳)$$

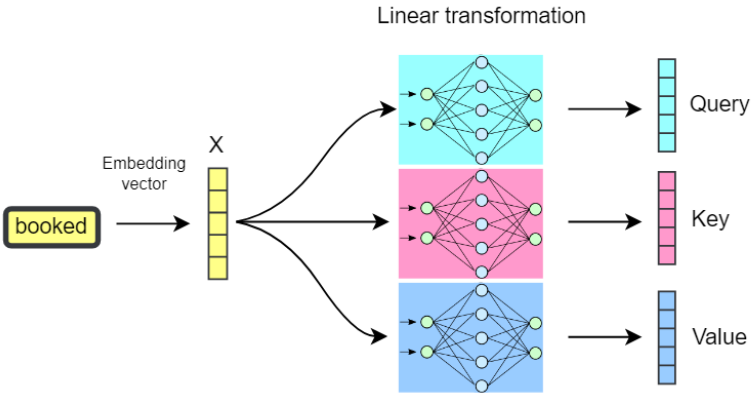
که در آن:

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

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

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

$W^Q, W^K, W^V \in \mathbb{R}^{d_{model} \times d_k}$ ماتریس‌های وزنی قابل آموزش هستند که طی فرآیند یادگیری به‌روزرسانی می‌شوند.




شکل ۲.۳.۴:

در واقع پرسش، کلید و مقدار با استفاده از mlp تولید میشود.

تقسیم بر $d_k$ باعث می‌شود مقدار ضرب داخلی در ابعاد بالا خیلی بزرگ نشود و شیب‌ها گرادیان پایدار بمانند.

$$\alpha = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) \qquad (۲.۳.۴)$$

$\alpha$ یک ماتریس با ابعاد $n \times n$ است که سطر $i$ـ‌ام آن ضرایب توجه برای توکن $i$ را نشان می‌دهد.

تفسیر ضرایب توجه: هر سطر از $\alpha$ 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 ۲.۳.۵: توجه

### ۴.۳.۲ توجه چند سر

در ایده چند سری [۲۴] 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 «سر» [۲۵] نام دارد) و به‌صورت موازی محاسبات توجه را انجام می‌دهیم. سپس خروجی همهٔ این سرها را کنار هم قرار داده [۲۶] و در نهایت با یک ماتریس وزن دیگر ضرب می‌کنیم تا به بعد اصلی بازگردیم.

فرمول مربوط به این ایده به‌شکل زیر است:

---

[۲۴] multi head attention

[۲۵] head

[۲۶] concatenate

$$\text{head}_i = \text{Attention}(Q_i, K_i, V_i) \qquad (۲.۳.۵)$$

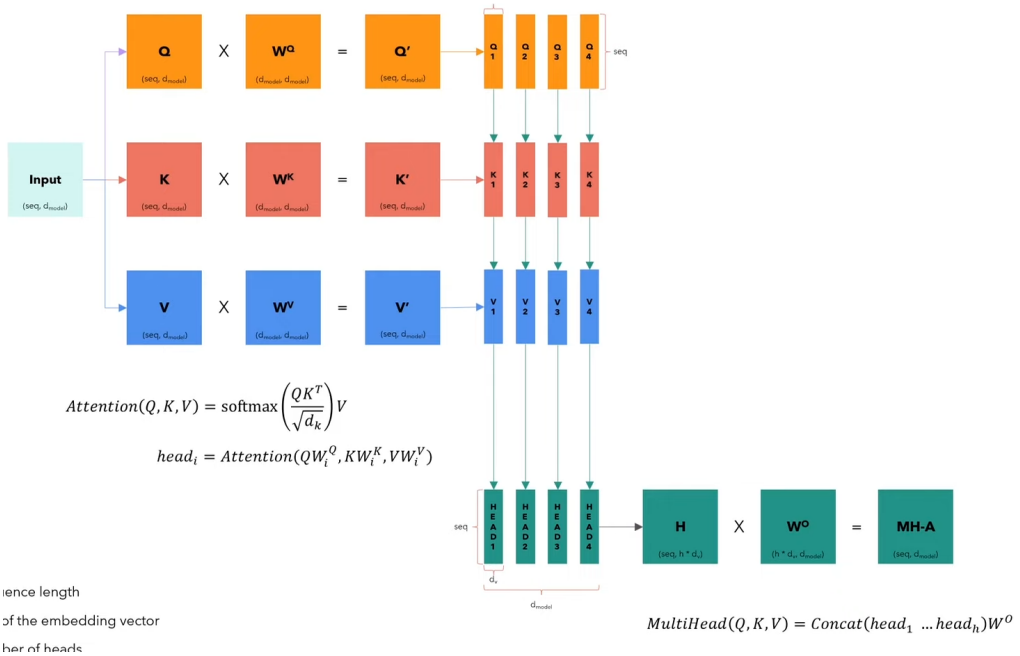
$$\text{MultiHead}(Q, K, V) = [\text{head}_1 \oplus \cdots \oplus \text{head}_h]W_O \qquad (۲.۳.۶)$$

که در آن $\oplus$ نشان‌دهندهٔ عمل الحاق [۲۷] 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 $d_{\text{model}}$ برمی‌گرداند.




شکل ۲.۳.۶: توجه چند سر

[۲۷] concatenate

### چرا چندین سر؟

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

افزایش ظرفیت مدل: با داشتن چند سر ، مدل می‌تواند قدرت بیان بیشتری داشته باشد.

ابعاد کمتر در هر سر: در عمل، اگر $d_{\text{model}}$ مثلاً ۵۱۲ باشد، و تعداد سر ها $h = 8$، آنگاه هر سر ابعادی در حدود $d_k = 64$ خواهد داشت؛ و این محاسبات ضرب داخلی را نیز مقیاس پذیر و قابل موازی‌سازی می‌کند.

## ۵.۳.۲ اتصال باقی مانده

در معماری‌های عمیق، هنگامی که تعداد لایه‌ها زیاد می‌شود، اغلب دچار ناپایداری گرادیان می‌شوند و این مشکل باعث دشواری در آموزش مدل می‌گردد [۲۰، ۳].

در مبدل ها [۴۸]، به جای این که خروجی توجه 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 [۲۸] 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:

$$x + \text{SubLayer}(x) \tag{۲.۳.۷}$$

این جمع به‌صورت عنصر به‌عنصر [۲۹] انجام می‌شود.

[۲۸] Residual Connection
[۲۹] Element-wise Addition

اتصال باقیمانده در مبدل ها چندین مزیت دارد که عبارت اند از:

### 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 ناپدید شدن گرادیان [۳۰] رخ دهد [۲۰، ۳].

### حفظ اطلاعات اصلی (هویت ورودی)

حتی اگر زیرماژول تغییری در اطلاعات ورودی ایجاد کند، با وجود اتصال باقیمانده [۳۱]، 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. اتصال باقی مانده تضمین می‌کند که اگر لایه‌ای به هر دلیل نتوانست الگوی بهینه را یاد بگیرد، اطلاعات قبلی حداقل به‌صورت دست‌نخورده تا حدی منتقل می‌شود.

## ۴.۲ نرمال سازی لایه ها

در یادگیری عمیق، نرمال‌سازی [۳۲] داده‌های یک لایه یا فعال‌سازی‌ها، اغلب به سرعت بخشیدن به همگرایی و پایدار کردن آموزش کمک شایانی می‌کند. شاید معروف‌ترین نوع نرمال‌سازی، نرمال

[۳۰] gradient vanishing
[۳۱] Residual Connection
[۳۲] Normalization

سازی بچ [۳۳] باشد که پیش‌تر در کارهای بینایی کانولوشنی[۳۴] بسیار مورداستفاده قرار گرفت [۲۲].

نرمال سازی لایه ها [۳۵] روشی جایگزین است که در ترانسفورمر استفاده می‌شود [۱، ۴۸]. علت اصلی این انتخاب، ماهیت توالی‌محور [۳۶] بودن داده‌ها در پردازش زبان طبیعی و عدم تمایل به وابستگی به آمار مینی‌بچ [۳۷] است.

## نرمال سازی بچ

در نرمال سازی بچ ها، برای نرمال‌سازی، میانگین و واریانس روی تمام نمونه‌های موجود در مینی‌بچ (و نیز در طول ابعاد ویژگی) محاسبه می‌شود [۲۲]. این موضوع در پردازش زبان طبیعی کمی دردسرساز است؛ چون ترتیب توکن‌ها، طول جمله‌ها و حتی اندازهٔ مینی‌بچ ممکن است نامنظم باشد. همچنین به خاطر تنوع طول توالی‌ها، پیاده‌سازی نرمال سازی بچ ها می‌تواند پیچیده شود.

## نرمال سازی لایه ها

در نرمال سازی لایه ها، برای هر توکن به‌صورت جداگانه (در طول بُعد ویژگی)، میانگین [۳۸] و واریانس [۳۹] 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 $d_{\text{model}}$ است. ما میانگین $\mu_i$ و واریانس $\sigma_i^2$ را از اجزای این بردار محاسبه می‌کنیم:

$$\mu_i = \frac{1}{d_{\text{model}}} \sum_{k=1}^{d_{\text{model}}} h_{i,k}, \quad \sigma_i^2 = \frac{1}{d_{\text{model}}} \sum_{k=1}^{d_{\text{model}}} (h_{i,k} - \mu_i)^2 \qquad (۲.۴.۸)$$

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

---

[۳۳] Batch Normalization
[۳۴] CNN
[۳۵] Layer Normalization
[۳۶] sequence
[۳۷] mini-batch
[۳۸] mean
[۳۹] variance

$$\hat{h}_{i,k} = \frac{h_{i,k} - \mu_i}{\sqrt{\sigma_i^2 + \epsilon}} \tag{۲.۴.۹}$$

در نهایت، برای این‌که مدل بتواند مقیاس و بایاس جدیدی یاد بگیرد، شبیه بچ نرم ، دو پارامتر $\gamma$ و $\beta$ نیز در طول بعد ویژگی اعمال می‌شوند:

$$\text{LayerNorm}(h_i) = \gamma \odot \hat{h}_i + \beta \tag{۲.۴.۱۰}$$

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

### مزایای نرمال سازی لایه در مبدل ها

- بی‌نیازی از وابستگی به ابعاد مینی‌بچ: با نرمال سازی لایه، 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.[۴۰] نرمال سازی لایه با نرمال‌سازی این توزیع، آموزش را پایدارتر و سریع‌تر می‌کند [۱، ۲۲].
- 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 مبدل ها، پس از خروجی هر زیرماژول، مراحل به‌شکل زیر است:

## ۱.۴.۲ اتصال باقی مانده

ابتدا ورودی همان زیرماژول (مثلاً بردار $x$) را با خروجی زیرماژول ($\text{SubLayer}(x)$) جمع می‌کنیم. حاصل این جمع را می‌توان چنین نوشت:

[۴۰] Internal Covariate Shift

$$z = x + \text{SubLayer}(x) \tag{۲.۴.۱۱}$$

این $z$ حالا ترکیبی از اطلاعات اصلی ورودی و اطلاعات یادگرفته‌شده توسط SubLayer است.

نرمال سازی لایه  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:

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

## ۵.۲ رمزگشا

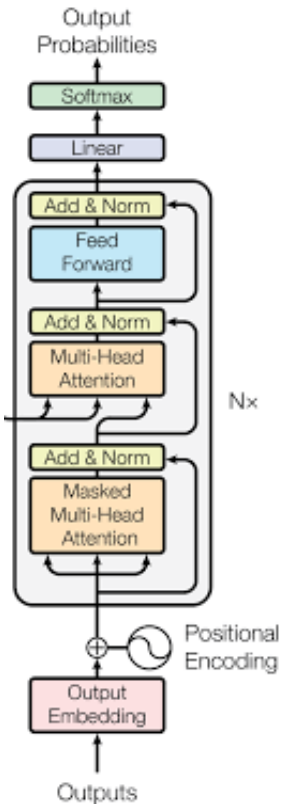
رمزگشا در معماری ترانسفورمرها وظیفهٔ تولید خروجی نهایی را بر عهده دارد. این خروجی معمولاً می‌تواند توالی هدف [۴۱] باشد، مانند ترجمه یک جمله یا پیش‌بینی توکن‌های بعدی در یک توالی [۴۸].

در این بخش، رمزگشا دو ورودی اصلی دارد: ۱. توالی هدف که معمولاً به‌صورت خودکار تولید می‌شود (مثلاً در ترجمهٔ ماشینی یا تولید متن)، ۲. نمایش اطلاعات کدشده که توسط انکودر تولید شده است و شامل ویژگی‌های استخراج‌شده از توالی ورودی می‌باشد.

رمزگشا از این ورودی‌ها استفاده می‌کند تا به‌صورت گام‌به‌گام، خروجی نهایی خود را تولید کند [۲، ۴۵].

همان‌طور که در شکل ۲.۵.۷ مشاهده می‌کنید، رمز گشا دو ورودی دارد.

[۴۱] Target Sequence




شکل ۲.۵.۷:

تمامی بخش‌های رمزگشا مانند رمزگذار هستند اما در دیکودر توجه چند سر ماسک شده [۴۲] وجود دارد [۴۸].

## ۶.۲ توجه چند سری ماسک شده

در مبدل‌ها، مکانیزم توجه چند سری [۴۳] در بخش دیکودر به‌صورت ماسک شده [۴۴] پیاده‌سازی می‌شود تا مدل نتواند توکن‌های آینده را ببیند و به‌صورت خودبازگشتی [۴۵] توکن بعدی را پیش‌بینی کند [۴۸].

در واقع ایده اصلی استفاده از ماسک جلوگیری از مشاهده آینده است.

در معماری‌های خودبازگشتی، مدل در گام $i$ از دیکودر تنها باید به توکن‌های قبلی $\{y_1, \ldots, y_{i-1}\}$

[۴۲] Masked Multi-Head Attention
[۴۳] Multi-Head Attention
[۴۴] Masked
[۴۵] Autoregressive

دسترسی داشته باشد؛ اما نه به توکن‌های $\{y_{i+1}, y_{i+2}, \dots\}$. اگر مدل بتواند توکن‌های آینده را «نگاه» کند، پیش‌بینی توکن بعدی آسان و غیرواقعی می‌شود (مشکل نشت اطلاعات) [۲، ۴۵].

به همین دلیل در توجه چند سری ماسک شده در دیکودر، از یک ماتریس ماسک $M$ استفاده می‌کنیم که اجازه نمی‌دهد هر توکن به توکن‌های آینده‌اش توجه کند.

## ۷.۲ مثال عددی توجه ماسک شده

فرض کنید دنبالهٔ ۴ توکنی داریم:

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

خروجی ضرب داخلی (قبل از `softmax`) یک ماتریس $4 \times 4$ 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].

## ۸.۲ مبدل های بینایی

ایده ترانسفورمرها در حوزهٔ بینایی [۴۶] از تعمیم ترانسفورمر متن به تصاویر به وجود آمده است [۱۱].

ما در این بخش از مبدل های بینایی برای وظیفه کلاس‌بندی استفاده می‌کنیم.

در روش‌های متداول برای پردازش تصویر، از کانولشن [۴۷] های 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 شود و شباهت یا ارتباطشان با یکدیگر سنجیده شود. در بخش‌های بعد، به‌طور مفصل روند انجام این کار را توضیح خواهیم داد.

### ۱.۸.۲ جاسازی پچ ها در مبدل های بینایی

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

حال همین ایده در تصویر پیاده‌سازی شده است. همان‌طور که در شکل ۲.۸.۸ مشاهده می‌کنید، در مبدل های بینایی، به‌جای استفاده از عملیات کانولوشن‌های متوالی که در شبکه‌های پیچشی [۴۸]

[۴۶] vision transformer
[۴۷] convolution
[۴۸] convolutional neural network

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 توجه برای ارتباط بین این بلاک‌ها استفاده کند [۱۱].

شکل ۲.۸.۸: بخش بندی تصاویر

### ۲.۸.۲ 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 $16 \times 16$)، 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.

### ۳.۸.۲ 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:

$$\left(\frac{H}{P}\right) \times \left(\frac{W}{P}\right) = \frac{H}{P} \times \frac{W}{P}. \qquad (۲.۸.۱۲)$$

برای مثال اگر:

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

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

در اکثر نسخه‌های مبدل‌های بینایی، پچ‌ها بدون هم‌پوشانی [۴۹] هستند. اندازه پچ‌های کوچک باعث می‌شود تعداد پچ‌ها زیاد شود و در نتیجه هزینه توجه بالا رود. از طرفی، پچ‌های بزرگ هزینه توجه را کاهش می‌دهند؛ اما ممکن است جزییات محلی [۵۰] را از دست بدهیم [۱۱].

### ۴.۸.۲ 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 پردازش زبان ظبیعی به مبدل ها بدهیم، باید آن‌ها را به یک بردار یک بعدی تبدیل کنیم. در صورت قرار دادن پیکسل‌های پچ به‌صورت ردیفی [۵۱]، طول این بردار خواهد بود:

$$P \times P \times C = P^2 \times C. \qquad (۲.۸.۱۳)$$

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

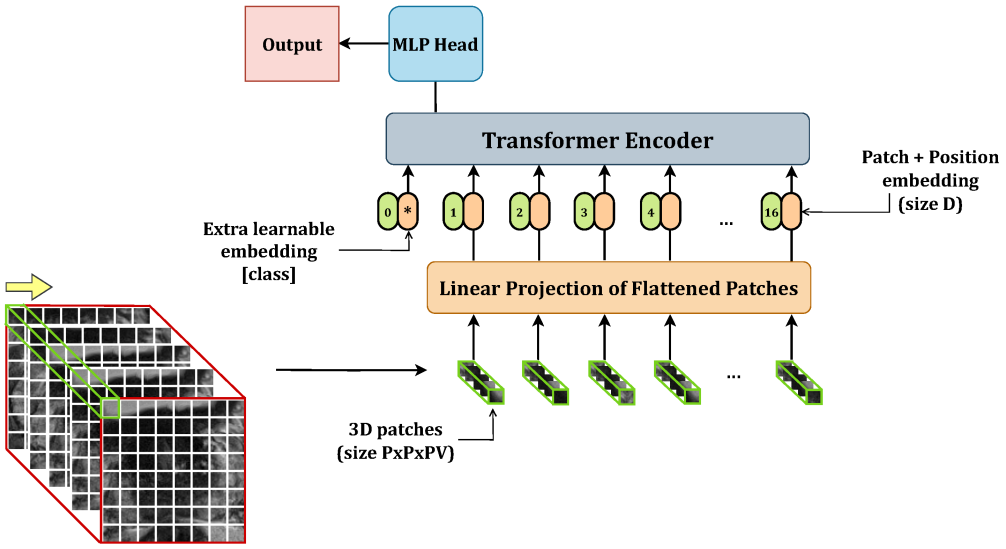
---

[۴۹] Non-overlapping
[۵۰] Local Details
[۵۱] Row-major

## ۹.۲ اعمال لایهٔ خطی

بعد از کنار هم چیدن پچ ها [۵۲]، معمولاً یک لایهٔ خطی [۵۳] روی این بردار اعمال می‌شود تا آن را به بعد $d_{\text{model}}$ (مثلاً ۷۶۸ یا ۱۰۲۴) ببرد. در حقیقت، این لایه یک تبدیل ویژگی [۵۴] انجام می‌دهد تا همه پچ‌ها یک نمایندگی (تعبیه شده) با ابعاد یکنواخت $d_{\text{model}}$ پیدا کنند:

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




شکل ۲.۹.۹: مبدل های بینایی

این مرحله شبیه ساخت توکن در پردازش زبان طبیعی است؛ با این تفاوت که در پردازش زبان طبیعی، توکن «کلمه» یا «زیرکلمه» است و از قبل دارای بردار تعبیه‌شده جاساز شده بوده است [۴۸]. در مبدل های بینایی [۱۱]، ما ابتدا باید تصاویر را پچ کنیم و سپس بردارهای جاساز را از این پچ‌ها به دست آوریم.

ترانسفورمر نیاز دارد ورودی‌اش توالی توکن‌ها باشد. در پردازش زبان طبیعی توالی کلمات داریم،

---

[۵۲] Flatten
[۵۳] Fully-Connected Layer
[۵۴] Feature Transformation

در مبدل های بینایی توالی «پچ»ها:

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

هر پچ اکنون یک بردار $d_{\text{model}}$ـبعدی است. پس یک مجموعه با طول $N$ (تعداد پچ‌ها) و عرض $d_{\text{model}}$ خواهیم داشت. اگر عدد پچ‌ها $N$ باشد (مثلاً ۱۹۶)، ترانسفورمر می‌تواند با مکانیزم توجه خود سر، وابستگی میان پچ‌ها را یاد بگیرد: کدام بخش از تصویر برای کدام بخش دیگر مهم‌تر است[۴۸، ۱۱].

معمولاً پچ‌ها را به‌صورت ردیفی 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 جاسازی موقعیتی هم اضافه کنیم، مکان دقیق هر پچ در بُعد دوم (ویژگی) کد می‌شود.

در مبدل بینایی [۱۱] دیگر به کانولوشن وابسته نیستیم. در عوض، از جاسازی استفاده می‌شود. تقسیم کردن تصویر به بلاک‌های $(P \times P)$، کنار هم چیدن و تبدیل آن به جاساز همگی عملیات ریاضی ساده‌ای هستند که به‌راحتی روی TPU/GPU قابل موازی‌سازی‌اند.

### ۱.۹.۲ توکن کلاس بندی

توکن کلاس بندی [۵۵] 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 مبدل بینایی، این توکن در ابتدای پچ‌های تصویری قرار می‌گیرد. این توکن یک بردار با ابعاد $d_{\text{model}}$ 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 کلاس بندی، هدف این است که یک پیش‌بینی کلی برای کل ورودی (مثلاً یک جمله یا یک تصویر) ارائه دهیم؛ توکن کلاس بندی دقیقاً همین وظیفه را بر عهده دارد [۹]. این

[۵۵] Cls Token

توکن از طریق مکانیزم توجه چند سر در مبدل ها با تمامی توکن‌های دیگر (پچ‌های تصویر) ارتباط می‌گیرد و اطلاعات مهم آن‌ها را در لایه‌های مختلف مبدل ها را به‌صورت تجمعی یاد می‌گیرد. به عبارتی، توکن کلاس بندی نقش نماینده کل تصویر یا متن را بر عهده دارد.

توکن کلاس بندی از طریق ضرب داخلی در مکانیزم توجه، می‌تواند به تمام پچ‌ها نگاه کند و با ضرایب توجه ($\alpha$) 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.

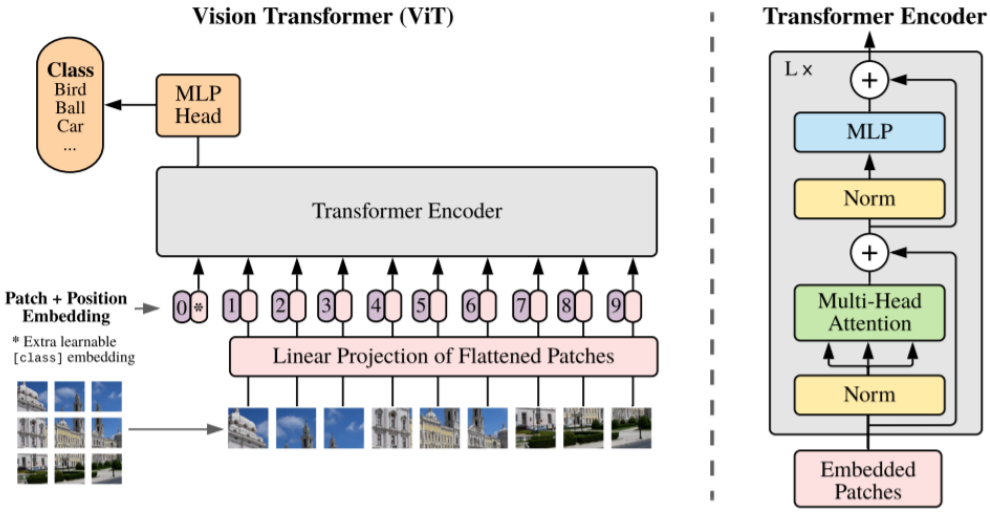
در طول لایه‌های ترانسفورمر، توکن کلاس بندی نقش محوری در خلاصه‌سازی بازنمایی کل تصویر ایفا می‌کند. این توکن به‌صورت پارامتر قابل یادگیری تعریف شده و در طول فرآیند آموزش به‌روزرسانی می‌شود [۹، ۱۱].

### ۲.۹.۲ رمزگذار در مبدل های بینایی

رمزگذار در ترانسفورمرها همانند مبدل اصلی است [۴۸]، با این تفاوت که در مبدل های بینایی [۱۱] دیگر به رمزگشا نمی‌رویم. پس از عبور از بلاک‌های ترانسفورمر، در ساده‌ترین حالت یک لایه خطی [۵۶] یا یک لایه MLP [۵۷] بر روی بردار نهایی اعمال می‌شود و این لایه‌ها به تعداد کلاس‌ها خروجی می‌دهند. سپس خروجی هر لایه با گذر از تابع سافت مکس [۵۸] به احتمال هر کلاس تبدیل می‌شود و در نهایت مدل کلاس با بیشترین احتمال را به‌عنوان خروجی پیش‌بینی می‌کند.

در مبدل ها، هر لایه رمزگشا و رمزگذار با پردازش عمیق‌تر روی توالی ورودی، می‌تواند نمایش بهتری از ویژگی‌ها به‌دست بیاورد [۴۸]. تکرار چندین‌باره رمزگشا یا رمزگذار 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

[۵۶] Fully Connected
[۵۷] Multi-Layer Perceptron
[۵۸] softmax




شکل ۲.۹.۱۰: توکن توجه در مبدل های بینایی

نشان می‌دهد.

## ۱۰.۲ مبدل پنجره‌ای متحرک

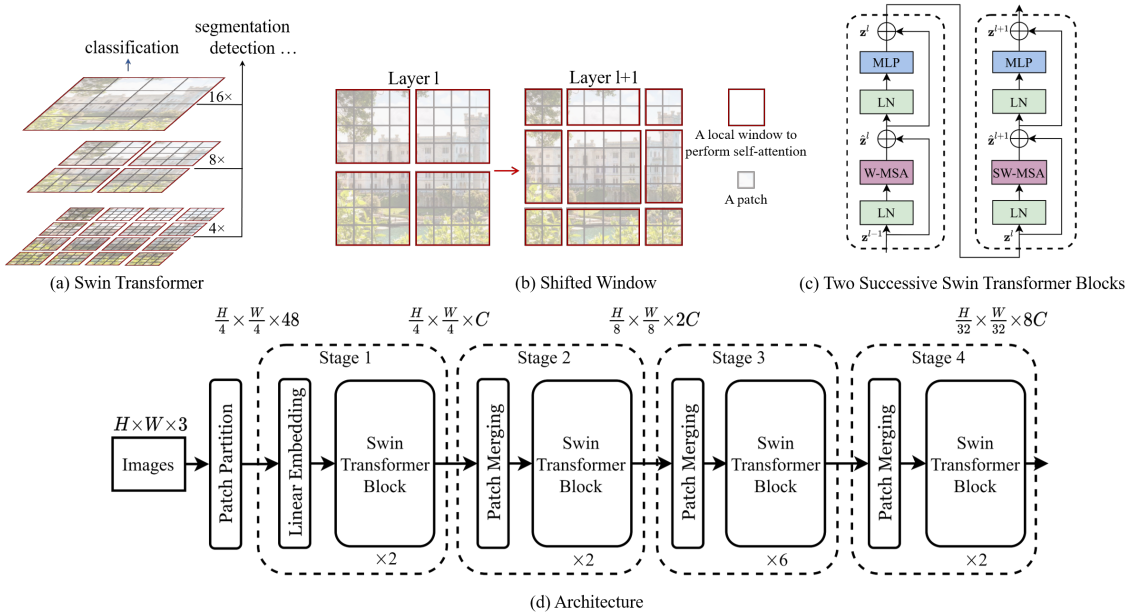
ایده مبدل پنجره‌ای متحرک [۵۹] از ترکیب چند مفهوم کلیدی در مدل‌های ترانسفورمر و شبکه‌های کانولوشنی شکل گرفت [۴۸، ۱۷، ۲۹].

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 می‌کرد و در مواقعی که تعداد پچ‌ها زیاد می‌شد، هزینه 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 مبدل پنجره‌ای متحرک [۲۹]، با دانش بر این موضوع توانسته‌اند هم

[۵۹] Swin Transformer

هزینه‌های محاسباتی را کاهش دهند و هم دقت مدل را افزایش دهند.




شکل ۲.۱۰.۱۱: مبدل پنجره متحرک

در مبدل پنجره‌ای متحرک، به‌جای آن‌که مدل به تمام پچ‌ها در یک سطح ویژگی نگاه کند، تصویر را به «پنجره‌های محلی» [۶۰] تقسیم می‌کند و توجه را محدود به همان ناحیه می‌سازد [۲۹]. سپس با تکنیک جابه‌جایی [۶۱] 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 کلیدی‌ای بود که باعث شد مدل هم محاسبات سبک‌تری داشته باشد و هم بتواند ارتباط‌های جهانی [۶۲] را در طول لایه‌ها به‌دست آورد.

یکی دیگر از ایده‌های مهم در در مبدل پنجره‌ای متحرک، کوچک کردن تدریجی نقشهٔ ویژگی در طول معماری است؛ مشابه کاری که در ResNet یا سایر CNNها انجام می‌شود [۱۷]. این امر ضمن کاهش هزینه محاسباتی، باعث می‌شود مدل بتواند با سطوح مختلفی از ویژگی‌ها کار کند و

---

[۶۰] Local Windows
[۶۱] Shift
[۶۲] Global

در نهایت خروجی نهایی باکیفیت‌تری ارائه دهد.

### ۱.۱۰.۲ قطعه‌بندی پچ در مبدل پنجره متحرک

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 [۶۳] 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:

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

برخلاف مبدل بینایی اصلی این کاردر مبدل پنجره متحرک با استفاده از کانولوشن [۶۴] انجام می شود. در واقع سایز کرنل در کانولوشن همان فضای برداری هر پچ هست که ما فرض میکنیم سایز کرنل برابر c است. پس درنتیجه

$$Z = X \cdot W_{\text{embed}} + b_{\text{embed}}, \quad Z \in \mathbb{R}^{\left(\frac{H}{P} \cdot \frac{W}{P}\right) \times C}. \tag{۲.۱۰.۱۴}$$

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

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

[۶۳] patch size
[۶۴] convolution

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 مبدل های پنجره متحرک خواهد بود [۲۹].

هر بلوک مبدل پنجره متحرک از چند بخش اصلی تشکیل شده است [۲۹]:

- پنجره‌بندی تصویر [۶۵] یا پنجره‌بندی جابه‌جاشده [۶۶]
- اعمال توجه چمد سر پنجره ای [۶۷]
- لایهٔ Skip Connection [۶۸] و Layer Norm [۶۹]
- مسیر پرسیپترون چندلایه [۷۰]:

### ۲.۱۰.۲ توجه چند سر پنجره ای

تعریف پنجره‌های محلی

در مبدل های پنجره متحرک، به‌جای آن‌که تمام پیکسل‌های یک نقشهٔ ویژگی بزرگ را یک‌جا در محاسبهٔ توجه درگیر کنیم، نقشهٔ ویژگی را به قطعه‌های کوچکی به‌اندازه $(M \times M)$ 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.

---

[۶۵] Window Partition
[۶۶] Shifted Window Partition
[۶۷] Window Multi-Head Self Attention
[۶۸] Skip Connection
[۶۹] Layer Norm
[۷۰] MLP

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 توجه را انجام می‌دهیم.
- این کار باعث می‌شود تعداد پیکسل‌هایی که درگیر محاسبهٔ توجه هستند، به‌مراتب کمتر شود و هزینهٔ محاسباتی کاهش یابد.

### ۳.۱۰.۲ توجه

برای هر بلوک، ابتدا بردارهای پرسش، کلید، مقدار 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 سر ها است. در توجه چند سر، 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 مبدل های پنجره متحرک، 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 مبدل های بینایی که همه‌چیز را با هم مقایسه می‌کند).

تعداد سربندی $h$ معمولاً طوری انتخاب می‌شود که $C = h \times d$. خروجی هر سر پس از محاسبه توجه 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.

### ۴.۱۰.۲ پنجره متحرک جا به جا شده

در مدبل های پنجر متحرک، ایدهٔ «پنجره‌های جابه‌جاشده» [۷۱] 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.

[۷۱] Shifted Windows

روش مبدل های پنجره متحرک برای رفع این محدودیت از یک تکنیک ساده اما مؤثر استفاده می‌کند [۲۹]:

- در یک لایه، محاسبات توجه 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.

توجه چند سری پنجره ای

در توجه چندسری پنجره ای [۷۲]، 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.

توجه چند سری پنجره ای جا به جا شده

مطابق شکل ۲.۱۰.۱۲، بعد از اینکه بلوک اول (توجه چند سری پنجره ای) کارش تمام شد، در بلوک دوم، قبل از تقسیم‌بندی به پنجره‌های $(M \times M)$، نقشهٔ ویژگی را جابه‌جا می‌کنیم [۲۹]. در مقالهٔ اصلی، این مقدار جابه‌جایی معمولاً نیمِ اندازهٔ پنجره $\frac{M}{2}$ در راستای افقی و عمودی است. به این ترتیب:

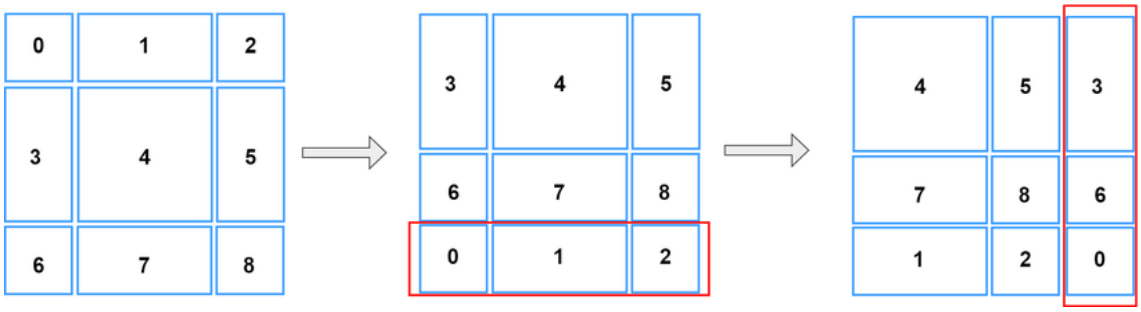
- پیکسل‌هایی که پیش از این در دو پنجرهٔ جداگانه قرار داشتند، ممکن است حالا به دلیل جابه‌جایی وارد یک پنجرهٔ مشترک شوند.

---

[۷۲] W-MSA

- مدل حالا می‌تواند بین این پیکسل‌های «مرزی» نیز توجه 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 جابجایی چرخه‌ای [۷۳] استفاده می‌شود. در جا به جایی چرخه ای ، 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 [۷۴]. مثالی از جا به جایی چرخه ای در شکل ۲.۱۰.۱۲ آمده است.




شکل ۲.۱۰.۱۲: جا به جایی چرخه ای

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].

---

[۷۳] Cyclic Shift

[۷۴] Wrap around

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 مبدل های پنجره متحرک تکمیل‌کنندهٔ یکدیگر می‌شوند [۲۹]:

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

### ۵.۱۰.۲ پرسپتروون چند لایه

پس از انجام توجه چند سری پنجره ای جا به جا شده خروجی به یک مسیر **MLP** می‌رود [۲۹]. ساختار این MLP به‌صورت زیر است:

$$X' = \mathrm{GELU}(XW_1 + b_1)\, W_2 + b_2, \qquad (۲.۱۰.۱۵)$$

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].

### ۶.۱۰.۲ ترکیب پچ ها

در مدل مبدل های پنجره متحرک، ساختار سلسله‌مراتبی به این معناست که ما در چند مرحله (Stage) مختلف، نقشهٔ ویژگی 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].

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 ترکیب پچ‌ها استفاده می‌کنیم. [۲۹]:

۱. انتخاب بلوک‌های $(2 \times 2)$

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

برای هر بلوک $(2 \times 2)$، این چهار پیکسل را در بُعد کانال به هم می‌چسبانیم. 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 $C$ـ بعدی را کنار هم می‌گذاریم، یک بردار $4C$ـ بعدی شکل می‌گیرد. حال با یک لایهٔ خطی، 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:

$$Z' \mapsto Z'' = Z' W_{\text{merge}} + b_{\text{merge}}, \qquad (۲.۱۰.۱۶)$$

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

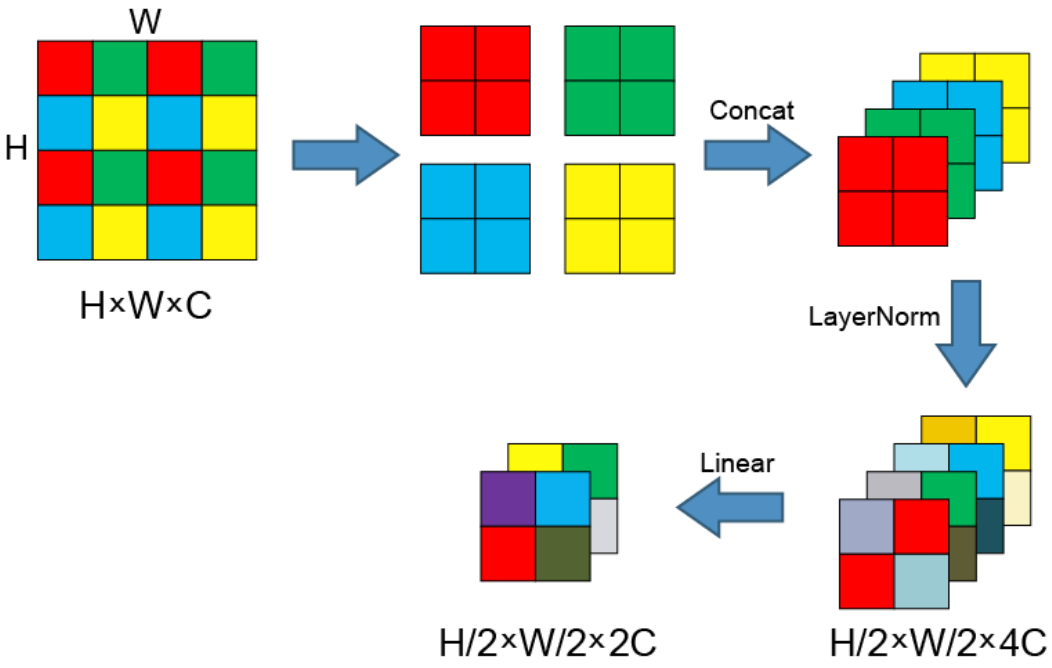
۴. کاهش ابعاد مکانی

در عین حال، وقتی هر چهار پیکسل $(2\times2)$ را ادغام می‌کنیم، نقشهٔ ویژگی ما ابعاد فضایی $\left(\frac{H}{2P} \times \frac{W}{2P}\right)$ خواهد داشت (چون هر بلوک $(2 \times 2)$ 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، مرتبا از لایه‌های ادغام [۷۵] یا کانولوشن با گام [۷۶] برای کوچک‌کردن ابعاد استفاده می‌شود تا اطلاعات سطح بالاتر (مثل ساختار کلی اشیا) راحت‌تر استخراج شود [۱۷]. در مبدل پنجره متحرک هم همین ایدهٔ سلسله‌مراتب را به دنیای مبدل ها آورده است [۲۹]. همچنین اگر ابعاد فضایی را کم نکنیم، هزینهٔ توجه به‌شدت زیاد می‌شود (چون باید در هر لایه برای همهٔ

[۷۵] Pooling
[۷۶] Stride-Convolution




شکل ۲.۱۰.۱۳: ادغام پچ ها

پیکسل‌ها توجه محاسبه گردد).

در معماری کلی کبدل های پنجره متحرک، پس از Stage 1 و عبور از بلوک‌های توجه چند سر پنجره ای و توجه چند سر پنجره ای جابه جا شده، عملیات ادغام پچ ها انجام می‌شود. سپس در Stage 2، ویژگی‌های کوچک‌تری داریم، اما تعداد کانال‌ها افزایش یافته است [۲۹]. مشابه معماری‌های کانولوشنی، با افزایش عمق [۷۷]، ابعاد فضایی کاهش و تعداد کانال‌ها افزایش پیدا می‌کند.

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

[۷۷] Depth

# فصل ۳

# پیشینه پژوهش

## استفاده از روش‌های تانسوری در شبکه‌های عصبی چندلایه (MLP)

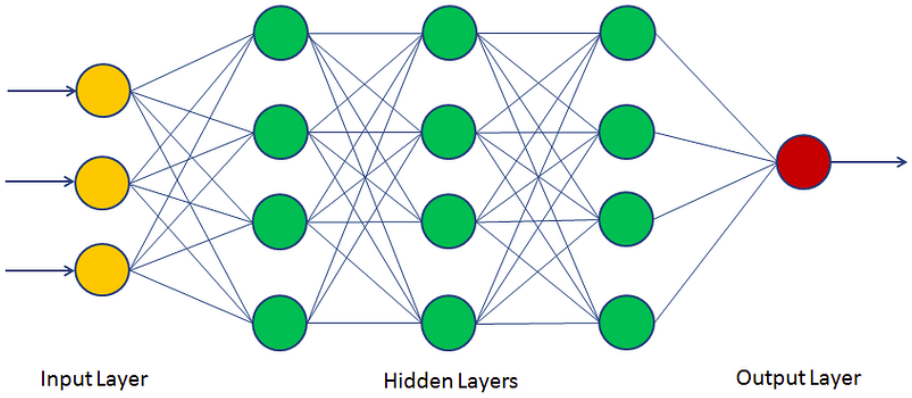
در سال‌های اخیر، استفاده از روش‌های تانسوری به‌عنوان روشی نوین در بهینه‌سازی معماری‌های شبکه‌های عصبی، به‌ویژه در مدل‌هایی که تعداد پارامترهای آن‌ها بسیار زیاد است، مانند شبکه‌های عصبی چندلایه [۱]، توجه بسیاری را به خود جلب کرده است. تانسورها تعمیمی از ماتریس‌ها به ابعاد بالاتر هستند و به‌طور طبیعی برای نمایش داده‌های چندبعدی همچون تصاویر، ویدیوها یا سری‌های زمانی چندکاناله مناسب‌اند. بهره‌گیری از ساختار تانسوری در معماری شبکه، این امکان را فراهم می‌سازد که بدون نیاز به فشرده‌سازی اولیه (مانند flatten کردن ورودی)، اطلاعات ساختاری میان ابعاد مختلف حفظ شده و مدل بتواند از روابط درون‌تعاملی موجود میان این ابعاد بهره‌برداری نماید.

در معماری سنتی MLP، نگاشت از ورودی $\mathbf{x} \in \mathbb{R}^n$ به خروجی $\mathbf{y} \in \mathbb{R}^m$ به‌وسیله ضرب ماتریسی انجام می‌گیرد:

$$\mathbf{y} = W\mathbf{x} + \mathbf{b}$$

[۱] Mlp

که در آن $W \in \mathbb{R}^{m \times n}$ ماتریس وزن و $\mathbf{b}$ بردار بایاس است.




شکل ۳.۰.۱: Mlp

در مقابل، در روش‌های تانسوری، وزن‌ها به‌صورت یک تانسور مرتبه بالاتر مدل‌سازی می‌شوند و نگاشت ورودی به خروجی با استفاده از ضرب‌های چندحالته (ضرب تانسوری در ابعاد مختلف) صورت می‌پذیرد:

$$\mathbf{y} = \mathcal{W} \times_1 \mathbf{x}_1 \times_2 \mathbf{x}_2 \times_3 \cdots + \mathbf{b}$$

در این رابطه، $\mathcal{W}$ یک تانسور وزن است و عملگر $\times_n$ نشان‌دهنده ضرب تانسوری در بعد $n$ام می‌باشد.

روش‌هایی نظیر TCL [۲] و TRL [۳] به‌عنوان نمونه‌هایی از این رویکرد، با بهره‌گیری از تکنیک‌های تجزیه تانسوری همچون Tucker یا CP decomposition، نه تنها باعث کاهش چشمگیر در تعداد پارامترها می‌شوند، بلکه ساختار چندبعدی داده‌ها را نیز حفظ می‌نمایند. این ویژگی به‌خصوص در مسائل دارای ورودی‌های دارای ساختار فضایی یا زمانی قابل‌توجه، بسیار حائز اهمیت است.

[۲] layer contraction tensor
[۳] Tensor regression layer

## مزایای استفاده از روش‌های تانسوری

استفاده از روش‌های تانسوری در شبکه‌های عصبی چندلایه (MLP) مزایای متعددی به همراه دارد که برخی از مهم‌ترین آن‌ها عبارت‌اند از:

- کاهش چشمگیر تعداد پارامترها: با بهره‌گیری از فشرده‌سازی تانسوری، می‌توان ابعاد تانسور وزن‌ها را به‌گونه‌ای کاهش داد که بدون افت محسوس در عملکرد مدل، مصرف حافظه و پیچیدگی محاسباتی به‌طور قابل توجهی کاهش یابد.
- حفظ ساختار داده‌های ورودی: برخلاف روش‌های سنتی که در آن‌ها داده‌ها پیش از ورود به لایه‌های چگال (Dense) باید مسطح‌سازی (Flatten) شوند، استفاده از ساختار تانسوری این امکان را فراهم می‌کند که ساختار فضایی، زمانی یا کانالی داده‌ها حفظ شده و ارتباط میان ابعاد مختلف ورودی بهتر درک و پردازش شود.

## محدودیت‌ها و چالش‌ها

با وجود مزایای متعدد، بهره‌گیری از روش‌های تانسوری در معماری‌های شبکه‌های عصبی با چالش‌ها و محدودیت‌هایی نیز همراه است که در ادامه به برخی از مهم‌ترین آن‌ها اشاره می‌شود:

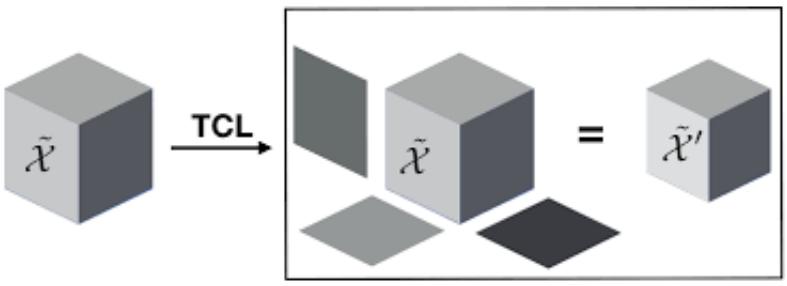
- پیچیدگی بالاتر در پیاده‌سازی: پیاده‌سازی لایه‌های مبتنی بر عملیات تانسوری معمولاً به ابزارها و کتابخانه‌های خاصی همچون Tensorly یا Tensor Toolbox نیاز دارد. این موضوع فرآیند طراحی و توسعه مدل را پیچیده‌تر از استفاده از لایه‌های استاندارد مانند Dense یا Conv می‌سازد.
- بهینه‌سازی دشوارتر: فرآیند آموزش مدل‌های تانسوری می‌تواند نسبت به مدل‌های معمولی کندتر باشد. الگوریتم‌های مبتنی بر گرادیان ممکن است در فضای پارامتری تانسورها با سطوح خطای غیرهموار یا چندوجهی مواجه شوند که روند همگرایی را دشوار می‌کند.

- احتمال کاهش دقت در فشرده‌سازی شدید: در صورتی‌که میزان فشرده‌سازی تانسورها بیش از حد بالا باشد، مدل ممکن است توانایی لازم برای نمایش روابط غیرخطی و الگوهای پیچیده را از دست داده و در نتیجه، دقت نهایی پیش‌بینی کاهش یابد.

### ۱.۰.۳ لایه فشرده‌سازی تانسوری (Tensor Contraction Layer)

در بسیاری از مدل‌های یادگیری عمیق، به‌ویژه در شبکه‌های عصبی کانولوشنی، فعال‌سازی‌های لایه‌های میانی به‌صورت تانسورهایی با مرتبه بالا ظاهر می‌شوند. به‌طور سنتی، برای اعمال لایه‌های Fully Connected، ابتدا این تانسورها با عملیات flattening به بردار تبدیل شده و سپس به فضای خروجی نگاشت داده می‌شوند. این فرآیند هرچند رایج است، اما موجب از بین رفتن ساختار چندخطی (multilinear structure) داده می‌شود و همچنین منجر به افزایش شدید تعداد پارامترهای شبکه می‌گردد.

برای مقابله با این چالش، لایه‌ای با عنوان لایه فشرده‌سازی تانسوری یا به اختصار TCL معرفی شده است. این لایه بدون نیاز به flatten کردن تانسور ورودی، ابعاد آن را در هر mode کاهش داده و در عین حال ساختار تانسوری داده را حفظ می‌کند.




شکل ۳.۰.۲: tensor contraction layer

فرمول‌بندی ریاضی

فرض شود تانسور فعال‌سازی ورودی به TCL به‌صورت زیر تعریف شده باشد:

$$\mathcal{X} \in \mathbb{R}^{S \times I_0 \times I_1 \times \cdots \times I_N}$$

که در آن $S$ اندازه‌ی دسته‌ی آموزشی (batch size) و $I_0, I_1, \ldots, I_N$ ابعاد تانسور هستند (برای مثال عرض، ارتفاع و کانال‌های رنگی یک تصویر).

هدف لایه TCL، کاهش هر یک از ابعاد $I_k$ به $R_k$ با استفاده از ماتریس‌های فشرده‌سازی قابل آموزش به فرم زیر است:

$$V^{(k)} \in \mathbb{R}^{R_k \times I_k}, \quad \text{برای} k = 0, 1, \ldots, N$$

عملیات فشرده‌سازی نهایی با ضرب‌های چندحالته (n-mode product) به‌صورت زیر انجام می‌شود:

$$\mathcal{X}' = \mathcal{X} \times_1 V^{(0)} \times_2 V^{(1)} \cdots \times_{N+1} V^{(N)}$$

که در آن $\times_n$ نشان‌دهنده‌ی ضرب تانسور در مد $n$ام است. خروجی لایه TCL، یعنی $\mathcal{X}'$، یک تانسور فشرده‌شده با ابعاد زیر خواهد بود:

$$\mathcal{X}' \in \mathbb{R}^{S \times R_0 \times R_1 \times \cdots \times R_N}$$

بدین ترتیب، به‌جای تخت‌سازی و از بین رفتن ساختار چندبعدی داده، عملیات فشرده‌سازی در هر مد به‌صورت مستقل و ساختارمند انجام می‌شود.

تحلیل تعداد پارامترها

استفاده از TCL منجر به کاهش قابل توجه در تعداد پارامترهای مدل می‌شود. تعداد پارامترهای موردنیاز برای این لایه برابر است با:

$$\text{تعداد پارامترهای TCL} = \sum_{k=0}^{N} I_k \cdot R_k$$

در حالی که در یک لایه Fully Connected کلاسیک، پس از flatten کردن ورودی، تعداد پارامترها برابر خواهد بود با:

$$\text{تعدا پارامتر های FC} = \left(\prod_{k=0}^{N} I_k\right) \cdot O$$

که در آن $O$ تعداد نرون‌های خروجی لایه است. همان‌گونه که پیداست، ساختار TCL باعث جایگزینی ضرب با جمع در فرمول محاسبه‌ی پارامترها می‌شود، که این امر منجر به کاهش چشمگیر حافظه و هزینه محاسباتی مدل می‌گردد.

### ۲.۰.۳ لایه رگرسیون تانسوری (Tensor Regression Layer)

این لایه به‌صورت مستقیم و بدون نیاز به flatten کردن، تانسورهای با مرتبه بالا را به بردار خروجی مدل نگاشت می‌دهد؛ به‌گونه‌ای که ساختار چندبعدی داده حفظ شده و نگاشت خروجی نیز به‌صورت low-rank مدل‌سازی می‌شود.

فرمول‌بندی ریاضی

فرض شود تانسور ورودی به TRL به‌صورت زیر باشد:

$$\mathcal{X} \in \mathbb{R}^{S \times I_0 \times I_1 \times \cdots \times I_N}$$

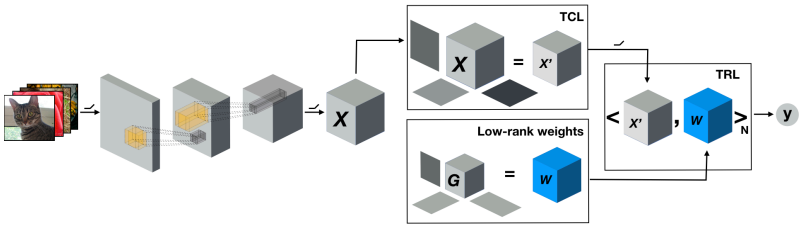
که در آن $S$ اندازه‌ی دسته‌ی آموزشی (batch size) و $I_k$ ابعاد تانسور هستند.

هدف لایه TRL نگاشت این تانسور به یک بردار خروجی $Y \in \mathbb{R}^{S \times O}$ است، که در آن $O$ تعداد نرون‌های خروجی است. نگاشت خطی میان تانسور ورودی و خروجی با استفاده از ضرب درونی تعمیم‌یافته صورت می‌گیرد:

$$\mathbf{Y} = \langle \mathcal{X}, \mathcal{W} \rangle_N + \mathbf{b}$$

که در آن:

- $\mathcal{W} \in \mathbb{R}^{I_0 \times I_1 \times \cdots \times I_N \times O}$ تانسور وزن‌های رگرسیون است.
- نماد $\langle \cdot, \cdot \rangle_N$ نشان‌دهنده‌ی ضرب داخلی بر روی $N$ بعد اول از $\mathcal{W}$ و $N$ بعد آخر از $\mathcal{X}$ است.
- $\mathbf{b} \in \mathbb{R}^O$ بردار بایاس است.




شکل ۳.۰.۳: tensor regression network

برای کنترل تعداد پارامترها و بهره‌گیری از ساختار تانسوری، تانسور $\mathcal{W}$ با استفاده از تجزیه Tucker مدل‌سازی می‌شود:

$$\mathcal{W} = \mathcal{G} \times_0 U^{(0)} \times_1 U^{(1)} \cdots \times_N U^{(N)} \times_{N+1} U^{(N+1)}$$

که در آن:

- $\mathcal{G} \in \mathbb{R}^{R_0 \times R_1 \cdots \times R_N \times R_{N+1}}$ هسته‌ی کم‌مرتبه (core tensor) است.
- $U^{(k)} \in \mathbb{R}^{I_k \times R_k}$ ماتریس‌های فشرده‌سازی برای ورودی هستند.
- $U^{(N+1)} \in \mathbb{R}^{O \times R_{N+1}}$ ماتریس فشرده‌سازی برای خروجی است.

فرمول نهایی خروجی مدل با جایگذاری $\mathcal{W}$ به‌صورت زیر خواهد بود:

$$\mathbf{Y} = \left\langle \mathcal{X}, \mathcal{G} \times_0 U^{(0)} \cdots \times_N U^{(N)} \times_{N+1} U^{(N+1)} \right\rangle_N + \mathbf{b}$$

یا به‌صورتی معادل و محاسباتی بهینه‌تر:

$$\mathbf{Y} = \left\langle \mathcal{X} \times_0 (U^{(0)})^\top \cdots \times_N (U^{(N)})^\top, \mathcal{G} \times_{N+1} U^{(N+1)} \right\rangle_N + \mathbf{b}$$

تحلیل تعداد پارامترها

در لایه Fully Connected سنتی، پس از flatten کردن ورودی، تعداد پارامترها برابر است با:

$$\text{تعداد پارامترهای FC} = \left( \prod_{k=0}^{N} I_k \right) \cdot O$$

اما در TRL، با در نظر گرفتن تجزیه Tucker، تعداد پارامترها به‌صورت زیر محاسبه می‌شود:

$$\text{تعداد پارامترهای TRL} = \left( \prod_{k=0}^{N+1} R_k \right) + \left( \sum_{k=0}^{N} I_k \cdot R_k \right) + R_{N+1} \cdot O$$

که معمولاً به‌طور چشم‌گیری کمتر از مدل سنتی است، به‌ویژه زمانی که مقادیر $R_k$ کوچک‌تر از $I_k$ انتخاب شوند.

### ۳.۰.۳ چرا در مبدل های بینایی از تانسور استفاده میکنیم؟

۱. کاهش تعداد پارامترها و حافظه مصرفی

یکی از چالش‌های اصلی در معماری‌های مبتنی بر Vision Transformer (ViT)، رشد نمایی تعداد پارامترها در لایه‌های Fully Connected به‌ویژه در بخش‌های انتهایی شبکه است.

با جایگزینی این لایه‌ها با ساختارهای تانسوری مانند Tensor Contraction Layer (TCL) و Tensor Regression Layer (TRL)، می‌توان از ساختار چندبعدی داده بهره برده و از طریق

تجزیه‌های کم‌مرتبه، تعداد پارامترها را به‌صورت چشمگیری کاهش داد. این جایگزینی نه‌تنها سبب صرفه‌جویی در حافظه می‌شود، بلکه پیچیدگی محاسباتی مدل را نیز کاهش داده و امکان به‌کارگیری آن را در محیط‌های کم‌منبع (مانند دستگاه‌های لبه‌ای و موبایل) فراهم می‌سازد.

افزایش تفسیرپذیری با حفظ ساختار چندخطی داده

در حالی‌که عملیات flattening روی تانسورهای ورودی منجر به از بین رفتن روابط ساختاری میان ابعاد مختلف داده می‌شود، استفاده از لایه‌های تانسوری مانند Tensor Contraction Layer (TCL) و Tensor Regression Layer (TRL)، این امکان را فراهم می‌سازد که ساختار چندبعدی داده حفظ شود. با حفظ این ساختار چندخطی، مدل قادر خواهد بود تا وابستگی‌ها و تعاملات میان ابعاد مختلف تصویر (مانند فضا، زمان و کانال‌های رنگی) را بهتر تحلیل کند.

این ویژگی به‌طور خاص در کاربردهایی که نیازمند تفسیرپذیری بالا هستند——مانند سیستم‌های تشخیص پزشکی، بینایی ماشین صنعتی، یا کاربردهای قانونی——می‌تواند نقش مهمی ایفا کند. زیرا در چنین حوزه‌هایی، درک تصمیمات مدل توسط انسان اهمیت بالایی دارد و تحلیل ساختار داخلی مدل بر پایه روابط تانسوری، درک بهتری از رفتار مدل فراهم می‌سازد.

۳. کاهش نیاز به داده‌های آموزشی بزرگ

مدل‌های Vision Transformer (ViT) به‌دلیل فقدان سوگیری مکانی ذاتی (مانند آنچه در شبکه‌های کانولوشنی وجود دارد)، نیازمند حجم عظیمی از داده برای آموزش مؤثر هستند. این مسئله در شرایطی که داده‌های برچسب‌خورده محدود هستند، به یک چالش جدی تبدیل می‌شود.

با استفاده از لایه‌های تانسوری مانند TCL و TRL، که نگاشت‌ها را به‌صورت چندخطی و فشرده مدل‌سازی کرده و ساختار درونی داده را حفظ می‌کنند، می‌توان از ظرفیت مدل به‌صورت بهینه‌تر بهره برد. این ساختار نه‌تنها به مدل اجازه می‌دهد که وابستگی‌های میان‌بعدی را بهتر درک کند، بلکه از بیش‌برازش در شرایط داده‌ی کم جلوگیری می‌نماید.

در نتیجه، معماری‌های مبتنی بر تانسور قادرند در دیتاست‌های کوچک نیز عملکردی قابل قبول داشته باشند و نیاز به حجم عظیم داده‌های آموزشی را کاهش دهند.

### ۴.۰.۳ روش تانسوری مبدل پنجره متحرک:

### ۵.۰.۳ پیاده‌سازی مرحله‌ی تعبیه پچ با استفاده از فشرده‌سازی تانسوری

در معماری‌های کلاسیک مبتنی بر مبدل های بیانی [۴] و همچنین در ساختار مبدل های پنجره ای [۵]، مرحله‌ی اولیه‌ی پردازش تصویر شامل تقسیم تصویر به قطعات کوچک (پچ‌ها) و سپس نگاشت هر پچ به یک بردار نهفته با ابعاد ثابت است. این نگاشت معمولاً با استفاده از یک لایه‌ی خطی [۶] یا پیچشی [۷] با کرنل و گام برابر با اندازه‌ی پچ انجام می‌شود.

در روش تانسوری به‌جای استفاده از نگاشت برداری ساده، از لایه‌ی فشرده‌سازی تانسوری [۸] برای تبدیل هر پچ به یک تانسور چندبعدی در فضای ویژگی استفاده شده است. این نگاشت تانسوری نه‌تنها باعث حفظ ساختار چندخطی پچ‌ها می‌شود، بلکه با فشرده‌سازی موثر، منجر به کاهش پارامترها می شود.

در ابتدا فرض میکنیم ورودی مدل تصویری با اندازه‌ی $(B, C, H, W)$ باشد که در آن:

- $B$ اندازه‌ی دسته‌ی آموزشی [۹] است.
- $C$ تعداد کانال‌های تصویر (برای مثال ۳ در تصاویر رنگی).
- $H \times W$ ابعاد تصویر است.

---

[۴] Vision transformer
[۵] swin transformer
[۶] linear
[۷] convolution
[۸] tensor contruction layer
[۹] Batch Size

این تصویر با استفاده از پچ‌هایی با اندازه‌ی $(P \times P)$ به $\frac{H}{P} \times \frac{W}{P}$ قطعه تقسیم می‌شود. سپس با استفاده از عملیات بازآرایی [۱۰]، ساختار ورودی به شکل زیر تبدیل می‌شود:

$$\mathcal{X} \in \mathbb{R}^{B \times P_1 \times P_2 \times C_1 \times C_2 \times C_3}$$

که در آن:

- $P_1 = \frac{H}{P}$ و $P_2 = \frac{W}{P}$ به‌ترتیب تعداد پچ‌ها در راستای ارتفاع و عرض تصویر هستند.
- $C_1 = P$ و $C_2 = P$ ابعاد مکانی هر پچ‌اند.
- $C_3 = C$ تعداد کانال‌های تصویر ورودی است.

در این بازنمایی، هر پچ در موقعیت $(i, j)$ به‌صورت یک تانسور سه‌بعدی با ابعاد $C_1 \times C_2 \times C_3$ نمایش داده می‌شود. این ساختار چندبعدی امکان استفاده‌ی مستقیم از عملیات تانسوری بدون نیاز به تخت‌سازی را فراهم می‌سازد.

برای نگاشت هر پچ به فضای نهفته، به جای استفاده از mlp از یک لایه‌ی فشرده ساز تانسوری استفاده می‌شود. در واقع سه تا بعد اخر با استفاده از لایه فشرده ساز تنسوری تعبیه میشود.

$$\mathcal{Z} \in \mathbb{R}^{B \times P_1 \times P_2 \times D_1 \times D_2 \times D_3}$$

در این ساختار، هر پچ به‌جای تبدیل به یک بردار تخت، به یک تانسور سه‌بعدی در فضای نهفته تبدیل می‌شود. این تانسور ساختار درونی پچ را حفظ کرد می کند.

فرمول‌بندی ریاضی عملیات فشرده‌سازی

فرض میکنیم $\mathcal{X}_{patch} \in \mathbb{R}^{C_1 \times C_2 \times C_3}$ نمایانگر یک پچ ورودی باشد. برای فشرده‌سازی این تانسور به فضای نهفته $(D_1, D_2, D_3)$، از سه ماتریس فشرده‌سازی قابل آموزش استفاده می‌شود:

[۱۰] rearrangement

$$V^{(0)} \in \mathbb{R}^{D_1 \times C_1}, \quad V^{(1)} \in \mathbb{R}^{D_2 \times C_2}, \quad V^{(2)} \in \mathbb{R}^{D_3 \times C_3}$$

عملیات فشرده‌سازی با استفاده از ضرب‌های چندحالته [۱۱] صورت می‌گیرد:

$$\mathcal{X}_{emb} = \mathcal{X}_{patch} \times_0 V^{(0)} \times_1 V^{(1)} \times_2 V^{(2)}$$

که در آن $\mathcal{X}_{emb} \in \mathbb{R}^{D_1 \times D_2 \times D_3}$ نگاشت نهفته‌ی تانسوری برای آن پچ خواهد بود.

در این مدل ساختار چند بعدی به پچ ها داده میشود و همچنین با استفاده از تانسور تعداد پارامتر ها کاهش پیدا میکنند و همچنین ظرفیت مدل نیز افزایش پیدا میکند.

در نتیجه، خروجی نهایی مرحله‌ی تعبیه پچ به‌صورت زیر تعریف می‌شود:

$$\mathcal{Z} \in \mathbb{R}^{B \times P_1 \times P_2 \times D_1 \times D_2 \times D_3}$$

که در آن هر پچ به‌صورت یک تانسور کم‌بعد در فضای نهفته نمایش داده شده است.

### ۶.۰.۳ ماژول توجه سلف چندسری مبتنی بر پنجره به‌صورت تانسوری

همانطور که در فصل قبل بیان شد در ساختار مبدل پنجره ای، جهت کاهش پیچیدگی محاسباتی، از روشی با عنوان [۱۲] بهره گرفته می‌شود. در این روش، ویژگی‌های مکانی تصویر به پنجره‌های کوچک با اندازه‌ی ثابت (بدون هم‌پوشانی) تقسیم شده و سپس عملیات خودتوجهی به‌صورت محلی و درون هر پنجره مستقل انجام می‌گیرد.

در روش تانسوری، به‌جای تولید بردارهای تخت‌شده‌ی $Q$، $K$ و $V$، این بردارها به‌صورت تانسورهای چندبعدی با استفاده از لایه فشرده ساز تانسوری تولید می‌شوند.

[۱۱] n-mode product

[۱۲] Window-based Multi-Head Self-Attention

ساختار ورودی و تقسیم به پنجره‌ها

خروجی مرحله تعبیه به‌صورت تانسوری به صورت زیر است

$$\mathcal{X} \in \mathbb{R}^{B \times H \times W \times D_1 \times D_2 \times D_3}$$

در گام اول، همانند مبدل های پنجره ای به پنجره‌های $w \times w$ تقسیم می‌شوند. و خروجی ما به صورت زیر خواهد بود.

$$\mathcal{X}_{win} \in \mathbb{R}^{B \times N_H \times N_W \times w \times w \times D_1 \times D_2 \times D_3}$$

که در آن $N_H = \frac{H}{w}$ و $N_W = \frac{W}{w}$ به‌ترتیب تعداد پنجره‌ها در راستای ارتفاع و عرض تصویر هستند.

۲. محاسبه‌ی Q، K و V با استفاده از **TCL**

برای هر پنجره‌ی مکانی، سه لایه‌ی فشرده‌سازی تانسوری مستقل (TCL) برای استخراج تانسورهای Q، K و V طراحی شده‌اند. فرض می‌شود تانسور ورودی هر پنجره دارای ابعاد:

$$\mathcal{X}_{patch} \in \mathbb{R}^{w \times w \times D_1 \times D_2 \times D_3}$$

باشد. با اعمال سه عملیات ضرب مد-n به‌ترتیب در ابعاد تانسوری، نگاشت‌های Q، K و V به‌صورت زیر حاصل می‌شوند:

$$\mathcal{Q} = \mathcal{X}_{patch} \times_2 V_Q^{(1)} \times_3 V_Q^{(2)} \times_4 V_Q^{(3)}$$

$$\mathcal{K} = \mathcal{X}_{patch} \times_2 V_K^{(1)} \times_3 V_K^{(2)} \times_4 V_K^{(3)}$$

$$\mathcal{V} = \mathcal{X}_{patch} \times_2 V_V^{(1)} \times_3 V_V^{(2)} \times_4 V_V^{(3)}$$

که در آن هر $V^{(i)}$ ماتریس فشرده‌سازی با ابعاد $\mathbb{R}^{D_i \times D_i}$ است. بنابراین، خروجی هر لایه به فضای تانسوری مشابه با ورودی بازمی‌گردد:

$$\mathcal{Q}, \mathcal{K}, \mathcal{V} \in \mathbb{R}^{w \times w \times D_1 \times D_2 \times D_3}$$

۳. تقسیم سرهای توجه چندگانه (Multi-Head)

برای پیاده‌سازی مکانیزم چندسر (multi-head)، ابعاد تانسوری خروجی به بخش‌های کوچک‌تری تقسیم می‌شود. فرض می‌شود:

$$D_1 = h_1 \cdot d_1, \quad D_2 = h_2 \cdot d_2, \quad D_3 = h_3 \cdot d_3$$

که در آن:

- $h_1, h_2, h_3$ تعداد سرها در هر بُعد تانسوری،
- $d_1, d_2, d_3$ ابعاد نهفته‌ی هر سر در هر بُعد هستند.

با استفاده از عملیات بازآرایی، تانسورهای Q، K و V به شکل زیر سازمان‌دهی می‌شوند:

$$\mathcal{Q}_{head} \in \mathbb{R}^{B \times N_H \times N_W \times w \times w \times h_1 \times h_2 \times h_3 \times d_1 \times d_2 \times d_3}$$

و ساختار مشابهی برای K و V در نظر گرفته می‌شود.

۴. محاسبه ماتریس توجه تانسوری

عملیات ضرب داخلی تعمیم‌یافته بین تانسورهای Q و K برای محاسبه‌ی ماتریس توجه انجام می‌شود. برای هر سر $(a, b, c)$ و برای موقعیت‌های $(i, j)$ و $(k, l)$ در پنجره، مقدار توجه به‌صورت زیر محاسبه می‌شود:

$$\text{Attention}_{ijkl}^{abc} = \sum_{x=1}^{d_1}\sum_{y=1}^{d_2}\sum_{z=1}^{d_3} Q_{ij}^{abcxyz} \cdot K_{kl}^{abcxyz}$$

که نتیجه نهایی یک ماتریس توجه با ابعاد زیر است:

$$\text{Attention} \in \mathbb{R}^{B\times N_H\times N_W\times h_1\times h_2\times h_3\times w^2\times w^2}$$

برای پایدارسازی محاسبات، نرمال‌سازی زیر اعمال می‌شود:

$$\text{scale} = \frac{1}{\sqrt{d_1 d_2 d_3}}$$

و از نسخه علامت‌دار تابع softmax برای اعمال توجه استفاده می‌گردد:

$$\text{Attention}_{\text{soft}} = \text{sign}(A)\cdot \text{softmax}(|A|)$$

۵. اعمال توجه و بازسازی ویژگی

پس از محاسبه ماتریس توجه، تانسور خروجی هر پنجره با استفاده از ترکیب توجه و تانسور V محاسبه می‌شود:

$$\mathcal{Y}_{ij}^{abcxyz} = \sum_{k,l} \text{Attention}_{ijkl}^{abc} \cdot V_{kl}^{abcxyz}$$

و پس از انجام توجه برای هر پنجره دوباره باز آرایی میشوند و به حالت اولیه باز میگردند و ابعاد آن به صورت زیر میشود.

$$\mathcal{Y} \in \mathbb{R}^{B\times H\times W\times D_1\times D_2\times D_3}$$

در این مدل چون سر ها در سه بعد تفسیم میشوند در نتیجه این مدل آزادی خیلی بیشتری دارد.

### ۷.۰.۳ توجه علامت‌دار

در مکانیزم‌های استاندارد خود توجهی که در مبدل ها مورد استفاده قرار می‌گیرند، ماتریس‌های $Q$، $K$ و $V$ با یکدیگر تعامل دارند تا وزن‌های توجه استخراج شوند. در این فرآیند، بردارهای $Q$ (Query) و $K$ (Key) با استفاده از ضرب داخلی محاسبه می‌شوند و نتیجه برای هر توکن، بیانگر میزان اهمیت نسبی سایر توکن‌ها در ارتباط با آن توکن است.

فرمول رایج محاسبه توجه به‌صورت زیر تعریف می‌شود:

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

در این ساختار، خروجی $QK^\top$ می‌تواند شامل مقادیر مثبت، منفی یا صفر باشد؛ اما پس از اعمال تابع Softmax، تمامی مقادیر به‌صورت نرمال‌سازی‌شده و مثبت خواهند بود. در نتیجه،اطلاعات قطبیت (علامت) که در نتیجه‌ی ضرب داخلی بین بردارهای $Q$ و $K$ وجود داشت، از بین می‌رود. این ویژگی باعث محدود شدن ظرفیت مدل در شناسایی «رابطه منفی یا متضاد» میان توکن‌ها می‌شود.

برای حفظ این اطلاعات قطبیت، د از رویکردی با عنوان توجه علامت دار استفاده شده است. ایده‌ی اصلی این روش آن است که پس از محاسبه‌ی امتیازهای توجه $A = QK^\top$، عملیات نرمال‌سازی Softmax بر قدرمطلق این امتیازها انجام شده و سپس علامت اولیه آن‌ها به نتیجه بازگردانده می‌شود.

فرمول این روش به‌صورت زیر تعریف می‌شود:

$$\text{Attention}_{\text{sign}} = \text{sign}(A) \cdot \text{Softmax}(|A|)$$

که در آن:

- $A = QK^\top$ ماتریس امتیازهای اولیه توجه است.
- $\text{sign}(A)$ عملیاتی است که علامت هر مقدار را حفظ می‌کند (مقادیر $+1$، $-1$ یا $0$).

- $|A|$ قدرمطلق مقادیر است که شدت شباهت را بدون در نظر گرفتن جهت نشان می‌دهد.

برای درک بهتر تفاوت میان مکانیزم‌های توجه معمولی و توجه علامت دار یک مثال عددی ساده اما گویای زیر ارائه می‌شود.

فرض کنیم بردار پرس‌وجو (Q) و دو بردار کلید (K) به‌صورت زیر تعریف شده‌اند:

$$Q = [1,\ 2,\ -1], \quad K_1 = [1,\ 0,\ -1], \quad K_2 = [-1,\ 1,\ 1]$$

محاسبه امتیاز توجه ابتدا شباهت میان Q و هر کلید با استفاده از ضرب داخلی محاسبه می‌شود:

$$Q \cdot K_1 = (1)(1) + (2)(0) + (-1)(-1) = 1 + 0 + 1 = 2$$

$$Q \cdot K_2 = (1)(-1) + (2)(1) + (-1)(1) = -1 + 2 - 1 = 0$$

بنابراین بردار امتیاز توجه به‌صورت زیر حاصل می‌شود:

$$A = [2,\ 0]$$

الف) توجه معمولی: در توجه معمولی، از تابع softmax مستقیماً روی مقادیر $A$ استفاده می‌شود:

$$\text{Softmax}(A) = \left[\frac{e^2}{e^2 + e^0},\ \frac{e^0}{e^2 + e^0}\right] \approx [0.88,\ 0.12]$$

در اینجا، هر دو کلید مقدار وزن مثبت می‌گیرند، حتی اگر امتیاز شباهت دومی برابر با صفر بوده باشد. مدل نمی‌تواند تفاوت میان بی‌اثر بودن و تأثیر منفی یا متضاد را به‌خوبی درک کند.

ب) توجه علامت‌دار: در این رویکرد، ابتدا بردار $A$ به دو بخش علامت و قدرمطلق تفکیک می‌شود:

$$\text{sign}(A) = [1,\ 0], \quad |A| = [2,\ 0]$$

سپس تابع softmax فقط روی قدرمطلق‌ها اعمال شده و خروجی زیر حاصل می‌شود:

$$\text{Softmax}(|A|) = \left[\frac{e^2}{e^2 + e^0},\ \frac{e^0}{e^2 + e^0}\right] \approx [0.88,\ 0.12]$$

در نهایت، با ضرب عنصربه‌عنصر با علامت‌ها:

$$\text{Attention}_{\text{sign}} = \text{sign}(A) \cdot \text{Softmax}(|A|) = [0.88,\ 0.00]$$

در این حالت، کلید دوم به‌طور کامل کنار گذاشته شده است، چرا که امتیاز آن صفر بوده و نشانی از تأثیر مثبت یا منفی ندارد.

در توجه معمولی، تمامی مقادیر صفر یا منفی به وزن‌های مثبت نرمال‌سازی می‌شوند. در مقابل، توجه علامت‌دار توانایی تمایز بین ارتباط مثبت، منفی و خنثی را دارد. این قابلیت می‌تواند در مدل‌سازی دقیق‌تر تضادهای معنایی یا شباهت‌های منفی نقش مهمی ایفا کند.

### ۸.۰.۳ توجه مبتنی بر پنجره‌های جابه‌جاشده به صورت تانسوری

توجه پنجره ای دارای یک محدودیت مهم است: پچ‌های واقع در پنجره‌های متفاوت هیچ‌گونه تبادل اطلاعاتی ندارند. در حالی که ممکن هست این پچ ها به هم نزدیک باشند به‌منظور رفع این محدودیت، مبدل های پنجره ای مکانیزم توجه مبتنی بر پنجره جا به جا شده معرفی شده است که در آن پنجره‌ها در برخی لایه‌ها به‌صورت جابه‌جاشده تعریف می‌شوند و از ماسک توجه برای جلوگیری از تداخل غیرمجاز استفاده می‌شود.

مراحل دقیق مکانیزم SW-MSA

۱. شیفت چرخشی تانسور ورودی: خروجی مرحله قبلی (مثلاً embedding patch یا لایه MSA قبلی) با یک cyclic shift به اندازه $\lfloor \frac{w}{2} \rfloor$ پیکسل در راستای افقی و عمودی جابه‌جا می‌شود. این جابه‌جایی باعث می‌شود که پچ‌هایی که پیش‌تر در پنجره‌های جداگانه بودند، اکنون در یک پنجره جدید قرار گیرند.

۲. اعمال پنجره‌بندی جدید: تانسور جابه‌جاشده به پنجره‌های بدون هم‌پوشانی جدید با اندازه $w \times w$ تقسیم می‌شود. پنجره‌های جدید از موقعیت‌های مختلف تصویر تشکیل شده‌اند.

۳. اعمال Self-Attention به‌صورت محلی با ماسک: از آنجا که پنجره‌ها پس از شیفت ممکن است شامل توکن‌هایی از مکان‌های نامرتبط باشند، نیاز است که توجه به‌صورت مقید (Masked) انجام شود. یک ماسک دودویی روی ماتریس توجه اعمال می‌شود که فقط اجازه توجه به توکن‌هایی را می‌دهد که واقعاً در یک پنجره معتبر قرار دارند.

۴. بازگرداندن شیفت (Reverse Shift): پس از انجام attention و بازآرایی خروجی، شیفت اولیه با همان اندازه اما در جهت معکوس انجام می‌شود تا تصویر به موقعیت مکانی اصلی خود بازگردد.

فرمول‌بندی ریاضی

فرض شود $\mathcal{X} \in \mathbb{R}^{B \times H \times W \times C}$ ویژگی ورودی باشد. مراحل به‌صورت زیر انجام می‌شوند:

- شیفت:

$$\mathcal{X}_{shifted} = \text{Roll}(\mathcal{X}, \text{shifts} = (-s, -s), \text{dims} = (1, 2))$$

که در آن $s = \lfloor \frac{w}{2} \rfloor$.

- پنجره‌بندی:

$$\mathcal{X}_{win} \in \mathbb{R}^{B \cdot N \times w \times w \times C}$$

که $N = \frac{H \cdot W}{w^2}$ تعداد پنجره‌هاست.

- اعمال attention:

$$\text{Attention}_{masked} = \text{Softmax}(QK^\top + M)$$

که $M$ یک ماسک با مقدار $-\infty$ در مکان‌های نامعتبر است.

- بازگشت به موقعیت اصلی:

$$\mathcal{Y} = \text{Roll}(\mathcal{Y}_{merged}, \text{shifts} = (+s, +s), \text{dims} = (1, 2))$$

ساخت ماسک توجه

هنگام شیفت تصویر، ممکن است پنجره‌ها حاوی پچ‌هایی از نواحی مختلف شوند که نباید با یکدیگر تعامل داشته باشند. بنابراین، برای حفظ محلی بودن attention، یک ماسک $M \in \mathbb{R}^{w^2 \times w^2}$ برای هر پنجره ساخته می‌شود که به صورت زیر تعریف می‌شود:

$$M_{ij} = \begin{cases} 0 & \text{اگر پچ‌های } i \text{ و } j \text{ در یک پنجره معتبر باشند} \\ -\infty & \text{در غیر این صورت} \end{cases}$$

افزودن این ماسک به logits قبل از softmax باعث می‌شود که attention فقط درون نواحی مجاز عمل کند و از نشت اطلاعات جلوگیری شود.

مزایای SW-MSA

- ایجاد ارتباطات میان‌پنجره‌ای: برخلاف W-MSA که کاملاً محلی است، SW-MSA اجازه می‌دهد که اطلاعات از چند ناحیه مختلف در یک پنجره تجمیع شوند.

- بدون افزایش پیچیدگی محاسباتی: همچنان توجه فقط در پنجره‌های $w \times w$ انجام می‌شود و هزینه محاسباتی کنترل‌شده باقی می‌ماند.
- افزایش بازنمایی محلی و جهانی: شبکه می‌تواند به‌صورت متناوب اطلاعات را از سطح محلی و سپس از سطح وسیع‌تر یاد بگیرد.
- قابلیت انطباق با Attention Masked در معماری‌های تانسوری: در نسخه‌های تانسوری Transformer Swin نیز امکان اعمال SW-MSA همراه با ماسک وجود دارد، بدون نیاز به flatten کردن یا از دست دادن ساختار چندبعدی داده.

### ۹.۰.۳ ادغام پچ‌ها به‌صورت تانسوری

در مدل‌های سلسله‌مراتبی بینایی مانند Swin Transformer، پس از هر مرحله از استخراج ویژگی، لازم است ابعاد مکانی تصویر کاهش یابد تا ویژگی‌ها در سطوح بالاتر به‌صورت فشرده‌تر و معنایی‌تر نمایش داده شوند. این فرایند با عنوان Patch Merging شناخته می‌شود. در نسخه‌ی اصلی این مدل، ادغام پچ‌ها با استفاده از ترکیب ۴ پچ مجاور (به‌صورت یک پنجره‌ی $2 \times 2$) و سپس اعمال یک لایه‌ی خطی پس از مسطح‌سازی انجام می‌شود. این کار باعث نصف شدن ابعاد مکانی و افزایش ابعاد ویژگی می‌شود.

در این پژوهش، رویکرد متفاوتی برای ادغام پچ‌ها ارائه شده است که مبتنی بر نگاشت تانسوری است. به‌جای انجام flatten بر روی پچ‌ها، از ساختار چندبعدی پچ‌ها استفاده شده و با کمک یک لایه‌ی فشرده‌سازی تانسوری (Tensor Contraction Layer)، ادغام پچ‌ها بدون از بین رفتن ساختار چندخطی آن‌ها انجام می‌شود.

ساختار ورودی

فرض شود خروجی مرحله قبلی شبکه دارای ساختار تانسوری زیر باشد:

$$\mathcal{X} \in \mathbb{R}^{B \times H \times W \times R_1 \times R_2 \times C}$$

که در آن:

- $B$ اندازه دسته آموزشی (batch size)،
- $H \times W$ ابعاد مکانی پچ‌ها،
- $(R_1, R_2, C)$ ابعاد فضای نهفته‌ی هر پچ به‌صورت تانسوری.

هدف این مرحله، کاهش ابعاد مکانی به $\left(\frac{H}{2}, \frac{W}{2}\right)$ و در عین حال، افزایش غنای بازنمایی ویژگی‌ها است.

ادغام پچ‌های مجاور

برای کاهش ابعاد مکانی، ابتدا هر چهار پچ مجاور در پنجره‌ای $2 \times 2$ انتخاب می‌شوند. این چهار پچ به‌صورت زیر دسته‌بندی می‌شوند:

- پچ بالا ـ چپ،
- پچ بالا ـ راست،
- پچ پایین ـ چپ،
- پچ پایین ـ راست.

در مرحله بعد، این چهار پچ با یکدیگر ترکیب می‌شوند، به این صورت که هرکدام به‌صورت مستقل نگه داشته شده و در امتداد بعد کانالی $(C)$ به یکدیگر متصل می‌شوند. در نتیجه، در هر موقعیت مکانی جدید، تانسوری با ابعاد $(R_1, R_2, 4C)$ خواهیم داشت. در سطح کلان، خروجی میان‌مرحله‌ای به‌صورت زیر خواهد بود:

$$\mathcal{X}_{\text{merged}} \in \mathbb{R}^{B\times\frac{H}{2}\times\frac{W}{2}\times R_1\times R_2\times 4C}$$

فشرده‌سازی ویژگی‌های ادغام‌شده

پس از ادغام کانالی، لازم است ابعاد ویژگی‌ها به صورت کنترل‌شده کاهش یابد تا بتوان آن را به شکل ورودی ماژول‌های بعدی تطبیق داد. برای این منظور، از یک لایه‌ی فشرده‌سازی تانسوری استفاده می‌شود که به‌صورت مستقیم بر روی ابعاد نهفته $(R_1, R_2, 4C)$ عمل می‌کند. برخلاف لایه‌های خطی کلاسیک، در اینجا عملیات فشرده‌سازی در چندین بعد به‌صورت همزمان و با حفظ ساختار چندخطی انجام می‌شود.

خروجی حاصل، تانسوری با ساختار:

$$\mathcal{Y} \in \mathbb{R}^{B\times\frac{H}{2}\times\frac{W}{2}\times R_1'\times R_2'\times C'}$$

که در آن $(R_1', R_2', C')$ ابعاد جدید ویژگی‌ها هستند و با توجه به محدودیت طراحی، باید رابطه زیر میان ابعاد قدیم و جدید برقرار باشد:

$$2 \cdot R_1' \cdot R_2' \cdot C' = 4 \cdot R_1 \cdot R_2 \cdot C$$

این رابطه تضمین می‌کند که ادغام از نظر اطلاعاتی قابل‌پشتیبانی و از نظر مدل‌سازی متوازن باقی می‌ماند.

مزایای طراحی تانسوری در ادغام پچ‌ها

- حفظ ساختار داده: برخلاف روش تخت‌سازی، این روش ساختار تانسوری داخلی هر پچ را حفظ می‌کند و به مدل اجازه می‌دهد تا وابستگی‌های میان ابعاد مختلف (مکانی، کانالی، و ویژگی‌های داخلی) را بهتر مدل کند.

- کاهش پیچیدگی پارامتری: عملیات فشرده‌سازی در TCL بر اساس ضرب‌های چندحالته است که در مقایسه با لایه‌های Dense کلاسیک، پارامتر کمتری نیاز دارد.
- تفسیرپذیری بهتر: خروجی تانسوری، امکان تحلیل مستقل ویژگی‌ها در هر بعد را فراهم می‌کند.
- هماهنگی با معماری تانسوری مدل: چون ساختار تانسوری در کل معماری حفظ می‌شود، نیاز به تبدیل‌های اضافی بین مراحل وجود ندارد.

# فصل ۴

# آزمایشات و نتایج

# کتاب‌نامه


[1] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016. 36, 37

[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. 22, 24, 25, 38, 40

[3] Yoshua Bengio et al. *Learning long-term dependencies with gradient descent is difficult*. IEEE Transactions on Neural Networks, 1994. 34, 35

[4] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, New York, 2006. 6, 7, 8, 12, 13, 14

[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. 25

[6] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995. 12

[7] Thomas M. Cover and Peter E. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1):21–27, 1967. 11

[8] Daniel Crevier. *AI: The Tumultuous History of the Search for Artificial Intelligence*. Basic Books, New York, 1993. 3, 4

[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. 24, 45, 46

[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. 12, 13

[11] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, et al. An image is worth 16x16 words: Transformers for image recognition at scale, 2020. 41, 42, 43, 44, 45, 46, 47, 49

[12] Richard O. Duda and Peter E. Hart. *Pattern Classification and Scene Analysis*. John Wiley & Sons, New York, 1973. 11

[13] Jeffrey L. Elman. Finding structure in time. *Cognitive Science*, 14(2):179–211, 1990. 15, 25, 30

[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. 3, 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. 13, 15, 17, 19, 20

[16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, Cambridge, MA, 2016. 6, 14, 16, 17, 20, 21, 22

[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. 34, 35, 42, 47, 48, 56, 57

[18] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016. 55

[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. 15, 16, 20, 21

[20] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997. 13, 16, 17, 18, 20, 21, 30, 34, 35

[21] John Hutchins. *Machine translation: past, present, future*. Ellis Horwood Chichester, 1986. 24

[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. 36, 37

[23] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An Introduction to Statistical Learning: with Applications in R*. Springer, New York, 2013. 7, 11

[24] Philipp Koehn. *Statistical Machine Translation*. Cambridge University Press, 2010. 25

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

[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. 42

[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. 42

[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. 47, 48, 50, 52, 53, 54, 55, 56, 57, 58

[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. 25

[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. 13

[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. 26, 27

[35] Tom M. Mitchell. *Machine Learning*. McGraw-Hill, New York, 1997. 6, 11, 12

[36] Douglas C. Montgomery, Elizabeth A. Peck, and Geoffrey G. Vining. *Introduction to Linear Regression Analysis*. John Wiley & Sons, Hoboken, NJ, 6th edition, 2021. 8

[37] Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective*. MIT Press, Cambridge, MA, 2012. 6, 7, 11, 12, 13

[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. 24

[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. 3

[40] Nils J. Nilsson. *The Quest for Artificial Intelligence: A History of Ideas and Achievements*. Cambridge University Press, Cambridge, 2010. 3, 4, 5

[41] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *Proc. of EMNLP*, pages 1532–1543, 2014. 26, 27

[42] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. OpenAI report, 2018. 24

[43] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986. 13, 14, 16, 21

[44] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson, London, 3rd edition, 2016. 4, 5

[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. 25, 38, 40

[46] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 2nd edition, 2018. 9

[47] Vladimir Vapnik. *Statistical Learning Theory*. Wiley, New York, 1998. 12

[48] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017. 21, 23, 24, 25, 28, 29, 30, 34, 35, 36, 37, 38, 39, 41, 44, 45, 46, 47

# 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:** Transformers , Vision Transformer , Attention , Swin Transformer .

**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