



دانشگاه تربیت مدرس

دانشکده علوم ریاضی

پایان نامه دوره کارشناسی ارشد علوم کامپیوتر

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

توسط

سید محمد بادزهره

استاد راهنما

آقای دکتر منصور رزقی

تابستان ۱۴۰۴

تقدیم به

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

## قدردانی

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

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

## چکیده

مبدل‌های بینایی در سال‌های اخیر به یکی از ساختارهای اصلی در مدل‌های یادگیری عمیق برای پردازش تصاویر تبدیل شده‌اند. با این حال، ساختارهای کلاسیک این مبدل‌ها معمولاً شامل لایه‌های کاملاً متصل هستند که نیازمند تبدیل تانسورهای چندبعدی ورودی به بردارهای مسطح می‌باشند. این فرآیند منجر به افزایش چشم‌گیر پارامترها و از بین رفتن ساختار فضایی و روابط میان‌بعدی داده می‌شود. در این پژوهش، چارچوبی تانسوری برای طراحی مدل مبدل تصویری پنجره متحرک پیشنهاد شده است که با بهره‌گیری از لایه‌های فشرده‌سازی تانسوری (TCL) و رگرسیون تانسوری (TRL)، نگاشت‌های چندخطی میان ابعاد داده را مدل‌سازی می‌کند. این روش با حفظ ساختار چندبعدی تصویر، موجب کاهش قابل‌توجه تعداد پارامترها، حفظ اطلاعات ساختاری، و افزایش تفسیرپذیری مدل می‌شود. نتایج حاصل از پیاده‌سازی و ارزیابی مدل در مجموعه داده‌های استاندارد نشان می‌دهد که استفاده از ساختارهای تانسوری نه تنها پیچیدگی محاسباتی را کاهش می‌دهد، بلکه دقت طبقه‌بندی را نیز بهبود می‌بخشد.

# فهرست مطالب

۵	فهرست جداول
۹	فهرست تصاویر
۱	پیش‌گفتار
۲	۱ مقدمه
۲	۱.۰.۱ آغاز هوش مصنوعی و هدف اصلی . . . . .
۳	۲.۰.۱ دوره‌طلاعی و پیشرفت‌های اولیه . . . . .
۳	۳.۰.۱ انتظارات بیش از حد و ظهور عصر تاریک . . . . .
۴	۴.۰.۱ عوامل اصلی عصر تاریک هوش مصنوعی . . . . .
۴	۵.۰.۱ پایان عصر تاریک و بازگشت هوش مصنوعی . . . . .
۵	۱.۱ انواع مدل یادگیری ماشین و شبکه‌های عصبی . . . . .
۶	۱.۱.۱ یادگیری ماشین: مروری کلی . . . . .
۶	۲.۱.۱ تقسیم‌بندی‌های اصلی در یادگیری ماشین . . . . .
۶	۳.۱.۱ یادگیری نظارت‌شده . . . . .
۸	۴.۱.۱ یادگیری بدون نظارت . . . . .
۹	۵.۱.۱ یادگیری تقویتی . . . . .

۹	معرفی چند مدل از الگوریتم یادگیری کلاسیک	۶.۱.۱
۱۰	نزدیک‌ترین همسایه	۷.۱.۱
۱۱	ماشین بردار پشتیبان	۸.۱.۱
۱۲	بیز ساده	۹.۱.۱
۱۳	شبکه‌های عصبی بازگشتی و شبکه‌های حافظه بلندمدت کوتاه‌مدت	۱۰.۱.۱
۱۳	شبکه‌های عصبی بازگشتی	۱۱.۱.۱
۱۴	ساختار و عملکرد شبکه‌های عصبی بازگشتی	۱۲.۱.۱
۱۵	مزایا و معایب شبکه‌های عصبی بازگشتی	۱۳.۱.۱
۱۶	شبکه‌های حافظه بلندمدت-کوتاه‌مدت	۱۴.۱.۱
۱۶	ناپدید شدن گرادیان در شبکه‌های بازگشتی	۱۵.۱.۱
۱۷	ظهور شبکه‌های حافظه بلندمدت-کوتاه‌مدت	۱۶.۱.۱
	راه‌حل شبکه‌های حافظه بلندمدت-کوتاه‌مدت برای پایداری جریان	۱۷.۱.۱
۱۷	گرادیان‌ها	
۱۸	ساختار شبکه‌های حافظه بلند-مدت کوتاه-مدت	۲.۱
۱۸	وضعیت سلولی	۱.۲.۱
۱۸	دروازه‌ها	۲.۲.۱
۲۰	به‌روزرسانی وضعیت سلولی	۳.۲.۱
۲۱	مشکلات کلی شبکه‌های بازگشتی و ظهور مبدل‌ها	۴.۲.۱
۲۱	مشکل وابستگی ترتیبی در شبکه‌های بازگشتی	۵.۲.۱
۲۲	پیچیدگی محاسباتی و حافظه در شبکه‌های بلندمدت کوتاه‌مدت	۶.۲.۱
۲۲	مشکل پردازش وابستگی‌های غیرمتوالی	۷.۲.۱
۲۲	گرادیان‌های ناپایدار و مشکلات بهینه‌سازی	۸.۲.۱

۲۴	۲	پیشینه پژوهش
۲۴	۱.۲	مشکلات ترجمه ماشینی و مبدل ها
۲۵	۲.۲	ظهور ترانسفورمرها
۲۵	۳.۲	معماری مبدل ها
۲۷	۱.۳.۲	جاسازی
۲۸	۲.۳.۲	جاسازی موقعیتی
۲۹	۳.۳.۲	توجه
۳۲	۴.۳.۲	توجه چند سر
۳۴	۵.۳.۲	اتصال باقی مانده
۳۵	۴.۲	نرمال سازی لایه ها
۳۷	۱.۴.۲	اتصال باقی مانده
۳۸	۵.۲	رمزگشا
۳۹	۶.۲	توجه چند سری ماسک شده
۴۰	۷.۲	مثال عددی توجه ماسک شده
۴۱	۸.۲	مبدل های بینایی
۴۱	۱.۸.۲	جاسازی پچ ها در مبدل های بینایی
۴۲	۲.۸.۲	شکل پچ ها:
۴۲	۳.۸.۲	تعداد پچ ها:
۴۳	۴.۸.۲	بردارکردن هر پچ
۴۴	۹.۲	اعمال لایه خطی
۴۵	۱.۹.۲	توکن کلاس بندی
۴۶	۲.۹.۲	رمزگذار در مبدل های بینایی
۴۷	۱۰.۲	مبدل پنجره ای متحرک

۴۹	۱.۱۰.۲	قطعه‌بندی پیچ در مبدل پنجره متحرک
۵۰	۲.۱۰.۲	توجه چند سر پنجره ای
۵۱	۳.۱۰.۲	توجه
۵۲	۴.۱۰.۲	پنجره متحرک جا به جا شده
۵۵	۵.۱۰.۲	پرسپترون چند لایه
۵۶	۶.۱۰.۲	ترکیب پیچ ها

### ۳ روش های پیشنهادی

۶۲	۱.۰.۳	لایه فشرده‌سازی تانسوری
۶۴	۲.۰.۳	لایه رگرسیون تانسوری (TRL)
۶۵	۳.۰.۳	چرا در مبدل‌های بینایی از تانسور استفاده می‌کنیم؟
۶۶	۴.۰.۳	روش تانسوری مبدل پنجره متحرک:
۶۶	۵.۰.۳	پیاده‌سازی مرحله‌ی تعبیه پیچ با استفاده از فشرده‌سازی تانسوری
۶۸	۶.۰.۳	ماژول توجه سلف چندسری مبتنی بر پنجره به‌صورت تانسوری
۷۲	۷.۰.۳	توجه علامت‌دار
۷۵	۸.۰.۳	توجه مبتنی بر پنجره‌های جابه‌جاشده به صورت تانسوری
۷۶	۹.۰.۳	ادغام پیچ‌ها به‌صورت تانسوری
۷۸	۱۰.۰.۳	مرحله‌ی طبقه‌بندی نهایی با استفاده از رگرسیون تانسوری

### ۴ آزمایشات و نتایج

#### کتاب‌نامه

#### آ جزئیات مدل‌ها و جدول پارامترها



# فهرست جداول

۱.۱	مقایسه ویژگی‌های RNN و LSTM	۲۰
-----	-----------------------------	----

## فهرست تصاویر

۷	کلاس بندی	۱.۱
۸	رگرسیون	۲.۱
۹	خوشه بندی	۳.۱
۱۰	یادگیری تقویتی	۴.۱
۱۴	شبکه های عصبی بازگشتی	۵.۱
۱۸	مدل حافظه بلند مدت کوتاه مدت	۶.۱
۲۶	معماری ترانسفورمرها	۱.۲
۲۷	جاسازی کلمه ای	۲.۲
۲۹	اضافه کردن جا سازی موقعیتی به جاسازی کلمه ای	۳.۲
۳۱	نحوه به دست آوردن پرسش، کلید و مقدار	۴.۲
۳۲	توجه	۵.۲
۳۳	توجه چند سر	۶.۲
۳۹	رمزگشا در مبدل ها	۷.۲
۴۲	بخش بندی تصاویر	۸.۲
۴۴	مبدل های بینایی	۹.۲
۴۷	توکن توجه در مبدل های بینایی	۱۰.۲

۴۸	۱۱.۲ مبدل پنجره متحرک
۵۴	۱۲.۲ جابه جایی چرخه ای
۵۸	۱۳.۲ ادغام پیچ ها
۶۰	۱.۳ شبکه عصبی پرسپترون چند لایه
۶۲	۲.۳ لایه فشرده ساز تانسوری

## پیش گفتار

قدشتمقدکنقصدبثقلدقفخدلqxفادخفادخ

# 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 مبدل‌ها<sup>۱</sup> 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 این رو، فصل حاضر به منظور 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 شبکه‌های حافظه کوتاه مدت طولانی<sup>۲</sup> اختصاص دارد.

### 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

---

Transformer<sup>۱</sup>  
LSTM<sup>۲</sup>

۱۹۵۶ این اصطلاح را به کار گرفت [۳۷] 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<sup>۳</sup> شروع به کار کردند [۴۵، ۸].

### ۲.۰.۱ دوره طلایی و پیشرفت‌های اولیه

در دهه ۵۰ و ۶۰ میلادی، هوش مصنوعی<sup>۴</sup> 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<sup>۵</sup> که در این دوره به وجود آمدند و زمینه‌ساز اولین دستاوردهای هوش مصنوعی در بازی‌های تخته‌ای<sup>۶</sup> همچون شطرنج<sup>۷</sup> شدند [۴۴]. در این دوران، پیشرفت‌های بیشتری در پردازش زبان طبیعی<sup>۸</sup> و سیستم‌های خبره<sup>۹</sup> نیز صورت گرفت که این امید را در دانشمندان و محققان تقویت کرد که دستیابی به هوش مصنوعی عمومی<sup>۱۰</sup> به‌زودی ممکن خواهد بود [۱۴].

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

با وجود پیشرفت‌های هوش مصنوعی، محدودیت‌های تکنولوژی (مثل عدم وجود پردازنده‌های گرافیکی<sup>۱۱</sup> پر قدرت 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.

---

<sup>۳</sup>AGI, Artificial General Intelligence

<sup>۴</sup>Artificial Intelligence (AI)

<sup>۵</sup>Tree Search Algorithms

<sup>۶</sup>Board Games

<sup>۷</sup>Chess

<sup>۸</sup>Natural Language Processing (NLP)

<sup>۹</sup>Expert Systems

<sup>۱۰</sup>Artificial General Intelligence (AGI)

<sup>۱۱</sup>GPU

در نتیجه، هوش مصنوعی در دهه ۷۰ به مرحله‌ای از رکود وارد شد که به آن عصر تاریک هوش مصنوعی یا زمستان هوش مصنوعی<sup>۱۲</sup> می‌گویند [۸، ۳۳]. 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 [۴۵].
- کمبود داده‌ها: در آن زمان، دسترسی به داده‌های کافی برای آموزش مدل‌های پیچیده ممکن نبود و الگوریتم‌های موجود به داده‌های بیشتری نیاز داشتند تا بتوانند به درستی آموزش ببینند و عملکرد مطلوبی داشته باشند [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 [۵۰].

#### ۵.۰.۱ پایان عصر تاریک و بازگشت هوش مصنوعی

پس از چندین سال رکود و عدم سرمایه‌گذاری در حوزه هوش مصنوعی، سرانجام در دهه ۱۹۸۰ و ۱۹۹۰ عصر تاریک هوش مصنوعی با تحولات تکنولوژی و از همه مهم‌تر ظهور سیستم‌های خبره به پایان رسید [۱۴]. سیستم‌های خبره به عنوان یکی از اولین تلاش‌های موفق برای کاربردهای صنعتی در هوش مصنوعی به وجود آمدند. برخلاف الگوریتم‌های اولیه، این سیستم‌ها از پایگاه بزرگ قواعد و قوانین<sup>۱۳</sup> استفاده می‌کردند. در سیستم‌های خبره، به جای تلاش برای شبیه‌سازی

---

<sup>۱۲</sup> AI Winter

<sup>۱۳</sup> Rule Based Systems

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 [۳۸].

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 [۴۵].

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 توسعه شبکه‌های عصبی هموار کرد [۵۰].

## 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، ۴۰، ۴۲].



### ۱.۱.۱ یادگیری ماشین: مروری کلی

یادگیری ماشین<sup>۱۴</sup> 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، ۴۰]. 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 دسته اصلی تقسیم می‌شود:

- یادگیری با نظارت<sup>۱۵</sup>

- یادگیری بدون نظارت<sup>۱۶</sup>

- یادگیری تقویتی<sup>۱۷</sup>

این طبقه‌بندی در بسیاری از کتاب‌ها و مراجع مهم یادگیری ماشین مطرح شده است [۴، ۴۲].

### 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 [۲۴]. 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<sup>۱۸</sup> و کلاس‌بندی<sup>۱۹</sup> می‌شود.

---

<sup>۱۴</sup> Machine Learning

<sup>۱۵</sup> Supervised Learning

<sup>۱۶</sup> Unsupervised Learning

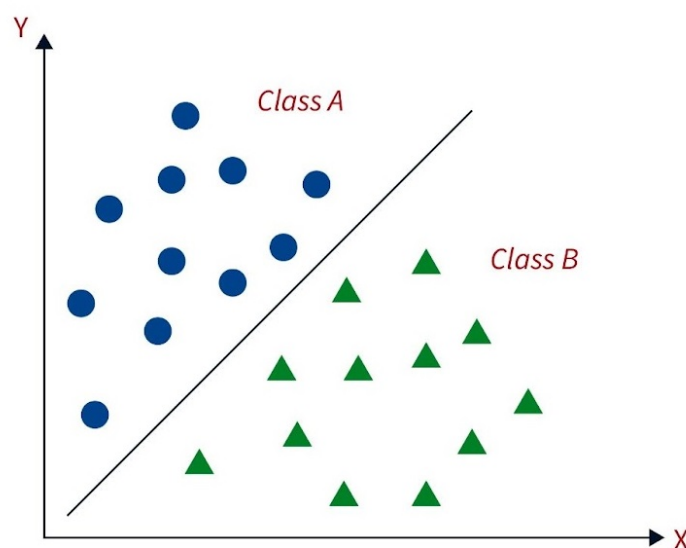
<sup>۱۷</sup> Reinforcement Learning

<sup>۱۸</sup> Regression

<sup>۱۹</sup> Classification

## کلاس‌بندی

در میان روش‌های یادگیری با نظارت، کلاس‌بندی<sup>۲۰</sup> 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<sup>۲۱</sup>، شناسایی بیماری‌ها، طبقه‌بندی تصاویر و تحلیل احساسات به‌طور گسترده مورد استفاده قرار می‌گیرد [۴، ۴۲].



شکل ۱.۱: کلاس‌بندی

## رگرسیون

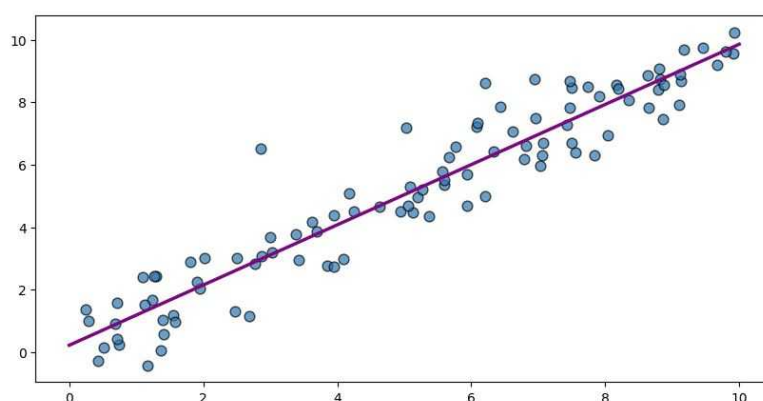
در مقابل، رگرسیون<sup>۲۲</sup> به مسئله پیش‌بینی مقادیر پیوسته می‌پردازد. در این نوع از یادگیری نظارت‌شده، مدل می‌کوشد رابطه‌ای میان متغیرهای ورودی و خروجی‌های عددی برقرار کند تا بر اساس آن، بتواند

<sup>۲۰</sup> Classification

<sup>۲۱</sup> Spam Detection

<sup>۲۲</sup> Regression

خروجی نمونه‌های جدید را تخمین بزنند. 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 وضعیت آب‌وهوا اشاره کرد [۴۱].



شکل ۲.۱: رگرسیون

#### ۴.۱.۱ یادگیری بدون نظارت

یادگیری بدون نظارت<sup>۲۳</sup> 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 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.

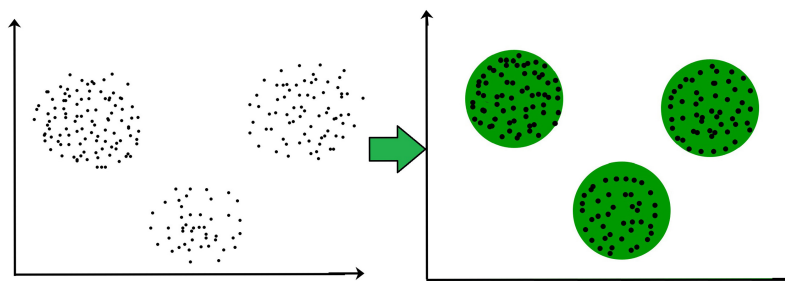
از کاربردهای رایج یادگیری بدون نظارت می‌توان به خوشه‌بندی<sup>۲۴</sup>، کاهش ابعاد<sup>۲۵</sup>، آشکارسازی

<sup>۲۳</sup> Unsupervised Learning

<sup>۲۴</sup> Clustering

<sup>۲۵</sup> Dimensionality Reduction

ناهنجاری‌ها<sup>۲۶</sup>، و استخراج ویژگی‌ها اشاره کرد.



شکل ۳.۱: خوشه بندی

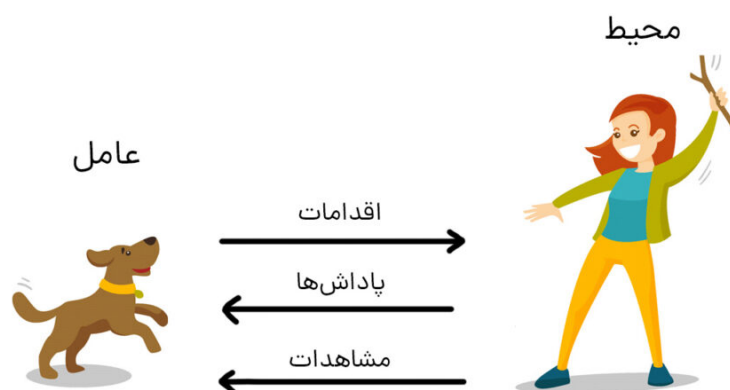
### ۵.۱.۱ یادگیری تقویتی

یادگیری تقویتی،<sup>۲۷</sup> نوعی یادگیری بر پایه پاداش و تنبیه است که در آن مدل با محیط تعامل می‌کند و بر اساس پاداش یا تنبیه یاد می‌گیرد [۵۲]. 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<sup>۲۸</sup>، کنترل رباتیک<sup>۲۹</sup> و سیستم‌های توصیه‌گر<sup>۳۰</sup> اشاره کرد.

### ۶.۱.۱ 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.

<sup>۲۶</sup> Anomaly Detection  
<sup>۲۷</sup> reinforcement learning  
<sup>۲۸</sup> Games  
<sup>۲۹</sup> Robotic Control  
<sup>۳۰</sup> Recommender Systems



شکل ۴.۱: یادگیری تقویتی

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 به کار 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 زمینه درک عمیق‌تر روش‌های نوین‌تر یادگیری، مانند شبکه‌های عصبی و مدل‌های عمیق، فراهم گردد.

### ۷.۱.۱ نزدیک‌ترین همسایه

الگوریتم نزدیک‌ترین همسایه<sup>۳۱</sup> یکی از روش‌های ساده و درعین حال کارآمد در یادگیری نظارت‌شده است که هم در دسته‌بندی و هم در رگرسیون کاربرد دارد [۷، ۱۲، ۴۰]. 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.

<sup>۳۱</sup>k-Nearest Neighbors

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 [۲۴].

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 نقطه جدید دارد و در داده‌های بزرگ بار محاسباتی بالایی خواهد داشت [۴۰].
- عدم کارایی در داده‌های با ابعاد بالا: در داده‌هایی با تعداد ویژگی‌های زیاد، کارایی الگوریتم کاهش می‌یابد [۴۲].

### ۸.۱.۱ ماشین بردار پشتیبان

الگوریتم ماشین بردار پشتیبان<sup>۳۲</sup> با یافتن یک ابرصفحه بهینه، داده‌ها را به کلاس‌های مختلف تقسیم می‌کند [۶، ۵۳]. 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 [۵۳].

---

Support Vector Machine, SVM<sup>۳۲</sup>

- مقاومت در برابر بیش‌برازش<sup>۳۳</sup>: با استفاده از هسته‌ها<sup>۳۴</sup>، داده‌های غیرخطی نیز به فضای بالاتر برده می‌شوند و جداسازی بهتری انجام می‌شود<sup>[۶]</sup>.

معایب:

- پیچیدگی محاسباتی: آموزش ماشین بردار پشتیبان به دلیل نیاز به حل مسائل بهینه‌سازی، در حجم‌های بالای داده محاسباتی زمان‌بر است<sup>[۲۲]</sup>.
- کارایی پایین در داده‌های پرت: در صورتی که داده‌ها شامل نقاط پرت زیادی باشند، دقت مدل کاهش می‌یابد<sup>[۴]</sup>.

### ۹.۱.۱ بیز ساده

بیز ساده<sup>۳۵</sup> مبتنی بر قضیه بیز<sup>۳۶</sup> است و فرض می‌کند ویژگی‌ها به صورت شرطی مستقل از هم هستند<sup>[۴۰، ۱۰]</sup>. این مدل برای اولین بار در حوزه پردازش متن به کار رفت و هنوز هم در بسیاری از کاربردها مانند طبقه‌بندی ایمیل و تحلیل احساسات مورد استفاده قرار می‌گیرد<sup>[۳۶]</sup>. 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 به عنوان جواب نهایی در نظر می‌گیرد<sup>[۴]</sup>.

مزایا:

- سرعت بالا: به دلیل محاسبات ساده و فرض استقلال ویژگی‌ها، بیز ساده بسیار سریع و کم‌حجم است<sup>[۳۶]</sup>.
- کارایی در داده‌های کوچک: حتی با داده‌های کم، این الگوریتم عملکرد نسبتاً خوبی دارد<sup>[۲۲]</sup>.

<sup>۳۳</sup> Overfitting

<sup>۳۴</sup> kernels

<sup>۳۵</sup> Bayes Naive

<sup>۳۶</sup> Bayes' theorem

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<sup>۳۷</sup> و مدل‌هایی با حافظه بلندمدت-کوتاه‌مدت<sup>۳۸</sup> با هدف پردازش داده‌های ترتیبی و وابسته به زمان توسعه یافتند [۴۹، ۲۱]. 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 لحظه حال و آینده استفاده می‌کنند [۱۵].

### ۱۱.۱.۱ شبکه‌های عصبی بازگشتی

مدل‌های اولیه شبکه‌های عصبی، مانند شبکه‌های چندلایه<sup>۳۹</sup>، 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 [۴۹].

---

RNN<sup>۳۷</sup>

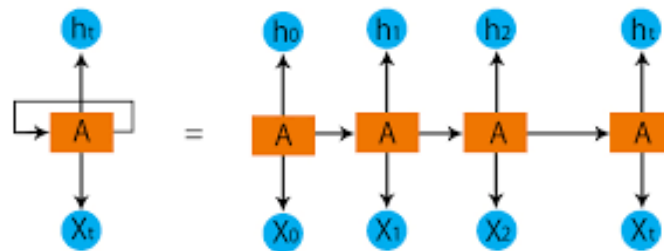
LSTM<sup>۳۸</sup>

MLP<sup>۳۹</sup>



### ۱۲.۱.۱ ساختار و عملکرد شبکه‌های عصبی بازگشتی

شبکه‌های شبکه‌های عصبی بازگشتی<sup>۴۰</sup> دارای حلقه 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 به عنوان ورودی دریافت کند [۱۶].



شکل ۵.۱: شبکه‌های عصبی بازگشتی

$$h_t = \sigma(W \cdot x_t + U \cdot h_{t-1} + b) \quad (۱.۱)$$

در اینجا:

- $h_t$  وضعیت مخفی یا حالت در گام زمانی  $t$  است.
- $W$  وزن‌هایی است که به ورودی  $x_t$  اعمال می‌شود.
- $U$  وزن‌های اعمال شده به وضعیت قبلی  $h_{t-1}$  است.
- $b$  بایاس مدل است.
- $\sigma$  تابع فعال‌سازی، معمولاً تانژانت هیپربولیک یا سیگموید.

<sup>۴۰</sup> recurrent neural network

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 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<sup>۴۱</sup>: در فرایند آموزش با روش پس‌انتشار<sup>۴۲</sup>، اگر توالی داده‌ها طولانی باشد، گرادیان‌ها ممکن است بسیار کوچک یا بزرگ شوند که منجر به ناپایداری در آموزش و کاهش دقت می‌شود [۲۰].
- 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 [۱۶، ۲۱].

---

Vanishing and Exploding Gradient<sup>۴۱</sup>  
back propagation<sup>۴۲</sup>

### 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 [۲۱]. یکی از برجسته‌ترین مشکلات موجود در شبکه‌های عصبی بازگشتی، معضل ناپدید شدن گرادیان بود که مانع یادگیری وابستگی‌های بلندمدت می‌شد [۲۰، 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 شبکه‌های 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<sup>۴۳</sup> استفاده می‌شود که گرادیان‌ها را جهت به‌روزرسانی وزن‌ها محاسبه می‌کند. [۴۹]. با این حال، شبکه‌های عصبی بازگشتی در یادگیری وابستگی‌های بلندمدت معمولاً ناکام می‌مانند. علت اصلی این امر شامل موارد زیر است:

- ضریب‌های بازگشتی کوچک‌تر از ۱: در فرایند محاسبه گرادیان‌ها، اگر مقدار مشتقات یا ضرایب در هر مرحله کوچک‌تر از ۱ باشد، ضرب مکرر این ضرایب در طول توالی منجر به کوچک شدن گرادیان‌ها به سمت صفر می‌شود؛ پدیده‌ای که به ناپدید شدن گرادیان<sup>۴۴</sup> معروف است [۲۰].

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

$$\frac{\partial L}{\partial W} = \prod_{k=1}^t \frac{\partial h_k}{\partial h_{k-1}} \cdot \frac{\partial h_t}{\partial L} \quad (2.1)$$

<sup>۴۳</sup> Backpropagation Through Time

<sup>۴۴</sup> vanishing gradient

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. [۲۱]. 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 شبکه‌های 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<sup>۴۵</sup> در شبکه‌های بازگشتی، جریان گرادیان‌ها را در طول توالی پایدار نگه می‌دارد. این کار از طریق اضافه کردن وضعیت سلولی<sup>۴۶</sup> و دروازه‌ها<sup>۴۷</sup> 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،

۳. اطلاعات مهم قبلی را حفظ کند.

---

<sup>۴۵</sup> long short-term memory

<sup>۴۶</sup> Cell State

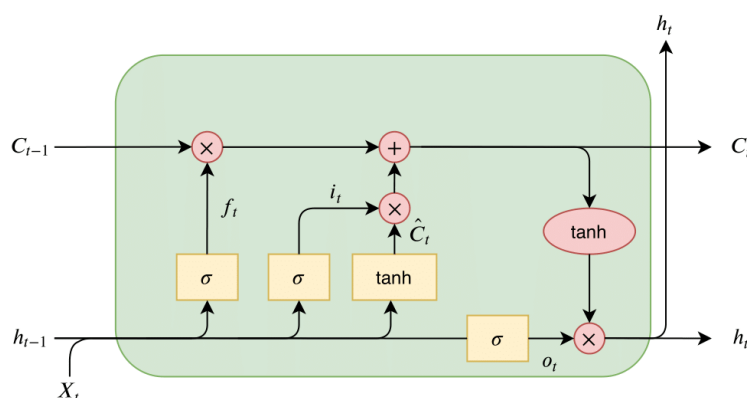
<sup>۴۷</sup> Gates

## ۲.۱ ساختار شبکه های حافظه بلند-مدت کوتاه-مدت

شبکه های حافظه بلند-مدت شامل اجزای جدیدی است که به آن امکان مدیریت بهتر اطلاعات را می دهد:

### ۱.۲.۱ وضعیت سلولی

مسیر اصلی ذخیره اطلاعات در شبکه های حافظه بلند-مدت کوتاه مدت است که می تواند اطلاعات مهم را در طول توالی حفظ کند. برخلاف شبکه های عصبی بازگشتی که عمدتاً بر خروجی های بازگشتی  $h_t$  متکی است، شبکه حافظه بلند مدت، کوتاه مدت یک مسیر جداگانه برای عبور اطلاعات از وضعیت سلولی دارد که به حفظ گرادیان ها کمک شایانی می کند [۲۱].



شکل ۶.۱: مدل حافظه بلند مدت کوتاه مدت

### ۲.۲.۱ دروازه ها

دروازه ها نقش فیلترهای اطلاعاتی را دارند که جریان اطلاعات را در طول فرایند یادگیری کنترل می کنند.

- دروازه فراموشی <sup>۴۸</sup> تعیین می کند چه اطلاعاتی از وضعیت سلولی باید حذف شود [۱۵].

<sup>۴۸</sup> Forget Gate

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \quad (۳.۱)$$

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.

- دروازه ورودی<sup>۴۹</sup>: تعیین می‌کند چه اطلاعات جدیدی باید به وضعیت سلولی اضافه شود:

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \quad (۴.۱)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \quad (۵.۱)$$

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

- دروازه خروجی<sup>۵۰</sup>:

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

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

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

---

Input Gate<sup>۴۹</sup>  
Output Gate<sup>۵۰</sup>

## ۳.۲.۱ به روزرسانی وضعیت سلولی

وضعیت سلولی  $C_t$  با استفاده از اطلاعات جدید و قدیمی به روزرسانی می شود:

$$C_t = f_t \cdot C_{t-1} + i_t \cdot \tilde{C}_t \quad (۶.۱)$$

این ساختار باعث می شود اطلاعات قدیمی مهم حفظ و داده های غیرضروری حذف شوند.

علت پایداری گرادیان در شبکه های حافظه بلند-مدت کوتاه مدت

- حذف ضرب های مکرر: برخلاف شبکه های بازگشتی که به ضرب های مکرر وزن ها و گرادیان ها وابسته است، شبکه های حافظه بلند-مدت کوتاه مدت با مسیر جداگانه وضعیت سلولی، از کاهش نمایی گرادیان جلوگیری می کند [۲۰].

- استفاده از توابع سیگموئید و تانژانت هیپربولیک: توابع سیگموئید در دروازه ها و تانژانت هیپربولیک در وضعیت سلولی مقادیر را محدود می کنند و مانع از انفجار گرادیان می شوند [۱۵]، [۱۶].

- مدیریت اطلاعات توسط دروازه ها: دروازه های فراموشی و ورودی به مدل اجازه می دهند تنها اطلاعات مهم حفظ شود و داده های غیرضروری حذف شوند. این موضوع از پیچیدگی محاسباتی غیرضروری جلوگیری می کند [۲۱].

ویژگی	RNN	LSTM
مشکل ناپدید شدن گرادیان	وجود دارد	برطرف شده
توانایی حفظ وابستگی های طولانی مدت	محدود به وابستگی کوتاه مدت	بسیار خوب
ساختار دروازه ها	ندارد	دارای دروازه های فراموشی، ورودی و خروجی
پایداری گرادیان	ضعیف	پایدار

جدول ۱.۱: مقایسه ویژگی های RNN و LSTM

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

شبکه های بازگشتی که به آن ها پرداخته شد توانستند بسیاری از مشکلات و محدودیت های مدل های اولیه را حل کنند؛ اما همچنان با چالش ها و محدودیت هایی مواجه بودند که در مسائل پیچیده تر، مانند ترجمه زبان یا تحلیل داده های بلندمدت و حجیم، مشکلات جدی ایجاد می کردند [۲۱، ۱۶]. این مشکلات در نهایت به پیدایش مبدل ها<sup>۵۱</sup> منجر شد [۵۴].

### ۵.۲.۱ مشکل وابستگی ترتیبی در شبکه های بازگشتی

شبکه های بازگشتی داده ها را به صورت ترتیبی پردازش می کنند؛ به این معنی که برای پردازش داده های گام زمانی  $t$ ، باید تمامی داده های قبلی  $(t-1)$  پردازش شده باشند [۲۱، ۴۹]. این ویژگی مشکلات زیر را ایجاد می کند:

- غیرقابل موازی سازی: به دلیل وابستگی ترتیبی، پردازش داده ها به صورت موازی ممکن نیست و همین امر باعث افزایش زمان محاسباتی می شود. در داده های بلند (مانند متن های طولانی یا سری های زمانی بزرگ)، این مشکل نمود بیشتری دارد.
- کندی آموزش و استنتاج: پردازش خطی داده ها موجب می شود زمان آموزش و پیش بینی مدل ها به شدت افزایش یابد، به ویژه زمانی که با حجم زیادی از داده مواجه هستیم.

محدودیت در یادگیری وابستگی های بسیار طولانی

با وجود پیشرفت شبکه های حافظه بلند-مدت کوتاه مدت در یادگیری وابستگی های بلندمدت نسبت به شبکه های بازگشتی معمولی، این مدل ها همچنان در یادگیری وابستگی های بسیار بلند، مانند ارتباط بین کلمات در جملات دور از هم یا درک ساختار کلی یک متن، محدودیت دارند [۲۰]:

- مشکل در داده های بسیار طولانی: حتی در شبکه های حافظه بلند-مدت کوتاه مدت نیز ظرفیت حفظ اطلاعات محدود است و با افزایش طول توالی، دقت مدل افت می کند.



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

### ۶.۲.۱ پیچیدگی محاسباتی و حافظه در شبکه‌های بلند مدت کوتاه مدت

شبکه‌های 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، شبکه‌های حافظه بلند-مدت کوتاه مدت به حافظه بیشتری نسبت به مدل‌های ساده‌تر احتیاج دارند.
- هزینه محاسباتی بالا: در داده‌های حجیم، انجام محاسبات سنگین می‌تواند اجرای مدل را بسیار کند سازد.

### ۷.۲.۱ مشکل پردازش وابستگی‌های غیرمتوالی

شبکه‌های 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. شبکه‌های 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 شبکه‌های بازگشتی معمولی در پایداری گرادیان ارائه داد، هنوز هم مشکلاتی در این شبکه‌ها 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 به دلیل محدودیت در حافظه و پردازش، پاسخ‌گوی این نیاز نیستند.
  - کارایی در داده‌های چندوجهی<sup>۵۲</sup>: 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 در نهایت به ظهور مبدل‌ها منتهی شد [۵۴]. مبدل‌ها با طراحی مبتنی بر موازی‌سازی و مکانیزم توجه<sup>۵۳</sup>، این محدودیت را برطرف کرده و راه‌حلی کارآمدتر برای پردازش داده‌های ترتیبی ارائه دادند.

## فصل ۲

### پیشینه پژوهش

با توجه به مشکلات مطرح شده شبکه های بازگشتی، معماری جدیدی به نام مبدل ها<sup>۱</sup> مطرح شد ظهور مبدل ها یکی از تحولات اساسی در حوزه پردازش زبان طبیعی<sup>۲</sup> و یادگیری ماشین به شمار می رود. [۵۴، ۲]. این مدل ها باعث تغییرات عمده ای در نحوه ساخت و آموزش مدل های زبانی و همچنین در بسیاری از کاربردهای دیگر یادگیری ماشین شده اند و توانستند بسیاری از مشکلات مدل های قبلی را حل کنند [۴۸، ۹].

#### ۱.۲ مشکلات ترجمه ماشینی و مبدل ها

در ابتدا، ترجمه ماشینی<sup>۳</sup> یک چالش اساسی در زمینه پردازش زبان طبیعی بود. مدل های اولیه ای مانند مدل های مبتنی بر قواعد<sup>۴</sup> برای ترجمه استفاده می شدند که در آن ها، ترجمه ها به صورت دستی با استفاده از قواعد زبانی مشخص تنظیم می شدند [۴۳، ۲۲]. این روش ها هرچند دقیق بودند، اما محدودیت های زیادی داشتند و نمی توانستند ویژگی های پیچیده تر زبان را مدل سازی کنند. سپس مدل های آماری<sup>۵</sup> معرفی شدند [۲۶، ۵]. این مدل ها از داده های ترجمه شده برای آموزش

---

<sup>۱</sup>Transformer

<sup>۲</sup>NLP

<sup>۳</sup>machine translation

<sup>۴</sup>Rule-based Models

<sup>۵</sup>Statistical Models

مدل‌های آماری استفاده می‌کردند که احتمال ترجمه صحیح را براساس شواهد آماری محاسبه می‌کردند. مدل‌هایی مانند مدل‌های ترجمه آماری مبتنی بر جمله<sup>۶</sup> [۲۷] 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<sup>۷</sup> به وجود آمدند که مشکلات آن‌ها در فصل گذشته بیان شد [۵۱، ۱۳]. در نهایت، این مشکلات باعث به وجود آمدن ترانسفورمرها شد [۲].

## ۲.۲ ظهور ترانسفورمرها

در سال ۲۰۱۷، مقاله‌ای توسط گوگل<sup>۸</sup> منتشر شد که مفهوم جدیدی به نام مبدل‌ها<sup>۹</sup> را معرفی کرد [۵۴]. 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 [۳۵].

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 مبدل‌ها در پردازش تصویر و متن بسیار سریع‌تر و دقیق‌تر از مدل‌های قبلی عمل کنند [۵۴].

## ۳.۲ معماری مبدل‌ها

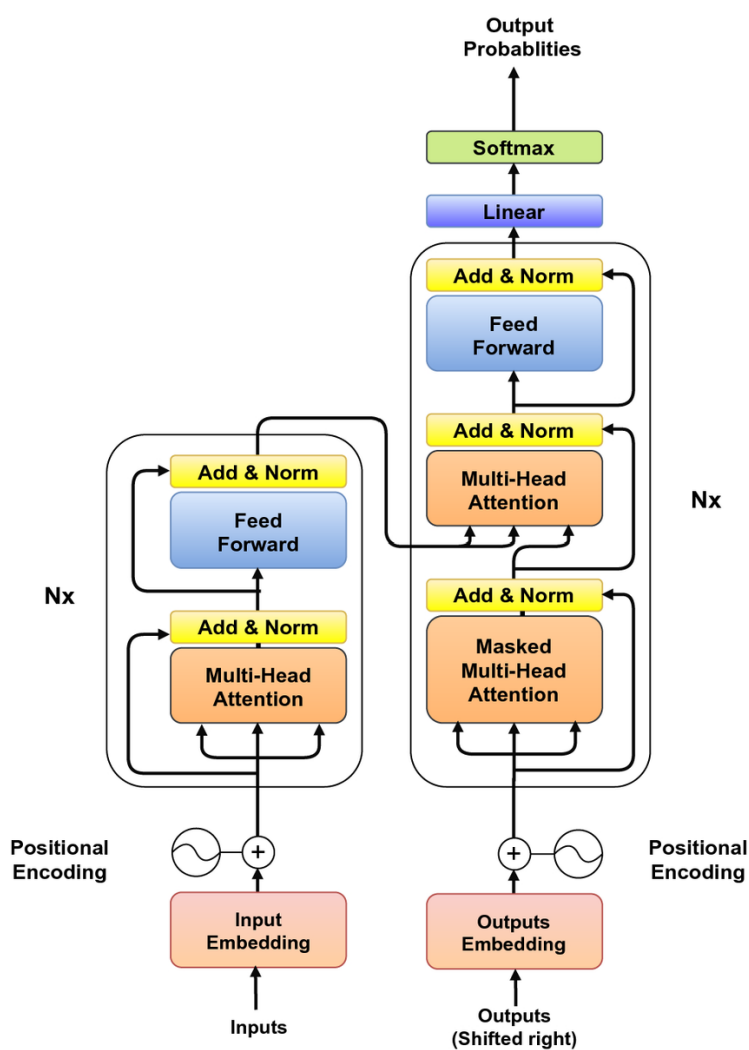
در تصویر ۱.۲، معماری مبدل نمایش داده شده است و بخش‌ها و اجزای مختلف آن مشخص شده است. معماری ترانسفورمر از دو بخش اصلی تشکیل شده است:

---

Phrase-based Statistical Models<sup>۶</sup>  
 Recurrent Models<sup>۷</sup>  
 google<sup>۸</sup>  
 Transformers<sup>۹</sup>

● رمزگذار<sup>۱۰</sup>: وظیفه رمزگذار این است که داده ورودی را دریافت کند و ویژگی‌های آن را استخراج کند.

● رمزگشا<sup>۱۱</sup>: وظیفه رمزگشا این است که ویژگی‌های استخراج‌شده را به زبان مقصد تبدیل کند.



شکل ۱.۲: معماری ترانسفورمرها

Encoder<sup>۱۰</sup>

Decoder<sup>۱۱</sup>

## 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 [۳۹] و GloVe [۴۷] اشاره کرد.

همان‌طور که در شکل ۲.۲ نشان داده شده است، هر کلمه که به صورت توکن است، ابتدا در دیکشنری تعریف شده پیدا می‌شود و پس از پیدا شدن در دیکشنری، با استفاده از روش‌های تعبیه کردن<sup>۱۲</sup>، هر کلمه به برداری از اعداد تبدیل می‌شود. این جاسازی‌ها شباهت‌های معنایی بین کلمات را مدل‌سازی می‌کنند و کلماتی که از نظر معنایی شبیه به هم هستند، بردار آن‌ها نیز به یکدیگر نزدیک‌تر است. به این ترتیب، کلمات برای مدل‌ها و شبکه‌های عصبی قابل فهم می‌شوند [۳۹، ۴۷].

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

شکل ۲.۲: جاسازی کلمه ای

## 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 به صورت موازی وارد می‌شوند، نیاز داریم تا جایگاه هر کلمه را بدانیم. به طور مثال، در جمله «من تو را دوست دارم» باید به طور دقیق بدانیم که «من» کلمه اول جمله است، «تو» کلمه دوم جمله است و ... .

حال باید به مدل توالی این کلمات را بفهمانیم. بنابراین، نیاز داریم به مدل یک سری اطلاعات اضافی بدهیم به طوری که مدل توالی کلمات را یاد بگیرد. روش‌های مختلفی برای اضافه کردن جاسازی موقعیتی<sup>۱۳</sup> به مدل وجود دارد. در ترانسفورمرها از روش جاسازی موقعیت سینوسی<sup>۱۴</sup> استفاده می‌شود [۵۴].

این روش قابل یادگیری نیست و صرفاً از یک سری فرمول‌های ساده برای جاسازی موقعیتی استفاده می‌کند. برای موقعیت مشخص<sup>۱۵</sup> در توالی و بُعد  $i$  در فضای برداری، تعبیه موقعیتی به صورت زیر تعریف می‌شود.

و برای مقادیر زوج:

$$PE(pos, 2i) = \sin\left(\frac{pos}{10000^{\frac{2i}{d}}}\right) \quad (1.2)$$

و برای مقادیر فرد:

$$PE(pos, 2i + 1) = \cos\left(\frac{pos}{10000^{\frac{2i}{d}}}\right) \quad (2.2)$$

•  $pos$ : موقعیت کلمه در توالی است (مثلاً از 0 تا  $N - 1$  برای یک توالی  $N$  - تایی).

<sup>۱۳</sup> Positional Embedding

<sup>۱۴</sup> Sinusoidal Positional Embedding

<sup>۱۵</sup> pos

- $i$ : 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 **۳.۲** 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 به گونه‌ای نگاشت می‌کنند که مدل بتواند از ترتیب کلمات در توالی آگاه باشد [۵۴]. 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.

YOUR	CAT	IS	A	LOVELY	CAT
952,207	171,411	621,659	776,562	642,693	171,411
5450,840	3276,350	1304,051	5567,288	6315,080	3276,350
1853,448	9192,819	5,565	38,942	9358,776	9192,819
1,458	3633,421	7679,805	2716,194	2141,081	3633,421
2671,529	8395,473	4506,025	5119,989	735,147	8395,473
+	+	+	+	+	+
...	1664,068	...	...	...	1281,458
...	8080,153	...	...	...	7902,890
...	2630,399	...	...	...	912,970
...	...	...	...	...	3821,102
...	9386,405	...	...	...	1459,217
...	3120,159	...	...	...	7016,620

شکل ۳.۲: اضافه کردن جا سازی موقعیتی به جاسازی کلمه ای

### ۳.۳.۲ توجه

در روش شبکه های بازگشتی، توالی ورودی (مثلاً یک جمله) معمولاً به صورت گام به گام پردازش می‌شد [۲۱، ۱۳]. اما در ترانسفورمر می‌خواهیم مدلی داشته باشیم که به هر موقعیت (مثلاً یک کلمه) در توالی نگاه کند و به همه موقعیت‌های دیگر نیز به صورت موازی دسترسی داشته باشد. به این مفهوم توجه<sup>۱۶</sup> می‌گوییم.

<sup>۱۶</sup> attention



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 این جا فعل «نشستن» ارتباط نزدیک‌تری به «گربه» دارد و از نظر معنایی مرتبط‌تر است.

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

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

در ضرب شباهت‌های توجه<sup>۱۷</sup> [۵۴]، ابتدا شباهت یا ارتباط بین پرسش<sup>۱۸</sup> و کلید<sup>۱۹</sup> را با محاسبه ضرب داخلی<sup>۲۰</sup> به دست می‌آوریم، سپس آن را نرمال می‌کنیم (با تقسیم بر  $d_k$ ) و از تابع سافت مکس<sup>۲۱</sup> استفاده می‌کنیم تا ضرایب توجه<sup>۲۲</sup> را به دست آوریم. در نهایت با همین ضرایب، ترکیبی خطی از بردارهای مقدار<sup>۲۳</sup> را می‌گیریم.

فرمول به شکل زیر است:

$$\text{Attention}(Q, K, V) = \text{softmax} \left( \frac{QK^T}{\sqrt{d_k}} \right) V \quad (۳.۲)$$

که در آن:

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

---

Scaled Dot-Product Attention<sup>۱۷</sup>

Query<sup>۱۸</sup>

Key<sup>۱۹</sup>

Dot Product<sup>۲۰</sup>

softmax<sup>۲۱</sup>

Attention Weights<sup>۲۲</sup>

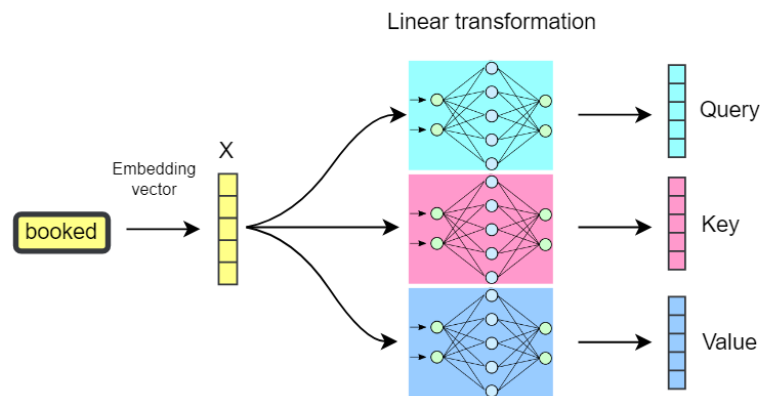
value<sup>۲۳</sup>

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 ۴.۲: نحوه به دست آوردن پرسش، کلید و مقدار

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.

$$\alpha = \text{softmax} \left( \frac{QK^T}{\sqrt{d_k}} \right) \quad (4.2)$$

$\alpha$  یک ماتریس با ابعاد  $n \times n$  است که سطر  $i$ -ام آن ضرایب توجه برای توکن  $i$  را نشان می‌دهد. تفسیر ضرایب توجه: هر سطر از  $\alpha$  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 ۵.۲: توجه

## ۴.۳.۲ توجه چند سر

در ایده چند سری<sup>۲۴</sup> به جای آنکه فقط یک بار  $Q, K, V$  بسازیم و عملیات توجه را انجام دهیم، چندین مجموعه متفاوت  $Q_i, K_i, V_i$  می‌سازیم (هر کدام یک «سر»<sup>۲۵</sup> نام دارد) و به صورت موازی محاسبات توجه را انجام می‌دهیم. سپس خروجی همه این سرها را کنار هم قرار داده<sup>۲۶</sup> و در نهایت با یک ماتریس وزن دیگر ضرب می‌کنیم تا به بعد اصلی بازگردیم.

فرمول مربوط به این ایده به شکل زیر است:

$$\text{head}_i = \text{Attention}(Q_i, K_i, V_i) \quad (۵.۲)$$

multi head attention<sup>۲۴</sup>

head<sup>۲۵</sup>

concatenate<sup>۲۶</sup>

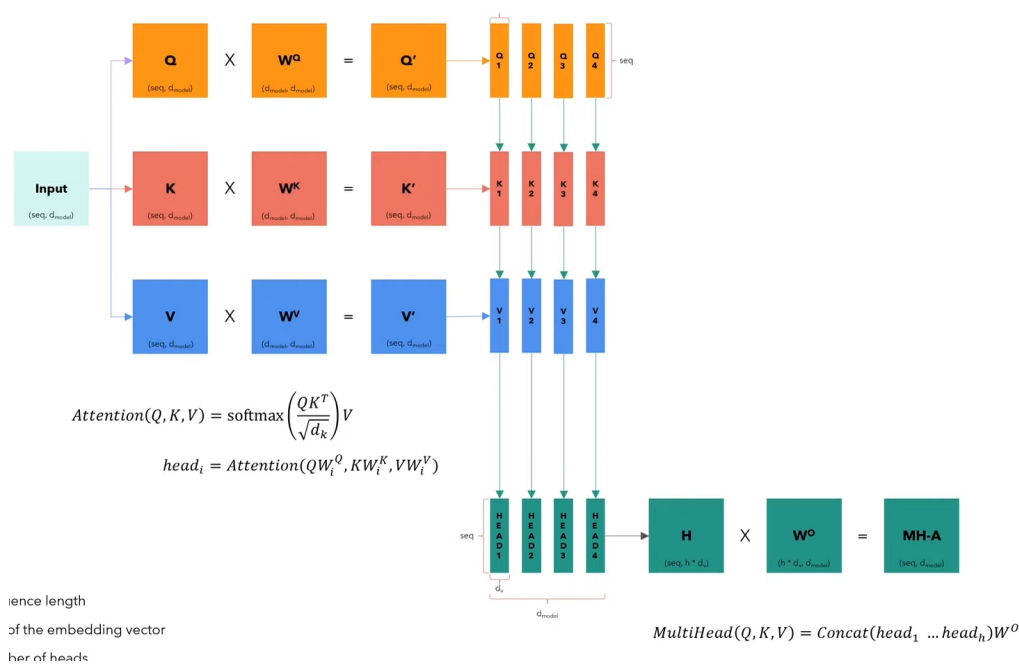
$$\text{MultiHead}(Q, K, V) = [\text{head}_1 \oplus \dots \oplus \text{head}_h] W_O \quad (۶.۲)$$

که در آن  $\oplus$  نشان‌دهنده عمل الحاق<sup>۲۷</sup> است.

ماتریس وزن  $W_O$  به شکل زیر است:

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

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



شکل ۶.۲: توجه چند سر

چرا چندین سر؟

مشاهده چند منظر متفاوت: هر سر می تواند الگوهای گوناگونی از وابستگی ها را بیاموزد (مثلاً یک سر می تواند یاد بگیرد کلمه فعلی با کلمات همسایه نزدیک خود بیشتر مرتبط شود، یک سر دیگر

<sup>۲۷</sup>concatenate

روی ارتباط با کلماتی در فاصله 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 سرها  $h = 8$ ، 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 [۲۱، ۳].

در مبدل‌ها [۵۴]، به جای این که خروجی توجه را به صورت مستقیم به لایه بعدی بدهیم، ورودی آن را نیز حفظ کرده و به خروجی اضافه می‌کنیم. ایده اصلی این روش از اتصالات باقی مانده<sup>۲۸</sup> در شبکه‌های عمیق الهام گرفته شده است [۱۸].

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

$$x + \text{SubLayer}(x) \quad (۷.۲)$$

این جمع به صورت عنصر به عنصر<sup>۲۹</sup> انجام می‌شود.

اتصال باقیمانده در مبدل‌ها چندین مزیت دارد که عبارت‌اند از:

<sup>۲۸</sup> Residual Connection

<sup>۲۹</sup> Element-wise Addition

## 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<sup>۳۰</sup> رخ دهد [۲۱، ۳].

## حفظ اطلاعات اصلی (هویت ورودی)

حتی اگر زیرماژول تغییری در اطلاعات ورودی ایجاد کند، با وجود اتصال باقیمانده<sup>۳۱</sup>، ورودی اصلی همواره در خروجی نهایی حضور دارد. این ویژگی باعث می‌شود در صورت ناکافی بودن یادگیری زیرماژول یا در مراحل اولیه آموزش، دست‌کم بخشی از سیگنال (اطلاعات) خام به لایه‌های بالاتر برسد [۱۸، ۵۴].

## 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 به صورت دست‌نخورده تا حدی منتقل می‌شود.

## ۴.۲ نرمال سازی لایه ها

در یادگیری عمیق، نرمال‌سازی<sup>۳۲</sup> داده‌های یک لایه یا فعال‌سازی‌ها، اغلب به سرعت بخشیدن به همگرایی و پایدار کردن آموزش کمک شایانی می‌کند. شاید معروف‌ترین نوع نرمال‌سازی، نرمال سازی بچ<sup>۳۳</sup> باشد که پیش‌تر در کارهای بینایی کانونلوشنی<sup>۳۴</sup> بسیار مورد استفاده قرار گرفت [۲۳].

<sup>۳۰</sup> gradient vanishing

<sup>۳۱</sup> Residual Connection

<sup>۳۲</sup> Normalization

<sup>۳۳</sup> Batch Normalization

<sup>۳۴</sup> CNN

نرمال سازی لایه ها <sup>۳۵</sup> روشی جایگزین است که در ترانسفورمر استفاده می شود [۱، ۵۴]. علت اصلی این انتخاب، ماهیت توالی محور <sup>۳۶</sup> بودن داده ها در پردازش زبان طبیعی و عدم تمایل به وابستگی به آمار مینی بچ <sup>۳۷</sup> است.

### نرمال سازی بچ

در نرمال سازی بچ ها، برای نرمال سازی، میانگین و واریانس روی تمام نمونه های موجود در مینی بچ (و نیز در طول ابعاد ویژگی) محاسبه می شود [۲۳]. این موضوع در پردازش زبان طبیعی کمی دردسرساز است؛ چون ترتیب توکن ها، طول جمله ها و حتی اندازه مینی بچ ممکن است نامنظم باشد. همچنین به خاطر تنوع طول توالی ها، پیاده سازی نرمال سازی بچ ها می تواند پیچیده شود.

### نرمال سازی لایه ها

در نرمال سازی لایه ها، برای هر توکن به صورت جداگانه (در طول بُعد ویژگی)، میانگین <sup>۳۸</sup> و واریانس <sup>۳۹</sup> گرفته می شود [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 می کنیم:

$$\mu_i = \frac{1}{d_{\text{model}}} \sum_{k=1}^{d_{\text{model}}} h_{i,k}, \quad \sigma_i^2 = \frac{1}{d_{\text{model}}} \sum_{k=1}^{d_{\text{model}}} (h_{i,k} - \mu_i)^2 \quad (۸.۲)$$

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

$$\hat{h}_{i,k} = \frac{h_{i,k} - \mu_i}{\sqrt{\sigma_i^2 + \epsilon}} \quad (۹.۲)$$

---

Layer Normalization<sup>۳۵</sup>

sequence<sup>۳۶</sup>

mini-batch<sup>۳۷</sup>

mean<sup>۳۸</sup>

variance<sup>۳۹</sup>

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

$$\text{LayerNorm}(h_i) = \gamma \odot \hat{h}_i + \beta \quad (10.2)$$

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

### مزایای نرمال سازی لایه در مبدل ها

- بی نیازی از وابستگی به ابعاد مینی بچ: با نرمال سازی لایه، می توان حتی با اندازه مینی بچ برابر ۱ نیز به خوبی آموزش دید، چرا که آمارها وابسته به ابعاد ویژگی اند و نه مینی بچ [۱].
  - پایداری توزیع فعال سازی ها: زمانی که مدل در حال یادگیری است، توزیع های داخلی لایه های میانی ممکن است تغییر کند.<sup>۴۰</sup> نرمال سازی لایه با نرمال سازی این توزیع، آموزش را پایدارتر و سریع تر می کند [۱، ۲۳].
  - سازگاری با داده های توالی محور: هر توکن را جداگانه نرمال می کند و نگرانی ای بابت ترتیب طول جمله ها، یا قرار گرفتن چند جمله کوتاه/بلند در یک مینی بچ نداریم [۵۴].
- در معماری مبدل ها، پس از خروجی هر زیرماژول، مراحل به شکل زیر است:

### ۱.۴.۲ اتصال باقی مانده

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

$$z = x + \text{SubLayer}(x) \quad (11.2)$$

---

<sup>۴۰</sup> Internal Covariate Shift



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

نرمال سازی لایه سپس این بردار  $z$  را وارد لایه LayerNorm می‌کنیم.

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

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

به عبارتی اگر بخواهیم در یک فرمول واحد بیان کنیم:

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

## ۵.۲ رمزگشا

رمزگشا در معماری ترانسفورمرها وظیفه تولید خروجی نهایی را بر عهده دارد. این خروجی معمولاً می‌تواند توالی هدف<sup>۴۱</sup> باشد، مانند ترجمه یک جمله یا پیش‌بینی توکن‌های بعدی در یک توالی [۵۴].

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

رمزگشا از این ورودی‌ها استفاده می‌کند تا به صورت گام به گام، خروجی نهایی خود را تولید کند

[۵۱، ۲].

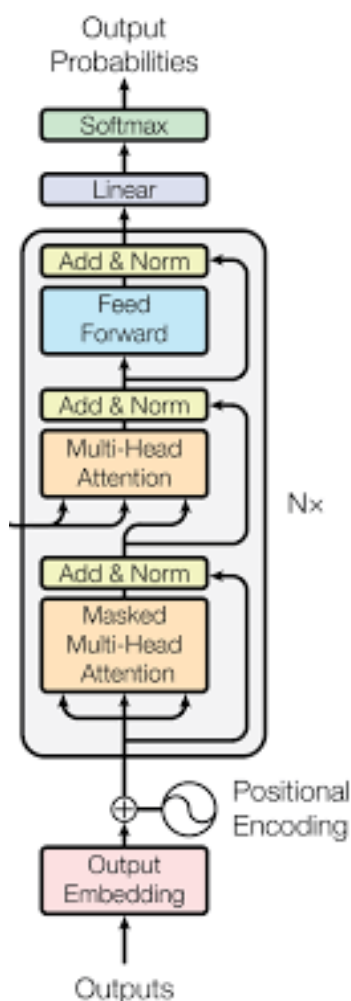
همان‌طور که در شکل ۷.۲ مشاهده می‌کنید، رمزگشا دو ورودی دارد.

تمامی بخش‌های رمزگشا مانند رمزگذار هستند اما در دیکودر توجه چند سر ماسک شده<sup>۴۲</sup> وجود

دارد [۵۴].

<sup>۴۱</sup> Target Sequence

<sup>۴۲</sup> Masked Multi-Head Attention



شکل ۷.۲: رمزگشا در مبدل ها

## ۶.۲ توجه چند سری ماسک شده

در مبدل ها، مکانیزم توجه چند سری <sup>۴۳</sup> در بخش دیکودر به صورت ماسک شده <sup>۴۴</sup> پیاده سازی می شود تا مدل نتواند توکن های آینده را ببیند و به صورت خودبازگشتی <sup>۴۵</sup> توکن بعدی را پیش بینی کند [۵۴]. در واقع ایده اصلی استفاده از ماسک جلوگیری از مشاهده آینده است.

در معماری های خودبازگشتی، مدل در گام  $i$  از دیکودر تنها باید به توکن های قبلی  $\{y_1, \dots, y_{i-1}\}$

<sup>۴۳</sup> Multi-Head Attention

<sup>۴۴</sup> Masked

<sup>۴۵</sup> 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، ۵۱].

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 دنباله 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 نشانه گذاری پایین مثلثی):

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

به این ترتیب، پس از جمع شدن با  $S$  و اجرای softmax در هر سطر، ضرایب توجه ستون‌های ماسک شده به صفر میل می‌کنند [۵۴].

## ۸.۲ مبدل های بینایی

ایده ترانسفورمرها در حوزه بینایی<sup>۴۶</sup> از تعمیم ترانسفورمر متن به تصاویر به وجود آمده است [۱۱]. ما در این بخش از مبدل های بینایی برای وظیفه کلاس بندی استفاده می‌کنیم. در روش های متداول برای پردازش تصویر، از کانولشن<sup>۴۷</sup> های متوالی استفاده می‌کردند؛ اما در ترانسفورمرها تصاویر به پچ‌های مختلف شکسته می‌شوند [۱۱]. هر پچ شکسته شده از تصویر می‌تواند با سایر پچ‌ها به صورت موازی وارد مکانیزم توجه شود و شباهت یا ارتباطشان با یکدیگر سنجیده شود. در بخش های بعد، به طور 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 [۵۴]. حال همین ایده در تصویر پیاده سازی شده است. همان طور که در شکل ۸.۲ مشاهده می‌کنید، در مبدل های بینایی، به جای استفاده از عملیات کانولشن های متوالی که در شبکه های پیچشی<sup>۴۸</sup>

<sup>۴۶</sup> vision transformer

<sup>۴۷</sup> convolution

<sup>۴۸</sup> convolutional neural network

مرسوم است [۳۲، ۳۰، ۱۸]، 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 ۸.۲: بخش بندی تصاویر

## ۲.۸.۲ شکل پیچ‌ها:

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

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

حال اگر اندازه هر پیچ ( $P \times P$ ) باشد (برای نمونه  $16 \times 16$ )، تصویر به صورت یک جدول مشبک از پیچ‌های کوچک تقسیم می‌شود. به هر پیچ می‌توان مانند یک «کاشی» از تصویر نگاه کرد: - پیچ اول: مختصات ( در ارتفاع ۱۵ تا ۰) و ( در عرض ۱۵ تا ۰)، - پیچ دوم: مختصات ( در ارتفاع ۱۵ تا ۰) و ( در عرض ۳۱ تا ۱۶)، - و به همین ترتیب تا کل تصویر پوشش داده شود.

## ۳.۸.۲ تعداد پیچ‌ها:

اگر پیچ‌ها بدون هم‌پوشانی باشند، ابعاد پیچ باید بر ابعاد تصویر بخش پذیر باشد.

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

در مجموع:

$$\left(\frac{H}{P}\right) \times \left(\frac{W}{P}\right) = \frac{H}{P} \times \frac{W}{P}. \quad (۱۲.۲)$$

برای مثال اگر:

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

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

در اکثر نسخه‌های مبدل‌های بینایی، پچ‌ها بدون هم‌پوشانی<sup>۴۹</sup> 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<sup>۵۰</sup> 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 طبیعی به مبدل‌ها بدهیم، باید آن‌ها را به یک بردار یک بعدی تبدیل کنیم. در صورت قرار دادن پیکسل‌های پچ به صورت ردیفی<sup>۵۱</sup>، طول این بردار خواهد بود:

$$P \times P \times C = P^2 \times C. \quad (۱۳.۲)$$

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

---

<sup>۴۹</sup>Non-overlapping

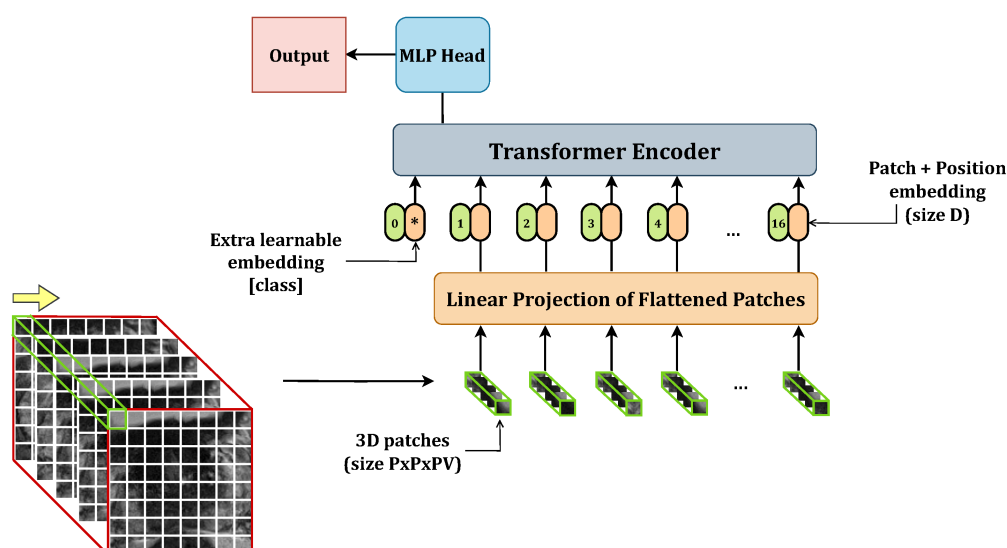
<sup>۵۰</sup>Local Details

<sup>۵۱</sup>Row-major

## ۹.۲ اعمال لایه خطی

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

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



شکل ۹.۲: مدل‌های 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 [۵۴]. در مدل‌های 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،

<sup>۵۲</sup> Flatten

<sup>۵۳</sup> Fully-Connected Layer

<sup>۵۴</sup> Feature Transformation

در مبدل های بینایی توالی «پچ» ها:

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

هر پچ اکنون یک بردار  $d_{\text{model}}$  - بعدی است. پس یک مجموعه با طول  $N$  (تعداد پچ ها) و عرض  $d_{\text{model}}$  خواهیم داشت. اگر عدد پچ ها  $N$  باشد (مثلاً ۱۹۶)، ترانسفورمر می تواند با مکانیزم توجه خود سر، وابستگی میان پچ ها را یاد بگیرد: کدام بخش از تصویر برای کدام بخش دیگر مهم تر است [۵۴، ۱۱].

معمولاً پچ ها را به صورت ردیفی شماره گذاری می کنند (ابتدا پچ های ردیف بالایی از چپ به راست، سپس ردیف بعدی و ...)، تا مدل در صورت نیاز بتواند از موقعیت ها، 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 می شود.

در مبدل بینایی [۱۱] دیگر به کانولوشن وابسته نیستیم. در عوض، از جاسازی استفاده می شود. تقسیم کردن تصویر به بلاک های  $(P \times P)$ ، کنار هم چیدن و تبدیل آن به جاساز همگی عملیات ریاضی ساده ای هستند که به راحتی روی TPU/GPU قابل موازی سازی اند.

## ۱.۹.۲ توکن کلاس بندی

توکن کلاس بندی <sup>۵۵</sup> یک بردار ویژه است که به ابتدای دنباله ورودی اضافه می شود و نقش آن، خلاصه کردن اطلاعات کل ورودی (چه متن، چه تصویر) است [۹، ۱۱].

در مبدل بینایی، این توکن در ابتدای پچ های تصویری قرار می گیرد. این توکن یک بردار با ابعاد  $d_{\text{model}}$  است (همان ابعاد سایر توکن ها) و پارامتری یادگرفته محسوب می شود؛ یعنی مدل طی آموزش، مقادیر آن را برای ذخیره و تجمیع اطلاعات بهینه می کند.

در وظایف دسته بندی کلاس بندی، هدف این است که یک پیش بینی کلی برای کل ورودی (مثلاً یک جمله یا یک تصویر) ارائه دهیم؛ توکن کلاس بندی دقیقاً همین وظیفه را بر عهده دارد [۹]. این

---

<sup>۵۵</sup> Cls Token



توکن از طریق مکانیزم توجه چند سر در مبدل ها با تمامی توکن های دیگر (پچ های تصویر) ارتباط می گیرد و اطلاعات مهم آن ها را در لایه های مختلف مبدل ها را به صورت تجمعی یاد می گیرد. 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، می تواند به تمام پچ ها نگاه کند و با ضرایب توجه ( $\alpha$ ) مشخص کند که از هر پچ چه مقدار اطلاعات بگیرد. بدین ترتیب، به طور ضمنی یاد می گیرد روی ویژگی هایی که برای دسته بندی مهم هستند (نظیر الگوها، اشکال و بخش های کلیدی تصویر) متمرکز شود.

در طول لایه های ترانسفورمر، توکن کلاس بندی نقش محوری در خلاصه سازی بازنمایی کل تصویر ایفا می کند. این توکن به صورت پارامتر قابل یادگیری تعریف شده و در طول فرآیند آموزش به روزرسانی می شود [۹، ۱۱].

## ۲.۹.۲ رمزگذار در مبدل های بینایی

رمزگذار در ترانسفورمرها همانند مبدل اصلی است [۵۴]، با این تفاوت که در مبدل های بینایی [۱۱] دیگر به رمزگشا نمی رویم. پس از عبور از بلاک های ترانسفورمر، در ساده ترین حالت یک لایه خطی<sup>۵۶</sup> یا یک لایه MLP<sup>۵۷</sup> بر روی بردار نهایی اعمال می شود و این لایه ها به تعداد کلاس ها خروجی می دهند. سپس خروجی هر لایه با گذر از تابع سافت مکس<sup>۵۸</sup> به احتمال هر کلاس تبدیل می شود و در نهایت مدل کلاس با بیشترین احتمال را به عنوان خروجی پیش بینی می کند.

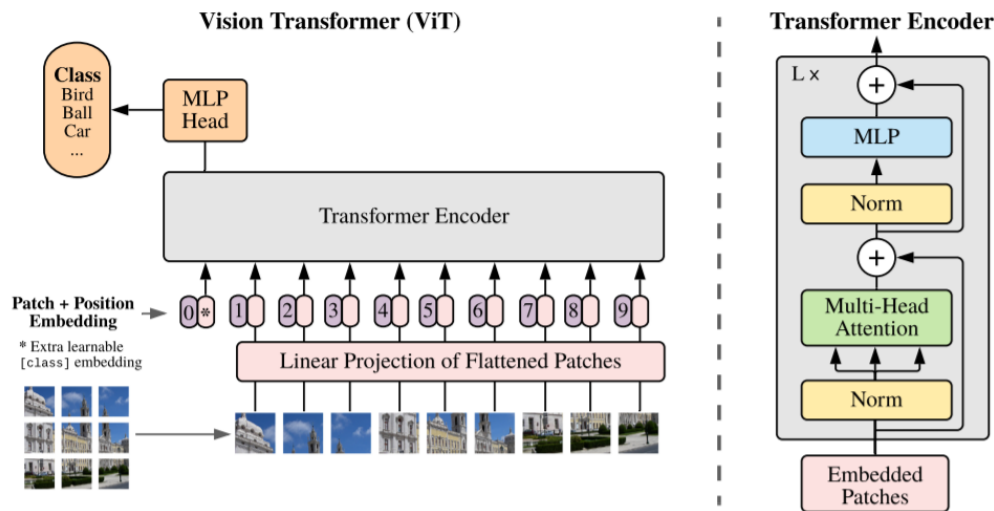
در مبدل ها، هر لایه رمزگشا و رمزگذار با پردازش عمیق تر روی توالی ورودی، می تواند نمایش بهتری از ویژگی ها به دست بیاورد [۵۴]. تکرار چندین باره رمزگشا یا رمزگذار موجب می شود مدل بتواند ساختارهای پیچیده ای را یاد بگیرد و کیفیت و دقت آن در شناسایی توالی های طولانی و معانی پنهان افزایش یابد [۱۱، ۵۴]. در نتیجه، مدل با تعداد لایه های بیشتر اغلب عملکرد بهتری از خود

---

Fully Connected<sup>۵۶</sup>

Multi-Layer Perceptron<sup>۵۷</sup>

softmax<sup>۵۸</sup>



شکل ۱۰.۲: توکن توجه در مبدل های بینایی

نشان می دهد.

## ۱۰.۲ مبدل پنجره ای متحرک

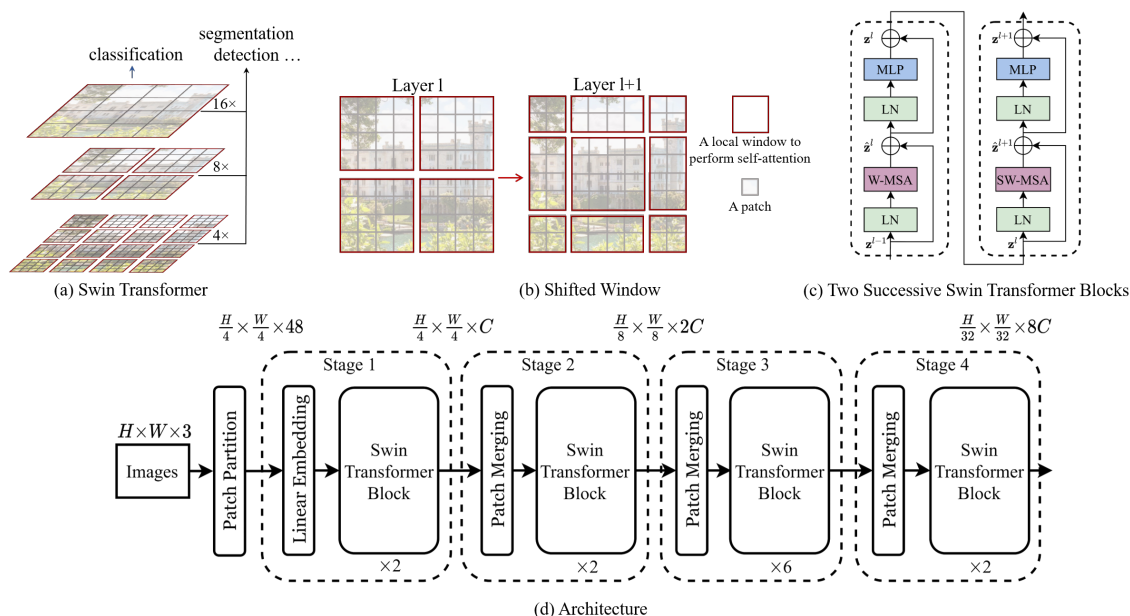
ایده مبدل پنجره ای متحرک <sup>۵۹</sup> از ترکیب چند مفهوم کلیدی در مدل های ترانسفورمر و شبکه های پیچشی شکل گرفت [۳۴، ۱۸، ۵۴].

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 پیچ به تمامی پیچ های دیگر توجه می کرد و در مواقعی که تعداد پیچ ها زیاد می شد، هزینه محاسباتی و حافظه به شدت افزایش پیدا می کرد.

در شبکه های پیچشی، معماری معمولاً به صورت سلسله مراتبی پیش می رود [۱۸]؛ یعنی ابتدا ویژگی های محلی استخراج می شود، سپس با عمیق تر شدن لایه ها، این ویژگی ها در سطوح بالاتر با یکدیگر ترکیب می شوند. در مبدل پنجره ای متحرک [۳۴]، با دانش بر این موضوع توانسته اند هم

<sup>۵۹</sup> Swin Transformer

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



شکل ۱۱.۲: مبدل پنجره متحرک

در مبدل پنجره‌ای متحرک، به جای آن‌که مدل به تمام پچ‌ها در یک سطح ویژگی نگاه کند، تصویر را به «پنجره‌های محلی»<sup>۶۰</sup> تقسیم می‌کند و توجه را محدود به همان ناحیه می‌سازد [۲۴]. سپس با تکنیک جابه‌جایی<sup>۶۱</sup> 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، ایده کلیدی‌ای بود که باعث شد مدل هم محاسبات سبک‌تری داشته باشد و هم بتواند ارتباط‌های جهانی<sup>۶۲</sup> را در طول لایه‌ها به دست آورد.

یکی دیگر از ایده‌های مهم در در مبدل پنجره‌ای متحرک، کوچک کردن تدریجی نقشه ویژگی در طول معماری است؛ مشابه کاری که در ResNet یا سایر CNN‌ها انجام می‌شود [۱۸]. این امر ضمن کاهش هزینه محاسباتی، باعث می‌شود مدل بتواند با سطوح مختلفی از ویژگی‌ها کار کند و

Local Windows<sup>۶۰</sup>

Shift<sup>۶۱</sup>

Global<sup>۶۲</sup>

در نهایت خروجی نهایی با کیفیت تری 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  $P$  اندازه پچ<sup>۶۳</sup> باشد، آنگاه تعداد پچ‌ها در بعد افقی و عمودی، به ترتیب  $\frac{H}{P}$  و  $\frac{W}{P}$  خواهد بود. هر پچ را می‌توان به صورت 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 به دست می‌آید:

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

برخلاف مبدل بینایی اصلی این کاردر مبدل پنجره متحرک با استفاده از کانولوشن<sup>۶۴</sup> انجام می‌شود. در واقع سائز کرنل در کانولوشن همان فضای برداری هر پچ هست که ما فرض میکنیم سائز کرنل برابر  $C$  است. پس در نتیجه

$$Z = X \cdot W_{\text{embed}} + b_{\text{embed}}, \quad Z \in \mathbb{R}^{\left(\frac{H}{P} \cdot \frac{W}{P}\right) \times C}. \quad (14.2)$$

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

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

---

<sup>۶۳</sup> patch size  
<sup>۶۴</sup> convolution

پس از این مرحله، ما در هر موقعیت  $(h, w)$  (از شبکه پچها) یک بردار  $z_{h,w} \in \mathbb{R}^C$  داریم. این ماتریس  $Z$  ورودی اولین مرحله (Stage) از مبدل های پنجره متحرک خواهد بود [۳۴]. هر بلوک مبدل پنجره متحرک از چند بخش اصلی تشکیل شده است [۳۴]:

- پنجره بندی تصویر<sup>۶۵</sup> یا پنجره بندی جابه جاشده<sup>۶۶</sup>

- اعمال توجه چمد سر پنجره ای<sup>۶۷</sup>

- لایه Skip Connection<sup>۶۸</sup> و Layer Norm<sup>۶۹</sup>

- مسیر پرسپترون چندلایه<sup>۷۰</sup>:

## ۲.۱۰.۲ توجه چند سر پنجره ای

تعریف پنجره های محلی

در مبدل های پنجره متحرک، به جای آنکه تمام پیکسل های یک نقشه ویژگی بزرگ را یک جا در محاسبه توجه درگیر کنیم، نقشه ویژگی را به قطعه های کوچکی به اندازه  $(M \times M)$  تقسیم می کنیم. این قطعه های کوچک را «پنجره های محلی» می نامیم.

اگر اندازه نقشه ویژگی در یک لایه  $(H' \times W')$  باشد، با تقسیم آن به پنجره های  $(M \times M)$ ، در راستای طول تقریباً  $\frac{H'}{M}$  پنجره خواهیم داشت و در راستای عرض هم  $\frac{W'}{M}$  پنجره. (برای راحتی، فرض می کنیم  $H'$  و  $W'$  دقیقاً مضربی از  $M$  باشند تا تقسیم بدون باقی مانده انجام شود). هر کدام از این پنجره های  $(M \times M)$  دارای  $M^2$  پیکسل (یا موقعیت مکانی) است، و در هر پیکسل هم یک بردار ویژگی با بعد  $C$  قرار دارد.

---

<sup>۶۵</sup>Window Partition

<sup>۶۶</sup>Shifted Window Partition

<sup>۶۷</sup>Window Multi-Head Self Attention

<sup>۶۸</sup>Skip Connection

<sup>۶۹</sup>Layer Norm

<sup>۷۰</sup>MLP

به بیان ساده‌تر:

- نقشه ویرگی مثل یک صفحه 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 محاسبه توجه هستند، به مراتب کمتر شود و هزینه 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 به صورت  $\frac{C}{h}$  در نظر گرفته می‌شود که در آن  $h$  تعداد سربندی سرها است. در توجه چند سر، خروجی نهایی با ترکیب  $h$  سر توجه محاسبه می‌شود. در یک سر توجه، توجه به صورت زیر تعریف می‌شود:

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

که در آن:

•  $Q, K, V$  به ترتیب 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 بیش از حد ضرب داخلی است.

در مبدل‌های پنجره متحرک، این محاسبات به صورت پنجره‌ای انجام می‌شوند؛ یعنی برای هر پنجره، تنها پیکسل‌های داخل همان پنجره در ماتریس‌های  $Q, K, V$  لحاظ می‌شوند. به این ترتیب، زمان محاسبه و مصرف حافظه به شدت کاهش می‌یابد (در مقایسه با مبدل‌های بینایی که همه چیز را با هم مقایسه می‌کند).

تعداد سربندی  $h$  معمولاً طوری انتخاب می‌شود که  $C = h \times d$  خروجی هر سر پس از محاسبه توجه به صورت زیر با هم ادغام می‌شوند:

$$\text{MultiHead}(Q, K, V) = [\text{head}_1, \text{head}_2, \dots, \text{head}_h] W_O,$$

که

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

ماتریس ترکیب نهایی است.

## ۴.۱۰.۲ پنجره متحرک جا به جا شده

در مبدل‌های پنجره متحرک، ایده «پنجره‌های جابه‌جاشده»<sup>۷۱</sup> به این منظور ارائه شده است تا مدل، ارتباط پیکسل‌های واقع در پنجره‌های مجاور را هم یاد بگیرد [۳۴]. 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 به خوبی تبادل نشود.

<sup>۷۱</sup> Shifted Windows

روش مبدل های پنجره متحرک برای رفع این محدودیت از یک تکنیک ساده اما مؤثر استفاده می کند [۳۴]:

- در یک لایه، محاسبات توجه در پنجره های محلی ثابت انجام می شود.
- در لایه بعدی، پنجره ها به اندازه ای مشخص جابه جا می شوند (به صورت شیفت افقی و عمودی) تا نواحی مرزی نیز در محاسبات گنجانده شوند.
- این فرآیند باعث می شود که پیکسل ها در پنجره های 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<sup>۷۲</sup>، نقشه ویژگی به پنجره های  $(M \times M)$  تقسیم می شود [۳۴]. هیچ جابه جایی در این تقسیم بندی وجود ندارد؛ یعنی اگر نقشه ویژگی را یک مستطیل بزرگ در نظر بگیریم، آن را شبیه کاشی کاری یا شطرنج بندی به بلوک های مربعی  $(M \times M)$  برش می زنیم. در این حالت، پیکسل های هر پنجره فقط با همدیگر (درون همان پنجره) ارتباط برقرار می کنند.

توجه چند سری پنجره ای جا به جا شده

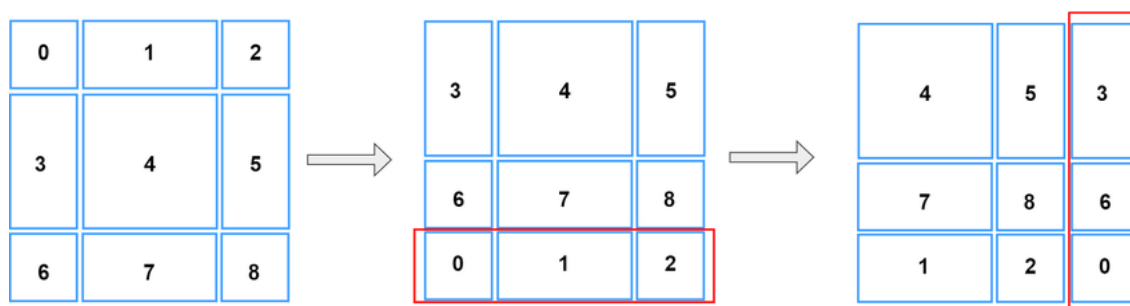
مطابق شکل ۱۲.۲، بعد از اینکه بلوک اول (توجه چند سری پنجره ای) کارش تمام شد، در بلوک دوم، قبل از تقسیم بندی به پنجره های  $(M \times M)$ ، نقشه ویژگی را جابه جا می کنیم [۳۴]. در مقاله اصلی، این مقدار جابه جایی معمولاً نیم اندازه پنجره  $\frac{M}{2}$  در راستای افقی و عمودی است. به این ترتیب:

- پیکسل هایی که پیش از این در دو پنجره جداگانه قرار داشتند، ممکن است حالا به دلیل جابه جایی وارد یک پنجره مشترک شوند.



● مدل حالا می‌تواند بین این پیکسل‌های «مرزی» نیز توجه برقرار کند و اطلاعات را بهتر مبادله کند.

با این جابه‌جایی، بخشی از پیکسل‌ها در نقشه ویژگی از یک طرف «خارج» می‌شوند. برای اینکه این پیکسل‌ها را از دست ندهیم، از ترفندی به نام جابه‌جایی چرخه‌ای<sup>۷۳</sup> استفاده می‌شود. در جا به جایی چرخه‌ای، پیکسل‌هایی که از سمت راست بیرون می‌روند دوباره از سمت چپ وارد می‌شوند و بالعکس؛ درست شبیه وقتی که یک تصویر را به صورت حلقه‌ای اسکرول می‌کنیم<sup>۷۴</sup>. مثالی از جا به جایی چرخه‌ای در شکل ۱۲.۲ آمده است.



شکل ۱۲.۲: جا به جایی چرخه‌ای

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

در بلوک دوم (جابه‌جاشده)، مرزهای پنجره‌ها تغییر می‌کند و برخی پیکسل‌هایی که قبلاً در پنجره‌های جدا بودند، اکنون در یک پنجره 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 هزینه محاسباتی کمتر (نسبت به توجه سراسری کامل) در اختیار داشته باشد [۳۴].

Cyclic Shift<sup>۷۳</sup>  
Wrap around<sup>۷۴</sup>

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-چپ یا پایین-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، سری اول و دوم بلوک‌های مبدل‌های پنجره متحرک تکمیل‌کننده یکدیگر می‌شوند [۳۴]:

• بلوک اول: محاسبه توجه در چهارچوب پنجره‌های ثابت.

• بلوک دوم: محاسبه توجه در پنجره‌های جابه‌جاشده که منجر به تعامل بیشتر بین مرزهای مختلف می‌شود.

## ۵.۱۰.۲ پرسپترون چند لایه

پس از انجام توجه چند سری پنجره ای جا به جا شده خروجی به یک مسیر MLP می‌رود [۳۴]. ساختار این MLP به‌صورت زیر است:

$$X' = \text{GELU}(XW_1 + b_1) W_2 + b_2, \quad (15.2)$$

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 این جا قابل استفاده است [۱۹].

## ۶.۱۰.۲ ترکیب پچ ها

در مدل مبدل های پنجره متحرک، ساختار سلسله مراتبی به این معناست که ما در چند مرحله (Stage) مختلف، نقشه ویزگی را کوچک تر می کنیم و در عین حال، عمق (تعداد کانال های ویزگی) را افزایش می دهیم. هدف اصلی از این کار عبارت است از:

- استخراج ویزگی های سطح بالاتر: وقتی نقشه ویزگی کوچک تر می شود، هر واحد از نقشه ویزگی بیانگر بخش گسترده تری از تصویر اصلی است؛ پس مدل به تدریج جزئیات محلی را با درک کلی تری از تصویر جایگزین می کند [۱۸].

- کاهش هزینه محاسبات: در مراحل بعدی، چون ابعاد فضایی کمتر می شود، مدل راحت تر می تواند با ویزگی های جدید کار کند (چون مثلاً به جای  $(H \times W)$  پیکسل، تعداد کمتری پیکسل داریم) [۳۴].

پس از چندین بلوک پردازشی، نقشه ویزگی، ابعادی به شکل  $(\frac{H}{P}, \frac{W}{P})$  با تعداد کانال  $C$  دارد. این یعنی پس از برش دادن تصویر به پچ ها و گذر از چند لایه، اکنون یک نقشه ویزگی داریم که کوچک تر از تصویر اصلی است، اما هنوز ممکن است خیلی بزرگ باشد.

در مرحله بعد (Stage بعدی)، می خواهیم 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 ویزگی های پیچیده تر بیشتر شود). برای انجام این کار از فرایندی به نام ترکیب پچ ها استفاده می کنیم. [۳۴]:

۱. انتخاب بلوک های  $(2 \times 2)$ 

ابتدا نقشه ویزگی را در بُعد مکانی به بلوک های  $(2 \times 2)$  تقسیم می کنیم. اگر  $Z_{i,j}$  ویزگی 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 بُعد  $C$  باشد، اکنون بُعد حاصل از کنار هم گذاشتن این چهار پیکسل می‌شود  $4C$ . نام این بردار ادغام‌شده را  $Z'$  می‌گذاریم.

## ۳. لایه خطی برای تغییر بُعد

وقتی چهار بردار  $C$  - بعدی را کنار هم می‌گذاریم، یک بردار  $4C$  - بعدی شکل می‌گیرد. حال با یک لایه خطی، بُعد  $4C$  را به بُعد جدیدی تبدیل می‌کنیم. معمولاً این بُعد جدید برابر  $2C$  در نظر گرفته می‌شود؛ یعنی دو برابر بزرگ‌تر از قبل اما نه چهار برابر:

$$Z' \mapsto Z'' = Z' W_{\text{merge}} + b_{\text{merge}}, \quad (۱۶.۲)$$

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

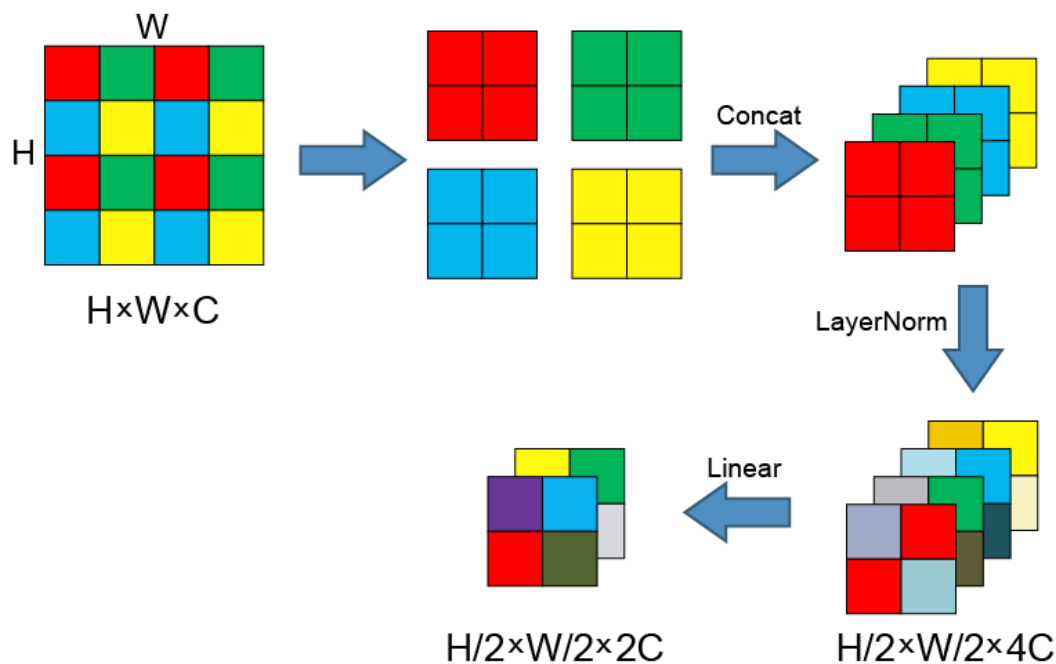
## ۴. کاهش ابعاد مکانی

در عین حال، وقتی هر چهار پیکسل  $(2 \times 2)$  را ادغام می‌کنیم، نقشه ویژگی ما ابعاد فضایی  $(\frac{H}{2P} \times \frac{W}{2P})$  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، مرتباً از لایه‌های ادغام<sup>۷۵</sup> یا کانولوشن با گام<sup>۷۶</sup> برای کوچک کردن ابعاد استفاده می‌شود تا اطلاعات سطح بالاتر (مثل ساختار کلی اشیا) راحت‌تر استخراج شود [۱۸]. در مبدل پنجره متحرک هم همین ایده سلسله‌مراتب را به دنیای مبدل‌ها آورده است [۳۴]. همچنین اگر ابعاد فضایی را کم نکنیم، هزینه توجه به شدت زیاد می‌شود (چون باید در هر لایه برای همه

Pooling<sup>۷۵</sup>Stride-Convolution<sup>۷۶</sup>



شکل ۱۳.۲: ادغام پیچ ها

پیکسل‌ها توجه محاسبه گردد).

در معماری کلی کبدل‌های 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 پیچ ها انجام می‌شود. سپس در Stage 2، ویژگی‌های کوچک‌تری داریم، اما تعداد کانال‌ها افزایش یافته است [۳۴]. مشابه معماری‌های کانولوشنی، با افزایش عمق<sup>۷۷</sup>، ابعاد فضایی کاهش و تعداد کانال‌ها افزایش پیدا می‌کند.

در انتهای Stage آخر، خروجی به یک لایه 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.

## فصل ۳

### روش های پیشنهادی

استفاده از روش های تانسوری در شبکه های عصبی چندلایه

در سال های اخیر، استفاده از روش های تانسوری به عنوان روشی نوین در بهینه سازی معماری های شبکه های عصبی، به ویژه در مدل هایی که تعداد پارامترهای آنها بسیار زیاد است، مانند شبکه های عصبی چندلایه<sup>۱</sup>، توجه بسیاری را به خود جلب کرده است [۴۶]. تانسورها تعمیمی از ماتریس ها به ابعاد بالاتر هستند و به طور طبیعی برای نمایش داده های چندبعدی همچون تصاویر، ویدیوها یا سری های زمانی چندکاناله مناسب اند. بهره گیری از ساختار تانسوری در معماری شبکه، این امکان را فراهم می سازد که بدون نیاز به فشردن سازی اولیه (مانند صاف کردن<sup>۲</sup> کردن ورودی)، اطلاعات ساختاری میان ابعاد مختلف حفظ شده و مدل بتواند از روابط درون تعاملی موجود میان این ابعاد بهره برداری نماید [۴۶].

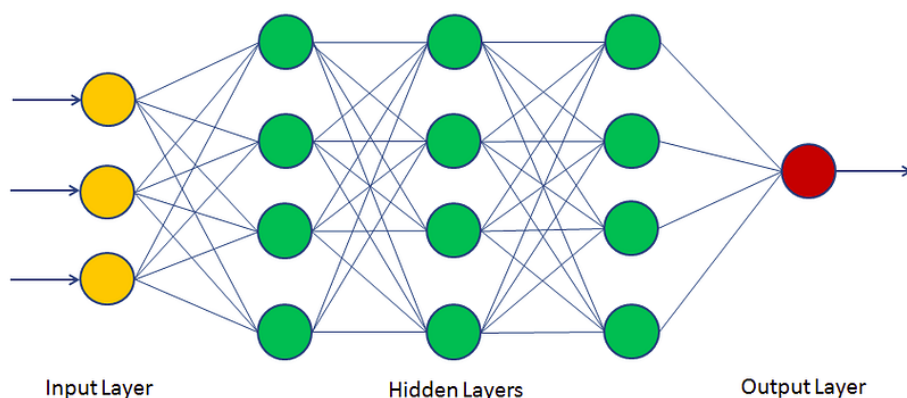
در معماری سنتی شبکه عصبی چند لایه، نگاشت از ورودی  $x \in \mathbb{R}^n$  به خروجی  $y \in \mathbb{R}^m$  به وسیله ضرب ماتریسی انجام می گیرد:

$$y = Wx + b$$

---

mlp<sup>۱</sup>  
flatten<sup>۲</sup>

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



شکل ۱.۳: شبکه عصبی پرسپترون چند لایه

در مقابل، در روش های تانسوری، وزن ها به صورت یک تانسور مرتبه بالاتر مدل سازی می شوند و نگاشت ورودی به خروجی با استفاده از ضرب های چندحالتی (ضرب تانسوری در ابعاد مختلف) صورت می پذیرد:

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

در این رابطه،  $\mathcal{W}$  یک تانسور وزن است و عملگر  $\times_n$  نشان دهنده ضرب تانسوری در بعد  $n$ ام می باشد.

روش هایی نظیر <sup>۳</sup> TCL و <sup>۴</sup> TRL به عنوان نمونه هایی از این رویکرد، با بهره گیری از تکنیک های تجزیه تانسوری همچون Tucker یا تجزیه CP، نه تنها باعث کاهش چشمگیر در تعداد پارامترها می شوند، بلکه ساختار چندبعدی داده ها را نیز حفظ می نمایند [۲۸، ۲۹].

---

<sup>۳</sup> layer contraction tensor  
<sup>۴</sup> layer regression Tensor

## مزایای استفاده از روش های تانسوری

استفاده از روش های تانسوری در شبکه های عصبی چندلایه مزایای متعددی به همراه دارد که برخی از مهم ترین آن ها عبارت اند از:

- کاهش چشمگیر تعداد پارامترها: با بهره گیری از فشرده سازی تانسوری، می توان ابعاد تانسور وزن ها را به گونه ای کاهش داد که بدون افت محسوس در عملکرد مدل، مصرف حافظه و پیچیدگی محاسباتی به طور قابل توجهی کاهش یابد [۲۹، ۳۱].
- حفظ ساختار داده های ورودی: برخلاف روش های سنتی که در آن ها داده ها پیش از ورود به لایه های چگال<sup>۵</sup> باید مسطح سازی<sup>۶</sup> شوند، استفاده از ساختار تانسوری این امکان را فراهم می کند که ساختار فضایی، زمانی یا کانالی داده ها حفظ شده و ارتباط میان ابعاد مختلف ورودی بهتر درک و پردازش شود [۲۹].

## محدودیت ها و چالش ها

با وجود مزایای متعدد، بهره گیری از روش های تانسوری در معماری های شبکه های عصبی با چالش ها و محدودیت هایی نیز همراه است که در ادامه به برخی از مهم ترین آن ها اشاره می شود:

- پیچیدگی بالاتر در پیاده سازی: پیاده سازی لایه های مبتنی بر عملیات تانسوری معمولاً به ابزارها و کتابخانه های خاصی همچون Tensorly یا Tensor Toolbox نیاز دارد. این موضوع فرآیند طراحی و توسعه مدل را پیچیده تر از استفاده از لایه های استاندارد مانند شبکه عصبی چند لایه می سازد [۲۸].
- بهینه سازی دشوارتر: فرآیند آموزش مدل های تانسوری می تواند نسبت به مدل های معمولی کندتر باشد. الگوریتم های مبتنی بر گرادیان ممکن است در فضای پارامتری تانسورها با

---

Dense<sup>۵</sup>  
Flatten<sup>۶</sup>



سطوح خطای غیرهموار یا چندوجهی مواجه شوند که روند همگرایی را دشوار می‌کند [۵۵].

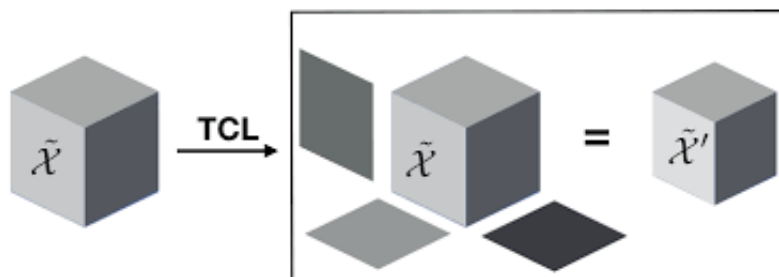
- احتمال کاهش دقت در فشرده‌سازی شدید: در صورتی که میزان فشرده‌سازی تانسورها بیش از حد بالا باشد، مدل ممکن است توانایی لازم برای نمایش روابط غیرخطی و الگوهای پیچیده را از دست داده و در نتیجه، دقت نهایی پیش‌بینی کاهش یابد [۲۹].

### ۱.۰.۳ لایه فشرده‌سازی تانسوری

#### Tensor Contraction Layer

در بسیاری از مدل‌های یادگیری عمیق — به‌ویژه شبکه‌های عصبی کانولوشنی (convolution) — فعال‌سازی‌های لایه‌های میانی به‌شکل تانسورهایی با مرتبه بالا ظاهر می‌شوند. به‌طور سنتی، برای اعمال لایه‌های کاملاً متصل (Fully Connected)، این تانسورها ابتدا با عملیات مسطح‌سازی (flatten) به بردار تبدیل و سپس به فضای خروجی نگاشت می‌شوند؛ روشی که ساختار چندخطی (multilinear structure) داده را از بین برده و به‌تبع، تعداد پارامترها افسارگسیخته افزایش می‌یابد [۲۹].

در پاسخ به این چالش، لایه‌ای با عنوان \*\*لایه فشرده‌سازی تانسوری\*\* یا به اختصار TCL معرفی شده است. این لایه بدون نیاز به مسطح‌سازی، هر مد (mode) از تانسور ورودی را مستقل فشرده‌سازی می‌کند و ساختار چندخطی داده را حفظ می‌نماید [۲۸، ۲۹].



شکل ۲.۳: لایه فشرده ساز تانسوری

فرمول بندی ریاضی

فرض کنید تانسور فعال سازی ورودی:

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

باشد (که  $S$  اندازه دسته بندی و  $I_k$  ها ابعاد کانال/فضایی را نشان می دهند). هدف لایه TCL، نگاشت:

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

است، که در آن:

$$V^{(k)} \in \mathbb{R}^{R_k \times I_k}, \quad \forall k = 0, \dots, N.$$

با این عملگرهای ضرب چندحالتی ( $\times_n$ ) تعداد پارامترها به شکلی چشمگیر کاهش می یابند [۲۸].

تحلیل تعداد پارامترها

در TCL، کل پارامترهای قابل آموزش برابر است با:

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

در حالی که در لایه متصل سنتی (پس از مسطح سازی):

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

که  $O$  تعداد نورون های خروجی لایه است. نمایش ریاضی بالا به وضوح نشان می دهد که TCL با جایگزینی ضرب با جمع ساده، به صرفه سازی قابل توجهی در حافظه و محاسبات مدل منجر می شود [۲۸، ۲۹].

## ۲.۰.۳ لایه رگرسیون تانسوری (TRL)

این لایه، خروجی مدل را مستقیماً از تانسور ورودی تولید می‌کند، بدون مسطح‌سازی؛ به‌طوری‌که ساختار چندبعدی ورودی حفظ و نگاشت خروجی نیز به‌صورت مرتبه پایین (low-rank) مدل شده است [۲۹].

فرمول‌بندی ریاضی

فرض مجدد تانسور:

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

و تانسور وزن:

$$\mathcal{W} \in \mathbb{R}^{I_0 \times I_1 \times \dots \times I_N \times O}.$$

نگاشت خروجی:

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

در اینجا عملگر  $\langle \cdot, \cdot \rangle_N$  ضرب داخلی روی  $N$  بعد اول  $\mathcal{W}$  و  $N$  بعد اول  $\mathcal{X}$  را نشان می‌دهد [۲۹].

برای کاهش پارامترها،  $\mathcal{W}$  با کمک تجزیه توکر به شکل زیر نوشته می‌شود:

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

که  $\mathcal{G}$  هسته کم‌مرتبه (core) tensor و  $U^{(k)}$  ماتریس‌های فشرده‌سازی هستند [۲۹].

فرمول نهایی:

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

که به‌صورت محاسباتی بهینه‌تر عمل می‌کند [۲۹].

## تحلیل تعداد پارامترها

در لایه TRL با تجزیه Tucker، تعداد پارامترها برابر است با:

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

که معمولاً بسیار کمتر از تعداد پارامترهای لایه FC است، به ویژه هنگامی که  $R_k \ll I_k$  انتخاب شوند [۲۹].

## ۳.۰.۳ چرا در مبدل های بینایی از تانسور استفاده می کنیم؟

## ۱. کاهش تعداد پارامترها و حافظه مصرفی

یکی از چالش های اصلی در معماری های مبتنی بر مبدل های بینایی، رشد نمایی تعداد پارامترها در لایه های کاملاً متصل، به ویژه در بخش های انتهایی شبکه است. با جایگزینی این لایه ها با ساختارهای تانسوری مانند TCL و TRL، می توان از ساختار چندبعدی داده بهره برده و از طریق تجزیه های کم مرتبه، تعداد پارامترها را به صورت چشمگیری کاهش داد [۴۶، ۲۸، ۲۹]. این جایگزینی نه تنها موجب صرفه جویی در حافظه می شود، بلکه پیچیدگی محاسباتی مدل را نیز کاهش داده و امکان به کارگیری آن را در محیط های کم منبع (مانند دستگاه های لبه ای و موبایل) فراهم می سازد [۱۷].

## ۲. افزایش تفسیرپذیری با حفظ ساختار چندخطی داده

در حالی که عملیات مسطح سازی بر روی تانسورهای ورودی، ساختار ارتباطی میان ابعاد داده را از بین می برد، استفاده از لایه های تانسوری (TCL) و (TRL) این ساختار چندخطی را حفظ می کند. این امر نه تنها مدل را برای کار با داده های فضایی یا زمانی چندکاناله مطلوب تر می سازد بلکه قابلیت تفسیرپذیری پیش بینی های مدل را نیز ارتقا می دهد. چرا که استحصال ساختارهای فیچر درون تانسوری (مثلاً بعدهای اساسی کابل ها) می تواند به تشریح بهتر مسیرهای تصمیم گیری مدل کمک کند [۱۷].

## ۳. کاهش نیاز به داده های آموزشی بزرگ

مدل های بینایی مبتنی بر مبدل، به دلیل فقدان سوگیری های مکانی ذاتی شبکه های کانولوشنی، نیازمند حجم عظیمی از داده برای آموزش مؤثر هستند. با استفاده از لایه های تانسوری مانند TCL و TRL که نگاشت ها را به صورت چندخطی و فشرده مدل سازی می کنند و ساختار درونی داده را حفظ می کنند، می توان از ظرفیت مدل به صورت بهینه تر بهره برد. این ساختار نه تنها کمک می کند وابستگی های میان بُعدی بهتر درک شوند بلکه از بیش برآزش در شرایط داده ی محدود جلوگیری می کند؛ به عنوان مثال، در روش FacT، (Factor-Tuning) فقط 0.01% پارامترهای ViT برای تعداد بسیار کمی از نمونه های آموزشی آموزش داده می شود و با صرفه جویی قابل توجه در پارامترها، عملکردی مشابه یا بهتر از تنظیم کامل (full-tuning) ارائه می دهد [۲۵].

## ۴.۰.۳ روش تانسوری مبدل پنجره متحرک:

## ۵.۰.۳ پیاده سازی مرحله ی تعبیه پیچ با استفاده از فشرده سازی تانسوری

در معماری های کلاسیک مبتنی بر مبدل های بیانی<sup>۷</sup> و همچنین در ساختار مبدل های پنجره ای<sup>۸</sup>، مرحله ی اولیه ی پردازش تصویر شامل تقسیم تصویر به قطعات کوچک (پیچ ها) و سپس نگاشت هر پیچ به یک بردار نهفته با ابعاد ثابت است. این نگاشت معمولاً با استفاده از یک لایه ی خطی<sup>۹</sup> یا پیچشی<sup>۱۰</sup> با کرنل و گام برابر با اندازه ی پیچ انجام می شود.

در روش تانسوری به جای استفاده از نگاشت برداری ساده، از لایه ی فشرده سازی تانسوری<sup>۱۱</sup> برای تبدیل هر پیچ به یک تانسور چندبندی در فضای ویژگی استفاده شده است. این نگاشت تانسوری نه تنها باعث حفظ ساختار چندخطی پیچ ها می شود، بلکه با فشرده سازی مؤثر، منجر به کاهش پارامترها می شود.

Vision transformer<sup>۷</sup>swin transformer<sup>۸</sup>linear<sup>۹</sup>convolution<sup>۱۰</sup>tensor contraction layer<sup>۱۱</sup>

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

•  $B$  اندازهی دستهی آموزشی<sup>۱۲</sup> است.

•  $C$  تعداد کانالهای تصویر (برای مثال ۳ در تصاویر رنگی).

•  $H \times W$  ابعاد تصویر است.

این تصویر با استفاده از پچهایی با اندازهی  $(P \times P)$  به  $\frac{H}{P} \times \frac{W}{P}$  قطعه تقسیم می شود. سپس با استفاده از عملیات بازآرایی<sup>۱۳</sup>، ساختار ورودی به شکل زیر تبدیل می شود:

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

که در آن:

•  $P_1 = \frac{H}{P}$  و  $P_2 = \frac{W}{P}$  به ترتیب تعداد پچه‌ها در راستای ارتفاع و عرض تصویر هستند.

•  $C_1 = P$  و  $C_2 = P$  ابعاد مکانی هر پچه‌اند.

•  $C_3 = C$  تعداد کانالهای تصویر ورودی است.

در این بازنمایی، هر پچه در موقعیت  $(i, j)$  به صورت یک تانسور سه بعدی با ابعاد  $C_1 \times C_2 \times C_3$  نمایش داده می شود. این ساختار چندبعدی امکان استفادهی مستقیم از عملیات تانسوری بدون نیاز به تخت سازی را فراهم می سازد.

برای نگاشت هر پچه به فضای نهفته، به جای استفاده از mlp از یک لایه فشرده ساز تانسوری استفاده می شود. در واقع سه تا بعد آخر با استفاده از لایه فشرده ساز تانسوری تعبیه می شود.

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

---

Batch Size<sup>۱۲</sup>  
rearrangement<sup>۱۳</sup>

در این ساختار، هر پیچ به جای تبدیل به یک بردار تخت، به یک تانسور سه بعدی در فضای نهفته تبدیل می شود. این تانسور ساختار درونی پیچ را حفظ کرد می کند.

فرمول بندی ریاضی عملیات فشرده سازی

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

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

عملیات فشرده سازی با استفاده از ضرب های چندحالت<sup>۱۴</sup> صورت می گیرد:

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

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

در این مدل ساختار چند بعدی به پیچ 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 مرحله ی تعبیه پیچ به صورت زیر تعریف می شود:

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

که در آن هر پیچ به صورت یک تانسور کم بعد در فضای نهفته نمایش داده شده است.

### ۶.۰.۳ ماژول توجه سلف چندسری مبتنی بر پنجره به صورت تانسوری

همانطور که در فصل قبل بیان شد در ساختار مبدل پنجره ای، جهت کاهش پیچیدگی محاسباتی، از روشی با عنوان<sup>۱۵</sup> بهره گرفته می شود. در این روش، ویژگی های مکانی تصویر به پنجره های کوچک

<sup>۱۴</sup> n-mode product

<sup>۱۵</sup> Window-based Multi-Head Self-Attention

با اندازه ی ثابت (بدون هم پوشانی) تقسیم شده و سپس عملیات خودتوجهی به صورت محلی و درون هر پنجره مستقل انجام می گیرد.

در روش تانسوری، به جای تولید بردارهای تخت شده ی  $Q$ ،  $K$  و  $V$ ، این بردارها به صورت تانسورهای چندبعدی با استفاده از لایه فشرده ساز تانسوری تولید می شوند.

ساختار ورودی و تقسیم به پنجره ها

خروجی مرحله تعبیه به صورت 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 پنجره های  $w \times w$  تقسیم می شوند. 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  $N_W = \frac{W}{w}$  به ترتیب تعداد پنجره ها در راستای ارتفاع و عرض تصویر هستند.

۲. محاسبه ی  $K$  و  $Q$  و  $V$  با استفاده از لایه فشرده ساز تانسوری

برای هر پنجره ی مکانی، سه لایه ی فشرده سازی تانسوری مستقل (TCL) برای استخراج تانسورهای  $K$  و  $Q$  طراحی شده اند. فرض می شود 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- $n$  به ترتیب در ابعاد تانسوری، نگاشت های  $K$  و  $Q$  و  $V$  به صورت زیر حاصل می شوند:



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

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

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

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

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

### ۳. تقسیم سرهای توجه چندگانه

برای پیاده سازی مکانیزم چندسر<sup>۱۶</sup>، ابعاد تانسوری خروجی به بخش های کوچک تری تقسیم می شود. فرض می شود:

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

که در آن:

•  $h_1, h_2, h_3$  تعداد سرها در هر بُعد تانسوری،

•  $d_1, d_2, d_3$  ابعاد نهفته ی هر سر در هر بُعد هستند.

با استفاده از عملیات بازآرایی، تانسورهای  $\mathcal{Q}$  و  $\mathcal{K}$  به شکل زیر سازمان دهی می شوند:

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

و ساختار مشابهی برای  $\mathcal{K}$  و  $\mathcal{V}$  در نظر گرفته می شود.

---

<sup>۱۶</sup>multi-head

## ۴. محاسبه ماتریس توجه تانسوری

عملیات ضرب داخلی تعمیم یافته بین تانسورهای Q و K برای محاسبه ی ماتریس توجه انجام می شود. برای هر سر  $(a, b, c)$  و برای موقعیت های  $(i, j)$  و  $(k, l)$  در پنجره، مقدار توجه به صورت زیر محاسبه می شود:

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

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، نرمال سازی زیر اعمال می شود:

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

و از نسخه علامت دار تابع softmax برای اعمال توجه استفاده می گردد:

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

## ۵. اعمال توجه و بازسازی ویژگی

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

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

و پس از انجام توجه برای هر پنجره دوباره باز آرای می شوند و به حالت اولیه باز می گردند و ابعاد آن به صورت زیر می شود.

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

در این مدل چون سر ها در سه بعد تقسیم می شوند در نتیجه این مدل آزادی خیلی بیشتری دارد.

### ۷.۰.۳ توجه علامت دار

در مکانیزم های استاندارد خود توجهی که در مبدل ها مورد استفاده قرار می گیرند، ماتریس های  $Q$ ،  $K$  و  $V$  با یکدیگر تعامل دارند تا وزن های توجه استخراج شوند. در این فرآیند، بردارهای  $Q$ <sup>۱۷</sup> و  $K$

<sup>۱۸</sup> با استفاده از ضرب داخلی محاسبه می شوند و نتیجه برای هر توکن، بیانگر میزان اهمیت نسبی سایر توکن ها در ارتباط با آن توکن است.

فرمول رایج محاسبه توجه به صورت زیر تعریف می شود:

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

در این ساختار، خروجی  $QK^T$  می تواند شامل مقادیر مثبت، منفی یا صفر باشد؛ اما پس از اعمال تابع Softmax، تمامی مقادیر به صورت نرمال سازی شده و مثبت خواهند بود. در نتیجه، اطلاعات قطبیت (علامت) که در نتیجهی ضرب داخلی بین بردارهای  $Q$  و  $K$  وجود داشت، از بین می رود. این ویژگی باعث محدود شدن ظرفیت مدل در شناسایی «رابطه منفی یا متضاد» میان توکن ها می شود. برای حفظ این اطلاعات قطبیت، د از رویکردی با عنوان توجه علامت دار استفاده شده است. ایده ی اصلی این روش آن است که پس از محاسبه ی امتیازهای توجه  $A = QK^T$ ، عملیات

---

Query<sup>۱۷</sup>

Key<sup>۱۸</sup>

نرمال سازی Softmax بر قدرمطلق این امتیازها انجام شده و سپس علامت اولیه آن ها به نتیجه بازگردانده می شود.

فرمول این روش به صورت زیر تعریف می شود:

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

که در آن:

- $A = QK^T$  ماتریس امتیازهای اولیه توجه است.
  - $\text{sign}(A)$  عملیاتی است که علامت هر مقدار را حفظ می کند (مقادیر +1، -1 یا 0).
  - $|A|$  قدرمطلق مقادیر است که شدت شباهت را بدون در نظر گرفتن جهت نشان می دهد.
- برای درک بهتر تفاوت میان مکانیزم های توجه معمولی و توجه علامت دار یک مثال عددی ساده اما گویای زیر ارائه می شود.
- فرض کنیم بردار پرس و جو (Q) و دو بردار کلید (K) به صورت زیر تعریف شده اند:

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

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

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

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

بنابراین بردار امتیاز توجه به صورت زیر حاصل می شود:

$$A = [2, 0]$$

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

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

در اینجا، هر دو کلید مقدار وزن مثبت می گیرند، حتی اگر امتیاز شباهت دومی برابر با صفر بوده باشد. مدل نمی تواند تفاوت میان بی اثر بودن و تأثیر منفی یا متضاد را به خوبی درک کند.

ب) توجه علامت دار: در این رویکرد، ابتدا بردار  $A$  به دو بخش علامت و قدر مطلق تفکیک می شود:

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

سپس تابع softmax فقط روی قدر مطلق ها اعمال شده و خروجی زیر حاصل می شود:

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

در نهایت، با ضرب عنصر به عنصر با علامت ها:

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

در این حالت، کلید دوم به طور 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 وزن های مثبت نرمال سازی می شوند. در مقابل، توجه علامت دار توانایی تمایز بین ارتباط مثبت، منفی و خنثی را دارد. این قابلیت می تواند در مدل سازی دقیق تر تضادهای معنایی یا شباهت های منفی نقش مهمی ایفا کند.

## ۸.۰.۳ توجه مبتنی بر پنجره های جابه جاشده به صورت تانسوری

توجه پنجره ای دارای یک محدودیت مهم است: پیچ های واقع در پنجره های متفاوت هیچ گونه تبادل اطلاعاتی ندارند. در حالی که ممکن هست این پیچ ها به هم نزدیک باشند به منظور رفع این محدودیت، مبدل های پنجره ای مکانیزم توجه مبتنی بر پنجره جا به جا شده معرفی شده است که در آن پنجره ها در برخی لایه ها به صورت جابه جاشده تعریف می شوند و از ماسک توجه برای جلوگیری از تداخل غیر مجاز استفاده می شود.

## مراحل دقیق مکانیزم

۱. شیفت چرخشی تانسور ورودی: خروجی مرحله قبلی با یک شیفت چرخه ای <sup>۱۹</sup> به اندازه  $\lfloor \frac{w}{2} \rfloor$  پیکسل در راستای افقی و عمودی جابه جا می شود. این جابه جایی باعث می شود که پیچ هایی که پیش تر در پنجره های جداگانه بودند، اکنون در یک پنجره جدید قرار گیرند. به این شیفت روی بعد اول و دوم تانسور که مکان پیچ را مشخص میکرد صورت میگیرد.
۲. اعمال پنجره بندی جدید: تانسور جابه جاشده به پنجره های بدون هم پوشانی جدید با اندازه  $w \times w$  تقسیم می شود. پنجره های جدید از موقعیت های مختلف تصویر تشکیل شده اند.
۳. اعمال خودتوجهی به صورت محلی با ماسک: از آنجا که پنجره ها پس از شیفت ممکن است شامل توکن هایی از مکان های نامرتب باشند، نیاز است که توجه به صورت مفید انجام شود. یک ماسک دودویی روی ماتریس توجه اعمال می شود که فقط اجازه توجه به توکن هایی را می دهد که واقعاً در یک پنجره معتبر قرار دارند.

خودتوجهی پنجره جا به جا شده باعث ارتباط میان پیچ های نزدیکی میشود که در خودتوجهی پنجره ای در یک پنجره قرار نداشته اند.

---

<sup>۱۹</sup> cyclic shift

## ۹.۰.۳ ادغام پچ ها به صورت تانسوری

در مدل های سلسله مراتبی بینایی مانند پس از هر مرحله از استخراج ویژگی، لازم است ابعاد مکانی تصویر کاهش یابد تا ویژگی ها در سطوح بالاتر به صورت فشرده تر و معنایی تر نمایش داده شوند. این فرایند با عنوان ادغام پچ ها<sup>۲۰</sup> شناخته می شود. در نسخه ی اصلی این مدل، ادغام پچ ها با استفاده از ترکیب ۴ پچ مجاور (به صورت یک پنجره ی  $2 \times 2$ ) و سپس اعمال یک لایه ی خطی پس از مسطح سازی انجام می شود. این کار باعث نصف شدن ابعاد مکانی و افزایش ابعاد ویژگی می شود. در این پژوهش، رویکرد متفاوتی برای ادغام پچ ها ارائه شده است که مبتنی بر نگاشت تانسوری است. به جای انجام<sup>۲۱</sup> بر روی پچ ها، از ساختار چندبعدی پچ ها استفاده شده و با کمک یک لایه ی فشرده سازی تانسوری، ادغام پچ ها بدون از بین رفتن ساختار چندخطی آن ها انجام می شود.

## ساختار ورودی

مانند 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،
- $H \times W$  ابعاد مکانی پچ ها،
- $(R_1, R_2, C)$  ابعاد فضای نهفته ی هر پچ به صورت تانسوری.

هدف این مرحله، کاهش ابعاد مکانی به  $(\frac{H}{2}, \frac{W}{2})$  و در عین حال، افزایش غنای بازنمایی ویژگی ها

است.

---

<sup>۲۰</sup> Patch Merging  
<sup>۲۱</sup> flatten

## ادغام پچ های مجاور

برای کاهش ابعاد مکانی، ابتدا هر چهار پچ مجاور در پنجره ای  $2 \times 2$  انتخاب می شوند. این چهار پچ به صورت زیر دسته بندی می شوند:

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

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

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

## فشرده سازی ویژگی های ادغام شده

پس از ادغام کانالی، لازم است ابعاد ویژگی ها به صورت کنترل شده کاهش یابد تا بتوان آن را به شکل ورودی ماژول های بعدی تطبیق داد. برای این منظور، از یک لایه ی فشرده سازی تانسوری استفاده می شود که به صورت مستقیم بر روی ابعاد نهفته  $(R_1, R_2, 4C)$  عمل می کند. برخلاف لایه های خطی کلاسیک، در اینجا عملیات فشرده سازی در چندین بعد به صورت همزمان و با حفظ ساختار چندخطی انجام می شود.

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 ویژگی ها 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 می کند که ادغام از نظر اطلاعاتی قابل پشتیبانی و از نظر مدل سازی متوازن باقی می ماند.

### ۱۰.۰.۳ مرحله ی طبقه بندی نهایی با استفاده از رگرسیون تانسوری

پس از عبور تصویر ورودی از مراحل سلسله مراتبی مختلف شامل تعبیه پیچ، مازول های خودتوجهی تانسوری، و لایه های ادغام و فشرده سازی، خروجی نهایی آخرین مرحله به صورت یک تانسور چندبعدی در فضای نهفته حاصل می شود. این تانسور حاوی بازنمایی غنی از ویژگی های سطح بالا و ساختار درونی تصویر است که برای تصمیم گیری نهایی مورد استفاده قرار می گیرد.

میانگین گیری تانسوری در ابعاد مکانی

فرض میکنیم خروجی مرحله ی نهایی دارای ساختار تانسوری زیر باشد:

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

برای استخراج یک بازنمایی کلی از کل تصویر، میانگین گیری در دو بُعد مکانی  $(H, W)$  انجام می شود. در واقع همانند مبدل پنجره ای متحرک از مکان های پیچ ها میانگین گیری گرفته میشود این عملیات باعث فشرده سازی اطلاعات مکانی و حفظ ساختار تانسوری در فضای ویژگی می شود.

$$\bar{\mathcal{X}} = \frac{1}{H \cdot W} \sum_{i=1}^H \sum_{j=1}^W \mathcal{X}_{[:,i,j,:,:,,:]} \in \mathbb{R}^{B \times D_1 \times D_2 \times D_3}$$

این عملیات نوعی تجميع جهانی در ابعاد فضايي محسوب مي شود كه مشابه با ميانگين گيري سراسري<sup>۲۲</sup> در شبكه هاي پيچشي عمل مي كند، با اين تفاوت كه ساختار تانسوري ويژگي ها را حفظ مي نمايد.

طبقه بندي با استفاده از لايه رگرسيون تانسوري

براي نگاشت خروجي  $\bar{\mathcal{X}}$  به برچسب نهايي، از يك لايه رگرسيون تانسوري (TRL) استفاده شده است. اين لايه با بهره گيري از ضرب داخلي تعميم يافته ميان تانسور ورودي و تانسور وزن، نگاشتي خطي از فضاي تانسوري به فضاي خروجي اعمال مي كند:

$$\mathbf{Y} = \langle \bar{\mathcal{X}}, \mathcal{W} \rangle_3 + \mathbf{b}$$

كه در آن:

- $\bar{\mathcal{X}} \in \mathbb{R}^{B \times D_1 \times D_2 \times D_3}$  بازنمايي فشرده ي ورودي،
- $\mathcal{W} \in \mathbb{R}^{D_1 \times D_2 \times D_3 \times C}$  تانسور وزن هاي رگرسيون،
- $\langle \cdot, \cdot \rangle_3$  ضرب داخلي تعميم يافته در سه بعد آخر  $\bar{\mathcal{X}}$  و سه بعد اول  $\mathcal{W}$  است،
- $\mathbf{b} \in \mathbb{R}^C$  بردار باياس،
- $C$  تعداد كلاس هاي خروجي است.

---

<sup>۲۲</sup> global average pooling

## 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. [5](#), [6](#), [7](#), [8](#), [12](#), [13](#)
- [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. [24](#)
- [6] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995. [11](#), [12](#)
- [7] Thomas M. Cover and Peter E. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1):21–27, 1967. [10](#), [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. [10](#), [11](#)
- [13] Jeffrey L. Elman. Finding structure in time. *Cognitive Science*, 14(2):179–211, 1990. [15](#), [25](#), [29](#)
- [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](#), [4](#)
- [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](#), [18](#), [20](#)
- [16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, Cambridge, MA, 2016. [6](#), [14](#), [15](#), [16](#), [17](#), [20](#), [21](#), [22](#)
- [17] Safa Hamrera, Sukhbinder Singh, and Román Orús. Tensorization as a powerful tool for compression and interpretability in neural networks: A position paper. *arXiv preprint arXiv:2505.20132*, 2025. [65](#)
- [18] 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](#)

- [19] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016. [55](#)
- [20] 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](#)
- [21] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997. [13](#), [15](#), [16](#), [17](#), [18](#), [20](#), [21](#), [29](#), [34](#), [35](#)
- [22] John Hutchins. *Machine translation: past, present, future*. Ellis Horwood Chichester, 1986. [24](#)
- [23] 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. [35](#), [36](#), [37](#)
- [24] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An Introduction to Statistical Learning: with Applications in R*. Springer, New York, 2013. [6](#), [11](#)
- [25] Shibo Jie and Zhi-Hong Deng. Fact: Factor-tuning for lightweight adaptation on vision transformer. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2023. [0.01%](#) ViT. [66](#)
- [26] Philipp Koehn. *Statistical Machine Translation*. Cambridge University Press, 2010. [24](#)
- [27] Philipp Koehn, Franz Josef Och, and Daniel Marcu. Statistical phrase-based translation. In *Proc. of NAACL/HLT*, pages 48–54, 2003. [25](#)
- [28] Jean Kossaifi, Aran Khanna, Zachary C. Lipton, Tommaso Furlanello, and Anima Anandkumar. Tensor contraction layers for parsimonious deep nets. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, 2017. [60](#), [61](#), [62](#), [63](#), [65](#)

- [29] Jean Kossaifi, Zachary C. Lipton, Arinbjörn Kolbeinsson, Aran Khanna, Tommaso Furlanello, and Anima Anandkumar. Tensor regression networks. *Journal of Machine Learning Research*, 21(123):1–21, 2020. [60](#), [61](#), [62](#), [63](#), [64](#), [65](#)
- [30] 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](#)
- [31] Vadim Lebedev, Yaroslav Ganin, Maksim Rakhuba, Ivan V. Oseledets, and Victor S. Lempitsky. Speeding-up convolutional neural networks using fine-tuned cp-decomposition. In *International Conference on Learning Representations (ICLR)*, 2015. [61](#)
- [32] 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](#)
- [33] James Lighthill. *Artificial Intelligence: A General Survey*. HM Stationery Office, London, 1973. Science Research Council Report. [4](#)
- [34] 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](#)
- [35] 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](#)
- [36] 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. [12](#)
- [37] 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](#)

- [38] 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](#)
- [39] 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. [27](#)
- [40] Tom M. Mitchell. *Machine Learning*. McGraw-Hill, New York, 1997. [5](#), [6](#), [10](#), [11](#), [12](#)
- [41] Douglas C. Montgomery, Elizabeth A. Peck, and Geoffrey G. Vining. *Introduction to Linear Regression Analysis*. John Wiley & Sons, Hoboken, NJ, 6th edition, 2021. [8](#)
- [42] Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective*. MIT Press, Cambridge, MA, 2012. [5](#), [6](#), [7](#), [11](#), [12](#)
- [43] 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](#)
- [44] 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](#)
- [45] Nils J. Nilsson. *The Quest for Artificial Intelligence: A History of Ideas and Achievements*. Cambridge University Press, Cambridge, 2010. [3](#), [4](#), [5](#)
- [46] Alexander Novikov, Dmitry Podoprikin, Anton Osokin, and Dmitry P. Vetrov. Tensorizing neural networks. In *Advances in Neural Information Processing Systems*, volume 28, pages 442–450, 2015. [59](#), [65](#)
- [47] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *Proc. of EMNLP*, pages 1532–1543, 2014. [27](#)
- [48] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. OpenAI report, 2018. [24](#)



- [49] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986. [13](#), [16](#), [21](#)
- [50] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson, London, 3rd edition, 2016. [4](#), [5](#)
- [51] 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](#)
- [52] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 2nd edition, 2018. [9](#)
- [53] Vladimir Vapnik. *Statistical Learning Theory*. Wiley, New York, 1998. [11](#)
- [54] 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](#)
- [55] Yinchong Yang, Denis Krompass, and Volker Tresp. Tensor-train recurrent neural networks for video classification. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*, pages 3891–3900, 2017. [62](#)

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