



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

نابستان 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

بی‌لعدد قفل‌ه‌عقدل‌خ‌ق‌ف‌د‌ل‌ق‌ف‌ل‌ق‌ف‌ا‌ق‌ا

# فهرست مطالب

ه	فهرست جداول
و	فهرست تصاویر
ا	پیش‌گفتار
ب	۱ مقدمه
پ	۱.۰.۱ آغاز هوش مصنوعی و هدف اصلی . . . . .
ت	۲.۰.۱ دوره‌طلابی و پیشرفت‌های اولیه . . . . .
ث	۳.۰.۱ انتظارات بیش از حد و ظهور عصر تاریک . . . . .
ج	۴.۰.۱ عوامل اصلی عصر تاریک هوش مصنوعی . . . . .
چ	۵.۰.۱ پایان عصر تاریک و بازگشت هوش مصنوعی . . . . .
ح	۱.۱ انواع مدل یادگیری ماشین و شبکه‌های عصبی . . . . .
خ	۱.۱.۱ یادگیری ماشین: مروری کلی . . . . .
د	۲.۱.۱ تقسیم‌بندی‌های اصلی در یادگیری ماشین . . . . .
ذ	۳.۱.۱ یادگیری نظارت‌شده . . . . .
ر	۴.۱.۱ یادگیری بدون نظارت . . . . .
ز	۵.۱.۱ یادگیری تقویتی . . . . .

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

۲	پیشینه پژوهش	۲۴
۱.۲	مشکلات ترجمه ماشینی و مبدل ها	۲۴
۲.۲	ظهور ترانسفورمرها	۲۵
۳.۲	معماری مبدل ها	۲۶
۱.۳.۲	جاسازی	۲۶
۲.۳.۲	جاسازی موقعیتی	۲۷
۳.۳.۲	توجه	۳۰
۴.۳.۲	توجه چند سر	۳۲
۵.۳.۲	اتصال باقی مانده	۳۴
۴.۲	نرمال سازی لایه ها	۳۵
۱.۴.۲	اتصال باقی مانده	۳۷
۵.۲	رمزگشا	۳۸
۶.۲	توجه چند سری ماسک شده	۳۹
۷.۲	مثال عددی توجه ماسک شده	۴۰
۸.۲	مبدل های بینایی	۴۱
۱.۸.۲	جاسازی پچ ها در مبدل های بینایی	۴۱
۲.۸.۲	شکل پچ ها:	۴۲
۳.۸.۲	تعداد پچ ها:	۴۲
۴.۸.۲	بردارکردن هر پچ	۴۳
۹.۲	اعمال لایه خطی	۴۴
۱.۹.۲	توکن کلاس بندی	۴۵
۲.۹.۲	رمزگذار در مبدل های بینایی	۴۶
۱۰.۲	مبدل پنجره ای متحرک	۴۷

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	پرسپتروون چند لایه
۵۶	۶.۱۰.۲	ترکیب پیچ ها

۵۹	۳	پیشینه پژوهش
۶۲	۱.۰.۳	لایه فشرده‌سازی تانسوری (Tensor Contraction Layer)
۶۴	۲.۰.۳	لایه رگرسیون تانسوری (Tensor Regression Layer)
۶۶	۳.۰.۳	چرا در مبدل های بینایی از تانسور استفاده میکنیم؟
۶۸	۴.۰.۳	روش تانسوری مبدل پنجره متحرک:

۶۹	۴	آزمایشات و نتایج
----	---	------------------

۷۰		کتاب‌نامه
----	--	-----------

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



## فهرست جداول

۲۰	۱.۲.۱ مقایسه ویژگی‌های RNN و LSTM . . . . .
----	---

## فهرست تصاویر

۸	۱.۱.۱ کلاس بندی . . . . .
۸	۱.۱.۲ رگرسیون . . . . .
۹	۱.۱.۳ خوشه بندی . . . . .
۱۰	۱.۱.۴ یادگیری تقویتی . . . . .
۱۴	۱.۱.۵ شبکه های عصبی بازگشتی . . . . .
۱۸	۱.۲.۶ LSTM . . . . .
۲۷	۲.۳.۱ معماری ترانسفورمرها . . . . .
۲۸	۲.۳.۲ . . . . .
۲۹	۲.۳.۳ . . . . .
۳۱	۲.۳.۴ . . . . .
۳۲	۲.۳.۵ توجه . . . . .
۳۳	۲.۳.۶ توجه چند سر . . . . .
۳۹	۲.۵.۷ . . . . .
۴۲	۲.۸.۸ بخش بندی تصاویر . . . . .
۴۴	۲.۹.۹ مبدل های بینایی . . . . .
۴۷	۲.۹.۱۰ توکن توجه در مبدل های بینایی . . . . .

۴۸	۱۰.۱۱	مبدل پنجره متحرک
۵۴	۱۰.۱۲	اجا به جایی چرخه ای
۵۸	۱۰.۱۳	ادغام پیچ ها
۶۰	۳.۰.۱	Mlp
۶۲	۳.۰.۲	tensor contraction layer
۶۵	۳.۰.۳	tensor regression network

## پیش گفتار

قدش مقدر کن قصد بشقصد قفخد ل قخفاد خفادخ

# 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 این‌رو، فصل حاضر به‌منظور ارائه بستر نظری لازم، به بررسی سیر تاریخی هوش مصنوعی، معرفی روش‌های یادگیری ماشین، مروری بر الگوریتم‌های کلاسیک، و تحلیل ساختار شبکه‌های بازگشتی از جمله شبکه‌های حافظه کوتاه مدت طولانی<sup>۲</sup> اختصاص دارد.

---

<sup>۱</sup>Transformer

<sup>۲</sup>LSTM

## ۱.۰.۱ آغاز هوش مصنوعی و هدف اصلی

هوش مصنوعی به عنوان شاخه‌ای از علوم کامپیوتر، در دهه ۱۹۵۰ با هدف ساخت سیستم‌ها و ماشین‌هایی که توانایی تقلید از هوش انسانی را دارند، آغاز شد. نخستین بار، مکاریتی در سال ۱۹۵۶ این اصطلاح را به کار گرفت [۲۲] 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 [۴۰، 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 [۳۹]. 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].

---

<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)

### 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. 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، ۲۸]. در این دوران، بسیاری از پروژه‌ها تعطیل و سرمایه‌گذاری‌ها قطع شدند و دولت‌ها و سازمان‌های سرمایه‌گذار به دلیل عدم دستیابی به نتایج مطلوب از ادامه سرمایه‌گذاری منصرف شدند.

### ۴.۰.۱ عوامل اصلی عصر تاریک هوش مصنوعی

- محدودیت‌های سخت‌افزاری: در آن زمان، سیستم‌های اولیه هوش مصنوعی به محاسبات سنگینی نیاز داشتند که با توان پردازشی محدود آن دوره همخوانی نداشت [۴۰].
- کمبود داده‌ها: در آن زمان، دسترسی به داده‌های کافی برای آموزش مدل‌های پیچیده ممکن نبود و الگوریتم‌های موجود به داده‌های بیشتری نیاز داشتند تا بتوانند 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 [۴۴].

---

GPU<sup>۱۱</sup>

AI Winter<sup>۱۲</sup>

## 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. بر خلاف الگوریتم‌های اولیه، این سیستم‌ها از پایگاه بزرگ قواعد و قوانین<sup>۱۳</sup> استفاده می‌کردند. در سیستم‌های خبره، به جای تلاش برای شبیه‌سازی کلی هوش مصنوعی، بر حل مسائل تخصصی برای صنایع و سازمان‌ها تمرکز می‌شد. برای مثال، سیستم‌های خبره در پزشکی برای تشخیص بیماری‌ها و پیشنهاد درمان، در صنعت برای مدیریت و پیش‌بینی خرابی ماشین‌آلات، و در امور مالی برای تحلیل و ارزیابی ریسک کاربرد داشتند [۳۳].

هرچند این سیستم‌ها نمی‌توانستند درک عمیق و هوشمندی عمومی را ایجاد کنند، اما برای رفع نیازهای پیچیده مناسب بودند. همزمان با موفقیت این سیستم‌ها، بهبودهای زیادی در سخت‌افزارها و کاهش هزینه‌های پردازش به وجود آمد. در دهه‌های ۱۹۸۰ و ۱۹۹۰، کامپیوترها به تدریج قوی‌تر و مقرون به صرفه‌تر شدند و امکان پردازش داده‌های بیشتر و اجرای الگوریتم‌های پیچیده‌تر فراهم شد. این افزایش توان محاسباتی، نیاز به پردازش داده‌های بزرگ و پیچیده را برآورده کرد و در نتیجه دسترسی به داده‌ها و انجام محاسبات سنگین برای توسعه الگوریتم‌های جدید تسهیل شد. از سوی دیگر، پیشرفت‌های انجام‌شده در ذخیره‌سازی داده و رشد اینترنت باعث دسترسی گسترده‌تر به داده‌ها و منابع اطلاعاتی گردید [۴۰].

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



## ۱.۱ انواع مدل یادگیری ماشین و شبکه‌های عصبی

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

### ۱.۱.۱ یادگیری ماشین: مروری کلی

یادگیری ماشین<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، ۳۵]. در واقع، هدف یادگیری ماشین این است که مدل‌ها بتوانند از داده‌ها الگوها و روابط پنهان را استخراج کنند و به نتایج و تصمیم‌های قابل اعتماد دست یابند.

### ۲.۱.۱ تقسیم‌بندی‌های اصلی در یادگیری ماشین

به طور کلی، یادگیری ماشین به سه 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، ۳۷].

---

<sup>۱۴</sup> Machine Learning

<sup>۱۵</sup> Supervised Learning

<sup>۱۶</sup> Unsupervised Learning

<sup>۱۷</sup> Reinforcement Learning

### ۳.۱.۱ یادگیری نظارت شده

یادگیری نظارت شده یکی از رایج ترین روش ها در یادگیری ماشین شناخته می شود که در آن از مجموعه داده های برچسب گذاری شده برای آموزش مدل استفاده می کنیم [۲۳]. هدف این الگوریتم تشخیص الگوها در میان داده های ورودی است تا بتواند روی داده های جدید پیش بینی یا طبقه بندی انجام دهد. این نوع شامل دو دسته الگوریتم رگرسیون<sup>۱۸</sup> و کلاس بندی<sup>۱۹</sup> می شود.

#### کلاس بندی

در میان روش های یادگیری با نظارت، کلاس بندی<sup>۲۰</sup> یکی از پرکاربردترین مسائل محسوب می شود. هدف در این مسئله، تخصیص هر نمونه ورودی به یکی از برچسب های گسسته از پیش تعریف شده است. مدل با استفاده از داده های آموزشی که شامل ویژگی ها و برچسب صحیح هستند، الگوهای حاکم بر داده را می آموزد و تلاش می کند بر اساس آن، داده های جدید را به دسته مناسب اختصاص دهد. این روش در کاربردهای متنوعی نظیر تشخیص هرزنامه<sup>۲۱</sup>، شناسایی بیماری ها، طبقه بندی تصاویر و تحلیل احساسات به طور گسترده مورد استفاده قرار می گیرد [۴، ۳۷].

#### رگرسیون

در مقابل، رگرسیون<sup>۲۲</sup> به مسئله پیش بینی مقادیر پیوسته می پردازد. در این نوع از یادگیری نظارت شده، مدل می کوشد رابطه ای میان متغیرهای ورودی و خروجی های عددی برقرار کند تا بر اساس آن، بتواند خروجی نمونه های جدید را تخمین بزند. تفاوت اصلی رگرسیون با کلاس بندی در ماهیت خروجی است؛ به طوری که در رگرسیون، خروجی به جای دسته بندی، یک مقدار عددی خواهد بود. از جمله کاربردهای رایج این نوع مدل ها می توان به پیش بینی قیمت مسکن، تخمین میزان فروش، و پیش بینی

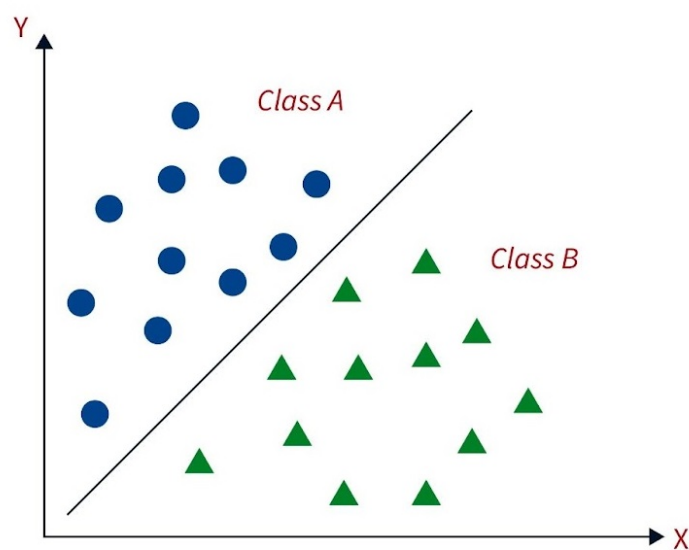
<sup>۱۸</sup> Regression

<sup>۱۹</sup> Classification

<sup>۲۰</sup> Classification

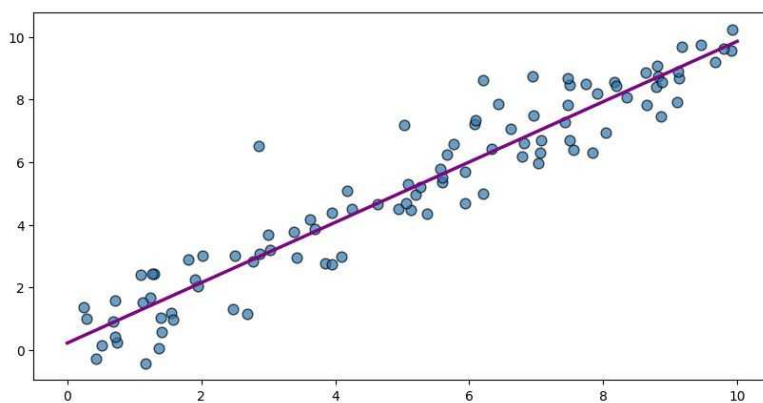
<sup>۲۱</sup> Spam Detection

<sup>۲۲</sup> Regression



شکل ۱.۱.۱: کلاس بندی

وضعیت آبوهوا اشاره کرد [۳۶].



شکل ۱.۱.۲: رگرسیون

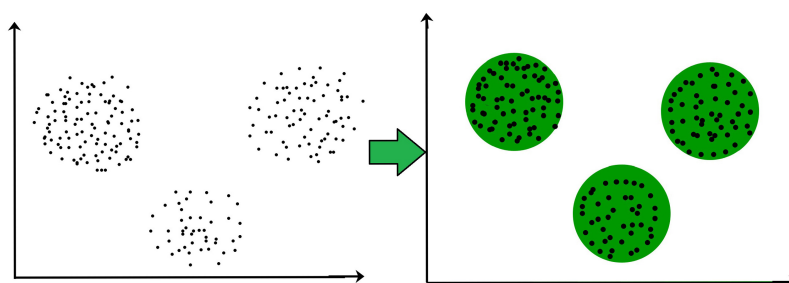
#### ۴.۱.۱ یادگیری بدون نظارت

یادگیری بدون نظارت<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

<sup>۲۳</sup> Unsupervised Learning

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

از کاربردهای رایج یادگیری بدون نظارت می‌توان به خوشه‌بندی<sup>۲۴</sup>، کاهش ابعاد<sup>۲۵</sup>، آشکارسازی ناهنجاری‌ها<sup>۲۶</sup>، و استخراج ویژگی‌ها اشاره کرد.



شکل ۱.۱.۳: 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 [۴۶]. برخلاف یادگیری نظارت‌شده و بدون نظارت، یادگیری تقویتی به مدل این امکان را می‌دهد تا از طریق آزمون و خطا بهترین راهکارها را برای انجام یک عمل یاد بگیرد. در این روش، مدل به جای برچسب، از یک تابع پاداش استفاده می‌کند که مشخص می‌کند چه اقداماتی باعث نتیجه بهینه می‌شود. از کاربردهای یادگیری تقویتی می‌توان به بازی‌ها<sup>۲۸</sup>،

<sup>۲۴</sup>Clustering

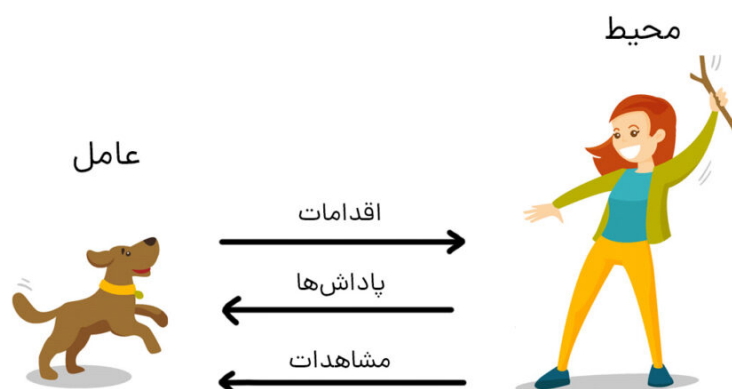
<sup>۲۵</sup>Dimensionality Reduction

<sup>۲۶</sup>Anomaly Detection

<sup>۲۷</sup>reinforcement learning

<sup>۲۸</sup>Games

کنترل رباتیک<sup>۲۹</sup> و سیستم‌های توصیه‌گر<sup>۳۰</sup> اشاره کرد.



شکل ۱.۱.۴: یادگیری تقویتی

### 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.

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.

<sup>۲۹</sup> Robotic Control

<sup>۳۰</sup> Recommender Systems

### 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، ۳۵]. این الگوریتم برای پیش‌بینی دسته‌بندی یک نمونه جدید، به  $k$  نزدیک‌ترین داده‌ها در فضای ویژگی نگاه می‌کند و بر اساس اکثریت نزدیکی همسایه‌ها، آن را به یک دسته اختصاص می‌دهد.

مزایا:

- سادگی و قابل فهم بودن: این الگوریتم به سادگی با اندازه‌گیری فاصله بین نقاط داده کار می‌کند و بدون نیاز به آموزش مدل پیچیده قابل استفاده است [۷].
- عملکرد خوب در داده‌های با تعداد ویژگی کم: در مسائلی که تعداد ویژگی‌ها کم است، این الگوریتم اغلب به خوبی عمل می‌کند [۲۳].

معایب:

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

---

<sup>۳۱</sup>k-Nearest Neighbors

### 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 [۴۷، ۶]. این الگوریتم یک ابرصفحه به دست می‌آورد که هدف آن حداکثر کردن فاصله میان داده‌های دو کلاس است و به این ترتیب می‌تواند طبقه‌بندی دقیقی داشته باشد.

مزایا:

- توانایی مقابله با داده‌های پیچیده و ابعاد بالا: ماشین بردار پشتیبان می‌تواند به خوبی با داده‌های چندبعدی و پیچیده کار کند [۴۷].
- مقاومت در برابر بیش‌برازش<sup>۳۳</sup>: با استفاده از هسته‌ها<sup>۳۴</sup>، داده‌های غیرخطی نیز به فضای بالاتر برده می‌شوند و جداسازی بهتری انجام می‌شود [۶].

معایب:

- پیچیدگی محاسباتی: آموزش ماشین بردار پشتیبان به دلیل نیاز به حل مسائل بهینه‌سازی، در حجم‌های بالای داده محاسباتی زمان‌بر است [۳۷].
- کارایی پایین در داده‌های پرت: در صورتی که داده‌ها شامل نقاط پرت زیادی باشند، دقت مدل کاهش می‌یابد [۴].

### ۹.۱.۱ بیز ساده

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

---

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

Overfitting<sup>۳۳</sup>

kernels<sup>۳۴</sup>

Bayes Naive<sup>۳۵</sup>

Bayes' theorem<sup>۳۶</sup>

از کاربردها مانند طبقه‌بندی ایمیل و تحلیل احساسات مورد استفاده قرار می‌گیرد [۳۱]. در بیز ساده بر اساس احتمالات محاسبه می‌شود که یک نمونه جدید به کدام دسته تعلق دارد. این الگوریتم بر اساس قضیه بیز، احتمال تعلق یک نمونه به هر دسته را به ازای هر ویژگی محاسبه کرده و در نهایت بالاترین احتمال را به عنوان جواب نهایی در نظر می‌گیرد [۴].

مزایا:

- سرعت بالا: به دلیل محاسبات ساده و فرض استقلال ویژگی‌ها، بیز ساده بسیار سریع و کم حجم است [۳۱].

- کارایی در داده‌های کوچک: حتی با داده‌های کم، این الگوریتم عملکرد نسبتاً خوبی دارد [۳۷].

معایب:

- فرض استقلال ویژگی‌ها: فرض استقلال ویژگی‌ها ممکن است در بسیاری از مسائل واقعی صادق نباشد و این می‌تواند دقت مدل را کاهش دهد [۱۰].

- حساسیت به داده‌های نادرست: در صورت وجود داده‌های نادرست یا پرت، مدل ممکن است دقت کمتری داشته باشد [۴].

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

شبکه‌های عصبی بازگشتی<sup>۳۷</sup> و مدل‌هایی با حافظه بلندمدت-کوتاه‌مدت<sup>۳۸</sup> با هدف پردازش داده‌های ترتیبی و وابسته به زمان توسعه یافتند [۲۰، ۴۳]. 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].

<sup>۳۷</sup>RNN

<sup>۳۸</sup>LSTM

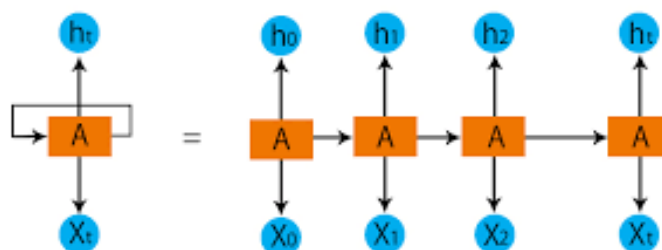


### ۱۱.۱.۱ شبکه‌های عصبی بازگشتی

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[۴۳].

### 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 ۱.۱.۵: شبکه‌های عصبی بازگشتی

$$h_t = \sigma(W \cdot x_t + U \cdot h_{t-1} + b) \quad (1.1.1)$$

در اینجا:

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

---

<sup>۳۹</sup>MLP  
<sup>۴۰</sup>recurrent neural network

- وزن‌هایی است که به ورودی  $x_t$  اعمال می‌شود.
  - وزن‌های اعمال‌شده به وضعیت قبلی  $h_{t-1}$  است.
  - بایاس مدل است.
  - تابع فعال‌سازی، معمولاً تانژانت هیپربولیک یا سیگموید.
- با استفاده از این فرایند، مدل این توانایی را دارد که اطلاعات گذشته را در خود ذخیره کرده و در پردازش‌های بعدی از آن‌ها بهره ببرد.

### ۱۳.۱.۱ مزایا و معایب شبکه‌های عصبی بازگشتی

در این قسمت به مزایا و معایب شبکه‌های 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 [۱۹].

<sup>۴۱</sup> Vanishing and Exploding Gradient

<sup>۴۲</sup> back propagation

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

### ۱۴.۱.۱ شبکه‌های حافظه بلندمدت - کوتاهمدت

شبکه‌های حافظه بلندمدت - کوتاهمدت به عنوان یک راه‌حل برای یکی از بزرگ‌ترین مشکلات شبکه‌های عصبی بازگشتی معرفی شدند [۲۰]. یکی از برجسته‌ترین مشکلات موجود در شبکه‌های عصبی بازگشتی، معضل ناپدید شدن گرادیان بود که مانع یادگیری وابستگی‌های بلندمدت می‌شد [۱۶، ۱۹]. 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. [۴۳].

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 [۱۹].

<sup>۴۳</sup> Backpropagation Through Time  
<sup>۴۴</sup> vanishing gradient

فرمول کلی گرادیان در زمان  $t$  به صورت زیر است:

$$\frac{\partial L}{\partial W} = \prod_{k=1}^t \frac{\partial h_k}{\partial h_{k-1}} \cdot \frac{\partial h_t}{\partial L} \quad (1.1.2)$$

در این فرمول،  $\frac{\partial h_k}{\partial h_{k-1}}$  ممکن است مقداری کوچک تر از ۱ باشد، و ضرب مکرر آن در طول توالی باعث کاهش شدید مقدار گرادیان می گردد.

• تأثیر مستقیم بر وزن ها: زمانی که گرادیان ها به صفر نزدیک می شوند، وزن های مدل عملاً به روزرسانی نمی شوند و این امر مانع از یادگیری وابستگی های طولانی مدت در داده ها می شود [۱۶].

### ۱۶.۱.۱ ظهور شبکه های حافظه بلندمدت - کوتاه مدت

در سال ۱۹۹۷، شبکه های حافظه بلندمدت - کوتاه مدت معرفی شد. [۲۰]. انگیزه اصلی توسعه این شبکه حل مشکل ناپدید شدن گرادیان در شبکه های عصبی بازگشتی بود. این مشکل در مسائل یادگیری داده های ترتیبی طولانی مانع می شد شبکه های عصبی بازگشتی وابستگی های بلندمدت را به درستی فراگیرد.

### ۱۷.۱.۱ راه حل شبکه های حافظه بلندمدت - کوتاه مدت برای پایداری جریان گرادیان ها

با معرفی شبکه حافظه بلند مدت، کوتاه مدت<sup>۴۵</sup> در شبکه های بازگشتی، جریان گرادیان ها را در طول توالی پایدار نگه می دارد. این کار از طریق اضافه کردن وضعیت سلولی<sup>۴۶</sup> و دروازه ها<sup>۴۷</sup> به ساختار شبکه های عصبی بازگشتی انجام می شود. [۱۵]. این اجزا به این شبکه این امکان را می دهند:

۱. اطلاعات غیرضروری را فراموش کند،

۲. اطلاعات مهم جدید را اضافه کند،

<sup>۴۵</sup> long short-term memory

<sup>۴۶</sup> Cell State

<sup>۴۷</sup> Gates

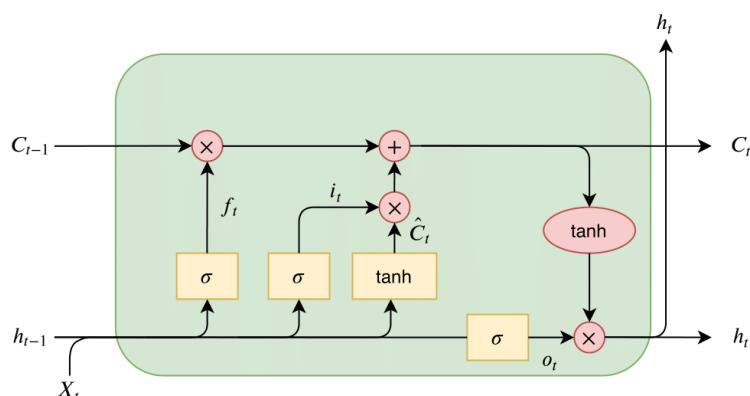
۳. اطلاعات مهم قبلی را حفظ کند.

## 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 [۲۰].



شکل ۱.۲.۶: LSTM

### 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 می‌کند چه اطلاعاتی از وضعیت سلولی باید حذف شود [۱۵].

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

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

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

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

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

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

- دروازه خروجی<sup>۵۰</sup>:

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

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

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

---

Forget Gate<sup>۴۸</sup>

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:

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

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 [۱۹].

- 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 [۲۰].

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 ۱.۲.۱: 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 [۲۰، ۱۶]. این مشکلات در نهایت به پیدایش مبدل ها<sup>۵۱</sup> منجر شد [۴۸].

## 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 [۲۰، ۴۳]. 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 [۱۹]:

- 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 [۴۸]. 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.[۴۸، 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[۴۲، 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 آنها، ترجمه ها به صورت دستی با استفاده از قواعد زبانی مشخص تنظیم می شدند [۳۸، ۲۱]. این روش ها هرچند دقیق بودند، اما

---

<sup>۱</sup>Transformer

<sup>۲</sup>NLP

<sup>۳</sup>machine translation

<sup>۴</sup>Rule-based Models

محدودیت‌های زیادی داشتند و نمی‌توانستند ویژگی‌های پیچیده‌تر زبان را مدل‌سازی کنند. سپس مدل‌های آماری<sup>۵</sup> معرفی شدند [۲۴، ۵]. این مدل‌ها از داده‌های ترجمه‌شده برای آموزش مدل‌های آماری استفاده می‌کردند که احتمال ترجمه صحیح را براساس شواهد آماری محاسبه می‌کردند. مدلهایی مانند مدل‌های ترجمه آماری مبتنی بر جمله<sup>۶</sup> [۲۵] از این نوع بودند که قادر به ترجمه جملات بهتر از مدل‌های مبتنی بر قواعد بودند، اما هنوز هم در ترجمه‌های پیچیده با مشکلاتی روبه‌رو بودند.

بعد از این مدل‌ها، مدل‌های بازگشتی<sup>۷</sup> به وجود آمدند که مشکلات آن‌ها در فصل گذشته بیان شد [۴۵، 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 [۴۸]. این مقاله به موضوع ترجمه ماشینی پرداخت و نشان داد که با استفاده از مکانیزم توجه می‌توان بسیاری از مشکلات مدل‌های قبلی را حل کرد [۳۰].

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

---

Statistical Models<sup>۵</sup>

Phrase-based Statistical Models<sup>۶</sup>

Recurrent Models<sup>۷</sup>

google<sup>۸</sup>

Transformers<sup>۹</sup>

## ۳.۲ معماری مبدل ها

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

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

- رمزگشا<sup>۱۱</sup>: وظیفه رمزگشا این است که ویژگی های استخراج شده را به زبان مقصد تبدیل کند.

## ۱.۳.۲ جاسازی

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

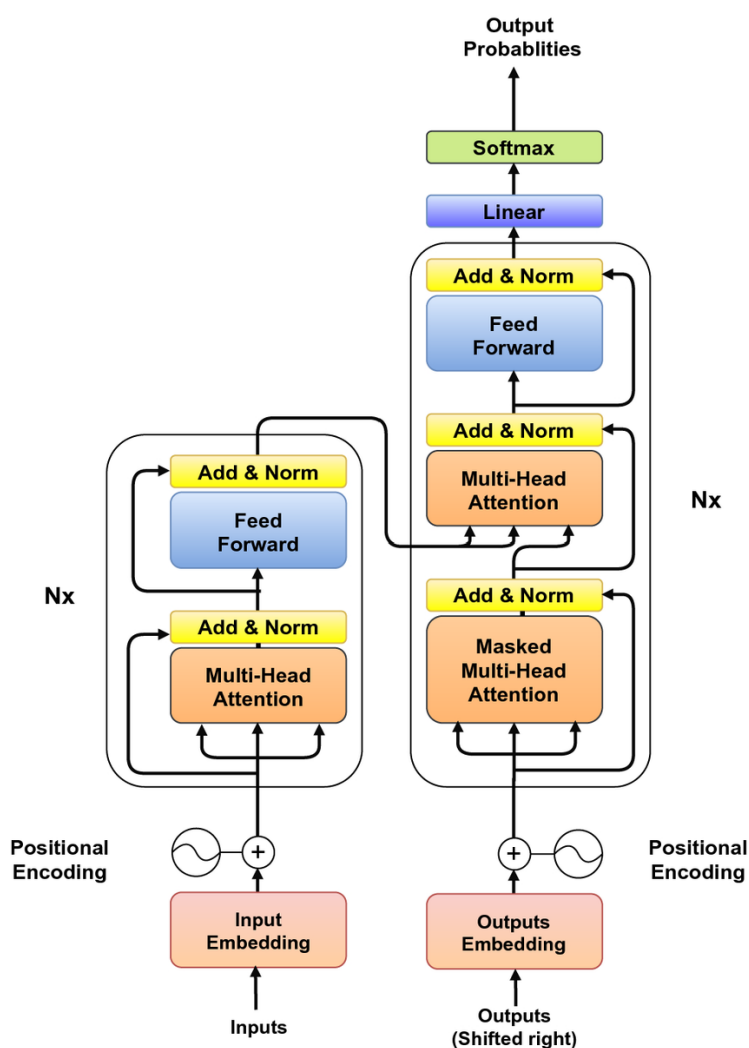
این بردارها ویژگی های کلمه را در فضای عددی نمایش می دهند. روش های مختلفی برای تبدیل متن به بردار وجود دارند. از جمله این روش ها می توان به روش های Word2Vec [۳۴] و GloVe [۴۱] اشاره کرد.

همان طور که در شکل ۲.۳.۲ نشان داده شده است، هر کلمه که به صورت توکن است، ابتدا در دیکشنری تعریف شده پیدا می شود و پس از پیدا شدن در دیکشنری، با استفاده از روش های تعبیه کردن<sup>۱۲</sup>، هر کلمه به برداری از اعداد تبدیل می شود. این جاسازی ها شباهت های معنایی بین کلمات

<sup>۱۰</sup> Encoder

<sup>۱۱</sup> Decoder

<sup>۱۲</sup> Embedding

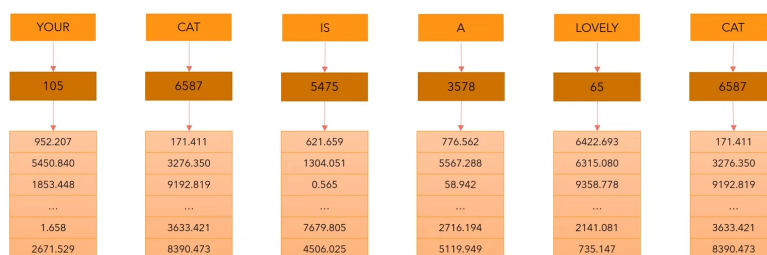


شکل ۲.۳.۱: معماری ترانسفورمرها

را مدل سازی می کنند و کلماتی که از نظر معنایی شبیه به هم هستند، بردار آن ها نیز به یکدیگر نزدیک تر است. به این ترتیب، کلمات برای مدل ها و شبکه های عصبی قابل فهم می شوند [۳۴، ۴۱].

## ۲.۳.۲ جاسازی موقعیتی

هر کلمه را به برداری از اعداد که برای مدل قابل فهم باشد، تبدیل کرده ایم. اما مدل های ترانسفورمر نمی توانند جایگاه هر کلمه را تشخیص دهند. در مدل های ترانسفورمر، برخلاف مدل های بازگشتی، به دلیل اینکه کلمات به صورت موازی وارد می شوند، نیاز داریم تا جایگاه هر کلمه را بدانیم. به طور



شکل ۲.۳.۲:

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. روش های مختلفی برای اضافه کردن جاسازی موقعیتی<sup>۱۳</sup> به مدل وجود دارد. در ترانسفورمرها از روش جاسازی موقعیت سینوسی<sup>۱۴</sup> استفاده می شود [۴۸].

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

و برای مقادیر زوج:

$$PE(pos, 2i) = \sin\left(\frac{pos}{10000^{\frac{2i}{d}}}\right) \quad (2.3.1)$$

و برای مقادیر فرد:

---

Positional Embedding<sup>۱۳</sup>  
 Sinusoidal Positional Embedding<sup>۱۴</sup>  
 pos<sup>۱۵</sup>

$$PE(pos, 2i + 1) = \cos\left(\frac{pos}{10000^{\frac{2i}{d}}}\right) \quad (2.3.2)$$

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

همان‌طور که در شکل [شکل ۲.۳.۳](#) مشاهده می‌کنید، بعد از جاسازی کلمات، به آن جاسازی 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 [۴۸]. این ویژگی به مدل کمک می‌کند تا توالی زمانی را درک کرده و الگوهای زمانی را شبیه‌سازی کند. از مزایای این روش می‌توان به عدم نیاز به آموزش و توزیع متوازن جایگاه کلمات اشاره کرد.

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	58,942	9358,776	9192,819
...	...	...	...	...	...
1,458	3633,421	7679,805	2716,194	2141,081	3633,421
2671,529	8395,473	4556,025	5119,989	735,147	8395,473
+	+	+	+	+	+
...	1664,068	...	...	...	1281,458
...	8680,153	...	...	...	7902,890
...	2630,399	...	...	...	912,970
...	...	...	...	...	3821,102
...	9386,405	...	...	...	1459,217
...	3120,159	...	...	...	7016,620

شکل ۲.۳.۳:



## ۳.۳.۲ توجه

در روش شبکه های بازگشتی، توالی ورودی (مثلاً یک جمله) معمولاً به صورت گام به گام پردازش می شود [۲۰، ۱۳]. اما در ترانسفورمر می خواهیم مدلی داشته باشیم که به هر موقعیت (مثلاً یک کلمه) در توالی نگاه کند و به همه موقعیت های دیگر نیز به صورت موازی دسترسی داشته باشد. به این مفهوم توجه<sup>۱۶</sup> می گوئیم.

به زبان ساده، وقتی توکن (کلمه)  $i$  به توکن های دیگر نگاه می کند، می خواهد بداند کدام توکن ها برای تفسیر معنای خودش مهم ترند.

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

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

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

در ضرب شباهت های توجه<sup>۱۷</sup> [۴۸]، ابتدا شباهت یا ارتباط بین پرسش<sup>۱۸</sup> و کلید<sup>۱۹</sup> را با محاسبه ضرب داخلی<sup>۲۰</sup> به دست می آوریم، سپس آن را نرمال می کنیم (با تقسیم بر  $d_k$ ) و از تابع سافت مکس<sup>۲۱</sup> استفاده می کنیم تا ضرایب توجه<sup>۲۲</sup> را به دست آوریم. در نهایت با همین ضرایب، ترکیبی خطی از بردارهای مقدار<sup>۲۳</sup> را می گیریم.

فرمول به شکل زیر است:

---


$$\begin{aligned} & \text{attention}^{16} \\ & \text{Scaled Dot-Product Attention}^{17} \\ & \quad \text{Query}^{18} \\ & \quad \text{Key}^{19} \\ & \quad \text{Dot Product}^{20} \\ & \quad \text{softmax}^{21} \\ & \quad \text{Attention Weights}^{22} \\ & \quad \text{value}^{23} \end{aligned}$$

$$\text{Attention}(Q, K, V) = \text{softmax} \left( \frac{QK^T}{\sqrt{d_k}} \right) V \quad (2.3.3)$$

که در آن:

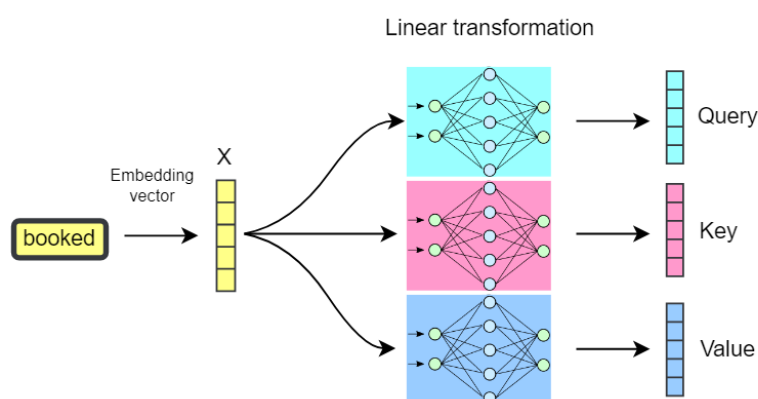
$Q \in \mathbb{R}^{n \times d_k}$  ماتریس پرسش برای

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

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

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

به‌روزرسانی می‌شوند.



شکل ۲.۳.۴:

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

تقسیم بر  $d_k$  باعث می‌شود مقدار ضرب داخلی در ابعاد بالا خیلی بزرگ نشود و شیب‌ها گرایان

پایدار بمانند.

$$\alpha = \text{softmax} \left( \frac{QK^T}{\sqrt{d_k}} \right) \quad (2.3.4)$$

$\alpha$  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 ۲.۳.۵: 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:

multi head attention<sup>۲۴</sup>

head<sup>۲۵</sup>

concatenate<sup>۲۶</sup>

$$\text{head}_i = \text{Attention}(Q_i, K_i, V_i) \quad (2.3.5)$$

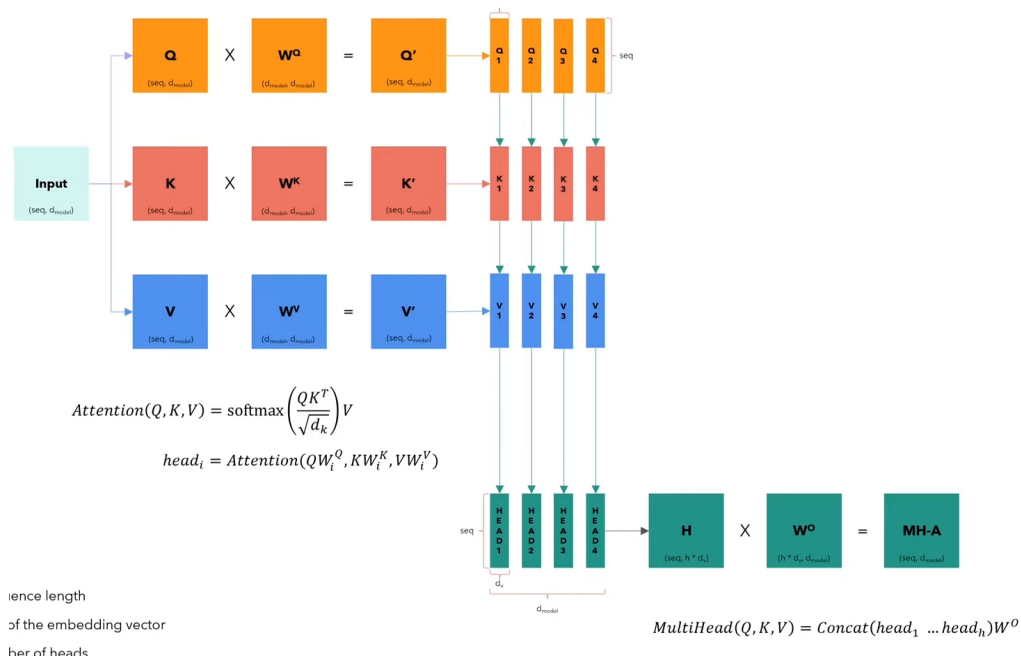
$$\text{MultiHead}(Q, K, V) = [\text{head}_1 \oplus \dots \oplus \text{head}_h] W_O \quad (2.3.6)$$

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 ۲.۳.۶: توجه چند سر

<sup>۲۷</sup>concatenate

## چرا چندین سر؟

مشاهده چند منظر متفاوت: هر سر می‌تواند الگوهای گوناگونی از وابستگی‌ها را بیاموزد (مثلاً یک سر می‌تواند یاد بگیرد کلمه فعلی با کلمات همسایه نزدیک خود بیشتر مرتبط شود، یک سر دیگر 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  $d_{\text{model}} = 512$  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 [۲۰، ۳].

در مبدل‌ها [۴۸]، 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 ۲۸ شبکه‌های عمیق الهام گرفته شده است [۱۷].

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

$$x + \text{SubLayer}(x) \quad (2.3.7)$$

این جمع به صورت عنصر به عنصر ۲۹ انجام می‌شود.

---

Residual Connection<sup>۲۸</sup>  
Element-wise Addition<sup>۲۹</sup>

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

### کمک به جریان یافتن گرادیان

وقتی ورودی مستقیماً به خروجی اضافه می شود، مسیری مستقیم برای عبور شیب (گرادیان) به عقب ایجاد می گردد. در صورت نبود این اتصال، اگر شبکه عمیق شود، گرادیان ها ممکن است در لایه های پایین محو شوند و عملاً ناپدید شدن گرادیان <sup>۳۰</sup> رخ دهد [۲۰، 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 می شود در صورت ناکافی بودن یادگیری زیرماژول یا در مراحل اولیه آموزش، دست کم بخشی از سیگنال (اطلاعات) خام به لایه های بالاتر برسد [۱۷، ۴۸].

### کاهش ریسک تخریب ویژگی ها

در شبکه های عمیق، یکی از مشکلات این است که هر لایه ممکن است بخشی از اطلاعات مفید را تخریب کند. اتصال باقی مانده تضمین می کند که اگر لایه ای به هر دلیل نتوانست الگوی بهینه را یاد بگیرد، اطلاعات قبلی حداقل به صورت دست نخورده تا حدی منتقل می شود.

## ۴.۲ نرمال سازی لایه ها

در یادگیری عمیق، نرمال سازی <sup>۳۲</sup> داده های یک لایه یا فعال سازی ها، اغلب به سرعت بخشیدن به همگرایی و پایدار کردن آموزش کمک شایانی می کند. شاید معروف ترین نوع نرمال سازی، نرمال

<sup>۳۰</sup> gradient vanishing

<sup>۳۱</sup> Residual Connection

<sup>۳۲</sup> Normalization

سازی بچ<sup>۳۳</sup> باشد که پیش‌تر در کارهای بینایی کانولوشنی<sup>۳۴</sup> بسیار مورد استفاده قرار گرفت [۲۲]. نرمال سازی لایه ها<sup>۳۵</sup> روشی جایگزین است که در ترانسفورمر استفاده می‌شود [۴۸، ۱]. علت اصلی این انتخاب، ماهیت توالی‌محور<sup>۳۶</sup> بودن داده‌ها در پردازش زبان طبیعی و عدم تمایل به وابستگی به آمار مینی‌بچ<sup>۳۷</sup> است.

### نرمال سازی بچ

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

### نرمال سازی لایه ها

در نرمال سازی لایه ها، برای هر توکن به صورت جداگانه (در طول بُعد ویژگی)، میانگین<sup>۳۸</sup> و واریانس<sup>۳۹</sup> گرفته می‌شود [۱]. فرض کنید در یک لایه، بردار  $h_i \in \mathbb{R}^{d_{\text{model}}}$  مربوط به توکن  $i$  باشد؛ یعنی ابعاد ویژگی آن  $d_{\text{model}}$  است. ما میانگین  $\mu_i$  و واریانس  $\sigma_i^2$  را از اجزای این بردار محاسبه می‌کنیم:

$$\mu_i = \frac{1}{d_{\text{model}}} \sum_{k=1}^{d_{\text{model}}} h_{i,k}, \quad \sigma_i^2 = \frac{1}{d_{\text{model}}} \sum_{k=1}^{d_{\text{model}}} (h_{i,k} - \mu_i)^2 \quad (2.4.8)$$

سپس نرمال‌سازی برای هر مؤلفه<sup>۳۷</sup>  $k$  در بردار توکن  $i$  به شکل زیر انجام می‌شود:

---

Batch Normalization<sup>۳۳</sup>  
 CNN<sup>۳۴</sup>  
 Layer Normalization<sup>۳۵</sup>  
 sequence<sup>۳۶</sup>  
 mini-batch<sup>۳۷</sup>  
 mean<sup>۳۸</sup>  
 variance<sup>۳۹</sup>

$$\hat{h}_{i,k} = \frac{h_{i,k} - \mu_i}{\sqrt{\sigma_i^2 + \epsilon}} \quad (2.4.9)$$

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:

$$\text{LayerNorm}(h_i) = \gamma \odot \hat{h}_i + \beta \quad (2.4.10)$$

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

### مزایای نرمال سازی لایه در مبدل ها

- 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، ۲۲].
  - 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 [۴۸].
- 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:

---

Internal Covariate Shift<sup>۴۰</sup>



$$z = x + \text{SubLayer}(x) \quad (2.4.11)$$

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 [۴۸].

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.

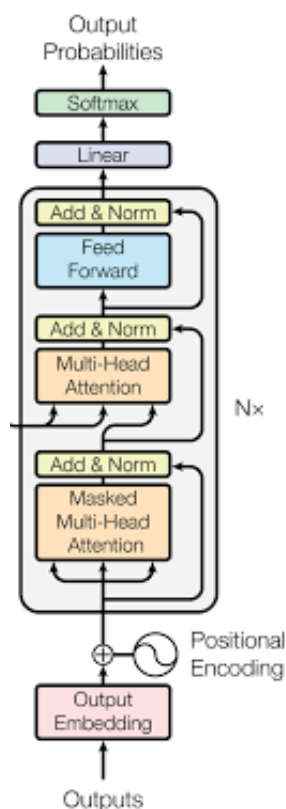
رمزگشا از این ورودی‌ها استفاده می‌کند تا به صورت گام‌به‌گام، خروجی نهایی خود را تولید کند

[۴۵، ۲].

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

---

<sup>۴۱</sup>Target Sequence



شکل ۲.۵.۷:

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

## ۶.۲ توجه چند سری ماسک شده

در مبدل‌ها، مکانیزم توجه چند سری<sup>۴۳</sup> در بخش دیکودر به صورت ماسک شده<sup>۴۴</sup> پیاده‌سازی می‌شود تا مدل نتواند توکن‌های آینده را ببیند و به صورت خودبازگشتی<sup>۴۵</sup> توکن بعدی را پیش‌بینی کند [۴۸]. در واقع ایده اصلی استفاده از ماسک جلوگیری از مشاهده آینده است.

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

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

<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) [۴۵، ۲].

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 [۴۸].

## 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 [۴۸]. حال همین ایده در تصویر پیاده سازی شده است. همان طور که در شکل ۲.۸.۸ 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 [۲۷، ۲۶، ۱۷]، تصویر را به بلاک‌های غیرهم‌پوشان ( $P \times P$ ) تقسیم می‌کنیم. این کار علاوه بر ساده‌سازی موازی‌سازی، به مدل اجازه می‌دهد از سازوکار توجه برای ارتباط بین این بلاک‌ها استفاده کند [۱۱].



شکل ۲.۸.۸: بخش بندی تصاویر

## ۲.۸.۲ شکل پچ‌ها:

فرض کنید ابعاد تصویر ورودی ( $H \times W \times C$ ) باشد. 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 پچ ( $P \times P$ ) باشد (برای نمونه  $16 \times 16$ )، تصویر به صورت یک جدول مشبک از پچ‌های کوچک تقسیم می‌شود. به هر پچ می‌توان مانند یک «کاشی» از تصویر نگاه کرد: - پچ اول: مختصات (در ارتفاع ۱۵ تا ۰) و (در عرض ۱۵ تا ۰)، - پچ 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 پچ‌ها:

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

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

در مجموع:

$$\left(\frac{H}{P}\right) \times \left(\frac{W}{P}\right) = \frac{H}{P} \times \frac{W}{P}. \quad (2.8.12)$$

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 بردار کردن 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:

$$P \times P \times C = P^2 \times C. \quad (2.8.13)$$

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

---

Non-overlapping<sup>۴۹</sup>

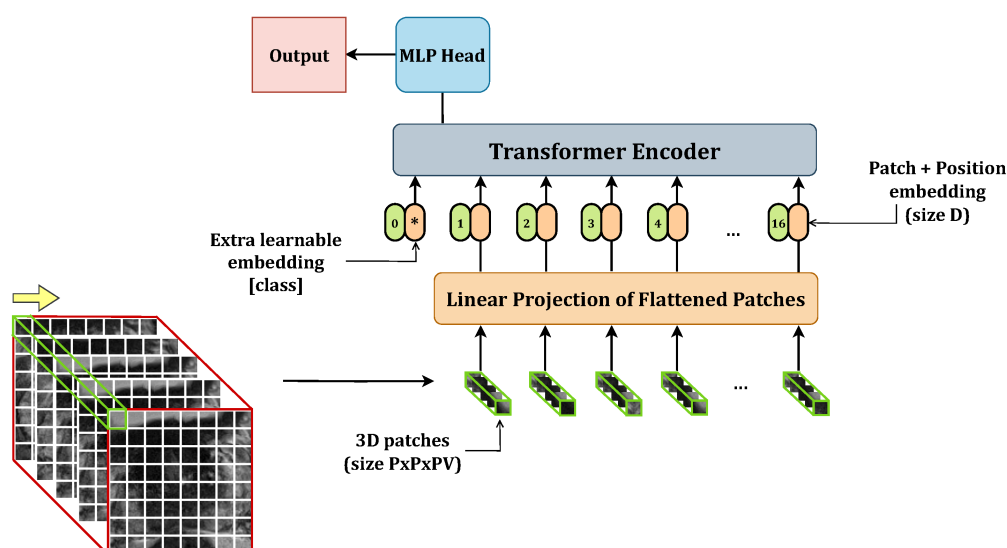
Local Details<sup>۵۰</sup>

Row-major<sup>۵۱</sup>

## ۹.۲ اعمال لایه خطی

بعد از کنار هم چیدن 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 ۲.۹.۹: مبدل‌های بینایی

این مرحله شبیه ساخت توکن در پردازش زبان طبیعی است؛ با این تفاوت که در پردازش زبان طبیعی، توکن «کلمه» یا «زیرکلمه» است و از قبل دارای بردار تعبیه‌شده جاساز شده بوده است [۴۸]. در مبدل‌های 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:

$$\{x_{patch_1}, x_{patch_2}, \dots, x_{patch_N}\}.$$

هر پچ اکنون یک بردار  $d_{model}$  - بعدی است. پس یک مجموعه با طول  $N$  (تعداد پچ ها) و عرض  $d_{model}$  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 [۴۸، 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  $d_{model}$  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

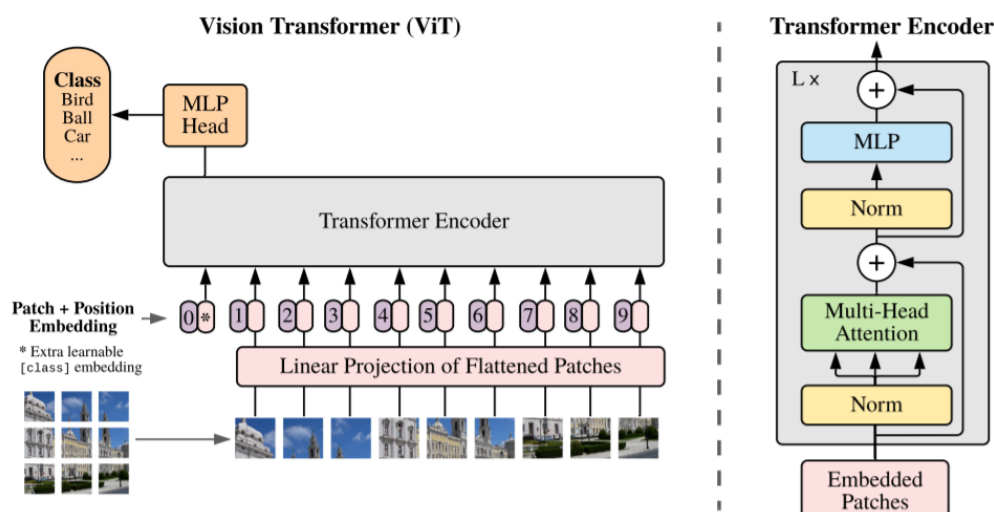
رمزگذار در ترانسفورمرها همانند مبدل اصلی است [۴۸]، با این تفاوت که در مبدل های بینایی [۱۱] دیگر به رمزگشا نمی رویم. پس از عبور از بلاک های ترانسفورمر، در ساده ترین حالت یک لایه خطی<sup>۵۶</sup> یا یک لایه MLP<sup>۵۷</sup> بر روی بردار نهایی اعمال می شود و این لایه ها به تعداد کلاس ها خروجی می دهند. سپس 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 [۴۸]. 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 [۴۸، ۱۱]. در نتیجه، مدل با تعداد لایه های بیشتر اغلب عملکرد بهتری از خود

<sup>۵۶</sup>Fully Connected

<sup>۵۷</sup>Multi-Layer Perceptron

<sup>۵۸</sup>softmax



شکل ۲.۹.۱۰: توکن توجه در مبدل های بینایی

نشان می دهد.

## ۱۰.۲ مبدل پنجره ای متحرک

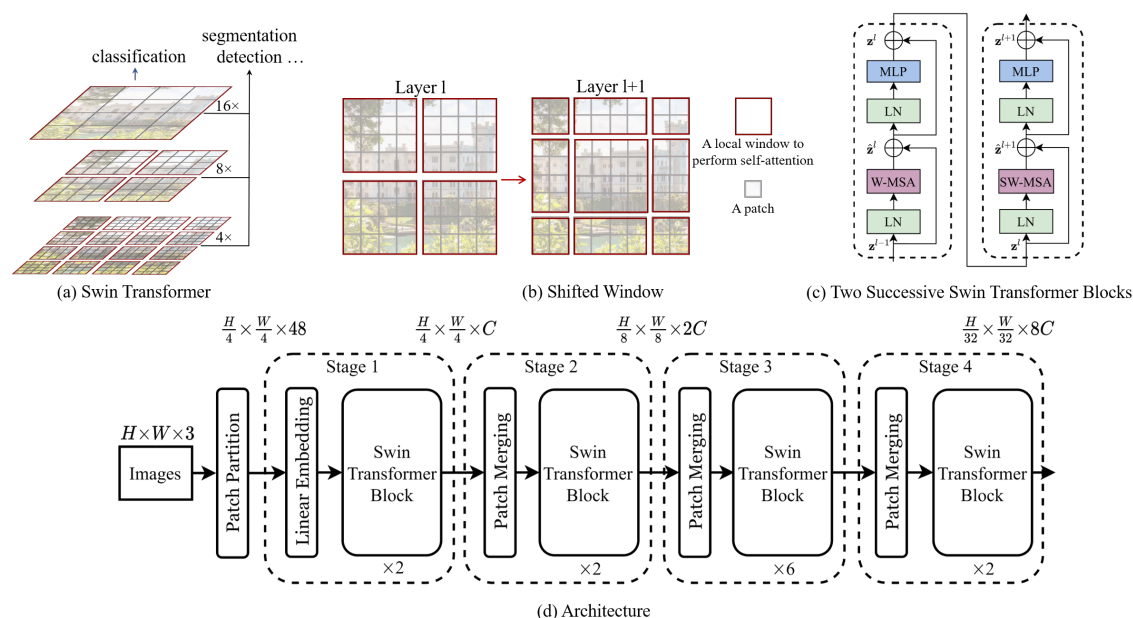
ایده مبدل پنجره ای متحرک<sup>۵۹</sup> از ترکیب چند مفهوم کلیدی در مدل های ترانسفورمر و شبکه های کانولوشنی شکل گرفت [۲۹، ۱۷، ۴۸].

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 کانولوشنی، معماری معمولاً به صورت سلسله مراتبی پیش می رود [۱۷]؛ 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 [۲۹]، 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 ۲.۱۰.۱۱: 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 [۲۹]. 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 [۱۷]. 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

$$Z = X \cdot W_{\text{embed}} + b_{\text{embed}}, \quad Z \in \mathbb{R}^{\left(\frac{H}{P} \cdot \frac{W}{P}\right) \times C}. \quad (2.10.14)$$

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 [۲۹].

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

- 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 ویزگی 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 [۲۹]. اگر فقط از پنجره‌های ثابت (بدون جابه‌جایی) استفاده کنیم، هر بلوک از تصویر تنها با پیکسل‌های همان پنجره در ارتباط خواهد بود و ممکن است اطلاعات نواحی مرزی با نواحی مجاور 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 [۲۹]:

- 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 [۲۹]. 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

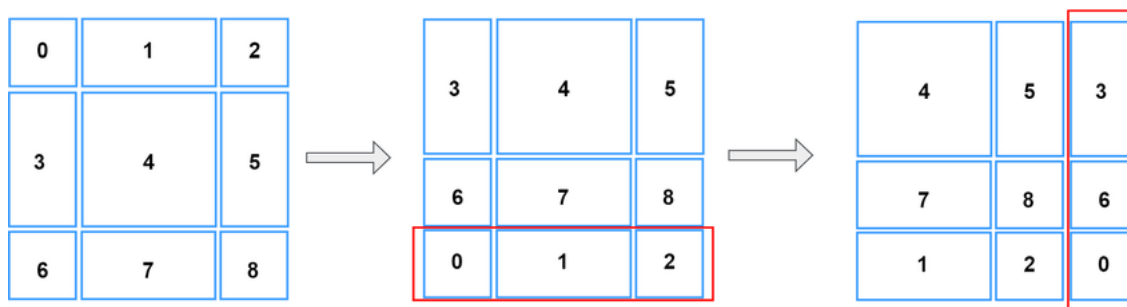
مطابق شکل ۲.۱۰.۱۲، بعد از اینکه بلوک اول (توجه چند سری پنجره ای) کارش تمام شد، در بلوک دوم، قبل از تقسیم بندی به پنجره های  $(M \times M)$ ، نقشه ویژگی را جابه جا می کنیم [۲۹]. در مقاله اصلی، این مقدار جابه جایی معمولاً نیم اندازه پنجره  $\frac{M}{2}$  در راستای افقی و عمودی است. به این ترتیب:

- 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 ۲.۱۰.۱۲ آمده است.



شکل ۲.۱۰.۱۲: 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 [۲۹].

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

[۲۹]:

• 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 [۲۹]. ساختار این MLP به‌صورت زیر است:

$$X' = \text{GELU}(XW_1 + b_1) W_2 + b_2, \quad (2.10.15)$$

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 [۱۸].

## 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 [۱۷].

- 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) [۲۹].

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. [۲۹]:

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:

$$Z' \mapsto Z'' = Z' W_{\text{merge}} + b_{\text{merge}}, \quad (2.10.16)$$

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).

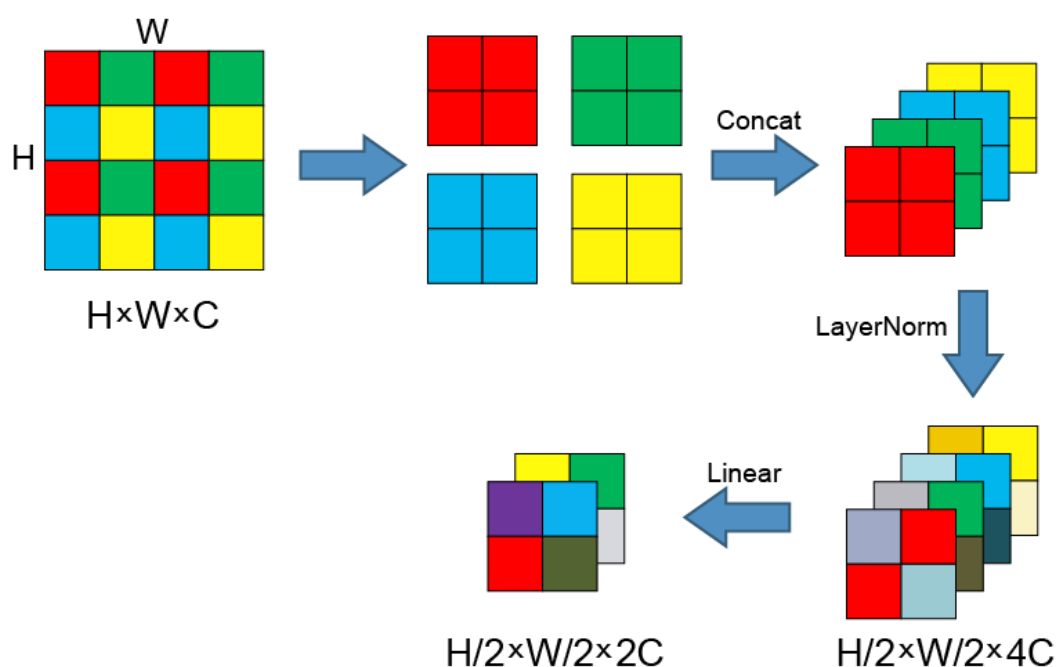
به عبارت دیگر، تعداد نقاط مکانی نصف می‌شود (هم در طول و هم در عرض)، اما کانال از  $C$  به  $2C$  افزایش می‌یابد.

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 [۱۷]. در مبدل پنجره متحرک هم همین ایده سلسله‌مراتب را به دنیای مبدل‌ها آورده است [۲۹]. همچنین اگر ابعاد فضایی را کم نکنیم، هزینه توجه به شدت زیاد می‌شود (چون باید در هر لایه برای همه

---

<sup>۷۵</sup> Pooling

<sup>۷۶</sup> Stride-Convolution



شکل ۲.۱۰.۱۳: 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 [۲۹]. 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

### پیشینه پژوهش

استفاده از روش‌های تانسوری در شبکه‌های عصبی چندلایه (MLP)

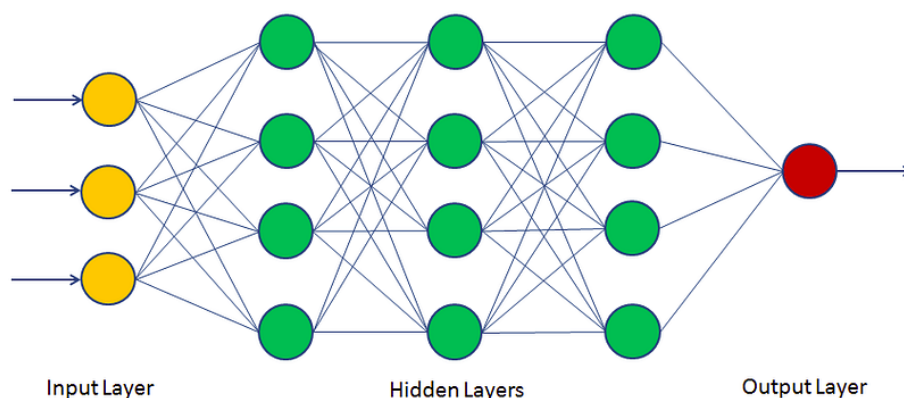
در سال‌های اخیر، استفاده از روش‌های تانسوری به‌عنوان روشی نوین در بهینه‌سازی معماری‌های شبکه‌های عصبی، به‌ویژه در مدل‌هایی که تعداد پارامترهای آن‌ها بسیار زیاد است، مانند شبکه‌های عصبی چندلایه<sup>۱</sup>، توجه بسیاری را به خود جلب کرده است. تانسورها تعمیمی از ماتریس‌ها به ابعاد بالاتر هستند و به‌طور طبیعی برای نمایش داده‌های چندبعدی همچون تصاویر، ویدیوها یا سری‌های زمانی چندکاناله مناسب‌اند. بهره‌گیری از ساختار تانسوری در معماری شبکه، این امکان را فراهم می‌سازد که بدون نیاز به فشردن اولیه (مانند flatten کردن ورودی)، اطلاعات ساختاری میان ابعاد مختلف حفظ شده و مدل بتواند از روابط درون‌تعاملی موجود میان این ابعاد بهره‌برداری نماید. در معماری سنتی، MLP نگاشت از ورودی  $x \in \mathbb{R}^n$  به خروجی  $y \in \mathbb{R}^m$  به‌وسیله ضرب ماتریسی انجام می‌گیرد:

$$y = Wx + b$$

---

Mlp<sup>۱</sup>

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



شکل ۳.۰.۱: Mlp

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

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

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

روش‌هایی نظیر  $\text{TCL}^2$  و  $\text{TRL}^3$  به عنوان نمونه‌هایی از این رویکرد، با بهره‌گیری از تکنیک‌های تجزیه تانسوری همچون Tucker یا CP decomposition، نه تنها باعث کاهش چشمگیر در تعداد پارامترها می‌شوند، بلکه ساختار چندبعدی داده‌ها را نیز حفظ می‌نمایند. این ویژگی به خصوص در مسائل دارای ورودی‌های دارای ساختار فضایی یا زمانی قابل توجه، بسیار حائز اهمیت است.

<sup>۲</sup> layer contraction tensor

<sup>۳</sup> Tensor regression layer

## مزایای استفاده از روش‌های تانسوری

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

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

## محدودیت‌ها و چالش‌ها

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

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

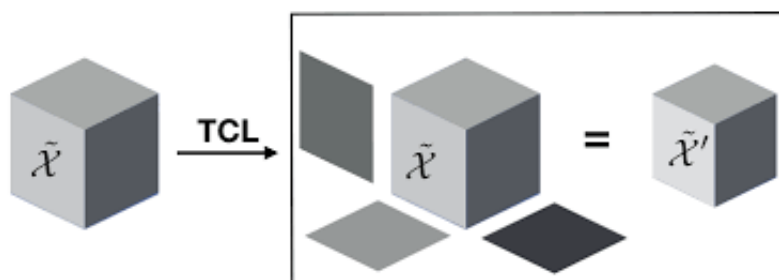


- احتمال کاهش دقت در فشرده‌سازی شدید: در صورتی که میزان فشرده‌سازی تانسورها بیش از حد بالا باشد، مدل ممکن است توانایی لازم برای نمایش روابط غیرخطی و الگوهای پیچیده را از دست داده و در نتیجه، دقت نهایی پیش‌بینی کاهش یابد.

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

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

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



شکل ۳.۰.۲: tensor contraction layer

فرمول‌بندی ریاضی

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

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

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

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

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

عملیات فشرده‌سازی نهایی با ضرب‌های چندحالتی (n-mode product) به صورت زیر انجام می‌شود:

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

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

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

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

تحلیل تعداد پارامترها

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

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

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

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

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

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

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

فرمول بندی ریاضی

فرض شود تانسور ورودی به TRL به صورت زیر باشد:

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

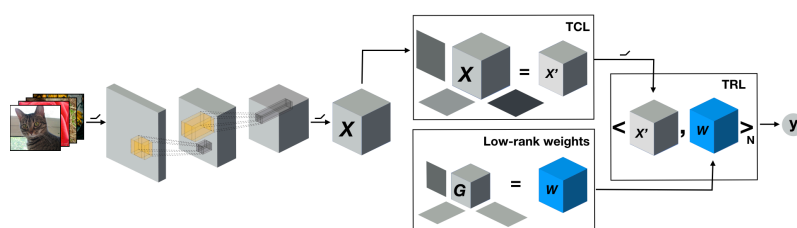
که در آن  $S$  اندازه‌ی دسته‌ی آموزشی (batch size) و  $I_k$  ابعاد تانسور هستند.

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

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

که در آن:

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



شکل ۳.۰.۳: tensor regression network

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

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

که در آن:

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

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

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

یا به صورتی معادل و محاسباتی بهینه تر:

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

تحلیل تعداد پارامترها

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

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

اما در TRL، با در نظر گرفتن تجزیه Tucker، تعداد پارامترها به صورت زیر محاسبه می شود:

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

که معمولاً به طور چشم گیری کمتر از مدل سنتی است، به ویژه زمانی که مقادیر  $R_k$  کوچک تر از  $I_k$  انتخاب شوند.

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

۱. کاهش تعداد پارامترها و حافظه مصرفی

یکی از چالش های اصلی در معماری های مبتنی بر Vision Transformer (ViT)، رشد نمایی تعداد پارامترها در لایه های Fully Connected به ویژه در بخش های انتهایی شبکه است.

با جایگزینی این لایه ها با ساختارهای تانسوری مانند Tensor Contraction Layer (TCL) و Tensor Regression Layer (TRL)، می توان از ساختار چندبعدی داده بهره برده و از طریق

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

افزایش تفسیرپذیری با حفظ ساختار چندخطی داده

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

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

### ۳. کاهش نیاز به داده‌های آموزشی بزرگ

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

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

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

۴.۰.۳ روش تانسوری مبدل پنجره متحرک:

## فصل ۴

### آزمایشات و نتایج



## کتاب‌نامه

- [1] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016. [36](#), [37](#)
- [2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *Proceedings of the 2015 International Conference on Learning Representations (ICLR)*, San Diego, CA, 2015. [22](#), [24](#), [25](#), [38](#), [40](#)
- [3] Yoshua Bengio et al. *Learning long-term dependencies with gradient descent is difficult*. IEEE Transactions on Neural Networks, 1994. [34](#), [35](#)
- [4] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, New York, 2006. [6](#), [7](#), [8](#), [12](#), [13](#), [14](#)
- [5] Peter F Brown, Vincent J. Della Pietra, Stephen A. Della Pietra, and Robert L Mercer. The mathematics of statistical machine translation: Parameter estimation. *Computational linguistics*, 19(2):263–311, 1993. [25](#)
- [6] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995. [12](#)
- [7] Thomas M. Cover and Peter E. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1):21–27, 1967. [11](#)
- [8] Daniel Crevier. *AI: The Tumultuous History of the Search for Artificial Intelligence*. Basic Books, New York, 1993. [3](#), [4](#)

- [9] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018. [24](#), [45](#), [46](#)
- [10] Pedro Domingos and Michael Pazzani. On the optimality of the simple bayesian classifier under zero-one loss. *Machine Learning*, 29(2–3):103–130, 1997. [12](#), [13](#)
- [11] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, et al. An image is worth 16x16 words: Transformers for image recognition at scale, 2020. [41](#), [42](#), [43](#), [44](#), [45](#), [46](#), [47](#), [49](#)
- [12] Richard O. Duda and Peter E. Hart. *Pattern Classification and Scene Analysis*. John Wiley & Sons, New York, 1973. [11](#)
- [13] Jeffrey L. Elman. Finding structure in time. *Cognitive Science*, 14(2):179–211, 1990. [15](#), [25](#), [30](#)
- [14] Edward A. Feigenbaum and Pamela McCorduck. *The Fifth Generation: Artificial Intelligence and Japan's Computer Challenge to the World*. Addison-Wesley, Reading, MA, 1983. [3](#), [5](#)
- [15] Felix A. Gers, Jürgen Schmidhuber, and Fred Cummins. Learning to forget: Continual prediction with lstm. In *Proceedings of the Ninth International Conference on Artificial Neural Networks (ICANN-99)*, pages 850–855, Edinburgh, UK, 1999. [13](#), [15](#), [17](#), [19](#), [20](#)
- [16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, Cambridge, MA, 2016. [6](#), [14](#), [16](#), [17](#), [20](#), [21](#), [22](#)
- [17] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016. [34](#), [35](#), [42](#), [47](#), [48](#), [56](#), [57](#)
- [18] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016. [55](#)

- [19] Sepp Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6(2):107–116, 1998. [15](#), [16](#), [20](#), [21](#)
- [20] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997. [13](#), [16](#), [17](#), [18](#), [20](#), [21](#), [30](#), [34](#), [35](#)
- [21] John Hutchins. *Machine translation: past, present, future*. Ellis Horwood Chichester, 1986. [24](#)
- [22] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456, 2015. [36](#), [37](#)
- [23] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An Introduction to Statistical Learning: with Applications in R*. Springer, New York, 2013. [7](#), [11](#)
- [24] Philipp Koehn. *Statistical Machine Translation*. Cambridge University Press, 2010. [25](#)
- [25] Philipp Koehn, Franz Josef Och, and Daniel Marcu. Statistical phrase-based translation. In *Proc. of NAACL/HLT*, pages 48–54, 2003. [25](#)
- [26] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012. [42](#)
- [27] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. [42](#)
- [28] James Lighthill. *Artificial Intelligence: A General Survey*. HM Stationery Office, London, 1973. Science Research Council Report. [4](#)
- [29] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted

- windows. In *Proc. of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 10012–10022, 2021. [47](#), [48](#), [50](#), [52](#), [53](#), [54](#), [55](#), [56](#), [57](#), [58](#)
- [30] Minh-Thang Luong, Hieu Pham, and Christopher D Manning. Effective approaches to attention-based neural machine translation. In *Proc. of EMNLP*, pages 1412–1421, 2015. [25](#)
- [31] Andrew McCallum and Kamal Nigam. A comparison of event models for naive bayes text classification. In *AAAI-98 Workshop on Learning for Text Categorization*, pages 41–48, Madison, WI, 1998. [13](#)
- [32] John McCarthy, Marvin Minsky, Nathaniel Rochester, and Claude E. Shannon. A proposal for the dartmouth summer research project on artificial intelligence. *Dartmouth College AI Archive*, 1956. [3](#)
- [33] Pamela McCorduck. *Machines Who Think: A Personal Inquiry into the History and Prospects of Artificial Intelligence*. A. K. Peters, Ltd., Natick, MA, 2nd edition, 2004. [5](#)
- [34] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013. [26](#), [27](#)
- [35] Tom M. Mitchell. *Machine Learning*. McGraw-Hill, New York, 1997. [6](#), [11](#), [12](#)
- [36] Douglas C. Montgomery, Elizabeth A. Peck, and Geoffrey G. Vining. *Introduction to Linear Regression Analysis*. John Wiley & Sons, Hoboken, NJ, 6th edition, 2021. [8](#)
- [37] Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective*. MIT Press, Cambridge, MA, 2012. [6](#), [7](#), [11](#), [12](#), [13](#)
- [38] Makoto Nagao. A framework of a mechanical translation between japanese and english by analogy principle. In *Proc. of the international NATO symposium on artificial and human intelligence*, pages 173–180, 1984. [24](#)
- [39] Allen Newell, J. Clifford Shaw, and Herbert A. Simon. Report on a general problem-solving program. In *Proceedings of the International Conference on Information Processing*, pages 256–264, 1959. [3](#)

- [40] Nils J. Nilsson. *The Quest for Artificial Intelligence: A History of Ideas and Achievements*. Cambridge University Press, Cambridge, 2010. [3](#), [4](#), [5](#)
- [41] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *Proc. of EMNLP*, pages 1532–1543, 2014. [26](#), [27](#)
- [42] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. OpenAI report, 2018. [24](#)
- [43] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986. [13](#), [14](#), [16](#), [21](#)
- [44] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson, London, 3rd edition, 2016. [4](#), [5](#)
- [45] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014. [25](#), [38](#), [40](#)
- [46] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 2nd edition, 2018. [9](#)
- [47] Vladimir Vapnik. *Statistical Learning Theory*. Wiley, New York, 1998. [12](#)
- [48] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017. [21](#), [23](#), [24](#), [25](#), [28](#), [29](#), [30](#), [34](#), [35](#), [36](#), [37](#), [38](#), [39](#), [41](#), [44](#), [45](#), [46](#), [47](#)

## پیوست آ

جزئیات مدل‌ها و جدول پارامترها

## Abstract

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

**Key Words:** Transformers , Vision Transformer , Attention , Swin Transformer

.



T. M. U.

## **Vision Trnasformer**

A Thesis Presented for the Degree of  
Master in Computer Science

Faculty of Mathematical Sciences

**Tarbiat Modares University**

by

**Seyed Mohammad Badzohreh**

Supervisor

**Dr. Mansoor Rezghi**

2024