



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

پاییز ۱۴۰۳

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.

سید محمد باذخره

پاییز ۱۴۰۳

## چکیدہ

بيعه عند قفله عقدا خف فدل للقفل فقا

# فهرست مطالب

ه	فهرست جداول
و	فهرست تصاویر
۱	پیش‌گفتار
۲	۱ مفاهیم اولیه
۲	۱.۱ مقدمه
۳	۲ مفاهیم اولیه
۳	۱.۲ مقدمه
۳	۱.۱.۲ آغاز هوش مصنوعی و هدف اصلی
۴	۲.۱.۲ دوره طلایی و پیشرفت‌های اولیه
۴	۳.۱.۲ انتظارات بیش از حد و ظهور عصر تاریک
۴	۴.۱.۲ عوامل اصلی عصر تاریک هوش مصنوعی
۵	۵.۱.۲ پایان عصر تاریک و بازگشت هوش مصنوعی
۶	۲.۲ انواع مدل یادگیری ماشین و شبکه‌های عصبی
۶	۱.۲.۲ یادگیری ماشین: مروری کلی
۶	۲.۲.۲ تقسیم‌بندی‌های اصلی در یادگیری ماشین

۳.۲.۲	یادگیری نظارت شده (Supervised Learning)	۷
۴.۲.۲	یادگیری تقویتی (Reinforcement Learning)	۸
۵.۲.۲	معرفی چند مدل از الگوریتم یادگیری کلاسیک	۸
۶.۲.۲	ماشین بردار پشتیبان (Support Vector Machine, SVM)	۹
۷.۲.۲	بیز ساده (Naive Bayes)	۱۰
۸.۲.۲	شبکه‌های عصبی بازگشتی (RNN) و شبکه‌های حافظه بلندمدت کوتاه‌مدت	
	(LSTM)	۱۱
۹.۲.۲	RNN	۱۱
۱۰.۲.۲	مزایا و معایب RNN	۱۲
۱۱.۲.۲	شبکه‌های حافظه بلندمدت-کوتاه‌مدت (LSTM)	۱۳
۱۲.۲.۲	ظهور LSTM	۱۴
۳.۲	اختار LSTM: نوآوری در مقایسه با RNN	۱۴
۱.۳.۲	وضعیت سلولی (Cell State)	۱۵
۲.۳.۲	دروازه‌ها (Gates)	۱۵
۳.۳.۲	به‌روزرسانی وضعیت سلولی	۱۶
۴.۳.۲	مشکلات کلی RNN و LSTM و ظهور ترانسفورمرها	۱۷
۳	پیشینه پژوهش	۲۱
۱.۳	مقدمه	۲۱
۲.۳	مشکلات ترجمه ماشینی و ترانسفورمرها:	۲۱
۳.۳	ظهور ترانسفورمرها:	۲۲
۴.۳	معماری ترانسفورمرها:	۲۲
۱.۴.۳	Embedding:	۲۳

۲۴	embedding: positional	۲.۴.۳
۲۶	attention:	۳.۴.۳
۳۰	(Add): Connection Residual	۴.۴.۳
۳۰	مزایای Connection Residual در ترانسفورمر	۵.۴.۳
۳۱	(Norm): Normalization Layer	۵.۳
۳۴	decoder:	۶.۳
۳۴	attention head multi masked	۷.۳
۳۵	attention: mask مثال عددی	۸.۳
۳۶	transformer: vision	۹.۳
۳۷	transformer: vision in embedding patch	۱.۹.۳
۳۷	شکل پیچ ها:	۲.۹.۳
۳۹	تعداد پیچ ها:	۳.۹.۳
۴۰	بردار کردن هر پیچ	۴.۹.۳
۴۰	اعمال لایه خطی: (Projection)	۱۰.۳
۴۲	Token: CLS	۱.۱۰.۳
۴۳	encoder: transformer vision	۲.۱۰.۳
۴۴	Transformer: Swin	۱۱.۳
۴۶	قطعه بندی پیچ (Patch): (Partition)	۱.۱۱.۳
۴۶	Embedding: Linear	۲.۱۱.۳
۴۷	Self-Attention: Multi-Head Window	۳.۱۱.۳
۴۸	Attention:	۴.۱۱.۳
۴۹	Windows: shifted	۵.۱۱.۳
۵۲	Mlp:	۶.۱۱.۳

۵۳ ..... merging: patch ۷.۱۱.۳

۵۷ ۴ روش های پیشنهادی

۵۸ ۵ آزمایشات و نتایج

۵۹ کتاب نامه

۶۲ آ جزئیات مدل ها و جدول پارامترها



## فهرست جداول

۱۷	۲.۳.۱ مقایسه ویژگی‌های LSTM و RNN . . . . .
----	---

## فهرست تصاویر

۲۳	.....	۳.۴.۱ معماری ترانسفورمرها
۲۴	.....	embedding word ۳.۴.۲
۲۵	.....	embedding word ۳.۴.۳
۲۶	.....	embedding word ۳.۴.۴
۲۸	.....	Attention ۳.۴.۵
۲۹	.....	attention head multi ۳.۴.۶
۳۵	.....	Decoder ۳.۶.۷
۳۷	.....	patch to iamge ۳.۹.۸
۳۸	.....	Image original ۳.۹.۹
۳۹	.....	Image patch ۳.۹.۱۰
۴۱	.....	Transformer Vision in Embedding ۱۰.۱۱
۴۴	.....	Transformer Vision in Token Cls ۱۰.۱۲
۴۵	.....	Transformer Swin ۱۱.۱۳
۵۰	.....	Window Shifted vs Window ۱۱.۱۴
۵۱	.....	Shift cycle ۱۱.۱۵
۵۵	.....	Merging Patch ۱۱.۱۶

## پیش گفتار

قدشتمقدکنقصدبثقلدقفخدلqxفادخفادخ

# فصل ۱

## مفاهیم اولیه

در این فصل به معرفی مقدمات و مفاهیم مورد نیاز در این پایان نامه می پردازیم.

### ۱.۱ مقدمه

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

## فصل ۲

# مفاهیم اولیه

در این فصل به معرفی مقدمات و مفاهیم مورد نیاز در این پایان نامه می پردازیم.

### ۱.۲ مقدمه

در این بخش به تاریخچه هوش مصنوعی، دستاوردهای اولیه، چالش ها، دلایل رکود هوش مصنوعی و پایان عصر تاریک هوش مصنوعی صحبت می کنیم.

#### ۱.۱.۲ آغاز هوش مصنوعی و هدف اصلی

هوش مصنوعی به عنوان شاخه ای از علوم کامپیوتر، در دهه ۱۹۵۰ با هدف ساخت سیستم ها و ماشین هایی که توانایی تقلید از هوش انسانی را دارند، آغاز شد. نخستین بار، مکاری در سال ۱۹۵۶ این اصطلاح را به کار گرفت [۱۷] و هوش مصنوعی به عنوان علمی که در آن به مطالعه الگوریتم هایی برای تقلید رفتار انسانی می پردازد، شناخته شد. اهداف اولیه هوش مصنوعی شامل توانایی درک زبان، یادگیری، حل مسئله و تولید موجودات هوشمند بود. در این دوران پروژه های تحقیقاتی زیادی

به امید دستیابی به هوش مصنوعی عمومی (AGI, Artificial General Intelligence) شروع به کار کردند [۵، ۲۳].

## ۲.۱.۲ دوره طلایی و پیشرفت‌های اولیه

در دهه ۵۰ و ۶۰ میلادی، هوش مصنوعی به عنوان یکی از پرچمداران پژوهش‌های نوین شناخته می‌شد. الگوریتم‌های اولیه با تکیه بر روش‌های منطقی و ریاضیاتی برای حل مسئله و بازی‌های ساده توسعه یافتند؛ مانند انواع الگوریتم‌های جستجوی درختی که در این دوره به وجود آمدند و زمینه‌ساز اولین دستاوردهای هوش مصنوعی در بازی‌های تخته‌ای همچون شطرنج شدند [۲۲]. در این دوران، پیشرفت‌های بیشتری در پردازش زبان طبیعی (NLP) و سیستم‌های خبره (Expert Systems) نیز صورت گرفت که این امید را در دانشمندان و محققان تقویت کرد که دستیابی به هوش مصنوعی عمومی به زودی ممکن خواهد بود [۹].

## ۳.۱.۲ انتظارات بیش از حد و ظهور عصر تاریک

با وجود پیشرفت‌های هوش مصنوعی، محدودیت‌های تکنولوژی (مثل عدم وجود GPUهای پر قدرت در آن زمان) و همچنین کمبود داده‌های کافی برای آموزش مدل‌های پیچیده‌تر، باعث شد که بسیاری از پروژه‌های تحقیقاتی نتوانند به نتایج پیش‌بینی شده دست یابند. در نتیجه، هوش مصنوعی در دهه ۷۰ به مرحله‌ای از رکود وارد شد که به آن عصر تاریک هوش مصنوعی یا AI Winter می‌گویند [۵، ۱۵]. 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.

## ۴.۱.۲ 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 [۲۳].

● کمبود داده‌ها: در آن زمان، دسترسی به داده‌های کافی برای آموزش مدل‌های پیچیده ممکن نبود و الگوریتم‌های موجود به داده‌های بیشتری نیاز داشتند تا بتوانند به‌درستی آموزش ببینند و عملکرد مطلوبی داشته باشند [۵].

● 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 [۲۵].

## ۵.۱.۲ 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 (Expert Systems) به پایان رسید [۹]. سیستم‌های خبره به عنوان یکی از اولین تلاش‌های موفق برای کاربردهای صنعتی در هوش مصنوعی به‌وجود آمدند. بر خلاف الگوریتم‌های اولیه، این سیستم‌ها از پایگاه بزرگ قواعد و قوانین (Rule-Based Systems) استفاده می‌کردند. در سیستم‌های خبره، به جای تلاش برای شبیه‌سازی کلی هوش مصنوعی، بر حل مسائل تخصصی برای صنایع و سازمان‌ها تمرکز می‌شد. برای مثال، سیستم‌های خبره در پزشکی برای تشخیص بیماری‌ها و پیشنهاد درمان، در صنعت برای مدیریت و پیش‌بینی خرابی ماشین‌آلات، و در امور مالی برای تحلیل و ارزیابی ریسک کاربرد داشتند [۱۸].

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 به‌وجود 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 [۲۳].

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 [۲۵].

## ۲.۲ انواع مدل یادگیری ماشین و شبکه‌های عصبی

یادگیری ماشین و شبکه‌های عصبی در سال‌های اخیر مورد توجه بسیاری قرار گرفته‌اند و در حوزه‌های متنوعی از جمله پردازش تصویر، پردازش زبان طبیعی و داده‌کاوی استفاده می‌شوند [۲، ۱۹، ۲۱].

### ۱.۲.۲ یادگیری ماشین: مروری کلی

یادگیری ماشین (Machine Learning) شاخه‌ای از هوش مصنوعی است که به مدل‌های محاسباتی این امکان را می‌دهد الگوها را از داده‌ها به شکل خودکار یاد بگیرند و بتوانند تصمیم‌گیری کنند [۱۱، ۱۹]. در واقع، هدف یادگیری ماشین این است که مدل‌ها بتوانند از داده‌ها الگوها و روابط پنهان را استخراج کنند و به نتایج و تصمیم‌های قابل اعتماد دست یابند.

### ۲.۲.۲ تقسیم‌بندی‌های اصلی در یادگیری ماشین

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

- یادگیری با نظارت (Supervised Learning)
- یادگیری بدون نظارت (Unsupervised Learning)
- یادگیری تقویتی (Reinforcement Learning)



این طبقه‌بندی در بسیاری از کتاب‌ها و مراجع مهم یادگیری ماشین مطرح شده است [۲، ۲۱].

### ۳.۲.۲ یادگیری نظارت‌شده (Supervised Learning)

یادگیری نظارت‌شده یکی از رایج‌ترین روش‌ها در یادگیری ماشین شناخته می‌شود که در آن از مجموعه داده‌های برچسب‌گذاری‌شده برای آموزش مدل استفاده می‌کنیم [۱۴]. هدف این الگوریتم تشخیص الگوها در میان داده‌های ورودی است تا بتواند روی داده‌های جدید پیش‌بینی یا طبقه‌بندی انجام دهد. این نوع شامل دو دسته الگوریتم Regression و Classification می‌شود.

#### طبقه‌بندی (Classification)

طبقه‌بندی یکی از مهم‌ترین و اصلی‌ترین وظایف در یادگیری نظارت‌شده است که هدف آن تخصیص هر داده به یک لیبل مشخص است [۲]. در این روش، مدل با داده‌های برچسب‌دار (Label) آموزش می‌بیند و یاد می‌گیرد که داده‌های جدید را بر اساس الگوها و ویژگی‌هایی که در داده‌های آموزشی دیده است، به دسته مناسب اختصاص دهد. از کاربردهای طبقه‌بندی می‌توان به تشخیص اسپم (Spam Detection)، تشخیص بیماری (مثلاً آیا یک فرد مبتلا به بیماری هست یا نه) و تشخیص چهره اشاره کرد [۲۱].

#### رگرسیون (Regression)

رگرسیون یکی از مهم‌ترین وظایف یادگیری ماشین است و هدف آن پیش‌بینی مقادیر پیوسته است [۲۰]. بر خلاف طبقه‌بندی که خروجی آن یک دسته‌بندی مجزا است، در رگرسیون خروجی یک مقدار پیوسته خواهد بود و مدل می‌آموزد روابط بین متغیرهای مستقل و متغیر هدف را شناسایی کند. از کاربردهای رگرسیون می‌توان به پیش‌بینی قیمت مسکن یا پیش‌بینی آب‌وهوا اشاره کرد.

## ۴.۲.۲ یادگیری تقویتی (Reinforcement Learning)

یادگیری تقویتی، نوعی یادگیری بر پایه پاداش و تنبیه است که در آن مدل با محیط تعامل می‌کند و بر اساس پاداش یا تنبیه یاد می‌گیرد [۲۶]. 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 (Games)، کنترل رباتیک (Robotic Control) و سیستم‌های توصیه‌گر (Recommender Systems) اشاره کرد.

## ۵.۲.۲ معرفی چند مدل از الگوریتم یادگیری کلاسیک

نزدیک‌ترین همسایه (k-Nearest Neighbors, kNN)

الگوریتم kNN یکی از روش‌های ساده و درعین حال کارآمد در یادگیری نظارت‌شده است که هم در دسته‌بندی و هم در رگرسیون کاربرد دارد [۴، ۷، ۱۹]. 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 [۴].
- عملکرد خوب در داده‌های با تعداد ویژگی کم: در مسائلی که تعداد ویژگی‌ها کم است، این الگوریتم اغلب به خوبی عمل می‌کند [۱۴].

معایب:

- حساسیت به داده‌های پرت: نقاط پرت می‌توانند به‌طور قابل توجهی بر نتایج تأثیر بگذارند [۷].
- کندی در داده‌های بزرگ: این الگوریتم نیاز به محاسبه فاصله برای هر نقطه جدید دارد و در داده‌های بزرگ بار محاسباتی بالایی خواهد داشت [۱۹].
- عدم کارایی در داده‌های با ابعاد بالا: در داده‌هایی با تعداد ویژگی‌های زیاد، کارایی الگوریتم کاهش می‌یابد [۲۱].

## ۶.۲.۲ ماشین بردار پشتیبان (Support Vector Machine, SVM)

الگوریتم SVM با یافتن یک ابرصفحه بهینه، داده‌ها را به کلاس‌های مختلف تقسیم می‌کند [۲۷، ۳]. این الگوریتم یک ابرصفحه به دست می‌آورد که هدف آن حداکثر کردن فاصله میان داده‌های دو کلاس است و به این ترتیب می‌تواند طبقه‌بندی دقیقی داشته باشد.

مزایا:

- توانایی مقابله با داده‌های پیچیده و ابعاد بالا: SVM می‌تواند به خوبی با داده‌های چندبعدی و پیچیده کار کند [۲۷].
- مقاومت در برابر بیش‌برازش (Overfitting): با استفاده از هسته‌ها (kernels)، داده‌های غیرخطی نیز به فضای بالاتر برده می‌شوند و جداسازی بهتری انجام می‌شود [۳].

معایب:

- پیچیدگی محاسباتی: آموزش SVM به دلیل نیاز به حل مسائل بهینه‌سازی، در حجم‌های بالای داده محاسباتی زمان‌بر است [۲۱].

- کارایی پایین در داده‌های پرت: در صورتی که داده‌ها شامل نقاط پرت زیادی باشند، دقت مدل کاهش می‌یابد [۲].

## ۷.۲.۲ بیز ساده (Naive Bayes)

بیز ساده مبتنی بر قضیه بیز است و فرض می‌کند ویژگی‌ها به صورت شرطی مستقل از هم هستند [۶، ۱۹]. این مدل برای اولین بار در حوزه پردازش متن به کار رفت و هنوز هم در بسیاری از کاربردها مانند طبقه‌بندی ایمیل و تحلیل احساسات مورد استفاده قرار می‌گیرد [۱۶]. در Naive Bayes 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 [۲].

مزایا:

- سرعت بالا: به دلیل محاسبات ساده و فرض استقلال ویژگی‌ها، Naive Bayes بسیار سریع و کم حجم است [۱۶].
- کارایی در داده‌های کوچک: حتی با داده‌های کم، این الگوریتم عملکرد نسبتاً خوبی دارد [۲۱].

معایب:

- فرض استقلال ویژگی‌ها: فرض استقلال ویژگی‌ها ممکن است در بسیاری از مسائل واقعی صادق نباشد و این می‌تواند دقت مدل را کاهش دهد [۶].
- حساسیت به داده‌های نادرست: در صورت وجود داده‌های نادرست یا پرت، مدل ممکن است دقت کمتری داشته باشد [۲].

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

شبکه‌های عصبی بازگشتی (RNN) و مدل‌هایی با حافظه بلندمدت - کوتاه‌مدت (LSTM) با هدف پردازش داده‌های ترتیبی و وابسته به زمان توسعه یافتند [۲۴، ۱۳]. 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 [۱۰].

## ۹.۲.۲ RNN

مدل‌های اولیه شبکه‌های عصبی، مانند شبکه‌های چندلایه (MLP)، قادر به پردازش داده‌های مستقل و ثابت بودند و نمی‌توانستند وابستگی‌های زمانی را یاد بگیرند [۲]. در بسیاری از مباحث دنیای واقعی مانند تحلیل متن و صدا، داده‌ها به توالی خاصی وابسته هستند. به همین دلیل، شبکه‌های RNN معرفی شدند تا بتوانند از اطلاعات پیشین در پردازش داده‌های بعدی استفاده کنند [۲۴].

### ساختار و عملکرد RNN

شبکه‌های RNN 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 [۱۱]:

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

در اینجا:

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

•  $W$  وزن‌هایی است که به ورودی  $x_t$  اعمال می‌شود.

• 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.

## ۱۰.۲.۲ مزایا و معایب RNN

در این قسمت به مزایا و معایب شبکه‌های RNN می‌پردازیم.

مزایا:

• حفظ وابستگی زمانی: RNN قادر به پردازش توالی‌های طولانی است و می‌تواند اطلاعات را در طول توالی به خاطر بسپارد [۸].

• کاربردهای گسترده در داده‌های ترتیبی: این مدل در تحلیل زبان طبیعی، پیش‌بینی سری‌های زمانی و پردازش صوت بسیار موفق عمل می‌کند [۱۰].

معایب:

• مشکل ناپدید شدن و انفجار گرادیان (Vanishing and Exploding Gradient):

در فرایند آموزش با روش پس‌انتشار، اگر توالی داده‌ها طولانی باشد، گرادیان‌ها ممکن است بسیار کوچک یا بزرگ شوند که منجر به ناپایداری در آموزش و کاهش دقت می‌شود [۱۲].

• محدودیت در پردازش توالی‌های بسیار بلند: RNN در حفظ اطلاعات طولانی‌مدت با مشکل مواجه است و برای پردازش وابستگی‌های طولانی، عملکرد ضعیفی دارد [۱۱، ۱۳].

## ۱۱.۲.۲ شبکه‌های حافظه بلندمدت - کوتاهمدت (LSTM)

### علل پیدایش LSTM

شبکه‌های LSTM به عنوان یک راه‌حل برای یکی از بزرگ‌ترین مشکلات شبکه‌های عصبی بازگشتی (RNN) معرفی شدند [۱۳]. یکی از برجسته‌ترین مشکلات موجود در RNN ها، معضل ناپدید شدن گرادیان (Vanishing Gradient) بود که مانع یادگیری وابستگی‌های بلندمدت می‌شد [۱۱، ۱۲]. برای درک عمیق‌تر این مسأله، ابتدا به توضیح مشکل ناپدید شدن گرادیان و سپس راهکار LSTM می‌پردازیم.

### Gradient Vanishing

شبکه‌های RNN برای پردازش داده‌های ترتیبی از حلقه‌های بازگشتی بهره می‌برند. در فرایند آموزش RNN، از الگوریتم پس‌انتشار خطا از طریق زمان (Backpropagation Through Time, BPTT) استفاده می‌شود که گرادیان‌ها را جهت به‌روزرسانی وزن‌ها محاسبه می‌کند [۲۴]. با این حال، RNN ها 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 معروف است [۱۲].

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

$$\frac{\partial L}{\partial W} = \prod_{k=1}^t \frac{\partial h_k}{\partial h_{k-1}} \cdot \frac{\partial h_t}{\partial L},$$

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

- تأثیر مستقیم بر وزن‌ها: زمانی که گرادیان‌ها به صفر نزدیک می‌شوند، وزن‌های مدل عملاً به‌روزرسانی نمی‌شوند و این امر مانع از یادگیری وابستگی‌های طولانی‌مدت در داده‌ها می‌شود [۱۱].

## ۱۲.۲.۲ ظهور LSTM

در سال ۱۹۹۷، Jürgen Schmidhuber و Sepp Hochreiter شبکه‌های حافظه بلندمدت - کوتاهمدت (LSTM) را معرفی کردند [۱۳]. انگیزه اصلی توسعه LSTM حل مشکل ناپدید شدن گرادیان در شبکه‌های RNN بود. این مشکل در مسائل یادگیری داده‌های ترتیبی طولانی مانع می‌شد RNN وابستگی‌های بلندمدت را به‌درستی فراگیرد.

راه‌حل LSTM برای پایداری جریان گرادیان‌ها

LSTM با معرفی معماری جدید در شبکه‌های بازگشتی، جریان گرادیان‌ها را در طول توالی پایدار نگه می‌دارد. این کار از طریق اضافه کردن وضعیت سلولی (Cell State) و دروازه‌ها (Gates) به ساختار RNN انجام می‌شود [۱۰]. این اجزا به LSTM امکان می‌دهند:

۱. اطلاعات غیرضروری را فراموش کند،

۲. اطلاعات مهم جدید را اضافه کند،

۳. اطلاعات مهم قبلی را حفظ کند.

## ۳.۲ اختار LSTM: نوآوری در مقایسه با RNN

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



### ۱.۳.۲ وضعیت سلولی (Cell State)

مسیر اصلی ذخیره اطلاعات در LSTM است که می‌تواند اطلاعات مهم را در طول توالی حفظ کند. برخلاف RNN که عمدتاً بر خروجی‌های بازگشتی  $h_t$  متکی است، LSTM یک مسیر جداگانه برای عبور اطلاعات از وضعیت سلولی دارد که به حفظ گرادیان‌ها کمک شایانی می‌کند [۱۳].

### ۲.۳.۲ دروازه‌ها (Gates)

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

- دروازه فراموشی (Forget Gate): تعیین می‌کند چه اطلاعاتی از وضعیت سلولی باید حذف شود [۱۰]:

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

, میزان فراموشی برای هر عنصر از وضعیت سلولی :  $f_t$

. تابع سیگموید (خروجی بین ۰ و ۱) :  $\sigma$

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

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

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

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

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

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

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

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

### ۳.۳.۲ به‌روزرسانی وضعیت سلولی

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

$$C_t = f_t \cdot C_{t-1} + i_t \cdot \tilde{C}_t.$$

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

### علت پایداری گرادیان در LSTM

- حذف ضرب‌های مکرر: برخلاف RNN که به ضرب‌های مکرر وزن‌ها و گرادیان‌ها وابسته است، LSTM با مسیر جداگانه وضعیت سلولی، از کاهش نمایی گرادیان جلوگیری می‌کند [۱۲].

- استفاده از توابع سیگموئید و تانژانت هیپربولیک: توابع سیگموئید در دروازه‌ها و تانژانت هیپربولیک در وضعیت سلولی مقادیر را محدود می‌کنند و مانع از انفجار گرادیان می‌شوند [۱۱، ۱۰].

- مدیریت اطلاعات توسط دروازه‌ها: دروازه‌های فراموشی و ورودی به مدل اجازه می‌دهند تنها اطلاعات مهم حفظ شود و داده‌های غیرضروری حذف شوند؛ این موضوع از پیچیدگی محاسباتی غیرضروری جلوگیری می‌کند [۱۳].

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

جدول ۲.۳.۱: مقایسه ویژگی‌های LSTM و RNN

## ۴.۳.۲ مشکلات کلی RNN و LSTM و ظهور ترنسفورمرها

شبکه‌های بازگشتی RNN و LSTM توانستند بسیاری از مشکلات و محدودیت‌های مدل‌های اولیه را حل کنند؛ اما همچنان با چالش‌ها و محدودیت‌هایی مواجه بودند که در مسائل پیچیده‌تر، مانند ترجمه زبان یا تحلیل داده‌های بلندمدت و حجیم، مشکلات جدی ایجاد می‌کردند [۱۱، ۱۳]. این مشکلات در نهایت به پیدایش ترنسفورمرها (Transformers) منجر شد [۲۸]. در ادامه، مهم‌ترین محدودیت‌های RNN و LSTM مورد بررسی قرار می‌گیرند.

### مشکل وابستگی ترتیبی در RNN و LSTM‌ها

RNN و LSTM‌ها داده‌ها را به صورت ترتیبی پردازش می‌کنند؛ به این معنی که برای پردازش داده‌های گام زمانی  $t$ ، باید تمامی داده‌های قبلی  $(t - 1)$  پردازش شده باشند [۱۳، ۲۴]. این ویژگی مشکلات زیر را ایجاد می‌کند:

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

محدودیت در یادگیری وابستگی‌های بسیار طولانی

با وجود پیشرفت LSTM در یادگیری وابستگی‌های بلندمدت نسبت به RNN‌های معمولی، این مدل‌ها همچنان در یادگیری وابستگی‌های بسیار بلند، مانند ارتباط بین کلمات در جملات دور از هم یا درک ساختار کلی یک متن، محدودیت دارند [۱۲]:

- مشکل در داده‌های بسیار طولانی: حتی در LSTM نیز ظرفیت حفظ اطلاعات محدود است و با افزایش طول توالی، دقت مدل افت می‌کند.

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

پیچیدگی محاسباتی و حافظه

LSTM‌ها به علت ساختار پیچیده‌ای که شامل چندین ماتریس ضرب (برای دروازه‌های فراموشی، ورودی و خروجی) و به‌روزرسانی وضعیت سلول است، به حافظه و محاسبات زیادی نیاز دارند [۱۱]:

- نیاز به حافظه بیشتر: برای ذخیره وضعیت سلولی و گرادیان‌ها، LSTM‌ها 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.

مشکل پردازش وابستگی‌های غیرمتوالی

RNN‌ها و LSTM‌ها 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 [۱].

به عنوان مثال، در جمله‌ای طولانی ممکن است کلمه‌ای در ابتدای جمله با کلمه‌ای در انتهای جمله ارتباط معنایی داشته باشد. RNN ها و LSTM ها برای یادگیری این گونه وابستگی ها محدودیت دارند.

گرادیان‌های ناپایدار و مشکلات بهینه‌سازی

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

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

نیاز به مدلی با ظرفیت بیشتر و سرعت بالاتر

- مدل‌های بزرگ‌تر: برای مسائل پیچیده‌تر، به مدل‌هایی با تعداد پارامتر بالاتر نیاز است؛ اما RNN ها و LSTM ها به دلیل محدودیت در حافظه و پردازش، پاسخگوی این نیاز نیستند.
- کارایی در داده‌های چندوجهی (Multimodal): برای داده‌هایی که ترکیبی از اطلاعات متنی، صوتی و تصویری هستند، RNN ها و LSTM ها توانایی لازم جهت پردازش موازی این اطلاعات را ندارند.

در مجموع، وابستگی ترتیبی در RNN و LSTM مانعی اساسی برای استفاده از این مدل‌ها در مسائل پیچیده و بزرگ بود که در نهایت به ظهور ترنسفورمرها منتهی شد [۲۸]. ترنسفورمرها با طراحی مبتنی بر موازی‌سازی و مکانیزم توجه (Attention Mechanism)، این محدودیت را برطرف کرده و راه‌حلی کارآمدتر برای پردازش داده‌های ترتیبی ارائه دادند.

## فصل ۳

### پیشینه پژوهش

#### ۱.۳ مقدمه

ظهور مدل‌های Transformer و انقلاب در یادگیری عمیق، یکی از تحولات اساسی در حوزه پردازش زبان طبیعی (NLP) و یادگیری ماشین به شمار می‌رود. این مدل‌ها باعث تغییرات عمده‌ای در نحوه ساخت و آموزش مدل‌های زبانی و همچنین در بسیاری از کاربردهای دیگر یادگیری ماشین شده‌اند. و توانستند بسیاری از مشکلات مدل‌های قبلی را حل کنند.

#### ۲.۳ مشکلات ترجمه ماشینی و ترانسفورمرها:

در ابتدا، ترجمه ماشینی (MT) یک چالش اساسی در زمینه پردازش زبان طبیعی بود. مدل‌های اولیه‌ای مانند مدل‌های مبتنی بر قواعد (Rule-based models) برای ترجمه استفاده می‌شدند که در آن‌ها، ترجمه‌ها به صورت دستی با استفاده از قواعد زبانی مشخص تنظیم می‌شدند. این روش‌ها هرچند دقیق بودند، اما محدودیت‌های زیادی داشتند و نمی‌توانستند ویژگی‌های پیچیده‌تر زبان را مدل‌سازی کنند. سپس مدل‌های آماری (Statistical Models) معرفی شدند. این مدل‌ها

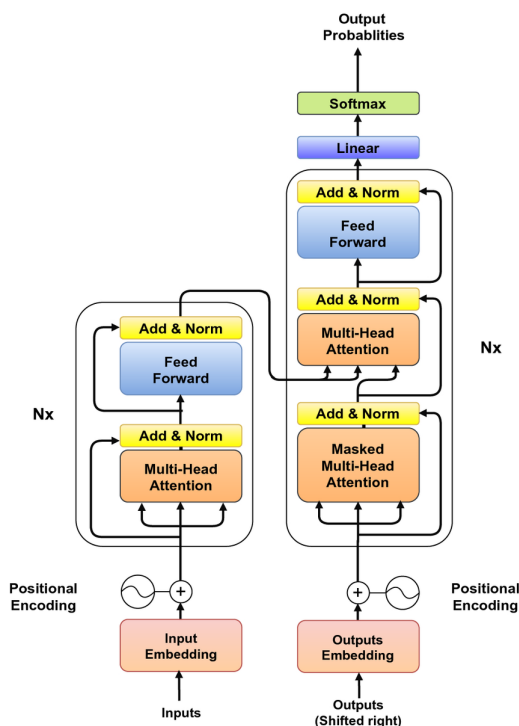
از داده‌های ترجمه‌شده برای آموزش مدل‌های آماری استفاده می‌کردند که احتمال ترجمه‌ای صحیح را براساس شواهد آماری محاسبه می‌کردند. مدل‌هایی مانند مدل‌های ترجمه آماری مبتنی بر جمله (Statistical (Phrase-based Models) از این نوع بودند، که قادر به ترجمه جملات بهتر از مدل‌های مبتنی بر قواعد بودند، اما هنوز هم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

### ۳.۳ ظهور ترانسفورمرها:

در سال ۲۰۱۷ مقاله ... توسط گوگل منتشر شد که مفهوم جدیدی به نام ترانسفورمرها را بیان کرد. این مقاله به موضوع ترجمه ماشینی پرداخت و نشان داده با استفاده از مفهوم مکانیزم توجه (Attention-Mechanism) می‌توان بسیاری از مشکلات مدل‌های قبلی را حل کرد. این مدل‌ها از پردازش موازی استفاده می‌کردند برخلاف مدل‌های قبلی که از پردازش‌های سریالی استفاده می‌کردند. و این مدل‌ها قادر هستند به طور همزمان به تمام بخش‌های ورودی توجه کنند. و این کار باعث شد که ترانسفورمرها در پردازش تصویر و متن بسیار سریع‌تر و دقیق‌تر از مدل‌های قبلی عمل کنند.

### ۴.۳ معماری ترانسفورمرها:

در تصویر ... معماری ترانسفورمر نمایش داده شده است و بخش‌ها و اجزای مختلف آن مشخص شده است که از دو بخش اصلی انکودر و دیکدر تشکیل شده است هدف انکودر این است که داده ورودی را بگیرد و ویژگی‌های آن را استخراج می‌کند و هدف دیکدر این است که ویژگی‌های استخراج شده را به زبان مقصد تبدیل می‌کند. که به طور مختصر به معماری و بخش‌های مختلف آن می‌پردازیم.



شکل ۳.۴.۱: معماری ترانسفورمر ها

### ۱.۴.۳ Embedding:

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

این بردار ها ویژگی های کلمه را فضای عددی نمایش میدهد. روش های مختلفی برای تبدیل متن به بردار وجود دارند. از جمله این روشها میتوان به روش Word2Vec و GloVe اشاره کرد. همانطور که در تصویر ... نشان داده شده است، هر کلمه که به صورت توکن است ابتدا در دیکشنری تعریف شده پیدا می شود و پس از پیدا شدن در دیکشنری با استفاده از روش های embedding، هر کلمه به برداری از اعداد تبدیل می شوند. این embedding ها شباهت های معنایی بین کلمات را مدل سازی میکنند به طوری که کلماتی که از نظر معنایی شبیه به هم هستند، بردار آن ها نیز به



یک دیگر نزدیک تر است. و به این ترتیب کلمات برای مدل ها و شبکه های عصبی قابل فهم می شود.

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

شکل ۳.۴.۲: word embedding

### ۲.۴.۳ positional embedding

ما تا الان هر کلمه را به برداری از اعداد که برای مدل قابل فهم باشد، تبدیل کرده ایم. اما مدل های ترانسفورمر نمیتوانند جایگاه هر کلمه را تشخیص دهند. در مدل های ترانسفورمر بر خلاف مدل های بازگشتی به دلیل این که کلمه های به صورت موازی وارد میشوند نیاز داریم تا جایگاه هر کلمه را بدانیم به طور مثال جمله من تو رو دوست داریم باید به طور دقیق بدانیم کلمه من، کلمه اول جمله است، کلمه تو کلمه دوم جمله است و ... حال ما باید به مدل توالی این کلمات را بفهمیم، پس ما نیاز داریم به مدل یک سری اطلاعات اضافی بدهیم به طوری که مدل توالی کلمات را یاد بگیرد. روش های مختلفی برای اضافه کردن positional embedding به مدل وجود دارد که در ترانسفورمر ها از روش Embedding Positional Sinusoidal استفاده میشود. این روش قابل یادگیری نیست و صرفاً از یک سری فرمول های ساده برای positional embedding استفاده میکند.

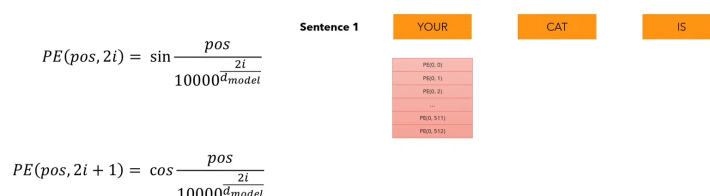
برای موقعیت  $pos$  در توالی و بعد  $i$  در فضای برداری، تعبیه موقعیتی به صورت زیر تعریف

می شود:

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

و برای مقادیر فرد:

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



شکل ۳.۴.۳: embedding word

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

$i$  - شاخص بعد در بردار موقعیتی است. این متغیر به اندیس بعدی که موقعیت کلمه در آن نمایش داده می‌شود اشاره دارد. برای مثال، اگر فضای برداری مدل دارای ابعاد  $d$  باشد،  $i$  از 0 تا  $d - 1$  تغییر می‌کند.

$d$  - ابعاد فضای برداری مدل است. این مقدار مشخص می‌کند که هر کلمه در توالی به چه تعداد ابعاد در فضای برداری نگاشت می‌شود. به عبارت دیگر،  $d$  نشان‌دهنده تعداد ویژگی‌ها (یا ابعاد) در بردار موقعیتی است.

10000 - یک مقدار ثابت است که برای تنظیم مقیاس توابع تناوبی استفاده می‌شود. این مقدار به گونه‌ای تنظیم شده است که از نوسانات زیاد جلوگیری کند و همچنین فرکانس‌های مختلفی را برای ابعاد مختلف به وجود آورد.

همانطور که در شکل زیر مشاهده میکنید بعد از embedding کلمات embed-positional ding به آن اضافه می شود. که در این روش از توابع سینوس و کسینوس استفاده میشود. این توابع موقعیت ها را در فضای برداری به گونه ای نگاشت میکنند که مدل بتواند از ترتیب کلمات در توالی آگاه باشد. این ویژگی به مدل کمک میکند تا مدل توالی زمانی را بتواند درک کند و الگوهای زمانی را شبیه سازی کند. از مزایای این مدل میتوان به عدم نیاز به آموزش و توزیع متوازن جایگاه هر کدام از کلمات اشاره کرد.

YOUR	CAT	IS	A	LOVELY	CAT
992.207	171.411	421.699	775.962	4422.493	171.411
5402.640	3276.350	1364.051	5567.258	4315.080	3276.350
1853.448	9192.819	0.345	56.942	9358.778	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
...	8080.133	...	...	...	7902.890
...	2620.399	...	...	...	912.870
...	...	...	...	...	3821.102
...	9386.405	...	...	...	1659.217
...	2120.159	...	...	...	7018.620

شکل ۳.۴.۴: embedding word

### ۳.۴.۳ attention:

در روش های قدیمی (مانند RNN یا lstm توالی ورودی (مثلاً یک جمله) معمولاً به صورت گام به گام پردازش می شد. 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 می گوئیم. م 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 یک گربه روی زمین نشسته است میخواد بداند کلمه گربه به واژه نشستن بیشتر توجه کند یا به زمین، مثلاً در این جا فعل نشستن ارتباط نزدیک تری به گربه دارد، و از نظر معنایی مرتبط تر است.

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

در Attention Dot-Product Scaled، ابتدا شباهت یا ارتباط بین Query و Key را با محاسبه ضرب داخلی (Dot Product) به دست می آوریم، سپس آن را نرمال می کنیم (با تقسیم بر  $d_k$ ) و از تابع softmax استفاده می کنیم تا ضرایب توجه (Attention Weights) را به دست آوریم. در نهایت با همین ضرایب، ترکیبی خطی از بردارهای Value را می گیریم. فرمول به شکل زیر است:

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

که در آن:

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

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

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

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

تقسیم بر  $d_k$  باعث می شود مقدار ضرب داخلی در ابعاد بالا خیلی بزرگ نشود و شیب ها (Gradients) پایدار بمانند.

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

$\alpha$  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 ۳.۴.۵: Attention

ایدهٔ چندسری: به جای آنکه فقط یک بار  $Q, K, V$  بسازیم و عملیات توجه را انجام دهیم، چندین مجموعهٔ متفاوت  $Q_i, K_i, V_i$  می‌سازیم (هر کدام یک «Head» یا سر نام دارد) و به صورت موازی محاسبات Attention را انجام می‌دهیم. سپس خروجی همهٔ این Headها را کنار هم قرار داده (Concat) و در نهایت با یک ماتریس وزن دیگر ضرب می‌کنیم تا به بعد اصلی بازگردیم. فرمول مربوط به این ایده به شکل زیر است:

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

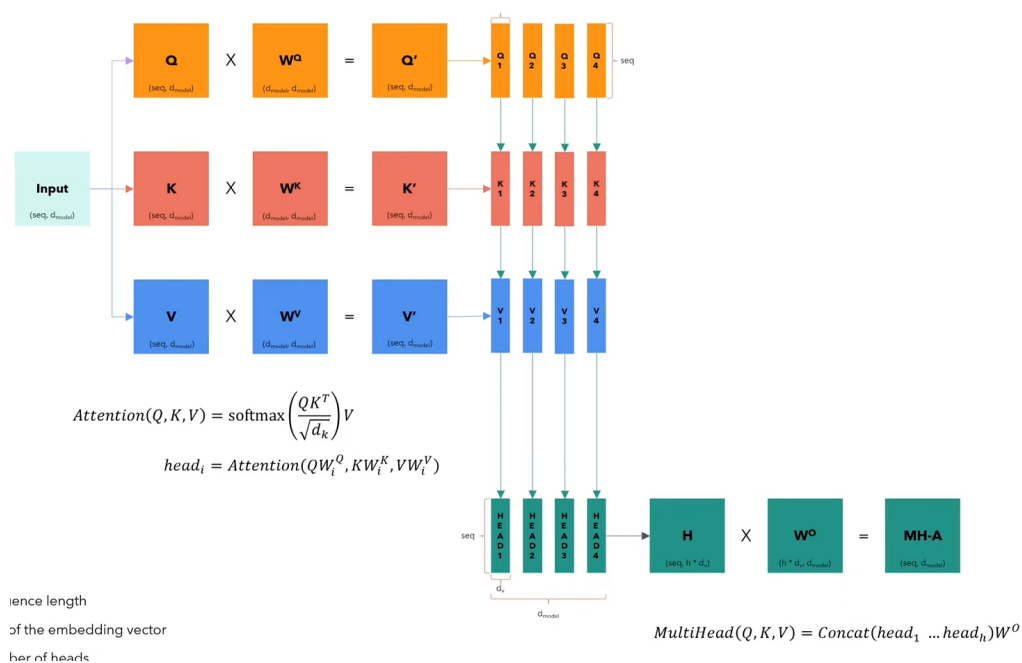
$$\text{MultiHead}(Q, K, V) = [\text{head}_1 \oplus \dots \oplus \text{head}_h] W_O$$

که در آن  $\oplus$  نشان‌دهنده عمل الحاق (Concatenation) 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 ۳.۴.۶: attention head multi

چرا چندین سر؟

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

افزایش ظرفیت مدل: با داشتن چند Head، مدل می‌تواند قدرت بیان بیشتری داشته باشد. ابعاد کمتر در هر Head: در عمل، اگر  $d_{\text{model}}$  مثلاً ۵۱۲ باشد، و تعداد هاها  $h = 8$ ، آنگاه هر Head ابعادی در حدود  $d_k = 64$  خواهد داشت؛ و این محاسبات ضرب داخلی را نیز مقیاس‌پذیر و قابل موازی‌سازی می‌کند.

### ۴.۴.۳ (Add): Connection Residual

در معماری‌های مختلف هنگامی که تعداد لایه‌ها زیاد می‌شود، اغلب دچار ناپایداری گرادیان (Vanishing/Exploding Gradients) می‌شوند. و باعث مشکل در آموزش مدل می‌شود. در مدل ترانسفورمر به جای این که خروجی attention را به صورت مستقیم به لایه بعدی بدهیم، ورودی آن را نیز حفظ کرده و به خروجی اضافه می‌کنیم. اگر  $x$  ورودی به زیرماژول و  $\text{SubLayer}(x)$  خروجی آن زیرماژول باشد، در انتهای کار، ما عبارت زیر را محاسبه می‌کنیم:

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

این جمع به صورت عنصر به عنصر (Element-wise Addition) انجام می‌شود.

### ۵.۴.۳ مزایای Connection Residual در ترانسفورمر

کمک به جریان یافتن گرادیان

وقتی ورودی مستقیماً به خروجی اضافه می‌شود، مسیری مستقیم برای عبور شیب (گرادیان) به عقب ایجاد می‌گردد. در صورت نبود این اتصال، اگر شبکه عمیق شود، گرادیان‌ها ممکن است در لایه‌های پایین محو شوند و عملاً vanishing gradient رخ می‌دهد.

### حفظ اطلاعات اصلی (هویت ورودی)

حتی اگر زیرماژول تغییری در اطلاعات ورودی ایجاد کند، با وجود Connection، Residual ورودی اصلی همواره در خروجی نهایی حضور دارد. این ویژگی باعث می‌شود در صورت ناکافی بودن یادگیری زیرماژول یا در مراحل اولیه آموزش، دست‌کم بخشی از سیگنال/اطلاعات خام به لایه‌های بالاتر برسد.

### کاهش ریسک تخریب ویژگی‌ها

در شبکه‌های عمیق، یکی از مشکلات این است که هر لایه ممکن است بخشی از اطلاعات مفید را تخریب کند. Connection Residual تضمین می‌کند که اگر لایه‌ای به هر دلیل نتوانست الگوی بهینه را یاد بگیرد، اطلاعات قبلی حداقل به صورت دست‌نخورده تا حدی منتقل می‌شود.

## ۵.۳ (Norm): Normalization Layer

در یادگیری عمیق، نرمال‌سازی (Normalization) داده‌های یک لایه یا فعال‌سازی‌ها، اغلب به سرعت بخشیدن به همگرایی و پایدار کردن آموزش کمک شایانی می‌کند. شاید معروف‌ترین نوع نرمال‌سازی، Normalization Batch باشد که پیش‌تر در کارهای بینایی (ها) (CNN) بسیار مورد استفاده قرار گرفت.

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



## تفاوت Norm Layer با Norm Batch

### Normalization: Batch

در Norm، Batch برای نرمال‌سازی، میانگین و واریانس روی تمام نمونه‌های موجود در مینی‌بچ (و نیز در طول ابعاد ویژگی) محاسبه می‌شود. این موضوع در NLP کمی دردسرساز است؛ چون ترتیب (Order) توکن‌ها، طول جمله‌ها و حتی اندازه مینی‌بچ ممکن است نامنظم باشد. همچنین به خاطر تنوع طول توالی‌ها (Sequence)، (Length) پیاده‌سازی Norm Batch می‌تواند پیچیده شود.

### Normalization: Layer

در Norm، Layer برای هر توکن به صورت جداگانه (در طول بُعد ویژگی)، میانگین و واریانس گرفته می‌شود. فرض کنید در یک لایه، بردار  $h_i \in \mathbb{R}^{d_{\text{model}}}$  مربوط به توکن  $i$  باشد؛ یعنی ابعاد ویژگی آن  $d_{\text{model}}$  است. ما میانگین  $\mu_i$  و واریانس  $\sigma_i^2$  را از اجزای این بردار محاسبه می‌کنیم:

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

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

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

در نهایت، برای این که مدل بتواند مقیاس و بایاس جدیدی یاد بگیرد، شبیه Norm، Batch دو

پارامتر  $\gamma$  (Scale) و  $\beta$  (Bias) نیز در طول بُعد ویژگی اعمال می‌شوند:

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

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

## مزایای Normalization Layer در ترانسفورمر

- بی‌نیازی از وابستگی به ابعاد مینی‌بچ: با Norm، Layer می‌توان حتی با اندازه مینی‌بچ برابر ۱ نیز به‌خوبی آموزش دید، چراکه آمارها وابسته به ابعاد ویژگی‌اند و نه مینی‌بچ.
  - پایدارسازی توزیع فعال‌سازی‌ها: زمانی که مدل در حال یادگیری است، توزیع‌های داخلی لایه‌های میانی ممکن است تغییر کند (پدیده Covariate Internal Shift). Norm Layer Shift). با نرمال‌سازی این توزیع، آموزش را پایدارتر و سریع‌تر می‌کند.
  - سازگاری با داده‌های توالی‌محور: هر توکن را جداگانه نرمال می‌کند و نگرانی‌ای بابت ترتیب طول جمله‌ها، یا قرار گرفتن چند جمله کوتاه/بلند در یک مینی‌بچ نداریم.
- در معماری ترانسفورمر، پس از خروجی هر زیرماژول (مثل Attention یا MLP)، مراحل به‌شکل زیر است:
- Connection: Residual ابتدا ورودی همان زیرماژول (مثلاً بردار  $x$ ) را با خروجی زیرماژول  $(\text{SubLayer}(x))$  جمع می‌کنیم. حاصل این جمع را می‌توان چنین نوشت:

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

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

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

## ۶.۳ decoder:

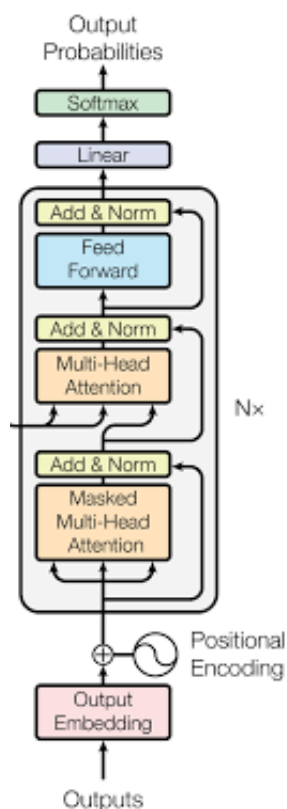
دیکودر در معماری ترانسفورمرها وظیفه تولید خروجی نهایی را بر عهده دارد. این خروجی معمولاً می‌تواند توالی هدف (Target) (Sequence) باشد، مثل ترجمه یک جمله یا پیش‌بینی توکن‌های بعدی در یک توالی. در این بخش دیکدر دو ورودی اصلی دارد: توالی هدف که معمولاً به صورت خودکار تولید می‌شود (مثلاً در ترجمه ماشینی یا تولید متن)، و نمایش (Representation) گذشته که توسط انکودر (Encoder) تولید شده است و شامل ویژگی‌های استخراج‌شده از توالی ورودی می‌باشد. دیکودر از این ورودی‌ها استفاده می‌کند تا به صورت گام‌به‌گام، خروجی نهایی خود را تولید کند.

همانطور که در تصویر مشاهده می‌کنید دیکدر دو ورودی دارد.

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

## ۷.۳ attention head multi masked

در ترانسفورمر، مکانیزم Attention Multi-Head در بخش دیکودر به صورت Masked پیاده‌سازی می‌شود تا مدل نتواند توکن‌های آینده را ببیند و به صورت خودبازگشتی (Autoregressive) توکن بعدی را پیش‌بینی کند. در واقع ایده اصلی استفاده از mask جلوگیری از مشاهده آینده است. در معماری‌های خودبازگشتی (Autoregressive)، مدل در گام  $i$  از دیکودر تنها باید به توکن‌های قبلی  $\{y_1, \dots, y_{i-1}\}$  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).



شکل ۳.۶.۷:

Decoder

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

### ۸.۳ مثال عددی mask attention:

فرض کنید دنباله ۴ توکنی داریم:

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

خروجی Dot-Product Scaled (قبل از softmax) یک ماتریس  $4 \times 4$  خواهد بود:

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

برای سطر ۱ (توکن اول): می‌تواند خودش (ستون ۱) را ببیند، اما ستون‌های ۲ تا ۴ را ماسک می‌کنیم. برای سطر ۲ (توکن دوم): می‌تواند به ستون‌های ۱ و ۲ نگاه کند، اما ۳ و ۴ ماسک می‌شوند. برای سطر ۳: ستون‌های ۱، ۲ و ۳ را ببیند، ستون ۴ ممنوع است. برای سطر ۴: ستون ۱، ۲، ۳، ۴ آزاد است. (چون چهارمین توکن می‌تواند توکن‌های قبلی را ببیند، و از طرفی این توکن «خودش» نیز موردی ندارد - بسته به پیاده‌سازی ممکن است تصمیم بگیریم توکن فعلی از خودش نیز استفاده کند یا نه؛ در معماری استاندارد، سطر  $i$  معمولاً به ستون  $i$  هم دسترسی دارد).

در عمل، ماتریس ماسک  $M$  به شکل زیر خواهد بود (اگر به شکل پایین‌مثلثی نشانه‌گذاری کنیم):

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

به این ترتیب، پس از جمع شدن با  $S$  و اجرای softmax در هر سطر، ضرایب توجه‌ی ستون‌های ماسک‌شده به صفر میل می‌کنند.

### ۹.۳ transformer: vision

ایده ترانسفورمرها در تصویر از تعمیم دادن ترانسفورمر متن به وجود آمده است.

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

در روش‌های متداول برای پردازش تصویر از convolution ها استفاده می‌کردند. اما در

ترانسفورمرها عکس‌ها به پیچ‌های مختلف شکسته می‌شوند. و این قسمت‌های شکسته شده

عکس به یک دیگر توجه میکنند که چقدر به یک دیگر شباهت دارند. در قسمت های بعد به طور مفصل به این کار ها میپردازیم.

### ۱.۹.۳ transformer: vision in embedding patch

در ترانسفورمر های مبتنی بر متن هر کلمه به توکن تبدیل می شود. و هر کدام از این کلمات به بردار هایی تبدیل میشود. و این بردار ها بعد از اضافه شده embedding positional وارد Attention در ترانسفورمر ها میرسید.

حال همین ایده در تصویر پیاده سازی شده است. همانطور که در تصویر ... مشاهده می کنید. در Transformer، Vision به جای عملیات کانولوشن، مستقیماً تصویر را به بلاک های غیرهم پوشان  $(P \times P)$  قطعه بندی می کنیم تا موازی سازی بهتری داشته باشیم و به مدل اجازه دهیم از سازوکار Self-Attention (توجه سراسری) برای ارتباط بین این بلاک ها استفاده کند.



شکل ۳.۹.۸: patch to image

### ۲.۹.۳ شکل پیچ ها:

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

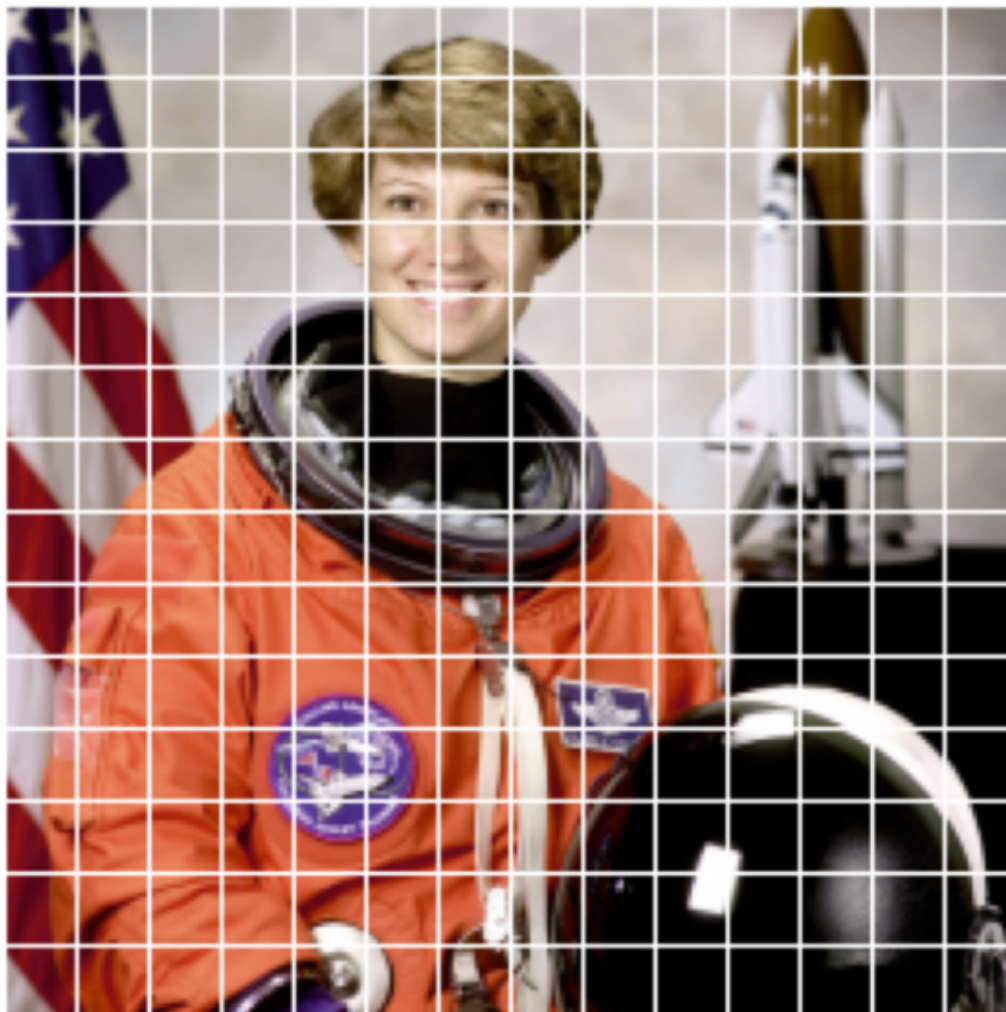
$$H = 224, \quad W = 224, \quad C = 3$$



شکل ۳.۹.۹: Image original

حال اگر اندازه هر پچ  $(P \times P)$  باشد (مثلاً  $16 \times 16$ )، 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 (۰ تا ۱۵ در ارتفاع) و (۰ تا ۱۵ در عرض)، پچ دوم: مختصات (۰ تا ۱۵ در ارتفاع) و (۱۶ تا ۳۱ در عرض)، و به همین ترتیب تا کل تصویر پوشش داده شود.



شکل ۳.۹.۱۰: Image patch

۳.۹.۳ تعداد پچ ها:

اگر پچ های ما بدون هم پوشانی باشند، ابعاد پچ دقیقاً باید بر ابعاد تصویر بخش پذیر باشد.

تعداد پچ ها افقی:

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

تعداد پچ ها عمودی:

$$\frac{H}{P}$$



در مجموع:

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

برای مثال اگر:

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

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

تر اکثر نسخ‌های مبدل‌های بینایی، چ‌ها بدون هم پوشانی هستند. (Non-overlapping) اندازه پچ‌های کوچک باعث می‌شود، تعداد پچ‌ها زیاد شوند. و با زیاد شدن تعداد پچ‌ها، هزینه attention زیاد می‌شود. و هم چنین اندازه پچ‌های بزرگ باعث می‌شود تعداد پچ‌ها کمتر شود، و هزینه‌های attention کمتر شود. اما در پچ‌های بزرگ باعث می‌شود که جزییات محلی (local) از بین برود.

### ۴.۹.۳ بردار کردن هر پچ

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

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

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

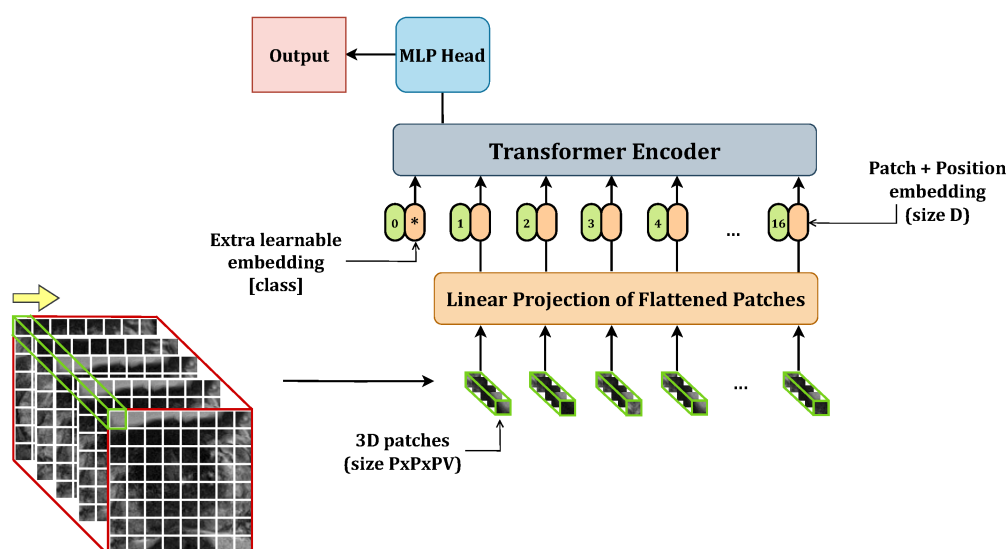
### ۱۰.۳ اعمال لایه خطی: (Projection)

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

(Transformation) را انجام می‌دهد تا همهٔ پچ‌ها یک نمایندگی (Embedding) با ابعاد یکنواخت

$d_{\text{model}}$  پیدا کنند:

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



شکل ۳.۱۰.۱۱: Transformer Vision in Embedding

این مرحله شبیه ساخت توکن در NLP است؛ با این تفاوت که در NLP توکن «کلمه» یا «زیرکلمه» است و پیش‌تر دارای بردار تعبیه‌شده Embedding شده بوده است. در trans-vision former ما ابتدا باید تصاویر را پچ کنیم و سپس بردارهای embedding شده را از این پچ‌ها به دست آوریم.

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

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

هر پچ اکنون یک بردار  $d_{\text{model}}$ -بعدی است. پس یک مجموعه با طول  $N$  (تعداد پچ‌ها) و عرض  $d_{\text{model}}$  خواهیم داشت. اگر عدد پچ‌ها  $N$  باشد (مثلاً ۱۹۶)، ترانسفورمر می‌تواند خودش با استفاده از مکانیزم Self-Attention، وابستگی (Relations) میان پچ‌ها را یاد بگیرد: کدام بخش از

تصویر برای کدام بخش دیگر مهم‌تر است، چگونه ترکیب جهانی (Global Context) ساخته شود، و غیره.

معمولاً پچ‌ها را به صورت ردیفی (Row by Row) شماره گذاری می‌کنند (ابتدا پچ‌های ردیف بالایی از چپ به راست، سپس ردیف بعدی و ...)، تا مدل در صورت نیاز بتواند از پوزیشن‌ها اطلاعات مکانی تقریبی داشته باشد. در عمل، چون قصد داریم (در مراحل بعد) به هر پچ یک Embedding Positional هم اضافه کنیم، مکان دقیق هر پچ در بعد دوم (ویژگی) رمز می‌شود. در ویژن ترانسفورمر دیگر به کانولوشن وابسته نیست. در عوض از embedding استفاده می‌شود.

Split کردن تصویر به بلاک‌های  $(P \times P)$ ، Flatten و Linear Projection، عملیات ریاضی ساده‌ای هستند و به راحتی قابل موازی‌سازی روی GPU/TPU هستند.

### ۱.۱۰.۳ Token: CLS

Token CLS یک بردار ویژه است که به ابتدای دنباله ورودی اضافه می‌شود و نقش آن این است که اطلاعات کلی و مرتبط با دنباله ورودی (چه متن، چه تصویر) را در خود خلاصه کند. token cls در به ابتدای پچ‌های تصویری قرار می‌گیرد.

این توکن یک بردار با ابعاد  $d_{\text{model}}$  است (همان ابعاد سایر توکن‌ها).

برداری CLS یک پارامتر یادگرفته‌نی است، یعنی مدل در طول آموزش مقادیر آن را برای ذخیره و جمع‌آوری اطلاعات بهینه می‌کند.

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

Token CLS از طریق مکانیزم Self-Attention در ترانسفورمر می‌تواند با همه توکن‌های دیگر (یعنی پچ‌های تصویر) ارتباط برقرار کند و اطلاعات مهم آن‌ها را در طول لایه‌های ترانسفورمر

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

Token CLS از طریق ضرب داخلی در مکانیزم Attention می تواند به تمام پچ ها نگاه کند. ضرایب توجه ( $\alpha$ ) تعیین می کند که Token CLS چه مقدار اطلاعات از هر پچ دریافت کند. و چقدر در پروسه مدل دخیل باشد.

Token CLS به طور ضمنی یاد می گیرد که روی ویژگی هایی که برای دسته بندی مهم هستند (مانند الگوها، اشکال و بخش های کلیدی تصویر) تمرکز کند.

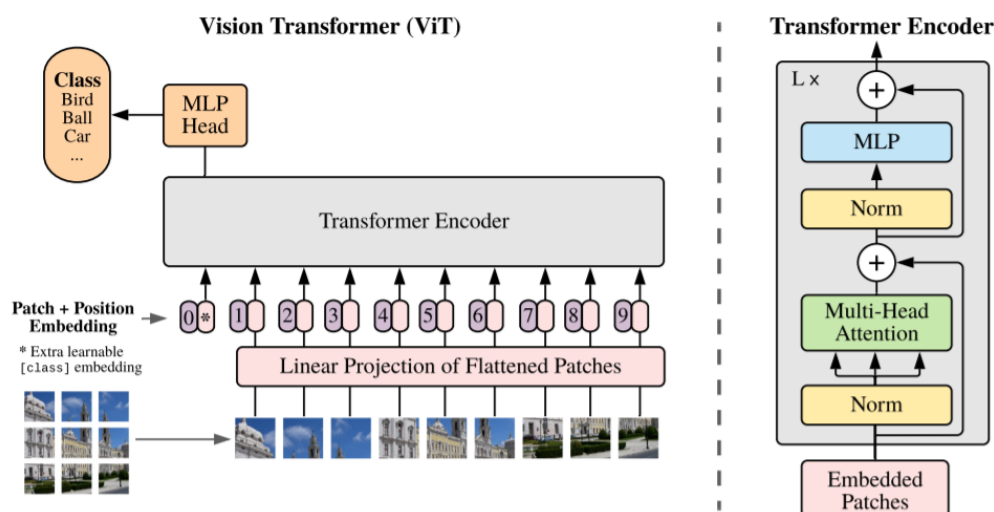
در طول لایه های ترانسفورمر، Token CLS نقش محوری در تنظیم بازنمایی کل تصویر ایفا می کند. به عبارت دیگر، این توکن به نوعی مرکز پردازش کل اطلاعات تصویر است.

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

### ۲.۱۰.۳ encoder: transformer vision

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

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



شکل ۱۱.۱۰.۳: Transformer Vision in Token Cls

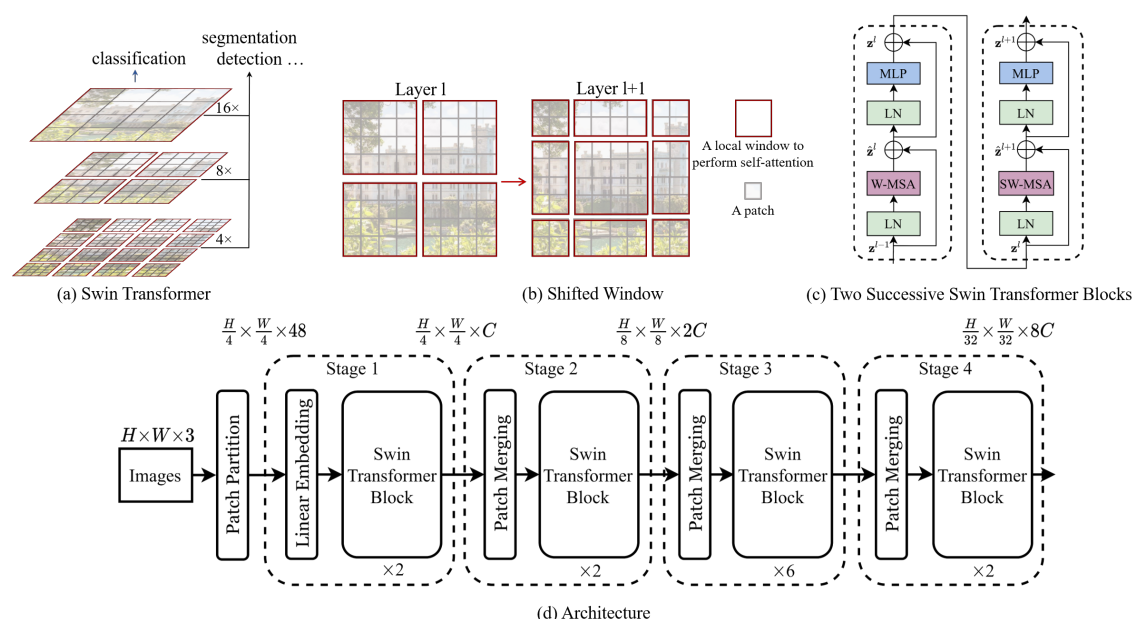
از خود نشان میدهد.

### ۱۱.۳ Transformer: Swin

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

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

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



شکل ۳.۱۱.۱۳: Transformer Swin

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

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

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

## ۱.۱۱.۳ Partition): (Patch

فرض میکنیم تصویر ورودی  $I$  دارای ابعاد  $(H \times W \times 3)$  باشد. گام نخست، تقسیم تصویر به پچ‌های کوچک  $(P \times P)$  است. اگر  $P$  اندازه پچ (Patch) (size) باشد، آنگاه تعداد پچ‌ها در بعد افقی و عمودی، به ترتیب  $\frac{W}{P}$  و  $\frac{H}{P}$  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:

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

## ۲.۱۱.۳ Embedding: Linear

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

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

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

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

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

● پنجره‌بندی تصویر (Partition Window) یا پنجره‌بندی جابه‌جاشده (Win-Shifted Partition dow)

● اعمال WMSA (Attention Self Multi-Head Window)

● لایه Norm Layer و Connection Skip

● مسیر MLP:

— یک لایه MLP شامل دو لایه Fully-Connected و تابع فعال‌ساز GeLU (یا تابع مشابه)

— لایه Norm Layer و Connection Skip

### ۳.۱۱.۳ Self-Attention: Multi-Head Window

تعریف پنجره‌های محلی:

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

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

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

به بیان ساده‌تر:

● نقشه ویژگی 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 Attention را انجام می‌دهیم.
- این کار باعث می‌شود تعداد پیکسل‌هایی که درگیر محاسبه Attention هستند، به مراتب کمتر شود و هزینه محاسباتی کاهش یابد.

### ۴.۱۱.۳ Attention:

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

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

که

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

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

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

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

### ۵.۱۱.۳ Windows: shifted

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

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

• در یک لایه، محاسبات Attention در پنجره‌های محلی ثابت انجام می‌شود.

• در لایه بعدی، پنجره‌ها به اندازه‌ای مشخص جابه‌جا می‌شوند (به‌صورت شیفت افقی و

عمودی) تا نواحی مرزی نیز در محاسبات گنجانده شوند.

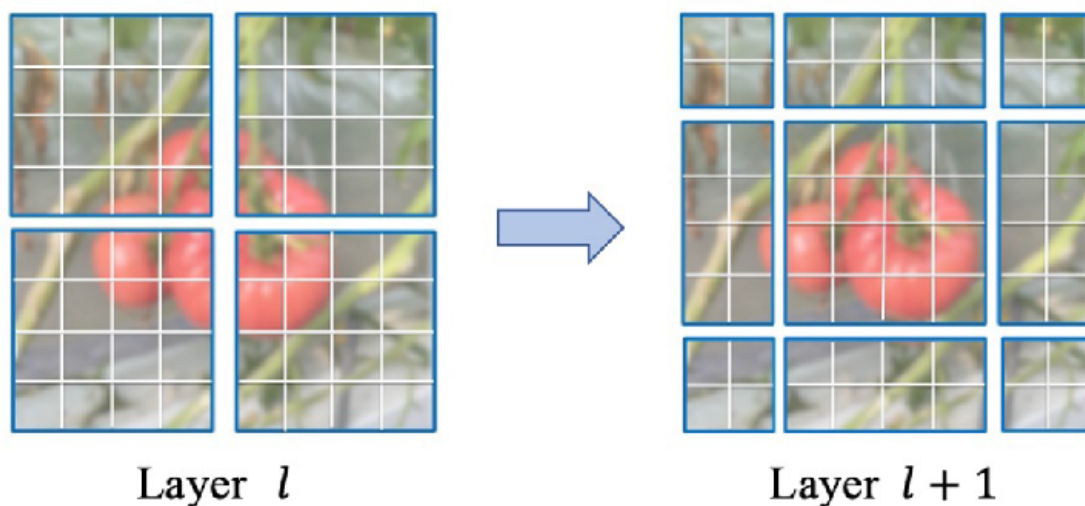
- این فرآیند باعث می‌شود که پیکسل‌ها در پنجره‌های مختلف (و در مرزهای مختلف) در محاسبات دخیل شوند و تبادل اطلاعات بهتری میان نواحی تصویر رخ دهد.

بلوک اول: (W-MSA)

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

بلوک دوم: (SW-MSA)

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



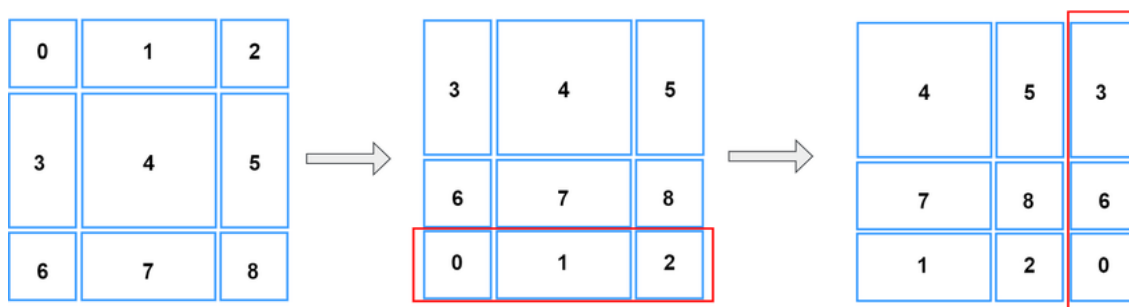
شکل ۳.۱۱.۱۴: Window Shifted vs Window

در راستای افقی و عمودی است. به این ترتیب:

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

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

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



شکل ۳.۱۱.۱۵: Shift cycle

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

اگر بخواهم با مثال توضیح دهم فرض کنید در یک تابلوی شطرنجی، خانه‌های کناری همدیگر را «نمی‌بینند» چون در دو بلوک مختلف هستند. اما اگر کمی تابلوی شطرنجی را به سمت بالا-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 Transformer Swin (W-MSA و SW-MSA) تکمیل‌کننده یکدیگر می‌شوند:

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

Mlp: ۶.۱۱.۳

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

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

که:

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

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

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

## ۷.۱۱.۳ merging: patch

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

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

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

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

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

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

۱. انتخاب بلوک‌های  $(2 \times 2)$ 

ابتدا نقشه ویژگی را به بلوک‌های  $(2 \times 2)$  تقسیم می‌کنیم (در بُعد مکانی). اگر  $Z_{i,j}$  ویژگی 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 (Concat) ویژگی‌های چهار پیکسل

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

## ۳. لایه خطی برای تغییر بُعد

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

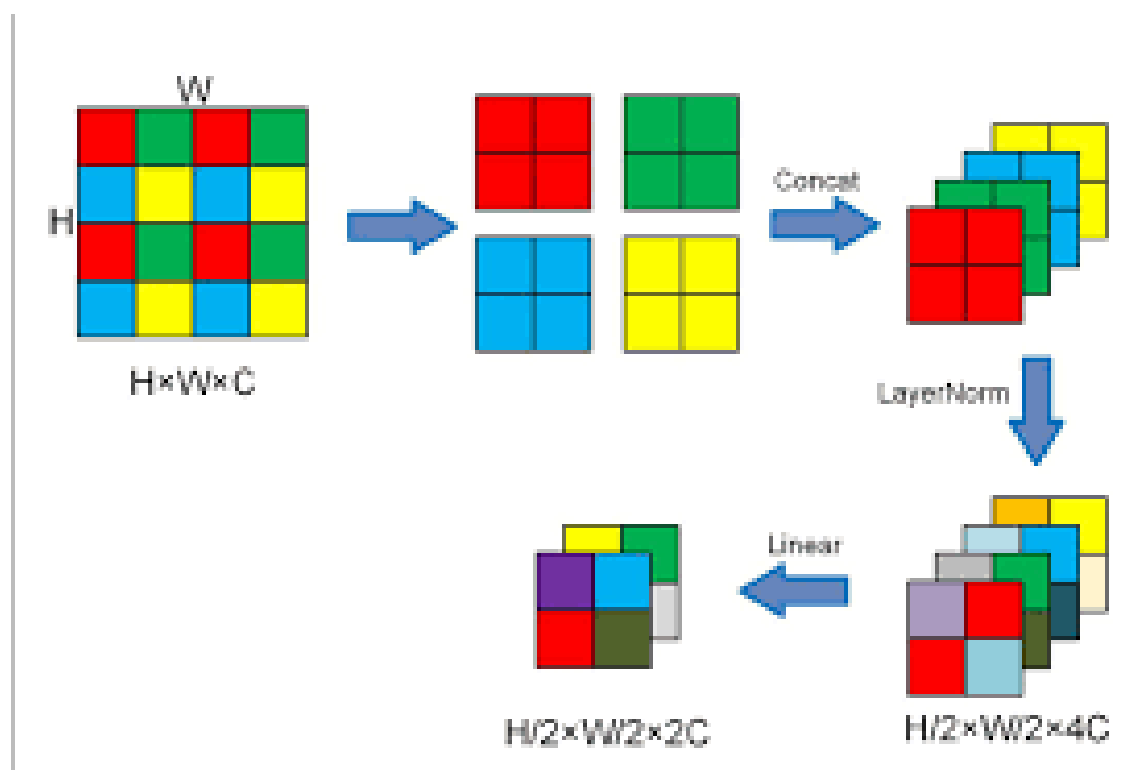
$$Z' \mapsto Z'' = Z'W_{\text{merge}} + b_{\text{merge}},$$

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

## ۴. کاهش ابعاد مکانی

در عین حال، وقتی هر چهار پیکسل  $(2 \times 2)$  را در یک ادغام می‌کنیم، یعنی نقشه ویژگی فضای  $(\frac{H}{2P} \times \frac{W}{2P})$  پیدا می‌کند (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، ما مرتباً با لایه‌های Pooling یا Convolution با استراید ۲ روبه‌رو می‌شویم تا ابعاد فضایی را پایین بیاوریم و در عوض تعداد کانال‌ها را بالا ببریم. این کار کمک می‌کند



شکل ۱۶. ۱۱. ۳: Merging Patch

اطلاعات سطح بالاتر (مانند ساختار کلی اشیا) راحت تر استخراج شود. Transformer Swin هم از همین ایده سلسله مراتب الهام گرفته است. همچنین اگر ابعاد فضایی را پیوسته پایین نیاوریم، هزینه Attention به شدت زیاد می شود (چون در هر لایه باید محاسبه Attention برای همه پیکسل ها انجام شود).

در معماری کلی Transformer Swin، در Stage ۱ ویژگی های اصلی گرفته می شود و بعد از گذر از W-MSA و SW-MSA، عملیات Merging Patch انجام می شود. حالا در Stage ۲ ویژگی های کوچکتری داریم، اما تعداد کانال ها افزایش یافته است. مطابق آنچه در کانولوشن داریم، با افزایش عمق، تعداد ویژگی ها کمتر و تعداد کانال ها افزایش پیدا می کند.

و در نهایت پس از stage آخر به یک لایه FC برای متصل می شود. و نتیجه کلاس ها پس از



عبور از Softmax به ما داده میشود.

## فصل ۴

### روش های پیشنهادی

در این پژوهش ما دو روش را بهبود داده ایم :

## فصل ۵

### آزمایشات و نتایج

## کتاب نامه

- [1] 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. [18](#)
- [2] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, New York, 2006. [6](#), [7](#), [10](#), [11](#)
- [3] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995. [9](#)
- [4] Thomas M. Cover and Peter E. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1):21–27, 1967. [8](#)
- [5] Daniel Crevier. *AI: The Tumultuous History of the Search for Artificial Intelligence*. Basic Books, New York, 1993. [4](#), [5](#)
- [6] Pedro Domingos and Michael Pazzani. On the optimality of the simple bayesian classifier under zero-one loss. *Machine Learning*, 29(2–3):103–130, 1997. [10](#)
- [7] Richard O. Duda and Peter E. Hart. *Pattern Classification and Scene Analysis*. John Wiley & Sons, New York, 1973. [8](#), [9](#)
- [8] Jeffrey L. Elman. Finding structure in time. *Cognitive Science*, 14(2):179–211, 1990. [12](#)
- [9] Edward A. Feigenbaum and Pamela McCorduck. *The Fifth Generation: Artificial Intelligence and Japan's Computer Challenge to the World*. Addison-Wesley, Reading, MA, 1983. [4](#), [5](#)

- [10] Felix A. Gers, Jürgen Schmidhuber, and Fred Cummins. Learning to forget: Continual prediction with lstm. In *Proceedings of the Ninth International Conference on Artificial Neural Networks (ICANN-99)*, pages 850–855, Edinburgh, UK, 1999. [11](#), [12](#), [14](#), [15](#), [16](#)
- [11] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, Cambridge, MA, 2016. [6](#), [11](#), [12](#), [13](#), [14](#), [16](#), [17](#), [18](#)
- [12] 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. [12](#), [13](#), [16](#), [18](#)
- [13] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997. [11](#), [12](#), [13](#), [14](#), [15](#), [16](#), [17](#)
- [14] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An Introduction to Statistical Learning: with Applications in R*. Springer, New York, 2013. [7](#), [8](#)
- [15] James Lighthill. *Artificial Intelligence: A General Survey*. HM Stationery Office, London, 1973. Science Research Council Report. [4](#)
- [16] Andrew McCallum and Kamal Nigam. A comparison of event models for naive bayes text classification. In *AAAI-98 Workshop on Learning for Text Categorization*, pages 41–48, Madison, WI, 1998. [10](#)
- [17] 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](#)
- [18] 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](#)
- [19] Tom M. Mitchell. *Machine Learning*. McGraw-Hill, New York, 1997. [6](#), [8](#), [9](#), [10](#)
- [20] Douglas C. Montgomery, Elizabeth A. Peck, and Geoffrey G. Vining. *Introduction to Linear Regression Analysis*. John Wiley & Sons, Hoboken, NJ, 6th edition, 2021. [7](#)

- [21] Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective*. MIT Press, Cambridge, MA, 2012. [6](#), [7](#), [9](#), [10](#)
- [22] Allen Newell, J. Clifford Shaw, and Herbert A. Simon. Report on a general problem-solving program. In *Proceedings of the International Conference on Information Processing*, pages 256–264, 1959. [4](#)
- [23] Nils J. Nilsson. *The Quest for Artificial Intelligence: A History of Ideas and Achievements*. Cambridge University Press, Cambridge, 2010. [4](#), [6](#)
- [24] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986. [11](#), [13](#), [17](#)
- [25] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson, London, 3rd edition, 2016. [5](#), [6](#)
- [26] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 2nd edition, 2018. [8](#)
- [27] Vladimir Vapnik. *Statistical Learning Theory*. Wiley, New York, 1998. [9](#)
- [28] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008, 2017. [17](#), [19](#)

## پیوست آ

جزئیات مدل‌ها و جدول پارامترها

## Abstract

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

**Key Words:** Clustering , DBSCAN , Voronoi diagrams , Delaunay triangulation , Outlier detection .





# **Enhancing MDBSCAN and MOGA-DBSCAN**

A Thesis Presented for the Degree of  
Master in Computer Science

Faculty of Mathematical Sciences

**Tarbiat Modares University**

by

**Seyed Mohammad Badzohreh**

Supervisor

**Dr. Mansoor Rezghi**

2024