پردیس علوم
دانشکده ریاضی، آمار و علوم کامپیوتر

بازشناسی فعالیت‌های انسانی در ویدیو با استفاده از یادگیری عمیق

نگارنده

کسری کاکایی

استاد راهنما: باقر باباعلی

پایاننامه برای دریافت درجه کارشناسی
در رشته علوم کامپیوتر

تابستان ۱۴۰۲

چکیده

امروزه مبدل‌ها موفقیت‌های چشم گیری در زمینه‌های مختلف، مخصوصا پردازش زبان طبیعی کسب کرده اند. تا کنون نیز مدل‌های زیادی بر پایهٔ توجه برای حل مسئلهٔ تشخیص عمل انسان ارائه شده است ولی همهٔ آن‌ها ترکیب شده با ایده‌های پیشین هستند. ما در این مقاله به بررسی روشی کاملاً بر پایهٔ خود توجه و مدل مبدل‌ها میپردازیم که تشخیص اعمال کوتاه مدل انسان را در زمانی کوتاه و دقتی بالاتر از مدل‌های پیشنهادی گذشته انجام میدهد. از مجموعه دادهٔ MPOSE۲۰۲۱ برای آموزش و تست مدلمان استفاده میکنیم و از بسته‌های OpenPose [۳] و Posenet [۱۷] برای استخراج حالت انسان‌ها در تصاویر استفاده میکنیم، که سپس این اطلاعات را به مدلمان میدهیم تا پردازش کند. در انتها مدل خود را با مدل‌های به روز و پایه در این زمینه مشخص میکنیم و نشان میدهیم که مدلمان برتر است هم از نظر زمانی و هم دفت و برای کاربرد در زمینه‌های آنی و در لحظه مناسب است.

پیشگفتار

تشخیص اعمال انسان یک مسئله در زمینهٔ بینایی ماشین است که از ابتدا یکی از مهم ترین کاربردهای بینایی ماشین بوده است و تلاش زیادی برای آن صورت گرفته است. توانایی تشخیص انسان‌ها در شرایط و صحنه‌های مختلف و پیش بینی اعمال و رفتارشان در عرصه‌های بسیاری کاربرد دارد، مانند روباتیک، امنیت و مراقبت، خودروهای خودران و زیرنویس نویسی خودکار ویدیوها. بیشتر مجموعه داده‌های مورد استفاده در این زمینه بر روی ویدیوهای بلند مدت تمرکز دارد، در نتیجه همیشه فرایند پردازش این مسئله بعد از انجام اعمال انسانی اتفاق افتاده است و سعی شده که اعمال پیچیده و طولانی را تشخیص بدهد با استفاده از تمامی اطلاعات موجود در کل زمان عمل. در نقطهٔ مقابل، ما در این مقاله سعی میکنیم راهی پیشنهاد دهیم که اعمال کوتاه انسان را تشخیص دهد. هدف از این کار این است که بتوانیم سریع و در لحظه اعمال انسان را تشخیص بدهیم. برای مثال، در کاربردهای روباتیک وقتی یک روبات میخواهد با یک انسان ارتباط برقرار کند نیاز دارد به سرعت و در لحظه بداند انسان چه عملی انجام میدهد تا بتواند به موقع پاسخ مناسب نشان بدهد. مخصوصاً زمانی که روبات‌ها برای نگه داری از بچه‌ها یا کهن سالان طراحی شده است نیاز است که بتوانند افتادن آنها یا راه رفتنشان را به سرعت تشخیص بدهد و پاسخ مناسب نشان بدهد.

در این مقاله ما یک مدل جدید برای این مسئله ارائه میدهیم به نام مبدل عمل[1] (AcT)، که در شکل ۱.۳ شمایل کلی آن را میتوانید ببینید. این مدل با الهام گیری از مدل اولیه و سادهٔ Vision Transformer[۷] ساخته شده است. ساختار مبدل[2][۳۱] سال‌هاست که یکی از بهترین و کاربردی ترین ساختارها در پردازش زبان طبیعی[3] است. همچنین ساختار خود توجه چند سره[4] کاربردهای فراوانی به جز NLP از خود نشان داده است مانند دسته بندی عکس‌ها، بالا بردن

[1] Action Transformer
[2] Transformer
[3] Natural Language Processing
[4] Multi-Head Self-Attention

کیفیت عکس و تشخیص صدا. در ادامه مبدل‌ها در کارهای لحظه ای و سیستم‌های جاسازی شده نیز استفاده‌های فراوانی داشتند که ثابت میکند میتواند در این مسئلهٔ ما کاربرد داشته باشد. به تازگی مدل‌های زیادی برای تشخیص اعمال انسان پیشنهاد شده است که با به کار گرفتن مبدل‌ها در کنار ساختارهای قدیمی مانند RNN و CNNها به نتایج خوبی دست یافته اند ولی هیچ مدلی که تنها با مبدل کار کند پیشنهاد داده نشده است.

در AcT مدلی که پیشنهاد میدهیم به طور کامل از ساختار مبدل‌ها الهام گرفته شده است و نتایجی که به دست آوردیم هم سریع و هم کم خطا است. ما تعداد پارامترهای مدل را تغییر میدهیم و آن را در مقیاس‌های مختلف تست میکنیم تا بهترین حالت را پیدا کنیم. برای ارزیابی مدلمان از مجموعه دادهٔ MPOSE۲۰۲۱ استفاده میکنیم. MPOSE2021 یک مجموعه داده از ویدیوهای اعمال انسان است که در کوتاه مدت هستند (زیر یک ثانیه) و تنها یک شخص در آن حضور دارد. همچنین از بسته‌های OpenPose [۳] و PoseNet [۱۷] استفاده میکنیم که قبلاً توسط دیگران نوشته شده است و کارشان استخراج اطلاعات موقعیت و حالت انسان در تصاویر و ویدیوها (مختصات نقاط کلیدی و بردار سرعتشان) است. در نهایت نیز مدلمان را با تعدادی مدل پایه و تعدادی مدل به روز و پیشرفته مقایسه میکنیم و کاربرد و برتری مدل خود را نشان میدهیم.

فهرست مطالب

فصل ۱

مفاهیم مقدماتی

در این بخش توضیحات مختصری در مورد مفاهیم پایه ای در علم بینایی ماشین و یادگیری عمیق میدهیم.

۱.۱ یادگیری ماشین

یادگیری ماشین[۲] شاخه ای از هوش مصنوعی است که داده را به کار میبرد تا عملکرد کامپیوتر را بهبود ببخشد به وسیله دادن توانایی یادگیری به ماشین‌ها.

الگوریتم‌های یادگیری ماشین مدلی بر حسب دادهٔ نمونه، به نام دادهٔ تمرین، میسازد تا پیش بینی‌هایی انجام دهد یا تصمیماتی بگیرد بدون اینکه به صراحت برای اینکار برنامه ریزی شده باشد. الگوریتم‌های یادگیری ماشین در زمینه‌های مختلفی استفاده‌های فراوانی دارند، مانند دارو، فیلتر کردن ایمیل، تشخیص صدا و بینایی ماشین، که در این زمینه‌ها سخت یا غیرممکن است که مسئله را از طریق الگوریتم‌های مرسوم حل کنیم.

چندین نوع مختلف الگوریتم یادگیری ماشین وجود دارد که شامل یادگیری با نظارت، یادگیری بدون نظارت ، یادگیری نیمه نظارتی و یادگیری تقویتی میشود.

[2]Machine Learning

۱.۱.۱ یادگیری نظارتی

الگوریتم‌های یادگیری نظارتی[1] یک مدل ریاضیاتی از روی مجموعه داده‌هایی میسازند که شامل ورودی و هم خروجی مورد نظر هستند. به این داده، دادهٔ تمرینی میگویند و شامل مجموعه ای از مثال‌های تمرینی است. توسط مرحله به مرحله بهینه کردن یک تابع هزینه [2]، الگوریتم‌های یادگیری نظارتی میتوانند تابعی را یاد بگیرند که با آن میتوان خروجی را برای ورودی‌های جدید پیش بینی کرد. یک تابع بهینه خروجی را کاملا درست برای یک ورودی جدید تشخیص میدهد.

۲.۱.۱ یادگیری بدون نظارت

الگوریتم‌های یادگیری بدون نظارت[3] یک مجموعه داده را دریافت میکند که فقط شامل ورودی است و در این داده ساختار پیدا میکند، مانند گروه بندی یا خوشه بندی نقطه‌های داده. در نتیجه این دسته الگوریتم‌ها از داده‌هایی آموزش می یابند که برچسب گذاری یا دسته بندی نشده اند. به جای پاسخ به بازخورد، یادگیری بدون نظارت شباهت‌ها و مشترکات را در داده پیدا میکند و با توجه به این مشترکات نسبت به دادهٔ جدید بازخورد نشان میدهد.

۳.۱.۱ یادگیری نیمه نظارتی

یادگیری نیمه نظارتی[4] بین یادگیری نظارتی و یادگیری بدون نظارت قرار دارد؛ بعضی از داده‌های آموزشی برچسب آموزشی ندارند. بسیاری از محققان در زمینه یادگیری ماشین به این مسئله پی برده اند که دادهٔ بدون برچسب وقتی در کنار تعداد کمی دادهٔ با برچسب استفاده میشود، میتواند دقت را به شدت افزایش دهد.

در یادگیری نظارتی ضعیف، برچسب‌های آموزشی خطا دار، محدود و یا نادقیق هستند. ولی در مقابل، به دست آوردن این برچسب‌ها سریع تر، ارزان تر و راحت تر است و در نتیجه مجموعه بزرگتری برای آموزش به درست می آورد.

---

[1] Supervised Learning
[2] Cost function
[3] Unsupervised Learning
[4] Semi-Supervised Learning

۴.۱.۱ یادگیری تقویتی

یادگیری تقویتی[1] یک حوزه از یادگیری ماشین است که بررسی میکند چگونه عوامل نرم افزاری [2] باید در یک محیط عمل کنند تا مجموع جوایزشان بیشینه شود. در یادگیری تقویتی، معمولا محیط را به صورت یک فرایند تصمیم گیری مارکف[3] در نظر میگیرند. از این گونه الگوریتم‌ها در ماشنی‌های خودران و یا بازی‌های کامپیوتری که در مقابل یک انسان بازی میکند استفاده میشود.

۵.۱.۱ کاهش ابعاد

کاهش ابعاد[4] پروسهٔ کاهش تعداد متغیرهای رندوم در حال بررسی است با استفاده از پیدا کردن مجموعه ای از متغیرهای اصلی. به زبانی دیگر، این پروسه، پروسهٔ کاهش بعد مجموعه ویژگی[5] ها، به نام تعداد ویژگی‌ها، است. اکثر تکنیک‌های کاهش بعد یا به صورت حذف ویژگی یا استخراج ویژگی هستند. یکی از پرطرفدار ترین روش‌های کاهش بعد، تجزیه و تحلیل اجزای اصلی[6] است.

۶.۱.۱ افزایش داده‌ها

افزایش داده‌ها[7] تکنیکی است برای افزایش مصنوعی اندازهٔ مجموعهٔ آموزشی توسط ایجاد کپی از روی داده‌های موجود. این روش شامل ایجاد تغییرهای کوچک روی مجموعه داده‌ها و یا استفاده از یادگیری عمیق برای ایجاد داده‌های جدید میشود.

فرق دادهٔ افزوده شده با دادهٔ مصنوعی این است که دادهٔ افزوده شده با استفاده از تغییر کوچک روی دادهٔ آموزشی ایجاد شده است و دادهٔ مصنوعی به صورت مصنوعی توسط شبکه‌های عصبی تولید شده است.

---

[1] Reinforcement Learning
[2] Software agents
[3] Markov Decision Process
[4] Dimensionality Reduction
[5] Feature set
[6] Principal Component Analysis
[7] Data Augmentation

۷.۱.۱ تعمیم، بیش برازش و کم برازش

چالش اصلی در یادگیری ماشین این است که الگوریتم ما باید در ورودی‌های جدید که قبلا دیده نشده بودند خوب عمل کند - نه فقط آن داده‌هایی که مدل ما بر اساس آن آموزش دیده است. توانایی عملکرد خوب در ورودی‌های مشاهده نشدهٔ قبلی، تعمیم[۱] نامیده میشود.

به طور معمول، هنگام آموزش یک مدل یادگیری ماشین به یک مجموعه دادهٔ آموزشی دسترسی داریم. میتوانیم برای این مجموعه معیار سنجش خطایی در نظر بگیریم که به آن خطای آموزش[۲] میگوییم و سعی میکنیم که این خطا را کاهش دهیم. حال در الگوریتم‌های یادگیری ماشین میخواهیم علاوه بر این، خطای تست هم کاهش پیدا کند. خطای تست[۳] را به صورت امید ریاضی خطا بر روی یک ورودی جدید تعریف میکنیم.

حال خطای تست معمولا بزرگتر مساوی خطای تمرین است. برای اینکه الگوریتم ما در بهترین حالت ممکن اجرا شود میخواهیم:

- خطای تمرین را کمینه کنیم،
- و فاصلهٔ خطای تست و تمرین را نیز کمینه کنیم.

این دو یکی از مشکلات اصلی و بزرگ یادگیری ماشین هستند. کم برازش[۴] وقتی اتفاق میفتد که مدل ما نمیتواند خطای کمی را در دادهٔ تمرین به دست بیاورد. و بیش برازش[۵] زمانی اتفاق میفتد که فاصلهٔ بین خطای تمرین و تست زیاد باشد[۱۱].

۲.۱ بینایی ماشین

بینایی ماشین[۶] یک رشته از هوش مصنوعی است که به کامپیوترها و سیستم‌ها این اجازه را میدهد که اطلاعات معنی داری را برداشت کنند از تصویرهای دیجیتالی، ویدیوها و دیگر ورودی‌های تصویری - و کارهایی انجام بدهند و یا پیشنهاداتی بدهند بر حسب آن اطلاعات به دست آورده

[1] Generalization
[2] Training Error
[3] Test Error
[4] Underfitting
[5] Overfitting
[6] Computer Vision

شده. اگر هوش مصنوعی قابلیت فکر کردن به کامپیوترها میدهد، بینایی ماشین به آنها قدرت دیدن میدهد و فهمیدن میدهد.

بینایی ماشین مانند بینایی انسان‌ها کار میکند، با این تفاوت که انسان‌ها فرصت برتری دارند. دید انسان این برتری را دارد که سال‌ها مداوم اشیاء مختلف را میبیند و یاد میگیرد آن را از همدیگر تشخیص دهد، بفهمد چقدر از هم فاصله دارند، دارند حرکت میکنند یا خیر و آیا مشکلی در عکسی که میبینند وجود دارد یا خیر.

بینایی ماشین به سیستم یاد میدهد تا این عملیات را انجام دهد، ولی باید این کارها را در زمان خیلی کمتر و خیلی سریع تر انجام بدهد و از دوربین‌ها و داده و الگوریتم استفاده کند به جای شبکیه و اعصاب بینایی. به علت سرعت بالای کامپیوترها، با پیشرفت الگوریتم‌ها و مجموعه داده‌ها، کامپیوترها در بسیاری از کارها مانند تشخیص مشکل در جنس تولیدی در یک کارخانه بسیار سریع تر و دقیق تر از انسان‌ها عمل میکنند.

۱.۲.۱ بینایی ماشین چگونه کار میکند؟

بینایی ماشین به دادهٔ زیادی نیاز دارد. آن روی داده بارها و بارها آنالیز انجام میدهد تا بتواند تفاوت‌ها را تشخیص بدهد و در نهایت تصویرها را تشخیص بدهد. برای مثال، برای اینکه به کامپیوتر آموزش بدهیم که چرخ ماشین را بشناسد، باید تعداد بسیار زیادی عکس چرخ ماشین و عکس‌های مربوط به چرخ را به آن نشان دهیم تا بتواند تفاوت‌ها را تشخیص بدهد . در نهایت چرخ را بشناسد.

دو تا از تکنولوژی‌های مهم که در این راستا استفاده میشود یادگیری عمیق[۱] و شبکه عصبی پیچشی[۲] (CNN) هستند که در ادامه مفصل به آنها میپردازیم.

CNN به مدل یادگیری ماشین یا یا یادگیری عمیق کمک میکند نگاه کند توسط شکستن تصویر به پیکسل‌هایی که بهشان برچسب زده میشود. آن از برچسب‌ها استفاده میکند تا عمل کانوولوشن را انجام دهد (در ادامه توضیح داده میشود) و پیشبینی‌هایی انجام دهد در مورد اینکه چه چیزی را دارد میبیند. شبکه عصبی کانوولوشن‌ها را اجرا میکند و درصد خطای پیشبینی خود را چک میکند در تکرارهای فراوان تا زمانی که پیشبینی‌هایش به حقیقت بپیوندد. سپس آن عکس‌ها را به شکلی مشابه انسان‌ها تشخیص میدهد یا "میبیند".

[1]Deep Learning

[2]Convolutional Neural Network

همانند انسان‌ها زمانی که تصویری را از دور نگاه میکنند، CNN اول مرزهای سخت و شکل‌های ساده را تشخیص میدهد، سپس بقیه اطلاعات را اضافه میکند در حالی که تشخیص خود را تکرار میکند. یک CNN برای تشخیص یک عکس به کار میرود. برای استفاده از کاربردهای آن در ویدیو از یک شبکه عصبی تکرار شونده[1] (RNN) استفاده میشود.

۲.۲.۱ تاریخچه بینایی ماشین

دانشمندان و مهندسان حدود ۶۰ سال است در تلاش بوده اند که روش‌هایی برای ماشین‌ها طراحی کنند تا بتوانند ببینند و داده تصویری را متوجه شوند. آزمایش‌ها از سال ۱۹۵۹ شروع شد زمانی که نوروفیزیولوژیست‌ها به یک گربه تعدادی عکس را نشان دادند در تلاش برای پیدا کردن همبستگی ای بین عکس‌ها و واکنشی در مغز گربه. آن‌ها کشف کردند که مغز گربه اول به خط‌ها و مرزهای سخت واکنش نشان میدهد، و این از لحاظ عملی به این معناست که تحلیل عکس با اشکال ساده مانند خط‌های صاف شروع میشود.

تقریبا در همان زمان، اولین تکنولوژی اسکن کردن عکس بوجود آمد، که به کامپیوترها این اجازه را میداد تا تصویرها را به عدد تبدیل کنند و به دست بیاورند. دستاورد بزرگ بعدی در ۱۹۶۳ بود زمانی که کامپیوترها توانستند عکس‌های دو بعدی را به فرم سه بعدی در بیاورند. در دهه ۱۹۶۰، هوش مصنوعی به عنوان یک شاخه تخصصی آکادمیک بوجود آمد و شروع جستجوی هوش مصنوعی برای حل کردن سوال بینایی ماشین بود.

۱۹۷۴ معرفی تکنولوژی تشخیص تصویری کاراکترها[2] بود که میتوانست متن چاپ شده را با هر فونتی تشخیص بدهد. مشابهاً، تشخیص کاراکتر هوشمند[3] میتوانست متن‌های دستنوشته را با استفاده از شبکه‌های عصبی تشخیص بدهد. از آن دوران، این دو تکنولوژی کاربردهای زیادی در زندگی روزمره انسان‌ها پیدا کرده اند از جمله تشخیص پلاک، ترجمه ماشینی، تحلیل اسناد و صورت حساب و دیگر کاربردهای فراوان آن.

سال ۱۹۸۲، دانشمند عصب شناس David Marr نشان داد که بینایی پله پله است و الگوریتم‌هایی را معرفی کرد برای تشخیص مرزها، گوشه‌ها، خم‌ها و اشکال سادهٔ دیگر. همزمان با او، دانشمند کامپیوتر Kunihiko Fukushima شبکه ای از سلول‌ها را بوجود آورد که میتوانست الگوها را تشخیص بدهد. آن شبکه، به نام Neocognitron، شامل لایه‌های پیچشی در یک شبکه

[1] Recurrent Neural Network

[2] Optical Character Recognition

[3] Intelligent Character Recognition

عصبی بود.

تا سال ۲۰۰۰، تمرکز تحقیقات بر روی تشخیص اشیاء بود، و تا ۲۰۰۱، اولین اپلیکیشن تشخیص صورت بلادرنگ بوجود آمد. استاندارد سازی چگونگی برچسب زدن مجموعه داده‌های تصویری در دههٔ ۲۰۰۰ اتفاق افتاد. در ۲۰۱۰، محجموعه دادهٔ ImageNet در دسترس قرار گرفت. آن شامل میلیون‌ها عکس برچسب دار و هزاران کلاس اشیاء بود و و پایه گذاری برای مدل‌های CNN و یادگیری عمیق امروزی بود. در ۲۰۱۲ یک تیم از دانشگاه تورنتو یک مدل CNN را وارد یک مسابقهٔ تشخیص شیء کردند. آن مدل، AlexNet، به شدت درصد خطا را برای تشخیص عکس کاهش داد. بعد از این مدل، درصد خطاها به زیر ۱۰ درصد رسیده اند.

۳.۲.۱ وظایف متداول بینایی ماشین

- دسته بندی عکس‌ها[1]:

دسته بندی عکس‌ها یکی از مسائل اصلی در زمینه بینایی ماشین است و اولین وظیفه[2] ای است که به موفقیت چشمگیری دست یافت در سال ۲۰۱۲. در این وظیفه، مدل به عکس یک کلاس نسبت میدهد با توجه به مجموعه ای از کلاس‌های از پیش تعریف شده. شکل صفحه بعد ۱.۱ مجموعه دادهٔ معروف CIFAR-10 است که شامل ۸۰ میلیون عکس در ۱۰ کلاس است. در وظیفهٔ دسته بندی عکس‌ها، مدل آموزش داده میشود تا عکس ورودی را به یکی از ۱۰ کلاس از پیش تعیین شده نسبت بدهد، همانطور که در شکل صفحهٔ بعد میبینید.

کامپیوتر تصویر را به صورت مجموعه ای از پیکسل‌ها میبیند و تحلیل میکند. این کار را به اینصورت انجام میدهد که هر عکس را به صورت آرایه ای از ماتریس‌ها در نظر میگرد که سایز ماتریس‌ها به کیفیت تصویر مربوط است. الگوریتم اول تصویر را به سلسله ای از مشخص ترین خصوصیاتش جدا میکند تا حجم کار برای تشخیص دهنده آخر کمتر شود. این خصوصایت به تشخیص دهنده کمک میکند تا تشخیص بدهد این تصویر به کدام دسته تعلق دارد.

[1] Image Classification
[2] Task

شکل ۱.۱: مثالی از مجموعه دادهٔ CIFAR-10[۱۸]

- شناسایی اشیاء[1]:

همانطور که گفته شده بود، دسته بندی عکس‌ها، به هر عکس یک کلاس یا برچسب اختصاص میدهد بر حسب مجموعه از قبل تعریف شده برچسب‌ها. شناسایی اشیاء[2] یک نمونه خاص از یک کلاس را میشناسد. مثلا در وظیفهٔ دسته بندی عکس‌ها، تمرکز بر روی دسته بندی عکس‌های صورت است، ولی در شناسایی اشیاء تمرکز بر روی شناختن شخص و متمایز شمردن او با بقیهٔ افراد است. در نتیجه به وظیفهٔ شناسایی اشیاء میتوان به چشم یک وظیفهٔ خوشه بندی[3] نگاه کرد.

- تشخیص اشیاء و محل یابی[4]:

یکی دیگر از وظیفه‌های مهم بینایی ماشین تشخیص اشیاء است. منظور از تشخیص اشیاء، تشخیص یک نمونه از یک شیء از یک کلاس خاص درون یک عکس است. تشخیص شیء معمولاً یکی از وظایف اولیه قبل از محاسبات بعدی است، مانند تشخیص چهره، که باید در آن اول صورت را تشخیص بدهیم و از تصویر جدا کنیم و سپس به سراغ تشخیص

[1] Object Identification
[2] Object Identification
[3] Clustering
[4] Object Detection and Localization

چهره برویم.

- بخش بندی تصویر[1]:

بخش بندی عکس، روشی است که در آن یک عکس دیجیتال را به زیرگروه‌هایی به نام بخش تصویر تقسیم بندی میکنیم و از پیچیدگی تصویر میکاهیم و محاسبات بعدی را ساده تر میکنیم. در واقع، بخش بندی تصویر اختصاص دادن برچسب به هر پیکسل است به طوری که اشیاء و بخش‌های مهم تصویر را از همدیگر جدا کند. در تصویر زیر هر بخش از تصویر با یک رنگ از دیگری جدا شده است.

شکل ۲.۱: تصویر نشان دهندهٔ بخش‌های یک تصویر که با رنگ‌های مختلف جدا شده اند

- تخمین حالت[2]:

تخمین حالت میتواند معانی مختلفی داشته باشد بسته به وظیفه ای که در نظر داریم. برای اشیاء جامد، معمولاً به این معناست که جهت و مختصات آن شیء را نسبت به دروبین در فضای سه بعدی تخمین بزنیم. این به طور خاص برای ربات‌ها بسیار کاربردی است که

[1]Image Segmentation
[2]Pose Estimation

بتوانند با محیطشان ارتباط فیزیکی برقرار کنند. همچنین در واقعیت مجازی برای قرار دادن اطلاعات مجازی بر روی دنیای واقعی استفاده میشود.

برای اجسام ناجامد، تخمین حالت به این معناست که مختصات تکه‌های مختلف آن جسم را نسبت به همدیگر مشخص کنیم. به طور دقیق تر، وقتی به انسان به چشم یک شیء ناجامد نگاه کنیم، تخمین حالت به معنای تشخیص حالت انسان (نشستن، ایستادن، دویدن و غیره) و یا تشخیص زبان اشاره است.

- دنبال کردن اشیاء[1]:

تا کنون تمامی کاربردها مربوط به تحلیل عکس‌ها و کار کردن با دادهٔ غیر زمانی بود، حال وارد کاربردهای بینایی ماشین در داده‌های زمانی و به طور خاص، ویدیو میشویم.

اولین کاربرد آن، دنبال کردن اشیاء است. دنبال کردن اشیاء پروسه ایست که در آن کامپیوتر میتواند شیء را تشخیص دهد، متوجه شود و آن را در طول رشته ای از عکس‌ها و یا ویدیوها دنبال کند. این یکی از پرکابردترین استفاده‌های هوش مصنوعی و یادگیری ماشین است، و این امکان را میدهد که نیازهای تحلیل داده‌های تصویری شما را اتوماتیک و راحت تر و سریع تر کند. ساختار یادگیری عمیق زیرزمینه ای الگوریتم‌های این بخش، الهام گرفته شده از سیستم عصبی طبیعی ما انسان‌هاست که یک شبکه ای لایه لایه و پیچیده ایجاد میکند که این لایه‌ها به همدیگر داده انتقال میدهند و از همدیگر یاد میگیرند.

با استفاده از دنبال کردن اشیاء میتوانیم کارهای معنی دار بر روی دادهٔ تصویری که توسط انواع مختلف دوربین به دست آمده است انجام دهیم. با به کار بردن الگوریتم‌های مناسب دنبال کردن اشیاء و ترکیب آن با مدل‌های دنبال کننده، میتوانیم یک ماشین را آموزش بدهیم تا نه تنها یک یا چند شیء مختلف یا انسان‌های مختلف را شناسایی کند بلکه آن‌ها را در فریم‌های متوالی نیز تشخیص دهد و مسیر آن‌ها را در یک رشتهٔ ویدیو شناسایی کند.

- تشخیص فعالیت‌های انسانی[2]:

تشخیص فعالیت‌های انسانی یعنی شناسایی زمانی که یک انسان در یک عکس یا فیلم، یک عمل خاص را انجام میدهد. این کار تعلق میگیرد به آن دسته از کارهایی که فقط میتوان روی یک دنباله از عکس‌ها انجام داد. مانند اینکه نمیتوانیم معنی یک جمله را فقط از روی

[1] Object Tracking
[2] Action Recognition

یک کلمه متوجه شویم، نمیتوانیم یک عمل را تنها از روی یک عکس تشخیص بدهیم. الگوریتم‌های بینایی ماشین میتوانند تربیت شوند تا انواع زیادی از اعمال را شناسایی کنند، از دویدن و خوابیدن گرفته تا نوشیدن، افتادن و یا دوچرخه سواری.

- تخمین حرکت[1]:

تخمین حرکت، حرکت اجسام را در یک دنباله از عکس‌ها بررسی میکند تا بتواند بردارهایی پیدا کند که نشان دهندهٔ تخمین حرکت این اجسام است. این کاربرد بسیار مهم است در کارهایی که تمرکز میکند بر روی سرعت واقعی یا جهت حرکتی که در ویدیو شکار شده است.

تخمین حرکت بسیار کاربردی است در صنایعی مانند کنترل ترافیک که در آن سرعت و جهت ماشین‌ها را تخمین میزنند و یا در صنعت سرگرمی که میخواهند حرکت را پیدا کنند تا بر روی آن جلوه‌های تصویری اعمال کنند.

[1]Motion Estimation

فصل ۲

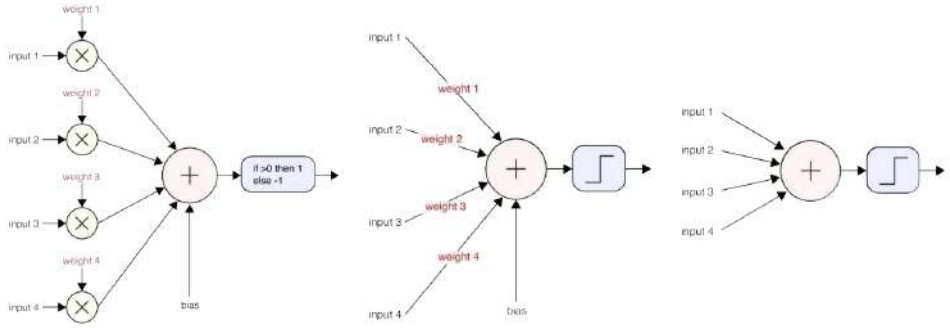
یادگیری عمیق

شبکه عمیق پیشخور[۲]، که با نام پرسپترون چندلایه[۳](MLP) نیز شناخته میشود، مدل‌های اساسی و واجب یادگیری عمیق[۴] هستند. هدف یک شبکه پیشخور این است که یک تابع f^* را تخمین بزند. برای مثال، برای یک دسته بندی کننده، $y = f^*(x)$ یک ورودی x را به یک دستهٔ y اختصاص میدهد. یک شبکه پیشخور، یک نگاشت را تعریف میکند $y = f(x;\theta)$ و مقدار θ ای را یاد میگیرد که بهترین تخمین تابع را نتیجه میدهد.

به این مدل‌ها پیشخور میگویند برای اینکه جریان اطلاعات از x شروع میشود، سپس به محاسبات میانی که f را تخمین میزند ادامه پیدا میکند و در نهایت به خروجی نهایی y میرسد. در این پروسه هیچ ارتباط بازخوردی[۵] ای وجود ندارد که در آن خروجی مدل به خودش بازگردانده شود. زمانی که شبکه‌های عصبی پیشخور گسترش می یابند تا ارتباط بازخوردی را نیز شامل شوند، به اینگونه از مدل‌ها شبکه عصبی تکرار شونده[۶] میگوییم که در آینده به آنها میپردازیم.

شبکه‌های پیشخور اهمیت زیادی برای کارکنان در عرصهٔ یادگیری ماشین دارند. آن‌ها پایهٔ بسیاری از مهم ترین کاربردهای یادگیری ماشین در صنعت هستند. برای مثال، شبکه‌های پیچش که برای تشخیص اشیاء در عکس‌ها استفاده میشوند یک مدل خاص از شبکه‌های پیشخور هستند. شبکه‌های پیشخور قدم اصلی در پیدایش شبکه‌های تکرار شونده هستند که مدل‌های پایه ای و اصلی

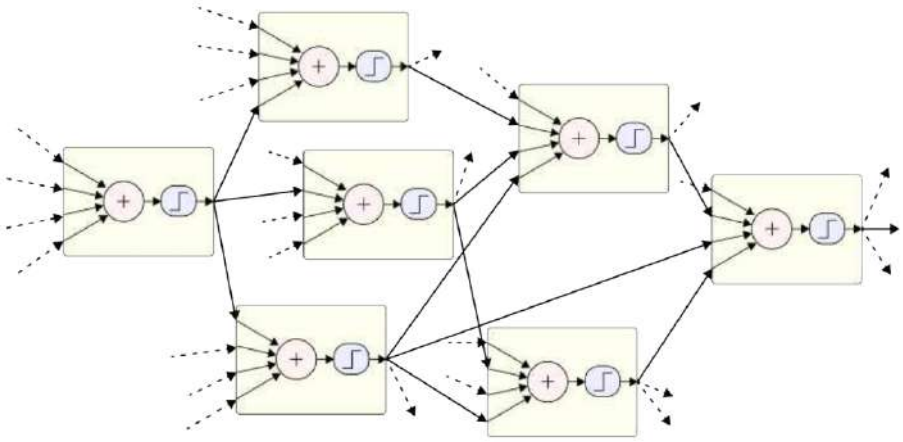
[2]Deep Feedforward Networks
[3]Multilayer Perceptron
[4]Deep Learning
[5]Feedback Connection
[6]Recurrent Neural Network



شکل ۱.۲: شکل سمت راست نشان میدهد چگونه وزن‌ها با ورودی‌ها ترکیب میشوند و خروجی را تولید میکنند در یک واحد. شکل‌های سمت راستی مدل‌هایی خلاصه شده از شکل سمت چپ هستند[۱۰]

برای تحلیل زبان[۱] هستند.

شبکه‌های عصبی پیشخور، شبکه نام دارند چون معمولاً به صورت شبکه ای از توابع هستند. مدل را به صورت یک گراف جهت دار بدون دور نشان میدهیم که نشان دهنده روابط توابع با همدیگر است و اینکه چگونه با هم ترکیب میشوند تا مدل را تشکیل دهند. برای مثال ما ممکن است سه تابع داشته باشیم $f^{(1)}, f^{(2)}, f^{(3)}$ که در یک زنجیره به هم متصل هستند تا $f(x) = f^{(3)}(f^{(2)}(f^{(1)}(x)))$ را شکل بدهند. این ساختارهای زنجیره ای پر استفاده ترین ساختار در شبکه‌های عصبی هستند. در این مثال، به $f^{(1)}$ لایهٔ اول، به $f^{(2)}$ لایهٔ دوم میگویند و همینطور تا آخر. به طول کلی مدل، عمق[۲] مدل میگویند و نام یادگیری عمیق نیز از همین اسم گذاری نشئت میگیرد. به لایهٔ پایانی شبکه لایهٔ خروجی[۳] و به لایه‌های میانی، لایه‌های پنهان[۴] میگویند. [۱۱]



شکل ۲.۲: یک شبکه چند لایه که نشان میدهد چگونه خروجی یک واحد به واحدهای بعدی ورودی داده میشود[۱۰].

۱.۲ وزن‌ها و لایه‌ها

شبکه‌های عصبی عمیق، گراف‌های محاسباتی پیشخور هستند که از هزاران نورون (واحد[۱]) به هم وصل شده تشکیل شده اند، که همانند رگرسیون خطی [۹]، عملیات جمع وزن دار ورودی‌ها را انجام میدهد $s_i = w_i^T x_i + b_i$ و در ادامه یک تابع فعال سازی[۲] غیرخطی روی آن اعمال میکند، $y_i = h(s_i),$ همانطور که در شکل ۱.۲ نشان داده شده است. x_i ورودی‌های واحد i ام هستند، w_i و b_i وزن‌ها و تعصب[۳] قابل آموزش هستند، s_i خروجی جمع خطی وزن دار است، و y_i خروجی نهایی است بعد از اینکه s_i به تابع فعال سازی h داده میشود. خروجی هر واحد که معمولاً به آنها فعالسازی میگویند، به واحدهای بعدی ورودی داده میشوند، همانطور که در شکل ۲.۲ نشان داده شده است.

[1]Natural Language Processing
[2]Depth
[3]Output Layer
[4]Hidden Layer
[1]Unit
[2]Activation Function
[3]Bias

اولین واحدهای به این صورت به نام پرسپترون[1] [۲۶] بودند همانطور که در شکل ۲.۲ میبینید. در شکل سمت چپ ۲.۲ مشخصا تمامی عملیات و وزن‌ها را میبینید ولی در شکل‌های بعدی حذف شده اند.

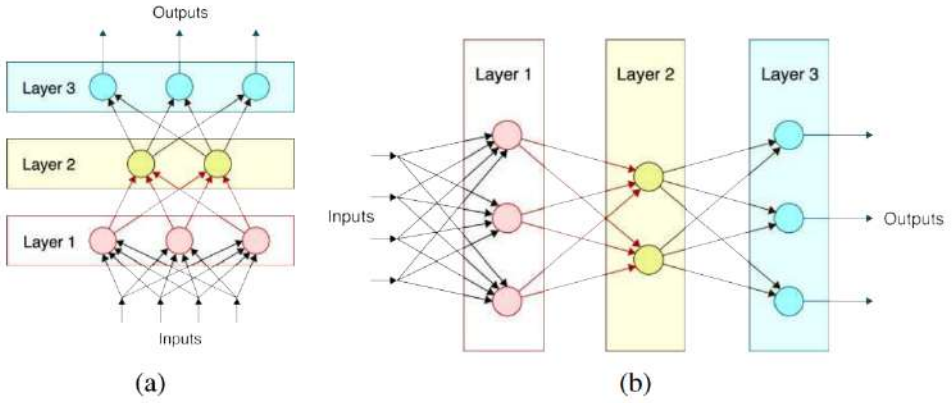
به جای اینکه در یک گراف محساباتی غیر منظم به هم متصل باشند مانند شکل ۲.۲ ، شبکه‌های عصبی معمولا به صورت لایه‌های متوالی طبقه بندی شده اند مانند شکل ۳.۲. میتوانیم به تمام واحدهای درون یک لایه به عنوان یک بردار نگاه کنیم، که ترکیب خطی متناظر آن را به صورت

$$s_l = W_l x_l, \tag{۱.۲}$$

مینویسیم، که در آن x_l ورودی‌های لایه l، W_l ماتریس وزن‌ها، و s_l جمع وزن دار است که بر روی آن تابع فعالسازی اعمال میشود،

$$x_{l+1} = y_l = h(s_l). \tag{۲.۲}$$

لایه ای که در آن یک ماتریس کامل[2] وزن‌ها استفاده میشود برای ترکیب خطی، لایهٔ کاملاً متصل[3] نام دارد، از انجایی که تمامی ورودی‌های یک لایه به تمام خروجای‌هایش متصل است. همانطور که در بخش‌های آینده خواهیم دید برای پردازش کردن پیکسل‌ها در لایه‌های اولیه به جای لایه‌های کاملاً متصل از لایه‌های پیچشی استفاده میکنیم، برای ناوردایی فضایی و بازده بالاتر. به شبکه ای که فقط از لایه‌های کاملاً متصل تشکیل شده است معمولا پرسپترون چند لایه(MLP) میگویند.



شکل ۳.۲: در روش متفاوت برای نشان دادن شبکه‌های عصبی. در شکل سمت چپ ورودی‌ها پایین هستند و در شکل سمت راست ورودی‌ها سمت چپ[۱۰].

۲.۲ تابع‌های فعالسازی

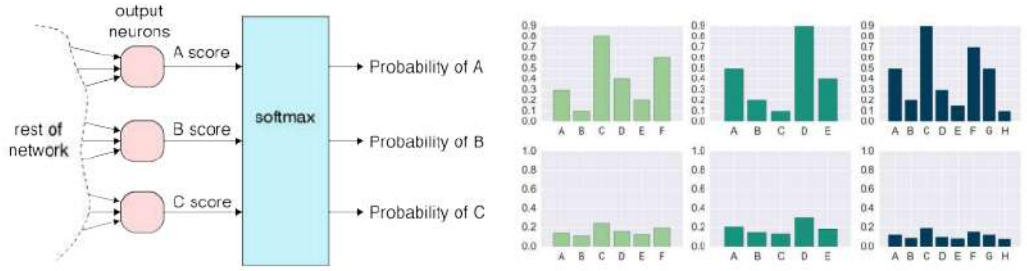
بیشتر شبکه‌های عصبی اولیه[۲۷]، از توابع سیگمویدی که مشابه آنها در رگرسیون خطی استفاده میشود استفاده میکردند. شبکه‌های جدیدتر با شروع از [۲۳]، از ReLU[1]ها و انواع آن استفاده کردند و میکنند. تابع فعال سازی ReLU به صورت

$$h(y) = max(0, y) \tag{۳.۲}$$

تعریف میشود و در گوشه بالا سمت چپ شکل ۵.۲ نشان داده شده است، همراه با دیگر توابع متداول که در مقاله‌های مختلف استفاده شده اند.

با اینکه ReLU در حال حاضر پرطرفدارترین تابع فعال سازی است، برخی دانشمندان این تذکر را میدهند که ReLU میتواند در تمرین شکننده باشد و اصطلاحاً بمیرد. برای مثال، یک گرادیان بزرگ که از یک واحد ReLU عبور میکند میتواند باعث شود وزن‌ها به طوری تغییر کنند که برخی نورون‌ها هیچ گاه دیگر با هیچ داده ای فعال نشوند. اگر این اتفاق بیفتد، جریان گرادیانی که از

[1]Perceptrons
[2]Dense
[3]Fully Connected
[1]Rectified Linear Units

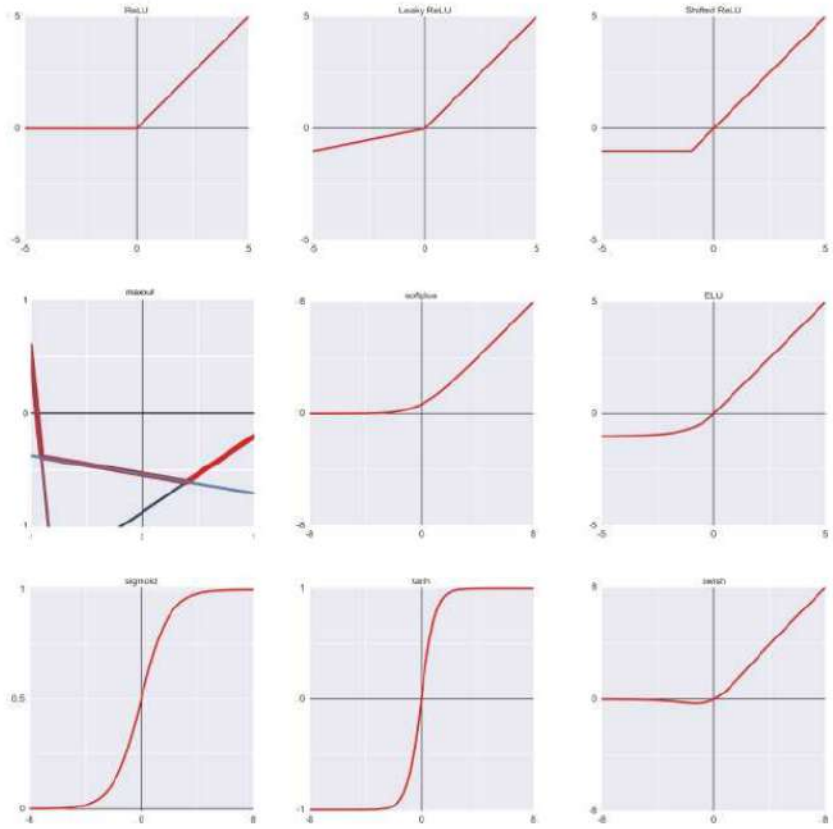


شکل ۴.۲: سمت چپ یک لایه تابع softmax برای تبدیل مقدار فعال سازی به احتمال حضور در یک کلاس. سمت راست ردیف بالا مقدار فعال سازی‌ها را نشان میدهد و در ردیف پایین نتیجهٔ رد کردن این مقادیر از یک لایه softmax را میبینیم[۱۰].

یک واحد عبود میکند همیشه صفر خواهد بود از آن به بعد. به این معنا که یک واحد ReLU میتواند به طور غیر قابل برگشت بمیرد در طول تمرین. با تنظیم کردن درست نرخ تمرین، این ایراد کمتر پیش می آید. همچنین توابع فعال سازی مشابه دیگری در شکل ۵.۲ ارائه شده است که سعی شده در آن از این ایراد جلوگیری کند.

برای لایهٔ آخر در شبکه‌هایی که برای دسته بندی استفاده میشوند، از تابع softmax استفاده میشود. این تابع به طور معمول برای تبدیل مقادیر واقعی که احتمال وجود در یک کلاس استفاده میشود، همانطور که در شکل ۴.۲ دیده میشود. در نتیجه میتوانیم به مجموعه یکی مانده به آخر نورون‌ها به این چشم نگاه کنیم که دارند جهت‌های در فضای فعال سازی مشخص میکنند که نزدیک ترین حالت به لگاریتم احتمال حضور در کلاس مربوط به خود باشد، و همزمان لگاریتم احتمال حضور در کلاس‌های دیگر را کمینه کند. از آن جایی که ورودی‌های در جهت جلو فقط جریان پیدا میکنند و به کلاس‌های پایانی و احتمالات میرسند، شبکه‌های پیشخور متمایز کننده[1] هستند. به این معنا که هیچ ایده ای از مدل آماری کلاس‌های خروجی ندارند و یا هیچ روشی برای تولید کردن مثال از آن فضای آماری ندارند.

[1]Discriminative

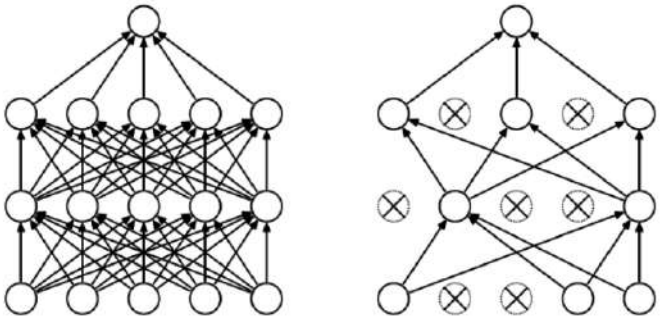


شکل ۵.۲: برخی از توابع فعال سازی پرطرفدار[۱۰]

۳.۲ نرمال سازی و منظم سازی

همانطور که در دیگر روش‌های یادگیری ماشین دیده ایم، از منظم سازی[1] برای جلوگیری از بیش برازش شبکه عصبی استفاده میشود تا مدلمان را بهتر به داده‌های دیده نشده تعمیم بدهیم. در این بخش متدهای متداول برای این کار را بررسی میکنیم، به طور خاص بیرون ریزی[2] و نرمال سازی دسته ای[3] که مخصوص شبکه‌های عصبی هستند.

[1]Regularization
[2]Dropout
[3]Batch Normalization



شکل ۶.۲: زمانی که از بیرون ریزی استفاده میکنیم، تعدادی از واحد را صفر میکنیم و در واقع از روند تمرین آن‌ها را حذف میکنیم. در سمت چپ شبکه را قبل از بیرون ریزی مشاهده میکنید و در سمت راست بعد از بیرون ریزی[۱۰].

۱.۳.۲ بیرون ریزی

بیرون ریزی[1] یک تکنیک منظم سازی است که در [۲۸] معرفی شده است، که در آن هر دستهٔ کوچک در حال تمرین، درصدی از واحدها (معمولاً ۵۰ درصد) را صفر میکنیم، همانطور که در شکل ۶.۲ نشان داده شده است. صفر کردن واحدها به طور رندم باعث ایجاد نویز و اختلال در فرایند تمرین میشود و همچنین جلوگیری میکند از اینکه شبکه واحدهایش برای دادهٔ خاصی شخصی سازی کند. هر دوی این کارها باعث میشود که بیش برازش جلوگیری شود و تعمیم پذیری را بهبود میبخشد.

چون صفر کردن p تا از واحدها امید ریاضی هر جمعی که این واحد عضوش است را با کسر $1-p$ کاهش میدهد، مجموع وزن دارن s_i در هر لایه را در طول تمرین ضرب در $(1-p)^{-1}$ میکنیم. در زمان تست کردن وقتی شبکه را اجرا میکنیم دیگر بیرون ریزی نداریم و همچنین مجموع‌ها را ضرب در عدد خاصی نمیکنیم برای جبران بیرون ریزی.

[1] Dropout

۲.۳.۲ نرمال سازی دسته ای

بهینه سازی وزن‌ها در یک شبکه عمیق کار پیچیده و سختی است و ممکن است بسیار کند باشد و یا هیچ گاه همگرا نشود. یکی از مشکلات قدیمی در بهینه سازی مرحله ای شایسته سازی[1] ضعیف است، که در آن اعضای گرادیان اندازه شان بسیار متفاوت است. در حالی که ممکن است در بعضی مواقع این مشکل را با پیش شایسته سازی حل کرد که در آن هر المان از گرادیان را قبل از قدم برداشتن تغییر مقیاس میدهیم، معمولاً ترجیح داده میشود که کنترل عدد شایستگی سیستم را موقع شکل گیری مسئله به دست گرفت.

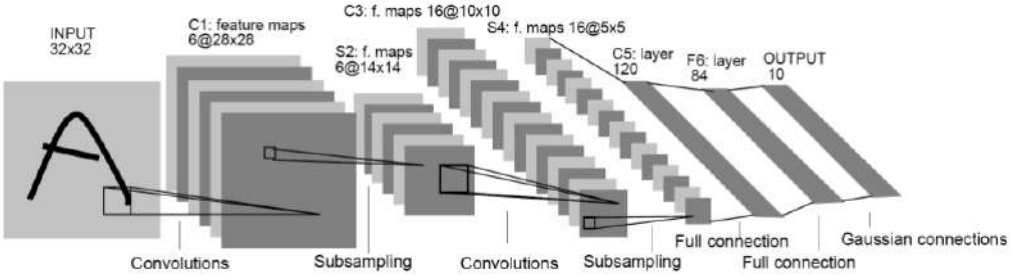
در شبکه‌های عمیق، یک روش که در آن شایسته سازی ضعیف میتواند خودش را نشان دهد این است که اندازه وزن‌ها یا فعال سازی‌ها در لایه‌های متوالی نامتوازن شود. در نظر بگیرید که شبکه ای داریم و وزن‌های یک لایه را ۱۰۰ برابر میکنیم و وزن‌های لایهٔ بعدی را تقسیم بر ۱۰۰. چون که فعال ساز ReLU در هر دو دامنه خود خطی است، خروجی لایهٔ دوم همچنان ثابت خواهد بود در صورتی که خروجی لایهٔ اول ۱۰۰ برابر خواهد بود. در طول یک قدم از نزول گرادیان[2]، مشتق‌ها نسبت به وزن‌ها بسیار متفاوت خواهند بود بعد از این تغییر مقیاس، و در واقع مخالف اندازه مقیاس وزن‌ها خواهند بود و نیازمند قدم‌های کوچک در نزول گرادیان خواهد بود تا از مقصد عبور نکنیم.

ایدهٔ پشت نرمال سازی دسته ای[۱۵] این است که مقیاس فعال سازی‌ها را در یک واحد داده شاده عوض کنیم تا آن‌ها واریانس واحد و میانگین صفر داشته باشند (که برای یک واحد ReLU به این معنا است که در نصف حالات فعال خواهد بود. ما این نرمال سازی را با در نظر گرفتن تمامی نمونه‌های تمرینی n در یک دسته بندی کوچک B انجام میدهیم و میانگین و واریانس را برای واحد i اینگونه محاسبه میکنیم

$$\mu_i = \frac{1}{|B|} \sum_{n \in B} s_i^{(n)} \tag{۴.۲}$$

$$\sigma_i^2 = \frac{1}{|B|} \sum_{n \in B} (s_i^{(n)} - \mu_i)^2 \tag{۵.۲}$$

[1] Conditioning
[2] Gradient Descent



شکل ۷.۲: ساختار شبکه LeNet-5، یک شبکه عصبی پیچشی برای تشخیص عدد. این شبکه از چندید کانال برای هر لایه استفاده میکند و بین لایه‌های پیچش از روش‌های نمونه کاهی استفاده میکند و در نهایشت چند لایهٔ کاملا متصل استفاده شده که برای هر یک از ۱۰ عدد در حال تشیص یک مقدار فعال سازی تولید میکند [۱۹].

$$\hat{s}_i^{(n)} = \frac{s_i^{(n)} - \mu_i}{\sqrt{\sigma_i^2 + \epsilon}}. \tag{۶.۲}$$

بعد از نرمال سازی دسته ای، فعال سازی $\hat{s_i^{(n)}}$ میانگین صفر و واریانس یک خواهد داشت. به این دلیل ضریب γ_i و تعصب β_i را به هر واحد i اضافه میکنیم و خروجی مرحله نرمال سازی دسته ای را

$$y_i = \gamma_i \hat{s}_i + \beta_i \tag{۷.۲}$$

تعریف میکنیم.

۴.۲ شبکه‌های عصبی پیچشی

در بخش قبلی تکه‌های اصلی برای ساختن یک شبکه عصبی عمیق را بررسی کردیم. ولی در این بخش مهم ترین المان یک شبکه عصبی عمیق برای تحلیل عکس‌ها و بینایی ماشین را معرفی میکنیم، یعنی تربیت و استفاده از لایه‌های پیچش[1]. ایدهٔ شبکه‌های عصبی پیچشی توسط [۱۹]

[1]Convolutional layer

محبوب شد که در آن LeNet-5 را معرفی کردند که در شکل ۷.۲ میبینید.

به جای وصل کردن تمامی واحدهای یک لایه به تمامی واحدهای لایهٔ بعدی، شبکه‌های پیچشی هر لایه را نقشه‌های خصوصیات[1] دسته بندی میکند[۱۹]، که میتوانیم به آن‌ها به چشم صفحه‌های موازی یا کانال‌ها که در شکل ۷.۲ مشاهده میکنید نگاه کنیم. در یک لایهٔ پیچشی، اجتماع وزن دار فقط بر روی یک بازهٔ کوچک اعمال میشود، و وزن‌ها برای تمام پیکسل‌ها یکسال هستند.

در شبکه‌های عصبی به جای اینکه یک فیلتر را بر روی هر کانال رنگی اعمال کنیم، پیچش شبکه عصبی به طور خطی فعال سازی‌های هر C_1 کانال ورودی از لایهٔ قبلی را ادغام میکند و از کرنل‌های پیچش مختلف برای هر C_2 کانال خروجی استفاده میکند همانطور که در شکل ۸.۲ میبینید. این کار منطقی است چون هدف شبکه‌های پیچشی این است که خصوصیت‌های محلی پیدا کند و سپس آن‌ها را با هم به روش‌های مختلف ترکیب کند تا خصوصیت‌های با معنی تر و جداکننده تر تولید کند.

با این توصیفات میتوانیم مجموع خطی که در لایه‌های پیچشی به کار میرود را به صورت زیر بنویسیم.

$$s(i,j,c_2) = \sum_{c_1 \in \{C_1\}} \sum_{(k,l)\in\mathbb{N}} w(k,l,c_1,c_2)x(i+k,j+l,c_1) + b(c_2). \qquad (۸.۲)$$

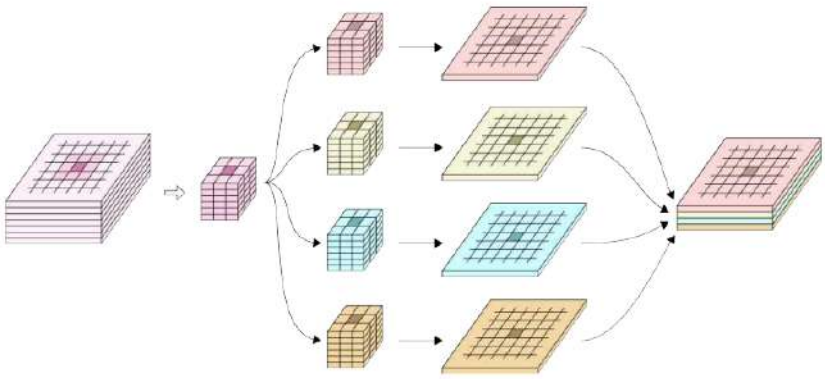
در اشکال شبکه عصبی مانند شکل ۷.۲ مرسوم است که اندازه هستهٔ پیچش S را نمایش بدهند و همچنین تعداد کانال‌های یک لایه C و تنها بعضی وقت‌ها ابعاد تصویر. دقت کنید که در بعضی از مدل‌ها مانند GoogLeNet[۲۹] از پیچش‌های 1×1 استفاده میکنند، که در واقع عمل پیچش را انجام نمیدهد و صرفاً کانال‌های مختلف را به ازای هر پیکسل ترکیب میکنند، عموماً به هدف کاهش ابعاد فضای خصوصیات.

چون که وزن‌ها در یک هستهٔ پیچش[2] برای تمام پیکسل‌ها در یک لایه و کانال داده شده یکسان است، این وزن‌ها در واقع مشترک هستند بین یال‌ها اگر تمامی اتصالات بین پیکسل‌های دو لایهٔ مختلف را بکشیم. این به این معناست که تعداد بسیار کمتری وزن را نسبت به لایهٔ کاملاً متصل نیاز است یاد بگیریم. همچنین به این معناسی که در زمان پس انتشار، تغییر وزن هسته‌ها جمع میشود روی تمام پیکسل‌ها در یک لایه یا کانال.

برای کاملا مشخص کردن رفتار یک لایهٔ پیچشی، همچنان باید چند پارامتر دیگر را نیز

[1] Feature maps

[2] Convolution



شکل ۸.۲: پیچش ۲ بعدی با چندین کانال ورودی و خروجی[۱۰]. هر هستهٔ ۲ بعدی به عنوان ورودی تمامی C_1 کانال لایهٔ قبلی را میگیرد، پنجرهٔ کوچکی را مشخص میکند، و سپس در یکی از C_2 کانال خروجی لایهٔ بعدی مقادیر را تولید میکند. برای هر کانال خروجی، $S^2 \times C_1 \times C_2$ وزن هسته داریم (S اندازه عرض پنجره محاسباتی است).

مشخص کنیم. این‌ها شامل:

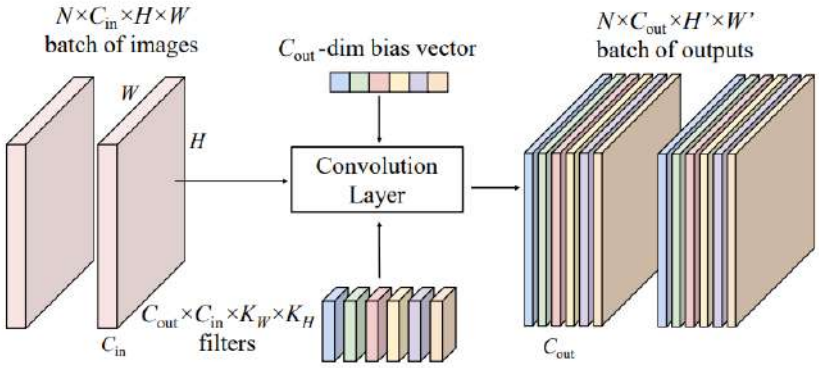
- لایه گذاری[۱]: شبکه‌های اولیه مانند LeNet-5 از لایه گذاری استفاده نمیکردند که این باعث کوچک شدن تصویر پس از هر لایهٔ پیچش میشد. شبکه‌های جدید از مدل‌های متخلف لایه گذاری و اندازه‌های مختلف استفاده میکنند.

- قدم زدن[۲]: قدم زدن پایه برای پیچش یک پیکسل است، ولی همچنین ممکن است که پیچش را در هر nth سطر و ستون محاسبه هست. برای مثال اولین لایهٔ پیچشی در AlexNet ۱۱.۲ از مقدار قدم زدن ۴ استفاده میکند. میزان معمول و رایج برای قدم زدن ۲ است.

- فاصله گذاری[۳]: فضای خالی اضافه میتواند بین پیکسل‌ها اضافه شود هین پیچش که به آن فاصله گذاری میگویند.

- گروه کردن[۴]: در حالی که به صورت پایه، تمام کانال‌های ورودی استفاده میشوند تا هر

[1] Padding
[2] Stride
[3] Dilation
[4] Grouping



شکل ۹.۲: پیچش ۲ بعدی با چندین دسته، کانال خروجی و ورودی. زمانی که نزول گرادیان دستهٔ کوچک انجام میدهیم، یک دستهٔ کامل از عکس‌های آموزشی یا خصوصیت‌ها به لایهٔ پیچشی پاس داده میشود، که آن به ازای هر کانال خروجی یک مقدار تولید میکند[۳۰].

کانال خروجی را تولید کنند، میتوانیم ورودی‌ها و خروجی‌ها را گروه بندی کنیم تا جداگانه پیچش رویشان انجام شود

۵.۲ ادغام و بازگشت ادغام

همانطور که در بخش قبل صحبت کردیم، قدم زدن بزرگتر از ۱ میتواند استفاده شدود تا کیفیت و ابعاد یک لایه را کاهش دهد، همانطور که در لایهٔ اول AlexNet استفاده شده است ۱۱.۲. زمانی که وزن‌های داخل هستهٔ پیچش یکسان هستند و مجموعشان برابر ۱ میشود، به این کار ادغام میانگین[1] گفته میشود و به طور معمول به صورت کانال به کانال اعمال میشود.

یکی دیگر از روش‌های بسیار مورد استفاده این است که بیشینه پاسخ در یک پنجرهٔ مربعی را انتخاب کنیم که به این ادغام بیشینه[2] میگویند. قدم زدن و اندازهٔ پنجرهٔ متداول برای ادغام بیشینه 2 و پنجره‌های 2×2 ، بدون اشتراک و یا 3×3 با اشتراک است. به لایه‌های ادغام بیشینه میتوان به صورت یای منطقی نگاه کرد، از آنجایی که نیاز است تنها یک واحد از آن محدوده روشن باشد

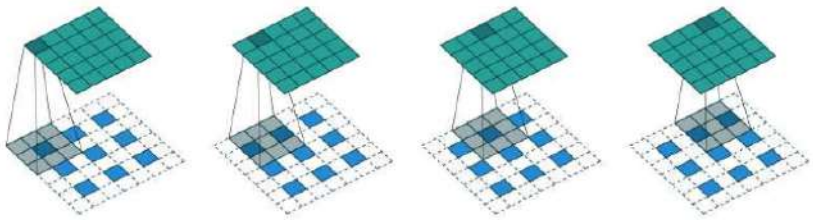
[1] Average Pooling
[2] Max Pooling

تا جوابی از کل دریافت کنیم. این‌ها انتظار میروند که بر روی ورودی‌ها جا به جایی ناپذیری ایجاد کنند ولی اکثر شبکه‌های عمیق آنقدر جا به جایی ناپذیر نیستند. برای اطلاعات بیشتر به [۳۲] مراجعه کنید.

یک مشکلی که به وفور بیان میشود این است که چگونه برای لایهٔ ادغام پس انتشار انجام دهیم. عملگر ادغام بیشینه مانند یک کلید عمل میکند که یکی از ورودی‌ها را به خروجی متصل میکند. در نتیجه در طول پس انتشار، ما فقط نیاز داریم که خطا و مشتق‌ها را پاس بدهیم به واحدی که بیشینه فعال است، تا زمانی که یادمان بماند کدام واحد این پاسخ بیشینه را داشت.

در حالی که میتوان از بازگشت ادغام[1] استفاده کرد تا تقریبا اثرات عملیات ادغام را برگرداند، اگر بخواهیم یک لایهٔ پیچش را برگردانیم، میتوانیم نگاه کنیم به متغیرهای یادگرفته شده از عمل الحاق[2]. آسان ترین راه برای به تصویر کشیدن این عملیات به این صورت است که ستون و سطر صفر بین پیکسل‌های ورودی اضافه کنیم، و در ادامه روی آن عمل پیچش معمولی انجام دهیم ۱۰.۲. به این عمل گاهاً پیچش به عقب[3] با قدم کسری[4] میگویند، ولی به صورت متداول تر آن را پیچش ترانهاده میخوانند[۸]، چون که وقتی پیچش را به صورت ماتریسی بنویسیم، این عملیات یک ضرب است با یک ماتریس ترانهاده وزن. همانطور مانند پیچش معمولی، پارامترهای لایه گذاری و غیره نیز برای این عملیات تعریف میشود، با این تفاوت که قدم گذاری این جا مشخص میکند که چقدر افزایش نمونه[5] داشته باشیم به جای کاهش نمونه[6].

[1] unpooling
[2] Interpolation
[3] Backward Convolution
[4] Fractional stride
[5] Upsampling
[6] Downsampling



شکل ۱۰.۲: پیچش ترانهاده میتواند برای افزایش اندازه یک تصویر استفاده شود. قبل از انجام عمل پیچش، $s-1$ ردیف و ستون اضافه از صفرها بین ورودی‌ها اضافه میکنیم، که در آن به s قدم زدن افزایش نمونه‌ها میگویند[۳۰].

۶.۲ شبکه‌های عصبی تکرار شونده و بازگشتی

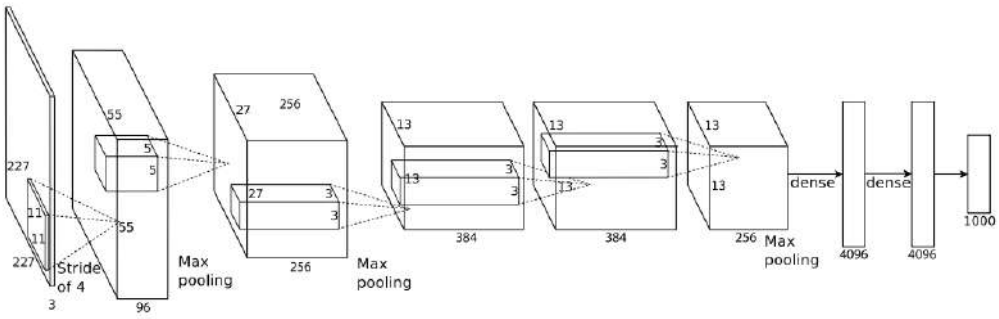
شبکه‌های عصبی تکرار شونده[1] (RNN)، یک خانواده از شبکه‌های عصبی هستند برای پردازش داده‌های دنباله ای[2]. مانند CNNها، RNNها نیز شخصی سازی شده اند برای پردازش داده‌های دنباله ای به صورت $x^{(1)}, ..., x^{(\tau)}$. همانطور که CNNها میتوانند تعداد زیادی عکس را پردازش کنند که شبکه‌های عصبی معمولی نمیتوانستند، RNNها نیز میتواند داده‌های دنباله ای را به گونه ای سریع پردازش کنند که در شبکه‌های دیگر امکان پذیر نیست. بسیاری از RNNها میتوانند حتی داده‌هایی با اندازه دنباله متغیر را نیز پردازش کنند[۱۱].

برای تبدیل شبکه‌های لایه ای به شبکه‌های تکرار شونده از یکی از قدیمی ترین ایده‌های شبکه‌های عصبی استفاده میکنیم: وزن‌های مشترک در بخش‌های مختلف مدل. اشتراک وزن‌ها این خاصیت را دارد که اندازه مدل را به شدت کاهش میدهد و به تعمیم بخشی مدل کمک میکند. چنین خاصیتی مهم میشود مخصوصا در مواقعی که ممکن است اطلاعاتی در جاهای مختلف از داده ظاهر شود و ما با کمک وزن‌های مشترک میتوانیم آن را هر کجای داده که باشد تشخیص دهیم.

برای اطلاعات بیشتر از سطح این گذارش به کتاب Graves[۱۲] مراجعه کنید.

[1] Recurrent Neural Networks

[2] Sequential Data



شکل ۱۱.۲: ساختار شبکه AlexNet. این شبکه شامل تعدادی لایهٔ پیچشی، ادغامی و در نهایت کاملاً متصل است که در انتها عمل دسته بندی را انجام میدهد.

۷.۲ باز کردن گراف‌های محاسباتی

یک گراف محاسباتی روشی است برای فرمال کردن ساختار مجموعه ای از محاسبات. در این بخش ایدهٔ باز کردن[۱] یک محاسبات بازگشتی[۲] یا تکرارشونده[۳] را بررسی میکنیم، که یک ساختار تکرار شونده دارد که معمولاً منتاظر با یک سری اتفاقات است. باز کردن این گراف باعث به اشتراک گذاشتن وزن‌ها در سطح یک شبکه عمیق میشود.

برای مثال معادله کلاسیک برای سیستم دینامیک زیر را در نظر بگیرید.

$$s^{(t)} = f(s^{(t-1)}; \theta), \tag{۹.۲}$$

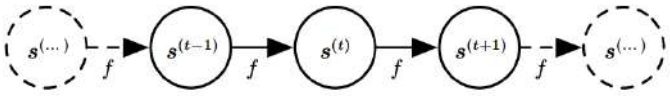
که به s_t حالت سیستم میگویند.

برای تعداد متناهی تا قدم زمانی[۴] τ، گراف میتواند باز شود به این صورت که تعریف را $\tau - 1$ بار اعمال میکنیم. برای مثال اگر این معادله را ۳ بار باز کنیم بدست خواهیم آورد

[1] Unfolding
[2] Recursive
[3] Recurrent
[4] Time Step

$$s^{(3)} = f(s^{(2)}; \theta) \quad (۱۰.۲)$$

$$= f(f(s^{(1)}; \theta); \theta) \quad (۱۱.۲)$$



شکل ۱۲.۲: ۱۱.۲ گراف باز شدهٔ معادلهٔ [۱۱]

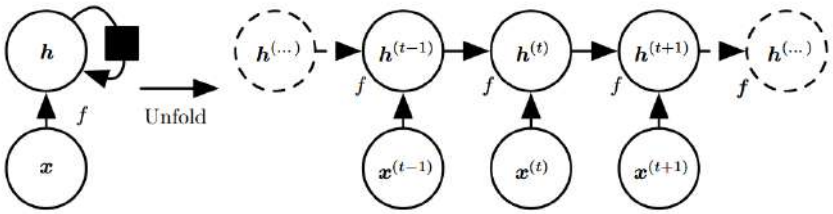
باز کردن این معادله توسط تکرار کردن این تعریف بارها و بارها به ما عبارتی را میدهد که خالی از خاصیت بازگشتی است. چنین عبارتی را میتوان حالا به صورت یک گراف جهت دار بدون دور متداول نشان داد. گراف محاسباتی معادله ۹.۲ را در شکل ۱۲.۲ ببینید.

بسیاری از شبکه‌های تکرار شونده از معادله ای شبیه معادلهٔ زیر برای حساب کردن مقدار واحد پنهان خود استفاده میکنند.

$$h^{(t)} = f(h^{(t-1)}, x^{(t)}; \theta) \quad (۱۲.۲)$$

که $h^{(t)}$ مقدار واحد پنهان در زمان t است و $x^{(t)}$ نیز ورودی زمان t است. در شکل ۱۳.۲ میتوانید باز شدهٔ این معادله را ببینید.

زمانی که RNN استفاده میشود برای پیشبینی اتفاقی در آینده با استفاده از اطلاعات گذشته، شبکه آموزش می بیند که از $h^{(t)}$ به عنوان خلاصهٔ از دست روندهٔ خصوصیات دنبالهٔ ورودیمان استفاده کند از اول تا الآن. این خلاصه حتما از دست رونده است چون یک دنباله به طول دلخواه را به یک بردار طول ثابت می نگارد. با توجه به سوالی که شبکه دارد به آن جواب میدهد ممکن است اطلاعاتی را بیشتر از دیگری نگه دارد یا حذف کند. در اکثر مسائل نیز نیازی نیست تا تمامی اطلاعات گذشته را برای پیدا کردن جوابی در آینده نگه داشت مانند کاربردهای زبان طبیعی، ولی در بعضی از کاربردها مانند تشخیص اعمال انسانی نیاز است که اطلاعات زیادی را در $h^{(t)}$ ذخیره کنیم که تقریبا غیر ممکن است. این مسئله باعث میشود دقت این نوع شبکه‌ها در دنباله‌های طولانی پایین بیاید.



شکل ۱۳.۲: یک شبکه تکرار شونده بدون خروجی. در سمت چپ به شکل بازگشتی مشاهده میکنید و. در سمت راست به صورت باز شده[۱۱].

میتوانیم تکرارهای باز شدهٔ تابعمان را به صورت زیر با یک تابع $g^{(t)}$ نمایش دهیم

$$h^{(t)} = g^{(t)}(x^{(t)}, x^{(t-1)}, ..., x^{(1)} \quad (۱۳.۲)$$

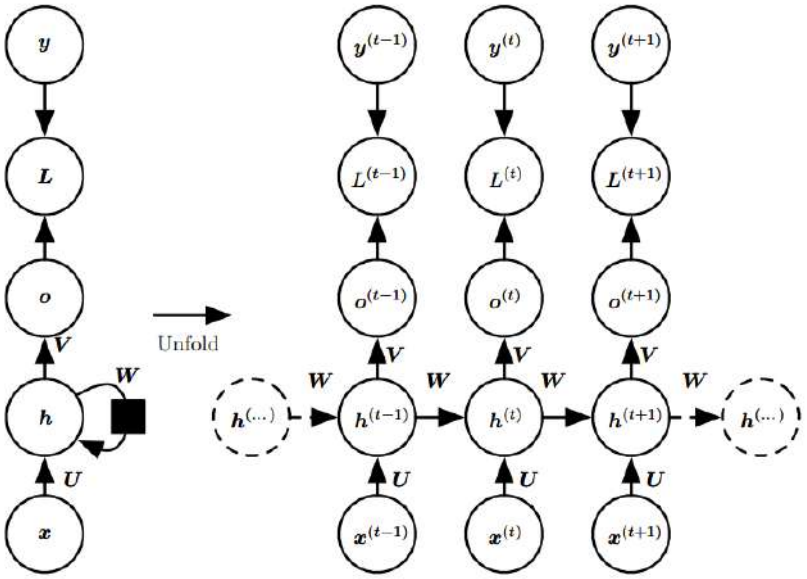
$$= f(h^{(t-1)}, x^{(t)}; \theta). \quad (۱۴.۲)$$

تابع $g^{(t)}$ تمامی $x^{(i)}$ها را تا کنون به عنوان ورودی میگیرد و حالت الان را خروجی میدهد، ولی با باز کردن تابع اصلی میتوانیم $g^{(t)}$ را خلاصه سازی کنیم به تعدادی تکرار از تابع f که همیشه ورودی مشخصی دارد. در نتیجه در روش باز کردن هم ورودی تابعمان همیشه ثابت است هم برای هر قدم زمانی یک تابع ثابتی را یاد میگیریم و پارامترها بین زمان‌های مختلف مشترک است. در نتیجه نیازی به یادگیری g به طور مستقیم نیست.

۸.۲ شبکه عصبی تکرار شونده

با توجه به اطلاعات بخش قبل در مورد باز کردن و وزن‌های مشترک، حال میتوانیم انواع مختلفی از شبکه‌های تکرار شونده بسازیم. برخی از مثال‌های شبکه‌ها را در ادامه میبینیم:

- شبکه‌های تکرار شونده ای که در هر پلهٔ زمانی یک خروجی تولید میکنند و اتصال‌های تکرار شونده بین واحدهای پنهان دارند. شکل ۱۴.۲
- شبکه‌های تکرار شونده ای که در هر پلهٔ زمانی یک خروجی تولید میکنند و اتصال‌های تکرار شونده دارند که فقط از خروجی یک پلهٔ زمانی به واحد پنهان پلهٔ زمانی بعدی است.



شکل ۱۴.۲: یک شبکه تکرار شونده همراه با خروجی که حالت‌های پنهان در هر پلهٔ زمانی به حالت پنهان در پلهٔ بعدی وصل است و در حل پله یک خروجی دارد[۱۱]

- شبکه‌های تکرار شونده با اتصال‌های تکرار شونده بین واحدهای پنهان که اول کل دنبالهٔ ورودی را میخوانند و سپس یک خروجی تولید میکنند.

شبکه عصبی تکرار شونده شکل ۱۴.۲ جهانی هستند، به این معنا که هر ماشین تورینگی میتواند توسط چنین شبکه ای با اندازهٔ متناهی محاسبه شود و معمولا وقتی از RNN صحبت میشود منظورشان این مدل از شبکه است.

حال برای شبکهٔ شکل ۱۴.۲ معادلات انتشار[1] مینویسیم. شکل مشخص نمیکند از چه تابع فعال سازی ای استفاده میشود. در اینجا فرض میکنیم که از تابع تانژانت‌هایپربولیک استفاده میکنیم. همچنین شکل مشخص نمیکند که خروجی و تابع ضرر چه شکلی دارند. در اینجا فرض میکنیم که خروجی گسسته است، چنان که RNN برای حدس زدن کلمه یا کاراکترها استفاده شود. یک راه طبیعی برای نشان دادن متغیرهای گسسته این است که در نظر بگیریم خروجی o لگاریتم احتمال نرمال نشدهٔ هر مقدار ممکن از متغیر گسسته است. در نتیجه میتوانیم عملیات softmax

[1] Propogation

را بعد از محاسبات اعمال کنیم تا به بردار $\hat{y}$ دست یابیم که مقدار نرمال شدهٔ احتمالات بر روی خروجی‌هاست. انتشار مستقیم با تصریح حالت اولیهٔ h^0 شروع میشود. سپس برای هر پلهٔ زمانی از $t = 1$ تا $t = \tau$، این معادلات تغییر زیر را اعمال میکنیم:

$$a^{(t)} = b + Wh^{(t-1)} + Ux^{(t)}, h^{(t)} \quad = \tanh(a^{(t)}), o^{(t)} = c + Vh^{(t)}, \qquad (۱۵.۲)$$

که پارامترهای آن بردارهای تعصب bو c هستند همراه با ماتریس‌های وزن U, V و W، به ترتیب برای ورودی به پنهان، پنهان به خروجی و پنهان به پنهان. این یک مثال است از یک شبکه تکرار شونده که دنباله ورودی را به یک دنباله خروجی به طول مساوی نگاشت میکند. ضرر کلی برای یک دنبالهٔ x از ورودی و دنبالهٔ y از خروجی میشود مجموعه ضررها در طول تمام پله‌های زمانی. برای مثال اگر $L^{(T)}$ منفی لگاریتم احتمال $y^{(t)}$ با داشتن $x^{(1)}, ..., x^{(t)}$ باشد، در نتیجه

$$L(\{x^{(1)}, ..., x^{(t)}\}, \{y^{(1)}, ..., y^{(t)}\}) \qquad (۱۶.۲)$$

$$= \sum_t L^{(t)} \qquad (۱۷.۲)$$

$$= -\sum_t \log p_{\text{model}}(y^{(t)} | \{x^{(1)}, ..., x^{(t)}\}), \qquad (۱۸.۲)$$

$$(۱۹.۲)$$

که $p_{\text{model}}(y^{(t)} | \{x^{(1)}, ..., x^{(t)}\})$ داده میشود با خواندن مقدار $y^{(t)}$ از بردار خروجی مدل $\hat{y^{(t)}}$ به دست میاید. حساب کردن گرادیان این تابع ضرر نسبت به پارامترها کار هزینه بری است. حساب کردن گرادیان عبارت است از انجام دادن یک انتشار مستقیم از چپ به راست در شکل باز شدهٔ ۱۴.۲، و سپس پس انتشار از راست به چپ در همان شکل. اوردر زمانی اجرای این کار $O(\tau)$ است و نمیتوان با موازی سازی آن را کاهش داد چون کارهایی که انجام میدهیم باطن دنباله ای دارند. هر پلهٔ زمانی فقط زمانی میتواند محاسبه شود که قبلی حساب شده باشد. به الگوریتم پس انتشار که بر روی گراف باز شده اجرا میشود پس انتشار در طول زمان[1] (BPTT) میگویند و در قسمت‌های آینده توضیح کامل تر میدهیم.

[1] Back-Propogation Through Time

۱.۸.۲ محاسبه گرادیان در شبکه عصبی تکرار شونده

محاسبهٔ گرادیان برای یک شبکه عصبی تکرار شونده کار مشخصی است. صرفاً کافی است الگوریتم پس انتشار کلی را روی گراف محاسباتی باز شده اعمال کنیم. هیچ الگوریتم خاصی نیاز نیست. سپس گرادیان‌های بدست آمده از طریق پس انتشار میتواند توسط هر تکنیک کلی که از گرادیان استفاده میکند به کار گرفته شود برای تربیت یک RNN.

۹.۲ حافظهٔ بلند کوتاه‌ـمدت

یکی از اولین و موفق ترین تکنیک‌ها برای حل مشکل کاهش پرادیان‌ها، به شکل مدل حافظهٔ بلند کوتاه‌ـمدت[۱] LSTM بود که توسط Hochreiter و Schmidhuber در سال ۱۹۹۷ معرفی شد[۱۴]. LSTMها شبیه به شبکه‌های عصبی بازگشتی استاندارد هستند، با این تفاوت که در اینجا هر گره[۲] بازگشتی عادی با یک سلول حافظه[۳] جایگزین شده است. هر سلول حافظه دارای یک وضعیت داخلی[۴] است، به این معنا که، یک گره با یک یال خود‌ـمتصل به وزن ۱ است، که تضمین میکند گرادیان میتواند از چندین مرحلهٔ زمانی بدون ناپدید شدن[۵] یا انفجار[۶] عبور کند.

۱.۹.۲ سلول حافظهٔ دریچه دار

هر سلول حافظه مجهز به یک وضعیت داخلی و تعدادی دریچهٔ ضرب کننده است که تعیین میکند که (الف) آیا ورودی داده شده باید روی وضعیت داخلی تاثیر بگذارد را خیر (دریچهٔ ورودی)، (ب) آیا وضعیت داخلی باید صفر و خالی شود (دریچهٔ فراموشی)، و (ج) وضعیت داخلی یک نورون دلخواه باید بتواند روی خروجی سلول تاثیر بگذارد یا خیر (دریچهٔ خروجی).

[1] Long Short-Term Memory
[2] Node
[3] Memory Cell
[4] Internal State
[5] Vanishin
[6] Exploding

۲.۹.۲ وضعیت پنهان دریچه دار

تفاوت اصلی بین RNNها و LSTMها این است که دومی دریچه دار کردن وضعیت پنهان را پشتیبانی میکند. این به این معناست که ما میتوانیم مکانیسم مخصوص داشته باشیم برای زمانی که وضعیت پنهان باید بروزرسانی شود و همچنین چه زمانی باید تنظیم مجدد شود. این مکانیسم‌ها آموزش دیده میشوند و به سوالات بالا پاسخ میدهند.

۳.۹.۲ دریچه ورودی، فراموشی، و خروجی

داده‌هایی که به دریچه‌های LSTM داده میشود، ورودی در گام زمانی فعلی است و وضعیت پنهان در گام زمانی قبلی همانطور که در شکل ۱۵.۲ میبینید. سه لایهٔ کاملاً متصل با توابع فعال سازی سیگموید مقادیر دریچهٔ ورودی، خروجی و فراموشی را محاسبه میکنند. در نتیجه به دلیل فعالسازی سیگموید، تمامی مقادیر هر سه دریچه در بازهٔ $(0, 1)$ قرار دارند. علاوه بر این، به یک گرهٔ ورودی نیاز داریم، که معمولاً با یک تابع فعال سازی tanh محاسبه میشود. همانطور که از اسمشان پیداست، دریچهٔ ورودی مشخص میکند چقدر از مقدار گره ورودی باید به وضعیت داخلی سلول حافظه مان اضافه شود. همچنین، دریچهٔ فراموشی مشخص میکند که آیا میخواهیم مقدار فعلی حافظه را نگه داریم یا بیرون بریزیم. و در نهایت دریچهٔ خروجی مشخص میکند که آیای سلول حافظه باید خروجی را در گام زمانی فعلی تحت تاثیر قرار بدهد یا خیر.

برای محاسبهٔ دریچه‌ها داریم:

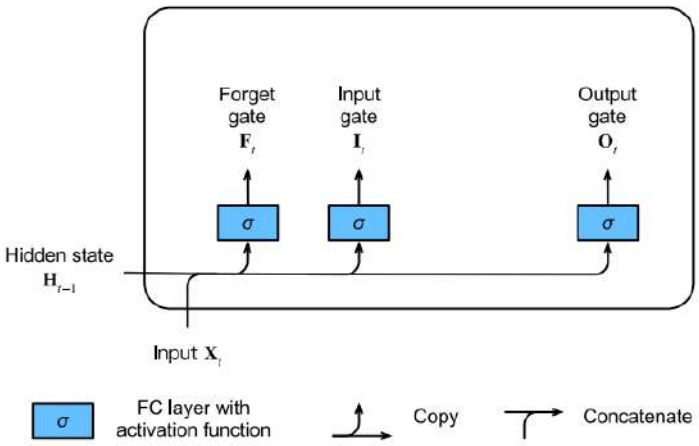
$$I_t = \sigma(X_t W_{xi} + H_{t-1} W_{hi} + b_i), \tag{۲۰.۲}$$

$$F_t = \sigma(X_t W_{xF} + H_{t-1} W_{hF} + b_F), \tag{۲۱.۲}$$

$$O_t = \sigma(X_t W_{xo} + H_{t-1} W_{ho} + b_o), \tag{۲۲.۲}$$

۴.۹.۲ گره ورودی

در ادامه، سلول حافظه را طراحی میکنیم. از آنجایی که ما عمل دریچه‌های مختلف را هنوز مشخص نکردیم، اول گره ورودی $\tilde{C}_t$ را معرفی میکنیم. حساب کردن آن شبیه سه دریچهٔ توصیف شده در بالا است، ولی با استفاده از تابع tanh که مقداری بین -۱ و ۱ به ما میدهد به عنوان تابع فعال



شکل ۱۵.۲: محاسبهٔ دریچهٔ ورودی، خروجی و فراموشی [۳۲]

سازی. در نتیجه در گام زمانی t خواهیم داشت:

$$\tilde{C}_t = \tanh(X_t W_{xc} + H_{t-1} W_{hc} + b_c). \qquad (۲۳.۲)$$

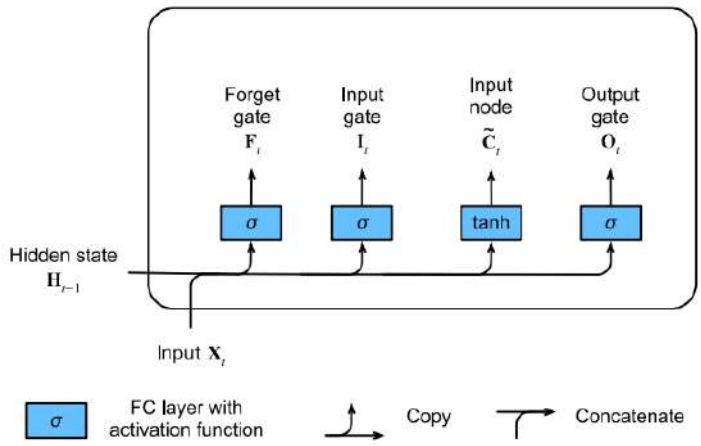
۵.۹.۲ وضعیت داخلی سلول حافظه

در LSTMها، دریچهٔ ورودی I_t حکم میکند که چقدر از دادهٔ جدید را از طریق $\tilde{C}_t$ به کار ببریم و دریچهٔ فراموشی F_t مشخص میکند که چه مقدار از وضعیت داخلی سلول قدیمی را نگه داریم. با استفاده از ضرب‌هادامارد (ضرب عضو به عضو) میرسیم به معادلهٔ زیر:

$$C_t = F_t \odot C_{t-1} + I_t \odot \tilde{C}_t. \qquad (۲۴.۲)$$

۶.۹.۲ وضعیت پنهانی

در نهایت، نیاز داریم تعریف کنیم که چگونه خروجی سلول حافظه H_t را محاسبه کنیم. در اینجا دریچهٔ خروجی به کار میاید. در LSTMها، ما اول tanh را بر روی وضعیت داخلی سلول اعمال



شکل ۱۶.۲: محاسبهٔ گره ورودی [۳۲]

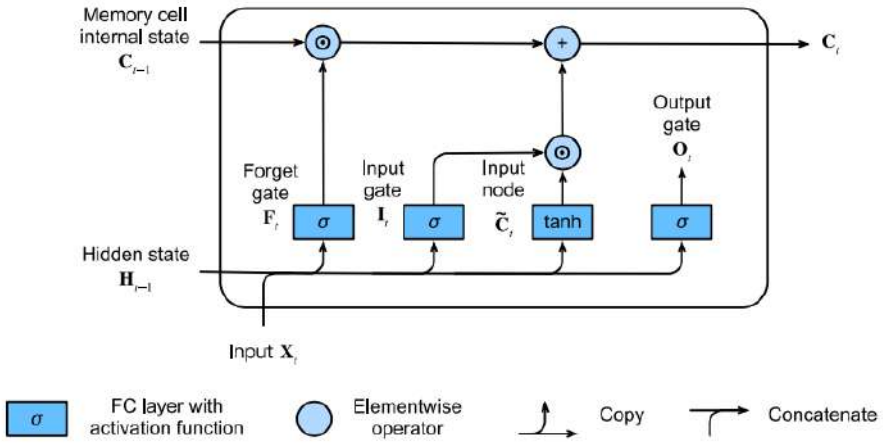
میکنیم و سپس یک ضرب عضو به عضو دیگر اعمال میکنیم، این دفعه همراه با دریچهٔ خروجی. این تضمین میکند که مقدار H_t همیشه در بازهٔ -۱ تا ۱ خواهد بود:

$$H_t = O_t \odot \tanh(C_t). \tag{۲۵.۲}$$

۱۰.۲ واحد تکرار شونده دریچه دار

همچنان که RNNها و LSTMها محبوب تر شدند، مقالهها شروع کردند به پیدا کردن ساختاری ساده تر که نتایجی مشابه داشته باشد، با حفظ ایدهٔ اصلی داشتن حالت داخلی و مکانیزم ضرب دریچه ای. واحد تکرار شونده دریچه دار[1] (GRU) [۴] یک نسخهٔ ساده شده از واحد حافظهٔ LSTM بود که نتایج قابل مقایسه به دست میاورد ولی با سرعت محاسبهٔ بیشتر[۵].

[1] Gated Recurrent Unit



شکل ۱۷.۲: محاسبهٔ وضعیت داخلی سلول حافظه [۳۲]

۱.۱۰.۲ دریچهٔ تنظیم مجدد و بروزرسانی

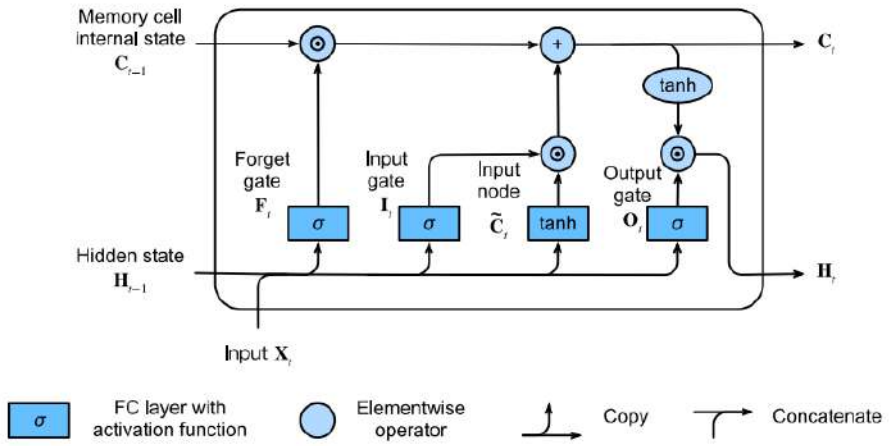
در اینجا، ۳ دریچهٔ LSTM با دو دریچه جایگزین شده است: دریچهٔ تنظیم مجدد[1] و دریچهٔ بروزرسانی[2]. مانند LSTMها، به این دریچه‌ها تابع سیگموید نسبت میدهیم، و مجبورشان میکنیم تا مقدارشان در بازهٔ $(0, 1)$ باشد. همانطور که از اسمشان پیداست، دریچهٔ تنظیم مجدد کنترل میکند که چقدر از حال قبلی را همچنان میخواهیم به یار داشته باشیم. دوباره به همین شکل، یک دریچهٔ بروزرسانی به ما این امکان را میدهد که کنترل کنیم چه مقدار از حالت جدید، کپی از حالت قبلی است. شکل ۱۹.۲ نشان میدهد ورودی‌ها را برای هر دو دریچهٔ بروزرسانی و تنظیم مجدد، با توجه به داشتن مقدار ورودی پلهٔ زمانی حالا و مقدار حالت پنهانی پلهٔ قبلی. خروجی این دو دریچه توسط دو لایهٔ کاملا متصل با یک تابع فعال سازی سیگموید به دست میاید.

برای محاسبهٔ مقادیر دریچه‌ها در زمان t داریم:

$$R_t = \sigma(X_t W_{xr} + H_{t-1} W_{hr} + b_r), \tag{۲۶.۲}$$

$$Z_t = \sigma(X_t W_{xz} + H_{t-1} W_{hz} + b_z), \tag{۲۷.۲}$$

[1] Reset gate
[2] Update Gate



شکل ۱۸.۲: محاسبهٔ وضعیت پنهانی [۳۲]

که در آن W_{xr}, W_{xz}, b_r, b_z به ترتیب وزن‌ها و مقادیر تعصب‌ها هستند و X_t ورودی است.

۲.۱۰.۲ کاندیدای حالت پنهان

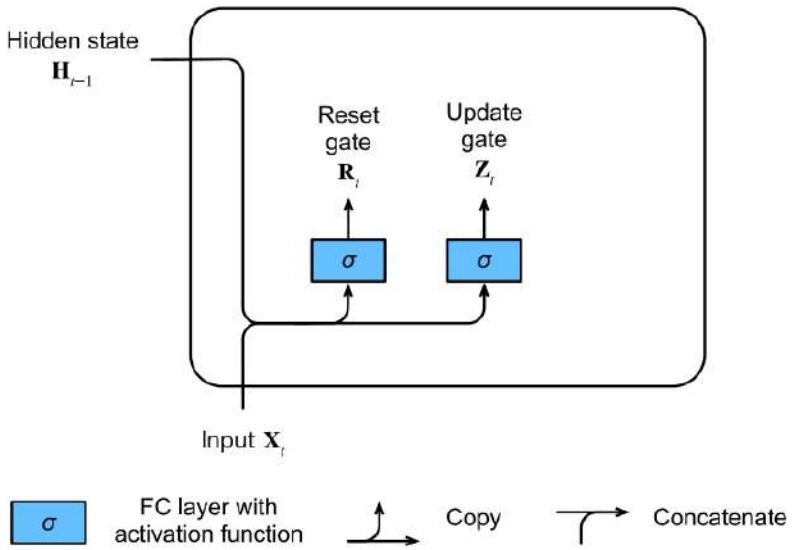
در ادامه، دریچهٔ تنظیم مجدد R_t را در فرمول بروزرسانی معمولی جاسازی میکنیم و به کاندیدای حالت جدید $\tilde{H}_t$ در پلهٔ زمانی t میرسیم:

$$\tilde{H}_t = \tanh(X_t W_{xh} + (R_t \odot H_{t-1}) W_{hh} + b_h). \qquad (۲۸.۲)$$

نتیجه به دست آمده یه کاندیدا است چون همچنان باید تاثیر عمل دریچهٔ بروزرسانی را نیز اعمال کنیم.

۳.۱۰.۲ حالت پنهان

در نهایت، ما باید تاثیر دریچهٔ Z_t را اعمال کنیم. این مشخص میکند که تا کجا و چقد میخواهیم حالت پنهان جدید شبیه حالت پنهان قدیمی باشد یا شبیه کاندیدای حالت پنهان جدید باشد. دریچهٔ بروزرسانی برای این کار استفاده میشود، به این صورت که ترکیب عنصر به عنصر محدب H_{t-1} و



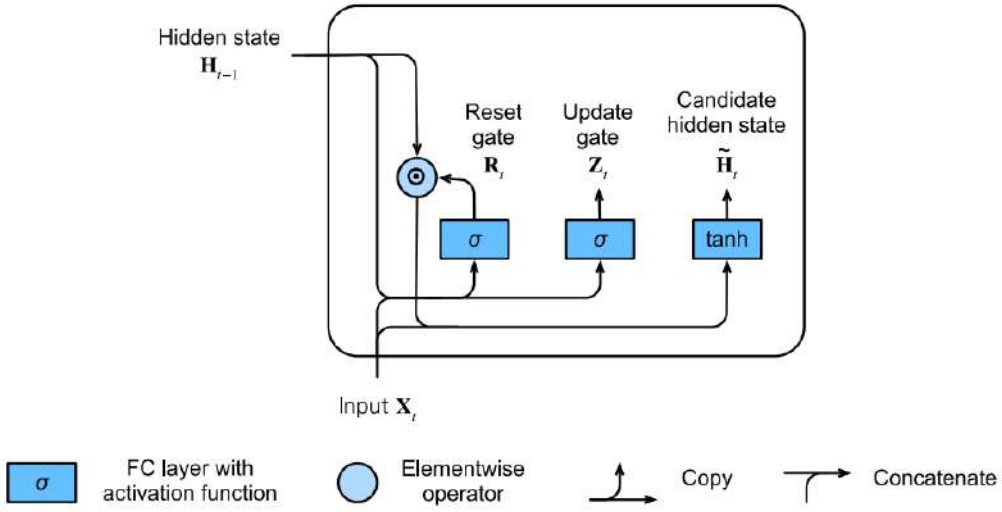
شکل ۱۹.۲: محاسبه دریچهٔ بروزرسانی و تنظیم مجدد [۳۲]

$\tilde{H}_t$ را محاسبه میکند. این نتیجه میدهد آخرین معادلهٔ بروزرسانی GRU را:

$$H_t = Z_t \odot H_{t-1} + (1 - Z_t) \odot \tilde{H}_t. \tag{۲۹.۲}$$

۱۱.۲ شبکه عصبی تکرار شونده عمیق

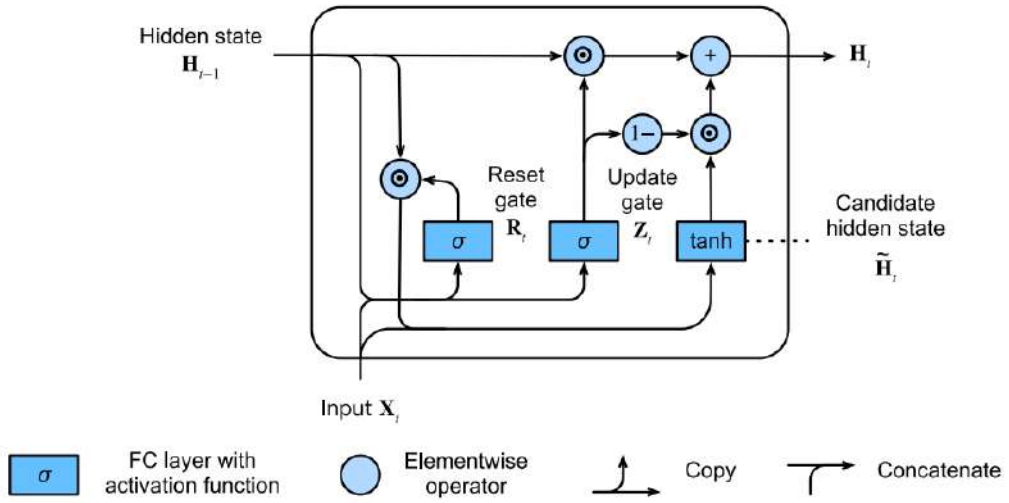
تا الان ما بر روی تعریف شبکه‌های متشکل از یک ورودی متوالی یک تک لایه پنهان RNN و یک لایه خروجی تمرکز کرده بودیم. با وجود فقط یک لایه مخفی بین ورودی در هر گام زمانی و خروجی متناظر، از نظریه‌ای، شبکه‌های از پیش تعیین شده درونی عمیق هستند. ورودی‌ها در گام زمانی اول می‌توانند بر روی خروجی‌ها در گام زمانی نهایی T (که اغلب صدها یا هزاران گام بعد است) تأثیر بگذارند. این ورودی‌ها از طریق T بار اجرای لایه بازگشتی عبور می‌کنند تا به خروجی نهایی برسند. با این حال، ما اغلب تمایل داریم تا قابلیت بیان روابط پیچیده بین ورودی‌ها در گام زمانی داده شده و خروجی‌ها در همان گام زمانی را حفظ کنیم. به همین دلیل اغلب هایRNN را میسازیم که نه تنها در جهت زمان ، بلکه در جهت ورودی به خروجی عمق داشته باشند. این دقیقاً



شکل ۲۰.۲: محاسبهٔ کاندیدای حالت پنهان [۳۲]

همان مفهوم عمق است که قبلاً در توسعه هاMLP و هاCNN مشاهده کرده‌ایم.
روش استاندارد برای ساختن این نوع شبکه‌های عصبی بازگشتی عمیق به طرحی بسیار ساده توجه می‌کند: ما هاRNN را در یکدیگر قرار می‌دهیم. با داشتن یک دنباله با طول T، RNN اول یک دنباله خروجی نیز با طول T تولید می‌کند. این خروجی‌ها در نتیجه ورودی‌های لایه بعدی از RNN قرار می‌گیرند. در این قسمت کوتاه، این الگوی طراحی را نشان می‌دهیم و یک مثال ساده برای نحوه برنامه‌نویسی این نوع شبکه‌های عصبی را ارائه می‌دهیم. در زیر، در شکل ، یک RNN عمیق با L لایه مخفی را نشان می‌دهیم. هر حالت مخفی بر روی ورودی توالی‌ای عمل می‌کند و یک خروجی توالی‌ای تولید می‌کند. علاوه بر این هر سلول RNN (جعبه سفید در شکل در هر مرحله زمانی، به همان مقدار لایه مشابه در مرحله قبلی زمانی و مقدار لایه قبلی در همان مرحله زمانی وابسته است.
فرض کنید ما یک گروه کوچک ورودی $X_t \in \mathbb{R}^{nxd}$ (تعداد نمونه‌ها : n و تعداد ورودی‌ها در هر نمونه : d) در زمان .t در همان مرحله زمانی، قرار دهید: حالت نهان l^{th} از لایه‌های پنهان $(l = 1, ..., L)$ بصورت :

$$H_t^{(l)} = \phi_l(H_t^{(l-1)} W_{xh}^{(l)} + H_{t-1}^{(l)} W_{hh}^{(l)} + b_h^{(l)}) \quad (10.3.1)$$



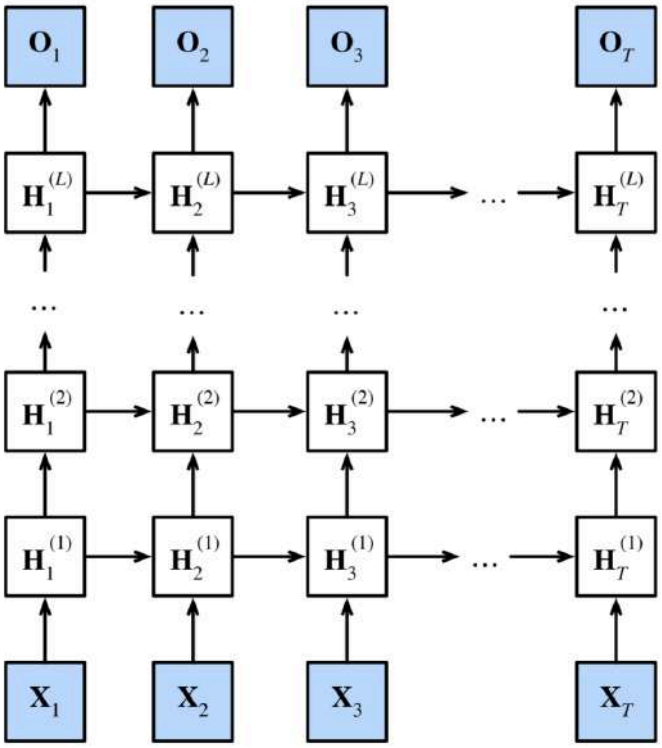
شکل ۲۱.۲: محاسبهٔ حالت پنهان [۳۲]

به طوری که وزن‌های $W_{xh} \in \mathbb{R}^{hxh}$ و $W_{hh} \in \mathbb{R}^{hxh}$، با هم با بایاس $b_h^{(l)} \in \mathbb{R}^{1xh}$ پارامترهای مدل L امین لایه پنهان اند. در آخر محاسبه لایه خروجی فقط بر حسب لایه مخفی از آخرین لایه پنهان L ام است.

$$O_t = H_t^{(L)} tWhq + bq \quad (10.3.2)$$

جایی که وزن $W_{hq} \in \mathbb{R}^{hxq}$ و بایاس $b_q \in \mathbb{R}^{1xq}$ پارامترهای مدل لایه خروجی اند.

همانطور که در MLP ، تعداد لایه‌های پنهان L و تعداد واحدهای پنهان h از جمله‌هایپرپارامترهایی هستند که می‌توانیم آن‌ها را تنظیم کنیم. برای لایه RNN با عرض (h) در بازه (۶۴ ، ۲۰۵۶) قرار دارند و عمق‌های رایج (L) در بازه (۸، ۱) قرار دارند. علاوه بر این، می‌توانیم به راحتی یک RNN گیت شده عمیق را با محاسبات LSTM یا GRU ایجاد کنیم.



شکل ۲۲.۲: ساختار یک RNN عمیق[۳۲]

۱۲.۲ ساختار توجه

از زمانی که شبکه‌های عصبی معرفی شدند (دهۀ ۱۹۸۰) تا دهۀ ۲۰۱۰، ایده‌های اصلی و پرطرفدار یادگیری ماشین از جمله هاCNN و هاRNN در این ۳۰ سال تغییر بسیاری نکردند و از آن همان ایده‌های ۳۰ سال پیش بهره میبرند (با تغییرات کوچک) و پیشرفت دیده شده در این زمینه بیشتر تحت تاثیر پیشرفت سخت افزاری و افزایش داده بوده است.

در سال‌های اخیر ساختاری کاملاً جدید معرفی شد که در این سال‌ها زمینۀ NLP را در سلطۀ خود دراورد و آن ساختار توجه[1] [۳۱] و مبدل‌ها[2] بود. در حال حاضر روش اصلی برای حل مسائل NLP این است که یک مدل بر پایۀ توجه از قبل آموزش دیدۀ بزرگ را بگیرند و لایه یا لایه‌های

[1] Attention
[2] Transformers

آخر آن را طبق نیاز دوباره آموزش بدهند. این ساختارها همچنین دارند کم کم به مدل‌های پیش فرض در زمینهٔ بینایی ماشین و تشخیص صدا نیز تبدیل میشوند.

در این بخش به ساختار توجه و مدل مبدل می پردازیم.

۱۳.۲ پرسش، کلید و مقدار

تا کنون تمامی شبکه‌هایی که بررسی کردیم به این وابسته بودند که ورودی آن‌ها خوب تعریف شده باشد. حتی در هاRNN نیز اندازهٔ ورودی در هر پلهٔ زمانی یکسان است. این نکته مشکلات زیادی ایجاد میکند مخصوصاً زمانی که ورودی ما به طور طبیعی اندازهٔ متغیر دارد با مقدار اطلاعات متفاوت.

این مسئله را با پایگاه داده مقایسه کنید. در ساده ترین شکل آن، پایگاه داده مجموعه ای از کلیدها (k) و مقادیر (v) است و برای گشتن در این پایگاه داده از پرسش (q) استفاده میکنیم. این تفکر باعث معرفی ساختاری جدید به نام توجه [۲] شد.

فرض کنید یک پایگاه داده داریم

$$D = (k_1, v_1), ..., (k_m, v_m), \tag{۳۰.۲}$$

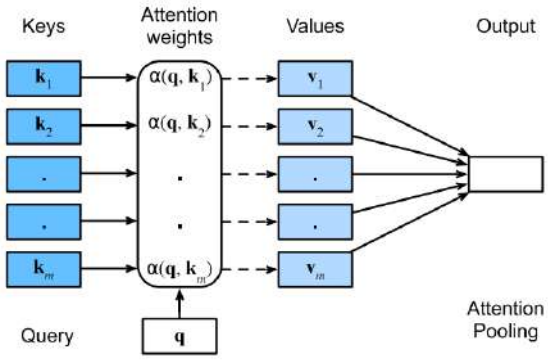
که m تا دو تایی مرتب از کلیدها و مقادیر دارد. در ادامه، یک پرسش را با q نشان میدهیم. در نتیجه میتوانیم توجه روی D را به صورت

$$\text{Attention}(q, D) = \sum_{i=1}^{m} \alpha(q, k_i) v_i, \tag{۳۱.۲}$$

تعریف کنیم که در آن $\alpha(q, k_i)$ وزن‌های اسکالر توجه هستند. به این عملیات معمولاً ادغام توجه[1] میگویند. نام توجه از آنجا میاید که این عملیات به جمله‌هایی که α شان مقادیر قابل توجهی دارند توجه بیشتری میکند. در نتیجه، این عمل یک ترکیب خطی از مقادیر موجود در پایگاه داده را به ما میدهد.

در یادگیری عمیق طوری تابع α طراحی میشود که ترکیب محدبی به ما بدهد. به این معنا که

[1] Attention Pooling



شکل ۲۳.۲: ساختار توجه یک مجموع خطی از مقادیر به دست میاورد با استفاده از ادغام توجه که وزن‌ها از شباهت بین پرسش و کلیدها نشأت میگیرد. [۳۲]

$\alpha(q, k_i) \geq 0$ و $\sum_i \alpha(q, k_i) = 1$ برای هر i باشد. برای اینکه مطمئن شویم که جمع وزن‌ها برابر 1 است و تمامی وزن‌ها بزرگتر مساوی 1 هستند، تابع دلخواه $a(q, k)$ را انتخاب میکنیم و از softmax استفاده میکنیم

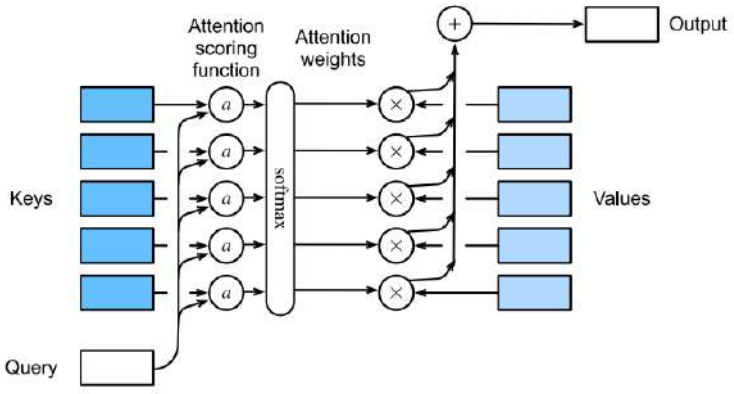
$$\alpha(q, k_i) = \frac{\exp(a(q, k_i))}{\sum_j \exp(a(q, k_j))}. \tag{۳۲.۲}$$

این عملیات مشتق پذیر است و هیچ گاه گرادیان آن ناپدید نمیشود که این برای یک مدل یادگیری عمیق مناسب است.

۱۴.۲ توابع امتیاز دهی توجه

توابع هسته ای[1] محاسبهٔ هزینه بری دارند نسبت به ضرب‌های داخلی. در نتیجه از توابع امتیاز دهی توجه[2] (a) که محاسبهٔ ساده تری دارند استفاده میکنیم.

[1] Kernel Functions
[2] Attention Scoring Functions



شکل ۲۴.۲: محاسبهٔ خروجی ادغام توجه که به صورت جمع وزن دار از مقادیر انجام میشود که وزن‌ها به کمک تابع امتیاز دهی و عملیات softmax محاسبه میشوند[۳۲].

تابع هسته ای گاوسی برای محاسبهٔ توجه را در نظر بگیرید:

$$a(q, k_i) = -\frac{1}{2}\|q - k_i\|^2 \tag{۳۳.۲}$$

$$= q^T k_i - \frac{1}{2}\|k_i\|^2 - \frac{1}{2}\|q\|^2. \tag{۳۴.۲}$$

اول دقت کنید که آخرین جمله فقط به q بستگی دارد و در نتیجه برای تمام زوج‌های (q, k_i) یکسان است. نرمالایز کردن وزن‌ها به روش گفته شده، به کل این جمله را حذف میکند. دوماً باید بزرگی آرگومان‌ها در تابع نمایی را تحت کنترل داشته باشیم. فرض کنید که تمام المان‌های پرسش q و کلید k_i مستقل و متغیرهای تصادفی ای هستند با میانگین صفر و واریانس یک. ضرب داخلی بین بردارها میانگین صفر و واریانس d خواهد داشت. برای اینکه همچنان واریانس یک بماند، ز تابع امتیاز دهی ضرب داخلی تنظیم شده استفاده میکنیم. به این معنا که ضرب داخلی را با تقسیم بر $\sqrt{d}$ تقسیم میکنیم تا مقیاس را تغییر دهیم. در نتیجه میرسیم به تابع توجه استفاده شده در مبدل‌ها [۳۱]:

$$a(q, k_i) = a^T k_i / \sqrt{d}. \tag{۳۵.۲}$$

به این نکته دقت کنید که همچنان وزن‌ها نیاز به نرمال سازی دارند و در نتیجه باید تابع softmax را اعمال کنیم. در نهایت خواهیم داشت:

$$\alpha(q, k_i) = \frac{\exp(a^T k_i/\sqrt{d})}{\sum_j \exp(a^T k_i/\sqrt{d})}. \qquad (۳۶.۲)$$

۱۵.۲ توجه چند سر

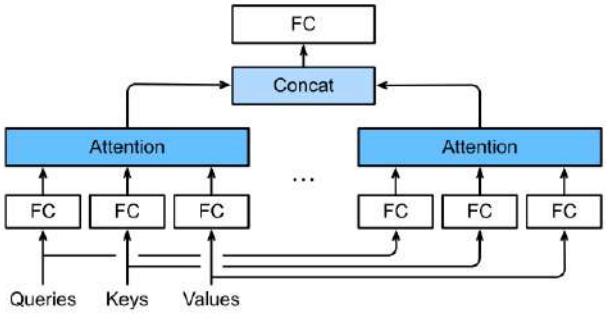
در عمل، با توجه به مجموعه پرسش‌ها، مقادیر و کلیدهای ثابت، شاید بخواهیم که مدلمان اطلاعات رفتارهای متفاوت از ساختار توجه را با هم ترکیب کند. برای مثال وابستگی‌های محدوده[۱] های مختلف (محدودهٔ بلند[۲] یا کوتاه[۳]) در یک دنباله. در نتیجه به نفع ماست که به ساختار توجهمان این اجازه را بدهیم که همزمان از نمایش زیرفضاهای مختلف پرسش‌ها، کلیدها و مقادیر استفاده کند.

برای این کار، به جای اینکه یک بار ادغام توجه انجام دهیم، پرسش‌ها، مقادیر و کلیدها میتوانند تبدیل شوند با h افکنش خطی[۴] که جداگانه آموزش دیده اند. سپس این h پرسش‌ها، مقادیر و کلیدهای افکنده شده به طور موازی به ساختار ادغام توجه داده میشوند. در نهایت، h خروجی ادغام توجه به هم اضافه میشوند و با یک افکنش خطی آموزش دیدهٔ دیگر خروجی نهایی را تولید میکنند. به این طراحی توجه چند سر[۵] گفته میشود چون هر یک از h خروجی ادغام توجه یک سر نام دارد[۳۱].

۱۶.۲ خود توجه

در یادگیری عمیق معمولاً از هاRNN یا هاCNN برای رمزگذاری[۶] دنباله‌ها استفاده میکنیم. حالا با در نظر داشتن ساختار توجه، فرض کنید که که یک دنباله از نشان‌ها را به یک ساختار توجه داده ایم به طوری که در هر پله، هر نشان پرسش، مقادیر و کلیدهای خود را دارد. در اینجا در زمان

[1] Range
[2] Short range
[3] Long range
[4] Linear Projection
[5] Multi-Head Attention
[6] Encoding



شکل ۲۵.۲: توجه چند سر که در آن چندین سر به هم میپیوندند و به طور خطی تبدیل میشوند[۳۲].

محاسبهٔ نمایش[1] نشان در لایهٔ بعدی، نشان میتواند با استفاده از بردار پرسش خود، به نشان‌های دیگر توجه کند (که با بردارهای کلید آن‌ها مطابقت دارد). با استفاده از مجموعهٔ کامل امتیازهای تطابق کلید و پرسش میتوانیم برای هر نشان یک نمایش محاسبه کنیم با ساختن مجموع وزن داری از روی نشان‌های دیگر. چون که هر نشان به هر نشان دیگر توجه میکند، به این ساختار خود توجه[2] میگویند [۲۰].

اگر ورودی دنبالهٔ نشان‌های $x_1, .., x_n$ را داشته باشیم، خود توجه آن یک دنبال با طول برابر $y_1, ..., y_n$ است که در آن

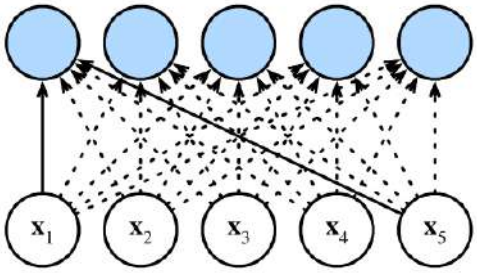
$$y_i = f(x_i, (x_1, x_1), ..., (x_n, x_n)) \qquad (۳۷.۲)$$

طبق تعریف خود توجه.

در خود توجه پرسش‌ها، مقادیر و کلیدها همه ماتریس‌های $n \times d$ هستند. در تابع توجه ضرب داخلی تنظیم شده، یک ماتریس $n \times d$ در یک ماتریس $d \times n$ ضرب میشود، سپس خروجی که یک ماتریس $n \times n$ است در یک ماتریس دیگر $n \times d$ ضرب میشود. در نتیجه خود توجه پیچیدگی زمانی $O(n^2 d)$ دارد. همانطور که در شکل زیر میبینیم، هر نشان مستقیم به هر نشان دیگر متصل است. در نتیجه محاسبات میتواند کاملاً موازی با $O(1)$ عملیات پشت سر هم انجام شود و بیشینه مسیر بین دو نشان هم از $O(1)$ است. با این حال، محاسبات با پیچیدگی درجه دو با توجه به اندازهٔ

[1] Representation
[2] Self Attention

دنباله باعث میشود خود توجه برای دنباله‌های طولانی کند عمل کند.



شکل ۲۶.۲: ساختار خود توجه[۳۲]

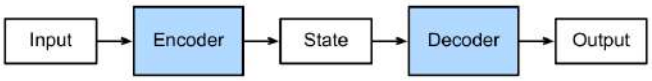
۱۷.۲ رمز گذاری موقعیتی

بر خلاف هاRNN که به طور تکرار شونده نشان‌های یک دنباله یا یک به یک پردازش میکند، خود توجه تکرار را برای موازی کردن محاسبات حذف میکند. دقت کنید که خود توجه، به خودی خود ترتیب دنباله را حفظ نمیکند. برای حفظ این اطلاعات از رمزگذاری موقعیتی استفاده میکنیم، به این معنا که اطلاعات در مورد موقعیت نشان‌ها را به عنوان یک ورودی اضافی که وابسته به نشان است به ساختار توجه میدهیم. به این ورودی اضافی رمز گذاری موقعیتی[1] گفته میشود. این ورودی‌ها هم میتوانند ثابت و از قبل مشخص شده باشند و همچنین میتوانند آموزش داده شوند.

۱۸.۲ مبدل‌ها

خود توجه هم قابلیت محاسبات موازی را دارد و هم مسیر کوتاهی بین ورودی‌ها دارد، به همین دلیل دلخواه است که شبکه‌های عمیق با آن طراحی کنیم. بر خلاف خود توجه‌های اولیه که همچنان وابسته به هاRNN بودند برای نشان دادن ورودی‌ها [۲۰]، مدل مبدل به تنهایی بر اساس ساختار توجه است بدون هیچ لایهٔ پیچشی یا تکرار شونده [۳۱]. با اینکه برای اولین بار این مدل برای یادگیری دنباله به دنباله بر روی دادهٔ متنی بوده، مبدل‌ها کاربرد خود را در زمینه‌ها مختلف یادگیری عمیق در سال‌های گذشته ثابت کرده اند، مانند زبان، بینایی، تشخیص صوت و یادگیری تقویتی.

[1] Positional Encoding



شکل ۲۷.۲: ساختار رمزگذار_رمزگشا[۳۲]

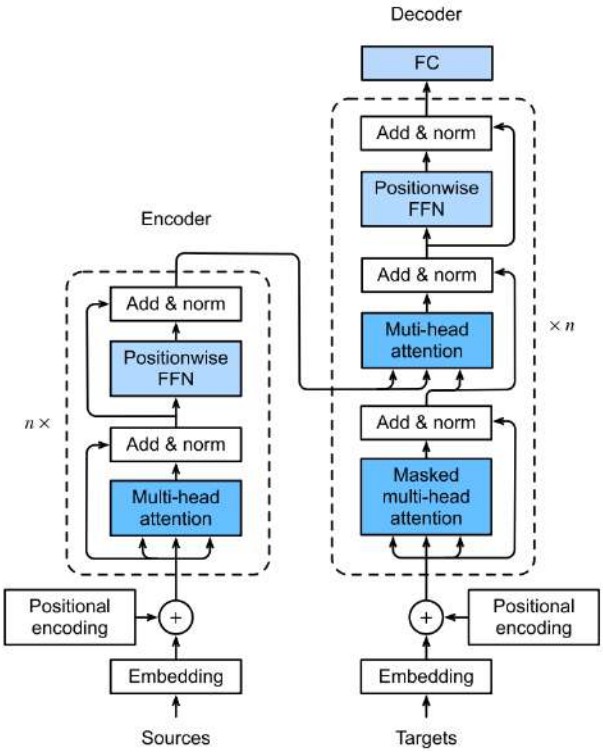
۱.۱۸.۲ مدل

به عنوان یک نمونه از ساختار رمزگذار_رمزگشا[1] (شکل ۲۷.۲)، ساختمان کلی مدل مبدل در شکل ۲۸.۲ دیده میشود. همانطوری که میبینید، مبدل به وجود آمده است از یک رمزگذار و یک رمزگشا. تعبیه[2]های دنبالهٔ ورودی و خروجی اضافه میشوند و رمزگذاری موقعیتی قبل از اینکه به رمزگذار و رمزگشا خورانده شوند که ماژول‌ها را بر روی هم قرار میدهد بر اساس خود توجه.

حالا یک نگاه کلی سطح بالا در شکل ۲۸.۲ ارائه میکنیم. رمزگذار مبدل پشته ای از چندین لایهٔ یکسان است که هر لایه دو زیر لایه[3] دارد. اولین زیرلایه یک ادغام خود توجه چند سر است و دومی یک شبکه پیشخور موقعیتی[4] است. به طور خاص، در رمزگذار خود توجه، پرسش‌ها، کلیدها و مقادیر همه از خروجی لایهٔ قبلی رمزگذار می آیند. با الهام از ساختار ResNet [۱۳] یک اتصال باقی مانده[5] در هر دو زیرلایه وجود دارد. برای هر ورودی $x \in \mathbb{R}^d$ در هر موقعیتی از دنباله، نیاز داریم که $\text{sublayer}(x) \in \mathbb{R}^d$ باشد تا اتصال باقی مانده $x + \text{sublayer}(x) \in \mathbb{R}^d$ امکان پذیر باشد. آین اتصال بلافاصله دنبال میشود با نرمال سازی لایه ای. در نتیجه، رمزگذار مبدل خروجی میدهد یک نمایش برداری d بعدی برای هر موقعیت از دنبالهٔ ورودی.

رمزگشای مبدل نیز همچنین پشته ای از لایه‌های یکسان است با اتصال‌های باقی مانده و نرمال سازی لایه ای. در کنار دو زیر لایهٔ توصیف شده در رمزگذار، رمزگشا یک زیرلایهٔ سوم نیز اضافه میکند، به نام توجه رمزگذار_رمزگشا که بین این دو لایه قرار دارد. در توجه رمزگذار_رمزگشا، پرسش‌ها از لایهٔ قبلی رمزگشا هستند و کلیدها و مقادیر از خروجی رمزگذار مبدل هستند. در رمزگشای خود توجه، پرسش‌ها، مقادیر و کلیدها همه از لایهٔ قبلی رمزگشا هستند. با این وجود، هر موقعیت در رمزگشا تنها اجازه دارد توجه کند به تمامی موقعیت‌ها در رمزگشا که قبل از این موقعیت

[1] Encoder-Decoder
[2] Embedding
[3] Sublayer
[4] Positionwise Feed-Forward Network
[5] Residual Connection



شکل ۲۸.۲: ساختار کلی یک مدل مبدل[۳۲]

آمده اند. این توجه ماسک دار خصوصیت خود برگشت کننده را حفظ میکند و تضمین میکند که پیش بینی فقط وابسه به نشان[1]های خروحی باشد که تا الان تولید شده است.

[1] Token

فصل ۳

بازشناسی فعالیت‌های انسانی

حالا با استفاده از دانش کسب کرده تا اینجا میخواهیم مدلی طراحی کنیم تا بتواند فعالیت‌های انسانی کوتاه مدت را تشخیص دهد و در آن واحد بتواند پردازش کند. برای تست مدلمان از دادگان MPOSE۲۰۲۱ استفاده میکنیم که مجموعه داده ای برای فعالیت‌های انسانی کوتاه مدت است.

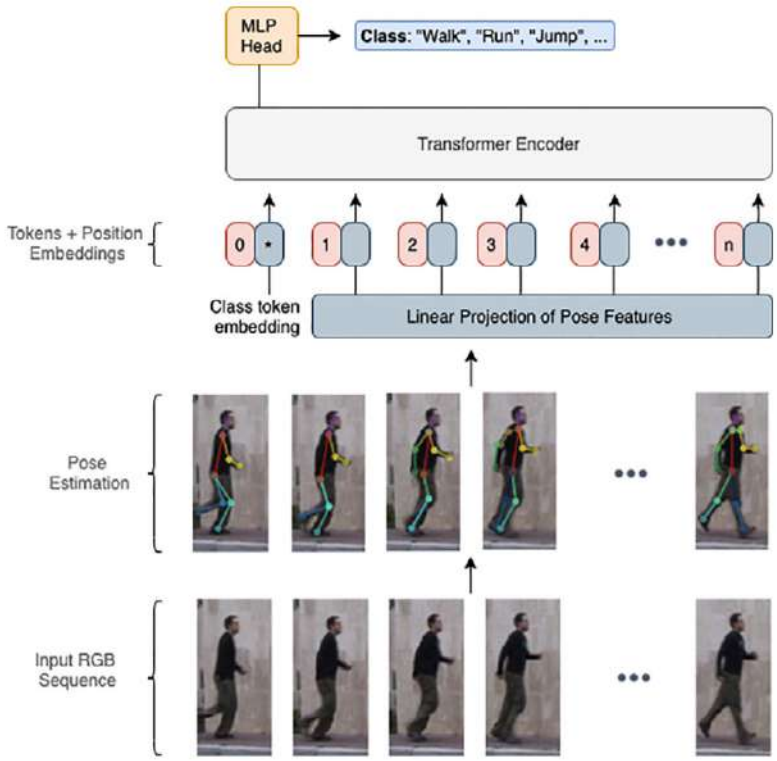
۱.۳ مدل مبدل عمل

در این بخش ساختمان مدل مبدل عمل[۲] (AcT) را توضیح میدهیم ۱.۳. یک دنبالهٔ ورودی ویدیو که T فریم دارد با ابعاد $H \times W$ و C کانال $X_{rgb} \in \mathbb{R}^{T \times H \times W \times C}$ پیش پردازش میشود توسط یک شبکه تخمین حالت چند نفرهٔ دو بعدی

$$X_{2Dpose} = F_{2Dpose}(X_{rgb}) \tag{۱.۳}$$

که حالت‌های دو بعدی انسان‌ها را اخراج میکند به ابعاد $N \times T \times P$، که N تعداد افراد تصویر و P تعداد نقاط کلیدی تشخیص داده شده توسط شبکه است (این تعداد همیشه ثابت است). ساختار مبدل به عنوان ورودی یک دنبالهٔ یک بعدی از نشان‌های تعبیه شده میگیرد، در نتیجه هر یک از N دنبالهٔ ماتریس‌های حالت $X_{2Dpose} \in \mathbb{R}^{T \times P}$ به صورت جداگانه توسط شبکه AcT پردازش میشوند. با این حال، در زمان اجرا، تمامی حالت‌های تشخیص داده شده در فریم ویدیو

[2] Action Transformer



شکل ۱.۳: ساختار کلی مدل مبدل عمل [۲۲].

میتوانند به صورت دسته ای پردازش شوند توسط AcT و همزمان یک پیش بینی برای تمامی N فاعل تولید کند.

اول، T حالت نگاشته میشوند به ابعاد بالاتر D_{model} با استفاده از یک نگاشت خطی $W^{l_0} \in \mathbb{R}^{P \times D_{model}}$. همانند Bert [۶] و مبدل‌های بینایی[1] [۷]، یک بردار قابل آموزش به ابعاد D_{model} به دنبالهٔ ورودی T اضافه میشود. این نشان کلاس (CLS) خود توجه را مجبور میکند که اطلاعات را به صورت یک نمایش ابعاد بالا تجمیع کند که کلاس‌های مختلف اعمال را جدا میکند. در ادامه، اطلاعات موقعیتی نیز به دنباله داده میشود توسط یک ماتریس رمزگذاری موقعیتی $X_{pos} \in \mathbb{R}^{(T+1) \times D_{model}}$ که به تمامی نشان‌ها اضافه میشود.

نشان‌های خطی نگاشت شده و CLS سپس به یک رمزگذار مبدل استاندارد F_{Enc} داده میشود

[1] Vision Transformers

که دارای L لایه است و به دست خواهیم آورد:

$$X^L = F_{Enc}(X^{l_0}) = F_{Enc}([X^{l_0}_{cls}; X_{2Dpos}] + X_{pos}), \qquad (۲.۳)$$

که در آن $X^L \in \mathbb{R}^{(T+1)\times D_{model}}$ نمایش کلی تولید شده توسط رمزگذار مبدل در لایهٔ آخر است. در نهایت، تنها نشان CLS، x_{cls}، به سر دسته بندی خطی MLP_{head} داده میشود که پیش بینی نهایی دسته بندی را انجام میدهد

$$\hat{z} = MLP_{head}(x^L_{cls}), \qquad (۳.۳)$$

که در آن $\hat{z}$ بردار خروجی مدل است. در زمان تمرین، سیگنال نظارت فقط از نشان CLS میاید، در حالی که تمامی T نشان باقیمانده تنها ورودی‌های مدل هستند. دقت کنید که چگونه طبیعت مدل این اجازه را میدهد که تعداد کمتری فریم را قبول کند حتی اگر با T ثابت آموزش دیده باشد. این یک درجه آزادی بیشتر در زمان اجرا به ما میدهد که باعث میشود AcT از مدل‌های موجود سازگارپذیر تر باشد.

شبکه به دست آمده یک جواب سبک است که قابلیت تشخیص همزمان اعمال چندین فرد را در یک ویدیو دارد با دقت بالا. مزیت استفاده از تشخیص حالت‌ها این است که باعث میشود بتوانیم کارایی در آن واحد داشته باشیم با تاخیر پایین و مصرف انرژی کم.

جدول ۱.۳: پارامترهای مبدل عمل برای ۴ اندازهٔ مختلف. ما فیکس میکنیم $D_{model}/\mathrm{H} = 64$ که H تعداد سرهای مدل است. H ، D_{mlp} و L را به طور خطی اضافه میکنیم تا ورژن‌های مختلف مدل را به دست بیاوریم [۲۲].

Model	H	D_{model}	D_{mlp}	L	Parameters
AcT-μ	1	64	256	4	227k
AcT-S	2	128	256	5	1,040k
Act-M	3	192	256	6	2,740k
Act-L	4	256	512	6	4,902k

جدول ۲.۳: هایپرپارامترهای استفاده شده در آزمایش‌ها [۲۲].

Training	
Training Epochs	350
Batch size	512
Optimizer	AdamW
Warmup Epochs	40%
Step epochs	80%
Regularization	
Weight decay	$1e-4$
Label smoothing	0.1
Dropout	0.3
Random flip	50%
Random noise σ	0.03

۲.۳ آزمایش‌ها و نتایج

در این بخش آزمایش‌ها اصلی انجام شده برای مطالعهٔ برتری‌های استفاده از یک مدل کاملاً خود توجه برای تشخیص عمل دو بعدی را میبینیم. اول، ۴ ورژن مختلف از AcT را که در ۱.۳ میبینیم مقایسه میشوند با مدل‌های به روز و پایه برای حل این مسئله بر روی دادگان MPOSE۲۰۲۱. فقط برای یک مقایسهٔ خاص با ST-TR [۲۴] و MS-G۳D [۲۱]، از ورژن‌های جمع شدهٔ مدلمان به نام AcT-$\mu(\times n)$ استفاده میکنیم که (n) نشان دهندهٔ تعداد نمونه[1]های جمع شده[2] است. سپس تاخیر[3] مدل را برای تمامی ساختارهای طراحی شده با هایCPU مختلف تست میکنیم و نشان میدهیم که AcT میتواند برای کابردهای آنی[4] استفاده شود.

۱.۲.۳ تنظیمات آزمایش‌ها

در آزمایش‌ها ما هم از OpenPose [۳] و هم از PoseNet [۱۷] بر روی MPOSE۲۰۲۱ استفاده میکنیم. هر کدام از مجموعه داده‌ها دارای $T = 30$ هستند و OpenPose دارای $P = 52$ و PoseNet دارای ۶۸ خصوصیت هستند. به طور خاص برای OpenPose ۱۳ نقطهٔ کلیدی با

[1] Instances
[2] Ensemble
[3] Latency
[4] Real-Time

۴ پارامتر برای هر کدام (مختصات دو بعدی و بردار سرعت دو بعدی) داریم، و برای PoseNet ۱۷ نقطهٔ کلیدی با همان ۴ پارامتر داریم. مجموعه دادهٔ تمرینی شامل 9421 نمونه است و 2867 تا نمونه هم برای تست داریم. 3141 نمونه باقیمانده به عنوان مجموعه تاییدی[1] استفاده میشود.

در ۱.۳ میتوانید به طور خلاصه پارامترها را برای هر یک از ورژن‌های مختلف AcT ببینید و همچنین در ۲.۳هایپرپارامترهای کلی مشترک بین ورژن‌ها برای آموزش را میبینید. از بهینه ساز[2] AdamW استفاده شده با کاهش نرخ آموزش $1e-4$ با درصد ثابت 80% از دوره‌ها[3]. از TensorFlow ۲ استفاده کرده ایم و بر روی یک کامپیوتر شخصی با GB-۳۲ رم و Intel i7-9700K CPU و یک Nvidia 2080 Super Gp-GPU اجرا کرده ایم. با این پارامترها و سیستم گفته شده تمرین تمامی ورژن‌های مدلمان حدود ۳۲ ساعت بر روی ۳ تقسیم[4] مختلف طول کشید. برای مدل‌های مقایسه شده نیز از کدهای ارائه شده توسط نویسنده‌های مقاله و با پارامترهای پیشنهادی خودشان استفاده میکنیم.

۲.۲.۳ آزمایش‌ها بر روی MPOSE۲۰۲۱

ما به طور کامل آزمایش‌ها را بر روی MPOSE۲۰۲۱ انجام داده ایم با استفاده از مدل‌های پایه، مدل‌های به روز و .AcT ما میانگین و انحراف معیار ۱۰ مدل را گذارش میکنیم با استفاده از ۱۰ تقسیم تایید. تقسیم‌های تاییدها برای تمامی مدل‌ها ثابت است و برابر 10% مجموعه آموزشی است. نتایج را هم بر اساس OpenPose و هم بر اساس PoseNet به دست میاوریم با استفاده از هر سه تقسیم آموزش/تست که توسط MPOSE۲۰۲۱ ارائه میشود. مدل‌های پایه انتخاب شده هستند یک پرسپترون چند لایه ،(MLP) یک مدل کاملا پیچشی (Conv۱D) و REMNet [۲۵] که یک شبکه پیچشی پیچیده تر است با توجه و بلوک‌های باقیمانده طراحی شده است برای استخراج ویژگی دادهٔ سری زمانی.

به طور دقیق تر، MLP طراحی شده است به صورت یک پشته از سه لایهٔ کاملا متصل هر کدام با ۵۱۲ نورون، و در ادامه یک لایهٔ بیرون ریز و یک لایهٔ نهایی کاملا متصل. Conv۱D نیز تشکیل شده است از به هم چسباندن ۵ لایهٔ یک بعدی پیچشی با ۵۱۲ فیلتر که بین آن‌ها لایهٔ نرمال سازی دسته ای وجود دارد و در ادامه یک عملیات ادغام سازی میانگین انجام میشود همراه با یک

[1] Validation set
[2] Optimizer
[3] Epochs
[4] Split

MPOSE2021 Split		OpenPose 1		OpenPose 2		OpenPose 3	
Model	Parameters	Accuracy [%]	Balanced [%]	Accuracy [%]	Balanced [%]	Accuracy [%]	Balanced [%]
MLP	1,319k	82.66 ± 0.33	74.56 ± 0.56	84.41 ± 0.60	74.58 ± 1.0	83.48 ± 0.58	76.60 ± 0.77
Conv1D	4,037k	88.18 ± 0.64	81.97 ± 1.4	88.93 ± 0.43	80.49 ± 0.95	88.67 ± 0.38	83.93 ± 0.58
REMNet	4,211k	89.18 ± 0.51	84.20 ± 0.84	88.77 ± 0.35	80.29 ± 0.88	89.80 ± 0.59	86.18 ± 0.40
ActionXPose	509k	87.60 ± 0.98	82.13 ± 1.5	88.42 ± 0.70	81.28 ± 1.4	89.96 ± 1.0	86.65 ± 1.6
MLSTM-FCN	368k	88.62 ± 0.74	83.55 ± 0.88	90.19 ± 0.68	83.84 ± 1.2	89.80 ± 0.94	87.33 ± 0.67
T-TR	3,036k	87.72 ± 0.87	81.99 ± 1.64	88.14 ± 0.53	80.23 ± 1.19	88.69 ± 0.95	85.03 ± 1.60
MS-G3D (J)	2,868k	89.90 ± 0.50	85.29 ± 0.98	90.16 ± 0.64	83.08 ± 1.1	90.39 ± 0.44	87.48 ± 1.2
AcT-μ	227k	90.86 ± 0.36	86.86 ± 0.50	91.00 ± 0.24	85.01 ± 0.51	89.98 ± 0.47	87.63 ± 0.54
AcT-S	1,040k	91.21 ± 0.48	87.48 ± 0.76	91.23 ± 0.19	85.66 ± 0.58	90.90 ± 0.87	88.61 ± 0.73
AcT-M	2,740k	**91.38 ± 0.32**	**87.70 ± 0.47**	91.08 ± 0.48	85.18 ± 0.80	91.01 ± 0.57	88.63 ± 0.51
AcT-L	4,902k	91.11 ± 0.32	87.27 ± 0.46	**91.46 ± 0.42**	**85.92 ± 0.63**	**91.05 ± 0.80**	**89.00 ± 0.74**
ST-TR	6,072k	89.20 ± 0.71	83.95 ± 1.11	89.29 ± 0.81	81.53 ± 1.39	90.49 ± 0.53	87.06 ± 0.70
MS-G3D (J+B)	5,735k	91.13 ± 0.33	87.25 ± 0.50	91.28 ± 0.29	85.10 ± 0.50	91.42 ± 0.54	89.66 ± 0.55
AcT-μ (x2)	454k	91.76 ± 0.29	88.27 ± 0.37	91.34 ± 0.40	86.88 ± 0.48	91.70 ± 0.57	88.87 ± 0.37
AcT-μ (x5)	1,135k	92.43 ± 0.24	89.33 ± 0.31	91.55 ± 0.37	87.80 ± 0.39	92.63 ± 0.55	89.77 ± 0.35
AcT-μ (x10)	2,271k	**92.54 ± 0.21**	**89.79 ± 0.34**	**92.03 ± 0.33**	**88.02 ± 0.31**	**93.10 ± 0.53**	**90.22 ± 0.31**

شکل ۲.۳: مقایسهٔ عملکرد مدل‌های مختلف برای تشخیص عمل کوتاه مدت بر روی مجموعه دادهٔ MPOSE۲۰۲۱ با استفاده از بستهٔ OpenPose [۲۲].

لایهٔ بیرون ریز و یک لایهٔ کاملا متصل برای خروجی. در نهایت، RemNet تشکیل شده است از دو ماژول باقیماندهٔ کاهشی (RRM) [۱] با ۵۱۲ فیلتر که در ادامه همان لایه‌های خروجی مدل‌های دیگر را دارد.

برای مدل‌های به روز، ۴ مدل پرطرفدار انتخاب و آزمایش شده اند. MLSTM-FCN [۱۶] لایه‌های پیچشی را ترکیب میکند با توجه مکانی و یک بلوک .LSTM همچنین ورژن بهبود یافتهٔ آن ActionXPose [۱] از پیش پردازش‌های بیشتری بهره میبرد که باعث میشود مدل در مقابل نویز و دادهٔ گم شده مقاوم تر باشد. در ادامه، MS-G۳D [۲۱] بهره میبرد از پیچش‌های گرافی زمانی_مکانی تا مدل را آگاه کند از روابط مکانی و زمانی نقاط کلیدی اسکلت با هم. در کنار این‌ها، ST-TR [۲۴] از اعمال ترکیب پیچش‌های گرافی با ساختار مبدل خود توجه بر روی زمان و مکان استفاده میکند. از آن جایی که دو مدل آخر همچنین یک مدل مجموع پیشنهاد میدهند، نتایج را با مجموع AcT به دست آمده از ۲، ۵ و ۱۰ مدل یکباره ای[۲] مقایسه میکنیم.

نتایج آزمایش‌ها با استفاده از OpenPose را در ۲.۳ میبینید. مدل پیچشی عملکرد بسیار بهتری از MLP دارد در حال که REMNet نشان میدهد که اضافه کردن ساختار توجه دقت را بیشتر افزایش میدهد. با نگاه کردن به نتایج MLSTM-FCN و ActionXPose میتوان دید که

[1] Residual Reduction Module

[2] Single-Shot

به صراحت مدل کردن ارتباط‌های مکانی و زمانی مقدار کمی دقت را نسبت به شبکه‌هایی مانند REMNet افزایش میدهد. MS-G۳D با به کاربرد گرفتن پیچش‌های گرافی و استخراج اطلاعات ارتباطات مکانی بین نقطه‌ها دقت را بیشتر افزایش میدهد ([۱]در ورژن تنها مفصل (J) خود). در کنار این‌ها، ST-TR نیز نتایجی قابل مقایسه با دیگر مدل‌های یکباره نشان میدهد با اینکه مانند MS-G۳D از اطلاعات گرافی استفاده میکند.

مدل پیشنهاد شده AcT در هر ۴ ورژن خود از بقیه مدل‌ها بهتر عمل میکند در حالی که انحراف معیار کمتری نیز دارند. حتی کوچکترین مدل μAcT نیز میتواند به خوبی اطلاعات را استخراج کند و دقت بسیار بالایی را نتیجه بدهد و با افزایش پارامترها بهبود در دقت را مشاهده میکنیم در هر سه تقسیم.

همانطور که گفته شد ST-TR از ترکیب دو مدل برای پردازش اطلاعات مکان و زمان استفاده میکند و همچنین MS-G۳d نیز از چند مدل جدا برای پردازش اطلاعات مختلف استفاده میکند. استفاده از چند مدل این خوبی را دارد که هر مدل یک خصوصیت جدا را یاد میگیرد و دقت بالاتری بوجود میاید. پس برای اینکه مقایسهٔ برابری داشته باشیم، سه ورژن مجموع از μAcT میسازیم که با ۲، ۵ و ۱۰ نمونه درست شده اند. برای پیش بینی مدل مجموع، ما خروجی‌های نمونه‌ها (لگاریتم‌های احتمال) را میانگین میگیریم و آن را به یک Softmax میدهیم. در نهایت میبینم که مدل‌های مجموع حتی دقت‌های بالاتری کسب کرده اند بر روی هر سه قسمت با تنها یک تا دو میلیون پارامتر. این نشان میدهد که حتی بدون اضافه کردن اطلاعات به مدل و فقط با مجموع سازی مدل‌ها میتوانیم پیش بینی بهتری داشته باشیم.

MPOSE2021 Split		PoseNet 1		PoseNet 2		PoseNet 3	
Model	Parameters	Accuracy [%]	Balanced [%]	Accuracy [%]	Balanced [%]	Accuracy [%]	Balanced [%]
Conv1D	4,062k	85.83 ± 0.71	79.96 ± 1.1	87.47 ± 0.35	78.51 ± 0.78	87.46 ± 0.67	81.31 ± 0.58
REMNet	4,269k	84.75 ± 0.65	77.23 ± 0.94	86.17 ± 0.68	75.79 ± 1.3	86.31 ± 0.60	79.20 ± 0.79
ActionXPose	509k	75.98 ± 0.72	64.47 ± 1.1	79.94 ± 1.1	67.05 ± 1.4	77.34 ± 1.4	66.86 ± 1.4
MLSTM-FCN	368k	76.17 ± 0.84	64.75 ± 1.1	79.04 ± 0.72	65.62 ± 1.4	77.84 ± 1.3	67.05 ± 1.2
AcT-μ	228k	86.66 ± 1,10	81.56 ± 1.60	87.21 ± 0.99	79.21 ± 1.60	87.75 ± 0.53	82.99 ± 0,87
AcT-S	1,042k	**87.63 ± 0.52**	**82.54 ± 0.87**	88.48 ± 0.57	81.53 ± 0.68	88.49 ± 0.65	83.63 ± 0.99
AcT-M	2,743k	87.23 ± 0.48	82.10 ± 0.66	**88.50 ± 0.51**	**81.79 ± 0.44**	**88.70 ± 0.57**	**83.92 ± 0.96**

شکل ۳.۳: مقایسهٔ دقت مدل‌های مختلف بر روی دادگان MPOSE۲۰۲۱ با استفاده از بستهٔ PoseNet [۲۲].

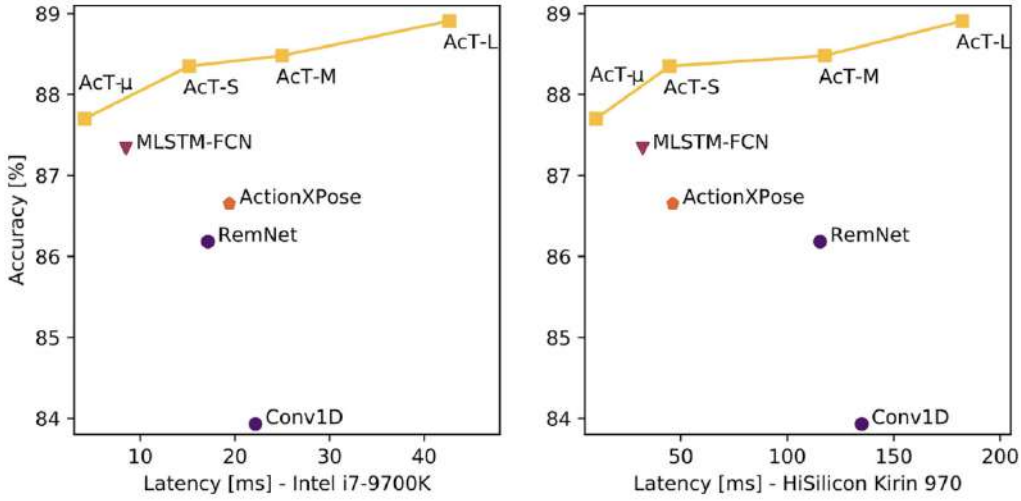
چون که دادهٔ PoseNet فقط برای کاربردهای آنی طراحی شده است، فقط از مدل‌هایی که

[1] Joint

برای این کار طراحی شده است استفاده میکنیم و از MS-G۳D و ST-TR و Act-L استفاده نمیکنیم. در کل نتایج مانند حالت قبلی است و میبینیم که مدل ما از بقیه مدل‌ها دقت بسیار بالاتری دارد. با این تفاوت که فرق دقت مدل ما با مدل‌های دیگر در این حالت چشمگیرتر است. همچنین میبینیم که در این حالت برعکس قبلی، Conv۱D دقت بسیار بالاتری از بقیه مدل‌های مورد مقایسه نشان میدهد و همچنین MLP دقت بسیار پایینی دارد چون PoseNet برای کاربردهای آنی و سریع طراحی شده است، در نتیجه خطا و نویز بیشتری دارد و همچنین داده‌های گمشده ی بیشتری، و این باعث میشود که برای MLP بیش برازش اتفاق بیفتد. همچنین قابل توجه است که در این حالت انحراف معیار نیز بیشتر از حالت قبلی است.

۳.۲.۳ بررسی تاخیر

ما عملکرد مدل‌های مختلف را بر روی دو مدل CPU مختلف یکی برای کامپیوترهای شخصی و یکی برای موبایل تست میکنیم. دو CPU ای که برای تست سرعت از آن‌ها استفاده شده یک Intel i7-9700K است و یک ARM-based HiSilicon Kirin 970. نتایج را در ۴.۳ میبینید که در آن نتایج MLP آورده نشده است، چون با وجود اینکه از بقیهٔ مدل‌ها سریع تر است ولی دقت پایین آن این مدل را بی ارزش میکند. در گراف‌ها دقت و سرعت بالای مدل‌های AcT را مشاهده میکنید که این مدل‌ها با وجود دقت بالاتر، سرعت بیشتری نیز دارند. تنها مدلی که نتیجه ای نزدیک به AcT داشت MLSTM-FCN بود که سرعت و دقتی مشابه AcT-μ داشت. با افزایش اندازهٔ مدل AcT، دقت مقداری اضافه تر میشد ولی تاخیر نیز به اندازهٔ قابل توجهی افزایش پیدا میکرد.



شکل ۴.۳: مقایسهٔ تاخیر جوابدهی مدل‌های مختلف بر روی دو نوع CPU مختلف[۲۲].

فصل ۴

نتیجه گیری

با دیدن نتایج آزمایش‌ها و بررسی مدل‌های مختلف با ایده‌های مختلف دیدیم که مدل ما که فقط بر اساس ساختار خودتوجه و مدل مبدل بود، بسیار نتایج بهتری (هم سریع تر هم دقت بیشتر) نشان داد. همچنین دیدیم که صرفا با مجموع کردن مدل‌ها دقت باز هم افزایش قابل توجهی داشت با اینکه اندازهٔ مدل همچنان خیلی بزرگ نبود. معرفی این مدل تحولی در زمینهٔ تشخیص عمل کوتاه مدت انسان بود و قطعاً در آینده پیشرفت‌های فراوانی را در این زمینه با استفاده از پیشرفت و ترکیب این مدل با مدل‌های دیگر خواهیم دید.

با توجه به ساختار طبیعی گراف مانند داده‌های ورودی ما به نظر الگوریتم‌های گرافی و شبکه‌های عصبی گرافی میتوانند قدم بعدی خوبی برای پیشرفت در این مسئله باشند. همچنین پیش پردازش بهتر با استفاده از هاCNN و یا ترکیب چندین ساختار خود توجه در نهایت با یک مدل RNN یا LSTM میتواند دقت مدل‌ها را افزایش بدهد و این امکان را بدهد که دنباله‌های بلند تری را در زمان کمتری پردازش کنیم.

واژه‌نامهٔ فارسی به انگلیسی

آنی ..... Real-Time
اتصال باقی مانده ..... Residual Connection
ادغام بیشینه ..... Max Pooling
ادغام توجه ..... Attention Pooling
ادغام میانگین ..... Average Pooling
ارتباط بازخوردی ..... Feedback Connection
افزایش نمونه ..... Upsampling
افکنش خطی ..... Linear Projection
الحاق ..... Interpolation
انتشار ..... Propogation
ایه‌های پنهان ..... Hidden Layer
باز کردن ..... Unfolding
بازگشت ادغام ..... unpooling
بازگشتی ..... Recursive
بخش بندی تصویر ..... Image Segmentation
بیرون ریزی ..... Dropout
بیش برازش ..... Overfitting
بینایی ماشین ..... Computer Vision
تابع فعال سازی ..... Activation Function
تابع هزینه ..... Cost function
تخمین حالت ..... Pose Estimation

تخمین حرکت ..... Motion Estimation
تشخیص تصویری کاراکترها ..... Optical Character Recognition
تشخیص فعالیت‌های انسانی ..... Action Recognition
تشخیص کاراکتر هوشمند ..... Intelligent Character Recognition
تعبیه ..... Embedding
تعصب ..... Bias
تعمیم ..... Generalization
توابع امتیاز دهی توجه ..... Attention Scoring Functions
توابع هسته ای ..... Kernel Functions
توجه ..... Attention
تکرارشونده ..... Recurrent
جمع شده (مجموع) ..... ensemble
حافظهٔ بلند کوتاه‌ـمدت ..... Long Short-Term Memory
خصوصیت ..... feature
خطای آموزش ..... Training Error
خطای تست ..... Test Error
خود توجه ..... Self Attention
خود توجه چند سره ..... Multi-Head Self-Attention
خوشه بندی ..... Clustering
دریچهٔ بروزرسانی ..... Update Gate
دریچهٔ تنظیم مجدد ..... Reset gate
دسته بندی عکس‌ها ..... Image Classification
رمز گذاری موقعیتی ..... Positional Encoding
رمزگذار ..... Encoder
رمزگذاری ..... Encoding
رمزگشا ..... Decoder
زیر لایه ..... Sublayer
سلول حافظه ..... Memory Cell
شایسته سازی ..... Conditioning

شبکه عصبی تکرار شونده ..... Recurrent Neural Network
شبکه عصبی پیچشی ..... Convolutional Neural Network
شبکه عصبی گرافی ..... Graph Neural Network
شبکه عمیق پیشخور ..... Deep Feedforward Networks
شبکه پیشخور ..... Feedforward Network
شبکه پیشخور موقعیتی ..... Positionwise Feed-Forward Network
شناسایی اشیاء ..... Object Identification
عمق ..... Depth
عوامل نرم افزاری ..... Software agents
فاصله گذاری ..... Dilation
فرایند تصمیم گیری مارکف ..... Markov Decision Process
فزایش داده‌ها ..... Data Augmentation
قدم زدن ..... Stride
قدم زمانی ..... Time Step
قدم کسری ..... Fractional stride
لایه گذاری ..... Padding
لایهٔ خروجی ..... Output Layer
مبدل ..... Transformer
مبدل بینایی ..... Vision Transformer
مبدل عمل ..... Action Transformer
متمایز کننده ..... Discriminative
مجموعه ویژگی ..... Feature set
مجموعهٔ تاییدی ..... Validation Set
محدوده ..... Range
مفصل ..... joint
منظم سازی ..... Regularization
ناپدید شدن ..... Vanishing
نرمال سازی دسته ای ..... Batch Normalization
نزول گرادیان ..... Gradient Descent

نشان ..... Token
نقشه‌های خصوصیات ..... Feature maps
نمایش ..... Representation
نمونه ..... Instance
نگاشت ..... mapping
واحد ..... Unit
واحد تکرار شونده دریچه دار ..... Gated Recurrent Unit
وضعیت ..... State
وضعیت داخلی ..... Internal State
وظیفه ..... Task
پردازش زبان طبیعی ..... Natural Language Processing
پس انتشار در طول زمان ..... Back-Propogation Through Time
پیچش ..... Convolution
پیچش به عقب ..... Backward Convolution
کامل ..... Dense
کاملاً متصل ..... Fully Connected
کاهش ابعاد ..... Dimensionality Reduction
کاهش نمونه ..... Downsampling
کم برازش ..... Underfitting
گره ..... Node
یادگیری بدون نظارت ..... Unsupervised Learning
یادگیری تقویتی ..... Reinforcement Learning
یادگیری عمیق ..... Deep Learning
یادگیری ماشین ..... Machine Learning
یادگیری نظارتی ..... Supervised Learning
یادگیری نیمه نظارتی ..... Semi-Supervised Learning

واژه‌نامهٔ انگلیسی به فارسی

تشخیص فعالیت‌های انسانی ..... Action Recognition
مبدل عمل ..... Action Transformer
تابع فعال سازی ..... Activation Function
ادغام توجه ..... Attention Pooling
توابع امتیاز دهی توجه ..... Attention Scoring Functions
توجه ..... Attention
ادغام میانگین ..... Average Pooling
پس انتشار در طول زمان ..... Back-Propogation Through Time
پیچش به عقب ..... Backward Convolution
نرمال سازی دسته ای ..... Batch Normalization
تعصب ..... Bias
خوشه بندی ..... Clustering
بینایی ماشین ..... Computer Vision
شایسته سازی ..... Conditioning
پیچش ..... Convolution
شبکه عصبی پیچشی ..... Convolutional Neural Network
تابع هزینه ..... Cost function
فزایش داده‌ها ..... Data Augmentation
رمزگشا ..... Decoder
شبکه عمیق پیشخور ..... Deep Feedforward Networks
یادگیری عمیق ..... Deep Learning

کامل ..... Dense
عمق ..... Depth
فاصله گذاری ..... Dilation
کاهش ابعاد ..... Dimensionality Reduction
متمایز کننده ..... Discriminative
کاهش نمونه ..... Downsampling
بیرون ریزی ..... Dropout
تعبیه ..... Embedding
رمزگذار ..... Encoder
رمزگذاری ..... Encoding
نقشه‌های خصوصیات ..... Feature maps
مجموعه ویژگی ..... Feature set
ارتباط بازخوردی ..... Feedback Connection
شبکه پیشخور ..... Feedforward Network
قدم کسری ..... Fractional stride
کاملاً متصل ..... Fully Connected
واحد تکرار شونده دریچه دار ..... Gated Recurrent Unit
تعمیم ..... Generalization
نزول گرادیان ..... Gradient Descent
شبکه عصبی گرافی ..... Graph Neural Network
ایه‌های پنهان ..... Hidden Layer
دسته بندی عکس‌ها ..... Image Classification
بخش بندی تصویر ..... Image Segmentation
نمونه ..... Instance
تشخیص کاراکتر هوشمند ..... Intelligent Character Recognition
وضعیت داخلی ..... Internal State
الحاق ..... Interpolation
توابع هسته ای ..... Kernel Functions
افکنش خطی ..... Linear Projection

حافظهٔ بلند کوتاه‌ـمدت ..... Long Short-Term Memory
یادگیری ماشین ..... Machine Learning
فرایند تصمیم گیری مارکف ..... Markov Decision Process
ادغام بیشینه ..... Max Pooling
سلول حافظه ..... Memory Cell
تخمین حرکت ..... Motion Estimation
خود توجه چند سره ..... Multi-Head Self-Attention
پردازش زبان طبیعی ..... Natural Language Processing
گره ..... Node
شناسایی اشیاء ..... Object Identification
تشخیص تصویری کاراکترها ..... Optical Character Recognition
لایهٔ خروجی ..... Output Layer
بیش برازش ..... Overfitting
لایه گذاری ..... Padding
تخمین حالت ..... Pose Estimation
رمز گذاری موقعیتی ..... Positional Encoding
شبکه پیشخور موقعیتی ..... Positionwise Feed-Forward Network
انتشار ..... Propogation
محدوده ..... Range
آنی ..... Real-Time
شبکه عصبی تکرار شونده ..... Recurrent Neural Network
تکرارشونده ..... Recurrent
بازگشتی ..... Recursive
منظم سازی ..... Regularization
یادگیری تقویتی ..... Reinforcement Learning
نمایش ..... Representation
دریچهٔ تنظیم مجدد ..... Reset gate
اتصال باقی مانده ..... Residual Connection
خود توجه ..... Self Attention

یادگیری نیمه نظارتی ..... Semi-Supervised Learning
عوامل نرم افزاری ..... Software agents
وضعیت ..... State
قدم زدن ..... Stride
زیر لایه ..... Sublayer
یادگیری نظارتی ..... Supervised Learning
وظیفه ..... Task
خطای تست ..... Test Error
قدم زمانی ..... Time Step
نشان ..... Token
خطای آموزش ..... Training Error
مبدل ..... Transformer
کم برازش ..... Underfitting
باز کردن ..... Unfolding
واحد ..... Unit
یادگیری بدون نظارت ..... Unsupervised Learning
دریچهٔ بروزرسانی ..... Update Gate
افزایش نمونه ..... Upsampling
مجموعهٔ تاییدی ..... Validation Set
ناپدید شدن ..... Vanishing
مبدل بینایی ..... Vision Transformer
جمع شده (مجموع) ..... ensemble
خصوصیت ..... feature
خصوصیت ..... feature
مفصل ..... joint
نگاشت ..... mapping
بازگشت ادغام ..... unpooling

کتاب‌نامه

[1] Angelini, Federico, Fu, Zeyu, Long, Yang, Shao, Ling, and Naqvi, Syed Mohsen. Actionxpose: A novel 2d multi-view pose-based algorithm for real-time human action recognition, 2018.

[2] Bahdanau, Dzmitry, Cho, Kyunghyun, and Bengio, Yoshua. Neural machine translation by jointly learning to align and translate, 2016.

[3] Cao, Z., Hidalgo Martinez, G., Simon, T., Wei, S., and Sheikh, Y. A. Openpose: Realtime multi-person 2d pose estimation using part affinity fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2019.

[4] Cho, Kyunghyun, van Merrienboer, Bart, Bahdanau, Dzmitry, and Bengio, Yoshua. On the properties of neural machine translation: Encoder-decoder approaches, 2014.

[5] Chung, Junyoung, Gulcehre, Caglar, Cho, KyungHyun, and Bengio, Yoshua. Empirical evaluation of gated recurrent neural networks on sequence modeling, 2014.

[6] Devlin, Jacob, Chang, Ming-Wei, Lee, Kenton, and Toutanova, Kristina. BERT: Pre-training of deep bidirectional transformers for language understanding. pages 4171–4186. Association for Computational Linguistics, June 2019.

[7] Dosovitskiy, Alexey, Beyer, Lucas, Kolesnikov, Alexander, Weissenborn, Dirk, Zhai, Xiaohua, Unterthiner, Thomas, Dehghani, Mostafa, Minderer, Matthias, Heigold, Georg, Gelly, Sylvain, Uszkoreit, Jakob, and Houlsby, Neil. An image is worth 16x16 words: Transformers for image recognition at scale, 2021.

[8] Dumoulin, Vincent and Visin, Francesco. A guide to convolution arithmetic for deep learning, 2018.

[9] Freedman, David. *Statistical Models : Theory and Practice*. Cambridge University Press, August 2005.

[10] Glassner, Andrew. *Deep Learning From Basics to Ptractice Volume 1*. No Startch Press, 2018.

[11] Goodfellow, Ian, Bengio, Yoshua, and Courville, Aaron. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[12] Graves, Alex. *Supervised Sequence Labelling with Recurrent Neural Networks*. Studies in computational intelligence. Springer, Berlin, 2012.

[13] He, Kaiming, Zhang, Xiangyu, Ren, Shaoqing, and Sun, Jian. Deep residual learning for image recognition, 2015.

[14] Hochreiter, Sepp and Schmidhuber, Jürgen. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, 11 1997.

[15] Ioffe, Sergey and Szegedy, Christian. Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015.

[16] Karim, Fazle, Majumdar, Somshubra, Darabi, Houshang, and Harford, Samuel. Multivariate LSTM-FCNs for time series classification. *Neural Networks*, 116:237–245, aug 2019.

[17] Kendall, Alex, Grimes, Matthew, and Cipolla, Roberto. Posenet: A convolutional network for real-time 6-dof camera relocalization, 2016.

[18] Krizhevsky, Alex, Nair, Vinod, and Hinton, Geoffrey. Cifar-10 (canadian institute for advanced research).

[19] LeCun, Yann, Bottou, Léon, Bengio, Yoshua, and Haffner, Patrick. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, volume 86, pages 2278–2324, 1998.

[20] Lin, Zhouhan, Feng, Minwei, dos Santos, Cicero Nogueira, Yu, Mo, Xiang, Bing, Zhou, Bowen, and Bengio, Yoshua. A structured self-attentive sentence embedding, 2017.

[21] Liu, Ziyu, Zhang, Hongwen, Chen, Zhenghao, Wang, Zhiyong, and Ouyang, Wanli. Disentangling and unifying graph convolutions for skeleton-based action recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 143–152, 2020.

[22] Mazzia, Vittorio, Angarano, Simone, Salvetti, Francesco, Angelini, Federico, and Chiaberge, Marcello. Action transformer: A self-attention model for short-time pose-based human action recognition. *Pattern Recognition*, 124:108487, apr 2022.

[23] Nair, Vinod and Hinton, Geoffrey E. Rectified linear units improve restricted boltzmann machines. In *ICML 2010*, pages 807–814, 2010.

[24] Plizzari, Chiara, Cannici, Marco, and Matteucci, Matteo. Skeleton-based action recognition via spatial and temporal transformer networks. *Computer Vision and Image Understanding*, 208-209:103219, jul 2021.

[25] Rafi, Abdul Muntakim, Tonmoy, Thamidul Islam, Kamal, Uday, Wu, Q. M. Jonathan, and Hasan, Md. Kamrul. Remnet: Remnant convolutional neural network for camera model identification, 2020.

[26] Rosenblatt, F. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958.

[27] Rumelhart, David E., Hinton, Geoffrey E., and Williams, Ronald J. Learning internal representations by error propagation. In Rumelhart, David E. and Mcclelland, James L., editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Volume 1: Foundations*, pages 318–362. MIT Press, Cambridge, MA, 1986.

[28] Srivastava, Nitish, Hinton, Geoffrey E., Krizhevsky, Alex, Sutskever, Ilya, and Salakhutdinov, Ruslan. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[29] Szegedy, Christian, Liu, Wei, Jia, Yangqing, Sermanet, Pierre, Reed, Scott, Anguelov, Dragomir, Erhan, Dumitru, Vanhoucke, Vincent, and Rabinovich, Andrew. Going deeper with convolutions, 2014.

[30] Szeliski, Richard. *Computer Vision - Algorithms and Applications 2nd Edition.* Texts in Computer Science. Springer, 2022.

[31] Vaswani, Ashish, Shazeer, Noam, Parmar, Niki, Uszkoreit, Jakob, Jones, Llion, Gomez, Aidan N., Kaiser, Lukasz, and Polosukhin, Illia. Attention is all you need. *CoRR*, abs/1706.03762, 2017.

[32] Zhang, Aston, Lipton, Zachary C., Li, Mu, and Smola, Alexander J. Dive into deep learning, 2023.

Abstract

Human action recognition has long been an important problem in the field of computer vision and for the past 30 years (since the proposal of neural networks), there has been many ways of tackling this problem such as CNNs and RNNs. Recently the introduction of attention has brought a completely new way of looking at deep learning models. We introduce here, Action Transformer (AcT) a fully attention based transformer model for short-time human action recognition. We use Already developed pose estimation methods to feed to our network and we only focus on short time actions and for our dataset we use MPOSE2021, a large-scale dataset designed for benchmarking short time human action recognition. Lastly, we compare our model to other methods to show effectiveness of our discovery.

College of Science
School of Mathematics, Statistics, and Computer Science

Human Action Recognition (HAR) Using Deep Learning

Kasra Kakaee

Advisor: Bagher Babaali

A thesis submitted to Graduate Studies Office
in partial fulfillment of the requirements for the degree of
B.Sc. in
Computer Science

Summer 2023