Software 2.0

بعضی اوقات مشاهده می‌کنم که افراد به شبکه‌های عصبی به عنوان "یکی دیگر از ابزارهای موجود در جعبه ابزار یادگیری ماشین" اشاره می‌کنند. آن‌ها دارای برخی مزایا و معایب هستند، در بعضی م وارد کارایی بیشتری دارند و می‌توانید در رقابت‌های کاگل از آن‌ها بهره ببرید. با این حال، این تفسیر کاملاً از " دست دادن جنگل برای درختان است " ؟ . به عبارت دیگر، شبکه‌های عصبی فقط یک طبقه‌بندی‌کننده دیگر نیستند. آن‌ها نشان دهنده شروع یک تغییر اساسی در نحوه‌ی توسعه نرم‌افزار هستند .

"پشته (استک) کلاسیک" نرم‌افزار ۱.۰ نوعی از نرم‌افزار است که با آن آشنایی داریم. این نوع نرم‌افزار با استفاده از زبان‌هایی مانند پایتون، سی پلاس پلاس و غیره نوشته می‌شود. در این نوع نرم‌افزار، دستورهای صریحی که برنامه‌نویس برای کامپیوتر نوشته است، اجرا میشود . برنامه‌نویس با نوشتن هر خط کد، نقطه‌ای خاص در فضای برنامه تعیین می‌کند که می‌خواهد رفتار خاصی در آن نقطه اتفاق بیفتد.

به طور مقابل، نرم‌افزار ۲.۰ با زبانی بسیار انتزاعی و ناهموار برای انسان نوشته شده است، مانند وزن‌های یک شبکه عصبی. هیچ فردی در نوشتن این کد شرکت ندارد زیرا تعداد وزن‌های آن بسیار زیاد است (شبکه‌های معمولی ممکن است میلیون‌ها وزن داشته باشیم) و نوشتن کد به صورت مستقیم با استفاده از وزن‌ها سختی بسیاری دارد .

به جای آن، رویکرد ما این است که برخی از اهداف رفتاری برنامه مطلوب را مشخص کنیم (به عنوان مثال "برد در بازی Go")، یک الگوی اولیه از کد (به عبارتی یک معماری شبکه عصبی) را که یک زیرمجموعه از فضای برنامه‌ها را برای جستجو مشخص می‌کند، نوشته و از منابع محاسباتی در دسترس خود برای جستجوی این فضای برنامه‌ها برای یافتن یک برنامه کارآمد استفاده می‌کنیم. در مورد شبکه‌های عصبی، جستجو را تنها در یک زیرمجموعه پیوسته از فضای برنامه‌ها محدود می‌کنیم که در آن فرآیند جستجو با استفاده از الگوریتم‌های پس‌انتشار خطا و کاهش گرادیان تصادفی به نحوی که (به طور نسبتاً شگفت‌انگیزی) کارآیی آن تضمین شده باشد، انجام می‌شود.

برای بیان این تشبیه، در نرم‌افزار ۱.۰، سورس کدهای طراحی شده توسط انسان (مانند برخی فایل‌های .cpp) به یک فایل باینری که کار مفیدی انجام می‌دهد، تبدیل می‌شوند. در نرم‌افزار ۲.۰، سورس کد بیشتر شامل ۱) مجموعه داده‌ای است که رفتار مطلوب را تعریف می‌کند و ۲) معماری شبکه عصبی است که الگوی اولیه کد را ارائه می‌دهد، اما بسیاری از جزئیات (وزن‌ها) باید تکمیل شوند. فرآیند آموزش شبکه عصبی، مجموعه داده را به باینری (شبکه عصبی نهایی) تبدیل می‌کند. در بیشتر برنامه‌های کاربردی امروزی، معماری‌های شبکه عصبی و سیستم‌های آموزش در حال استانداردسازی به عنوان کالایی عرضه می‌شوند، بنابراین بیشتر "توسعه نرم‌افزار" به شکل تأمین و توسعه مجموعه داده‌های برچسب‌خورده، انجام می‌شود. این تغییر در فراهم آوری پارادایم برنامه‌نویسی را بنیادی تغییر می‌دهد، زیرا تیم‌ها به دو بخش تقسیم می‌شوند: برنامه‌نویسان ۲.۰ (برچسب‌گذار های داده) مجموعه

داده‌ها را ویرایش و گسترش می‌دهند، در حالی که تعداد کمی از برنامه‌نویسان ۱.۰ برای حفظ و توسعه زیرساخت کد آموزش، تجزیه و تحلیل، تجسم و رابط‌های برچسب‌گذاری، فعالیت می‌کنند.
در واقع، یک بخش بزرگی از مسائل واقعی جهان، دارای ویژگی هایی هستند که جمع‌آوری داده (یا به طور کلی شناسایی رفتار مطلوب) در آن‌ها به طور قابل توجهی آسان‌تر است نسبت به نوشتن برنامه. به همین دلیل این مسئله و مزایای بسیار دیگر نرم‌افزار ۲.۰ که در ادامه به آن‌ها خواهم پرداخت، در صنعت شاهد یک انتقال عظیم هستیم که بسیاری از کدهای ۱.۰ به کدهای ۲.۰ منتقل می‌شود. نرم‌افزار (۱.۰) دنیا را می‌خورد و حالا هوش مصنوعی (نرم‌افزار ۲.۰) نرم‌افزار را می‌خورد.
بیایید به طور خلاصه به برخی از مثال‌های ملموس این انتقال پایدار بپردازیم. در هر یک از این حوزه‌ها، در چند سال گذشته بهبودهایی را دیدیم که زمانی به دنبال رهایی از سعی در حل یک مسئله پیچیده با نوشتن کد بوده‌ایم، و به جای آن، کد را به مجموعه نرم‌افزار ۲.۰ منتقل کرده‌ایم.

تشخیص تصویر در گذشته شامل ویژگی‌های طراحی شده با یک کمی یادگیری ماشین در انتها بود (مانند یک SVM). از آن زمان به بعد، با جمع‌آوری مجموعه‌های داده بزرگ (مانند ImageNet) و جستجو در فضای معماری‌های شبکه‌های عصبی کانولوشنی، ویژگی‌های تصویری قدرتمندتری کشف کرده‌ایم. اخیراً، حتی به خودمان اعتماد نداریم که معماری‌ها را به صورت دستی کد کنیم و شروع به جستجو در آن‌ها کرده‌ایم.

تشخیص گفتار در گذشته شامل بسیاری از پیش‌پردازش‌ها، مدل‌های مخلوط گاوسی و مدل‌های مخفی مارکوف بود، اما امروزه تقریباً به طور کامل از موارد شبکه‌های عصبی تشکیل شده است. یک نقل قول بسیار مرتبط و طنزآمیز که معمولاً به فرد جلینک از سال ۱۹۸۵ نسبت داده می‌شود، به شرح زیر است: "هر بار که یک زبان‌شناس را اخراج می‌کنم، عملکرد سیستم تشخیص گفتار ما بهبود می‌یابد".

تولید گفتار در گذشته با روش‌های مختلف انجام شده بود، اما امروزه مدل‌های حالت هنری بزرگ (مانند WaveNet) با استفاده از شبکه‌های عصبی کانولوشنی (ConvNets) بزرگ، که خروجی سیگنال صوتی خام تولید می‌کنند، به عنوان بهترین روش شناخته می‌شوند.

در گذشته، ترجمه ماشینی با استفاده از تکنیک‌های آماری مبتنی بر عبارات انجام می‌شد، اما شبکه‌های عصبی به سرعت به عنوان روش اصلی شناخته شده‌اند. معماری‌های مورد علاقه من در محیط چندزبانه آموزش داده می‌شوند، به طوری که یک مدل ترجمه از هر زبان مبدأ به هر زبان مقصد را انجام می‌دهد، و در محیط‌های نیمه‌نظارتی (یا به طور کامل بدون نظارت) آموزش داده می‌شوند.

در بازی‌ها، بازی هایی مانند Go که به صورت سنتی (دستی) کد شده اند (نوشته شده) برای مدت طولانی توسعه داده شده‌اند، اما AlphaGo Zero (یک ConvNet که وضعیت خام صفحه بازی را مشاهده کرده و حرکتی را انجام می‌دهد) به دور از شکست، قوی‌ترین بازیکن این بازی شده است. من انتظار دارم که نتایج بسیار مشابهی را در حوزه‌های دیگری مانند DOTA 2 یا StarCraft ببینیم.

پایگاه‌داده‌ها. سیستم‌های سنتی خارج از هوش مصنوعی نیز نشانه‌های اولیه انتقال را مشاهده می‌کنند. به عنوان مثال، "دلایل استفاده از ساختارهای فهرست یادگیری شده"، اجزای اصلی یک سیستم مدیریت داده را با یک شبکه عصبی جایگزین می‌کند که با عملکرد بهینه‌سازی حافظه فهرست B-Tree، به حداکثر ۷۰٪ سرعت بیشتری را ارائه می‌دهد و همچنین یک تغییر در شرایط محیطی را نیز به دنبال دارد.

شما متوجه خواهید شد که بسیاری از پیوندهایی که در بالا ذکر کردم، شامل کارهایی است که در شرکت گوگل انجام شده‌اند. این به دلیل این است که گوگل در حال حاضر در راس این حرکت برای بازنویسی بخش‌های بزرگ خود به کد نرم‌افزار ۲.۰ قرار دارد. "یک مدل برای حکمرانی بر همه" یک طرح اولیه از آنچه که ممکن است به نظر برسد، را ارائه می‌دهد، جایی که قدرت آماری دامنه‌های مختلف به یک تفهیم همسان از دنیا تلفیق می‌شود.

The benefits of Software 2.0

چرا باید برنامه‌های پیچیده را به نرم‌افزار ۲.۰ منتقل کنیم؟ به طور واضح، یک پاسخ آسان این است که در عمل بهتر کار می‌کنند. با این حال، دلایل متعددی دیگر برای ترجیح این پشته (استک) وجود دارد. بیایید به برخی از مزایای نرم‌افزار ۲.۰ (مانند: یک ConvNet) نسبت به نرم‌افزار ۱.۰ (مانند: یک پایگاه کد C++ تولیدی) نگاهی بیندازیم. نرم‌افزار ۲.۰ عبارت است از:

- ساختار بسیار ساده‌تر: با استفاده از شبکه‌های عصبی، به جای نوشتن کد پیچیده و توابع پیچیده، می‌توانیم با ساختار ساده‌تری که توسط شبکه‌های عصبی پوشش داده شده است، مسئله را برطرف کنیم.
- قابلیت تعمیم‌پذیری: شبکه‌های عصبی به خوبی در تعمیم‌پذیری به مسائل مشابه عمل می‌کنند، به خصوص در برابر داده‌های جدید.
- توانایی یادگیری خودکار: شبکه‌های عصبی می‌توانند با دادن داده‌های بیشتر، خودکاراً یاد بگیرند و کیفیت خود را بهبود ببخشند.
- سرعت بسیار بالا: شبکه‌های عصبی می‌توانند در سرعت بسیار بالایی اجرا شوند، به خصوص در مقایسه با الگوریتم‌های سنتی مانند الگوریتم‌های محاسباتی غیرخطی.
- مقیاس‌پذیری بالا: شبکه‌های عصبی به خوبی به محیط‌های بزرگ مقیاس‌پذیری می‌کنند و به سادگی می‌توانند بر روی سیستم‌های توزیع‌شده اجرا شوند.

یکنواختی محاسباتی. یک شبکه عصبی معمول، در درجه اول، از دو عمل ماتریسی‌ضرب و یک عمل ترشولدینگ در صفر (ReLU) تشکیل شده است. این را با مجموعه دستورات نرم‌افزار کلاسیک مقایسه

کنید که بسیاری نه تنها نامنظم و پیچیده است، بلکه متنوع است. به دلیل اینکه شما تنها باید پیاده‌سازی نرم‌افزار ۱.۰ برای تعداد کمی از اصول محاسباتی مرکزی (به عنوان مثال، ضرب ماتریسی) را فراهم کنید، بسیار آسان‌تر است که گارانتی‌های مختلف درستی/عملکرد را ارائه دهید.

به عنوان یک نتیجه، از آنجایی که مجموعه دستورات یک شبکه عصبی نسبتاً کوچک است، پیاده‌سازی این شبکه‌ها به سمت سیلیکون بسیار آسان‌تر است، به عنوان مثال با استفاده از ASICهای سفارشی، چیپ‌های neuromorphic و غیره. دنیا زمانی تغییر می‌کند که هوش با مصرف کمتر به دور ما پراکنده می‌شود. به عنوان مثال، چیپ‌های کوچک و ارزان می‌توانند با یک ConvNet پیش‌آموزش‌دیده، یک شناسایی‌کننده گفتار و یک شبکه ترکیب گفتار WaveNet همگرا شوند و همگی در یک پروتوبرین کوچکی که می‌توانید به چیزی متصل کنید، یکپارچه شوند.

زمان اجرای ثابت. هر بار اجرای یک عبور جلویی (forward pass) از یک شبکه عصبی معمولی دقیقاً همان تعداد FLOPS را می‌طلبد. هیچ گونه متغیری بر اساس مسیرهای اجرایی مختلف کد شما از طریق یک پایگاه کد C++ پراکنده وجود ندارد. البته می‌توانید گراف‌های محاسباتی پویا داشته باشید، اما جریان اجرا در اغلب موارد به شدت محدود است. به این ترتیب، تقریباً تضمین می‌شود که هرگز در حلقه‌های بی‌انتهای ناخواسته قرار نخواهیم گرفت.

استفاده ثابت از حافظه. با توجه به موارد فوق، در هیچ نقطه‌ای حافظه‌ای به‌صورت پویا تخصیص داده نمی‌شود، بنابراین احتمال کمتری برای مبادله با دیسک یا نشتی حافظه وجود دارد که شما باید در کد خود جستجو کنید.

قابل حمل بودن بسیار بالا. دنباله‌ای از ضرب ماتریسی به طور قابل توجهی آسان‌تر در تنظیمات محاسباتی دلخواه اجرا می‌شود، نسبت به باینری‌ها و اسکریپت‌های کلاسیک.

بسیار چابک است. اگر کد C++ داشته باشید و کسی از شما بخواهد که سرعت آن را دو برابر کنید (با هزینه عملکرد اگر لازم باشد)، به طور قابل توجهی غیرممکن است که سیستم را برای مشخصات جدید آماده کنید. با این حال، در نرم‌افزار ۲.۰ می‌توانیم شبکه خود را بگیریم، نصف کانال‌ها را حذف کنیم، دوباره آموزش دهیم و در نتیجه، آن را با دقت کمتری دو برابر سرعت اجرا کنیم. این یک جادوست. به علاوه، اگر اتفاقی دیتا/محاسبات بیشتری در اختیار داشته باشید، با اضافه کردن کانال‌های بیشتر و دوباره آموزش دادن، می‌توانید برنامه خود را بهبود بخشید.

ماژول‌ها می‌توانند به یک کل بهینه تبدیل شوند. نرم‌افزار ما اغلب به ماژول‌هایی تجزیه می‌شود که از طریق توابع عمومی، APIها یا نقاط پایانی (gateways) ارتباط برقرار می‌کنند. با این حال، اگر دو ماژول Software 2.0 که در ابتدا به صورت جداگانه آموزش داده شده‌اند، با یکدیگر تعامل کنند، می‌توانیم به راحتی از طریق پشت‌اندازی مجدد به سراسر سیستم برگردیم. فکر کنید چقدر شگفت‌انگیز

است که مرورگر وب شما بتواند به‌طور خودکار دستورالعمل‌های سیستم سطح پایین ۱۰ استک پایین‌تر را به‌صورت مجدد طراحی کند تا در بارگذاری صفحات وب با کارایی بالاتری روبرو شود. یا اگر کتابخانه دید کامپیوتری (مانند OpenCV) که وارد کرده‌اید بر روی داده‌های خاص شما به‌صورت خودکار تنظیم شود. با ۲.۰، این رفتار پیش‌فرض است.

نرم افزار ۲.۰ بهتر از شماست. سرانجام و بیشترین اهمیت، یک شبکه عصبی بهترین قطعه کد است که شما یا من می‌توانیم در یک قسمت قابل توجهی از عمودهای ارزشمند، که در حال حاضر حداقل با هر چیزی مرتبط با تصاویر/ویدئو و صدا/گفتار است، بسازیم.

The limitations of Software 2.0

همچنین، پشته ۲.۰ دارای برخی معایب است. در پایان بهینه‌سازی، ما با شبکه‌های بزرگی با کارایی بالا تنها می‌مانیم، اما بسیار سخت است که بفهمیم دقیقاً چگونه کار می‌کنند. در طول بسیاری از حوزه‌های کاربردی، ما به انتخاب بین استفاده از یک مدل با دقت ۹۰٪ که آن را درک می‌کنیم یا یک مدل با دقت ۹۹٪ که آن را درک نمی‌کنیم، می‌رسیم.

پشته ۲.۰ ممکن است به شکل‌های غیرقابل پیش‌بینی و حتی خجالت‌آوری شکست خورد، یا بدتر، ممکن است به طور «بی‌صدا» شکست خورده باشد، به عنوان مثال با پذیرش بی‌صدا از تعصبات در داده‌های آموزشی، که بسیار سخت است که در اغلب موارد با اندازه‌های میلیونی به درستی تجزیه و تحلیل شوند و بررسی شوند.

در نهایت، ما هنوز در حال کشف برخی از خاصیت‌های عجیب این پشته هستیم. به عنوان مثال، وجود مثال‌ها و حملات معانضی نشان می‌دهد که طبیعت غیرقابل پیش‌بینی این پشته چیزی است که باید با آن مواجه شویم.

Programming in the 2.0 stack

نرم افزار 1.0 همان برنامه هایی است که ما می‌نویسیم. اما نرم افزار 2.0 یک برنامه است که توسط یک فرایند بهینه‌سازی، براساس معیارهایی مانند "تشخیص درست داده‌های آموزشی" نوشته شده است.

شاید در هر موقعیتی که برنامه در آن روشن نباشد، اما می‌توانیم عملکرد آن را تکرار کنیم (مانند تشخیص درست تصاویر یا برنده شدن در بازی‌های گو)، به این گذار از نرم افزار دچار شویم. به دلیل اینکه بهینه‌سازی می‌تواند کد بهتری از آنچه انسان می‌تواند بنویسد پیدا کند.
درک روندها از این طریق لنز مشخصی نیاز دارد. اگر شما Software 2.0 را به عنوان یک پارادایم برنامه‌نویسی جدید و در حال ظهور تشخیص دهید، به جای ساده نگرفتن شبکه‌های عصبی به عنوان یک طبقه‌بند خوب در رده‌ی تکنیک‌های یادگیری ماشین، استنتاجات بیشتری قابل پیش‌بینی خواهد بود و روش کار برای انجام بیشتر کارها بهتر مشخص می‌شود.
به طور خاص، ما یک میزان بسیار زیادی از ابزارها را برای کمک به انسان‌ها در نوشتن کد 1.0 ساخته‌ایم، مانند محیط توسعه یکپارچه (IDE)های قدرتمند با ویژگی‌هایی مانند برجسته‌سازی سینتکس، ابزارهای دیباگ، پروفایلرها، رفتن به تعریف، ادغام گیت و غیره. در مجموعه‌ی 2.0، برنامه‌نویسی با جمع‌آوری، بازیابی و پاک‌سازی مجموعه‌داده‌ها انجام می‌شود. به عنوان مثال، اگر شبکه در برخی موارد سخت یا نادر شکست بخورد، ما آن پیش‌بینی‌ها را با نوشتن کد نمی‌توانیم تعمییر کنیم، بلکه با اضافه کردن مثال‌های بیشتر برچسب‌گذاری شده از آن موارد، مشکل را حل می‌کنیم. کی اولین IDEهای Software 2.0 را توسعه خواهد داد که در کلیه‌ی جریان‌های جمع‌آوری، تصویرسازی، پاک‌سازی، برچسب‌گذاری و منبع‌یابی مجموعه‌داده‌ها کمک کند؟ شاید IDE تصاویری را که شبکه براساس از دست دادن در هر مثال، اشتباه برچسب‌گذاری شده است، بیاورد، یا با استفاده از پیش‌بینی‌ها در برچسب‌گذاری کمک کند، یا مثال‌های مفیدی برای برچسب‌گذاری بر اساس عدم اطمینان پیش‌بینی شبکه پیشنهاد دهد.

به طور مشابه، Github برای کد Software 1.0 یک محل بسیار مهم است. آیا برای Software 2.0 چنین فضایی وجود خواهد داشت ؟ در این حالت، مخازن، مجموعه‌داده‌ها هستند و commitها از افزودن و ویرایش برچسب‌ها تشکیل شده‌اند.
سامانه‌های مدیریت بسته‌های سنتی و زیرساخت‌های مربوطه مانند docker، conda، pip و غیره به ما در استقرار و ترکیب باینری‌ها به طور آسان‌تر کمک می‌کنند. چطور می‌توانیم باینری‌های Software 2.0 را به صورت موثر استقرار، به اشتراک بگذاری، وارد کردن و با آن کار کنیم؟ معادل conda برای شبکه‌های عصبی چیست؟
در دوره کوتاه مدت، Software 2.0 در هر حوزه‌ای که ارزیابی تکراری ممکن و ارزان باشد و الگوریتم خود به صراحت طراحی شود، به طور فزاینده‌ای پراکنده خواهد شد. در حال حاضر، فرصت‌های جالبی برای بررسی کل اکوسیستم توسعه نرم‌افزار و نحوه سازگاری آن با این پارادایم برنامه‌نویسی جدید وجود دارد. و در آینده، آینده این پارادایم روشن است زیرا واضح است که هنگامی که AGI را توسعه دهیم، حتماً با استفاده از Software 2.0 نوشته خواهد شد.

keywords

Accessible	Inaccessible

neural networks	silicon
machine learning	
Artificial Intelligence	
training data	
predictions	
computer vision	
Speech recognition	
optimization	
backpropagation	
unsupervised	
multilingual setting	
classical stack	
Software 2.0	
neural net architecture	
labeled datasets	
Visual Recognition	
SVM	
Speech synthesis	
stitching mechanisms	
ConvNets	
WaveNet	
order-of-magnitude	
cache-optimized	

Computationally homogeneous	
matrix multiplication	
thresholding at zero (ReLU)	
ASICs, neuromorphic chips	
FLOPS	
dynamic compute	
agile	
backpropagate	
AGI	
mislabeled	

Links

Title	Link	Accessible
Searching over those	https://arxiv.org/abs/1703.01041	Yes
today	https://github.com/syhw/wer_are_we	Yes
WaveNet	https://deepmind.com/blog/wavenet-launches-google-assistant/	Yes
Multilingual setting	https://arxiv.org/abs/1611.04558	Yes
unsupervised	https://arxiv.org/abs/1710.11041	Yes
The Case for Learned Index Structure	https://arxiv.org/abs/1712.01208	Yes

One Model to Rule them all	https://arxiv.org/abs/1706.05137	Yes
ASICs neuromorphic chips	https://www.forbes.com/sites/moorinsights/2017/08/04/will-asic-chips-become-the-next-big-thing-in-ai/#7d6d7c0511d9	Yes
Fail in unintuitive and embarrassing ways	https://motherboard.vice.com/en_us/article/nz7798/weve-already-taught-artificial-intelligence-to-be-racist-sexist	Yes
Adversarial examples	https://blog.openai.com/adversarial-example-research/	Yes
Attacks	https://github.com/yenchenlin/awesome-adversarial-machine-learning	Yes

Notes

In regards to Software 2.0, it is noteworthy that there are limited resources available. Through diligent research, I have discovered that the primary source of information on this topic is a series of three-hour videos presented by Andrej Karpathy , who is credited with introducing the concept of Software 2.0.

Link to videos : https://www.youtube.com/watch?v=cdiD-9MMpb0
https://www.youtube.com/watch?v=y57wwucbXR8

Due to the novelty of this topic, I have encountered significant difficulty in locating relevant resources. Regrettably, there appears to be a paucity of articles or other publications available on the subject at this time.