به نام خداوند جان و خرد

دانشگاه تهران

دانشکدگان فنی

دانشکده مهندسی برق و کامپیوتر

سامانه‌های یادگیری‌ماشین توزیع شده

تمرین نوشتاری ۱

نام و نام خانوادگی: علی خرم فر

شماره دانشجویی: ۸۱۰۱۰۲۱۲۹

آذرماه ۱۴۰۳

فهرست مطالب

فهرست جداول

فهرست اشکال

۱_ پاسخ سوال شماره ۱

۱-۱_ مرور سخت‌افزارها

در درس با سخت‌افزارهای زیر آشنا شدیم که برای جبر خطی مناسب بودند ولی قبل از رواج یادگیری ماشین توسعه داده شده بودند:

Systolic Arrays

این سخت‌افزار برای انجام محاسبات ماتریسی و خصوصا ضرب طراحی شد و با سرعت بسیار بیشتری نسبت به CPU عادی می‌تواند عملیات ضرب ماتریسی را انجام دهد.

GPU

کارت‌های گرافیک که اول برای بازی‌ها و کارهای گرافیکی ساخته شده بودند، اما به خاطر توانایی بالایی که در پردازش موازی دارند، در محاسبات جبر خطی هم کاربرد دارند.

Intel SIMD Extensions

اینتل مجموعه‌ای از فناوری‌ها مثل MMX، SSE، AVX و AVX-512 را ارائه داد که برای انجام سریع‌تر محاسبات جبر خطی و ماتریسی روی پردازنده‌هایش استفاده می‌شدند.

نکته:

TPU جز این لیست نیست، چون به طور خاص برای یادگیری ماشین توسط گوگل ساخته شد و قبل از رواج یادگیری ماشین وجود نداشته است.

۱-۲_ کاربردهای غیر از یادگیری ماشین

پردازش تصویر و ویدئو: GPU به دلیل توانایی بالا در پردازش موازی، برای افزایش سرعت در پردازش تصاویر و ویدئوها به کار می‌رود. برای نمونه در نرم‌افزارهای پردازش تصاویر مثل Adobe Photoshop استفاده از GPU برای اعمال تغییرات مثل اعمال فیلتر درتصاویر ضروری است و اعمال برخی فیلترها با کمک CPU امکان‌پذیر نیست و اگر هم امکان‌پذیر باشد زمان اعمال آن بسیار بیشتر از GPU خواهدبود. همچنین در گذشته برخی مادربردها خروجی تصویر نداشتند و برای تبدیل سیگنال‌های دیجیتال به آنالوگ و گرفتن خروجی تصویر از کامپیوترها از کارت‌های گرافیک استفاده می‌شد. همچنین پخش ویدئو در کامپیوترهای قدیمی‌تر هم به دلیل CPU های ضعیف چالش‌هایی وجود داشت. [1]

البته این نکته هم باید ذکر شود که در معماری SIMD اینتل، عملیات Frame Difference برای مقایسه دو تصویر و استخراج تفاوت‌های آن‌ها به کار می‌رود. این روش در پردازش ویدئو، تشخیص حرکت و

فشرده‌سازی تصویر کاربرد دارد. با استفاده از دستورالعمل‌های SIMD، می‌توان به‌طور هم‌زمان چندین پیکسل را پردازش کرد و تفاوت آن‌ها را محاسبه نمود و نیازی به ذخیره تمام اطلاعات هر فریم در یک ویدئو نیست.

بازی‌های رایانه‌ای: GPU نقش کلیدی در رندرکردن محیط گرافیکی پیچیده در بازی‌ها داشته و یکی دلایل موفقیت شرکت NVIDIA همین مورد است. با پیشرفت GPU ها بازی‌ها هم پیشرفت زیادی داشتند و با قدرت پردازشی آن‌ها محیط بازی به صورتReal-Time رندر می‌شد. خصوصا برای بازی‌های OpenWorld این مورد بسیار کاربردی است. یکی از نیازهای اصلی بازیکنان، روان بودن بازی است که با نرخ فریم بالا محقق می‌شود. با پیشرفت GPUها ، امکان اجرای بازی‌ها با نرخ فریم بالا هم فراهم شد.همچنین تکنیک‌های پیشرفته‌ای برای شبیه‌سازی دقیق‌تر نورپردازی، بازتاب‌ها و سایه‌ها استفاده می‌شود، که به واقع‌گرایی بیشتر در بازی‌ها کمک می‌کند.

پردازش سیگنال دیجیتال

در پردازش سیگنال دیجیتال (Digital Signal Processing)، عملیات‌هایی مانند فیلتر دیجیتال، تبدیل فوریه سریع (Fast Fourier Transform) و Pattern matching نیازمند انجام محاسبات ماتریسی و برداری هستند.استفاده از Systolic Arrays با و قابلیت پردازش موازی، به انجام این محاسبات را با سرعت بالا کمک می‌کنند. [2]

انجام محاسبات و شبیه‌سازی‌های مهندسی

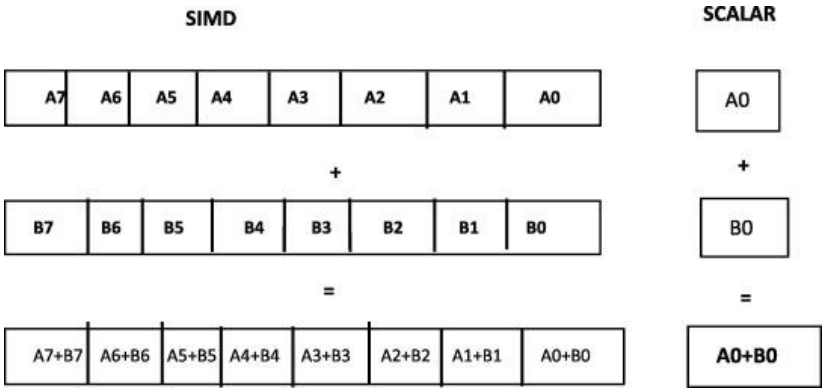
استفاده از توسعه‌های SIMD اینتل در شبیه‌سازی‌های علمی و مهندسی، با افزایش پهنای باند پردازش با کاهش زمان محاسبات و افزایش دقت نتایج، به مهندسان کمک می‌کند تا مدل‌های پیچیده‌تری را با کارایی بالاتر شبیه‌سازی کنند. برای نمونه مقاله‌ی [3] به بررسی بهینه‌سازی محاسبات Lennard-Jones Potential با استفاده از AVX2 و AVX-512 پرداخته است که چگونه بهره‌گیری از این توسعه‌ها می‌تواند سرعت و کارایی شبیه‌سازی‌های مولکولی را افزایش دهد.

الگوریتم‌های رمزنگاری

توسعه‌های SIMD اینتل با ارائه قابلیت‌های پردازش موازی، امکان اجرای سریع‌تر و کارآمدتر عملیات‌های رمزنگاری را فراهم می‌کنند. به‌عنوان مثال در مقاله‌ای که توسط اینتل منتشرشده به بررسی استفاده از توسعه‌های SIMD اینتل برای بهینه‌سازی محاسبات الگوریتم‌های رمزنگاری پرداخته و نشان می‌دهد که چگونه استفاده از این دستورالعمل‌ها می‌تواند سرعت و کارایی الگوریتم‌های رمزنگاری را افزایش دهد. [4]

۲_پاسخ سوال شماره ۲

SISD (پردازش اسکالر) فقط یک عملیات روی یک داده را در هر چرخه انجام می‌دهد. در مقابل، SIMD (مثل AVX) چندین عملیات را به‌طور هم‌زمان روی چندین داده اجرا می‌کند. برای این سوال مقاله [5] بررسی شد.

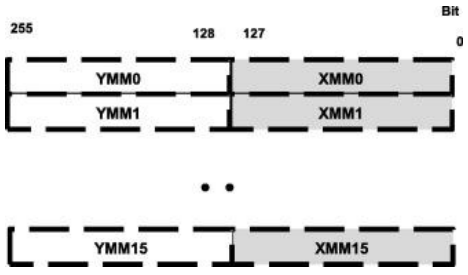


شکل ۱-۲ مقایسه پردازش SCALAR و SIMD [5]

در پردازنده‌های جدید، واحدهای SIMD و SISD معمولاً از واحدهای محاسبه و منطق (ALU) جداگانه‌ای استفاده می‌کنند. این تفکیک به پردازنده کمک می‌کند تا به‌طور هم‌زمان عملیات برداری را با کارایی بالاتری انجام دهد.

پردازنده‌های سری Intel Core

در پردازنده‌های سری Intel Core، افزونه‌های SIMD مانند SSE[1] و AVX[2] معرفی شده‌اند. این توسعه‌ها از واحدهای اختصاصی استفاده می‌کنند. به‌عنوان مثال، در معماری‌های دارای AVX، واحدهای برداری ۲۵۶ بیتی وجود دارند که می‌توانند به‌طور هم‌زمان چندین داده را پردازش کنند.



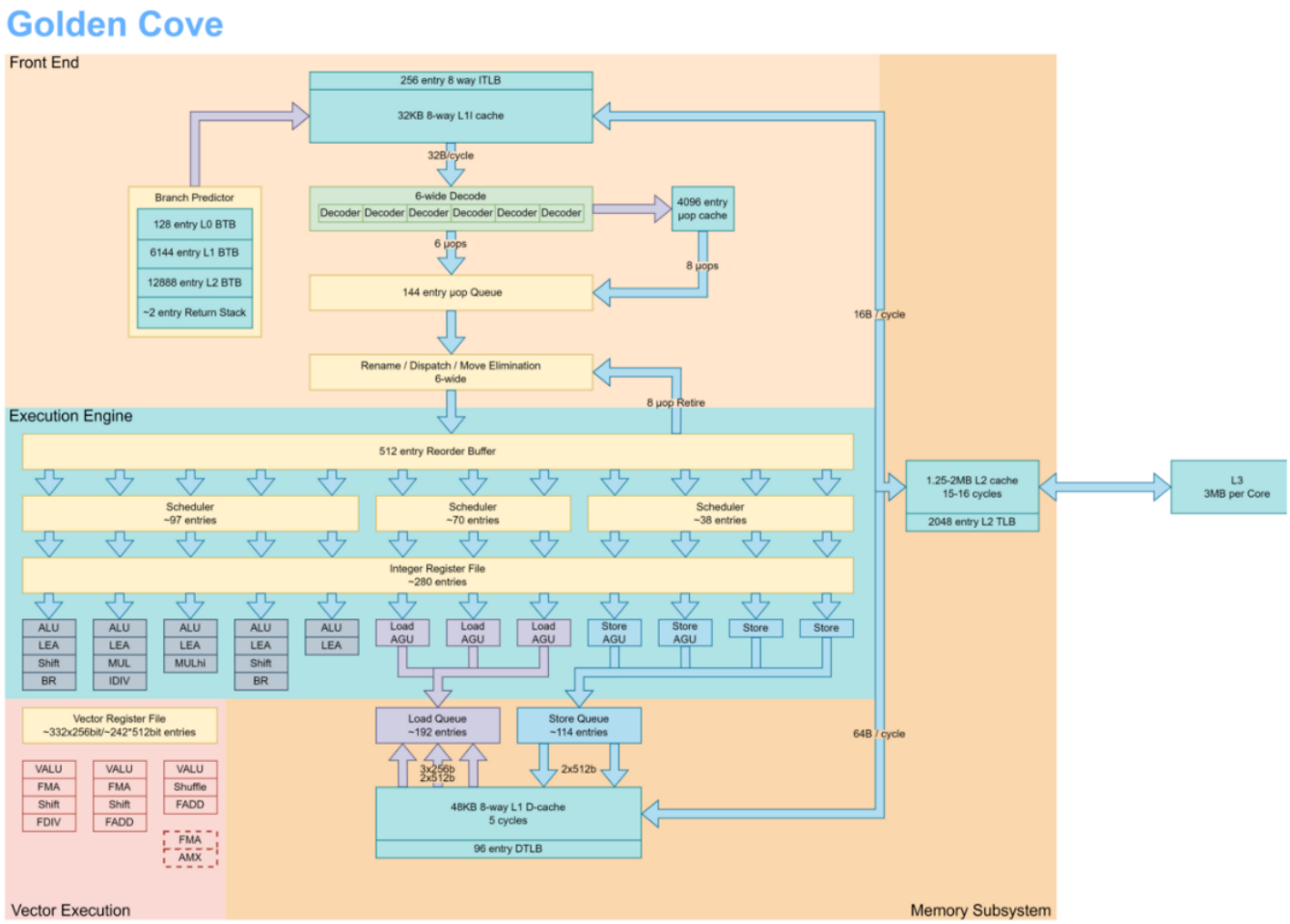
شکل ۲-۲ رجیسترهای ۲۵۶ بیتی SIMD [5]

[1] Streaming SIMD Extensions
[2] Advanced Vector Extensions

مثال پردازنده: در پردازنده‌های اینتل Core i7، AVX از ۱۶ ثبات ۲۵۶ بیتی (YMM0-YMM15) استفاده می‌کند. این ثبات‌ها برای عملیات برداری طراحی شده‌اند و با معماری SISD هم‌پوشانی ندارند. SISD فقط یک عملیات روی یک داده را در هر سیکل انجام می‌دهد. در مقابل، SIMD چندین عملیات را به‌طور هم‌زمان روی چندین داده اجرا می‌کند. مثال ارائه‌شده در شکل نشان می‌دهد که AVX می‌تواند ۸ عنصر داده (۳۲ بیتی) یا ۴ عنصر داده (۶۴ بیتی) را در یک سیکل پردازش کند، در حالی که SISD این کار را فقط برای یک عنصر انجام می‌دهد.

پردازنده‌های با معماری GoldenCove

در پردازنده‌های جدید مثل Golden Cove، واحدهای محاسباتی برداری (برای اجرای SIMD) معمولاً از مسیرهای محاسباتی SISD جدا هستند. که این موضوع را در شکل زیر مشاهده می‌کنیم



شکل ۳-۲ دیاگرام پردازنده های Intel's Golden Cove

در Golden Cove، Vector Execution بخشی مجزا در Execution Engine دارد. این بخش شامل رجیسترهای برداری و واحدهای محاسباتی مثل FMA، Shuffle و Floating Point Add/Divide است که

مخصوص عملیات برداری هستند. بخش SISD از مسیرهای محاسباتی عمومی تشکیل شده و برای اجرای دستورات غیر برداری استفاده می‌شوند. رجیسترهای برداری از رجیسترهای عمومی مثلا برای اعداد صحیح جدا هستند. دستورات SIMD از طریق Vector Executionاجرا می‌شوند ولی دستورات SISD در واحدهای عمومی محاسباتی مثل Integer ALU اجرا می‌شوند.

پردازنده‌های مبتنی بر Golden Cove از دستورالعمل‌های AVX-512 پشتیبانی می‌کنند. AVX-512 شامل افزونه‌هایی مثل AVX512-FP16 برای بهینه‌سازی محاسبات اعشاری ۱۶ بیتی Half-Precision است. یکی از ویژگی‌های جدید در این معماری، پشتیبانی از AVX-VNNI یا Vector Neural Network Instructions است که احتمالا برای کاربردهای شبکه عصبی استفاده می‌شود به دلیل محدودیت زمان تمرین مطالب مربوط به این افزونه مطالعه نشد.

تصویر و اطلاعات بالا از صفحه ویکی‌پدیا مربوط به این معماری دریافت شده است.

۳_ پاسخ سوال شماره ۳

ابتدا یک مقایسه بر روی هر ۳ روش داشته باشیم. برای پاسخ به این سوال و جدول زیر از مقاله [6] استفاده شده است.

Float16	Float32	Optimized for	Comm. Functions	Backend
No	Yes	CPU, GPU	All	MPI
Yes	Yes	CPU	All (on CPU), broadcast & all-reduce (on GPU)	GLOO
Yes	Yes	GPU only	broadcast, all reduce, reduce and all gather (on GPU)	NCCL

جدول ۱ مقایسه کتابخانه‌های MPI, GLOO و NCCL

۳-۱ _MPI

نوع ارتباط: برای ارتباطات CPU و GPU مناسب است.

مزایا: عملکرد بهتر در پردازش تنسورهای کوچک.

محدودیت‌ها: عدم پشتیبانی از نوع داده‌ی Float16.

کاربرد: در ارتباطات CPU محور و همچنین زمانی که اندازه تنسور کوچک باشد، مفید است.

۳-۲ _Gloo

نوع ارتباط: از ارتباطات CPU و GPU پشتیبانی می‌کند.

مزایا: پشتیبانی از نوع داده‌ی Float16 روی CPU.

محدودیت‌ها: حساس به افزایش تعداد workers. زمان در کلاسترهای بزرگ‌تر افزایش می‌یابد.

کاربرد: زمانی که نیاز به پردازش Float16 روی CPU باشد.

NCCL _۳-۳

نوع ارتباط: مختص GPU و بهینه‌شده برای ارتباطات بین GPUها.

مزایا: بهترین عملکرد در پردازش تنسورهای بزرگ، پشتیبانی کامل از Float16 و Float32.

محدودیت‌ها: تنها روی GPU کار می‌کند.

کاربرد: در سیستم‌هایی که از GPU استفاده می‌کنند و نیاز به پردازش سریع تنسورهای بزرگ دارند.

به طور کلی باتوجه به موارد بالا برای تنسورهای کوچک MPI بهترین انتخاب است، خصوصاً زمانی که کلاستر شامل تعداد زیادی Worker نباشد. برای CPU با نوع داده Float16 Gloo مزیت دارد و باید از آن استفاده شود. برای GPU و تنسورهای بزرگ NCCL به دلیل عملکرد برتر، بهترین گزینه است. البته باید توجه داشته باشیم که Pytorch به صورت پیشفرض فقط NCCL و Gloo را نصب می‌کند.

۴_پاسخ سوال شماره ۴

NVLink و NVSwitch دو فناوری پیشرفته از شرکت NVIDIA هستند که به‌طور خاص برای رفع محدودیت‌های پهنای باند و تأخیر در ارتباطات بین GPUها طراحی شده‌اند. فناوری‌های NVLink و NVSwitch به‌عنوان اجزای اصلی ارتباطات پرسرعت در بین GPUها طراحی شده‌اند تا محدودیت‌های پهنای باند و تأخیر ارتباط را رفع کنند.

۴-۱_ مقایسه NVLink و NVSwitch

NVLink یک پروتکل ارتباط مستقیم GPU-to-GPU است که پهنای باندی بالا و تأخیر بسیار کم فراهم می‌کند. نسل پنجم NVLink، که با معماری NVIDIA Blackwell همراه است، پهنای باندی معادل ۱.۸ ترابایت بر ثانیه برای هر GPU ارائه می‌دهد که این میزان ۱۴ برابر پهنای باند PCIeنسخه پنجم است.

NVSwitch به‌عنوان یک فناوری مبتنی بر استفاده از NVLink، امکان اتصال تعداد زیادی GPU را در یک محیط دارای چند نود فراهم می‌کند. این فناوری از ۱۴۴ پورت NVLink با پهنای باند ۱۴.۴ ترابایت بر ثانیه استفاده می‌کند که در سرورهای NVIDIA GB200 NVL72 می‌تواند تا ۵۷۶ GPU را پشتیبانی کند.

NVLink بسیار سریع‌تر از PCIe است و مناسب برای سیستم‌های تک گره‌ای (single-node) است، جایی که تعداد محدودی GPU باید مستقیماً با هم ارتباط داشته باشند. در مقابل، NVSwitch یک فناوری توسعه‌یافته‌تر است که از NVLink برای اتصال تعداد بیشتری GPU در یک سیستم یا شبکه استفاده می‌کند و برای کاربردهای multi-node طراحی شده است.

جدول زیر که از وب‌سایت NVIDIA برداشته‌شده، پیشرفت‌های فناوری NVLink در نسل‌های مختلف را نشان می‌دهد:

نسل	پهنای باند برای هر GPU	تعداد NVLink	معماری پشتیبانی‌شده
نسل دوم	300 GB/s	6	NVIDIA Volta™
نسل سوم	600 GB/s	12	NVIDIA Ampere™
نسل چهارم	900 GB/s	18	NVIDIA Hopper™
نسل پنجم	1.8 TB/s	18	NVIDIA Blackwell™

جدول ۲ پیشرفت‌های فناوری NVLink در نسل‌های مختلف

به طور کلی، NVLink برای سیستم‌های کوچک‌تر و ارتباط مستقیم بین چند GPU مناسب است، در حالی که NVSwitch برای شبکه‌های بزرگ‌مقیاس و خوشه‌های عظیم GPU طراحی شده است و مقیاس‌پذیری بالاتری ارائه می‌دهد.

۲-۴_ مزیت استفاده نسبت به PCIe

پهنای باند بالا

NVLink و NVSwitch پهنای باند بسیار بیشتری نسبت به PCIe ارائه می‌دهند. برای مثال، نسل پنجم NVLink پهنای باندی ۱.۸ ترابایت بر ثانیه برای هر GPU دارد، در حالی که PCIe Gen5 تنها ۱۲۸ گیگابایت بر ثانیه ارائه می‌دهد.

تأخیر کمتر

این فناوری‌ها با حذف باس‌های مشترک و ارائه ارتباط مستقیم بین GPUها، تأخیر ارتباطی را به حداقل می‌رسانند. این تأخیر کمتر در کاربردهای انتقال پارامترها و داده‌های سیستم‌های یادگیری ماشین توزیع‌شده ضروری است.

ارتباطات All-to-All

NVSwitch با امکان ارتباطات گسترده بین GPUها در یک سیستم واحد یا چندین سیستم، کارایی را در آموزش مدل‌های بزرگ افزایش می‌دهد.

مقیاس‌پذیری بالا

با استفاده از NVSwitch، می‌توان تعداد زیادی GPU را در یک کلاستر به‌صورت یکپارچه متصل کنیم. این فناوری در سیستم‌هایی مانند GB200 NVL72 تا ۱ پتابایت در ثانیه پهنای باند فراهم می‌کند.

۳-۴_ ارتباط NVLink با PCIe

در حالی که NVLink و NVSwitch برای ارتباط بین GPUها طراحی شده‌اند، PCIe همچنان برای اتصال CPU به GPU و دستگاه‌های دیگر استفاده می‌شود که نیاز است داده‌ها و مدل و دیگر موارد از حافظه اصلی به حافظه GPU منتقل شوند. فرضا برای مدل‌های زبانی بزرگ که حجم داده‌ها بسیار زیاد است و حافظه GPU ها محدود ارتباط بین GPU و CPU کاملا ضروری است.

۵_پاسخ سوال شماره ۵

روش Data Parallelism به این صورت عمل می‌کند که داده‌ها بین چندین نود تقسیم می‌شوند و هر نود یک نسخه از مدل را نگه‌داری می‌کند. سپس، هر نود به‌طور مستقل گرادیان‌ها را محاسبه کرده و این گرادیان‌ها با یکدیگر به اشتراک گذاشته می‌شوند (برای نمونه میانگین گرفته می‌شود) تا به‌روزرسانی‌های مدل انجام شود.

باتوجه به مطالب اسلاید درس این روش برای بسیاری از الگوریتم‌های یادگیری ماشین که فرض می‌کنند داده‌ها به‌صورت مستقل و توزیع یکسانی دارند (فرض i.i.d)، مناسب است. با این حال، همه الگوریتم‌ها نمی‌توانند به خوبی با Data Parallelism پیاده‌سازی شوند و مسئله به نحوی است که نمیتوانیم فرض کنیم همه‌ی داده‌ها برای ما ارزش یکسانی دارند که بتوانیم آن‌ها را در نودهای مختلف تقسیم کنیم.

فرض i.i.d: الگوریتم‌هایی که نیاز به ترتیب خاصی از داده‌ها دارند یا داده‌ها در آن‌ها مستقل نیستند، با Data Parallelism سازگار نیستند. به‌عنوان مثال، در Curriculum Learning ، داده‌ها باید به ترتیب خاصی (از ساده به پیچیده) به مدل بدهیم و فرایند یادگیری خاصی دارد. این ترتیب‌دهی با تقسیم داده‌ها بین گره‌ها در Data Parallelism مشکل دارد.

در زمینه Reinforcement Learning هم فرض استقلال و توزیع یکسان داده‌ها اغلب برقرار نیست، زیرا داده‌ها به ترتیب زمانی و وابستگی‌های متوالی وابسته‌اند.

البته باید دقت شود که منظور در اینجا جنس خود داده‌هاست و منظور این نیست که در Data Parallel دیتا تا حد زیادی شکسته شود. برای نمونه در تسک ترجمه ماشینی قصد نداریم که تا حد کلمات داده شکسته شود که چالشی برای آموزش شبکه RNN وجود داشته باشد.

۶_پاسخ سوال شماره ۶

برای پاسخ به این سوال از مقاله‌ی [7] توسط Google Brain استفاده شد که به بررسی اثرات موازی‌سازی داده‌ها در آموزش شبکه‌های عصبی می‌پردازد و تأثیرات آن بر دقت مدل و سرعت همگرایی را تحلیل می‌کند.

روش Data Parallel به دلیل نیاز به هماهنگی بین گره‌ها و تقسیم داده‌ها، ممکن است منجر به کاهش خیلی کم دقت نهایی مدل نسبت به آموزش عادی شود. با این حال، این کاهش به شدت به تنظیمات خاص الگوریتم (مانند نوع بهینه‌سازی) و داده‌ها بستگی دارد.

بر اساس مقاله‌ی مذکور، دقت مدل‌هایی که با استفاده از روش data parallel آموزش دیده‌اند در مقایسه با مدل‌های عادی معمولاً **کاهش خاصی نمی‌یابد**. در رابطه با اندازه batch size این مقاله هیچ شواهدی پیدا نکرده است که نشان دهد افزایش اندازه‌ی batch size به تنهایی باعث کاهش عملکرد می‌شود. این موضوع حتی برای اندازه‌های بسیار بزرگ نیز برقرار بوده است.

تنظیم Hyperparameters

کاهش عملکردی که ممکن است در برخی مقایسه‌ها مشاهده شود، اغلب به دلیل تنظیم نادرست Hyperparameter ها مانند نرخ یادگیری است. در این مقاله گفته شده که برای هر مقدار batch size، تنظیم مجدد و دقیق Hyperparameter ها ضروری است و اگر به درستی تنظیم شوند، روش data parallel می‌تواند به همان سطح دقتی برسد که مدل‌های عادی دارند.

اثر اندازه‌ی batch size بر کارایی مدل

افزایش اندازه‌ی batch size ابتدا باعث بهبود سرعت همگرایی می‌شود، اما بعد از یک نقطه مشخص، روند مثبت کاهش می‌یابد و افزایش بیشتر batch size تأثیر کمی در کاهش تعداد مراحل همکرایی دارد. با این حال، این تغییرات به دقت نهایی مدل آسیب نمی‌رساند، به شرط اینکه هایپرپارامترها به‌درستی برای حالت‌های مختلف بهینه شوند.

در نتیجه، با استفاده از روش data parallel، دقت مدل کاهش چشمگیری نمی‌یابد، به شرط آنکه تعیین هایپرپارامترها به‌درستی انجام شود. مشکلات احتمالی کاهش دقت خیلی کم بیشتر ناشی از تنظیم نادرست هایپرپارامترها هستند تا خود روش پردازش موازی.

۷_پاسخ سوال شماره ۷

برای بهینه‌سازی زمان آموزش و استفاده حداکثری از توان پردازشی هر پردازنده، باید داده‌ها را به نسبت توان پردازشی آن‌ها تقسیم کنیم. از آنجا که توان پردازشی پردازنده iام برابر 2^i است، می‌خواهیم زمان پردازش هر پردازنده یکسان باشد تا هیچ پردازنده‌ای منتظر دیگری نماند.

در اینجا فرض می‌کنیم که D_i مقدار داده تخصیص‌یافته به پردازنده iام باشد. زمان پردازش برای هر پردازنده T_i اسست که برابر است با:

$$T_i = \frac{D_i}{P_i} = \frac{D_i}{2^i}$$

برای اینکه زمان پردازش در همه پردازنده‌ها یکسان باشد داریم $T = T_i$

$$\mathrm{T} = \frac{D_i}{2^i} \text{ - > } D_i = T \times 2^i$$

جمع کل داده‌ها برابر M است، بنابراین:

$$\sum_{i=1}^{K} D_i = M \text{ - > } \sum_{i=1}^{K} T \times 2^i = M$$

T یک مقدار ثابت بوده و از سیگما خارج می‌کنیم. خود سیگما برابر است با :

$$\sum_{i=1}^{K} 2^i = 2^{K+1} - 2 \text{ - > } \mathrm{T}(2^{K+1} - 2) = M$$

$$T = \frac{M}{2^{K+1} - 2}$$

پس مقدار داده تخصیص‌یافته به هر پردازنده:

$$D_i = T \times 2^i = \frac{M \times 2^i}{2^{K+1} - 2}$$

ساده‌سازی عبارت بالا:

$$D_i = \frac{M \times 2^i}{2 \times (2^K - 1)} = \frac{M \times 2^{i-1}}{2^K - 1}$$

بنابراین، باید داده‌ها را به نسبت بالا بین پردازنده‌ها تقسیم کنیم.

۸_پاسخ سوال شماره ۸

۸-۱_ ویژگی‌های اصلی ماتریس‌های تنک

۱. تعداد عناصر غیرصفر کم است:

یک ماتریس زمانی تنک در نظر گرفته می‌شود که تعداد عناصر غیرصفر آن n_{nz} به طور قابل توجهی کمتر از تعداد کل عناصر ماتریس $n \times m$ باشد یعنی $n_{nz} \ll n \times m$

۲. الگوی مشخص:

موقعیت تمامی عناصر غیرصفر به صورت مشخص است. این ویژگی باعث می‌شود بتوان اطلاعات ماتریس را با روش‌های بهینه‌تر ذخیره و پردازش کرد و نیازی نباشد تمام عناصر را برای هر کار پردازشی بخوانیم.

۸-۲_ مزایای ماتریس‌های تنک

کاهش مصرف حافظه:

برای ذخیره ماتریس‌های تنک از حافظه کمتری استفاده می‌شود. برای نمونه اگر بخوانیم داده از نوع Double را ذخیره کنیم به 8nm حافظه نیاز است ولی اگر ماتریس تنک باشد 16nnz نیاز خواهدبود.

کاهش محاسبات:

عملیات روی ماتریس‌های تنک مانند ضرب ماتریس تنها بر روی عناصر غیرصفر انجام می‌شود که محاسبات را به مرتبه O(nnz) می رساند.

۸-۳_ نمایش ماتریس‌های تنک

روش‌های متنوعی برای نمایش ماتریس‌های تنک وجود دارد. یک روش ساده این است که اطلاعات ماتریس در یک آرایه سه‌بعدی ذخیره شود:

ردیف‌ها: در ردیف اول شماره سطر هر عنصر غیرصفر.

ستون‌ها: در ردیف دوم شماره ستون هر عنصر غیرصفر.

مقادیر: در ردیف سوم مقدار عنصر غیرصفر.

۴-۸_ پاسخ قسمت ب

$$A = \begin{bmatrix} 0 & 5 & 0 & 0 & 0 & 0 \\ 12 & 0 & 0 & 0 & 0 & 3 \\ 0 & 0 & 0 & 71 & 0 & 0 \\ 0 & 0 & 4 & 0 & 1 & 0 \end{bmatrix}$$

در این سوال دقیقا مشخص نشده که باید ماتریس را به روش تنک ذخیره کنیم یا خیر. به همین دلیل تمام حالت‌های ممکن را پاسخ می‌دهیم.

پاسخ بر اساس اندیس‌گذاری فرترن

ابتدا ماتریس را به فرمت فرترن می‌نویسیم:

$$column - major = [0,12,0,0,\ 5,0,0,0,\ 0,0,0,4,\ 0,0,71,0,\ 0,0,0,1,\ 0,3,0,0]$$

سپس عملیات را روی بردار بالا انجام می‌دهیم:

$$2A + 5 = [5,29,5,5,\ 15,5,5,5,\ 5,5,5,13,\ 5,5,147,5,\ 5,5,5,7,\ 5,11,5,5]$$

در نهایت نمایش ماتریس به فرم زیر خواهد بود

$$2A + 5 = \begin{bmatrix} 5 & 15 & 5 & 5 & 5 & 5 \\ 29 & 5 & 5 & 5 & 5 & 11 \\ 5 & 5 & 5 & 147 & 5 & 5 \\ 5 & 5 & 13 & 5 & 7 & 5 \end{bmatrix}$$

پاسخ بر اساس تنک‌بودن ماتریس

ابتدا اطلاعات ماتریس را در قالب فرمت تنک یعنی یک ماتریس با ۳ ردیف ارائه دهیم، سپس همان اطلاعات را به صورت Column-Major مرتب کنیم. در این قسمت فرض میکنیم ماتریس همیشه باید بر اساس همین فرمت باشد و مقادیر صفر بمانند.

$$A = \begin{bmatrix} 0 & 5 & 0 & 0 & 0 & 0 \\ 12 & 0 & 0 & 0 & 0 & 3 \\ 0 & 0 & 0 & 71 & 0 & 0 \\ 0 & 0 & 4 & 0 & 1 & 0 \end{bmatrix}$$

$$row\_indices = [1,2,2,3,4,4]$$

$$column\_indices = [2,1,6,4,3,5]$$

$$values = [5,12,3,71,4,1]$$

انجام عملیات:

$$row\_indices = [1,2,2,3,4,4]$$

$$column\_indices = [2,1,6,4,3,5]$$

$$values = [15,29,11,147,13,7]$$

$$A = \begin{bmatrix} 1 & 2 & 2 & 3 & 4 & 4 \\ 2 & 1 & 6 & 4 & 3 & 5 \\ 15 & 29 & 11 & 147 & 13 & 7 \end{bmatrix}$$

$$column - major = [1,2,15,\ 2,1,29,\ 2,6,11,\ 3,4,147,\ 4,3,13,\ 4,5,7]$$

۹_پاسخ سوال شماره ۹

تعداد داده‌ها: ۳ میلیارد و هر نمونه داده با فرمت fp32 ذخیره می‌شود که ۳۲ بیت (۴ بایت) نیاز دارد.

پس حجم هر نمونه: ۴ بایت است و حجم کل داده ها حدود ۱۲ گیگابایت خواهدبود.

حافظهGPU ۸ گیگابایت است. این حافظه باید علاوه بر داده‌ها، پارامترهای مدل و گرادیان‌ها را نیز نگهداری کند. این محدودیت باعث می‌شود کل داده‌ها به‌صورت یکجا در حافظه GPU جای نگیرند.

۹-۱_ روش‌های ممکن برای آموزش شبکه عصبی

روش Batch Training:

در این روش ما کل داده‌های موجود در دیتاست را یک‌جا پردازش می‌کنیم. این باعث می‌شود که گرادیان با دقت خیلی بالایی محاسبه شود و الگوریتم هم سریع‌تر و پایدارتر به حالت بهینه برسد، چون همه داده‌ها در هر مرحله استفاده می‌شوند. اما مشکل اصلی اینجاست که به حافظه خیلی زیادی نیاز داریم. با توجه به این که GPU ما فقط ۸ گیگابایت حافظه دارد، چنین روشی در عمل امکان‌پذیر نیست، چون حجم داده‌ها خیلی بیشتر از ظرفیت حافظه است. علاوه بر این، حتی اگر حافظه کافی هم داشته باشیم، زمان پردازش برای یک‌جا محاسبه کردن کل داده‌ها بسیار زیاد می‌شود. پس، با این که این روش از نظر تئوری دقیق‌ترین است، برای مسئله ما مناسب نیست.

روش Mini-Batch Training:

در این روش، ما داده‌ها را به بخش‌های کوچک‌تر به نام Mini-Batch تقسیم می‌کنیم و هر بار فقط یک Mini-Batch را روی GPU پردازش می‌کنیم. این کار باعث می‌شود که حافظه کمتری نیاز داشته باشیم و بتوانیم حتی با همین GPU محدود آموزش را پیش ببریم. این روش تعادلی بین دقت و سرعت برقرار می‌کند که هم نیاز به حافظه زیاد نداریم و هم مدل با سرعت معقولی همگرا می‌شود. البته، اگر اندازه Mini-Batch

را خیلی کوچک در نظر بگیریم، ممکن است نوسانات گرادیان زیاد شود و مدل دیرتر به جواب برسد. از طرف دیگر، انتقال مکرر داده‌ها بین حافظه اصلی و GPU ممکن است کمی زمان اضافه به آموزش اضافه کند. با این وجود، این روش بهترین انتخاب برای مسئله ما است، چون هم با محدودیت حافظه سازگار است و هم در کل کارایی خوبی دارد.

در این روش، داده‌ها را به دسته‌های کوچک‌تر به نام Mini-Batch تقسیم می‌کنیم و هر بار فقط یک Mini-Batch را در GPU پردازش می‌کنیم. با این کار، GPU نیازی ندارد کل داده‌ها را همزمان ذخیره کند، بلکه فقط با یک زیرمجموعه کوچک از داده‌ها کار می‌کند. این روش کمک می‌کند مدل‌های پیچیده‌تر را حتی با GPUهایی که حافظه کمی دارند، آموزش دهیم. البته وقتی مینی‌بچ‌ها کوچک باشند، ممکن است نوسانات گرادیان‌ها بیشتر شود و همگرایی مدل کندتر پیش برود. همچنین، انتقال مکرر داده‌ها بین حافظه اصلی و GPU می‌تواند زمان آموزش را افزایش دهد و سربار commonucation بیشتری خواهیم داشت.

روش Stochastic:

در این روش، ما هر بار فقط یک نمونه از داده‌ها را برای محاسبه گرادیان استفاده می‌کنیم و وزن‌ها را به‌روزرسانی می‌کنیم. این روش حافظه خیلی کمی نیاز دارد، چون فقط با یک داده سر و کار داریم، و همین باعث می‌شود که بتوانیم حتی با محدودیت‌های سخت‌افزاری هم از آن استفاده کنیم. اما مشکل اصلی اینجاست که چون گرادیان فقط با یک داده محاسبه می‌شود، نوسانات زیادی دارد و این باعث می‌شود همگرایی مدل خیلی کندتر پیش برود. علاوه بر این، چون تعداد داده‌ها خیلی زیاد است، به‌روزرسانی وزن‌ها برای کل دیتاست زمان بسیار زیادی می‌گیرد. به همین دلیل، اگرچه این روش حافظه کمی مصرف می‌کند، اما به دلیل سرعت پایین برای مسئله ما مناسب نیست.

هر یک از این روش‌ها مزایا و معایبی دارند. روش Batch Training از نظر دقت و پایداری بهترین است، زیرا گرادیان را با استفاده از کل داده‌ها محاسبه می‌کند و در نتیجه همگرایی سریع‌تر و دقیق‌تری دارد. اما از نظر حافظه و زمان مورد نیاز ناکارآمد است و برای دیتاست‌های بزرگ و سخت‌افزار محدود عملی نیست. روش Mini-Batch Training در مقایسه با Batch Training، با تقسیم داده‌ها به بخش‌های کوچک‌تر نیاز به حافظه را به شدت کاهش می‌دهد و سرعت آموزش را بالاتر می‌برد، اما این کاهش حافظه به قیمت افزایش نوسانات گرادیان تمام می‌شود و ممکن است همگرایی کندتر شود. با این حال، این روش یک تعادل خوب بین دقت، زمان و منابع سخت‌افزاری ایجاد می‌کند و برای مسئله ما بهترین انتخاب است. روش Stochastic از نظر مصرف حافظه و سادگی پردازش، نسبت به هر دو روش دیگر برتری دارد، زیرا فقط با یک نمونه از داده‌ها کار می‌کند. اما از نظر دقت و کارایی، به دلیل نوسانات شدید گرادیان و نیاز به تعداد زیادی به‌روزرسانی، بسیار ضعیف‌تر عمل می‌کند و برای دیتاست‌های بزرگ زمان بسیار زیادی می‌طلبد. در نهایت، با توجه به محدودیت

سخت‌افزاری و حجم بالای داده‌ها، Mini-Batch Training بهترین گزینه است، چون تعادلی منطقی بین دقت، سرعت و استفاده از منابع فراهم می‌کند.

در این سوال گفته شده fp32 هستند ولی خب اگر میخواستیم حافظه کمتری مصرف کنیم میتوانیم Mixed Precision هم در نظر داشته باشیم. در این روش از دقت اعداد اعشاری مختلف استفاده می‌کنیم: fp16 برای محاسبات و fp32 برای ذخیره‌سازی مقادیر حساس. با کاهش دقت عددی در برخی بخش‌ها، می‌توانیم میزان حافظه موردنیاز را کاهش دهیم و سرعت محاسبات را افزایش دهیم. Mixed Precision Training معمولاً تا مصرف حافظه را کاهش می‌دهد و محاسبات را سریع‌تر می‌کند. البته، باید مراقب باشیم تا مشکلاتی مثل vanishing gradients پیش نیاید.

۱۰_ پاسخ سوال شماره ۱۰

۱-۱۰_ پاسخ قسمت الف)

ابتدا مروری به سطوح مختلف مطرح شده در اسلاید درسی بپردازیم.

سطح ۱ برای عملیات روی بردارها $O(n)$ است.

سطح ۲ برای عملیات ماتریس-بردار $O(n^2)$ است.

سطح ۳ برای عملیات ماتریس-ماتریس $O(n^3)$ است.

محاسبه αA شامل ضرب یک بردار و یک ماتریس است. ضرب بردار-ماتریس یک عملیات سطح ۲ BLAS است. در ادامه حاصل که یک بردار n تایی خواهد بود با β یک جمع بردار-بردار است که یک عملیات سطح ۱ است، اما از آنجایی که زمان محاسباتی آن در مقایسه با ضرب بردار-ماتریس ناچیز است، سطح عملیات همان سطح ۲ است. پس باتوجه به موارد مذکور و با توجه به اندازه‌های بسیار بزرگ ابعاد، **سطح BLAS مناسب سطح ۲ است**.

البته می توان از **سطح ۳** هم استفاده کرد ولی سطح ۳ از بهینه‌سازی‌های سطح بالا برای بهره‌برداری از محلی بودن داده‌ها و بهینه‌سازی حافظه کش استفاده می‌کند. اما این بهینه‌سازی‌ها زمانی مؤثر هستند که هر دو ماتریس بزرگ باشند و بتوان آن‌ها را به بلوک‌های کوچکتر تقسیم کرد. از طرف دیگر اگر ما خودمان به صورت دستی از **سطح ۱** استفاده کنیم ممکن است کدمان به اندازه پیاده‌سازی خود سطح ۲ نشود.

۱۱_ پاسخ سوال شماره ۱۱

۱-۱۱_ پاسخ قسمت الف)

در روش Data Parallelism، کل مدل عصبی باید در حافظه هر GPU قرار دهیم تا هر کدام را با داده‌های متفاوت به صورت مستقل آموزش دهیم. در این سناریو، اندازه مدل ۳۶ گیگابایت است، در حالی که هر GPU فقط ۸ گیگابایت حافظه دارد. از این ۸ گیگابایت، حداقل ۲ گیگابایت برای ذخیره داده‌ها در حین آموزش مورد نیاز است، بنابراین تنها ۶ گیگابایت برای مدل باقی می‌ماند. بنابراین، مدل ۳۶ گیگابایتی نمی‌تواند در حافظه یک GPU قرار گیرد و به همین دلیل استفاده از روش موازی‌سازی داده امکان‌پذیر نیست.

۲-۱۱_ پاسخ قسمت ب)

در روش Model Parallelism، مدل به بخش‌های کوچکتر شکسته می‌شود و هر بخش را در یک GPU قرار می‌دهیم. در اینجا هر GPU دارای ۸ گیگابایت حافظه است که ۲ گیگابایت آن برای داده‌ها مورد نیاز است، بنابراین ۶ گیگابایت برای مدل باقی می‌ماند. برای جای‌دادن مدل ۳۶ گیگابایتی، به حداقل تعداد GPU نیاز داریم که مجموع حافظه قابل استفاده آن‌ها برای مدل حداقل ۳۶ گیگابایت باشد.

حالت اول)

حافظه مورد نیاز برای اینکه فقط مدل را بارگذاری کنیم ۳۶ گیگابایت که اگر ما از **۵ عدد GPU** استفاده کنیم، ۴۰ گیگابایت خواهیم داشت. در این سناریو ۲ گیگابایت فضا برای داده هم خواهیم داشت ولی فقط در شرایط خاصی می‌تواند در تئوری جواب دهد که داده‌ها روی تمامی GPU ها نباشد که **در اینجا پاسخگوی مسئله ما نخواهد بود.**

حالت دوم)

در این حالت فرض میکنیم منظور مسئله از اندازه مدل، شامل تمامی اجزای مختلف مثل گرادیان‌ها و بهینه‌ساز هم باشد. یعنی تمام حافظه‌ای که برای آموزش نیاز است ۳۶ گیگابایت است. در این حالت میتوان از **۶ عدد GPU** استفاده کرد که به طور کلی ۴۸ گیگابایت خواهیم داشت. اگر روی هر GPU ۶ گیگابایت از مدل را قرار دهیم، و دیتا هم بر روی هر GPU باشد شرایطی می‌شود که یعنی از تمامی حافظه تمامی GPU ها استفاده کرده‌ایم. هرچند این حداقلی به نظر می‌رسد ولی **در حالت تئوری پاسخگوست** و احتمالا در عمل جواب نمی‌دهد و خطای کمبود حافظه خواهیم داشت.

حالت سوم)

در این حالت هم فرض می‌کنیم منظور از مسئله ۳۶ گیگابایت این است که این حجم از مدل شامل تمامی موارد مورد نیاز برای آموزش است. اگر بخواهیم از تئوری فاصله بگیریم از **۷ عدد GPU** استفاده می‌کنیم و بر روی هر کدام حدود ۵ گیگابایت از مدل را قرار می‌دهیم. و ۲ گیگابایت هم برای داده در نظر می‌گیریم که به طور کلی هر GPU ۷ گیگابایت از حافظه‌اش استفاده خواهدشد.

حالت چهارم)

در طول فرآیند آموزش (محاسبه گرادیان‌ها، BackPropagation ، و به‌روزرسانی وزن‌ها) حافظه اضافی موقتی نیز مورد نیاز است بنابراین، نیاز به فضای بیشتری نسبت به سناریوهای ساده قبل داریم و ممکن است تعداد بسیار بیشتری GPU نیاز داشته باشیم و این عدد به بیش از ۱۰ تا ۱۵ عدد بسته به نوع Optimizer و دیگر موارد برسید که بعید است فرض مسئله این بوده باشد.

۳-۱۱_ پاسخ قسمت ج)

در روش Data Parallelism ، پس از هر مرحله از آموزش، هر GPU گرادیان‌های محاسبه‌شده را باید با سایر GPUها به اشتراک بگذارد تا به‌روزرسانی مدل به صورت سراسری انجام شود. این فرآیند نیازمند انتقال حجم بزرگی از داده‌ها بین GPUها است که می‌تواند به عنوان یک گلوگاه عمل کند و سرعت آموزش را کاهش دهد، به‌ویژه اگر ارتباط بین GPUها پهنای باند کافی نداشته باشد. بنابراین، یکی از معایب این روش، سربار بالای ارتباط بین GPUها برای بروزرسانی پارامترها است.

۱۲_ پاسخ سوال شماره ۱۲

Type	Mantissa (no. of bits)	Exponent (no. of bits)	Source
Single precision (`fp32`)	23	8	IEEE standard
Half precision (`fp16`)	10	5	IEEE standard
Double precision (`fp64`)	52	11	IEEE standard
Brain floating point (`bfloat16`)	8	8	Google (optimized for precision and memory, suitable for precision-intensive workloads such as machine learning/deep learning)
TensorFloat (`tf32`)	10	8	NVIDIA (hybrid float giving range of `fp32` but precision of `fp16`)
Hopper E4M3 (`fp8`)	3	4	NVIDIA (optimized for memory and throughput)
Hopper E5M2 (`fp8`)	2	5	NVIDIA (optimized for memory and throughput)

شکل ۱-۱۲ فرمت‌های معمول اعداد Floating Point استفاده شده در یادگیری عمیق

به طور کلی استانداردهای مختلفی برای اعداد اعشاری وجود دارند که در یادگیری عمیق استفاده می‌کنیم. جدول بالا از کتاب مرجع درس است که فرمت‌های معمول مورد استفاده را به همراه مشخصات آن‌ها نمایش می‌دهد.

۱-۱۲_ تفاوت‌های میان fp32 و fp16

fp32 (Single Precision Floating Point)

ساختار: دارای ۳۲ بیت است که شامل ۱ بیت برای علامت، ۸ بیت برای نما، و ۲۳ بیت برای قسمت اعشاری (mantissa) است.

مزایا:

دقت بالا: مناسب برای محاسبات حساس که به دقت زیادی نیاز دارند.

پایداری بیشتر: مشکلاتی مانند vanishing gradients در این فرمت کمتر دیده می‌شود.

معایب:

نیاز به حافظه بیشتر: هر مقدار ۳۲ بیت فضا اشغال می‌کند، که محدودیت حافظه در سخت‌افزار را بیشتر می‌کند و ممکن است نتوانیم یک مدل را در حافظه GPU بارگذاری کنیم.

کندی در محاسبات: به دلیل اندازه بزرگ‌تر، سرعت محاسبات در مقایسه با fp16 کمتر است. سربار ارتباط نیز در سیستم‌های توزیع‌شده بیشتر خواهد بود.

fp16 (Half Precision Floating Point)

ساختار: دارای ۱۶ بیت است که شامل ۱ بیت برای علامت، ۵ بیت برای نما، و ۱۰ بیت برای قسمت اعشاری است.

مزایا:

مصرف حافظه کمتر: هر مقدار تنها ۱۶ بیت فضا اشغال می‌کند، که برای مجموعه داده‌های بزرگ یا مدل‌های پیچیده ایده‌آل است.

سرعت بالاتر: به دلیل حجم کمتر داده‌ها، عملیات محاسباتی سریع‌تر انجام می‌شود و سربار ارتباط هم کمتر خواهدبود.

معایب:

دقت کمتر: مناسب برای مدل‌هایی با حساسیت کمتر به دقت عددی.

حساسیت به اختلاف مقادیر: در مواردی مانند جمع دو مقدار بسیار متفاوت ممکن است خطا رخ دهد.

و احتمال بروز مشکلاتی مانند vanishing gradients. ممکن است اعداد خیلی کوچک وقتی تبدیل به FP16 شوند تبدیل به صفر شوند.

۲-۱۲_ مزایا و معایب استفاده از هر فرمت

مرحله آموزش:

استفاده از fp32، به دلیل دقت بالاتر، برای جلوگیری از ناپایداری‌های عددی و مسائل مربوط به گرادیان مناسب‌تر است.fp16 سرعت آموزش را افزایش می‌دهد و مصرف حافظه را کاهش می‌دهد، اما ممکن است به قابلیت‌هایی مانند mixed precision نیاز داشته باشیم تا دقت مدل حفظ شود.

پس از آموزش (Inference)

استفاده از fp16 برای inference خیلی خوب است زیرا مدل آموزش‌دیده به دقت fp32 نیازی ندارد. این روش به کاهش مصرف انرژی و سرعت اجرای مدل کمک می‌کند از طرفی استفاده از fp32 تنها در کاربردهایی که نیاز به پیش‌بینی‌های بسیار دقیق دارند توصیه می‌شود.

به طور کلی انتخاب بین این دو به نیاز دقت، سخت‌افزار موجود و حساسیت مدل بستگی دارد.

۳-۱۲_ پشتیبانی سخت‌افزارها

GPUهای جدید (NVIDIA Ampere و Volta) از هر دو فرمت fp32 و fp16 پشتیبانی می‌کنند و Tensor Cores برای mixed precision طراحی شده‌اند.GPUهای قدیمی یا سایر سخت‌افزارها ممکن است از fp16 پشتیبانی نکنند یا بهینه‌سازی کافی برای آن نداشته باشند.

	Supported CUDA Core Precisions								Supported Tensor Core Precisions							
	FP16	FP32	FP64	INT1	INT4	INT8	TF32	BF16	FP16	FP32	FP64	INT1	INT4	INT8	TF32	BF16
Nvidia Tesla P4	No	Yes	Yes	No	No	Yes	No	No	No	No	No	No	No	No	No	No
Nvidia P100	Yes	Yes	Yes	No	No	No	No	No	No	No	No	No	No	No	No	No
Nvidia Volta	Yes	Yes	Yes	No	No	Yes	No	No	Yes	No	No	No	No	No	No	No
Nvidia Turing	Yes	Yes	Yes	No	No	No	No	No	Yes	No	No	Yes	Yes	Yes	No	No
Nvidia A100	Yes	Yes	Yes	No	No	Yes	No	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes

شکل ۲-۱۲ پشتیبانی نسل‌های مختلف GPU از فرمت مختلف داده

بر اساس جدول داده‌شده دهمه سخت‌افزارها از FP32 پشتیبانی می‌کنند. اما FP16 بدر سخت‌افزارهای جدیدتر مانند NVIDIA Volta، Turing، و A100 پشتیبانی می‌شود. برخی سخت‌افزارهای قدیمی‌تر مانند NVIDIA Tesla P4 فقط از FP32 و FP64 پشتیبانی می‌کنند و FP16 را ارائه نمی‌دهند.

همچنین TPUها که برای یادگیری عمیق بهینه شده‌اند و BFLOAT16 را برای بهبود عملکرد به کار می‌گیرند و به طور مستقیم FP16 پشتیبانی نمی‌کنند ولی از FP32 پشتیبانی می‌کند. در آخرین نسخه از TPU ، تمام عملیات ضرب در فرمت عددی BFLOAT16 انجام شده و جمع‌ها در فرمت عددی FP32 انجام می‌شود. همچنین شتاب‌دهنده و پردازنده گرافیکی داخلی نسل ۱۵ CPU های اینتل از FP16 پشتیبانی خواهندکرد.

۱۳_ مراجع

[1] W. J. Dally, S. W. Keckler, and D. B. Kirk, “Evolution of the graphics processing unit (GPU),” IEEE Micro, vol. 41, no. 6, pp. 42–51, 2021.

[2] Y. H. Hu and S.-Y. Kung, “Systolic Arrays,” in Handbook of Signal Processing Systems, S. S. Bhattacharyya, E. F. Deprettere, R. Leupers, and J. Takala, Eds., New York, NY: Springer New York, 2013, pp. 1111–1143. doi: 10.1007/978-1-4614-6859-2_34.

[3] H. Watanabe and K. M. Nakagawa, “SIMD vectorization for the Lennard-Jones potential with AVX2 and AVX-512 instructions,” Comput. Phys. Commun., vol. 237, pp. 1–7, 2019, doi: https://doi.org/10.1016/j.cpc.2018.10.028.

[4] V. Gopal, W. Feghali, J. Guilford, E. Ozturk, G. Wolrich, M. Dixon, M. Lochtyuhin, and M. Perminov, “Fast Cryptographic Computation on IntelÂo Architecture Processors via Function Stiching,” White Pap. Intel, 2010.

[5] A. Hemeida, S. Hassan, S. Alkhalaf, M. Mahmoud, M. A. Saber, A. Eldin, T. Senjyu, and A. Alayed, “Optimizing matrix-matrix multiplication on intel’s advanced vector extensions multicore processor,” Ain Shams Eng. J., vol. 11, 2020, doi: 10.1016/j.asej.2020.01.003.

[6] E. Hölzl, “Communication Backends, Raw performance benchmarking,” 2020. [Online]. Available: https://mlbench.github.io/2020/09/08/communication-backend-comparison/

[7] C. J. Shallue, J. Lee, J. Antognini, J. Sohl-Dickstein, R. Frostig, and G. E. Dahl, “Measuring the effects of data parallelism on neural network training,” J. Mach. Learn. Res., vol. 20, no. 112, pp. 1–49, 2019.