بسم الله الرحمن الرحیم

پروژه دوم درس یادگیری ماشین توزیع شده

دکتر دوستی

مهدی وجهی - ۸۱۰۱۰۱۵۵۸

سوال ۱

مفاهیم اولیه

CUDA Kernel

CUDA یک سکو و مدل برنامه نویسی محاسبات موازی است که توسط NVIDIA توسعه داده شده که بتواند از مزیت و قدرت پردازنده های گرافیکی آن استفاده کرد. این سکو به برنامه نویس امکان می دهد که برنامه های محاسبه محور خود را شتاب دهد.

cuDNN

NVIDIA CUDA Deep Neural Network library یا cuDNN کتابخانه ای برای عناصر اولیه شبکه عصبی است که از شتابدهی GPU بهره می برد. این کتابخانه به صورت دقیق و کامل برای کاربرد شبکه های عصبی عمیق تنظیم شده. مواردی مانند ضرب ماتریسی، کانولوشن، سافت مکس و ... .
دلیل استفاده فریم ورک ها از آن کارایی بالای آن در GPU های NVIDIA هست.
محاسبات یادگیری عمیق اکثرا تکرار شونده هستند بنابراین انویدیا با دانش خود از سخت افزار کتابخانه ای طراحی کرده تا این عملیات ها را با سرعت خوبی انجام دهند و برنامه نویس ها درگیر پیاده سازی محاسبات نشوند. به صورت خلاصه مزیت آن کارایی و سادگی است.
بلس تمام مواردی که ما برای شبکه عصبی را می خواهیم ندارد مانند تباع های فعال سازی همچنین ما در پیاده سازی شبکه عصبی طراحی خود را ماتریسی انجام نمی دهیم از این لحاظ cuDNN مزیت دارد. همچنین تا جایی که متوجه شدم خود cuDNN برای محاسبات ماتریسی از بلس استفاده می کند.

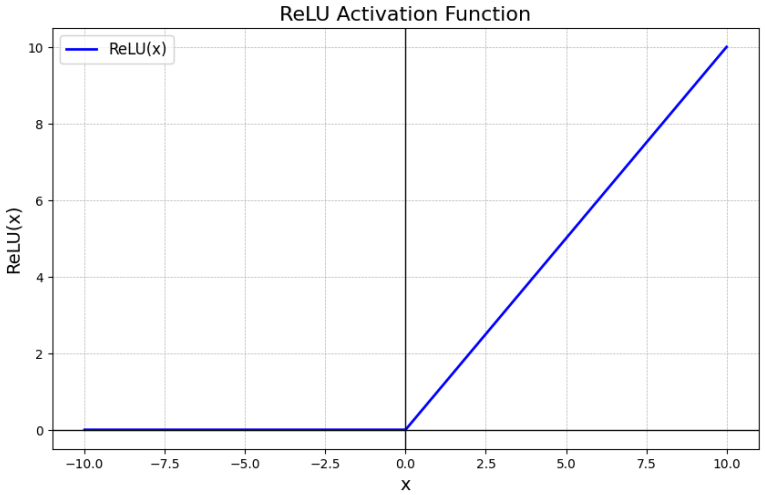
شرح کلاس FusedConvReLUFn

متد forward

این متد بسیار ساده است چون پیاده سازی ای که ما کردیم حالت خاص است و فقط با ابعاد خاص کار می کند ورودی ها بررسی می شود که ساختاری مطابق با آن داشته باشند. سپس تابع CUDA صدا زده می شود و مقادیر برای بکوارد و گرادیان گیری ذخیره می شود.

متد backward

ابتدا مقادیری که در forward ذخیره کردیم را بازیابی می کنیم سپس عملیات مشتق گیری را شروع می کنیم. خروجی تابع از ReLU رد می شد تصویر آن به شکل زیر است.



همانطور که می بینید مشتق آن در مقدار ورودی مثبت ۱ و در منفی ۰ است. اما در خود صفر مشتق ناپذیر است. اما برای حل این مشکل نقطه ی صفر هم برابر صفر در نظر میگیریم و داریم:

$$f'(x) = \begin{cases} 0 \text{ for } x < 0 \\ 1 \text{ for } x \geq 0 \end{cases}$$

پس گرادیان برابر با y > 0 می شود و سپس طبق قاعده زنجیره ای در مشتق لایه بعدی (که به عنوان ورودی داده شده) ضرب می کنیم و داریم:

grad_z = grad_output * mask

سپس باید باقی موارد را مشتق بگیریم. فرمول کانولوشن به شکل زیر است:

$$\text{out}(N_i, C_{\text{out}_j}) = \text{bias}(C_{\text{out}_j}) + \sum_{k=0}^{C_{\text{in}}-1} \text{weight}(C_{\text{out}_j}, k) \star \text{input}(N_i, k)$$

برای محاسبه گرادیان ورودی و وزن ها از توابع آماده پایتورچ استفاده کردیم و شکل و تنظیمات کانولوشن را به عنوان آرگومان دادیم همچین گرادیان ورودی آن هم طبق قاعده زنجیره این همان grad_z دادیم.

برای گرادیان بایاس هم طبق قاعده زنجیری داریم:

$$\frac{\partial L}{\partial b} = \frac{\partial L}{\partial z} \cdot \frac{\partial z}{\partial b}$$

چون بایاس بین تمام نورون های آن لایه مشترک است باید به صورت تجمیعی حساب شود. خوشبخانه مشتق اپراتور خطی است و می توانیم برای این کار تمام مقادیر مشتق را جداگانه حساب کنیم و سپس جمع بزنیم یعنی:

grad_z.sum(dim=[0, 2, 3])

همچین برای محاسبه گرادیان z نسبت به بایاس داریم که:

$$\frac{\partial z}{\partial b} = \frac{\partial}{\partial b}(C + b) = \frac{\partial C}{\partial b} + \frac{\partial b}{\partial b} = 0 + 1 = 1$$

بنابراین مقدار گرادیان بایاس همان جمع مقادیر گرادیان است.

آموزش و آزمایش مدل

در زمان اجرا متوجه شدیم که هر دو مدل یادگیری ای ندارند و به صورت تصادفی عمل می کنند. با بررسی کد متوجه شدیم که به جای این که چند لایه کانولوشن تعریف شود روی یک لایه حلقه می زنیم. اما طبق صحبتی که با طراح پروژه شد، فرمودند که نیازی به اصلاح نیست. نتایج به شرح زیر است:

مدل	زمان forward	زمان backward	دقت
شبکه با ۱۰ لایه			
baseline CNN (Conv2d + ReLU) - cuDNN	7.04s	13.88s	11.35%
fused CNN (FusedConvReLU)	3.41s	19.17s	11.35%
baseline CNN (Conv2d + ReLU) - cuDNN off	97.08s	220.32s	11.35%
baseline CNN (Conv2d + ReLU) - CPU	20.56s	109.53s	11.35%
شبکه با ۲ لایه			
baseline CNN (Conv2d + ReLU) - cuDNN	2.27s	4.87s	94.14%
fused CNN (FusedConvReLU)	1.88s	6.39s	94.12%
baseline CNN (Conv2d + ReLU) - cuDNN off	23.52s	43.88s	93.79%
baseline CNN (Conv2d + ReLU) - CPU	6.67s	22.03s	93.76%

تحلیل و بررسی

نکات جالب درباره اختلاف زمانی و دقت

یکی از نکات این نتایج این است که پیاده سازی ما در نهایت بدتر است اما با نگاه به نتایج می بینیم که زمان مسیر روبه جلو تقریبا نصف زمان حالت اصلی است ولی این موضوع در مسیر برگشت برعکس است. علت مسیر رفت می تواند این باشد که در پیاده سازی ما دیگر لازم نیست نتایج در حافظه سراسری نوشته و مجدد برای ReLU خوانده شود. در مسیر برگشت احتمالا پیاده سازی پیش فرض از الگوریتم های بهتر و سریع تری استفاده می کند و زمان کمتری دارد.

همچنین می بینم که عملکرد در صورت عدم استفاده از cuDNN می شکند و حتی بدتر از CPU می شود!!

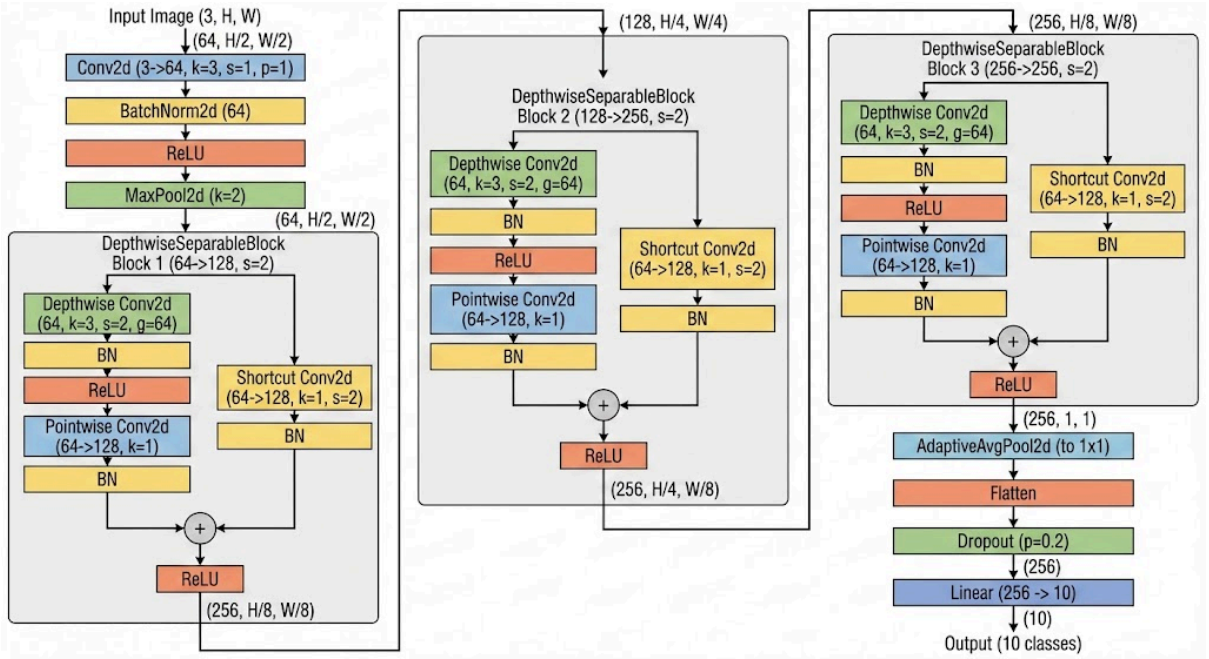
درباره دقت در ۲ لایه تفاوت زیادی مشاهده نمی شود و نکته جالب این است که وزن های لایه برای ۲ تکرار بهینه شده است و تا حد خوبی جواب گرفته. اما در ۱۰ لایه دیگر نمی توان وزن یکسانی که در همه به لایه ها به خوبی جواب بده پیدا کند و همچنین با توجه به این که دقت کلا ثابت مانده احتمالا پیده Dying ReLU رخ داده.

استفاده از کرنل اختصاصی

تقریبا هیچ موقع نوشتن کرنل اختصاصی ایده خوبی نیست و بهتر است از نهایت بهینه سازی و یکپارچگی در طراحی سخت افزار، محاسبات جبر خطی و ریاضی، پیاده سازی و کد نویسی، الگوریتم توسط خبرگان و طراحان انجام شده استفاده کنیم. زیرا به این راحتی نمی توان پیاده سازی ای به این کیفیت داشت.
تنها در زمانی که پیاده سازی خاصی انجام دادیم که با هیچ روش مناسبی نمی توان با کرنل های فعلی پیاده سازی کرد یا احیانا هیچ راه دیگری جز بهینه سازی در این سطح کرنل نمی بینیم از کرنل اختصاصی استفاده کنیم.
همچنین کرنل پیشفرض قابلیت حمل بیشتری دارد و باگ های آن به مراتب کمتر از پیاده سازی ما خواهد بود.

سوال ۲

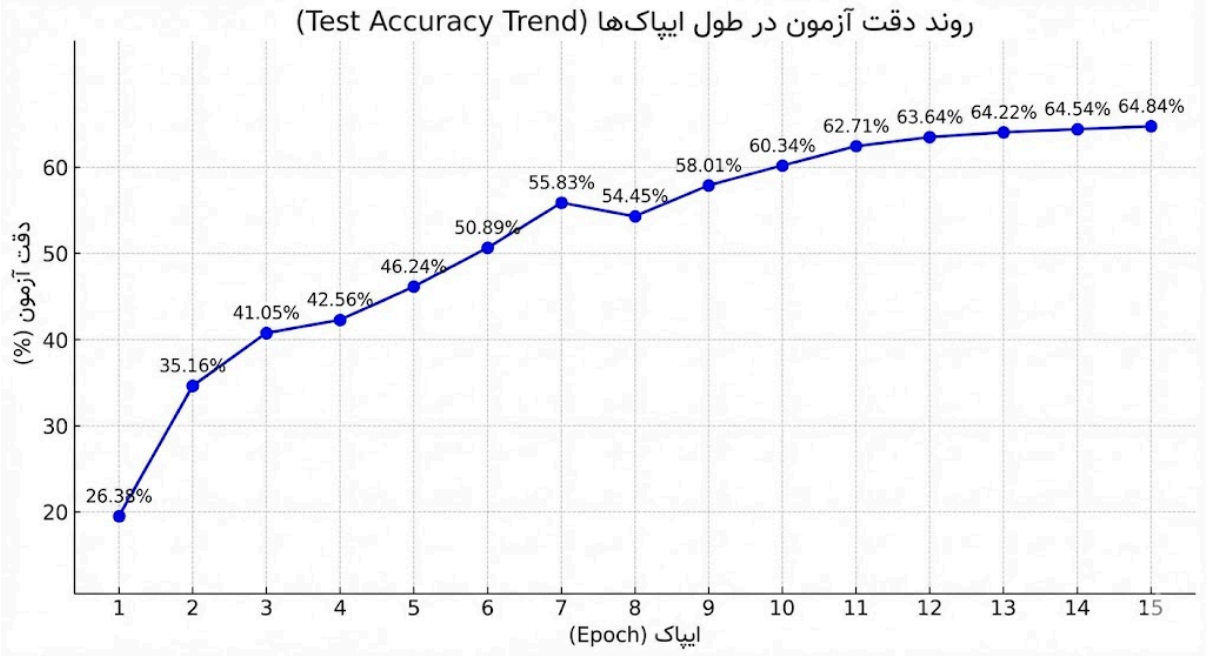
طراحی و آموزش مدل



ساختار مدل را به صورت شکل بالا طراحی کردیم. در نهایت با پیکربندی زیر مدل را آموزش می دهیم:

- **پیش‌پردازش (Augmentation)**: شامل ColorJitter، RandomHorizontalFlip، RandomCrop و RandomRotation برای افزایش تنوع داده‌های آموزشی.
- **بهینه‌ساز (Optimizer)**: الگوریتم AdamW با weight_decay=5e-2.
- **زمان‌بندی نرخ یادگیری (Scheduler)**: استفاده از OneCycleLR با حداکثر نرخ یادگیری 1e-3 برای همگرایی سریع‌تر در ۱۵ ایپاک.

در نهایت نمودار دقت به صورت زیر است:



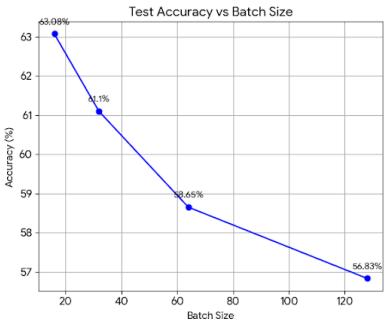
روند آموزش ۱۰۷ ثانیه زمان برد و ۳۱۲MB از حافظه کارت گرافیک را استفاده کرد.

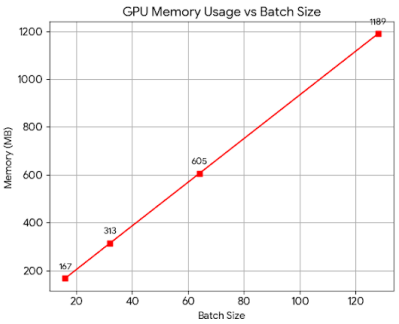
مقایسه نتایج با حالت توزیع شده

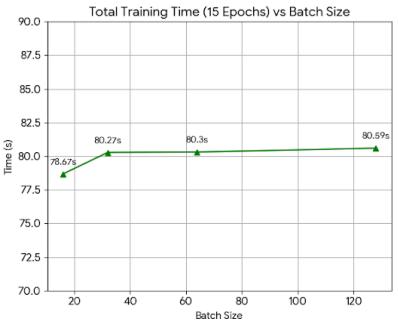
معیار ارزیابی	Single GPU (Rank 0)	Multi-GPU (DDP - 2 GPUs)	تغییرات
اندازه دسته (Local Batch Size)	32	32 (Global: 64)	-
زمان کل آموزش (ثانیه)	107.87	67.21	حدود 38% کاهش زمان
دقت نهایی روی داده تست	64.84%	61.85% (Rank 0)	حدود 3% افت دقت
حافظه مصرفی (به ازای هر GPU)	312.08 MB	313.80 MB	تقریباً ثابت

در نهایت زمان آموزش ۳۸ درصد کم شد ولی بدون سربار ما ۵۰ درصد انتظار داشتیم، سربار هایی که باعث این موضوع شده اند سربار ارتباط و سربار هماهنگ کردن گرادیان ها است و همچنین در زمان اجرا دیدیم که از تمام توان استفاده نمی شود یکی از دلایل آن این است که پردازه ها در انتظار حافظه می مانند.
چون هر پردازه بچ های ۳۲ تایی استفاده می کند در نهایت اندازه هر گام می شود بچ های ۶۴ تایی و با توجه به این که باقی پارامتر های ثابت است کمی اختلاف در دقت آن هم در ۱۵ ایپاک طبیعی است.
میزان دسترسی به حافظه در هر gpu برابر ماند زیرا هر دو کارت گرافیک داده را جداگانه بچ های ۳۲ تایی برگزاری می کنند. (در سوالات بعدی که با بچ ۱۶ اجرا کردیم دیدیم که تقریبا نصف شد)

سوال ۳





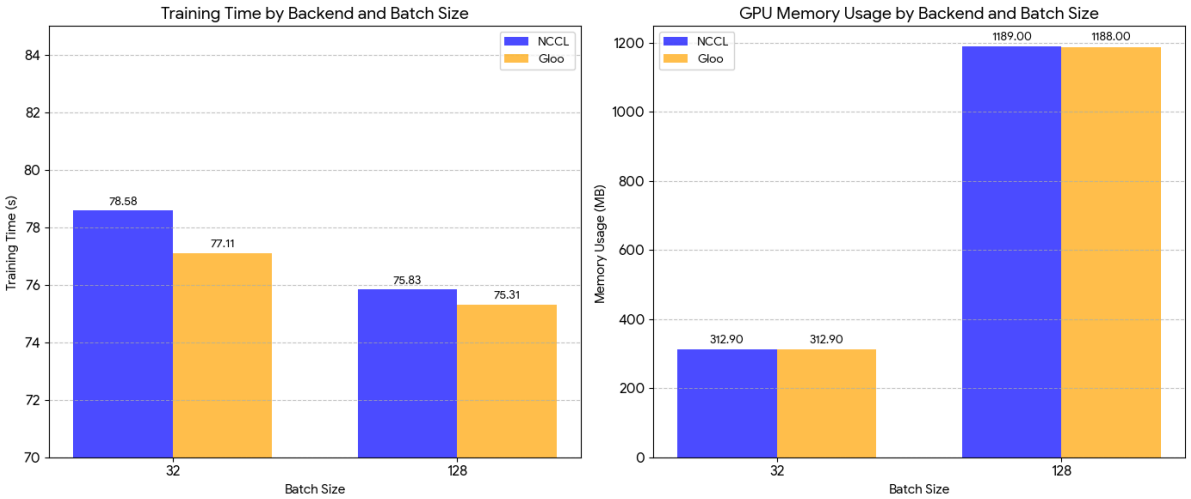


Batch Size (Local)	Total Batch Size (Global)	Training Time (s)	GPU Memory (MB)	Test Accuracy (%)
16	32	78.67	**167.25**	**63.08%**
32	64	80.27	313.80	61.10%
64	128	80.30	605.17	58.65%
128	256	80.59	1189.30	56.83%

طبق انتظار با حافظه مورد استفاده با اندازه بچ رابطه خطی دارد ولی برخلاف انتظار با افزایش اندازه بچ زمان کاهش پیدا نکرده دلیل این موضوع هم احتمالا به این دلیل است که شبکه زیاد پیچیده نیست و گلوگاه آن

حافظه است. دقت مدل به نظر با انداز بچ رابطه عکس دارد که این موضوع واقعا عجیب است تنها دلیلی احتمالی ای که به نظر می آیند این است که بچ های کوچک در حرکت تصادفی بیشتری دارد و این باعث می شود مدل در کمینه محلی گیر نکند.

سوال ۴



Backend	Batch Size (Local)	Total Training Time (s)	GPU Memory Usage (MB)
NCCL	32	78.58	312.9
Gloo	32	77.11	312.9
NCCL	128	75.83	1189
Gloo	128	75.31	1188

همانطور که مشاهده می کنید واقعا تفاوت معنادار بین حالت ها نیست.

منابع

https://docs.nvidia.com/cuda/cuda-c-programming-guide/
https://docs.nvidia.com/deeplearning/cudnn/latest/
https://datascience.stackexchange.com/questions/5706/what-is-the-dying-relu-problem-in-neural-networks
https://gemini.google.com/share/2b0496e02bee
https://gemini.google.com/share/084010819247
https://gemini.google.com/share/a9e818adb28a
https://gemini.google.com/share/4561de26bbe0
https://gemini.google.com/share/fb6ef21d04fc
https://gemini.google.com/share/8dd8c40a1dc6
https://gemini.google.com/share/4ece3c668b6b
https://gemini.google.com/share/6ece86d110c3
https://gemini.google.com/share/6ece86d110c3
https://gemini.google.com/share/59f0a2ee5e5b
https://gemini.google.com/share/fe5efa900d42
https://gemini.google.com/share/fb855b8e1d81
https://gemini.google.com/share/ab1c82baf0b5
https://gemini.google.com/share/0a08f9e7ef3f
https://gemini.google.com/share/6858ebc3ad58
https://gemini.google.com/share/99a7b94e54d4
https://gemini.google.com/share/68b2ff864909
https://gemini.google.com/share/3130e903da4b
https://gemini.google.com/share/dbafa12d8aca
https://gemini.google.com/share/d449fe4c5754
https://gemini.google.com/share/fd078e4a6d99
https://gemini.google.com/share/30d42d464faa
https://chatgpt.com/share/6925d240-2138-8001-a545-9a34502d7674
https://github.com/copilot/share/02224006-4a84-8081-a911-864ca0612834
https://github.com/copilot/share/486a4194-4aa0-8ca3-a902-0445a4212134