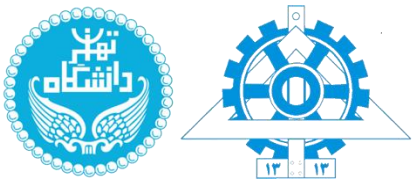
به نام خدا



سامانه‌های یادگیری ماشین توزیع‌شده

نمونه امتحان پایان‌ترم

مدت امتحان: ۱۲۰ دقیقه (۲ ساعت)

---

۱. (۱۵ نمره) فرض کنید توپولوژی حلقه را با n ماشین پیاده کرده‌اید. هر کدام از این n ماشین، به احتمال p_m ممکن است خراب شوند. اگر بتوانید یک ماشین دیگر با احتمال خرابی p_b به عنوان پشتیبان تهیه کنید تا در صورت خرابی یکی از ماشین‌ها با آن جایگزین شود، احتمال خرابی این سیستم چقدر **کاهش** می‌یابد؟ فرض کنید احتمال خرابی هر ماشین مستقل از ماشین‌های دیگر است.

پاسخ:

احتمال خرابی بدون پشتیبان:

$$1-(1-p_m)^n$$

احتمال خرابی با پشتیبان:

$$1-\left((1-p_m)^n+\binom{n}{1}p_m(1-p_m)^{n-1}(1-p_b)\right)$$

در نتیجه احتمال خرابی به میزان زیر کاهش می‌یابد:

$$\binom{n}{1}p_m(1-p_m)^{n-1}(1-p_b)=n\times p_m(1-p_m)^{n-1}(1-p_b)$$

۲. (۲۰ نمره) می‌دانیم در چند دهه‌ی گذشته، نرخ افزایش ظرفیت دیسک‌ها بسیار بیشتر از سرعت آن‌ها بوده است.

الف) توضیح دهید روش‌های ذخیره‌سازی توزیع‌شده چگونه مشکل کندی سرعت دیسک‌ها را برطرف می‌کنند؟
ب) آیا از ایده‌ی روش قسمت الف، برای افزایش سرعت ذخیره‌سازی روی یک ماشین نیز می‌توان استفاده کرد؟ توضیح دهید.

پاسخ:

الف) (۱۰ نمره) در روش‌های ذخیره‌سازی توزیع شده، داده بین چندین دیسک پخش شده و به طور همزمان می‌توان از روی چند دیسک خواند یا روی آن‌ها نوشت. بدین صورت، سرعت خواندن و نوشتن (اگر پهنای باند شبکه اجازه دهد) برابر با تعداد پارتیشن‌های داده می‌گردد.

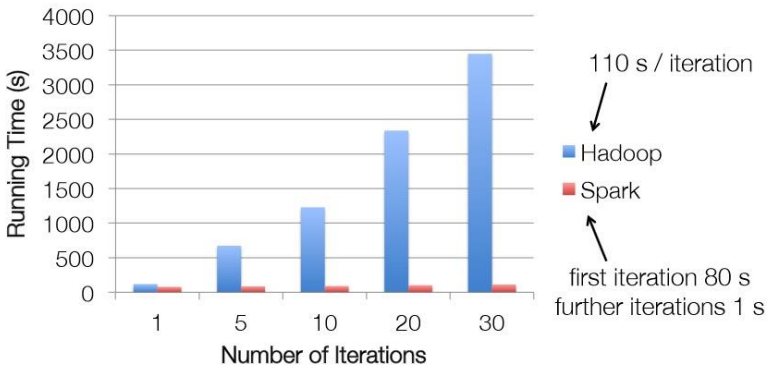
ب) (۱۰ نمره) به طور مشابه می‌توان با اتصال چندین دیسک به یک ماشین، داده‌ها را روی آن‌ها به طور توزیع شده (با روش‌هایی نظیر RAID) نوشت و خواند.

۳. (۳۰ نمره) نمودار زمان آموزش رگرسیون خطی به ازای تعداد تکرار (iteration) برای دو پلتفرم Spark و Hadoop در زیر آمده است.

الف) توضیح دهید چرا Spark در این نمودار برای تکرارهای بیشتر، بهتر از Hadoop عمل می‌کند؟

ب) چرا در اولین تکرار، Spark بسیار کند عمل کرده و پس از آن، بسیار سریع‌تر (۸۰ برابر) عمل می‌کند؟

ج) اگر میزان حافظه ماشین‌ها نصف شود، انتظار دارید نمودار پایین چه تفاوتی از لحاظ سرعت برای کلاسترهای Hadoop و Spark بکند؟



پاسخ:

الف) (۱۰ نمره) می‌دانیم که سرعت نوشتن و خواندن در حافظه (RAM) بسیار بیشتر از دیسک است. برخلاف Hadoop، Spark در هر تکرار، داده را روی دیسک ننوشته و نمی‌خواند. در عوض، عملیات در حافظه انجام شده که باعث افزایش سرعت آن نسبت به Hadoop می‌گردد.

ب) (۱۰ نمره) در اولین تکرار، Spark داده را از HDFS که در دیسک ذخیره شده می‌خواند. استفاده از دیسک در این مرحله موجب کند شدن Spark می‌گردد. پس از آن، Spark عملیات را در حافظه انجام می‌دهد که بسیار سریع‌تر است.

ج) (۱۰ نمره) سرعت Spark بسیار به میزان حافظه وابسته است. به همین خاطر، کاهش میزان حافظه، سرعت آن‌را به شدت کاهش می‌دهد. از طرف دیگر، Hadoop به اندازه Spark به حافظه احتیاجی ندارد و به همین دلیل عملکرد آن بسیار کمتر تحت تاثیر قرار می‌گیرد.

۴. (۳۰ نمره) در درس با مفهوم گراف محاسباتی (computational graph) برای یک مدل یادگیری ماشین آشنا شدیم.

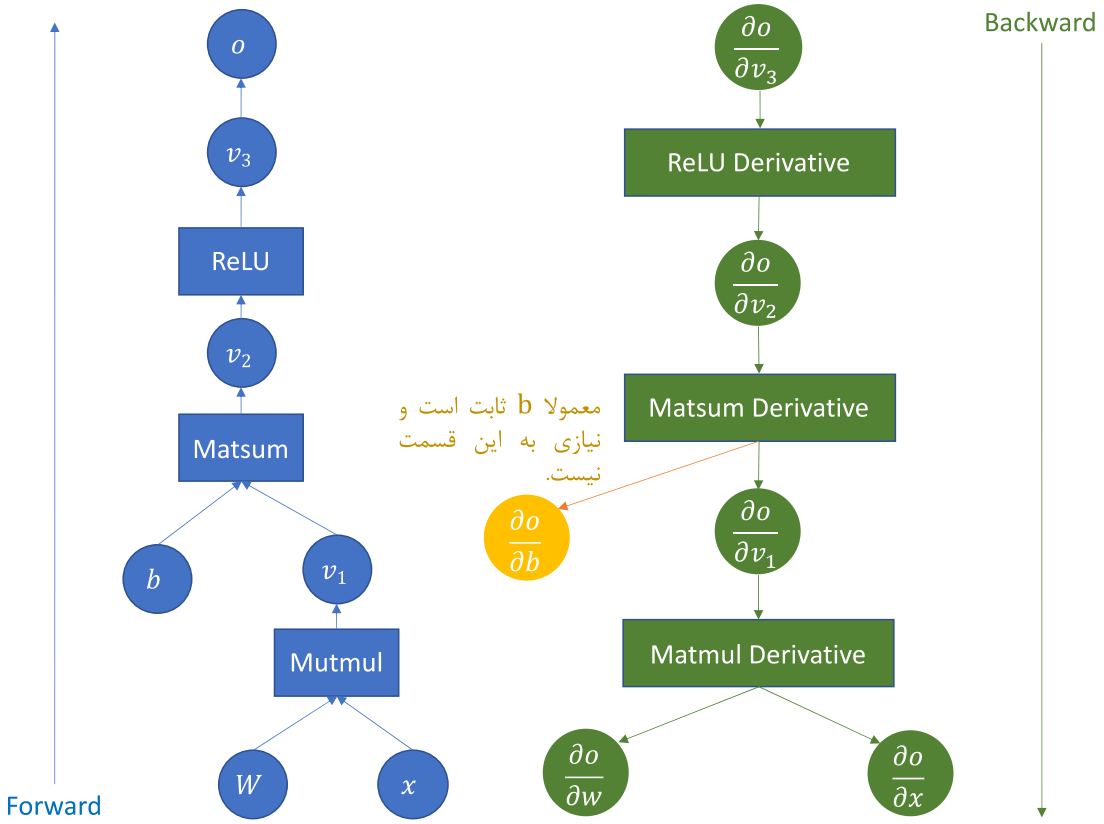
الف) گراف محاسباتی عبارت زیر و نیز گراف متناظر برای مشتق‌گیری خودکار را رسم کنید.

$$o(x, b) = \text{ReLU}(Wx + b)$$

ب) فرض کنید گراف بسیار بزرگ محاسباتی G به شما داده شده و می‌خواهید محاسبات آن را بین n ماشین توزیع کنید به طوری که یادگیری مدل در سریع‌ترین زمان ممکن انجام شود. این مسئله را به صورت یک مسئله‌ی بهینه‌سازی تعریف کنید. به عبارتی، تابع هدف (objective function) و قیود (constraints) مسئله بهینه‌سازی را مشخص کرده و در مورد آن توضیح مختصری ارائه دهید. برای راحتی، فرض کنید همه‌ی محاسبات در n ماشین به صورت موازی و همزمان (سنکرون) انجام می‌شود.

پاسخ:

الف) (۱۰ نمره: ۵ نمره گراف forward و ۵ نمره گراف backward)



ب) (۲۰ نمره: ۵ نمره تابع هدف، ۱۰ نمره قیود، ۵ نمره توضیح، در صورت ارائه توضیح دقیق برای تابع هدف و قیود، نمره با ارفاق داده می‌شود) پاسخ دقیق این مسئله، نیازمند زمان‌بندی گراف محاسباتی با روشی مثل ASAP/ALAP دارد. سپس روی این گراف، بر اساس همزمانی رئوس شبکه، طول مسیر بحرانی (طولانی‌ترین مسیر گراف) باید کمینه شود. به دلیل پیچیدگی این فرمول‌بندی، صرفا سعی در کمینه‌سازی زمان لازم برای تبادل داده بین ماشین‌ها خواهیم کرد (مطلبی که در کلاس به آن پرداخته شد.)

فرض کنید گراف داده شده $G = (V, E)$ است. مسئله‌ی بهینه‌سازی را با متغیر بهینه‌سازی a_i تعریف می‌کنیم که نشان می‌دهد محاسبات راس i در گراف روی کدام ماشین اجرا می‌شود. در این گراف، وزن بین دو راس i و j را به صورت $w(i,j)$ نوشته و اندازه‌ی آن‌را به صورت زیر تعریف می‌کنیم:

$$w(i,j) = \begin{cases} \text{computation node } i & \text{if } a_i = a_j \\ \text{computation node } i + \text{communication } (i \to j) & \text{otherwise} \end{cases}$$

به علاوه تابع $m_i(x)$ را به صورت زیر تعریف می‌کنیم:

$$m_i(x) = \begin{cases} 1 & \text{if } x = i \\ 0 & \text{otherwise} \end{cases}$$

بر اساس تعریف بالا می‌توان مسئله‌ی بهینه‌سازی را به صورت زیر تعریف کرد:

$$\arg\min_{a_i} \sum_{\substack{1 \le i,j \le |V| \\ a_i \ne a_j, e(i,j) \in E}} w(i,j)$$

subject to:

$$1 \le a_i \le n, \quad a_i \in \mathbb{N}$$

$$\forall i \in \{1, \dots, n\}: \quad \left| \sum_{1 \le j \le |V|} m_i(a_j) - \frac{|V|}{n} \right| \le \varepsilon$$

تابع هدف سعی در کاهش میزان برقراری ارتباط بین کامپیوترها دارد. قید اول، حدود متغیر a_i را تعریف می‌کند و قید دوم سعی در ایجاد توازن بین میزان محاسبات بین کامپیوترها دارد. ثابت ε، میزان تحمل عدم توازن بین کامپیوترها را تعیین می‌کند.

۵. (۱۵ نمره) در هر گام از آموزش یک شبکه‌ی توزیع شده، ممکن است گرادیان همه وزن‌ها در همه‌ی پروسه‌ها نیاز به بروزرسانی نداشته باشند. به عبارتی، یک گرادیان، در یک گام آموزش ممکن است در یک پروسه شرکت کند و در پروسه‌ی دیگر در همان گام آموزش شرکت نکند. چگونه و با چه ساز و کاری PyTorch DistributedDataParallel پارامترهایی را که در هیچ پروسه‌ای در گام جاری نیاز به بروزرسانی نداشته‌اند، شناسایی می‌کند؟

پاسخ: در مقاله PyTorch Distributed: Experiences on Accelerating Data Parallel Training، روشی به کمک نقشه‌ی بیتی (bitmap) ارائه شده که در ادامه تشریح می‌گردد. هر پروسه، یک نقشه‌ی بیتی برای پارامترهای استفاده نشده نگهداری می‌کند (مثلا صفر نشان‌دهنده پارامتر استفاده نشده و یک نشان‌دهنده پارامتر استفاده شده). سپس با فراخوانی دستور AllReduce، همه‌ی نقشه‌های

بیتی جمع شده و عمل OR روی آن‌ها انجام می‌گیرد. خانه‌های این نقشه‌ی بیتی که صفر باقی مانده‌اند، پارامترهای مورد نظر ما هستند.