تکلیف سوم

مجتبی ملائی
۴۰۱۳۱۳۸۳

۱

۱. یکی از چالش‌های اصلی در استفاده از شبکه‌های عصبی معمولی برای پردازش تصاویر، تعداد بسیار زیاد پارامترهاست. در این نوع شبکه‌ها، هر نورون به‌صورت کامل به تمام نورون‌های لایه قبلی متصل است، که در مورد تصاویر با ابعاد بالا، منجر به تولید میلیون‌ها پارامتر می‌شود. به‌عنوان مثال، یک تصویر رنگی با ابعاد ۲۲۴×۲۲۴ دارای بیش از ۱۵۰ هزار ورودی است، که اتصال آن به حتی یک لایه پنهان متوسط باعث افزایش نمایی تعداد وزن‌ها می‌شود. این موضوع نه تنها بار محاسباتی را به‌طرز چشم‌گیری افزایش می‌دهد، بلکه باعث دشواری در آموزش شبکه، نیاز به منابع پردازشی قوی، و افزایش ریسک overfitting می‌شود.

۲. دومین مشکل اساسی شبکه‌های عصبی معمولی در پردازش تصاویر، عدم توانایی در بهره‌برداری از ساختار فضایی و ویژگی‌های محلی تصویر است. این شبکه‌ها داده‌های ورودی را به‌صورت یک بردار یک‌بعدی در نظر می‌گیرند و هیچ توجهی به موقعیت مکانی پیکسل‌ها یا الگوهای محلی (نظیر لبه‌ها و بافت‌ها) ندارند. در نتیجه، اطلاعات مکانی که برای تحلیل تصویر بسیار حیاتی هستند، نادیده گرفته می‌شوند. این محدودیت باعث می‌شود که مدل نتواند به‌درستی ویژگی‌های بصری را استخراج کرده و درک عمیقی از محتوای تصویر به‌دست آورد، در حالی‌که شبکه‌های کانولوشنی با بهره‌گیری از فیلترهای محلی و اشتراک وزن، این ساختارهای فضایی را به‌شکل موثرتری مدل‌سازی می‌کنند.

۲

در بینایی کامپیوتر کلاسیک، ویژگی‌ها به‌صورت دستی و با استفاده از الگوریتم هاو یا فیلتر های از پیش تعریف شده استخراج می‌شوند. این ویژگی‌ها بر اساس دانش انسان و تجربه طراحی می‌شوند و سپس به یک مدل یادگیرنده ساده برای طبقه‌بندی یا شناسایی داده می‌شوند. این فرآیند نیازمند تخصص انسانی و تنظیمات دقیق است و در مواجهه با داده‌های متنوع یا نویزدار ممکن است عملکرد ضعیفی داشته باشد.

در مقابل، روش‌های CNN ویژگی‌ها را به‌صورت خودکار و سلسله‌مراتبی از داده خام یاد می‌گیرند. شبکه با یادگیری از داده‌های برچسب‌دار، فیلترهایی را آموزش می‌بیند که به‌صورت لایه‌به‌لایه از ویژگی‌های ساده (مانند لبه‌ها) تا ویژگی‌های پیچیده (مانند اشیاء یا چهره‌ها) را استخراج می‌کنند. این روش نه‌تنها نیاز به طراحی دستی ویژگی‌ها را حذف می‌کند، بلکه باعث افزایش دقت، انعطاف‌پذیری و عملکرد در وظایف مختلف بینایی کامپیوتر شده است.

در واقع در روش های CNN تنظیم این فیلتر هاتوسط یادگیری خود مدل تعیین می‌شوند نه براساس طراحی انسان ها.

۳

استفاده از Max Pooling به جای Average Pooling در شبکه‌های عصبی کانولوشنی معمولاً ترجیح داده می‌شود، زیرا Max Pooling ویژگی‌های برجسته‌تر و مهم‌تر تصویر را بهتر حفظ می‌کند و در عمل به بهبود عملکرد مدل در وظایف بینایی کامپیوتر منجر می‌شود.

در Max، Pooling بیشترین مقدار در یک ناحیه مشخص انتخاب می‌شود، که به معنای حفظ قوی‌ترین فعال‌سازی‌ها (activations strongest) است. این فعال‌سازی‌ها معمولاً نشان‌دهنده وجود ویژگی‌های مهم مانند لبه‌ها، گوشه‌ها یا بافت‌های خاص هستند. در مقابل، Average Pooling میانگین مقادیر را محاسبه می‌کند که می‌تواند اطلاعات کلیدی را با مقادیر غیرمهم ترکیب کرده و در نتیجه ویژگی‌های برجسته را "تضعیف" کند. بنابراین، Max Pooling با تمرکز روی برجسته‌ترین نشانه‌ها در تصویر، به استخراج ویژگی‌هایی کمک می‌کند که برای تشخیص و طبقه‌بندی دقیق‌تر حیاتی هستند.

در واقع، Max Pooling همچنین نوعی مقاومت در برابر نویز ایجاد می‌کند؛ زیرا با نادیده گرفتن مقادیر ضعیف‌تر، تأثیر ناهنجاری‌های کوچک یا تغییرات جزیی در تصویر کاهش می‌یابد. در مجموع، Max Pooling باعث حفظ اطلاعات مهم، کاهش ابعاد، افزایش کارایی محاسباتی، و بهبود قابلیت تعمیم شبکه می‌شود، که دلایل اصلی ترجیح آن بر Average Pooling در اکثر کاربردهای عملی است.

۴

سایز خروجی لایه کانولوشنی:

$$n_h[l] = \lfloor \frac{n_h[l-1] - 2p - f_h}{s} \rfloor + 1$$
$$n_w[l] = \lfloor \frac{n_w[l-1] - 2p - f_w}{s} \rfloor + 1$$
$$n_c[l] = K$$

تعداد پارامتر های فیلتر های کانولوشنی:

$$(n_h[l] * n_w[l] * n_c[l] + 1) * K$$

سایز خروجی لایه های پولینگ:

$$n_h[l] = \lfloor \frac{n_h[l-1] - f_h}{s} \rfloor + 1$$
$$n_w[l] = \lfloor \frac{n_w[l-1] - f_w}{s} \rfloor + 1$$
$$n_c[l] = n_c[l-1]$$

تعداد پارامتر های FC:

$$(1 + n[l-1]) * n[l]$$

۱. لایه کانولوشنی اول:

$$n = \lfloor \frac{28 - 2 \times 0 - 5}{1} \rfloor + 1 = 24$$
$$\text{size} = 24 \times 24 \times 6$$
$$\text{params} = (5 \times 5 \times 1 + 1) \times 6 = 156$$

۲. لایه avg pooling اول:

$$n = \lfloor \frac{24 - 2}{2} \rfloor + 1 = 12$$
$$\text{size} = 12 \times 12 \times 6$$
$$\text{params} = 0$$

۳. لایه کانولوشنی دوم:

$$n = \lfloor \frac{12 - 2 \times 0 - 5}{1} \rfloor + 1 = 8$$
$$\text{size} = 8 \times 8 \times 16$$
$$\text{params} = (5 \times 5 \times 6 + 1) \times 16 = 2416$$

۴. لایه avg pooling دوم:

$$n = \lfloor \frac{8 - 2}{2} \rfloor + 1 = 4$$
$$\text{size} = 4 \times 4 \times 16 = 256$$
$$\text{params} = 0$$

۵. لایه FC1:

$$\begin{aligned}\text{input} &= 4 \times 4 \times 16 = 256\\ \text{size} &= 128\\ \text{params} &= (256+1) \times 128 = 32,896\end{aligned}$$

۶. لایه FC2:

$$\begin{aligned}\text{input} &= 128\\ \text{size} &= 64\\ \text{params} &= (128+1) \times 64 = 8,256\end{aligned}$$

۷. لایه خروجی (softmax):

$$\begin{aligned}\text{input} &= 64\\ \text{size} &= 10\\ \text{params} &= (64+1) \times 10 = 650\end{aligned}$$

۸. مجموع پارامترها:

$$\begin{aligned}\text{layers Conv} &= 156 + 2416 = 2,572\\ \text{layers FC} &= 32,896 + 8,256 + 650 = 41,802\\ \text{Total} &= \boxed{44,374}\end{aligned}$$

۵

خیر، نمونه‌برداری تصادفی ساده از دیتاست برای تقسیم آن به زیرمجموعه‌های آموزش، ارزیابی و آزمایش، همواره کفایت نمی‌کند؛ زیرا در بسیاری از موارد ممکن است این روش منجر به تقسیم نامتوازن یا نامناسب داده‌ها شود که بر عملکرد مدل اثر منفی می‌گذارد.
در شرایطی که توزیع کلاس‌ها نامتوازن است (یعنی برخی کلاس‌ها نمونه‌های بسیار بیشتری نسبت به سایر کلاس‌ها دارند)، نمونه‌برداری تصادفی ممکن است باعث شود که برخی کلاس‌ها در یکی از زیرمجموعه‌ها (مثلاً تست یا validation) به‌درستی نمایندگی نشوند یا حتی کاملاً حذف شوند. این امر باعث می‌شود مدل نتواند به‌درستی یاد بگیرد یا ارزیابی دقیق انجام دهد. به همین دلیل، معمولاً از روش‌های دقیق‌تری مانند نمونه‌برداری طبقه‌بندی‌شده (stratified sampling) استفاده می‌شود تا نسبت کلاس‌ها در هر زیرمجموعه حفظ شود.
همچنین در برخی کاربردها (مثلاً تشخیص چهره یا داده‌های زمانی)، لازم است نمونه‌های وابسته به هم (مانند فریم‌های یک ویدئو یا تصاویر یک فرد) در یک زیرمجموعه باقی بمانند تا از نشت اطلاعات (data leakage) جلوگیری شود. در این موارد نیز نمونه‌برداری تصادفی ساده ناکافی و حتی خطرناک است. در نتیجه، تقسیم‌بندی مؤثر دیتاست نیازمند توجه به ساختار داده، توزیع کلاس‌ها و هدف مدل‌سازی است و صرفاً تکیه بر تصادفی‌سازی کافی نیست.

۶

خیر، اضافه کردن یک مقدار ثابت به تمام ورودی‌های تابع Softmax خروجی آن را تغییر نمی‌دهد. دلیل این موضوع به ویژگی نرمال‌سازی نسبی در تابع Softmax بازمی‌گردد.
اثبات: برای یک بردار ورودی $\mathbf{z} = [z_1, z_2, \ldots, z_n]$، خروجی Softmax به‌صورت زیر تعریف می‌شود:

$$\text{Softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}}$$

افزودن مقدار ثابت c به همه ورودی‌ها:
فرض کنید بردار جدید $\mathbf{z}' = [z_1 + c, z_2 + c, \ldots, z_n + c]$ باشد. حال خروجی Softmax برای $z_i + c$ برابر است با:

$$\text{Softmax}(z_i + c) = \frac{e^{z_i + c}}{\sum_{j=1}^{n} e^{z_j + c}} = \frac{e^c \cdot e^{z_i}}{e^c \cdot \sum_{j=1}^{n} e^{z_j}} = \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}}$$

همان‌طور که مشاهده می‌شود، e^c از صورت و مخرج ساده می‌شود، بنابراین خروجی تغییر نمی‌کند.

۷

توصیه می‌شود که در مدل‌های دارای Dropout از نمودار هزینه (Loss) بر حسب ایپاک (Epoch) برای تعیین نرخ یادگیری به‌تنهایی استفاده نشود، زیرا نوسانات ذاتی Dropout باعث ناپایداری در مقدار هزینه می‌شود و ممکن است اطلاعات گمراه‌کننده درباره روند یادگیری ارائه دهد.
در فرآیند Dropout، در هر بار forward pass، زیرمجموعه‌ای تصادفی از نورون‌ها غیرفعال می‌شوند. این امر باعث می‌شود که مقدار تابع هزینه از یک ایپاک به ایپاک دیگر رفتار نوسانی و پر از جهش‌های تصادفی داشته باشد. در نتیجه، نمودار هزینه ممکن است دارای افت‌وخیزهای زیاد باشد که تشخیص روند کلی همگرایی یا واگرایی مدل را دشوار می‌کند. این امر می‌تواند موجب انتخاب نامناسب نرخ یادگیری شود، چرا که تحلیل‌گر ممکن است نوسانات را به نرخ یادگیری نامطلوب نسبت دهد در حالی که منشأ آن Dropout است.

۸

۱. **کاهش یا تغییر ابعاد کانال‌ها:** کانولوشن ۱×۱ تنها بر بعد عمقی (تعداد کانال‌ها) اثر می‌گذارد و بدون تغییر در ابعاد مکانی (طول و عرض تصویر)، می‌تواند تعداد کانال‌ها را کاهش یا افزایش دهد. این قابلیت برای کاهش پیچیدگی محاسباتی یا افزایش قدرت بازنمایی در لایه‌های خاص بسیار مفید است. به عنوان مثال، در U-Net از کانولوشن ۱×۱ برای تطبیق تعداد کانال‌ها قبل از اتصال لایه‌های encoder و decoder استفاده می‌شود.

۲. **افزایش غیرخطی بودن و ترکیب ویژگی‌ها:** با اعمال فیلتر ۱×۱ روی چندین کانال، مدل قادر است بین ویژگی‌های مختلف ترکیب‌های غیرخطی جدیدی یاد بگیرد. این عملیات شبیه به یک لایه Connected Fully بر روی هر موقعیت مکانی (pixel) است و به مدل اجازه می‌دهد تا وابستگی‌های میان‌کانالی را در همان مکان بررسی کند، که برای استخراج ویژگی‌های سطح بالا مفید است.

۹

۱.۹ نشانه ها

۱. **اختلاف زیاد بین دقت آموزش و دقت اعتبارسنجی:** مدل در داده‌های آموزش عملکرد بسیار خوبی دارد (خطای کم یا دقت بالا)، اما در داده‌های اعتبارسنجی (Validation) عملکرد ضعیفی نشان می‌دهد. این اختلاف نشان‌دهنده یادگیری بیش از حد ویژگی‌های خاص داده‌های آموزش است، نه الگوهای عمومی.

۲. **کاهش پیوسته خطای آموزش، اما ثابت یا افزایش‌یافتن خطای اعتبارسنجی:** اگر با گذشت ایپاک‌ها، خطای آموزش کاهش یابد ولی خطای اعتبارسنجی بهبود نیابد یا حتی افزایش یابد، این نشانه واضحی از overfitting است.

۲.۹ دو تکنیک کاربردی برای کاهش Overfitting:

۱. **استفاده از Regularization مانند Dropout یا Regularization L۲:**

- **Dropout:** به‌صورت تصادفی برخی نورون‌ها را در طول آموزش غیرفعال می‌کند، که از وابستگی بیش‌ازحد مدل به مسیرهای خاص جلوگیری می‌کند. در CNN ها معمولاً در لایه‌های Connected Fully استفاده می‌شود.
- **Regularization L۲:** با اضافه‌کردن جریمه به وزن‌های بزرگ، مدل را وادار می‌کند تا پارامترهای ساده‌تر و عمومی‌تری بیاموزد.

۲. **افزایش داده‌ها از طریق Data Augmentation:** با ایجاد تغییرات تصادفی در تصاویر آموزشی (چرخش، برش، مقیاس‌گذاری، وارونه‌سازی، تغییر روشنایی و غیره)، داده‌های متنوع‌تری ایجاد می‌شود که کمک می‌کند مدل ویژگی‌های پایدار و عمومی‌تری یاد بگیرد و از حفظ ویژگی‌های خاص و جزئی تصویر اصلی اجتناب کند.

۱۰

در مدل‌های یادگیری عمیق، خروجی Softmax معمولاً به‌عنوان توزیع احتمال روی کلاس‌ها تفسیر می‌شود. اما در عمل، این احتمال‌ها اغلب overconfident هستند، یعنی مدل حتی در شرایط عدم قطعیت، احتمال بسیار بالایی برای یک کلاس خاص اعلام می‌کند. این موضوع باعث کاهش قابلیت اطمینان مدل در سیستم‌های حساس به تصمیم‌گیری می‌شود. برای کالیبره‌سازی (Calibrating) بهتر خروجی، از پارامتر دمایی (Temperature Scaling) استفاده می‌شود. در این روش، خروجی logits مدل (بردار قبل از Softmax) بر یک پارامتر T > ۰ تقسیم می‌شود:

$$\text{Softmax}_T(z_i) = \frac{e^{z_i/T}}{\sum_j e^{z_j/T}}$$

- وقتی $T = 1$: خروجی معمولی Softmax تولید می‌شود.
- وقتی $T > 1$: توزیع نرم‌تر می‌شود؛ مدل با اطمینان کمتری پیش‌بینی می‌کند.
- وقتی $T < 1$: توزیع تیزتر می‌شود؛ مدل با اطمینان بیشتری تصمیم می‌گیرد.

در مرحله کالیبراسیون، مقدار T روی مجموعه اعتبارسنجی (Validation) بهینه‌سازی می‌شود تا فاصله بین توزیع پیش‌بینی‌شده و واقعیت آماری (مثلاً با معیارهایی مانند Expected Calibration Error) کمینه شود.

مثال کاربردی: سیستم‌های تصمیم‌گیری در پزشکی

در یک سیستم تشخیص پزشکی با استفاده از شبکه عصبی کانولوشنی، خروجی مدل ممکن است احتمال "سرطان بودن" یا "سالم بودن" یک بافت را گزارش دهد. اگر مدل بدون کالیبره‌سازی Softmax مقدار ۹۸.۰ برای "سرطان" تولید کند، ولی در واقعیت تنها در ۸۰٪ موارد این پیش‌بینی درست باشد، سیستم تصمیم‌گیر ممکن است به‌شدت گمراه شود. با اعمال Temperature Scaling، این احتمال ممکن است به مقدار واقع‌گرایانه‌تری مانند ۸۰.۰ تنظیم شود، که منجر به تصمیم‌گیری قابل اعتمادتر توسط پزشک یا سیستم اتوماسیون می‌شود.

۱۱

در معماری‌های بسیار عمیق CNN، دو چالش اصلی وجود دارد: مشکل ناپدید شدن/انفجار گرادیان‌ها (vanishing/exploding gradients) و مشکل افت عملکرد با افزایش عمق شبکه (degradation problem). به‌طور خلاصه، هنگامی که شبکه‌ها عمیق می‌شوند، گرادیان‌های backpropagation در لایه‌های اولیه به‌طور تدریجی کاهش یا افزایش می‌یابند که فرآیند یادگیری را ناامن و ناپایدار می‌سازد. علاوه بر این، افزایش عمق منجر به کاهش عملکرد شبکه در برخی موارد می‌شود چرا که شباهت‌های اطلاعاتی و وابستگی‌های پیچیده بین لایه‌ها به‌طور ناکارآمد منتقل می‌گردد و مدل نتواند الگوهای کلی را به‌طور بهینه استخراج کند.

تکنیک‌هایی مانند skip connections در معماری ResNet با ارائه مسیرهای معادلی (identity mapping) برای انتقال، این مشکلات را برطرف می‌کنند. این اتصالات با ایجاد ارتباط مستقیم بین لایه‌های اولیه و نهایی شبکه، اجازه می‌دهند تا گرادیان‌ها بدون از دست دادن اطلاعات اصلی به عقب منتقل شوند؛ به عبارت دیگر، مدل قادر است وابستگی‌های عمیق‌تری را بدون ریسک از دست دادن اطلاعات مفید، یاد بگیرد. از منظر عملکردی، این ساختار باعث می‌شود که بهبودهای قابل توجهی در همگرایی، دقت و تعمیم‌پذیری مدل حاصل شود، به گونه‌ای که ساختارهای بسیار عمیق بدون مشکلات کاهش عملکرد، به‌راحتی قابل آموزش باشند.

۱۲ بخش الگوریتم ژنتیک

۱.۱۲

آ) یک کروموزم به طول تعداد دانشجویان در نظر میگیریم. در این کروموزوم خانه i نشان دهنده انتخاب دانشجوی iام است و مقدار آن نشان دهنده پروژه آن است.

$$\text{Chromosome} = [Pj_1, Pj_2, Pj_2, ..., Pj_s]$$

به عنوان مثال: $\text{Chromosome} = [1, 7, 9, 2, 3]$ یعنی دانشجوی اول پروژه اول، دانشجوی دوم پروژه هفتم و ...

ب)

دانشجو	۱ P	۲ P	۳ P	۴ P	۵ P	۶ P	۷ P	۸ P	۹ P	۱۰ P	۱۱ P	۱۲ P	۱۳ P
۱	۱۰۰۰	۱	۱۰۰۰	۱۰۰۰	۲	۱۰۰۰	۱۰۰۰	۳	۱۰۰۰	۴	۱۰۰۰	۱۰۰۰	۵
۲	۲	۱	۱۰۰۰	۱۰۰۰	۱۰۰۰	۳	۱۰۰۰	۱۰۰۰	۴	۵	۱۰۰۰	۱۰۰۰	۱۰۰۰
۳	۱۰۰۰	۲	۱۰۰۰	۱	۴	۱۰۰۰	۵	۱۰۰۰	۱۰۰۰	۱۰۰۰	۳	۱۰۰۰	۱۰۰۰
۴	۱۰۰۰	۱۰۰۰	۳	۱۰۰۰	۱۰۰۰	۱	۱۰۰۰	۲	۵	۴	۱۰۰۰	۱۰۰۰	۱۰۰۰
۵	۱	۴	۱۰۰۰	۱۰۰۰	۳	۱۰۰۰	۲	۱۰۰۰	۱۰۰۰	۱۰۰۰	۱۰۰۰	۱۰۰۰	۵
۶	۱۰۰۰	۱۰۰۰	۱۰۰۰	۲	۱۰۰۰	۳	۱۰۰۰	۱۰۰۰	۱	۱۰۰۰	۵	۴	۱۰۰۰
۷	۳	۱۰۰۰	۴	۱۰۰۰	۱۰۰۰	۱۰۰۰	۵	۱	۱۰۰۰	۲	۱۰۰۰	۱۰۰۰	۱۰۰۰
۸	۱۰۰۰	۳	۱۰۰۰	۵	۱	۲	۱۰۰۰	۱۰۰۰	۱۰۰۰	۱۰۰۰	۴	۱۰۰۰	۱۰۰۰
۹	۱۰۰۰	۱۰۰۰	۵	۱۰۰۰	۱۰۰۰	۱۰۰۰	۱	۲	۴	۱۰۰۰	۱۰۰۰	۱۰۰۰	۳

جدول ۱: جدول انتخاب دانشجویان

مقدار جریمه در این مثال ۱۰۰۰ در نظر گرفته شده است. **جمعیت اولیه**

Chromosome	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8	S_9
1	2	1	2	1	1	1	1	1	1
2	5	3	4	2	7	4	10	6	9
3	8	9	11	3	5	6	1	2	12
4	1	6	5	10	2	12	3	11	13
5	10	12	7	9	13	11	7	4	3

ج)

- اگر دو دانشجو یک پروژه را بردارند باید برای آن جریمه تعریف کنیم. (مثلا ۱۰۰۰)
- اگر دانشجویی نیز پروژه ای دارد که در ۵ اولولیت آن نبوده باید برای آن جریمه تعریف کنیم. (قبلا تعریف شده است)
- حداکثر جریمه ممکن برابر است با وقتی که همه دانشجویان یک پروژه یکسان بردارند که در اولویت هیچ کدام نیست. $2*1000*|S|$
- همچنین باید تلاش شود تا دانشجویان اولویت های بالا تر خود برسند.
- تابع dupcnt تکراری ها را میشمارد.

تابع fitness زیر اینکار را انجام میدهد:

$$f(c) = \frac{1}{\sum_i r_{ij} + dupcnt(c)*1000} * 9 * 2000$$

این تابع همه شرایط لازم را دارد. و خروجی آن بین ۱ تا ۲۰۰۰ برای این مسئله می‌باشد.

```
student_preferences = [
[1000, 1, 1000, 1000, 2, 1000, 1000, 3, 1000, 4, 1000, 1000, 5],
[2, 1, 1000, 1000, 1000, 3, 1000, 1000, 4, 5, 1000, 1000, 1000],
[1000, 2, 1000, 1, 4, 1000, 5, 1000, 1000, 1000, 3, 1000, 1000],
[1000, 1000, 3, 1000, 1000, 1, 1000, 2, 5, 4, 1000, 1000, 1000],
[1, 4, 1000, 1000, 3, 1000, 2, 1000, 1000, 1000, 1000, 1000, 5],
[1000, 1000, 1000, 2, 1000, 3, 1000, 1000, 1, 1000, 5, 4, 1000],
[3, 1000, 4, 1000, 1000, 1000, 5, 1, 1000, 2, 1000, 1000, 1000],
[1000, 3, 1000, 5, 1, 2, 1000, 1000, 1000, 1000, 4, 1000, 1000],
[1000, 1000, 5, 1000, 1000, 1000, 1, 2, 4, 1000, 1000, 1000, 3]
]
chromosomes = [
[2, 1, 2, 1, 1, 1, 1, 1, 1],
[5, 3, 4, 2, 7, 4, 10, 6, 9],
[8, 9, 11, 3, 5, 6, 1, 2, 12],
[1, 6, 5, 10, 2, 12, 3, 11, 13],
[10, 12, 7, 9, 13, 11, 7, 4, 3]
]
def fitness(chromosome, preferences):
  total_cost = 0
  duplicate_penalty = 0
  seen = set()

  for student_index, project in enumerate(chromosome):
    project_index = project - 1
    cost = preferences[student_index][project_index]
    total_cost += cost

    if project in seen:
      duplicate_penalty += 1
    else:
      seen.add(project)

  penalty = duplicate_penalty * 1000
  adjusted_cost = total_cost + penalty

  fitness_value = (1 / adjusted_cost) * 9 * 2000
  return fitness_value

# Calculate fitness for all chromosomes
fitness_values = [fitness(ch, student_preferences) for ch in chromosomes]
for i, fval in enumerate(fitness_values, start=1):
  print(f"Fitness of Chromosome {i}: {fval:.4f}")


```

output:

```
    Fitness of Chromosome 1: 1.6350
    Fitness of Chromosome 2: 5.9701
    Fitness of Chromosome 3: 17.5610
    Fitness of Chromosome 4: 17.4757
    Fitness of Chromosome 5: 8.8279
```

د)

- ترکیب کروموزم سوم و چهارم

$p3 = [8, 9, 11, 3, 5, 6, 1, 2, 12]$
$p4 = [1, 6, 5, 10, 2, 12, 3, 11, 13]$

اگر از uniform crossover استفاده کنیم:

$c1 = [8, 6, 5, 3, 2, 12, 1, 11, 13] \rightarrow 580.6451$
$c2 = [1, 9, 11, 10, 5, 6, 3, 2, 12] \rightarrow 8.8932$

- ترکیب کروموزم دوم و سوم

$p2 = [5, 3, 4, 2, 7, 4, 10, 6, 9]$
$p3 = [8, 9, 11, 3, 5, 6, 1, 2, 12]$

اگر از uniform crossover استفاده کنیم:

$c3 = [8, 3, 11, 2, 5, 4, 7, 6, 9] \rightarrow 8.9020$
$c4 = [5, 9, 4, 3, 7, 6, 10, 2, 12] \rightarrow 17.6470$

ه) برگزیدگان:

c1	580.6451
c4	17.6470
p3	17.5610
p4	17.4757
c3	8.9020

برازندگی کل نسل قبلی : 51.4697
برازندگی کل نسل جدید : 642.2308

۲.۱۲

آ) نمایش مناسب برای کروموزوم‌ها: یک کروموزوم می‌تواند به‌صورت یک رشته از شناسه‌های مشتریان به همراه شناسه وسیله نقلیه در قالب لیست‌های مجزا نمایش یابد. برای مثال:

$$\text{Chromosome} = [[v_1 : c_3 \rightarrow c_5 \rightarrow c_1], [v_2 : c_2 \rightarrow c_6], [v_3 : c_4]]$$

که در آن v_i نمایانگر وسیله نقلیه و c_j نشان‌دهنده مشتری jام است. این نمایش هم اختصاص مشتریان به وسایل نقلیه و هم ترتیب ملاقات را مشخص می‌کند.

ب) تعریف اپراتور قیاسی با جریمه تابع هزینه به‌صورت زیر تعریف می‌شود:

$$f = \sum_{i=1}^{m_1} d_i + \lambda_1 \cdot P_{\text{capacity}} + \lambda_2 \cdot P_{\text{timewindow}}$$

که:

- d_i مسافت طی‌شده توسط وسیله نقلیه i است.
- P_{capacity} مجموع جریمه‌های ناشی از تجاوز ظرفیت وسایل نقلیه است.
- $P_{\text{timewindow}}$ مجموع جریمه‌های ناشی از تخطی پنجره زمانی مشتریان است.
- λ_1 و λ_2 ضرایب جریمه هستند.

ج) طراحی عملگرهای Crossover و Mutation

Crossover:

استفاده از عملگر **Route-Based Crossover (RBX)**:

- یک یا چند مسیر (route) تصادفی از والد اول انتخاب شده و بدون تغییر به فرزند منتقل می‌شوند.
- مشتریانی که در این مسیرها نیستند، از والد دوم و به ترتیبی که در والد دوم آمده‌اند، در جای خالی باقی‌مانده قرار می‌گیرند به‌گونه‌ای که تکراری نباشند.

مثال:

Parent 1: $[[v_1 : c_1 \to c_2], [v_2 : c_3 \to c_4]]$
Parent 2: $[[v_1 : c_4 \to c_1], [v_2 : c_2 \to c_3]]$

فرض کنید مسیر $[v_1 : c_1 \to c_2]$ از والد اول انتخاب شود. مشتریان باقی‌مانده (c_3, c_4) طبق ترتیب والد دوم به فرزند افزوده می‌شوند، نتیجه:

$$\text{Offspring:}[[v_1 : c_1 \to c_2], [v_2 : c_4 \to c_3]]$$

Mutation:

استفاده از دو عملگر جهش:

- **Mutation: Swap** تعویض موقعیت دو مشتری در یک مسیر یا بین مسیرهای مختلف.
- **Mutation: Relocate** جابجایی یک مشتری از مسیر فعلی‌اش به موقعیتی دیگر در همان مسیر یا مسیر متفاوت.

مثال Swap:

$$[v_1 : c_1 \to c_2 \to c_3] \Rightarrow [v_1 : c_1 \to c_3 \to c_2]$$

مثال Relocate:

$$[v_1 : c_1 \to c_2], [v_2 : c_3] \Rightarrow [v_1 : c_2], [v_2 : c_3 \to c_1]$$

د) انتخاب والد و بازمانده

برای انتخاب والدین از روش (Tournament Selection) استفاده می‌شود:

- k کروموزوم به‌صورت تصادفی انتخاب شده و بهترین آن‌ها بر اساس تابع هزینه به‌عنوان والد انتخاب می‌شود.

برای انتخاب بازماندگان از روش **Elitism** استفاده می‌شود:

- درصدی از بهترین کروموزوم‌های نسل فعلی مستقیماً به نسل بعد منتقل می‌شوند.
- بقیه جمعیت از بین فرزندان تولیدشده انتخاب می‌گردند.

ه) شرط توقف

شرایط توقف می‌تواند یکی از موارد زیر باشد:

- رسیدن به تعداد نسل معین G_{max}.
- عدم بهبود تابع هزینه برای k نسل متوالی.
- رسیدن به مقدار حداقل مطلوب برای تابع هزینه.