یادگیری ژرف

تمرین اول

جواد راضی (۴۰۱۲۰۴۳۵۴)

سوال اول: Dilated Convolution

۱) ابعاد خروجی هر یک از لایه‌ها:

$$O_1 = (i - k_1 + 1) \times (i - k_1 + 1) \times d_1 \rightarrow$$
$$O_2: (i - k_1 + 1 - k_2 + 1) = (i - k_1 - k_2 + 2) \rightarrow O_2$$
$$= (i - k_1 - k_2 + 2) \times (i - k_1 - k_2 + 2) \times d_2 \rightarrow$$
$$O_3:\ considering\ dilation\ n: out\ filter\ covers\ j\ to\ j + \ (n-1) \times k_{filter} \rightarrow O_3$$
$$= \left(O_{2_1} - [(n-1) \times (k_3 - 1) + k_3] + 1\right)$$
$$\times\ \left(O_{2_2} - [(n-1) \times (k_3 - 1) + k_3] + 1\right) \times d_3 \rightarrow$$
$$O_3 = (i - k_1 - k_2 + 2 - nk_3 - n) \times (i - k_1 - k_2 + 2 - nk_3 - n) \times d_3$$

۲)

- در لایه اول، هر نورون به صورت عادی محدوده دید k-1 را خواهد داشت، که وقتی پارامتر گسترش داریم، این نرخ در آن ضرب می‌شود. در انتها نیز خود نورون هم محسوب شده و یکی به جمع افزوده می‌گردد:

$$RF_1 = (k-1).d_1 + 1$$

به همین طریق برای نورون‌های لایه‌های دوم و سوم هم داریم:

$$RF_2 = (k-1).d_2 + 1$$
$$RF_3 = (k-1).d_3 + 1$$

آن‌چه که در اینجا نادیده گرفته‌شد، لایه‌های قبلی بودند. محدوده دید هر واحد در یک لایه، از لایه قبل تاثیر می‌پذیرد. برای لایه خروجی، محدوده دید برابر جمع سه محدوده دید بدست آورده خواهد بود:

$$Field\ of\ View = RF_1 + RF_2 + RF_3 = (k-1).(d1 + d2 + d3) + 3$$

سوال دوم: انتشار به عقب

الف)

از قواعد مشتق زنجیره‌ای استفاده می‌کنیم. در خصوص مشتق تابع سیگموید، می‌دانیم که :

$$\frac{\partial\sigma(x)}{\partial x} = \sigma(x).\big(1-\sigma(x)\big)$$

$$\frac{\partial Loss}{\partial C_{i,j}} = \frac{\partial Loss}{\partial y'}\cdot\frac{\partial y'}{\partial z}\cdot\sum_{\text{l=1 to 9}}\cdot\frac{\partial z}{\partial A_l}\cdot\frac{\partial A_l}{\partial C_{ij}}$$

$$= \frac{\partial Loss}{\partial y'}\,.y'(1-y').W_2\sum_{l=1}^{l=9}\frac{\partial\frac{1}{4}\big(C_{ij}+C_{(i+1,j)}+C_{i,j+1},C_{(i+1,j+1)}\big)}{\partial C_{ij}}$$

$$= \frac{\partial Loss}{\partial y'}\,.y'(1-y').W_2.\frac{1}{4}.9 = DV1$$

ب)

می‌دانیم:

$$C_{ij} = \sum_{k,p,q} W^1_{p,q,k}\,.X_{i-p,j-q,k} \rightarrow \frac{\partial C_{ij}}{\partial W^1_{m,n,k}} = \{p=m,q=n\}\sum_k X_{i-m,j-n,k} = DV2 \rightarrow$$

$$\frac{\partial Loss}{\partial W^1_{i,j,k}} = \sum_{m,n}\frac{\partial Loss}{\partial C_{mn}}\cdot\frac{\partial C_{mn}}{W^1_{i,j,k}} = \sum_{m=1}^{m=6}\sum_{n=1}^{n=6}DV1_{mn}\cdot\sum_{k=1}^{k=3}X_{m-i,n-j,k}$$

سوال سوم: کانولوشن عمقی جدایی‌پذیر

الف) حالت عادی

هر فیلتر، ۹ پارامتر دارد. چهار کانال خروجی داریم و تصویر ورودی نیز سه کاناله است. بنابراین، ۱۲ فیلتر خواهیم داشت. در نتیجه تعداد پارامترهای قابل یادگیری در این حالت برابرست با ۱۲*۹=۱۰۸

ب) حالت لایه عمقی جدایی‌پذیر

۱. اجرای کانولوشن عمقی:

هر فیلتر ۹ پارامتر دارد، و در این حالت در این مرحله ۳ فیلتر داریم (به ازای هر کانال)، بنابراین، این مرحله ۲۷ پارامتر قابل یادگیری دارد.

۲. اجرای کانولوشن نقطه‌ای:

۴ فیلتر داریم و هر کدام ۳ پارامتر قابل یادگیری دارند (یکی برای هر کانال ورودی اولیه). بنابراین این مرحله ۱۲ پارامتر یادگیری دارد.

در نتیجه، کل پارامترهای قابل یادگیری در حالت عمقی جدایی‌پذیر، برابر ۳۹ پارامتر می‌باشند که به طرز قابل‌توجهی نسبت به حالت استاندارد کم‌تر است.

سوال چهار: Pooling Layers

۱)

فرض کنیم که خروجی دارای ابعاد $o_w \times o_h \times c$ باشد. می‌دانیم که برای بدست آوردن هر یک از عناصر این tensor، تعدادی عمل باید انجام دهیم که بستگی به اندازه پنجره pooling دارد، که برابر $p_w \times p_h$ می‌باشد. اکنون باید ابعاد خروجی را بدست آورده، و در این مقدار ضرب کنیم. مقدار o_w برابر با $w - p_w + 1$ بوده و مقدار o_h نیز برابر با $w - p_h + 1$ می‌باشد. در نهایت، برای محاسبه اردر پیچیدگی محاسباتی، داریم:

$$O(Pooling) = O\left(\frac{[h - p_h].[w - p_w].c \times p_w.p_h}{s_w.s_h}\right) = O\left(\frac{h.w.c.p_w.p_h}{s_w.s_h}\right)$$

در پیوست تمرین فایل پایتونی که عملیات Pooling را انجام می‌دهد پیاده‌سازی شده، و تعداد محاسبات در تئوری و در عمل مقایسه شده‌اند. مشاهده می‌شود که با ورودی بزرگ، این اعداد بسیار نزدیک به هم هستند.

```python
if __name__ == "__main__":
    input = np.random.rand(1000, 1000, 3)
    p_w, p_h, s_w, s_h = 8, 6, 4, 3
    output, complexity = pooling(input, p_w, p_h, s_w, s_h)
    theoretical_complexity = float(1000 * 1000 * 3 * p_w * p_h) / float(s_w * s_h)
    print(
        output.shape,
        complexity,
        theoretical_complexity,
        float(complexity) / theoretical_complexity,
    )

```

PROBLEMS 92 OUTPUT DEBUG CONSOLE TERMINAL POLYGLOT NOTEBOOK PORTS JUPYTER GITLENS COMME

```
PS C:\Users\jrazi\Desktop\repo\deep-learning-assignments-fall-2023\HW2> & C:/Python312/python.ex
assignments-fall-2023/HW2/Solution/pooling_complexity.py
(249, 332, 3)
PS C:\Users\jrazi\Desktop\repo\deep-learning-assignments-fall-2023\HW2> & C:/Python312/python.ex
assignments-fall-2023/HW2/Solution/pooling_complexity.py
(249, 332, 3) 12152196
PS C:\Users\jrazi\Desktop\repo\deep-learning-assignments-fall-2023\HW2> & C:/Python312/python.ex
assignments-fall-2023/HW2/Solution/pooling_complexity.py
(249, 332, 3) 12152196 12000000.0 1.012683
```

۲)

در max pooling، برای هر پنجره، پیکسلی که بیشترین intensity را دارد را به عنوان پیکسل نماینده آن پنجره بر می‌گزینیم. در Average Pooling، از مقادیر تک‌تک پیکسل‌های پنجره میانگین گرفته و این میانگین را به عنوان پیکسل نماینده آن پنجره انتخاب می‌نماییم. اولین روش، انتظار می‌رود تصویری بدهد که Saturation بالایی دارد. در مواقعی که بک‌گراند سیاه است، روش max pooling می‌تواند عملکرد مناسبی داشته‌باشد، چرا که به دنبال Bold ترین پیکسل‌های تصویر است. در صورتی که تصویر نویزی باشد و نویز از نوع salt and pepper باشد، این فیلتر نویزها را بازتقویت خواهد کرد. در مقابل، روش average pooling انتظار می‌رود تصویری به ما دهد که نویز کم‌تری نسبت به max pooling دارد، و smoothتر است؛ اما ممکن است تصویر اندکی مات یا تار باشد. از لحاظ پیچیدگی محاسباتی، دو روش اردر یکسانی دارند، اما از نظر استفاده از حافظه، روش average pooling نیاز به حافظه‌ای به اندازه خانه‌های پنجره دارد.

۳)

بله؛ برای Average Pooling، اگر سایز پنجره m x m باشد، می‌توان فیلتری قرار داد که مقدار هر خانه، برابر $\frac{1}{m}$ باشد. با اینکار، هر بار اعمال کانولوشن با فیلتر روی قسمتی از تصویر، معادل اعمال average pooling است.

برای max pooling، صرفا با استفاده از فیلترها نمی‌توان کاری کرد. چرا که این کار نیاز به نگه‌داشتن یک مقدار بیشینه در حافظه دارد، که با نحوه عملکرد کانولوشن، که مشابه لغزش یک تابع روی تابع دیگر است، در تضاد می‌باشد. (به بیان دیگر، می‌توان گفت max pooling یک عملگر غیرخطی است، در حالی که کانولوشن خطی می‌باشد.)

۴)

$$\max(a, b) = ReLU(a-b) + b$$

در این معادله، اگر a بزرگ‌تر باشد، عبارت سمت چپ خودش است و با جمع با b، a بدست می‌آید. اگر کوچکتر یا مساوی باشد، حاصل ReLU برابر صفر است، و b بدست می‌آید.

برای پیاده‌سازی max pooling با ReLU، می‌توان از رابطه‌ای که در قسمت قبل بدست آوردیم چنین استفاده‌ای کرد:

$$\max\left(A_{00}, A_{01}, \ldots., A_{p_h p_w}\right) = \max\left(\mathrm{A}_{00}, \max\left(\mathrm{A}_{01}, \ldots, \mathrm{A}_{\mathrm{p_h} p_w}\right) = \cdots (recursive\ formula)\right.$$

با توجه به اینکه max دو عدد را با تابع ReLU توانستیم بنویسیم، بسته به ابعاد پنجره لایه Pooling، با چند لایه متوالی با تابع فعال‌ساز ReLU، می‌توان max pooling را پیاده‌سازی کرد. البته احتمالا راه کارامدتری نیز باشد، همچون استفاده از فیلترهای متعدد که ورودی را در جهات مختلف شیفت می‌دهند، که می‌توانند به محاسبه ماکسیمم با ReLU کمک کنند.

۵)

با softmax نمی‌توان این کار را انجام داد، چرا که خروجی softmax یک توزیع احتمالات است که جمع آن یک می‌باشد. اگر تابع softmax را روی پنجره‌ای اعمال کنیم، خروجی که خواهیم داشت تعداد احتمال خواهند بود که مقدار بیشینه، بالاترین احتمال را خواهد داشت، اما این با همان صورت مسئله اول تفاوتی ندارد و در عمل نمی‌توان با آن max pooling را پیاده‌سازی کرد.