

دانشکده مهندسی برق و کامپیوتر

استاد درس:

دستورکار آزمایشگاه هوش محاسباتی
جلسه ؟

اهداف این جلسه

شما در این جلسه یاد خواهید گرفت که :

- مراحل پیش^۱ را توضیح دهید.
- دو نوع مختلف از عملیات تجميع^۲ را پیاده سازی کنید.
- اجزای استفاده شده در یک شبکه عصبی پیشی^۳ (مانند لایه گذاری^۴، گام^۵، فیلتر و ...) و دلیل استفاده از آنها را بشناسید.
- یک شبکه عصبی پیشی بسازید

نشانه گذاری:

- نشانه ی $[l]$ نشان دهنده ی شیء ای است که در لایه ی l - اُم حضور دارد.
- نشانه ی (i) نشان دهنده ی شیء ای در نمونه ی i - اُم است.
- نشانه ی i نشان دهنده ی i - اُمین ورودی بردار است.
- نشانه های n_H, n_W, n_C به ترتیب نشان دهنده ی طول، عرض و تعداد کانال های یک لایه ی داده شده می باشند. اگر بخواهیم که به یک لایه ی خاص اشاره کنیم، می توانیم از نشانه گذاری^۴ $n_H^{[l]}, n_W^{[l]}, n_C^{[l]}$ استفاده کنیم.
- نشانه های $n_{H_{prev}}, n_{W_{prev}}, n_{C_{prev}}$ نیز به ترتیب طول، عرض و تعداد کانال های لایه ی قبلی را نشان می دهند. اگر بخواهیم به یک لایه ی خاص اشاره کنیم می توانیم از $n_H^{[l-1]}, n_W^{[l-1]}, n_C^{[l-1]}$ استفاده کنیم.
- توجه داشته باشید که شما در این جلسه باید بر کتابخانه ی `numpy` تسلط نسبی داشته باشید.

۱ پکیج ها

در قسمت اول، باید پکیج هایی که در این جلسه استفاده خواهید کرد را فراخوانی کنید.

- `NumPy` پکیج اصلی محاسبات علمی در پایتون است.
- `matplotlib` یک کتابخانه برای مصورسازی نتایج در پایتون است.
- از `np.random.seed(1)` برای هماهنگ نگه داشتن تمام توابع تصادفی استفاده خواهیم کرد. این کار به درجه بندی کار شما کمک خواهد کرد.

کد مربوط به این قسمت را می توانید در فایل ژوپیتِر مربوط به این جلسه، در قسمت `Packages - 1` مشاهده کنید.

۲ دورنمای تمرین های این جلسه

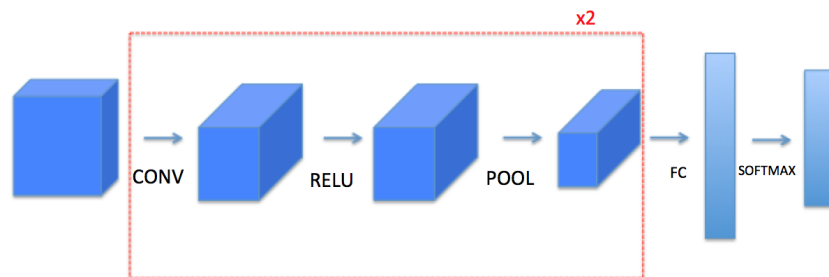
در ادامه شما بلوک هایی از شبکه عصبی پیشی را خواهید ساخت. هر تابعی که پیاده سازی می کنید، راهنماهایی برای کمک کردن به شما برای گذراندن مراحل زیر خواهد داشت:

- توابع پیشی شامل^۱
- لایه گذاری با صفر^۶
- پنجره ی پیشی^۷

^۱convolution
^۲pooling
^۳Network Neural Convolutional
^۴padding
^۵stride
^۶zero padding
^۷convolve window

- پیش‌پیش‌رو^۸
- پیش‌پس‌رو^۹
- توابعِ تجميعِ شامل:
- تجميعِ پیش‌رو^{۱۰}
- ساختِ ماسک
- مقدارِ توزیع^{۱۱}
- توزیعِ پس‌رو^{۱۲}

نکته: در این جلسه شما تمامی موارد بالا را با استفاده از NumPy، از صفر تا صد پیاده سازی خواهید کرد. در جلسه ی بعدی، موارد مشابه این توابع را در TensorFlow، برای پیاده سازی مدل زیر، یاد خواهید گرفت:

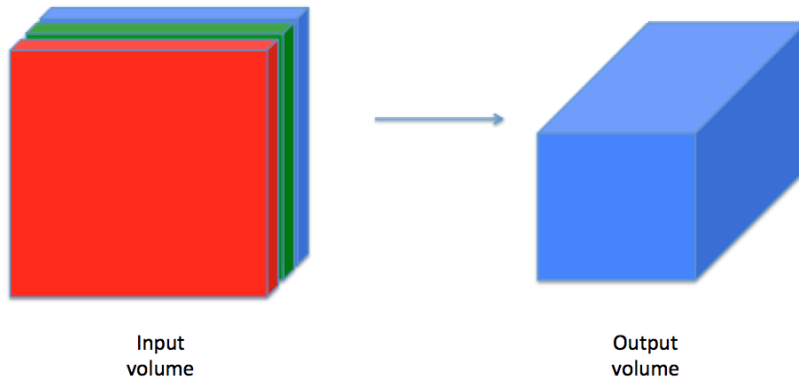


نکته: برای هر تابع پیش‌رو^{۱۳} یک تابع پس‌رو^{۱۴} ی متناظر نیز وجود دارد. به همین دلیل، در هر قدم از مرحله ی پیش‌رو، شما برخی از پارامترها را در یک حافظه ی کوتاه مدت ذخیره سازی خواهید کرد. این پارامترها برای محاسبه ی گرادیان‌ها در حین انتشارِ پس‌رو^{۱۵} استفاده خواهند شد.

convolution forward^۸
convolution backward^۹
pooling forward^{۱۰}
Distribute value^{۱۱}
pooling backward^{۱۲}
forward^{۱۳}
backward^{۱۴}
backpropagation^{۱۵}

۳ شبکه‌های عصبی پیچشی

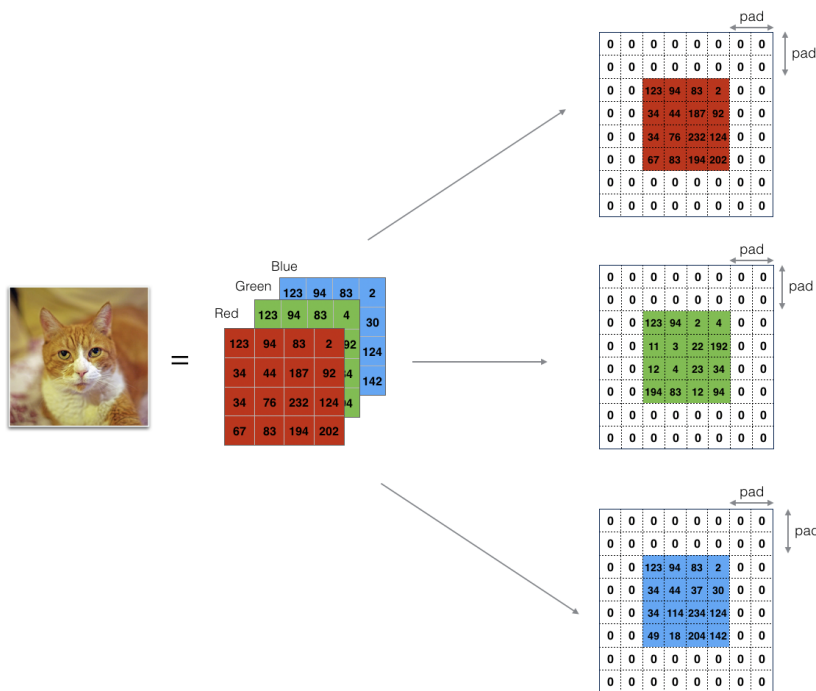
اگر چه چهارچوب های برنامه نویسی،^{۱۶} باعث ساده شدن استفاده از پیچش ها شده اند، اما آنها هنوز هم یکی از مفاهیم سخت در یادگیری عمیق محسوب می‌شوند. یک لایه ی پیچش ، همانطور که در شکل زیر نشان داده شده است، یک مقدار ورودی را به یک مقدار خروجی با سایزی متفاوت تبدیل می‌کند. در این قسمت، شما هر مرحله از لایه ی پیچش را خواهید ساخت. در ابتدا دو تابع کمک کننده^{۱۷} : یکی برای لایه‌گذاری



با صفر و دیگری برای محاسبه ی تابع پیچش را پیاده سازی خواهید کرد.

۱.۳ لایه‌گذاری با صفر

لایه‌گذاری با صفر، لایه ای از مقادیر صفر را در مرز های یک تصویر، اضافه می‌کند: فواید اصلی لایه‌گذاری در ذیل توضیح داده شده است:



شکل ۱: تصویر سه کاناله، همراه با لایه‌ای با مقدار ۲

- این کار به ما اجازه می‌دهد که بدون نیاز به کاهش دادن طول و عرض مقادیر، از یک لایه‌ی CONV استفاده کنیم. این عمل برای ساختن شبکه های عمیق تر، اهمیت پیدا می کند زیرا در غیر این صورت، طول و عرض به نسبت رفتن به لایه های عمیق تر، کاهش پیدا خواهند کرد.

^{۱۶} frameworks programming
^{۱۷} helper function

- این کار به ما کمک می‌کند که اطلاعاتی که در مرزهای تصاویر وجود دارند را حفظ کنیم. بدون لایه‌گذاری، مقادیر بسیار کمی در لایه‌ی بعدی، از پیکسل‌های مرزی تصویر، تاثیر می‌پذیرند.

تمرین اول : zero_pad

در فایل ژوپیتر این جلسه، در قسمت Exercise 1 - zero_pad، تابع `zero_pad(X, pad)` را که تمام تصاویر یک دسته مانند `X` را، با صفر، لایه‌گذاری می‌کند، پیاده‌سازی کنید. در این جا باید از `np.pad` استفاده کنید. این نکته را در نظر داشته باشید که اگر لایه‌ای با `pad = 1` برای بُعد دوم را در آرایه‌ی "a" با شکل (5,5,5,5) و همچنین `pad = 3` برای بُعد چهارم و `pad = 0` برای بقیه، بگذارید، می‌توانید به صورت زیر عمل کنید:

```
a = np.pad(a, ((0,0), (1,1), (0,0), (3,3),(0,0)), mode='constant', constant_values = (0,0))
```

۲.۳ یک مرحله از پیش‌پردازش

در این قسمت، شما باید یک مرحله از پیش‌پردازش را پیاده‌سازی نمایید که در واقع در آن، فیلتر را بر روی یک قسمت از ورودی اعمال می‌کنید. این کار برای ساخت یک واحد پیش‌بینی که :

- یک مقدار ورودی را می‌گیرد.
- یک فیلتر را بر هر قسمت از ورودی اعمال می‌کند.
- یک مقدار دیگر (که معمولاً اندازه‌ی آن با اندازه ورودی متفاوت است) را به عنوان خروجی می‌دهد.

استفاده خواهد شد.

در کاربردهای بینایی کامپیوتر، هر مقدار در ماتریس سمت چپ، متناظر یک تک مقدار پیکسل خواهد بود. شما یک فیلتر 3x3 را با تصویر، به هم پیش ۱۸ خواهید کرد، به این صورت که مقادیر آن را به صورت درایه ای با ماتریس اصلی، ضرب خواهید کرد و سپس آن‌ها را با هم جمع کرده و در نهایت یک بایاس به آن اضافه خواهید نمود. شما یک مرحله از پیش‌پردازش را پیاده‌سازی خواهید نمود که متناظر است با اعمال کردن یک فیلتر بر روی فقط یک قسمت از تصویر و گرفتن یک خروجی تک مقداری. در قسمت‌های بعدی، شما این تابع را بر روی قسمت‌های دیگر ورودی نیز اعمال خواهید کرد تا در نهایت یک عمل پیش‌پردازش کامل را پیاده‌سازی کرده باشید.

تمرین دوم: conv_single_step

در فایل این جلسه، در قسمت Exercise 2-conv_single_step، تابع `conv_single_step()` را پیاده‌سازی کنید. راهنمایی نکته: متغیر `b` به صورت یک بردار numpy منتقل خواهد شد. اگر شما یک مقدار اسکالر را با یک بردار numpy جمع کنید، خروجی یک بردار numpy خواهد بود. در مورد خاصی که بردار numpy حاوی یک تک مقدار باشد، شما می‌توانید آن را با `cast` کردن به یک مقدار float، به مقدار اسکالر تبدیل کنید.

۳.۳ شبکه‌های عصبی پیش‌بینی - گذر پیش‌رو

در گذر پیش‌رو ۱۹ شما تعداد زیادی از فیلترها را گرفته و آنها را بر روی ورودی، اعمال خواهید کرد. هر پیش‌بینی، به شما یک ماتریس دو بُعدی به عنوان خروجی می‌دهد. سپس شما باید این خروجی‌ها را بر روی هم قرار داده و یک مقدار سه بُعدی را بسازید : تصویر پنج

تمرین سوم: conv_forward

در فایل این جلسه، در قسمت Exercise 3-conv_forward، تابع `conv_forward(A_prev, W, b, hparameters)` را به منظور به هم پیش‌بینی فیلترهای `W` بر روی یک ورودی فعال‌سازی `A_prev`، پیاده‌سازی کنید. این تابع ورودی‌های زیر را دریافت می‌کند:

convolve^{۱۸}
Forward Pass^{۱۹}

• `A_prev` ، فعالسازهای خروجی ای که از لایه ی قبلی حاصل شده اند. (برای دسته ای از `m` ورودی)

• وزن هایی که با `W` نشان داده شده اند و اندازه ی فیلتر، `f` در `f` است.

• بردارِ بایاس `b` ، که هر فیلتر بایاس مخصوص به خودش را دارد.

شما همچنین به هایپرپارامتر هایی مانند `padding` و `stride` نیز دسترسی دارید.

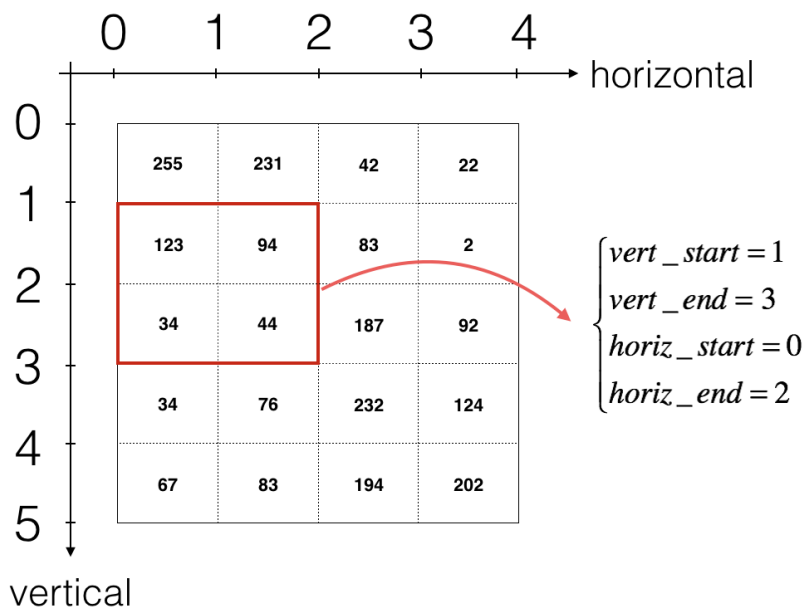
تصویر ششم
راهنمایی:

• برای انتخاب یک قطعه ی 2×2 از گوشه ی بالا سمت چپِ ماتریسِ `a_prev` ، (ابعاد: $(5, 5, 3)$) ، شما می بایست مانند قطعه کد زیر عمل کنید:

```
a_slice_prev = a_prev[0:2,0:2,:]
```

توجه کنید که این کار، یک تکه ی سه بُعدی که ارتفاع آن ۲، عرض آن ۲ و عمق آن ۳ است را به ما می دهد که عمق همان تعداد کانال ها می باشد. هنگامی که `a_slice_prev` را تعریف می کنید، استفاده از نمایه های `start/end` سودمند خواهد بود.

• برای تعریف کردن `a_slice` احتمالا در ابتدا به تعریف گوشه های آن : `vert_start` ، `vert_end` ، `horiz_start` و `horiz_end` نیاز پیدا خواهید کرد. به شما کمک خواهد کرد که گوشه ها را با استفاده از `f` ، `w` ، `h` و `s` در کد مربوط به این قسمت، پیدا کنید.



شکل ۲: تعریف یک تکه با استفاده از آغاز/پایان عمودی و افقی با یک فیلتر 2×2 ، این تصویر یک تک کانال را نشان می دهد.

یادآوری:

فرمول هایی که مربوط به قالب خروجی پیچش نسبت به قالب ورودی آن هستند، در ذیل آمده است:

$$n_H = \left\lfloor \frac{n_{H_{prev}} - f + 2 \times pad}{stride} \right\rfloor + 1$$

$$n_W = \left\lfloor \frac{n_{W_{prev}} - f + 2 \times pad}{stride} \right\rfloor + 1$$

n_C = تعداد فیلتر های استفاده شده در کانولوشن

راهنمایی:

indexes^{۲۰}

- از تکه کردن بردارها (مثلا `varname[0:1, :, 3:5]`) برای متغیرهای `a_prev_pad` ، `W` و `b` استفاده کنید.
 - کد استارتر تابع را کپی و آن را خارج از تعریف تابع در یک سلول جداگانه، اجرا کنید
 - چک کنید که زیرمجموعه‌ی هر آرایه و بُعد آن، همانی باشد که انتظارش را دارید.
 - برای اینکه تصمیم بگیرید که
 - `vert_start` ، `vert_end` ، `horiz_start` و `horiz_end` را چگونه بدست بیاورید، یادتان باشد که این‌ها، نمایه‌های لایه‌ی قبلی هستند.
 - یک نمونه از لایه‌ی لایه‌گذاری شده‌ی قبلی (برای مثال 8×8) و لایه‌ی کنونی (لایه‌ی خروجی) (برای مثال 2×2) را ترسیم کنید.
 - نمایه‌های لایه‌ی خروجی با `h` و `w` نشان داده شده‌اند.
 - مطمئن شوید که `a_slice_prev` ارتفاع، عرض و عمق دارد
 - یادتان باشد که `a_prev_pad` زیرمجموعه‌ای از `A_prev_pad` است.
 - به این فکر کنید که از کدامیک باید در حلقه‌ها استفاده کنید.
- در نهایت یک لایه‌ی CONV همچنین باید شامل یک فعال ساز باشد، پس شما می‌توانید قطعه‌ی زیر را به کد خود اضافه کنید:

```
# Convolve the window to get back one output neuron
Z[i, h, w, c] = ...
# Apply activation
A[i, h, w, c] = activation(Z[i, h, w, c])
```

۴ لایه‌ی تجمع

وظیفه‌ی لایه‌ی تجمع (POOL) ، کاهش طول و عرض ورودی است. با این کار، حجم محاسبات کاهش می‌یابد و همچنین کمک می‌کند تا آشکارسازهای ویژگی با موقعیت خود در ورودی تغییرناپذیرتر شوند
دو نوع لایه‌ی تجمع عبارتند از:

- لایه Max-pooling: یک پنجره‌ی (f, f) را بر روی ورودی حرکت می‌دهد و مقدار بیشینه‌ی داخل پنجره را در خروجی ذخیره می‌کند.
- لایه Average-pooling: یک پنجره‌ی (f, f) را بر روی ورودی حرکت می‌دهد و مقدار متوسط مقادیر زیر پنجره را در خروجی ذخیره می‌کند

Max Pool

2	3	1	9
4	7	3	5
8	2	2	2
1	3	4	5

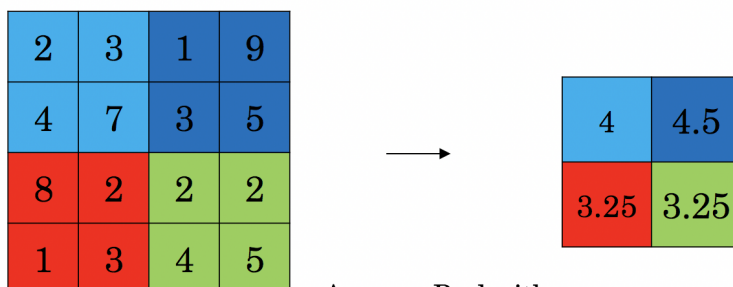
→

7	9
8	5

Max-Pool with a
2 by 2 filter and
stride 2.

این لایه‌ها، هیچ پارامتری برای استفاده در انتشار پس‌رو برای آموزش، ندارند.

Average Pool



Average Pool with
a 2 by 2 filter and
stride 2.

۱.۴ جمع پیش‌رو

در این قسمت می‌خواهیم در یک تابع، MAX-POOL و AVG-POOL را پیاده سازی کنیم.

تمرین چهارم: pool_forward

در فایل ژوپیتِر مربوط به این جلسه، در قسمت Exercise 4-pool_forward، گذرِ پَس‌رو را برای لایه‌ی جمع، پیاده سازی کنید. از راهنمایی‌هایی که در قسمت کامنت‌های کد نوشته شده است، استفاده کنید.

یادآوری: از آنجایی که padding نداریم، فرمول‌هایی که شکل ورودی را به شکل خروجی متصل می‌کنند، به صورت زیر تعریف می‌شوند:

$$n_H = \left\lfloor \frac{n_{H_{prev}} - f}{stride} \right\rfloor + 1$$

$$n_W = \left\lfloor \frac{n_{W_{prev}} - f}{stride} \right\rfloor + 1$$

$$n_C = n_{C_{prev}}$$

خروجی مورد انتظار در حالت اول:

```
Case 1: stride of 1
mode = max
A.shape = (2, 3, 3, 3)
A[1, 1] =
[[1.96710175 0.84616065 1.27375593]
 [1.96710175 0.84616065 1.23616403]
 [1.62765075 1.12141771 1.2245077 ]]

mode = average
A.shape = (2, 3, 3, 3)
A[1, 1] =
[[ 0.44497696 -0.00261695 -0.31040307]
 [ 0.50811474 -0.23493734 -0.23961183]
 [ 0.11872677  0.17255229 -0.22112197]]
```


خروجی مورد انتظار در حالت دوم:

```

Case 2: stride of 2
mode = max
A.shape = (2, 2, 2, 3)
A[0] =
[[[1.74481176 0.90159072 1.65980218]
  [1.74481176 1.6924546 1.65980218]]

 [[1.13162939 1.51981682 2.18557541]
  [1.13162939 1.6924546 2.18557541]]]

mode = average
A.shape = (2, 2, 2, 3)
A[1] =
[[[-0.17313416 0.32377198 -0.34317572]
  [0.02030094 0.14141479 -0.01231585]]

 [[0.42944926 0.08446996 -0.27290905]
  [0.15077452 0.28911175 0.00123239]]]

```

۵ انتشار پس رو در شبکه های عصبی پیچشی

در چهارچوب های مدرن یادگیری عمیق ، شما فقط نیاز دارید که گذر پیش رو را پیاده سازی کنید و خود چهارچوب، به قسمت گذر پس رو^{۲۱} رسیدگی می کند. پس اکثر مهندسين یادگیری عمیق، نیازی به عمیق شدن در مرحله ی گذر پس رو ندارند. گذر پس رو برای شبکه های پیچشی، مرحله ای پیچیده است. در قسمت بعدی قصد داریم با مرحله ی انتشار پس رو در یک شبکه ی پیچشی، آشنا شویم. در شبکه های پیچشی، شما می توانید مشتقات را با توجه به هزینه، برای به روزرسانی پارامترها، استفاده کنید.

۱.۵ لایه ی پیچشی - گذر پس رو

در این قسمت می خواهیم برای لایه ی CONV ، گذر رو به عقب را پیاده سازی نماییم.

۱.۱.۵ محاسبه ی dA

فرمول محاسبه ی dA با داشتن هزینه برای یک فیلتر خاص W_c و نمونه آموزشی داده شده، بصورت زیر می باشد:

$$dA+ = \sum_{h=0}^{n_H} \sum_{w=0}^{n_W} W_c \times dZ_{hw} \quad (1)$$

که در آن W_c فیلتر و dZ_{hw} مقدار اسکالر متناظر با گرادیان هزینه با توجه به خروجی لایه ی کانولوشن Z در سطر h -ام و ستون w -ام می باشد. نکته آن که در هنگام به روزرسانی dA در هر دفعه، شما همان فیلتر W_c را در مقدار متفاوتی از dZ ضرب می کنید. این کار بیشتر برای آن انجام می شود که در هنگام محاسبه ی انتشار پیش رو، هر فیلتر توسط یک a_slice متفاوت ضرب داخلی و جمع می شود. به همین جهت، هنگام محاسبه ی انتشار پس رو برای dA شما فقط گرادیان ها را به همه ی a_slice ها اضافه می کنید. در کد، داخل حلقه های for مناسب، این فرمول، به صورت زیر نوشته می شود:

```
da_prev_pad[vert_start:vert_end, horiz_start:horiz_end, :] += W[:, :, :, c] * dZ[i, h, w, c]
```

۲.۱.۵ محاسبه ی dW

فرمول محاسبه ی dW_C (مقدار متناظر مشتق یک فیلتر) با توجه به هزینه، بصورت:

$$dW_c+ = \sum_{h=0}^{n_H} \sum_{w=0}^{n_W} a_{slice} \times dZ_{hw} \quad (2)$$

^{۲۱}backward pass

می‌باشد که a_{slice} به slice ای مربوط است که هنگام ایجادِ فعال‌سازِ Z_{ij} استفاده شده است. از این رو، در نهایت ما گرادینِ W را با توجه به آن slice خواهیم داشت. از آنجایی که این همان W است، ما فقط همه‌ی گرادین‌ها را با هم جمع می‌کنیم تا dW را بدست بیاوریم. در کد، در حلقه‌های for مناسب، این فرمول بصورت زیر نوشته می‌شود:

```
dW[:, :, :, c] += a_slice * dZ[i, h, w, c]
```

۳.۱.۵ محاسبه‌ی db

در ذیل فرمول محاسبه‌ی db را با توجه به هزینه برای یک فیلترِ خاص W_c داریم:

$$db = \sum_h \sum_w dZ_{hw} \quad (۳)$$

همانطور که می‌دانید، db با جمع کردن dZ بدست می‌آید. در این مورد، شما صرفاً در حال جمع کردن بر روی تمام گرادین‌های خروجیِ کانولوشن (Z) با توجه به هزینه، هستید. در حلقه‌های for مناسب، این فرمول به صورت زیر نوشته می‌شود:

```
db[:, :, :, c] += dZ[i, h, w, c]
```

تمرین پنجم conv_backward

در فایل این جلسه، در قسمتِ Exercise 5-conv_backward، تابع `conv_backward` را تکمیل کنید. شما باید بر روی تمامی نمونه‌های آموزشی، فیلترها، طول و عرض‌ها، عمل جمع را انجام دهید. سپس باید مشتقات را با استفاده از فرمول‌های ۱، ۲ و ۳ که در بالا نوشته ایم، بدست آورید

خروجی مورد انتظار:

```
dA_mean = 1.45243777754
dW_mean = 1.72699145831
db_mean = 7.83923256462
```

۲.۵ لایه‌ی تجمیع - گذرِ پس‌رو

در ادامه، گذرِ پس‌رو را برای لایه‌ی تجمیع پیاده سازی خواهیم کرد. برای این کار از لایه‌ی MAX-POOL شروع خواهیم کرد. اگرچه یک لایه‌ی تجمیع، هیچ پارامتری برای به‌روزرسانی هنگام انتشارِ پس‌رو ندارد، شما همچنان نیاز دارید که گرادین‌ها را در طول لایه‌ی تجمیع، انتشارِ پس‌رو کنید تا بتوانید گرادین‌ها را برای لایه‌هایی که قبل از لایه‌ی تجمیع می‌آیند، محاسبه کنید.

۱.۲.۵ تجمیعِ بیشینه - گذرِ پس‌رو

قبل از آنکه سراغ انتشارِ پس‌رو برای لایه‌ی تجمیع برویم، شما باید تابعِ کمکی `create_mask_from_window()` را بسازید که بصورت زیر کار می‌کند:

$$X = \begin{bmatrix} 1 & 3 \\ 4 & 2 \end{bmatrix} \rightarrow M = \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix} \quad (۴)$$

همانطور که می‌بینید، این تابع یک ماتریسِ mask تولید می‌کند تا مشخص کند که مقدار بیشینه‌ی ماتریس را در کجا نگهداری کند. مقدارِ صحیح (۱) محل مقدار بیشینه را در ماتریسِ X معین می‌سازد و بقیه مقادیر، صفر هستند.

پیاده سازی create_mask_from_window

در فایل مربوط به این جلسه، در قسمت Exercise 6 - create_mask_from_window تابع `create_mask_from_window()` را پیاده سازی کنید. این تابع در هنگام جمعِ پس رو، مفید واقع خواهد شد.

- `np.max()` ممکن است مثر ثمر واقع شود، این تابع، بیشینه ی یک بردار را محاسبه می کند.
- اگر شما ماتریس X و مقدار اسکالر x را داشته باشید، $A = (X == x)$ یک ماتریس A با اندازه ای مشابه X را بر می گرداند، به طوری که:

$$A[i, j] = \text{if True } X[i, j] = x$$

$$A[i, j] = \text{if False } X[i, j] \neq x$$

- در اینجا شما نیاز نیست که مواردی که در یک ماتریس، تعداد بیشینه ها بیشتر از یک عدد باشد را در نظر بگیرید.

خروجی مورد انتظار:

```
x =
[[ 1.62434536 -0.61175641 -0.52817175]
 [-1.07296862  0.86540763 -2.3015387 ]]
mask = [[ True False False]
 [False False False]]
```

چرا موقعیت بیشینه را ردیابی کنیم؟ زیرا این مقدار ورودی است که در نهایت بر خروجی و در نتیجه هزینه تأثیر می گذارد. انتشارِ پس رو، محاسبه ی گرادیان ها با توجه به هزینه، است، بنابراین هر چیزی که بر هزینه نهایی تأثیر می گذارد باید دارای گرادیان غیر صفر باشد. پس انتشارِ پس رو، گرادیان را به سمت این مقدار خاص که بر روی هزینه تأثیر گذاشته است، رو به عقب، منتشر می کند.

۲.۲.۵ جمع متوسط - گذرِ پس رو

در جمع بیشینه، برای هر پنجره ی ورودی، تمامی تأثیراتِ روی خروجی، از یک مقدارِ ورودی که همان مقدار بیشینه بود، می آمد. در جمع متوسط^{۲۲}، هر جزء از پنجره ی ورودی، تأثیر مشابهی نسبت به بقیه بر روی خروجی دارد، پس برای پیاده سازیِ پس رو، شما ابتدا باید تابع کمکی ای که این امر را بازتاب دهد، پیاده سازی کنید. برای مثال، اگر ما بر روی گذرِ پیش رو، جمع متوسط را با استفاده از یک فیلتر 2×2 انجام بدهیم، آنگاه ماسکی که برای گذرِ پس رو استفاده خواهید کرد، به صورت زیر، درخواهد آمد:

$$dZ = 1 \rightarrow dZ = \begin{bmatrix} 1/4 & 1/4 \\ 1/4 & 1/4 \end{bmatrix} \quad (5)$$

نتیجه آنکه هر نقطه در ماتریس dZ ، به طور مساوی بر روی خروجی، تأثیر خواهد گذاشت.

۳.۵ تمرین هفت distribute_value

در فایل این جلسه، در قسمت Exercise 7-distribute_value تابع `distribute_value(dz, shape)` را تکمیل کنید. این تابع به طور مساوی، مقدار dZ را بر روی مقادیر ماتریسی با بُعد `shape` پخش خواهد کرد. راهنمایی

خروجی مورد نظر:

```
distributed_value = [[ 0.5 0.5]
 [ 0.5 0.5]]
```

۱۰.۳.۵ جمع یندی نهایی : جمعِ پس‌رو

اکنون شما تمامی پیشنیازها را برای محاسبه‌ی پخشِ پس‌رو بر روی یک لایه‌ی جمع در اختیار دارید.

تمرین هشتم pool_backward

در فایل این جلسه، در قسمتِ Exercise 8-pool_backward تابعِ pool_backward را در هر دو حالتِ "max" و "average" پیاده سازی کنید. شما از ۴ حلقه‌ی for برای گذر در طول نمونه های آموزشی، طول، عرض و کانال ها، استفاده خواهید کرد. شما باید از یک متغیر شرطی if/elif برای کنترل بین مقدار بیشینه یا مقدار متوسط، استفاده کنید. اگر این مقدار، برابر مقدار متوسط بود، باید از تابع distribute_value() که در تمرین های قبلی پیاده سازی کردید، برای ساختن یک ماتریس با شکلی مشابه با a_slice استفاده کنید. در غیر این صورت، حالت بر روی max است و شما باید یک ماسک با استفاده از create_mask_from_window() استفاده کنید و آن را در مقدار متناظرِ dA ضرب کنید.

خروجی مورد نظر:

```
mode = max:
mean of dA =
0.145713902729
dA_prev[1,1] = [[ 0. 0. ]
[ 5.05844394 -1.68282702]
[ 0. 0. ]]

mode = average
mean of dA =
0.145713902729
dA_prev[1,1] = [[ 0.08485462 0.2787552 ]
[ 1.26461098 -0.25749373]
[ 1.17975636 -0.53624893]]
```