شبکه‌های عصبی و یادگیری عمیق

سید ناصر رضوی n.razavi@tabrizu.ac.ir

۱۳۹۶

معرفی

n.razavi@tabrizu.ac.ir

- سید ناصر رضوی
  - عضو هیئت علمی گروه مهندسی کامپیوتر
  - دانشکده مهندسی برق و کامپیوتر، دانشگاه تبریز
  - سرپرست آزمایشگاه هوش محاسباتی و یادگیری ماشین

- زمینه‌های پژوهشی
  - یادگیری ماشین و یادگیری عمیق
  - پردازش زبان‌های طبیعی، ترجمه ماشینی
  - پردازش تصویر و بینایی ماشین
  - خودروهای هوشمند و رباتیک

پروژه‌ها: حوزه یادگیری عمیق

- پردازش زبان طبیعی.
  - سامانه مترجم چندزبانه [فارسی، انگلیسی و فرانسه]
  - توسعه پیکره موازی فارسی و انگلیسی
  - سامانه پژوهش‌یار
- پردازش تصویر.
  - جویشگر تصویر مبتنی بر محتوا
  - جویشگر چهره
  - سامانه ترافیک و تشخیص خودکار جرایم رانندگی [راهنمایی و رانندگی تبریز]

سامانه مترجم

ترجمه آنلاین

کلمه یا متن مورد نظر را وارد کرده و روی "ترجمه کن" کلیک نمایید

فارسی

کشورهای همسایه جهان از جمله متحدان آمریکا و رقبای رقیب در تبریک گفتن به عراق برای پیروزی اخیر خود بر روی گروه شبه نظامی دولت اسلام گرا در دژ سابق خود گرد هم آمدند

ترجمه کن

انگلیسی

Countries around the world including US allies and rivals have come together in congratulating Iraq for its recent victory over the Islamic State militant group ISIS in its former stronghold

سامانه مترجم

Assad and his Russian ally President Vladimir Putin vehemently denied the role of the Syrian government in the attacks and the incident led to a falling out between Washington and Moscow over Syria

ترجمه انسانی

اسد و ولادمیر پوتین رئیس جمهور روسیه با قاطعیت استفاده ارتش سوریه از جنگ افزار شیمیایی را انکار کردند و حمله موشکی آمریکا به سوریه، مناسبات آمریکا و روسیه را بر سر مسایل سوریه تیره کرد

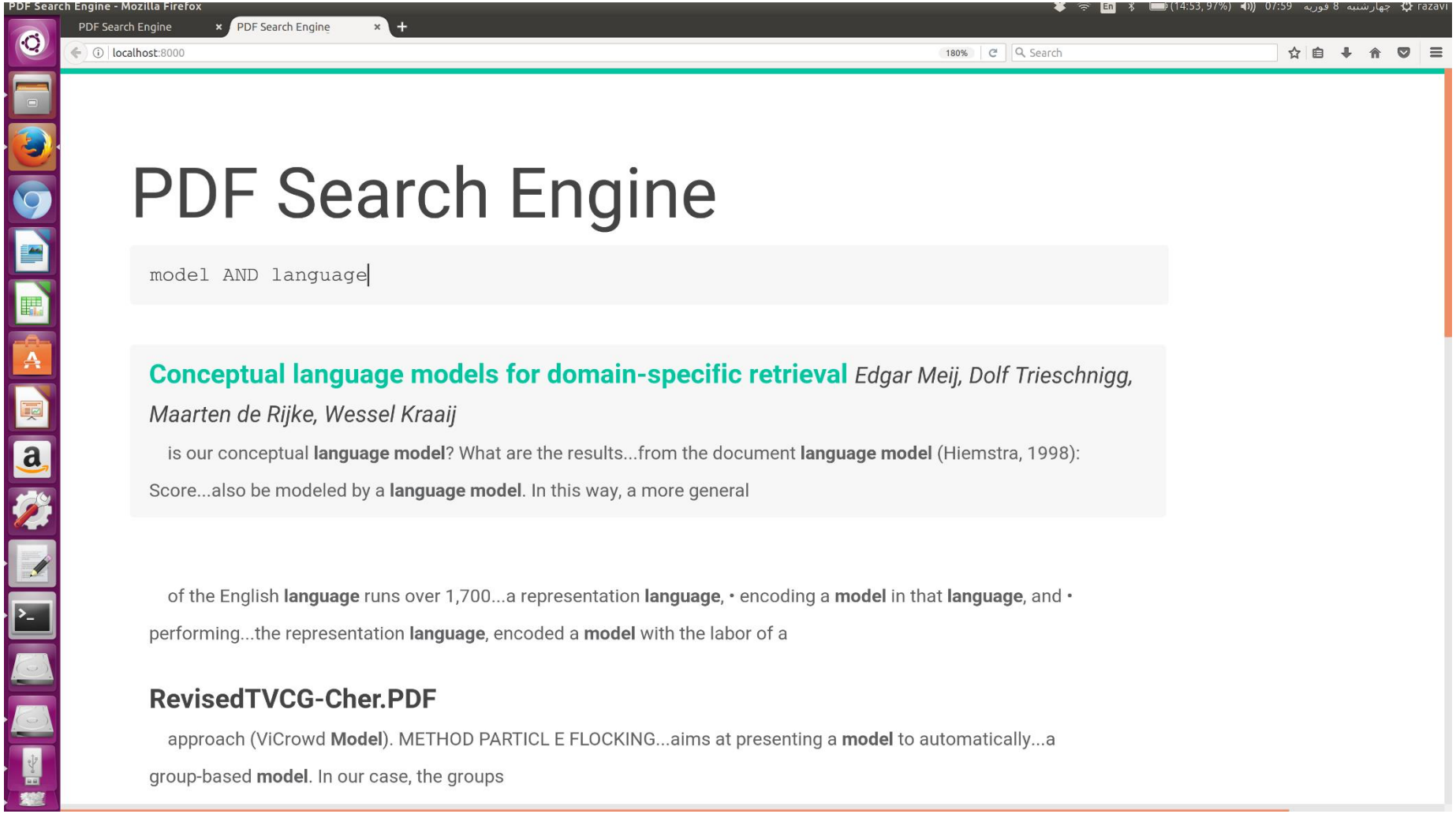
ترجمه توسط سامانه دانشگاه تبریز

اسد و رئیس جمهور روسیه ولادمیر پوتین به شدت نقش دولت سوریه در حملات را تکذیب کردند و حادثه منجر به سقوط بین واشنگتن و مسکو بر سر سوریه شد

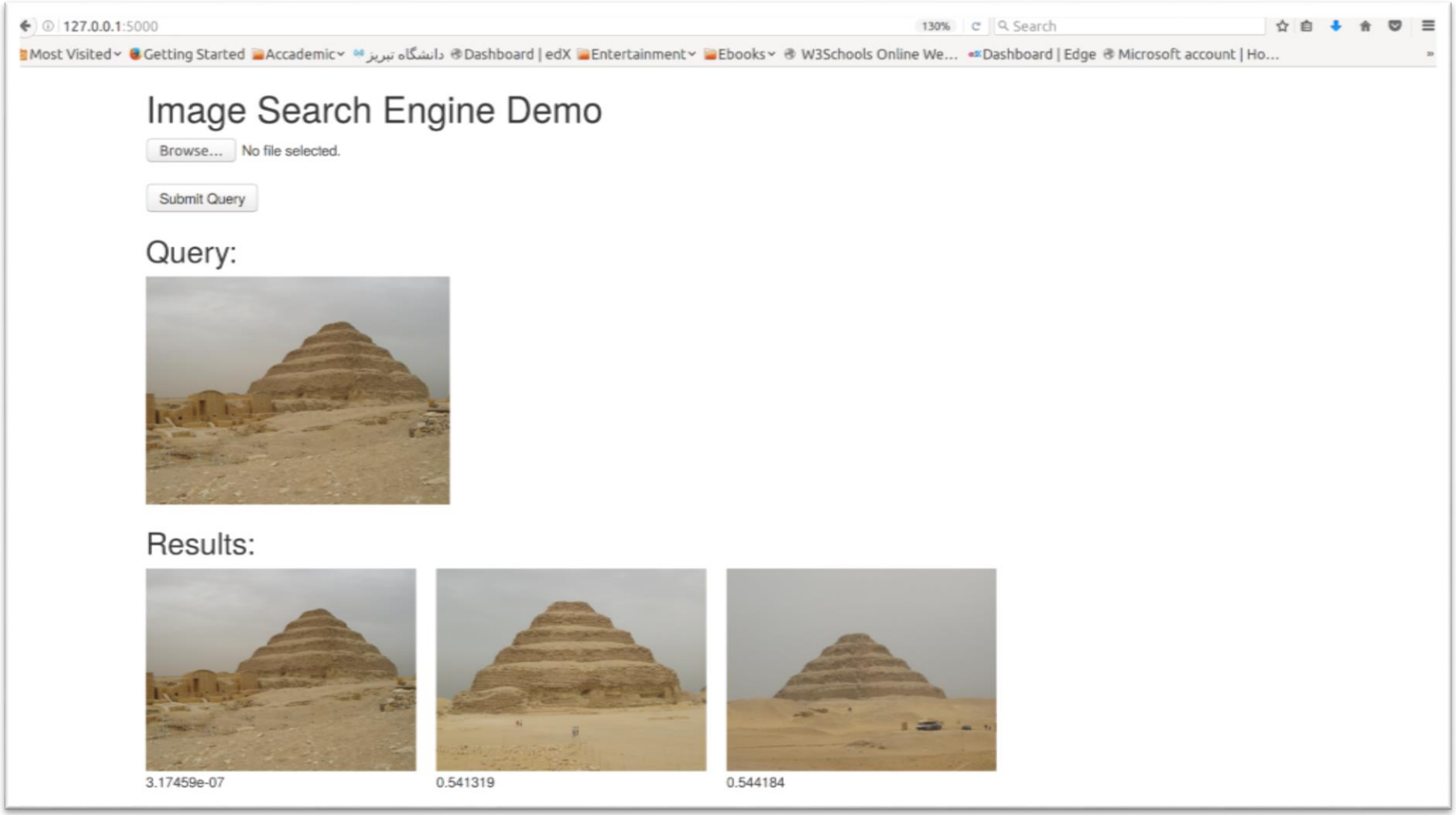
ترجمه توسط گوگل

سد و متحد روسی او، رئیس جمهور ولادیمیر پوتین، به شدت نقش دولت سوریه در حملات را رد کردند و این حادثه منجر به افتادن بین واشنگتن و مسکو بر سوریه شد

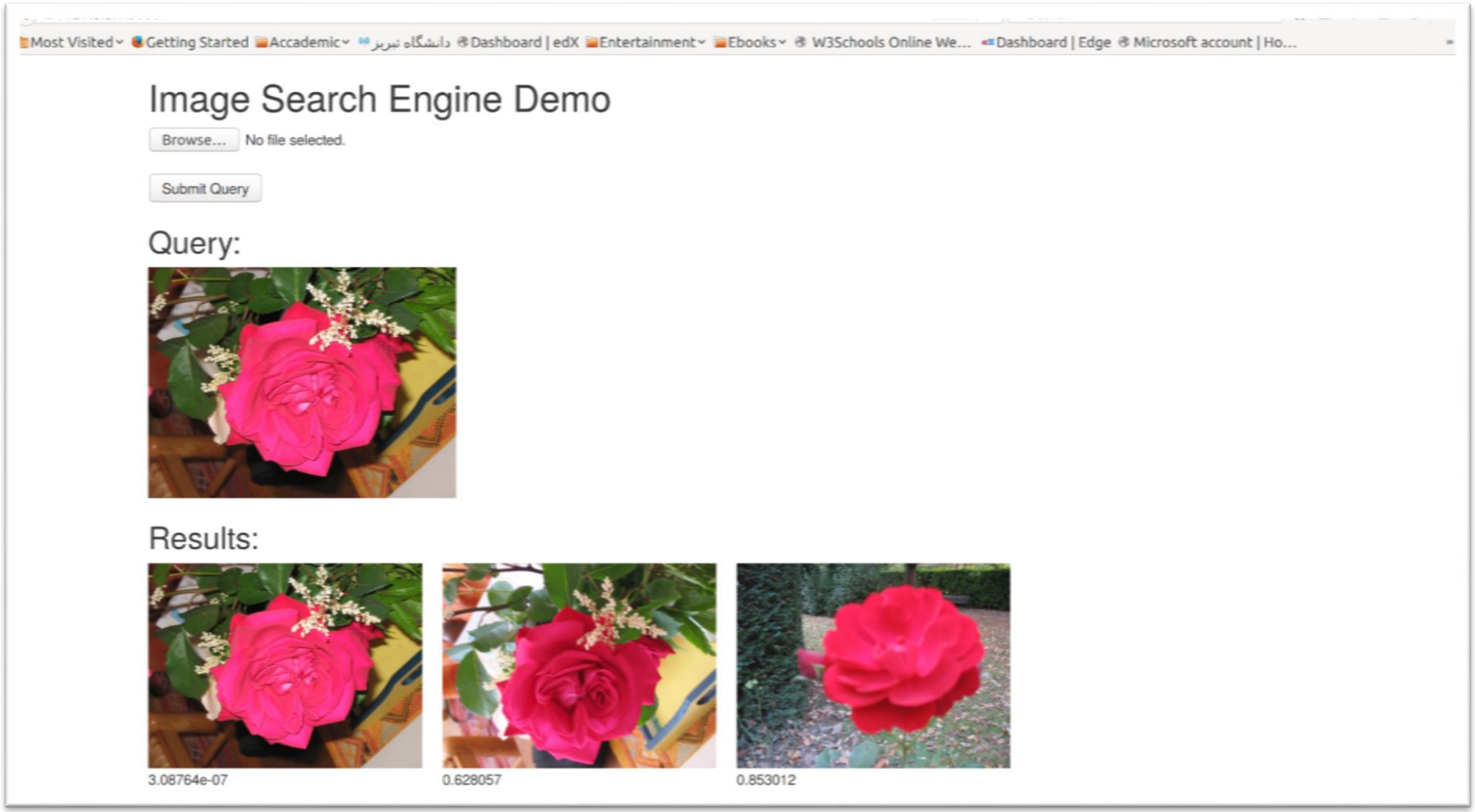
سامانه پژوهش‌یار



جویشگر تصویر (۱)



جویشگر تصویر (۲)



فهرست مطالب

مقدمه: دسته‌بندی تصویر

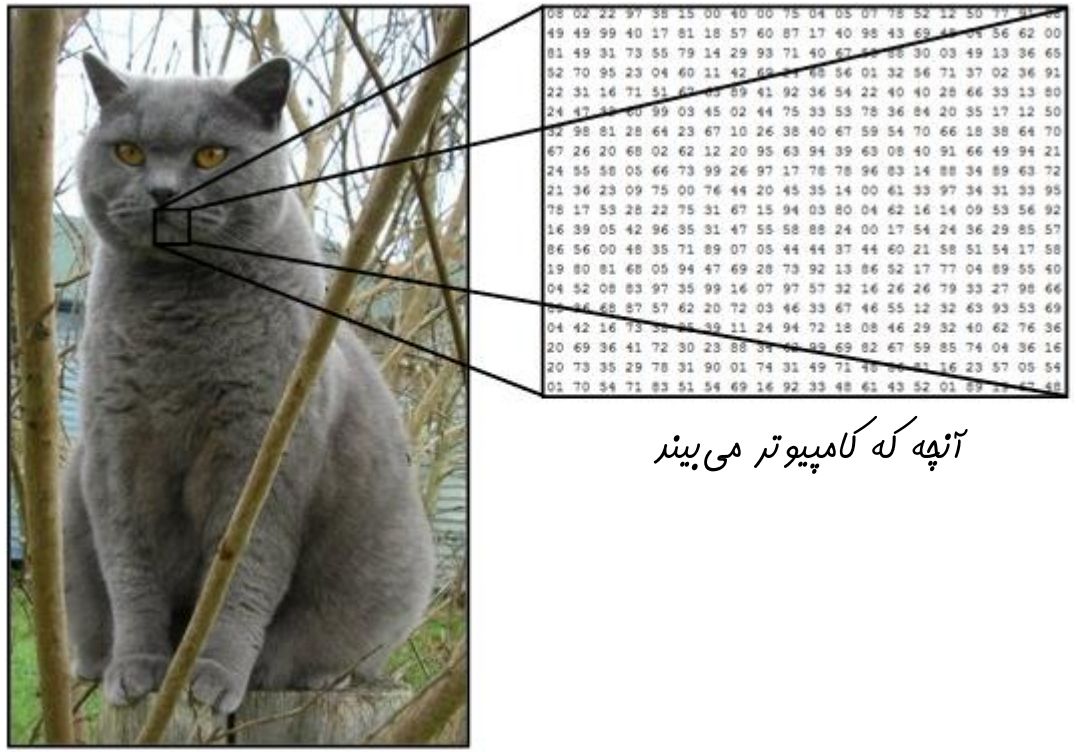
دسته‌بندی تصاویر

- یک مجموعه گسسته از برچسب‌ها به ما داده شده است:
  - تصویر ورودی به کدام دسته (برچسب) تعلق دارد؟

هواپیما
خودرو
پرنده
گربه
گوزن
سگ
قورباغه
اسب
کشتی
کامیون

مشکل: فاصله معنایی

- بازنمایی تصاویر.
  - تصاویر در کامپیوتر به صورت آرایه‌های سه بعدی از اعداد صحیح در بازه [۰، ۲۵۵] ذخیره می‌شوند.



آنچه که کامپیوتر می‌بیند

- مثال.

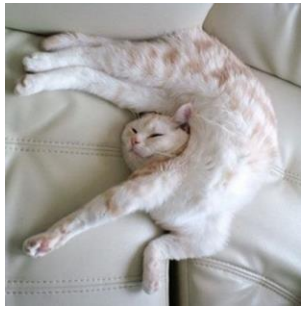
$$۳۰۰ \times ۱۰۰ \times ۳$$

RGB

چالش‌های دسته‌بندی تصاویر

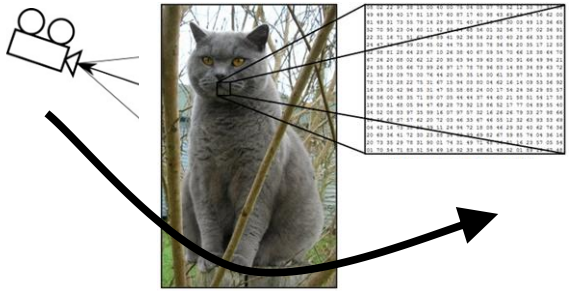
انسداد

تغییر شکل



نورپردازی

زاویه‌ی دوربین



تنوع درون کلاسی

ترکیب با پس‌زمینه

یک دسته‌بند تصویر

- هیچ روش واضحی برای کد نویسی الگوریتم تشخیص گربه یا دسته‌های دیگر وجود ندارد!
  - برخلاف مرتب‌سازی لیستی از اعداد!

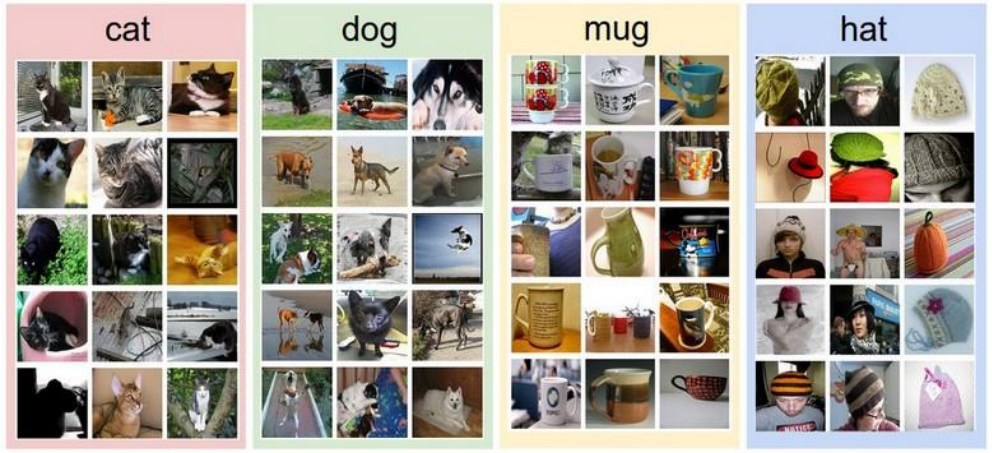
```
def predict(image):
    # ????
    return class_label
```

هواپیما
خودرو
پرنده
گربه
گوزن
سگ
قورباغه
اسب
کشتی
کامیون

رویکرد مبتنی بر داده

- رویکرد مبتنی بر داده.
  - یک مجموعه از تصاویر و برچسب مربوط به هر کدام را جمع‌آوری کن.
  - با استفاده از یادگیری ماشین، یک دسته‌بند برای دسته‌بندی تصاویر آموزش بده.
  - عملکرد دسته‌بند را بر روی یک مجموعه از تصاویر آزمایشی ارزیابی کن.

مجموعه آموزشی




```
def train(train_images, train_labels):
    # Build a model for images --> labels...
    return model

def predict(model, test_images):
    # predict test labels using the model...
    return test_labels
```

اولین دسته‌بند

- دسته‌بند نزدیک‌ترین همسایه.

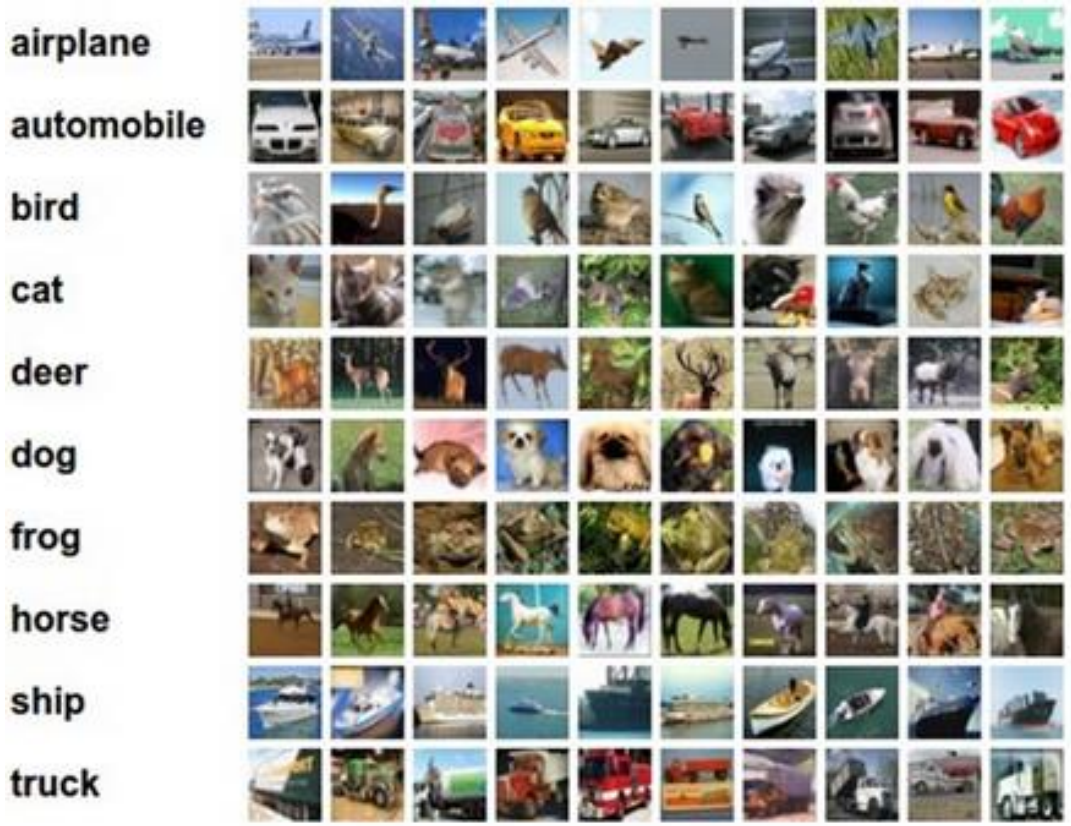
```
def train(train_images, train_labels):
    # Build a model for images --> labels...
    return model

def predict(model, test_images):
    # predict test labels using the model...
    return test_labels
```

همه تصاویر مجموعه آموزشی و برچسب‌های مربوط به هر کدام را به خاطر بسپار

برچسب تصویر آزمایشی ورودی را با توجه به برچسب شبیه‌ترین تصویر در مجموعه آموزشی پیش‌بینی کن

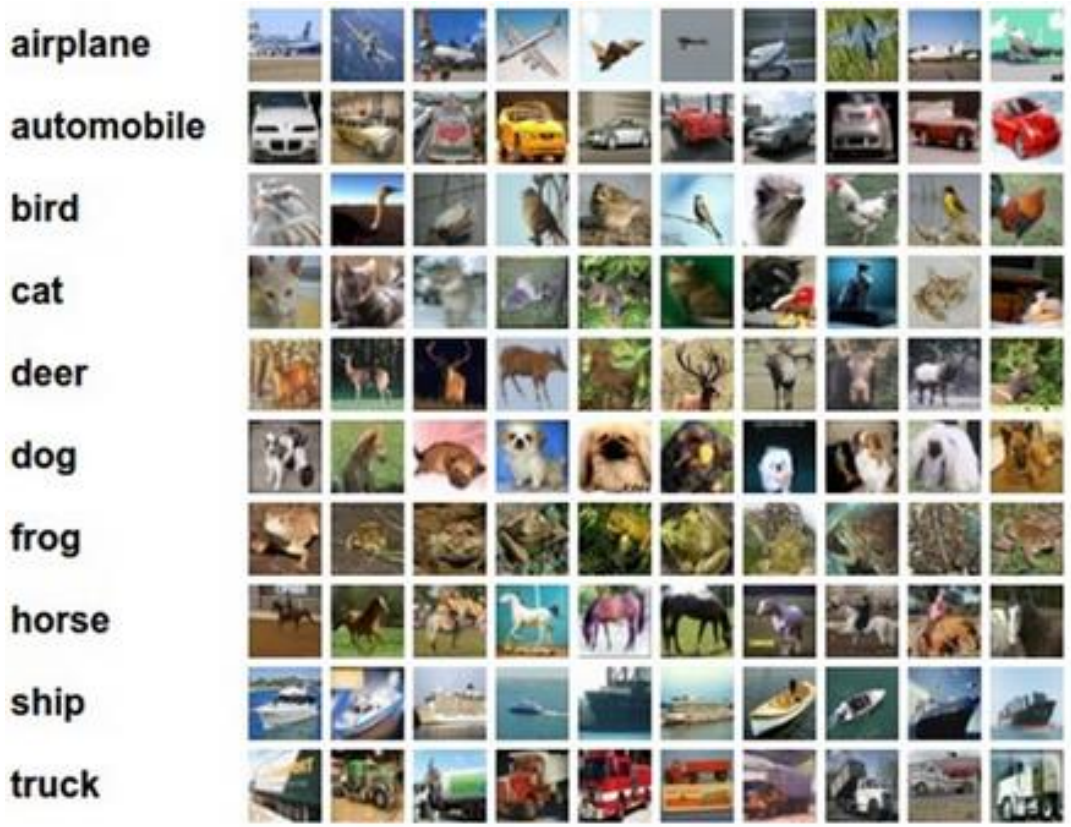
مجموعه داده CIFAR-10

- مجموعه داده CIFAR-10.
  - ۱۰ دسته، ۵۰۰۰۰ تصویر آموزشی، ۱۰۰۰۰ تصویر آزمایشی
  - اندازه هر تصویر: ۳۲ × ۳۲ پیکسل



مجموعه داده CIFAR-10

مجموعه آموزشی



به ازای هر تصویر آزمایشی (ستون اول)، تعدادی از نزدیک‌ترین همسایه‌های آن در سطر مقابل آن تصویر نشان داده شده‌اند.

مقایسه شباهت دو تصویر

- معیار فاصله. معیار فاصله L1

$$d_1(I_1, I_2) = \sum_p |I_1^p - I_2^p|$$

تصویر آزمایشی					تصویر آموزشی					اختلاف مقدار پیکسل‌ها				
56	32	10	18	−	10	20	24	17	=	46	12	14	1	
90	23	128	133		8	10	89	100		82	13	39	33	
24	26	178	200		12	16	178	170		12	10	0	30	
2	0	255	220		4	32	233	112		2	32	22	108	

جمع → 456

دسته‌بند نزدیک‌ترین همسایه: آموزش


```
import numpy as np

Class NearestNeighbor:

    def __init__(self):
        pass

    def train(self, X, y):
        """ X is N × D where each row is an example. Y is 1D of size N """
        # The nearest neighbor classifier simply remembers all the training data
        self.Xtr = X
        self.ytr = y

    ...
```

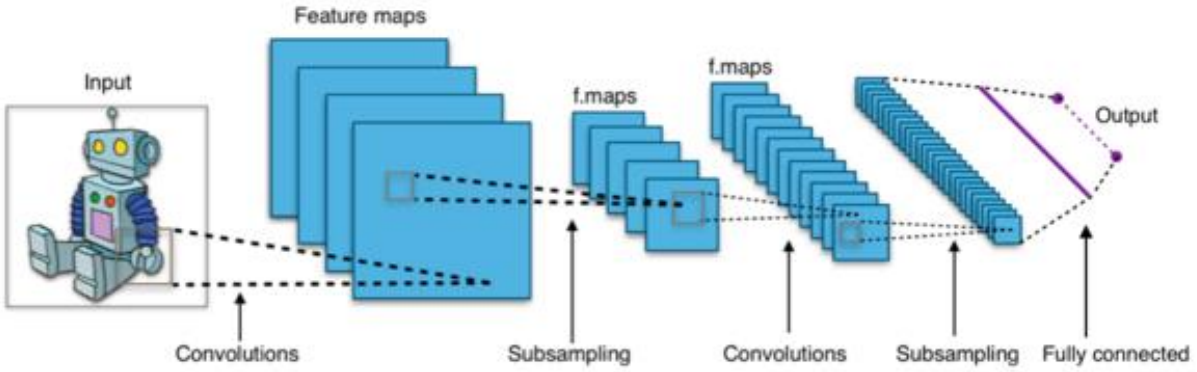
دسته‌بند نزدیک‌ترین همسایه: پیش‌بینی


```
    def predict(self, X):
        """ X is N × D where each row is an example we wish to predict label for """
        num_test = X.shape[0]
        # make sure that output type matches the input type
        Ypred = np.zeros(num_test, dtype=self.ytr.dtype)

        # loop over all test rows
        for i in xrange(num_test):
            # find the nearest training image to the i'th test image using L1 distance
            distances = np.sum(np.abs(self.Xtr - X[i, :]), axis=1)
            min_index = np.argmin(distances)
            Ypred[i] = self.ytr[min_index]

        return Ypred
```

دسته‌بند نزدیک‌ترین همسایه

- پرسش. در دسته‌بندی به روش نزدیک‌ترین همسایه، تأثیر افزایش اندازه مجموعه آموزشی بر سرعت دسته‌بندی (پیش‌بینی) چگونه است؟
- پاسخ. خطی!

- توجه. در عمل سرعت اجرای یک دسته‌بند در زمان پیش‌بینی بسیار مهم‌تر از سرعت اجرای آن در زمان یادگیری است.
  - در مورد دسته‌بند نزدیک‌ترین همسایه، دقیقاً برعکس است!



- یادگیری عمیق.
  - یادگیری (آموزش) بسیار زمان‌بر
  - پیش‌بینی (آزمایش) بسیار سریع

معیار فاصله

- معیار فاصله.
  - در دسته‌بندی به روش نزدیک‌ترین همسایه، انتخاب معیار فاصله یک ابر پارامتر است.

- انتخاب‌های متداول.

فاصله‌ی اقلیدسی (L_2)

$$d_2(I_1, I_2) = \sqrt{\sum_p \left(I_1^p - I_2^p\right)^2}$$

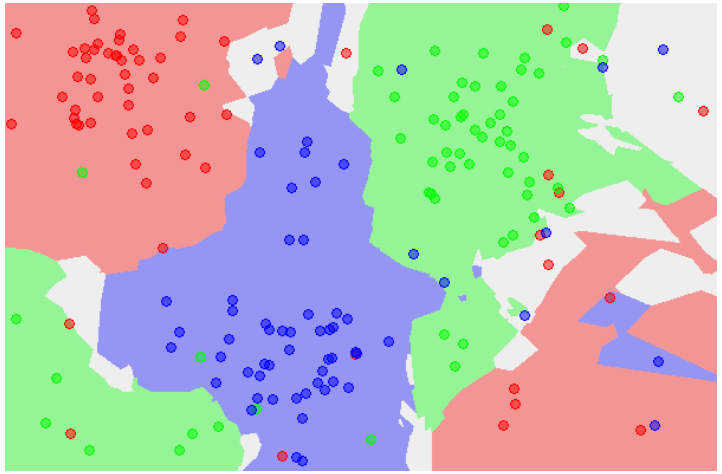
فاصله‌ی مانهاتانی (L_1)

$$d_1(I_1, I_2) = \sum_p \left|I_1^p - I_2^p\right|$$

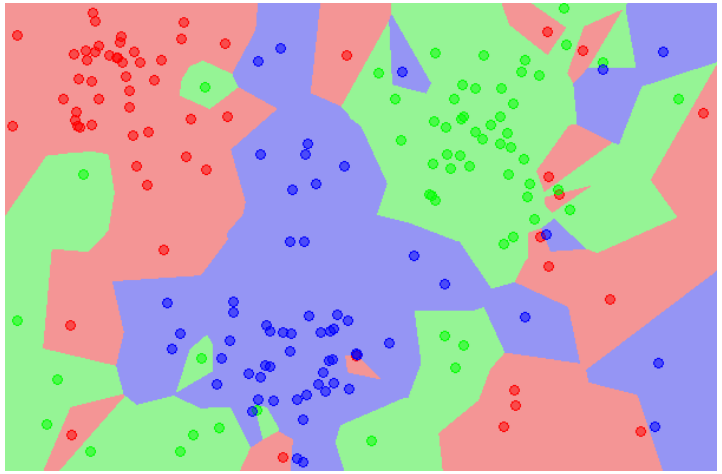
روش k-نزدیک‌ترین همسایه

- روش k - نزدیک‌ترین همسایه.
  - ابتدا k همسایه نزدیک‌تر را پیدا کن.
  - سپس در میان آنها رأی‌گیری کن.

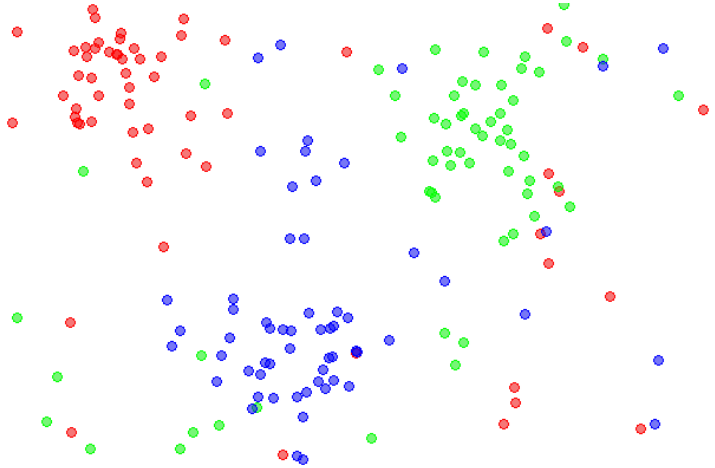
دسته‌بند ۵-نزدیک‌ترین همسایه



دسته‌بند نزدیک‌ترین همسایه

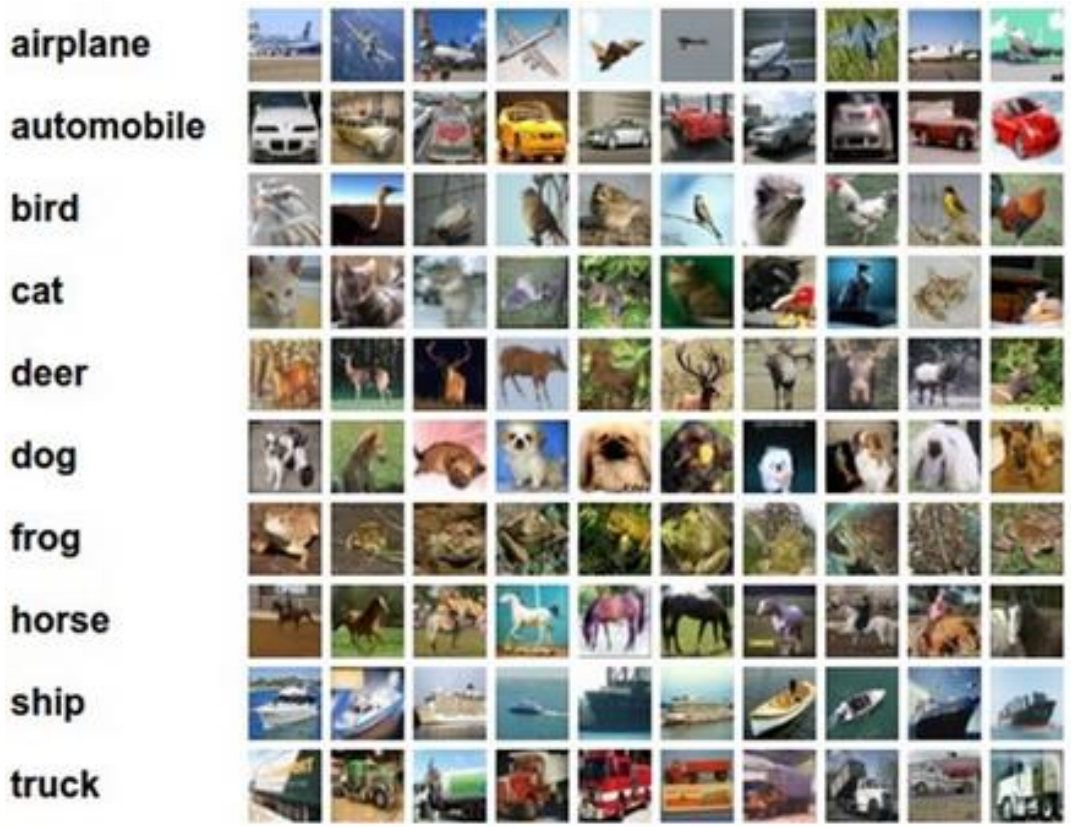


داده‌ها



روش k-نزدیک‌ترین همسایه

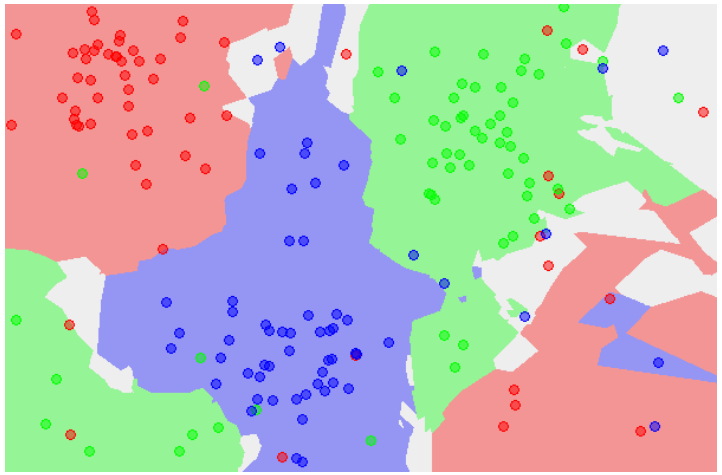
مجموعه داده **CIFAR-10**



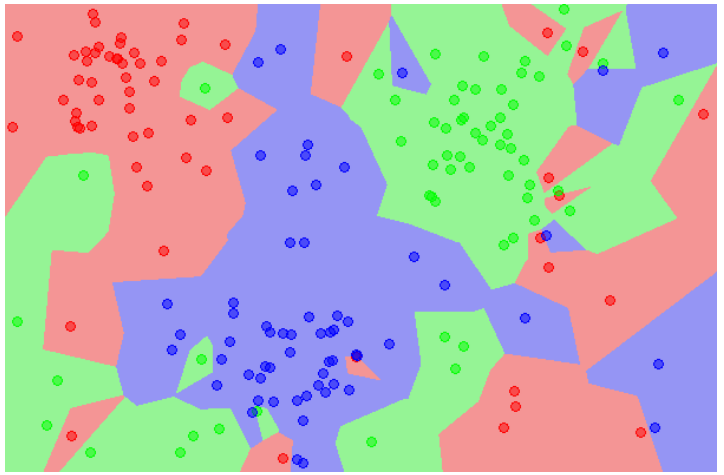
به ازای هر تصویر آزمایشی (ستون اول)، تعدادی از نزدیک‌ترین همسایه‌های آن در سطر مقابل آن تصویر نشان داده شده‌اند.

روش k-نزدیک‌ترین همسایه

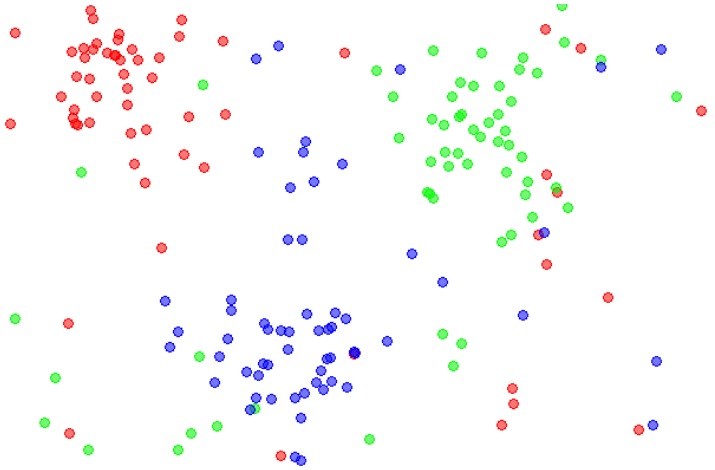
دسته‌بند ۵-نزدیک‌ترین همسایه



دسته‌بند نزدیک‌ترین همسایه



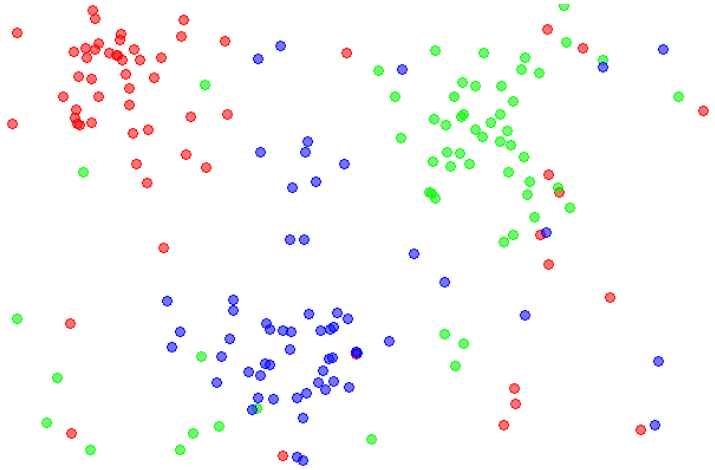
داده‌ها



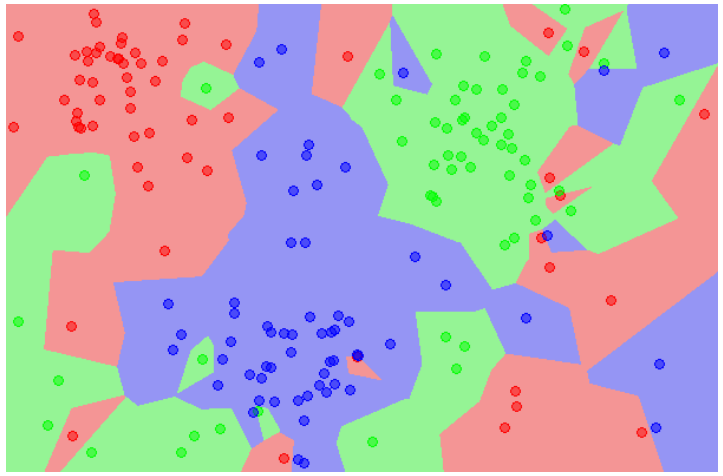
- پرسش. با استفاده از معیار فاصله اقلیدسی، دقت روش نزدیک‌ترین همسایه بر روی مجموعه آموزشی چه میزان است؟

روش k-نزدیک‌ترین همسایه

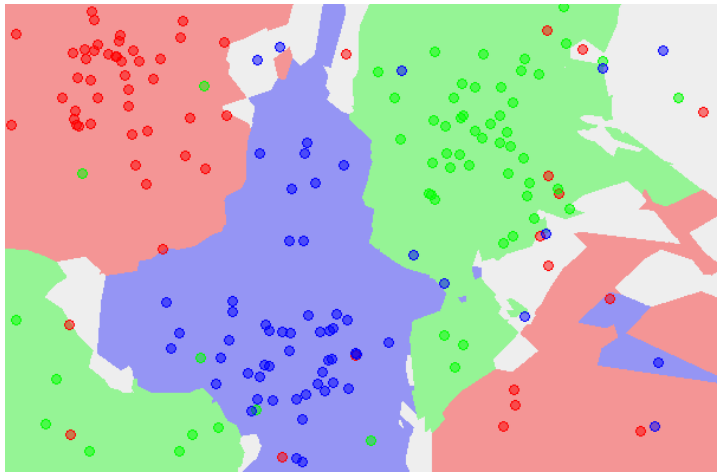
داده‌ها



دسته‌بند نزدیک‌ترین همسایه



دسته‌بند ۵-نزدیک‌ترین همسایه



- پرسش. با استفاده از معیار فاصله اقلیدسی، دقت روش k-نزدیک‌ترین همسایه بر روی مجموعه آموزشی چه میزان است؟

ابر پارامترها

- انتخاب بهترین مقدار ممکن برای ابر پارامترها.
  - بهترین معیار فاصله کدام است؟
  - بهترین مقدار برای k چیست؟

- انتخاب بهترین مقدار ممکن برای ابر پارامترها کاملاً وابسته به مسئله است.
  - برای هر ابر پارامتر، ابتدا باید تمامی (!) مقادیر ممکن را آزمایش نماییم،
  - و سپس بهترین مقدار را انتخاب کنیم.

ابر پارامترها

- راه‌حل اول. [یک ایده‌ی بسیار بد]
  - انتخاب مقداری که منجر به بالاترین دقت دسته‌بندی بر روی مجموعه آزمایشی می‌شود.

داده‌های آزمایشی	داده‌های آموزشی

- توجه. [بسیار مهم]
  - از مجموعه آزمایشی در انتهای مراحل و تنها برای تخمین قابلیت تعمیم دسته‌بند استفاده کنید.

ابر پارامترها

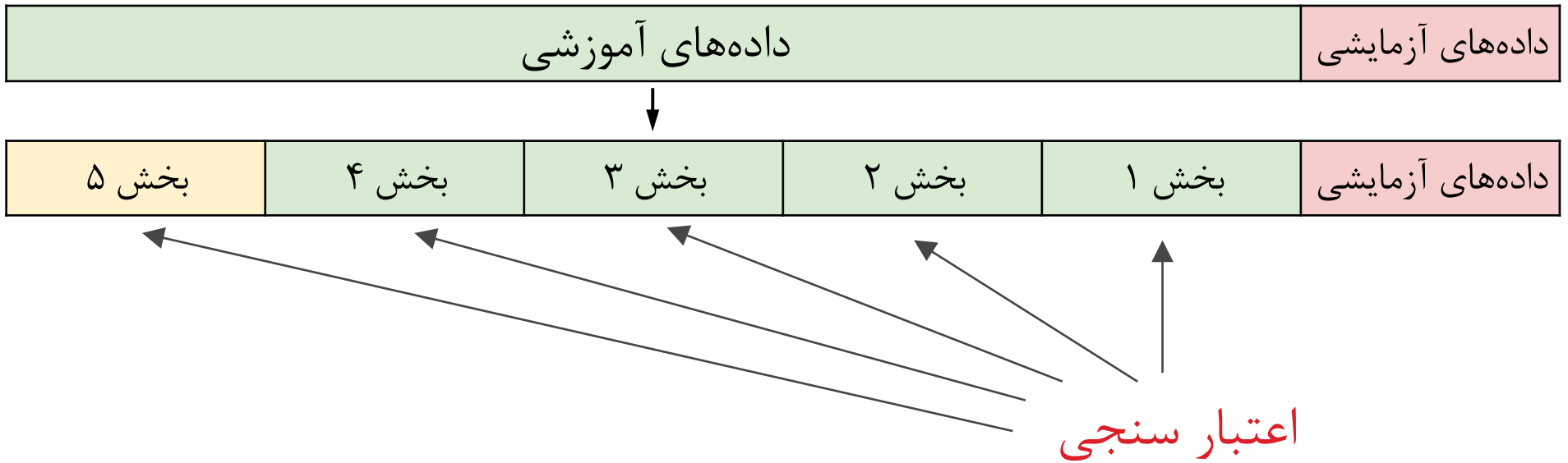
- راه‌حل دوم. [اعتبارسنجی چند بخشی]

داده‌های اعتبار سنجی
[برای تعیین مقدار ابرپارامترها]

داده‌های آزمایشی	داده‌های آموزشی				
داده‌های آزمایشی	بخش ۱	بخش ۲	بخش ۳	بخش ۴	بخش ۵
داده‌های آزمایشی	بخش ۱	بخش ۲	بخش ۳	بخش ۴	بخش ۵
داده‌های آزمایشی	بخش ۱	بخش ۲	بخش ۳	بخش ۴	بخش ۵
داده‌های آزمایشی	بخش ۱	بخش ۲	بخش ۳	بخش ۴	بخش ۵
داده‌های آزمایشی	بخش ۱	بخش ۲	بخش ۳	بخش ۴	بخش ۵

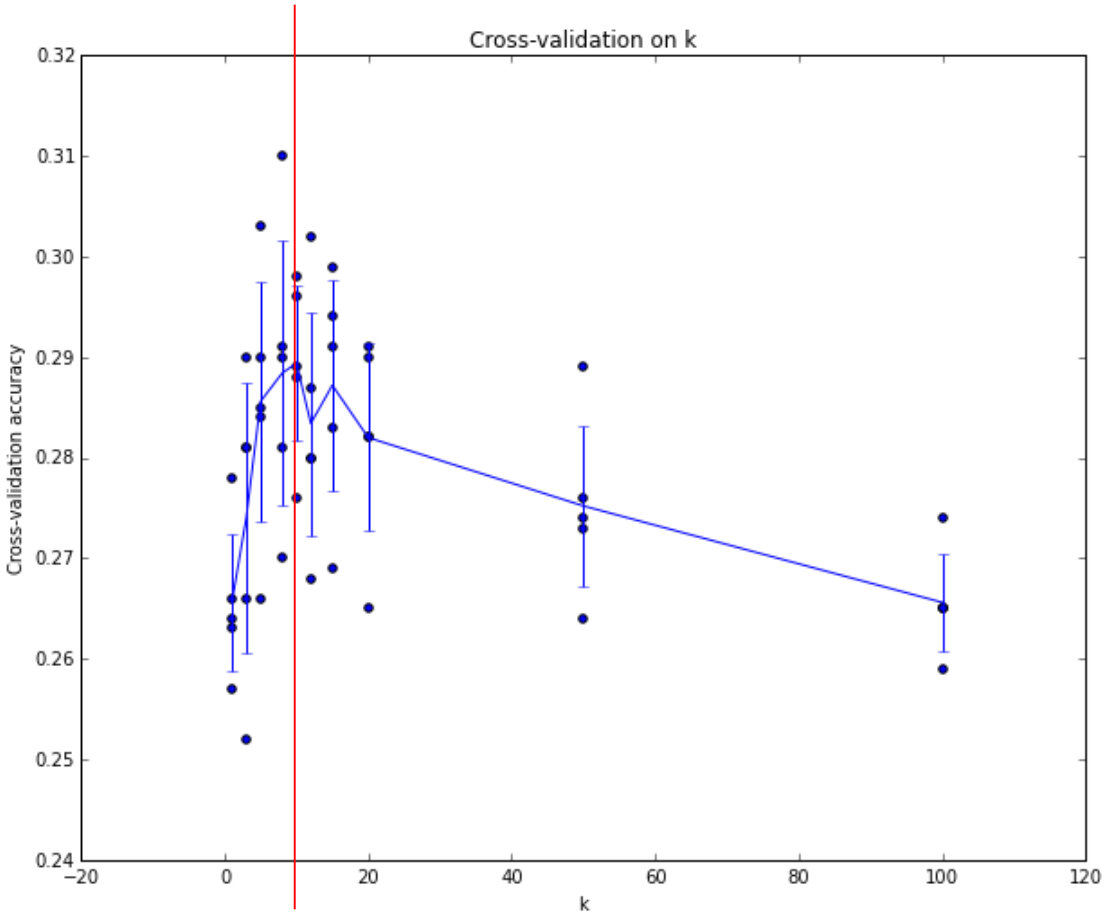
ابر پارامترها

- راه‌حل دوم. [اعتبارسنجی چند بخشی]



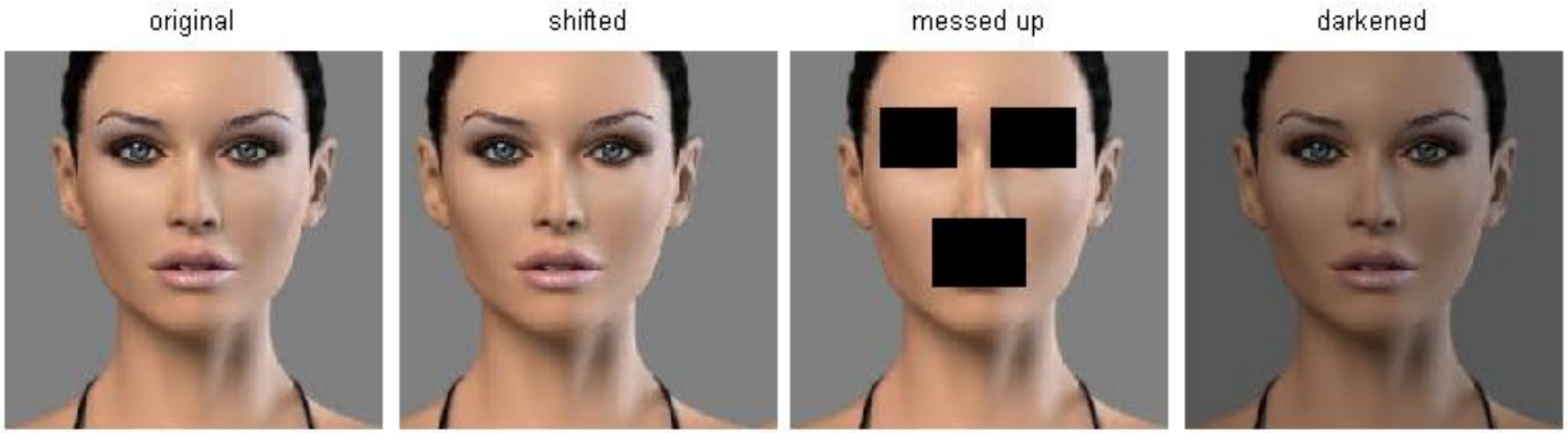
هر بار یک بخش را به عنوان داده‌های اعتبارسنجی انتخاب کن و سپس از نتایج به دست آمده میانگین بگیر

اعتبارسنجی چند بخشی



روش k-نزدیک‌ترین همسایه و دسته‌بندی تصاویر

- توجه. **هرگز** از روش k-نزدیک‌ترین همسایه برای دسته‌بندی تصاویر استفاده نکنید.
  - سرعت بسیار کم در زمان پیش‌بینی (آزمایش)!
  - نامناسب بودن استفاده از معیارهای فاصله در سطح کل تصویر!



هر سه تصویر تغییر یافته، فاصله اقلیدسی یکسانی با تصویر اصلی (تصویر سمت چپ) دارند.

جمع‌بندی

- دسته‌بندی تصاویر.
  - مجموعه آموزشی و مجموعه آزمایشی
- دسته‌بندی به روش k-نزدیک‌ترین همسایه.
  - معیار فاصله مانهاتانی و اقلیدسی برای محاسبه شباهت بین دو تصویر
- تعیین مقدار مناسب برای ابرپارامترها.
  - مجموعه اعتبارسنجی
  - اعتبارسنجی چندبخشی
- تخمین قابلیت تخمین دسته‌بند با استفاده از مجموعه آزمایشی.

دسته‌بندی خطی

فهرست مطالب

رویکرد پارامتری: دسته‌بندی خطی

- دسته‌بندی خطی.

[۳۲ × ۳۲ × ۳]

برداری از اعداد در بازه‌ی [۰ ، ۱]

تصویر

پارامترها

f(x, W)

۱۰ عدد

هر عدد بیانگر امتیاز مربوط به یک کلاس

رویکرد پارامتری: دسته‌بندی خطی

- دسته‌بندی خطی.

پارامترها، یا «وزن‌ها»

$$f(x, W) = W \quad x \quad (+b)$$

۱۰ × ۱ ۱۰ × ۳۰۷۲ ۳۰۷۲ × ۱ ۱۰ × ۱

۱۰ عدد

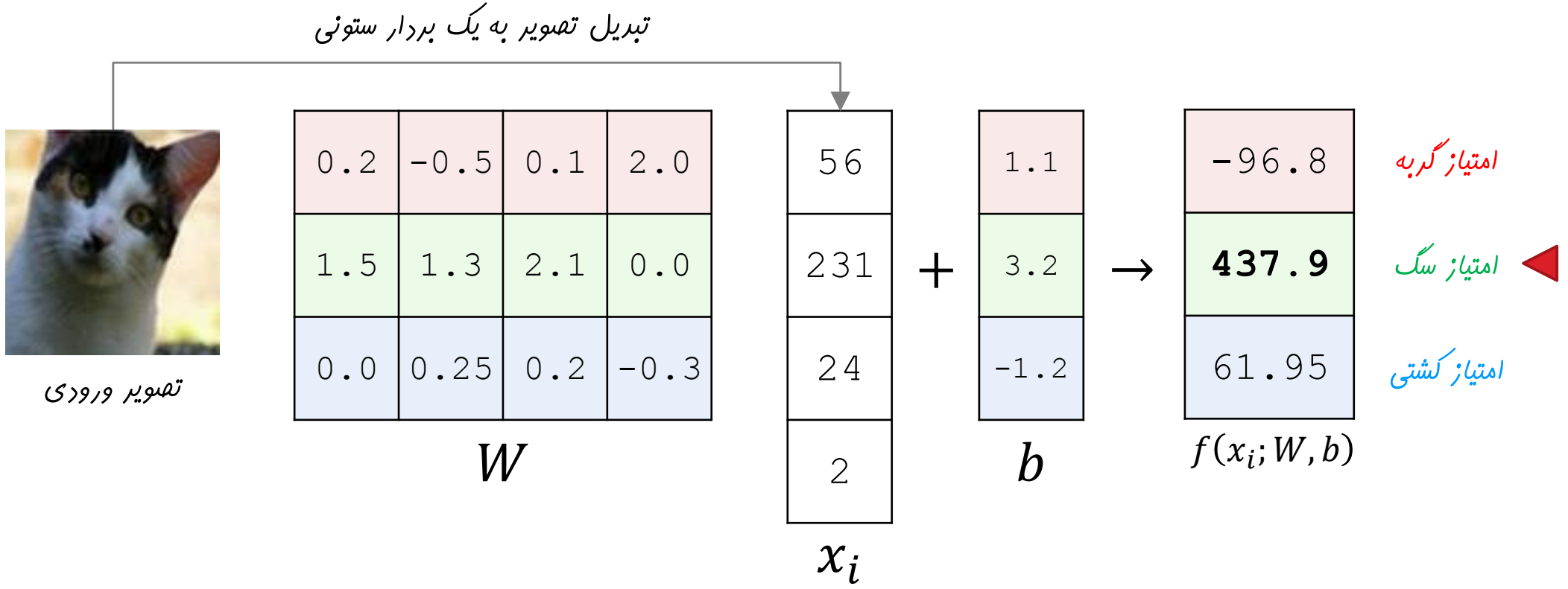
هر عدد بیانگر امتیاز مربوط به یک کلاس

[۳۲ × ۳۲ × ۳]

برداری از اعداد در بازه‌ی [۰ , ۱]

دسته‌بندی خطی: مثال

- چهار ویژگی و سه کلاس [سگ، گربه، کِشتی]



تفسیر دسته‌بندی خطی

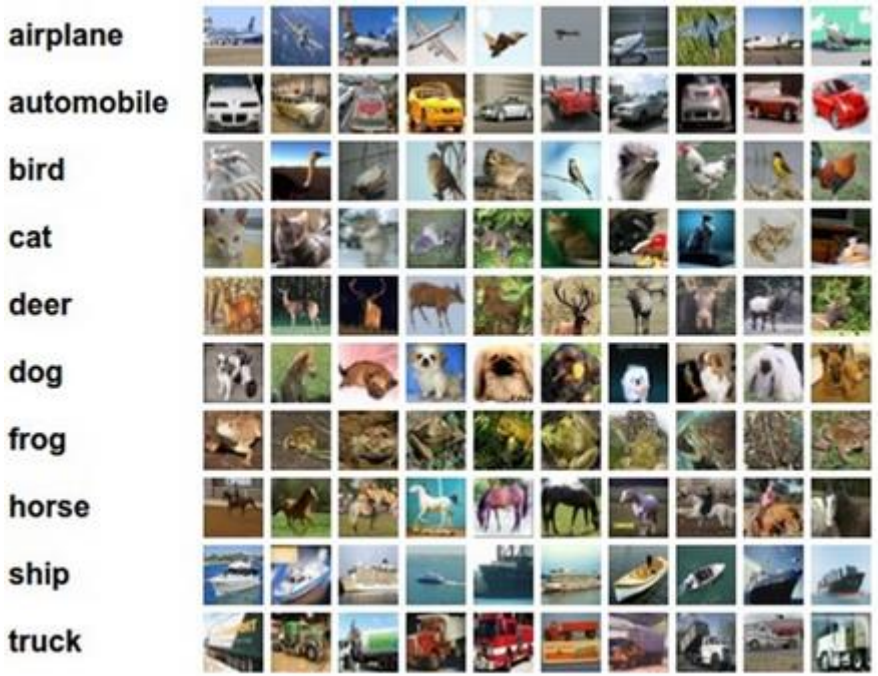
- یک دسته‌بند خطی چه عملی انجام می‌دهد؟

$$f(x_i; W, b) = Wx_i + b$$

تفسیر دسته‌بندی خطی

- یک دسته‌بند خطی چه عملی انجام می‌دهد؟

$$f(x_i; W, b) = Wx_i + b$$



نمونه‌ای از وزن‌های آموزش داده شده با استفاده از یک دسته‌بند خطی بر روی مجموعه‌ی CIFAR-10

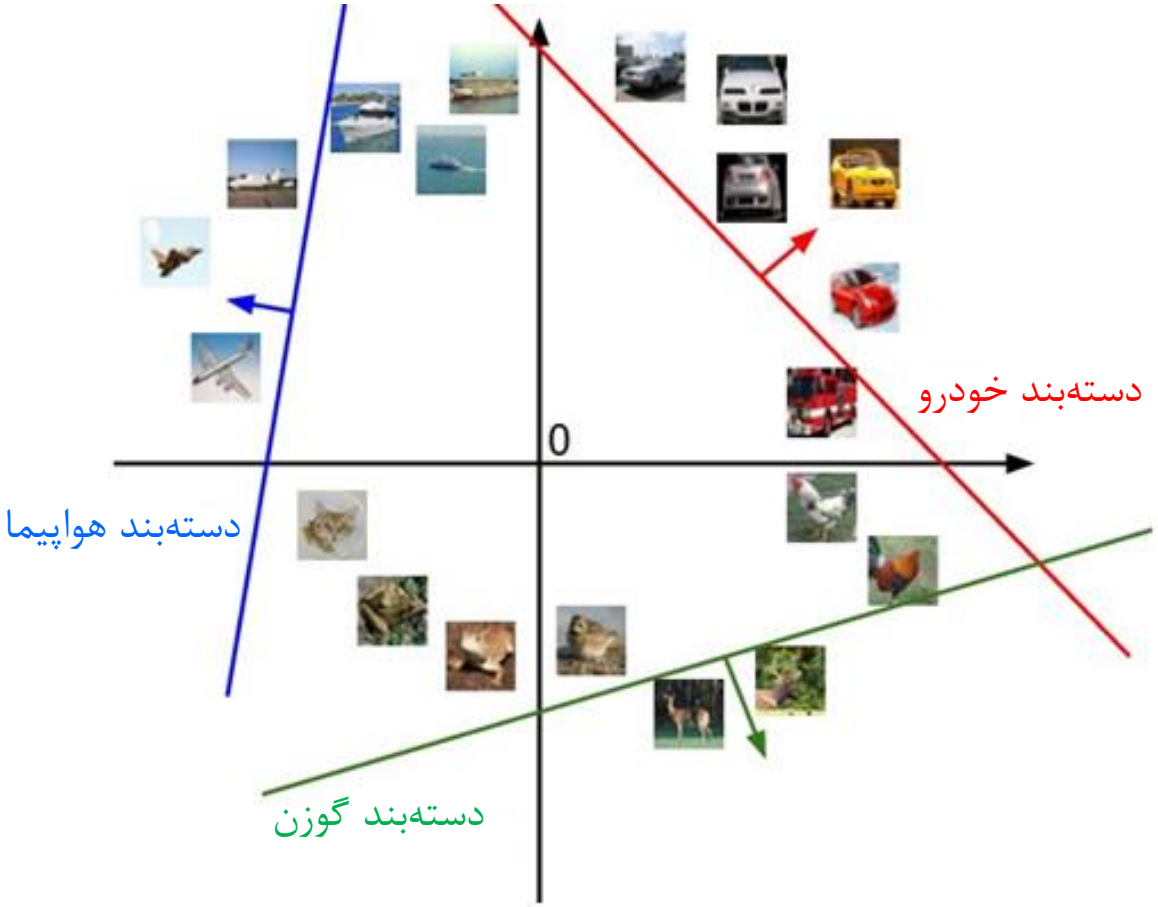
تفسیر دسته‌بندی خطی

- یک دسته‌بند خطی چه عملی انجام می‌دهد؟

$$f(x_i; W, b) = Wx_i + b$$

$[۳۲ \times ۳۲ \times ۳]$

برداری از اعداد در بازه‌ی $[۰ , ۱]$



دسته‌بندی خطی: تابع امتیاز

- امتیاز کلاس‌های مختلف برای سه تصویر ورودی با استفاده از وزن‌های تصادفی.



-3.45	-0.51	3.42	هواپیما
-8.87	**6.04**	4.64	خودرو
0.09	5.31	2.65	پرنده
2.90	-4.22	5.10	گربه
4.48	-4.19	2.64	گوزن
8.02	3.58	5.55	سگ
3.78	4.49	**-4.34**	قورباغه
1.06	-4.37	-1.50	اسب
-0.36	-2.09	-4.79	کشتی
-0.72	-2.93	6.14	کامیون

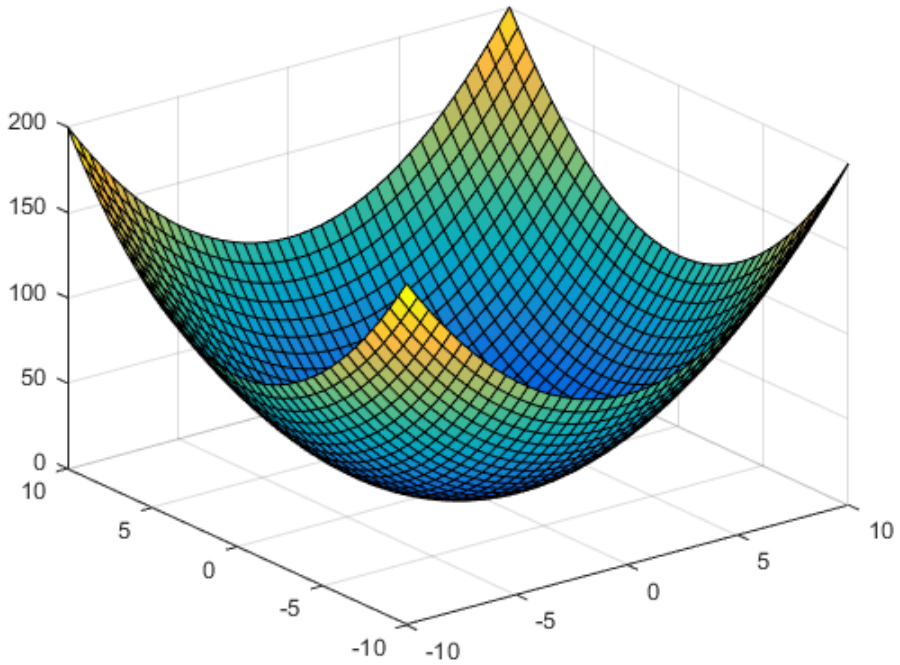
$$f(x_i; W, b) = Wx_i + b$$

$[۳۲ \times ۳۲ \times ۳]$

برداری از اعداد در بازه‌ی $[۰ , ۱]$

در ادامه ...

- تابع هزینه [تابع زیان].
  - بیانگر میزان بد بودن ماتریس وزن به صورت یک مقدار عددی!
- بهینه‌سازی.
  - یافتن یک ماتریس وزن به گونه‌ای که تابع هزینه را کمینه نماید.
- دسته‌بندی غیر خطی.



تابع هزینه

تابع هزینه و بهینه‌سازی

- تعریف تابع هزینه.
  - به گونه‌ای که بیانگر میزان ناخشنودی ما از امتیازهای محاسبه شده بر روی داده‌های مجموعه آموزشی باشد.
- بهینه‌سازی.
  - یافتن مقادیر پارامترها به گونه‌ای که تابع هزینه کمینه گردد.



هواپیما	3.42	-0.51	-3.45
خودرو	4.64	**6.04**	-8.87
پرنده	2.65	5.31	0.09
گربه	5.10	-4.22	**2.90**
گوزن	2.64	-4.19	4.48
سگ	5.55	3.58	**8.02**
قورباغه	**-4.34**	4.49	3.78
اسب	-1.50	-4.37	1.06
کشتی	-4.79	-2.09	-0.36
کامیون	**6.14**	-2.93	-0.72

تابع هزینه و بهینه‌سازی

- مجموعه آموزشی.
  - شامل ۳ نمونه‌ی آموزشی از ۳ کلاس مختلف.

- تابع امتیاز.
  - محاسبه امتیاز هر کلاس برای هر یک از داده‌ها.

$$s = f(W, x) = Wx$$



گربه	**3.2**	1.3	2.2
خودرو	5.1	**4.9**	2.5
قورباغه	-1.7	2.0	**-3.1**

تابع هزینه SVM چند کلاسی

- محاسبه هزینه برای داده (x_i, y_i)
  - x_i: تصویر ورودی
  - y_i: برچسب تصویر ورودی (یک عدد صحیح)

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$



گربه	**3.2**	1.3	2.2
خودرو	5.1	**4.9**	2.5
قورباغه	-1.7	2.0	**-3.1**

تابع هزینه SVM چند کلاسی

- محاسبه هزینه برای داده (x_i, y_i)
  - x_i: تصویر ورودی
  - y_i: برچسب تصویر ورودی (یک عدد صحیح)



گربه	**3.2**	1.3	2.2
خودرو	5.1	**4.9**	2.5
قورباغه	-1.7	2.0	**-3.1**
	2.9		

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

```
= max(0,  5.1 - 3.2 + 1) +
  max(0, -1.7 - 3.2 + 1)
= max(0, 2.9) + max(0, -3.9)
= 2.9 + 0
= 2.9
```

تابع هزینه SVM چند کلاسی

- محاسبه هزینه برای داده (x_i, y_i)
  - x_i: تصویر ورودی
  - y_i: برچسب تصویر ورودی (یک عدد صحیح)

گربه	**3.2**	1.3	2.2
خودرو	5.1	**4.9**	2.5
قورباغه	-1.7	2.0	**-3.1**
	2.9	**0.0**	

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

```
= max(0, 1.3 - 4.9 + 1) +
  max(0, 2.0 - 4.9 + 1)
= max(0, -2.6) + max(0, -1.9)
= 0 + 0
= 0
```

تابع هزینه SVM چند کلاسی

- محاسبه هزینه برای داده (x_i, y_i)
  - x_i: تصویر ورودی
  - y_i: برچسب تصویر ورودی (یک عدد صحیح)

گربه	**3.2**	1.3	2.2
خودرو	5.1	**4.9**	2.5
قورباغه	-1.7	2.0	**-3.1**
	2.9	**0.0**	**12.9**

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

```
= max(0, 2.2 - (-3.1) + 1) +
  max(0, 2.5 - (-3.1) + 1)
= max(0, 6.3) + max(0, 6.6)
= 6.3 + 6.6
= 12.9
```

تابع هزینه SVM چند کلاسی

- محاسبه هزینه برای داده (x_i, y_i)
  - x_i: تصویر ورودی
  - y_i: برچسب تصویر ورودی (یک عدد صحیح)

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

- هزینه کل.
  - میانگین هزینه بر روی تمام داده‌های آموزشی.

$$L = \frac{1}{N} \sum_i L_i$$



گربه	**3.2**	1.3	2.2
خودرو	5.1	**4.9**	2.5
قورباغه	-1.7	2.0	**-3.1**
	2.9	**0.0**	**12.9**

L = (2.9 + 0 + 12.9) / 3 = 5.27

تابع هزینه SVM چند کلاسی

- محاسبه هزینه برای داده (x_i, y_i)
  - x_i: تصویر ورودی
  - y_i: برچسب تصویر ورودی (یک عدد صحیح)

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

اگر در این رابطه، مجموع روی تمام کلاس‌ها محاسبه شود، تابع هزینه چگونه تغییر می‌کند؟

گربه	**3.2**	1.3	2.2
خودرو	5.1	**4.9**	2.5
قورباغه	-1.7	2.0	**-3.1**
	2.9	0.0	12.9

تابع هزینه SVM چند کلاسی

- محاسبه هزینه برای داده (x_i,y_i)
  - x_i: تصویر ورودی
  - y_i: برچسب تصویر ورودی (یک عدد صحیح)

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

اگر در این رابطه، به جای مجموع از میانگین استفاده شود، تابع هزینه چگونه تغییر می‌کند؟



گربه	**3.2**	1.3	2.2
خودرو	5.1	**4.9**	2.5
قورباغه	-1.7	2.0	**-3.1**
	2.9	0.0	12.9

تابع هزینه SVM چند کلاسی

- محاسبه هزینه برای داده‌ی (x_i,y_i)
  - x_i: تصویر ورودی
  - y_i: برچسب تصویر ورودی (یک عدد صحیح)

گربه	**3.2**	1.3	2.2
خودرو	5.1	**4.9**	2.5
قورباغه	-1.7	2.0	**-3.1**
	2.9	0.0	12.9

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

اگر به جای این رابطه، از رابطه زیر استفاده شود، تابع هزینه چگونه تغییر می‌کند؟

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)^2$$

تابع هزینه SVM چند کلاسی

- محاسبه هزینه برای داده (x_i, y_i)
  - x_i: تصویر ورودی
  - y_i: برچسب تصویر ورودی (یک عدد صحیح)

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

کمترین و بیشترین مقدار ممکن برای تابع هزینه چقدر است؟



گربه	**3.2**	1.3	2.2
خودرو	5.1	**4.9**	2.5
قورباغه	-1.7	2.0	**-3.1**
	2.9	0.0	12.9

تابع هزینه SVM چند کلاسی

- محاسبه هزینه برای داده (x_i,y_i)
  - x_i: تصویر ورودی
  - y_i: برچسب تصویر ورودی (یک عدد صحیح)

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

به طور معمول در لحظه شروع، مقادیر W با مقادیر تصادفی کوچک مقداردهی می‌شوند و بنابراین تمام امتیازها تقریبا صفر هستند. در این حالت مقدار تابع هزینه کدام است؟



گربه	**3.2**	1.3	2.2
خودرو	5.1	**4.9**	2.5
قورباغه	-1.7	2.0	**-3.1**
	2.9	0.0	12.9

تابع هزینه SVM چند کلاسی

- پیاده‌سازی در پایتون.

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

```
def L_i_vectorized(x, y, W):
    scores = W.dot(x)
    margins = np.maximum(0, scores - scores[y] + 1)
    margins[y] = 0
    loss_i = np.sum(margins)
    return loss_i
```

وجود مشکل در تابع هزینه

- فرض کنید یک W پیدا کرده‌ایم به طوری که $L = 0$. آیا W یکتا است؟

$$f(x, W) = Wx$$

$$L = \frac{1}{N} \sum_{i=1}^{N} \sum_{j \neq y_i} \max(0, f(x_i; W)_j - f(x_i; W)_{y_i} + 1)$$

وجود مشکل در تابع هزینه

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

گربه	**3.2**	1.3	2.2
خودرو	5.1	**4.9**	2.5
قورباغه	-1.7	2.0	**-3.1**
	2.9	**0.0**	

قبلاً:

```
= max(0, 1.3 - 4.9 + 1) +
  max(0, 2.0 - 4.9 + 1)
= max(0, -2.6) + max(0, -1.9)
= 0 + 0
= 0
```

اگر مقدار W را ۲ برابر کنیم:

```
= max(0, 2.6 - 9.8 + 1) +
  max(0, 4.0 - 9.8 + 1)
= max(0, -6.2) + max(0, -4.8)
= 0 + 0
= 0
```

راه‌حل: تنظیم وزن‌ها

- تنظیم وزن‌ها.

ضریب تنظیم

$$L = \frac{1}{N}\sum_{i=1}^{N}\sum_{j\neq y_i}\max(0, f(x_i; W)_j - f(x_i; W)_{y_i} + 1) + \boxed{\lambda R(W)}$$

- چند روش متداول.

$$R(W) = \sum_k \sum_l W_{k,l}^2$$ L2-Regularization

$$R(W) = \sum_k \sum_l |W_{k,l}|$$ L1-Regularization

$$R(W) = \sum_k \sum_l \beta W_{k,l}^2 + |W_{k,l}|$$ Elastic net (L1 + L2)

دسته‌بند سافت‌مکس (رگرسیون لجستیک چند کلاسی)

- امتیازها. لگاریتم احتمال کلاس‌ها به صورت نرمال نشده!

$$P(Y = k|X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}} \qquad s = f(x_i; W)$$

تابع سافت‌مکس

گربه	**3.2**
خودرو	5.1
قورباغه	-1.7

- هدف.
  - بیشینه‌سازی لگاریتم درست‌نمایی (یا کمینه‌سازی منفی لگاریتم درست‌نمایی)!

$$L_i = -\log P(Y = y_i|X = x_i) = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right)$$

دسته‌بند سافت‌مکس (رگرسیون لجستیک چند کلاسی)

- امتیازها. لگاریتم احتمال کلاس‌ها به صورت نرمال نشده!

$$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right)$$

$$L_i = -\log(0.13) = 0.89$$

	لگاریتم احتمال‌ها به صورت غیرنرمال		احتمال‌ها به صورت غیرنرمال		احتمال‌ها
گربه	3.2	توان رسانی	24.5	نرمال‌سازی	0.13
خودرو	5.1		164.0		0.87
قورباغه	-1.7		0.2		0.0

کمترین و بیشترین مقدار ممکن برای تابع هزینه چقدر است؟

دسته‌بند سافت‌مکس (رگرسیون لجستیک چند کلاسی)

- امتیازها. لگاریتم احتمال کلاس‌ها به صورت نرمال نشده!

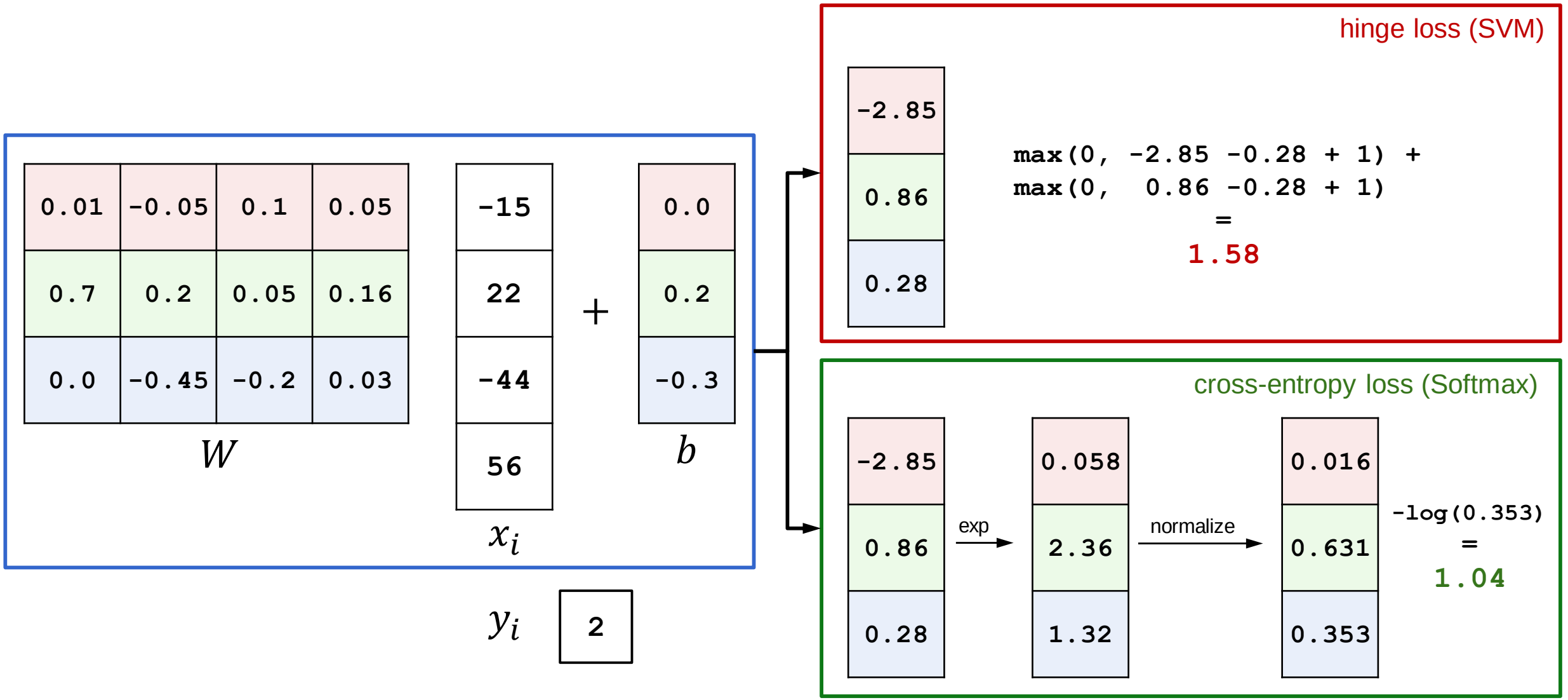
$$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right)$$

$$L_i = -\log(0.13) = 0.89$$

	لگاریتم احتمال‌ها به صورت غیرنرمال	توان رسانی	احتمال‌ها به صورت غیرنرمال	نرمال‌سازی	احتمال‌ها
گربه	3.2		24.5		0.13
خودرو	5.1		164.0		0.87
قورباغه	-1.7		0.2		0.0

به طور معمول در لحظه شروع، مقادیر W با اعداد تصادفی کوچک مقداردهی می‌شوند و بنابراین تمام امتیازها تقریبا صفر هستند. در این حالت مقدار تابع هزینه کدام است؟

توابع هزینه


hinge loss (SVM)
0.01 -0.05 0.1 0.05
0.7 0.2 0.05 0.16
0.0 -0.45 -0.2 0.03
W
-15
22
-44
56
x_i
+
0.0
0.2
-0.3
b
y_i
2
-2.85
0.86
0.28
max(0, -2.85 -0.28 + 1) +
max(0, 0.86 -0.28 + 1)
=
1.58
cross-entropy loss (Softmax)
-2.85
0.86
0.28
exp
0.058
2.36
1.32
normalize
0.016
0.631
0.353
-log(0.353)
=
1.04

مقایسه

- پرسش. فرض کنید یک داده را برداریم و آن را اندکی جابه‌جا کنیم (امتیازهای آن را کمی تغییر دهیم). در مورد هر یک از دو تابع هزینه داده شده، مقدار هزینه چگونه تغییر می‌کند؟

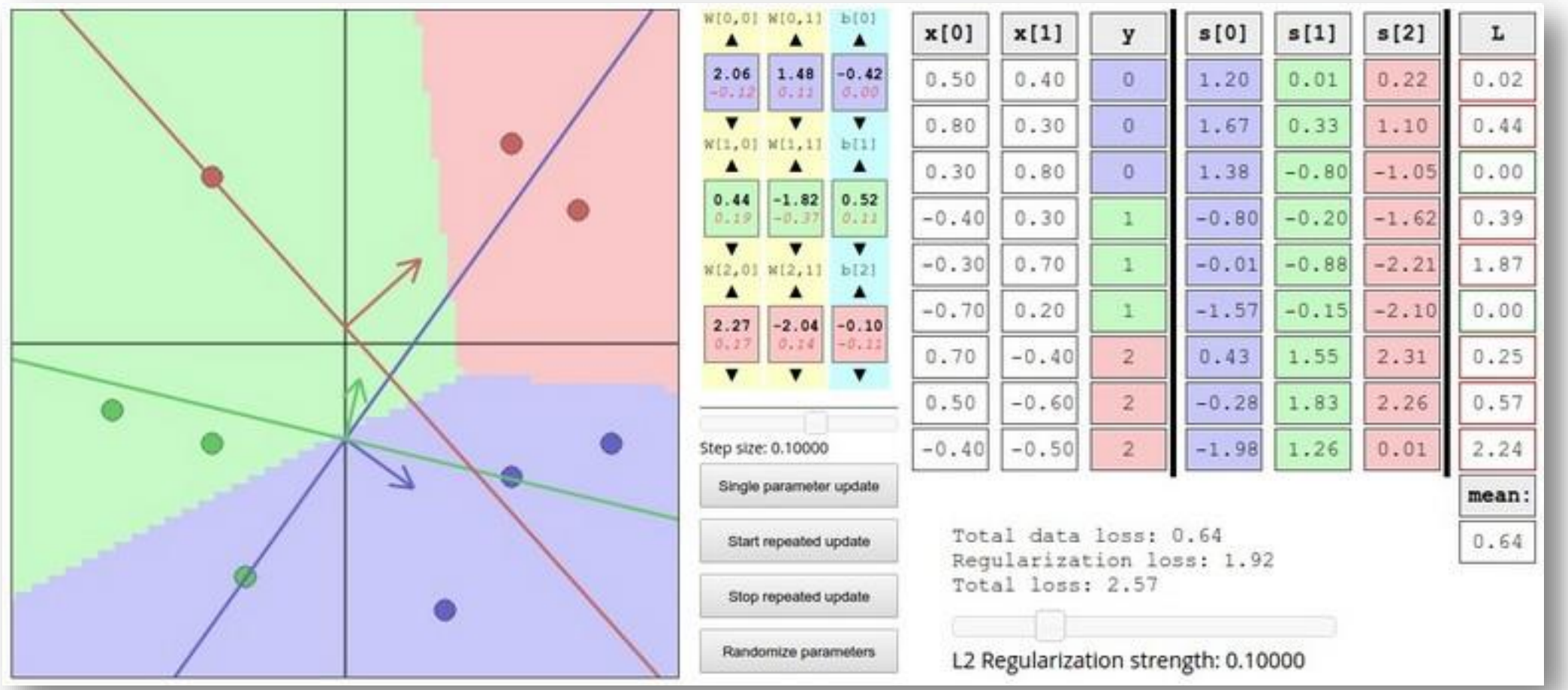
$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

$$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right)$$

- امتیازها. $[y_i = 0]$

```
[10,   -2,    3]
[10,    9,    9]
[10, -100, -100]
```

اجرای نمایشی



http://vision.stanford.edu/teaching/cs231n/linear-classify-demo/

بهینه‌سازی

یادآوری

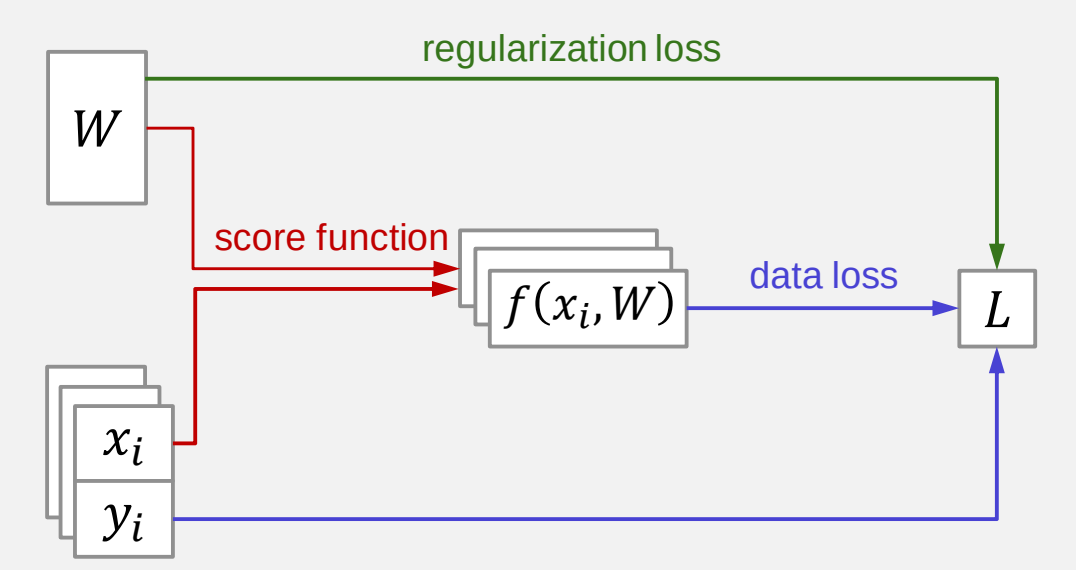
- یک مجموعه از داده‌های آموزشی به صورت (x,y)
- یک تابع امتیاز: $s = f(x; W) = Wx$
- یک تابع هزینه:

هدف. کمینه‌سازی تابع هزینه به منظور یافتن مقادیر W

$$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right)$$ hinge loss (SVM)

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$ cross-entropy loss (Softmax)

$$L = \left(\frac{1}{N}\sum_{i=1}^{N} L_i\right) + \lambda R(W)$$ Full loss



استراتژی اول: جستجوی تصادفی (یک ایده‌ی بسیار بد)


```
best_loss = float("inf")
for i in xrange(1000):
    W = np.random.randn(10, 3073) * 0.0001
    loss = L(X_train, y_train, W)
    if loss < best_loss:
        best_loss = loss
        best_W = W
    print "In attempt %d the loss was %f, best %f" % (i, loss, best_loss)
```

```
In attempt 0 the loss was 9.401632, best 9.401632
In attempt 1 the loss was 8.959668, best 8.959668
In attempt 2 the loss was 9.044034, best 8.959668
In attempt 3 the loss was 9.278948, best 8.959668
In attempt 4 the loss was 8.857370, best 8.857370
In attempt 5 the loss was 8.943151, best 8.857370
```

یک نمونه از اجرای الگوریتم جستجوی تصادفی

آزمایش جستجوی تصادفی


```
# assume X_test is (3073, 10000), y_test is (10000, 1)
scores = np.dot(best_W, X_test)
# find the index with max score in each column (the predicted class)
y_pred = np.argmax(scores, axis=0)
# calculate accuracy (fraction of predictions that are correct)
accuracy = np.mean(y_pred == y_test)
# returns 0.1555 as accuracy
```

دقت ۱۵/۵ درصد! خیلی بد نیست!

(اما دقت بهترین راه‌حل در حدود ۹۵ درصد است)

گرادیان کاهشی

استراتژی دوم: دنبال کردن شیب

- مشتق تابع. در یک فضای ۱- بُعدی

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

- در یک فضای چند بُعدی، گرادیان تابع برداری است از مشتق‌های جزئی.

محاسبه عددی گرادیان

W	$W + h$		dW
0.34	0.34	+ 0.0001	?
-1.11	-1.11		?
0.78	0.78		?
0.12	0.12		?
0.55	0.55		?
2.81	2.81		?
-3.10	-3.10		?
-1.50	-1.50		?
0.33	0.33		?
...	...		...

Loss 1.25347

Loss 1.25322

محاسبه عددی گرادیان

W

0.34
-1.11
0.78
0.12
0.55
2.81
-3.10
-1.50
0.33
...

Loss 1.25347

$W + h$

0.34	+ 0.0001
-1.11	
0.78	
0.12	
0.55	
2.81	
-3.10	
-1.50	
0.33	
...	

Loss 1.25322

dW

-2.50
?
?
?
.
?
...

(1.25322 - 1.25347) / 0.0001

= -2.5

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

محاسبه عددی گرادیان

W
0.34
-1.11
0.78
0.12
0.55
2.81
-3.10
-1.50
0.33
...

Loss 1.25347

$W + h$
0.34
-1.11
0.78
0.12
0.55
2.81
-3.10
-1.50
0.33
...

+ 0.0001

Loss 1.25353

dW
-2.50
?
?
?
?
?
?
?
?
...

محاسبه عددی گرادیان

W
0.34
-1.11
0.78
0.12
0.55
2.81
-3.10
-1.50
0.33
...

Loss 1.25347

$W + h$
0.34
-1.11
0.78
0.12
0.55
2.81
-3.10
-1.50
0.33
...

Loss 1.25353

+ 0.0001

dW
-2.50
0.60
?
?
.
?
...

(1.25353 - 1.25347) / 0.0001

= 0.6

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

محاسبه عددی گرادیان

W	$W + h$		dW
0.34	0.34		-2.50
-1.11	-1.11		0.60
0.78	0.78	+ 0.0001	?
0.12	0.12		?
0.55	0.55		?
2.81	2.81		?
-3.10	-3.10		?
-1.50	-1.50		?
0.33	0.33		?
...	...		...

Loss 1.25347

Loss 1.25347

محاسبه عددی گرادیان

W
0.34
-1.11
0.78
0.12
0.55
2.81
-3.10
-1.50
0.33
...

Loss 1.25347

$W + h$
0.34
-1.11
0.78
0.12
0.55
2.81
-3.10
-1.50
0.33
...

Loss 1.25347

+ 0.0001

dW
-2.50
0.60
0.00
?
.
?
...

(1.25347 - 1.25347) / 0.0001

= **0.0**

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

محاسبه عددی گرادیان


```
def eval_numerical_gradient(f, x):

    fx = f(x)
    grad = np.zeros(x.shape)
    h = 0.00001

    it = np.nditer(x, flags=['multi_index'], op_flags=['readwrite'])
    while not it.finished:

        ix = it.multi_index
        old_value = x[ix]
        x[ix] += h
        fxh = f(x)  # evaluate f(x + h)
        x[ix] = old_value

        grad[ix] = (fxh - fx) / h  # compute the partial derivative
        it.iternext()  # step to next dimension

return grad
```

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

- معایب.
  - تقریبی
  - بسیار زمان‌بر

محاسبه گرادیان به صورت تحلیلی

- تابع هزینه یک تابع از پارامترهای W است.
  - محاسبه گرادیان به صورت عددی چندان هوشمندانه به نظر نمی‌رسد.

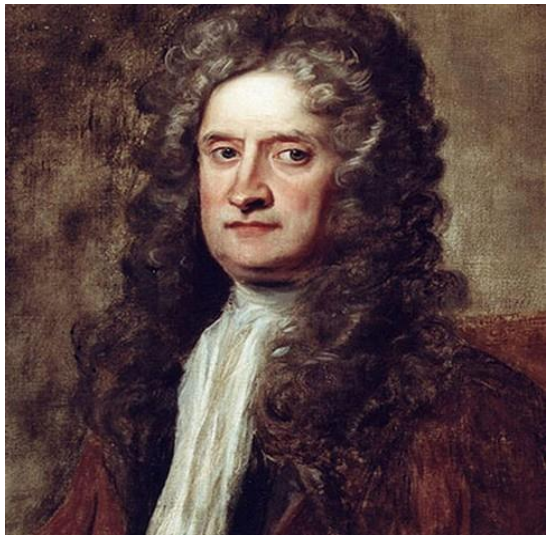
$$L = \frac{1}{N}\sum_{i=1}^{N} L_i + \lambda \sum_k W_k^2$$

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

$$s = f(x; W) = Wx$$

$$\nabla_W L = \cdots$$

ویلهلم لایب‌نیتس (۱۶۴۶ – ۱۷۱۶)



ایزاک نیوتن (۱۶۴۲ – ۱۷۲۷)

محاسبه گرادیان به صورت تحلیلی

- تابع هزینه یک تابع از پارامترهای W است.
  - محاسبه گرادیان به صورت عددی چندان هوشمندانه به نظر نمی‌رسد.

$$L = \frac{1}{N}\sum_{i=1}^{N} L_i + \lambda \sum_k W_k^2$$

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

$$s = f(x; W) = Wx$$

$$\nabla_W L = \cdots$$

حسابان

محاسبه گرادیان به صورت تحلیلی

W

0.34
-1.11
0.78
0.12
0.55
2.81
-3.10
-1.50
0.33
...

Loss 1.25347

$$\nabla_W L = \cdots$$

تابعی از داده‌ها و پارامترهای W

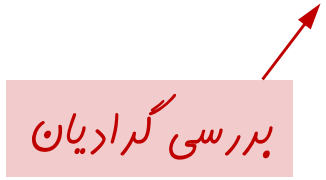
dW

-2.50
0.60
0.00
0.20
0.70
-0.50
1.10
1.30
-2.10
...

بررسی گرادیان

- به طور خلاصه.
  - گرادیان عددی: تقریبی، زمان‌بر، پیاده‌سازی آسان!
  - گرادیان تحلیلی: دقیق، سریع، امکان بروز خطا در پیاده‌سازی!

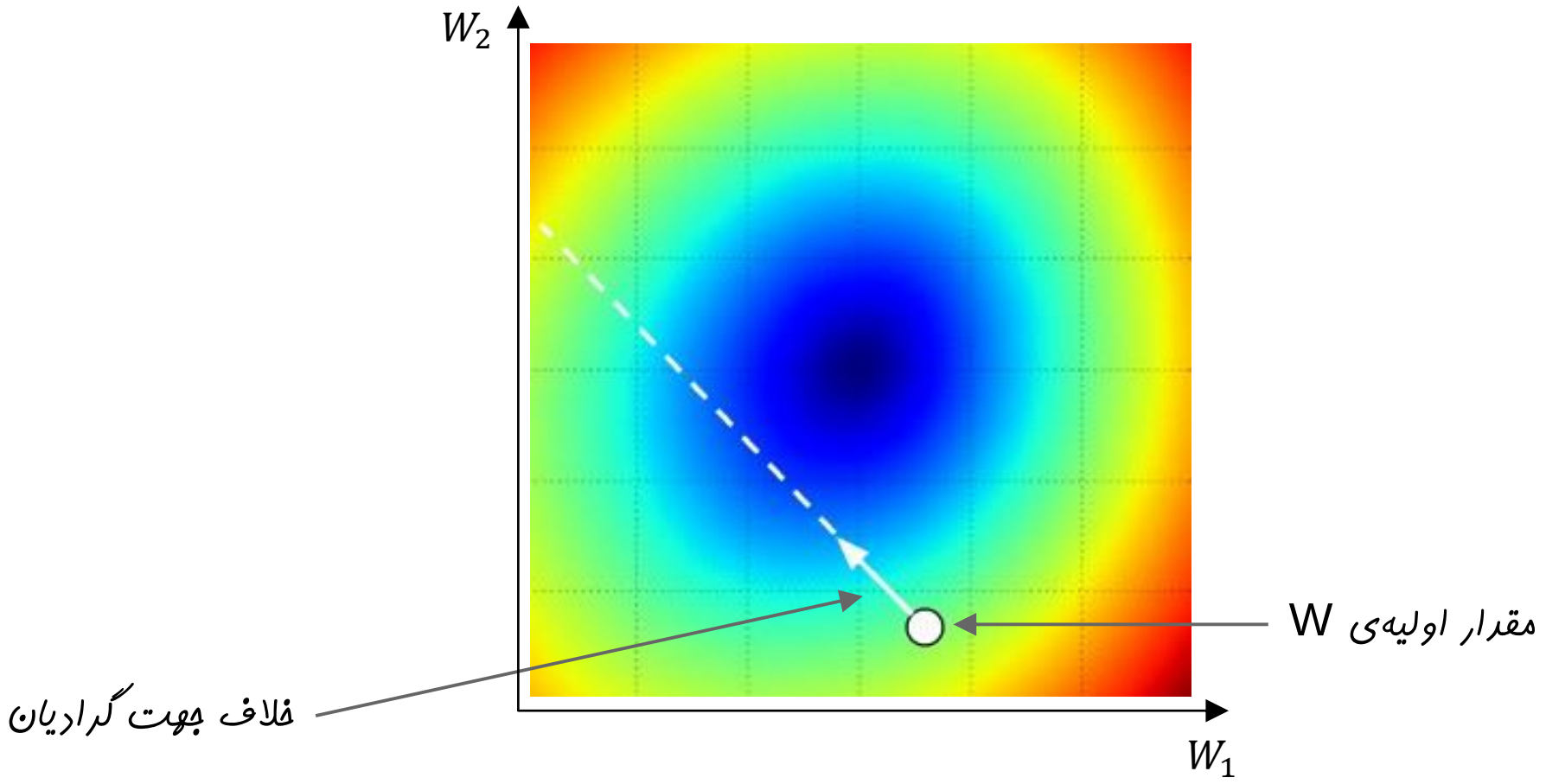
- در عمل.
  - همیشه از گرادیان تحلیلی استفاده می‌کنیم.
  - اما به منظور اطمینان از صحت پیاده‌سازی، گرادیان تحلیلی را با گرادیان عددی مقایسه می‌کنیم.



الگوریتم گرادیان کاهشی


```
# Vanilla Gradient Descent

while True:
    gradient = evaluate_gradient(loss_fun, data, weights)
    weights += -step_size * gradient  # weight update
```

الگوریتم گرادیان کاهشی



یک نسخه کاراتر از الگوریتم گرادیان کاهشی

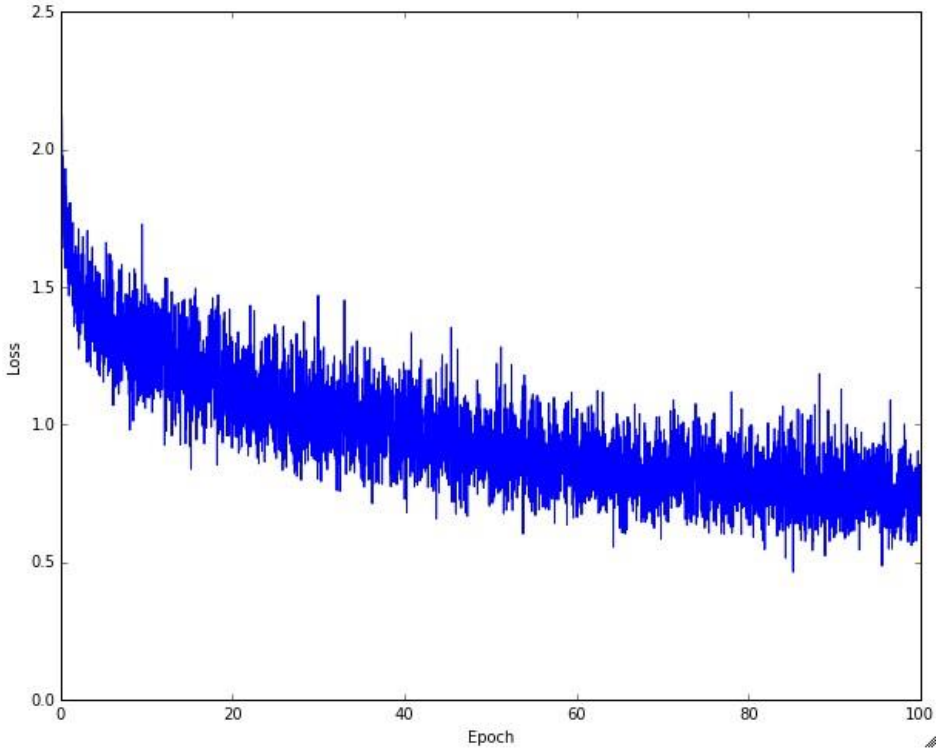
- گرادیان کاهشی دسته‌ای.
  - برای محاسبه گرادیان تابع هزینه در هر تکرار، تنها از بخش کوچکی از داده‌های آموزشی استفاده کن.

```
# Mini-batch Gradient Descent

while True:
    data_batch = sample_training_data(data, 256)  # sample 256 examples
    gradient = evaluate_gradient(loss_fun, data_batch, weights)
    weights += -step_size * gradient  # weight update
```

  - مقادیر متداول برای اندازه دسته: ۳۲، ۶۴، ۱۲۸ و ۲۵۶.

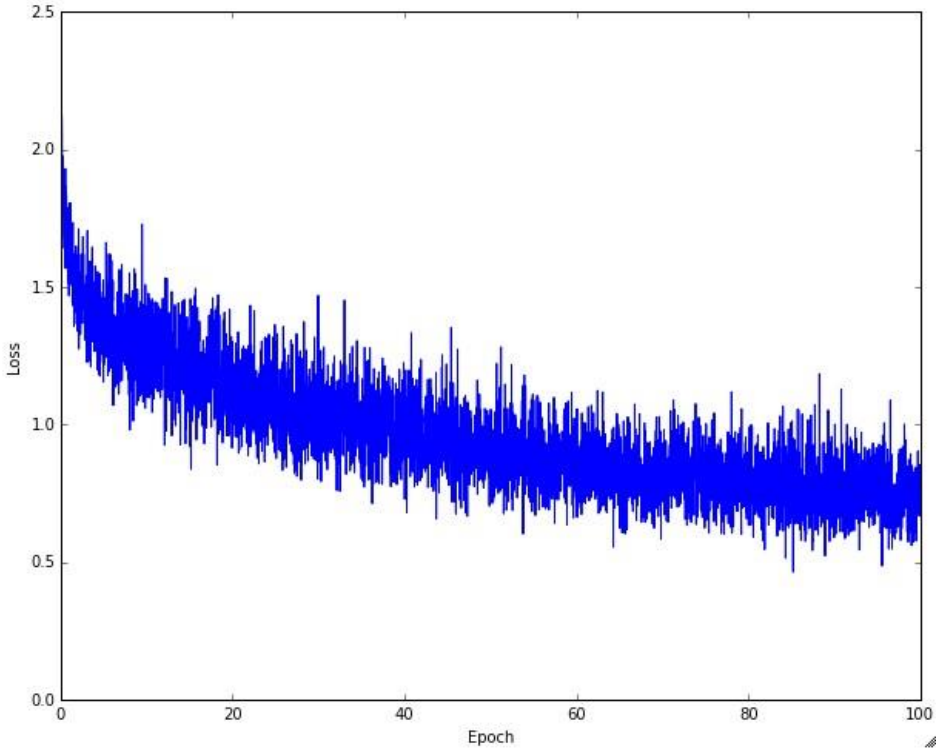
گرادیان کاهشی دسته‌ای

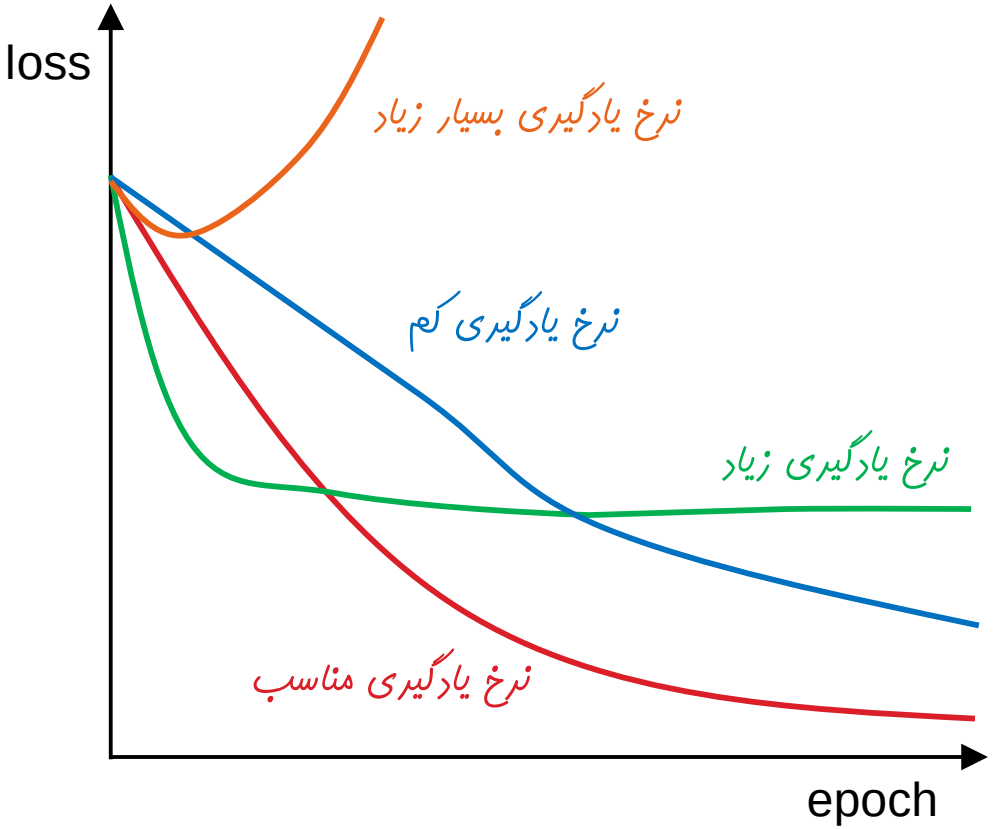
- اجرای الگوریتم گرادیان کاهشی دسته‌ای به منظور بهینه‌سازی وزن‌های یک شبکه عصبی.



هزینه در طول زمان کاهش می‌یابد.

تأثیر نرخ یادگیری (اندازه گام)





یک نسخه کاراتر از الگوریتم گرادیان کاهشی

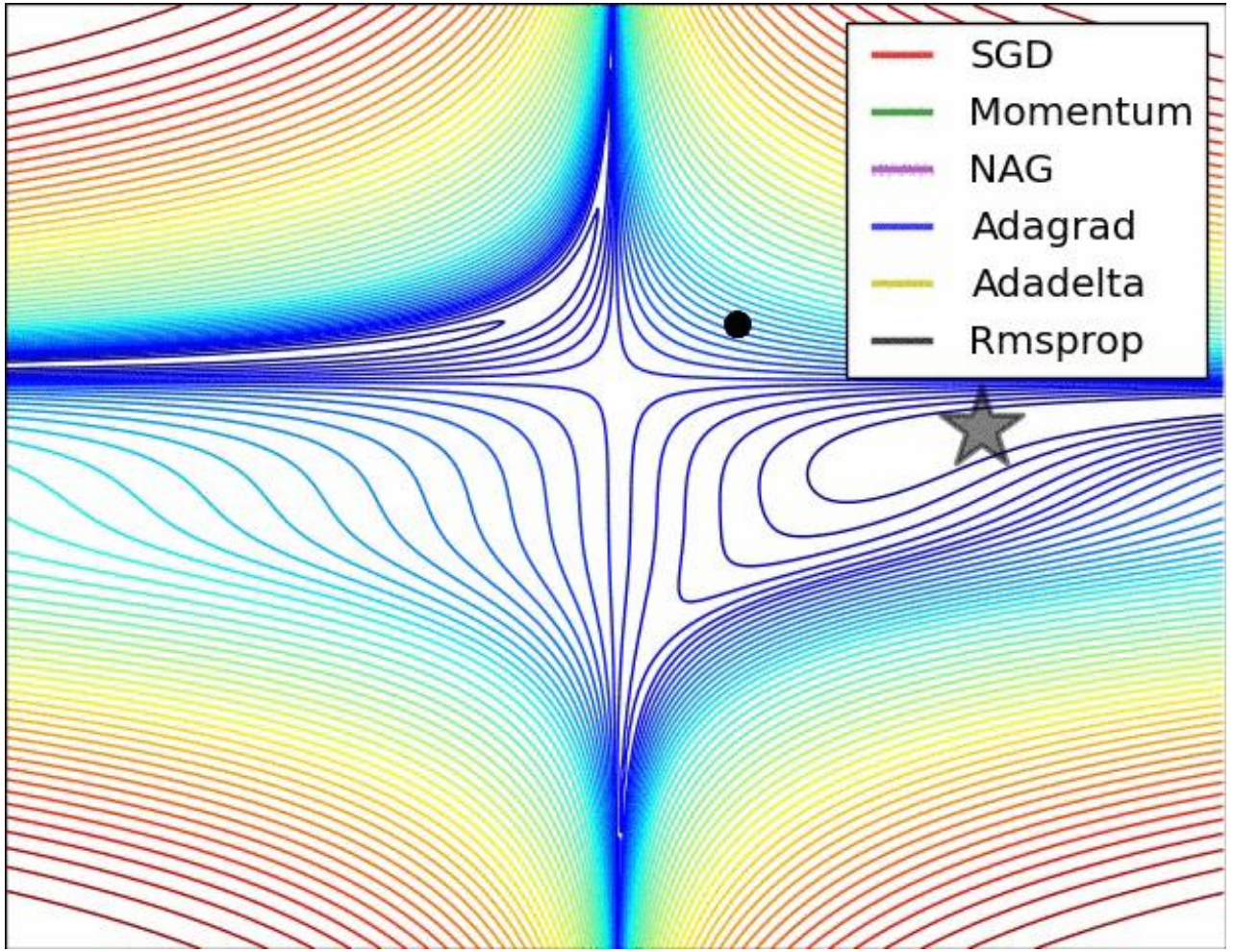
- گرادیان کاهشی دسته‌ای.
  - برای محاسبه گرادیان تابع هزینه در هر تکرار، تنها از بخش کوچکی از داده‌های آموزشی استفاده کن.

```
# Mini-batch Gradient Descent

while True:
    data_batch = sample_training_data(data, 256)  # sample 256 examples
    gradient = evaluate_gradient(loss_fun, data_batch, weights)
    weights += -step_size * gradient  # weight update
```

- مقادیر متداول برای اندازه‌ی دسته: ۳۲، ۶۴، ۱۲۸ و ۲۵۶.

روش‌های دیگر به روز رسانی مقدار وزن‌ها.
ممنتوم، آداگراد، آدام و غیره. [!!!]

به روز رسانی مقدار وزن‌ها



الگوریتم پس‌انتشار خطا

یادآوری: محاسبه گرادیان به صورت تحلیلی

- تابع هزینه یک تابع از پارامترهای W است.

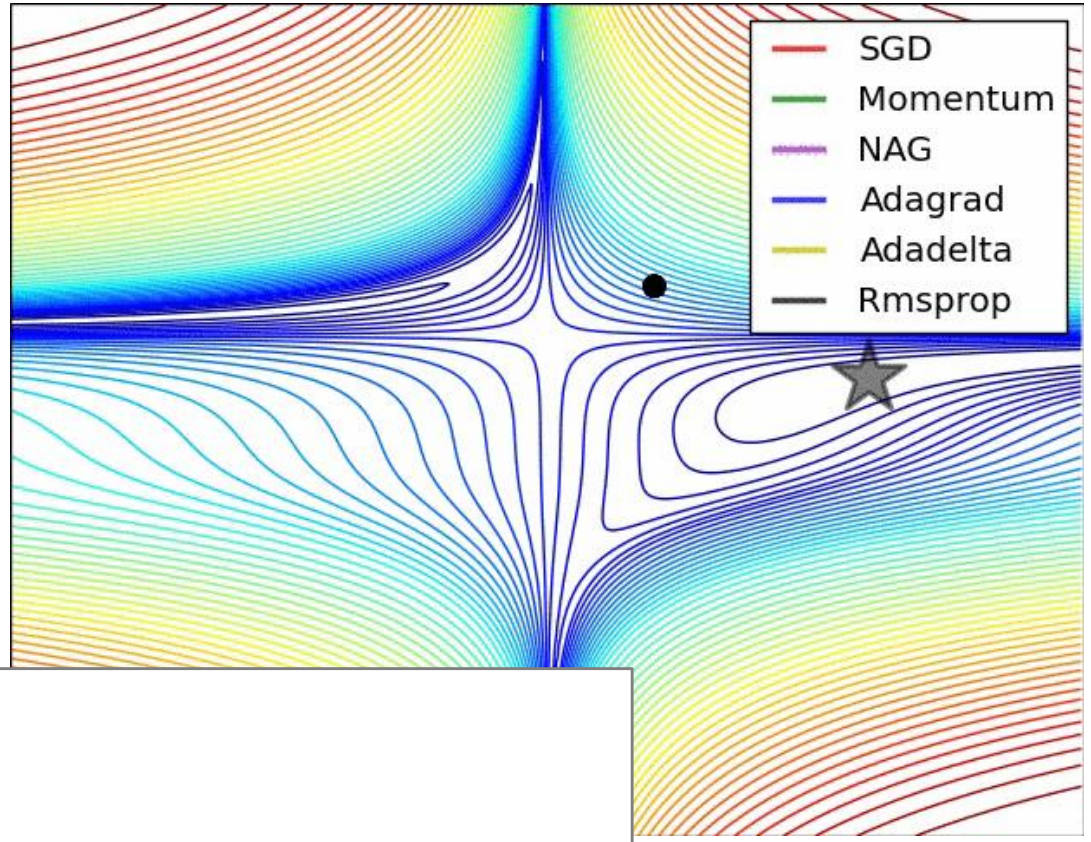
$$s = f(x; W) = Wx$$

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

$$L = \frac{1}{N} \sum_{i=1}^{N} L_i + \lambda \sum_k W_k^2$$

$$\nabla_W L = \cdots$$

یادآوری: بهینه‌سازی




```
# Vanilla Gradient Descent

while True:
    gradient = evaluate_gradient(loss_fun, data, weights)
    weights += -step_size * gradient  # weight update
```

یادآوری: بررسی گرادیان

- به طور خلاصه.
  - گرادیان عددی: تقریبی، زمان‌بر، پیاده‌سازی آسان!
  - گرادیان تحلیلی: دقیق، سریع، امکان بروز خطا در پیاده‌سازی!

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

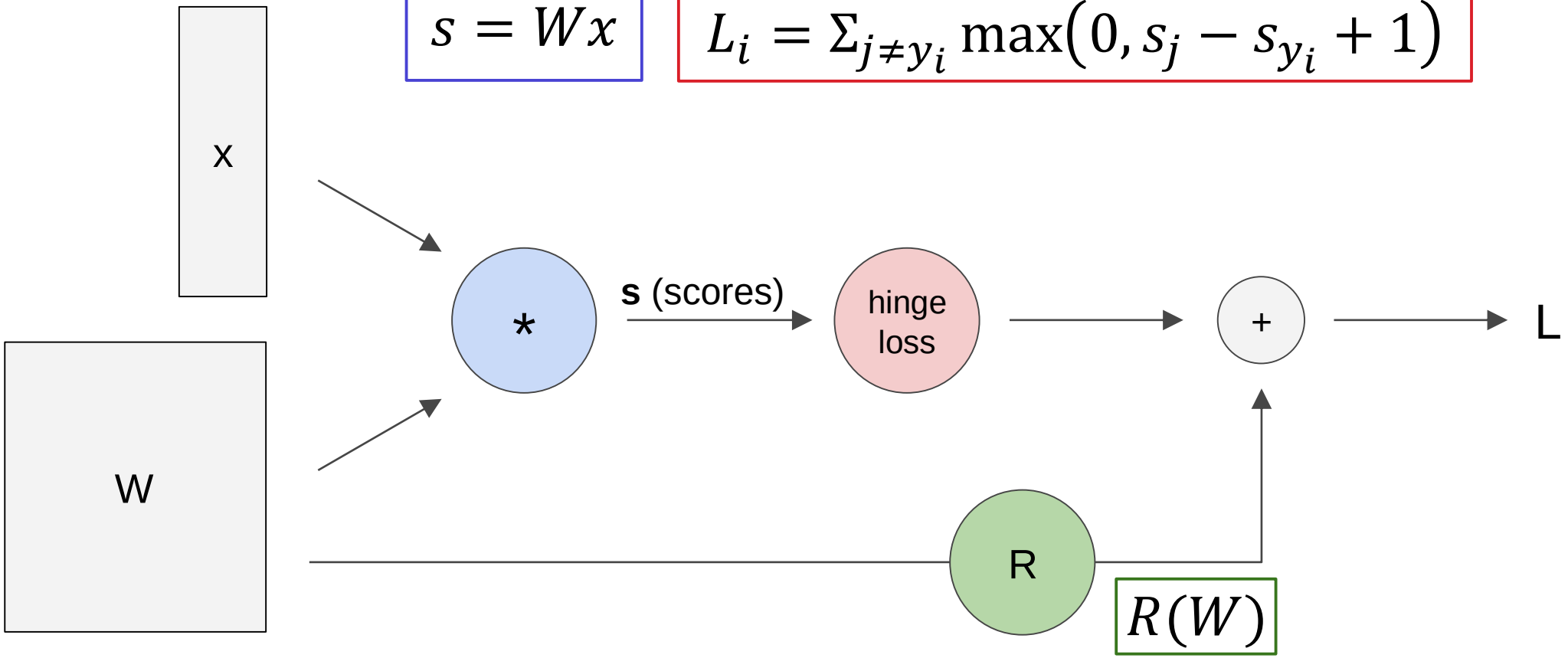
- در عمل.
  - همیشه از گرادیان تحلیلی استفاده می‌کنیم.
  - اما به منظور اطمینان از صحت پیاده‌سازی، گرادیان تحلیلی را با گرادیان عددی مقایسه می‌کنیم.

بررسی گرادیان

گراف محاسباتی

$$s = Wx$$

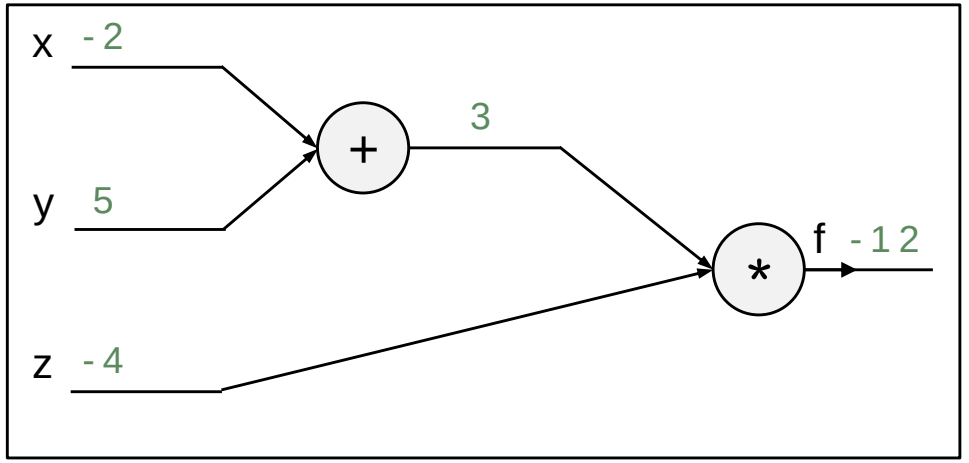
$$L_i = \Sigma_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$



گراف محاسباتی

$$f(x, y, z) = (x + y) \cdot z$$

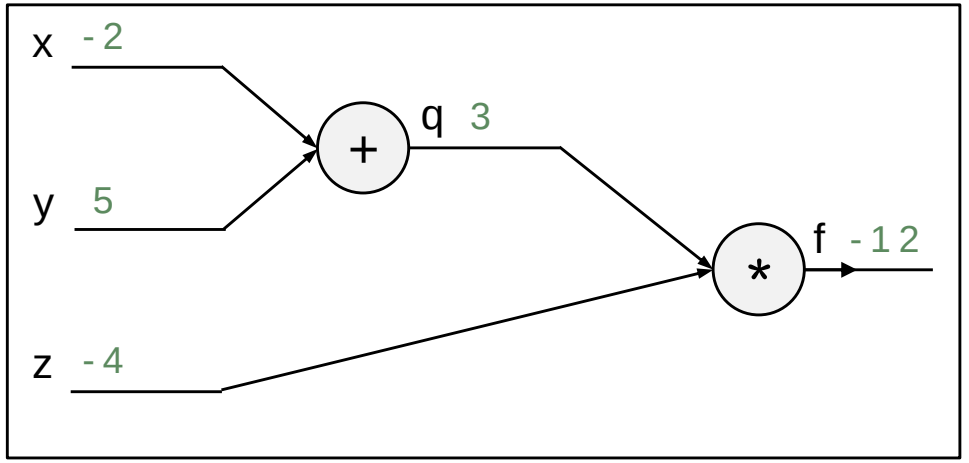
x = -2, y = 5, z = -4



گراف محاسباتی

$$f(x,y,z) = (x+y)\cdot z$$

x = -2, y = 5, z = -4



$$q = x + y \qquad \frac{\partial q}{\partial x}=1,\ \frac{\partial q}{\partial y}=1$$

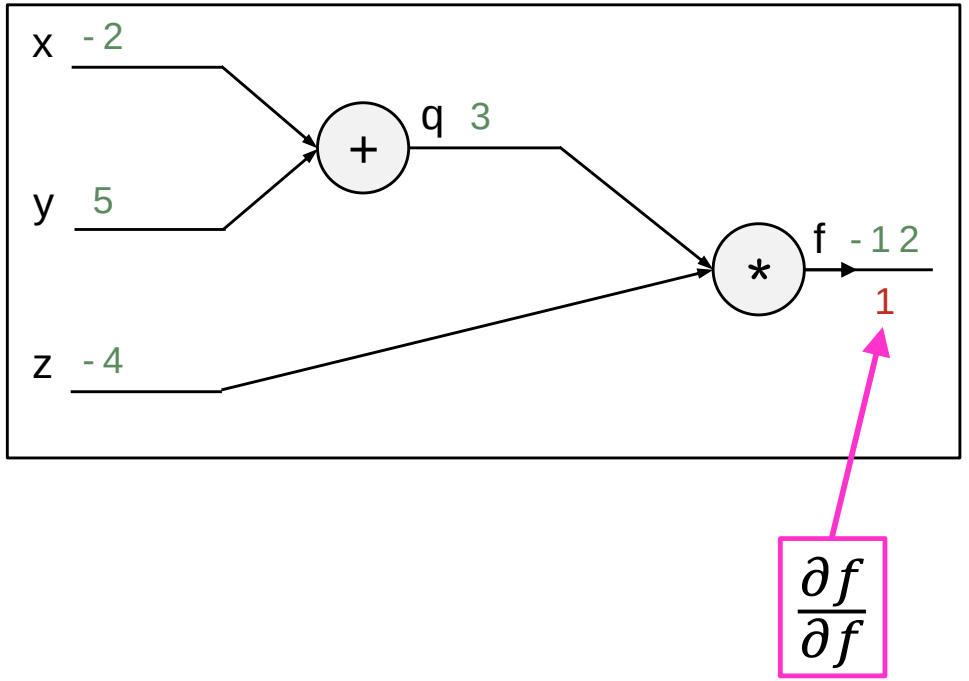
$$f = qz \qquad \frac{\partial f}{\partial q}=z,\ \frac{\partial q}{\partial z}=q$$

$\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$ مقادیر مورد نیاز

گراف محاسباتی

$$f(x,y,z) = (x+y)\cdot z$$

x = -2, y = 5, z = -4



$$q = x + y \qquad \frac{\partial q}{\partial x}=1,\ \frac{\partial q}{\partial y}=1$$

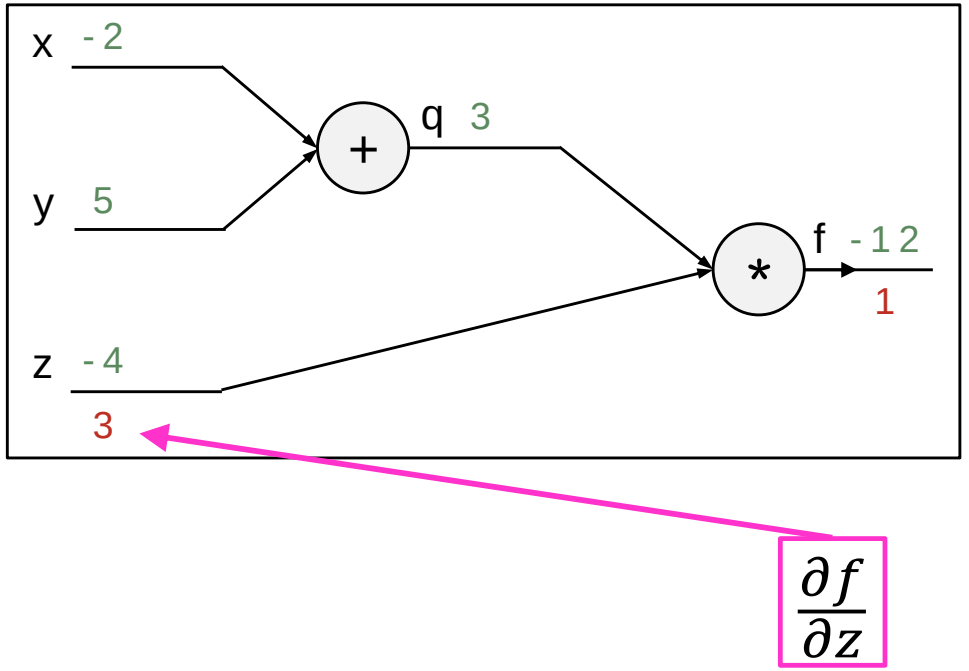
$$f = qz \qquad \frac{\partial f}{\partial q}=z,\ \frac{\partial q}{\partial z}=q$$

$\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$ مقادیر مورد نیاز

گراف محاسباتی

$$f(x,y,z) = (x+y)\cdot z$$

x = -2, y = 5, z = -4



$$q = x + y \qquad \frac{\partial q}{\partial x}=1,\ \frac{\partial q}{\partial y}=1$$

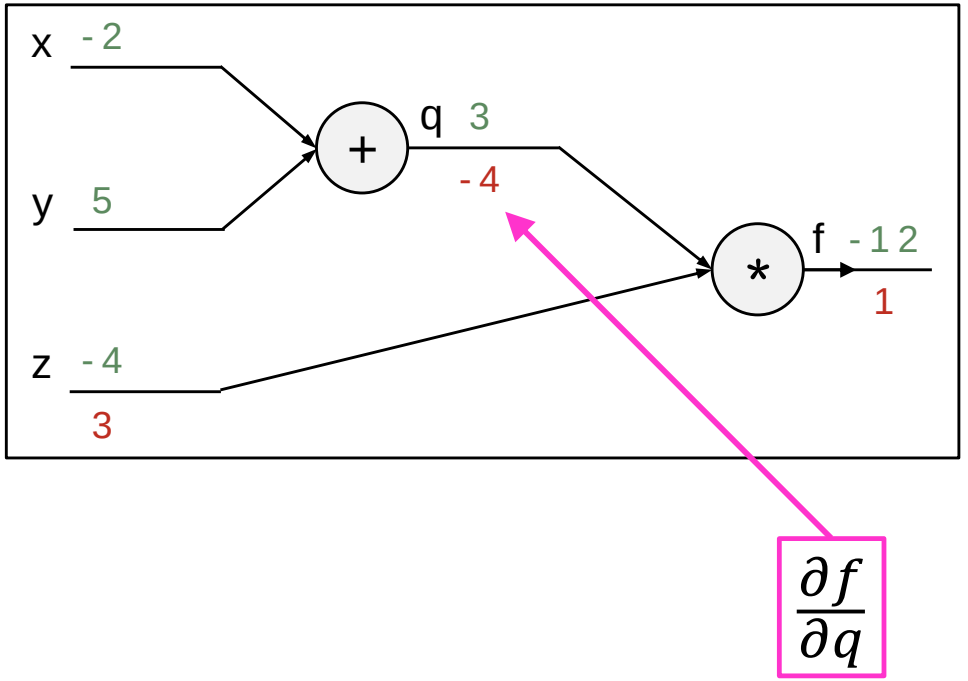
$$f = qz \qquad \frac{\partial f}{\partial q}=z,\ \frac{\partial q}{\partial z}=q$$

$\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$ مقادیر مورد نیاز

گراف محاسباتی

$$f(x,y,z) = (x+y)\cdot z$$

x = -2, y = 5, z = -4



$$q = x + y \qquad \frac{\partial q}{\partial x}=1,\ \frac{\partial q}{\partial y}=1$$

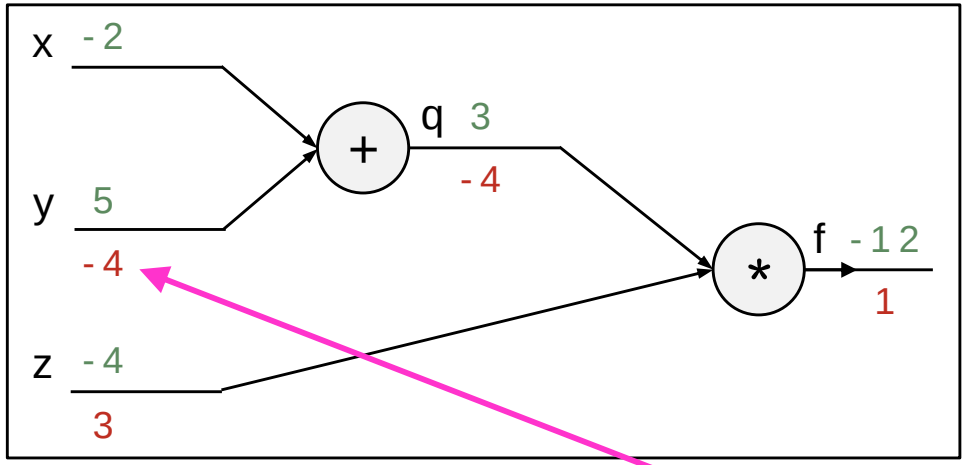
$$f = qz \qquad \frac{\partial f}{\partial q}=z,\ \frac{\partial q}{\partial z}=q$$

$\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$ مقادیر مورد نیاز

گراف محاسباتی

$$f(x,y,z) = (x+y)\cdot z$$

x = -2, y = 5, z = -4



$$q = x + y \qquad \frac{\partial q}{\partial x}=1,\ \frac{\partial q}{\partial y}=1$$

$$f = qz \qquad \frac{\partial f}{\partial q}=z,\ \frac{\partial q}{\partial z}=q$$

$\frac{\partial f}{\partial y}$

قاعده‌ی زنجیری

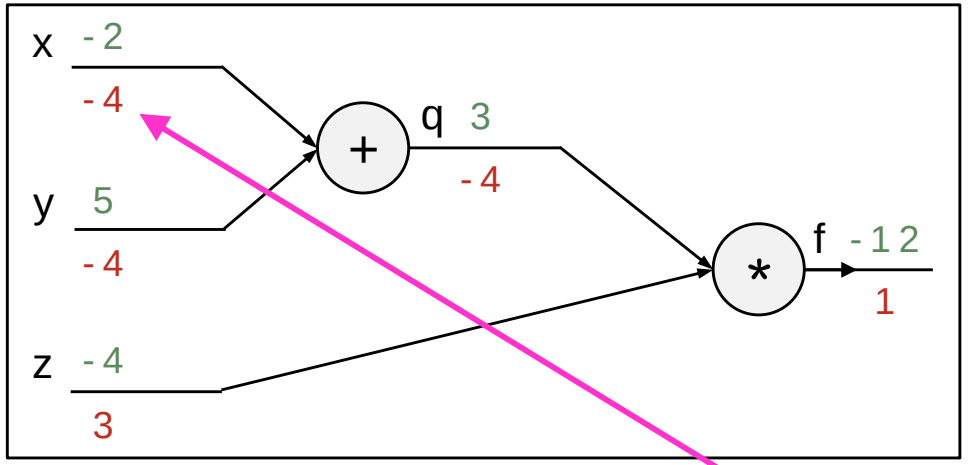
$$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial q}\cdot\frac{\partial q}{\partial y}$$

$\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$ مقادیر مورد نیاز

گراف محاسباتی

$$f(x,y,z) = (x+y)\cdot z$$

x = -2, y = 5, z = -4



$$q = x + y \qquad \frac{\partial q}{\partial x}=1,\ \frac{\partial q}{\partial y}=1$$

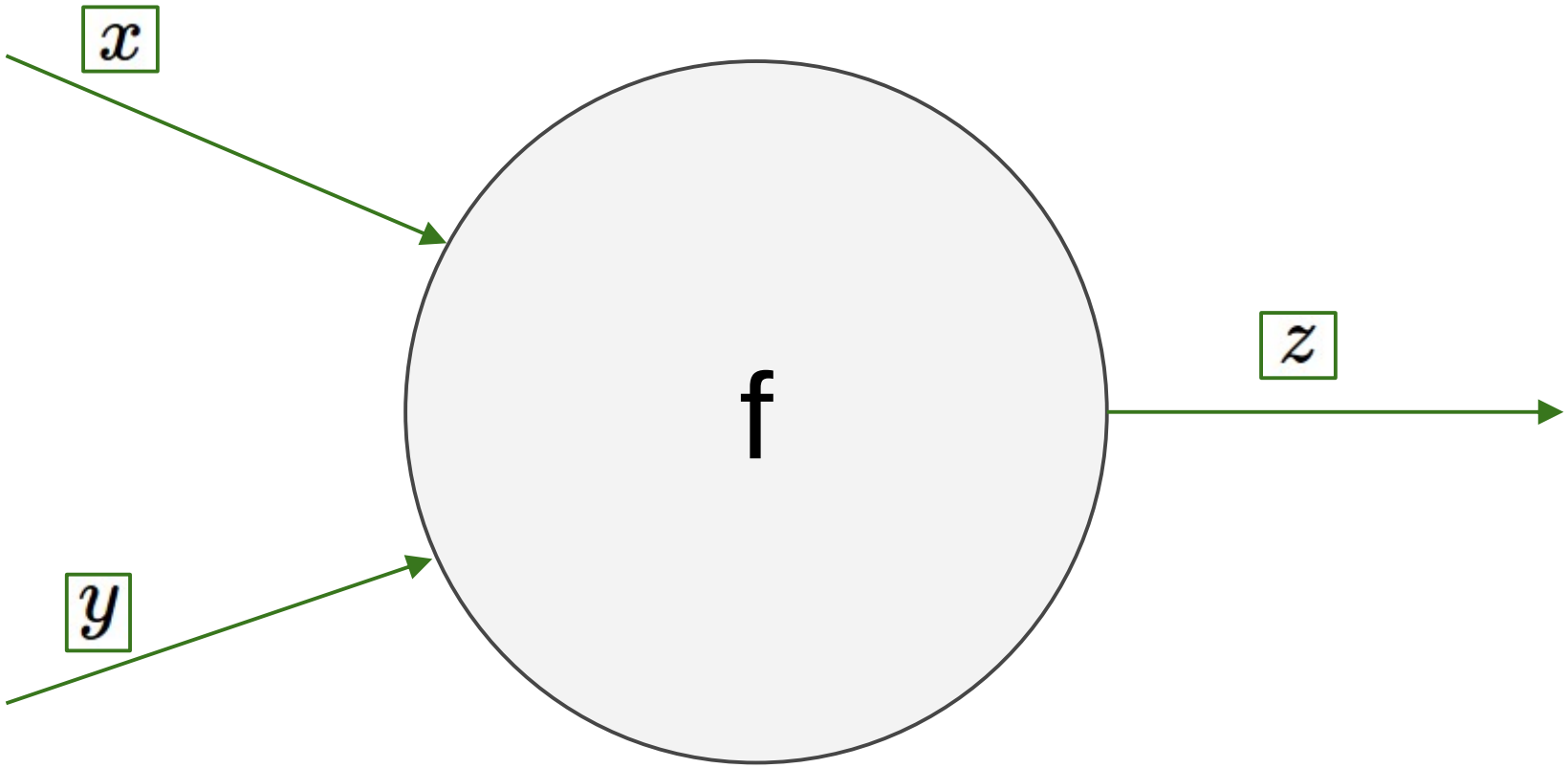
$$f = qz \qquad \frac{\partial f}{\partial q}=z,\ \frac{\partial q}{\partial z}=q$$

$\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$ مقادیر مورد نیاز

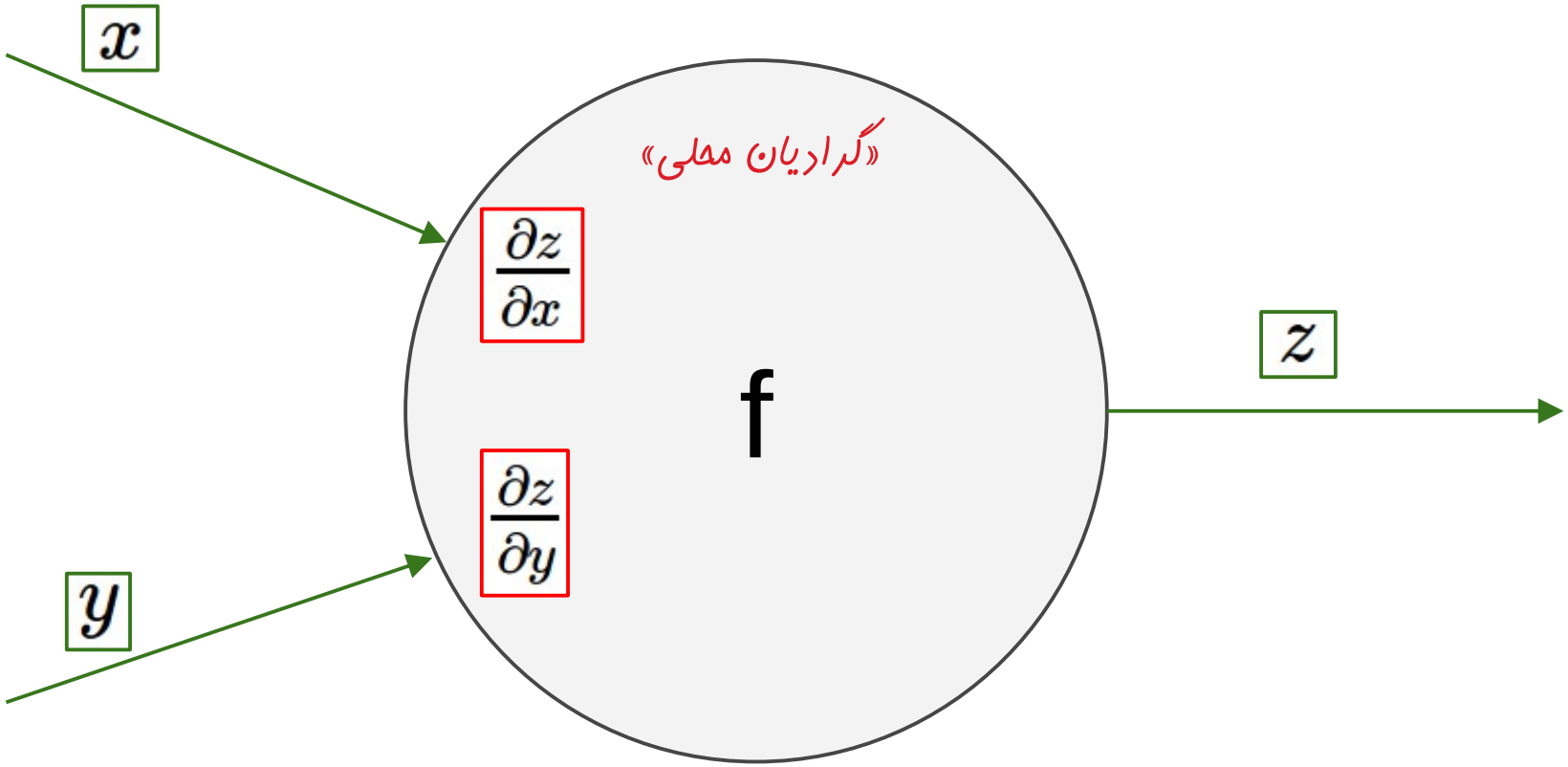
قاعده‌ی زنجیری

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q}\cdot\frac{\partial q}{\partial x}$$

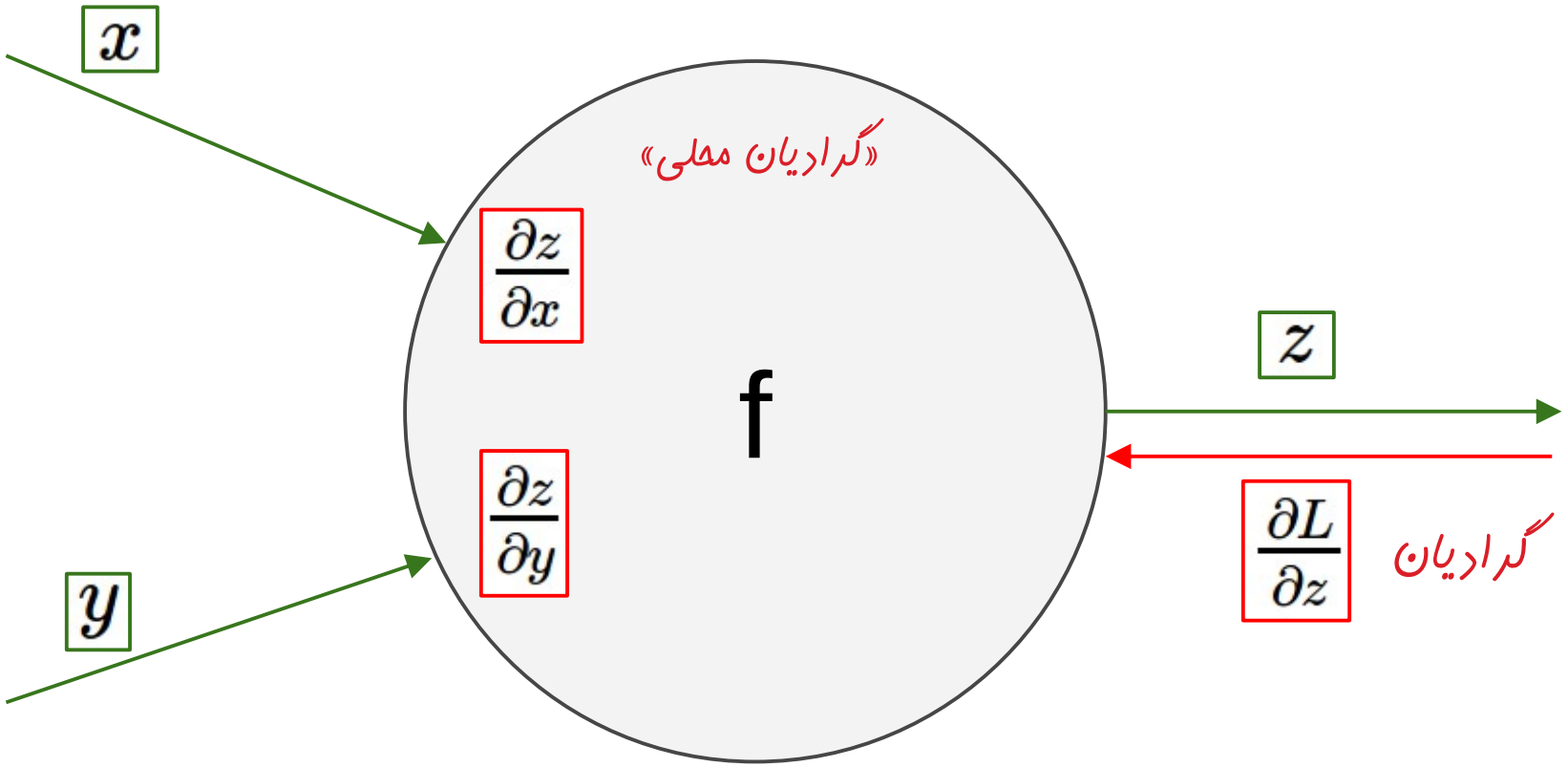
گراف محاسباتی



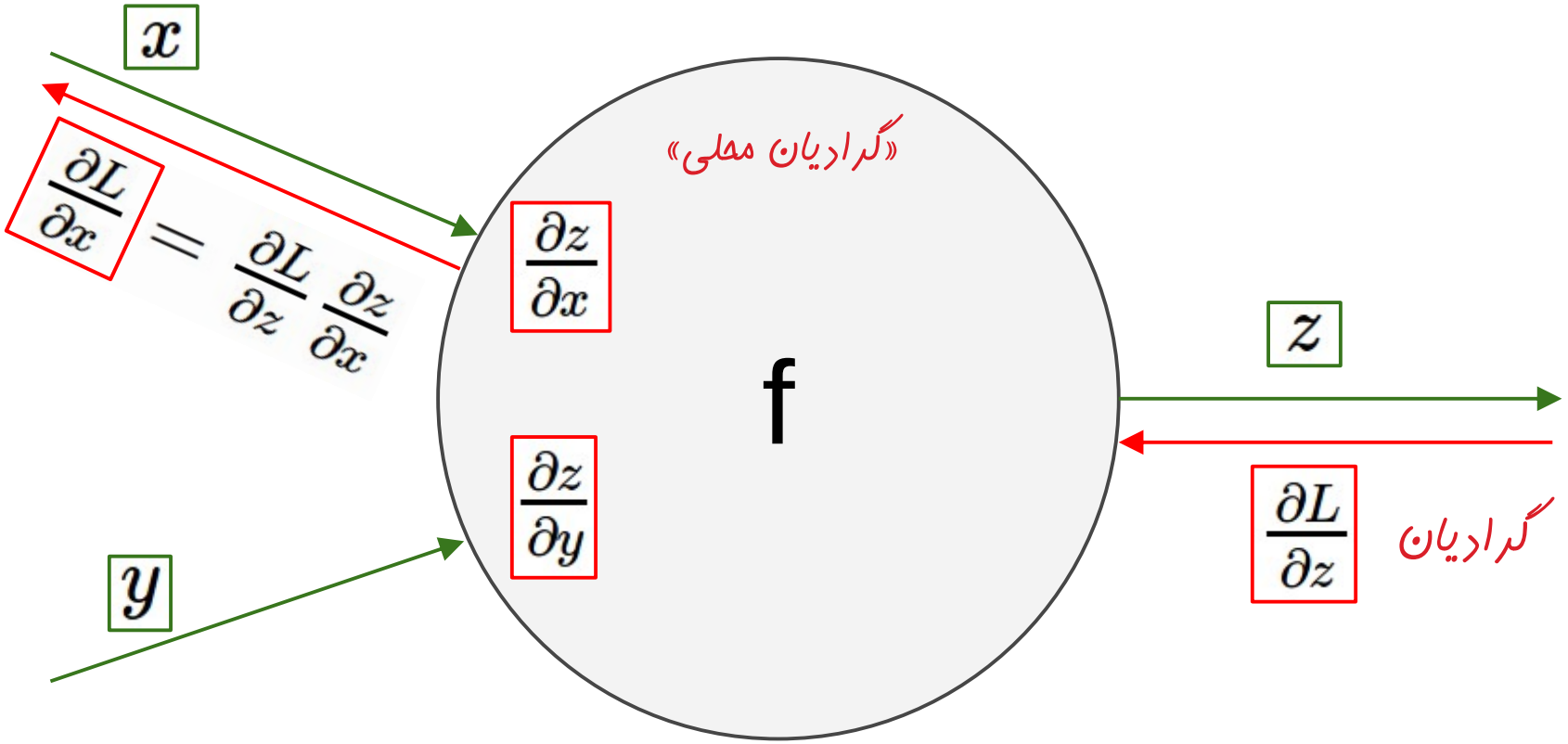
گراف محاسباتی



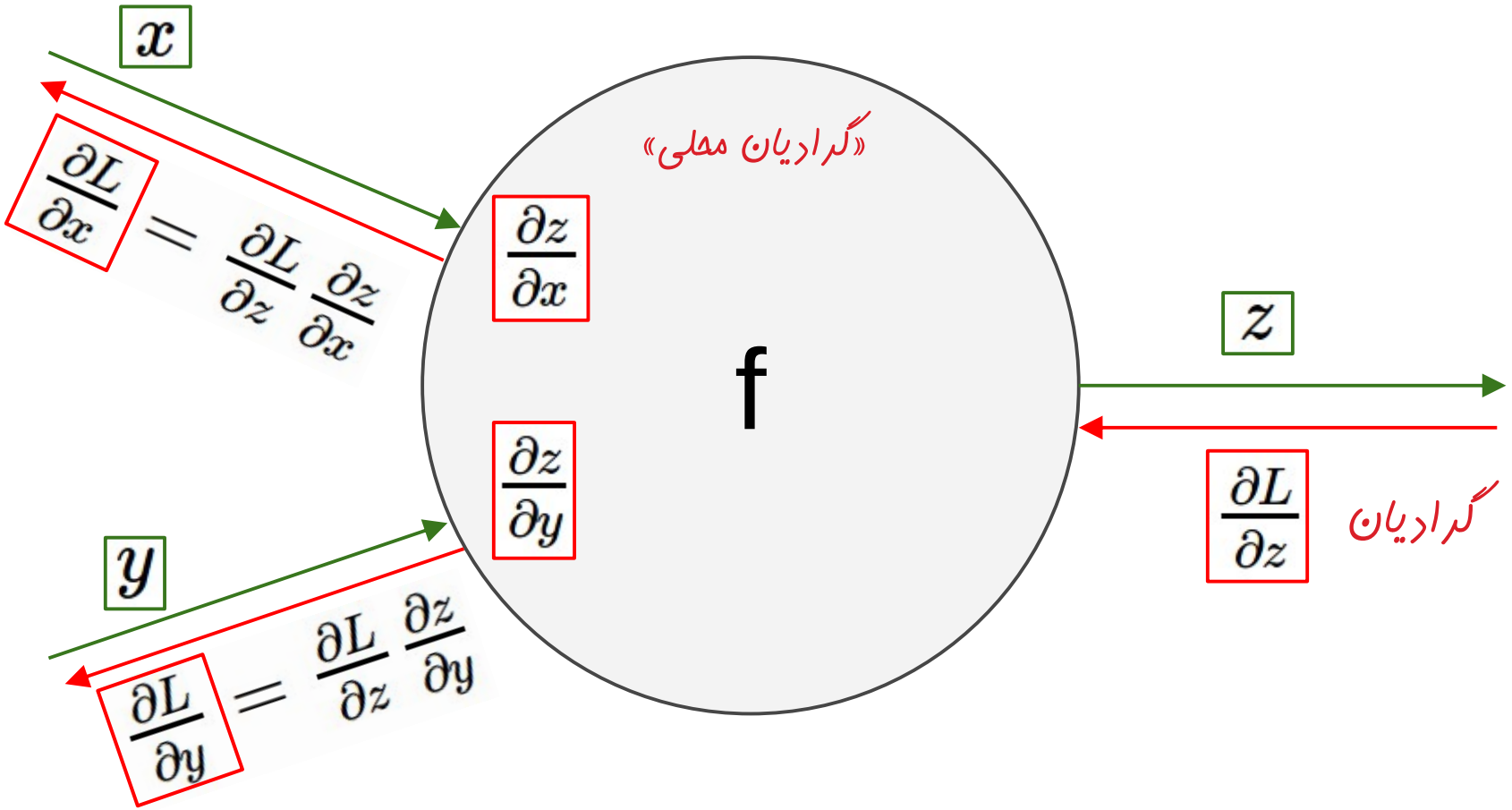
گراف محاسباتی



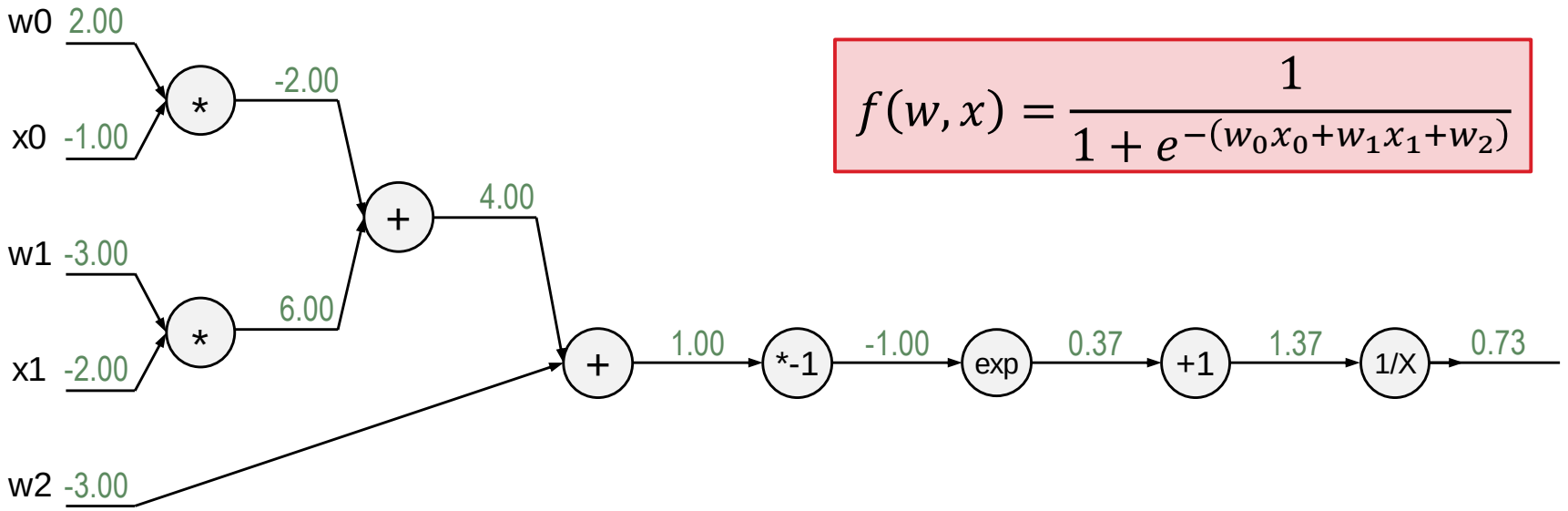
گراف محاسباتی



گراف محاسباتی



گراف محاسباتی: یک مثال دیگر



$$f(w,x) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w_2)}}$$

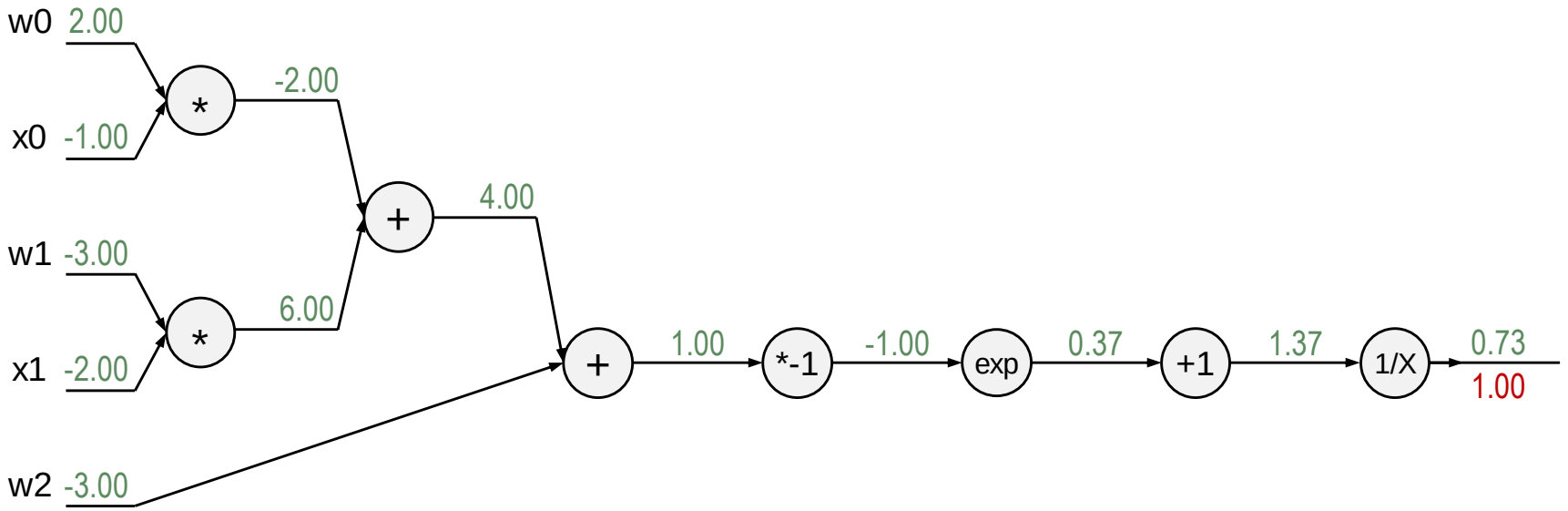
$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

$$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

$$f_c(x) = x + c \quad \rightarrow \quad \frac{df}{dx} = 1$$

گراف محاسباتی: یک مثال دیگر



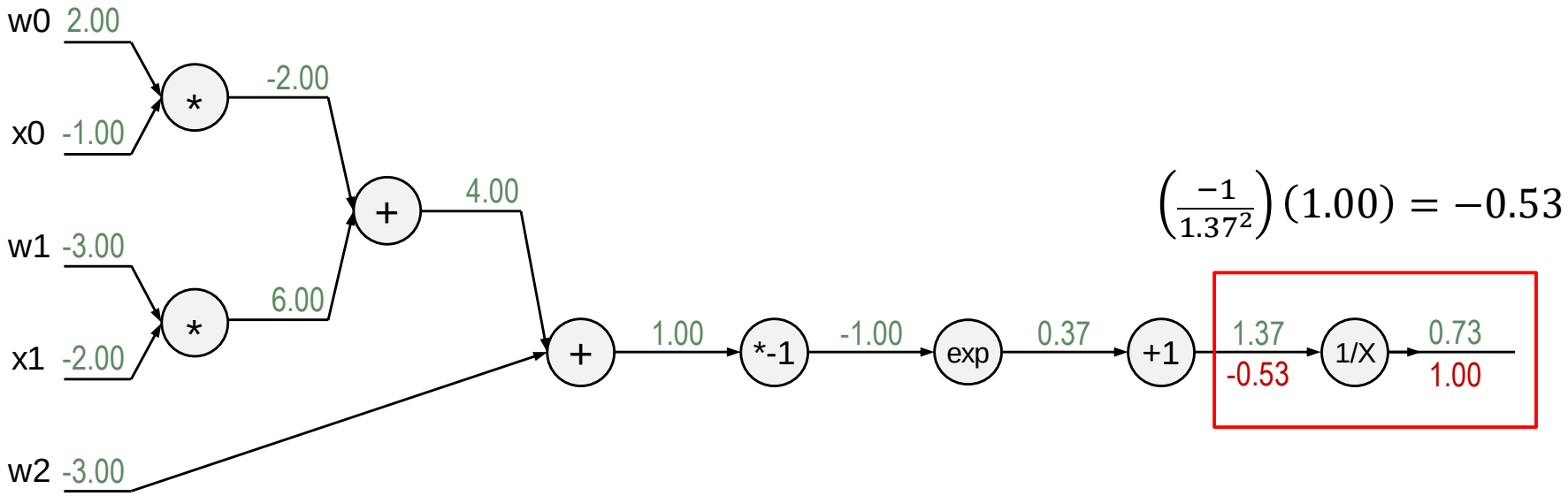
$$f(x) = e^x \rightarrow \frac{df}{dx} = e^x$$

$$f_a(x) = ax \rightarrow \frac{df}{dx} = a$$

$$f(x) = \frac{1}{x} \rightarrow \frac{df}{dx} = -1/x^2$$

$$f_c(x) = x + c \rightarrow \frac{df}{dx} = 1$$

گراف محاسباتی: یک مثال دیگر



$$\left(\frac{-1}{1.37^2}\right)(1.00) = -0.53$$

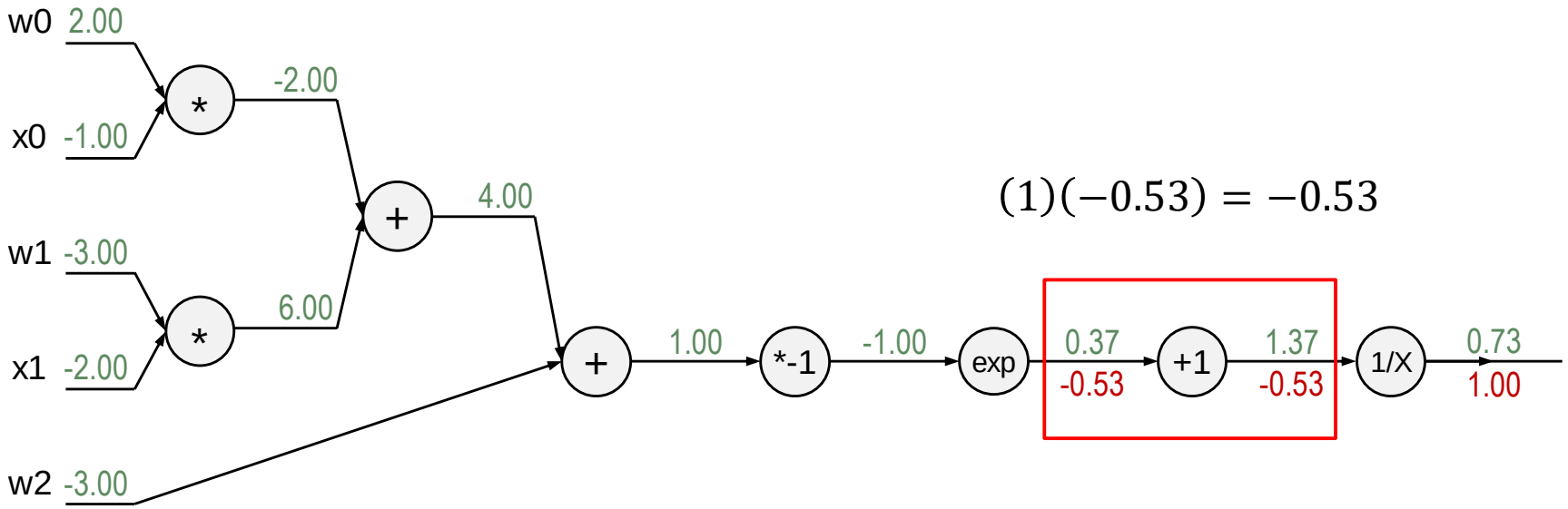
$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

$$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

$$f_c(x) = x + c \quad \rightarrow \quad \frac{df}{dx} = 1$$

گراف محاسباتی: یک مثال دیگر



$$(1)(-0.53) = -0.53$$

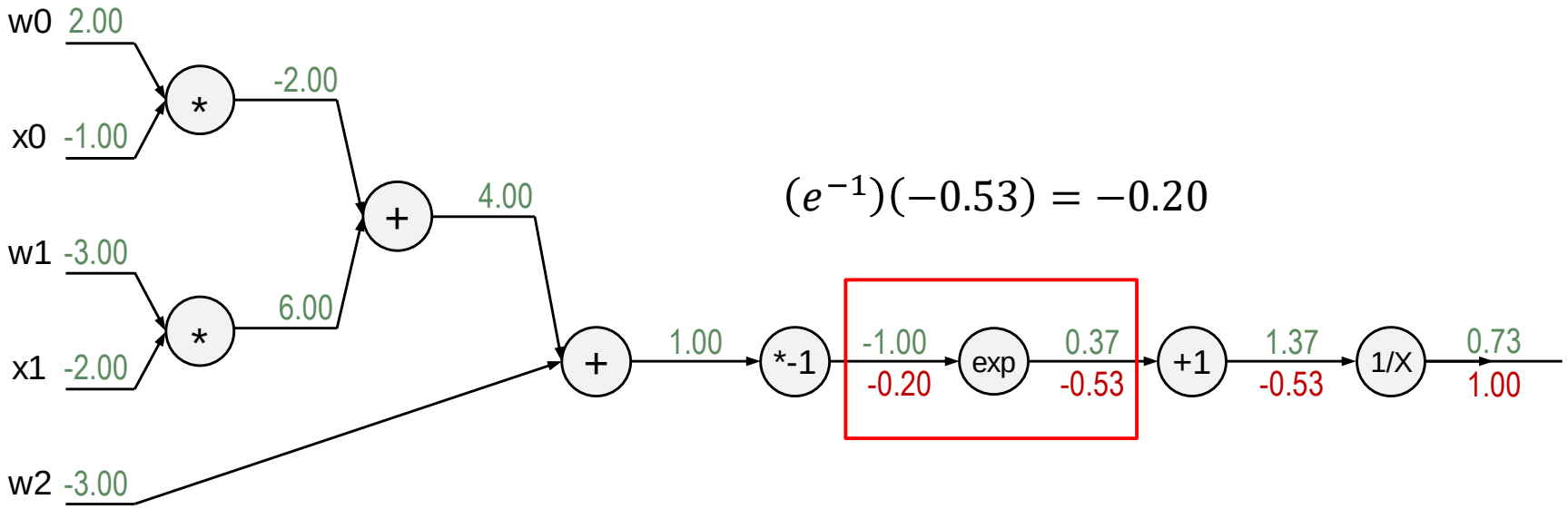
$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

$$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

$$f_c(x) = x + c \quad \rightarrow \quad \frac{df}{dx} = 1$$

گراف محاسباتی: یک مثال دیگر



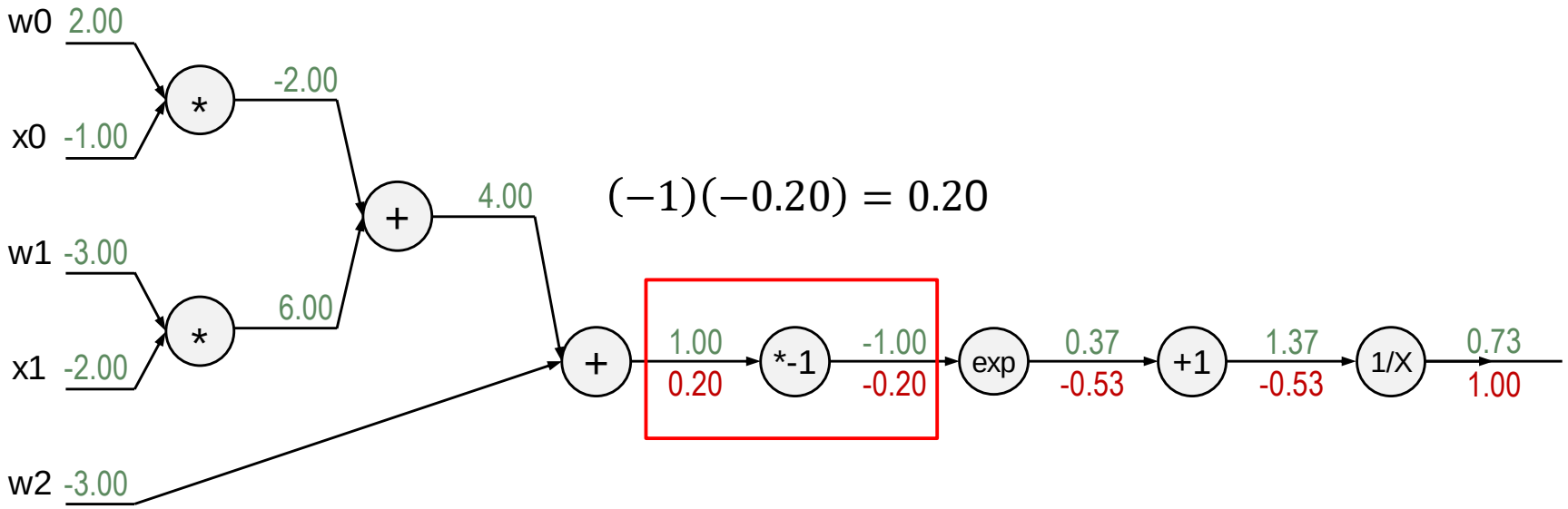
$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

$$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

$$f_c(x) = x + c \quad \rightarrow \quad \frac{df}{dx} = 1$$

گراف محاسباتی: یک مثال دیگر



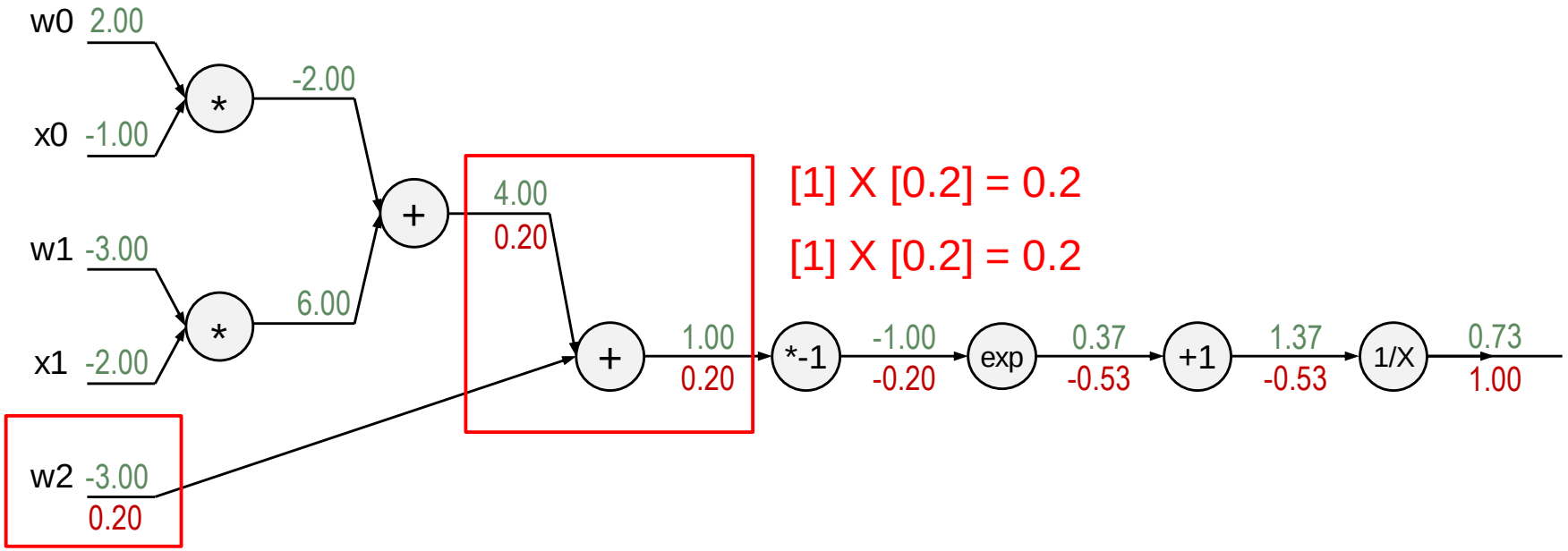
$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

$$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

$$f_c(x) = x + c \quad \rightarrow \quad \frac{df}{dx} = 1$$

گراف محاسباتی: یک مثال دیگر



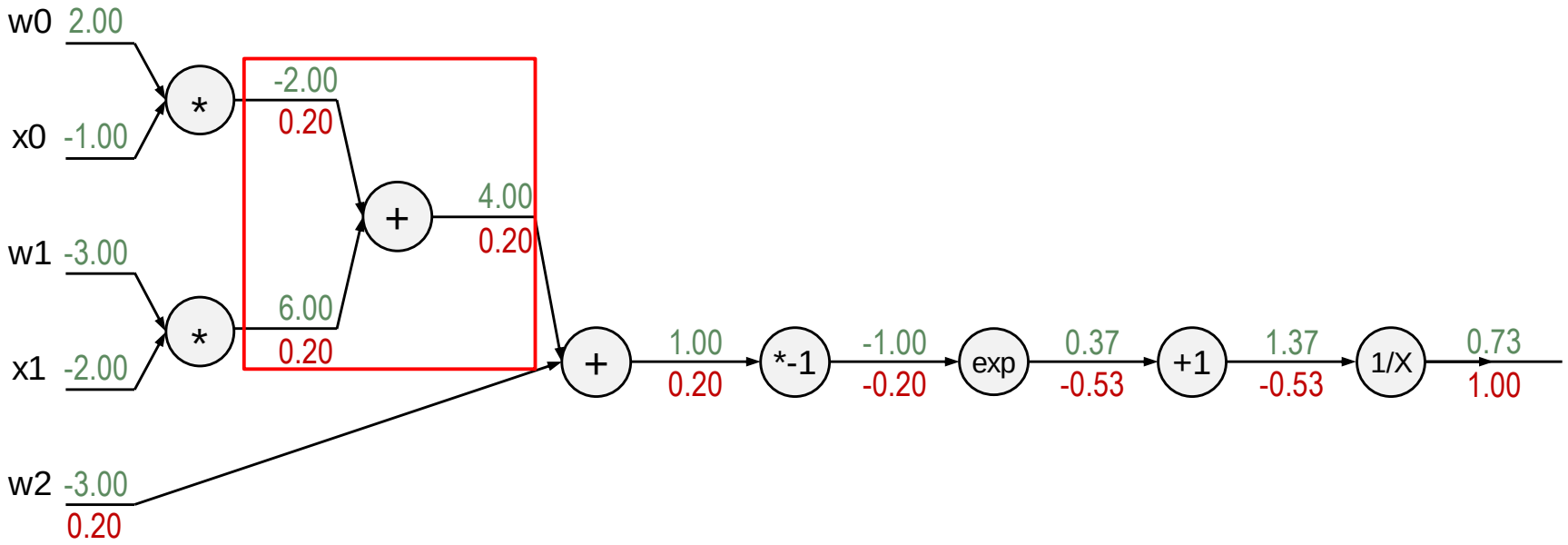
$$f(x) = e^x \rightarrow \frac{df}{dx} = e^x$$

$$f_a(x) = ax \rightarrow \frac{df}{dx} = a$$

$$f(x) = \frac{1}{x} \rightarrow \frac{df}{dx} = -1/x^2$$

$$f_c(x) = x + c \rightarrow \frac{df}{dx} = 1$$

گراف محاسباتی: یک مثال دیگر



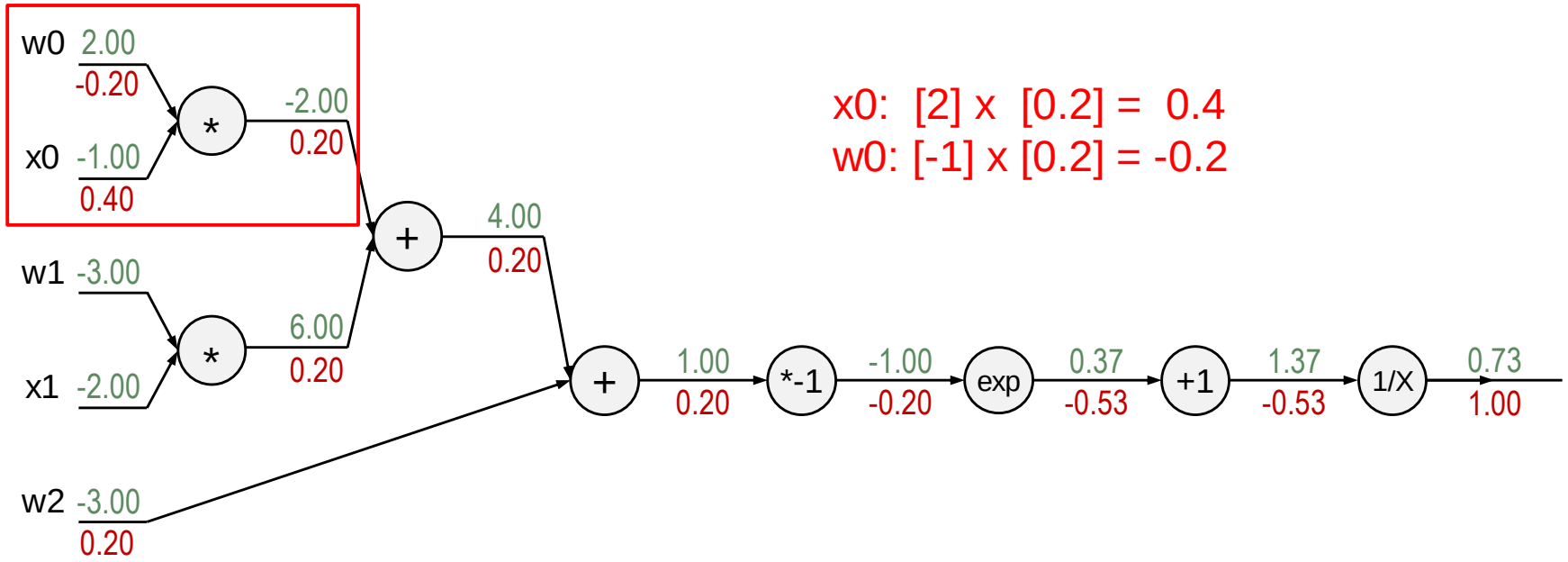
$$f(x) = e^x \rightarrow \frac{df}{dx} = e^x$$

$$f_a(x) = ax \rightarrow \frac{df}{dx} = a$$

$$f(x) = \frac{1}{x} \rightarrow \frac{df}{dx} = -1/x^2$$

$$f_c(x) = x + c \rightarrow \frac{df}{dx} = 1$$

گراف محاسباتی: یک مثال دیگر



x0: [2] x [0.2] = 0.4
w0: [-1] x [0.2] = -0.2

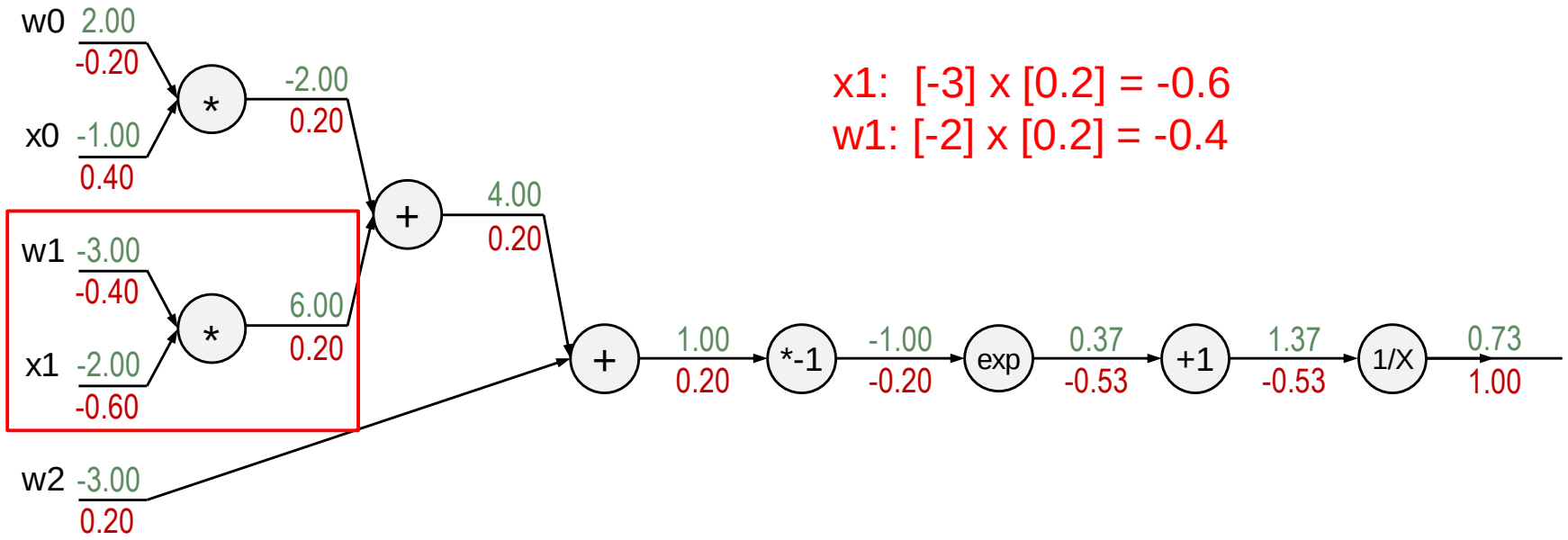
$$f(x) = e^x \rightarrow \frac{df}{dx} = e^x$$

$$f_a(x) = ax \rightarrow \frac{df}{dx} = a$$

$$f(x) = \frac{1}{x} \rightarrow \frac{df}{dx} = -1/x^2$$

$$f_c(x) = x + c \rightarrow \frac{df}{dx} = 1$$

گراف محاسباتی: یک مثال دیگر



x1: [-3] x [0.2] = -0.6
w1: [-2] x [0.2] = -0.4

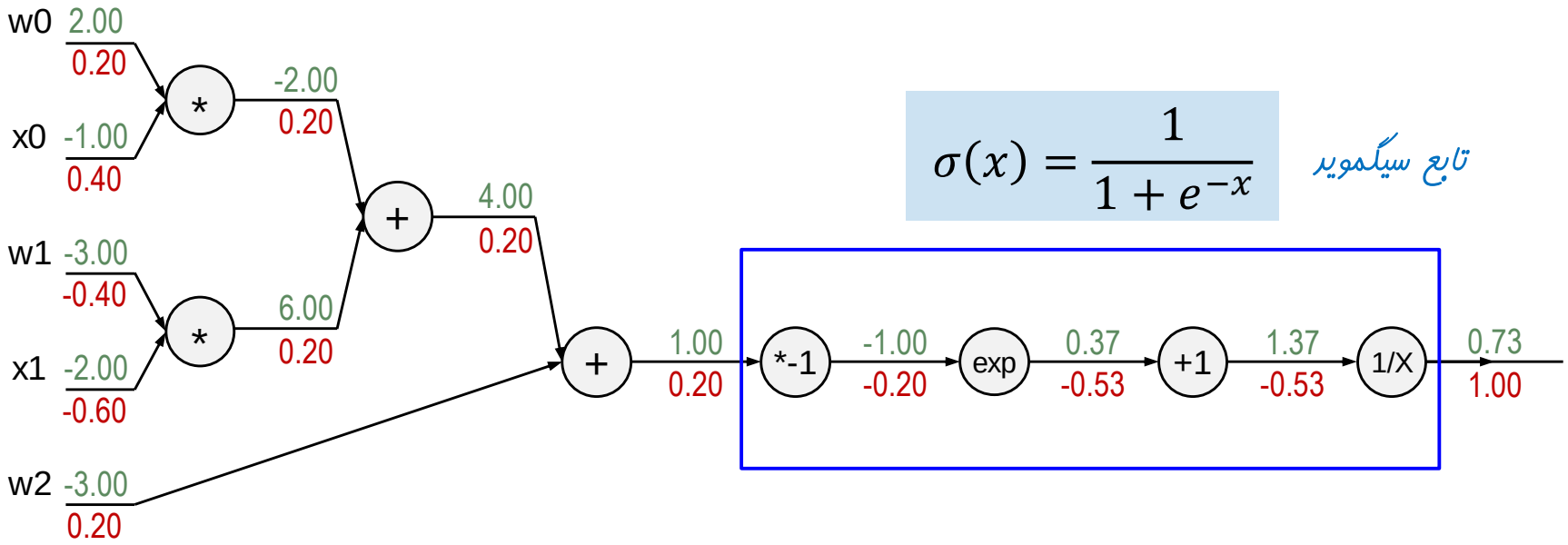
$$f(x) = e^x \rightarrow \frac{df}{dx} = e^x$$

$$f_a(x) = ax \rightarrow \frac{df}{dx} = a$$

$$f(x) = \frac{1}{x} \rightarrow \frac{df}{dx} = -1/x^2$$

$$f_c(x) = x + c \rightarrow \frac{df}{dx} = 1$$

گراف محاسباتی: یک مثال دیگر

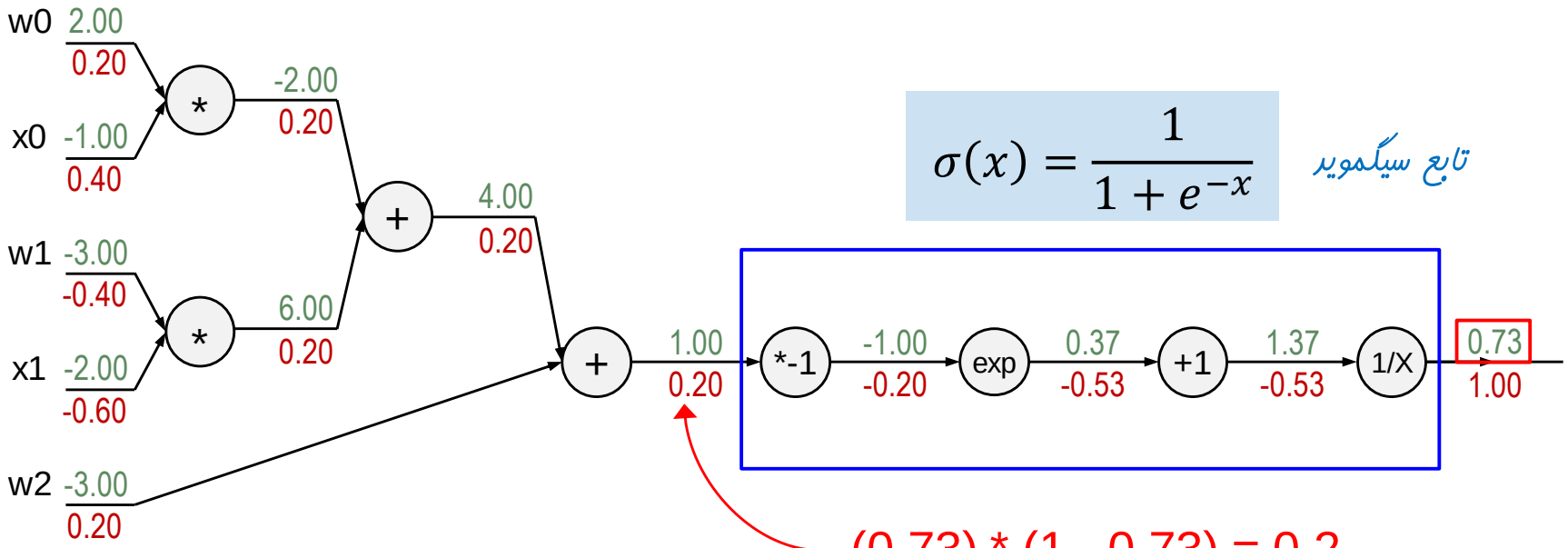


$$\sigma(x) = \frac{1}{1+e^{-x}}$$

تابع سیگموید

$$\frac{d\sigma(x)}{dx} = \frac{e^{-x}}{(1+e^{-x})^2} = \left(\frac{1}{1+e^{-x}}\right)\left(\frac{1+e^{-x}-1}{1+e^{-x}}\right) = \sigma(x)\big(1-\sigma(x)\big)$$

گراف محاسباتی: یک مثال دیگر



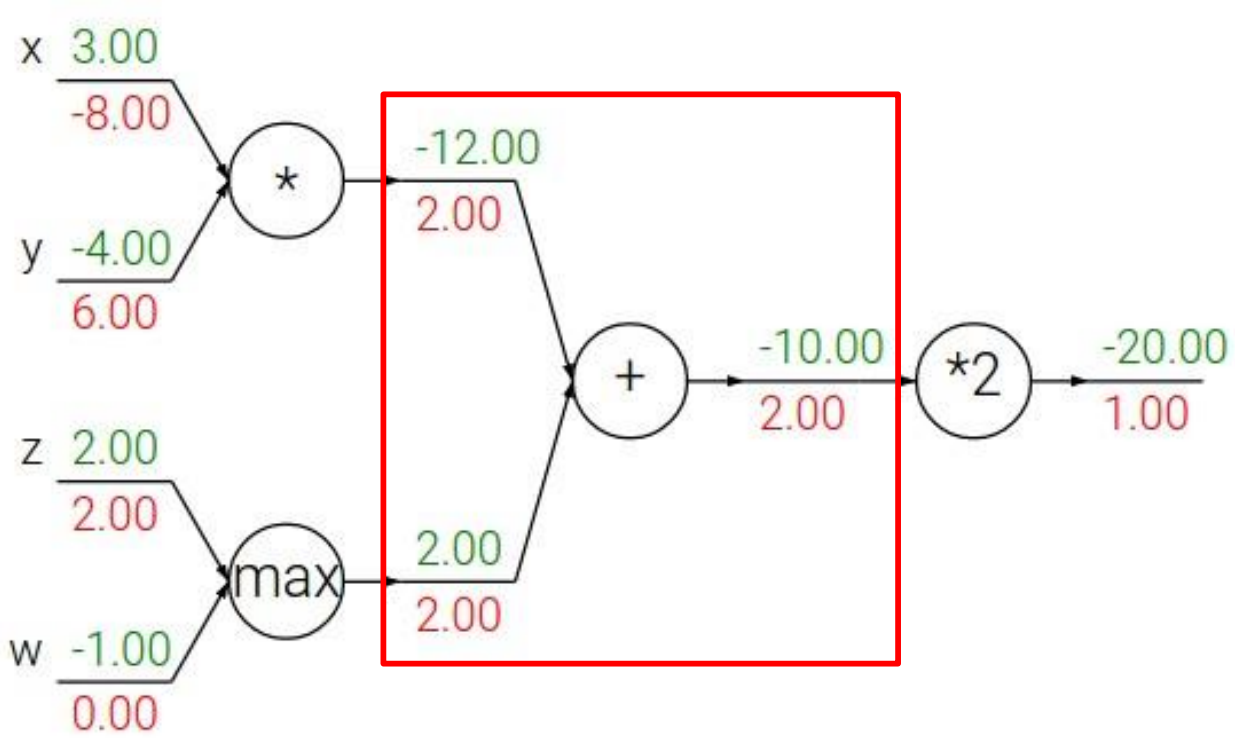
$$\sigma(x) = \frac{1}{1+e^{-x}}$$

تابع سیگموید

$$\frac{d\sigma(x)}{dx} = \frac{e^{-x}}{(1+e^{-x})^2} = \left(\frac{1}{1+e^{-x}}\right)\left(\frac{1+e^{-x}-1}{1+e^{-x}}\right) = \sigma(x)\big(1-\sigma(x)\big)$$

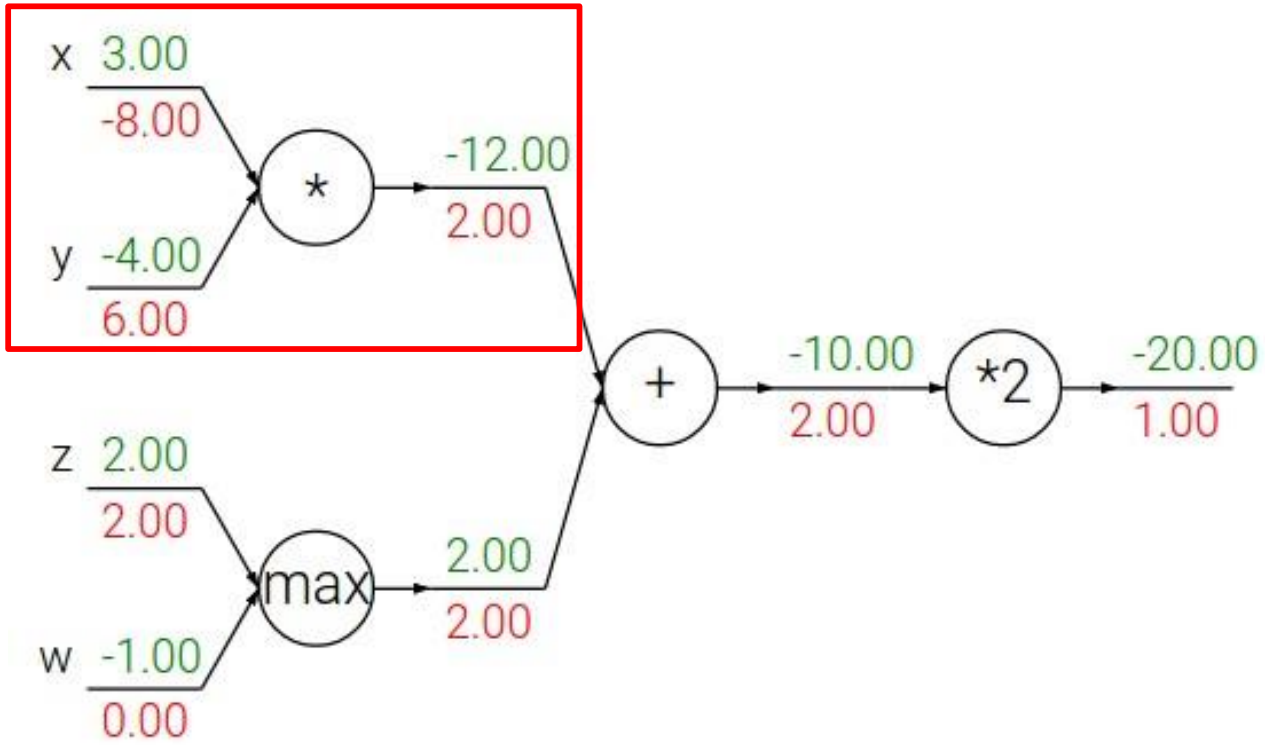
چند الگو در محاسبات رو به عقب

- عمل جمع. پخش کننده گرادیان!



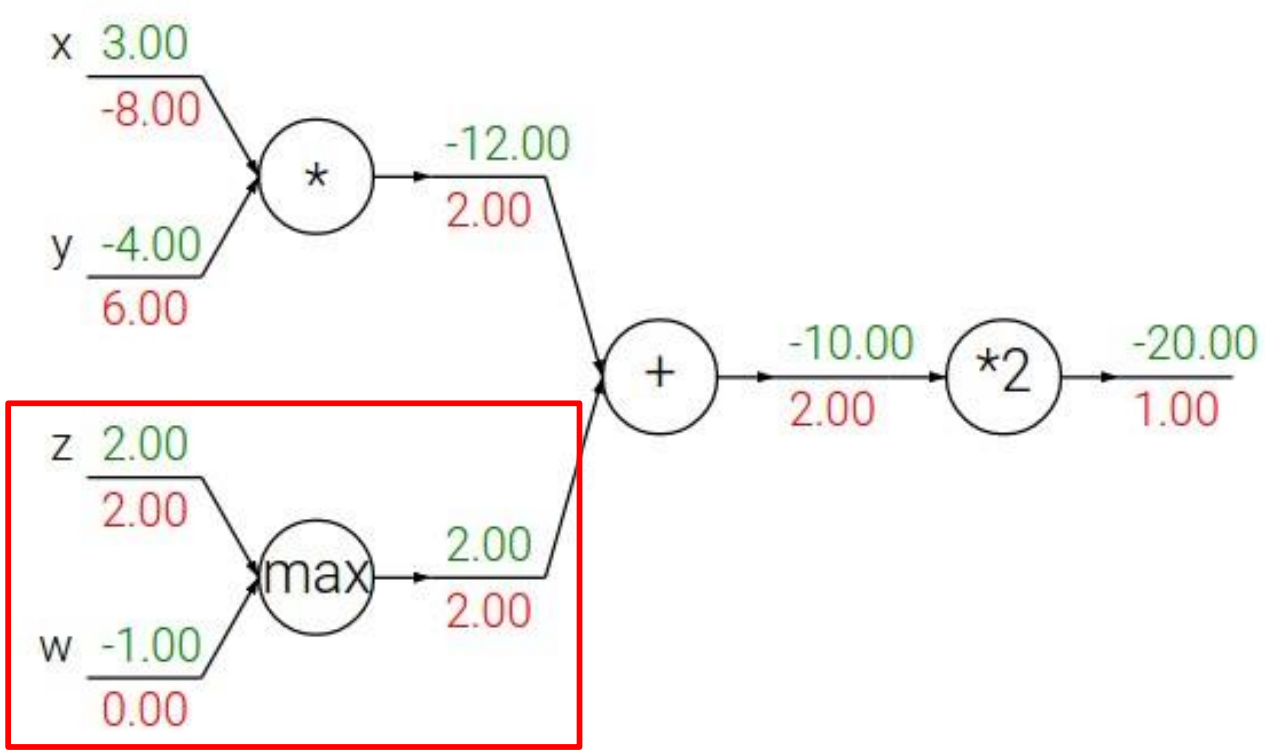
چند الگو در محاسبات رو به عقب

- عمل ضرب. جابه‌جا کننده گرادیان!



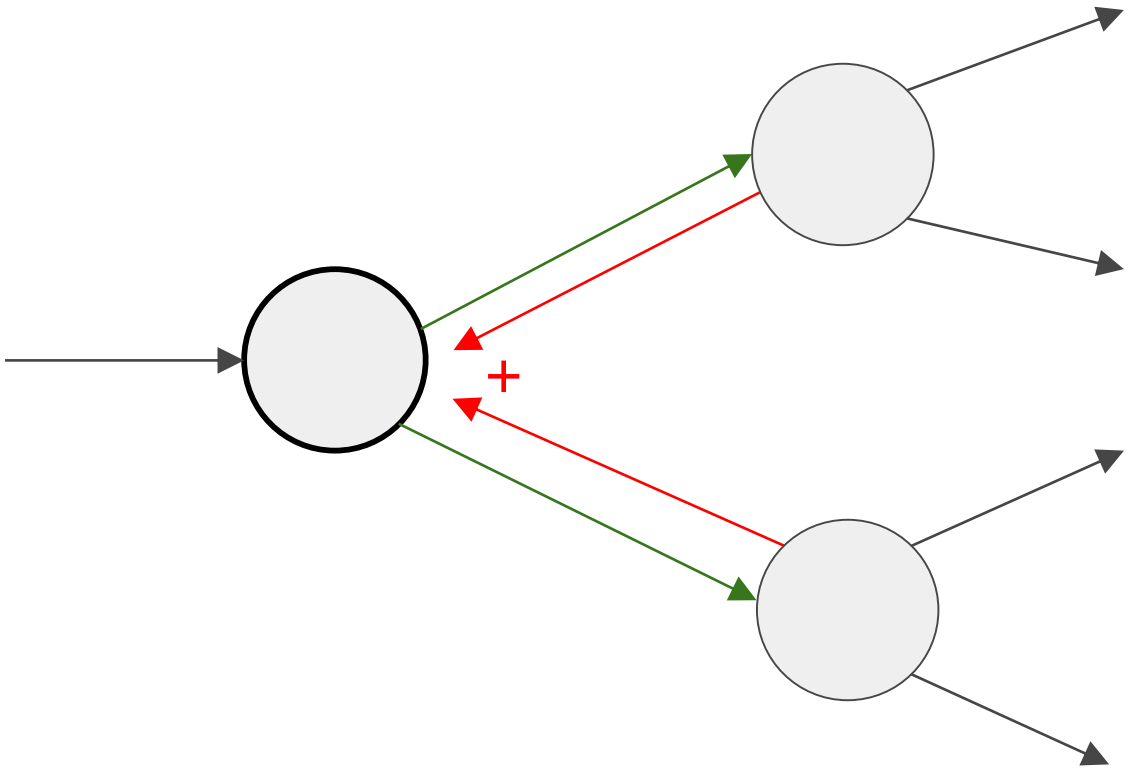
چند الگو در محاسبات رو به عقب

- عمل بیشینه‌یابی. مسیریابی گرادیان!

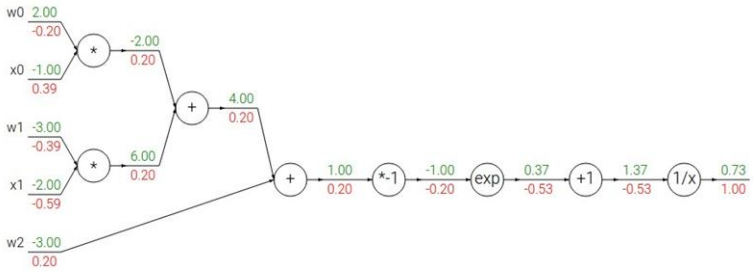


چند الگو در محاسبات رو به عقب

- جمع شدن گرادیان‌ها در انشعاب‌ها.

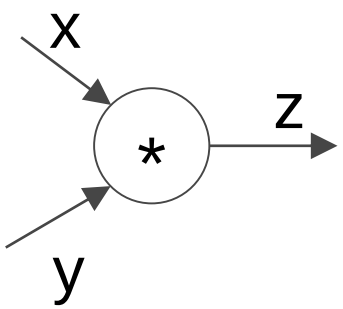


پیاده‌سازی: محاسبات رو به جلو و رو به عقب




```python
class ComputationalGraph(object):
    #...
    def forward(inputs):
        # 1. [pass inputs to input gates...]
        # 2. forward the computational graph:
        for gate in self.graph.nodes_topologically_sorted():
            gate.forward()
        return loss # the final gate in the graph outputs the loss
    def backward():
        for gate in reversed(self.graph.nodes_topologically_sorted()):
            gate.backward() # little piece of backprop (chain rule applied)
        return inputs_gradients
```

پیاده‌سازی: محاسبات رو به جلو و رو به عقب

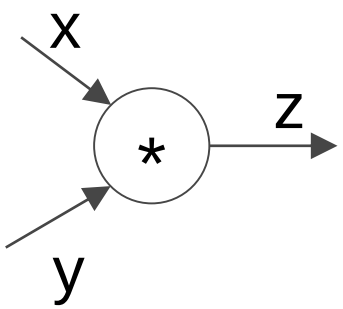



```python
class MultiplyGate(object):
    def forward(x,y):
        z = x*y
        return z
    def backward(dz):
        # dx = ... #todo
        # dy = ... #todo
        return [dx, dy]
```

$\frac{\partial L}{\partial z}$

$\frac{\partial L}{\partial x}$

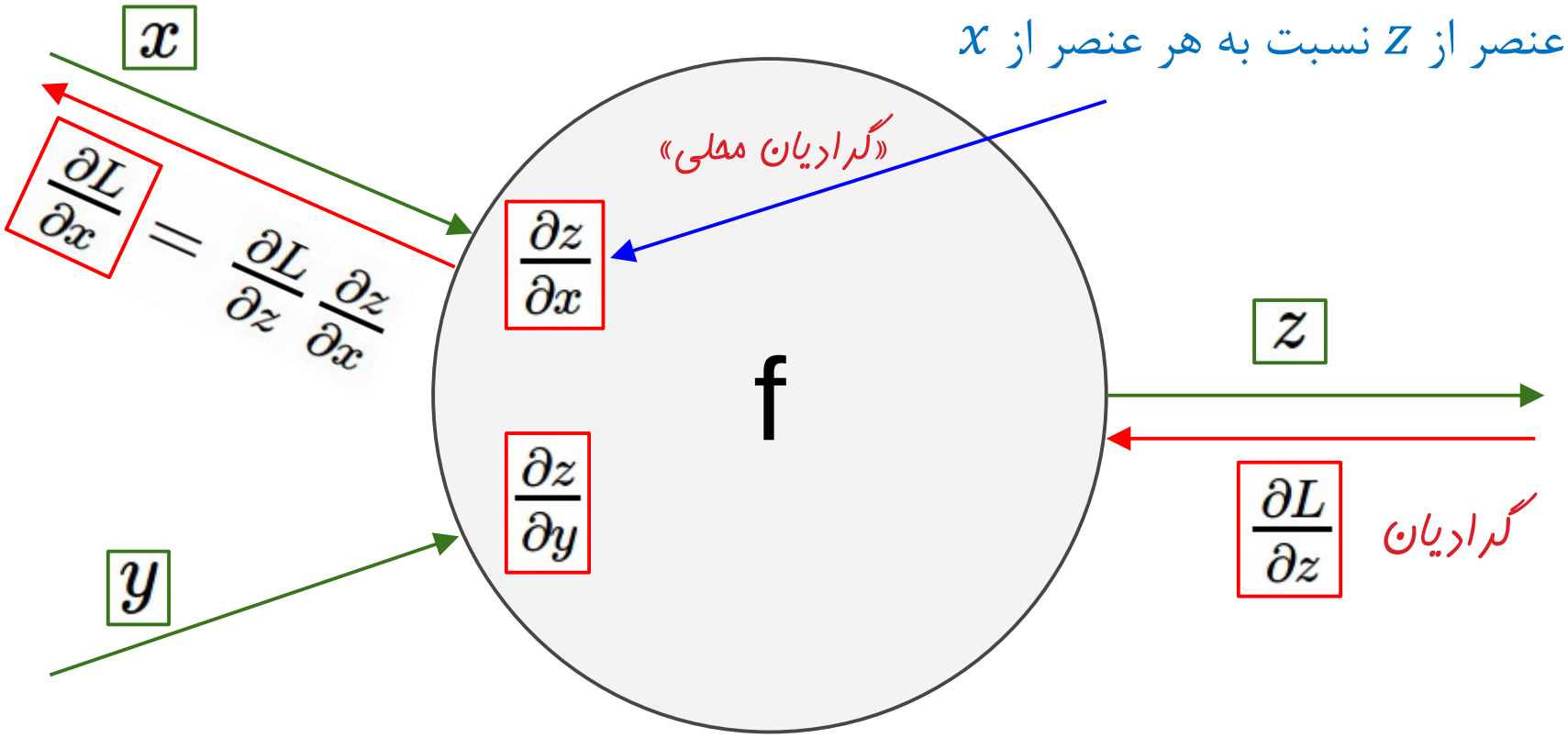
پیاده‌سازی: محاسبات رو به جلو و رو به عقب



```
class MultiplyGate(object):
    def forward(x,y):
        z = x*y
        self.x = x # must keep these around!
        self.y = y
        return z
    def backward(dz):
        dx = self.y * dz # [dz/dx * dL/dz]
        dy = self.x * dz # [dz/dy * dL/dz]
        return [dx, dy]
```

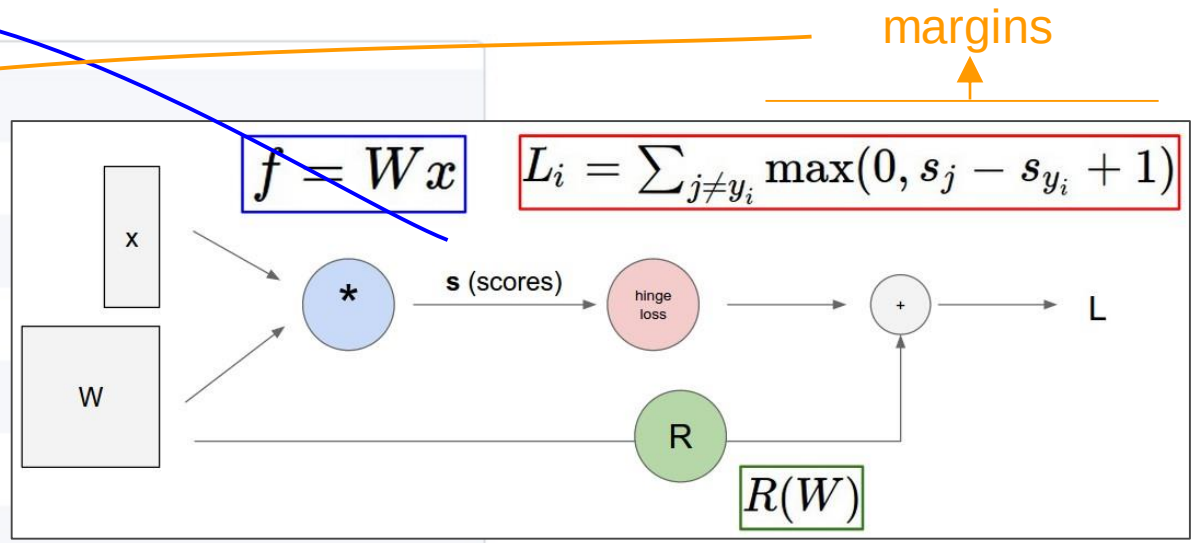
گرادیان برای بردارها

اکنون این گرادیان محلی یک **ماتریس ژاکوبی** است.
یعنی مشتق هر عنصر از z نسبت به هر عنصر از x



پیاده‌سازی توابع هزینه


```
# receive W (weights), X (data)
# forward pass (we have 8 lines)
scores = #...
margins = #...
data_loss = #...
reg_loss = #...
loss = data_loss + reg_loss
# backward pass (we have 5 lines)
dmargins = # ... (optionally, we go direct to dscores)
dscores = #...
dW = #...
```

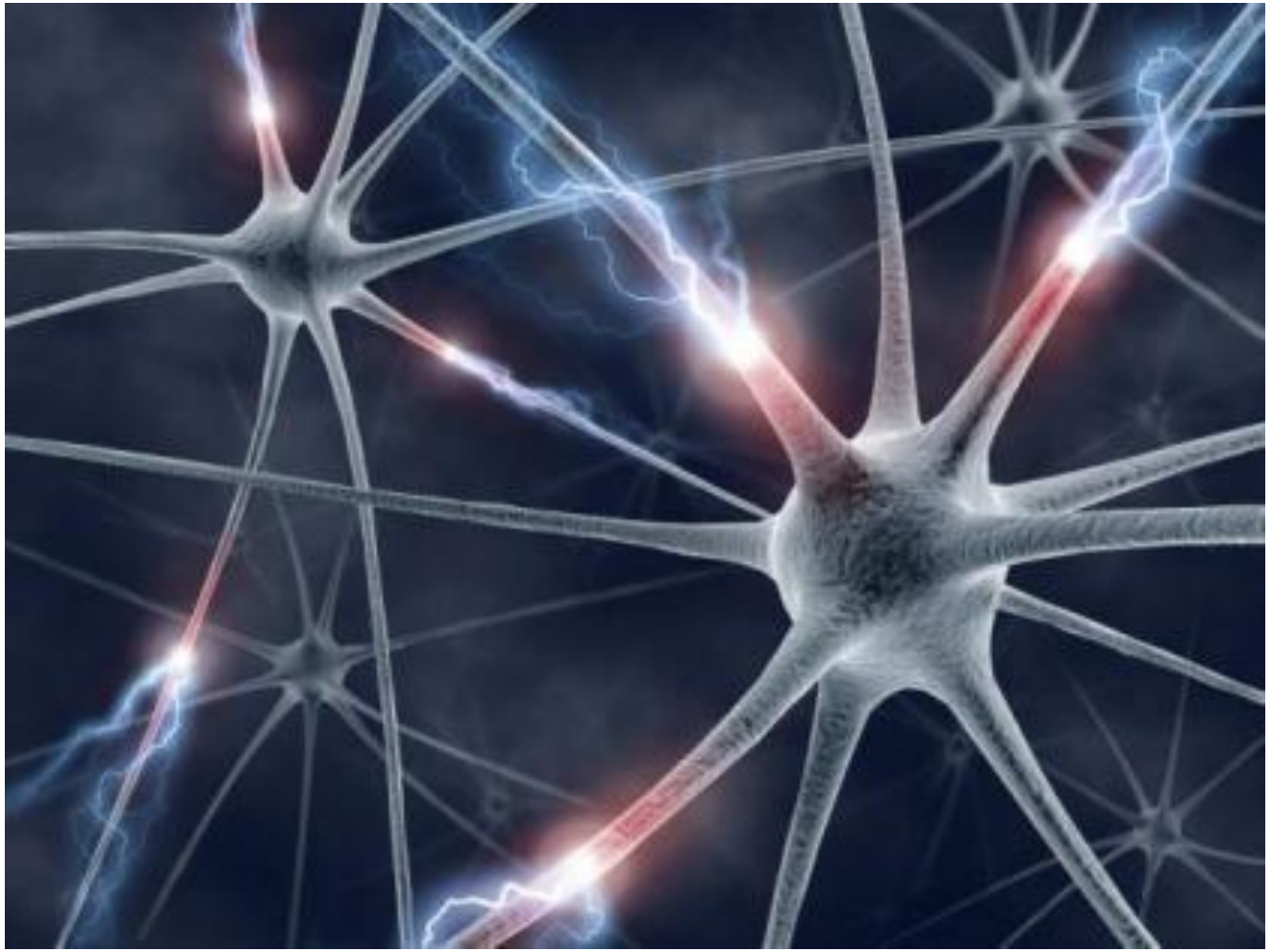


جمع‌بندی

- تعداد پارامترها در یک شبکه عصبی می‌تواند بسیار زیاد باشد:
  - نوشتن رابطه مربوط به گرادیان تمام پارامترها به صورت دستی غیر ممکن است!
- پس‌انتشار. به کار بردن قاعده زنجیری به صورت بازگشتی در طول یک گراف محاسباتی به منظور محاسبه گرادیان تابع هزینه نسبت به پارامترها، ورودی‌ها و مقادیر میانی.
- گراف محاسباتی. یک ساختار گرافی که هر گره آن محاسبات رو به جلو و محاسبات رو به عقب را پیاده‌سازی می‌کند.
  - **محاسبات رو به جلو.** محاسبه‌ی نتیجه یک عمل و ذخیره مقادیر میانی مورد نیاز برای محاسبه گرادیان.
  - **محاسبات رو به عقب.** استفاده از قاعده زنجیری به منظور محاسبه گرادیان تابع هزینه نسبت به ورودی‌ها.

شبکه‌های عصبی

شبکه‌های عصبی



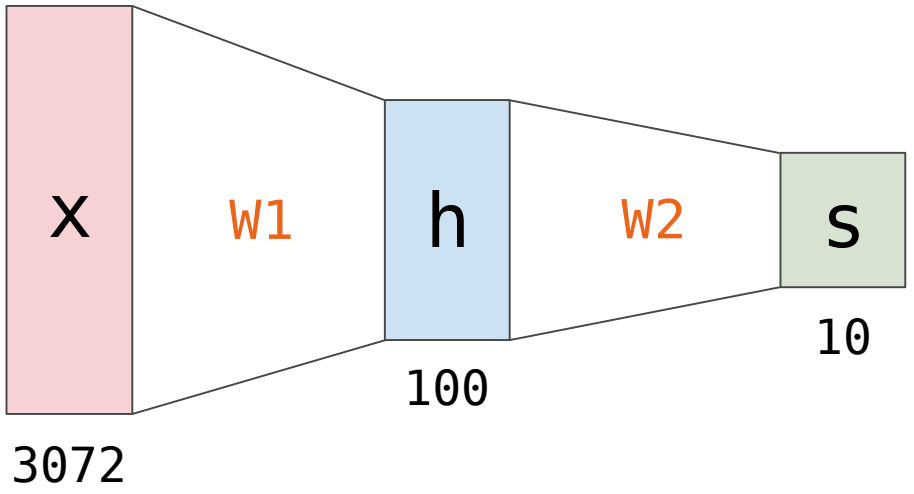
شبکه‌های عصبی

(**قبلاً**) تابع امتیاز خطی:

$$f = Wx$$

(**اکنون**) شبکه عصبی ۲ لایه:

$$f = W_2 \max(0, W_1 x)$$



شبکه‌های عصبی

(**قبلاً**) تابع امتیاز خطی:

$$f = Wx$$

(**اکنون**) شبکه عصبی ۲ لایه:

$$f = W_2 \max(0, W_1 x)$$

یا شبکه عصبی ۳ لایه:

$$f = W_3 \max(0, W_2 \max(0, W_1 x))$$

شبکه‌های عصبی

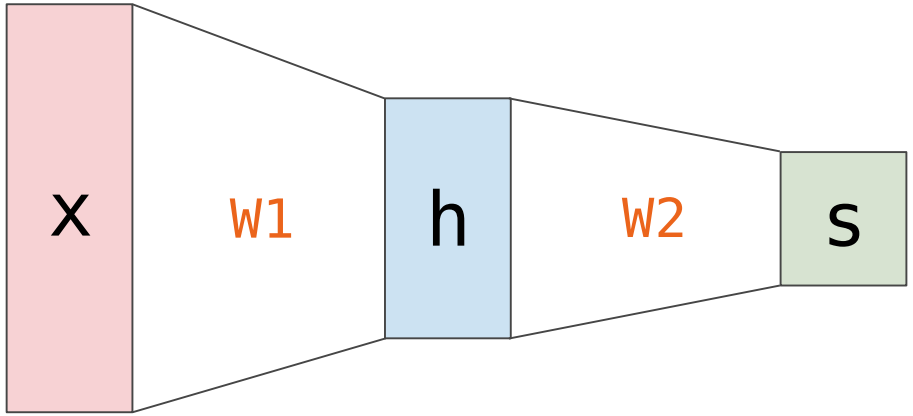
- پیاده‌سازی آموزش یک شبکه عصبی ۲ لایه. [در ۱۱ خط]

```
01. X = np.array([ [0,0,1],[0,1,1],[1,0,1],[1,1,1] ])
02. y = np.array([[0,1,1,0]]).T
03. syn0 = 2*np.random.random((3,4)) - 1
04. syn1 = 2*np.random.random((4,1)) - 1
05. for j in xrange(60000):
06.     l1 = 1/(1+np.exp(-(np.dot(X,syn0))))
07.     l2 = 1/(1+np.exp(-(np.dot(l1,syn1))))
08.     l2_delta = (y - l2)*(l2*(1-l2))
09.     l1_delta = l2_delta.dot(syn1.T) * (l1 * (1-l1))
10.     syn1 += l1.T.dot(l2_delta)
11.     syn0 += X.T.dot(l1_delta)
```

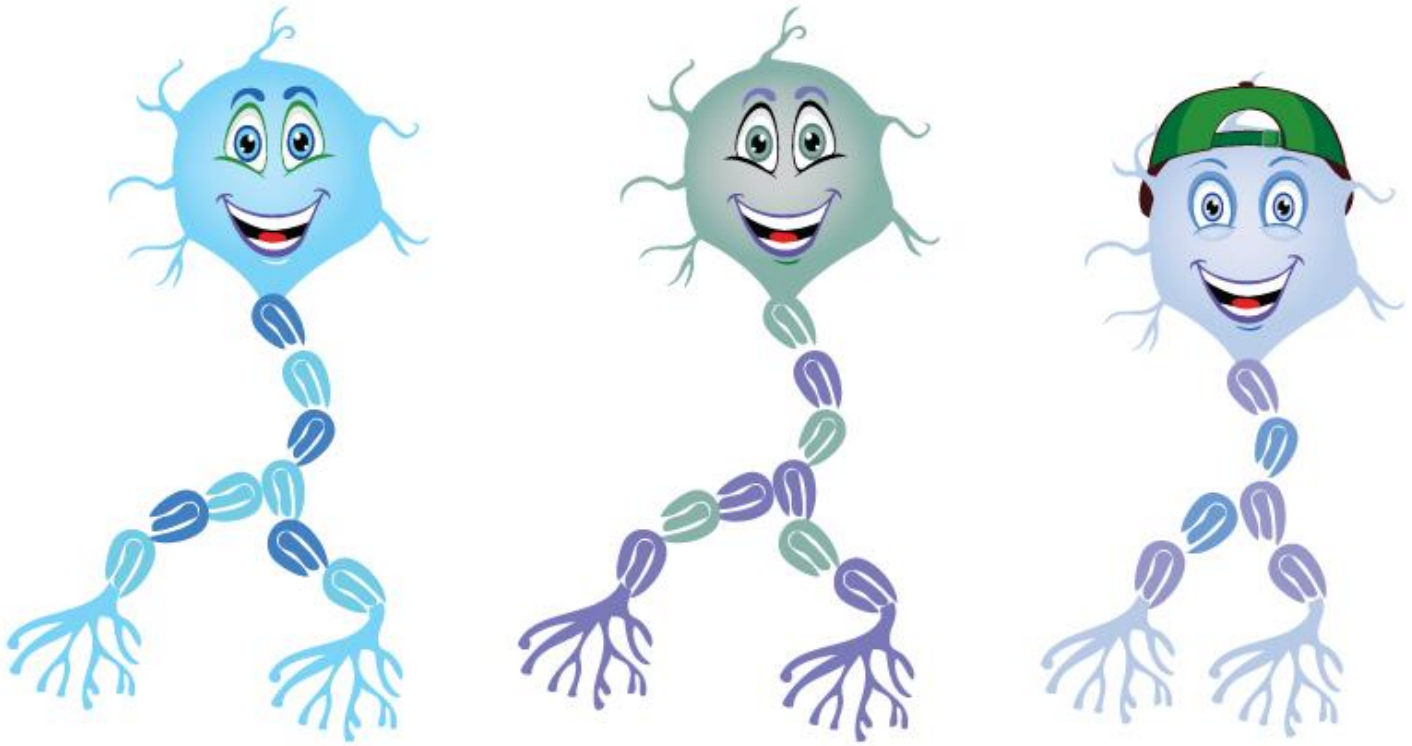
from @iamtrask, http://iamtrask.github.io/2015/07/12/basic-python-network/

پیاده‌سازی یک شبکه عصبی ۲ لایه

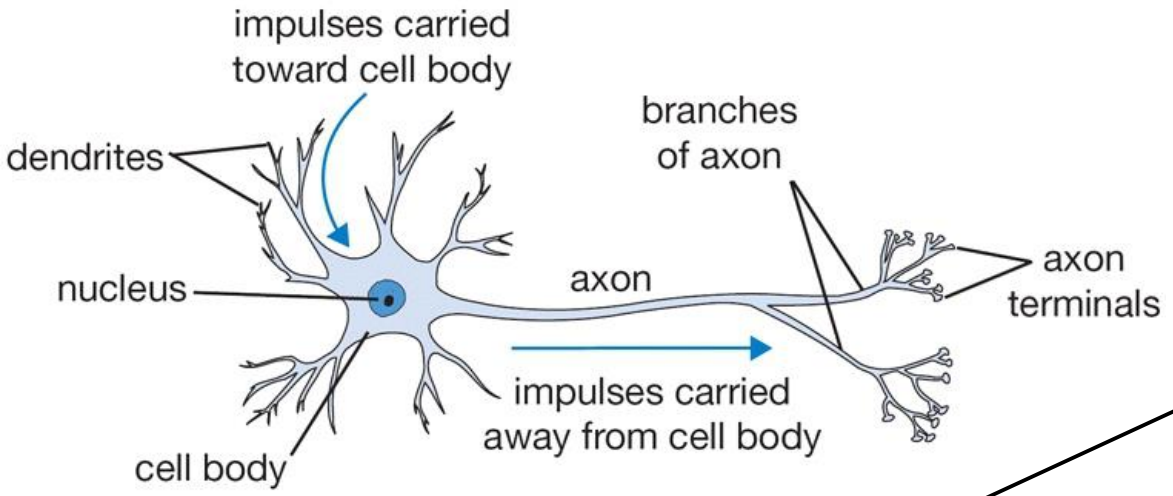

```
# receive W1, W2, b1, b2 (weights/biases), X (data)

# forward pass:
h =          #... function of X, W1, b1
scores =     #... function of h, W2, b2
loss =       #... (several lines of code to evaluate Softmax loss)

# backward pass:
dscores =           #...
dh, dW2, db2 =      #...
dW1, db1 =          #...
```

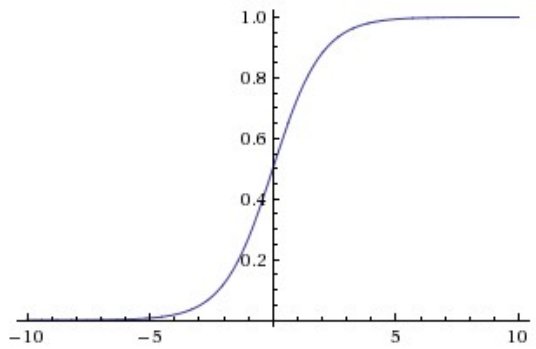


نورون‌ها و شبکه‌های عصبی



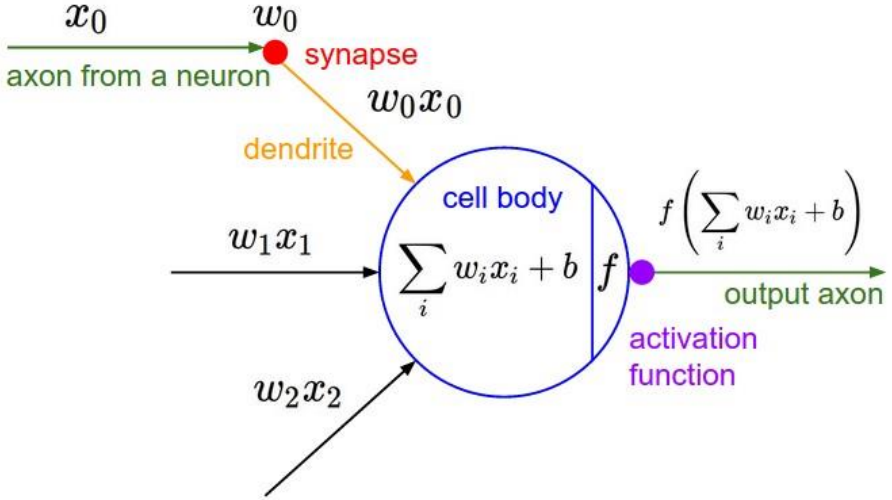
نورون‌ها و شبکه‌های عصبی



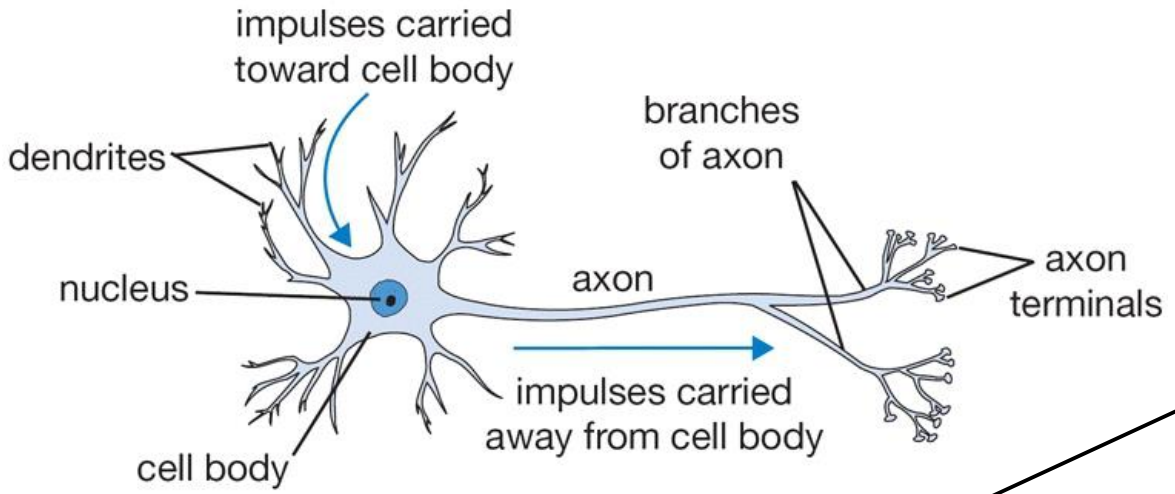


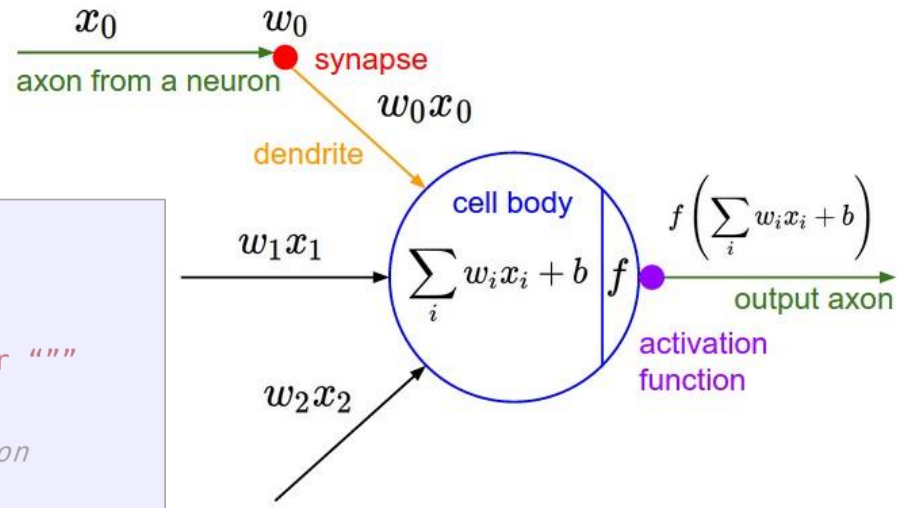
$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

تابع فعالیت سیگموید



نورون‌ها و شبکه‌های عصبی

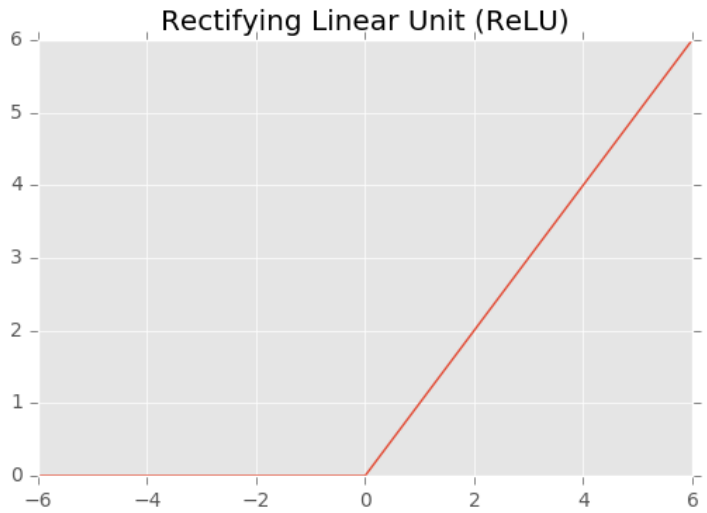





```python
class Neuron:
    #...
    def neuron_tick(inputs):
        """ assume inputs and weights are 1-D numpy arrays and bias is a number """
        cell_body_sum = np.sum(inputs * self.weights) + self.bias
        firing_rate = 1.0 / (1.0 + math.exp(-cell_body_sum))  # Sigmoid function
        return firing_rate
```

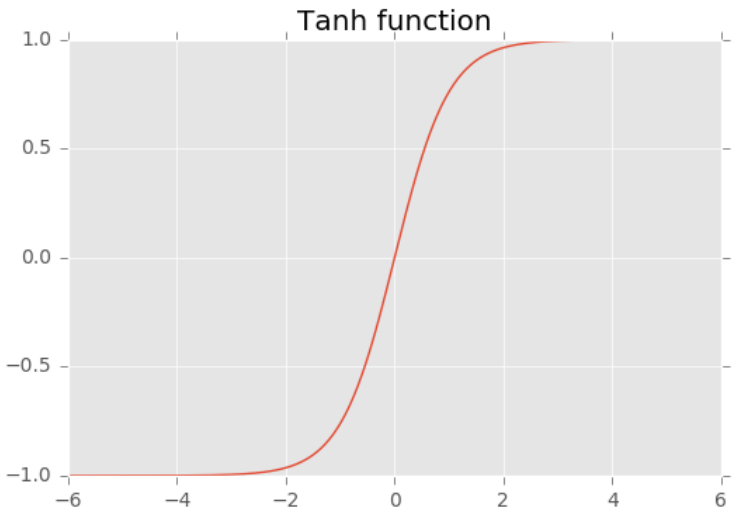
توابع فعالیت

ReLU



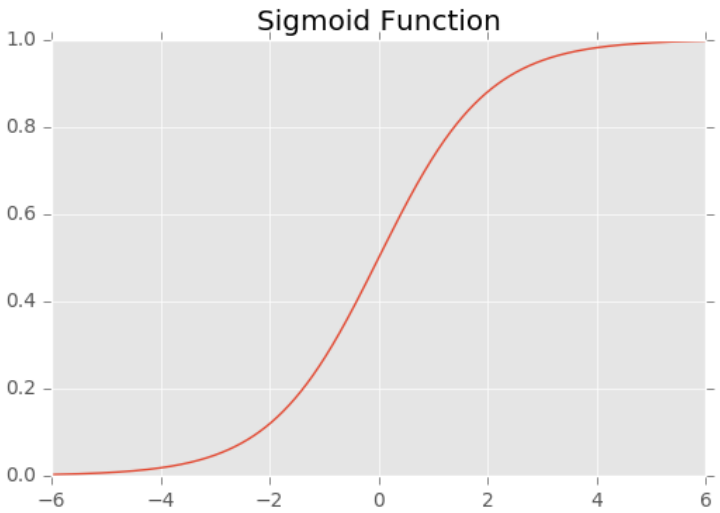
$$\max(0, x)$$

تانژانت هایپربولیک



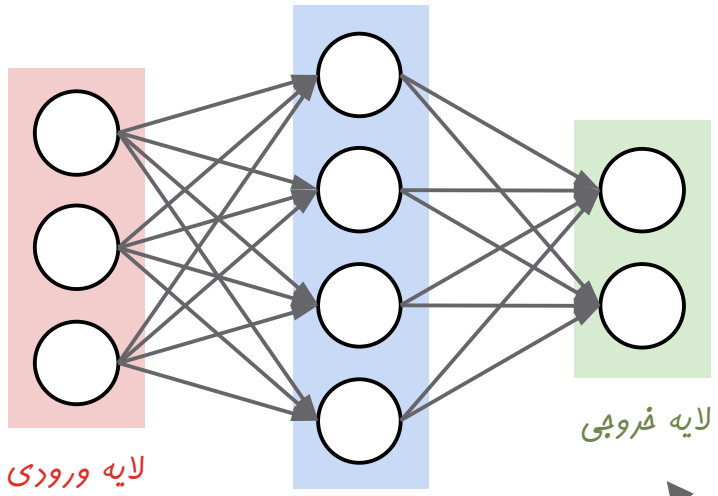
$$\tanh(x)$$

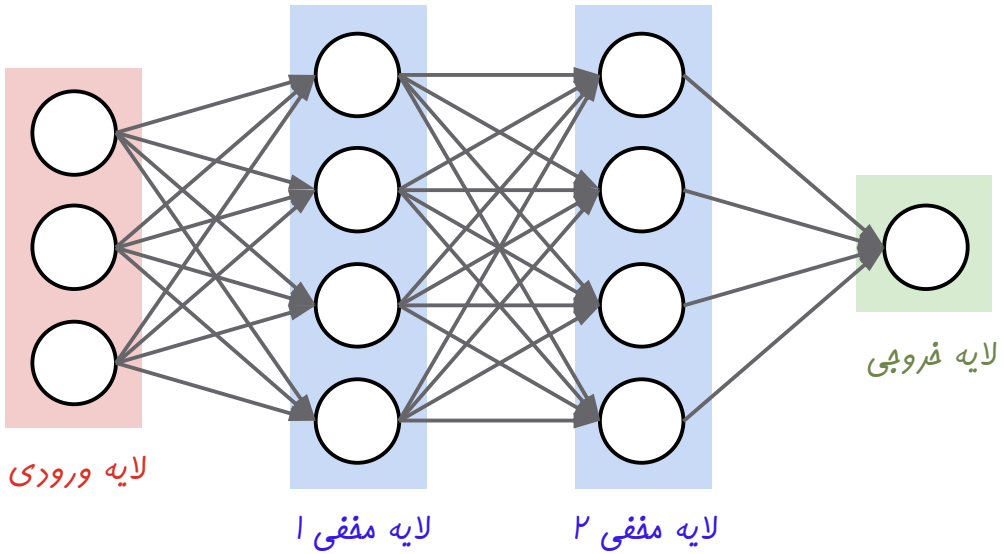
سیگموید



$$\sigma(x) = 1/(1 + e^{-x})$$

شبکه‌های عصبی: معماری





لایه‌های «کاملاً متصل»

شبکه عصبی ۲ لایه

[شبکه عصبی با ۱ لایه‌ی مخفی]

شبکه عصبی ۳ لایه

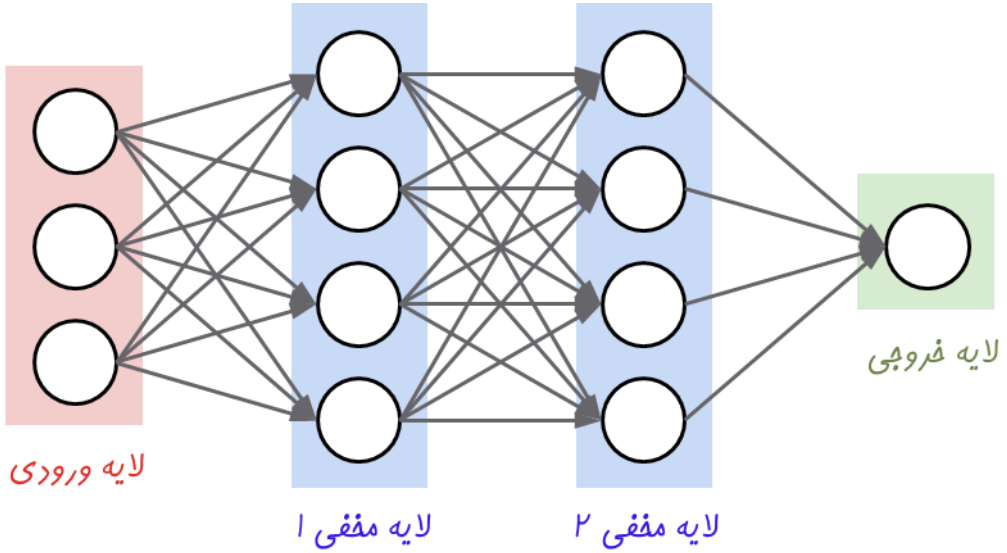
[شبکه عصبی با ۲ لایه‌ی مخفی]

محاسبات روبه‌جلو در یک شبکه عصبی


```
class Neuron:

    #...

    def neuron_tick(inputs):
        """ assume inputs and weights are 1-D numpy arrays and bias is a number """
        cell_body_sum = np.sum(inputs * self.weights) + self.bias
        firing_rate = 1.0 / (1.0 + math.exp(-cell_body_sum))  # Sigmoid function
        return firing_rate
```

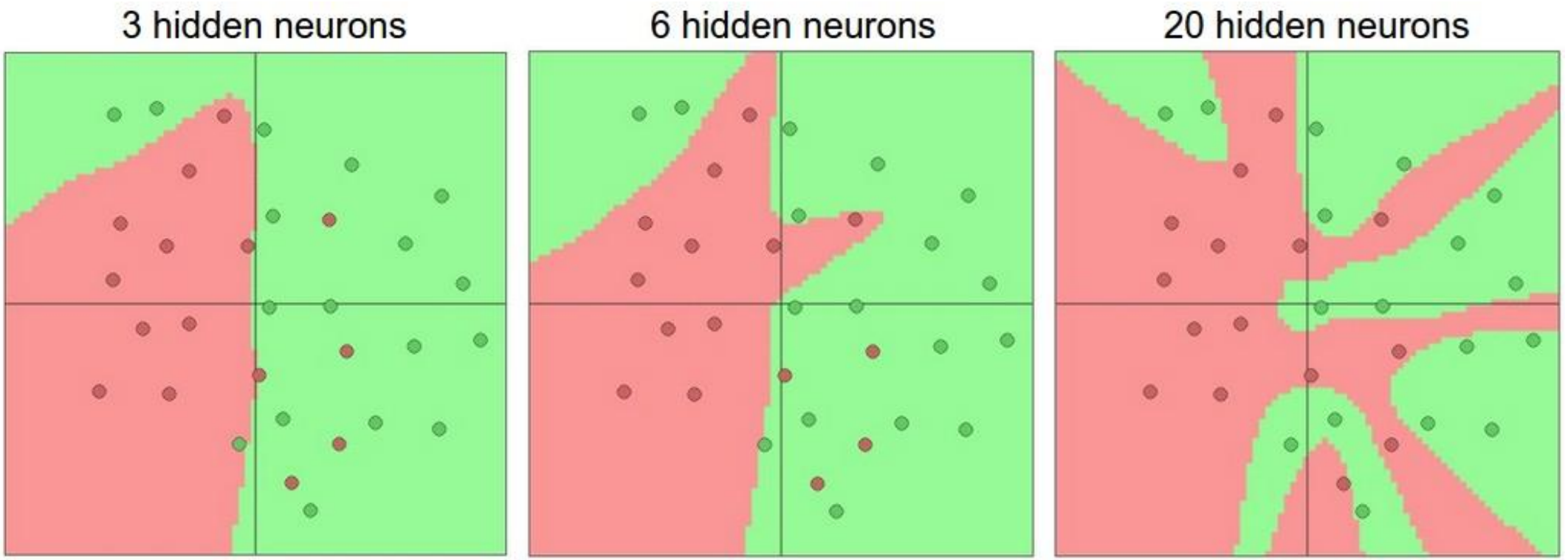
- توجه. می‌توان محاسبات یک لایه کامل از نورون‌ها را به صورت کارآ پیاده‌سازی نمود. [پیاده‌سازی برداری]

محاسبات روبه‌جلو در یک شبکه عصبی



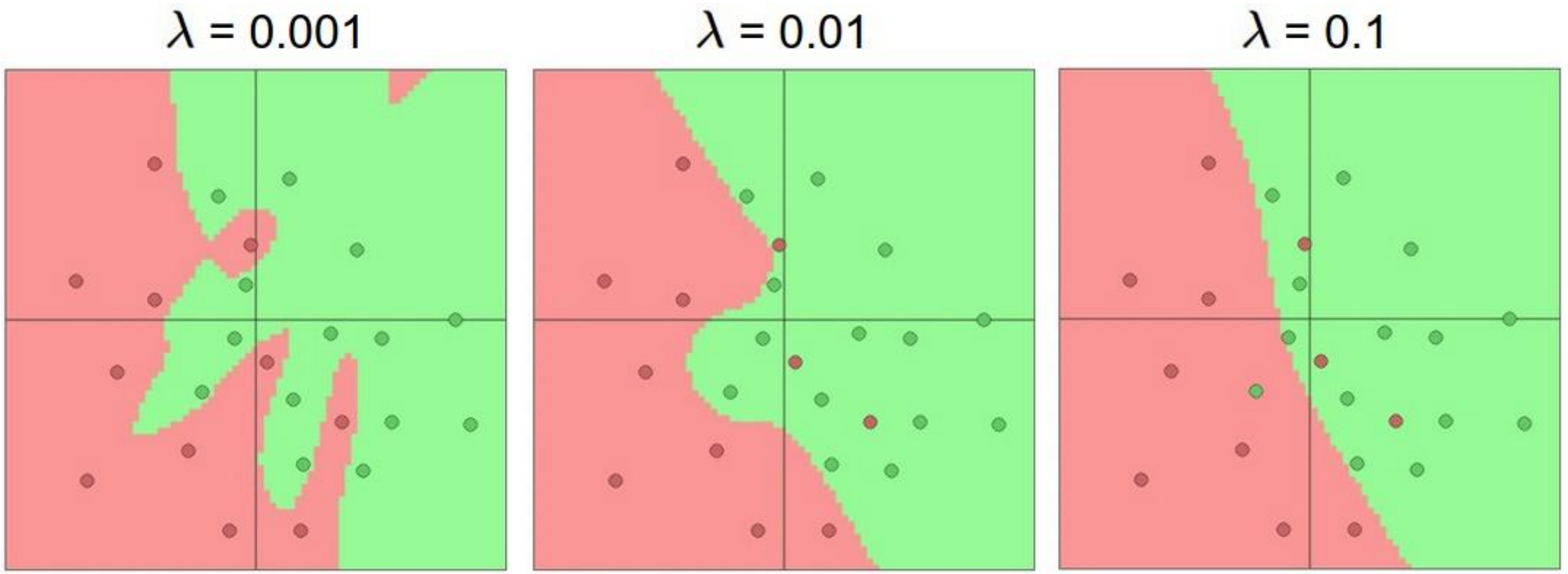

```
# forward pass of a 3-layer neural network:
f = lambda x: 1.0 / (1.0 + np.exp(-x))  # activation function (use sigmoid)
x = np.random.randn(3, 1)      # random input vector of three numbers (3x1)
h1 = f(np.dot(W1, x ) + b1)    # calculate first hidden layer activations (4x1)
h2 = f(np.dot(W2, h1) + b2)    # calculate second hidden layer activations (4x1)
out =  np.dot(W3, h2) + b3     # output neuron (1x1)
```

تعیین تعداد و اندازه لایه‌ها



نورون‌های بیشتر = ظرفیت بیشتر

تعیین تعداد و اندازه لایه‌ها



- از اندازه شبکه عصبی به عنوان تنظیم کننده استفاده نکنید و به جای آن از یک **روش قوی‌تر** استفاده کنید.

تنظیم L2

جمع‌بندی

- در یک شبکه عصبی، نورون‌ها را در **لایه‌های کاملاً متصل** قرار می‌دهیم.
- استفاده از مفهوم **لایه** به ما امکان می‌دهد از پیاده‌سازی کارا به صورت برداری استفاده کنیم.
  [مانند ضرب ماتریسی]
- نورون‌های عصبی مصنوعی، مدل‌های بسیار ساده شده‌ای از نورون‌های زیستی هستند.
  - در واقع، شبکه‌های عصبی اصلاً عصبی نیستند!
- هر چه اندازه یک شبکه عصبی بزرگ‌تر باشد بهتر است:
  - به شرطی که از یک تنظیم کننده قوی برای تنظیم وزن‌ها استفاده شود.

شبکه‌های عصبی: آموزش

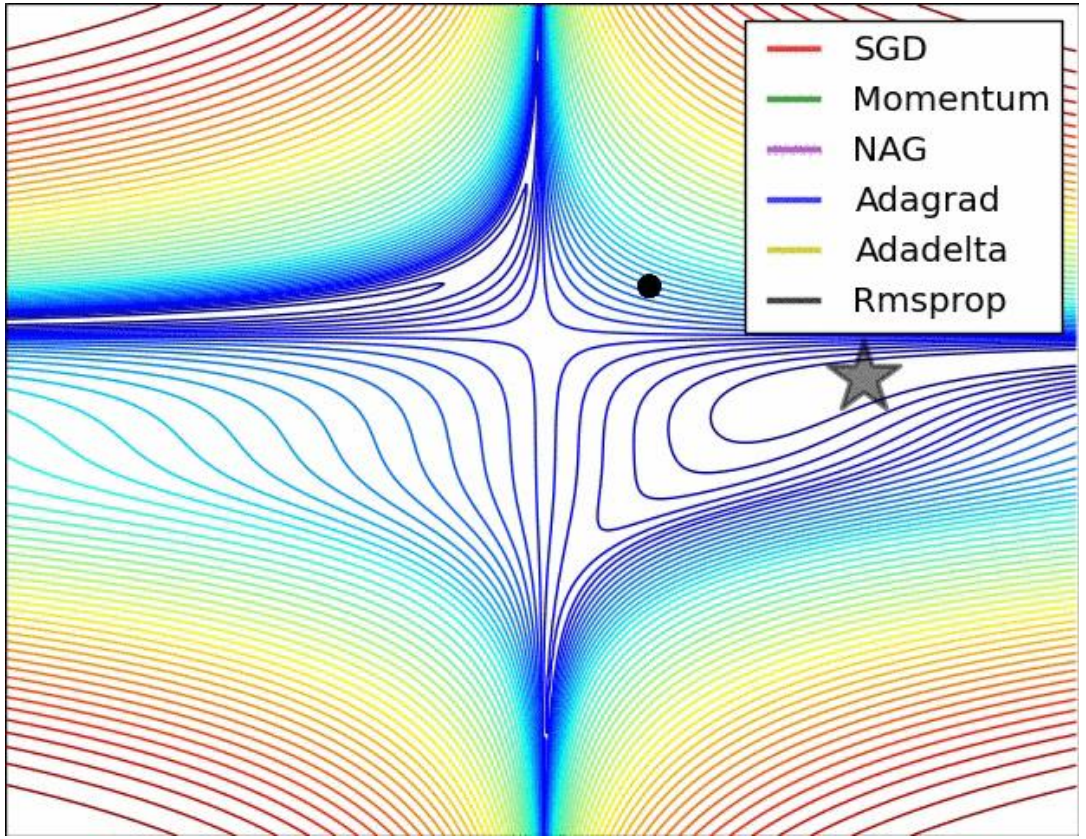
یادآوری: بهینه‌سازی

- گرادیان کاهشی. [با دسته‌های کوچک]

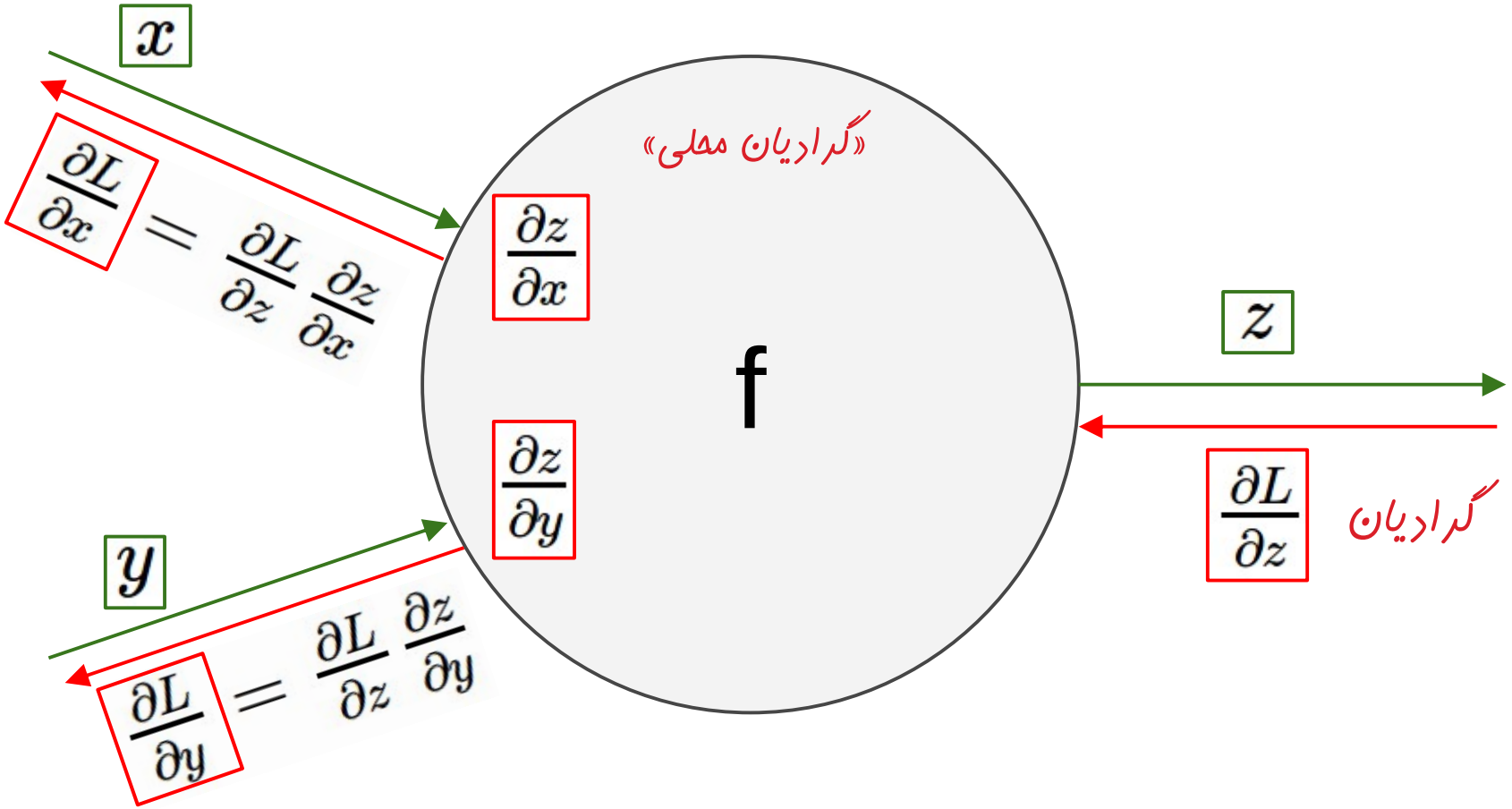
حلقه:

۱. یک دسته از داده‌ها را به طور تصادفی **انتخاب** کن.
۲. **محاسبات رو به جلو** را در طول گراف انجام بده و مقدار تابع هزینه را محاسبه کن.
۳. **محاسبات رو به عقب** را به منظور محاسبه گرادیان‌ها انجام بده.
۴. مقدار پارامترها را با استفاده از گرادیان‌های محاسبه شده **به روز رسانی** کن.

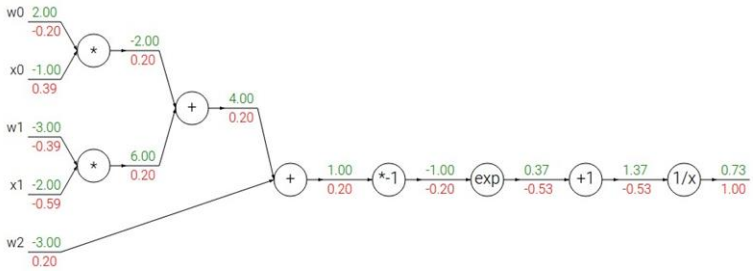
یادآوری: بهینه‌سازی



یادآوری: گراف محاسباتی

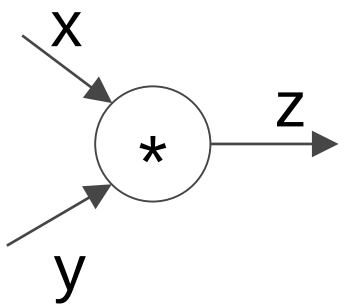


یادآوری: گراف محاسباتی




```python
class ComputationalGraph(object):
    #...
    def forward(inputs):
        # 1. [pass inputs to input gates...]
        # 2. forward the computational graph:
        for gate in self.graph.nodes_topologically_sorted():
            gate.forward()
        return loss # the final gate in the graph outputs the loss
    def backward():
        for gate in reversed(self.graph.nodes_topologically_sorted()):
            gate.backward() # little piece of backprop (chain rule applied)
        return inputs_gradients
```

یادآوری: گراف محاسباتی



```
class MultiplyGate(object):
    def forward(x,y):
        z = x*y
        self.x = x # must keep these around!
        self.y = y
        return z
    def backward(dz):
        dx = self.y * dz # [dz/dx * dL/dz]
        dy = self.x * dz # [dz/dy * dL/dz]
        return [dx, dy]
```

یادآوری: شبکه‌های عصبی

(**قبلاً**) تابع امتیاز خطی:

$$f = Wx$$

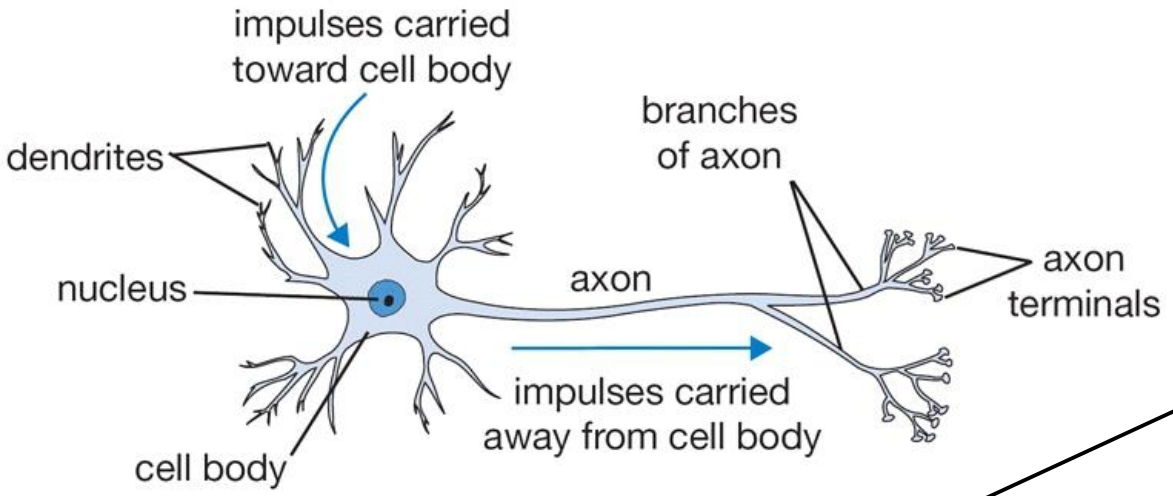
(**اکنون**) شبکه عصبی ۲ لایه:

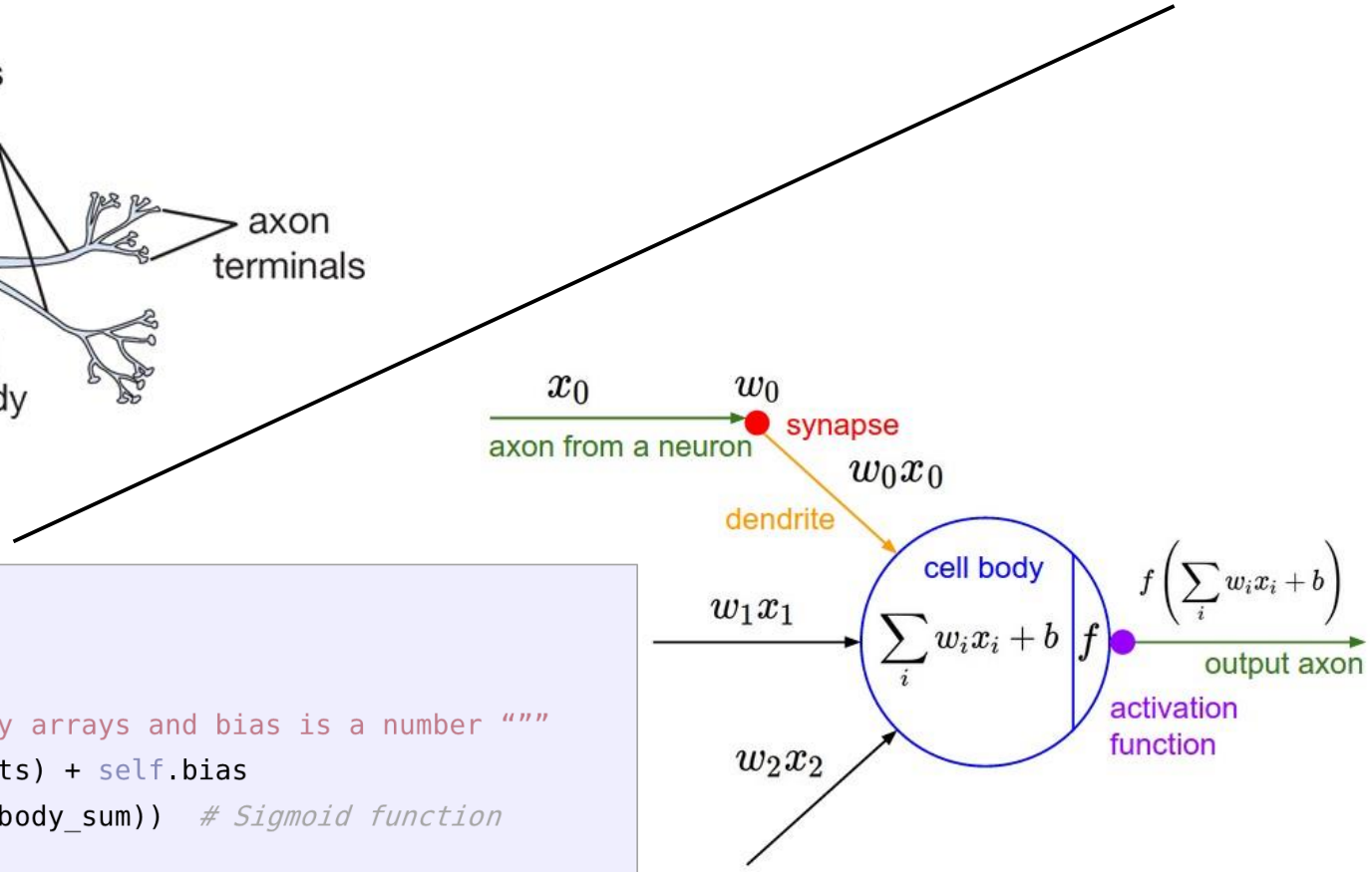
$$f = W_2 \max(0, W_1 x)$$

یا شبکه عصبی ۳ لایه:

$$f = W_3 \max(0, W_2 \max(0, W_1 x))$$

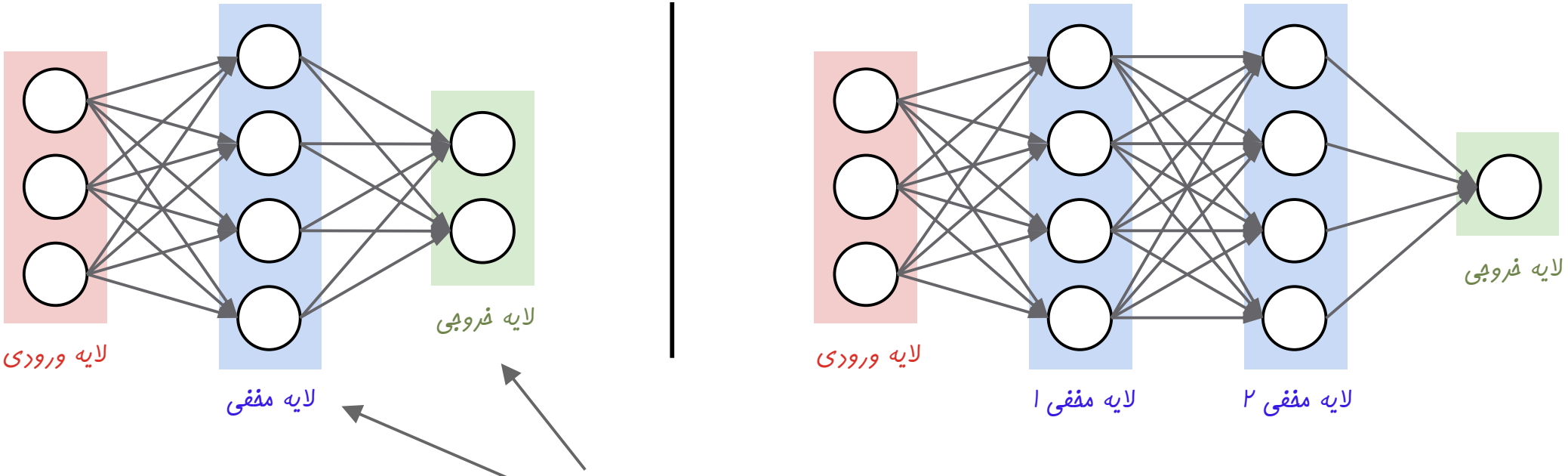
یادآوری: نورون‌ها و شبکه‌های عصبی






```python
class Neuron:
    #...
    def neuron_tick(inputs):
        """ assume inputs and weights are 1-D numpy arrays and bias is a number """
        cell_body_sum = np.sum(inputs * self.weights) + self.bias
        firing_rate = 1.0 / (1.0 + math.exp(-cell_body_sum))  # Sigmoid function
        return firing_rate
```

یادآوری: معماری شبکه‌های عصبی



لایه‌های «کاملاً متصل»

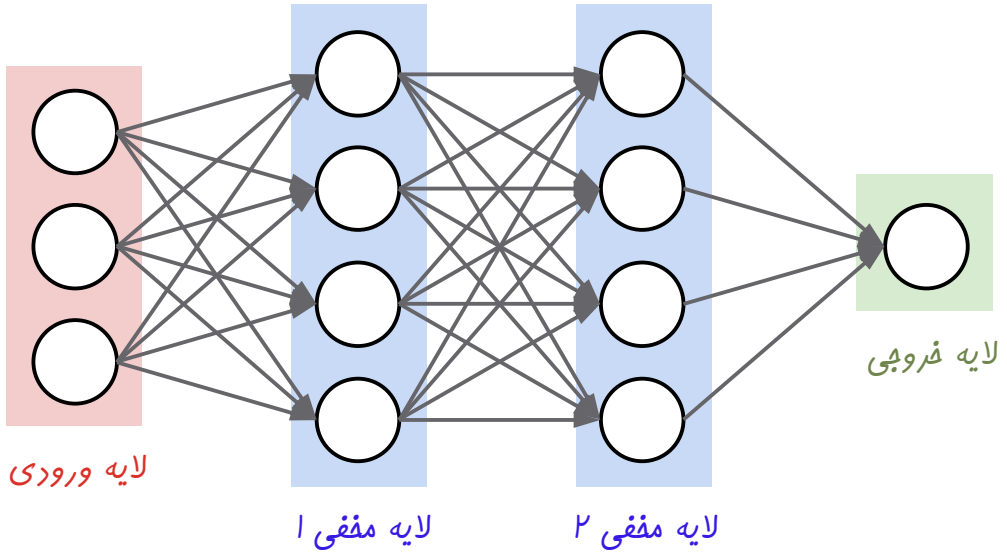
شبکه عصبی ۳ لایه

[شبکه عصبی با ۲ لایه‌ی مخفی]

شبکه عصبی ۲ لایه

[شبکه عصبی با ۱ لایه‌ی مخفی]

یادآوری: محاسبات روبه‌جلو در یک شبکه عصبی




```
# forward pass of a 3-layer neural network:
f = lambda x: 1.0 / (1.0 + np.exp(-x))  # activation function (use sigmoid)
x = np.random.randn(3, 1)      # random input vector of three numbers (3x1)
h1 = f(np.dot(W1, x ) + b1)    # calculate first hidden layer activations (4x1)
h2 = f(np.dot(W2, h1) + b2)    # calculate second hidden layer activations (4x1)
out =  np.dot(W3, h2) + b3     # output neuron (1x1)
```

آموزش شبکه‌های عصبی: مرور

(۱) راه‌اندازی.

- توابع فعالیت، پیش‌پردازش، مقداردهی اولیه به وزن‌ها، تنظیم، بررسی گرادیان

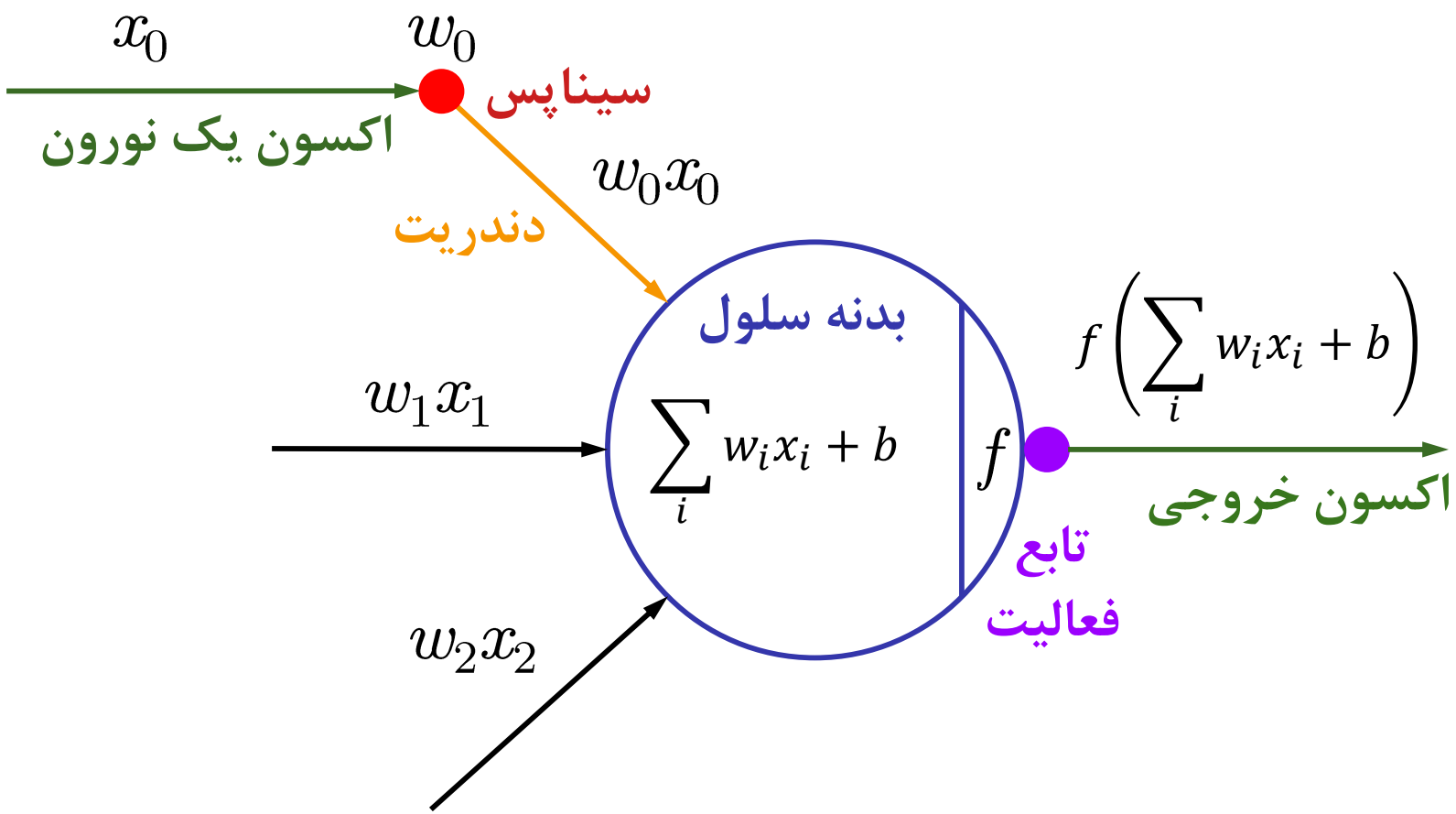
(۲) آموزش.

- مراحل توسعه فرایند یادگیری
- چگونگی به روز رسانی پارامترها، بهینه‌سازی ابرپارامترها

(۳) ارزیابی.

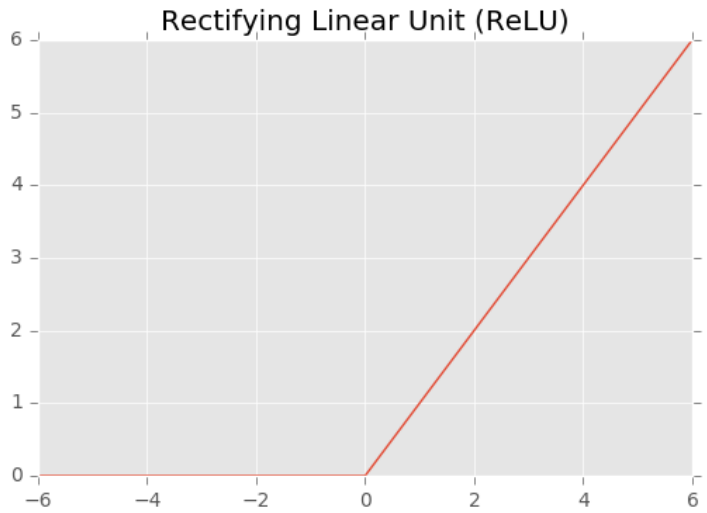
توابع فعالیت

توابع فعالیت



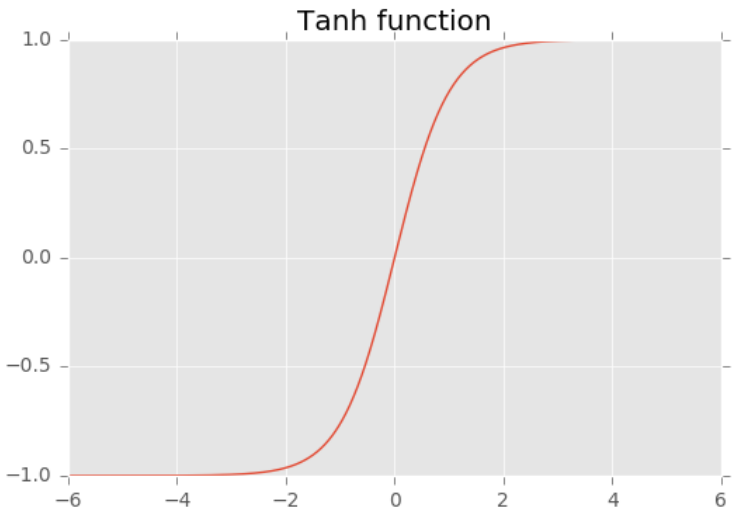
توابع فعالیت

ReLU



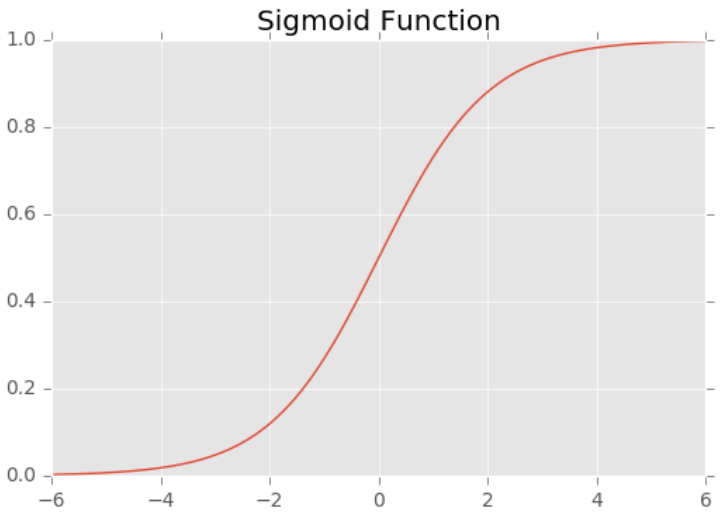
$$\max(0, x)$$

تانژانت هایپربولیک



$$\tanh(x)$$

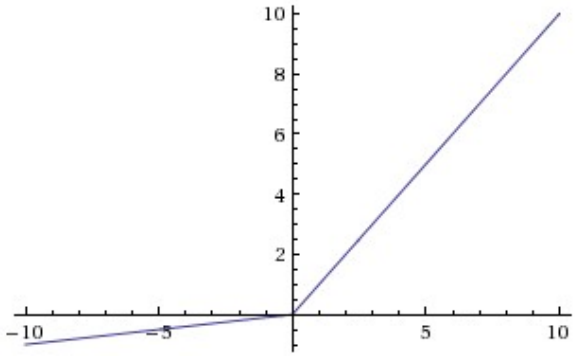
سیگموید



$$\sigma(x) = 1/(1 + e^{-x})$$

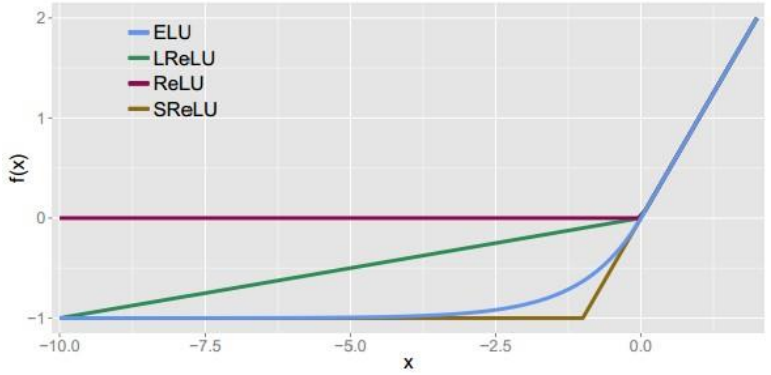
توابع فعالیت

Leaky ReLU



$$\max(0.1x, x)$$

ELU



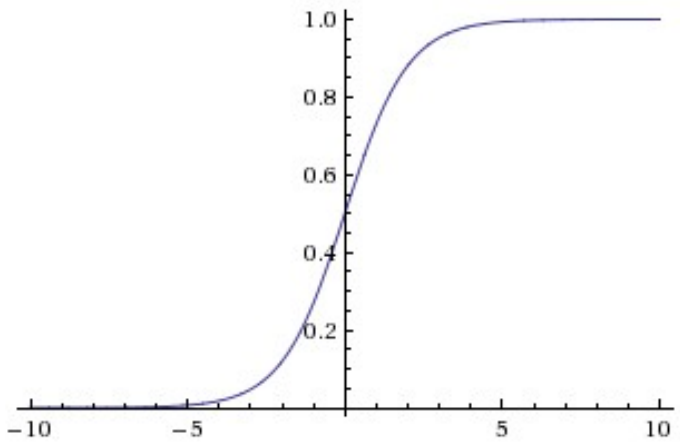
$$f(x) = \begin{cases} x, & x > 0 \\ \alpha(\exp(x) - 1), & x \geq 0 \end{cases}$$

توابع فعالیت

- تابع فعالیت سیگموید.
  - نگاشت اعداد به بازه‌ی [۰، ۱]
  - دارای محبوبیت تاریخی در تاریخچه‌ی شبکه‌های عصبی

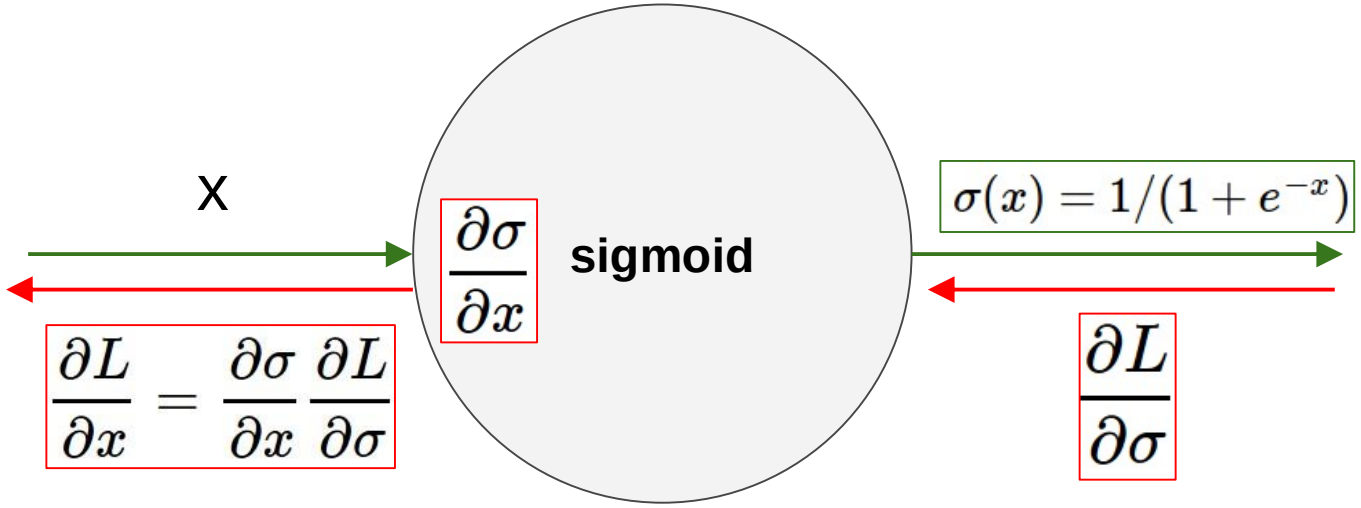
- مشکلات.
  ۱. نورون‌های اشباع شده، گرادیان را از بین می‌برند.
  ۲. خروجی‌های تابع سیگموید دارای میانگین صفر نیستند.
  ۳. توان رسانی عمل نسبتاً پر هزینه‌ای است.

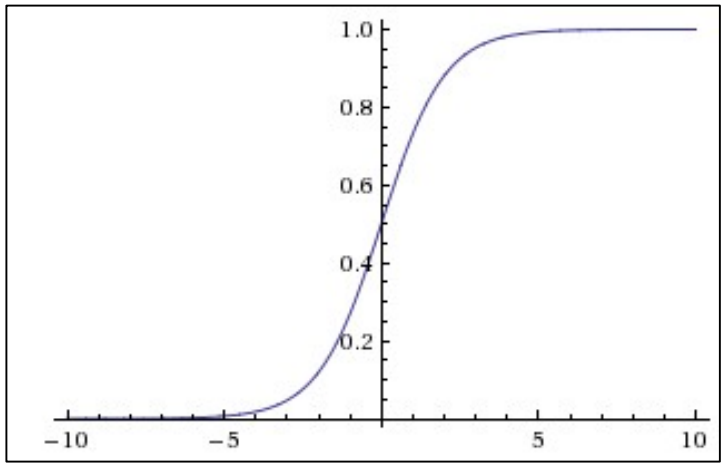
$$\sigma(x) = 1/(1 + e^{-x})$$



تابع سیگموید

توابع فعالیت





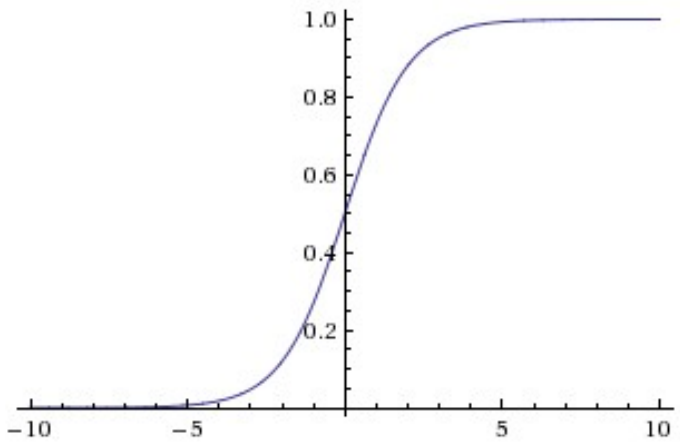
- چه اتفاقی می‌افتد اگر $x = -10$ ؟
- چه اتفاقی می‌افتد اگر $x = 0$ ؟
- چه اتفاقی می‌افتد اگر $x = +10$ ؟

توابع فعالیت

- تابع فعالیت سیگموید.
  - نگاشت اعداد به بازه‌ی [۰، ۱]
  - دارای محبوبیت تاریخی در تاریخچه‌ی شبکه‌های عصبی

- مشکلات.
  ۱. نورون‌های اشباع شده، گرادیان را از بین می‌برند.
  ۲. خروجی‌های تابع سیگموید دارای میانگین صفر نیستند.
  ۳. توان رسانی عمل نسبتاً پر هزینه‌ای است.

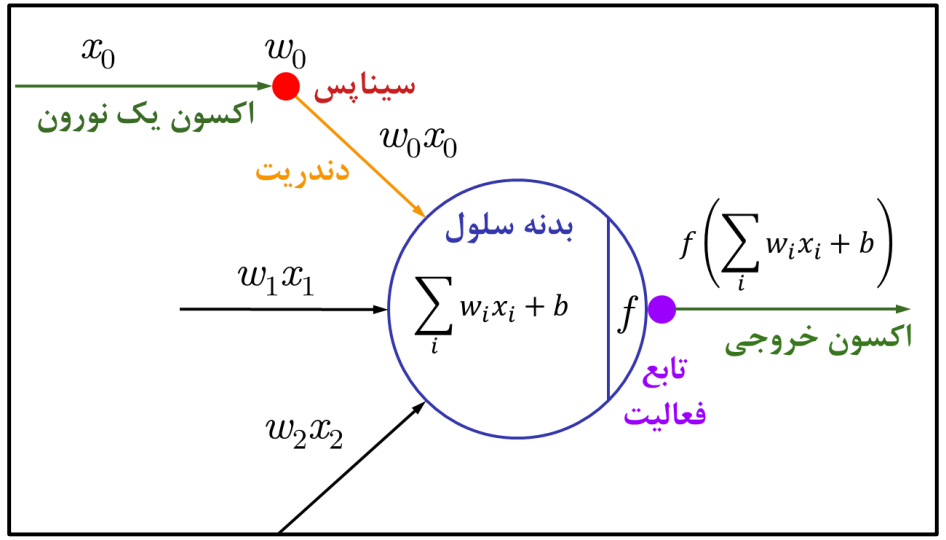
$$\sigma(x) = 1/(1 + e^{-x})$$



تابع سیگموید

توابع فعالیت

- اگر مقدار ورودی‌های یک نورون (x) همواره مثبت باشند، چه اتفاقی می‌افتد؟

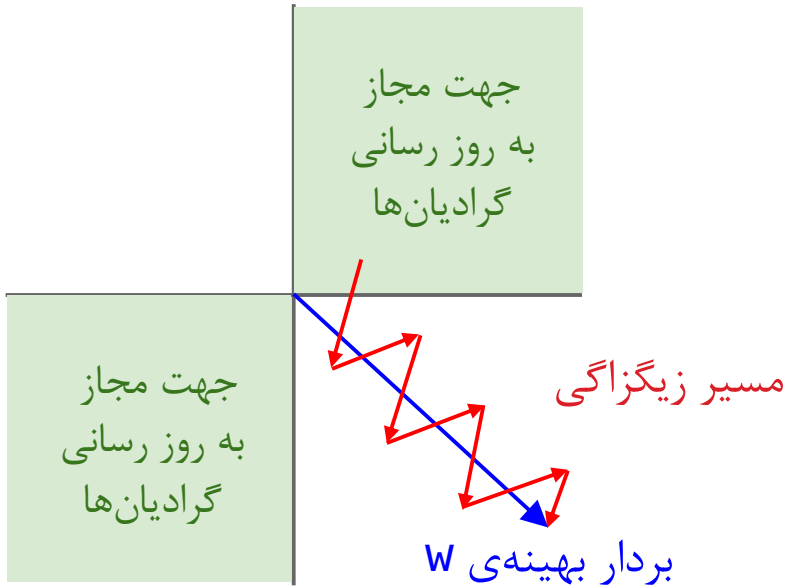


$$f\left(\sum_i w_i x_i + b\right)$$

- در مورد گرادیان‌ها نسبت به W چه می‌توان گفت؟

توابع فعالیت

- اگر مقدار ورودی‌های یک نورون (x) همواره مثبت باشند، چه اتفاقی می‌افتد؟



$$f\left(\sum_i w_i x_i + b\right)$$

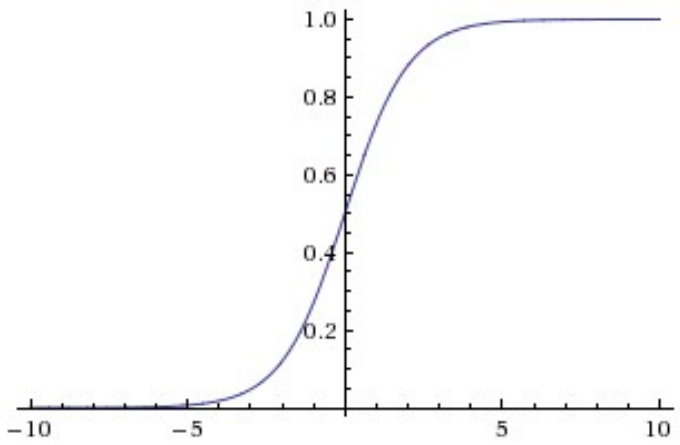
- در مورد گرادیان‌ها نسبت به W چه می‌توان گفت؟
  - همگی مثبت یا همگی منفی خواهند بود!
  - به همین دلیل می‌خواهیم داده‌ها نیز دارای میانگین صفر باشند.

توابع فعالیت

- تابع فعالیت سیگموید.
  - نگاشت اعداد به بازه‌ی [۰، ۱]
  - دارای محبوبیت تاریخی در تاریخچه‌ی شبکه‌های عصبی

- مشکلات.
  ۱. نورون‌های اشباع شده، گرادیان را از بین می‌برند.
  ۲. خروجی‌های تابع سیگموید دارای میانگین صفر نیستند.
  ۳. توان‌رسانی عمل نسبتاً پر هزینه‌ای است.

$$\sigma(x) = 1/(1 + e^{-x})$$

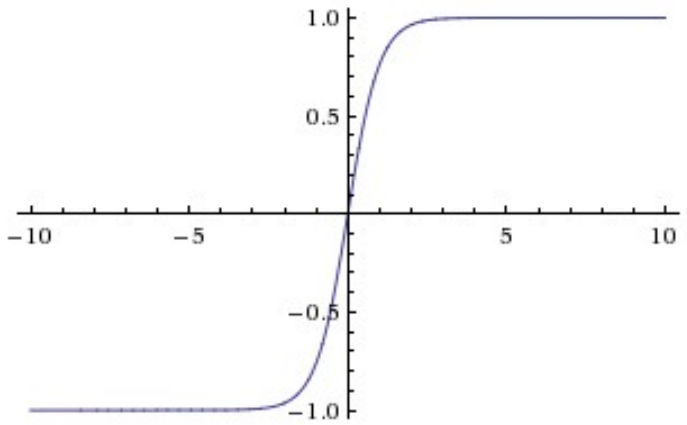


تابع سیگموید

توابع فعالیت

- تابع فعالیت تانژانت هایپربولیک.

$$\tanh(x)$$



تانژانت هایپربولیک

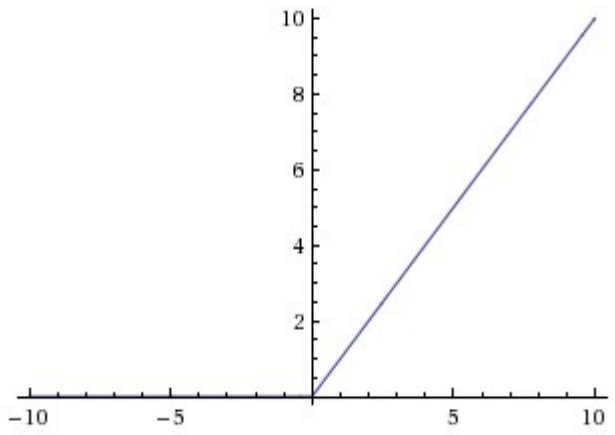
- نگاشت اعداد به بازه‌ی [۱ ،۱-]
- دارای میانگین صفر
- هنوز نورون‌های اشباع شده، گرادیان را از بین می‌برند.

[LeCun et al., 1991]

توابع فعالیت

- اشباع نمی‌شود (در نواحی مثبت)
- از نظر محاسباتی بسیار کارا است.
- در عمل، بسیار سریع‌تر از توابع سیگموید و تانژانت هایپربولیک همگرا می‌شود. (مثلاً ۶ برابر)
- دارای میانگین صفر نیست!

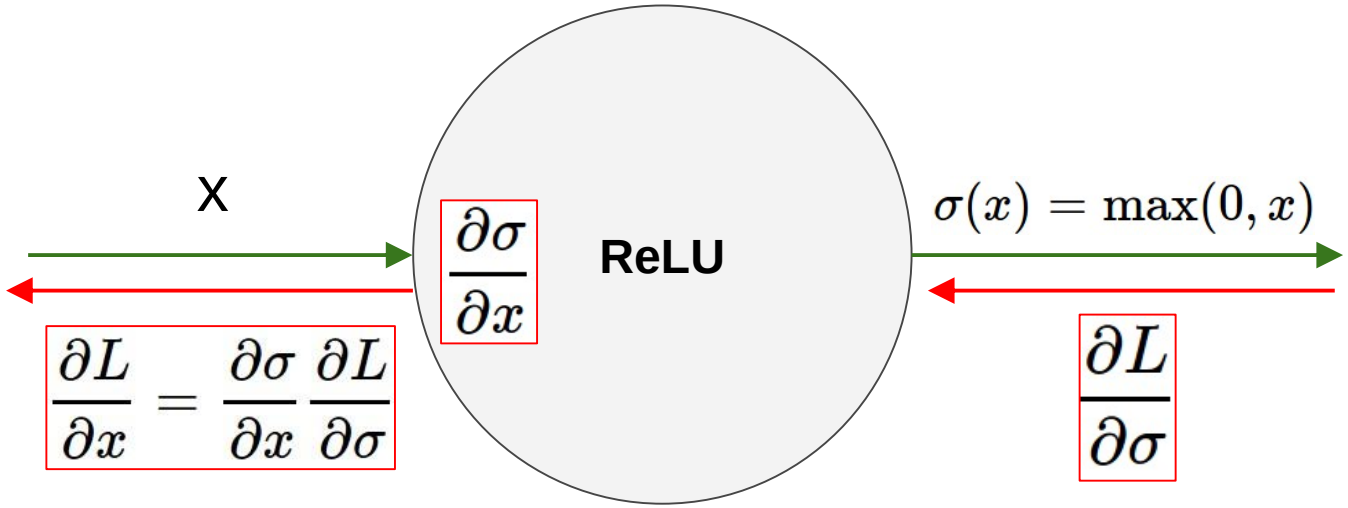
$$\max(0, x)$$

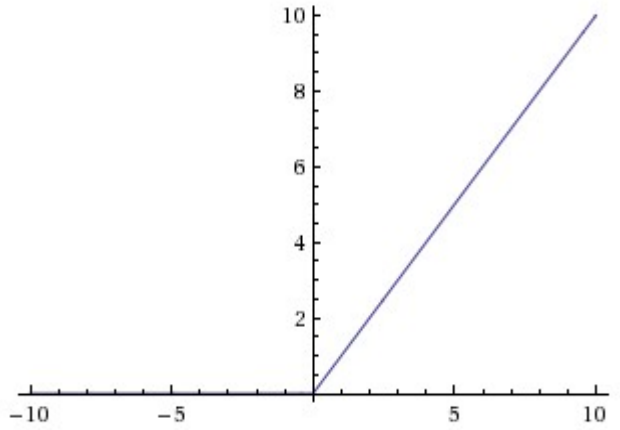


ReLU

[Krizhevsky et al., 2012]

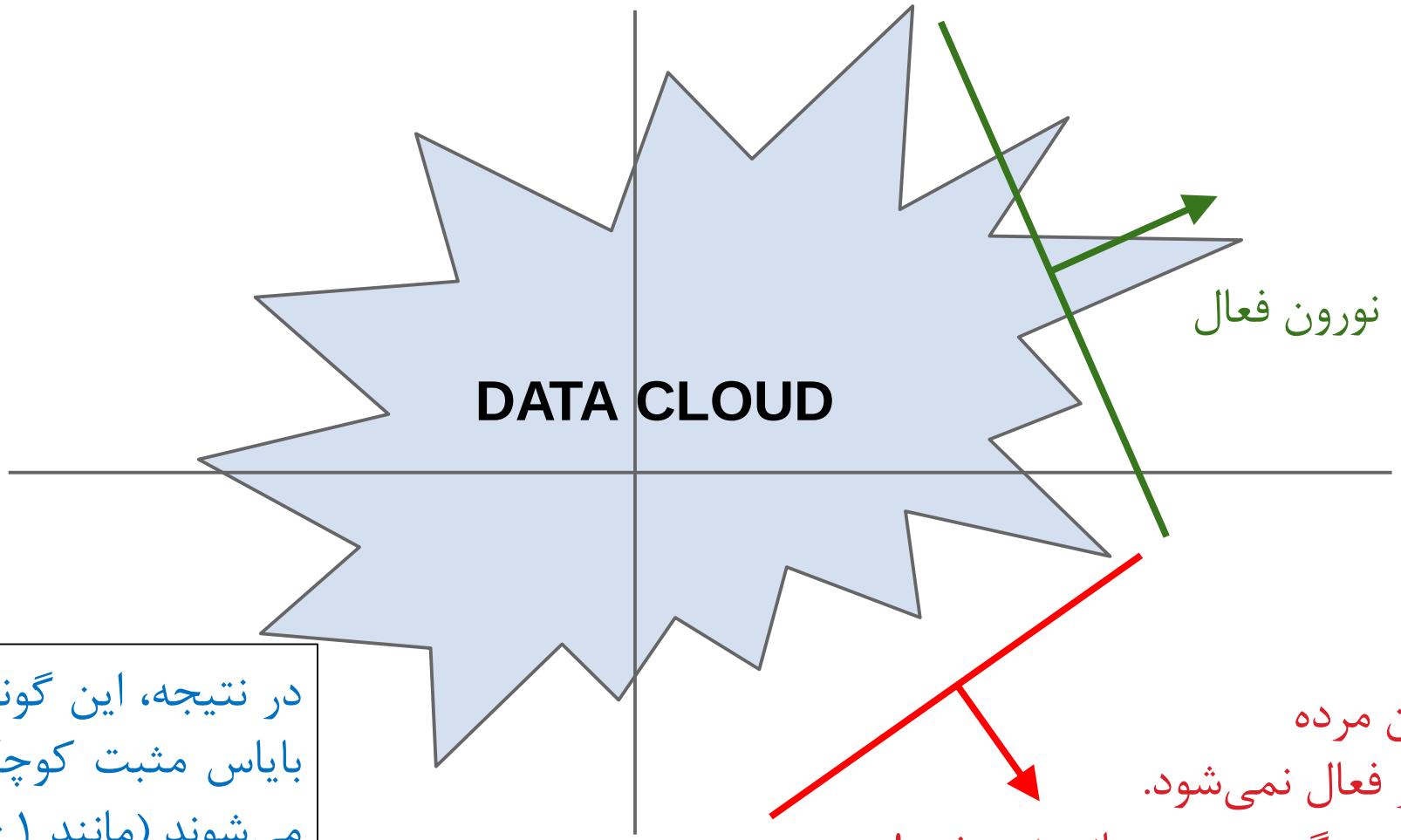
توابع فعالیت





- چه اتفاقی می‌افتد اگر $x = -10$ ؟
- چه اتفاقی می‌افتد اگر $x = 0$ ؟
- چه اتفاقی می‌افتد اگر $x = +10$ ؟

توابع فعالیت

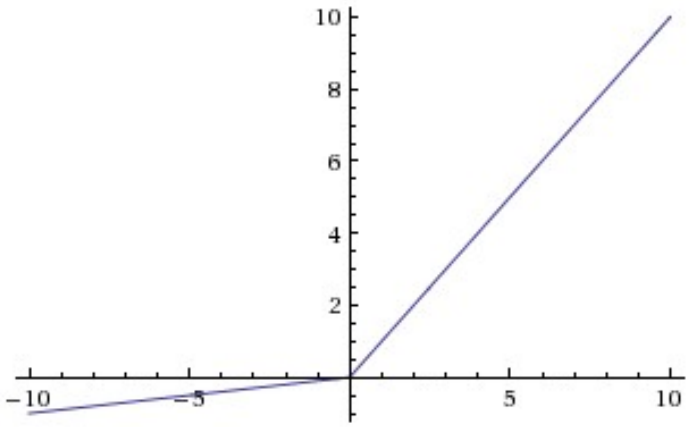


در نتیجه، این گونه نورون‌ها با مقادیر بایاس مثبت کوچک مقداردهی اولیه می‌شوند (مانند ۰/۰۱)

توابع فعالیت

- اشباع نمی‌شود (در نواحی مثبت)
- از نظر محاسباتی بسیار کارا است.
- در عمل، بسیار سریع‌تر از توابع سیگموید و تانژانت هایپربولیک همگرا می‌شود. (مثلاً ۶ برابر)
- **غیرفعال نمی‌شود!**

$$\max(0.01x, x)$$

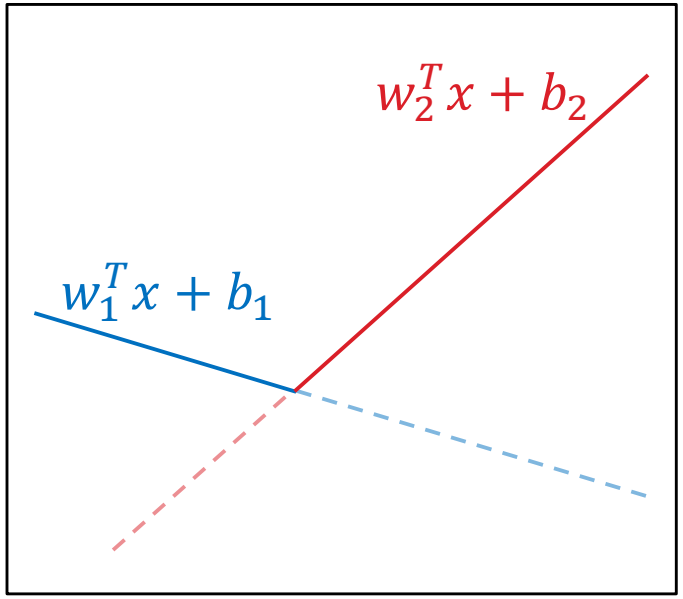


Leaky ReLU

[Mass et al., 2013] [He et al., 2015]

تابع فعالیت Maxout

- یک تابع تکه‌ای خطی. [مجموعه‌ای از چند تابع خطی]
- تعمیم دهنده‌ی توابع فعالیت ReLU و Leaky ReLU
- رفتار خطی! هرگز اشباع نمی‌شود! هرگز غیرفعال نمی‌شود!



$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

مشکل. دو برابر (چند برابر) شدن تعداد پارامترها به ازای هر نورون!

[Goodfellow et al., 2013]

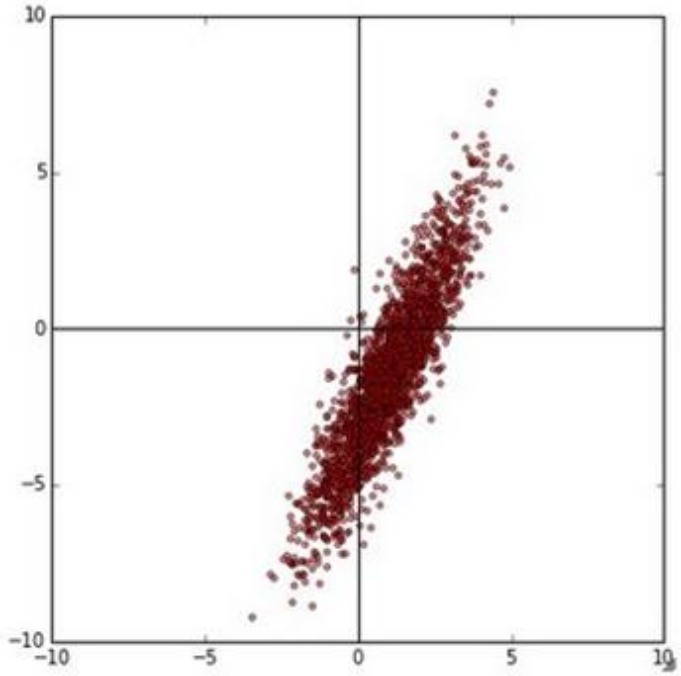
انتخاب تابع فعالیت

- در عمل.
  - از ReLU استفاده کنید. مراقب نرخ یادگیری باشید.
  - توابع Leaky ReLU و Maxout را امتحان کنید.
  - تابع تانژانت هایپربولیک را نیز امتحان کنید، اما انتظار زیادی از آن نداشته باشید.
  - از تابع سیگموید استفاده نکنید.

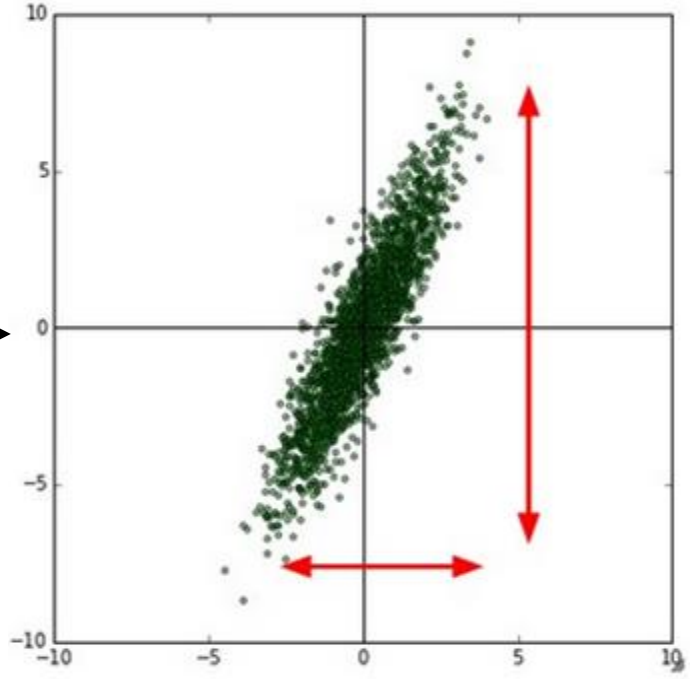
پیش‌پردازش داده‌ها

پیش‌پردازش داده‌ها

داده‌های اصلی

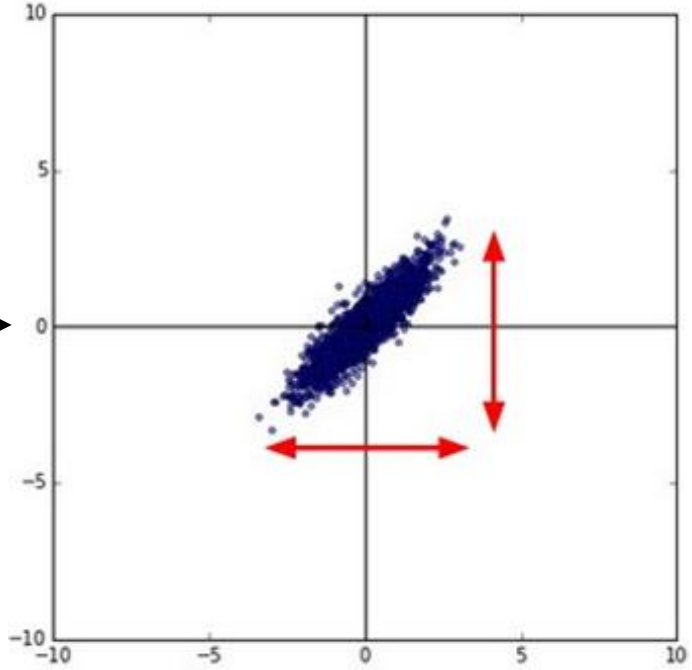


داده‌ها با میانگین صفر



```
X -= np.mean(X, axis=0)
```

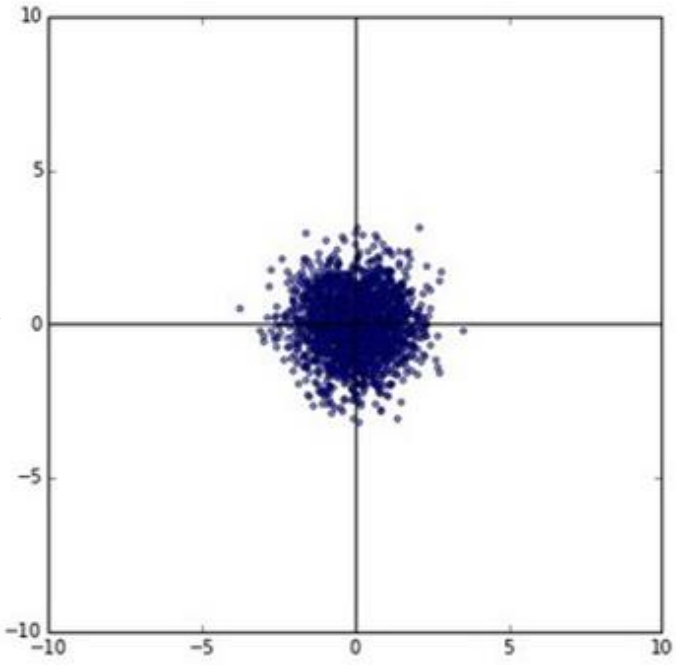
داده‌های نرمال شده



```
X /= np.std(X, axis=0)
```

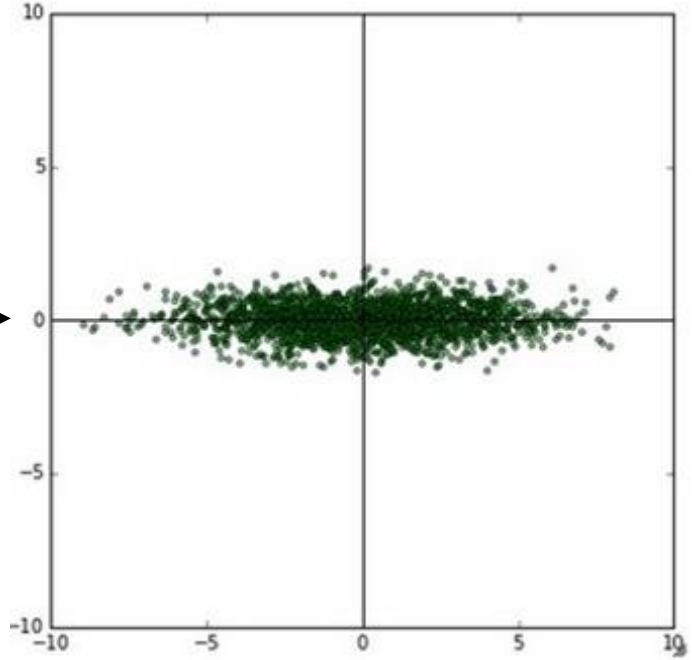
پیش‌پردازش داده‌ها

داده‌های سفید شده



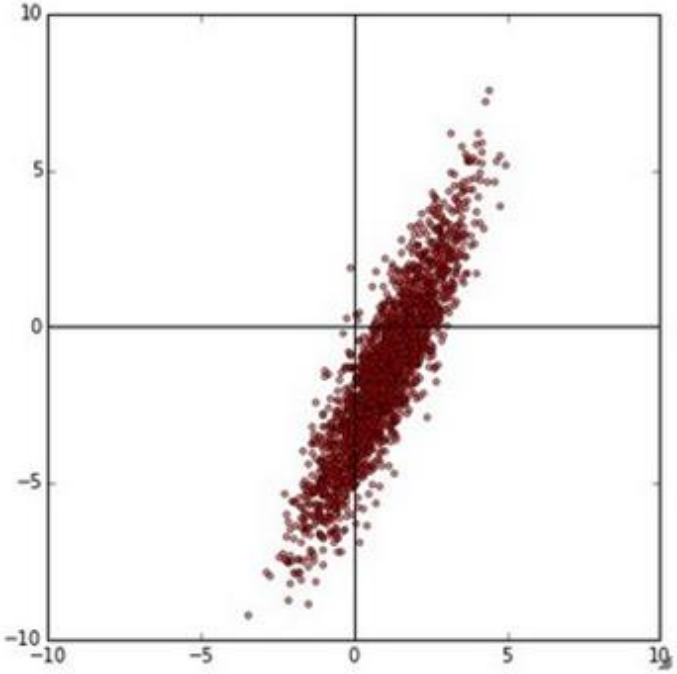
تبدیل ماتریس کوواریانس به یک ماتریس همانی

داده‌های ناهمبسته شده



تبدیل ماتریس کوواریانس به یک ماتریس قطری

داده‌های اصلی



پیش‌پردازش برای تصاویر

- در عمل برای داده‌های تصویری تنها از صفر کردن مرکز استفاده می‌شود.

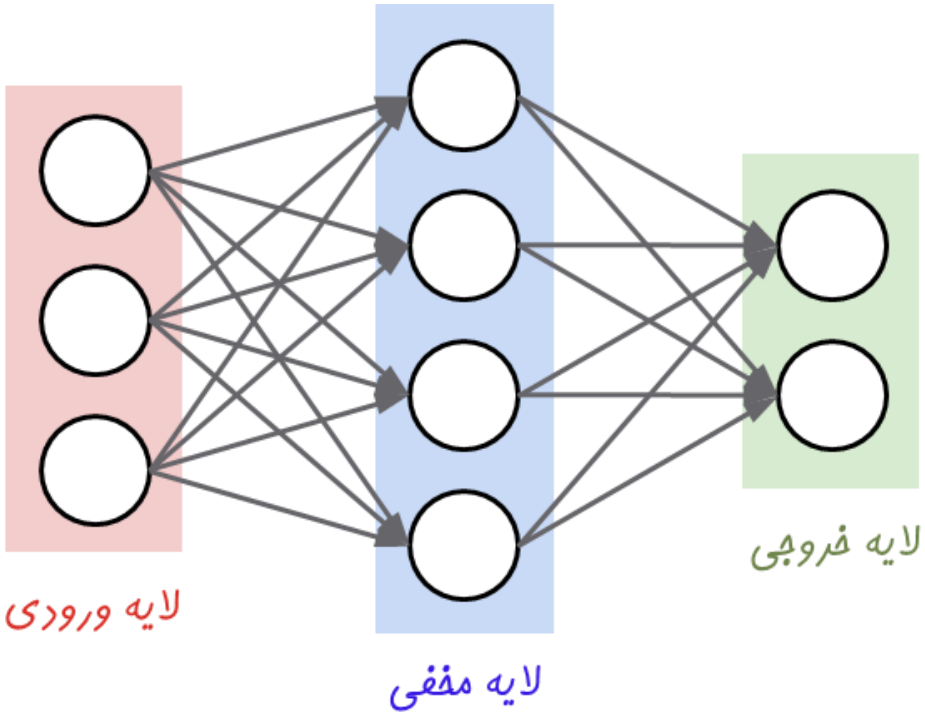
- به عنوان نمونه، تصاویر مجموعه داده‌ی CIFAR-10 را با ابعاد [۳، ۳۲، ۳۲] در نظر بگیرید:
  - **روش اول:** کم کردن تصویر میانگین از تمام تصاویر [مانند AlexNet]
    - ابعاد بردار میانگین: [۳، ۳۲، ۳۲]
  - **روش دوم:** کم کردن میانگین به ازای هر کانال رنگ [مانند VGGNet]
    - ابعاد بردار میانگین: ۳ عدد

نرمال‌سازی واریانس، ناهمبسته‌سازی و سفیدسازی، برای داده‌های تصویری چندان مرسوم نیست.

مقداردهی اولیه به وزن‌ها

مقداردهی اولیه به وزن‌ها

- پرسش. اگر در ابتدا تمام وزن‌ها با صفر مقداردهی شوند، چه روی خواهد داد؟



مقداردهی اولیه به وزن‌ها

- ایده‌ی اول. اعداد تصادفی کوچک.

[توزیع گاوسی با میانگین صفر و انحراف معیار ۰/۰۱]

```
W = 0.01 * np.random.randn(D, H)
```

- برای شبکه‌های کوچک خوب است، اما برای شبکه‌های بزرگ‌تر ممکن است به توزیع ناهمگن مقدار فعالیت‌ها در طول لایه‌های مختلف شبکه منجر شود!

مقداردهی اولیه به وزن‌ها

- مثال.
  - یک شبکه با ۱۰ لایه، و
  - ۵۰۰ نورون در هر لایه، و
  - تابع فعالیت tanh

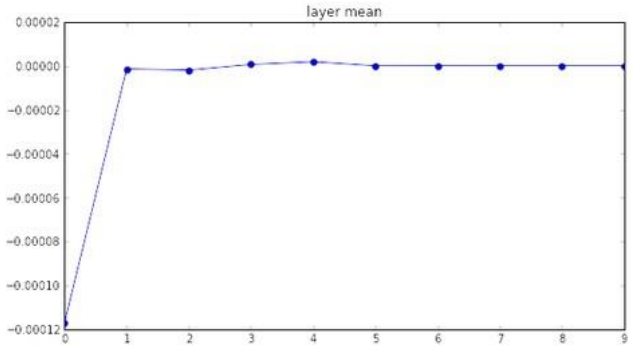
```
# assume some unit gaussian 10-D input data
D = np.random.randn(1000, 500)
hidden_layer_sizes = [500]*10
nonlinearities = ['tanh']*len(hidden_layer_sizes)
```

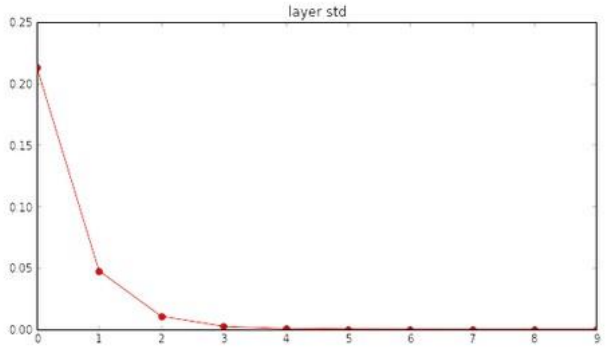
```
act = {'relu':lambda x:np.maximum(0,x), 'tanh':lambda x:np.tanh(x)}
Hs = {}
for i in xrange(len(hidden_layer_sizes)):
    X = D if i == 0 else Hs[i-1] # input at this layer
    fan_in = X.shape[1]
    fan_out = hidden_layer_sizes[i]
    W = np.random.randn(fan_in, fan_out) * 0.01 # layer initialization

    H = np.dot(X, W) # matrix multiply
    H = act[nonlinearities[i]](H) # nonlinearity
    Hs[i] = H # cache result on this layer
```

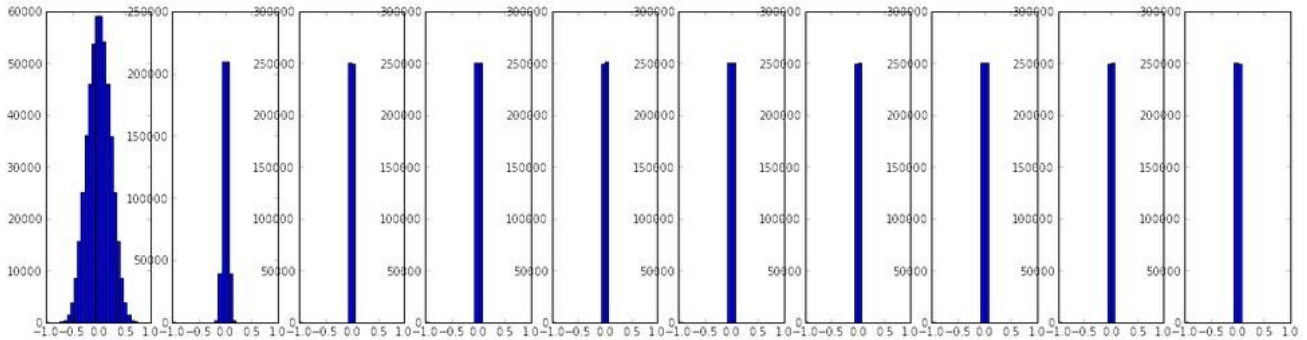
```
# look at distributions at each layer
print 'input layer had mean %f and std %f' % (np.mean(D), np.std(D))
layer_means = [np.mean(H) for i,H in Hs.iteritems()]
layer_stds = [np.std(H) for i,H in Hs.iteritems()]
for i,H in Hs.iteritems():
    print 'hidden layer %d had mean %f and std %f' % (i+1, layer_means[i], layer_stds[i])

# plot the means and standard deviations
plt.figure()
plt.subplot(121)
plt.plot(Hs.keys(), layer_means, 'ob-')
plt.title('layer mean')
plt.subplot(122)
plt.plot(Hs.keys(), layer_stds, 'or-')
plt.title('layer std')

# plot the raw distributions
plt.figure()
for i,H in Hs.iteritems():
    plt.subplot(1,len(Hs),i+1)
    plt.hist(H.ravel(), 30, range=(-1,1))
```

مقداردهی اولیه به وزن‌ها


```
input layer had mean 0.000927 and std 0.998388
hidden layer 1 had mean -0.000117 and std 0.213081
hidden layer 2 had mean -0.000001 and std 0.047551
hidden layer 3 had mean -0.000002 and std 0.010630
hidden layer 4 had mean 0.000001 and std 0.002378
hidden layer 5 had mean 0.000002 and std 0.000532
hidden layer 6 had mean -0.000000 and std 0.000119
hidden layer 7 had mean 0.000000 and std 0.000026
hidden layer 8 had mean -0.000000 and std 0.000006
hidden layer 9 had mean 0.000000 and std 0.000001
hidden layer 10 had mean -0.000000 and std 0.000000
```







```
W = np.random.randn(fan_in, fan_out) * 0.01
```

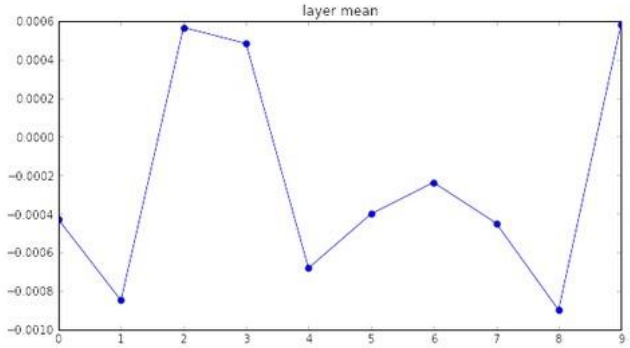
- مقدار فعالیت نورون‌ها صفر می‌شود!

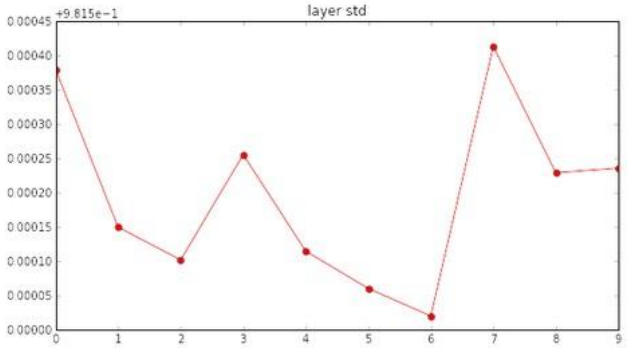
- پرسش. در طول محاسبات رو به عقب، مقدار گرادیان‌ها به چه صورت خواهد بود؟

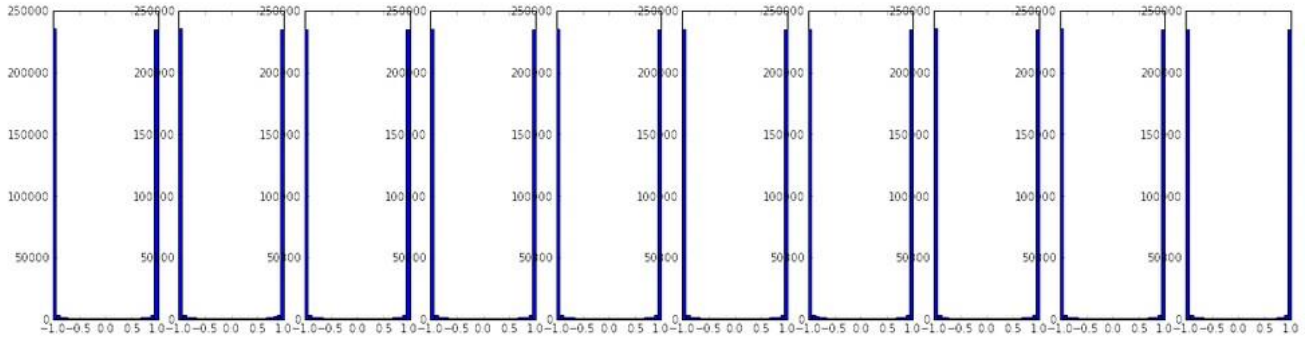
راهنمایی: در گراف محاسباتی، گره‌هایی را در نظر بگیرید که عمل W×X را انجام می‌دهند.

مقداردهی اولیه به وزن‌ها


```
input layer had mean 0.001800 and std 1.001311
hidden layer 1 had mean -0.000430 and std 0.981879
hidden layer 2 had mean -0.000849 and std 0.981649
hidden layer 3 had mean 0.000566 and std 0.981601
hidden layer 4 had mean 0.000483 and std 0.981755
hidden layer 5 had mean -0.000682 and std 0.981614
hidden layer 6 had mean -0.000401 and std 0.981560
hidden layer 7 had mean -0.000237 and std 0.981520
hidden layer 8 had mean -0.000448 and std 0.981913
hidden layer 9 had mean -0.000899 and std 0.981728
hidden layer 10 had mean 0.000584 and std 0.981736
```







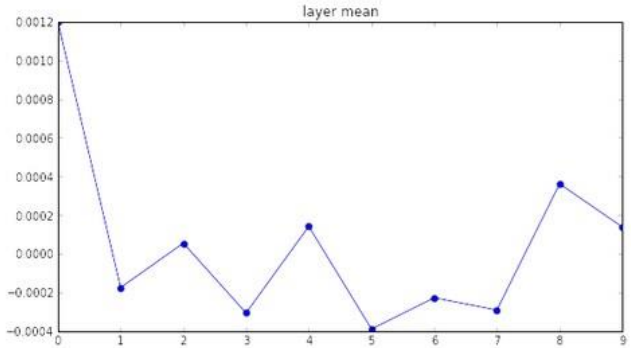
- تقریباً تمام نورون‌ها اشباع می‌شوند. [۱+ یا ۱-].
- مقدار تمام گرادیان‌ها برابر با صفر خواهد بود.

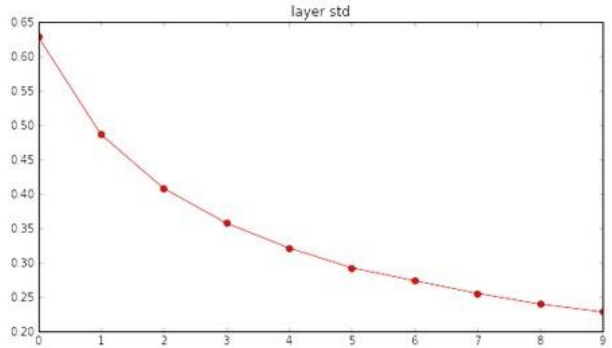
وزن‌دهی با مقادیر بزرگ

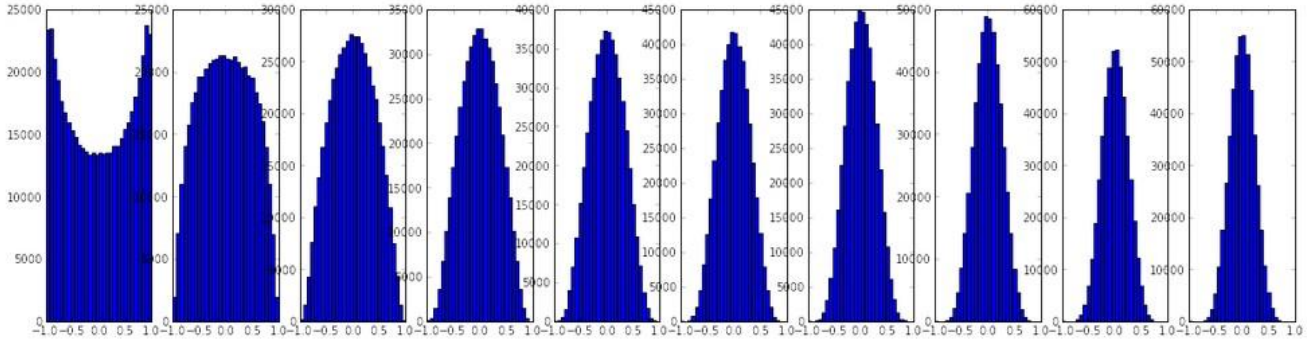
```
W = np.random.randn(fan_in, fan_out) * 1.0
```

مقداردهی اولیه به وزن‌ها


```
input layer had mean 0.001800 and std 1.001311
hidden layer 1 had mean 0.001198 and std 0.627953
hidden layer 2 had mean -0.000175 and std 0.486051
hidden layer 3 had mean 0.000055 and std 0.407723
hidden layer 4 had mean -0.000306 and std 0.357108
hidden layer 5 had mean 0.000142 and std 0.320917
hidden layer 6 had mean -0.000389 and std 0.292116
hidden layer 7 had mean -0.000228 and std 0.273387
hidden layer 8 had mean -0.000291 and std 0.254935
hidden layer 9 had mean 0.000361 and std 0.239266
hidden layer 10 had mean 0.000139 and std 0.228008
```





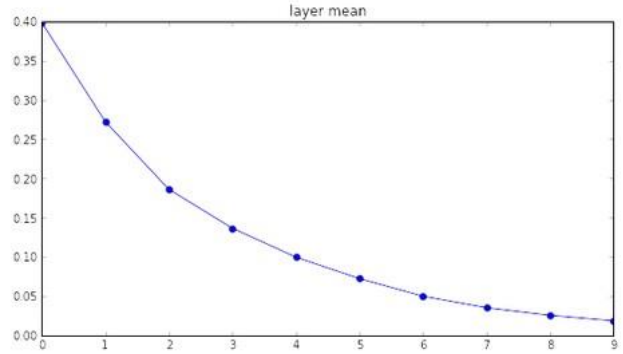


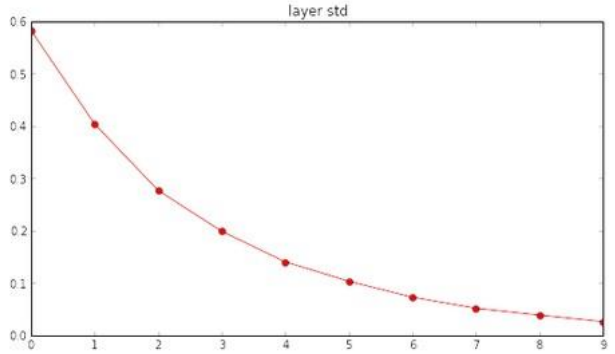
- وزن‌دهی خاویر. [Gloret et al., 2010]

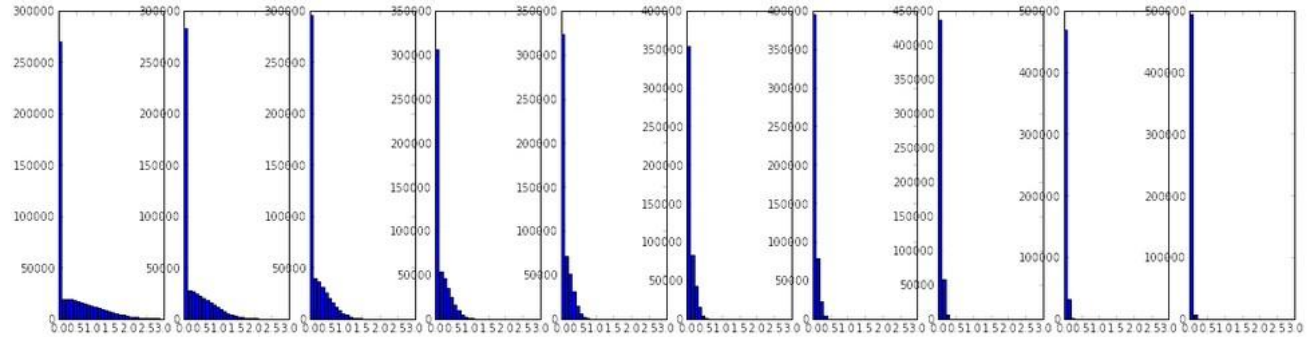
```
W = np.random.randn(fan_in, fan_out) / np.sqrt(fan_in)
```

مقداردهی اولیه به وزن‌ها


```
input layer had mean 0.000501 and std 0.999444
hidden layer 1 had mean 0.398623 and std 0.582273
hidden layer 2 had mean 0.272352 and std 0.403795
hidden layer 3 had mean 0.186076 and std 0.276912
hidden layer 4 had mean 0.136442 and std 0.198685
hidden layer 5 had mean 0.099568 and std 0.140299
hidden layer 6 had mean 0.072234 and std 0.103280
hidden layer 7 had mean 0.049775 and std 0.072748
hidden layer 8 had mean 0.035138 and std 0.051572
hidden layer 9 had mean 0.025404 and std 0.038583
hidden layer 10 had mean 0.018408 and std 0.026076
```







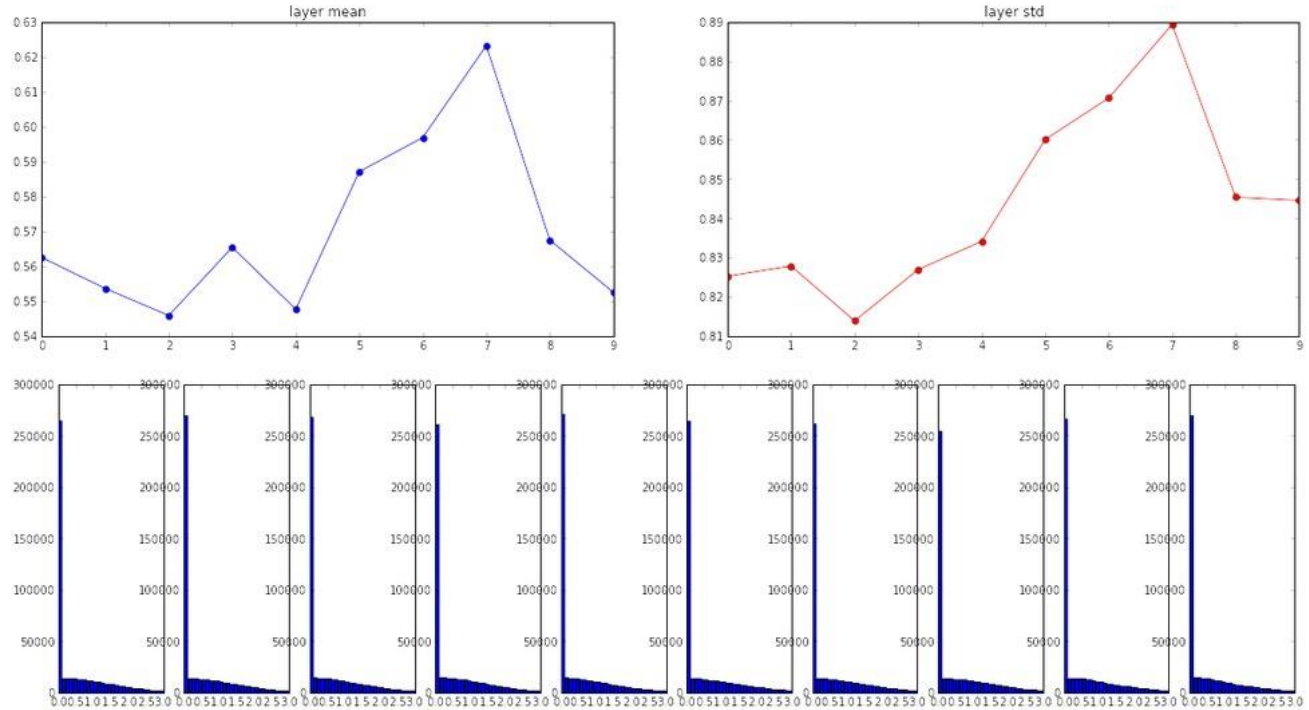
- وزن‌دهی خاویر. [Gloret et al., 2010]

اما در صورت استفاده از تابع فعالیت ReLU، این روش دیگر به درستی عمل نخواهد کرد!

```
W = np.random.randn(fan_in, fan_out) / np.sqrt(fan_in)
```

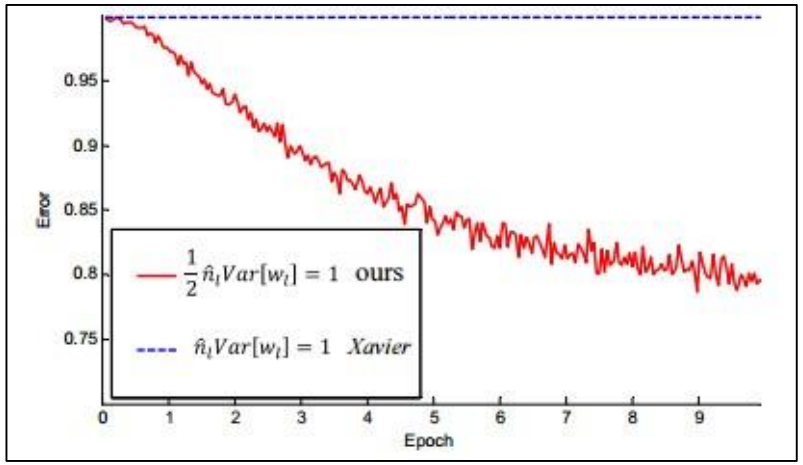
مقداردهی اولیه به وزن‌ها


```
input layer had mean 0.000501 and std 0.999444
hidden layer 1 had mean 0.562488 and std 0.825232
hidden layer 2 had mean 0.553614 and std 0.827835
hidden layer 3 had mean 0.545867 and std 0.813855
hidden layer 4 had mean 0.565396 and std 0.826902
hidden layer 5 had mean 0.547678 and std 0.834092
hidden layer 6 had mean 0.587103 and std 0.860035
hidden layer 7 had mean 0.596867 and std 0.870610
hidden layer 8 had mean 0.623214 and std 0.889348
hidden layer 9 had mean 0.567498 and std 0.845357
hidden layer 10 had mean 0.552531 and std 0.844523
```



- روش بهتر. [He et al., 2015]

[به ضریب $\frac{۱}{۲}$ توجه کنید]

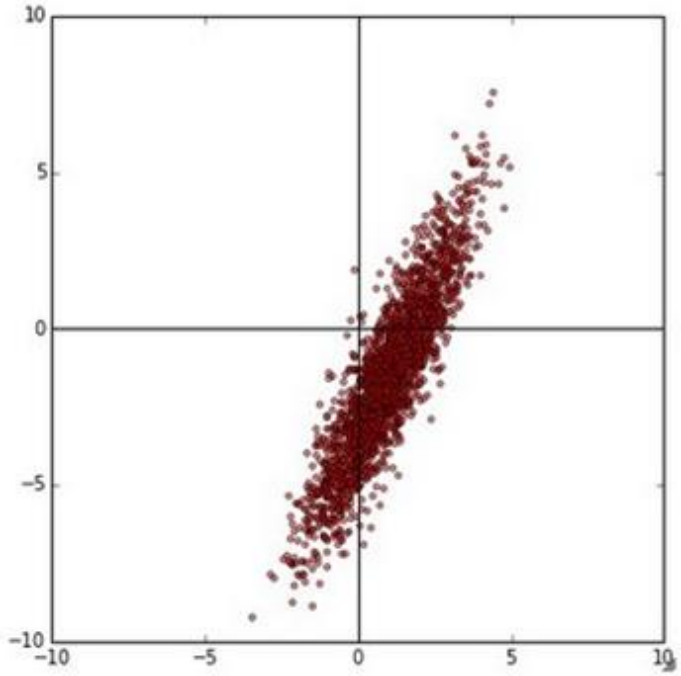



```
W = np.random.randn(fan_in, fan_out) / np.sqrt(fan_in/2)
```

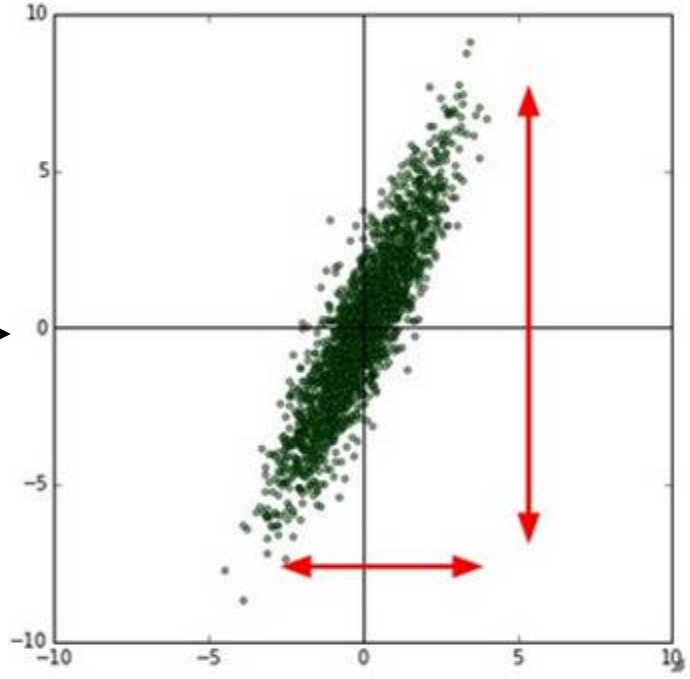
مراحل توسعه فرایند آموزش

گام اول: پیش‌پردازش داده‌ها

داده‌های اصلی

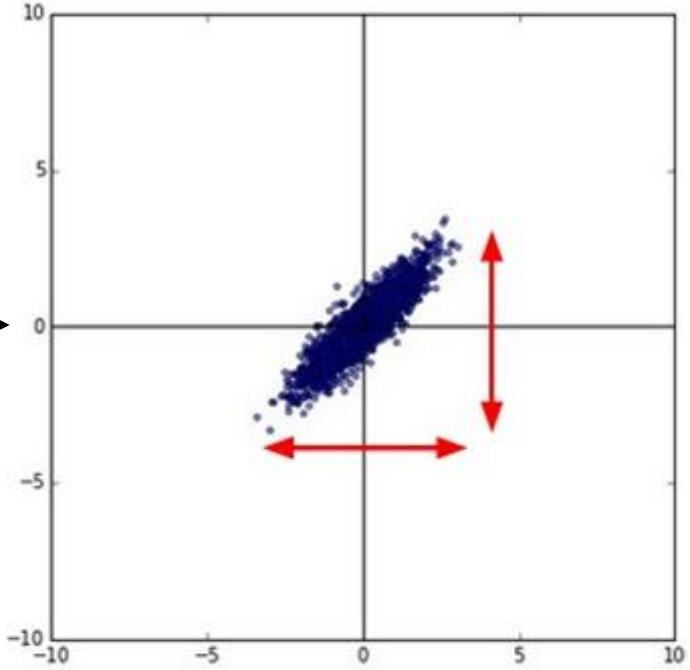


داده‌ها با میانگین صفر



```
X -= np.mean(X, axis=0)
```

داده‌های نرمال شده



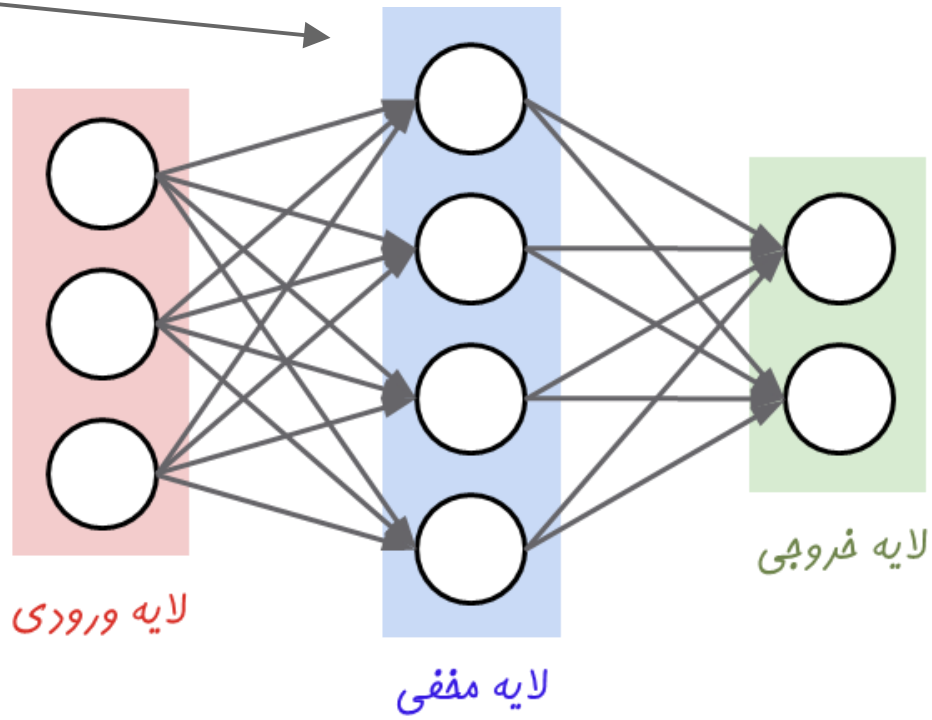
```
X /= np.std(X, axis=0)
```

گام دوم: انتخاب معماری

- فرض کنید کار خود را با یک لایه مخفی شامل ۵۰ نورون شروع کنیم.

۵۰ نورون مخفی

[تصمیم طراحی]



۳۰۷۲ ورودی

[تصاویر CIFAR-10]

۱۰ نورون خروجی

[یکی به ازای هر کلاس]

گام سوم: بررسی مقدار تابع هزینه (بدون تنظیم)


```
def init_two_layer_model(input_size, hidden_size, output_size):
    # initialize a model
    model = {}
    model['W1'] = 0.0001 * np.random.randn(input_size, hidden_size)
    model['b1'] = np.zeros(hidden_size)
    model['W2'] = 0.0001 * np.random.randn(hidden_size, output_size)
    model['b2'] = np.zeros(output_size)
    return model
```

```
model = init_two_layer_model(32*32*3, 50, 10)  # input size, hidden size, number of classes
loss, grad = two_layer_net(X_train, y_train, model, 0.0 )
print loss
```

غیرفعال کردن تنظیم

```
2.30261216167
```

برگردان مقدار تابع هزینه
و گرادیان تمام پارامترها

این مقدار برای ۱۰ کلاس
«درست» به نظر می‌رسد!

گام سوم: بررسی مقدار تابع هزینه (با تنظیم)


```
def init_two_layer_model(input_size, hidden_size, output_size):
    # initialize a model
    model = {}
    model['W1'] = 0.0001 * np.random.randn(input_size, hidden_size)
    model['b1'] = np.zeros(hidden_size)
    model['W2'] = 0.0001 * np.random.randn(hidden_size, output_size)
    model['b2'] = np.zeros(output_size)
    return model
```

```
model = init_two_layer_model(32*32*3, 50, 10)  # input size, hidden size, number of classes
loss, grad = two_layer_net(X_train, y_train, model, 1e3 )
print loss
```

فعال کردن تنظیم

```
3.06859716482
```

مقدار تابع هزینه باید افزایش یابد

گام چهارم: آموزش مقدماتی


```
model = init_two_layer_model(32*32*3, 50, 10) # input size, hidden size, number of classes
trainer = ClassifierTrainer()
X_tiny = X_train[:20] # take 20 examples
y_tiny = y_train[:20]
best_model, stats = trainer.train(X_tiny, y_tiny, X_tiny, y_tiny,
                                  model, two_layer_net,
                                  num_epochs=200, reg=0.0,
                                  update='sgd', learning_rate_decay=1,
                                  sample_batches = False,
                                  learning_rate=1e-3, verbose=True)
```

- نکته. مطمئن شوید آموزش شبکه بر روی یک بخش بسیار کوچک از مجموعه آموزشی، باعث بروز بیش‌برازش می‌شود!

در تکه کد بالا:

- ۲۰ نمونه اول از مجموعه‌ی آموزشی انتخاب می‌شود.
- تنظیم غیرفعال می‌شود.
- از نسخه ساده گرادیان کاهشی استفاده می‌شود.

گام چهارم: آموزش مقدماتی


```
model = init_two_layer_model(32*32*3, 50, 10) # input size, hidden size, number of classes
trainer = ClassifierTrainer()
X_tiny = X_train[:20] # take 20 examples
y_tiny = y_train[:20]
best_model, stats = trainer.train(X_tiny, y_tiny, X_tiny, y_tiny,
                                  model, two_layer_net,
                                  num_epochs=200, reg=0.0,
                                  update='sgd', learning_rate_decay=1,
                                  sample_batches = False,
                                  learning_rate=1e-3, verbose=True)
```

```
Finished epoch 1 / 200: cost 2.302603, train: 0.400000, val 0.400000, lr 1.000000e-03
Finished epoch 2 / 200: cost 2.302258, train: 0.450000, val 0.450000, lr 1.000000e-03
Finished epoch 3 / 200: cost 2.301849, train: 0.600000, val 0.600000, lr 1.000000e-03
Finished epoch 4 / 200: cost 2.301196, train: 0.650000, val 0.650000, lr 1.000000e-03
Finished epoch 5 / 200: cost 2.300044, train: 0.650000, val 0.650000, lr 1.000000e-03
Finished epoch 6 / 200: cost 2.297864, train: 0.550000, val 0.550000, lr 1.000000e-03
Finished epoch 7 / 200: cost 2.293595, train: 0.600000, val 0.600000, lr 1.000000e-03
Finished epoch 8 / 200: cost 2.285096, train: 0.550000, val 0.550000, lr 1.000000e-03
Finished epoch 9 / 200: cost 2.268094, train: 0.550000, val 0.550000, lr 1.000000e-03
Finished epoch 10 / 200: cost 2.234787, train: 0.500000, val 0.500000, lr 1.000000e-03
Finished epoch 11 / 200: cost 2.173187, train: 0.500000, val 0.500000, lr 1.000000e-03
Finished epoch 12 / 200: cost 2.076862, train: 0.500000, val 0.500000, lr 1.000000e-03
Finished epoch 13 / 200: cost 1.974090, train: 0.400000, val 0.400000, lr 1.000000e-03
Finished epoch 14 / 200: cost 1.895885, train: 0.400000, val 0.400000, lr 1.000000e-03
Finished epoch 15 / 200: cost 1.820876, train: 0.450000, val 0.450000, lr 1.000000e-03
Finished epoch 16 / 200: cost 1.737430, train: 0.450000, val 0.450000, lr 1.000000e-03
Finished epoch 17 / 200: cost 1.642356, train: 0.500000, val 0.500000, lr 1.000000e-03
Finished epoch 18 / 200: cost 1.535239, train: 0.600000, val 0.600000, lr 1.000000e-03
Finis
Finished epoch 195 / 200: cost 0.002694, train: 1.000000, val 1.000000, lr 1.000000e-03
Finished epoch 196 / 200: cost 0.002674, train: 1.000000, val 1.000000, lr 1.000000e-03
Finished epoch 197 / 200: cost 0.002655, train: 1.000000, val 1.000000, lr 1.000000e-03
Finished epoch 198 / 200: cost 0.002635, train: 1.000000, val 1.000000, lr 1.000000e-03
Finished epoch 199 / 200: cost 0.002617, train: 1.000000, val 1.000000, lr 1.000000e-03
Finished epoch 200 / 200: cost 0.002597, train: 1.000000, val 1.000000, lr 1.000000e-03
finished optimization. best validation accuracy: 1.000000
```

- نکته. مطمئن شوید آموزش شبکه بر روی یک بخش بسیار کوچک از مجموعه آموزشی، باعث بروز بیش‌برازش می‌شود!

مقدار تابع هزینه بسیار کم،
دقت آموزش ۱۰۰ درصد،

گام چهارم: آموزش مقدماتی


```
model = init_two_layer_model(32*32*3, 50, 10) # input size, hidden size, number of classes
trainer = ClassifierTrainer()
best_model, stats = trainer.train(X_train, y_train, X_val, y_val,
                                  model, two_layer_net,
                                  num_epochs=10, reg=0.000001,
                                  update='sgd', learning_rate_decay=1,
                                  sample_batches = True,
                                  learning_rate=1e-6, verbose=True)
```

- به طور معمول، با یک ضریب تنظیم بسیار کوچک شروع می‌کنیم.
- سعی می‌کنیم یک مقدار برای نرخ یادگیری بیابیم، به گونه‌ای که باعث شود مقدار تابع هزینه در هر تکرار کاهش یابد.

گام چهارم: آموزش مقدماتی

- به طور معمول، با یک ضریب تنظیم بسیار کوچک شروع می‌کنیم.
- سعی می‌کنیم یک مقدار برای نرخ یادگیری بیابیم، به گونه‌ای که باعث شود مقدار تابع هزینه در هر تکرار کاهش یابد.

```
model = init_two_layer_model(32*32*3, 50, 10) # input size, hidden size, number of classes
trainer = ClassifierTrainer()
best_model, stats = trainer.train(X_train, y_train, X_val, y_val,
                                  model, two_layer_net,
                                  num_epochs=10, reg=0.000001,
                                  update='sgd', learning_rate_decay=1,
                                  sample_batches = True,
                                  learning_rate=1e-6, verbose=True)
```

```
Finished epoch 1 / 10: cost 2.302576, train: 0.080000, val 0.103000, lr 1.000000e-06
Finished epoch 2 / 10: cost 2.302582, train: 0.121000, val 0.124000, lr 1.000000e-06
Finished epoch 3 / 10: cost 2.302558, train: 0.119000, val 0.138000, lr 1.000000e-06
Finished epoch 4 / 10: cost 2.302519, train: 0.127000, val 0.151000, lr 1.000000e-06
Finished epoch 5 / 10: cost 2.302517, train: 0.158000, val 0.171000, lr 1.000000e-06
Finished epoch 6 / 10: cost 2.302518, train: 0.179000, val 0.172000, lr 1.000000e-06
Finished epoch 7 / 10: cost 2.302466, train: 0.180000, val 0.176000, lr 1.000000e-06
Finished epoch 8 / 10: cost 2.302452, train: 0.175000, val 0.185000, lr 1.000000e-06
Finished epoch 9 / 10: cost 2.302459, train: 0.206000, val 0.192000, lr 1.000000e-06
Finished epoch 10 / 10: cost 2.302420, train: 0.190000, val 0.192000, lr 1.000000e-06
finished optimization. best validation accuracy: 0.192000
```

مقدار تابع هزینه بسیار کم تغییر می‌کند!

کاهش نیافتن تابع هزینه:

نرخ یادگیری بیش از حد کوچک است!

گام چهارم: آموزش مقدماتی


```
model = init_two_layer_model(32*32*3, 50, 10) # input size, hidden size, number of classes
trainer = ClassifierTrainer()
best_model, stats = trainer.train(X_train, y_train, X_val, y_val,
                                  model, two_layer_net,
                                  num_epochs=10, reg=0.000001,
                                  update='sgd', learning_rate_decay=1,
                                  sample_batches = True,
                                  learning_rate=1e6, verbose=True)
```

اکنون یک مقدار بزرگ مانند 1e6 را برای نرخ یادگیری امتحان می‌کنیم. چه مشکلی ممکن است بروز کند؟

- به طور معمول، با یک ضریب تنظیم بسیار کوچک شروع می‌کنیم.
- سعی می‌کنیم یک مقدار برای نرخ یادگیری بیابیم، به گونه‌ای که باعث شود مقدار تابع هزینه در هر تکرار کاهش یابد.

کاهش نیافتن تابع هزینه:

نرخ یادگیری بیش از حد کوچک است!

گام چهارم: آموزش مقدماتی


```
model = init_two_layer_model(32*32*3, 50, 10) # input size, hidden size, number of classes
trainer = ClassifierTrainer()
best_model, stats = trainer.train(X_train, y_train, X_val, y_val,
                                  model, two_layer_net,
                                  num_epochs=10, reg=0.000001,
                                  update='sgd', learning_rate_decay=1,
                                  sample_batches = True,
                                  learning_rate=1e6, verbose=True)
```

```
/home/karpathy/cs231n/code/cs231n/classifiers/neural_net.py:50: RuntimeWarning: divide by zero en
countered in log
  data_loss = -np.sum(np.log(probs[range(N), y])) / N
/home/karpathy/cs231n/code/cs231n/classifiers/neural_net.py:48: RuntimeWarning: invalid value enc
ountered in subtract
  probs = np.exp(scores - np.max(scores, axis=1, keepdims=True))
Finished epoch 1 / 10: cost nan, train: 0.091000, val 0.087000, lr 1.000000e+06
Finished epoch 2 / 10: cost nan, train: 0.095000, val 0.087000, lr 1.000000e+06
Finished epoch 3 / 10: cost nan, train: 0.100000, val 0.087000, lr 1.000000e+06
```

مقدار تابع هزینه: NaN

تقریباً همیشه به این معناست که مقدار نرخ یادگیری بیش از حد بزرگ است.

- به طور معمول، با یک ضریب تنظیم بسیار کوچک شروع می‌کنیم.
- سعی می‌کنیم یک مقدار برای نرخ یادگیری بیابیم، به گونه‌ای که باعث شود مقدار تابع هزینه در هر تکرار کاهش یابد.

کاهش نیافتن تابع هزینه:

نرخ یادگیری بیش از حد کوچک است!

رشد انفجاری تابع هزینه:

نرخ یادگیری بیش از حد بزرگ است!

گام چهارم: آموزش مقدماتی


```
model = init_two_layer_model(32*32*3, 50, 10) # input size, hidden size, number of classes
trainer = ClassifierTrainer()
best_model, stats = trainer.train(X_train, y_train, X_val, y_val,
                                  model, two_layer_net,
                                  num_epochs=10, reg=0.000001,
                                  update='sgd', learning_rate_decay=1,
                                  sample_batches = True,
                                  learning_rate=3e-3, verbose=True)
```

```
Finished epoch 1 / 10: cost 2.186654, train: 0.308000, val 0.306000, lr 3.000000e-03
Finished epoch 2 / 10: cost 2.176230, train: 0.330000, val 0.350000, lr 3.000000e-03
Finished epoch 3 / 10: cost 1.942257, train: 0.376000, val 0.352000, lr 3.000000e-03
Finished epoch 4 / 10: cost 1.827868, train: 0.329000, val 0.310000, lr 3.000000e-03
Finished epoch 5 / 10: cost inf, train: 0.128000, val 0.128000, lr 3.000000e-03
Finished epoch 6 / 10: cost inf, train: 0.144000, val 0.147000, lr 3.000000e-03
```

- به طور معمول، با یک ضریب تنظیم بسیار کوچک شروع می‌کنیم.
- سعی می‌کنیم یک مقدار برای نرخ یادگیری بیابیم، به گونه‌ای که باعث شود مقدار تابع هزینه در هر تکرار کاهش یابد.

مقدار 3e-3 هنوز خیلی زیاد است.

تابع هزینه رشد انفجاری دارد.

در نتیجه بازه مناسب برای مقدار نرخ یادگیری به صورت [1e-5, 1e-3] است.

کاهش نیافتن تابع هزینه:

نرخ یادگیری بیش از حد کوچک است!

رشد انفجاری تابع هزینه:

نرخ یادگیری بیش از حد بزرگ است!

بهینه‌سازی ابرپارامترها: استراتژی اعتبارسنجی متقاطع

- استراتژی درشت به ریز. شروع با یک بازه بزرگ جستجو و سپس محدود کردن این بازه به صورت مکرر.

- مراحل اولیه:
  - تعداد کمی دوره آموزشی برای به دست آوردن یک ایده‌ی تقریبی در مورد مقدار مناسب ابرپارامترها
- مراحل بعدی:
  - اجراهای طولانی‌تر، جستجوی دقیق‌تر (تکرار در صورت نیاز)

- یک راهنمایی برای تشخیص زودهنگام رشد انفجاری تابع هزینه.
  - هر زمان مقدار هزینه از ۳ برابر هزینه اولیه بیشتر شد، اجرا را پایان دهید.

بهینه‌سازی ابرپارامترها: استراتژی اعتبارسنجی متقاطع

- مثال. اجرای جستجوی درشت با ۵ دوره آموزشی.

```
max_count = 100
for count in xrange(max_count):
      reg = 10**uniform(-5, 5)
      lr = 10**uniform(-3, -6)

      trainer = ClassifierTrainer()
      model = init_two_layer_model(32*32*3, 50, 10) # input size, hidden size, number of classes
      trainer = ClassifierTrainer()
      best_model_local, stats = trainer.train(X_train, y_train, X_val, y_val,
                                              model, two_layer_net,
                                              num_epochs=5, reg=reg,
                                              update='momentum', learning_rate_decay=0.9,
                                              sample_batches = True, batch_size = 100,
                                              learning_rate=lr, verbose=False)
```

بهینه‌سازی در فضای لگاریتمی

```
val_acc: 0.412000, lr: 1.405206e-04, reg: 4.793564e-01, (1 / 100)
val_acc: 0.214000, lr: 7.231888e-06, reg: 2.321281e-04, (2 / 100)
val_acc: 0.208000, lr: 2.119571e-06, reg: 8.011857e+01, (3 / 100)
val_acc: 0.196000, lr: 1.551131e-05, reg: 4.374936e-05, (4 / 100)
val_acc: 0.079000, lr: 1.753300e-05, reg: 1.200424e+03, (5 / 100)
val_acc: 0.223000, lr: 4.215128e-05, reg: 4.196174e+01, (6 / 100)
val_acc: 0.441000, lr: 1.750259e-04, reg: 2.110807e-04, (7 / 100)
val_acc: 0.241000, lr: 6.749231e-05, reg: 4.226413e+01, (8 / 100)
val_acc: 0.482000, lr: 4.296863e-04, reg: 6.642555e-01, (9 / 100)
val_acc: 0.079000, lr: 5.401602e-06, reg: 1.599828e+04, (10 / 100)
val_acc: 0.154000, lr: 1.618508e-06, reg: 4.925252e-01, (11 / 100)
```

خوب

بهینه‌سازی ابرپارامترها: استراتژی اعتبارسنجی متقاطع

- مثال. اجرای جستجوی درشت با ۵ دوره آموزشی.

```
max_count = 100
for count in xrange(max_count):
      reg = 10**uniform(-5, 5)
      lr = 10**uniform(-3, -6)
```

تنظیم بازه‌ی جستجو

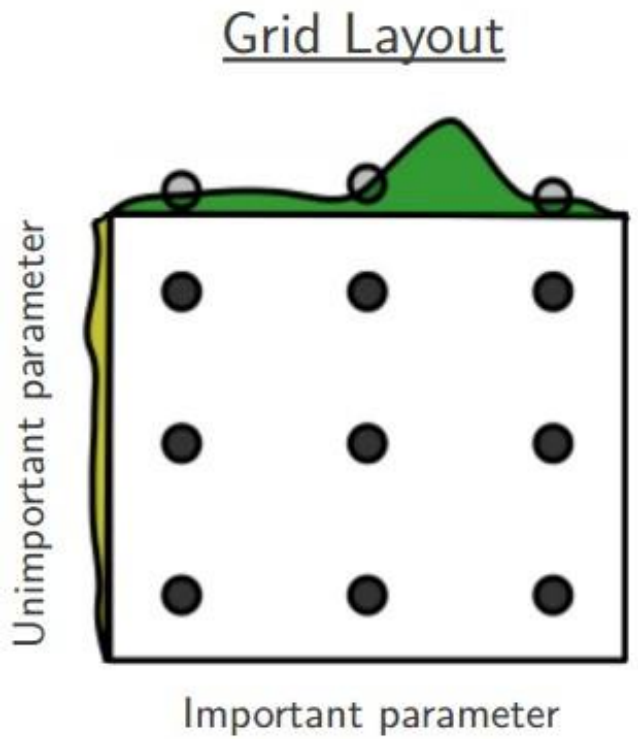
```
max_count = 100
for count in xrange(max_count):
      reg = 10**uniform(-4, 0)
      lr = 10**uniform(-3, -4)
```

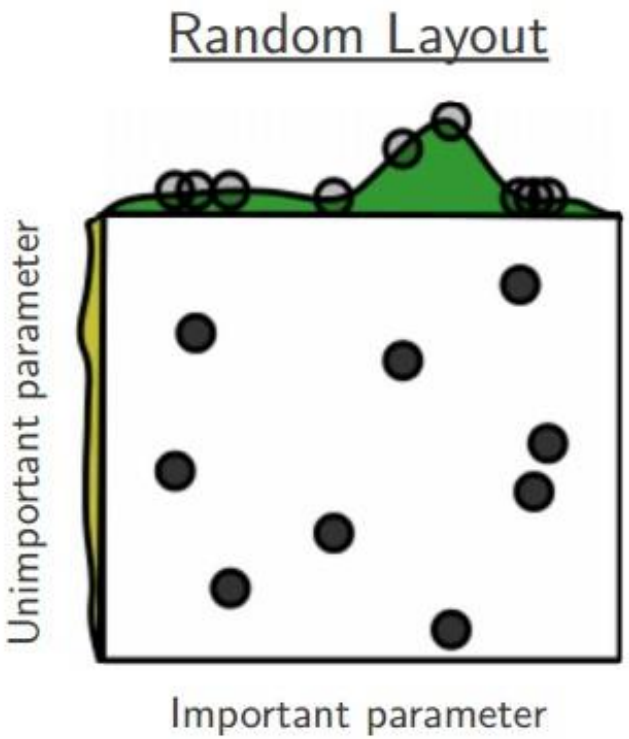
```
val_acc: 0.527000, lr: 5.340517e-04, reg: 4.097824e-01, (0 / 100)
val_acc: 0.492000, lr: 2.279484e-04, reg: 9.991345e-04, (1 / 100)
val_acc: 0.512000, lr: 8.680827e-04, reg: 1.349727e-02, (2 / 100)
val_acc: 0.461000, lr: 1.028377e-04, reg: 1.220193e-02, (3 / 100)
val_acc: 0.460000, lr: 1.113730e-04, reg: 5.244309e-02, (4 / 100)
val_acc: 0.498000, lr: 9.477776e-04, reg: 2.001293e-03, (5 / 100)
val_acc: 0.469000, lr: 1.484369e-04, reg: 4.328313e-01, (6 / 100)
val_acc: 0.522000, lr: 5.586261e-04, reg: 2.312685e-04, (7 / 100)
val_acc: 0.530000, lr: 5.808183e-04, reg: 8.259964e-02, (8 / 100)
val_acc: 0.489000, lr: 1.979168e-04, reg: 1.010889e-04, (9 / 100)
val_acc: 0.490000, lr: 2.036031e-04, reg: 2.406271e-03, (10 / 100)
val_acc: 0.475000, lr: 2.021162e-04, reg: 2.287807e-01, (11 / 100)
val_acc: 0.460000, lr: 1.135527e-04, reg: 3.905040e-02, (12 / 100)
val_acc: 0.515000, lr: 6.947668e-04, reg: 1.562808e-02, (13 / 100)
val_acc: 0.531000, lr: 9.471549e-04, reg: 1.433895e-03, (14 / 100)
val_acc: 0.509000, lr: 3.140888e-04, reg: 2.857518e-01, (15 / 100)
val_acc: 0.514000, lr: 6.438349e-04, reg: 3.033781e-01, (16 / 100)
val_acc: 0.502000, lr: 3.921784e-04, reg: 2.707126e-04, (17 / 100)
val_acc: 0.509000, lr: 9.752279e-04, reg: 2.850865e-03, (18 / 100)
val_acc: 0.500000, lr: 2.412048e-04, reg: 4.997821e-04, (19 / 100)
val_acc: 0.466000, lr: 1.319314e-04, reg: 1.189915e-02, (20 / 100)
val_acc: 0.516000, lr: 8.039527e-04, reg: 1.528291e-02, (21 / 100)
```

اما بهترین نتیجه‌ی به دست آمده نگران کننده است. چرا؟

٪ **۵۳** – برای یک شبکه ۲ لایه با ۵۰ نورون مخفی نسبتاً نتیجه خوبی است!

بهینه‌سازی ابرپارامترها: جستجوی تصادفی یا منظم؟



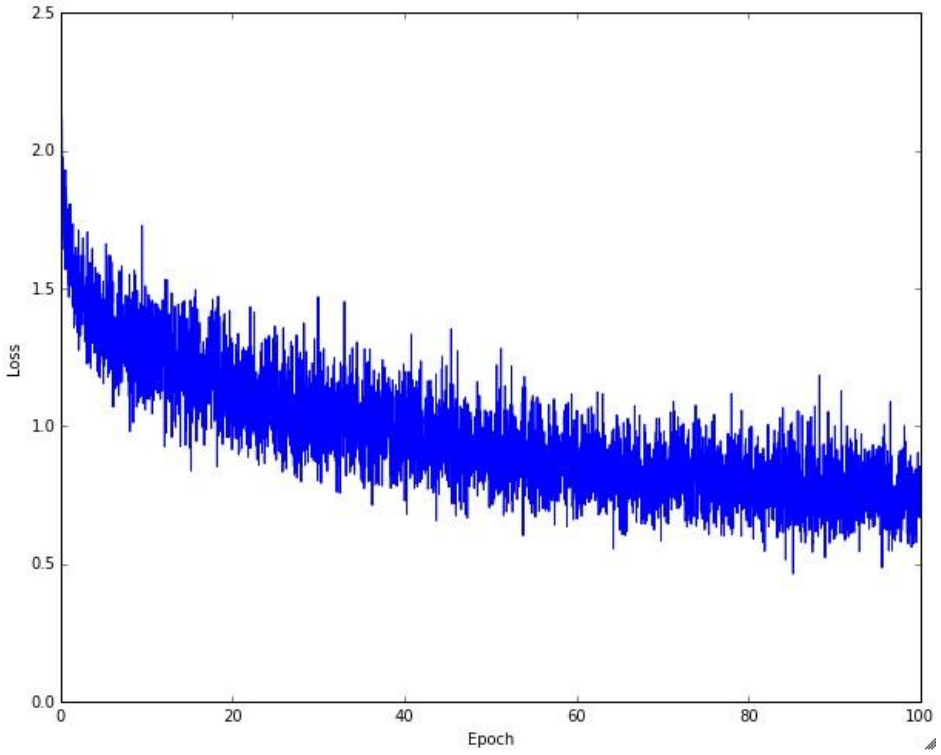


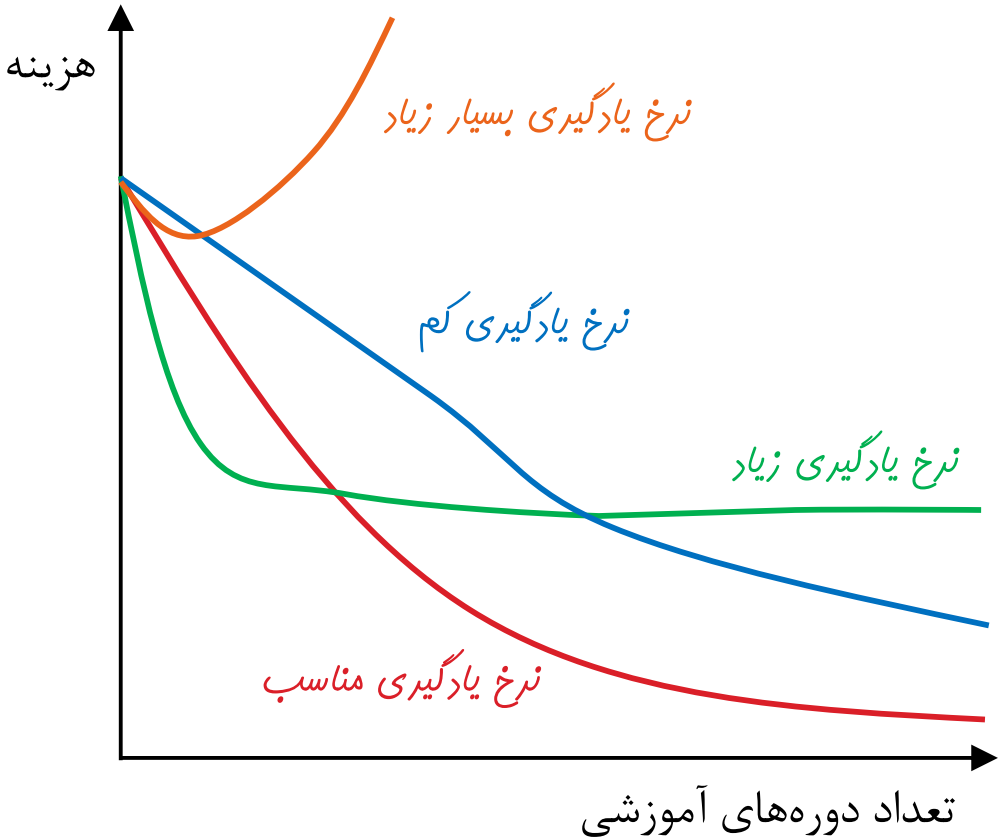
[Bergstra and Bengio, 2012]

بهینه‌سازی ابرپارامترها

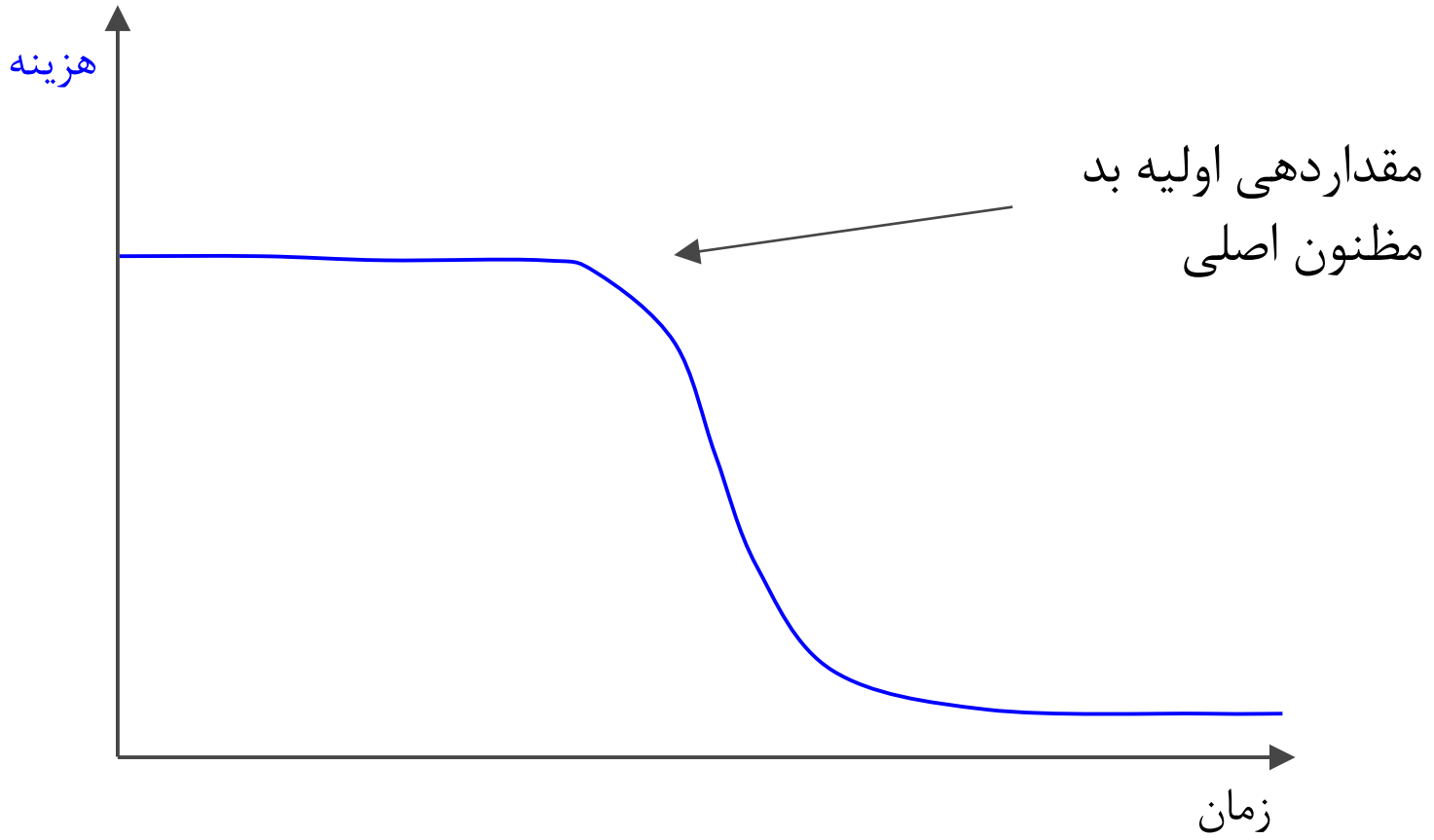
- ابرپارامترهایی که باید برای آنها مقدار تعیین نمود:
  - ساختار شبکه [تعداد و اندازه لایه‌ها]
  - نرخ یادگیری، چگونگی کاهش آن، قاعده به روز رسانی
  - چگونگی تنظیم و شدت تنظیم

نظارت و ترسیم تغییر تابع هزینه



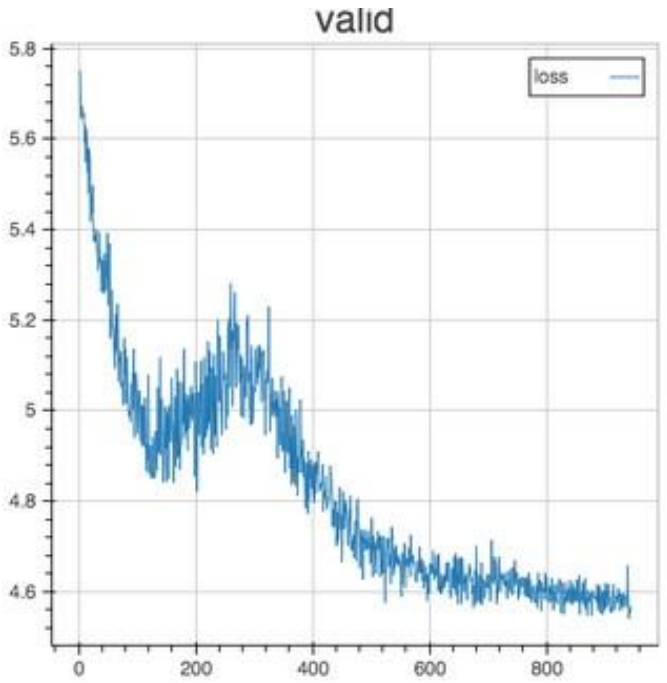


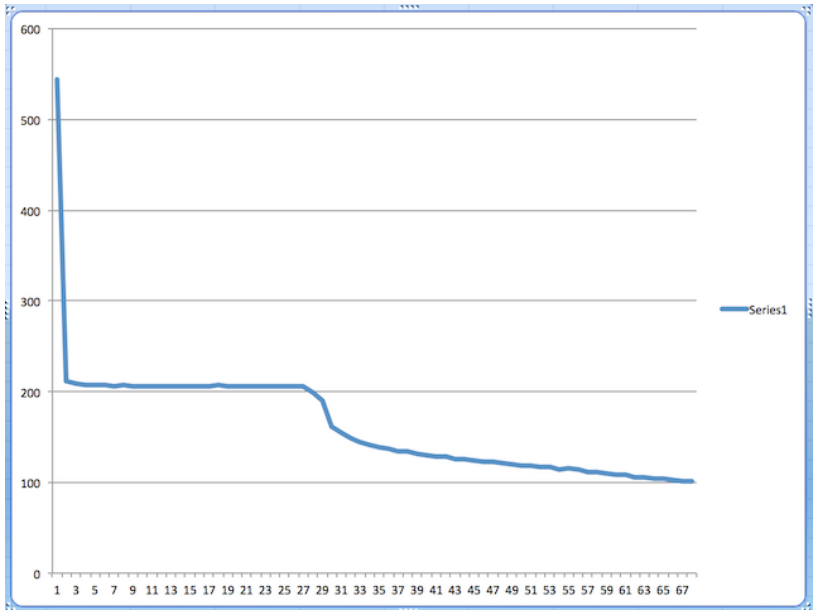
نظارت و ترسیم تغییر تابع هزینه

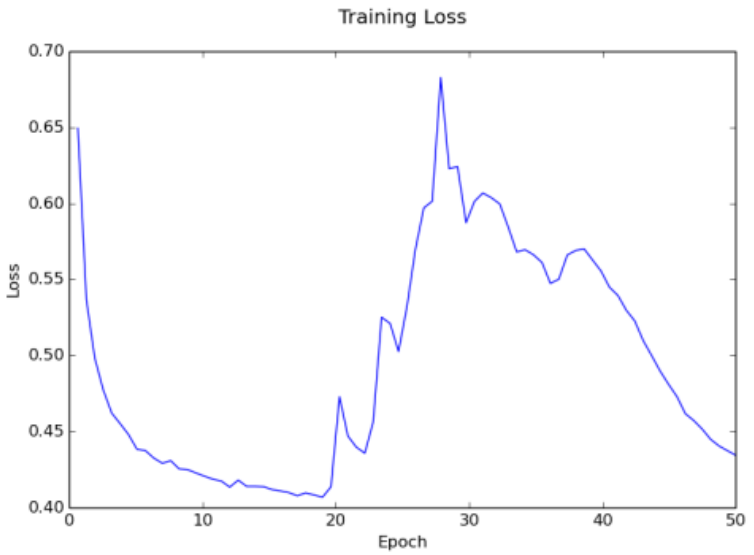


نظارت و ترسیم تغییر تابع هزینه

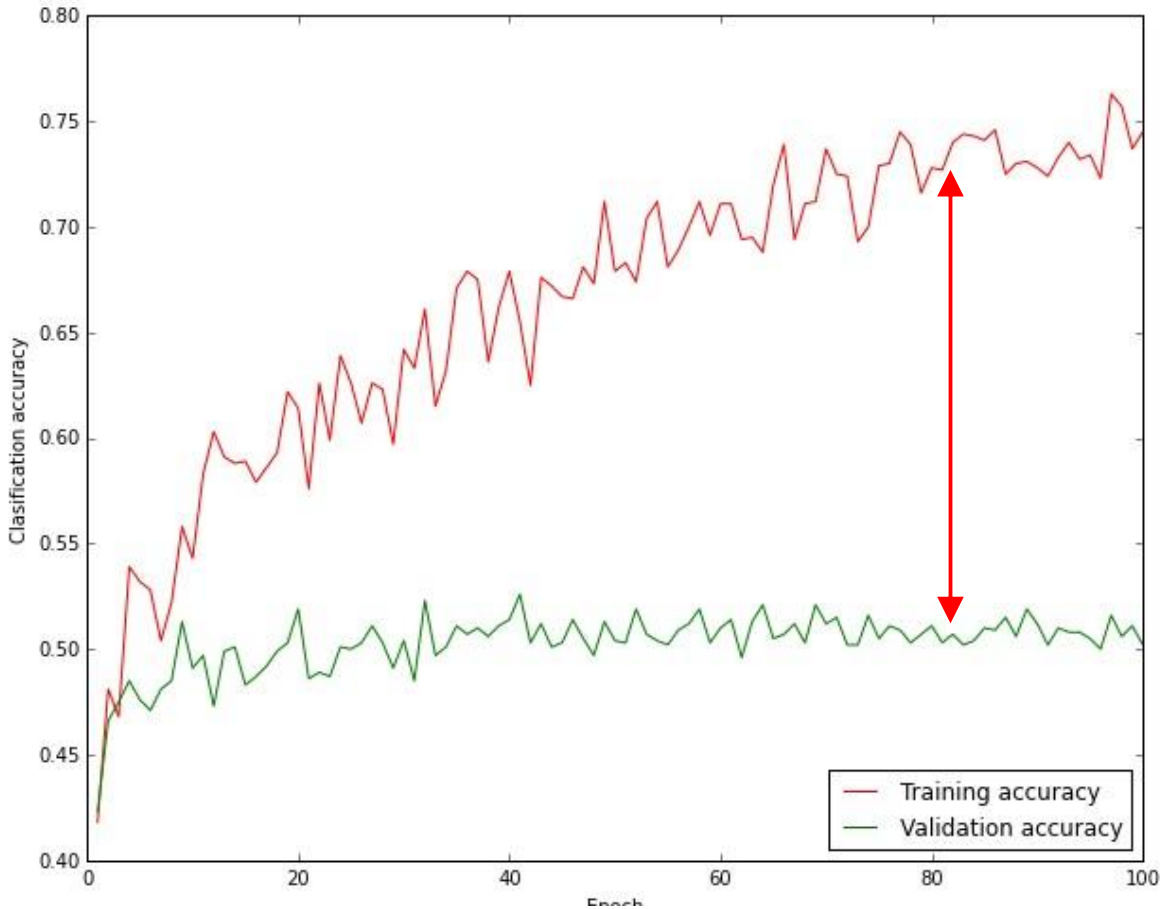
lossfunctions.tumblr.com







نظارت و ترسیم تغییر تابع هزینه



اختلاف زیاد = بیش‌برازش

⇐ افزایش شدت تنظیم؟

عدم اختلاف

⇐ افزایش ظرفیت مدل؟

نسبت تغییر وزن‌ها به اندازه وزن‌ها


```
# assume parameter vector W and its gradient vector dW
param_scale = np.linalg.norm(W.ravel())
update = -learning_rate*dW # simple SGD update
update_scale = np.linalg.norm(update.ravel())
W += update # the actual update
print update_scale / param_scale # want ~1e-3
```

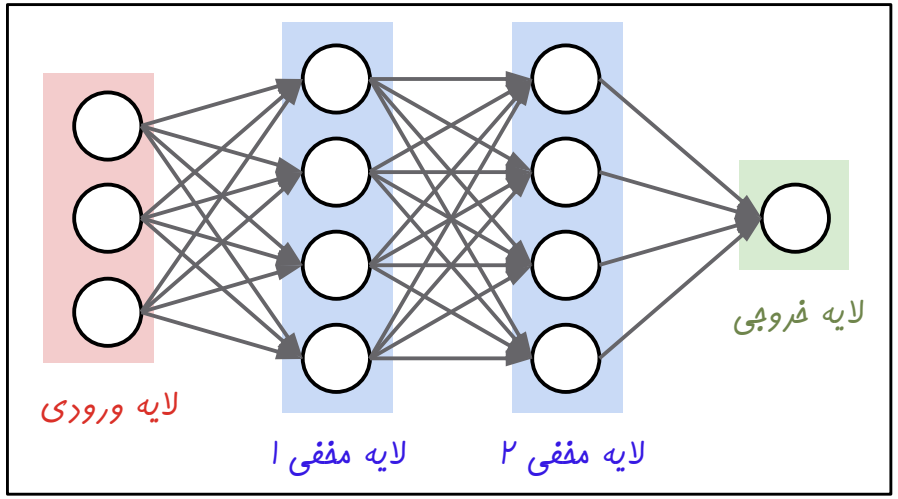
- نسبت مقدار به روز رسانی به مقدار وزن‌ها: `~ 0.0002 / 0.02 = 0.01`
- نسبت مطلوب در حدود ۰/۰۰۱ یا بیشتر است.

جمع‌بندی

- توابع فعالیت. [از ReLU استفاده کنید]
- پیش‌پردازش داده‌ها. [برای تصویر: کم کردن میانگین]
- مقداردهی اولیه به وزن‌ها. [از روش خاویر استفاده کنید]
- مراحل توسعه فرایند آموزش. [پیش‌پردازش، انتخاب معماری، آموزش اولیه، بهینه‌سازی ابرپارامترها]
- بهینه‌سازی ابرپارامترها. [جستجوی تصادفی در فضای لگاریتمی]

یادآوری: بهینه‌سازی

- گرادیان کاهشی. [با دسته‌های کوچک]



حلقه:

۱. یک دسته از داده‌ها را به طور تصادفی **انتخاب** کن.

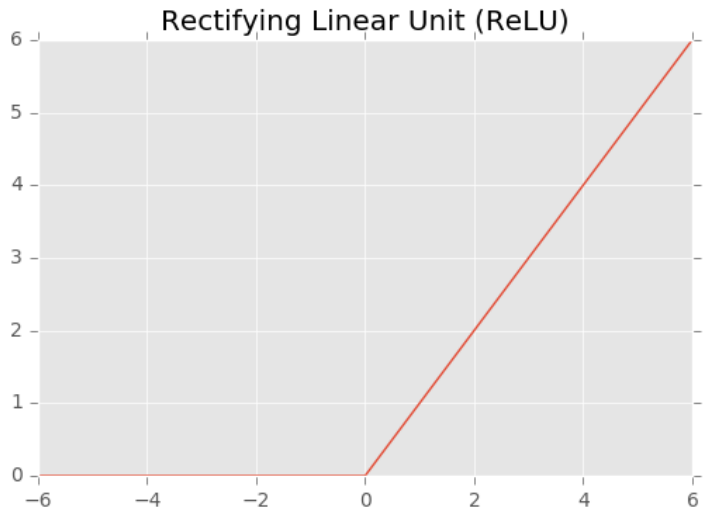
۲. **محاسبات رو به جلو** را در طول گراف انجام بده و مقدار تابع هزینه را محاسبه کن.

۳. **محاسبات رو به عقب** را به منظور محاسبه گرادیان‌ها انجام بده.

۴. مقدار پارامترها را با استفاده از گرادیان‌های محاسبه شده **به روز رسانی** کن.

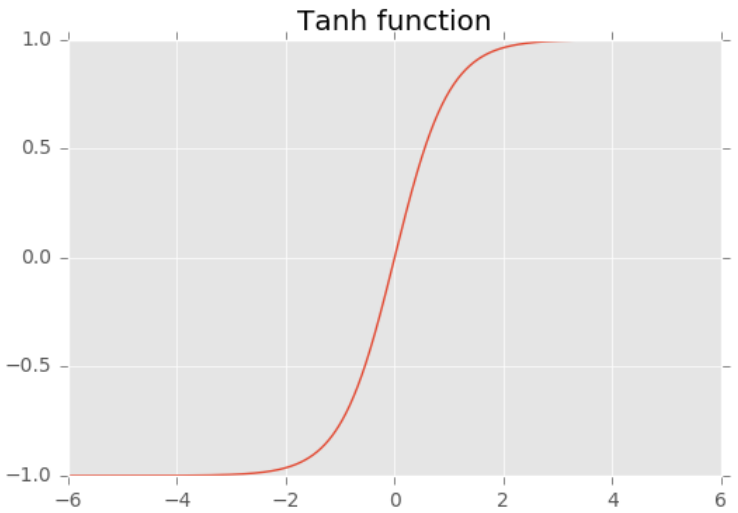
یادآوری: توابع فعالیت

ReLU



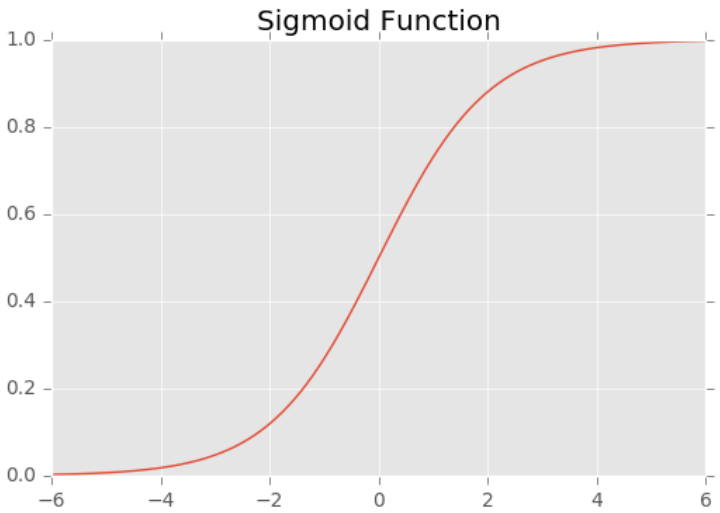
$$\max(0, x)$$

تانژانت هایپربولیک



$$\tanh(x)$$

سیگموید

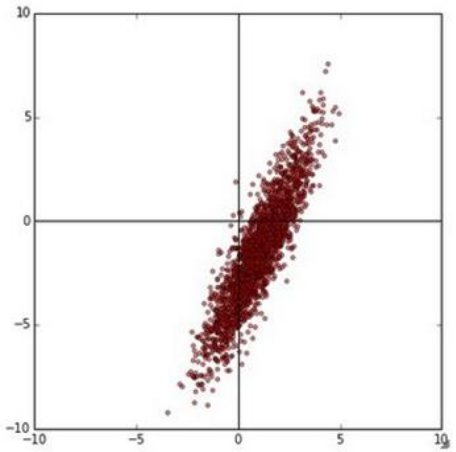


$$\sigma(x) = 1/(1 + e^{-x})$$

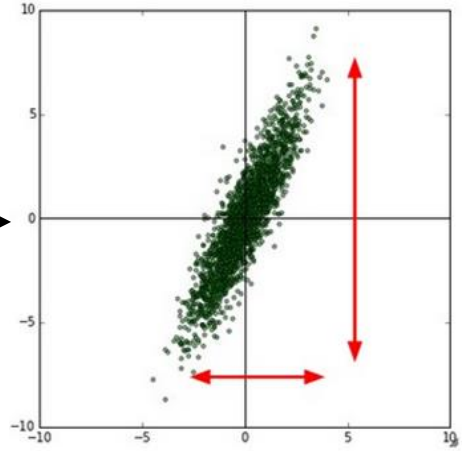
یادآوری: پیش‌پردازش داده‌ها

- پیش‌پردازش داده‌ها.

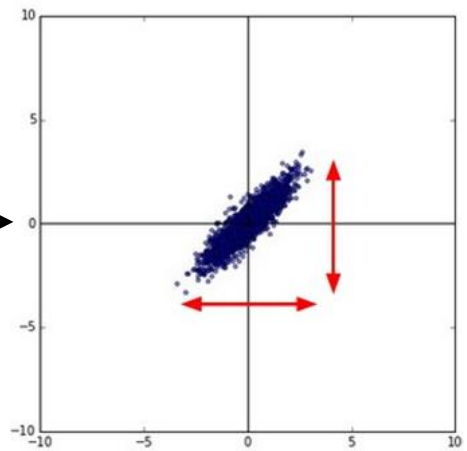
داده‌های اصلی



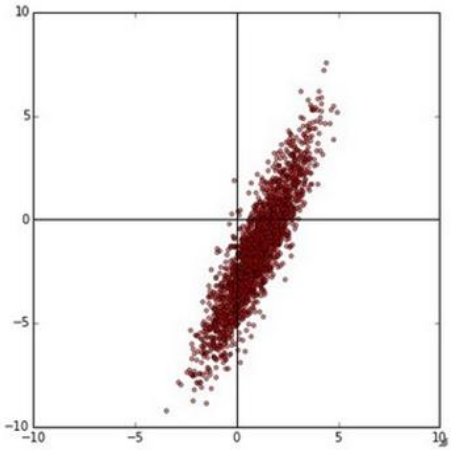
داده‌ها با میانگین صفر



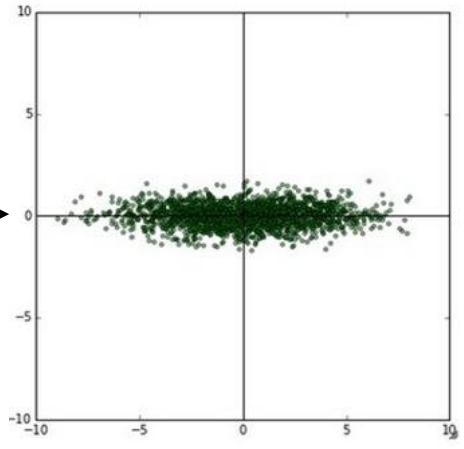
داده‌های نرمال شده



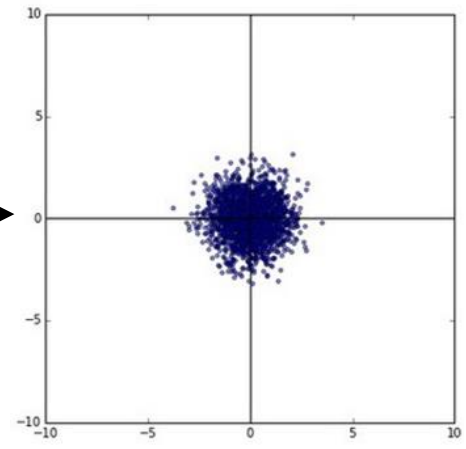
داده‌های اصلی



داده‌های ناهمبسته شده



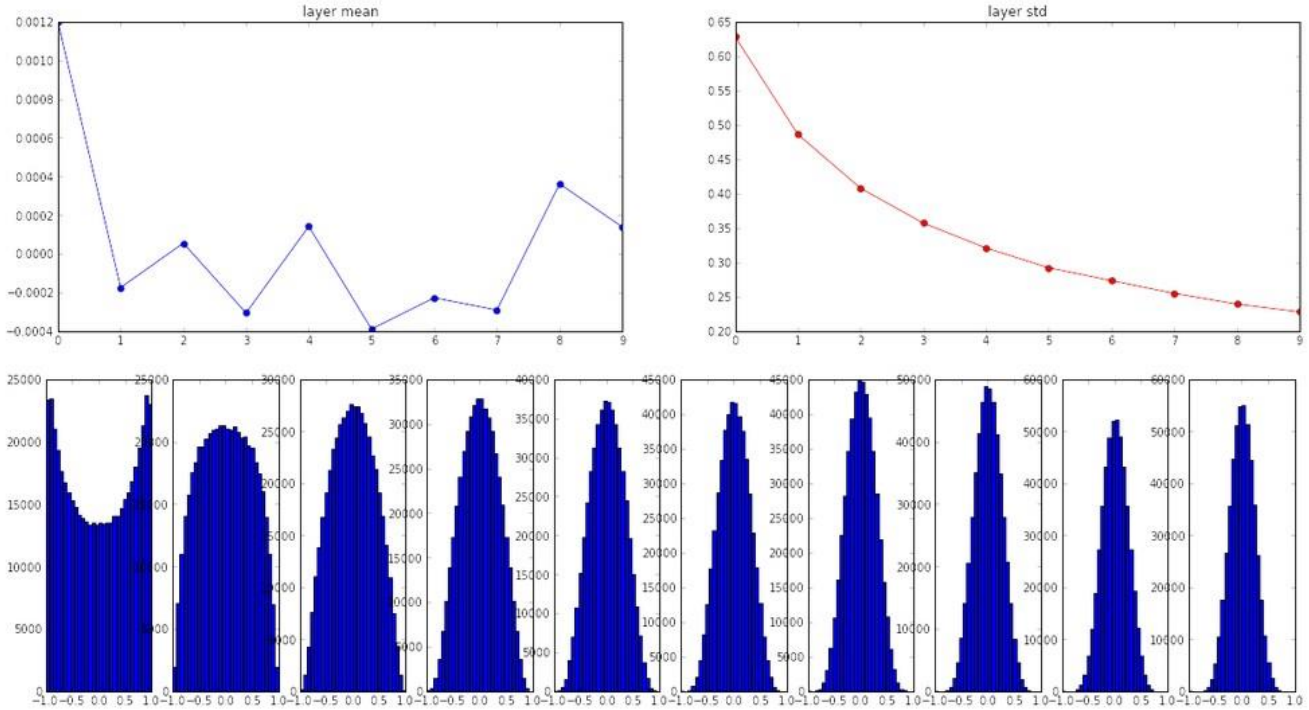
داده‌های سفید شده



یادآوری: مقداردهی اولیه به وزن‌ها

- وزن‌دهی خاویر. [Gloret et al., 2010]

```
input layer had mean 0.001800 and std 1.001311
hidden layer 1 had mean 0.001198 and std 0.627953
hidden layer 2 had mean -0.000175 and std 0.486051
hidden layer 3 had mean 0.000055 and std 0.407723
hidden layer 4 had mean -0.000306 and std 0.357108
hidden layer 5 had mean 0.000142 and std 0.320917
hidden layer 6 had mean -0.000389 and std 0.292116
hidden layer 7 had mean -0.000228 and std 0.273387
hidden layer 8 had mean -0.000291 and std 0.254935
hidden layer 9 had mean 0.000361 and std 0.239266
hidden layer 10 had mean 0.000139 and std 0.228008
```




```
W = np.random.randn(fan_in, fan_out) / np.sqrt(fan_in)
```

گام چهارم: آموزش مقدماتی

- به طور معمول، با یک ضریب تنظیم بسیار کوچک شروع می‌کنیم.
- سعی می‌کنیم یک مقدار برای نرخ یادگیری بیابیم، به گونه‌ای که باعث شود مقدار تابع هزینه در هر تکرار کاهش یابد.

```
model = init_two_layer_model(32*32*3, 50, 10) # input size, hidden size, number of classes
trainer = ClassifierTrainer()
best_model, stats = trainer.train(X_train, y_train, X_val, y_val,
                                  model, two_layer_net,
                                  num_epochs=10, reg=0.000001,
                                  update='sgd', learning_rate_decay=1,
                                  sample_batches = True,
                                  learning_rate=1e-6, verbose=True)
```

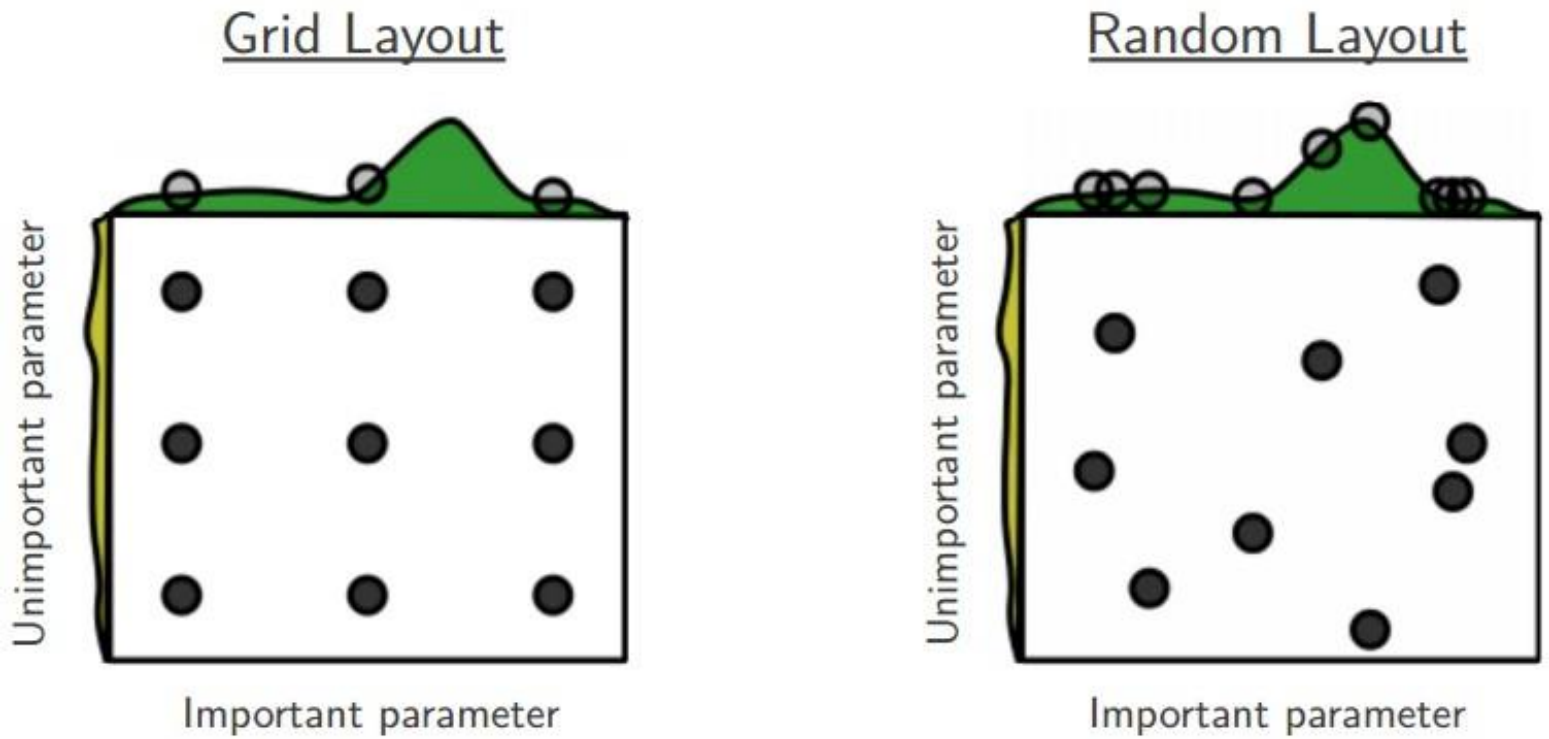
```
Finished epoch 1 / 10: cost 2.302576, train: 0.080000, val 0.103000, lr 1.000000e-06
Finished epoch 2 / 10: cost 2.302582, train: 0.121000, val 0.124000, lr 1.000000e-06
Finished epoch 3 / 10: cost 2.302558, train: 0.119000, val 0.138000, lr 1.000000e-06
Finished epoch 4 / 10: cost 2.302519, train: 0.127000, val 0.151000, lr 1.000000e-06
Finished epoch 5 / 10: cost 2.302517, train: 0.158000, val 0.171000, lr 1.000000e-06
Finished epoch 6 / 10: cost 2.302518, train: 0.179000, val 0.172000, lr 1.000000e-06
Finished epoch 7 / 10: cost 2.302466, train: 0.180000, val 0.176000, lr 1.000000e-06
Finished epoch 8 / 10: cost 2.302452, train: 0.175000, val 0.185000, lr 1.000000e-06
Finished epoch 9 / 10: cost 2.302459, train: 0.206000, val 0.192000, lr 1.000000e-06
Finished epoch 10 / 10: cost 2.302420, train: 0.190000, val 0.192000, lr 1.000000e-06
finished optimization. best validation accuracy: 0.192000
```

مقدار تابع هزینه بسیار کم تغییر می‌کند!

کاهش نیافتن تابع هزینه:

نرخ یادگیری بیش از حد کوچک است!

یادآوری: بهینه‌سازی ابرپارامترها (جستجوی تصادفی یا منظم)



[Bergstra and Bengio, 2012]

فهرست مطالب

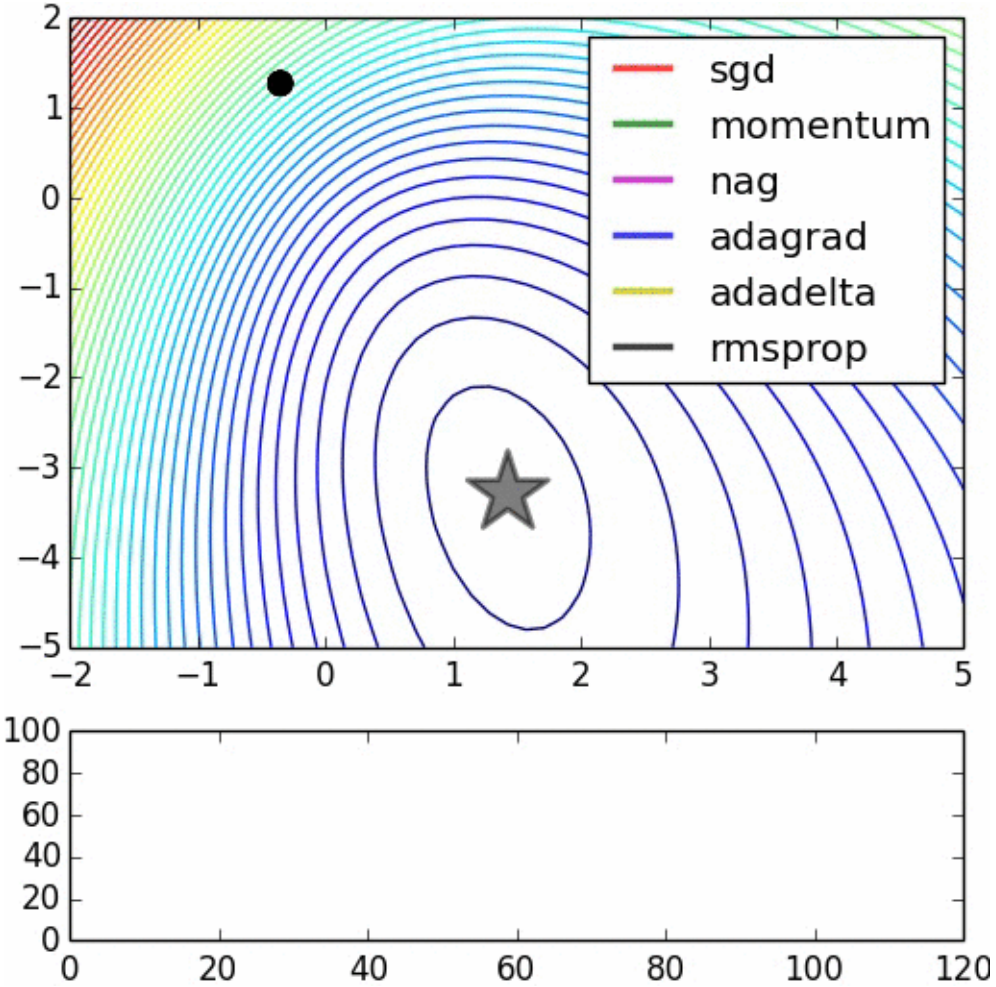
به روز رسانی پارامترها

آموزش یک شبکه عصبی: حلقه اصلی


```
while True:
    data_batch = dataset.sample_data_batch()
    loss = network.forward(data_batch)
    dx = network.backward()
    x += -learning_rate * dx
```

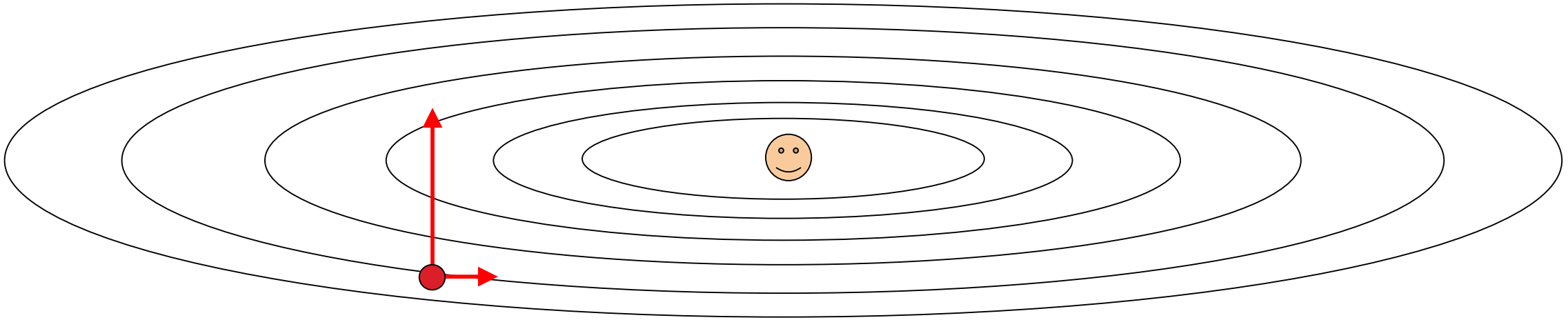
قاعده به روز رسانی: گرادیان کاهشی ساده

به روز رسانی پارامترها



به روز رسانی پارامترها

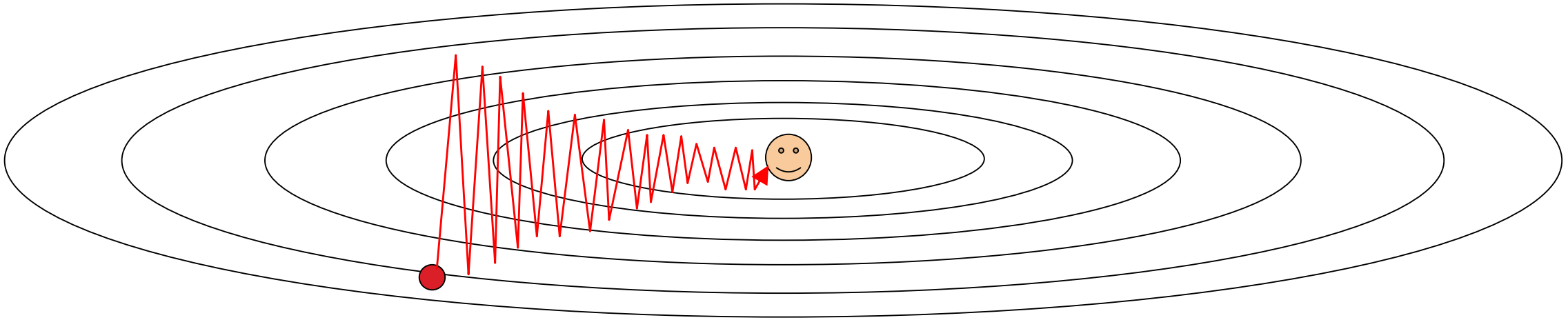
- تابع هزینه. در جهت عمودی دارای شیب تند و در جهت افقی دارای شیب ملایم.



در صورت استفاده از گرادیان کاهشی، مسیر همگرایی به سمت نقطه کمینه چگونه خواهد بود؟

به روز رسانی پارامترها

- تابع هزینه. در جهت عمودی دارای شیب تند و در جهت افقی دارای شیب ملایم.



در جهت افقی پیشروی بسیار آهسته و در جهت عمودی دارای حرکات نامنظم خواهد بود!

روش ممنتوم

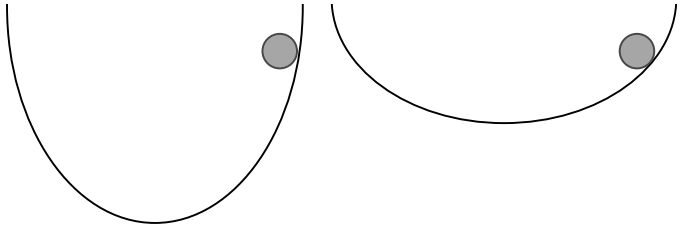

```
# Gradient descent update
x += -learning_rate * dx
```

```
# Momentum update
v = mu * v - learning_rate * dx  # integrate velocity
x += v  # integrate position
```

- تفسیر فیزیکی. به عنوان یک توپ که بر روی سطح تابع هزینه به سمت پایین حرکت می‌کند + اصطکاک (ضریب mu)
  - مقدار ضریب mu: معمولاً ۰/۵، ۰/۹ یا ۰/۹۹

روش ممنتوم

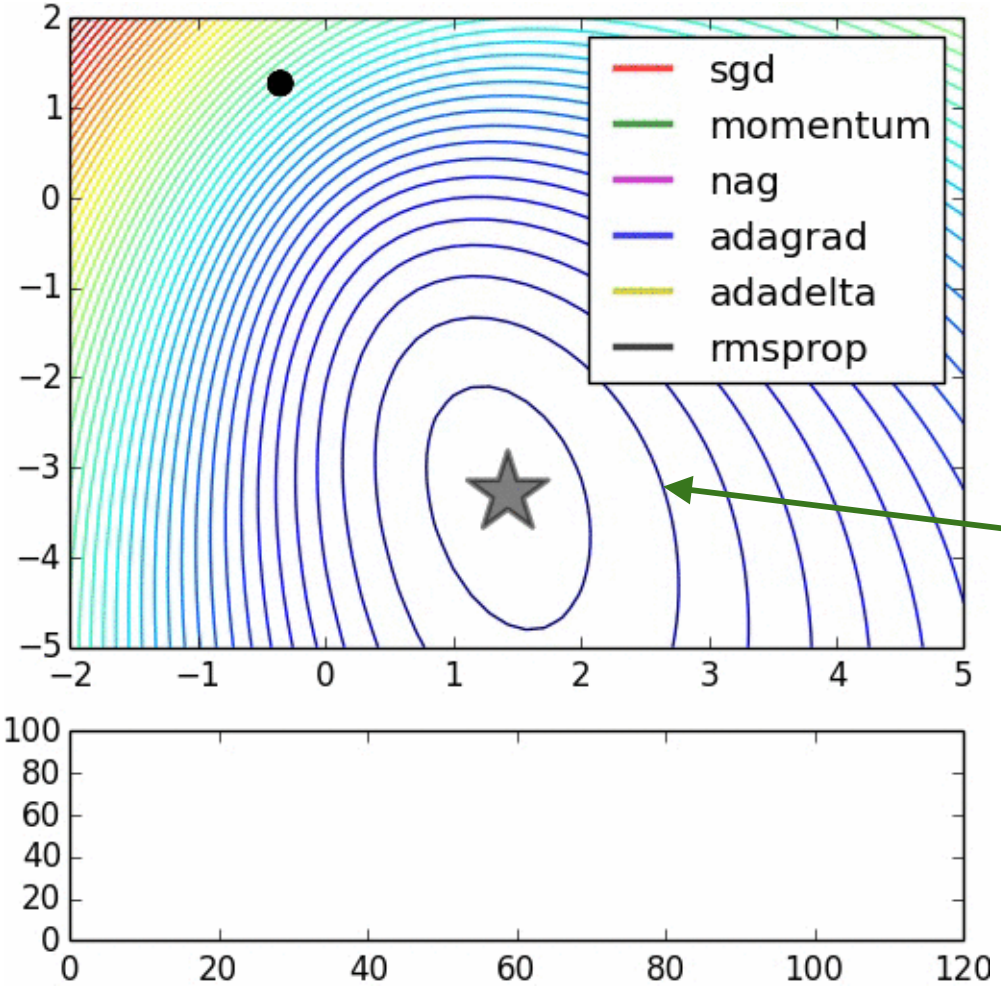

```
# Gradient descent update
x += -learning_rate * dx
```



```
# Momentum update
v = mu * v - learning_rate * dx  # integrate velocity
x += v  # integrate position
```

- در جهت‌های پرشیب به دلیل تغییر سریع علامت، به تدریج سرعت کاهش می‌یابد.
- در جهت‌های کم‌شیب، به دلیل هم جهت بودن بردار سرعت و بردار گریان، به تدریج سرعت افزایش می‌یابد.

روش ممنتوم



توجه کنید که در روش ممنتوم اگرچه از هدف عبور می‌کنیم اما این روش در کل نسبت به گرادیان کاهشی سریع‌تر همگرا می‌شود.

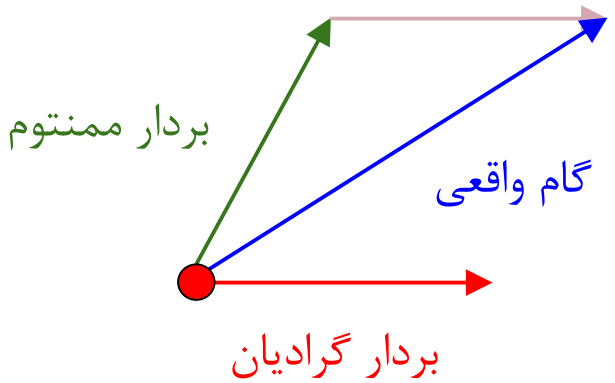
به روز رسانی مقدار پارامترها: روش‌های دیگر

- روش‌های مرتبه اول
  - روش ممنتوم نستروف
  - روش AdaGrad
  - روش RMSProp
  - روش Adam
- روش‌های مرتبه دوم
  - روش نیوتن
  - روش‌های شبه نیوتنی (BFGS و L-BFGS)

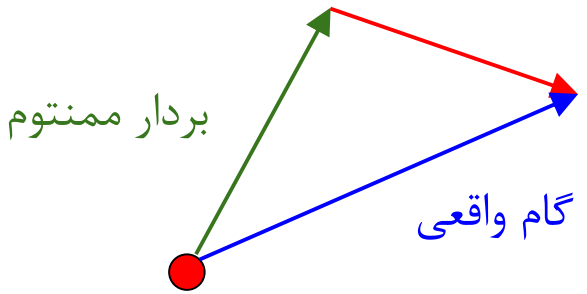
روش ممنتوم «نستروف»


```
# Momentum update
v = mu * v - learning_rate * dx  # integrate velocity
x += v  # integrate position
```

روش ممنتوم معمولی



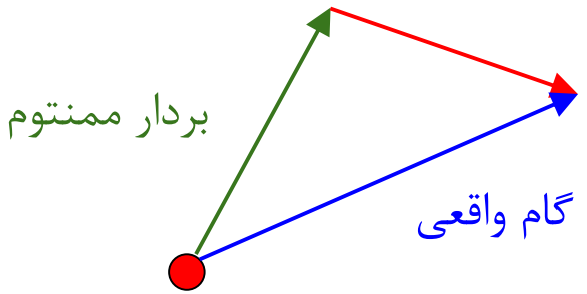
روش ممنتوم نستروف



گرادیان «پیش‌بینی شده» (اندکی متفاوت با گرادیان اصلی)

روش ممنتوم «نستروف»

روش ممنتوم نستروف



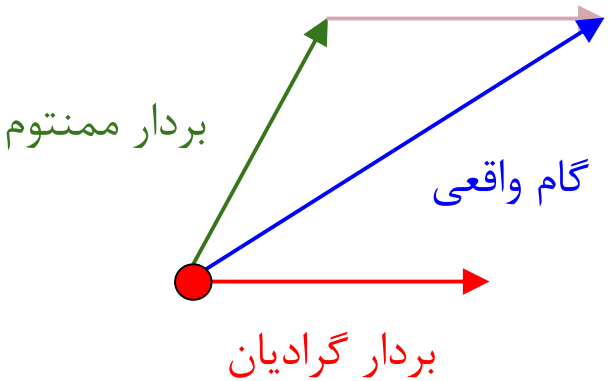
گرادیان «پیش‌بینی شده» (اندکی متفاوت با گرادیان اصلی)

نستروف: تنها تفاوت ...

$$v_t = \mu v_{t-1} - \epsilon \nabla f(\theta_{t-1} + \mu v_{t-1})$$

$$\theta_t = \theta_{t-1} + v_t$$

روش ممنتوم معمولی



روش ممنتوم «نستروف»

$$v_t = \mu v_{t-1} - \epsilon \nabla f(\theta_{t-1} + \mu v_{t-1})$$

$$\theta_t = \theta_{t-1} + v_t$$

نسبتاً نامناسب ...

به طور معمول بردارهای زیر را داریم:

$$\theta_{t-1}, \nabla f(\theta_{t-1})$$

با یک تغییر متغیر و بازنویسی مشکل حل می‌شود:

$$\phi_{t-1} = \theta_{t-1} + \mu v_{t-1}$$

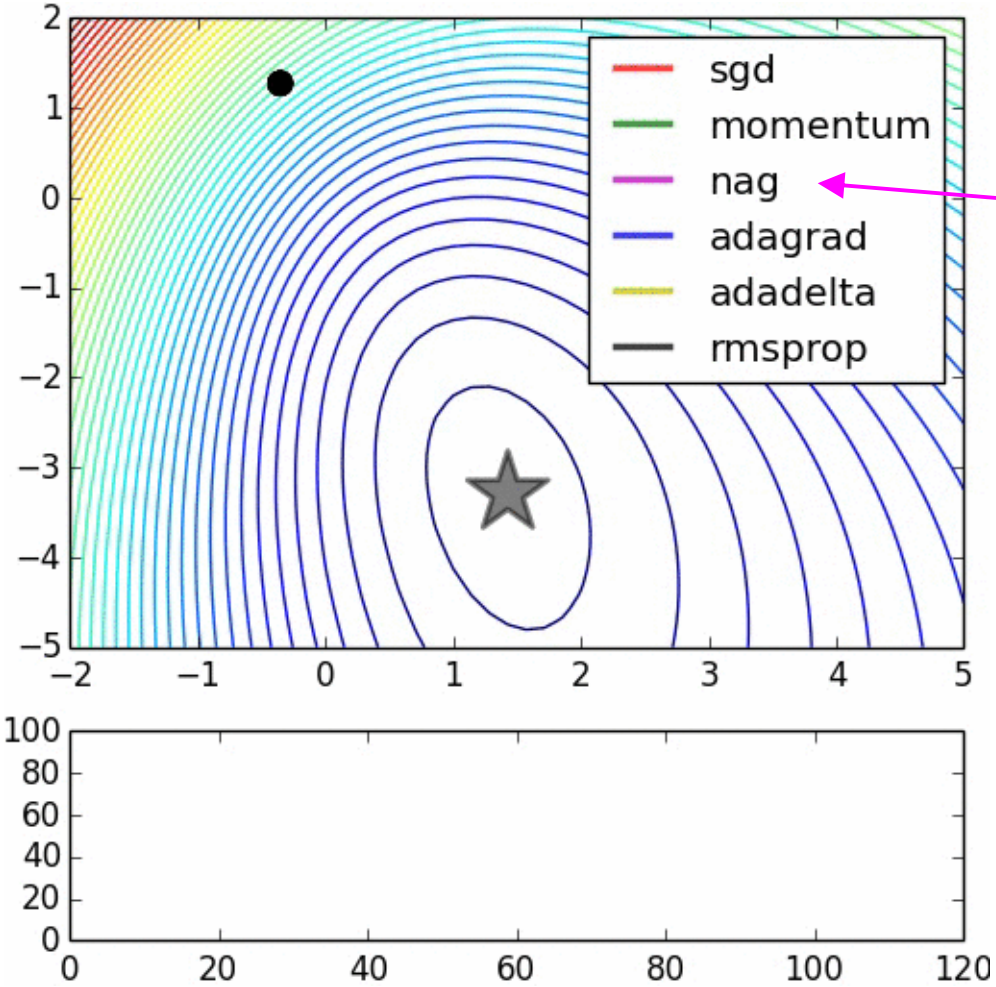
$$v_t = \mu v_{t-1} - \epsilon \nabla f(\phi_{t-1})$$

$$\phi_t = \phi_{t-1} - \mu v_{t-1} + (1 + \mu) v_t$$

$$\begin{aligned}\phi_t &= \theta_t + \mu v_t = \theta_{t-1} + v_t + \mu v_t \\ &= \phi_{t-1} - \mu v_{t-1} + (1 + \mu) v_t\end{aligned}$$

```
# Nesterov Momentum update rewrite
v_prev = v
v = mu * v - learning_rate * dx
x += -mu * v_prev + (1 + mu) * v
```

روش ممنتوم «نستروف»



Nesterov Accelerated Gradient

روش AdaGrad

- مقیاس‌بندی گرادیان‌ها در هر بعد بر اساس مجموع مربعات مقادیر قبلی گرادیان در همان بعد.

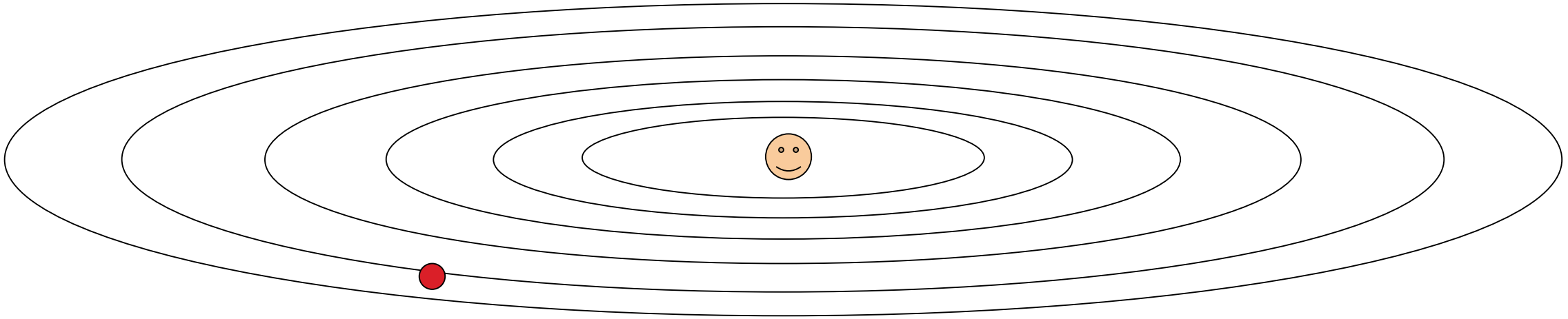
```
# AdaGrad update
cache += dx ** 2
x += -learning_rate * dx / (np.sqrt(cache) + 1e-7)
```

[Duchi et al., 2011]

روش AdaGrad

- در صورت استفاده از این روش، همگرایی چگونه خواهد بود؟

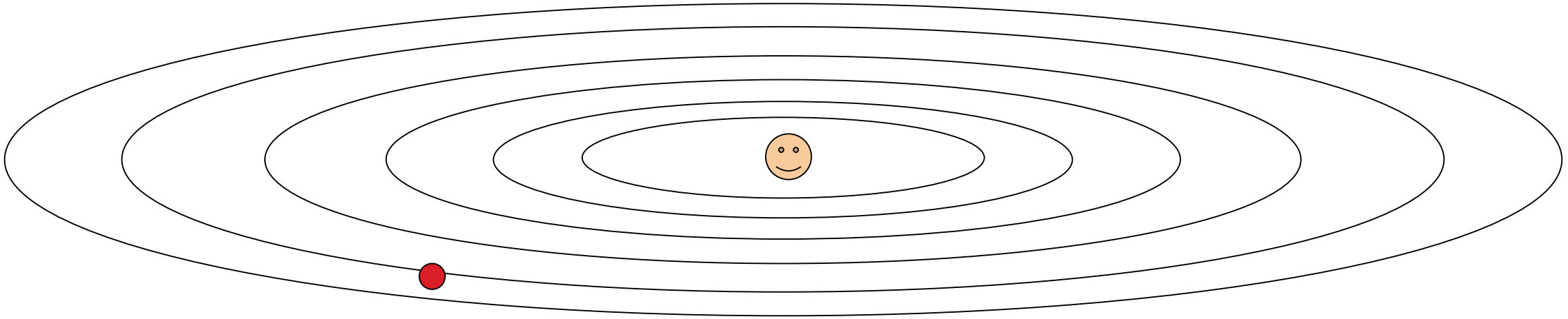
```
# AdaGrad update
cache += dx ** 2
x += -learning_rate * dx / (np.sqrt(cache) + 1e-7)
```



روش AdaGrad

- در یک مدت زمان طولانی برای نرخ یادگیری چه اتفاقی می‌افتد؟

```
# AdaGrad update
cache += dx ** 2
x += -learning_rate * dx / (np.sqrt(cache) + 1e-7)
```



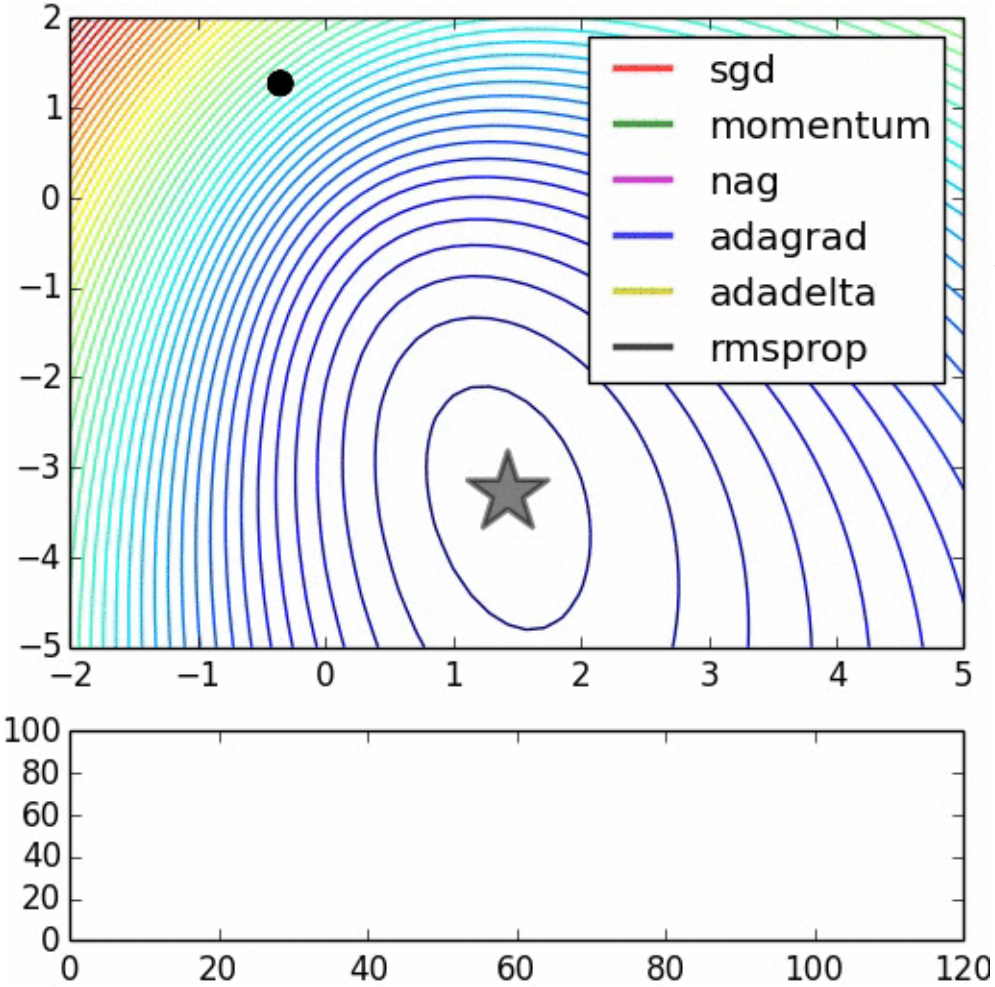
روش RMSProp


```
# AdaGrad update
cache += dx ** 2
x += -learning_rate * dx / (np.sqrt(cache) + 1e-7)
```

```
# RMSProp update
cache = decay_rate * cache + (1 - decay_rate) * dx ** 2
x += -learning_rate * dx / (np.sqrt(cache) + 1e-7)
```

[Tieleman and Hinton, 2012]

روش‌های AdaGrad و RMSProb



adagrad
rmsprop

روش Adam


```
# Adam update
m = beta1 * m + (1 - beta1) * dx          # first moment
v = beta2 * v + (1 - beta2) * dx ** 2  # second moment
x += -learning_rate * m / (np.sqrt(v) + 1e-7)
```

- تا حدی شبیه به روش RMSProp به همراه ممنتوم

```
# RMSProp update
cache = decay_rate * cache + (1 - decay_rate) * dx ** 2
x += -learning_rate * dx / (np.sqrt(cache) + 1e-7)
```

[Kingma and Ba, 2014]

روش Adam

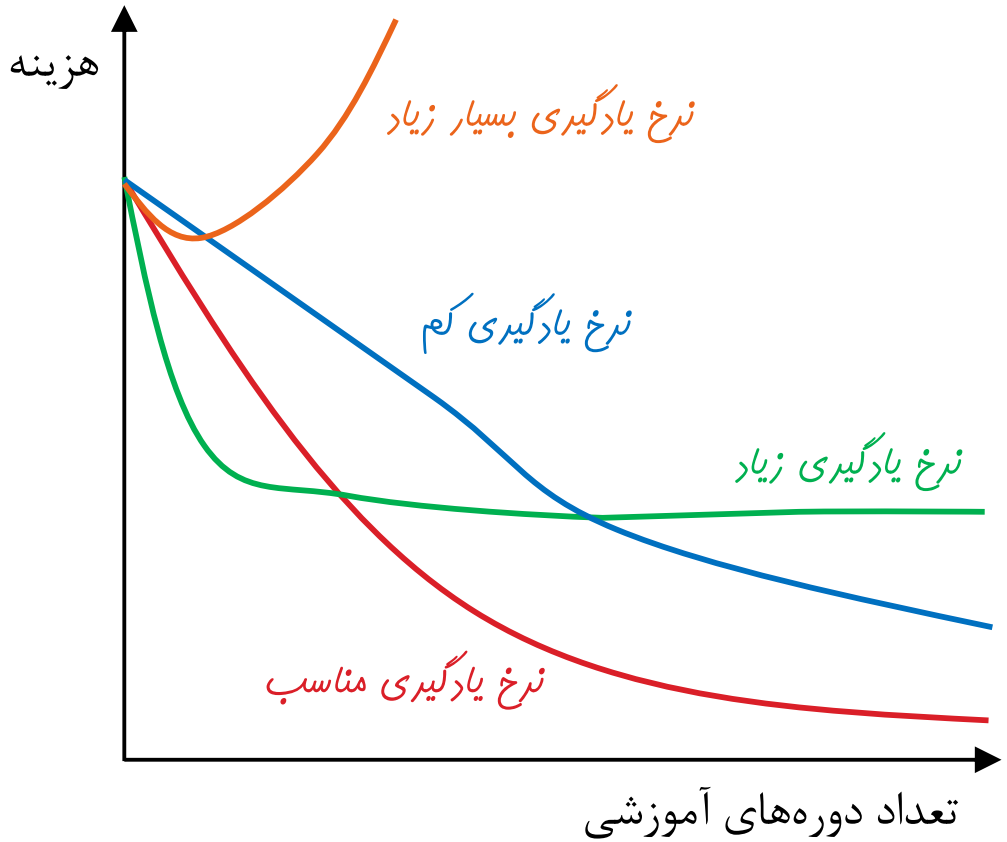

```
# Adam update
m, v =  #... initialize caches to zeros
for t in xrange(1, big_number):
    dx =  #... evaluate gradient
    m  = beta1 * m + (1 - beta1) * dx         # first moment
    v  = beta2 * v + (1 - beta2) * dx ** 2    # second moment
    mb = m / (1 - beta1 ** t)  # correct bias
    vb = v / (1 - beta2 ** t)  # correct bias
    x += -learning_rate * mb / (np.sqrt(vb) + 1e-7)
```

- دلیل تصحیح بایاس‌ها این است که m و v در ابتدا صفر هستند و مدتی زمان نیاز است تا به مقدار مناسبی برسند.

[Kingma and Ba, 2014]

نرخ یادگیری

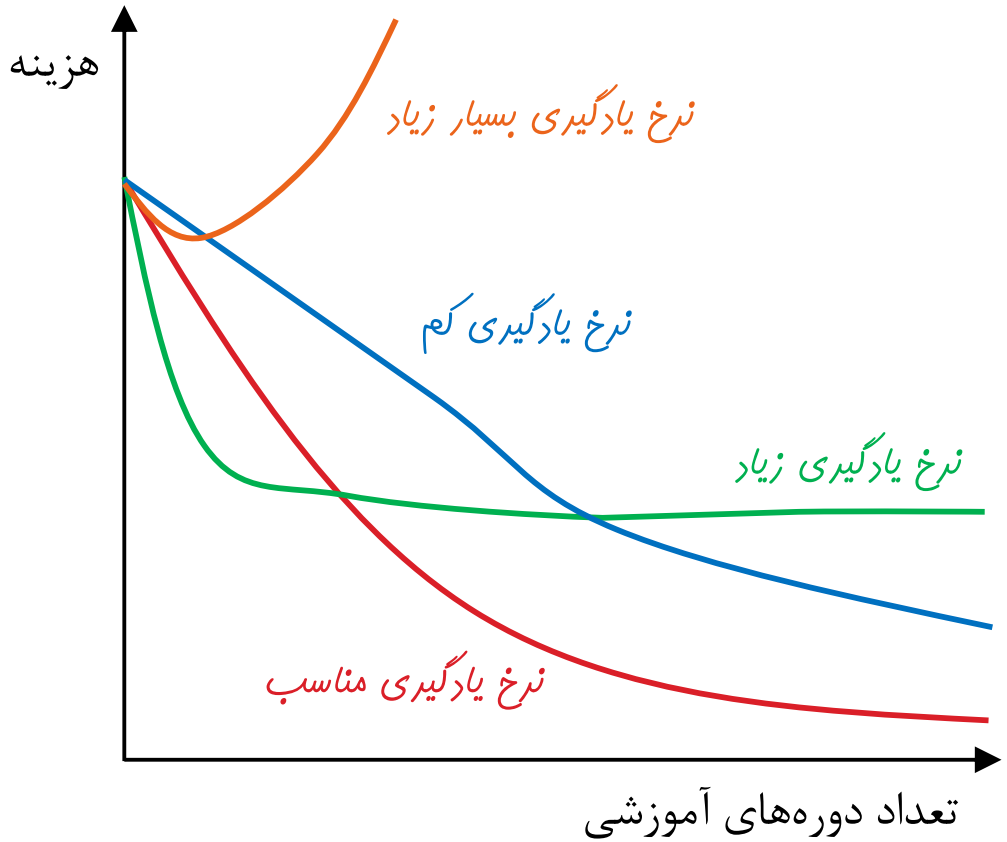
- در تمام روش‌های به روز رسانی بیان شده، مقدار **نرخ یادگیری** به عنوان یک ابرپارامتر باید به دقت تعیین شود.



بهترین مقدار برای نرخ یادگیری کدام است؟

نرخ یادگیری

- در تمام روش‌های به روز رسانی بیان شده، مقدار **نرخ یادگیری** به عنوان یک ابرپارامتر باید به دقت تعیین شود.



⇐ کاهش نرخ یادگیری در طول زمان!

کاهش مرحله‌ای:

مثلاً پس از هر چند دوره‌ی آموزش نرخ یادگیری را نصف کن.

کاهش نمایی:

$$\alpha = \alpha_0 e^{-kt}$$

کاهش متناسب با معکوس t:

$$\alpha = \alpha_0 / (1 + kt)$$

روش‌های بهینه‌سازی درجه دوم

- بسط تیلور مرتبه دوم.

$$J(\boldsymbol{\theta}) \approx J(\boldsymbol{\theta}_0) + (\boldsymbol{\theta} - \boldsymbol{\theta}_0)^T \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_0) + \frac{1}{2}(\boldsymbol{\theta} - \boldsymbol{\theta}_0)^T \boldsymbol{H} (\boldsymbol{\theta} - \boldsymbol{\theta}_0)$$

- پس از حل این رابطه برای یافتن نقطه‌ی بحرانی، روش به روز رسانی نیوتون را به دست می‌آوریم:

$$\boldsymbol{\theta}^* = \boldsymbol{\theta}_0 - \boldsymbol{H}^{-1} \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_0)$$

محاسبه‌ی مشتق و برابر قرار دادن آن با صفر

این روش به روز رسانی چه ویژگی جالبی دارد؟

روش‌های بهینه‌سازی درجه دوم

- بسط تیلور مرتبه دوم.

$$J(\boldsymbol{\theta}) \approx J(\boldsymbol{\theta}_0) + (\boldsymbol{\theta} - \boldsymbol{\theta}_0)^T \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_0) + \frac{1}{2}(\boldsymbol{\theta} - \boldsymbol{\theta}_0)^T \boldsymbol{H} (\boldsymbol{\theta} - \boldsymbol{\theta}_0)$$

- پس از حل این رابطه برای یافتن نقطه‌ی بحرانی، روش به روز رسانی نیوتون را به دست می‌آوریم:

$$\boldsymbol{\theta}^* = \boldsymbol{\theta}_0 - \boldsymbol{H}^{-1} \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_0)$$

توجه: هیچ ابرپارامتری وجود ندارد! (مانند نرخ یادگیری)

چرا در عمل استفاده از این روش برای آموزش شبکه‌های عصبی عمیق ممکن نیست؟

روش‌های بهینه‌سازی درجه دوم

$$\boldsymbol{\theta}^* = \boldsymbol{\theta}_0 - \boldsymbol{H}^{-1} \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_0)$$

- روش‌های شبه نیوتونی. [BFGS پرطرفدارترین]

به جای محاسبه‌ی معکوس ماتریس هسین، معکوس ماتریس هسین را در طول زمان و با به روز رسانی‌های مرتبه اول تقریب بزن. [پیچیدگی زمانی درجه دوم به جای پیچیدگی زمانی درجه سوم]

- روش L-BFGS. [BFGS با حافظه‌ی محدود]

تمام ماتریس هسین را ذخیره (ایجاد) نکن.

روش‌های بهینه‌سازی درجه دوم

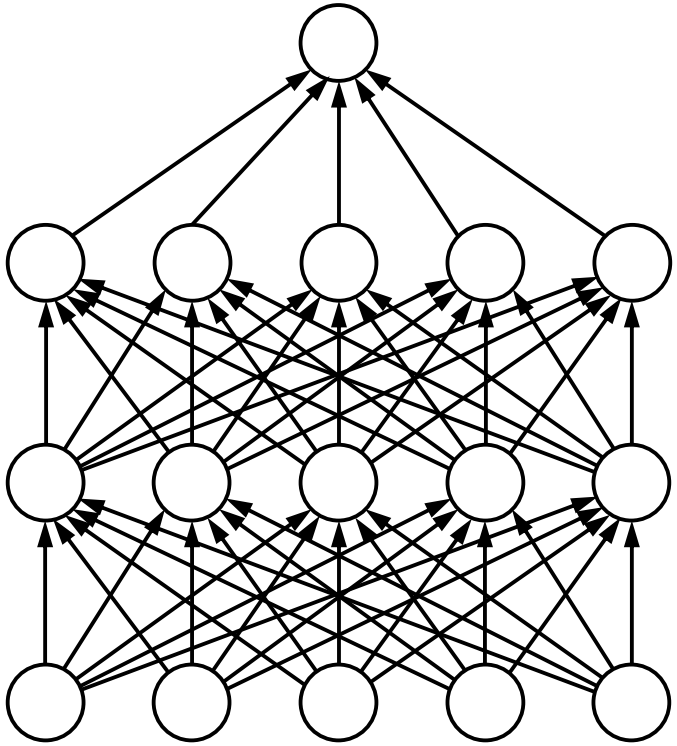
در عمل:

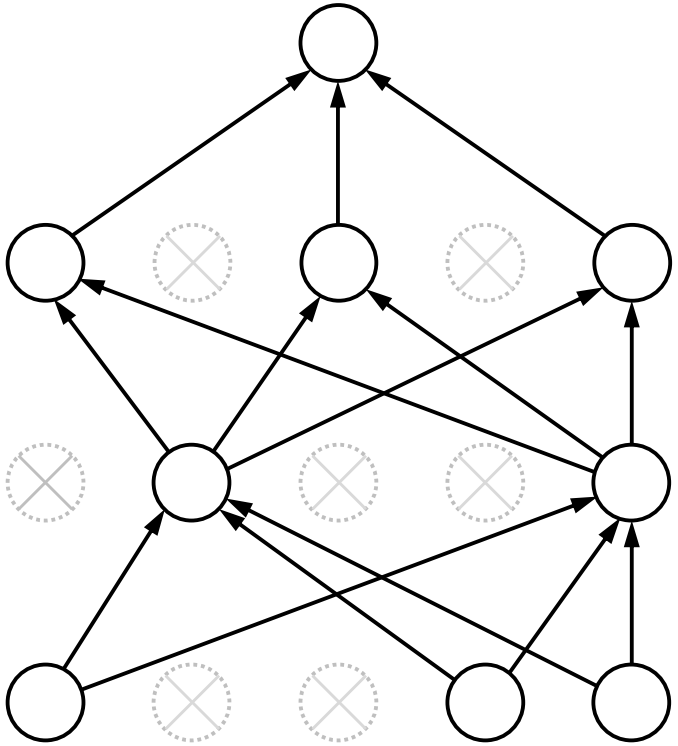
- روش Adam به عنوان انتخاب پیش‌فرض، در بیشتر موارد به خوبی کار می‌کند.
- اگر امکان به روز رسانی با استفاده از تمام داده‌ها به طور کامل در هر تکرار برای شما وجود دارد، روش L-BFGS را امتحان کنید. [در این صورت یادتان باشد که باید تمامی منابع نویز را غیر فعال کنید]

تنظیم: حذف تصادفی

تنظیم: حذف تصادفی [Dropout]

- در محاسبات رو به جلو، به صورت تصادفی خروجی بعضی از نورون‌ها را صفر کن.

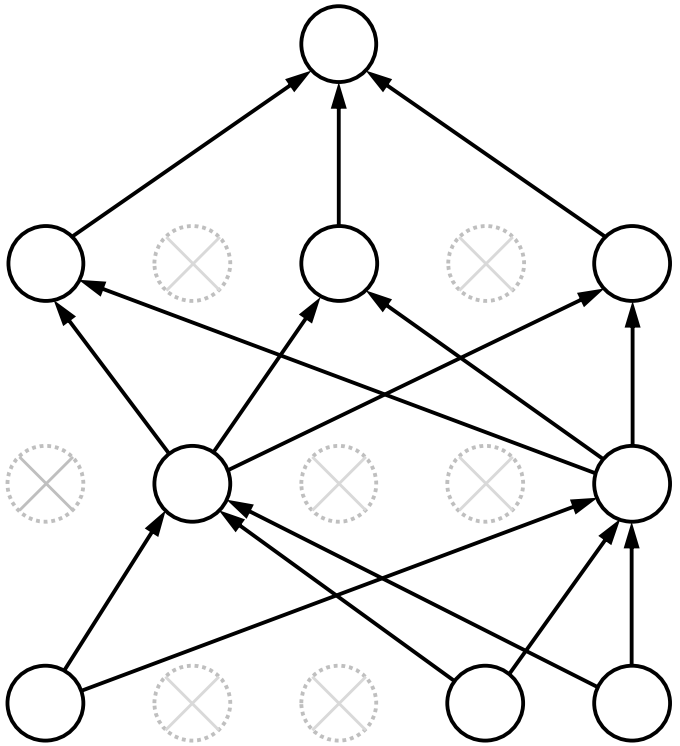




[Srivastava et al., 2014]

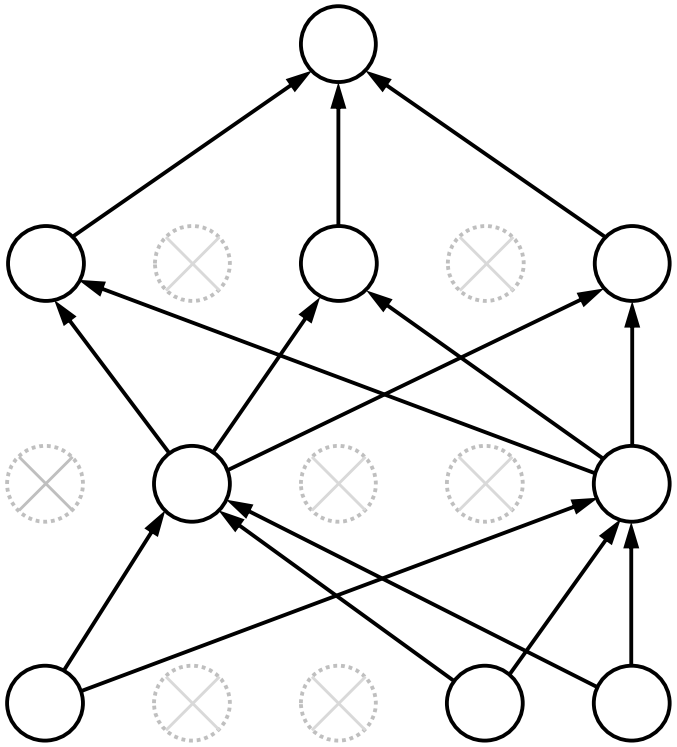
تنظیم: حذف تصادفی


```
p = 0.5  # probability of keeping a unit active

def train_step(X):
    """ X contains the data """

    # forward pass for a 3-layer neural network
    H1 = np.maximum(0, np.dot(W1, X) + b1)
    U1 = np.random.rand(*H1.shape) < p  # first dropout mask
    H1 *= U1  # drop!

    H2 = np.maximum(0, np.dot(W2, H1) + b2)
    U2 = np.random.rand(*H2.shape) < p # second dropout mask
    H2 *= U2  # drop!

    out = np.dot(W3, H2) + b3

    # backward pass: compute gradients... (not shown)
    # perform parameter update... (not shown)
```

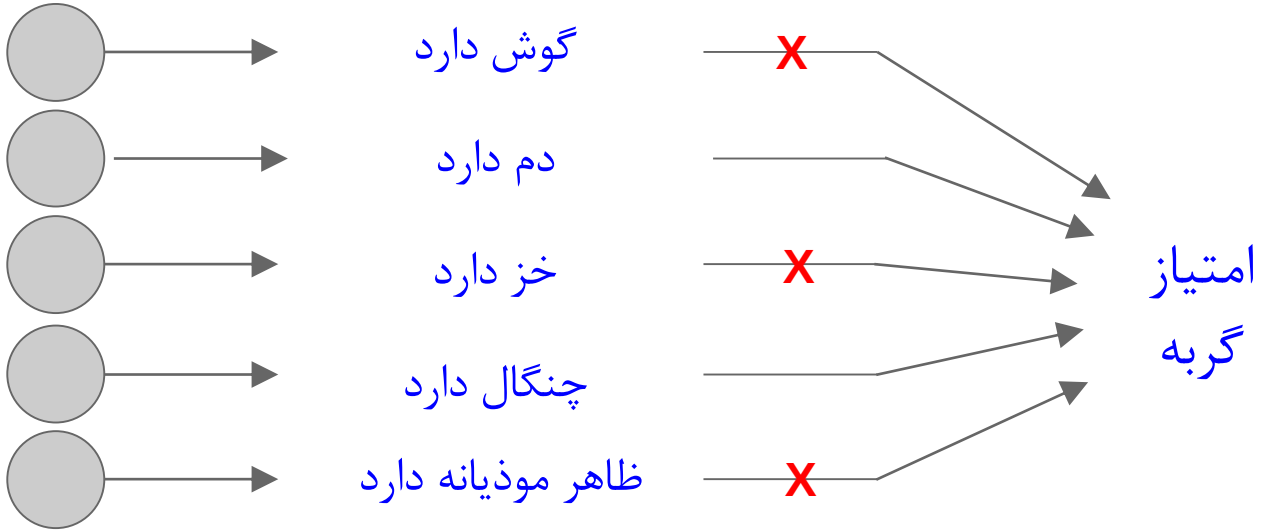


تنظیم: حذف تصادفی

- چگونه ممکن است حذف تصادفی برخی نورون‌ها ایده خوبی باشد؟

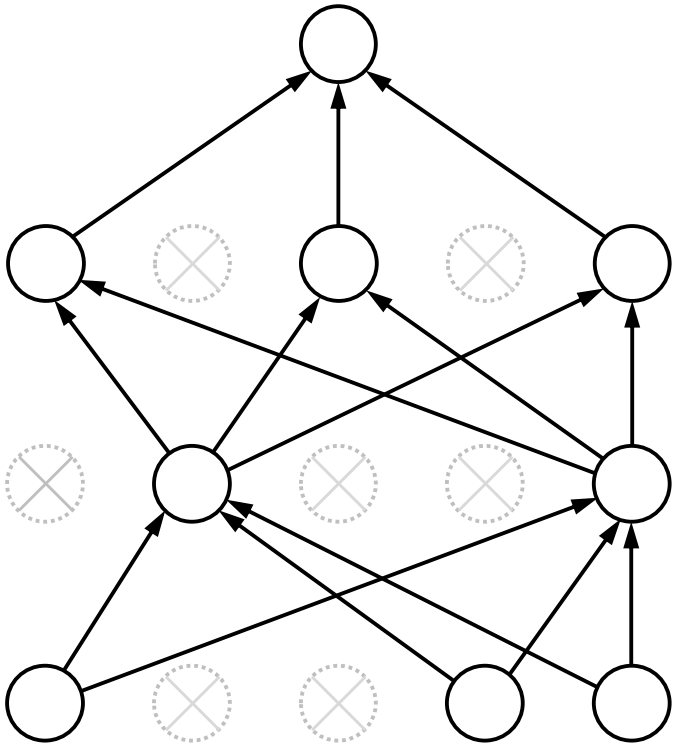


اجبار شبکه به داشتن بازنمایی دارای افزونگی



تنظیم: حذف تصادفی

- چگونه ممکن است حذف تصادفی برخی نورون‌ها ایده خوبی باشد؟

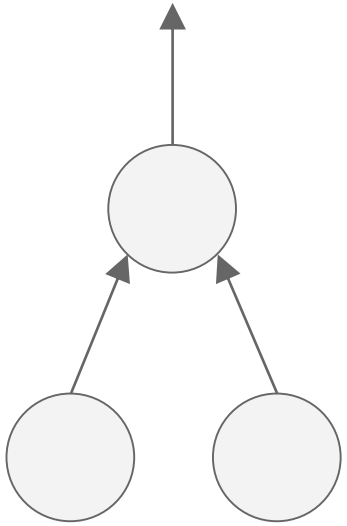


یک تفسیر دیگر:

با حذف تصادفی، یک مجموعه بزرگ از مدل‌ها آموزش داده می‌شوند (که دارای پارامترهای مشترک هستند)

حذف تصادفی در زمان پیش‌بینی...

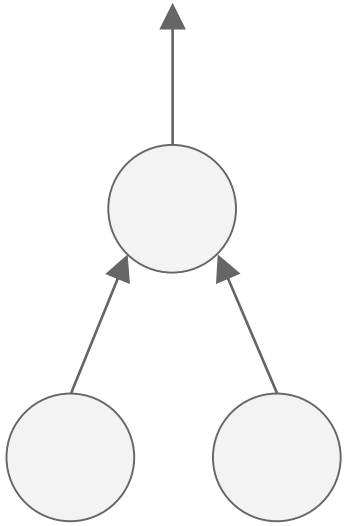
- می‌توان این کار را با یک مرحله از محاسبات رو به جلو انجام داد! (تقریب)
  - با فعال نگاه داشتن همه‌ی نورون‌ها (بدون حذف تصادفی)



(می‌توان نشان داد این کار تقریبی از ارزیابی اجتماع همه‌ی مدل‌ها است)

حذف تصادفی در زمان پیش‌بینی...

- می‌توان این کار را با یک مرحله از محاسبات رو به جلو انجام داد! (تقریب)
  - با فعال نگاه داشتن همه‌ی نورون‌ها (بدون حذف تصادفی)

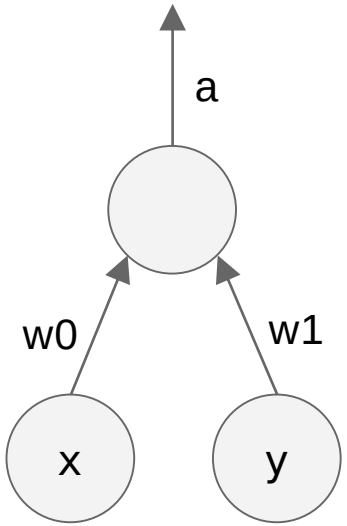


فرض کنید با فعال بودن همه‌ی ورودی‌ها در زمان پیش‌بینی، خروجی این نورون برابر با a باشد.

خروجی این نورون در زمان آموزش به طور متوسط چه مقدار است؟ (اگر p = 0.5)

حذف تصادفی در زمان پیش‌بینی...

- می‌توان این کار را با یک مرحله از محاسبات رو به جلو انجام داد! (تقریب)
  - با فعال نگاه داشتن همه‌ی نورون‌ها (بدون حذف تصادفی)



در زمان پیش‌بینی:

$$a = w0*x + w1*y$$

در زمان آموزش:

$$E[a] = \tfrac{1}{4} * (w0*0 + w1*0 + w0*0 + w1*y + w0*x + w1*0 + w0*x + w1*y)$$
$$= \tfrac{1}{4} * (2\ w0*x + 2\ w1*y)$$
$$= \tfrac{1}{2} * (w0*x + w1*y)$$

اگر $p = 0.5$ باشد، خروجی در زمان پیش‌بینی ۲ برابر خروجی در زمان آموزش خواهد بود!

⇐ برای جبران این مسئله باید مقدار فعالیت نورون در زمان پیش‌بینی در ضریب $1/2$ ضرب شود.

حذف تصادفی در زمان پیش‌بینی...


```
def predict(X):
    # ensemble forward pass
    H1 = np.maximum(0, np.dot(W1, X) + b1) * p  # NOTE: scale the activations
    H2 = np.maximum(0, np.dot(W2, H1) + b2) * p  # NOTE: scale the activations
    out = np.dot(W3, H2) + b3
```

- در زمان پیش‌بینی همه‌ی نورون‌ها همواره فعال هستند:
- بنابراین باید به گونه‌ای مقادیر فعالیت نورون‌ها را تغییر دهیم که:
- خروجی در زمان پیش‌بینی = متوسط خروجی در زمان آموزش

حذف تصادفی

```python
p = 0.5  # probability of keeping a unit active

def train_step(X):
    """ X contains the data """

    # forward pass for a 3-layer neural network
    H1 = np.maximum(0, np.dot(W1, X) + b1)
    U1 = np.random.rand(*H1.shape) < p  # first dropout mask!
    H1 *= U1  # drop!

    H2 = np.maximum(0, np.dot(W2, H1) + b2)
    U2 = np.random.rand(*H2.shape) < p # second dropout mask!
    H2 *= U2  # drop!

    out = np.dot(W3, H2) + b3

    # backward pass: compute gradients... (not shown)
    # perform parameter update... (not shown)

def predict(X):
    # ensemble forward pass
    H1 = np.maximum(0, np.dot(W1, X) + b1) * p  # NOTE: scale the activations
    H2 = np.maximum(0, np.dot(W2, H1) + b2) * p  # NOTE: scale the activations
    out = np.dot(W3, H2) + b3
```

حذف تصادفی در زمان آموزش

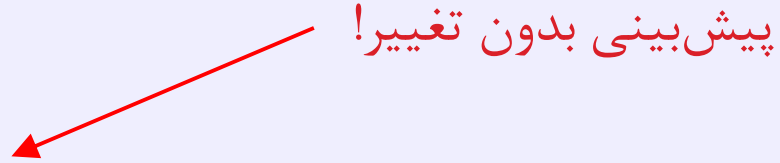
مقیاس‌بندی در زمان پیش‌بینی

این پیاده‌سازی توصیه نمی‌شود!

حذف تصادفی

پیاده‌سازی متداول

```python
p = 0.5  # probability of keeping a unit active

def train_step(X):
    """ X contains the data """

    # forward pass for a 3-layer neural network
    H1 = np.maximum(0, np.dot(W1, X) + b1)
    U1 = (np.random.rand(*H1.shape) < p) / p  # first dropout mask! Notice /p
    H1 *= U1  # drop!

    H2 = np.maximum(0, np.dot(W2, H1) + b2)
    U2 = (np.random.rand(*H2.shape) < p) / p # second dropout mask! Notice /p
    H2 *= U2  # drop!

    out = np.dot(W3, H2) + b3

    # backward pass: compute gradients... (not shown)
    # perform parameter update... (not shown)

def predict(X):
    # ensemble forward pass
    H1 = np.maximum(0, np.dot(W1, X) + b1)  # NOTE: no scaling necessary
    H2 = np.maximum(0, np.dot(W2, H1) + b2)  # NOTE: no scaling necessary
    out = np.dot(W3, H2) + b3
```

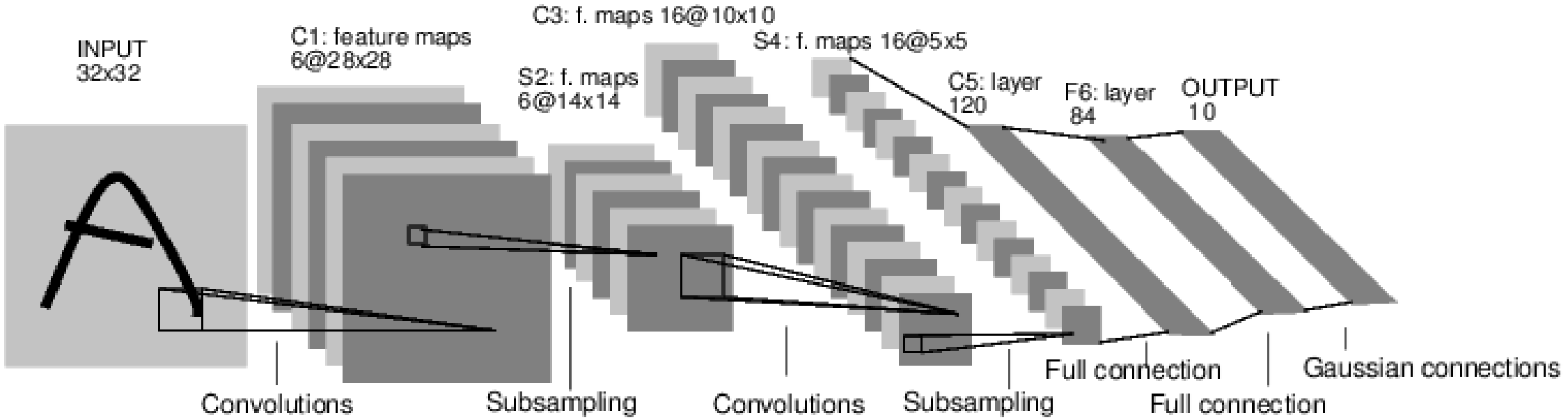


یادگیری عمیق - شبکه‌های عصبی کانولوشنال

چرا یادگیری عمیق

- یادگیری توبع پیچیده‌تر با ترکیب دنباله‌ای از توابع ساده‌تر در هر لایه.
  - لایه‌های بیشتر یعنی ظرفیت یادگیری بیشتر و حل مسائل پیچیده‌تر.
  - یادگیری در سطوح مختف انتزاع.
- استفاده از تمامی داده‌های در دسترس به منظور آموزش.
  - جلوگیری از بیش‌برازش!
- استفاده از محاسبات ارزان به صورت موازی.
  - پردازنده‌های گرافیکی [GPU]

شبکه‌های عصبی کانولوشنال

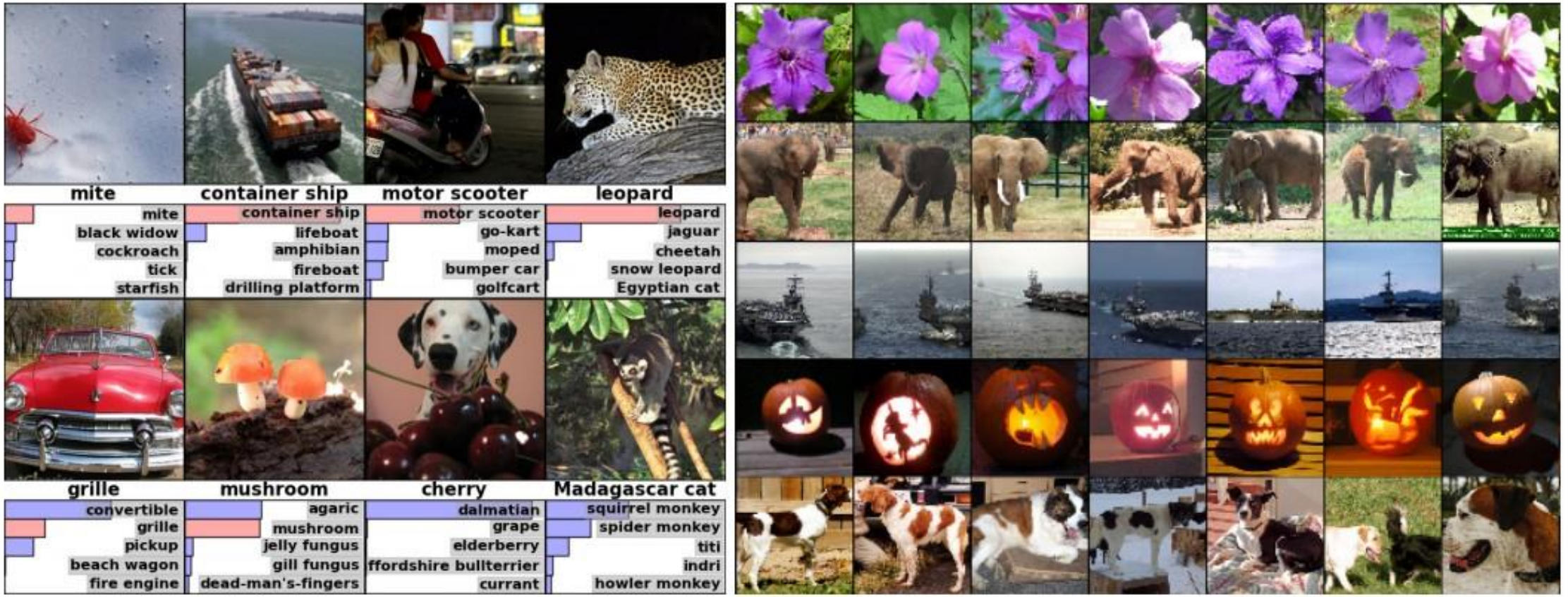


[LeNet-5, LeCun 1980]

شبکه‌های عصبی کانولوشنال

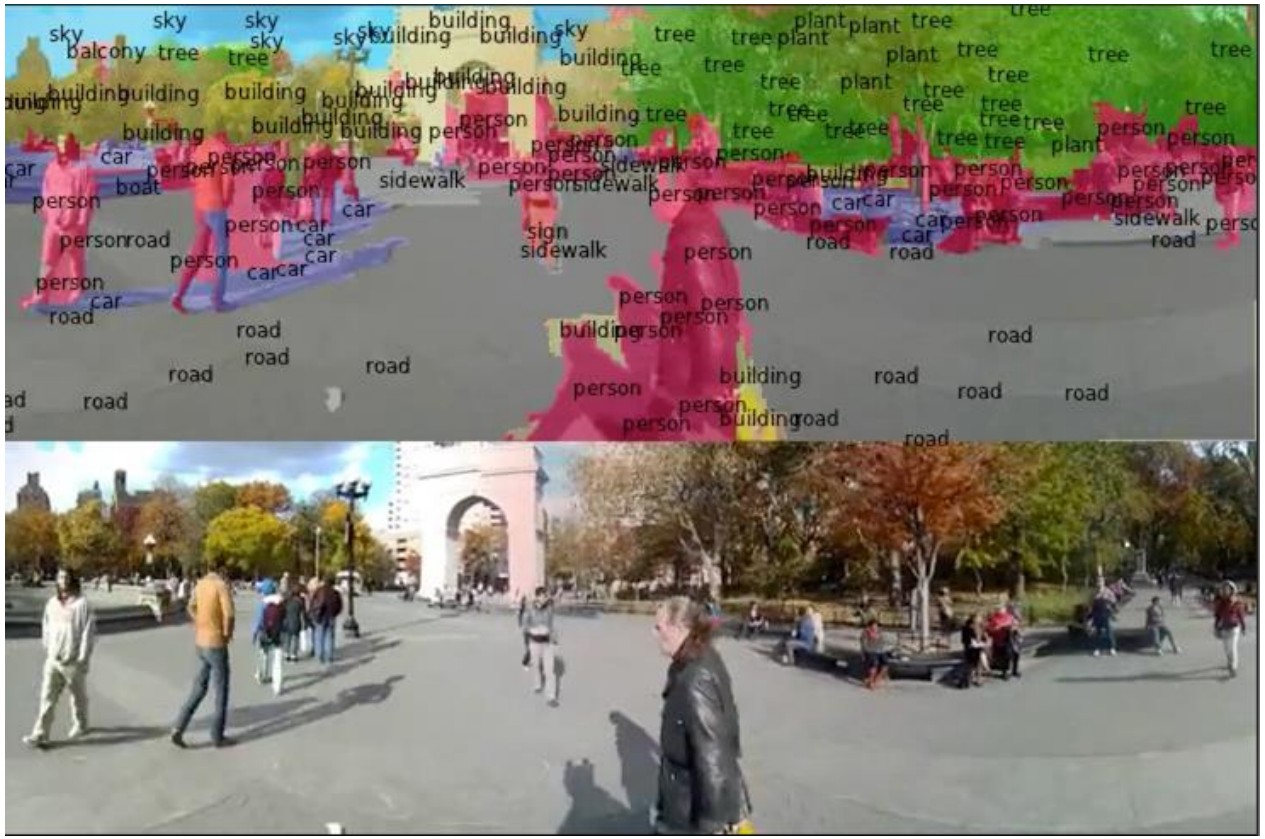
دسته‌بندی

بازیابی

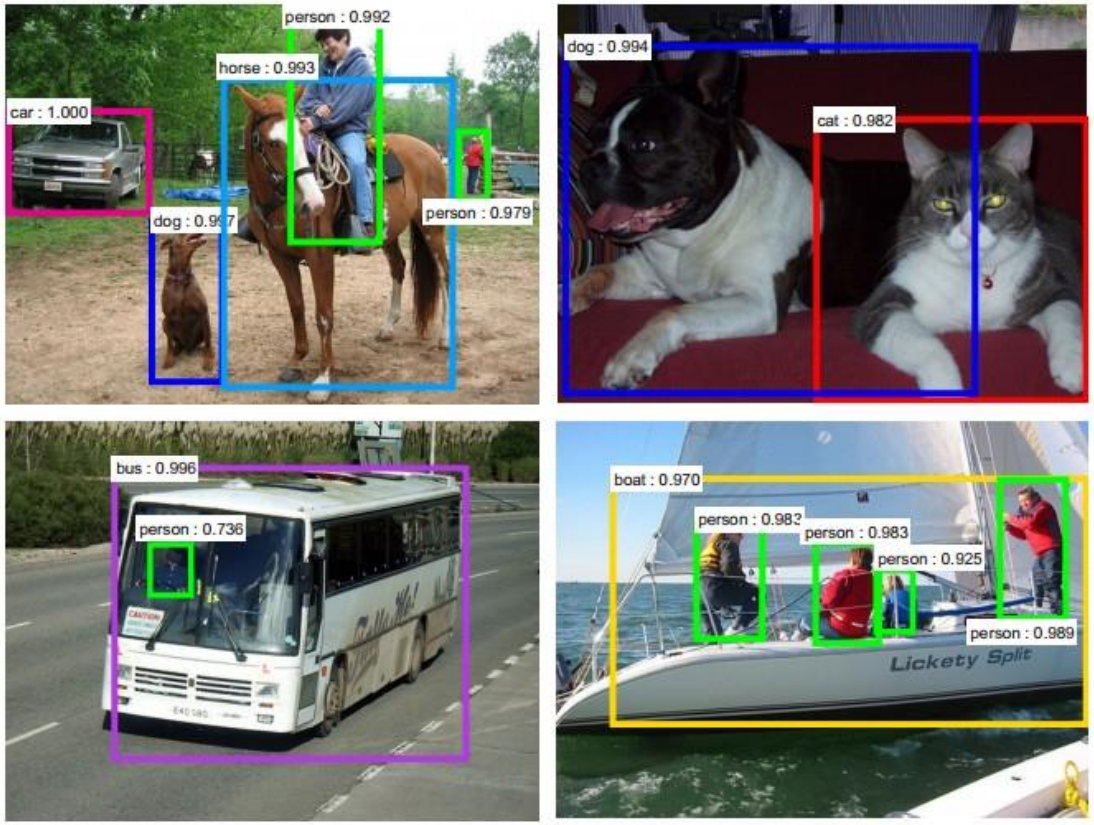


شبکه‌های عصبی کانولوشنال

قطعه‌بندی



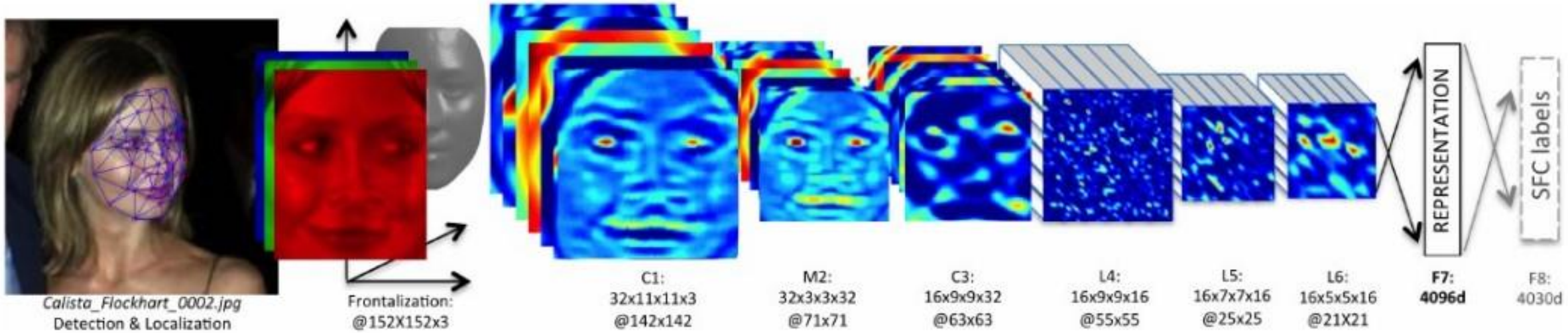
شناسایی

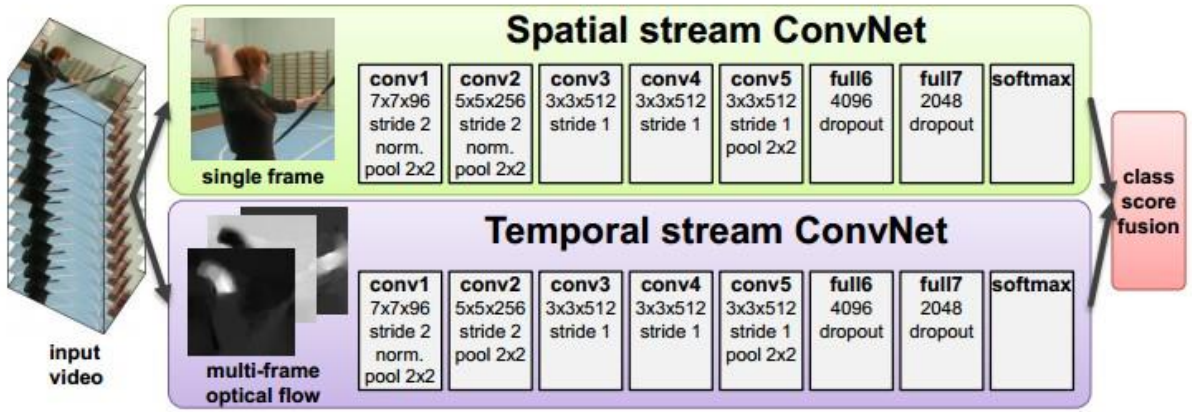


شبکه‌های عصبی کانولوشنال

خودروی بدون راننده

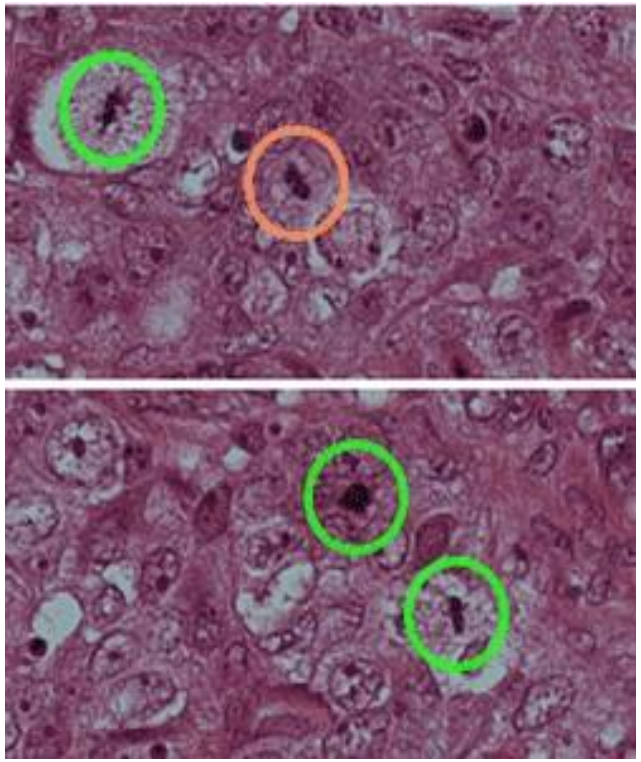
شبکه‌های عصبی کانولوشنال



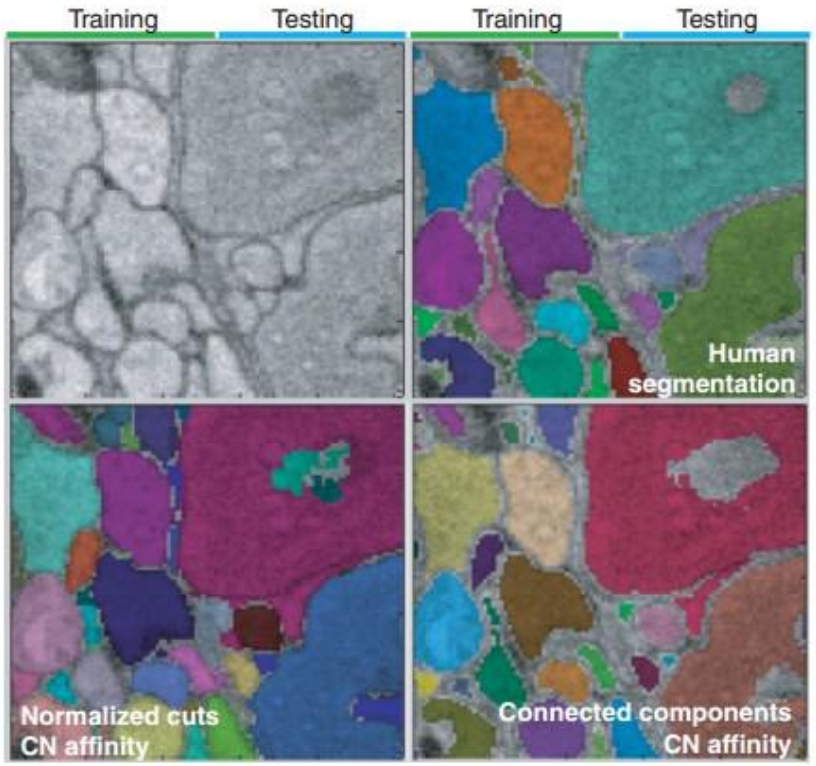


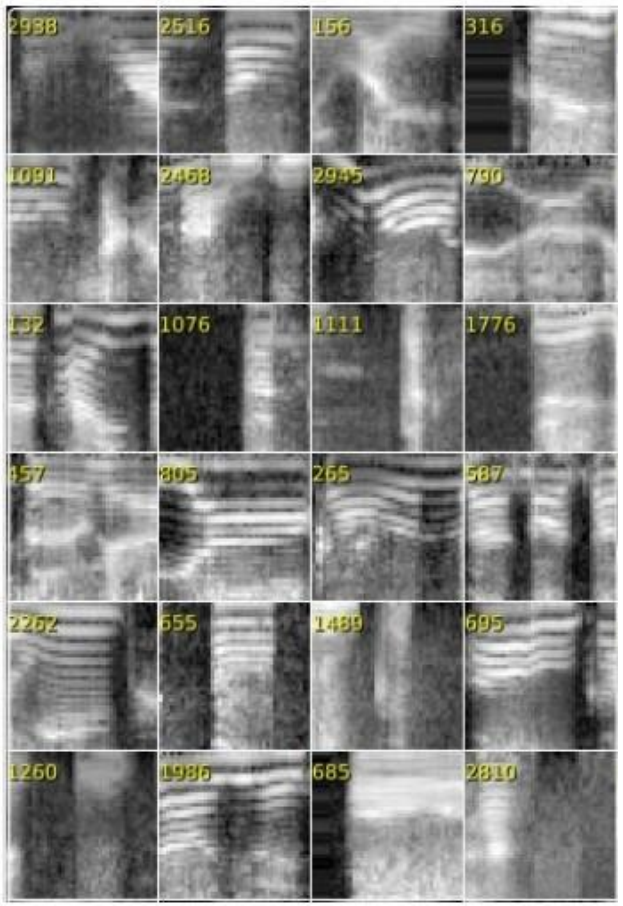
شبکه‌های عصبی کانولوشنال

شبکه‌های عصبی کانولوشنال



شبکه‌های عصبی کانولوشنال



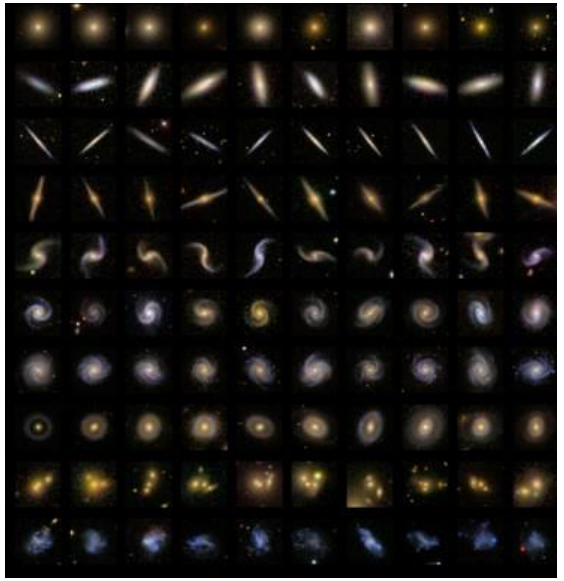


I caught this movie on the Sci-Fi channel recently. It actually turned out to be pretty decent as far as B-list horror/suspense films go. **Two guys (one naive and one loud mouthed a **) take a road trip to stop a wedding but have the worst possible luck when a maniac in a freaky, make-shift tank/truck hybrid decides to play cat-and-mouse with them.** Things are further complicated when they pick up a ridiculously whorish hitchhiker. What makes this film unique is that the combination of comedy and terror actually work in this movie, unlike so many others. The two guys are likable enough and there are some good chase/suspense scenes. Nice pacing and comic timing make this movie more than passable for the horror/slasher buff. **Definitely worth checking out.**

I just saw this on a local independent station in the New York City area. **The cast showed promise but when I saw the director, George Cosmotos, I became suspicious. And sure enough, it was every bit as bad, every bit as pointless and stupid as every George Cosmotos movie I ever saw.** He's like a stupid man's Michael Bey – with all the awfulness that accolade promises. There's no point to the conspiracy, no burning issues that urge the conspirators on. We are left to ourselves to connect the dots from one bit of graffiti on various walls in the film to the next. Thus, the current budget crisis, the war in Iraq, Islamic extremism, the fate of social security, 47 million Americans without health care, stagnating wages, and the death of the middle class are all subsumed by the sheer terror of graffiti. A truly, stunningly idiotic film.

Graphics is far from the best part of the game. **This is the number one best TH game in the series.** Next to Underground. **It deserves strong love. It is an insane game.** There are massive levels, massive unlockable characters... it's just a massive game. **Waste your money on this game. This is the kind of money that is wasted properly.** And even though graphics suck, thats doesn't make a game good. Actually, the graphics were good at the time. Today the graphics are crap. WHO CARES? As they say in Canada, This is the fun game, aye. (You get to go to Canada in THPS3) Well, I don't know if they say that, but they might. who knows. Well, Canadian people do. Wait a minute, I'm getting off topic. This game rocks. Buy it, play it, enjoy it, love it. It's PURE BRILLIANCE.

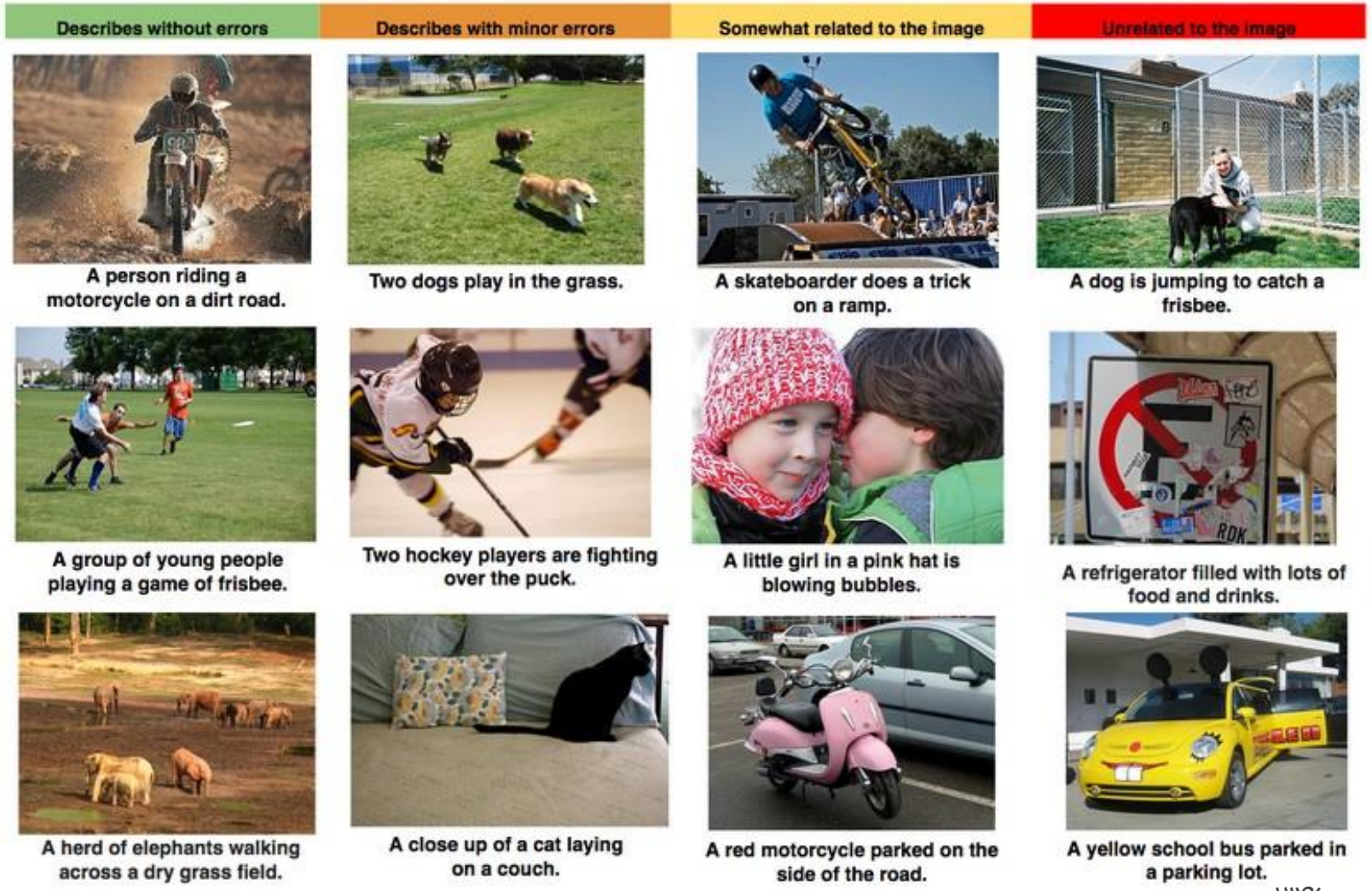
The first was good and original. I was a not bad horror/comedy movie. So I heard a second one was made and I had to watch it . What really makes this movie work is Judd Nelson's character and the sometimes clever script. **A pretty good script for a person who wrote the Final Destination films and the direction was okay.** Sometimes there's scenes where it looks like it was filmed using a home video camera with a grainy - look. Great made - for - TV movie. **It was worth the rental and probably worth buying just to get that nice eerie feeling and watch Judd Nelson's Stanley doing what he does best.** I suggest newcomers to watch the first one before watching the sequel, just so you'll have an idea what Stanley is like and get a little history background.



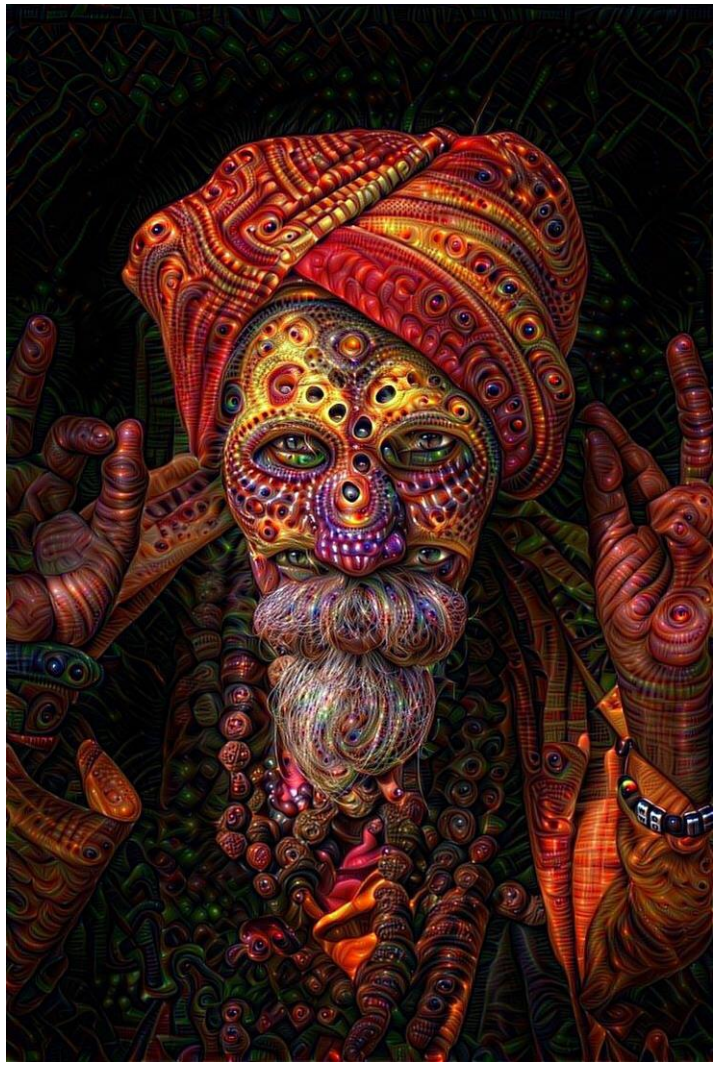
شبکه‌های عصبی کانولوشنال

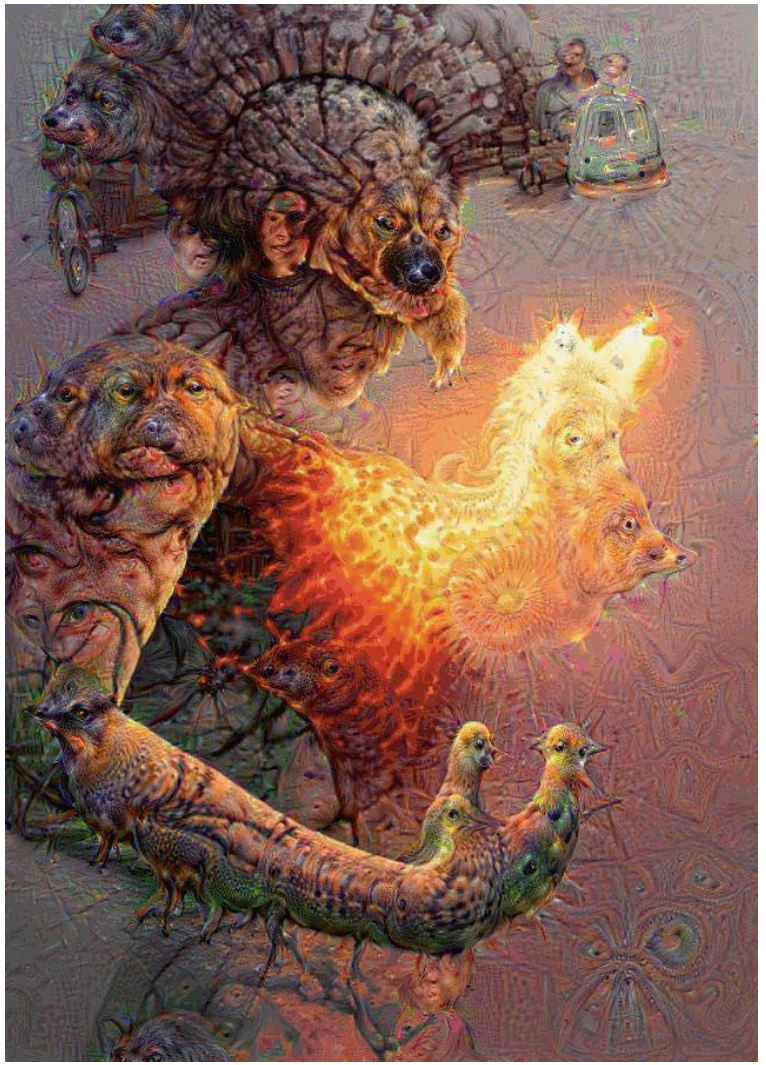
شبکه‌های عصبی کانولوشنال

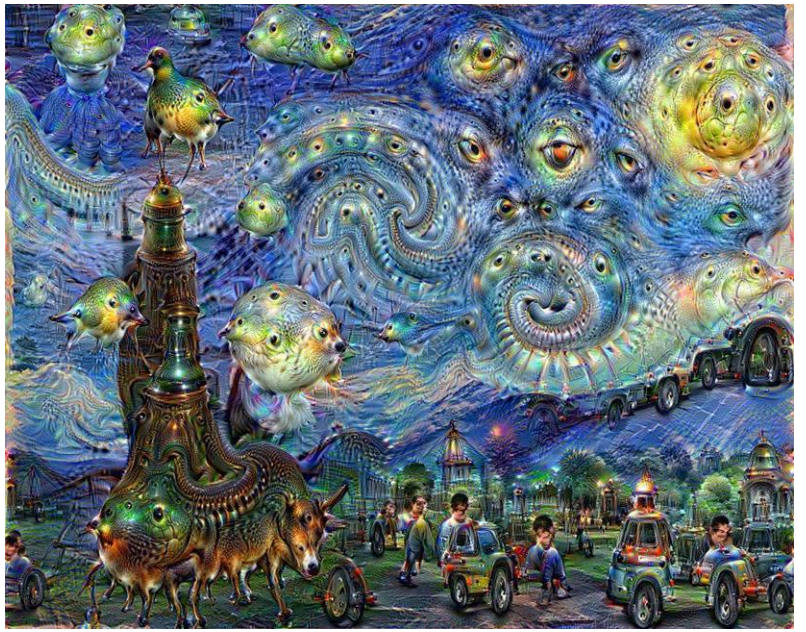
عنوان‌بندی تصاویر



شبکه‌های عصبی کانولوشنال

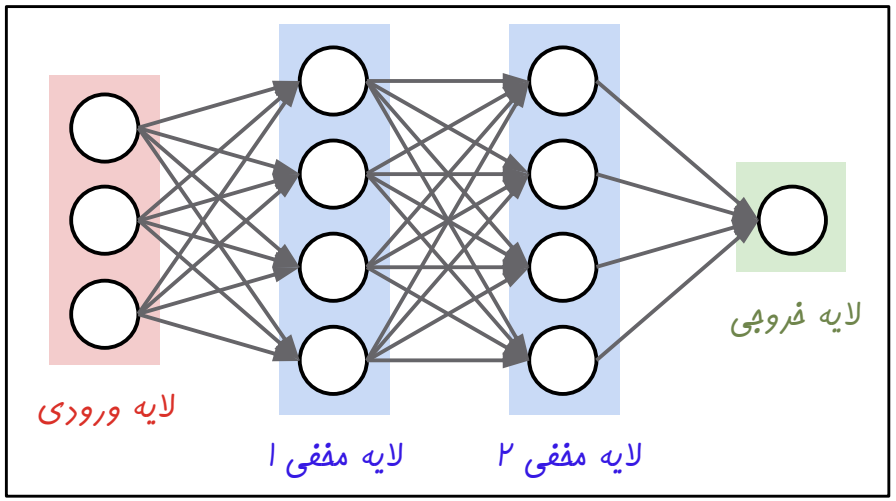






یادآوری: بهینه‌سازی

- گرادیان کاهشی. [با دسته‌های کوچک]



حلقه:

۱. یک دسته از داده‌ها را به طور تصادفی **انتخاب** کن.

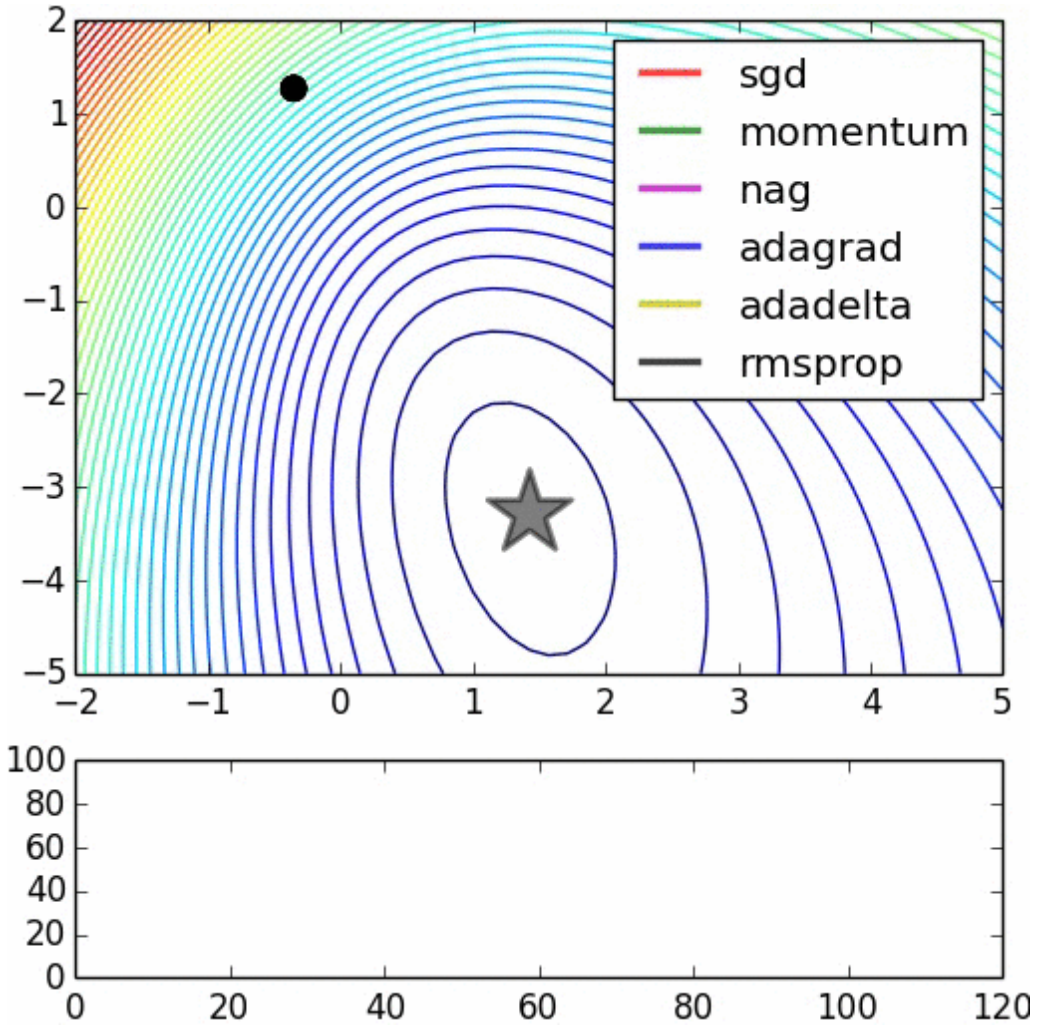
۲. **محاسبات رو به جلو** را در طول گراف انجام بده و مقدار تابع هزینه را محاسبه کن.

۳. **محاسبات رو به عقب** را به منظور محاسبه گرادیان‌ها انجام بده.

۴. مقدار پارامترها را با استفاده از گرادیان‌های محاسبه شده **به روز رسانی** کن.

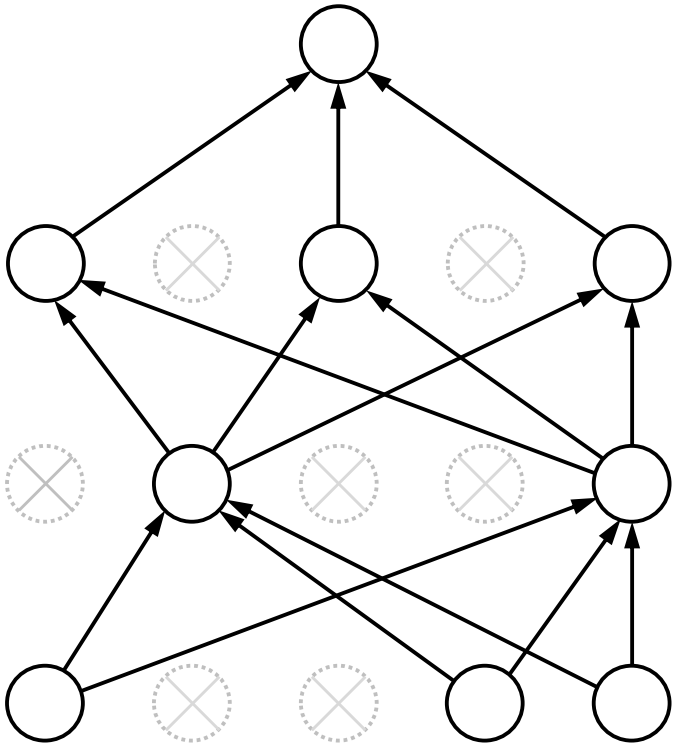
یادآوری: روش‌های به روز رسانی پارامترها

- به روز رسانی پارامترها.

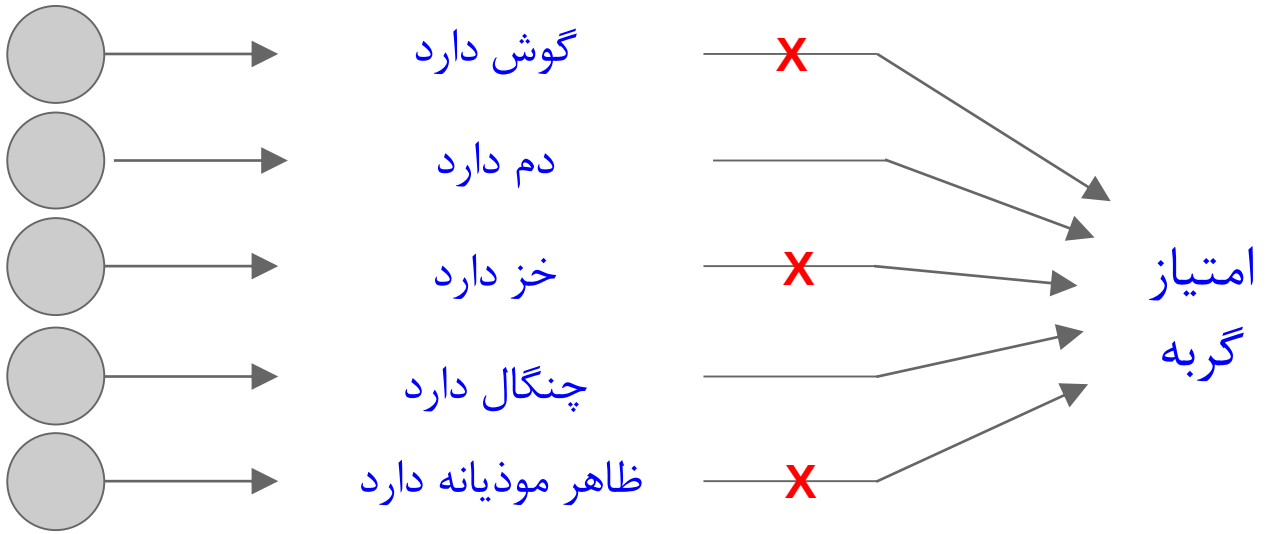


یادآوری: حذف تصادفی [Dropout]

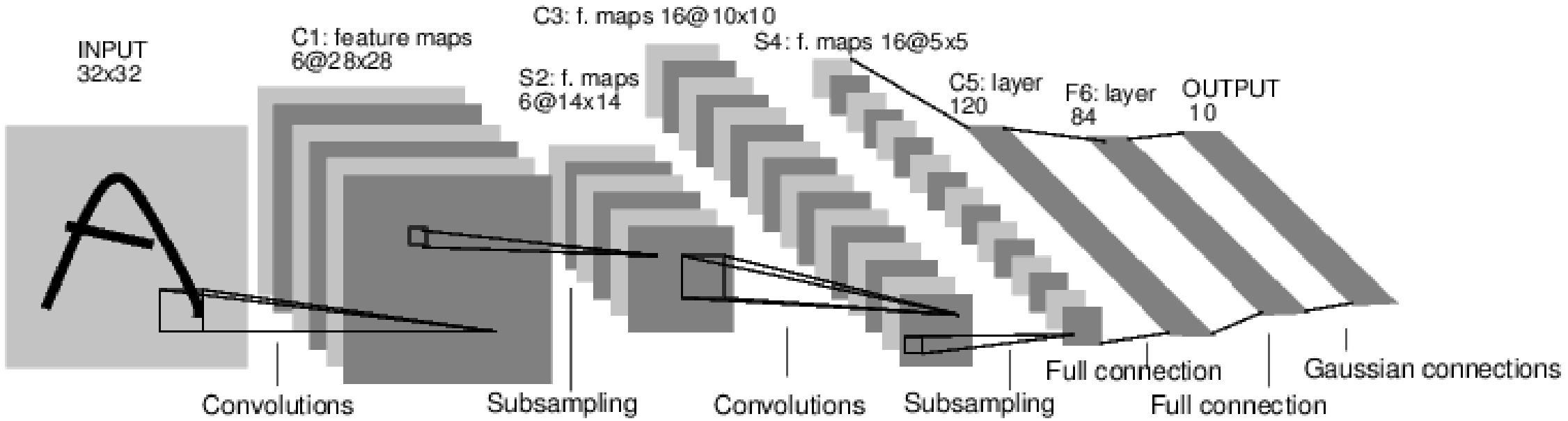
- چگونه ممکن است حذف تصادفی برخی نورون‌ها ایده‌ی خوبی باشد؟



اجبار شبکه به داشتن بازنمایی دارای افزونگی



شبکه‌های عصبی کانولوشن



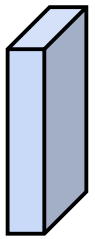
[LeNet-5, LeCun 1980]

لایه کانولوشن

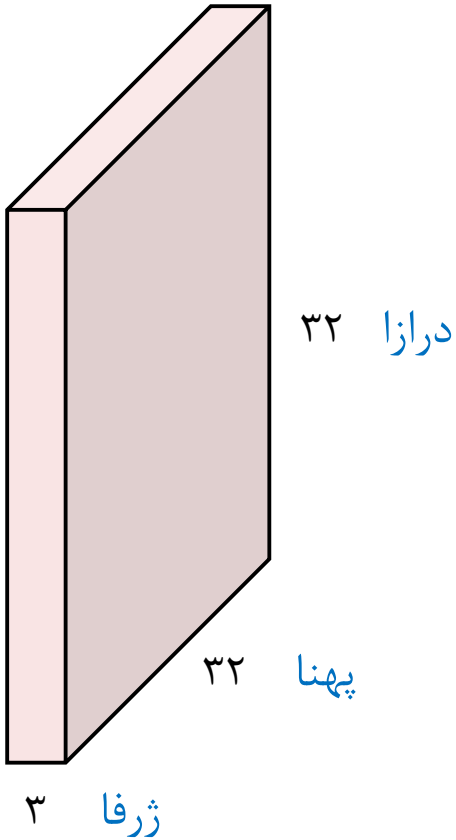
کانولوشن فیلتر با تصویر

یعنی «فیلتر را بر روی تصویر حرکت بده و هر بار ضرب داخلی فیلتر و بخشی از تصویر که فیلتر بر روی آن قرار دارد را محاسبه کن.»

فیلتر ۳×۵×۵



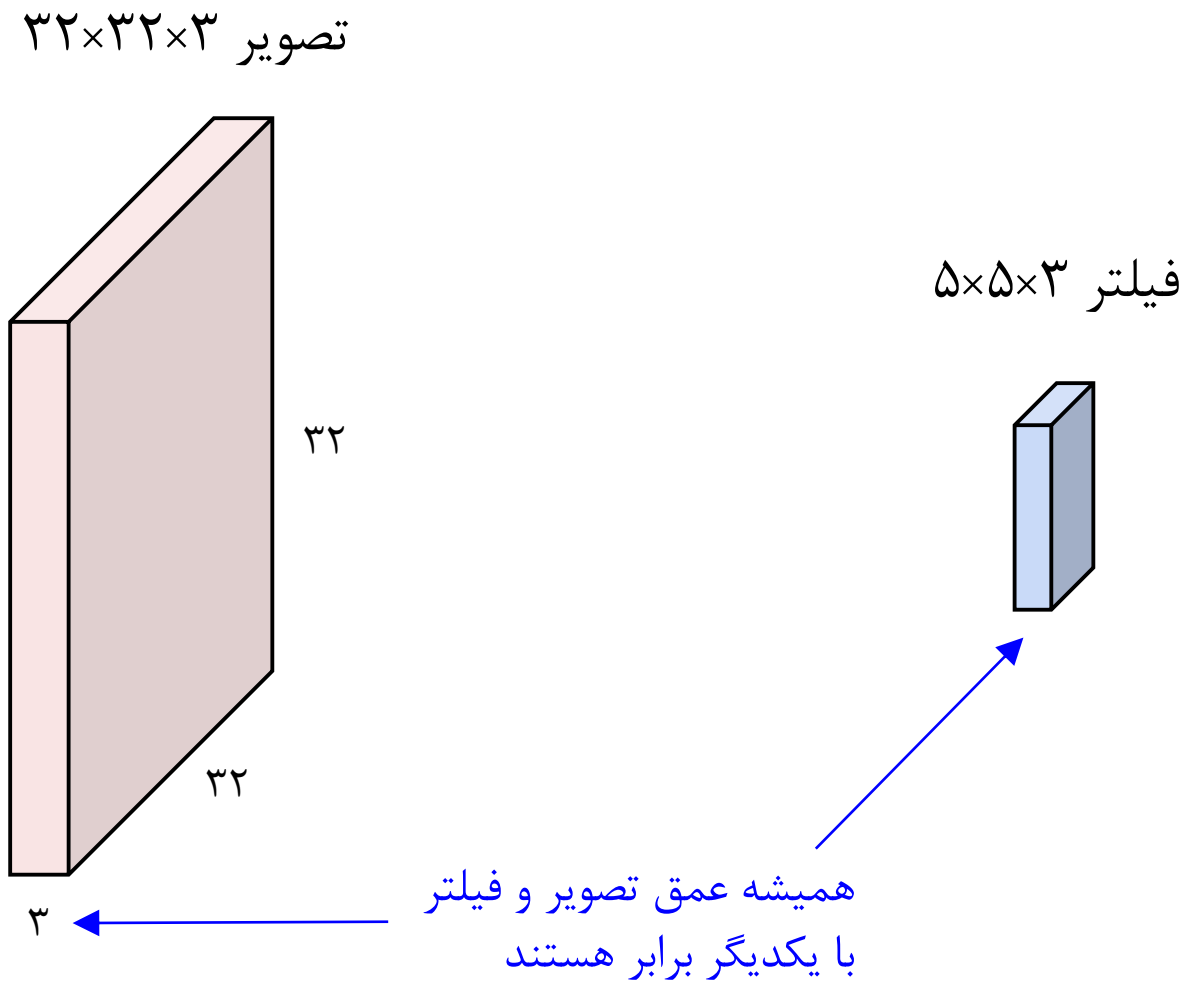
تصویر ۳×۳۲×۳۲



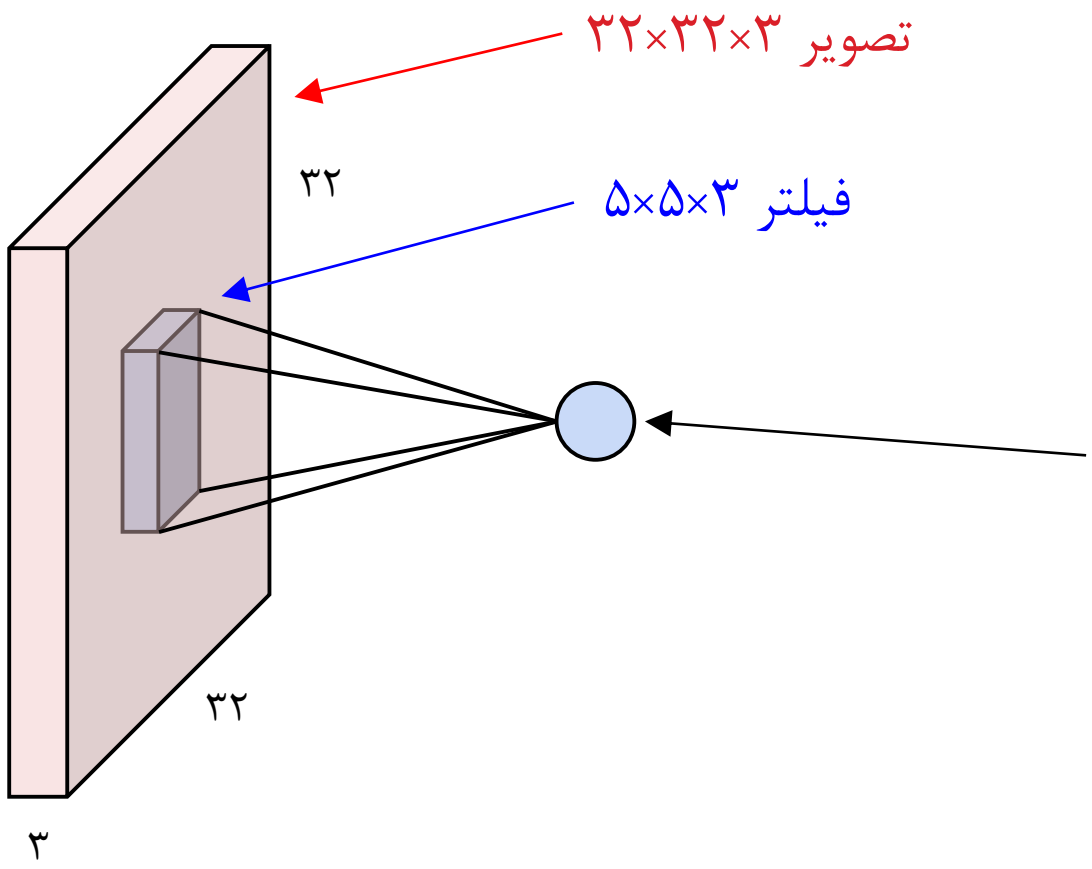
لایه کانولوشن

کانولوشن فیلتر با تصویر

یعنی «فیلتر را بر روی تصویر حرکت بده و هر بار ضرب داخلی فیلتر و بخشی از تصویر که فیلتر بر روی آن قرار دارد را محاسبه کن.»



لایه کانولوشن

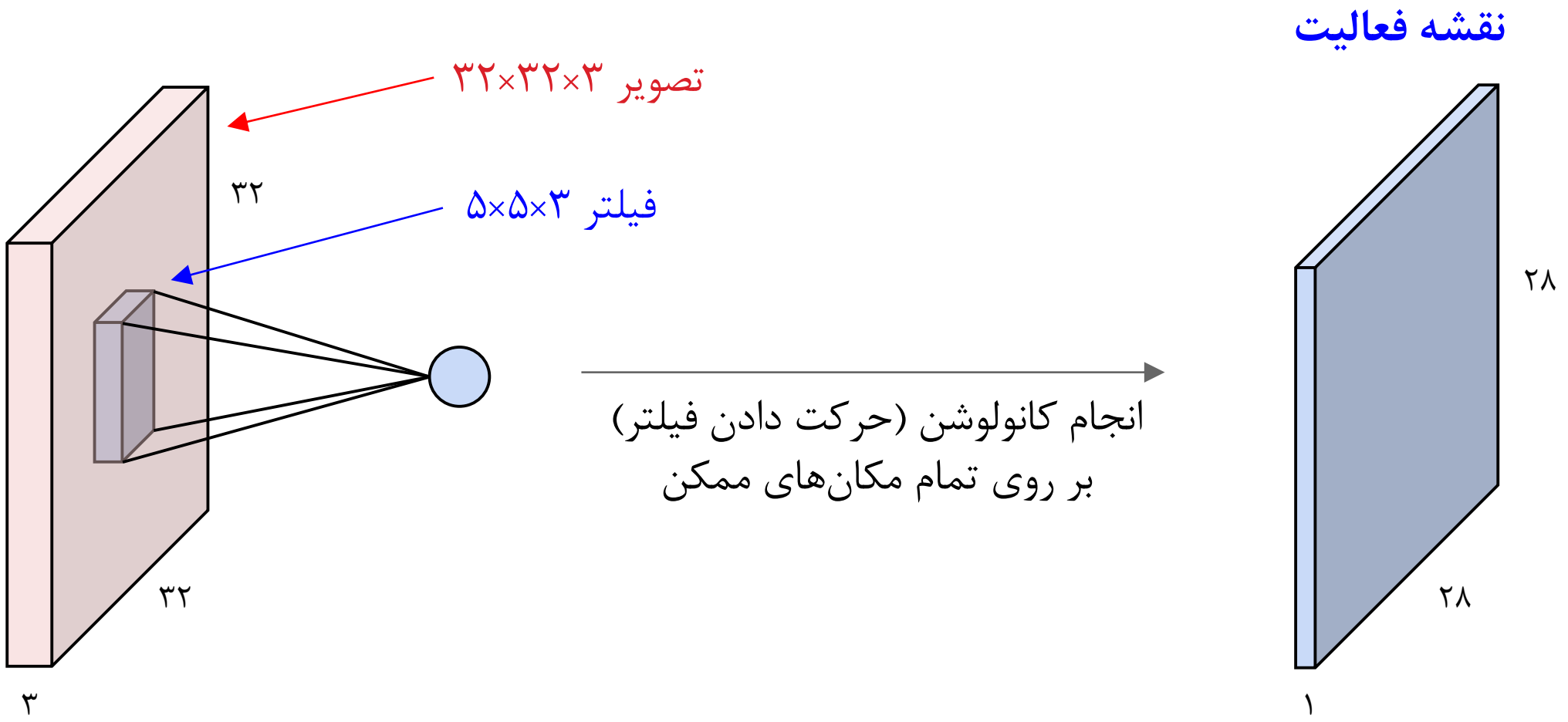


۱ عدد:

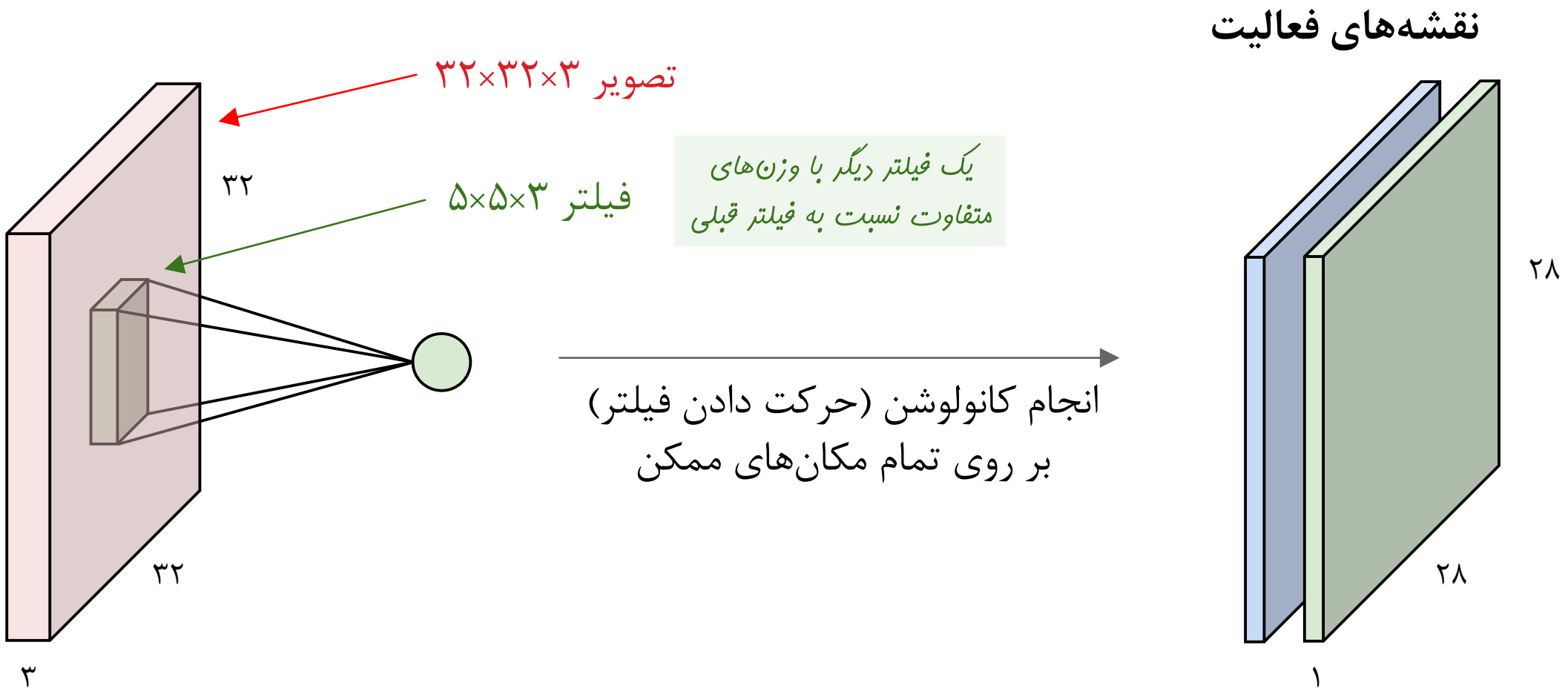
به دست آمده از ضرب داخلی فیلتر و بخش کوچکی از تصویر که فیلتر بر روی آن قرار دارد.
(یعنی، ضرب داخلی دو بردار ۷۵ بعدی + بایاس)

$$w^T x + b$$

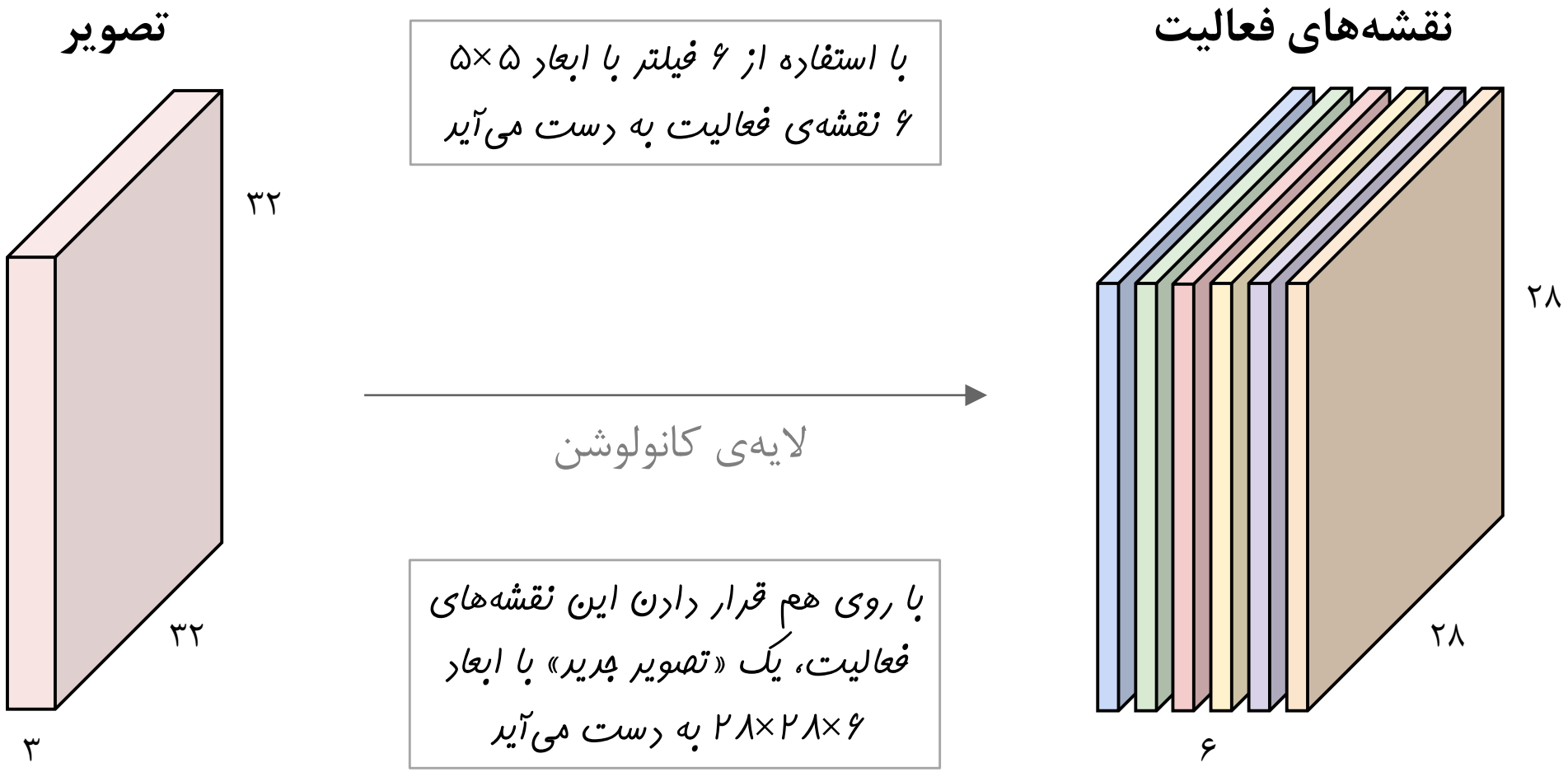
لایه کانولوشن



لایه کانولوشن

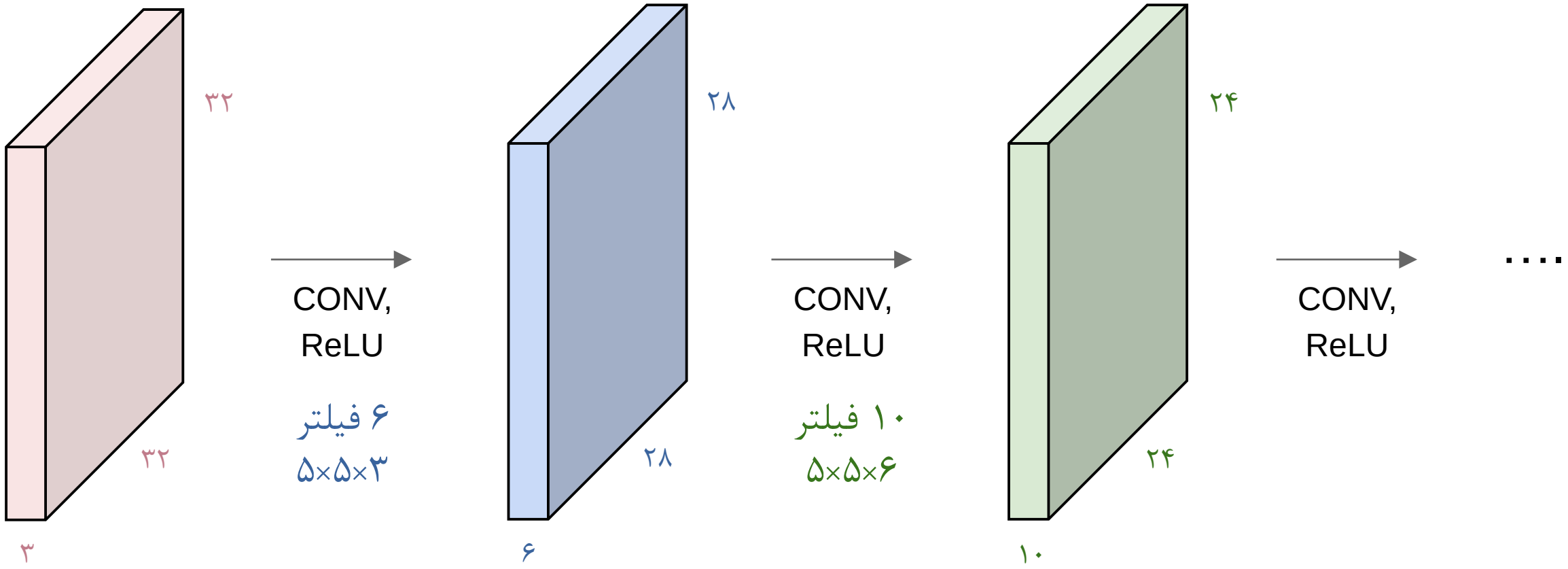


لایه کانولوشن

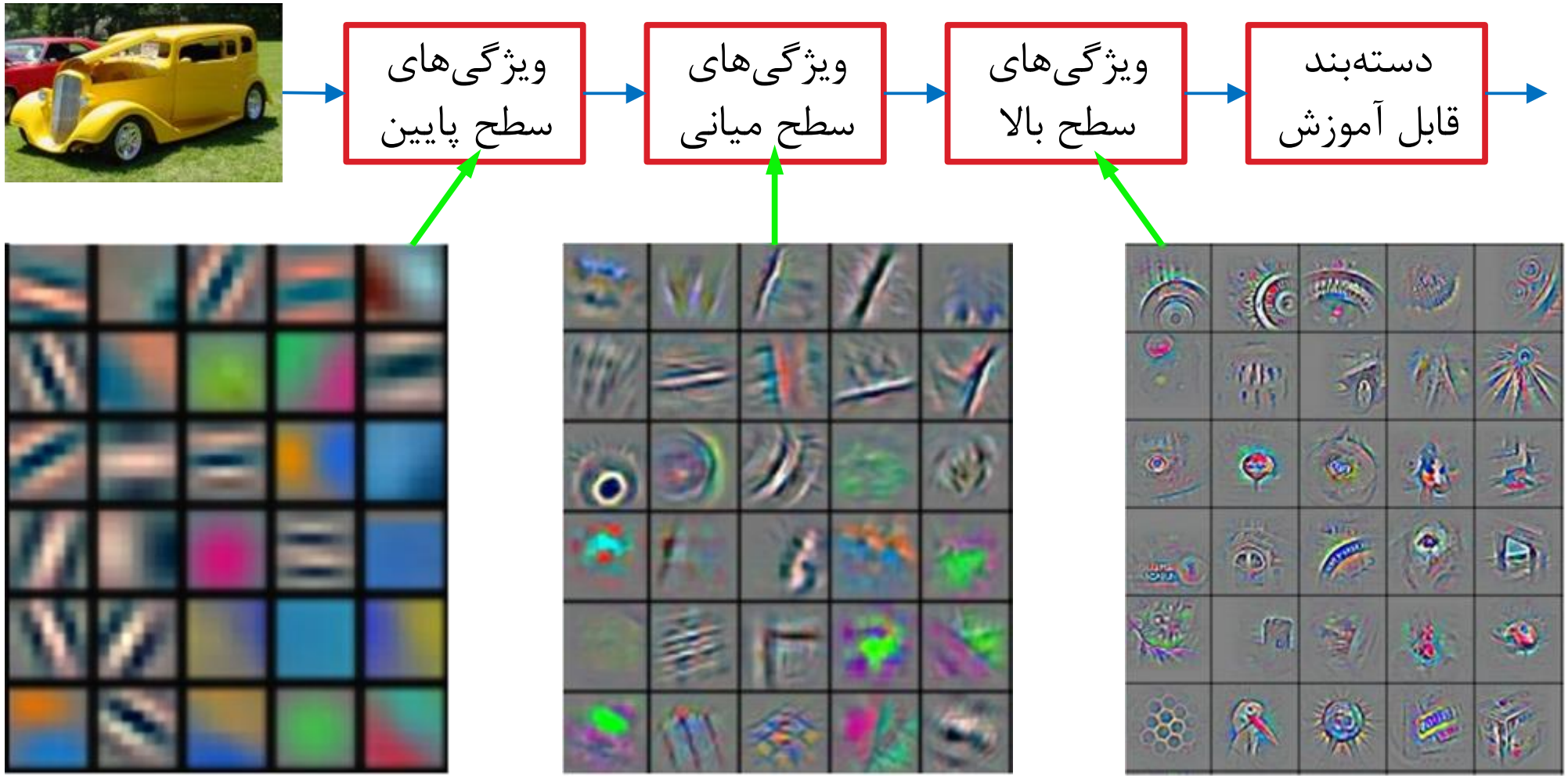


شبکه‌های عصبی کانولوشن

- شبکه‌های عصبی کانولوشن. یک دنباله از لایه‌های کانولوشن و توابع فعالیت در میان آنها.

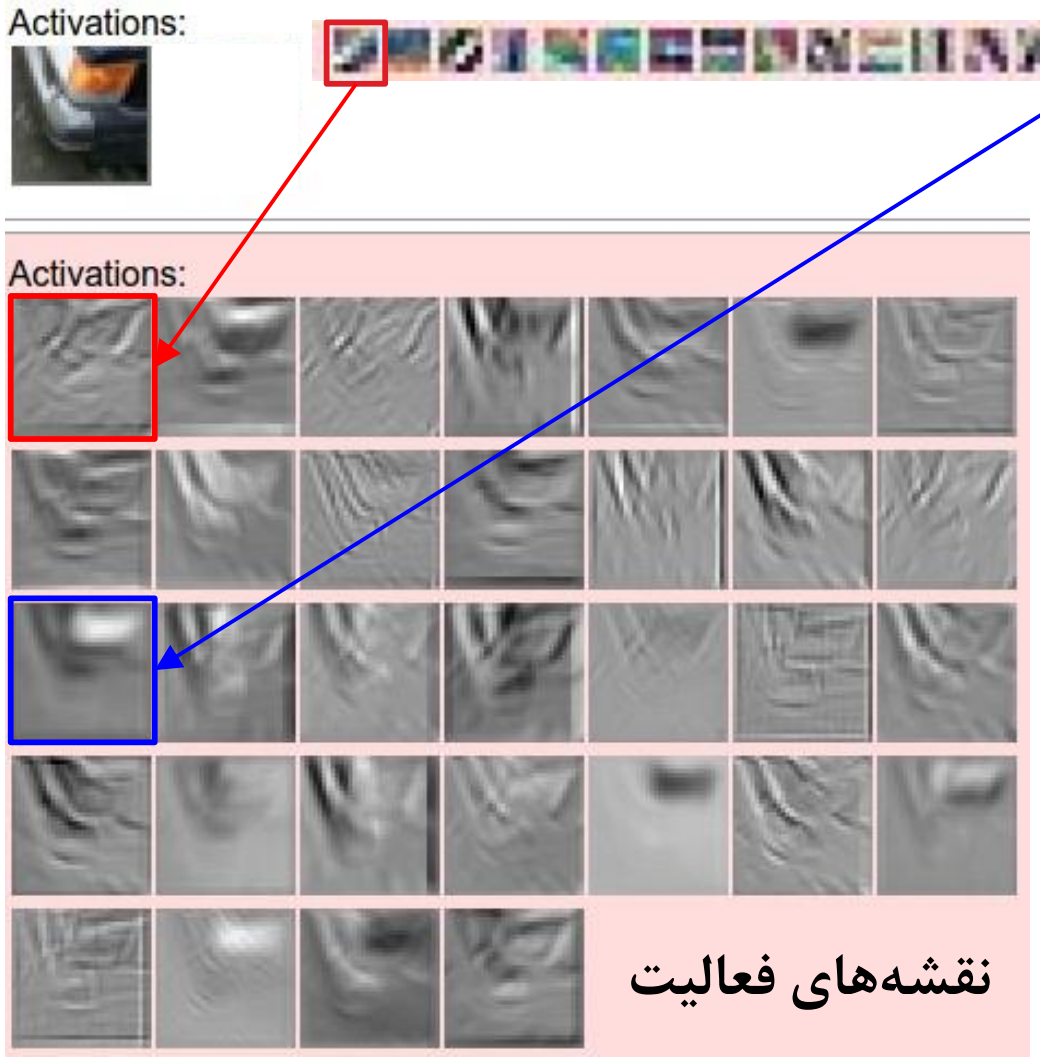


شبکه‌های عصبی کانولوشن



[Zeiler & Fergus 2013]

شبکه‌های عصبی کانولوشن



فیلترهای نمونه با ابعاد ۵×۵ [۳۲ فیلتر]

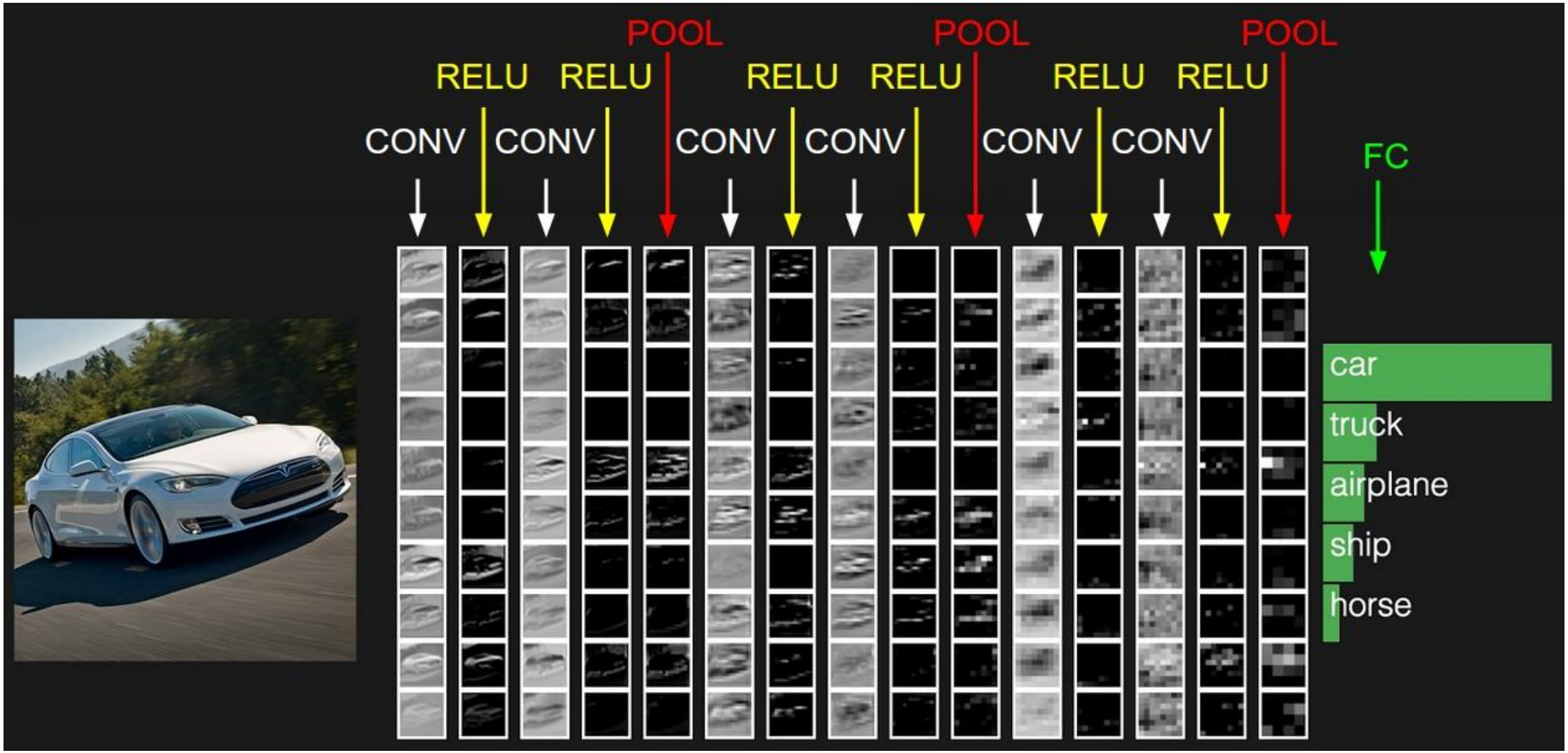
[هر فیلتر ⇐ یک نقشه‌ی فعالیت]

این لایه‌ها را به این دلیل که با کانولوشن دو سیگنال مرتبط هستند، لایه‌های کانولوشن می‌نامیم:

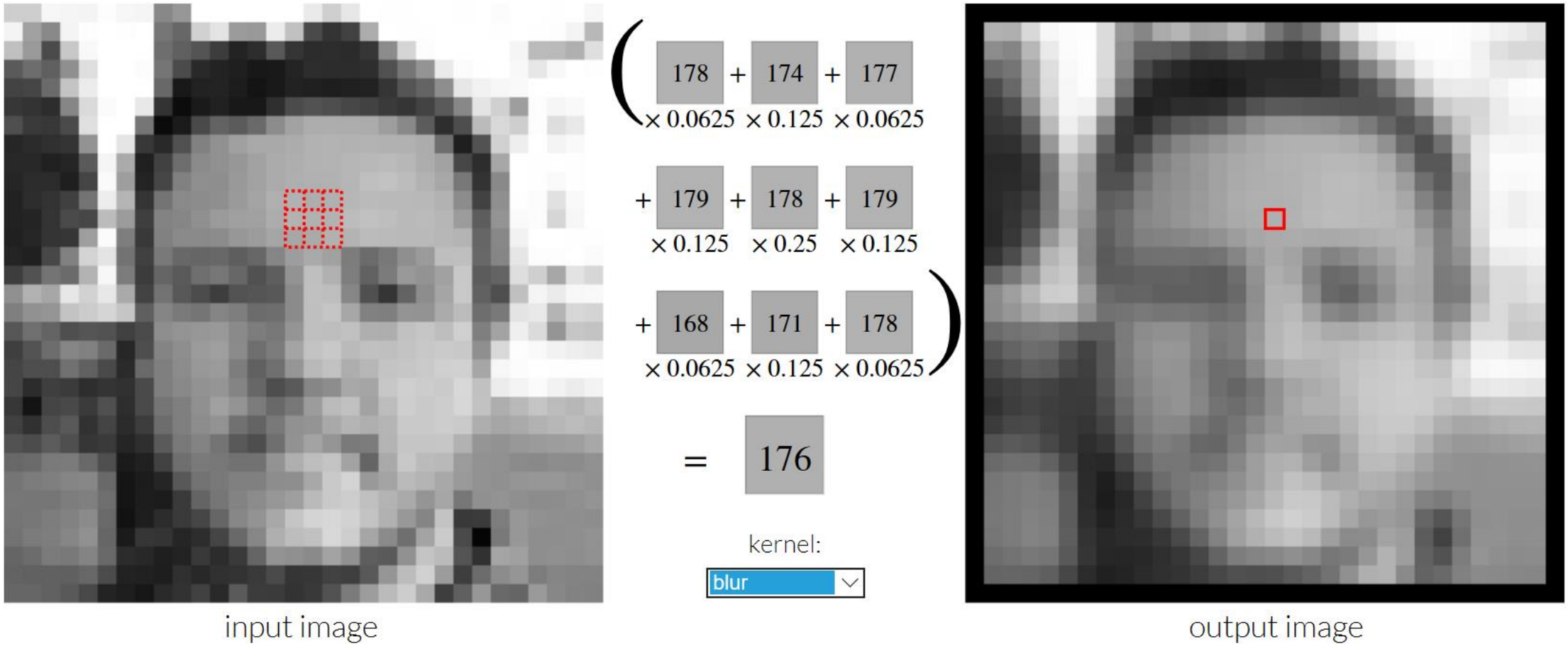
$$f[x,y] * g[x,y] = \sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} f[n_1,n_2] \cdot g[x-n_1, y-n_2]$$

ضرب عنصر به عنصر و جمع یک فیلتر و سیگنال

شبکه‌های عصبی کانولوشن

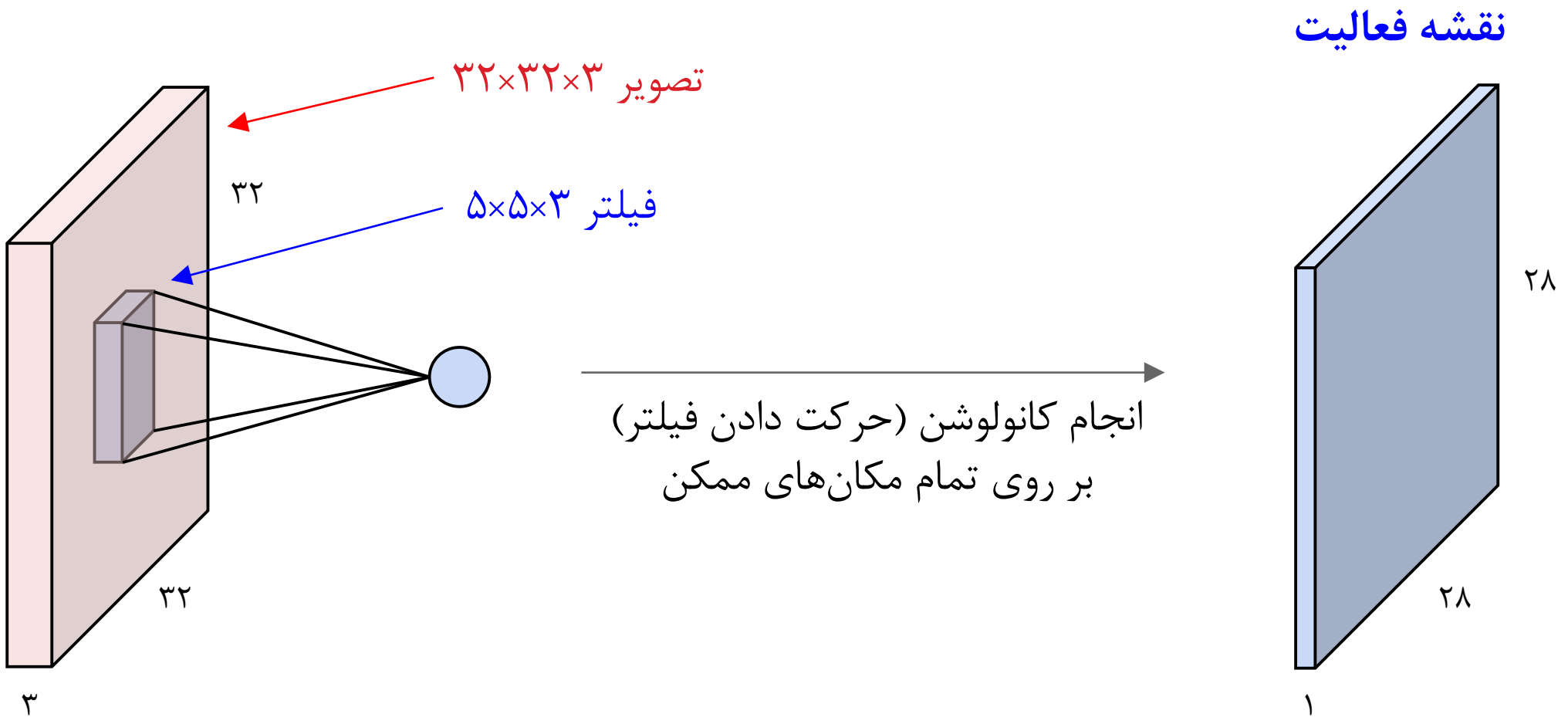


لایه کانولوشن: یک نگاه دقیق‌تر

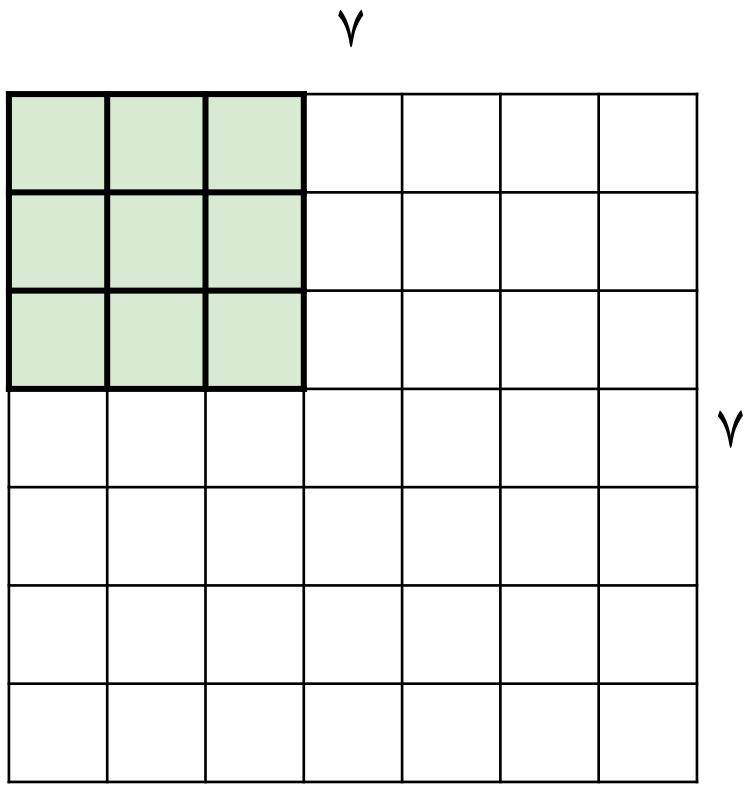


http://setosa.io/ev/image-kernels/

لایه کانولوشن: یک نگاه دقیق‌تر



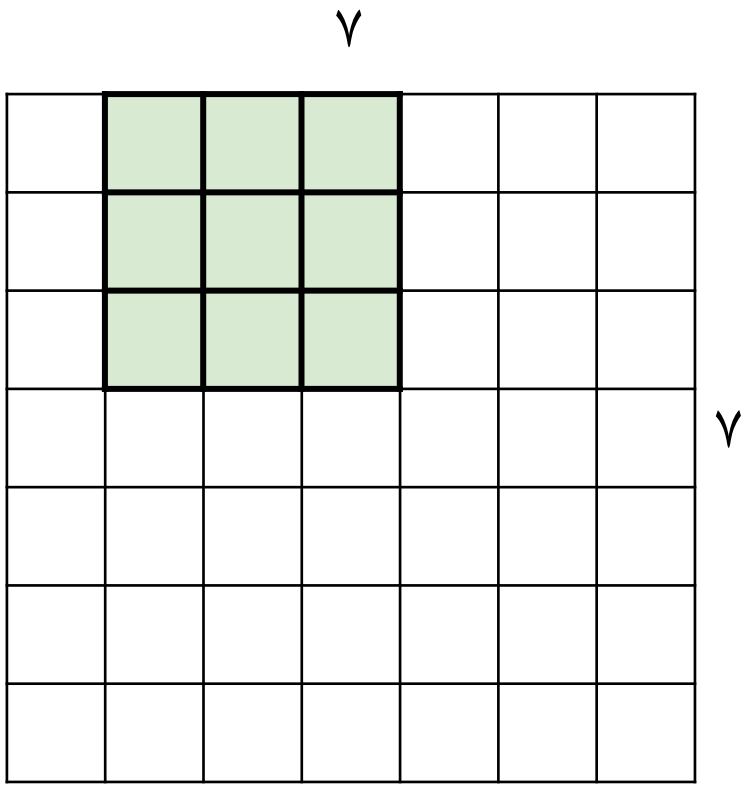
لایه کانولوشن: یک نگاه دقیق‌تر



ورودی ۷×۷

فیلتر ۳×۳

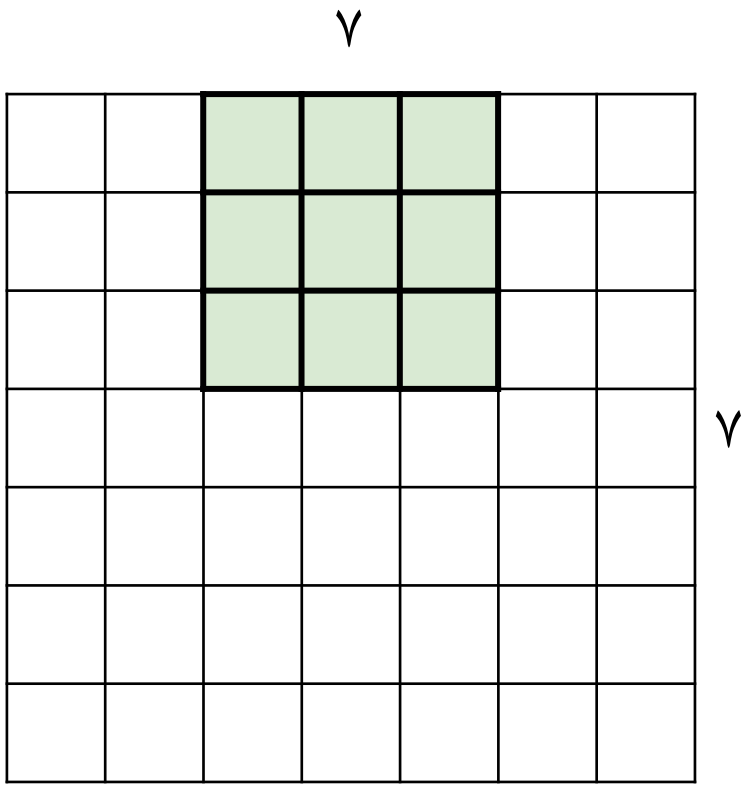
لایه کانولوشن: یک نگاه دقیق‌تر



ورودی ۷×۷

فیلتر ۳×۳

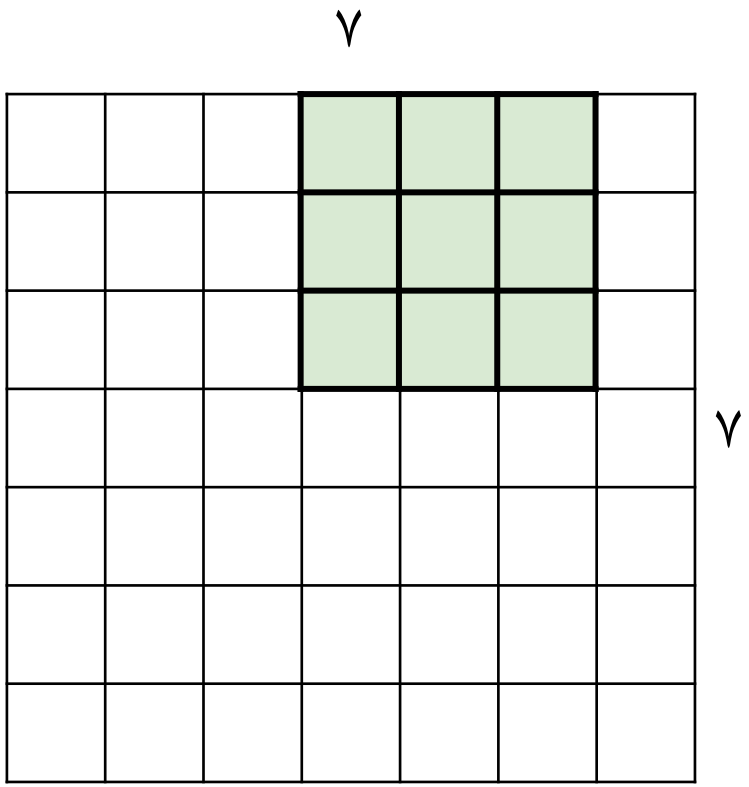
لایه کانولوشن: یک نگاه دقیق‌تر



ورودی ۷×۷

فیلتر ۳×۳

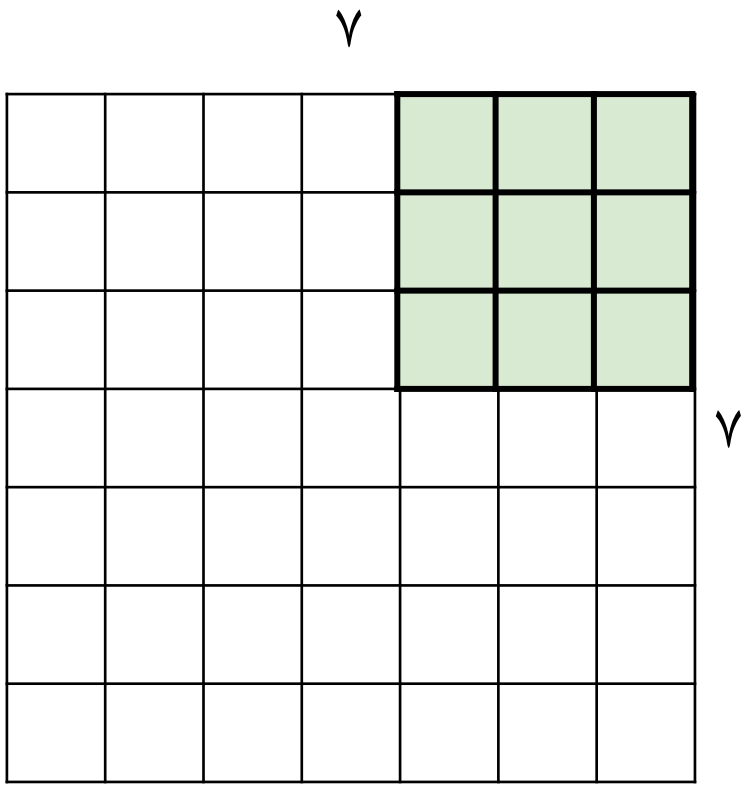
لایه کانولوشن: یک نگاه دقیق‌تر



ورودی ۷×۷

فیلتر ۳×۳

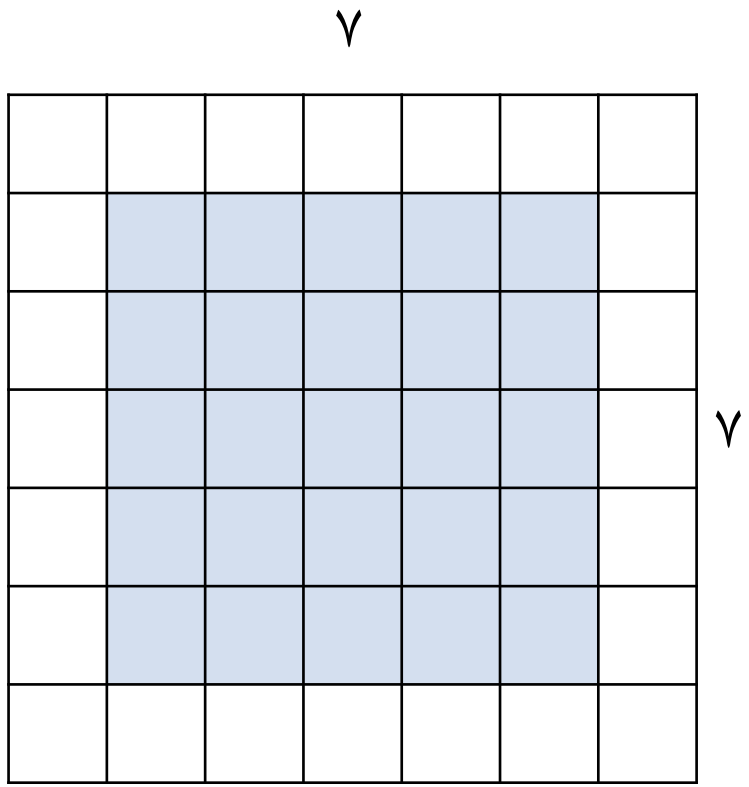
لایه کانولوشن: یک نگاه دقیق‌تر



ورودی ۷×۷

فیلتر ۳×۳

لایه کانولوشن: یک نگاه دقیق‌تر

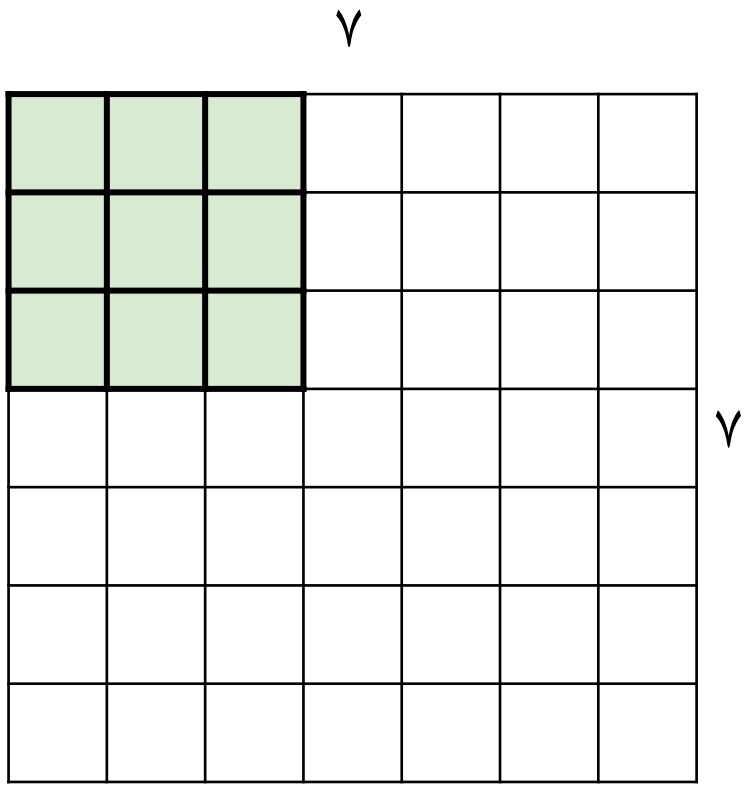


ورودی ۷×۷

فیلتر ۳×۳

⇐ **خروجی ۵×۵**

لایه کانولوشن: یک نگاه دقیق‌تر

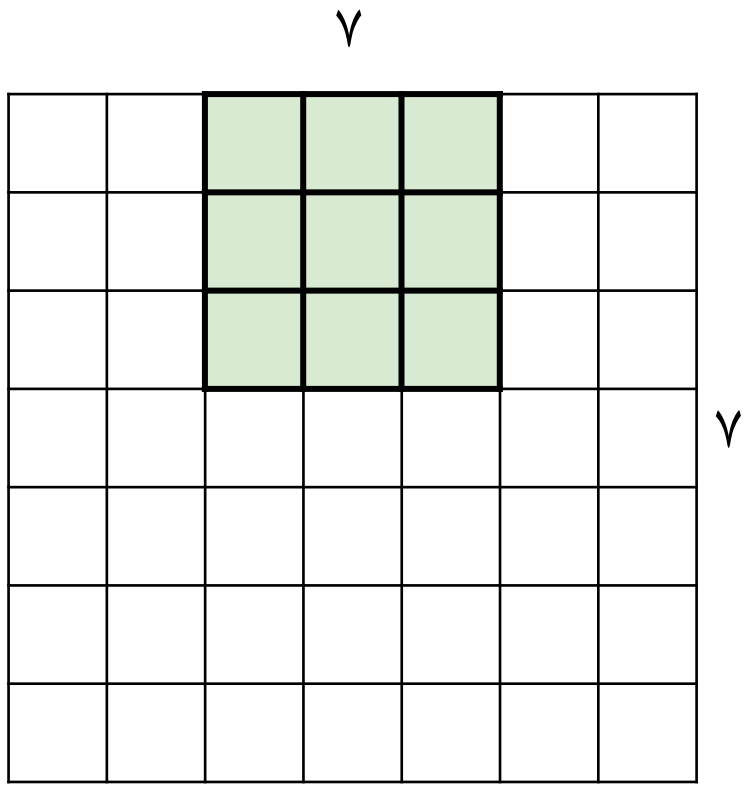


ورودی ۷×۷

فیلتر ۳×۳

اندازه گام: ۲

لایه کانولوشن: یک نگاه دقیق‌تر

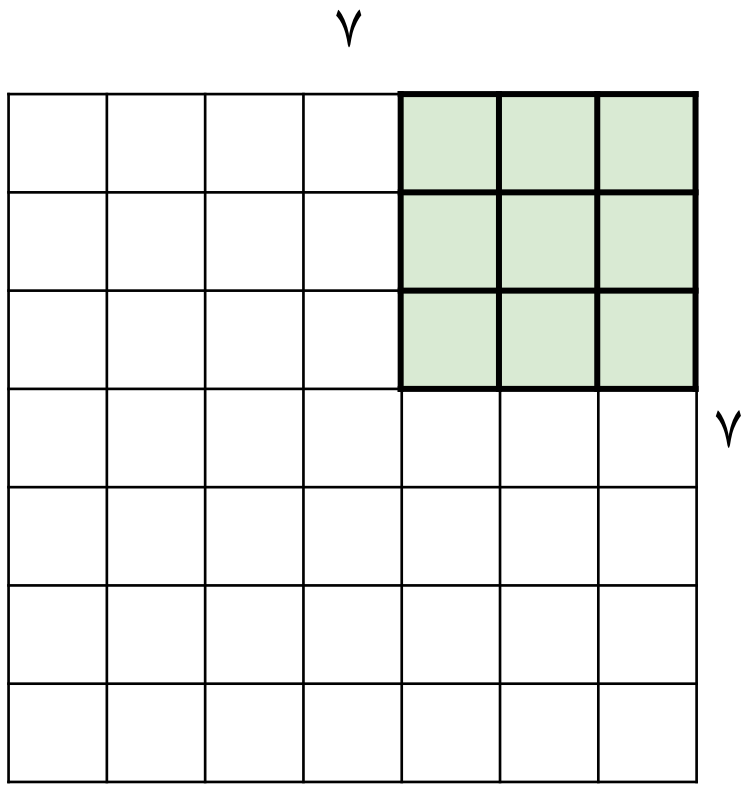


ورودی ۷×۷

فیلتر ۳×۳

اندازه گام: ۲

لایه کانولوشن: یک نگاه دقیق‌تر

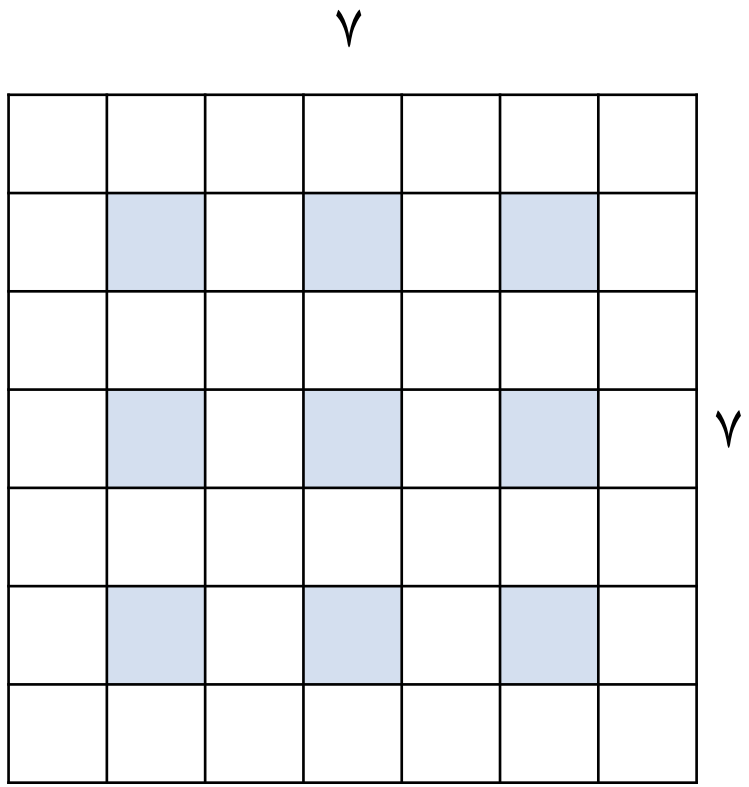


ورودی ۷×۷

فیلتر ۳×۳

اندازه گام: ۲

لایه کانولوشن: یک نگاه دقیق‌تر



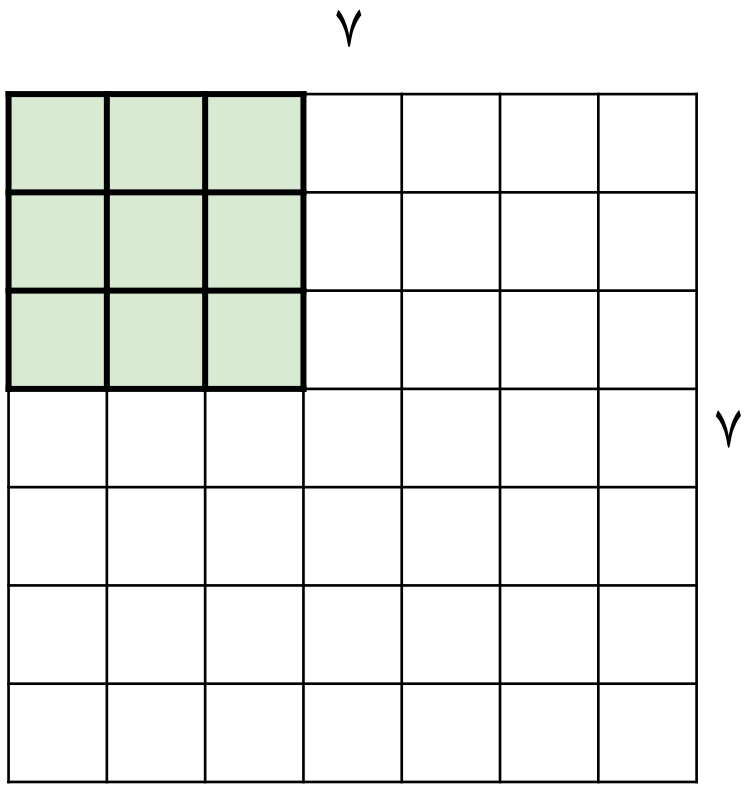
ورودی ۷×۷

فیلتر ۳×۳

اندازه گام: ۲

⇐ خروجی ۳×۳

لایه کانولوشن: یک نگاه دقیق‌تر

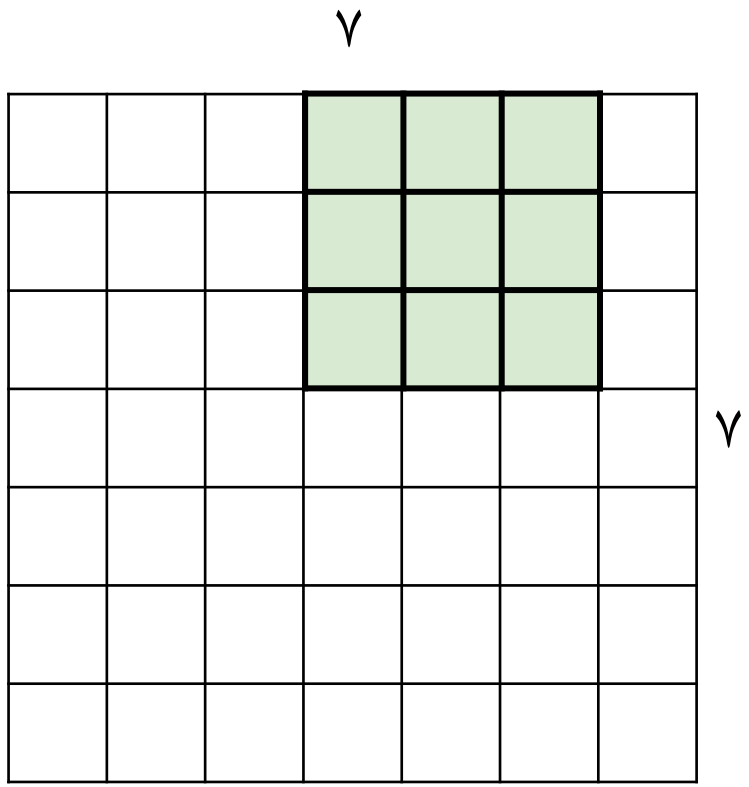


ورودی ۷×۷

فیلتر ۳×۳

اندازه گام: ۳

لایه‌ی کانولوشن: یک نگاه دقیق‌تر



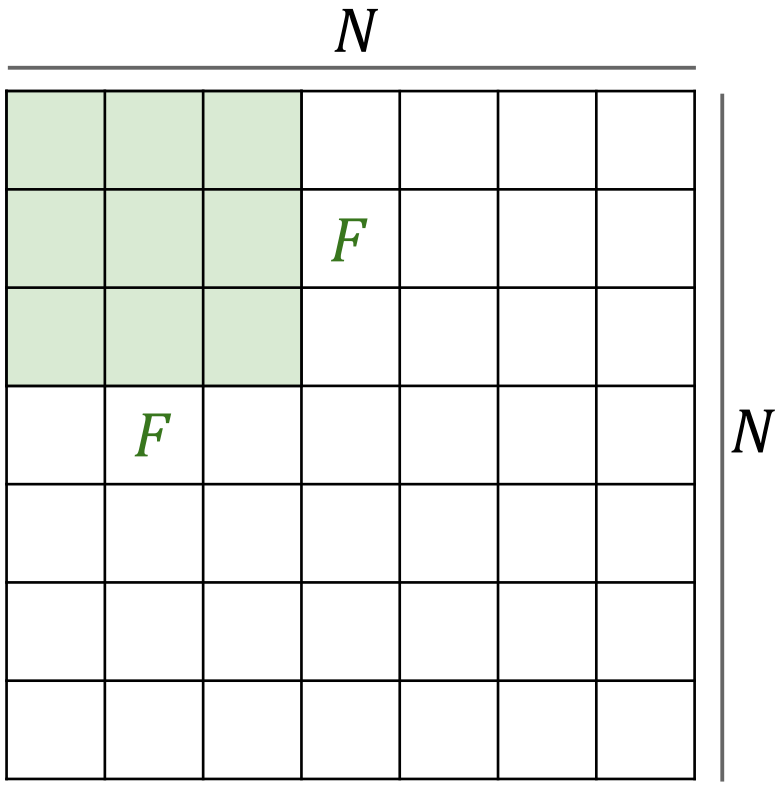
ورودی ۷×۷

فیلتر ۳×۳

اندازه گام: ۳

امکان پذیر نیست!

لایه کانولوشن: یک نگاه دقیق‌تر



اندازه خروجی:

(N - F) / stride + 1

N = 7, F = 3

stride 1 => (7 - 3) / 1 + 1 = 5
stride 2 => (7 - 3) / 2 + 1 = 3
stride 3 => (7 - 3) / 3 + 1 = 2.33

لایه کانولوشن: گسترش دادن مرزها با صفر

0	0	0	0	0	0	0	0	0
0								0
0								0
0								0
0								0
0								0
0								0
0								0
0	0	0	0	0	0	0	0	0

- مثال. تصویر ۷×۷
- فیلتر ۳×۳، با اندازه‌ی گام برابر با ۱
- **گسترش مرزها با ۱ پیکسل**، اندازه‌ی خروجی؟

خروجی ۷×۷!

لایه کانولوشن: گسترش دادن مرزها با صفر

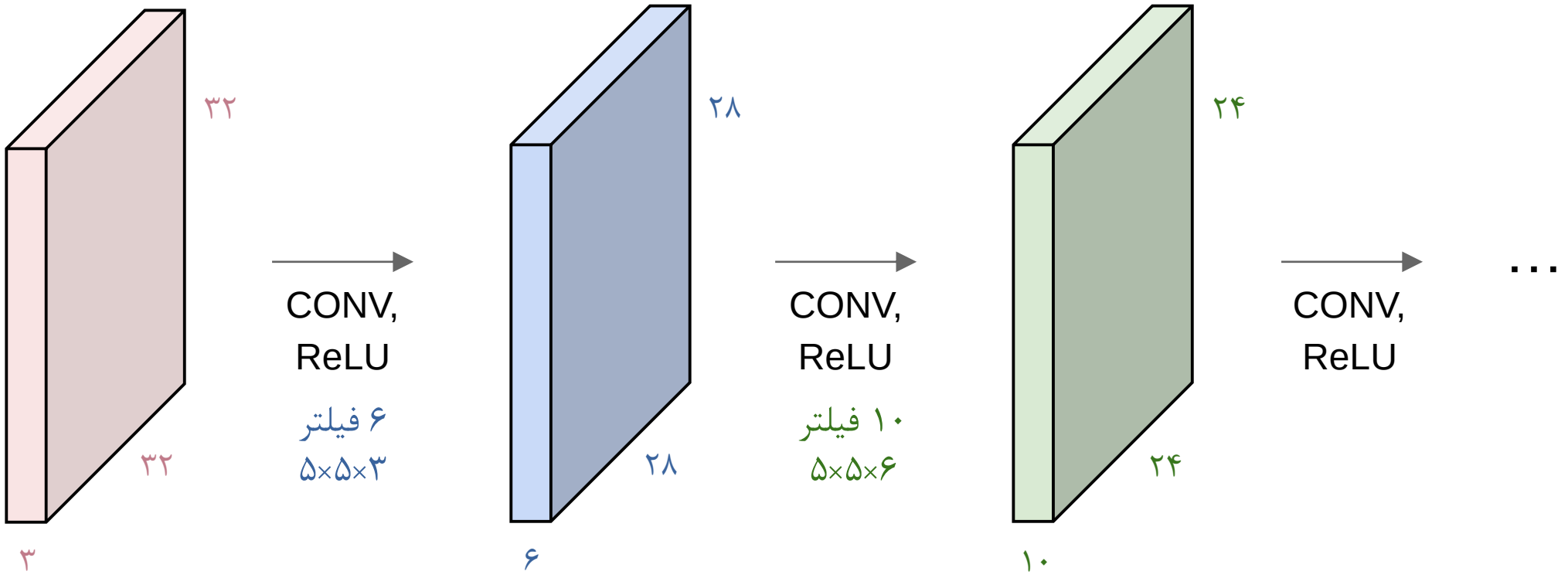
0	0	0	0	0	0	0	0	0
0								0
0								0
0								0
0								0
0								0
0								0
0								0
0	0	0	0	0	0	0	0	0

- به طور کلی، لایه‌های کانولوشن با اندازه گام برابر ۱، اندازه فیلتر $F \times F$ و صفرگذاری مرزها به اندازه $(F - 1) / 2$ متداول هستند. [اندازه‌ی ورودی بدون تغییر]

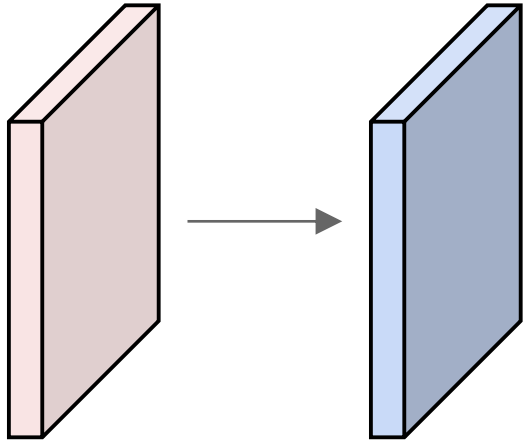
- مثال.
  - فیلتر ۳×۳ ⇐ صفرگذاری به اندازه‌ی ۱
  - فیلتر ۵×۵ ⇐ صفرگذاری به اندازه‌ی ۲
  - فیلتر ۷×۷ ⇐ صفرگذاری به اندازه‌ی ۳

یادآوری: شبکه‌های عصبی کانولوشن

- انجام کانولوشن به صورت پی در پی بر روی ورودی، باعث کوچک و کوچک‌تر شدن اندازه ورودی می‌شود.
- کاهش سریع اندازه، ایده خوبی نیست! [... → ۲۴ → ۲۸ → ۳۲]



لایه کانولوشن: مثال



- اندازه ورودی: ۳×۳۲×۳۲
- ۱۰ فیلتر ۵ × ۵، با گام ۱ و صفرگذاری ۲ پیکسلی
- اندازه خروجی؟

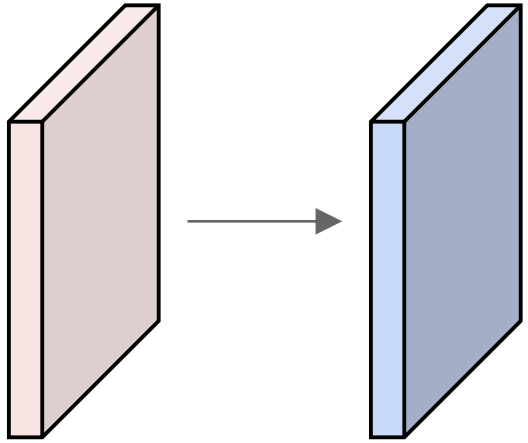
$$(۳۲ - ۵ + ۲ \times ۲) / ۱ + ۱ = ۳۲$$

اندازه هر یک از نقشه‌های فعالیت

$$۳۲ \times ۳۲ \times ۱۰$$

تعداد نقشه‌های فعالیت

لایه کانولوشن: مثال



- اندازه ورودی: ۳×۳۲×۳۲
- ۱۰ فیلتر ۵ × ۵، با گام ۱ و صفرگذاری ۲ پیکسلی
- تعداد کل پارامترها در این لایه؟

برای بایاس

$$۵ \times ۵ \times ۳ + ۱ = ۷۶$$

تعداد پارامترها برای هر فیلتر

$$۷۶ \times ۱۰ = ۷۶۰$$

تعداد کل پارامترها

لایه کانولوشن: جمع‌بندی

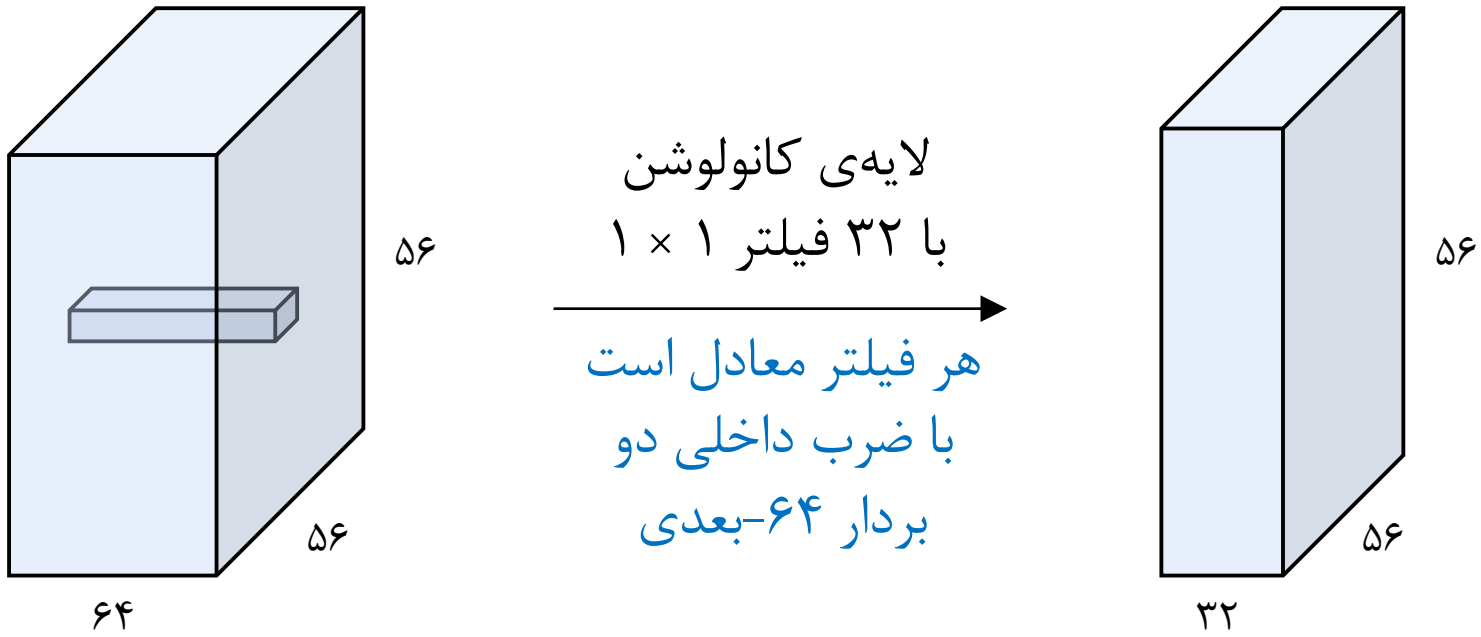
- دریافت یک ورودی با ابعاد $W_1 \times H_1 \times D_1$
- نیاز به چهار ابرپارامتر:
  - تعداد فیلترها K
  - ابعاد هر فیلتر $F \times F$
  - اندازه‌ی گام S
  - میزان صفرگذاری P
- تولید یک خروجی با ابعاد $W_2 \times H_2 \times D_2$، به گونه‌ای که:
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$
  - $D_2 = K$
- دارای $F \times F \times D_1$ وزن به ازای هر فیلتر و تعداد کل $K \times (F \times F \times D_1)$ وزن و K بایاس.

مقادیر متداول:

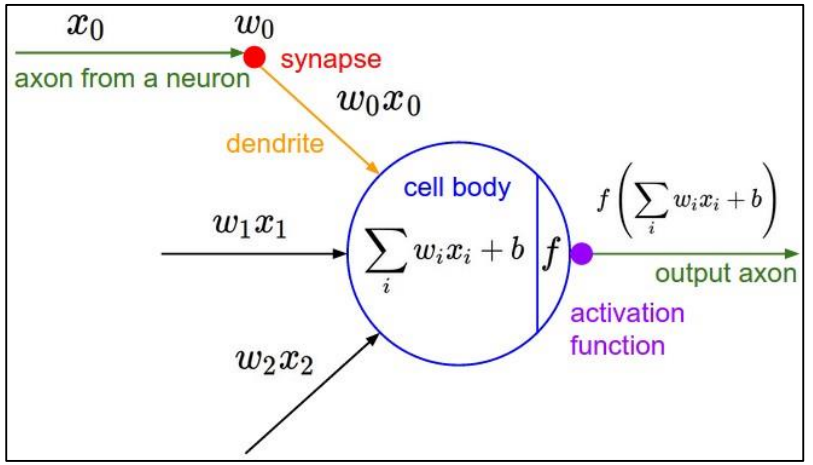
K = *توانی از ۲ (مانند ۳۲، ۶۴، ۱۲۸ و ۵۱۲)*

- F = ۳، S = ۱، P = ۱
- F = ۵، S = ۱، P = ۲
- F = ۵، S = ۲، P = ? (*هر مقدار مناسب*)
- F = ۱، S = ۱، P = ۰

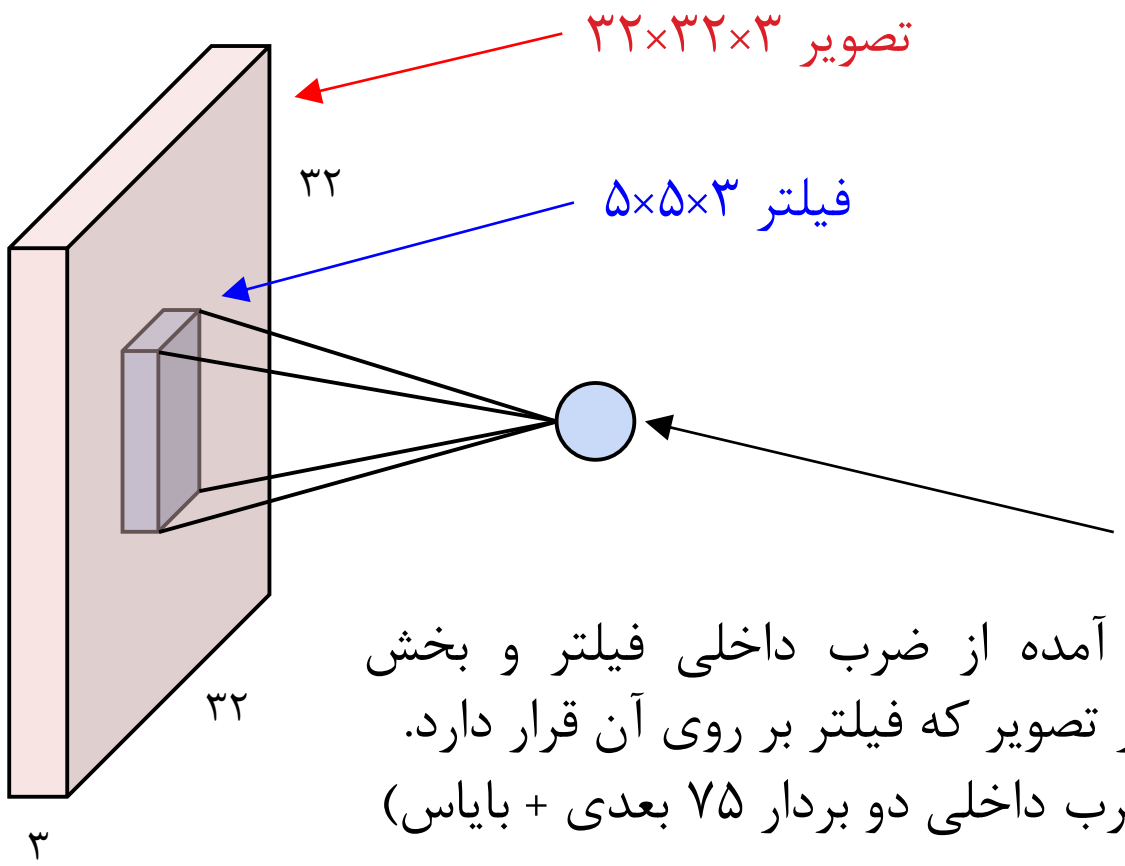
لایه کانولوشن: با فیلترهای ۱ × ۱



دیدگاه نورونی از لایه کانولوشن



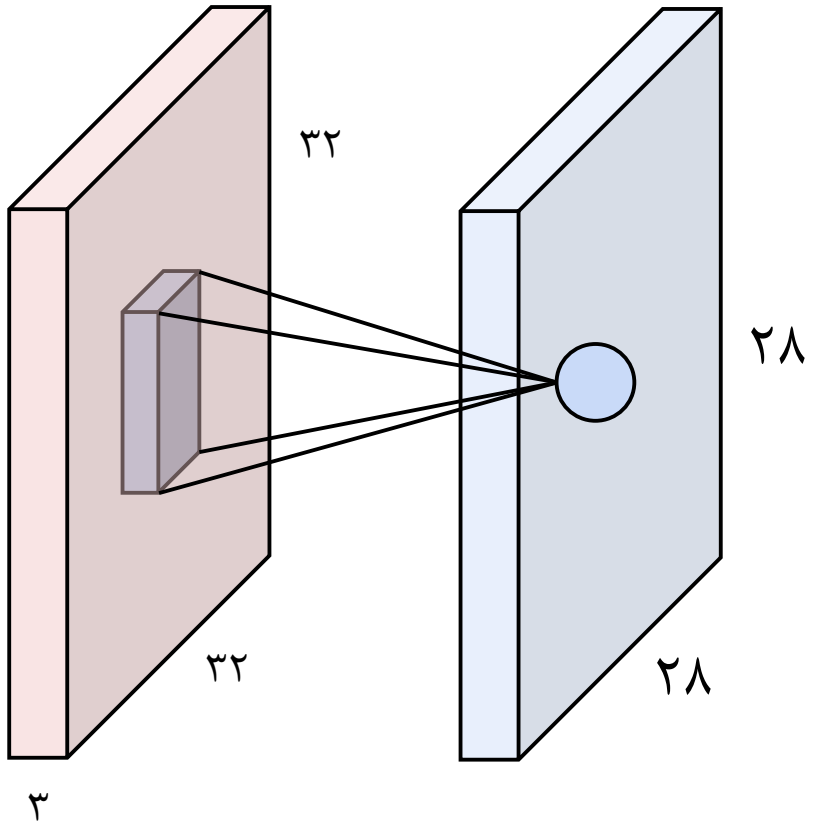
یک نورون با اتصالات محلی...

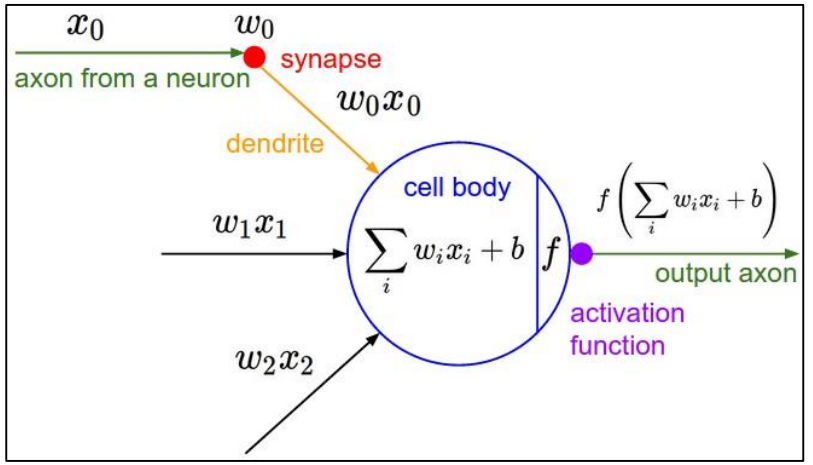


۱ عدد:

به دست آمده از ضرب داخلی فیلتر و بخش کوچکی از تصویر که فیلتر بر روی آن قرار دارد.
(یعنی، ضرب داخلی دو بردار ۷۵ بعدی + بایاس)

دیدگاه نورونی از لایه کانولوشن





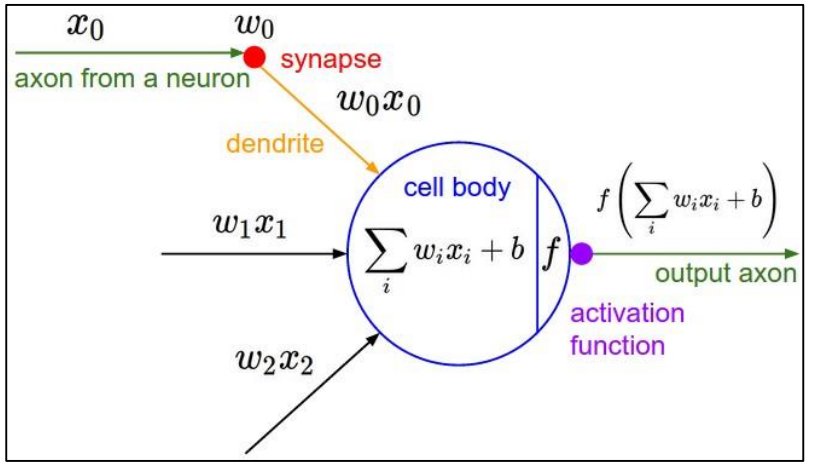
هر نقشه‌ی فعالیت یک لایه‌ی ۲۶×۲۶ از خروجی نورون‌ها است:

۱. هر نورون به یک ناحیه‌ی کوچک از ورودی متصل است.

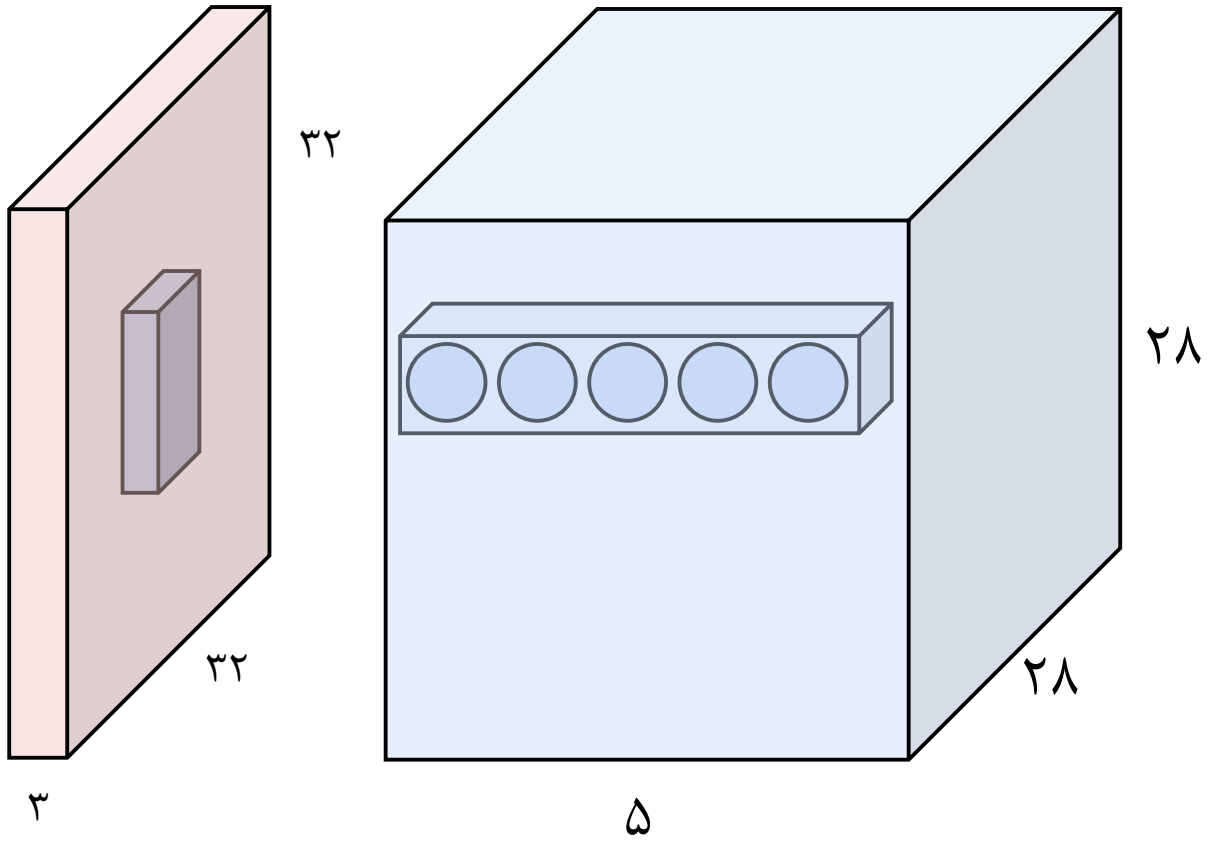
۲. همه نورون‌ها دارای پارامترهای مشترک هستند.

«فیلتر ۵×۵»: یک میدان دید ۵×۵ برای هر نورون!

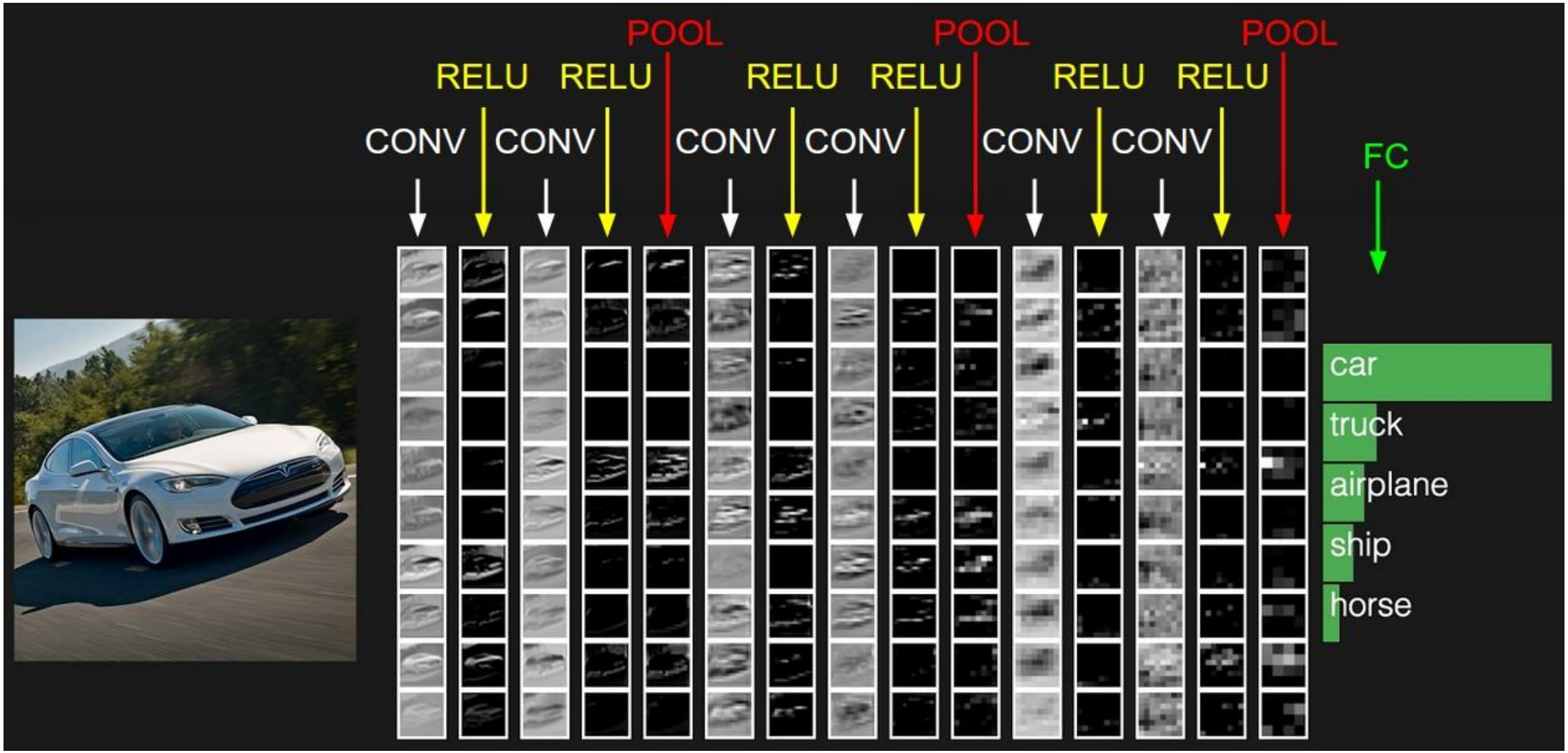
دیدگاه نورونی از لایه کانولوشن



به عنوان مثال با ۵ فیلتر، لایه کانولوشن از تعدادی نورون تشکیل شده است که به صورت یک گرید ۳-بعدی قرار گرفته‌اند.
(۵ × ۲۸ × ۲۸)

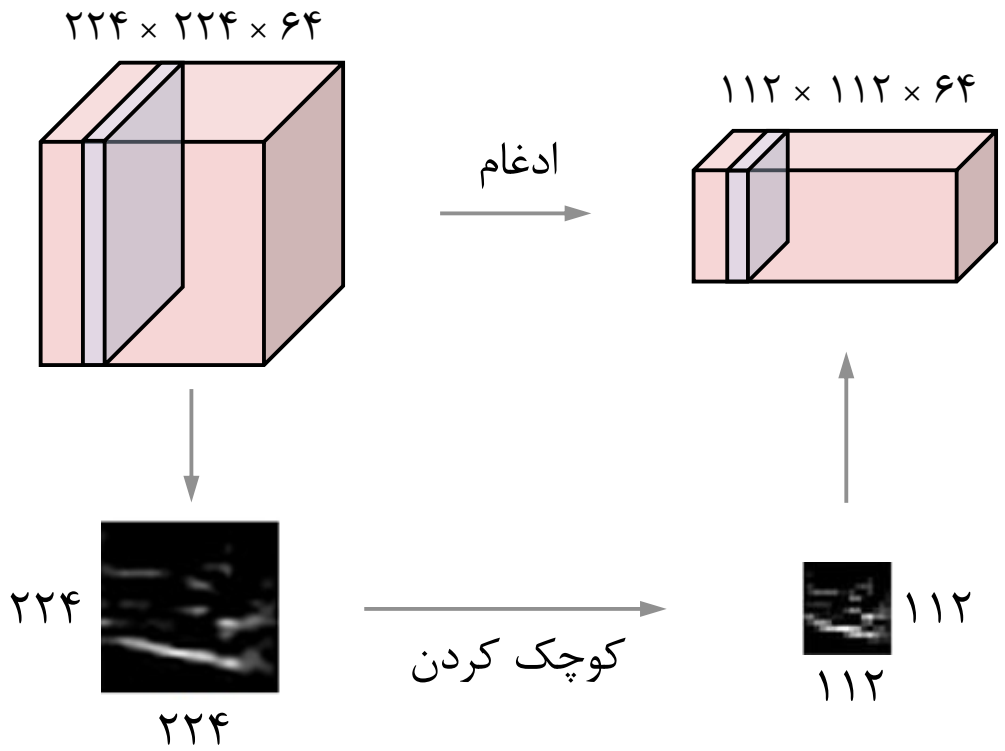


شبکه‌های عصبی کانولوشن: لایه‌های ادغام



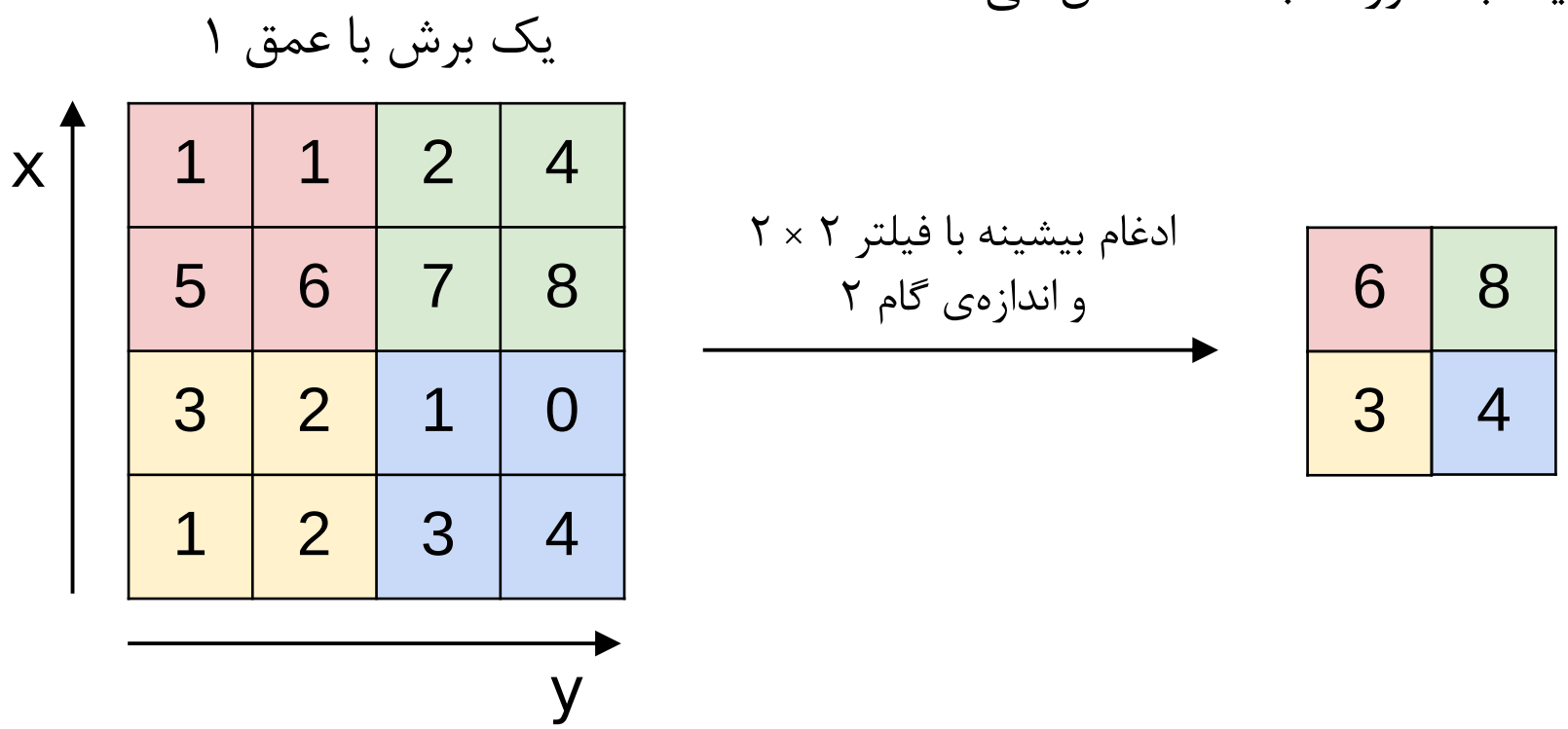
شبکه‌های عصبی کانولوشن: لایه‌های ادغام

- لایه ادغام. کوچک‌تر کردن بازنمایی!
  - بر روی هر نقشه فعالیت به صورت جداگانه عمل می‌کند.



شبکه‌های عصبی کانولوشن: ادغام بیشینه

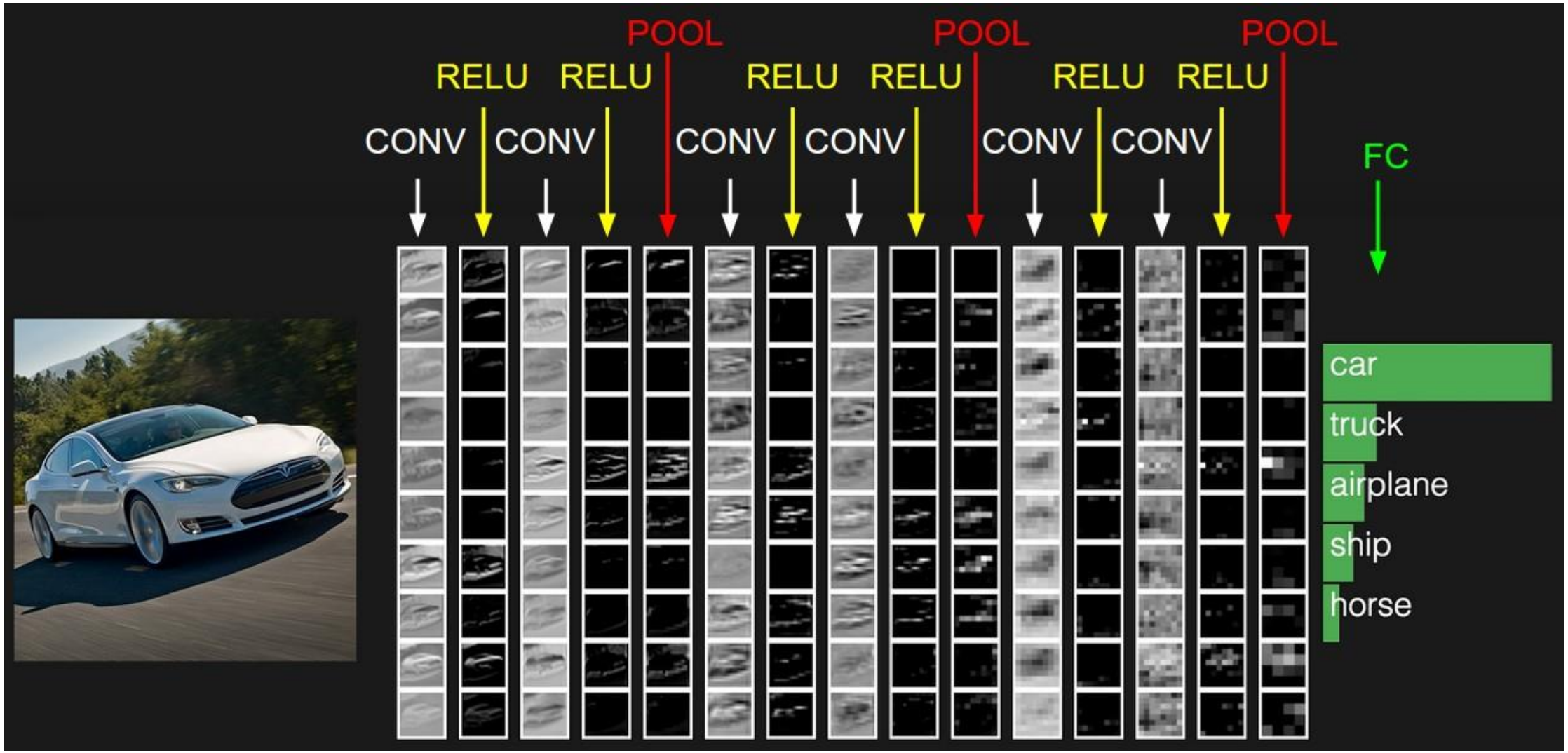
- لایه‌ی ادغام. کوچک‌تر کردن بازنمایی!
  - بر روی هر نقشه فعالیت به صورت جداگانه عمل می‌کند.



لایه ادغام: جمع‌بندی

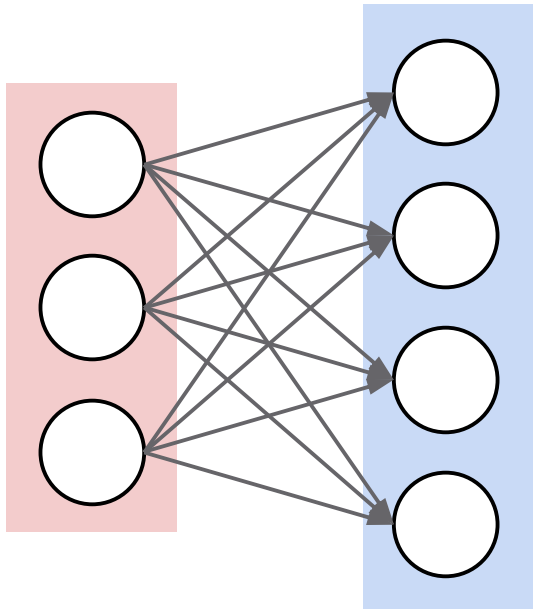
- دریافت یک ورودی با ابعاد $W_1 \times H_1 \times D_1$
- نیاز به سه ابرپارامتر:
  - ابعاد هر فیلتر $F \times F$ [مانند ۲ یا ۳]
  - اندازه‌ی گام S
- تولید یک خروجی با ابعاد $W_2 \times H_2 \times D_2$، به گونه‌ای که:
  - $W_2 = (W_1 - F)/S + 1$
  - $H_2 = (H_1 - F)/S + 1$
  - $D_2 = D_1$
- تعداد پارامترها برابر است با صفر!
- توجه. استفاده از صفرگذاری برای لایه‌های ادغام متداول نیست.

شبکه‌های عصبی کانولوشن: لایه‌های کاملاً متصل



شبکه‌های عصبی کانولوشن: لایه‌های کاملاً متصل

- لایه‌های کاملاً متصل. شامل نورون‌هایی که به تمام ورودی‌ها وصل هستند!
  - همانند لایه‌های به کار رفته در یک شبکه عصبی معمولی.



شبکه‌های عصبی کانولوشن: اجرای یک نمونه نمایشی

اجرای یک نمونه‌ی نمایشی بر روی مجموعه داده‌ی CIFAR-10

شبکه‌های عصبی کانولوشن: جمع‌بندی

- شبکه‌های عصبی کانولوشن. دنباله‌ای از لایه‌های CONV، POOL و FC
- گرایش به سمت فیلترهای کوچک‌تر و ساختارهای عمیق‌تر.
- گرایش به سمت حذف کامل لایه‌های ادغام و لایه‌های کاملاً متصل.
- یک الگوی رایج برای ساختار شبکه‌های کانولوشن:

[(CONV-RELU)*N-POOL?]*M-(FC-RELU)*K, SOFTMAX

  - به طوری که N حدکثر برابر با ۵، M یک مقدار بزرگ و K بین ۰ و ۲ است.
  - اما پیشرفت‌های اخیر مانند ResNet و GoogLeNet این الگو را به چالش کشیده‌اند!