یادگیری عمیق

دکتر امین گلزاری اسکوئی

a.golzari@azaruniv.ac.ir

a.golzari@tabrizu.ac.ir

https://github.com/Amin-Golzari-Oskouei

دانشگاه شهید مدنی آذربایجان

پاییز ۱۴۰۱

فصل ۲
قبل از شروع: مروری بر عناصر ریاضی تشکیل‌دهنده شبکه‌های عصبی

مطالب این فصل

- اولین نمونه شبکه عصبی
- تنسورها و عملیات تنسورها
- نحوه یادگیری شبکه‌های عصبی از طریق پس‌انتشار و گرادیان نزولی چگونه است

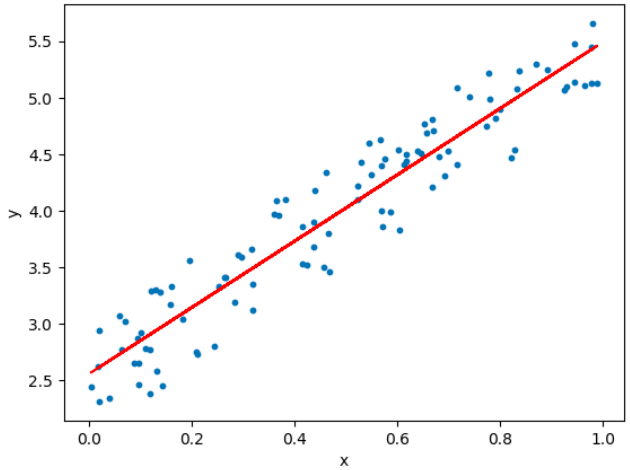
برای درک و فهم یادگیری عمیق، آشنایی با بسیاری از مفاهیم ساده ریاضی الزامی است: تنسورها، عملیات تنسورها، مشتق‌گیری، گرادیان نزولی و غیره. هدف ما در این فصل این است که بدون فنی کردن زیاده از حد موضوع، شما را با این مفاهیم آشنا کنیم.

پیش نیاز شبکه‌های عصبی

$$y = ax + b$$

$$J = \sum_{i=1}^{N} (e_i)^2$$

Min a, b

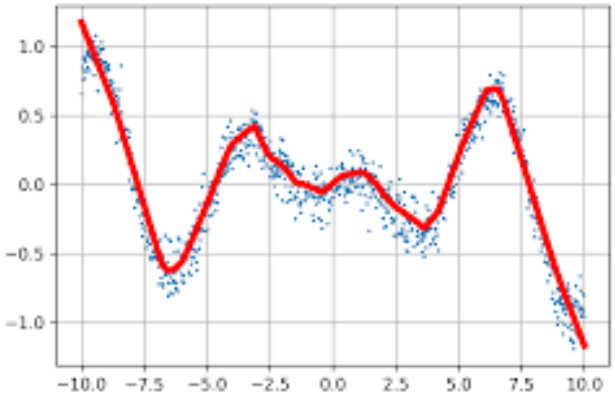


پیش نیاز شبکه‌های عصبی

$$y = ax^n + bx^{n-1} + cx^{n-2} + \cdots$$

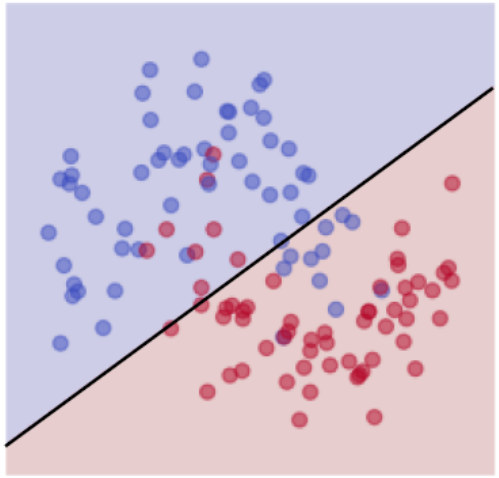
$$J = \sum_{i=1}^{N} (e_i)^2$$

Min a, b, c, …

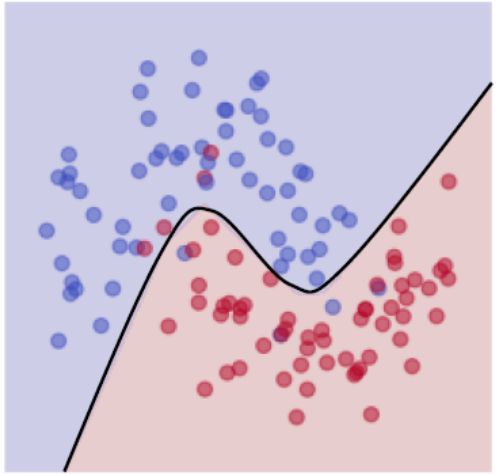


پیش نیاز شبکه‌های عصبی

- اگر داده‌ها به صورت خطی تفکیک پذیر نباشند، نیاز به افزودن ویژگی‌های مرتبه بالاتر داریم.
- از طرفی در برخی مسائل تعداد ضرایب از تعداد داده بیشتر می‌شود.



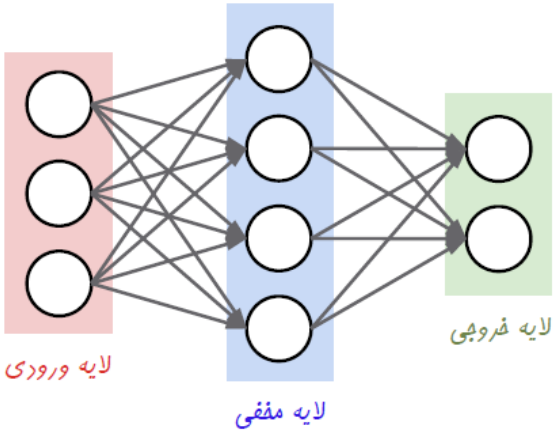
مرز تصمیم‌گیری خطی



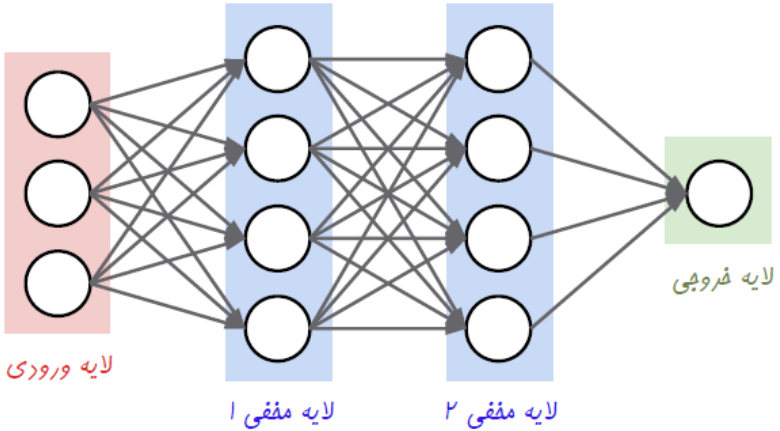
مرز تصمیم‌گیری غیرخطی

شبکه‌های عصبی

شبکه های عصبی می توانند با ترکیب ویژگی های سطح پایین، ویژگی های سطح بالای مورد نیاز خود را یاد بگیرند.



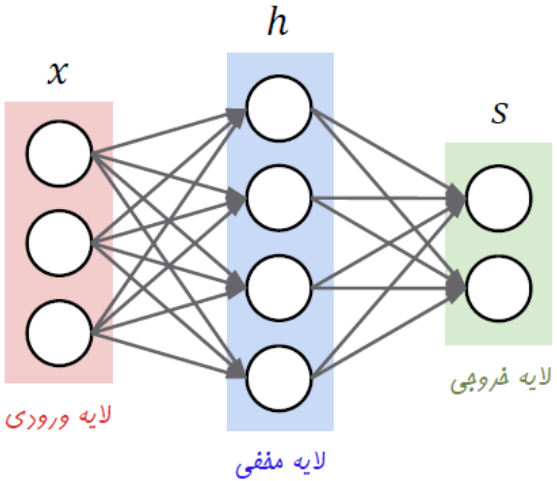
شبکه عصبی ۲ لایه



شبکه عصبی ۳ لایه

شبکه‌های عصبی

شبکه های عصبی می توانند با ترکیب ویژگی های سطح پایین، ویژگی های سطح بالای مورد نیاز خود را یاد بگیرند.



دسته‌بندی خطی

$$s = Wx + b$$

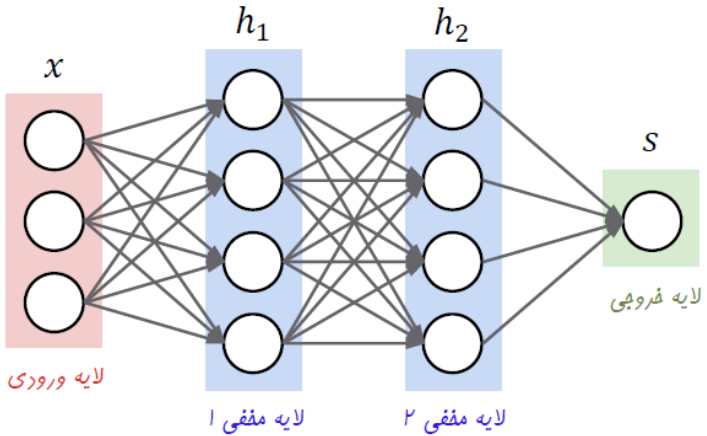
شبکه عصبی دو لایه

$$h = f(W_1 x + b_1)$$

$$s = W_2 h + b_2$$

شبکه‌های عصبی

شبکه های عصبی می توانند با ترکیب ویژگی های سطح پایین، ویژگی های سطح بالای مورد نیاز خود را یاد بگیرند.



شبکه عصبی دو لایه

$$h = f(W_1 x + b_1)$$
$$s = W_2 h + b_2$$

شبکه عصبی سه لایه

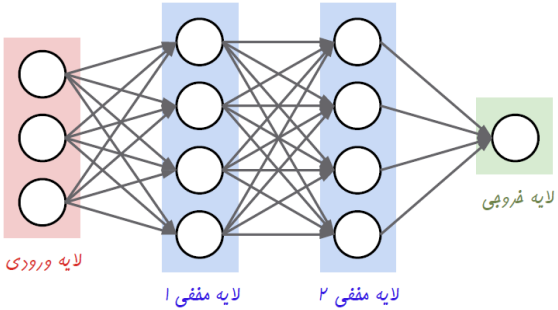
$$h_1 = f(W_1 x + b_1)$$
$$h_2 = f(W_2 h_1 + b_2)$$
$$s = W_3 h_2 + b_3$$

تابع فعال‌سازی در شبکه‌های عصبی

- شبکه های عصبی سه لایه:

$$S = W_3 f(W_2 f(W_1 x))$$

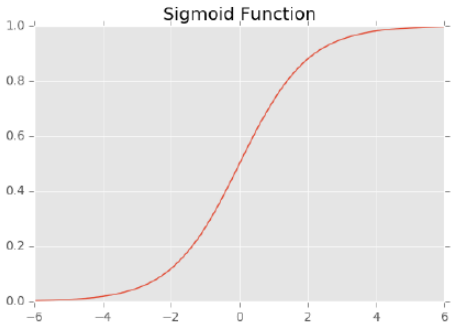
- اهمیت توابع فعال‌سازی غیرخطی در لایه های مخفی.
- عدم استفاده از توابع فعال‌سازی غیرخطی در لایه های مخفی، باعث می‌شود شبکه عصبی به یک دسته‌بند خطی ساده تبدیل گردد!



$$S = W_3 \left(W_2 \left(W_1 x\right)\right) = (W_3 W_2 W_1)x = \mathrm{W}x$$

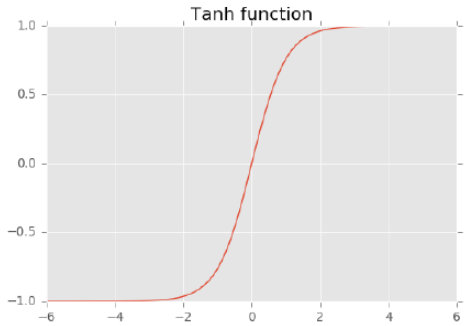
تابع فعال‌سازی در شبکه‌های عصبی

سیگموید



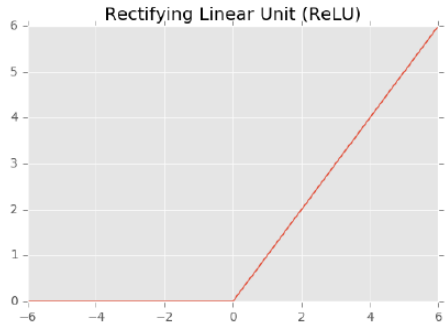
$$\sigma(x) = 1/(1 + e^{-x})$$

تانژانت هایپربولیک



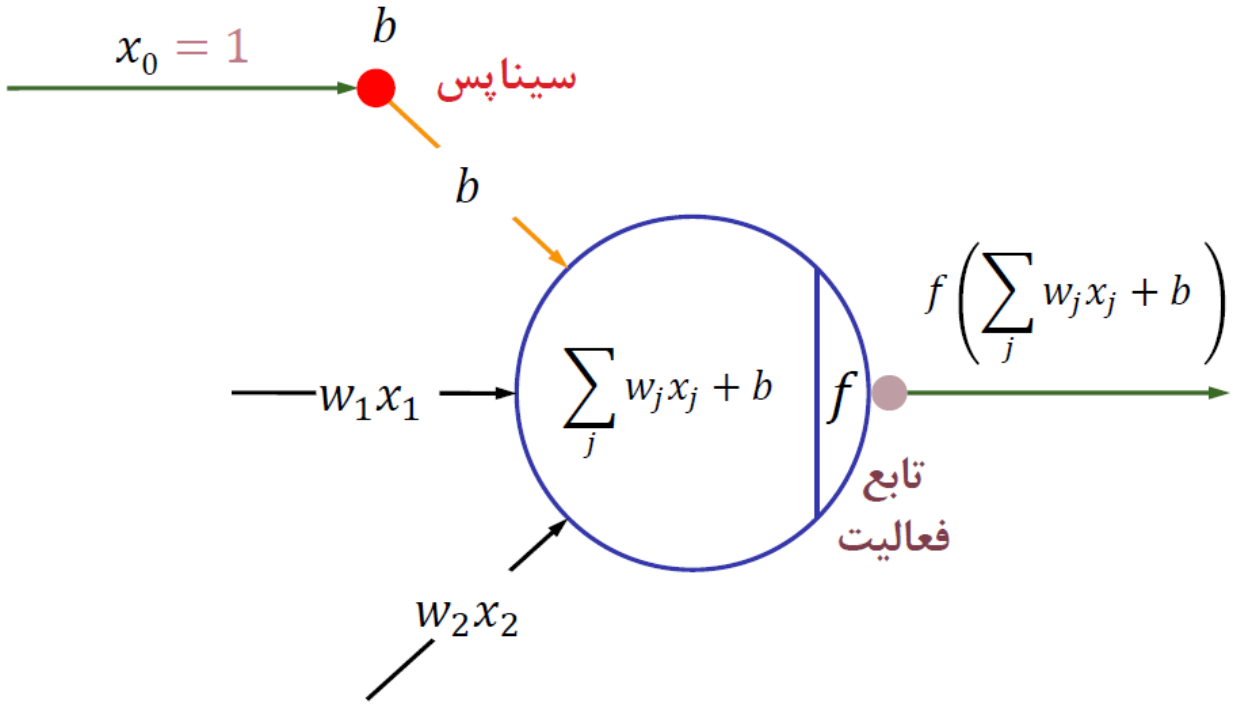
$$\tanh(x)$$

ReLU



$$\max(0, x)$$

نورون‌ها (واحدها) و شبکه‌های عصبی



نخستین نگاه به شبکه عصبی

- مسئله‌ای که می‌خواهیم در اینجا حل کنیم، دسته‌بندی تصاویر سیاه و سفید ارقام دست‌نوشته (۲۸×۲۸ پیکسل) به ۱۰ دسته (صفر تا ۹) مربوط به خودشان است.

- ما از مجموعه داده MNIST استفاده خواهیم کرد که در جامعه یادگیری ماشین مشهور بوده و قدمت آن به اندازه قدمت خود این رشته است و بارها و بارها مورد مطالعه قرارگرفته است.
- این مجموعه داده متشکل از شصت هزار تصویر آموزشی به علاوه ده هزار تصویر برای آزمایش است.

GOING TIME

2_1_a_first_look_at_a_neural_network

تنسور

- تمامی سامانه‌های موجود یادگیری ماشین از تنسورها به عنوان ساختار پایه‌ی داده‌هایشان استفاده می‌کنند.
- تنسورها اهمیت بنیادی در این رشته دارند (تا اندازه‌ای مهم و بنیادین هستند که تنسورفلو ی گوگل بر اساس آن‌ها نام‌گذاری شده است).
- در اصل، تنسور آرایه‌های چند بعدی (برای داده‌های عددی) است.
  - اسکالرها (تنسورهای صفر بعدی)
  - بردارها (تنسورهای یک‌بعدی)
  - ماتریس (تنسورهای ۲ بعدی)
  - تنسورهای سه‌بعدی و تنسورهای ابعاد بالاتر

صفات تنسور

یک تنسور با سه صفت کلیدی تعریف می‌شود:

- **تعداد محورها (رتبه):** به عنوان مثال، یک تنسور سه‌بعدی دارای سه محور است و یک ماتریس دو محور دارد. در کتابخانه‌های پایتون، مانند نام‌پای، تعداد محور یک تنسور در صفت ndim آن ثبت می‌شود.
- **شکل :** یک چندتایی از اعداد صحیح است که تعداد ابعاد تنسورها در طول هر محور را توضیح می‌دهد.
- **نوع داده‌ها (عموماً در کتابخانه‌های پایتون dtype نامیده می‌شود):** این نوع داده‌ای است که در تنسور قرار دارد؛ به عنوان مثال، نوع تنسور می‌تواند float32، unit8، float64و غیره باشد. در موارد بسیار نادر، ممکن است یک تنسور charببینید. توجه داشته باشید که تنسورهای رشته‌ای در نام‌پای (و در اکثر کتابخانه‌های دیگر) وجود ندارند، چراکه تنسورها در تکه‌های حافظه از پیش تخصیص‌یافته و پیوسته حافظه قرار می‌گیرند و رشته‌ها (دنباله‌ها) به‌واسطه طول متغیرشان، مانع استفاده از این پیاده‌سازی هستند.

صفات تنسور

```
>>> print(train_images.ndim)
3
```

```
>>> print(train_images.shape)
(60000, 28, 28)
```

```
>>> print(train_images.dtype)
uint8
```

صفات تنسور

```
>>> import numpy as np
>>> x = np.array(12)
>>> x
array(12)
>>> x.ndim
0
```

```
>>> x = np.array([[5, 78, 2, 34, 0],
                  [6, 79, 3, 35, 1],
                  [7, 80, 4, 36, 2]])
>>> x.ndim
2
```

```
>>> x = np.array([12, 3, 6, 14])
>>> x
array([12, 3, 6, 14])
>>> x.ndim
1
```

1D tensor, and 4D vector!

```
>>> x = np.array([[[5, 78, 2, 34, 0],
                   [6, 79, 3, 35, 1],
                   [7, 80, 4, 36, 2]],
                  [[5, 78, 2, 34, 0],
                   [6, 79, 3, 35, 1],
                   [7, 80, 4, 36, 2]],
                  [[5, 78, 2, 34, 0],
                   [6, 79, 3, 35, 1],
                   [7, 80, 4, 36, 2]]])
>>> x.ndim
3
```

مثال‌هایی از تنسورهای داده در دنیای واقعی

داده‌های برداری

مجموعه داده آماری افراد که در آن سن، کدپستی و درآمد هر فرد در نظر گرفته می‌شود. هر فرد را می‌توان به عنوان یک بردار با سه مقدار در نظر گرفت و به این ترتیب کل مجموعه داده‌ی 100 هزار نفری را می‌توان در تنسور دوبعدی با شکل (3, 100000) ذخیره کرد.

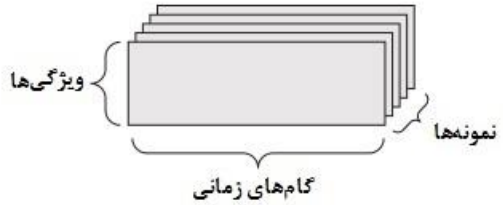
مجموعه داده اسناد نوشتاری که در آن هر سند با شمارش تعداد دفعات تکرار یک کلمه در آن (با استناد به لغت‌نامه‌ای با 20 هزار کلمه متداول) بازنمایی می‌شود. هر سند به عنوان برداری با 20 هزار مقدار (هر کلمه در لغت‌نامه یک مورد شمرده می‌شود) کدگذاری می‌شود و بنابراین مجموعه داده‌ای با 500 سند را می‌توان در تنسوری با شکل (20000, 500) ذخیره کرد.

مثال‌هایی از تنسورهای داده در دنیای واقعی

داده‌های سری زمانی یا دنباله‌ها

مجموعه داده قیمت سهام. هر دقیقه، قیمت فعلی سهام، بالاترین قیمت دقیقه قبل و پایین‌ترین قیمت دقیقه قبل را ذخیره می‌کنیم؛ بنابراین، هر دقیقه به عنوان بردار سه‌بعدی کدگذاری شده و کل روز تجاری به عنوان تنسور دوبعدی با شکل (3, 390) کدگذاری می‌شود (در هر روز تجاری 390 دقیقه داریم) و 250 روز کاری را می‌توان در تنسور سه‌بعدی با شکل (3, 390, 250) ذخیره کرد. در اینجا، هر نمونه، یک روز کاری خواهد بود.

مجموعه داده توییت‌ها که در آن هر توییت به عنوان دنباله 280 حرفی مبتنی بر یک الفبای منحصر به فرد 128 حرفی کدگذاری می‌شود. در این ساختار، هر حرف را می‌توان به عنوان بردار دودویی با اندازه 128 (یک بردار تمام-صفر به استثنای تنها عنصر 1 در اندیس مربوط به حرف موردنظر) کدگذاری کرد. سپس، می‌توان هر توییت را به عنوان تنسور دوبعدی با شکل (128, 280) کدگذاری کرده و مجموعه داده 1 میلیون توییت را در تنسوری با شکل (128, 280, 1000000) ذخیره کرد.



مثال‌هایی از تنسورهای داده در دنیای واقعی

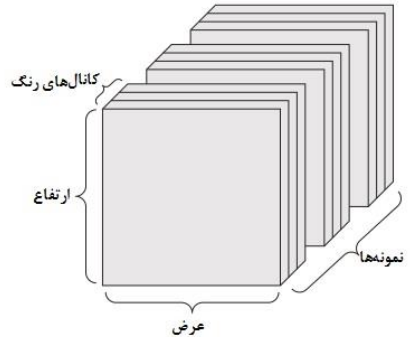
داده‌های تصویر

یک دسته 128 تایی تصاویر رنگی را می‌توان در تنسوری با شکل (3, 265, 256, 128) ذخیره کرد.

دو روش برای شکل تنسورهای تصاویر وجود دارد:

روش آخر-کانال (که تنسورفلو از آن استفاده می‌کند): (smaples, height, width, color_depth)

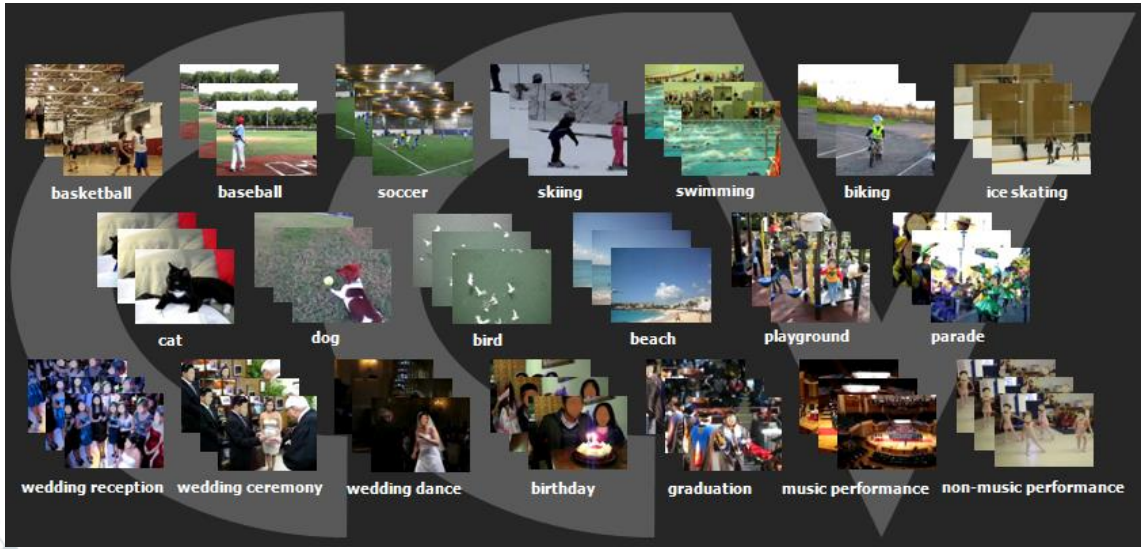
روش اول-کانال (که ثیانو از آن استفاده می‌کند): (samples, color_depth, height, width)



مثال‌هایی از تنسورهای داده در دنیای واقعی

داده‌های ویدیویی

یک کلیپ ویدیویی 60 ثانیه‌ای 256 × 144 یوتیوب با نرخ نمونه‌برداری 4 فریم در هر ثانیه، 240 فریم خواهد داشت. یک دسته چهارتایی از چنین فریم‌هایی در یک تنسور با شکل (3, 256, 144, 240, 4) ذخیره می‌شوند.



مثال‌هایی از تنسورهای داده در دنیای واقعی

Type	ndim	shape
Vector data	2	(samples, features)
Timeseries data or sequence data	3	(samples, timesteps, features)
Images	4	(samples, height, width, channels) or (samples, channels, height, width)
Video	5	(samples, frames, height, width, channels) or (samples, frames, channels, height, width)

به طورکلی، اولین محوری که در تمامی تنسورهای داده در یادگیری عمیق با آن مواجه خواهید شد، محور نمونه خواهد بود.

مفهوم دسته‌های داده

مدل‌های یادگیری عمیق کل مجموعه داده را یکجا پردازش نمی‌کنند؛ بلکه داده‌ها را به دسته‌های کوچک تقسیم می‌کنند.

```
batch = train_images[:128]
```

```
batch = train_images[128:256]
```

```
batch = train_images[128 * n:128 * (n + 1)]
```

کار با تنسورها در نام‌پای

انتخاب اجزای بخصوص در تنسور، برش تنسور نامیده می‌شود که در اینجا به بررسی آن در آرایه‌های نام‌پای می‌پردازیم.

```
>>> my_slice = train_images[10:100]
>>> print(my_slice.shape)
(90, 28, 28)
```

```
my_slice = train_images[:, 14:, 14:]
```

انتخاب ۱۴ × ۱۴ پیکسل در گوشه راست و پایین تمامی تصاویر

```
my_slice = train_images[:, 7:-7, 7:-7]
```

به دسته‌های ۱۴ × ۱۴ پیکسل که در وسط قرارگرفته‌اند

چرخ‌دنده‌های شبکه‌های عصبی: عملیات تنسور

- تمامی تبدیل‌هایی را که شبکه‌های عصبی عمیق یاد می‌گیرند می‌توان به تعداد انگشت‌شماری عملیات تنسور کاهش داد. . به عنوان مثال، می‌توان تنسورها را باهم جمع زد، آن‌ها را در هم ضرب کرد و الی‌آخر.
- به عنوان مثال، تکه کد زیر یک بردار با 784 ویژگی را دریافت و به 512 بعدی نگاشت می‌کند.

```
layers.Dense(512, activation='relu', input_shape=(28 * 28,))
```

چرخ‌دنده‌های شبکه‌های عصبی: عملیات تنسور

- لایه زیر را می‌توان به عنوان تابعی تفسیر نمود که یک تنسور دوبعدی را به عنوان ورودی گرفته و تنسور دوبعدی دیگری را بازمی‌گرداند که در حقیقت بازنمایی جدیدی برای تنسور ورودی است. به طور اختصاصی، تابع به‌قرار زیر است (که در آن W تنسور دوبعدی و b بردار است و هر دو به عنوان صفت‌هایی برای لایه هستند):

```
output = relu(dot(input, W) + b)
```

- در اینجا سه عملیات تنسور داریم: ضرب نقطه‌ای (dot) بین تنسور ورودی و تنسور W؛ یک جمع (+) بین تنسور دوبعدی حاصل‌شده و بردار b؛ و در نهایت، عملیات relu(x) که همان max(x, 0) است.

ضرب نقطه‌ای تنسورها

```
In [1]: import numpy as np

        a = np.array([1,2,3])
        b = np.array([4,5,6])

        a*b
```

```
Out[1]: array([ 4, 10, 18])
```

```
In [2]: np.dot(a,b)
```

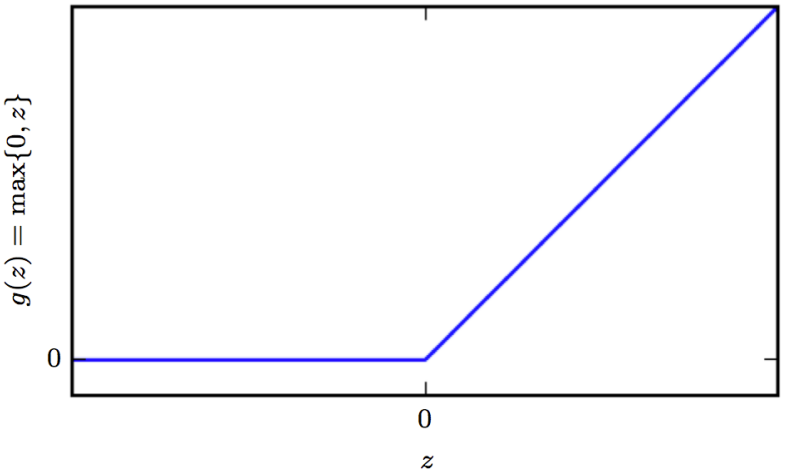
```
Out[2]: 32
```

```
In [3]: a = np.random.rand(2,3)
        b = np.random.rand(3,4)

        np.dot(a,b)
```

```
Out[3]: array([[ 1.01448459,  0.54732069,  0.78790565,  0.71547521],
               [ 0.9647976 ,  0.47166004,  0.74593664,  0.69319332]])
```

عملیات عضو به عضو

- همانند جمع بردارها، و عملیات relu
- این عملیات در پیاده‌سازی‌ها به شدت قابلیت موازی‌سازی را دارند.



نکته

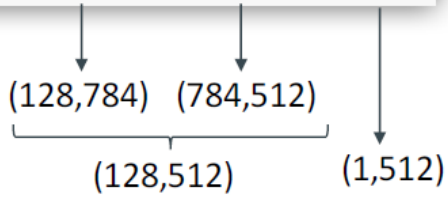
در عمل، هنگام مواجهه با آرایه‌های نام‌پای، این عملیات به عنوان توابع داخلی و بهینه نام‌پای قابل‌دسترسی هستند، توابع نام‌پای از زیر برنامه‌های ویژه جبر خطی به نام BLAS برای انجام عملیاتشان استفاده قابل‌توجهی خواهند کرد، البته اگر در کامپیوتر نصب شده باشند. BLAS عبارت است از پیاده‌سازی سطح پایین (نزدیک به سخت‌افزار)، به شدت موازی که معمولاً در فرترن یا C پیاده‌سازی می‌شوند.

```
In [1]: import numpy as np
        import time

In [2]: n = 1000000
        a = np.random.rand(n)
        b = np.random.rand(n)

In [3]: c = np.zeros(n)

        tic = time.time()
        for i in range(n):
            c[i] = a[i] + b[i]
        toc = time.time()
        print('Naive version: %.4fs'%(toc-tic))

        Naive version: 0.4858s

In [4]: tic = time.time()
        c = a+b
        toc = time.time()
        print('Vectorized version: %.4fs'%(toc-tic))

        Vectorized version: 0.0064s
```

انتشار

زمانی که تنسورهای دوبعدی که باید جمع زده شوند هم‌شکل نیستند، چه اتفاقی برای عمل جمع می‌افتد؟

```
output = relu(dot(input, W) + b)
```



در صورت امکان و در صورتی که هیچ ابهامی پیش نیاید، تنسور کوچک‌تر طوری منتشر می‌شود که با شکل تنسور بزرگ‌تر همخوانی داشته باشد.

تغییر شکل تنسور

```
>>> x = np.array([[0., 1.],
                  [2., 3.],
                  [4., 5.]])
>>> print(x.shape)
(3, 2)
>>> x = x.reshape((6, 1))
>>> x
array([[ 0.],
       [ 1.],
       [ 2.],
       [ 3.],
       [ 4.],
       [ 5.]])

 >>> x = x.reshape((2, 3))
 >>> x
 array([[ 0.,  1.,  2.],
        [ 3.,  4.,  5.]])
```

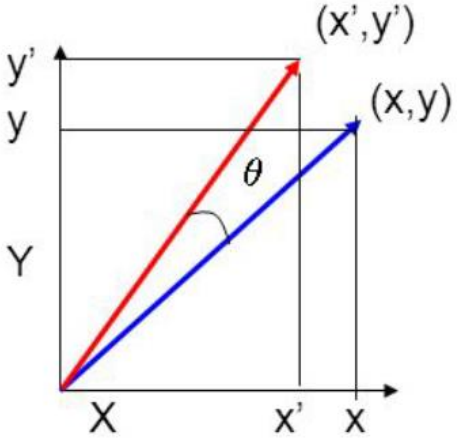
```
>>> x = np.zeros((300, 20))
>>> x = np.transpose(x)
>>> print(x.shape)
(20, 300)
```

تفسیر هندسی عملیات تنسور

تمامی عملیات تنسور، تفسیر هندسی دارند. به عنوان مثال

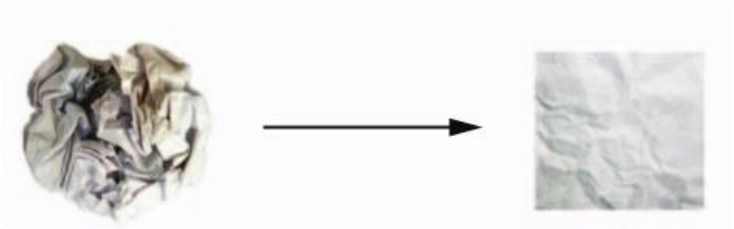
rotation matrix

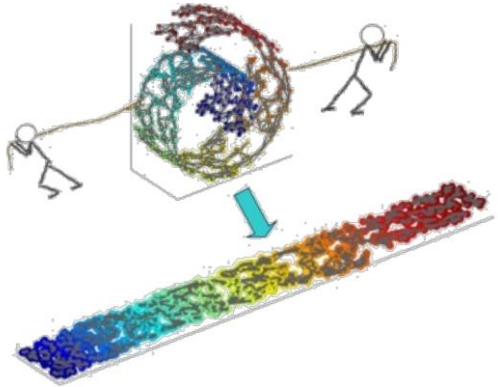
$$\begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$



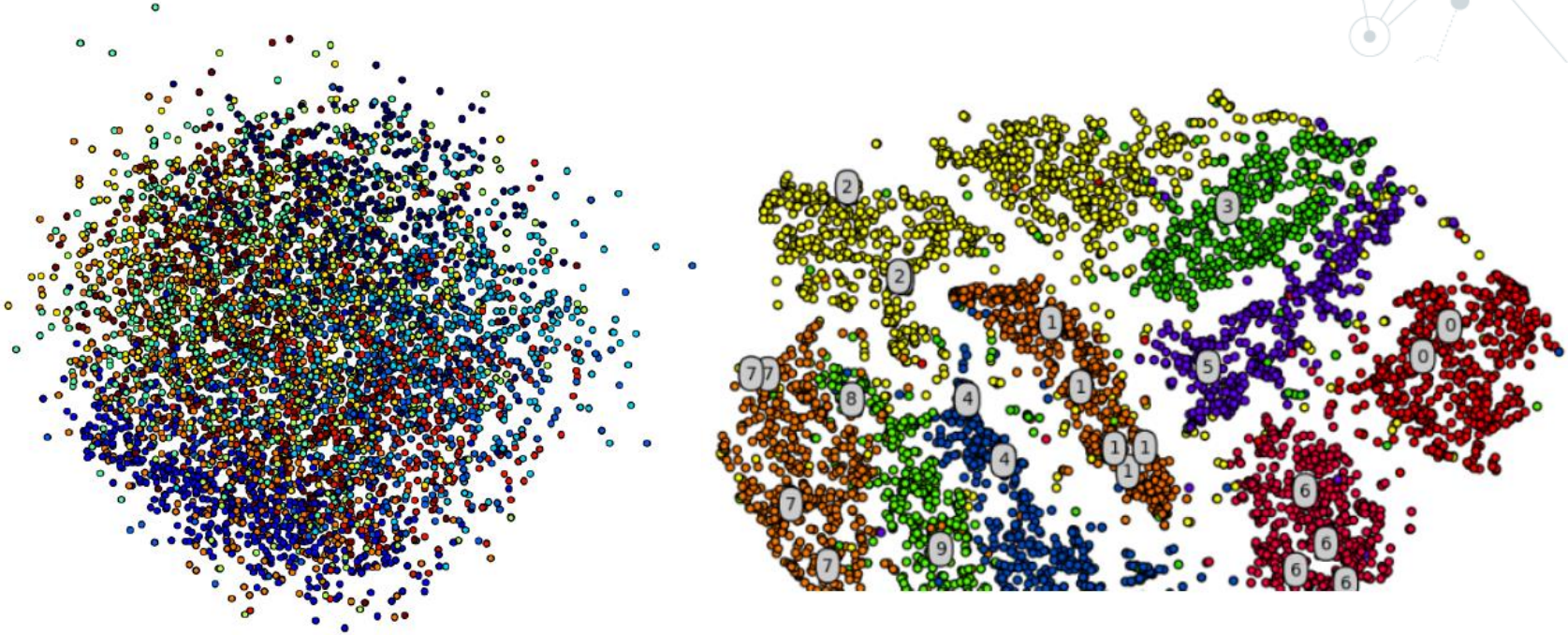
تفسیر هندسی یادگیری عمیق

- شبکه‌های عصبی به طور کامل متشکل از عملیات تنسور بوده و این عملیات تبدیل هندسی نمونه‌های ورودی هستند.
- شبکه عصبی را می‌توان تبدیل هندسی بسیار پیچیده در فضای ابعاد-بالا دانست که به واسطه زنجیره طولانی از گام‌های ساده پیاده‌سازی می‌شود.





تفسیر هندسی یادگیری عمیق



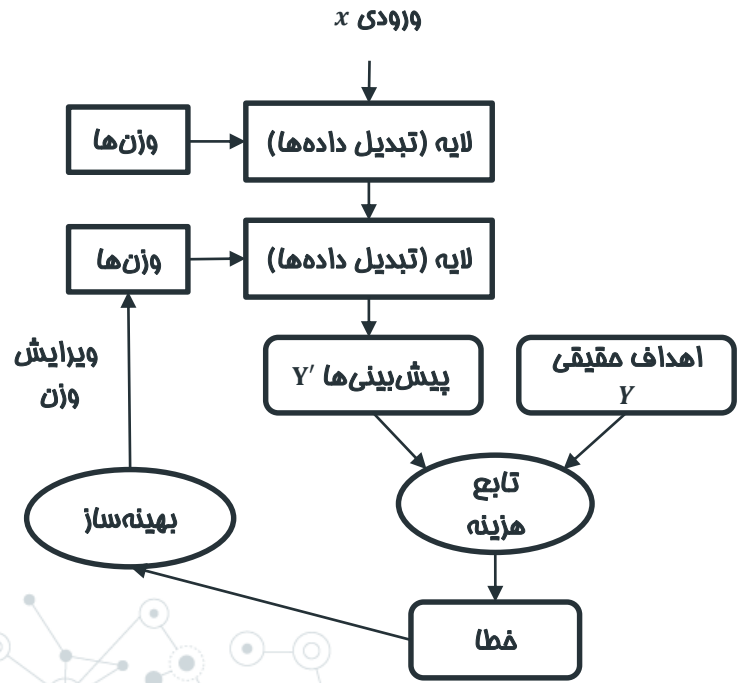
آموزش

- W و b وزن‌ها یا پارامترهای آموزش پذیر لایه نامیده می‌شوند. این وزن‌ها حاوی اطلاعاتی هستند که شبکه به واسطه دریافت داده‌های آموزشی یاد گرفته است.

```
output = relu(dot(W, input) + b)
```

- ابتدا، این ماتریس‌های حاوی وزن با مقادیر تصادفی و کوچک پر می‌شوند (مرحله‌ای که مقداردهی ابتدایی تصادفی نامیده می‌شود). مشخصاً، زمانی که W و bتصادفی هستند، نمی‌توان انتظار داشت که relu(dot(W, input)+b) خروجی مفیدی داشته باشد. خروجی‌های حاصل فاقد معنی هستند اما آن‌ها نقطه شروع هستند. بعد از این مرحله، وزن‌ها به تدریج براساس سیگنال بازخورد تنظیم می‌شوند. این تنظیم تدریجی که آموزش نیز نامیده می‌شود، در واقع همان یادگیری است که یادگیری ماشین مبتنی بر آن است.

آموزش



این مراحل را تا هر جا که ضرورت داشته باشند در حلقه تکرار کنید:

1. یک دسته از نمونه‌های آموزشی x و خروجی مورد انتظار آن‌ها را از y انتخاب کنید.
2. شبکه را روی x اجرا کنید (مرحله گذر روبه‌جلو) تا پیش‌بینی‌های y_pred به دست بیایند.
3. خطای شبکه روی دسته را محاسبه کنید، معیاری برای میزان عدم همخوانی بین y_pred و y.
4. تمامی وزن‌های شبکه را به‌گونه‌ای به‌روز کنید که موجب کاهش جزئی خطای این دسته شود.

آموزش

راه‌حل ساده برای بروزرسانی وزن‌ها

- به استثنای وزنی که مدنظر است *تمامی* وزن‌های شبکه را منجمد کنید
- مقادیر مختلف (افزایش یا کاهش) را برای وزن موردنظر امتحان کنید.
- این کار را باید برای *تمامی* وزن‌ها در شبکه تکرار کنید؛

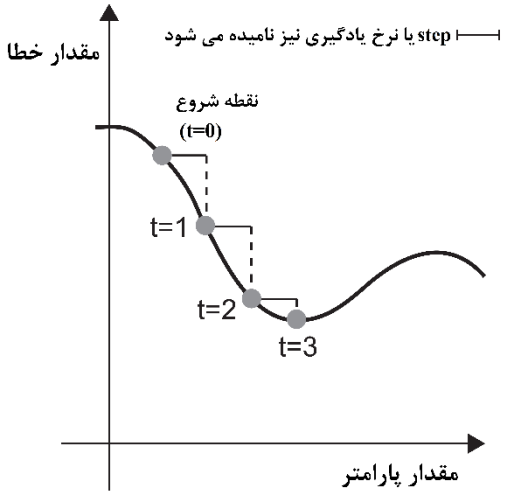
چنین رویکردی به شدت ناکارآمد خواهد بود، چرا که مجبور خواهید بود برای هر وزن (که تعدادشان بسیار زیاد است و اغلب به چندین هزار و گاهی به چند میلیون می‌رسد) دو گذر روبه‌جلو (که بسیار پرهزینه هستند) محاسبه کنید.

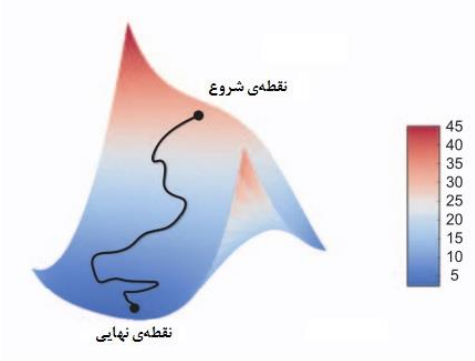
رویکرد بهتر، سود بردن از این واقعیت است که *تمامی* عملیات به کار رفته در شبکه، مشتق‌پذیر هستند. اگر گرادیان خطا را نسبت به وزن‌های شبکه محاسبه کنید، می‌توانید تغییر وزن‌ها را در خلاف جهت گرادیان انجام داده و از مقدار خطا بکاهید: گرادیان نزولی

گرادیان نزولی

گرادیان تابع خطا را با توجه به پارامترهای شبکه (گذر معکوس) محاسبه کنید.

پارامترها را به طور جزئی در خلاف جهت گرادیان تغییر دهید. به عنوان مثال W -= step * gradient
تا خطای دسته به طور جزئی کاهش پیدا کند.

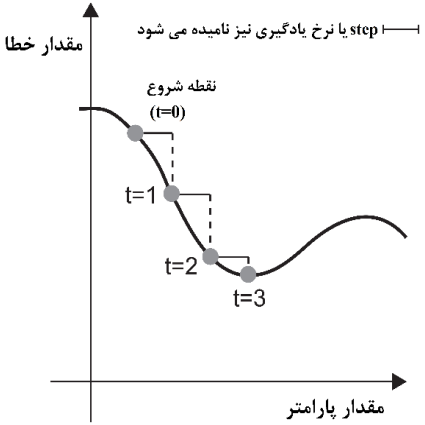


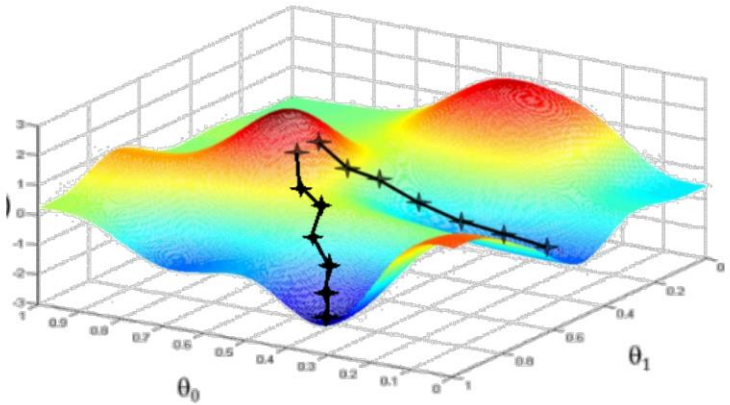


گرادیان نزولی

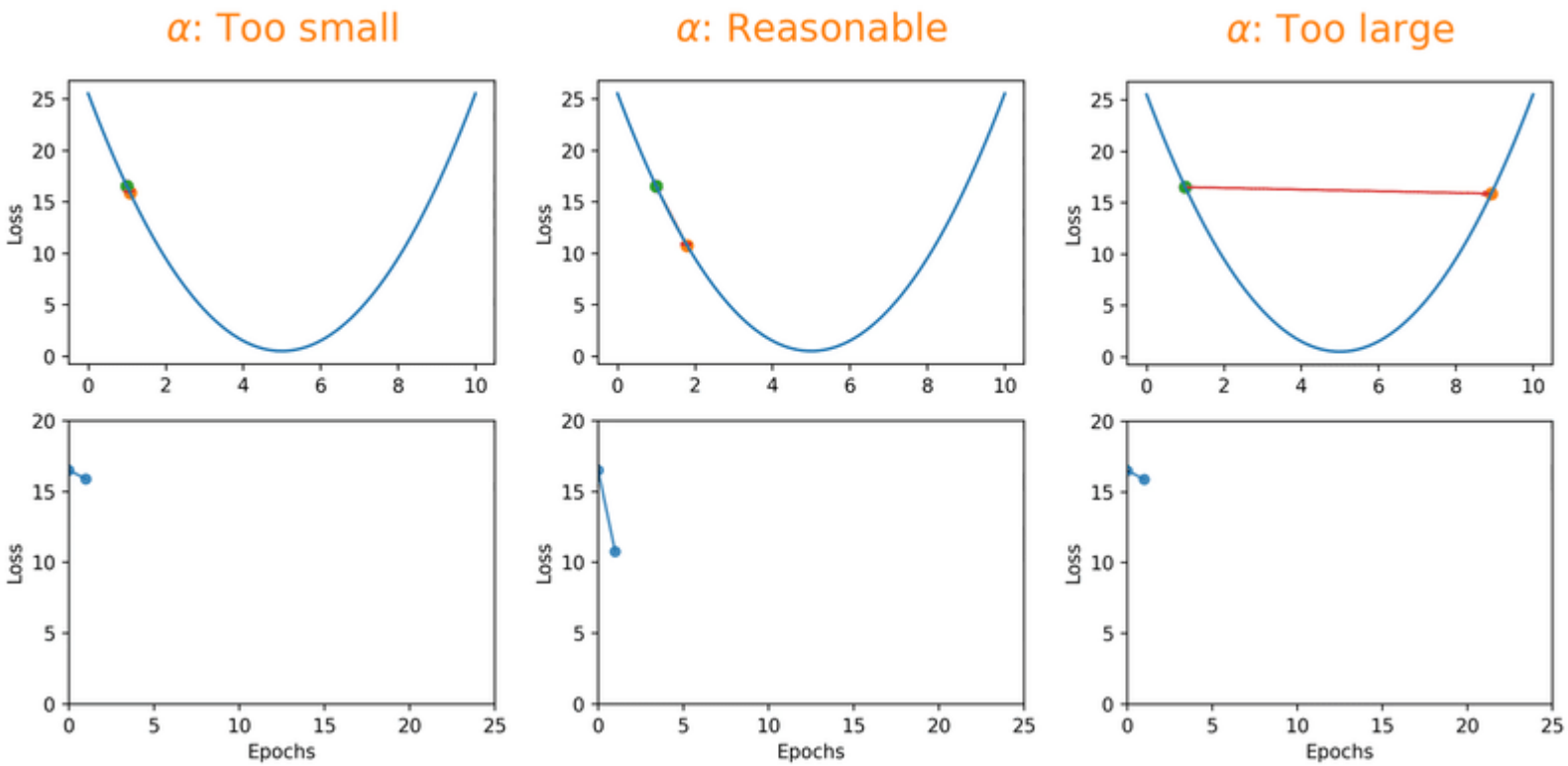
نرخ یادگیری

- خیلی کوچک: استفاده شود، فرایند بهینه‌سازی به‌جای پیش‌روی به‌سوی کمینه سراسری در کمینه محلی گیر خواهد کرد. همچنین آموزش کند می‌شود.
- خیلی بزرگ: تصادفی عمل می‌کند





گرادیان نزولی

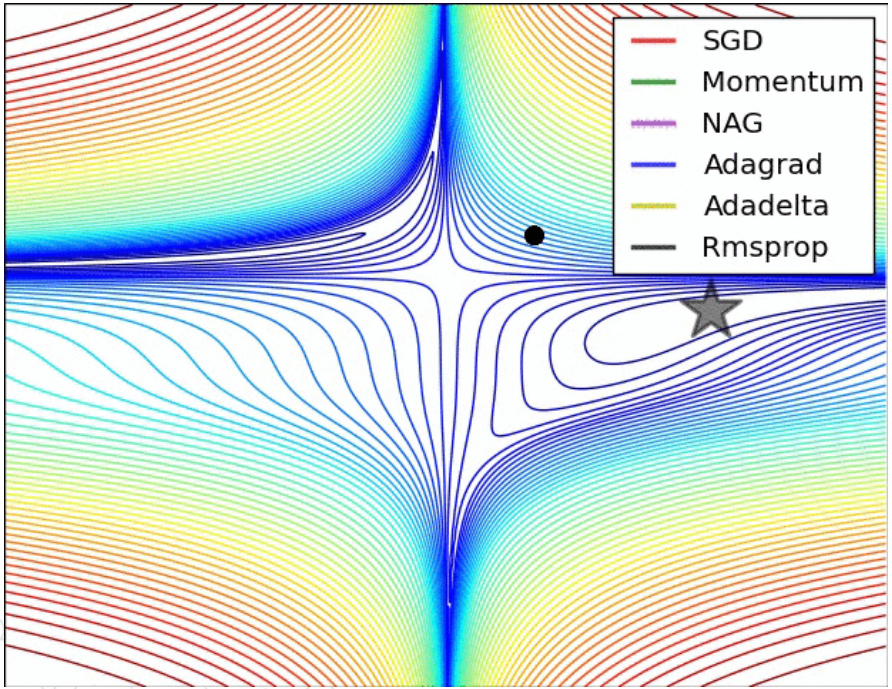


گرادیان نزولی تصادفی

- ورژن‌های مختلف گرادیان نزولی:
- گرادیان نزولی تصادفی دسته‌ای (Batch SGD): هر مرحله را روی تمامی داده اجرا می‌شود.
- گرادیان نزولی تصادفی (SGD): هر مرحله را روی یک نمونه داده اجرا می‌شود.
- گرادیان نزولی تصادفی با دسته‌های کوچک (Mini-batch SGD): هر مرحله را روی دسته‌ای کوچکی از داده‌ها اجرا می‌شود.

- حد وسط این دو روش افراطی Batch SGD و SGD، این است که از دسته‌های کوچک (Mini-batch SGD) با اندازه معقول استفاده کنید.

گرادیان نزولی تصادفی

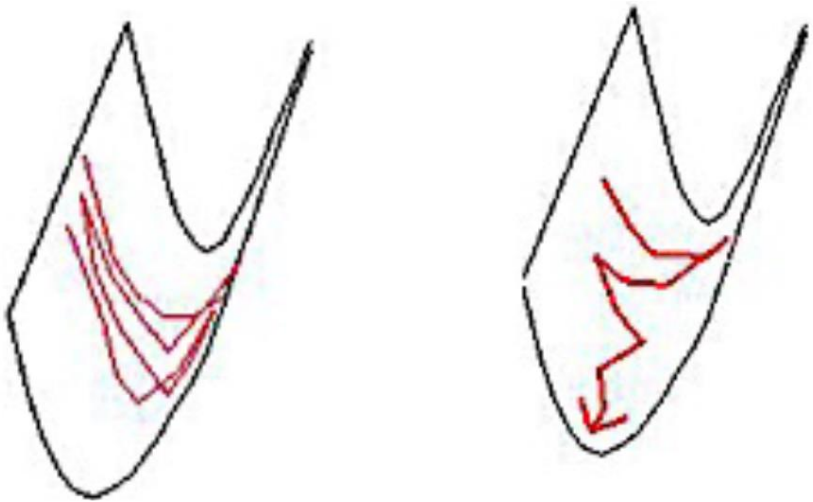


انواع مختلف روش بهینه‌سازی وجود دارند:

- SGD with momentum
- Nesterov accelerated gradient
- Adagrad
- Adadelta
- RMSProp
- Adam

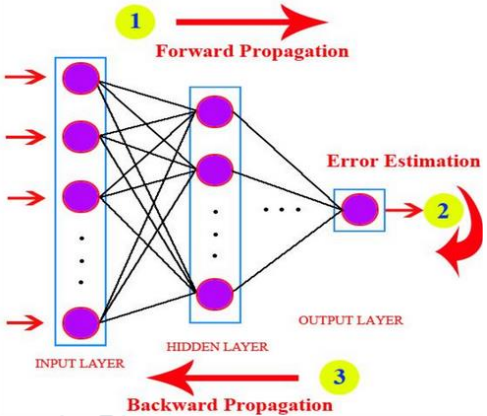
گرادیان نزولی تصادفی

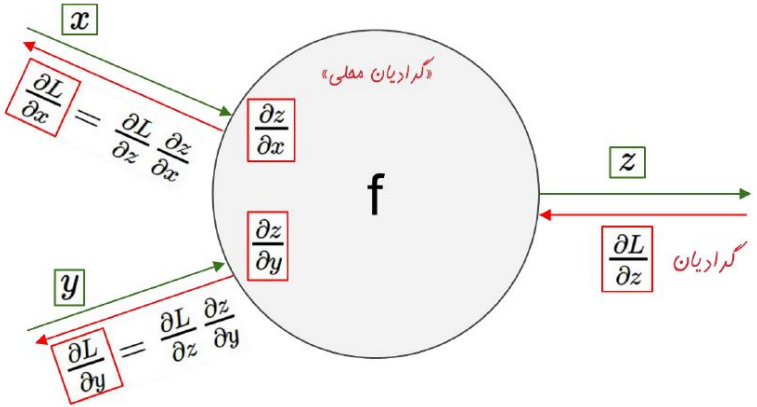
- وزن‌های مختل SGD آپدیت قبلی وزن‌ها در نظر می‌گیرد:



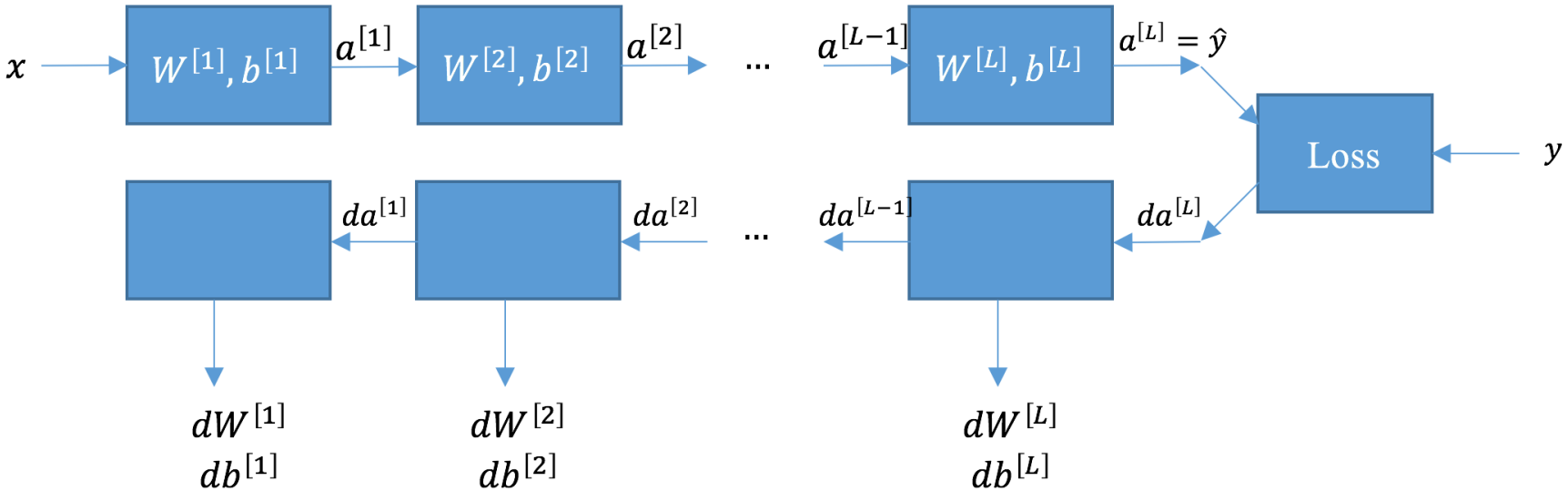
الگوریتم پس‌انتشار خطا

- با مقدار خطای نهایی شروع می‌شود و با پس‌روی از لایه‌های بالا به‌طرف لایه‌های پایین از **قاعده زنجیری** برای محاسبه سهم هر پارامتر در مقدار خطا استفاده می‌کند.
- به هیچ عنوان مجبور به پیاده‌سازی دستی الگوریتم پس‌انتشار نخواهید شد. تنها چیزی که باید به خوبی درک کنید چگونگی کارکرد بهینه‌سازی مبتنی بر گرادیان است.





الگوریتم پس‌انتشار خطا



الگوریتم پس‌انتشار خطا

```
# 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
```

تشکر

سوال؟

a.golzari@azaruniv.ac.ir

a.golzari@tabrizu.ac.ir

https://github.com/Amin-Golzari-Oskouei