شبکه‌های عصبی: آموزش

سید ناصر رضوی www.snrazavi.ir

۱۳۹۷

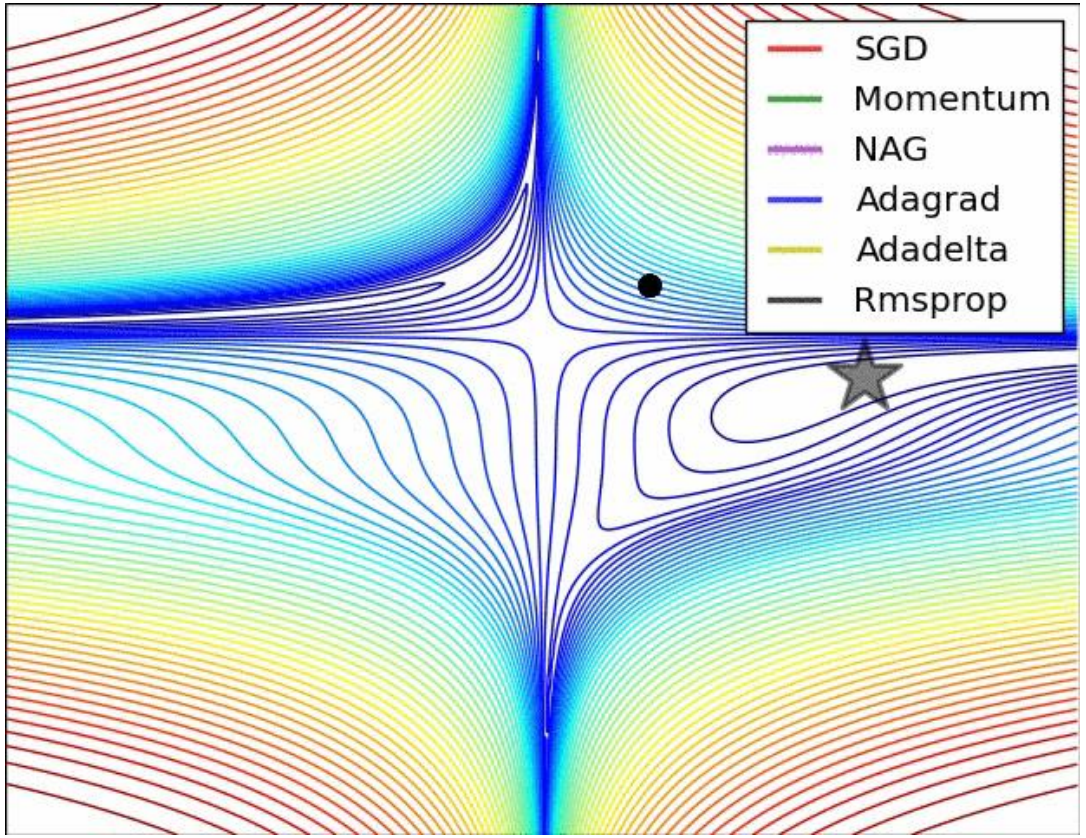
یادآوری: بهینه‌سازی

- گرادیان کاهشی. [با دسته‌های کوچک]

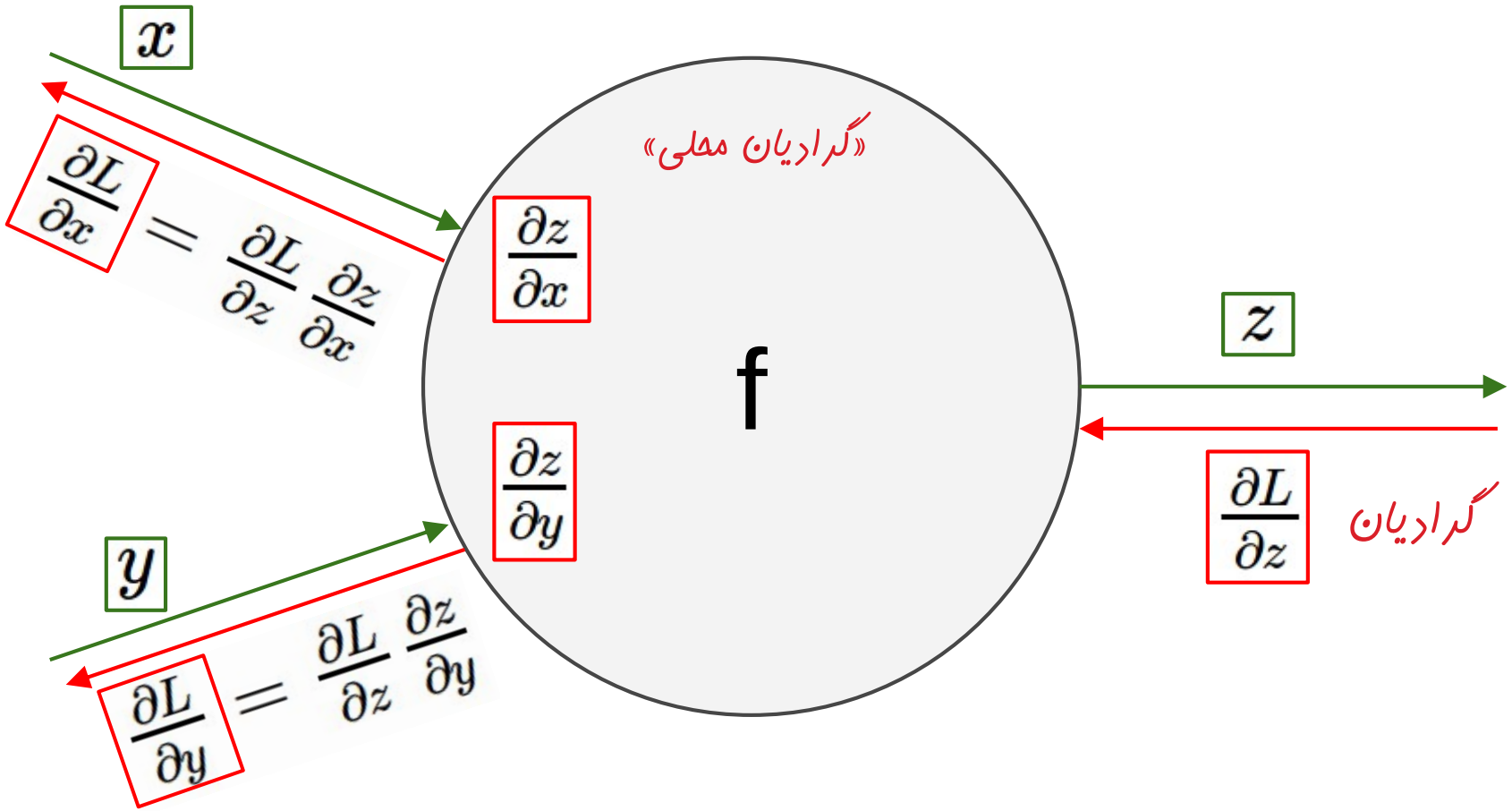
حلقه:

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.

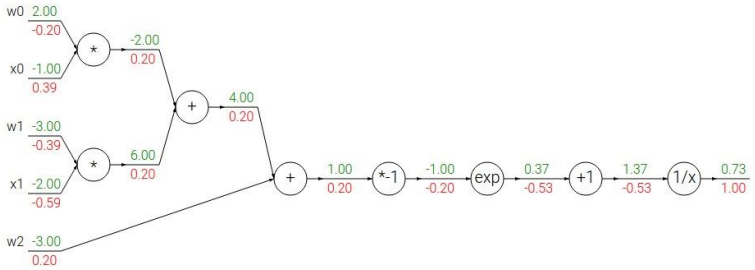
یادآوری: بهینه‌سازی



یادآوری: گراف محاسباتی

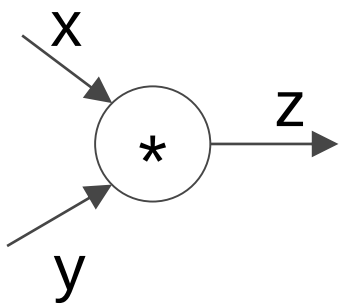


یادآوری: گراف محاسباتی




```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:

$$f = Wx$$

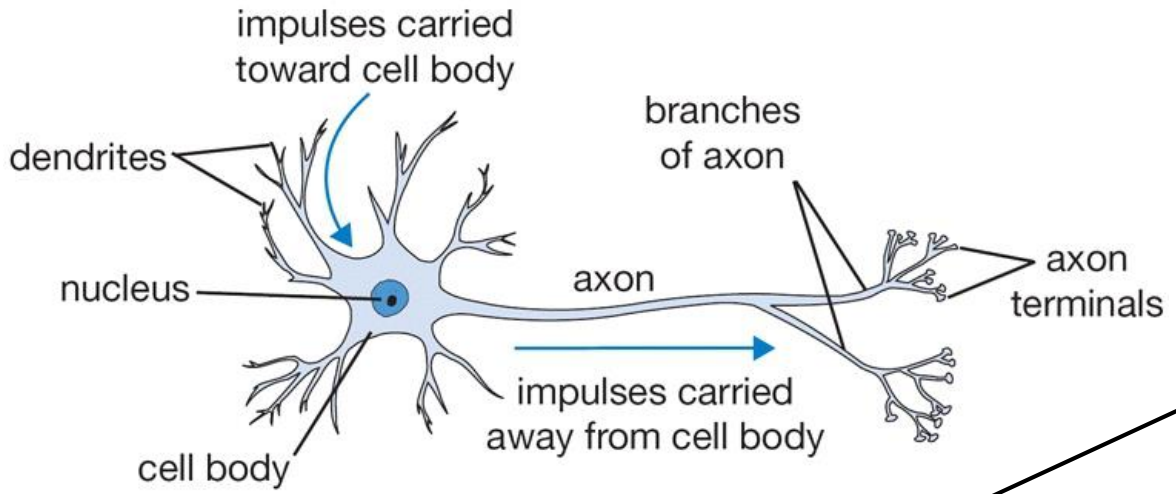
(**اکنون**) شبکه عصبی ۲ لایه:

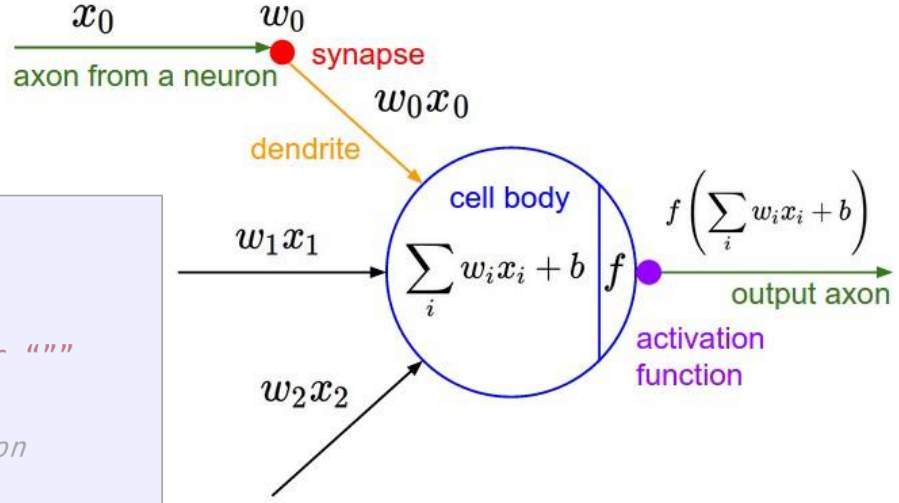
$$f = W_2 \max(0, W_1 x)$$

یا شبکه عصبی ۳ لایه:

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

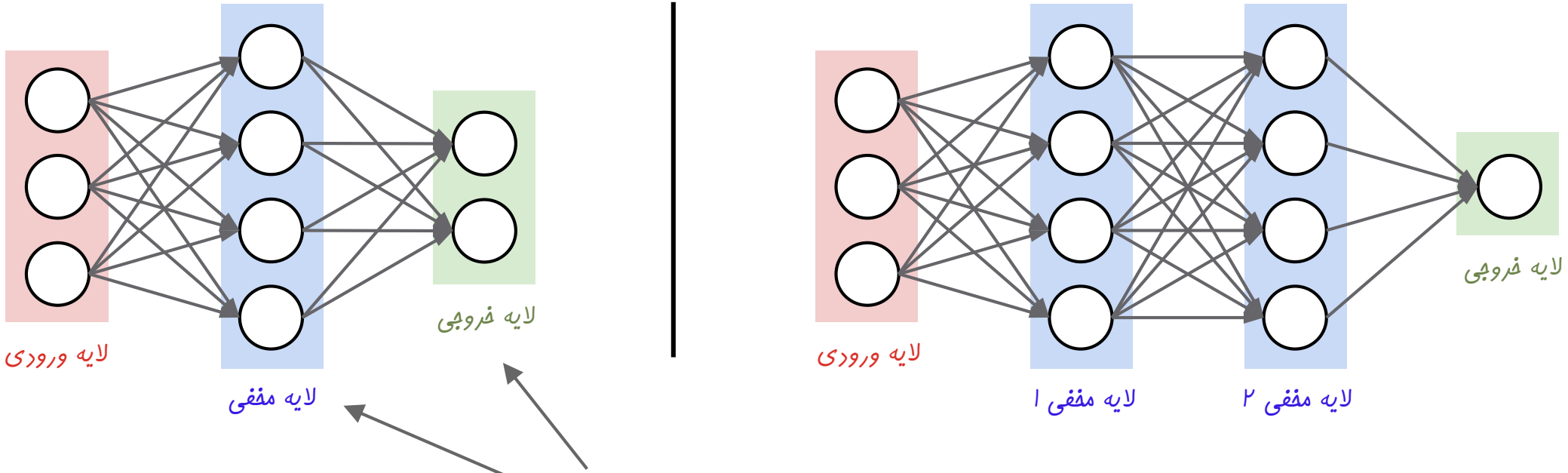
یادآوری: نورون‌ها و شبکه‌های عصبی






```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»

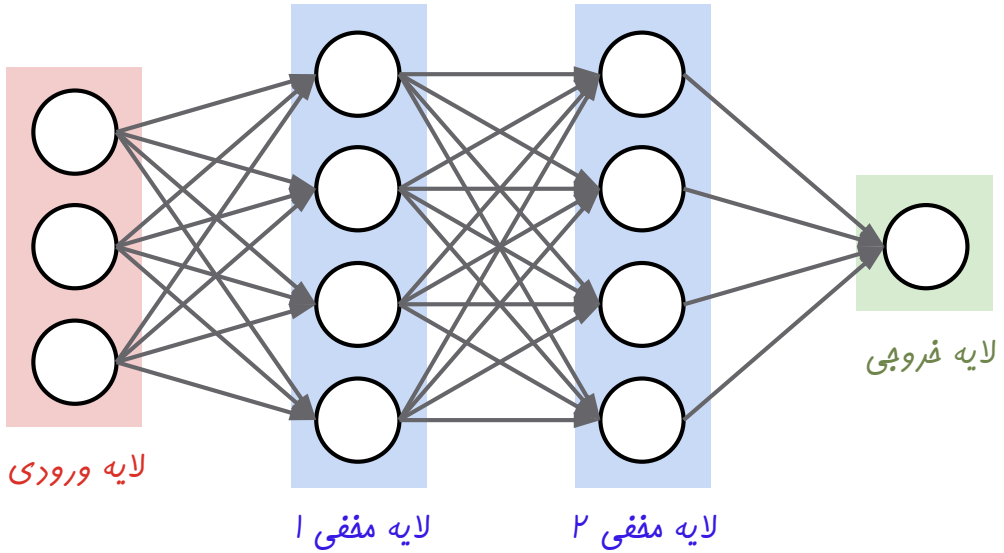
شبکه عصبی ۳ لایه

[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

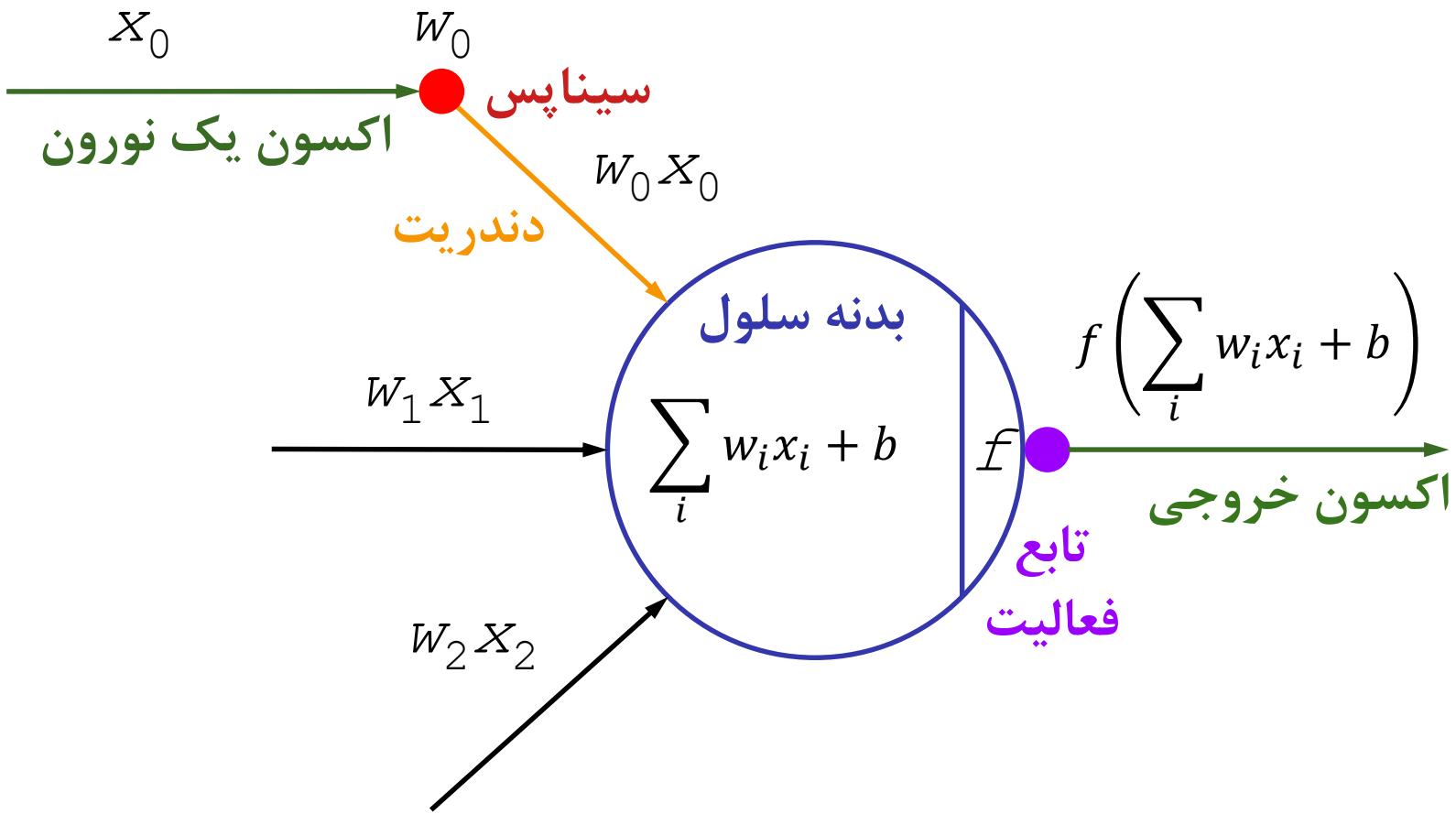
(۲) آموزش.

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

(۳) ارزیابی.

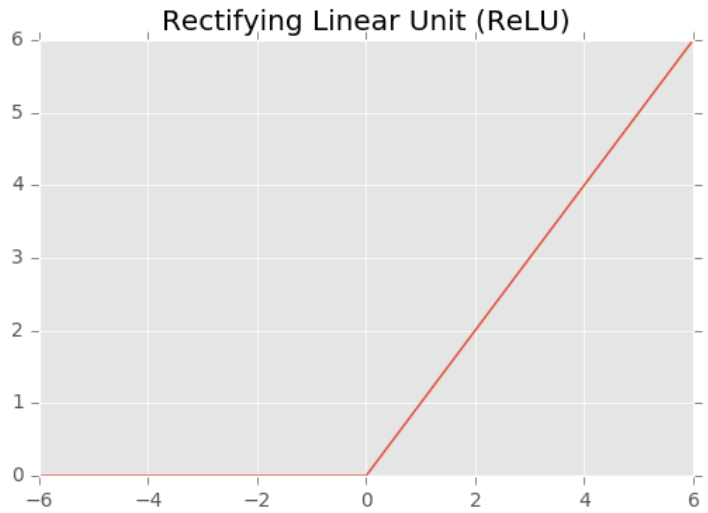
توابع فعالیت

توابع فعالیت



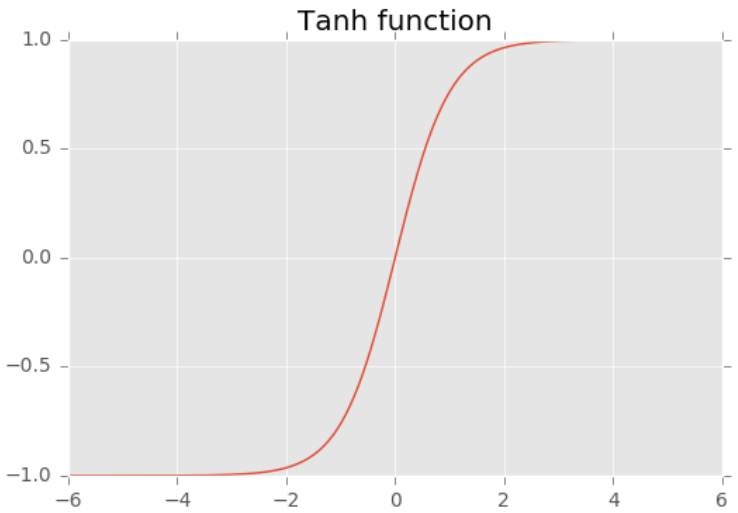
توابع فعالیت

ReLU



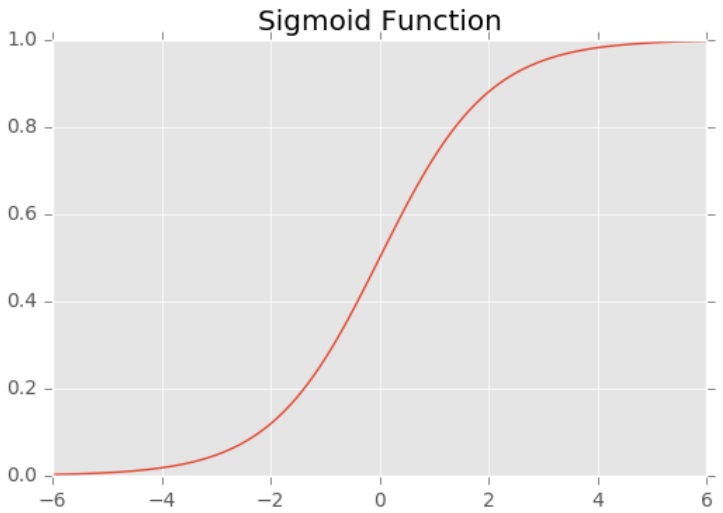
$$\max(0, x)$$

تانژانت هایپربولیک



$$\tanh(x)$$

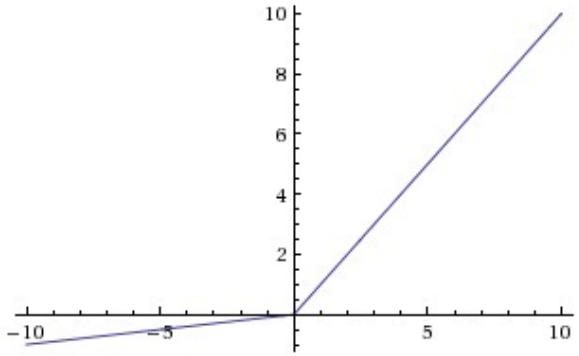
سیگموید



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

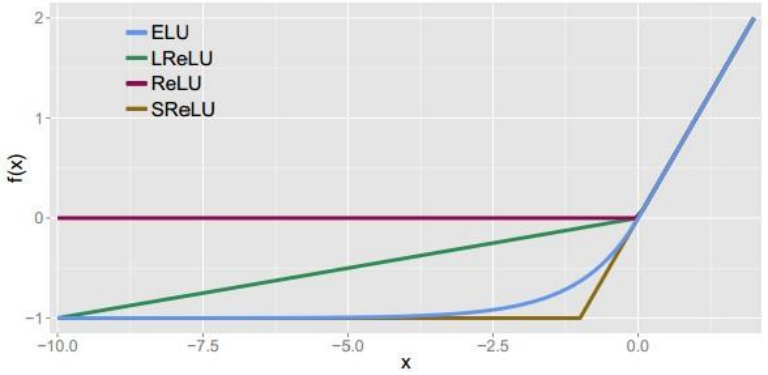
توابع فعالیت

Leaky ReLU



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

ELU

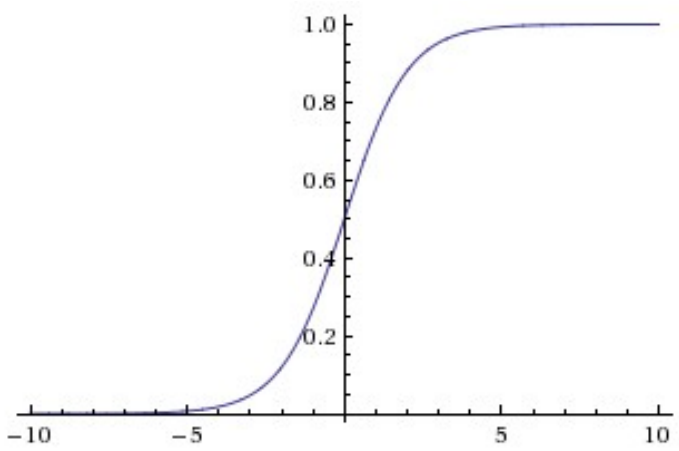


$$f(x) = \begin{cases} x, & x > 0 \\ \alpha(\exp(x) - 1), & x \geq 0 \end{cases}$$

توابع فعالیت

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

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



تابع سیگموید

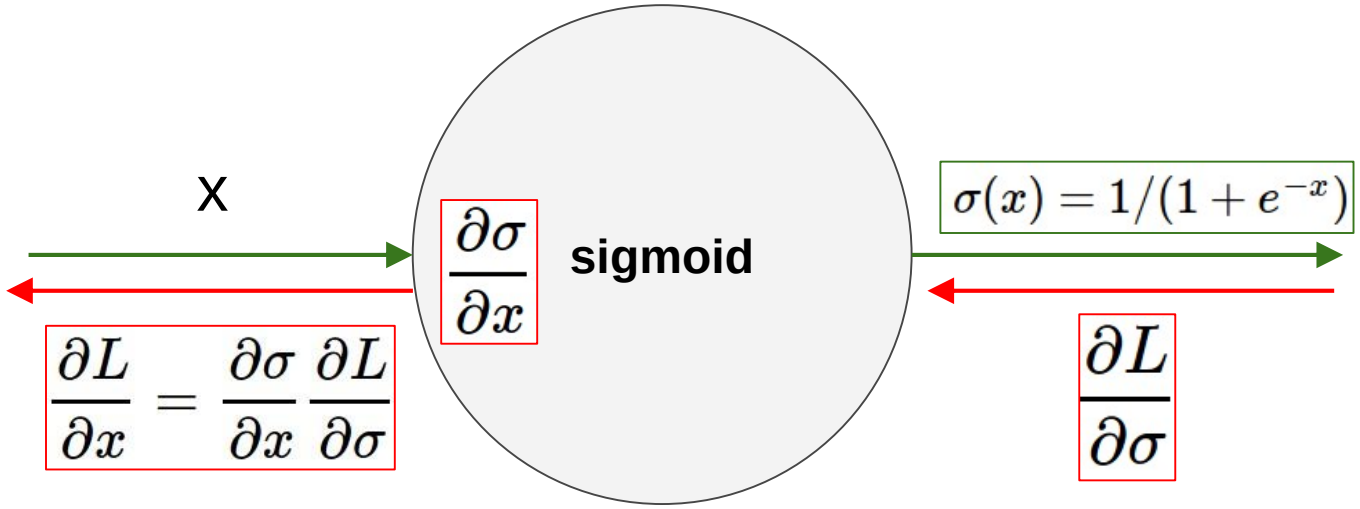
- مشکلات.

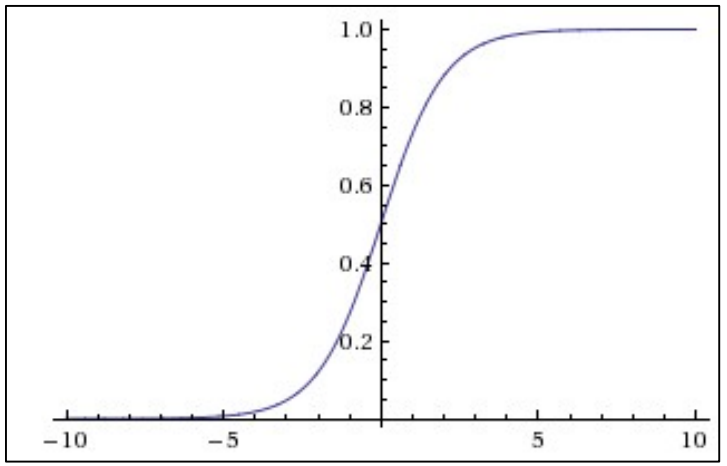
۱. نورون‌های اشباع شده، گرادیان را از بین می‌برند.

۲. خروجی‌های تابع سیگموید دارای میانگین صفر نیستند.

۳. توان رسانی، عمل نسبتاً پر هزینه‌ای است.

توابع فعالیت



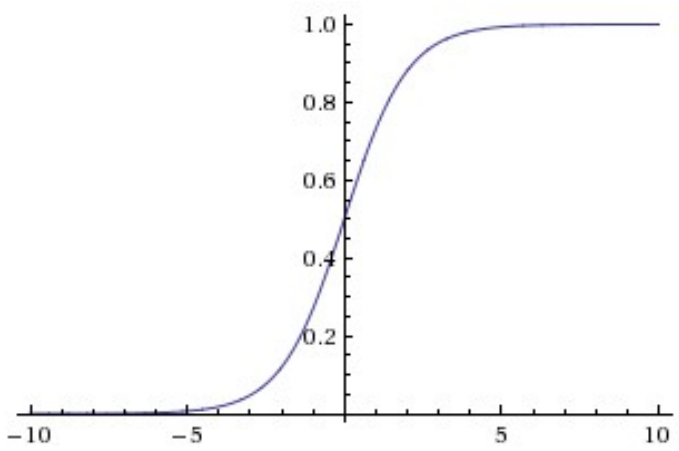


- چه اتفاقی می‌افتد اگر $x = -10$ ؟
- چه اتفاقی می‌افتد اگر $x = 0$ ؟
- چه اتفاقی می‌افتد اگر $x = +10$ ؟

توابع فعالیت

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

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



تابع سیگموید

- مشکلات.

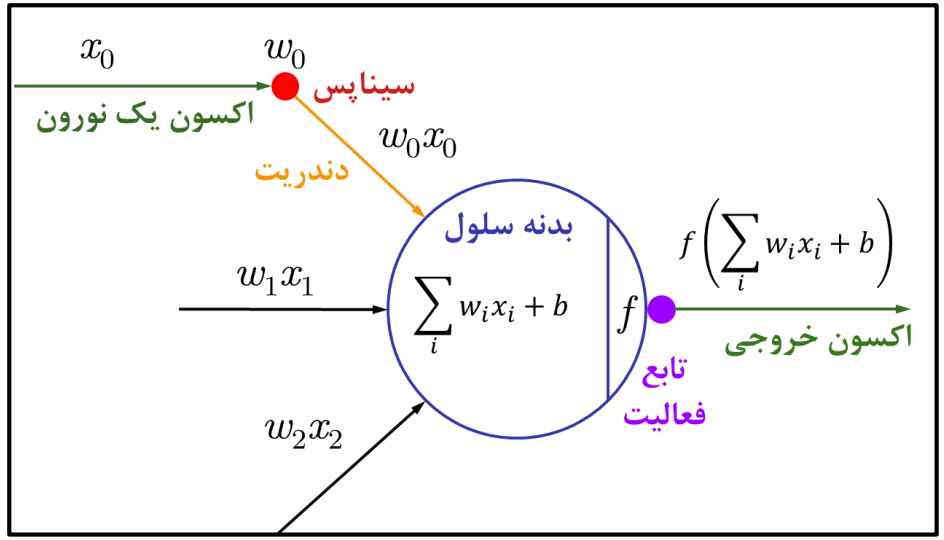
۱. نورون‌های اشباع شده، گرادیان را از بین می‌برند.

۲. خروجی‌های تابع سیگموید دارای میانگین صفر نیستند.

۳. توان رسانی، عمل نسبتاً پر هزینه‌ای است.

توابع فعالیت

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

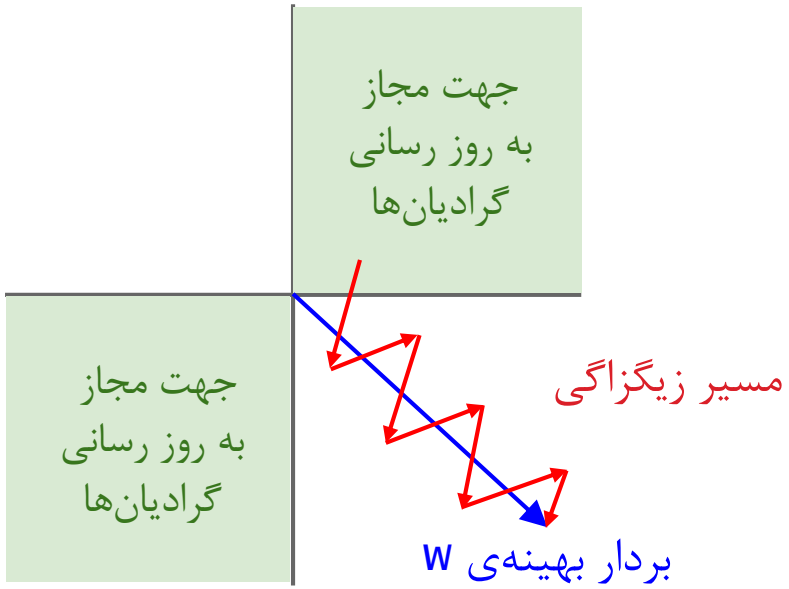


$$f\left(\sum_i w_i x_i + b\right)$$

- در مورد گرادیان‌ها نسبت به W چه می‌توان گفت؟

توابع فعالیت

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



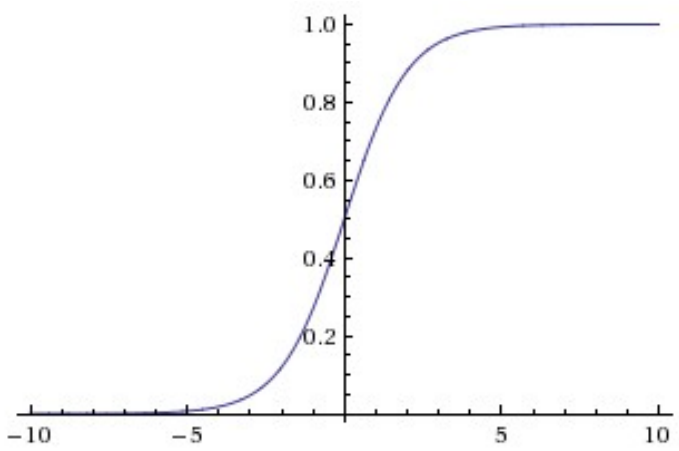
$$f\left(\sum_i w_i x_i + b\right)$$

- 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 بازه [۰، ۱]
  - دارای محبوبیت تاریخی در تاریخچه شبکه‌های عصبی

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



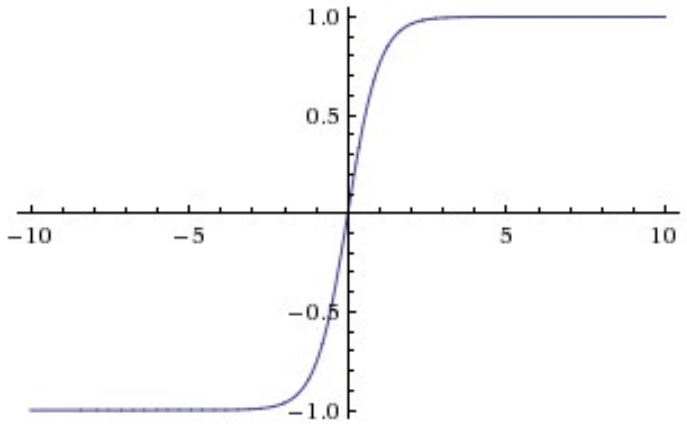
تابع سیگموید

- 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.

توابع فعالیت

- تابع فعالیت تانژانت هایپربولیک.

$$\tanh(x)$$



تانژانت هایپربولیک

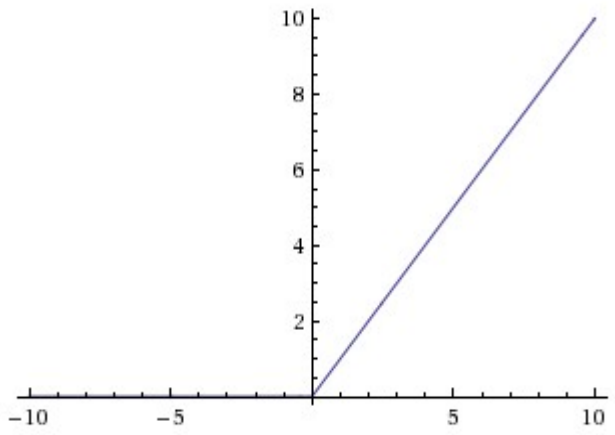
- نگاشت اعداد به بازه [۱، ۱-]
- دارای میانگین صفر
- هنوز نورون‌های اشباع شده، گرادیان را از بین می‌برند.

[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!

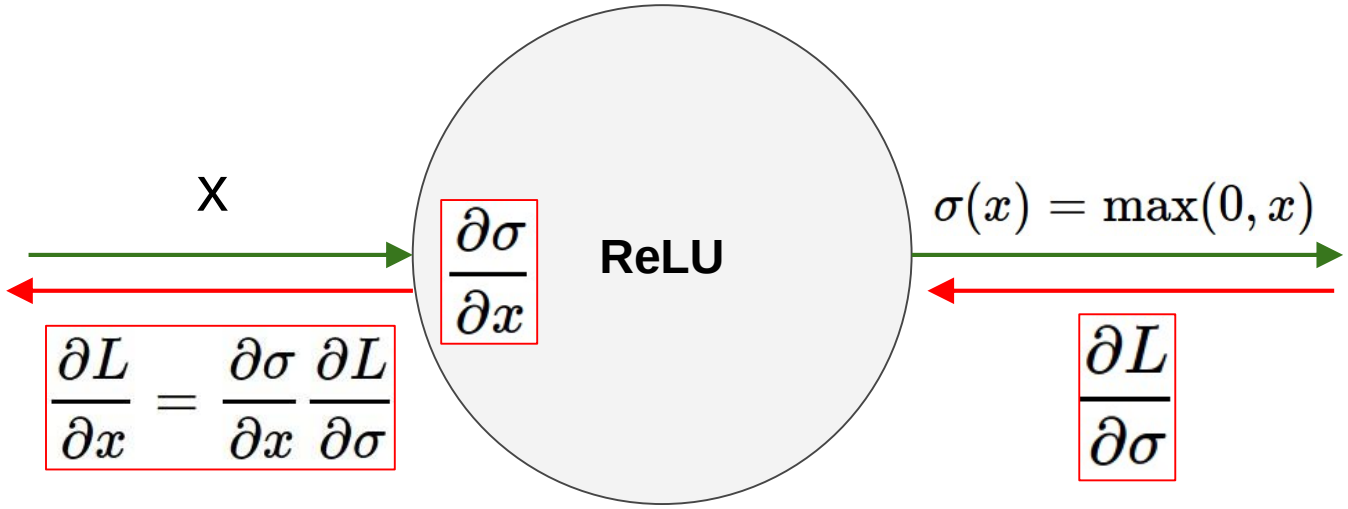
$$\max(0, x)$$

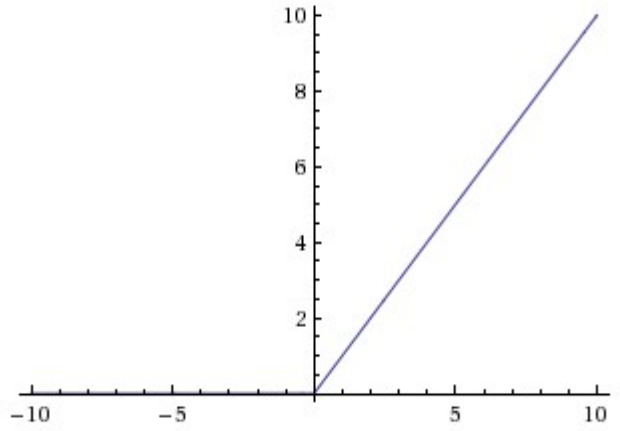


ReLU

[Krizhevsky et al., 2012]

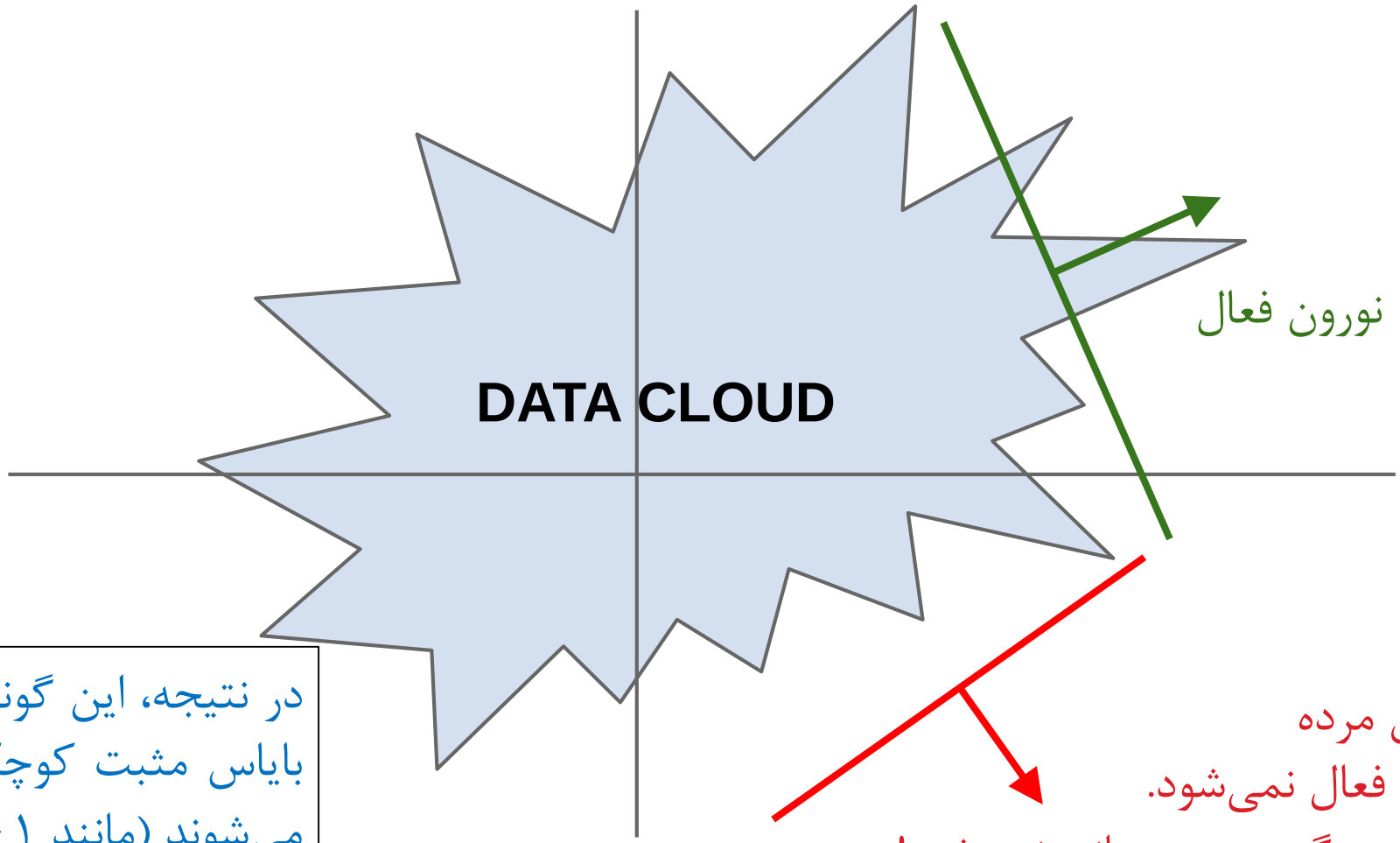
توابع فعالیت





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

توابع فعالیت

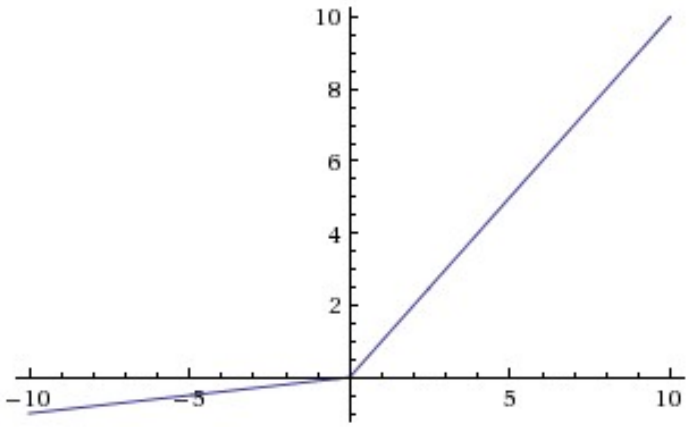


در نتیجه، این گونه نورون‌ها با مقادیر بایاس مثبت کوچک مقداردهی اولیه می‌شوند (مانند ۰/۰۱)

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!**

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

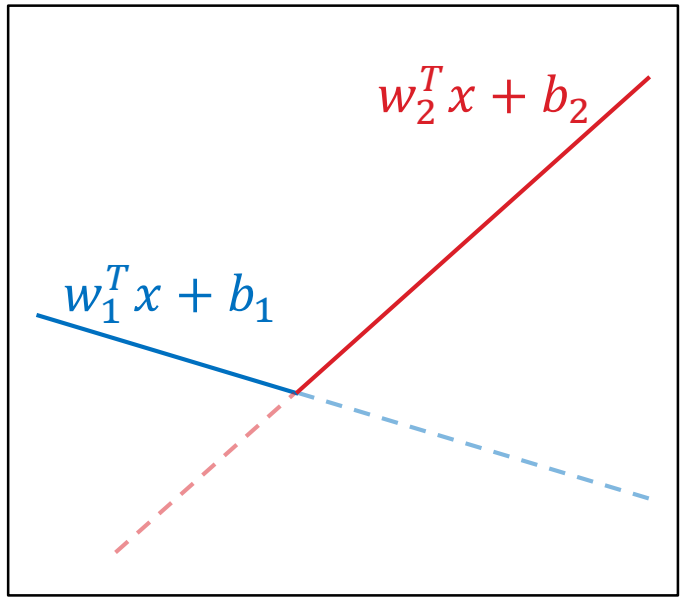


Leaky ReLU

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

نورون Maxout

- یک تابع تکه‌ای خطی. [مجموعه‌ای از چند تابع خطی]
- تعمیم دهنده 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]

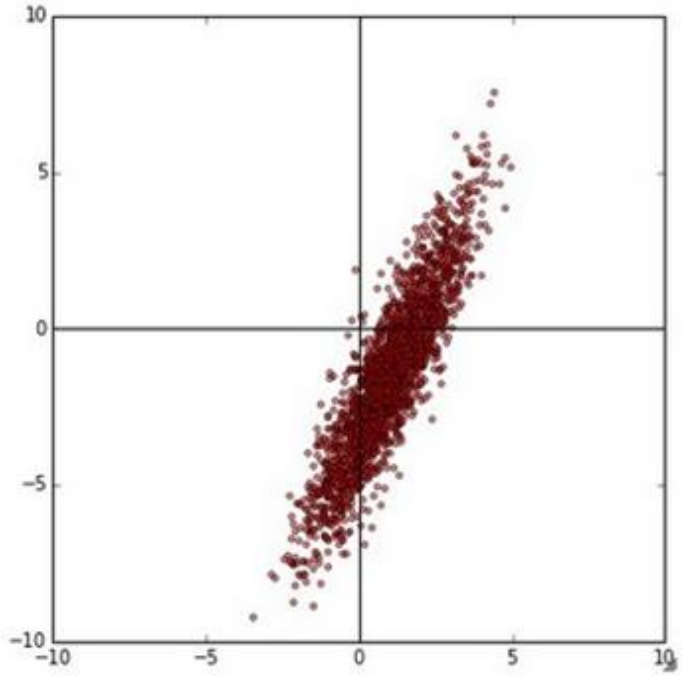
انتخاب تابع فعالیت

- 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.

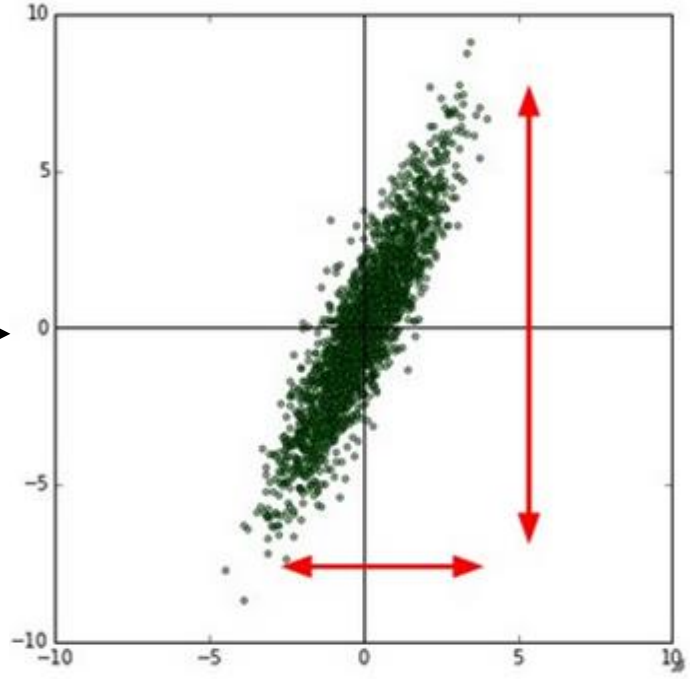
پیش‌پردازش داده‌ها

پیش‌پردازش داده‌ها

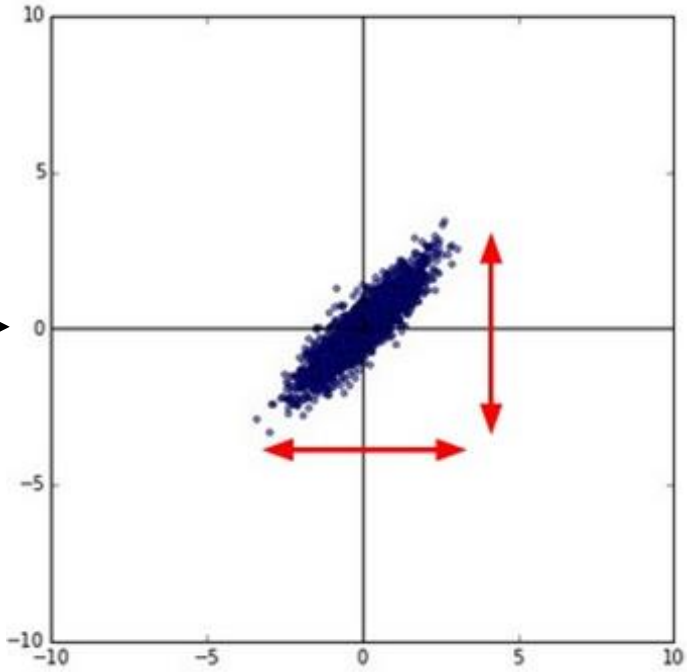
داده‌های اصلی



داده‌ها با میانگین صفر



داده‌های نرمال شده

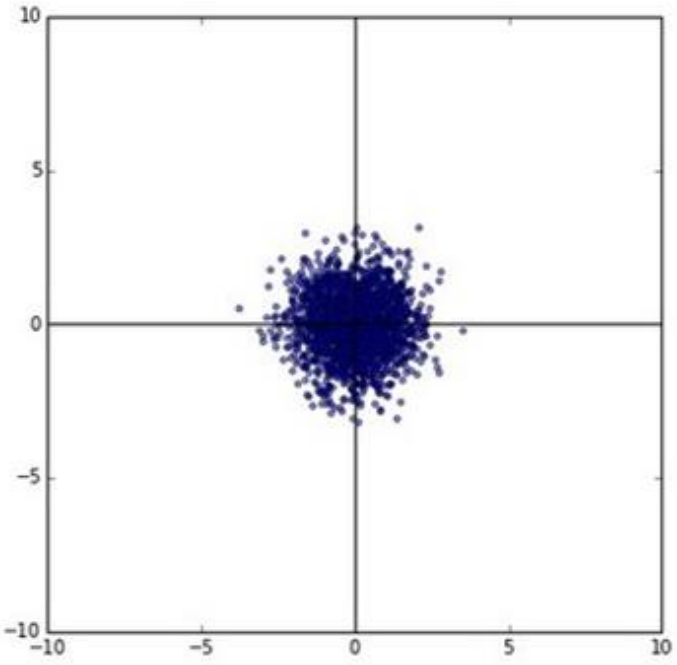


```
X -= np.mean(X, axis=0)
```

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

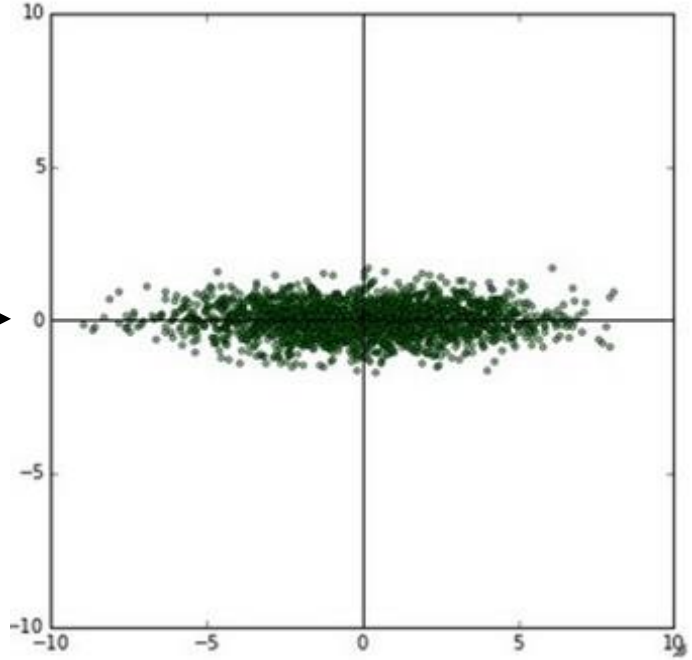
پیش‌پردازش داده‌ها

داده‌های سفید شده



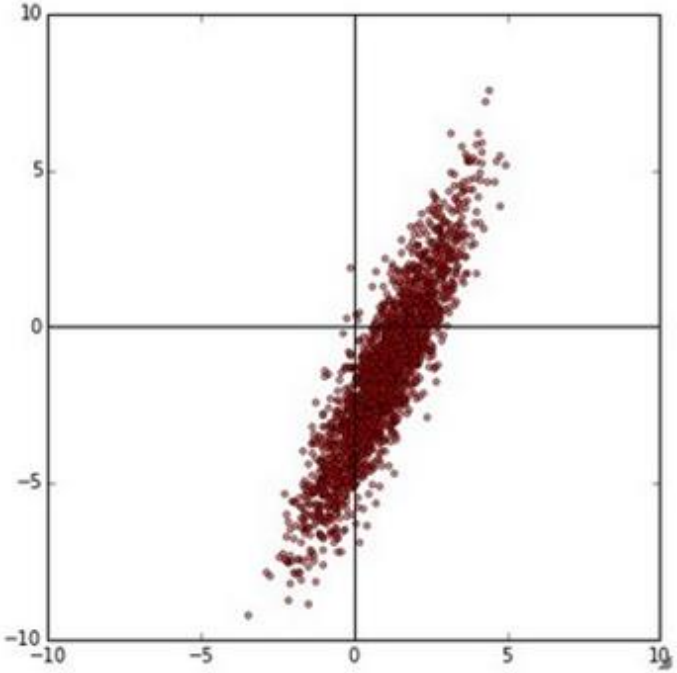
تبدیل ماتریس کوواریانس به یک ماتریس همانی

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.

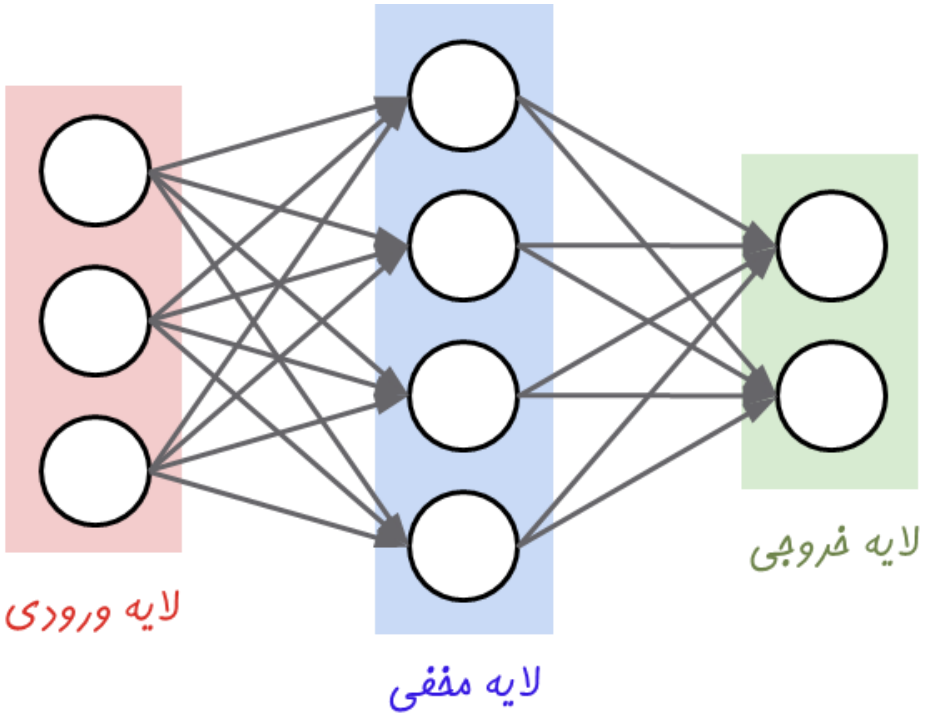
- به عنوان نمونه، تصاویر مجموعه داده CIFAR-10 را با ابعاد [۳، ۳۲، ۳۲] در نظر بگیرید:
  - **روش اول:** کم کردن تصویر میانگین از تمام تصاویر [مانند AlexNet]
    - ابعاد بردار میانگین: [۳، ۳۲، ۳۲]
  - **روش دوم:** کم کردن میانگین به ازای هر کانال رنگ [مانند VGGNet]
    - ابعاد بردار میانگین: ۳ عدد

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

- ایده 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

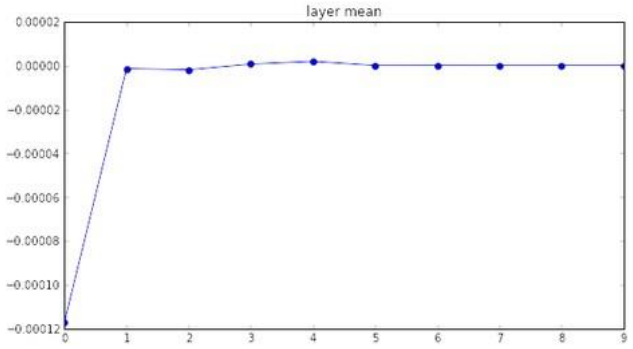
```
# assume some unit gaussian 10-D input data
D = np.random.randn(1000, 500)
hidden_layer_sizes = [500]*10
nonlinearities = ['tanh']*len(hidden_layer_sizes)
```

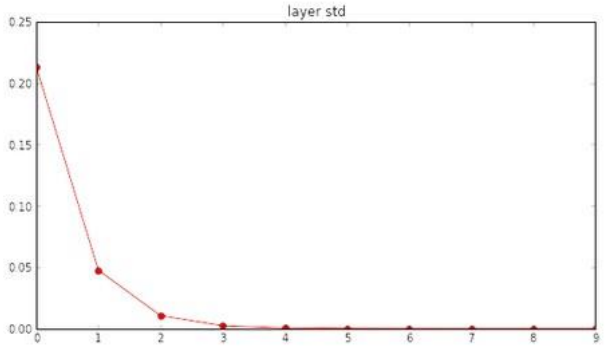
```
act = {'relu':lambda x:np.maximum(0,x), 'tanh':lambda x:np.tanh(x)}
Hs = {}
for i in xrange(len(hidden_layer_sizes)):
    X = D if i == 0 else Hs[i-1] # input at this layer
    fan_in = X.shape[1]
    fan_out = hidden_layer_sizes[i]
    W = np.random.randn(fan_in, fan_out) * 0.01 # layer initialization

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

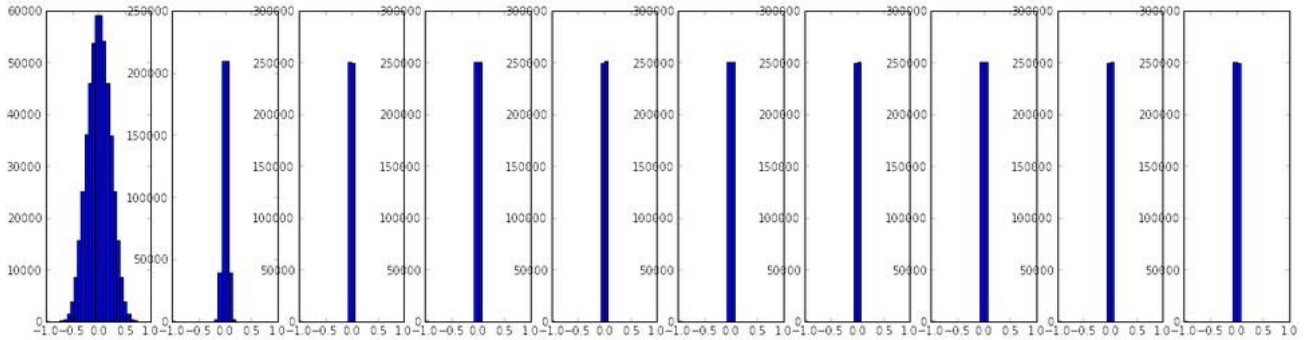
```
# look at distributions at each layer
print 'input layer had mean %f and std %f' % (np.mean(D), np.std(D))
layer_means = [np.mean(H) for i,H in Hs.iteritems()]
layer_stds = [np.std(H) for i,H in Hs.iteritems()]
for i,H in Hs.iteritems():
    print 'hidden layer %d had mean %f and std %f' % (i+1,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

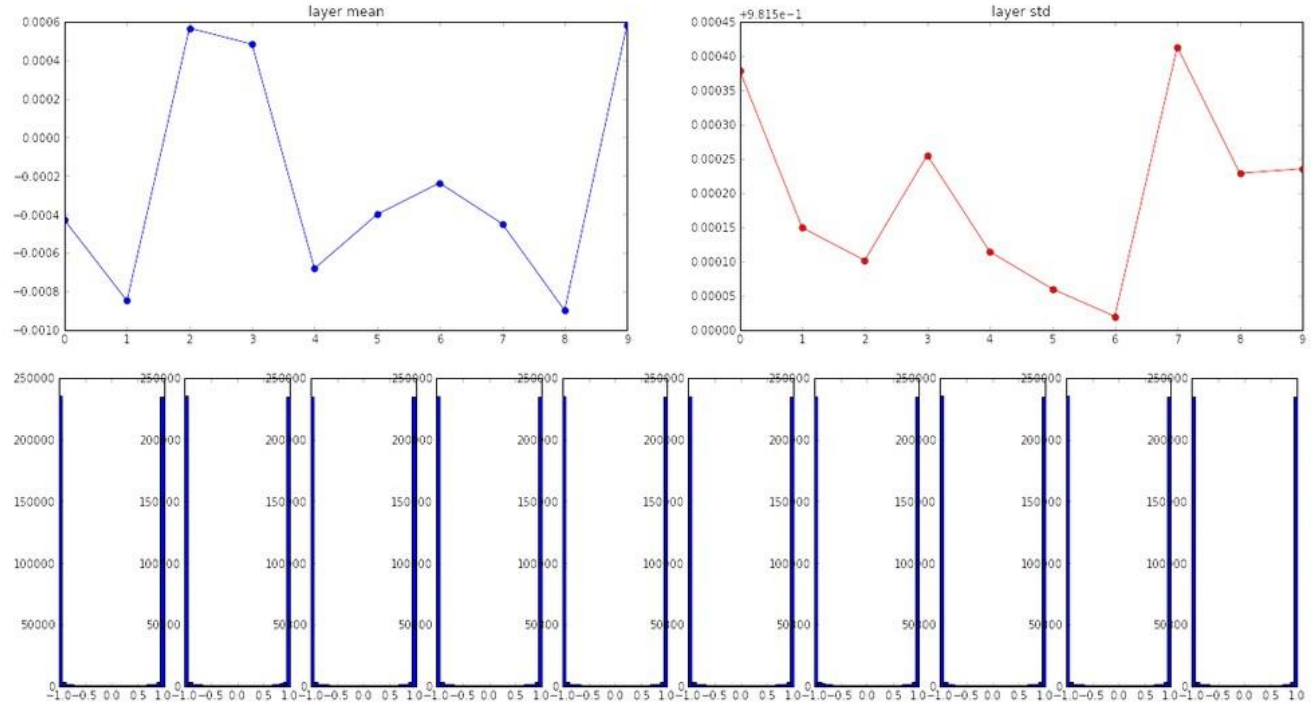
- مقدار فعالیت نورون‌ها صفر می‌شود!

- 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.

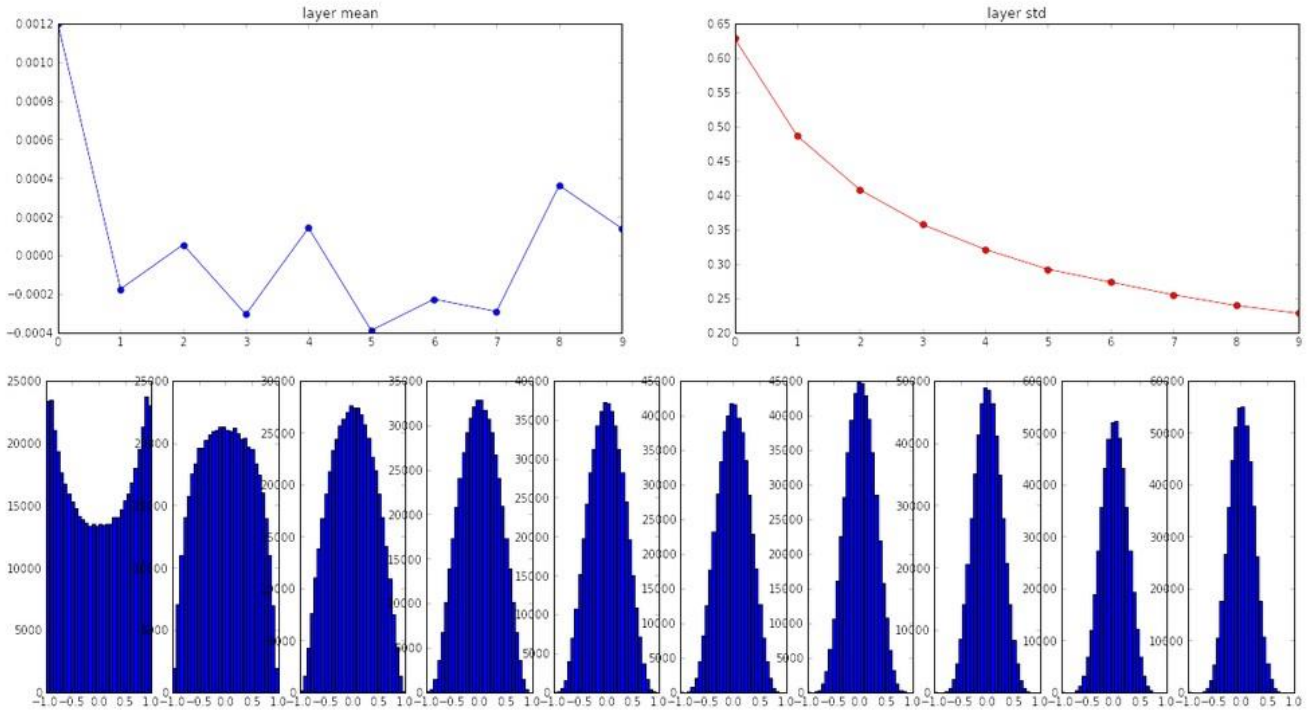
وزن‌دهی با مقادیر بزرگ ←

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

مقداردهی اولیه به وزن‌ها

- وزن‌دهی زاویر. [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
```

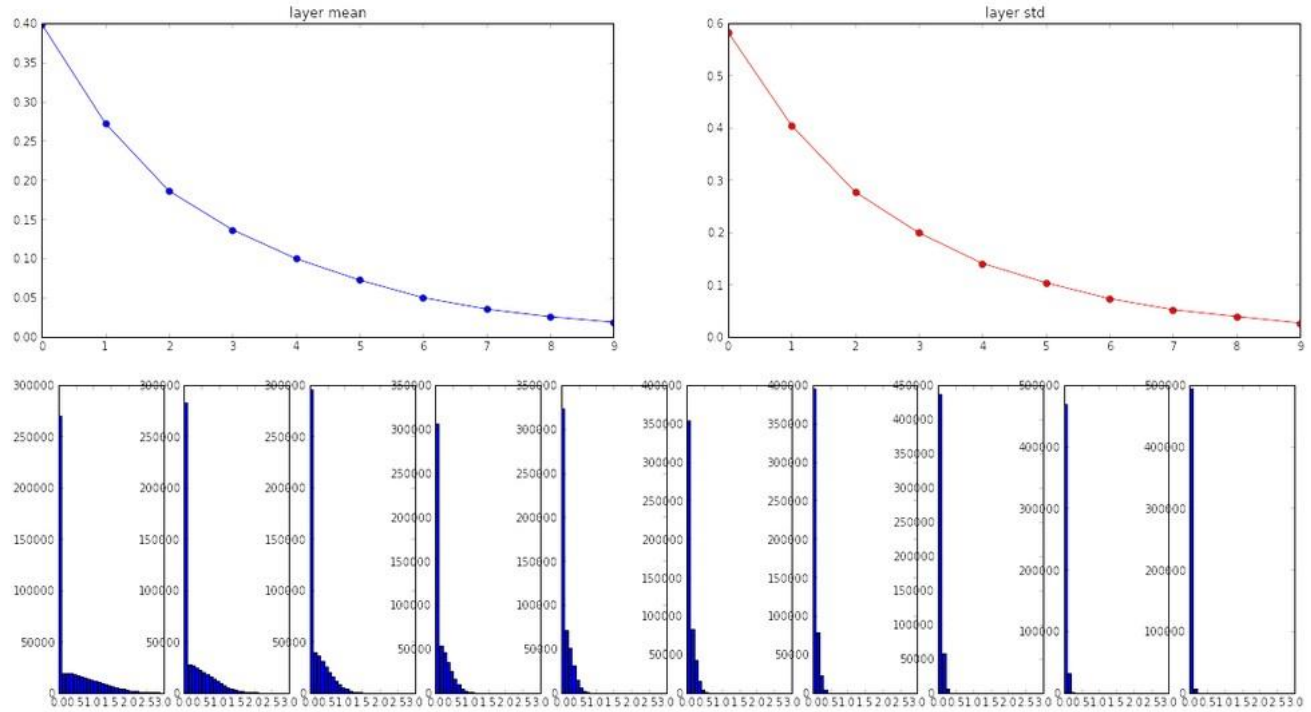



```
W = np.random.randn(fan_in, fan_out) / np.sqrt(fan_in)
```

مقداردهی اولیه به وزن‌ها

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

```
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
```

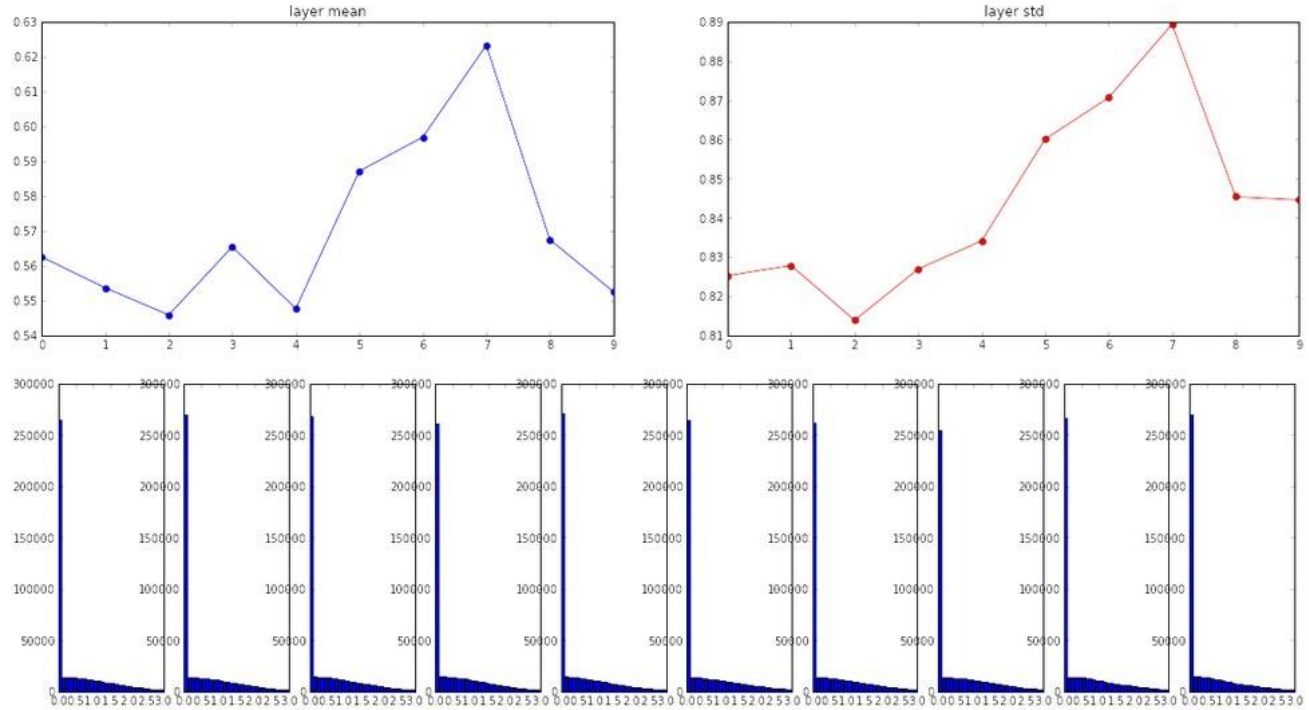


اما در صورت استفاده از تابع فعالیت ReLU، این روش دیگر به درستی عمل نخواهد کرد!

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

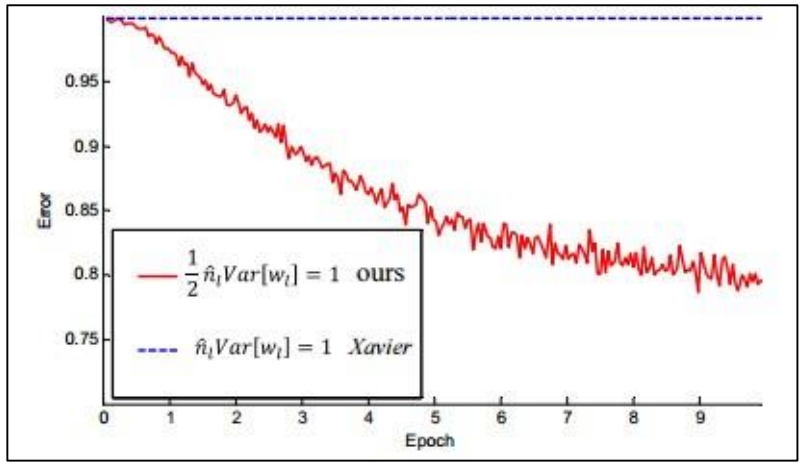
مقداردهی اولیه به وزن‌ها


```
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)
```

نرمال‌سازی دسته‌ای

نرمال‌سازی دسته‌ای

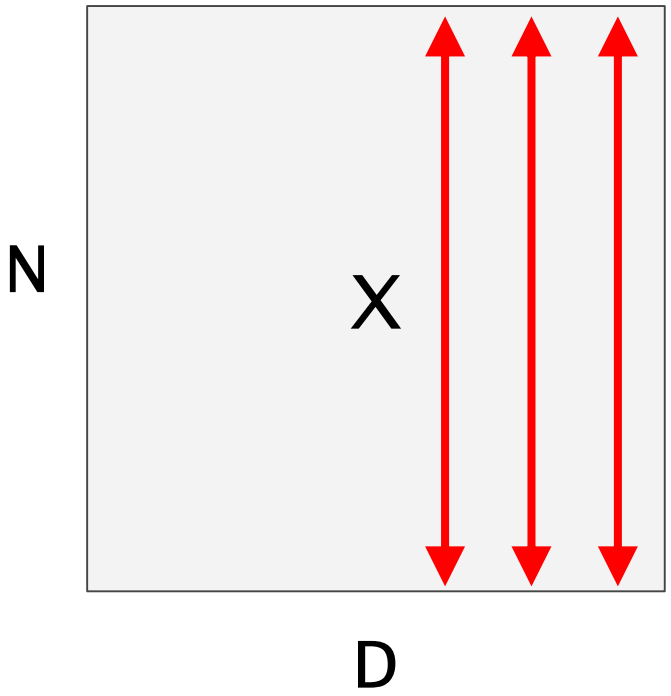
- نرمال‌سازی دسته‌ای. [Loffe et al., 2015]
  - یک دسته از مقادیر فعالیت را در یک لایه در نظر بگیرید:
  - تبدیل هر بُعد به یک توزیع گاوسی نرمال [میانگین ۰ و انحراف معیار ۱]

$$\hat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}\left[x^{(k)}\right]}{\sqrt{\mathrm{Var}\left[x^{(k)}\right]}}$$

این تابع یک تابع مشتق‌پذیر است ...

نرمال‌سازی دسته‌ای

- نرمال‌سازی دسته‌ای.



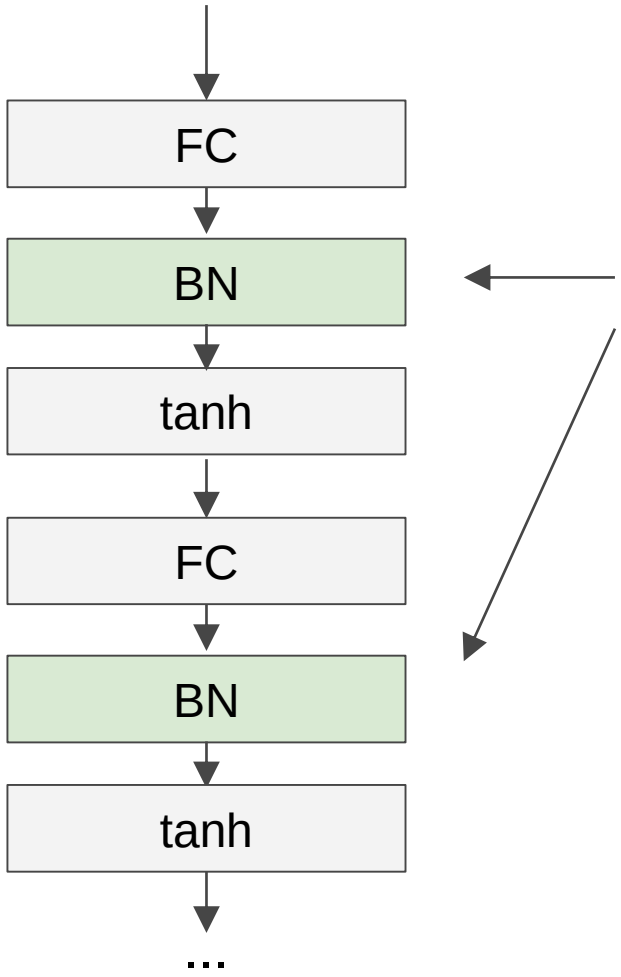
۱. میانگین و واریانس را به طور مستقل برای هر بُعد محاسبه کنید.

۲. نرمال‌سازی کنید.

$$\hat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}\left[x^{(k)}\right]}{\sqrt{\mathrm{Var}\left[x^{(k)}\right]}}$$

نرمال‌سازی دسته‌ای

- نرمال‌سازی دسته‌ای.



نرمال‌سازی دسته‌ای به طور معمول پس از لایه‌های کاملاً متصل و پیش از تابع فعالیت غیرخطی انجام می‌شود.

مشکل. آیا واقعاً نیاز داریم ورودی‌های تابع فعالیت tanh دارای توزیع گوسی نرمال باشند؟

نرمال‌سازی دسته‌ای

- نرمال‌سازی دسته‌ای.

نرمال‌سازی کنید:

$$\hat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}\left[x^{(k)}\right]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

سپس اجازه دهید شبکه در صورت نیاز میانگین و واریانس را تغییر دهد:

$$y^{(k)} = \gamma^{(k)}\hat{x}^{(k)} + \beta^{(k)}$$

توجه. شبکه می‌تواند مقادیر زیر را یاد بگیرد:

$$\gamma^{(k)} = \sqrt{\mathrm{Var}[x^{(k)}]}$$

$$\beta^{(k)} = \mathrm{E}\left[x^{(k)}\right]$$

که باعث می‌شود نرمال‌سازی انجام شده خنثی گردد.

نرمال‌سازی دسته‌ای: مزایا

Inputs: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\dots m}\}$;
Parameters to be learned: γ, β

Outputs: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\hat{x}_i \leftarrow (x_i - \mu_{\mathcal{B}})/\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\hat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

- نرمال‌سازی دسته‌ای.
  - جریان گرادیان را در شبکه بهبود می‌بخشد.
  - امکان استفاده از مقادیر بزرگ‌تر را برای نرخ یادگیری فراهم می‌کند.
  - وابستگی به روش مقداردهی اولیه وزن‌ها را کاهش می‌دهد.
  - به عنوان نوعی تنظیم کننده عمل می‌کند و در نتیجه نیاز به استفاده از دراپ‌آوت را کاهش می‌دهد.

نرمال‌سازی دسته‌ای: در زمان پیش‌بینی

Inputs: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\dots m}\}$;
Parameters to be learned: γ, β

Outputs: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\hat{x}_i \leftarrow (x_i - \mu_{\mathcal{B}})/\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon} \qquad \text{// normalize}$$

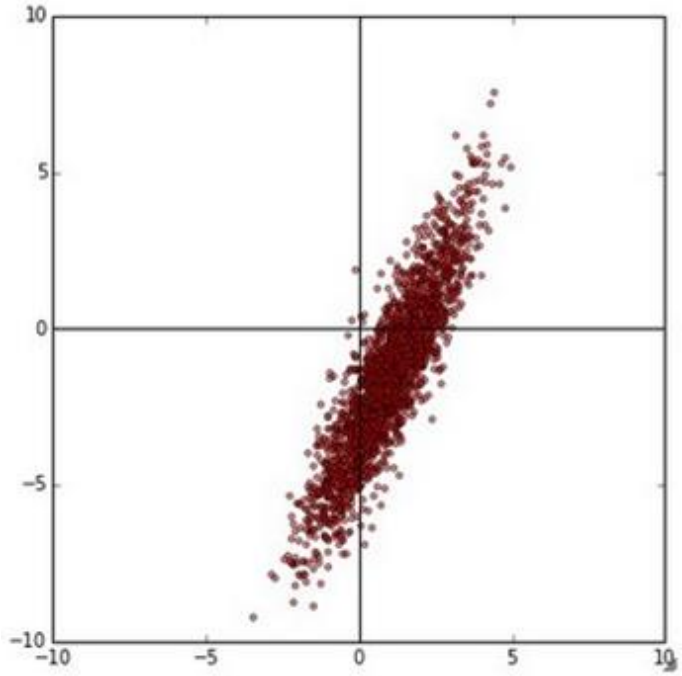
$$y_i \leftarrow \gamma\hat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

- نرمال‌سازی دسته‌ای در زمان آزمایش به گونه‌ی متفاوتی عمل می‌کند.
  - میانگین و انحراف معیار بر مبنای دسته داده‌ها محاسبه نمی‌شوند.
  - به جای آن، یک مقدار میانگین ثابت که در طی فرایند آموزش به دست آمده است، استفاده می‌شود.
  - یعنی، تخمین میانگین و انحراف معیار در طی فرایند آموزش و سپس استفاده از این مقادیر تخمینی در مرحله آزمایش!

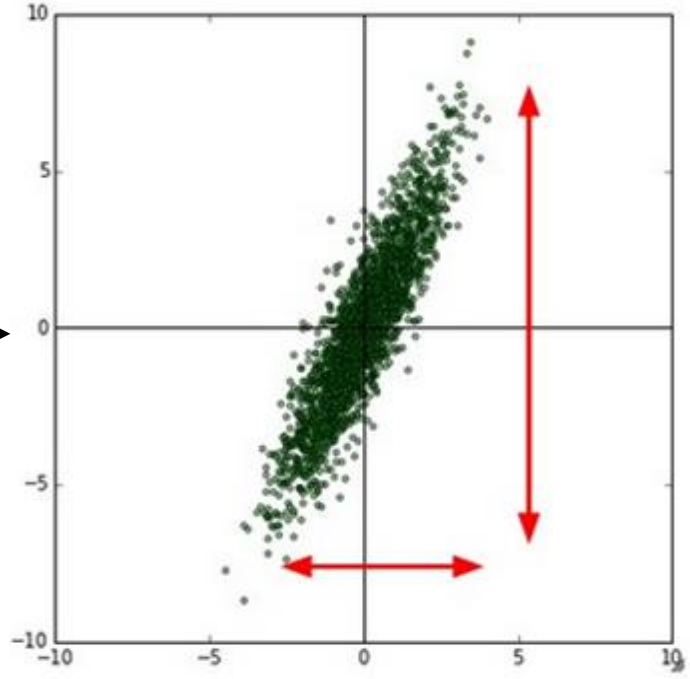
مراحل توسعه فرایند آموزش

گام اول: پیش‌پردازش داده‌ها

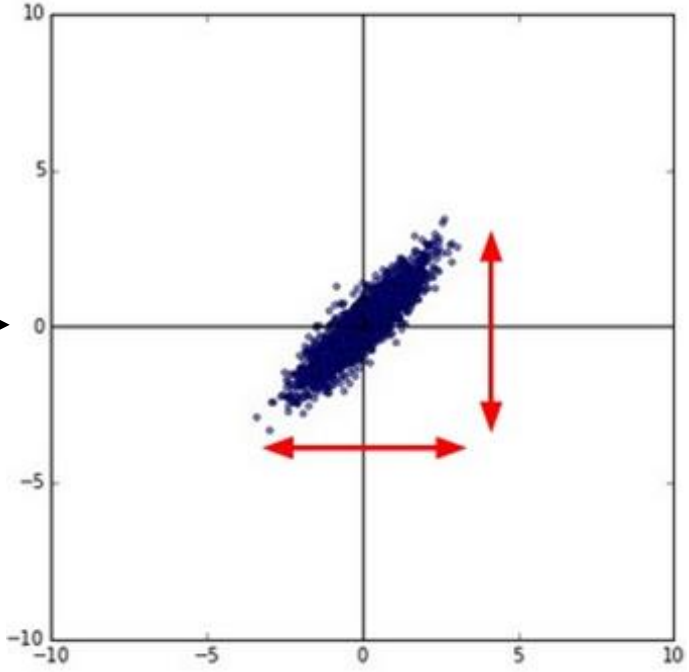
داده‌های اصلی



داده‌ها با میانگین صفر



داده‌های نرمال شده

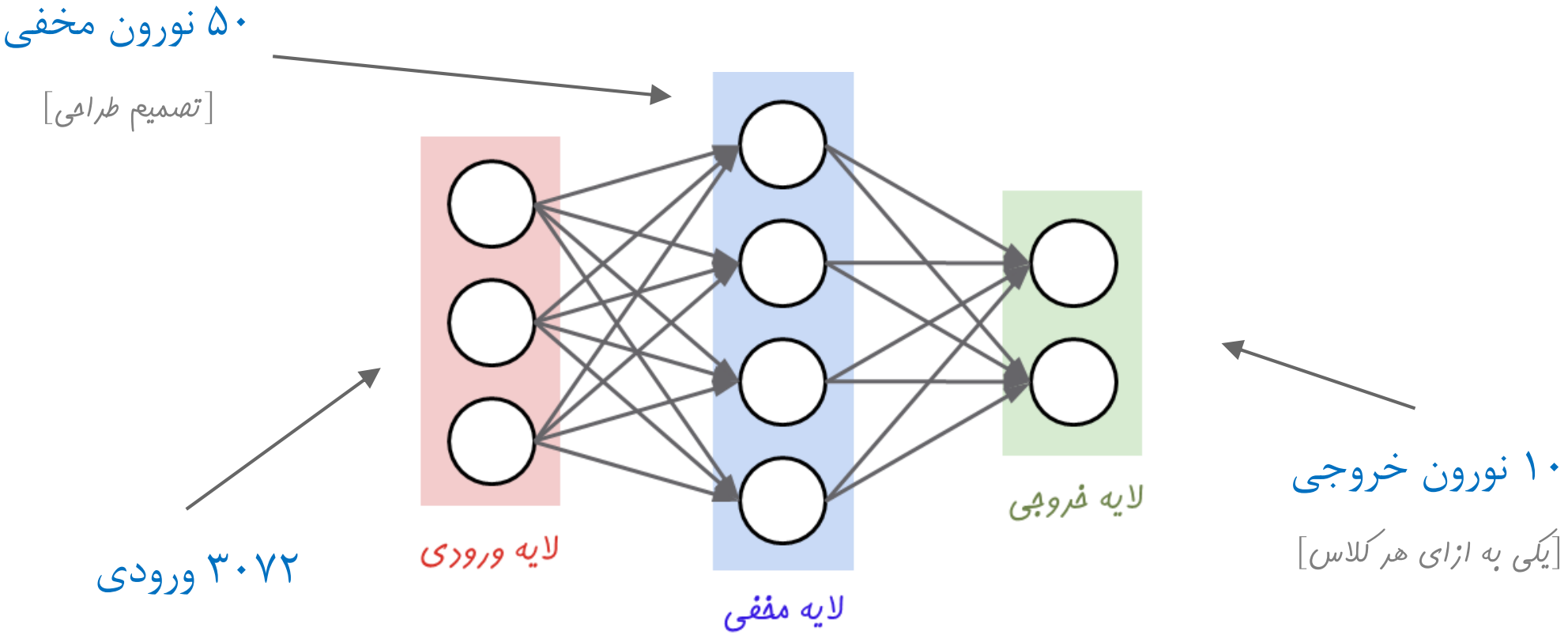


```
X -= np.mean(X, axis=0)
```

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

گام دوم: انتخاب معماری

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



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 مجموعه 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 ایده 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

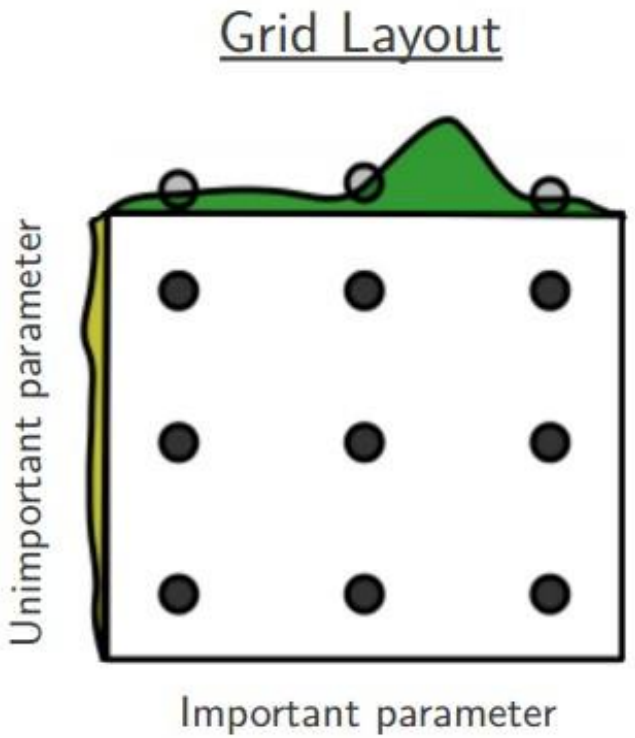
```
max_count = 100
for count in xrange(max_count):
      reg = 10**uniform(-4, 0)
      lr = 10**uniform(-3, -4)
```

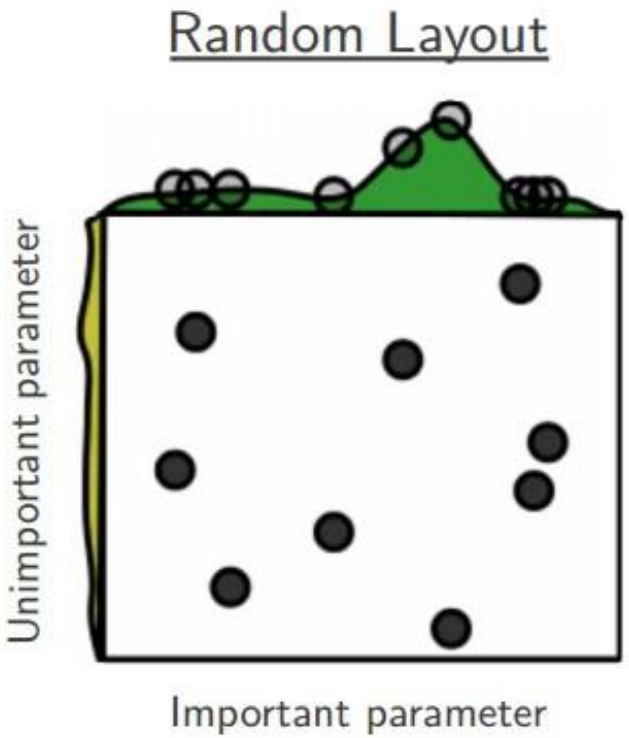
```
val_acc: 0.527000, lr: 5.340517e-04, reg: 4.097824e-01, (0 / 100)
val_acc: 0.492000, lr: 2.279484e-04, reg: 9.991345e-04, (1 / 100)
val_acc: 0.512000, lr: 8.680827e-04, reg: 1.349727e-02, (2 / 100)
val_acc: 0.461000, lr: 1.028377e-04, reg: 1.220193e-02, (3 / 100)
val_acc: 0.460000, lr: 1.113730e-04, 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 نتیجه به دست آمده نگران کننده است. چرا؟

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

بهینه‌سازی ابرپارامترها: جستجوی تصادفی یا منظم؟



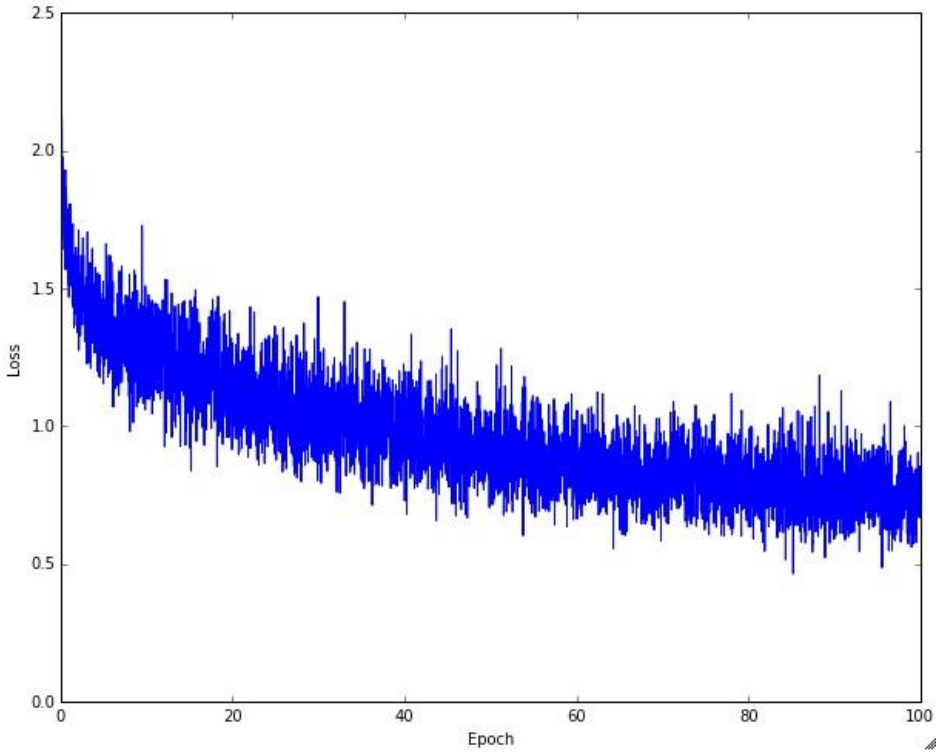


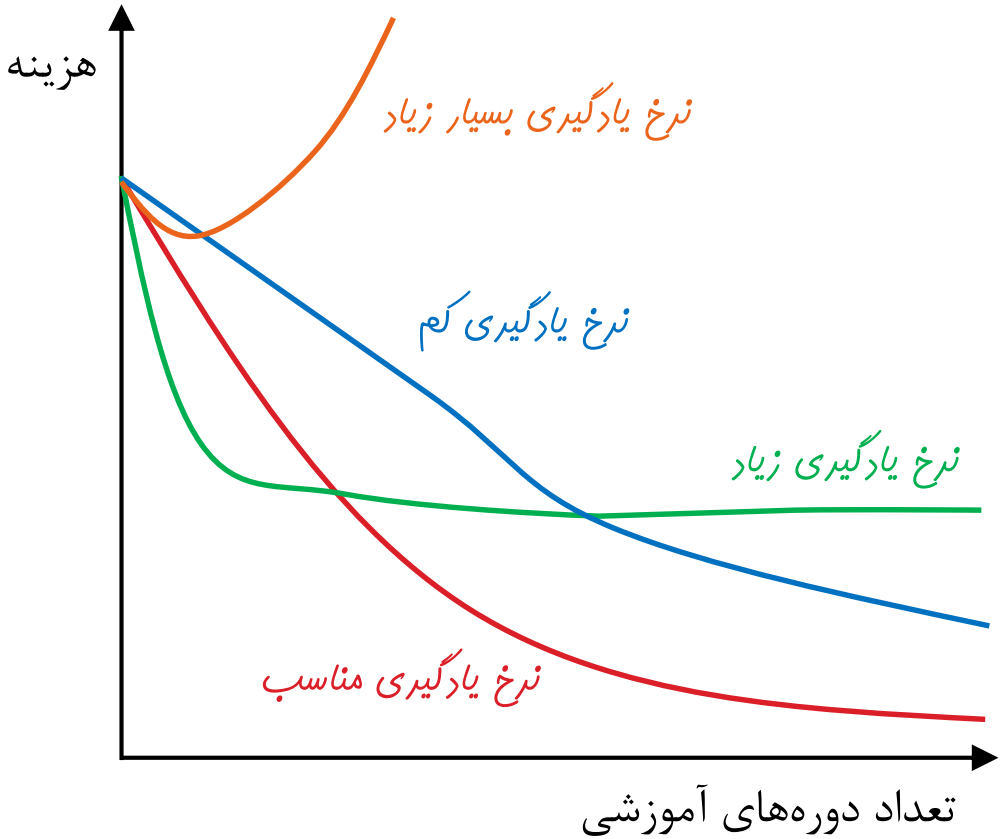
[Bergstra and Bengio, 2012]

بهینه‌سازی ابرپارامترها

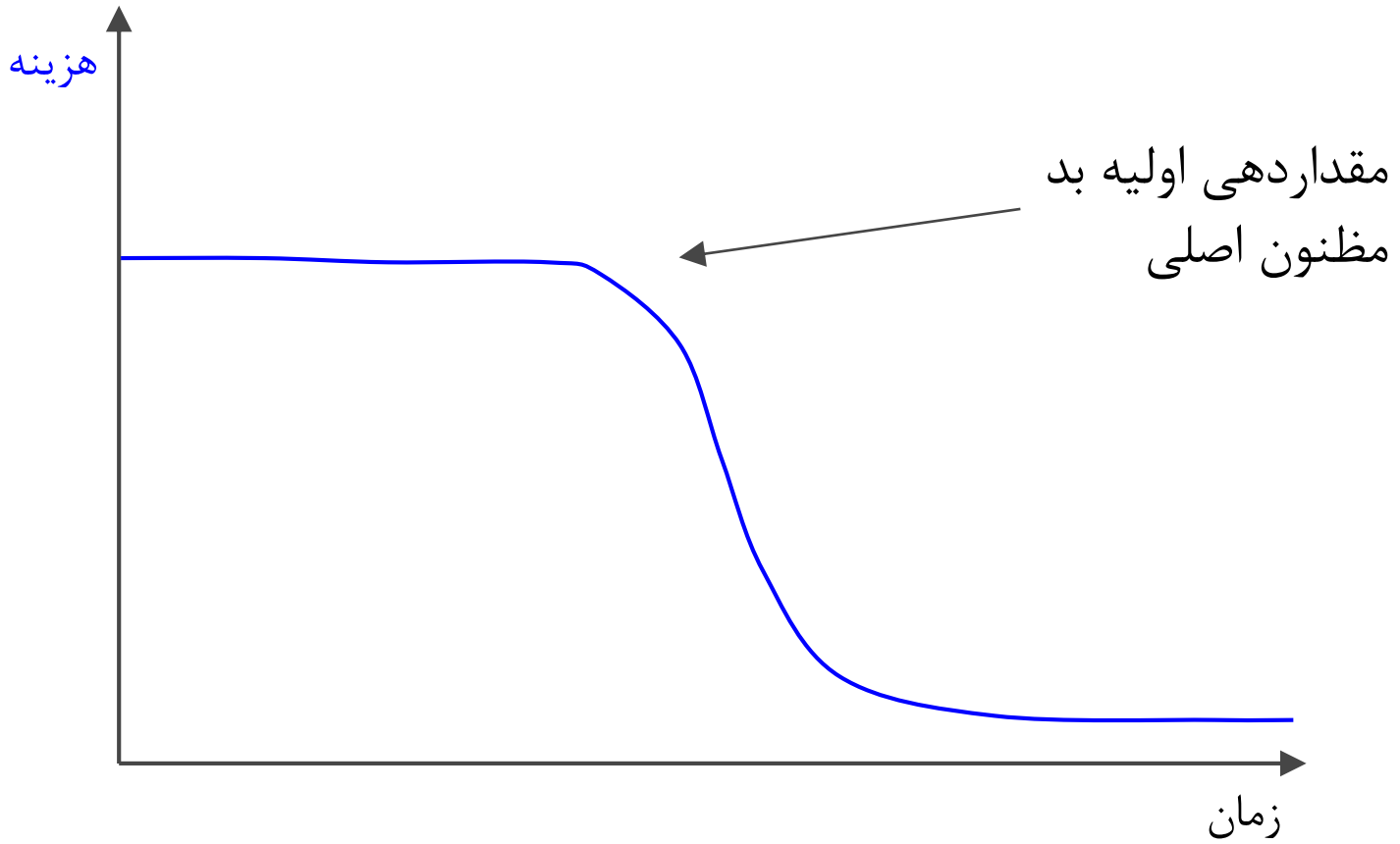
- ابرپارامترهایی که باید برای آنها مقدار تعیین نمود:
  - ساختار شبکه [تعداد و اندازه لایه‌ها]
  - نرخ یادگیری، چگونگی کاهش آن، قاعده به روز رسانی
  - چگونگی تنظیم و شدت تنظیم

نظارت و ترسیم تغییر تابع هزینه



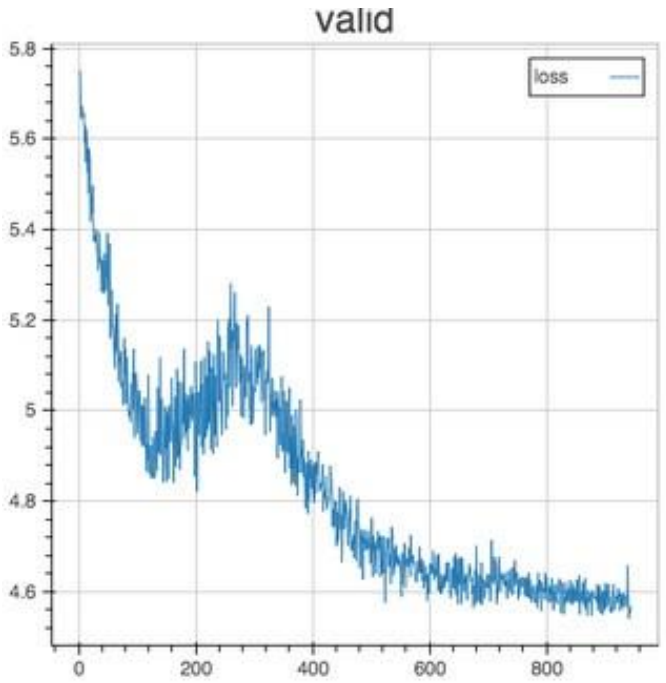


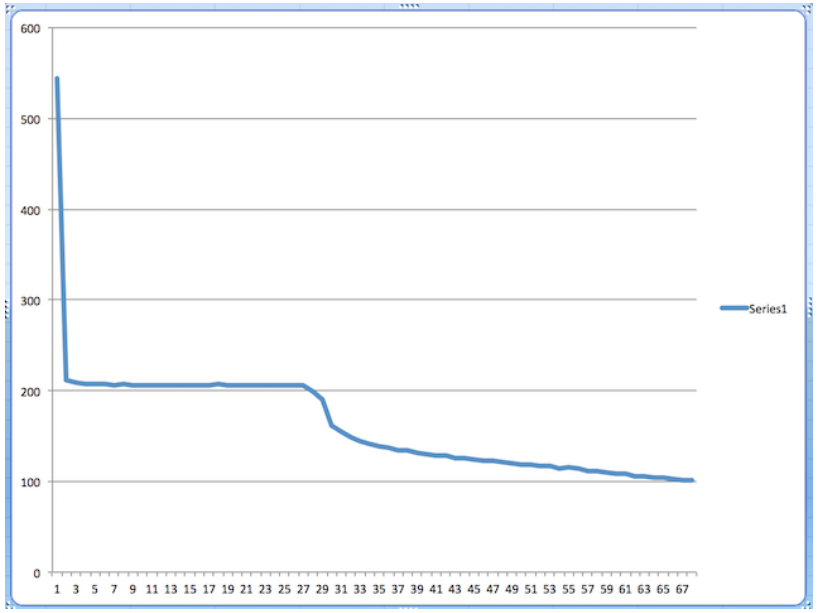
نظارت و ترسیم تغییر تابع هزینه

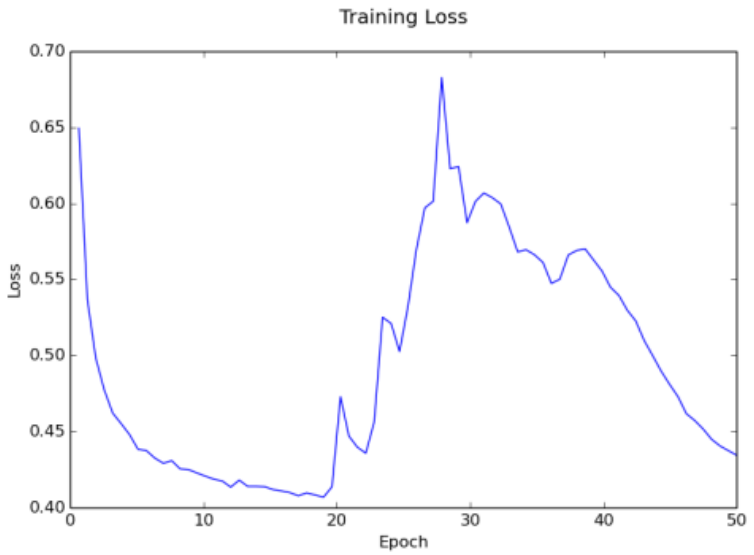


نظارت و ترسیم تغییر تابع هزینه

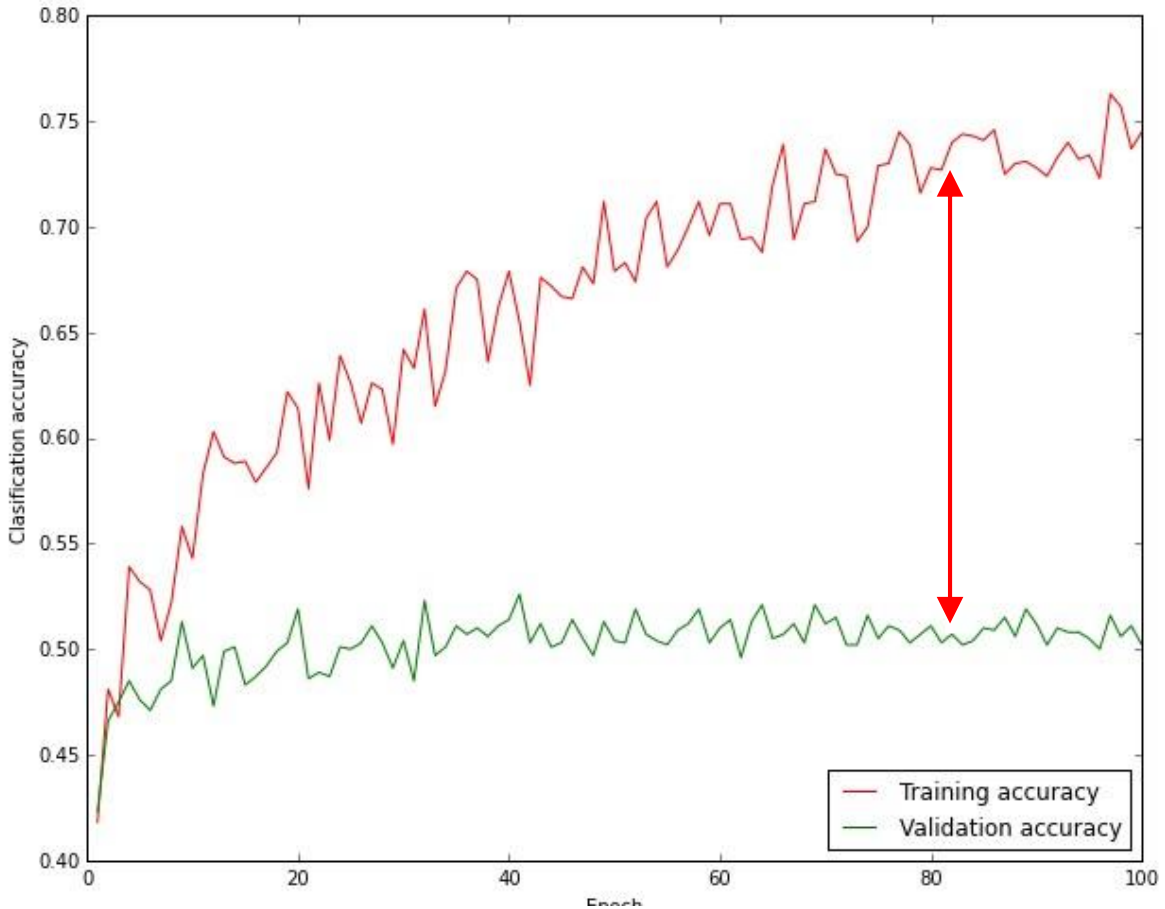
lossfunctions.tumblr.com







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
```

$||W||$

$||dW||$

$||dW|| \;/\; ||W||$

- 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 زاویر استفاده کنید]
- نرمال‌سازی دسته‌ای. [حتماً استفاده کنید]
- توسعه فرایند آموزش.
- بهینه‌سازی ابرپارامترها. [جستجوی تصادفی در فضای لگاریتمی]

جلسه آینده

- روش‌های به روز رسانی پارامترها.
- زمان‌بندی نرخ یادگیری.
- بررسی گرادیان.
- تنظیم. [حذف تصادفی]
- ارزیابی.