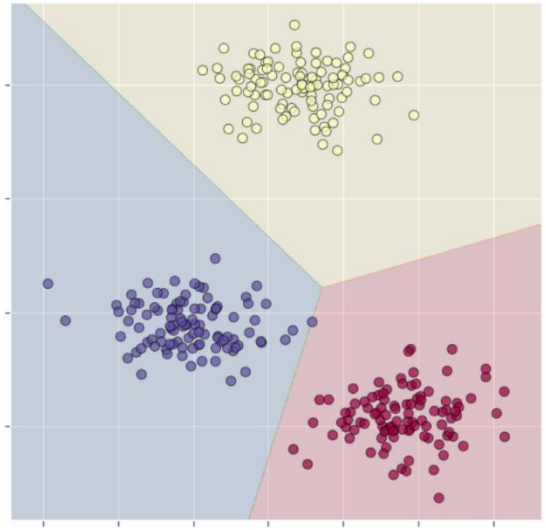
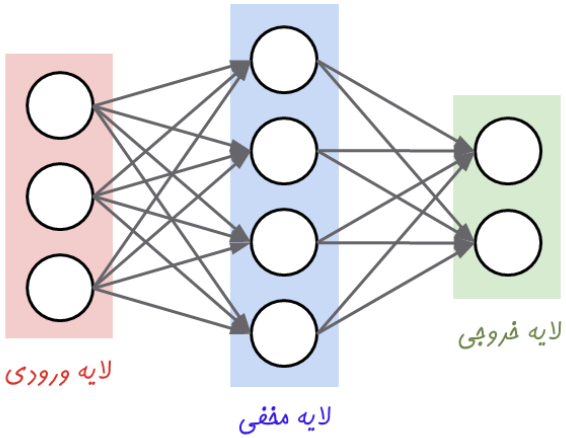
شبکه‌های عصبی مصنوعی

سید ناصر رضوی www.snrazavi.ir

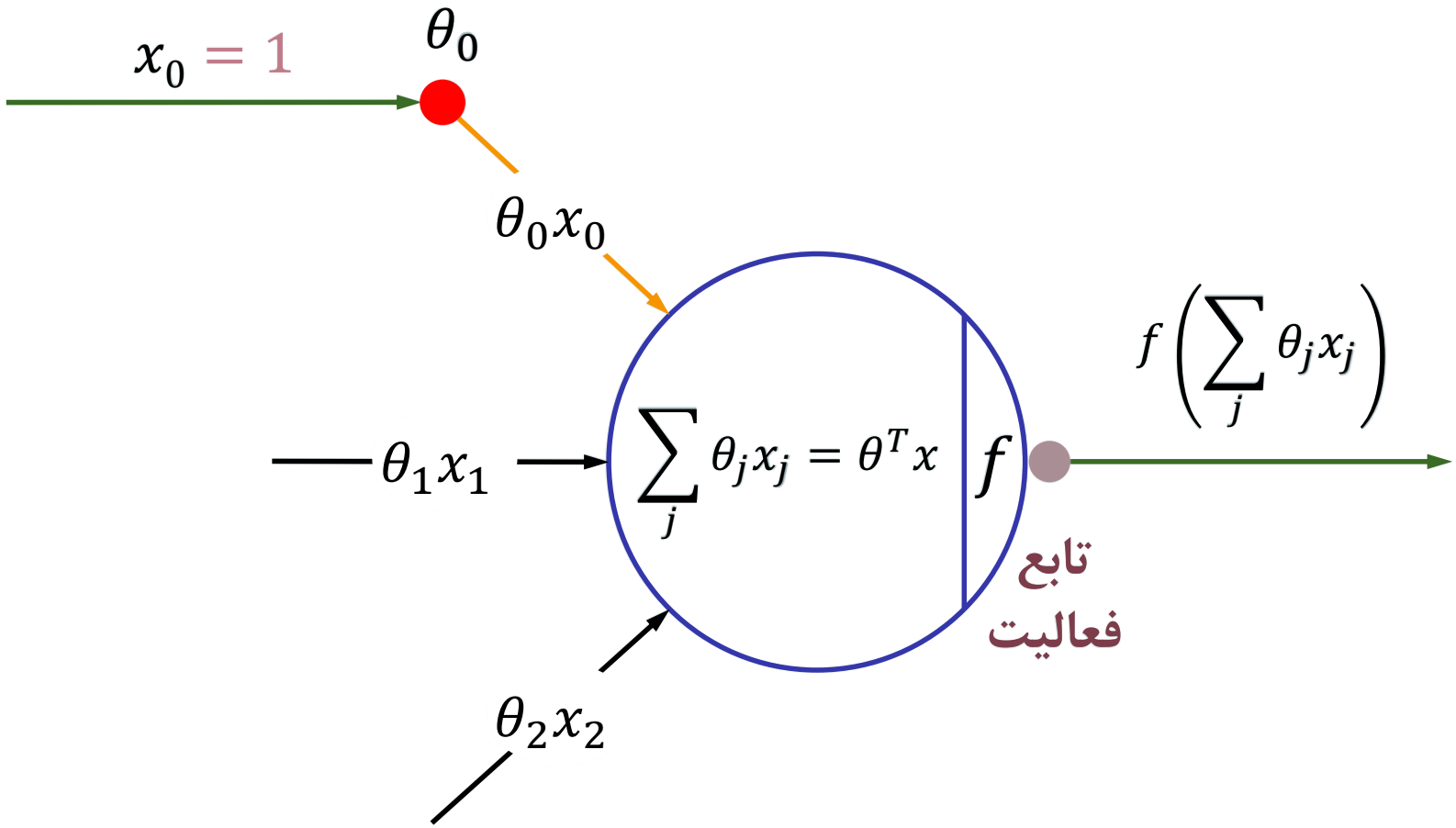
۱۳۹۷

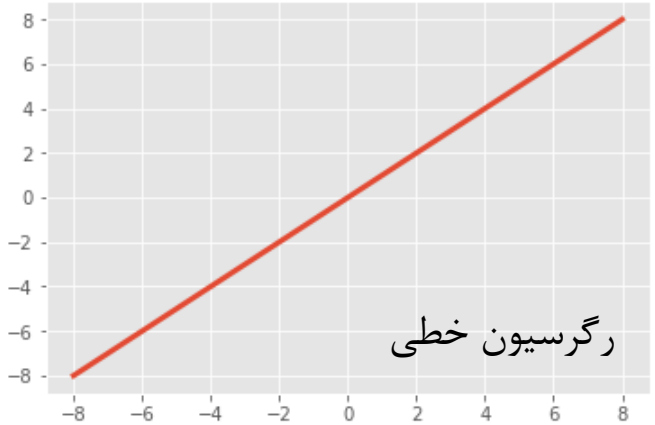
فهرست

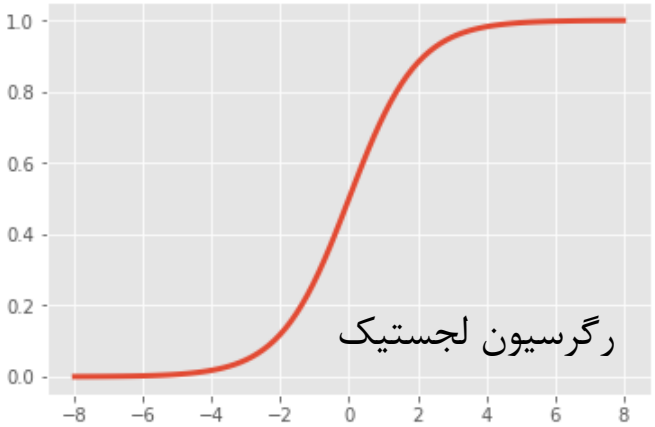




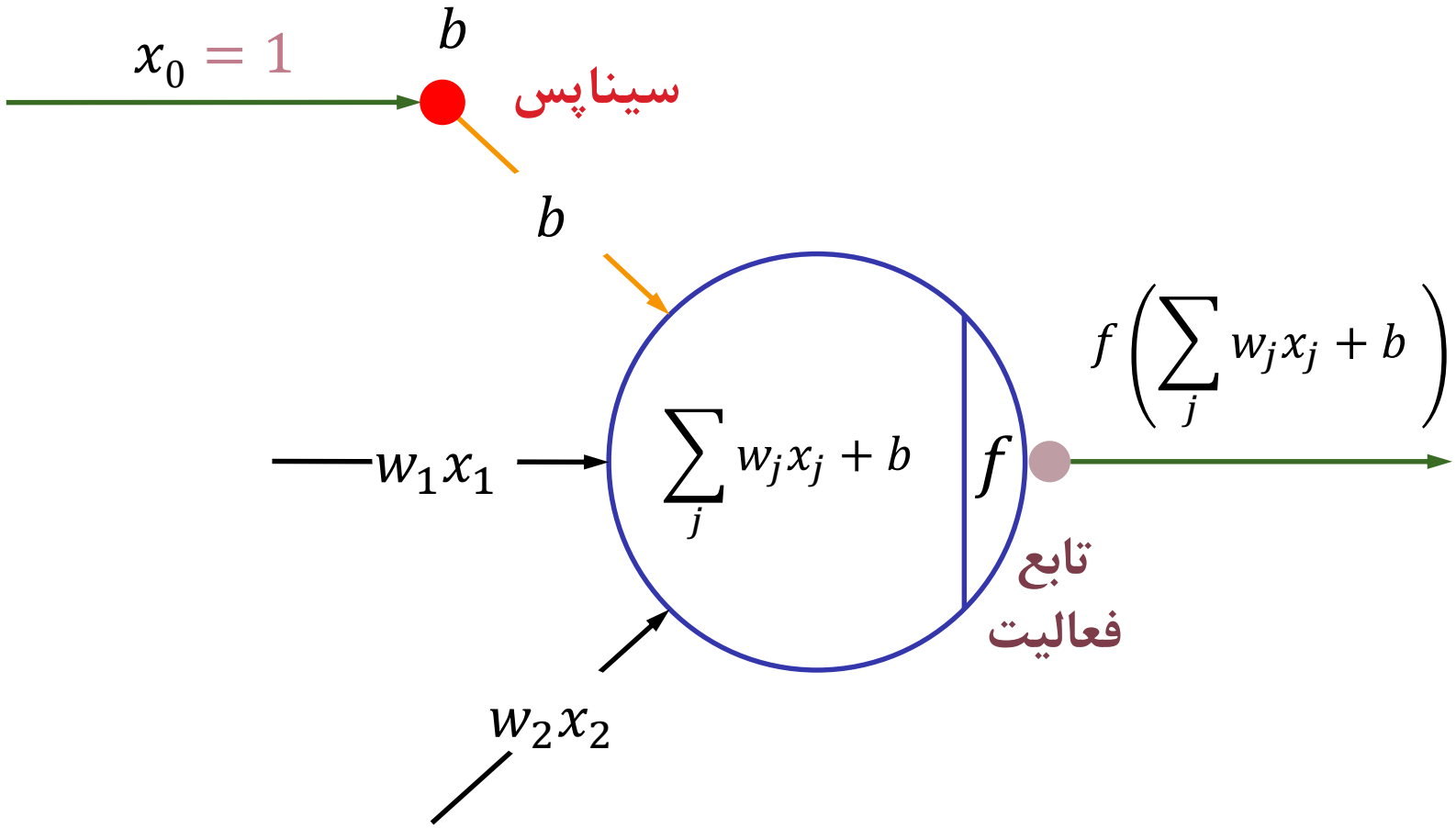
یادآوری: رگرسیون لجستیک دودویی

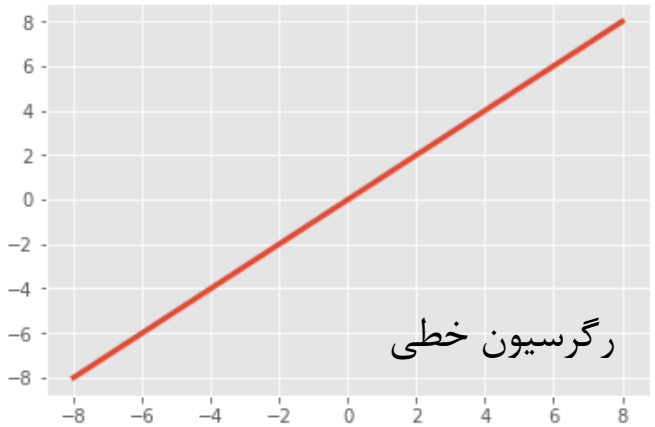


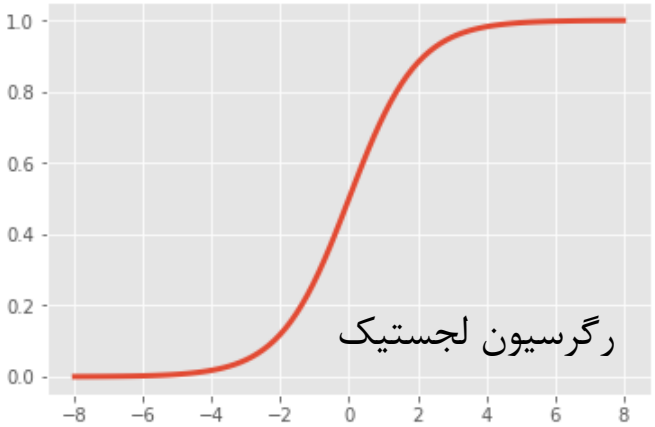




یادآوری: رگرسیون لجستیک دودویی

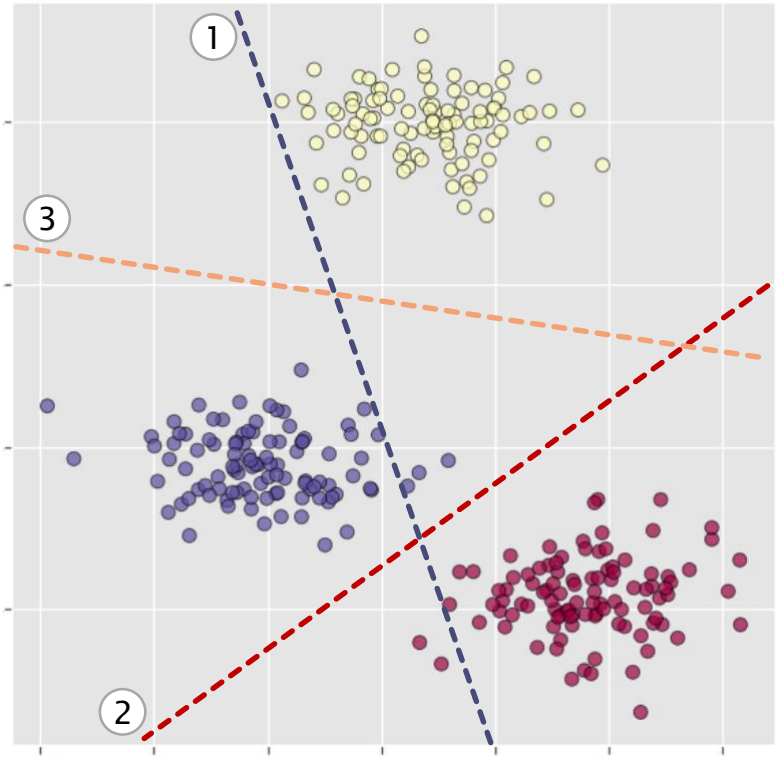






یادآوری: رگرسیون لجستیک چند دسته‌ای

- یکی در برابر بقیه. ایجاد یک دسته‌بند به ازای هر دسته.



$$h^{(1)}(x) = g\left(\left(\theta^{(1)}\right)^T x\right)$$

$$h^{(2)}(x) = g\left(\left(\theta^{(2)}\right)^T x\right)$$

$$h^{(3)}(x) = g\left(\left(\theta^{(3)}\right)^T x\right)$$

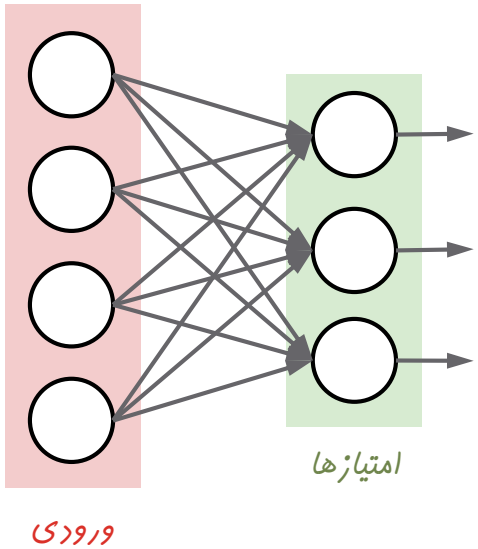
- دسته‌بندی داده جدید.

$$y = \arg\max_c h^{(c)}(x)$$

یادآوری: رگرسیون لجستیک چند دسته‌ای

- یکی در برابر بقیه. ایجاد یک دسته‌بند به ازای هر دسته.

- دسته‌بندی داده جدید.
  - مثال. چهار ویژگی و سه دسته.

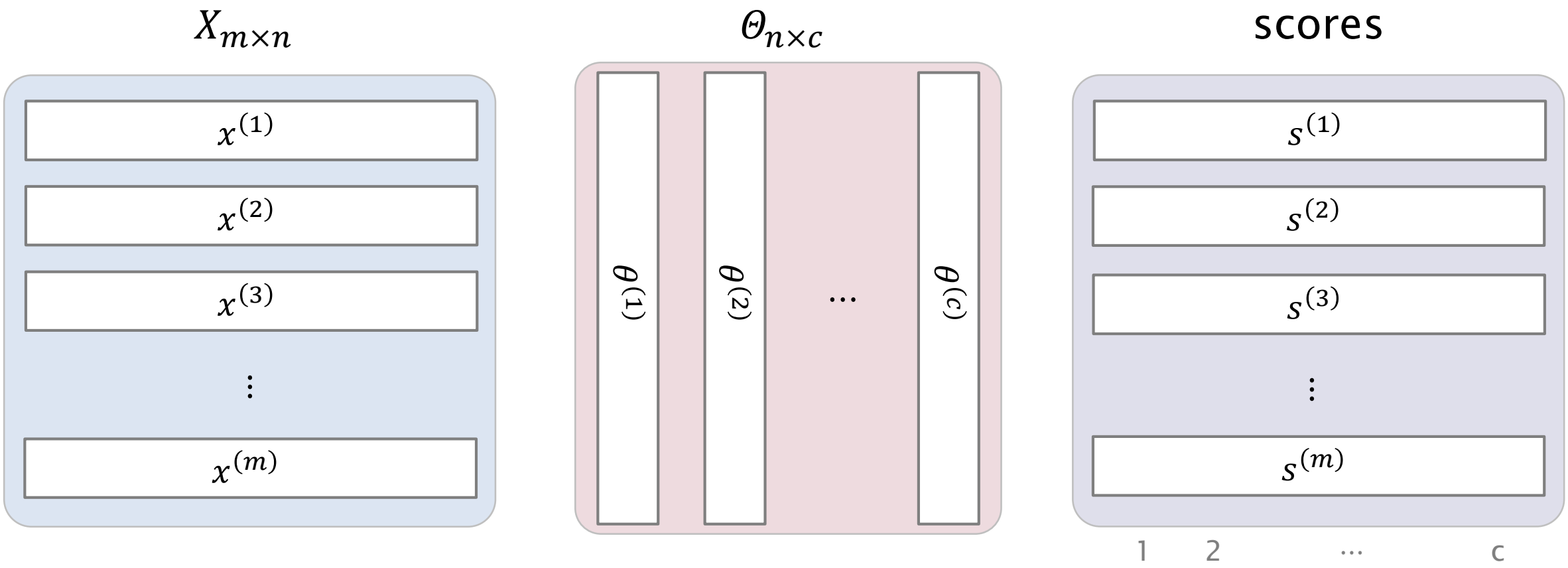


$$h^{(1)}(x) = g\left(\left(\theta^{(1)}\right)^T x\right)$$
$$h^{(2)}(x) = g\left(\left(\theta^{(2)}\right)^T x\right)$$
$$h^{(3)}(x) = g\left(\left(\theta^{(3)}\right)^T x\right)$$

دسته‌بندی

$$y = \arg\max_c h^{(c)}(x)$$

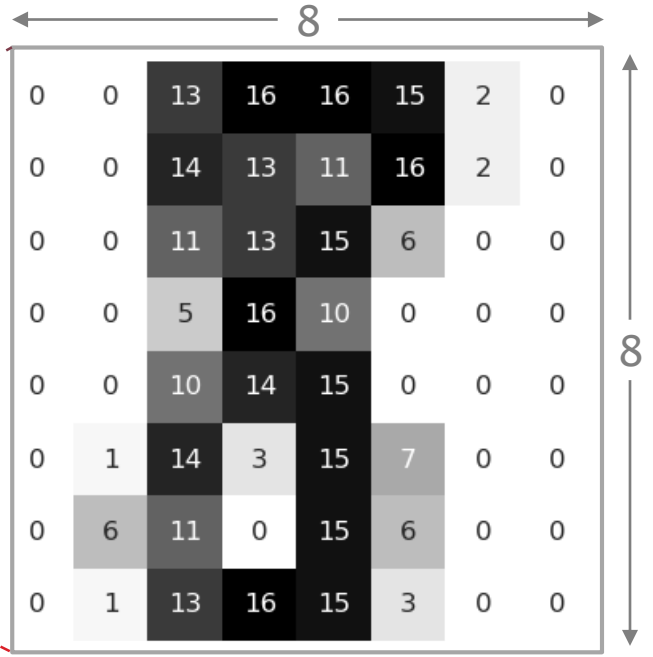
رگرسیون لجستیک چند دسته‌ای: برداری‌سازی




```
scores = X @ Theta + theta0
```

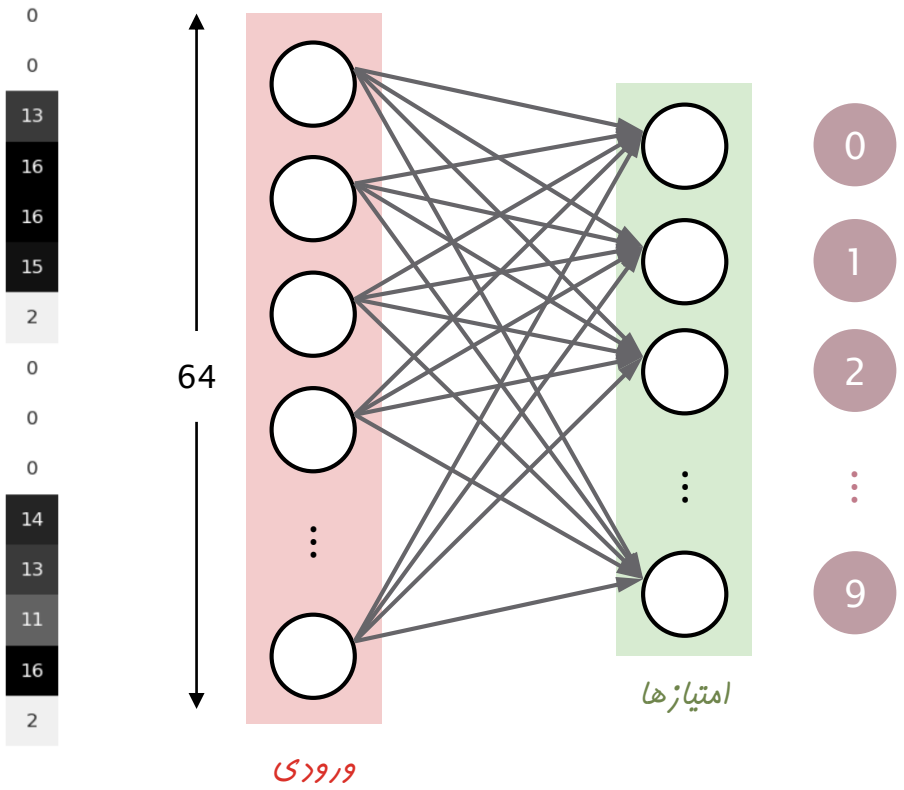
رگرسیون لجستیک چند دسته‌ای: تشخیص ارقام دست‌نویس

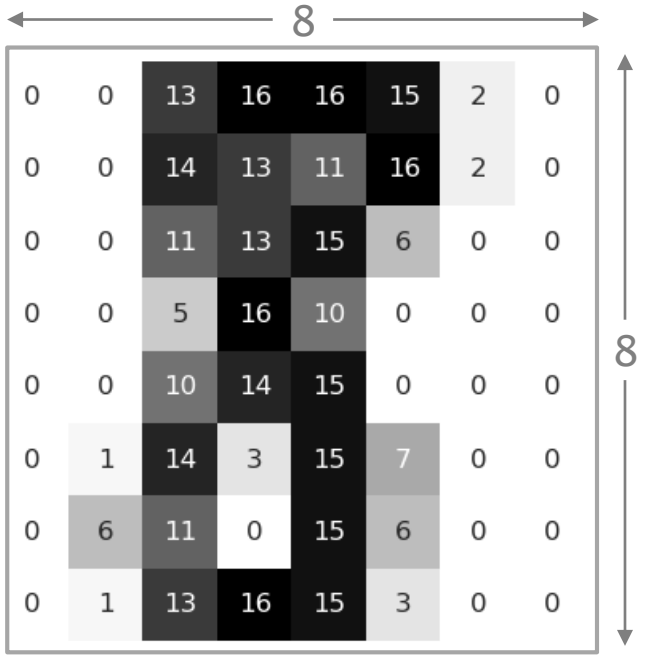
۱۰ دسته و ۶۴ ویژگی



رگرسیون لجستیک چند دسته‌ای: تشخیص ارقام دست‌نویس

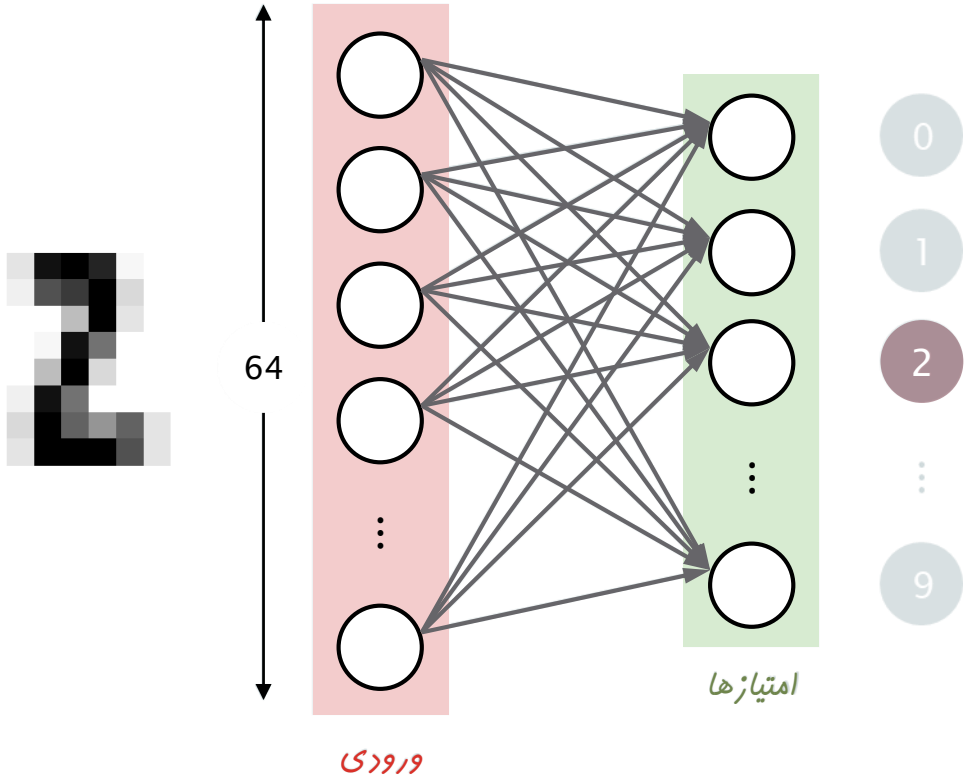
۱۰ دسته و ۶۴ ویژگی





رگرسیون لجستیک چند دسته‌ای: پیش‌بینی

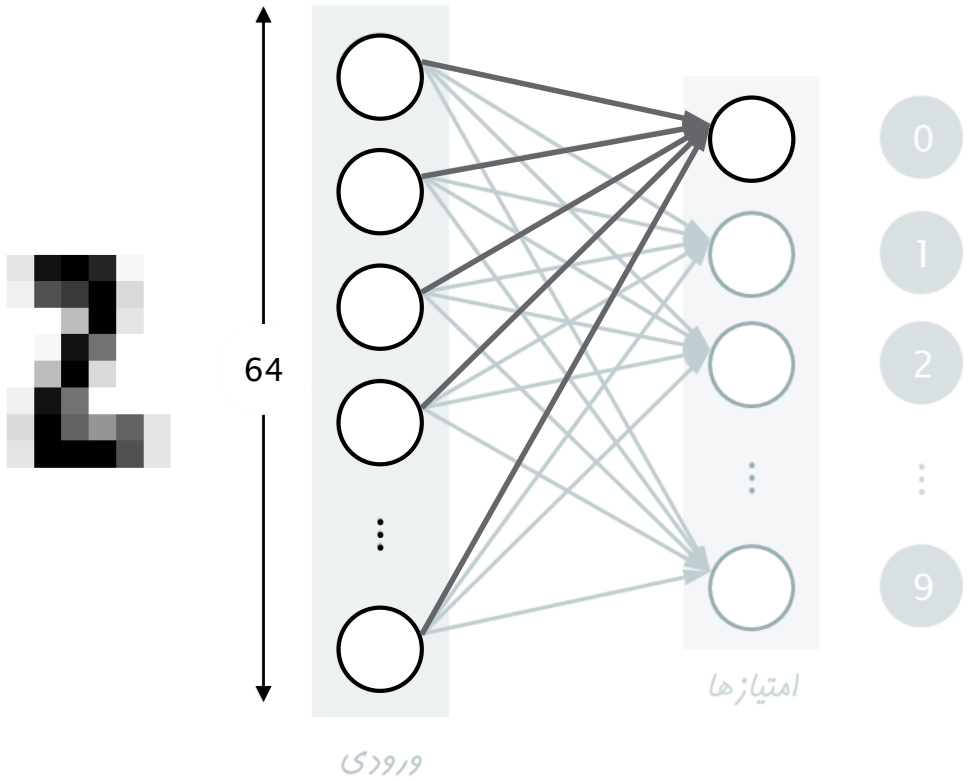
- تفسیر هندسی. محاسبه شباهت بردار ورودی با بردارهای وزن مربوط به تک‌تک دسته‌ها.




```
def predict(W, b, X):
    scores = X @ W + b
    return np.argmax(scores, axis=1)
```

رگرسیون لجستیک چند دسته‌ای: پیش‌بینی

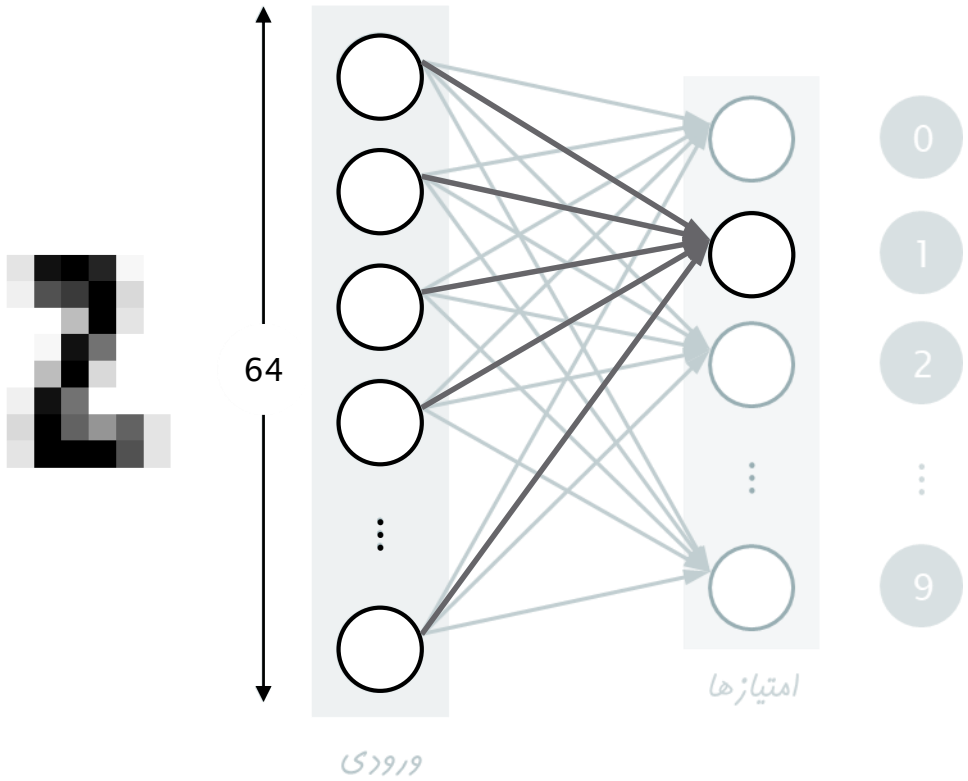
محاسبه شباهت بردار ورودی با
پارامترهای مربوط به کلاس صفر




```
def predict(W, b, X):
    scores = X @ W + b
    return np.argmax(scores, axis=1)
```

رگرسیون لجستیک چند دسته‌ای: پیش‌بینی

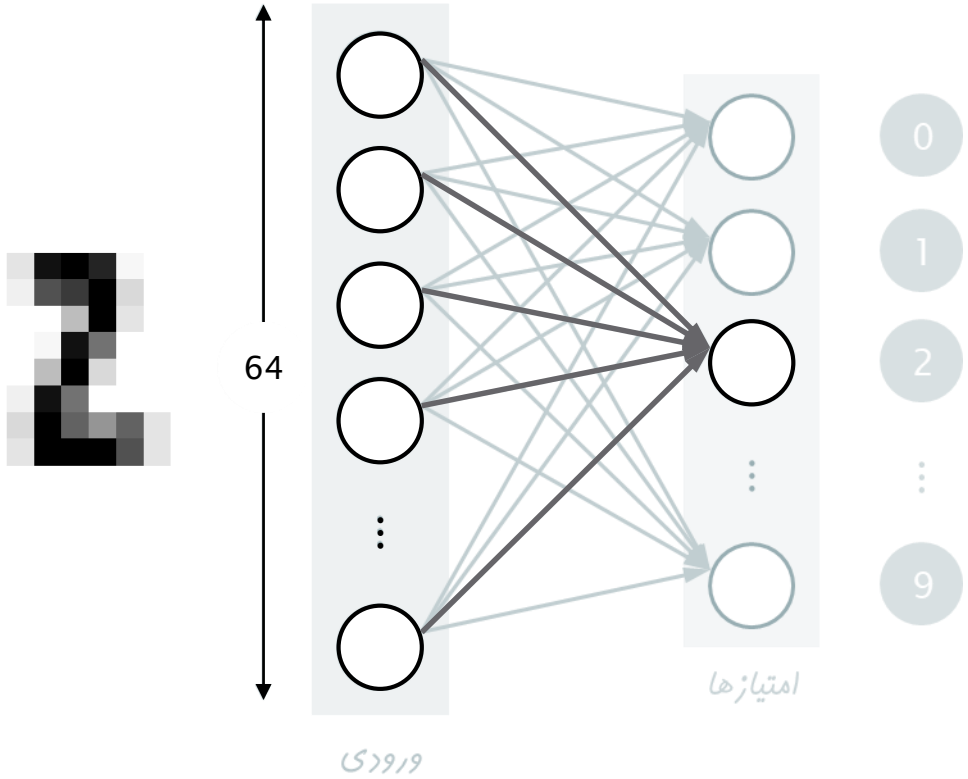
محاسبه شباهت بردار ورودی با
پارامترهای مربوط به کلاس یک




```
def predict(W, b, X):
    scores = X @ W + b
    return np.argmax(scores, axis=1)
```

رگرسیون لجستیک چند دسته‌ای: پیش‌بینی

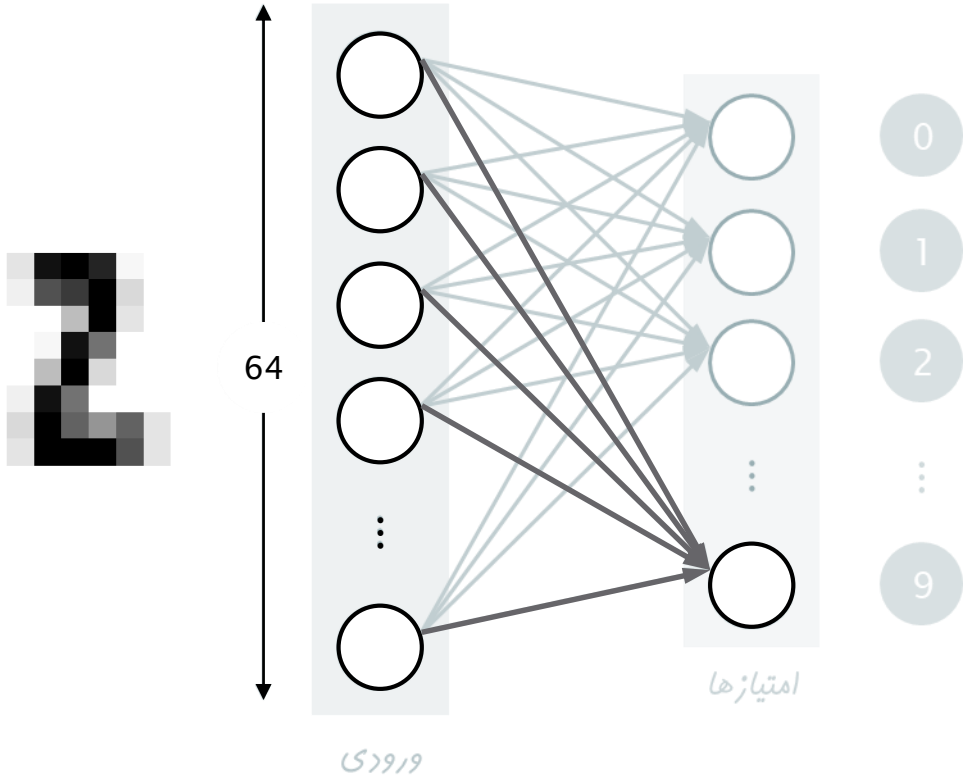
محاسبه شباهت بردار ورودی با
پارامترهای مربوط به کلاس دو




```
def predict(W, b, X):
    scores = X @ W + b
    return np.argmax(scores, axis=1)
```

رگرسیون لجستیک چند دسته‌ای: پیش‌بینی

محاسبه شباهت بردار ورودی با
پارامترهای مربوط به کلاس نُه

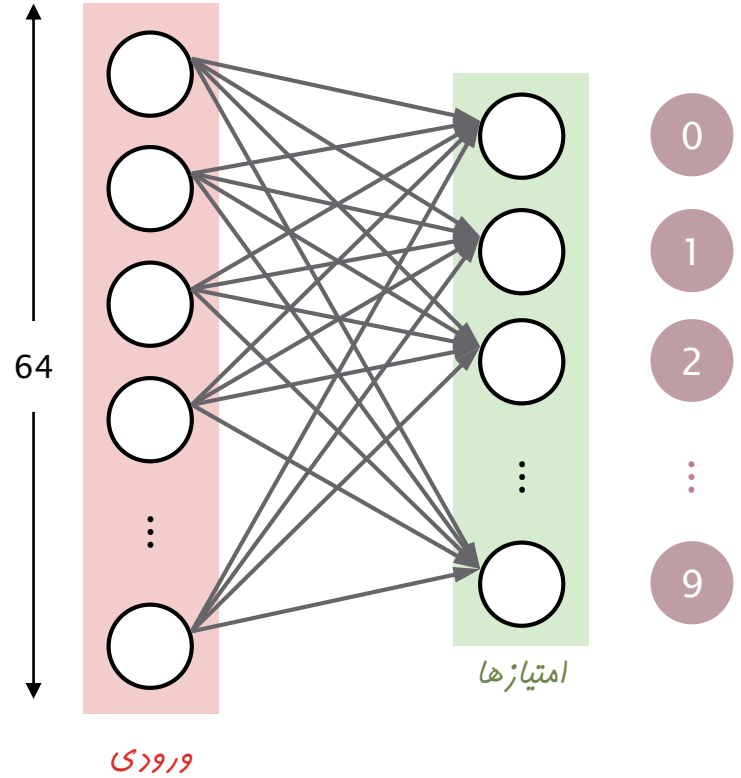



```
def predict(W, b, X):
    scores = X @ W + b
    return np.argmax(scores, axis=1)
```

دسته‌بند سافت‌مکس

رگرسیون لجستیک دودویی: تابع سیگموید

- دسته‌بندی دودویی. مقدار تابع فرضیه بیانگر احتمال تعلق داده ورودی به دسته ۱ است.

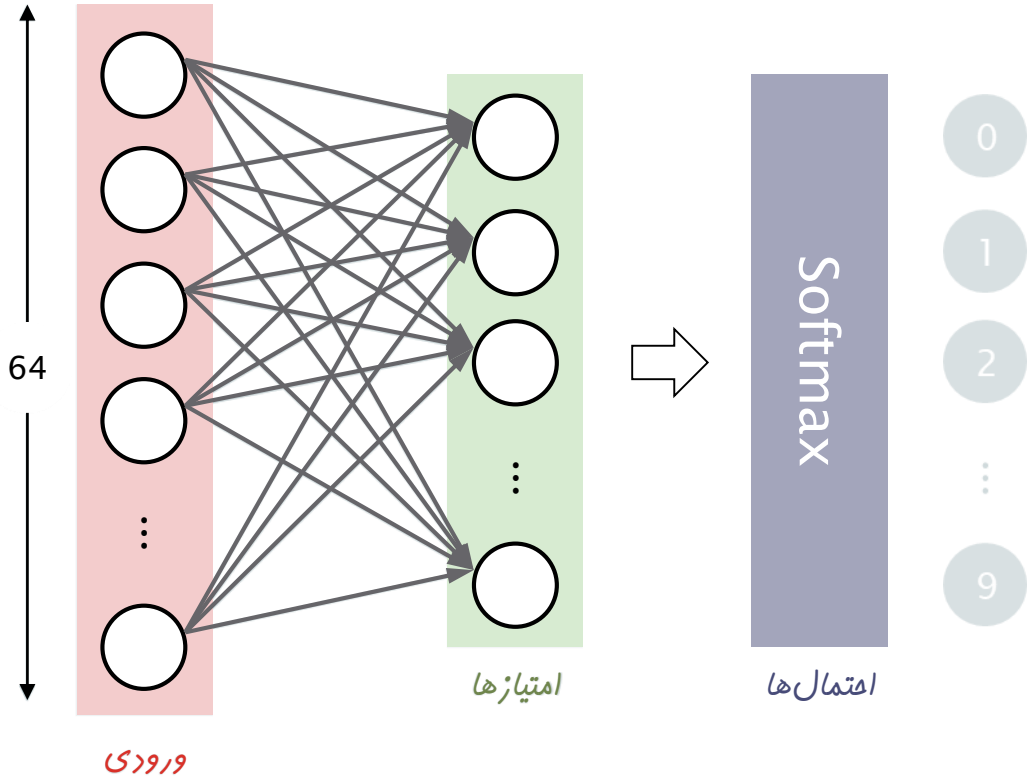



```
p = sigmoid(X @ w + b)
```

توجه. در صورت استفاده از تابع سیگموید (لجستیک) در یک مسئله دسته‌بندی چند دسته‌ای، دیگر مجموع مقادیر فرضیه‌ها لزوماً برابر با یک نخواهد بود.

رگرسیون لجستیک چند دسته‌ای: تابع سافت‌مکس

- تابع سافت‌مکس. در دسته‌بندی چند دسته‌ای به منظور محاسبه احتمال تعلق بردار ورودی به هر یک از دسته‌های مختلف، به جای تابع سیگموید از تابع سافت‌مکس استفاده می‌کنیم.




```
p = softmax(X @ W + b)
```

$$softmax\left(s^{(i)}\right)_k = \frac{e^{s_k}}{\sum_{j=1}^{c} e^{s_j}} = p_k^{(i)}$$

احتمال تعلق داده ورودی $x^{(i)}$ به کلاس $y = k$

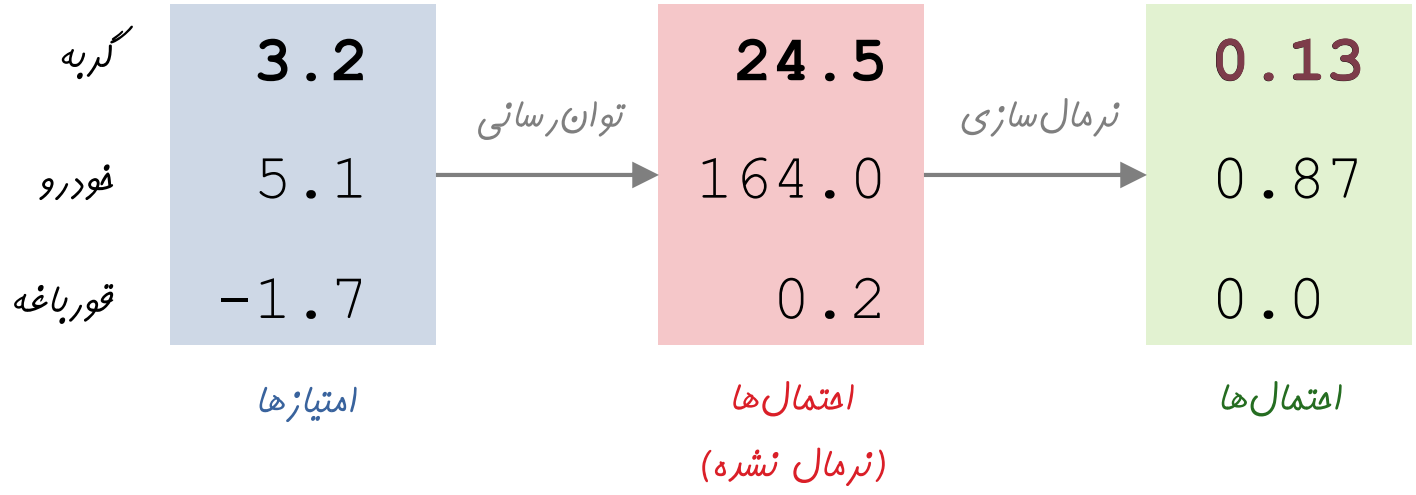
دسته‌بند سافت‌مکس (رگرسیون لجستیک چند دسته‌ای)

- ایده. تبدیل بردار امتیازها به یک بردار توزیع احتمال!

$(x^{(i)}, y^{(i)})$

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

$$L_i = -\log(0.13) = \mathbf{0.89}$$



کمترین و بیشترین مقدار ممکن برای تابع هزینه چقدر است؟

دسته‌بند سافت‌مکس (رگرسیون لجستیک چند دسته‌ای)

- امتیازها. لگاریتم احتمال دسته‌ها به صورت نرمال نشده!

$$s = f(x^{(i)}; W)$$

$$P(Y = k | X = x^{(i)}) = \frac{e^{s_k}}{\sum_j e^{s_j}}$$

تابع سافت‌مکس

$(x^{(i)}, y^{(i)})$

گربه	**3.2**
خودرو	5.1
قورباغه	-1.7

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

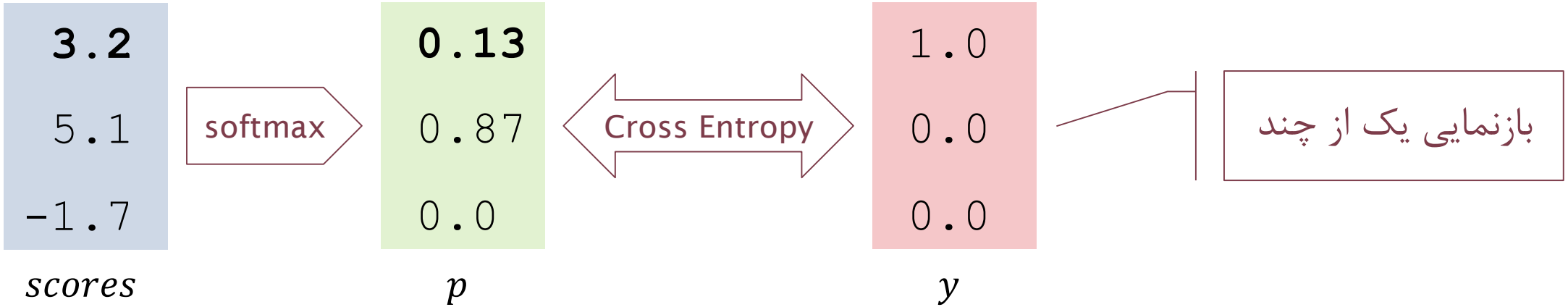
$$L_i = -\log P(Y = y^{(i)} | X = x^{(i)}) = -\log\left(\frac{e^{s_{y^{(i)}}}}{\sum_j e^{s_j}}\right)$$

تابع هزینه سافت‌مکس: پیاده‌سازی


```
def softmax_loss(scores, y):
      # softmax loss implementation.
      # y is encoded as a one-hot vector.
      p = softmax(scores)
      return -np.sum(y * np.log(p))
```

$$(x^{(i)}, y^{(i)})$$

$$L_i = \sum_{k=1}^{c} -y_k \log p_k = -\log p_{y^{(i)}}$$



دسته‌بند سافت‌مکس: آموزش

- مجموعه آموزشی.

$$\{(x^{(i)}, y^{(i)})\}_{i=1}^{m} \qquad x^{(i)} = [x_1^{(i)} \quad x_2^{(i)} \quad \dots \quad x_n^{(i)}]^T \qquad y^{(i)} \in \{1,2,\dots,c\}$$

- پارامترها.

$$W \in \mathbb{R}^{n \times c} \qquad b \in \mathbb{R}^{c}$$

- تابع هزینه.

$$L(W,b) = \frac{1}{m}\sum_{i=1}^{m} L_i + \lambda R(W) \quad = \frac{1}{m}\sum_{i=1}^{m} \left(-\log p_{y^{(i)}}\right) + \lambda \|W\|_2^2$$

دسته‌بند سافت‌مکس: آموزش

- تابع هزینه.

$$L(W,b) = \frac{1}{m}\sum_{i=1}^{m}\left(-\log p_{y^{(i)}}\right) + \lambda\|W\|_2^2$$

$$= \frac{1}{m}\sum_{i=1}^{m}\left(-\log\left(\frac{e^{s_{y^{(i)}}}}{\sum_{j=1}^{c} e^{s_j}}\right)\right) + \lambda\|W\|_2^2$$

$$= \frac{1}{m}\sum_{i=1}^{m}\left(-\log\left(\frac{e^{\left(\left(w^{y^{(i)}}\right)^T x^{(i)} + b^{y^{(i)}}\right)}}{\sum_{j=1}^{c} e^{\left(\left(w^{(j)}\right)^T x^{(i)} + b^{(j)}\right)}}\right)\right) + \lambda\|W\|_2^2$$

دسته‌بند سافت‌مکس: آموزش

$$L_i = -\log p_{y^{(i)}} \quad p_k = \frac{e^{s_k}}{\sum_{j=1}^{c} e^{s_j}} \quad s_k = \left(w^{(k)}\right)^T x^{(i)} + b^{(k)}$$

$$\frac{\partial L_i}{\partial w^{(k)}} = ? \quad k \in \{1,2,\cdots,c\}$$

$$\frac{\partial L_i}{\partial w^{(k)}} = \frac{\partial L_i}{\partial p_{y^{(i)}}} \cdot \frac{\partial p_{y^{(i)}}}{\partial s_k} \cdot \frac{\partial s_k}{\partial w^{(k)}}$$

$$= \left(-\frac{1}{p_{y^{(i)}}}\right) p_k(1 - p_k) \; x^{(i)}$$

$$= (p_k - 1)x^{(i)}$$

$k = y^{(i)}$

$$\frac{\partial L_i}{\partial w^{(k)}} = \frac{\partial L_i}{\partial p_{y^{(i)}}} \cdot \frac{\partial p_{y^{(i)}}}{\partial s_k} \cdot \frac{\partial s_k}{\partial w^{(k)}}$$

$$= \left(-\frac{1}{p_{y^{(i)}}}\right) p_{y^{(i)}}(-p_k) \; x^{(i)}$$

$$= (p_k)x^{(i)}$$

$k \neq y^{(i)}$

دسته‌بند سافت‌مکس: آموزش

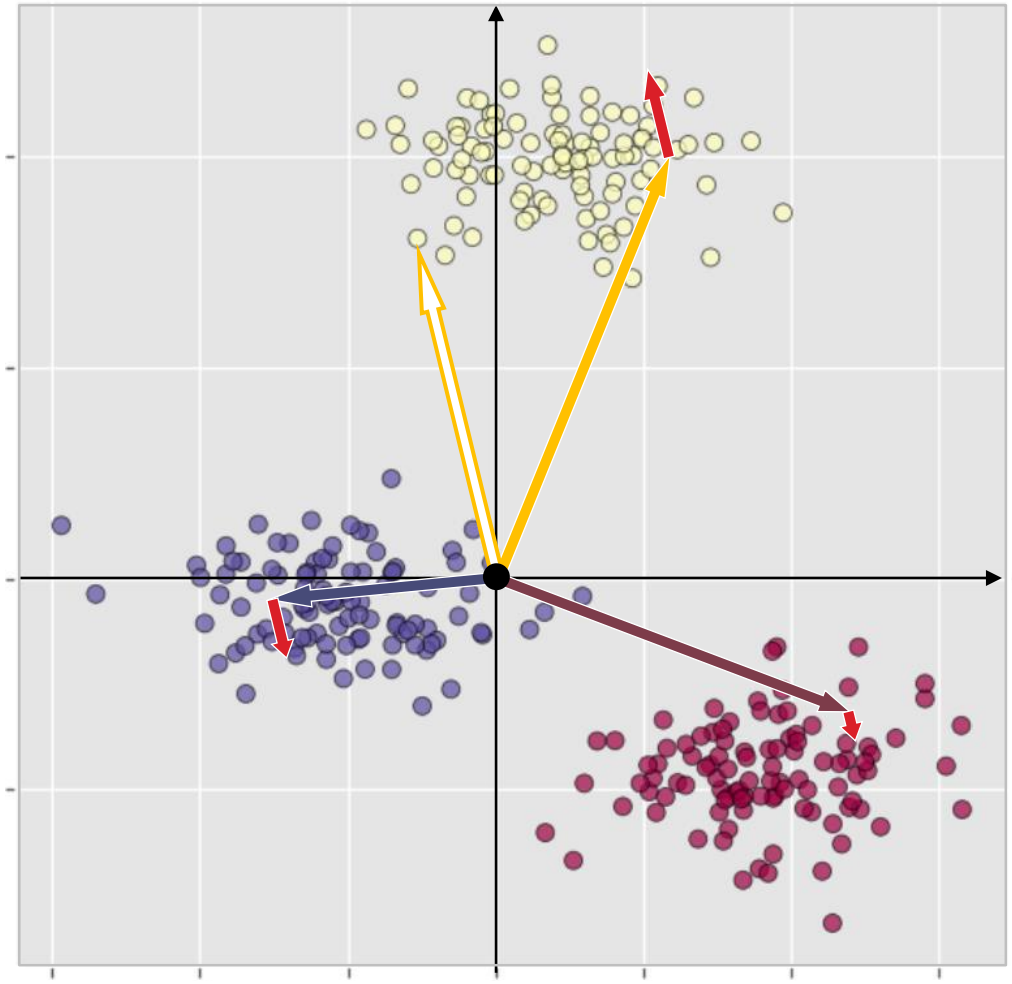
$$L_i = -\log p_{y^{(i)}} = -\log\left(\frac{e^{s_{y^{(i)}}}}{\sum_{j=1}^{c} e^{s_j}}\right)$$

- الگوریتم گرادیان کاهشی اتفاقی.

$$\frac{\partial L_i}{\partial w^{(k)}} = \left(p_{y_k} - 1\right)x^{(i)} \rightarrow \boxed{w^{(k)} = w^{(k)} - \alpha\left(p_{y_k} - 1\right)x^{(i)}} \quad \left(k = y^{(i)}\right)$$

$$\frac{\partial L_i}{\partial w^{(k)}} = \left(p_{y_k}\right)x^{(i)} \rightarrow \boxed{w^{(k)} = w^{(k)} - \alpha\left(p_{y_k}\right)x^{(i)}} \quad \left(k \neq y^{(i)}\right)$$

دسته‌بند سافت‌مکس: آموزش



- تفسیر هندسی.

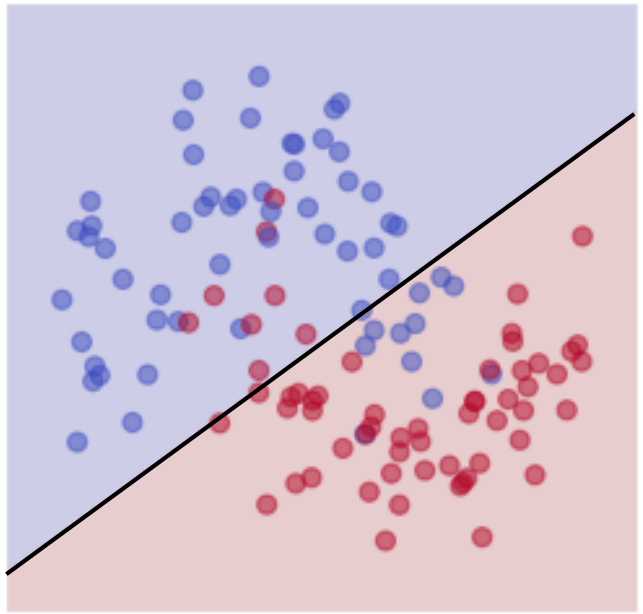
$$w^{(k)} = w^{(k)} - \alpha\left(p_{y_k} - 1\right)x^{(i)} \qquad \left(k = y^{(i)}\right)$$

$$w^{(k)} = w^{(k)} - \alpha\left(p_{y_k}\right)x^{(i)} \qquad \left(k \neq y^{(i)}\right)$$

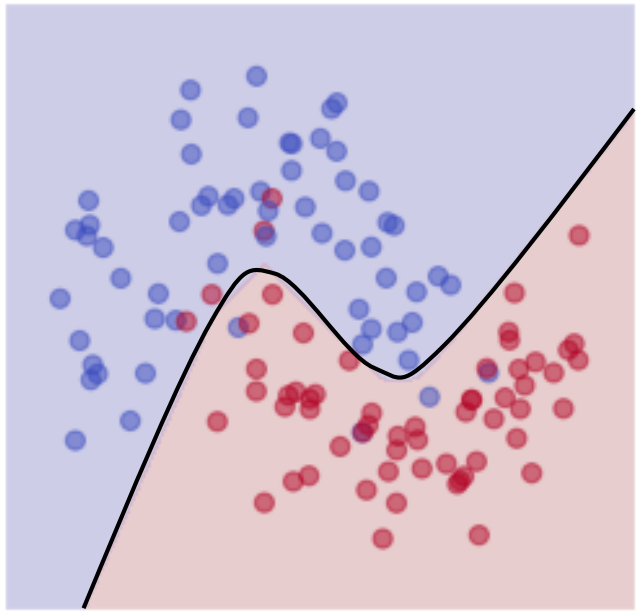
شبکه‌های عصبی

شبکه‌های عصبی: انگیزه

- رگرسیون لجستیک یک روش دسته‌بندی خطی است.
  - اگر داده‌ها به صورت خطی تفکیک‌پذیر نباشند، نیاز به افزودن ویژگی‌های مرتبه بالاتر داریم.



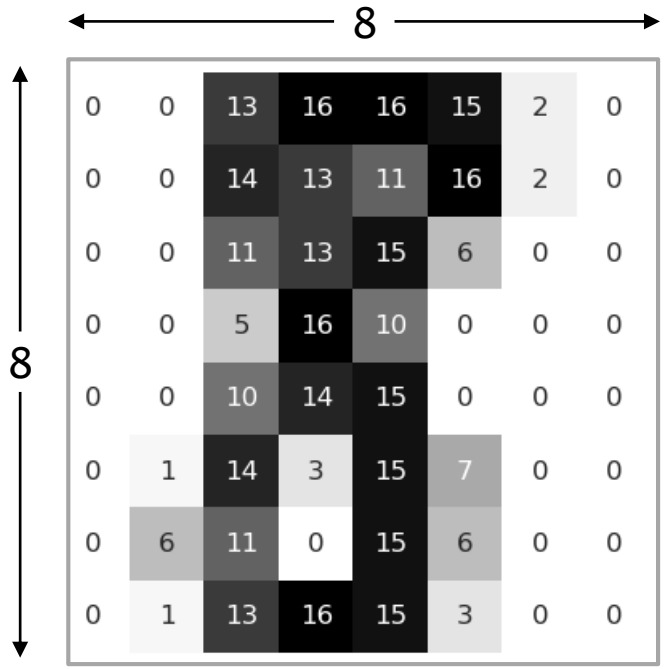
مرز تصمیم‌گیری خطی



مرز تصمیم‌گیری غیرخطی

شبکه‌های عصبی: انگیزه

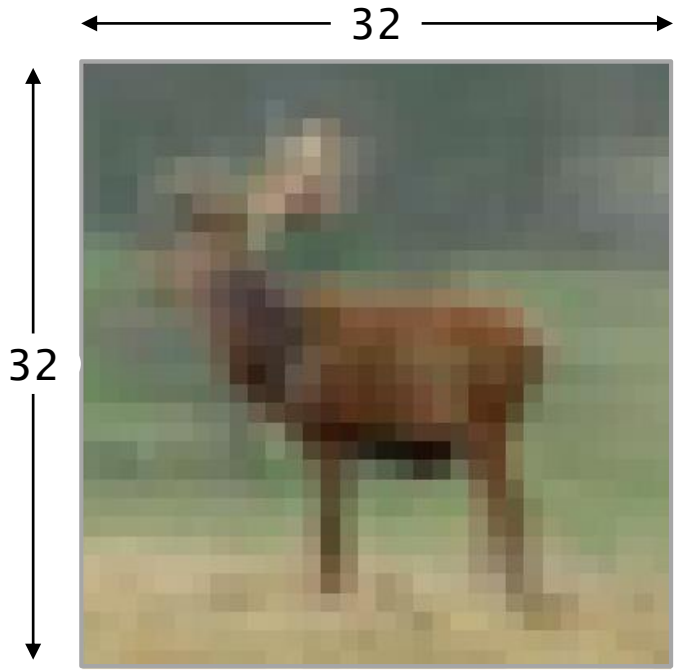
- رگرسیون لجستیک یک روش دسته‌بندی خطی است.
  - اگر داده‌ها به صورت خطی تفکیک‌پذیر نباشند، نیاز به افزودن ویژگی‌های مرتبه بالاتر داریم.



- مثال. تشخیص ارقام دست‌نویس [تصاویر ۸ در ۸]
  - تعداد ویژگی‌های مرتبه دوم: بیش از ۲,۰۰۰
  - تعداد ویژگی‌های مرتبه سوم: بیش از ۴۰,۰۰۰
  - ...

شبکه‌های عصبی: انگیزه

- رگرسیون لجستیک یک روش دسته‌بندی خطی است.
  - اگر داده‌ها به صورت خطی تفکیک‌پذیر نباشند، نیاز به افزودن ویژگی‌های مرتبه بالاتر داریم.

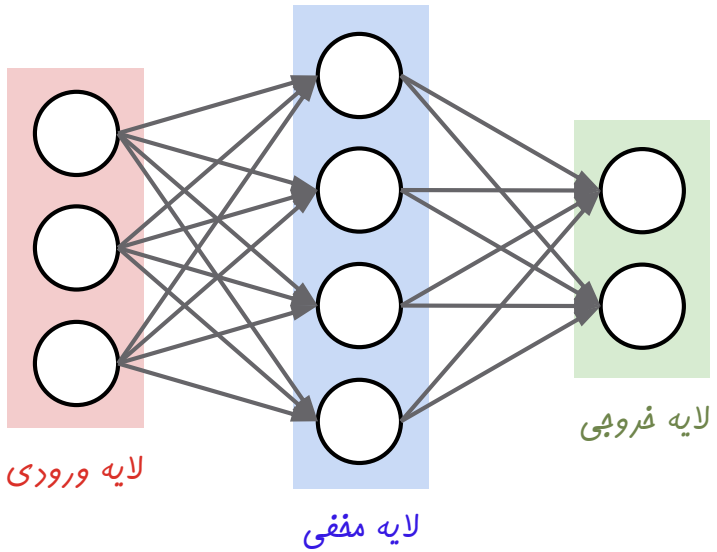


- مثال. تصاویر رنگی [۳۲ در ۳۲ در ۳]
  - تعداد ویژگی‌های مرتبه دوم: بیش از ۴۷۰,۰۰۰
  - تعداد ویژگی‌های مرتبه سوم: بیش از ۵,۰۰۰,۰۰۰
  - ...

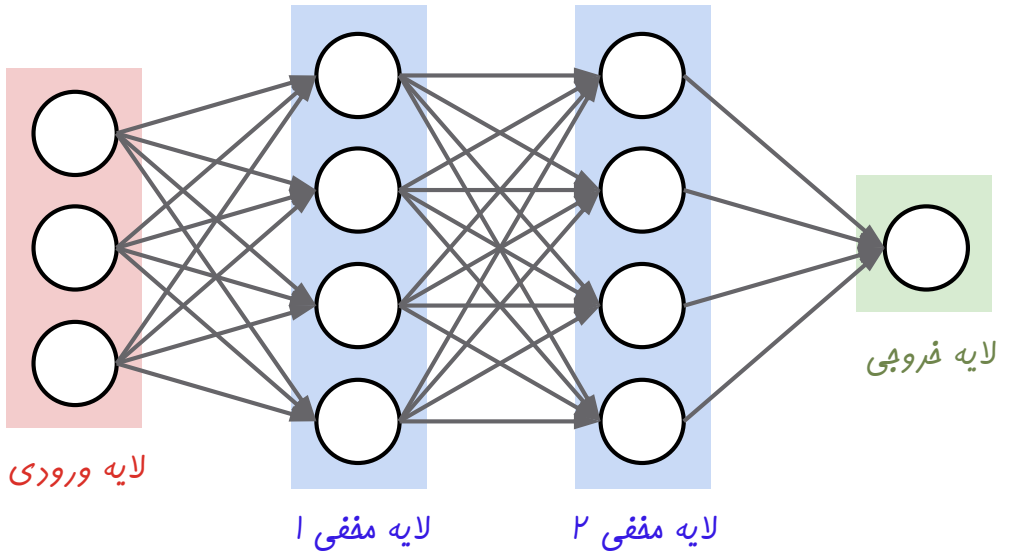
می‌توانیم بسیاری از ویژگی‌ها را با استفاده از تنظیم حذف کنیم!!!

شبکه‌های عصبی: یادگیری ویژگی‌های جدید

- شبکه‌های عصبی می‌توانند با ترکیب ویژگی‌های سطح پایین، ویژگی‌های سطح بالای مورد نیاز خود را یاد بگیرند.



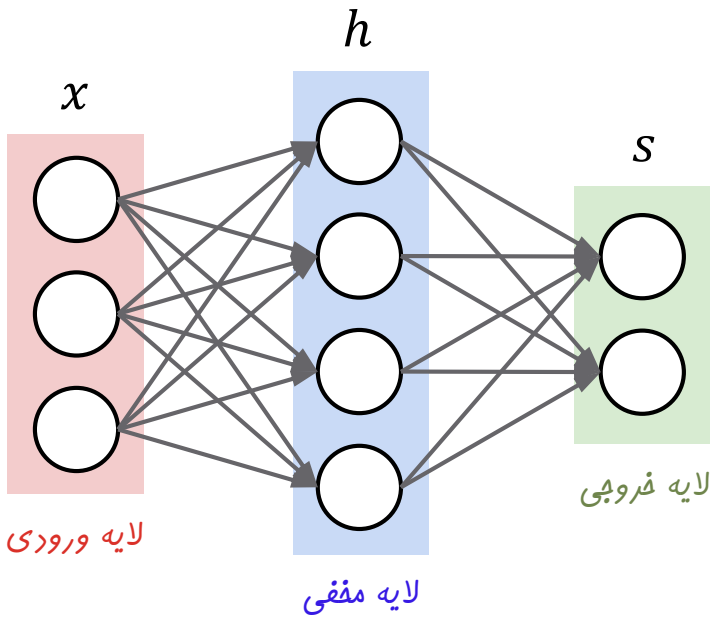
شبکه عصبی ۲ لایه



شبکه عصبی ۳ لایه

شبکه‌های عصبی: یادگیری ویژگی‌های جدید

- شبکه‌های عصبی می‌توانند با ترکیب ویژگی‌های سطح پایین، ویژگی‌های سطح بالای مورد نیاز خود را یاد بگیرند.



دسته‌بندی خطی

$$s = Wx + b$$

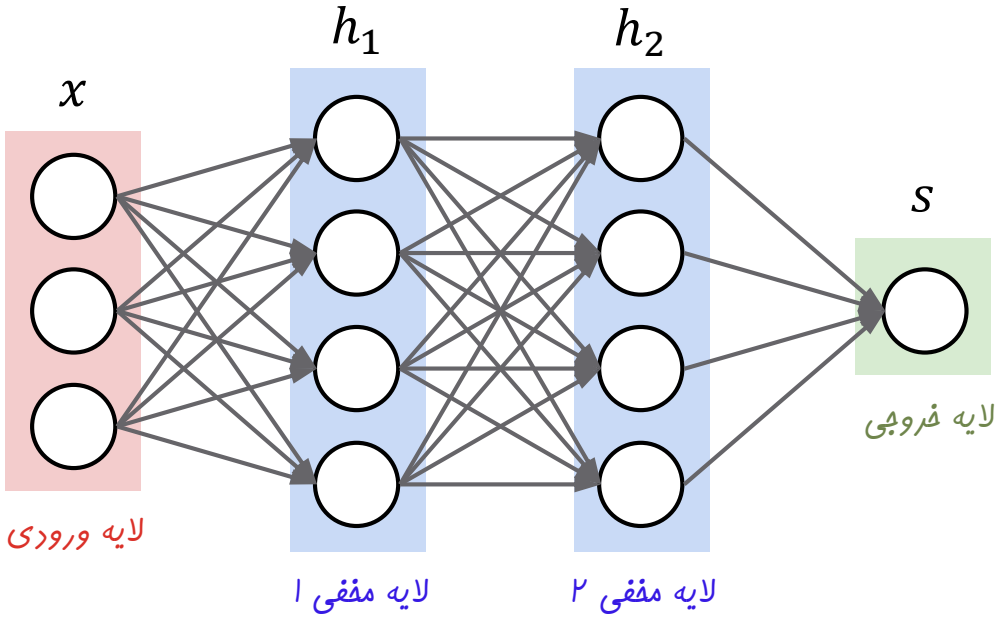
شبکه عصبی دو لایه

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

$$s = W_2 h + b_2$$

شبکه‌های عصبی: یادگیری ویژگی‌های جدید

- شبکه‌های عصبی می‌توانند با ترکیب ویژگی‌های سطح پایین، ویژگی‌های سطح بالای مورد نیاز خود را یاد بگیرند.



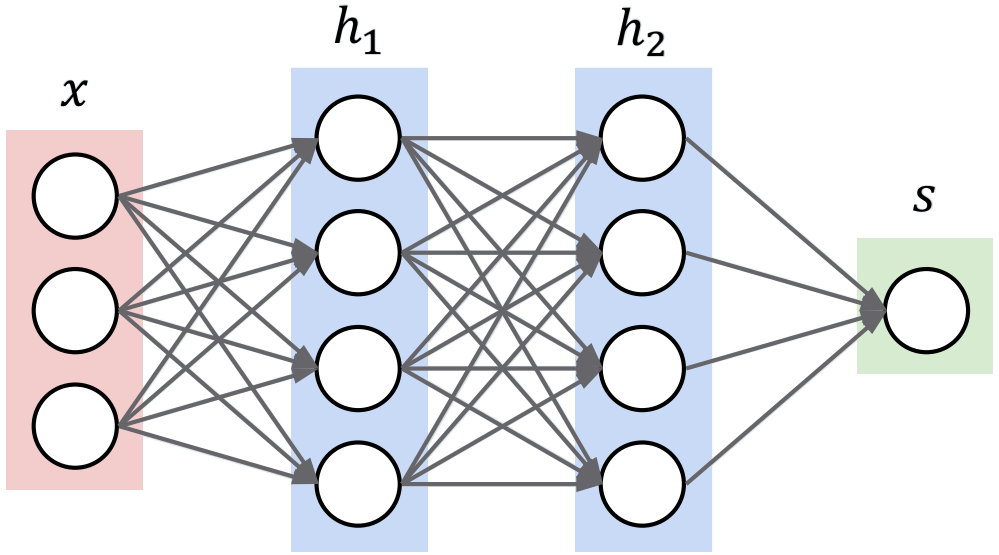
شبکه عصبی دو لایه

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

شبکه عصبی سه لایه

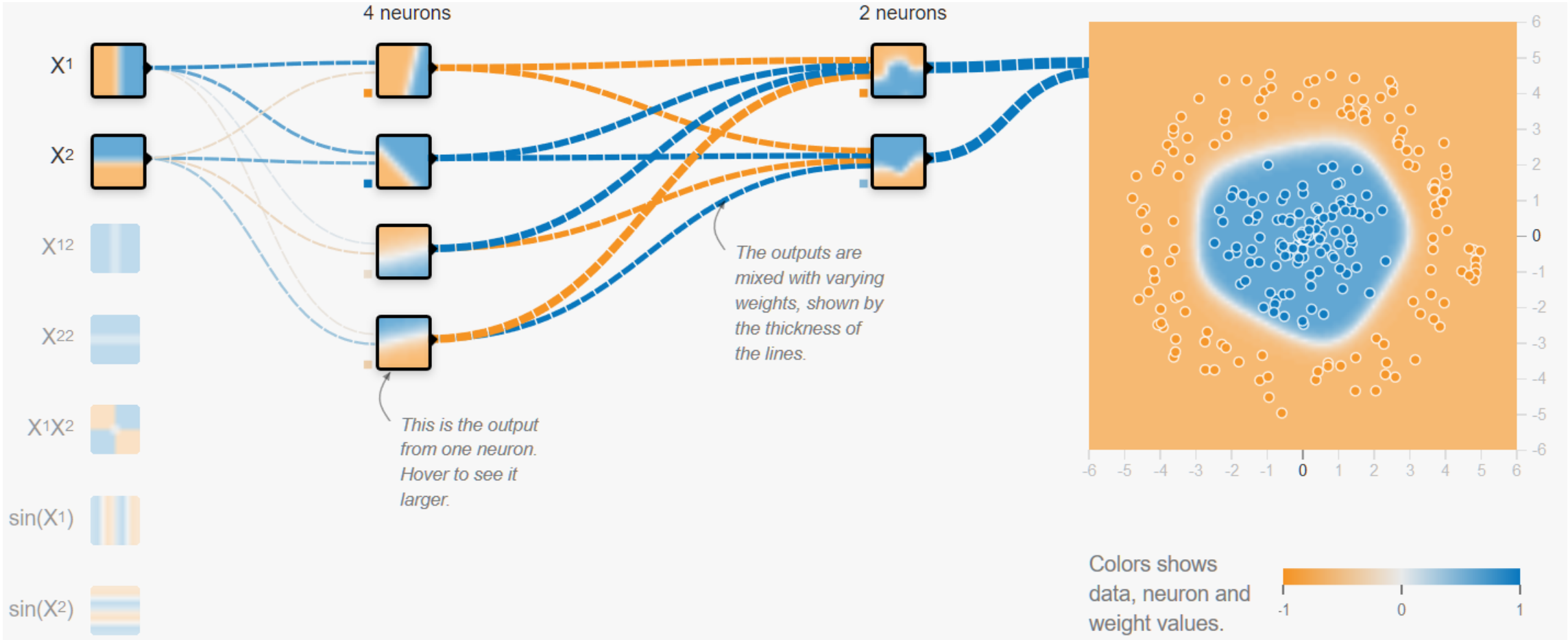
$$h_1 = f(W_1 x + b_1)$$
$$h_2 = f(W_2 h_1 + b_2)$$
$$s = W_3 h_2 + b_3$$

شبکه‌های عصبی: پیاده‌سازی انتشار پیش‌رو




```
f = lambda x: 1.0 / (1.0 + np.exp(-x))     # activation function (sigmoid)
x = np.random.randn(3, 1)                  # random input vector (3x1)

h1 = f(W1 @ x  + b1)                  # first  hidden layer activations (4x1)
h2 = f(W2 @ h1 + b2)                  # second hidden layer activations (4x1)
s  =   W3 @ h2 + b3                   # scores (1x1)
```

اجرای نمایشی



https://playground.tensorflow.org

توابع فعالیت

- شبکه عصبی سه لایه.

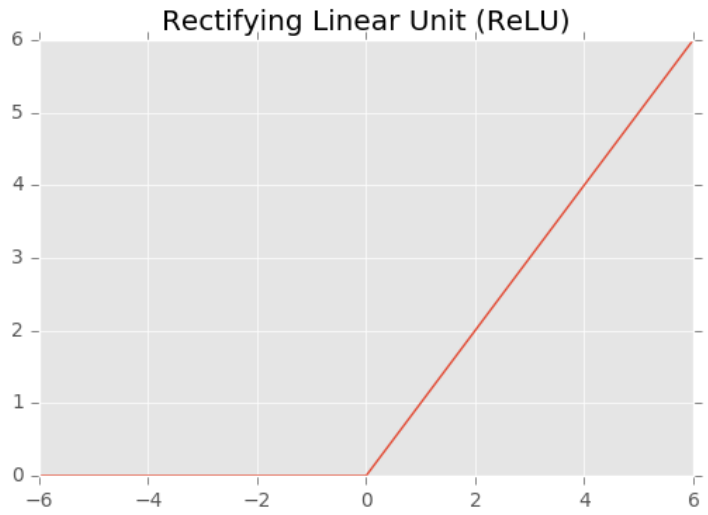
$$s = W_3 f\left(W_2 f(W_1 x)\right)$$

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

$$\begin{aligned} s &= W_3\left(W_2(W_1 x)\right) \\ &= (W_3 W_2 W_1)x \\ &= Wx \end{aligned}$$

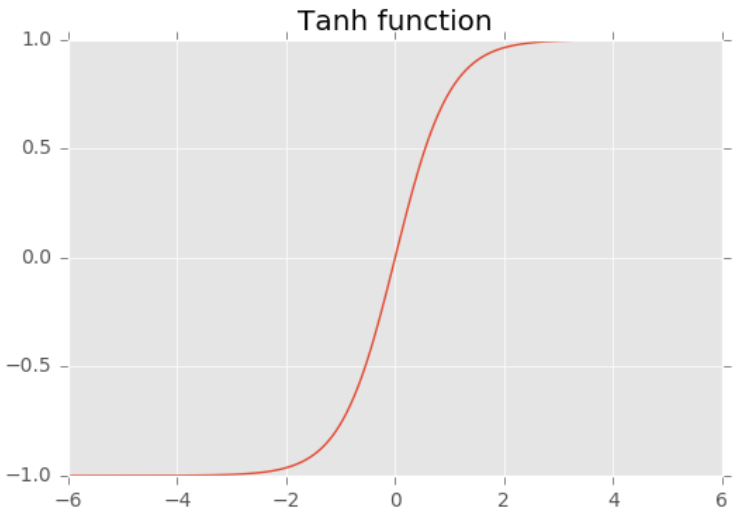
توابع فعالیت

ReLU



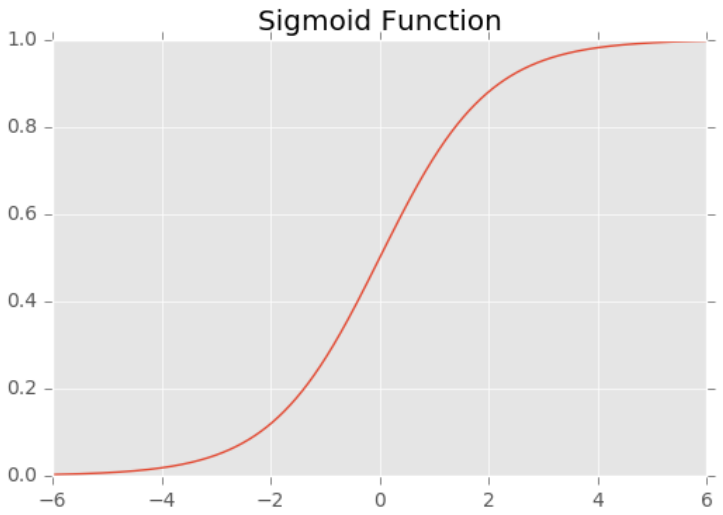
$$\max(0, x)$$

تانژانت هایپربولیک



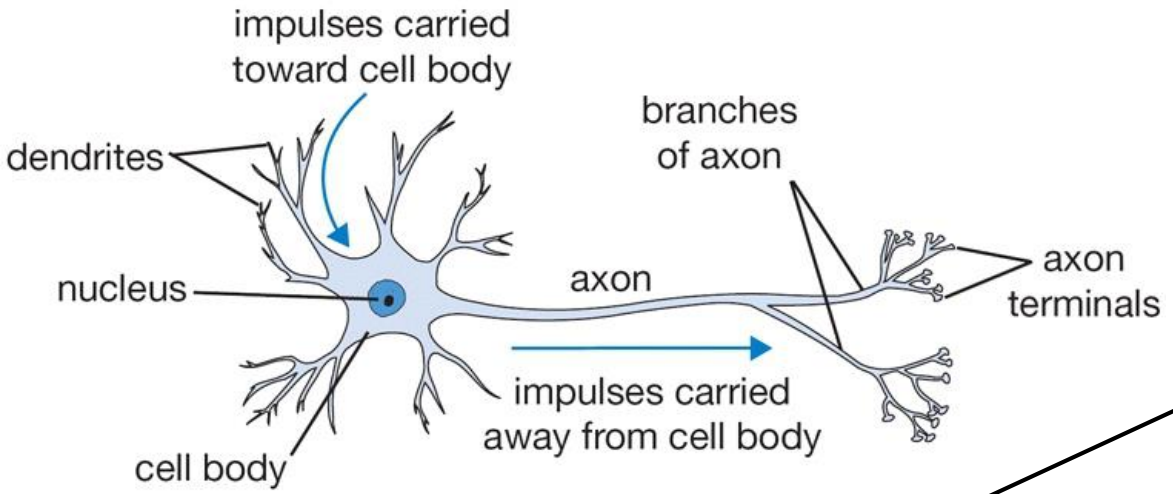
$$\tanh(x)$$

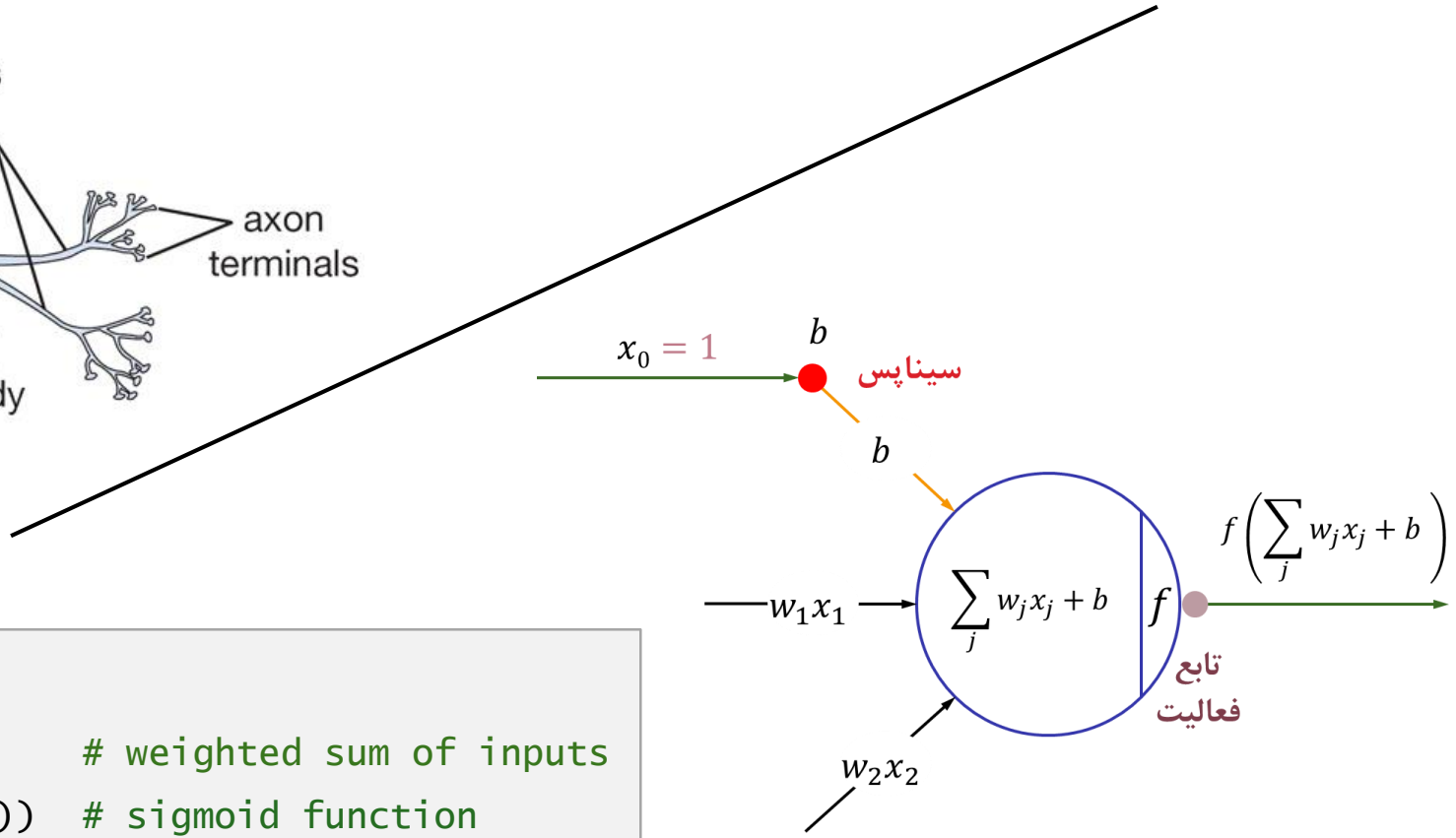
سیگموید



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

نورون‌ها و شبکه‌های عصبی

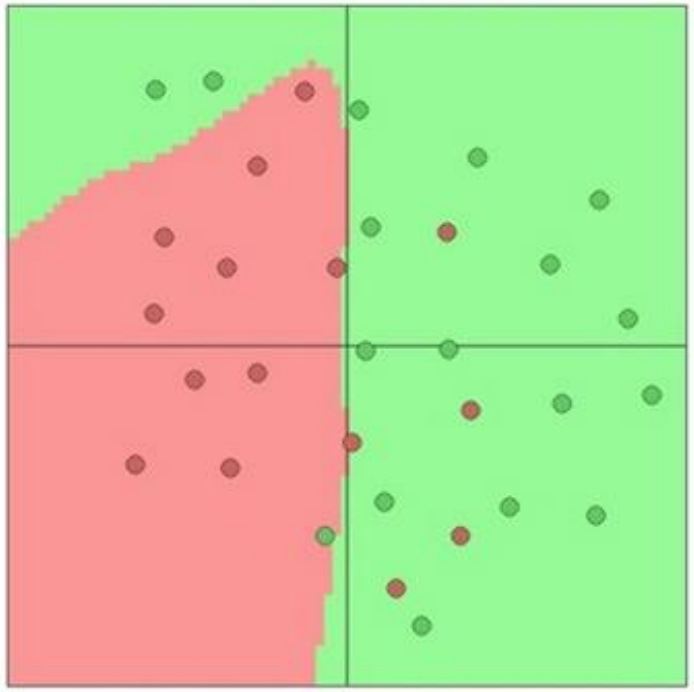




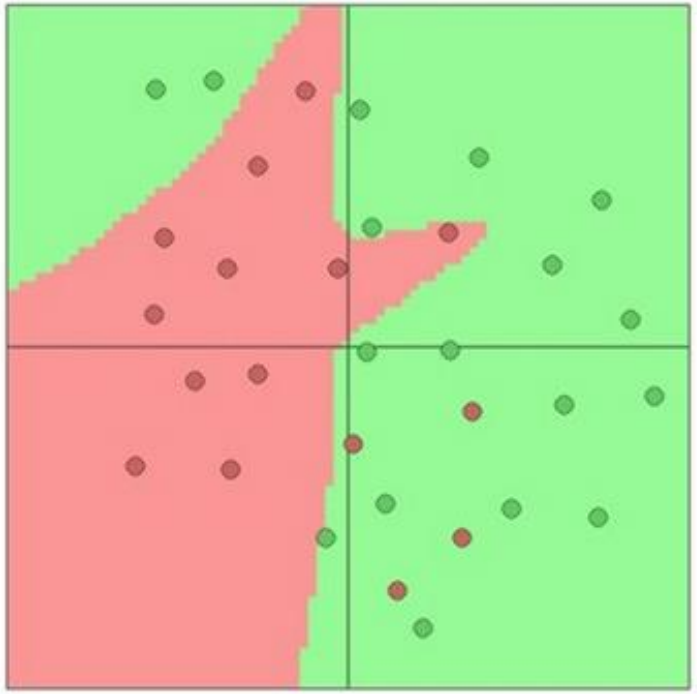

```
# assume w and x are 1d numpy arrays

net_input = np.sum(w * x) + b                # weighted sum of inputs
output = 1.0 / (1.0 + np.exp(-net_input))    # sigmoid function
```

تعیین تعداد و اندازه لایه‌ها

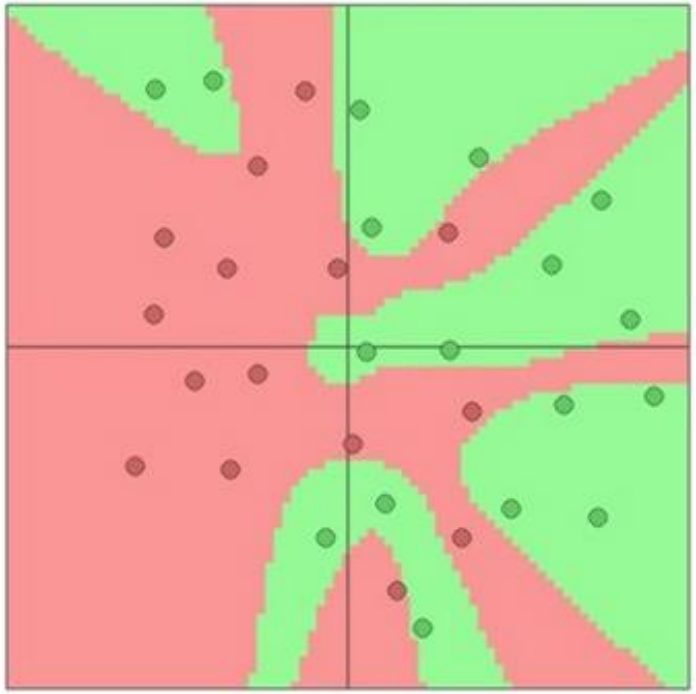
۳ نورون در لایه مخفی



۶ نورون در لایه مخفی

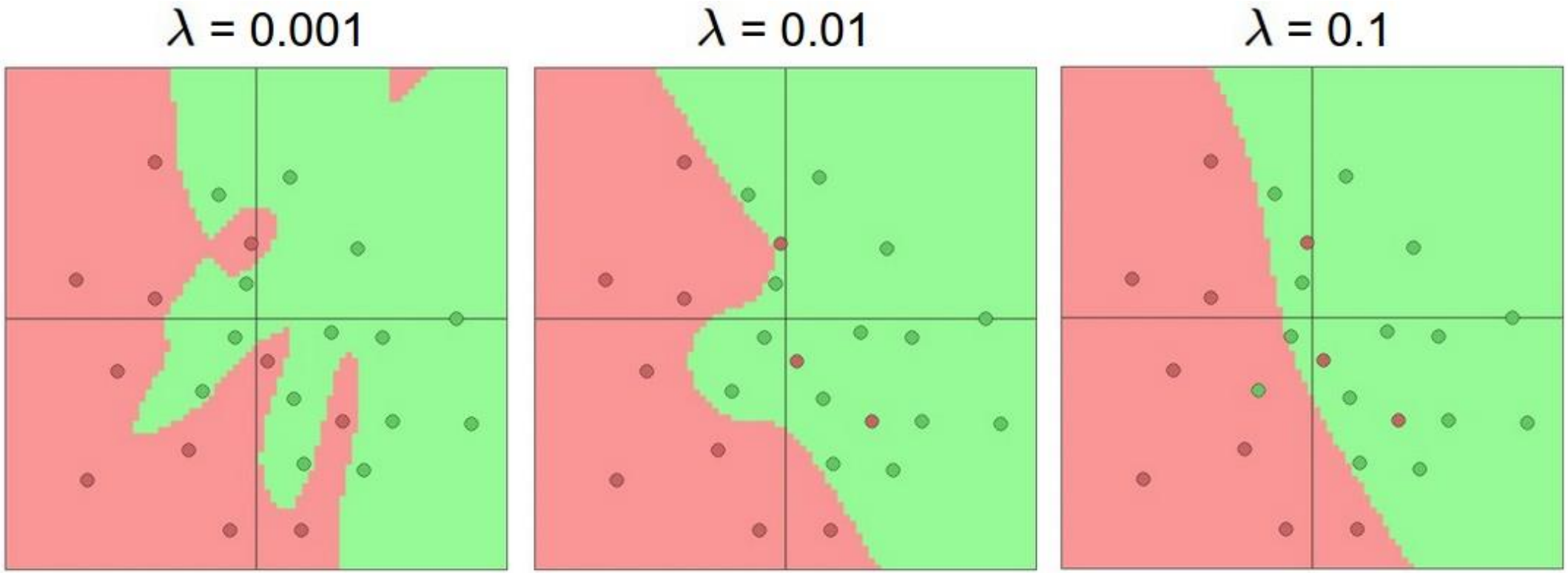


۲۰ نورون در لایه مخفی



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.

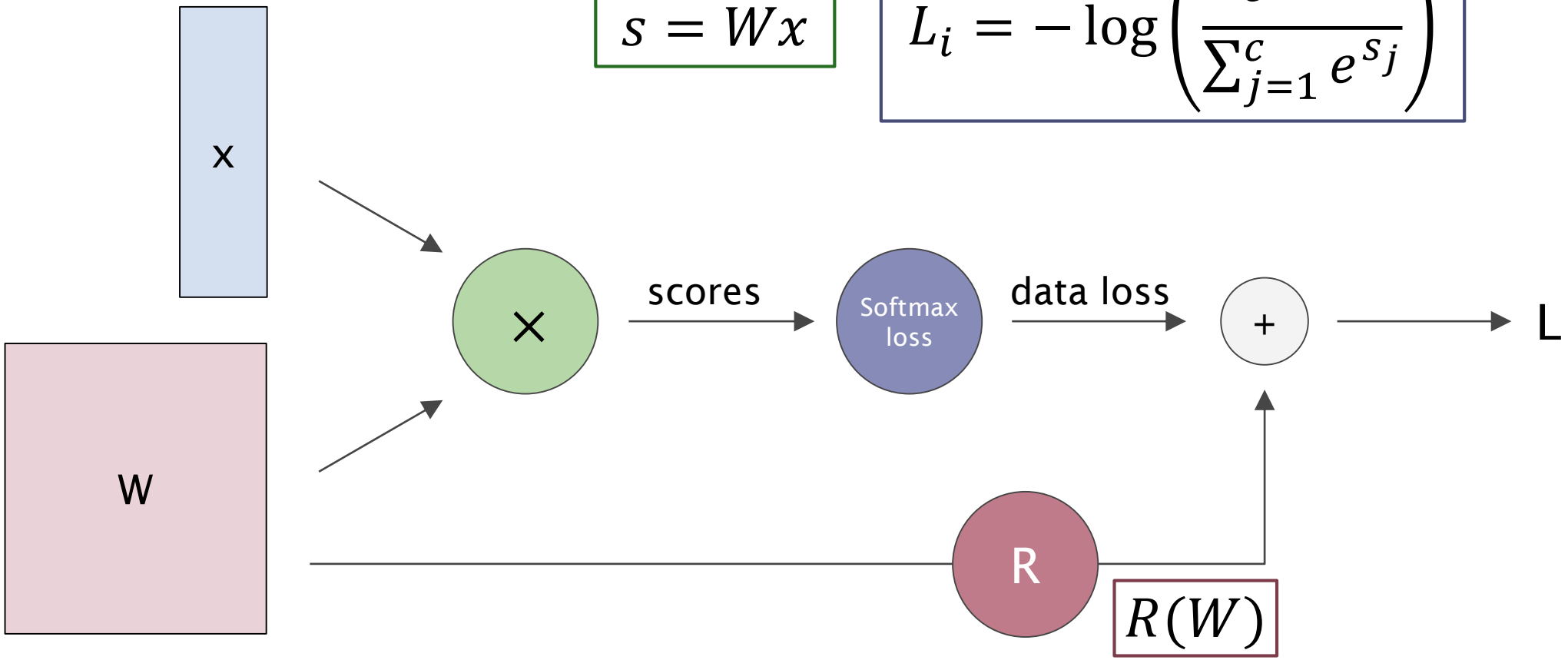
تنظیم L2

الگوریتم پس انتشار خطا

دسته‌بند سافت‌مکس: گراف محاسباتی

$$s = Wx$$

$$L_i = -\log\left(\frac{e^{s_{y(i)}}}{\sum_{j=1}^{c} e^{s_j}}\right)$$

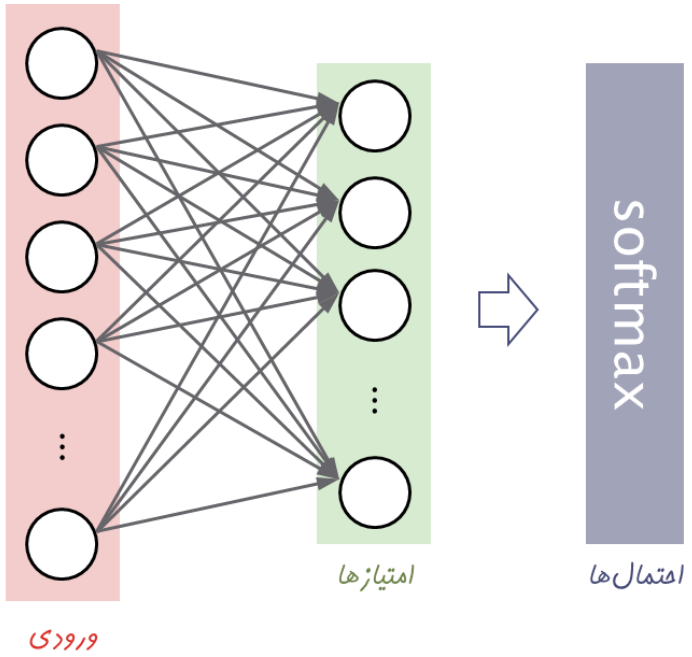


یادآوری: محاسبه گرادیان‌ها در دسته‌بند سافت‌مکس

- تابع هزینه.

$$L = \frac{1}{m}\left(\sum_{i=1}^{m} -\log\left(\frac{e^{s_{y^{(i)}}}}{\sum_{j=1}^{c} e^{s_j}}\right)\right) + \lambda \|W\|_2^2$$

- محاسبه گرادیان‌ها.

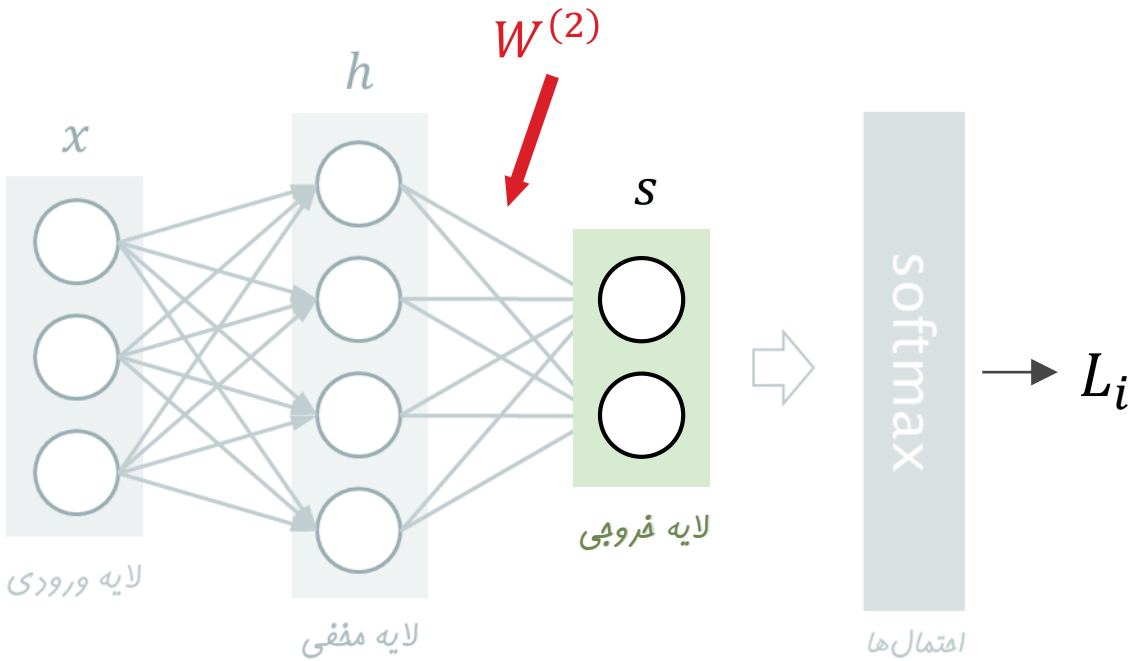


$$\frac{\partial L_i}{\partial w^{(k)}} = \frac{\partial L_i}{\partial s_k} \cdot \frac{\partial s_k}{\partial w^{(k)}}$$

$$= \left(p_k - \{y^{(i)} == k\}\right) x^{(i)}$$

$$\{y^{(i)} == k\} = \begin{cases} 1, & y^{(i)} = k \\ 0, & y^{(i)} \neq k \end{cases}$$

الگوریتم پس‌انتشار: محاسبه گرادیان‌ها



$$\frac{\partial L_i}{\partial W^{(2)}} = \frac{\partial L_i}{\partial s_k} \cdot \frac{\partial s_k}{\partial W^{(2)}}$$

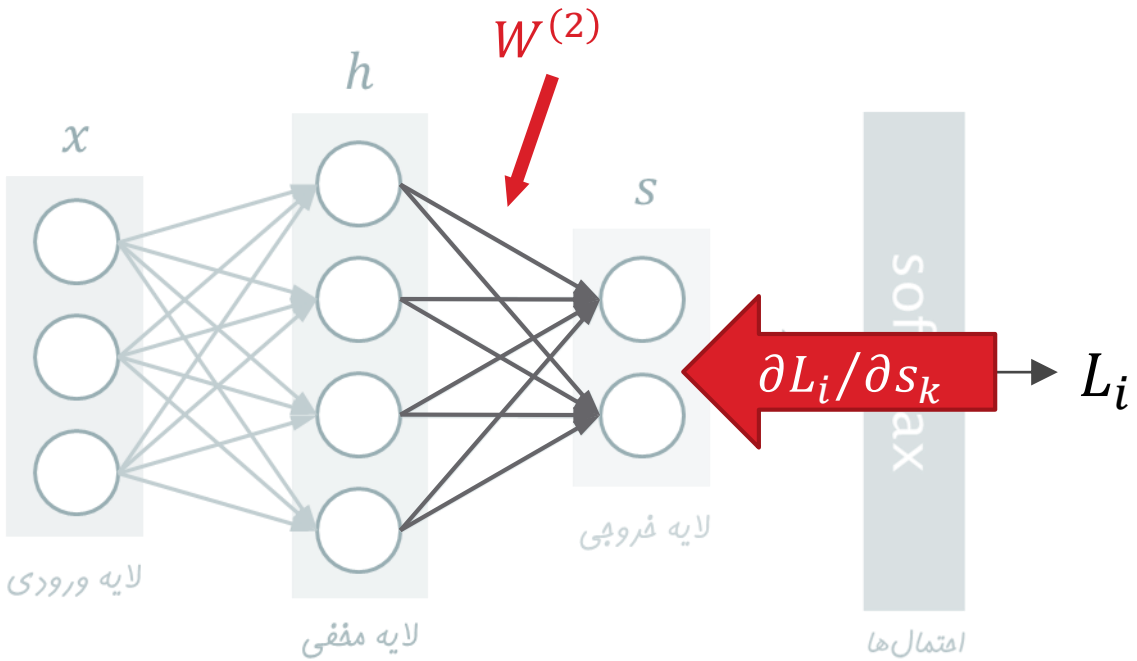
$$= \left(p_k - \{y^{(i)} == k\}\right)$$

$$L = \frac{1}{m}\left(\sum_{i=1}^{m} -\log\left(\frac{e^{s_{y^{(i)}}}}{\sum_{j=1}^{c} e^{s_j}}\right)\right) + \lambda \|W\|_2^2$$

الگوریتم پس‌انتشار: محاسبه گرادیان‌ها

$$\frac{\partial L_i}{\partial W^{(2)}} = \frac{\partial L_i}{\partial s_k} \cdot \frac{\partial s_k}{\partial W^{(2)}}$$

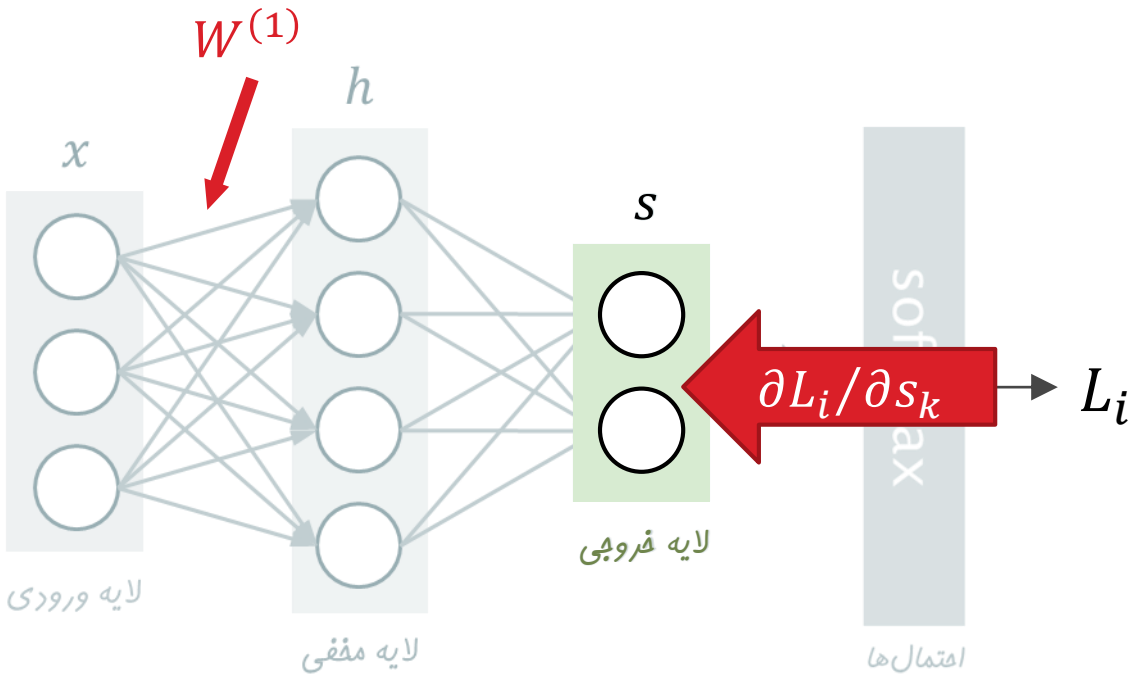
$$= \left(p_k - \{y^{(i)} == k\}\right) h$$



$$L = \frac{1}{m}\left(\sum_{i=1}^{m} -\log\left(\frac{e^{s_{y^{(i)}}}}{\sum_{j=1}^{c} e^{s_j}}\right)\right) + \lambda \|W\|_2^2$$

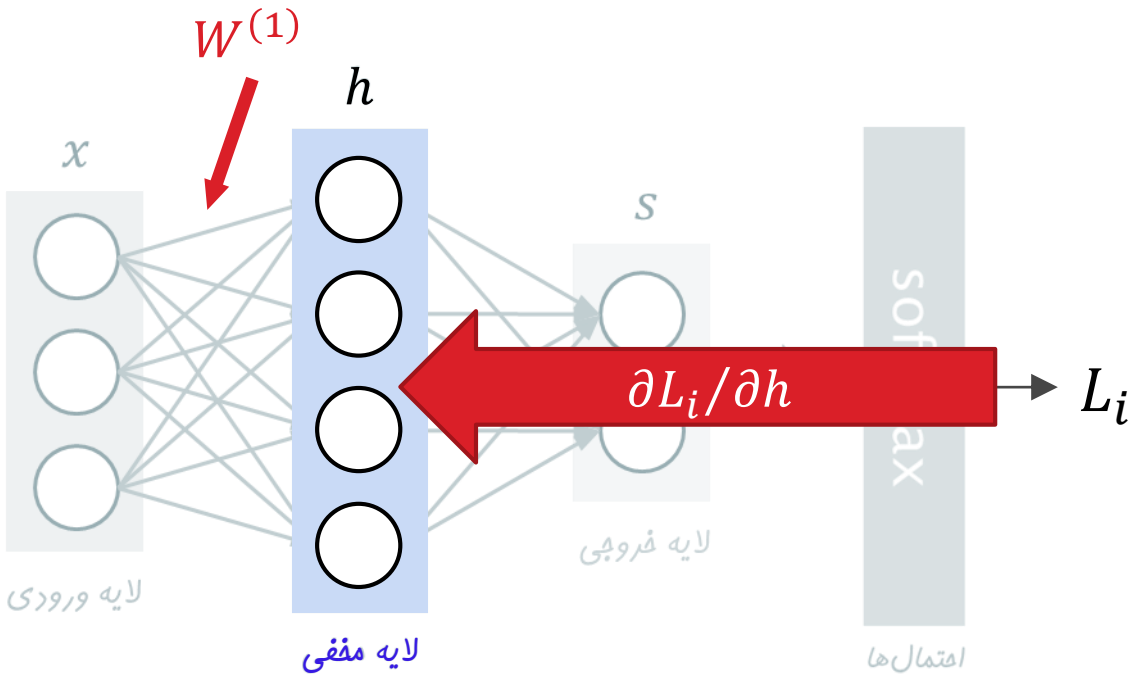
الگوریتم پس‌انتشار: محاسبه گرادیان‌ها

$$\frac{\partial L_i}{\partial W^{(1)}} = \frac{\partial L_i}{\partial s_k} \cdot \frac{\partial s_k}{\partial h} \cdot \frac{\partial h}{\partial f} \cdot \frac{\partial f}{\partial W^{(1)}}$$



$$L = \frac{1}{m}\left(\sum_{i=1}^{m} -\log\left(\frac{e^{s_{y^{(i)}}}}{\sum_{j=1}^{c} e^{s_j}}\right)\right) + \lambda \|W\|_2^2$$

الگوریتم پس‌انتشار: محاسبه گرادیان‌ها

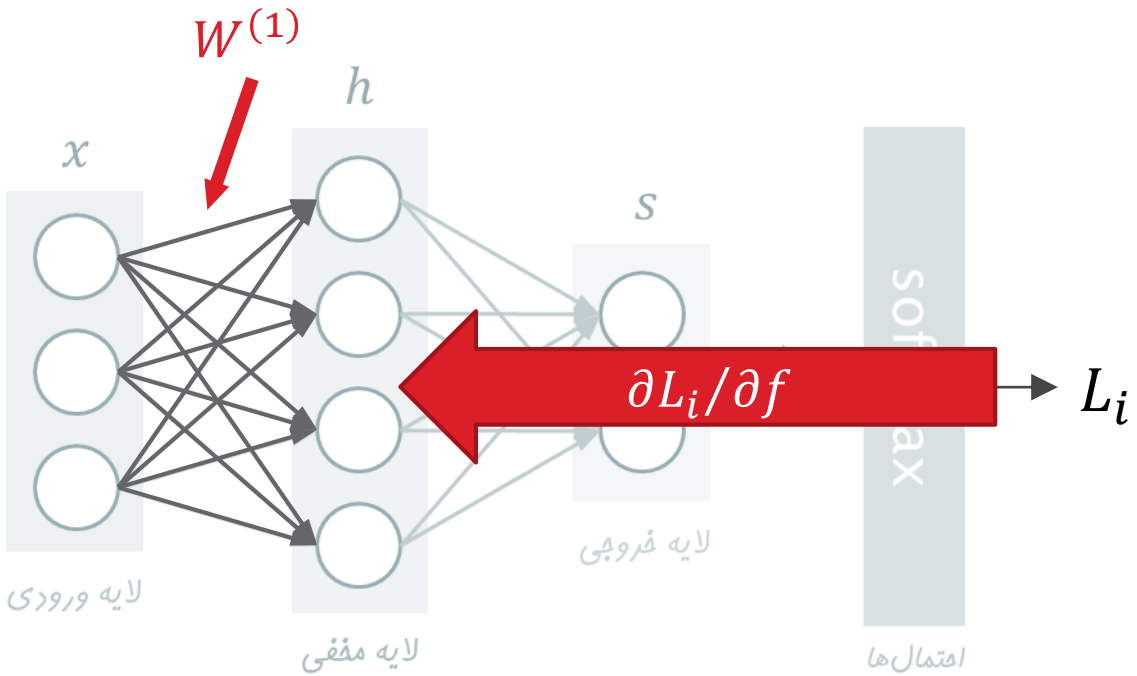


$$\frac{\partial L_i}{\partial W^{(1)}} = \underbrace{\frac{\partial L_i}{\partial s_k} \cdot \frac{\partial s_k}{\partial h}} \cdot \frac{\partial h}{\partial f} \cdot \frac{\partial f}{\partial W^{(1)}}$$

$$= \frac{\partial L_i}{\partial h} \cdot \frac{\partial h}{\partial f} \cdot \frac{\partial f}{\partial W^{(1)}}$$

$$L = \frac{1}{m}\left(\sum_{i=1}^{m} -\log\left(\frac{e^{s_{y^{(i)}}}}{\sum_{j=1}^{c} e^{s_j}}\right)\right) + \lambda \|W\|_2^2$$

الگوریتم پس‌انتشار: محاسبه گرادیان‌ها



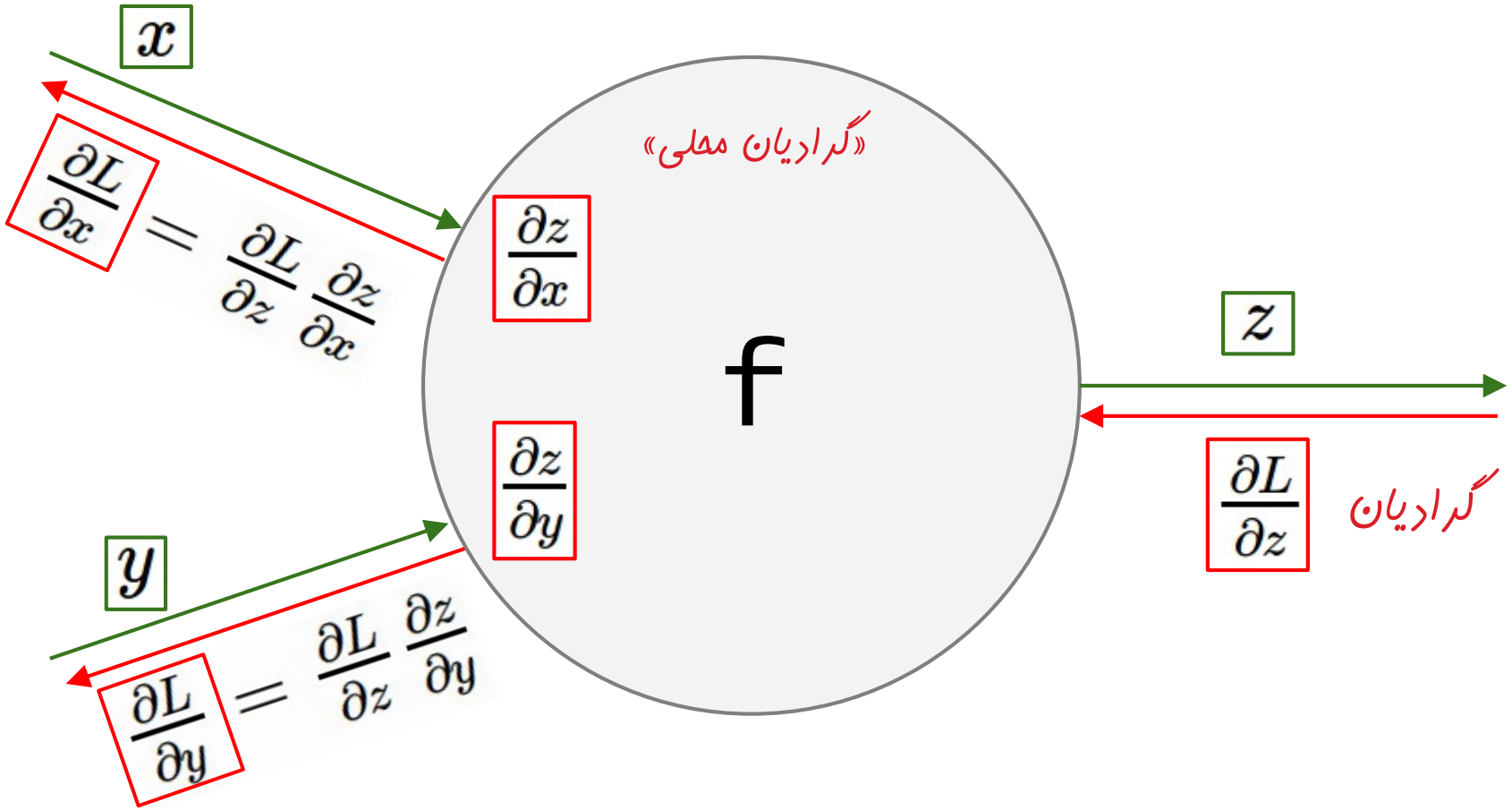
$$\frac{\partial L_i}{\partial W^{(1)}} = \underbrace{\frac{\partial L_i}{\partial s_k} \cdot \frac{\partial s_k}{\partial h} \cdot \frac{\partial h}{\partial f}} \cdot \frac{\partial f}{\partial W^{(1)}}$$

$$= \frac{\partial L_i}{\partial h} \cdot \frac{\partial h}{\partial f} \cdot \frac{\partial f}{\partial W^{(1)}}$$

$$= \frac{\partial L_i}{\partial f} \cdot \frac{\partial f}{\partial W^{(1)}}$$

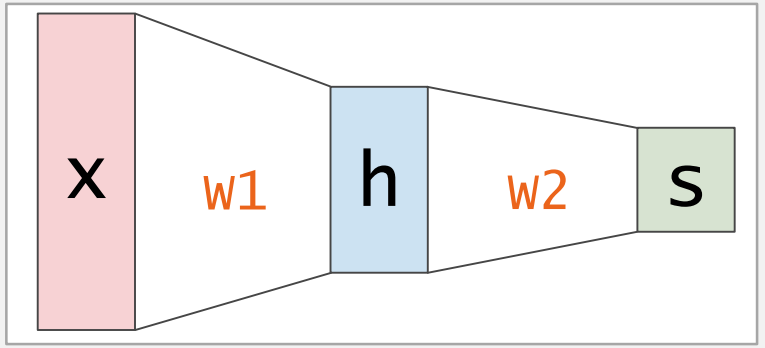
$$L = \frac{1}{m}\left(\sum_{i=1}^{m} -\log\left(\frac{e^{s_{y^{(i)}}}}{\sum_{j=1}^{c} e^{s_j}}\right)\right) + \lambda \|W\|_2^2$$

الگوریتم پس‌انتشار: محاسبه گرادیان‌ها



پیاده‌سازی یک شبکه عصبی دو 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 =           #... dL/dscores
dh, dW2, db2 =      #... dL/dh, dL/dW2, dl/db2
dW1, db1 =          #... dL/dW1, dL/db1
```



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.** محاسبه 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.

روش‌های بهینه‌سازی پیشرفته

بهینه‌سازی پیشرفته


```
from scipy.optimize import minimize

minimize(J, x0, args=(X_train, y_train), method='CG', jac=True)
```

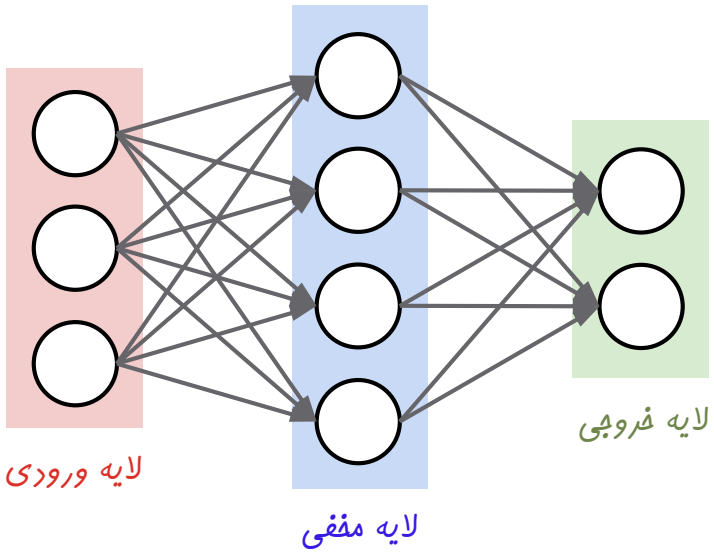
پارامترها و گرادیان‌ها باید به صورت یک بردار باشند

پیاده‌سازی تابع هزینه

$$L = \frac{1}{m}\left(\sum_{i=1}^{m} -\log\left(\frac{e^{s_{y(i)}}}{\sum_{j=1}^{c} e^{s_j}}\right)\right) + \lambda \|W\|_2^2$$

بهینه‌سازی پیشرفته

- ارسال پارامترها. پیش از ارسال پارامترها به تابع بهینه‌سازی باید همه آنها را به یک بردار تبدیل کنیم.



$F = 784$ $H = 20$ $C = 10$

$$W^{(1)} \in \mathbb{R}^{F \times H} \quad b^{(1)} \in \mathbb{R}^{H}$$

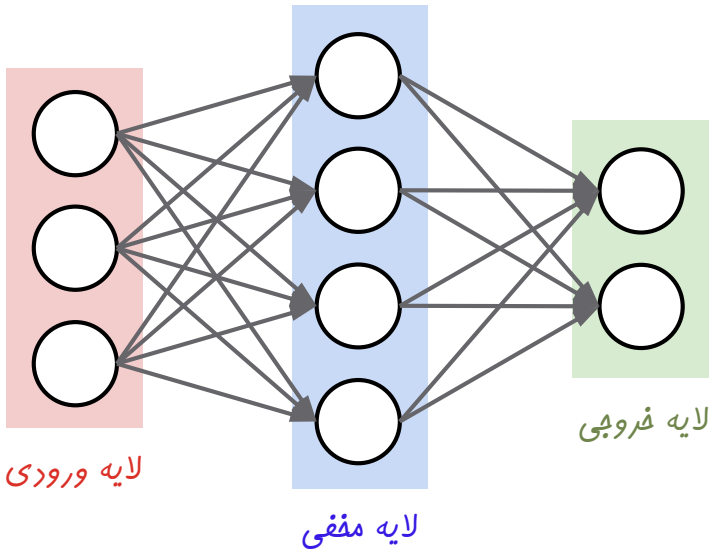
$$W^{(2)} \in \mathbb{R}^{H \times C} \quad b^{(2)} \in \mathbb{R}^{C}$$

$\Rightarrow W$

```
W = np.concatenate((W1.ravel(), b1, W2.ravel(), b2), axis=0)
```

بهینه‌سازی پیشرفته

- دریافت گرادیان‌ها. پیش از دریافت گرادیان‌ها، باید همه آنها را به یک بردار تبدیل کنیم.



$$F = 784 \qquad H = 20 \qquad C = 10$$

$$dW^{(1)} \in \mathbb{R}^{F \times H} \quad db^{(1)} \in \mathbb{R}^{H}$$

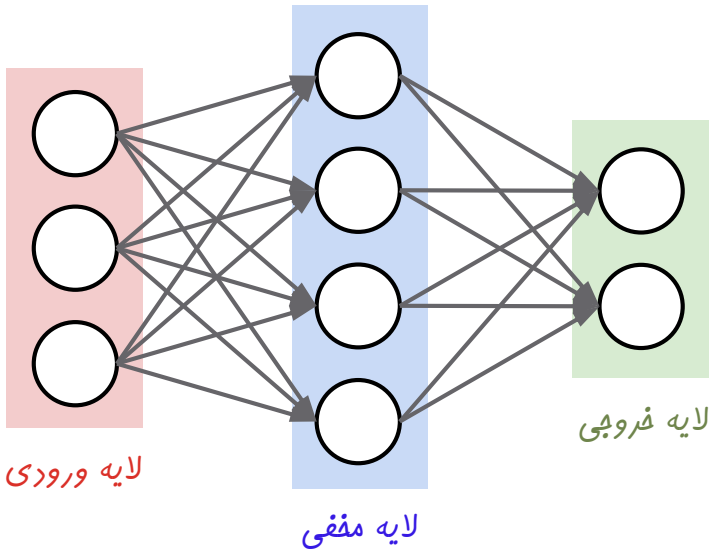
$$dW^{(2)} \in \mathbb{R}^{H \times C} \quad db^{(2)} \in \mathbb{R}^{C}$$

$\Rightarrow dW$

```
dW = np.concatenate((dW1.ravel(), db1, dW2.ravel(), db2), axis=0)
```

بهینه‌سازی پیشرفته

- دریافت پارامترها. پس از بهینه‌سازی، باید پارامترهای مختلف را از هم جدا کنیم.



$F = 784$ $H = 20$ $C = 10$

$$W^{(1)} \in \mathbb{R}^{F \times H} \qquad b^{(1)} \in \mathbb{R}^{H}$$

$$W^{(2)} \in \mathbb{R}^{H \times C} \qquad b^{(2)} \in \mathbb{R}^{C}$$

W

```
W1 = np.reshape(W[: F * H], (F, H))
b1 = W[F * H: (F + 1) * H]

W2 =  # ... get W2 and reshape it
b2 =  # ... get b2
```

بهینه‌سازی پیشرفته: مراحل

- ماتریس‌ها و بردارهای $W^{(i)}$ و $b^{(i)}$ را ایجاد و به صورت تصادفی مقداردهی کنید.

```
# create and init parameters W1, W2
W1 = np.random.randn(F, H) * 0.001
W2 = np.random.randn(H, C) * 0.001

b1 = np.zeros((H,))
b2 = np.zeros((C,))
```

- ماتریس‌ها و بردارهای $W^{(i)}$ و $b^{(i)}$ را به بردار W تبدیل کنید.

```
W = np.concatenate((W1.ravel(), b1, W2.ravel(), b2), axis=0)
```

بهینه‌سازی پیشرفته: مراحل

- تابع بهینه‌سازی را به صورت زیر فراخوانی نمایید:

```
result = minimize(J, x0=W, args=(X_train, y_train), method='CG', jac=True)
```

- پس از بهینه‌سازی، مقدار پارامترها را به صورت زیر ذخیره کنید:

```
W = result.x
```

- ماتریس‌ها و بردارهای $W^{(i)}$ و $b^{(i)}$ را از بردار W استخراج کنید.

```
W1 = np.reshape(W[: F * H], (F, H))
b1 = W[F * H: (F + 1) * H]

W2 =  # ... get W2 and reshape it
b2 =  # ... get b2
```

بررسی گرادیان

تخمین عددی گرادیان‌ها

- 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.

تخمین عددی گرادیان‌ها

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

تخمین عددی گرادیان‌ها

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

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

تخمین عددی گرادیان‌ها

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
?
?
?
?
?
?
?
?
...

تخمین عددی گرادیان‌ها

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

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

تخمین عددی گرادیان‌ها

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

تخمین عددی گرادیان‌ها

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

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

تخمین عددی گرادیان‌ها


```
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
```

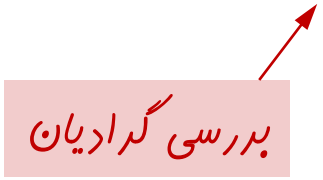
$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

- معایب.
  - تقریبی
  - بسیار زمان‌بر

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 درستی پیاده‌سازی، گرادیان تحلیلی را با گرادیان عددی مقایسه می‌کنیم.

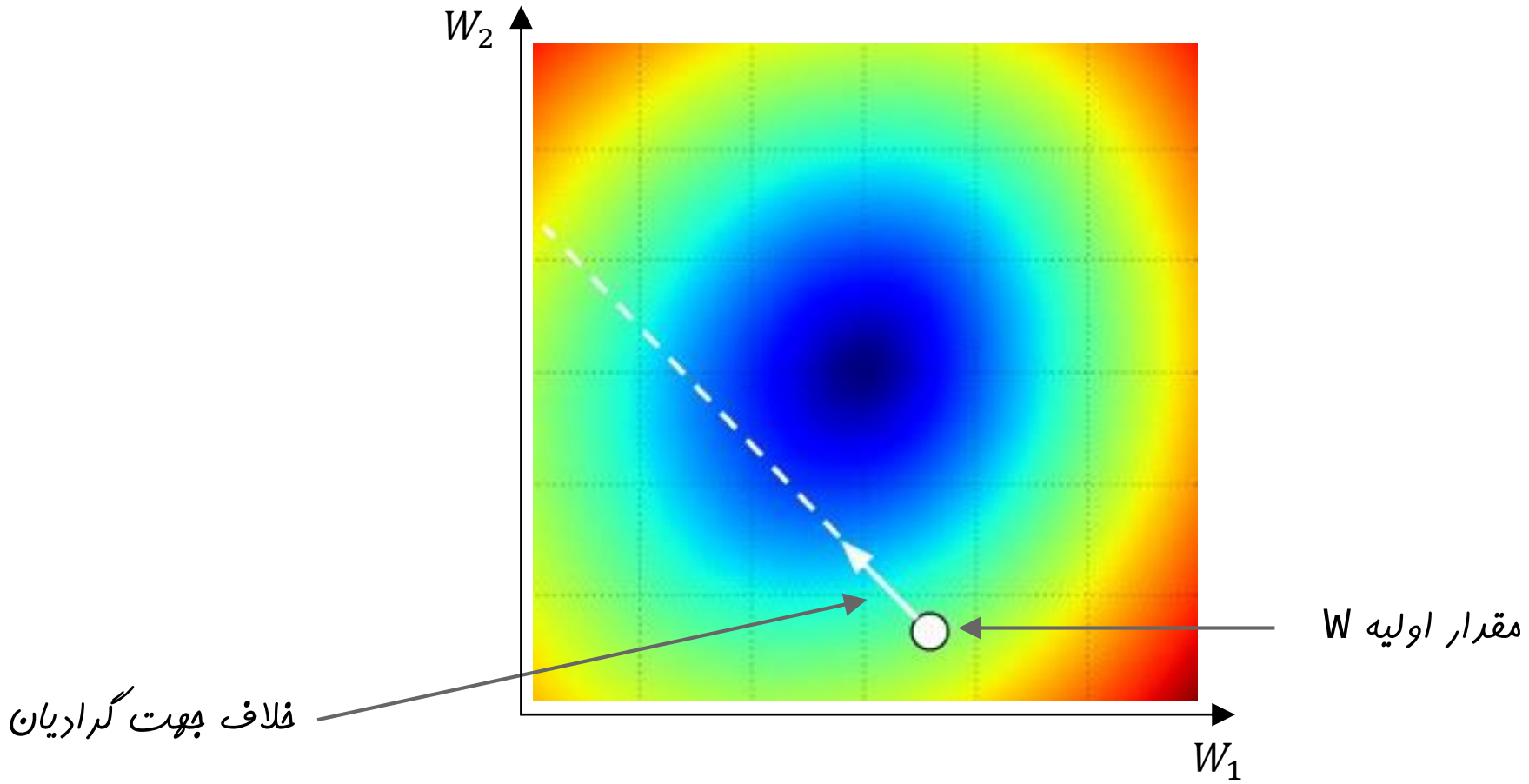


گرادیان کاهشی با دسته‌های کوچک

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

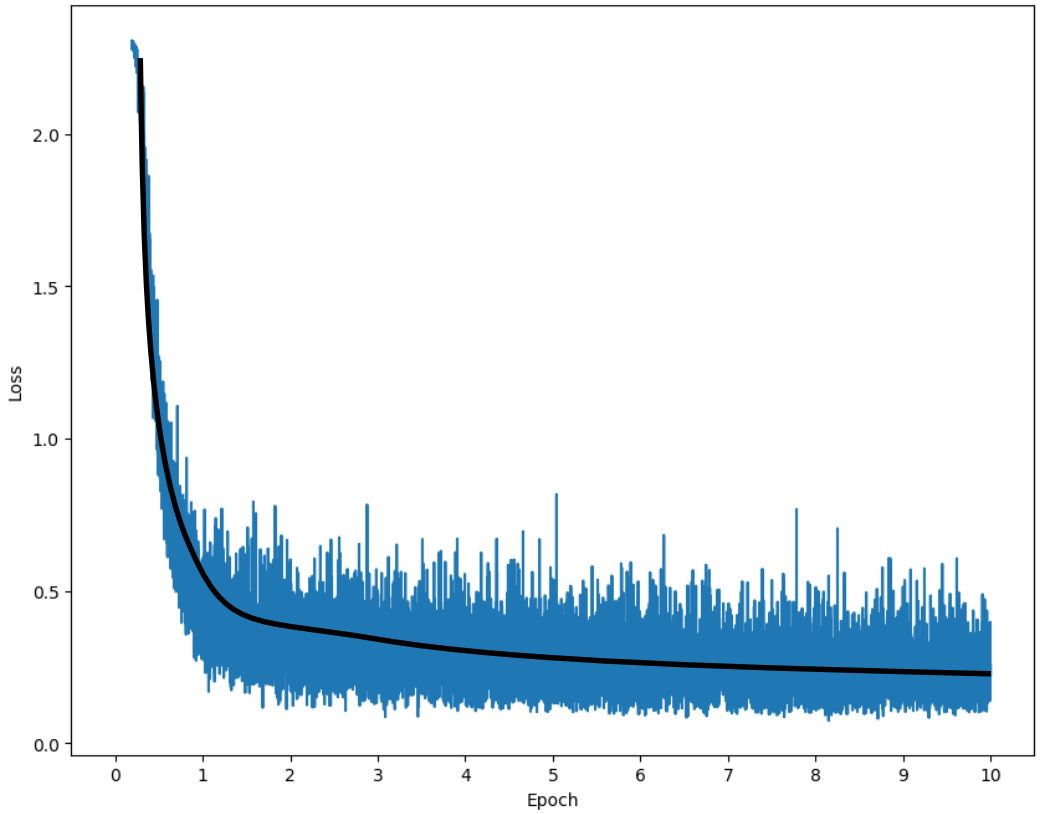
- گرادیان کاهشی با دسته‌های کوچک.
  - برای محاسبه گرادیان تابع هزینه در هر تکرار، تنها از بخش کوچکی از داده‌های آموزشی استفاده کن.

```
# 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 با دسته‌های کوچک

- نمونه اجرای الگوریتم گرادیان کاهشی با دسته‌های کوچک در یک شبکه عصبی.



هزینه در طول زمان کاهش می‌یابد.