Neural Network Training

Geena Kim

review: Learning in Perceptron

Learning in perceptron

- Perceptron rule
- Delta rule (Gradient Descent)

review: Learning in Perceptron

Perceptron algorithm

- Cycle through the training instances
- Only update W on misclassified instances
- If instance misclassified:
  - If instance is positive class (positive misclassified as negative)

$$W = W + X_i$$

  - If instance is negative class (negative misclassified as positive)

$$W = W - X_i$$

Perceptron

Delta rule (Gradient Descent)

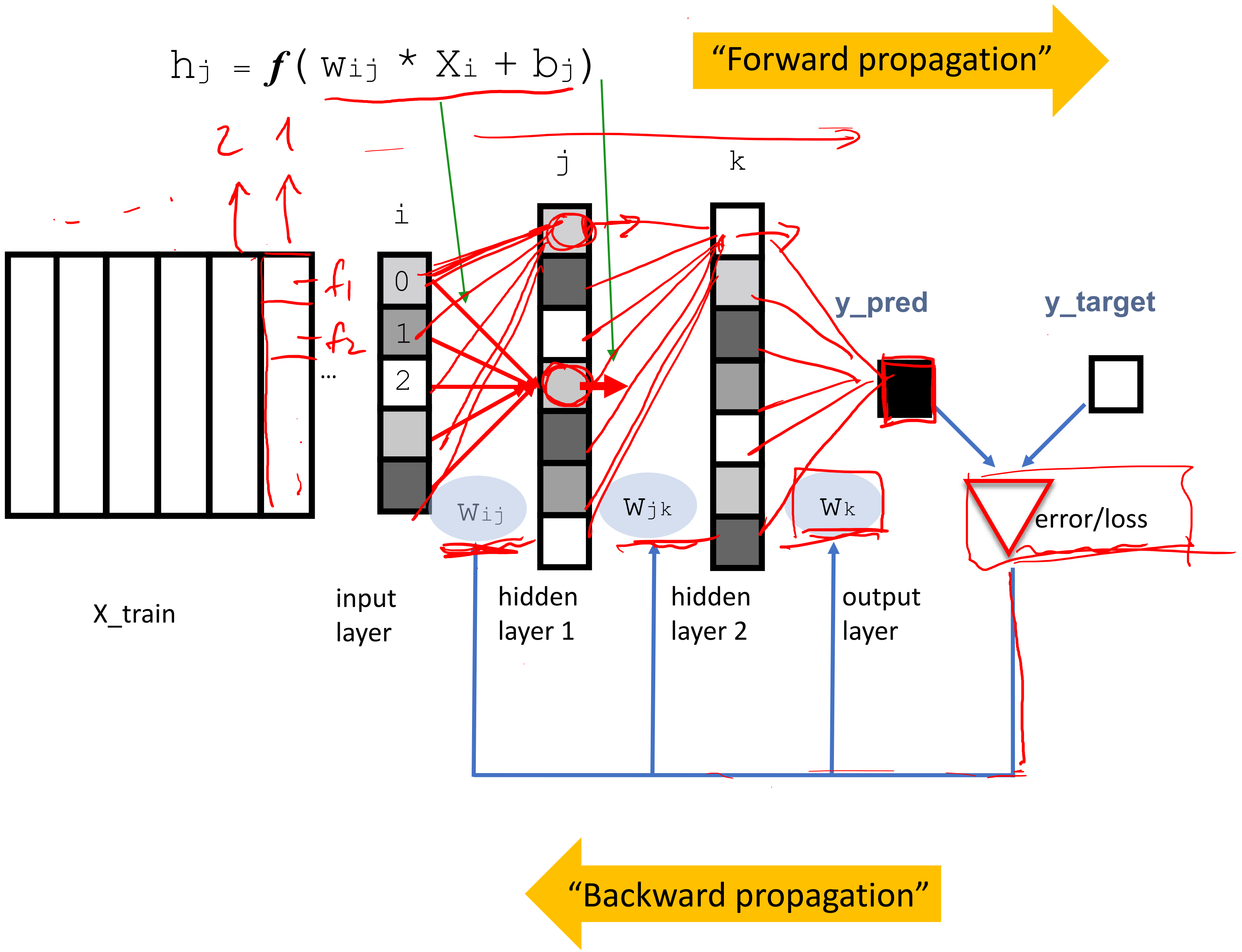
$$\omega_j \leftarrow \omega_j - \alpha \frac{\partial \mathcal{L}}{\partial \omega_j}$$

$$\mathcal{L} = \frac{1}{2}(\hat{y}_i - y_i)^2$$

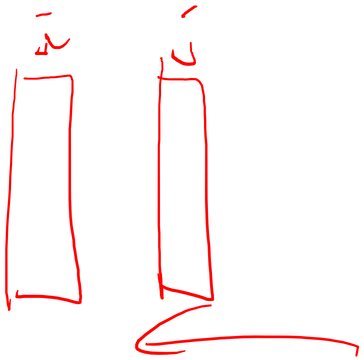
$$\hat{y}_i = \sum_j \omega_j X_{ij}$$

$$\omega_j \leftarrow \omega_j - \alpha(\hat{y}_i - y_i)X_{ij}$$

How Neural Network Training Works



Weight Update in deep neural nets



$$W_{ij} \leftarrow W_{ij} - \alpha \boxed{\frac{\partial \mathcal{L}}{\partial W_{ij}}}$$

Weight Update Rule

Loss Function

Gradient (Chain Rule)

Back Propagation

Chain Rule Reminder

g

$$\frac{\partial f(g(x))}{\partial x} = \underbrace{\frac{\partial f}{\partial g} \frac{\partial g}{\partial x}}$$

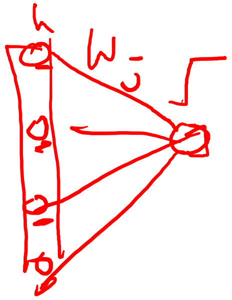
$f(g(h(\cdots\cdots)))$

$$\frac{\partial \sigma}{\partial z} = \underbrace{\frac{\partial}{\partial z}\left(\frac{1}{f}\right)}$$

$$f = \frac{1}{1+e^{-z}}$$

$$-\frac{1}{f^2} \, e^{-z} \cdot (-1)$$

Example: gradient of sigmoid



$$\frac{\partial \sigma}{\partial w_j}$$

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

$$z = \sum_{j=0} w_j \cdot \underbrace{x_j}$$

$$\frac{\partial \sigma}{\partial w_j} = \underbrace{\boxed{\frac{\partial \sigma}{\partial z}}}_{\sigma \cdot (1-\sigma)} \cdot \boxed{\frac{\partial z}{\partial w_j}}$$

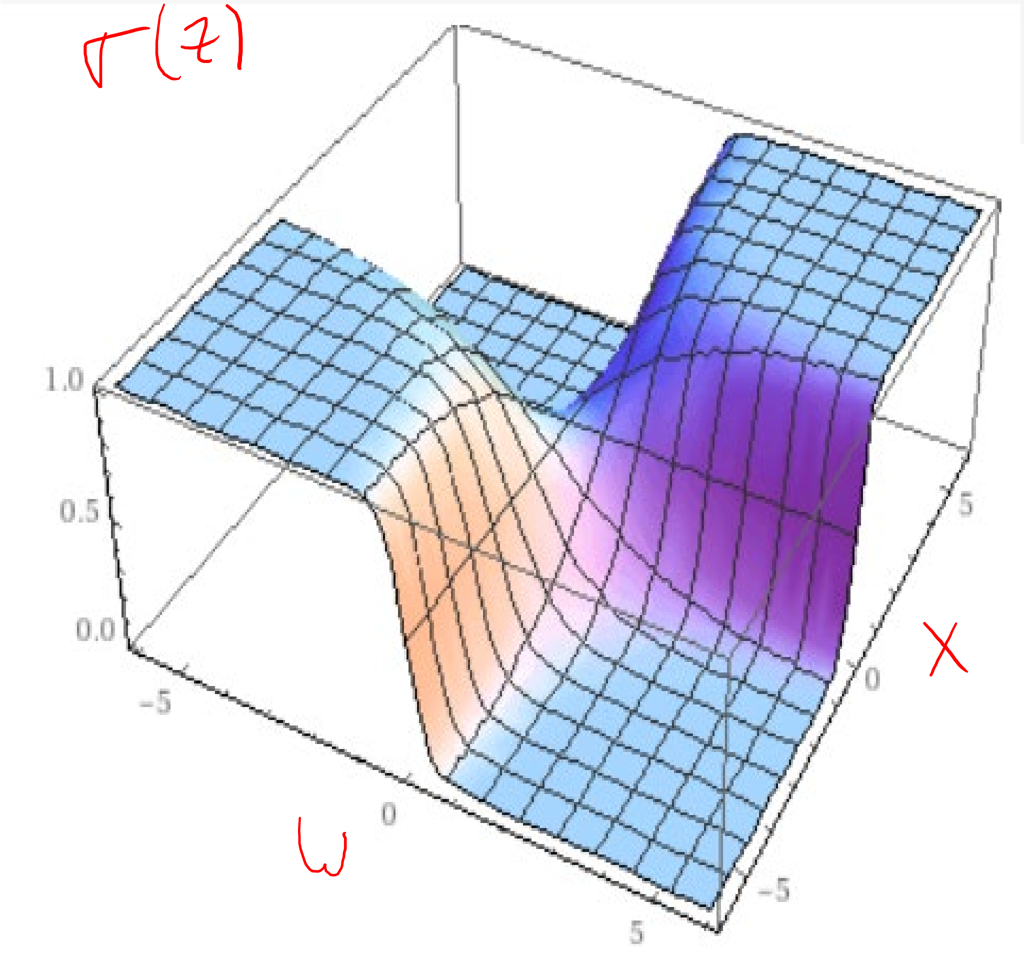
$$z = \sum_j w_j \cdot x_j$$

x_j

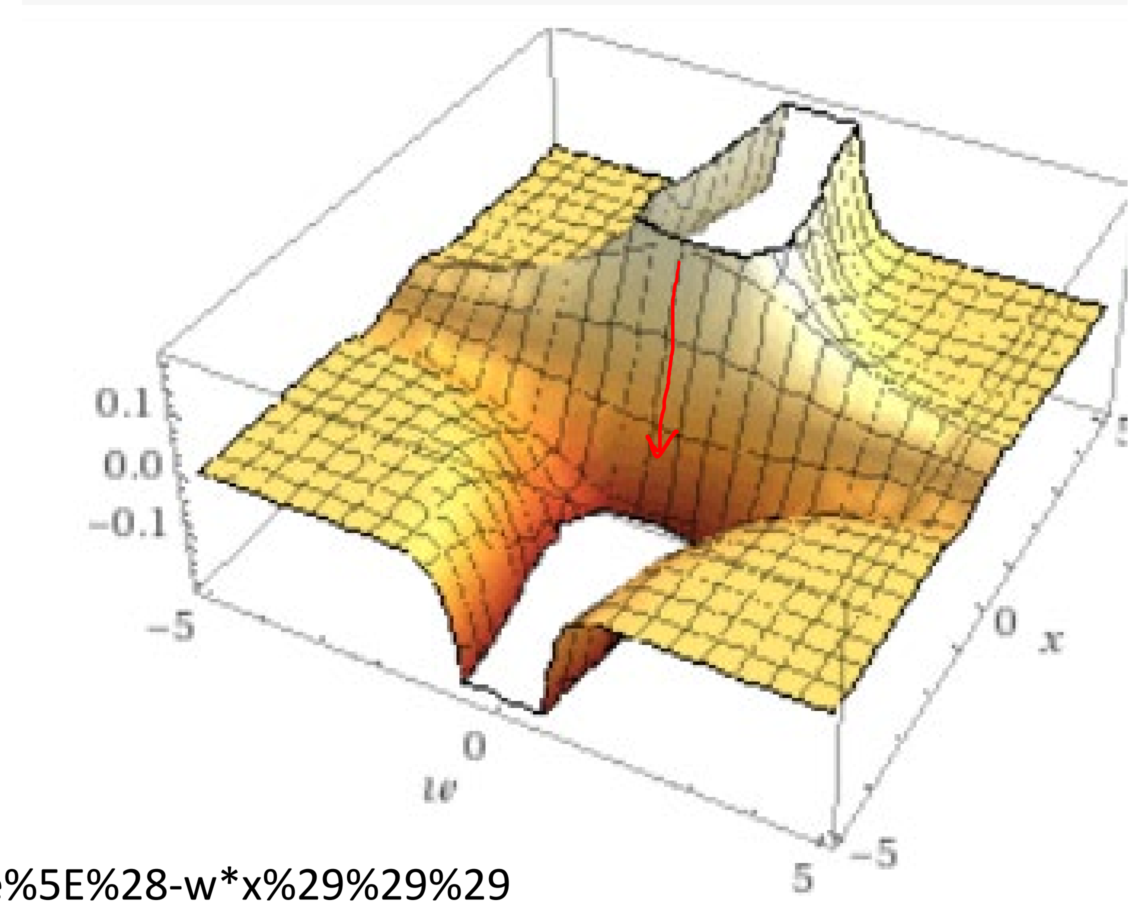
$$\frac{\partial \sigma}{\partial w_j} = \sigma \cdot (1-\sigma) \cdot x_j$$

Chain Rule Reminder

$$\frac{1}{1+e^{-w\,x}}$$



$$\frac{\partial}{\partial w}\left(\frac{1}{1+e^{-w\,x}}\right)=\frac{x\,e^{-w\,x}}{(e^{-w\,x}+1)^2}$$



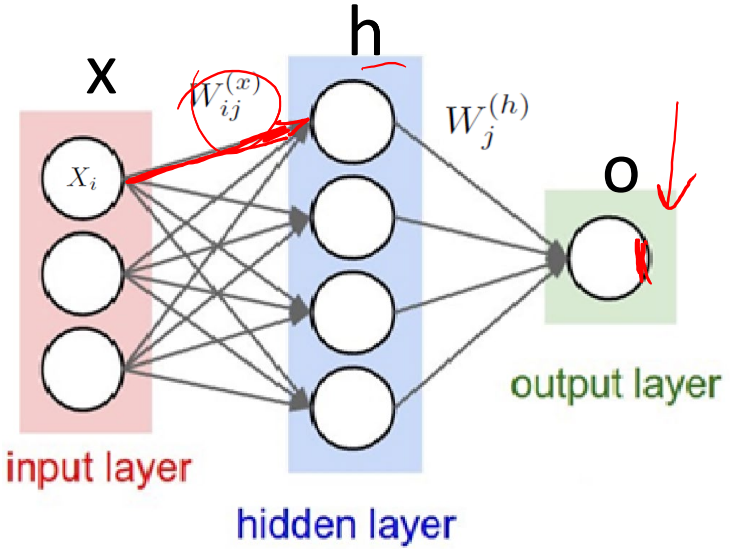
https://www.wolframalpha.com/input/?i=d%2Fdw+%281%2F%281%2Be%5E%28-w*x%29%29%29

Calculating Gradient- Chain Rule

$$\mathcal{L} = -\frac{1}{m}\sum_{s}^{m} y_s \log a_s + (1 - y_s)\log(1 - a_s)$$

$$W_{ij} \leftarrow W_{ij} - \alpha \frac{\partial \mathcal{L}}{\partial W_{ij}}$$

$$\frac{\partial f(g(x))}{\partial x} = \frac{\partial f}{\partial g}\frac{\partial g}{\partial x}$$



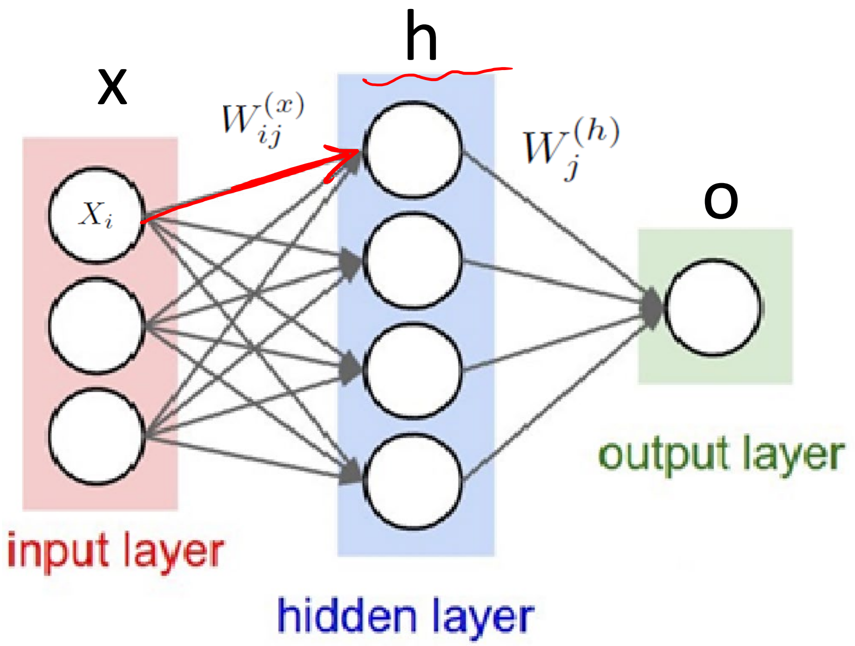
$$\frac{\partial \mathcal{L}}{\partial W_{ij}^{(x)}} = \boxed{\frac{\partial \mathcal{L}(a^{(o)})}{\partial a^{(o)}}}\boxed{\frac{\partial a^{(o)}}{\partial W_{ij}^{(x)}}}$$

$$= \left(\frac{y}{a^{(o)}} - \frac{1-y}{1-a^{(o)}}\right)\boxed{\frac{\partial a^{(o)}}{\partial W_{ij}^{(x)}}}$$

$$\frac{\partial a^{(o)}}{\partial W_{ij}^{(x)}} = \frac{\partial \sigma(z^{(o)})}{\partial z^{(o)}}\frac{\partial z^{(h)}}{\partial W_{ij}^{(x)}}$$

$$= \frac{\partial \sigma(z^{(o)})}{\partial z^{(o)}}\frac{\partial z^{(o)}}{\partial W_{ij}^{(x)}} = \sigma(z^{(o)})(1 - \sigma(z^{(o)}))\frac{\partial z^{(o)}}{\partial W_{ij}^{(x)}}$$

Calculating Gradient- Chain Rule



$$\frac{\partial z^{(o)}}{\partial W_{ij}^{(x)}} = \sum_j^{n^{(h)}} W_j^{(h)} \frac{\partial a_j^{(h)}}{\partial W_{ij}^{(x)}}$$

$$= \sum_j^{n^{(h)}} W_j^{(h)} \frac{\partial \sigma(z_j^{(h)})}{\partial W_{ij}^{(x)}} = \sum_j^{n^{(h)}} W_j^{(h)} \sigma(z_j^{(h)})(1-\sigma(z_j^{(h)})) \frac{\partial (z_j^{(h)})}{\partial W_{ij}^{(x)}} = \sum_j^{n^{(h)}} W_j^{(h)} \sigma(z_j^{(h)})(1-\sigma(z_j^{(h)}))X_i$$

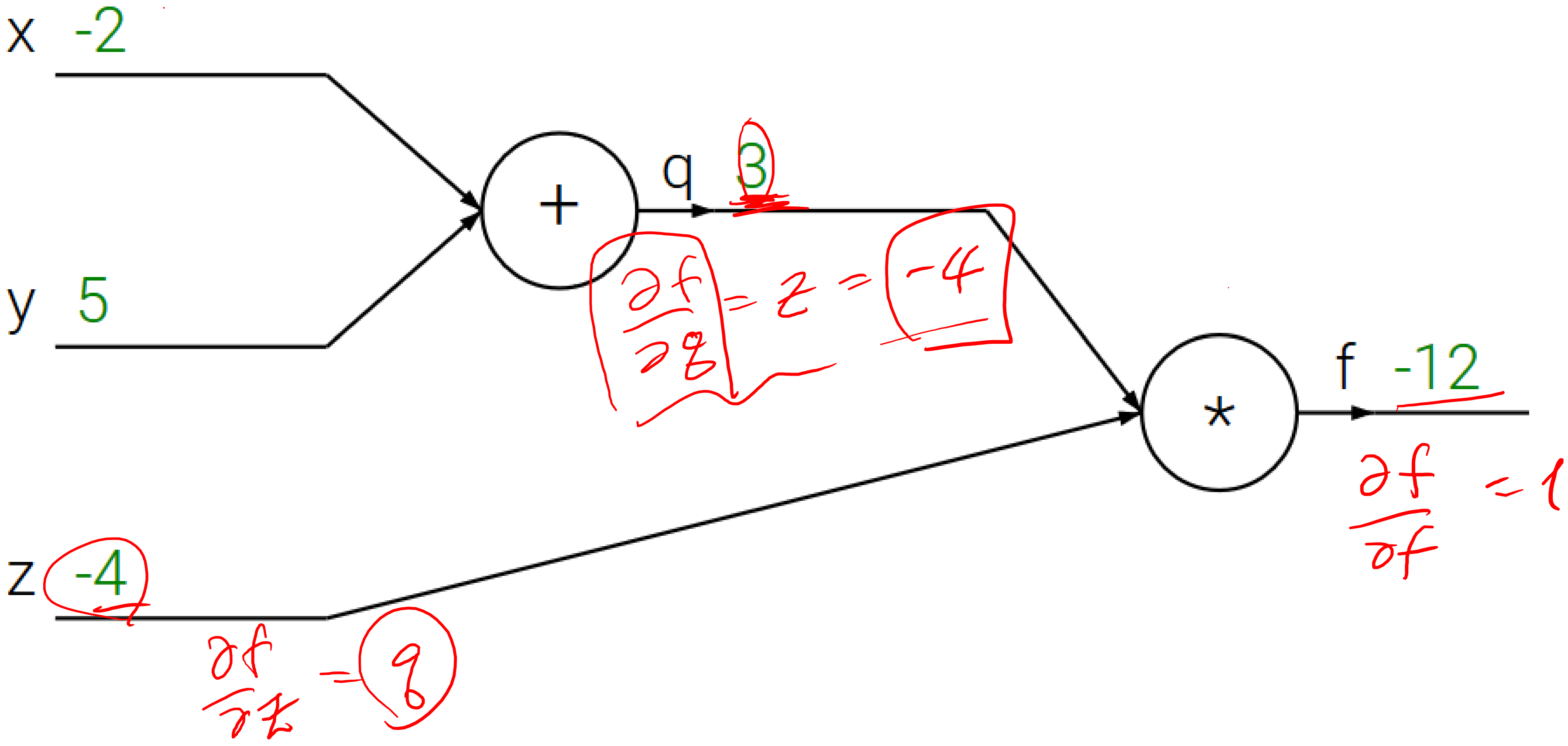
$$\frac{\partial \mathcal{L}}{\partial W_{ij}^{(x)}} = \left(\frac{y}{a^{(o)}} - \frac{1-y}{1-a^{(o)}}\right) \sigma(z^{(o)})(1-\sigma(z^{(o)})) \sum_j^{n^{(h)}} W_j^{(h)} \sigma(z_j^{(h)})(1-\sigma(z_j^{(h)}))X_i$$

Back Propagation- Computation Graph

$\boxed{q = x+y}$

$f = q*z$

forward pass →



Back Propagation- Computation Graph

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$

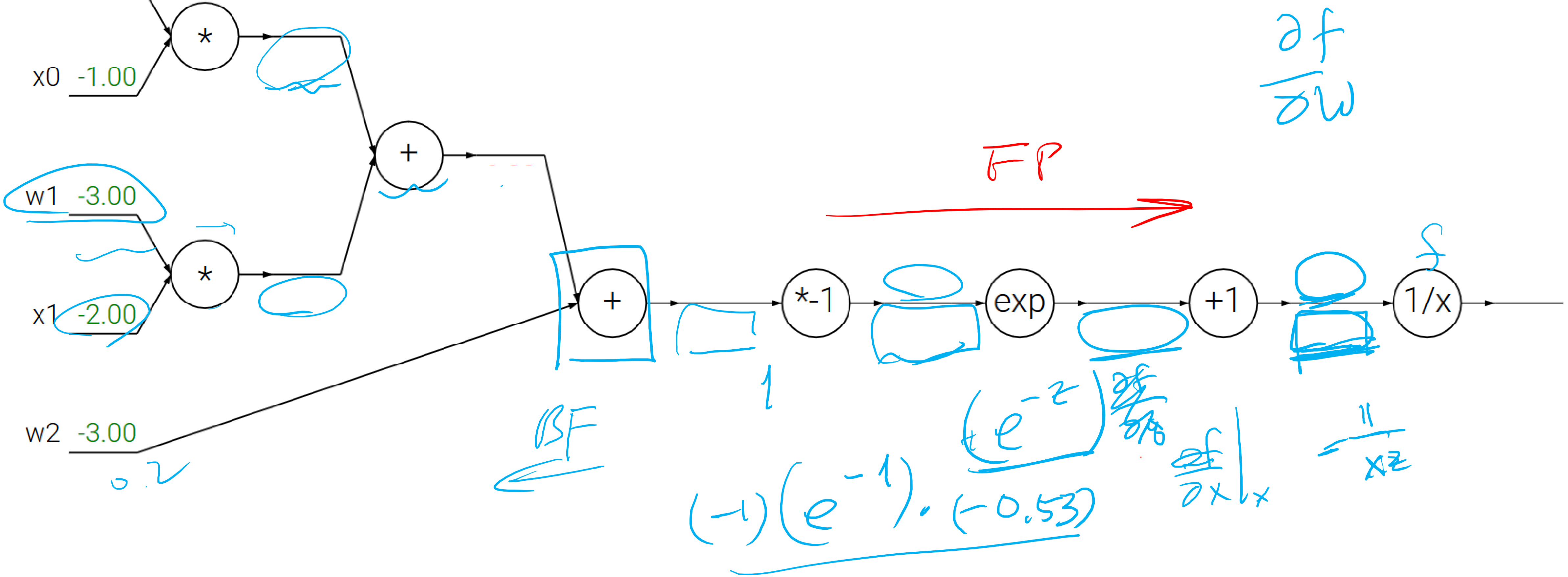


Image borrowed from Stanford cs231n

Back Propagation- Computation Graph

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$

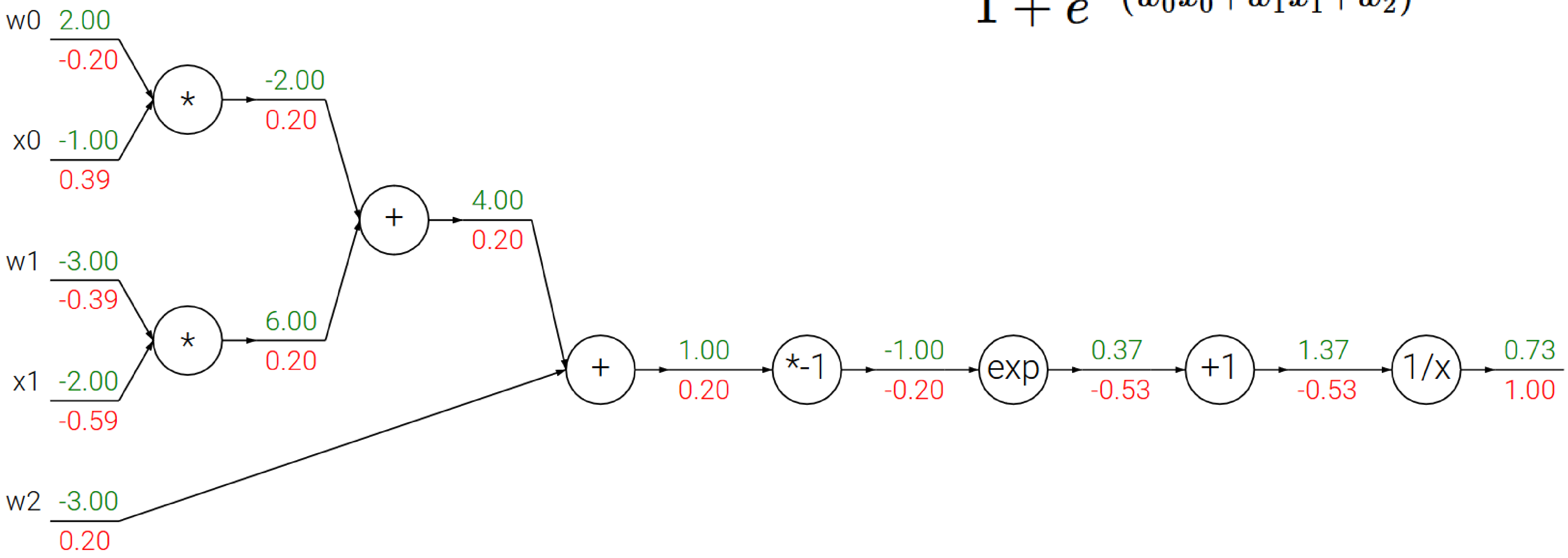


Image borrowed from Stanford cs231n

How does the computer perform differentiation?

Automatic Differentiation (Autodiff)

```
defvjp(anp.add,         lambda g, ans, x, y : unbroadcast(x, g),
                        lambda g, ans, x, y : unbroadcast(y, g))
defvjp(anp.multiply,    lambda g, ans, x, y : unbroadcast(x, y * g),
                        lambda g, ans, x, y : unbroadcast(y, x * g))
defvjp(anp.subtract,    lambda g, ans, x, y : unbroadcast(x, g),
                        lambda g, ans, x, y : unbroadcast(y, -g))
defvjp(anp.divide,      lambda g, ans, x, y : unbroadcast(x,   g / y),
                        lambda g, ans, x, y : unbroadcast(y, - g * x / y**2))
defvjp(anp.true_divide, lambda g, ans, x, y : unbroadcast(x,   g / y),
                        lambda g, ans, x, y : unbroadcast(y, - g * x / y**2))
defvjp(anp.power,
    lambda g, ans, x, y: unbroadcast(x, g * y * x ** anp.where(y, y - 1, 1.)),
    lambda g, ans, x, y: unbroadcast(y, g * anp.log(replace_zero(x, 1.)) * x ** y))
```

$f = \frac{1}{y}$ $\frac{\partial f}{\partial y}$ $-\frac{1}{y^2} \cdot g$

exp

log

https://github.com/mattjj/autodidact/blob/master/autograd/numpy/numpy_vjps.py

https://www.cs.toronto.edu/~rgrosse/courses/csc321_2018/slides/lec10.pdf