

CSPB 3202 Artificial Intelligence

# 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)

$$W = W + \underline{X_i}$$

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

$$W = W - \underline{X_i}$$

# Perceptron

## Delta rule (Gradient Descent)

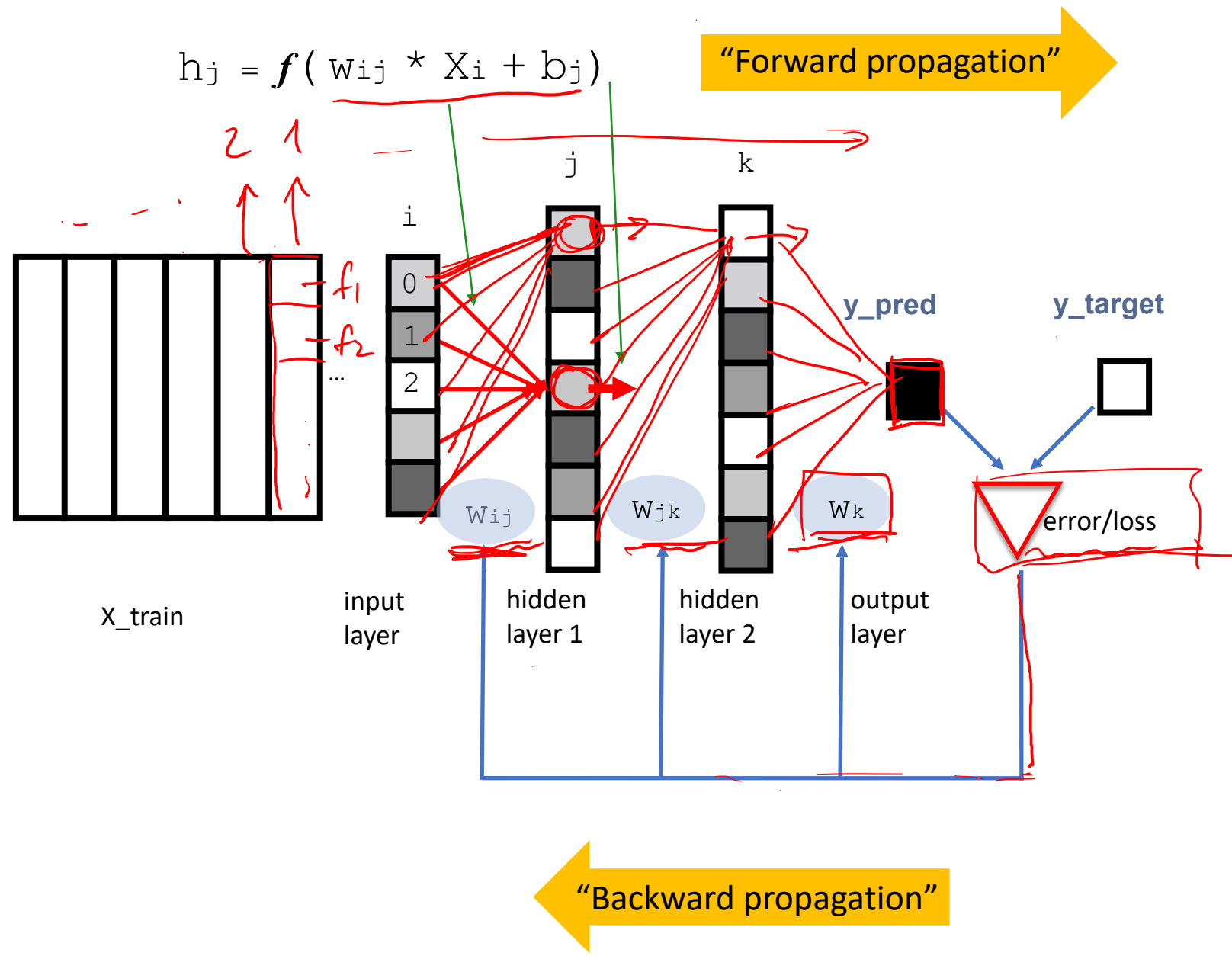
$$\omega_j \leftarrow \omega_j - \alpha \frac{\partial \mathcal{L}}{\partial \omega_j}$$

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

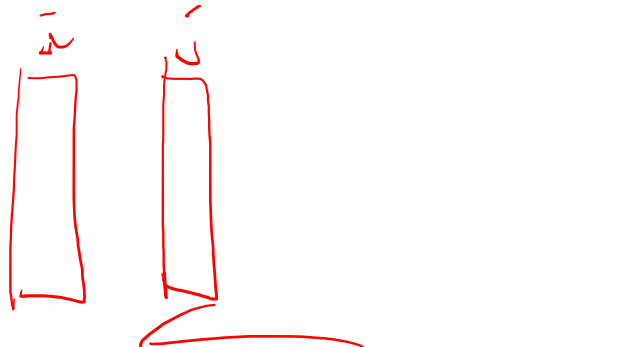
$$\hat{y}_i = \sum_j \omega_j X_{ij}$$

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

# How Neural Network Training Works



# Weight Update in deep neural nets


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

**Weight Update Rule**

**Loss Function**

**Gradient (Chain Rule)**

**Back Propagation**

# Chain Rule Reminder

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

*(Note: In the original image, 'g' is circled in red with an arrow pointing to it, and the partial derivative symbols are underlined in red.)*

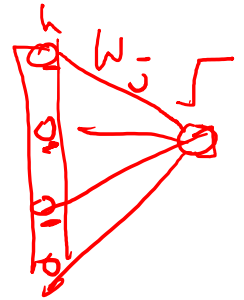
*(Note: In the original image, 'f(g(h(...)))' is written in red above the equation.)*

$$\frac{\partial \sigma}{\partial z} = \frac{\partial}{\partial z} \left( \frac{1}{1 + e^{-z}} \right)$$

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

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

Example: gradient of sigmoid



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

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

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

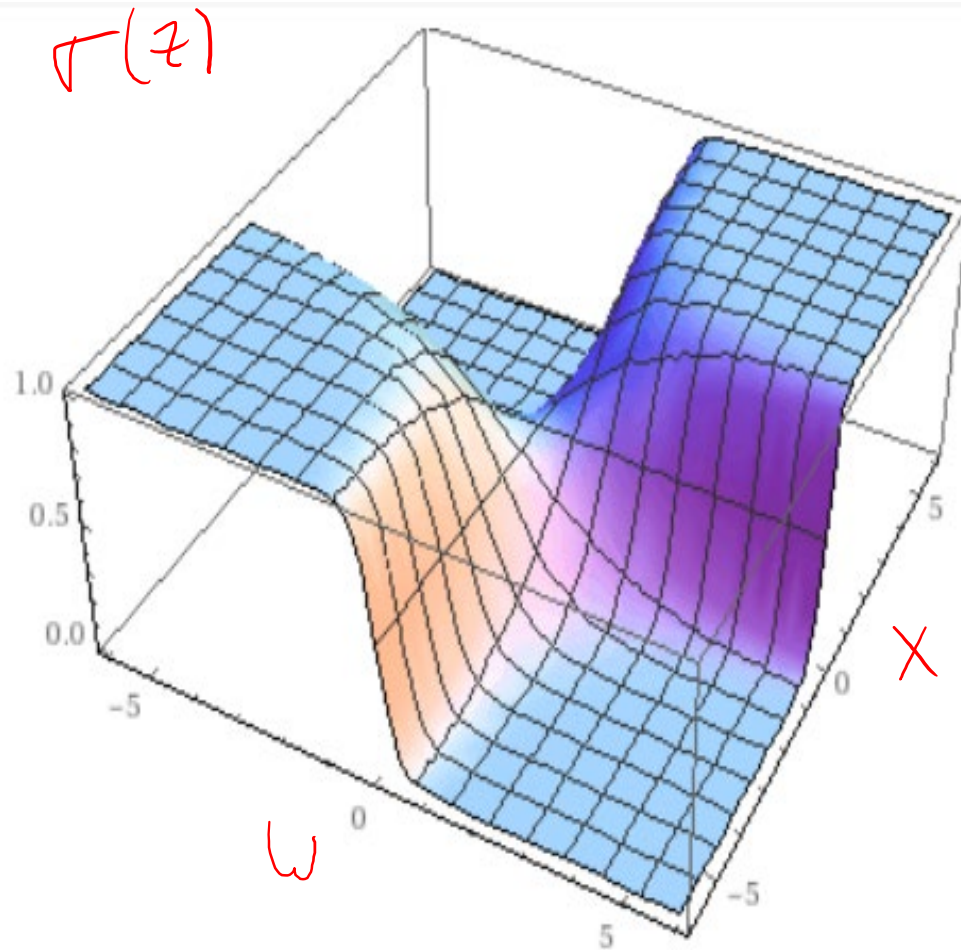
$$\frac{\partial \sigma}{\partial w_j} = \underbrace{\frac{\partial \sigma}{\partial z}}_{\sigma \cdot (1 - \sigma)} \cdot \underbrace{\frac{\partial z}{\partial w_j}}_{x_j}$$

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

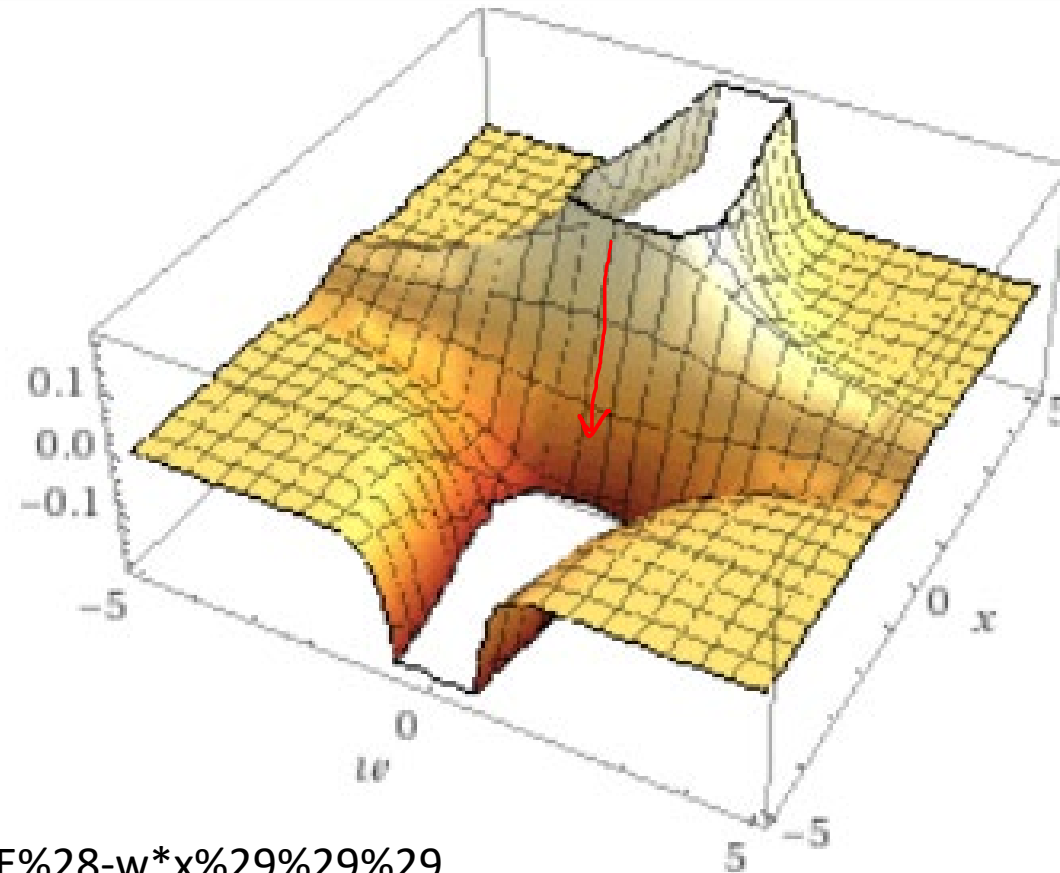
$$\frac{\partial \sigma}{\partial w_j} = \sigma \cdot (1 - \sigma) \cdot x_j$$

# Chain Rule Reminder

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



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



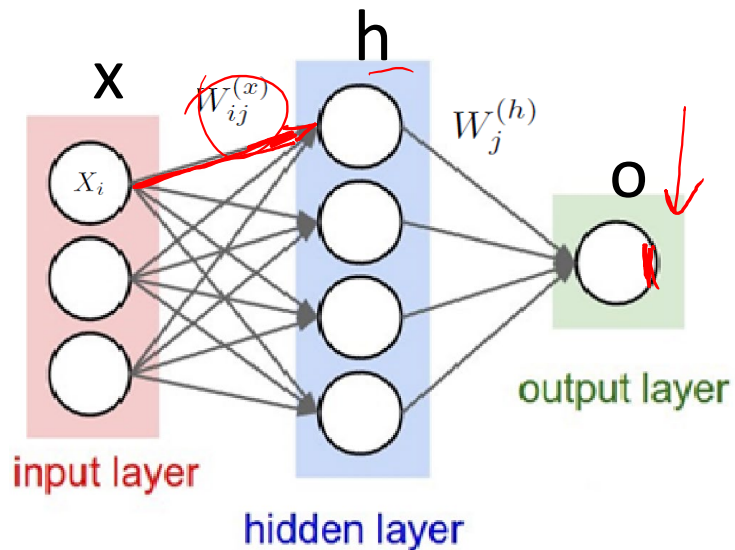


# Calculating Gradient- Chain Rule

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

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

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



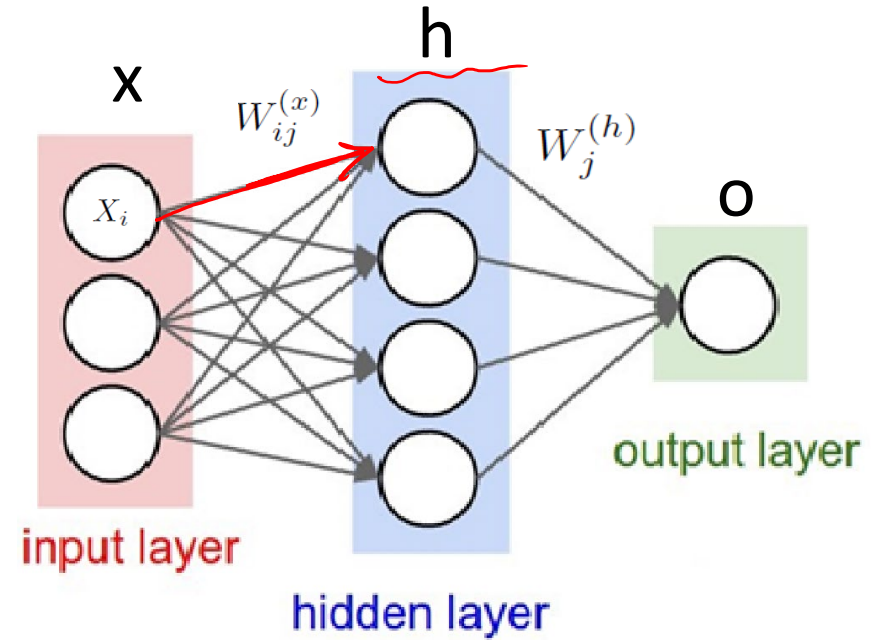
$$\frac{\partial \mathcal{L}}{\partial W_{ij}^{(x)}} = \frac{\partial \mathcal{L}(a^{(o)})}{\partial a^{(o)}} \frac{\partial a^{(o)}}{\partial W_{ij}^{(x)}}$$

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

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

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

# Calculating Gradient- Chain Rule



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

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

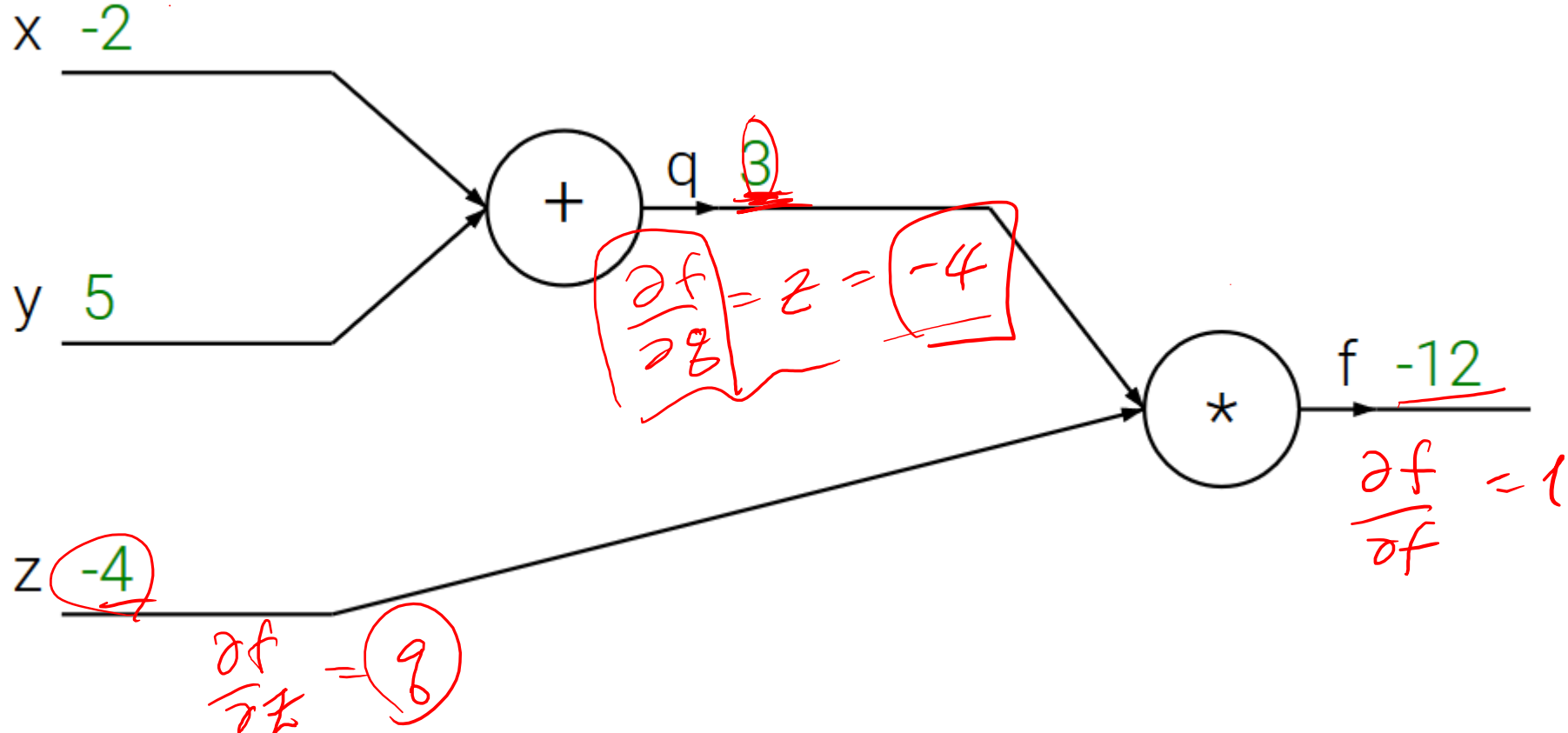
$$\frac{\partial \mathcal{L}}{\partial W_{ij}^{(x)}} = \left( \frac{y}{a^{(o)}} - \frac{1-y}{1-a^{(o)}} \right) \sigma(z^{(o)}) (1 - \sigma(z^{(o)})) \sum_j^{n^{(h)}} W_j^{(h)} \sigma(z_j^{(h)}) (1 - \sigma(z_j^{(h)})) X_i$$

# Back Propagation- Computation Graph

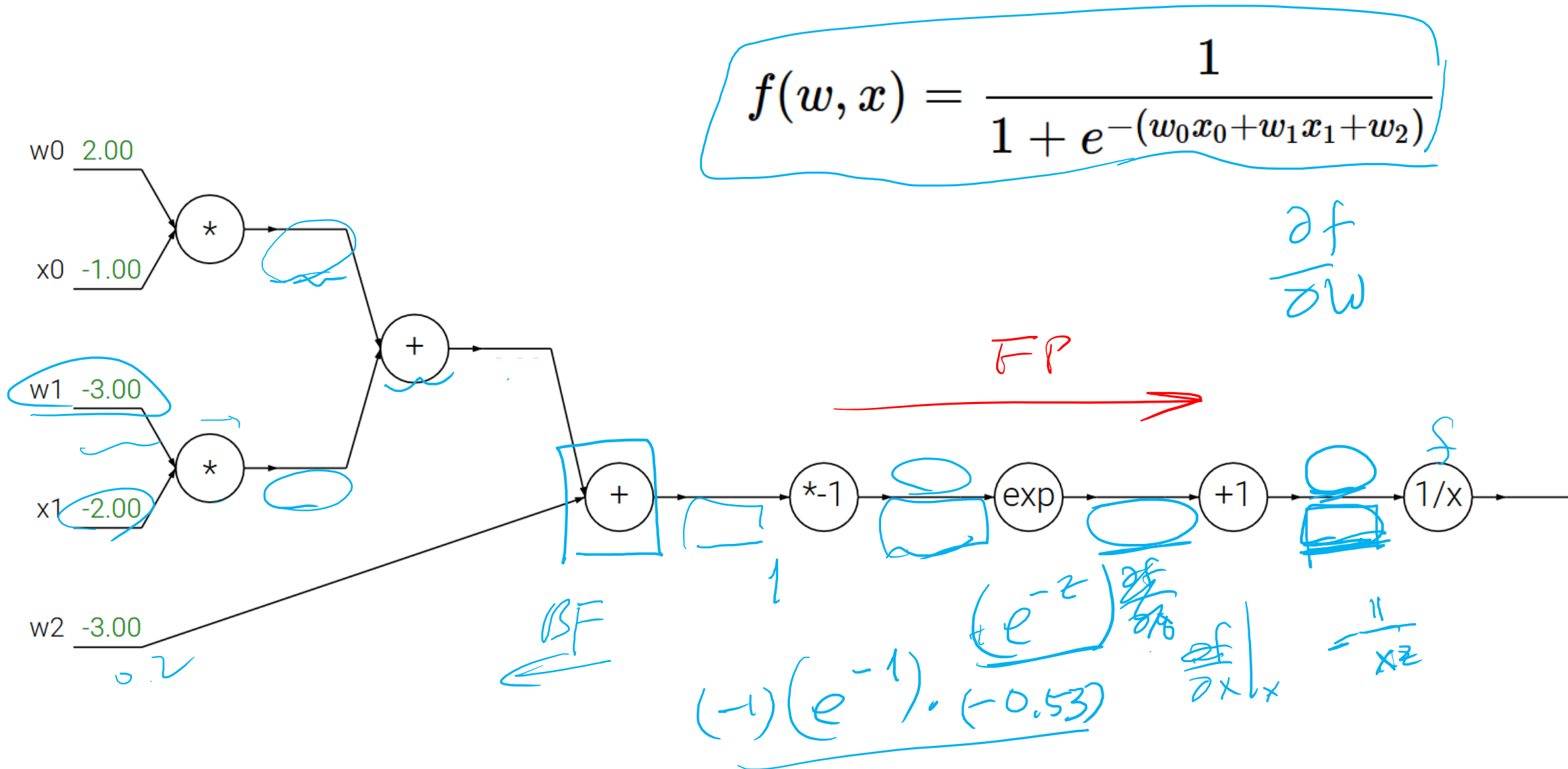
$$q = x + y$$

$$f = q * z$$

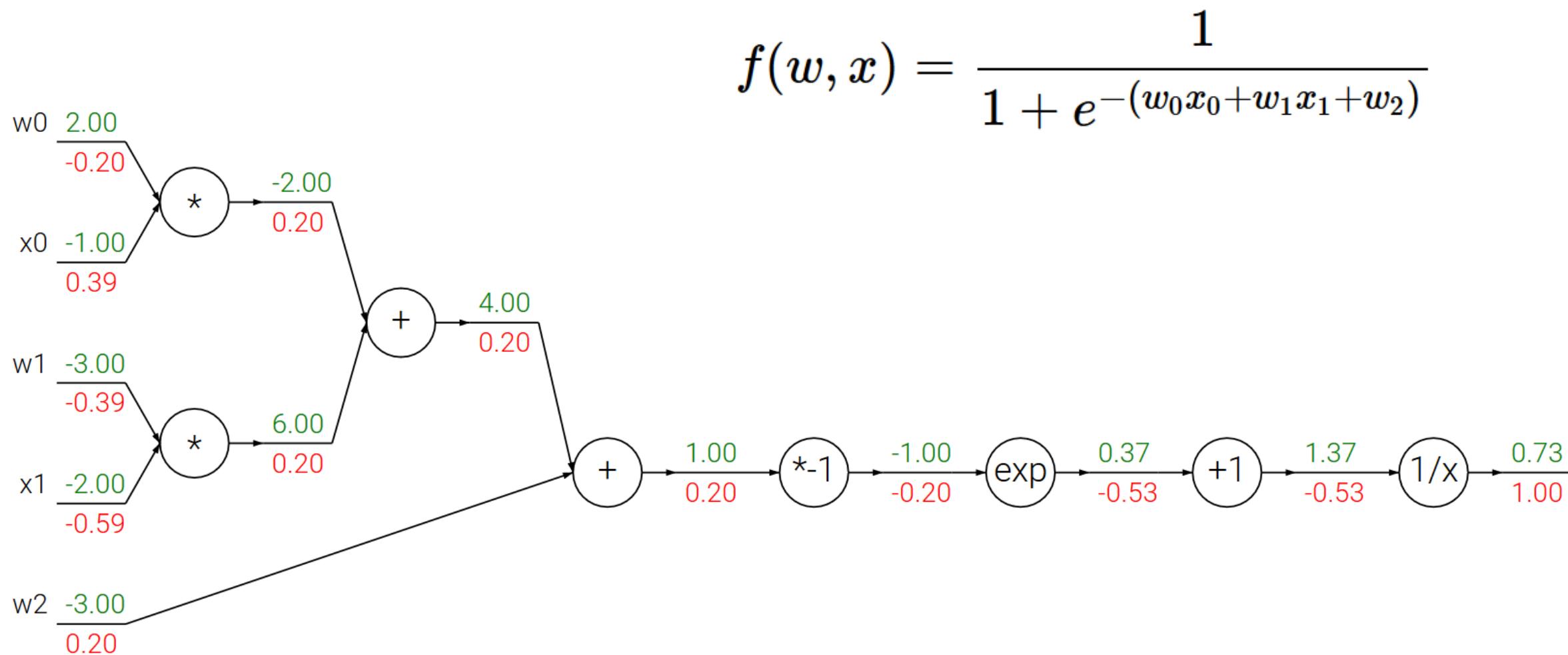
forward pass



# Back Propagation- Computation Graph



# Back Propagation- Computation Graph



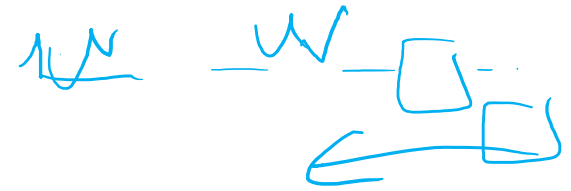
# How does the computer perform differentiation?

## Automatic Differentiation (Autodiff)

```
10 defvjp(anp.add,          lambda g, ans, x, y : unbroadcast(x, g),
11                        lambda g, ans, x, y : unbroadcast(y, g))
12 defvjp(anp.multiply,    lambda g, ans, x, y : unbroadcast(x, y * g),
13                        lambda g, ans, x, y : unbroadcast(y, x * g))
14 defvjp(anp.subtract,    lambda g, ans, x, y : unbroadcast(x, g),
15                        lambda g, ans, x, y : unbroadcast(y, -g))
16 defvjp(anp.divide,      lambda g, ans, x, y : unbroadcast(x, g / y),
17                        lambda g, ans, x, y : unbroadcast(y, -g * x / y**2))
18 defvjp(anp.true_divide, lambda g, ans, x, y : unbroadcast(x, g / y),
19                        lambda g, ans, x, y : unbroadcast(y, -g * x / y**2))
20 defvjp(anp.power,
21        lambda g, ans, x, y: unbroadcast(x, g * y * x ** anp.where(y, y - 1, 1.)),
22        lambda g, ans, x, y: unbroadcast(y, g * anp.log(replace_zero(x, 1.)) * x ** y))
```

[https://github.com/mattjj/autodidact/blob/master/autograd/numpy/numpy\\_vjps.py](https://github.com/mattjj/autodidact/blob/master/autograd/numpy/numpy_vjps.py)

→ [https://www.cs.toronto.edu/~rgrosse/courses/csc321\\_2018/slides/lec10.pdf](https://www.cs.toronto.edu/~rgrosse/courses/csc321_2018/slides/lec10.pdf)



$$\frac{\partial f}{\partial y} \downarrow$$
$$f = \left(\frac{1}{y}\right) \quad \left(-\frac{1}{y^2}\right) \cdot g$$

exp  
log