

Computational Intelligence

Samaneh Hosseini

Isfahan University of Technology

Outline

- Training Neural Networks
 - Loss Optimization
 - Gradient Descent
 - Backpropagation

Training Neural Networks

$$\mathcal{L}(\hat{y}, y)$$

Loss Optimization

We want to find the network weights that **achieve the lowest loss**

$$\mathbf{W}^* = \operatorname{argmin}_{\mathbf{W}} \frac{1}{n} \sum_{i=1}^n \mathcal{L}(\underbrace{f(x^{(i)}; \mathbf{W}), y^{(i)}}_{\substack{\hat{y}^{(i)} \\ \text{Empirical loss}}})$$

$$\mathbf{W}^* = \operatorname{argmin}_{\mathbf{W}} J(\mathbf{W})$$

Loss Optimization

We want to find the network weights that **achieve the lowest loss**

$$\mathbf{W}^* = \operatorname{argmin}_{\mathbf{W}} \frac{1}{n} \sum_{i=1}^n \mathcal{L}(f(x^{(i)}; \mathbf{W}), y^{(i)})$$

$$\mathbf{W}^* = \operatorname{argmin}_{\mathbf{W}} J(\mathbf{W})$$

Remember:

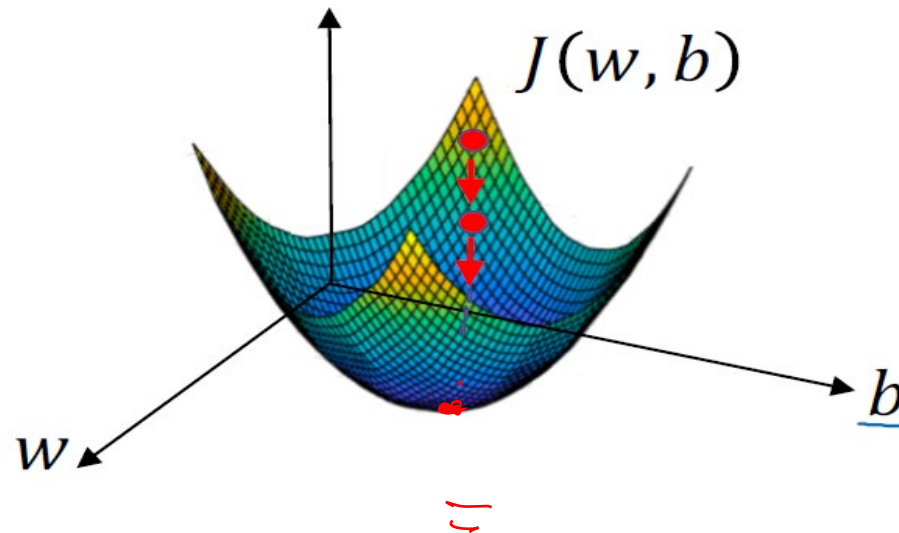
$$\mathbf{W} = \{ \mathbf{W}^{(0)}, \mathbf{W}^{(1)}, \dots \}$$

Cross Entropy Loss Optimization

Recap: $\hat{y} = \sigma(w^T x + b)$, $\sigma(z) = \frac{1}{1+e^{-z}}$

$$J(w, b) = \frac{1}{m} \sum_{i=1}^m \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) = -\frac{1}{m} \sum_{i=1}^m y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})$$

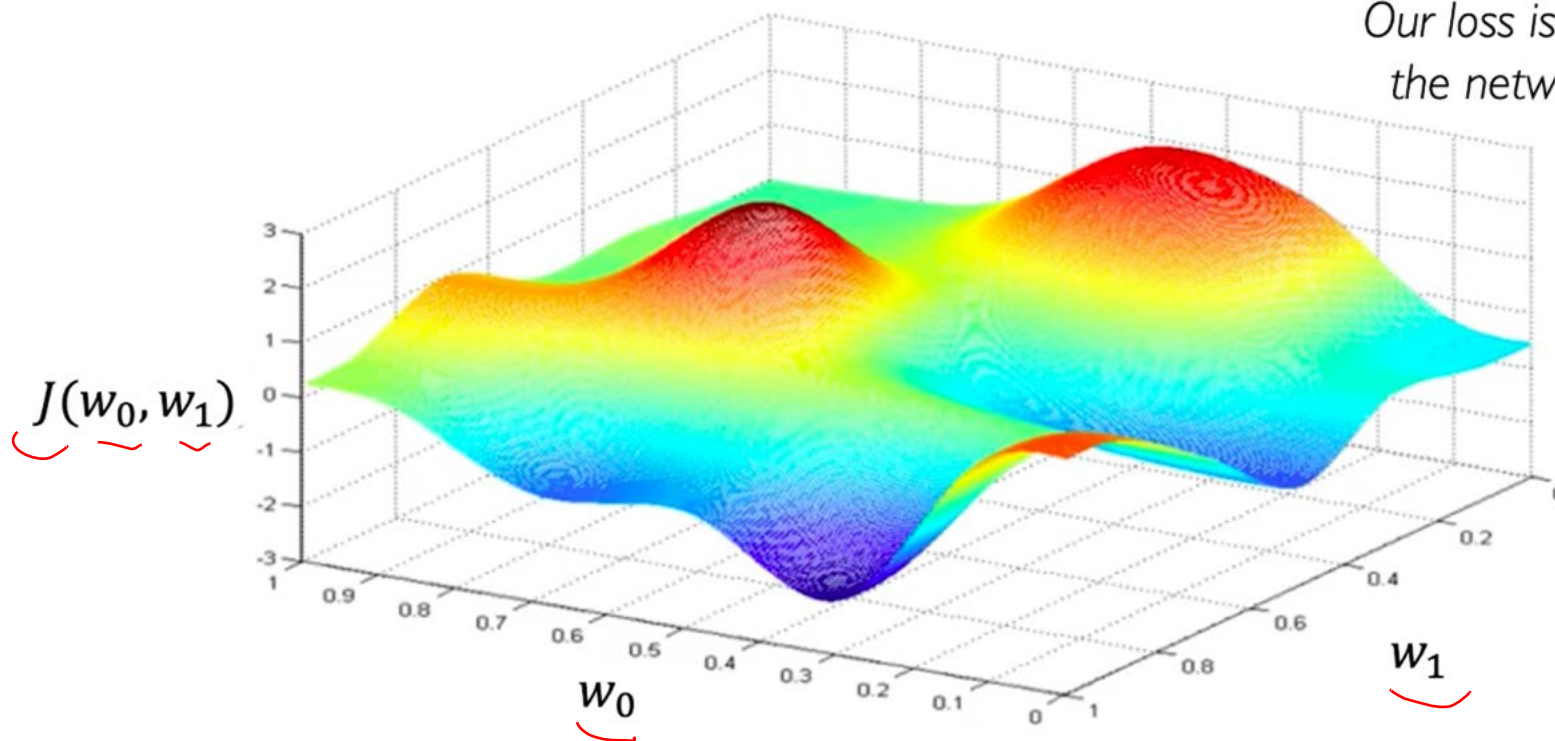
Want to find w, b that minimize $J(w, b)$



Loss Optimization

$$W^* = \underset{W}{\operatorname{argmin}} J(W)$$

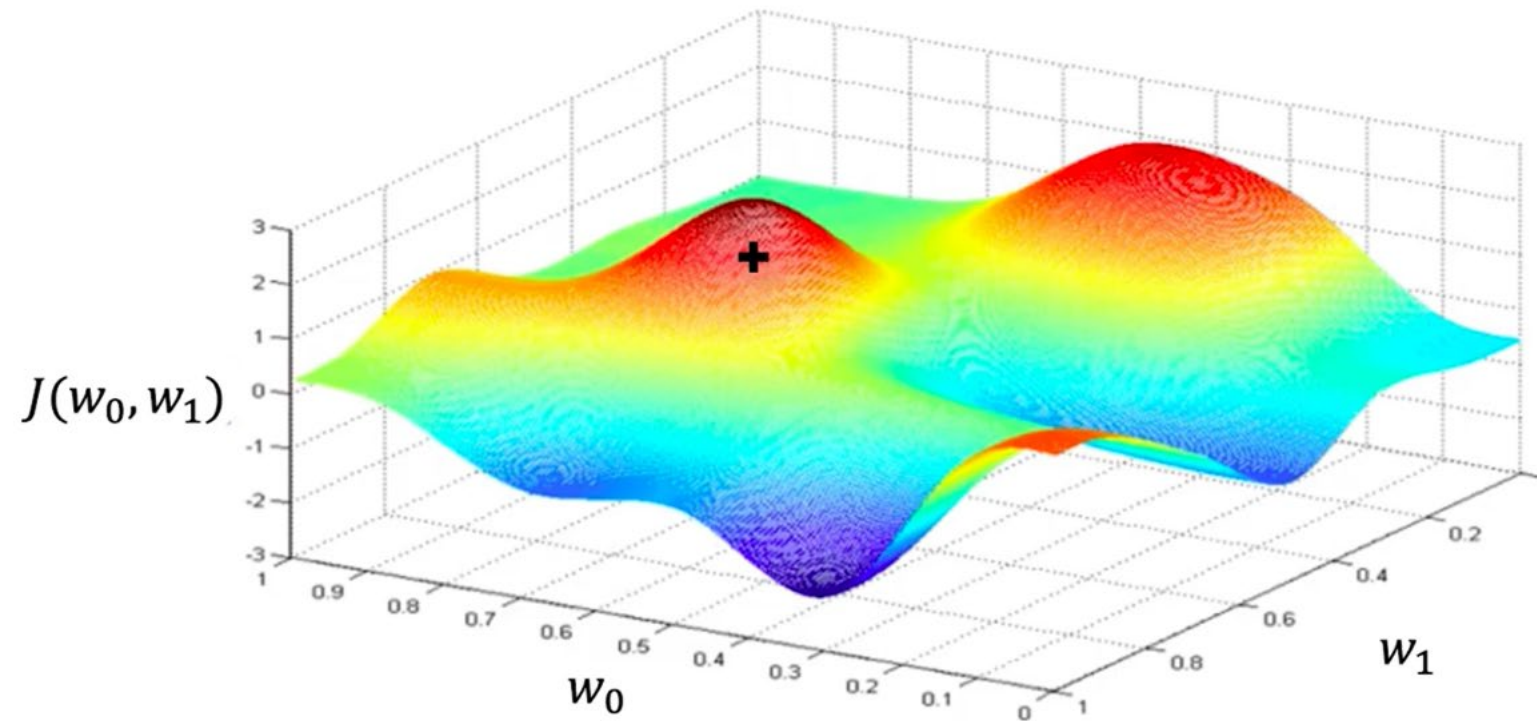
Remember:
Our loss is a function of
the network weights!



non-convex

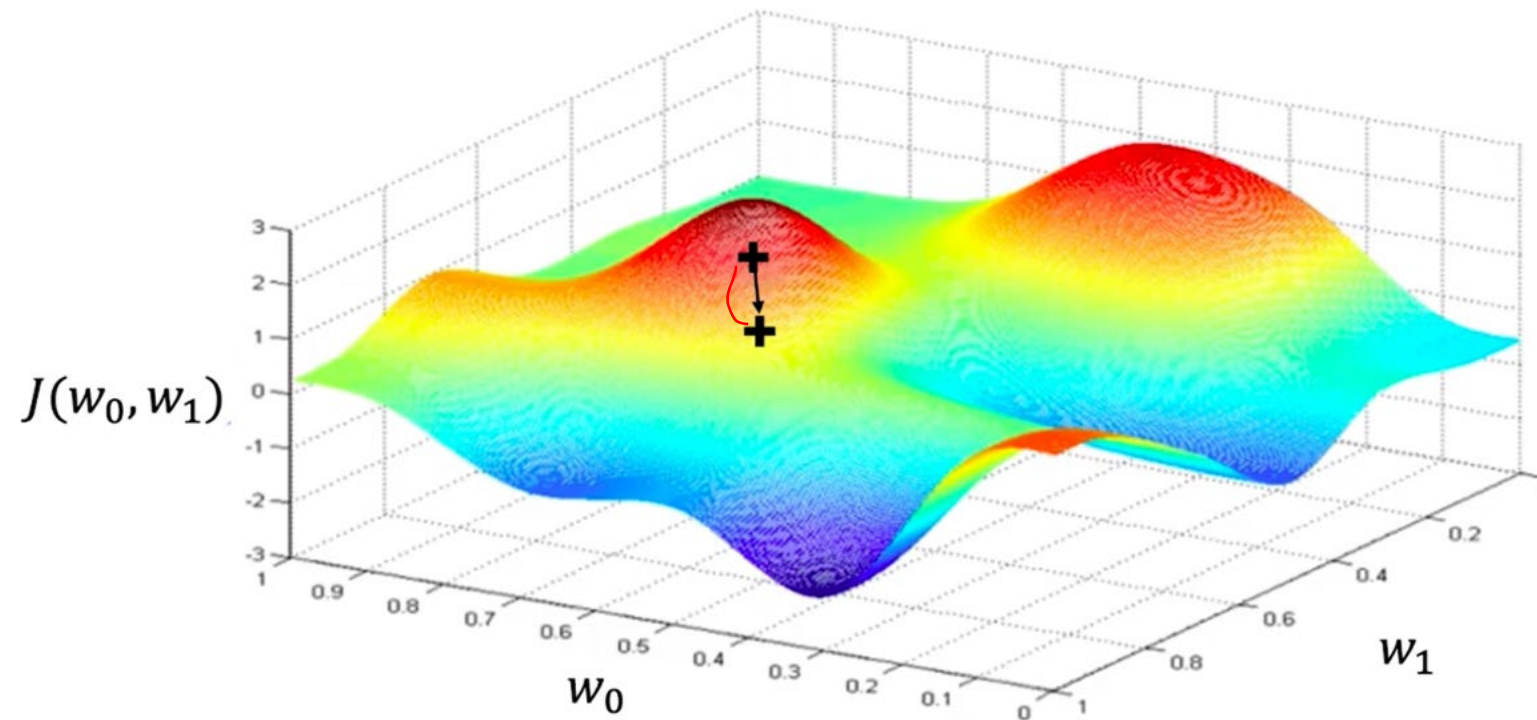
Loss Optimization

Randomly pick an initial (w_0, w_1)



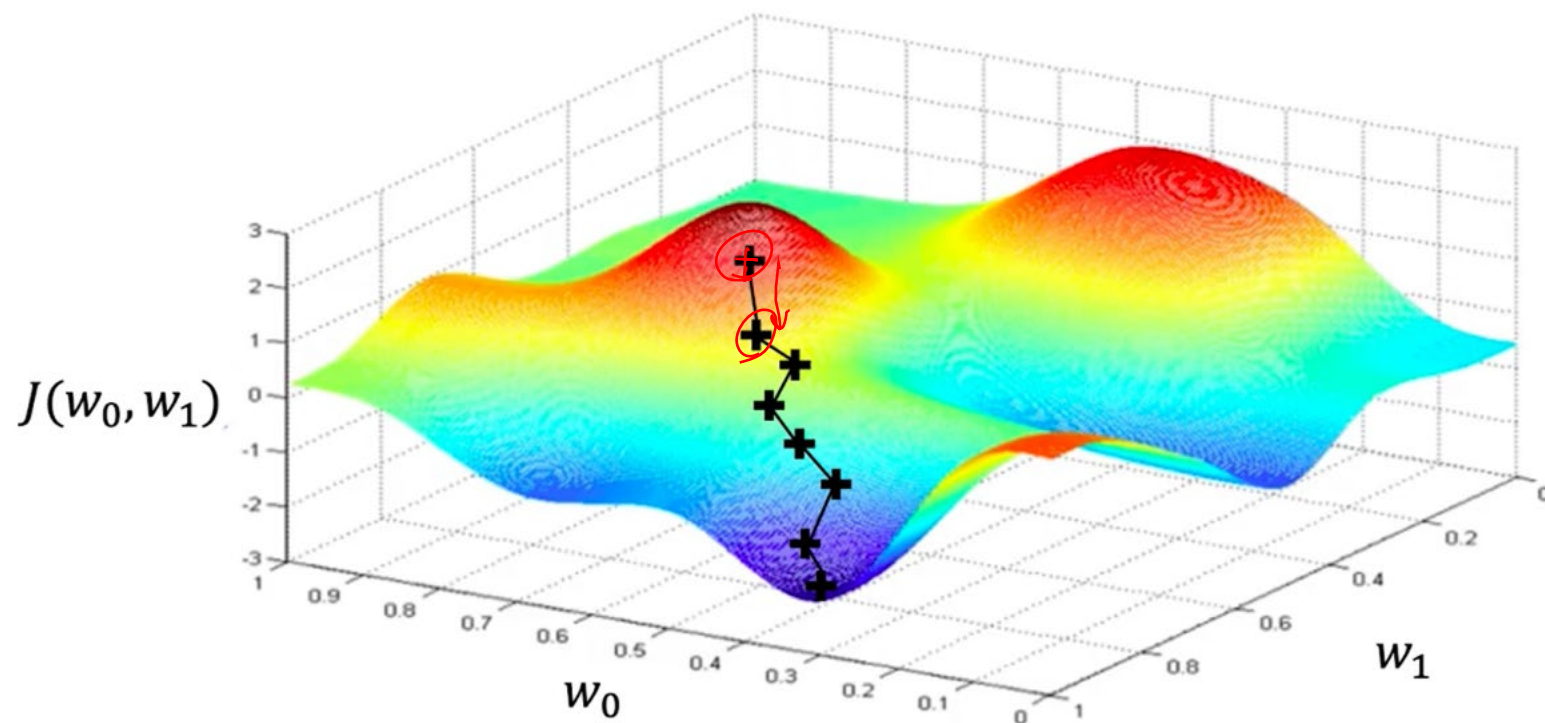
Loss Optimization

Take small step in opposite direction of gradient



Loss Optimization

Repeat until convergence



Gradient Descent

“Walking downhill and always taking a step in the direction that goes down the most.”

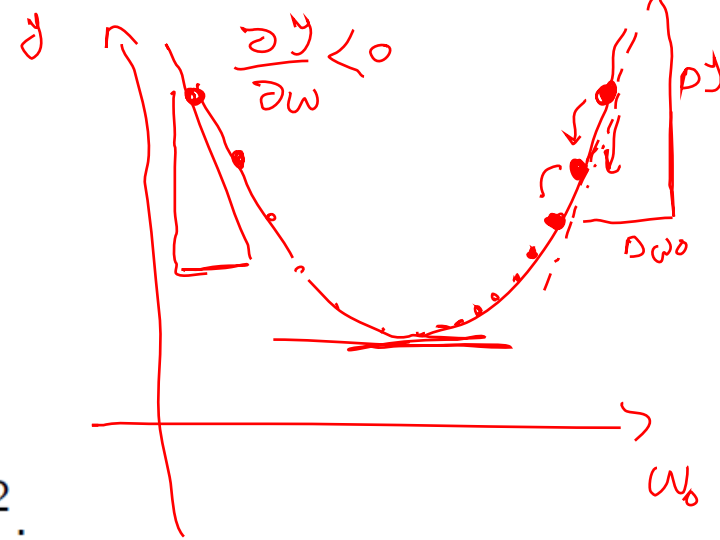
Algorithm

1. Initialize weights randomly $\sim \mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Compute gradient, $\frac{\partial J(\mathbf{w})}{\partial \mathbf{w}}$ $\frac{dJ(\mathbf{w})}{d\mathbf{w}}$
4. Update weights, $\mathbf{w} \leftarrow \mathbf{w} - \eta \frac{\partial J(\mathbf{w})}{\partial \mathbf{w}}$ η learning rate
5. Return weights

Why update the weight proportional to the negative of the partial derivative?

- ▶ Suppose that we want to find the minimum of $y = x^2$.
- ▶ Start with $x = x_0$.
- ▶ In what direction should we change the value of x ?

$$\frac{\partial y}{\partial w}$$
- ▶ By what amount should we change the value of x ?
 What is the step size? η
$$\frac{\partial y}{\partial w}$$



$$\frac{\partial y}{\partial w_0} > 0 \Rightarrow -\eta \left(\frac{\partial y}{\partial w_0} \right) < 0$$

$$\frac{\partial y}{\partial w} < 0 \Rightarrow -\eta \frac{\partial y}{\partial w} > 0$$

Gradient Descent

Algorithm

1. Initialize weights randomly $\sim \mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Compute gradient, $\frac{\partial J(W)}{\partial W}$
4. Update weights, $\underline{W} \leftarrow \underline{W} - \eta \frac{\partial J(W)}{\partial W}$
5. Return weights



```
import tensorflow as tf

weights = tf.Variable([tf.random.normal()])

while True:    # loop forever
    with tf.GradientTape() as g:
        loss = compute_loss(weights)
        gradient = g.gradient(loss, weights)

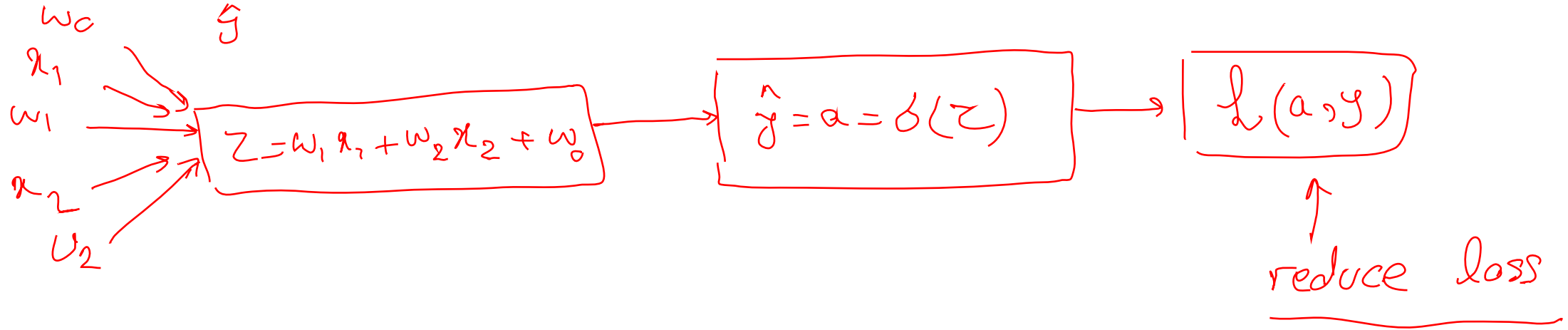
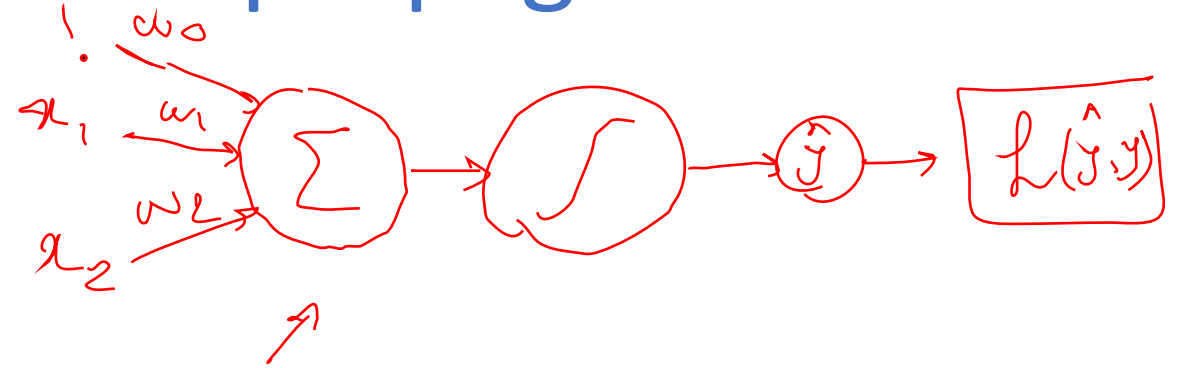
    weights = weights - lr * gradient
```

Computing Gradients: Backpropagation

$$z = w^T x + w_0$$

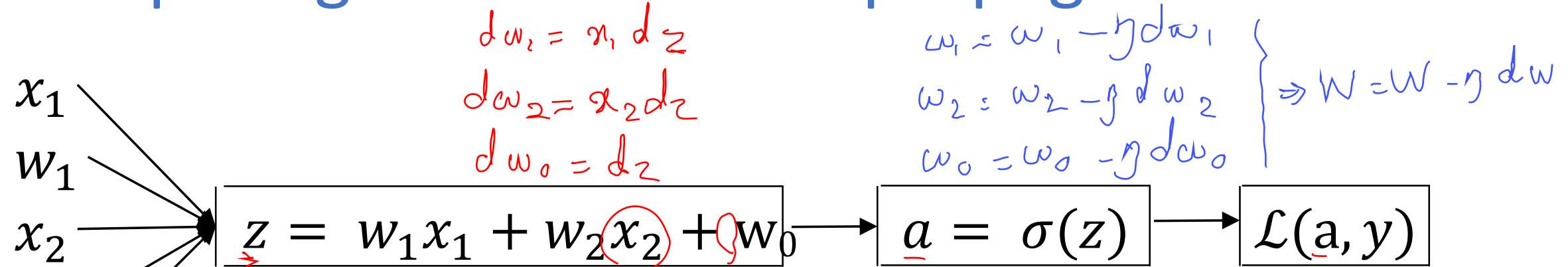
$$\hat{y} = a = \sigma(z)$$

$$\Rightarrow \mathcal{L}(a, y) = -(y \log(a) + (1 - y) \log(1 - a))$$



$$\frac{\partial \mathcal{L}}{\partial w_1} = d w_1$$

Computing Gradients: Backpropagation



$$\frac{\partial \mathcal{L}(a, y)}{\partial w_1} = \frac{\partial \mathcal{L}(a, y)}{\partial a} \cdot \frac{\partial(a)}{\partial(z)} \cdot \frac{\partial(z)}{\partial w_1} = x_1 (a - y) = x_1 d z$$

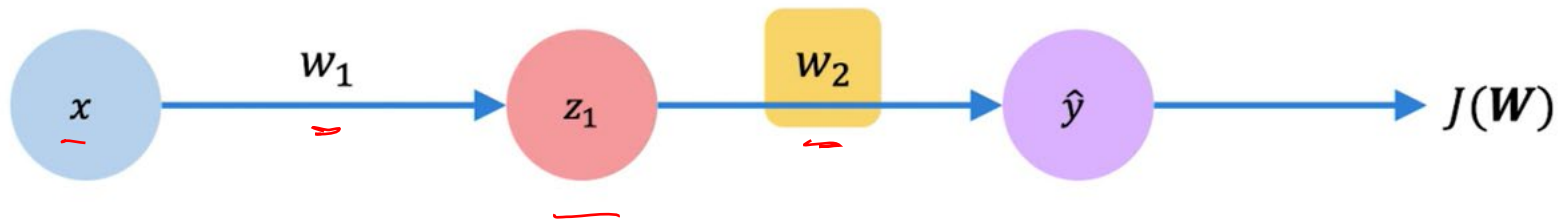
Derivative of the sigmoid function:

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

$$\frac{\partial \mathcal{L}}{\partial w_1}, \frac{\partial \mathcal{L}}{\partial w_2}, \frac{\partial \mathcal{L}}{\partial w_0}$$

$$a - y \rightarrow d z = \frac{\partial \mathcal{L}(a, y)}{\partial a} \cdot \frac{\partial(a)}{\partial z} = \frac{\partial \mathcal{L}}{\partial z} = d z$$

Computing Gradients: Backpropagation



How does a small change in one weight (ex. w_2) affect the final loss $J(\mathbf{W})$?

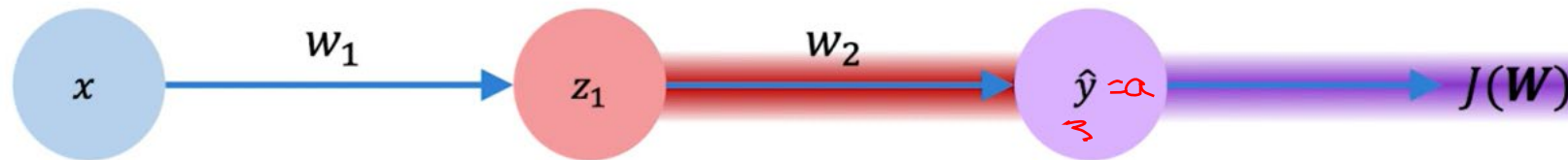
Computing Gradients: Backpropagation



$$\frac{\partial J(W)}{\partial w_2} =$$

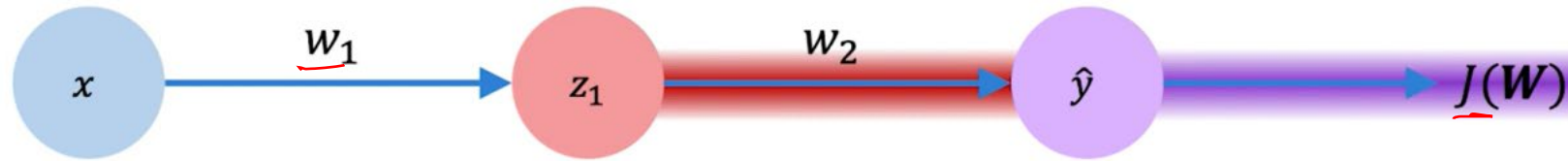
Let's use the chain rule!

Computing Gradients: Backpropagation



$$\frac{\partial J(W)}{\partial w_2} = \underbrace{\frac{\partial J(W)}{\partial \hat{y}}}_{\text{purple}} * \underbrace{\frac{\partial \hat{y}}{\partial w_2}}_{\text{red}}$$

Computing Gradients: Backpropagation

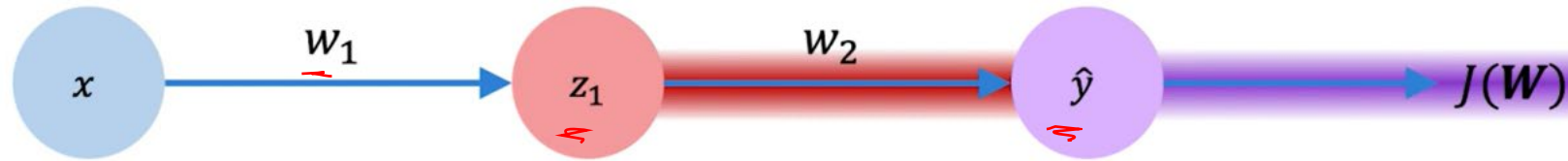


$$\frac{\partial J(W)}{\partial w_2} = \underbrace{\frac{\partial J(W)}{\partial \hat{y}}}_{\text{purple}} * \underbrace{\frac{\partial \hat{y}}{\partial w_2}}_{\text{red}}$$

$$\frac{\partial J(W)}{\partial w_1} =$$

Apply chain rule!

Computing Gradients: Backpropagation



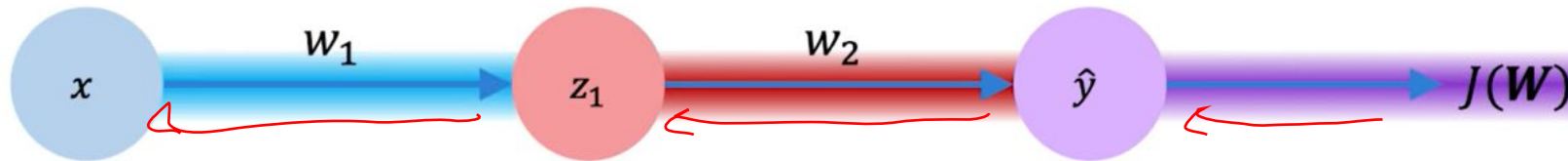
$$\frac{\partial J(W)}{\partial w_2} = \underbrace{\frac{\partial J(W)}{\partial \hat{y}}}_{\text{purple}} * \underbrace{\frac{\partial \hat{y}}{\partial w_2}}_{\text{red}}$$

$$\frac{\partial J(W)}{\partial w_1} = \underbrace{\frac{\partial J(W)}{\partial \hat{y}}}_{\text{purple}} * \underbrace{\frac{\partial \hat{y}}{\partial w_1}}_{\text{red}}$$

Apply chain rule!

Apply chain rule!

Computing Gradients: Backpropagation



$$\frac{\partial J(W)}{\partial w_2} = \underbrace{\frac{\partial J(W)}{\partial \hat{y}}}_{\text{purple}} * \underbrace{\frac{\partial \hat{y}}{\partial w_2}}_{\text{red}}$$

$$\frac{\partial J(W)}{\partial w_1} = \underbrace{\frac{\partial J(W)}{\partial \hat{y}}}_{\text{purple}} * \underbrace{\frac{\partial \hat{y}}{\partial w_1}}_{\text{red}}$$

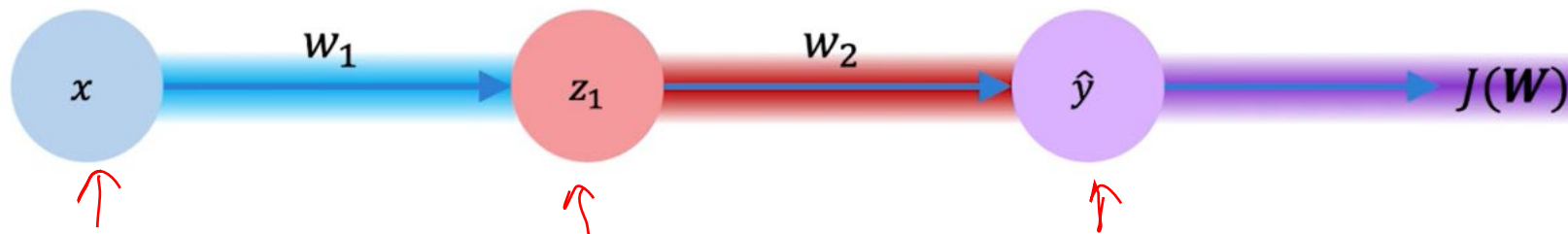
Apply chain rule! (pointing to $\frac{\partial J(W)}{\partial w_1}$)

Apply chain rule! (pointing to $\frac{\partial \hat{y}}{\partial w_1}$)

$$\frac{\partial J(W)}{\partial w_1} = \underbrace{\frac{\partial J(W)}{\partial \hat{y}}}_{\text{purple}} * \underbrace{\frac{\partial \hat{y}}{\partial z_1}}_{\text{red}} * \underbrace{\frac{\partial z_1}{\partial w_1}}_{\text{blue}}$$

Handwritten red marks: a bracket under the first two terms and a squiggle under the third term.

Computing Gradients: Backpropagation



$$\frac{\partial J(W)}{\partial w_1} = \underbrace{\frac{\partial J(W)}{\partial \hat{y}}}_{\text{purple}} * \underbrace{\frac{\partial \hat{y}}{\partial z_1}}_{\text{red}} * \underbrace{\frac{\partial z_1}{\partial w_1}}_{\text{blue}}$$

Repeat this for **every weight in the network** using gradients from later layers

Gradient descent on n examples

$$J(\omega, \omega_0) = \frac{1}{n} \sum_{i=1}^n \ell(\underset{\hat{y}}{a^{(i)}}, y^{(i)})$$

$$a^{(i)} = \hat{y}^{(i)} = \sigma(z^{(i)}) = \sigma(\omega^T x^{(i)} + \omega_0)$$

$$\frac{\partial J(\omega)}{\partial \omega_1} = \frac{1}{n} \sum_{i=1}^n \frac{\partial \ell(a^{(i)}, y^{(i)})}{\partial \omega_1}$$

$$\underbrace{\frac{\partial \ell(a^{(i)}, y^{(i)})}{\partial \omega_1}}_{d_{\omega_1}^{(i)}} = (x^{(i)}, y^{(i)})$$

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

$$d_{\omega_1}^{(i)}, d_{\omega_2}^{(i)}$$

$$d_{\omega_0}^{(i)}$$

$$d\omega_1 = \frac{\partial J}{\partial \omega_1}$$

Gradient descent on n examples

$$J=0 \ ; \ d\omega_1 = d\omega_2 = d\omega_0 = 0 \ ;$$

→ For $i=1$ to \underline{m}

$$z^{(i)} = \omega^T X^{(i)} + \omega_0$$

$$a^{(i)} = \hat{y}^{(i)} = \sigma(z^{(i)})$$

$$J += - [y^{(i)} \log a^{(i)} + (1 - y^{(i)}) \log (1 - a^{(i)})]$$

$$dz^{(i)} = a^{(i)} - y^{(i)}$$

$$d\omega_0 += dz^{(i)}$$

$$d\omega_1 += x_1^{(i)} dz^{(i)}$$

$$d\omega_2 += x_2^{(i)} dz^{(i)}$$

$$J = J/n \ ; \ d\omega_0 = d\omega_0/n \ ; \ d\omega_1 /= n \ ; \ d\omega_2 /= n \ ;$$

$$\omega_1 := \omega_1 - \eta d\omega_1$$

$$\omega_2 := \omega_2 - \eta d\omega_2$$

$$\omega_0 := \omega_0 - \eta d\omega_0$$

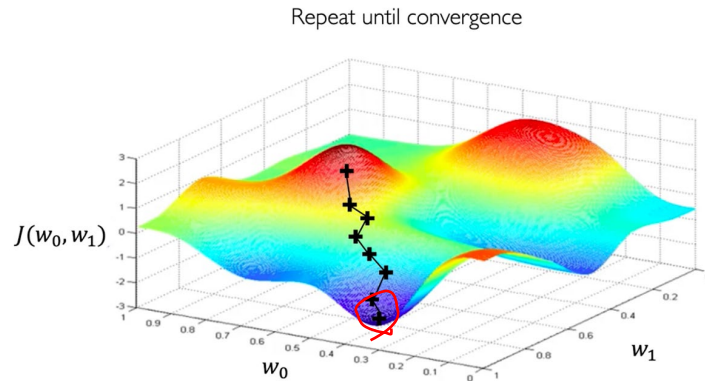
Vectorization

Core Foundation Review

Loss Optimization

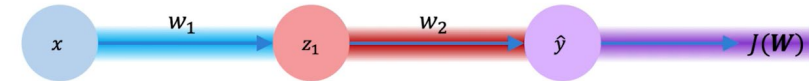
Gradient Descent

Backpropagation



Algorithm

1. Initialize weights randomly $\sim \mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Compute gradient, $\frac{\partial J(W)}{\partial W}$
4. Update weights, $W \leftarrow W - \eta \frac{\partial J(W)}{\partial W}$
5. Return weights



$$\frac{\partial J(W)}{\partial w_1} = \frac{\partial J(W)}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial z_1} * \frac{\partial z_1}{\partial w_1}$$

The equation shows the chain rule for backpropagation. The terms are color-coded: $\frac{\partial J(W)}{\partial \hat{y}}$ is purple, $\frac{\partial \hat{y}}{\partial z_1}$ is red, and $\frac{\partial z_1}{\partial w_1}$ is blue. A red arrow points from the $\frac{\partial J(W)}{\partial W}$ term in the algorithm step 3 to the purple term in the equation.

Repeat this for **every weight in the network** using gradients from later layers