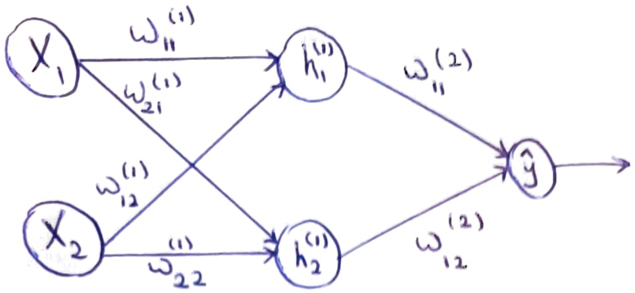
Backpropagation



Convention:

$w_{ij}^{(l)}$ → # of layer (l)

i → ith neuron

j → jth input

Firstly, let's get the forward pass equations:

First Hidden Layer

$$z_1^{(1)} = w_{11}^{(1)} x_1 + w_{12}^{(1)} x_2 + b_1^{(1)} \Rightarrow h_1^{(1)} = ReLU(z_1^{(1)})$$

$$z_2^{(1)} = w_{21} x_1 + w_{22} x_2 + b_2^{(1)} \Rightarrow h_2^{(1)} = ReLU(z_2^{(1)})$$

Output Layer

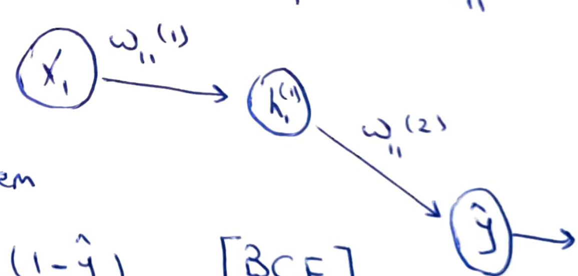
$$z_1^{(2)} = w_{11}^{(2)} h_1^{(1)} + w_{12}^{(2)} h_2^{(1)} + b_1^{(2)} \Rightarrow \hat{y} = \sigma(z_1^{(2)})$$

Secondly, let's try to get eqn of weight update using back propagation

if we want to see how we want to update $w_{11}^{(1)}$

Assume $L = \frac{1}{2}(y - \hat{y})^2$ [MSE]

if we have regression



But since that's a classification problem

Assume $L = -y \log(\hat{y}) - (1-y)\log(1-\hat{y})$ [BCE]

$$\frac{\partial L}{\partial w_{11}^{(1)}} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z_1^{(2)}} \cdot \frac{\partial z_1^{(2)}}{\partial h_1^{(1)}} \cdot \frac{\partial h_1^{(1)}}{\partial z_1^{(1)}} \cdot \frac{\partial z_1^{(1)}}{\partial w_{11}^{(1)}}$$

$$\frac{\partial L}{\partial \hat{y}} = -\frac{y}{\hat{y}} + \frac{1-y}{1-\hat{y}} \quad , \quad \frac{\partial \hat{y}}{\partial z_1^{(2)}} = \hat{y}(1-\hat{y})$$

$$\frac{\partial z_1^{(2)}}{\partial h_1^{(1)}} = w_{11}^{(2)} \quad , \quad \frac{\partial h_1^{(1)}}{\partial z_1^{(1)}} = \begin{cases} 1 & z_1^{(1)} > 0 \\ 0 & z_1^{(1)} \leq 0 \end{cases}$$

$$\frac{\partial z_1^{(1)}}{\partial w_{11}^{(1)}} = X_1$$

$$\frac{\partial L}{\partial w_{11}^{(1)}} = \begin{cases} \left[\frac{-y}{\hat{y}} + \frac{1-y}{1-\hat{y}}\right] \hat{y}(1-\hat{y})\, w_{11}^{(2)} X_1 & z_1^{(1)} > 0 \\ 0 & z_1^{(1)} \leq 0 \end{cases}$$

$$\frac{\partial L}{\partial w_{11}^{(1)}} = \begin{cases} (\hat{y}-y)\, w_{11}^{(2)} X_1 & z_1^{(1)} > 0 \\ 0 & z_1^{(1)} \leq 0 \end{cases}$$

Think about $\frac{\partial L}{\partial b_1^{(1)}}$

$$\frac{\partial L}{\partial b_1^{(1)}} = \underbrace{\frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z_1^{(2)}} \cdot \frac{\partial z_1^{(2)}}{\partial h_1^{(1)}} \cdot \frac{\partial h_1^{(1)}}{\partial z_1^{(1)}}}_{\text{It is already Calculated}} \cdot \underbrace{\frac{\partial z_1^{(1)}}{\partial b_1^{(1)}}}_{=1}$$

Layer

$\delta_1^{(1)}$ (error signal) of neuron one in first hidden layer

no. of neuron

δ typically tells how much that neuron contributed to overall loss.

$$\frac{\partial L}{\partial b_1^{(1)}} = \delta_1^{(1)}$$

$$\frac{\partial L}{\partial w_{12}^{(1)}} = \delta_1^{(1)} x_2 \quad , \quad \frac{\partial L}{w_{11}^{(1)}} = \delta_1^{(1)} x_1$$

So we need gradients of each weight and each bias

Gradients tell you how much changing the weight ould change the loss

$$\frac{\partial L}{\partial w_{ij}} = \delta_i^{(l)} a_j^{(l-1)} \quad , \quad \frac{\partial L}{\partial b_i^{(l)}} = \delta_i^{(l)}$$

So, if $a_j^{(l-1)}$ (input) is large

and $\delta_i^{(l)}$ (Sensitivity to loss) is large

∴ Weight has big effect on loss (should be adjusted)

After calculating the gradient, update the weights using learning rate η

$$w_{ij}^{(l)} \leftarrow w_{ij}^{(l)} - \eta \frac{\partial L}{\partial w_{ij}^{(l)}}$$

$$b_i^{(l)} \leftarrow b_i^{(l)} - \eta \frac{\partial L}{\partial b_i^{(l)}}$$

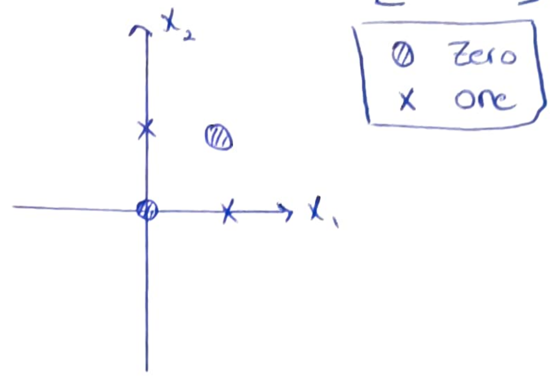
Note: Variants like Momentum, Adam, etc.. modify this basic rule by adding terms that depend on the past gradients But the core idea remains subtracting (learning rate × gradient)

Key Points:

- Learning rate η controls how big is the step
  Too large η → overshoot
  Too small η → slow learning
- You perform these updates after each forward + backward pass (each training example or mini-batch)

For Simplicity, let's look at this dataset [XOR]:

x_1	x_2	target
0	0	0
0	1	1
1	0	1
1	1	0



∴ not <u>linear</u> data ∴ NN

<u>Tabular</u> data $\longrightarrow$ MLP

let's initialize random weights: $b^1_{2\times1} = \begin{bmatrix} b^1_1 \\ b^1_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 0.5 \end{bmatrix}$

$$\omega^1_{2\times2} = \begin{bmatrix} \omega^1_{11} & \omega^1_{12} \\ \omega^1_{21} & \omega^1_{22} \end{bmatrix} = \begin{bmatrix} 1 & 0.5 \\ -0.5 & 1 \end{bmatrix}$$

$$\omega^2_{1\times2} = \begin{bmatrix} \omega^2_{11} & \omega^2_{12} \end{bmatrix} = \begin{bmatrix} -1 & 0.5 \end{bmatrix}, \; b^2_1 = 1$$

x_1	x_2	$z^{(1)}_1$	$h^{(1)}_1$	$h^{(2)}_2$	z^2_1	$\hat{y}$	y	BCE
0	0	1	1	0.5	0.25	0.562	0	0.82
0	1	1.5	1.5	1.5	0.25	0.562	1	0.57
1	0	2	2	0	-1	0.27	1	1.3
1	1	2.5	2.5	1	-1	0.27	0	0.31

<u>Understand the numbers</u>

$\hookrightarrow z^{(1)}_1$ is always <u>+ve</u>

After Sample 1

$$\frac{\partial L}{\partial b^{(1)}_1} = \delta^{(1)}_1 = (\hat{y} - y)\,\omega^{(2)}_{11} = (0.562 - 0)(-1) = -0.562$$

let $\eta = 0.1$

$$\frac{\partial L}{\partial \omega^{(1)}_{11}} = -\delta^{(1)}_1 X_1 = -0.562(0) = 0$$

$$b_1 \leftarrow b_1 - \eta \frac{\partial L}{\partial b_1} = 1 - 0.1(-0.562) = 1.056$$

no update for weight

Note: This NN won't yield 100% accuracy (it should since it's XOR), but why?

It's so simple, adding extra hidden layer will solve this problem, but I used this NN for simplicity, now you got the core idea.

Updating weights after each sample isn't a good idea

- Noisy
- Inefficiency
- Convergence Stability

Solution?

Batch or Mini-batch gradient descent

$$\frac{\partial L}{\partial b_1^{(1)}} = \frac{1}{n} \sum_{i=1}^{n} (\hat{y}_i - y_i)\, w_{11}^{(2)}$$

$$\frac{\partial L}{\partial w_{11}^{(1)}} = \frac{1}{n} \sum_{i=1}^{n} (\hat{y}_i - y_i)\, w_{11}^{(2)}\, x_{1i}$$

Notice how it updates the weights only once but we subtract each y from $\hat{y}$

In other words, we don't average $\hat{y}_i - y_i$

But we average the gradient itself.

#