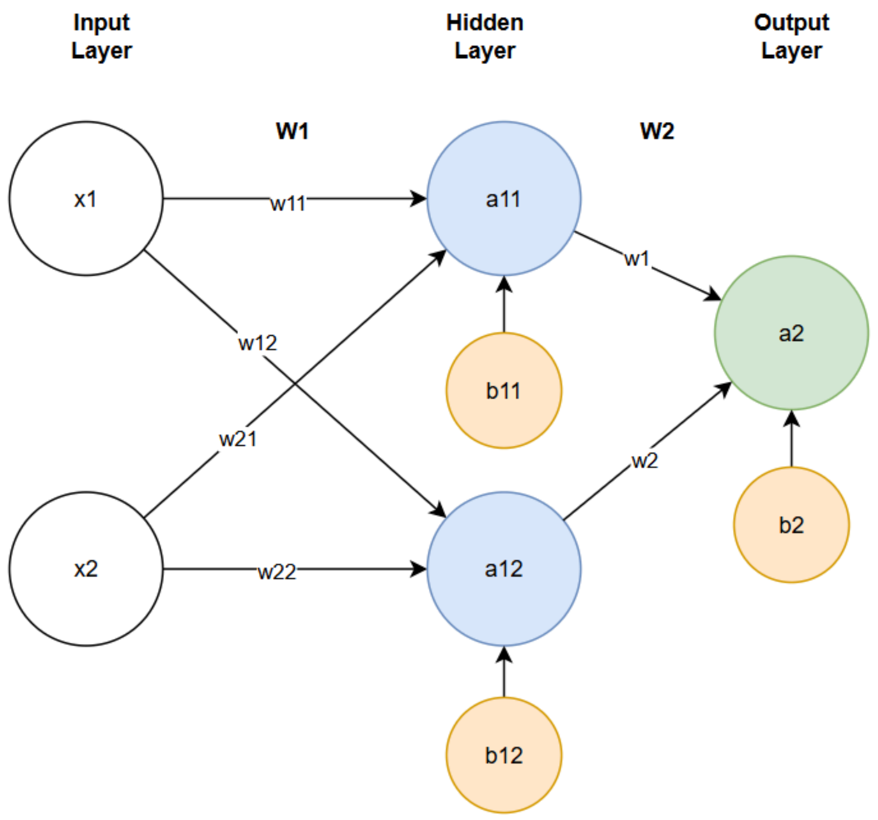
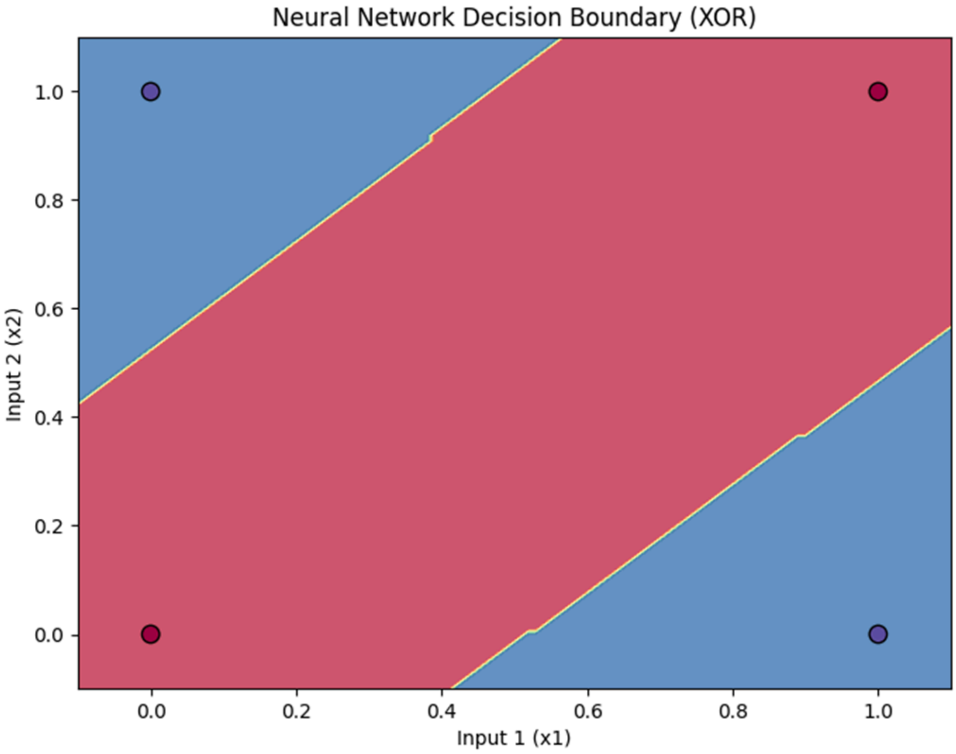
Introduction

Goal: Train a neural network with one hidden layer to create a decision boundary that matches the XOR gate. Uses the binary cross entropy loss.



$$x = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix},\ y = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix},\ W1 = \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{bmatrix},\ W2 = \begin{bmatrix} w_{11} \\ w_{21} \end{bmatrix},\ b_1 = \begin{bmatrix} b_{11} \\ b_{12} \end{bmatrix},\ b_2 = \begin{bmatrix} b_2 \end{bmatrix}$$

Expected results like:



Forward Pass

$$
\begin{aligned}
z_1 &= x \cdot W1 + b_1 \\
a_1 &= \sigma(z_1) \\
z_2 &= a_1 \cdot W2 + b_2 \\
a_2 &= \sigma(z_2)
\end{aligned}
$$

Working out for z_1, similar logic for z_2

$$
\begin{aligned}
z_1 &= \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{bmatrix} + \begin{bmatrix} b_{11} \\ b_{12} \end{bmatrix} \\
&= \begin{bmatrix} 0 + 0 & 0 + 0 \\ 0 + w_{21} & 0 + w_{22} \\ w_{11} + 0 & w_{12} + 0 \\ w_{11} + w_{21} & w_{12} + w_{22} \end{bmatrix} + \begin{bmatrix} b_{11} \\ b_{12} \end{bmatrix} \\
&= \begin{bmatrix} 0 & 0 \\ w_{21} & w_{22} \\ w_{11} & w_{12} \\ w_{11} + w_{21} & w_{12} + w_{22} \end{bmatrix} + \begin{bmatrix} b_{11} \\ b_{12} \end{bmatrix}
\end{aligned}
$$

$$= \begin{bmatrix} b_{11} & b_{12} \\ w_{21} + b_{11} & w_{22} + b_{12} \\ w_{11} + b_{11} & w_{12} + b_{12} \\ w_{11} + w_{21} + b_{11} & w_{12} + w_{22} + b_{12} \end{bmatrix}$$

$$a_1 = \begin{bmatrix} \sigma(b_{11}) & \sigma(b_{12}) \\ \sigma(w_{21} + b_{11)} & \sigma(w_{22} + b_{12}) \\ \sigma(w_{11} + b_{11}) & \sigma(w_{12} + b_{12}) \\ \sigma(w_{11} + w_{21} + b_{11}) & \sigma(w_{12} + w_{22} + b_{12}) \end{bmatrix}$$

$$z_2 = \begin{bmatrix} \sigma(b_{11}) & \sigma(b_{12}) \\ \sigma(w_{21} + b_{11)} & \sigma(w_{22} + b_{12}) \\ \sigma(w_{11} + b_{11}) & \sigma(w_{12} + b_{12}) \\ \sigma(w_{11} + w_{21} + b_{11}) & \sigma(w_{12} + w_{22} + b_{12}) \end{bmatrix} \cdot \begin{bmatrix} w_{11} \\ w_{21} \end{bmatrix}$$

$$= \begin{bmatrix} \sigma(b_{11}) \cdot w_{11} + \sigma(b_{12}) \cdot w_{21} \\ \sigma(w_{21} + b_{11)} \cdot w_{11} + \sigma(w_{22} + b_{12}) \cdot w_{21} \\ \sigma(w_{11} + b_{11}) \cdot w_{11} + \sigma(w_{22} + b_{12}) \cdot w_{21} \\ \sigma(w_{11} + w_{21} + b_{11}) \cdot w_{11} + \sigma(w_{12} + w_{22} + b_{12}) \cdot w_{21} \end{bmatrix}$$

Weight Gradients

The fundamental structure of the gradient, regardless of which loss function or activation function you pick, is **always the upstream error scaled by the derivative of the activation function .**

$$\delta_2 = a_2 - y$$

$$\delta_2 = \frac{\partial L}{\partial z_2} = \frac{\partial L}{\partial a_2} \cdot \frac{\partial a_2}{\partial z_2}$$

For $\frac{\partial a_2}{\partial z_2}$, since $a_2 = \sigma(z_2)$, then $\frac{\partial a_2}{\partial z_2}$ = $\sigma'(z_2)$ = $\sigma(z_2)(1 - \sigma(z_2)) = a_2(1 - a_2)$

For $\frac{\partial L}{\partial a_2}$, since the loss function is the binary cross entropy,

$L(y, a_2) = -(y \log(a_2) + (1 - y)\log(1 - a_2))$, then $\frac{\partial L}{\partial a_2} = \frac{a_2 - y}{a_2(1 - a_2)}$

Therefore $\frac{\partial L}{\partial a_2} \cdot \frac{\partial a_2}{\partial z_2} = \frac{a_2 - y}{a_2(1 - a_2)} \cdot a_2(1 - a_2) = a_2 - y$

Logic for $W2$

First we want to calculate $dW2$, the **average gradient** across all 4 training examples, where x is an $(m \times n)$ size matrix.

$$
\begin{aligned}
dW2 &= \frac{\partial L}{\partial W2} \\
&= \frac{\partial L}{\partial z_2} \cdot \frac{\partial z_2}{\partial W2} \\
&= \frac{\partial L}{\partial a_2} \cdot \frac{\partial a_2}{\partial z_2} \cdot \frac{\partial z_2}{\partial W2} \\
&= \frac{1}{m} \left(a_1^T \cdot \delta_2 \right)
\end{aligned}
$$

For $\frac{\partial z_2}{\partial W2}$, recall $z_2 = a_1 \cdot W2 + b_2$. Therefore, differentiating w.r.t. $W2$, $\frac{\partial z_2}{\partial W2}$ = a_1.
Why transpose a_1? Because recall a_2 has shape $(4, 2)$. Therefore we can't immediately dot product it with δ_2, which has shape $(4,\ 1)$. a_1^T has shape $(2, 4)$, and since num cols now matches with num rows, we can do the dot product.

Logic behind a_1, or $\frac{\partial z_2}{\partial W2}$

We can think of a_1 as $a_1 = \begin{bmatrix} a_{[0,0],1} & a_{[0,0],2} \\ a_{[0,1],1} & a_{[0,1],2} \\ a_{[1,0],1} & a_{[1,0],2} \\ a_{[1,1],1} & a_{[1,1],2} \end{bmatrix}$ If a certain $a_{1,i,j}$ is small/close to 0 for a specific example i, for example $a_1 = \begin{bmatrix} a_{[0,0],1} & a_{[0,0],2} \\ a_{[0,1],1} & a_{[0,1],2} \\ a_{[1,0],1} & a_{[1,0],2} \\ 0.001 & a_{[1,1],2} \end{bmatrix}$, then the neuron a_{11} contributed very little to the output when input was $[1, 1]$, since it "barely" fired, and therefore changing the corresponding weight $W_{2,j,k}$ has almost no effect on the output z_2 .

We can think of $\frac{\partial z_2}{\partial W2}$ as measuring how much changing some $W_{2,j,k}$ connecting neuron j to neuron k changes the output z_2, e.g. $W_{2,\,a11,\,a2}$ or $W_{2,\,a12,\,a2}$. Recall, $z_2 = a_1 \cdot W2 + b_2$. Meaning, **the only factor that scales** $W2$ is a_1.

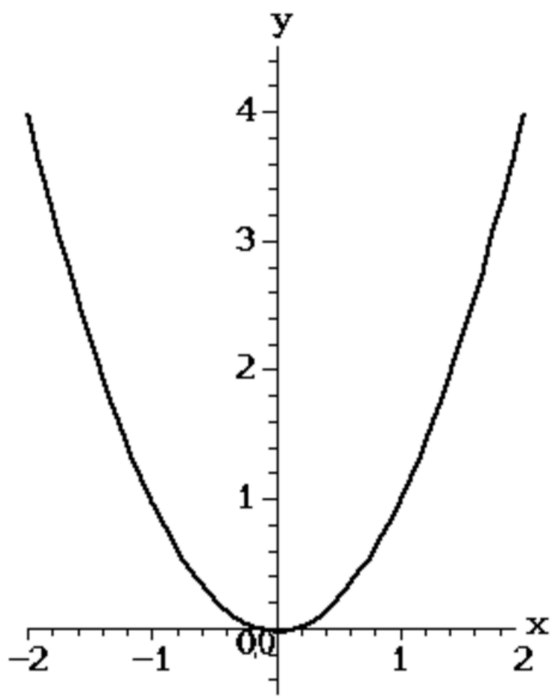
If $a_{1,i,j}$ is small, then the neuron was inactive, thus changing the weight $W_{2,j,k}$ has almost no effect on output z_2. **Low a_1 = low impact of W_2 = low contribution of W_2 to the output.**

Logic behind δ_2

δ_2 or $a_2 - y$ or $\frac{\partial L}{\partial z_2}$ is the **error signal.** measures how **wrong was the prediction** is in terms of the **magnitude and direction** of the loss at point z_2.

Importantly, we think of $a_2 - y$ as the **gradient,** and not the literal absolute error. It just so happens that with the binary cross entropy and with the sigmoid activation it simplifies to this result, but that's not the case for other loss functions/other activation functions.

The sign of δ_2 signifies whether the network over-estimated or underestimated the result. If $a_2 > y$, meaning δ_2 is positive, the network **over-estimated** the result, and $W2$ (which feeds into z_2) needs to be decreased to lower the next prediction. Why W2? because the only thing we can control in the network are weights and biases.



Think of a simple graph, where the x axis is a weight, and y axis is the amount of loss. If we have a positive gradient, that means we are moving "upwards" in the loss function, when we want to find the value of the weight that *minimises* the loss, and therefore we should reduce the weight.

Simplified, if our "ideal" weight is 0, then if we are at 1, we want to decrease the weight from 1→0.

The goal of backpropagation is to find out how much to change the weights to reduce the loss. The absolute value of δ_2 tells the network how big the mistake is.
Mistake A: $a_2 = 0.9$ when target $y = 1.0$ then $\delta_2 = -0.1$.
Mistake B: $a_2 = 0.6$ when target $y = 1.0$ then $\delta_2 = -0.4$.
When δ_2 is propagated backwards at mistake B, it causes the corresponding weight $W2$ to be changed by a much larger amount.

Logic behind final weight gradient

$$\text{Final Weight Gradient} \propto \underbrace{\delta}_{\text{How Wrong the Prediction Is}} \times \underbrace{\mathbf{a}}_{\text{How Much the Weight Contributed}}$$

$\frac{\partial L}{\partial W2}$ determines **how much the weights should be changed.** We do the dot product of a_1 with the **error signal** because **"how much the weights should be changed = how wrong the prediction is $\times$ how much the weight contributed to the output."**

Logic for b_2

$$\begin{aligned} db2 &= \frac{\partial L}{\partial b_2} = \frac{\partial L}{\partial a_2} \cdot \frac{\partial a_2}{\partial z_2} \frac{\partial z_2}{\partial b_2} \\ &= \sum \delta_2 \cdot \frac{1}{m} \end{aligned}$$

Recalling that $\delta_2 = \frac{\partial L}{\partial z_2} = \frac{\partial L}{\partial a_2} \cdot \frac{\partial a_2}{\partial z_2}$, and $\frac{\partial z_2}{\partial b_2} = 1$, by rules of differentiation, since we are differentiating $z_2 = a_1 W2 + b_2$, and recalling that $\delta_2 = \frac{\partial L}{\partial a_2} \cdot \frac{\partial a_2}{\partial z_2}$, and since δ_2 is across all four inputs of x, we need to take the mean.

Error signal, or δ_L in general

Chain rule of backpropagation

First, we need to calculate δ_1.

$$\delta_1 \ = \frac{\partial L}{\partial z_1} \cdot \ \frac{\partial L}{\partial a_1} \cdot \frac{\partial a_1}{\partial z_1}$$

We can't directly calculate $\frac{\partial L}{\partial a_1}$.

The reason why the loss function was direct to calculate for $\frac{\partial L}{\partial a_2}$ is because the loss function L is defined **explicitly using** a_2, i.e. $L \ = \ f(a_2, y)$. **The loss function doesn't contain** a_1.

$$\mathbf{a}_1 \xrightarrow{\mathbf{W}_2, \mathbf{b}_2} \mathbf{z}_2 \xrightarrow{\text{Sigmoid}} \mathbf{a}_2 \xrightarrow{\text{BCE}} \mathcal{L}$$

That's why we need to use the Chain Rule to calculate $\frac{\partial L}{\partial a_1}$ through these above steps, where we use the previous layer's error $\frac{\partial L}{\partial z_2}$ i.e. δ_2, as well as the connection weight $\frac{\partial z_2}{\partial a_1}$ i.e. $W2$.

$$\begin{aligned} \frac{\partial L}{\partial a_1} \ &= \ \frac{\partial L}{\partial z_2} \cdot \frac{\partial z_2}{\partial a_1} \\ &= \ \delta_2 \ \cdot W2 \end{aligned}$$

Recall $z_2 \ = \ a_1 W2 \ + \ b_2$, therefore we get $\frac{\partial z_2}{\partial a_1}$ by simple rules of differentiation.

$$\frac{\partial a_1}{\partial z_1} \ = \ \sigma'(z_1) \ = \ a_1(1 - a_1)$$

Therefore,

$$\delta_1 \ = \left(\delta_2 \ \cdot W2^T \right) \ \odot \ \sigma'(z_1)$$

Why take the transpose? Because δ_2 has shape $(4, 1)$ and $W2$ has shape $(2, 1)$, and therefore if we take the tranpose you can now take the dot product of the two matrices by rule of matrix multiplication.

Why dot product for one and element wise product $\odot$ for the other?
Dot product is used for **propagation** across layers, i.e. between Layer 1 and Layer 2, the weighted influence that layer 1 neurons have on layer 2.
Element-wise product is used for **local scaling** within a layer -- i.e. applying the derivative of the activation function.

Same logic as the backpropagation logic for $W2$. For any weight, $\frac{\partial L}{\partial W_i} \ = \ a_{L-1}^T \ \cdot \delta_L$, where L in the subscript indicates the layer.

$$\begin{aligned} \frac{\partial L}{\partial W1} \ &= \frac{\partial L}{\partial a_1} \cdot \ \frac{\partial a_1}{\partial z_1} \cdot \frac{\partial z_1}{\partial W1} \\ &= \ x^T \cdot \delta_1 \end{aligned}$$