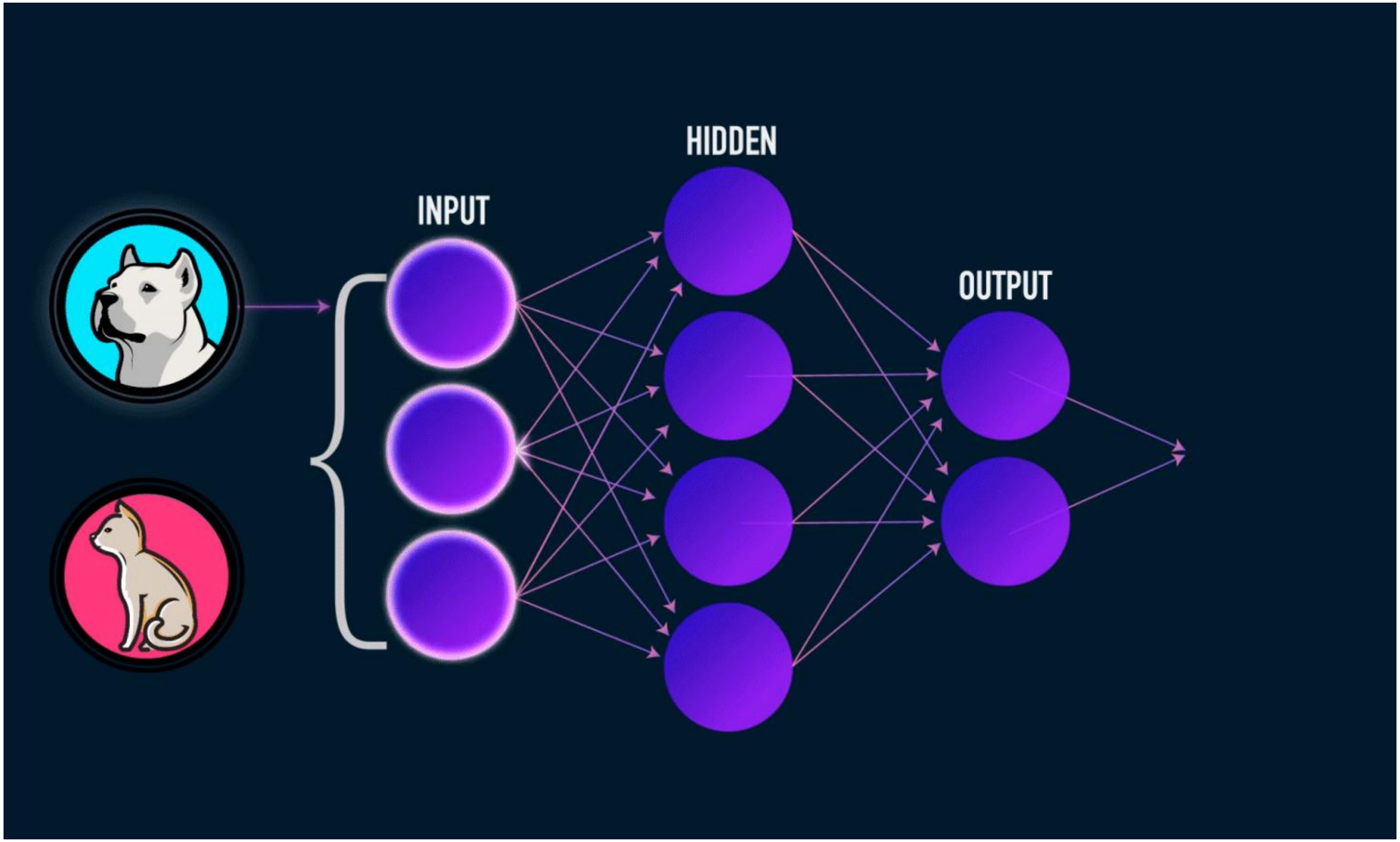
# MLP Math

Simple worked example

# Neural Networks



# What is a Neural Layer?

- A layer mixes the input numbers using weights.
- Each output is: weighted sum of inputs + bias.
- Using matrices makes this efficient and compact.

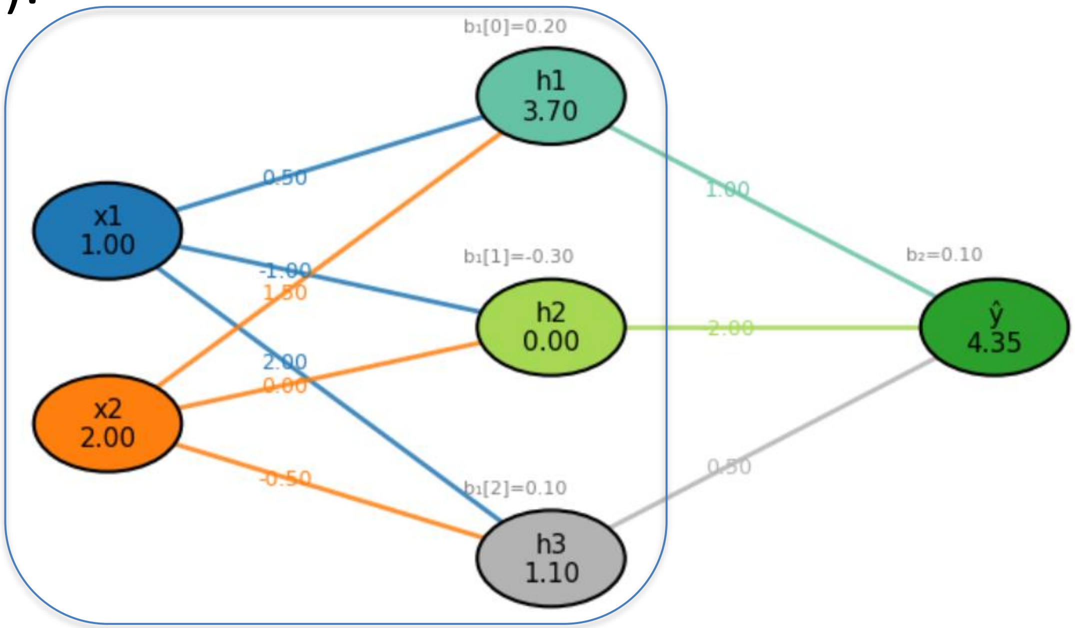
# Matrix Form of a Layer

- Formula: $Z = XW + b$

- X: inputs ($N \times d$)
- W: weights ($d \times H$)
- b: bias added to each row ($1 \times H$)
- Z: output before activation ($N \times H$)

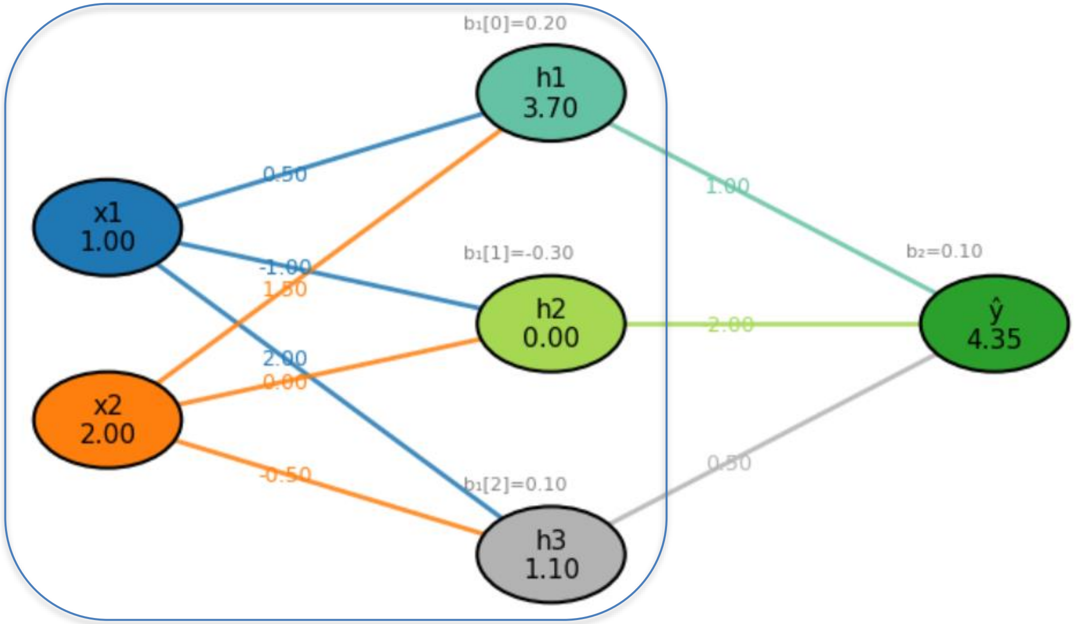
# Example — Input Matrix

- Suppose we have 1 samples with 2 features:
- === INPUT ===
- X (1 sample, 2 features):
- [[1. 2.]]



# Example — Weight Matrix

- Let the weight matrix be:
- === LAYER 1 PARAMETERS ===
- W1 (weights from 2 inputs → 3 hidden neurons):
- [[ 0.5 -1.   2. ]
-  [ 1.5  0.  -0.5]]



# Matrix Multiplication Step

- Compute XW: (matrix multiplication)

| | |
|---|---|
| **1** | **2** |

Dimension: 1*2

| | | |
|---|---|---|
| 0.5 | -1 | 2 |
| 1.5 | 0 | -0.5 |

Dimension: 2*3

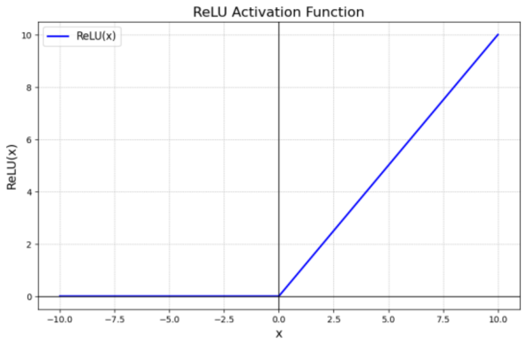
XW =

| | | |
|---|---|---|
| **3.5** | **-1** | **1** |

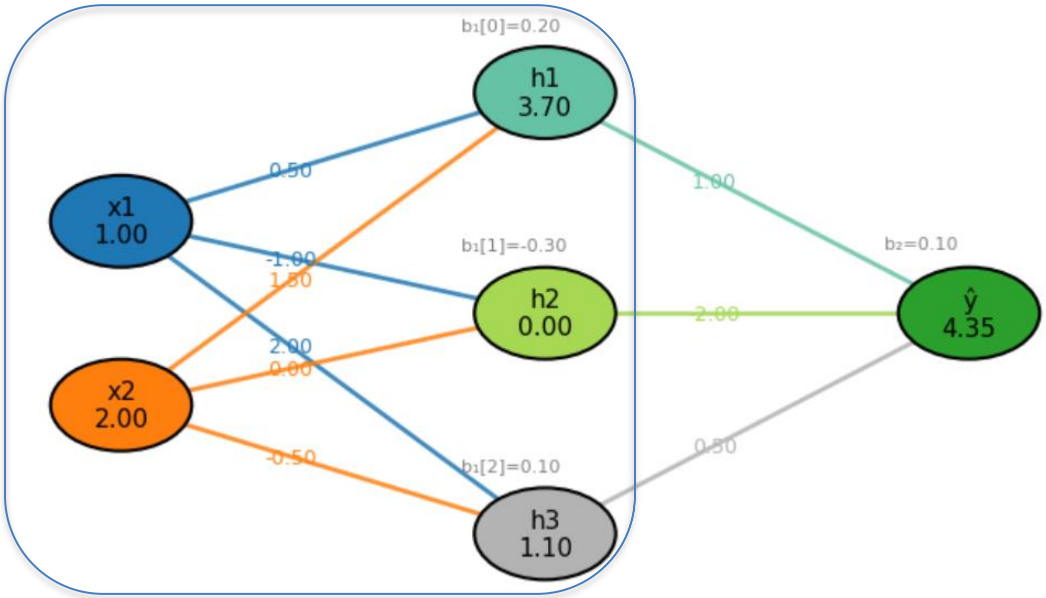
# Add Bias

- b1 (bias for each of the 3 hidden neurons):
- [[ 0.2 -0.3  0.1]]
- Z = XW + b (weighted sums before activation)
- === LAYER 1: LINEAR COMBINATION ===
- [[ 3.7 -1.3  1.1]]

# Activation: ReLU

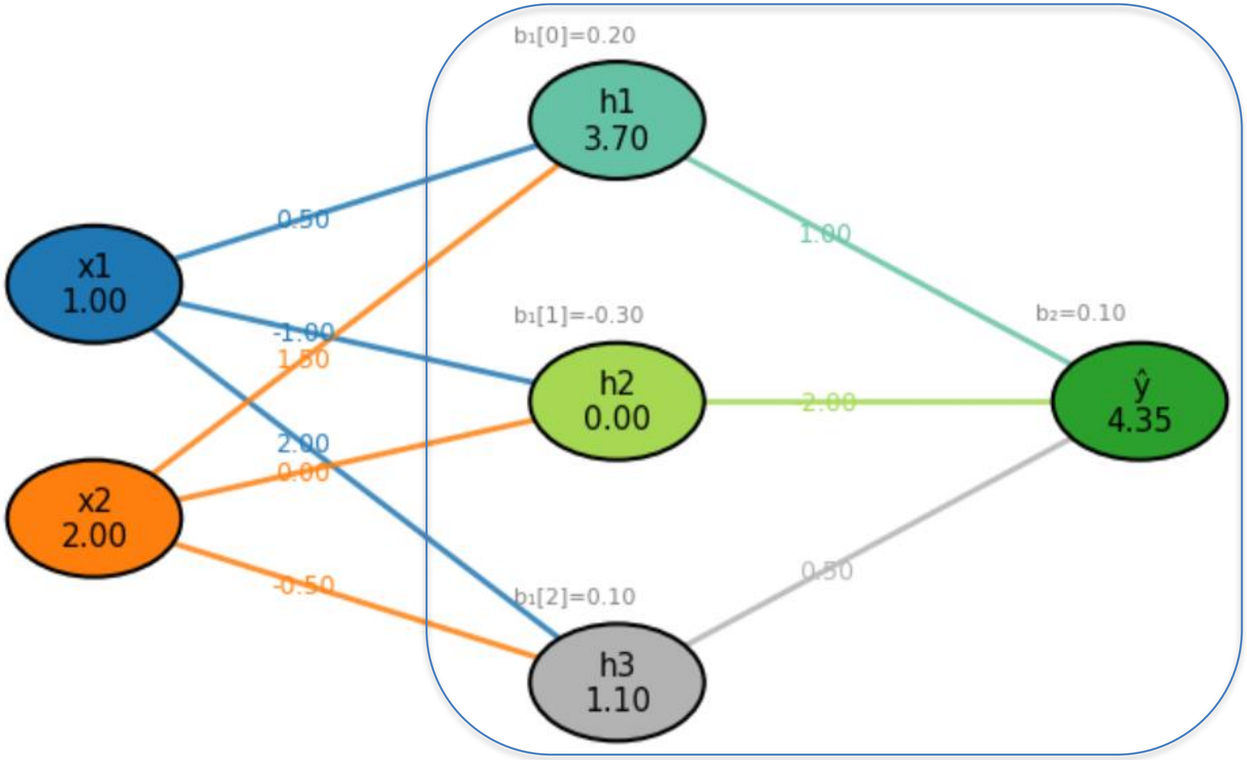



- ReLU(x) = max(0, x)
- This produces the hidden layer output.
- === LAYER 1: ACTIVATION ===
- H1 = ReLU(Z1)
- [[3.7  0.  1.1]]



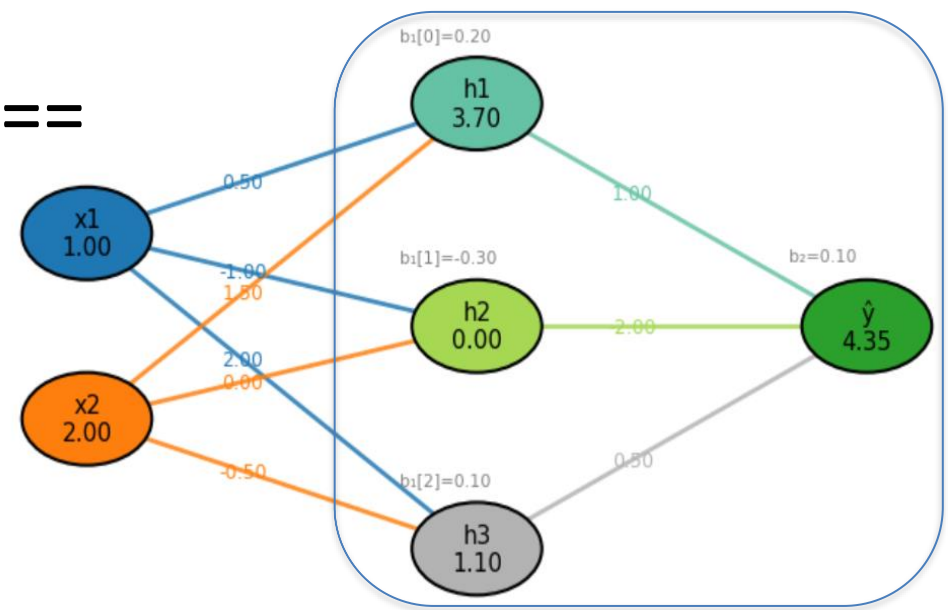
# Layer 2

- W2 (weights from 3 hidden neurons → 1 output):
- [[ 1. ] [-2. ] [ 0.5]]
- b2 (output bias):
- [[0.1]]



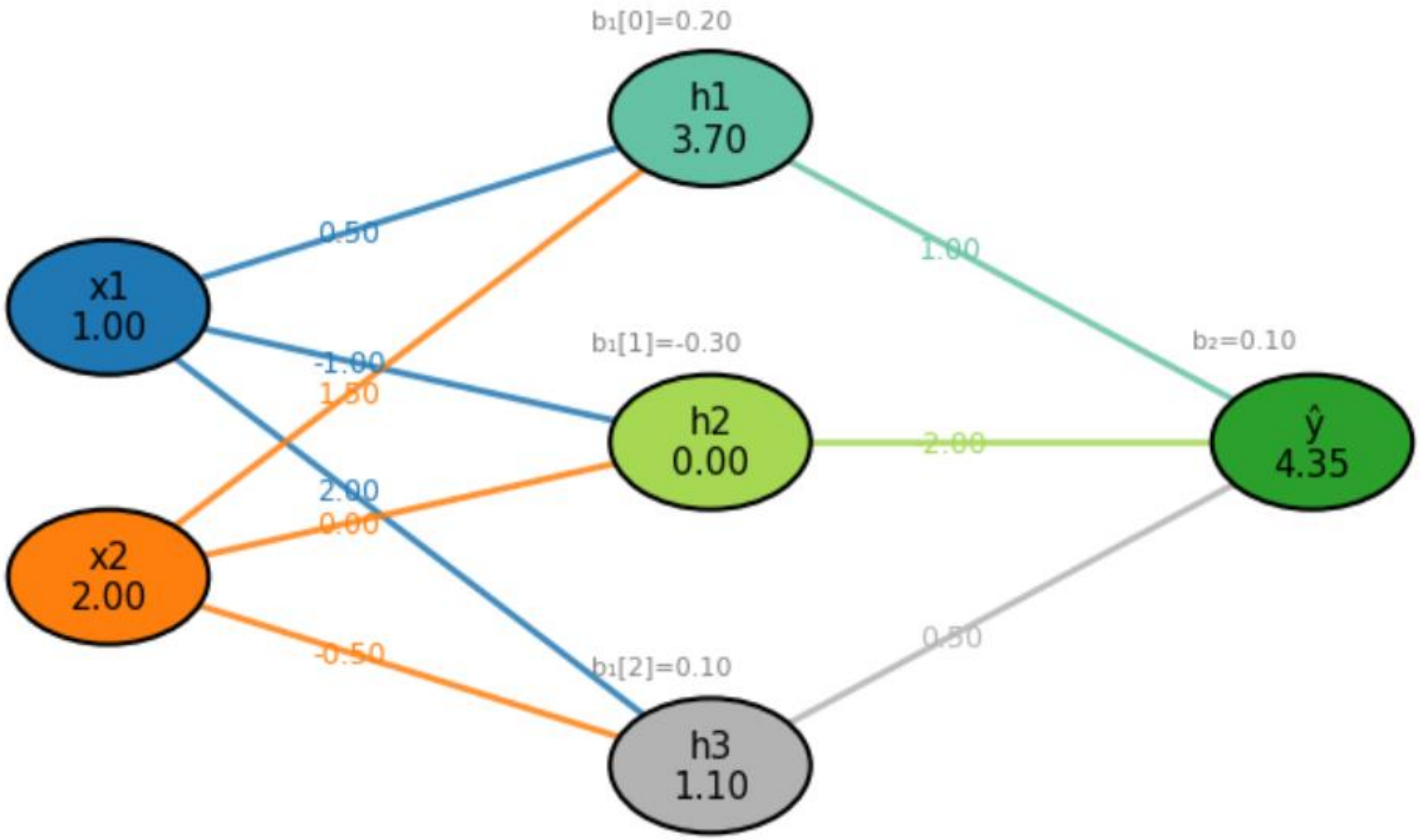
# Layer 2 activation function

- Linea activation function f(x) = x
- === OUTPUT LAYER ===
- Z2 = H1.W2 + b2 (final prediction before activation):
- [[4.35]]
- === FINAL OUTPUT ===
- Predicted y value:
- [[4.35]]



# Final Regression Result

- X = [[1, 2]] → y_hat = 4.35



# Backpropagation

Step-by-step

# What is Backpropagation?

- A method to find how much each weight contributed to the error.
- Goal: reduce the error by adjusting weights.
- Flow: output → error → send error backward.

# Step 1 – Forward Pass

- Inputs move through the network.
- Each neuron computes: (inputs × weights) + bias.
- Activation function is applied.
- We get the prediction ŷ.

# Step 2 – Compute the Error (Loss)

- Compare network output ($\hat{y}$) with target ($y$).
- Loss = how wrong the network is.
- Example: MSE = $(\hat{y} - y)^2$.

# Step 3 – Delta at Output Layer

- Delta = how much the output neuron caused the error.
- Comes from derivative of the loss.
- This signal goes backward.

# Step 4 – Gradients for Output Weights

- Core rule: gradient = (error delta) × (input value).
- Large neuron value → large influence → larger update.
- Small or zero neuron value → small influence.

# Step 5 – Send Error Back to Hidden Layer

- Output delta is multiplied by connecting weights.
- Each hidden neuron receives a portion of the error.
- This becomes its delta.

# Step 6 – Activation Derivative

- Hidden neurons apply derivative of activation function.
- ReLU derivative: 1 for positive input, 0 for negative.
- If derivative = 0 → neuron gets no error → no update.

# Step 7 – Gradients for First-Layer Weights

- Same rule: gradient = input × delta of hidden neuron.
- Shows how much each weight contributed to final error.

# Step 8 – Bias Gradients

- Bias has constant input = 1.
- So gradient of bias = delta of that neuron.

# Step 9 – Weight Update

- Weights change to reduce error:
- new_weight = old_weight − learning_rate × gradient.
- Repeating this trains the network.

# Backprop Summary

- Forward: compute outputs.
- Loss: compare with target.
- Backward: compute deltas.
- Gradients = input × delta.
- Update weights to reduce error.
- This is how neural networks learn.

# Forward pass

## 1) Forward pass (with one sample)

**Given**

$X = [1,\ 2]$

$$W_1 = \begin{bmatrix} 0.5 & -1 & 2 \\ 1.5 & 0 & -0.5 \end{bmatrix}, \quad b_1 = [0.2,\ -0.3,\ 0.1]$$

$$W_2 = \begin{bmatrix} 1 \\ -2 \\ 0.5 \end{bmatrix}, \quad b_2 = [0.1]$$

$y = 3.5$

**Hidden pre-activation** $z_1 = XW_1 + b_1$

- First hidden: $1 \cdot 0.5 + 2 \cdot 1.5 + 0.2 = 0.5 + 3 + 0.2 = 3.7$
- Second hidden: $1 \cdot (-1) + 2 \cdot 0 + (-0.3) = -1 - 0.3 = -1.3$
- Third hidden: $1 \cdot 2 + 2 \cdot (-0.5) + 0.1 = 2 - 1 + 0.1 = 1.1$

So $z_1 = [3.7,\ -1.3,\ 1.1]$

**Hidden activation** $h_1 = \mathrm{ReLU}(z_1)$

$\Rightarrow h_1 = [3.7,\ 0,\ 1.1]$

**Output** $z_2 = h_1^\top W_2 + b_2$

$= 3.7 \cdot 1 + 0 \cdot (-2) + 1.1 \cdot 0.5 + 0.1 = 3.7 + 0 + 0.55 + 0.1 = 4.35$

Prediction $\hat{y} = 4.35$

# Backward pass - error

MSE = mean of squared errors

## 2) Loss (MSE)

With one sample, $L = (\hat{y} - y)^2 = (4.35 - 3.5)^2 = 0.85^2 = \boxed{0.7225}$

# Backward pass – gradient w2

## 3) Output gradient

For MSE with $\hat{y} = z_2$:

$$\frac{\partial L}{\partial \hat{y}} = 2(\hat{y} - y) = 2 \cdot 0.85 = \boxed{1.7}$$

Since $\hat{y} = z_2$, $\frac{\partial L}{\partial z_2} = \boxed{1.7}$.

Bias gradient at output: $\boxed{db_2 = 1.7}$

Weights to output:

$dW_2 = h_1^\top \cdot \frac{\partial L}{\partial z_2}$ (outer product with a scalar)

$$dW_2 = \begin{bmatrix} 3.7 \\ 0 \\ 1.1 \end{bmatrix} \cdot 1.7 = \boxed{\begin{bmatrix} 6.29 \\ 0 \\ 1.87 \end{bmatrix}}$$

# Backpropagation to hidden

## 4) Backprop to hidden

First, push error back through $W_2$:

$$dH_1 = \frac{\partial L}{\partial z_2} W_2^\top = 1.7\,[\,1,\ -2,\ 0.5\,] = [\,1.7,\ -3.4,\ 0.85\,]$$

Apply ReLU derivative (1 if $z_1 > 0$, else 0):

$$z_1 = [3.7, -1.3, 1.1] \Rightarrow \mathrm{ReLU}' = [1, 0, 1]$$

Hidden deltas:

$$\delta_1 = dZ_1 = dH_1 \odot \mathrm{ReLU}' = [\,1.7,\ 0,\ 0.85\,]$$

Bias gradients at hidden (mean over batch of size 1):

$$\boxed{db_1 = [\,1.7,\ 0,\ 0.85\,]}$$

# Backward pass – gradient w1

## 5) Gradients for $W_1$

$dW_1 = X^\top \cdot \delta_1$ (with $X = [1, 2]$)

Row 1 (from input $x_1 = 1$): $1 \cdot [1.7,\ 0,\ 0.85] = [1.7,\ 0,\ 0.85]$

Row 2 (from input $x_2 = 2$): $2 \cdot [1.7,\ 0,\ 0.85] = [3.4,\ 0,\ 1.7]$

$$\boxed{dW_1 = \begin{bmatrix} 1.7 & 0 & 0.85 \\ 3.4 & 0 & 1.7 \end{bmatrix}}$$

# Final Results

## 6) Final collected results

- **Loss (MSE):** $\boxed{0.7225}$
- $dW_1$: $\boxed{\begin{bmatrix} 1.7 & 0 & 0.85 \\ 3.4 & 0 & 1.7 \end{bmatrix}}$
- $dW_2$: $\boxed{\begin{bmatrix} 6.29 \\ 0 \\ 1.87 \end{bmatrix}}$
- $db_1$: $\boxed{[\,1.7,\ 0,\ 0.85\,]}$
- $db_2$: $\boxed{[\,1.7\,]}$

# Why Matrix Multiplication?

- Processes many samples at once.
- More efficient than doing each weighted sum manually.
- Matches how real neural networks run on GPUs/CPUs.