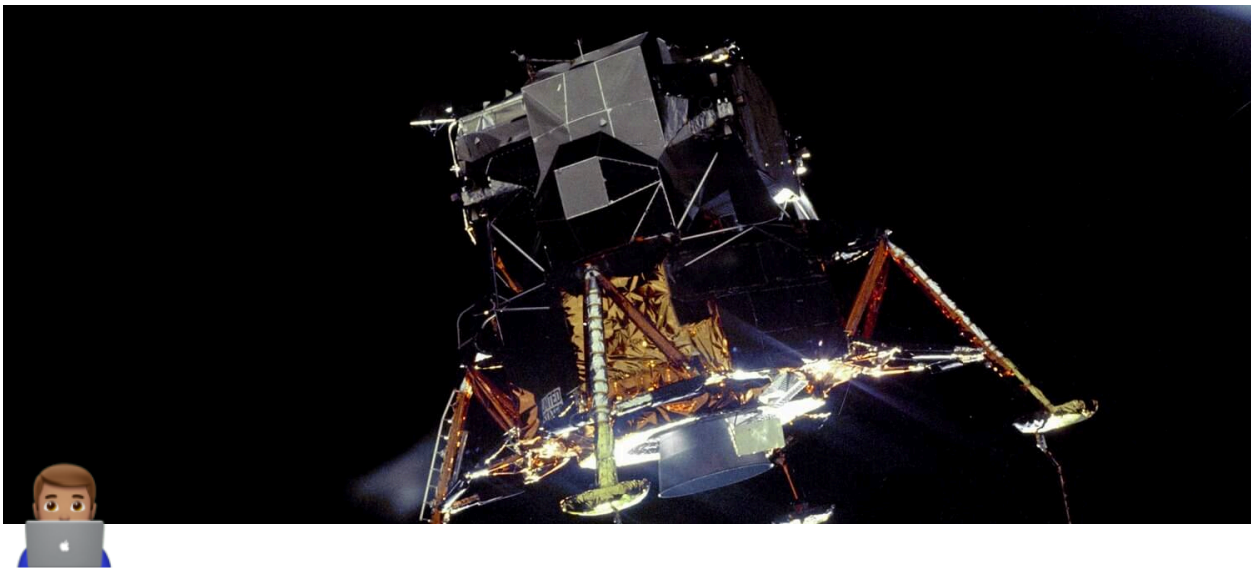
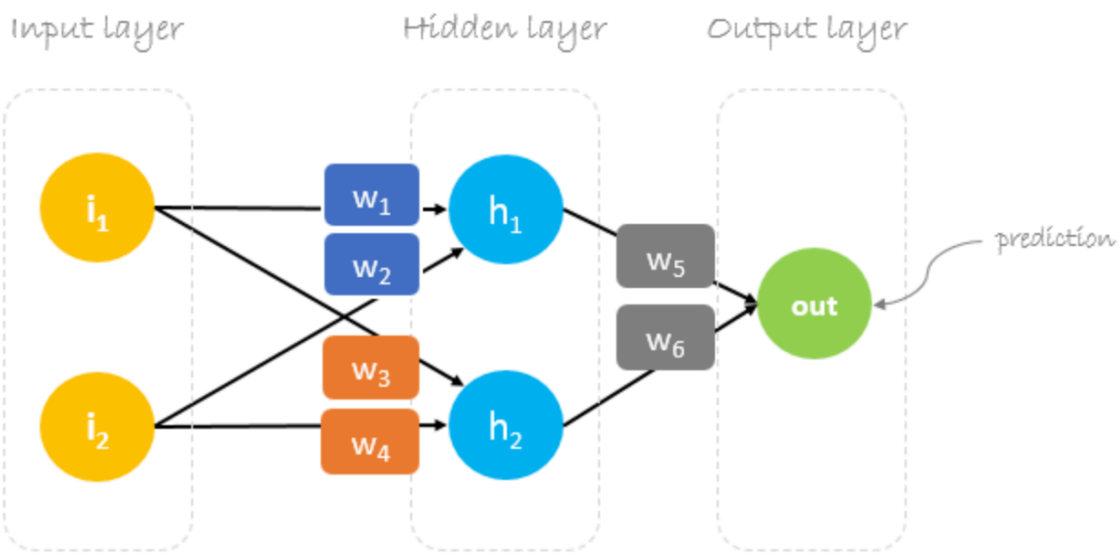


Forward & Backpropagation

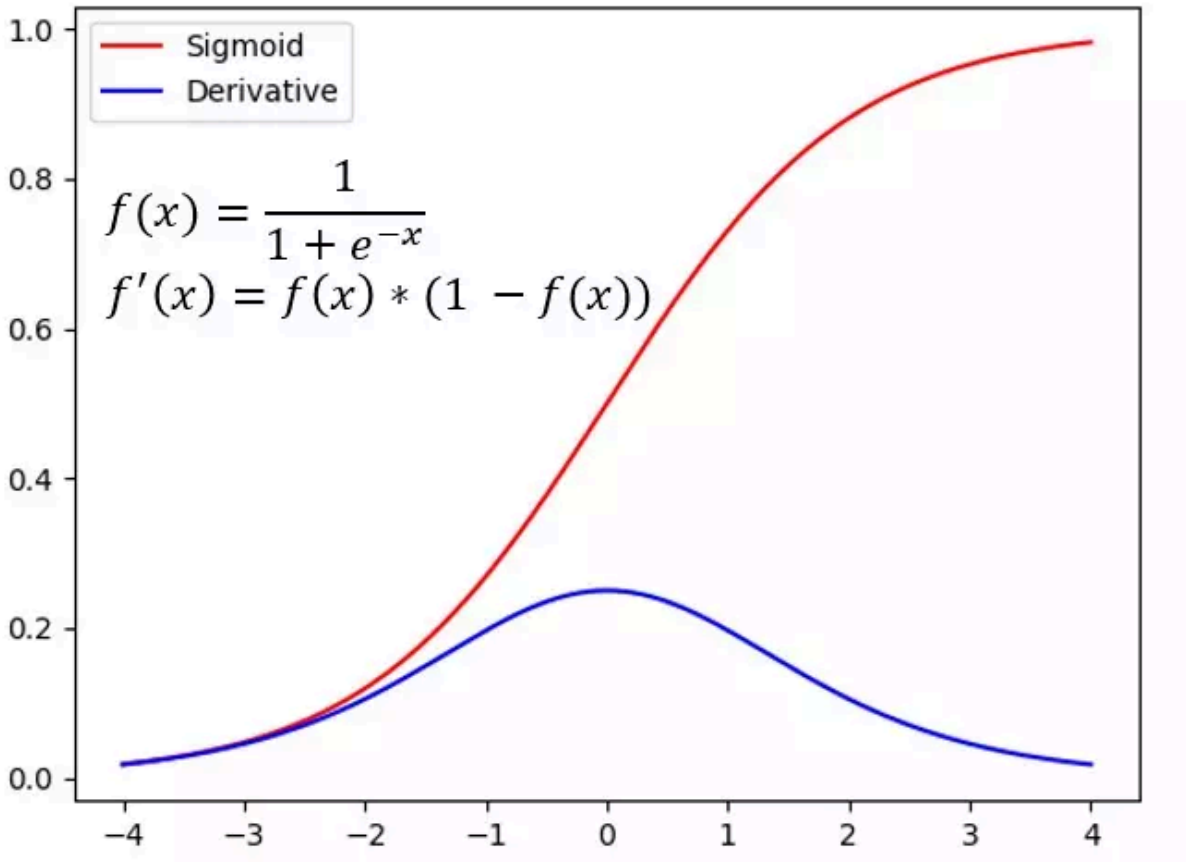
Network Architecture

- **Input Layer**: 2 neurons → x1, x2
- **Hidden Layer**: 2 neurons → h1, h2
- **Output Layer**: 1 neuron → y_hat



We'll use the **sigmoid activation function** for both layers:

```
sigmoid(z) = 1 / (1 + exp(-z))
```



1. Initial Values

1.1 Inputs:

x1 = 0.05
x2 = 0.10

1.2 Weights:

Input to Hidden Layer:

w1 = 0.15 (x1 → h1)
w2 = 0.20 (x2 → h1)
w3 = 0.25 (x1 → h2)
w4 = 0.30 (x2 → h2)

Hidden to Output Layer:

w5 = 0.40 (h1 → output)
w6 = 0.45 (h2 → output)

1.3 Biases:

`b1 = 0.35 (for hidden layer neurons)`
`b2 = 0.60 (for output layer)`

Bias helps the neural network make better decisions, even when the input alone isn't strong enough. It gives the model more flexibility.

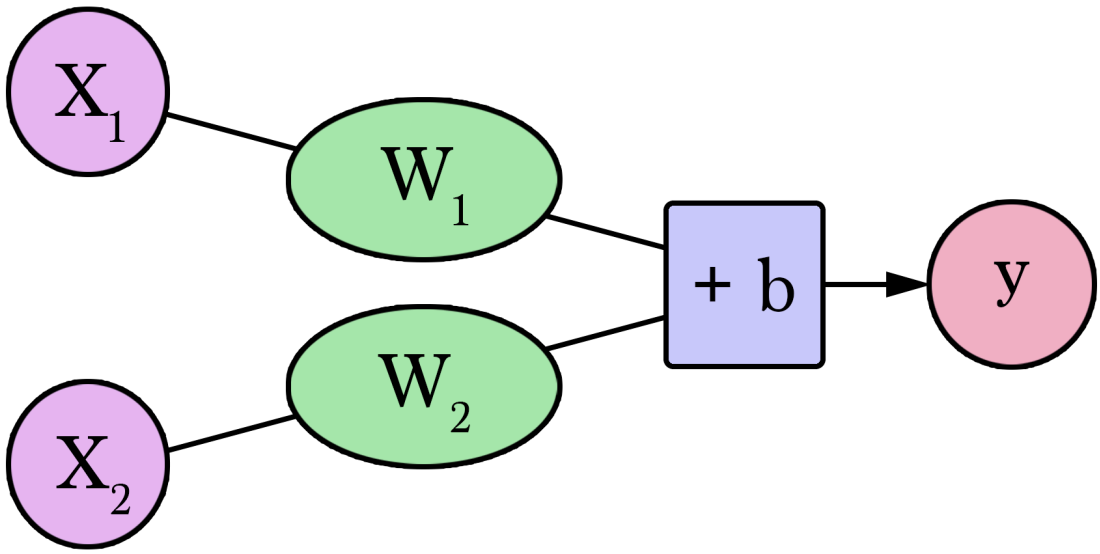
*Think of it like the **intercept term in linear regression** (y = mx + b).*

Actual Target Output:

`y_true = 0.01`

---

2. Forward Propagation



Step 1: Hidden Layer

Calculate net input to each hidden neuron:

z_h1 = x1 *w1 + x2* w2 + b1 = 0.05 *0.15 + 0.10* 0.20 + 0.35 = 0.3775
z_h2 = x1 *w3 + x2* w4 + b1 = 0.05 *0.25 + 0.10* 0.30 + 0.35 = 0.3925

$$y = W x + b$$

$$y = W_1 x_1 + W_2 x_2 + b$$

Apply sigmoid activation:

a_h1 = sigmoid(0.3775) ≈ 0.5933
a_h2 = sigmoid(0.3925) ≈ 0.5969

Sigmoid Calculator

https://www.tinkershop.net/ml/sigmoid_calculator.html

Step 2: Output Layer

z_out = a_h1 *w5 + a_h2* w6 + b2 = 0.5933 *0.40 + 0.5969* 0.45 + 0.60 ≈ 1.1059
a_out = sigmoid(1.1059) ≈ 0.7514

Predicted Output (y_hat) = 0.7514

Compute Loss (Error)

We'll use Mean Squared Error (MSE):

```
Loss = 0.5 * (y_true - y_hat)^2
Loss = 0.5 * (0.01 - 0.7514)^2 ≈ 0.2748
```

---

3. Backward Propagation

Backpropagation, short for "backward propagation of errors", is a mechanism used to update the **weights** using gradient descent. It calculates the gradient of the error function with respect to the neural network's weights.

> Gradient descent is an iterative optimization algorithm for finding the minimum of a function; in our case we want to minimize th error function. To find a local minimum of a function using gradient descent, one takes steps proportional to the negative of the gradient of the function at the current point.

We now compute gradients of the loss with respect to each weight, using chain rule.

Step-by-step Gradient for w5

Let's compute the gradient for w5:

$$\frac{\partial Loss}{\partial w5} = \frac{\partial Loss}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z_{out}} \cdot \frac{\partial z_{out}}{\partial w5}$$

w5 connects: h1 (activation a_h1) → output neuron

∂Loss/∂y_hat — Derivative of Loss with respect to predicted output

```
dL/dy_hat = (y_hat - y_true) = 0.7514 - 0.01 = 0.7414
```

Loss = 0.5 * (y_true - y_hat)^2

∂y_hat/∂z_out — Derivative of the output activation with respect to the weighted sum z_out

sigmoid(z) = 1 / (1 + exp(-z))

Derivative of sigmoid:

sigmoid'(z) = sigmoid(z) * (1 - sigmoid(z)) = y_hat * (1 - y_hat)

```
dy_hat/dz_out = y_hat * (1 - y_hat) ≈ 0.7514 * 0.2486 ≈ 0.1868
```

∂z_out/∂w5 — Derivative of the output neuron's weighted sum with respect to w5

The output neuron computes: z_out = w5 * a_h1 + w6 * a_h2 + b2

Applying this rule from basic calculus

$$\frac{d}{dx}(x \cdot c) = c \quad (\text{for constant } c)$$

∂z_out/∂w5 = a_h1

```
dz_out/dw5 = a_h1 = 0.5933
```

Finally

```
dL/dw5 = dL/dy_hat * dy_hat/dz_out * dz_out/dw5
dL/dw5 ≈ 0.7414 * 0.1868 * 0.5933 ≈ 0.0822
```

$$\frac{\partial Loss}{\partial w5} = \frac{\partial Loss}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z_{out}} \cdot \frac{\partial z_{out}}{\partial w5}$$

Here are the derivatives:

$$\frac{\partial L}{\partial \hat{y}} = (\hat{y} - y_{\text{true}}) = 0.7514 - 0.01 = 0.7414$$
$$\frac{\partial \hat{y}}{\partial z_{\text{out}}} = \hat{y} \cdot (1 - \hat{y}) \approx 0.7514 \cdot 0.2486 \approx 0.1868$$
$$\frac{\partial z_{\text{out}}}{\partial w_5} = a_{h1} = 0.5933$$
$$\frac{\partial L}{\partial w_5} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z_{\text{out}}} \cdot \frac{\partial z_{\text{out}}}{\partial w_5} \approx 0.7414 \cdot 0.1868 \cdot 0.5933 \approx 0.0822$$

Step-by-step Gradient for w6

$$\frac{\partial Loss}{\partial w6} = \frac{\partial Loss}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z_{out}} \cdot \frac{\partial z_{out}}{\partial w6}$$

dz_out/dw6 = a_h2 = 0.5969
dL/dw6 ≈ 0.7414 * 0.1868 * 0.5969 ≈ 0.0827

$$dz_out/dw6 = a_h2 = 0.5969$$
$$dL/dw6 \approx 0.7414 * 0.1868 * 0.5969 \approx 0.0827$$

Step-by-step Gradient for w1

Hidden to Input Layer Gradients

Now we backpropagate the error from output to hidden neurons.

$$\frac{\partial Loss}{\partial w1} = \frac{\partial Loss}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z_{out}} \cdot \frac{\partial z_{out}}{\partial a_{h1}} \cdot \frac{\partial a_{h1}}{\partial z_{h1}} \cdot \frac{\partial z_{h1}}{\partial w1}$$

y_hat = 0.7514
y_true = 0.01

1. ∂Loss/∂y_hat = 0.7514 - 0.01 = 0.7414
2. **∂y_hat/∂z_out = y_hat * (1 - y_hat)**

Because the output neuron uses **sigmoid activation**:

= 0.7514 * (1 - 0.7514) ≈ 0.7514 ∗ 0.2486 ≈ 0.1868

3. **∂z_out/∂a_h1 = w5**

   From the output neuron:

   z_out = w5 * a_h1 + w6 * a_h2 + b2

   ⇒ ∂z_out/∂a_h1 = w5 = 0.40

4. **∂a_h1/∂z_h1 = a_h1 * (1 - a_h1)**

Because a_h1 also uses sigmoid:

a_h1 = sigmoid(z_h1) = 0.5933

∂a_h1/∂z_h1 = 0.5933 * (1 - 0.5933) ≈ 0.5933 ∗ 0.4067 ≈ 0.2413

5. **∂z_h1/∂w1 = x1**

z_h1 = w1 * x1 + w2 * x2 + b1

⇒ ∂z_h1/∂w1 = x1 = 0.05

Step-by-step:

- 0.7414 ∗ 0.1868 ≈ 0.1385
- 0.1385 ∗ 0.40 ≈ 0.0554
- 0.0554 ∗ 0.2413 ≈ 0.0134
- 0.0134 ∗ 0.05 ≈ **0.00067**

Finally

$$\frac{\partial Loss}{\partial w1} = \frac{\partial Loss}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z_{out}} \cdot \frac{\partial z_{out}}{\partial a_{h1}} \cdot \frac{\partial a_{h1}}{\partial z_{h1}} \cdot \frac{\partial z_{h1}}{\partial w1}$$

∂Loss/∂w1 ≈ 0.00067

Step-by-step Gradient for w2

$$\frac{\partial Loss}{\partial w2} = \frac{\partial Loss}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z_{out}} \cdot \frac{\partial z_{out}}{\partial a_{h1}} \cdot \frac{\partial a_{h1}}{\partial z_{h1}} \cdot \frac{\partial z_{h1}}{\partial w2}$$

dz_h1/dw2 = x2 = 0.10
dL/dw2 = 0.0555 * 0.2413 * 0.10 ≈ 0.00135

Step-by-step:

- 0.7414 * 0.1868 ≈ 0.1385
- 0.1385 * 0.40 ≈ 0.0554
- 0.0554 * 0.2413 ≈ 0.0134
- 0.0134 * 0.10 ≈ **0.00135**

∂Loss/∂w2 ≈ 0.00135

Step-by-step Gradient for w4

$$\frac{\partial Loss}{\partial w4} = \frac{\partial Loss}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z_{out}} \cdot \frac{\partial z_{out}}{\partial a_{h2}} \cdot \frac{\partial a_{h2}}{\partial z_{h2}} \cdot \frac{\partial z_{h2}}{\partial w4}$$

1. **∂Loss/∂y_hat = y_hat - y_true**

y_hat = 0.7514
y_true = 0.01

∂Loss/∂y_hat = 0.7514 - 0.01 = 0.7414

2. ∂y_hat/∂z_out = y_hat * (1 - y_hat)
= 0.7514 * (1 - 0.7514) ≈ 0.7514 * 0.2486 ≈ 0.1868

3. ∂z_out/∂a_h2 = w6
w6 = 0.45

4. ∂a_h2/∂z_h2 = a_h2 * (1 - a_h2)

a_h2 ≈ 0.5969

∂a_h2/∂z_h2 = 0.5969 * (1 - 0.5969) ≈ 0.5969 * 0.4031 ≈ 0.2407

5. ∂z_h2/∂w4 = x2
x2 = 0.10

Finally

∂Loss/∂w4 = 0.7414 * 0.1868 * 0.45 * 0.2407 * 0.10

$$\frac{\partial Loss}{\partial w4} = \frac{\partial Loss}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z_{out}} \cdot \frac{\partial z_{out}}{\partial a_{h2}} \cdot \frac{\partial a_{h2}}{\partial z_{h2}} \cdot \frac{\partial z_{h2}}{\partial w4}$$

Step-by-step:

- 0.7414 * 0.1868 ≈ 0.1385
- 0.1385 * 0.45 ≈ 0.0623
- 0.0623 * 0.2407 ≈ 0.0150
- 0.0150 * 0.10 ≈ **0.00150**

Step-by-step Gradient for w3

$$\frac{\partial Loss}{\partial w3} = \frac{\partial Loss}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z_{out}} \cdot \frac{\partial z_{out}}{\partial a_{h2}} \cdot \frac{\partial a_{h2}}{\partial z_{h2}} \cdot \frac{\partial z_{h2}}{\partial w3}$$

∂z_h2/∂w3 = x1
x1 = 0.05

∂Loss/∂w3 = 0.7414 * 0.1868 * 0.45 * 0.2407 * 0.05

Step-by-step:

- 0.7414 * 0.1868 ≈ 0.1385
- 0.1385 * 0.45 ≈ 0.0623
- 0.0623 * 0.2407 ≈ 0.0150

- 0.0150 * 0.05 ≈ **0.00075**

∂Loss/∂w3 ≈ 0.00075

ALL Gradient Calculation Completed

Summary of Gradients for Hidden Layer

Weight	Gradient (dL/dw)
w1	≈ 0.00067
w2	≈ 0.00135
w3	≈ 0.00075
w4	≈ 0.00150
w5	≈ 0.0822
w6	≈ 0.0827

Bias Gradient Calculation

1. For Output Bias (b2)

Bias behaves like a weight connected to a constant input of 1.

b2 is used in the output neuron's computation:

z_out = w5 * a_h1 + w6 * a_h2 + b2

$$\frac{\partial Loss}{\partial b2} = \frac{\partial Loss}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z_{out}} \cdot \frac{\partial z_{out}}{\partial b2}$$

∂Loss/∂y_hat = y_hat - y_true = 0.7514 - 0.01 = 0.7414

∂y_hat/∂z_out = y_hat * (1 - y_hat) = 0.7514 * 0.2486 ≈ 0.1868

Since b2 is **added directly** to z_out, the derivative is:

∂z_out/∂b2 = 1

Combine all:

∂Loss/∂b2 = 0.7414 * 0.1868 * 1 = **0.1385**

∂Loss/∂b2 ≈ 0.1385

2. For Hidden Bias (b1)

Since both h1 and h2 use the same bias b1, we need to compute the gradients for both neurons separately, and add them.

• b1 is added into the calculation of:

z_h1 = w1 * x1 + w2 * x2 + b1
z_h2 = w3 * x1 + w4 * x2 + b1

So we compute:

∂Loss/∂b1 = ∂Loss/∂b1_from_h1 + ∂Loss/∂b1_from_h2

$$\frac{\partial Loss}{\partial b1} = \frac{\partial Loss}{\partial b1\_from\_h1} + \frac{\partial Loss}{\partial b1\_from\_h2}$$

Part 1: Gradient from h1

$$\frac{\partial Loss}{\partial b1\_from\_h1} = \frac{\partial Loss}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z_{out}} \cdot \frac{\partial z_{out}}{\partial a_{h1}} \cdot \frac{\partial a_{h1}}{\partial z_{h1}} \cdot \frac{\partial z_{h1}}{\partial b1}$$

Known values:

- ∂Loss/∂y_hat = 0.7414
- ∂y_hat/∂z_out = 0.1868
- ∂z_out/∂a_h1 = w5 = 0.40
- a_h1 = 0.5933 → ∂a_h1/∂z_h1 = 0.5933 * (1 - 0.5933) ≈ 0.2413

- ∂z_h1/∂b1 = 1 (because b1 is added directly)

∂Loss/∂b1_from_h1 = 0.7414 * 0.1868 * 0.40 * 0.2413 * 1 ≈ 0.0134

Part 2: Gradient from h2

$$\frac{\partial Loss}{\partial b1\_from\_h2} = \frac{\partial Loss}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z_{out}} \cdot \frac{\partial z_{out}}{\partial a_{h2}} \cdot \frac{\partial a_{h2}}{\partial z_{h2}} \cdot \frac{\partial z_{h2}}{\partial b1}$$

Known values:

- ∂z_out/∂a_h2 = w6 = 0.45
- a_h2 = 0.5969 → ∂a_h2/∂z_h2 = 0.5969 * (1 - 0.5969) ≈ 0.2407
- ∂z_h2/∂b1 = 1

∂Loss/∂b1_from_h2 = 0.7414 * 0.1868 * 0.45 * 0.2407 * 1 ≈ 0.0150

Total Gradient for b1

∂Loss/∂b1 = 0.0134 + 0.0150 ≈ 0.0284

∂Loss/∂b1 ≈ 0.0284

Weight and Bias Updates

Weight Updation Formula

`new_weight = old_weight - learning_rate * gradient`

Let’s assume learning rate = 0.5

w5_new = 0.40 - 0.5 * 0.0822 ≈ 0.3589
w6_new = 0.45 - 0.5 * 0.0827 ≈ 0.4087

w4_new = 0.30 - 0.5 * 0.00150 ≈ 0.2993
w3_new = 0.25 - 0.5 * 0.00075 ≈ 0.2496

w2_new = 0.20 - 0.5 * 0.00135 ≈ 0.199325

w1_new = 0.15 - 0.5 * 0.00067 ≈ 0.149665

Bias Updation Formula

`new_bias = old_bias - learning_rate * gradient`

b1_new = b1 - 0.5 * 0.0284 = 0.35 - 0.0142 = 0.3358

Update b2:

b2_new = b2 - α * dL/db2
= 0.60 - 0.5 * 0.1385
= 0.60 - 0.0693
≈ 0.5307

Weight & BIAS Update Tables

Weight	Old Value	Gradient (dL/dw)	New Value
w6	0.45	0.0827	0.40865
w5	0.40	0.0822	0.35890
w4	0.30	0.00150	0.29925
w3	0.25	0.00075	0.249625
w2	0.20	0.00135	0.199325
w1	0.15	0.00067	0.149665

Bias	Old Value	Gradient (dL/db)	New Value
b2	0.600	0.1385	0.5307
b1	0.350	0.0284	0.3358

Summary

- **Forward pass** calculates activations and predictions
- **Loss** is computed comparing prediction to true value
- **Backward pass** propagates the error to compute gradients

- **Update step** adjusts weights and biases to reduce future error

This process is repeated for all training examples over multiple epochs until the model converges.

External Resources

Backpropagation Step by Step

Backpropagation is a technique used for training neural network. There are many resources explaining the technique, but this post will explain backpropagation with concrete example in a very detailed colorful steps.

https://hmkcode.com/ai/backpropagation-step-by-step/

Backprop Explainer

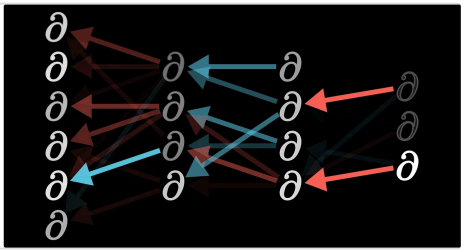
Backpropagation Explainer, an explanation with interactive tools

https://xnought.github.io/backprop-explainer/

3Blue1Brown - Backpropagation calculus

The math of backpropagation, the algorithm by which neural networks learn.

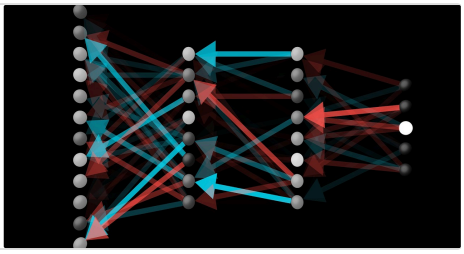
https://www.3blue1brown.com/lessons/backpropagation-calculus



3Blue1Brown - What is backpropagation really doing?

An overview of backpropagation, the algorithm behind how neural networks learn.

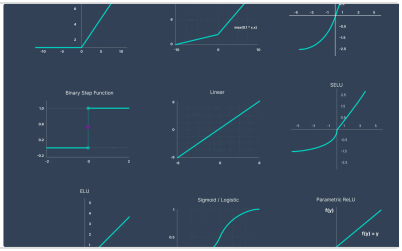
https://www.3blue1brown.com/lessons/backpropagation



Activation Functions in Neural Networks [12 Types & Use Cases]

A neural network activation function is a function that is applied to the output of a neuron. Learn about different types of activation functions and how they work.

https://www.v7labs.com/blog/neural-networks-activation-functions



A Visual and Interactive Guide to the Basics of Neural Networks

Discussions: Hacker News (63 points, 8 comments), Reddit r/programming (312 points, 37 comments) Translations: Arabic, French, Spanish Update: Part 2 is now live: A Visual And Interactive Look at Basic Neural Network Math Motivation I'm not a machine learning expert. I'm a software engineer by training and

https://jalammar.github.io/visual-interactive-guide-basics-neural-networks/

www.iro.umontreal.ca

https://www.iro.umontreal.ca/~vincentp/ift3395/lectures/backprop_old.pdf