BP

2024年9月27日 14:19

Introduction

In simplest terms:

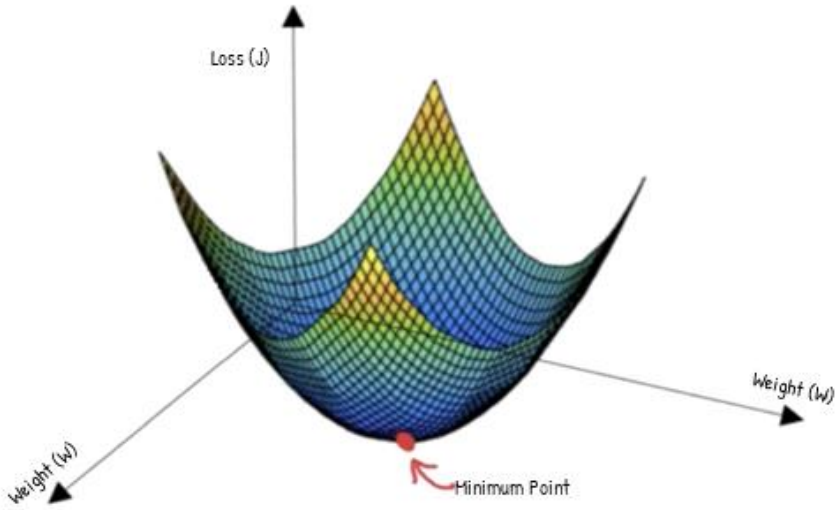
- To understand how sensitive the cost function is to these variables
- To know which adjustments to those terms will cause the most efficient decrease to the function

Cost and Lost Function:

- **Definition:**
  - Loss Function: calculating loss we consider only a single data point
  - Cost Function: calculating the sum of error for multiple data (or an average of the loss functions over an entire training data)
- **Differences:**
  - Loss function is to capture the diff between the actual & predicted values for a single record
  - Cost functions aggregate the difference for the entire training dataset
- **Examples:**
  - Lost Function:
    - MSE: Mean-Squared Error
    - MAE: Mean Absolute Error
  - Cost Function:
    - Average MSE
    - Cross-Entropy Loss

Terms:

- **Gradient Descent (GD)**: focuses on descending the gradient
- **Backpropagation (BP)**: focuses on calculating the gradient
- **Cost Function:** Indicates how accurate the model performs, goal is minimize cost function



How it's related?

Since the weights (of the neural network) affect the error, we will need to readjust the weights. We have to adjust the weights such that we have a combination of weights that minimizes the cost function.

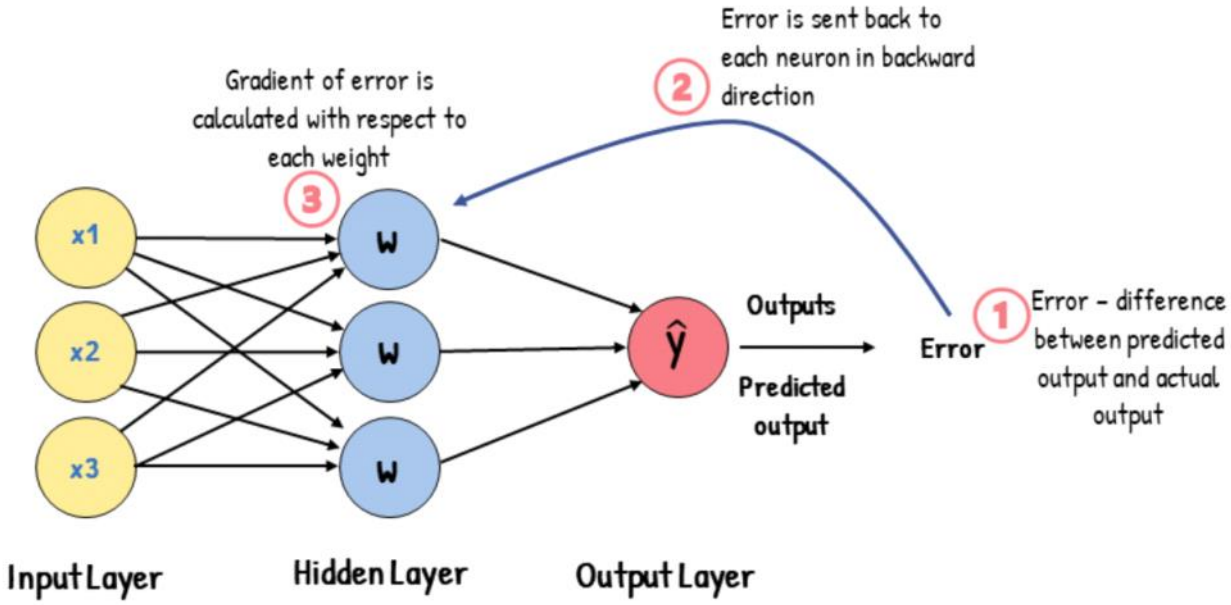
Backpropagation allows us to readjust our weights to reduce output error. The error is propagated backward during backpropagation from the output to the input layer. This error is then used to calculate the gradient of the cost function with respect to each weight.

	Backpropagation	Gradient Descent
Definition	An algorithm for calculating the gradients of the cost function	Optimization algorithm used to find the weights that minimize the cost function
Requirements	Differentiation via the chain rule	•Gradient via Backpropagation •Learning rate
Process	Propagating the error backwards and calculating the gradient of the error function with respect to the weights	Descending down the cost function until the minimum point and find the corresponding weights

Backpropagation

Backpropagation (BP):

- **Intro:** Involves computing gradients of functions using the chain rule (for designing, developing, and debugging neural networks)
- **Core problem:**
  - Compute the gradient of a function $f(x)$ at a given point x, where x is a vector of inputs
  - We want to know, how the changes in the input x, affect the output $f(x)$
- **Motivation:**
  - **Loss function:** $f(x)$ typically represents the loss function L, the inputs x include both training data, network's weights and biases
  - **Training data & weights:** The training data (x_i, y_i) usually fixed, while weights and biases (W, b) can be adjusted
  - **Gradient computation:**
    - BP is used to compute gradient of the loss function, with respect to the weights and biases
    - This gradient is then used to update the parameters to minimize the loss



Simple expressions and interpretation of the gradient

Multiplication function:

- Function: $f(x, y) = xy$
- Partial derivatives:

- $\frac{\partial f}{dx} = y, \quad \frac{\partial f}{dy} = x$

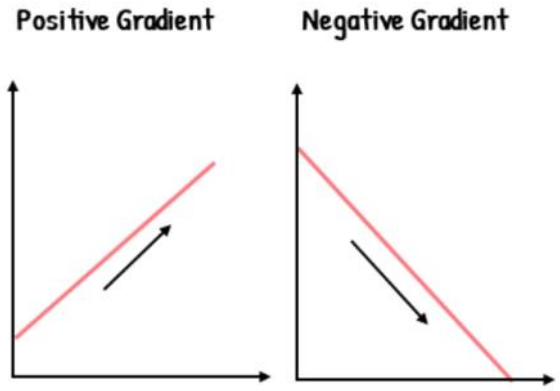
Interpretation of derivatives:

- **Rate of change:** indicate how the function changes with respect to a variable near a particular point

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

- **Example:**
  - If $x = 4$, $y = 3$, $f(x, y) = -12$
  - $\frac{\partial f}{\partial x} = -3$, increase x slightly by h, f decrease by $3h$.
  - $\frac{\partial f}{\partial y} = 4$, increase y slightly by h, f increase by $4h$.
- **Gradient:**
  - Definition: ∇f is a vector of partial derivatives

$$\nabla f = [\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}] = [y, x]$$



Note: We aim to find the negative gradient, as indicates a decreasing slope, which means that moving downward will lead us to the minimum point.

Compound expressions with chain rule

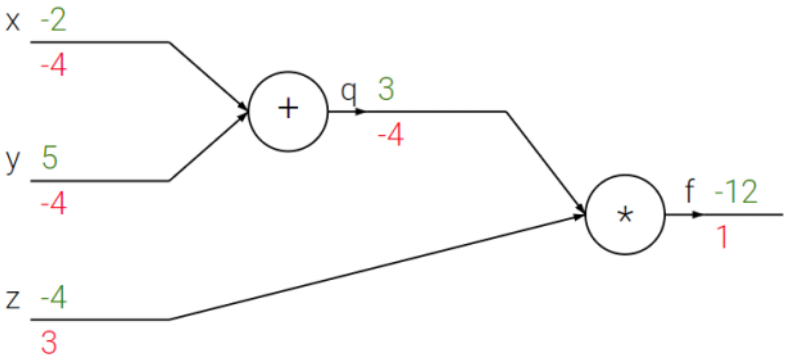
Function: $f(x, y, z) = (x + y)z$

Forward pass: (Red)

- $x = -2, y = 5, z = -4$
- $q = x + y = 3$
- $f = q \times z = -12$

Backward Pass: (Green)

- Step 1:
  - $\frac{\partial f}{\partial z} = q = 3$
  - $\frac{\partial f}{\partial q} = z = -4$
- Step 2:
  - $\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q} \times \frac{\partial q}{\partial x} = -4 \times (1) = -4$
  - $\frac{\partial f}{\partial y} = \frac{\partial f}{\partial q} \times \frac{\partial q}{\partial y} = -4 \times (1) = -4$



Intuitive understanding of backpropagation

Local Process in Backpropagation:

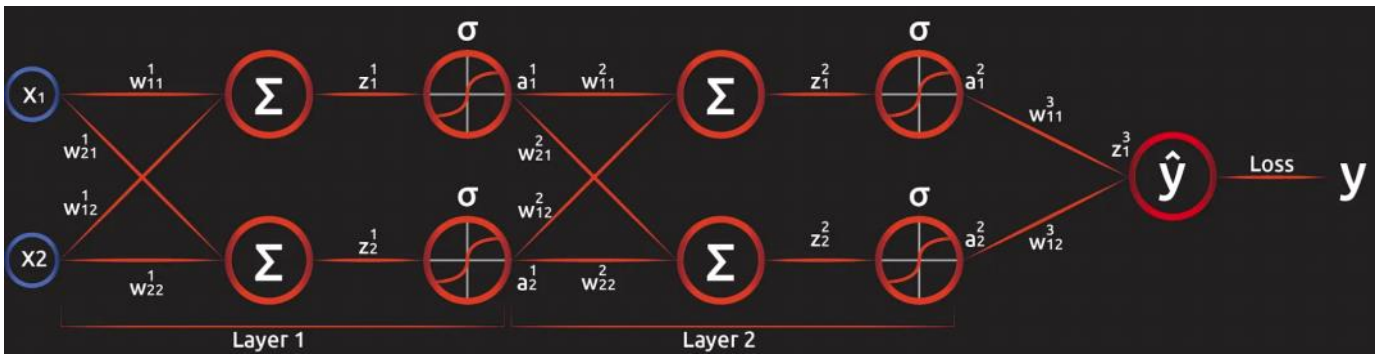
- **Local Computation:** Each gate in a neural network can independently compute:
  - Output value
  - The local gradient of its output with respect to its inputs.
- **Independence:** Gates do this without needing to know the details of the entire network.

Forward and Backward Pass:

- **Forward Pass:** each gate computes its output based on its inputs.
- **Backward Pass (Backpropagation):**
  - Each gate learns about the gradient of its output with respect to the final output of the network.
  - The chain rule is applied here, meaning each gate multiplies this gradient by the local gradients it computed during the forward pass.

Previous Example:

- **Addition Gate:**
  - Receives inputs ([-2, 5]) and computes the output (3)
  - The local gradient for both inputs is (+1)
- **Multiplication Gate:**
  - Uses the output of the addition gate
  - If the final output of the network is (-12), during backpropagation, the addition gate learns that the gradient of its output with respect to the final output is (-4)
- **Gradient Multiplication:**
  - The addition gate multiplies this gradient by its local gradients, resulting in gradients of (-4) for both (x) and (y)
  - This means that decreasing (x) or (y) would decrease the output of the addition gate, which in turn would increase the output of the multiplication gate



Modularity: Sigmoid example

Any differentiable function can act as a gate in a neural network. We can group multiple gates into one or decompose a function into multiple gates for convenience.

2D Neuron with Sigmoid Activation:

- **Function:**

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$

It describes a 2D neuron with inputs x and weights w, using the sigmoid activation function

- **Components:**

$$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1$$

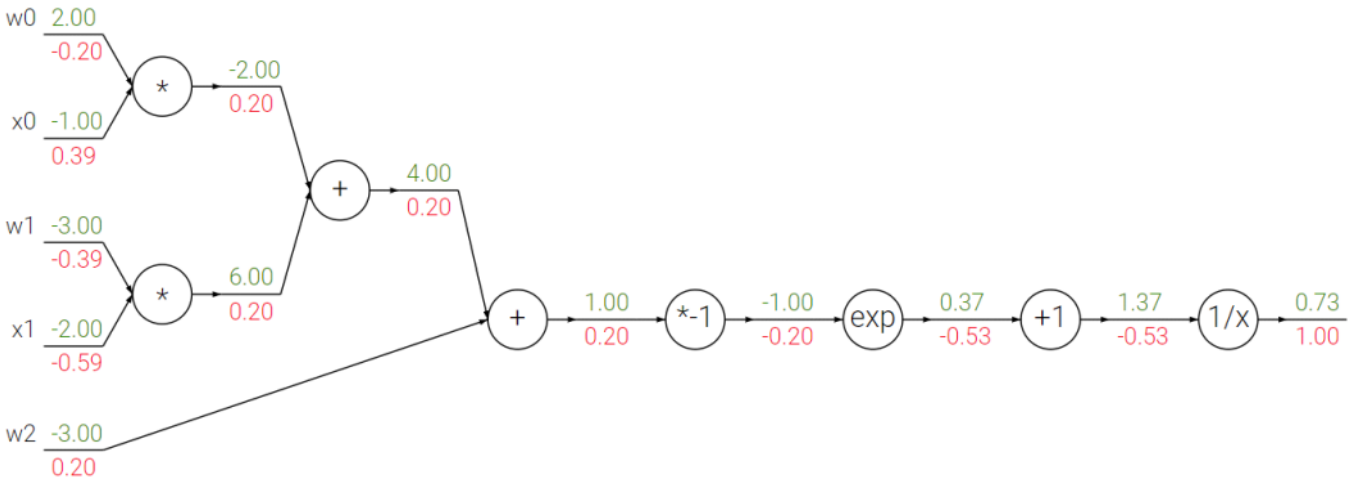
$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

- **Sigmoid Function:**

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

$$\rightarrow \quad \frac{d\sigma(x)}{dx} = \frac{e^{-x}}{(1+e^{-x})^2} = \left(\frac{1+e^{-x}-1}{1+e^{-x}}\right)\left(\frac{1}{1+e^{-x}}\right) = (1-\sigma(x))\,\sigma(x)$$



References

BP Links:

BP Stanford CS231n Lecture Note	CS231n BP
3b1b BP Part 1	YouTube
3b1b BP Part 2	YouTube