Tuesday • September 28, 2021

The Backpropagation Algorithm

Yale

LING 380/780

Neural Network Models of Linguistic Structure

Review: Three Ingredients of Machine Learning

- **Architecture:** Neural networks (multi-layer perceptrons)
- **Loss Function:** Cross-entropy loss function
- **Optimization Algorithm:** Stochastic gradient descent

Review: Multi-Layer Perceptrons

Linear Layer with Activation Function

$$\mathbf{a}^{(l)} = f(\mathbf{W}^{(l)}\mathbf{a}^{(l-1)} + \mathbf{b}^{(l)})$$

- $\mathbf{W}^{(l)}$: weight
- $\mathbf{b}^{(l)}$: bias
- f: activation function (sigmoid, tanh, softmax, ReLU)

Review: Cross-Entropy Loss Function

Cross-Entropy Loss Function:

$$L_{\mathrm{CE}}(\hat{\mathbf{y}}, y) = -\ln(\hat{y}_y)$$

- $\hat{\mathbf{y}} \in \mathbb{R}^n$: probability vector, where $\hat{y}_i$ is the probability that the correct label is class i
- $y \in \{1,2,\ldots,n\}$: the correct label

Review: Stochastic Gradient Descent

- **Input:** Dataset $\mathbb{D}$, architecture $f(\,\cdot\,, \theta)$
- **Hyperparameters:** Batch size b, learning rate η
- Initialize θ to a random value.
- Partition $\mathbb{D}$ into mini-batches $\mathbb{B}_1, \mathbb{B}_2, \ldots, \mathbb{B}_k$ of size b.

Review: Stochastic Gradient Descent

- Repeat indefinitely:
  - For each mini-batch $\mathbb{B}_i$:
    - Let

$$\mathscr{L} = \sum_{\mathbf{x},\mathbf{y} \in \mathbb{B}_i} L(f(\mathbf{x}, \theta), \mathbf{y})$$

    - Set $\theta \leftarrow \theta - \eta \nabla_\theta \mathscr{L}$.

How do you compute gradients?

Baby Steps: Single-Variable Linear Gradient

Consider a single-variable linear model:

$$y = wx + b$$

What is dy/dx?

Answer: w

Baby Steps: Single-Variable Logistic Gradient

Consider a single-variable logistic model:

$$y = \sigma(wx + b)$$

What is dy/dx? **Hint:**

$$\frac{d}{dx}\sigma(x) = \sigma(x)(1 - \sigma(x))$$

Single Variable Chain Rule

$$\frac{dy}{dx} = \frac{d}{dx}\sigma(wx + b)$$

Multi-Variable Linear Gradient

Consider a multi-variable linear model:

$$y = \mathbf{w}^\top \mathbf{x} + b$$

What is $\nabla_{\mathbf{x}} y$?

Answer: $\mathbf{w}$

Multi-Variable Linear Gradient

Single partial derivative:

$$\frac{\partial y}{\partial x_i} = \frac{\partial}{\partial x_i}\left(w_1 x_1 + w_2 x_2 + \ldots + w_n x_n + b\right) = w_i$$

Gradient vector:

$$\nabla_{\mathbf{x}} y = \begin{bmatrix} \partial y / \partial x_1 \\ \vdots \\ \partial y / \partial x_n \end{bmatrix} = \begin{bmatrix} w_1 \\ \vdots \\ w_n \end{bmatrix} = \mathbf{w}$$

Multi-Variable Logistic Gradient

Consider a multi-variable logistic model:

$$y = \sigma(\mathbf{w}^\top \mathbf{x} + b)$$

What is $\nabla_{\mathbf{x}} y$?

We need a multi-variable chain rule!

Multi-Input, Multi-Output Gradients

Let $f : \mathbb{R}^m \to \mathbb{R}^n$, and let $\mathbf{y} = f(\mathbf{x})$. The ***Jacobian of* $\mathbf{y}$ *with respect to* $\mathbf{x}$** is the $n \times m$ matrix

$$\frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \begin{bmatrix} (\nabla_{\mathbf{x}} y_1)^\top \\ (\nabla_{\mathbf{x}} y_2)^\top \\ \vdots \\ (\nabla_{\mathbf{x}} y_n)^\top \end{bmatrix} = \begin{bmatrix} \partial y_1/\partial x_1 & \partial y_1/\partial x_2 & \dots & \partial y_1/\partial x_m \\ \partial y_2/\partial x_1 & \partial y_2/\partial x_2 & \dots & \partial y_2/\partial x_m \\ \vdots & \vdots & \ddots & \vdots \\ \partial y_n/\partial x_1 & \partial y_n/\partial x_2 & \dots & \partial y_n/\partial x_m \end{bmatrix}$$

Multivariable Chain Rule

Single-Variable Chain Rule

$$\frac{dz}{dx} = \frac{dz}{dy} \cdot \frac{dy}{dz}$$

Multivariable Chain Rule

$$\frac{\partial \mathbf{z}}{\partial \mathbf{x}} = \frac{\partial \mathbf{z}}{\partial \mathbf{y}} \frac{\partial \mathbf{y}}{\partial \mathbf{x}}$$

Multi-Variable Logistic Gradient

$$\nabla_{\mathbf{x}} y = \left(\frac{\partial}{\partial \mathbf{x}} \sigma(\mathbf{w}^\top \mathbf{x} + b) \right)^\top$$

Automatic Differentiation

Automatic Differentiation

- So far, we have been computing gradients by hand.
- But the whole point of stochastic gradient descent is that we don't have to compute anything by hand.
- Therefore we will use the **backpropagation algorithm** to **compute gradients automatically**.

Building Blocks: The Algebra of Functions

- We would like an algorithm that computes the gradient of any function.
- But what does "any" mean?
- Take some **elementary functions** we already know the gradient of, and **put them together using the chain rule**.

Backpropagation: Forward Pass

Consider a function $\alpha = (x + y)z$.

Backpropagation: Backward Pass

Consider a function $\alpha = (x + y)z$.

Backpropagation: Backward Pass

Consider a function $\alpha = (x + y)z$.

$$\frac{\partial \alpha}{\partial (x+y)} = \frac{\partial \alpha}{\partial \alpha} \cdot \frac{\partial \alpha}{\partial (x+y)}$$

$$\frac{\partial \alpha}{\partial z} = \frac{\partial \alpha}{\partial \alpha} \cdot \frac{\partial \alpha}{\partial z}$$

Backpropagation: Backward Pass

Consider a function $\alpha = (x + y)z$.

Computation Graphs

- A **computation graph** is a **data structure** representing an **algebraic expression**.
- Computation graph node properties:
  - Node children, value, gradient
  - forward($\mathbf{x}$) $=$ node value given input $\mathbf{x}$
  - backward(δ) $= \delta^\top \dfrac{\partial \text{value}}{\partial \mathbf{x}} =$ node gradient where $\delta^\top = \dfrac{\partial \mathscr{L}}{\partial \text{value}}$

Computation Graphs

- **Example:** Variable node
- forward($\mathbf{x}$) $= \mathbf{x}$
- backward(δ) $= \delta^\top$

Computation Graphs

- **Example:** $+$ node
- forward$(\mathbf{a}, \mathbf{b}) = \mathbf{a} + \mathbf{b}$
- backward$(\delta) = (\delta^\top, \delta^\top)$

Computation Graphs

- **Example:** $\odot$ node
- forward$(\mathbf{a}, \mathbf{b}) = \mathbf{a} \odot \mathbf{b}$
- backward$(\delta) = ?$

Computation Graphs

$$\frac{\partial(\mathbf{a} \odot \mathbf{b})_i}{\partial a_j} = \frac{\partial(a_i b_i)}{\partial a_j} = \begin{cases} b_i, & i = j \\ 0, & i \neq j \end{cases}$$

In matrix form:

$$\frac{\partial(\mathbf{a} \odot \mathbf{b})}{\partial \mathbf{a}} = \begin{bmatrix} b_1 & 0 & \dots & 0 \\ 0 & b_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & b_n \end{bmatrix} = \text{diag}(\mathbf{b})$$

Computation Graphs

$$\frac{\partial(\mathbf{a} \odot \mathbf{b})_i}{\partial b_j} = \frac{\partial(a_i b_i)}{\partial b_j} = \begin{cases} a_i, & i = j \\ 0, & i \neq j \end{cases}$$

In matrix form:

$$\frac{\partial(\mathbf{a} \odot \mathbf{b})}{\partial \mathbf{b}} = \begin{bmatrix} a_1 & 0 & \dots & 0 \\ 0 & a_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & a_n \end{bmatrix} = \mathrm{diag}(\mathbf{a})$$

Computation Graphs

In general, if

$$f(\mathbf{x}) = \begin{bmatrix} f_1(x_1) & f_2(x_2) & \dots & f_n(x_n) \end{bmatrix}^\top,$$

then

$$\frac{\partial f(\mathbf{x})}{\partial \mathbf{x}} = \text{diag}\left(\begin{bmatrix} \frac{df_1(x_1)}{dx_1} & \frac{df_2(x_2)}{dx_2} & \dots & \frac{df_n(x_n)}{dx_n} \end{bmatrix}^\top \right)$$

Computation Graphs

- **Example:** $\odot$ node
- forward$(\mathbf{a}, \mathbf{b}) = \mathbf{a} \odot \mathbf{b}$
- backward$(\delta) = (\delta^\top \text{diag}(\mathbf{b}), \delta^\top \text{diag}(\mathbf{a}))$

$$\mathbf{U}\text{diag}(\mathbf{v}) = \mathbf{U}(\mathbf{I}_n \odot \mathbf{v}) = (\mathbf{U}\mathbf{I}_n) \odot \mathbf{v} = \mathbf{U} \odot \mathbf{v}$$

Computation Graphs

- **Example:** $\odot$ node
- $\text{forward}(\mathbf{a}, \mathbf{b}) = \mathbf{a} \odot \mathbf{b}$
- $\text{backward}(\delta) = (\delta \odot \mathbf{b}, \delta \odot \mathbf{a})$

Computation Graphs

- **Example:** Sigmoid node
- forward$(\mathbf{x}) = \sigma(\mathbf{x})$
- backward$(\delta) = \delta \odot \sigma(\mathbf{x})(1 - \sigma(\mathbf{x}))$

Complex Computation Graphs

- **Example:** Linear layer
- forward($\mathbf{x}$) $= \mathbf{W}\mathbf{x} + \mathbf{b}$, where $\mathbf{W}$ and $\mathbf{b}$ are parameters
- backward(δ) $= \left(\delta^\top \frac{\partial \text{value}}{\partial \mathbf{x}}, \delta^\top \frac{\partial \text{value}}{\partial \mathbf{W}}, \delta^\top \frac{\partial \text{value}}{\partial \mathbf{b}}\right)$

Complex Computation Graphs

Let $\mathbf{y} = \mathbf{W}\mathbf{x} + \mathbf{b}$.

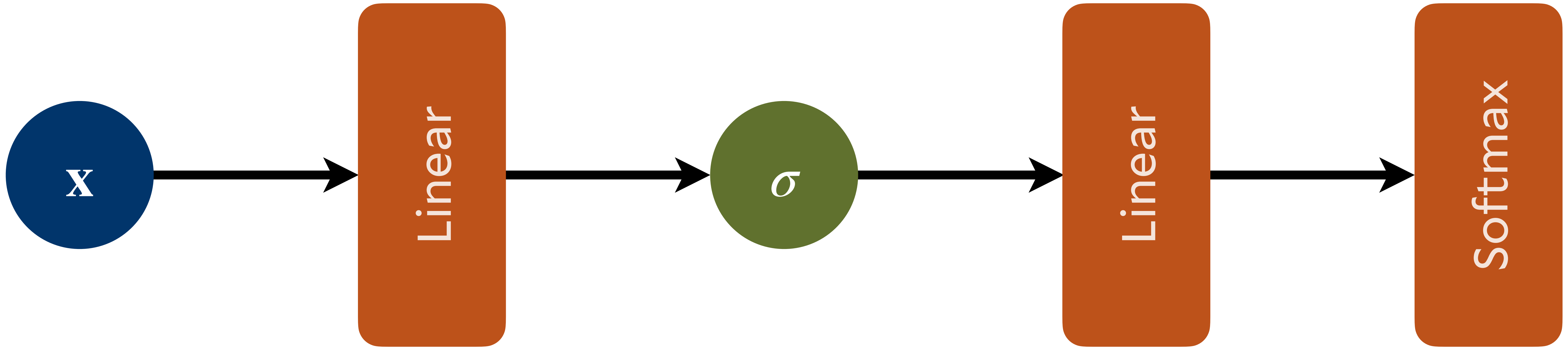
$$\frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \mathbf{W}$$

$$\frac{\partial \mathbf{y}}{\partial \mathbf{W}} = \mathbf{I}_{:,:,\mathrm{newaxis}} \odot \mathbf{x}_{\mathrm{newaxis},\mathrm{newaxis},:}$$

$$\frac{\partial \mathbf{y}}{\partial \mathbf{b}} = \mathbf{I}$$

Neural Network Computation Graphs

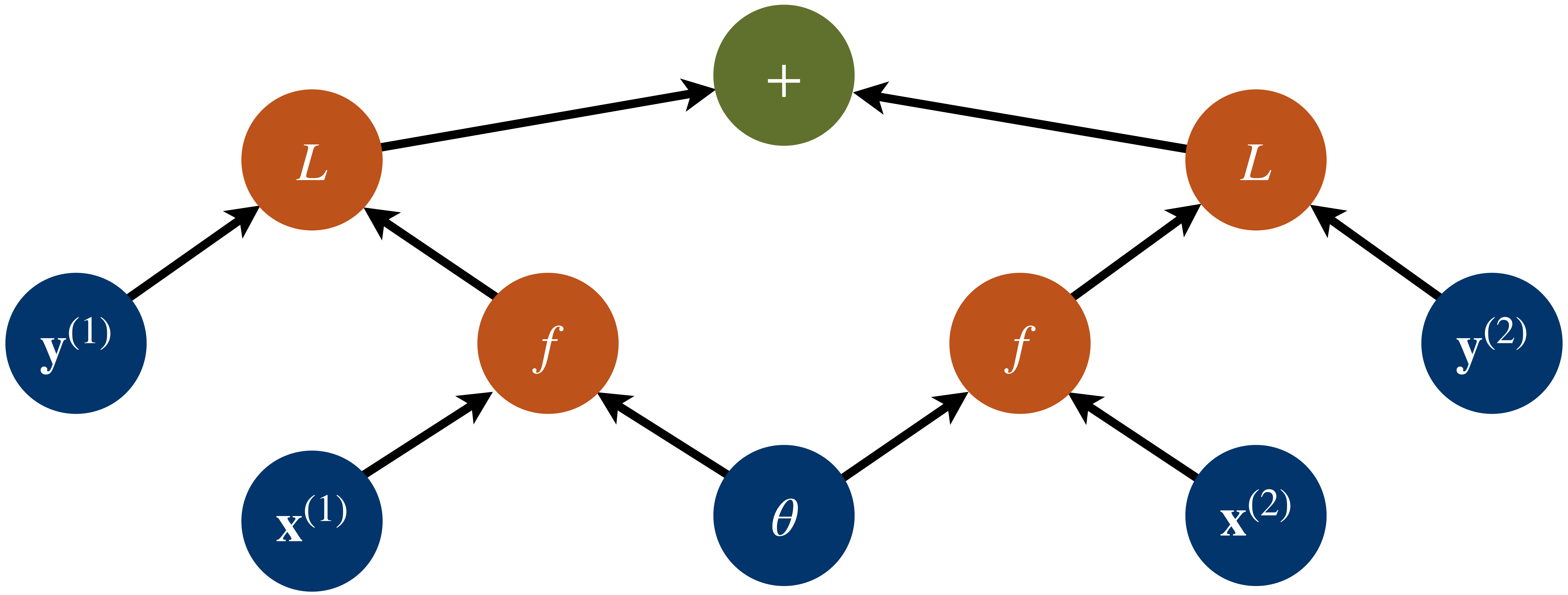
Computation Graph for a Multi-Layer Perceptron



Computation Graph for an Objective

Objective for a Mini-Batch of Size 2

$$\mathscr{L} = L(f(\mathbf{x}^{(1)}; \theta), \mathbf{y}^{(1)}) + L(f(\mathbf{x}^{(2)}; \theta), \mathbf{y}^{(2)})$$



Full Backpropagation Algorithm

- **Input:**
  - Directed acyclic computation graph with root node
  - Values for variable nodes
  - Values for layer parameters

Full Backpropagation Algorithm

- **Forward Pass:**
  - Start at the root node.
  - If this is a variable node:
    - Set the value of this node to be the value designated for this variable.

Full Backpropagation Algorithm

- **Forward Pass (Continued):**
  - If this is not a variable node:
    - Recursively call the forward pass on this node's children.
    - Set this node's value to be the output of the forward function applied to the values of this node's children.

Full Backpropagation Algorithm

- **Backward Pass:**
  - Start at the root node.

Full Backpropagation Algorithm

- **Backward Pass (Continued):**
  - If the current node is the root node, set the gradient to 1.
  - If the current node is a variable node, terminate.
  - Set the gradients of this node's children to the outputs of their backward functions, applied to this node's gradient.
  - Recursively call the backward pass on this node's children.