Recitation 4
Neural Networks, Automatic Differentiation

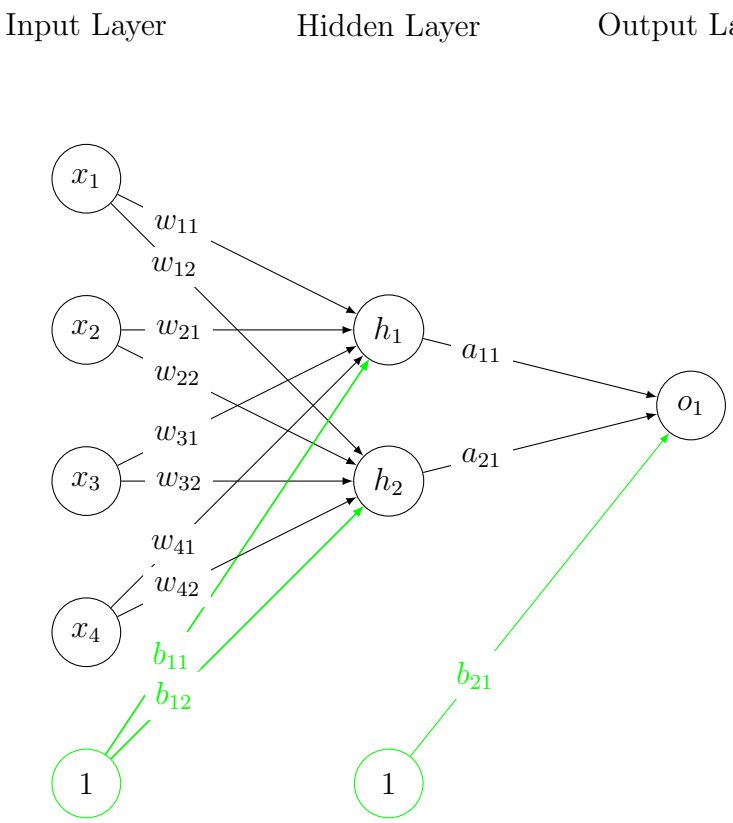
10-701: Introduction to Machine Learning

09/26/2025

1 Neural Networks

Neural Networks Warmup

For this section, as shown in the figure below, let x_i for $i = 1, \ldots, 4$ denote input layer nodes. Let h_1 and h_2 be hidden layer nodes, and let o_1 be the output node. w_{ij} are the weights between the input layer and the hidden layer, a_{ij} are the weights between the hidden layer and the output layer, and b_{ij} are bias terms. Lastly, suppose the hidden layer nodes use some function σ as their activation function.



1. Let us denote t_1 to be the input to the h_1 node and t_2 to be the input to the h_2 node after the input layer has been processed. What are t_1 and t_2? Express your answer in terms of the inputs, weights, and biases.

   The input to h_1 is $w_{11}x_1 + w_{21}x_2 + w_{31}x_3 + w_{41}x_4 + b_{11}$ and the input to h_2 is $w_{12}x_1 + w_{22}x_2 + w_{32}x_3 + w_{42}x_4 + b_{12}$

2. What is the output of the h_1 node?

   $\sigma(w_{11}x_1 + w_{21}x_2 + w_{31}x_3 + w_{41}x_4 + b_{11})$

3. Denoting h_1 as the output of node h_1 and h_2 as the output of the node h_2, what is the output o_1?

   $o_1 = a_{11}h_1 + a_{21}h_2 + b_{21}$

1.1 Translating to Vector Notation

Define $\mathbf{X} = (x_1, x_2, x_3, x_4)^T$, $\mathbf{b}_1 = (b_{11}, b_{12})^T$ and

$$\mathbf{W} = \begin{pmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \\ w_{31} & w_{32} \\ w_{41} & w_{42} \end{pmatrix} \qquad \mathbf{A} = \begin{pmatrix} a_{11} & a_{21} \end{pmatrix}$$

1. Let be $\mathbf{t} = (t_1, t_2)^T$ be the vector of inputs to the hidden layer. What is the vector equation that produces $\mathbf{t}$? Express your answer in terms of $\mathbf{x}$, $\mathbf{b}_1$, and $\mathbf{W}$.

   $\mathbf{t} = \mathbf{W}^T\mathbf{x} + \mathbf{b}_1$

2. Let $\mathbf{h} = (h_1, h_2)^T$ be the vector of outputs from the hidden layer. Express $\mathbf{h}$ in terms of $\mathbf{t}$.

   $\mathbf{h} = \sigma(\mathbf{t})$, where the notation denotes that the activation function σ is applied element-

wise to $\mathbf{h}$.

3. Express o_1 in terms of $\mathbf{h}$, b_{21}, and $\mathbf{A}$.

$o_1 = \mathbf{A}\mathbf{h} + b_{21}$

4. Represent the output of the neural network, o_1, with a single equation using all of the variables mentioned above.

$o_1 = \mathbf{A}(\sigma(\mathbf{W}^T\mathbf{x} + \mathbf{b}_1)) + b_{21}$

2 Computation Graphs and Automatic Differentiation

Automatic differentiation is a cornerstone technique that enables efficient computation of derivatives, which is essential for efficiently training neural networks. This section guides you through the basic concepts and approaches to differentiation, with a focus on symbolic and automatic differentiation.

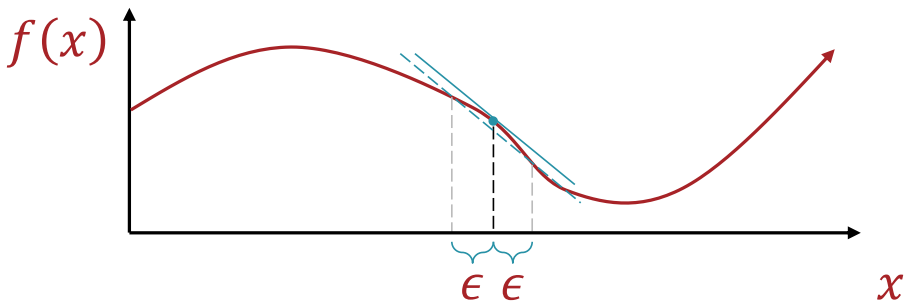
Given $f : \mathcal{R}^D \to \mathcal{R}$, our goal is to compute the gradient $\nabla_{\mathbf{x}} f(\mathbf{x}) = \frac{\partial f(\mathbf{x})}{\partial \mathbf{x}}$. Let's explore three fundamental approaches.

2.1 Approach 1: Finite difference method

The finite difference method approximates the derivative of f with respect to x_i as follows:

$$\frac{\partial f(\mathbf{x})}{\partial x_i} \approx \frac{f(\mathbf{x} + \epsilon \mathbf{d}_i) - f(\mathbf{x} - \epsilon \mathbf{d}_i)}{2\epsilon}$$

where $\mathbf{d}_i$ is a one-hot vector with a 1 in the i-th position.



The choice of ϵ is critical; it must be small for accuracy but not too small to avoid floating-point issues. Getting the full gradient requires computing the above approximation for each dimension of the input.

Pros:

1. Useful for verifying more complex differentiation methods.

Cons:

1. Requires the ability to call $f(\mathbf{x})$.
2. Computationally expensive, especially for high-dimensional inputs.

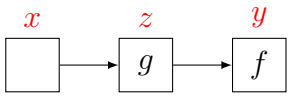
2.2 Approach 2: Symbolic Differentiation

We will use computation graphs to help with understanding differentiation. A computation graph is a topologically ordered DAG representing the flow of information in an algorithm and is structured as follows:

- **Nodes** are represented as rectangles with one node corresponding to an intermediate variable within the algorithm. Each node is labeled with the function that it computes (inside the box) and the variable name (outside the box).
- For neural networks, each weight, feature value, label and *bias term* appears as a node.
- **Edges** indicate the flow of information between nodes. Edges are directed and do not have labels.

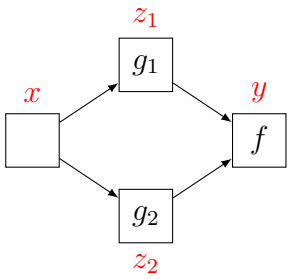
Symbolic differentiation uses mathematical expressions to obtain gradients, where the objectives are often composite functions. To compute gradients, we use the chain rule:

1. If $y = f(z)$ and $z = g(x)$, then the corresponding computation graph is



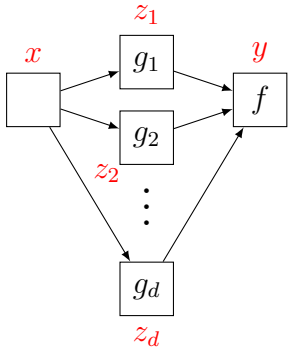
$$\implies \quad \frac{\partial y}{\partial x} = \frac{\partial y}{\partial z}\frac{\partial z}{\partial x}$$

2. If $y = f(z_1, z_2)$ and $z_1 = g_1(x)$, $z_2 = g_2(x)$, then



$$\implies \quad \frac{\partial y}{\partial x} = \frac{\partial y}{\partial z_1}\frac{\partial z_1}{\partial x} + \frac{\partial y}{\partial z_2}\frac{\partial z_2}{\partial x}$$

3. If $y = f(\mathbf{z})$ and $\mathbf{z} = g(x)$, then



$$\implies \quad \frac{\partial y}{\partial x} = \sum_{i=1}^{d} \frac{\partial y}{\partial z_i}\frac{\partial z_i}{\partial x}$$

Consider the function:

$$y = f(x, z) = e^{xz} + \frac{xz}{\ln(x)} + \frac{\sin(\ln(x))}{xz}$$

Compute $\frac{\partial y}{\partial x}$ and $\frac{\partial y}{\partial z}$ at $x = 2$, $z = 3$.

$$\begin{aligned}
\frac{\partial y}{\partial x} &= \frac{\partial}{\partial x}(e^{xz}) + \frac{\partial}{\partial x}\left(\frac{xz}{\ln(x)}\right) + \frac{\partial}{\partial x}\left(\frac{\sin(\ln(x))}{xz}\right) \\
&= ze^{xz} + \frac{z}{\ln(x)} - \frac{z}{\ln(x)^2} + \frac{\cos(\ln(x))}{x^2 z} - \frac{\sin(\ln(x))}{x^2 z} \\
&= 3e^6 + \frac{3}{\ln(2)} - \frac{3}{\ln(2)^2} + \frac{\cos(\ln(2))}{12} - \frac{\sin(\ln(2))}{12} \\
\frac{\partial y}{\partial z} &= \frac{\partial}{\partial z}(e^{xz}) + \frac{\partial}{\partial z}\left(\frac{xz}{\ln(x)}\right) + \frac{\partial}{\partial z}\left(\frac{\sin(\ln(x))}{xz}\right) \\
&= 2e^6 + \frac{2}{\ln(2)} - \frac{\sin(\ln(2))}{18}
\end{aligned}$$

Pros:

1. Provides exact derivatives.

Cons:

1. Requires systematic knowledge of derivatives.
2. Can be computationally expensive if poorly implemented.

2.3 Approach 3: Automatic Differentiation (reverse mode)

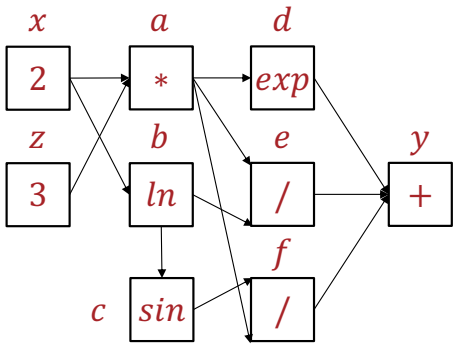
Automatic differentiation computes derivatives efficiently by utilizing the chain rule in a structured manner. For a given function, we first perform a forward pass to compute intermediate variables and then a reverse pass to calculate gradients. Specifically, this method constructs a computation graph that breaks down complex functions into simpler operations. Derivatives are then computed by propagating values backward through this graph.

Given the same function as before:

$$y = f(x, z) = e^{xz} + \frac{xz}{\ln(x)} + \frac{\sin(\ln(x))}{xz}$$

we first define some intermediate quantities, draw the computation graph, and run the forward computation:

$$a = xz$$
$$b = \ln(x)$$
$$c = \sin(b)$$
$$d = e^a$$
$$e = a/b$$
$$f = c/a$$
$$y = d + e + f$$



With the above computation graph, find $\frac{\partial y}{\partial x}$ and $\frac{\partial y}{\partial z}$ at $x = 2$, $z = 3$.

(Hint: Use the chain rule to find the derivative of y with respect to each intermediate variable in a reverse manner starting from y. Remember to reuse previously computed quantities.)

$$g_y = \frac{\partial y}{\partial y} = 1$$

$$g_d = g_e = g_f = 1$$

$$g_c = \frac{\partial y}{\partial c} = \frac{\partial y}{\partial f}\frac{\partial f}{\partial c} = g_f\left(\frac{1}{a}\right)$$

$$g_b = \frac{\partial y}{\partial b} = \frac{\partial y}{\partial e}\frac{\partial e}{\partial b} + \frac{\partial y}{\partial c}\frac{\partial c}{\partial b}$$
$$= g_e\left(-\frac{a}{b^2}\right) + g_c(\cos(b))$$

$$g_a = \frac{\partial y}{\partial a} = \frac{\partial y}{\partial f}\frac{\partial f}{\partial a} + \frac{\partial y}{\partial e}\frac{\partial e}{\partial a} + \frac{\partial y}{\partial d}\frac{\partial d}{\partial a}$$
$$= g_f\left(\frac{-c}{a^2}\right) + g_e\left(\frac{1}{b}\right) + g_d(e^a)$$

$$g_x = \frac{\partial y}{\partial x} = \frac{\partial y}{\partial b}\frac{\partial b}{\partial x} + \frac{\partial y}{\partial a}\frac{\partial a}{\partial x} = g_b\left(\frac{1}{x}\right) + g_a(z)$$

$$g_z = \frac{\partial y}{\partial z} = \frac{\partial y}{\partial a}\frac{\partial a}{\partial z} = g_a(x)$$

Pros:

1. Computational cost of computing $\frac{\partial f(\mathbf{x})}{\partial \mathbf{x}}$ is proportional to the cost of computing $f(\mathbf{x})$.
2. Reduces computational overhead through reuse of intermediate results.

Cons:

1. Requires systematic knowledge of derivatives *and* an algorithm for computing $f(\mathbf{x})$.
2. Implementation complexity due to the need for maintaining a computation graph.

2.4 Summary

The finite difference method is straightforward but computationally demanding. Likewise, symbolic differentiation offers precision but can become heavy if poorly optimized. Automatic differentiation, on the other hand, provides a balance between efficiency and complexity with its reuse of computation, making it the backbone of modern computational frameworks like PyTorch.