Lecture 2

Linear Algebra and Probability Review

CSE465: Pattern Recognition and Neural Network
Sec: 3
Faculty: Silvia Ahmed (SvA)
Summer 2025

Today’s Topics

- Linear Algebra Essentials
  - Vectors
  - Matrices
  - Dot Product
- Matrix Calculus for Deep Learning
  - Scalar derivatives
  - Vector derivatives
  - Matrix derivatives
  - DL context
- Probability Distributions for DL
  - Review of basic probability concepts
  - Key distributions

Why are Linear Algebra & Probability Critical for DL?

- Linear Algebra:
  - The "language" of neural networks.
  - Data represented as vectors and matrices.
  - Network operations (transformations, weights) are matrix multiplications.
- Probability:
  - Understanding data distributions.
  - Interpreting model outputs (e.g., classification probabilities).
  - Formulating loss functions and regularization.
  - Uncertainty modeling.

Linear Algebra Essentials

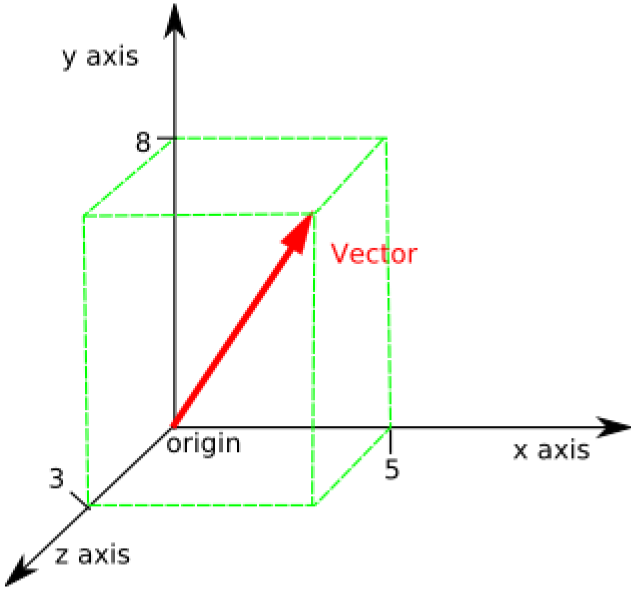
Vectors: Definition

- Ordered list of numbers (e.g., [x1, x2, x3]).
- In deep learning, vectors are commonly used to represent individual data points, features of an input, or learned representations (embeddings).
- Definition: A vector v of dimension n can be written as:

$$v = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix}$$

  - This is a **column vector**

  [conventionally used in DL for inputs and features]
  - A **row vector** would be $v^T = [v_1 \quad v_2 \quad \cdots \quad v_n]$
- Geometric interpretation:
  - a point in n-dimensional space
  - or a directed line segment from the origin to that point.



Vectors: Operations

- Vector Addition: Element-wise sum of two vectors of the same dimension.

$$\begin{bmatrix} v_1 \\ v_2 \end{bmatrix} + \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} = \begin{bmatrix} v_1 + w_1 \\ v_2 + w_2 \end{bmatrix}$$

- Scalar Multiplication: Multiplying a vector by a scalar (a single number) scales its magnitude.

$$c \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = \begin{bmatrix} cv_1 \\ cv_2 \end{bmatrix}$$

Vectors: DL context

- An image can be "flattened" into a vector of pixel values
- A word can be represented by a "word embedding" vector
- The features describing a data point (e.g., height, weight, age) form a feature vector

Matrices

- Rectangular arrays of numbers (e.g., 2x3 matrix)
- Dimensions: `rows x columns`
- In deep learning, matrices are predominantly used to represent collections of data points (mini-batches), weights connecting layers in a neural network, or learned transformations.
- Definition: A matrix A with m rows and n columns (an $m \times n$ matrix) is written as:

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}$$

Special Matrices

- **Identity matrix (*I*):** A square matrix with ones on the main diagonal and zeros elsewhere. When multiplied by another matrix, it leaves the matrix unchanged.

$$I = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

- **Diagonal Matrix:** A square matrix where all off-diagonal elements are zero.
- **Symmetric Matrix:** A square matrix where $A = A^T$ (transpose of A equals A).

Matrices: Operations

- **Matrix Addition/Subtraction:** Element-wise operations, requiring matrices to have the same dimensions.
- **Scalar Multiplication:** Multiplying every element of the matrix by a scalar.
- **Matrix Transpose (A^T):** Swapping rows and columns of a matrix. If A is m x n, then A^T is n x m.

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \implies A^T = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$$

Matrix Multiplication

- Not element-wise! Dot product of rows and columns
- **Definition:** The product of two matrices A (size: m x k) and B (size: k × n) results in a matrix C (size: m x n). Each element c_{ij} of C is the dot product of the i-th row of A and the j-th column of B:

$$c_{ij} = \sum_{l=1}^{k} a_{il} b_{lj}$$

- **Rules for Multiplication:** For A×B, the number of columns in A *must* equal the number of rows in B.

$$(m \times k) \times (k \times n) \rightarrow (m \times n)$$

Matrix Multiplication (contd.)

- **Non-Commutativity:** In general, $AB \neq BA$. The order of multiplication matters.
- **Geometric Interpretation:** Matrix multiplication can represent various geometric transformations such as scaling, rotation, reflection, and projection.
- DL Context:
  - The core operation within a neural network layer: $y=Wx+b$, where x is the input vector, W is the weight matrix, b is the bias vector, and y is the output vector.
  - Processing mini-batches: If X is a matrix where each row is a data point (or each column, depending on convention), and W is a weight matrix, then XW or WX processes the entire batch efficiently.

Dot Product (Vector Inner Product)

- **Definition:** For two vectors $v = \begin{bmatrix} v_1 \\ \vdots \\ v_n \end{bmatrix}$ and $w = \begin{bmatrix} w_1 \\ \vdots \\ w_n \end{bmatrix}$ of the same dimension n, their dot product is a scalar:

$$v \cdot w = v^T w = \sum_{i=1}^{n} v_i w_i = v_1 w_1 + v_2 w_2 + \cdots + v_n w_n$$

- **Geometric Interpretation:**
  - If $v \cdot w = 0$, the vectors are orthogonal (perpendicular).
  - It relates to the angle θ between vectors: $v \cdot w = ||v|| \cdot ||w|| \cos(\theta)$. This is used for cosine similarity, a measure of how similar two vectors are.

Dot Product (Vector Inner Product) (contd.)

- DL Context:
  - The "weighted sum" in a neuron's activation: The dot product of input features x and weights w (i.e., $w \cdot x$) forms the core of many activation functions before a non-linearity is applied.
  - Attention mechanisms in advanced architectures use dot products to compute similarity scores between different parts of the input.

Matrix Calculus for Deep Learning

Matrix Calculus - Introduction

- Why is this hard but crucial?
  - Deep Learning involves optimizing functions (loss functions) with millions of parameters.
  - We need to compute gradients to update these parameters efficiently.
  - "Backpropagation" is essentially repeated application of the chain rule.
- Review:
  - Scalar Derivatives: $\frac{dy}{dx}$ (slope)
  - Chain Rule: $\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}$ (how changes propagate)

Matrix Calculus - Vector Derivatives

- **Gradient ($\nabla_x f(x)$):** Derivative of a **scalar** function (f) with respect to a **vector** input (x).
- Result: A vector of partial derivatives.

  - Let $f(x)$ be a scalar function where $x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$. The gradient of f with respect to x is a vector containing all the partial derivatives of f with respect to each component of x:

$$\nabla_x f(x) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{bmatrix}$$

Vector Derivatives (contd.)

- **Interpretation:** The gradient vector points in the direction of the steepest increase of the function f. In optimization algorithms like gradient descent, we move in the opposite direction of the gradient to find a minimum.
- **DL Context:** This is used to compute the gradients of the *loss function* (a scalar value) with respect to the network's *parameters* (weights and biases, which are vectors or matrices). For instance, $\nabla_w \mathcal{L}$ tells us how to adjust the weight vector w to reduce the loss $\mathcal{L}$.

Vector Derivatives (contd.)

- **Jacobian Matrix (J):** Derivative of a vector-valued function with respect to a vector input.

  - Let $y = f(x)$ be a function that maps an n-dimensional input vector x to an m-dimensional output vector $y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix}$. The Jacobian matrix of f with respect to x is an $m \times n$ matrix:

$$\mathbf{J} = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_1}{\partial x_2} & \cdots & \frac{\partial y_1}{\partial x_n} \\ \frac{\partial y_2}{\partial x_1} & \frac{\partial y_2}{\partial x_2} & \cdots & \frac{\partial y_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial y_m}{\partial x_1} & \frac{\partial y_m}{\partial x_2} & \cdots & \frac{\partial y_m}{\partial x_n} \end{bmatrix}$$

Vector Derivatives (contd.)

- **DL Context:** Jacobians are essential for understanding how errors propagate between layers. If we have a sequence of operations like $z = g(y)$ and $y = f(x)$, then $\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}$ (using the chain rule), where these "derivatives" are actually Jacobian matrices.

Matrix Derivatives

- In deep learning, we frequently need to compute the derivative of a scalar loss function with respect to a matrix of weights.
- **Derivative of a scalar with respect to a matrix:**

Let $f(X)$ be a scalar function where X is an $m \times n$ matrix. The derivative of f with respect to X is an $m \times n$ matrix where each element (i, j) is $\frac{\partial f}{\partial X_{ij}}$:

$$\frac{\partial f}{\partial X} = \begin{bmatrix} \frac{\partial f}{\partial X_{11}} & \frac{\partial f}{\partial X_{12}} & \cdots & \frac{\partial f}{\partial X_{1n}} \\ \frac{\partial f}{\partial X_{21}} & \frac{\partial f}{\partial X_{22}} & \cdots & \frac{\partial f}{\partial X_{2n}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f}{\partial X_{m1}} & \frac{\partial f}{\partial X_{m2}} & \cdots & \frac{\partial f}{\partial X_{mn}} \end{bmatrix}$$

Backpropagation as Chain Rule Application

Steps

epoch

0) initialize weights, bias

loop 100 – 1000

1) for i in range(4):

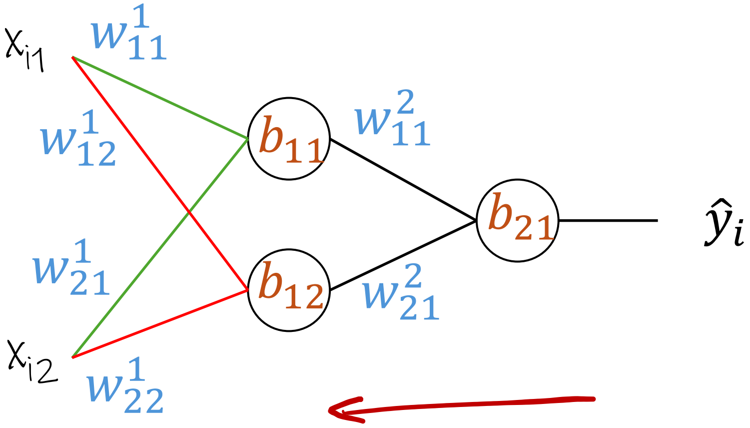
1a) 1 student → FP →

b) Loss (mse)

c) Adjust all w, b

4 times

$$\boxed{w_{new} = w_{old} - \eta \frac{\partial L}{\partial w_{old}}}$$



Probability Distributions for Deep Learning

Probability Review - Basics

- **Random Variables (RV):** Outcomes of random phenomena.
  - Discrete: Countable outcomes (e.g., coin flip).
  - Continuous: Infinite outcomes within a range (e.g., height, temperature).
- **Probability Mass Function (PMF):** For discrete RVs, it gives the probability that the random variable takes on a specific value. P(X = x).
- **Probability Density Function (PDF):** For continuous RVs, it describes the likelihood of the random variable falling within a particular range of values. The probability of X being in an interval [a,b] is $\int_a^b f(x)dx$.

Probability Review – Basics (contd.)

- **Expectation (E[X]):** The average or mean value of a random variable.
  - For discrete RV: $E[X]=\sum xP(X=x)$.
  - For continuous RV: $E[X]=\int xf(x)dx$.
- **Variance (Var[X]):** A measure of how spread out the values of a random variable are from its mean.
  - $Var[X] = E[(X - E[X])^2]$.
- Standard Deviation is $\sigma = \sqrt{Var[X]}$

Key Probability Distributions for DL - Bernoulli

- Describes a single experiment with only two possible outcomes (e.g., success/failure, 0/1), where the probability of "success" is p.
- **PMF:**

$$f(x) = \begin{cases} p, & if\ k = 1 \\ 1 - p, & if\ k = 0 \end{cases}$$

This can also be written as $P(X = k) = p^k(1-p)^{1-k}$

- DL context:
  - **Binary Classification:** Output of a sigmoid layer can be interpreted as p.
  - **Dropout:** Each neuron is independently "dropped" (set to 0) with a Bernoulli probability.

Key Probability Distributions for DL - Categorical

- A generalization of the Bernoulli distribution for discrete random variables with more than two possible outcomes (e.g., rolling a die, classifying an image into one of several categories).
- It has k mutually exclusive possible outcomes, each with a specific probability p_i, where $\sum_{i=1}^{k} p_i = 1$
- **PMF:** $P(X = i) = p_i \; for \; i \; \in \{1, 2, \dots, k\}$.
- DL context:
  - **Multi-class Classification:** Output of a softmax layer gives probabilities for each class.
  - **Language Modeling:** Predicting the next word from a vocabulary.

Key Probability Distributions for DL - Normal (Gaussian)

- The "bell curve" distribution.
- Parameters: Mean (μ) and Standard Deviation (σ).
- PDF:

$$f(x|\mu,\sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

- DL context:
  - **Weight Initialization:** Often initialized from Normal distributions (e.g., Xavier, He).
  - **Generative Models:** VAEs often model latent spaces or outputs as Gaussian.
  - **Loss Functions:** L2 Loss (MSE) is related to maximizing likelihood under Gaussian noise.
  - **Regularization:** Adding Gaussian noise to inputs/activations.

Summary

- Linear Algebra:
  - **Matrix Multiplication:** The fundamental operation of neural networks.
  - **Matrix Calculus / Gradients:** How neural networks learn (backpropagation).
  - **Chain Rule:** The mathematical backbone of backpropagation.
- Probability:
  - **Bernoulli:** Binary outcomes, dropout.
  - **Categorical (Softmax):** Multi-class classification.
  - **Normal:** Weight initialization, generative models, noise.