Data 188 Introduction to Deep Learning
Spring 2026 Eric Kim

Discussion 01

Welcome to Data 188 - we're excited to have you here and have some *deep* conversations with you! This discussion will cover some matrices, vectors, and gradients review.

1. Matrix Multiplication

Compute each of the following matrix multiplications by hand. If the multiplication is not valid, explain why.

(a) $\begin{bmatrix} 1 & 6 \\ 7 & 5 \end{bmatrix} \begin{bmatrix} 3 \\ 0 \\ 2 \end{bmatrix}$

(b) $\begin{bmatrix} 4 & 3 & 6 \\ 2 & 1 & 0 \\ 0 & 5 & 7 \end{bmatrix} \begin{bmatrix} 2 \\ 1 \\ 3 \end{bmatrix}$

(c) $\begin{bmatrix} 0 & 6 & 5 \\ 2 & 3 & 3 \\ 5 & 7 & 8 \end{bmatrix} \begin{bmatrix} 1 & 9 \\ 1 & 4 \\ 3 & 0 \end{bmatrix}$

2. Jacobians

Recall that the Jacobian (or Jacobian matrix) of a vector-valued function is the matrix of all its first-order partial derivatives. If $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^k$ is a vector-valued function with n inputs and k outputs, then its *Jacobian* or *gradient* $\nabla \mathbf{f}$ is an $k \times n$ matrix defined as

$$\nabla \mathbf{f} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_k}{\partial x_1} & \frac{\partial f_k}{\partial x_2} & \cdots & \frac{\partial f_k}{\partial x_n} \end{bmatrix}$$

Numerator convention: Note that in this class, we will use the "numerator" convention rather than the "denominator" convention for the dimensions of the Jacobian. In other courses or textbooks, you may see that if $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^k$ then $\nabla \mathbf{f} \in \mathbb{R}^{n \times k}$. However, in this course, $\nabla \mathbf{f} \in \mathbb{R}^{k \times n}$.

We call it the "numerator" convention because if we call the input vector $\mathbf{x} \in \mathbb{R}^n$ and the output vector $\mathbf{y} \in \mathbb{R}^k$, then each entry of the Jacobian is of the form $\frac{\partial y_i}{\partial x_j}$, where the variable in the numerator corresponds to the output dimension and the variable in the denominator corresponds to the input dimension. In other words, the Jacobian dimensions are $dim(\text{numerator}) \times dim(\text{denominator})$.

Matrix-valued functions: Also note that in this class, we will often work with functions that accept matrices (not just vectors) as input/output. When we define the Jacobian of these matrix-valued functions, the "true" shape of the Jacobian is still 2D, not 3D/4D.

Example: Suppose $\mathbf{f}$ takes an $n \times k$ matrix, and outputs a scalar. Its Jacobian $\nabla \mathbf{f}(x)$ has shape $1 \times n * k$, where the first dimension is "flattened" (eg row-wise). Notably, the shape of $\nabla \mathbf{f}(x)$ is NOT $n \times k$.

However, for convenience, people often represent $\nabla \mathbf{f}(x)$ as a matrix (with shape $n \times k$) rather than as a long vector (with shape $1 \times n * k$). This is fine, as long as you recognize when this happens!

Instructions: For each of the following functions, state the dimensions of the Jacobian and compute it.

(a) $\mathbf{f}(x, y) = \begin{bmatrix} x^3 y \\ \cos^2 y + 5x \\ y e^x \end{bmatrix}$

(b) $\mathbf{f}(\begin{bmatrix} p & q \\ r & s \end{bmatrix}) = \begin{bmatrix} p + 4q \\ \sin(r^2 s) \end{bmatrix}$

3. Deriving the Linear Softmax Gradient Update Equations

Recall from Lecture 2 that we can formulate linear softmax regression as an optimization problem where we want to find the *parameters* (aka *weights*) θ that minimizes the cross-entropy loss function $f(\theta)$ using the gradient descent algorithm. In gradient descent, we iteratively update θ using the *gradient update equation*:

$$\theta := \theta - \alpha \nabla_\theta f(\theta)$$

where α is the *learning rate* (aka *step size*).

In this problem, we will derive the gradient update equation for linear softmax regression step-by-step.

Consider a k-class classification problem where we have:

- Training data: $x^{(i)} \in \mathbb{R}^n$, $y^{(i)} \in \{1, \ldots, k\}$ for $i = 1, \ldots, m$
- n = dimensionality of input data
- k = number of classes/labels
- m = number of points in the training set

Our hypothesis function maps inputs $x \in \mathbb{R}^n$ to k-dimensional vectors $h : \mathbb{R}^n \to \mathbb{R}^k$ where $h(x)$ indicates some measure of "belief" in how much likely the label is to be class i (i.e., "most likely" prediction is coordinate i with largest $h(x)$).

A linear hypothesis function uses a linear operator (i.e. matrix multiplication) for this transformation $h_\theta(x) = \theta^T x$ for parameters $\theta \in \mathbb{R}^{n \times k}$.

To convert the values from our hypothesis function to probabilities (e.g. make sure they are all non-negative values that sum to 1), we apply the softmax function z to $h_\theta(x)$:

$$z_i = p(\text{label} = i) = \frac{e^{h_i(x)}}{\sum_{j=1}^{k} e^{h_j(x)}}$$

Then the *cross-entropy loss* function l_{ce} is defined as:

$$\begin{aligned} l_{ce}(h(x), y) &= -\log p(\text{label} = y) \\ &= -\log z_y \\ &= -\log \frac{e^{h_y(x)}}{\sum_{j=1}^{k} e^{h_j(x)}} \\ &= -(\log e^{h_y(x)} - \log \sum_{j=1}^{k} e^{h_j(x)}) \\ &= -h_y(x) + \log \sum_{j=1}^{k} e^{h_j(x)} \\ &= f(\theta) \end{aligned}$$

Recall the multivariate chain rule for computing the derivative of a composition of functions. Let $h = \theta^T x$. We can compute $\nabla_\theta f(\theta)$ by breaking it down into the product of 2 partial derivatives:

$$\begin{aligned} \nabla_\theta f(\theta) &= \frac{\partial l_{ce}(h, y)}{\partial \theta} \\ &= \frac{\partial l_{ce}(h, y)}{\partial h} \cdot \frac{\partial h}{\partial \theta} \end{aligned}$$

(a) Show that $\frac{\partial l_{ce}(h,y)}{\partial h} = (z - e_y)^T$ where e_y is the one-hot vector with a 1 in coordinate y and 0s elsewhere.

Hint 1: Start by computing $\frac{\partial l_{ce}(h,y)}{\partial h_i}$ and then generalize that into vector form.

Hint 2: How does the expression in Hint 1 differ when $i = y$ versus when $i \neq y$?

(b) Compute $\frac{\partial h_i}{\partial \theta}$.

(c) Recall that in practice, we do not want to compute $\frac{\partial h}{\partial \theta}$ directly because it would take too much compute time and memory. Instead, we can use the results from parts (a) and (b) and take advantage of structure to compute $\nabla_\theta f(\theta)$ more efficiently.

Hint 1: A matrix-vector multiplication Ax can be expressed as scaling the i-th column of A by the i-th entry in x and summing each scaled column. That is, $Ax = \sum_{i=1}^{n} A_{:,i} x_i$ where $A_{:,i}$ is the i-th column of A.

Hint 2: It can be useful to define a vector $c = \frac{\partial l_{ce}}{\partial h}$ and let c_i be its i-th entry.

Hint 3: A matrix of the form

$$\begin{bmatrix} \vdots & \vdots & \vdots & \vdots \\ v_1 u & v_2 u & \cdots & v_k u \\ \vdots & \vdots & \vdots & \vdots \end{bmatrix}$$

where u and v are column vectors can be expressed as $u * v^T$.

Contributors:

- Rebecca Dang.

