Tutorial 02: Mathematical Background

This tutorial aims to provide an overview of core mathematical concepts background needed to understand the concepts of Deep Learning, specifically we will review linear algebra and differential calculus. Most of the topics should already be known from undergraduate lectures, but will be repeated here, as they are crucial to understanding Deep Learning. For a more detailed and more extensive overview we refer to [1].

1 Linear Algebra

For efficient processing as well as representation of data, matrices and tensors are important elements to gain an understanding of Deep Learning algorithms.

1.1 Scalars and Vectors

For a **scalar** we denote a single real-valued number

$$c \in \mathbb{R}. \tag{1}$$

A **vector** of dimension n can be denoted as an object with n real-valued components in an array

$$\mathbf{b} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix} \in \mathbb{R}^n. \tag{2}$$

An important (e.g. to determine the angle between two vectors in Euclidean space) operation on two vectors of equal dimension is the **dot product**, also called **inner product** or **scalar product**

$$\mathbf{b} \cdot \mathbf{c} = \sum_{i=1}^{n} b_i c_i \in \mathbb{R}, \tag{3}$$

which results in a scalar.

The magnitude of a vector in Euclidean space is given by its Euclidean norm (L2 norm)

$$\|\mathbf{b}\|_2 = \sqrt{b_1^2 + b_2^2 + ... + b_n^2}, \tag{4}$$

which can also be expressed using the dot product

$$\|\mathbf{b}\|_2 = \sqrt{\mathbf{b} \cdot \mathbf{b}}. \tag{5}$$

1.2 Matrices

For a **matrix** of dimension $m \times n$ we denote an object with m rows and n columns, totalling mn components.

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix} \in \mathbb{R}^{m \times n}. \tag{6}$$

A **matrix multiplication** is defined as an operation between a matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$ and a matrix $\mathbf{B} \in \mathbb{R}^{n \times p}$ as follows:

$$(\mathbf{AB})_{ij} = \sum_{l=1}^{n} a_{in} b_{nj}. \tag{7}$$

We commonly call this a multiplication of rows times columns, as we can rewrite (7) as a dot product according to (3) between the vector $\bar{\mathbf{a}}_i$ being the ith row of $\mathbf{A}$ and $\hat{\mathbf{b}}_j$ being the jth row of $\mathbf{B}$ as

$$(\mathbf{AB})_{ij} = \bar{\mathbf{a}}_i \cdot \hat{\mathbf{b}}_j. \tag{8}$$

Note that the number n of columns of the first matrix must be equal to the number of rows in the second matrix of the multiplication. The resulting matrix has dimension $m \times p$.

Matrix multiplications are associative, but not commutative, i.e.

$$(\mathbf{AB})\mathbf{C} = \mathbf{A}(\mathbf{BC}), \tag{9}$$

however, in **most cases**

$$\mathbf{AB} \neq \mathbf{BA}. \tag{10}$$

The **transpose**, an operator which flips the orientation, is defined by

$$(\mathbf{A}^{\mathrm{T}})_{ij} = \mathbf{A}_{ji}. \tag{11}$$

In order to ease the use of matrices and vectors we often consider vectors as a special case of matrices. We therefore distinguish between column vectors (matrices with only one column)

$$\mathbf{b} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix} \in \mathbb{R}^{n \times 1}. \tag{12}$$

and row vectors ($\in \mathbb{R}^{1 \times n}$) (matrices with only one row),

$$\mathbf{b}^{\mathrm{T}} = (b_1, b_2, \dots, b_n) \in \mathbb{R}^{1 \times n}. \tag{13}$$

Using this notation, we can rewrite the dot product (3) as a matrix multiplication (7) as

$$\mathbf{b} \cdot \mathbf{c} = \mathbf{b}^{\mathrm{T}} \mathbf{c}. \tag{14}$$

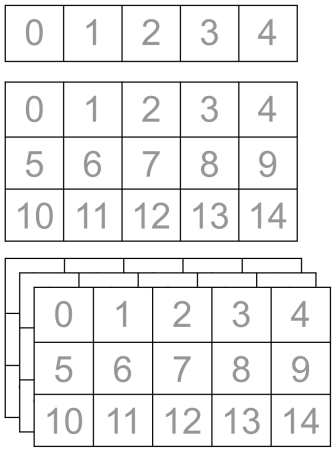

Figure 1: Visualisation of a vector, a matrix and a tensor (top to down). The single numbers represent scalars.

1.3 Tensor

For a **tensor** we denote an object of dimension $n_1 \times n_2 \times \ldots \times n_m$ with $n_1 n_2 \ldots n_m$ elements. We can interpret a tensor as a generalisation of matrices to include higher-dimensional objects, e.g.,

$$\mathbb{A} \in \mathbb{R}^{m \times n \times p}, \tag{15}$$

for which we need three indices to specify an element, i.e. $\mathbb{A}_{ijk} \in \mathbb{R}$. Operations like the multiplication are not as commonly defined as for matrices, but can be defined element-wise, similar as the matrix multiplication (7). A visualisation of vectors, matrices and tensors is depicted in Figure 1.

2 Differential Calculus

Differential calculus is essential in understanding and evaluating the training process of a neural network, which is in most cases through the observation of the gradient descent algorithm.

2.1 Onedimensional Differentiation

Consider a **function** $y = f(x)$ that provides a value y for every possible value of x. For the moment both y and x are to be scalars, making f a onedimensional function ($\mathbb{R} \to \mathbb{R}$).

The **derivative** of a function at a given point is supposed to indicate how sensitive right a function is to small changes of the input value, also known as the **slope** at that point.

More formally, the derivative allows us to know the ratio of the change in a function caused by a small change in the input and the change of the input itself. In the limit of very small changes, we obtain the definition of a derivative as

$$f'(x_0) = \lim_{h \to 0} \frac{f(x_0 + h) - f(x_0)}{h}. \tag{16}$$

For various functions the derivative is not defined for some or even all points, as e.g., our definition would provide infinite values for the derivative (e.g., non-steady functions). However, in Deep Learning, we usually encounter functions for which the derivative exists in (almost) all points and we can write the derivative as a function $f'(x)$ of x itself. Nevertheless, as we are using computers to do discrete calculations we are always prone to some numerical errors, especially when working with very small differences, as indicated in (16)

We will often use the infinitesimal notation for derivatives, where $\mathrm{d}f$ and $\mathrm{d}x$ are infinitesimal small changes in f and x respectively:

$$f'(x) = \frac{\mathrm{d}f}{\mathrm{d}x} = \frac{\mathrm{d}}{\mathrm{d}x} f(x). \tag{17}$$

2.2 Derivatives of basic functions

The most commonly needed derviations are those of the following elementary functions.

- Posynomials

$$f(x) = x^n \quad \rightarrow \quad \frac{\mathrm{d}}{\mathrm{d}x} f(x) = n x^{n-1} \quad \begin{cases} x \in \mathbb{R}, & n \geq 1, n = 0 \\ x \in \mathbb{R} \backslash \{0\} & \text{else} \end{cases}. \tag{18}$$

- Exponential function

$$f(x) = e^x \quad \rightarrow \quad \frac{\mathrm{d}}{\mathrm{d}x} f(x) = e^x \qquad x \in \mathbb{R}. \tag{19}$$

- Natural Logarithm

$$f(x) = \ln(x) \quad \rightarrow \quad \frac{\mathrm{d}}{\mathrm{d}x} f(x) = \frac{1}{x} \qquad x \in \mathbb{R}^+. \tag{20}$$

- Sine

$$f(x) = \sin(x) \quad \rightarrow \quad \frac{\mathrm{d}}{\mathrm{d}x} f(x) = \cos(x) \qquad x \in \mathbb{R}. \tag{21}$$

- Cosine

$$f(x) = \cos(x) \quad \rightarrow \quad \frac{\mathrm{d}}{\mathrm{d}x} f(x) = -\sin(x) \qquad x \in \mathbb{R}. \tag{22}$$

2.3 Differentiation Rules

Most functions appearing in the context of Deep Learning can be derived using the derivatives of elementary functions from the previous section, as well as from the following rules:

- Constant factor rule

$$f(x) = ag(x) \quad \rightarrow \quad \frac{\mathrm{d}}{\mathrm{d}x} f(x) = a \frac{\mathrm{d}}{\mathrm{d}x} g(x), \qquad a \text{ constant.} \tag{23}$$

- Sum rule

$$f(x) = g(x) + h(x) \quad \rightarrow \quad \frac{\mathrm{d}}{\mathrm{d}x} f(x) = \frac{\mathrm{d}}{\mathrm{d}x} g(x) + \frac{\mathrm{d}}{\mathrm{d}x} h(x). \tag{24}$$

- product rule

$$f(x) = g(x)h(x) \quad \rightarrow \quad \frac{\mathrm{d}}{\mathrm{d}x} f(x) = h(x) \frac{\mathrm{d}}{\mathrm{d}x} g(x) + g(x) \frac{\mathrm{d}}{\mathrm{d}x} h(x). \tag{25}$$

- Chain rule

$$f(x) = g(h(x)) \quad \rightarrow \quad \frac{\mathrm{d}}{\mathrm{d}x} f(x) = \frac{\mathrm{d}}{\mathrm{d}h} g(h) \cdot \frac{\mathrm{d}}{\mathrm{d}x} h(x). \tag{26}$$

  Note the differentiation of g with respect to h. The chain rule is the most important key to understanding the learning algorithm backpropagation, i.e., the fundamental method that is used in gradient descent algorithms to train neural networks.

2.4 Partial Derivatives

Until this point we only discussed the case of onedimensional calculus, meaning our functions map from a scalar x to a scalar y. However, in Deep Learning, we will most of the time have to deal with multidimensional problems. This means that our function can map from vectors $\mathbf{x}$ to vectors $\mathbf{y}$:

$$\mathbf{y} = (y_1, y_2, \ldots, y_m) = f(\mathbf{x}) = f(x_1, x_2, \ldots, x_n). \tag{27}$$

In Deep Learning in particular we will often use functions from vectors to scalars ($y = f(\mathbf{x})$).

A multidimensional function can be derived with respect to any of its scalar variables (the components of the vector $\mathbf{x}$), which is called a **partial derivative**. This means at a given point $\mathbf{a}$ we are looking at how the value of the function changes when varying only one of the variables, while keeping the other variables fixed, i.e.,

$$\frac{\mathrm{d}}{\mathrm{d}x_i} f(\mathbf{a}) = \lim_{h \to 0} \frac{f(a_1, a_2, \ldots, a_i + h, \ldots, a_n) - f(a_1, a_2, \ldots, a_i, \ldots, a_n)}{h}. \tag{28}$$

As variables other than x_i are held constant for the derivative, we can see the partial derivative as a onedimensional derivative (if we have a vector $\mathbf{y}$ the same applies to every component of the vector $\mathbf{y}$), where our actual variable is x_i and other variables are treated as constants, e.g.,

$$f(\mathbf{x}) = x_1^2 + x_1 x_2 + x_2^2 \quad \rightarrow \quad \frac{\mathrm{d}}{\mathrm{d}x_1} f(\mathbf{x}) = \frac{\mathrm{d}}{\mathrm{d}x_1} x_1^2 + \frac{\mathrm{d}}{\mathrm{d}x_1}(x_1 x_2) + \frac{\mathrm{d}}{\mathrm{d}x_1} x_2^2 = 2x_1 + x_2 + 0. \quad (29)$$

2.5 Gradient

As mentioned before, the most commonly used optimization algorithm in Deep Learning is called gradient descent (or also method of steepest descent), which obviously gets its name from the word gradient.

The **gradient** of a function is a vector that has the partial derivatives with respect to each of the function's variables as its elements. For our notation, we use the "nabla" symbol ∇ with the vector of the variables as an index:

$$\nabla_{\mathbf{x}} f(\mathbf{x}) = \begin{pmatrix} \frac{\mathrm{d}}{\mathrm{d}x_1} \\ \frac{\mathrm{d}}{\mathrm{d}x_2} \\ \vdots \\ \frac{\mathrm{d}}{\mathrm{d}x_n} \end{pmatrix} f(\mathbf{x}) = \begin{pmatrix} \frac{\mathrm{d}}{\mathrm{d}x_1} f(\mathbf{x}) \\ \frac{\mathrm{d}}{\mathrm{d}x_2} f(\mathbf{x}) \\ \vdots \\ \frac{\mathrm{d}}{\mathrm{d}x_n} f(\mathbf{x}) \end{pmatrix} \quad (30)$$

Note that the gradient is only defined, as we did in the equation above if $f(\mathbf{x})$ has a scalar value.

The gradient of a function, having the same dimension as the vector variable of that function, points into the direction of the fastest increase (steepest direction) of the function at a given point. The negative gradient accordingly points to the "steepest way downhill". This is the key property of the gradient we will be using to optimize our Deep Learning models.

References

[1] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Deep Learning. MIT Press, 2016. http://www.deeplearningbook.org.