Hyperiondev

TASK

Discrete Maths

Visit our website

Introduction

WELCOME TO THE DISCRETE MATHS TASK!

It's imperative that you understand enough maths and stats to grapple with some of the more advanced concepts in this course. Let's get started!

If calculus is the engine of Machine Learning, linear algebra is the body of it. We use vectors and matrices to represent our various models. We will begin by taking a look at the basics and use linear algebra to extend our basic example of linear regression.

WHAT IS LINEAR ALGEBRA?

Let's start with a simpler question: what is algebra? If you think back to your days of high school, algebra is just replacing numbers with letters. If `x + 3 = 8`, then what is `x`? Well, according to what we learn in algebra, it should be 5. We do this kind of stuff everyday in our own programming: we define variables to hold values, and use that to achieve our goals.

Linear algebra extends regular algebra by making x not just a single number, but rather a whole bunch of numbers. This is what we call **matrices** and **vectors**.

Vectors

A **vector** is a list of numbers. In programming terms, you can think of a vector as an array of values. This is just a way of representing data with multiple values. You have used this many times without even realising it!

For the purposes of this task, we will use matrix notation to represent our vectors:

$$\mathbf{V} = \begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix}.$$

There are two different aspects to look at with this vector: the number of elements, and which way it goes. In this case, there are 3 elements, so we call this a **3-vector**. Because it is vertical, like a column, we call it a **column vector**. Let's look at the column vector's sibling, the **row vector**:

$$\mathbf{V} = \begin{pmatrix} v_1 & v_2 & v_3 \end{pmatrix}.$$

And that's all you need to know about vectors for now! See, maths really is easy!

Matrices

What is a matrix? According to the movie The Matrix, it is something that hacks real life and gives you superpowers. Sadly, this is nothing like how matrices work in maths.

A matrix is a set of values written up to represent something. You can think of it as an excel spreadsheet, with rows and columns. Or, framing it as a programming construct, you can think of it as a multi-dimensional array. It is typically written in the form of:

$$\mathbf{X} = \begin{pmatrix} x_{1,1} & x_{1,2} \\ x_{2,1} & x_{2,2} \\ x_{3,1} & x_{3,2} \end{pmatrix}$$

Now, this is a lot to unpack. Firstly, the standard notation for matrices in equations is a capital bold letter. In this case, **X** is our matrix name. You can see that **X** is made up of a bunch of different values $x_{i,j}$. In this notation, i denotes which row we are in, and j denotes which column we are in.

When looking at matrices, it is important to know the shape of the matrix. We write this as $R \times C$, where R is the number of rows, and C is the number of columns. In this case, we have a 3×2 matrix. When dealing with matrix operators, knowing this is incredibly important.

One important note: not all matrices have only two dimensions. In fact, you can have as many dimensions or as few dimensions as possible. In some (highly complex) mathematics, there have been matrices existing with an infinite number of dimensions. While having matrices with 2 dimensions, you can also have a matrix with one dimension. This is actually called a **vector**. You can even have a matrix with no dimensions. This is just a single number. In the world of linear algebra, we call this a **scalar**.

A NOTE ON VECTORS AND MATRICES

Is an apple a fruit? Yes, it absolutely is. Now, is a fruit an apple? Well, not necessarily. It is, by definition, some type of fruit, but not necessarily an apple, as not all fruits are apples.

The same concept applies to vectors and matrices. A vector is a type of matrix with one dimension. That means that all vectors are matrices. However, not all matrices are vectors, as a matrix with more than one dimension is not a vector.

What's important to remember is that everything that applies to matrices applies to vectors, but not necessarily the other way round.

LINEAR OPERATIONS

Okay, so now we know the data we are working with, let's look at some things we can do with this.

A quick note on these operations: some operations require compatibility between matrices.

Addition And Subtraction

Let's start with the easy one. How do we add or subtract two matrices from one another?

Compatibility: Both matrices need to have the same shape. If Matrix **A** is 3×2, then Matrix **B** must also be 3×2.

Keeping this in mind, let's define our Matrices **A** and **B**:

$$\mathbf{A} = \begin{matrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \\ a_{3,1} & a_{3,2} \end{matrix}$$

$$\mathbf{B} = \begin{matrix} b_{1,1} & b_{1,2} \\ b_{2,1} & b_{2,2} \\ b_{3,1} & b_{3,2} \end{matrix}$$

Now, let's say we want to create Matrix **C**, which is **A** + **B**. This new matrix will have the same shape as the first two, and $c_{i,j} = a_{i,j} + b_{i,j}$ for all indices. Therefore, we get:

$$\mathbf{A} + \mathbf{B} = \begin{pmatrix} (a_{1,1} + b_{1,1}) & (a_{1,2} + b_{1,2}) \\ (a_{2,1} + b_{2,1}) & (a_{2,2} + b_{2,2}) \\ (a_{3,1} + b_{3,1}) & (a_{3,2} + b_{3,2}) \end{pmatrix}$$

The same principle applies for subtraction.

Scalar Multiplication

If you remember above, a **scalar** is just a single number. In the world of linear algebra, we denote a scalar variable as a simple x (lower-case, not bold). So scalar multiplication, in English terms, is just multiplying a matrix with a single number. Let's say we want to multiply our Matrix **A** with x. We write this as

$x\mathbf{A}$.

See why the notation is important? Keeping to the notation makes it easy to know what is our scalar, and what is our matrix.

Compatibilities: none.

To multiply **A** by a scalar, you just need to multiply each element in **A** with that scalar. This makes it:

$$x\mathbf{A} = \begin{pmatrix} x(a_{1,1}) & x(a_{1,2}) \\ x(a_{2,1}) & x(a_{2,2}) \\ x(a_{3,1}) & x(a_{3,2}) \end{pmatrix}$$

And this is all there is to it!

Dot Product

The dot product is one of the two ways of multiplying two matrices together. You can either use the **cross product** or the **dot product**. Don't worry about the **cross product**, we are only interested in the **dot product**.

Compatibilities: For two matrices **A** and **B** to be compatible with the dot product, you need to look at their shapes. If Matrix **A** has a shape $x \times y$, then Matrix **B** *must* have a shape $y \times z$. The dot product will then have a shape $x \times z$.

This compatibility simply means that Matrix **B** needs to have a number of rows equal to the number of columns in Matrix **A**. When making the dot product, you will see why.

Let's update our example of defining **A** to something like:

$$\mathbf{A} = \begin{pmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \\ a_{3,1} & a_{3,2} \\ a_{4,1} & a_{4,2} \end{pmatrix}.$$

But now, to add a twist, our **B** is:

$$\mathbf{B} = \begin{pmatrix} b_{1,1} & b_{1,2} & b_{1,3} \\ b_{2,1} & b_{2,2} & b_{2,3} \end{pmatrix}$$

Let's first see if they are compatible. Matrix **A** is of shape 4×2 and Matrix **B** is of shape 2×3. Does **B** have the same number of rows as **A** has columns? Yes. Therefore, they are compatible. We also know that the resulting matrix will be of shape 4×3.

So, now we need to actually compute the matrix. Let's say we are creating a Matrix **C** such that $\mathbf{C} = \mathbf{A} \cdot \mathbf{B}$, where $\cdot$ is our operator for the dot product. Then, $c_{1,1} = (a_{1,1}b_{1,1}) + (a_{1,2} + b_{2,1})$. This can be a lot to wrap our heads around, so just think of it this way: the element of **C** at position **1,1** is each element of **A** in **row 1** multiplied with its corresponding element in **B** at **column 1**. Now you can see why it's important that **B** must have the same number of rows as **A** has columns: this means that there will always be something to multiply together. For a full definition of the dot product:

$$\mathbf{A} \cdot \mathbf{B} = \begin{pmatrix} (a_{1,1}b_{1,1} + a_{1,2}b_{2,1}) & (a_{1,1}b_{1,2} + a_{1,2}b_{2,2}) & (a_{1,1}b_{1,3} + a_{1,2}b_{2,3}) \\ (a_{2,1}b_{1,1} + a_{2,2}b_{2,1}) & (a_{2,1}b_{1,2} + a_{2,2}b_{2,2}) & (a_{2,1}b_{1,3} + a_{2,2}b_{2,3}) \\ (a_{3,1}b_{1,1} + a_{3,2}b_{2,1}) & (a_{3,1}b_{1,2} + a_{3,2}b_{2,2}) & (a_{3,1}b_{1,3} + a_{3,2}b_{2,3}) \\ (a_{4,1}b_{1,1} + a_{4,2}b_{2,1}) & (a_{4,1}b_{1,2} + a_{4,2}b_{2,2}) & (a_{4,1}b_{1,3} + a_{4,2}b_{2,3}) \end{pmatrix}$$

Easy, right? Takes a bit of getting used to, but you will get to understand it more when we use it in practice.

Please Note: The dot product is not mutable. This means that $\mathbf{A} \cdot \mathbf{B}$ is **not** the same as saying $\mathbf{B} \cdot \mathbf{A}$.

For some more ease, let's look at the dot product of two vectors. From earlier, you should remember that a vector is a one-dimensional matrix. Let's define two vectors **P** and **Q**:

$$\mathbf{P} = \begin{pmatrix} p_1 \\ p_2 \\ p_3 \end{pmatrix}.$$

$$\mathbf{Q} = \begin{pmatrix} q_1 & q_2 & q_3 & q_4 \end{pmatrix}.$$

So, you'll notice that **P** is a **3-vector** and **Q** is a **4-vector**. You will also notice that **P** is a column vector, while **Q** is a row vector. Now, let's cheat the system a bit and write these vectors as matrices (because vectors are matrices). That means that **P** is a 3×1 matrix, and **Q** is a 1×4 matrix. This means that they are compatible, because **P** has 1 column, and **Q** has 1 row. This also means that their dot product will be of shape 3×4. Their dot product is:

$$\mathbf{P} \cdot \mathbf{Q} = \begin{pmatrix} p_1 q_1 & p_1 q_2 & p_1 q_3 & p_1 q_4 \\ p_2 q_1 & p_2 q_2 & p_2 q_3 & p_2 q_4 \\ p_3 q_1 & p_3 q_2 & p_3 q_3 & p_3 q_4 \end{pmatrix}.$$

Transpose

The transpose is an operator that works on a single matrix. The general formula for transpose is easy to define. But first, let's get some facts straight about the transpose:

Properties: If Matrix **A** is composed with a shape of $x \times y$, its transpose, denoted $\mathbf{A^T}$, will have dimensions $y \times x$.

This means you just flip it over. Mathematically, we say that the element $a'_{i,j}$ in $\mathbf{A^T}$ is just defined as:

$$a'_{i,j} = a_{j,i}.$$

Let's try an example with actual values:

$$\mathbf{A} = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}.$$

That means that:

$$\mathbf{A^T} = \begin{pmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{pmatrix}.$$

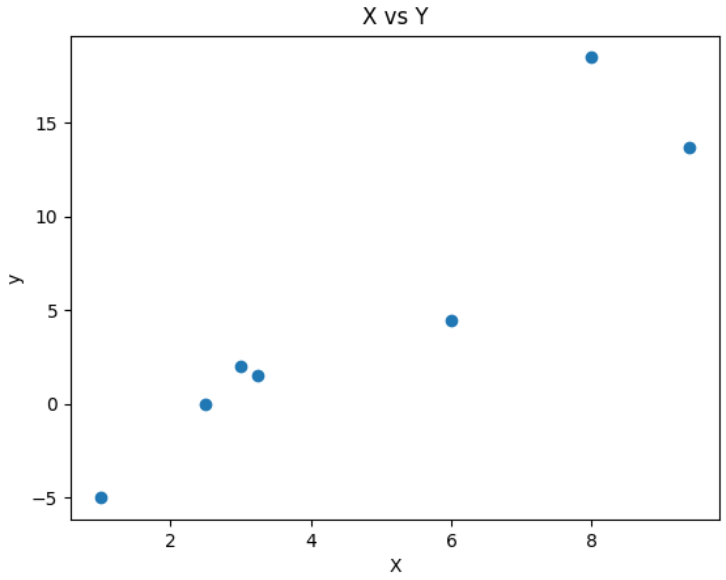
In the world of vectors, this definition is easier. The transpose of any row vector is a column vector, and the transpose of any column vector is a row vector.

MULTIVARIATE LINEAR REGRESSION

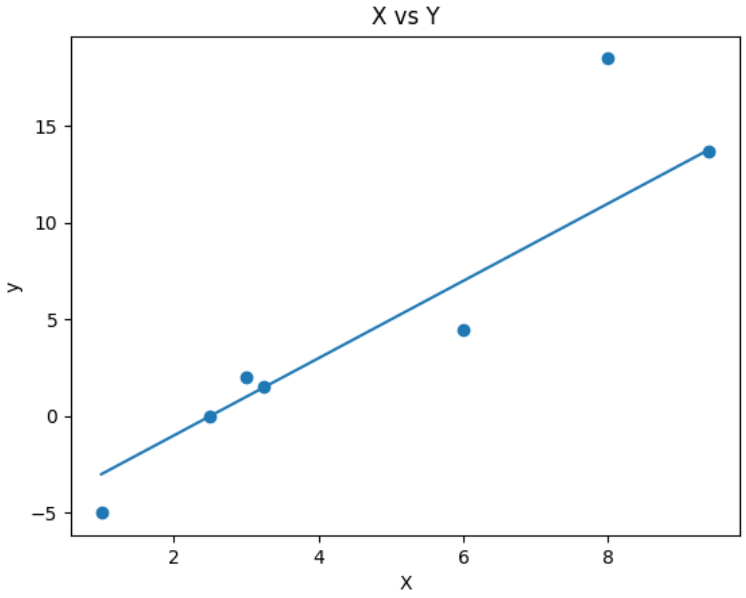
The straight line equation is an important aspect of linear regression as it allows us to model the relationship between multiple variables, make predictions, and extract valuable insights from their collective impact.

Commonly represented as $f(x) = mx + c$ or $y = mx + c$, the straight line equation estimates the optimal values of the slope and y-intercept to best fit the data points.

In this equation, $f(x)$ (or y) represents the dependent variable we aim to predict, x represents the independent variable, m denotes the slope illustrating the change in the dependent variable for each unit change in the independent variable, and c represents the y-intercept.



We would probably want to make a line through the data that looks something like:



The y-intercept (where the line intercepts the y-axis) looks like it is roughly at y = -5. So, now we have two choices: we could calculate, derive and create an update rule for each and every parameter, or we could do this for a single parameter vector (containing all parameters) using linear algebra. Keep in mind, this line could get more complicated with more parameters. A typical Machine Learning algorithm can sometimes have millions of parameters: that's a lot of maths!

This is where linear algebra comes in handy. We just need to define a way of representing this data using vectors and matrices, and then we can extend this to any size of vector and matrix. If you look at our equation for a straight line, you might notice something similar: this is the equation for the **dot product**! Don't believe me? Watch this:

$$f(x) = \begin{pmatrix} m & c \end{pmatrix} \cdot \begin{pmatrix} x \\ 1 \end{pmatrix}.$$

Because the first matrix is of shape 1×2 and the second matrix is of shape 2×1, this will result in a matrix of shape 1×1 – that is just a single number! And, what is this single number? Well, it is $(mx) + (1c)$, which is our y-value!

Crazy, right? This is why linear algebra is so useful in Machine Learning. However, we can do one better: we can calculate all values of y using a single calculation! Let's change our definition a bit: we have a matrix $\mathbf{X}$ of all our x values:

$$\mathbf{X} = \begin{pmatrix} x_1 & x_2 & x_3 \\ 1 & 1 & 1 \end{pmatrix}.$$

Don't worry about the 1's at the bottom: you will see where they factor in for now. Let's define our **parameter matrix P** as the matrix containing our m and c values as a row vector:

$$\mathbf{P} = \begin{pmatrix} m & c \end{pmatrix}$$

We can obtain our **Y** values using:

$$\mathbf{Y} = \mathbf{P} \cdot \mathbf{X}$$

Let's take a closer look at this equation:

$$\mathbf{Y} = \begin{pmatrix} m & c \end{pmatrix} \times \begin{pmatrix} x_1 & x_2 & x_3 \\ 1 & 1 & 1 \end{pmatrix}$$

P is a 1×2 matrix, and **X** is a 2×3 matrix. This means that **Y** will end up being a 1×3 matrix. This is perfect, because there are three x values, which means that there should be three y values. When we apply the formula for **dot product** we get:

$$\begin{pmatrix} m & c \end{pmatrix} \times \begin{pmatrix} x_1 & x_2 & x_3 \\ 1 & 1 & 1 \end{pmatrix} = \begin{pmatrix} (mx_1 + c) & (mx_2 + c) & (mx_3 + c) \end{pmatrix}$$

Wow, that is a powerful notation! You can represent so much using so little! This means that if **X** is a $2 \times n$ matrix, you will get y values with a shape of $1 \times n$, which means that you can have as many x values as you want!

A quick note on the 1's in the X array: this is sometimes just called the bias. This comes around due to the fact that there is a y-intercept that isn't multiplied with any x values.

NUMPY

NumPy is a numerical computation framework that is very commonly used in the field of data science. The code in this framework has been **vectorized**. This is just a fancy way of saying that, while the code runs on the CPU, it is highly optimised. For example, running `np.sum()` on an array of values is much faster than making your

own function to sum the values up. It also helps you to convert Maths equations to code much more easily!

Rate us

Share your thoughts

HyperionDev strives to provide internationally-excellent course content that helps you achieve your learning outcomes.

Think that the content of this task, or this course as a whole, can be improved, or think we've done a good job?

Click here to share your thoughts anonymously.

REFERENCES

IEEE. (1993). *IEEE Standards Collection: Software Engineering*. IEEE Standard 610.12-1990.