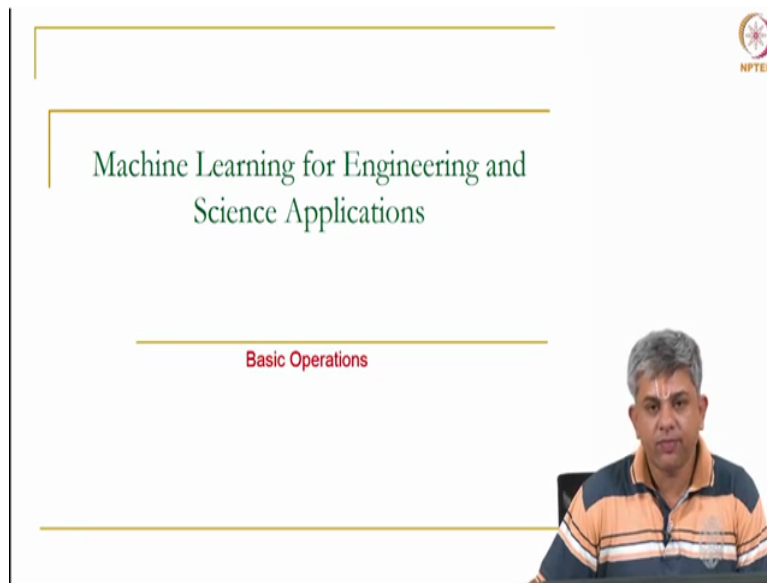


Machine Learning for Engineering and Science Applications
Professor Dr. Balaji Srinivasan
Department of Mechanical Engineering
Indian Institute of Technology, Madras
Basic Operations


(Refer Slide Time: 0:16)



The slide features a white background with a yellow border. At the top right is the NPTEL logo. The main title 'Machine Learning for Engineering and Science Applications' is centered in green. Below it, 'Basic Operations' is written in red. A video inset in the bottom right shows Professor Dr. Balaji Srinivasan, a man with grey hair wearing a blue and orange striped polo shirt.

In the last video we looked at basic notations for scalars, vectors and tensors. In this video we will look at some very basic operations that we can do on scalars, vectors and tensors you should be familiar with most of this already there are couple of special operators that we will be looking at, but most of you are familiar with this even from high school.

(Refer Slide Time: 0:40)



The slide features a white background with a yellow border. At the top right is the NPTEL logo. The title 'Tensor operations covered in this video' is in green. Below it is a bulleted list: 'Addition, Broadcasting', 'Multiplication' (with sub-bullets 'Matrix Product, Dot Product, Hadamard Product (Elementwise multiply)'), 'Transpose', and 'Inverse'. A red line of text at the bottom says 'Skip this video if you know this basic material'. A video inset in the bottom right shows Professor Dr. Balaji Srinivasan, a man with grey hair wearing a blue and orange striped polo shirt.

So the operations that we will be covering in this video are addition and a special type of addition called broadcasting, then we will be looking at multiplication within this we will look at the matrix product the dot product and something called the Hadamard product all it is an elementwise multiplication. Finally matrix transpose and the inverse, okay. So I would suggest that you can skip this video if you already know this, this is very basic material.

(Refer Slide Time: 1:10)

Addition

■ Normal Matrix addition: $A_{ij} + B_{ij} = C_{ij}$
Elementwise
Sizes need to match

■ Broadcasting: $A_{ij} + b_j = C_{ij}$
Matrix *vector* *Matrix*

- Adding a vector to a matrix by repeating the vector
- Done automatically in MATLAB and Numpy

MATLAB

```

>> A = [1 2 3; 4 5 6]
A =
     1     2     3
     4     5     6
>> B = [1 5 6; 1 7 8]
B =
     1     5     6
     1     7     8
>> C = A + B
C =
     2     7     9
     5    12    14
  
```

```

>> A
A =
     1     2     3
     4     5     6
>> b = [2 1 1]
b =
     2     1     1
>> A + b
A + b =
     3     3     4
     6     6     7
  
```

So addition as you know a normal matrix addition is simply you take one matrix and you add the other matrix to it elementwise. For this of course the sizes need to match, so if A is m cross n, B also has to be m cross n and C also has to be m cross n. An example is shown on your screen, this is basically MATLAB output. So you take some simple matrix A in this case 1 2 3 4 5 6 add another matrix B and it gives you the output C, you can see any element if you see the element in C which is 12 this is simply the corresponding element in A added to the corresponding element in B.

Now a special type of addition which we will be using within machine learning this is usually a programmatic thing rather than you really mathematical, but in a program we often add a matrix this is actually just a vector and this gives back a matrix. So let us say you have an m cross n matrix and C also has to be m cross n, what you do in this case is in case either the number of rows or all the number of columns of B matches you tend to make multiple copies of the same vector and add it this is called broadcasting all it is it is adding a vector to a matrix by simply repeating the vectors so that it comes to the same size as the original matrix.

Obviously it can be done only if the vector that we are choosing either has the same number of rows or the same number of columns, okay this is automatically done in MATLAB and Numpy especially in the recent versions of MATLAB, Numpy is a python library which we will be looking at in the third week. So this is done automatically in both of these. Just to show you an example let us say A is the same old 1 2 3 4 5 6 and B is the well in this case the row matrix 1 1 1, then all you need to do is this 1 1 1 gets added to the first row gives you 2 3 4, it also gets added to the second row which is 4 5 6 and gives you 5 6 7.

So this happens automatically within MATLAB, we did not anything special written here I just put A plus B. Similarly in Numpy also this kind of addition is done automatically it is assumed that if you have a size mismatch then you actually have broadcasting going on.

(Refer Slide Time: 4:14)

The slide is titled "Multiplication" and features the NPTEL logo in the top right corner. It is divided into two main sections: "Matrix Product" and "Hadamard Product".

Matrix Product: $C = AB$

- $C_{ij} = \sum_k A_{ik} B_{kj}$
- Sizes must match

Handwritten notes include "m1 x n1" and "n2 x p2" with arrows pointing to dimensions of matrices A and B. A diagram shows a 2x3 matrix A multiplied by a 3x1 column vector B to result in a 2x1 column vector C.

Hadamard Product: $C = A \odot B$

- Elementwise multiplication
- A, B and C must be of the same size

Handwritten notes include "Elementwise Multiplication" and $C_{ij} = A_{ij} B_{ij}$. A diagram shows two 2x3 matrices A and B being multiplied element-wise to produce a 2x3 matrix C.

On the right side of the slide, there are MATLAB code snippets and their outputs:

```

>> A
    1     2     3
    4     5     6
>> B
    1
    1
    1
>> C = A*B
    2     3     4
    5     6     7
  
```

```

>> A
    1     2     3
    4     5     6
>> B
    1     5     6
    1     7     8
>> A.*B
    1    10    18
    4    35    48
  
```

In the bottom right corner, there is a video feed of a male presenter with grey hair, wearing a blue and orange striped polo shirt.

Multiplication, so of course all of us are familiar with the matrix product so this also needs a size match, you know that the first element of the product comes from a dot product essentially of the first row with the first column. Similarly if I have the ijth element this is the ith row and the jth column multiplied together give me the ij element mathematically it can be written in this way summation over k of $A_{ik} B_{kj}$ is equal to C_{ij} we know that as usual if you have m cross n and n cross p, then C's size is m cross p, okay.

Sizes of course must match in the sense that you must have the number of the second element in this case match the first element in this case, okay. So the number of columns and the number of rows have to match. You can also have as usual a matrix multiplying a column vector, okay. So we have seen one such example here. You also have something called the

Hadamard product, this is simply an elementwise multiplication, all we have to do is to say that if A is m cross n and B is m cross n, then C is also m cross n all you are saying is C_{ij} is equal to A_{ij} multiplied by B_{ij} , no summation unlike here, okay you just take the corresponding element here, take one element here multiply it and it gives you the corresponding element there, this is analogous to what we did with addition, okay.

Why is it that the normal matrix product is defined in this weird way it turns out that it has several advantages linear algebra wise we do not have the time to go through that in this course. But it turns out that more often they are not when the products occur they usually occur as matrix products rather than as Hadamard products, but none the less in deep learning and in machine learning we will encounter this kind of elementwise multiplication multiple times which is what that is being written out as a separate thing.

So just as an example is flashed on your screen here, we have taken the same old A, again the same B and you have a corresponding multiplication for example the 2 2 element here is 5, 2 2 element in B is 7 and 2 2 element in C is 35.

(Refer Slide Time: 7:14)

Multiplication (contd)

(a_1, a_2, \dots, a_n) $a = [1 \ 2 \ 3]^T$
 (b_1, b_2, \dots, b_n) $b = [4 \ 5 \ 6]^T$
 $\alpha = 32$

Dot Product: $\vec{a} \cdot \vec{b} = \alpha$

- $\alpha = \sum_i a_i b_i$
- Can also be written as $\alpha = a^T b = b^T a$

Vector notation: $\vec{a} \cdot \vec{b} = \alpha$

Matrix notation: $\alpha = a^T b = b^T a$

$a = [1 \ 2 \ 3]$
 $b = [4 \ 5 \ 6]$
 $\alpha = 32$

$\vec{a} \cdot \vec{b} = a^T b = b^T a$

Used very often. Ensure you can switch between vector and matrix notations

Finally we have the vector product we are not going to look at cross products really as far as this course is concerned that comes more in physics, in machine learning usually we are dealing with dot products. So a dot product you remember is the product simply between two vectors which gives you a scalar, we know that this dot product simply is $a_1 b_1$ plus $a_2 b_2$ so on and so forth up till $a_n b_n$, of course the assumption is a and b are of the same size.

What is more important here is you know one example is given there if a is 1 2 3 and b is 4 5 6 then alpha the dot product is 4 plus 10 plus 18 gives you 32. Now more importantly as far as machine learning is concerned we can usually also write it in this notation, this is what I would call matrix notation, this is vector notation. Since we will deal a lot with matrices it is sometimes useful to see this as matrix notation.

So in this case I can write it as a as really speaking vector should always be represented as columns, so you write one of this as a column vector and one of this as a row vector, this would be a transpose b that also gives you the same product using the matrix product rule 1 times 4 plus 2 times 5 plus 3 times 6, so the dot product can be written as a transpose b so a dot b can be written as a transpose b.

Of course since alpha is a scalar if you take transpose of this whole thing you can also write this as b transpose a. So if I had written it as 4 5 6 1 2 3 it will not have mere difference because I am still making the same product. So we will be using this kind of notation repeatedly, okay so if I do a transpose times b transpose in MATLAB is represented by a prime, okay so a prime here denotes a transpose, okay obviously that is going to give you the same result, okay.

So this kind of thing we will be using extremely often so please get comfortable with this which is denoting a dot product between two vectors as a transpose b or b transpose a and also as a dot b. So all these three will be used interchangeably I am going to write this again a dot b in case a and b are vectors is the same as a transpose b is the same as b transpose a.

(Refer Slide Time: 10:15)

Transpose and Inverse

■ **Transpose:** $B = A^T$

- $B_{ij} = A_{ji}$


```


>> A
A =
     1     2     3
     4     5     6

>> B = A'
B =
     4     5     6
     1     2     3
            
```

■ **Inverse:** $I = A^{-1}A = AA^{-1}$ *A_n is square*

- Above definition is for a square matrix
- Not all square matrices have an inverse
- For non-square cases we can define something called the **pseudoinverse**





A couple of other operators that we will be looking at first is of course the transpose, transpose all of you know is simply written as A^T mathematically all you do is you take sort of across the diagonal you take a mirror image in case it is a square matrix. So B_{ij} will be A_{ji} , okay so if you have the matrix 1 2 3 4 5 6 the same matrix we have been using it gets flipped so if A has size 2 cross 3 A^T becomes a 3 cross 2 matrix and the elements flip.

Inverse is a matrix which gives you when multiplied by the original matrix it gives you back the identity whether you left multiply it or right multiply it we are going to assume here that A is square, so if A is of size n cross n , A inverse is also of size n cross n and when you multiply A and A inverse, inverse is that matrix which when you multiply it by A you are going to recover I , I is of course the identity matrix which has 1 in the diagonal and 0's everywhere else so I will just represent it as a big 0, so this is I you are going to get an n cross n I matrix, okay.

Now it is important to know that not all square matrices have a good valid inverse, it is also possible to find out (matrices sorry) inverses in the case of non-square matrices in such cases we use something called the pseudoinverse which we will be looking at later on in the course.

(Refer Slide Time: 12:08)

The slide is titled "Transpose and Inverse" and features the NPTEL logo in the top right corner. It contains the following content:

- Transpose:** $B = A^T$
 - $B_{ij} = A_{ji}$
- Inverse:** $I = A^{-1}A = AA^{-1}$
 - Above definition is for a square matrix
 - Not all square matrices have an inverse
 - For non-square cases we can define something called the pseudoinverse

On the right side, there is a code execution window showing the following steps:

```

>> A = rand(3)
A =
    0.7814    0.5567    0.7802
    0.2880    0.3945    0.3374
    0.4925    0.0416    0.6079

>> B = A'
B =
    1    4
    2    5
    3    6

>> inv(A)
ans =
    50.2170   -66.1993   -27.6885
    13.3927   -16.8948   -8.9171
   -50.5494    76.9290    34.0937

>> A*inv(A)
ans =
    1.0000    0.0000   -0.0000
    0.0000    1.0000   -0.0000
    0.0000    0.0000    1.0000
  
```

Handwritten red annotations on the slide include circles around the `inv(A)` command and the resulting matrix, and the text "Finite Precision" with an arrow pointing to the small non-zero values in the identity matrix.

In the bottom right corner, there is a video feed of a male presenter with grey hair wearing a blue and orange striped polo shirt.

So just as an example of an inverse if I take a random 3 by 3 matrix and take its inverse, if I multiply the matrix by its inverse you recover the identity matrix, one thing I would like to point out is you will notice that the 0's are not quite 0, you have some decimal places of accuracy this is something very important this is called finite precision, which means just like in your calculator you have only certain number of digits that you can represent accurately,

now depending on the calculator you might have 8 or 10 places, okay and certain other digits are actually uncertain you are not sure what happens after the eighth digit.

In many cases the result that the computer gives will actually be accurate only up till a certain number of digits, typically 16 number of digits, okay if you have what is called double precision. We will be looking at the implications of this later on but you can kind of see it that some of the of diagonal terms it calls positive and some of them it calls negative because you know may be the 10th or 11th digit is 1 or 2 or something of that sort, okay.

So in this video, what we looked at are some basic simple operators we looked at addition, more importantly broadcasting which is an extension of addition see you would do it using over loading the operator, we also saw elementwise multiplication and the other operators are somethings that you should have been already familiar with, thank you.