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

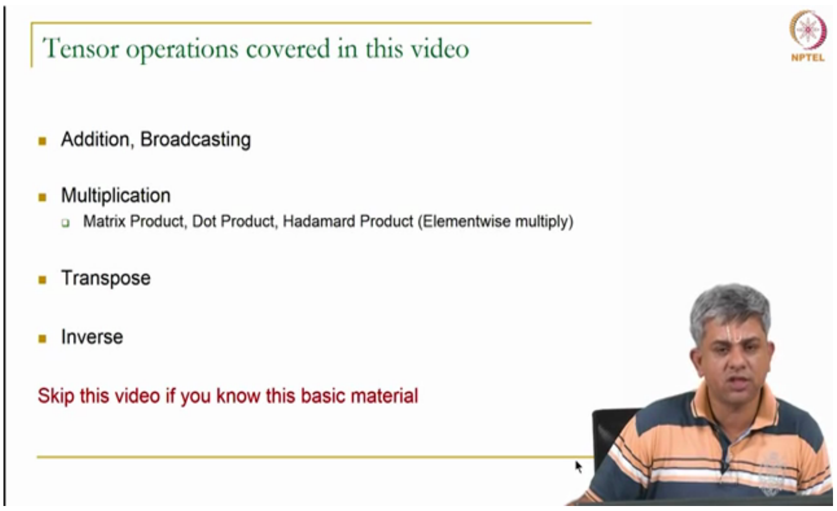
(Refer Slide Time: 0:16)

Machine Learning for Engineering and Science Applications

Basic Operations

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 a 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)



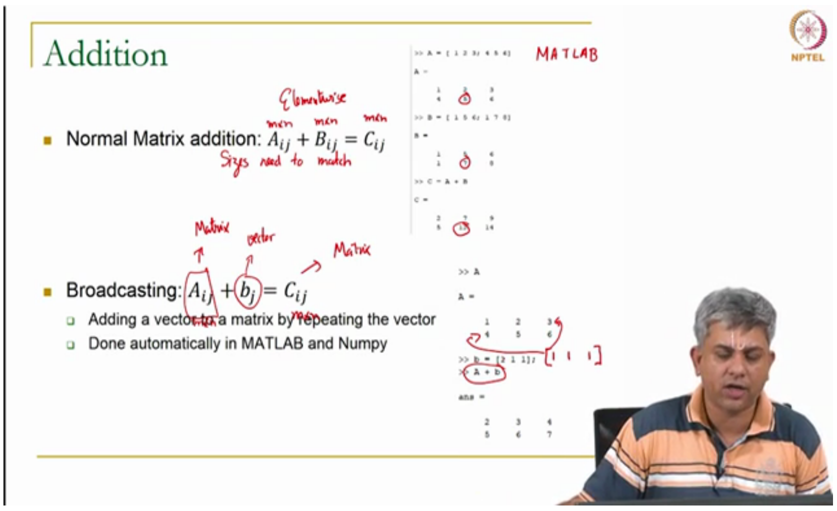
Tensor operations covered in this video

- Addition, Broadcasting
- Multiplication
  - Matrix Product, Dot Product, Hadamard Product (Elementwise multiply)
- Transpose
- Inverse

Skip this video if you know this basic material

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: $\overset{m \times n}{A_{ij}} + \overset{m \times n}{B_{ij}} = \overset{m \times n}{C_{ij}}$ Elementwise; Sizes need to match
- Broadcasting: $\boxed{A_{ij}} + b_j = C_{ij}$
  - 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 = [1 1 1];
>> A + b
ans =
     2     3     4
     5     6     7
```

Matrix
vector
Matrix
[1 1 1]

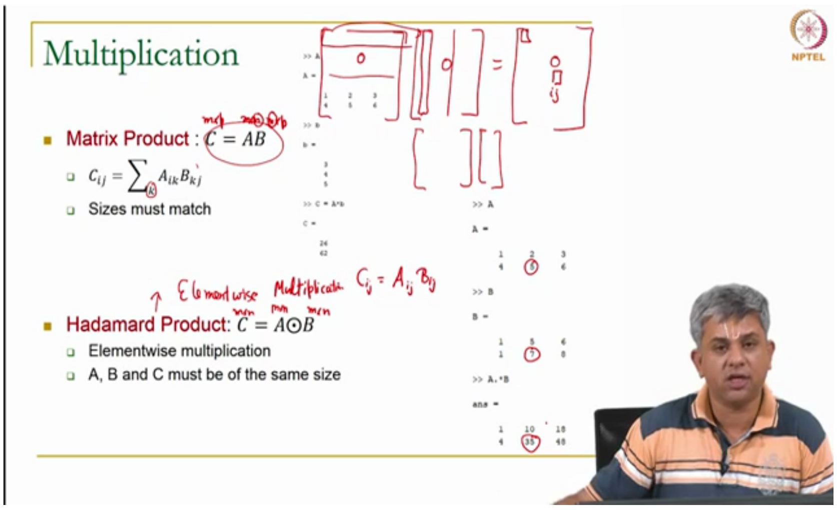
So, addition as you know, a normal matrix addition is simply you take one matrix, and you add the other matrix to it, elementwise. For this of course, the sizes need to match, so, if $\mathbf{A}$ is $m \times n$, $\mathbf{B}$ also has to be $m \times n$, and $\mathbf{C}$ also has to be $m \times n$.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

Now, a special type of addition, which we will be using, within machine learning, this is usually a programmatic thing rather than, you know, 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 a $m \times n$ matrix, and $\mathbf{C}$ also has to be $m \times n$. What you do in this case, is in case either the number of rows or all the number of columns of $\mathbf{B}$ 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, $\mathbf{A}$ is the same old, $\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$, and $\mathbf{B}$ 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 have, anything special, written here, I just put $\mathbf{A} + \mathbf{B}$. 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)



Multiplication

- Matrix Product : $C = AB$
  - $C_{ij} = \sum_{k} A_{ik} B_{kj}$
  - Sizes must match
- Hadamard Product: $C = A \odot B$
  - Elementwise multiplication
  - A, B and C must be of the same size

```
>> A

A =

     1     2     3
     4     5     6

>> b

b =

     3
     4
     5

>> C = A*b

C =

    26
    62
```

```
>> A

A =

     1     2     3
     4     5     6

>> B

B =

     1     5     6
     1     7     8

>> A.*B

ans =

     1    10    18
     4    35    48
```

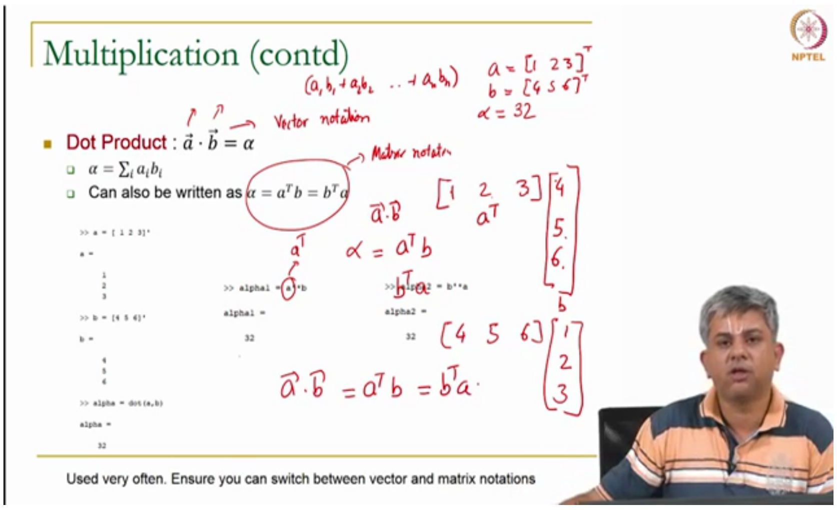
Multiplication, so of course, all of us are familiar with the matrix product. So, this also needs a size match, you know that, the first element of the product comes from, a dot product, essentially of the first row with the first column. Similarly, if I have the ijth element, this is the ith row and the jth column multiplied together, give me the ij element. Mathematically, it can be written in this way, $\sum_k A_{ik} B_{kj}$, is equal to C_{ij}. We know that, as usual, if you have $m \times n$, and $n \times p$, then **C**s size is, $m \times p$, 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 $m \times n$ and **B** is $m \times n$, then **C** is also $m \times n$, 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 why 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, 22 element here is 5, 22 element in **B** is 7, and 22 element in **C** is 35.

(Refer Slide Time: 7:12)



Multiplication (contd)

- Dot Product : $\vec{a} \cdot \vec{b} = \alpha$
  - $\alpha = \sum_i a_i b_i$
  - Can also be written as $\alpha = a^T b = b^T a$

```
>> a = [ 1 2 3]'

a =

     1
     2
     3

>> b = [4 5 6]'

b =

     4
     5
     6

>> alpha = dot(a,b)

alpha =

    32
```

```
>> alpha1 = a'*b

alpha1 =

    32
```

```
>> alpha2 = b'*a

alpha2 =

    32
```

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, $(a_1b_1+a_2b_2, \ldots, a_nb_n)$, of course the assumption is **a** and **b** are of the same size.

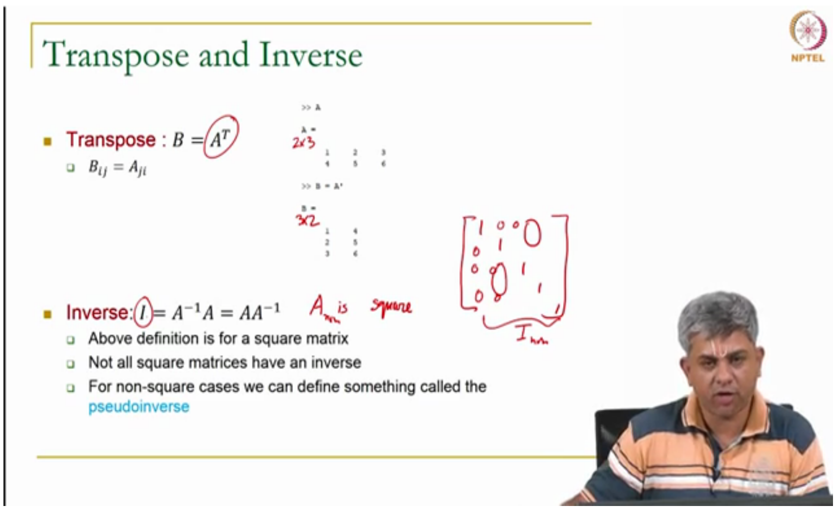
What is more important here, is you know one example, is given there, if **a** is, $[1\ 2\ 3]^\top$, and **b** is, $[4\ 5\ 6]^\top$, then, $\alpha = 4+10+18 = 32$. Now, more importantly, as far as machine learning is concerned, we can usually also write it in this notation, this is what I would call 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 these, as a column vector and one of these, as a row vector, this would be $\mathbf{a}^\top\mathbf{b}$, that also gives you the same product using the matrix product rule, $(1\times4)+(2\times5)+(3\times6)$, so, the dot product can be written as, $\mathbf{a}^\top\mathbf{b}$. So, $\mathbf{a}\cdot\mathbf{b}$ can be written as, $\mathbf{a}^\top\mathbf{b}$.

Of course, since, α is a scalar if you take transpose of this whole thing you can also write this as $\mathbf{b}^\top\mathbf{a}$. So, if I had written it as, $[4\ 5\ 6]\cdot\begin{bmatrix}1\\2\\3\end{bmatrix}$, it will not have made a difference, because, I am still making the same product. So, we will be using this kind of notation repeatedly, okay. So, if I do $\mathbf{a}^\top\mathbf{b}$, transpose in MATLAB is represented by a prime, okay, so a prime here denotes $\mathbf{a}^\top$, 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, $\mathbf{a}^\top\mathbf{b}$ or $\mathbf{b}^\top\mathbf{a}$, and also as $\mathbf{a}\cdot\mathbf{b}$. So, all these three will be used interchangeably. I am going to, write this again $\mathbf{a}\cdot\mathbf{b}$, in case, $\mathbf{a}$ and $\mathbf{b}$, are vectors, is the same as, $\mathbf{a}^\top\mathbf{b}$, is the same as $\mathbf{b}^\top\mathbf{a}$.

(Refer Slide Time: 10:15)



Transpose and Inverse

- Transpose : $B = \boxed{A^T}$
  - $B_{ij} = A_{ji}$
- Inverse: $\boxed{I} = A^{-1}A = AA^{-1}$ $A_{n\times n}$ 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

A =
2x3
     1     2     3
     4     5     6

>> B = A'

B =
3x2
     1     4
     2     5
     3     6
```

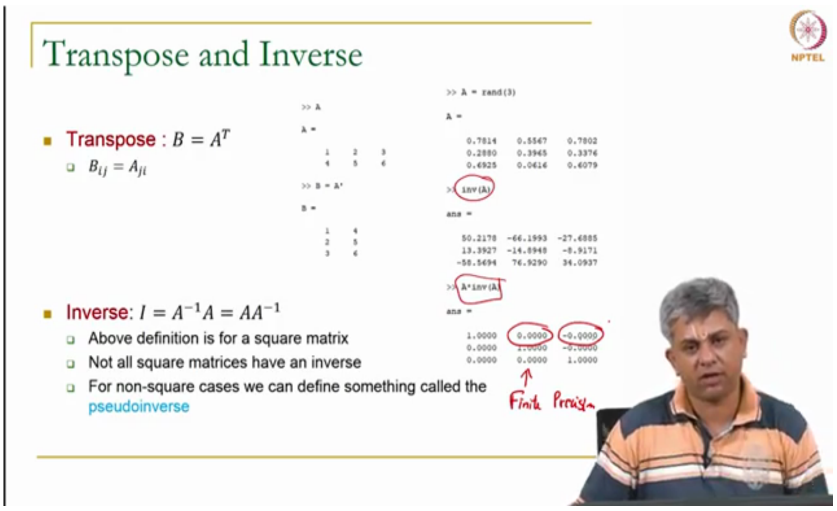
$$\underbrace{\begin{bmatrix}1&0&0&0\\0&1&&\\0&0&1&\\0&0&&1\end{bmatrix}}_{I_{n\times n}}$$

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, $\mathbf{A}^\top$. 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, $\begin{bmatrix}1&2&3\\4&5&6\end{bmatrix}$, the same matrix we have been using, it gets flipped so, if **A** has size 2×3, **B** becomes a 3×2, 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 $n \times n$, **A** inverse is also of size $n \times n$, and when you multiply **A**, and $\mathbf{A}^{-1}$. 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 0s everywhere else, so, I will just represent it as a big 0, so this is, **I**, you are going to get an $n \times n$, **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)



Transpose and Inverse

- 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

```
>> A

A =

     1     2     3
     4     5     6

>> B = A'

B =

     1     4
     2     5
     3     6
```

```
>> A = rand(3)

A =

    0.7814    0.5567    0.7802
    0.2880    0.3965    0.3376
    0.6925    0.0616    0.6079

>> inv(A)

ans =

   50.2178  -66.1993  -27.6885
   13.3927  -14.8948   -8.9171
  -58.5694   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
```

So, just as an example, of an inverse, if I take a random 3×3, 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 0s 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 a 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 eight 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 off 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 overloading the operator. We also saw, elementwise multiplication, and the other operators are some things that you should have been already familiar with, thank you.