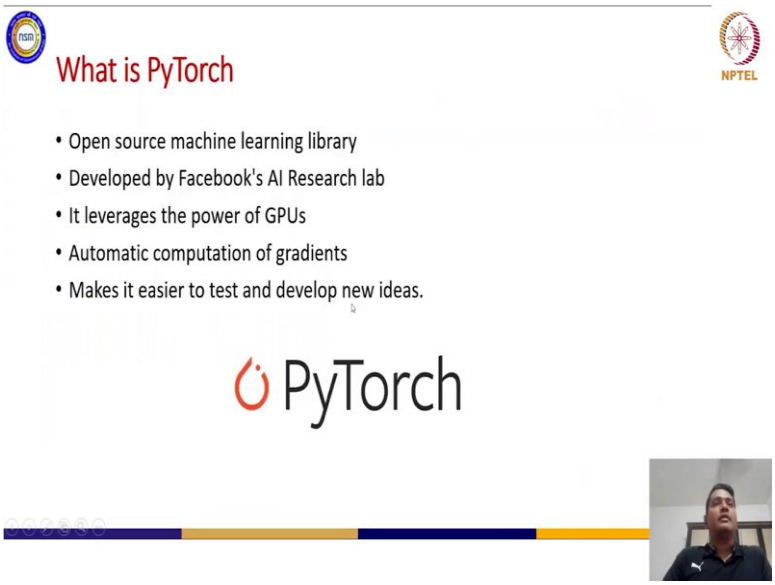
**Applied Accelerated Artificial Intelligence**
**Dr. Satyajit Das**
**Department of Computer Science and Engineering**
**Indian Institute of Technology, Palakkad**

**Lecture - 18**
**Introduction to PyTorch Part - 1**

Good evening. So, today, we will start with Introduction to the PyTorch. So, let us see what we will see today and subsequent classes, we will see more of PyTorch. But today, we will start with some introduction to this framework.

(Refer Slide Time: 00:33)

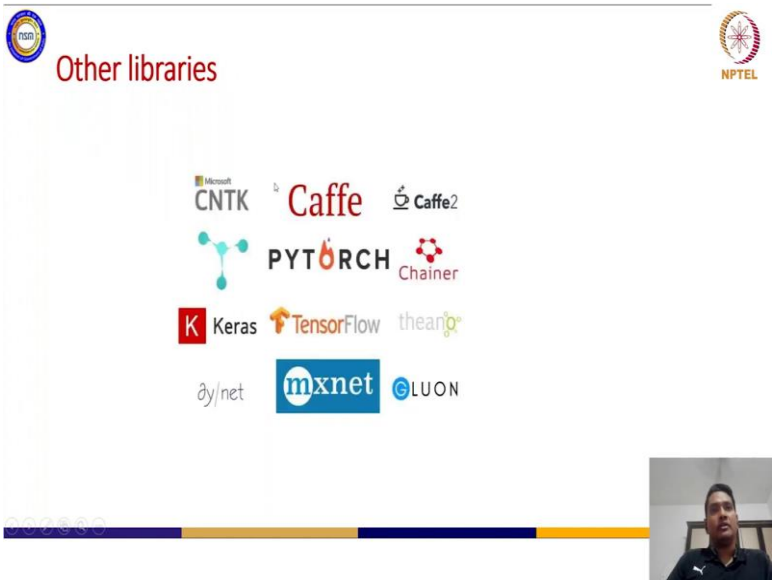



So, this framework is one open source machine learning library and there are other libraries as well. But it was developed by Facebook's AI Research lab and it became very popular and nowadays, it is very popular in the community of researchers as well as developers. Now, why it is so popular? It leverages the power of GPUs.

Basically, it is very very GPU friendly and programming is seamless and automatic computation of gradients. So, when you have gone through the DNNs, so you can see that you need the most important step for training one neural network is computing the gradients and in this framework, the computation of gradients are very easy. It is just one line of code. So, we will see that.

And of course, the structure of PyTorch is very pythonic in nature and the development is very easy. So, it makes it easier to test develop your new ideas basically if you are developing your own neural network or own machine learning algorithms, this very easy to adopt steps and implement your ideas and do experiments tune the parameters and also, easy to test also right. So, we will see all of these flexibilities while using the PyTorch.

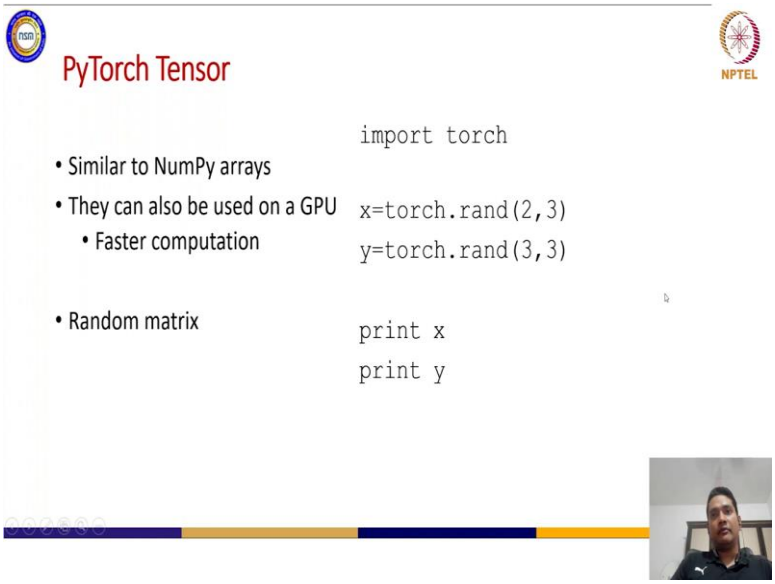
(Refer Slide Time: 02:25)




And so, let us go forward with the other libraries. So, what are the other libraries available instead of PyTorch? We have CNTK from Microsoft, Caffe is there, Caffe 2 one updated version is there, TensorFlow, TensorFlow 2 is there, Keras, theano. So, all these frameworks are also available which does pretty much the same job as PyTorch you will see.

So, it depends on the developers choice which framework to go with and at the end of this course, you will at least see two of these frameworks. So, PyTorch and TensorFlow and you will see the usage of these two frameworks and how to accelerate your deep neural frameworks mostly and how to do that in both the frameworks that we will see.

(Refer Slide Time: 03:22)

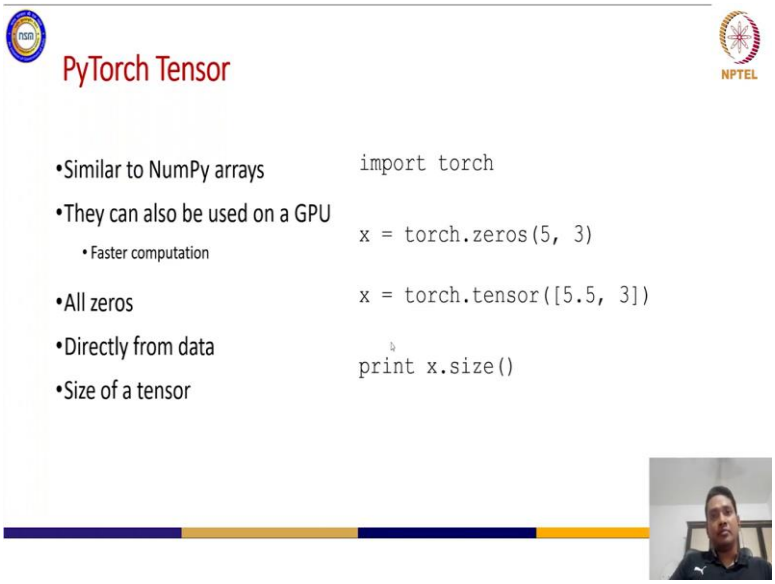



So, the tensors are the most important data structures in PyTorch. So, when you have arrays in NumPy, where you can actually represent a set of data in PyTorch, you have tensors and why the tensors are most important in PyTorch? Because they can also be used in GPUs and arrays cannot be directly used in GPUs rather tensors in PyTorch which is the data structure, we are talking about can be easily adopted into GPU computation and which will give you or provide you faster computational faster training of the neural networks that we will see.

So, talking about these frameworks, we will just first let us discuss about the random matrix that you can have in PyTorch tensor right. So, let us say I want to use PyTorch. So, you just use import torch, torch like will be imported as torch itself and then let us say I want to define one matrix 2 by 3 matrix which is essentially one tensor. So, x =torch.rand(2,3). So, a random 2 by 3 matrix will be generated which will be used as tensor.

So, x is a tensor of 2 by 3 dimension. So, two-dimension it has; one is 2 and another is the size of another dimension is 3, let us say I have defined another tensor y which is torch.rand(3,3). function. So, rand function is essentially used is used to give you random values in the tensors which will be defined in these shapes and sizes right. So, the shape is here two-dimensional and you have 3 x 3, then as the size of this y. You can just print x and y just to see the values you have generated.

(Refer Slide Time: 05 38)

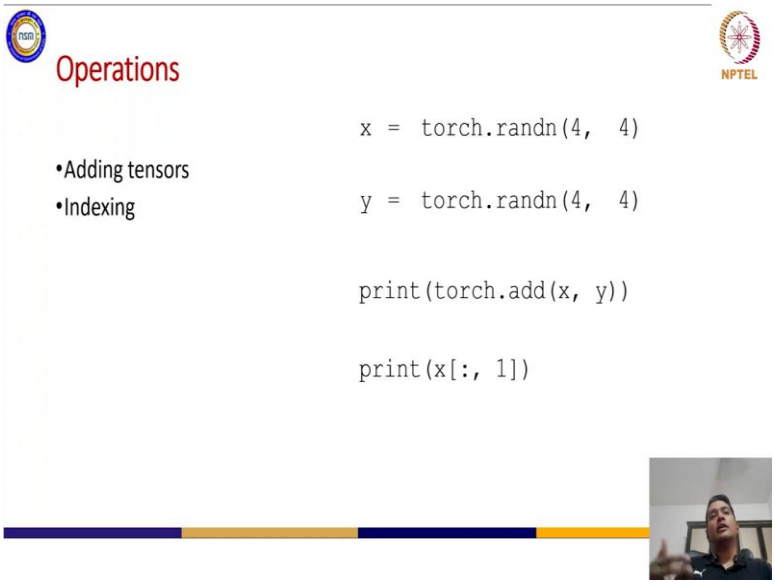



Now, if you want to introduce or if you want to initialize one tensor with all the zero values. So, this is becomes very very important for several machine learning and deep learning algorithms to generate a set of random zeros and the number of zeros you can define in this in the argument. But let us say how we can define such tensors. So, torch.zeros(5,3). will give you set of zeros. Now, how many the dimension we defining this arguments? So, two-dimensional tensor that we are defining here as x with 5 in first dimension, 5 elements and 3 is the second dimensions size.

Now, if you have already one data. So, let us say 5.5 and 3, you have this data already and you want to convert it into a tensor. So, this is essentially one NumPy array and you want to convert it into a tensor. So, torch.tensor method or function will give you the converted tensor from the existing data that you have. So, this function is very handy when you have already a dataset present and you want to convert it into a tensor.

So, that you want you will use that for your GPUs that are available at your instances. Now, to know the sizes of the tensors are very important because while designing your algorithm, while you are working with large tensors.

(Refer Slide Time: 07:36)



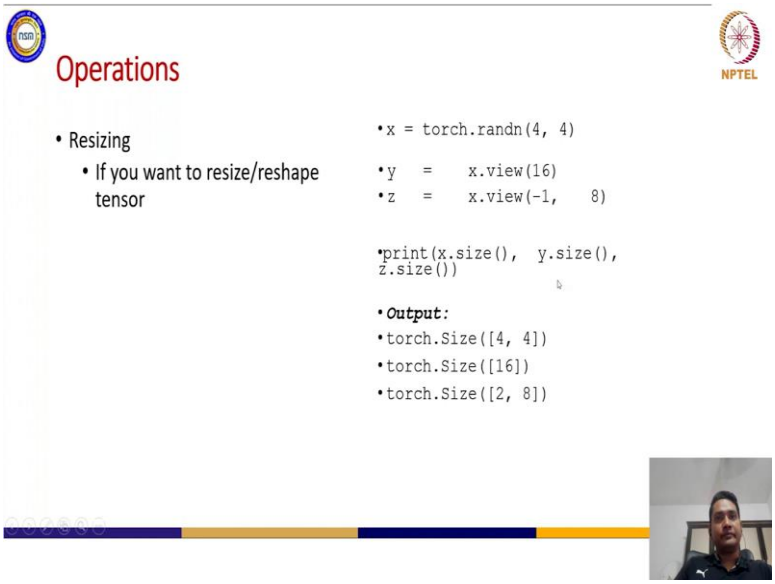

So, at times, you need to get the sizes of different tensors that you are defining and in that case dot size function will give you the sizes of these tensors. So, x.size() will give you the size of the tensor. Now, different operations you can define on the tensors like adding the tensors, indexing the tensors. So, all these are operations, basic operations that you will do while designing your algorithm basically machine learning and deep learning algorithms in PyTorch.

So, x and y here are two tensors we are defining. One is 4x4 random tensor and another is 4x4 random tensor as y. So, x and y two tensors. If we want to add torch.add() function will add these two tensors and remember here of while adding, so while operating on the tensors, any operations you are applying, you need to know whether the sizes are compatible or not.

So, if the sizes are not compatible, then it will return 1 error. So, here as you can see x and y these two tensors having 4 by 4 and 4 by 4, these two are compatible to add and it can be added. Now, indexing is very very essential tool to actually get access to several elements depending on the start and stop of your indexes ok.

So, this is very very similar to your NumPy arrays. So, indexing slicing whatever you say it is same as the array slicing or array indexing that you use for your NumPy arrays and you can directly use slicing on the tensors as well with the same concept.

(Refer Slide Time: 09:17)




We can also resize the tensors and if you want to resize or reshape the tensors. So, let us say we have one random tensor are as x 4 by 4 and we want to resize it to let us say one-dimension. So, this is a two-dimensional tensor that we have defined as x; y is equal to x.view(). So, view function is used to resize your tensor. So, here total 16 elements were there in that two-dimensional tensor. Now, we have one-dimensional tensor where we have 16 elements. So, view has actually flattened that two-dimensional tensor into one-dimensional tensor.

So, this view function will be very very useful, while you are working with neural networks. Be it artificial neural networks or convolutional neural networks, you will be using view function very very randomly. Now, you can also use this minus 1 in as your one argument in your dimensions. So, what does these two? This also comes very handy when you are working with dynamic dimensions. So, let us say you are working with one neural network training, where you need to define your number of batches or maybe number of dimensions that you are using for your input data.

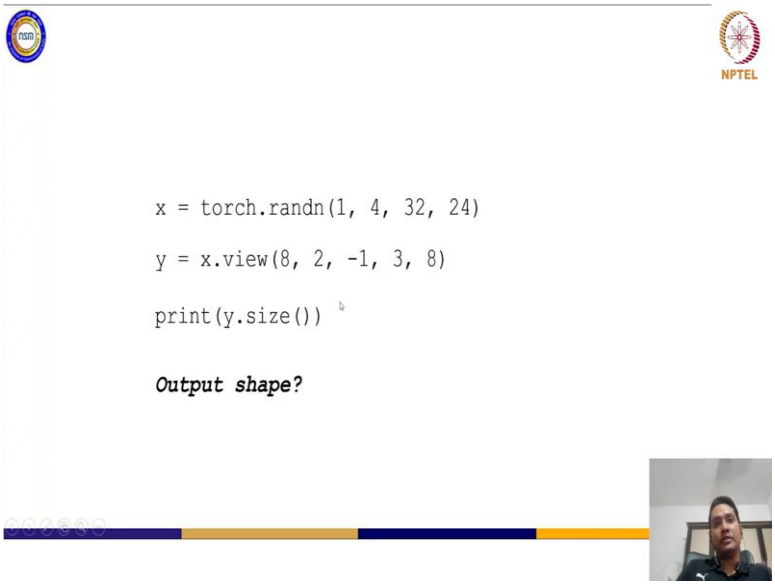
So, if your input data dimensions or let us say number of batches vary, so you cannot hard core the resizing factors. So, let us say I want to resize this x tensor into another two-dimensional tensor as z ok. So, x is one two-dimensional tensor 4 by 4. Now, I want to resize this it into another two-dimensional tensor as z. Now, x.view() function I will

call, but here I will not hard code the first dimension let us say; I want to hardcode only the second dimension.

So, in the second dimension, I want 8 as the size and the first dimension, I want it to how to calculate and reshape it automatically and that is when we will use minus one as the argument. So, it will automatically when you will print the x.size() and y.size() and z.size(). You can see the torch size for x is 4 by 4; torch size for y is 16 and torch size for z is your 2, 8.

So, basically 2 is automatically calculated. So, this is a very simple example, but when you are working with multi-dimensional tensors, let us say four dimension or eight dimensional tensors, then figuring out one dimension with randomly or dynamically varying input sizes, it is very very handy.

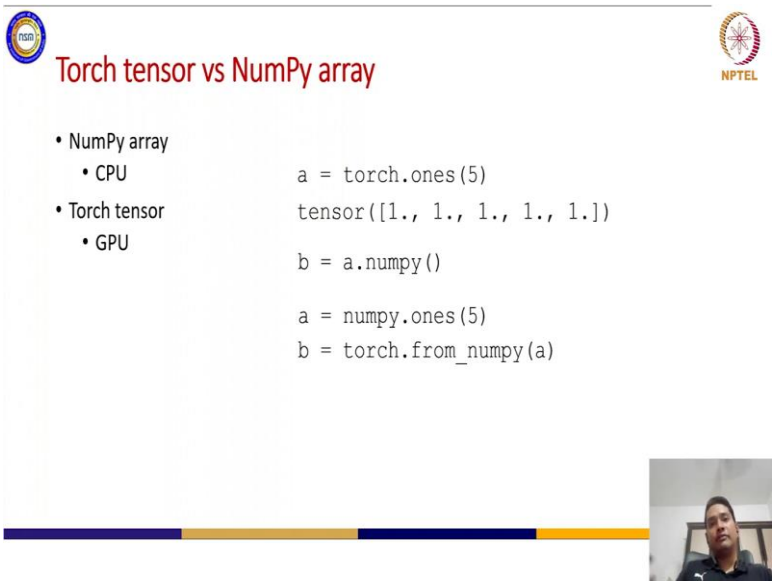
(Refer Slide Time: 12:19)




Now, let us say you have one tensor here, four-dimensional tensor as I was mentioning, where x is randomly initiated four-dimensional tensor; when we have 1, 4, 32 and 24 as dimension sizes for 1, 2, 3, 4 and you want to resize it let us say first dimension you want to keep it as 8, second dimension you want to keep it as 2 and third dimension you do not know because let us say your input is varying for different dimensions and third and fourth dimension, you want to keep it as 3 and 8.

So, what will be the third dimension? So, it will automatically be calculated depending on the values you are getting as input tensors and it will be automatically revised and added. So, here if you want to print the y.size(), then you will get what the output shape. So, try it out and you will see what the output you are getting whether you are calculating it perfectly or not.

(Refer Slide Time: 13:21)




Now, there are tensors and there are arrays. So, as I was mentioning that NumPy which is very very basic library that PyTorch uses for let us say array and list data structures mostly they are designed for your CPUs and while you were working with tensors, this is designed for your GPUs. So, you need to know the difference between these two. But the scope of tensors is not only confined to your GPUs; you can run also the tensors into your CPUs also. So, that we will see in this session as well, how we can run both in CPU and GPUs.

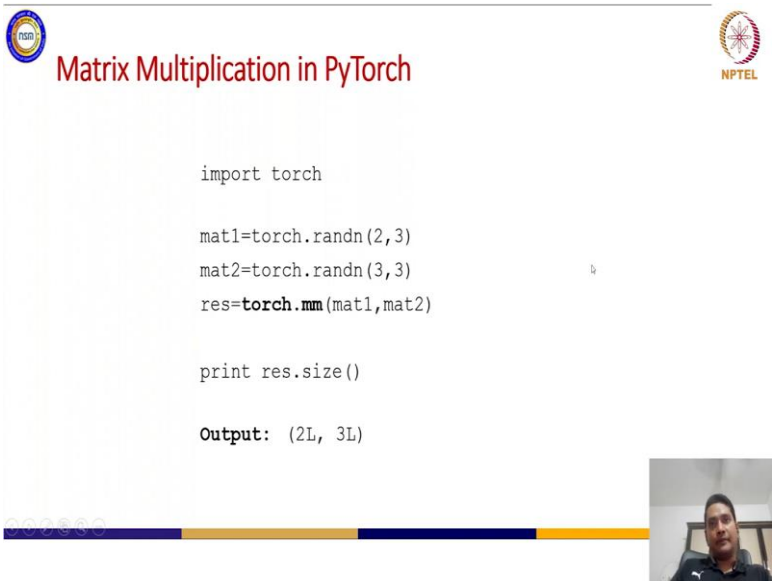
Now, let us say I have one tensor here for randomly sequenced. So, here we are initiating one tensor with 5 ones and this is the value of a which is one tensor with 5 ones as you can see. Now, b is one NumPy array, I want to convert the tensor into one NumPy array. So, we can go in both the dimensions or both the realm of array and tensors by using this simple conversion.

So, if you want to go from tensor to array, a.numpy() will actually convert your tensors into your array or numpy arrays and if you have let us say I have one numpy array which

is set of ones equal to numpy.ones(5). So, as you can see that the PyTorch torch tensors and numpy arrays are almost similar in structure or definition.

So, that is why it is very very popular. Now, as you can see here I have one set of numpy array. Now, I want to convert it into tensor. So, from numpy, from underscore numpy function will transform your array into the tensors. So, if you have already data available as array elements in your program, then you can convert it into tensors using this simple function and you can leverage the flexibility of available GPUs to run this ok.

(Refer Slide Time: 15:53)

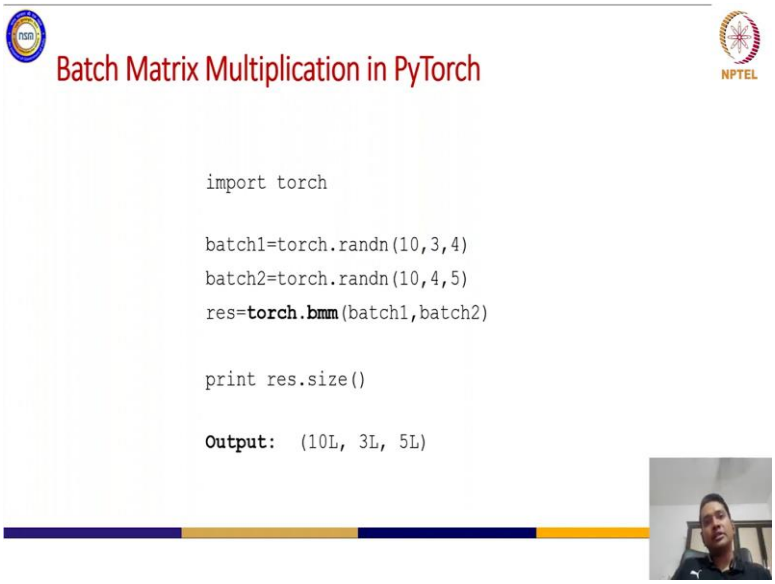



Now, I will talk about matrix multiplication in PyTorch which is very very fundamental basic and very very useful and widely used operation in AI applications. So, first, we will just report the torch library. So, we are defining or initiating two matrices. So, mat1 first matrix having 2 by 3 elements with random values; second matrix, we are having 3 x 3 as the random values.

So, as you can see when we are trying to matrix multiply these two, we need to be again compatible for multiplication. So, you cannot multiply 2 x 4 and 3 x 3 matrix right. So, the columns and the rows for these two matrices need to be same and that is when you can do or apply this dot mm function to multiply these two matrices and these matrix multiplication is on the tensors not on the arrays.

So, this matrix multiplication you can actually do it or you can use GPUs to do this matrix multiplication. But usually we do not do and you can check the size of this result and the size of the result will be 2 by 3 because you are multiplying 2 by 3 into 3 by 3 matrix. So, your resultant matrix will be 2 by 3.

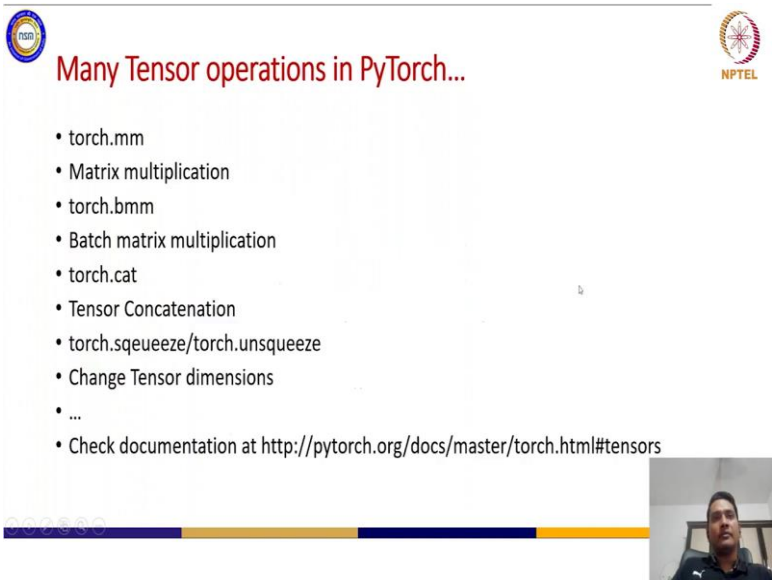
(Refer Slide Time: 17:28)




Now, we do mostly the batch matrix multiplication. So, when you are working with machine learning applications, you do not do single matrix multiplication right. So, batch matrix multiplication is widely used mostly and for that, we need bmm function to apply on the batches. Now, how you define batch matrices? So, let us say batch1 is we want to define one batch matrix. So, the first dimension is basically the batch size. So, 10 is the batch size; that means, 10 samples with this special structure, what is the special torch structure? 3 by 4.

So, 10 3 by 4 matrices you have and another 10 4 by 5 matrices you have as batch 2 and you want to multiply these two batches; batch1 and batch2, bmm function batch1 and batch2. And if you want to see the resultant batch matrix size, now the size will be 10 because 10 matrices you have multiplied with 10 other matrices. So, resultant you will get 10 matrices. So, 10 samples, 10 samples, resultant 10 samples you will get. 3 by 4, 4 by 5 you have done the matrix multiplication for. So, you will get 3 by 5 as the resultant matrix dimension.

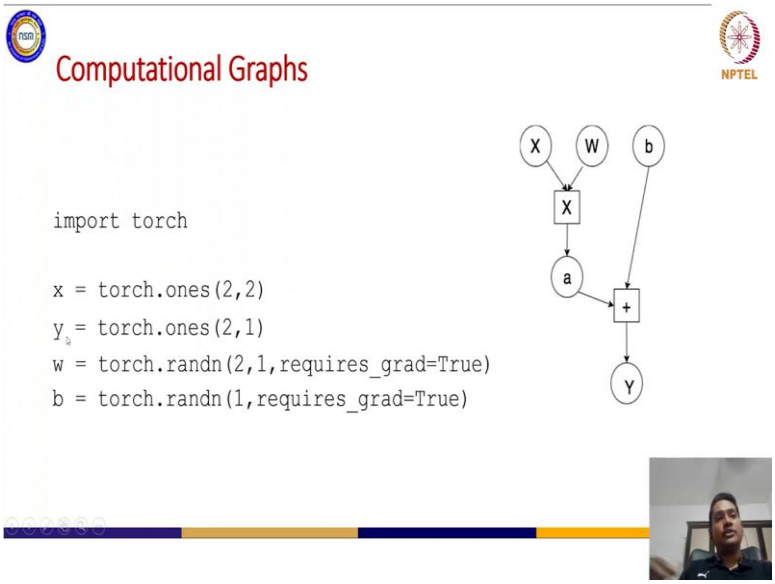
(Refer Slide Time: 19:01)




Apart from batch matrix multiplication, there are other operations also which are very very important for your AI applications like concatenation. So, you can concatenate two tensors, you can actually squeeze or un squeeze two tensors. So, let us say you want to reduce the dimension of one tensor, let us say from four dimensional to your two-dimensional tensors you want to squeeze, you want to then use squeezed function, you want to increase the dimension, you will use un squeezed function and so on and so forth.

So, for each definition of these functions, we can go to PyTorch.org/docs/master/torch.html#tensors. So, here you can define get the definition of each functions or each operations that you can apply on the tensors and their descriptions and their usage with examples.

(Refer Slide Time: 20:05)




Now, while working with neural networks, mostly from the deep neural networks perspective, computational graphs are very very important. So, computation graphs are essentially the graphs which defines the neural network operation. So, here x is input tensor, w is your input weights. So, these are the parameters, you want to; you want to get the value after training this neural network form and b is the bias. So, let us say these three are the input to your neural network. So, I hope you have revised the neural network concept to understand the computational graphs. Now, a is.

So, essentially what you are doing here is that x * w + b is your output tensor right. So, this is a linear operation you are doing. So, w * x + b and w is your parameter set or parameter tensor that you want to get the value from the training. So, how you will do the training? Once you have the prediction as the output here y and once you have the ground truth let us say y', then you compute the loss and depending on the loss, you compute the gradients with back propagation and then, you update the weights and biases with this gradient values and then, again you do the forward pass.

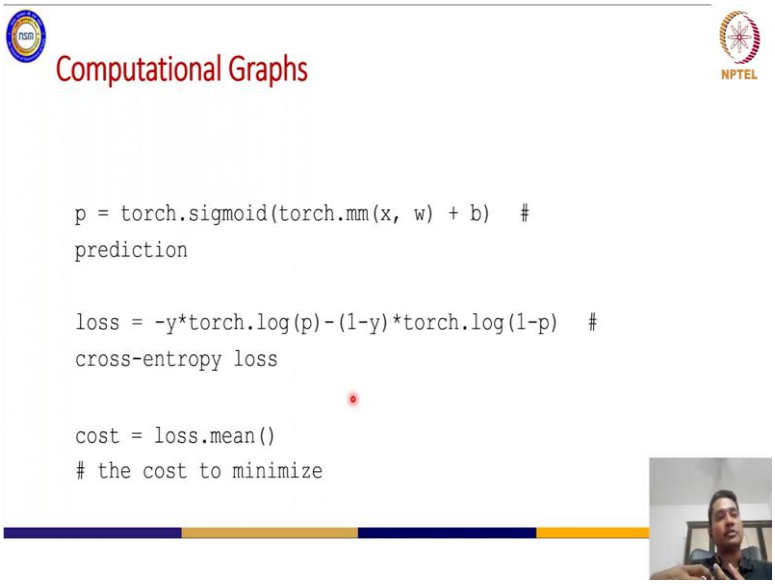
So, once you do the flow complete from x w b to y that is called one forward pass right. So, to define this forward passes in PyTorch, let us say we have x as our input tensor. So, one input tensor we are defining as 2 by 2 as all ones. So, basically this is we are initiating the values. For your training, you will get the values from your inputs. Now, we have y as the outputs. So, this is also we are initializing as 2 by 1 ok and now, the

value for the parameters, we are defining it as torch.rand (2,1)we are initiating requires underscore grad describes true.

So, once you have defined the neural network or you want to go to define the neural network, you need to know which are the input variables of input tensors and which are the parameter tensors or weight tensors. So, basically w and b are the parameters which you want to update with each training iteration and at the end, when you have minimized the loss then you will get the finalized w and b values right and that is your target. So, that means, with we want to compute the gradients for these parameters.

So, w and b parameters you want to compute the gradients. So, requires grad equal to True. So, this grad will let the PyTorch framework know that w and b are the parameters that we are working with and when we will compute the gradients, these parameters will get updated not x and y. Because by default require grad is false and we did not define the true flag here for x and y which is the input and output tensor right.

(Refer Slide Time: 24:08)

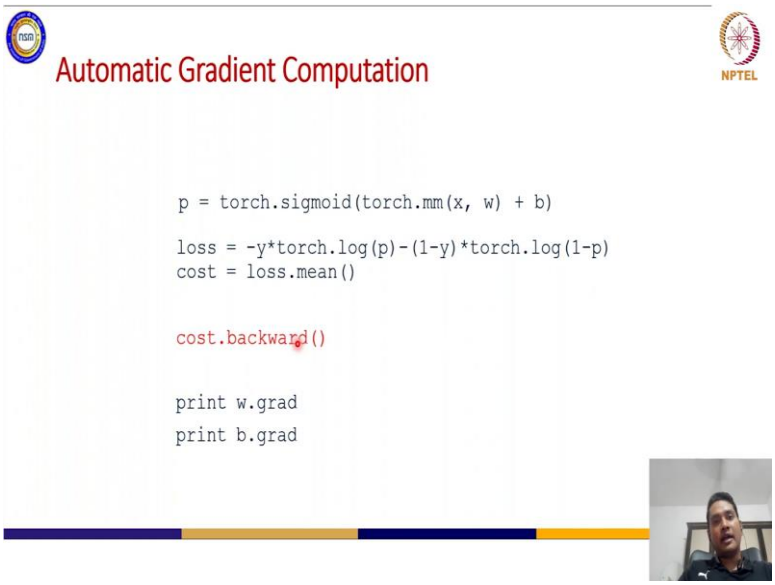



So, once we have the computational graph defined, so basically here p is the prediction of the output and as you can see x into w, we are doing it matrix multiplication plus b. So, this is the linear operation that we are doing and then, we are applying the sigmoid function. So, sigmoid function is essentially the non-linear function you want to apply onto your output of the linear function and then, you are applying this prediction for computing your loss right.

So, this is the p that you are interested in computing ok. Until you compute the loss, so which is let us say here we are trying to compute the cross entropy loss which is minus y into. So, this is the positive loss and this is the negative loss prediction. So, this is the loss basically you will not use any formula for that, you have already library functions that are available to compute the losses. So, we will see such use of the loss functions with just calling the function. Here, we are defining the function to compute the loss.

And then, once you have computed the loss for let us say many losses, you just compute the mean or average for all the losses which is the cost and once the cost is defined, then you are trying to minimize the cost function or cost value to get the prediction more concrete right.

(Refer Slide Time: 25:48)



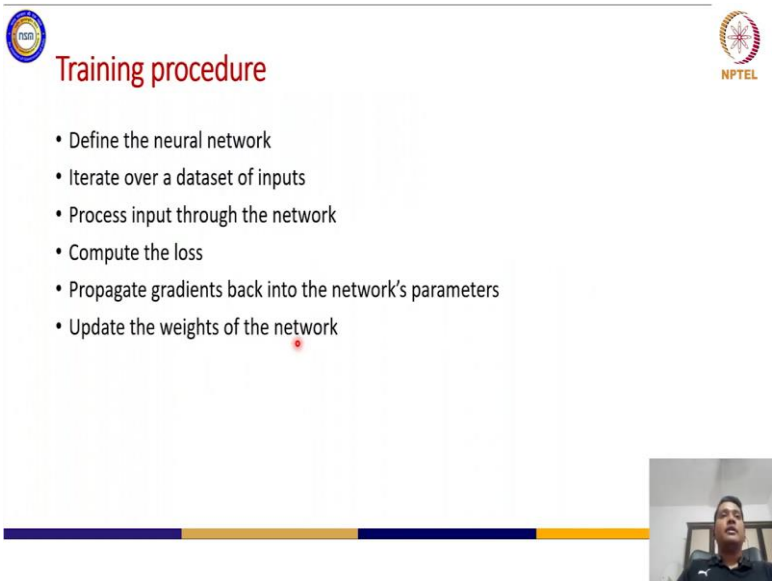

So, that means, with your gradient computation, you will update the parameters which you have defined as w and b and then, you are actually use or compute the gradients and update those parameters using what that you will see now. So, first we have completed the predictions, then the loss and with this cost, just calling this backward function we will compute the gradients of it and computing the gradients is the most complex function or complex steps for training one neural network.

So, how we are training the neural network? We have one set of input tensors or set of weights and biases. So, basically the parameters. So, parameters, we are combining with combining the weights and biases; together, we have parameters. And then, we are

applying this linear function, the non-linear function; then, we are computing the loss with the ground root. And then, we are computing this backward pass which is essentially computing the gradients.

So, this backward pass will actually run your back propagation and compute the gradients with respect to all the parameters that we have defined required grad equal to true. Now, if you want to see what the gradients for each parameter, you can print w. grad and b.grad to see the weights.

(Refer Slide Time: 27:27)




So, in a nutshell, what are the training procedure? So, that we will use PyTorch functionalities to define the training of neural networks. So, and of course, when you will define deep neural networks; that means, when number of layers. So, this whatever we have seen as the computationals graph so far that is just the one layer and if you have deep neural networks; that means, the number of layers will be increased.

And you might have also convolutional neural networks, where several of these layers will be convolutional operation. So, convolutional operation also inherently uses matrix multiplication because you can represent your convolution operation as matrix multiplication and that is why batch matrix multiplication is very very important because all these deep neural networks that we will train that we will use batch data. So, let us define the training procedure.

So, each step here you can see define first the neural network, then iterate over the data set of the inputs. So, this is very very important step, where you get or prepare the data and then, you iterate over the data set to feed to this model that we have defined in this first step. Because this neural network model will take all the training all the training data set that you have processed and it will actually train your neural network and update the parameters until or unless you get the desired accuracy.

So, then what the step is the process of the input through the network, then computing the loss. So, computing the loss we have seen how to compute the loss, but with respect to predefined functions how we compute the loss and how will define it inside a PyTorch framework specifically that we will see in the next and then, once the loss is computed, then you can propagate the gradients back into your network parameters to update them. And last step is that to update the weights of the network.

So, this is just the one step of the entire training process of your neural network and you will continue this step until or unless you get the desired accuracy or predefined number of steps that you want to have.