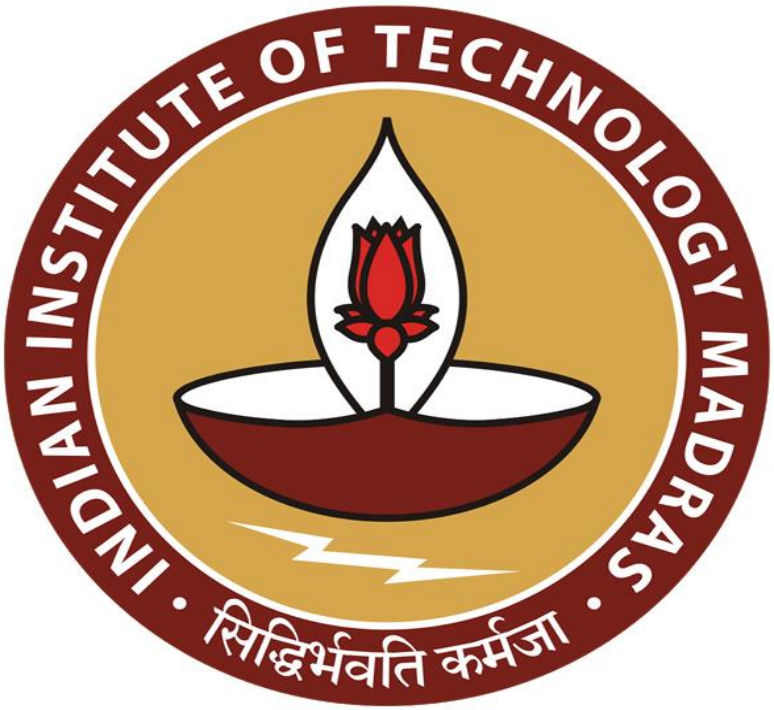
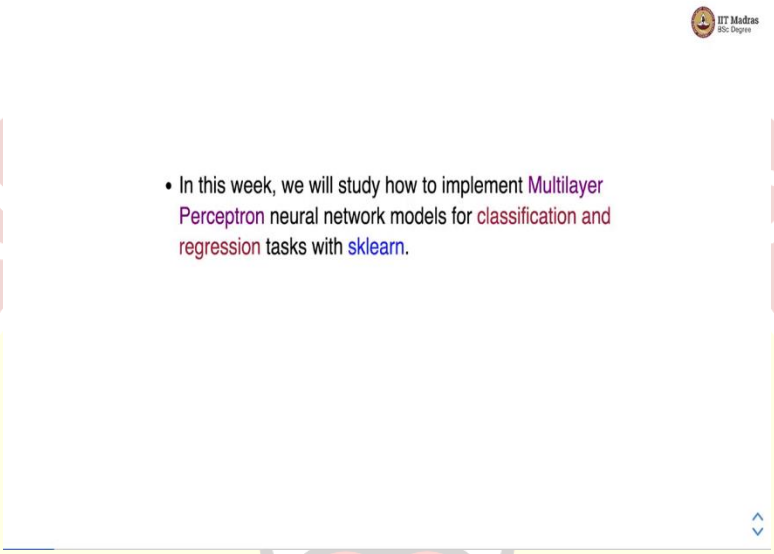


IIT Madras

ONLINE DEGREE

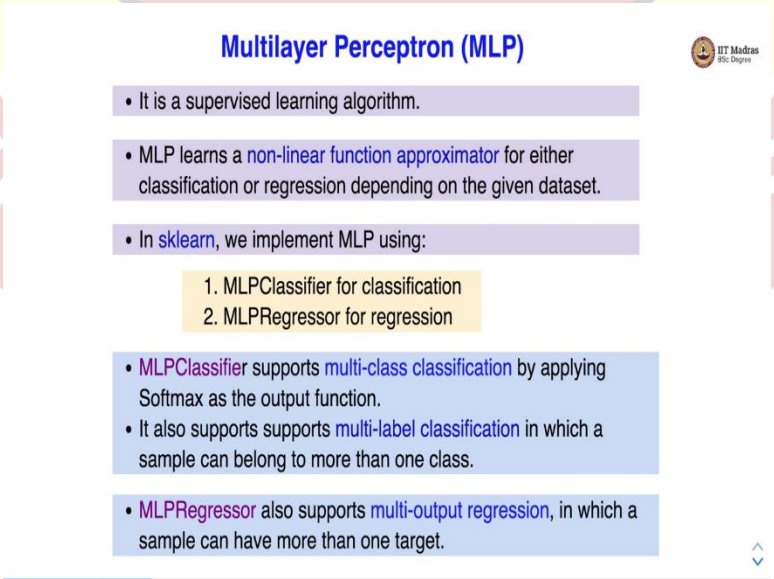
Neural Networks
Professor. Doctor. Ashish Tendulkar
Indian Institute of Technology, Madras
Machine Learning Practice

(Refer Slide Time: 0:14)



Namaste! Welcome to the last week of Machine Learning Practice Course. In this week, we will study how to implement artificial neural network or multilayer perceptron neural network models for classification and regression task with sklearn.

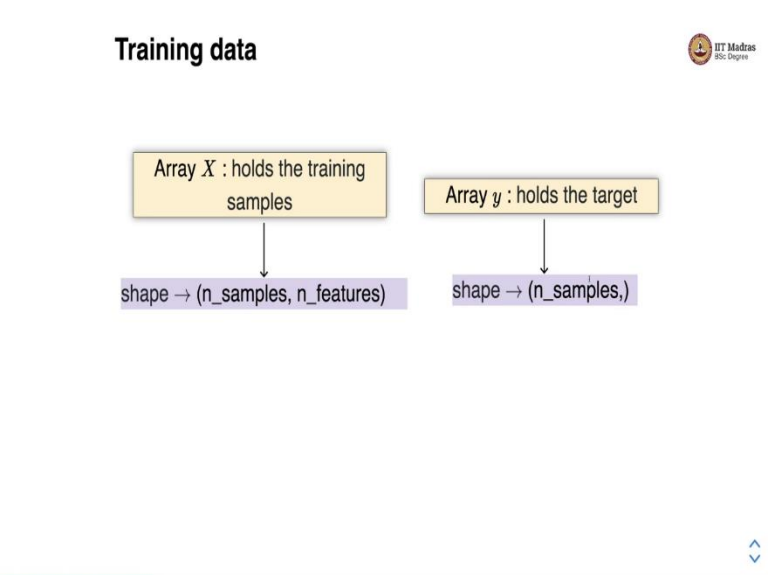
(Refer Slide Time: 00:25)



Let us look at multilayer perceptron. So, there are different names for artificial neural network it goes by the name multilayer perceptron, or feed forward neural network. So, we will be using multilayer perceptron as the name throughout this slide deck. So, multilayer perceptron is a supervised learning algorithm. It learns a nonlinear function approximator for either classification or regression tasks depending on the dataset.

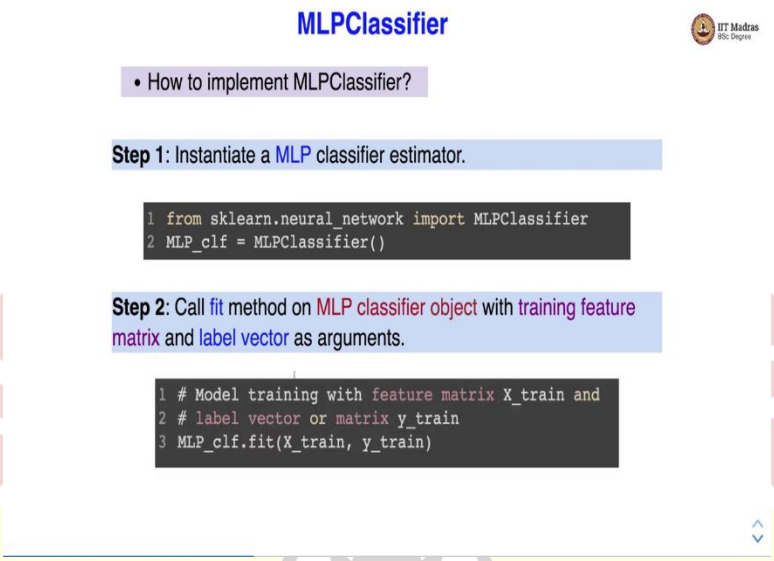
In sklearn, we implement MLP using MLPClassifier, and MLPRegressor for classification and regression respectively. MLPClassifier supports multi-class classification but applying softmax as the activation in the output layer. It also supports multi-label classification in which a sample can belong to more than one class. MLPRegressor also supports multi output regression in which a sample can have more than one target values.

(Refer Slide Time: 01:33)



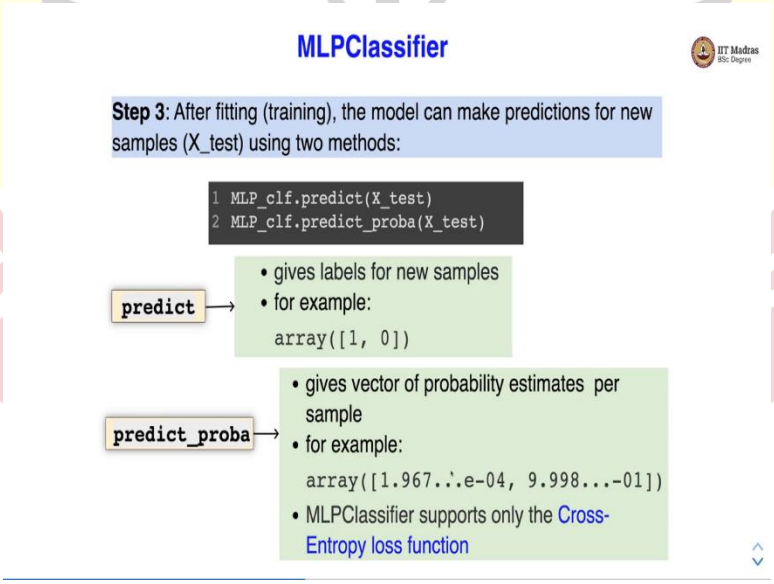
The training data is represented in form of a feature matrix X, which shape number of samples, number of features. And output is an array y or a vector y that holds the target variable and this vector has number of components = number of samples.

(Refer Slide Time: 01:56)



Let us see how to implement the MLPClassifier. We first instantiate MLPClassifier estimator object. So, MLPClassifier is implemented in sklearn.neural _network module. We first instantiate MLPClassifier with all default arguments, then we call the Fit method on MLPClassifier object with training feature matrix and label vector as arguments.

(Refer Slide Time: 02:24)

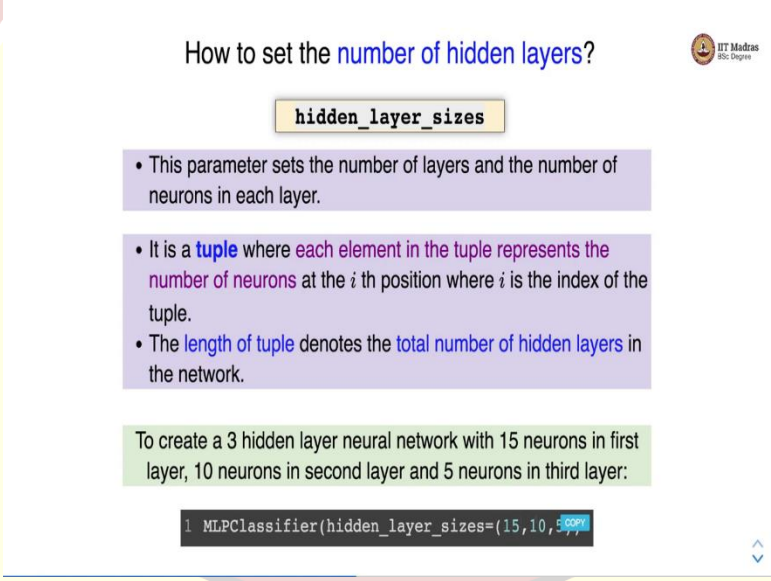


In the third step, after fitting the model, the model can make predictions for new samples using one of the 2 methods. We can use predict to obtain the class labels and predict _proba method to obtain the probability distribution over k different classes. So, predict as I said gives label for new

examples. So, the new examples are passed as input. This is the feature matrix corresponding to the test examples.

And we obtain the labels for each example in the feature matrix, whereas predict _proba gives a probability distribution over the class labels. So, here for a binary classification, we obtained 2 values to predict _proba this is the probability of sample belonging to class 0 and disability of sample belonging to class 1. And you can see that this particular sample will belong to class 1 if we perform, predict or arg max because this probability value is much higher than the other probability value. So, note that MLPClassifier supports only cross-entropy loss function.

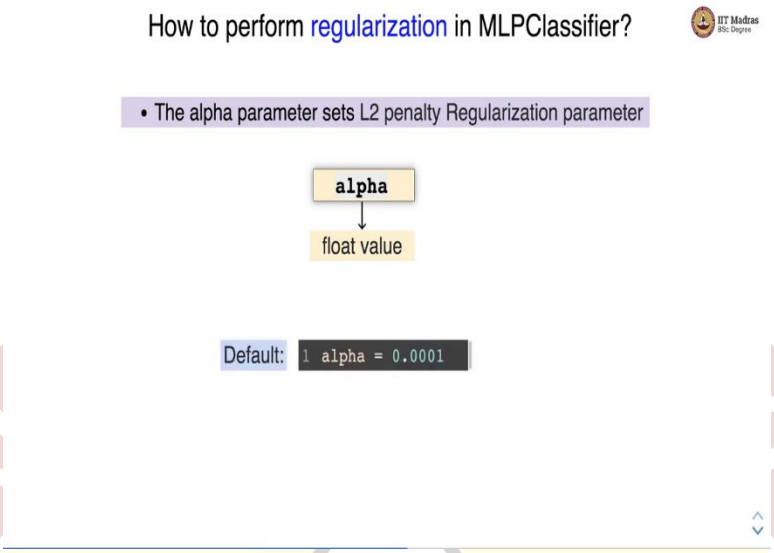
(Refer Slide Time: 03:41)



Let us see how to set the number of hidden layers. We can set it with parameter hidden _layer _sizes. This parameter sets the number of layers and the number of neurons in each layer. It is a tuple, where each element in the tuple represents the number of neurons at the ith position where i is the index of the tuple. The length of the tuple denote the number of hidden layers in the network.

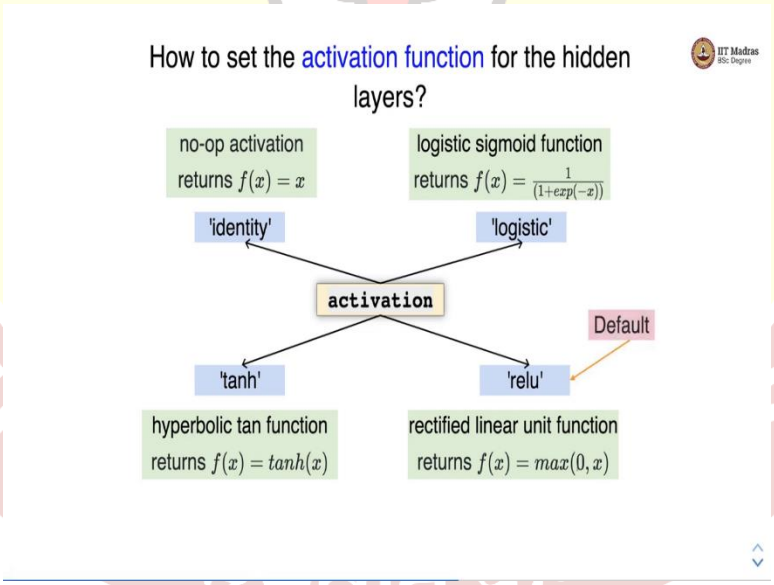
To create a 3 hidden layer neural network with 15 neurons in the first layer 10 neurons in the second layer and 5 neuron in the third layer, we essentially set the parameter hidden _layer _sizes with a tuple with 3 values, which is 15 , 10 , 5. So, the length of the tuple implicitly represent the number of hidden layers. And the first hidden layer has 15 neurons second one has 10 neurons, and the final one has 5 neurons.

(Refer Slide Time: 04:43)



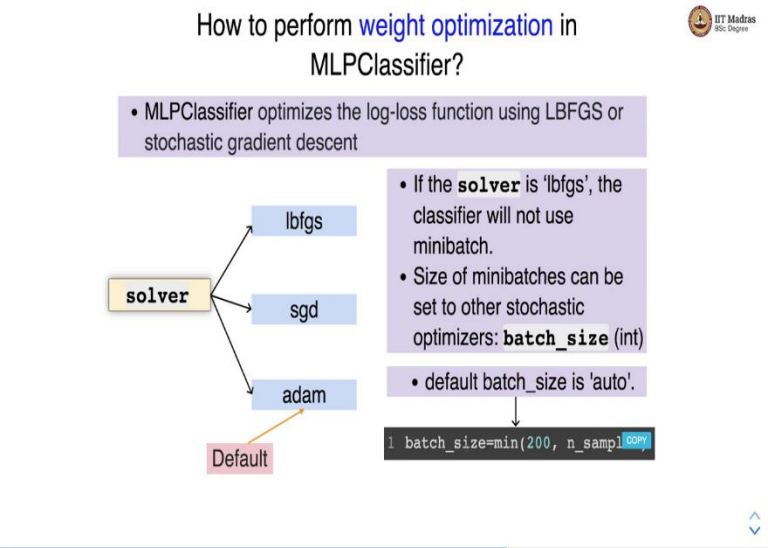
Let us see how to perform regularization MLPClassifier. The alpha parameter sets into penalty of regularization parameter. So, alpha is a parameter which takes float values, and by default it takes value 0.0001.

(Refer Slide Time: 05:02)



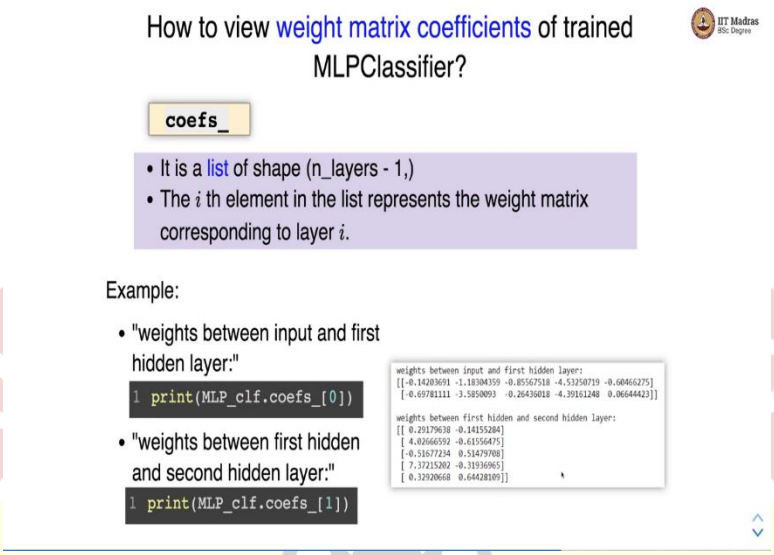
Let us see how to set the activation functions for the hidden layers. So, there are 4 activations 1 is ‘identity’ that returns the parameter itself. Then there is a ‘logistic’ which computes the logistic sigmoid function on the input value, then there is a ‘tannish’ function that calculates hyperbolic tanh function on the parameter and ‘relu’ that is rectified linear unit which finds the max between 0 and the input. Relu is a default activation function for the hidden layers.

(Refer Slide Time: 05:39)



Let us see how to perform weight optimization in MLPClassifier. MLPClassifiers optimizes the log-loss function using LBFGS or stochastic gradient descent. It can use 1 of the following solvers it could be a lbfgs, sgd or adam. Adam is used a solver by default if solver is lbfgs. The classifier will not use mini batch. The size of mini batch can be set to other stochastic optimizers by using a parameter batch _size, and it should take an integer value. The default batch size is auto and it is set to minimum of 200, number of samples. So, it is minimum of these 2 numbers that is set as a batch size.

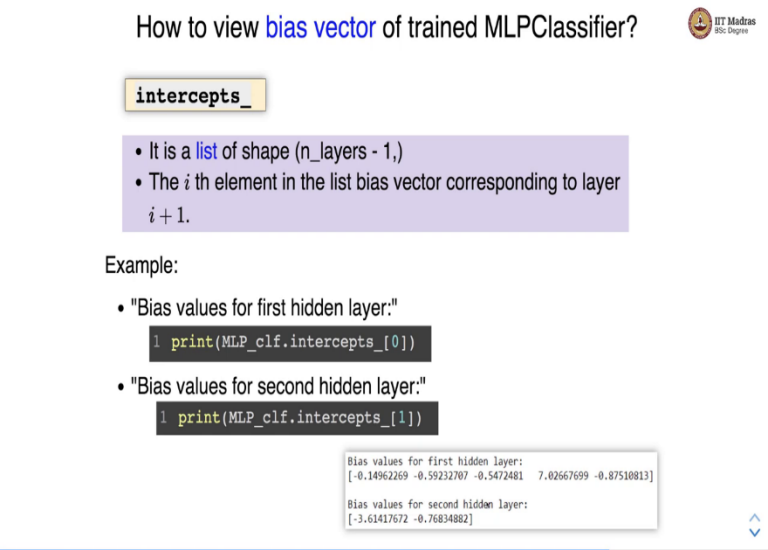
(Refer Slide Time: 06:29)



Let us see how to view the weight matrix coefficients of the trained MLPClassifier. We can obtain it using coefs_ just like other estimators. In this case it is list of shapes and layers -1. So, basically, we have vectors = the number of layers and ith element in the list represent a weight matrix corresponding to layer i. So, if you want to find out weights between input and the first hidden layer, we can obtain it by finding the coefs of 0.

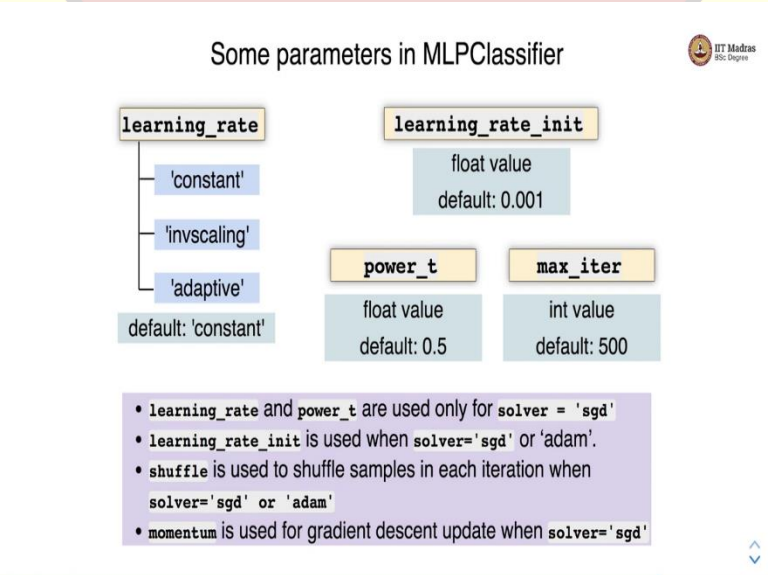
So, coefs of 0 will give weights between input and the first hidden layer. Whereas, coefs of 1 will give us weights between first hidden layer and then the second hidden layer. So, these are sample weights between first and the input into first hidden layer and first and the second hidden layer.

(Refer Slide Time: 07:34)



Let us see how to view bias vector of trained MLPClassifier it can be obtained using intercepts _ member variable. You can see the bias value for the first hidden layers using the 0th index of intercept_ member variable. Whereas bias value for the second hidden layer can be obtained by using index 1. So, these are sample bias values for the first hidden layer and the second hidden layer.

(Refer Slide Time: 08:07)

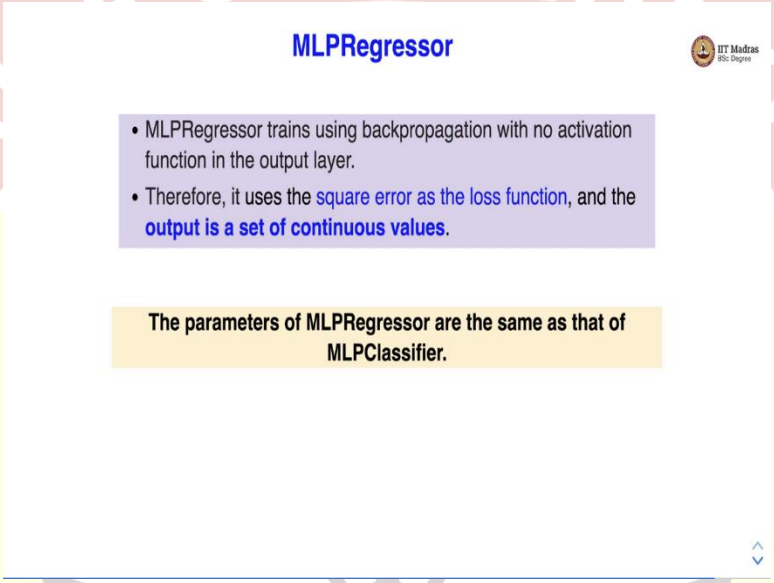


Let us look at some other parameters in MLPClassifier, there are parameters like learning _rate, which can be either constant, inverse scaling or adaptive. The default is the constant learning _rate.

The learning _rate is initialized to float value, which is a default value of 0.001. There are other parameters like power _t with default value of 0.5 and max _iter with default value of 500.

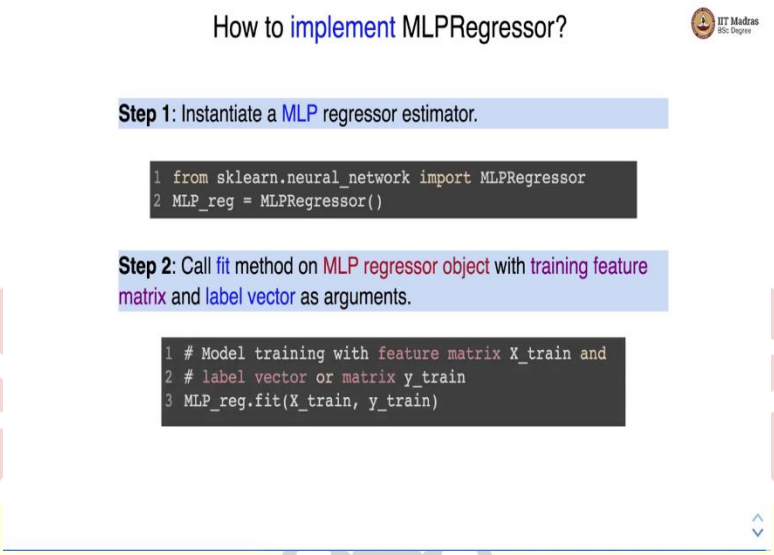
Learning _rate and power _t are used only for solver sgd learning rate in it is used for solver sgd or adam. And shuffle is used to shuffle samples in each iteration when solver is sgd or adam and momentum is used for gradient descent update when solver is = sgd.

(Refer Slide Time: 08:55)



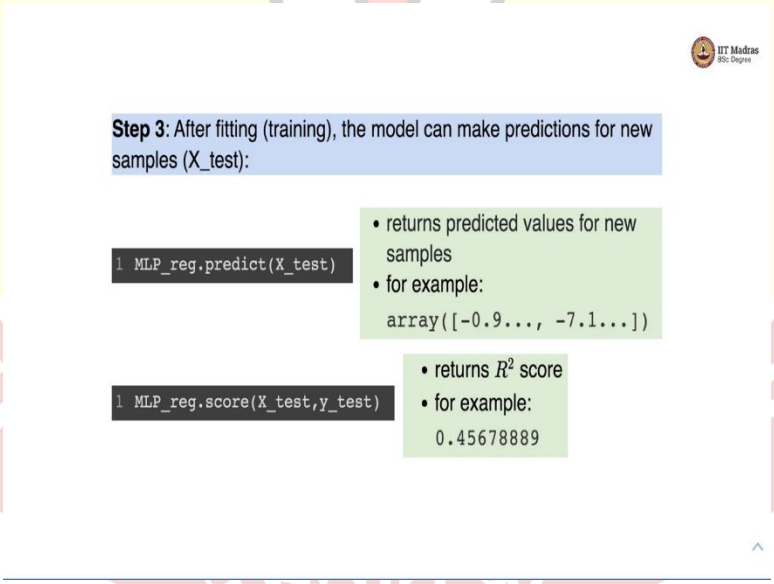
Let us look at MLPRegressor, which is used to train the regression models. MLPRegressor trains the regression model with backpropagation and in this case, there is no activation function in output layer. It uses square error as the loss function. The parameters of MLPRegressor are same that of MLPClassifier.

(Refer Slide Time: 09:16)



Let us see how to implement MLPRegressor. Instantiate MLPRegressor estimator. We call the Fit method with training feature matrix and label vectors as arguments.

(Refer Slide Time: 09:30)



After training the model, we can use it for making predictions for the new samples. So, we can use predict method with test feature matrix as an argument. It returns predicted values for new samples. We can also obtain the score by providing the test feature matrix and test output labels. It returns R^2 score for the regression. In this video, we looked at couple of MLP implementations which is

MLPClassifier and MLPRegressor that is used for implementing classification and regression problems in sklearn using artificial neural network or multilayer perceptron.