

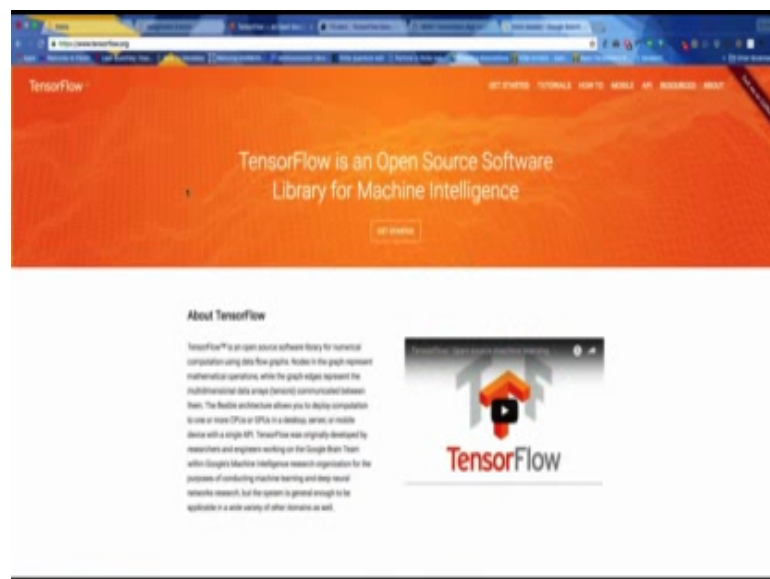
Introduction to Machine Learning
Prof. Mr Anirban Santara
Department of Computer Science and Engineering
Indian Institute of Technology Kharagpur

Lecture – 31
Python Exercise on Neural Network

Hello friends, welcome to the tutorial session of the 6th week of this course. I am Anirban. Today's topic is artificial neural network. As you people may already know that artificial neural networks form the foundation of a class of machine learning algorithms which are called deep learning, and currently these are doing wonders in the field of artificial intelligence.

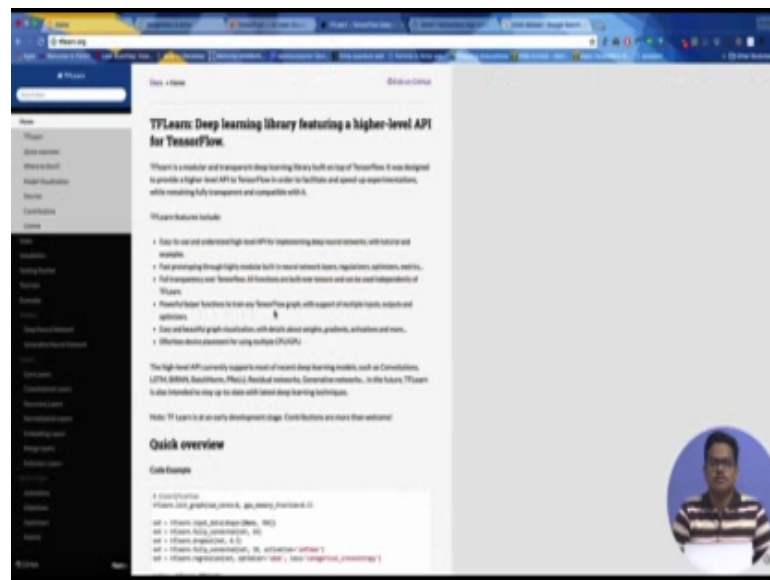
So, today I am going to teach you how to make simple neural network on a simple task, and show you how the performance changes when different changes are made to the architecture and the learning algorithm.

(Refer Slide Time: 01:00)



And the library that we are going to use for machine learning for deep learning is called tensor flow. So, tensor flow is a deep learning library which is open source and it was open source by Google and it is pretty state of the art and it is pretty popular as well both in the academy and the industry. And it has a large number of resources.

(Refer Slide Time: 01:20)



And a certain you know set of rappers have been developed around tensor flow, which come by the name of TF learn, so it gives the same s k learner scikit learn interface the API interface that you have been like that I have been using in my previous lectures. And that nice interface that model dot just you declare the model, you do a model dot fit it trains the architecture and then you do model dot predicted means a predictions and the same beautiful TF learn in this s k learn API is present in this TF learn library. So, TF learn is a set of rappers, so python rappers on top of the generic tensor flow library and we will be using that in this lecture in this session. It will be pretty interesting.

(Refer Slide Time: 02:12)



And the kind of work that the tasks that we people are going to we are going to solve in this session is the recognition of handwritten digits.

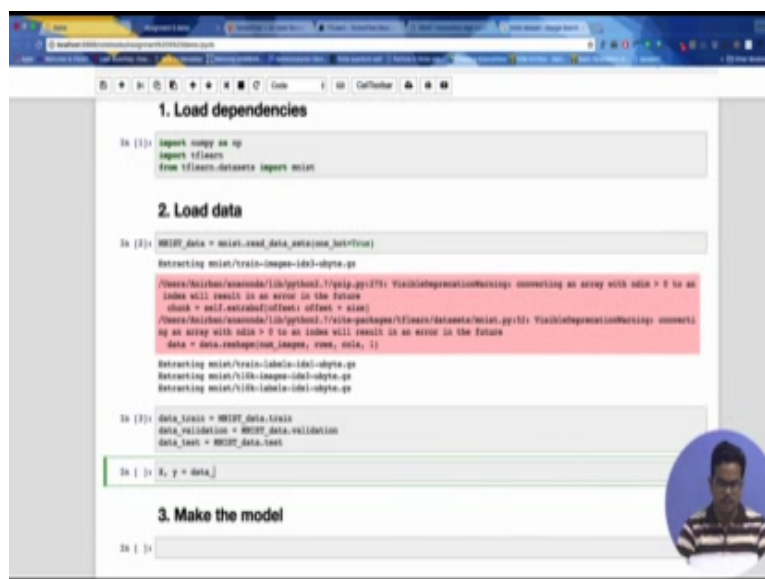
(Refer Slide Time: 02:24)



So, the MNIST database is it is stands for so it is the mixed MNIST database. It is a database of handwritten digits, and these digits where actually they were extracted from post postal codes, the pin codes, the zip codes that people write on letters, so they were scanned and the individual handwritten these letters these digits were extracted. And the task is to automatically identify which handwritten digit was written in that particular you know that particular instance.

So, here are some examples from the data set, and so there are some pretty nasty looking characters right like this 8, which looks really bad; this may be a 7 or something, so it is not a trivial task of identifying the handwritten digits. And this MNIST database is one of the first choices of data set which people would like to use just to check out the performance of a new learning algorithm that they have come to their mind. So, this is pretty basic stuff and I think that this particular exercise will give you a good head start in these deep learning algorithms.

(Refer Slide Time: 03:39)



```
In [1]: import numpy as np
import tflearn
from tflearn.datasets import mnist

2. Load data

In [2]: MNIST_data = mnist.read_data_sets('./mnist', one_hot=True)
Retrieving mnist/train-images-idx3-ubyte.gz
/mnists/mnist/train-images-idx3-ubyte.gz: ViolatesDeprecationWarnings converting an array with index > 0 to an
index will result in an error in the future
check = mnist.read_data_sets('./mnist', one_hot=True)
/mnists/mnist/train-images-idx3-ubyte.gz: ViolatesDeprecationWarnings converting an array with index > 0 to an
index will result in an error in the future
data = data.reshape(num_images, rows, cols, 1)
Retrieving mnist/train-labels-idx1-ubyte.gz
Retrieving mnist/train-images-idx1-ubyte.gz
Retrieving mnist/train-labels-idx1-ubyte.gz

In [3]: data_train = MNIST_data.train
data_validation = MNIST_data.validation
data_test = MNIST_data.test

In [4]: X, y = data_1

3. Make the model

In [5]:
```

So, without further I do let us go ahead and start writing our code. So, the first step is to load the dependencies. And yes, I am here we writing the code from scratch and in ipython notebook or jupyter notebook, and I will share this notebook on githoc and link in the description, so the link in the description so that you can also use and referred to it later on right. So, let me first load the dependencies so I have to import (Refer Time: 04:03) as np import TF learn. And from TF learn dot datasets import MNIST. So, we just go ahead and execute this. So, once this has been done, this may be I can zoom in a little bit, so that you can see. So, these three things are the necessary dependencies the execution is complete.

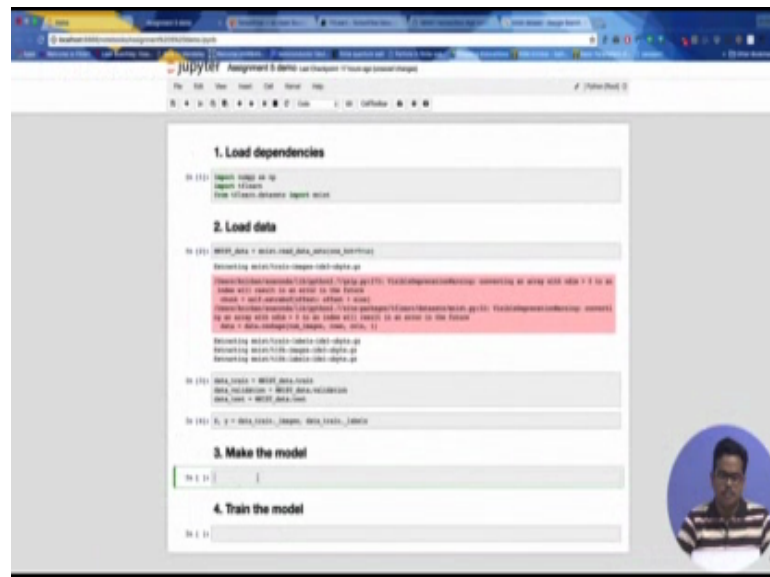
Let us go ahead and load the data. So, let us declare this as MNIST data equal to MNIST dot read datasets. So, the read datasets is of function, so I just make sure yes. So, the read datasets is a function which is there in within the MNIST this (Refer Time: 05:09) that we are invoking, so you can actually look up the structure of TF learn dot datasets. And it has a number of really good utilities, and this read datasets function will read the data set; so if the data set is not there in your computer, you just going to download it first. And then it is going to I do some preprocessing on top of that bring everything into the format that is easy for machine learning and then written the data set in a nice format. So, we will see the format of the data in a minute.

So, I will just write one hot equal to true. So what does these do, let us execute and then speak. So, it is already done. So, as you can see that the down datasets were already downloaded in my system. So, they just you know read the files and you have the data here right. So, I will just add a new cell and start speaking, add cell block. So one hot target are targets represented as a vector in which just one of the terms is a 1 and rest all of them are 0. So, it is like if you have 10 different digits to identify then the each digit each handwritten digit may be represented by a 10 long vector, vector of 10 elements.

And say it is an image of a 4, 4, so the one hot vector corresponding to this 4 will be 0 0 0 1, and the rest zeros. So, all zeros except the fourth position which is a one so this one hot representation of vectors like of targets is useful for using along with different kinds of loss functions like cross and (Refer Time: 07:02) loss or even like it is like describing it is a good way of describing it is a very usual popular way of describing categorical targets so in the form of a binary vector. It is called one hot vector because of just one of those elements in that vector is one, the element corresponding to the entry of that particular class.

So, I just said one hot equal to true over here, now let us have a look at the data. So it is going to make the targets in a give written the targets in a one hot format. So, let us first divide the data set into training validation and test bit. So, maybe I can say data underscore train equal to MNIST data dot train so it is organize this way data underscore validation equal to MNIST data dot validation and data underscore test equal to MNIST underscore data dot test. So, we just did the training validation and test split is and added on the cell and let us have a look at the training data. So, let us say that we load them into the variables x and y, so this is going to be equal to data train, so the training data and underscore images gives is the field within the class this like there is a class call data set.

(Refer Slide Time: 08:48)



And you can look up the structure; I will give the links in the description of the video. So, this underscore images this particular attribute gives the inputs and data underscore train dot underscore, so it is target labels. So this will, so now you have the inputs and the target values in x and y respectively. Let us go ahead and see what the shapes of these entries are like this we can do x dot shape and see, so there are that entire data set has 60000 thousand images, out of them 55,000 are going to be used for training, and the rest will be so have been distributed among the validation and test sets.

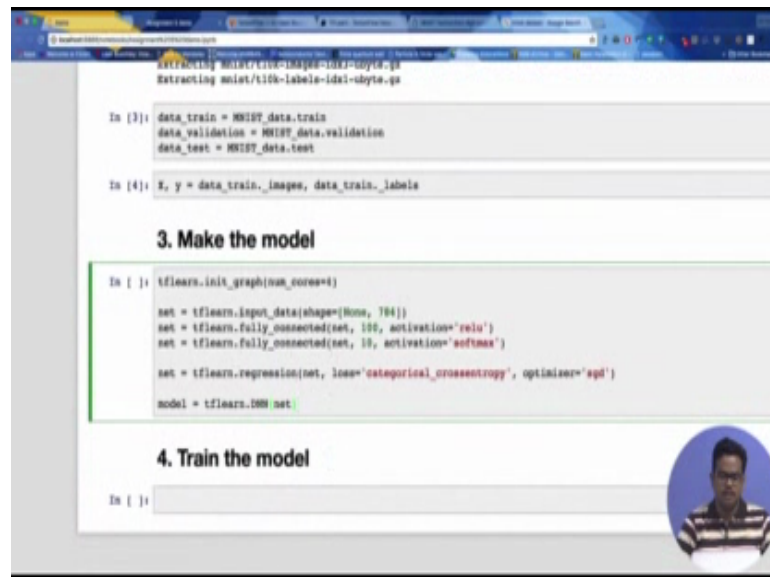
So, x dot shape and each of these numbers that is you saw over here, so each of these images is a 28 cross 28 image black and white image. So, 28 times 28 make the number 784, so that is why you can see that the length of each training input vector is 784, and there are 55,000 of them. So, they have been organized in the form of this metrics.

Let us see how y looks like, y there are also like for every single of the 55,000 training images, we have their corresponding label encoded as a one hot vector and let us have a look at what this one hot vector looks like let us see what y zero looks like see. So, it is a ten long vector there are 10 entries and the corresponding like this, this happen to be the image of a eight right sorry seven, so it is like 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. So, this was the image of a seven that example of and hence that is the position for seven is one the rest are zero. So, this was just to show you how things look like.

So, after this has been done, let us go ahead let me delete this cell, it is useless. So, just go ahead and delete the cell. So, once we have the training data ready, let us go ahead

and make the model. So, the tensor flow it organizes the entire architecture of the neural network in the form of a graph, so it is called the tensor flow graph. So, the first thing that we have to do while making your neural network is to first define the graph structure so initialize a graph by invoking this function.

(Refer Slide Time: 12:14)



```
Extracting mnist/t10k-images-idx1-ubyte.gz
Extracting mnist/t10k-labels-idx1-ubyte.gz

In [3]: data_train = MNIST_data.train
        data_validation = MNIST_data.validation
        data_test = MNIST_data.test

In [4]: X, y = data_train.images, data_train.labels

3. Make the model

In [ ]: tflearn.init_graph(nom_cores=4)

        net = tflearn.input_data(shape=(None, 784))
        net = tflearn.fully_connected(net, 100, activation='relu')
        net = tflearn.fully_connected(net, 10, activation='softmax')
        net = tflearn.regression(net, loss='categorical_crossentropy', optimizer='sgd')
        model = tflearn.nn.net

4. Train the model

In [ ]:
```

So you just call TF learn, and just a second, now, yes. So, TF learn dot in it underscore graph and you specify what all system resources that will be necessary for this particular exercise. And just as a matter of fact that deep learning requires a special kind of you know computing infrastructure which is known as graphics processing unit. So, the same graphic cards that again we that are used for playing games that you can find that is used for like heavy duty, visual renderings, the same graphics cards, so they are high the capable of doing metrics multiplications very efficiently and that is what a comes to a great help in doing this like deep learning applications.

So, tensor flow is it is compatible to it is completely optimized for GPU for this graphics processing unit is so how much memory from a GPU, you would like to use in a particular experiment you can specify over here. But my PC does not have a GPU now, so we will just say specify that we are going to use just all of the four cores of my PC. So, I just say that this is the compute requirement of the algorithm number of cores equal to 4.

Next, we go ahead and start making our network. We just we store our network within the variable called net, so first we add an input data layer. So, it is just called TF learn dot input underscore data and you say what the shape of the input data should be. So, you just so the shape is in this case is equal to a list first element is none I will tell you what it means followed by 784. So, have a look at this, so the first element in the shape this corresponds to the number of yeah this element none, the first entry in this shaped vector it corresponds to the number of samples that needs to be presented in a particular batch. So, in the tutorial session I have already talked about what is batch learning, what is mini batch learning, what is stochastic learning three different kinds of gradient design that you can use.

So, the first term gives the number of elements that you want to take in each batch and that is variable in this case, so that is why you do not specify any entry over here, you keep it none, so that the graph can be modified accordingly. So and 784 is the dimensionality of the image that you are going to feed. So, in this case we have like factorize the 28 28 image into a single vector of length 784, so that is why you have this entry, so this is your input size.

Next, we are going to add the next layer. And so this is the first hidden layer TF learn dot fully connected and check this index, perfect. So, in the input will be the net, so which layers so when you start you know without the manner in which you build a network in TF learn is by like if you add the layers one by one. So, there is one network object, it is called net, you first put a input layer in then you add the next hidden layer, then you add the next hidden layer then another as deep as you want to go. And then finally, you have an output layer, and then you go ahead and declare your loss functions and the optimization algorithm and set it to train.

So, we pass this net object as an input to the function, so it is going to add a particular layer and the shape of the layer. The number of unit is in the layer is going to be say one hundred we keep it 100 and the activation function of the nodes of the layer is going to be sigmoid, very sorry let us give it relu first rectified linear unit. So, it has already been covered in class that what rectified linear unit are so it is one of the best suited for deep learning for deep neural networks. And it works pretty well and there is a lot of theory about why rectified linear unit is so awesome.

So, let us go ahead and add the output layer. So, a single hidden layer neural network and we will do will (Refer Time: 17:26) play using it, play with it, let us see. So, as the number of units in the output layer is 10, and it is just because you have 10 outputs unit is right represented as a like 10 output classes and the output labels are coming in the form of one hot vector. So, the activation here is going to be softmax, so perfect. So, the softmax layer, it is going to it is a kind of logistic regression so multi multiclass logistic regression is called softmax. So, you can actually look up the web and figure out what it looks like perfect. Next, this or this makes our network the networks build is complete.

Now we have to define what kind of loss function we want to use to optimize the neural network and parameters of the neural network, and what kind of algorithm update algorithm that we should use, let we want to use. So, we add another layer this is not a an actual like you know hardware layer to the neural network rather it is a specification of the a learning algorithm options. So, it is called regression. So, the regression layer actually does either a linear regression or a logistic regression.

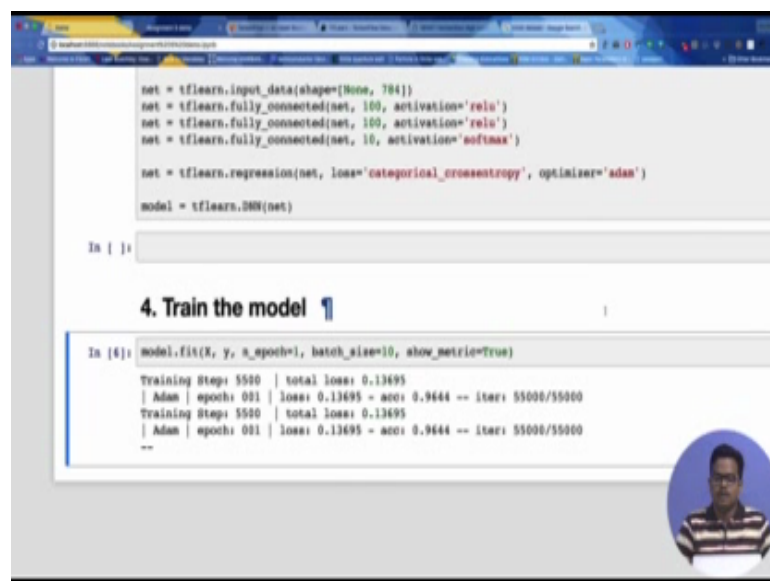
So, the first input as always is net, and then you have to specify the loss function. So, the loss is equal to categorical cross entropy, so you can look up the web what categorical cross entropy looks like what it actually is. But I will just few words that categorical cross entropy is a loss function which tries to match the probability distribution of the actual observed data samples to the probability distribution that is getting modeled by the neural network. So, when the cross entropy is actually cross entropy loss the categorical cross entropy loss is actually an adaptation of the k l divergence distance between the statistical distance term, statistical distance metric and it directly tries to match the actual the data distribution and the model distributions in course of the training. So, you can look up and read more about this, but this is outside the preview of this course.

So, we will put an optimizer, and let it be stochastic gradient descent. Now we have defined what kind of learning algorithm, and what kind of loss function will be used to train the neural network. Now, we define, we write this so we define our model. So, again now the scikit learn API comes in, so see how beautiful it is, so we are just going to put `TF learn dot DNN`. So, yes, so we are now that you have already defined what kind of a neural network you want, and how it should be trained and everything.

Now you initialize this TF learn dot DNN this is like sk learn dot linear regression or something like that, so a model learning model learning algorithm or a machine learning model, just define this as a machine learning model. And then you train this model. So, I will just leave this spot in this particular cell, and we can make new cell for define for training the model, so it is all compiled.

So, what did we do here we first declare what kind of see compute resources we are going to use, then we initialized the structure of the neural network. And said what the different layer sizes should be, what different kinds of activation should be, and then we said what kind of loss function and what kind of update rule should be used to optimize the learning algorithm. To learn the weights rather and then we declare the model. And then it is time to train, oops so I already made another slide another for this. So we are now going to do model dot fit yeah as we always do.

(Refer Slide Time: 22:16)



```
net = tflearn.input_data(shape=(None, 784))
net = tflearn.fully_connected(net, 100, activation='relu')
net = tflearn.fully_connected(net, 100, activation='relu')
net = tflearn.fully_connected(net, 10, activation='softmax')

net = tflearn.regression(net, loss='categorical_crossentropy', optimizer='adam')

model = tflearn.DNN(net)

In [ ]:
```

4. Train the model

```
In [6]: model.fit(X, y, n_epoch=1, batch_size=10, show_metric=True)

Training Step: 5500 | total loss: 0.13695
| Adam | epoch: 001 | loss: 0.13695 - acc: 0.9644 -- iter: 55000/55000
Training Step: 5500 | total loss: 0.13695
| Adam | epoch: 001 | loss: 0.13695 - acc: 0.9644 -- iter: 55000/55000
---
```

So model dot fit, now the model is already declared we train it on x and y. So, these are your input and targets of the training data. And now you specify a couple of more options. You specify that the number of epochs of training in epoch is going to be one. So, what is epoch so it is a very common concept in machine learning, so when you are actually trying to find out the like you want to do gradient decent, you have a bunch of parameters which you want to find optimum values of.

And what you do is you show the learning algorithm the same training examples time and over and over again. So, say you have one million training examples; and in the first epoch, you are going to show the neural network say it is a neural network the learning model. So, you show all the in the first epoch you show all the training examples to the neural network. So, it does some updates everything.

Now in the second epoch you randomize you randomly permute all the, you shuffle all the training examples you have, and you push it again. And it has been shown that this helps to break the sequence the like when you are presenting the training examples in a particular sequence to your machine learning model to a neural network, then the neural network might actually memorize a sequence.

And so the sequence in which different training examples appear to the neural network, may actually you know actually we reflected in the optimization in the values of which the weights can take up. Just to break that sequence what we do is we randomize the samples and pass the same training said over and over again. And this like carries on the training, and we need to like the more you train, the better the models fit is on to the training data. So, it is a concept and I just keep I am going to keep n epoch equal to 1, number of epochs equal to 1 to like show you the other thing is ok. And it is just going to speed up the learning process a little bit take a little bit lesser time.

Now you show now it is batch size right, so sorry oops batch size. So, when you as you are doing stochastic gradient decent, the batch size is going to matter. So, it is the number of examples that you want to show per epoch per you know per instance of learning or like after how seeing how many examples you want to do one update. So let us keep batch size is equal to 10, and another option is show metric. So in the progress of training as training proceeds it is going to show us the value of accuracy. Let us just go ahead and see what happens.

Notice here, so as you can see that that the accuracy is increasing; the training accuracy is increasing the loss is decreasing. And you know it is going to make you divided the entire training set into batches of size 10 and it is like pushing every single batch and then updating. So, one update is happening after every batch has been processed and you can see that the number of examples that have been shown to the neural network is been counted over here.

So, when all the 55,000 examples have been shown, the training is complete, so one epoch is complete. And it reports the final accuracy over here. So, let us go ahead and see how things change when we increase the number of or increase or decrease the number of nodes of the hidden layer. So, let us go ahead and make the nodes, so let us make a node that previously therefore, hundred nodes in for neural network with one hidden layer and 100 nodes accuracy was just 69.5 percent. So, we are now decreasing the number of nodes and we will see how things change. So, I will restart and run all, just see here what happens, it will restart, so the performance yeah.

So, now, you can see that the accuracy is increasing slowly because the models learning capacity has been reduced. The number of parameters is reduced, so the capacity to learn has also reduced. And however, the training is progressing much faster, so you can just see that yeah like you know in flash 30,000 examples are processed, because as the amount of compute whenever a parameters has reduced, now computation has been faster. What happened my operation and close the files oops restart yeah (Refer Time: 27:47), just a second run all, do not take much time. So, we can expect that the accuracy will be lesser this time.

Let us see, so you see the accuracy would not increase beyond some 26 percent, and this is just a training accuracy. As the models learning capacity has reduced, so this has gone to like 35 percent or something; see 44 percent previously it was 69 percent. So, as the learning capacity was reduced the neural network could not learn well, and hence the accuracy is lesser. So, let us go ahead and rest over the number of hidden unit is to 100 previous values and let us add another hidden layer. See, what happens oops so let me shrink it a little bit and go ahead and add another hidden layer. And see how things change. So, previously it was 69 percent. Let us go ahead and restart and clear output yes.

now run all set, let us see how the accuracy changes this time. So, now, the training is a bit slower, because the size of the network has increased. And but we can like hope that this times the accuracy will a bit higher, but it is not certain at all it is should not necessarily be higher, and that is where the entire trick comes in. So, it was like really you know unfortunate that on increasing the number of hidden layers, the accuracy actually fail, so this is the actually the trick of deep learning and it was it was seen that deep neural network with any hidden layers actually fail to arrive at a good solution.

And you can see that the accuracy is a 12 percent, so it just stuck in a bad local minimum and it had to come out. And so let us go ahead and make a small change to the optimizer over here. So, there is an optimizer which is called Adam and it is I will put the link in the description of the video that what it actually does, but it is an you know a sophisticated optimization algorithm and we can hope that this does a better job and gives us a better accuracy.

Let us say so see this, see this, it is started with an accuracy of 89 percent look at this. So, what the Adam optimizer does is it uses a different learning rate for every single parameter. So, it is like every single parameter are should be may be needed to be treated differently in course of the algorithm the optimization algorithm. And see you can you can see just the accuracy see it has it has touch 95 percent 97 percent, so it has a different learning rate for a different optimizer, different parameter and also the learning rate changes with time. And it is called annealing and it has a separate annealing scheme for every single parameters.

So, I will put a link in the description to this paper and it is actually you see there, accuracy is 96.44 percent. So, these are the different things that so what is you are take away from this exercise, the take away is that when we have a single hidden layer neural network, the more is a number of unit is in the hidden layers the more complicated is the models function - the hypothesis function and hence the more is it capacity to learn. So, given enough data, it can learn better so that is why 100 unit hidden layers gave an accuracy of around 69 percent. Whereas, if the when the number of unit was reduced to say 10 the accuracy drop to 29 or something it was in the twenties.

But and the accuracy now the models complication, the complexity of the model could be increasing in two ways. Either you increase the number of nodes in the hidden layer in one hidden layer so you have a just one hidden layer and you put many nodes in that or you could stack many hidden layers one after the other. And it has been shown so what we saw here that when we increase the number of hidden layers, then simple stochastic gradient decent fail to convert show a good local minimum.

So, but the accuracy was just the training accuracy was just twelfth percent, so it is close to the it is good to call that it did not learn at all it just got lost somewhere. Whereas, so when you like have so many hidden layers right two hidden layers of 100 unit is each

you have a lot of parameters and that is why you need to choose a very good optimization algorithm. An optimization algorithm which gives special care to every single parameter and make sure that these parameters are tuned properly in course of training, and finally, we can like arrive at a good optimum which may not be the global optimum, which it may never be reached at all.

So, it may not be a global optimum, but it should be a good local optimum, so that is why when we used Adam optimizer, and this just I told you in few words what it does so I will link into the paper, so that you can read it and understand get the full detail about it. So when we used Adam the performance improved a lot and we got you know 96 percent accuracy. And the current state of the accuracy on MNIST data set is something like 0.227 percent error, and it is beyond human abilities.

So, the machines today can recognize handling characters be better than human beings. And so you can like go to the MNIST, Wikipedia page and there it gives list of algorithms which are doing very good which are the state of the art in this task, but they use much advance concept like convolutional neural networks and others, which you can explore yourself. So, both the motivation behind me introducing you to TF learn and the larger and the like the parent of t f learn which is tensor flow is that just to give you a heads off in that directions.

So, if you people are enthusiastic about deep learning if you really want to check out what the magic is all about, how are machine becoming really intelligent like as intelligent as human beings, and beating human beings in they are own tasks. So, if you want to check out how deep learning actually works, so this is a very good tool which you can use for implementing your own algorithms and like there are a lot of like turn of implementer algorithms already implementer algorithms.

A lot of resources available for this particular library, and people are actively and Google is actually actively developing this particular library. So, you can use this and you know explore the possibilities of deep learning in your own field. So, I hope you enjoy this video, and I recommend you to take this ahead.

Bye, bye, see you in the next video.