Fast.ai lesson 4

29 June 2021 22:03

Mnist loss function :

Torch.cat : concatenates the given amount of tensors in given dimensions, all tensors must have same shape or they must have to be empty

Torch.view : create the views and does the task of reshaping them without repeating the whole tensor or storing the whole tensor

Torch.cat().view example
Torch.cat([stacked_threes, stacked_sevens]).view(-1,28*28)
Here the view is to reshape the tensors to the list of the tensors for doing this thing we use view
Here the –1 is for the maximum no of rows which are in the data
28*28 are number of columns available into the data

Generating the labels for our data
For genreating the labels we are using simple python
As
Torch.tensor([1]*len(threes) + [0] * len(sevens)).unsqueeze(1)
So here the 1 is getting repeated for all the 3 and 0 is for all the sevens and unsqueeze is for adding the one more dimension to the tensor so that all the inputs will be in different arrays

Converting the training data and labels into the dataset
Dset = list(zip(train_x,train_y))
X, y = dset[0]
x.shape,y

Here the zip function takes the 1 element from the 1st para and 1st from second parameter and concatenates them in each tuple
We can unpack with indexing on the dataset and unpacks the number of the parameters which we are passing

We do the same thing for training and the validation set

Initializing the parameters using the random parameters using torch.randn()
We want gradients for this parameters so we do state the requires_grad_() here we are telling pytorch that we are required to change the gradients of this function

Weights * pixels won't be flexible enough when the pixel are equal to 0 it will give out the zero output for solving this problems we are getting the random bias one value for bias which will be added to the weights

Y = wx + b
W = weights , b = biases x is the function parameter

Doing matrix multiplication in python

In python the matrix multiplication is represented by the @ sign so for doing the matrix multiplication of the weights with the x pixel value we can use the python operator it runs in the c code so it's optimized

For finding that our model did the right predictions or it did the wrong predictions we use the simple python as
Corrects = (preds>0.0).float() == trian_y
Which will return true if both the values are same otherwise it will return false
So that for getting the accuracy of this
we do
Corrrects.float().mean() # which will return the tensor
If we applied the item() then it will return the items inside the tensors

When we are doing the change in the small scale in weights the accuracy can't be act as the loss function and we will not see even a slight change in accuracy so we have to define the loss function for measuring the loss of our model

Torch.where is similar to the list comprehensions

Torch.where(a==b , c, d)
If a==b is true then do c otherwise do D

```
# here in the above loss functions when the predictions are right with the high
prediction proba then the loss
# is less if the predictions is wrong with the higher probability the loss will
be higher
```

SGD and the mini batches :

When we have to calculate the loss and have to update on every loss parameter then we have to do complicated task and as we do it it can get much longer to train specially when we have big dataset so to avoid increase in the training time we do SGD with mini batches so we select like 1 image and then update our weights according to the losses there are

In pytorch there alrady is the class for converting the data into the batches so for converting the data into the batches all we have to do is to pass the data to our batches

Coll = range(15)
Dl = DataLoader(coll, batch_size = no of theh batchsize , shuffle = true/false as per need)
List (dl)

Defining our own optimizer

While definign our own optimizer we should remember that each optimizer is haivng the learning rate parameter built in step function with the zero_grad module which is responsible for updation of params and basic steps of optimizer

For initilizing the parameters of the optimizers we've to pass the parameters of the model and the learning rate by which we want or model to learn

Before defining the model we need to have the trianing data in terms of weights and biases we can do the data into the weights and biases format as
Linear_model = nn.Linear(28*28, 1)

Here the linear model from pytorch is initialized with the input size which we give nad if we open it using linear_model.parameters() it will give out the weights and biases as parameters
w.shape , b.shape

Train epoch is the function which takes the all the parameters and puts that in model calculat es the steaep and the zero bgradient step

Optimizer which are already defined in the pytorch can be imported with similar way
Just like we did imported the model
As linear_model = nn.Linear(28*28, 1)
Opt = SGD(lienar_model.parameters(), lr)
Train_model(linear_model, 20)

Dls fast ai already has the DataLoaders(dl , valid_dl) class which is really handy class it differentiates in between the train and test data and we can access the train and the test data using dls.train , dls.test

Learner class:
all the stuff which we've done manually this class is gonna do it for su
learn .fit (10 , lr = lr)
Will be fitting the same thign and giving the same results

Adding the non linearity :
for getting the non linearity we combine the 2 linear neural networks
while combining we first define the first linear neural network and after that we do replace the negative digits wiht 0
after that we define the 2nd neural network and pass the replaced neural network digits to it

Res.max(0) is the rectified linear unit when negative it gives 0 and if positive it gives the positive value

For defining the same code in fast ai we did all above in really simple manner is
Simple_net = nn.Sequential(nn.linear(28*28,30),
Nn.ReLU(),
Nn.Linear(30,1)
)

We can pass the model as the parameter to Learner() which in turn creates the model and then we can do same training as we did above in the fast ai type of models

Jargon : now we know how to build the neural networks from scratch

We also know that the parameters are learnt but the activations are calcualted (sigmoid , softmax etc func)

Rank 0 : scalor tensor
Rank 1 : vector
Rank 2 ; matrix ;