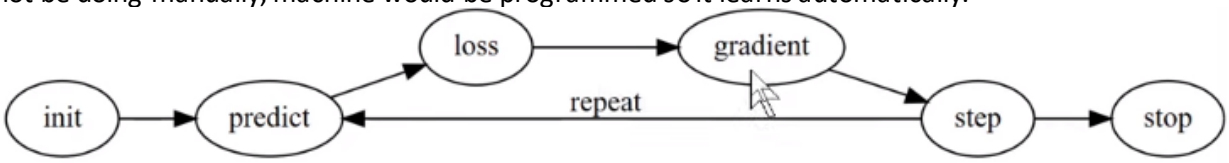
Lesson 4

Saturday, September 26, 2020
6:03 PM

What happens when we trainig neural networks?

1. What happens in SGD?
    a. We automatically assign few weight parameters and calculate the performance, we would be changing/updating parameters accordingly in order to maximize the performance. Need not be doing manually, machine would be programmed so it learns automatically.



    2. Initialize weights randomly.
    3. Predict outcome,
    4. How good is prediction, Calculate loss,
    5. Check how loss changes, by changing one weight parameter. gridents
    6. Using the step, to check loss,
    7. Repeat process, until loss doesn't change much.
2. Applying SGD to MNIST Model
    a. Prepare training and validation datasets, in order to align them in set of parameters.
    b. Initialize the weight parameters, torch.randn initialize the tensor with set of weight parameters.
    c. Requires_grad_ is a option which we apply to a function, which automatically calculates gradients for you.
    d. Weights and bias are called parameters.
    e. Initialize some bias.
    f. Difference between GD and SGD?
    g. y=wx+b, where x is our input, as we initialized weight vector with dimensions of input, we do matrix multiplication between weights parameter and input data. In pytorch/fastapi, @ does this matrix multiplication.
    h. Batch * weights + Bias
    i. He then started doing random model, just to check what is the accuracy, with that random model he got 0.49 as accuracy.
    j. He then changed the one of the parameters, precisely, he then muliplied all the weight parameters by 1.0001 just to check how much accuracy changes? He then received same accuracy.
    k. Gradient is rise over run.
    l. He mentioned gradient is rise over run, i.e. change of y w.r.t x. i.e. here our y values didn't change, because we got same accuracy. i.e. derivate is 0. i.e we received 0 gradient in all over the place. If we get 0 gradient we can't alter weight parameters. So, we use something other than accuracy as our loss function. Lets create another function other than accuracy.
    m. Loss will be smaller, if our predictions are exactly right. If predictions are identical to targets, then our loss function returned value is smaller, if they are exactly opposite then our loss function returned value is maximum.

n. The function that he defined i.e. mnist_loss works very well, when we have loss function between 0 and 1, otherwise it behaves funny.
o. So, we need some function, that can take big numbers, and returns anywhere in between 0 and 1. i.e. sigmoid. If we pass number small number, we get close 0, big number returns 1. Squashes every number between 0 and 1.
p. 1/(1+e-x). metric, is thing what we actually care about, loss is something we similar to metric.
q. So, we update mnist_loss function using sigmoid function, so, all our predictions between 0 and 1.
r. So, we do this for every step, i.e. calculate prediction for every image, and calculate loss, and then step the parameters, and then predict for the second image. And repeat process. But this would take huge time. By passing, whole data is called epoch. We are taking single step for single image, which could take longer time.
s. We sometimes, have tens of millions of data, huge data we get at sometimes. So, it takes hell out of time to complete. So, what if we do on small data or on mini batch of data.

3. SGD for mini-batch
    a. Updating weights for each step will be taking longer time, as it has to predict and calculate loss for each image, it is actually very slow. We call it as epoch. At times, we could have millions or tens of millions of data points, which would definetly take longer time, so, lets pick few items from our input dataset, to calculate loss or step. That is called **minibatch**, mini batch is nothing but few items from your data. The bigger the batch value, takes longer time to calculate loss value, smaller the batch size it runs faster and calculate loss value, but it wouldn't such accurate when we consider bigger data set.
    b. Somebody asked, why do we consider mean of the calculations, why don't we consider median which is less prone to outliers? He then answered like Problem with median is it doesn't care about other numbers, it only cares about only one thing which is in the middle. Contribution of other items are nothing, if we consider median.

    c. How do we pass few items to dataset?

        d. We have a function called DataLoader, which does this random sampling thing, upon mentioning reasonable batch_size. If you code shuffle=True, it makes more randomized and passes not redundant data.

        e. First is fast.ai function, it returns arbitrary element from your dataset. Returns 2 items x and y. predictor and target variables.

        f. Gradient is nothing but, change in loss wrt to change in parameter. Gradient function not only calculate gradient of the value, but also adds to existing variables. So, this could be one of the reasons why do we see different values for each gradient whenever we execute the gradient function.

        g. How do you do perform SGD using minibatch? You have your data in your dataloader, we declare the batch_size, and return few items, by keeping shuffle=true, it makes random data to your model. So, when you have dataset, you can pass it to dataloader, and ask for data.

h. When you execute the three functions, i.e preds, loss, loss.backward(), you could see it isn't same everytime. That's because loss.backward() calculate the gradient and update the gradient. So, you don't see them the same everytime you execute it. So, we need to update weights and grad to zero.

i. Gradient descent run on whole dataset, and update datasets, but in sgd it takes in minibatches.

j. He then calculated accuracy by including the sigmoid of the function, now it returns accuracy of 52%. He then ran the model for another epoch on the datasets, now the validation accuracy has increased to 68%. And then iterated for 20 epoch, he was able to get 97% accurate on validation data.

k. As these are many function, he is trying to refactorize all these with functions, basically creating an optimizer by taking rid of linear1 function.

4. Creating an optimizer
    a. First thing is to get rid of linear function. xb@weights, instead of that we use nn.linear function.
    a. Nn.linear is a function which creates a random weight parameters, matrix of input size (in our case 28*28) and bias of size =1 put requires grad.
    b. SGD is the fast.ai function which does all these functions, we don't need to all these things explicitly. SGD function does all these things.
    c. Learner class is something interesting, where we need to pass all the inputs in below orders, and it does all things for you.
        1. Dataloaders.
        2. Nn.linear
        3. Optimizer function in our case SGD
        4. Loss function
        5. Metrics in our case accuracy.

        Learner does all these things in inside.
        And we give learner.fit we get same results.

    d. Till now, we did do with linear function. i.e x*w + b. we would need to convert to non linear function.
5.
6. He created a simple non linear function using linear function.
    a. Res = xb@w1+b
    b. Res = res.max(0) # replaces negatives with 0's.
    c. Res = xb@w1+b
    d. Return res
7. How do we turn into neural netwrok? Non linearity
    a. Res.max(0) is called Rectified linear unit. ReLu.
    If you add rectified linear unit or ReLu inshort it becomes Neural network. This tems sounds scary, but this is very simple.

b. Why do we do linear layer, relu, and linear layer? If we skip relu in between, then we can write both linear layers combinely, ultimaely it becomes linear layer. We can't make effective, than simple linear model. If you put non-linearity between linear layers, this is called **universal approximation theorem**. This can approxiate any function.

c. So, everything from here, simple tweaks.

d. So, he then created simple neural network using, nn.sequential nn.Linear, nn.ReLu, nn.Linear

e. Someone asked, as we are making negatives to zeros, doesnt it make any problem, yes it does, but it wont be problem. One trick is to make leaky Relu. We also look into howmany zeros we have, and change accordingly.

f. So, we can use smple_net instead linear1 function and call fit function. And trained for 40 epochs, it reached around 98% of accuracy.

g. So, we can solve any problem, to any level of accuracy given correct set of parameters.

h. We can use .model to see whats inside the model, which displays model information with parameters

i. Numbers that we learning are parameters, numbers that we are calculating are called activations. Didn't undersand

j. Activations are calculated, parameters are learned.

k. Rank 0: scaler, rank 1: vector, rank 2: matrix. Rank 3 and rank 4 tensors.

l. Is there any non-linearity to choose, as we have many of non-linearity functions.

m.

Pet_breeds

1. L is basically a enhanced list, it shows number of elemets in the list.
2. Most functions and methods in fastasi returns a collection, usually called L.
3. All the things that we learned in SGD he explained in this lesson, on pet classification.
4. First of all he pulled data of images from internet. i.e the upper case in the name is catt or lower case in te name is dot
5. He used to regular expression in order to get the name files using ls function which is available in the fastai.
6. He then used dataloader to prepare the dataset, i.e declare y values, define x values, define set of functions.
7. Splitter functions and resize function in the data loader function.
8. Aug_transforms we have seen in previous notebooks, it generates more synthetic data that we have. More like, generating more variety of data.

9. When we say, presize, it picks part of image randomly, and second aug transform, does kind of rotation and zooming. Second step done on GPU.
10. Left size is presize version, but the right side other libraries does, which is little distrotined, loss some information.
11. Presizing: when we say image size, it grabs part of image, and does the rotation/zooming and make it available using random crop and augmentation
12. It rans on GPU, running them on CPU makes it slower.
13. Presizing makes it more clear, and standard version makes little noise
14. When you make any mistakes while defining datablock function, we have something called summary function which helps us to decode/debug Datablock function.
15. There was a question, how does presize option works, when standard/original size is smaller?
    1. We can use squish, pad, crop to resize/presize the images in the class.

16. As soon as you have datablock is created and available, you can start using it/ passing to model. So, using cnn_learner.

17. Fastai picks, some lossfunction for you. At times it picks up cross entropyloss as function, What is cross entropy loss? Cross entropy loss is smiliar to sigmoid loss, but it is extended version of mnist_loss function, i.e. torch.where but this is binary loss.

18. Torch.where works when we have binary outcome, but this cross entropy works nicely for more than two categories.

19. Are we cleaning? Why are we not clearning data here? We are using top_losses function, and image classifer clearner helps us to clean the data.

20. Cross entropy loss: dls.one_batch() helps us to pick batch 1 for the dataset, which returns two variables, x and y. lets predict the probabilities of being an image gets classified to any one of the categories. When we see, this all probabilities are sum up to 1. How do we get these probabilities which are sum up to 1. Using a function called softmax.

21. **Softmax**: extension of sigmoid, to handle more than two outcomes. We usually have activation function(defines the output of class, using given several inputs) for each class in the output. Using softmax function we could have all the probabilities will sum up to 1. Since, each activation function returns the value,

we take exponent of that value and sum up all exponents, and divide each exponents with sum of all exponents.

22. As we aware that, exponents grow bigger and bigger, so, the bigger number we get from activation function, the bigger it is the probability and vice versa, i.e. smaller the number we get from activation function we get smaller the probability. Intutively, Softmax function is taking only one class at the end, and it is based on probability.

23. And softmax is the first part of cross entropy, and second part is log likelihood. We achieve by using nll_loss function, nothing but negative log loss. Which doesn't have any log in their formula. So, nll function assumes that you have already taken log, pytorch has a function called log_softmax which combines log and softmax.

24. As we have probabitlity values are in between 0 and 1, when we multiply each numbers, because of systems precision they get rounded off. But, when we take addition of logs and take exponents, it makes it easy in doing calculations.

25. So, we calculate softmax function, and then log likelihood of that, that combination is called cross entropy loss. So, the function nn.crossentropyloss() does log_softmax and then nll_loss. Because it is easier to take log of the softmax function and it does the log at the softmax function itself.

26. Nll_loss doesn't do any log of the function, it assumes that you have done log in softmax.

27. Why does the loss function need to be negative,

Img_show → how to show img_show