What does optimizer step do in pytorch

Last Updated: 08 Sep 2021

Recipe Objective

What does optimizer.step do?

As we have discussed earlier only about torch.optim package, in this package we have an optimizer.step method which will updates the parameters. There are two ways where this method can be implemented:
-- optimizer.step() This is a method which is simplified version that is supported by most optimizers, the function can be called once the gradients are computed using e.g .backward().
-- optimizer.step(closure) Here this method is being used when, some optimization algorithms like LBFGS and the Conjugate gradient need to reevaluate the function multiple times, so it is needed to pass a closure that allows them to recompute our model. The compute the loss, closure should clear the gradients, and return it.

Step 1 - Import library

```
import torch
```

Step 2 - Define parameters

```
batch, dim_in, dim_h, dim_out = 128, 2000, 200, 20
```

Here we are defining various parameters which are as follows:
batch - batch size
dim_in - Input dimension.
dim_out - Output dimension.

dim_h - hidden dimension.

Step 3 - Create Random tensors

```
input_X = torch.randn(batch, dim_in)
output_Y = torch.randn(batch, dim_out)
```

Here we are creating random tensors for holding the input and output data.

Step 4 - Define model and loss function

```
Adam_model = torch.nn.Sequential( torch.nn.Linear(dim_in, dim_h),
torch.nn.ReLU(), torch.nn.Linear(dim_h, dim_out), )
loss_fn = torch.nn.MSELoss(reduction='sum')
```

Step 5 - Define learning rate

```
rate_learning = 1e-4
```

Step 6 - Initialize optimizer

```
optim = torch.optim.Adam(SGD_model.parameters(),
lr=rate_learning)
```

Here we are Initializing our optimizer by using the "optim" package which will update the weights of the model for us. We are using SGD optimizer here the "optim" package which consist of many optimization algorithms.

Step 7 - Forward pass

```
for values in range(500):
   pred_y = Adam_model(input_X)
   loss = loss_fn(pred_y, output_Y)
   if values % 100 == 99:
      print(values, loss.item())
```

```
99 698.3545532226562
199 698.3545532226562
299 698.3545532226562
399 698.3545532226562
499 698.3545532226562
```

Here we are computing the predicted y by passing input_X to the model, after that computing the loss and then printing it.

Step 8 - Zero all gradients

```
optim.zero_grad()
```

Here before the backward pass we must zero all the gradients for the variables it will update which are nothing but the learnable weights of the model.

Step 9 - Backward pass

```
loss.backward()
```

Here we are computing the gradients of the loss w.r.t the model parameters.

Step 10 - Call step function

```
step = optim.step() step
```

Here we are calling the step function on an optimizer which will makes an update to its parameters.

{"mode":"full","isActive":false}