PyTorch

Machine Learning & Deep Learning (Barcha uchun)

<05> Chiziqli Regressiya (Linear Regression)

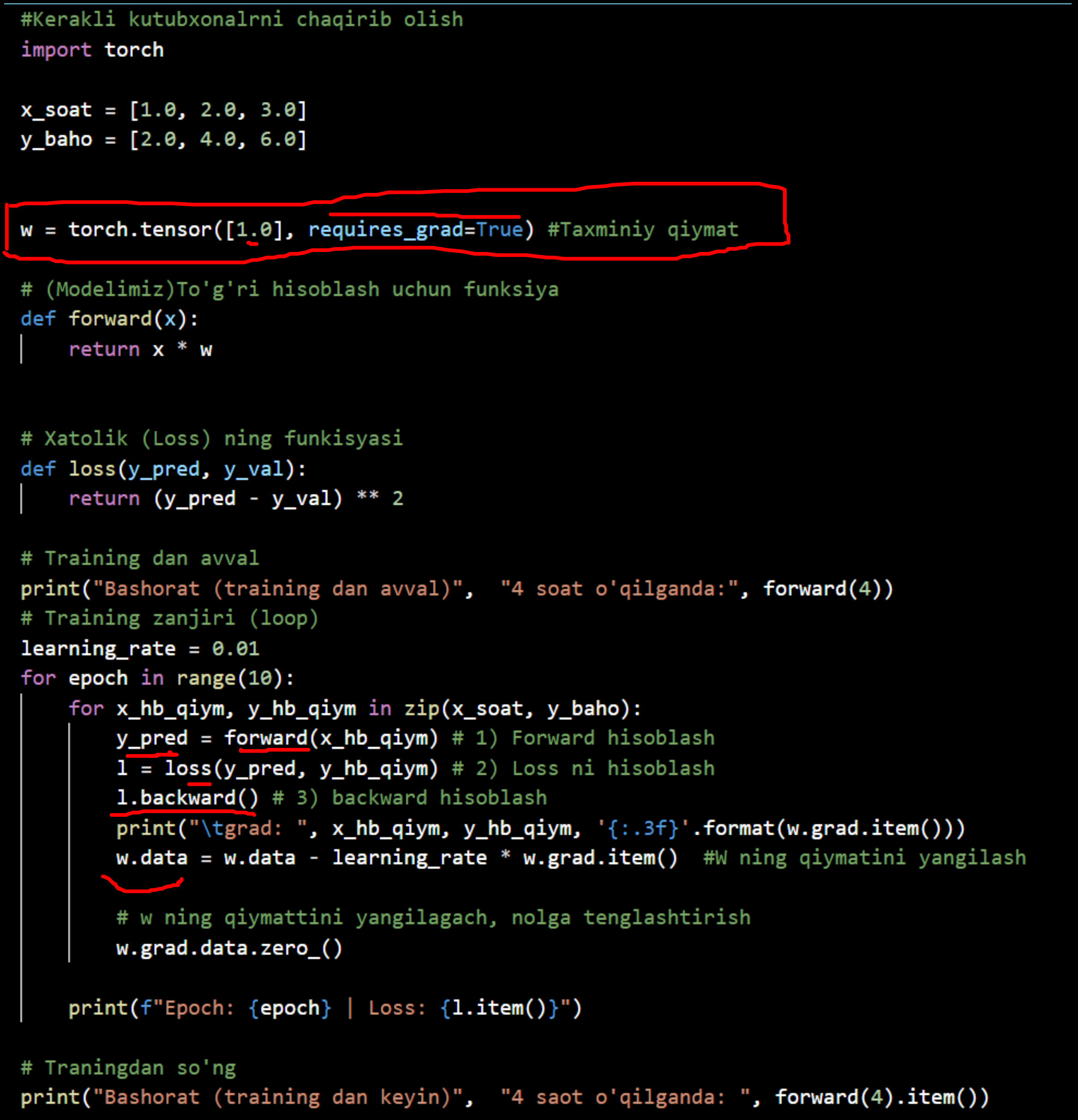
Mansurbek Abdullaev

https://uzbek.gitbook.io/ai/

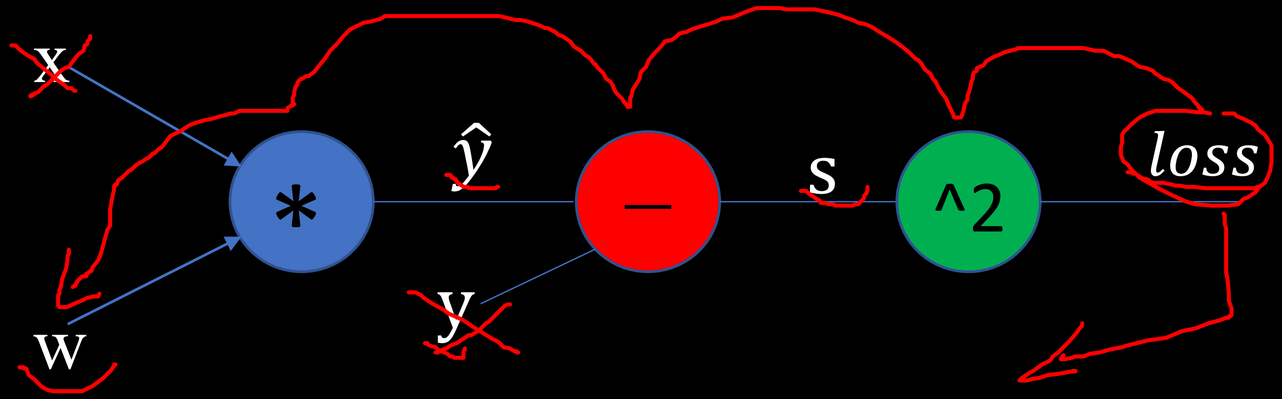
mansurbek.comchemai@gmail.com

@MansurbekUST

Pytorch forward&backward



```
#Kerakli kutubxonalrni chaqirib olish
import torch

x_soat = [1.0, 2.0, 3.0]
y_baho = [2.0, 4.0, 6.0]

w = torch.tensor([1.0], requires_grad=True) #Taxminiy qiymat

# (Modelimiz)To'g'ri hisoblash uchun funksiya
def forward(x):
    return x * w


# Xatolik (Loss) ning funkisyasi
def loss(y_pred, y_val):
    return (y_pred - y_val) ** 2

# Training dan avval
print("Bashorat (training dan avval)",  "4 soat o'qilganda:", forward(4))
# Training zanjiri (loop)
learning_rate = 0.01
for epoch in range(10):
    for x_hb_qiym, y_hb_qiym in zip(x_soat, y_baho):
        y_pred = forward(x_hb_qiym) # 1) Forward hisoblash
        l = loss(y_pred, y_hb_qiym) # 2) Loss ni hisoblash
        l.backward() # 3) backward hisoblash
        print("\tgrad: ", x_hb_qiym, y_hb_qiym, '{:.3f}'.format(w.grad.item()))
        w.data = w.data - learning_rate * w.grad.item()  #W ning qiymatini yangilash

        # w ning qiymattini yangilagach, nolga tenglashtirish
        w.grad.data.zero_()

    print(f"Epoch: {epoch} | Loss: {l.item()}")

# Traningdan so'ng
print("Bashorat (training dan keyin)",  "4 saot o'qilganda: ", forward(4).item())
```



PyTorch

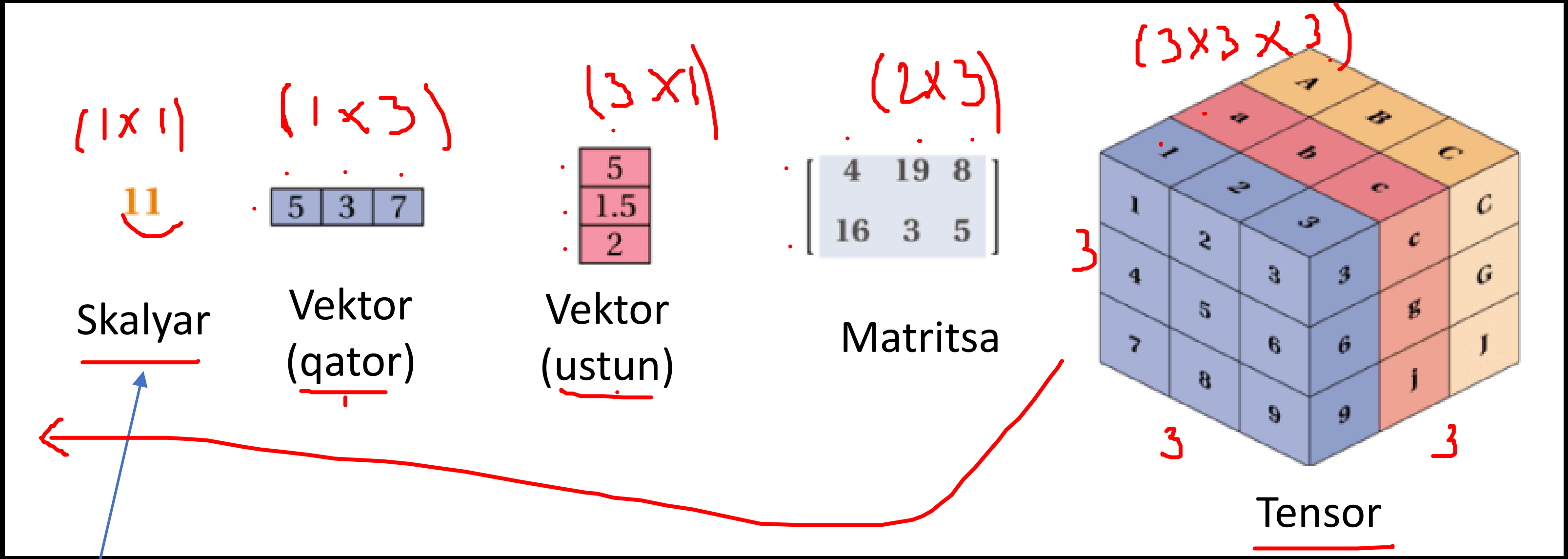
PyTorch Ritmi

1 Class (OOP) yordamida modelni qurib olish

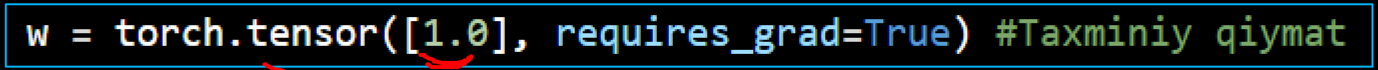
2 Loss va optimizer larni tanlash (PyTorch API dan)

3 O’rgatish (Training) sikli → forward, backward, step

Skalyar, Vektor, Matritsa, Tensor



torch.tensor

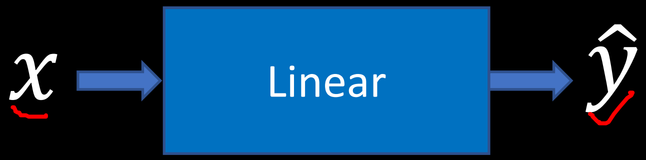


```
w = torch.tensor([1.0], requires_grad=True) #Taxminiy qiymat
```

1 Class (OOP) yordamida modelni qurib olish

```python
#Kerakli kutubxonalarni chaqirib olish
import torch
import numpy as np
#Ma'lumotlarni tensor ko'rinishida yuklab olish
x_soat = torch.Tensor([[1.0],
                       [2.0],
                       [3.0]])
y_baho = torch.Tensor([[2.0],
                       [4.0],
                       [6.0]])

#(1) Class yordamida model qurib olish --> "Model"
class Model(torch.nn.Module):
    def __init__(self):
        #Bu yerda torch.nn.Module bu yerda super class(Pytorch)
        super().__init__()
        #torch.nn.Linear(#kirish, #chiqish) chiziqli model
        self.linear = torch.nn.Linear(1,1) #1ta kirish & 1ta chiqish
    #Metod yordamida to'g'ri hisoblash funksiyasini kiritamiz(forward pass)
    def forward(self, x):
        y_pred = self.linear(x)
        return y_pred

#Bizning model
model=Model()
```

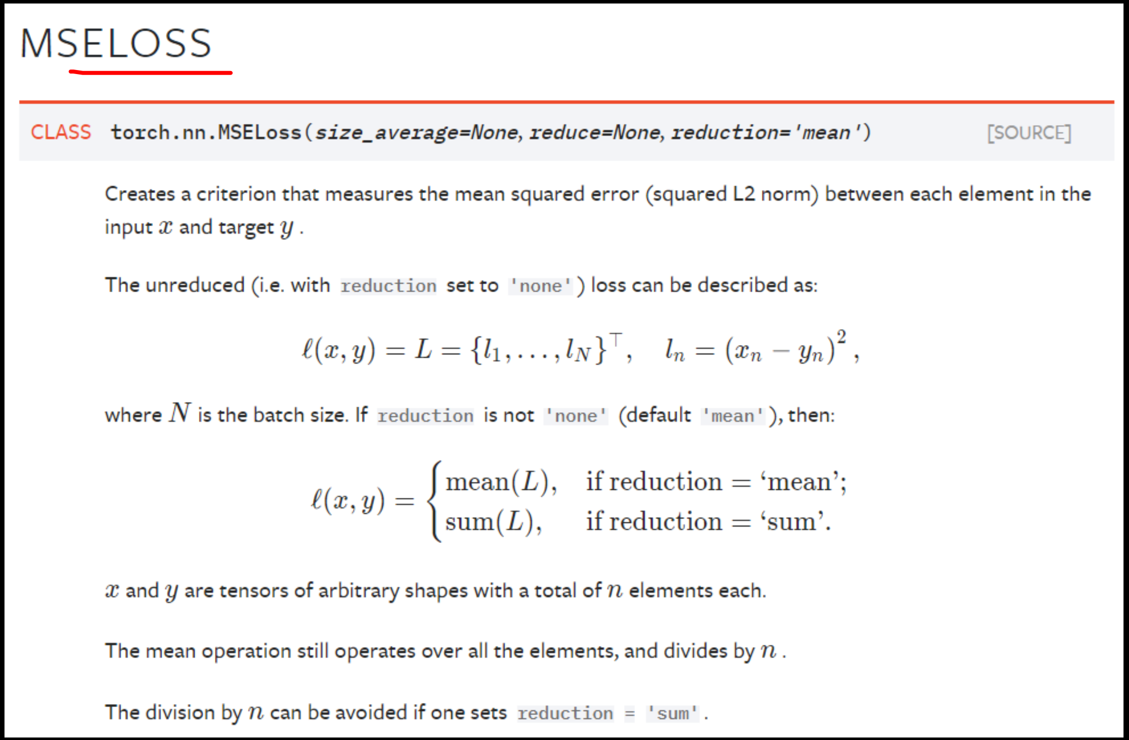
Soat (x)	Baho(y)
1	2
2	4
3	6
4	?



2 Loss va optimizer larni tanlash (PyTorch API dan)

```
#(2) Loss va optimizer larni tanlab olish
criterion = torch.nn.MSELoss(reduction='sum')
optimizer = torch.optim.SGD(model.parameters(), lr=0.01)
```

$$\boxed{w := w - \alpha \frac{\partial L}{\partial w}}$$



MSELOSS

CLASS `torch.nn.MSELoss(`*`size_average=None, reduce=None, reduction='mean'`*`)` [SOURCE]

Creates a criterion that measures the mean squared error (squared L2 norm) between each element in the input x and target y.

The unreduced (i.e. with `reduction` set to `'none'`) loss can be described as:

$$\ell(x, y) = L = \{l_1, \dots, l_N\}^\top, \quad l_n = (x_n - y_n)^2,$$

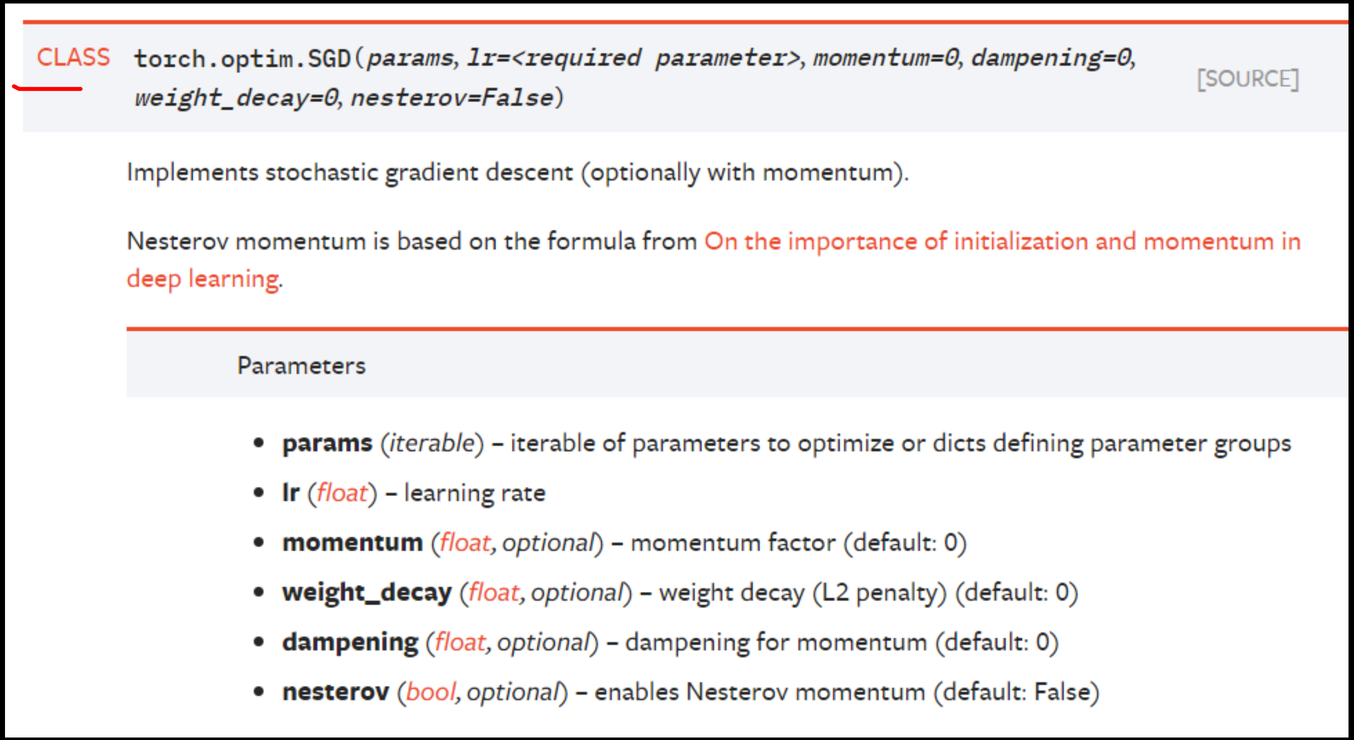
where N is the batch size. If `reduction` is not `'none'` (default `'mean'`), then:

$$\ell(x, y) = \begin{cases} \operatorname{mean}(L), & \text{if reduction} = \text{`mean';} \\ \operatorname{sum}(L), & \text{if reduction} = \text{`sum'.} \end{cases}$$

x and y are tensors of arbitrary shapes with a total of n elements each.

The mean operation still operates over all the elements, and divides by n.

The division by n can be avoided if one sets `reduction = 'sum'`.



CLASS `torch.optim.SGD(`*`params, lr=<required parameter>, momentum=0, dampening=0, weight_decay=0, nesterov=False`*`)` [SOURCE]

Implements stochastic gradient descent (optionally with momentum).

Nesterov momentum is based on the formula from On the importance of initialization and momentum in deep learning.

Parameters

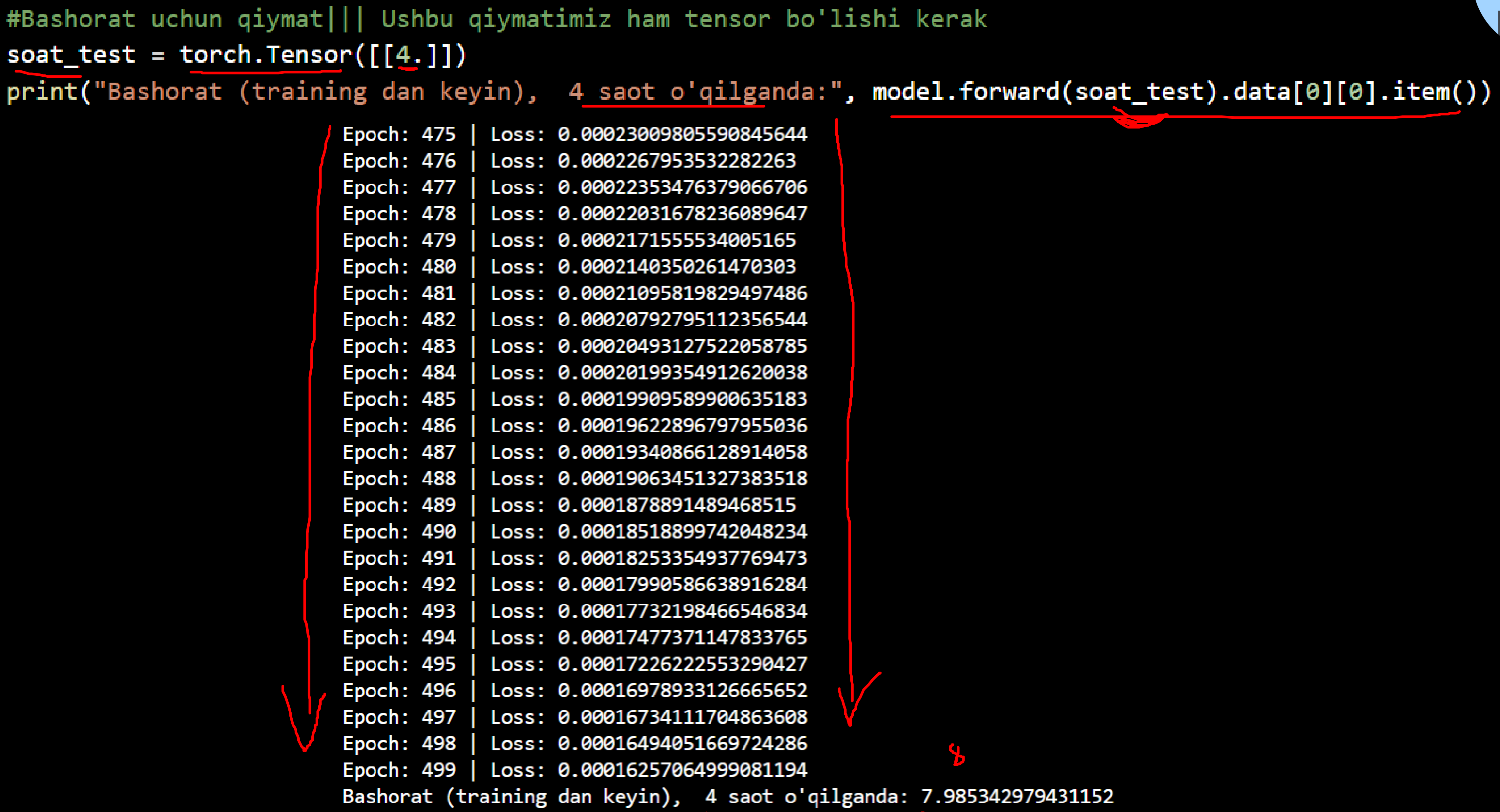
- **params** (*iterable*) – iterable of parameters to optimize or dicts defining parameter groups
- **lr** (*float*) – learning rate
- **momentum** (*float, optional*) – momentum factor (default: 0)
- **weight_decay** (*float, optional*) – weight decay (L2 penalty) (default: 0)
- **dampening** (*float, optional*) – dampening for momentum (default: 0)
- **nesterov** (*bool, optional*) – enables Nesterov momentum (default: False)

3 O’rgatish (Training) sikli → forward, backward, step

```
#(3) Training(3.1),   Backward(3.2), Step(3.3)
#(3.1)-->Training
for epoch in range(500):    #Epochlar soni 500
    y_pred = model(x_soat)
    #Loss|||xatolikni hisoblash va chop qilish
    loss = criterion(y_pred, y_baho)
    print(f'Epoch: {epoch} | Loss: {loss.item()} ')

    optimizer.zero_grad() #Har bir epoch uchun grad ni 0 ga tenglashtirib olish
    #(3.2)-->Backpropagation|||Teskari hisoblash
    loss.backward()
    #(3.3)--> Step||| w ning qiymatini  yangilash
    optimizer.step()
```

```
for x_hb_qiym, y_hb_qiym in zip(x_soat, y_baho):
        .

        .

        .
    w.data = w.data - learning_rate * w.grad.item()  #W ning qiymatini yangilash
```

TEST

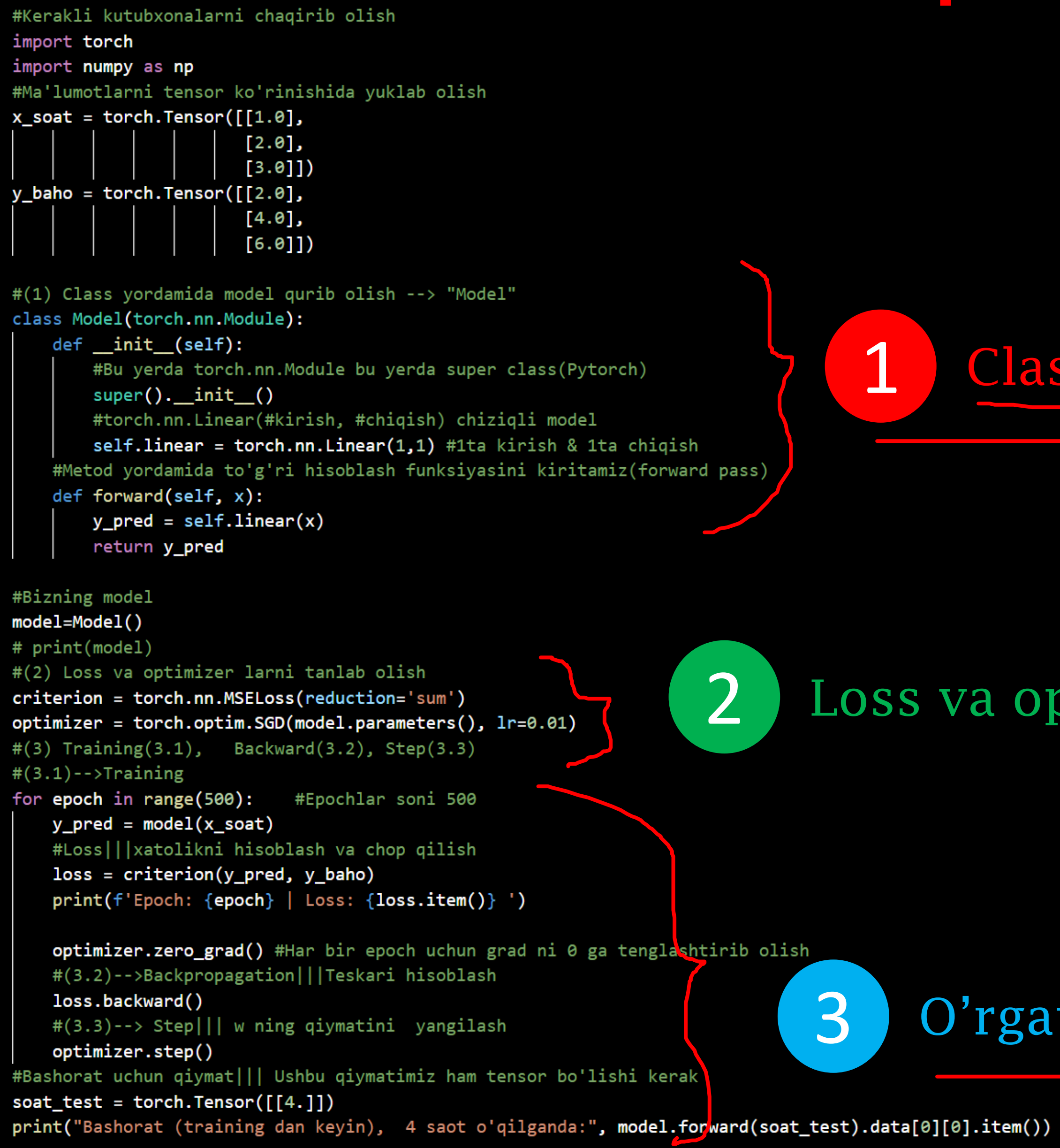


```
#Bashorat uchun qiymat||| Ushbu qiymatimiz ham tensor bo'lishi kerak
soat_test = torch.Tensor([[4.]])
print("Bashorat (training dan keyin),  4 saot o'qilganda:", model.forward(soat_test).data[0][0].item())
```

```
Epoch: 475 | Loss: 0.00023009805590845644
Epoch: 476 | Loss: 0.0002267953532282263
Epoch: 477 | Loss: 0.00022353476379066706
Epoch: 478 | Loss: 0.00022031678236089647
Epoch: 479 | Loss: 0.0002171555534005165
Epoch: 480 | Loss: 0.0002140350261470303
Epoch: 481 | Loss: 0.00021095819829497486
Epoch: 482 | Loss: 0.00020792795112356544
Epoch: 483 | Loss: 0.00020493127522058785
Epoch: 484 | Loss: 0.00020199354912620038
Epoch: 485 | Loss: 0.00019909589900635183
Epoch: 486 | Loss: 0.00019622896797955036
Epoch: 487 | Loss: 0.00019340866128914058
Epoch: 488 | Loss: 0.00019063451327383518
Epoch: 489 | Loss: 0.0001878891489468515
Epoch: 490 | Loss: 0.00018518899742048234
Epoch: 491 | Loss: 0.00018253354937769473
Epoch: 492 | Loss: 0.00017990586638916284
Epoch: 493 | Loss: 0.00017732198466546834
Epoch: 494 | Loss: 0.00017477371147833765
Epoch: 495 | Loss: 0.00017226222553290427
Epoch: 496 | Loss: 0.00016978933126665652
Epoch: 497 | Loss: 0.00016734111704863608
Epoch: 498 | Loss: 0.00016494051669724286
Epoch: 499 | Loss: 0.00016257064999081194
Bashorat (training dan keyin),  4 saot o'qilganda: 7.985342979431152
```

8

To’liq kod



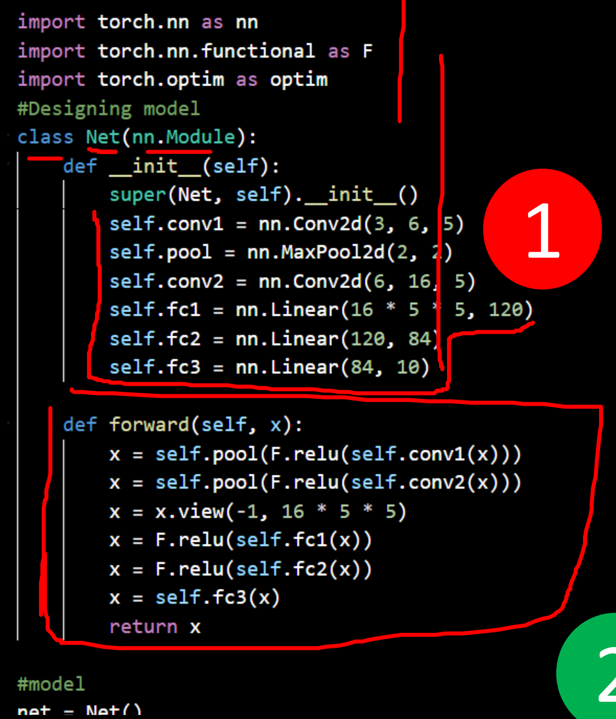
```python
#Kerakli kutubxonalarni chaqirib olish
import torch
import numpy as np
#Ma'lumotlarni tensor ko'rinishida yuklab olish
x_soat = torch.Tensor([[1.0],
                       [2.0],
                       [3.0]])
y_baho = torch.Tensor([[2.0],
                       [4.0],
                       [6.0]])

#(1) Class yordamida model qurib olish --> "Model"
class Model(torch.nn.Module):
    def __init__(self):
        #Bu yerda torch.nn.Module bu yerda super class(Pytorch)
        super().__init__()
        #torch.nn.Linear(#kirish, #chiqish) chiziqli model
        self.linear = torch.nn.Linear(1,1) #1ta kirish & 1ta chiqish
    #Metod yordamida to'g'ri hisoblash funksiyasini kiritamiz(forward pass)
    def forward(self, x):
        y_pred = self.linear(x)
        return y_pred

#Bizning model
model=Model()
# print(model)
#(2) Loss va optimizer larni tanlab olish
criterion = torch.nn.MSELoss(reduction='sum')
optimizer = torch.optim.SGD(model.parameters(), lr=0.01)
#(3) Training(3.1),   Backward(3.2), Step(3.3)
#(3.1)-->Training
for epoch in range(500):    #Epochlar soni 500
    y_pred = model(x_soat)
    #Loss|||xatolikni hisoblash va chop qilish
    loss = criterion(y_pred, y_baho)
    print(f'Epoch: {epoch} | Loss: {loss.item()} ')

    optimizer.zero_grad() #Har bir epoch uchun grad ni 0 ga tenglashtirib olish
    #(3.2)-->Backpropagation|||Teskari hisoblash
    loss.backward()
    #(3.3)--> Step||| w ning qiymatini  yangilash
    optimizer.step()
#Bashorat uchun qiymat||| Ushbu qiymatimiz ham tensor bo'lishi kerak
soat_test = torch.Tensor([[4.]])
print("Bashorat (training dan keyin),  4 saot o'qilganda:", model.forward(soat_test).data[0][0].item())
```

1 Class (OOP) yordamida modelni qurib olish

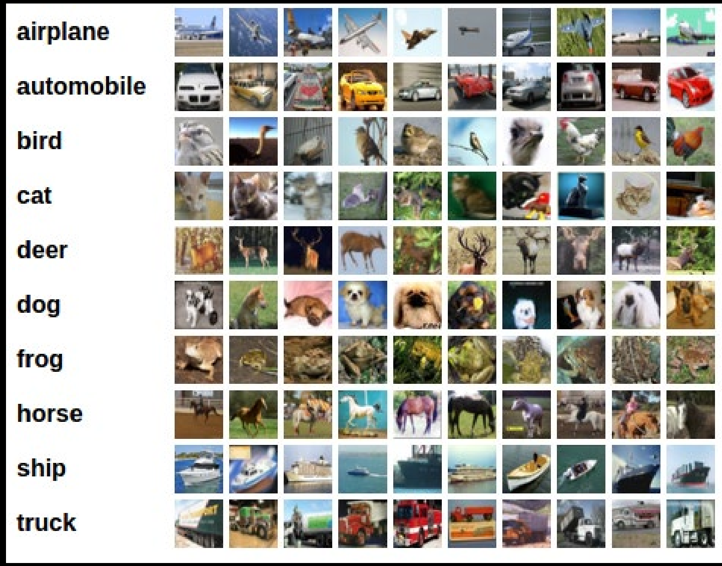
2 Loss va optimizer larni tanlash (PyTorch API dan)

3 O’rgatish (Training) sikli → forward, backward, step

Training CIFAR10 Classifiers



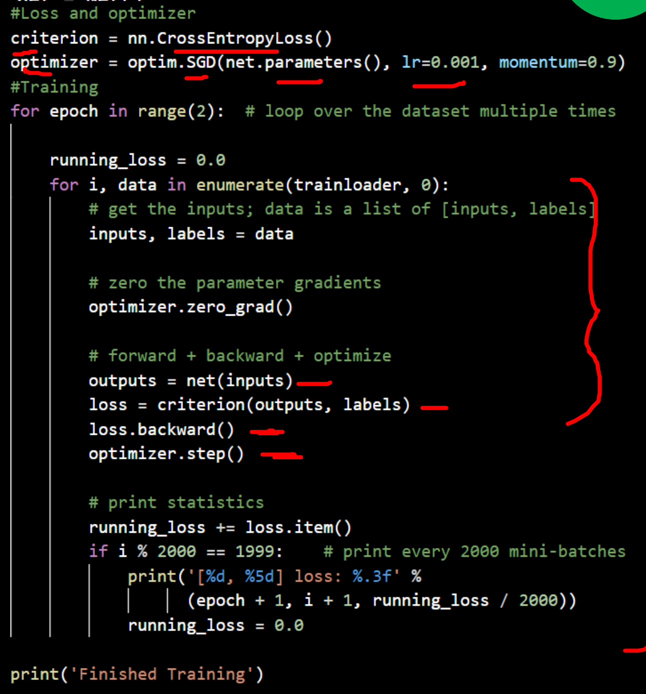
```
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
#Designing model
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(-1, 16 * 5 * 5)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x

#model
net = Net()
```



1 Class (OOP) yordamida modelni qurib olish

2 Loss va optimizer larni tanlash (PyTorch API dan)

3 O’rgatish (Training) sikli → forward, backward, step



```
#Loss and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)
#Training
for epoch in range(2):  # loop over the dataset multiple times

    running_loss = 0.0
    for i, data in enumerate(trainloader, 0):
        # get the inputs; data is a list of [inputs, labels]
        inputs, labels = data

        # zero the parameter gradients
        optimizer.zero_grad()

        # forward + backward + optimize
        outputs = net(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        # print statistics
        running_loss += loss.item()
        if i % 2000 == 1999:    # print every 2000 mini-batches
            print('[%d, %5d] loss: %.3f' %
                  (epoch + 1, i + 1, running_loss / 2000))
            running_loss = 0.0

print('Finished Training')
```

https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html

Vazifa 5-1 :

- torch.optim.Adagrad
- torch.optim.Adam
- torch.optim.Adamax
- torch.optim.ASGD
- torch.optim.LBFGS
- torch.optim.RMSprop
- torch.optim.Rprop
- torch.optim.SGD