Introduction to PyTorch

October 27, 2022

1 Lecture 16: PyTorch Crash Course

1.1 Tensor Initialization

See the full API at https://pytorch.org/docs/stable/tensors.html

[1]:
```
import torch
import numpy as np
```

[2]:
```
x=torch.tensor([1,2,3,4])
print(x)
```

```
tensor([1, 2, 3, 4])
```

[3]:
```
x.shape
```

[3]:
```
torch.Size([4])
```

[4]:
```
type(x)
```

[4]:
```
torch.Tensor
```

[5]:
```
x.dtype
```

[5]:
```
torch.int64
```

[6]:
```
x=torch.tensor([1,2,3,4]).type(torch.float)
x
```

[6]:
```
tensor([1., 2., 3., 4.])
```

[7]:
```
x.dtype
```

[7]:
```
torch.float32
```

[8]:
```
x=torch.FloatTensor([1,2,3,4])
x
```

[8]:
```
tensor([1., 2., 3., 4.])
```

```
[9]: x.dtype
```

```
[9]: torch.float32
```

```
[10]: y = torch.FloatTensor([[1,2,3,4],[5,6,7,8]])
      y
```

```
[10]: tensor([[1., 2., 3., 4.],
              [5., 6., 7., 8.]])
```

```
[11]: torch.zeros(5)
```

```
[11]: tensor([0., 0., 0., 0., 0.])
```

```
[12]: torch.ones(5)
```

```
[12]: tensor([1., 1., 1., 1., 1.])
```

```
[13]: torch.rand(5)
```

```
[13]: tensor([0.2323, 0.2190, 0.2919, 0.2940, 0.9902])
```

```
[14]: torch.eye(5)
```

```
[14]: tensor([[1., 0., 0., 0., 0.],
              [0., 1., 0., 0., 0.],
              [0., 0., 1., 0., 0.],
              [0., 0., 0., 1., 0.],
              [0., 0., 0., 0., 1.]])
```

```
[15]: torch.FloatTensor(np.random.randn(5))
```

```
[15]: tensor([-0.9962, -0.0668,  0.1160,  0.5321,  2.0040])
```

1.2 Linear Algebra

```
[16]: x=torch.FloatTensor([[1,2,3,4]])
      z=torch.FloatTensor([[1,2,3,4],[5,6,7,8]])
      print("x=",x)
      print("z=",z)
```

```
x= tensor([[1., 2., 3., 4.]])
z= tensor([[1., 2., 3., 4.],
        [5., 6., 7., 8.]])
```

```
[17]: 5*x
```

[17]: tensor([[5., 10., 15., 20.]])

[18]:
```
x+5
```

[18]: tensor([[6., 7., 8., 9.]])

[19]:
```
x+x
```

[19]: tensor([[2., 4., 6., 8.]])

[20]:
```
x.T
```

[20]:
```
tensor([[1.],
        [2.],
        [3.],
        [4.]])
```

[21]:
```
x@x.T
```

[21]: tensor([[30.]])

[22]:
```
x.T@x
```

[22]:
```
tensor([[ 1.,  2.,  3.,  4.],
        [ 2.,  4.,  6.,  8.],
        [ 3.,  6.,  9., 12.],
        [ 4.,  8., 12., 16.]])
```

[23]:
```
x@z.T
```

[23]: tensor([[30., 70.]])

1.3 Broadcasting

[24]:
```
x=torch.FloatTensor([[1,2,3,4]])
y=torch.FloatTensor([[-1,-2]]).T
z=torch.FloatTensor([[1,2,3,4],[5,6,7,8]])
print("x=",x)
print("y=",y)
print("z=",z)
```

```
x= tensor([[1., 2., 3., 4.]])
y= tensor([[-1.],
        [-2.]])
z= tensor([[1., 2., 3., 4.],
        [5., 6., 7., 8.]])
```

[25]:
```
x+y
```

[25]:
```
tensor([[ 0.,  1.,  2.,  3.],
        [-1.,  0.,  1.,  2.]])
```

[26]:
```
x*y
```

[26]:
```
tensor([[-1., -2., -3., -4.],
        [-2., -4., -6., -8.]])
```

[27]:
```
x+z
```

[27]:
```
tensor([[ 2.,  4.,  6.,  8.],
        [ 6.,  8., 10., 12.]])
```

[28]:
```
x*z
```

[28]:
```
tensor([[ 1.,  4.,  9., 16.],
        [ 5., 12., 21., 32.]])
```

[29]:
```
y+z
```

[29]:
```
tensor([[0., 1., 2., 3.],
        [3., 4., 5., 6.]])
```

[30]:
```
y*z
```

[30]:
```
tensor([[ -1.,  -2.,  -3.,  -4.],
        [-10., -12., -14., -16.]])
```

1.4 Elementwise Functions

[31]:
```
x=torch.FloatTensor([[1,2,-3,-4]])
x
```

[31]:
```
tensor([[ 1.,  2., -3., -4.]])
```

[32]:
```
torch.pow(x,2)
```

[32]:
```
tensor([[ 1.,  4.,  9., 16.]])
```

[33]:
```
x**2
```

[33]:
```
tensor([[ 1.,  4.,  9., 16.]])
```

[34]:
```
torch.exp(x)
```

[34]: tensor([[2.7183, 7.3891, 0.0498, 0.0183]])

[35]:
```
torch.log(x)
```

[35]: tensor([[0.0000, 0.6931,   nan,   nan]])

[36]:
```
torch.sin(x)
```

[36]: tensor([[0.8415, 0.9093, -0.1411, 0.7568]])

[37]:
```
torch.abs(x)
```

[37]: tensor([[1., 2., 3., 4.]])

1.5 Matrix Functions

[38]:
```
z = torch.FloatTensor([[2,1],[1,3]])
z
```

[38]:
```
tensor([[2., 1.],
        [1., 3.]])
```

[39]:
```
torch.det(z)
```

[39]: tensor(5.)

[40]:
```
torch.inverse(z)
```

[40]:
```
tensor([[ 0.6000, -0.2000],
        [-0.2000,  0.4000]])
```

[41]:
```
torch.trace(z)
```

[41]: tensor(5.)

1.6 Aggregation Functions

[42]:
```
x=torch.FloatTensor([[1,2,3,4]])
z=torch.FloatTensor([[1,2,3,4],[5,6,7,8]])
print("x=",x)
print("z=",z)
```

```
x= tensor([[1., 2., 3., 4.]])
z= tensor([[1., 2., 3., 4.],
        [5., 6., 7., 8.]])
```

[43]:
```
x.sum()
```

[43]: tensor(10.)

[44]:
```
z.sum()
```

[44]: tensor(36.)

[45]:
```
z.sum(axis=0)
```

[45]: tensor([6., 8., 10., 12.])

[46]:
```
z.sum(axis=1)
```

[46]: tensor([10., 26.])

[47]:
```
z.sum(axis=1,keepdims=True)
```

[47]:
```
tensor([[10.],
        [26.]])
```

[48]:
```
x.mean()
```

[48]: tensor(2.5000)

[49]:
```
x.prod()
```

[49]: tensor(24.)

[50]:
```
x.min()
```

[50]: tensor(1.)

[51]:
```
x.max()
```

[51]: tensor(4.)

[52]:
```
z.min()
```

[52]: tensor(1.)

[53]:
```
v,ind = z.min(axis=0)
print(v)
print(ind)
```

```
tensor([1., 2., 3., 4.])
tensor([0, 0, 0, 0])
```

[54]:
```
v,ind = z.min(axis=1)
print(v)
print(ind)
```

```
tensor([1., 5.])
tensor([0, 0])
```

1.7 Automatic Differentiation

[55]:
```
x=torch.FloatTensor([[1,2,3,4],[-1,-2,-3,-4]])
w=torch.tensor([[-1.0],[-2.0],[2.0],[1.0]],requires_grad=True)
b=torch.tensor(-1.0,requires_grad=True)
y=torch.tensor([[1.0],[0.0]])
print("x=",x)
print("w=",w)
print("b=",b)
```

```
x= tensor([[ 1.,  2.,  3.,  4.],
        [-1., -2., -3., -4.]])
w= tensor([[-1.],
        [-2.],
        [ 2.],
        [ 1.]], requires_grad=True)
b= tensor(-1., requires_grad=True)
```

[56]:
```
x.requires_grad
```

[56]: False

[57]:
```
w.requires_grad
```

[57]: True

[58]:
```
b.requires_grad
```

[58]: True

[59]:
```
a=x[0,:]@w +b
a
```

[59]: tensor([4.], grad_fn=<AddBackward0>)

[60]:
```
a.requires_grad
```

[60]: True

[61]:
```
a.backward()
print("grad_w:",w.grad,"grad_b:",b.grad)
```

```
grad_w: tensor([[1.],
        [2.],
        [3.],
        [4.]]) grad_b: tensor(1.)
```

[62]:
```
a=x[0,:]@w +b
a.backward()
print("grad_w:",w.grad,"grad_b:",b.grad)
```

```
grad_w: tensor([[2.],
        [4.],
        [6.],
        [8.]]) grad_b: tensor(2.)
```

[63]:
```
w.grad=0*w.grad
b.grad=0*b
a=x[0,:]@w +b
a.backward()
print("grad_w:",w.grad,"grad_b:",b.grad)
```

```
grad_w: tensor([[1.],
        [2.],
        [3.],
        [4.]]) grad_b: tensor(1., grad_fn=<MulBackward0>)
```

[64]:
```
def mse(y,yhat):
    return torch.mean((y-yhat)**2)

w.grad=0*w.grad
b.grad=0*b

yhat = x@w+b
loss = mse(y,yhat)
loss.backward()
print("grad_w:",w.grad,"grad_b:",b.grad)
```

```
grad_w: tensor([[ 9.],
        [18.],
        [27.],
        [36.]]) grad_b: tensor(-3., grad_fn=<MulBackward0>)
```

1.8 Neural Network Modules

See the full API at https://pytorch.org/docs/stable/nn.html

[65]:
```
X = torch.randn(1000,2)
Y = -3*X[:,[0]] + 2*X[:,[1]]**2 + 3
Y[:5,:]
```

[65]: tensor([[8.1985],
        [4.5534],
        [6.4657],
        [6.9933],
        [2.6479]])

[66]:
```
import torch.nn as nn

class linear(nn.Module):
    def __init__(self,d,k):
        super(linear, self).__init__()
        self.w = nn.Parameter(torch.rand(d,k))
        self.b = nn.Parameter(torch.rand(1,k))

    def forward(self,x):
        return x@self.w + self.b


model = linear(2,1)
model.forward(X[:10,:])
```

[66]: tensor([[0.8807],
        [0.7144],
        [1.2655],
        [0.5497],
        [0.8091],
        [0.4771],
        [0.8583],
        [1.0352],
        [0.9309],
        [0.8387]], grad_fn=<AddBackward0>)

[67]:
```
Yhat = model.forward(X)
loss = mse(Yhat,Y)
loss
```

[67]: tensor(35.4643, grad_fn=<MeanBackward0>)

[68]:
```
class relu_mlp(nn.Module):
    def __init__(self,d,k):
        super(relu_mlp, self).__init__()
        self.l1   = nn.Linear(d,k)
        self.relu = nn.ReLU()
        self.out  = nn.Linear(k,1)

    def forward(self,x):
        return self.out(self.relu(self.l1(x)))
```

[69]:
```python
model = relu_mlp(2,3)
model.forward(X[:10,:])
```

[69]:
```
tensor([[-0.0879],
        [ 0.0092],
        [ 0.0627],
        [ 0.0424],
        [ 0.0261],
        [ 0.0063],
        [-0.1009],
        [ 0.3049],
        [ 0.0181],
        [ 0.0198]], grad_fn=<AddmmBackward0>)
```

[70]:
```python
Yhat = model.forward(X)
loss = mse(Yhat,Y)
loss
```

[70]:
```
tensor(43.2361, grad_fn=<MeanBackward0>)
```

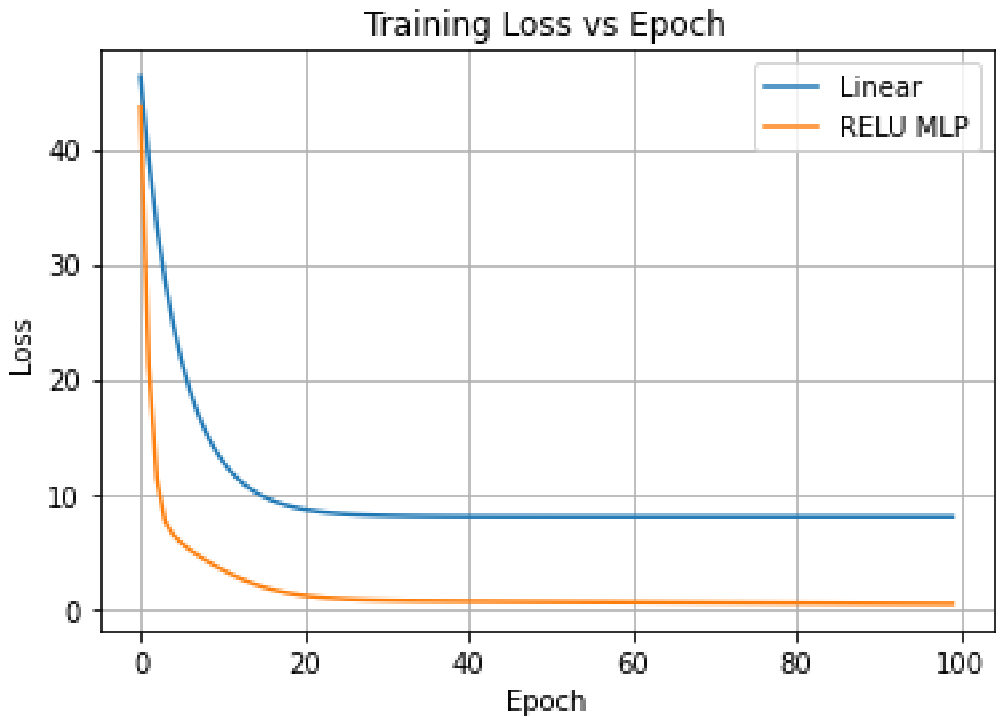
1.9 Optimization

[71]:
```python
def fit(model, lr, max_iter):
    optimizer = torch.optim.SGD(model.parameters(), lr=lr)
    losses=[]
    for i in range(max_iter):
        Yhat = model.forward(X)
        loss = mse(Yhat,Y)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        losses.append(loss.detach().numpy().item())
        if(i%10==0):
            print("%d %.2f"%(i, losses[-1]))

    return(losses)

models = {"Linear":linear(2,1),"RELU MLP":relu_mlp(2,20)}
losses = {}
for m in models:
    print("Learning model: %s"%m)
    losses[m] = fit(models[m], 0.05, 100)
    print()
```

```
Learning model: Linear
0 46.35
```

```
10 12.85
20 8.63
30 8.09
40 8.02
50 8.02
60 8.01
70 8.01
80 8.01
90 8.01

Learning model: RELU MLP
0 43.69
10 3.38
20 1.11
30 0.71
40 0.64
50 0.60
60 0.56
70 0.52
80 0.48
90 0.44
```

[72]:
```
import matplotlib.pyplot as plt
plt.figure()
for m in models:
    plt.plot(losses[m])
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.legend(list(models.keys()))
plt.title("Training Loss vs Epoch")
plt.grid(True)
```



[73]:
```
Xte = torch.randn(1000,2)
Yte = -3*Xte[:,[0]] + 2*Xte[:,[1]]**2 + 3

te_err={}
for m in models:
    te_err[m]=mse(Yte,models[m].forward(Xte))
    print("%s test error: %.2f"%(m,te_err[m]))
```

```
Linear test error: 7.47
RELU MLP test error: 0.40
```