PyTorch Cheat Sheet

In [1]:
```
import torch
import torch.nn as nn
from torch.utils.data import Dataset, DataLoader
import torchvision
import torchvision.transforms as transforms
```

Creating Tensors

In [2]:
```
torch.empty(2), torch.ones(2), torch.zeros(2)
```

Out[2]:
```
(tensor([1.1422e-40, 4.5703e-35]), tensor([1., 1.]), tensor([0., 0.]))
```

In [3]:
```
torch.rand(5), torch.randn(5)
```

Out[3]:
```
(tensor([0.9261, 0.7819, 0.3198, 0.2448, 0.3161]),
 tensor([ 0.0261, -1.8937, -1.3276,  2.1708, -1.8366]))
```

In [4]:
```
t = torch.tensor(1)
t
```

Out[4]:
```
tensor(1)
```

In [5]:
```
t = torch.tensor([1])
t
```

Out[5]:
```
tensor([1])
```

In [6]:
```
torch.tensor([1, 2, 3]).dtype
```

Out[6]:
```
torch.int64
```

In [7]:
```
torch.tensor([1.0, 2.0, 3.0]).dtype
```

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

In [8]:
```
torch.tensor([1], dtype=torch.int32).dtype
```

Out[8]:
```
torch.int32
```

In [9]:
```
v = np.array([1, 2, 3])
t = torch.from_numpy(v)
t
```

Out[9]:
```
tensor([1, 2, 3])
```

In [10]:
```
v += 1
t
```

Out[10]:
```
tensor([2, 3, 4])
```

In [11]:
```
u = t.numpy()
u
```

Out[11]:
```
array([2, 3, 4])
```

In [12]:
```
t.add_(1)
u
```

Out[12]:
```
array([3, 4, 5])
```

In [13]:
```
t = torch.tensor(range(1,7)).view(-1, 3)
t
```

Out[13]:
```
tensor([[1, 2, 3],
        [4, 5, 6]])
```

Indexing & Slicing

In [14]:
```
t[0]
```

Out[14]:
```
tensor([1, 2, 3])
```

In [15]:
```
t[0, 1]
```

Out[15]:
```
tensor(2)
```

In [16]:
```
s = t[0, 1].item()
s, type(s)
```

Out[16]:
```
(2, int)
```

In [17]:
```
t[:, 2], t[1, :]
```

Out[17]:
```
(tensor([3, 6]), tensor([4, 5, 6]))
```

Arithmetic

In [18]:
```
t + t
```

Out[18]:
```
tensor([[ 2,  4,  6],
        [ 8, 10, 12]])
```

In [19]:
```
t * t
```

Out[19]:
```
tensor([[ 1,  4,  9],
        [16, 25, 36]])
```

In [20]:
```
t ** 2
```

Out[20]:
```
tensor([[ 1,  4,  9],
        [16, 25, 36]])
```

In [21]:
```
t @ t.T  # 2x3 . 3x2 = 2x2
```

Out[21]:
```
tensor([[14, 32],
        [32, 77]])
```

In [22]:
```
t.add(2)
```

Out[22]:
```
tensor([[3, 4, 5],
        [6, 7, 8]])
```

In [23]:
```
t.add_(2), t
```

Out[23]:
```
(tensor([[3, 4, 5],
         [6, 7, 8]]),
 tensor([[3, 4, 5],
         [6, 7, 8]]))
```

Aggregation

In [24]:
```
t
```

Out[24]:
```
tensor([[3, 4, 5],
        [6, 7, 8]])
```

In [25]:
```
t.max(), torch.max(t)
```

Out[25]:
```
(tensor(8), tensor(8))
```

In [26]:
```
t.max(dim=0), t.min(dim=0)
```

Out[26]:
```
(torch.return_types.max(
 values=tensor([6, 7, 8]),
 indices=tensor([1, 1, 1])),
 torch.return_types.min(
 values=tensor([3, 4, 5]),
 indices=tensor([0, 0, 0])))
```

In [27]:
```
torch.max(t, dim=1).values, t.max(dim=1).indices
```

Out[27]:
```
(tensor([5, 8]), tensor([2, 2]))
```

In [28]:
```
t.argmax(dim=1), t.argmax(dim=1, keepdim=True)
```

Out[28]:
```
(tensor([2, 2]),
 tensor([[2],
         [2]]))
```

AutoGrad

In [29]:
```
t = torch.tensor(1.0, requires_grad=True)
t
```

Out[29]:
```
tensor(1., requires_grad=True)
```

In [30]:
```
t.detach(), t
```

Out[30]:
```
(tensor(1.), tensor(1., requires_grad=True))
```

In [31]:
```
t.requires_grad_(False), t
```

Out[31]:
```
(tensor(1.), tensor(1.))
```

In [32]:
```
x = torch.tensor(1.5, requires_grad=True)
y = 3*x**2
y
```

Out[32]:
```
tensor(6.7500, grad_fn=<MulBackward0>)
```

In [33]:
```
y.backward()
x.grad
```

Out[33]:
```
tensor(9.)
```

In [34]:
```
x.grad.zero_(), x.grad
```

Out[34]:
```
(tensor(0.), tensor(0.))
```

In [35]:
```
with torch.no_grad():
    z = 0.5*x
try:
    z.backward()
except:
    print('Exception')
z, x.grad
```

Out[35]:
```
Exception
(tensor(0.7500), tensor(0.))
```

nn.CrossEntropyLoss(), nn.NLLLoss()

In [36]:
```
logits = torch.tensor([[1.0, 2.0, 3.0]])
probs = torch.softmax(logits, dim=-1)
probs
```

Out[36]:
```
tensor([[0.0900, 0.2447, 0.6652]])
```

In [37]:
```
ce_loss = nn.CrossEntropyLoss()
y0 = torch.tensor([0])
y2 = torch.tensor([2])
# CrossEntropyLoss()() takes in *logits (NOT probs)* and *class labels*
ce_loss(logits, y0), ce_loss(logits, y2)
```

Out[37]:
```
(tensor(2.4076), tensor(0.4076))
```

In [38]:
```
-np.log(probs.numpy())
```

Out[38]:
```
array([[2.408, 1.408, 0.408]], dtype=float32)
```

In [39]:
```
nll_loss = nn.NLLLoss()
nll_loss(probs, y0), nll_loss(probs, y2)
```

Out[39]:
```
(tensor(-0.0900), tensor(-0.6652))
```

nn.BCELoss()

In [40]:
```
bce_loss = nn.BCELoss()
bce_loss(torch.tensor([0.9]), torch.tensor([0.0]))
```

Out[40]:
```
tensor(2.3026)
```

In [41]:
```
-(1-0)*np.log(1 - 0.9)
```

Out[41]:
```
2.303
```

Backpropagation

In [42]:
```
X = torch.tensor([1, 2, 3, 4], dtype=torch.float32).view(-1, 1)
Y = torch.tensor([2, 4, 6, 8], dtype=torch.float32).view(-1, 1)

w = torch.tensor([[0.0]], dtype=torch.float32, requires_grad=True)

def forward(X, w):
    return X @ w  # 4x1 . 1x1 = 4x1

def loss(Y_pred, Y):
    return ((Y_pred - Y)**2).mean()

learning_rate = 0.01

for epoch in range(40):
    Y_pred = forward(X, w)
    j = loss(Y, Y_pred)
    j.backward()
    with torch.no_grad():
        w -= learning_rate * w.grad
    w.grad.zero_()

    if epoch % 10 == 9:
        print(f'epoch {epoch+1}: w = {w.item():.3f}, loss = {j.item():.8f}')
```

```
epoch 10: w = 1.606, loss = 1.60939169
epoch 20: w = 1.922, loss = 0.06237914
epoch 30: w = 1.985, loss = 0.00241778
epoch 40: w = 1.997, loss = 0.00009371
```

Optimizer & LR-Scheduler

In [43]:
```
w = torch.tensor([[0.0]], dtype=torch.float32, requires_grad=True)
optimizer = torch.optim.SGD([w], lr=learning_rate)

loss = nn.MSELoss()

for epoch in range(40):
    Y_pred = forward(X, w)
```

```
    j = loss(Y_pred, Y)
    j.backward()
    optimizer.step()
    optimizer.zero_grad()

    if epoch % 10 == 9:
        print(f'epoch {epoch+1}: w = {w.item():.3f}, loss = {j.item():.8f}')
```

```
epoch 10: w = 1.606, loss = 1.60939169
epoch 20: w = 1.922, loss = 0.06237914
epoch 30: w = 1.985, loss = 0.00241778
epoch 40: w = 1.997, loss = 0.00009371
```

In [44]:
```
lr_scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=10, gamma=0

for epoch in range(30):
    Y_pred = forward(X, w)
    j = loss(Y_pred, Y)
    j.backward()
    optimizer.step()
    optimizer.zero_grad()
    lr_scheduler.step()

    if epoch % 10 == 9:
        lr = optimizer.state_dict()['param_groups'][0]['lr']
        print(f'epoch {epoch+1}: w = {w.item():.3f}, lr = {j.item():.8f}')
```

```
epoch 10: w = 1.999, lr = 0.00000363
epoch 20: w = 2.000, lr = 0.00000064
epoch 30: w = 2.000, lr = 0.00000028
```

nn.Linear()

In [45]:
```
model = nn.Linear(in_features=1, out_features=1)
optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate)

for epoch in range(40):
    Y_pred = model(X)
    j = loss(Y_pred, Y)
    j.backward()
    optimizer.step()
    optimizer.zero_grad()

    if epoch % 10 == 9:
        w, b = model.parameters()  # unpack parameters
        print(f'epoch {epoch+1}: w = {w.item():.3f}, loss = {j.item():.8f}')
```

```
epoch 10: w = 1.948, loss = 0.65129113
epoch 20: w = 2.122, loss = 0.05157164
epoch 30: w = 2.147, loss = 0.03403451
epoch 40: w = 2.147, loss = 0.03167749
```

In [46]:
```
predicted = model(X).detach()
predicted
```

Out[46]:
```
tensor([[1.7123],
        [3.8591],
        [6.0059],
        [8.1526]])
```

In [47]:
```
with torch.no_grad():
    predicted = model(X)
predicted
```

Out[47]:
```
tensor([[1.7123],
        [3.8591],
        [6.0059],
        [8.1526]])
```

nn.Module, Logistic Regression

In [48]:
```
X = torch.tensor([[2, 1], [1, 2]], dtype=torch.float32)
Y = torch.tensor([0, 1], dtype=torch.float32).view(-1, 1)

class LogR(nn.Module):
    def __init__(self, in_features):
        super().__init__()
        self.linear = nn.Linear(in_features, 1)

    def forward(self, x):
        x = self.linear(x)
        x = torch.sigmoid(x)
        return x

model = LogR(X.shape[-1])
optimizer = torch.optim.SGD(model.parameters(), lr=0.1)
loss = nn.BCELoss()

for epoch in range(50):
    Y_pred = model(X)
    j = loss(Y_pred, Y)
    j.backward()
    optimizer.step()
    optimizer.zero_grad()

    if epoch % 10 == 9:
        print(f'epoch {epoch+1}: loss = {j.item():.8f}')
```

```
epoch 10: loss = 0.58357650
epoch 20: loss = 0.47398025
epoch 30: loss = 0.40872616
epoch 40: loss = 0.35784590
epoch 50: loss = 0.31676430
```

In [49]:
```
with torch.no_grad():
    Y_pred = model(X)
Y_pred
```

Out[49]:
```
tensor([[0.2693],
        [0.7317]])
```

Misc

In [50]:
```
device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
torch.tensor([1], device=device), torch.tensor([2]).to(device)
```

Out[50]:
```
(tensor([1]), tensor([2]))
```

Model Saving & Loading

In [51]:
```
class Model(nn.Module):
    def __init__(self):
        super().__init__()
        self.linear = nn.Linear(2, 1)

    def forward(self, x):
        x = self.linear(x)
        x = torch.sigmoid(x)
        return x
```

In [52]:
```
model = Model()
torch.save(model, 'model.pth')
# ---
model = torch.load('model.pth')
model.eval()
for param in model.parameters():
    print(param)
```

```
Parameter containing:
tensor([[-0.0505,  0.6279]], requires_grad=True)
Parameter containing:
tensor([0.0782], requires_grad=True)
```

In [53]:
```
model = Model()
torch.save(model.state_dict(), 'model_state.pth')
print(model.state_dict())
# ---
model = Model()
model.load_state_dict(torch.load('model_state.pth'))
# model.load_state_dict(torch.load('model_state.path', map_location=device))
model.eval()
print(model.state_dict())
```

```
OrderedDict([('linear.weight', tensor([[ 0.3975, -0.2836]])), ('linear.bias',
tensor([0.2074]))])
OrderedDict([('linear.weight', tensor([[ 0.3975, -0.2836]])), ('linear.bias',
tensor([0.2074]))])
```

In [54]:
```
checkpoint = {
    "model_state": model.state_dict(),
    "optim_state": optimizer.state_dict()
}
torch.save(checkpoint, 'checkpoint.pth')
# ---
model = Model()
optimizer = torch.optim.SGD(model.parameters(), lr=0.01)
checkpoint = torch.load('checkpoint.pth')
model.load_state_dict(checkpoint['model_state'])
optimizer.load_state_dict(checkpoint['optim_state'])
optimizer
```

Out[54]:
```
SGD (
Parameter Group 0
    dampening: 0
    lr: 0.1
    momentum: 0
    nesterov: False
    weight_decay: 0
)
```

Dataset & DataLoader

In [55]:
```
class MyDataset(Dataset):
    def __init__(self):
        super().__init__()
        self.X = torch.tensor([[2, 1], [1, 2]], dtype=torch.float32)
        self.Y = torch.tensor([0, 1], dtype=torch.float32).view(-1, 1)

    def __getitem__(self, idx):
        return self.X[idx], self.Y[idx]

    def __len__(self):
        return len(self.X)

ds = MyDataset()
```

In [56]:
```
len(ds), ds[0]
```

Out[56]:
```
(2, (tensor([2., 1.]), tensor([0.])))
```

In [57]:
```
dl = DataLoader(ds, batch_size=2, shuffle=True)
next(iter(dl))
```

Out[57]:
```
[tensor([[2., 1.],
         [1., 2.]]),
 tensor([[0.],
         [1.]])]
```

In [58]:
```
for epoch in range(50):
    for X, Y in dl:
        X = X.to(device)
        Y = Y.to(device)
        Y_pred = model(X)
        j = loss(Y_pred, Y)
        optimizer.zero_grad()
        j.backward()
        optimizer.step()

    if epoch % 10 == 9:
        print(f'epoch {epoch+1}: loss = {j.item():.8f}')
```

```
epoch 10: loss = 0.74077058
epoch 20: loss = 0.61800981
epoch 30: loss = 0.52327919
epoch 40: loss = 0.44903362
epoch 50: loss = 0.39019936
```

In [59]:
```
# ds = torchvision.datasets.ImageFolder(path)
```

Torchvision Transforms

In [60]:
```
transform = transforms.Compose([transforms.RandomHorizontalFlip(),
                                transforms.ToTensor(),
                                transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5
```

In [61]:
```
train_dataset = torchvision.datasets.CIFAR10(
    root='~/pytorch_datasets', download=True, train=True, transform=transform)
```

```
Files already downloaded and verified
```

In [62]:
```
train_dataset[0][0].shape  # NOTE: PyTorch uses Channel-FIRST (TF uses Channel-
```

Out[62]:
```
torch.Size([3, 32, 32])
```

Transfer Learning

In [63]:
```
model = torchvision.models.resnet18(pretrained=True)
model.fc = nn.Linear(model.fc.in_features, 10)
for param in model.parameters():
    param.requires_grad = False
```

In [ ]:
```

```