### QUICKSTART

This section runs through the API for common tasks in machine learning. Refer to the links in each section to dive deeper.

Working with data

PyTorch has two [primitives to work with data](https://pytorch.org/docs/stable/data.html): torch.utils.data.DataLoader and torch.utils.data.Dataset. Dataset stores the samples and their corresponding labels, and DataLoader wraps an iterable around the Dataset.

**import** torch

**from** torch **import** **nn**

**from** torch.utils.data **import** **DataLoader**

**from** torchvision **import** **datasets**

**from** torchvision.transforms **import** **ToTensor,** **Lambda,** **Compose**

**import** matplotlib.pyplot **as** plt

PyTorch offers domain-specific libraries such as [TorchText](https://pytorch.org/text/stable/index.html), [TorchVision](https://pytorch.org/vision/stable/index.html), and [TorchAudio](https://pytorch.org/audio/stable/index.html), all of which include datasets. For this tutorial, we will be using a TorchVision dataset.

The torchvision.datasets module contains Dataset objects for many real-world vision data like CIFAR, COCO ([full list here](https://pytorch.org/vision/stable/datasets.html)). In this tutorial, we use the FashionMNIST dataset. Every TorchVision Dataset includes two arguments: transform and target\_transform to modify the samples and labels respectively.

*# Download training data from open datasets.*

**training\_data** **=** **datasets.FashionMNIST(**

**root=**"data"**,**

**train=True,**

**download=True,**

**transform=ToTensor(),**

**)**

*# Download test data from open datasets.*

**test\_data** **=** **datasets.FashionMNIST(**

**root=**"data"**,**

**train=False,**

**download=True,**

**transform=ToTensor(),**

**)**

Out:

Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/train-images-idx3-ubyte.gz

Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/train-images-idx3-ubyte.gz to data/FashionMNIST/raw/train-images-idx3-ubyte.gz

Extracting data/FashionMNIST/raw/train-images-idx3-ubyte.gz to data/FashionMNIST/raw

Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/train-labels-idx1-ubyte.gz

Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/train-labels-idx1-ubyte.gz to data/FashionMNIST/raw/train-labels-idx1-ubyte.gz

Extracting data/FashionMNIST/raw/train-labels-idx1-ubyte.gz to data/FashionMNIST/raw

Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/t10k-images-idx3-ubyte.gz

Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/t10k-images-idx3-ubyte.gz to data/FashionMNIST/raw/t10k-images-idx3-ubyte.gz

Extracting data/FashionMNIST/raw/t10k-images-idx3-ubyte.gz to data/FashionMNIST/raw

Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/t10k-labels-idx1-ubyte.gz

Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/t10k-labels-idx1-ubyte.gz to data/FashionMNIST/raw/t10k-labels-idx1-ubyte.gz

Extracting data/FashionMNIST/raw/t10k-labels-idx1-ubyte.gz to data/FashionMNIST/raw

We pass the Dataset as an argument to DataLoader. This wraps an iterable over our dataset, and supports automatic batching, sampling, shuffling and multiprocess data loading. Here we define a batch size of 64, i.e. each element in the dataloader iterable will return a batch of 64 features and labels.

**batch\_size** **=** 64

*# Create data loaders.*

**train\_dataloader** **=** **DataLoader(training\_data,** **batch\_size=batch\_size)**

**test\_dataloader** **=** **DataLoader(test\_data,** **batch\_size=batch\_size)**

**for** **X,** **y** **in** **test\_dataloader:**

print**(**"Shape of X [N, C, H, W]: "**,** **X.shape)**

print**(**"Shape of y: "**,** **y.shape,** **y.dtype)**

**break**

Out:

Shape of X [N, C, H, W]: torch.Size([64, 1, 28, 28])

Shape of y: torch.Size([64]) torch.int64

Read more about [loading data in PyTorch](https://pytorch.org/tutorials/beginner/basics/data_tutorial.html).

Creating Models

To define a neural network in PyTorch, we create a class that inherits from [nn.Module](https://pytorch.org/docs/stable/generated/torch.nn.Module.html). We define the layers of the network in the \_\_init\_\_ function and specify how data will pass through the network in the forward function. To accelerate operations in the neural network, we move it to the GPU if available.

*# Get cpu or gpu device for training.*

**device** **=** "cuda" **if** **torch.cuda.is\_available()** **else** "cpu"

print**(**f"Using {**device**} device"**)**

*# Define model*

**class** **NeuralNetwork(nn.Module):**

**def** \_\_init\_\_**(**self**):**

super**(NeuralNetwork,** self**).**\_\_init\_\_**()**

self**.flatten** **=** **nn.Flatten()**

self**.linear\_relu\_stack** **=** **nn.Sequential(**

**nn.Linear(**28**\***28**,** 512**),**

**nn.ReLU(),**

**nn.Linear(**512**,** 512**),**

**nn.ReLU(),**

**nn.Linear(**512**,** 10**)**

**)**

**def** **forward(**self**,** **x):**

**x** **=** self**.flatten(x)**

**logits** **=** self**.linear\_relu\_stack(x)**

**return** **logits**

**model** **=** **NeuralNetwork().to(device)**

print**(model)**

Out:

Using cuda device

NeuralNetwork(

(flatten): Flatten(start\_dim=1, end\_dim=-1)

(linear\_relu\_stack): Sequential(

(0): Linear(in\_features=784, out\_features=512, bias=True)

(1): ReLU()

(2): Linear(in\_features=512, out\_features=512, bias=True)

(3): ReLU()

(4): Linear(in\_features=512, out\_features=10, bias=True)

)

)

Read more about [building neural networks in PyTorch](https://pytorch.org/tutorials/beginner/basics/buildmodel_tutorial.html).

## Optimizing the Model Parameters

To train a model, we need a [loss function](https://pytorch.org/docs/stable/nn.html#loss-functions) and an [optimizer](https://pytorch.org/docs/stable/optim.html).

**loss\_fn** **=** **nn.CrossEntropyLoss()**

**optimizer** **=** **torch.optim.SGD(model.parameters(),** **lr=**1e-3**)**

In a single training loop, the model makes predictions on the training dataset (fed to it in batches), and backpropagates the prediction error to adjust the model’s parameters.

**def** **train(dataloader,** **model,** **loss\_fn,** **optimizer):**

**size** **=** len**(dataloader.dataset)**

**model.train()**

**for** **batch,** **(X,** **y)** **in** enumerate**(dataloader):**

**X,** **y** **=** **X.to(device),** **y.to(device)**

*# Compute prediction error*

**pred** **=** **model(X)**

**loss** **=** **loss\_fn(pred,** **y)**

*# Backpropagation*

**optimizer.zero\_grad()**

**loss.backward()**

**optimizer.step()**

**if** **batch** **%** 100 **==** 0**:**

**loss,** **current** **=** **loss.item(),** **batch** **\*** len**(X)**

print**(**f"loss: {**loss**:>7f} [{**current**:>5d}/{**size**:>5d}]"**)**

We also check the model’s performance against the test dataset to ensure it is learning.

**def** **test(dataloader,** **model,** **loss\_fn):**

**size** **=** len**(dataloader.dataset)**

**num\_batches** **=** len**(dataloader)**

**model.eval()**

**test\_loss,** **correct** **=** 0**,** 0

**with** **torch.no\_grad():**

**for** **X,** **y** **in** **dataloader:**

**X,** **y** **=** **X.to(device),** **y.to(device)**

**pred** **=** **model(X)**

**test\_loss** **+=** **loss\_fn(pred,** **y).item()**

**correct** **+=** **(pred.argmax(**1**)** **==** **y).type(torch.float).sum().item()**

**test\_loss** **/=** **num\_batches**

**correct** **/=** **size**

print**(**f"Test Error: \n Accuracy: {**(**100**\*correct)**:>0.1f}%, Avg loss: {**test\_loss**:>8f} \n"**)**

The training process is conducted over several iterations (epochs). During each epoch, the model learns parameters to make better predictions. We print the model’s accuracy and loss at each epoch; we’d like to see the accuracy increase and the loss decrease with every epoch.

**epochs** **=** 5

**for** **t** **in** range**(epochs):**

print**(**f"Epoch {**t+**1}\n-------------------------------"**)**

**train(train\_dataloader,** **model,** **loss\_fn,** **optimizer)**

**test(test\_dataloader,** **model,** **loss\_fn)**

print**(**"Done!"**)**

Out:

Epoch 1

-------------------------------

loss: 2.309376 [ 0/60000]

loss: 2.295801 [ 6400/60000]

loss: 2.272989 [12800/60000]

loss: 2.263796 [19200/60000]

loss: 2.252100 [25600/60000]

loss: 2.225338 [32000/60000]

loss: 2.232905 [38400/60000]

loss: 2.204130 [44800/60000]

loss: 2.200894 [51200/60000]

loss: 2.172664 [57600/60000]

Test Error:

Accuracy: 42.0%, Avg loss: 2.162939

Epoch 2

-------------------------------

loss: 2.172714 [ 0/60000]

loss: 2.163641 [ 6400/60000]

loss: 2.107901 [12800/60000]

loss: 2.124609 [19200/60000]

loss: 2.083108 [25600/60000]

loss: 2.017031 [32000/60000]

loss: 2.049765 [38400/60000]

loss: 1.972885 [44800/60000]

loss: 1.972378 [51200/60000]

loss: 1.919778 [57600/60000]

Test Error:

Accuracy: 52.8%, Avg loss: 1.907939

Epoch 3

-------------------------------

loss: 1.929562 [ 0/60000]

loss: 1.906155 [ 6400/60000]

loss: 1.791957 [12800/60000]

loss: 1.841964 [19200/60000]

loss: 1.738918 [25600/60000]

loss: 1.679015 [32000/60000]

loss: 1.714024 [38400/60000]

loss: 1.608983 [44800/60000]

loss: 1.634542 [51200/60000]

loss: 1.543917 [57600/60000]

Test Error:

Accuracy: 58.2%, Avg loss: 1.548668

Epoch 4

-------------------------------

loss: 1.605491 [ 0/60000]

loss: 1.572702 [ 6400/60000]

loss: 1.425616 [12800/60000]

loss: 1.503090 [19200/60000]

loss: 1.390305 [25600/60000]

loss: 1.377193 [32000/60000]

loss: 1.397487 [38400/60000]

loss: 1.316429 [44800/60000]

loss: 1.354412 [51200/60000]

loss: 1.258275 [57600/60000]

Test Error:

Accuracy: 61.8%, Avg loss: 1.278839

Epoch 5

-------------------------------

loss: 1.352444 [ 0/60000]

loss: 1.330832 [ 6400/60000]

loss: 1.170559 [12800/60000]

loss: 1.276487 [19200/60000]

loss: 1.159532 [25600/60000]

loss: 1.180266 [32000/60000]

loss: 1.200387 [38400/60000]

loss: 1.132442 [44800/60000]

loss: 1.176094 [51200/60000]

loss: 1.092559 [57600/60000]

Test Error:

Accuracy: 63.8%, Avg loss: 1.109808

Done!

Read more about [Training your model](https://pytorch.org/tutorials/beginner/basics/optimization_tutorial.html).

## Saving Models

A common way to save a model is to serialize the internal state dictionary (containing the model parameters).

**torch.save(model.state\_dict(),** "model.pth"**)**

print**(**"Saved PyTorch Model State to model.pth"**)**

Out:

Saved PyTorch Model State to model.pth

## Loading Models

The process for loading a model includes re-creating the model structure and loading the state dictionary into it.

**model** **=** **NeuralNetwork()**

**model.load\_state\_dict(torch.load(**"model.pth"**))**

This model can now be used to make predictions.

**classes** **=** **[**

"T-shirt/top"**,**

"Trouser"**,**

"Pullover"**,**

"Dress"**,**

"Coat"**,**

"Sandal"**,**

"Shirt"**,**

"Sneaker"**,**

"Bag"**,**

"Ankle boot"**,**

**]**

**model.eval()**

**x,** **y** **=** **test\_data[**0**][**0**],** **test\_data[**0**][**1**]**

**with** **torch.no\_grad():**

**pred** **=** **model(x)**

**predicted,** **actual** **=** **classes[pred[**0**].argmax(**0**)],** **classes[y]**

print**(**f'Predicted: "{**predicted**}", Actual: "{**actual**}"'**)**

Out:

Predicted: "Ankle boot", Actual: "Ankle boot"

Read more about [Saving & Loading your model](https://pytorch.org/tutorials/beginner/basics/saveloadrun_tutorial.html).

**Total running time of the script:** ( 0 minutes 54.548 seconds)