

Quickstart

Created On: Feb 09, 2021 | Last Updated: Jan 24, 2025 | Last Verified: Not Verified

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](#): `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
```

PyTorch offers domain-specific libraries such as [TorchText](#), [TorchVision](#), and [TorchAudio](#), 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](#)). 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:

```
0%|          | 0.00/26.4M [00:00<?, ?B/s]
0%|          | 65.5k/26.4M [00:00<01:13, 360kB/s]
1%|          | 229k/26.4M [00:00<00:38, 677kB/s]
3%|3        | 918k/26.4M [00:00<00:10, 2.55MB/s]
7%|7        | 1.93M/26.4M [00:00<00:06, 4.07MB/s]
21%|##1     | 5.64M/26.4M [00:00<00:01, 12.8MB/s]
36%|###6    | 9.60M/26.4M [00:00<00:00, 20.2MB/s]
50%|####9   | 13.1M/26.4M [00:01<00:00, 20.6MB/s]
65%|#####5 | 17.2M/26.4M [00:01<00:00, 25.9MB/s]
81%|##### | 21.4M/26.4M [00:01<00:00, 29.9MB/s]
96%|#####5 | 25.3M/26.4M [00:01<00:00, 27.1MB/s]
100%|##### | 26.4M/26.4M [00:01<00:00, 19.1MB/s]

0%|          | 0.00/29.5k [00:00<?, ?B/s]
100%|##### | 29.5k/29.5k [00:00<00:00, 327kB/s]

0%|          | 0.00/4.42M [00:00<?, ?B/s]
1%|1        | 65.5k/4.42M [00:00<00:12, 363kB/s]
5%|5        | 229k/4.42M [00:00<00:06, 692kB/s]
```

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(f"Shape of X [N, C, H, W]: {X.shape}")
    print(f"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](#).

Creating Models

To define a neural network in PyTorch, we create a class that inherits from `nn.Module`. 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 `accelerator` such as CUDA, MPS, MTIA, or XPU. If the current accelerator is available, we will use it. Otherwise, we use the CPU.

```

device = torch.accelerator.current_accelerator().type if torch.accelerator.is_available() else "cpu"
print(f"Using {device} device")

# Define model
class NeuralNetwork(nn.Module):
    def __init__(self):
        super().__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](#).

Optimizing the Model Parameters

To train a model, we need a `loss function` and an `optimizer`.

```

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
        loss.backward()
        optimizer.step()
        optimizer.zero_grad()

    if batch % 100 == 0:
        loss, current = loss.item(), (batch + 1) * 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)/size}>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: 2.303494    [  64/60000]
loss: 2.294637    [ 6464/60000]
loss: 2.277102    [12864/60000]
loss: 2.269977    [19264/60000]
loss: 2.254235    [25664/60000]
loss: 2.237146    [32064/60000]
loss: 2.231055    [38464/60000]
loss: 2.205037    [44864/60000]
loss: 2.203240    [51264/60000]
loss: 2.170889    [57664/60000]
Test Error:
Accuracy: 53.9%, Avg loss: 2.168588

Epoch 2
-----
loss: 2.177787    [  64/60000]
loss: 2.168083    [ 6464/60000]
```

Read more about [Training your model](#).

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().to(device)
model.load_state_dict(torch.load("model.pth", weights_only=True))
```

Out: <All keys matched successfully>

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():
    x = x.to(device)
    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](#).

Total running time of the script: (1 minutes 2.544 seconds)

[< Previous](#)

[Next >](#)

Rate this Tutorial ☆☆☆☆

© Copyright 2024, PyTorch.

Built with [Sphinx](#) using a [theme](#) provided by [Read the Docs](#).

Docs

Access comprehensive developer documentation for PyTorch

[View Docs](#)

Tutorials

Get in-depth tutorials for beginners and advanced developers

[View Tutorials](#)

Resources

Find development resources and get your questions answered

[View Resources](#)

PyTorch

Get Started

Features

Ecosystem

Blog

Contributing

Resources

Tutorials

Docs

Discuss

Github Issues

Brand Guidelines

Stay up to date

Facebook

Twitter

YouTube

LinkedIn

PyTorch Podcasts

Spotify

Apple

Google

Amazon

Terms | Privacy

© Copyright The Linux Foundation. The PyTorch Foundation is a project of The Linux Foundation. For web site terms of use, trademark policy and other policies applicable to The PyTorch Foundation please see www.linuxfoundation.org/policies/. The PyTorch Foundation supports the PyTorch open source project, which has been established as PyTorch Project a Series of LF Projects, LLC. For policies applicable to the PyTorch Project a Series of LF Projects, LLC, please see www.lfprojects.org/policies/.