Machine Learning
PyTorch Tutorial

TA：張恆瑞 (Heng-Jui Chang)
2021.03.05

Outline

- Prerequisites
- What is PyTorch?
- PyTorch v.s. TensorFlow
- Overview of the DNN Training Procedure
- Tensor
- How to Calculate Gradient?
- Dataset & Dataloader
- torch.nn
- torch.optim
- Neural Network Training/Evaluation
- Saving/Loading a Neural Network
- More About PyTorch

Prerequisites

- We assume you are already familiar with...
  - **Python3**
    - `if-else, loop, function, file IO, class, ...`
    - refs: link1, link2, link3
  - **NumPy**
    - array & array operations
    - ref: link

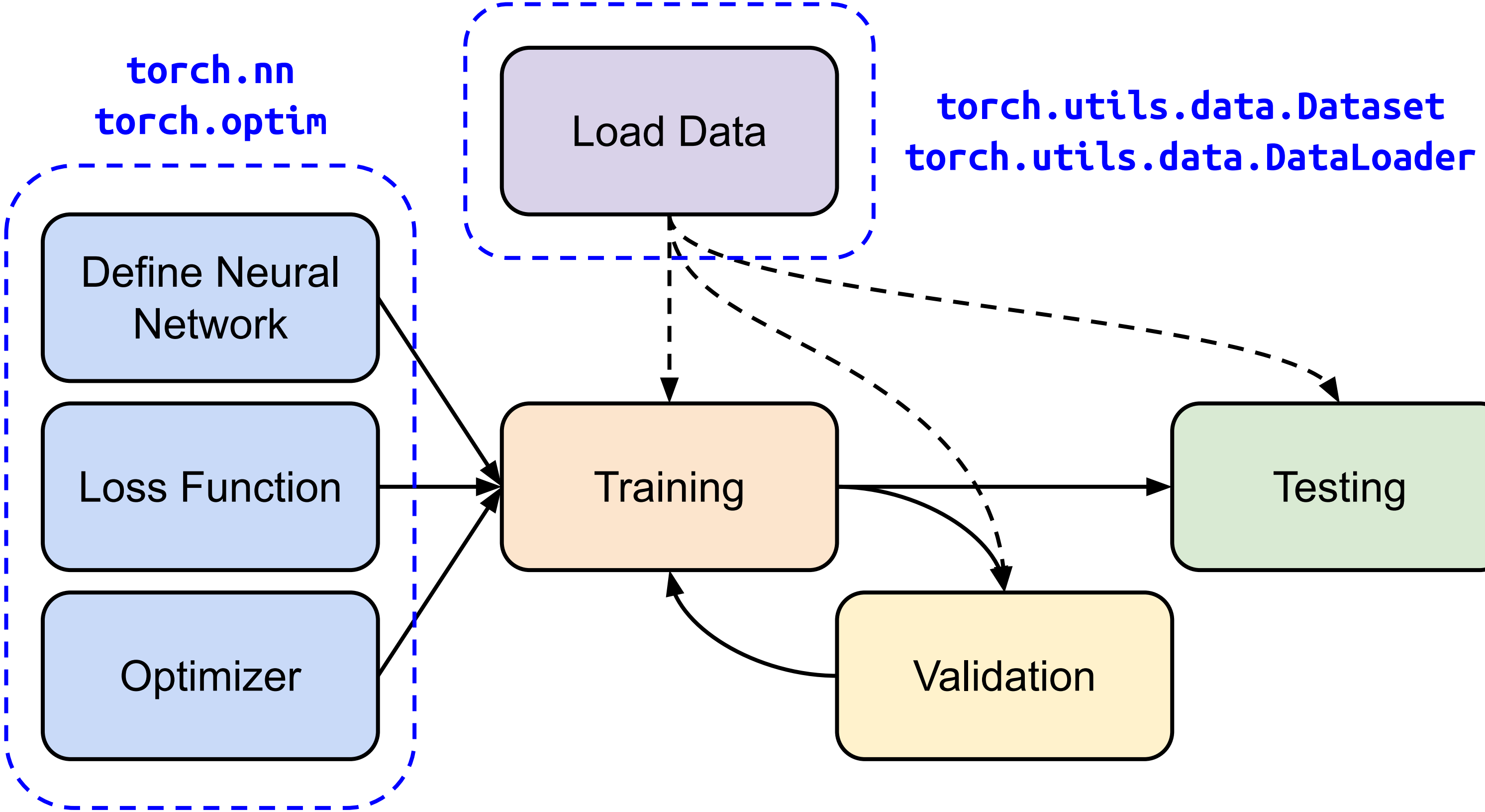
What is PyTorch?

- An open source **machine learning framework**.
- A Python package that provides two high-level features:
  - **Tensor** computation (like NumPy) with strong **GPU acceleration**
  - Deep neural networks built on a **tape-based autograd** system

PyTorch v.s. TensorFlow

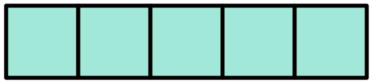
	PyTorch	TensorFlow
Developer	Facebook AI	Google Brain
Interface	Python & C++	Python, C++, JavaScript, Swift
Debug	Easier	Difficult (easier in 2.0)
Application	Research	Production

Overview of the DNN Training Procedure

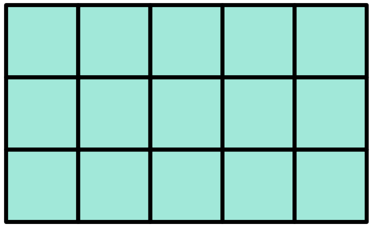


Tensor

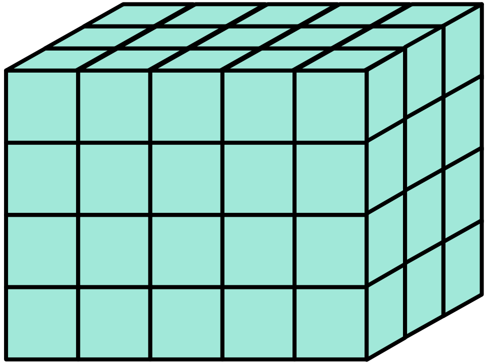
- High-dimensional matrix (array)



1-D tensor



2-D tensor



3-D tensor

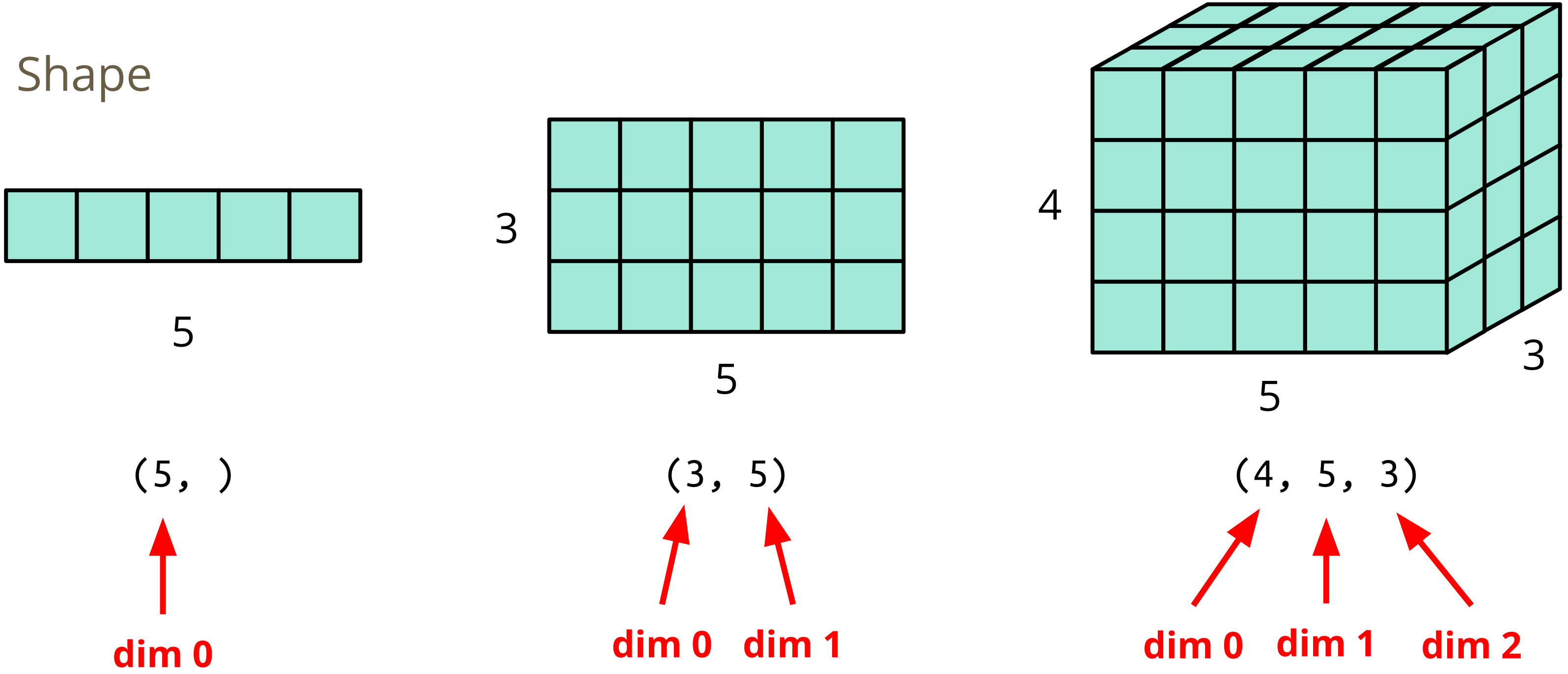
Tensor -- Data Type

Data type	dtype	tensor
32-bit floating point	`torch.float`	`torch.FloatTensor`
64-bit integer (signed)	`torch.long`	`torch.LongTensor`

ref: https://pytorch.org/docs/stable/tensors.html

Tensor -- Shape of Tensors

- Shape



Note: **dim** in PyTorch == **axis** in NumPy

Tensor -- Constructor

- From list / NumPy array

```
x = torch.tensor([[1, -1], [-1, 1]])
```

```
x = torch.from_numpy(np.array([[1, -1], [-1, 1]]))
```

```
tensor([[1., -1.],
        [-1., 1.]])
```

- Zero tensor

```
x = torch.zeros([2, 2])
```

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

- Unit tensor

```
x = torch.ones([1, 2, 5])
```

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

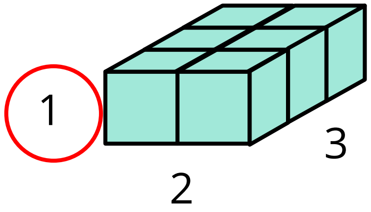
shape

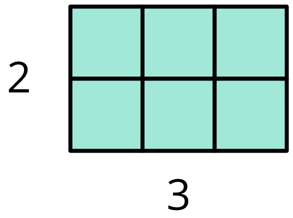
Tensor -- Operators

- **Squeeze**: remove the specified dimension with length = 1

```
>>> x = torch.zeros([1, 2, 3])
>>> x.shape
torch.Size([1, 2, 3])
>>> x = x.squeeze(0)
>>> x.shape
torch.Size([2, 3])
```

(dim = 0)



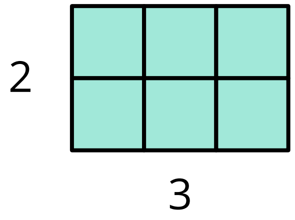


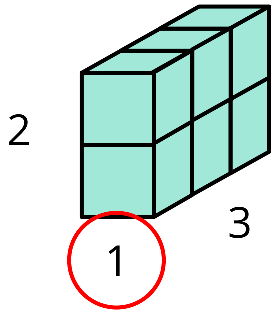
Tensor -- Operators

- **Unsqueeze**: expand a new dimension

```
>>> x = torch.zeros([2, 3])
>>> x.shape
torch.Size([2, 3])
>>> x = x.unsqueeze(1)
>>> x.shape
torch.Size([2, 1, 3])
```

(dim = 1)

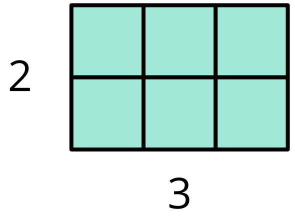


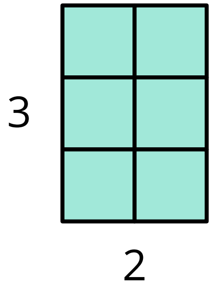


Tensor -- Operators

- **Transpose**: transpose two specified dimensions

```
>>> x = torch.zeros([2, 3])
>>> x.shape
torch.Size([2, 3])
>>> x = x.transpose(0, 1)
>>> x.shape
torch.Size([3, 2])
```

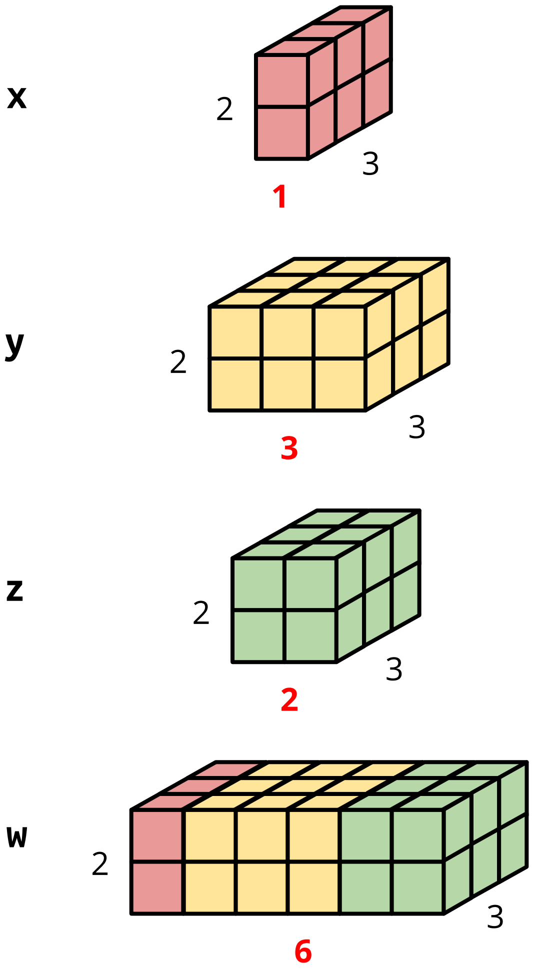




Tensor -- Operators

- **Cat**: concatenate multiple tensors

```
>>> x = torch.zeros([2, 1, 3])
>>> y = torch.zeros([2, 3, 3])
>>> z = torch.zeros([2, 2, 3])
>>> w = torch.cat([x, y, z], dim=1)
>>> w.shape
torch.Size([2, 6, 3])
```



Tensor -- Operators

- Addition

```
z = x + y
```

- Subtraction

```
z = x - y
```

- Power

```
y = x.pow(2)
```

Tensor -- Operators

- Summation

```
y = x.sum()
```

- Mean

```
y = x.mean()
```

more operators: https://pytorch.org/docs/stable/tensors.html

Tensor -- PyTorch v.s. NumPy

- Attributes

PyTorch	NumPy
x.shape	x.shape
x.dtype	x.dtype

ref: https://github.com/wkentaro/pytorch-for-numpy-users

Tensor -- PyTorch v.s. NumPy

- Shape manipulation

PyTorch	NumPy
`x.reshape / x.view`	`x.reshape`
`x.squeeze()`	`x.squeeze()`
`x.unsqueeze(1)`	`np.expand_dims(x, 1)`

ref: https://github.com/wkentaro/pytorch-for-numpy-users

Tensor -- Device

- Default: tensors & modules will be computed with **CPU**

- CPU

```
x = x.to('cpu')
```

- GPU

```
x = x.to('cuda')
```

Tensor -- Device (GPU)

- Check if your computer has NVIDIA GPU

```
torch.cuda.is_available()
```

- Multiple GPUs: specify `‘cuda:0’, ‘cuda:1’, ‘cuda:2’, ...`

- Why GPU?
    - Parallel computing
    - https://towardsdatascience.com/what-is-a-gpu-and-do-you-need-one-in-deep-learning-718b9597aa0d

How to Calculate Gradient?

1
```
>>> x = torch.tensor([[1., 0.], [-1., 1.]], requires_grad=True)
```
2
```
>>> z = x.pow(2).sum()
```
3
```
>>> z.backward()
```
4
```
>>> x.grad
tensor([[ 2.,  0.],
        [-2.,  2.]])
```

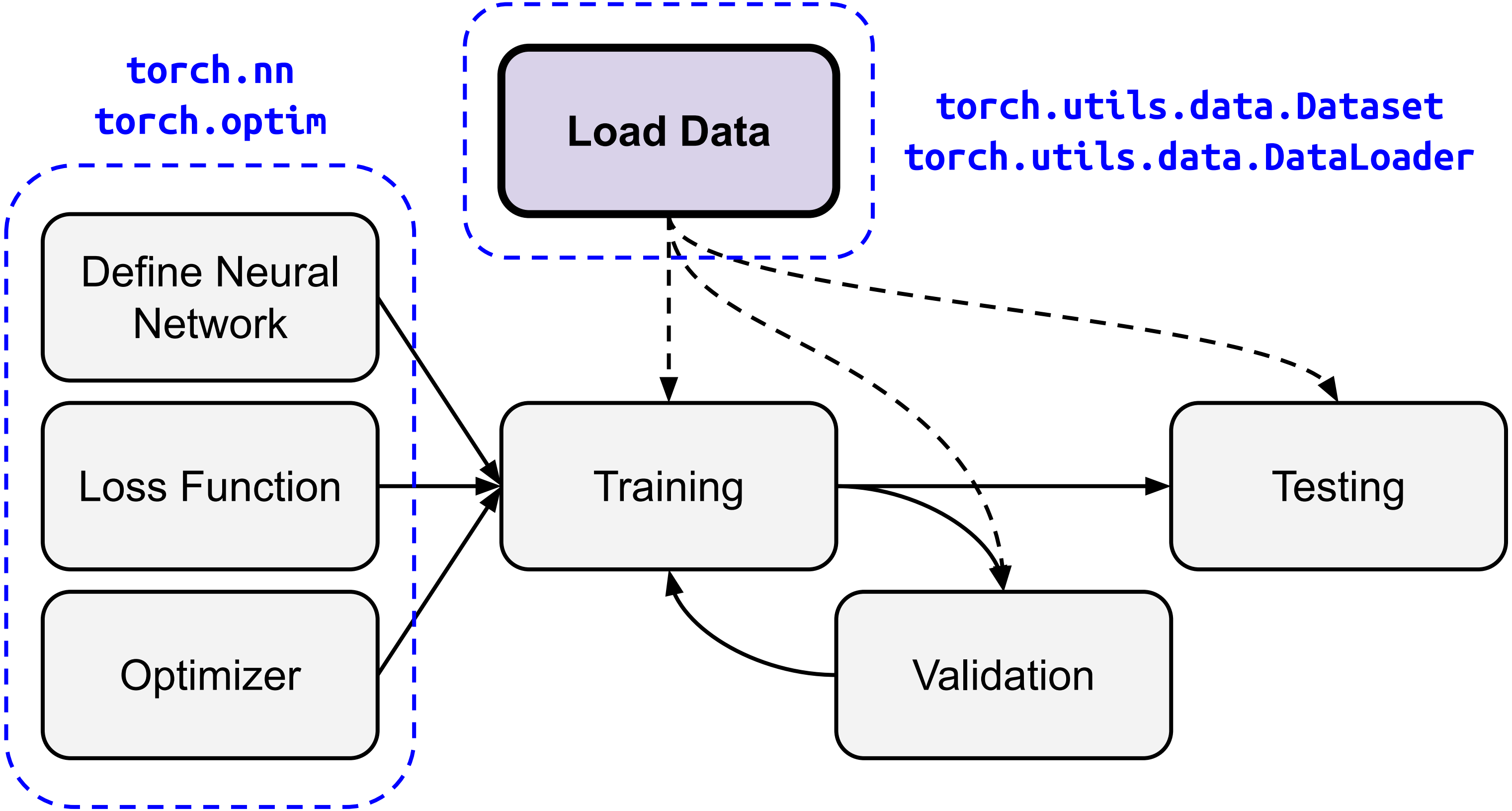
1
$$x = \begin{bmatrix} 1 & 0 \\ -1 & 1 \end{bmatrix}$$

2
$$z = \sum_i \sum_j x_{i,j}^2$$

3
$$\frac{\partial z}{\partial x_{i,j}} = 2x_{i,j}$$

4
$$\frac{\partial z}{\partial x} = \begin{bmatrix} 2 & 0 \\ -2 & 2 \end{bmatrix}$$

Overview of the DNN Training Procedure



Dataset & Dataloader

```
from torch.utils.data import Dataset, DataLoader

class MyDataset(Dataset):
    def __init__(self, file):
        self.data = ...

    def __getitem__(self, index):
        return self.data[index]

    def __len__(self):
        return len(self.data)
```

Read data & preprocess

Returns one sample at a time

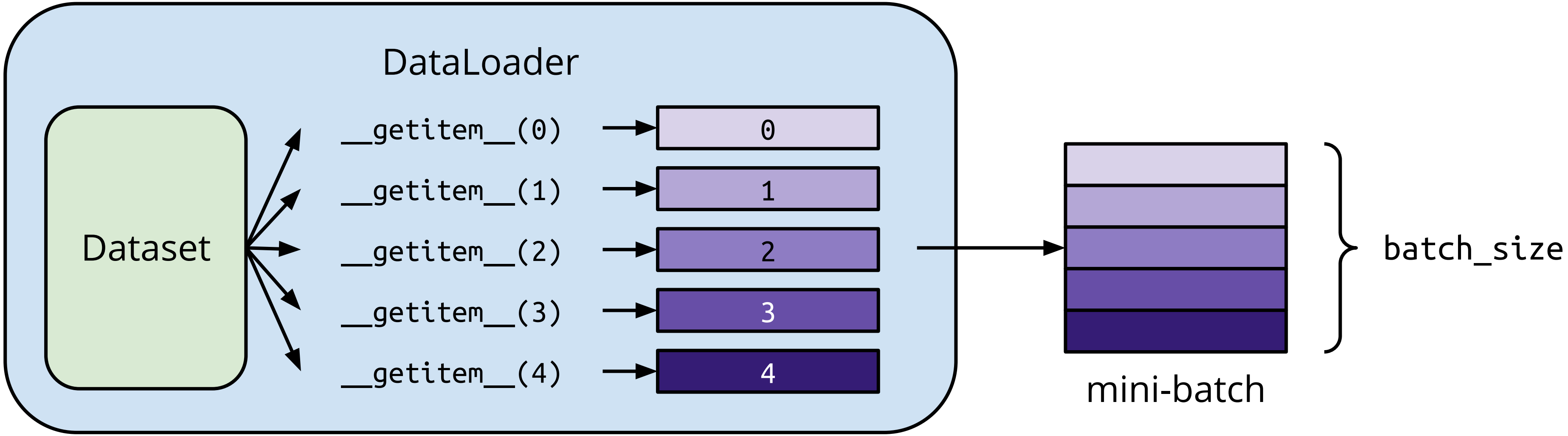
Returns the size of the dataset

Dataset & Dataloader

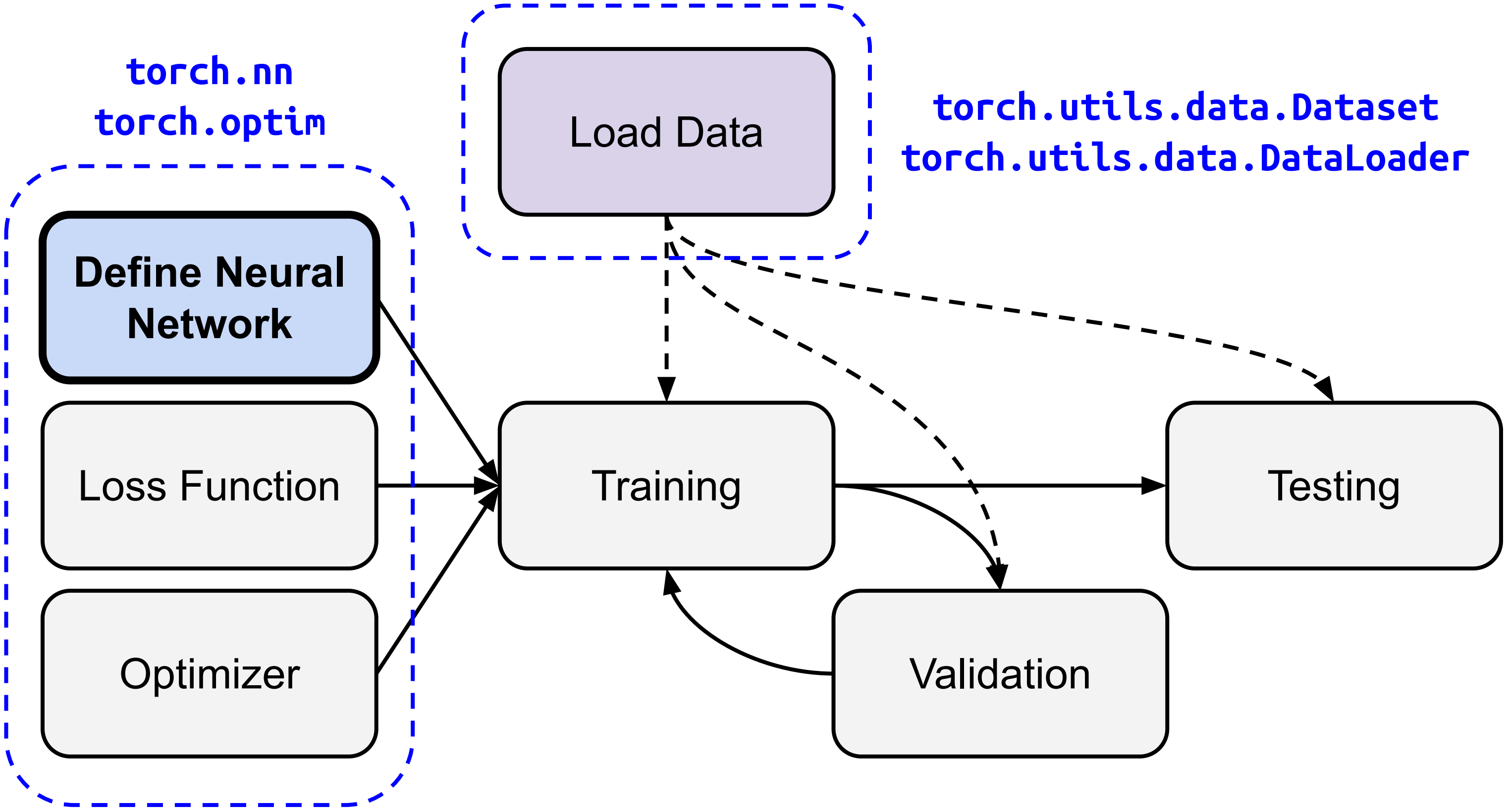
Training: True
Testing: False

```
dataset = MyDataset(file)

dataloader = DataLoader(dataset, batch_size, shuffle=True)
```



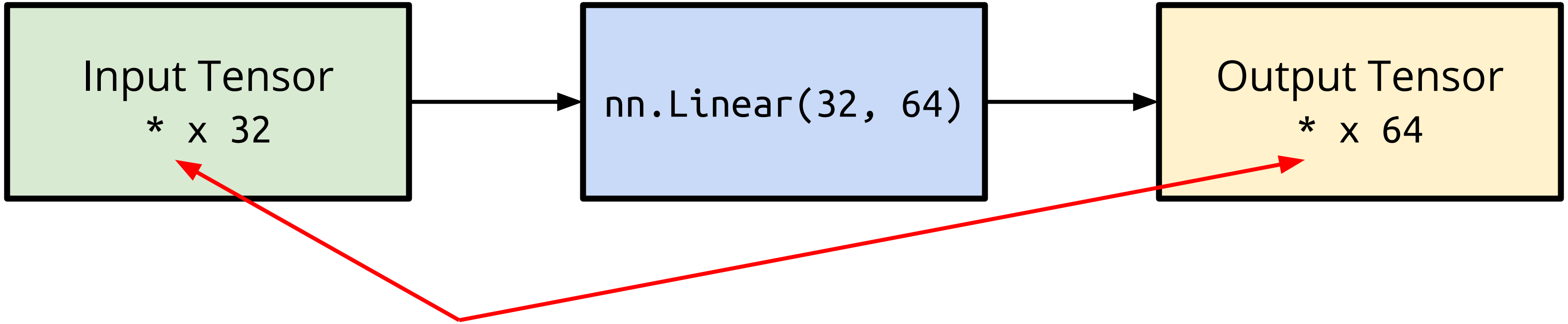
Overview of the DNN Training Procedure



torch.nn -- Neural Network Layers

- Linear Layer (**Fully-connected** Layer)

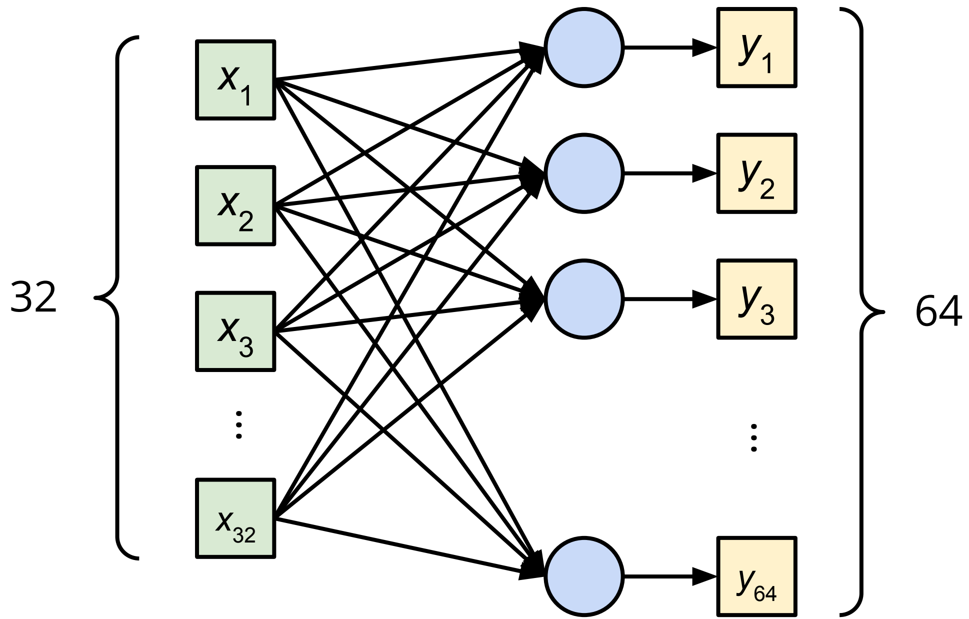
  `nn.Linear(in_features, out_features)`

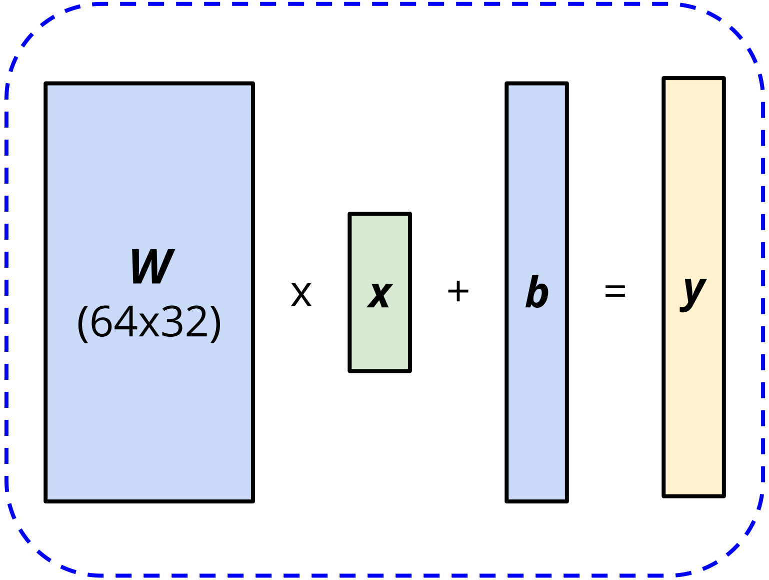


can be any shape but the last dimension must be 32
e.g. `(10, 32)`, `(10, 5, 32)`, `(1, 1, 3, 32)`, ...

torch.nn -- Neural Network Layers

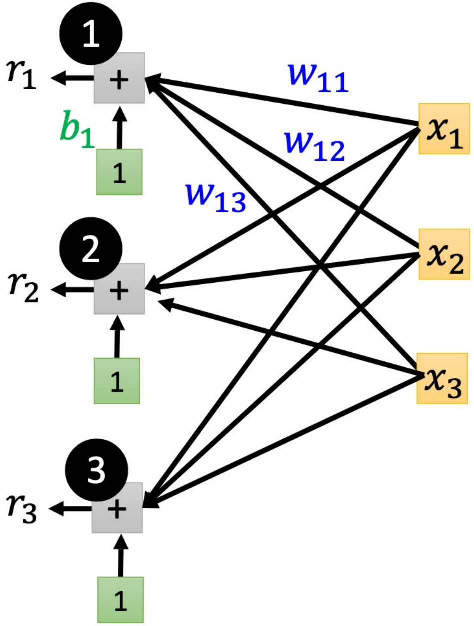
- Linear Layer (**Fully-connected** Layer)





torch.nn -- Neural Network Layers

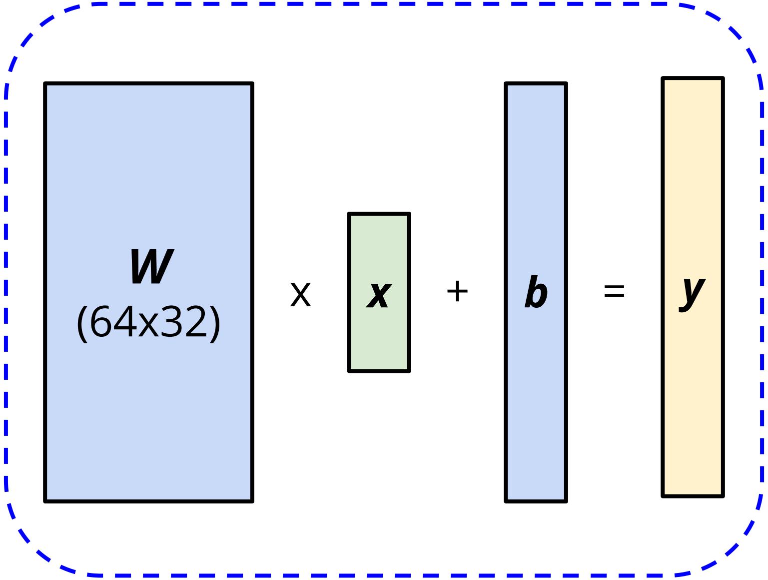
- Linear Layer (**Fully-connected** Layer)



torch.nn -- Neural Network Layers

- Linear Layer (**Fully-connected** Layer)

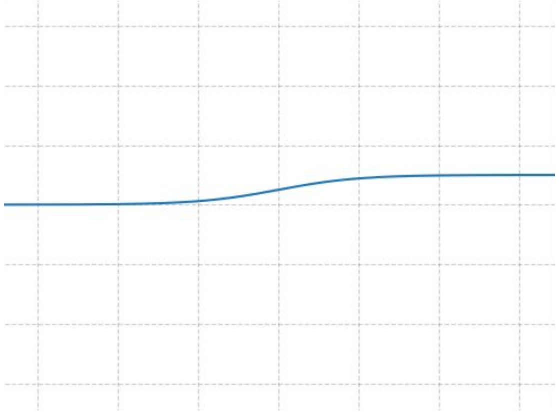
```
>>> layer = torch.nn.Linear(32, 64)
>>> layer.weight.shape
torch.Size([64, 32])
>>> layer.bias.shape
torch.Size([64])
```



torch.nn -- Activation Functions

- Sigmoid Activation

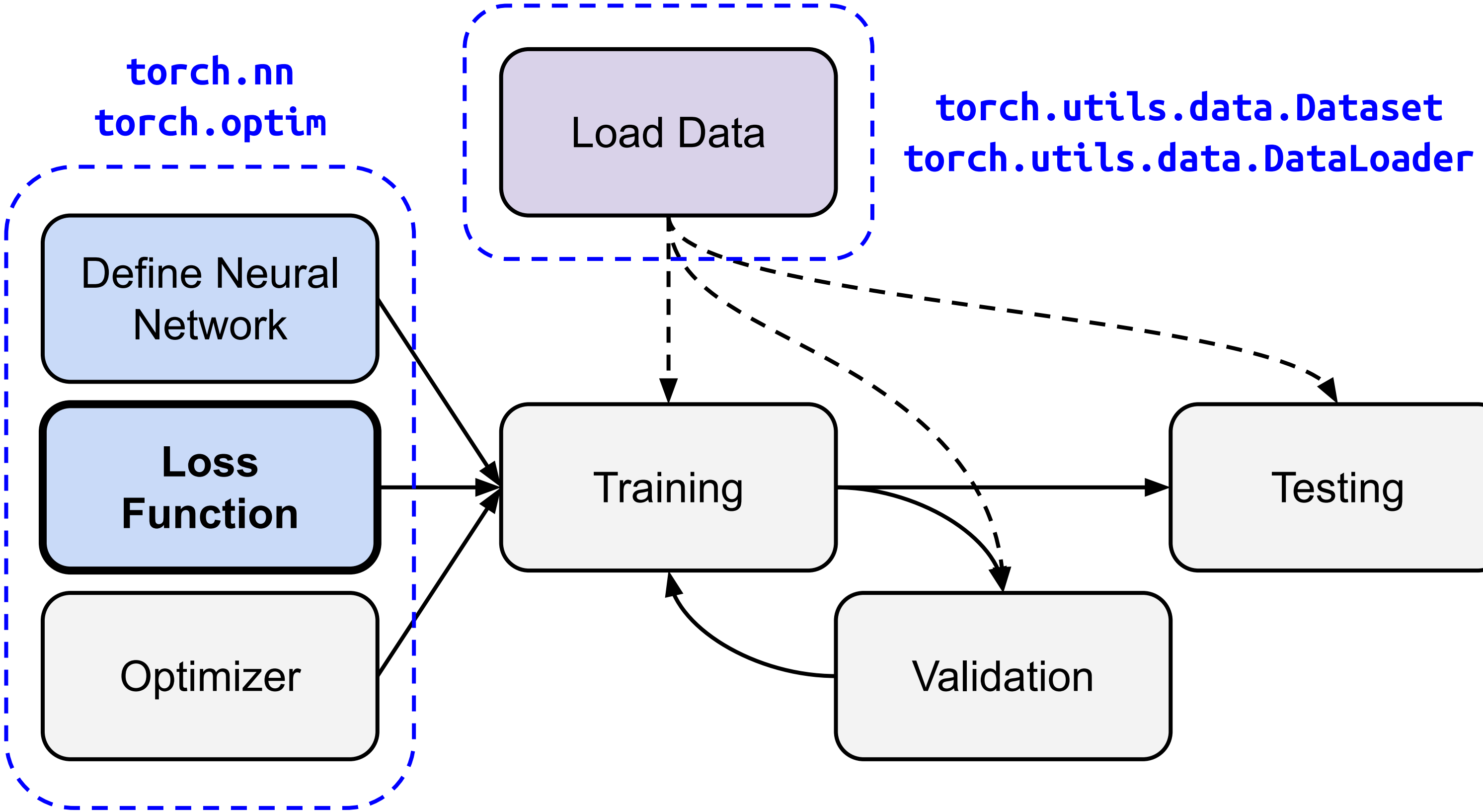
`nn.Sigmoid()`



- ReLU Activation

`nn.ReLU()`

Overview of the DNN Training Procedure



torch.nn -- Loss Functions

- Mean Squared Error (for linear regression)

```
nn.MSELoss()
```

- Cross Entropy (for classification)

```
nn.CrossEntropyLoss()
```

torch.nn -- Build your own neural network

```
import torch.nn as nn

class MyModel(nn.Module):
    def __init__(self):
        super(MyModel, self).__init__()
        self.net = nn.Sequential(
            nn.Linear(10, 32),
            nn.Sigmoid(),
            nn.Linear(32, 1)
        )

    def forward(self, x):
        return self.net(x)
```

Initialize your model & define layers

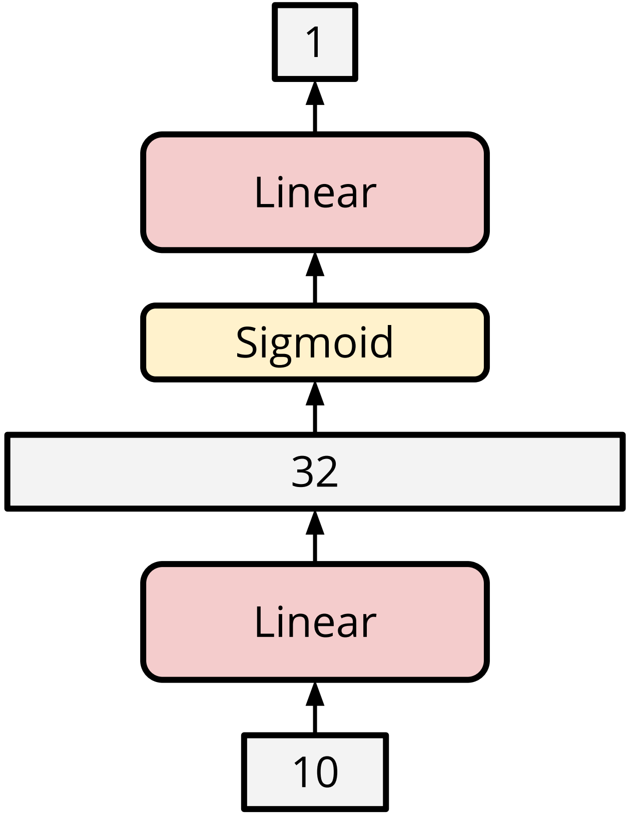
Compute output of your NN

torch.nn -- Build your own neural network

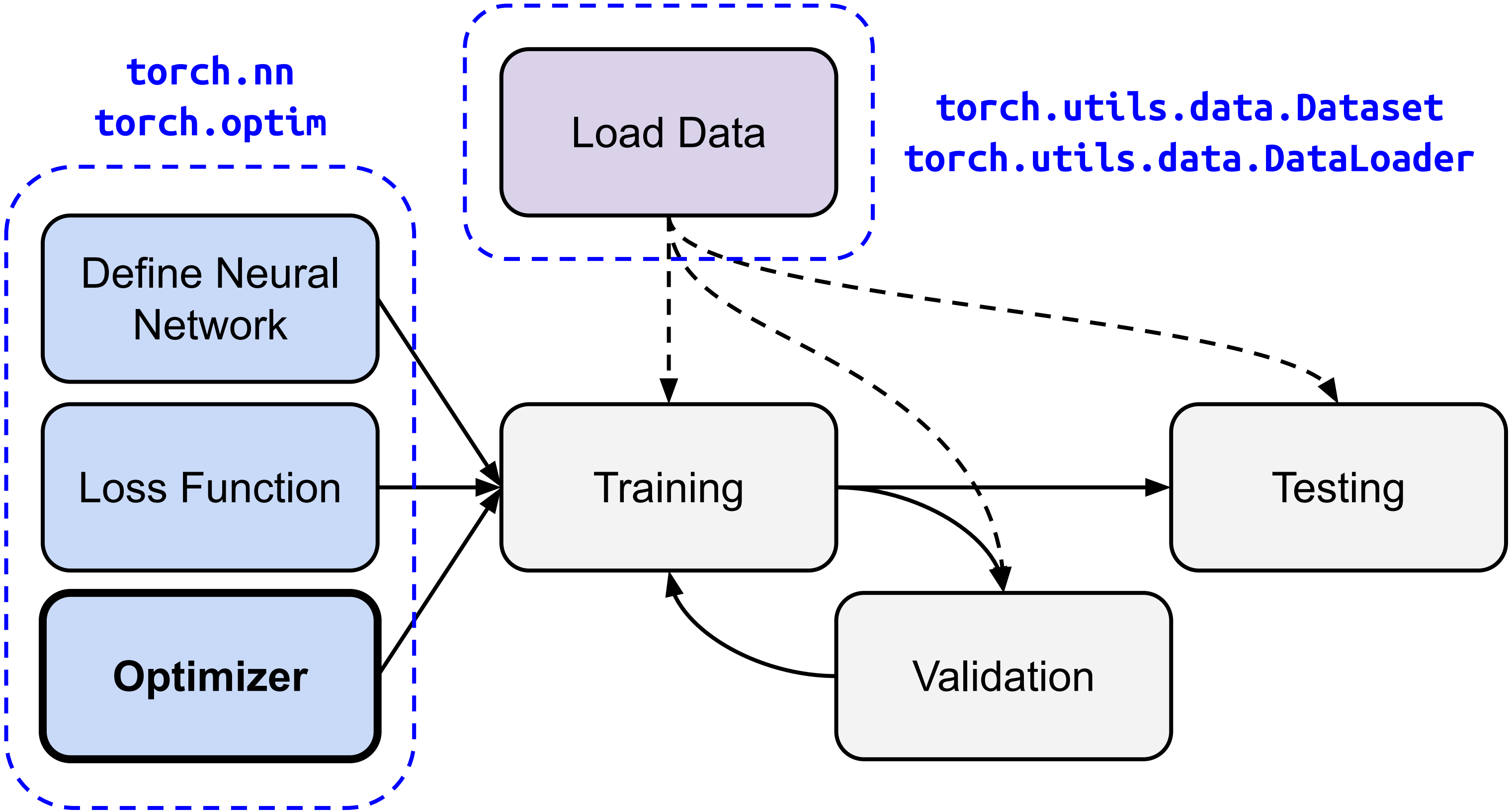
```
import torch.nn as nn

class MyModel(nn.Module):
    def __init__(self):
        super(MyModel, self).__init__()
        self.net = nn.Sequential(
            nn.Linear(10, 32),
            nn.Sigmoid(),
            nn.Linear(32, 1)
        )

    def forward(self, x):
        return self.net(x)
```



Overview of the DNN Training Procedure



torch.optim

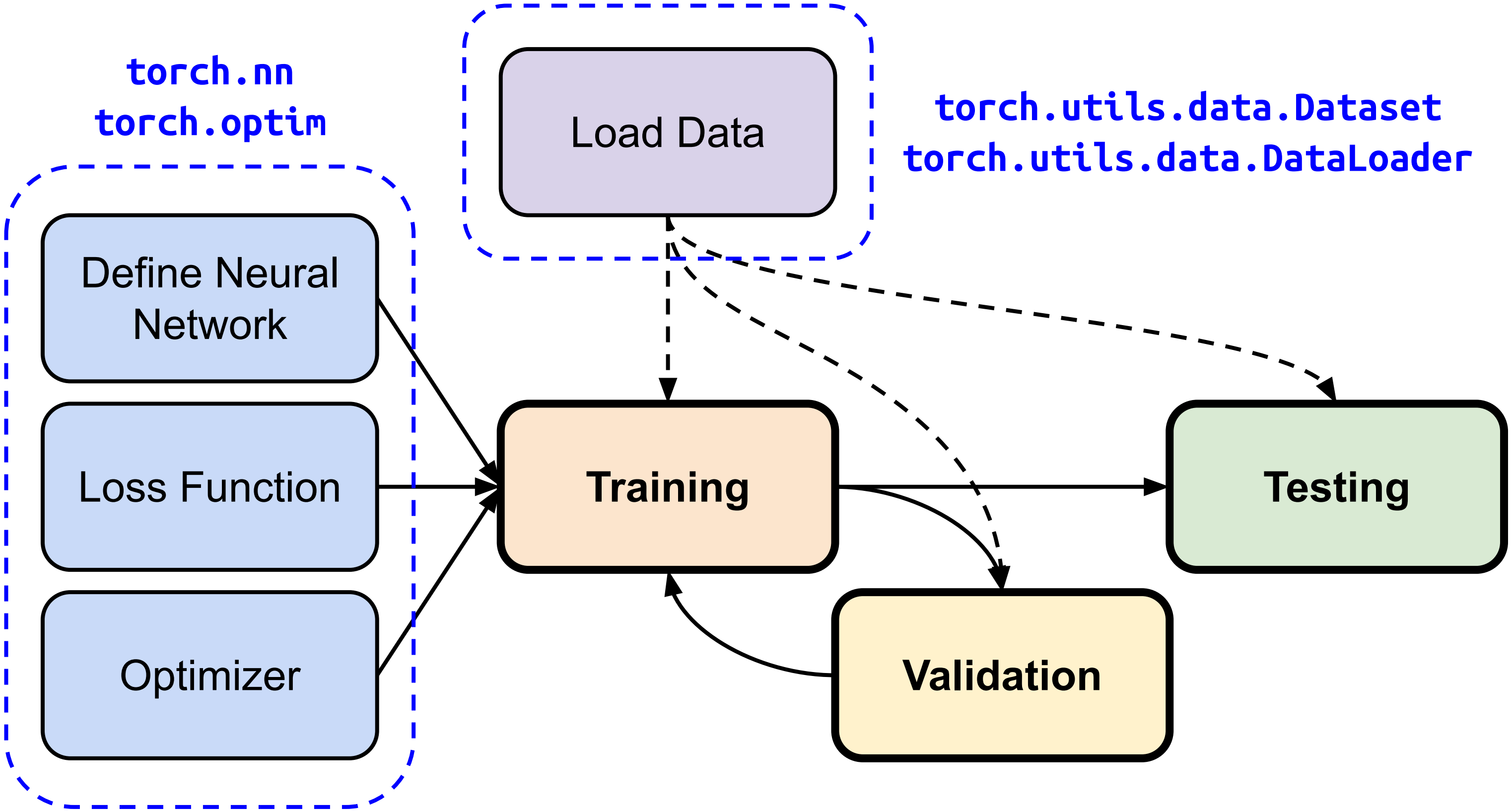
- **Optimization algorithms** for neural networks (gradient descent)

- Stochastic Gradient Descent (SGD)

```
torch.optim.SGD(params, lr, momentum = 0)
```

```
model.parameters()
```

Overview of the DNN Training Procedure



Neural Network Training

```
dataset = MyDataset(file)                                   read data via MyDataset
tr_set = DataLoader(dataset, 16, shuffle=True)              put dataset into Dataloader
model = MyModel().to(device)                                contruct model and move to device (cpu/cuda)
criterion = nn.MSELoss()                                    set loss function
optimizer = torch.optim.SGD(model.parameters(), 0.1)        set optimizer
```

Neural Network Training

```
for epoch in range(n_epochs):                    iterate n_epochs
    model.train()                                set model to train mode
    for x, y in tr_set:                          iterate through the dataloader
        optimizer.zero_grad()                    set gradient to zero
        x, y = x.to(device), y.to(device)        move data to device (cpu/cuda)
        pred = model(x)                          forward pass (compute output)
        loss = criterion(pred, y)                compute loss
        loss.backward()                          compute gradient (backpropagation)
        optimizer.step()                         update model with optimizer
```

Neural Network Evaluation (Validation Set)

```
model.eval()                                        set model to evaluation mode
total_loss = 0
for x, y in dv_set:                                 iterate through the dataloader
    x, y = x.to(device), y.to(device)               move data to device (cpu/cuda)
    with torch.no_grad():                           disable gradient calculation
        pred = model(x)                             forward pass (compute output)
        loss = criterion(pred, y)                   compute loss
    total_loss += loss.cpu().item() * len(x)        accumulate loss
    avg_loss = total_loss / len(dv_set.dataset)     compute averaged loss
```

Neural Network Evaluation (Testing Set)

```
model.eval()                          set model to evaluation mode
preds = []
for x in tt_set:                      iterate through the dataloader
    x = x.to(device)                  move data to device (cpu/cuda)
    with torch.no_grad():             disable gradient calculation
        pred = model(x)               forward pass (compute output)
        preds.append(pred.cpu())      collect prediction
```

Save/Load a Neural Network

- Save

```
torch.save(model.state_dict(), path)
```

- Load

```
ckpt = torch.load(path)
model.load_state_dict(ckpt)
```

More About PyTorch

- torchaudio
  - speech/audio processing
- torchtext
  - natural language processing
- torchvision
  - computer vision
- skorch
  - scikit-learn + pyTorch

More About PyTorch

- Useful github repositories using PyTorch
  - Huggingface Transformers (transformer models: BERT, GPT, ...)
  - Fairseq (sequence modeling for NLP & speech)
  - ESPnet (speech recognition, translation, synthesis, ...)
  - Many implementation of papers
  - ...

Reference

- https://pytorch.org/
- https://github.com/pytorch/pytorch
- https://github.com/wkentaro/pytorch-for-numpy-users
- https://blog.udacity.com/2020/05/pytorch-vs-tensorflow-what-you-need-to-know.html
- https://www.tensorflow.org/
- https://numpy.org/

Any questions?