# Pytorch 基础

内容大多引自台湾大学李宏毅老师2022

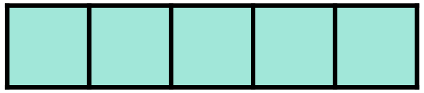
春季机器学习课程

# 什么是PyTorch?

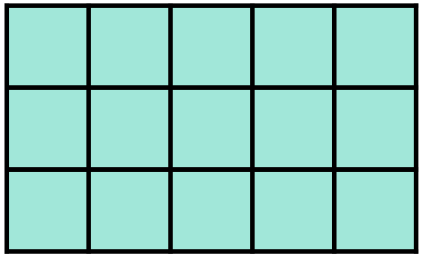
- 一种基于Python语言的深度学习开发框架.
- 特征:
    - 使用Pytorch可实现在GPU上进行N维张量的计算操作
    - 对训练网络自动进行求导

# Tensors—张量

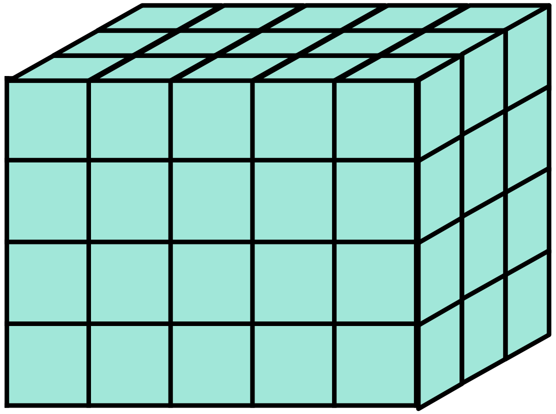
- High-dimensional matrices (arrays)



1-D tensor
e.g. audio



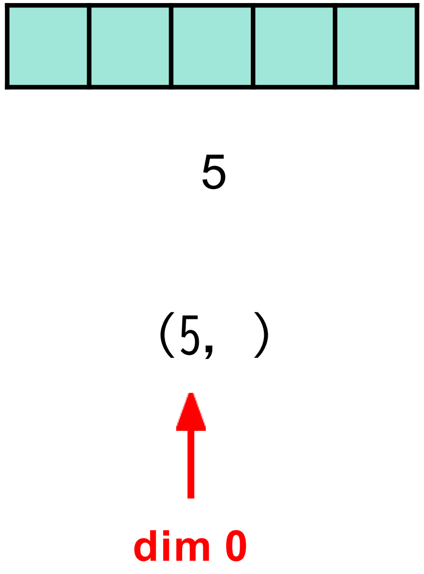
2-D tensor
e.g. gray images



3-D tensor
e.g. RGB images

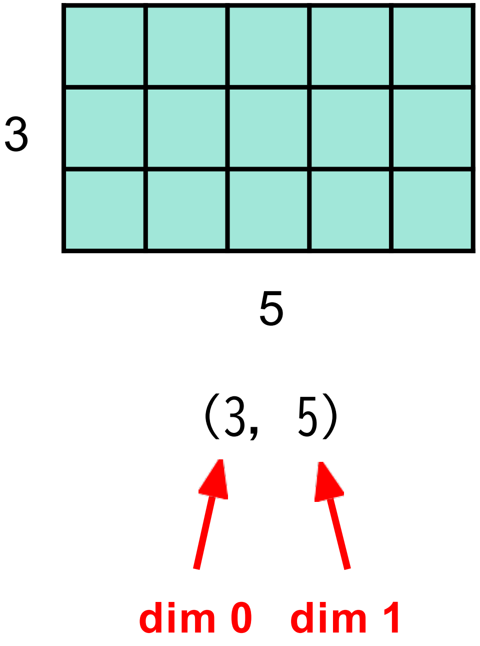
# Tensors – Shape of Tensors

- Check with .shape()



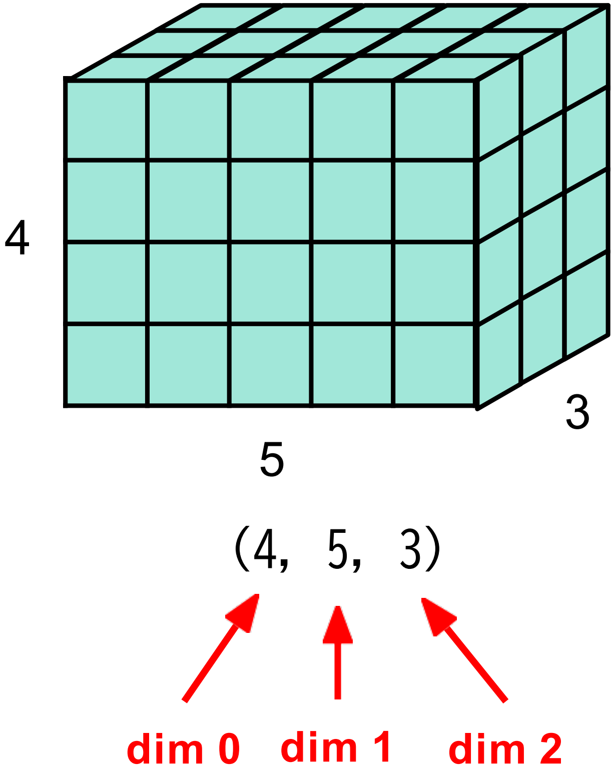
(5, )

dim 0



(3, 5)

dim 0 dim 1



(4, 5, 3)

dim 0 dim 1 dim 2

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

# Tensors – 创建 Tensors

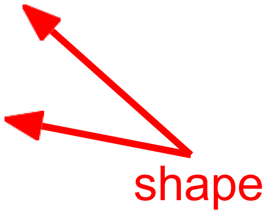
- 从已有数据创建(list or numpy.ndarray)

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

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

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

- Tensor of constant zeros & ones

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

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



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

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

# Tensors – 常见的操作

Pytorch支持一些常见的数学运算，如:

- Addition

```
z = x + y
```

- Subtraction

```
z = x - y
```

- Power

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

- Summation

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

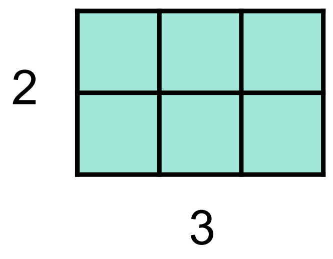
- Mean

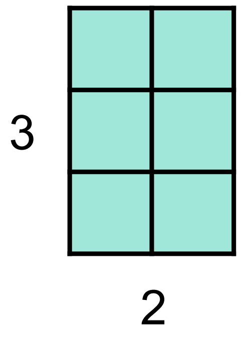
```
y = x.mean()
```

# Tensors – 常见的操作

- **Transpose**: 交换张量中两个指定的维度

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





# Tensors – 常见的操作

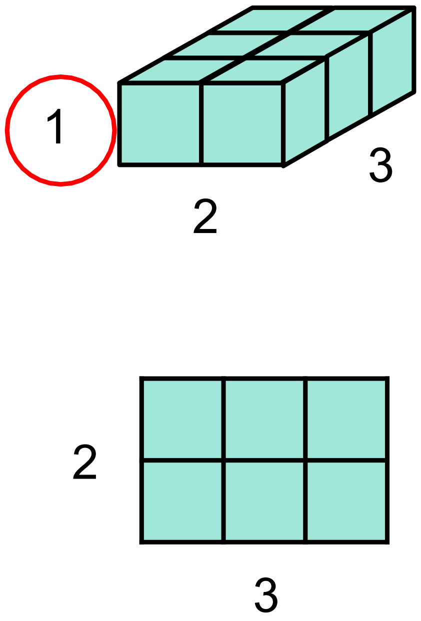
- **Permute**: 交换张量中任意的维度

```
>>> x = torch.zeros([2, 3, 4, 5])
>>> x.shape
torch.Size([2, 3,4,5])
>>> x = x.permute(3,1,0,2).contiguous()
>>> x.shape
torch.Size([5,3,2,4])
```

# Tensors – Common Operations

- **Squeeze**: 删除张量中指定的大小为1的维度

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

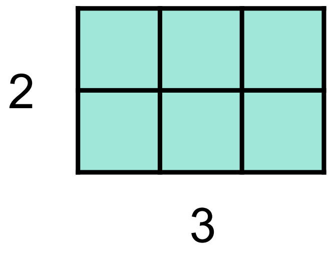


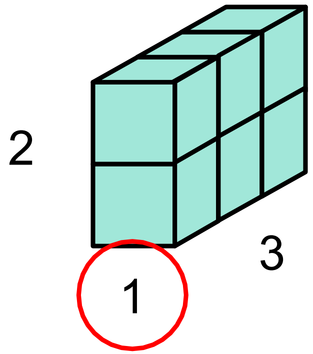
# Tensors – 常见的操作

- **Unsqueeze**: 扩充一个新维度

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

(dim = 1)

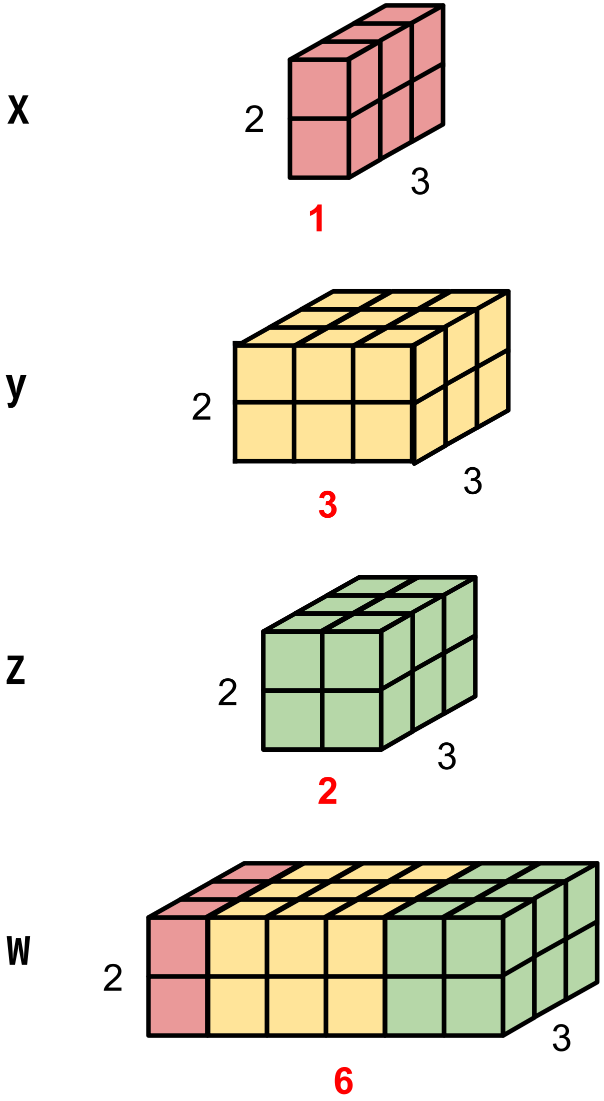




# Tensors – 常见的操作

- **Cat**: 将多个张量沿着指定维度进行拼接

```
>>> 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])
```



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

# Tensors – 数据类型

- Using different data types for model and data will cause errors.

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

see official documentation for more information on data types.

# Tensors – PyTorch v.s. NumPy

- 相似的属性

| PyTorch | NumPy |
|---|---|
| x.shape | x.shape |
| x.dtype | x.dtype |

see official documentation for more information on data types.

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

# Tensors – PyTorch v.s. NumPy

- 许多函数具有相同的名字

| 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

# Tensors – Device （GPU or CPU）

- 创建的张量以及模型默认在CPU上进行相关计算

  使用 **.to()** 可将张量以及模型移动到指定的设备上.
- CPU

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

- GPU

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

# Tensors – Device (GPU)

- Check if your computer has NVIDIA GPU

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

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

- Why use GPUs?
  - Parallel computing with more cores for arithmetic calculations
  - See What is a GPU and do you need one in deep learning?

# Tensors – 梯度计算

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

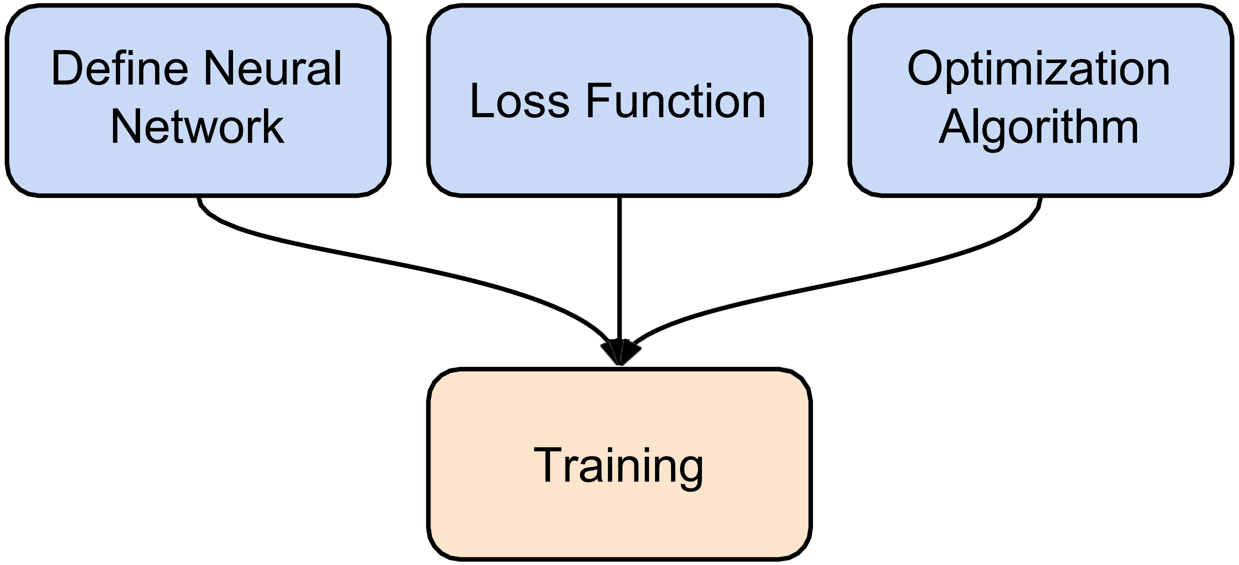
2 $z = \sum_i \sum_j x_{i,j}^2$

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

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

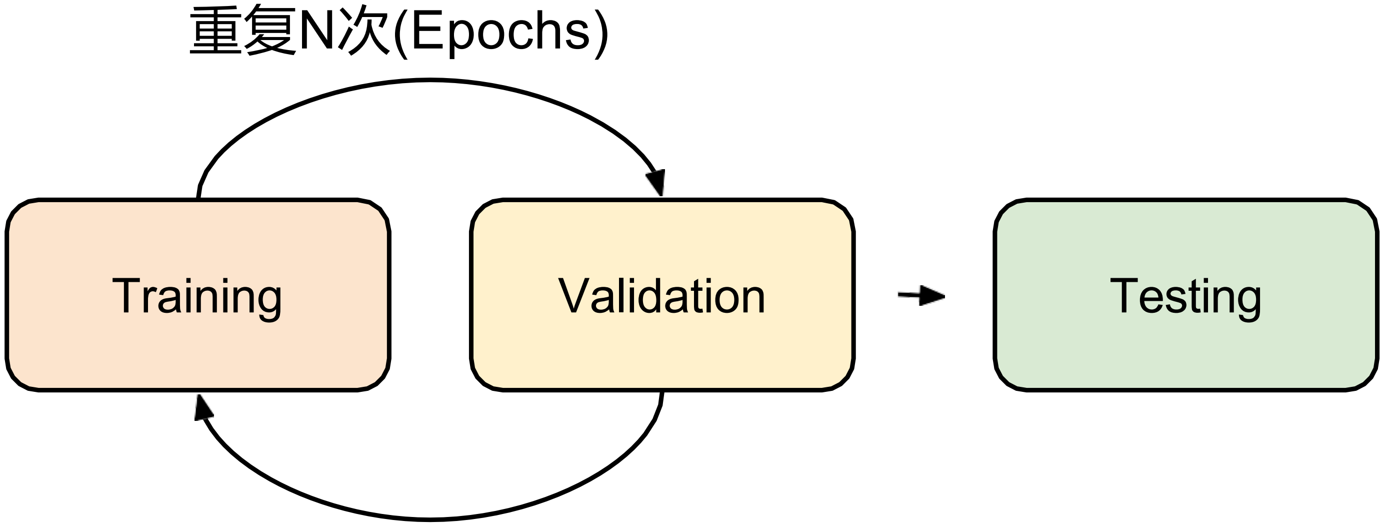
See here to learn about gradient calculation.

# 训练网络任务构建



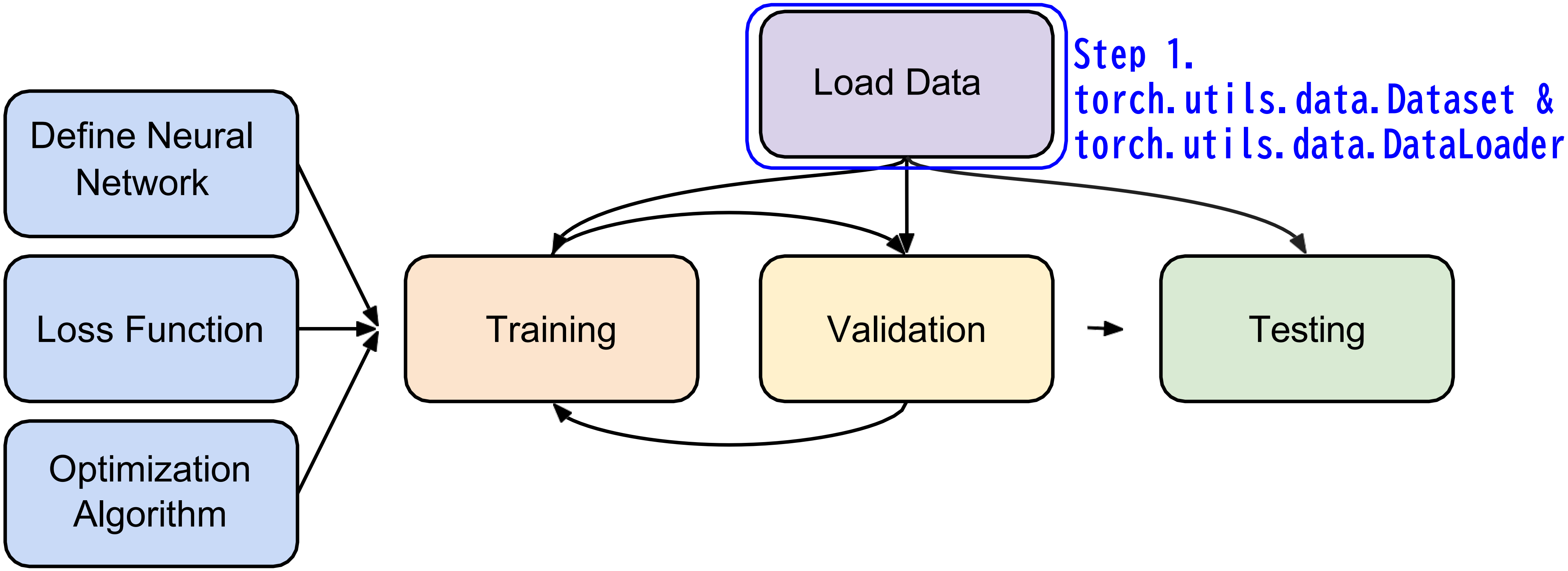
More info about the training process in last year's lecture video.

# 训练 & 测试神经网络



Guide for training/validation/testing can be found here.

# 训练 & 测试神经网络步骤- in Pytorch



Step 1.
torch.utils.data.Dataset &
torch.utils.data.DataLoader

# Dataset & Dataloader

- Dataset: 存储数据的每一个样本及其真值 ( ground-truth )
- Dataloader: 将多个样本打包成一个batch

- `dataset = MyDataset(file)`
- `dataloader = DataLoader(dataset, batch_size, shuffle=True)`

Training: True
Testing: False

More info about batches and shuffling here.

# Dataset & Dataloader

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

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

Read data & preprocess

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

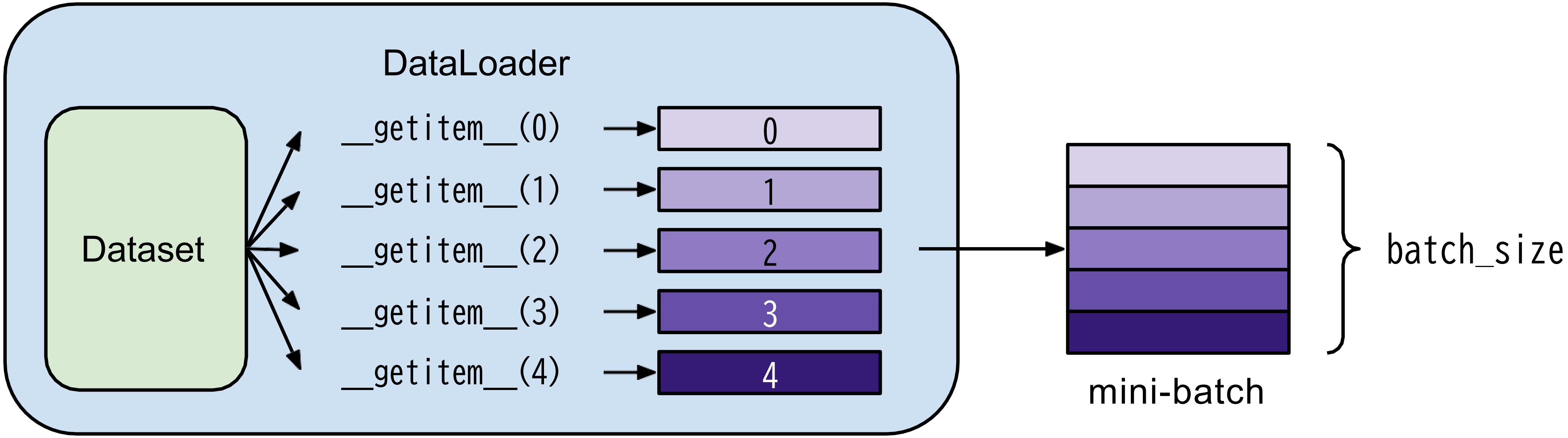
Returns one sample at a time

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

Returns the size of the dataset

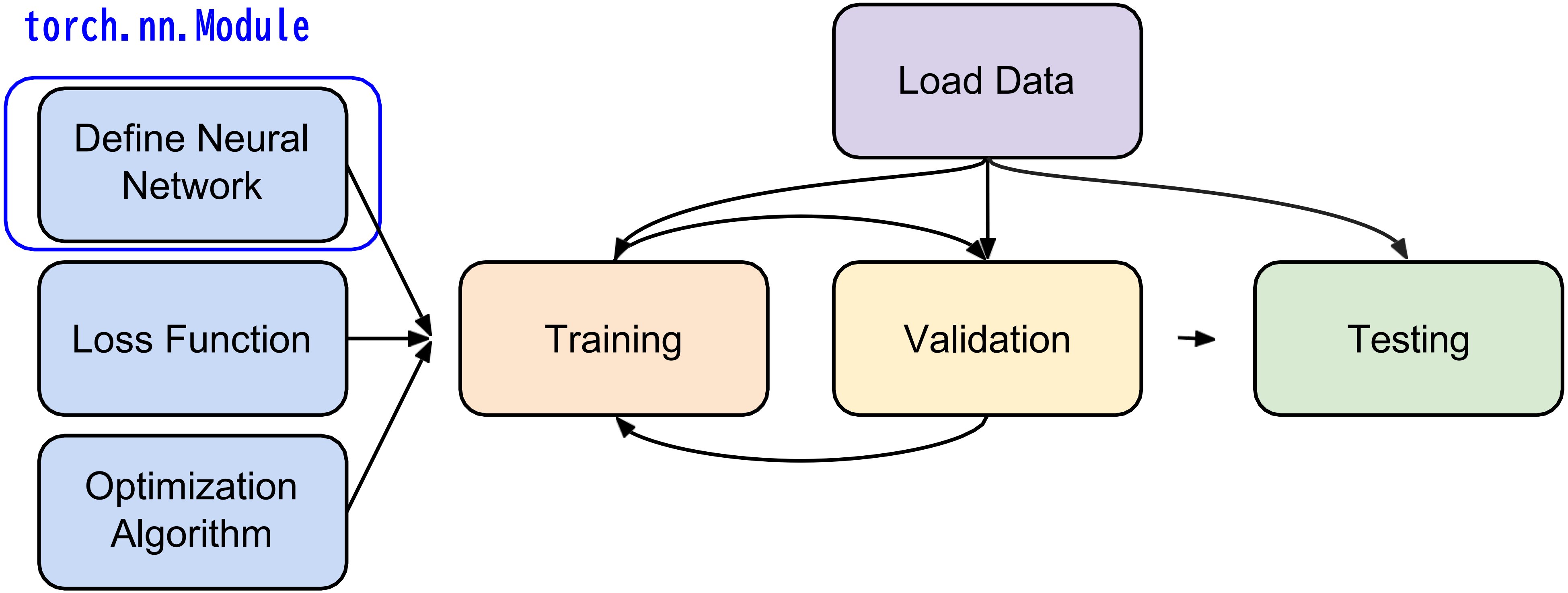
# Dataset & Dataloader

```
dataset = MyDataset(file)

dataloader = DataLoader(dataset, batch_size=5, shuffle=False)
```



# 训练 & 测试神经网络步骤– in Pytorch

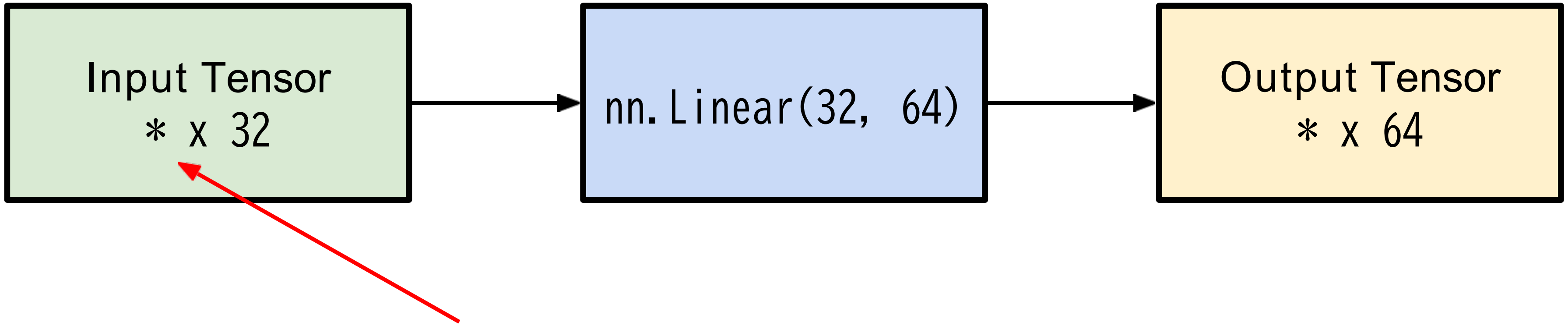
Step 2.
torch.nn.Module



# torch.nn – 网络层

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

```
nn.Linear(in_features, out_features)
```



Input Tensor
* x 32

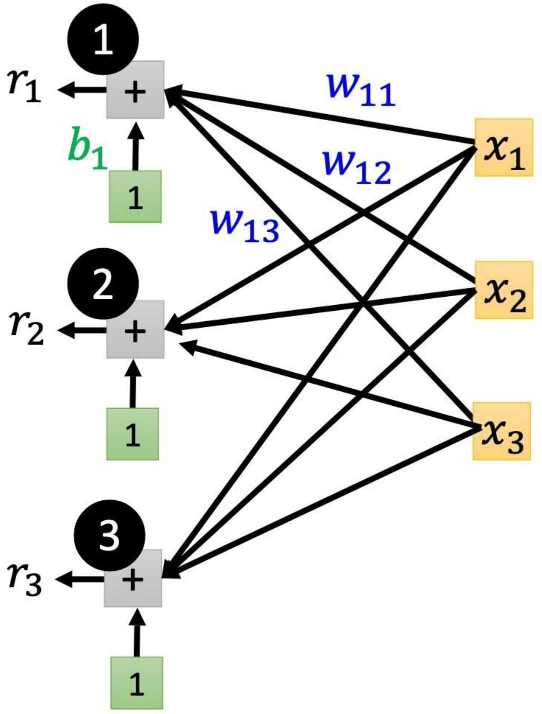
nn.Linear(32, 64)

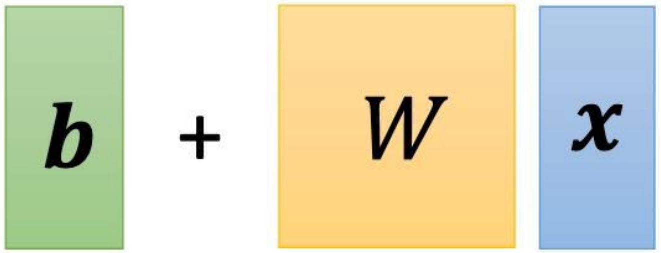
Output Tensor
* x 64

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

# torch.nn – 网络层

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

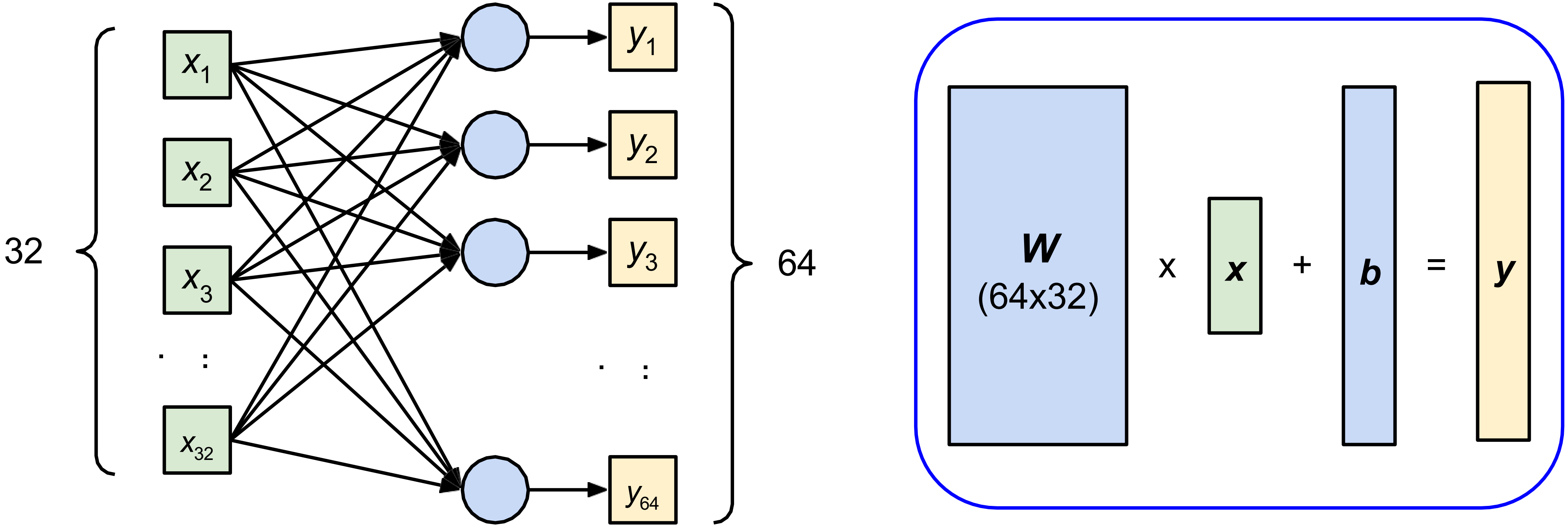




ref: last year's lecture video

# torch.nn – 网络层

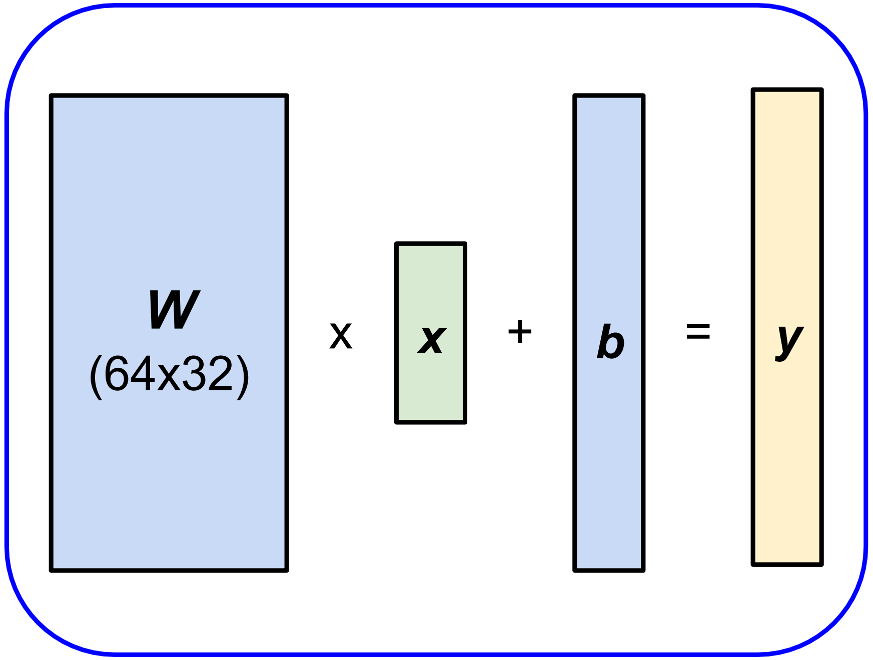
- Linear Layer (**Fully-connected** Layer)



# torch.nn – 网络参数

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

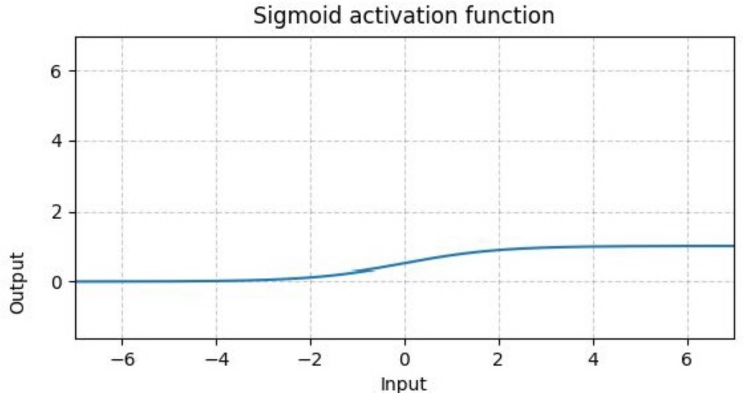
```
>>> layer = torch.nn.Linear(32, 64)
>>> layer.weight.shape
torch.Size([64, 32])
>>> layer.bias.shape
torch.Size([64])
```



# torch.nn – 非线性激活函数

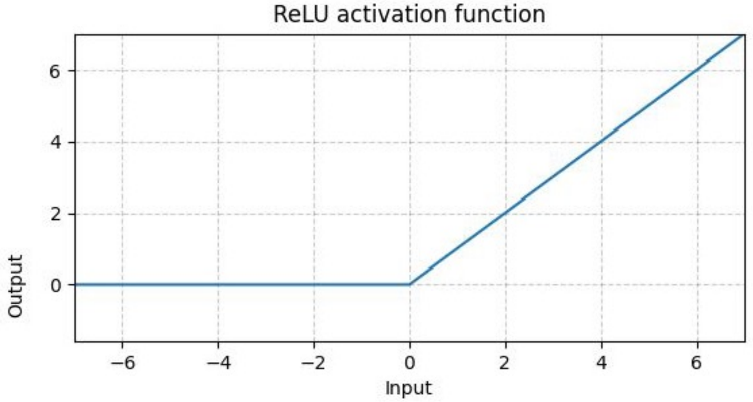
- Sigmoid Activation

```
nn.Sigmoid()
```



- ReLU Activation

```
nn.ReLU()
```



See here to learn about why we need activation functions.

# torch.nn – 自定义一个神经网络

```
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 – 自定义一个神经网络

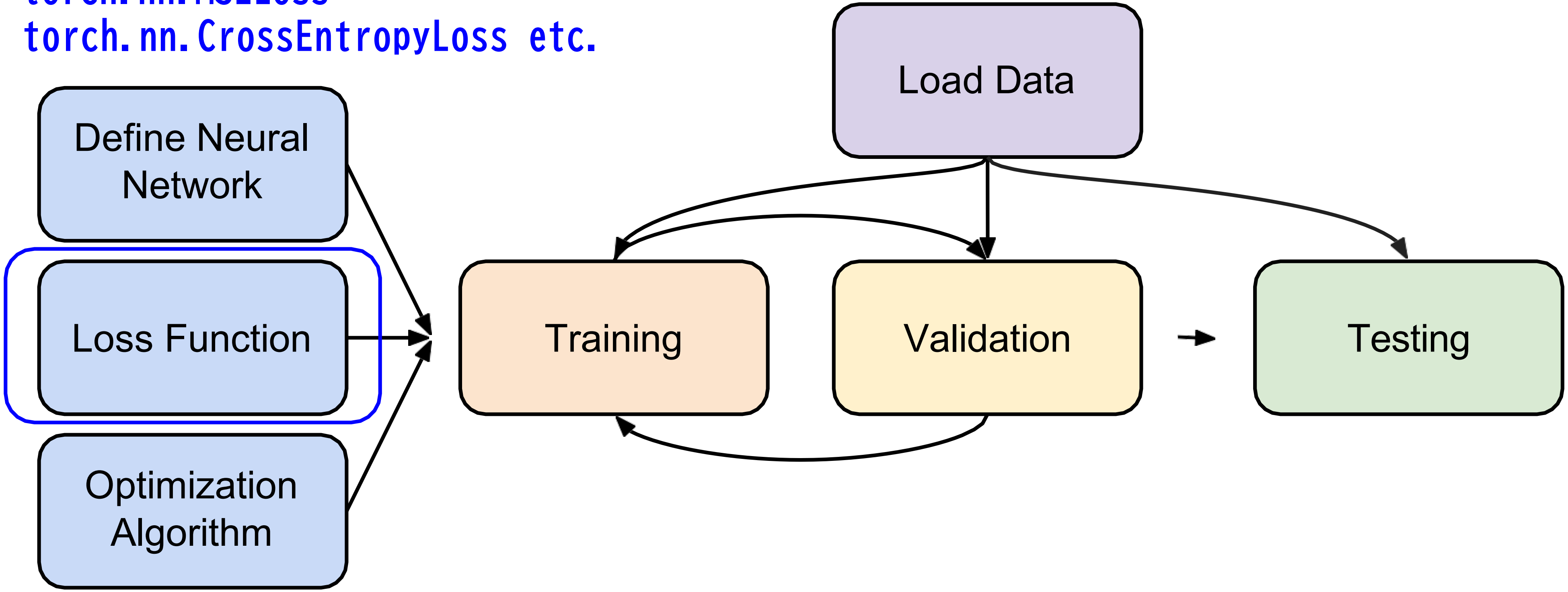
```
import torch.nn as nn

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)
```

=

```
import torch.nn as nn

Class MyModel(nn.Module):
    def __init__(self):
        super(MyModel, self).__init__()
        self.layer1 = nn.Linear(10, 32)
        self.layer2 = nn.Sigmoid(),
        self.layer3 = nn.Linear(32,1)

    def forward(self, x):
            out                 =
        self.layer1(x)
        out = self.layer2(out)
        out= self.layer3(out)
        return out
```

# 训练 & 测试神经网络步骤 – in Pytorch

```
Step 3.
torch.nn.MSELoss
torch.nn.CrossEntropyLoss etc.
```



# torch.nn – 损失函数

- Mean Squared Error (for regression tasks)

  ```
  criterion = nn.MSELoss()
  ```

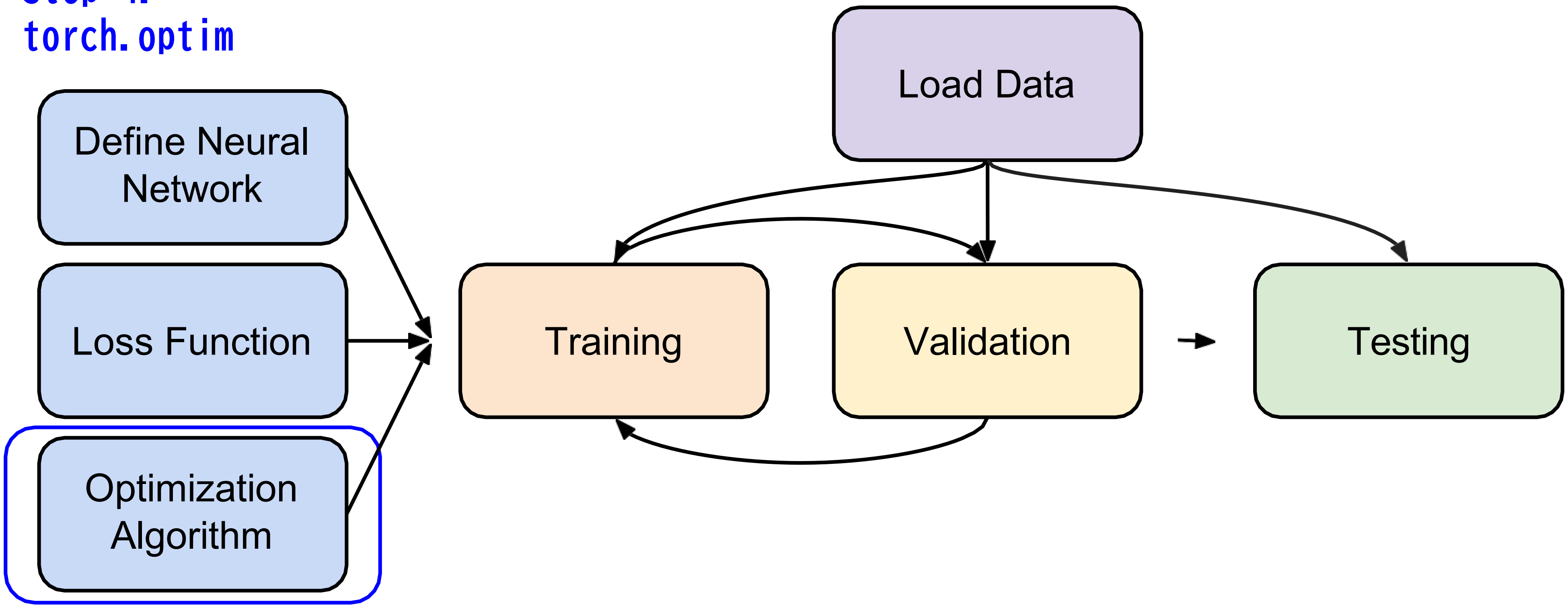
- Cross Entropy (for classification tasks)

  ```
  criterion = nn.CrossEntropyLoss()
  ```

- `loss = criterion(model_output, ground_truth_value)`

# 训练 & 测试神经网络步骤 – in Pytorch

Step 4.
torch.optim



# torch.optim

- 包含了基于梯度下降优化网络参数以减小预测损失的算法. (See Adaptive Learning Rate lecture video)

- E.g. Stochastic Gradient Descent (SGD)—随机梯度下降

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

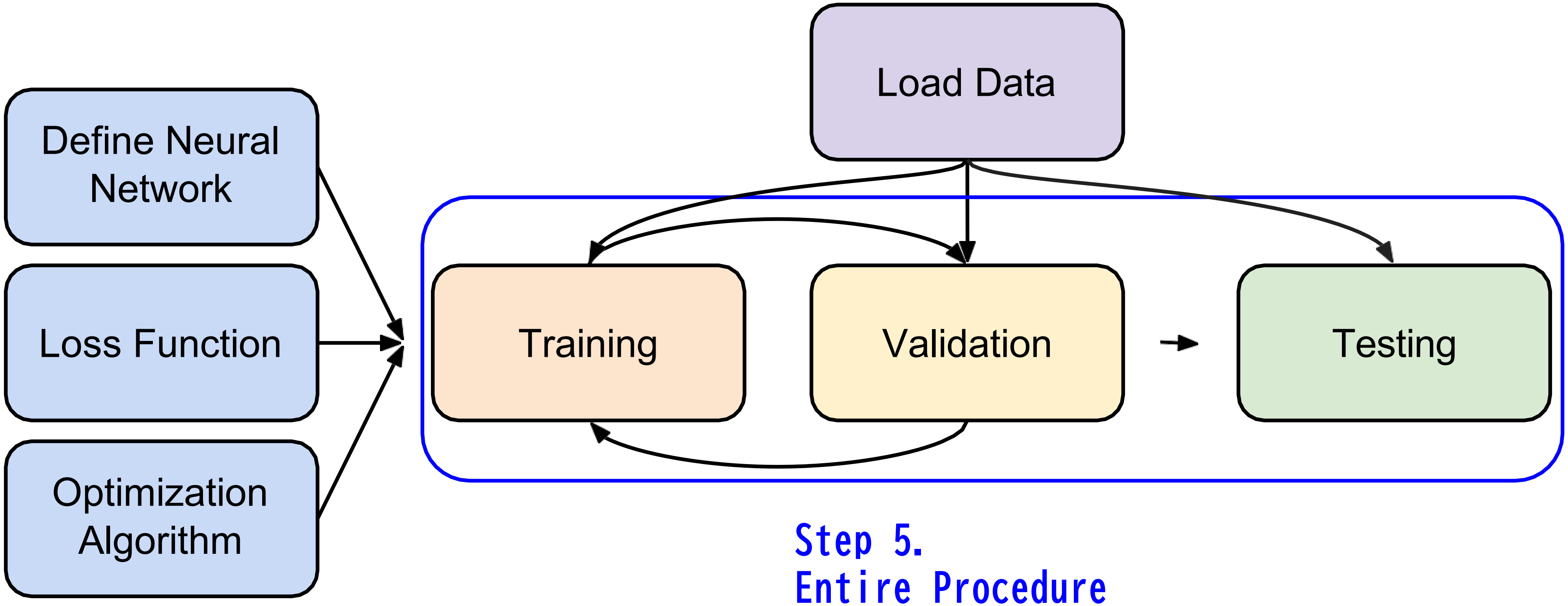
# torch.optim

```
optimizer = torch.optim.SGD(model.parameters(), lr, momentum = 0)
```

- 对于训练迭代中的每一个batch的数据:
  1. 调用 optimizer.zero_grad() 将参数的梯度重置为0.
  2. 调用 loss.backward()对预测得到的损失进行方向传播来计算参数梯度.
  3. 调用optimizer.step()根据梯度值来调整模型参数.

See official documentation for more optimization algorithms.

# 训练&测试神经网络步骤5 – in Pytorch



Step 5.
Entire Procedure

# Neural Network Training Setup

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

# 循环训练过程

```
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
```

# 循环验证过程

```
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
```

# 循环测试过程

```
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
```

# 注意 - model.eval(), torch.no_grad()

- `model.eval()`

  改变一些层的某些操作, 如dropout和 batch normalization.

- `with torch.no_grad()`

  在该代码域中网络不计算参数梯度，能够减少显存消耗，加快推理速度.

# 保存/加载训练好的模型

- 保存

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

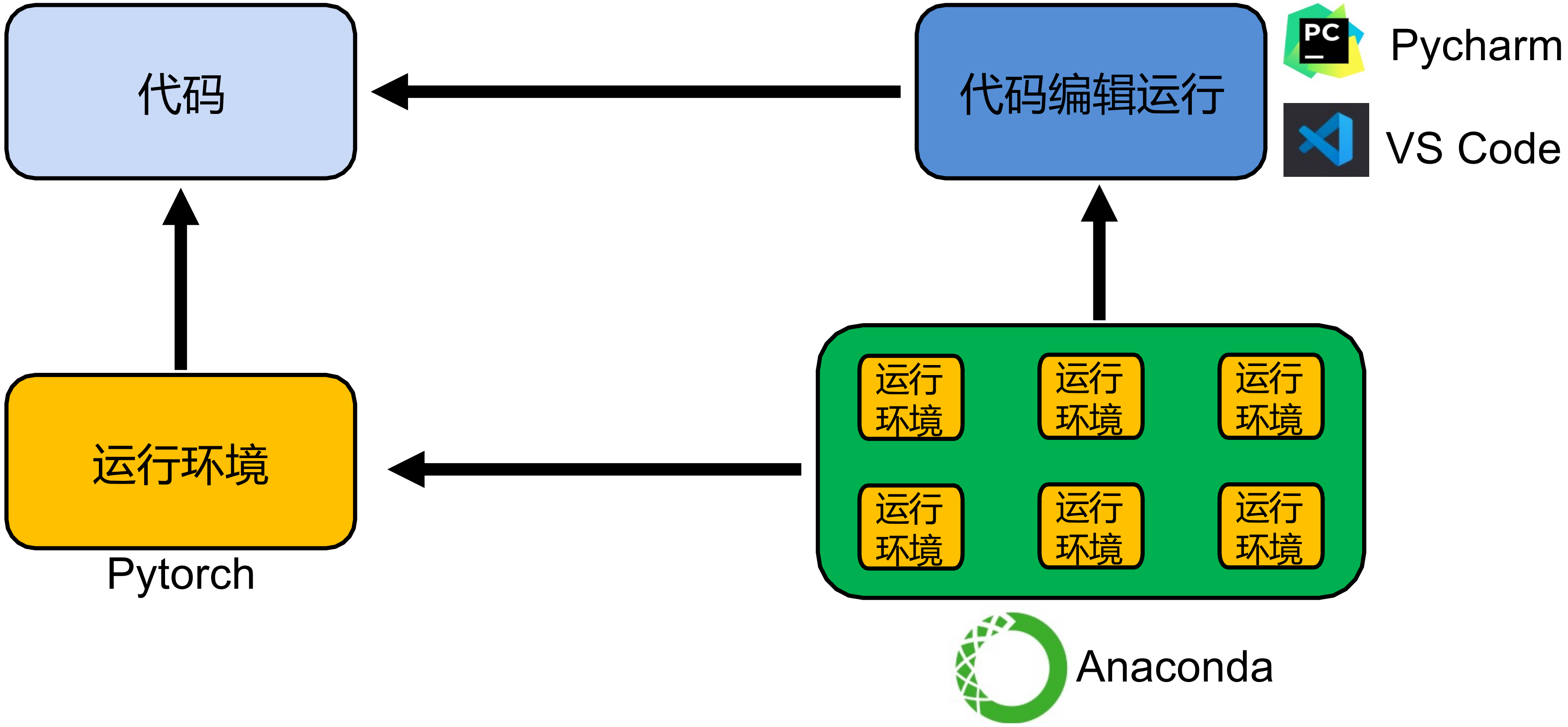
- 加载

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

# More About PyTorch

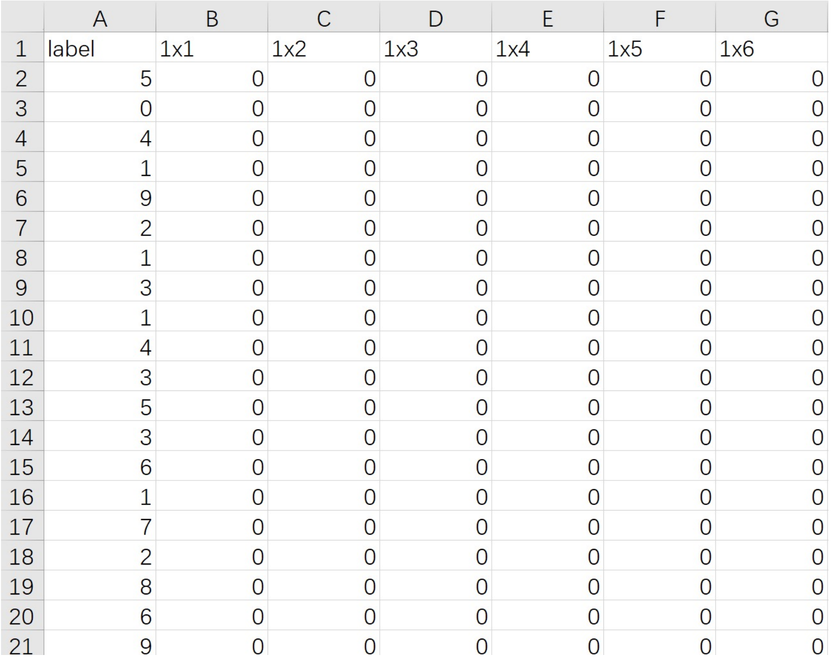
- torchaudio
  - speech/audio processing
- torchtext
  - natural language processing
- torchvision
  - computer vision
- skorch
  - scikit-learn + pyTorch

# 手写数字识别实例—构建运行环境



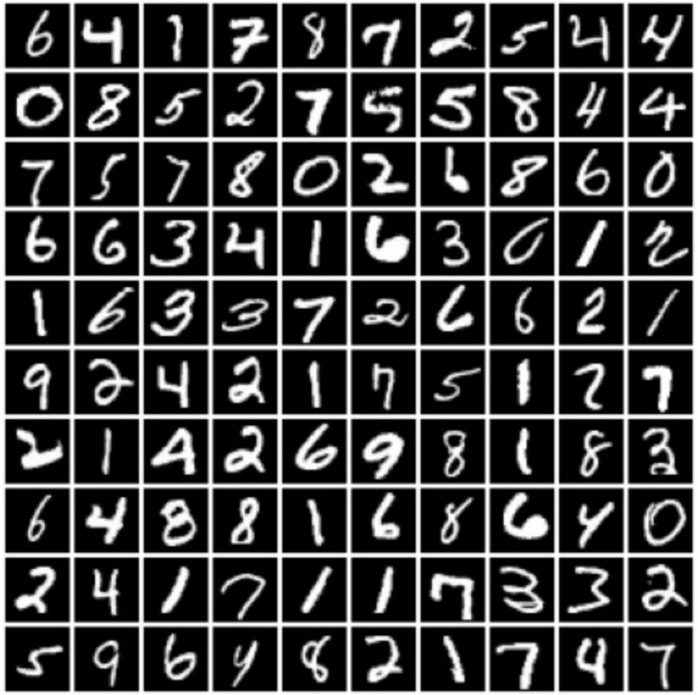
# 手写数字识别实例—数据集

- MNIST
- Train:60000张; Test:10000张
- 重分配 Train:50000张; Val:10000张; Test:10000张

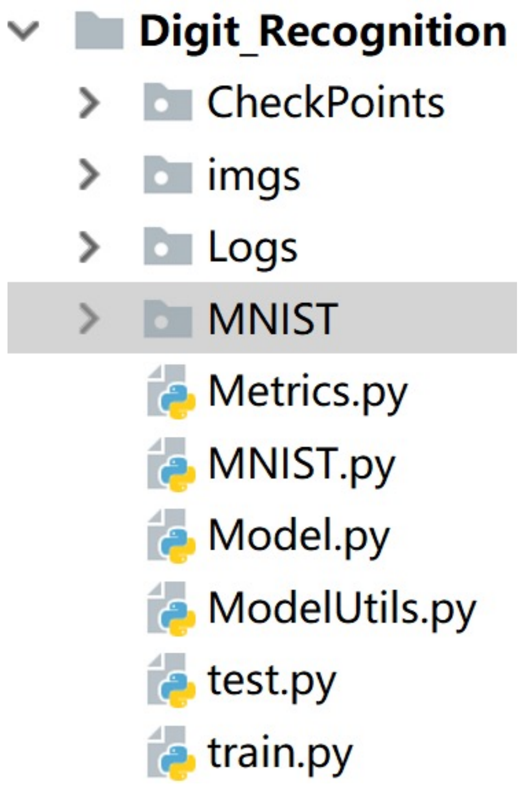


| | A | B | C | D | E | F | G |
|---|---|---|---|---|---|---|---|
| 1 | label | 1x1 | 1x2 | 1x3 | 1x4 | 1x5 | 1x6 |
| 2 | 5 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 4 | 4 | 0 | 0 | 0 | 0 | 0 | 0 |
| 5 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 6 | 9 | 0 | 0 | 0 | 0 | 0 | 0 |
| 7 | 2 | 0 | 0 | 0 | 0 | 0 | 0 |
| 8 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 9 | 3 | 0 | 0 | 0 | 0 | 0 | 0 |
| 10 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 11 | 4 | 0 | 0 | 0 | 0 | 0 | 0 |
| 12 | 3 | 0 | 0 | 0 | 0 | 0 | 0 |
| 13 | 5 | 0 | 0 | 0 | 0 | 0 | 0 |
| 14 | 3 | 0 | 0 | 0 | 0 | 0 | 0 |
| 15 | 6 | 0 | 0 | 0 | 0 | 0 | 0 |
| 16 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 17 | 7 | 0 | 0 | 0 | 0 | 0 | 0 |
| 18 | 2 | 0 | 0 | 0 | 0 | 0 | 0 |
| 19 | 8 | 0 | 0 | 0 | 0 | 0 | 0 |
| 20 | 6 | 0 | 0 | 0 | 0 | 0 | 0 |
| 21 | 9 | 0 | 0 | 0 | 0 | 0 | 0 |

训练集部分数据



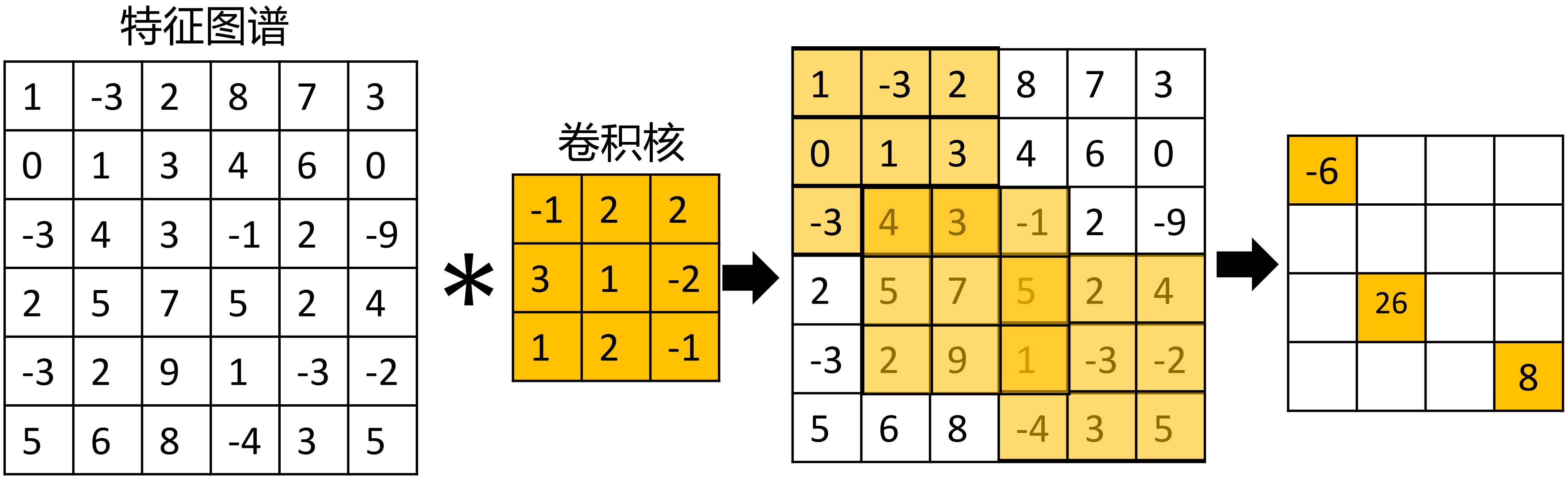
# 手写数字识别实例—代码结构



```
Digit_Recognition
    CheckPoints
    imgs
    Logs
    MNIST
    Metrics.py
    MNIST.py
    Model.py
    ModelUtils.py
    test.py
    train.py
```

- MNIST: 加载和处理数据集，构建DataLoader
- Model:定义数字识别神经网络
- Metrics:定义评价指标（识别准确率）
- ModelUtils:用于保存模型和加载模型
- train:训练模型
- test:测试模型

# 手写数字识别实例—CNN运算



特征图谱

|  |  |  |  |  |  |
|---|---|---|---|---|---|
| 1 | -3 | 2 | 8 | 7 | 3 |
| 0 | 1 | 3 | 4 | 6 | 0 |
| -3 | 4 | 3 | -1 | 2 | -9 |
| 2 | 5 | 7 | 5 | 2 | 4 |
| -3 | 2 | 9 | 1 | -3 | -2 |
| 5 | 6 | 8 | -4 | 3 | 5 |

*

卷积核

|  |  |  |
|---|---|---|
| -1 | 2 | 2 |
| 3 | 1 | -2 |
| 1 | 2 | -1 |

|  |  |  |  |  |  |
|---|---|---|---|---|---|
| 1 | -3 | 2 | 8 | 7 | 3 |
| 0 | 1 | 3 | 4 | 6 | 0 |
| -3 | 4 | 3 | -1 | 2 | -9 |
| 2 | 5 | 7 | 5 | 2 | 4 |
| -3 | 2 | 9 | 1 | -3 | -2 |
| 5 | 6 | 8 | -4 | 3 | 5 |

|  |  |  |  |
|---|---|---|---|
| -6 |  |  |  |
|  |  |  |  |
|  | 26 |  |  |
|  |  |  | 8 |

# 手写数字识别实例—CNN特征图谱输出大小计算

- 输入数据input的shape为$[B, C, H, W]$ ➔B: batch_size, C: channel, H: height, W:width
- 卷积核大小$k \times k$, 卷积核滑动步长为$s$, 输入数据上下左右填充数为$p$
- 卷积后输出的$H_{out} = \left\lfloor \frac{H+2p-k}{s} \right\rfloor + 1, W_{out} = \left\lfloor \frac{W+2p-k}{s} \right\rfloor + 1$
- 若要保证$H_{out} = H, W_{out} = W$, 一般令$s = 1, p = \left\lfloor \frac{k}{2} \right\rfloor$
- 若要保证$H_{out} = \frac{H}{2}, W_{out} = \frac{W}{2}$, 一般令$s = 2\ p = \left\lfloor \frac{k}{2} \right\rfloor$
- conv=torch.nn.Conv2d($C$,$C_{out}$,kernel_size = $k$ , stride= $s$, padding= $p$)
- output = conv(input)

| | | | | | |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 3 | 4 | 6 | 0 |
| 0 | 4 | 3 | -1 | 2 | 0 |
| 0 | 5 | 7 | 5 | 8 | 0 |
| 0 | 2 | 9 | 6 | 7 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 |

带填充的特征图谱

# 资源链接

- Fafa-DL/Lhy_Machine_Learning: 李宏毅2021/2022/2023春季机器学习课程课件及作业 (github.com)
- (强推)李宏毅2021/2022春机器学习课程_哔哩哔哩_bilibili(强推)李宏毅2021/2022春机器学习课程_哔哩哔哩_bilibili

PyTorch

结束