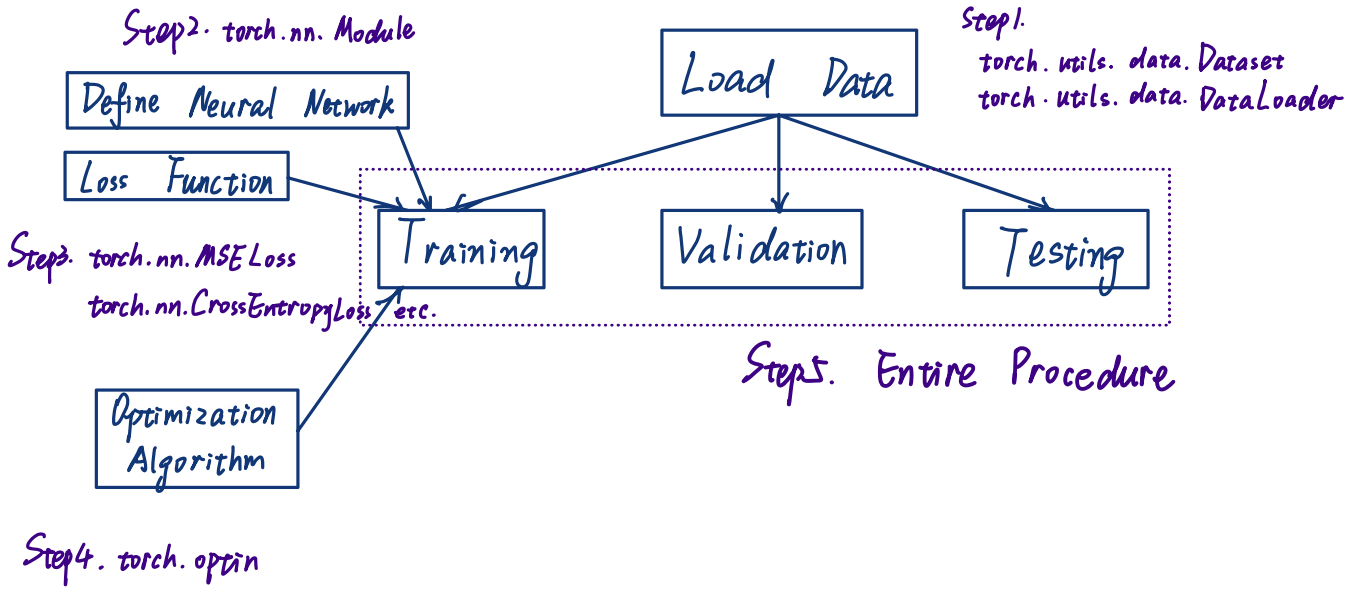
△ Pytorch. (优点：可用GPU加速；计算梯度十分方便).

- 训练和测试神经网络：



Dataset：读入原始资料，并用一个class将其打包好。

Dataloader：把资料合成一个 batches.

Testing = False.
Training = True

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

- 定义自己的 dataset：

```
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.

- Tensors：(矩阵/阵列).
  - High-dimensional matrices (arrays)
    一维：一维阵列　二维：黑白图像　三维：彩色图像.
  - Check with .shape() 查看阵列有几个维度，每个维度大小是多少.
  - x = x.transpose(0,1) 维度互换.
  - x = x.squeeze(0) 删除维度.
  - x = x.unsqueeze(1) 添加一个维度.
  - 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)

- `torch.cuda.is_available()` 查看电脑是否可用GPU.
- `x = torch.tensor([[1., 0.], [-1., 1.]], requires_grad=True)`

  `z = x.pow(2).sum()`

  `z.backward()` 计算偏微分.

  `x.grad`

  `tensor([[2., 0.], [-2., 2.]])`
- `nn.Linear(in_features, out_features)`
- `>>> layer = torch.nn.Linear(32, 64)`

  `>>> layer.weight.shape` 查看矩阵大小.

  `torch.Size([64, 32])`

  `>>> layer.bias.shape`

  `torch.Size([64])`
- `nn.Sigmoid()` `nn.ReLU()` } 在Pytorch中已写好.可直接使用
- 用layer组成神经网络.

```
import torch.nn as nn
class MyModel(nn.Model):
    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)
```

初始化模型&定义层.

计算nn的输出.

- Loss Functions

```
criterion = nn.MSELoss()
```
计算均方误差.

```
criterion = nn.CrossEntropyLoss()
```
交叉熵 ?

```
loss = criterion(model_output, expected_value)
```

- 建立神经网络训练模型.

```
dataset = MyDataset(file)
```
从MyDataset中读入数据.

```
tr_set = DataLoader(dataset, 16, shuffle = True)
```
将数据集写入DataLoader

```
model = MyModel().to(device)
criterion = nn.MSELoss()
optimizer = torch.optim.SGD(model.parameters(), 0.1)
for epoch in range(n_epochs):
    model.train()
    for x, y in tr_set:
        optimizer.zero_grad()
        x, y = x.to(device), y.to(device)
        pred = model(x)
        loss = criterion(pred, y)
        loss.backward()
        optimizer.step()
```