一、模型训练过程

```
for X, y in train_iter:          #每次送入batch_size大小
                                   的数据.
    y_pre = net(X)                 #前向传播
     l = loss(y_pre, y).sum()      #计算损失
     optimizer.zero_grad()         #梯度清零
     l.backward()                  #后向传播
     optimizer.step()              #更新参数.
```

二、模型构建

```
class netModule (nn.Module):
    def __init__(self, num_input, num_hidelayer1 ..., num_
                                                        output
        super(self, netModule).__init__()
        self.hidelayer1 = nn.Linear(num_input, num_hidelayer
        ...
        self.outputlayer = nn.Linear(num_hidelayer, num_output
    def forward(self, X)
        X1 = F.relu(self.hidelayer1(X.view(X.shape[0], -1)))
        ...
```

y = self.outputlayer(X1)
return y.

注：nn.functional 和 nn 区别：
若使用 nn.functional 中的参数，我们需要手动维护 weight 和 bias，stride 等值。
因此，需要自动更新的使用 nn 模块，如 nn.Linear，
而 relu() 可以使用 nn.functional

二、损失函数.

(1) torch.nn.L1Loss(size_average = True)

$$\rightarrow \text{loss}(x,y) = \frac{1}{n}\sum|x_i - y_i|$$

其中 size_average 若为 False，则无 $\frac{1}{n}$.
若不平均，loss较大，梯度下降收敛慢，……

(2) torch.nn.MSELoss(～)

$$\rightarrow \text{loss}(x,y) = \frac{1}{n}\sum(x_i - y_i)^2$$

(3) torch.nn.CrossEntropyLoss(Weight = None, ～)

$$\rightarrow \text{loss}(x,\text{class}) = -\log\frac{\exp(x[\text{class}])}{\sum_j \exp(x[j])}$$

$$= -x[\text{class}] + \log(\sum_j \exp(x[j]))$$

☆. 当 weight 参数被指定，

loss′(x, class) = weights[class] * loss(x, class)

三. 优化函数 (torch.optim)

1. torch.optim.SGD (params, lr, momentum=0, dampening=0, weight_decay=0, nesterov=False).

随机梯度下降

→ momentum : 动量因子 ☆

weight_decay : 权重衰减 (L_2 正则化)

dampening : 动量的抑制因子 ☆.

...

⇒ 如何使用 optimizer

① 构建

optimizer = optim.SGD(model.parameters(), lr=0.01, momentum=0.9)

② 为每个参数设置选项

optim.SGD([{'params': }, 'params': }])

)

③ 梯度清零

optimizer.zero_grad()

④ 单步优化

optimizer.step()

四、读取数据 (torch.utils.data)

```
dataset = TensorDataset (data_tensor, target_tensor)
                          样本数据      样本标签

loader = Dataloader (dataset, batch_size=1, shuffle=False,
            sampler=None, num_workers=0, pin_memory=False,
            drop_last=False)
```

- shuffle：每个epoch加载数据都会打乱顺序.
- num_workers： 用多少个子进程加载数据
- pin_memory：锁页内存，pin_memory=True，则意味着生成Tensor数据最开始属于内存中的锁页内存，这样将内存的Tensor转到GPU的显存会快.