信息网络专题研究大作业之机器学习

一、线性回归（Linear Regression）：“回归分析”是确定两种或两种以上变量间相互依赖的定量关系的一种统计分析方法。线性回归是由一个神经元组成的神经网络，结构比较简单。编程的核心思想可以概括为以下四部分：

① Prepare data set——确定数据集 x_data 和 y_data。

② Design model using Class——建立模型并根据权重求解 y_hat（预测结果）。

③ Construct loss and optimizer——建立损失和优化函数。

④ Training cycle——训练迭代（前馈+反馈+更新）。

【核心代码解析】：

1. 构造模型：计算图当中的线性单元主要完成 y_hat=w*x+b 的计算。需要提前知道 x、y 的维度（数据矩阵列数）以便于进行 torch 张量运算。直接调用 nn.Linear 构造出线性模型。

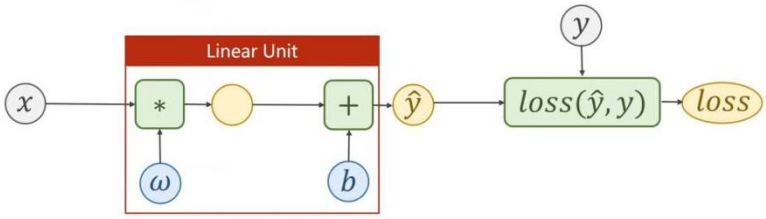
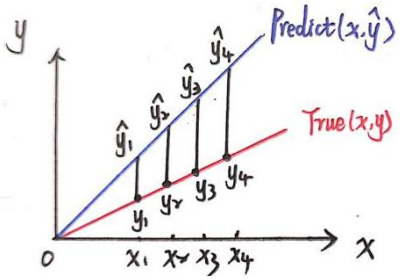


图 1 计算图

```
# Linear regression model
model = nn.Linear(input_size, output_size)
```

2. 损失函数与优化器：损失函数用于前馈，优化器用于调整权重控制收敛。

■ Loss：损失函数的构造是基于 MSE（平均误差函数）。

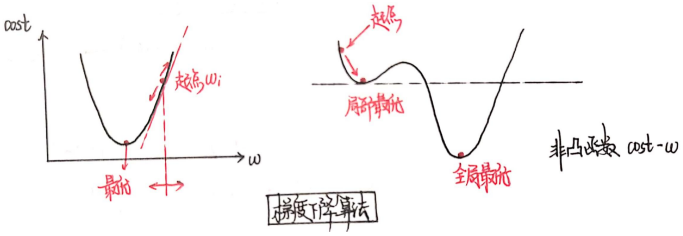


$$Loss_i = \left(\hat{y_i} - y_i\right)^2 = \left(y_i - \omega x_i - b\right)^2$$

$$MSE = Cost = mean\ Loss = \frac{1}{N}\sum_{i=1}^{N}\left(\hat{y_i} - y_i\right)^2$$

$$Goal = \min_{\omega,b} MSE$$

■ Optimizer：基于随机梯度下降算法 SGD，以单一权重 w 为例讲解：

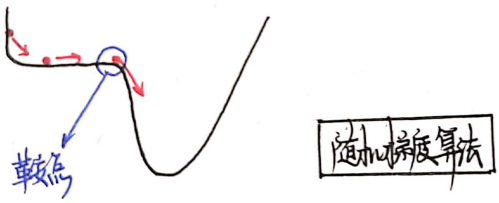


权重调整：$\omega = \omega - \alpha\frac{\partial Cost}{\partial \omega}$

SGD 的前身是梯度下降算法，随机选取权重起点，通过计算梯度得到上升方向，

取负梯度方向进行权重 w 的优化调整，逐步逼近最优权重，但仅能做到局部收敛。学习率α控制每次权重调整的幅度，通常取小一点易于收敛。

梯度下降算法主要受“鞍点”（梯度为 0）的限制。SGD 主要利用单个数据受到的噪声扰动，在训练中可以顺利划过“鞍点”，以克服此瓶颈，达到收敛。



权重：$\omega = \omega - \alpha\frac{\partial Loss}{\partial \omega}$　　$b = b - \alpha\frac{\partial Loss}{\partial b}$

$$\frac{\partial \mathrm{Loss}}{\partial \omega} = 2\left(\omega\sum_{i=1}^{N} x_i^2 - \sum_{i=1}^{N}(y_i - b)x_i\right)$$

$$\frac{\partial \mathrm{Loss}}{\partial b} = 2\left(Nb - \sum_{i=1}^{N}(y_i - \omega x_i)\right)$$

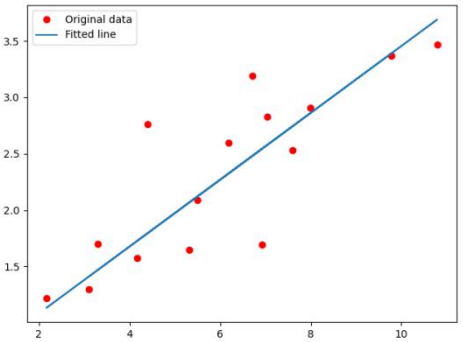
```
# Loss and optimizer
criterion = nn.MSELoss()
optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate)
```

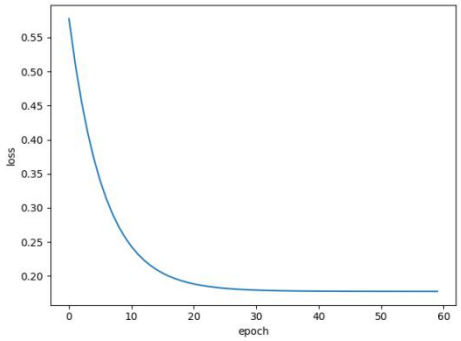
3. 训练：每一轮对所有训练数据的迭代（Epoch）包括，一次 Forward 前馈来计算出 y_predict 和损失 Loss；一次 Backward 反向传播，利用 Loss 输出进行梯度优化，对 model 中所有成员的权重进行自动更新（每次迭代）。重复训练 60 轮。

```
# Forward pass
outputs = model(inputs)
loss = criterion(outputs, targets)
```

```
# Backward and optimize
optimizer.zero_grad()
loss.backward()
optimizer.step()
```

【代码运行结果】： 在源码中添加了权重值打印和损失曲线的绘制。可以看到随着对数据集(Original data）的训练迭代轮次的增加，Loss 逐渐趋于收敛（每 5 轮打印），训练模型输出的预测值 y_hat-x 形成线性 Fitted line；且在每一次模型训练完成后，权重 w、b 都会被不断自动调整，在最后一轮输出的权重(w,b)最接近最优解(w*,b*)。






```
Epoch [5/60], Loss: 0.3717
w= 0.2365158498287201
b= 0.4805639088153839
Epoch [10/60], Loss: 0.2563
w= 0.25850045680999756
b= 0.48406198620796204
Epoch [15/60], Loss: 0.2095
w= 0.27247610688209534
b= 0.4864085614681244
Epoch [20/60], Loss: 0.1905
w= 0.28135421872138977
b= 0.4880220890045166
Epoch [25/60], Loss: 0.1828
w= 0.2869878113269806
b= 0.4891689121723175
Epoch [30/60], Loss: 0.1797
w= 0.29055631160736084
b= 0.4900185167789459
Epoch [35/60], Loss: 0.1784
w= 0.2928104102611542
b= 0.4906787872314453
Epoch [40/60], Loss: 0.1779
w= 0.2942279577255249
b= 0.4912183880805969
Epoch [45/60], Loss: 0.1776
w= 0.29511305689811707
b= 0.4916810393333435
Epoch [50/60], Loss: 0.1775
w= 0.2956593334674835
b= 0.49209460616111755
Epoch [55/60], Loss: 0.1775
w= 0.2959899604320526
b= 0.4924767017364502
Epoch [60/60], Loss: 0.1774
w= 0.296183317899704
b= 0.4928386509418487
```

图 2　Linear Regression 代码运行结果

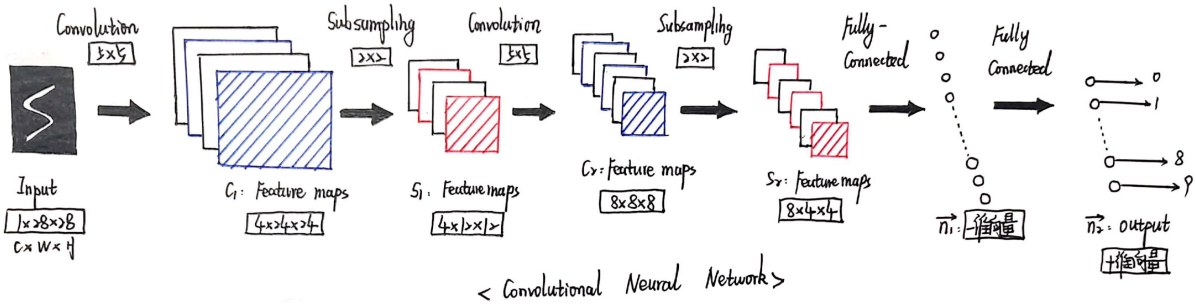
二、卷积神经元网络（CNN）：是一类包含卷积计算且具有深度结构的前馈神经网络。CNN 具有表征学习能力，能够按其阶层结构对输入信息进行平移不变分类。此代码主要是利用 CNN 实现 MNIST 手写数字识别。

【基本原理解析】：

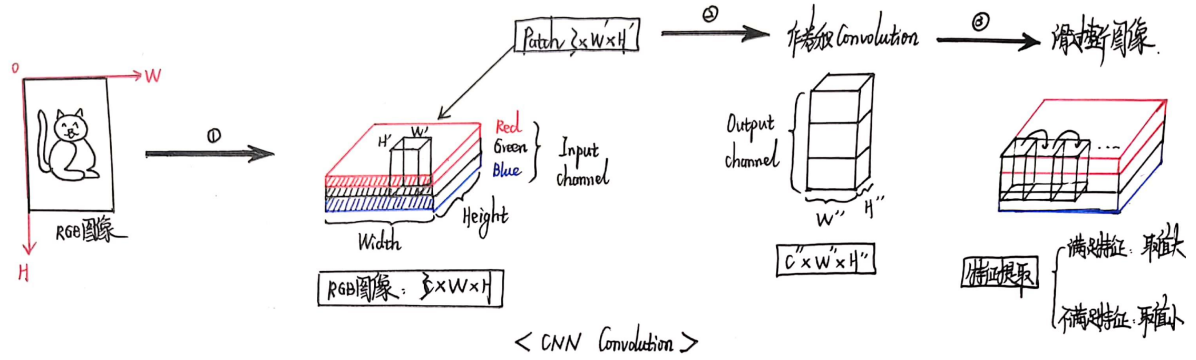
1.MNIST 数据集：简单来说，它就是一个手写体数据集（如图 3 所示）。这个数据集主要由四部分组成（训练图片集 x_train、训练标签集 y_label，测试图片集、测试标签集）。一共包含 0-9 这 10 个 label 类别，需要通过 CNN 对其进行特征提取和分类识别。

图 3　MINIST

2.CNN 网络结构：由卷积层（Convolution）和池化层（下采样 Sub-sampling）组成特征提取器，全连接层（Fully Connected）作为分类器。



例如（下图所示），对于 3×H×W 的栅格图像 RGB，每次取一个 3×H′×W′的小图像块（Patch），对其进行卷积得到 C″×H″×W″。在迭代的过程中，使 Patch 滑动遍历整个图像，对每个 Patch 都进行 Convolution，把每次的卷积结果进行拼接作为 Output 送进下一个卷积器重复上述步骤。CNN 对图片进行识别，是通过观察图片局部的信息来进行分类的，只要权重选取得当，就可以实现对图像中某种特征的扫描。



【核心代码解析】：

1.Prepare Data set

- MNIST data set 的构建：构建 MINIST 实例得到训练数据集和测试数据集，这里用到了对图像的原始处理 transforms.ToTensor（如图 4）。

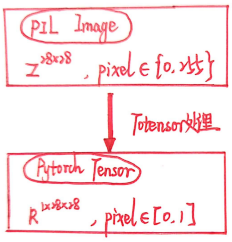
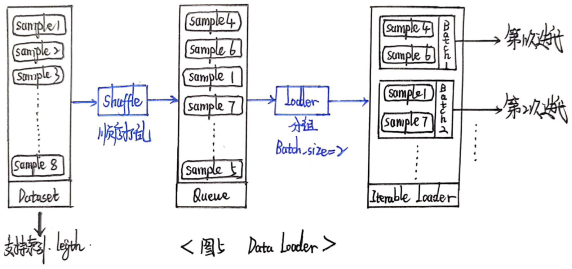


图 4

```
# MNIST dataset
train_dataset = torchvision.datasets.MNIST(root='../../data/',
                                           train=True,
                                           transform=transforms.ToTensor(),
                                           download=True)

test_dataset = torchvision.datasets.MNIST(root='../../data/',
                                          train=False,
                                          transform=transforms.ToTensor())
```

- Data loader 形成 Batch：主要是将抽象化的 Data set 加载为 Mini-Batch（如图 5 示意，Batch_size=2），能够在计算性能和速度上取得折中效果。



<图5 Data Loader>

```
# Data loader
train_loader = torch.utils.data.DataLoader(dataset=train_dataset,
                                           batch_size=batch_size,
                                           shuffle=True)

test_loader = torch.utils.data.DataLoader(dataset=test_dataset,
                                          batch_size=batch_size,
                                          shuffle=False)
```

2. Design Model

- 初始化对象：代码中一共定义了 5 层（卷积层两层、池化层两层、全连接层一层），其中 ReLU 为激活函数。代码实现的 CNN 结构如图 6 所示。对卷积计算、下采样的实现及 Padding、Stride 初始化参数的功能用图示做简要说明。

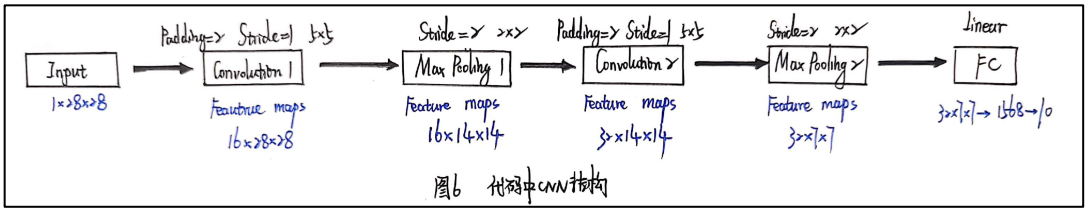
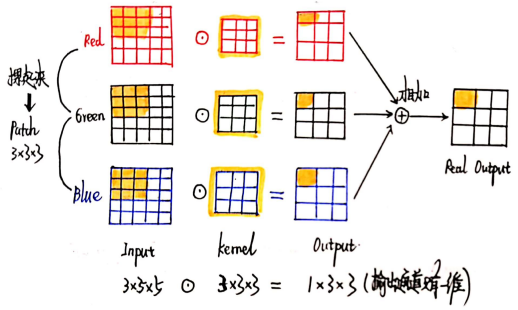
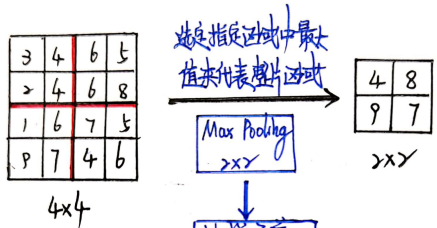


图6 代码中CNN结构





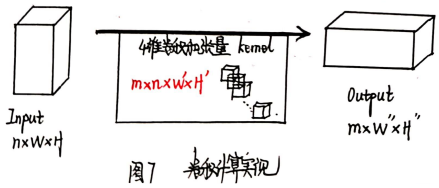
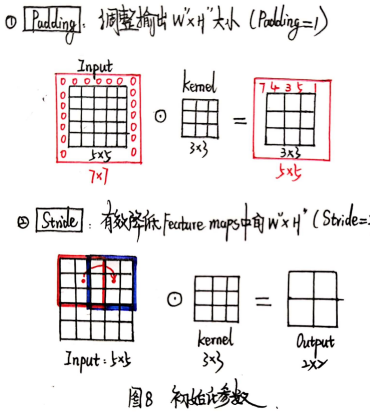


图7

图8 Max Pooling



```
def __init__(self, num_classes=10):
    super(ConvNet, self).__init__()
    self.layer1 = nn.Sequential(
        nn.Conv2d(1, 16, kernel_size=5, stride=1, padding=2),
        nn.BatchNorm2d(16),
        nn.ReLU(),
        nn.MaxPool2d(kernel_size=2, stride=2))
    self.layer2 = nn.Sequential(
        nn.Conv2d(16, 32, kernel_size=5, stride=1, padding=2),
        nn.BatchNorm2d(32),
        nn.ReLU(),
        nn.MaxPool2d(kernel_size=2, stride=2))
    self.fc = nn.Linear(7 * 7 * 32, num_classes)
```

- 前馈计算：将输入数据通过卷积层和池化层，然后进行 reshape 形状变换得到（batch，1568）并送入全连接层。（batch，10），即为最终特征提取器输出的 10 维向量 y_predict（模型预测标签结果）。

```
def forward(self, x):
    out = self.layer1(x)
    out = self.layer2(out)
    out = out.reshape(out.size(0), -1)
    out = self.fc(out)
    return out
```

3.Loss Construct and Optimizer

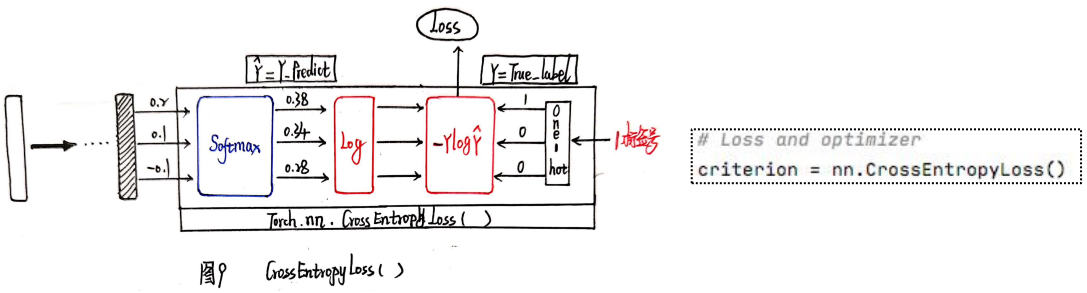
- 损失函数：这里用到的损失函数是 Cross Entropy（交叉熵），它描述了两个概率分布之间的距离，当交叉熵越小说明二者之间越接近。尽管交叉熵刻画的是两个概率分布之间的距离，但是神经网络的输出却不一定是一个概率分布。为此我们常用 Softmax 回归将神经网络前向传播得到的结果变成概率分布，然后再将其通过交叉熵计算损失。

  Softmax 常用于多分类问题中，它将多个神经元的输出，归一化到$(0, 1)$区间内：

$$P(y=i)=\frac{e^{z_i}}{\sum_{j=0}^{k-1} e^{z_j}}\ ,\ i \in \{0,\ \cdots,\ K-1\}$$

  Cross Entropy 计算损失公式如下：

$$Loss = Loss\left(\widehat{Y},\ Y\right) = -Y\log\widehat{Y}$$



```
# Loss and optimizer
criterion = nn.CrossEntropyLoss()
```

- 优化器：这里使用了比 SGD 性能更优越的优化算法 Adam。它相当于 RMSprop + Momentum。利用梯度的一阶矩估计和二阶矩估计来动态调整每个权重的学习率。优点在于经过偏置校正后，每一次迭代学习率都有一个确定范围，使得参数比较平稳。

```
# Loss and optimizer
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)
```

4.Training cycle

因为 CNN 中使用了 Mini-Batch 来提升计算速度，所以训练进行了两层循环嵌套。外层是对 Epoch 的训练轮次，内层的迭代是对一个 Batch_size 的数据而言且内层循环的次数为 Iteration。

```
for epoch in range(num_epochs):
    for i, (images, labels) in enumerate(train_loader):
```

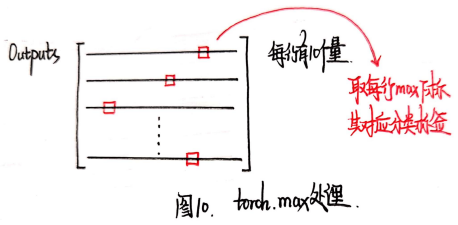
每次迭代都先进行一次前向传播，利用 model 计算得到 y_predict 预测标签，并根据 y_predict 和 y_labels 计算损失 loss；然后进行一次后向传播，利用 Adam 优化器自动更新权重大小以接近最优解。

```
# Forward pass
outputs = model(images)
loss = criterion(outputs, labels)
```

```
# Backward and optimize
optimizer.zero_grad()
loss.backward()
optimizer.step()
```

5.Test the model

最后就是利用 MINIST 测试数据集对该模型分类的准确度进行检测，该过程无需进行梯度优化。定义预测正确数为 correct，测试集总数为 total。对模型的 outputs 进行 torch.max 处理，输出样本矩阵每行的最大值及最大值对应的下标（其对应该样本的预测分类标签）。利用条件判断，更新 correct 值即可。

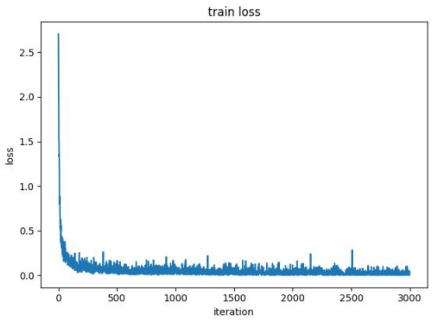



```
# Test the model
model.eval()
with torch.no_grad():
    correct = 0
    total = 0
    for images, labels in test_loader:
        outputs = model(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()
```

【代码运行结果】：结果如下，每 100 次 Iteration 对 Loss 进行打印，在源码中补充了损失曲线的绘制。我们可以看到随着训练迭代的进行，Loss 逐渐趋于收敛；测试数据集输出的 Accuracy=100* (correct / total)，准确率高达 99.03%，说明训练得到的模型性能很好，MINIST 手写数字识别功能成功实现。

```
Epoch [1/5], Step [100/600], Loss: 0.1969  Epoch [2/5], Step [100/600], Loss: 0.0450  Epoch [3/5], Step [100/600], Loss: 0.0678
Epoch [1/5], Step [200/600], Loss: 0.1251  Epoch [2/5], Step [200/600], Loss: 0.0077  Epoch [3/5], Step [200/600], Loss: 0.0488
Epoch [1/5], Step [300/600], Loss: 0.0579  Epoch [2/5], Step [300/600], Loss: 0.0636  Epoch [3/5], Step [300/600], Loss: 0.0879
Epoch [1/5], Step [400/600], Loss: 0.1055  Epoch [2/5], Step [400/600], Loss: 0.0989  Epoch [3/5], Step [400/600], Loss: 0.0233
Epoch [1/5], Step [500/600], Loss: 0.0247  Epoch [2/5], Step [500/600], Loss: 0.0199  Epoch [3/5], Step [500/600], Loss: 0.0057
Epoch [1/5], Step [600/600], Loss: 0.0375  Epoch [2/5], Step [600/600], Loss: 0.0884  Epoch [3/5], Step [600/600], Loss: 0.0847

Epoch [4/5], Step [100/600], Loss: 0.0131
Epoch [4/5], Step [200/600], Loss: 0.0160
Epoch [4/5], Step [300/600], Loss: 0.0131
Epoch [4/5], Step [400/600], Loss: 0.0032
Epoch [4/5], Step [500/600], Loss: 0.0314
Epoch [4/5], Step [600/600], Loss: 0.0082
Epoch [5/5], Step [100/600], Loss: 0.0145
Epoch [5/5], Step [200/600], Loss: 0.0008
Epoch [5/5], Step [300/600], Loss: 0.0110
Epoch [5/5], Step [400/600], Loss: 0.0051
Epoch [5/5], Step [500/600], Loss: 0.0233
Epoch [5/5], Step [600/600], Loss: 0.0446
Test Accuracy of the model on the 10000 test images: 99.03 %
```



三、实验总结与感想：

本次大作业中主要实现了线性回归和卷积神经元网络的 MNIST 数字识别。

回归问题和分类问题是监督学习的两大种类。可以概括为，回归解决的是对具体数值的预测。比如股票预测、销量预测等。这些问题需要预测的不是一个事先定义好的类别，而是一个任意实数。与回归问题相比，分类问题的输出则不再是连续值，而是离散值，用来指定其属于哪个类别。其应用包括垃圾邮件识别、人脸识别、语音识别等。它们的神经网络模型的效果及优化的目标都是通过损失函数 Loss 来定义的。

然后，对学习和实验过程中的心路历程做一个总结。因为我原先对于 Python 接触很少，处于一个“小白”的状态，对于比较复杂的机器学习神经元网络的领会还是有一定的困难。所以我选取了 GitHub 上面一个比较经典的 Pytorch tutorial 进行学习，跑了机器学习中一些经典模型的代码实例。因为对于 Machine Learning 的了解还是太少，所以我找了很多相关的课程视频进行理论知识的填充，比如 B 站上面的莫烦 python 还有 CNN 上面的代码解读。真的接触下来，对模型理解以后，感觉机器学习还是相当有趣的！在以后的学习当中，我也非常有热情去继续了解其中更深层次的、更 advanced 的机器学习模型，学会利用好开源平台上面的资源，拓宽自己的知识面、提升自己的综合学习能力!