回归 { 线性回归
分类 { 逻辑斯蒂回归

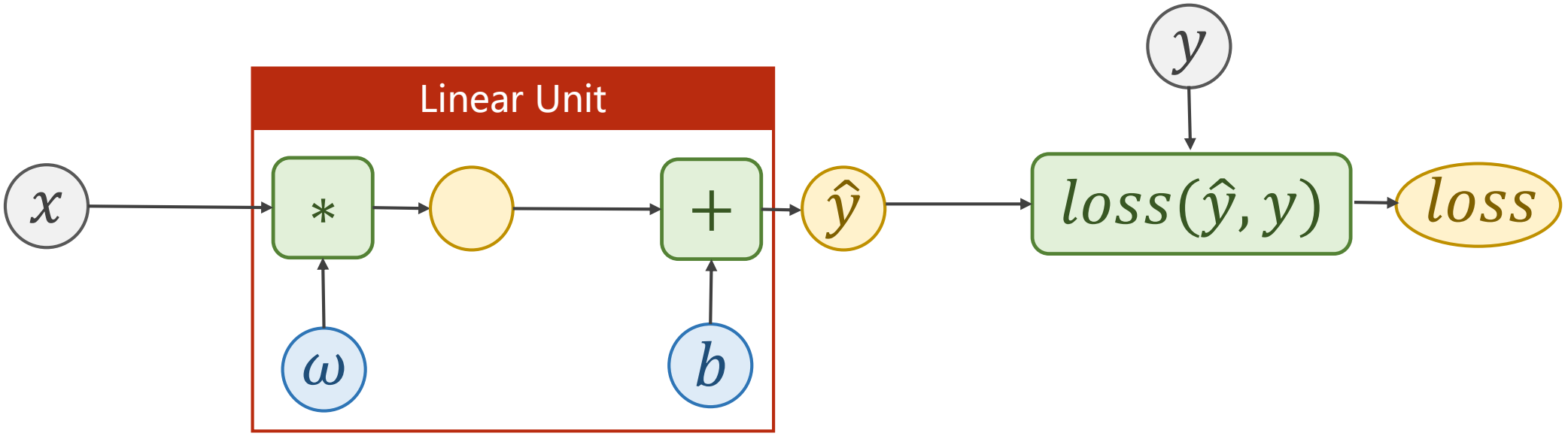
PyTorch Tutorial

06. Logistic Regression

名字是回归，但它的任务是分类

Revision - Linear Regression

线性回归



Affine Model

$$\hat{y} = x * \omega + b$$

Loss Function

$$loss = (\hat{y} - y)^2 = (x \cdot \omega - y)^2$$

Revision - Linear Regression

x (hours)	y (points)
1	2
2	4
3	6
4	?

Affine Model

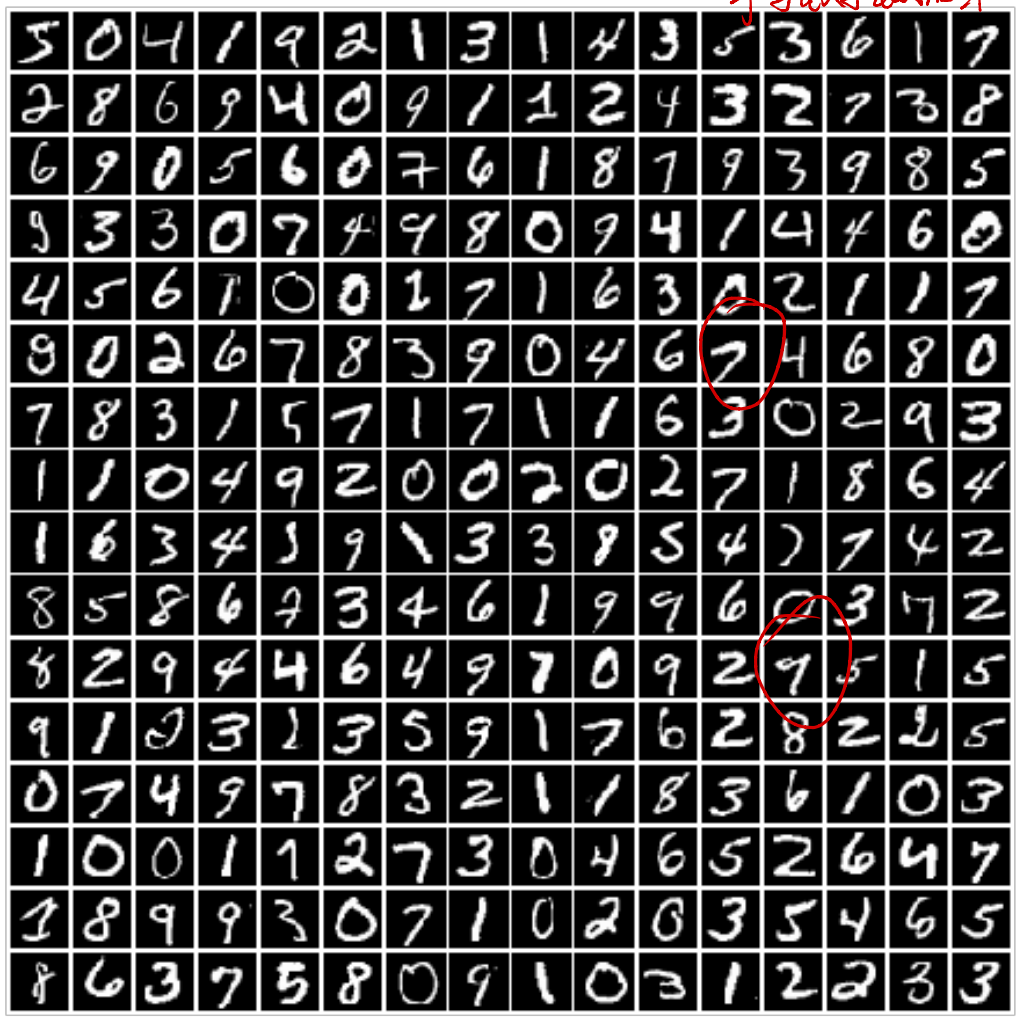
$$\hat{y} = x * \omega + b$$

Loss Function

$$loss = (\hat{y} - y)^2 = (x \cdot \omega - y)^2$$

Classification – The MNIST Dataset

手写数字数据集



The database of handwritten digits

- Training set: 60,000 examples,
- Test set: 10,000 examples.
- Classes: 10

排序

$y \in \{0, 1, 2, 3, 4, \cdots, 9\}$ 这样并不合适

```
import torchvision  可以提供比较流行的数据集
train_set = torchvision.datasets.MNIST(root='../dataset/mnist', train=True,  download=True)
test_set  = torchvision.datasets.MNIST(root='../dataset/mnist', train=False, download=True)
```

是训练集还是测试集

数据集下载路径

因为数字的大小与数字的形状无关

即 7和9相似，但数轴上7和9并不相邻

- 故，我们不再需要计算输出的值为多少，
- 对于分类问题，我们的输出是概率，而不是直接输出0-9的某个值

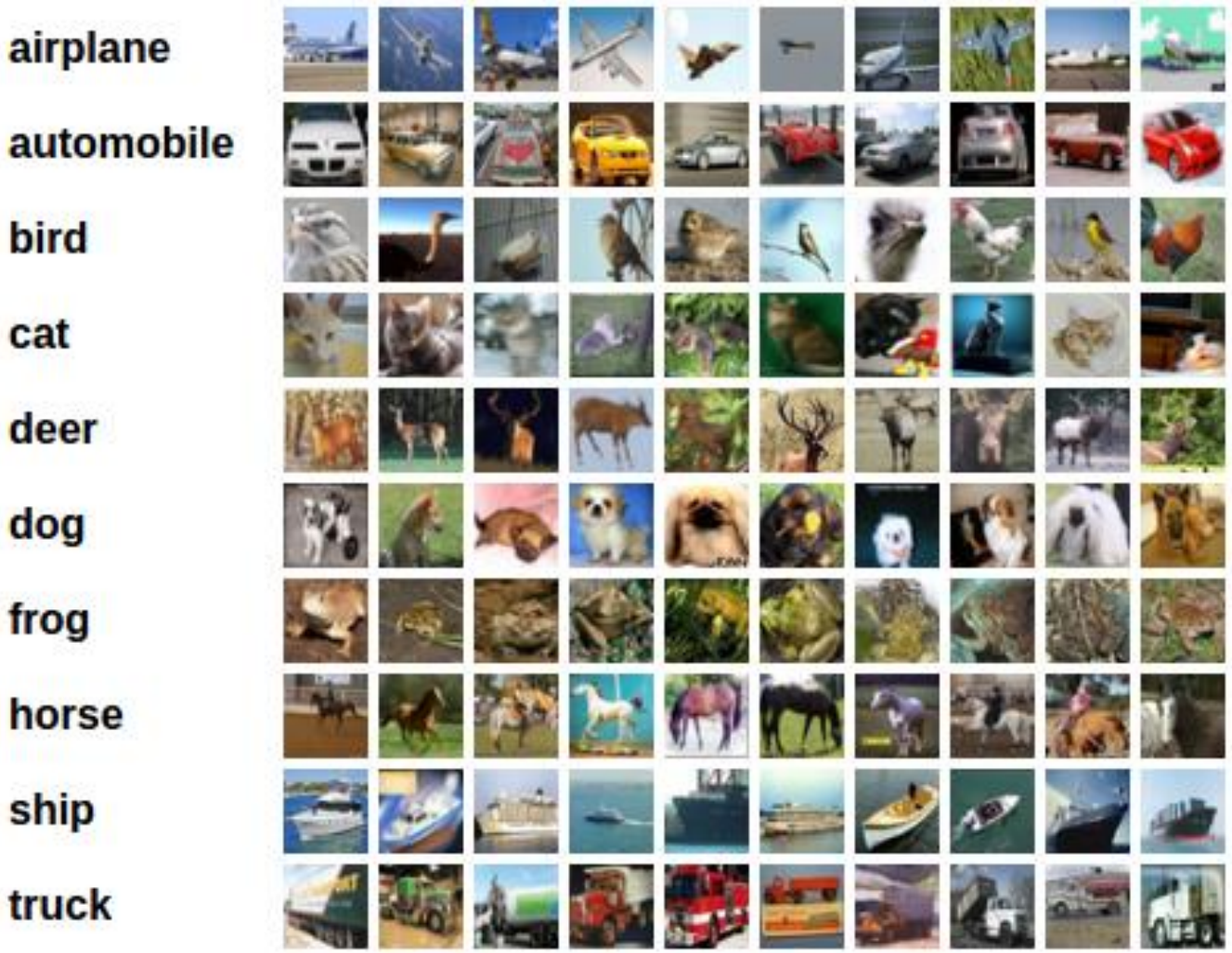
各个类别的概率，选择概率最大的即可

Classification – The CIFAR-10 dataset

32×32的小图片

- Training set: 50,000 examples,
- Test set: 10,000 examples.
- Classes: 10

```
import torchvision
train_set = torchvision.datasets.CIFAR10(…)
test_set  = torchvision.datasets.CIFAR10(…)
```



Regression vs Classification

回归 分类

回归

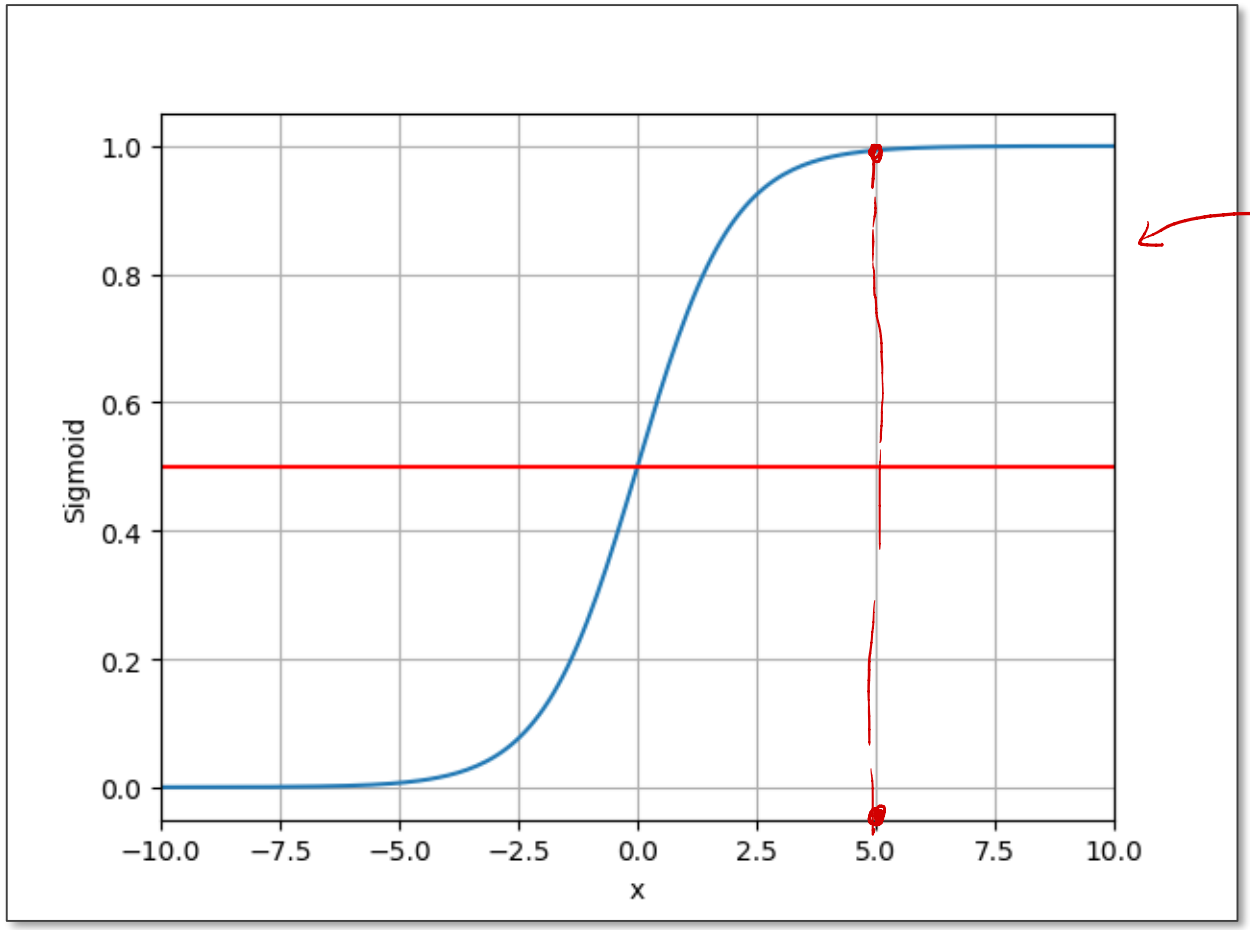
x (hours)	y (points)
1	2
2	4
3	6
4	?

分类

x (hours)	y (pass/fail)
1	0 (fail)
2	0 (fail)
3	1 (pass)
4	?

In classification, the output of model is the probability of input belongs to the exact class.

How to map: $\mathbb{R} \rightarrow [0, 1]$



Logistic Function

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

logistic 逻辑斯蒂函数

回归：$y = wx + b \in R$

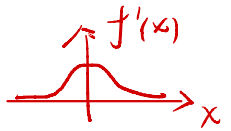
分类：$y \in [0,1]$

故需要将线性模型的输出值从R映射到[0,1]

即在线性回归后加一个 logistic 函数

https://en.wikipedia.org/wiki/Logistic_function

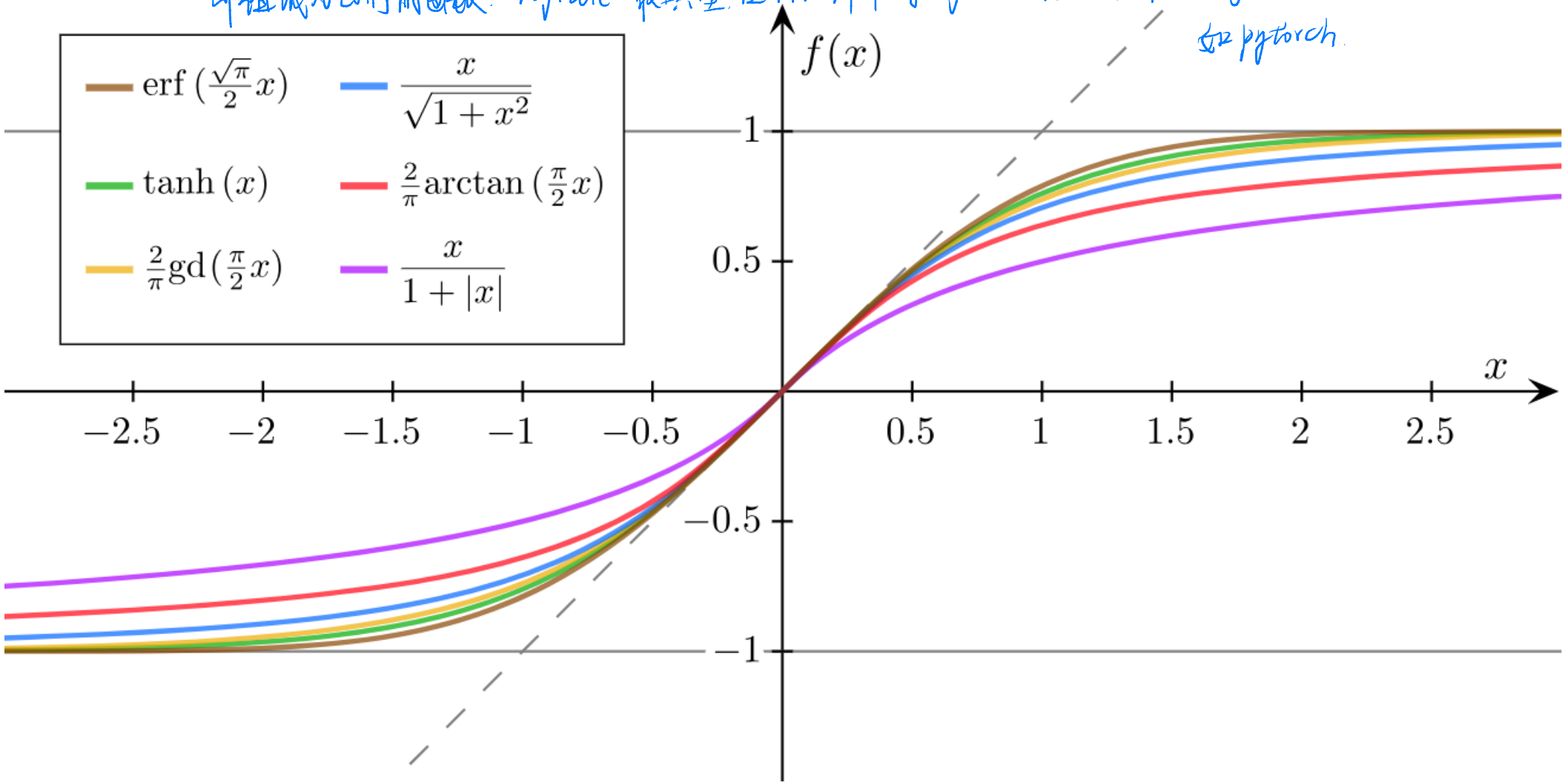
特点：超过某个域值后，增长非常慢（导数趋于0）—— 饱和函数



类似正态分布

Sigmoid functions

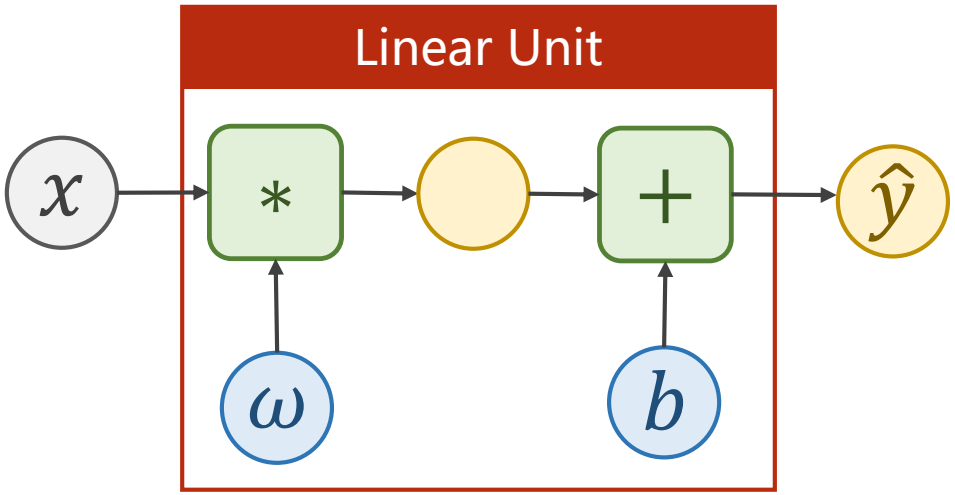
即值域为[0,1]的函数. logistic 最典型, 但有的库中将 logistic 函数直接称为 sigmoid 函数. 如 pytorch.



Logistic Regression Model

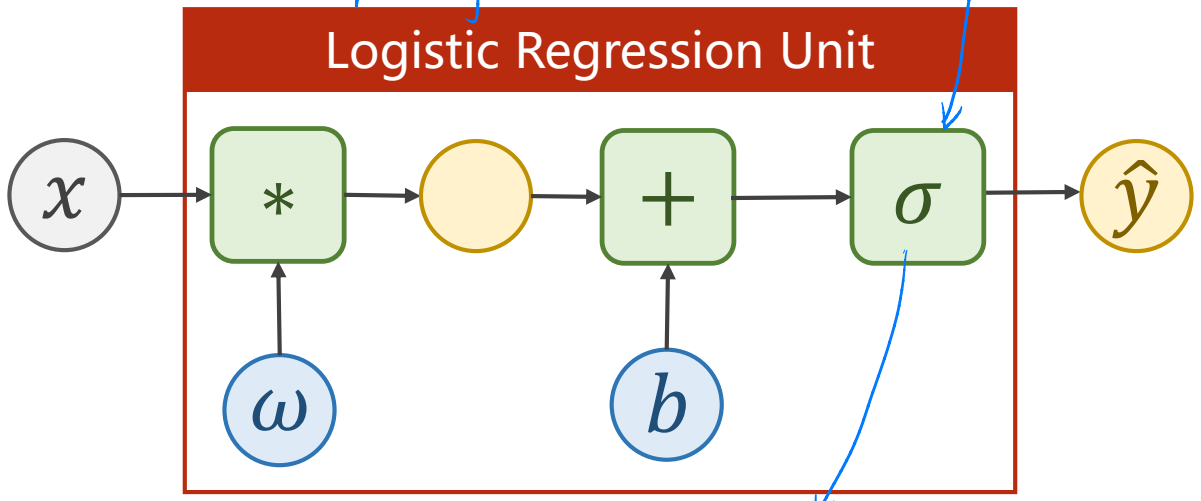
Affine Model

$$\hat{y} = x * \omega + b$$



Logistic Regression Model

$$\hat{y} = \sigma(x * \omega + b)$$



Loss function for Binary Classification

Loss Function for Linear Regression

MSE: $$loss = (\hat{y} - y)^2 = (x \cdot \omega - y)^2$$

数轴上的距离，不再使用概率分布了，

我们现在输出不再是数轴，而是一个分布，所以现在需要计算分布之间的差异

2分类问题，数据集中的y只有两个可能：0和1（属于此类or不属于此类）

Loss Function for Binary Classification

$$loss = -(y \log \hat{y} + (1 - y) \log(1 - \hat{y}))$$

若 $y=1$，$loss = -y\log\hat{y}$，此时 $\hat{y}$ 越趋近于1，loss越小。

若 $y=0$，$loss = -(1-y)\log(1-\hat{y})$，此时 $\hat{y}$ 越趋近于0，loss越小。

计算分布的差异 { KL散度
交叉熵 cross-entropy

交叉熵，如2对于分布

$$\begin{cases} P_D(X=1)=0.2 \\ P_D(X=2)=0.3 \\ P_D(X=3)=0.5 \end{cases} \text{和} \begin{cases} P_T(X=1)=0.3 \\ P_T(X=2)=0.4 \\ P_T(X=3)=0.3 \end{cases}$$

其交叉熵为：$\sum_i P_D(X=i)\cdot \ln P_T(X=i)$，可以用此来衡量这两个分布的差异大小

交叉熵越大，差异越小

Mini-Batch Loss function for Binary Classification

Loss Function for Binary Classification

$$loss = -(y \log \hat{y} + (1 - y) \log(1 - \hat{y}))$$

Mini-Batch Loss Function for Binary Classification

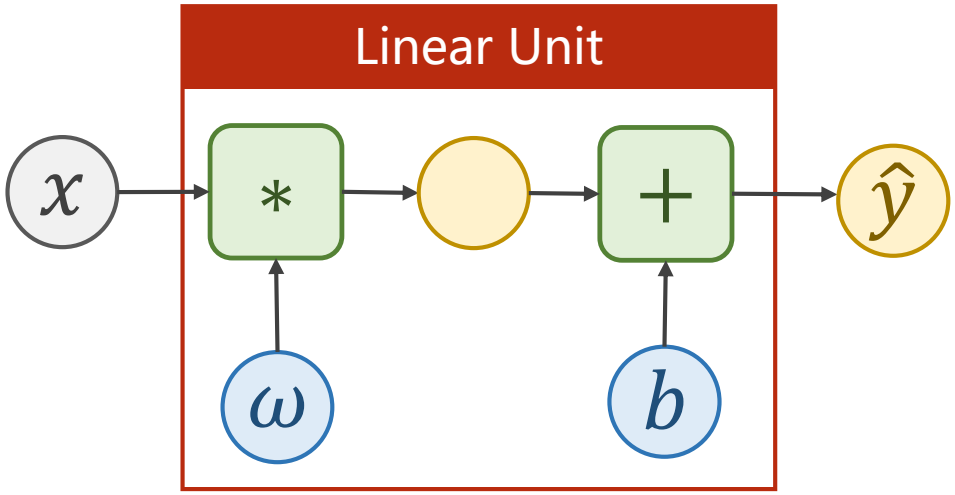
$$loss = -\frac{1}{N} \sum_{n=1}^{N} y_n \log \hat{y}_n + (1 - y_n) \log(1 - \hat{y}_n)$$

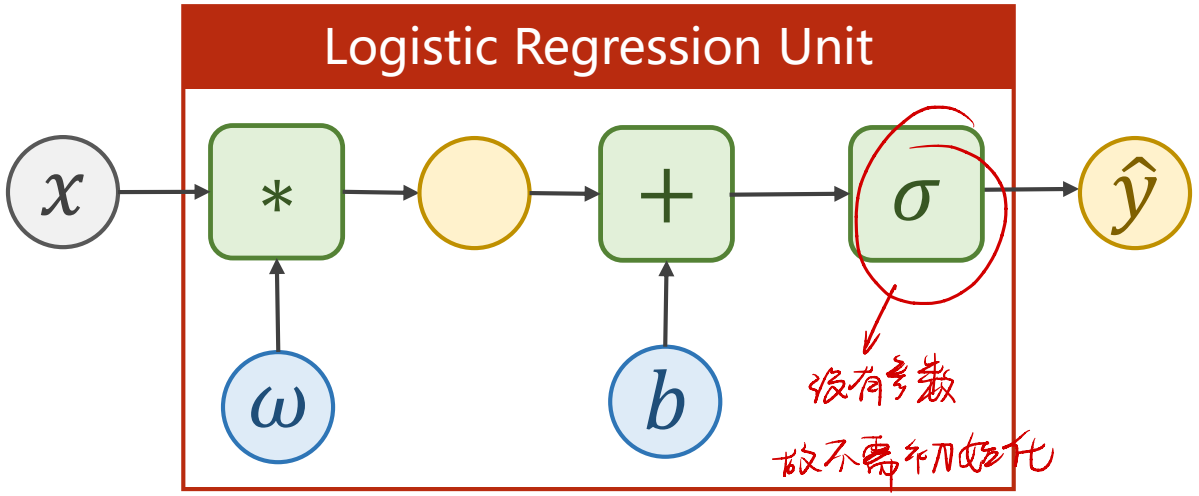
对3个样本，求和求平均

y	$\hat{y}$	BCE Loss
1	0.2	**1.6094**
1	0.8	**0.2231**
0	0.3	**0.3567**
0	0.7	**1.2040**
Mini-Batch Loss		**0.8483**

y和y^越接近，损失越小

Implementation of Logistic Regression






```python
class LinearModel(torch.nn.Module):
    def __init__(self):
        super(LinearModel, self).__init__()
        self.linear = torch.nn.Linear(1, 1)

    def forward(self, x):
        y_pred = self.linear(x)
        return y_pred
```

模型构造上的变化：

```python
import torch.nn.functional as F

class LogisticRegressionModel(torch.nn.Module):
    def __init__(self):
        super(LogisticRegressionModel, self).__init__()
        self.linear = torch.nn.Linear(1, 1)

    def forward(self, x):
        y_pred = F.sigmoid(self.linear(x))
        return y_pred
```

Implementation of Logistic Regression

Mini-Batch Loss Function for Binary Classification

$$loss = -\frac{1}{N}\sum_{n=1}^{N} y_n \log \hat{y}_n + (1 - y_n)\log(1 - \hat{y}_n)$$

变化2：损失由MSE变为BCE（BCE — cross-entropy，交叉熵）

二分类交叉熵

```
criterion = torch.nn.BCELoss(size_average=False)
```

是否求均值，（是否乘N）

会影响学习率，因为求均值后，$\frac{\partial loss}{\partial w}$中也会乘N，

会变小，故 lr 会相应变大(?) lr·$\frac{\partial loss}{\partial w}$

Implementation of Logistic Regression

```
x_data = torch.Tensor([[1.0], [2.0], [3.0]])
y_data = torch.Tensor([[0], [0], [1]])
#-----------------------------------------------------#
class LogisticRegressionModel(torch.nn.Module):
    def __init__(self):
        super(LogisticRegressionModel, self).__init__()
        self.linear = torch.nn.Linear(1, 1)

    def forward(self, x):
        y_pred = F.sigmoid(self.linear(x))
        return y_pred
model = LogisticRegressionModel()
#-----------------------------------------------------#
criterion = torch.nn.BCELoss(size_average=False)
optimizer = torch.optim.SGD(model.parameters(), lr=0.01)
#-----------------------------------------------------#
for epoch in range(1000):
    y_pred = model(x_data)
    loss = criterion(y_pred, y_data)
    print(epoch, loss.item())

    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
```

1. **Prepare dataset**
   we shall talk about this later
2. **Design model using Class**
   inherit from nn.Module
3. **Construct loss and optimizer**
   using PyTorch API
4. **Training cycle**
   forward, backward, update

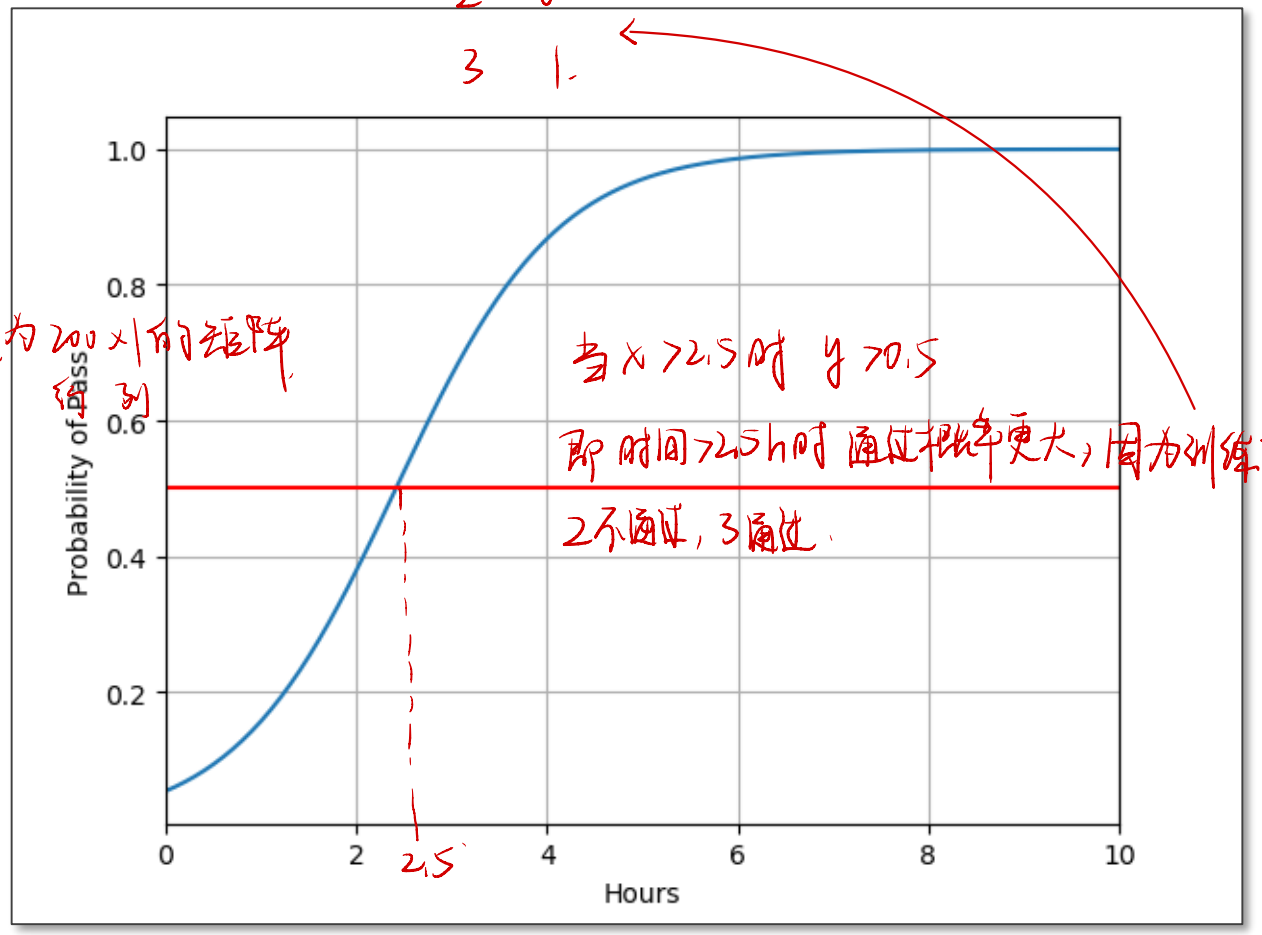
Result of Logistic Regression

测试模型.

```
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(0, 10, 200)
x_t = torch.Tensor(x).view((200, 1)) // 将x变为200×1的矩阵
y_t = model(x_t) // y_t是模型输出值
y = y_t.data.numpy() // 将 y_t.data 转为数组
plt.plot(x, y)
plt.plot([0, 10], [0.5, 0.5], c='r')
plt.xlabel('Hours')
plt.ylabel('Probability of Pass')
plt.grid()
plt.show()
```

训练数据 1 0
2 0
3 1.



PyTorch Tutorial

06. Logistic Regression