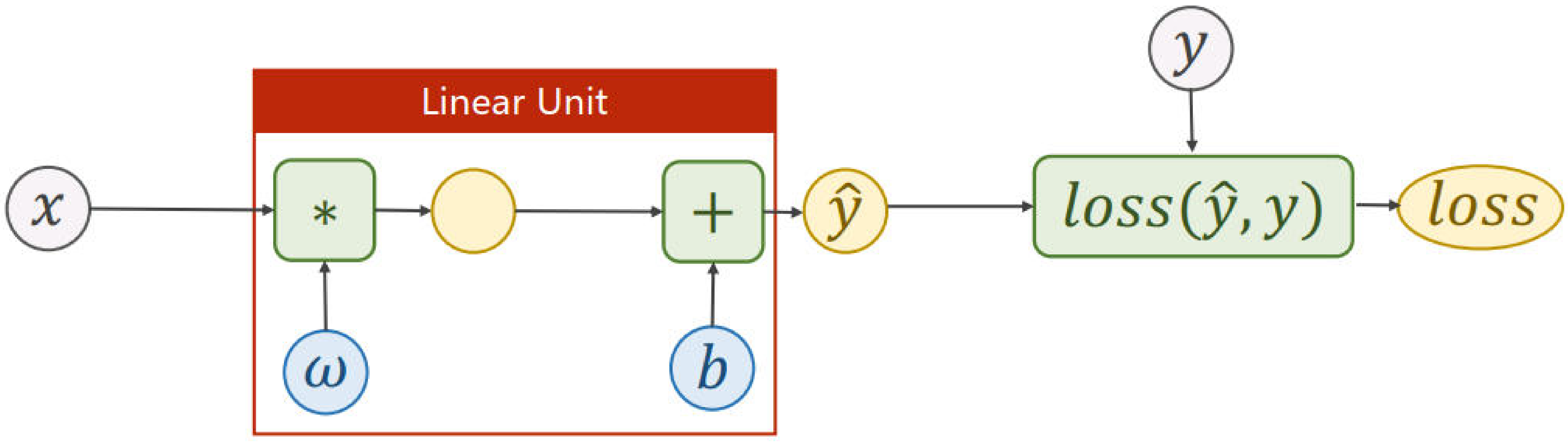
PyTorch Tutorial

06. Logistic Regression



Affine Model

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

Loss Function

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

Revision - Linear Regression

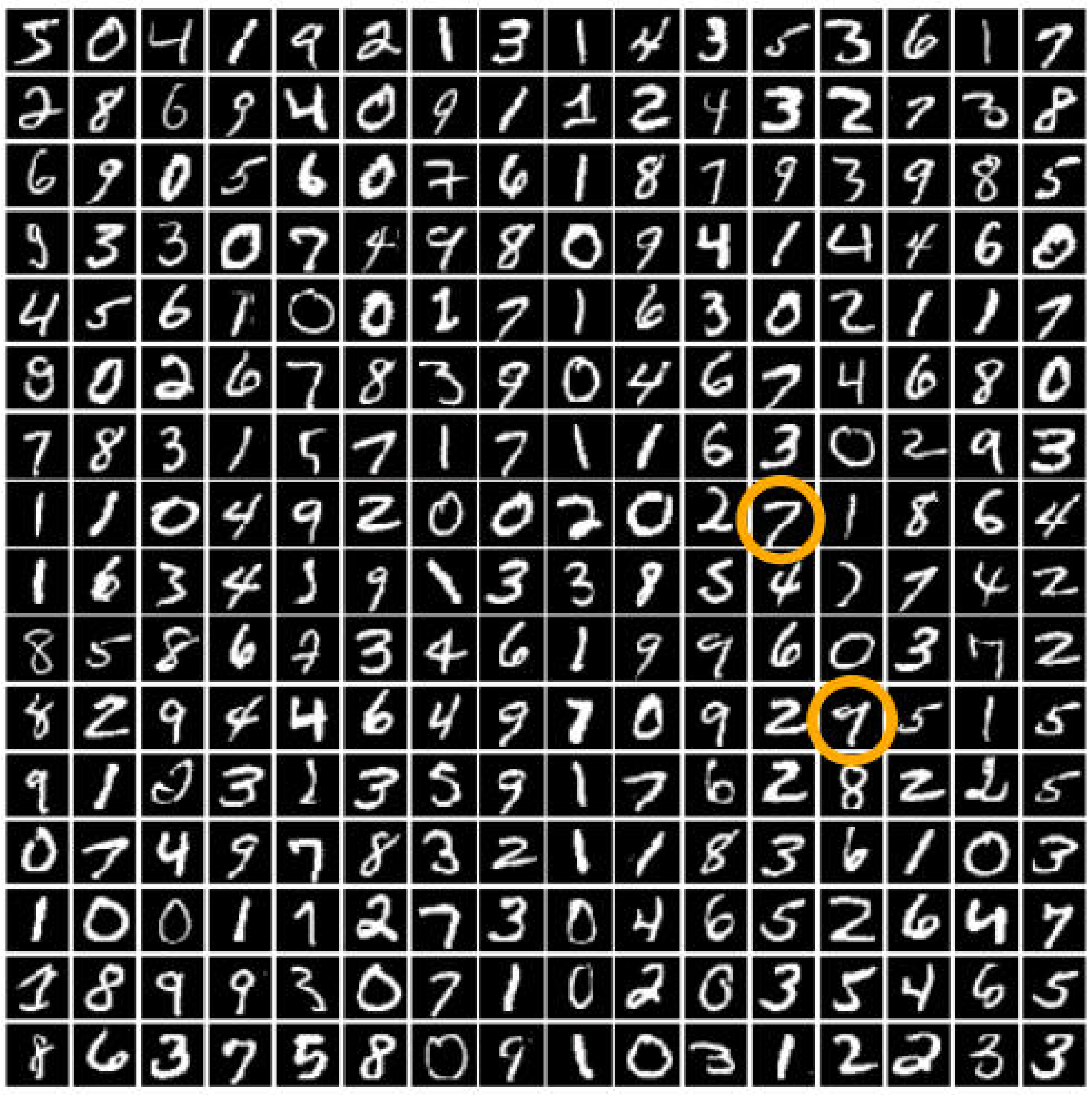
x (hours)	y (points)
1	2
2	4
3	6
4	?

Affine Model

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

Loss Function

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



The database of handwritten digits

- Training set: 60,000 examples,
- Test set: 10,000 examples.
- Classes: 10 $y \in \{0,1,2,3,4,5,6,7,8,9\}$ → 离散空间

分类问题：不可用线性回归的模型简单输出0-9

```python
import torchvision
train_set = torchvision.datasets.MNIST(root='../dataset/mnist', train=True,  download=True)
test_set  = torchvision.datasets.MNIST(root='../dataset/mnist', train=False, download=True)
```

不是做数字大小的比较 $7<8<9$ ×

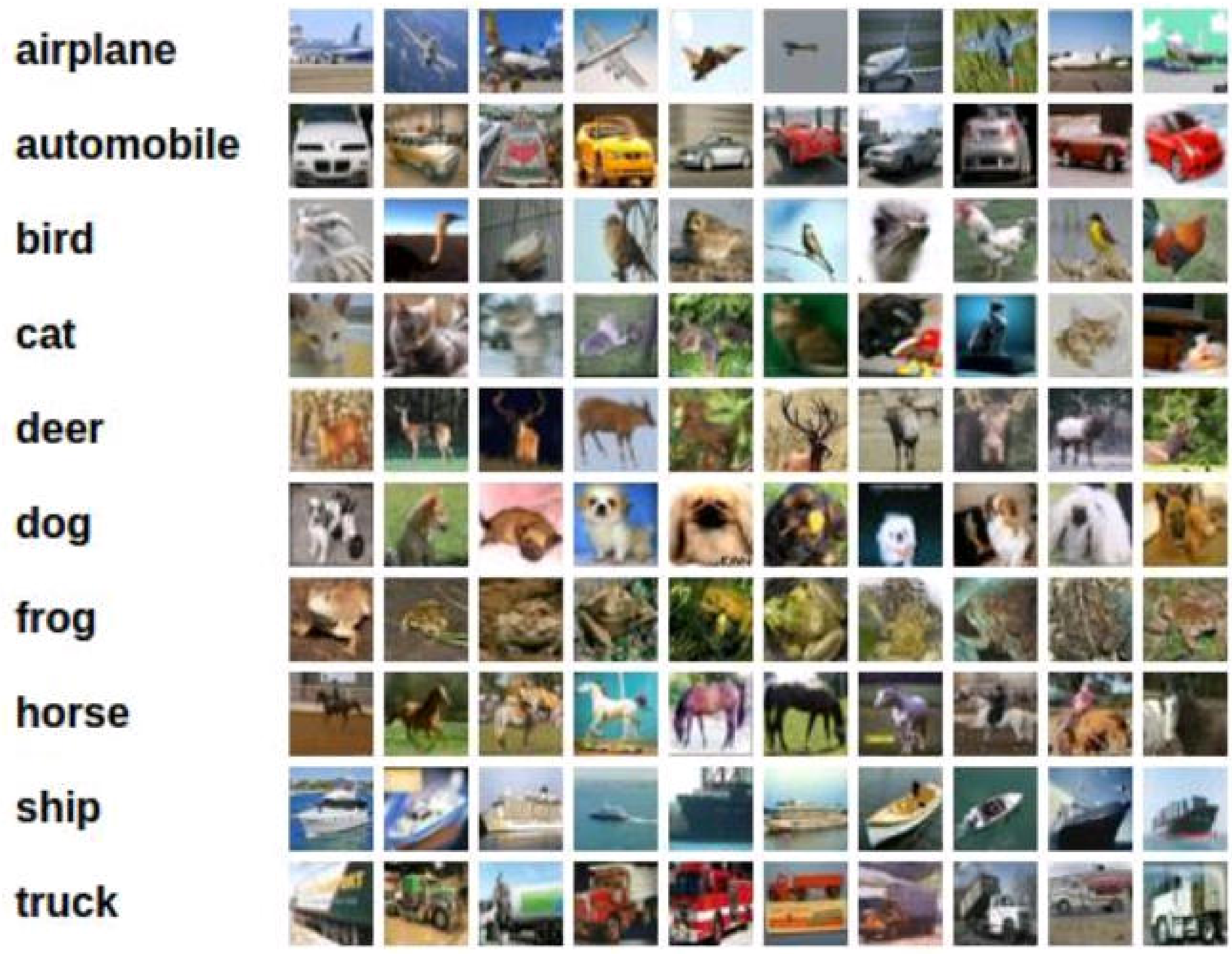
而是所属类别判断 $P(0)\ P(1) \cdots P(9)$ ✓

概率 max ✓

Classification – The CIFAR-10 dataset

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

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



Regression vs Classification

$P(\hat{y}=1) + P(\hat{y}=0) = 1$

实数

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

1 / 0

能否通过、二分类 pass / fail

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

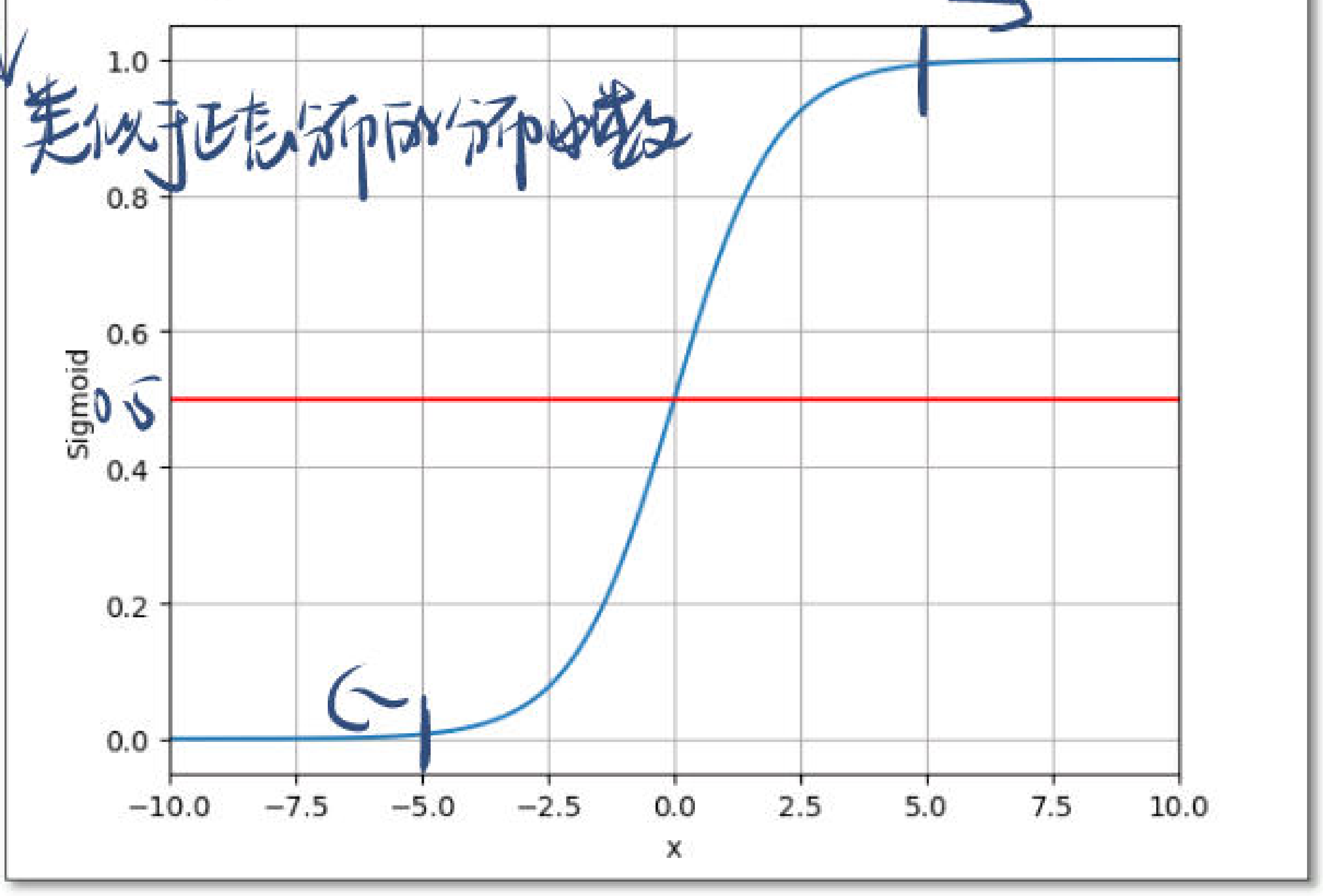
In classification, the output of model is the probability of input belongs to the exact class.

分类问题：输出 = 输入所属类别的概率 (0~1)

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

饱和函数

类似于正态分布的分布函数



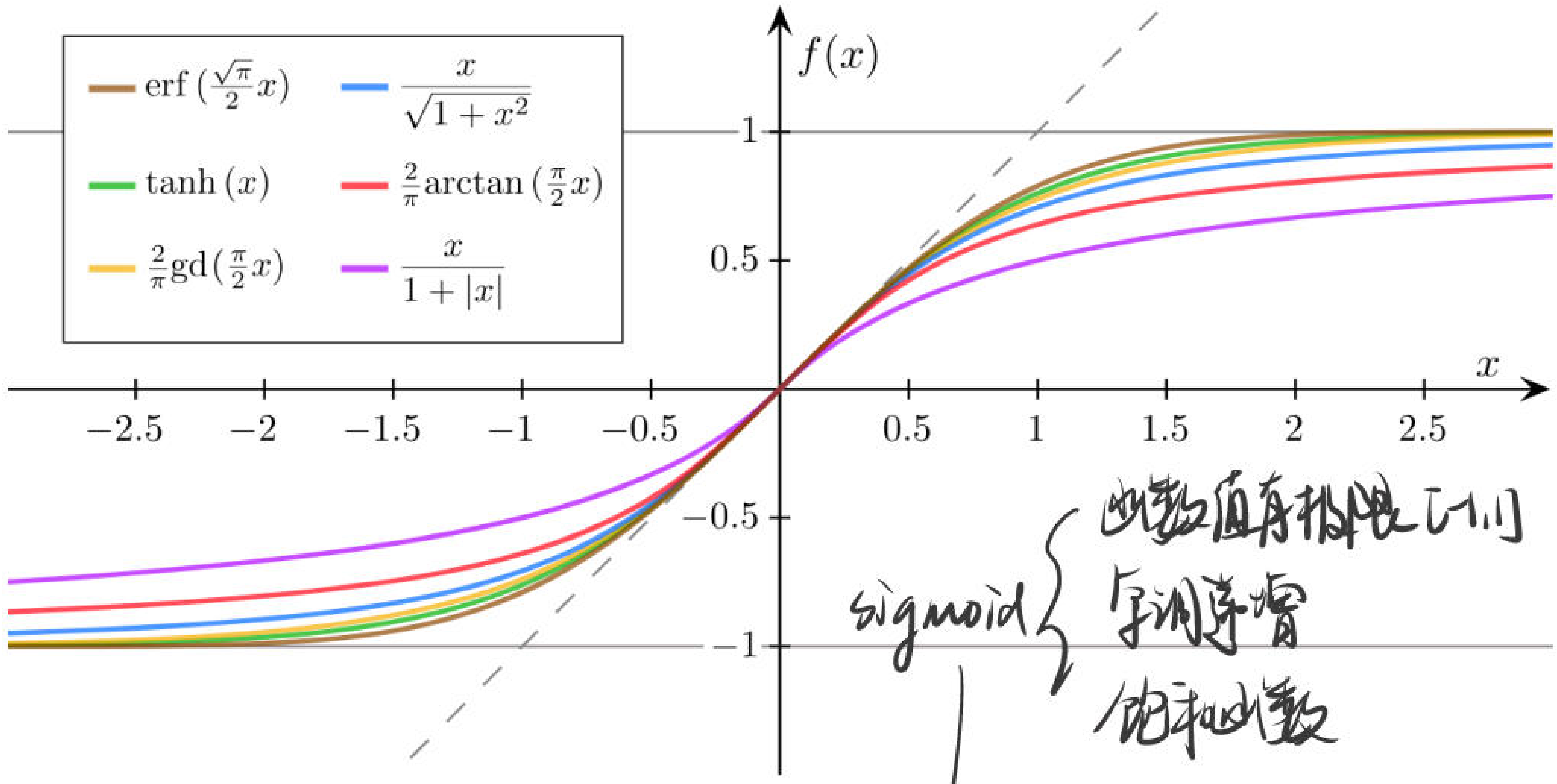
Logistic Function

$$\sigma(x) = \frac{1}{1+e^{-x}} \begin{cases} x \to +\infty & \sigma(x) \to 1 \\ x \to 0 & \sigma(x) \to 0.5 \\ x \to -\infty & \sigma(x) \to 0 \end{cases}$$

$$\begin{cases} \hat{y} = wx + b \in \mathbb{R} \\ \hat{y} \in [0,1] \end{cases}$$

σ 实数映射为概率

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



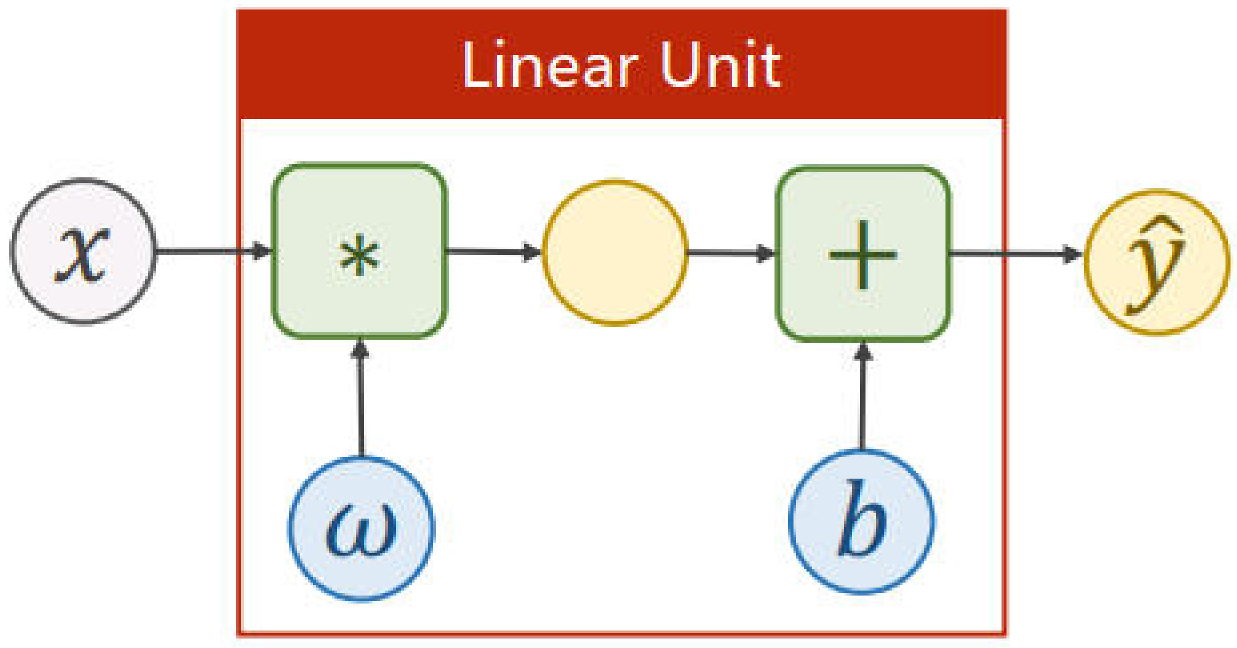
sigmoid {
函数值有极限 [-1,1]
单调递增
饱和函数

logistic {
函数值 [0,1]
是最出名的sigmoid函数
所以有时直接被叫做sigmoid

Logistic Regression Model

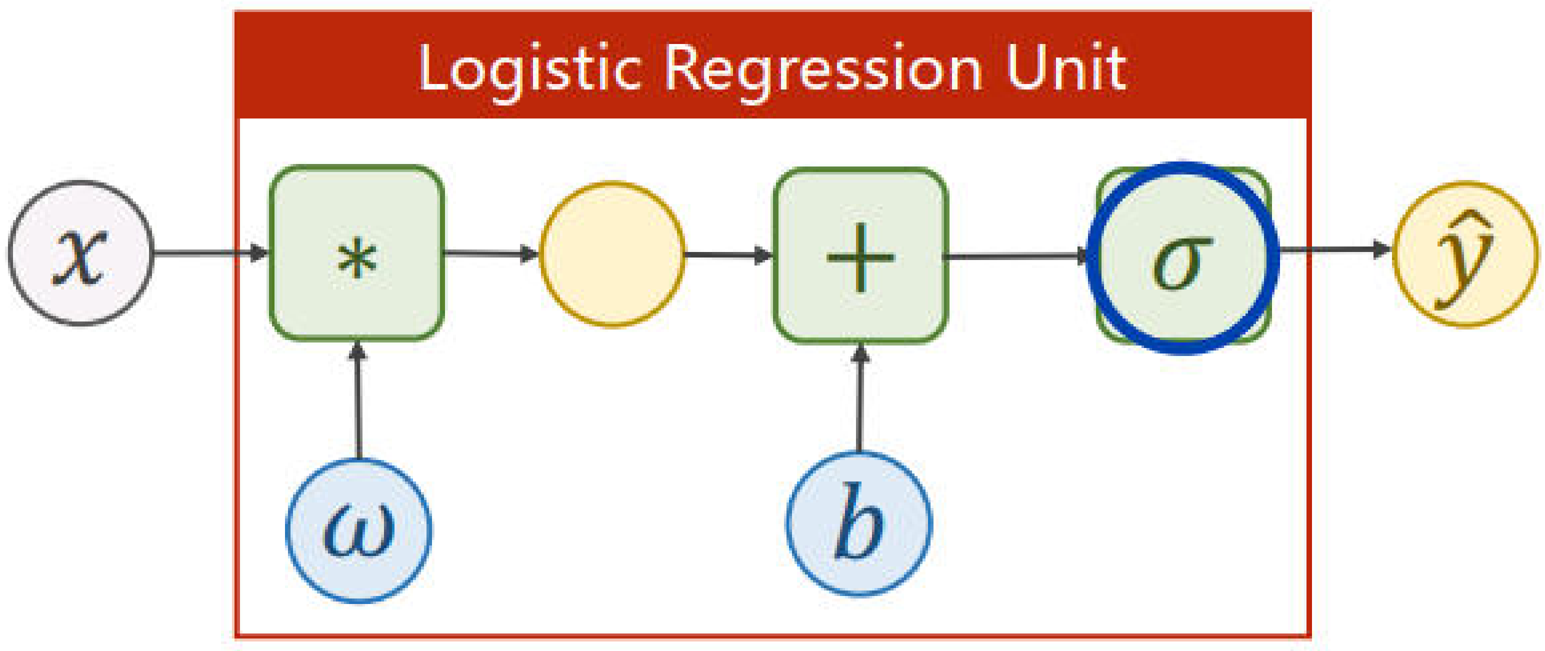
Affine Model

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



Logistic Regression Model

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

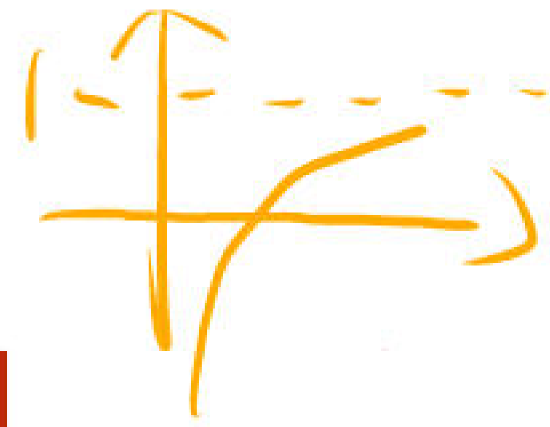


Loss Function for Linear Regression
$loss = (\hat{y} - y)^2 = (x \cdot \omega - y)^2$

$\hat{y} = P(class=1)$

$1-\hat{y} = P(class=0)$

$y=1$ $loss = -y\log\hat{y}$



BCE

Loss Function for Binary Classification
$loss = -(y \log \hat{y} + (1-y)\log(1-\hat{y}))$

$y=0$ $loss = -\log(1-\hat{y})$ 0

$y=0$时 $y = P(class=1) = 0$, $1-y = P(class=0) = 1$

$y=1$时 $y = P(class=1) = 1$, $1-y = P(class=0) = 0$

Cross-entropy：计算 $\sum_{x_i} P_{D_1}(x_i) \ln P_{D_2}(x_i)$

$P_D(x=1) = 0.2$ $P_T(x=1) = 0.3$ 表示两个分布间的差异：越大越好

$P_D(x=2) = 0.3$ $P_T(x=2) = 0.4$

$P_D(x=3) = 0.5$ $P_T(x=3) = 0.3$

Mini-Batch Loss function for Binary Classification

BCE Loss Function for Binary Classification

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

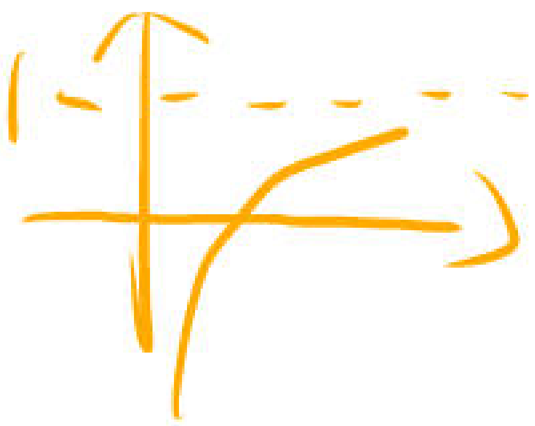
Mini-Batch Loss Function for Binary Classification

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

Mini-Batch BCE (求均值)

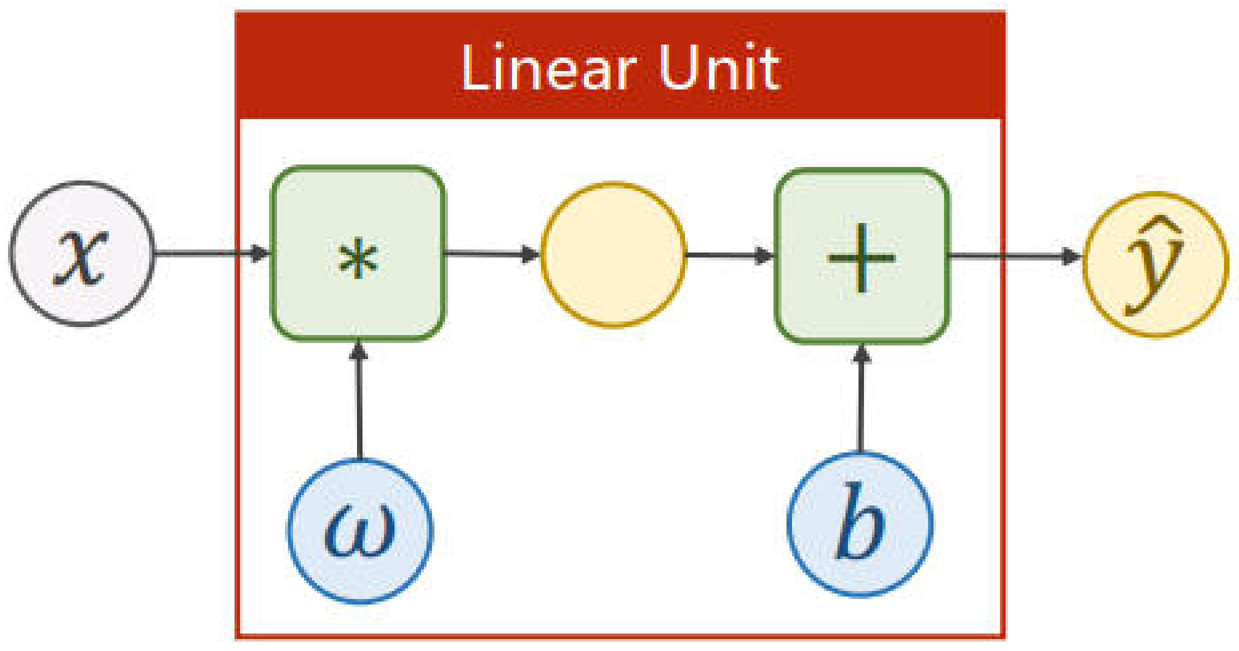
y	$\hat{y}$	BCE Loss
1	0.2	**1.6094**
1	0.8 ↑	**0.2231** ↓
0	0.3 ↓	**0.3567** ↓
0	0.7	**1.2040**
Mini-Batch Loss		**0.8483**

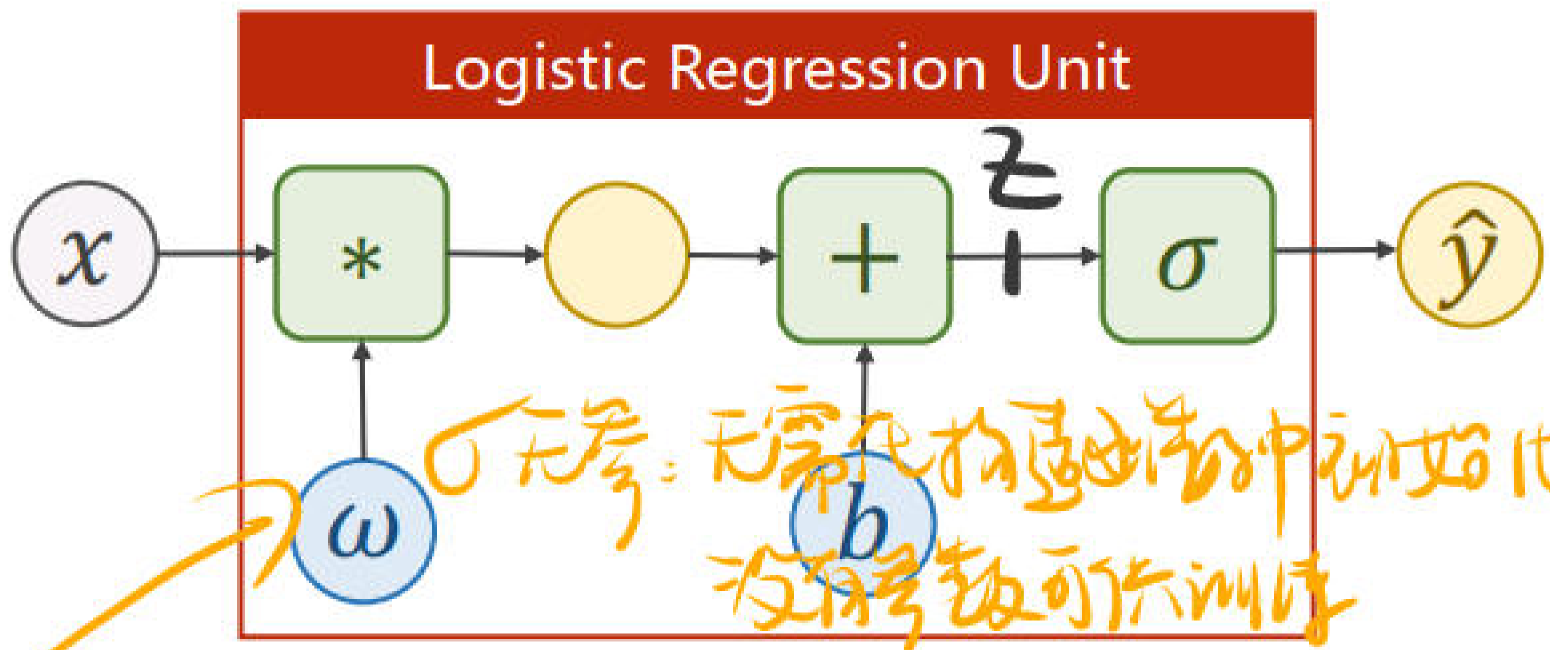
$y=1$ ↓ $loss = -y\log \hat{y}$ ↑ 1



$y=0$ ↓ $loss = -\log(1-\hat{y})$ ↓ 0

Implementation of Logistic Regression





σ无参：无需在构造函数中初始化
没有参数可供训练

```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
```

基本无区别

```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
```

$\sigma(z)$ 先做线性变换→z

Mini-Batch Loss Function for Binary Classification

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

MSE

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

交叉熵 cross-entropy

是否求均值
设置为true，损失值变小
会影响到如何选择学习率

Implementation of Logistic Regression

```
x_data = torch.Tensor([[1.0], [2.0], [3.0]])
y_data = torch.Tensor([[0], [0], [1]]) 实际数值→1集
#------------------------------------------------------#
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

```
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):  Logistic非线性变换
        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()
```

2 Design model using Class

inherit from nn.Module

Implementation of Logistic Regression

```python
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()
```

损失函数 二分类交叉熵

3 Construct loss and optimizer

using PyTorch API

```python
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()
```

4

Training cycle

forward, backward, update

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

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

x = np.linspace(0, 10, 200)
x_t = torch.Tensor(x).view((200, 1))
y_t = model(x_t)
y = y_t.data.numpy()
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()
```

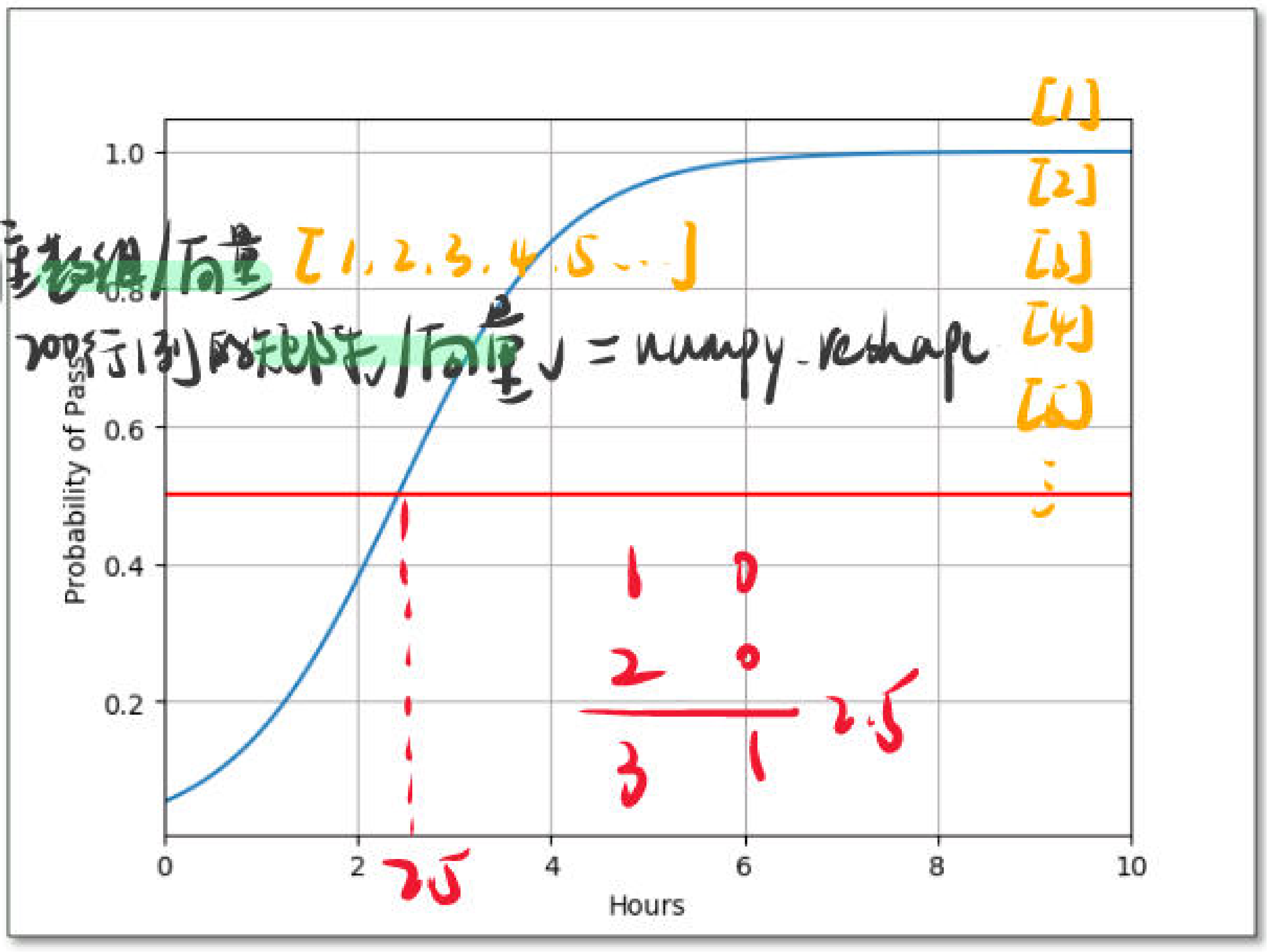
数组 reshape Tensor 数组

0-10h采200个点、1维数组/向量 [1, 2, 3, 4, 5 ...]

改变格式 200行1列的矩阵/向量 = numpy.reshape

测试

获得numpy的1维数组



PyTorch Tutorial

06. Logistic Regression