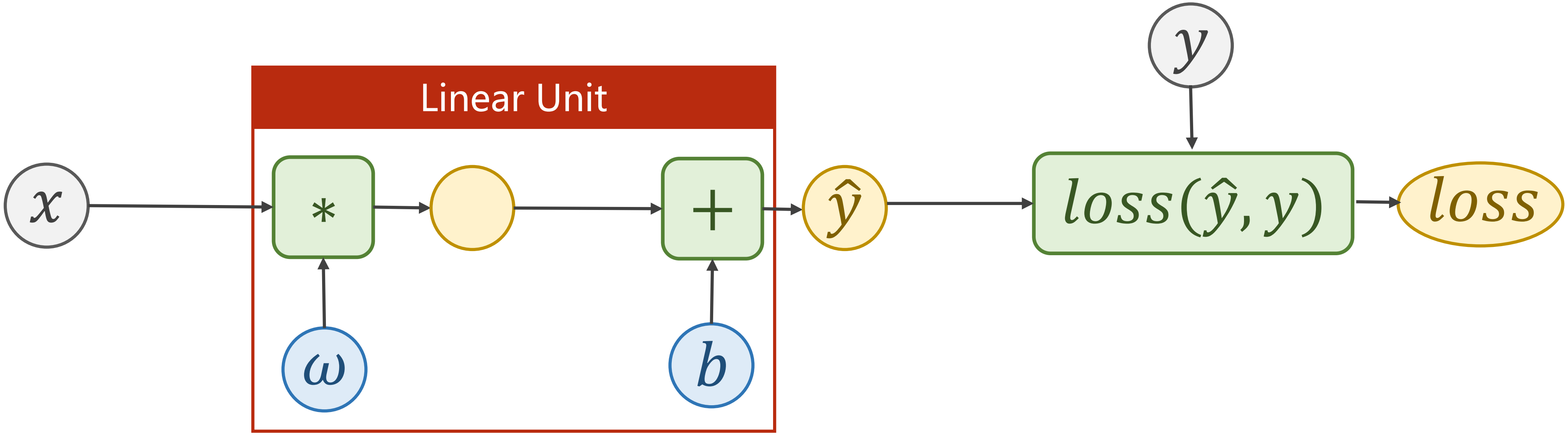
PyTorch Tutorial

06. Logistic Regression

Revision - Linear 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	?

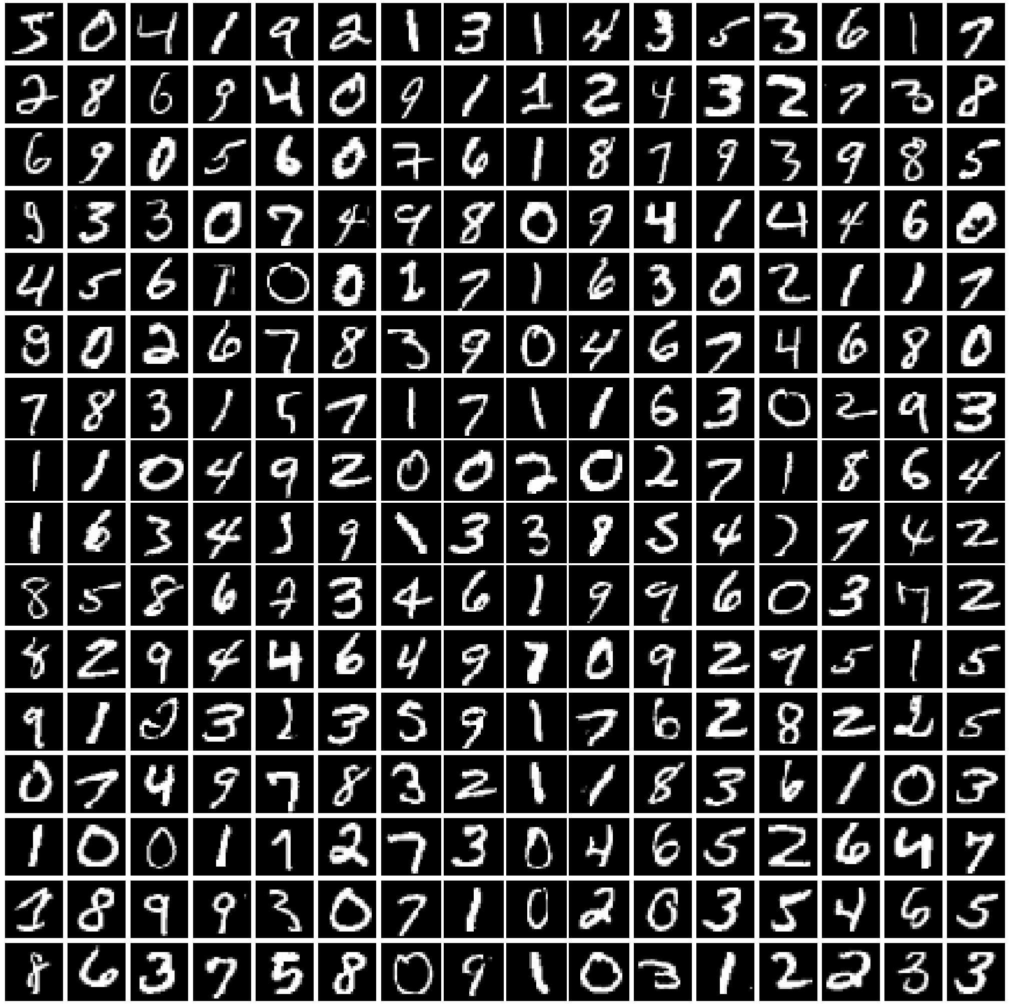
Affine Model

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

Loss Function

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

Classification – The MNIST Dataset



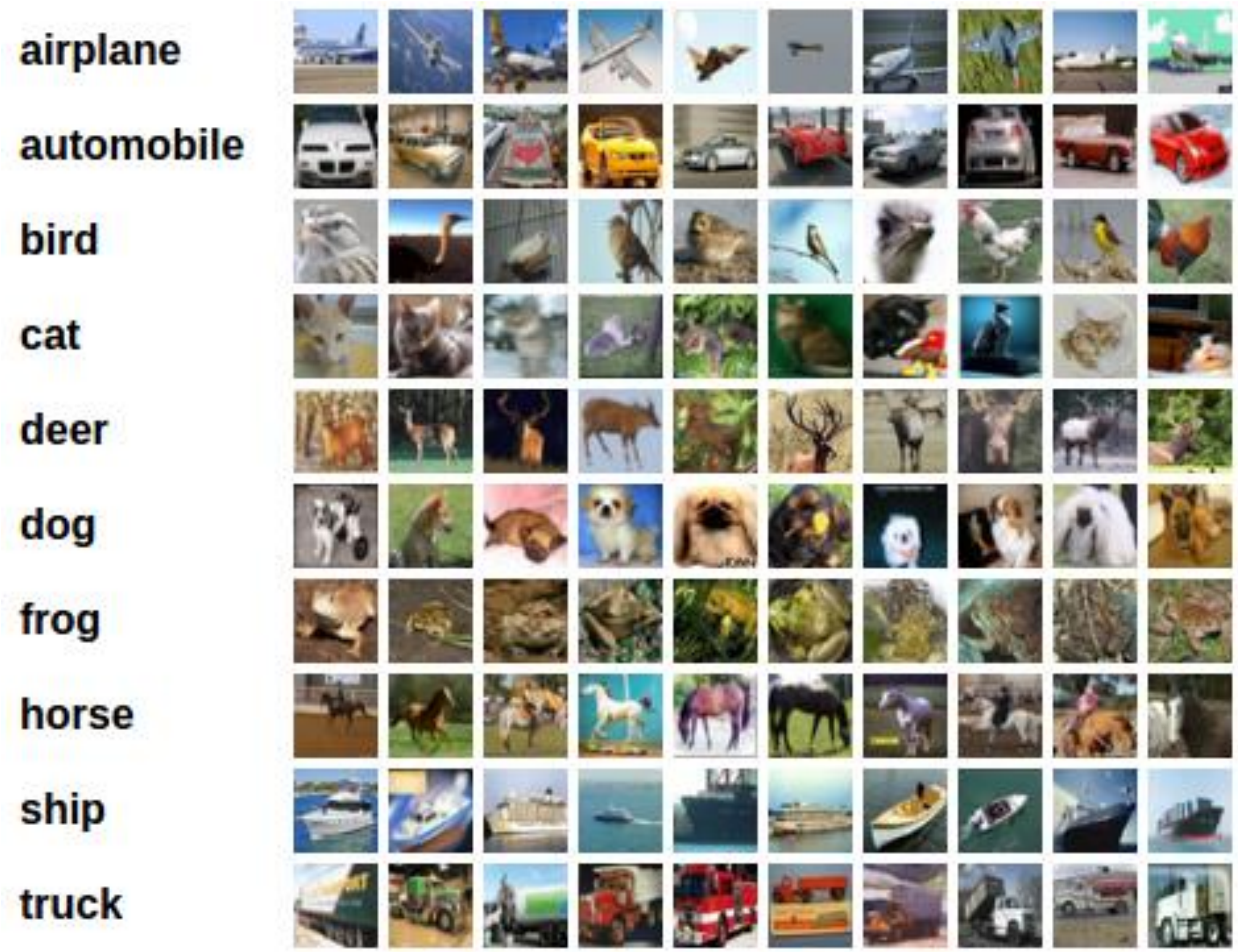
The database of handwritten digits

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

```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)
```

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

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



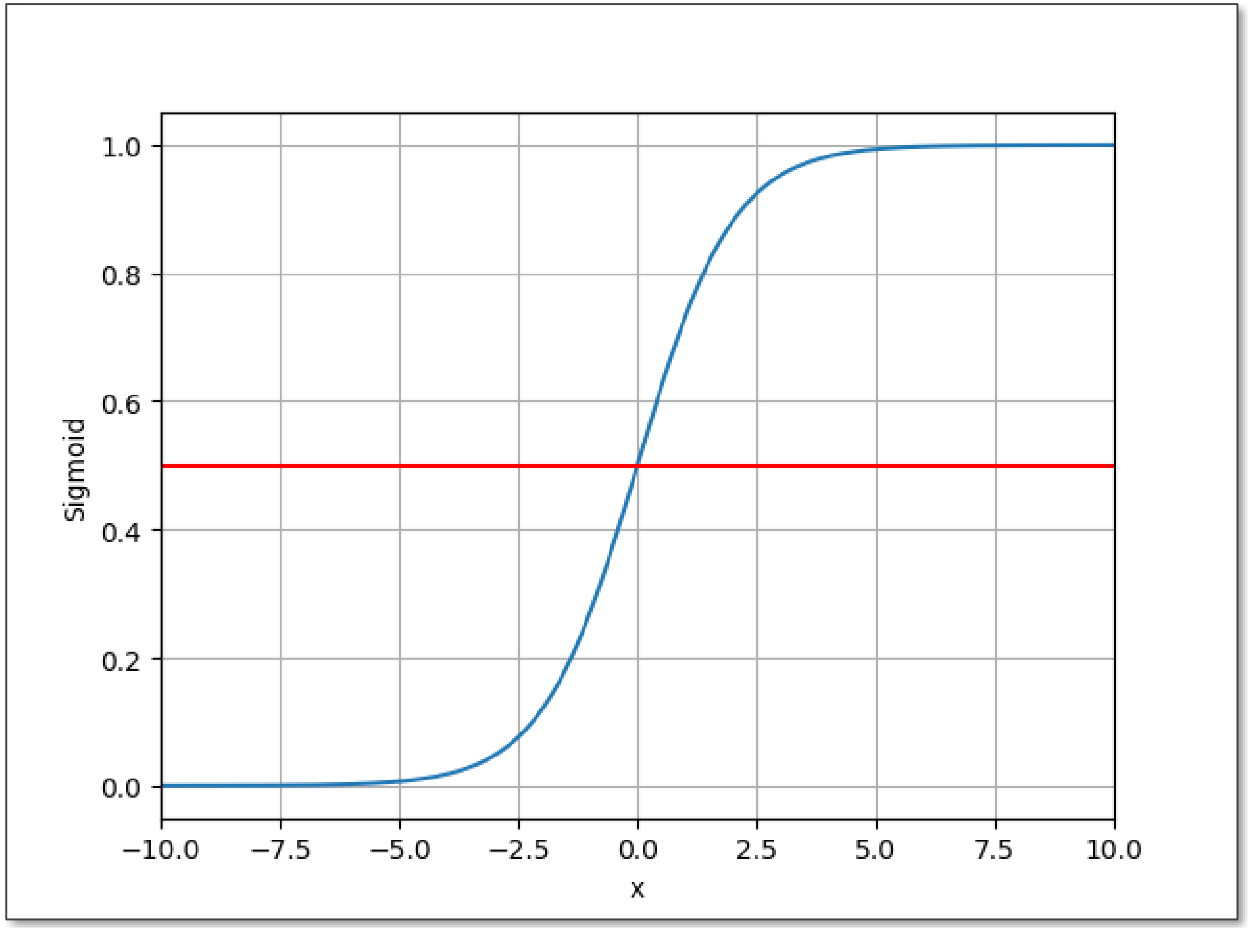
Regression vs Classification

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

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

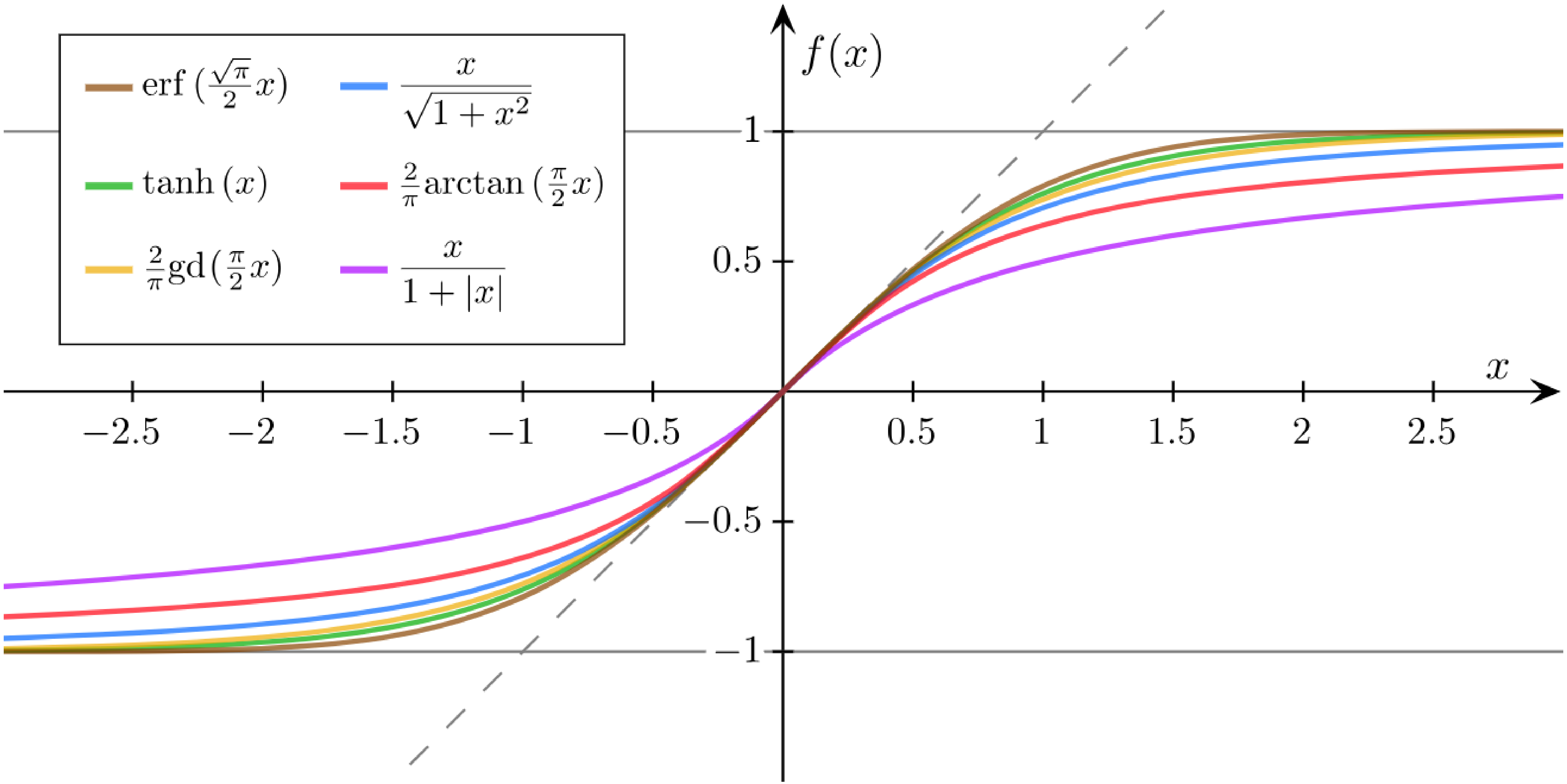


Logistic Function

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

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

Sigmoid functions



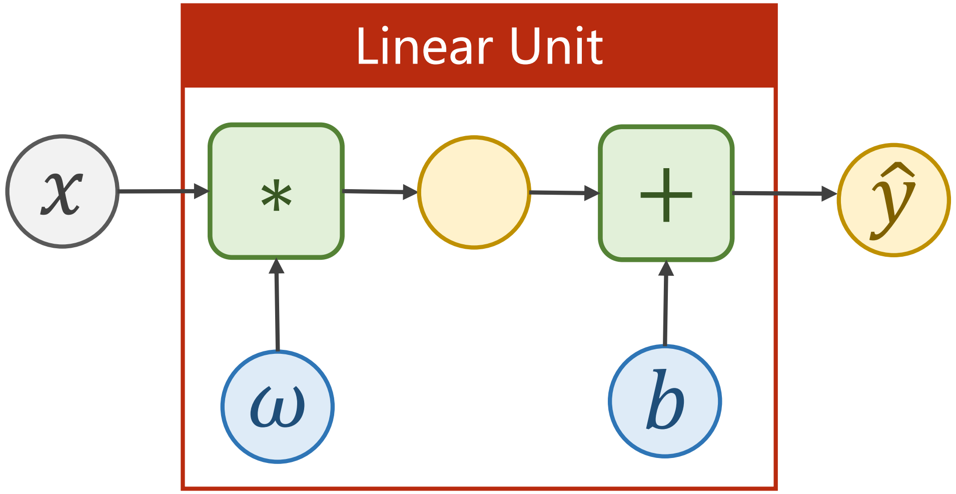
Logistic Regression Model

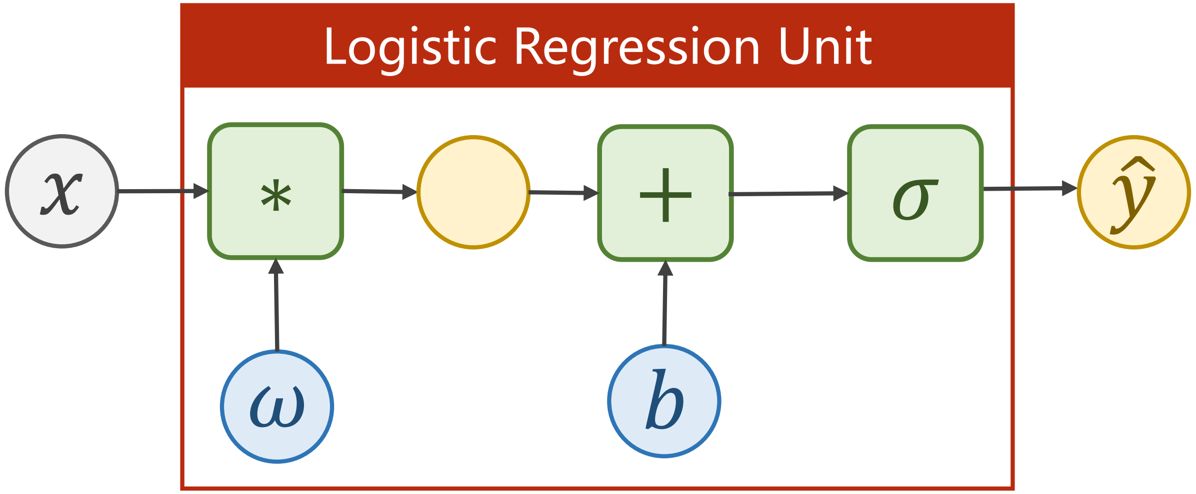
Affine Model

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

Logistic Regression Model

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





Loss function for Binary Classification

Loss Function for Linear Regression

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

Loss Function for Binary Classification

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

Mini-Batch Loss function for Binary Classification

Loss Function for Binary Classification

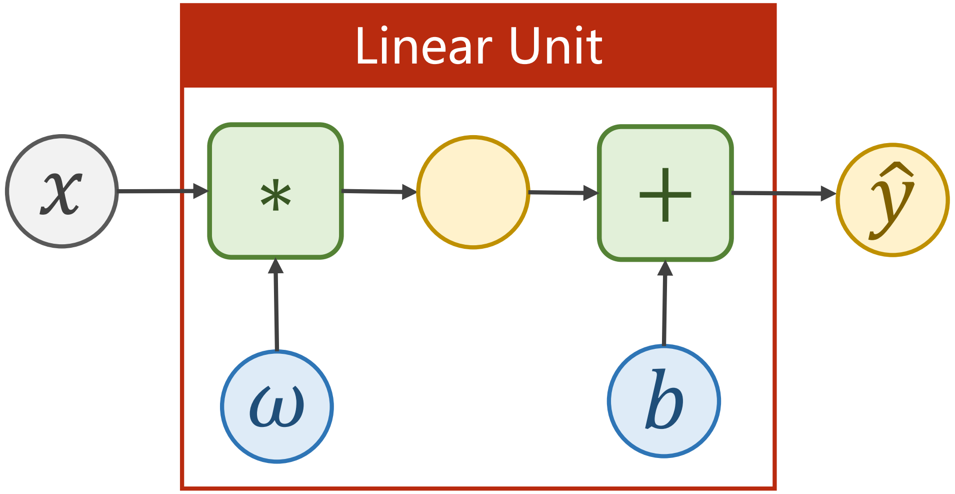
$$loss = -(y \log \hat{y} + (1 - y)\log(1 - \hat{y}))$$

Mini-Batch Loss Function for Binary Classification

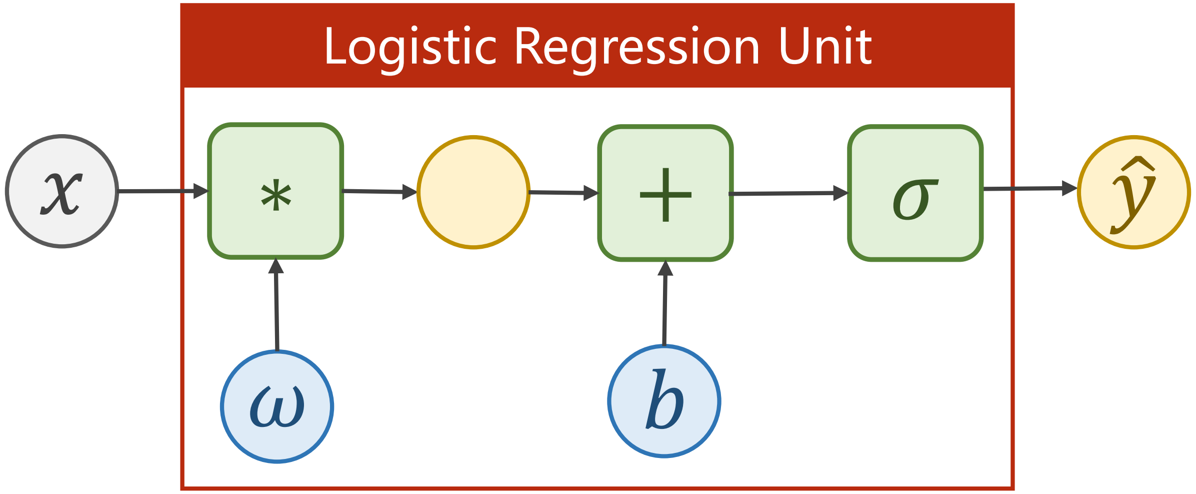
$$loss = -\frac{1}{N}\sum_{n=1}^{N} y_n \log \hat{y}_n + (1 - y_n)\log(1 - \hat{y}_n)$$

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**

Implementation of Logistic Regression




```
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
```




```
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

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

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

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()
```

1 Prepare dataset

we shall talk about this later

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()
```

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()
```

3

Construct loss and optimizer

using PyTorch API

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()
```

4

Training cycle

forward, backward, update

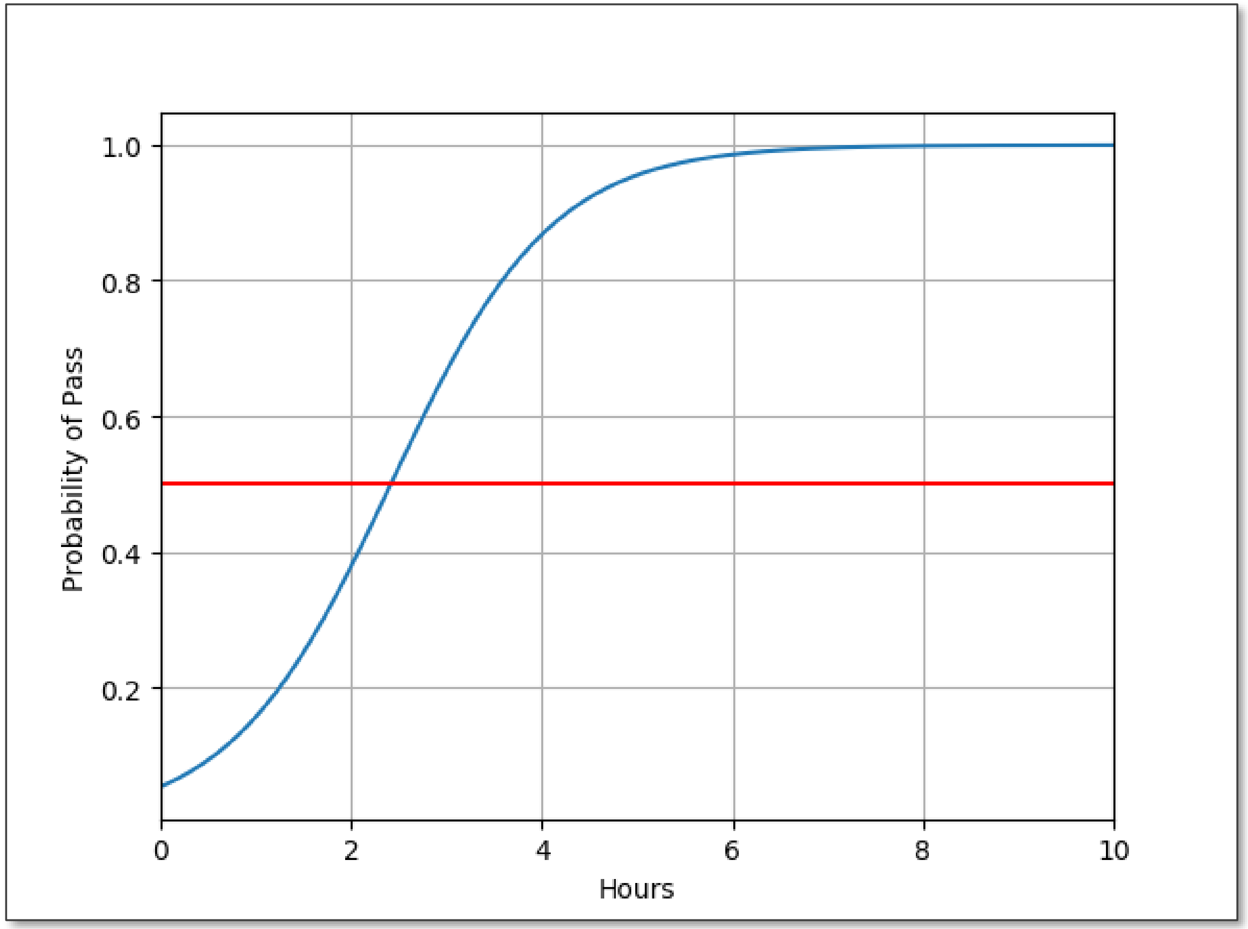
Implementation of Logistic Regression

```
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

```python
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()
```



PyTorch Tutorial

06. Logistic Regression