第二章 神经网络与深度学习基础

目录
CONTENTS

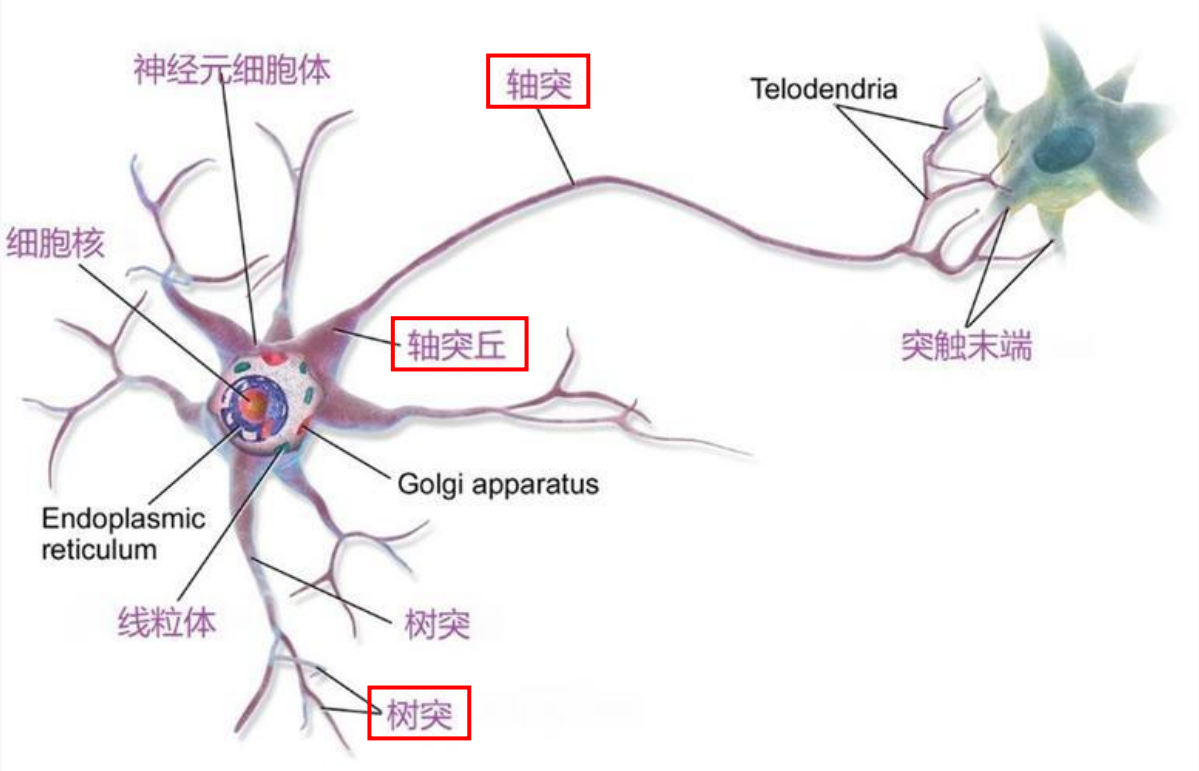
Chapter 2.1

全连接神经网络

全连接神经网络

生物神经元

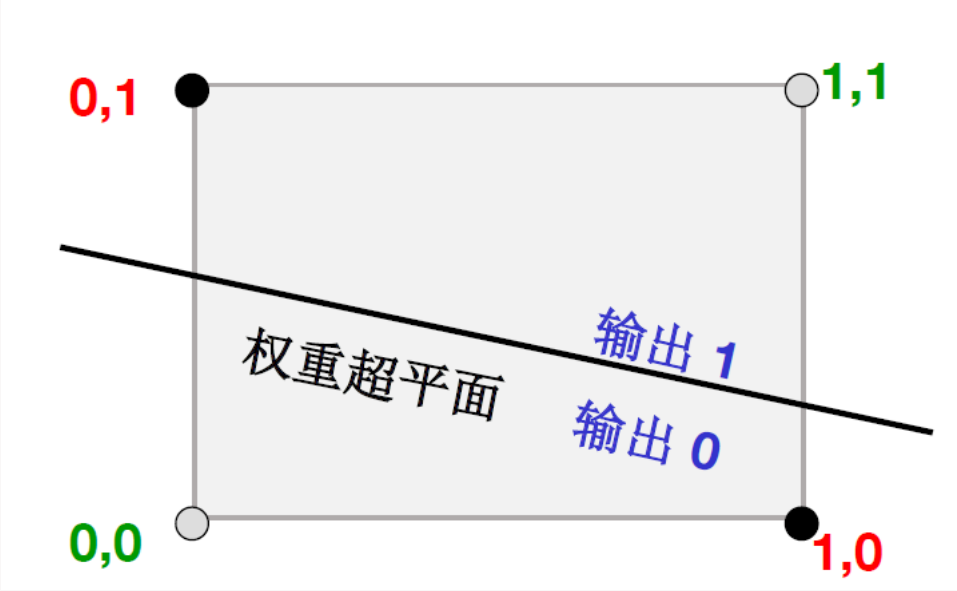
- 人脑主要由互相连接的神经元构成，神经元具有特殊的分枝状神经突起结构——树突和轴突。



感知机模型

- 感知机是一种二分类的线性分类模型，输入为实例的特征向量，输出为实例的类别。

$$y = \begin{cases} 1, & \text{if } \sum_{i=1}^{n} w_i \cdot x_i + b \geq 0 \\ 0, & \text{otherwise} \end{cases}$$

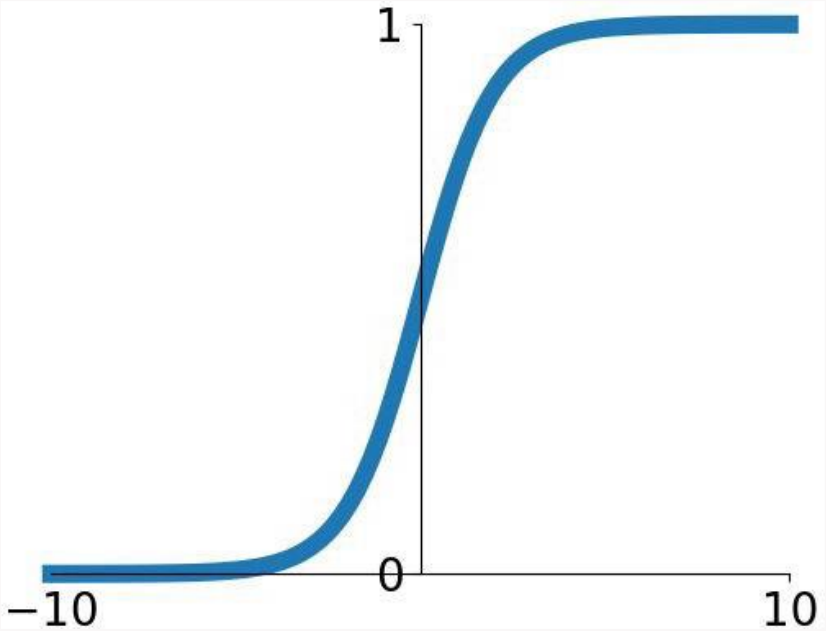


感知机的局限性是只能解决线性可分问题，无法处理复杂的非线性关系。

全连接神经网络

激活函数

- 激活函数是一种非线性映射函数，通常需要满足以下三个性质：
  - 单调性
  - 可微性
  - 非线性



Sigmoid激活函数

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

缺点：

1. 梯度消失问题；
2. 计算复杂度高；
3. sigmoid函数的输出非零均值。

全连接神经网络

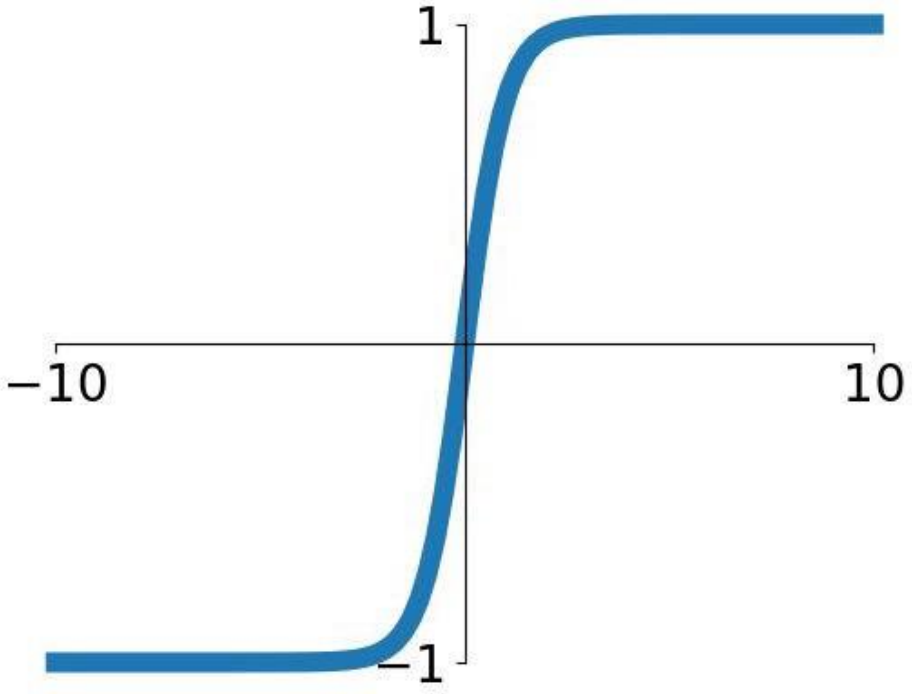
- 非零均值问题
- 非零均值问题会降低梯度下降法的效率

$$w \leftarrow w - \eta \cdot \frac{\partial L}{\partial w}$$

$$f(z) = f(\sum_i w_i x_i + b)$$

全连接神经网络

- 其他激活函数



tanh激活函数

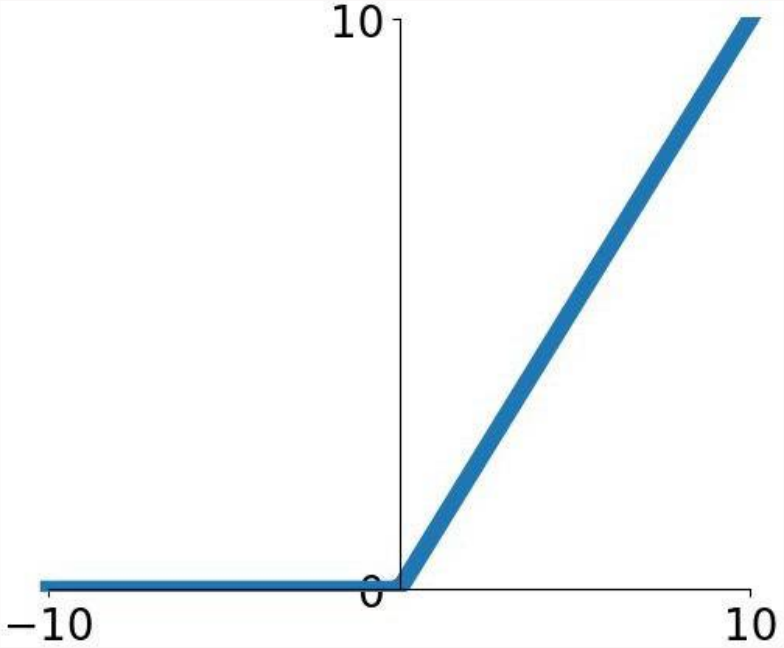
$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

特点：

1. 梯度消失问题；
2. 输出是零均值；
3. 计算复杂度高。

全连接神经网络

- 其他激活函数



ReLU激活函数

$$\mathrm{ReLU}(x) = \max(0, x)$$

特点：

1.在x>0区域不会出现梯度消失的问题；

2. 计算复杂度低；

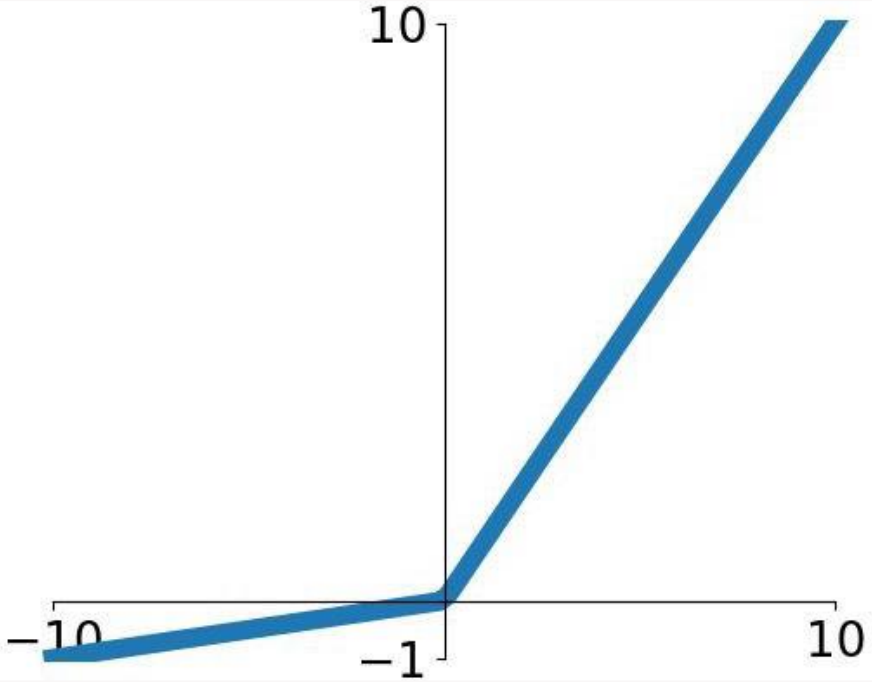
3. 参数稀疏性；

4. 输出非零均值；

5. 神经元坏死问题，在x<0区域不会更新参数。

全连接神经网络

- 其他激活函数



Leaky ReLU激活函数

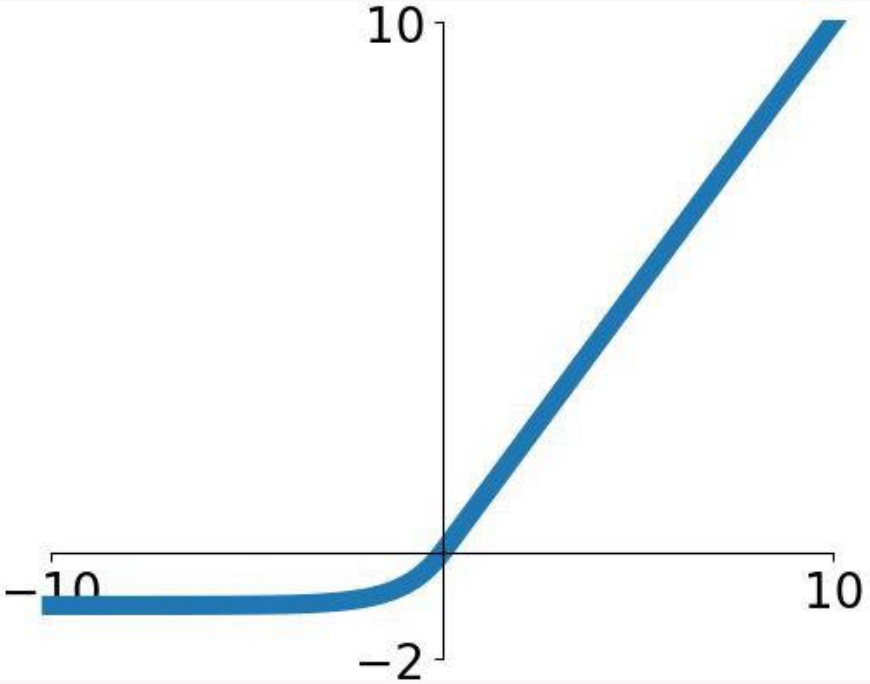
$$\text{Leaky ReLU}(x) = \begin{cases} x, & \text{if } x > 0 \\ \alpha x, & \text{otherwise} \end{cases}$$

特点：

1. 解决神经元坏死问题；
2. 输出均值接近零；
3. 非线性映射能力变弱。

全连接神经网络

- 其他激活函数



ELU激活函数

$$\mathrm{ELU}(x) = \begin{cases} x, & \text{if } x > 0 \\ \alpha(e^x - 1), & \text{otherwise} \end{cases}$$

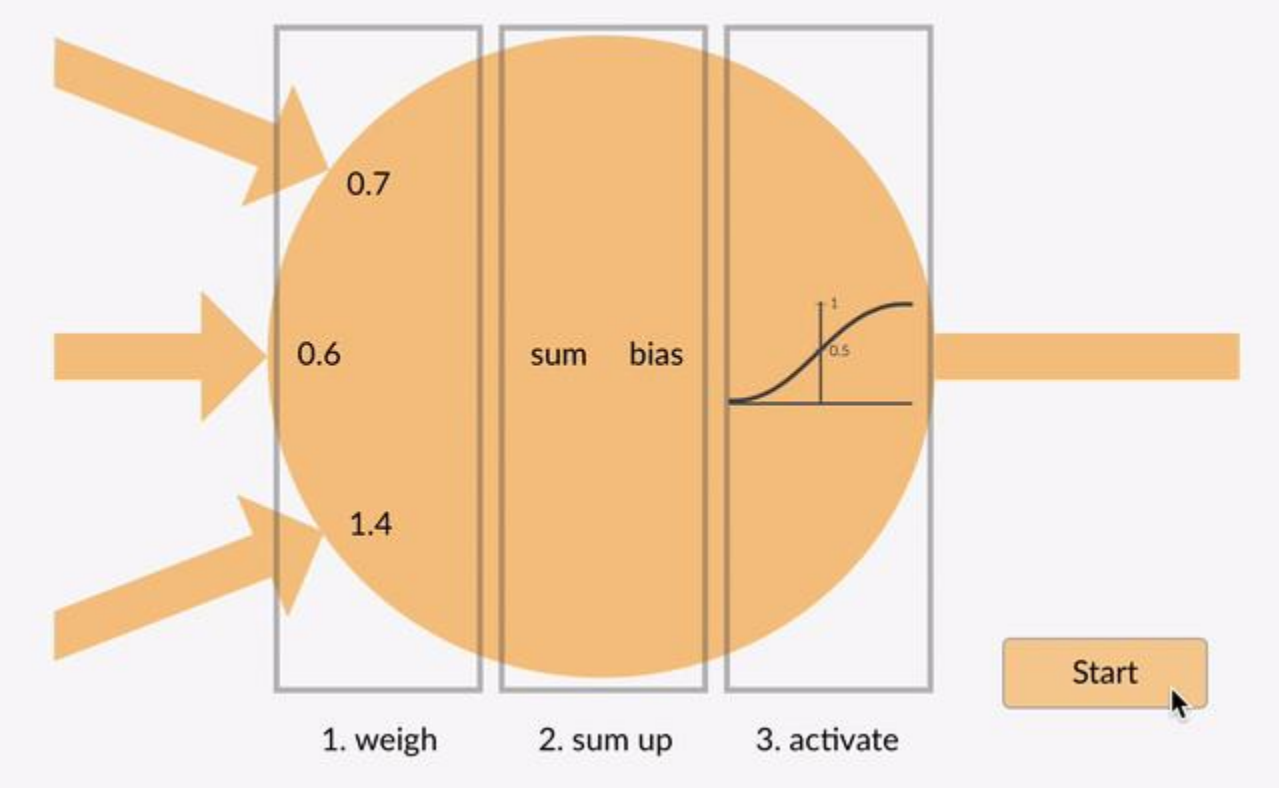
特点：

1. 解决神经元坏死问题；
2. 输出均值接近零；
3. 运算速度更慢。

全连接神经网络

人工神经元

- 人工神经元利用输入权重模拟树突连接，采用加权求和的方式模拟细胞体，使用非线性激活函数模拟轴丘。

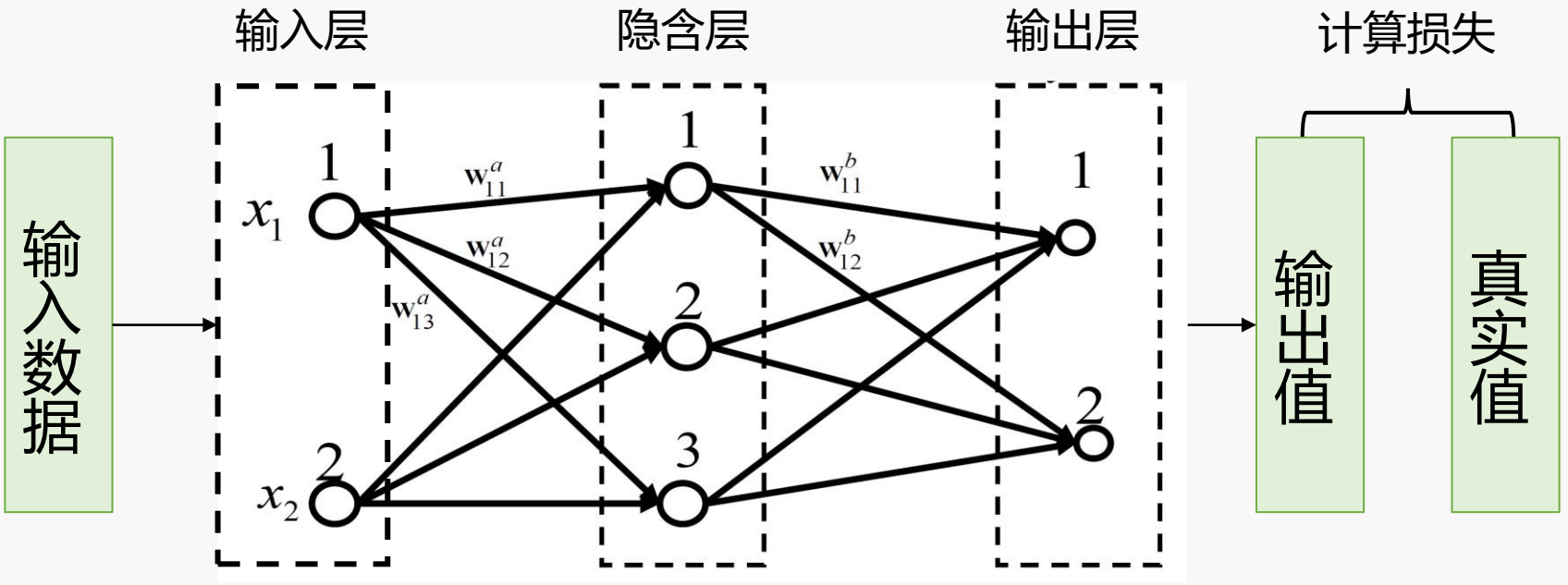


全连接神经网络

- 人工神经网络
- 人工神经网络包含三个部分：
  - 输入神经元
  - 多层神经网络
  - 输出神经元
- 采用多层神经元结构，同层的神经元为并列关系，相邻层的神经元稠密连接。这种网络也成为多层感知机（Multi-Layer Perceptron，MLP）或全连接神经网络（Fully Connected Neural Network，FCNN）。

全连接神经网络

- 人工神经网络

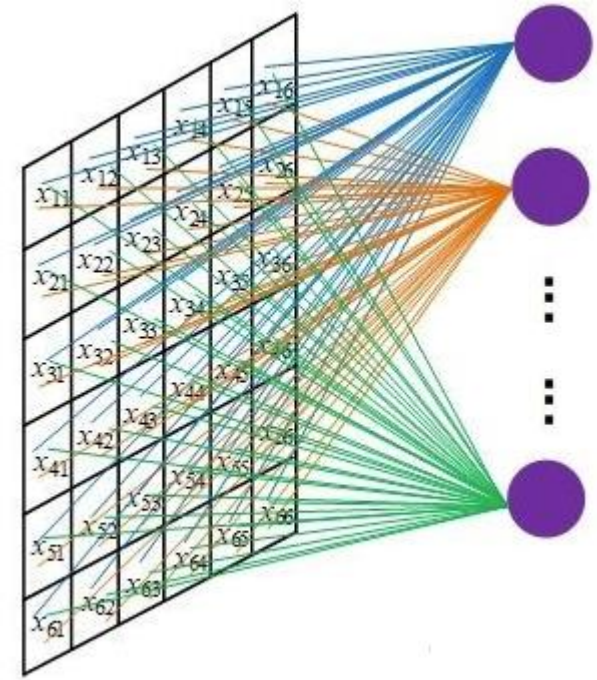


Chapter 2.2

卷积神经网络

卷积操作

- 人工神经网络是模仿生物大脑内的神经元结构设计的。为视觉任务设计的神经网络应该考虑视觉系统的特点。

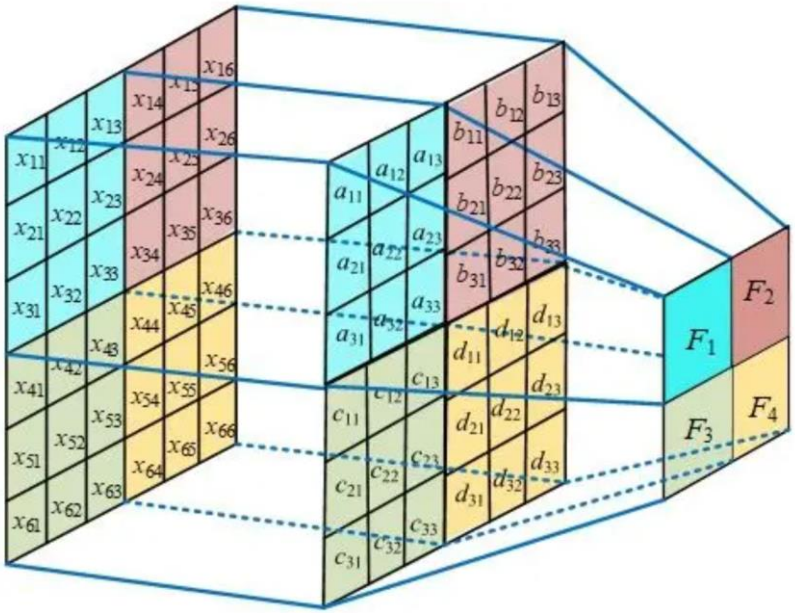


如果图像分辨率为1024×1024，采用全连接方式，单个神经元需要1百万个参数。

卷积神经网络

卷积操作

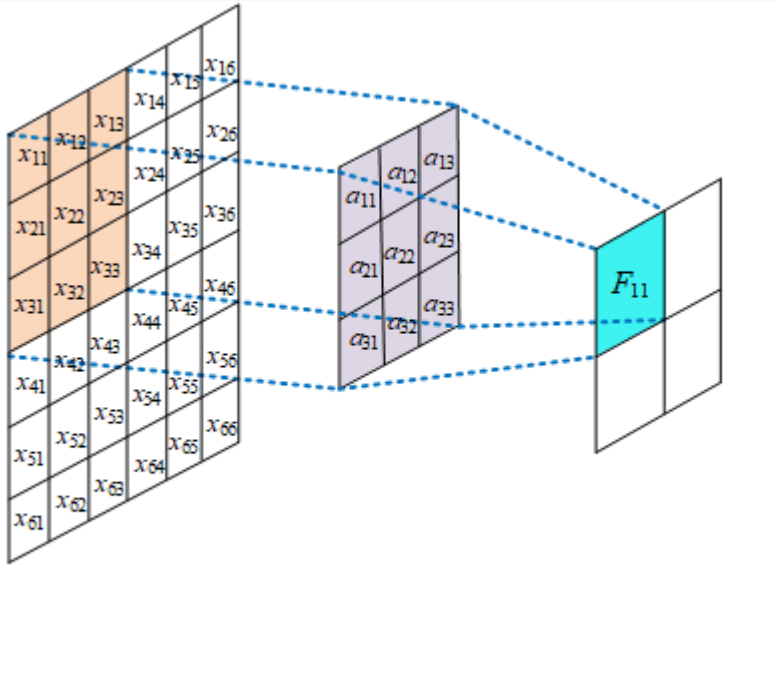
- 人类视觉系统的启发下，通过将全局连接改为**局部连接**（Local Connectivity），将每个神经元的感受野限制在较小范围。



如果图像分辨率为1024×1024，采用3×3的局部连接方式，共需要1百万个参数。

卷积操作

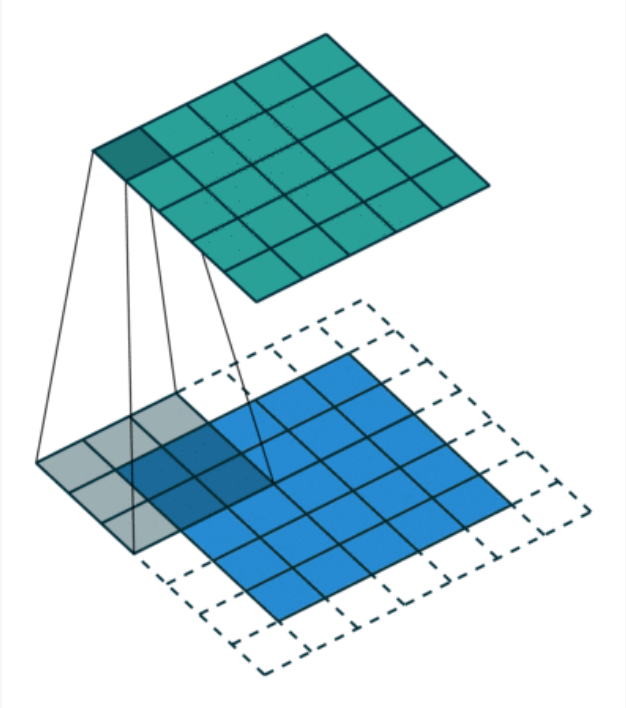
- 利用**参数共享**（Parameter sharing）将同一个特征图的所有神经元共享相同的权重，进一步减少参数量。



如果图像分辨率为1024×1024，3x3的局部连接方式仅需9个参数。

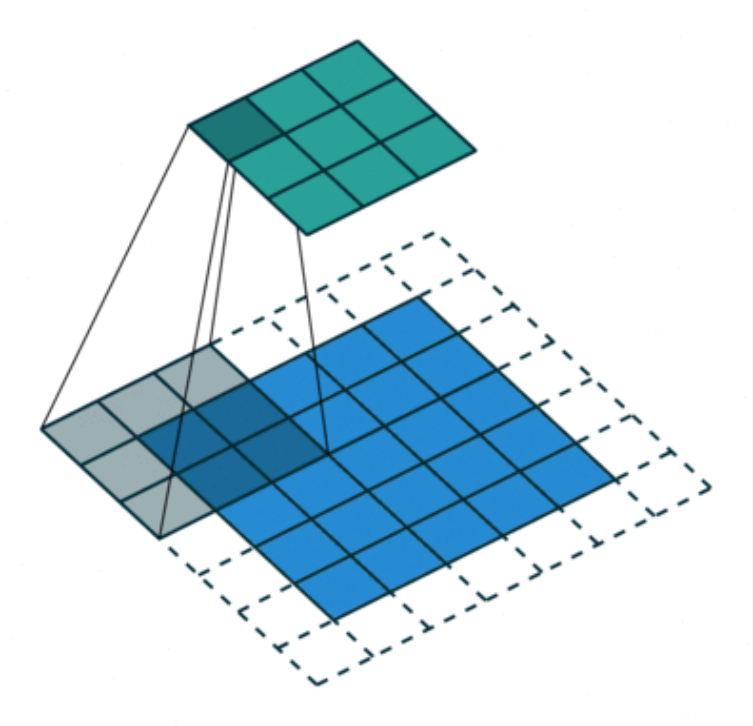
填充

- 卷积操作可能导致输入和输出的特征图不一致，在卷积操作前可以对输入特征图使用**填充**（Padding）以避免这个问题。
  - 零填充
  - 边界填充



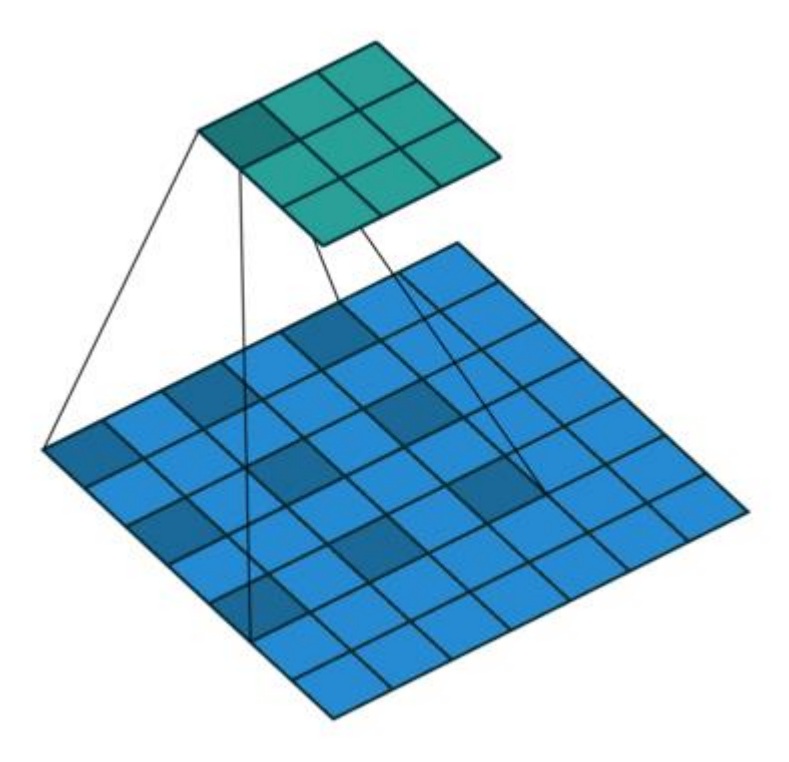
步幅

- **步幅**（Stride）控制卷积核在输入特征图上滑动的距离，可以大幅减少输出特征图的尺寸。



扩张率

- 通过在标准卷积核中插入空洞来扩大感受野，**扩张率**（Dilation rate）定义了卷积核中像素点之间的间隔数量。



卷积核输出计算公式

- 假设输入特征图的尺寸为 $H \times W$，输出特征图的尺寸为 $H' \times W'$，则它们与卷积核大小 K 、填充量 P 以及步幅 S 的关系为

$$H' = \left\lfloor \frac{H - K_H + 2P}{S} \right\rfloor + 1$$

$$W' = \left\lfloor \frac{W - K_W + 2P}{S} \right\rfloor + 1$$

- 使用扩张率 D 后，卷积核变为

$$K'_H = K_H + (K_H - 1)(\mathrm{D}_H - 1)$$

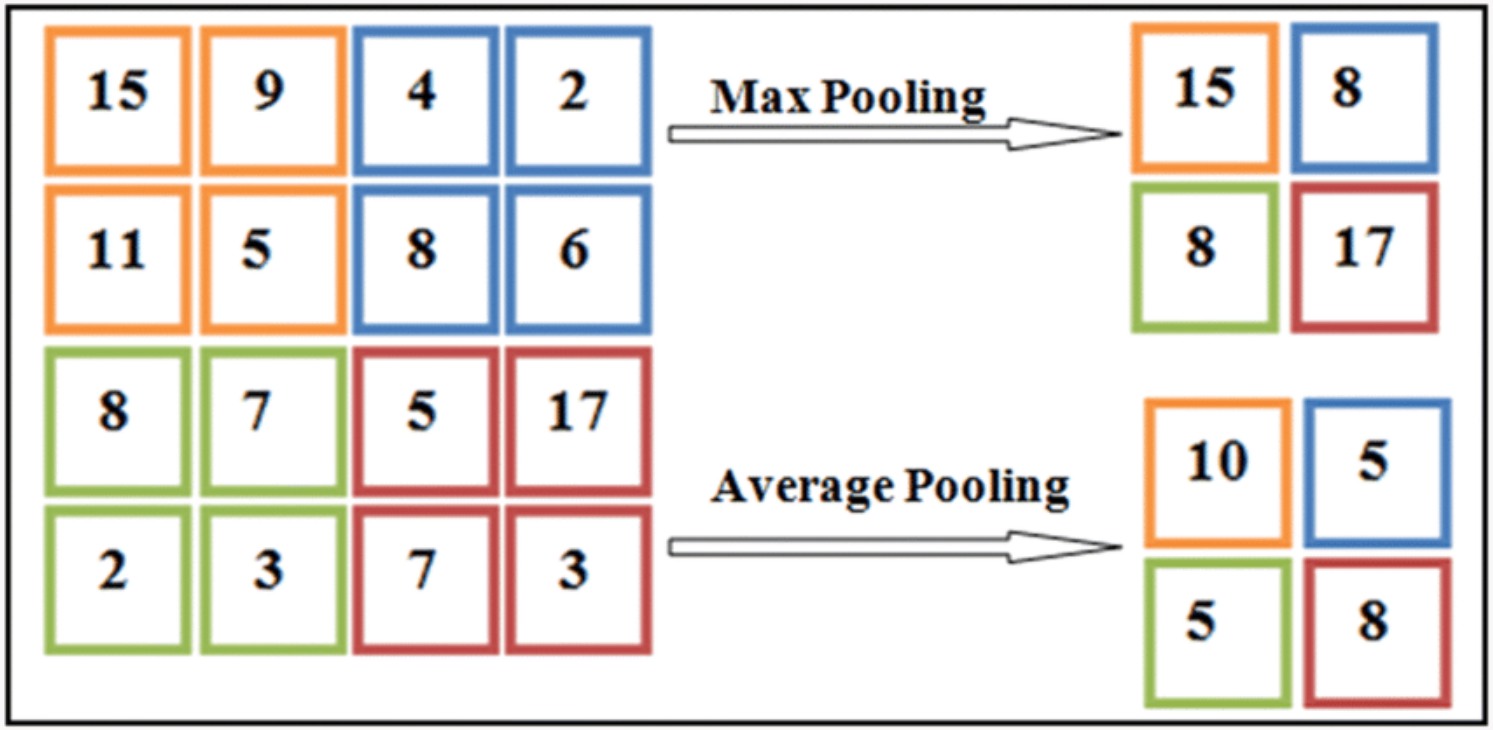
$$K'_W = K_W + (K_W - 1)(\mathrm{D}_W - 1)$$

降采样处理

- 卷积神经网络中还需要对特征图进行降采样处理（Downsampling），可以减少后续层的输入特征图的尺寸，通过扩大感受野更好地捕获全局信息。
  - 池化（Pooling）：最大池化和平均池化
  - 重采样：插值算法

池化

- 将输入特征图中的每个局部区域用一个汇总统计值来代表，通常采用**最大池化**（Max pooling）和**平均池化**（Average pooling）。



池化

- 最大池化和平均池化具有以下区别：
  - 最大池化更关注于图像中的细节纹理信息，一般应用于网络浅层；
  - 平均池化更关注于图像中的全局形状信息，一般应用于网络深层。

卷积神经网络

- 池化

原图

均值池化

最大池化

Chapter 2.3

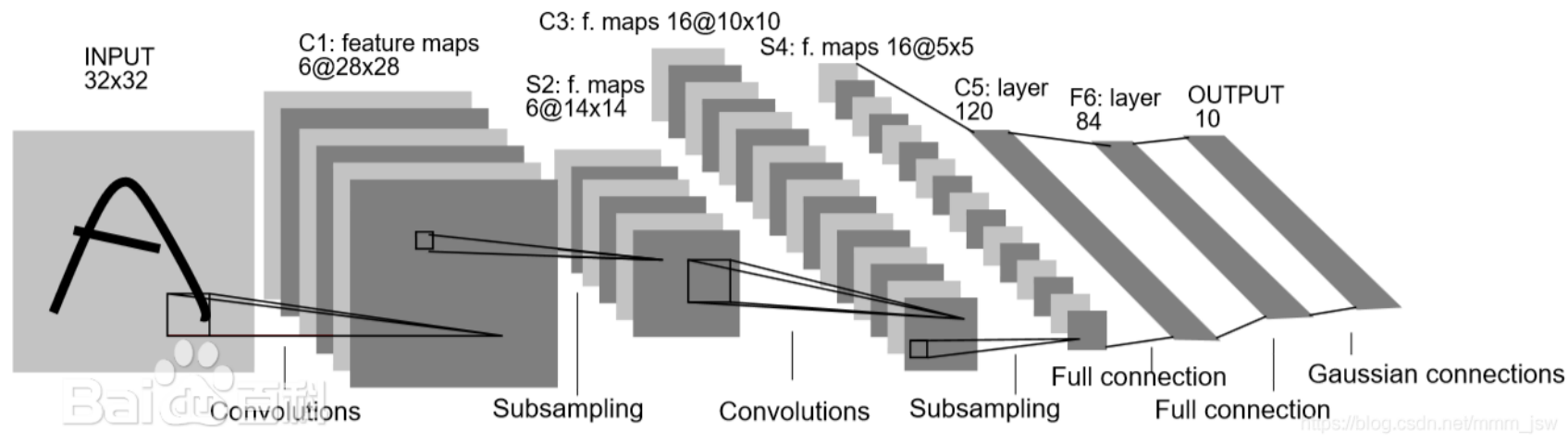
训练卷积神经网络

训练卷积神经网络

- 训练神经网络
- Pytorch框架下的神经网络训练过程：
  - 搭建网络
  - 准备数据
  - 计算前向传播结果
  - 计算损失函数
  - 反向传播
  - 更新参数

训练卷积神经网络

- 搭建网络



训练卷积神经网络

- 搭建网络

```
import torch.nn as nn
import torch.nn.functional as F
# 继承一个nn.Module，实现了构造函数和forward方法，就是一个网络模型
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        # 2维卷积，输入通道3，输出通道6，卷积核大小5x5
        # 还有其它参数可以设置（stride, padding）
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        # fc fully connected，全连接层
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)
```

训练卷积神经网络

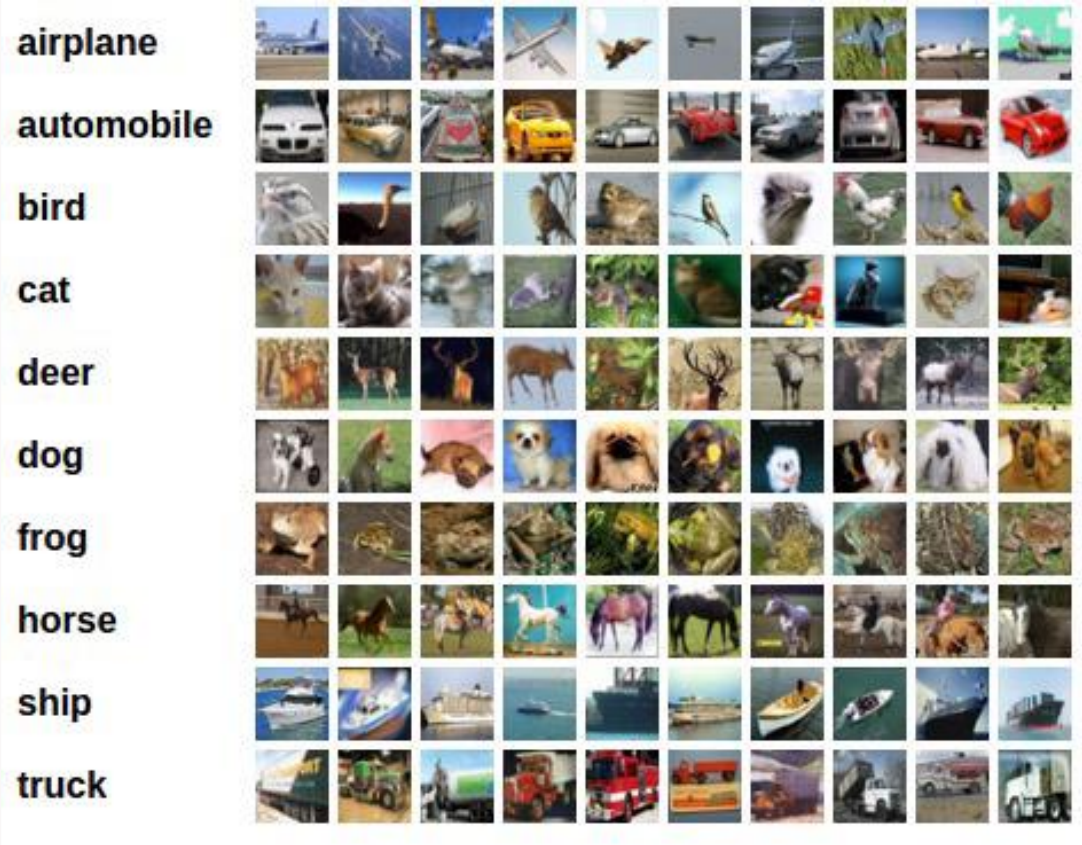
- 搭建网络

```
def forward(self, x):
    x = self.pool(F.relu(self.conv1(x)))
    x = self.pool(F.relu(self.conv2(x)))
    x = x.reshape(-1, 16 * 5 * 5)
    x = F.relu(self.fc1(x))
    x = F.relu(self.fc2(x))
    x = self.fc3(x)
    return x
```

训练卷积神经网络

准备数据

- CIFAR-10数据集中每个图片的尺寸为32 × 32 ,每个类别有6000个图像，共有50000 张训练图片和10000 张测试图片。



训练卷积神经网络

- 准备数据

```
import torch
import torchvision
import torchvision.transforms as transforms
import numpy as np

class CIFAR10Dataset(torch.utils.data.Dataset):
  def __init__(self, transform, data, label):
    super(CIFAR10Dataset, self).__init__()
    self.transform = transform
    self.images = data
    self.labels = label

  def __getitem__(self, idx):
    img = self.images[idx]
    img = self.transform(img)
    label = self.labels[idx]
    return img, label

  def __len__(self):
    return len(self.images)
```

训练卷积神经网络

- 准备数据

```
class CIFAR10Dataset(torch.utils.data.Dataset):
  def __init__(self, transform, data, label):
    super(CIFAR10Dataset, self).__init__() # 调用父类的构造函数

    self.transform = transform              # 设置对象属性 transform

    self.images = data # 假设data的shape为（图片数，32，32，3）
                       # 假设data的数据类型是 np.float32，值域 [0,1]

    self.labels = label # 假设label的shape为（图片数，）
                        # PyTorch 会在计算交叉熵时
                        # 自动转为 one-hot 编码
```

训练卷积神经网络

- 准备数据

```
class CIFAR10Dataset(torch.utils.data.Dataset):
  ……

  def __getitem__(self, idx):
    img = self.images[idx]
    img = self.transform(img)
    label = self.labels[idx]
    return img, label

  # 对于一个这个类的对象，可以用 len(obj) 来获取长度
  def __len__(self):
    return len(self.images)
```

训练卷积神经网络

- 准备数据

```
# 定义 transform：包括两个顺序步骤
# 1 把numpy数组转化为pytorch张量
# 2 归一化到 [-0.5, 0.5]，有利于 ReLU
transform = transforms.Compose(
    [transforms.ToTensor(),
     transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))]
    )

# 假设我们把数据都做好了
train_data  = np.load('train_data.npy')
train_label = np.load('train_label.npy')
test_data   = np.load('test_data.npy')
test_label  = np.load('test_label.npy')
```

训练卷积神经网络

- 准备数据

```
# trainset 是一个 CIFAR10Dataset 实例，可以用下标索引
# 下标索引会返回一个 sample 的 data 和 label
trainset = CIFAR10Dataset(transform=transform,
                          data=train_data, label=train_label)

# PyTorch 提供的 dataloader 可以方便地控制 batchsize 和 shuffle
# 以及提供了异步接口。如果出现异步问题，设置num_workers=0
trainloader = torch.utils.data.DataLoader(trainset,
                    batch_size=4, shuffle=True, num_workers=2)
```

训练卷积神经网络

- 准备数据

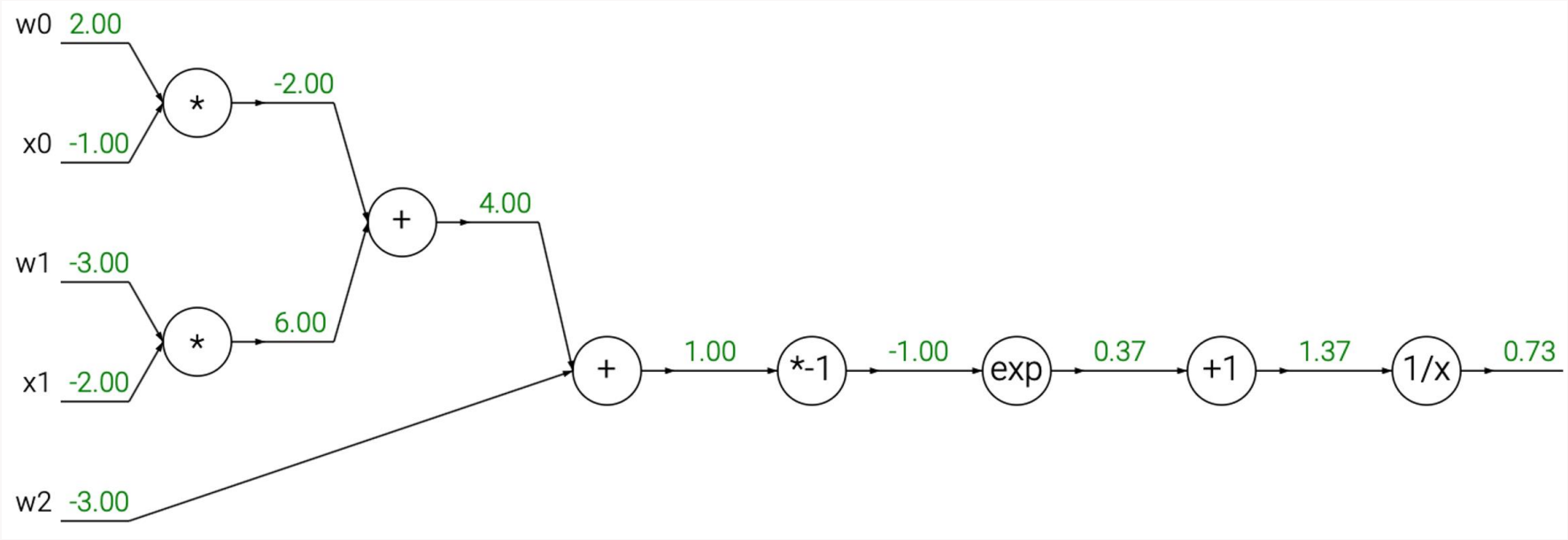
```
# 类似地构造testset
testset = CIFAR10Dataset(transform=transform,
                         data=test_data, label=test_label)
testloader = torch.utils.data.DataLoader(testset,
              batch_size=4, shuffle=False, num_workers=2)

classes = ('plane', 'car', 'bird', 'cat',
           'deer', 'dog', 'frog', 'horse', 'ship', 'truck')
```

训练卷积神经网络

计算前向传播结果

- 通常采用有向无环图（Directed Acyclie Graph，DAG）的形式组织成计算图进行表示和执行。

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$



训练卷积神经网络

- 计算前向传播结果

```
# 实例化一个网络
net = Net()
# net = Net().cuda() 改成这个把网络放到GPU上

import torch.optim as optim
# 交叉熵，PyTorch 默认自带 SoftMax
criterion = nn.CrossEntropyLoss()

# Stochastic Gradient Descent
optimizer = optim.SGD(net.parameters(),
                      lr=0.001, momentum=0.9)
```

训练卷积神经网络

- 计算前向传播结果

```
for epoch in range(10):
  for i, data in enumerate(trainloader, 0):
    # 获取输入数据
    inputs, labels = data

    optimizer.zero_grad()
    # 前传
    outputs = net(inputs)
    # 计算 loss
    loss = criterion(outputs, labels)
    # 反传
    loss.backward()
    # 更新
    optimizer.step()
```

训练卷积神经网络

- 计算损失

```
# 实例化一个网络
net = Net()
# net = Net().cuda() 改成这个把网络放到GPU上

import torch.optim as optim
# 交叉熵，PyTorch 默认自带 SoftMax
criterion = nn.CrossEntropyLoss()

# Stochastic Gradient Descent
optimizer = optim.SGD(net.parameters(),
                      lr=0.001, momentum=0.9)
```

训练卷积神经网络

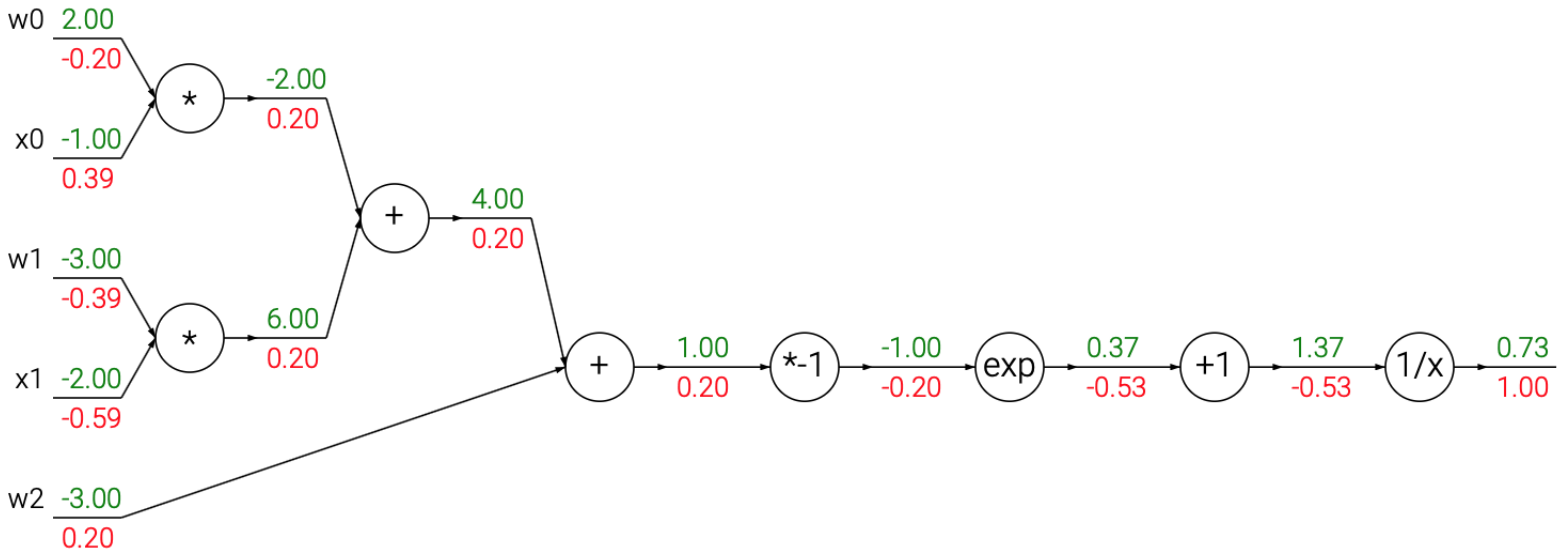
- 计算损失

```
for epoch in range(10):
  for i, data in enumerate(trainloader, 0):
    # 获取输入数据
    inputs, labels = data

    optimizer.zero_grad()
    # 前传
    outputs = net(inputs)
    # 计算 loss
    loss = criterion(outputs, labels)
    # 反传
    loss.backward()
    # 更新
    optimizer.step()
```

训练卷积神经网络

反向传播

- 梯度的反向传播利用了链式法则和有向无环图的结构，在当前操作梯度和反向传播梯度的基础上，自顶向下地传播梯度。



训练卷积神经网络

- 反向传播

```
for epoch in range(10):
  for i, data in enumerate(trainloader, 0):
    # 获取输入数据
    inputs, labels = data

    optimizer.zero_grad()
    # 前传
    outputs = net(inputs)
    # 计算 loss
    loss = criterion(outputs, labels)
    # 反传
    loss.backward()
    # 更新
    optimizer.step()
```

训练卷积神经网络

- 更新参数

```
# 实例化一个网络
net = Net()
# net = Net().cuda() 改成这个把网络放到GPU上

import torch.optim as optim
# 交叉熵，PyTorch 默认自带 SoftMax
criterion = nn.CrossEntropyLoss()

# Stochastic Gradient Descent
optimizer = optim.SGD(net.parameters(),
                      lr=0.001, momentum=0.9)
```

训练卷积神经网络

- 更新参数

```
for epoch in range(10):
  for i, data in enumerate(trainloader, 0):
    # 获取输入数据
    inputs, labels = data

    optimizer.zero_grad()
    # 前传
    outputs = net(inputs)
    # 计算 loss
    loss = criterion(outputs, labels)
    # 反传
    loss.backward()
    # 更新
    optimizer.step()
```

训练卷积神经网络

- 更新参数

```
for epoch in range(10):
  for i, data in enumerate(trainloader, 0):
    # 获取输入数据
    inputs, labels = data

    optimizer.zero_grad()
    # 前传
    outputs = net(inputs)
    # 计算 loss
    loss = criterion(outputs, labels)
    # 反传
    loss.backward()
    # 更新
    optimizer.step()
```

训练卷积神经网络

- 训练神经网络
  - 神经网络训练过程：
    - 搭建网络
    - 准备数据
    - 计算前向传播结果
    - 计算损失函数
    - 反向传播
    - 更新参数
  - 实践05：实现一个简单的实用神经网络

第二章 完结