

■ 人工智能、机器学习与深度学习概述

- 什么是人工智能:人工智能是让机器具备模拟人类智能的能力,从而实现学习、推理、解决问题和感知环境等功能的技术。
- 三者之间的关系



计算机可以像人类一样思考吗?



当时认为: **完全可以**

■ 人工智能、机器学习与深度学习概述

● 什么是机器学习

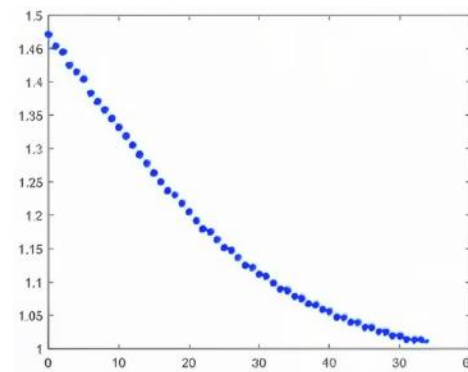


$$y = a + b$$

```
def add_calc(a,b):  
    return a + b
```

$$y = 3a + b$$

```
def add_calc(a,b):  
    return 3*a + b
```



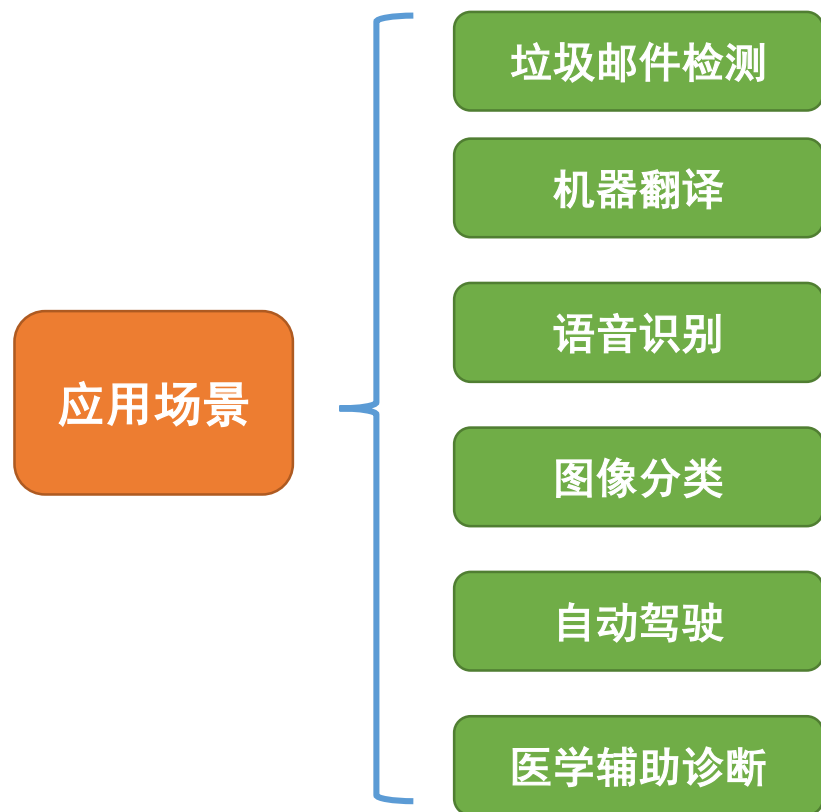
机器学习作用

无法用传统编程思维
逻辑去解决问题

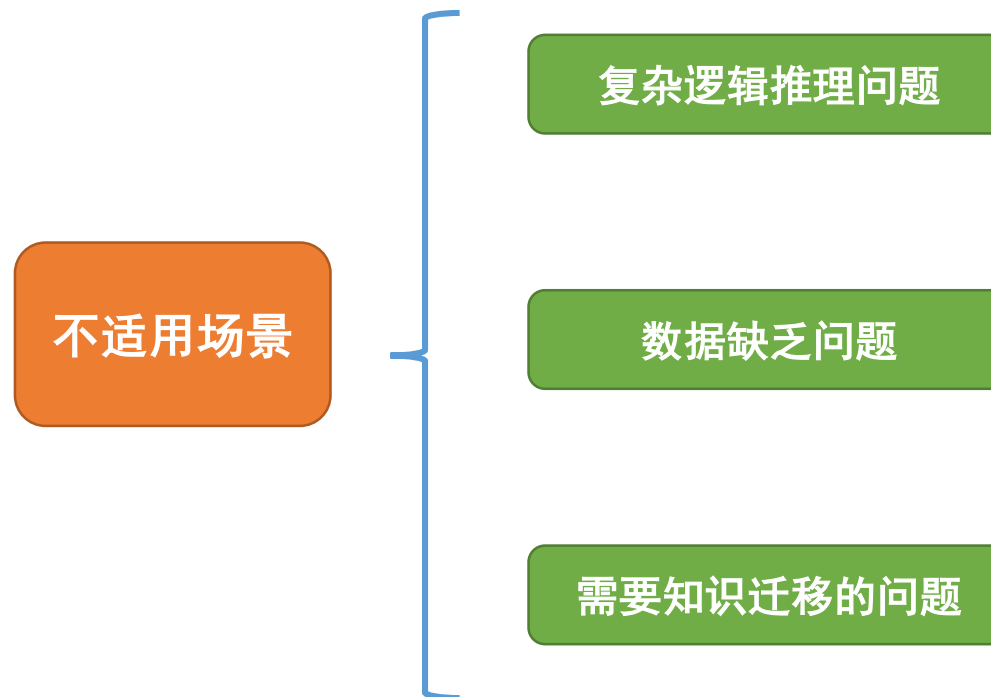


■ 人工智能、机器学习与深度学习概述

● 机器学习应用场景

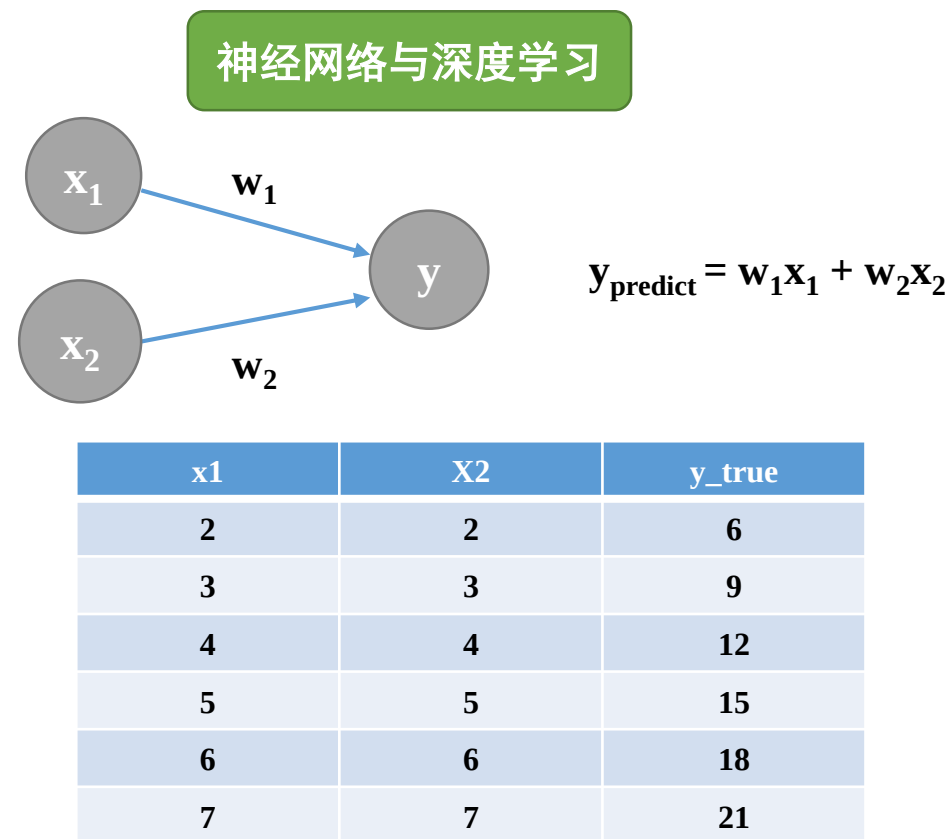
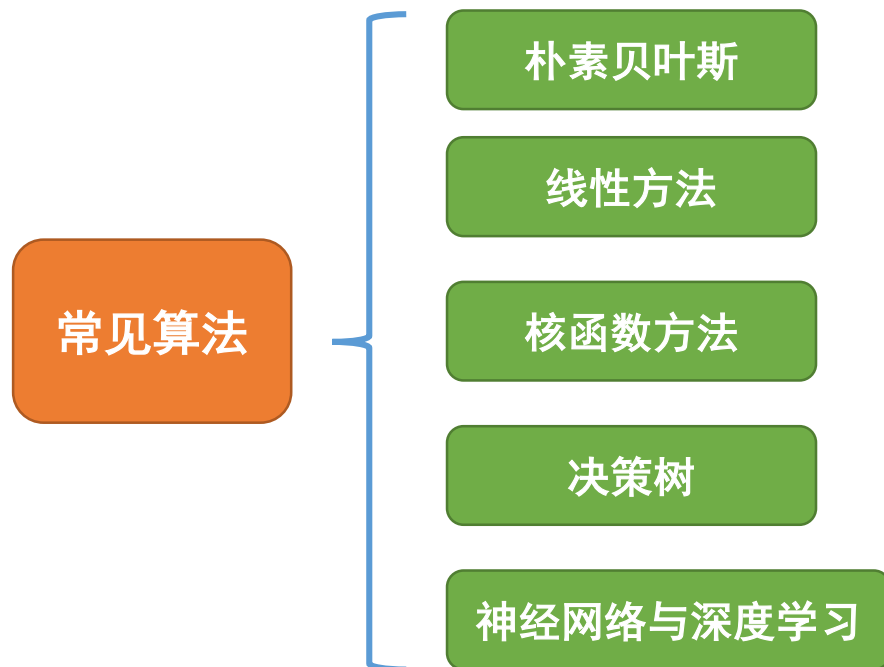


● 机器学习算法不适用场景



■ 人工智能、机器学习与深度学习概述

● 常见机器学习算法



■ 人工智能、机器学习与深度学习概述

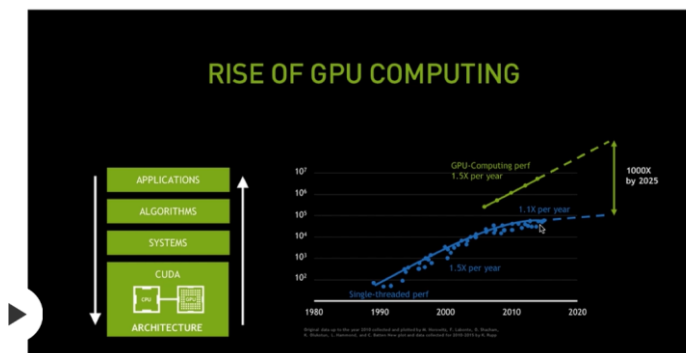
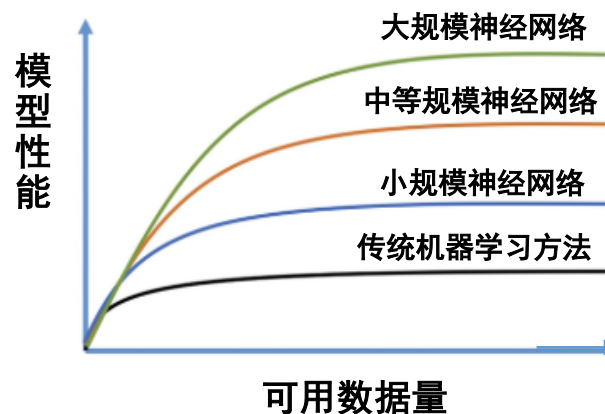
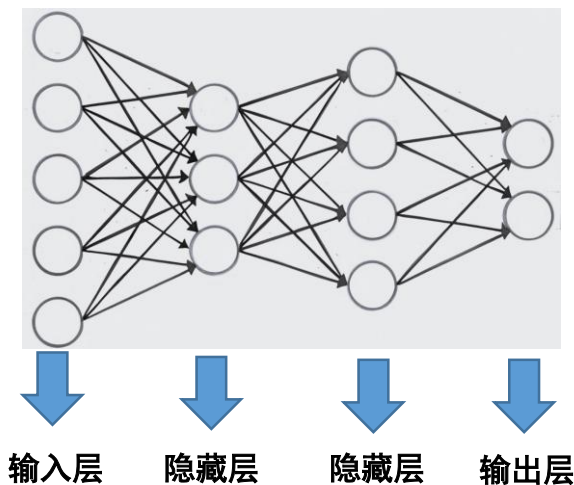
● 深度学习的特别之处

特别之处

大数据

简单层数叠加带来的极佳拟合能力

硬件运算性能的支持



摩尔定理

■ Pytorch简介及相关数学原理与算法

● Pytorch框架简介

Pytorch开源地址

<https://github.com/pytorch/pytorch>

<https://pytorch.org/>



END-TO-END MACHINE LEARNING FRAMEWORK

PyTorch enables fast, flexible experimentation and efficient production through a user-friendly front-end, distributed training, and ecosystem of tools and libraries.

```
import torch
class MyModule(torch.nn.Module):

    def __init__(self, N, M):
        super(MyModule, self).__init__()
        self.weight = torch.nn.Parameter(torch.rand(N, M))

    def forward(self, input):
        if input.sum() > 0:
            output = self.weight.mv(input)
        else:
            output = self.weight + input
        return output

# Compile the model code to a static representation
my_script_module = torch.jit.script(MyModule(3, 4))

# Save the compiled code and model data so it can be
my_script_module.save("my_script_module.pt")
```

PRODUCTION READY

With TorchScript, PyTorch provides ease-of-use and flexibility in eager mode, while seamlessly transitioning to graph mode for speed, optimization, and functionality in C++ runtime environments.

TORCHSERVE

TorchServe is an easy to use tool for deploying PyTorch models at scale. It is cloud and environment agnostic and supports features such as multi-model serving, logging, metrics and the creation of RESTful endpoints for application integration.

```
import torch.distributed as dist
from torch.nn.parallel import DistributedDataParallel

dist.init_process_group(backend='gloo')
model = DistributedDataParallel(model)
```

```
import torchvision.models as models
resnet18 = models.resnet18(pretrained=True)
alexnet = models.alexnet(pretrained=True)
squeezenet = models.squeezenet1_0(pretrained=True)
vgg16 = models.vgg16(pretrained=True)
densenet = models.densenet161(pretrained=True)
inception = models.inception_v3(pretrained=True)
```

```
## Convert the model from PyTorch to TorchServe format
torch-model-archiver --model-name densenet161 \
--version 1.0 --model-file serve/examples/image_classif
--serialized-file densenet161-8d451a50.pth \
--extra-files serve/examples/image_classifier/index_to_
--handler image_classifier

## Host your PyTorch model

torchserve --start --model-store model_store --models de
```

DISTRIBUTED TRAINING

Optimize performance in both research and production by taking advantage of native support for asynchronous execution of collective operations and peer-to-peer communication that is accessible from Python and C++.

ROBUST ECOSYSTEM

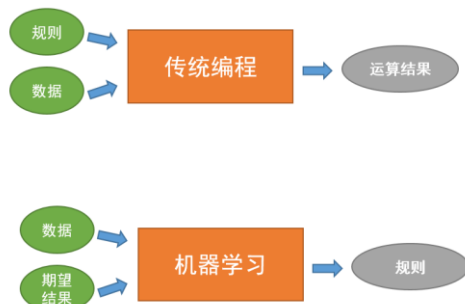
An active community of researchers and developers have built a rich ecosystem of tools and libraries for extending PyTorch and supporting development in areas from computer vision to reinforcement learning.

Pytorch简介及相关数学原理与算法

数学基础与算法

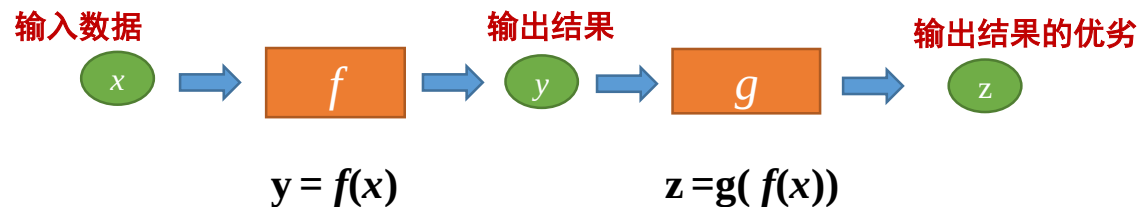
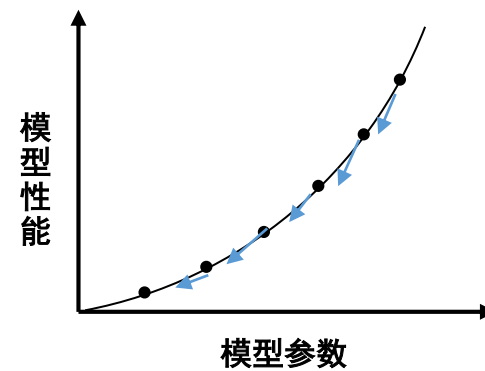
函数与导数

函数



导数

导数在机器学习中的意义:
1) 评估模型性能—假设 z 越小越好
2) 朝导数的反方向进行调整

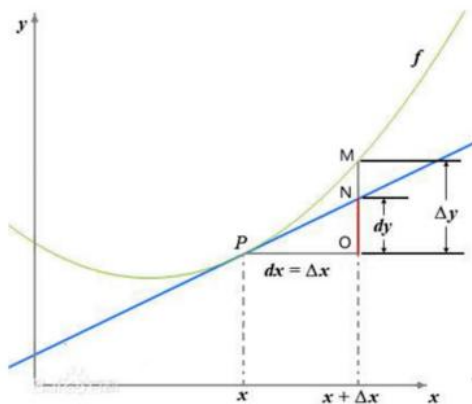


■ Pytorch简介及相关数学原理与算法

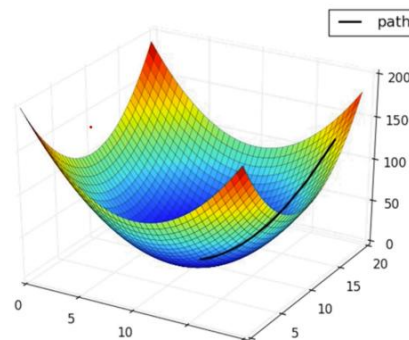
● 数学基础与算法

□ 梯度下降、局部极小值与鞍点

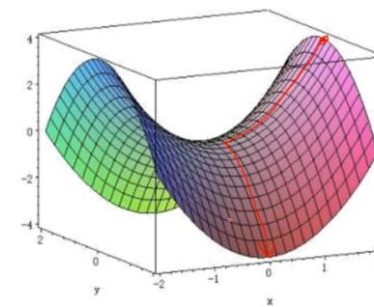
梯度下降法



局部极小值



鞍点



总结

- 模型性能(表现优劣)可以用函数描述
- 模型参数和模型性能存在函数关系
- 使用求导方法(梯度下降法)获知如何调整模型参数来让模型性能更好
- 使用梯度下降法模型参数存在意外的可能性，不一定能达到全局最优

■ Pytorch简介及相关数学原理与算法

● 数学基础与算法

□ 矩阵乘法

```
#函数化思想
nums = [1,2,3,4,5,6,7,8,9,10]
weights = [1,2,3,4,5,6,7,8,9,10]

def plain_mul_sum(nums, weights):
    sum_result = 0
    for i in range(len(nums)):
        sum_result += nums[i] * weights[i]
    return sum_result

#执行函数
plain_mul_sum(nums, weights)
```

$$1. y = w_1x_1 + w_2x_2 + w_3x_3$$

$$2. y = w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4$$

$$3. y = w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + \cdots + w_{100}x_{100}$$

$$4. \sum_{i=1}^N w_i x_i = w_1x_1 + w_2x_2 + w_3x_3 + \cdots + w_Nx_N$$

缺点: **for-loop**方法在数据量较大的情况下计算速度缓慢



向量化思想: 采用矩阵乘法运算减少运算量

Pytorch简介及相关数学原理与算法

● 数学基础与算法

□ 向量化和矩阵运算

➤ 向量化的意义

代码可读性和减少bug数量

计算向量每五个元素的累加和

$$1+2+3+4+5$$

$$1+2+3+4+5+6+7+8+9+10$$

.....

```
import numpy as np
def main():
    x = np.array(list(range(1, 900000*1)))
    ylength = len(x) // 5
    y = []
    for i in range(ylength):
        y.append(np.sum(x[5*(i+1)]))

    print(y)
    print(len(y))

main()
✓ 11.0s
```



```
def main():
    x = np.array(list(range(1, 900000*1)))
    xsums = np.cumsum(x)
    y = xsums[list(range(4, len(xsums), 5))]

    print(y)
    print(len(y))

main()
✓ 0.0s
```

随着数据量扩大，for循环方法计算时间开始高于向量化方法

核心思想: for loop → Vectorize

从for循环到向量化转变

Pytorch简介及相关数学原理与算法

● 数学基础与算法

□ 向量化和矩阵运算

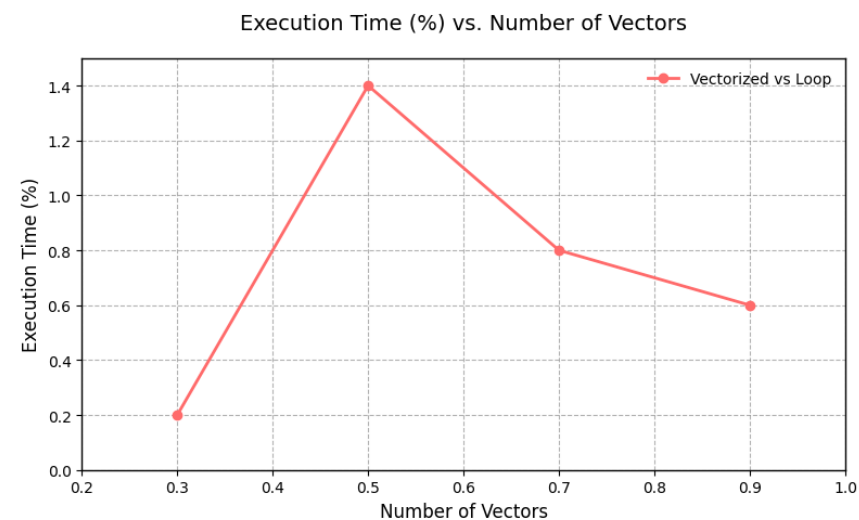
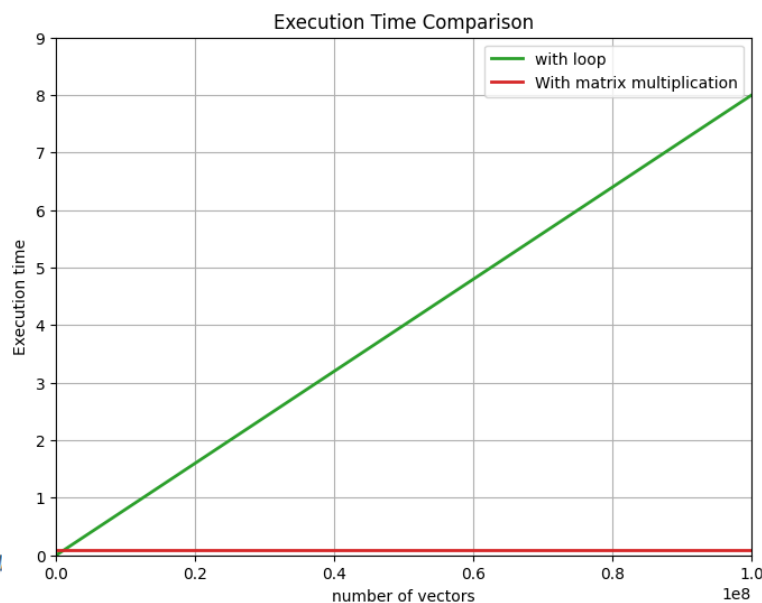
➤ 向量化的意义

计算性能的提升

$$1. \vec{v}_1 = (a_1, a_2, \dots, a_n)$$

$$2. \vec{v}_2 = (b_1, b_2, \dots, b_n)$$

$$3. \text{res} = (a_1 \times b_1, a_2 \times b_2, \dots, a_n \times b_n)$$



这里用 $\vec{v}$ 表示向量，“ $\times$ ”表示对应元素相乘，如果在向量点乘情境下，通常写作 $\vec{v}_1 \cdot \vec{v}_2 = \sum_{i=1}^n a_i b_i$

Pytorch简介及相关数学原理与算法

- 数学基础与算法

- 向量化和矩阵运算

- 向量化的意义

利于并行/分布式运算



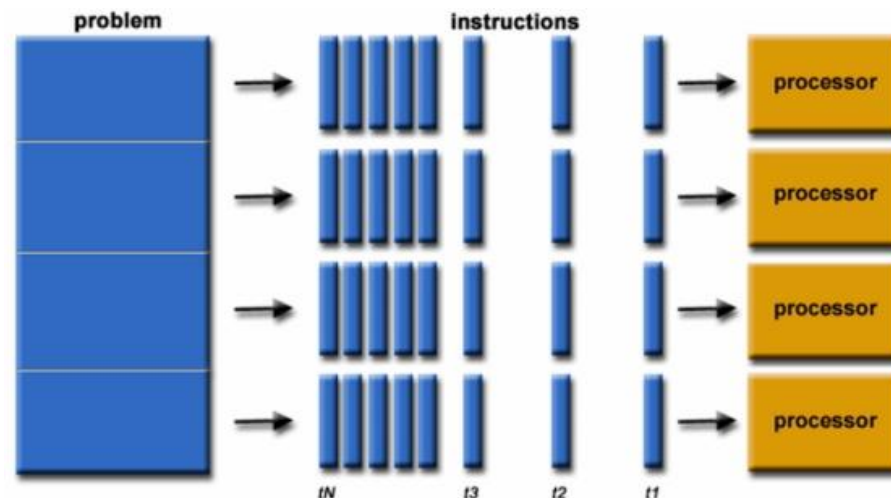
在深度学习中，并行运算和分布式运算是用来加速模型训练和推理的关键技术，特别是在模型参数量巨大、数据集庞大时尤为重要

并行运算是指在单台机器上，利用多个处理器核心（如CPU多核、GPU多卡）同时处理多个任务或将一个任务拆解后并行执行。

分布式运算是指在多台机器（节点）上，每台机器有自己的CPU/GPU，通过网络通信来完成训练任务。



A GPU has more **Arithmetic Logic Units (ALU)** than a typical CPU. Increased **ability** to process **simple operations in parallel**



■ Pytorch简介及相关数学原理与算法

● 数学基础与算法

□ 向量化和卷积运算

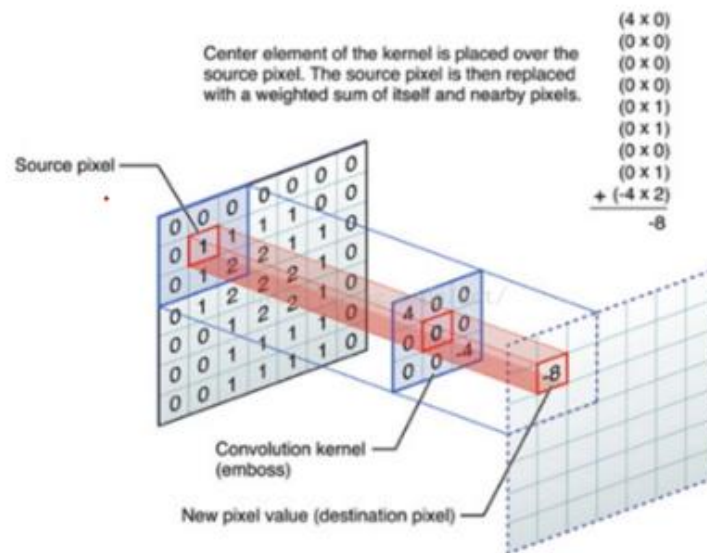
➤ 卷积运算

卷积运算（Convolution Operation）是深度学习中，尤其是在卷积神经网络（CNN）**中，广泛使用的一种数学运算。它的主要作用是提取输入数据（如图像）的局部特征，并且能够保留空间结构信息。

卷积本质上是加权求和的过程，通过将一个小过滤器（或叫卷积核）滑动（或称为“卷积”）在输入数据上，逐步计算出一个新的特征图（feature map）。

对于图像数据，卷积运算通常涉及以下几个步骤：

1. **卷积核 (Filter)**：选择一个小的矩阵（比如3x3、5x5的矩阵），它会在输入图像上滑动，逐位置与图像的局部区域进行加权求和。
2. **滑动窗口**：卷积核在图像上滑动，每次在一个局部区域（例如3x3的区域）与卷积核的对应元素相乘，然后将这些乘积相加，得到一个数值作为该位置的输出。
3. **步幅 (Stride)**：决定了卷积核在输入图像上每次滑动的距离。例如，步幅为1时，卷积核每次滑动一个像素；步幅为2时，卷积核每次跳过一个像素。
4. **填充 (Padding)**：为了保持输出图像的尺寸或防止边界效应，通常对图像边缘做零填充，使卷积核可以覆盖到图像的所有区域。



■ Pytorch简介及相关数学原理与算法

● 数学基础与算法

□ 向量化和卷积运算

➤ 卷积操作例子

假设输入图像 X 是一个 5×5 的矩阵，卷积核 W 是一个 3×3 的矩阵，步幅为1，填充为0。卷积运算可以这样表示：

输入图像 X ：

$$\begin{bmatrix} 1 & 2 & 3 & 0 & 1 \\ 4 & 5 & 6 & 1 & 2 \\ 7 & 8 & 9 & 2 & 3 \\ 1 & 3 & 4 & 3 & 5 \\ 2 & 6 & 7 & 4 & 6 \end{bmatrix}$$

卷积核 W ：

$$\begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$$

计算卷积：

卷积核会从图像的左上角开始，取一个 3×3 的区域，与卷积核按元素相乘，然后求和。接着滑动卷积核到下一个位置，重复这个过程。假设第一个区域是图像的左上角：

$$\text{区域} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

卷积结果是：

$$y = (1 \times 1 + 0 \times 2 + (-1) \times 3) + (2 \times 4 + 0 \times 5 + (-2) \times 6) + (1 \times 7 + 0 \times 8 + (-1) \times 9) = 1 - 3 + 8 - 12 + 7 - 9 = -8$$

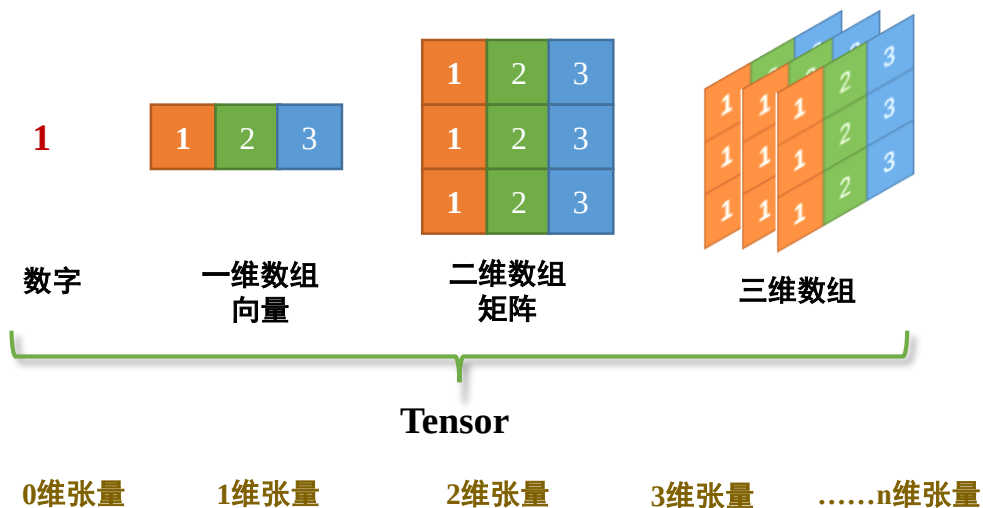
这样得到的一个数值 -8 就是输出特征图的第一个位置的值。继续对图像进行相同的运算，就可以得到整个输出特征图。

Pytorch简介及相关数学原理与算法

● 数学基础与算法

□ 张量运算

- 张量 (Tensor) 机器学习算法是和数学高度相关的领域



标量 (0维张量)

```
import torch

# 创建标量 (单个数值)
scalar = torch.tensor(3.14) # 方法1: 直接传入数值
scalar_from_list = torch.tensor([3.14]) # 方法2: 传入单元素列表

print("标量:", scalar)
print("形状:", scalar.shape) # 输出: torch.Size([])
print("值:", scalar.item()) # 获取Python数值
```

标量: tensor(3.1400)
形状: torch.Size([])
值: 3.140000104904175

1维张量 (向量)

```
# 创建1维张量 (向量)
vector = torch.tensor([1.0, 2.0, 3.0]) # 从列表创建
vector_arange = torch.arange(0, 5) # 生成0-4的整数序列
vector_zeros = torch.zeros(3) # 创建全零向量
vector_ones = torch.ones(3) # 创建全1向量
vector_rand = torch.rand(3) # 创建随机值向量 (0-1均匀分布)

print("\n1维张量:", vector)
print("形状:", vector.shape) # 输出: torch.Size([3])
```

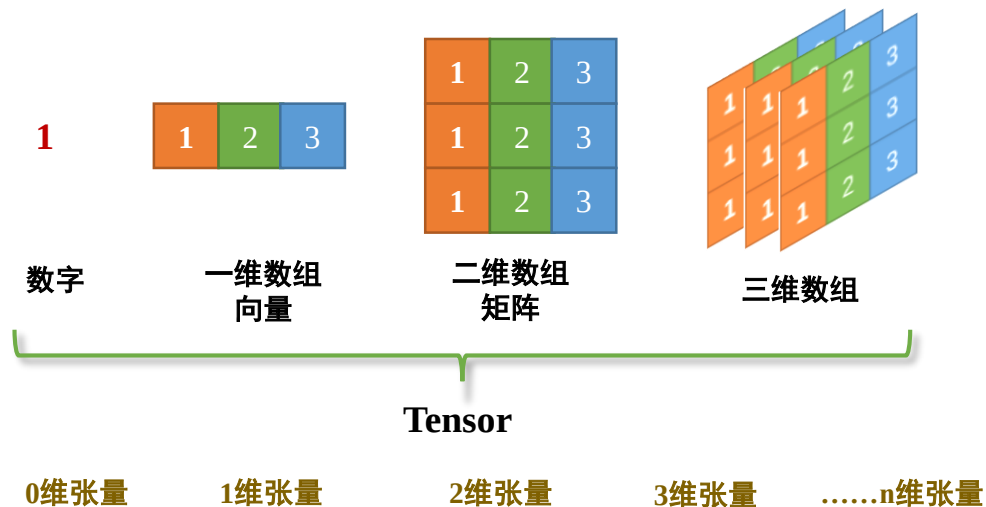
1维张量: tensor([1., 2., 3.])
形状: torch.Size([3])

■ Pytorch简介及相关数学原理与算法

● 数学基础与算法

□ 张量运算

- 张量 (Tensor) 机器学习算法是和数学高度相关的领域



2维张量 (矩阵)

```
# 创建2维张量 (矩阵)
matrix = torch.tensor([[1, 2], [3, 4], [5, 6]]) # 从嵌套列表创建
matrix_zeros = torch.zeros(2, 3) # 2行3列全零矩阵
matrix_ones = torch.ones(2, 3) # 2行3列全1矩阵
matrix_eye = torch.eye(3) # 3x3单位矩阵
matrix_rand = torch.rand(2, 3) # 2x3随机矩阵

print("\n2维张量:", matrix)
print("形状:", matrix.shape) # 输出: torch.Size([3, 2])
```

2维张量: tensor([[1, 2],
[3, 4],
[5, 6]])
形状: torch.Size([3, 2])

3维张量

```
# 创建3维张量
tensor_3d = torch.tensor([[[1, 2], [3, 4]],  
[[5, 6], [7, 8]]]) # 从3层嵌套列表创建
tensor_zeros = torch.zeros(2, 3, 4) # 2个3x4的零矩阵
tensor_randn = torch.randn(2, 3, 4) # 2个3x4的正态分布随机矩阵
tensor_range = torch.arange(24).reshape(2, 3, 4) # 重塑为3维张量

print("\n3维张量:", tensor_3d)
print("形状:", tensor_3d.shape) # 输出: torch.Size([2, 2, 2])
```

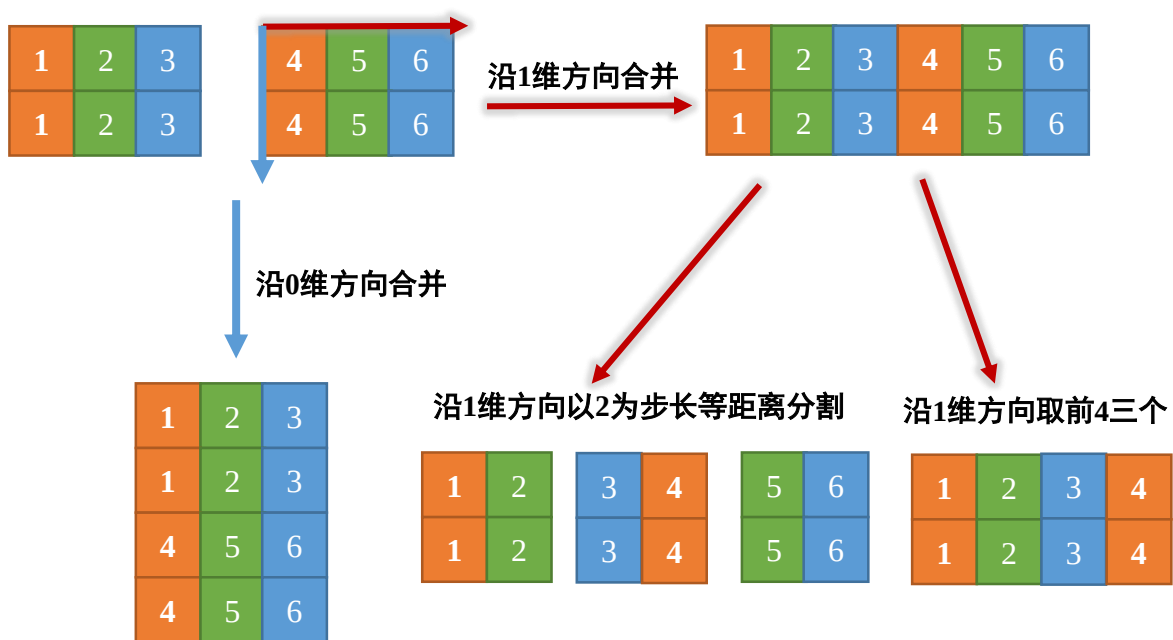
3维张量: tensor([[[1, 2],
[3, 4]],
[[5, 6],
[7, 8]]])
形状: torch.Size([2, 2, 2])

Pytorch简介及相关数学原理与算法

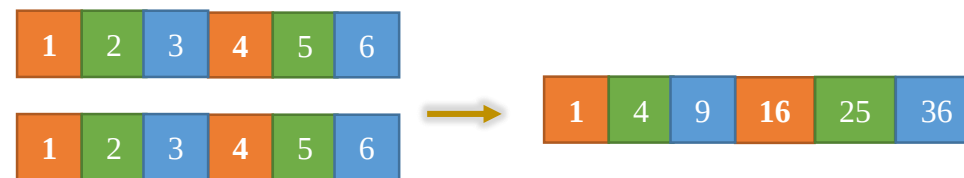
数学基础与算法

张量运算

张量拼接



哈达玛积-Hadamard Product (按位相乘)



内积求和



`torch.tensor` `torch.cat` `torch.split`

`merged_dim1[:, :4]` `torch.mul` `torch.mm`