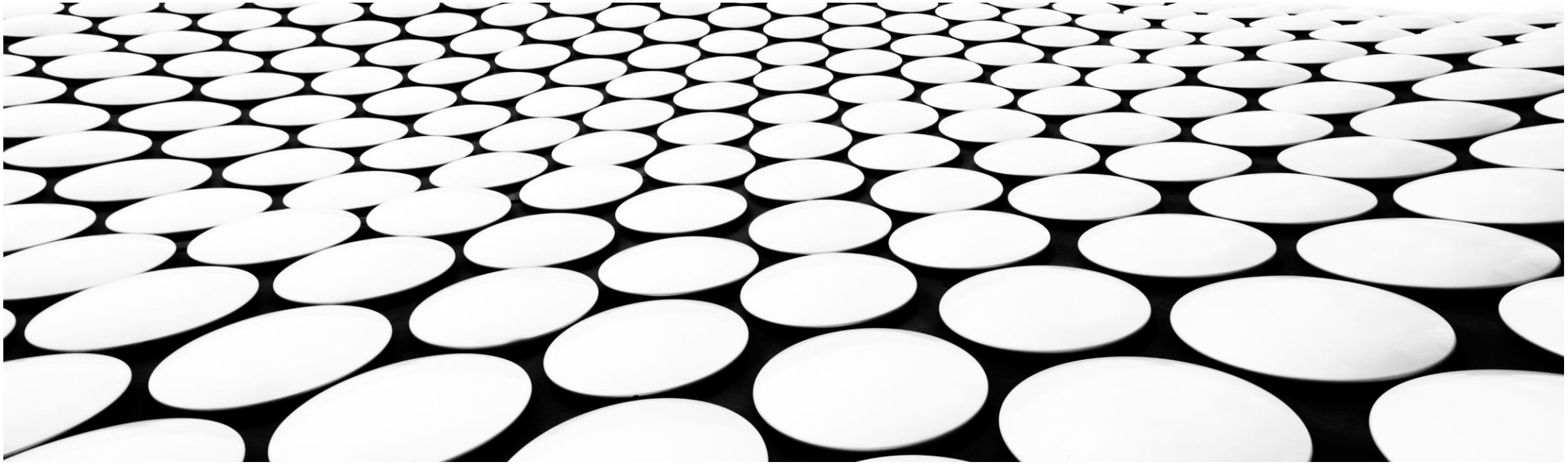
深度学习

邱怡轩



今天的主题

- 常用优化方法简介
- 深度学习计算环境

优化方法

传统模型

- 在传统统计模型中，经常使用牛顿法迭代
- 同时利用一阶导和二阶导的信息
- 如果参数数量是 p
- 一阶导是梯度 g，大小为 $p \times 1$
- 二阶导是 Hessian 矩阵 H，大小为 $p \times p$
- 牛顿法需计算 $H^{-1}g$，复杂度为 $O(p^3)$

一阶方法

- 对于深度学习模型，参数数量非常多
- 几乎只能依赖于梯度
- 通常称为一阶优化方法

GD

- Gradient descent
- 梯度下降法，或最速下降法

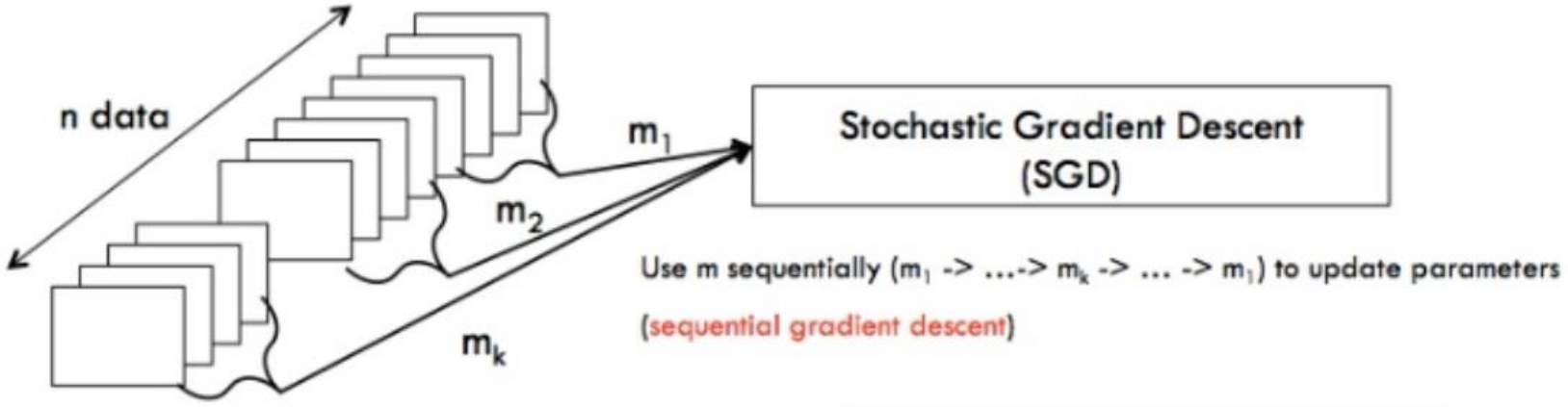
- $\theta^{(k+1)} = \theta^{(k)} - \eta^{(k)} \cdot \nabla_\theta L(\theta^{(k)})$
- η 称为步长或学习率
- $\nabla_\theta L(\theta^{(k)})$ 是精确的梯度，由所有观测计算

SGD

- Stochastic gradient descent
- 随机梯度下降

- $\theta^{(k+1)} = \theta^{(k)} - \eta^{(k)} \cdot \nabla_\theta l(\theta^{(k)})$
- $\nabla_\theta l(\theta^{(k)})$ 是 $\nabla_\theta L(\theta^{(k)})$ 的一个无偏估计
- 例如通过 mini-batch 计算而来

Mini-batch


n data
m_1
m_2
m_k
Stochastic Gradient Descent
(SGD)
Use m sequentially (m_1 -> ...-> m_k -> ... -> m_1) to update parameters
(sequential gradient descent)

思考题

- 数据共有 n 个观测，每个 mini-batch 大小为 m，如有剩余则单独形成一个 mini-batch
- 如何简单计算 mini-batch 的数量?
- 用 Python 实现

SGD

- SGD 的重要意义在于，即使利用随机的、不精确的梯度
- 在一定条件下也可以保证收敛到导数为0的点
- $\sum_{k=1}^{\infty} \eta^{(k)} = \infty,\ \sum_{k=1}^{\infty} \left(\eta^{(k)}\right)^2 < \infty$
- 由统计学家 Herbert Robbins 和 Sutton Monro 在1951年提出
- A Stochastic Approximation Method

SGD

- $\eta^{(k)}$ 的选取
- 令 $S_n = \frac{1}{1} + \frac{1}{2} + \cdots + \frac{1}{n}$，$T_n = \frac{1}{1^2} + \frac{1}{2^2} + \cdots + \frac{1}{n^2}$
- 则 $S_n - \log(n) \to 0.577216\ldots$，故 $S_n \to \infty$
- $T_n \to \frac{\pi^2}{6} < \infty$

常用模板

```
obs_id = np.arange(n)  # [0, 1, ..., n-1]
# Run the whole data set `nepoch` times
for i in range(nepoch):
    # Shuffle observation IDs
    np.random.shuffle(obs_id)

    # Update on mini-batches
    for j in range(0, n, batch_size):
        # Create mini-batch
        x_mini_batch = x[obs_id[j:(j + batch_size)]]
        # Compute loss
        loss = model(x_mini_batch)
        # Compute gradient and update parameters
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
```

改进 SGD

- SGD 虽然具有较好的理论性质
- 但在实际中会遇到各种挑战，如：
  - 确定合适的学习率 η，过大导致优化不收敛，过小耗费大量迭代次数
  - 对每个参数使用了相同的学习率 η

改进SGD

改进 SGD

- SGD 虽然具有较好的理论性质
- 但在实际中会遇到各种挑战，如：
  - 确定合适的学习率 η，过大导致优化不收敛，过小耗费大量迭代次数
  - 对每个参数使用了相同的学习率 η

Adagrad

- 核心思想是对每一个参数计算一个单独的 η
- 令 $g_i^{(k)}$ 为第 i 个参数在第 k 次迭代的导数
- SGD 即为 $\theta_i^{(k+1)} = \theta_i^{(k)} - \eta \cdot g_i^{(k)}$
- Adagrad 为 $\theta_i^{(k+1)} = \theta_i^{(k)} - \frac{\eta}{\sqrt{G_i^{(k)} + \varepsilon}} \cdot g_i^{(k)}$
- 其中 $G_i^{(k)} = \left(g_i^{(1)}\right)^2 + \cdots + \left(g_i^{(k)}\right)^2$

Adagrad

- 优点：
  - 学习率自动衰减
  - 每个参数使用自适应的学习率
- 缺点：
  - 学习率衰减非常快，后期动力不足

RMSprop

- 对 Adagrad 加以改进
- 使用滑动平均计算 G_i
- $G_i^{(k)} = \gamma G_i^{(k-1)} + (1-\gamma)\left(g_i^{(k)}\right)^2$
- $\theta_i^{(k+1)} = \theta_i^{(k)} - \frac{\eta}{\sqrt{G_i^{(k)}+\varepsilon}} \cdot g_i^{(k)}$

Adadelta

- 与 RMSprop 独立地对 Adagrad 加以改进
- 加入分子的滑动平均，保持量纲一致
- $G_i^{(k)} = \gamma G_i^{(k-1)} + (1-\gamma)\left(g_i^{(k)}\right)^2$
- $\theta_i^{(k+1)} = \theta_i^{(k)} \underbrace{- \frac{\sqrt{D_i^{(k)}+\varepsilon}}{\sqrt{G_i^{(k)}+\varepsilon}} \cdot g_i^{(k)}}_{\Delta\theta_i^{(k)}}$
- 分子为 $D_i^{(k)} = \gamma D_i^{(k-1)} + (1-\gamma)\left(\Delta\theta_i^{(k)}\right)^2$

Adam

- 对梯度也进行滑动平均
- $m_i^{(k)} = \beta_1 m_i^{(k-1)} + (1-\beta_1) g_i^{(k)}$
- $v_i^{(k)} = \beta_2 v_i^{(k-1)} + (1-\beta_2) \left(g_i^{(k)}\right)^2$
- 修正偏差 $\hat{m}_i^{(k)} = m_i^{(k)} / (1-\beta_1^k)$，$\hat{v}_i^{(k)} = v_i^{(k)} / (1-\beta_2^k)$
- $\theta_i^{(k+1)} = \theta_i^{(k)} - \frac{\eta}{\sqrt{\hat{v}_i^{(k)}} + \varepsilon} \cdot \hat{m}_i^{(k)}$

对比

- 理论性质的研究尚在进行
- 实际应用中往往比 SGD 更快收敛
- 前期用自适应的方法，后期用 SGD

- 见 `lec8-optimizer.ipynb`

Ali Rahimi 的演讲

https://www.bilibili.com/video/BV1BW411Y78t
13:27-16:00

计算环境

计算效率

- 神经网络的计算效率由众多不同的因素决定

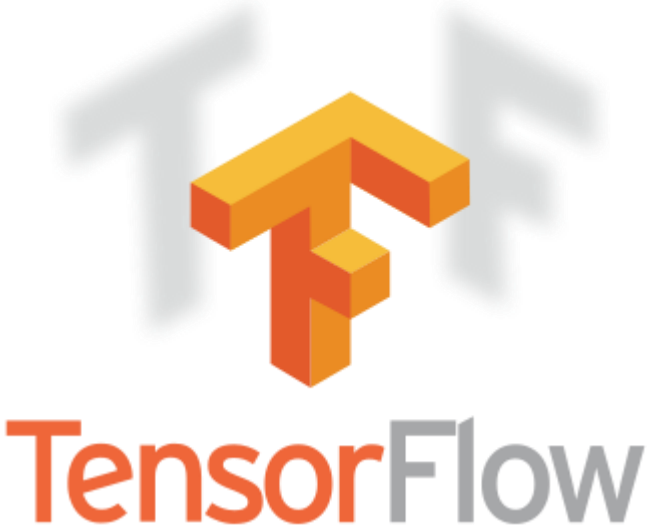
- 算法收敛速度
- 软件实现
- 硬件设备
- ……

软件

- 主流的深度学习框架都对核心运算，如矩阵乘法、卷积等进行了高度优化

Tensorflow

- https://www.tensorflow.org/
- 免费、开源
- 主要由 Google 开发维护



PyTorch

- https://pytorch.org/
- 免费、开源
- 主要由 Facebook 开发维护

MXNet

- https://mxnet.apache.org/
- 免费、开源
- 由陈天奇、李沐等人发起
- 当前主要由 Apache 软件基金会和 Amazon 团队开发维护

MindSpore

- https://www.mindspore.cn/
- 免费、开源
- 由华为开发维护

对比

- 不同的软件框架之间并无绝对的优劣之分
- 语法通常非常相似
- 个人使用可依喜好选择
- 商业应用往往考虑兼容性和维护成本等因素

硬件

- 像绝大多数应用程序一样，神经网络模型可以运行在 CPU 上
- 随着深度学习的流行，更多专用设备如 GPU、TPU 等被用来加速计算

硬件

- GPU 的优势在于可以高度并行，特别适合神经网络的结构

	GeForce RTX 5090	GeForce RTX 5080	GeForce RTX 5070 Ti
GPU Engine Specs:			
NVIDIA CUDA® Cores	21760	10752	8960
Shader Cores	Blackwell	Blackwell	Blackwell
Tensor Cores (AI)	5th Generation 3352 AI TOPS	5th Generation 1801 AI TOPS	5th Generation 1406 AI TOPS
Ray Tracing Cores	4th Generation 318 TFLOPS	4th Generation 171 TFLOPS	4th Generation 133 TFLOPS
Boost Clock (GHz)	2.41	2.62	2.45
Base Clock (GHz)	2.01	2.30	2.30

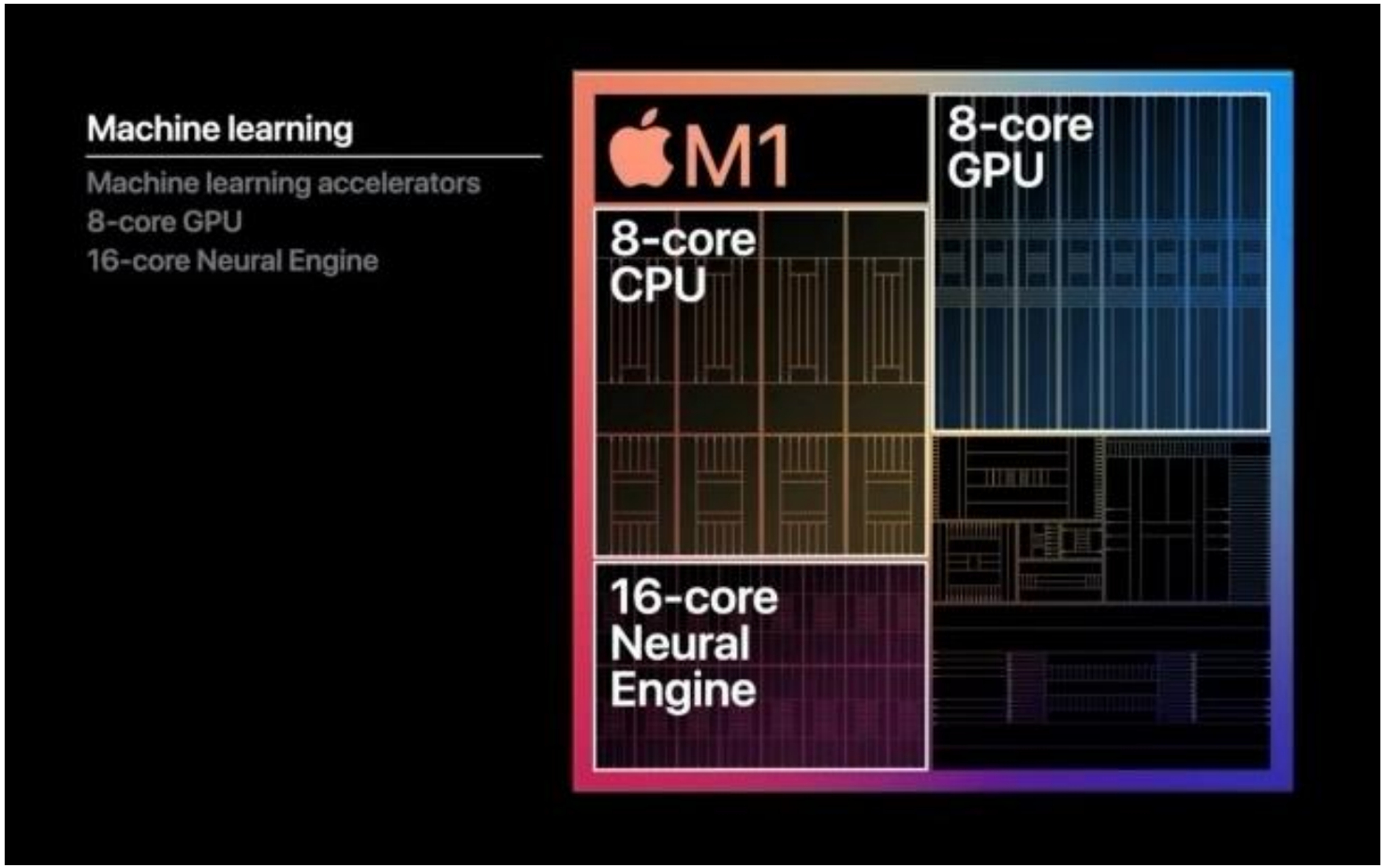
硬件

- 但并非 GPU 上运行的神经网络就一定比 CPU 上快
- 数据传输到 GPU 的计算核心需要时间
- GPU 单核的性能一般不如 CPU
- 数据量少、网络简单时并行效果不明显

- 对于较复杂的网络，GPU 的运算效率往往有很大的提升

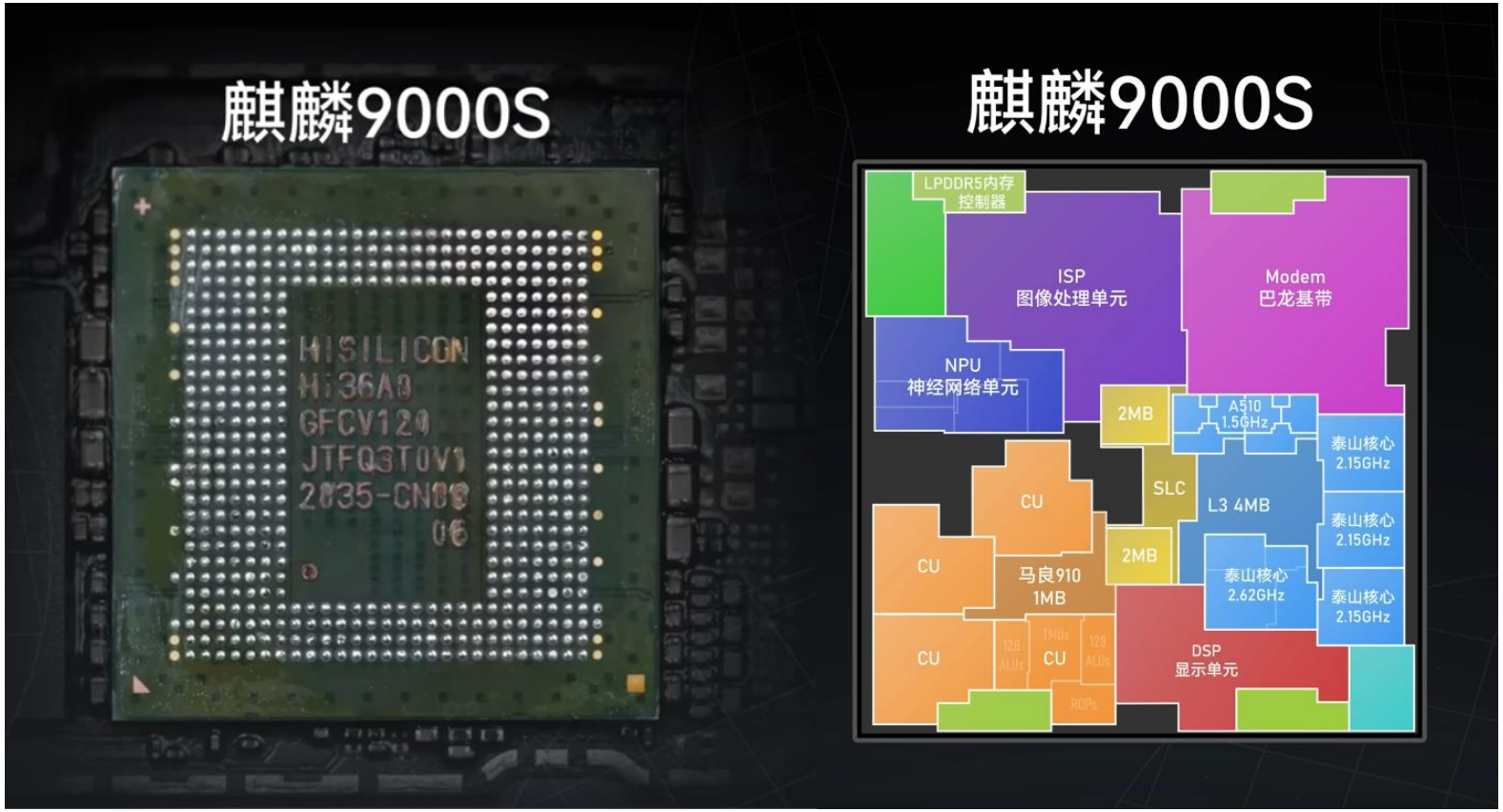
硬件

- 一些集成芯片上还带有专门的神经网络处理器



硬件

- 一些集成芯片上还带有专门的神经网络处理器



SUFE AI Lab

- https://ai.sufe.edu.cn/
- 上财校内计算资源
- 可使用 GPU

Google Colab

- https://colab.research.google.com/
- 免费计算资源（免费版有时间限制）
- 可使用 GPU