计算机基础

csapp

计算机网络

OSI 5层，TCP可靠传输，VPN, port , socket...

操作系统

进程，线程，调度算法，并发控制，地址转换，页表，cache...

数学基础

高数

微分，偏导，链式法则....等相关定义和计算

概率论

贝叶斯公式

参考博客：https://zhuanlan.zhihu.com/p/26262151

已知某个事件发生的概率，并且知道在该事件发生的条件下的一些事件发生的概率，构建贝叶斯模型。根据“一些事件”是否发生和“某个事件”发生的概率，去预测在”一些事情“的特征已经具备的条件下，”某个事件“发生的概率

大数定理

参考博客：https://zhuanlan.zhihu.com/p/259280292

大数定理讨论的是多个随机变量的平均$\frac{1}{n}\sum_{i=1}^{n}X_i$的渐进性质

对于一系列随机变量$\{X_n\}$，设每个随机变量都有期望。由于随机变量之和$\sum_{i=1}^{n}X_i$很有可能发散到无穷大，我们转而考虑随机变量的均值$\overline{X_n}=\frac{1}{n}\sum_{i=1}^{n}X_i$和其期望$\mathbb{E}[\overline{X_n}]$之间的距离。若$X_n$满足一定条件，当n足够大时，这个距离会以非常大的概率接近0，这就是大数定律的主要思想。

- 定义: 对于任意$\epsilon>0$, 若恒有$lim_{n->+\infty}P(|\overline{X_n}-\mathbb{E}(\overline{X_n})|<\epsilon)=1$, 则称随机变量序列$\{X_n\}$满足大数定理

中心极限定理

中心极限定理讨论的是独立随机变量和$Y_n=\sum_{i=1}^{n}X_i$的极限分布

Y_n可以看成是很多微小的随机因素$X_1,X_2,...X_n$之和,n很大，我们关心在什么条件下面Y_n的极限分布是正态分布

- 独立同分布中心极限定理（林德伯格-列维中心极限定理)

  如果随机变量$X_1,X_2....X_n$相互独立，且分布相同，他们的数学期望μ和方差σ^2一致,则随机变量$Y_n=\frac{\sum_{i=1}^{n}X_i-n\mu}{\sqrt{n}\sigma}$,当n较大的时候$Y_n\sim N(0,1)$,近似标准正态分布，即$\sum_{i=1}^{n}X_i\sim N(n\mu,n\sigma^2)$
- 二项分布中心极限定理(棣莫弗-拉普拉斯中心极限定理)

  X是n次伯努利实验中事件A出现的次数,p是每次时间A发生的概率，即$X\sim B(n,p)$

当n较大时 $X \sim N(np, np(1-p))$

最大似然估计

参考博客: https://zhuanlan.zhihu.com/p/26614750

- 极大似然估计，通俗理解来说，**就是利用已知的样本结果信息，反推最具有可能（最大概率）导致这些样本结果出现的模型参数值！**
- 具体步骤:

1. 给据给定样本和总体分布类型构造似然函数(目的就是让已知的样本能真实反映总体的分布，也就是，似然函数必须要最大)
2. 对似然函数取对数，求导数，令导数=0,此时根据等式可以求出估计的参数

线性代数

视频

- 3B1b 视频讲解，如何理解线性代数里面的操作

博客

- 3B1b博客笔记
- 顶级的线性代数的理解

AI

Python 基础

基本语法和概念

参考博客：https://www.liujiangblog.com/course/python/78

基础的第三方库

matplotlib,numpy,sklearn等，看对应的手册和文档

拓展学习

threading

- 线程

  线程是比进程更小的调度单位。具体来说，一个程序的运行状态称做进程，进程被译码称多个指令，由1个或者多个线程分别承担一部分指令的工作，这样指令与指令之间执行的时候，发生切换的是线程。

  此时cpu的轮转就不再局限于进程之间，而是可能在同一个进程的不同线程之间的切换，或者不同进程之间的线程的切换。

  多线程程序编写的核心在与共享数据的保护和不同线程之间的通信

- 如果同一个进程的线程之间由需要共享的资源，如何实现，如何避免资源请求冲突（实现互斥锁）？

multiprocess

- 进程

  每个进程都拥有一个GIL，这样子多个进程之间就不会受一个GIL的限制，可能并发性高

- 思考

  同样的，如何实现多个进程之间的资源共享?

async

- 协程

  又称作微线程，相比于线程，线程之间的切换是由**程序本身控制的**，省去了切换进程之间的开销

- 思考

  协程适合用于那种场景呢？为什么，试着使用协程写一个小型爬虫爬取任意一个网站的图片吧

传统机器学习

对于这些传统的机器学习，聚焦于这些算法的思想就可以了

KNN

- 主要思想：

  在某个点最近的N个邻居中，哪一类的类别最大，就将该点分类为哪一个类别

Kmeans

- 主要思想: 无监督聚类，初始化聚类中心，每个点被分配到离他最近的聚类中心上，所有点被分配完之后，更新聚类中心，反复迭代，直到满足条件为止！

SVM

- 主要思想：最大化类别之间的间隔

PCA

参考博客:

- https://www.zhihu.com/question/41120789/answer/481966094

逻辑回归

参考博客:

- https://zhuanlan.zhihu.com/p/74874291

梯度下降法证明

- 最优化问题,局部最小值不一定是全局最优解，通过添加一些随机噪声/扰动能够跳出局部最优解
- 训练误差指的是在训练集上的表现，泛化误差指的是在全部数据集上的表现，有时可以说是在测试集上的表现

一维梯度下降

考虑一维函数的随机梯度下降，一个连续可微实值函数$f:\mathbb{R}\rightarrow\mathbb{R}$利用泰勒展开可以得到

$$f(x+\epsilon)=f(x)+\epsilon f'(x)+O(\epsilon^2)$$

即在一阶近似中，$f(x+\epsilon)$可通过x出的函数值$f(x)$和一阶导数$f'(x)$近似得出。我们可以假定负梯度方向上移动的ϵ会减少f。为了简单起见，我们选择固定的步长$\eta>0$,然后令$\epsilon=-\eta f'(x)$，然后将其代入泰勒展开式可以得到

$$f(x-\eta f'(x))=f(x)-\eta f'^2(x)+O(\eta^2 f'^2(x))$$

如果$f'(x)\neq 0$导数并没有消失，那么可以将上面的泰勒展开式继续展开，因为$\eta^2 f'^2(x)>0$。此外，我们也可以令η小到让高阶函数不那么相关，因此

$$f(x-\eta f'(x))\approx f(x)$$

这就意味着我们可以使用$x\leftarrow x-\eta f'(x)$来迭代x。直到某个终止条件停止迭代

多维梯度下降

和一维梯度下降类似的过程,考虑变量$\mathbf{x}=[x_1,x_2,\dots x_d]^T$ 的情况。即目标函数$f:\mathbb{R}^d\rightarrow\mathbb{R}$，将向量映射为标量，相应的他的梯度也是多元的，由d个偏导数组成的向量:

$$\nabla f=[\frac{\partial f((x))}{\partial x_1},\frac{\partial f(\mathbf{x})}{\partial x_2},\dots\frac{\partial f((x))}{\partial x_d}]^T$$

梯度中的每个偏导数$\partial f/\partial x_i$代表了当输入了x_i时f在$\mathbf{x}$处的变化率。和单变量一样，考虑使用泰勒展开式来近似

$$f(\mathbf{x}+\epsilon)=f(\mathbf{x})+\epsilon^T\nabla f(\mathbf{x})+O(||\epsilon^2||)$$

通过$\mathbf{x}\leftarrow\mathbf{x}-\eta\nabla f(\mathbf{x})$来迭代求解

深度学习

- PyTorch深度学习快速入门教程 b站小土堆
- 跟李沐学AI b站李沐
- 《动手学深度学习》 — 动手学深度学习 2.0.0 documentation (d2l.ai)

第一遍重在理解，整个深度学习任务的流程，理解计算图，反向传播的过程，以及矩阵计算

第二遍重在代码编写，深入理解每个网络设计的原理，常见的接口最好都熟悉

第三遍深入torch框架

卷积神经网络CNN

这部分看李沐的视频够了

图神经网络GNN

参考博客:

- https://zhuanlan.zhihu.com/p/75307407
- https://github.com/SivilTaram/Graph-Neural-Network-Note

GNN 就是做了这么一件事情：**利用图的节点信息去生成节点（图）的 Embedding 表示**。就是那么一个 Embedding 的方法。

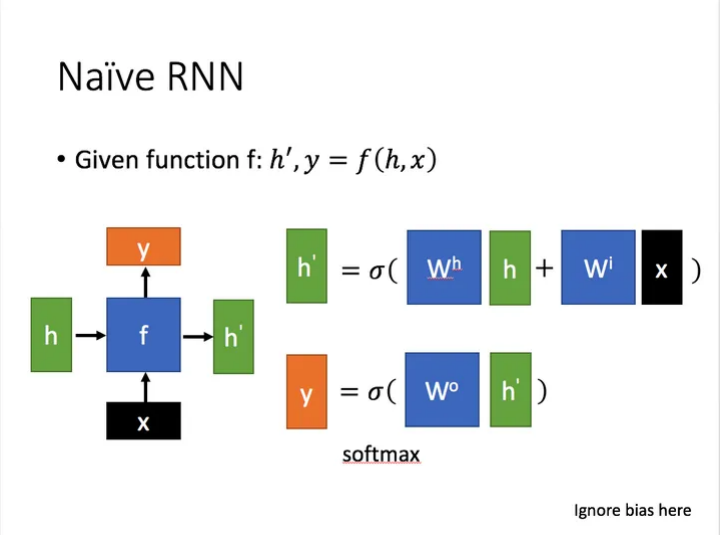
循环神经网络RNN

RNN

参考博客：

- https://zhuanlan.zhihu.com/p/32085405
- https://colah.github.io/posts/2015-08-Understanding-LSTMs/

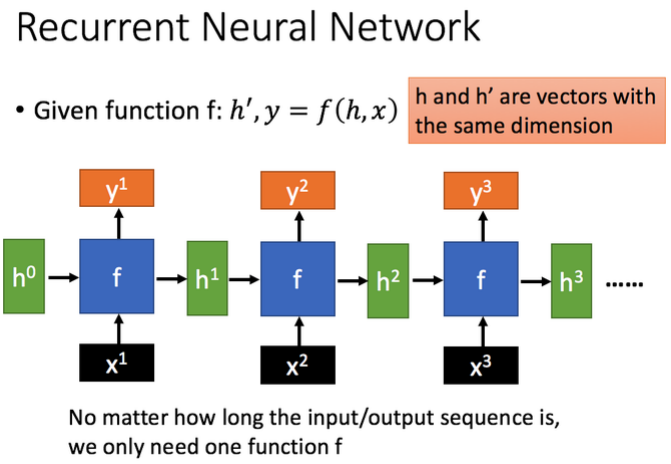
RNN 中的单个神经元如下所示



x表示当前状态的输入, **h**表示接受的上一个节点的输入, **y**是当前状态的输出，**h'**是传递给下一个状态的输入

从上面的图片可以看到，**h'**的计算与当前状态**x**和上一节点的输入**h**有关，**y**的计算通常由**h'**计算得来

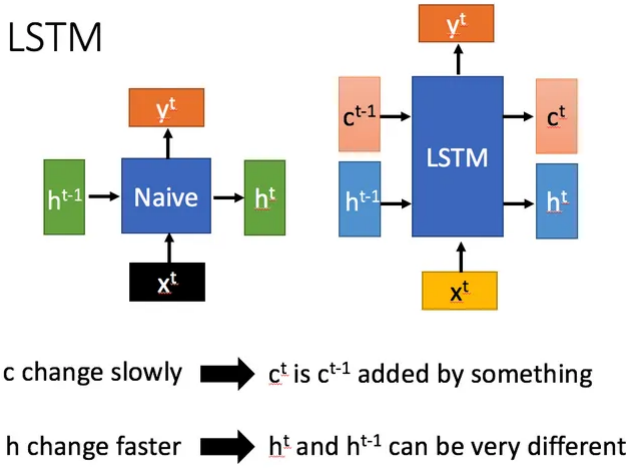
如干个这个样的单元组成一个序列即为**RNN(recurrent neural network)**循环神经网络,如下图所示



LSTM

长短期记忆（Long short-term memory, LSTM）是一种特殊的RNN，主要是为了解决长序列训练过程中的梯度消失和梯度爆炸问题。简单来说，就是相比普通的RNN，LSTM能够在更长的序列中有更好的表现。

LSTM和RNN的输入的区别如下图



相比于RNN只有一个传递状态**h**, LSTM有两个传递状态c^t和h^t, (lstm 的h^t应该对应的是rnn的h^t)

通常c^t是上一个状态传来的c^{t-1}加上某些数值

具体的计算结构如下

1. 遗忘阶段: 计算f_t,选择那些元素需要被遗忘
2. 记忆阶段: 计算i_t和$\tilde{C}_t$,然后将两者按元素相乘，选择那些元素需要被记忆
3. 更新阶段:根据 f_t与c_{t-1}和i_t与$\tilde{C}_t$计算c_t
4. 输出阶段: h_t经过某些变化和c_t 计算当前单元的输出

Transformer

参考博客

- The Illustrated Transformer – Jay Alammar
- Transformer模型详解
- Attention注意力机制与self-attention自注意力机制
- 注意力机制综述
- 张俊林讲解attention
- kv cache
- 为什么用kv cache 不 cache q?
- B站视频讲解:王木头学科学

Llama2

参考博客

- 知乎llama2结构详解

代码仓库:

- llama-factory

Llama 2的模型结构与标准的Transformer Decoder结构基本一致，主要由32个 Transformer Block 组成，不同之处主要包括以下几点：

1. 前置的**RMSNorm**层
2. Q在与K相乘之前，先使用**RoPE**进行位置编码
3. **K V Cache**，并采用**Group Query Attention（GQA)**
4. FeedForward层

RoPE旋转位置编码

- https://zhuanlan.zhihu.com/p/642884818
- https://zhuanlan.zhihu.com/p/647109286

GQA分组注意力查询机制

- MHA, MQA, GQA

大模型训练

参考博客：

- https://zhuanlan.zhihu.com/p/688873027

传统并行手段

了解下torch的通信原语

- 知乎教程
- pytorch文档教程

根据教程完成p2p通信和collective communication

数据并行

参考:

- DP与DDP原理解读
- 原理简单解读和DDP详细使用教程

torch.nn.DP(Data Parallel)

- DP是单进程多线程的形式,torch源码，受python的GIL的限制，至于什么是GIL,回顾python的基础知识

DDP(Data Distributed Parallel)

- 多进程的形式，一般一张显卡对应一个进程,通过多进程，绕过了GIL,性能较多线程可能更好

流水线并行

视频讲解:https://www.bilibili.com/video/BV1v34y1E7zu/?spm_id_from=333.999.0.0

张量并行（模型并行）

就是下面的Megatron

Megatron-LM

参考博客:https://zhuanlan.zhihu.com/p/366906920

视频讲解：https://www.bilibili.com/video/BV1nB4y1R7Yz/?spm_id_from=333.999.0.0

Deepspeed

区别于其他框架的最大特点是Zero

文档:https://www.deepspeed.ai/getting-started/

视频讲解：https://www.bilibili.com/video/BV1tY411g7ZT/?spm_id_from=333.999.0.0

Flashattention

理解原理！现在很多算法已经集成了Flashattention了，大部分不需要自己实现。

大模型推理部署

论文推荐阅读：A Survey on Efficient Inference for Large Language Models

vLLM

仓库：https://github.com/vllm-project/vllm

Light-llm

仓库：https://github.com/ModelTC/lightllm

TensorRT

仓库：https://github.com/NVIDIA/TensorRT-LLM

大模型微调

全参数微调 与 高效参数微调

- 全参数微调: 将预训练模型作为初始化权重，对全部参数都进行更新
- 高效参数微调: 通常指对部分参数进行更新

LoRA

论文地址: https://arxiv.org/pdf/2106.09685

参考博客: https://zhuanlan.zhihu.com/p/623543497

代码仓库:

LoRA，全称 Low-Rank Adaptation

对于预训练模型的参数H，我们在其上面进行微调（参数的更新），假设参数的变化为ΔH，那么更新过后的模型可以表示为$H+\Delta H$

具体一点，对于模型内的某一层的矩阵W，我们假设预训练模型这一层的参数为W_0，假设其变化的参数为ΔW，那么这一层参数上的更新可以表示为$W_0 + \Delta W$。其中$W \in \mathbb{R}^{d\times k}$, 则也有$W_0, \Delta W \in \mathbb{R}^{d\times k}$，（形状要一样的啊，要不然两个矩阵怎么相加）

进一步，ΔW是不是可以表示成两个矩阵相乘的形式呢？。我们假设$\Delta W = AB$，其中$A \in \mathbb{R}^{d\times r}, B \in \mathbb{R}^{r\times k}, r \ll min(d, r)$，那么对于那么这一层参数的更新就可以表示为$W + \Delta W = W + AB$。$r$就是LoRA中的秩序了，通常$r = 1, 2, 3, 4, 8$都不是一个太大的值，所以这就叫低秩。到这里LoRA的主要思想就讲完了。

QLoRA

区别于LoRA,是在训练时进行量化,原理也是可以大致了解

量化技术

GPTQ

论文：https://arxiv.org/abs/2210.17323

代码仓库: https://github.com/IST-DASLab/gptq

AutoGPTQ,这个仓库集成了更多功能，支持很多模型

AWQ

原理大致了解即可

论文:https://arxiv.org/abs/2306.00978

代码仓库:https://github.com/mit-han-lab/llm-awq

上下文缓存技术

如何解决 kv cache过长的问题!

https://arxiv.org/abs/2406.17565

阅读推荐

深入理解pytorch机制

llm-action 大模型实战和技术路线