Beijing Forest Studio
北京理工大学信息系统及安全对抗实验中心

从任务划分就开始与众不同的元学习

硕士研究生 潘珲

2022年10月03日

内容提要

- 背景简介
- 基本概念
- 任务架构
- 优化方向
- 参考文献

内容提要

- 预期收获
  - **1.**了解元学习的基本概念
  - **2.**掌握元学习的基本训练过程
  - **3.**理解元学习与常规网络的不同
  - 4.了解元学习的应用场景

背景介绍

- 好的机器学习模型经常需要大量的数据来进行训练，但人却恰恰相反
- 那么有没有可能让机器学习模型也具有相似的性质呢?
- 如何才能让模型仅仅用少量的数据就学会新的概念和技能呢?
- 什么是元学习呢?

狗

猫

基本概念——版本1

- 1987年，科学家就已经提出了元学习的概念
  - 机器通过与环境进行交互，不断获取信息，进行自我更新调整，从而不断地适应环境，在没有任何人为干预的场景下，机器能自发地适应并且进化，智能化地学会怎么解决遇到的任何任务

- 3090算力能达到35.6 **TFLOPS**

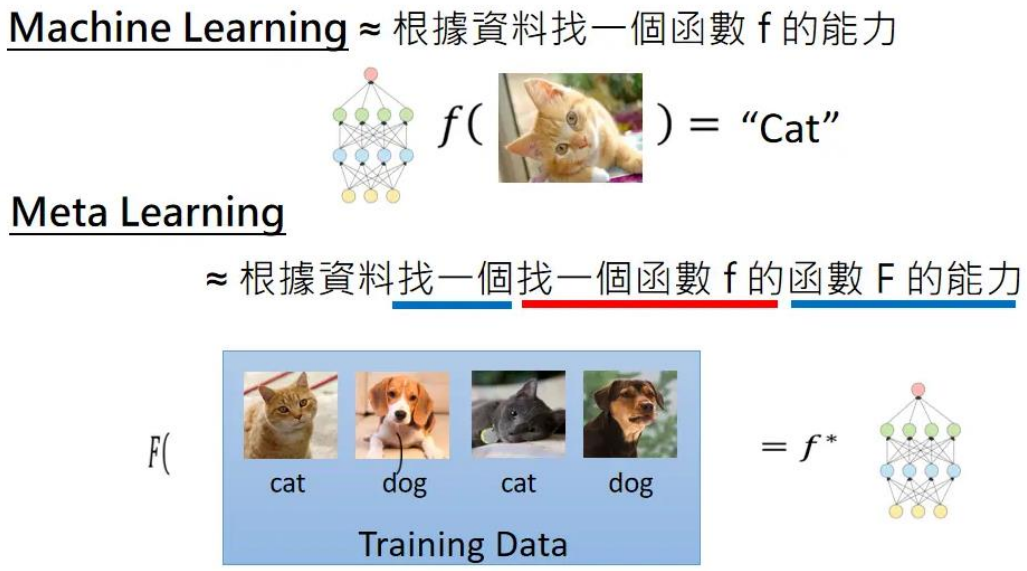
- 智械危机?

基本概念——版本2

- 元学习的概念属于强人工智能的范畴，是指让机器学会学习从而让机器变得更加智能，代替人类完成复杂多变的任务
  - 让机器学会学习，使其具备分析和解决问题的能力， 机器通过完成任务获取经验，提高完成任务的能力
  - 让机器学习模型可以更好地泛化到新领域中， 从而完成差异很大的新任务

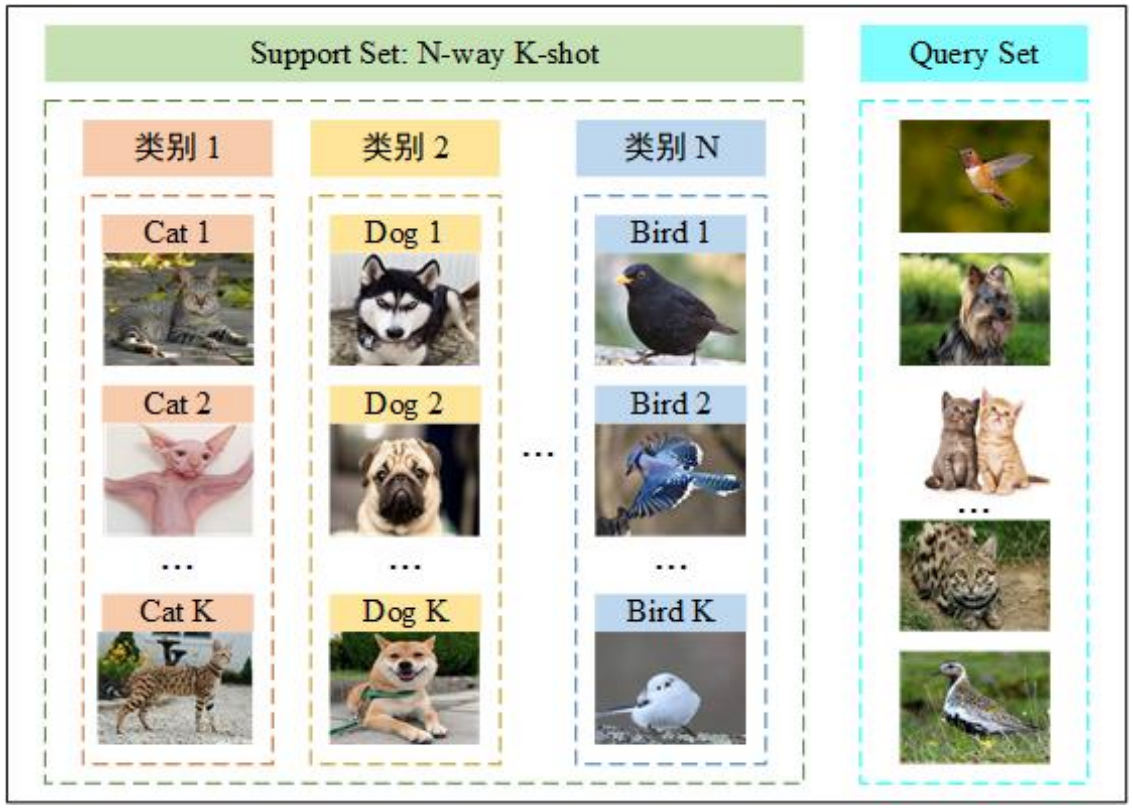
基本概念——版本3

- 元学习 (**Meta-Learning**) 通常被理解为“学会学习 (**Learning-to-Learn**)”
  - 常见的深度学习模型，目的是学习一个用于预测的数学模型
  - 元学习面向的不是学习的结果，而是学习的过程
  - 其学习的不是一个直接用于预测的数学模型，而是学习“如何更快更好地学习一个数学模型 “
- 核心思想
  - 训练一个可以快速适应任务的模型



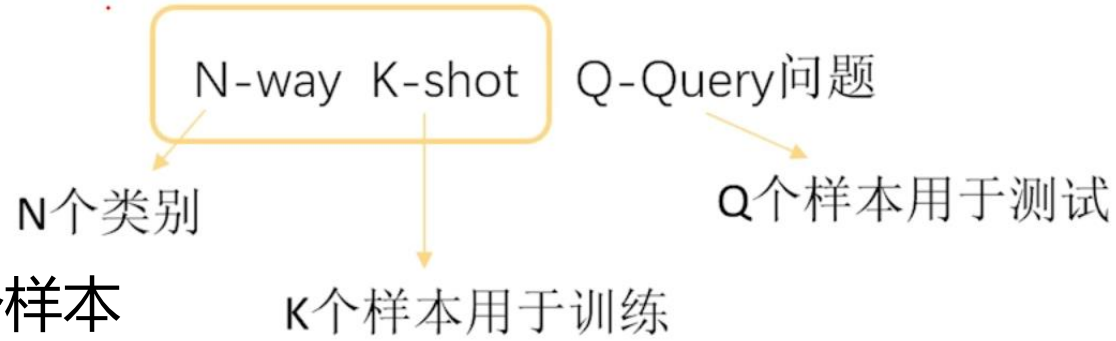
任务构建

- **Task**
  - 元学习的基本单元是任务
    - 元训练任务
    - 元测试任务
  - 每个任务都有自己的训练集和测试集
  - 元训练任务的训练集和测试集一般称为支持集 (**Support Set**) 和查询集 (**Query Set**)
  - 支持集又是一个**N-Way K-Shot**问题，即有**N**个类别，每个类有**K**个样例



任务构建

- **Task**
  - 把已有的数据集切分转化成多个任务
  - 以**mini-imagnet**为例：
  - 训练数据集共有64个类别，每个类别有**600**个样本
  - 构建**5-way 5-shot 1-query**的任务集
    - 每次随机抽取五个类别
    - 每类中抽取**5-shot + 1-query = 6**张图片作为任务的资料库
    - 一个任务共有**5*6=30**张图片
    - 随机抽取**10**万次形成**10**万个任务作为训练任务集
  - 同样的方法也可以构造验证集和测试集

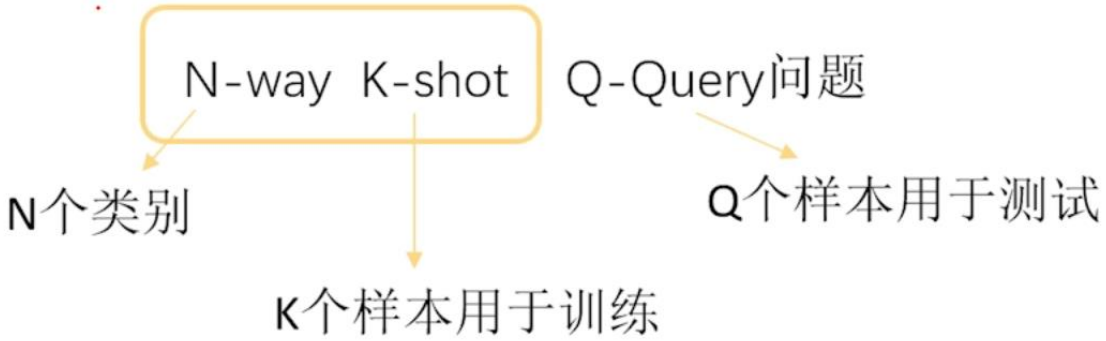


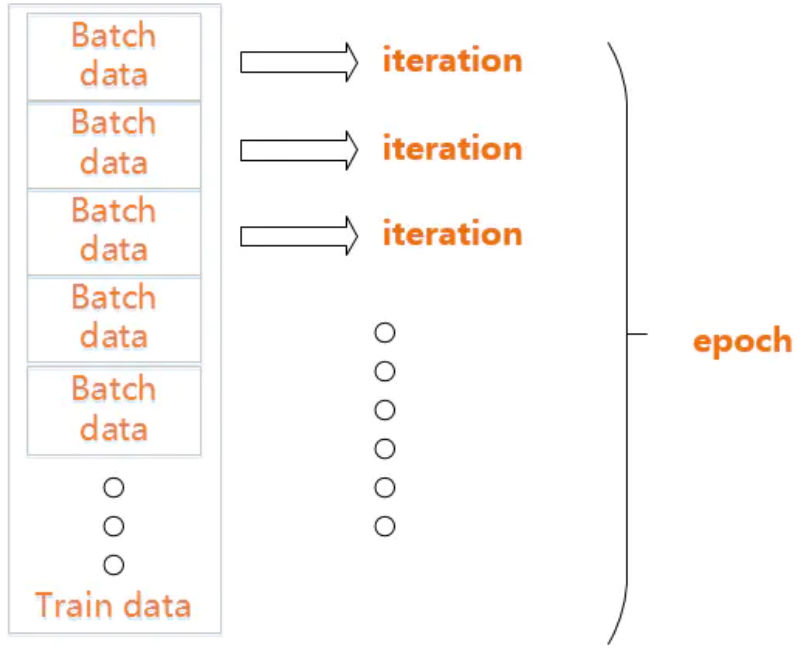
5-way 5-shot 1-Query

Way 1 (C1)	Way 2 (C2)	Way3 (C3)	Way 4 (C4)	Way5 (C5)	
C~11~	C~21~	C~31~	C~41~	C~51~	Support
C~12~	C~22~	C~32~	C~42~	C~52~	Support
C~13~	C~23~	C~33~	C~43~	C~53~	Support
C~14~	C~24~	C~34~	C~44~	C~54~	Support
C~15~	C~25~	C~35~	C~45~	C~55~	Support
C~16~	C~26~	C~36~	C~46~	C~56~	Query

任务构建

- **Epoch**
  - 跑一遍完整数据集
- **Batch**
  - 每迭代一次网络训练的样本数量
- **Task**
  - 假设我们设定**batch_size = 4**
  - 在元训练阶段，按照**batch**输入成**tensor**数据
  - 一个batch有四个任务，每个任务30张图像/标签
  - 用**slice**来切分，分成**support**和**query**
  - 每次都拿这样的**batch**和**task**来训练、优化参数





训练范式

- 元学习本质上是双层优化问题
  - 其中一个优化问题嵌套在另一个优化问题中
- 两层优化问题涉及两个参与器：
  - 外层的参与者是元学习器
  - 内层的参与者是基学习器
- 元学习器的最优决策依赖于基学习器的反应，基学习器自身会优化自己内部的决策
- 这两个层次有各自不同的目标函数、约束条件和决策变量

算法1：模型不确定元学习

输入：$p(T)$:任务调度函数

输入：α, β: 学习率超参数

1：随机初始化θ

2：**while not done do** — 元学习器(外层)

3：使用任务调度函数采样一个batch的tasks: $T_i \sim p(T)$

4：**for all** T_i **do** — 基学习器(内层)

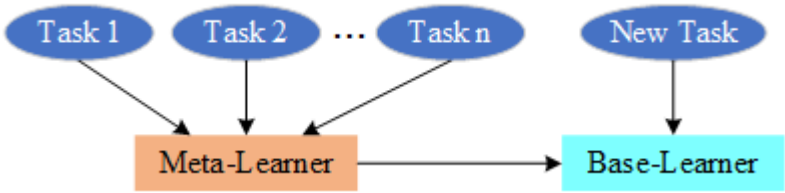
5：计算每类中K个样本的损失函数$\nabla_\theta L_{T_i}(f_\theta)$

6：用梯度下降算法计算适应的参数：$\theta_i' = \theta - \alpha \nabla_\theta L_{T_i}(f_\theta)$

7：**end for** — 支持集

8：更新参数$\theta \leftarrow \theta - \beta \nabla_\theta \sum_{T_i \sim p(T)} L_{T_i}\left(f_{\theta_i'}\right)$ — 查询集

9：**end while**



基学习器&元学习器

- 基学习器是内层中的模型
  - 在单个任务上训练模型，学习任务特性
  - 从元学习器获取对完成单个任务有帮助的经验，包括初始模型和初始参数等
  - 在单个任务上训练完成后，将训练的模型和参数都反馈给元学习器
- 元学习器是外层中的模型
  - 综合多个任务上基学习器训练的结果
  - 对多个任务的共性做归纳，进行快速准确的推理
  - 将推理输送给基学习器，作为初始模型和初始参数值

算法1：模型不确定元学习

输入：$p(T)$:任务调度函数

输入：α, β: 学习率超参数

1：随机初始化θ

2：**while not done do** — 元学习器(外层)

3：　使用任务调度函数采样一个batch的tasks: $T_i \sim \boldsymbol{p}(\boldsymbol{T})$

4：　**for all** T_i **do** — 基学习器(内层)

5：　　计算每类中K个样本的损失函数$\nabla_\theta L_{T_i}(f_\theta)$

6：　　用梯度下降算法计算适应的参数：$\theta_i' = \theta - \alpha \nabla_\theta L_{T_i}(f_\theta)$

7：　**end for** — 支持集

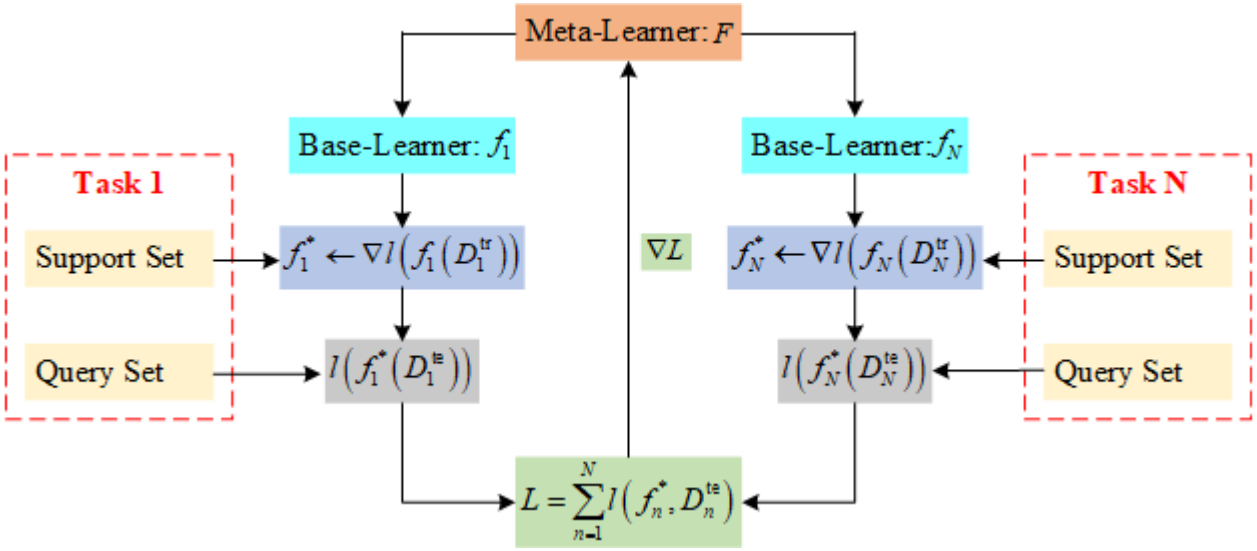
8：　更新参数$\theta \leftarrow \theta - \beta \nabla_\theta \sum_{T_i \sim p(T)} L_{T_i}\left(f_{\theta_i'}\right)$ — 查询集

9：**end while**

优化目标

- 元学习的主要目的是寻找元学习器F
  - 在F的指导下基学习器f在支持集的作用下经过几步微调就可以得到适应当前新任务的最优状态f^*
  - 而F的优化需要当前所有任务损失的累计和

$$\nabla_\theta \sum_{T_i \sim p(T)} L_{T_i}\left(f_{\theta_i'}\right)$$



算法1：模型不确定元学习

输入：$p(T)$:任务调度函数

输入：α, β: 学习率超参数

1：随机初始化θ

2：**while not done do** — 元学习器(外层)

3：　使用任务调度函数采样一个batch的tasks: $T_i \sim \boldsymbol{p}(\boldsymbol{T})$

4：　**for all** T_i **do** — 基学习器(内层)

5：　　计算每类中K个样本的损失函数$\nabla_\theta L_{T_i}(f_\theta)$

6：　　用梯度下降算法计算适应的参数：$\theta_i' = \theta - \alpha \nabla_\theta L_{T_i}(f_\theta)$

7：　**end for** — 支持集

8：　更新参数$\theta \leftarrow \theta - \beta \nabla_\theta \sum_{T_i \sim p(T)} L_{T_i}\left(f_{\theta_i'}\right)$ — 查询集

9：**end while**

模型训练

- 输入为一个给定的任务调度函数，比如随机抽样以及指定内外两层更新的步长α和β
- 取出一个**batch**的任务，假定4个任务为1个**batch**
- 每个任务T_i都是由样本和标签组成
- 利用任务的支持集计算**loss**，对于**MAML**使用的交叉熵：

$$L_{T_i}(f_\theta)$$

$$= \sum_{x^{(j)},y^{(j)}\sim T_i} y^{(j)} log f_\theta\left(x^{(j)}\right) + (1 - y^{(j)} \log(1 - f_\theta(x^{(j)})))$$

- 训练误差： $L_{T_i}(f_\theta)$：参数为θ时， T_i中支持集的损失

算法1：模型不确定元学习

输入：$p(T)$:任务调度函数

输入：α, β: 学习率超参数

1：随机初始化θ

2：**while not done do** — 元学习器(外层)

3：使用任务调度函数采样一个batch的tasks: $T_i \sim \boldsymbol{p}(\boldsymbol{T})$

4：**for all** T_i **do** — 基学习器(内层)

5：计算每类中K个样本的损失函数$\nabla_\theta L_{T_i}(f_\theta)$

6：用梯度下降算法计算适应的参数：$\theta_i' = \theta - \alpha \nabla_\theta L_{T_i}(f_\theta)$

7：**end for** — 支持集

8：更新参数$\theta \leftarrow \theta - \beta \nabla_\theta \sum_{T_i \sim p(T)} L_{T_i}\left(f_{\theta_i'}\right)$ — 查询集

9：**end while**

模型训练

- 利用梯度下降算法带入更新得到θ_i'
  - 梯度的迭代更新可以进行若干次，但是不能太大
  - 重复次数过多整个跑起来会非常缓慢
  - 本身是小样本，数据量很少，迭代次数多了容易过拟合

- 元学习器更新使用查询集数据进行
  - 损失是指查询集损失
  - 利用这个损失对初始化参数θ做的梯度下降优化

- 进入下一个**batch**任务，重复以上过程

算法1：模型不确定元学习

输入：$p(T)$:任务调度函数

输入：α, β: 学习率超参数

1：随机初始化θ

2：**while not done do**　　元学习器(外层)

3：　使用任务调度函数采样一个batch的tasks: $T_i \sim \boldsymbol{p}(\boldsymbol{T})$

4：**for all** T_i **do**　　基学习器(内层)

5：　计算每类中K个样本的损失函数$\nabla_\theta L_{T_i}(f_\theta)$

6：　用梯度下降算法计算适应的参数：$\theta_i' = \theta - \alpha \nabla_\theta L_{T_i}(f_\theta)$

7：**end for**　　支持集

8：　更新参数$\theta \leftarrow \theta - \beta \nabla_\theta \sum_{T_i \sim p(T)} L_{T_i}\left(f_{\theta_i'}\right)$

9：**end while**　　查询集

模型计算

- 在计算元更新梯度的时候参数是θ_i'，但是整体是对θ求梯度，为了不混淆可以先将元更新的θ记为ϕ：

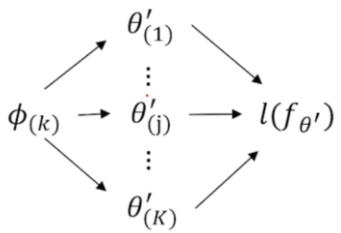
$$\nabla_\theta L(\theta) = \nabla_\theta \sum_{T_i \sim p(T)} l_{T_i}\left(f_{\theta_i'}\right) = \sum_{T_i \sim p(T)} \nabla_\theta l_{T_i}\left(f_{\theta_i'}\right)$$

$$\rightarrow \nabla_\phi L(\phi) = \sum_{T_i \sim p(T)} \nabla_\phi l_{T_i}(f_{\theta_i'})$$

- 对于单独的一个任务T_i来说，求这个任务的梯度，设为θ'
  - 每一个维度都是这个损失函数对ϕ的第k的参数的导，取出某一个ϕ，

$$\nabla_\phi\, l(f_{\theta'}) = \begin{bmatrix} \partial l(f_{\theta'}) / \partial \phi_{(1)} \\ \partial l(f_{\theta'}) / \partial \phi_{(2)} \\ \vdots \\ \partial l(f_{\theta'}) / \partial \phi_{(k)} \\ \vdots \\ \partial l(f_{\theta'}) / \partial \phi_{(K)} \end{bmatrix}$$

  即从更新后的θ'的每一维元素在链式法则到损失函数：



$$\frac{\partial l(f_{\theta'})}{\partial \phi_{(k)}} = \sum_j \frac{\partial l(f_{\theta'})}{\partial \theta'_{(j)}} \frac{\partial \theta'_{(j)}}{\partial \phi_{(k)}}$$

模型计算

- 由于涉及到ϕ和θ'的关系，回顾一下ϕ通过任务，更新优化，得到θ'的时候，其实是下式：

$$\theta_i' = \phi - \alpha\nabla_\theta L_{T_i}(f_\phi) \rightarrow \theta'_{(j)} = \phi_{(j)} - \alpha\nabla_\theta L_{T_i}(f_\phi)$$

  - $\phi_{(j)}$和$\phi_{(k)}$线性无关
    - 如果$j \neq k$，导出来的第一项就是0，只剩负的α乘以一个二阶导
    - 假设$j = k$，导出来就是1减去一个二阶导
- 实际运算起来求二阶导非常麻烦，直接近似为0和1

$$\frac{\partial l(f_{\theta'})}{\partial \phi_{(k)}} = \sum_j \frac{\partial l(f_{\theta'})}{\partial \theta'_{(j)}} \frac{\partial \theta'_{(j)}}{\partial \phi_{(k)}} \approx \frac{\partial l(f_{\theta'})}{\partial \theta'_{(k)}}$$

算法1：模型不确定元学习

输入：$p(T)$:任务调度函数

输入：α, β: 学习率超参数

1：随机初始化θ

2：**while not done do** — 元学习器(外层)

3：使用任务调度函数采样一个batch的tasks: $T_i \sim p(T)$

4：**for all** T_i **do** — 基学习器(内层)

5：计算每类中K个样本的损失函数$\nabla_\theta L_{T_i}(f_\theta)$

6：用梯度下降算法计算适应的参数：$\theta_i' = \theta - \alpha\nabla_\theta L_{T_i}(f_\theta)$

7：**end for** — 支持集

8：更新参数$\theta \leftarrow \theta - \beta\nabla_\theta \sum_{T_i \sim p(T)} L_{T_i}\left(f_{\theta_i'}\right)$ — 查询集

9：**end while**

$$j \neq k: \quad \frac{\partial \theta'_{(j)}}{\partial \phi_{(k)}} = -\alpha \cdot \partial \frac{\partial l(f_\phi)}{\partial \phi_{(j)} \partial \phi_{(k)}}$$

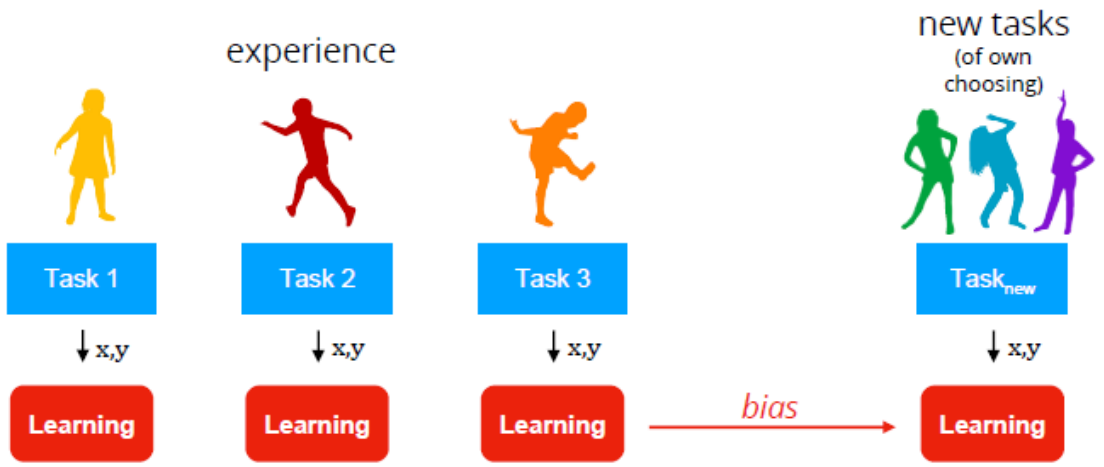
$$j = k: \quad \frac{\partial \theta'_{(j)}}{\partial \phi_{(k)}} = 1 - \alpha \cdot \partial \frac{\partial l(f_\phi)}{\partial \phi_{(j)} \partial \phi_{(k)}}$$

$$\theta_i' = \phi - \alpha\nabla_\phi L_{T_i}(f_\phi)$$

$$\theta'_{(j)} = \phi_{(j)} - \alpha \frac{\partial l(f_\phi)}{\partial \phi_{(j)}}$$

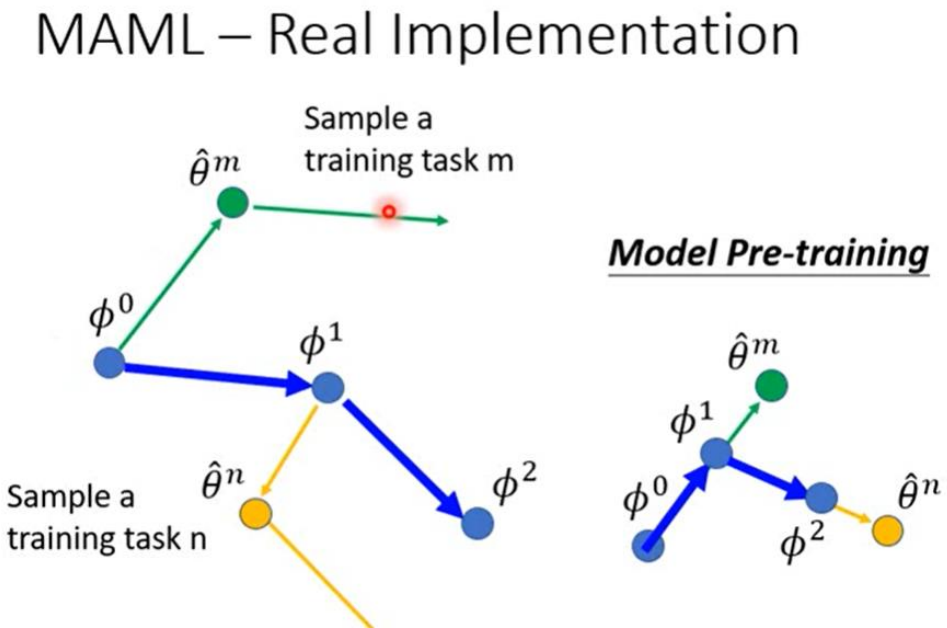
meta-learning & few-shot learning

- 为什么meta-learning总是和few-shot learning搭配使用?
  - meta-learning是因为小样本学习而产生的算法 ×
  - meta-learning更适合处理小样本问题，而不是样本很多的情况 √
- 如果不是做few-shot问题， 数据较多的话，每一个任务跑起来都特别慢
  - 每一个任务都相当于在训练神经网络做分类
  - 甚至是做few-shot任务时，使用MAML跑3600个iteration，跑一晚上才跑5%



meta-learning & pre-training

- 都是通过有监督或自监督方式，先在数据集充足的任务A上训练模型，然后利用该模型的参数来初始化数据量比较少的任务B，让数据量比较少的任务B也能够训练起来
- 预训练的优化目标是找到一个当前最好的表现的位置
- 元学习的优化目标是找到一个可以训练出最好表现的位置



meta-learning存在的问题

- 在训练时，不同任务的表现是有偏差的——数据不平衡问题
  - 模型会对某个任务打到过分好的结果，但是，一旦出现测试任务与训练任务存在差异的情况，测试效果就会被影响
  - 应该避免模型在某些任务上被过度执行，调整至任务无偏模型应当在困难任务上多执行，以保证应对测试任务时的困难情况加大偏重
- 一个良好的模型，应该在任务上保持一致的性能
  - 如果不能保持一致性，那么**meta-learner**在更新参数时，未必能有一个足够通用的更新方案，因此很难在测试任务上有好的表现

任务不确定元学习

算法原理——最大熵方法

T	使用最大熵来确定任务选择概率
I	任务集$p(T)$
P	1.使用任务调度函数采样一个**batch**的**tasks**: $T_i \sim \boldsymbol{p(T)}$ 2.计算每类中**K**个样本的损失函数$\nabla_\theta L_{T_i}(f_\theta)$ 3.用梯度下降算法计算适应的参数：$\theta_i' = \theta - \alpha \nabla_\theta L_{T_i}(f_\theta)$ 4.更新参数$\theta \leftarrow \theta - \beta \nabla_\theta \{\sum_{T_i \sim p(T)} L_{T_i}\left(f_{\theta_i'}\right) - \lambda(H_{T_i}(f_\theta) - H_{T_i}(f_{\theta_i})\}$
O	元学习参数θ

算法原理——最大熵方法

- 信息熵与概率分布之间的关系：

$$H(P) = -\sum_{i=1}^{n} p_i * \log(p_i); \sum_{i=1}^{n} p_i = 1$$

- 如果p_i是等概率的，熵可以取到最大，所以需要让p取到等概率
- 已知，模型的初始参数是在 $task\ T_i$上进行训练
- 根据一定的更新规则，比如梯度下降方法，将θ更新为θ_i：

$$\theta_i \leftarrow \theta - \alpha \nabla_\theta L_{T_i}(f_\theta)$$

算法原理——最大熵方法

- 对于**MAML**，**loss**是分类任务T_i在更新前的交叉熵损失，希望他在相同概率下对标签进行预测，引入最大熵的概念：

$$H_{T_i}(f_\theta) = -E_{x_i \sim P_{T_i}(x)} \sum_{n=1}^{N} \hat{y}_{i,n} \log(\hat{y}_{i,n})$$

- 最大熵原理认为，在所有可能的概率模型分布中，熵最大的模型是最好的模型
  - 对一个随机事件的概率分布进行预测时，预测应当满足全部已知的约束，而对未知的情况不要做任何主观假设
  - 在这种情况下，概率分布最均匀，预测的风险最小，因此得到的概率分布的熵是最大

算法原理——最大熵方法

- 希望在更新之前，保持这个熵最大
  - 预测为等概率
- 更新之后，这个熵尽可能小
  - 熵变小就意味着模型在将θ更新到θ_i后，对标签的可信度更高
- 如果把二者放一起，实际上就是将更新前后熵的差值最大化

$$maximize(H_{T_i}(f_\theta) - H_{T_i}(f_{\theta_i}))$$

- 该熵项可以与典型的**Meta-training**的目标结合：

$$minimize(E_{T_i \sim P(T)} L_{T_i}(f_{\theta_i}) - \lambda(H_{T_i}(f_\theta) - H_{T_i}(f_{\theta_i})))$$

算法2：任务不确定元学习最大熵方法

输入：$p(T)$:任务调度函数

输入：α, β: 学习率超参数

1：随机初始化θ

2：**while not done do**

3：　使用任务调度函数采样一个batch的tasks: $T_i \sim \boldsymbol{p}(\boldsymbol{T})$

4：　**for all** T_i **do**

5：　　计算每类中K个样本的损失函数$\nabla_\theta L_{T_i}(f_\theta)$

6：　　用梯度下降算法计算适应的参数：$\theta_i' = \theta - \alpha \nabla_\theta L_{T_i}(f_\theta)$

7：　**end for**

8：　更新参数$\theta \leftarrow \theta - \beta \nabla_\theta \{\sum_{T_i \sim p(T)} L_{T_i}(f_{\theta_i'}) - \lambda(H_{T_i}(f_\theta) - H_{T_i}(f_{\theta_i}))\}$

9：**end while**

算法原理——差异最小化方法

T	使用的收入不平等代替任务偏差
I	任务集$p(T)$
P	1.使用任务调度函数采样一个**batch**的**tasks**: $T_i \sim p(T)$ 2.计算每类中K个样本的损失函数$\nabla_\theta L_{T_i}(f_\theta)$ 3.用梯度下降算法计算适应的参数：$\theta_i' = \theta - \alpha \nabla_\theta L_{T_i}(f_\theta)$ 4.更新参数$\theta \leftarrow \theta - \beta \nabla_\theta \{\sum_{T_i \sim p(T)} L_{T_i}\left(f_{\theta_i'}\right) - \lambda I_\varepsilon(\{L_{T_i}(f_\theta)\})\}$
O	元学习参数θ

算法原理——差异最小化方法

- 收入不平等
  - 源于经济学的概念，有一些常用的指标，用来衡量个人之间、地区之间的收入不平等
  - 可以使用收入不平等代替**loss**不平等

loss inequality => income inequality

- 有了这样一个度量，在最终更新θ的时候，相当于最小化：

$$minimize(E_{T_i \sim P\,(T)} L_{T_i}(f_{\theta_i}) - \lambda I_{\varepsilon}(\{L_{T_i}(f_{\theta})\}))$$

- 第一项是期望损失，第二项是原模型f_{θ}在更新之前，在任务上的不平等性，不平等越少越好，可以选择的收入标准有两种：
  - 泰尔熵标准：$T_T = \frac{1}{M}\sum_{i=1}^{M} \frac{l_i}{\bar{l}} ln \frac{l_i}{\bar{l}}$
  - 广义熵指数

用元学习解决用户冷启动问题

算法原理——MeLU

T	对冷启动用户的偏好做估计
I	用户基本信息、项目基本信息和用户项目交互信息
P	1. 构建一个由决策层和带有用户和项目emb层组成的用户偏好估计模型 2. 基于MAML设计可快速适应新任务（新用户）的个性化用户偏好估计模型 3. 构建基于个性化用户偏好估计模型的候选商品选择策略
O	用户偏好估计概率

P	如何有效利用元学习优化用户评估网络
C	有足够的非冷启动用户数据
D	冷启动用户的交互数据不足
L	**KDD 2019 (CCF A)**

用户冷启动问题

- 冷启动问题
  - 数据稀缺问题通常存在于新用户来访问在线平台或出现新项目的冷启动情况下
  - 由于观察到的用户项目相互通常受到限制，传统的协作过滤方法或深度学习方法，需要大量的训练数据很难执行
- 在元学习中，冷启动推荐可以作为小样本学习的应用
  - 针对新用户或具有稀疏互动的项目的推荐任务自然分为元训练任务

- 因此，元学习技术被广泛用于减轻冷启动推荐任务的数据不足问题

构建用户偏好估计模型

- 输入层：离散变量嵌入后拼接，连续变量直接拼接
- 嵌入层：根据离散的特征，通过嵌入层嵌入到连续的向量空间中
- 拼接层：将多个特征向量直接拼接起来
- 决策层：由于用户和项目的向量维度不完全一致，所以无法使用矩阵分解，而使用多层神经网络
- 输出层：优化的目标，可以是点击率，隐式反馈、停留时长等

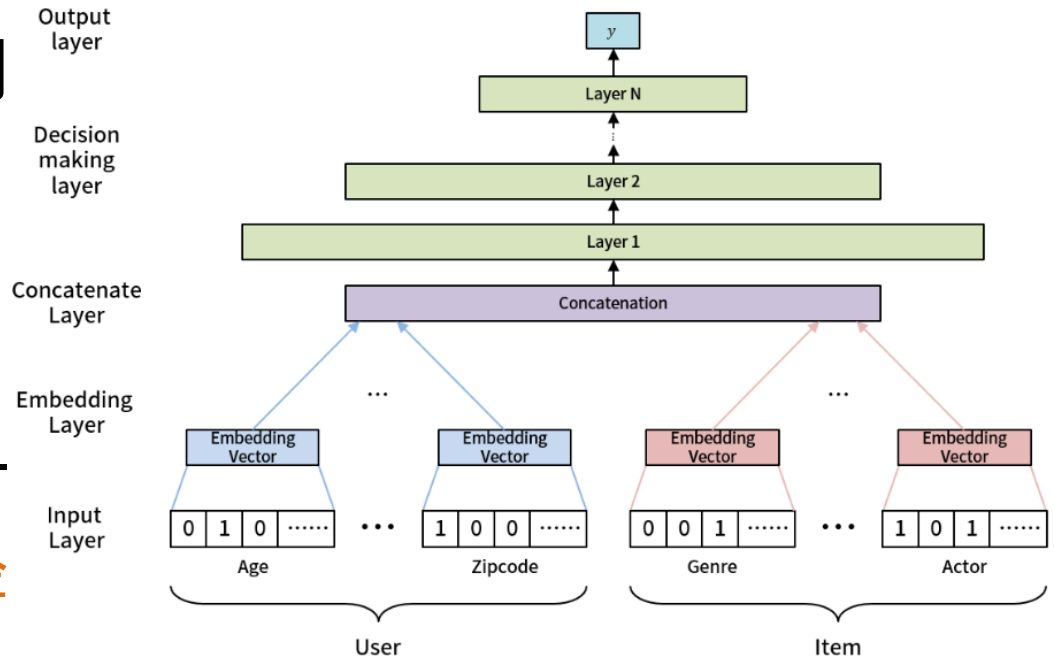
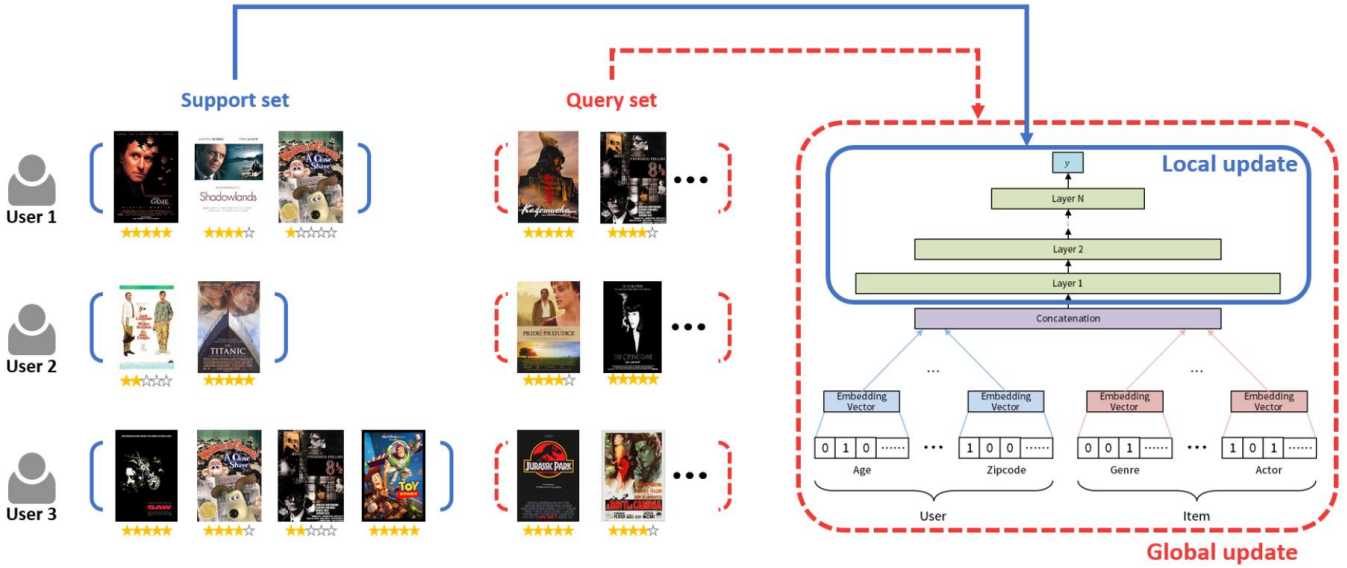


Figure 2: User preference estimation.

构建个性化用户偏好估计模型

- 将输入和嵌入合并，决策和输出合并
- 初始化θ_1 和 θ_2 进行多轮迭代训练
  - 每一轮训练过程中，挑选一定数量的用户，对每个用户（相当于MAML中的Task）进行采样得到支持集，计算在支持集上得到的训练loss，并计算对应的梯度实现局部更新



算法3：用户冷启动元学习方法

输入：α, β: 学习率超参数

1：随机初始化θ_1, θ_2

2：**while not converge do**

3：　采样一个batch的用户: $\boldsymbol{B} \sim \boldsymbol{p}(\boldsymbol{B})$

4：　**for all** $\boldsymbol{user\ i\ in\ B}$ **do**

5：　　计算每类中K个样本的损失函数$\nabla_\theta L_{T_i}(f_\theta)$

6：　　用梯度下降算法计算适应的参数：$\theta_2^i = \theta_2^i - \alpha \nabla_{\theta_2^i} L_{T_i}\left(f_{(\boldsymbol{\theta_1}, \theta_2^i)}\right)$

7：　**end for**

8：　更新参数$\theta_1 \leftarrow \theta_1 - \beta \nabla_{\theta_1} \sum_{\mathbf{i} \sim \boldsymbol{B}} L_{T_i}\left(f_{(\boldsymbol{\theta_1}, \theta_2^i)}\right)$

$$\theta_2 \leftarrow \theta_2 - \beta \nabla_{\theta_2} \sum_{\mathbf{i} \sim \boldsymbol{B}} L_{T_i}\left(f_{(\boldsymbol{\theta_1}, \theta_2^i)}\right)$$

9：**end while**

构建个性化用户候选商品选择策略

- 偏好估计选择策略
  - 梯度价值表示每个用户在局部更新时的梯度的F范数，并进行归一化

$$||\nabla_{\theta_2^i} L_{T_i}\left(f_{(\boldsymbol{\theta_1},\theta_2^i)}\right)||_{\boldsymbol{F}}$$

  - 热门价值是该物品的热门情况，即该物品被用户交互的相对频率，可以反映用户-物品的交互是否频繁

- 如果梯度的F范数越大，或者交互越频繁，我们认为这个物品可能是该用户的偏好，给与更高的分数（两个**value**归一化的乘积）

评价指标

- 归一化折损累计增益（**Normalized Discounted Cumulative Gain，NDCG**），是一种考虑了返回顺序的评价指标。取值范围[0,1]，越大效果越好

$$\mathrm{NDCG@k} = \frac{\mathrm{DCG@k}}{\mathrm{IDCG@k}}$$

- 其中，$\mathrm{DCG@k} = \sum_{i=1}^{k} \frac{\mathrm{rel_i}}{\log_2(i+1)}$，$\mathrm{rel_i}$指的是第$i$个结果的真实相关性分数
- $\mathrm{IDCG@k} = \sum_{i=1}^{|REL|} \frac{\mathrm{rel_i}}{\log_2(i+1)}$
- **IDCG也就是理想的DCG（Ideal DCG）。|REL|**表示，结果按照真实相关性从大到小排序，取前k个结果组成的集合的个数
- **MAE**是平均绝对误差：$\mathrm{MAE} = \frac{1}{|U|}\sum_{i\in U}\frac{1}{|H_i'|}\sum_{j\in H_i'}|y_{ij} - \hat{y}_{ij}|$

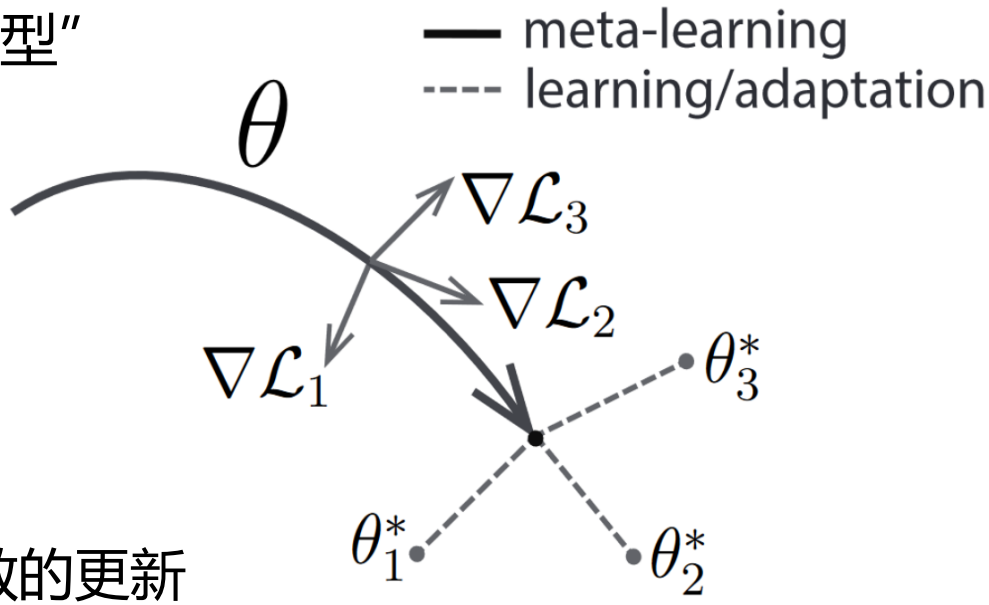
实验结果

Type	Method	MovieLens			Bookcrossing		
		MAE	$nDCG_1$	$nDCG_3$	*MAE*	$nDCG_1$	$nDCG_3$
Recommendation of existing items for existing users	PPR	**0.1820**	**0.9796**	**0.9831**	3.8092	0.8242	0.8494
	Wide & Deep	0.9047	0.9090	0.9117	1.6206	0.9012	0.9172
	MeLU-1	0.7661	0.8866	0.8904	**0.7799**	**0.9563**	**0.9572**
	MeLU-5	0.7567	0.8870	0.8919	0.7955	0.9546	0.9552
Recommendation of existing items for new users	PPR	1.0748	0.8299	0.8468	3.8430	0.8201	0.8434
	Wide & Deep	1.0694	0.8559	0.8639	2.0457	0.8238	0.8515
	MeLU-1	0.7884	0.8799	0.8810	**1.8701**	**0.8265**	0.8527
	MeLU-5	**0.7854**	**0.8803**	**0.8812**	1.8767	0.8263	**0.8532**
Recommendation of new items for existing users	PPR	1.2441	0.7289	0.7632	3.6821	0.8115	0.8367
	Wide & Deep	1.2655	0.7420	0.7721	2.2648	0.8190	0.8437
	MeLU-1	0.9361	**0.7715**	0.7990	**2.1047**	**0.8202**	**0.8441**
	MeLU-5	**0.9275**	0.7697	**0.8005**	2.1236	0.8190	0.8440
Recommendation of new items for new users	PPR	1.2596	0.7292	0.7634	3.7046	0.8171	0.8381
	Wide & Deep	1.3114	0.7680	0.7874	2.3088	0.8160	0.8405
	MeLU-1	0.9299	**0.7760**	**0.8011**	**2.1475**	**0.8184**	0.8410
	MeLU-5	**0.9235**	0.7752	0.8008	2.1721	**0.8184**	**0.8422**

总结

总结

- 元学习
  - 训练一个用于学习“数学模型” 的“数学模型”
  - 模型分两层
    - 外层用于更新内层参数
    - 内层用于完成任务
  - 数据按照**task**为单位划分
    - 每一个**task**相当于一个内层的学习任务
    - 每一个内层学习任务对应着一次元学习参数的更新



参考文献

- [1] Finn C, Abbeel P, Levine S. Model-agnostic meta-learning for fast adaptation of deep networ ks[C]//International conference on machine learning. PMLR, 2017: 1126-1135.
- [2] Jamal M A, Qi G J. Task agnostic meta-learning for few-shot learning[C]//Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2019: 11719-11727.
- [3] Lee H, Im J, Jang S, et al. Melu: Meta-learned user preference estimator for cold-start recom mendation[C]//Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. 2019: 1073-1082.
- [4] Wang C, Zhu Y, Liu H, et al. Deep Meta-learning in Recommendation Systems: A Survey[J]. arXiv preprint arXiv:2206.04415, 2022.

谢谢！

大成若缺，其用不弊。大盈若冲，其用不穷。大直若屈。大巧若拙。大辩若讷。静胜躁，寒胜热。清静为天下正。