

Lecture 14: 元学习 (Meta Learning)

Introduction of Meta Learning

Step 1

Step 2

Step 3

总结: meta learning的框架

比较: Machine Learning v.s. Meta Learning

Goal

Training Data

within-task training/testing v.s. across-task training/testing

Loss

training过程

Meta和ML的相似处

实例说明: learning algorithm中哪些是可以被“学习”的?

Learning to Initialize

Learning a Optimizer

学网络架构: Network Architecture Search (NAS)

Data Processing

Beyond GD (待填的坑)

应用

用meta learning实现few shot learning

meta learning在具体领域的可能应用

Lecture 14: 元学习 (Meta Learning)

Lectured by HUNG-YI LEE (李宏毅)

Recorded by Yusheng zhao (yszhao0717@gmail.com)

meta的意味: meta-X = X about X

meta learning: **学习**如何学习

DeepLearning大部分时间都是在爆调超参数, 工业界的方法: 大力出奇迹, 拿好多张GPU同时跑几组不同的超参数组, 看看哪个最好用。学术界(学校的实验室里边)往往资源贫穷/(ToT)/~~

Industry



Using 1000 GPUs to try
1000 sets of hyperparameters

Academia



“Telepathize” (通灵) a set of
good hyperparameters

所以我们想, 既然机器学习可以自动学一个模型, 那么hyper parameter可不可以也用学的呢? ——这就是meta learning所做的事情。

Machine Learning的知识回顾（事实上，meta learning和machine learning没有太大区别）

三个步骤：（目的：looking for a function）

- **step 1: Function with unknown**

其中神经元的权重（weights）和偏置（biases）就是需要学习得到的unknown的参数，用 θ 来标识

- **step 2: Define loss function**

$L(\theta) = \sum_{i=1}^n e_i$ ，其中每一个 e_i 都是train结果和ground truth之间的距离（如果是分类任务就是交叉熵）

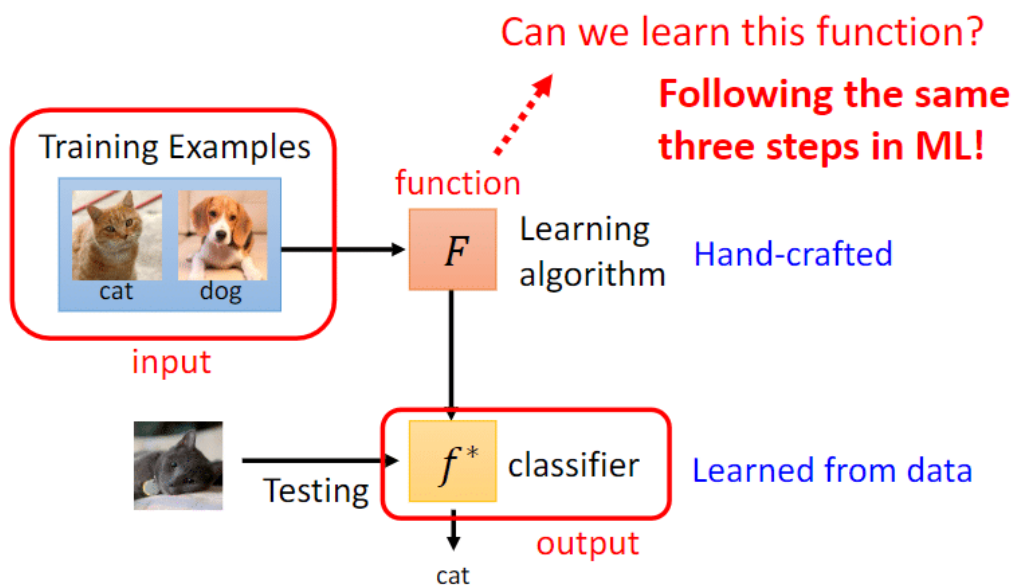
- **step 3: Optimization**

找一个 θ^* 使得loss越小越好，即优化任务： $\theta^* = \arg \min_{\theta} L(\theta)$.

本课程中都是用梯度下降法来解决这个优化问题，我们得到一组loss足够下的参数组 θ^* ，那么参数带入黑箱函数 f_{θ^*} 中，实现我们需要的端到端的任务（输入-输出）

Introduction of Meta Learning

一个ML算法“简化”来看也是一个function，这个function的输入是一个数据集（training example），输出训练的结果（如果是分类任务）那就是一个classifier；把test set测试集丢进这个classifier中，这个算法的期望当然就是分类正确率越高越好。



这个算法 F 通常是Hand-crafted（人想出来的），我们以下借鉴ML的三个步骤来学习这个 F 。

Step 1



ML里边的step 1，其中learnable的是neuron神经元的weight和biases

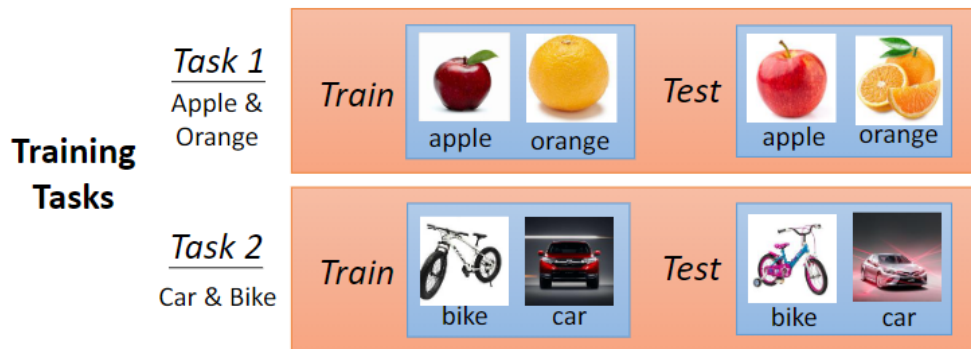
在meta learning里边可以学出来的东西——网络架构 (Net Architecture)、初始化参数 (Initial parameters)、学习率 (Learning Rate) 等。以上之前课程中我们都是人为设置，现在希望使用meta learning来进行学习。

- 用 ϕ 来统称需要元学习的成分 (learnable components ϕ)：网络架构 (Net Architecture)、初始化参数 (Initial parameters)、学习率 (Learning Rate) 等
- 以下都把learning algorithm记作 F_{ϕ} ， ϕ 代表了未知的参数是
- 不同的meta learning的方法其实就是想办法去学不同的component

Categorize meta learning based on what is learnable $\Rightarrow \phi$

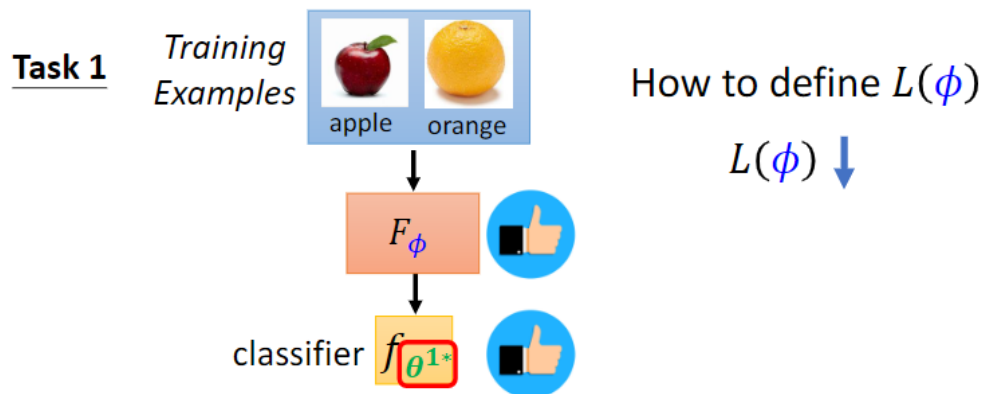
Step 2

- 针对 **learning algorithm** F_ϕ 定义 **loss function**
这个 loss function 记作 $L(\phi)$, 如果 $L(\phi)$ 比较小, 说明这个 $F(\phi)$ 比较好。
 $L(\phi) \downarrow$  $L(\phi) \uparrow$ 
- 我们用训练任务 (training tasks) 来作为训练资料喂给 meta learning 的模型 F_ϕ 。如下图



如上, 以训练二元分类器为例, 每个任务里面都有训练资料和测试资料。

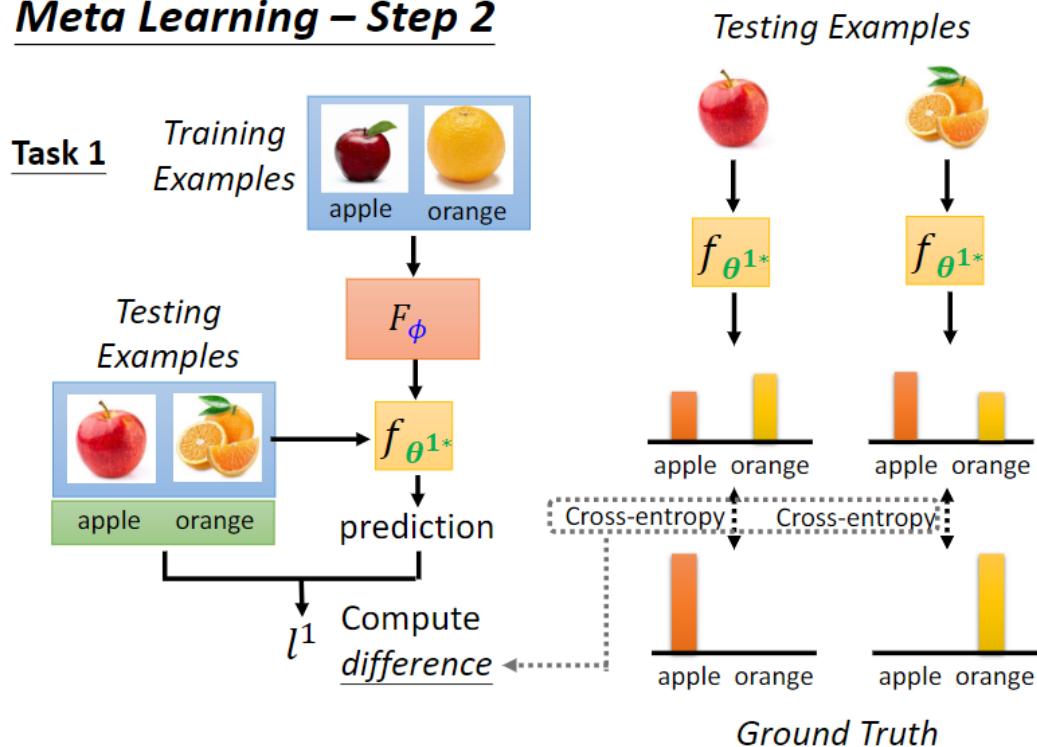
- 定义 $L(\phi)$: 把某一个任务的资料拿出来丢给 learning algorithm F_ϕ , 输出一个具体的分类器 (output), 任务一的 classifier 记作 $f_{\theta^{1*}}$



θ^{1*} : parameters of the classifier learned by F_ϕ using the training examples of task 1

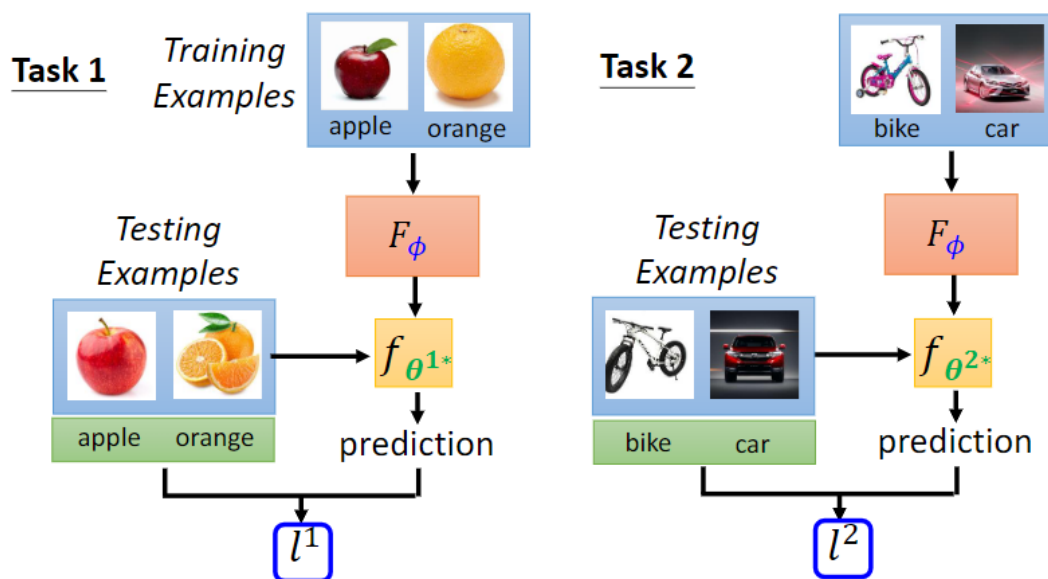
- 确定 classifier $f_{\theta^{1*}}$ 的性能好坏: 用任务的测试资料对该分类器进行评估

Meta Learning – Step 2



如上图，测试资料丢进这个classifier做一个prediction，计算（预测和ground truth）交叉熵统计加起来得到 l^1

- 类似如上过程，用其他任务来确定各自的classifier（这个例子中meta learning只有两个任务）



在任务一和任务二的表现分别为 l^1 和 l^2 ，将两者加起来，得到总loss为 $l^1 + l^2$ ，对于n个任务的meta learning来说

$$L(\phi) = \sum_{i=1}^n l^i \quad (1)$$

在一般的ML中，我们用训练资料计算loss，而在meta Learning中我们用测试资料来计算loss。这是因为meta learning的训练单位是“training task”，换言之，评估meta learning的性能是基于“testing task”上表现如何，在单一训练单元（tasks）上，计算loss可以采用测试资料。而typical ML的评估则是根据测试资料上的结果，因而不能用测试资料来计算loss。

In typical ML, you compute the loss based on **training examples**
In meta, you compute the loss based on **testing examples** of **training tasks**.

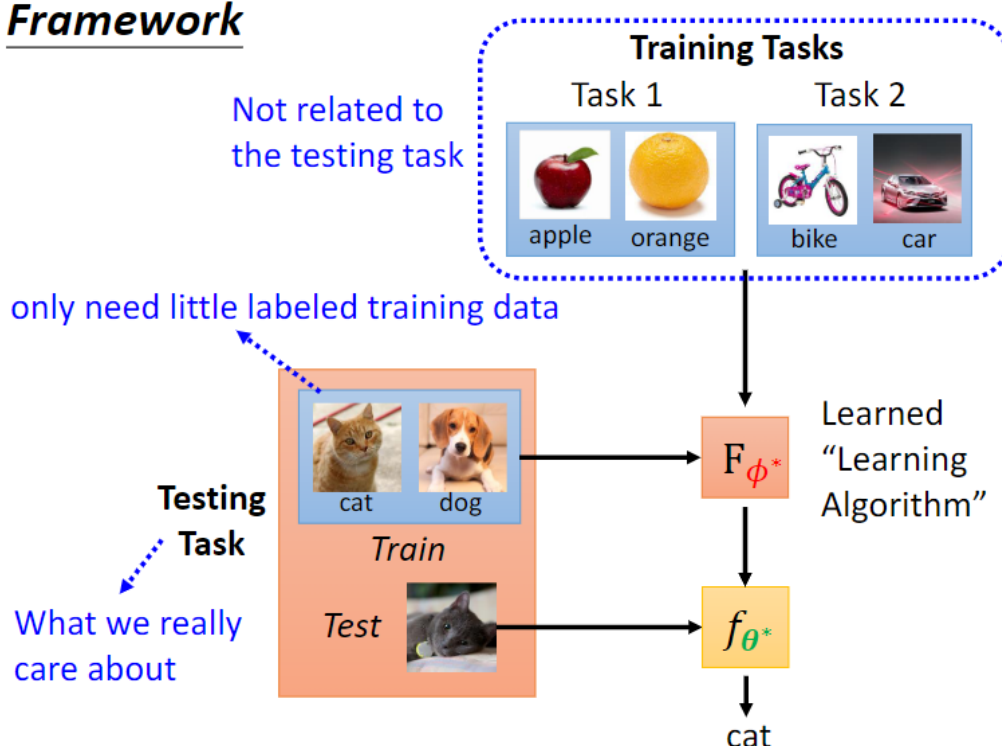
Step 3

- 对于learning algorithm $F(\phi)$ 已知loss function $L(\phi) = \sum_{i=1}^n l^i$
- 本步骤目的：找到一个 ϕ 去 minimize $L(\phi)$ ，即优化问题： $\phi^* = \arg \min_{\phi} L(\phi)$
- 解这个优化问题
 - 如果知道loss对 ϕ 偏导，即 $\frac{\partial L}{\partial \phi}$ 易于计算，那就不用梯度下降
 - 经常的情况，在meta里边，loss的偏导不易于计算。这时候需要 **Reinforcement Learning** 硬train一发，或者用进化算法（Evolutionary Algorithm）

经过以上三步，最终我们learned出来一个learning algorithm F_{ϕ} 。

总结：meta learning的框架

Framework



简而言之，meta learning就是在训练任务上找出类比人类想出来（譬如SGD等等）的learning algorithm，再将该优化方法用在测试任务上，从而得到较好表现的目标函数。

few shot learning，有人觉得和meta learning很像。因为小样本学习（few shot learning）的learning algorithm（通常是不易人类想出来的）基本都是通过meta learning得到的。换言之，我们可以通过meta learning实现few shot learning。

对于整个meta learning的框架而言“training data”指的就是training task，具体到testing task中，“training data”指的就是learning algorithm用在testing task所做ML任务的常规意义的训练数据。

比较：Machine Learning v.s. Meta Learning

Goal

Machine Learning \approx find a function f

Dog-Cat
Classification

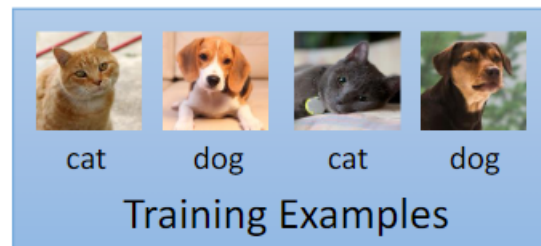
$$f(\text{image of cat}) = \text{"cat"}$$

Meta Learning

\approx find a function F that finds a function f

Learning
Algorithm

F



$) = f$

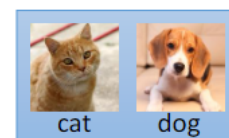
典型的ML任务是为了找到一个black-box的目标函数，而Meta则是为了找到能找到这个目标函数 f 的优化方法 F 。

Training Data

Training Data

Machine Learning

One task

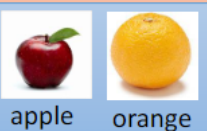


Meta Learning

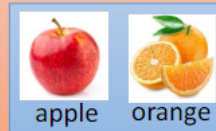
Training tasks

Task 1
Apple &
Orange

Train

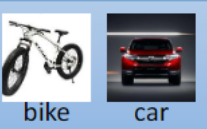


Test

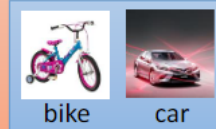


Task 2
Car & Bike

Train



Test



Support set

Query set

(in the literature of "learning to compare")

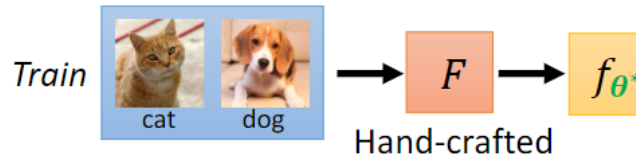
ML: 训练资料、测试资料

Meta: 训练任务、测试任务（其中的训练资料称之为Support Set、测试资料称之为Query Set）

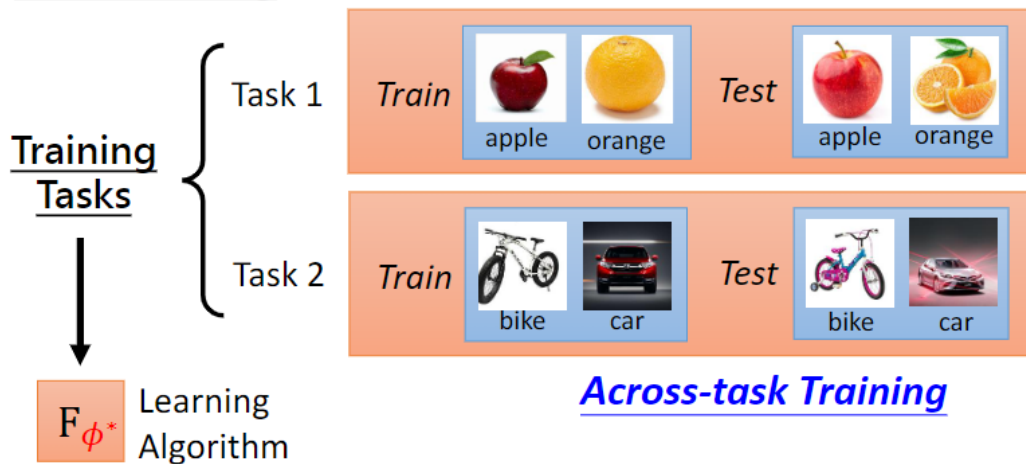
within-task training/testing v.s. across-task training/testing

Machine Learning

Within-task Training

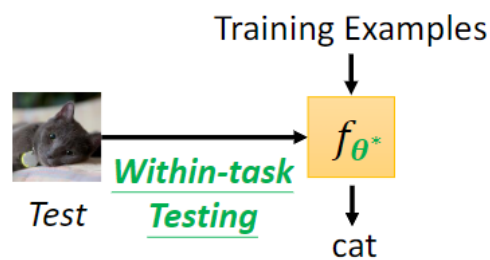


Meta Learning

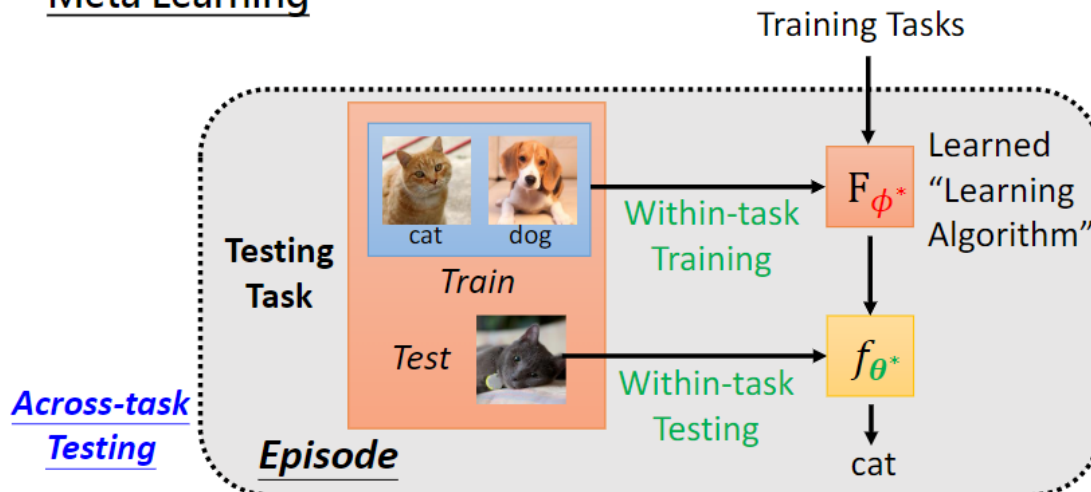


一般的ML是**Within-task Training**，而meta里边根据一堆任务学出一个learning algorithm叫做**Across-task Training**。

Machine Learning



Meta Learning



学习出来的东西（分类器/learning algorithm）如何处理？在ML中是test data丢进去，跑出来结果，结束。这个流程就是**within-task testing**。而在meta中，我们需要把学习出来的优化方法放进test task做一次常规的ML，包括within-task training和testing（如上图各一次），这两个流程有时也称之为**Episode**。整个meta这部分流程称之为**Across-task Testing**

Loss

Machine Learning

$$L(\theta) = \sum_{k=1}^K e_k$$

Sum over training examples in one task

Meta Learning

$$L(\phi) = \sum_{n=1}^N l^n$$

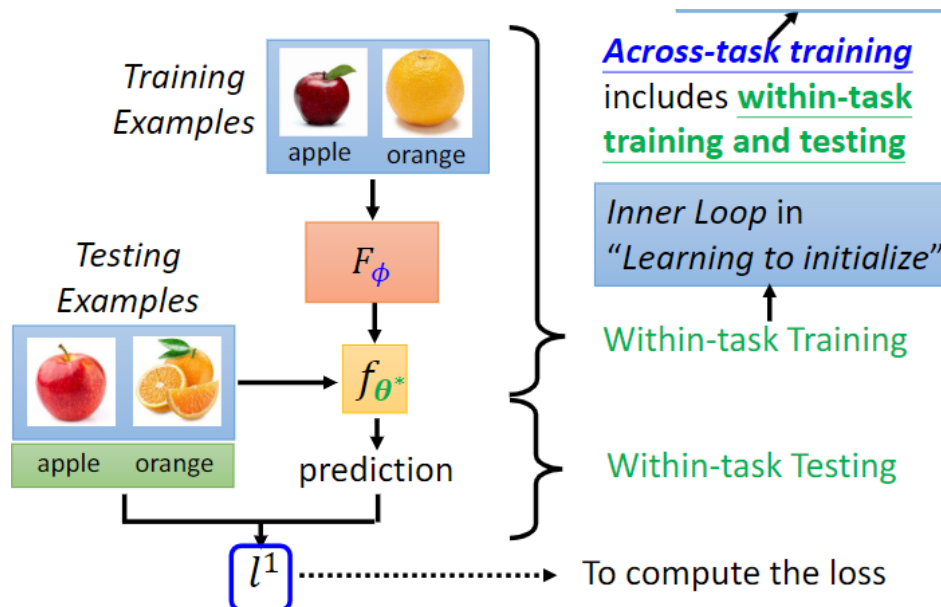
Sum over testing examples in one task

Sum over training tasks

ML: 一个任务中, 不同数据单元的loss之和; Meta: 一把任务, 每个训练任务中的测试数据综合loss之和

training过程

要算每个任务的 l , 需要经过一次**Within-task Training**、一次**within-task testing**



在meta一个流程 (即Across-task training) 中包含若干个within-task training/testing。在一些文献中, 两者分别称之为**Outer Loop** (Across-task training) 和**Inner Loop** (within-task training/testing)

Meta和ML的相似处

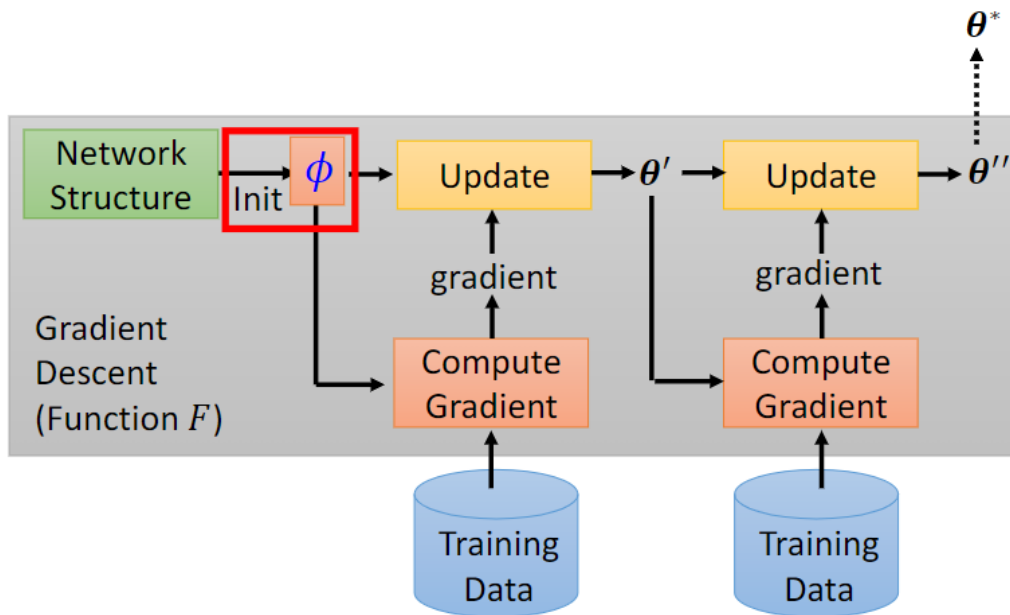
- training tasks上的过拟合
- 拿更多的训练任务来训练你的模型, 提高meta learning最后在test task的性能 (performance)
- “任务增强” (Task Augmentation)
- learning algorithm中的也有许多超参数, 该死居然还有调参.....套娃爆调参 (😓难道meta不就是为了新任务新模型不浪费时间调参麼)
- Development task 😊 (~~类比development set)

开发集 (development set) 用于调整参数, 选择特征, 以及对学习算法作出其它决定。有时也称为**留出交叉验证集 (hold-out cross validation set)**。在supervised ML中经常用于确定网络结构或者控制模型复杂程度的参数。

很多meta的论文实际上都没有使用development task（用来调整learning algorithm的复杂程度），或许这也是可以做一做的点。

实例说明：learning algorithm中哪些是可以被“学习”的？

在梯度下降中：我们有一个Network Structure；根据training data不断更新梯度，直到得到满意的结果。



以上，初始化参数是可以被学习的。

Learning to Initialize

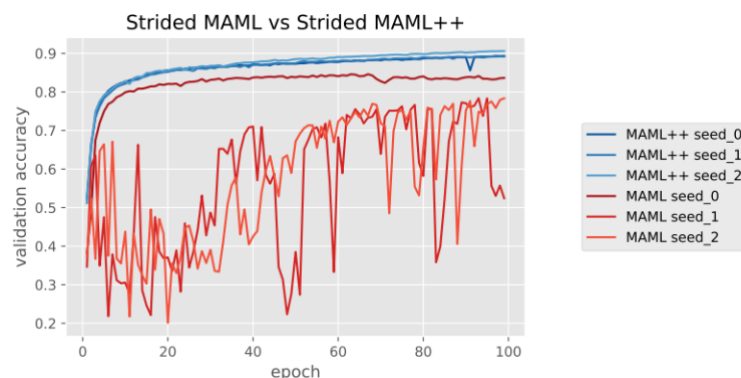
- Model-Agnostic Meta-Learning (MAML)

Chelsea Finn, Pieter Abbeel, and Sergey Levine, “Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks”, ICML, 2017

- How to train MAML

Antreas Antoniou, Harrison Edwards, Amos Storkey, How to train your MAML, ICLR, 2019

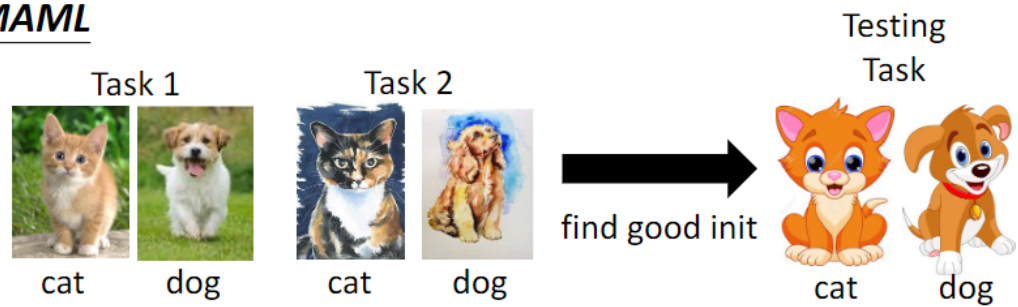
需要调参，random seed



- 联想到Pre-training、lifelong learning和transfer learning

在MAML里面，通过一把子训练任务找到好的init，最后用在测试任务上。在self-supervised learning里面（也有类似的做法），在proxy tasks上训练，最后用在测试数据，譬如BERT就是做句子的填空、也有一些工作可以做图像像素的填空（masking）。kaiming的MAE一样的思路

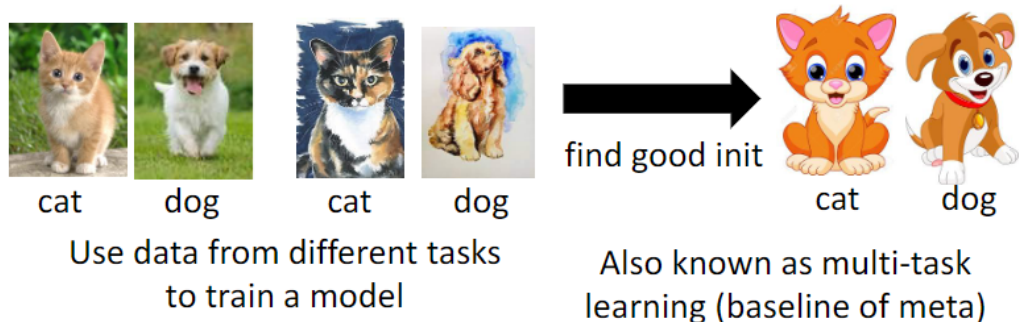
MAML



Pre-training (Self-supervised Learning)



Pre-training (more typical ways)



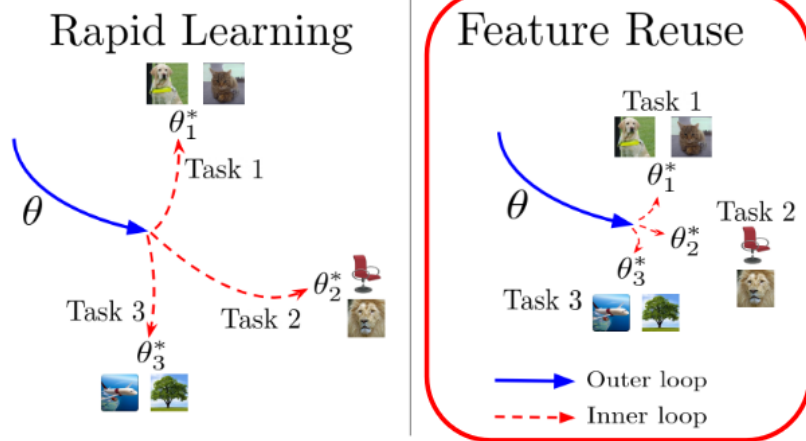
两者的不同包括：pre-training用unlabelled的资料，而MAML训练用到labelled资料。早期的self-supervised learning实际上会把所有资料放一块训练一个model，这种方法也称之为多任务学习（multi-task learning）

- 我们甚至可以认为不同的任务就是不同的domain，那我们在不同的任务上的meta learning是一种解决domain adaptation的方法。DA更是一个问题而非方法，尝试解决这个问题方法有很多...

更多关于MAML的...

- ANIL (Almost No Inner Loop)

Aniruddh Raghu, Maithra Raghu, Samy Bengio, Oriol Vinyals, Rapid Learning or Feature Reuse? Towards Understanding the Effectiveness of MAML, ICLR, 2020



Feature Reuse是MAML效果好的关键。

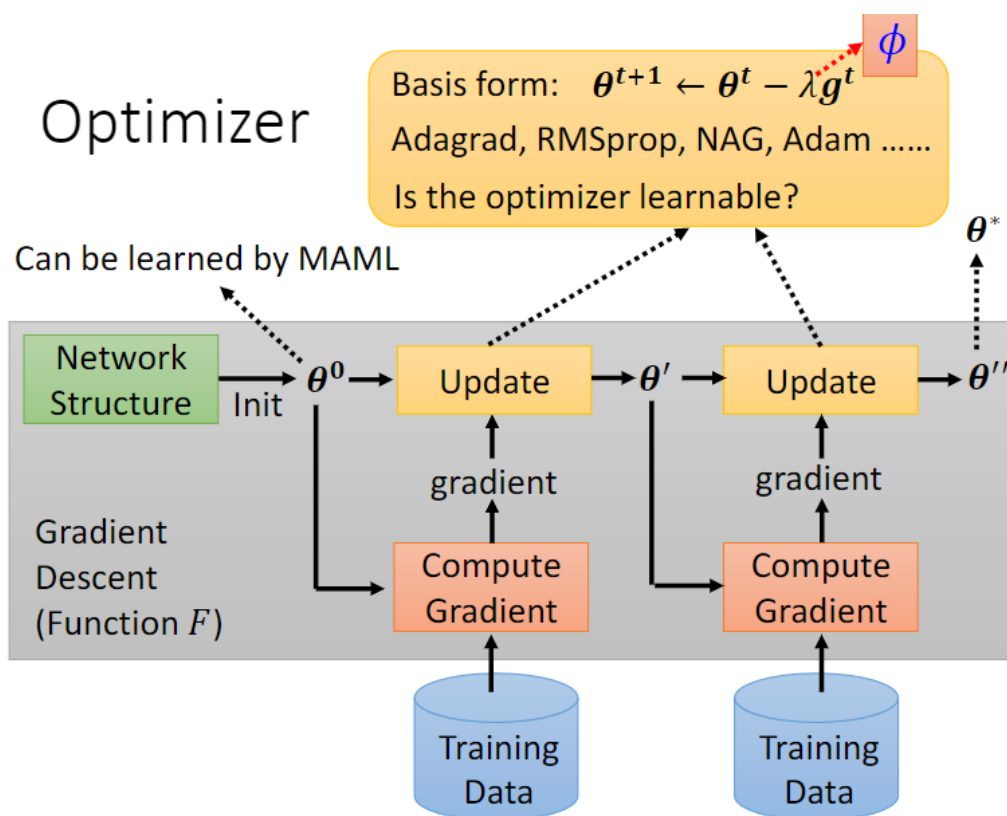
- [Reptile](#)
- First order MAML (FOMAML)

大幅简化MAML运算

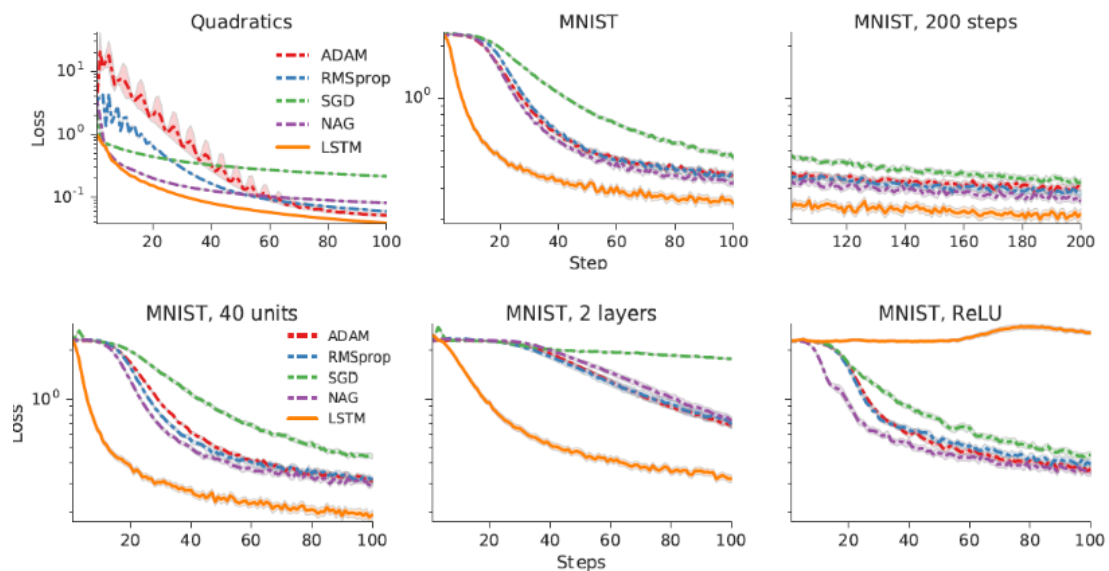
没细讲了，注意做作业。ovo

Learning a Optimizer

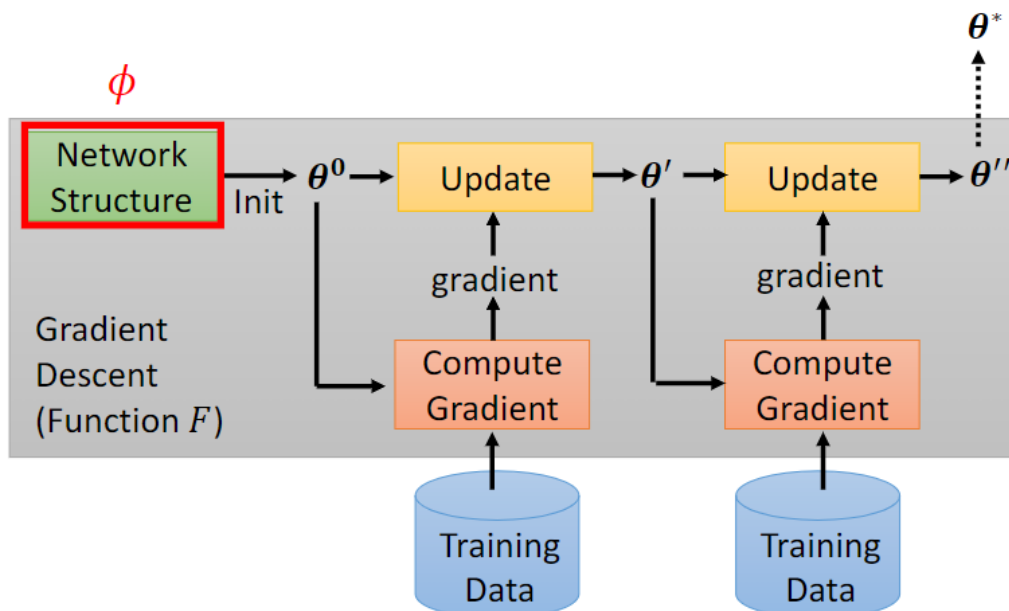
在update参数的时候，把优化器参数自动学习出来。



在文章Marcin Andrychowicz, et al., Learning to learn by gradient descent by gradient descent, NIPS2016中，其中ADAM、RMSprop、SGD、NAG都是人为制作的，作者基于“LSTM”设计了自动制作 Optimizer，效果如下



学网络架构：Network Architecture Search (NAS)



把Network Structure当作 ϕ ，不过 $\nabla_{\phi} L(\phi)$ 无法计算。

- 没法算微分——用Reinforcement Learning硬做或许阔以。

阅读材料有：

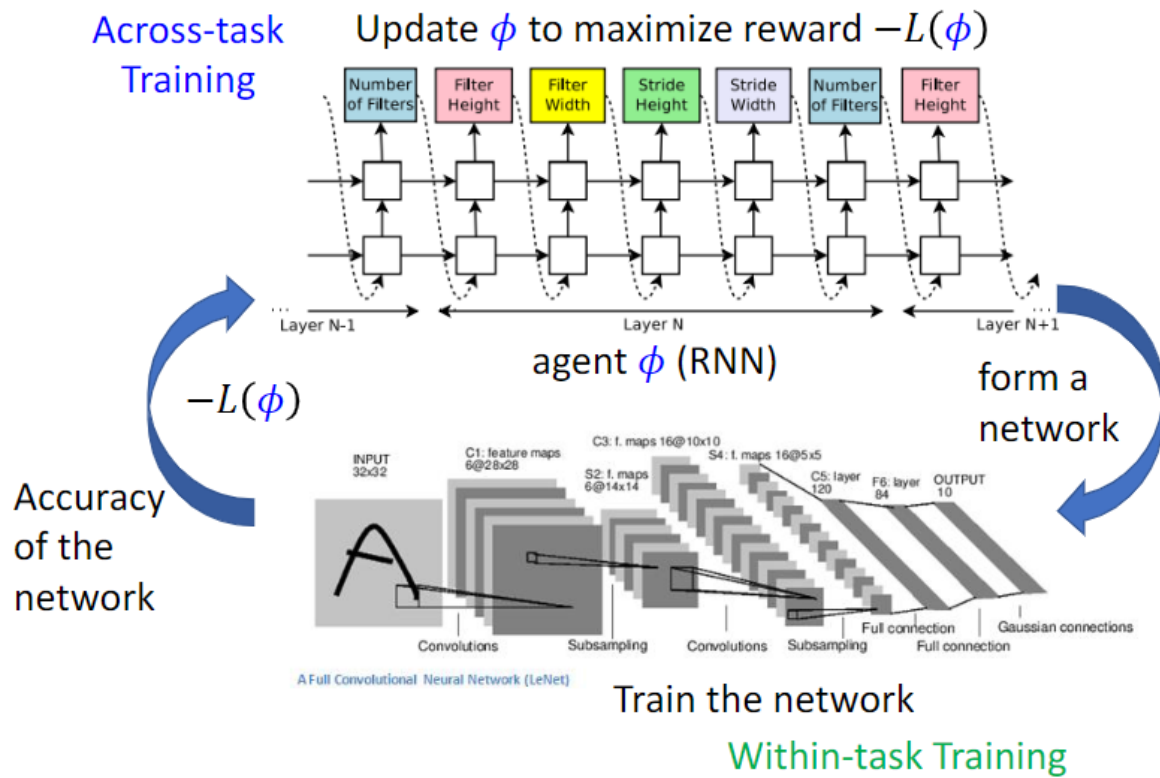
Barret Zoph, et al., Neural Architecture Search with Reinforcement Learning, ICLR 2017

Barret Zoph, et al., Learning Transferable Architectures for Scalable Image Recognition, CVPR, 2018

Hieu Pham, et al., Efficient Neural Architecture Search via Parameter Sharing, ICML, 2018

An agent uses a set of actions to determine the network architecture.

待学习的 ϕ 就是agent的参数，则 $-L(\phi)$ 就是去maximize的reward。用RL硬train一发...示例如下



- 用进化算法 (Evolution Algorithm) 做NAS

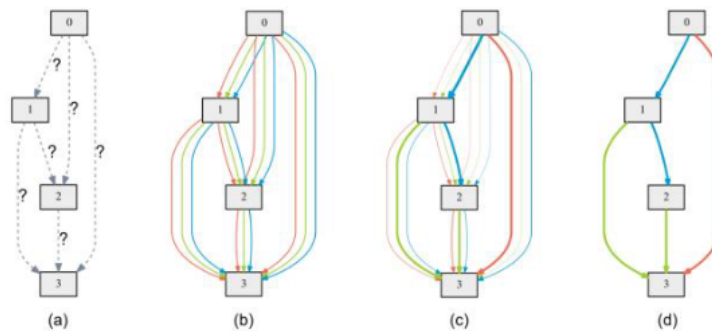
Esteban Real, et al., Large-Scale Evolution of Image Classifiers, ICML 2017

Esteban Real, et al., Regularized Evolution for Image Classifier Architecture Search, AAAI, 2019

Hanxiao Liu, et al., Hierarchical Representations for Efficient Architecture Search, ICLR, 2018

- 折腾点, 设计方法让AES可微分, 来自DARTS

Hanxiao Liu, et al., DARTS: Differentiable Architecture Search, ICLR, 2019



Data Processing

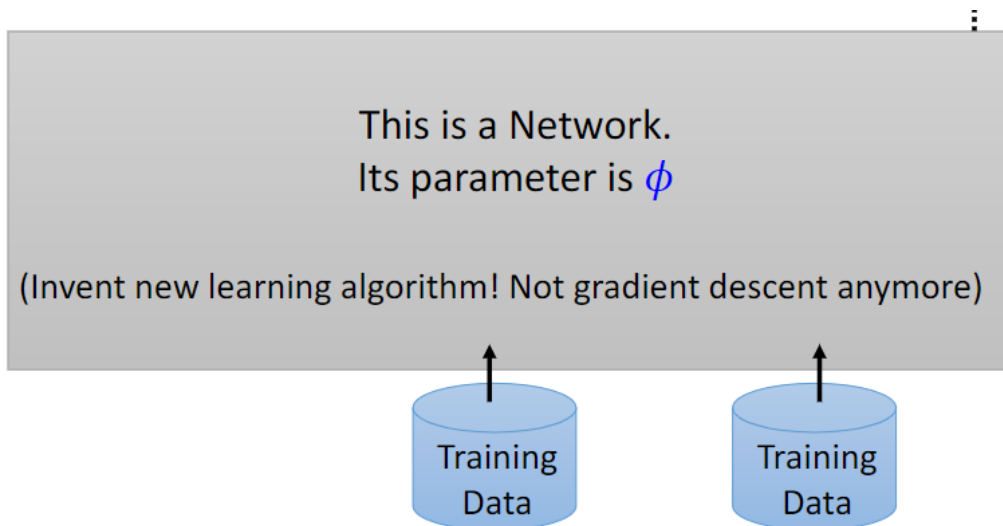
θ^* 

- θ^*

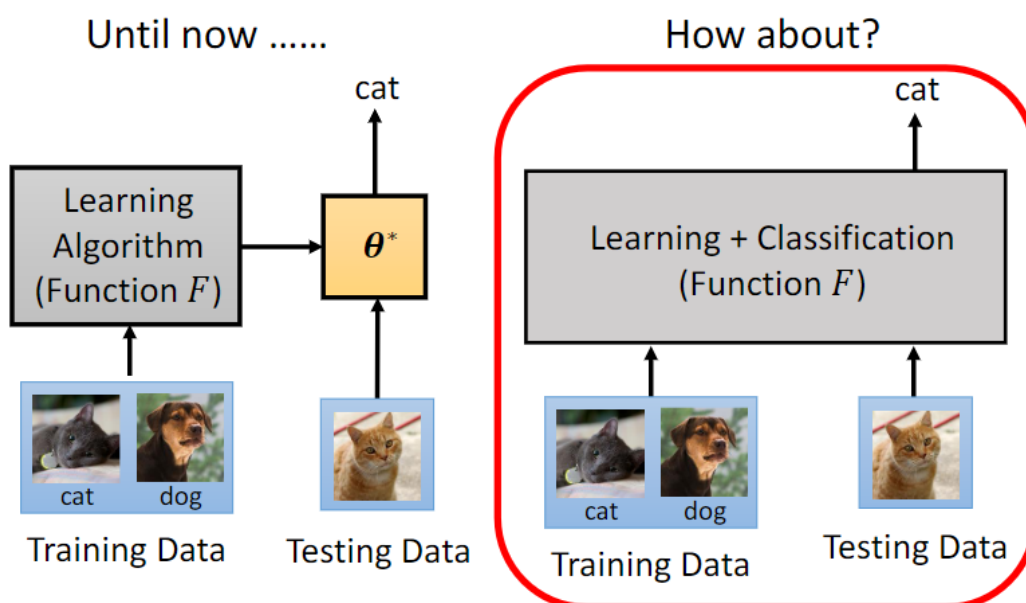
 θ^* θ^* θ^* θ^*

- θ^*

 θ^* θ^* θ^* θ^* θ^* θ^*



把一次training/testing (即一个episode) 包在一个network里边: 把training data丢进去, 再丢进 testing data, network就直接给出测试资料的答案。模糊掉episode里边train和test的边界, 放在一个网络中实现。



这种方法已有了, 详细可去了解 **Learning to compare** (metric-based approach)

应用

用meta learning实现few shot learning

以Few-shot Image Classification为例, 如果每一个class都只有少量的资料

N-ways K-shot classification: 每个任务里边, 有N个classes, 每个class有K个example



3-ways 2-shot

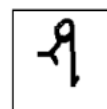
在meta learning中我们需要准备许多N-ways K-shot的任务来作为训练和测试任务。最常见的途径是通过 **Omniglot**。总共有1623个character, 每个character有20个example

以制作20-ways -1shot为例, 在Omniglot中选择20个character, 每个character选择一个example

20 ways
1 shot

Each character
represents a class

ᱠ	ᱡ	ᱢ	ᱣ	ᱤ
ᱥ	ᱦ	ᱧ	ᱨ	ᱩ
ᱪ	ᱫ	ᱬ	ᱭ	ᱮ
ᱯ	ᱰ	ᱱ	ᱲ	ᱳ



Testing set
(Query set)

Training set
(Support set)

Split your characters into training and testing characters

- Sample N training characters, sample K examples from each sampled characters →one training task
- Sample N testing characters, sample K examples from each sampled characters →one testing task

meta learning在具体领域的可能应用

	(A) Learning to initialize	(B) Learning to compare	(C) Other
Sound Event Detection	(Shi et al., 2020)	(Shimada et al., 2020a) (Chou et al., 2019) (Wang et al., 2020) (Shimada et al., 2020b) (Shi et al., 2020)	Network architecture search: (Li et al., 2020)
Keyword Spotting	(Chen et al., 2020a)	(Huh et al., 2020)	Net2Net: (Veniat et al., 2019) Network architecture search: (Mazzawi et al., 2019) Network architecture search: (Mo et al., 2020)
Text Classification	(Dou et al., 2019) (Bansal et al., 2019)	(Yu et al., 2018) (Tan et al., 2019) (Geng et al., 2019) (Sun et al., 2019)	Learning the learning algorithm: (Wu et al., 2019)
Voice Cloning			Learning the learning algorithm: (Chen et al., 2019b) (Serrà et al., 2019)
Sequence Labeling	(Wu et al., 2020)	(Hou et al., 2020)	
Machine Translation	(Gu et al., 2018) (Indurthi et al., 2020)		
Speech Recognition	(Hsu et al., 2020) (Klejch et al., 2019) (Winata et al., 2020a) (Winata et al., 2020b)		Learning to optimize: (Klejch et al., 2018) Network architecture search: (Chen et al., 2020b) (Baruwa et al., 2019)
Knowledge Graph	(Obamuyide and Vlachos, 2019) (Bose et al., 2019) (Lv et al., 2019) (Wang et al., 2019)	(Ye and Ling, 2019) (Chen et al., 2019a) (Xiong et al., 2018) (Gao et al., 2019)	
Dialogue / Chatbot	(Qian and Yu, 2019) (Madotto et al., 2019) (Mi et al., 2019)		Learning to optimize: (Chien and Lieow, 2019)
Parsing	(Guo et al., 2019) (Huang et al., 2018)		
Word Embedding	(Hu et al., 2019)	(Sun et al., 2018)	
Multi-model		(Eloff et al., 2019)	Learning the learning algorithm: (Surís et al., 2019)