Meta Learning

N4A

2018 年 7 月 29 日

1 Introduction

Meta Learning 最早源于上世纪八九十年代 [6], 最近成为研究的热点，这是一个很好的可以用来解决 Learn to learn 问题的框架。17 年 NIPS 有一个 Workshop on Meta Learning 。与迁移学习相比，Meta Learning 可以视为一个更泛化的概念。

传统的机器学习方法为解决某一个特定的任务总是需要大量的训练数据，有一个很直观的原因是因为传统的机器学习方法在训练一个模型时，总是从零开始学习。但是人类的学习过程并不是这样，显然人的学习是一个连续的过程，当一个人想要解决某一个问题时，他会使用之前跟这个任务相关的知识。以图像分类任务为例，传统的机器学习方法，例如普通的 CNN 模型，或者 AlexNet，VGG，ResNet 这些模型都需要大量的训练数据。为了解决数据依赖的问题，现在的一个研究热点就是 “one shot learning” (few shot learning)[4]。许多解决这个问题的方法 [9, 8] 就是基于 Meta Learning。

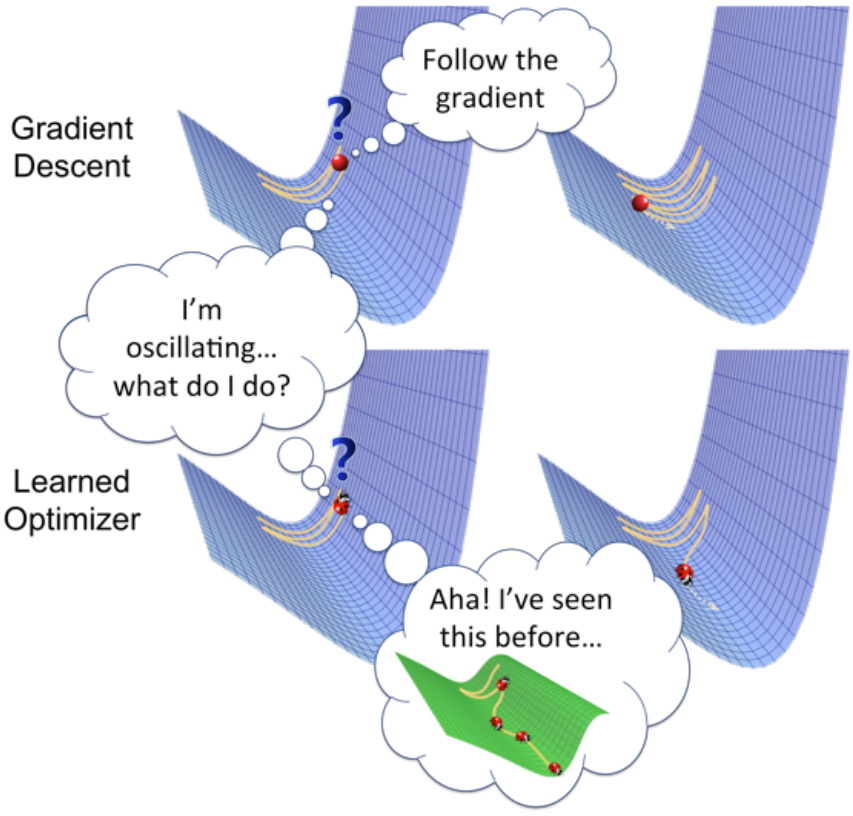


图 1: 自动学习参数更新模型设想效果示意图

做一个粗略的分类可以将 Meta Learning 的应用分为三类：

1. Learn what to learn: 可以理解为学习某个模型在多个相似任务中可以共享的知识，比如说先验，超参，网络前几层的参数。
2. Learn Which model to Learning: 这种方法就是学习设计一个在可以解决同一个任务的多个模型中选择一个优秀的模型。例如自动设计网络结构的应用。
3. Learn How to Learn(Optimize): 一般的网络参数更新方法都是基于反向梯度下降，例如 SGD，ADAM，RMSProp，总的来说都是人为设计的好的更新方法。那为何不设计模型自动学习更新方法呢？现在基本是基于强化学习或者 RNN(LSTM) 来设计自动学习更新参数的模型 [8, 5, 2]。图1是一个示意图以说明自动学习参数更新的一点优势，针对于鞍点问题，自动学习的梯度可能根据记忆而聪明的选择正确的方向。

2 Meta Leaning Framework

我还没有找到一个好的定义，通常，应用 Meta Learning 的任务中包含多个子任务。将子任务的数据集记为 $D_1, ..., D_i, ..., D_N$，一般这些数据集都由训练集和验证集（测试集）组成，每个子任务都有一个学习目标 $L_\lambda(D_i, \theta_i)$，其中 θ_i 为子任务学习模型的参数，一般将子任务的学习模型称为 Learner，在 Meta Learning 框架中，除了各个子任务的 Learner 之外，还有一个 Meta-Learner，Meta-Learner 的目标 $L(D, \theta, \lambda)$ 是综合考量所有子任务，使之在验证集上的损失最小。λ 是各个子任务公共的参数，也是 Meta-Learner 的参数。

3 Overview of some recent works

3.1 Learn what to learn

Matching Networks for One Shot Learning[NIPS16][9]: 这是一个做 one shot learning 任务的模型。本文设计的子任务为给定一个图片集合，该集合中每个类型的图片数为 1 至 5，然后给一张测试图片，判断该图片属于集合中图片的哪一类。在该模型中，不同的子任务使用相同的网络结构来学习图片的特征表示，所以按照 Meta Learning 的框架，在训练过程中，就是要综合考量已有的所有子任务，学习到该场景下特征提取模型的最优参数，然后在新来的子任务中就可以直接复用特征提取模型。

Meta-Learning by Adjusting Priors Based on Extended PAC-Bayes Theory[ICML18][1]: 基于贝叶斯理论的模型中都会有一个先验知识，一般单任务模型中，先验都是人为设定的。本文中，就是综合考量多个相似子任务学习一个先验分布的分布，然后在之后新的类似任务中，就可以直接从先验分布的分布中采样一个先验分布，而不需人为设定。

Transfer Learning via Learning to Transfer[ICML18][10]：迁移学习任务中，有两个难点，一是 what to transfer, 二是 how to transer。一般的迁移学习任务中，需要设计一个模型来提取要迁移的信息的特征表示，例如跨域推荐中的矩阵分解模型, 然后可以使用粗暴的特征共享这一方法解决 how to transfer 的问题。本文中，每个子任务都是一个迁移学习的任务，作者基于不同域中迁移的信息是共享的特征（解决 how to transfer）这一假设，然

后设计一个带参数算法 a 提取要迁移的信息的特征表示。给定样本 x，设 a(x) 是共享的特征，然后设计一个评价函数 f 评价迁移的效果如何，综合考量所有子任务，学习到评价函数 f 最优参数 (文中对于训练集，特征提取算法 a 是预训练好的)，在之后新的子任务中就可以直接利用评价函数 f 学习特征提取算法 a 的参数，也就是学到了 what to transer.

3.2 Learn how to learn(optimize)

Optimization as a method for few-shot learning[ICLR17][8]: 作者利用 LSTM 设计自动学习参数的模型，普通的参数更新公式如下:

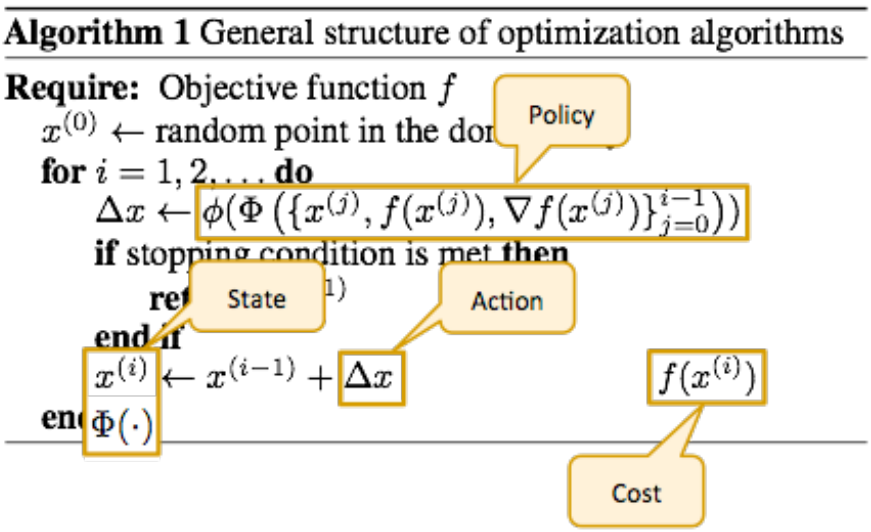
$$\theta_t = \theta_{t-1} - \alpha_t \nabla_{\theta_{t-1}} L_t \tag{1}$$

LSTM 的细状态更新公式如下:

$$c_t = f_t \odot c_{t-1} + i_t \odot \hat{c}_t \tag{2}$$

比较之下只要令 $f_t = 1, c_{t-1} = \theta_{t-1}, i_t = \alpha_t, \hat{c}_t = -\nabla_{\theta_{t-1}} L_t$ 就可以得到普通参数更新方法的这个特例。所以利用 LSTM 自动学习参数的更新方法是一个很自然的选择。

Learning to Optimize[5]: 这个思路是一样的，将梯度下降规约为加强学习的一个特例，然后就可以用加强学习学习自动更新参数的方法，如下图所示:



3.3 Theory works

Bilevel Programming for Hyperparameter Optimization and Meta-Learning[ICML18][3]: 作者提出了一个更泛化的框架统一解释 Hyperparameter Optimization and Meta-Learning，然后给出了统一的优化方式。文中引用的超参优化方法是先在训练集上优化参数，然后在验证集上优化超参数。我的理解是有点强行定义概念啊，超参优化应该是一个任务，Meta-Learning 是一种方法，meta-learning 也可以做超参优化。文中给的该是超参优化的一种方式。以这种超参优化方法和 meta-learning 并列相比实在是有点弱了，所以所谓泛化的框架就像一个再解释的 meta-learning 框架。

Pseudo-task Augmentation: From Deep Multitask Learning to Intratask Sharing—and Back[ICML18][7]: 文中归纳普通的 Multi-task learning 中多个任务有共享的信息，设计模型时有一个底层的共享的 embedding 模型，然后每个任务有各自的 decoder 模型做自己的工作。本文的工作就是作者提出一个任务也可以用多个 decoder 来处理，每个 decoder 来处理这个任务就称为一个 Pseudo-task，然后作者在理论和实验上证明了这样做确实是有效的。

4 Matching Networks for One Shot Learning

4.1 Model Overview

这篇文章的工作是做 One（Few）Shot Learning, 作者强调一个机器学习模型设计的基本原理：test and train conditions must match。既然我们想做 One Shot Learning，那么在训练的时候每个类别的图片就不能太多，所以作者设计了训练的（子）任务：给定一个图片集合，称为 Support Set，该集合中每个类型的图片数为 1 至 5，然后给定任意一张测试图片，训练设计好的模型判断该图片属于 Support Set 中的哪一类图片。综合考量多个这样的（子）任务，确定模型的参数。然后做图片分类测试的时候也是一样，先选取一个有 Label 的 Support Set, 然后对于给定的任意一张图片，利用之前训练好的模型做出分类，即决定该图片属于 Support Set 中的哪一类。

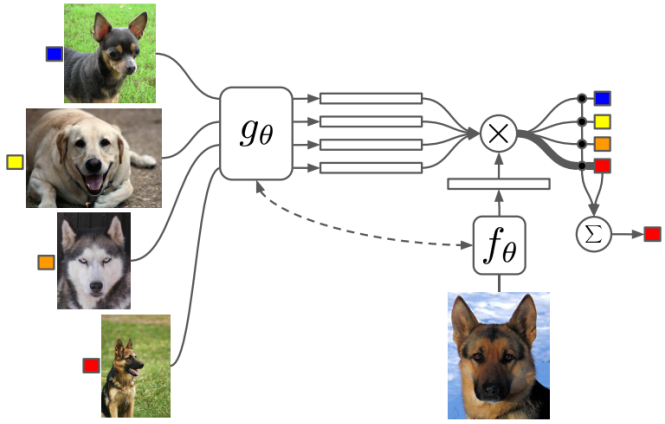
作者设计的子任务分类模型，也就是 Meta Learning 中的 Learner 就是一个非参的 KNN 模型的一个变种。对于一个样本 $\hat{x}$, 使用如下公式预测其 label $\hat{y}$:

$$\hat{y} = \sum_{i=1}^{k} a(\hat{x}, x_i) y_i \tag{3}$$

其中, x_i, y_i 就是 Support Set $S = \{(x_i, y_i\}_{i=1}^{k}$ 中的样本和 label。函数 a 文中称为注意力计算函数，也就是 KNN 中的相似度计算函数，其定义如下：

$$a(\hat{x}, x_i) = e^{c(f(\hat{x}), g(x_i))} / \sum_{j=1}^{k} e^{c(f(\hat{x}), g(x_j))}$$

其中，函数 f, g 是两个特征提取函数。这两个函数都用有参数的神经网络模型实现。该预测过程网络示意图如下，其中符号 $\otimes$ 表示前面公式 3 的操作：



然后训练目标就是综合考量多个子任务，以确定特征提取函数 f, g 的参数，如下：

$$\theta = \arg\max_{\theta} E_{L\sim T}\left[E_{S\sim L, B\sim L}\left[\sum_{(x,y)\in B} \log P_\theta\left(y|x, S\right)\right]\right]$$

其中，T 是关于所有 label 的一个分布，从中采样一个 Label 集合 L，然后从 L 采样 S 即为 Support Set，采样训练样本的 Batch B，$E_{S\sim L, B_L}[.]$ 即为子任务 Learner 的目标，$E_{L\sim T}$ 即为 Meta-Learner 的目标。注意对比一下第二部分中的 Meta Learning Framework，这里 θ 相当于 λ，而子任务是只有公共的参数，没有各自独有的参数，因为选取的 Learner 是非参的 KNN 模型。

作者指出在提取样本特征的时候如果只考虑样本本身是非常局限的，所以作者提出了要让 support set 的所有样本调节每一个样本的特征提取，包括 support set 里的样本和用于测试的样本。接下来介绍作者设计的函数 f 和 g。

4.2 The Fully Conditional Embedding g

这部分介绍一下前文的特征提取函数 g 的结构设计，作者使用双向 LSTM 来实现特征提取，如下图：

In section 2.1.2 we described the encoding function for the elements in the support set S, $g(x_i, S)$, as a bidirectional LSTM. More precisely, let $g'(x_i)$ be a neural network (similar to f' above, e.g. a VGG or Inception model). Then we define $g(x_i, S) = \vec{h}_i + \overleftarrow{h}_i + g'(x_i)$ with:

$$\begin{aligned} \vec{h}_i, \vec{c}_i &= \text{LSTM}(g'(x_i), \vec{h}_{i-1}, \vec{c}_{i-1}) \\ \overleftarrow{h}_i, \overleftarrow{c}_i &= \text{LSTM}(g'(x_i), \overleftarrow{h}_{i+1}, \overleftarrow{c}_{i+1}) \end{aligned}$$

where, as in above, $\text{LSTM}(x, h, c)$ follows the same LSTM implementation defined in [23] with x the input, h the output (i.e., cell after the output gate), and c the cell. Note that the recursion for $\overleftarrow{h}$ starts from $i = |S|$. As in eq. 4, we add a skip connection between input and outputs.

值得注意的是，双向 LSTM 的迭代过程中，输入的顺序是很重要的，但是文中 Support Set 中的元素不是有序的，文中没有交代怎么处理顺序的问题。

4.3 The Fully Conditional Embedding f

这部分介绍一下前文的特征提取函数 f 的结构设计。前面的函数 g 提取出 support set 中元素的特征，每次的输入是集合中下一个元素，但是用于判别的样本每次只有一个，所以要用新的结构来提取特征。作者使用 LSTM 来实现。迭代过程中，输入都是不变的样本原始特征，如下图所示。其次为了考虑 Supoort set 的影响，每次迭代过程中隐藏层要根据 Support set 中样本的特征做一点变换。值得注意：下图公式（5）中 r_{k-1} 应该是 r_k，网络上已经指出了这个笔误，但是论文中没有修改。

As described in section 2.1.2, the embedding function for an example $\hat{x}$ in the batch B is as follows:

$$f(\hat{x}, S) = \text{attLSTM}(f'(\hat{x}), g(S), K)$$

where f' is a neural network (e.g., VGG or Inception, as described in the main text). We define K to be the number of "processing" steps following work from [26] from their "Process" block. $g(S)$ represents the embedding function g applied to each element x_i from the set S.

Thus, the state after k processing steps is as follows:

$$\hat{h}_k, c_k = \text{LSTM}(f'(\hat{x}), [h_{k-1}, r_{k-1}], c_{k-1}) \quad (3)$$

$$h_k = \hat{h}_k + f'(\hat{x}) \quad (4)$$

$$r_{k-1} = \sum_{i=1}^{|S|} a(h_{k-1}, g(x_i)) g(x_i) \quad (5)$$

$$a(h_{k-1}, g(x_i)) = \text{softmax}(h_{k-1}^T g(x_i)) \quad (6)$$

noting that $\text{LSTM}(x, h, c)$ follows the same LSTM implementation defined in [23] with x the input, h the output (i.e., cell after the output gate), and c the cell. a is commonly referred to as "content" based attention, and the softmax in eq. 6 normalizes w.r.t. $g(x_i)$. The read-out r_{k-1} from $g(S)$ is concatenated to h_{k-1}. Since we do K steps of "reads", $\text{attLSTM}(f'(\hat{x}), g(S), K) = h_K$ where h_k is as described in eq. 3.

5 Meta-Learning by Adjusting Priors Based on Extended PAC-Bayes Theory

5.1 PAC-Bayes Theory on single task

记 S 为训练数据的集合，D 为数据样本符合的分布，h 为假设解决问题的模型，那么可以定义泛化误差（generalization error）如下：

$$er(h, D) = \mathbb{E}_{z \sim D} l(h, z)$$

定义经验误差（empirical error）如下：

$$\hat{er}(h, S) = (1/m) \sum_{i=1}^{m} l(h, z_i)$$

PAC-Bayes 方法假设所有解决问题的算法在空间 H 内，PAC-Bayes 方法就是要求得关于该空间的后验分布 Q, 然后可以计算关于 Q, D 的泛化误差和经验误差如下：

$$er(Q, D) = \mathbb{E}_{h \sim Q} er(h, D)$$

$$\hat{er}(Q, S) = \mathbb{E}_{h \sim Q} \hat{er}(h, S)$$

PAC-Bayes 方法的目标就是求得 Q 使得泛化误差最小。但是由于泛化误差依赖数据分布 D，但是该分布又是我们不知道的，该如何是好。有理论估计了泛化误差的上界，该上界依赖于经验误差和其它已知元素。这样我们又可以通过优化泛化误差上界来优化泛化误差。泛化误差上界结论如下：

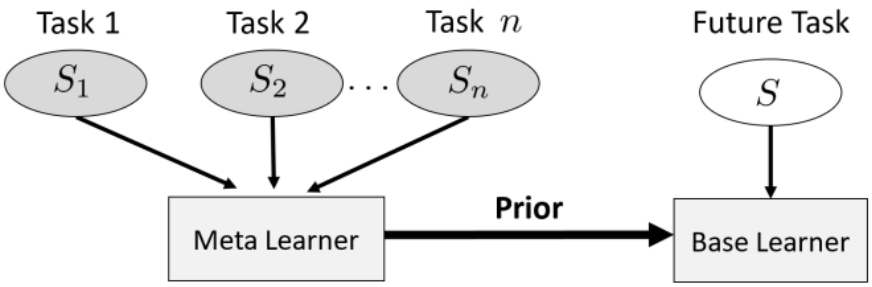
Theorem 1 (McAllester's single-task bound) *Let $P \in \mathcal{M}$ be some prior distribution over $\mathcal{H}$. Then for any $\delta \in (0,1]$, the following inequality holds uniformly for all posteriors distributions $Q \in \mathcal{M}$ with probability at least $1-\delta$,*

$$er(Q,\mathcal{D}) \leq \hat{er}(Q,S) + \sqrt{\frac{D(Q||P) + \log\frac{m}{\delta}}{2(m-1)}}$$

where $D(Q||P)$ is the Kullback-Leibler (KL) divergence, $D(Q||P) \triangleq \mathop{\mathbb{E}}_{h\sim Q} \log\frac{Q(h)}{P(h)}$.

5.2 PAC-Bayes Meta Learning

在单任务模型中，我们要人为设定假设模型的先验分布 P, 然后根据数据求得后验分布 Q 使得泛化误差的上界最小。在 Meta Learning 环境中，作者设想要从之前的相似子任务中学习到先验分布的公共知识。然后在之后的类似任务中就可以直接采样一个先验分布，而不需人为设定（避免人为设定需要不断手工调节超参的麻烦，效果也有所提升）。示意图如下：



下面从公式化角度做出推导。记所有的 (子) 任务拥有共同的样本空间 Z, 假设模型空间 H, 损失函数 $l: HXZ \to [0,1]$。对于每一（子）任务 t, 其数据分布记为 D_t，观测到的样本数据集合记为 S_t, 样本的个数记为 m_t。假设每个数据分布 D_t 都是从分布 τ 采样得到，满足独立同分布性质。每个子任务关于假设模型 h_t 的公共先验 $P(h)$ 都采样于相同的超先验分布 $\mathbb{P}(p)$。我们的目标是综合考量多个子任务，让每个子任务的 learner 会学习到各自的后验分布 $Q(P,S_t)$，让 meta-learner 学习到这个超先验分布的后验分布，记为超后验 $\mathbb{Q}(P)$。这样对于新来的任务，我们就可以直接从超后验分布中采样先验知识。学习超后验的泛化误差定义如下 (公式中需注意超后验符号 $\mathbb{Q}$ 和每个子任务的后验符号 Q 的不同)：

$$er(\mathbb{Q},\tau) = \mathbb{E}_{P\sim\mathbb{Q}} er(P,\tau)$$

经验误差如下：

$$\hat{er}(\mathbb{Q},S_1,...,S_n) = \mathbb{E}_{P\sim Q}\frac{1}{n}\sum_{i=1}^{n}\hat{er}(Q(S_i,P),S_i)$$

同样，泛化误差不可以直接计算，我们需要估计其上界，如下 (注意图中超后验符号上文略有不同，我暂时不会输他这样的符号)：

Theorem 2 (Meta-learning PAC-Bayes bound) *Let $Q : \mathcal{Z}^m \times \mathcal{M} \to \mathcal{M}$ be a base learner, and let $\mathcal{P}$ be some pre-defined hyper-prior distribution. Then for any $\delta \in (0, 1]$ the following inequality holds uniformly for all hyper-posterior distributions $\mathcal{Q}$ with probability at least $1 - \delta$,* [4]

$$er\left(\mathcal{Q}, \tau\right) \leq \frac{1}{n} \sum_{i=1}^{n} \mathop{\mathbb{E}}_{P \sim \mathcal{Q}} \widehat{er}_i\left(Q_i, S_i\right) \tag{4}$$

$$+\frac{1}{n} \sum_{i=1}^{n} \sqrt{\frac{D(\mathcal{Q}||\mathcal{P}) + \mathop{\mathbb{E}}_{P \sim \mathcal{Q}} D(Q_i||P) + \log \frac{2nm_i}{\delta}}{2(m_i - 1)}}$$

$$+\sqrt{\frac{D(\mathcal{Q}||\mathcal{P}) + \log \frac{2n}{\delta}}{2(n-1)}},$$

where $Q_i \triangleq Q(S_i, P)$.

5.3 Implementation details

上图公式 4 的上界涉及到一个采样的过程，也难以直接使用随机梯度下降的方法，所以我们要使用蒙特卡洛方法估计这个上界，但是同时我们想在采样的先验 P 中保留超后验的 $\mathbb{Q}$ 的参数。不然在优化过程中就学习不到这个后验，在接下来的任务中也难以使用这个超后验知识了。文中采用了类似于 VAE 中的处理方法。使得在采样的 P 中保留 $\mathbb{Q}$ 的参数。

文中选择的超后验分布就是常用的高斯分布。定义如下，其中 $k_{\mathbb{Q}} > 0$ 是人工设定的超参数。

$$\mathbb{Q}_\theta = N(\theta, k_{\mathbb{Q}}^2 I_{N_P X N_P})$$

超先验分布定义为以 0 为均值的高斯分布，如下：

$$\mathbb{P} = N(0, k_{\mathbb{P}}^2 I_{N_P X N_P})$$

这样上图泛化误差上界中 KL 散度可以求得为：$D(\mathbb{Q}_\theta \| \mathbb{P}) = \frac{1}{2k_{\mathbb{P}}^2} \|\theta\|_2^2$。同时，与 VAE 中相同，从 $\mathbb{Q}_\theta$ 中采样一个分布 P 的参数 $\hat{\theta}$ 可以视为在参数 θ 上加一个随机采样的噪声，如下：

$$\hat{\theta} = \theta + \epsilon_P, \epsilon_P \sim N(0, k_{\mathbb{Q}}^2 I_{N_P X N_P})$$

采样 K 个样本 $\hat{\theta}_1, ..., \hat{\theta}_K$，就可以对上图中误差上界中的 $E_{P \sim \mathbb{Q}}$ 做一个估计，如下：

$$\mathbb{E}_{P \sim \mathbb{Q}_\theta} \hat{er}_i(Q_i, S_i) \approx \frac{1}{K} \sum_{j=1}^{K} \hat{er}_i(Q_i(S_i, P_{\hat{\theta}_j}), S_i)$$

然后，文中分布 P, Q 也是使用带参数的高斯分布，如下图所示。其中，$\phi_i = (\mu_i, \rho_i), \theta = (\mu_P, \rho_P)$

$$P_\theta(w) = \prod_{k=1}^{d} \mathcal{N}\left(w_k; \mu_{P,k}, \sigma_{P,k}^2\right)$$

$$Q_{\phi_i}(w) = \prod_{k=1}^{d} \mathcal{N}\left(w_k; \mu_{i,k}, \sigma_{i,k}^2\right)$$

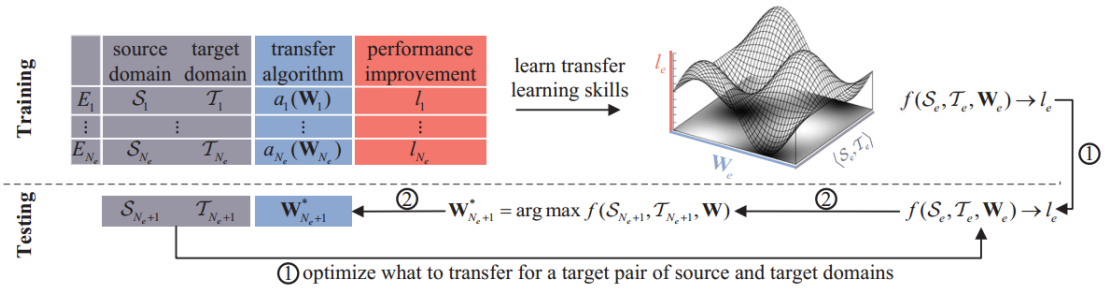
总体来说，该泛化误差上界的具体实现包含两部分参数，其一是每个子任务的 learner 要学习的后验 Q 的参数 ϕ_i, 相当于前文 meta-learning 框架中的 θ_i, 其次是 meta-learner 要学习的超后验 $\mathbb{Q}$ 的参数 θ, 相当于前文框架中的参数 λ。本文中采用了两阶段训练的方法，首先独自训练每个子任务，学习参数 ϕ, 这时候共同先验 $P_{\hat{\theta}}$ 的参数就固定不变，这就和单任务训练一样，相当于超参。待子任务训练好之后，再训练 meta-learner 的参数 θ。（文中没有交代是否划分训练集训练子任务，验证集训练 meta-learner, 一般是这么做的）

6 Transfer Learning via Learning to Transfer

6.1 Model overview

迁移学习任务中，有两个难点，一是 what to transfer, 二是 how to transer。一般的迁移学习任务中，需要设计一个模型来提取要迁移的信息的特征表示，例如跨域推荐中的矩阵分解模型, 然后可以使用粗暴的特征共享这一方法解决 how to transfer 的问题。

本文中，每个子任务都是一个迁移学习的任务，作者基于不同域中迁移的信息是共享的特征（解决 how to transfer）这一假设，然后设计一个带参数算法 a 提取要迁移的信息的特征表示。给定样本 x，设 a(x) 是共享的特征，然后设计一个评价函数 f 评价迁移的效果如何，综合考量所有子任务，学习到评价函数 f 最优参数 (文中对于训练集，特征提取算法 a 是预训练好的)，在之后新的子任务中就可以直接利用评价函数 f 学习特征提取算法 a 的参数，也就是学到了 what to transer. 框架示意图如下图所示, 图中 Training 过程即学习 f，Testing 过程即学习 a 的参数 W。



根据该框架，模型的要点就是如何设计 a 和 f，接下来两部分做出介绍。

6.2 Design a

作者做了一个大胆的假设就是样本在不同域中有些特征是不变的 (domain-invariant feature)，比如在图像分类任务中，眼睛，嘴巴，尾巴这些特征。然后如何提取特征，作者分为 Common Latent Space Based 和 Manifold Ensemble Based 讨论，结论就是基本上都可以用线性方法近似，即对于一个样本 x，其共享特征提取算法都近似为 $a(x) \approx Wx$。其中，W 是线性方法的参数矩阵。

6.3 Design f

函数 f 是用来衡量迁移效果的好坏，作者首先考量的就是源域和目标域之间共享特征的距离应当小，作者定义如下图所示的距离函数：

$$
\begin{aligned}
&\hat{d}_e^2(\mathbf{X}_e^s\mathbf{W}_e, \mathbf{X}_e^t\mathbf{W}_e) \\
=&\left\| \frac{1}{n_e^s}\sum_{i=1}^{n_e^s}\phi(\mathbf{x}_{ei}^s\mathbf{W}_e) - \frac{1}{n_e^t}\sum_{j=1}^{n_e^t}\phi(\mathbf{x}_{ej}^t\mathbf{W}_e)\right\|_{\mathcal{H}}^2 \\
=&\frac{1}{(n_e^s)^2}\sum_{i,i'=1}^{n_e^s}\mathcal{K}(\mathbf{x}_{ei}^s\mathbf{W}_e, \mathbf{x}_{ei'}^s\mathbf{W}_e) \\
+&\frac{1}{(n_e^t)^2}\sum_{j,j'=1}^{n_e^t}\mathcal{K}(\mathbf{x}_{ej}^t\mathbf{W}_e, \mathbf{x}_{ej'}^t\mathbf{W}_e) \\
-&\frac{2}{n_e^s n_e^t}\sum_{i,j=1}^{n_e^s,n_e^t}\mathcal{K}(\mathbf{x}_{ei}^s\mathbf{W}_e, \mathbf{x}_{ej}^t\mathbf{W}_e),
\end{aligned}
$$

作者认为只使用距离还不够，限制所有对共享特征的距离的方差可以提高稳定性，如下：

$$
d_e^2(\mathbf{X}_e^s\mathbf{W}_e, \mathbf{X}_e^t\mathbf{W}_e) = \mathbf{E}_{\mathbf{x}_e^s\mathbf{x}_e^{s\prime}\mathbf{x}_e^t\mathbf{x}_e^{t\prime}} h(\mathbf{x}_e^s, \mathbf{x}_e^{s\prime}, \mathbf{x}_e^t, \mathbf{x}_e^{t\prime})
$$

$$
\begin{aligned}
\sigma_e^2(\mathbf{X}_e^s\mathbf{W}_e, \mathbf{X}_e^t\mathbf{W}_e) =& \mathbf{E}_{\mathbf{x}_e^s\mathbf{x}_e^{s\prime}\mathbf{x}_e^t\mathbf{x}_e^{t\prime}}[(h(\mathbf{x}_e^s, \mathbf{x}_e^{s\prime}, \mathbf{x}_e^t, \mathbf{x}_e^{t\prime}) \\
&- \mathbf{E}_{\mathbf{x}_e^s\mathbf{x}_e^{s\prime}\mathbf{x}_e^t\mathbf{x}_e^{t\prime}} h(\mathbf{x}_e^s, \mathbf{x}_e^{s\prime}, \mathbf{x}_e^t, \mathbf{x}_e^{t\prime}))^2].
\end{aligned}
$$

之后，要考量目标域样本的可区别性，要保持相似的样本距离近，不相似的样本距离远，如下：

$$\tau_e = \mathrm{tr}(\mathbf{W}_e^T \mathbf{S}_e^N \mathbf{W}_e)/\mathrm{tr}(\mathbf{W}_e^T \mathbf{S}_e^L \mathbf{W}_e),$$

where $\mathbf{S}_e^L = \sum_{j,j'=1}^{n_e^t} \frac{H_{jj'}}{(n_e^t)^2}(\mathbf{x}_{ej}^t - \mathbf{x}_{ej'}^t)(\mathbf{x}_{ej}^t - \mathbf{x}_{ej'}^t)^T$ is the local scatter covariance matrix with the neighbour information $H_{jj'}$ defined as $H_{jj'} = \begin{cases} \mathcal{K}(\mathbf{x}_{ej}^t, \mathbf{x}_{ej'}^t), & \text{if } \mathbf{x}_{ej}^t \in \mathcal{N}_r(\mathbf{x}_{ej'}^t) \text{ and } \mathbf{x}_{ej'}^t \in \mathcal{N}_r(\mathbf{x}_{ej}^t) \\ 0, & \text{otherwise} \end{cases}$.

然后函数 f 就是以上三部分的组合：，如下：

$$1/f = \boldsymbol{\beta}^T \hat{\mathbf{d}}_e + \lambda \boldsymbol{\beta}^T \hat{\mathbf{Q}}_e \boldsymbol{\beta} + \frac{\mu}{\boldsymbol{\beta}^T \boldsymbol{\tau}_e} + b$$

最后作者优化的目标不是最小化 1/f，即使得目标域和源域的共享隐特征之间距离变小，方差变小，目标域样本可识别性变大。而是使之趋向于某个值，作者定义了 improvement ratio: $l_e = p_e^{st}/p_e^t$，其中 p_e^t 是没有使用迁移信息时目标域的效果（performance），p_e^{st} 时使用了迁移信息之后的目标域的效果。训练目标是使得 1/f 趋近 $1/l_e$，如下:

$$\begin{aligned} &\boldsymbol{\beta}^*, \lambda^*, \mu^*, b^* = \\ &\arg\min_{\boldsymbol{\beta},\lambda,\mu,b} \sum_{e=1}^{N_e} \mathcal{L}_h \left(\boldsymbol{\beta}^T \hat{\mathbf{d}}_e + \lambda \boldsymbol{\beta}^T \hat{\mathbf{Q}}_e \boldsymbol{\beta} + \frac{\mu}{\boldsymbol{\beta}^T \boldsymbol{\tau}_e} + b, \frac{1}{l_e} \right) \\ &\qquad + \gamma_1 R(\boldsymbol{\beta}, \lambda, \mu, b), \\ &\text{s.t.} \quad \beta_k \geq 0, \ \forall k \in \{1, \cdots, N_k\}, \ \lambda \geq 0, \ \mu \geq 0, \end{aligned} \qquad (\text{}$$

其中，$L_h(x, y)$ 是 Huber regression loss, 目标是使得 x 趋向于 y。这么做是要使得 f 的评价标准与真实的提高效率有正相关的对应吧。也没有设计实验对比效果。

6.4 generalization error bound

文章最后推导了泛化误差上界，如下图所示，但是前文作者已经经验化的定义好了优化目标，最后推导一下泛化误差上界，可以从理论上增强说服力，但是其具体实现不是根据泛化误差上界设计优化目标，其说服力并不是很强。

Theorem 2. *Let* $\delta' = \delta - (N_e)^{\frac{1}{2(r+1)} - \frac{1}{4}}$ *(*$r > 1$ *is required). Then for any sample* $\mathbf{S}$ *of size* N_e *drawn according to an algebraic* β*-mixing stationary distribution, and* $\delta \geq 0$ *such that* $\delta' \geq 0$*, the following generalization bound holds with probability at least* $1 - \delta$*:*

$$\left| R(\mathbf{L}(\mathbf{S})) - R_{N_e}(\mathbf{L}(\mathbf{S})) \right| < \mathcal{O}\left((N_e)^{\frac{1}{2(r+1)} - \frac{1}{4}} \sqrt{\log(\frac{1}{\delta'})} \right),$$

where $R(\mathbf{L}(\mathbf{S}))$ *and* $R_{N_e}(\mathbf{L}(\mathbf{S}))$ *denote the expected risk and the empirical risk of L2T over meta-samples, respectively. A larger mixing parameter* r*, indicating more independence, would lead to a tighter bound.*

7 Optimization as a model for few shot learning

7.1 Form Gradient descent to LSTM

虽然有很多的变种，但是普通的参数更新公式基本如下：

$$\theta_t = \theta_{t-1} - \alpha_t \nabla_{\theta_{t-1}} L_t$$

LSTM 的细状态更新公式如下：

$$c_t = f_t \odot c_{t-1} + i_t \odot \hat{c}_t$$

比较之下只要令 $f_t = 1, c_{t-1} = \theta_{t-1}, i_t = \alpha_t, \hat{c}_t = -\nabla_{\theta_{t-1}} L_t$ 就可以得到普通参数更新方法的这个特例。所以利用 LSTM 自动学习参数的更新方法是一个很自然的选择。

在这个环境下，meta-learner 就是 LSTM，Learner 就是图像分类模型，其参数为 θ，作者令细胞状态 $c_t = \theta_t$，于是就可以通过 meta-learner LSTM 学习各个子任务的参数。与前面的 meta-learning 框架比较，这里的 θ 相当于 λ, 每个子任务没有自己独立的参数。

在普通的参数更新方法下 learning rate 就是一个常数 α_t, 但是在 LSTM 中，我们可以控制输入门，使得细胞状态得到更有效的更新。如下：

$$i_t = \sigma\left(\mathbf{W}_I \cdot \left[\nabla_{\theta_{t-1}} \mathcal{L}_t, \mathcal{L}_t, \theta_{t-1}, i_{t-1}\right] + \mathbf{b}_I\right),$$

这表示当前的学习率于 θ_{t-1} $\nabla_{\theta_{t-1}} L_t, L_t, i_{t-1}$ 相关，使用这些信息，模型希望学习到更好的策略控制学习率。

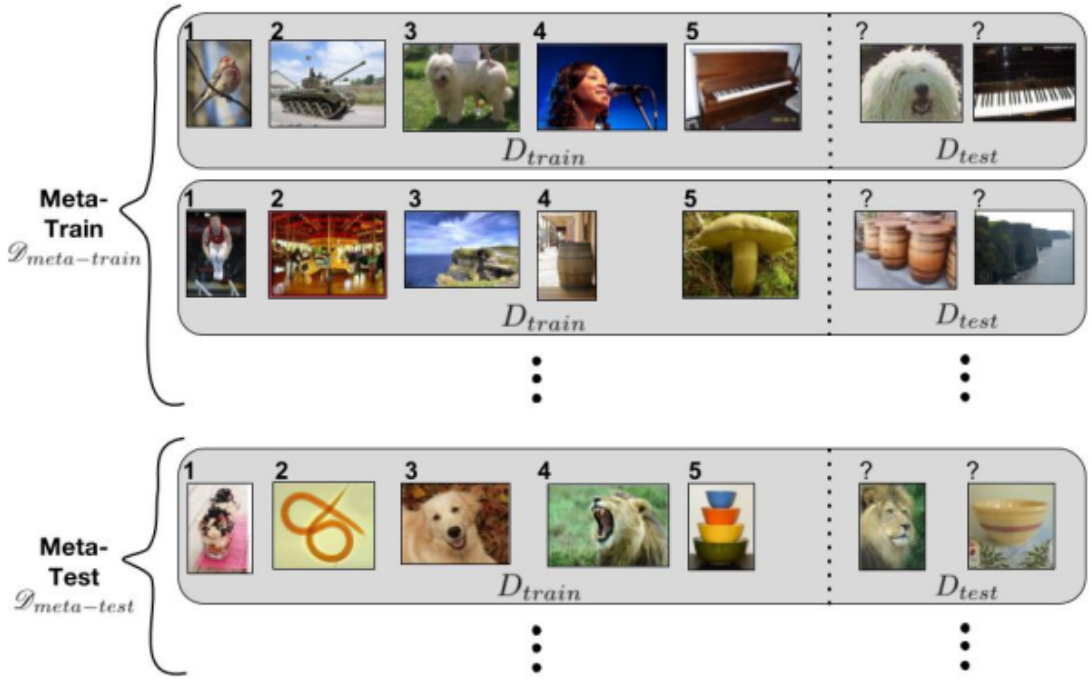
对于遗忘门也是一样，总是取常数 1 肯定不是最优的，作者希望学习到一个好的遗忘策略，如下：

$$f_t = \sigma\left(\mathbf{W}_F \cdot \left[\nabla_{\theta_{t-1}} \mathcal{L}_t, \mathcal{L}_t, \theta_{t-1}, f_{t-1}\right] + \mathbf{b}_F\right)$$

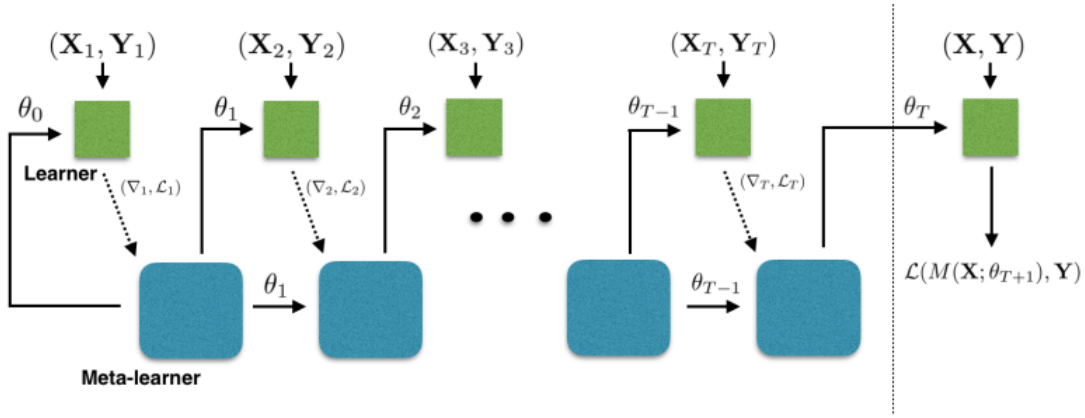
7.2 For few shot Learning

模型的一个子任务就是普通的图片分类任务，如下图是一个 One Shot Learning 的示意图，每一个子任务，都有一个 D_{train} 用以更新 learner 的参数（在 Meta-learner 的调控下更新），然后在 D_{test} 数据上评估，Meta-learner 就是综合考量多个子任务 D_{test} 上的评估来更新 Meta-learner 的参数，以学会自动更新 Learner 的参数。

当 Meta-learner 的参数更新好以后，对于一个新的子任务 Meta-test, Meta-learner 就会根据这个子任务 D_{train} 中的数据，自动更新适应这个子任务的 learner（图片分类器）的参数，更新好之后，就可以在 D_{test} 上测试评估。



模型一个子任务的学习框架如下图所示，图中 (X_i, Y_i) 是一个 batch 的数据，(X, Y) 为测试样本，虚线箭头表示参数更新时，不沿着该线反向传播，这是一个简化，否则更新 meta-learner 的参数就会变得复杂（这条线的依赖会带来 meta-learner 参数的二阶导数）。在所有的子任务中，Meta-leaner 是不变的，Learner 是重新初始化参数的同结构网络，数据是新的子任务的数据。在多个子任务上学习之后，对于一个新的子任务，Meta-learner 就会根据这个子任务 D_{train} 中的数据，自动跟新适应这个子任务的 Learner 的参数，更新好之后，就可以在 D_{test} 上测试评估。



模型 Meta-Learner 的更新算法如下图所示，第三行表示综合考量 n 个子任务，7-12 行表示一个子任务的前向迭代过程，14-16 行表示使用当前子任务的信息更新 Meta-learner 的参数。

Algorithm 1 Train Meta-Learner

Input: Meta-training set $\mathscr{D}_{meta-train}$, Learner M with parameters θ, Meta-Learner R with parameters Θ.

$\Theta_0 \leftarrow$ random initialization

for $d = 1, n$ **do**

  $D_{train}, D_{test} \leftarrow$ random dataset from $\mathscr{D}_{meta-train}$

  $\theta_0 \leftarrow c_0$ ▷ Intialize learner parameters

  for $t = 1, T$ **do**

    $\mathbf{X}_t, \mathbf{Y}_t \leftarrow$ random batch from D_{train}

    $\mathcal{L}_t \leftarrow \mathcal{L}(M(\mathbf{X}_t; \theta_{t-1}), \mathbf{Y}_t)$ ▷ Get loss of learner on train batch

    $c_t \leftarrow R((\nabla_{\theta_{t-1}} \mathcal{L}_t, \mathcal{L}_t); \Theta_{d-1})$ ▷ Get output of meta-learner using Equation 2

    $\theta_t \leftarrow c_t$ ▷ Update learner parameters

  end for

  $\mathbf{X}, \mathbf{Y} \leftarrow D_{test}$

  $\mathcal{L}_{test} \leftarrow \mathcal{L}(M(\mathbf{X}; \theta_T), \mathbf{Y})$ ▷ Get loss of learner on test batch

  Update Θ_d using $\nabla_{\Theta_{d-1}} \mathcal{L}_{test}$ ▷ Update meta-learner parameters

end for

7.3 other tricks

parameter sharing and preprocessing: 为了降低复杂度，参数 θ 是一个向量，将其直接作为整体更新复杂度就很高，作者提出将该向量的每一个维度都使用相同的 meta-learner 更新，意即将前面的 ∇_{θ_t} 输入换为一组 $\nabla_{\theta_{t,i}}$，这样 θ 的每一个维度的更新共享相同的参数。preprocessing 如下图所示, x 为 LSTM 每一次迭代的输入，在这里即为每次 Leaner 的损失函数值和其梯度。

$$x \rightarrow \begin{cases} \left(\frac{\log(|x|)}{p}, \operatorname{sgn}(x)\right) & \text{if } |x| \geq e^{-p} \\ (-1, e^p x) & \text{otherwise} \end{cases}$$

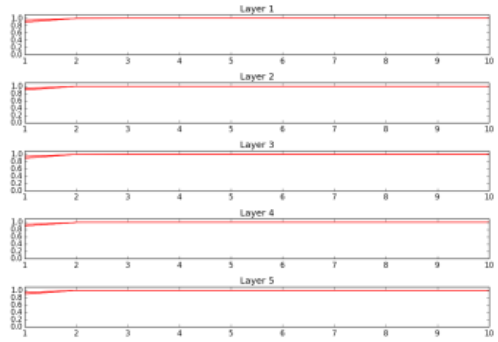
gradient independence assumption: 即为子任务框架图中虚线处理方法。此外还有 Batch normalization 以增强学习效果。

7.4 Experiments

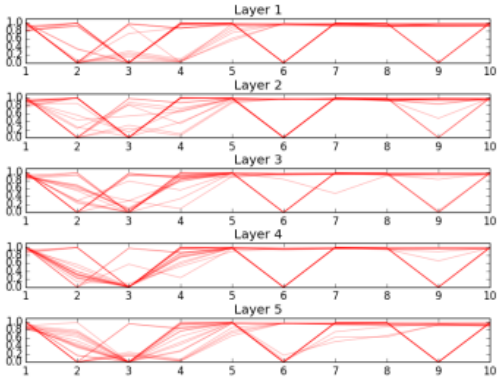
文中做实验分析了训练过程中，遗忘门和输入门到底学习了什么策略。如下图所示，每张图中横轴代表 LSTM 的迭代过程, 每条线代表不同子任务中的情况记录。

先看左边两幅图，遗忘门学习到一个非常统一的策略，越久的信息权重越小。这也是最常见的现象。

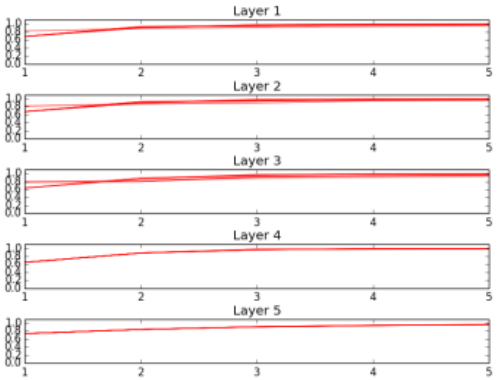
再看右边两幅图，输入门看不出学习到了什么策略，针对不同种类的任务，例如 1-shot learning 和 5-shot learning。输入策略整体相差很大。在同一个任务中，不同子任务的情况相差也很大，在 1-shot learning 的任务中尤为明显。这个说明对于不同的输入输入门的控制都非常敏感。



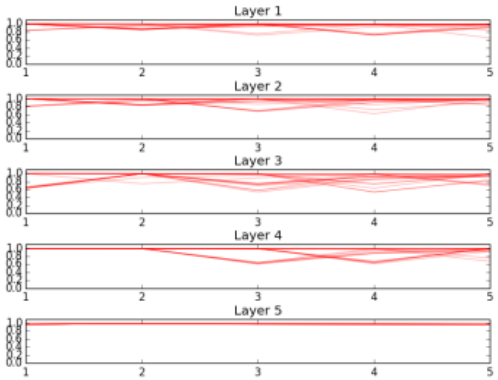
(a) Forget gate values for 1-shot meta-learner



(b) Input gate values for 1-shot meta-learner



(c) Forget gate values for 5-shot meta-learner



(d) Input gate values for 5-shot meta-learner

8 Bilevel Programming for Hyperparameter Optimization and Meta-Learning

这篇文章很多地方我都不太理解，就觉得不顺畅，介绍的就简略些。

8.1 A bilevel framework

首先作者提出了一个 bilevel 的优化框架，如下图所示：

In this paper, we consider bilevel optimization problems (see e.g. Colson et al., 2007) of the form

$$\min\{f(\lambda) : \lambda \in \Lambda\}, \tag{1}$$

where function $f : \Lambda \to \mathbb{R}$ is defined at $\lambda \in \Lambda$ as

$$f(\lambda) = \inf\{E(w_\lambda, \lambda) : w_\lambda \in \arg\min_{u \in \mathbb{R}^d} L_\lambda(u)\}. \tag{2}$$

We call $E : \mathbb{R}^d \times \Lambda \to \mathbb{R}$ the *outer objective* and, for every $\lambda \in \Lambda$, we call $L_\lambda : \mathbb{R}^d \to \mathbb{R}$ the *inner objective*. Note that

图中，可以看出这基本与前面描述的 Meta-Learning 框架差不多，inner objective 就相当于每个子任务的 loss，outer objective 就相当于 meta-learner 的 loss。

8.2 Bilevel framework for HO

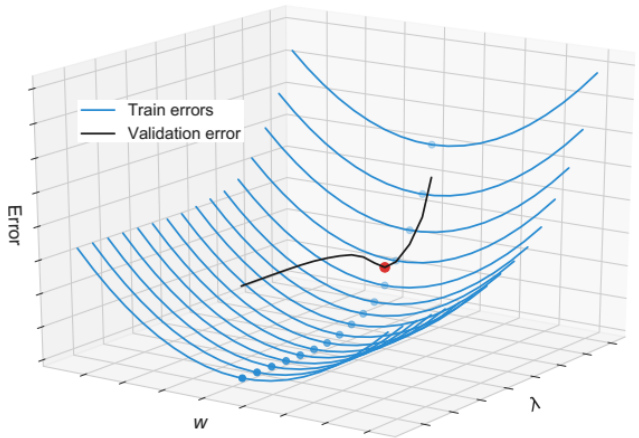
作者描述的超参优化方法，就分为两部分，第一部分是在训练集上训练模型，目标如下：

$$L_\lambda(w) = \sum_{(x,y)\in D_{\text{tr}}} \ell(g_w(x), y) + \Omega_\lambda(w),$$

其次是使用验证集选择一个 λ 表现较好的模型，目标如下：

$$E(w, \lambda) = \sum_{(x,y)\in D_{\text{val}}} \ell(g_w(x), y).$$

第二部分没有显示包含 λ，因为对于这个问题就是，第一步以多个不同 λ 训练多个子模型，然后在验证集上选择一个表现较好的模型。示意图如下：



8.3 Bilevel framework for Meta Learning

作者总结 meta-learning 是这样的：

$$L_\lambda(w) = \sum_{j=1}^{N} L^j(w^j, \lambda, D^j_{\text{tr}}),$$
$$E(w, \lambda) = \sum_{j=1}^{N} L^j(w^j, \lambda, D^j_{\text{val}})$$

作者总结了条件：meta-learner 的参数 λ 是在各个子任务中共享的，Meta-learner 的目标是建立在验证集上。

这个总结概括层次太高了，让人看不出 Leaner 和 Meta-learner 的区分。从设计上考虑 Learner 都有自己损失函数，前面的文章中，第 4 部分的 Learner 是非参的，第 7 部分的 Learner 的 loss 是模型内部的东西，并且也没有参数，其参数是细胞状态，由 meta-learner

调节。这两种情况 meta-learner 的 loss 是独立设计的，在所有的子任务上优化，也符合上图公式。第 5 部分的 meta-learner 的考量就是每个 learner loss 求和，这个和上图直接对应。虽然内部变化形式很多，但最终，meta-learner 的 loss 大概就如上图所示。

8.4 To Learn Model

作者将 inner objective 换成了一个动态系统。如下图所示，其中，$\Phi_t(.)$ 为参数迭代公式，例如可取：$\Phi_t(w_t, \lambda) = w_t - {}_t\nabla L_\lambda(.)$。整个更新过程就像一个交替更新的过程，在时间 t, 先用公式 6 计算新的参数 w_t，然后再用公式 5 更新参数 λ，到时间 T 的时候停止。

$$\min_{\lambda} f_T(\lambda) = E(w_{T,\lambda}, \lambda), \tag{5}$$

where E is a smooth scalar function, and[2]

$$w_{0,\lambda} = \Phi_0(\lambda),\ w_{t,\lambda} = \Phi_t(w_{t-1,\lambda}, \lambda),\ t \in [T], \tag{6}$$

然后作者论证满足下列条件，这个方法就可以找到准确的解。

(i) Λ is a compact subset of $\mathbb{R}^m$;

(ii) $E : \mathbb{R}^d \times \Lambda \to \mathbb{R}$ is jointly continuous;

(iii) the map $(w, \lambda) \mapsto L_\lambda(w)$ is jointly continuous and such that $\arg\min L_\lambda$ is a singleton for every $\lambda \in \Lambda$;

(iv) $w_\lambda = \arg\min L_\lambda$ remains bounded as λ varies in Λ.

(v) *$E(\cdot, \lambda)$ is uniformly Lipschitz continuous;*

(vi) *The iterates $(w_{T,\lambda})_{T\in\mathbb{N}}$ converge uniformly to w_λ on Λ as $T \to +\infty$.*

9 Pseudo-task Augmentation: From Deep Multitask Learning to Intratask Sharing—and Back

9.1 Joint training of models for multiple tasks

multi-task learning 一般是应用于多个相似的任务，这些任务有一些潜在的共享信息。Joint training of models for multiple tasks 很早就被人提出，后来被不断扩展，包括与深度学习相结合。在这个模型中，多个任务共享的一个底层模型（可视为 encoder 或者 embedding），记为 F，然后每个任务都有各自的 decoder 模型，记为 D_t。这样每个任务的预测值有如下公式计算得到：

$$\hat{y}_{ti} = D_t(F(x_{ti}; \theta_F); \theta_{D_t})$$

然后综合考量所有任务做出优化，优化方式如下，其中，T 表示任务数，N_t 表示每个任务中数据样本数。

$$\theta^* = \arg\min_\theta \frac{1}{T} = \sum_{t=1}^{T} \frac{1}{N_t} \sum_{i=1}^{N_t} L(y_{ti}, \hat{y}_{ti})$$

9.2 Pseudo-task Augmentation

本文的想法就是一个任务也要用多个 Decoder 来处理，如下是任务 t 中的 decoder d 对样本 i 的预测：

$$\hat{\boldsymbol{y}}_{tdi} = \mathcal{D}_{td}(\mathcal{F}(\boldsymbol{x}_{ti}; \theta_{\mathcal{F}}); \theta_{\mathcal{D}_{td}})$$

自然地，优化目标如下：

$$\theta^* = \underset{\theta}{\operatorname{argmin}} \frac{1}{TD} \sum_{t=1}^{T} \frac{1}{N_t} \sum_{i=1}^{N_t} \sum_{d=1}^{D} \mathcal{L}(\boldsymbol{y}_{ti}, \hat{\boldsymbol{y}}_{tdi})$$

由于有多个 decoder 处理同一个任务，文中将每个 decoder 处理这个任务地过程称为 Pseudo-task。普通的多任务学习可以称为 one-pseudo-task learning。

接下来作者从理论上论证 multi-pseudo-task learning 确实比 one-pseudo-task learning 的效果更好。

作者从两个方向证明，一是 one-pseudo-task learning 不能处理所有 multi-pseudo-task learning 能处理地问题，结论如下：

> **Definition 1** (Pseudo-task Simulation). *A set of pseudo-tasks S_1* simulates *another S_2 on $\mathcal{F}$ if for all $\theta_{\mathcal{F}}$ the gradient update to $\theta_{\mathcal{F}}$ when trained with S_1 is equal to that with S_2.*
>
> **Theorem 1** (Augmented Training Dynamics). *There exist differentiable functions $\mathcal{F}$ and sets of pseudo-tasks of a single task that cannot be simulated by a single pseudo-task of that task, even when all decoders are linear.*

其次就是 multi-pseudo-task learning 可以处理所有 one-pseudo-task learning 可以处理地任务。这两点直观上都是很自然的。但是 multi-pseudo-task learning 会增加模型复杂度，这也是要考虑的问题。

9.3 Control of Multiple Pseudo-task Trajectories

作者提出了六种方法来调节参数更新过程。实验的设计也有各种组合的相互比较。模型整体训练过程如下，除去 8-12 行就是普通的参数更新过程。在 8-12 行中，作者考虑了每个任务中，多个 decoder 该如何影响训练，包含在六种调节策略之中。12 行参数更新的默认策略是保持不变。

1. Greedy(G): 在第 10 行会计算每个 decoder 的 cost，在第 12 行更新 decoder 的参数的时候，一个贪心的策略就是将所有 decoder 都使用 cost 最小的 decoder 的参数。

2. Perturb(P): 在第 12 行，每个 decoder 的参数更新都要加上小小的噪声以避免多个 decoders 趋向于相同。

3. Hyperperturb(H): 在第 12 行，在每个 decoder 的超参数上加小小的噪声。文中，decoder 的超参是网络中 dropout 层的 dropout rate。

4. Independent Initialization (I): 在第二行初始化 decoder 参数的的时候，确保每个 decoder 参数初始化的独立性。

5. Freeze (F): 这个使之在第 7 行反向传播的时候只能通过同任务中的一个 decoder 传播，其它 decoder 不改变也不去影响底层的 F 模块。

6. Independent Dropout (D): 每个 decoder 的 dropout 层设计相互独立。这个和策略 P 一样可以保证每个 decoder 的不同性。

Algorithm 1 PTA Training Framework

1: Given T tasks $\{\{\boldsymbol{x}_{ti}, \boldsymbol{y}_{ti}\}_{i=1}^{N_t}\}_{t=1}^{T}$, and D decoders per task
2: $\{\{\theta_{\mathcal{D}_{td}}\}_{d=1}^{D}\}_{t=1}^{T} \leftarrow \textit{DecInitialize}()$
3: Initialize $\theta_{\mathcal{F}}$
4: Initialize decoder costs $c_{td} \leftarrow \infty \; \forall \; (t, d)$
5: **while** not done training **do**
6: **for** $m = 1$ **to** M **do** ▷ M is meta-iteration length
7: Update $\theta_{\mathcal{F}}$ and $\theta_{\mathcal{D}_{td}}$ via a joint gradient step.
8: **for** $t = 1$ **to** T **do**
9: **for** $d = 1$ **to** D **do**
10: $c_{td} \leftarrow \text{evaluate}(\theta_{\mathcal{D}_{td}}, \theta_{\mathcal{F}}, t)$ ▷ e.g., get validation error
11: **for** $d = 1$ **to** D **do**
12: $\theta_{\mathcal{D}_{td}} \leftarrow \textit{DecUpdate}\big(d, \{\theta_{\mathcal{D}_{td_o}}, c_{td_o}\}_{d_o=1}^{D}\big)$
13: **return** $\big(\{\{\theta_{\mathcal{D}_{td}}\}_{d=1}^{D}\}_{t=1}^{T}, \theta_{\mathcal{F}}\big)$

9.4 Experiments

除了性能对比之外，作者也设计实验探究各个策略对实验的影响，下图是图片分类的任务，D 表示每个任务 decoder 的数目。

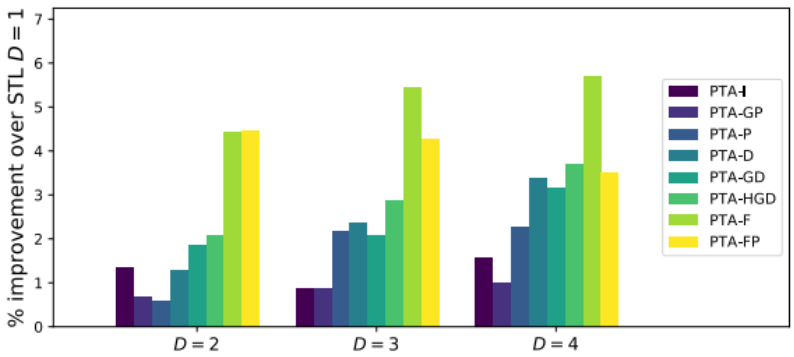


Figure 2. **Omniglot single-task learning results.** For each num-

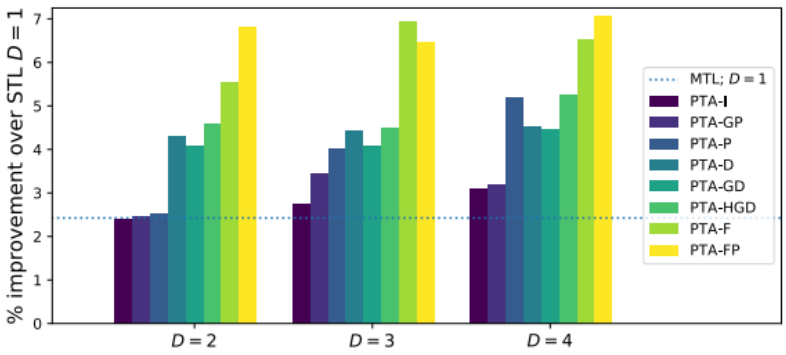


Figure 3. **Omniglot multitask learning results.** For each number

参考文献

[1] Ron Amit and Ron Meir. Meta-learning by adjusting priors based on extended pac-bayes theory. In *International Conference on Machine Learning*, pages 205–214, 2018.

[2] Marcin Andrychowicz, Misha Denil, Sergio Gomez, Matthew W Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, and Nando De Freitas. Learning to learn by gradient descent by gradient descent. In *Advances in Neural Information Processing Systems*, pages 3981–3989, 2016.

[3] Luca Franceschi, Paolo Frasconi, Saverio Salzo, and Massimilano Pontil. Bilevel programming for hyperparameter optimization and meta-learning. *arXiv preprint arXiv: 1806.04910*, 2018.

[4] Gregory Koch, Richard Zemel, and Ruslan Salakhutdinov. Siamese neural networks for one-shot image recognition. In *ICML Deep Learning Workshop*, volume 2, 2015.

[5] Ke Li and Jitendra Malik. Learning to optimize. *arXiv preprint arXiv:1606.01885*, 2016.

[6] Donald B Maudsley. A theory of meta-learning and principles of facilitation: An organismic perspective. 1980.

[7] Elliot Meyerson and Risto Miikkulainen. Pseudo-task augmentation: From deep multitask learning to intratask sharing—and back. *arXiv preprint arXiv:1803.04062*, 2018.

[8] Sachin Ravi and Hugo Larochelle. Optimization as a model for few-shot learning. 2016.

[9] Oriol Vinyals, Charles Blundell, Tim Lillicrap, Daan Wierstra, et al. Matching networks for one shot learning. In *Advances in Neural Information Processing Systems*, pages 3630–3638, 2016.

[10] WEI Ying, Yu Zhang, Junzhou Huang, and Qiang Yang. Transfer learning via learning to transfer. In *International Conference on Machine Learning*, pages 5072–5081, 2018.