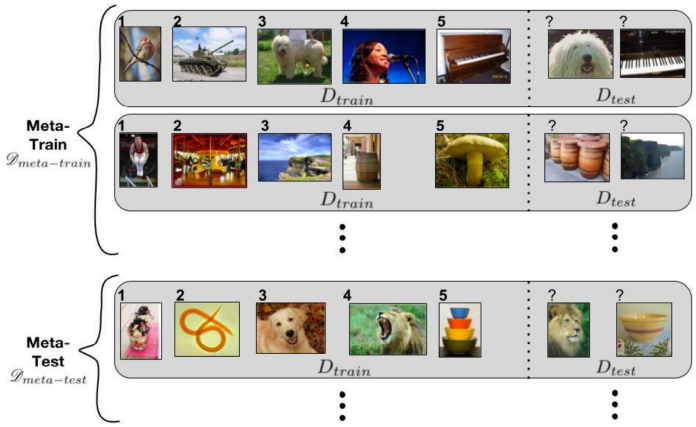
本文是一篇关于 few-shot learning 的 memo。Few-shot learning 属于 meta learning 的范畴，meta learning 通过观察一些机器学习方法在一些 task 上的表现，得到的经验（或称为 meta data）用于新的 task 上来得到更快或者更好的学习效果。而 few-shot learning 是指在监督学习中，对于只有少量训练样本的情况（比如学习一个分类器，但是训练集中每个类只有少量的样本），通过利用相似的一系列 task 上的经验来快速的得到新的 task 上的分类结果。

为了方便起见，我们将每个 task 上的分类器称为 learner，将保留所有 training task 的经验的模型称为 meta learner，对于 meta learner 来说每一个 task 作为一个样本，new task 作为测试样本。



第一类方法是通过非参数方法（KNN、Kmeans 等）作为 learner 进行分类，因此要为每一个样本得到一个低维的表示来计算距离，而将图片映射为低维向量的 embedding 网络就是 meta learner。对于新的 task，我们不需要进行训练就可以直接得到样本的分类。这种方法往往有两个点，一是如何设计 embedding 网络，二是使用什么距离。

1. Matching Networks for One Shot Learning

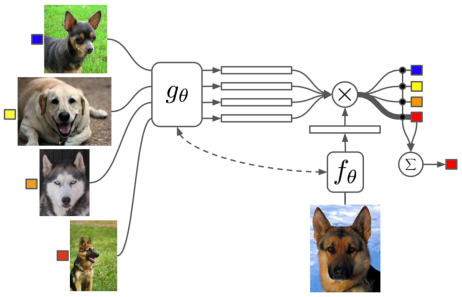


Figure 1: Matching Networks architecture

该方法基于 KNN 和 attention 机制。对于一个 task，网络会保留所有的训练样本，直接利用样本集 S 得到一个分类器：

$$P(\hat{y}|\hat{x}, S) = \sum_{i=1}^{k} a(\hat{x}, x_i) y_i$$

其中 a 是一个注意力核函数，对于所有 k 个样本求和为 1，因此可以看做是一个注意力权重，y'的是某个 y 的概率由 x'与类别为 y 的样本的相似程度决定，如果将权重特别小的样本都的权重都置为 0，则类似于找 x'的 k 近邻，然后利用

k 近邻的类别来决定测试样本的类别。具体来说，a 计算两个图片 embedding 后的 cos 距离：

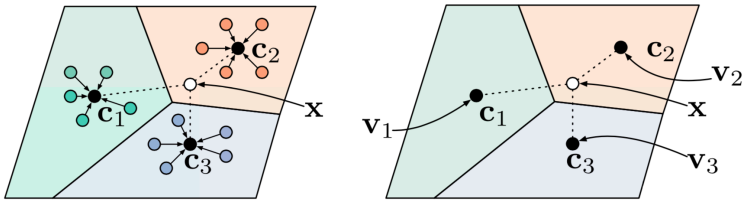
$$a(\hat{x}, x_i) = e^{c(f(\hat{x}),g(x_i))} / \sum_{j=1}^{k} e^{c(f(\hat{x}),g(x_j))}$$

本文提出，对训练集 S 进行 embedding 的模型 g，应该能看到整个 S 而不是仅仅利用样本 x，因此用了双向 lstm 来实现。对测试样本进行 embedding 的模型 f 应该可以根据训练样本进行调整，使用基于 attention 的 lstm 实现。最后两个模型的训练目标为在测试集上分类的最大似然损失：

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

2. Prototypical Networks for Few-shot Learning

原型网络的思路是将所有训练集中的点用一个网络 embedding 到一个相同的空间，然后找到某一类的中心点作为类的原型，当对新的样本进行预测时，只要计算样本到各个类的原型的距离，将样本分到最近的类别即可。



(a) Few-shot (b) Zero-shot

类别 k 的中心为：

$$\mathbf{c}_k = \frac{1}{|S_k|} \sum_{(\mathbf{x}_i, y_i)\in S_k} f_{\phi}(\mathbf{x}_i)$$

测试样本的分类：

$$p_{\phi}(y = k \,|\, \mathbf{x}) = \frac{\exp(-d(f_{\phi}(\mathbf{x}), \mathbf{c}_k))}{\sum_{k'} \exp(-d(f_{\phi}(\mathbf{x}), \mathbf{c}_{k'}))}$$

文章采用了 bregman 散度作为对距离的度量：

$$d_{\varphi}(\mathbf{z}, \mathbf{z}') = \varphi(\mathbf{z}) - \varphi(\mathbf{z}') - (\mathbf{z} - \mathbf{z}')^T \nabla\varphi(\mathbf{z}')$$

当上式中 φ 函数为平方 2 范数时，我们就得到了平方欧式距离||z-z'||。文章发现，如果使用平方欧氏距离，对某个样本 x 进行预测 P(y|x)实际上建模为了一个线性模型（关于变量的线性函数+指数族分布）：

$$-\|f_{\phi}(\mathbf{x}) - \mathbf{c}_k\|^2 = -f_{\phi}(\mathbf{x})^{\top} f_{\phi}(\mathbf{x}) + 2\mathbf{c}_k^{\top} f_{\phi}(\mathbf{x}) - \mathbf{c}_k^{\top}\mathbf{c}_k$$

如果 embedding 网络是固定的，我们把 f(x)看做一个变量。上式中第一项看做常数，不影响第 k 类的概率。则我们得到了如下线性函数：

$$2\mathbf{c}_k^{\top} f_{\phi}(\mathbf{x}) - \mathbf{c}_k^{\top}\mathbf{c}_k = \mathbf{w}_k^{\top} f_{\phi}(\mathbf{x}) + b_k, \text{where } \mathbf{w}_k = 2\mathbf{c}_k \text{ and } b_k = -\mathbf{c}_k^{\top}\mathbf{c}_k$$

作者将该距离表现好的原因归结为在 embedding 函数中已经学到了足够的非线性，而对测试样本 x 只需要一个线性模型就可以实现好的预测。

最后本方法还可以方便的拓展到 zero-shot，如右图，embedding 从不同的 domain 中学到。Prototypical Networks 和 match network 实际上是很类似的，当

做 one-shot 的时候两种结构等价。

第二类方法是基于优化的的 meta learning 方法，

3. OPTIMIZATION AS A MODEL FOR FEW-SHOT LEARNING

我们常用的基于梯度的优化方法，包括 GD，adagrad，adam 等在设计之初就不适合少量训练样本，几步迭代的情况，这些方法往往是通过大量样本构建 loss，经过很多步迭代后收敛到一个较优的点，而且为了适应不同的数据，一般都是随机选择初始点的。

本文提出了用一个模型（meta learner）来学习更新规则，具体来说：在每个 task 上用一个 learner 来学习，learner 的参数更新由一个通用的 meta learner 来引导。文章用 lstm 模型来描述 meta learner，将梯度下降过程：

$$\theta_t = \theta_{t-1} - \alpha_t \nabla_{\theta_{t-1}} \mathcal{L}_t$$

类比 lstm 更新过程：

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

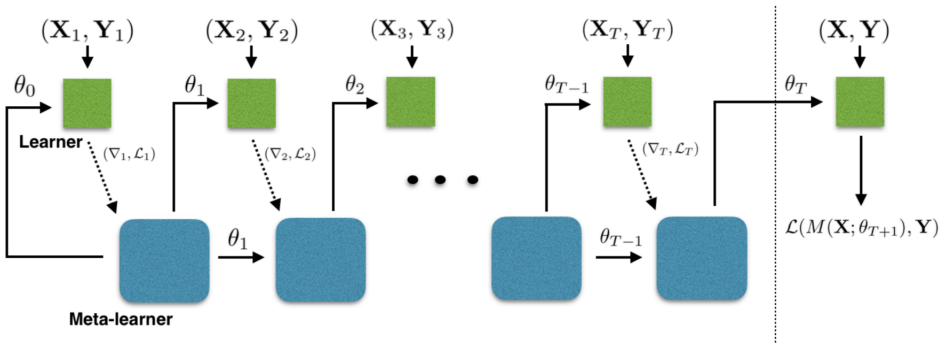
$$\text{if } f_t = 1, c_{t-1} = \theta_{t-1}, i_t = \alpha_t, \text{ and } \tilde{c}_t = -\nabla_{\theta_{t-1}} \mathcal{L}_t$$

实际上只是将梯度下降的步长 α 以及上一步的参数 θ 前的系数 1 也设置成某个模型的输出，比起像 adam 等我们设置步长自适应（对于大梯度给小步长）等人工规则，这里是用一个模型来在可能的规则“搜索空间”中来学习，系数 1 进行调整的目的是为了跳出局部最小值。模型的输入由上一步 learner 参数的值，对应的 loss，参数的梯度等等：

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

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

有的参数的维度可能很高，为了不用单独为每一个参数学习一个 meta learner（比如对于 10 维的参数，W 应该为 10*output_dim，对于 100 维，要定义新的 W 100*output_dim），参数的不同维度都是共享 meta learner 的参数的（10 维的参数，当做是 batch_size 为 10 的数据，W 可以是 1*output_dim）。



训练过程如下，learner 以某个 task 上训练集的误差作为 loss，通过 meta learner 更新参数，训练完成后，在测试集上的损失作为 meta learner 的 loss：

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

4. Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks

在上一篇文章中，参数的起始点也被作为一个 meta learner 参数进行训练。同样强调起始点的重要性的还有这一篇 MAML。我们很常见的一种情况是，我们有时会在大数据集上进行训练，训练好的结构对于新的数据集只要进行微调就能够达到很好的效果。MAML 就是要找到这样的起始点 θ，使得对于 training task，经过少量的迭代后得到 θ‘，可以达到不错的分类效果。

$$\theta_i' = \theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta).$$

meta learner 的参数就是 θ，该参数的 loss 为：

$$\min_\theta \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta_i'}) = \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)})$$

其中 τ 是某个 task，θ 的更新过程如下，注意到由于涉及到对梯度求梯度，我们实际上是要算出 loss 过于参数的海森矩阵：

$$\theta \leftarrow \theta - \beta \nabla_\theta \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta_i'})$$

算法过程如下：

Algorithm 1 Model-Agnostic Meta-Learning

Require: $p(\mathcal{T})$: distribution over tasks
Require: α, β: step size hyperparameters
randomly initialize θ
while not done **do**
  Sample batch of tasks $\mathcal{T}_i \sim p(\mathcal{T})$
  for all $\mathcal{T}_i$ **do**
    Evaluate $\nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$ with respect to K examples
    Compute adapted parameters with gradient descent: $\theta_i' = \theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$
  end for
  Update $\theta \leftarrow \theta - \beta \nabla_\theta \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta_i'})$
end while

具体来说，在有监督问题中，算法流程如下：

Algorithm 2 MAML for Few-Shot Supervised Learning

Require: $p(\mathcal{T})$: distribution over tasks
Require: α, β: step size hyperparameters
1: randomly initialize θ
2: **while** not done **do**
3: Sample batch of tasks $\mathcal{T}_i \sim p(\mathcal{T})$
4: **for all** $\mathcal{T}_i$ **do**
5: Sample K datapoints $\mathcal{D} = \{\mathbf{x}^{(j)}, \mathbf{y}^{(j)}\}$ from $\mathcal{T}_i$
6: Evaluate $\nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$ using $\mathcal{D}$ and $\mathcal{L}_{\mathcal{T}_i}$ in Equation (2) or (3)
7: Compute adapted parameters with gradient descent: $\theta_i' = \theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$
8: Sample datapoints $\mathcal{D}_i' = \{\mathbf{x}^{(j)}, \mathbf{y}^{(j)}\}$ from $\mathcal{T}_i$ for the meta-update
9: **end for**
10: Update $\theta \leftarrow \theta - \beta \nabla_\theta \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta_i'})$ using each $\mathcal{D}_i'$ and $\mathcal{L}_{\mathcal{T}_i}$ in Equation 2 or 3
11: **end while**

注意到，对于一个 task，是独立采样了训练集 D 和测试集 D’，在训练集上学习 learner 的参数 θ’，θ’是关于 θ 的函数，之后再测试集上，我们更新 θ 使得各个 task 上测试集误差最小。