# 第 13 章 小样本学习

## 13.1 研究背景及意义

### 13.1.1 机器算法性能依赖于大量数据导致的问题

机器算法已经取得了令人瞩目的成就，并在各行各业成功应用。然而，机器算法依赖于大规模的标注样本，从而对模型含有的参数进行充分的学习。例如，在图像分类任务中，想要正确识别某类物体，往往需要对成千上万幅人工标注的样本上进行学习，才能达到令人满意的结果。然而，对于在某些实际问题或者应用场景中，获得如此多的标注样本是十分困难，甚至是不现实的。原因主要包括以下几点：1）难以接受的标注成本，在某些领域中，人们即使能够获得大量的样本，对样本的标注也需要大量的人力物力财力，甚至需要特殊的专业人员进行指导。譬如，对于 SAR 目标识别任务来说，由于 SAR 图像成像机制的限制，只有经过训练的专业人员才能较为准确的在图像中辨别哪些是需要关注的目标；2）某些领域可用样本数量少，譬如在教育、医疗、交管等公共部门中有大量的优质数据，但是这些数据开放程度低，一般的研究人员难以获得并加以有效利用；3）某些领域本身就是小样本问题，譬如生物化学领域对于新发现的分子结构的预测，此种分子结构是唯一的，不存在其他相同类型的样本，或者对于某些新研制出的军事装备，每种类型的存在样本数也是十分稀少的，这种场景对于目前的小样本学习方法是非常有挑战性的。因此，近年来研究者们越来越关注降低对大量标注数据的依赖的方法，即小样本学习方法。

### 13.1.2 小样本学习意义

（1）降低人工智能应用的使用成本。小样本问题的设定不依赖于大规模的标注训练样本，从而避免了在某些特定应用中数据准备的高昂成本。小样本学习的目的是在已经成熟的标注充分的数据集上，训练模型的小样本分类能力。当遇到新的场景，新的任务时，可以在有限的标注样本基础上达到可观的结果。

（2）缩小人类智能与人工智能之间的差距，减少人工干预。 以深度学习的交通标识识别为例。在获取交通标识数据集的时候，由于某些交通标识不太常见，例如连续急转弯的标识主要出现在偏远山区，数据获取困难，样本量较少，导致深度学习训练模型时产生过拟合，模型预测效果较差。与此相反，人类只需要通过少量数据就能做到快速学习。一个五六岁的小孩子从未了解过交通标识的具体

定义，但给他看过一张连续急转弯的交通标识图片，当他在马路上看到对应标识的时候，就会马上认出这还是自己曾经学习过的连续转弯的交通标识。这种人类与生俱来的小样本识别能力正是如今的机器所缺乏的。从人类的认知世界的角度出发，我们更加希望机器也拥有这种举一反三的能力，也就是说在机器已经学会分不同种类的猫之后，我们希望他能够触类旁通，在相似的任务如分不同种类的狗上的表现要比从头开始的算法效果更好，这也是通向通用人工智能的一个重要研究方向。

（3）可以为一个新兴的任务实现低成本、快速的模型部署，对于任务早期的潜在规律的揭示是有益的。当遇到新的任务时，训练样本非常少时，可以将小样本学习的知识迁移到新的任务上，达到一个相对理想的结果。然后将模型部署到应用场景下，通过应用中的反馈，搜集新的样本，进一步提高模型的准确率。

小样本学习就是期望机器学习能够更加接近人类思维，主要研究数据集中样本种类多、每类样本数量少的情况下的相关问题。使用远小于深度学习所需要的数据样本量，达到接近甚至超越大数据深度学习的效果。为了达到这一目标，目前的主流小样本学习过程中利用了样本本身之外的一些信息，例如样本的属性和先验知识，或者其他相似任务下的样本信息，从而获得类似举一反三与触类旁通的能力。

### 13.1.3 小样本问题定义

小样本学习可以看做为 N-way、K-shot 问题，即训练集包含 N 类样本，每类样本包含 K 个带标签训练数据。图 13.1 展示了相比于样本充足的情况（图（a）），小样本条件下学到的最优近似解$h_I$与实际数据可能达到的最优解$h^*$相差更多，这使得小样本条件下模型更容易过拟合。

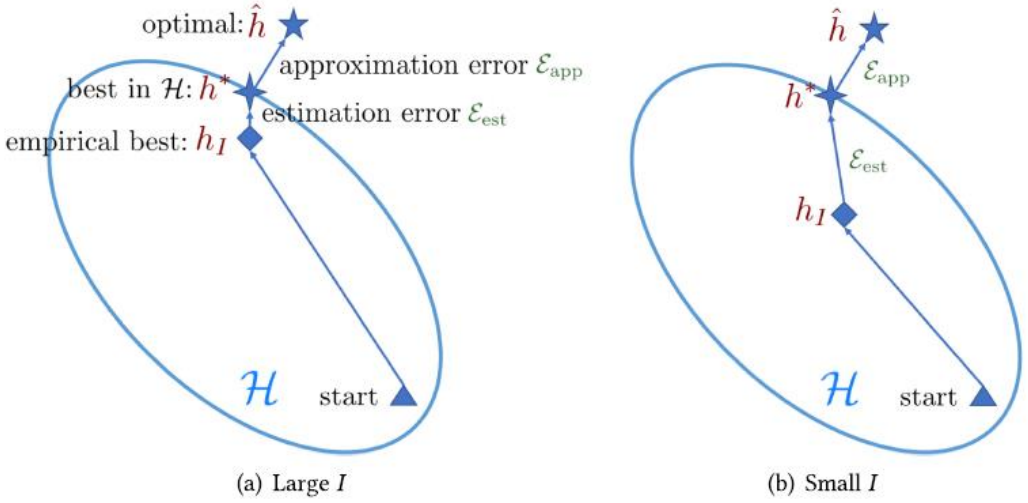


图 13.1 参数优化过程对比图

## 13.2 基于数据增强的小样本学习算法

本节中的小样本学习方法使用先验知识来扩充训练数据，从而丰富数据中的监督信息。在小样本学习方法中，通过人工制定的规则进行的数据扩充通常用作预处理。例如，在图像上，可以使用平移、翻转、剪切、缩放、反射、裁剪和旋转。然而，扩充规则可能特定于数据集，这使得它们很难应用于其他数据集。此外，人类不太可能列举所有可能的不变性。因此，手动数据扩充无法完全解决小样本学习问题。因此，研究者们研究了一系列的策略来进行数据增强。

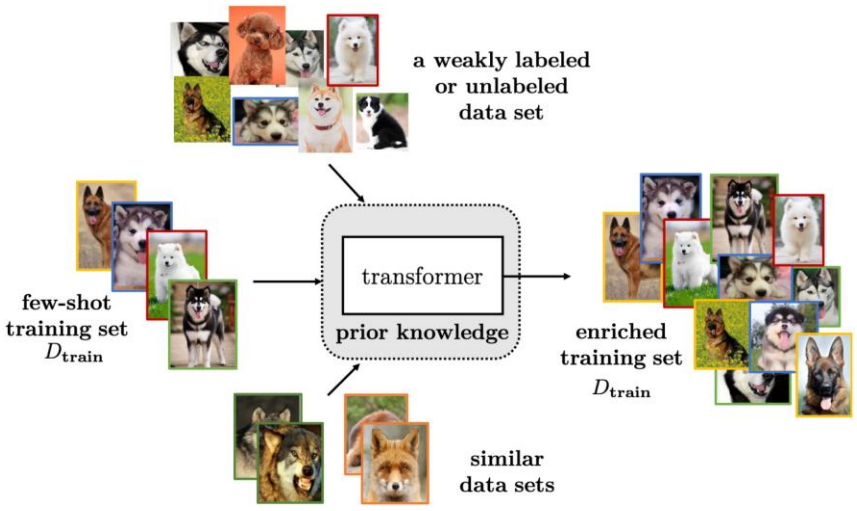


**图 13.2 数据增强示意图**

### 13.2.1 学习转化函数生成新样本

此种方法利用先验信息生成附加样本。早期的相关方法通过迭代地将每个样

本与其他样本对齐，从类似的类中学习一组几何变换。将学习到的几何变换应用于每个训练样本以形成一个大型数据集，然后可以通过标准机器学习方法学习这些数据集。近年赖，不是枚举对中的变量，一些方法使用从大量场景图像中学习的一组独立属性强度回归器将每个训练样本变换为多个样本，并将原始训练样本的标签指定给这些新样本。

### 13.2.2 基于弱标记或无标记数据集获得样本

该策略通过从弱标记或未标记的大型数据集中选择具有目标标签的样本来扩充训练样本。例如，在监控摄像机拍摄的照片中，有人、有车、有路，但都没有贴标签。另一个例子是长时间演示的视频。这包含演讲者的一系列手势，但没有一个是明确注释的。此外，收集这样的数据集更容易，因为标记不需要人为的努力。然而，尽管采集成本很低，但一个主要问题是如何选择带有目标标签的样本，以增加训练样本。有研究者对样本中的每个目标标签学习一个样本支持向量机，然后用于预测弱标签数据集中样本的标签。然后将具有目标标签的样本添加到样本中。另外一种方法不是学习分类器，而是直接使用标签传播来标记未标记的数据集，或者使用渐进策略选择信息性未标记样本。然后为选定的样本指定伪标签，并用于更新模型参数。

### 13.2.3 基于相似数据集获得样本

该策略通过聚合和调整来自相似但较大数据集的输入输出对来增强训练样本。聚合权重通常基于样本之间的某种相似性度量。由于这些样本可能不来自目标类，直接将聚合样本增加到训练集中可能会产生误导。因此，生成性对抗网络（GAN）被设计用于从多个样本的数据集中生成不可区分的合成样本。它有两个生成器，一个将小规模类的样本映射到大规模类，另一个将大规模类的样本映射到小规模类（以补偿 GAN 训练中样本的不足）。

GAN 模型由两部分组成：生成器（generator）和判别器（discriminator）。这里我们认为它们都是由参数确定的神经网络：G 和 D。判别网络的参数为最大化正确区分真实数据和伪造数据（生成网络伪造的数据）的概率这一目标而优化，而生成网络的目标是最大化判别网络不能识别其伪造的样本的概率。

生成网络生成样本：接受一个输入向量 z，该向量取样自一个潜分布，应用

由网络定义的函数 G 至该向量，得到 G(z)。判别网络交替接受 G(z)和 x（一个真实数据样本），输出输入为真的概率。通过适当的超参数调优和足够的训练迭代次数，生成网络和判别网络将一起收敛（通过梯度下降方法进行参数更新）至描述伪造数据的分布和取样真实数据的分布相一致的点。

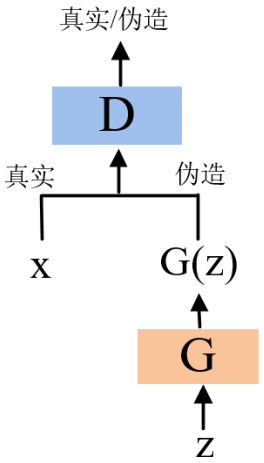


图 13.4 GAN 模型示意图

样本数据服从概率分布：$x{\sim}P_{data}(x)$，对于以$x$为输入的分布$P_G(x;\theta)$，通过学习参数$\theta$，使得$P_G(x;\theta)$接近$P_{data}(x)$，就可以确定生成器。其中参数$\theta$的确定需用利用极大似然估计。

我们从$x{\sim}P_{data}(x)$随机采集一组样本$\{x^1,x^2,\dots,x^m\}$，得到似然函数：$\mathrm{L}(\theta)=\prod_{i=1}^{m}P_G(x^i;\theta)$，使得$\mathrm{L}(\theta)$取得最大值的$\theta^*$就是我们需要的值：

$$\theta^* = argmax_\theta \prod_{i=1}^{m} P_G(x^i;\theta)$$

$$= argmax_\theta log \prod_{i=1}^{m} P_G(x^i;\theta)$$

$$= argmax_\theta \prod_{i=1}^{m} logP_G(x^i;\theta)$$

由于$x{\sim}P_{data}(x)$，所以：

$$\theta^* = argmax_\theta \prod_{i=1}^{m} logP_G(x^i;\theta)$$

$$= argmax_\theta E_{x\sim P_{data}(x)}[logP_G(x;\theta)]$$

$$= argmax_\theta \int_x P_{data}(x)\, logP_G(x;\theta)dx - \int_x P_{data}(x)\, logP_{data}(x)dx$$

$$= argmin_\theta KL(P_{data}(x)||P_G(x;\theta))$$

我们需要找的$\theta^*$就是使得$KL(P_{data}(x)||P_G(x;\theta))$取得最小值的$\theta$参数。

通常情况下$P_G(x;\theta)$是一个神经网络，对于一个随机向量z，通过$\mathrm{G}(\mathrm{z}) = x$可以生成样本$x$。那么就可以通过一组z，生成一个分布$P_G$，从而与真实分布$P_{data}$进行比较。由于神经网络可以拟合任意函数，那么也可以拟合任意分布，所以可以用正态分布，取样去训练一个神经网络，学习得到一个复杂的分布。这个过程可以看作：$P_G(x) = \int_z P_{rrior}(z)I_{G(z)=x}dz$，其中$I_{G(z)=x}$表示神经网络输入$z$时，输出恰好为$x$。

对于上述过程，如果用最大似然估计就会存在问题，因为神经网络的参数量太大，通过计算最大似然对神经网络的参数进行估计是不现实的，所以就需要使用生成对抗网络 GAN，用神经网络更新参数代替上述计算过程：用 Generator 代替$P_G(x)$，用 Discriminator 代替$P_{data}(x)$去约束 Generator。

生成器 Generator 输入为$z\sim P_z(z)$，输出为$\mathrm{G}(\mathrm{z})$。判别器 Discriminator 是一个全连接分类网络，输入为x，输出为相应的样本标签。定义其损失函数：

$$\mathrm{V}(\mathrm{D},\mathrm{G}) = E_{x\sim P_{data}(x)}[\log D(x)] + E_{z\sim P_z(z)}[1-\mathrm{D}(\mathrm{G}(\mathrm{z}))]$$

所以期望的 Generator：$G^* = min_G max_D V(D,G)$。

整个训练过程主要是交替执行下述两个步骤：（1）固定G中所有参数，用梯度下降法修正D。（2）固定D中所有参数，用梯度下降法修正G。以二分类模型为例，判断输入的图片是 Real 还是 Fake。假定判别器D输入$x_i$，对应标签为$y_i$，则 N 个样本的交叉熵为：

$$\mathrm{Sum}(CE) = -\sum_{i=1}^{N} y_i logD(x_i) - \sum_{i=1}^{N}(1-y_i)\log(1-\mathrm{D}(x_i))$$

考虑有$y_i$概率的样本$x_i\sim P_{data}(x)$，$1-y_i$概率的样本$x_i = G(Z_i)$，其中$z\sim P_z(z)$，那么：

$$\text{Sum(CE)} = -\sum_{i=1}^{N} y_i log D(x_i) - \sum_{i=1}^{N} P_i(z_i) \log\left(1 - \text{D}\left(G(Z_i)\right)\right)$$

$$= -E_{x\sim P_{data}(x)}[\log D(x)] - E_{z\sim P_z(z)}\left[1 - \text{D}\left(\text{G(z)}\right)\right] = -\text{V(D, G)}$$

所以：$G^* = min_G max_D V(D,G) = max_G min_D Sum(CE)$。在$min \leftrightarrow max$的博弈中GAN 达到平衡，从而通过 GAN 模型获得增强数据样本。

### 13.2.4 代码实例

本章基于 DCGAN 网络实代码进行简单讲解，代码地址：https://github.com/eriklindernoren/PyTorch-GAN。

1、生成器和判别器的定义及构建

1）定义

```
class Generator(nn.Module):
    def __init__(self):
        super(Generator, self).__init__()

        self.init_size = opt.img_size // 4
        self.l1 = nn.Sequential(nn.Linear(opt.latent_dim, 128 * self.init_size ** 2

        self.conv_blocks = nn.Sequential(
            nn.BatchNorm2d(128),
            nn.Upsample(scale_factor=2),
            nn.Conv2d(128, 128, 3, stride=1, padding=1),
            nn.BatchNorm2d(128, 0.8),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Upsample(scale_factor=2),
            nn.Conv2d(128, 64, 3, stride=1, padding=1),
            nn.BatchNorm2d(64, 0.8),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Conv2d(64, opt.channels, 3, stride=1, padding=1),
            nn.Tanh(),
        )

    def forward(self, z):
        out = self.l1(z)
        out = out.view(out.shape[0], 128, self.init_size, self.init_size)
        img = self.conv_blocks(out)
        return img
```

**图 13.5 DCGAN 网络生成器的定义**

```python
class Discriminator(nn.Module):
    def __init__(self):
        super(Discriminator, self).__init__()

        def discriminator_block(in_filters, out_filters, bn=True):
            block = [nn.Conv2d(in_filters, out_filters, 3, 2, 1),
                     nn.LeakyReLU(0.2, inplace=True), nn.Dropout2d(0.25)]
            if bn:
                block.append(nn.BatchNorm2d(out_filters, 0.8))
            return block

        self.model = nn.Sequential(
            *discriminator_block(opt.channels, 16, bn=False),
            *discriminator_block(16, 32),
            *discriminator_block(32, 64),
            *discriminator_block(64, 128),
        )

        # The height and width of downsampled image
        ds_size = opt.img_size // 2 ** 4
        self.adv_layer = nn.Sequential(nn.Linear(128 * ds_size ** 2, 1), nn.Sigmoid())

    def forward(self, img):
        out = self.model(img)
        out = out.view(out.shape[0], -1)
        validity = self.adv_layer(out)

        return validity
```

**图 13.6 DCGAN 网络判别器的定义**

2）生成器与判别器的构建

```python
# Initialize generator and discriminator
generator = Generator()


discriminator = Discriminator()
```

**图 13.6 DCGAN 网络生成器与判别器的构建**

2、网络训练

网络训练主要分为三步：1）基于随机噪声和生成器输出生成图像；2）计算生成损失及梯度并更新生成器参数；3）计算生成损失及梯度并更新判别器器参数。具体如下图所示。

```
# Generate a batch of images
gen_imgs = generator(z)

# Loss measures generator's ability to fool the discriminator
g_loss = adversarial_loss(discriminator(gen_imgs), valid)

g_loss.backward()
optimizer_G.step()

# ---------------------
#  Train Discriminator
# ---------------------

optimizer_D.zero_grad()

# Measure discriminator's ability to classify real from generated samples
real_loss = adversarial_loss(discriminator(real_imgs), valid)
fake_loss = adversarial_loss(discriminator(gen_imgs.detach()), fake)
d_loss = (real_loss + fake_loss) / 2

d_loss.backward()
optimizer_D.step()
```

**图 13.7 DCGAN 网络的训练**

### 13.2.5 分析与总结

选择使用哪种增强策略取决于应用程序。有时，目标任务（或类）存在大量弱监督或未标记的样本，但由于收集注释数据和/或计算成本高，因此很少使用。在这种情况下，可以通过转换弱标记或未标记数据集中的样本来执行扩充。当一个大规模的未标记数据集很难收集，但少数快照类有一些相似的类时，可以从这些相似的类中转换样本。如果只有一些已学习的变压器而不是原始样本可用，则可以通过转换来自训练样本的原始样本来进行扩充。

通常，通过增加训练样本来解决小样本学习问题是简单易懂的。通过利用目标任务的先验信息来扩充数据。然而，通过数据扩充解决小样本学习问题的缺点是，扩充策略通常是以特定方式为每个数据集量身定制的，并且不能轻松地用于其他数据集（尤其是来自其他域的数据集）。

## 13.3 基于模型的小样本算法

本节中的小样本学习方法通过先验知识将解空间约束到更小的空间来进行学习。这降低了损失函数学习的难度，过度拟合的风险也降低了。

### 13.3.1 多任务学习

在存在多个相关任务的情况下，多任务学习通过利用任务通用信息和任务特

定信息来同时学习这些任务。因此，它们可以自然地用于小样本问题。

给定 C 个相关任务$T_1, T_1 \ldots T_c$，其中有的任务属于小样本情况，有的任务含有大量样本。我们将小样本情况对应的任务看做目标任务（target tasks），其余的看做为源任务（source tasks）。由于所有的任务是同时进行学习的，对于第 c 个任务参数的更新就受限于其他任务的学习，从而缩小了目标任务的解空间。

根据任务参数的约束方式，我们将该策略中的方法分为参数共享和参数绑定。

（1）参数共享

如图 13.8 所示，此策略在任务之间直接共享一些参数。自然而然的，出现多种不同的参数共享策略，譬如有方法共享网络通用信息的前几个层，并学习不同的最终层来处理不同的输出，或者从源任务预先训练变分自动编码器，然后克隆到目标任务。两个可变自动编码器中的某些层共享以捕获通用信息，同时允许两个任务都有一些特定于任务的层。目标任务只能更新其特定于任务的层，而源任务可以同时更新共享层和特定于任务的层

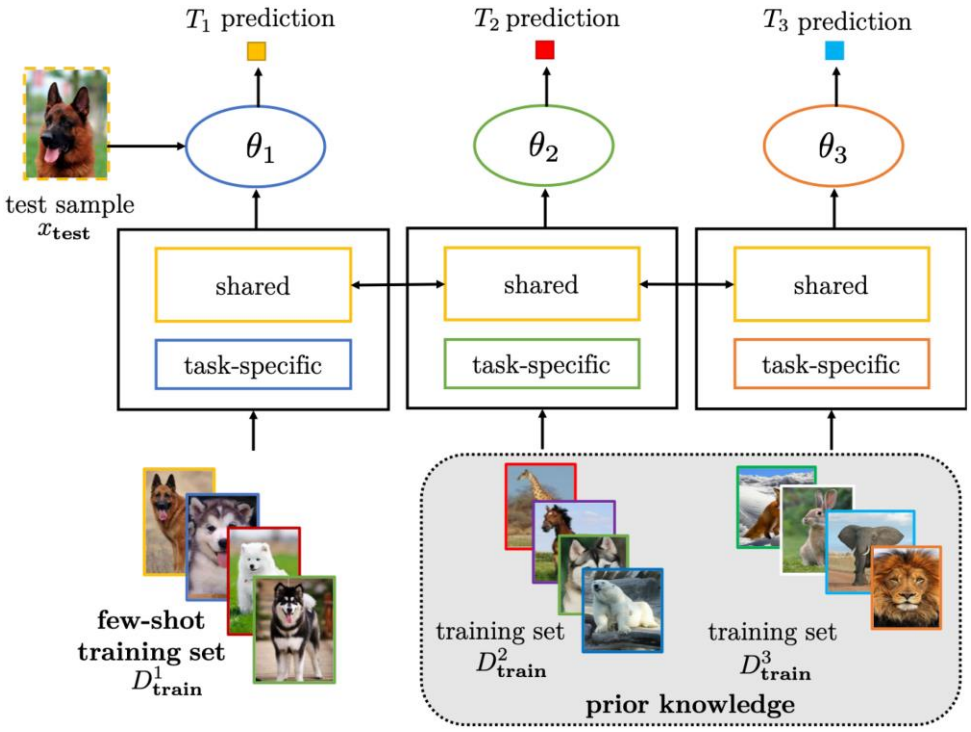


**图 13.8 参数共享小样本学习图示**

（2）参数绑定

如图 13.9 所示，这一策略期望不同任务的参数尽可能的相似。譬如，有一个卷积神经网络负责源任务的学习，另一个卷积神经网络负责目标任务的学习。这

两个卷积神经网络的特征层使用一些专门设计的正则化项对齐，从而使得参数尽可能的相似。

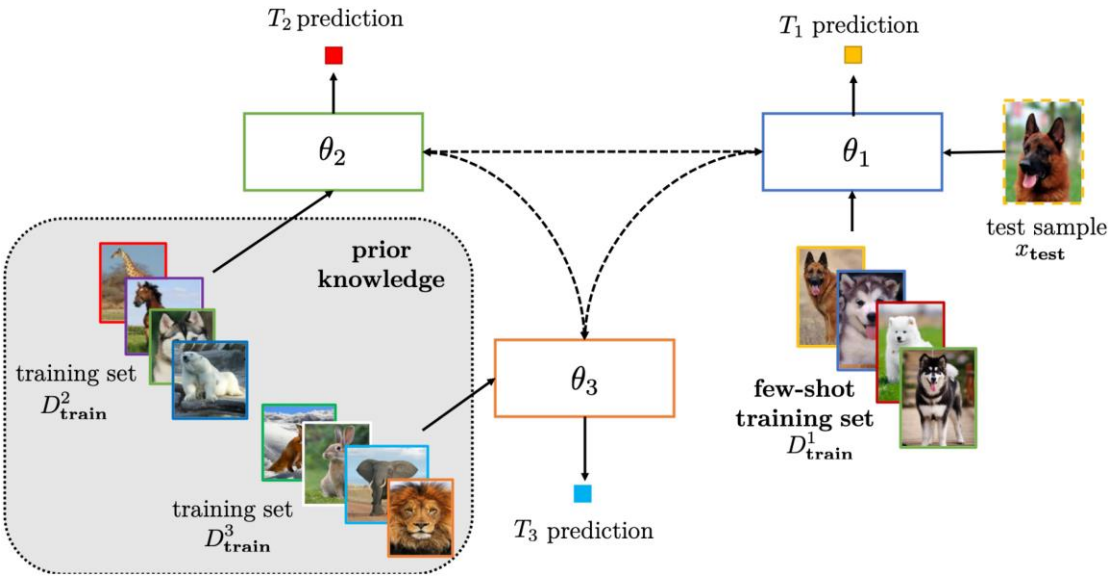


**图 13.9 参数绑定小样本学习图示**

### 13.3.2 嵌入学习

嵌入（Embedding）的本质是“压缩”，用较低维度的 k 维特征去描述有冗余信息的较高维度的 n 维特征，也可以叫用较低维度的 k 维空间去描述较高维度的 n 维空间。在思想上，与线性代数的主成分分析 PCA，奇异值分解 SVD 异曲同工，事实上，PCA 和 SVD 也可以叫做嵌入方法。

嵌入学习将每个样本$x_i \in X \subseteq R^d$嵌入到低维$z_i \in Z \subseteq R^m$，这样相似的样本会紧密靠近，而异类的样本则更容易区分。然后，在这个较低维的Z中，可以构造一个较小的假设空间H，随后需要较少的训练样本。嵌入功能主要是从先验知识中学到的，并且可以额外使用训练样本任务特定信息。

嵌入学习具有以下关键组成部分：

（1）将训练样本$x_{train} \in D_{train}$嵌入Z的函数$\mathrm{f}(x_{train})$；

（2）将测试样本$x_{test} \in D_{test}$嵌入Z的函数$\mathrm{g}(x_{test})$

（3）相似性函数 s 来测量$\mathrm{f}(x_{train})$和Z中的$\mathrm{g}(x_{test})$之间的相似度。

根据该类的嵌入$\mathrm{f}(x_{train})$与Z中的$\mathrm{g}(x_{test})$最相似这个原则，将测试样本$x_{test}$分配给$x_i$类。尽管可以为$x_i$和$x_{test}$使用通用的嵌入函数，但是使用两个单独的嵌入函数可以获得更好的准确性。根据嵌入函数 f 和 g 的参数是否随任务而变化，

我们将这些 FSL 方法分类为：特定于任务的嵌入模型；任务不变(即一般)嵌入模型；混合嵌入模型，可同时编码特定于任务和不变任务的信息。

（1）特定任务嵌入模型

通过仅使用来自该任务的信息为每个任务量身定制嵌入函数。例如：给定任务$Task_c$的少量数据$D^c_{train}$，$D^c_{train}$中样本之间的所有成对排名都列举为样本对，训练样本的数量因此增加，仅使用特定任务的信息也可以学习嵌入函数。

（2）任务不变嵌入模型

任务不变的嵌入方法从包含足够样本的大规模数据集中学习一个通用的嵌入函数，无需重新训练将其直接用于新的小样本$D_{train}$。第一个 FSL 嵌入模型使用内核嵌入样本。

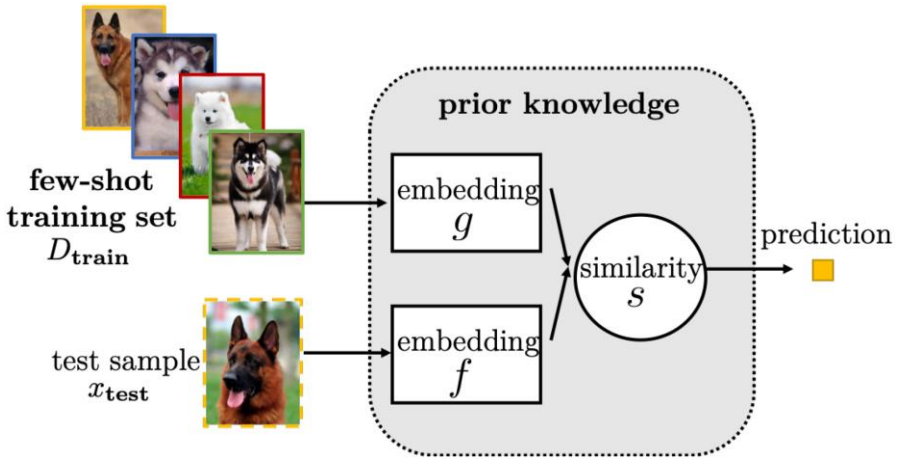


**图 13.10 任务不变嵌入模型**

（3）混合嵌入模型

尽管任务不变嵌入方法可以应用在新任务上且计算成本低,但不利用当前任务的特定知识。当任务特性是$D_{train}$只有少量样本的情况下，简单地应用任务不变的嵌入函数可能是不合适的。为了缓解这一问题，混合嵌入模型采用了一般的任务不变嵌入模型，该模型是根据数据训练中的特定任务信息从先验知识中学习而来的。这是通过学习一个函数来实现的，该函数将从$D_{train}$中提取的信息作为输入并返回一个嵌入，该嵌入作为 f 的参数。

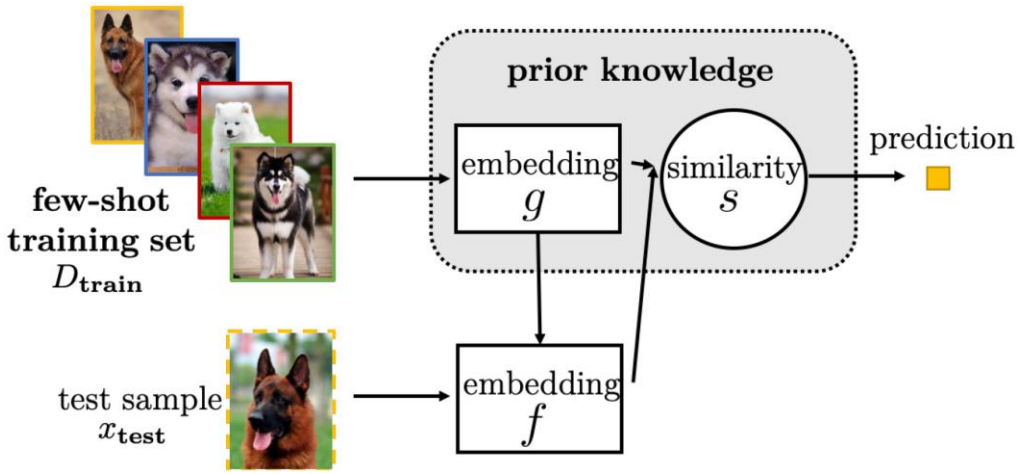


**图 13.1 混合嵌入模型**

### 13.3.3 基于外部记忆

基于外部记忆学习：使用外部记忆学习从训练数据中提取知识，并将其存储在外部存储器中，如图 13.13 所示。然后，每个新样本$x_{test}$用从内存中提取内容的加权平均值表示。这限制了$x_{test}$由内存中的内容表示，因此实质上减小了解空间 H 的大小。这个引入辅助记忆单元一般称为外部记忆，以区别与循环神经网络的内部记忆。

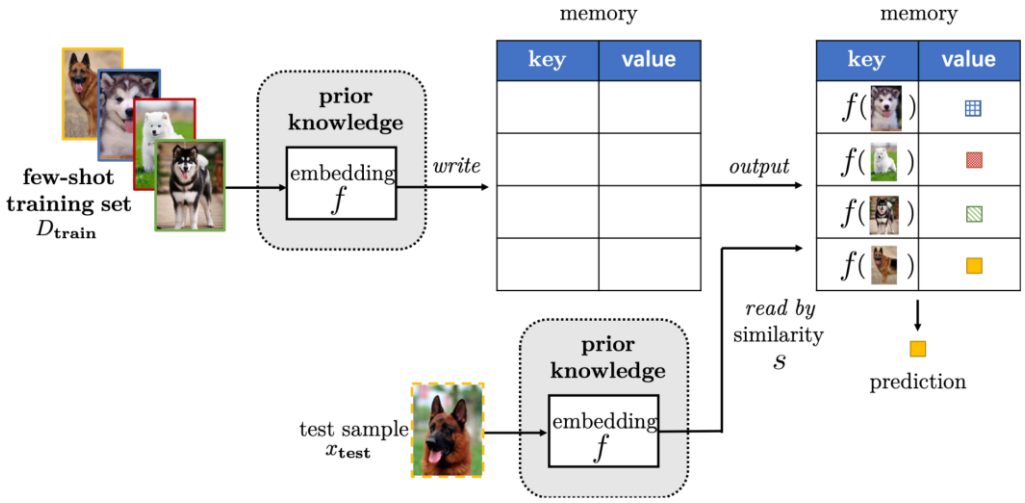


**图 13.13 外部记忆结构示意图**

如图 13.13 时外部记忆结构的示意图。

外部记忆单元 M 用来存储信息，一般可以分为很多记忆片段（Memory Segment），这些记忆片段按照一定的结构来进行般用向量来表示，外部记忆单元可以用一组向量来表示。这些向量的组织方式可以是集合、树、栈或队列等。大

部分信息存储于外部记忆中，不需要全时参与主网络的运算。

读取模块 R 根据主网络生成的查询向量$q_r$，从外部记忆单元中读取相应的信息。写入模块 W：根据主网络生成的查询向量$q_w$和要写入的信息 a 来更新外部记忆。

这种结构化的外部记忆是带有地址的，即每个记忆片段都可以按地址读取和写入。要实现类似于人脑神经网络的联想记忆能力，就需要按内容寻址的方式进行定位，然后进行读取或写入操作。按内容寻址通常使用注意力机制来进行。通过注意力机制可以实现一种“软性”的寻址方式，即计算一个在所有记忆片段上的分布，而不是一个单一的绝对地址。

比较典型的记忆网络是端到端记忆网络，可以多次从外部记忆中读取信息。在端到端记忆网络中，外部记忆单元是只读的。给定需要存储的信息 M，将其转换成两组记忆片段：用于寻址的片段 A 和用于进行输出的片段 C。主网络根据输入x来生成 q，并从外部记忆中读取相关信息 r，产生输出：$\mathrm{y} = \mathrm{f}(\mathrm{q} + \mathrm{r})$。为了实现更复杂的计算，可以进行 k 轮交互，第 k 轮交互产生的查询向量则为：$q^{(k)} = r^{(k-1)} + q^{(k-1)}$，相应的输出为：$\mathrm{y} = \mathrm{f}(q^{(k)} + r^{(k)})$。

### 13.3.4 生成建模方法

生成方法是由数据学习联合概率分布$\mathrm{P}(\mathrm{X},\mathrm{Y})$，然后由$\mathrm{P}(\mathrm{Y}|\mathrm{X}) = \mathrm{P}(\mathrm{X},\mathrm{Y})/\mathrm{P}(\mathrm{X})$求出概率分布$\mathrm{P}(\mathrm{Y}|\mathrm{X})$作为预测的模型，这就是需要确定的生成模型（如图 13.8 所示）。该方法表示了给定输入 X 与产生输出 Y 的生成关系。生成建模方法借助先验知识从观测到的$x_i$估计概率分布，此类中的方法可以处理许多任务，例如生成，识别，重构和图像翻转。根据变量代表的信息，现有方法可分为梁种：基于组件分解、基于相似类的先验概率分布。

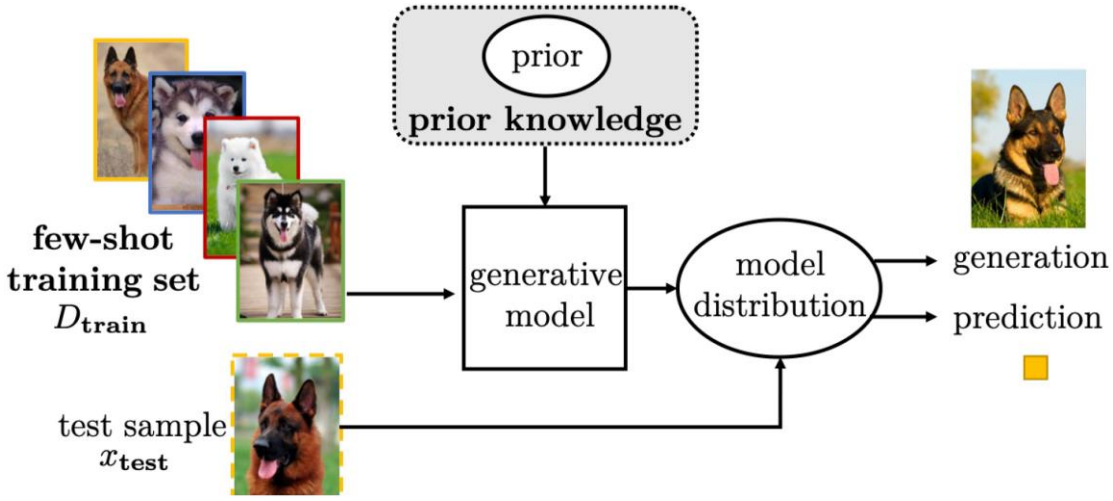


**图 13.13 基于生成模型的小样本算法**

（1）基于组件分解

尽管在小样本学习问题中具有监督信息的样本很少，但它们可能与其他任务的样本共享一些较小的可分解组件。例如，考虑只使用少量面部照片来识别一个人。虽然相似的面孔可能很难找到，但人们很容易找到眼睛、鼻子或嘴巴相似的照片。通过大量的样本，可以很容易地学习这些可分解组件的模型。然后，只需找到这些可分解组件的正确组合，并确定此组合属于哪个目标类。由于可分解的组件是由人选择的，因此该策略更易于解释。例如，Bayesian One shot 方法使用生成模型捕获可分解组件（即对象的形状和外观）与目标类（即待识别对象）之间的交互。贝叶斯进程学习（BPL）通过将字符分为类型、标记和进一步的模板、部分、原语来建模。要生成新角色，需要搜索包含这些组件的大型组合空间。

（2）基于相似类的先验概率分布

通常，相似的任务具有相似的先验概率，这以信息可以在小样本学习中得到利用。例如，考虑“橙色猫”、“豹”和“孟加拉虎”的三类分类。这三个物种是相似的，但孟加拉虎是濒危物种，而橙色的猫和豹是丰富的。因此，可以从“橙色猫”和“豹”中学习先验概率，并将其用作少数射击类“孟加拉虎”的先验概率。基于这一思想，通过无监督学习将一组数据集{Dc}分组到一个层次结构中。每组中的数据集一起学习类先验概率。对于一个新的小样本类，首先找到这个新类所属的组，然后根据从分组共享先验概率中提取的先验类对其进行建模。

### 13.3.5 基于度量学习

度量学习是一种空间映射的方法，其能够学习到一种特征空间，在此空间中，所有的数据都被转换成一个特征向量，并且相似样本的特征向量之间距离小，不相似样本的特征向量之间距离大，从而对数据进行区分。在度量空间中的学习异常高效，在小样本分类时效果很好。

在训练模型的过程，我们随意的选取两个样本，使用模型提取特征，并计算他们特征之间的距离。如果这两个样本属于同一个类别，那我们希望他们之间的距离应该尽量的小，甚至为 0；如果这两个样本属于不同的类别，那我们希望他们之间的距离应该尽量的大，甚至是无穷大。

大边界最近邻算法 LMNN 是最常使用的一种度量学习算法，其可以通过对训练集学习来得到一种原始数据的新度量，这种方法可以在一定程度上对原始数据分布进行重构，得到一个更加合理的数据分类空间。

LMNN 学习了为 k 近邻分类设计的伪测量，是以监督方式学习该全局度量的算法，以提高 k 最近邻规则的分类准确性。该算法基于半定规划，是凸优化的子类。LMNN 背后的主要直觉是学习伪测量，在该伪测量下，训练集中的所有数据实例被至少 k 个共享相同类标签的实例包围，该算法学习该类型的伪测量：$\mathrm{d}(\overrightarrow{x_i}-\overrightarrow{x_j})=(\overrightarrow{x_i}-\overrightarrow{x_j})^T M(\overrightarrow{x_i}-\overrightarrow{x_j})$，其中矩阵 M 需要是正半正定的。

以文本分类为例，首先要对文本进行特征提取将待测试文本和训练文本表示成向量空间模型。定义$\mathrm{D}=\{x_1,x_2,\dots,x_n\}$表示训练文本集合，$\mathrm{C}=\{c_1,c_2,\dots,c_m\}$为类别集合，其中$x_i=(d_1,d_2,\dots d_k)$表示第 i 篇文章，$d_i$表示文本向量的第 i 维，采用 IG 算法作为特征提权算法，然后采用 LMNN 方法对训练数据集进行重构，最后使用 K 近邻分类器来实现文本分类，评价标准使用 F1 值和查准率、查全率。

具体流程：

（1）首先，对中文文本进行分词、去停用词等预处理。

（2）对文本进行特征选择，可以选用 IG 这种常用的的特征提取算法来对文本进行特征提取。

（3）构造向量空间模型(Vector SpaceModel，VSM)，可以采用经典 TF*IDF 法。

（4）对训练样本以欧氏距离用留一法计算出训练集中每个数据点的先验知识 K 近邻，并做好标签，设定此 K 值为 Kp 。

（5）利用 LMNN 算法对训练集进行学习，求出映射矩阵 L。

（6）对训练样本和测试样本分别作映射。

（7）跟据基于 LMNN 的文本分类算法对测试集进行分类。

### 13.3.6 代码实例

进一步，读者也可以基于孪生网络与对比损失的深度度量学习方法进行实例学习。代码地址：

https://github.com/harveyslash/Facial-Similarity-with-Siamese-Networks-in-Pytorch/blob/master/Siamese-networks-medium.ipynb

网络训练过程如下图所示。

```
for epoch in range(0,Config.train_number_epochs):
    for i, data in enumerate(train_dataloader,0):
        img0, img1 , label = data
        img0, img1 , label = img0.cuda(), img1.cuda() , label.cuda()
        optimizer.zero_grad()
        output1,output2 = net(img0,img1)
        loss_contrastive = criterion(output1,output2,label)
        loss_contrastive.backward()
        optimizer.step()
```

**图 13.14 孪生网络训练过程**

对比损失的具体计算方法如下图所示：

```
class ContrastiveLoss(torch.nn.Module):
    """
    Contrastive loss function.
    Based on: http://yann.lecun.com/exdb/publis/pdf/hadsell-chopra-lecun-06.pdf
    """

    def __init__(self, margin=2.0):
        super(ContrastiveLoss, self).__init__()
        self.margin = margin

    def forward(self, output1, output2, label):
        euclidean_distance = F.pairwise_distance(output1, output2, keepdim = True)
        loss_contrastive = torch.mean((1-label) * torch.pow(euclidean_distance, 2) +
                                      (label) * torch.pow(torch.clamp(self.margin - euclidean_distance, min=0.0), 2))


        return loss_contrastive
```

**图 13.15 对比损失计算方法**

### 13.3.7 分析与总结

当存在相似任务或辅助任务时，多任务学习可以用来约束小样本任务的解空间。但是，请注意，需要对所有任务进行联合培训。因此，当一个新的少量任务到达时，整个多任务模型必须再次训练，这可能代价高昂且速度缓慢。此外，整体数据集和小样本数据集的尺寸不应具有可比性，否则，小样本任务可能会被具有大量样本的任务所淹没。

当存在包含各种类的足够样本的大规模数据集时，可以使用嵌入学习方法。这些方法将样本映射到一个良好的嵌入空间，在该空间中，来自不同类的样本可以很好地被分离，因此对解空间有了良好的限制。但是，当小样本任务与其他任务关系不密切时，它们可能无法很好地工作。此外，对如何混合任务的不变信息和特定于任务的信息进行更多的探索也是有帮助的。

当有内存网络时，通过在内存上训练一个简单的模型（例如，分类器），它可以很容易地用于 FSL。通过使用精心设计的更新规则，可以有选择地保护内存插槽。这种策略的缺点是它会带来额外的空间和计算成本，这会随着内存大小的增加而增加。因此，当前外部存储器的大小有限。

最后，当除小样本任务外还需要执行生成或重构等任务时，可以使用生成模型。他们从其他数据集中学习先验概率 p（z；γ），从而将解空间 H 减小到较小的$H^{\sim}$。所学习的生成模型也可用于生成用于数据扩充的样本。然而，生成性建模方法具有较高的推理成本，并且比确定性模型更难推导。

## 13.4 基于求解方法的小样本算法

本节的方法目的是研究如何在假设空间中得到更好的局部最优解，例如最流行的随机梯度下降法。当监督信息丰富时，有足够的训练样本更新模型参数。然而，在小样本问题中，监督信息不够，导致得到的局部最优解不可靠。本节的方法利用其他信息影响局部最优解的获得方式。具体可分为 1）微调现有参数；2）微调元学习参数；3）学习优化器三个类型，我们将逐一介绍。

### 13.4.1 微调现有参数

这类方法首先通过其他任务获得一个较为合适的参数初始值$\theta_0$，之后使用目标数据集对参数进行收到一系列约束的微调训练，如图 13.16。

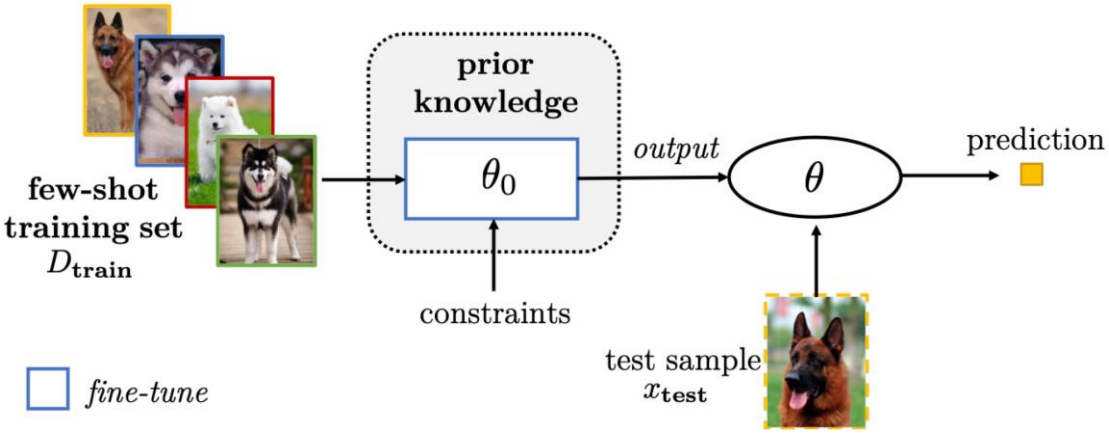


**图 13.16 基于微调现有参数的小样本方法流程**

这些约束主要是为了减少过拟合，主要包括：

（1）提前结束训练。它需要将验证集与训练集分离，以监控训练过程。当验证集没有性能改进时，停止学习。

（2）选择更新参数。只更新一部分参数来减弱过拟合现象。譬如有的方法利用一组预训练好的卷积层，只更新与这组卷积层有关的参数。

（3）联合更新参数。每组参数使用相同的更新信息来更新。

### 13.4.2 聚合参数

有时，我们没有一个合适的初始参数开始。相反，我们有许多从相关任务中学习的模型。例如，在人脸识别中，我们可能已经有了眼睛、鼻子和耳朵的识别模型。因此，可以将这些模型参数聚合到合适的模型中，然后直接使用或由训练样本进行微调（如图 13.17 所示）。

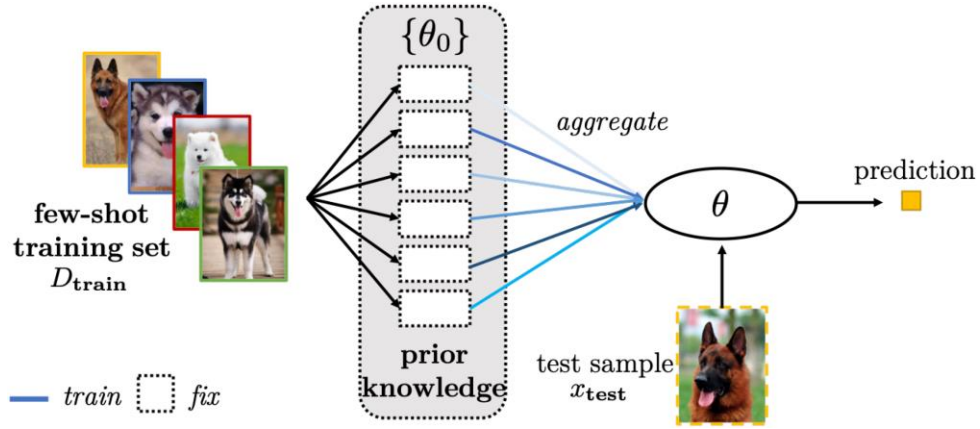


**图 13.17 基于聚合参数的小样本学习方法**

### 13.4.3 利用新的参数进行微调

预先训练的参数$\theta_0$可能不足以完全编码新的小样本任务。因此，可以使用附

加参数$\delta$来约束训练样本的特性，如图 13.18 所示。具体来说，该策略将模型参数扩展为$\theta = \{\theta_0, \delta\}$，并在学习$\delta$的同时微调$\theta_0$。举例来说，一些方法使用其他数据对卷积神经网络的特征提取部分进行预训练，之后利用提取的特征对分类部分的参数进行从头训练。

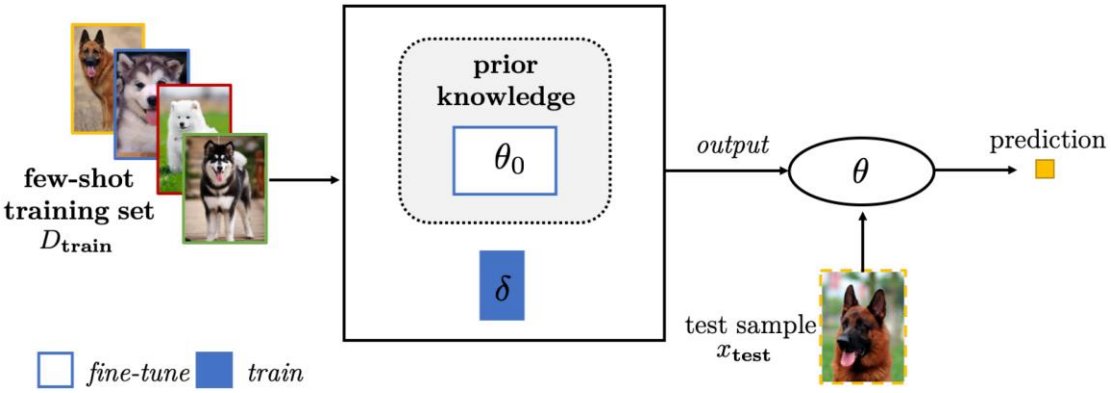


**图 13.18 利用新的参数进行微调的小样本学习方法**

### 13.4.4 元学习

本节中的方法使用元学习的思想来解决小样本学习问题，如图 13.13 所示。

元学习（Meta-Learing），又称“学会学习”（Learning to learn），即利用以往的知识经验来指导新任务的学习，使网络具备学会学习的能力。元学习中要准备许多任务来进行学习，而每个任务又有各自的训练集和测试集，是解决小样本问题（Few-shot Learning）常用的方法之一。

元学习的本质是增加学习器在多任务的泛化能力，元学习对于任务和数据都需要采样，因此学习到的 F(x)可以在未出现的任务中迅速（依赖很少的样本）建立起 mapping。因此“元”体现在网络对于每个任务的学习，通过不断的适应每个具体任务，使网络具备了一种抽象的学习能力。

Meta-learning 中为了区别概念，将训练过程定义为“Meta-training”、测试过程定义为“Meta-testing”，如下图所示：

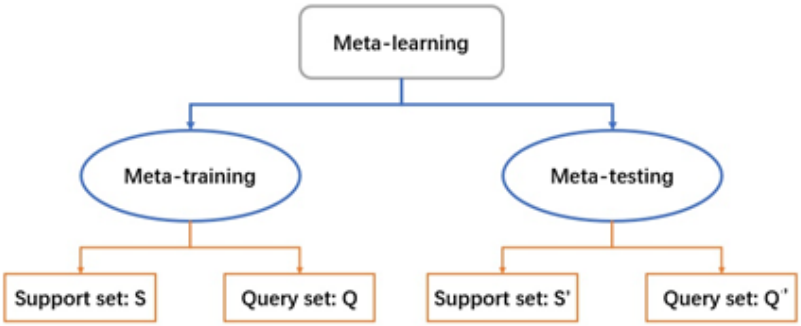


图 13.19 元学习示意图

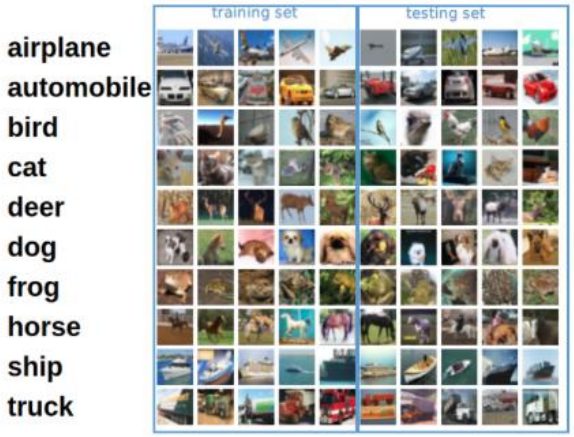


**图 13.20 传统深度学习数据集划分示意图**

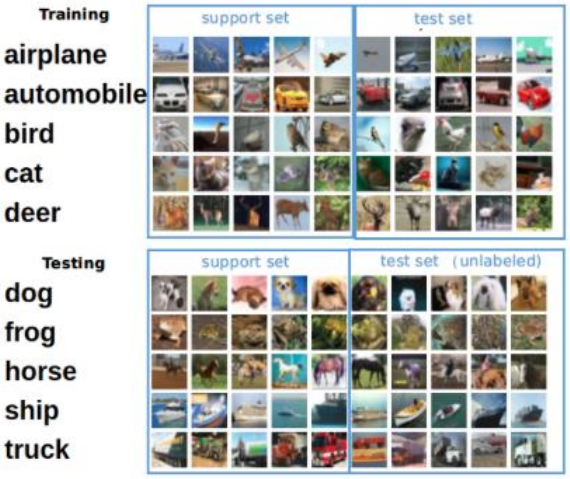


**图 13.21 元学习数据集划分示意图**

如图 13.10 所示，传统的图像分类任务是基于左边的给定训练数据，获得 model，然后在右边的数据集上测试 model 的好坏。而对于小样本问题，其训练数据和测试数据如图 13.11 所示。我们拥有的是大量的上方这些数据，也就是对

于 training 中的 airplane、automobile 等，我们有很多类数据，而对于下方 Testing 中像 dog、frog 等新的分类问题，就没有那么多类的标注数据。区别于一般神经网络端到端的训练方式，元学习的训练过程和测试过程各需要两类数据集（Support/Query set）。小样本分类任务属于“N-way， k-shot”问题，其中，N 代表选择的 Testing data 中样本的种类，k 代表选择的 K 类 Testing data 中每类样本的数量，一般来说 N 小于 Testing data 的总类别数。

我们从 Testing data 中随机选出 N 个类。然后，再从这 N 个类中按照类别依次随机选出 k+x 个样本（x 代表可以选任意个），其中的 k 个样本将被用作 Support set S'，另外的 x 个样本将被用作 Query set Q'。S 和 Q 的构建同理，不同的是从 Training data 中选择的样本类别和每类样本数量均不做约束。

Meta-learning 通常采用一种被称为 Episodic Training 的方法来进行训练，它偏重于任务和数据的双重采样，即任务和数据一样是需要采样的。具体来说对于一个 10 分类任务，元学习通过可能只会建立起一个 5 分类器，每个训练的 episode 都可以看成是一个子任务，而学习到的 F(x)可以帮助在未见过的任务里迅速建立 mapping。

元学习的具体训练流程如下表所示：

表 13-1 元学习训练流程

1. 准备 N 个训练任务，每个任务都有对应的 Support Set 和 Query Set。再准备几个测试任务，测试任务用于评估 meta learning 学习到的参数效果。训练任务和测试任务均从整体数据集中采样得到。
2. 定义网络结构，譬如 CNN。并初始化一个 meta 网络的参数为$\phi^0$，meta 网络是最终要用来应用到新的测试任务中的网络，该网络中存储了“先验知识”。
3. 开始执行迭代“预训练”：

a. 采样一个训练任务 m（或几个训练任务）。将 meta 网络的参数$\phi^0$赋值给任务 m 对应的 CNN，得到$\theta^m$。

b. 使用任务 m 的 Support Set，基于任务 m 的学习率$\alpha_m$，对$\theta^m$进行 1 次更新，得到$\hat{\theta}^m$

c.基于 1 次优化后的$\hat{\theta}^m$，使用 Query Set 计算任务 m 的 loss——$l^m(\hat{\theta}^m)$，并计算其对$\hat{\theta}^m$的梯度。

d.用该梯度，乘 meta 网络的学习率$\alpha_{meta}$，更新$\phi^0$，得到$\phi^1$。

e. 继续采样一个任务 n，将$\phi^1$赋值给任务 n 对应的 CNN 网络，得到$\theta^n$；

f. 使用任务 n 的训练数据对$\theta^n$进行优化得到$\hat{\theta}^n$；

g.基于 1 次优化后的$\hat{\theta}^n$，使用 Query Set 计算任务 n 的 loss——$l^n(\hat{\theta}^n)$，并计算其对$\hat{\theta}^n$的梯度；

h. 用该梯度，乘 meta 网络的学习率$\alpha_{meta}$，更新$\phi^1$，得到$\phi^2$；

i. 在数据集中重复 a-h 的过程；

4. 通过 3 得到的 meta 网络可用于测试任务中。使用测试任务的 Support Set 对 meta 网络进行微调。

5. 最终使用测试任务的 Query Set 评估 meta learning 效果。

元学习与普通模型预训练的是有本质不同的。Meta learning 是使用子任务的参数，**第二次更新**的 gradient 的方向来更新参数（所以图 13.14 中，第一个蓝色箭头的方向与第二个绿色箭头的方向平行；下图第二个蓝色箭头的方向与第二个橘色箭头的方向平行。model pretraining 是使用子任务第一步更新的 gradient 的方向来更新参数(子任务的梯度往哪个方向走，model 的参数就往哪个方向走)

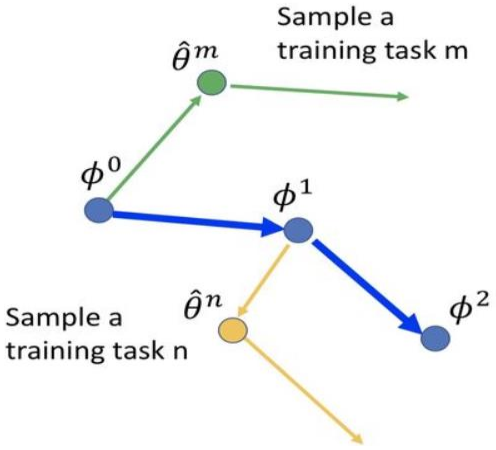


**图 13.22 元学习参数更新过程**

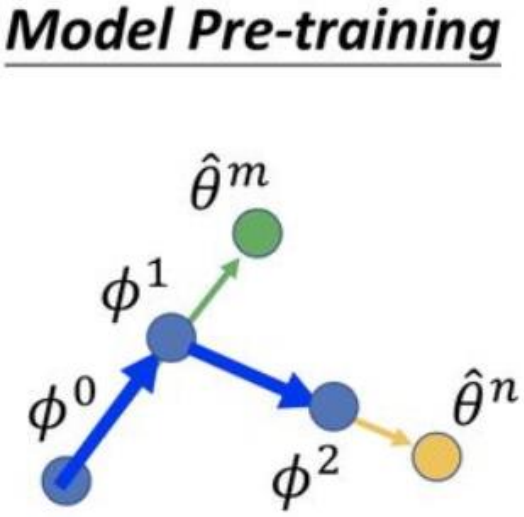


**图 13.23 模型预训练参数更新过程**

### 13.4.5 代码实例

本节基于 MAML 元学习算法代码对元学习步骤进行展示。代码地址：https://github.com/dragen1860/MAML-Pytorch

**1）元学习预训练阶段——更新任务网络参数**

```
for i in range(task_num):

    # 1. run the i-th task and compute loss for k=0
    logits = self.net(x_spt[i], vars=None, bn_training=True)
    loss = F.cross_entropy(logits, y_spt[i])
    grad = torch.autograd.grad(loss, self.net.parameters())
    fast_weights = list(map(lambda p: p[1] - self.update_lr * p[0], zip(grad, self.net.parameters())))

    # this is the loss and accuracy before first update
    with torch.no_grad():
        # [setsz, nway]
        logits_q = self.net(x_qry[i], self.net.parameters(), bn_training=True)
        loss_q = F.cross_entropy(logits_q, y_qry[i])
        losses_q[0] += loss_q

        pred_q = F.softmax(logits_q, dim=1).argmax(dim=1)
        correct = torch.eq(pred_q, y_qry[i]).sum().item()
        corrects[0] = corrects[0] + correct

    # this is the loss and accuracy after the first update
    with torch.no_grad():
        # [setsz, nway]
        logits_q = self.net(x_qry[i], fast_weights, bn_training=True)
        loss_q = F.cross_entropy(logits_q, y_qry[i])
        losses_q[1] += loss_q
        # [setsz]
        pred_q = F.softmax(logits_q, dim=1).argmax(dim=1)
        correct = torch.eq(pred_q, y_qry[i]).sum().item()
        corrects[1] = corrects[1] + correct

    for k in range(1, self.update_step):
        # 1. run the i-th task and compute loss for k=1~K-1
        logits = self.net(x_spt[i], fast_weights, bn_training=True)
        loss = F.cross_entropy(logits, y_spt[i])
        # 2. compute grad on theta_pi
        grad = torch.autograd.grad(loss, fast_weights)
        # 3. theta_pi = theta_pi - train_lr * grad
        fast_weights = list(map(lambda p: p[1] - self.update_lr * p[0], zip(grad, fast_weights)))

        logits_q = self.net(x_qry[i], fast_weights, bn_training=True)
        # loss_q will be overwritten and just keep the loss_q on last update step.
        loss_q = F.cross_entropy(logits_q, y_qry[i])
        losses_q[k + 1] += loss_q

        with torch.no_grad():
            pred_q = F.softmax(logits_q, dim=1).argmax(dim=1)
            correct = torch.eq(pred_q, y_qry[i]).sum().item()  # convert to numpy
            corrects[k + 1] = corrects[k + 1] + correct
```

图 13.24 更新任务网络参数

2）预训练阶段——更新元网络参数

```
loss_q = losses_q[-1] / task_num

# optimize theta parameters
self.meta_optim.zero_grad()
loss_q.backward()
# print('meta update')
# for p in self.net.parameters()[:5]:
#   print(torch.norm(p).item())
self.meta_optim.step()
```

图 13.25 更新元网络参数

3）Fine-tune 阶段

```
    net = deepcopy(self.net)

    # 1. run the i-th task and compute loss for k=0
    logits = net(x_spt)
    loss = F.cross_entropy(logits, y_spt)
    grad = torch.autograd.grad(loss, net.parameters())
    fast_weights = list(map(lambda p: p[1] - self.update_lr * p[0], zip(grad, net.parameters())))

    # this is the loss and accuracy before first update
    with torch.no_grad():
        # [setsz, nway]
        logits_q = net(x_qry, net.parameters(), bn_training=True)
        # [setsz]
        pred_q = F.softmax(logits_q, dim=1).argmax(dim=1)
        # scalar
        correct = torch.eq(pred_q, y_qry).sum().item()
        corrects[0] = corrects[0] + correct

    # this is the loss and accuracy after the first update
    with torch.no_grad():
        # [setsz, nway]
        logits_q = net(x_qry, fast_weights, bn_training=True)
        # [setsz]
        pred_q = F.softmax(logits_q, dim=1).argmax(dim=1)
        # scalar
        correct = torch.eq(pred_q, y_qry).sum().item()
        corrects[1] = corrects[1] + correct

    for k in range(1, self.update_step_test):
        # 1. run the i-th task and compute loss for k=1~K-1
        logits = net(x_spt, fast_weights, bn_training=True)
        loss = F.cross_entropy(logits, y_spt)
        # 2. compute grad on theta_pi
        grad = torch.autograd.grad(loss, fast_weights)
        # 3. theta_pi = theta_pi - train_lr * grad
        fast_weights = list(map(lambda p: p[1] - self.update_lr * p[0], zip(grad, fast_weights)))

        logits_q = net(x_qry, fast_weights, bn_training=True)
        # loss_q will be overwritten and just keep the loss_q on last update step.
        loss_q = F.cross_entropy(logits_q, y_qry)

        with torch.no_grad():
            pred_q = F.softmax(logits_q, dim=1).argmax(dim=1)
            correct = torch.eq(pred_q, y_qry).sum().item()  # convert to numpy
            corrects[k + 1] = corrects[k + 1] + correct
```

**图 13.26 元学习 fine-tune 阶段**

### 13.4.6 分析与总结

微调现有参数可以减少假设空间中的搜索工作量。通过使用现有$\theta_0$作为初始化，这些方法通常需要较低的计算成本来获得良好的假设 h∈ H.学习的重点是完善这些现有参数。然而，由于$\theta_0$是从与当前任务不同的任务中学习的，因此该策略可能会牺牲精度来提高速度。

另外两种策略依赖于元学习。通过从一组相关任务中学习，基于元学习得到的$\theta_0$可以更接近新任务$T_t$的任务特定参数$\phi_t$。元学习学习搜索步骤可以直接指导学习算法。换句话说，元学习者充当优化器。然而，诸如如何跨不同粒度（例如动物的粗粒度分类与狗种的细粒度分类）或不同数据源（例如图像与文本）学习数据等重要问题仍有待解决。从这个角度来看，元学习和多任务学习是相似的，因此如何避免负迁移也是一个值得关注的问题。

## 13.5 主动学习算法

机器学习中的有监督学习和半监督学习都需要大量的标注数据，但是在实际的业务场景或生产环境中，标记样本的获得是一个困难、耗时且昂贵的过程。因此如何通过较少的成本来获得较大价值的标注数据，并进一步地提升算法地效果成为一个值得思考的问题。主动学习（Active Learning）就是为了解决此类问题而提出的一种算法，在统计学领域也叫查询学习、最优实验设计。

在没有使用主动学习（Active Learning）的时候，通常来说系统会从样本中随机选择或者使用一些人工规则的方法来提供待标记的样本供人工进行标记。这样虽然也能够带来一定的效果提升，但是其标注成本总是相对大的。

主动学习是解决小样本问题的关键技术之一。相对于被动学习而言，被动学习基于已经标注好的数据集上进行模型训练，这些标记数据集是人工随机选取，造成有限标记训练样本中的信息量严重匮乏；主动学习通过人机交互方式，基于查询函数，有选择性的从大量未标记数据中选取信息量最丰富的样本进行人工标记，使得小样本数据中的信息尽可能的代表未标记数据中信息量，从而实现小样本下对大数据的分析处理，同时降低耗费。

例如，一个高中生通过做高考的模拟试题以希望提升自己的考试成绩，那么在做题的过程中就有几种选择。一种是随机地从历年高考和模拟试卷中随机选择一批题目来做，以此来提升考试成绩。但是这样做的话所需要的时间也比较长，针对性也不够强；另一种方法是每个学生建立自己的错题本，用来记录自己容易做错的习题，反复地巩固自己做错的题目，通过多次复习自己做错的题目来巩固自己的易错知识点，逐步提升自己的考试成绩。主动学习就是选择一批容易被错分的样本数据，让人工进行标注，再让机器学习模型训练的过程。

### 13.5.1 算法说明

主动学习可以使用尽可能少的标注数据集训练一个模型，使得模型性能可以达到一个由大量的标注数据集按照普通方法（随机选择训练数据）训练得到的模型的性能。以数据集 Cifar10 为例，该数据集共有 60000 张彩色图像，这些图像大小是 32*32，分为 10 个类，每类 6000 张图。 主动学习的过程如下。

（1）数据：active_samples = 50000，val_samples = 10000；train_samples = 0。

（2）初始化alexnet模型，随机权重得到最初的模型，记为cifar10_alexnet_imagenet_init.7t；num_train_samples = 0。

（3）分别对 active_samples 目录下的 50000 − num_train_samples 张数据进行预测，得到 10 个类别对应的 10 个概率值；

（4）重点关注每个样本预测结果的最大概率值：p_pred_max。我们初步认为 p_pred_max > 0.5 的情况表示当前模型对该样本有个确定的分类结果（此处分类结果的正确与否不重要）；反之，当前模型对该样本的判断结果模棱两可，标记为hard sample；比如：模型进行第一次预测，得到 10 个概率值，取其最大的概率p_pred_max；

（5）对P(real lable) < p_threshold（此处的 10 分类任务取p_threshold = 0.5）的样本进行排序，取前N个样本加入集合train_samples中；

（6）基于当前的训练数据集train_samples对模型进行微调，得到新的模型记为model_fine_tuned.7t；

（7）重复（3）到（6）步骤，直到 active_samples 样本数为 0 或者当前模型 model_fine_tuned.7t 已经达到理想效果

实验结果如表 13.2 所示，实验结果表明引入 active learning 不仅能够得到减少样本标注代价，还能够提升分类的准确率。验证集准确率 val_acc 就能够达到 99.04%，将剩余的训练样本扔到训练好的模型进行预测，剩余训练样本的准确率 Acc_left_active_samples 能够达到 99.39%。

表 13.2 Cifar10 分类实验结果

| Method | Train Set | Val Set | Model | Val-acc | Acc_left_active_samples |
|---|---|---|---|---|---|
| Just Train | 50000 | 10000 | AlexNet | 90.00% | - |
| Random select | 44000 | 10000 | AlexNet | 90.06% | - |
| Active learning | 17750 | 10000 | AlexNet | 90.16% | 99.39% |

### 13.5.2 理论基础与基本模型

主动学习的大致思路就是：通过机器学习的方法获取到那些比较“难”分类

的样本数据，让人工再次确认和审核，然后将人工标注得到的数据再次使用有监督学习模型或者半监督学习模型进行训练，逐步提升模型的效果，将人工经验融入机器学习的模型中。

在实际应用中，很多问题是动态变化的，需要不断修正数据，否则很难保证训练模型的准确性。比如，在微博文本分类中，用户随着兴趣爱好的改变，微博状态也会随之改变，以前用户关注的是生活、旅游和科技，可能在某个时期对武器、军事等产生兴趣，微博转发内容发生改变，这时候就需要人工标记少许文本和更新模型，从而提供更加个性化的服务。主动学习的适用场景较为灵活和宽泛，仅仅需要少量的标记样本，就可以逐步建立可靠数据集。

主动学习技术主要通过提升训练样本的质量来提升分类模型的性能。通过样本数量进行训练集信息量的扩充，同样可以达到提升分类器性能的目的。其模型如下: A=(C,Q,S,L,U)，其中 C 为一组或者一个分类器，L 是用于训练已标注的样本。Q 是查询函数，用于从未标注样本池 U 中查询信息量大的信息，S 是督导者，可以为 U 中样本标注正确的标签。学习者通过少量初始标记样本 L 开始学习，通过一定的查询函数 Q 选择出一个或一批最有用的样本，并向督导者询问标签，然后利用获得的新知识来训练分类器和进行下一轮查询。主动学习是一个循环的过程，直至达到某一停止准则为止。

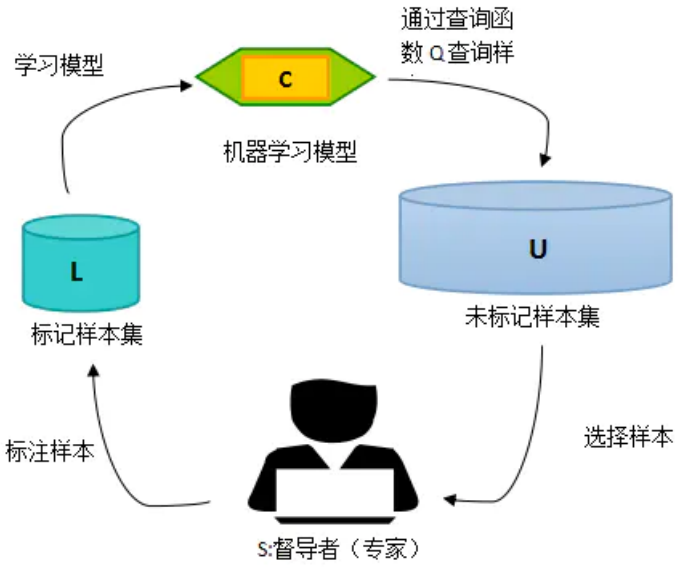


图 13.27 主动学习示意图

主动具体流程如下：

（1）选取合适的分类器（网络模型）记为 current_model 、主动选择策略、数据划分为 train_sample（带标注的样本，用于训练模型）、validation_sample（带标注的样本，用于验证当前模型的性能）、active_sample（未标注的数据集，对应于 ublabeled pool）；

（2）初始化：随机初始化或者通过迁移学习（source domain）初始化；如果有 target domain 的标注样本，就通过这些标注样本对模型进行训练；

（3）使用当前模型 current_model 对 active_sample 中的样本进行逐一预测（预测不需要标签），得到每个样本的预测结果。此时可以选择 Uncertainty Strategy 衡量样本的标注价值，预测结果越接近 0.5 的样本表示当前模型对于该样本具有较高的不确定性，即样本需要进行标注的价值越高。

（4）专家对选择的样本进行标注，并将标注后的样本放至 train_sapmle 目录下。

（5）使用当前所有标注样本 train_sample 对当前模型 current_model 进行 fine-tuning，更新 current_model；

（6）使用 current_model 对 validation_sample 进行验证，如果当前模型的性能得到目标或者已不能再继续标注新的样本（没有专家或者没有钱），则结束迭代过程。否则，循环执行步骤（3）-（6）。

### 13.5.3 查询函数设计准则

在各种主动学习方法中，查询函数的设计最常用的策略是：不确定性准则、代表性准则以及两者的联合准则。

不确定性准则是主动学习中最受关注的准则，其研究比较宽泛，包括边缘采样，专家委员会，期望误差减小等。对于不确定性，我们可以借助信息熵的概念来进行理解。我们知道信息熵是衡量信息量的概念，也是衡量不确定性的概念。信息熵越大，就代表不确定性越大，包含的信息量也就越丰富。

事实上，有些基于不确定性的主动学习查询函数就是使用了信息熵来设计的，比如熵值装袋查询（Entropy query-by-bagging）。所以，不确定性策略就是要想方

设法地找出不确定性高的样本，因为这些样本所包含的丰富信息量，对我们训练模型来说就是有用的。

查询函数每次迭代中查询一个或者一批样本。我们当然希望所查询的样本提供的信息是全面的，各个样本提供的信息不重复不冗余，即样本之间具有一定的差异性。在每轮迭代抽取单个信息量最大的样本加入训练集的情况下，每一轮迭代中模型都被重新训练，以新获得的知识去参与对样本不确定性的评估可以有效地避免数据冗余。但是如果每次迭代查询一批样本，那么就应该想办法来保证样本的差异性，避免数据冗余。

代表性准则，在去除冗余的同时，通过分布估计使得标记数据集和未标记数据集之间具有相似的分布，挖掘数据内部结构信息。不同于一般的差异性策略，代表性准则不需要预先依据不确定性准则进行样本的预筛选，而是可以直接看作是主动学习查询函数，其目标是保证选择的样本与未标记数据集具有独立同分布的特性。

主动学习过程中准则的联合学习主要是将不确定性和代表性结合到同一个主动学习框架中，查询既具有不确定性又具有代表性的样本。例如，利用稀疏表达进行样本相似性表达，将不确定性的衡量作为稀疏表达中稀疏系数的权重，实现从不确定性高的样本中选择相似性表达能力强的样本。

### 13.5.4 样本选择策略

从主动学习的核心过程可以发现，主动学习过程中最重要的部分是查询函数 F 的确定。上一节介绍了查询函数的设计准则，本节将具体介绍目前较为常用的样本选择策略，主要有不确定采样、专家委员会、期望模型变化和期望模型最小化四种策略。

**1.不确定采样**

不确定性采样是主动学习中最常用和最经典的方法，其查询的是在分类过程中分类结果可靠性最低的样本。比如，边缘采样策略，通过衡量点到面的距离，选择离分类平面最近的点作为查询的最具信息量的样本

$$x^* = argmin_{x \in U} |f(x)|$$

边缘采样对于样本不确定性的描述较为直接，衡量值越小，说明样本可靠性越低。

目前很多主动学习方法都是基于此选择策略进行主动学习算法的进一步优化和提升。

边缘采样仅考虑了一种标签的可能性，针对多类别问题可以采用最大的类别预测概率和第二大的类别预测概率进行不确定性的衡量，称为 BvSB(Bestvs Second Best)，表达式为

$$x^* = \ argmin_{x \in U} P(y_1|x) - P(y_2|x)$$

$P(y_1|x)$表示 $x$类别归属概率最大值，$P(y_2|x)$表示类别归属第二大概率值。$x^*$越小则表明样本$x$被判断为类别$y_1$和$y_2$的概率越相同，其主要查找位于类别$y_1$和$y_2$边界上的样本。

但是当类别数较多时，BvSB 则会忽视其它类别的分类信息，为此对所有的分类损失进行综合考虑，提出了基于熵的主动学习方法：

$$x^* = \ argmax_{x \in U} \sum_{i=1}^{C} -p_i(y_i|x) log p(y_i|x)$$

上述三种方法是目前主动学习算法中较为常用的策略。

**2.专家委员会**

专家委员会是主动学习中另一种样本选择机制，主要利用有限的训练样本构造多个弱分类模型，将这多个分类模型看成是专家委员会对未标记样本进行预测，选取委员会预测结果最不一致的样本, 作为最具有信息量的样本。常用的机制有投票熵：

$$x^* = argmax_{x \in U} - \sum_{y} \frac{V(y)}{N} log \frac{V(y)}{N}$$

其中 N 表示的是专家委员会的个数，V(y) 表示专家委员会中对x预测标签为 y 的个数。

平均 KL(Kullback-Leibler) 散度也是常用的投票不一致性指标，表达式为:

$$x^* = argmax_{x \in U} \frac{1}{N} \sum_{i=1}^{N} D(p_c || P_N)$$

其中：

$$D(p_c||P_N)=\sum_{i=1}^{c}p_c(y_i|x)log\frac{p_c(y_i|x)}{P_N(y_i|x)}$$

$p_c(y_i|x)$表示专家委员会中第 c 个分类模型将 x 判属于$y_i$的概率，$P_N(y_i|x)$表示专家委员会中所有模型将 x 判属于$y_i$的概率和的平均值。专家会员会相当于利用多个模型进行集成，具有集成学习的优点，在专家委员会个数较少时便可以查询到最具信息量的样本。对于专家委员会的不一致性度量，还有很多指标比如 Jensen-Shannon 散度等。

**3.期望模型变化**

期望模型变化策略与不确定性采样和专家委员会不同，其主要是选取能够使当前模型产生最大变化的样本。就是当选取的样本 x 标记以后加入训练集时，训练模型能够产生最大变化。定义 $\nabla l_\theta(\mathrm{L})$ 表示目标函数中模型参数 θ 的梯度，使 $\nabla l_\theta(\mathrm{L})\cup\langle \mathrm{x},\mathrm{y}\rangle$表示训练集L加入训练样本⟨x, y⟩以后模型参数θ的新梯度。由于x是待查的样本，其标签y是未知的，因此，必须用对所有标签进行梯度的计算作为期望来代替真实的模型变化衡量，其目标函数表达式为:

$$x^*=argmax_{x\in U}\sum_{i=1}^{c}P_\theta(y_i|x)\,||\nabla l_\theta(\mathrm{L}\cup<\mathrm{x},y_i>)||$$

其中$||\cdot||$表示加入样本梯度向量的欧式范数。应该注意到，相对于加入的单个样本，训练集 L 的数量在梯度中作用更大，因此为了加快学习效率，可以在训练模型梯度$\nabla l_\theta(\mathrm{L})$上直接进行梯度求解，即$\nabla l_\theta(\mathrm{L}\cup<\mathrm{x},y_i>)\approx\nabla l_\theta(<\mathrm{x},y_i>)$，这样对于数目多的数据集可以进行并行计算。但是如果要真实的计算其期望模型变化，样本数量和标签数量大时，此方法的计算时间复杂度较高。事实上，模型期望变化就是衡量样本在无标签下对于模型改变的影响。

**4.期望误差减小**

期望误差减小不同于期望模型改变，其主要查询能够使模型泛化误差最大限度减小的样本。通过$\mathrm{L}\cup<\mathrm{x},\mathrm{y}>$进行模型训练，在剩余的未标记样本集 U 上进行模型泛化误差的衡量，由于假定未标记集和测试集是独立同分布的，因此，在 U 上的测试可以表示模型的泛化能力。但是由于样本的标签是未知的，因此仍然采用当前模型参数θ对所有标签进行遍历，其目标函数就是期望减少预测错误标

签的数量，可以表示为最小化期望的 log-loss：

$$x^* = argmin_{x \in U} \sum_{i=1}^{c} P_\theta(y_i|x) \left(-\sum_{u=1}^{U}\sum_{j=1}^{C} P_{\theta+<x,y_i>}(y_j|x^{(u)}) log P_{\theta+<x,y_i>}(y_j|x^{(u)})\right)$$

除了熵值形式，还可以通过最大化样本的增益信息或者互信息，来选取最具信息量的样本。

### 13.5.5 代码实例

本章对深度主动学习方法实现进行简要介绍。代码地址：https://github.com/ej0cl6/deep-active-learning。

如下图所示，首先通过一系列策略对深度网络进行训练以及验证。之后将大量未标记样本输入网络，得到这些未标记样本对应标签的概率，并取概率最大的标签作为未标记样本的真实标签。这些策略包括但不限于"RandomSampling","LeastConfidence","MarginSampling","EntropySampling","LeastConfidenceDropout","MarginSamplingDropout","EntropySamplingDropout","KMeansSampling","KCenterGreedy","BALDDropout","AdversarialBIM","AdversarialDeepFool","ActiveLearningByLearning"等。

```
strategy.train()
preds = strategy.predict(dataset.get_test_data())
print(f"Round 0 testing accuracy: {dataset.cal_test_acc(preds)}")

for rd in range(1, args.n_round+1):
    print(f"Round {rd}")

    # query
    query_idxs = strategy.query(args.n_query)

    # update labels
    strategy.update(query_idxs)
    strategy.train()

    # calculate accuracy
    preds = strategy.predict(dataset.get_test_data())
    print(f"Round {rd} testing accuracy: {dataset.cal_test_acc(preds)}")
```

**图 13.28 主动学习方法代码实例**

### 13.5.6 分析与总结

主动学习算法的核心思想是通过算法对大量的无标记样本自动的滤出一些值得标记的样本，并通过手工标记的方法得到含有更多高质量样本的数据集。从而缓解小样本问题中的过拟合问题。

## 13.6 小样本学习的延伸——零样本学习

小样本学习试图仅从一个或少量几个有标签训练样本中解决目标学习，而当目标任务的训练数据完全缺失时，如何解决该学习问题，则是零样本学习（zero-shot learning）的学习目标。

简单来说，零样本学习就是让计算机具备人类的推理能力，使其可以识别出一个从未见过的新事物。比如，我们告诉一个从没见过斑马的小朋友：“斑马长得像马，身上有像老虎一样的条纹，并且像熊猫一样是黑白色的”，他就可以很轻松地在动物园里找出哪个是斑马。

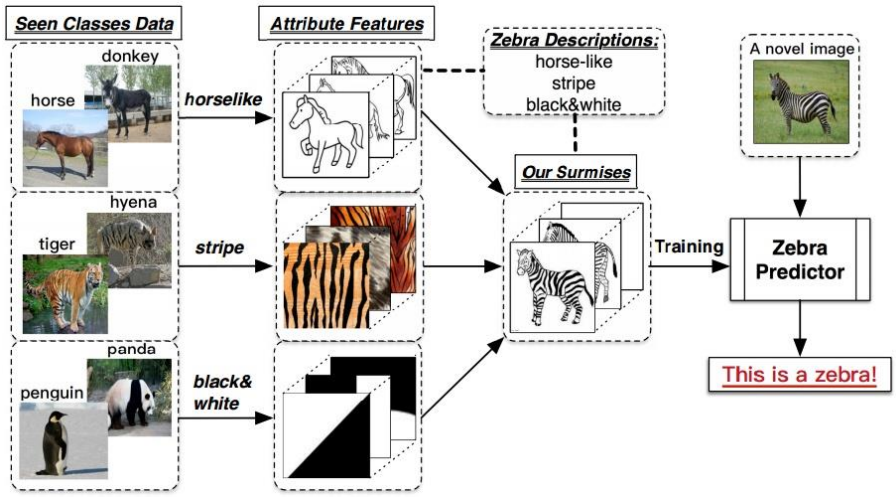


图 13.29 斑马案例

零样本学习就是将上述的推理过程抽象为通过已知信息加上辅助信息进而推断出新出现对象的类别的过程。因此，推理过程中已知的信息（马，老虎，熊猫）为训练集，辅助信息（马的外形、老虎的条纹、熊猫的颜色）为训练集与测试集相关联的语义信息，新出现对象（斑马）为测试集。

### 13.6.1 算法说明

零样本学习主要涉及三类数据：已知类、未知类和辅助信息。其中已知类是模型训练时用到的带类别标签的图像。未知类是模型测试、训练时不知道类别标签的图像。辅助信息是对已知类和未知类图像的描述、语义属性或者词嵌入等信息，该信息充当了已知类和未知类之间的桥梁。


图 13.30 数据集示意图

如图 13.30 所示，数据的集合由图像 x、标签 y 和辅助信息组成，辅助信息存储为一个由 5 个元素 (尾巴，喙，羽毛，胡须，毛茸茸) 组成的向量。猫有尾巴，而且毛茸茸，因此在向量中，对应于尾巴和毛茸茸的位子为 1，而在其他位置为 0。

S 表示已知类的数据集，U 表示未知类的数据集。集合$X^S$表示已知类的图像集，集合$Y^S$表示已知类的类别标签集，集合$A^S$已知类的属性集。而集合$X^U$表示未知类的图像集，集合$Y^U$表示未知类的类别标签集，集合$A^U$未知类的属性集。

基于嵌入的方法的主要目标是使用投影函数将图像特征和语义属性映射到一个公共的嵌入空间，该投影函数是使用深度网络学习的。共同的嵌入空间可以是视觉特征空间、语义空间或重新学习的中介空间。大多数基于嵌入的方法都使用语义空间作为公共嵌入空间。

在训练期间，目标是使用来自已知类的数据学习从视觉空间（即图像特征）到语义空间（即词向量/语义嵌入）的投影函数。由于神经网络可以用作函数逼近器，因此投影函数自然可以选用深度网络来学习。

在测试阶段，将未知类图像特征输入到前面训练好的网络里，获得相应的语义嵌入。然后在语义属性空间中进行最近邻搜索，以找到与网络输出最接近的匹配项。最后，将与最接近的语义嵌入相对应的标签预测为输入图像特征的输出标签。

图 13.30 展示的是基于嵌入的零样本学习方法的流程框图。首先将输入图像通过特征提取器网络（通常是深度神经网络），以获取图像的 N 维特征向量。此特征向量充当我们主投影网络的输入，而主投影网络又输出 D 维输出向量。目的是学习投影网络的权重/参数，以便从视觉空间中的 N 维输入映射到语义空间

中的 D 维输出。损失函数用于度量 D 维输出与真实语义属性之间的误差。然后就是训练网络的权重，使 D 维输出尽可能接近真实语义属性。

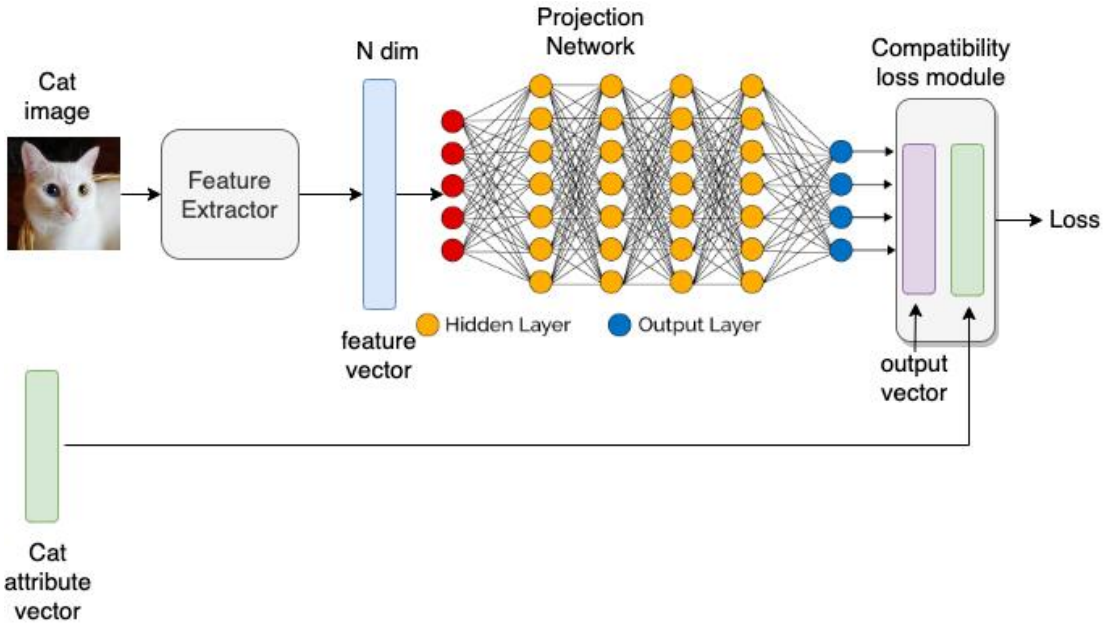


图 13.31 基于嵌入的零样本学习方法示意图

Top-1 准确率是图像识别模型常用评估指标。但零样本识别模型通常会使用每类 Top-1 的平均准确率，即分别找到每个类别的识别准确率，然后在所有类别上取平均值。

### 13.6.2 理论基础与基本模型

在零样本学习问题中，测试类与训练类之间相互互斥，我们不能简单地从有标签训练数据集中学习测试类的分类器。为了对这些未知测试类进行预测，我们必须在未知类和已知类之间引入耦合关系，从而建立两者之间的语义关联，这样我们就可以从已知类的训练数据集中抽取到相关知识，并使用这些知识进行未知类的预测。因此，我们可以在图片特征空间与物体标签空间中引入共享的中间层语义嵌入空间，从而将共享的概念或知识从已知类迁移到未知类中。

这样，每个物体类不再仅仅被视为一个互相之间相互独立的原子标签，而是用语义嵌入空间中的一个语义向量表示（也称类原型“class prototype”），这些语义向量刻画了类与类之间的语义关联关系。基于语义嵌入空间，我们可以首先使用已知类的训练数据学习图片特征与类原型之间的关系，然后使用该关系对测试数据进行两步法预测，即首先预测测试数据对应的语义向量，其次找到与该语义

向量最匹配的类，这个类就是我们要预测的未知类。正式地，我们使用$K \in \mathbb{R}^m$表示 m 维语义嵌入空间，分类器$l:\ X \to Y_{te}$可表示为以下形式:

$$l = g(f(\cdot))$$

$$f: X \to K$$

$$g: K \to Z$$

其中函数$f$通常由训练数据集学习而来，函数$g$可以是空间$K$中的最近邻算法，也可以是基于其他更复杂的相似度的分类方法。

基于语义嵌入空间$K$，根据未知类数据在零样本学习模型中的使用情况，我们主要有两种类型的零样本学习方法，即归纳式（inductive）零样本学习方法和直推式（transductive）零样本学习方法。

### 13.6.3 归纳式零样本学习

归纳式零样本学习方法假设在模型训练过程中函数$f$的学习只使用训练数据，而且在测试过程中，函数$g$以并行方式对所有测试数据进行类标签预测，即所有测试数据的标签预测过程是相互独立的。任意测试数据$x_i$的标签$y_i$预测为：

$$y_i = l(x_i) = g(f(x_i))$$

由于其具有良好的灵活性和可扩展性，归纳式学习方法是目前零样本学习问题中应用最广泛的一种解决思路。其中，最经典的学习模型是 Christoph 等人在 2009 年提出的直接属性预测模型（direct attribute prediction，简称为 DAP）和间接属性预测模型（indirect attribute prediction，简称为 IAP），其分别采用不同的方式构造图片特征空间与语义嵌入空间的关系函数$f$。作为零样本学习问题的基本解决方案，DAP 和 IAP 在很大程度上启发了后续的基于语义嵌入空间的零样本学习方法。

（1） DAP 模型

DAP 模型使用训练数据集直接学习图片特征到属性的映射，如图 13.32 所示。

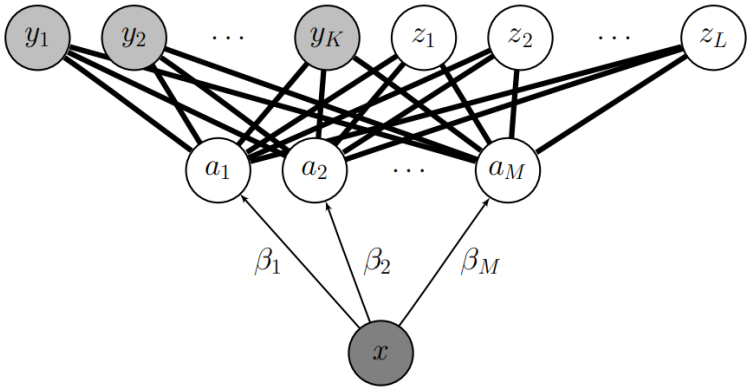


图 13.32 直接属性预测（DAP）示例图

假设我们有 $K$ 个已知类$y_1,\cdots,y_k$，$L$ 个未知类$z_1,\cdots,z_L$和 M 个语义属性$a_1,\cdots,a_M$，我们使用 0/1 值表示属性与物体类之间的关系，0 表示物体类不具备该属性，1 表示物体类具有该属性，每个类可表示为属性空间中的二值向量$a^y$和$a^z$。这样，我们可以使用分类器建模图片特征与属性之间的关系。对于每个属性$a_m$，我们使用训练数据学习概率分类器$p(a_m|x)$。在测试时，我们使用公式计算每个未知类的后验概率，

$$p(z|x)=\sum_{a\in\{0,1\}^M}p(z|a)p(a|x)=\frac{p(z)}{p(a^z)}\prod_{m=1}^{M}p(a_m^z|x)$$

其中$p(z)$为每个未知类的先验概率。我们将数据$x$分类为具有最大后验概率的类，即

$$l(x)=argmax_{l=1,\cdots,L}p(z=l|x)=argmax_{l=1,\cdots,L}\prod_{m=1}^{M}p(a_m^z|x)$$

此外，我们也可以使用连续值的属性向量来表示每个类中具有多大强度的属性，在这种情况下，我们使用回归模型来预测函数$f:X\to K$。

（2） IAP 模型

与 DAP 不同，IAP 模型通过使用已知类的类标签间接学习图片特征到属性的映射，如图 13.33 所示。首先，我们使用传统的有监督学习方法为每个已知类$y_k$学习概率分类器$p(y_k|x)$。其次，我们使用已知的属性向量$a^y$来预测每个测试数据的属性，即

$$p(a_m|x) = \sum_{k=1}^{K} p(a_m|y_k)\, p(y_k|x)$$

与 DAP 模型一样，我们使用公式来预测未知类标签。

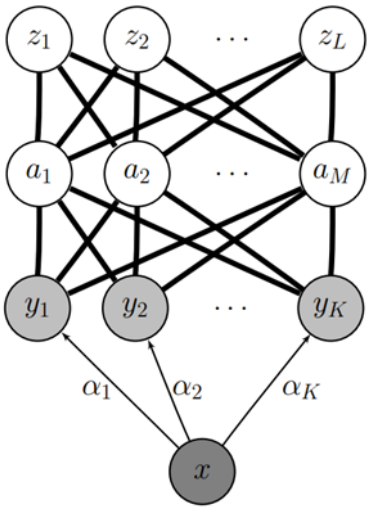


图 13.33 直接属性预测(DAP)示例图

### 13.6.4 直推式零样本学习

与归纳式方法不同，直推式零样本学习方法允许在分类器 $l$ 的学习过程中同时使用有标签训练数据和无标签测试数据。在基于语义嵌入空间的零样本学习方法中，我们有以下两种应用方式:

（1）与上文中的半监督学习类似，一方面，测试数据可以单独用在函数 $f$ 的学习过程中，从而提高 $f$ 在未知类上的泛化能力或迁移能力；另一方面，测试数据也可用来联合优化 $l$ 通过在训练过程中迭代优化 $f$ 和 $g$，我们可以一次性得到所有测试数据 $\{x_j\}_{j=1}^{N_u}$ 的标签 $\{y_j\}_{j=1}^{N_u}$，即

$$\{y_j\}_{j=1}^{N_u} = l\left(\{(x_i, k_i, y_i)\}_{i=1}^{N}, \{x_j\}_{j=1}^{N_u}\right), y_j \in Z$$

其中 $k_i \in K$ 是训练数据 $x_i$ 在空间 $K$ 中的语义嵌入向量，$y_i$ 是 $x_i$ 的类标签。

（2）在测试过程中，所有测试数据同时用于函数 $g$ 中，通过使用整个测试数据集的结构信息，从而一次性预测所有数据的类标签。也就是说，测试数据的类标签预测过程之间不再是相互独立的，而是彼此之间相互影响。所有测试数据 $\{x_j\}_{j=1}^{N_u}$ 的标签 $\{y_j\}_{j=1}^{N_u}$ 预测为：

$$\{y_j\}_{j=1}^{N_u} = g\left(f(\{x_j\}_{j=1}^{N_u})\right)$$

## 13.7 总结与展望

小样本学习旨在弥合人工智能和人类学习之间的差距。它可以学习新的任务，只包含几个例子与监督信息通过合并先验知识。FSL 作为人工智能的测试平台，使罕见案例的学习成为可能，或有助于减轻在工业应用中收集大规模监督数据的负担。

小样本学习的核心问题是不可靠的经验风险最小化，这使得小样本任务很难学习。理解核心问题有助于根据不同作品如何使用先验知识解决核心问题，具体分为数据、模型和算法。

- 数据增加了 FSL 的监督经验，
- 模型限制了 FSL 的假设空间更小，
- 算法改变了在给定假设空间中搜索最优假设的策略。

由于样本稀少，需要小样本学习方法来减少数据收集工作和计算成本，或者作为模仿人类学习的垫脚石。因此，许多现实世界的应用程序都涉及小样本学习。计算机视觉是小样本算法最早的测试平台之一。小样本学习在机器人技术、自然语言处理和声学信号处理等领域也引起了广泛关注。

- **计算机视觉领域：**字符识别；图像分类；目标识别；字体样式转换；图像检索；目标跟踪等。
- **机器人技术领域：**机器人手臂运动的模拟学习；从几个演示中学习操纵动作等。
- **自然语言处理：**语句分析；翻译；句子填空；基于简短评论的情绪分类等。
- **声学信号处理：**识别口语；克隆音频；语音转换等。

现有的小样本学习方法通常使用单一模态（如图像、文本或视频）的先验知识。然而，尽管目前使用的模式中有几个示例，但可能存在另一种模式，其中监督样本非常丰富。对灭绝动物的研究就是一个例子。虽然这种动物物种可能只有有限数量的视觉例子，但在文本领域（如教科书或网页）可能有很多关于它的信息，因为人们往往特别注意这种罕见的类别。因此，来自多种模式的先验知识可以为补充视图提供先验知识。

在未来的研究中，一个很有前途的方向是考虑多模态信息在设计小样本方法

中的应用。

在前面的内容中，根据如何使用小样本问题中的先验知识，可以从数据、模型和算法的角度对小样本学习方法进行分类。这些角度中的每一个都方法可以改进。

基于元学习的小样本学习通过跨任务学习，可以以较小的推理代价快速适应新任务。然而，元学习中考虑的任务通常被假定为来自单个任务分布 p（T）。在实践中，我们可以有大量的任务，它们的任务相关性未知或难以确定。在这种情况下，直接从所有这些任务中学习会导致负迁移。因此，当**任务相关性未知，或相关性较差时**的元学习算法值得研究。

## 习题

1. MNIST 数据集是机器学习领域中非常经典的一个数据集,由 60000 个训练样本和 10000 个测试样本组成,每个样本都是一张 28 * 28 像素的灰度手写数字图片。请尝试使用 GAN 生成 MNIST 数据。

2. IMDB-WIKI 数据集包含超过 50 万张带有年龄和性别标签的人脸图像（https://data.vision.ee.ethz.ch/cvl/rrothe/imdb-wiki/ ）。请尝试对数据集进行微笑检测、情绪识别和性别分类的多任务学习，可以参考（https://github.com/truongnmt/multi-task-learning ）中的方法。

3. MiniImageNet 包含 100 类共 60000 张彩色图片，其中每类有 600 个样本。地址：https://few-shot.yyliu.net/miniimagenet.htm 跟踪了关于该数据集的前沿算法，请尝试使用元学习的方法对数据集进行分类。

4. Omniglot 数据集包含来自 5050 个不同字母的 16231623 个不同手写字符。每一个字符都是由 2020 个不同的人通过亚马逊的 Mechanical Turk 在线绘制的（数据集和样例地址：https://github.com/brendenlake/omniglot ）。请尝试使用小样本相关算法，尽可能提升模型分类的准确率。

5. AWA 数据集包含 50 个动物类别，85 个属性，数据集下载地址：http://cvml.ist.ac.at/AwA2/AwA2-data.zip ，请尝试使用零样本学习方法对数据集进行分类。

6. SUN 数据集包含 131067 个图像，由 908 个场景类别和 4479 个物体类别组成，其中背景标注的物体有 313884 个。数据集下载地址：http://groups.csail.mit.edu/vision/SUN/ ，请尝试使用零样本学习方法对数据集进行分类。

7. CIFAR-10 数据集共有 60000 张彩色图像，这些图像是 32*32，分为 10 个类，每类 6000 张图，请尝试使用主动学习的方法对数据集进行分类，提高分类准确率。

8.请尝试采用不同的主动学样本寻找策略（RS 策略、LC 策略、BT 策略），对 MNIST 数据集进行分类。（RS 策略：随机选择，作为对照。LC 策略：寻找分类器最没有信心的预测样本，即预测的最可能类别的概率也很低的样本。BT 策略：寻找分类器最“左右为难”的预测样本，即预测的最可能的两个类别的概率很接近的样本。）