前　言

本书解释元学习的基本原理，并帮助你理解“学会学习”（learning to learn）的概念。你将学习各种单样本学习算法，例如孪生网络（siamese network）、原型网络（prototypical network）、关系网络（relation network）和记忆增强网络（memory-augmented network），并在 TensorFlow 与 Keras 中实现它们；了解先进的元学习算法，如模型无关元学习（model-agnostic meta learning，MAML）、Reptile 和元学习的上下文适应（context adaptation via meta learning，CAML）；探索如何使用元随机梯度下降法（meta stochastic gradient descent，Meta-SGD）来快速学习，以及如何使用元学习来进行无监督学习。

本书读者

本书主要写给那些想了解元学习并将其作为先进方法来训练机器学习模型的机器学习爱好者、人工智能研究人员和数据科学家。读者应当具备机器学习实战经验和坚实的 Python 编程知识。

本书内容

第 1 章，元学习简介，帮助你理解什么是元学习，并介绍它的不同类型。我们将研究元学习如何通过学习少量数据点来进行少样本学习，然后熟悉梯度下降（gradient descent）以及少样本学习的模型优化。

第 2 章，使用孪生网络进行人脸识别与音频识别，首先解释什么是孪生网络，以及如何在单样本学习中使用孪生网络，然后研究孪生网络的架构及其一些应用，以及如何使用孪生网络来建立人脸识别模型与音频识别模型。

第 3 章，原型网络及其变体，解释什么是原型网络以及如何在少样本学习中使用它。我们将建立一个对 omniglot 字符集进行分类的原型网络，并学习原型网络的不同变体，如高斯原型网络和半原型网络。

第 4 章，使用 TensorFlow 构建关系网络与匹配网络，帮助你理解关系网络的架构，及其在单样本、少样本和零样本学习中的用途。我们将介绍如何使用 TensorFlow 构建关系网络并将学习匹配网络及其架构，还将探索完整的上下文嵌入，以及如何使用 TensorFlow 构建匹配网络。

第 5 章，记忆增强神经网络，介绍神经图灵机（NTM）的概念，以及 NTM 如何利用外部存储空间存储和检索信息。我们将研究 NTM 中使用的不同寻址机制，并了解记忆增强神经网络（MANN）及其与 NTM 架构的区别。

第 6 章，MAML 及其变种，介绍一种流行的元学习算法——MAML。我们将探索什么是 MAML，如何在监督和强化学习环境中使用它，以及如何从头构建它。此外还会学习对抗性元学习和 CAML，其中后者用于元学习中的快速上下文适应。

第 7 章，Meta-SGD 和 Reptile，解释如何使用 Meta-SGD 学习梯度下降算法的所有内容，如初始权重、学习率和更新方向。我们将了解如何从头开始构建 Meta-SGD；学习 Reptile 算法，它可以被视为 MAML 的改进版；掌握如何使用 Reptile 算法进行正弦曲线回归。

第 8 章，梯度一致作为优化目标，介绍如何在元学习环境中使用梯度一致作为优化目标。我们将学习什么是梯度一致及其如何增强元学习算法，还将学习如何从头构建梯度一致算法。

第 9 章，新进展与未来方向，首先解释什么是任务无关元学习；然后介绍如何在模仿学习中使用元学习，以及如何使用 CACTUs 算法将 MAML 应用到无监督学习中；最后在概念空间中探索一种叫作“学会学习”的深度元学习算法。

如何充分利用本书

你需要安装下列软件：

- Python
- Anaconda
- TensorFlow
- Keras

下载示例代码

你可以用你的账户从 www.packtpub.com 下载已购买的 Packt 图书的示例代码文件。如果你是从其他地方购买的本书，可以访问 www.packtpub.com/support 并注册，我们将通过电子邮件把文件发送给你。[①]

你可以通过以下步骤下载本书示例代码：

(1) 在 www.packtpub.com 登录或注册；
(2) 选择 SUPPORT 标签；

① 示例代码也可以从本书图灵社区页面（https://www.ituring.com.cn/book/2697）获取。——编者注

(3) 点击 Code Downloads & Errata；
(4) 在 Search 框输入书名并遵循屏幕上的指示。

下载完文件后，确保使用如下软件的最新版本来解压或提取文件夹。

- Windows 上建议使用 WinRAR/7-Zip。
- Mac 上建议使用 Zipeg/iZip/UnRarX。
- Linux 上建议使用 7-Zip/PeaZip。

本书的代码包也可在 GitHub 上获取，在 GitHub 网站搜索 Hands-On-Meta-Learning-with-Python 即可。如果代码更新了，也会在现有的 GitHub 仓库上更新。

GitHub 网站 PackPublishing 页面上也列出了我们所出版的各类图书和视频的代码包。欢迎查看！

排版约定

本书采用如下排版约定。

文本中的代码采用等宽字体，例如：“`read_image` 函数将一张图片作为输入，并返回一个 NumPy 数组。”

代码块的格式如下：

```
import re
import numpy as np
from PIL import Image
```

黑体字：表示新术语、重要的词，或界面上显示的内容。

此图标表示警告或重要信息。

此图标表示提示或诀窍。

保持联系

我们始终欢迎读者的反馈。

一般反馈：如果你对本书有任何疑问，请通过 questions@packtpub.com联系我们，并在邮件主题中注明书名。

勘误：虽然我们已经竭尽全力来确保本书内容的准确性，但错误在所难免。如果你发现书中有错，请告知我们，我们将不胜感激。请访问 www.packtpub.com/submit-errata，选择图书，点击勘误提交表单链接，并输入详细信息。

反盗版：如果你在网上发现以任何形式复制我们作品的非法行为，请立即将地址或网站名称告知我们，我们将不胜感激。请联系 copyright@packtpub.com提供有盗版嫌疑的链接。

成为作者：如果你是某个领域的专家，并且有兴趣编写图书，请访问 authors.packtpub.com。

评论

请留下你的评论。阅读并使用本书之后，为何不在购买网站上发表评论呢？其他读者可以参考你的评价来做出购买决定，Packt 出版社可以了解你对我们产品的看法，作者也可以看到你对本书的反馈。谢谢！

想了解更多关于 Packt 的信息，请访问 packt.com。

电子书

扫描如下二维码，即可购买本书中文版电子版。

第 1 章 元学习简介

元学习是目前人工智能领域最有前景和最热门的研究方向之一，被视为实现**通用人工智能**（Artificial General Intelligence，AGI）的基础。本章，我们将学习什么是元学习，以及为什么元学习是目前人工智能领域最令人兴奋的研究方向；了解什么是少样本、单样本和零样本学习，以及它们在元学习中的应用；学习不同类型的元学习技术；探索“通过梯度下降来学习如何通过梯度下降来学习”（learning to learn by gradient descent by gradient descent）的概念，以及如何使用元学习器学习梯度下降优化；了解如何将优化作为一个模型用于少样本学习，并看到如何在少样本学习中将元学习器用作优化算法。

本章内容包括：

- 元学习；
- 元学习与少样本学习；
- 元学习的类型；
- 通过梯度下降来学习如何通过梯度下降来学习；
- 少样本学习中的优化模型。

1.1 元学习

元学习是目前人工智能领域中一个令人振奋的研究方向。随着大量研究论文的发表和研究进展的取得，元学习在人工智能领域取得了重大突破。在开始探讨元学习之前，先来了解一下当前的人工智能模型的工作原理。

近年来，随着生成对抗网络和胶囊网络等优秀算法的出现，深度学习得到了快速的发展。但问题是，深度神经网络需要大规模的训练集来训练模型。当数据点很少时，它会突然失效。假设我们训练了一个深度学习模型来执行任务 A。当我们有一个和 A 紧密相关的新任务 B 时，就不能使用相同的模型，而是需要从零开始为任务 B 训练模型。因此，虽然每个任务可能是相关的，但都需要从零开始训练模型。

深度学习真的是真正的人工智能吗？答案是否定的。我们人类是如何学习的呢？我们将学到的东西归纳为多个概念并从中学习。不过目前的学习算法只能处理一项任务。这就是元学习的用武之地。元学习能够生成一个通用的人工智能模型来学习执行各种任务，而无须从零开始训练它们。我们可以用很少的数据点来训练元学习模型去完成各种相关的任务，因此对于一个新任务，元学习模型可以利用之前从相关任务中获得的知识，无须从零开始训练。许多研究人员和科学家认为，元学习可以让我们更接近 AGI。接下来的几节将阐释元学习模型如何学会学习的过程。

元学习与少样本学习

少样本学习（few-shot learning）或 ***k* 样本学习**（*k*-shot learning）指的是利用较少的数据点进行学习，其中 *k* 表示数据集各个类别中数据点的数量。假设我们正在对狗和猫的图像进行分类。如果只有 1 张狗的图像和 1 张猫的图像，那就叫作**单样本学习**（one-shot learning）。也就是说，对于每个类别，我们只从 1 个数据点学习。如果有 10 张狗的图像和 10 张猫的图像，那就叫作 10 样本学习。因此，*k* 样本学习中的 *k* 表示每个类别中数据点的数量。还有一种**零样本学习**（zero-shot learning），即所有类别中都没有数据点。等等，如果没有数据点，那可怎么学习呢？在这种情况下，我们虽然没有数据点，但是拥有关于每个类别的元信息，并将从中学习。因为数据集中有两个类别，即狗和猫，所以可以称之为双（$n=2$）类别 *k* 样本学习——*n* 表示数据集中类别的数量。

为了使模型从少量的数据点中学习，我们将用同样的方法训练它们。因此，当有一个数据集 *D* 时，我们从数据集中的每个类别中挑选几个数据点，称之为**支撑集**（support set）。同样，从每个类别中挑选一些不同的数据点，称之为**查询集**（query set）。于是，我们用一个支撑集训练模型，并用查询集来测试模型。我们以一种**阶段式的方式**（episodic fashion）训练模型，即在每个阶段中，从数据集 *D* 中抽取少量数据点，准备支撑集和查询集，并使用支撑集进行训练，使用查询集进行测试。因此，在多个阶段后，模型将学会如何从较小的数据集中学习。接下来的章节会对此进行更详细的探讨。

1.2　元学习的类型

元学习有多种分类标准，例如寻找最优的权重集与学习优化器。我们将元学习分为以下 3 类：

- 学习度量空间
- 学习初始化
- 学习优化器

1.2.1　学习度量空间

在基于度量的元学习场景中，我们将学习合适的度量空间。假设我们想学习两幅图像之间的相似性。在基于度量的场景中，我们使用一个简单的神经网络从两幅图像中提取特征，并通过计

算两幅图像特征之间的距离找到相似性。这种方法被广泛应用于数据点较少的少样本学习中。接下来的章节将介绍基于度量的学习算法，如孪生网络、原型网络和关系网络。

1.2.2 学习初始化

在这个方法中，我们尝试学习最优的初始参数值。这是什么意思呢？假设我们正在构建一个神经网络来对图像进行分类。我们首先初始化随机权重，计算损失，并通过梯度下降来最小化损失。因此，我们将通过梯度下降找到最优权重，使损失最小。如果不随机初始化权重，而是用最优值或者接近最优值的值来初始化权重，那么就可以更快地收敛，并快速学习。接下来的章节将介绍如何通过 MAML、Reptile 和 Meta-SGD 等算法来精确地找到这些最优的初始权重。

1.2.3 学习优化器

在这个方法中，我们尝试学习优化器。一般如何优化神经网络呢？答案是通过基于大数据集的训练来优化神经网络，并使用梯度下降来最小化损失。但是在少样本的学习场景中，梯度下降失效了，因为我们的数据集较小。因此，在这种情况下，我们将学习优化器本身。我们将有两个网络：试图学习的基网络和优化基网络的元网络。后面的章节会详细探讨。

1.3 通过梯度下降来学习如何通过梯度下降来学习

现在，我们来看一个有趣的元学习算法——通过梯度下降来学习如何通过梯度下降来学习。这个名字是不是有点吓人？实际上，它是最简单的元学习算法之一。我们知道，在元学习中，目标是学习学习的过程。一般如何训练神经网络呢？答案是通过梯度下降来计算损失和最小化损失以训练网络。因此，我们使用梯度下降法来优化模型。如果不使用梯度下降，我们能自动学习这个优化过程吗？

但是应该如何学习呢？我们用**递归神经网络**（Recurrent Neural Network，RNN）代替传统的梯度下降优化器。这是如何实现的呢？如何用 RNN 代替梯度下降法？如果你仔细观察梯度下降的行为，就会发现，它基本上是一个从输出层到输入层的更新序列。我们将这些更新存储在一个状态中，这样就可以使用 RNN 并将更新存储在 RNN 单元中。

该算法的主要思想是用 RNN 代替梯度下降法。但问题是 RNN 如何学习？如何优化 RNN？为了优化 RNN，我们使用梯度下降法。简而言之，**我们正在学习通过 RNN 来执行梯度下降，而这个 RNN 是通过梯度下降来优化的。**这就是该算法名称的由来。

我们称 RNN 为优化器，称基网络（base network）为优化对象。假设有一个由参数 θ 影响的模型 f。需要找到这个最优参数 θ，以将损失最小化。一般情况下，通过梯度下降法来寻找最优参数，但现在用 RNN 来寻找最优参数。因此，RNN（优化器）找到了最优参数并将其发送给优

化对象（基网络）。优化对象使用这个参数，计算损失，并将损失发送给 RNN。基于该损失，RNN 通过梯度下降优化自身，并更新模型参数 θ。

感到困惑吗？请看图 1-1：优化对象（基网络）是通过优化器（RNN）优化的。优化器将更新后的参数（即权重）发送给优化对象，优化对象使用这些权重计算损失，并将损失发送给优化器。基于损失，优化器通过梯度下降来改进自身。

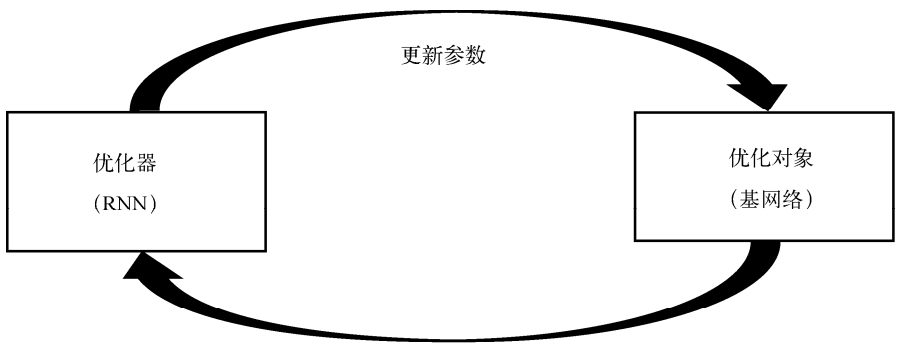

图　1-1

假设基网络（优化对象）以 θ 作为参数，而 RNN（优化器）以 ϕ 作为参数。优化器的损失函数是什么？我们知道优化器（RNN）用于减少优化对象（基网络）的损失。因此，优化器的损失是优化对象的平均损失，它可以表示为

$$L(\phi) = \mathbb{E}_f[f(\theta(f,\phi))]$$

怎样才能把损失降到最低呢？通过梯度下降找到合适的 ϕ 来最小化这种损失。RNN 接受什么作为输入？它又输出什么呢？优化器，也就是 RNN，将优化对象的梯度 ∇_t 以及它的上一个状态 h_t 作为输入，并返回输出——可以最小化优化器损失的更新 g_t。我们用函数 m 来表示 RNN：

$$(g_t, h_{t+1}) = m(\nabla_t, h_t, \phi)$$

以上方程的参数解释如下：

- ∇_t 是模型 f（优化对象）的梯度，即 $\nabla_t = \nabla_t f(\theta_t)$；
- h_t 是 RNN 的隐藏状态；
- ϕ 是 RNN 的参数；
- 输出 g_t 和 h_{t+1} 分别是（提供给优化器的）更新与 RNN 的下一个状态。

于是，可以使用 $\theta_{t+1} = \theta_t + g_t$ 来更新模型参数值。

如图 1-2 所示，优化器 m 在时间 t 处，以隐藏状态 h_t 和相对于 θ_t 的梯度 ∇_t 为输入，计算出 g_t 并将其发送到优化对象，再与 θ_t 相加，成为下一步更新的 θ_{t+1}。

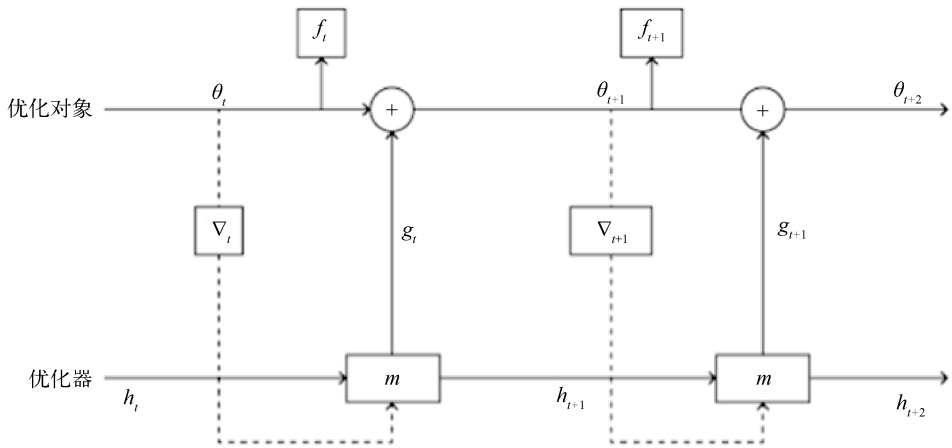

图 1-2

由此，我们通过梯度下降学会了梯度下降优化。

1.4 少样本学习的优化模型

我们知道，少样本学习基于较少的数据点，那么如何将梯度下降应用到少样本学习中呢？在少样本学习中，梯度下降会由于数据点非常少而突然失效。梯度下降优化需要更多的数据点来达到收敛和损失最小化。因此，在少样本学习中需要一种更好的优化技术。假设有一个由参数 θ 影响的模型 f。我们用一些随机值来初始化参数 θ，并尝试使用梯度下降法找到最优值。让我们回忆一下梯度下降的更新方程：

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

以上方程的参数解释如下：

- θ_t 是更新参数；
- θ_{t-1} 是上一步的参数值；
- α_t 是学习率；
- $\nabla_{\theta_{t-1}} L_t$ 是相对于 θ_{t-1} 的损失函数的梯度。

梯度下降的更新方程是不是看起来很熟悉？是的，你猜对了，它类似于长短期记忆网络（LSTM）的细胞状态更新方程，可以写成：

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

可以将 LSTM 细胞更新方程与梯度下降完全对应起来，设 $f_t = 1$，可得：

$$\begin{aligned} c_{t-1} &= \theta_{t-1} \\ i_t &= \alpha_t \\ \tilde{c}_t &= \nabla_{\theta_{t-1}} L_t \end{aligned}$$

因此，在少样本学习中，可以使用 LSTM 而非梯度下降作为优化器。LSTM 是元学习器，它将学习用于训练模型的更新规则。因此，我们使用两个网络：一个是基学习器，它学会执行任务；另一个是元学习器，它试图找到最优的参数。这是如何实现的呢?

我们知道，LSTM 使用遗忘门（forget gate）来丢弃存储器中不需要的信息，它可以表示为

$$f_t = \sigma(w_f \cdot [h_{t-1}, x_t] + b_f)$$

这个遗忘门在我们的优化场景中有什么用呢？假设我们处在一个损失很大，梯度接近于零的位置。怎样才能摆脱这种局面呢？在这种情况下，可以收缩模型的参数，并忘记其前一个值的某些部分。我们可以使用遗忘门来实现这一点，它以当前参数值 θ_{t-1}、当前损失 L_t、当前梯度 $\nabla_{\theta_{t-1}}$ 以及前一个遗忘门作为输入。它可以表示为

$$f_t = \sigma(w_f \cdot [\theta_{t-1}, L_t, \nabla_{\theta_{t-1}}, f_{t-1}] + b_f)$$

下面来看看输入门（input gate）。我们知道 LSTM 中的输入门是用来决定更新什么值的，它可以表示为

$$i_t = \sigma(w_i \cdot [h_{t-1}, x_t] + b_i)$$

在少样本学习中，可以使用这个输入门来调整学习率，从而在防止发散的同时快速学习：

$$i_t = \sigma(w_i \cdot [\theta_{t-1}, L_t, \nabla_{\theta_{t-1}}, i_{t-1}] + b_i)$$

因此，元学习器在多次更新之后得到了 i_t 与 f_t 的最优值。

可是，这是如何运作的呢?

假设有一个由 θ 影响的基网络 M、由 ϕ 影响的 LSTM 元学习器 R，以及数据集 D。我们将数据集分割为训练集 D^{train} 和测试集 D^{test}。首先随机初始化元学习器参数 ϕ。

在 T 次迭代中，随机从 D^{train} 中抽取数据点，计算损失以及相对于模型参数 θ 的损失梯度。将这个梯度、损失和元学习器参数 ϕ 提供给元学习器。元学习器 R 会返回细胞状态 c_t，然后在时间 t 将基网络 M 的参数 θ_t 更新为 c_t。重复 N 次，如图 1-3 所示。

for $t = 1, \cdots, T$ do

$X_t, Y_t \longleftarrow D^{\text{train}}$中的一批随机样本

$\text{Loss}_t \longleftarrow L(M(X_t; \theta_{t-1}), Y_t)$

细胞状态（c_t） $\longleftarrow R((\nabla_{\theta_{t-1}} \text{Loss}_t, \text{Loss}_t), \phi)$

$\theta_t \longleftarrow c_t$

end for

图 1-3

因此，经过 T 次迭代，我们会得到一个最优参数 θ_T。不过如何检查 θ_T 的性能并更新元学习器参数呢？使用测试集和参数 θ_T 计算测试集的损失。然后，计算相对于元学习器参数 ϕ 的损失梯度，并更新 ϕ，如图 1-4 所示。

$X, Y \longleftarrow D^{\text{test}}$

$\text{Loss}_{\text{test}} \longleftarrow L(M(X; \theta_T), Y)$

用$\nabla_{\phi}\text{Loss}_{\text{test}}$更新$\phi$

图 1-4

迭代 n 次，并更新元学习器。完整的算法如图 1-5 所示。

$\phi_0 \longleftarrow$ 随机初始化

for $d = 1, \cdots, n$ do

$D^{\text{train}}, D^{\text{test}} \longleftarrow$ 数据集 D 中的随机样本

$\theta_0 \longleftarrow c_0$

for $t = 1, \cdots, T$ do

$X_t, Y_t \longleftarrow D^{\text{train}}$中的一批随机样本

$\text{Loss}_t \longleftarrow L(M(X_t; \theta_{t-1}), Y_t)$

细胞状态（c_t） $\longleftarrow R((\nabla_{\theta_{t-1}} \text{Loss}_t, \text{Loss}_t), \phi_{d-1})$

$\theta_t \longleftarrow c_t$

end for

$X, Y \longleftarrow D^{\text{test}}$

$\text{Loss}_{\text{test}} \longleftarrow L(M(X; \theta_T), Y_t)$

用$\nabla_{\theta_{t-1}} \text{Loss}_{\text{test}}$更新$\phi_d$

end for

图 1-5

1.5　小结

我们首先学习了什么是元学习，以及如何在元学习中使用单样本学习、少样本学习和零样本学习，并了解到支撑集和查询集与训练集和测试集十分相似，只不过每个类别中都有 k 个数据点，还知道了 n 类别 k 样本学习的含义。然后，我们了解了不同类型的元学习技术，探讨了通过梯度下降来学习如何通过梯度下降来学习，并知道了如何使用 RNN 作为优化器优化基网络。之后，我们将优化看作一个用于少样本学习的模型，并将 LSTM 用作元学习器在少样本学习中进行优化。

1.6　思考题

(1) 什么是元学习？
(2) 什么是少样本学习？
(3) 什么是支撑集？
(4) 什么是查询集？
(5) 基于度量的学习又称为什么？
(6) 如何在元学习中进行训练？

1.7　延伸阅读

- 通过梯度下降来学习如何通过梯度下降来学习：参见 Marcin Andrychowicz、Misha Denil、Sergio Gomez 等人的文章 *Learning to Learn by Gradient Descent by Gradient Descent*。
- 少样本学习场景下的优化模型：参见 Sachin Ravi 和 Hugo Larochelle 的文章 *Optimization as a Model for Few-shot Learning Setting*。