模型独立的学习方式

集成学习（Ensemble Learning）

给定一个学习任务，假设输入$\mathbf{x}$和输出$\mathbf{y}$的真实关系为$\mathbf{y}=h(\mathbf{x})$。对于M个不同的模型$f_1(\mathbf{x}),\cdots,f_M(\mathbf{x})$，每个模型的期望错误为

$$\mathcal{R}(f_m)=\mathbb{E}_{\mathbf{x}}\Big[\big(f_m(\mathbf{x})-h(\mathbf{x})\big)^2\Big] \tag{10.1}$$

$$=\mathbb{E}_{\mathbf{x}}\Big[\epsilon_m(\mathbf{x})^2\Big], \tag{10.2}$$

其中$\epsilon_m(\mathbf{x})=f_m(\mathbf{x})-h(\mathbf{x})$为模型$m$在样本$\mathbf{x}$上的错误。

那么所有的模型的平均错误为

$$\bar{\mathcal{R}}(f)=\frac{1}{M}\sum_{m=1}^{M}\mathbb{E}_{\mathbf{x}}[\epsilon_m(\mathbf{x})^2]. \tag{10.3}$$

通过某种策略将多个模型集成起来，通过群体决策来提高决策准确率。集成学习首要的问题是如何集成多个模型。比较常用的集成策略有直接平均、加权平均等。

定理 10.1： 对于M个不同的模型$f_1(\mathbf{x}),\cdots,f_M(\mathbf{x})$，其平均期望错误为$\bar{\mathcal{R}}(f)$。基于简单投票机制的集成模型$F(\mathbf{x})=\frac{1}{M}\sum_{m=1}^{M}f_m(\mathbf{x})$，$F(\mathbf{x})$的期望错误在$\frac{1}{M}\bar{\mathcal{R}}(f)$和$\bar{\mathcal{R}}(f)$之间。

$$\mathcal{R}(F)=\mathbb{E}_{\mathbf{x}}\Big[\Big(\frac{1}{M}\sum_{m=1}^{M}f_m(\mathbf{x})-h(\mathbf{x})\Big)^2\Big] \tag{10.5}$$

$$=\frac{1}{M^2}\mathbb{E}_{\mathbf{x}}\Big[\Big(\sum_{m=1}^{M}\epsilon_m(\mathbf{x})\Big)^2\Big] \tag{10.6}$$

$$=\frac{1}{M^2}\mathbb{E}_{\mathbf{x}}\Big[\sum_{m=1}^{M}\sum_{n=1}^{M}\epsilon_m(\mathbf{x})\epsilon_n(\mathbf{x})\Big] \tag{10.7}$$

$$=\frac{1}{M^2}\sum_{m=1}^{M}\sum_{n=1}^{M}\mathbb{E}_{\mathbf{x}}\Big[\epsilon_m(\mathbf{x})\epsilon_n(\mathbf{x})\Big], \tag{10.8}$$

其中$\mathbb{E}_{\mathbf{x}}[\epsilon_m(\mathbf{x})\epsilon_n(\mathbf{x})]$为两个不同模型错误的相关性。如果每个模型的错误不相关，即$\forall m\neq n,\mathbb{E}_{\mathbf{x}}[\epsilon_m(\mathbf{x})\epsilon_n(\mathbf{x})]=0$。如果每个模型的错误都是相同的，则$\forall m\neq n,\epsilon_m(\mathbf{x})=\epsilon_n(\mathbf{x})$。并且由于$\epsilon_m(\mathbf{x})\geq 0,\forall m$，可以得到

$$\bar{\mathcal{R}}(f)\geq\mathcal{R}(F)\geq\frac{1}{M}\bar{\mathcal{R}}(f), \tag{10.9}$$

即集成模型的期望错误是大于等于所有模型的平均期望错误的$1/M$，小于等于所有模型的平均期望错误。 □

要求每个模型之间具备一定的差异性。并且随着模型数量的增多，其错误率也会下降，并趋近于 0。采取 Bagging 类和 Boosting 类两类方法。

Bagging 类方法 Bagging 类方法是通过随机构造训练样本、随机选择特

征等方法来提高每个基模型的独立性，代表性方法有 Bagging 和随机森林等。

Bagging（Bootstrap Aggregating）是一个通过不同模型的训练数据集的独立性来提高不同模型之间的独立性。我们在原始训练集上进行有放回的随机采样，得到 M 比较小的训练集并训练 M 个模型，然后通过投票的方法进行模型集成。

随机森林（Random Forest）是在 Bagging 的基础上再引入了随机特征，进一步提高每个基模型之间的独立性。在随机森林中，每个基模型都是一棵决策树。

Boosting 类方法 Boosting 类方法是按照一定的顺序来先后训练不同的基模型，每个模型都针对前序模型的错误进行专门训练。根据前序模型的结果，来调整训练训练样本的权重，从而增加不同基模型之间的差异性。Boosting 类方法是一种非常强大的集成方法，只要基模型的准确率比随机猜测好，就可以通过集成方法来显著地提高集成模型的准确率。Boosting 类方法的代表性方法有 AdaBoost 等

AdaBoost 算法

Boosting 类集成模型的目标是学习一个加性模型（additive model）。Boosting 类方法的关键是如何训练每个弱分类器$f_m(x)$以及对应的权重α_m，按照一定的顺序依次改变数据分布来训练每个弱分类器。在学习了第 m 个弱分类器后，增加其分错样本的权重，使得第 m+1 个弱分类器“更关注”于前面弱分类器分错的样本。这样增加每个弱分类器的差异，最终提升的集成分类器的准确率。

算法 10.1: 两类分类的 AdaBoost 算法

输入: 训练集 $\{(\mathbf{x}^{(n)}, y^{(n)})\}_{n=1}^{N}$，迭代次数 M

初始样本权重：$w_1^{(n)} \leftarrow \frac{1}{N}, \forall n \in [1, N]$;

for $m = 1 \cdots M$ **do**

  按照样本权重 $w_m^{(1)}, \cdots, w_m^{(N)}$，学习弱分类器 f_m;

  计算弱分类器 f_m 在数据集上的加权错误为 ϵ_m;

  计算分类器的集成权重：

$$\alpha_m \leftarrow \frac{1}{2} \log \frac{1 - \epsilon_m}{\epsilon_m};$$

  调整样本权重：

$$w_{m+1}^{(n)} \leftarrow w_m^{(n)} \exp\big(-\alpha_m y^{(n)} f_m(\mathbf{x}^{(n)})\big), \forall n \in [1, N];$$

end

输出: $F(\mathbf{x}) = \operatorname{sgn}\Big(\sum_{m=1}^{M} \alpha_m f_m(\mathbf{x})\Big)$

损失函数：

$$\mathcal{L}(F) = \exp\big(-yF(\mathbf{x})\big) \tag{10.11}$$

$$= \exp\Big(-y \sum_{m=1}^{M} \alpha_m f_m(\mathbf{x})\Big), \tag{10.12}$$

假设经过 $m-1$ 次迭代，得到

$$F_{m-1}(\mathbf{x}) = \sum_{t=1}^{m-1} \alpha_t f_t(\mathbf{x}), \tag{10.13}$$

则第 m 次迭代的目标是找一个 α_m 和 $f_m(\mathbf{x})$ 使得下面的损失函数最小。

$$\mathcal{L}(\alpha_m, f_m(\mathbf{x})) = \sum_{n=1}^{N} \exp\Big(- y^{(n)} \big(F_{m-1}(\mathbf{x}^{(n)}) + \alpha_m f_m(\mathbf{x}^{(n)})\big)\Big). \tag{10.14}$$

令 $w_m^{(n)} = \exp\big(- y^{(n)} F_{m-1}(\mathbf{x}^{(n)})\big)$，则损失函数可以写为

$$\mathcal{L}(\alpha_m, f_m(\mathbf{x})) = \sum_{n=1}^{N} w_m^{(n)} \exp\big(- \alpha_m y^{(n)} f_m(\mathbf{x}^{(n)})\big). \tag{10.15}$$

因为 $y, f_m(\mathbf{x}) \in \{+1, -1\}$，有

$$y f_m(\mathbf{x}) = 1 - 2I(y \neq f_m(\mathbf{x})), \tag{10.16}$$

将损失函数在 $f_m(\mathbf{x}) = 0$ 处进行二阶泰勒展开，有

$$\mathcal{L}(\alpha_m, f_m(\mathbf{x})) = \sum_{n=1}^{N} w_m^{(n)} \big(1 - \alpha_m y^{(n)} f_m(\mathbf{x}^{(n)}) + \frac{1}{2}\alpha_m^2\big) \tag{10.17}$$

$$\propto \alpha_m \sum_{n=1}^{N} w_m^{(n)} I\big(y^{(n)} \neq f_m(\mathbf{x}^{(n)})\big). \tag{10.18}$$

在求解出 $f_m(\mathbf{x})$ 之后，公式(10.15)可以写为

$$\mathcal{L}(\alpha_m, f_m(\mathbf{x})) = \sum_{y^{(n)} = f_m(\mathbf{x}^{(n)})} w_m^{(n)} \exp(-\alpha_m) + \sum_{y^{(n)} \neq f_m(\mathbf{x}^{(n)})} w_m^{(n)} \exp(\alpha_m) \tag{10.19}$$

$$\propto (1 - \epsilon_m) \exp(-\alpha_m) + \epsilon_m \exp(\alpha_m), \tag{10.20}$$

其中 ϵ_m 为分类器 $f_m(\mathbf{x})$ 的加权错误率，

$$\epsilon_m = \frac{\sum_{y^{(n)} \neq f_m(\mathbf{x}^{(n)})} w_m^{(n)}}{\sum_n w_m^{(n)}}. \tag{10.21}$$

求上式关于 α_m 的导数并令其为0，得到

$$\alpha_m = \frac{1}{2} \log \frac{1 - \epsilon_m}{\epsilon_m}. \tag{10.22}$$

自训练和协同训练

自训练（Self-Training）

自训练是首先使用标注数据来训练一个模型，并使用这个模型来预测无标注样本的标签，把预测置信度比较高的样本及其预测的伪标签加入训练集，然后重新训练新的模型，并不断重复这个过程。

自训练和密度估计中 EM 算法有一定的相似之处，通过不断地迭代来提高模型能力。但自训练的缺点是无法保证每次加入训练集的样本的伪标签是正确的。如果选择样本的伪标签是错误的，反而会损害模型的预测能力。因此，自训练最关键的步骤是如何设置挑选样本的标准。

算法 10.2: 自训练的训练过程

输入: 标注数据集 $\mathcal{L}=\{(\mathbf{x}^{(n)},y^{(n)})\}_{n=1}^{N}$;
无标注数据集 $\mathcal{U}=\{\mathbf{x}^{(m)}\}_{m=1}^{M}$;
迭代次数 T; 每次迭代增加样本数量 P;

1 **for** $t=1\cdots T$ **do**
2 根据训练集 $\mathcal{L}$，训练模型 f;
3 使用模型 f 对未标记数据集 $\mathcal{U}$ 的样本进行预测，选出预测置信度高的 P 个样本 $\mathcal{P}=\{(\mathbf{x}^{(p)},f(\mathbf{x}^{(p)}))\}_{p=1}^{P}$;
4 更新训练集:

$$\mathcal{L}\leftarrow\mathcal{L}\cup\mathcal{P},\qquad \mathcal{U}\leftarrow\mathcal{U}-\mathcal{P}.$$

5 **end**

输出: 模型 f

协同训练（Co-Training）

算法 10.3: 协同训练的训练过程

输入: 标注数据集 $\mathcal{L}=\{(\mathbf{x}^{(n)},y^{(n)})\}_{n=1}^{N}$;
无标注数据集 $\mathcal{U}=\{\mathbf{x}^{(m)}\}_{m=1}^{M}$;
迭代次数 T; 候选池大小 K; 每次迭代增加样本数量 $2P$;

1 **for** $t=1\cdots T$ **do**
2 根据训练集 $\mathcal{L}$ 的视角 V_1 训练训练模型 f_1;
3 根据训练集 $\mathcal{L}$ 的视角 V_2 训练训练模型 f_2;
4 从无标记数据集 $\mathcal{U}$ 上随机选取一些样本放入候选池 $\mathcal{U}'$，使得 $|\mathcal{U}'|=K$;
5 **for** $f\in f_1,f_2$ **do**
6 使用模型 f 预测候选池 $\mathcal{U}'$ 中的样本的伪标签;
7 **for** $p=1\cdots P$ **do**
8 根据标签分布，随机选取一个标签 y;
9 从 $\mathcal{U}'$ 中选出伪标签为 y，并且预测置信度最高的样本 $\mathbf{x}$;
10 更新训练集:

$$\mathcal{L}\leftarrow\mathcal{L}\cup\{(\mathbf{x},y)\},\qquad \mathcal{U}'\leftarrow\mathcal{U}'-\{(\mathbf{x},y)\}.$$

11 **end**
12 **end**
13 **end**

输出: 模型 f_1,f_2

协同算法要求两种视图是条件独立的。如果两种视图完全一样，则协同训练退化成自训练算法。

多任务学习（Multi-task Learning）

同时学习多个相关任务，让这些任务在学习过程中共享知识，利用多个任务之间的相关性来改进模型在每个任务的性能和泛化能力。通过利用包含在相关任务中的信息作为归纳偏置（Inductive Bias）来提高泛化能力。

如何设计多任务之间的共享机制：

- **硬共享模式**：让不同任务的神经网络模型共同使用一些共享模块（一般是低层）来提取一些通用特征，然后再针对每个不同的任务设置一些私有模块（一般是高层）来提取一些任务特定的特征。
- **软共享模式**：不显式地设置共享模块，但每个任务都可以从其它任务中“窃取”一些信息来提高自己的能力。窃取的方式包括直接复制使用其它任务的隐状态，或使用注意力机制来主动选取有用的信息。
- **层次共享模式**：一般神经网络中不同层抽取的特征类型不同。底层一般抽取一些低级的局部特征，高层抽取一些高级的抽象语义特征。因此如果多任务学习中不同任务也有级别高低之分，那么一个合理的共享模式是让低级任务在底层输出，高级任务在高层输出。
- **共享-私有模式**：一个更加分工明确的方式是将共享模块和任务特定（私有）模块的责任分开。共享模块捕捉一些跨任务的共享特征，而私有模块只捕捉和特定任务相关的特征。最终的表示由共享特征和私有特征共同构成。

学习步骤 在多任务学习中，每个任务都可以有自己单独的训练集。为了让所有任务同时学习，我们通常会使用交替训练的方式来“近似”地实现同时学习。

假设这 M 任务对应的模型分别为 $f_m(\mathbf{x},\theta), 1 \leq m \leq M$，多任务学习的联合目标函数为所有任务损失函数的线性加权。

$$\mathcal{L}(\theta) = \sum_{m=1}^{M} \sum_{n=1}^{N_m} \eta_m \mathcal{L}_m \left(f_m(x^{(m,n)}, \theta), y^{(m,n)} \right), \tag{10.25}$$

（1）联合训练阶段：每次迭代时，随机挑选一个任务，然后从这个任务中随机选择一些训练样本，计算梯度并更新参数；（2）单任务精调阶段：基于多任务的学习到的参数，分别在每个单独任务进行精调。其中单任务精调阶段为可选阶段。当多个任务的差异性比较大时，在每个单任务上继续优化参数可以进一步提升模型能力。

算法 10.4: 多任务学习中联合训练过程

输入: M 个任务的数据集 $\mathcal{D}_m, 1 \leq m \leq M$;
每个任务的批量大小 $K_m, 1 \leq m \leq M$;
最大迭代次数 T，学习率 α;

随机初始化参数 θ_0;
for $t = 1 \cdots T$ **do**
  // 准备 M 个任务的数据
  for $m = 1 \cdots M$ **do**
    将任务 m 的训练集 $\mathcal{D}_m$ 中随机划分为 $c = \frac{N_m}{K_m}$ 个小批量集合：

```
        | | $\mathcal{B}_m = \{\mathcal{I}_{m,1}, \cdots, \mathcal{I}_{m,c}\}$;
    5   | end
    6   | 合并所有小批量样本 $\bar{\mathcal{B}} = \mathcal{B}_1 \cup \mathcal{B}_2 \cup \cdots \cup \mathcal{B}_M$;
    7   | 随机排序 $\bar{\mathcal{B}}$;
    8   | foreach $\mathcal{I} \in \bar{\mathcal{B}}$ do
    9   | | 计算小批量样本 $\mathcal{I}$ 上的损失 $\mathcal{L}(\theta)$ ; // 只计算 $\mathcal{I}$ 在对应任务上
        | |   的损失
   10   | | 更新参数: $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \nabla_\theta \mathcal{L}(\theta)$;
   11   | end
   12 end
      输出: 模型 $f_m, 1 \leq m \leq M$
```

1. 多任务学习在多个任务的数据集上进行训练，比单任务学习的训练集更大。由于多个任务之间有一定的相关性，因此多任务学习相当于是一种隐式的数据增强，可以提高模型的泛化能力。

2. 多任务学习中的共享模块需要兼顾所有任务，这在一定程度上避免了模型过拟合到单个任务的训练集，可以看作是一种正则化。

3. 既然一个好的表示通常需要适用于多个不同任务，多任务学习的机制使得它会比单任务学习可以获得一个更好的表示。

4. 在多任务学习中，每个任务都可以“选择性”利用其他任务中学习到的隐藏特征，从而提高自身的能力。

迁移学习（Transfer Learning）

两个不同领域的知识迁移过程，利用源领域（Source Domain）D_S中学到的知识用来帮助目标领域（Target Domain）D_T上的学习任务。源领域的训练样本数量一般远大于目标领域。

一般的机器学习都是指归纳学习，即希望在训练数据集上学习到使得期望风险（即真实数据分布上的错误率）最小的模型。而转导学习的目标是学习一种在给定测试集上错误率最小的模型，在训练阶段可以利用测试集的信息。

归纳迁移学习是指在源领域和任务上学习出一般的规律，然后将这个规律迁移到目标领域和任务上；而转导迁移学习是一种从样本到样本的迁移，直接利用源领域和目标领域的样本进行迁移学习。

归纳迁移学习

归纳迁移学习要求源领域和目标领域是相关的，并且源领域D_S有大量的训练样本，这些样本可以是有标注的样本也可以是无标注样本。

当源领域只有大量无标注数据时，源任务可以转换为无监督学习任务；当源领域有大量的标注数据时，可以直接将源领域上训练的模型迁移到目标领域上。

- **基于特征的方式**：将预训练模型的输出或者是中间隐藏层的输出作为特征直接加入到目标任务的学习模型中。目标任务的学习模型可以是一般的浅层分类器（比如支持向量机等）或一个新的神经网络模型。

- **精调的方式**：在目标任务上复用预训练模型的部分组件，并对其参数进行精调（fine-tuning）。

（1）多任务学习是同时学习多个不同任务，而归纳迁移学习是通常分为两个阶段，即源任务上的学习阶段，和目标任务上的迁移学习阶段；（2）归纳迁移学习是单向的知识迁移，希望提高模型在目标任务上的性能，而多任务学习是希望提高所有任务的性能。

转导迁移学习

1. 协变量偏移（Covariate Shift）：源领域和目标领域的输入边际分布不同 $p_S(\mathbf{x}) \neq p_T(\mathbf{x})$，但后验分布相同 $p_S(y|\mathbf{x}) = p_T(y|\mathbf{x})$，即学习任务相同 $\mathcal{T}_S = \mathcal{T}_T$。
2. 概念偏移（Concept Shift）：输入边际分布相同 $p_S(\mathbf{x}) = p_T(\mathbf{x})$，但后验分布不同 $p_S(y|\mathbf{x}) \neq p_T(y|\mathbf{x})$，即学习任务不同 $\mathcal{T}_S \neq \mathcal{T}_T$。
3. 先验偏移（Prior Shift）：源领域和目标领域中的输出 y 先验分布不同 $p_S(y) \neq p_T(y)$，条件分布相同 $p_S(\mathbf{x}|y) = p_T(\mathbf{x}|y)$。在这样情况下，目标领域必须提供一定数量的标注样本。

习领域无关（Domain-Invariant）的表示。假设 $p_S(y|\mathbf{x}) = p_T(y|\mathbf{x})$，领域适应的目标是学习一个模型 $f : \mathcal{X} \to \mathcal{Y}$ 使得

$$\mathcal{R}_T(\theta_f) = \mathbb{E}_{(\mathbf{x},y)\sim p_T(\mathbf{x},y)}[\mathcal{L}(f(\mathbf{x},\theta_f),y)] \tag{10.26}$$

$$= \mathbb{E}_{(\mathbf{x},y)\sim p_S(\mathbf{x},y)}\frac{p_T(\mathbf{x},y)}{p_S(\mathbf{x},y)}[\mathcal{L}(f(\mathbf{x},\theta_f),y)] \tag{10.27}$$

$$= \mathbb{E}_{(\mathbf{x},y)\sim p_S(\mathbf{x},y)}\frac{p_T(\mathbf{x})}{p_S(\mathbf{x})}[\mathcal{L}(f(\mathbf{x},\theta_f),y)], \tag{10.28}$$

如果我们可以学习一个映射函数 $g : \mathcal{X} \to \mathbb{R}^d$，将 $\mathbf{x}$ 映射到一个特征空间中，并在这个特征空间中使得源领域和目标领域的边际分布相同 $p_S\big(g(\mathbf{x},\theta_g)\big) = p_T\big(g(\mathbf{x},\theta_g)\big), \forall \mathbf{x} \in \mathcal{X}$，其中 θ_g 为映射函数的参数，那么目标函数可以近似为

$$\mathcal{R}_T(\theta_f,\theta_g) = \mathbb{E}_{(\mathbf{x},y)\sim p_S(\mathbf{x},y)}\left[\mathcal{L}\Big(f\big(g(\mathbf{x},\theta_g),\theta_f\big),y\Big)\right] + \gamma d_g(S,T) \tag{10.29}$$

$$= \mathcal{R}_S(\theta_f,\theta_g) + \gamma d_g(S,T), \tag{10.30}$$

$$\mathcal{D}_S = \{(\mathbf{x}_S^{(n)}, y_S^{(n)})\}_{n=1}^N \sim p_S(\mathbf{x},y), \tag{10.31}$$

$$\mathcal{D}_T = \{\mathbf{x}_T^{(m)}\}_{m=1}^M \sim p_T(\mathbf{x},y), \tag{10.32}$$

对于训练集中的每一个样本 $\mathbf{x}$，我们都赋予 $z \in \{1,0\}$ 表示它是来自于源领域还是目标领域，领域判别器 $c\big(\mathbf{h},\theta_c\big)$ 根据其映射特征 $\mathbf{h} = g(\mathbf{x},\theta_g)$ 来预测它自于源领域的概率 $p(z=1|\mathbf{x})$。由于领域判别是一个两分类问题，$\mathbf{h}$ 来自于目标领域的概率为 $1 - c\big(\mathbf{h},\theta_c\big)$。

因此，领域判别器的损失函数为：

$$\mathcal{L}_c(\theta_g, \theta_c) = \frac{1}{N}\sum_{n=1}^{N} \log c(\mathbf{h}_S^{(n)}, \theta_c) + \frac{1}{M}\sum_{m=1}^{M} \log\left(1 - c(\mathbf{x}_D^{(m)}, \theta_c)\right), \tag{10.33}$$

其中 $\mathbf{h}_S^{(n)} = g(\mathbf{x}_S^{(n)}, \theta_g)$，$\mathbf{h}_D^{(m)} = g(\mathbf{x}_D^{(m)}, \theta_g)$ 分别为样本 $\mathbf{x}_S^{(n)}$ 和 $\mathbf{x}_D^{(m)}$ 的特征向量。

这样，领域迁移的目标函数可以分解为两个对抗的目标。一方面，要学习参数 θ_c 使得领域判别器 $c(\mathbf{h}, \theta_c)$ 尽可能区分出一个表示 $\mathbf{h} = g(\mathbf{x}, \theta_g)$ 是来自于哪个领域；另一方面，要学习参数 θ_g 使得提取的表示 $\mathbf{h}$ 无法被领域判别器 $c(\mathbf{h}, \theta_c)$ 预测出来，并同时学习参数 θ_g 使得模型 $f(\mathbf{h}, \theta_f)$ 在源领域的损失最小。

$$\min_{\theta_c} \quad \mathcal{L}_c(\theta_f, \theta_c), \tag{10.34}$$

$$\min_{\theta_f, \theta_g} \quad \sum_{n=1}^{N} \mathcal{L}\left(f\left(g(\mathbf{x}_S^{(n)}, \theta_g), \theta_f\right), y_S^{(n)}\right) - \gamma \mathcal{L}_c(\theta_f, \theta_c). \tag{10.35}$$

终生学习（Lifelong Learning）

灾难性遗忘（Catastrophic Forgetting），即按照一定顺序学习多个任务时，在学习新任务的同时不忘记先前学会的历史任务——弹性权重巩固（Elastic Weight Consolidation）。

不失一般性，以两个任务的持续学习为例，假设任务 $\mathcal{T}_A$ 和任务 $\mathcal{T}_B$ 的数据集分别为 $\mathcal{D}_A$ 和 $\mathcal{D}_B$。从贝叶斯的角度来看，我们希望计算给定两个任务时参数的后验分布：

$$\log p(\theta|\mathcal{D}) = \log p(\mathcal{D}|\theta) + \log p(\theta) - \log p(\mathcal{D}), \tag{10.36}$$

根据独立同分布假设，上式可以写为

$$\log p(\theta|\mathcal{D}) = \underline{\log p(\mathcal{D}_A|\theta)} + \log p(\mathcal{D}_B|\theta) + \underline{\log p(\theta)} - \underline{\log p(\mathcal{D}_A)} - \log p(\mathcal{D}_B), \tag{10.37}$$

$$= \log p(\mathcal{D}_B|\theta) + \log p(\theta|\mathcal{D}_A) - \log p(\mathcal{D}_B), \tag{10.38}$$

其中 $p(\theta|\mathcal{D}_A)$ 包含了所有从任务 $\mathcal{T}_A$ 的学习的信息。当顺序地学习任务 $\mathcal{T}_B$ 时，参数在两个任务上的后验分布和其在任务 $\mathcal{T}_A$ 的后验分布有关。

因此，对于任务 $\mathcal{T}_A$ 的数据集 $\mathcal{D}_A$，我们可以用 Fisher 信息矩阵来衡量一个参数携带的关于 $\mathcal{D}_A$ 的信息量。

$$F^A(\theta) = \frac{1}{N}\sum_{(\mathbf{x},y)\in\mathcal{D}_A} \nabla_\theta \log p(y|\mathbf{x}, \theta)(\nabla_\theta \log p(y|\mathbf{x}, \theta))^{\mathrm{T}}. \tag{10.49}$$

通过上面的近似，在训练任务 $\mathcal{T}_B$ 时的损失函数为

$$\mathcal{L}(\theta) = \mathcal{L}_B(\theta) + \sum_{i=1}^{N} \frac{\lambda}{2} F_i^A \cdot (\theta_i - \theta_{A,i}^*)^2, \tag{10.50}$$

其中 $\mathcal{L}_B(\theta)$ 为任务 $p(\theta|\mathcal{D}_B)$ 的损失函数，F_i^A 为 Fisher 信息矩阵的第 i 个对角线元素，λ 为平衡两个任务重要性的超参数，N 为参数的总数量。

元学习（Meta-Learning）

从已有任务中学习一种学习方法或元知识，可以加速新任务的学习。元学习十分类似于归纳迁移学习，但元学习更侧重从多种不同（甚至是不相关）的任务中归纳出一种学习方法。

基于优化器的元学习和模型无关的元学习。

基于优化器的元学习

我们用函数$g_t(\cdot)$来预测第t步时参数更新的差值$\Delta\theta_t = \theta_t - \theta_{t-1}$。函数$g_t(\cdot)$称为优化器，输入是当前时刻的梯度值，输出是参数的更新差值$\Delta\theta_t$。这样第t步的更新规则可以写为

$$\theta_{t+1} = \theta_t + g_t\big(\nabla\mathcal{L}(\theta_t), \phi\big) \tag{10.52}$$

任务集合　　元学习算法

在优化器的元学习中，我们希望在每步迭代的目标是L(θ)最小，具体的目标函数为

$$\mathcal{L}(\phi) = \mathbb{E}_f\big[\sum_{t=1}^{T} w_t\mathcal{L}(\theta_t)\big], \tag{10.53}$$

$$\theta_t = \theta_{t-1} + \mathbf{g}_t, \tag{10.54}$$

$$[\mathbf{g}_t; \mathbf{h}_t] = \mathbf{LSTM}\big(\nabla\mathcal{L}(\theta_{t-1}), \mathbf{h}_{t-1}, \phi\big), \tag{10.55}$$

在每步训练时，随机初始化模型参数，计算每一步的$L(\theta_t)$，以及元学习的损失函数$L(\emptyset)$，并使用梯度下降更新参数。为每个参数都使用一个共享的 LSTM 网络来进行更新，这样可以使用一个非常小的共享 LSTM 网络来更新参数。

模型无关的元学习（Model-Agnostic Meta-Learning， MAML）

假设所有的任务都来自于一个任务空间，其分布为$p(\mathcal{T})$，我们可以在这个任务空间的所有任务上学习一种通用的表示，这种表示可以经过梯度下降方法在一个特定的单任务上进行精调。假设一个模型为$f(\theta)$，如果我们让这个模型适应到一个新任务$\mathcal{T}_m$上，通过一步或多步的梯度下降更新，学习到的任务适配参数为

$$\theta'_m = \theta - \alpha\nabla_\theta\mathcal{L}_{\mathcal{T}_m}(f_\theta), \tag{10.56}$$

MAML 的目标是学习一个参数θ使得其经过一个梯度迭代就可以在新任务上达到最好的性能。

$$\min_\theta \sum_{\mathcal{T}_m\sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_m}\big(f(\theta'_m)\big) = \sum_{\mathcal{T}_m\sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_m}\Big(f\big(\underline{\theta - \alpha\nabla_\theta\mathcal{L}_{\mathcal{T}_m}(f_\theta)}\big)\Big) \tag{10.57}$$

在所有任务上的元优化(Meta-Optimization)也采用梯度下降来进行优化，

$$\theta \leftarrow \theta - \beta\nabla_\theta \sum_{m=1}^{M} \mathcal{L}_{\mathcal{T}_m}(f_{\theta'_m}), \tag{10.58}$$

算法 10.5: 模型无关的元学习

输入: 任务分布$p(\mathcal{T})$;

最大迭代次数T，学习率α, β;

1 随机初始化参数θ;

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

  根据 $p(\mathcal{T})$ 采样一个任务集合 $\{\mathcal{T}_m\}_{m=1}^{M}$ **for** $m = 1 \cdots M$ **do**

    计算 $\nabla_\theta \mathcal{L}_{\mathcal{T}_m}(f_\theta)$;

    计算任务适配的参数：$\theta'_m \leftarrow \theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_m}(f_\theta)$;

  end

  更新参数：$\theta \leftarrow \theta - \beta \nabla_\theta \sum_{m=1}^{M} \mathcal{L}_{\mathcal{T}_m}(f_{\theta'_m})$;

end

输出: 模型 f_θ