

暑期讲义

作者：有80%信心的DSY

主旨：阿巴阿巴随便写点，慢慢丰富

序言：暂无

参考书籍：《机器学习》（周志华西瓜书），《统计学习方法》（李航），《deep learning book》，《Understanding Machine Learning: From Theory to Algorithms》，《High-dimensional probability: an introduction with applications in data science》，pytorch doc，《点集拓扑讲义》

私货极多欢迎批评，邮箱：madsy@mail.scut.edu.cn

想感谢的人：感谢所有帮助关心我的小伙伴

献给XXXX

最近前面内容好像有个主旨：本讲义想从机器学习基本问题出发，讲明白对于神经网络甚至更广义的可微模型编程建模逻辑。

1.1机器学习基础概念

1.1集合与模型

1.1.1 事件集，Definition：

真实世界里我们研究对象数值建模化的集合 Ω 。

eg：如各种各样的图片，声音，文本，信号。

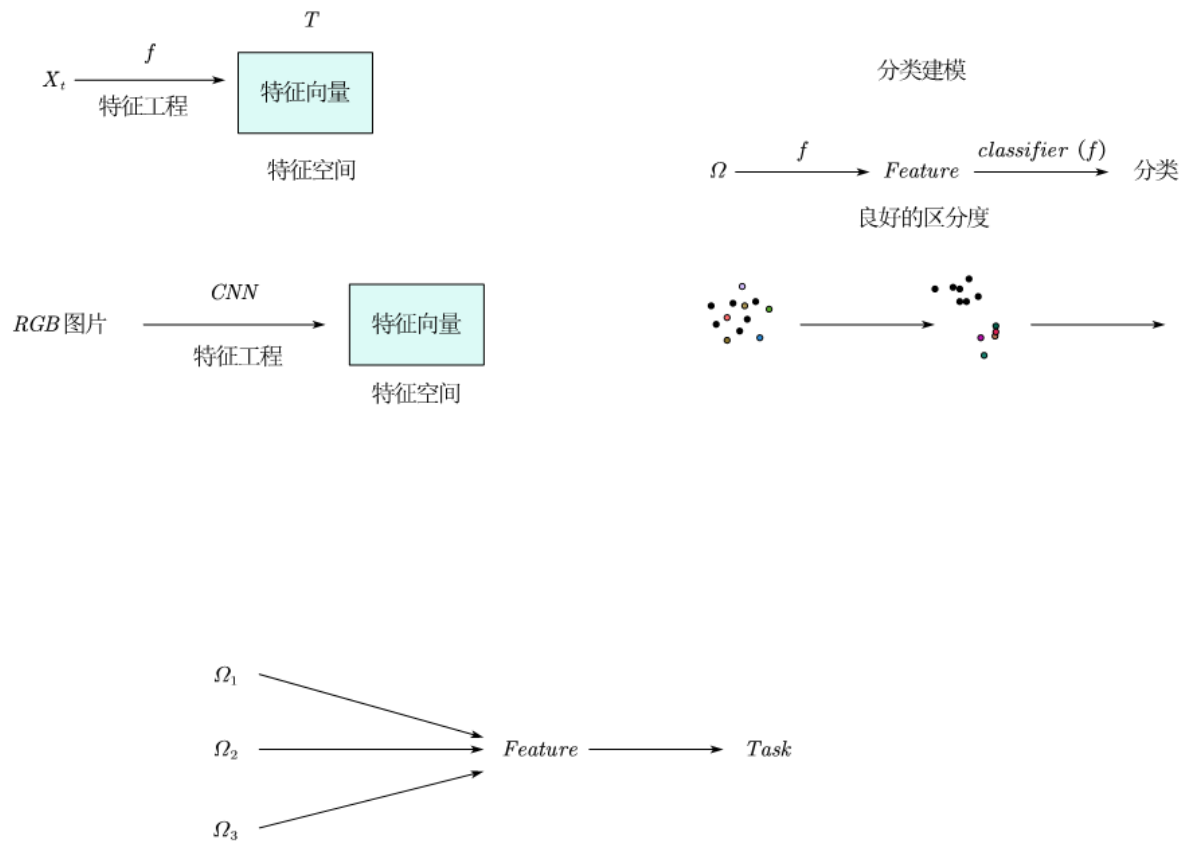
1.1.2 输入空间，Definition：

所有可能的输入元素组成的集合 \mathcal{X} ，事件集一定是输入空间。

1.1.3 输出空间，Definition：

所有可能的输出元素组成的集合 \mathcal{Y} ，事件集可以是输出空间。

输入输出空间，事件集eg：



1.1.4联合分布假设:x

对于定义在 \mathcal{X} 和 \mathcal{Y} 上的随机变量 X, Y ，我们假设在 $\mathcal{X} \times \mathcal{Y}$ 上其存在联合分布。

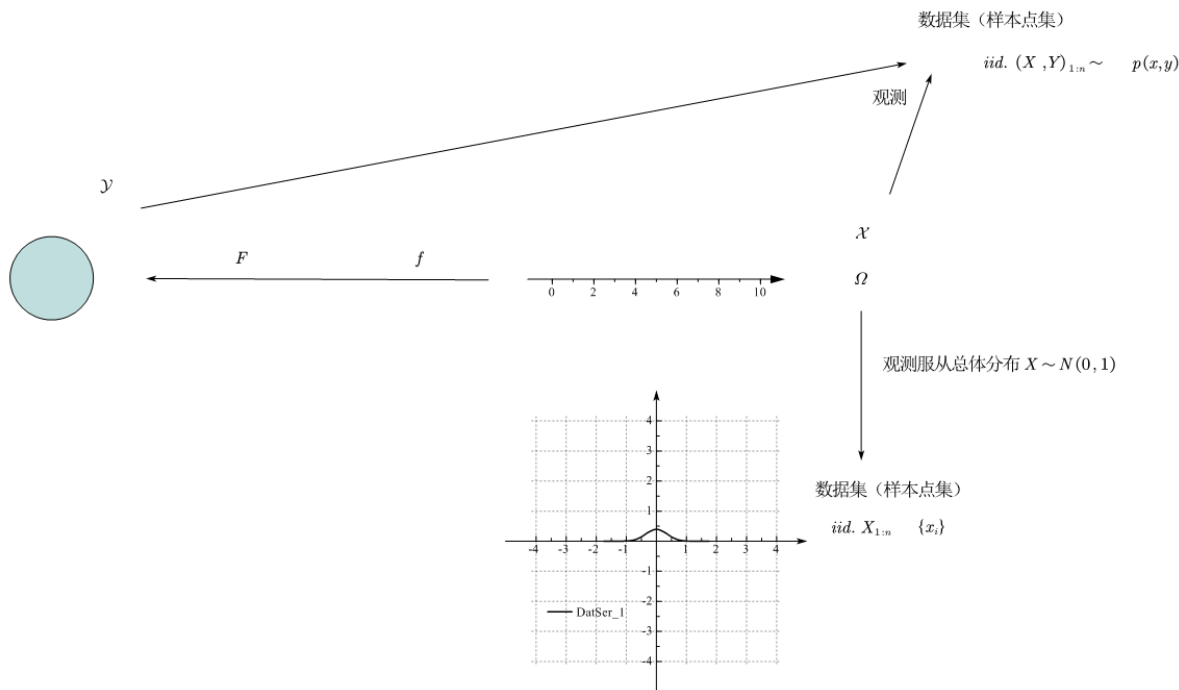
1.1.5真实映射假设:

对于定义在 \mathcal{X} 和 \mathcal{Y} 上的随机变量 X, Y ，我们假设存在最优的映射 $y = F(x), P(x, y)$ 可以转换为 $P'(x)$.
 $(P(x, y) = P(x, F(x)))$

1.1.6数据集（样本点观测集）， Definition:

对应集合上定义的随机变量，观察/采样到的独立同分布的样本点集。

eg:比如对于 $\mathcal{X} \times \mathcal{Y}$ 上的数据集 $\{(x_i, y_i)\} \sim P(x, y)$



1.1.7模型, Definition:

定义在由 \mathcal{X} 到 \mathcal{Y} 的一个映射 f , 该模型既可以是概率模型 $P(x)$, 也可以是非概率模型 $f(x)$

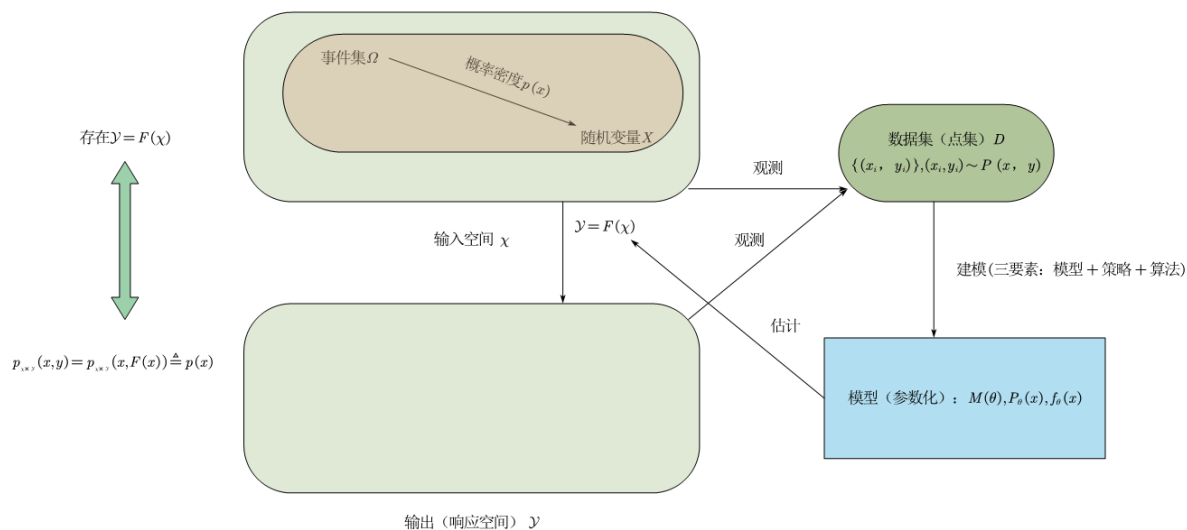
1.1.8假设空间, Definition:

对于我们假设所有可能模型 f 的集合 \mathcal{H} .

1.1.9参数化模型及其假设空间, Definition:

定义在由 \mathcal{X} 到 \mathcal{Y} 的映射 f 由参数 θ 参数化, $\theta \in \Theta$,我们可以由 Θ 定义我们的假设空间 \mathcal{H}_Θ

eg: $\{f|f(x) = ax + b, a \in R, b \in R\}$ 定义了所有直线构成的假设空间。



1.2泛化误差及变分问题

1.2.1损失函数, Definition:

对于输出空间 $\mathcal{Y}, \forall a, b \in \mathcal{Y}$ 对于二元函数 $L(a, b)$, L 满足:

$$L(a, b) \geq 0$$

$$L(a, b) = 0, \text{ if and only if } a = b$$

eg:

如果 \mathcal{Y} 是一个度量空间，则其上任意一个距离 d 都是损失函数。

1.2.3泛化误差, Definition:

对于一对输入输出空间 \mathcal{X} 和 \mathcal{Y} 上的随机变量 X, Y ，对任意模型 f ，我们称：

$$R_{\text{exp}}(f) \triangleq E_{P, \mathcal{X} \times \mathcal{Y}}[L(\mathcal{Y}, f(X))] = \int_{\mathcal{X} \times \mathcal{Y}} L(\mathcal{Y}, f(x))p(x, y)dx dy$$

$$\text{or } R_{\text{exp}}(f) \triangleq E_{P, \mathcal{X}}[L(F(X), f(X))] = \int_{\mathcal{X}} L(F(x), f(x))p'(x)dx$$

What does Machine Learning do?

$$\min_{f \in \mathcal{H}} R_{\text{exp}}(f) = R(f)$$

1.2 expand:泛函

1.2.3泛函, Definition:

集合 \mathcal{H} 是一个函数集合， R 是实数域，我们定义 \mathcal{H} 到 R 的一个映射 $y = J(f)$ 为泛函。（本质是向量空间到标量的映射）。泛函一般由积分定义，实际上我们已经接触过很多可以定义泛函的东西。

eg:

$$J(y) = \int_{x_0}^{x_1} \sqrt{1 + y'^2} dx$$

为二维平面上，过点 x_0, x_1 的所有可微曲线的泛函，该泛函的意义是这些曲线的长度。

$$J(p) = - \int p(x) \log p(x) dx$$

这是信息熵，衡量随机变量不确定性的泛函。

$$J(p) = KL(p||q) = \int p(x) \log \frac{p(x)}{q(x)}$$

这是关于 p 和 q 的一个损失。

当然泛化误差也可以定义一种泛函：

$$J(f) = R_{\text{exp}}(f) \triangleq E_{P, \mathcal{X} \times \mathcal{Y}}[L(\mathcal{Y}, f(X))] = \int_{\mathcal{X} \times \mathcal{Y}} L(\mathcal{Y}, f(x))p(x, y)dx dy$$

那么机器学习和泛函有什么关系？

1.2.4泛函极值 (变分问题) Definition:

我们称对于一个定义在 \mathcal{H} 上的泛函 $J(f)$ ，问题

$$\min_{f \in \mathcal{H}} J(f)$$

称之为泛函极值问题，也可以叫做变分问题。

机器学习的数学抽象本质是变分问题，我们试图要最小化泛化误差这个泛函：

$$\min_{f \in \mathcal{H}} R_{exp}(f) = R(f)$$

对于由损失函数定义的变分问题，最优解是显然的，若真实映射 $F \in \mathcal{H}$, F 是最优解，不过很可惜，和传统的泛函分析面对的变分问题不同，我们无法解析的求解该变分问题。

原因有以下这么几个：

1.最重要的原因：我们无法解析知道输入空间输出空间确切的分布，和整个集合。我们所能拿到的只有数据集（样本点观测集）。

2.真实映射 F 是不可知的，我们的假设空间很难包含 F

...

那么如何去求解该变分问题？

答案：借助概率论，实分析，测度论的工具进行估计，用统计量建立的学习策略，统计量是可以计算的，再通过参数化模型把变分问题转换成数值优化问题。

1.3 PAC Theory

1.3.1 经验损失, Definition:

对于可观测到的 N 个点集——数据集 $\{(x_i, y_i)\} \sim P(x, y)$

我们定义一种常见的学习策略 $\hat{R}(f)$ ：

$$\hat{R}(f) = \frac{1}{N} \sum_{i=1}^N L(F(x_i), f(x_i))$$

为经验损失。

变分问题裂解

考虑：

$$R(f) = [R(f) - \hat{R}(f)] + \hat{R}(f)$$

1.3.2 Generalization Gap, Definition:

我们定义如上裂解中方括号中的内容 $[R(f) - \hat{R}(f)]$ 叫做 **Generalization Gap**，此后记为 **GP**。

这是个很重要的分解，可以这么说，这个看起来平常的式子，把一个变分问题拆成两部分，前面交给统计学家，后面交给优化学家。

抽象理解：

对于GP问题，统计学家要干的事情就是通过概率论，实分析，测度论等工具，对假设空间 \mathcal{H} 建立如下依概率不等式结论：

$$\forall \varepsilon > 0, P(\exists f \in \mathcal{H}, GP \geq \varepsilon) \leq \frac{m(d_{\mathcal{H}})}{g(N) * h(\varepsilon)} \quad (1)$$

或者等价的

$$\forall \varepsilon > 0, P(\forall f \in \mathcal{H}, GP \leq \varepsilon) \geq 1 - \frac{m(d_{\mathcal{H}})}{g(N) * h(\varepsilon)} \quad (2)$$

其中 $d_{\mathcal{H}}$ 的含义是假设空间的一个复杂度度量，如果 \mathcal{H} 是一个有限集合，则d可以是其元素个数，对于无限集合，我们后面会介绍他的一个复杂度度量VC Dimension，N是观测到的数据量， ε 是我们想要GP误差限，其中m, g, f都是一个多项式或者指数级别的函数。

也就是说根据我们的数据量，我们能以 $1 - \frac{m(d_{\mathcal{H}})}{g(N) * h(\varepsilon)}$ 的概率把握把GP控制在 ε 以内。

1.3.3 Probability Approximate Correct Learnable (PAC可学习) , Definition:

若假设 \mathcal{H} 空间对于真实映射 F ,存在 $\hat{R}(f)$ 使得不等式 (2) 成立，则我们称 F 对于 \mathcal{H} 是PAC Learnable的 (PAC可学习)。

注意！：

改概念不等价于GP依概率收敛到0！！！！！！！！！！！！！！！！（依测度收敛到0只是必要条件）

这里比GP依概率收敛到0要求更强！！！！

PAC Learnable GP 一定依概率收敛到0

GP依据概率收敛到0不一定PAC Learnable

重要的是一定要有一个和 $d_{\mathcal{H}}, N, \varepsilon$ 有关的概率Bound，要能控制。

从统计的角度理解就是 $\hat{R}(f)$ 是 $R(f)$ 的弱相合统计量是PAC Learnable的一个必要条件。

有限假设集定理

定理1.3.4：有限假设集原理

对于输入空间 \mathcal{X} 和布尔输出空间 $\mathcal{Y} = \{0, 1\}$,我们的假设空间 \mathcal{H} 有限，即 $\mathcal{H} = \{f_1, f_2, \dots, f_d\}$ 。

我们有：

$$\forall \varepsilon > 0, P(\forall f \in \mathcal{H}, GP \leq \varepsilon) \geq 1 - \frac{d}{\exp\{2N\varepsilon^2\}}$$

即有限假设空间对于任意布尔映射是PAC Learnable。

该证明需要用到Hoeffding inequality:

定理1.3.4.1 Hoeffding inequality (霍夫丁不等式)

X_1, \dots, X_n 为有界独立随机变量, 即 $A_i \leq X_i \leq B_i, 1 \leq i \leq n$. 对于任意 $\varepsilon > 0$, 和任意 $t > 0$,

$$P\left(\sum_{i=1}^n (X_i - E[X_i]) \geq \varepsilon\right) \leq \exp\left(-t\varepsilon + \frac{t^2}{8} \sum_{i=1}^n (B_i - A_i)^2\right)$$

进一步:

$$P\left(\sum_{i=1}^n (X_i - E[X_i]) \geq \varepsilon\right) \leq \exp\left(-\frac{2\varepsilon^2}{\sum_{i=1}^n (B_i - A_i)^2}\right)$$

而要证明 **Hoeffding inequality** 我们先要需要证明高维统计如下几个引理:

引理1.3.4a: Markov inequality (马尔可夫不等式)

$$X \geq 0, \forall \varepsilon > 0, P(x \geq \varepsilon) \leq \frac{1}{\varepsilon} EX.$$

proof:

$$\begin{aligned} EX &= \int_0^{+\infty} x dF(x) \geq \int_{\varepsilon}^{+\infty} x dF(x) \geq \varepsilon \int_{\varepsilon}^{+\infty} dF(x) = \varepsilon P(x \geq \varepsilon). \\ \therefore P(x \geq \varepsilon) &\leq \frac{1}{\varepsilon} EX. \end{aligned}$$

定理1.3.4.1 Hoeffding inequality

proof:

我们考虑:

$$Y_i \triangleq X_i - EX_i, EY_i = 0$$

可以看出:

$$a_i = A_i - EX_i \leq Y_i \leq b_i = B_i - EX_i$$

原问题等价:

$$P(\sum Y_i \geq \varepsilon) \leq \exp\left(-t\varepsilon + \frac{t^2}{8} \sum (b_i - a_i)^2\right)$$

考虑:

$$P(\sum Y_i \geq \varepsilon) = P(e^{t\sum Y_i} \geq e^{t\varepsilon})$$

由 **Markov inequality**:

$$P(e^{t\sum Y_i} \geq e^{t\varepsilon}) \leq e^{-t\varepsilon} \cdot E[e^{t\sum Y_i}]$$

因为 Y_i 独立:

$$P(e^{t\sum Y_i} \geq e^{t\varepsilon}) \leq e^{-t\varepsilon} \cdot \prod_i E[e^{tY_i}]$$

又因为 $a_i \leq Y_i \leq b_i$,

$$Y_i = (1 - \alpha)a_i + \alpha b_i, \quad \alpha = \frac{Y_i - a_i}{b_i - a_i} \in [0, 1]$$

因为 e^t 是凸的:

$$\begin{aligned} E[e^{tY_i}] &\leq E\left[\frac{b_i - Y_i}{b_i - a_i}e^{ta_i} + \frac{Y_i - a_i}{b_i - a_i}e^{tb_i}\right] \\ E[e^{tY_i}] &\leq \frac{b_i}{b_i - a_i}e^{ta_i} - \frac{a_i}{b_i - a_i}e^{tb_i}. \end{aligned}$$

令 $u = t(b_i - a_i) > 0$, $\gamma = -\frac{a_i}{b_i - a_i} \Rightarrow \frac{b_i}{b_i - a_i} = 1 - \gamma$, 得到:

$$-\frac{a_i}{b_i - a_i}e^{ta_i} + \frac{b_i}{b_i - a_i}e^{tb_i} = \gamma e^{(1-\gamma)u} + (1-\gamma)e^{-\gamma u} = e^{-\gamma u}(\gamma e^u + 1 - \gamma) \quad (3)$$

令 $g(u) \triangleq \log(3) = -\gamma u + \log(\gamma e^u + 1 - \gamma)$, 考虑0点泰勒展开:

$$\begin{aligned} g(0) &= 0. \quad g'(u) = -\gamma + \frac{\gamma e^u}{\gamma e^u + 1 - \gamma}. \quad g'(0) = 0 \\ g''(u) &= \frac{\gamma e^u(\gamma e^u + 1 - \gamma) - \gamma e^u(\gamma e^u)}{(\gamma e^u + 1 - \gamma)^2} = \frac{(1-\gamma)\gamma e^u}{(\gamma e^u + 1 - \gamma)^2} \leq \frac{(1-\gamma)\gamma e^u}{\left(2\sqrt{\gamma e^u(1-\gamma)}\right)^2} = \frac{1}{4} \end{aligned}$$

得到:

$$g(u) = g(0) + ug'(0) + \frac{u^2}{2}g''(\xi) \quad 0 \leq \xi \leq u$$

得到:

$$g(u) = \frac{u^2}{2}g''(\xi) \leq \frac{u^2}{8}.$$

得到

$$E[e^{tY_i}] \leq e^{g(u)} \leq e^{\frac{1}{8}u^2} = e^{\frac{t^2}{8}(b_i - a_i)^2}$$

得到:

$$\begin{aligned} P\left(\sum Y_i \geq \varepsilon\right) &\leq \exp\left(-t\varepsilon + \frac{t^2}{8} \sum (b_i - a_i)^2\right) \\ P\left(\sum Y_i \geq \varepsilon\right) &\leq \exp\left(-\frac{2\varepsilon^2}{\sum (b_i - a_i)^2}\right) \end{aligned}$$

得证。

Expend PS:

实际上Hoeffding只是更抽象不等式的一个特例, 实际上指数型上界的概率不等式不止对有界独立变量成立, 对于许多无界的分布也都成立, 例如高斯分布, 在高维统计中我们有专门研究一类具有可控概率界中心矩的分布(次高斯分布, 次指数分布), Bernstein不等式等等就是用来描述这类过程的。这类不等式叫做**Concentration inequality**(集中不等式), 描述的是样本随着数量增多朝着期望的收敛的一类概率bound(向总测度集中), 这类不等式原理在机器学习的learning theory和很多前沿随机过程问题中非常常见。

定理1.3.4 有限假设集原理

proof:

对任意函数 $f \in \mathcal{H}$, $\hat{R}(f)$ 是 N 个独立的随机变量 $L(Y, f(X)) + \varepsilon$ 的样本均值, $R(f)$ 是随机变量 $L(Y, f(X))$ 的期望, 由 **Hoeffding inequality**:

$$P(R(f) - \hat{R}(f) \geq \varepsilon) \leq \exp(-2N\varepsilon^2)$$

又因为 $\mathcal{H} = \{f_1, f_2, \dots, f_d\}$ 有限:

$$\begin{aligned} P(\exists f \in \mathcal{H} : R(f) - \hat{R}(f) \geq \varepsilon) &= P\left(\bigcup_{f \in \mathcal{H}} \{R(f) - \hat{R}(f) \geq \varepsilon\}\right) \\ &\leq \sum_{f \in \mathcal{H}} P(R(f) - \hat{R}(f) \geq \varepsilon) \\ &\leq d \exp(-2N\varepsilon^2) \end{aligned}$$

1.4 VC Dimension (Vapnik-Chervonenkis Dimension) (待补充)

1.4.3 Empirical processes via VC dimension:

$$E \sup_{f \in \mathcal{H}} \left| \frac{1}{n} \sum_{i=1}^n f(X_i) - Ef(X) \right| \leq C \sqrt{\frac{vc(\mathcal{H})}{n}}$$

$$R(f) - \hat{R}(f) \leq E \sup_{f \in \mathcal{H}} |R(f) - \hat{R}(f)| = E \sup_{f \in \mathcal{H}} \left| \frac{1}{n} \sum_i (f(X_i) - \mathbb{E}f(X_i)) \right| \leq C \sqrt{\frac{vc(\mathcal{H})}{n}}$$

1.5 Uniform Convergence (待补充)

1.6 变分问题及参数优化问题

参数优化问题

1.6.1 参数化的假设空间, Definition:

我们这里讨论研究的是连续的参数空间 Θ , 不讨论离散的 Θ 。

f 是由参数 θ 定义, $\theta \in \Theta$, 参数化假设空间:

$$\mathcal{H} = \{f_\theta(x) | \theta \in \Theta\}$$

并定义参数化过程 $p: \Theta \rightarrow \mathcal{H}$ 。

1.6.2 经验损失变分问题, Definition:

对于可观测到的N个点集——数据集 $\{(x_i, y_i)\} \sim P(x, y)$, 参数化假设空间 $\mathcal{H} = \{f_\theta(x) | \theta \in \Theta\}$, 某种建模策略的经验损失 $\hat{R}(f)$, 定义:

$$\min_{f \in \mathcal{H}} \hat{R}(f) = \frac{1}{N} \sum_{i=1}^N L(F(x_i), f(x_i))$$

1.6.3模型参数优化问题, Definition:

对于可观测到的N个点集——数据集 $\{(x_i, y_i)\} \sim P(x, y)$, 参数化假设空间 $\mathcal{H} = \{f_\theta(x) | \theta \in \Theta\}$, 某种建模策略的经验损失 $\hat{R}(f)$, 定义:

$$\min_{\theta \in \Theta} L(\theta) = \hat{R}(f_\theta) = \frac{1}{N} \sum_{i=1}^N L(F(x_i), f_\theta(x_i))$$

为参数优化问题。

注意: 在我们1.6的讨论场景中 $\hat{R}(f)$ 仍然是一个泛函, 而**问题1.6.2**和**问题1.6.3**并不能想当然的认为是等价的, 两者等价需要满足一定的充要条件。最为重要的环节在于研究 \mathcal{H} 和 Θ 之间的关系, 要研究这两种之间的关系, 我们需要借助一点拓扑工具。

PS:

这里讨论的重要目的是保障柯西列收敛

$$\{x_i\} \in X, \{y_i\} \in Y, \lim x_i = x^*, \lim y_i = y^*, f(x^*) = y^*.$$

1.6 expand: 点集拓扑与同胚

(此处只是稍微展开背后的思想, 具体弄明白同胚的概念要学完整本点集拓扑)

1.6.4拓扑, Definition:

令 X 为一个集合, \mathcal{T} 是 X 的一个子集族 (族的概念就是集合的集合), 如果 \mathcal{T} 满足如下条件:

- (1) $X, \emptyset \in \mathcal{T}$
- (2) $\forall A, B \in \mathcal{T}$, 则 $A \cap B \in \mathcal{T}$
- (3) 若 $\mathcal{T}_1 \subset \mathcal{T}$, 则 $\bigcup_{A \in \mathcal{T}_1} A \in \mathcal{T}$

则称 \mathcal{T} 是 X 的一个拓扑。

且称集合 X 是一个相对于拓扑 \mathcal{T} 而言的拓扑空间, 记为偶对 (X, \mathcal{T})

1.6.5同胚, Definition:

使 (X, \mathcal{T}_X) 和 (Y, \mathcal{T}_Y) 为两个拓扑空间, 若存在使 $f: (X, \mathcal{T}_X) \rightarrow (Y, \mathcal{T}_Y)$ 。 f 被称为这两个空间之间的同胚映射当且仅当

- (1) f 是一个双射;

(2) f 是连续的;

(3) f^{-1} 是连续的。

我们也称 (X, \mathcal{T}_X) 和 (Y, \mathcal{T}_Y) 是同胚的。

假设空间及参数空间的同胚定理

1.6.6 参数冗余, Definition:

我们称参数化过程 p 是冗余的, 当且仅当对于 $\forall x \in \mathcal{X}, \exists \theta_1, \theta_2 \in \Theta, \theta_1 \neq \theta_2, f_{\theta_1}(x) = f_{\theta_2}(x)$ 。

eg: $\{f | f(x) = ax + b + c, a \in R, b \in R, c \in R\}$ 是冗余的。

定理1.6.7 假设空间及参数空间的同胚定理

问题1.6.2和问题1.6.3等价:

当且仅当 \mathcal{H} 和 Θ 同胚, 且 p 为其一个同胚映射。

或者说当且仅当参数化过程 p 不是冗余的。

proof:

因为 \mathcal{H} 和 Θ 同胚, 且 p 为其一个同胚映射, 所以 p 是一个双射, 所以对于问题1.6.2任意局优 f^* , 和收敛于问题1.6.2任意局优的柯西列 $\{f_i\}$, 都在 Θ 中存在

θ^* 及柯西列 $\{\theta_i\}$ 唯一对应。反过来对问题1.6.3同理, 因此我们自然得到了等价性的证明。

而由参数化过程 p 的构造过程, 我们知道 p 一定是 Θ 到 \mathcal{H} 的连续满射, 而参数冗余本质是在说明 p 还是 Θ 到 \mathcal{H} 的单射, 所以我们可以得到 p 是 Θ 到 \mathcal{H} 的同胚映射。

PS:

实际上此处可以更多理解一下泛函这个概念, 泛函的定义不是狭隘的函数到标量 (实数域) 的映射, 他本质是向量空间到实数域的映射, 只不过我们可以把函数当成一种向量。这里的同胚, 本质上是两个向量空间之间的同胚, 是最简单的同胚。

eg: 如果我们有 n 个参数, 即 $\theta = (\theta_1, \theta_2, \dots, \theta_n)$, 且每个参数的定义域都是全体实数, 即 $\Theta = R^n$, 那么我们的假设空间 \mathcal{H} 就是同胚于 R^n 的, 如此我们在假设空间 \mathcal{H} 上的任意变分问题, 等价于在 R^n 上针对参数 θ 的无约束数值优化问题。若 Θ 不是 R^n , 类似的也可以证明假设空间 \mathcal{H} 同胚于 R^n 的某个子空间, 转换为约束数值优化问题。

eg:

$$\{a_0 + a_1x + \dots + a_{n-1}x^{n-1} | a \in R^n\}$$

eg:

定义所有在0点一个邻域 B 内解析, 且他在0点麦克劳林级数收敛, 且其所有大于 $n+1$ 阶的导数都为0的函数的集合 \mathcal{H} 。

2.1Pytorch及连续参数假设空间建模理论

请大家必备网站：<https://pytorch.org/docs/stable/index.html>

Torch基础概念

2.1.1基础数据torch.tensor（数据点，参数，所有参与数值计算的数据）：

1 | torch.Tensor

tensor是包含单一数据类型元素的多维矩阵。

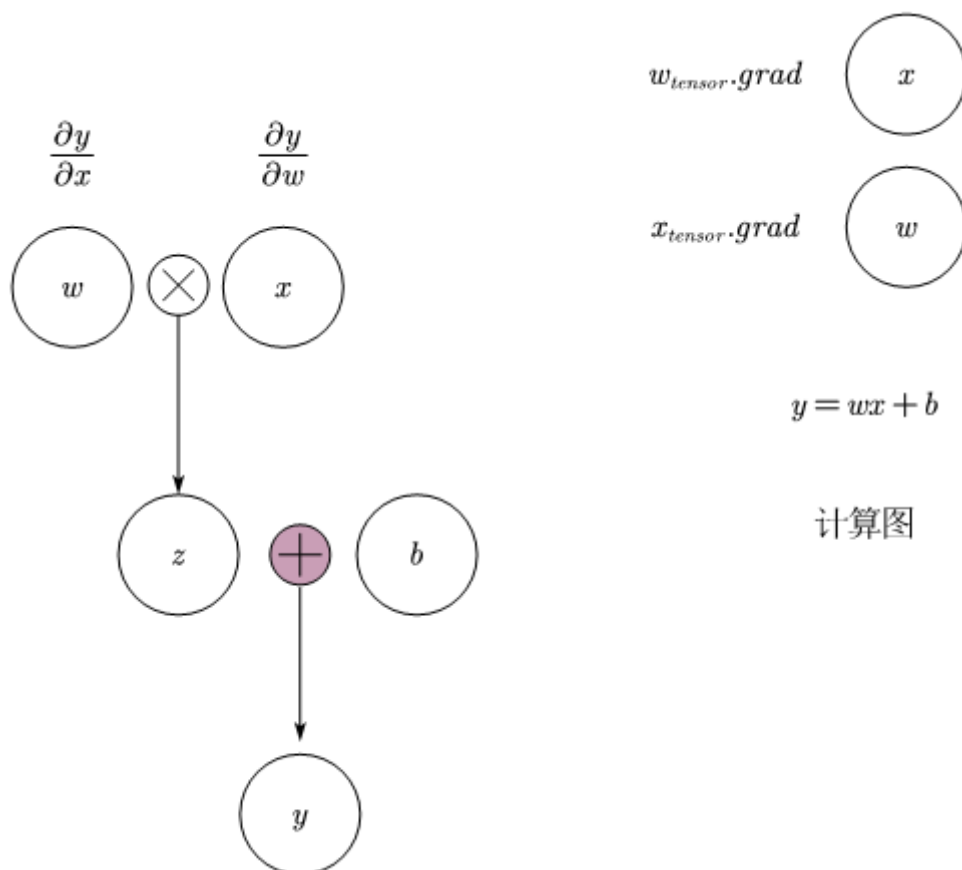
Torch 定义了 10 种具有 CPU 和 GPU 变体的张量类型：

数据类型	类型	CPU张量	GPU张量
32 位浮点	<code>torch.float32</code> 或者 <code>torch.float</code>	<code>torch.FloatTensor</code>	<code>torch.cuda.FloatTensor</code>
64 位浮点	<code>torch.float64</code> 或者 <code>torch.double</code>	<code>torch.DoubleTensor</code>	<code>torch.cuda.DoubleTensor</code>
16 位浮点数 ¹	<code>torch.float16</code> 或者 <code>torch.half</code>	<code>torch.HalfTensor</code>	<code>torch.cuda.HalfTensor</code>
16 位浮点数 ²	<code>torch.bfloat16</code>	<code>torch.BFloat16Tensor</code>	<code>torch.cuda.BFloat16Tensor</code>
32 位复数	<code>torch.complex32</code>		
64 位复数	<code>torch.complex64</code>		
128 位复数	<code>torch.complex128</code> 或者 <code>torch.cdouble</code>		

128 位复数	<code>torch.complex128</code> 或者 <code>torch.cdouble</code>		
8 位整数 (无符号)	<code>torch.uint8</code>	<code>torch.ByteTensor</code>	<code>torch.cuda.ByteTensor</code>
8 位整数 (有符号)	<code>torch.int8</code>	<code>torch.CharTensor</code>	<code>torch.cuda.CharTensor</code>
16 位整数 (有符号)	<code>torch.int16</code> 或者 <code>torch.short</code>	<code>torch.ShortTensor</code>	<code>torch.cuda.ShortTensor</code>
32 位整数 (有符号)	<code>torch.int32</code> 或者 <code>torch.int</code>	<code>torch.IntTensor</code>	<code>torch.cuda.IntTensor</code>
64 位整数 (有符号)	<code>torch.int64</code> 或者 <code>torch.long</code>	<code>torch.LongTensor</code>	<code>torch.cuda.LongTensor</code>
布尔值	<code>torch.bool</code>	<code>torch.BoolTensor</code>	<code>torch.cuda.BoolTensor</code>

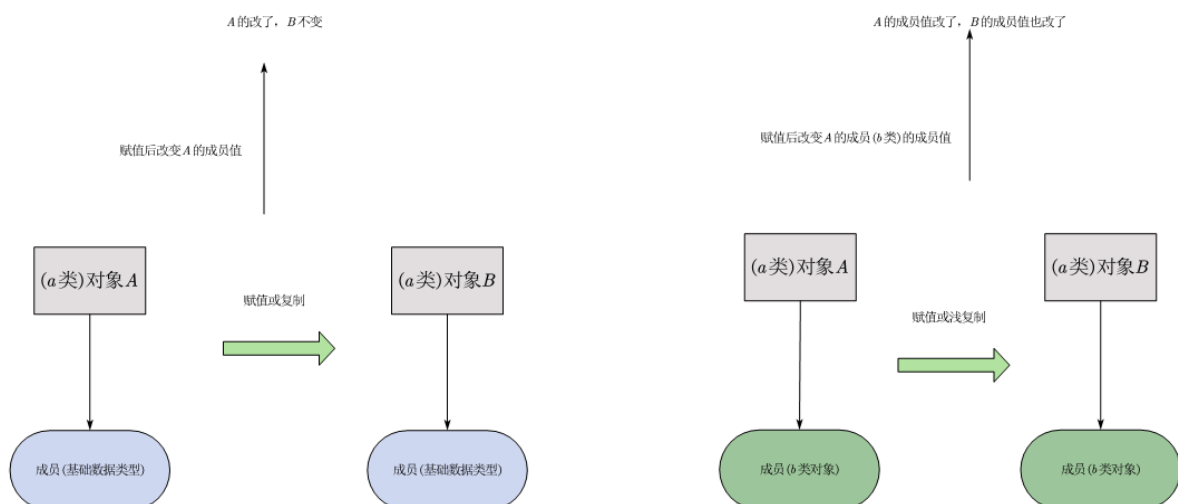
tensor他本身含有多个**对象**成员数据和多个类内方法，对象成员如有tensor.item(自身带的数值)，tensor.grad(**该tensor所带的计算图，重要！**)，这里需要细致理解一些python本身对象嵌套之间的关系逻辑。

计算图eg:

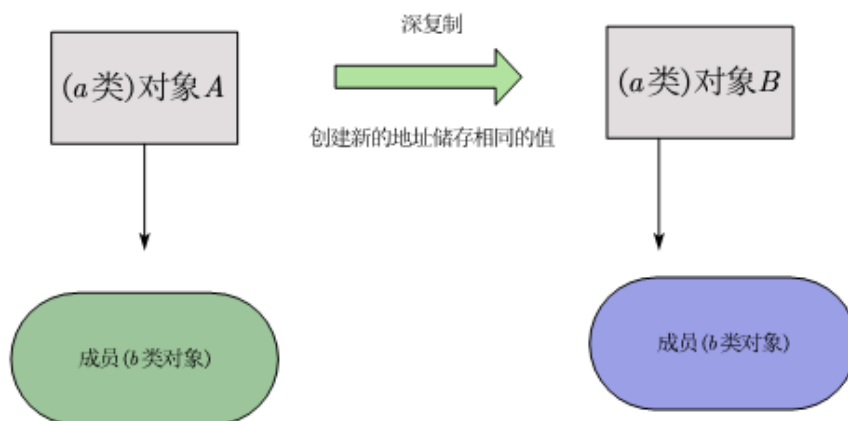
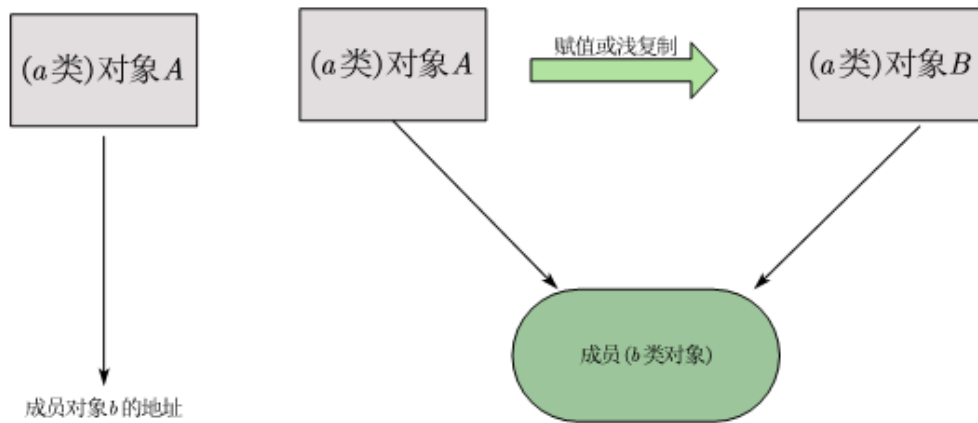


Python复合对象的逻辑:

```
1 from copy import deepcopy
2 #对象内对象的关系:
3
4 class main_class:
5     def __init__(self,value=0):
6         self.value=value
7
8 class abstract_class:
9     def __init__(self,data=main_class(0)):
10         self.data=data
11
12
13 main1=main_class(1)
14 main2=main_class()
15
16 ab1=abstract_class(main2)
17 print(ab1.data.value)
18 #现在ab1的数据是main2
19 print("现在ab1的数据是main2")
20 main2.value=1
21 print(ab1.data.value)
22 #ab1的data成员地址一直是指着作为对象的main2
23 print("ab1的data成员地址一直是指着作为对象的main2")
24 ab2=ab1
25 print(ab1.data==ab2.data)
26 main2.value=3
27 print(ab1.data.value)
28 print(ab2.data.value)
29 #我们普通的赋值, 对象地址并不会变, 仍然还是指向那个对象
30 print("我们普通的赋值, 对象地址并不会变, 仍然还是指向那个对象")
31 ab3=deepcopy(ab1)
32 print(ab3.data==ab1.data)
33 main2.value=4
34 print(ab1.data.value)
35 print(ab3.data.value)
36 #深层赋值, 对象内部的对象也重新创建了
37 print("深层赋值, 对象内部的对象也重新创建了")
38
```



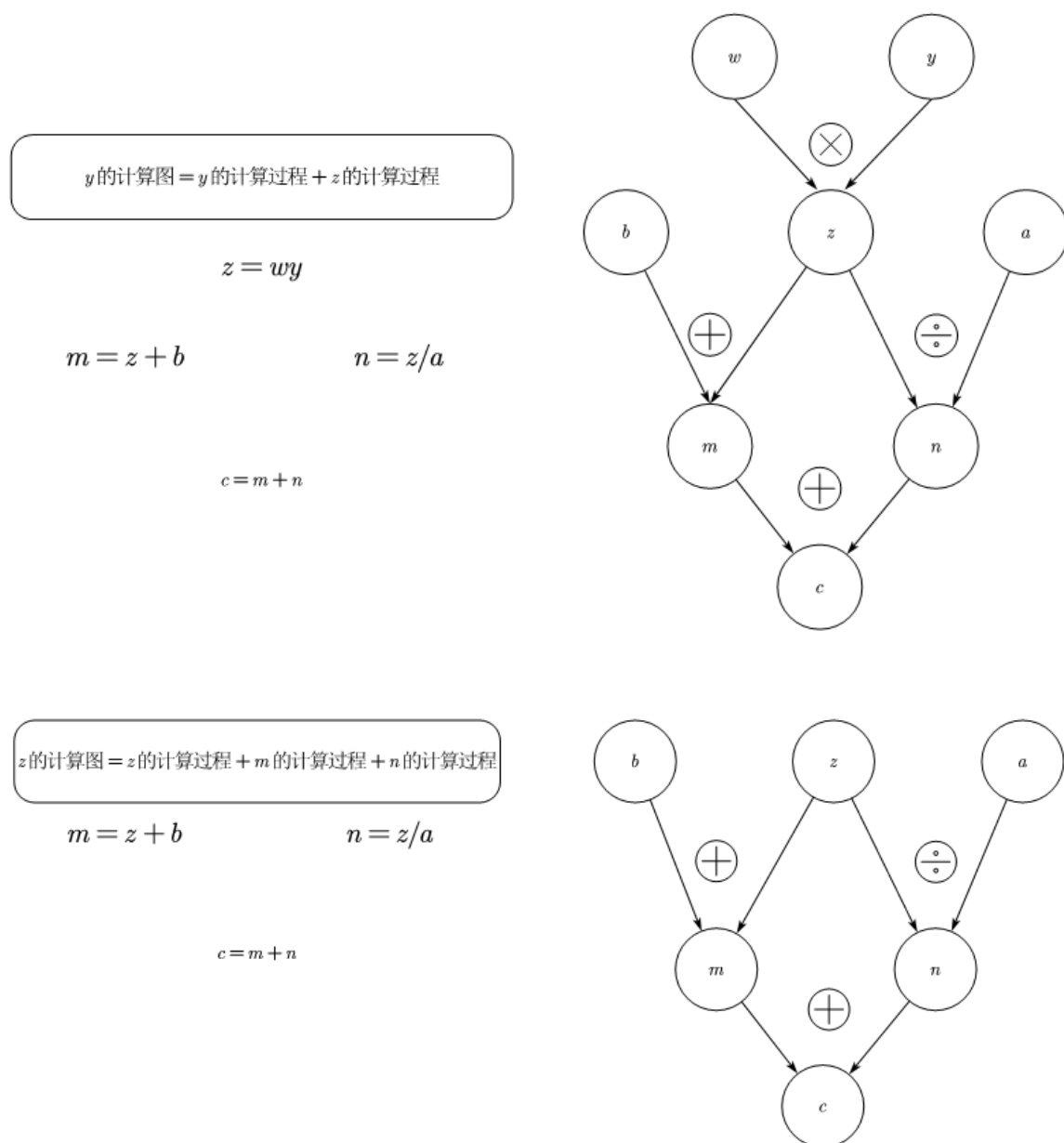
== 在判断两个对象的时候，是判断两个地址相同



自动微分

2.1.2 计算图:

pytorch（或者说所有的神经网络框架）带有自动微分的框架，他们的工作原理也大体一致，自动微分的过程总体上概括可以总结为，计算过程中，tensor会自动储存所有他参与过及由他计算出来的结果参与过的**计算过程**，这称之为计算图（其数据类型是一个对象），例如如下的案例：



y 的计算图是由 y 的计算过程和由 y 算出的变量 tensor 的计算过程组成，这是个递归的过程，**如果 z 是由 y 算出的，那么 z 的计算图就一定是 y 计算图的子图。**

2.1.3 算子，微分算子，与链式法则复合求导:

要仔细弄清楚自动微分的原理，我们需要仔细从理论上弄明白以下两种概念，和链式法则的含义：

泛函/函数：定义在**向量空间**到**标量**（如实数域）的映射。

算子：定义在**向量空间**到**向量空间**的映射。

我们其实已经接触过一类非常常见的算子，只不过可能还没有形成一个非常抽象本质的概念，那就是**微分算子**：

集合 \rightarrow 集合	映射	
向量空间 \rightarrow 标量(实数域)	泛函/函数	$x \in R^n \quad f \quad y \in R$
向量空间 \rightarrow 向量空间	算子	$x \in R^n \quad \nabla f \quad y \in R^n$
		$x \in R^n \quad \nabla^2 f \quad y \in R^{n^2}$

而对于**自动微分**这个问题，回忆一下，是不是我们可以通过**链式法则**，把复杂的**微分算子**分解成简单的微分算子表示？

定理2.1.3.a：设 F 是 R^n 到 R^m 的映射， g 是 R^m 到 R 的映射，则复合算子 $\nabla h(x) = \nabla F(x) \cdot \nabla g(F(x))$ ，其中 $h(x) = g(F(x))$ 。

proof:

$$F(x) = (f_1(x), f_2(x), \dots, f_m(x)), x \in R^n, x = (x_1, x_2, \dots, x_n)^T$$

$$g(y), y \in R^m, y = (y_1, y_2, \dots, y_m)^T, y = F(x)$$

$$\Rightarrow g(F(x)) = g(f_1(x), f_2(x), \dots, f_m(x))$$

$$\Rightarrow \frac{\partial g(F(x))}{\partial x_1} = \sum_{i=1}^m \frac{\partial g(y)}{\partial y_i} \cdot \frac{\partial y_i}{\partial x_1}$$

$$\Rightarrow \nabla h(x) = \begin{pmatrix} \frac{\partial g(F(x))}{\partial x_1} \\ \dots \\ \frac{\partial g(F(x))}{\partial x_n} \end{pmatrix} = \begin{pmatrix} \sum_{i=1}^m \frac{\partial g(y)}{\partial y_i} \cdot \frac{\partial y_i}{\partial x_1} \\ \dots \\ \sum_{i=1}^m \frac{\partial g(y)}{\partial y_i} \cdot \frac{\partial y_i}{\partial x_n} \end{pmatrix}$$

$$= \begin{pmatrix} \frac{\partial y_1}{\partial x_1} & \dots & \frac{\partial y_1}{\partial x_n} \\ \dots & \dots & \dots \\ \frac{\partial y_m}{\partial x_1} & \dots & \frac{\partial y_m}{\partial x_n} \end{pmatrix}^T \cdot \begin{pmatrix} \frac{\partial g(y)}{\partial y_1} \\ \dots \\ \frac{\partial g(y)}{\partial y_n} \end{pmatrix} = \nabla F(x) \nabla g(F(x))$$

这样我们就证明了微分算子之间的链式法则规律，如何对于自动微分，我们只需要设计基础代数运算和一些常见运算的微分算子，就能根据**计算图和链式法则**，复合出非常复杂的微分算子：

基础的代数微分算子	$G(x) = x + y$	$W(x) = x - y$
	$\nabla G = 1$	$\nabla W = 1$
	$J(x) = x^*y$	$L(x) = \frac{1}{x}$
	$\nabla J = y$	$\nabla L = -\frac{1}{x^2}$
	$P(x) = \exp(x)$	
	$\nabla P = \exp(x)$	

比如对于如下案例，我们要求复杂微分算子 $\nabla f_{\theta}(x)$ ，我们可以通过计算图和链式法则将复杂的微分算子用简单的常见微分算子表示出来：

$$f_{\theta}(x) \quad \nabla f_{\theta}(x) = g(\nabla G(x), \nabla P(x), \dots)$$

g 是由我们计算图和链式法则确定的

$$f_{\theta}(x) = a * \exp((\theta x) + b)$$

$$\nabla_{\theta} f =$$

$$h(\theta) = \theta x$$

$$c(h) = h + b$$

$$z(c) = \exp(c)$$

$$final(z) = a * \exp((\theta x) + b) = a * z$$

假设空间抽象定义/建模

2.1.4 模组：

```
1 torch.nn.Module
```

`nn.Module`是pytorch里面关于模型的基本组件，我们可以通过`nn.Module`定义我们的参数空间 Θ 和我们的参数化过程 p ，以此来得到定义假设空间 \mathcal{H} 的效果。在`nn.Module`其中有另外一个非常重要的对象数据类型叫做：

```
1 torch.nn.parameter
```

这个是我们定义的模型可优化（可学习）参数 θ ，我们建模的本质是通过追溯计算图计算对所有可优化参数的梯度进行数值优化，所有继承`nn.Module`的子类都可以通过：

```
1 | module.parameters()
```

迅速递归式调用其成员内所有的参数对象（即`nn.Parameter`），在后面的操作中，我们把`nn.Parameter`传入优化器（optimizer）即可以进行参数优化问题的求解。（Parameter的本质是一个默认带计算追溯功能的tensor（即其`requires_grad_()`默认设置成True）。）

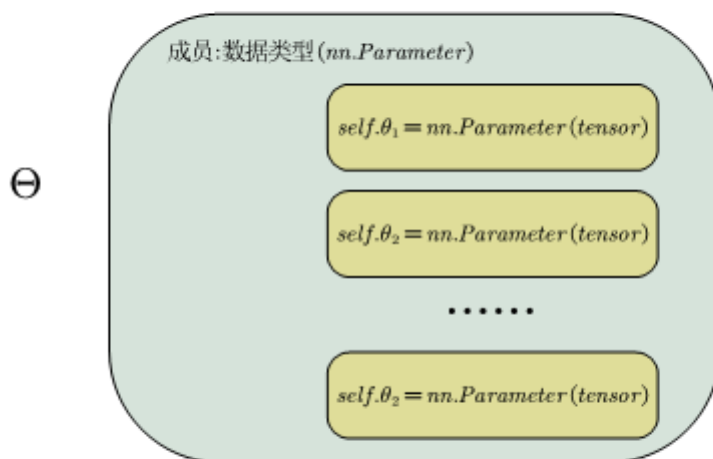
我们最基础定义`nn.Module`的一个子类的方法是，定义其所有的参数成员（`nn.Parameter`），然后定义所有参数成员参数化假设空间的过程（对于神经网络来说就是前向传播计算）

一个模块 M /假设空间 \mathcal{H} 最基础的定义方式：

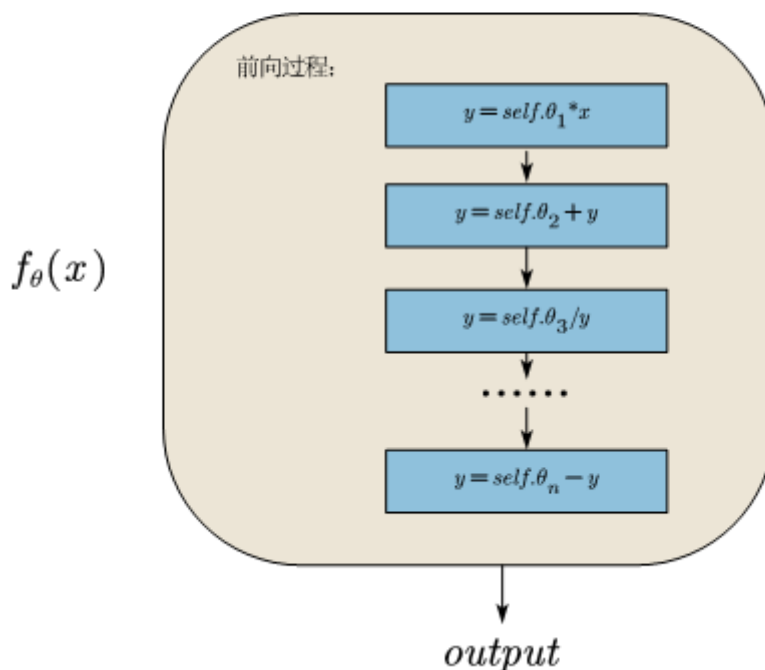
Class Mymodel

nn.Module

初始化阶段: `def __init__`: (定义有哪些参数 θ)



参数化过程阶段: `def forward(self,x)`: (定义参数 θ 怎么参数化一个函数)



eg: $\mathcal{H}_1 : \{f|f(x) = ax + b, a \in R, b \in R\}$ 定义了所有直线构成的假设空间。

```
1 import torch
2 from torch import nn
3 class my_Hypothesis(torch.nn.Module):
4     def __init__(self):
5         super().__init__()
6         self.w=torch.nn.Parameter(torch.tensor(2.1))
7         self.b=torch.nn.Parameter(torch.tensor(2.1))
8     def forward(self, x):
9         z=self.w*x
10        y=z+self.b
11        return y
12
13 model=my_Hypothesis()
14 print(model)
```

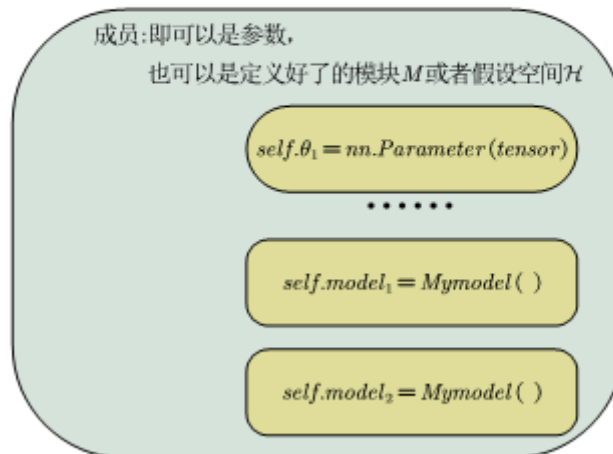
而我们一般实际中遇到的模型都是复合式Module子类定义方式，我们会在Module子类中的成员中定义一个实例化的Module子类。而在参数化的过程中复合嵌套参数化过程。

更常见的高阶模块 M /假设空间 \mathcal{H} 定义方式：

Class Mymodel_Advance

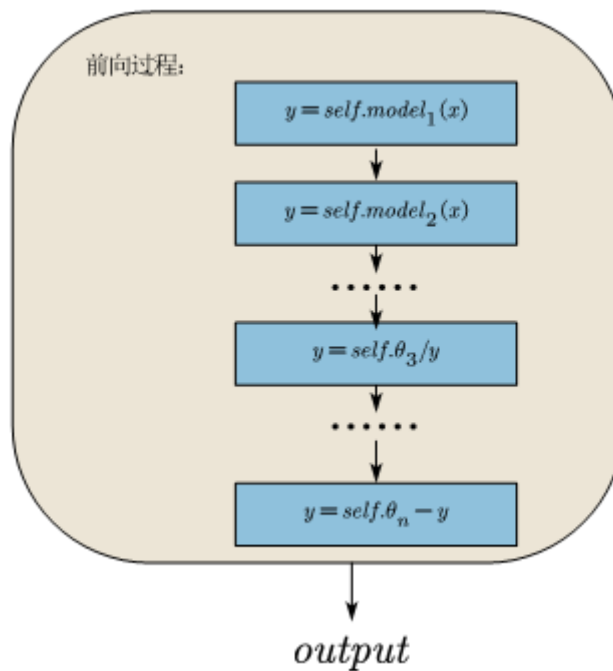
nn.Module

初始化阶段: `def __init__`: (定义有哪些参数 θ)



参数化过程阶段: `def forward(self,x):`

(定义参数 θ , 和实例化的 $Module$ 子类怎么复合参数化一个函数)



eg:这里复合嵌套了两个假设空间 $\mathcal{H}_2 : \{f|f(x) = g(x) + b, b \in R, g \in \mathcal{H}_3\}$,
 $\mathcal{H}_3 : \{g|g(x) = ax, a \in R\}$,与前文案例相同, 同样定义了所有直线构成的假设空间。

```
1 class my_Hypothesis_slope(torch.nn.Module):
2     def __init__(self):
3         super().__init__()
4         self.w=torch.nn.Parameter(torch.tensor(2.1))
5     def forward(self, x):
6         z=self.w*x
7         return z
8
9 class my_Hypothesis_line2(torch.nn.Module):
10     def __init__(self):
11         super().__init__()
12         self.slope=my_Hypothesis_slope()
```

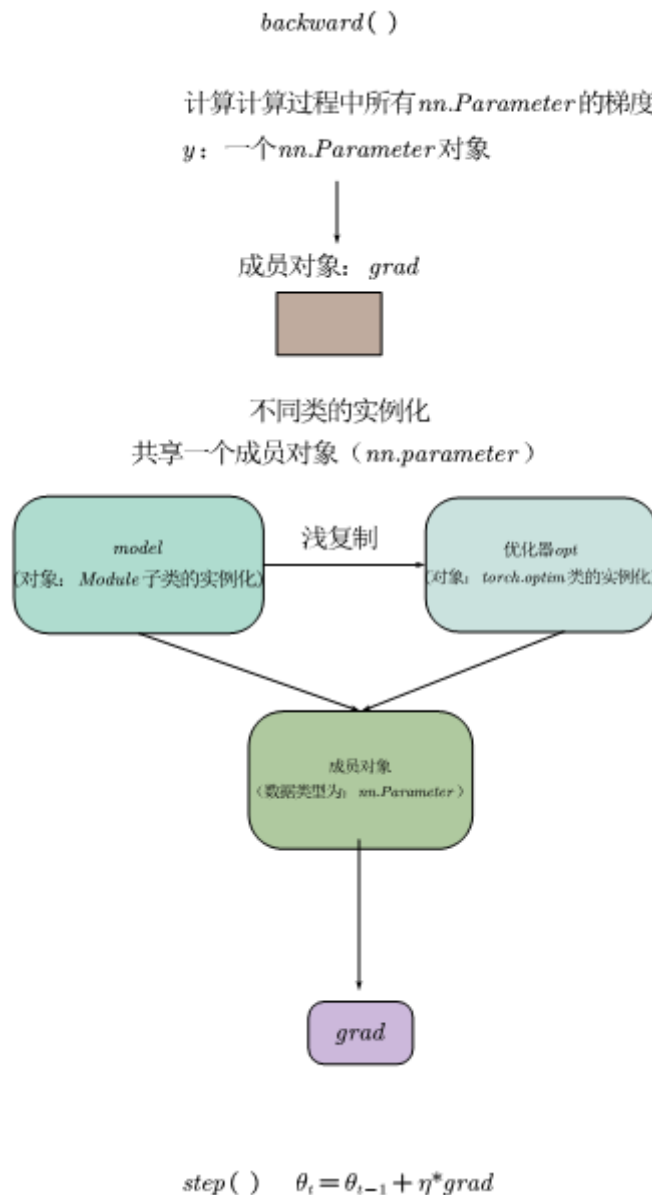
```

13     self.b = torch.nn.Parameter(torch.tensor(2.1))
14     def forward(self, x):
15         z=self.slope(x)
16         y=z+self.b
17         return y

```

假设空间上的优化:

我们通过将模型的可优化参数（nn.Parameter）对象，在实例化优化器（即调用优化器初始化方法__init__时）时将模型的参数成员对象浅复制给优化器，来使得优化器可以操作对可学习参数的数值优化。



以下为用我们前文定义的直线假设空间，取进行解决最接近 $y = x$ 的变分问题:

```

1  import torch
2  from torch import nn
3  class my_Hypothesis(torch.nn.Module):
4      def __init__(self):
5          super().__init__()
6          self.w=torch.nn.Parameter(torch.tensor(2.1))

```

```
7         self.b=torch.nn.Parameter(torch.tensor(2.1))
8     def forward(self, x):
9         z=self.w*x
10        y=z+self.b
11        return y
12
13    model=my_Hypothesis()
14    x1=torch.tensor(1)
15    x2=torch.tensor(2)
16    y1=torch.tensor(1)
17    y2=torch.tensor(2)
18    n=1000
19    opt=torch.optim.SGD(model.parameters(),lr=0.1)
20    for epoch in range(n):
21        opt.zero_grad()
22        loss=(model(x1)-y1)**2+(model(x2)-y2)**2
23        loss.backward()
24        opt.step()
25        print(loss)
26    for param in model.named_parameters():
27        print(param)
```