

# Regularization and Optimization for Deep learning

张晓龙 – 2017-10-4

# 目录

1. 机器学习基本概念回顾
2. 正则化 ( Regularization )
3. 模型优化 ( Model Optimization )

## What is Machine Learning?

- Machine Learning is a field of study that computers the ability to learn without being explicitly programmed  
-- Arthur Samuel (1959)
- A computer program is said to learn from **experience E** with respect to some class of **tasks T** and performance **measure P**, if its performance at tasks in T, as measured by P, improves with experience E.  
-- Tom Mitchell (1988)
- Machine learning is the study of computer algorithms that improve automatically through experience  
-- Tom Mitchell (1998)

# tasks T、experience E、measure P

- Task T: 分类 ( $f: R^n \rightarrow \{1, \dots, k\}$ )、聚类、回归 ( $f: R^n \rightarrow R$ )、转录（表达、*word2Vec*）、机器翻译、结构化输出、异常检测（信用卡检测、阿里支付）、合成和采样（*gan*、音频）、缺失值填补、去噪（马赛克、条件概率分布）、密度估计或概率质量函数估计
- Measure P: 评估学习算法的能力，设计其性能的度量（准确率、样本上概率对数的平均值），P是特定于系统执行的任务T的
- Experience E:
  - 数据集、样本、特征
  - 无监督算法：概率分布  $P(x)$
  - 监督算法：估计  $P(y|x)$

# 构建机器学习算法

- 机器学习算法：特定的数据集、代价函数、优化过程和模型
- Ex: 线性回归算法组成成分有 $\mathbf{x}$ 和 $y$ 构成的数据集，损失函数是

$$J(\mathbf{w}, b) = -\mathbb{E}_{\mathbf{x}, y \sim \hat{p}_{\text{data}}} \log p_{\text{model}}(y \mid \mathbf{x}),$$

- 模型是  $p_{\text{model}}(y \mid \mathbf{x}) = \mathcal{N}(y; \mathbf{x}^\top \mathbf{w} + b, 1)$ ，在多数情况下优化算法可以定义为求解损失函数梯度为0的解
- 最常见的损失函数是负对数似然，最小化损失函数导致的最大似然估计
- 损失函数有可能有附加项，如正则化，

$$J(\mathbf{w}, b) = \lambda \|\mathbf{w}\|_2^2 - \mathbb{E}_{\mathbf{x}, y \sim \hat{p}_{\text{data}}} \log p_{\text{model}}(y \mid \mathbf{x}).$$

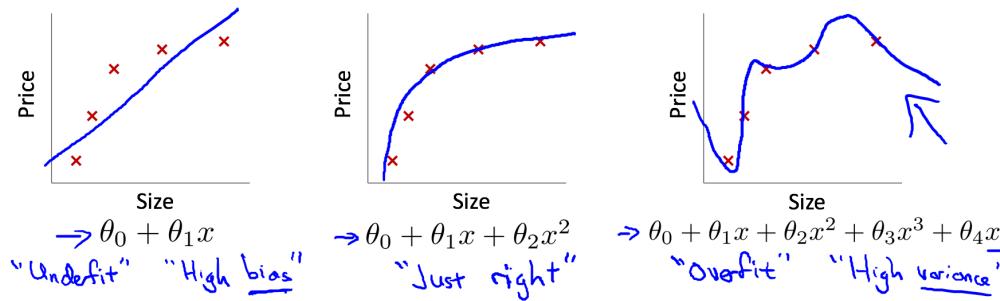
- 非线性的模型，需要选择迭代数值优化过程（SGD.et）

# 机器学习领域中心问题

1. Regularization ( 正则化 )
2. Model Optimization ( 模型优化 )

- 为什么需要正则化? - 防止过拟合

Example: Linear regression (housing prices)



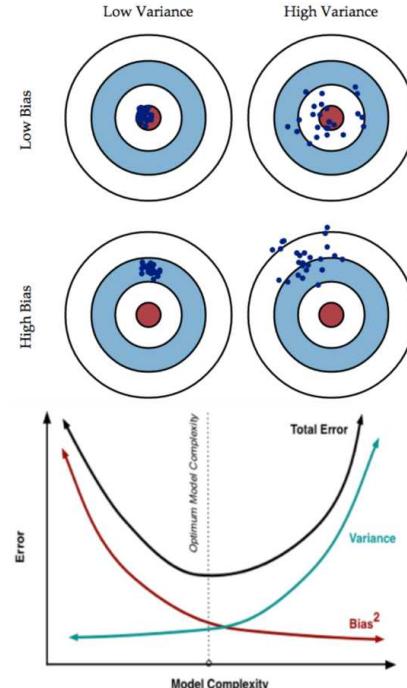
**Overfitting:** If we have too many features, the learned hypothesis may fit the training set very well ( $J(\theta) = \frac{1}{2m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)})^2 \approx 0$ ), but fail to generalize to new examples (predict prices on new examples).

- 什么是正则化? 目的是什么?
  - 对学习算法的一些修改策略----旨在减少泛化误差而不是训练误差
- 正则化策略会对估计进行正则化
- 估计的正则化以偏差的增加换取方差的减少

# Bias(偏差), Error(误差), 和Variance(方差)

- $\text{Error} = \text{Bias} + \text{Variance} + \text{Noise}$
- bias描述的是根据样本拟合出的模型的输出预测结果的期望与样本真实结果的差距
- 要想在bias上表现好, low bias, 就是复杂化模型 (过拟合)
- 要想在variance上表现好, low variance, 就要简化模型, 减少模型的参数 (欠拟合)
- bias和variance的选择是一个tradeoff

准与确



# Regularization ( 正则化策略 )

- 1. 参数范数惩罚(L1、L2)
- 2. 多任务学习
- 3. 提前终止
- 4. 参数绑定和参数共享
- 5. 稀疏表示
- 6. Bagging等集成方法
- 7. Dropout
- 8. 对抗训练
- 9. 切面距离、正切传播和流行正切分类器



提高泛化能力，减少泛化误差

# 参数范数惩罚 (L1范数, L2范数)

- 通过对目标函数J添加一个参数范数惩罚 $\Omega(\theta)$

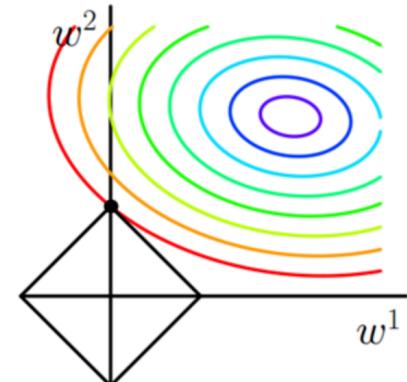
$$\tilde{J}(\theta; \mathbf{X}, \mathbf{y}) = J(\theta; \mathbf{X}, \mathbf{y}) + \alpha \Omega(\theta),$$

- L1正则化和L2正则化可以看做是损失函数的惩罚项
  - L1正则化是指权值向量w中各个元素的绝对值之和，通常表示为 $\| w \|_1$
  - L2正则化是指权值向量w中各个元素的平方和然后再求平方根，通常表示为 $\| w \|_2$
- L1和L2正则化的作用
  - L1正则化可以产生稀疏权值矩阵，即产生一个稀疏模型，可以用于特征选择
  - L2正则化可以防止模型过拟合（overfitting）；一定程度上，L1也可以防止过拟合

# L1正则化和特征选择

- 带L1正则化的损失函数

$$J = J_0 + \alpha \sum_w |w|$$

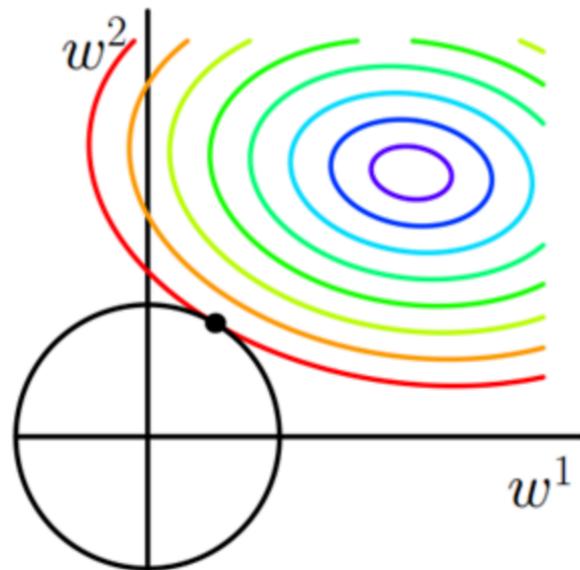


- 在原始损失函数 $J_0$ 后添加L1正则化项时，相当于对 $J_0$ 做了一个约束。
- 令 $L = \alpha \sum_w |w|$ ，则 $J = J_0 + L$ ，此时我们的任务变成在 $L$ 约束下求出 $J_0$ 取最小值的解
- 二维的情况(只有两个权值 $w^1$ 和 $w^2$ )，求解 $J_0$ 的过程可以画出等值线，L1正则化的函数 $L$ 在二维平面上画出
- 当 $J_0$ 等值线与 $L$ 图形首次相交的地方就是最优解，顶点的值是 $(w^1, w^2) = (0, w)$
- 正则化前面的系数 $\alpha$ ，可以控制 $L$ 图形的大小。 $\alpha$ 越小， $L$ 的图形越大

- 带L2正则化的损失函数

$$J = J_0 + \alpha \sum_w w^2$$

- 二维平面下L2正则化的函数图形是个圆
- $J_0$ 与L相交时使得 $w_1$ 或 $w_2$ 等于零的机率小许多
- 为什么L2正则化不具有稀疏性的原因



# L2正则化和过拟合

- 拟合过程中通常都倾向于让权值尽可能小，最后构造一个所有参数都比较小的模型
- 线性回归中的梯度下降法为例，线性回归的代价函数如下

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)})^2$$

- 在梯度下降法中，最终用于迭代计算参数 $\theta$ 的迭代式为

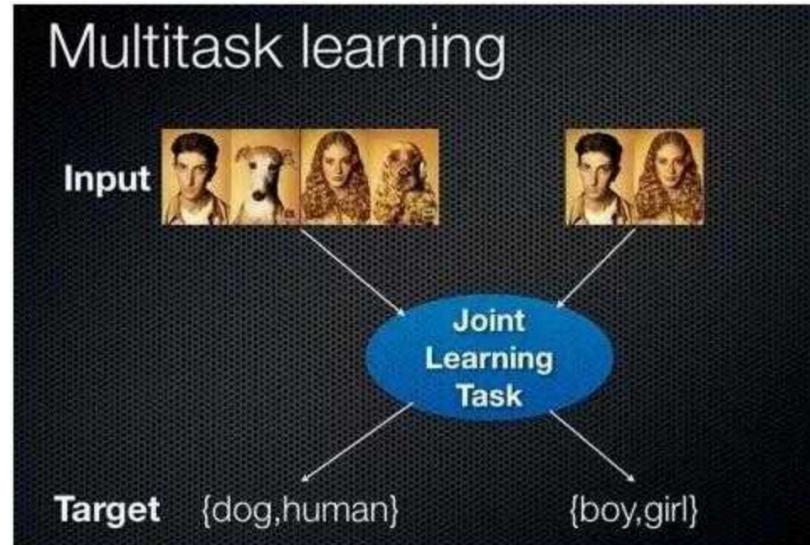
$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

$$\theta_j := \theta_j \left(1 - \alpha \frac{\lambda}{m}\right) - \alpha \frac{1}{m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

- 每一次迭代， $\theta_j$ 都要先乘以一个小于1的因子，从而使得 $\theta_j$ 不断减小，因此总体来看， $\theta$ 是不断减小的（权重衰减收缩的）

# 多任务学习

- 多任务学习是一种归纳迁移机制基本目标是提高泛化性能。
- 多任务学习在学习一个问题的同时，可以通过使用共享表示来获得其他相关问题的知识
- 多任务学习的基本假设是多个任务之间具有相关性，因此能够利用任务之间的相关性互相促进。
- 多任务深度学习已经广泛应用于人脸识别、细粒度车辆分类、面部关键点定位与属性分类等多个领域



# 提前终止

- Early stopping是一种迭代次数截断的方法来防止过拟合的方法，即在模型对训练数据集迭代收敛之前停止迭代来防止过拟合。
- 具体算法（“No-improvement-in-n”）：
  - 在每一个Epoch结束时计算validation data的accuracy，当accuracy不再提高时，就停止训练。
  - 怎样才认为validation accuracy不再提高？
    - 在训练的过程中，记录到目前为止最好的validation accuracy
    - 当连续10次Epoch（或者更多次）没达到最佳accuracy时，可以认为accuracy不再提高。此时便可以停止迭代了

# 参数绑定、参数共享

- 全连接层缺点是参数很多，卷积层可以减少参数，减少计算量，因为卷积层的参数共享特性。
- 卷积神经网络中的参数共享
  - CNN一个牛逼的地方就在于通过感受野和权值共享减少了神经网络需要训练的参数的个数。
  - <http://blog.csdn.net/zouxy09/article/details/8781543>
  - <https://www.zhihu.com/question/47158818>

# 稀疏表示

- 稀疏表示的本质：用尽可能少的资源表示尽可能多的知识，这种表示还能带来一个附加的好处，即计算速度快。
- 稀疏表示就是指训练出的参数是稀疏的，即包含很多的0
- 适应性（泛化能力）和稀疏性之间找平衡，最优取决于代价函数。
- 稀疏表达的意义在于降维
- 这种降维主要表现于虽然原始信号 的维度很高，但实际的有效信息集中在一个低维的空间里。这种性质使得不适定的问题变得适定(well-posed)，进而获得“好的解”成为可能

- ReLU - 神经网络的一种激活函数，可以做到稀疏表示

$$f(x) = \max(0, x + N(0, 1)) \quad f(x) = \log(1 + e^x)$$

- L0、L1、L2

# Bagging、集成方法

- Bagging结合多个模型降低泛化误差的技术
  - 分别训练几个不同模型，所有模型表决测试样例输出，称为模型平均，采用这种策略的技术称为集成方法
- 模型平均有效的原因
  - 不同模型通常不会在测试集上产生完全相同的误差
- Bagging是一种允许重复多次使用同一种模型、训练算法和目标函数的方法
  - Bagging涉及构造  $k$  个不同的数据集。每个数据集从原始数据集中重复采样构成，和原始数据集具有相同数量的样例。

# Bagging、集成方法

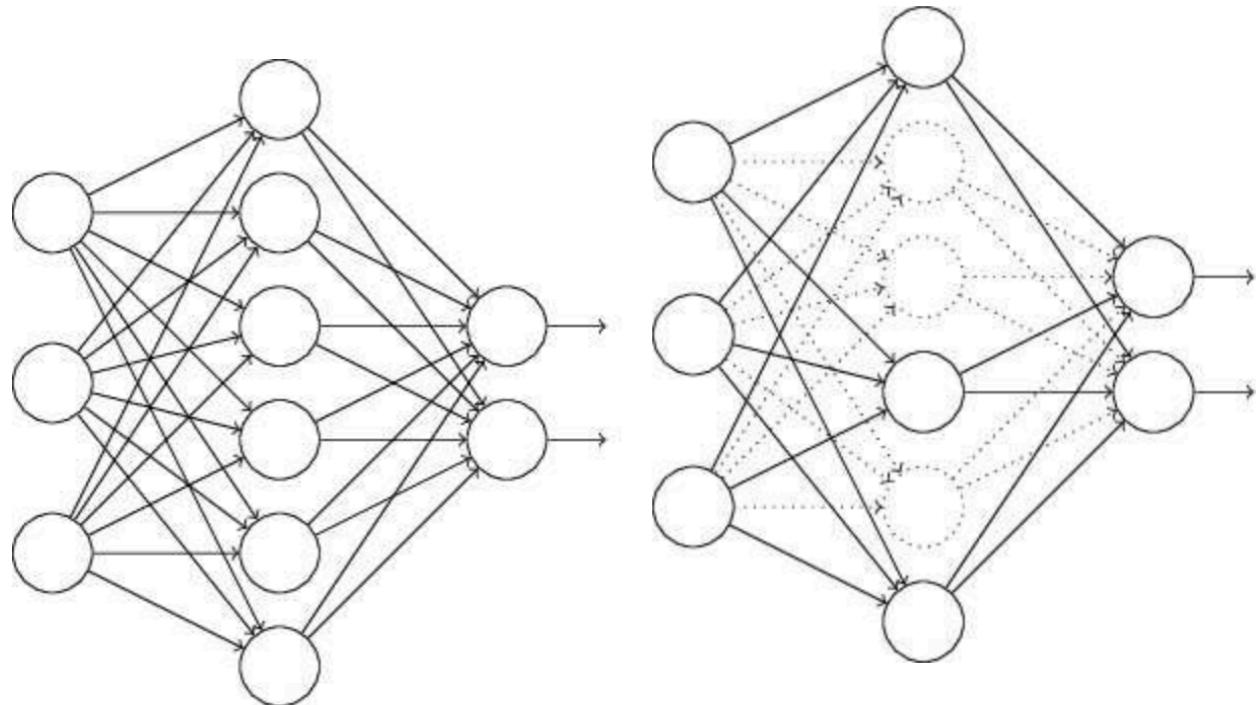


# Dropout

- Dropout是hinton最近2年提出的
- 源于其文章*Improving neural networks by preventing co-adaptation of feature detectors*（通过阻止特征检测器的共同作用来提高神经网络的性能）
- L1、L2正则化是通过修改代价函数来实现的
- Dropout则是通过修改神经网络本身来实现的，它是在训练网络时用的一种技巧（trick）

# Dropout

- 一次迭代的过程
- 在训练开始时，我们随机地“删除”一半的隐层单元，视它们为不存在
- 保持输入输出层不变，按照BP算法更新上图神经网络中的权值



# Dropout

- Dropout优点
  - 计算方便。训练过程中使用Dropout产生  $n$  个随机二进制数与状态相乘即可。每个样本每次更新的时间复杂度:  $O(n)$ , 空间复杂度:  $O(n)$ 。
  - 适用广。Dropout不怎么限制适用的模型或训练过程, 几乎在所有使用分布式表示且可以用随机梯度下降训练的模型上都表现很好。包括: 前馈神经网络、概率模型、受限玻尔兹曼机、循环神经网络等。
  - 相比其他正则化方法(如权重衰减、过滤器约束和稀疏激活)更有效。也可与其他形式的正则化合并, 得到进一步提升。
- Dropout缺点
  - 不适合宽度太窄的网络。否则大部分网络没有输入到输出的路径。
  - 不适合训练数据太小(如小于5000)的网络。训练数据太小时, Dropout没有其他方法表现好。
  - 不适合非常大的数据集。数据集大的时候正则化效果有限(大数据集本身的泛化误差就很小), 使用Dropout的代价可能超过正则化的好处。

# 对抗训练

- 深度学习中对抗样本问题
  - 数据集中通过故意添加细微的干扰所形成的输入样本，受干扰之后的输入导致模型以高置信度给出一个错误的输出
- 深度学习的对抗训练
  - 所谓深度学习对抗训练，就是通过在对抗样本上训练模型
- Ian Goodfellow: 对抗样本是一个非常难的问题

# 对抗训练

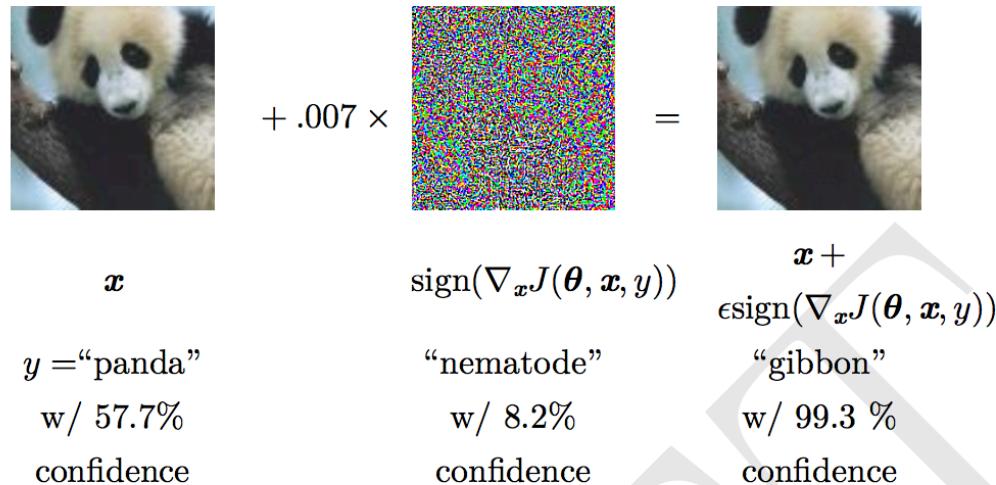


图 7.8: 在 ImageNet 上应用于 GoogLeNet (Szegedy *et al.*, 2014a) 的对抗样本生成的演示。通过添加一个不可察觉的小向量 (其中元素等于代价函数相对于输入的梯度元素的符号), 我们可以改变 GoogLeNet 对此图像的分类结果。经 Goodfellow *et al.* (2014b) 许可转载。

# 切面距离、正切传播和流行正切分类器

- 流形 (manifold) 指连接在一起的区域
  - 数学上, 它是指一组点, 且每个点都有其邻域。给定一个任意的点, 其流形局部看起来像是欧几里得空间
  - 每个点周围邻域的定义取决于其位置

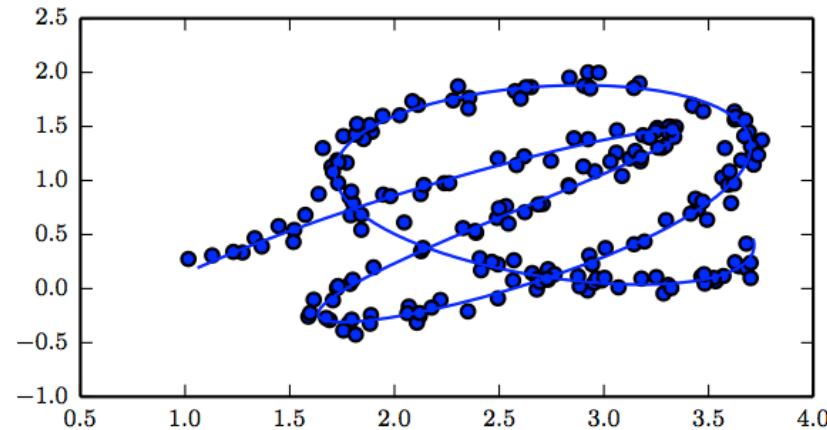


图 5.11: 从一个聚集在一维流形的二维空间的分布中抽取的数据样本, 像一个缠绕的带子一样。线代表了学习者想要推断的隐含的流形。

- 机器学习算法学习  $\mathbb{R}^n$  上的所有感兴趣的函数，很多机器学习问题看上去都是不可解的。
- 流形学习 (manifold learning) 算法通过一个假设来克服这个障碍
  - 该假设认为  $\mathbb{R}^n$  中大部分区域都是无效的输入，感兴趣的输入只分布在包含少量点的子集构成的一组流形中，而学习函数中感兴趣输出的变动只位于流形中的方向，或者感兴趣的变动只发生在我们从一个流形移动到另一个流形的时候
- 关键假设仍然是概率质量高度集中

- 一个利用流形假设的早期尝试是切面距离 (tangent distance) 算法 (Simard *et al.*, 1993, 1998)。
- 正切传播 (tangent prop) 算法 (Simard *et al.*, 1992) 训练 带有额外惩罚的神经网络分类器，使神经网络的每个输出  $f(x)$  对已知的变化因素是局部不变的。这些变化因素对应于沿着的相同样本聚集的流形的移动。
- 局部不变性 是通过要求  $\nabla_x f(x)$  与已知流形的切向  $v(i)$  正交实现的，或者等价地通过正则化惩罚  $\Omega$  使  $f$  在  $x$  的  $v(i)$  方向的导数是小的：

$$\Omega(f) = \sum_i \left( (\nabla_x f(\mathbf{x})^\top \mathbf{v}^{(i)}) \right)^2.$$

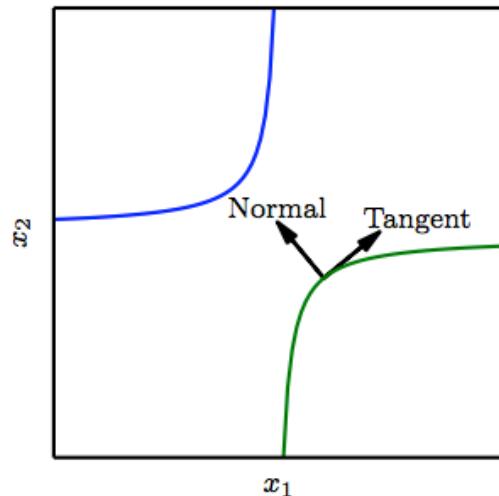


图 7.9: 正切传播算法 (Simard *et al.*, 1992) 和流形正切分类器主要思想的示意图 (Rifai *et al.*, 2011c)，它们都正则化分类器的输出函数  $f(\mathbf{x})$ 。每条曲线表示不同类别的流形，这里表示为嵌入在二维空间中的一维流形。在一条曲线上，我们选择单个点并绘制了一个与类别流形（平行并接触流形）相切的向量以及与类别流形（与流形正交）垂直的向量。在多维情况下，可以存在许多切线方向和法线方向。我们希望分类函数沿垂直于流形方向上的移动而快速改变，并且不随着沿类别流形方向的移动而改变。正切传播和流形正切分类器都正则化  $f(\mathbf{x})$  不随  $\mathbf{x}$  沿流形的移动而大量改变。正切传播需要用户手动指定计算正切方向的函数（例如指定小平移后的图像保留在相同类别的流形中），而流形正切分类器通过训练自编码器拟合训练数据来估计流形的正切方向。我们将在第十四章中讨论使用自编码器来估计流形。



好课就在跟谁学