人工智能与深度学习

3-2 正则化

王中雷

经济学院、王亚南经济研究院、邹至庄经济研究院

厦门大学

（第一版）

内容摘要

1. 显式正则化

2. 隐式正则化

3.噪声注入

4. 集成学习

直观

1．神经网络模型的参数规模往往较大

- 复杂模型往往会对训练集进行过拟合
- 降低模型泛化能力
- 可能导致灾难性的后果

2．当然，我们可以选择更为简单的神经网络模型

- 然而，简单的模型，往往对数据进行欠拟合

3．我们希望，在不改变模型框架的基础上，降低模型复杂度

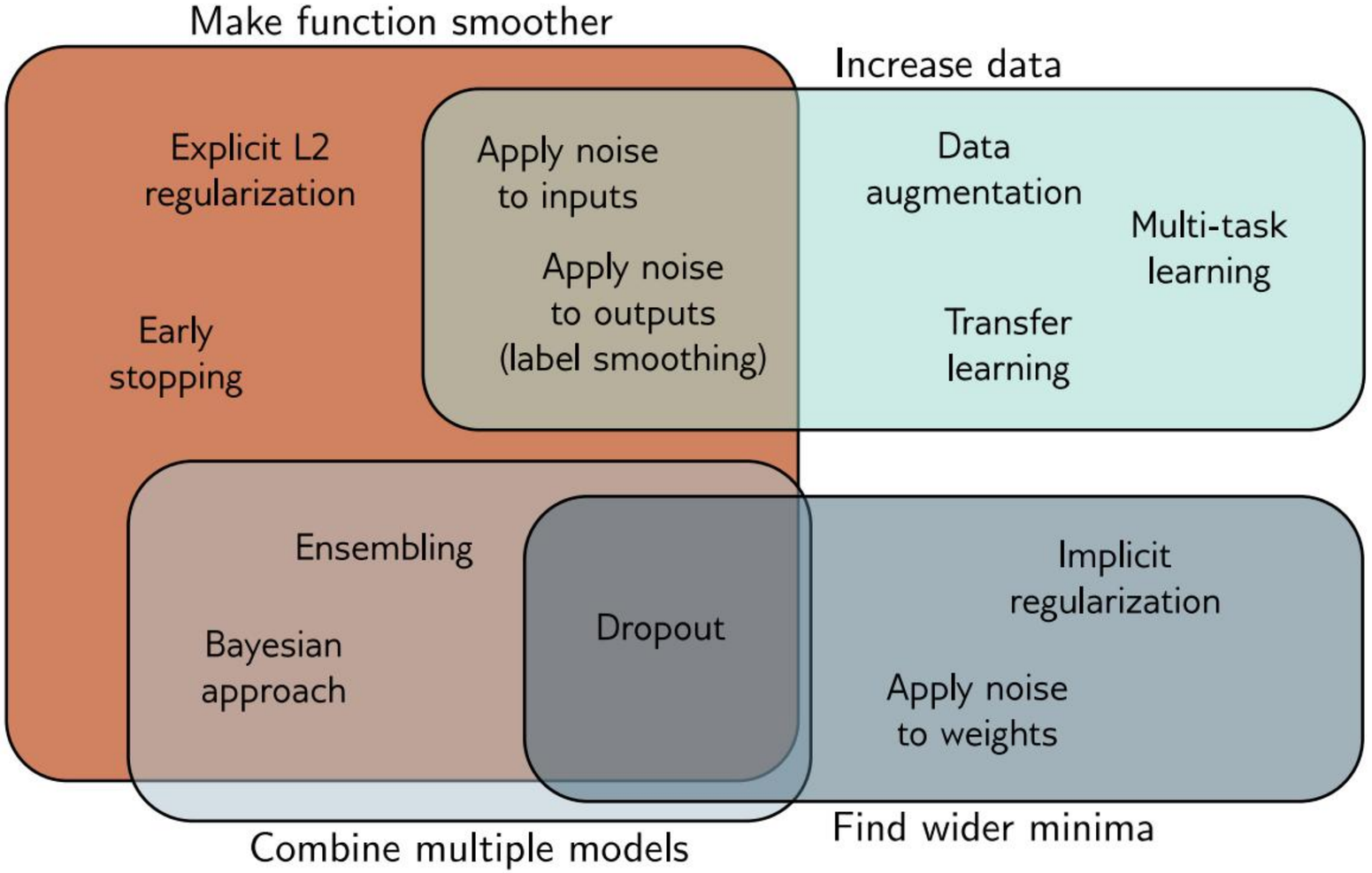
直观

1．在深度学习领域中，正则化往往包括

- 显式正则化（惩罚项）
- 隐式正则化（梯度下降算法、早停、dropout）
- 噪声注入（在训练集或训练过程中实施）
- 集成学习（bagging, boosting, stacking）

直观

1. 下图来自于 Prince (2024) 的图 9.14



显式正则化

1．形式：

$$\mathcal{J} = \mathcal{J}_1 + \mathcal{J}_2$$

- $\mathcal{J}_1 = \mathcal{J}_1(\boldsymbol{\theta})$：之前讨论的代价函数，比如交叉熵
- $\mathcal{J}_2 = \mathcal{J}_2(\boldsymbol{\theta})$：施加在模型参数上的惩罚项，比如 l_2 或者 l_1 惩罚项

2．为了更新模型参数，我们需要

$$\frac{\partial \mathcal{J}}{\partial \boldsymbol{\theta}} = \frac{\partial \mathcal{J}_1}{\partial \boldsymbol{\theta}} + \frac{\partial \mathcal{J}_2}{\partial \boldsymbol{\theta}}$$

- 第一项偏导数，可通过后向传播得到
- 第二项偏导数，取决于惩罚项的形式

显式正则化

1. l_2-惩罚项（岭回归问题或者 Tikhonov 正则化）

$$\mathcal{J}_2 = \frac{\lambda}{2n} \sum_{l=1}^{L} \|\boldsymbol{W}^{[l]}\|_F^2$$

- $\lambda > 0$: 超参数
- $\|\boldsymbol{A}\|_F = (\sum_i \sum_j a_{ij}^2)^{1/2}$: 矩阵$\boldsymbol{A} = (a_{ij})$ 的 Frobenius 范数
- 偏导数结果

$$\frac{\partial \mathcal{J}_2}{\partial \boldsymbol{b}^{[l]}} = 0, \quad \frac{\partial \mathcal{J}_2}{\partial \boldsymbol{W}^{[l]}} = \frac{\lambda}{n} \boldsymbol{W}^{[l]} \quad (l = 1, \ldots, L)$$

显式正则化

1. 给定学习率 α，

   - 偏置项 $\{\boldsymbol{b}^{[l]}: l=1,\ldots,L\}$ 的更新过程，不受惩罚项的影响
   - 权重矩阵 $\{\boldsymbol{W}^{[l]}: l=1,\ldots,L\}$ 的更新，会涉及一个新的梯度项

2. 权重矩阵的更新过程为

$$
\begin{aligned}
\boldsymbol{W}^{[l]} &\leftarrow \boldsymbol{W}^{[l]} - \alpha \cdot \left(\frac{\partial \mathcal{J}_1}{\partial \boldsymbol{W}^{[l]}} + \frac{\partial \mathcal{J}_2}{\partial \boldsymbol{W}^{[l]}} \right) \\
&= \boldsymbol{W}^{[l]} - \alpha \cdot \left(\frac{\partial \mathcal{J}_1}{\partial \boldsymbol{W}^{[l]}} + \frac{\lambda}{n} \boldsymbol{W}^{[l]} \right) \\
&= \left(1 - \frac{\alpha \cdot \lambda}{n} \right) \cdot \boldsymbol{W}^{[l]} - \alpha \cdot \frac{\partial \mathcal{J}_1}{\partial \boldsymbol{W}^{[l]}}
\end{aligned}
$$

3. 权重衰减正则化：首先缩小权重矩阵，再根据梯度，对模型参数进行更新

关于权重衰减正则化的备注

1. 对于经典梯度下降算法，权重衰减正则化与 l_2 正则化等价
2. 对于自适应梯度下降法（如 Adam），这两种正则化方法不等价 （Loshchilov and Hutter, 2019)
3. 考虑下面这个实际任务
   - 实现权重衰减正则化
   - 用 Adam 优化器，更新模型参数
4. 为了达到上述效果，一个直观的想法是考虑
   - 考虑代价函数：$\mathcal{J} = \mathcal{J}_1 + \mathcal{J}_2$
   - 采用经典 Adam 优化器进行优化
5. 然而，以上的想法并不能达到我们预期的效果

关于权重衰减正则化的备注

1. 问题出在哪里呢?
   - 认真观察这个代价函数：$\mathcal{J} = \mathcal{J}_1 + \mathcal{J}_2$
   - l_2 惩罚项的梯度也会受到自适应缩放，不再等价于对所有权重实施统一的乘法衰减
   - 导致自适应梯度下降法（如 Adam），在一些任务中表现不佳
2. 为了解决这个问题，Loshchilov 和 Hutter (2019) 对 Adam 优化器进行了改进，并提出了 AdamW (Adam with decoupled Weight decay) 算法
   - 我们只需稍微修改 Adam，便可得到 AdamW 优化器
   - AdamW 将权重衰减与损失梯度更新解耦
   - 对于诸多任务而言，AdamW 已经成为默认优化器

AdamW

1. 目标：在模型训练过程中，实现权重衰减正则化

   - 对于 Adam with l_2-regularization，考虑代价函数：$\mathcal{J} = \mathcal{J}_1 + \mathcal{J}_2$
   - 对于 AdamW，考虑代价函数 $\mathcal{J}_1$，在参数更新过程中，实施权重衰减正则化

2. 以下算法为 Loshchilov 和 Hutter (2019) 的 Algorithm 2

Algorithm 2 Adam with L_2 regularization and Adam with decoupled weight decay (AdamW)

given $\alpha = 0.001, \beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}, \lambda \in \mathbb{R}$
initialize time step $t \leftarrow 0$, parameter vector $\boldsymbol{\theta}_{t=0} \in \mathbb{R}^n$, first moment vector $\boldsymbol{m}_{t=0} \leftarrow \boldsymbol{0}$, second moment vector $\boldsymbol{v}_{t=0} \leftarrow \boldsymbol{0}$, schedule multiplier $\eta_{t=0} \in \mathbb{R}$
repeat
  $t \leftarrow t + 1$
  $\nabla f_t(\boldsymbol{\theta}_{t-1}) \leftarrow \text{SelectBatch}(\boldsymbol{\theta}_{t-1})$ ▷ select batch and return the corresponding gradient
  $\boldsymbol{g}_t \leftarrow \nabla f_t(\boldsymbol{\theta}_{t-1}) + \lambda \boldsymbol{\theta}_{t-1}$
  $\boldsymbol{m}_t \leftarrow \beta_1 \boldsymbol{m}_{t-1} + (1 - \beta_1) \boldsymbol{g}_t$ ▷ here and below all operations are element-wise
  $\boldsymbol{v}_t \leftarrow \beta_2 \boldsymbol{v}_{t-1} + (1 - \beta_2) \boldsymbol{g}_t^2$
  $\hat{\boldsymbol{m}}_t \leftarrow \boldsymbol{m}_t / (1 - \beta_1^t)$ ▷ β_1 is taken to the power of t
  $\hat{\boldsymbol{v}}_t \leftarrow \boldsymbol{v}_t / (1 - \beta_2^t)$ ▷ β_2 is taken to the power of t
  $\eta_t \leftarrow \text{SetScheduleMultiplier}(t)$ ▷ can be fixed, decay, or also be used for warm restarts
  $\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \eta_t \left(\alpha \hat{\boldsymbol{m}}_t / (\sqrt{\hat{\boldsymbol{v}}_t} + \epsilon) + \lambda \boldsymbol{\theta}_{t-1} \right)$
until *stopping criterion is met*
return optimized parameters $\boldsymbol{\theta}_t$

显式正则化

1. l_1-惩罚项

$$\mathcal{J}_2 = \frac{\lambda}{n}\sum_{l=1}^{L}\|\boldsymbol{W}^{[l]}\|_1$$

- $\|\boldsymbol{A}\|_1 = \sum_i \sum_j |a_{ij}|$
- 偏导数结果（用次导数定义不可导点的导数）

$$\frac{\partial \mathcal{J}_2}{\partial \boldsymbol{b}^{[l]}} = 0; \quad \frac{\partial \mathcal{J}_2}{\partial \boldsymbol{W}^{[l]}} = \frac{\lambda}{n}\text{sign}(\boldsymbol{W}^{[l]})$$

- 该惩罚常被用于
  - 控制模型复杂度
  - 类似于 LASSO，可实现模型参数的稀疏化

备注

1. 我们“错误地使用”$\|\boldsymbol{A}\|_1$，表达矩阵$\boldsymbol{A}$ 元素绝对值的加和
   - 一般来讲，$\|\boldsymbol{A}\|_1 = \max_j \sum_i |a_{ij}|$，它表示每列元素绝对值加和的最大值
2. 我们仅将l_2- 或者l_1-惩罚，施加于权重项$\{\boldsymbol{W}^{[l]} : l = 1, \ldots, L\}$
   - 我们不对偏置项 $\{\boldsymbol{b}^{[l]} : l = 1, \ldots, L\}$ 施加任何惩罚
3. 关于 l_2 惩罚项和 l_1 惩罚项的几何直观，请参见 Hastie 等（2009）的第 3.4 节

梯度下降法的隐正则化

1. 考虑下面所展示的梯度下降法

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \alpha \cdot \nabla \mathcal{J}(\boldsymbol{\theta}^{(t)})$$

   - $\boldsymbol{\theta}^{(t)}$：第 t 步更新后的模型参数
   - α：学习率
   - $\nabla \mathcal{J}(\boldsymbol{\theta}^{(t)}) = \partial \mathcal{J}(\boldsymbol{\theta}^{(t)}) / \partial \boldsymbol{\theta}$

2. 我们可以将 $\boldsymbol{\theta}^{(t)}$，看成是关于自变量 t 的分段线性的函数
   （在离散迭代点之间对 $\boldsymbol{\theta}^{(t)}$ 作分段线性插值的结果）
3. Barrett 和 Dherin（2021）利用常微分方程（ODE）的相关想法，讨论了梯度下降法所具有的隐正则化效果

梯度下降法的隐正则化

1. 梯度下降法，与求解下列初值问题时，所用的龙格-库塔法类似

$$\dot{\boldsymbol{\theta}} = -\nabla \mathcal{J}(\boldsymbol{\theta}), \quad \boldsymbol{\theta}(0) = \boldsymbol{\theta}^{(0)}$$

   - 我们假设 $\boldsymbol{\theta} = \boldsymbol{\theta}(t)$ 是关于自变量 t 的函数
   - $\dot{\boldsymbol{\theta}} = \mathrm{d}\boldsymbol{\theta}/\mathrm{d}t, \nabla \mathcal{J}(\boldsymbol{\theta}) = \partial \mathcal{J}(\boldsymbol{\theta})/\partial \boldsymbol{\theta}$
   - $\boldsymbol{\theta}(0)$：对应于 $t = 0$ 的初值
   - $\boldsymbol{\theta}^{(0)}$：对应于梯度下降法的初值

2. 当我们利用龙格-库塔法，求解以上微分方程时，我们往往将自变量 t 离散化
3. 这种离散化，将不可避免地造成误差
   - 微分方程的（连续）解与（离散化的、分段线性的）数值近似解之间的差异

梯度下降法的隐正则化

1. 例子：考虑下列代价函数

$$\mathcal{J}(\boldsymbol{\theta}) = (\theta_1^2 + 1)\theta_2^2$$

- $\boldsymbol{\theta} = (\theta_1, \theta_2)^{\mathrm{T}}$
- 最优解：$\theta_2 = 0$ 对应的水平线
- $\nabla \mathcal{J}(\boldsymbol{\theta}) = (2\theta_1\theta_2^2, 2\theta_2 + 2\theta_1^2\theta_2)^{\mathrm{T}}$

2. 考虑下列初值问题

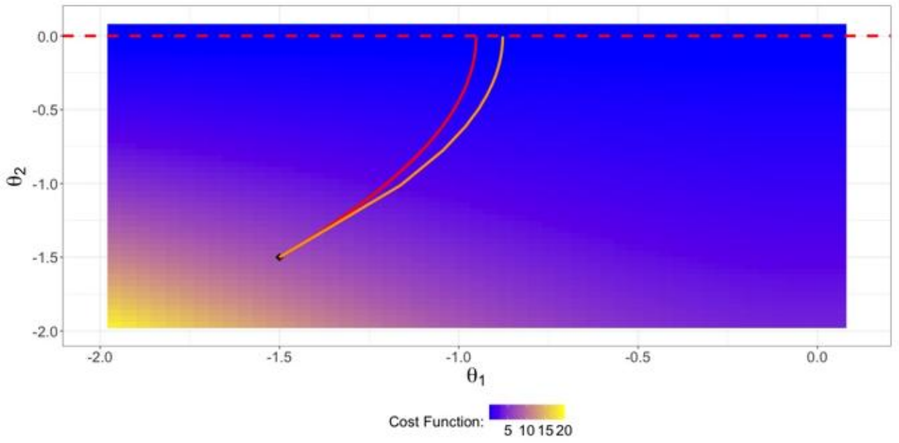
$$\dot{\boldsymbol{\theta}} = -\nabla \mathcal{J}(\boldsymbol{\theta}) = -\begin{pmatrix} 2\theta_1\theta_2^2 \\ 2\theta_2 + 2\theta_1^2\theta_2 \end{pmatrix}, \quad \boldsymbol{\theta}(0) = (-1.5, 1.5)$$

- θ_1 与 θ_2 均是自变量 t 的函数

梯度下降法的隐正则化

1．将梯度下降法中的学习率，设置为 $\alpha = 0.05$

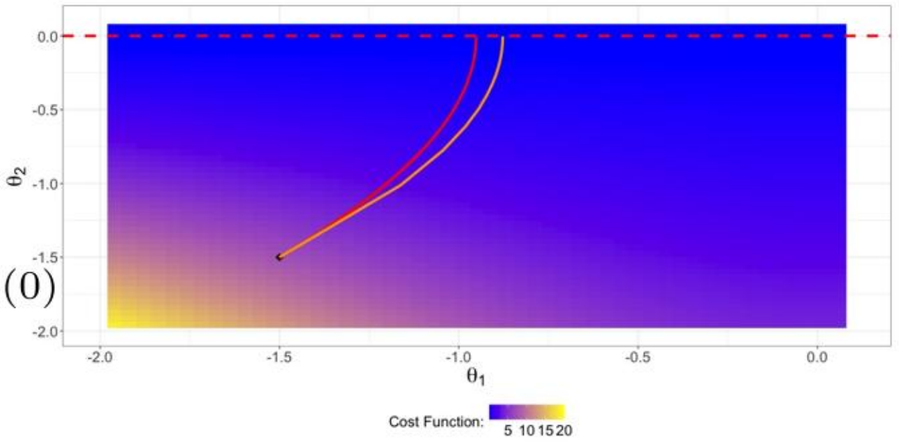
- 真实解
- 数值近似解（GD）



梯度下降法的隐正则化

1. 数值近似解距下列（自治）微分方程的真实解更近

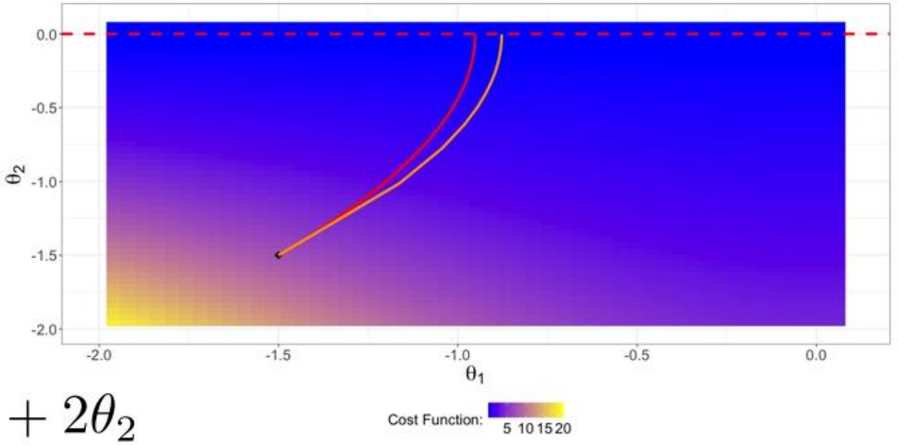
$$\dot{\boldsymbol{\theta}} = -\nabla \tilde{\mathcal{J}}(\boldsymbol{\theta}), \quad \boldsymbol{\theta}(0) = \boldsymbol{\theta}^{(0)}$$



- $\tilde{\mathcal{J}}(\boldsymbol{\theta}) = \mathcal{J}(\boldsymbol{\theta}) + \lambda \cdot R_{IG}(\boldsymbol{\theta})$
- $\lambda = \alpha \cdot p/4$
- α：学习率
- p：$\boldsymbol{\theta} = (\theta_1, \ldots, \theta_p)^{\mathrm{T}}$ 的维度
- $$R_{IG}(\boldsymbol{\theta}) = p^{-1} \sum_{i=1}^{p} (\partial \mathcal{J}(\boldsymbol{\theta}) / \partial \theta_i)^2$$

2. 对于由梯度下降法得到的数值解法而言，$R_{IG}(\boldsymbol{\theta})$ 是一个隐正则化（IGR）
3. 对于 IGR 项，Barrett 和 Dherin (2021) 列出了下面四个猜测
   - IGR 对应的解倾向于选择梯度范数较小的轨迹
   - IGR 能够使算法找到可能偏向较平坦的极小值
   - 在实验中，有时表现出更好的测试性能
   - IGR 可能使算法找到，对参数扰动更加稳健的极小值解

梯度下降法的隐正则化



1. 例子：考虑下列代价函数

$$\mathcal{J}(\boldsymbol{\theta}) = (\theta_1^2 + 1)\theta_2^2$$

- 我们已经有: $\partial\mathcal{J}(\boldsymbol{\theta})/\partial\theta_1 = 2\theta_1\theta_2^2, \quad \partial\mathcal{J}(\boldsymbol{\theta})/\partial\theta_2 = 2\theta_1^2\theta_2 + 2\theta_2$
- IGR 项为

$$R_{IG}(\boldsymbol{\theta}) = \frac{1}{2}\left\{4\theta_1^2\theta_2^4 + (2\theta_1^2\theta_2 + 2\theta_2)^2\right\}$$

- 对应于学习率 $\alpha = 0.05$，我们有 $\lambda = 0.05 \times 4/2 = 0.1$
- 令$\tilde{\mathcal{J}}(\boldsymbol{\theta}) = \mathcal{J}(\boldsymbol{\theta}) + \lambda \cdot R_{IG}(\boldsymbol{\theta})$

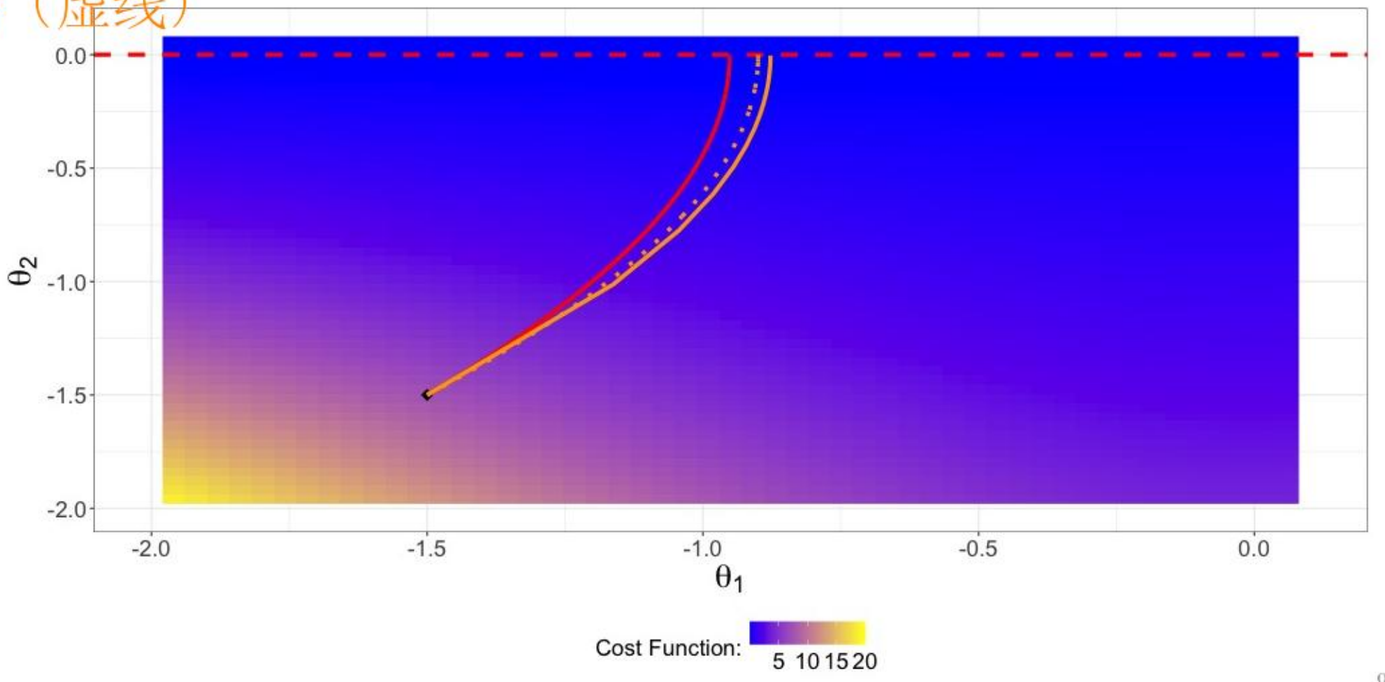
2. 考虑下列初值问题

$$\dot{\boldsymbol{\theta}} = -\nabla\tilde{\mathcal{J}}(\boldsymbol{\theta}), \quad \boldsymbol{\theta}(0) = (-1.5, 1.5)$$

梯度下降法的隐正则化

1. 将梯度下降法中的学习率，设置为 $\alpha = 0.05$

- 真实解
- 数值近似解（GD）
- 对于带有 IGR 项微分方程的真实解（虚线）



早停

1. 模型在训练集上的表现会逐渐变好
2. 然而，模型可能对训练集进行了过拟合
   - 模型在验证集验证集或者测试集上的表现，并不会随着训练次数的增加而一直变好
3. 早停技术监控模型在验证集上的表现
   - 当模型的表现在验证集上不再提高时，模型训练停止

早停

1. 早停法与双下降现象的权衡，本质上是模型训练中，稳健性与潜能的博弈
   - 当模型规模或训练时间跨越某个阈值后，模型性能可能出现双下降现象
   - 早停是一个在第一次风险谷底就及时止盈，得到一个稳健模型
2. 用早停得到一个稳健的基线模型
3. 如果资源允许，尝试增加训练轮次，探索是否存在双下降现象

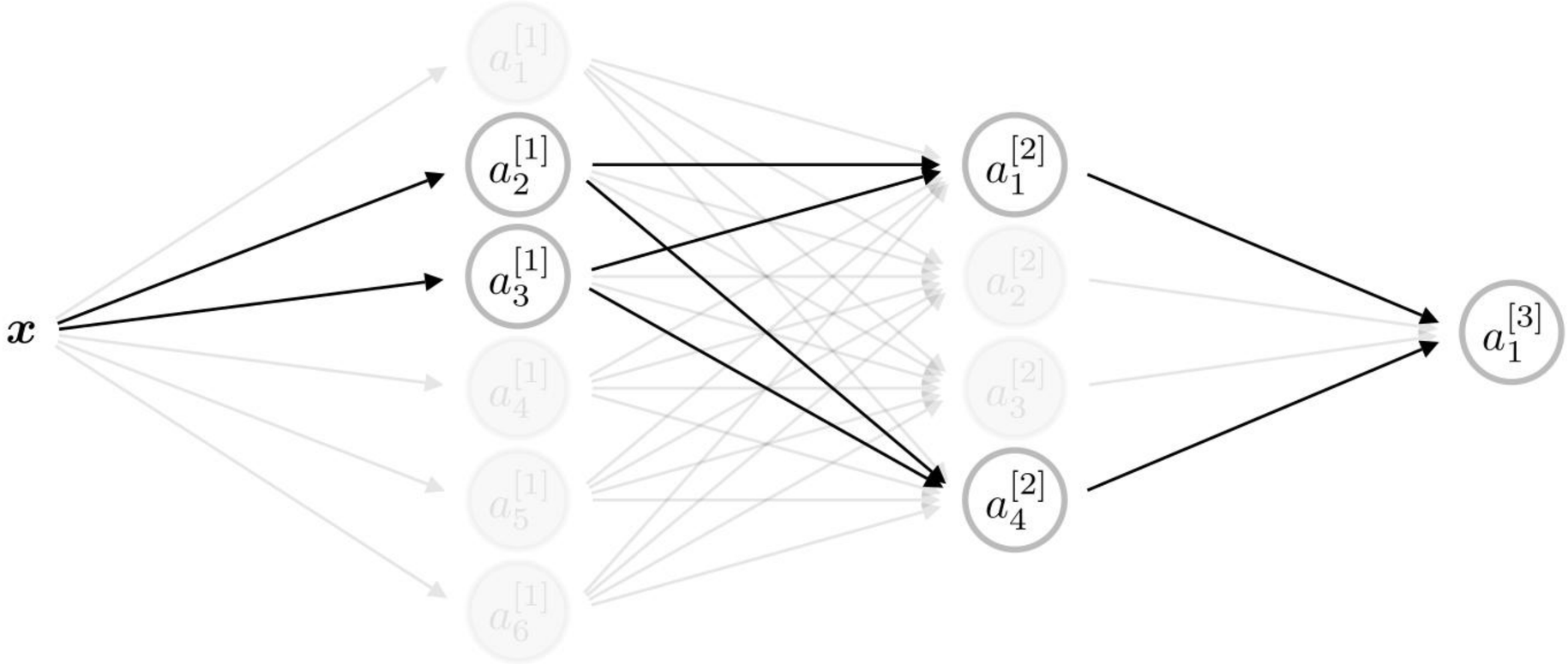
Dropout

1. 模型训练阶段常用的隐正则化方法（Srivastava 等，2014）
   - 随机在隐藏层中，“删除”神经元
   - 如果某神经元被“删除”，那么它对应的激活值被设置为 0
   - “删除”对每个训练样本而言，是独立随机进行的
2. Dropout 也是一种注入随机噪声的正则化方法
3. 提高模型的稳健性和泛化能力

Dropout

1. 对于第 l 层中的每个神经元$(l = 1, \ldots, L-1)$
   - 令 dropout 概率为$p^{[l]}$
   - 以$p^{[l]}$ 为概率，将对应神经元的激活值设置为 0
   - 基于因子$(1 - p^{[l]})^{-1}$ 对激活值进行放缩，弥补 dropout 带来的信息损失
2. 我们通常不对输出层进行 dropout 操作
3. 是否对输入层实施 dropout，取决于具体任务
4. 在模型训练完毕后，我们也不再进行 dropout 操作

Dropout


$\boldsymbol{x}$
$a_1^{[1]}$
$a_2^{[1]}$
$a_3^{[1]}$
$a_4^{[1]}$
$a_5^{[1]}$
$a_6^{[1]}$
$a_1^{[2]}$
$a_2^{[2]}$
$a_3^{[2]}$
$a_4^{[2]}$
$a_1^{[3]}$
输入层
第一隐藏层
第二隐藏层
输出层

Dropout

1. 我们通过掩码矩阵，实现 dropout 正则化

2. 令$\boldsymbol{M}^{[l]} \in \{0,1\}^{n \times d^{[l]}}$ 表示一个掩码矩阵

   - $\boldsymbol{M}^{[l]}$中的每个元素，来自于成功概率为$(1-p^{[l]})$ 的伯努利分布

3. 前向传播

$$\boldsymbol{A}_d^{[l]} = \frac{\boldsymbol{A}^{[l]} \circ \boldsymbol{M}^{[l]}}{1-p^{[l]}}.$$

4. 后向传播（容易得到）

   - Dropout 不涉及额外模型参数

Dropout

1. 讨论

- 在操作上，dropout 表现为随机删除部分隐藏单元
- 在实现上，dropout 依赖掩码矩阵和缩放因子
- 在本质上，dropout 对网络隐藏层施加乘法型噪声
- Dropout 的效果是：抑制共适应、提升鲁棒性、减少过拟合

Dropout

1．Dropout 是将一个伯努利随机变量，乘在激活值上

2．更一般地，我们可将其他随机噪声，加在或者乘在激活值上

- 在训练集上加随机噪声
- 在模型权重项上加噪声
- 标签平滑，或对标签进行小幅扰动

噪声注入

1. 噪声注入也是模型训练中，常用的正则化方法之一

- 输入数据注入噪声（Matsuoka，1992；Grandvalet 等, 1997）
- 标签平滑（Müller 等，2019；Lukasik 等, 2020）
- 数据增强（Shorten 和 Khoshgoftaar, 2019）
- ……

集成学习

1．为了提高模型的泛化能力，我们可以考虑多个模型

- 降低可能来自于一个过拟合模型的方差
- 降低可能来自于一个欠拟合模型的偏差

集成学习

1．主流的集成学习，大致可以分为以下三类

- Bagging 旨在利用 bootstrap 方法产生多个训练集，并在每个训练集上，训练相同的模型，通过模型平均降低方差
  - 随机森林
- Boosting 旨在串行不同的模型，以更正之前模型可能的错误
  - AdaBoost, Gradient Boosted Decision Trees, XGBoost
- Stacking 旨在学习如何结合不同的模型，来提高模型的预测精度（Wolpert, 1992）
  - 元模型

课程总结

1．正则化有助于提升模型的泛化能力，降低对训练样本的偶然性依赖

- 显式正则化：在代价函数上，施加惩罚项
- 隐式正则化：由训练算法或训练过程隐式产生参数或函数偏好
- 噪声注入：在训练过程中故意引入随机扰动
- 集成学习：通过组合多个模型获得更稳定的预测