# 机器学习总结

FU Hanlin

April 16, 2019

# Contents

# Chapter 1

# 特征工程

1. 特征缩放

   - 线性函数归一化。它对原始数据进行线性变换，使结果映射到 $[0, 1]$ 的范围。

     $$X_{norm} = \frac{X - X_{min}}{X_{max} - X_{min}} \tag{1.1}$$

     其中 $X$ 为原始数据，$X_{max}, X_{min}$ 分别为数据最大值和最小值。
   - 零均值归一化. 它会将原始数据映射到均值为 0，标准差为 1 的分布上。

   - 实际应用中，通过梯度下降法求解的模型通常是需要归一化的，更容易通过梯度下降找到最优解。但对于决策树模型并不适用。

2. Word2Vec

   - CBOW 的目标是根据上下文出现的词语来预测当前词的生成概率，Skip-gram 是根据当前词来预测上下文中各词的生成概率。
   - 神经网络部分：训练权重，使得语料库中所有单词的整体生成概率最大化。
   - 上下文-单词矩阵

3. 避免过拟合的方法

   - 基于模型的方法：
     简化模型（将非线性转化为线性），添加约束项以缩小假设空间（L1/L2 正则），集成学习，Dropout 超参数

- 基于数据的方法：
  1. 一定程度内的随机旋转，平移，缩放，裁剪，填充，左右翻转等。
  2. 对图像中的像素添加噪声扰动。
  3. 颜色变换。
  4. 改变图像的亮度，清晰度，对比度，锐度等。

# Chapter 2

# SVM

1. 原理
   给定训练样本集，分类学习最基本的想法就是基于训练集 D 再样本空间中找打一个划分超平面，将不同类别的样本分开。
   在样本空间中，划分超平面可通过如下线性方程来描述：

$$\omega^{\mathrm{T}}x + b = 0 \tag{2.1}$$

   其中 $\omega$ 为法向量，决定了超平面的方向；$b$ 为位移项，决定了超平面与原点之间的距离。
   样本中任意点 x 到超平面 $(w, b)$ 的距离可写为

$$r = \frac{\omega^{\mathrm{T}}x + b}{||\omega||} \tag{2.2}$$

   假设超平面能将训练样本正确分类，

$$\omega^{\mathrm{T}}x + b \geq +1, y_i = +1 \tag{2.3}$$
$$\omega^{\mathrm{T}}x + b \leq -1, y_i = -1 \tag{2.4}$$

   距离超平面最近的这几个训练样本点使等号成立，它们被称为支持向量，两个支持向量到超平面的距离为

$$\gamma = \frac{2}{||\omega||} \tag{2.5}$$

它被称为间隔。欲找到具有最大间隔的划分平面也就是

$$max_{\omega,b}\frac{2}{||\omega||} \tag{2.6}$$

$$s.t.y_i(\omega^{\mathrm{T}}x+b)\geq +1 \tag{2.7}$$

最大化 $||\omega||^{-1}$，这等价于最小化 $||\omega||^2$，于是可重写为 $min_{\omega,b}\frac{1}{2}||w||^2$

2. 对偶问题
原问题本身是一个凸二次规划问题（目标函数是二次的，约束条件是线性的），能直接用现成的优化计算报求解，但我们有更高效的办法。对式使用拉格朗日乘子法可得到其对偶问题。

$$L(\omega,b,\alpha)=\frac{1}{2}||w||^2+\sum_{m}^{i=1}\alpha_i(1-y_i(\omega^{\mathrm{T}}x+b)) \tag{2.8}$$

令 $L(\omega,b,\alpha)$ 对 $\omega$ 和 b 的偏导为零可得

$$\omega=\sum_{i=1}^{m}\alpha_i y_i x_i, \tag{2.9}$$

$$0=\sum_{i=1}^{m}\alpha_i y_i \tag{2.10}$$

代入 $L(\omega,b,\alpha)$ 中，

$$L(\omega,b,\alpha)=\frac{1}{2}\omega\omega^{\mathrm{T}}+\sum_{m}^{i=1}\alpha_i-b\sum_{i=1}^{m}\alpha_i y_i-\omega^{\mathrm{T}}\sum_{i=1}^{m}\alpha_i x_i y_i \tag{2.11}$$

$= \frac{1}{2}\omega\omega^{\mathrm{T}}+\sum_{i=1}^{m}\alpha_i-b\cdot 0-\omega^{\mathrm{T}}\sum_{i=1}^{m}\alpha_i x_i y_i$
$= \frac{1}{2}\omega^{\mathrm{T}}\sum_{i=1}^{m}\alpha_i x_i y_i+\sum_{m}^{i=1}\alpha_i-\omega^{\mathrm{T}}\sum_{i=1}^{m}\alpha_i x_i y_i$
$= \sum_{m}^{i=1}\alpha_i-\frac{1}{2}(\sum_{i=1}^{m}\alpha_i x_i y_i)^{\mathrm{T}}(\sum_{i=1}^{m}\alpha_i x_i y_i)$
$= \sum_{m}^{i=1}\alpha_i-\frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m}\alpha_i\alpha_j x_i^{\mathrm{T}}x_j y_i y_j$
$\alpha_i\geq 0, \sum_{i=1}^{m}\alpha_i y_i=0$

3. KKT 条件

- 为什么转换成对偶问题：
  1. 对偶问题将原始问题中的约束转为了对偶问题中的等式约束
  2. 方便核函数的引入
  3. 改变了问题的复杂度。由求特征向量 $\omega$ 转化为求比例系数 $\alpha$，在原始问题下，求解的复杂度与样本的维度有关，即 $\omega$ 的维度。在对偶问题下，只与样本数量有关。

- KKT 条件:
  1. $\alpha_i \geq 0$
  2. $y_i(\omega^{\mathrm{T}}x + b - 1) \geq 0$
  3. $\sum \alpha_i(y_i(\omega^{\mathrm{T}}x + b - 1)) = 0$

4. 核函数

- 高斯核为什么有效?
  在现实任务中，原始样本空间内也许并不存在一个能正确划分两类样本的超平面。对这样的问题，可将样本从原始空间映射到一个更高维的特征空间，使得样本在这个特征空间内线性可分。令 $\phi(x)$ 表示将 $x$ 映射后的特征向量，于是，在特征空间中划分超平面所对应的模型可表示为 $f(x) = \omega^{\mathrm{T}}\phi(x) + b$
- 常用的核函数
  1. 线性核
  2. 多项式核
  3. 高斯核
  4. 拉普拉斯核
  5.Sigmoid 核

5. 过拟合

- 松弛变量 $\xi$
  约束条件变为:$s.t.y_i(\omega^{\mathrm{T}}x + b) \geq 1 - \xi_i$
  引入松弛变量使 SVM 能够容忍异常点的存在。为什么? 因为引入松弛变量后，所有点到超平面的距离约束不需要大于等于 1 了，而是大于 0.8 就行了（如果 =0.2 的话），那么异常点就可以不是支持向量了，它就作为一个普通的点存在，我们的支持向量和超平面都不会受到它的影响。
- 软间隔支持向量机、、在最大化间隔的同时，不满足约束的样本应尽可能少。

$$min_{\omega,b,\xi_i}\frac{1}{2}||w||^2 + C\sum_{i=1}^{m}\xi_i \tag{2.12}$$

# Chapter 3

# 线性模型

1. 线性回归原理
   给定数据集 D，线性回归试图学得一个线性模型以尽可能准确地预测实值输出标记：

$$f(x_i) = \omega x_i + b, \quad f(x_i) \simeq y_i \tag{3.1}$$
$$\tag{3.2}$$

   均方误差是回归任务中最常用的性能度量，因此我们可以试图让均方误差最小化。均方误差有非常好的几何意义，它对应了常用的欧式距离。在线性回归中，最小二乘法就是试图找到一条直线，使所有样本到直线上的欧式距离之和最小。

$$(w_*, b_*) = \arg\min_{(w,b)} \sum_{i=1}^{m} (f(x_i) - y_i)^2 \tag{3.3}$$
$$= \arg\min_{(w,b)} \sum_{i=1}^{m} (y_i - \omega x_i - b)^2 \tag{3.4}$$

   求解 $\omega$ 和 b 使得方程最小化的过程，称为线性回归模型的最小二乘参数估计。

2. 多元线性回归原理
   为了便于讨论，我们把 $\omega$ 和 b 吸收入向量形式 $\hat{\omega} = (\omega; b)$, 相应的把数据集 D 表示为一个 $m \times (d+1)$ 大小的矩阵 X，其中每行对应一个示例，该行前 d 个元素对应于示例的 d 个属性值。有，

$$\hat{\omega}^* = \arg\min_{\hat{\omega}}(y - X\hat{\omega})^T(y - X\hat{\omega}) \tag{3.5}$$

$$\tag{3.6}$$

对 $\hat{\omega}$ 求导得到

$$\frac{\partial E_{\hat{\omega}}}{\partial \hat{\omega}} = 2X^T(X\hat{\omega} - y) \tag{3.7}$$

3. 对数几率回归

- Sigmoid 函数
  它将 z 值转化为一个接近 0 或 1 的 y 值，并且其输出值在 $z = 0$ 附近变化很陡。

  $$y = \frac{1}{1 + e^{-z}} \tag{3.8}$$

  $$\tag{3.9}$$

  将对数几率函数代入得到

  $$y = \frac{1}{1 + e^{-(\omega^T x + b)}} \tag{3.10}$$

  $$\tag{3.11}$$

  可变化为

  $$\ln \frac{y}{1 - y} = \omega^T x + b \tag{3.12}$$

  $$\tag{3.13}$$

  由此可看出，实际上是在用线性回归模型的预测结果去逼近真实标记的对数几率，这种方法有很多优点，例如它是直接对分类可能性进行建模，无需事先假设分布；它不是仅预测出类别，而是可得到近似概率预测。
- 极大似然法将式重写为

  $$\ln \frac{p(y = 1|x)}{p(y = 0|x)} = \omega^T x + b \tag{3.14}$$

  $$p(y = 1|x) = \frac{e^{\omega^T x + b}}{1 + \omega^T x + b} \tag{3.15}$$

  $$p(y = 0|x) = \frac{1}{1 + \omega^T x + b} \tag{3.16}$$

  $$\tag{3.17}$$

于是我们可通过极大似然法来估计 $\omega$ 和 b，对数几率回归模型最大化对数似然，即令每个样本属于其真实标记的概率越大越好，连乘操作易造成下溢，通常使用对数似然。

$$l(\omega, b) = \sum_{i=1}^{m} \ln p(y_i|x_i; \omega, b) \tag{3.18}$$

$$l(\beta, b) = \sum_{i=1}^{m} \ln(y_i p_1(\hat{x_i}; \beta) + (1 - y_i) p_0(\hat{x_i}; \beta) \tag{3.19}$$

$$p1(\hat{x_i}; \beta) = \frac{e^{\beta^T \hat{x_i}}}{1 + e^{\beta^T \hat{x_i}}}, p0(\hat{x_i}; \beta) = \frac{1}{1 + e^{\beta^T \hat{x_i}}} \tag{3.20}$$

$$, \tag{3.21}$$

$$l(\omega, b) = \sum_{i=1}^{m} \ln \frac{y_i e^{\beta^T \hat{x_i}} + 1 - y_i}{1 + e^{\beta^T \hat{x_i}}} \tag{3.22}$$

$$l(\omega, b) = \sum_{i=1}^{m} (ln(y_i e^{\beta^T \hat{x_i}} + 1 - y_i) - \ln(1 + e^{\beta^T \hat{x_i}}) \tag{3.23}$$

$$y_i = 0 or 1 l(\omega, b) = \sum_{i=1}^{m} (-\ln(1 + e^{\beta^T \hat{x_i}})), \quad y_i = 0 \tag{3.24}$$

$$\sum_{i=1}^{m} (\beta^T \hat{x_i} - \ln(1 + e^{\beta^T \hat{x_i}})), \quad y_i = 1 \tag{3.25}$$

$$l(\beta) = \sum_{i=1}^{m} (y_i \beta^T \hat{x_i} - \ln(1 + e^{\beta^T \hat{x_i}})) \tag{3.26}$$

由于此式仍为极大似然估计的似然函数，所以最大化似然函数等价于最小化似然函数的相反数，也即在似然函数前添加负号即可得

4. L0,L1,L2 正则

- L0 正则
  L0 范数是指向量中非 0 的元素的个数。如果我们用 L0 范数来规则化一个参数矩阵 W 的话，就是希望 W 的大部分元素都是 0。
- L1 正则 (Lasso)
  L1 正则化是权值的绝对值之和，是带有绝对值符号的函数，因此是不完全可微的。机器学习的任务就是要通过一些方法（比如梯度下降）求出损失函数的最小值。当我们在原始损失函数后添

加 L1 正则化项时，相当于对做了一个约束。令，则，此时我们的任务变成在约束下求出取最小值的解。优点：

- 特征选择，稀疏的矩阵可以起到特征选择的作用。我们知道，稀疏矩阵是只有少数值为非零，大多数值为 0 的矩阵。有些场景中特征数量可能会非常多，不利于直接使用机器学习算法。如果这是可以将参数矩阵转换稀疏的，那么就可以只保留一小部分特征（只有系数为非零的那一部分特征被保留）。这样转换成稀疏矩阵就可以起到特征选择的作用。
- 可解释性。另一个青睐于稀疏的理由是，模型更容易解释。例如患某种病的概率是 y，然后我们收集到的数据 x 是 1000 维的，也就是我们需要寻找这 1000 种因素到底是怎么影响患上这种病的概率的。假设我们这个是个回归模型：$y = \omega_1 * x_1 + \omega_2 * x_2 + ... + \omega_1 000 * x_1 000 + b$（当然了，为了让 y 限定在 [0,1] 的范围，一般还得加个 Logistic 函数）。通过学习，如果最后学习到的 w* 就只有很少的非零元素，例如只有 5 个非零的 wi，那么我们就有理由相信，这些对应的特征在患病分析上面提供的信息是巨大的，决策性的。也就是说，患不患这种病只和这 5 个因素有关，那医生就好分析多了。

- L2 正则 (Ridge) L2 范数是指向量各元素的平方和然后求平方根。我们让 L2 范数的规则项 $||\omega||^2$ 最小，可以使得 $\omega$ 的每个元素都很小，都接近于 0，但与 L1 范数不同，它不会让它等于 0，而是接近于 0。而越小的参数说明模型越简单，越简单的模型则越不容易产生过拟合现象。通过 L2 范数，我们可以实现了对模型空间的限制，从而在一定程度上避免了过拟合。
- L1 和 L2 的区别
  - L1 的下降速度比 L2 的下降速度要快

  - L1 范数：L1 范数在正则化的过程中会趋向于产生少量的特征，而其他的特征都是 0（L1 会使得参数矩阵变得稀疏）。因此 L1 不仅可以起到正则化的作用，还可以起到特征选择的作用。L2 范数：L2 范数是通过使权重衰减，进而使得特征对于总体的影响减小而起到防止过拟合的作用的。L2 的优点在于求解稳定、快速。

# Chapter 4

# 决策树

1. 分类与回归的区别

   - 回归与分类的根本区别在于输出空间是否为一个度量空间。
   - 对于回归问题，其输出空间 B 是一个度量空间，即所谓“定量”。对于分类问题，其输出空间 B 不是度量空间，即所谓“定性”。

2. 决策树的特征选择我们希望决策树的分支结点所包含的样本尽可能属于同一类别，即结点的纯度越来越高。

   - ID3——最大信息增益
     信息熵：
     对于样本集合 D，类别数为 K，数据集 D 的经验熵表示为 H(D)，然后计算某个特征 A 对于数据集 D 的经验条件熵 H(D|A)，于是信息增益 g(D,A) 可以表示为二者之差，可得 g(D,A)=H(D)-H(D|A)

$$H(D) = -\sum_{k=1}^{K} \frac{|C_k|}{|D|} log_2 \frac{|C_k|}{|D|} \tag{4.1}$$

$$H(D|A) = \sum_{i=1}^{n} \frac{|D_i|}{|D|} H(D_i) \tag{4.2}$$

$$= \sum_{i=1}^{n} \frac{|D_i|}{|D|} (-\sum_{i=1}^{k} \frac{|D_{ik}|}{|D_i|} \log_2 \frac{|D_{ik}|}{|D_i|}) \tag{4.3}$$

$$gain = H(D) - H(D|A) \tag{4.4}$$

- C4.5——增益率特征 A 对数据集 D 的信息增益比定义为:

$$g_R = (D, A) = \frac{g(D, A)}{H_A(D)} \tag{4.5}$$

其中,

$$H_A(D) = -\sum_{i=1}^{n} \frac{|D_i|}{|D|} \log_2 \frac{|D_i|}{|D|} \tag{4.6}$$

- CART——基尼系数 Gini 描述的是数据的纯度，与信息熵含义类似。

$$Gini(D) = 1 - \sum_{k=1}^{n} (\frac{|C_k|}{|D|})^2 \tag{4.7}$$

与 ID3,C4.5 不同的是，CART 是一颗二叉树，采用二元分割法，每一步将数据按特征 A 的取值切分成两份，分别进入左右子树。对于样本 D，如根据特征 A 的某个值 A，把 D 分为 D1 和 D2 两部分，则在特征 A 的条件下，特征 A 的 Gini 指数定义为:

$$Gini(D, A) = \frac{|D_1|}{|D|} Gini(D_1) + \frac{|D_2|}{|D|} Gini(D_2) \tag{4.8}$$

基尼系数与熵之半的曲线非常接近，因此基尼系数可以做为熵模型的一个近似替代。

3. 随机森林
随机森林是 Bagging 的一个扩展变体，RF 在以决策树为基学习器构建 Bagging 集成的基础上，进一步在决策树的训练过程中引入了随机属性选择。

- 随机性 1
在训练时，随机森林中的每棵树都会从数据点的随机样本中学习。样本被有放回的抽样，称为自助抽样法 (bootstrapping)，这意味着一些样本将在一棵树中被多次使用。背后的想法是在不同样本上训练每棵树，尽管每棵树相对于特定训练数据集可能具有高方差，但总体而言，整个森林将具有较低的方差，同时不以增加偏差为代价。

- 随机性 2
  随机森林中的另一个主要概念是，只考虑所有特征的一个子集来拆分每个决策树中的每个节点。通常将其设置为 sqrt 以进行分类，这意味着如果有 16 个特征，则在每个树中的每个节点处，只考虑 4 个随机特征来拆分节点。

4. 随机森林优缺点优点：

   - 随机森林抗过拟合能力比较强
   - 随机森林算法有很强的抗干扰能力（数据存在大量缺失）
   - 训练速度快，
   - 由于采用了随机采样，训练出的模型的方差小，泛化能力强。

   缺点：

   - 在某些噪音比较大的样本集上，RF 模型容易陷入过拟合。
   - 取值划分比较多的特征容易对 RF 的决策产生更大的影响，从而影响拟合的模型的效果。

5. 剪枝

   (a) 决策树的损失函数
   根据叶结点中的预测误差来衡量，即训练数据的拟合程度。考虑到所有的叶结点中每个叶节点中的样例个数不同，我们采用

$$C(T) = -\sum_{t=1}^{|T|}\sum_{k=1}^{K} N_{tk} \log \frac{N_{tk}}{N_t} \tag{4.9}$$

   来衡量模型对训练数据的整体误差。但是如果仅仅使用 $C(T)$ 来作为优化目标函数，就会导致模型走向过拟合的结果。
   为了避免过拟合，我们需要给优化目标函数增加一个正则项，正则项应该包含模型的复杂度信息。对于决策树来说，其叶节点的数量 $|T|$ 越多就越复杂，我们用添加正则项的

$$C_\alpha(T) = C(T) + \alpha|T| \tag{4.10}$$

   来作为优化的目标函数，也就是树的损失函数。参数 $\alpha$ 控制了两者之间的影响程度，较大的 $\alpha$ 促使选择较简单的模型，较小的 $\alpha$ 促使选择复杂的模型。为了提高决策树的泛化能力，需要对树进行剪枝 (Pruning)，把过于细分的叶结点（通常是数据量过少导致噪声数据的影响增加）去掉而回退到其父结点或更高

的结点，使其父结点或更高的结点变为叶结点。
剪枝的目的不是为了最小化损失函数，剪枝的目的是为了达到一个更好的泛化能力。而对于决策树来说，叶结点的数量越多，反应了决策树对训练数据的细节反应的越多，继而弱化了泛化能力。

(b) 预剪枝通过提前停止树的构建而对树剪枝，一旦停止，节点就是树叶，该树叶持有子集元祖最频繁的类。停止决策树生长最简单的方法有：1. 定义一个高度，当决策树达到该高度时就停止决策树的生长 2. 达到某个节点的实例具有相同的特征向量，及时这些实例不属于同一类，也可以停止决策树的生长。这个方法对于处理数据的数据冲突问题比较有效。3. 定义一个阈值，当达到某个节点的实例个数小于阈值时就可以停止决策树的生长 4. 定义一个阈值，通过计算每次扩张对系统性能的增益，并比较增益值与该阈值大小来决定是否停止决策树的生长。

(c) 后剪枝它首先构造完整的决策树，允许树过度拟合训练数据，然后对那些置信度不够的结点子树用叶子结点来代替，该叶子的类标号用该结点子树中最频繁的类标记。相比于先剪枝，这种方法更常用，正是因为在先剪枝方法中精确地估计何时停止树增长很困难。

i. REP 错误率降低剪枝对于完全决策树中的每一个非叶子节点的子树，我们尝试着把它替换成一个叶子节点，该叶子节点的类别我们用子树所覆盖训练样本中存在最多的那个类来代替，这样就产生了一个简化决策树，然后比较这两个决策树在测试数据集中的表现，如果简化决策树在测试数据集中的错误比较少，那么该子树就可以替换成叶子节点。该算法以 bottom-up 的方式遍历所有的子树，直至没有任何子树可以替换使得测试数据集的表现得以改进时，算法就可以终止。

ii. PEP 悲观剪枝

6. GBDT

(a) GDBT 概述 GBDT 也是集成学习 Boosting 家族的成员，但是却和传统的 Adaboost 有很大的不同。回顾下 Adaboost，我们是利用前一轮迭代弱学习器的误差率来更新训练集的权重，这样一轮轮的迭代下去。GBDT 也是迭代，使用了前向分布算法，但是弱学习器限定了只能使用 CART 回归树模型，同时迭代思路和 Adaboost 也有所不同。
在 GBDT 的迭代中，假设我们前一轮迭代得到的强学习器是 $f_{t-1}(x)$，损失函数是 $L(y, f_{t-1}(x))$，我们本轮迭代的目标是找

到一个 CART 回归树模型的弱学习器 $h_t(x)$, 让本轮损失函数 $L(y, f_t(x)) = L(y, f_{t-1}(x) + h_t(x)$ 最小。

(b) GDBT 原理

(c) GBDT 的优缺点优点:

- 预测阶段计算速度快，树与树之间可并行化计算
- 在分布稠密的数据集上，泛化能力和表达能力都很好
- 具有较好的解释性和鲁棒性

缺点:

- 在高维稀疏的数据集上，表现不如 SVM 或 NN
- 在处理文本分类特征问题上优势不大
- 训练过程需要串行训练

# Chapter 5

# RNN

1. RNN 原理 基础的神经网络只在层与层之间建立了权连接，RNN 最

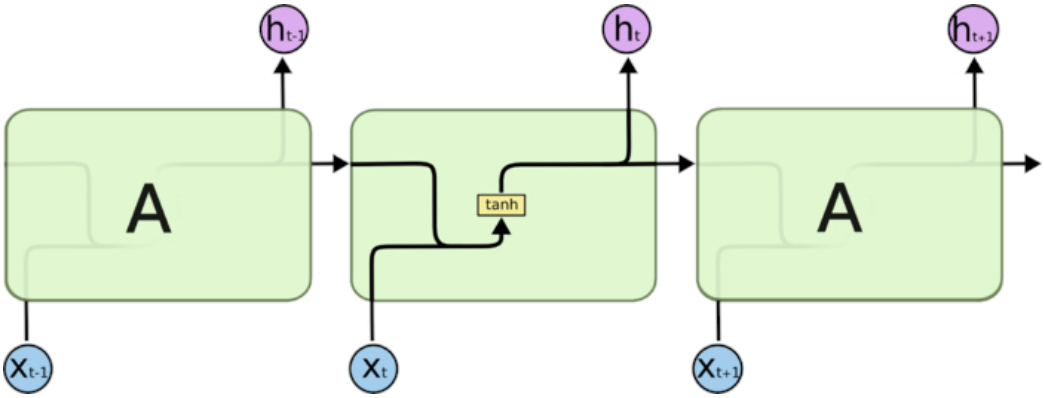



Figure 5.1: RNN 原理图

大的不同之处就是在层之间的神经元间也建立了权的连接。其中，$x$ 是输入，$h$ 是隐含单元，$o$ 是输出，$L$ 是损失函数，$y$ 是训练集的标签。$V, W, U$ 是权值。
对于 $t$ 时刻的前向传播算法：

$$h^{(t)} = \phi(Ux^{(t)} + Wh^{(t-1)} + b) \tag{5.1}$$

其中 $\phi()$ 为激活函数，$b$ 为偏置。
t 时刻的输出：

$$o^{(t)} = Vh^{(t)} + c \tag{5.2}$$

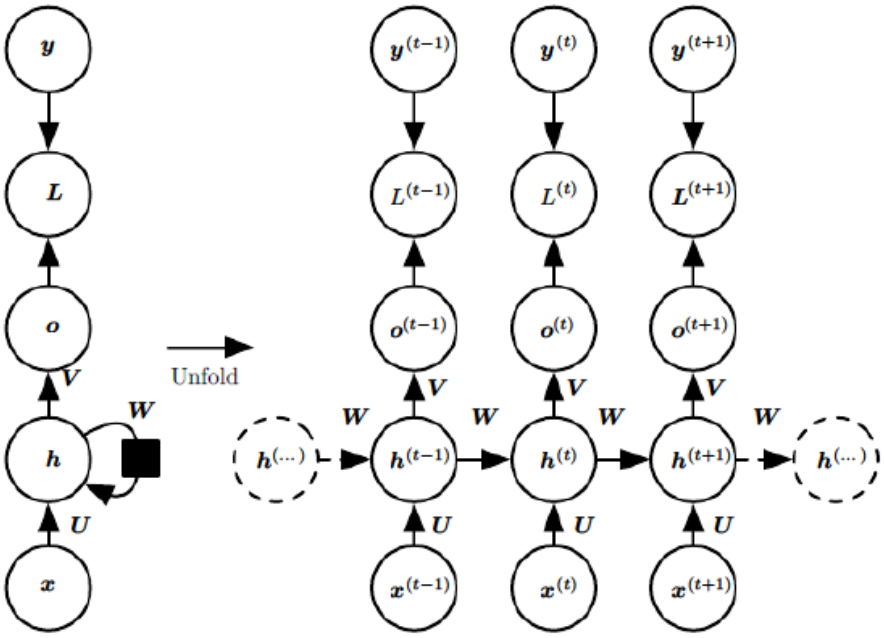



Figure 5.2: RNN 展开

最终模型的预测输出为:

$$\hat{y}^{(t)} = \theta(o^{(t)}) \tag{5.3}$$

其中 $\theta$ 为激活函数，通常 RNN 用于分类，故这里一般用 softmax 函数。
第 $t$ 层的隐含状态 $h_t$ 编码了序列中前 $t$ 个输入的信息，可以通过当前的输入 $x_t$ 和上一层神经网络的状态 $h_{t-1}$ 计算得到；最后一层的状态编码 $h_T$ 编码了整个序列的信息。
通过最小化损失误差（即输出的 y 与真实类别之间的距离），我们可以不断训练网络。

2. RNN 的训练方法————BPTT BPTT（back-propagation through time）算法是常用的训练 RNN 的方法，其实本质还是 BP 算法，只不过 RNN 处理时间序列数据，所以要基于时间反向传播，故叫随时间反向传播。BPTT 的中心思想和 BP 算法相同，沿着需要优化的参数的负梯度方向不断寻找更优的点直至收敛。综上所述，BPTT 算法本质还是 BP 算法，BP 算法本质还是梯度下降法，那么求各个参数的梯度便成了此算法的核心。在某个时刻对 $W$ 或是 $U$ 的偏导数，需

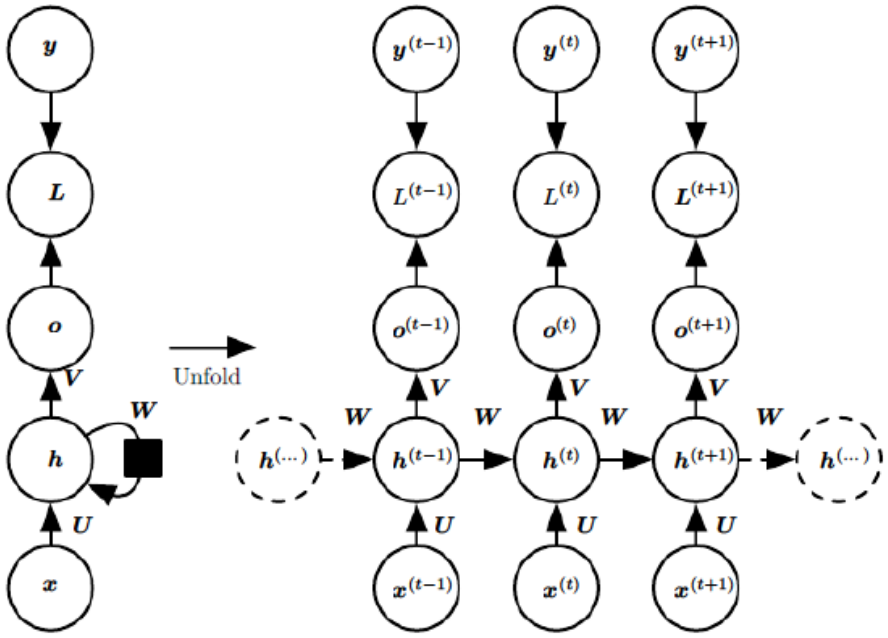



Figure 5.3: RNN 展开

要追溯这个时刻之前所有时刻的信息。

$$\frac{\partial L^{(t)}}{\partial W} = \sum_{k=0}^{t} \frac{\partial L^{(t)}}{\partial o^{(t)}} \frac{\partial o^{(t)}}{\partial h^{(t)}} \left( \prod_{j=k+1}^{t} \frac{\partial h^{(j)}}{\partial h^{(j-1)}} \right) \frac{\partial h^{(k)}}{\partial W} \tag{5.4}$$

$$\frac{\partial L^{(t)}}{\partial U} = \sum_{k=0}^{t} \frac{\partial L^{(t)}}{\partial o^{(t)}} \frac{\partial o^{(t)}}{\partial h^{(t)}} \left( \prod_{j=k+1}^{t} \frac{\partial h^{(j)}}{\partial h^{(j-1)}} \right) \frac{\partial h^{(k)}}{\partial U} \tag{5.5}$$

我们会发现累乘会导致激活函数导数的累乘，进而会导致“梯度消失“和“梯度爆炸“现象的发生。如果取 sigmoid 函数作为激活函数的话，那么必然是一堆小数在做乘法，结果就是越乘越小。随着时间序列的不断深入，小数的累乘就会导致梯度越来越小直到接近于 0，这就是“梯度消失“现象。
解决“梯度消失“的方法主要有：

(a) 选取更好的激活函数 (relu)

(b) 改变传播结构 (LSTM)

3. LSTM
LSTM 有通过精心设计的称作为“门”的结构来去除或者增加信息到

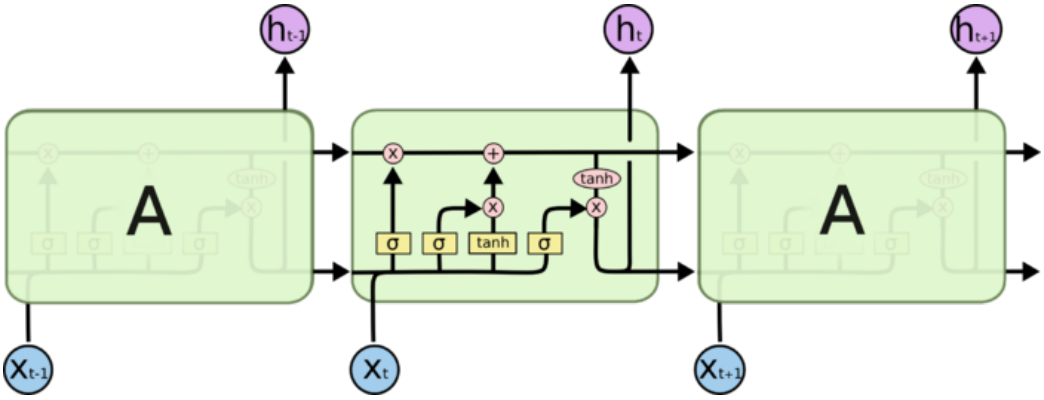
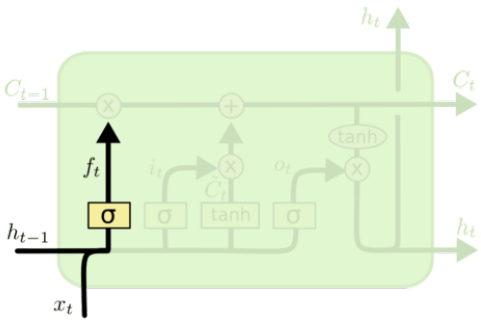



Figure 5.4: LSTM 原理

细胞状态的能力。门是一种让信息选择式通过的方法。他们包含一个 sigmoid 神经网络层和一个 pointwise 乘法操作。Sigmoid 层输出 0 到 1 之间的数值，描述每个部分有多少量可以通过。0 代表“不许任何量通过”，1 就指“允许任意量通过”!

(a) 遗忘门
在 LSTM 中第一步是我们决定会从细胞状态中丢弃什么信息，该门会读取 $h_{t-1}$ 和 $x_t$，输出一个在 0 到 1 之间的数据给每个在细胞状态 $C_{t-1}$ 中的数字。1 表示完全保留，0 表示完全舍弃。

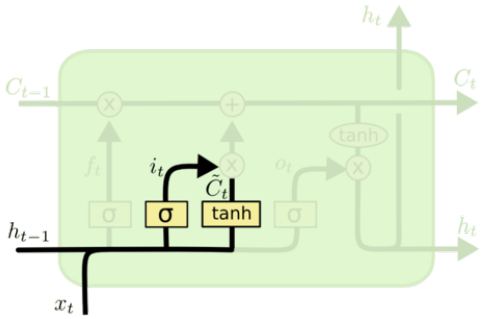



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

Figure 5.5: LSTM 遗忘门

(b) 输入门
下一步是确定什么样的新信息被存放在细胞状态中。这里包含两个部分。第一，sigmoid 称“输入门层”决定什么值我们将要更新。然后，一个 tanh 层创建一个新的候选值向量，$\tilde{C}_t$，会被

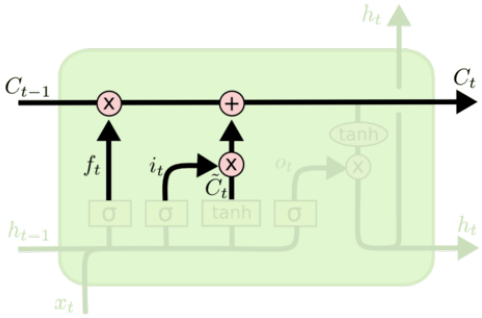
加入到状态中。下一步，我们会讲这两个信息来产生对状态的更新。




$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

Figure 5.6: LSTM 输入门

(c) 更新状态我们把旧状态与 $f_t$ 相乘，丢弃掉我们确定需要丢弃的信息。接着加上 $i_t * \tilde{C}_t$。这就是新的候选值，根据我们决定更新每个状态的程度进行变化。

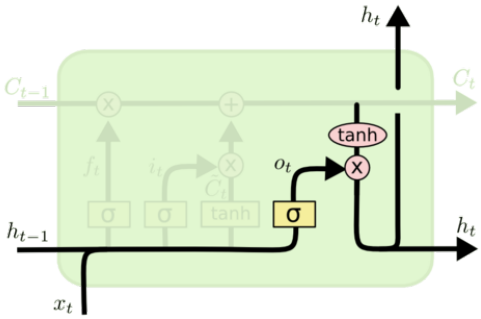



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Figure 5.7: LSTM 输入门

(d) 输出门最后我们需要确定输出什么值，这个输出将会基于我们的细胞状态。首先，我们运行一个 sigmoid 层来确定细胞状态的哪个部分将输出出去。接着，我们把细胞状态通过 tanh 进行处理（得到一个在 -1 到 1 之间的值）并将它和 sigmoid 门的输出相乘，最终我们仅仅会输出我们确定输出的那部分。

4. 为什么 LSTM 可以解决梯度消失？Gradient Vanish 的本质原因是矩阵高次幂导致的。
使用一个合适激活函数，它的梯度在一个合理的范围。LSTM 使用 gate function，有选择的让一部分信息通过。gate 是由一个 sigmoid 单元和一个逐点乘积操作组成，sigmoid 单元输出 1 或 0，用来判断




$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

Figure 5.8: LSTM 输出门

通过还是阻止，然后训练这些 gate 的组合。所以，当 gate 是打开的（梯度接近于 1），梯度就不会 vanish。并且 sigmoid 不超过 1，那么梯度也不会 explode。

5. 梯度裁剪

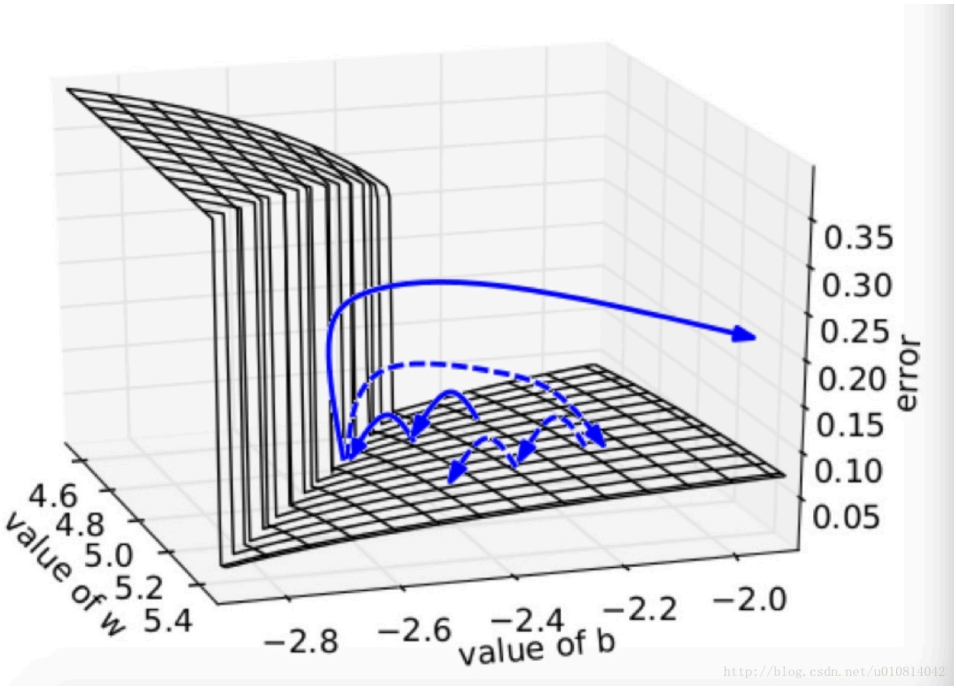



Figure 5.9: 梯度裁剪

- 首先设置一个梯度阈值：clip_gradient

- 在后向传播中求出各参数的梯度，这里我们不直接使用梯度进去参数更新，我们求这些梯度的 l2 范数
- 然后比较梯度的 l2 范数 $||g||$ 与 clip_gradient 的大小
- 如果前者大，求缩放因子 clip_gradient||g||，　由缩放因子可以看出梯度越大，则缩放因子越小，这样便很好地控制了梯度的范围
- 最后将梯度乘上缩放因子便得到最后所需的梯度

6. GRU 它是 LSTM 的简化版本，但在大多数任务中其表现与 LSTM 不相伯仲，因此也成为了常用的 RNN 算法之一。门控制循环单元（gated recurrent unit，GRU）网络是另一种基于门控制的循环神经网络，GRU 的网络结构相比 LSTM 要简单一些。GRU 将 LSTM 中的输入门和遗忘门合并成了一个门，称为更新门（update gate）。在 GRU 网络中，没有 LSTM 网络中的内部状态和外部状态的划分，而是通过直接在当前网络的状态 其中，r,z 分别被称为 Reset Gate 和

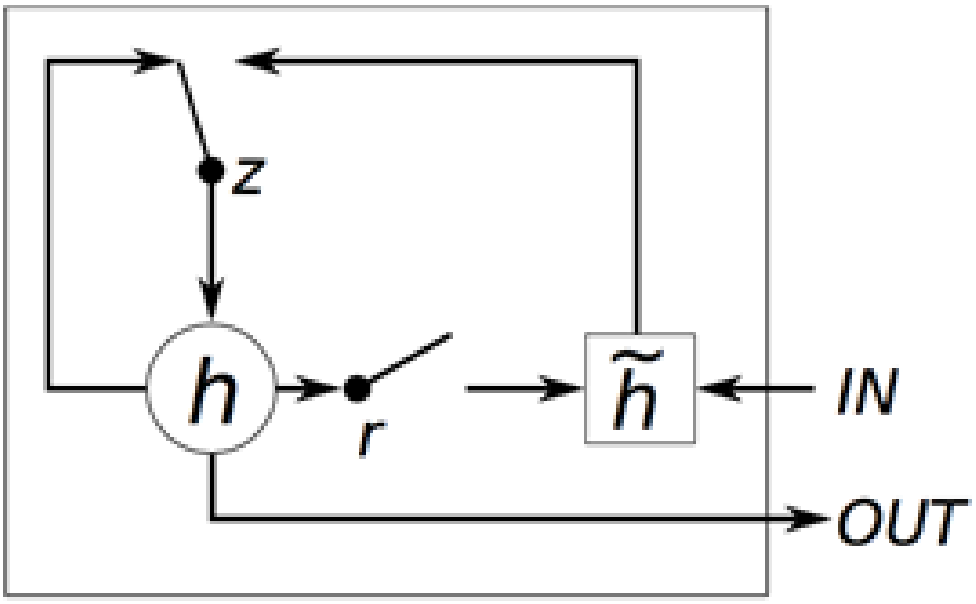



Figure 5.10: LSTM 输出门

Update Gate。可以看出，GRU 与 LSTM 有一定的相似性，而区别主要在于：

(a) LSTM 有三个 Gate，而 GRU 仅两个

(b) GRU 没有 LSTM 中的 Cell，而是直接计算输出

(c) GRU 中的 Update Gate 类似于 LSTM 中 Input Gate 和 Forget Gate 的融合；而观察它们结构中与上一时刻相连的 Gate，就能看出 LSTM 中的 Forget Gate 其实分裂成了 GRU 中的 Update Gate 和 Reset Gate

很多实验都表明 GRU 跟 LSTM 的效果差不多，而 GRU 有更少的参数，因此相对容易训练且过拟合的问题要轻一点，在训练数据较少时可以试试。