机器学习中的矩阵、向量求导

写在前面

本文的目标读者是想快速掌握矩阵、向量求导法则的学习者，主要面向矩阵、向量求导在机器学习中的应用。因此，本教程而非一份严格的数学教材，而是希望帮助读者尽快熟悉相关的求导方法并在实践中应用。另外，本教程假定读者熟悉一元函数的求导。

所谓**矩阵求导，本质上只不过是多元函数求导**，仅仅是把把函数的自变量以及求导的结果排列成了矩阵的形式，方便表达与计算而已。复合函数的求导法则本质上也是多元函数求导的链式法则，只是将结果整理成了矩阵的形式。只是对矩阵的每个分量逐元素地求导太繁琐而且容易出错，因此推导并记住一些常用的结论在实践中是非常有用的。

矩阵求导本身有很多争议，例如：

- 对于求导结果是否需要转置？
  - 不同教材对此处理的结果不一样，这属于不同的Layout Convention。本文**以不转置为准**，即求导结果与原矩阵/向量同型，术语叫Mixed Layout。
- 矩阵对向量、向量对矩阵、矩阵对矩阵求导的结果是什么？
  - 最自然的结果当然是把结果定义成三维乃至四维张量，但是这并不好算。也有一些绕弯的解决办法（例如把矩阵抻成一个向量等），但是这些方案都不完美（例如复合函数求导的链式法则无法用矩阵乘法简洁地表达等）。在本教程中，我们认为，**这三种情形下导数没有定义**。凡是遇到这种情况，都通过其他手段来绕过，后面会有具体的示例。

因此，本教程的符号体系有可能与其他书籍或讲义不一致，求导结果也可能不一致（例如相差一次矩阵转置，或者是结果矩阵是否平铺成向量等），使用者需自行注意。另外，本教程中有很多笔者自己的评论，例如关于变形的技巧、如何记忆公式、如何理解其他的教程中给出的和本教程中形式不同的结果等。

文中如有错漏，欢迎联系 ruanchong_ruby@163.com，我会尽快订正。

符号表示

- 标量用普通小写字母或希腊字母表示，如t, α等。
- 向量用粗体小写字母或粗体希腊字母表示，如 $\mathbf{x}$ 等，其元素记作 x_i（注意这里 x 没有加粗。加粗的小写字母加下标，例如 $\mathbf{x}_1, \mathbf{x}_2$等，表示这是两个不同的常数向量）。向量默认为列向量，行向量需要用列向量的转置表示，例如 $\mathbf{x}^T$ 等。
- 矩阵用大写字母表示，如A等，其元素记作a_{ij}（注意这里 a 用的是小写字母。大写字母加下标，例如 A_1, A_2 等，表示不同的常数矩阵）。
- 用字母表中靠前的字母（如 a, b, c 等）表示常量，用 f, g, h 或字母表中靠后的字母（如 u, v 等）等表示变量或函数。
- 有特殊说明的除外。

综上所述，本文进行如下约定：

- 矩阵/向量值函数对实数的导数：
  - 要点：求导结果与函数值同型，且每个元素就是函数值的相应分量对自变量 x 求导
  - 若函数$F : \mathbf{R} \rightarrow \mathbf{R}^{\mathbf{m} \times \mathbf{n}}$，则$\partial F / \partial x$也是一个$m \times n$维矩阵，且$(\partial F / \partial x)_{ij} = \partial f_{ij} / \partial x$，也可用劈形算子将导数记作$\nabla_x F$，或记作 F'_x。
  - 由于向量是矩阵的特殊情形，根据上面的定义也可以得到自变量为向量时的定义：若函数$\mathbf{f} : \mathbf{R} \rightarrow \mathbf{R}^{\mathbf{m}}$，则$\partial \mathbf{f} / \partial x$也是一个m维向量，且$(\partial \mathbf{f} / \partial x)_i = \partial f_i / \partial x$。若函数值 $\mathbf{f}^T$ 是行向量则结果为行向量，可记作 $\nabla_x \mathbf{f}^T$ 或 $\partial \mathbf{f}^T / \partial x$；若函数值 $\mathbf{f}$ 是列向量则求导结果为列向量，可记作 $\nabla_x \mathbf{f}$ 或 $\partial \mathbf{f} / \partial x$。
  - 注：本文开头即说明过，变量为向量时仅仅是将其看作多个实数，无所谓行向量与列向量之分。这里用行向量或列向量的说法仅仅为了把公式用矩阵相乘的方式表示出来方便，因为在数学公式总要指定向量是行向量或者列向量中的某一个，才能与公式里的其他部分做矩阵运算时维度相容。下同。
- 实值函数对矩阵/向量的导数：
  - 要点：求导结果与自变量同型，且每个元素就是f对自变量的相应分量求导
  - 若函数$f : \mathbf{R}^{\mathbf{m} \times \mathbf{n}} \rightarrow \mathbf{R}$，则$\partial f / \partial X$也是一个$m \times n$维矩阵，且$(\partial f / \partial X)_{ij} = \partial f / \partial x_{ij}$。也可使用劈形算子将导数记作 $\nabla_X f$。
  - 由于向量是矩阵的特殊情形，根据上面的定义也可以得到自变量为向量时的定义：若函数 $f : \mathbf{R}^{\mathbf{m}} \rightarrow \mathbf{R}$，则 $\partial f / \partial \mathbf{x}$ 也是一个m维向量，且$(\partial f / \partial \mathbf{x})_i = \partial f / \partial x_i$。若自变量是行向量则结果为行向量，可记作 $\nabla_{\mathbf{x}^T} f$ 或 $\partial f / \partial \mathbf{x}^{\mathbf{T}}$；若自变量是列向量则求导结果为列向量，可记作 $\nabla_{\mathbf{x}} f$ 或 $\partial f / \partial \mathbf{x}$ 。

- 向量值函数对向量的导数（**雅克比矩阵**）：
  - 若函数$\mathbf{f}:\mathbf{R^n}\rightarrow\mathbf{R^m}$，则 $\partial\mathbf{f}/\partial\mathbf{x}$ 是一个 $m\times n$ 维矩阵，且$(\partial\mathbf{f}/\partial\mathbf{x})_{ij}=\partial f_i/\partial x_j$。用劈形算子表示时可记作$\nabla_{\mathbf{x}}\mathbf{f}$。
  - 注：如前所述，本教程仅仅是把变量都看成多个实数，无所谓行与列之分，因此在表述从向量 $\mathbf{x}\in\mathbf{R}^n$ 到 $\mathbf{f}\in\mathbf{R}^m$ 的雅克比矩阵时，不区分 $\mathbf{x}$ 或者 $\mathbf{f}$ 到底是行向量还是列向量，统一用 $\nabla_{\mathbf{x}}\mathbf{f}$ 表示，维度也都是 m-by-n。有些教程可能会区分行对列、列对列、行对行、列对行几种不同情形的求导，认为有些结果相差一个转置，有些组合不能求导等等。本教程则认为只有一种求导结果，就是雅克比矩阵。
  - 有一点需要注意的是，若 $\mathbf{f}$ 退化成标量 f，则 $\mathbf{x}$ 到 $\mathbf{f}$ 的雅克比矩阵 $\dfrac{\partial\mathbf{f}}{\partial\mathbf{x}}$ 是一个行向量，是梯度（列向量）的转置，即 $\dfrac{\partial\mathbf{f}}{\partial\mathbf{x}}=\left(\dfrac{\partial f}{\partial\mathbf{x}}\right)^T$。注意这里使用的记号：左边 $\mathbf{f}$ 加粗，是把它看做一个长度为 1 的向量，表示求向量 $\mathbf{x}$ 到向量 $\mathbf{f}$ 的雅克比矩阵；右边 f 为普通字体，表示实函数 f 对向量 $\mathbf{x}$ 的导数。
- 劈形算子∇：
  - 在求导的变量比较明确时，可以省略劈形算子的下标写成 ∇f。
  - 劈形算子和偏导数两种记号大体上可以认为是相同的，只不过在涉及到变量分量的推导过程（例如用链式法则推神经网络的BP算法）中，偏导数那一套符号更加常用；而劈形算子的优势是书写简单，在对传统的机器学习模型的目标函数求导时，劈形算子有时更常用。
  - 对于一个实函数$f:\mathbf{R^m}\rightarrow\mathbf{R}$，其梯度记为 $\nabla f=\dfrac{\partial f}{\partial\mathbf{x}}$，也可记作$\mathbf{grad}f$，是一个 m 维向量。Hessian矩阵记为 $\nabla^2 f$，其中 $(\nabla^2 f)_{ij}=\dfrac{\partial^2 f}{\partial x_i\partial x_j}$，是一个 $m\times m$ 的矩阵。根据上述定义可以发现，Hessian 矩阵其实是 $\mathbf{x}$ 到 ∇f 的雅克比矩阵，因此 $\nabla^2 f$ 不光是一个形式记号，而是可以用 $\nabla^2 f=\nabla(\nabla f)$ 来计算。
    - 注：某些教材区分对行向量和列向量求导，认为 Hessian 矩阵是先对行向量 $\mathbf{x}^T$ 求导，再对列向量 $\mathbf{x}$ 求导（或者反过来），因此写作 $\dfrac{\partial^2 f}{\partial\mathbf{x}\partial\mathbf{x}^T}$（或者 $\dfrac{\partial^2 f}{\partial\mathbf{x}^T\partial\mathbf{x}}$）。
  - 对于一个实函数$f:\mathbf{R^{m\times n}}\rightarrow\mathbf{R}$，其梯度规定为 $m\times n$ 维矩阵 $\nabla f=\dfrac{\partial f}{\partial X}$，Hessian矩阵不作定义。

对上述约定的理解

- 对于实值函数 f，上面的定义满足转置关系（f 对某个变量和其转置的导数互为转置）：
  - 即：$\nabla_x f=(\nabla_{x^T}f)^T$（其中 x 代表任意维度的向量或矩阵）
- 函数增量的线性主部与自变量增量的关系：
  - 实值函数对矩阵/向量的导数：
    - $\delta f\approx\sum_{i,j}(\nabla_X f)_{ij}(\delta X)_{ij}=tr((\nabla f)^T\delta X)$
      - 此式用到的技巧非常重要：**两个同型矩阵对应元素相乘再求和时常用上面第二个等式转化为迹**，从而简化表达和运算。
      - 从另一个角度讲，这是矩阵导数的另一种定义。即：对于函数$f(X):\mathbf{R^{m\times n}}\rightarrow\mathbf{R}$，若存在矩阵$A$，使得$||\delta X||\rightarrow 0$时（$||\cdot||$**为任意范数**），成立$\delta f=tr(A^T\delta X)+o(||\delta X||)$，则定义$\nabla f=A$。
      - **矩阵乘积的迹是一个线性算子**。事实上，如果有两个同型矩阵 A,B，他们的内积即定义为 $<A,B>=tr(A^TB)$。容易验证，向量内积也符合这个定义，因此此式可以看成是向量内积的推广。
    - $\delta f\approx(\nabla f)^T\delta\mathbf{x}$
      - 此式右边是向量内积，可看做前一个式子的退化情形。
  - 向量值函数对向量的导数：
    - $\delta\mathbf{f}\approx(\nabla_{\mathbf{x}}\mathbf{f})\delta\mathbf{x}$
      - 此式即为重积分换元时用于坐标变换的Jacobian矩阵。

变量多次出现的求导法则

规则：**若在函数表达式中，某个变量出现了多次，可以单独计算函数对自变量的每一次出现的导数，再把结果加起来。**

这条规则很重要，尤其是在推导某些共享变量的模型的导数时很有用，例如 antoencoder with tied weights（编码和解码部分的权重矩阵互为转置的自动编码器）和卷积神经网络（同一个 feature map 中卷积核的权重在整张图上共享）等。

举例（该规则对向量和矩阵也是成立的，这里先用标量举一个简单的例子）：假设函数表达式是 $f(x)=(2x+1)x+x^2$，可以先把三个 x 看成三个不同的变量，即把 f 的表达式看成 $(2x_1+1)x_2+x_3^2$，然后分别计算 $\partial f/\partial x_1=2x_2$，$\partial f/\partial x_2=2x_1+1$，和 $\partial f/\partial x_3=2x_3$，最后总的导数就是这三项加起来：$2x_2+(2x_1+1)+2x_3$，此时再把 x 的下标抹掉并化简，就得到 $6x+1$。熟悉这个过程之后，可以省掉添加下标再移除的过程。

如果用计算图（computation graph，描述变量间依赖关系的示意图，后面会举例）的语言来描述本条法则，就是：若变量 x 有多条影响函数 f 的值的路径，则计算 $\partial f/\partial x$ 时需要对每条路径求导最后再加和。 如果想更多地了解计算图和反向传播，推荐阅读 Colah 君的文章。其中详细讲述了计算图如何工作，不仅讲反向传播还讲了前向传播（前向传播对于目前的机器学习算法来说似乎没有太大的用处，但是对于加深计算图的理解很有帮助。RNN 有一种学习算法叫 RTRL 就是基于前向传播的，不过近年来不流行了）。

有了上面的基础，我们就可以推导 Batch normalization（以下简称 BN）的求导公式了。 BN 的计算过程为：

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\ldots m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_\mathcal{B} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

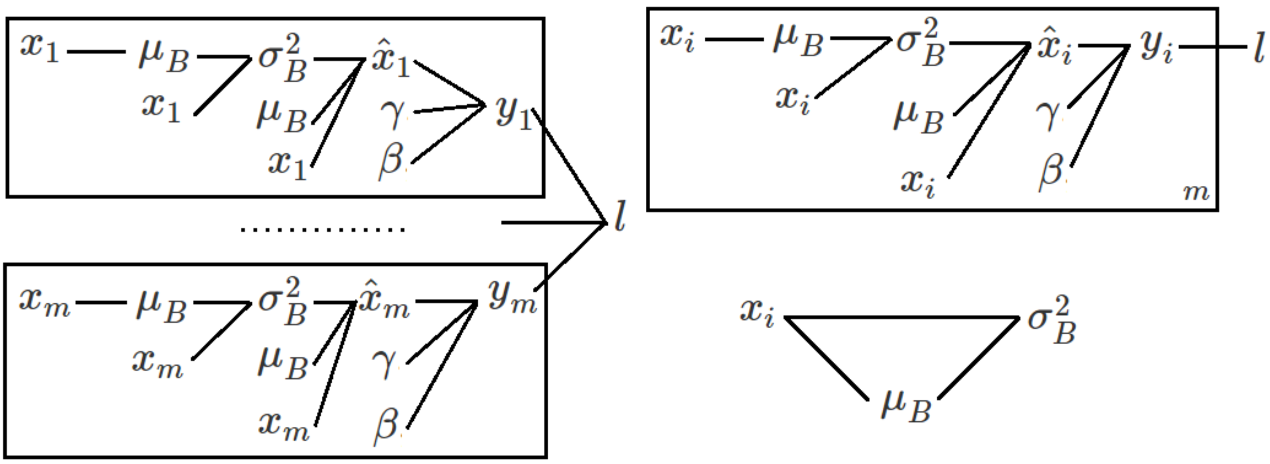
$$\sigma_\mathcal{B}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_\mathcal{B})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_\mathcal{B}}{\sqrt{\sigma_\mathcal{B}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

其中 m 是批的大小，x_1 到 x_m 分别是 m 个不同样本对于某个神经元的输入，l 是这个批的总的损失函数，所有变量都是标量。求导的第一步是画出变量依赖图，如下所示（根据左边的变量可以计算出右边的变量，如果为了强调，也可以在边上添加从左向右的箭头）：



左侧，右上，右下分别是三种不同的画法（读者也可以尝试其他的画法）：左边的图是把所有变量 x_i 都画了出来，比较清楚，如果想不清楚变量之间是如何相互依赖的，这样画可以帮助梳理思路；右上是我自创的一种方法，借鉴了概率图模型中的盘记号（plate notation），把带下标的变量用一个框框起来，在框的右下角指明重复次数；右下我只画了一个局部，只是为了说明在有些资料中，相同的变量（如本例中的 μ_B）只出现一次，而非像左图那样出现多次，从而图中会出现环。不过要不要复制同一个变量的多个拷贝没有本质的区别。在右下这种表示法中，如果要求 $\dfrac{\partial \sigma_B^2}{\partial x_i}$，需要对$x_i \to \sigma_B^2$ 和 $x_i \to \mu_B \to \sigma_B^2$ 这两条路径求导的结果做加和。（事实上，这种带下标的画法有点儿丑，因为我们现在的计算图里的变量都是标量……如果用 $\mathbf{x}$ 表示 $x_{1..m}$ 组成的向量，计算图会更简洁，看起来更舒服。不过这种丑陋的表示对于我们现在的目的已经够用了。）

BN 原论文中也给出了反向传播的公式，不过我们不妨试着自己手算一遍：

- $\hat{x}_i$ 影响损失函数只有唯一的路径 $\hat{x}_i \to y_i \to l$，根据链式法则，得到：$\frac{\partial l}{\partial \hat{x}_i} = \frac{\partial l}{\partial y_i}\frac{\partial y_i}{\partial \hat{x}_i} = \frac{\partial l}{\partial y_i}\gamma$。
- γ 影响损失函数有 m 条路径：对任意一个 i，$\gamma \to y_i \to l$ 都是一条路径，需要对这些路径分别求导再加和：$\frac{\partial l}{\partial \gamma} = \sum_i \frac{\partial l}{\partial y_i}\frac{\partial y_i}{\partial \gamma} = \sum_i \frac{\partial l}{\partial y_i}\hat{x}_i$。
- $\frac{\partial l}{\partial \beta}$ 的计算与上面类似：$\frac{\partial l}{\partial \beta} = \sum_i \frac{\partial l}{\partial y_i}\frac{\partial y_i}{\partial \beta} = \sum_i \frac{\partial l}{\partial y_i} \cdot 1 = \sum_i \frac{\partial l}{\partial y_i}$。

- σ_B^2 影响损失函数的路径也有 m 条：$\forall i, \sigma_B^2 \to \hat{x}_i \to l$（此处忽略中间变量 y_i，直接把 l 看成的 $\hat{x}_i$ 函数。）所以 $\frac{\partial l}{\partial \sigma_B^2} = \sum_i \frac{\partial l}{\partial \hat{x}_i} \frac{\partial \hat{x}_i}{\partial \sigma_B^2} = \sum_i \frac{\partial l}{\partial \hat{x}_i} \cdot -\frac{1}{2}(x_i - \mu_B)(\sigma_B^2 + \epsilon)^{-3/2}$。注意求导的时候把 σ_B^2 当成一个整体，想象这就是一个字母，而不要把它想成标准差的平方。
- μ_B 影响损失函数共有 $2m$ 条路径：$\forall i, \mu_B \to \hat{x}_i \to l, \mu_B \to \sigma_B^2 \to l$（分别对应于右上图中较短和较长的路径）。故有：$\frac{\partial l}{\partial \mu_B} = \sum_i (\frac{\partial l}{\partial \hat{x}_i} \frac{\partial \hat{x}_i}{\partial \mu_B} + \frac{\partial l}{\partial \sigma_B^2} \frac{\partial \sigma_B^2}{\partial \mu_B}) = \sum_i \frac{\partial l}{\partial \hat{x}_i} \frac{-1}{\sqrt{\sigma_B^2+\epsilon}} + \sum_i \frac{\partial l}{\partial \sigma_B^2} \sum_j \frac{-2}{m}(x_j - \mu_B) = \sum_i \frac{\partial l}{\partial \hat{x}_i} \frac{-1}{\sqrt{\sigma_B^2+\epsilon}}$。其中最后一步的理由是根据 μ_B 的定义，后一项为零。
- $\forall i, x_i$ 影响损失函数有 3 条路径：$x_i \to \hat{x}_i \to l, x_i \to \sigma_B^2 \to l, x_i \to \mu \to l$ 所以 $\frac{\partial l}{\partial x_i} = \frac{\partial l}{\partial \hat{x}_i} \frac{\partial \hat{x}_i}{\partial x_i} + \frac{\partial l}{\partial \sigma_B^2} \frac{\partial \sigma_B^2}{\partial x_i} + \frac{\partial l}{\partial \mu_B} \frac{\partial \mu_B}{\partial x_i} = \sum_i \frac{\partial l}{\partial \hat{x}_i} \frac{1}{\sqrt{\sigma_B^2+\epsilon}} + \frac{\partial l}{\partial \sigma_B^2} \frac{2}{m}(x_i - \mu_B) + \frac{\partial l}{\partial \mu_B} \frac{1}{m}$

常用公式

向量求导的链式法则

- 易发现**雅克比矩阵的传递性**：若多个向量的依赖关系为 $\mathbf{u} \to \mathbf{v} \to \mathbf{w}$，则：$\frac{\partial \mathbf{w}}{\partial \mathbf{u}} = \frac{\partial \mathbf{w}}{\partial \mathbf{v}} \frac{\partial \mathbf{v}}{\partial \mathbf{u}}$。
  - 证明：只需逐元素求导即可。$\frac{\partial w_i}{\partial u_j} = \sum_k \frac{\partial w_i}{\partial v_k} \frac{\partial v_k}{\partial u_j}$，即 $\frac{\partial \mathbf{w}}{\partial \mathbf{u}}$ 的 (i,j) 元等于矩阵 $\frac{\partial \mathbf{w}}{\partial \mathbf{v}}$ 的 i 行 和 矩阵 $\frac{\partial \mathbf{v}}{\partial \mathbf{u}}$ 的第 j 列的内积，这正是矩阵乘法的定义。
  - 注：**将两项乘积的和转化成向量内积或矩阵相乘来处理，是很常用的技巧。**
- 雅克比矩阵的传递性可以很容易地推广到多层中间变量的情形，采用数学归纳法证明即可。
- 若中间变量都是向量，但最后的结果变量是一个实数，例如变量依赖关系形如 $\mathbf{x} \to \mathbf{v} \to \mathbf{u} \to f$，则：
  - 由雅克比矩阵的传递性知：$\frac{\partial \mathbf{f}}{\partial \mathbf{x}} = \frac{\partial \mathbf{f}}{\partial \mathbf{u}} \frac{\partial \mathbf{u}}{\partial \mathbf{v}} \frac{\partial \mathbf{v}}{\partial \mathbf{x}}$。再根据 $\mathbf{f}$ 退化时雅克比矩阵和函数导数的关系，有：$\frac{\partial \mathbf{f}}{\partial \mathbf{x}} = \frac{\partial f}{\partial \mathbf{x}^T}, \frac{\partial \mathbf{f}}{\partial \mathbf{u}} = \frac{\partial f}{\partial \mathbf{u}^T}$。以上三式相结合，可以得到如下链式法则：$\frac{\partial f}{\partial \mathbf{x}^T} = \frac{\partial f}{\partial \mathbf{u}^T} \frac{\partial \mathbf{u}}{\partial \mathbf{v}} \frac{\partial \mathbf{v}}{\partial \mathbf{x}}$
  - 上面的结果显然也可以推广到任意多层复合的情形（可用于 RNN 的 BPTT 的推导）。
- 上面的公式是把导数视为行向量（即以 $\frac{\partial f}{\partial \mathbf{x}^T}$ 和 $\frac{\partial f}{\partial \mathbf{u}^T}$ 的形式）给出的。如果需要把导数视为列向量，只需将公式两边同时转置即可。由于实践中复合一次的情形较常用，这里只给出将变量视为列向量时复合一次的公式：
  - 若 $y = f(\mathbf{u}), \mathbf{u} = \mathbf{g}(\mathbf{x})$，则：$\frac{\partial f}{\partial \mathbf{x}} = \left(\frac{\partial \mathbf{u}}{\partial \mathbf{x}}\right)^T \frac{\partial f}{\partial \mathbf{u}}$，或写作 $\nabla_{\mathbf{x}} f = (\nabla_{\mathbf{x}} \mathbf{u})^T \nabla_{\mathbf{u}} f$
    - 这里再给出一个特例：若变量依赖关系为 $\mathbf{x} \to \mathbf{u} \to f$，$\mathbf{u}$ 和 $\mathbf{x}$ 维度相同且 u_i 仅由 x_i 计算出而与 x 的其他分量无关，则易知 $\frac{\partial \mathbf{u}}{\partial \mathbf{x}}$ 是对角阵，所以上面的公式可以化简为：$\frac{\partial f}{\partial \mathbf{x}} = \mathrm{vec}\left(\frac{\partial \mathbf{u}}{\partial \mathbf{x}}\right) \odot \frac{\partial f}{\partial \mathbf{u}}$，其中 $\mathrm{vec}(D)$ 表示取对角矩阵 D 的对角线上的元素组成列向量，$\odot$ 表示两个向量逐元素相乘。
    - 由于最终的结果是两个向量逐元素相乘，所以也可以交换一下相乘的顺序，写成：$\frac{\partial f}{\partial \mathbf{x}} = \frac{\partial f}{\partial \mathbf{u}} \odot \mathrm{vec}\left(\frac{\partial \mathbf{u}}{\partial \mathbf{x}}\right)$。
    - 本条规则在神经网络中也很常用，常见的情形包括但不限于：逐元素地应用激活函数（$\mathbf{z} \to \mathbf{a} = \sigma(\mathbf{z}) \to l$），以及现代 RNN 单元中的门限操作（以 LSTM 为例：$c_{t-1} \to c_t = f_t \odot c_{t-1} + (1 - f_t) \odot \hat{c}_t \to l$）。
    - 因为依赖关系简单，本公式也可以直接根据导数逐分量的定义直接推出来：$\frac{\partial f}{\partial x_i} = \sum_k \frac{\partial f}{\partial u_k} \frac{\partial u_k}{\partial x_i} = \frac{\partial f}{\partial u_i} \frac{\partial u_i}{\partial x_i}$，此即前述公式的分量形式。
- 记忆：只需记住结果是一堆雅克比矩阵的乘积，相乘的顺序根据**维度相容原则**调整即可（假设每个中间变量的维度都不一样，看怎么摆能把雅克比矩阵的维度摆成矩阵乘法规则允许的形式。只要把矩阵维度倒腾顺了，公式也就对了。）
- 注：网络上各种资料质量参差不齐，在其他教程中时常会见到向量对矩阵求导的表达式。例如介绍RNN的梯度消失问题的文章中，经常会见到 $\frac{\partial l}{\partial W} = \frac{\partial l}{\partial \mathbf{h}_T} \frac{\partial \mathbf{h}_T}{\partial \mathbf{h}_{T-1}} \cdots \frac{\partial \mathbf{h}_{i+1}}{\partial \mathbf{h}_i} \frac{\partial \mathbf{h}_i}{\partial W}$ 这种式子。如果文中出现这个式子是定性的，只是为了说明链式法则中出现了很多连乘项导致了梯度消失，那么读者也只需定性地理解即可。如果文中出现这个式子是定量的，是为了推导反向传播的公式，那么笔者建议读者用如下两种方式之一理解：
  - 其一是把 $\frac{\partial \mathbf{h}_i}{\partial W}$ 理解成一种简写形式：先把 W 抻成一个向量，然后公式中的每一个雅克比矩阵就都可以计算了，最后再把结果向量重新整理成 W 的同型矩阵。但是这种方法非常复杂，因为把 W 抻成向量以后目标函数关于 W 的表达式就变了，很难推导 $\frac{\partial \mathbf{h}_i}{\partial W}$ 这个雅克比矩阵。一个具体的算例见 Optimizing RNN performance 一文中最后的推导。（如果你不打算熟练掌握这种方法，只浏览一下看看大意即可。相信我，如果你学了本文中的方法，你不会再想用这种把矩阵抻开的方法求导的。）
  - 其二是把最后一项分母中的 W 理解成矩阵 W 中的任一个元素 w_{ij}，从而上述表达式中的四项分别是向量（此处看作行向量）、矩阵、矩阵、向量（列向量），从而该表达式可以顺利计算。但是这也很麻烦，因为得到的结果不是直接关于

W 的表达式，而是关于其分量的，最后还要合并起来。
- 其他理解方式，恕我直言，基本上都是作者自己就没弄懂瞎糊弄读者的。

实值函数对向量求导

- 未作特殊说明即为对变量 $\mathbf{x}$ 求导。
- 几个基本的雅克比矩阵：
  - $\nabla A\mathbf{x} = A$，特别地，$\nabla\ \mathbf{x} = I$。
- 向量内积的求导法则：
  - 内积是一个实数，因此本节相当于实数对向量求导，结果是与自变量同型的向量。
  - $\nabla(\mathbf{a^T x}) = \mathbf{a}$
    - 这是最基本的公式，正确性是显然的，因为 $\dfrac{\partial \mathbf{a^T x}}{\partial x_i} = \dfrac{\partial \sum_j a_j x_j}{\partial x_i} = \dfrac{\partial a_i x_i}{\partial x_i} = a_i$。
  - $\nabla||\mathbf{x}||_2^2 = \nabla(\mathbf{x^T x}) = 2\mathbf{x}$
    - 正确性是显然的，因为 $\dfrac{\partial ||\mathbf{x}||_2^2}{\partial x_i} = \dfrac{\partial \sum_j x_j^2}{\partial x_i} = \dfrac{\partial x_i^2}{\partial x_i} = 2x_i$。另外，也可以用变量多次出现的求导法则结合上一条公式证明。
  - $\nabla(\mathbf{x}^T A\mathbf{x}) = (A + A^T)\mathbf{x}$
    - 利用变量多次出现的求导法则以及前面的公式容易证明。另外，若 A 是对称矩阵，上式右边可以化简为 $2A\mathbf{x}$。
  - 向量内积的求导法则：$\nabla(\mathbf{u^T v}) = (\nabla_\mathbf{x}\mathbf{u})^T\mathbf{v} + (\nabla_\mathbf{x}\mathbf{v})^T\mathbf{u}$
    - 利用变量多次出现的求导法则（$\mathbf{x}$ 同时在 $\mathbf{u}, \mathbf{v}$ 中出现）+复合函数求导法则（列向量形式）易证。

向量数乘求导公式

- $\nabla_\mathbf{x}(\alpha(\mathbf{x})\mathbf{f}(\mathbf{x})) = \mathbf{f}(\mathbf{x})\nabla_{\mathbf{x^T}}\alpha(\mathbf{x}) + \alpha(\mathbf{x})\nabla_\mathbf{x}\mathbf{f}(\mathbf{x})$
  - 推导：$\dfrac{\partial \alpha f_i}{\partial x_j} = f_i\dfrac{\partial \alpha}{\partial x_j} + \alpha\dfrac{\partial f_i}{\partial x_j}$，两边逐分量对比一下便知等式成立。
  - 记忆：按两个标量函数相乘的求导法则记，再注意一下**维度相容原理**即可。向量数乘的结果还是一个向量，所以此处相当于向量对向量求导，结果是一个雅克比矩阵，形状为 $\mathbf{f}$ 的维度 乘 $\mathbf{x}$ 的维度。

矩阵迹求导

- 未作特殊说明即为对 X 求导。迹是一个实数，所以相当于实数对矩阵求导，结果是一个和 X 同型的矩阵。
- 先回顾一下迹的基本性质：
  - 线性性质：$tr(\sum_i c_i A_i) = \sum_i c_i tr(A_i)$
  - 转置不变性：$tr(A) = tr(A^T)$
  - 轮换不变性：$tr(A_1A_2\cdots A_n) = tr(A_2A_3\cdots A_nA_1) = \cdots = tr(A_{n-1}A_nA_1\cdots A_{n-2}) = tr(A_nA_1\cdots A_{n-2}A_{n-1})$。特别地，$tr(AB) = tr(BA)$。**注意，轮换不变性不等于交换性**。例如：$tr(ABC) = tr(BCA) = tr(CAB)$，但是一般情况下 $tr(ABC) \neq tr(ACB)$。
- 基本公式：
  - $\nabla tr(A^TX) = \nabla tr(AX^T) = A$
  - 推导：逐元素求导验证：$\dfrac{\partial tr(A^TX)}{\partial x_{ij}} = \dfrac{\partial \sum_{i,j}(a_{ij}x_{ij})}{\partial x_{ij}} = a_{ij}$。（事实上这个公式就是矩阵导数的另一种定义，前面也有叙述。）
  - 根据此式容易得到另一个式子：$\nabla tr(AX) = \nabla tr(XA) = A^T$
- 迹方法的核心公式（非常重要）：
  - $\nabla tr(XAX^TB) = B^TXA^T + BXA$
  - 推导：利用变量多次出现的求导法则：

$$\begin{aligned}\nabla tr(XAX^TB) &= \nabla tr(XAX_c^TB) + \nabla tr(X_cAX^TB) = (AX_c^TB)^T + \nabla tr(BX_cAX^T)\\ &= B^TX_cA^T + BX_cA = B^TXA^T + BXA\end{aligned}$$

  （X_c 表示将 X 的此次出现视作常数）
  - 这个公式非常重要，在推导最小二乘解等问题上都会遇到。公式的名字是我瞎起的，我不知道它叫什么名字。
- 其他与矩阵迹有关的公式
  - 大部分都是上述核心公式的简单推论，不必强记
  - $\nabla \mathbf{a}^TX\mathbf{b} = \mathbf{ab}^T$
    - 推导：$LHS = \nabla tr(\mathbf{a}^TX\mathbf{b}) = \nabla tr(X\mathbf{ba}^T) = \mathbf{ab}^T = RHS$。

        - 注：将实数看作是 1*1 矩阵的迹是很常用的技巧。
    - $\nabla \mathbf{a}^T X^T X \mathbf{a} = 2X\mathbf{a}\mathbf{a}^T$
        - 推导：使用迹方法的核心公式。过程略。
    - $\nabla (X\mathbf{a}-\mathbf{b})^T(X\mathbf{a}-\mathbf{b}) = 2(X\mathbf{a}-\mathbf{b})\mathbf{a}^T$
        - 推导：将左式的括号相乘展开，然后用上面的关于矩阵迹的公式。
    - $\nabla ||XA^T-B||_F^2 = 2(XA^T-B)A$
        - 推导同上，只需注意到 $||A||_F^2 = tr(A^TA)$ 即可。特别地，$\nabla ||X||_F^2 = \nabla(X^TX) = 2X$（此式也可逐元素求导直接验证）
- 行列式的求导公式：$\nabla_X |X| = |X|(X^{-1})^T$
    - 实数对矩阵求导，结果是和 X 同型的矩阵。此条证明较繁琐，大致过程是用逐元素求导 + 伴随矩阵的性质推导，过程可参考 math overflow。最好能直接记住。

矩阵求导的链式法则

- 设 $y=f(U), U=G(X)$，则：
    - $\frac{\partial y}{\partial x_{ij}} = \sum_{k,l} \frac{\partial y}{\partial u_{kl}} \frac{\partial u_{kl}}{\partial x_{ij}}$，或简写为$\frac{\partial y}{\partial x_{ij}} = tr((\frac{\partial y}{\partial U})^T \frac{\partial U}{\partial x_{ij}})$
    - 关于维度的说明：X 是矩阵，中间变量 U 也是矩阵（未必与 X 同型），最终结果 y 是实数。因此求导结果是和 X 同型的矩阵。
    - 注：此式似乎用的不多，毕竟这仅仅是对 x_{ij} 这一个分量求导的结果，很难直接得到对 X 求导的结果。而且这个式子只是最基础的多元函数复合的链式法则而已，没有得到什么特别有趣或者重要的结论。
- 设 $y=f(u), u=g(X)$，则：$\frac{\partial y}{\partial X} = \frac{\partial y}{\partial u}\frac{\partial u}{\partial X}$（等式右边是实数和矩阵的数乘）
    - 关于维度的说明：X, u, y 分别是矩阵、实数、实数，因此相当于实数对矩阵求导，结果是与 X 同型的矩阵。
    - 证明是显然的，逐元素求导验证即可：$\frac{\partial y}{\partial x_{ij}} = \frac{\partial y}{\partial u}\frac{\partial u}{\partial x_{ij}}$。
- 线性变换的导数（非常重要。由于线性变换很常用，记住此式可以简化很多公式的推导过程）：
    - 设有$f(Y): \mathbf{R}^{m\times p} \to \mathbf{R}$及线性映射 $X \mapsto Y = AX+B: \mathbf{R}^{n\times p} \to \mathbf{R}^{m\times p}$（因此 $A\in\mathbf{R}^{m\times n}, B \in \mathbf{R}^{m\times p}$），则：
        - $\nabla_X\ f(AX+B) = A^T\nabla_Y f$
            - 推导：$\frac{\partial f}{\partial x_{ij}} = \sum_{k,l}\frac{\partial f}{\partial y_{kl}}\frac{\partial y_{kl}}{\partial x_{ij}}$，而 $\frac{\partial y_{kl}}{\partial x_{ij}} = \frac{\partial \sum_s a_{ks}x_{sl}}{\partial x_{ij}} = \frac{\partial a_{ki}x_{il}}{\partial x_{ij}} = a_{ki}\delta_{lj}$（$\delta_{lj}$ 是 Kronecker delta 符号：若 $l=j$ 值为1，否则为0），将后式代入前式，得：$\frac{\partial f}{\partial x_{ij}} = \sum_{k,l}\frac{\partial f}{\partial y_{kl}}a_{ki}\delta_{lj} = \sum_k \frac{\partial f}{\partial y_{kj}}a_{ki}$，即矩阵 A^T 的第 i 行 和 矩阵 $\nabla_Y f$ 的第 j 列的内积。
        - 向量的线性变换是上式的退化情形，即：$\nabla_\mathbf{x}\ f(A\mathbf{x}+\mathbf{b}) = A^T\nabla_\mathbf{y} f$
        - 向量的线性变换还可以求二阶导：$\nabla_\mathbf{x}^2\ f(\mathbf{Ax}+\mathbf{b}) = A^T(\nabla_\mathbf{y}^2 f)A$
            - 推导：记 $\mathbf{u(y)} = \nabla_\mathbf{y} f, \mathbf{w(u)} = A^T\mathbf{u}$，则 $\nabla_\mathbf{x}^2\ f(\mathbf{Ax}+\mathbf{b}) = \nabla_\mathbf{x}(\nabla_\mathbf{x}\ f(A\mathbf{x}+\mathbf{b})) = \nabla_\mathbf{x}\mathbf{w} = \frac{\partial \mathbf{w}}{\partial \mathbf{x}} = \frac{\partial \mathbf{w}}{\partial \mathbf{u}}\frac{\partial \mathbf{u}}{\partial \mathbf{y}}\frac{\partial \mathbf{y}}{\partial \mathbf{x}} = A^T(\nabla_\mathbf{y}^2 f)A$
            - 记忆：同上，记住大概的形状（对线性变换来说，求一次导就是乘一个矩阵），然后根据维度相容原则摆顺了就行。
        - 由于线性变换很常用，这里不妨把给 X 右乘一个矩阵时的公式一并给出，以便查阅：设有$f(Y): \mathbf{R}^{m\times p} \to \mathbf{R}$及线性映射 $X \mapsto Y = XC+D: \mathbf{R}^{m\times n} \to \mathbf{R}^{m\times p}$（因此 $C\in\mathbf{R}^{n\times p}, D\in\mathbf{R}^{m\times p}$），则：$\nabla_X\ f(XC+D) = (\nabla_Y f)C^T$
            - 证明：若令 $\widetilde{Y} = Y^T, \widetilde{X} = X^T$，则变量依赖关系变为：$\widetilde{X} \to \widetilde{Y} \to f$，且 $\widetilde{Y} = C^T\widetilde{X} + D^T$，根据线性变换的求导法则，知：$\nabla_{\widetilde{X}}\ f = (C^T)^T\nabla_{\widetilde{Y}} f = C\nabla_{\widetilde{Y}} f$，所以 $\nabla_X\ f = (\nabla_{\widetilde{X}}\ f)^T = (C\nabla_{\widetilde{Y}} f)^T = (\nabla_{\widetilde{Y}} f)^T C^T = (\nabla_Y f)C^T$。
            - 记忆：先做线性变换再求导就等于先求导再做线性变换。剩下的细节（如左乘还是右乘等）根据维度相容原则倒腾即可。
            - 注：此式很有用，在神经网络中，经常有形如 $W \to \mathbf{z} = W\mathbf{x}+\mathbf{b} \to l$ 的依赖关系。其中 $\mathbf{x}$ 是神经网络某一层的输入数据（不是训练神经网络时要求导的变量！不过在构造对抗样本时可能需要对 $\mathbf{x}$ 求导），$W, \mathbf{b}$ 是该层的参数（这才是训练神经网络时要求导的变量），$\mathbf{z}$ 是经过变换后预备输入给下一层的值，l 是最终的损失函数。根据上述线性变换的求导公式，立即可以得到 BP 算法的核心步骤：$\nabla_W l = \nabla_\mathbf{z} l\ \mathbf{x}^T$。（另注：标准的 BP 算法通常将 $\nabla_\mathbf{z} l$ 定义为变量 δ。）

其他公式

- 这一部分在机器学习中遇到的不多（毕竟常见的情况是求一个标量损失函数对其他变量的导数），不是特别重要，不过偶尔在凸优化里会碰到一些。这里收集整理这几个式子主要是为了资料完整、查阅方便。以下假定 F 是可逆方阵。

- $(|F|)'_x = |F|tr(F^{-1}F'_x)$
  - 自变量和函数值都是实数，求导结果也是实数。推导过程较困难，主要用到了矩阵的雅克比公式（不是雅克比矩阵）。建议记住，或者用时查表。
- $(\ln|F|)'_x = tr(F^{-1}F'_x)$
  - 自变量和函数值都是实数，求导结果也是实数。
  - 推导：根据最基本的一元函数复合的求导法则即可。令 $u=|F|, y=\ln u$，则：$y'_x = y'_u u'_x = 1/u(|F|tr(F^{-1}F'_x)) = tr(F^{-1}F'_x)$。
- $(F^{-1})'_x = -F^{-1}F'_xF^{-1}$
  - 矩阵对实数求导，结果是和 F^{-1} 同型的矩阵（也即和 F 同型的矩阵）。
  - 推导：对恒等式 $FF^{-1}=I$ 两边同时求导，再结合 $|F|$ 的导数易得。

常见技巧及注意事项

- 实数在与一堆矩阵、向量作数乘时可以随意移动位置。且实数乘行向量时，向量数乘与矩阵乘法（1×1 矩阵和 $1\times m$ 矩阵相乘）的规则是一致的。
- 遇到相同下标求和就联想到矩阵乘法的定义，即$c_{ij}=\sum_j a_{ij}b_{jk}$。特别地，一维下标求和联想到向量内积$\sum_i u_iv_i = \mathbf{u}^T\mathbf{v}$，二维下标求和联想到迹 $\sum_{ij} a_{ij}b_{ij} = tr(AB^T)$（$A, B$应为同型矩阵）。
- 如果在一个求和式中，待求和项不是实数而是矩阵的乘积，不要想着展开求和式，而要按照上面的思路，看成分块矩阵的相乘！
- 向量的模长平方（或实数的平方和）转化为内积运算：$\sum_i x_i^2 = \mathbf{x}^T\mathbf{x}$。矩阵的F范数的平方转化为迹运算：$||A||_F^2 = tr(AA^T)$。
- 多个矩阵相乘时，多用矩阵迹的求导公式转化、循环移动各项。实数也可看成 1×1 矩阵的迹！
- 需要用到向量（或矩阵）对矩阵求导的情形，要么把矩阵按列拆开转化成向量对向量求导（最终很有可能通过分块矩阵乘法再合并起来。本文后面的算例 PRML (3.33) 说明了这种方法怎么用），要么套用线性变换的求导公式（常见于神经网络的反向传播过程）。

算例

最小二乘法

- 方法一：展开括号，再使用几个常用公式化简即可：
  - $$\begin{aligned}\nabla_{\mathbf{x}}||A\mathbf{x}-\mathbf{b}||_2^2 &= \nabla_{\mathbf{x}}(A\mathbf{x}-\mathbf{b})^T(A\mathbf{x}-\mathbf{b}) \\ &= \nabla_{\mathbf{x}}(\mathbf{x}^TA^TA\mathbf{x}-\mathbf{b}^TA\mathbf{x}-\mathbf{x}^TA^T\mathbf{b}+\mathbf{b}^T\mathbf{b}) \\ &= \nabla_{\mathbf{x}}(\mathbf{x}^TA^TA\mathbf{x})-2\nabla_{\mathbf{x}}(\mathbf{b}^TA\mathbf{x})+\nabla_{\mathbf{x}}(\mathbf{b}^T\mathbf{b}) \\ &= 2A^TA\mathbf{x}-2A^T\mathbf{b}+O \\ &= 2A^T(A\mathbf{x}-\mathbf{b})\end{aligned}$$
- 方法二：使用线性变换的求导公式：
  - $$\begin{aligned}\nabla_{\mathbf{x}}||A\mathbf{x}-\mathbf{b}||_2^2 &= A^T\nabla_{A\mathbf{x}-\mathbf{b}}||A\mathbf{x}-\mathbf{b}||_2^2 \\ &= A^T(2(A\mathbf{x}-\mathbf{b})) \\ &= 2A^T(A\mathbf{x}-\mathbf{b})\end{aligned}$$

F范数的求导公式推导

- 方法一：先转化为迹，再裂项，最后通过恰当的轮换，用迹方法的核心公式处理。
  - $$\begin{aligned}\nabla||XA^T-B||_F^2 &= \nabla tr((XA^T-B)^T(XA^T-B)) \\ &= \nabla tr(AX^TXA^T-B^TXA^T-AX^TB+B^TB) \\ &= \nabla tr(AX^TXA^T)-2tr(AX^TB)+tr(B^TB) \\ &= 2tr(XA^TAX^TI)-2tr(X^TBA)+O \\ &= 2(I^TX(A^TA)^T+IX(A^TA))-2BA \\ &= 2XA^TA-2BA \\ &= 2(XA^T-B)A\end{aligned}$$
- 方法二：用线性变换的求导公式证。（注意矩阵转置不改变其F范数，并且实值函数对 X 和 X^T 的导数互为转置）

- $$\begin{aligned}\nabla\|XA^T-B\|_F^2 &= \nabla\|AX^T-B^T\|_F^2 \\ &= (\nabla_{X^T}\|AX^T-B^T\|_F^2)^T \\ &= (A^T(2(AX^T-B^T)))^T \\ &= 2(XA^T-B)A\end{aligned}$$
- 方法三：根据定义逐元素地算，然后合并成向量、再合并成矩阵。（太原始，易出错，不推荐）

PRML (3.33)求导

- 题目：
  - 求$f(W)=\ln p(T|X,W,\beta)=\text{const}-\frac{\beta}{2}\sum_n\|\mathbf{t}_n-W^T\phi(\mathbf{x}_n)\|_2^2$关于 W 的导数。
  - 说明：上面的 $\phi(\mathbf{x}_n)$ 的结果应当是一个向量，但是希腊字母打不出加粗的效果。
- 方法一：用矩阵的F范数推导：
  - $$\begin{aligned}\nabla f &= \nabla\left(-\frac{\beta}{2}\sum_n\|\mathbf{t}_n-W^T\phi(\mathbf{x}_n)\|_2^2\right) \\ &= -\frac{\beta}{2}\nabla\|T^T-W^T\Phi^T\|_F^2 \\ &= -\frac{\beta}{2}\nabla\|T-\Phi W\|_F^2 \\ &= -\frac{\beta}{2}\nabla\|\Phi W-T\|_F^2 \\ &= -\frac{\beta}{2}\Phi^T(2(\Phi W-T)) \\ &= -\beta\Phi^T(\Phi W-T)\end{aligned}$$
  - 上述几步的依据分别是：
    - 将若干个列向量拼成一个矩阵，因此它们的二范数平方和就等于大矩阵的F范数的平方。
    - 矩阵转置不改变其F范数。
    - 矩阵数乘(-1)不改变其F范数。
    - 线性变换的求导公式 + F范数的求导公式。
    - 实数在和矩阵作数乘时位置可以任意移动。
  - 有了导数，再另导数等于零，即得 W 的最大似然解：$W_{ML}=\Phi^\dagger T=(\Phi^T\Phi)^{-1}\Phi^T$。
- 方法二： 将向量二范数用内积代替，然后逐项展开，最后利用分块矩阵相乘消掉求和号：
  - $$\begin{aligned}\nabla f &= \nabla\left(-\frac{\beta}{2}\sum_n\|\mathbf{t}_n-W^T\phi(\mathbf{x}_n)\|_2^2\right) \\ &= -\frac{\beta}{2}\nabla\left(\sum_n(\mathbf{t}_n-W^T\phi(\mathbf{x}_n))^T(\mathbf{t}_n-W^T\phi(\mathbf{x}_n))\right) \\ &= -\frac{\beta}{2}\sum_n\left\{\nabla(\mathbf{t}_n^T\mathbf{t}_n)-2\nabla(\phi(\mathbf{x}_n)^TW\mathbf{t}_n)+\nabla(\phi(\mathbf{x}_n)^TWW^T\phi(\mathbf{x}_n))\right\} \\ &= -\frac{\beta}{2}\sum_n\left\{O-2\phi(\mathbf{x}_n)\mathbf{t}_n^T+\nabla(WIW^T\phi(\mathbf{x}_n)\phi(\mathbf{x}_n)^T)\right\} \\ &= -\frac{\beta}{2}\sum_n\left\{-2\phi(\mathbf{x}_n)\mathbf{t}_n^T+(\phi(\mathbf{x}_n)\phi(\mathbf{x}_n)^T)^TWI^T+(\phi(\mathbf{x}_n)\phi(\mathbf{x}_n)^T)WI\right\} \\ &= -\frac{\beta}{2}\sum_n\left\{-2\phi(\mathbf{x}_n)\mathbf{t}_n^T+2\phi(\mathbf{x}_n)\phi(\mathbf{x}_n)^TWI\right\} \\ &= -\beta\sum_n\left\{-\phi(\mathbf{x}_n)\mathbf{t}_n^T+\phi(\mathbf{x}_n)\phi(\mathbf{x}_n)^TW\right\} \\ &= -\beta\sum_n\phi(\mathbf{x_n})\left\{-\mathbf{t}_n^T+\phi(\mathbf{x}_n)^TW\right\} \\ &= -\beta\Phi^T(\Phi W-T)\end{aligned}$$
  - 最后一步的化简的思考过程是把对 n 求和视为两个分块矩阵的乘积：
    - 第一个矩阵是分块行向量，共 $1\times N$ 个块，且第n个分量是 $\phi(\mathbf{x}_n)$。因此第一个矩阵是 $(\phi(\mathbf{x}_1),\phi(\mathbf{x}_2),\cdots,\phi(\mathbf{x}_n))=\mathbf{\Phi^T}$
    - 第二个矩阵是分块列向量，共$N\times 1$个块，且第n个分量是 $-\mathbf{t}_n^T+\phi(\mathbf{x}_n)^TW$。因此，第二个矩阵是：

$$\begin{aligned}\begin{pmatrix}-\mathbf{t}_1^T+\phi(\mathbf{x}_1)^TW\\-\mathbf{t}_2^T+\phi(\mathbf{x}_2)^TW\\\vdots\\-\mathbf{t}_N^T+\phi(\mathbf{x}_n)^TW\end{pmatrix} &= \begin{pmatrix}\phi(\mathbf{x}_1)^TW\\\phi(\mathbf{x}_2)^TW\\\vdots\\\phi(\mathbf{x}_n)^TW\end{pmatrix}-\begin{pmatrix}\mathbf{t}_1^T\\\mathbf{t}_2^T\\\vdots\\\mathbf{t}_N^T\end{pmatrix}\\ &= \begin{pmatrix}\phi(\mathbf{x}_1)^T\\\phi(\mathbf{x}_2)^T\\\vdots\\\phi(\mathbf{x}_n)^T\end{pmatrix}W-\begin{pmatrix}\mathbf{t}_1^T\\\mathbf{t}_2^T\\\vdots\\\mathbf{t}_N^T\end{pmatrix}\\ &= \Phi W-T\end{aligned}$$

，注意第二个等号的推导过程中，前一项能够拆开是因为它被看做两个分块矩阵的乘积，两个分块矩阵分别由 $N\times 1$ 和 1×1 个块组成。

  - 这种方法虽然比较繁琐，但是更具有一般性。

RNN 的梯度消失/爆炸问题

- 通常 RNN 的状态方程的更新定义为 $\mathbf{h}_t=f(W\mathbf{h}_{t-1}+U\mathbf{x}_t+\mathbf{b})$（$f$ 表示一个逐元素的激活函数，例如 $\tanh(\cdot)$ 等。），而这里我们采用 Pascanu 等人的论文 On the difficulty of training Recurrent Neural Networks 中的定义，即认为 $\mathbf{h}_t=Wf(\mathbf{h}_{t-1})+U\mathbf{x}_t+\mathbf{b}$（这两种方程其实是等价的，只是前一种表述把隐层状态定义成激活后的值，后一种表述把隐层状态定义成激活前的值，前述论文中的脚注里也有说明。这里采用后一种方式，是因为它稍微好算一点）。展开后的网络结构示意图参见 CS224d-lecture8 中的 Slide 15。以下内容建议对照这份讲义的 15-19 页一起观看（另注：建议用 Stanford 的讲义梳理大致的思路，但是按照本讲稿下述步骤进行具体的求导运算。个人认为本讲稿中的过程更加清楚）。
- 现在我们来计算损失函数 l 对循环连接的权重矩阵 W 的导数：假设每一时间步都有一个误差 l_t（例如建立一个语言模型，每一步都要预测下一个词的概率分布，与语料库里的真实值计算交叉熵），总的误差等于每一步的误差加起来：$l=\sum_t l_t$，因此 $\dfrac{\partial l}{\partial W}=\sum_t\dfrac{\partial l_t}{\partial W}$（对一元函数来说，和的导数等于导数的和。根据多元函数偏导数的定义，很容易推广到多元函数上，进而推广到矩阵求导上）。
- 考虑到矩阵 W 出现了多次，计算 $\dfrac{\partial l_t}{\partial W}$ 需要计算 l_t 对 W 的每一次出现的导数，然后再求和。若用 $W^{(k)}$ 表示 $\mathbf{h}_{k-1}$ 与 $\mathbf{h}_k$ 之间的转移矩阵 W，则 $\dfrac{\partial l_t}{\partial W}=\sum_{k=1}^t\dfrac{\partial l_t}{\partial W^{(k)}}=\sum_{k=1}^t\dfrac{\partial l_t}{\partial\mathbf{h}_k}(f(\mathbf{h}_{k-1}))^T=\sum_{k=1}^t\left(f(\mathbf{h}_{k-1})\dfrac{\partial l_t}{\partial\mathbf{h}_k^T}\right)^T$。其中第二个等号用到的是线性变换的求导公式（类似标准 BP 算法的核心步骤）
- 然后根据雅克比矩阵的运算规则计算损失函数对隐层的导数（$\mathrm{diag}(\cdot)$ 表示将括号里的向量变成一个对角矩阵，跟前文的 $\mathrm{vec}(\cdot)$ 互为逆运算。）：$\dfrac{\partial l_t}{\partial\mathbf{h}_k^T}=\dfrac{\partial l_t}{\partial\mathbf{h}_t^T}\dfrac{\partial\mathbf{h}_t}{\partial\mathbf{h}_{t-1}}\cdots\dfrac{\partial\mathbf{h}_{k+1}}{\partial\mathbf{h}_k}=\dfrac{\partial l_t}{\partial\mathbf{h}_t^T}W\mathrm{diag}(f'(\mathbf{h}_{t-1}))\cdots W\mathrm{diag}(f'(\mathbf{h}_k))$，再将该式带入上一步中的式子，就得到：$\dfrac{\partial l_t}{\partial W}=\sum_{k=1}^t\mathrm{diag}(f'(\mathbf{h}_k))W^T\mathrm{diag}(f'(\mathbf{h}_{k+1}))W^T\cdots\mathrm{diag}(f'(\mathbf{h}_{t-1}))W^T\dfrac{\partial l_t}{\partial\mathbf{h}_t}(f(\mathbf{h}_{k-1}))^T$，这就是 vanilla RNN 的 BPTT 的公式。（中间很多个隐层之间的雅克比相乘那一部分可以用求积符号来书写，这里的写法更直观一些）
- 注：实践中具体计算梯度的时候，一般还是先定义一组类似于 BP 神经网络的 δ_t 的变量，使用循环逐层进行求导，而不是强行直接展开。这里展开是为了理论分析方便。
- 另注：Stanford 的讲义和前述论文中，均认为 $\dfrac{\partial\mathbf{h}_{i+1}}{\partial\mathbf{h}_i}=W^T\mathrm{diag}(f'(\mathbf{h}_i))$，这一点应该是错的，矩阵 W 不应该被转置，根据雅克比矩阵的定义写一个梯度检查的程序即可快速验证这一点。

Autoencoder with Tied-weight

- 求函数 $f(W,\mathbf{b}_1,\mathbf{b}_2)=l(\mathbf{b}_2+W^T\sigma(W\mathbf{x}+\mathbf{b}_1))$ 对 W 的导数，其中 $\sigma(\cdot)$ 是逐元素求 Sigmoid。
- 根据变量多次出现的求导法则计算即可：$\nabla_Wf=\nabla_Wl(\mathbf{b}_2+W^T\sigma(W_c\mathbf{x}+\mathbf{b}_1))+\nabla_Wl(\mathbf{b}_2+W_c^T\sigma(W\mathbf{x}+\mathbf{b}_1))$，其中 W_c 的含义是将 W 此次出现看做常数。
- 上式右边第一项计算如下：

$$\begin{aligned}\nabla_W\,l(\mathbf{b}_2+W^T\sigma(W_c\mathbf{x}+\mathbf{b}_1)) &= \left(\nabla_{W^T}l(\mathbf{b}_2+W^T\sigma(W_c\mathbf{x}+\mathbf{b}_1))\right)^T\\ &= \left(\nabla_{\mathbf{z}=\mathbf{b}_2+W^T\sigma(W_c\mathbf{x}+\mathbf{b}_1)}l(\mathbf{z})\ \nabla_{W^T}\mathbf{z}\right)^T\\ &= \left(\nabla_{\mathbf{z}}l(\mathbf{z})\ (\sigma(W_c\mathbf{x}+\mathbf{b}_1))^T\right)^T\\ &= \sigma(W_c\mathbf{x}+\mathbf{b}_1)(\nabla_{\mathbf{z}}l(\mathbf{z}))^T\end{aligned}$$

- 第二项计算如下：

$$
\begin{aligned}
\nabla_W\, l(\mathbf{b}_2 + W_c^T \sigma(W\mathbf{x} + \mathbf{b}_1)) &= \nabla_{\mathbf{u}=W\mathbf{x}+\mathbf{b}_1} l(\mathbf{b}_2 + W_c^T \sigma(\mathbf{u}))\ \mathbf{x}^T \\
&= \left(\nabla_{\mathbf{u}^T} l(\mathbf{b}_2 + W_c^T \sigma(\mathbf{u}))\right)^T\ \mathbf{x}^T \\
&= \left(\nabla_{\mathbf{t}^T} l(\mathbf{t})\ \frac{\partial \mathbf{t}}{\partial \mathbf{v}}\ \frac{\partial \mathbf{v}}{\partial \mathbf{u}}\right)^T \mathbf{x}^T \\
&= \left(\nabla_{\mathbf{t}^T} l(\mathbf{t}) W_c^T \mathrm{diag}(\sigma'(\mathbf{u}))\right)^T \mathbf{x}^T \\
&= \mathrm{diag}(\sigma'(\mathbf{u})) W_c \nabla_{\mathbf{t}} l(\mathbf{t}) \mathbf{x}^T
\end{aligned}
$$

  ，其中第三个等号里定义 $\mathbf{v} = \sigma(\mathbf{u}), \mathbf{t} = \mathbf{b}_2 + W_c^T \mathbf{v}$。
- 最终结果就是将以上两项合并起来，并去掉所有 W_c 中的下标，从略。