Differentials for Applied Computer Vision

Samuel Albanie

Abstract

The purpose of this short note is to collect together a set of useful calculus derivations using the framework of *matrix differentials*. The goal is to develop it over time and therefore it should be considered a work-in-progress. Consequently any contributions, feedback or notifications of mistakes are much appreciated[1]).

[1]Feel free to contact me by raising an issue on the github page, submitting a pull request directly (https://github.com/albanie/derivations)

Contents

Chapter 1

Introduction

Calculus plays a central role in computer vision. As the community has integrated ever more closely with techniques from machine learning, statistics and optimisation, the ability to differentiate vector and matrix functions has become more useful to computer vision researchers. Unfortunately, multivariate calculus, when performed with indices over element locations is a fairly tricky business to build an intuition for. Although the index-based approach has the majority of the market share in the research world, there is an alternative: the slightly lesser known method of *matrix differentials*. We have found this alternative approach to be significantly simpler, more intuitive and faster to work with.

Finally, we note that the widespread adoption of automatic differentiation (autodiff) has been a major boon to the whole community, in many cases obviating the need to perform arduous pen and paper derivations. The goal of these notes is not to encourage the reader to avoid using autodiff, but to assist them in understanding, on a mathematical level, what is going on "under the hood".

1.1 Resources

There are a number of extremely good references for getting to grips with the calculus of vector and matrix functions. Below are a few that we have found particularly useful, providing much of the material for these notes:

- Written by econometricians Magnus and Neudecker (originally in 1988, but revised several times), *Matrix differential calculus with applications in statistics and econometrics* is the canonical reference for matrix differential calculus (Magnus and Neudecker, 1988).
- The fundamentals of working with matrices (Searle and Khuri, 2017).
- (Kinghorn, 1996), (Minka, 2000) and (Petersen et al., 2008) each contain a large number of highly useful results and are good as quick references.
- The MatConvNet manual (Vedaldi, 2015) provides carefully worked through examples of matrix derivatives. While the emphasis is primarily on index-based derivations, it provides the derivatives of many common neural network computation blocks in matrix form and is very useful as a reference.

Chapter 2

Vector Functions

2.1 Element-wise operators

Often, we are tasked we computing the gradient of a function $\phi : \mathbb{R}^n \to \mathbb{R}^n$ that takes the form of an *element-wise operator*. By this, we mean that there exists some scalar function $f : \mathbb{R} \to \mathbb{R}$ such that for any $\mathbf{x} \in \mathbb{R}^n$

$$[\phi(\mathbf{x})]_i = f(\mathbf{x}_i), \quad \forall i \in \{1, \dots, n\} \tag{2.1}$$

For these operators, it is helpful to first compute the derivative of the scalar component function, f'. This can then be used in combination with differentials to compute the gradient of ϕ as follows:

$$\underbrace{\mathrm{d}\phi(\mathbf{x})}_{n\times 1} = \underbrace{\mathrm{diag}(f'[\mathbf{x}])}_{n\times n}\underbrace{\mathrm{d}\mathbf{x}}_{n\times 1} \tag{2.2}$$

where we have used the notation $f'[\mathbf{x}]$ to denote that f' is applied element-wise to each element of $\mathbf{x}$ (this notation is used throughout the document), and the dimensions of each term are displayed beneath the equation.

2.2 Sigmoid

The sigmoid function is often used as an elementwise operator in neural networks. The sigmoid is a scalar map $\sigma : \mathbb{R} \to \mathbb{R}$ defined by:

$$\sigma(x) = \frac{1}{1 + e^{-x}} = \frac{e^x}{1 + e^x} \tag{2.3}$$

We can compute its gradients using the quotient rule:

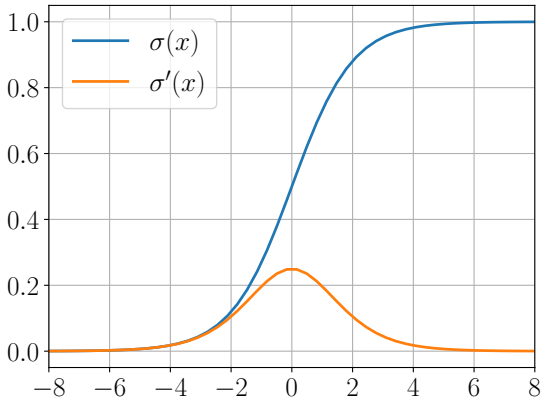

Figure 2.1: The sigmoid function and its derivative.

$$\begin{aligned}
\mathrm{d}(\sigma(x)) &= \mathrm{d}\left(\frac{e^x}{1+e^x}\right) \\
&= \frac{(1+e^x)\,\mathrm{d}(e^x) - e^x\,\mathrm{d}(1+e^x)}{(1+e^x)^2} = \frac{(1+e^x)e^x\,\mathrm{d}\,x - e^x e^x\,\mathrm{d}\,x}{(1+e^x)^2} \\
&= \left(\frac{e^x((1+e^x)-e^x)}{(1+e^x)^2}\right)\mathrm{d}\,x = \left(\frac{e^x}{(1+e^x)}\frac{1}{(1+e^x)}\right)\mathrm{d}x \\
&= \sigma(x)(1-\sigma(x))\mathrm{d}x
\end{aligned}$$

which tells us that $\sigma'(x) = \sigma(x)(1-\sigma(x))$. The sigmoid and its derivative are shown in Fig.2.1. We can now apply Eqn.2.2 to obtain the Jacobian of the element-wise sigmoid:

$$\mathrm{d}(\sigma[\mathbf{x}]) = \mathrm{diag}\left(\sigma[\mathbf{x}] \circ (\mathbf{1} - \sigma[\mathbf{x}])\right)\mathrm{d}\mathbf{x} \tag{2.4}$$

Since the diagonal matrix is symmetric, it follows that $\nabla\sigma(\mathbf{x}) = \mathrm{diag}\left(\sigma[\mathbf{x}] \circ (\mathbf{1} - \sigma[\mathbf{x}])\right)$.

2.3 Softmax

The softmax function represents another vector function that is commonly used as a computational block in neural networks. It can be viewed as a form of generalised sigmoid (for this reason it is common to overload the σ symbol for the softmax, $\sigma : \mathbb{R}^n \to \mathbb{R}^n$). It may be helpful to keep in mind that although it produces an output with the same shape as its input, it is not an element-wise operator and $\sigma(\mathbf{x}) \neq \sigma[\mathbf{x}]$ (i.e. it does not produce the same output as an element-wise sigmoid). It is defined by:

$$\sigma(\mathbf{x}) = \frac{\exp[\mathbf{x}]}{\mathbf{1}^T \exp[\mathbf{x}]} \tag{2.5}$$

Note that the denominator here is a scalar, while the numerator has shape $n \times 1$. To compute the gradient of this function, we can proceed with differentials:

$$\begin{aligned}
\mathrm{d}(\sigma(\mathbf{x})) &= \mathrm{d}\left(\frac{\exp[\mathbf{x}]}{\mathbf{1}^T \exp[\mathbf{x}]}\right) && (2.6)\\
&= \frac{(\mathbf{1}^T \exp[\mathbf{x}])\,\mathrm{d}(\exp[\mathbf{x}]) - \exp[\mathbf{x}]\,\mathrm{d}(\mathbf{1}^T \exp[\mathbf{x}])}{(\mathbf{1}^T \exp[\mathbf{x}])(\mathbf{1}^T \exp[\mathbf{x}])} && (2.7)\\
&= \frac{(\mathbf{1}^T \exp[\mathbf{x}])\,\mathrm{diag}(\exp[\mathbf{x}])\mathrm{d}\mathbf{x} - \exp[\mathbf{x}](\mathbf{1}^T \mathrm{diag}(\exp[\mathbf{x}])\mathrm{d}\mathbf{x})}{(\mathbf{1}^T \exp[\mathbf{x}])(\mathbf{1}^T \exp[\mathbf{x}])} \quad \text{(using 2.2)} && (2.8)\\
&= \left(\frac{\mathrm{diag}(\exp[\mathbf{x}])}{(\mathbf{1}^T \exp[\mathbf{x}])} - \frac{\exp[\mathbf{x}](\mathbf{1}^T \mathrm{diag}(\exp[\mathbf{x}]))}{(\mathbf{1}^T \exp[\mathbf{x}])(\mathbf{1}^T \exp[\mathbf{x}])}\right)\mathrm{d}\mathbf{x} && (2.9)\\
&= \left(\mathrm{diag}(\sigma(\mathbf{x})) - \sigma(\mathbf{x})\sigma(\mathbf{x})^T\right)\mathrm{d}\mathbf{x} \quad \text{(using A.9)} && (2.10)
\end{aligned}$$

Since both terms are symmetric matrices, it follows that $\nabla\sigma(\mathbf{x}) = \mathrm{diag}(\sigma(\mathbf{x})) - \sigma(\mathbf{x})\sigma(\mathbf{x})^T$.

There are also often cases where we would like to apply the softmax operator to a higher order tensor. The simplest way to address this notationally is define the softmax operator $\sigma(\cdot)$ to always act along the first non-singleton dimension. Thus for an $m \times n$ matrix $X = [\mathbf{x}_1|\ldots|\mathbf{x}_n]$ (with each $\mathbf{x}_i \in \mathbb{R}^m$), the action of the softmax operator produces the following effect:

$$\begin{aligned}
\sigma(X) &= \sigma\Big(\left[\begin{array}{c|c|c} \mathbf{x}_1 & \ldots & \mathbf{x}_n \end{array}\right]\Big)\\
&= \left[\begin{array}{c|c|c} \sigma(\mathbf{x}_1) & \ldots & \sigma(\mathbf{x}_n) \end{array}\right]
\end{aligned}$$

We can derive the gradient of this operator with respect to an input matrix $X \in \mathbb{R}^{m\times n}$, using the vec operator. This is a little more involved than the previous derivations, so the equations are interspersed with visualisations. The overall strategy is to reach an expression of the form $\mathrm{d}\,\mathrm{vec}(\sigma(X)) = A\,\mathrm{d}\,\mathrm{vec}(X)$ and apply the identification theorem (Magnus and Neudecker, 1988) to read off the Jacobian of the softmax as $A \in \mathbb{R}^{mn\times mn}$.

$$\mathrm{d}(\mathrm{vec}(\sigma(X))) = \mathrm{d}\left(\begin{bmatrix}\sigma(\mathbf{x}_1)\\ \vdots \\ \sigma(\mathbf{x}_n)\end{bmatrix}\right) = \left[\begin{array}{c|c|c} \mathrm{d}(\sigma(\mathbf{x}_1);\mathbf{x}_1) & \ldots & \mathrm{d}\sigma(\mathbf{x}_1;\mathbf{x}_n) \\ \hline \vdots & \ddots & \vdots \\ \hline \mathrm{d}(\sigma(\mathbf{x}_n);\mathbf{x}_1) & \ldots & \mathrm{d}(\sigma(\mathbf{x}_n);\mathbf{x}_n) \end{array}\right] \tag{2.11}$$

Here, we are writing the differential operator $\mathrm{d}(\cdot;\cdot)$ with an explicit dependence on its increment as the second argument (i.e. as defined in Eqn. B.5), rather than leaving this to be determined implicitly as was done previously. The off-diagonal terms disappear so we are left with the block diagonal matrix A where block ii contains $\mathrm{d}(\sigma(\mathbf{x}_i);\mathbf{x}_i)$ i.e. the vector differential of the softmax.

We therefore see that the $\sigma(\cdot)$ operator has several different meanings depending on the shape of its input (e.g. sigmoid, standard vector softmax, column-wise softmax for scalars, vectors and matrices respectively). Thus we can replace the last term in Eqn. 2.11 with the differentials found in Eqn. 2.10 to proceed:

$$= \left[\begin{array}{c|c|c} \mathrm{diag}(\sigma(\mathbf{x}_1)) - \sigma(\mathbf{x}_1)\sigma(\mathbf{x}_1)^T\,\mathrm{d}\,\mathbf{x}_1 & \ldots & 0_m \\ \hline \vdots & \ddots & \vdots \\ \hline 0_m & \ldots & \mathrm{diag}(\sigma(\mathbf{x}_n)) - \sigma(\mathbf{x}_n)\sigma(\mathbf{x}_n)^T\,\mathrm{d}\,\mathbf{x}_n \end{array}\right] \tag{2.12}$$

where 0_m denotes an $m \times m$ matrix of zeros. We can summarise this a little more concisely as follows:

$$\mathrm{d}(\mathrm{vec}(\sigma(X))) = \left(\mathrm{diag}(\sigma(X)) - (I_n \otimes \underbrace{\mathbf{1}\mathbf{1}^T}_{m \times m}) \circ (\sigma(X)\sigma(X)^T) \right) \mathrm{d}\,\mathrm{vec}(X) \tag{2.13}$$

We see that the resulting Jacobian closely mirrors that of the vector function derived in Eqn. 2.10, except that there is an additional block matrix of ones that prevents enforces that partial derivatives are zero between elements in the outputs of one column and elements in the inputs from different columns.

Chapter 3

Loss Functions

Loss functions are scalar functions that commonly require derivatives.

3.1 Cross Entropy

Cross entropy measures the distance between two distributions by computing the average number bits required to encode symbols drawn from one distribution using the optimal coding scheme of the other. Given discrete distributions, p and q, we define the cross entropy between them as:

$$H[p,q] = -\sum_{\mathbf{x}} p(\mathbf{x}) \log q(\mathbf{x}) = -\mathbb{E}_{\mathbf{x}\sim p} \log q(\mathbf{x}) \tag{3.1}$$

$$= \underbrace{-\sum_{\mathbf{x}} p(\mathbf{x}) \log p(\mathbf{x})}_{H[p]} + \underbrace{\sum_{\mathbf{x}} p(\mathbf{x}) \log \frac{p(\mathbf{x})}{q(\mathbf{x})}}_{\mathrm{D}_{\mathrm{KL}}(p||q)} \tag{3.2}$$

In the machine learning context, we typically cannot evaluate Eqn.3.1 across the full distribution $p(\mathbf{x})$. Instead, we often have some number (e.g. m) of samples, m, and the respective probability of each sample under p (the ground truth) and q (the predictions we are trying to match against the ground truth). Stacking the probabilities into vectors $\hat{\mathbf{p}}, \hat{\mathbf{q}} \in \mathbb{R}^m$, we can rewrite Eqn.3.1 as the loss corresponding to m samples as $\hat{\mathbf{p}}^T \log[\hat{\mathbf{q}}]$ (here we have made use of the element-wise $\log[\cdot]$ operator notation). As before, we compute differentials:

$$\mathrm{d}\, H = \mathrm{d}(\hat{\mathbf{p}}^T \log[\hat{\mathbf{q}}]) = \underbrace{\hat{\mathbf{p}}^T}_{1\times n} \underbrace{\mathrm{diag}(\hat{\mathbf{q}})^{-1}}_{n\times n} \underbrace{\mathrm{d}\,\hat{\mathbf{q}}}_{n\times 1}$$

(where we have used Eqn.2.2 for the last equality), so the gradient with respect to the predicted distribution across samples is $\nabla_{\hat{\mathbf{q}}} H = \mathrm{diag}(\hat{\mathbf{q}})^{-1}\hat{\mathbf{p}}$.

3.2 Binary Logistic Regression

Logistic regression is a common tool for learning a binary classifier (the problem setup described here is based on (de Freitas)). The loss is often optimised with IRLS (Iterative Reweighted Least Squares), which requires not only the gradient, but also the hessian of the loss w.r.t the parameters. We will perform the derivations with differentials. Suppose that we are given n training pairs

$\{\mathbf{x}_i, y_i\}_{i=1}^n$ consisting of observations $\mathbf{x}_i \in \mathbb{R}^d$ and labels $y_i \in \mathbb{R}$, and would like to find the parameters $\boldsymbol{\theta} \in \mathbb{R}^d$ that maximise the likelihood of the labels conditioned on the observations under a Bernoulli model:

$$p(\mathbf{y}|X, \theta) = \prod_{i=1}^{n} \mathrm{Ber}(y_i|\sigma(\mathbf{x}_i^T\boldsymbol{\theta}))$$

where we have used X to denote the $d \times n$ matrix formed by stacking the observations as columns and $\mathbf{y} \in \mathbb{R}^n$ as a column vector formed from the labels. Rather than maximising likelihood, we find $\boldsymbol{\theta}$ by minimising the negative log-likelihood:

$$\mathcal{L}(\boldsymbol{\theta}) = -\sum_{i=1}^{n} y_i \log \sigma(\mathbf{x}_i^T\boldsymbol{\theta}) + (1 - y_i) \log(1 - \sigma(\mathbf{x}_i^T\boldsymbol{\theta}))$$

We can rewrite this in a more convenient form, again using $[\cdot]$ element-wise operator notation:

$$\mathcal{L}(\boldsymbol{\theta}) = -\mathbf{y}^T \log[\sigma[X^T\boldsymbol{\theta}]] - (\mathbf{1} - \mathbf{y})^T \log[\mathbf{1} - \sigma[X^T\boldsymbol{\theta}]]$$

To find the gradient, let's write $\boldsymbol{\pi} = \sigma[X^T\boldsymbol{\theta}]$, noting that $\mathrm{d}\,\boldsymbol{\pi} = \mathrm{diag}(\boldsymbol{\pi} \circ (\mathbf{1} - \boldsymbol{\pi}))X^T\,\mathrm{d}\,\boldsymbol{\theta}$, and take differentials:

$$\begin{aligned}
\mathrm{d}\,\mathcal{L} &= -\mathbf{y}^T\,\mathrm{d}(\log[\boldsymbol{\pi}]) - (\mathbf{1} - \mathbf{y})^T\,\mathrm{d}(\log[\mathbf{1} - \boldsymbol{\pi}]) && (3.3)\\
&= -\mathbf{y}^T\,\mathrm{diag}(\boldsymbol{\pi})^{-1}\,\mathrm{diag}(\boldsymbol{\pi} \circ (\mathbf{1} - \boldsymbol{\pi}))X^T\,\mathrm{d}\,\boldsymbol{\theta} \\
&\quad - (\mathbf{1} - \mathbf{y})^T\,\mathrm{diag}(\mathbf{1} - \boldsymbol{\pi})^{-1}\,\mathrm{diag}(-\boldsymbol{\pi} \circ (\mathbf{1} - \boldsymbol{\pi}))X^T\,\mathrm{d}\,\boldsymbol{\theta} && \text{(using 2.2)} \quad (3.4)\\
&= -\mathbf{y}^T\,\mathrm{diag}(\mathbf{1} - \boldsymbol{\pi})X^T\,\mathrm{d}\,\boldsymbol{\theta} - (\mathbf{1} - \mathbf{y})^T\,\mathrm{diag}(-\boldsymbol{\pi})X^T\,\mathrm{d}\,\boldsymbol{\theta} && \text{(using A.7)} \quad (3.5)\\
&= \Big(-\mathbf{y}^T\,\mathrm{diag}(\mathbf{1} - \boldsymbol{\pi}) + (\mathbf{1} - \mathbf{y})^T\,\mathrm{diag}(\boldsymbol{\pi})\Big)X^T\,\mathrm{d}\,\boldsymbol{\theta} && (3.6)\\
&= \Big(-\mathbf{y}^T(I_n - \mathrm{diag}(\boldsymbol{\pi})) + (\mathbf{1} - \mathbf{y})^T\,\mathrm{diag}(\boldsymbol{\pi})\Big)X^T\,\mathrm{d}\,\boldsymbol{\theta} && \text{(using A.9)} \quad (3.7)\\
&= \Big(-\mathbf{y}^T + (\mathbf{y} \circ \boldsymbol{\pi})^T + \boldsymbol{\pi}^T - (\mathbf{y} \circ \boldsymbol{\pi})^T\Big)X^T\,\mathrm{d}\,\boldsymbol{\theta} && (3.8)\\
&= (\boldsymbol{\pi}^T - \mathbf{y}^T)X^T\,\mathrm{d}\,\boldsymbol{\theta} && (3.9)\\
& && (3.10)
\end{aligned}$$

thus we see that $\nabla\mathcal{L}(\boldsymbol{\theta}) = X(\boldsymbol{\pi} - \mathbf{y}) \in \mathbb{R}^n$ (note that to be consistent with the derivations in this document, this differs by a transpose from the derivation in (de Freitas)). Next, we derive the Hessian of the loss by taking the second differential:

$$\begin{aligned}
\mathrm{d}^2\,\mathcal{L} = \mathrm{d}(\mathrm{d}\,\mathcal{L}) &= \mathrm{d}((\boldsymbol{\pi}^T - \mathbf{y}^T)X^T\,\mathrm{d}\,\boldsymbol{\theta}) && (3.11)\\
&= (\mathrm{d}\,\boldsymbol{\pi})^T X^T\,\mathrm{d}\,\boldsymbol{\theta} && (3.12)\\
&= (\mathrm{diag}(\boldsymbol{\pi} \circ (\mathbf{1} - \boldsymbol{\pi}))X^T\,\mathrm{d}\,\boldsymbol{\theta})^T X^T\,\mathrm{d}\,\boldsymbol{\theta} && (3.13)\\
&= \mathrm{d}\,\boldsymbol{\theta}^T X\,\mathrm{diag}(\boldsymbol{\pi} \circ (\mathbf{1} - \boldsymbol{\pi}))X^T\,\mathrm{d}\,\boldsymbol{\theta} && (3.14)\\
& && (3.15)
\end{aligned}$$

Since this matrix is symmetric, we can use the second identification theorem (Magnus and Neudecker, 1988) to determine that the Hessian is given by $\nabla^2\mathcal{L}(\boldsymbol{\theta}) = X\,\mathrm{diag}(\boldsymbol{\pi} \circ (\mathbf{1} - \boldsymbol{\pi}))X^T$

3.3 Multinomial Logistic Regression

Logistic regression can be extended from binary to k-way classification by replacing the sigmoid function with a softmax. Suppose that we have a collection of training pairs $\{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^n$, with each $\mathbf{x}_i \in \mathbb{R}^d$ and $\mathbf{y}_i$ a k-dimensional one-hot vector (i.e. $\mathbf{y}_i \in \{0,1\}^k$ and $\mathbf{1}^T\mathbf{y}_i = 1$). Then our objective is to optimise a collection of weights $\Theta \in \mathbb{R}^{d\times k}$ to maximise the conditional likelihood of the labels under a multinomial model. Writing Y for the matrix of labels $[\mathbf{y}_1|\ldots|\mathbf{y}_n]$, we would like to maximise:

$$P(Y|X,\Theta) = \prod_{i=1}^{n} \text{Mult}(\mathbf{y}_i|\sigma(\Theta^T\mathbf{x}_i))$$

Here, the notation indicates that the softmax is operating column-wise (as discussed in Sec.2.3). As previously, we will optimise Θ by minimising the negative log-likelihood:

$$\begin{aligned}\mathcal{L}(\Theta) &= -\sum_{i=1}^{n} \underbrace{\mathbf{y}_i^T}_{1\times k} \underbrace{\log[\sigma(\Theta^T\mathbf{x}_i)]}_{k\times 1} \\ &= \underbrace{\mathbf{1}^T}_{1\times n} \underbrace{Y^T}_{n\times k} \underbrace{\log[\sigma(\Theta^T X)]}_{k\times n} \underbrace{\mathbf{1}}_{n\times 1}\end{aligned}$$

where the dimension of each matrix has been listed beneath each equation (recall that $\log[\cdot]$ is an element-wise operator). To keep the notation concise, we will use Π to denote the $k \times n$ matrix $\sigma[\Theta^T X]$, noting that from Eqn. 2.13 that $\mathrm{d}\,\text{vec}(\Pi) = \text{diag}(\Pi) - (I_n \otimes \mathbf{1}\mathbf{1}^T) \circ (\Pi\Pi^T)X^T\,\mathrm{d}\,\text{vec}(\Theta)$.

[S: Need to rewrite this in terms of traces]

$$\begin{aligned}\mathrm{d}\,\mathcal{L}(\Theta) &= \mathrm{d}(\mathbf{1}^T Y^T \log[\Pi]\mathbf{1}) && (3.16)\\ &= \mathbf{1}^T Y^T\,\mathrm{d}(\log[\Pi])\mathbf{1} && (3.17)\\ &= \mathbf{1}^T Y^T\,\text{diag}(\Pi)^{-1}\,\mathrm{d}(\Pi)\mathbf{1} && (3.18)\\ &= \mathbf{1}^T Y^T\,\text{diag}(\Pi)^{-1}(\text{diag}(\Pi) - (I_n \otimes \mathbf{1}\mathbf{1}^T) \circ (\Pi\Pi^T)X^T)\,\mathrm{d}\,\Theta\mathbf{1} && (3.19)\\ &= \mathbf{1}^T Y^T (I_{kn} - \text{diag}(\Pi)^{-1}(I_n \otimes \mathbf{1}\mathbf{1}^T) \circ (\Pi\Pi^T)X^T)\,\mathrm{d}\,\Theta\mathbf{1} && (3.20)\end{aligned}$$

Appendix A

Matrix Manipulation

A.1 Operators

A.1.1 The vec operator

When working with differentials, much of the complexity of matrix manipulation (and dealing with higher order tensors in general) can be resolved using the vec$(\cdot)$ operator, which simply lays out the elements of a tensor in column major order. In this manner, computations can be reduced to operations on vectors (Chap. 2).

For a concrete example, vec$(\cdot)$ produces the following result when applied to a 2×3 matrix X:

$$\text{vec}\left(\begin{bmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \end{bmatrix}\right) = \begin{bmatrix} x_{11} \\ x_{21} \\ x_{12} \\ x_{22} \\ x_{13} \\ x_{23} \end{bmatrix} \tag{A.1}$$

While we express a preference for dealing with tensors using the vec operator, we note that is also perfectly valid to determine derivatives using tensor Jacobians (see e.g. Johnson for a description of how this can be done in the context of neural networks).

A.1.2 Hadamard product

The Hadamard product of two matrices $A, B \in \mathbb{R}^{m \times n}$ is simply the result of element-wise multiplication between them:

$$(A \circ B)_{ij} = (A)_{ij}(B)_{ij}$$

A.1.3 Kronecker product

The Kronecker product of two matrices $A \in \mathbb{R}^{m \times n}$ and $B \in \mathbb{R}^{p \times q}$ is defined to be the $mp \times nq$ block matrix:

$$A \otimes B = (a_{ij}B) \tag{A.2}$$

A.1.4 Commutation matrices

Commutation matrices are permutation matrices that re-order vectorised elements. Given $A \in \mathbb{R}^{m\times n}$, then K_{mn} is the $mn \times mn$ orthogonal matrix such that

$$K_{mn}\,\mathrm{vec}(A) = \mathrm{vec}(A^T)$$

The usefulness of this operator (and the source of its name), is that it allows the two matrices in a Kronecker product to be exchanged (Eqn. A.6).

A.1.5 Khatri-Rao product

Given structured matrices E, F where the submatrix E_{ij} has order $m_i \times n_j$ and submatrix F_{ij} has order $p_i \times q_j$, then the **Khatri-Rao** product is defined as the Kronecker product of the submatrices:

$$E * F = (E_{ij} \otimes F_{ij})_{ij}$$

The resulting matrix has order $(\sum_i m_i p_i) \times (\sum_j n_j q_j)$. For a concrete example, suppose

$$E = \left[\begin{array}{c|c} E_{11} & E_{12} \\ \hline E_{21} & E_{22} \end{array}\right], \quad F = \left[\begin{array}{c|c} F_{11} & F_{12} \\ \hline F_{21} & F_{22} \end{array}\right] \tag{A.3}$$

then

$$E * F = \left[\begin{array}{c|c} E_{11} \otimes F_{11} & F_{12} \otimes F_{12} \\ \hline E_{21} \otimes F_{21} & F_{22} \otimes F_{22} \end{array}\right] \tag{A.4}$$

This can be further extended to the Tracy-Singh product (Tracy and Singh, 1972), which generates a new structured matrix by taking the Kronecker product between all possible pairwise blocks combinations.

A.2 The diag operator

The $\mathrm{diag} : \mathbb{R}^d \to \mathbb{R}^{d\times d}$ operator arranges the elements of a vector along the diagonal of a square matrix. It also has a useful pseudo-inverse (Minka, 2000), $\mathrm{diag}^{-1} : \mathbb{R}^{d\times d} \to \mathbb{R}^d$, which takes the elements from the diagonal of a matrix and stacks them into a vector. Note that this is only a pseudo-inverse, since for any vector $\mathbf{x} \in \mathbb{R}^d$, $\mathbf{x} = \mathrm{diag}^{-1}(\mathrm{diag}(\mathbf{x}))$, but for a square matrix $A \in \mathbb{R}^{d\times d}$, we have $A = \mathrm{diag}^{-1}(\mathrm{diag}(A))$ if and only if A is a diagonal matrix.

A.3 Identities

Throughout this section, we will assume that $A \in \mathbb{R}^{m\times n}$, $B \in \mathbb{R}^{p\times q}$ and $C \in \mathbb{R}^{r\times s}$. The following identity comes up frequently when working with matrix differentials:

$$\mathrm{vec}(ABC) = (C^T \otimes A)\,\mathrm{vec}(B) \tag{A.5}$$

As mentioned in Sec. A.1.4, the usefulness of the commutation matrix K_{mn} lies in the following result (Magnus and Neudecker, 1988):

$$K_{mp}(A \otimes B) = (B \otimes A)K_{nq} \tag{A.6}$$

We can prove this with the help of Eqn. A.5. For any matrix $X \in \mathbb{R}^{q \times n}$, we have that

$$\begin{aligned} K_{mp}(A \otimes B)\,\mathrm{vec}(X) &= K_{mp}\,\mathrm{vec}(BXA^T) = \mathrm{vec}(AX^TB^T) \\ &= (B \otimes A)\,\mathrm{vec}(X^T) = (B \otimes A)K_{nq}\,\mathrm{vec}(X) \end{aligned}$$

Since this holds for any X, Eqn. A.6 follows.

There are several useful identities for working with Hadamard products, the $\mathrm{diag}(\cdot)$ and $\mathrm{diag}^{-1}(\cdot)$ (these are from Minka (2000)). Suppose $E, F \in \mathbb{R}^{m \times n}, \alpha, \beta \in \mathbb{R}$:

$$\mathrm{diag}(E \circ F) = \mathrm{diag}(E) \circ \mathrm{diag}(F) \tag{A.7}$$

$$\mathrm{diag}^{-1}(E \circ F) = \mathrm{diag}^{-1}(E) \circ \mathrm{diag}^{-1}(F) \tag{A.8}$$

$$\mathrm{diag}(\alpha E + \beta F) = \alpha\,\mathrm{diag}(E) + \beta\,\mathrm{diag}(F) \tag{A.9}$$

$$\mathrm{diag}^{-1}(\alpha E + \beta F) = \alpha\,\mathrm{diag}^{-1}(E) + \beta\,\mathrm{diag}^{-1}(F) \tag{A.10}$$

The trace operator can often get you out of a tight spot. Suppose $X \in \mathbb{R}^{m \times n}$, $Y \in \mathbb{R}^{n \times o}$ and $Z \in \mathbb{R}^{o \times p}$. Then we can cycle the arguments:

$$\mathrm{tr}(XYZ) = \mathrm{tr}(ZXY) = \mathrm{tr}(YZX)$$

We can sum the result of a Hadamard product

$$\mathrm{tr}(E^TF) = \mathrm{vec}(E)^T\,\mathrm{vec}(F) = \mathbf{1}^T\,\mathrm{vec}(E \circ F)$$

More generally, we have the following relation (Minka, 2000):

$$\mathrm{tr}(E^TF) = \mathrm{tr}((E^*)^TF^*) \tag{A.11}$$

for any operator $(\cdot)^*$ that rearranges the elements of a matrix.

Appendix B

Differentials

The core idea behind differentials is based on examining the Taylor expansion of a function about a certain location. For a rigorous treatment of the topic, see Magnus and Neudecker (1988). In this section, we give a brief overview.

B.1 Scalar functions

Recall that for a scalar function $f : \mathbb{R} \to \mathbb{R}$, we can define the derivative at a point p by the following limit, subject to the condition that this limit exists:

$$f'(p) = \lim_{h \to 0} \frac{f(p+h) - f(p)}{h}$$

However, we note that by using Taylor's theorem, we can also pursue an equivalent formulation:

$$f(p+h) = f(p) + \underbrace{hf'(p)}_{\text{linear in } h} + r_p(h) \tag{B.1}$$

which comprises an affine map $f(p) + hf'(p)$ (i.e. an offset and a linear component) and a final term $r_p(h)$ that is dominated by h:

$$\lim_{h \to 0} \frac{r_p(h)}{h} = 0 \tag{B.2}$$

We define the *differential* of f at p with increment h to be the component in Eqn. B.1 that is linear in h:

$$\mathrm{d}f(p; h) = hf'(p) \tag{B.3}$$

B.2 Vector functions

The same logic can be applied directly to vector functions (Magnus and Neudecker, 1988). Given a vector function $\phi : \mathbb{R}^n \to \mathbb{R}^m$ and that there exists some matrix A such that at a point $\mathbf{p} \in \mathbb{R}^n$

$$\phi(\mathbf{p}+\mathbf{h}) = \phi(\mathbf{p}) + \underbrace{A(\mathbf{p})}_{m\times n}\mathbf{h} + r_{\mathbf{p}}(\mathbf{h}) \tag{B.4}$$

where $r_{\mathbf{p}}(\mathbf{h})$ satisfies the condition

$$\lim_{\mathbf{h}\to 0} \frac{r_p(\mathbf{h})}{||\mathbf{h}||} = 0$$

then we state that the function is differentiable at p with first derivative $A(\mathbf{p})$ and define

$$\mathrm{d}\,\phi(p;\mathbf{h}) = A(\mathbf{p})\mathbf{h} \tag{B.5}$$

to be the first differential of ϕ at $\mathbf{p}$ with increment $\mathbf{h}$.

Bibliography

Nando de Freitas. Lecture notes on logistic regression: A simple ann. https://www.cs.ox.ac.uk/people/nando.defreitas/machinelearning/lecture6.pdf.

Justin Johnson. Derivatives, backpropagation, and vectorization. http://cs231n.stanford.edu/handouts/derivatives.pdf.

Donald B Kinghorn. Integrals and derivatives for correlated gaussian functions using matrix differential calculus. *International journal of quantum chemistry*, 57(2):141–155, 1996.

Jan R Magnus and Heinz Neudecker. Matrix differential calculus with applications in statistics and econometrics. *Wiley series in probability and mathematical statistics*, 1988.

Thomas P Minka. Old and new matrix algebra useful for statistics. 2000. http://www.iro.umontreal.ca/~pift6266/A06/refs/minka-matrix.pdf.

Kaare Brandt Petersen, Michael Syskind Pedersen, et al. The matrix cookbook. *Technical University of Denmark*, 7(15):510, 2008.

Shayle R Searle and Andre I Khuri. *Matrix algebra useful for statistics*. John Wiley & Sons, 2017.

Derrick S Tracy and Rana P Singh. A new matrix product and its applications in partitioned matrix differentiation. *Statistica Neerlandica*, 26(4):143–157, 1972.

Lenc Karel Gupta Ankush Vedaldi, Andrea. Matconvnet reference manual. ACM, 2015. http://www.vlfeat.org/matconvnet/matconvnet-manual.pdf.