Matrix Differential Calculus with Tensors (for *Machine Learning*)

Massimiliano Tomassoli
(reverse(*5102mnhuik*)@gmail.com)

08/21/2016
(*alpha* version)

Contents

1 Introduction

The goal of this tutorial is to give the reader a deep understanding of *Matrix Differential Calculus* (MDC) and simplify previous methods by using *tensors* (i.e. *multidimensional arrays*).

MDC is a specialization of regular calculus for easily differentiating functions which involve matrices and vectors. With MDC, we can compute the gradient of, say, $f(x) = x^T Ax$ directly without having to deal with the single elements of x and A.

The classic approach to MDC is that by Magnus and Neudecker's[1]. For a quick overview you can read a concise introduction I wrote a few years ago[3].

Recently I came across a paper by Pollock[2] which claims that MDC is still not well understood by all practitioners and proposes to use a notation (basically, *Einstein notation*[5]) which reveals the tensor structure of the matrices.

Both approaches rely on the concept of *vectorization*. Basically, we convert matrices into vectors so that we can keep using the usual rules for vectors. While this is undoubtedly a smart move, I think it makes things more complicated than necessary, at least on a conceptual level.

My idea is to get rid of vectorization completely by using full-fledged tensors. One shouldn't need to remember properties about vectorization and the *Kronecker* operator, or tricks involving the *trace* operator.

I propose a simple alternative to index notation which I call *bracket notation*, and which, I believe, is more convenient for our goal. Note that I'm not claiming that my approach reduces computations, but just that it makes them more regular and conceptually simpler.

Since I rely exclusively on the community for suggestions and corrections, I warmly welcome any kind of constructive feedback.

Part I
Theory

2 Tensors

2.1 Informal introduction

Think of *tensors* as *multidimensional arrays*. In particular, a *scalar* is a 0-dimensional tensor, a *vector* is a 1-dimensional tensor and a *matrix* is a 2-dimensional tensor. What about *row vectors* and *column vectors*? Some authors use an *index-notation* called *Einstein notation*[5] where x_i is a row vector and x^i a column vector, or vice versa[2], but we'll introduce a different notation which doesn't make this distinction because there's no need to.

Let A be a 3-dimensional array where A_{ijk} is the element $A[i, j, k]$, $A[i][j][k]$ or similar depending on the programming language used. We'll use e_{ijk} to indicate a 3-dimensional array whose elements are all 0 except for a single 1 in position i, j, k. These e_{ijk} tensors form the canonical base for 3-dimensional tensors.

If arrays behave analogously to vectors and matrices when multiplied by a scalar, then

$$A = \sum_{i,j,k} A_{ijk} e_{ijk}.$$

This is analogous to rewriting a vector $v = (v_1, \ldots, v_n)$ as

$$v = \sum_{i=1}^{n} v_i e_i.$$

Now let's say we have two 2-dimensional arrays, i.e. matrices, A and B:

$$A = \sum_{i,j} A_{ij} e_{ij}$$
$$B = \sum_{l,m} B_{lm} e_{lm}.$$

The product $C = AB$ is a matrix whose generic element in position $[i, m]$ is equal to $\sum_k A_{ik} B_{km}$ as dictated by the *row-column multiplication rule*.

The product is thus computed as follows:

for all i:
    for all m:
        $C[i, m] = \sum_k A_{ik} B_{km}$

A more general way to see this is the following:

1. We form
$$M^1 = A \otimes B = \sum_{i,j,l,m} A_{ij} B_{lm} e_{ijlm},$$
which is a 4-dimensional array whose generic element in position i, j, l, m is $A_{ij} B_{lm}$.

2. We fuse the second and third dimensions, corresponding to the indices j and l, by performing an *element-wise* multiplication by walking along the second and third dimensions in parallel:
$$M^2 = \sum_{i,j,m} A_{ij} B_{jm} e_{ijm}.$$
Note that M^2 is a 3-dimensional tensor.

3. Finally, we sum along the second dimension corresponding to the index j:
$$M^3 = M^2.\text{sum}(\text{axis} = (2-1)) = \sum_{i,m} \left(\sum_j A_{ij} B_{jm} \right) e_{im},$$
where we included *numpy* notation for the same operation (remember that axis is 0-based, which explains the "-1"). Note that M^3 is exactly $C = AB$.

If you understand the $e_{ij\ldots}$ notation then you should have no problems understanding the 3 steps above.

Let's simplify the notation a little. We'll rewrite the expressions above as

$$\begin{aligned} M^1 &= (A_{ij} B_{lm} | ijlm) \\ M^2 &= (A_{ij} B_{jm} | ijm) \\ M^3 &= (A_{ij} B_{jm} | im)\,. \end{aligned} \tag{1}$$

Note that *repeated indices* which don't appear on the right of "|" are summed over. For instance, j is repeated in equation 1 and doesn't appear on the right of "|" so equation 1 is equivalent to

$$M^3 = \left(\sum_j A_{ij} B_{jm} \middle| im \right).$$

Note also that the order of the dimensions is indicated by the part on the right of "|" alone. For instance,

$$(A_{ij} B_{jm} | im) = (B_{jm} A_{ij} | im)\,.$$

2.2 Formal definitions

Let's define the 3 operations introduced above more formally.

We'll be using the following two tensors:

$$A = (A_{i_1 i_2 \cdots i_N} \mid i_1 i_2 \cdots i_N)$$
$$B = (B_{j_1 j_2 \cdots j_N} \mid j_1 j_2 \cdots j_N).$$

We define the *Kronecker product* as

$$A \otimes B = (A_{i_1 \cdots i_N} B_{j_1 \cdots j_M} \mid i_1 \cdots i_N j_1 \cdots j_M)$$

Note that, in general, $A \otimes B \neq B \otimes A$.

As an example, consider the *outer product* between vectors:

$$vw^T = (v_i w_j \mid ij) = v \otimes w.$$

The *element-wise product* is hard to write in general form, so let's consider an example:

$$A \,{}^{23}\!\odot^{14} B = (A_{i_1 k h i_4 \cdots i_N} B_{k j_2 j_3 h j_5 \cdots j_M} \mid i_1 k h i_4 \cdots i_N j_2 j_3 j_5 \cdots j_M). \tag{2}$$

In words, the second dimension of A is multiplied element-wise by the first dimension of B and the third dimension of A is multiplied element-wise by the fourth dimension of B. Note that the positions of the multiplied dimensions in the result are indicated by the numbers on the left of the operator "$\odot$", i.e. 23 in this example.

Let's give a general definition. Let $I = i_1 \cdots i_N$ and $J = j_1 \cdots j_M$ be lists of indices. In general, if $L = l_1 \cdots l_S$ is a list and $P = p_1 \cdots p_K$ a list of positions in $\{1, \ldots, S\}$ *without repetitions* (i.e. $p_i = p_j \iff i = j$), then $L[P]$ is the sublist selected from L by the positions in P, i.e.,

$$L[P] = l_{p_1} l_{p_2} \cdots l_{p_K}.$$

From now on, we'll always assume that lists of positions are* without repetitions*. This simplifies both definitions and results.

$L[P] \leftarrow k_1 \cdots k_K$ is the list L after the elements at positions $p_1, \ldots, p_K$ have been replaced with the elements $k_1, \ldots, k_K$, respectively. $Q \setminus R$ is the list Q without the elements in R, where the remaining elements have the same relative order as in Q. If P is a list of positions in $\{1, \ldots, S\}$, then $-P = (1 \cdots S) \setminus P$. Therefore, $L[-P]$ is the list L after the elements at positions $p_1, \ldots, p_K$ have been deleted.

With these definitions, example 2 can be rewritten as

$$A \,{}^{23}\!\odot^{14} B = (A_{I[23] \leftarrow kh} B_{J[14] \leftarrow kh} \mid (I[23] \leftarrow kh) J[-(14)]).$$

In general, we have

$$A \,{}^{P}\!\odot^{Q} B = (A_{I[P] \leftarrow K} B_{J[Q] \leftarrow K} \mid (I[P] \leftarrow K) J[-Q]), \tag{3}$$

where $\text{length}(K) = \text{length}(P) = \text{length}(Q)$.

A ***very important*** point to understand is that, in general, $A\,{}^{P}\!\odot^{Q} B$ and $B\,{}^{Q}\!\odot^{P} A$ are different but they only differ by a permutation of their dimensions. For instance, if $A = (A_{ijk} \mid ijk)$ and $B = (B_{lmn} \mid lmn)$, then:

$$A\,{}^{12}\!\odot^{32} B = (A_{ijk} B_{lmn} \mid ijklmn) \rightsquigarrow (A_{ijk} B_{lji} \mid ijkl) \tag{4}$$

$$B\,{}^{32}\!\odot^{12} A = (B_{lmn} A_{ijk} \mid lmnijk) \rightsquigarrow (B_{lji} A_{ijk} \mid ljik), \tag{5}$$

where $A_1 \rightsquigarrow \cdots \rightsquigarrow A_n = A_n$, which means that using "$\rightsquigarrow$" is just a trick to show intermediate steps without losing rigor.

If W is an n-dimensional tensor and we want to square each one of its elements, we can compute $W\,{}^{1\cdots n}\!\odot^{1\cdots n} W$, but this is needlessly verbose, so, as a convention, let's omit the positions when they go from 1 to the last one:

$$W\,{}^{1\cdots n}\!\odot^{1\cdots n} W = W \odot W.$$

Note that "$\odot$" is the usual *Hadamard product operator*[6] generalized to tensors.

We can extend this definition by deciding that if any indices are *marked* with a *bar*, then the corresponding dimensions will be summed over:

$$A\,{}^{\overline{P}Q}\!\odot^{RS} B = (A_{I[PQ]\leftarrow K} B_{J[RS]\leftarrow K} \mid (I[PQ] \leftarrow K)[-P]J[-(RS)]),$$

where $P \cap Q = R \cap S = \emptyset$ and $\text{length}(K) = \text{length}(PQ) = \text{length}(RS)$. (Note that the point made before about the non commutativity of "$\odot$" still applies.)

This lets us express the usual matrix product as

$$\begin{aligned} AB &= \left(\sum_k A_{ik} B_{km} \middle| im \right) \\ &= (A_{ik} B_{km} \mid im) \\ &= A\,{}^{\overline{2}}\!\odot^{1} B. \end{aligned}$$

The *inner product* between vectors can be expressed simply as

$$v^T w = v\,{}^{\overline{1}}\!\odot^{1} w.$$

If all positions are marked, we can use the *dot operator* as a shortcut:

$$A\,{}^{P}\!\cdot^{Q} B = A\,{}^{\overline{P}}\!\odot^{Q} B.$$

Note that the dot operator generalizes the usual *dot product* (or *inner product*). Now we can rewrite the product between matrices as

$$AB = A\,{}^{\overline{2}}\!\odot^{1} B = A\,{}^{2}\!\cdot^{1} B,$$

and the inner product between vectors as

$$v^T w = v \cdot w,$$

where we used the same convention as before of omitting the positions when they go from 1 to the last one (here there's just one position).

We may define another handy shortcut for "$\cdot$" and "$\odot$". If X is an n-dimensional tensor and Y an m-dimensional tensor, then

$$X^{(n-k+1)\cdots n}\odot^{1\cdots k}Y = X \odot_k Y$$
$$X^{(n-k+1)\cdots n}\cdot^{1\cdots k}Y = X \cdot_k Y$$

Now we can write

$$AB = A^{\overline{2}}\odot^1 B = A^{2.1}B = A \cdot_1 B.$$

There's another useful piece of notation. If A is an N-dimensional tensor and $P = p_1 \cdots p_N$ is a permutation of the list $1, 2, \ldots, N$, then

$$A_P = (A_{i_1\cdots i_N} \mid i_{p_1} i_{p_2} \cdots i_{p_N}). \tag{6}$$

We can extend this notation by enclosing in *square brackets* dimensions which we want to multiply together element-wise. For instance,

$$A_{p_1[p_2p_5p_7]p_3[p_4p_6]p_8\cdots} = \left(A_{I[p_2p_5p_7p_4p_6]\leftarrow kkkhh}\middle| i_{p_1}ki_{p_3}hi_{p_8}\cdots\right).$$

This should be clear enough without giving a general definition. As before, we can mark the positions we want to sum over. If some positions are between square brackets together, then the mark must be put only on one of them (it doesn't matter which one). We'll see why later. As an example, consider this:

$$A_{\overline{p_1}[p_2p_5p_7]p_3[p_4\overline{p_6}]p_8\cdots} = \left(A_{I[p_2p_5p_7p_4p_6]\leftarrow kkkhh}\middle| ki_{p_3}i_{p_8}\cdots\right).$$

Note that we're summing over i_{p_1} and h (which corresponds to $[p_4p_6]$).

Finally, since this is the subsection of formal definitions, let's introduce a trivial, but handy, piece of notation. Let $\mathbb{T}_n$ be the set of *generic* n-dimensional tensors. By *generic* we mean that we abstract over the *field* (e.g. $\mathbb{R}$) and the "*sizes*" of the dimensions, so we'll write $A \in \mathbb{T}_n$ instead of $A \in \mathbb{K}^{s_1\times\cdots\times s_n}$ for some field $\mathbb{K}$ and sizes $s_1, \ldots, s_n$.

2.3 General Bracket Notation

We've already seen a light version of the *bracket notation*. Instead of giving a rigorous specification of this notation, we'll see simplified definitions and a few examples.

Let A be an n-dimensional tensor. Let's start with a simple equality:

$$A = A_{1\cdots n} = A_1 A_2 \cdots A_n.$$

Note that this is a generalization of notation 6. First, A_1 wasn't allowed before because 1 is not a permutation of $1 \cdots n$, of course. Second, we didn't define what it means to *juxtapose* two terms like in A_1A_2. Conceptually, juxtaposition should be equivalent to the Kronecker operator, but there's an important

difference: juxtaposition is a *formal* Kronecker product. What I mean is that it's just a way to carry out manipulations with portions of a tensor. It is *not semantically* equivalent to the Kronecker product. For instance, if u, v, w are 3 vectors, then

$$u \otimes v \otimes w = (u_i v_j w_k \mid ijk) = uvw, \tag{7}$$

while $A_1 \otimes A_{23} \otimes A_4$ doesn't mean anything because the portion $A_{5\cdots n}$ is completely missing. So what's the point of writing $A_1 A_{23} A_4$ in the first place? The point is that it lets us do useful manipulations with the portions of A. The important thing is that no term in the final expression misses any portions of any tensors. We'll see some examples later.

Note that uvw in example 7 is written in blue. We'll write the parts expressed in bracket notation in blue for more clarity.

Square brackets multiply the dimensions of all the terms within them (all of the same dimension) element-wise:

$$[A_P B_Q A_R (C_S C_T)] = A_P \odot B_Q \odot A_R \odot (C_S \otimes C_T),$$

where $\dim(A_P) = \dim(B_Q) = \dim(A_R) = \dim(C_S C_T)$. Note that "$\odot$" is also used *formally* here because the terms are not complete tensors but only portions.

We can reorder the terms withing square brackets however we want, so, in general,

$$[AB] = [BA].$$

To sum over all the dimensions of a term, it's enough to put a *bar* on it. For instance,

$$A_P \overline{A_Q} A_R$$

means that we sum over all the dimensions of A_Q.

In a sense, terms with a bar over them become *scalars* so we can put them wherever we want:

$$A_P \overline{A_Q} A_R = \overline{A_Q} A_P A_R = A_P A_R \overline{A_Q}.$$

We can also sum over dimensions represented by pairs of square brackets, of course:

$$[A_P \overline{B_Q} A_R (C_S C_T)] = [A_P B_Q \overline{A_R} (C_S C_T)] = [A_P B_Q A_R \overline{(C_S C_T)}].$$

Note that we put the bar over just a single term, no matter which one. Why not putting the bar over all the terms then? The idea is that we want to be able to write things like this:

$$[A_P B_Q A_R (\overline{C_S} C_T)],$$

which means that we only sum over the first $\dim(C_S)$ dimensions. This may be useful when we take something out:

$$[\overline{A_{12}} B_{12} (C_1 D_1)] D_2 = [A_{12} B_{12} (\overline{C_1} 1)] \cdot_1 D_{12}.$$

Note that

$$[A_{12}B_{12}(\overline{C_1}1)] = [\overline{A_1}B_1C_1][A_2B_2],$$

where 1 is a vector of all 1. In general, $1_{1\cdots n}$ is an n-dimensional tensor of all 1. Feel free to come up with your notation. Maybe you prefer 1_n, $1_{1:n}$ or even $1_{\#n}$ where "#" indicates that n is a count and not a position.

As before, we can move summed over terms wherever we want:

$$X[\overline{A}BC]Y = [\overline{A}BC]XY = XY[\overline{A}BC].$$

This holds no matter the *nesting level*:

$$X[AB(C_1[\overline{C_2}D_1]D_2)]Y = [\overline{C_2}D_1]X[AB(C_1D_2)]Y.$$

In conclusion, we may say that, for instance,

$$X[\overline{A}B(C_1[\overline{C_2}D_1]D_2)]Y = \begin{cases} XY \\ [\overline{A}B(C_1D_2)] \\ [\overline{C_2}D_1]. \end{cases}$$

Rewriting a formula as a *system* of formulas may simplify things, because the individual formulas are simpler to parse.

As we said in subsection 2.2, we need to pay special attention to the order of the dimensions. For instance, let's reconsider equations 4 and 5:

$$A\,{}^{12}\!\odot^{32}B = (A_{ijk}B_{lmn} \mid ijklmn) \rightsquigarrow (A_{ijk}B_{lji} \mid ijkl)$$
$$B\,{}^{32}\!\odot^{12}A = (B_{lmn}A_{ijk} \mid lmnijk) \rightsquigarrow (B_{lji}A_{ijk} \mid ljik)$$

With bracket notation, we should proceed as follows:

$$A\,{}^{12}\!\odot^{32}B = A \rightsquigarrow [A_1B_3]A_{23} \rightsquigarrow [A_1B_3][A_2B_2]A_3 \rightsquigarrow [A_1B_3][A_2B_2]A_3B_1 \quad (8)$$
$$B\,{}^{32}\!\odot^{12}A = B \rightsquigarrow B_{12}[B_3A_1] \rightsquigarrow B_1[B_2A_2][B_3A_1] \rightsquigarrow B_1[B_2A_2][B_3A_1]A_3 \quad (9)$$

Note that the dimensions of the left operand always appear in increasing order in the result. The *free dimensions* of the right operand appear also in increasing order in the result, but all at the end.

2.3.1 A remark about vectors

While a 1-dimensional tensor has no "orientation", vectors can be *column vectors* or *row vectors*. This may be a problem:

$$(\mathbf{1}^Tu)v = [\overline{1}u]v = \mathbf{1} \cdot_1 uv = \mathbf{1} \cdot_1 (uv^T) \neq \mathbf{1}^T(uv^T). \quad (10)$$

The last step is *wrong* because it changes the orientation of the vector: $(\mathbf{1}^Tu)v$ is a column vector but $\mathbf{1}^T(uv^T)$ is a row vector.

We can solve this "problem" by regarding row vectors as matrices, i.e. 2-dimensional tensors. We can keep pretending that column vectors are 1-dimensional tensors because their elements lie along the first dimension.

To do so, we can introduce a new symbol, "ι" (iota), to represent a dimension of size 1 (as in $u^T \in \mathbb{R}^{1\times n}$), so, if u is a column vector, then

$$u^T = \iota u$$

This is useful to remind us that $\mathbf{1} \cdot_1 (uv^T) = [\overline{1}u]v$ is 1-dimensional so it's a column vector. Indeed, there's no "ι". So here's how to fix example 10:

$$(\mathbf{1}^T u)v = [\overline{1}u]v = v[u\overline{1}] = vu^T\mathbf{1}$$

Here's an example of when we need to use "ι":

$$(\mathbf{1}^T u)v^T = [\overline{1}u]\iota v = \iota[\overline{1}u]v = \iota 1 \cdot_1 uv = 1^T uv^T$$

which means that we can drop the parentheses, as it should be obvious. We're just checking that things works out, here. Note that we can't drop the parentheses in $(1^T u)v$, unless we rewrite it as $v(1^T u)$.

2.4 Associativity

Since "$\odot$" and "$\cdot$" are so general, we must pay particular attention to *associativity*. Sometimes it holds, other times it doesn't.

Let's consider 3 matrices A, B and C and their product

$$A \,{}^{1\cdot 1}B \,{}^{1\cdot 1}C.$$

If we evaluate it from left to right we get

$$\begin{aligned}(A \,{}^{1\cdot 1}B) \,{}^{1\cdot 1}C &= ([\overline{A_1}B_1]A_2B_2) \,{}^{1\cdot 1}C \\ &= [\overline{A_1}B_1][\overline{A_2}C_1]B_2C_2,\end{aligned}$$

whereas if we evaluate it from right to left we get

$$\begin{aligned}A \,{}^{1\cdot 1}(B \,{}^{1\cdot 1}C) &= A \,{}^{1\cdot 1}([\overline{B_1}C_1]B_2C_2) \\ &= [\overline{B_1}C_1][\overline{A_1}B_2]A_2C_2.\end{aligned}$$

As we can see, we get two different results.

Why is that? The problem is that summing over dimensions changes the positions of the following dimensions.

Let's try with the element-wise product:

$$\begin{aligned}(A \,{}^1\odot^1 B) \,{}^1\odot^1 C &= ([A_1B_1]A_2B_2) \,{}^1\odot^1 C \\ &= [A_1B_1C_1]A_2B_2C_2 \\ A \,{}^1\odot^1 (B \,{}^1\odot^1 C) &= A \,{}^1\odot^1 ([B_1C_1]B_2C_2) \\ &= [A_1B_1C_1]A_2B_2C_2\end{aligned}$$

We get the same result. This is not always true, though:

$$\begin{aligned}(A \,{}^2\odot^1 B) \,{}^1\odot^1 C &= (A_1[A_2B_1]B_2) \,{}^1\odot^1 C \\ &= [A_1C_1][A_2B_1]B_2C_2 \\ A \,{}^2\odot^1 (B \,{}^1\odot^1 C) &= A \,{}^2\odot^1 ([B_1C_1]B_2C_2) \\ &= A_1[A_2B_1C_1]B_2C_2\end{aligned}$$

2.5 Associative Operators

First of all, note that $\otimes$ is already associative for the simple reason that it doesn't let us specify indices like the other operators.

Now let's review two variants of "$\odot$" and "$\cdot$" that we presented in subsection 2.2 as simple shortcuts but that are actually quite interesting for their associativity.

If $A \in \mathbb{T}_n$ and $B \in \mathbb{T}_m$, then:

$$A \odot_k B = A\,{}^{(n-k+1)\cdots n}\!\odot^{1\cdots k} B$$
$$A \cdot_k B = A\,{}^{(n-k+1)\cdots n}\!.^{1\cdots k} B$$

In words, these two operators combine the last k dimensions of A with the first k dimensions of B.

For the two operators to be associative there are some conditions though:

$$\max(k,h) \leq \dim(\mathrm{B}) \implies (A \odot_k B) \odot_h C = A \odot_k (B \odot_h C)$$
$$k + h \leq \dim(B) \implies (A \cdot_k B) \cdot_h C = A \cdot_k (B \cdot_h C)$$

The implication in the first formula is not strictly necessary because if the left part (*antecedent*) doesn't hold then the right part (*consequent*) doesn't make sense. The second formula says that the left "portion" of B influenced by A *must not overlap* with the right "portion" of B influenced by C.

Let's consider the first formula. $A \odot_k B$ ends with the m dimensions of B, although the first k dimensions of B are fused with the last k of A. Anyway, C will fuse exactly with the last h dimensions of B and, possibly, with some dimensions of A which are already fused with the dimensions of B. Since the fusions of the dimensions at the same position are associative (and commutative), nothing changes if one first computes $B \odot_h C$ instead.

The second formula is even easier to justify because A and C don't interact at all so the order of association is clearly irrelevant.

Note that we also have

$$k + h \leq \dim(B) \implies (A \cdot_k B) \odot_h C = A \cdot_k (B \odot_h C)$$
$$k + h \leq \dim(B) \implies (A \odot_k B) \cdot_h C = A \odot_k (B \cdot_h C).$$

Finally, note that, if $d = \dim(U) = \dim(V)$, then

$$U \odot_d V = U \odot V = V \odot U = V \odot_d U$$
$$U \cdot_d V = U \cdot V = V \cdot U = V \cdot_d U.$$

3 Gradients, Jacobians and Differentials

Let $f : V \to W$ be a differentiable function where V and W are two *Banach Spaces*, i.e. *complete normed vector spaces*. $\mathbb{R}^n$ is just a particular Banach

Space, so you can just assume that f goes from $\mathbb{R}^n$ to $\mathbb{R}^m$, if you want. The *differential* of f at x_0 is the *linear* function $df(x_0; dx)$ such that

$$f(x_0 + dx) = f(x_0) + df(x_0; dx) + o(dx), \tag{11}$$

which means that $df(x_0; dx)$ is the *best linear approximation* to the function $dx \mapsto f(x_0 + dx) - f(x_0)$.

Since $df(x_0; dx)$ is linear, we can rewrite equation 11 as

$$f(x_0 + dx) = f(x_0) + ||dx|| df\left(x_0; \frac{dx}{||dx||}\right) + o(dx),$$

which will be useful later.

If V and W are finite-dimensional then the differential can be expressed as Adx where A is the *Jacobian* of f at x_0.

A function $f : \mathbb{R}^{n_1 \times \cdots \times n_N} \to \mathbb{R}^{m_1 \times \cdots \times m_M}$, i.e. a function from an N-dimensional array to an M-dimensional array is still a function from a vector space to another. We just need to choose appropriate *bases* for the two vector spaces. The *one-hot* arrays $e_{j_1 j_2 \cdots j_N}$ we defined before form the canonical basis for $\mathbb{R}^{n_1 \times \cdots \times n_N}$. Many authors, to simplify things, *vectorize* the matrices and arrays so that they can be treated as vectors. We'll work *directly* with the arrays instead.

As applied mathematicians, we're not very interested in all the theory about *Fréchet derivatives* and Banach Spaces. We just want a way to *compute* and then *retrieve* all the *partial derivatives*

$$\frac{\partial f_{i_1 i_2 \cdots i_M}}{\partial x_{j_1 j_2 \cdots j_N}}.$$

For $M = N = 1$ we have the usual Jacobian we all know from *Multivariate Calculus.*

The interesting thing about the differential $df(x_0; dx)$ is that it can be rewritten as follows:

$$df(x_0; dx)_I = \sum_J \frac{\partial f_I(x_0)}{\partial x_J} dx_J,$$

where $I = i_1 \cdots i_M$, $J = j_1 \cdots j_N$, and $f_I(x_0)$ is really $f(x_0)_I$. This implies the following identity:

$$df(x_0; e_J)_I = \frac{\partial f_I(x_0)}{\partial x_J} = \frac{\partial f(x_0)_I}{\partial x_J}. \tag{12}$$

Let's see a few easy examples. For a function $f : \mathbb{R} \to \mathbb{R}$ we get

$$df(x_0; dx) = \frac{df}{dx} dx,$$

so the coefficient of dx is the derivative $f'(x_0)$.

For a function $f : \mathbb{R}^n \to \mathbb{R}$ we get something slightly more complex:

$$df(x_0; dx) = \sum_{j=1}^{n} \frac{\partial f}{\partial x_j} dx_j.$$

Analogously to before, the coefficient of dx_i is the partial derivative $f_{x_i}(x_0)$. Note that we don't care about whether the partial derivatives form a column vector (the gradient) or a row vector (the Jacobian). What's important is that dx tells us exactly how the partial derivatives are laid out.

For a function $f : \mathbb{R}^n \to \mathbb{R}^m$, where $f = (f_1, \dots f_m)$, we get

$$df(x_0; dx)_i = \sum_{j=1}^{n} \frac{\partial f_i}{\partial x_j} dx_j$$

which can also be written as

$$\begin{aligned} df(x_0; dx) &= \sum_{j=1}^{n} dx_j \frac{\partial f}{\partial x_j} \\ &= \sum_{j=1}^{n} dx_j \left(\frac{\partial f_1}{\partial x_j}, \dots, \frac{\partial f_m}{\partial x_j} \right). \end{aligned}$$

In matrix form, this becomes

$$df(x_0; dx) = M dx = \sum_{j=1}^{n} dx_j M_{\cdot j}$$

where M is the Jacobian matrix and $M_{\cdot j}$ is the j-th column of M.

Now, let's consider $f : \mathbb{R}^{n_1 \times n_2 \times n_3} \to \mathbb{R}$, i.e. a function from a 3-dimensional array to a scalar. Here's the differential:

$$\begin{aligned} df(x_0; dx) &= M \cdot dx \\ &= M^{\,123}{}_{\cdot}{}^{123} dx \\ &= \sum_{i_1=1}^{n_1} \sum_{i_2=1}^{n_2} \sum_{i_3=1}^{n_3} M_{i_1 i_2 i_3} dx_{i_1 i_2 i_3}. \end{aligned}$$

What's M? A gradient? It would be if we had vectorized M, but we didn't. We don't care about what's M, because dx tells us how the partial derivatives are laid out.

The general case is that of a function $\mathbb{T}_N \to \mathbb{T}_M$. For instance, let's consider $f : \mathbb{R}^{n_1 \times n_2 \times n_3} \to \mathbb{R}^{m_1 \times m_2}$ which gives

$$df(x_0; dx) = M^{\,345}{}_{\cdot}{}^{123} dx = M \cdot_3 dx \tag{13}$$

or

$$df(x_0; dx) = M^{\,512}{}_{\cdot}{}^{123} dx \tag{14}$$

or something like that. Note that M has 5 dimensions. The term dx tells us exactly where the partial derivatives are. For instance, equality 14 tells us that

$$\frac{\partial f_{ij}}{\partial x_{p_1 p_2 p_3}} = M_{p_2 p_3 i j p_1}.$$

This can be verified by remembering identity 12:

$$\begin{aligned}\frac{\partial f_{ij}}{\partial x_{p_1 p_2 p_3}} &= df(x_0; e_{p_1 p_2 p_3})_{ij} \\ &= \left[M^{\,512}_{\cdot\,123} e_{p_1 p_2 p_3}\right]_{ij} \\ &= \left[M_{p_2 p_3 \cdot\cdot p_1}\right]_{ij} \\ &= M_{p_2 p_3 i j p_1}.\end{aligned}$$

If you really want to thoroughly understand this derivation, skip ahead, read subsections 4.3.2 and 4.3.3, and then come back, but I wouldn't worry too much about this example for now.

In practice, it's more convenient to permute the dimensions of M so that the differential can be expressed as in equality 13. In that case, we get

$$\frac{\partial f_{ij}}{\partial x_{pqr}} = M_{ijpqr},$$

which should remind us of the usual Jacobian J where

$$\frac{\partial f_i}{\partial x_p} = J_{ip}.$$

3.1 Differentials VS derivatives

Let f, g be two vector functions and $h = g \circ f$ their composition. We know that the derivative of h at x_0 is

$$D_h(x_0) = D_g(f(x_0)) D_f(x_0),$$

or, if $x = x_0$, $f = f(x)$ and $g = g(f)$,

$$\frac{\partial g}{\partial x} = \frac{\partial g}{\partial f} \frac{\partial f}{\partial x} \tag{15}$$

which suggests that the two ∂f cancel out, some say, as if the two factors were real fractions. This is not the right way to look at it, though. They don't cancel out: they *contract*. Let's rewrite equality 15 as

$$\frac{\partial g}{\partial x^T} = \frac{\partial g}{\partial f^T} \frac{\partial f}{\partial x^T}$$

so to emphasize that the numerator varies vertically (along the columns) and the denominator horizontally (along the rows). Because of the *row-by-column*

multiplication between matrices, we see that, in a way, the denominator of the first factor is multiplied element-wise with the numerator of the second factor and then the resulting dimension is summed over. More explicitly

$$\frac{\partial g}{\partial x^T} = \frac{\partial g}{\partial f^T} \,{}^{2}\cdot^{1}\, \frac{\partial f}{\partial x^T}.$$

The fact that the denominator is transposed tells us that it varies along the second dimension.

Now let's take two functions $f : \mathbb{R}^{n_1 \times n_2 \times n_3} \to \mathbb{R}^{m_1 \times m_2}$ and $g : \mathbb{R}^{m_1 \times m_2} \to \mathbb{R}^{l_1}$ and let their differentials be

$$\begin{aligned} df &= M_f \,{}^{345}\cdot^{123} dx \\ dg &= M_g \,{}^{13}\cdot^{12} df. \end{aligned}$$

Note that M_f has $3 + 2 = 5$ dimensions while M_g has $2 + 1 = 3$ dimensions. The two derivatives appear to be M_f and M_g, but how do we multiply them to give $M_{g \circ f}$?

The beauty of differentials is that they tell us exactly how we can combine them. Instead of simply multiplying derivatives according to some arbitrary convention about their structure, we can just combine differentials by *substitution*:

$$\begin{aligned} dg &= M_g \,{}^{13}\cdot^{12} df \\ &= M_g \,{}^{13}\cdot^{12} (M_f \,{}^{345}\cdot^{123} dx) \\ &= (M_g \,{}^{13}\cdot^{12} M_f) \,{}^{345}\cdot^{123} dx. \end{aligned}$$

Note that the associativity holds because the left and the right operators influence non-overlapping portions of M_f (dimensions $1, 2$ and $3, 4, 5$, respectively).

In Machine Learning, we often need to compute the gradient of a *loss function* with respect to some parameters. In general, if

$$df = A \cdot dW = \sum_{i_1 \cdots i_N} A_{i_1 \cdots i_N} dW_{i_1 \cdots i_N},$$

then A is the "gradient" of f with respect to W. Gradient is written between quotation marks because W is not a vector but a full tensor.

If you want, you can also use the symbol "∂" instead of "d":

$$\partial f = A \cdot \partial W.$$

We can also compute the gradients of multiple parameters all at once:

$$df = A_1 \cdot dW_1 + \cdots + A_k \cdot dW_k \implies \nabla_{W_i} f = A_i, \qquad i = 1, \ldots, k.$$

4 Useful rules about differentials

4.1 A few general results

Before proceeding, let's decide that the $d\cdot$ operator binds stronger than any other operator, so that we can just write dX^T instead of $(dX)^T$ without any ambiguity.

As we said in section 3, if f and g are differentiable, then

$$\begin{aligned} f(x_0 + dx) &= f(x_0) + df(x_0; dx) + o(dx) \\ g(x_0 + dx) &= g(x_0) + dg(x_0; dx) + o(dx), \end{aligned}$$

which lets us derive a few simple rules about differentials and the $d\cdot$ operator in particular.

First of all, $d\cdot$ is *linear*, i.e. $d(af + g) = ad(f) + d(g)$, where a is a scalar:

$$\begin{aligned} (af + g)(x_0 + dx) &= af(x_0 + dx) + g(x_0 + dx) \\ &= a(f(x_0) + df(x_0; dx) + o(dx)) + \\ &\quad g(x_0) + dg(x_0; dx) + o(dx) \\ &= (af + g)(x_0) + [adf(x_0; dx) + dg(x_0; dx)] + o(dx). \end{aligned}$$

Second, $d(fg) = dfg + fdg$:

$$\begin{aligned} (fg)(x_0 + dx) &= f(x_0 + dx)g(x_0 + dx) \\ &= [f(x_0) + df(x_0; dx) + o(dx)][g(x_0) + dg(x_0; dx) + o(dx)] \\ &= f(x_0)g(x_0) + df(x_0; dx)g(x_0) + f(x_0)dg(x_0; dx) + \\ &\quad df(x_0; dx)dg(x_0; dx) + \\ &\quad o(dx)[f(x_0) + df(x_0; dx) + g(x_0) + dg(x_0; dx)] + o(dx^2) \\ &= f(x_0)g(x_0) + df(x_0; dx)g(x_0) + f(x_0)dg(x_0; dx) + \\ &\quad ||dx||^2 df\left(x_0; \frac{dx}{||dx||}\right) dg\left(x_0; \frac{dx}{||dx||}\right) + \\ &\quad o(dx)[f(x_0) + df(x_0; dx) + g(x_0) + dg(x_0; dx)] + o(dx^2) \\ &= f(x_0)g(x_0) + [df(x_0; dx)g(x_0) + f(x_0)dg(x_0; dx)] + o(dx). \end{aligned}$$

Third, $dg(x_0; dx) = dg(f(x_0); df(x_0; dx))$. This just means what we said in subsection 3.1 about substitutions, i.e. that if

$$\begin{aligned} df &= F(dx) \\ dg &= G(df), \end{aligned}$$

then

$$dg = G(df) = G(F(dx)).$$

That's it: simple substitution!

Let's prove it:

$$\begin{aligned}(g \circ f)(x_0 + dx) &= g(f(x_0 + dx)) \\ &= g(f(x_0) + df(x_0; dx) + o(dx)) \\ &= g(f(x_0) + [df(x_0; dx) + o(dx)]) \\ &= g(f(x_0)) + dg(f(x_0); [df(x_0; dx) + o(dx)]) + \\ &\quad o([df(x_0; dx) + o(dx)]) \\ &= g(f(x_0)) + dg(f(x_0); df(x_0; dx)) + dg(f(x_0); o(dx)) + \\ &\quad o(df(x_0; dx)) + o(dx^2) \\ &= g(f(x_0)) + dg(f(x_0); df(x_0; dx)) + o(dx).\end{aligned}$$

These results hold for all our tensor operators for the simple reason that the derivations above use only properties shared by all our operators (basically, *distributivity* with respect to addition).

Another useful result is that

$$d(X_P) = (dX)_P,$$

where $X \mapsto X_P$ is any operation which rearranges the elements or dimensions of X. For instance, $d(X^T) = dX^T$ and, if Y is a 3-dimensional tensor, $d(Y_{312}) = dY_{312}$. Remember that $d\cdot$ binds stronger than any other operator, so $dX_P = (dX)_P$.

4.2 Differentiation in General Bracket Notation

In general, if $B(X, \ldots, X)$ is a term in bracket notation where X appears one or more times (in any order or split into portions), then the differential of B with respect to X is

$$\begin{aligned}d(B(X, X, \ldots, X)) = {} & B(dX, X, \ldots, X) + B(X, dX, \ldots, X) + \cdots + \\ & B(X, X, \ldots, dX).\end{aligned}$$

For example, let X be a 3-dimensional tensor and

$$B(X, X, X) = A_1 X[X_2 A_2] X_3[X_1 C_2] C_1 D X_{23}[\overline{X_1} C_3].$$

Then

$$\begin{aligned}dB &= d(A_1 X[X_2 A_2] X_3[X_1 C_2] C_1 D X_{23}[\overline{X_1} C_3]) \\ &= A_1 \boldsymbol{dX}[X_2 A_2] X_3[X_1 C_2] C_1 D X_{23}[\overline{X_1} C_3] + \\ &\quad A_1 X[\boldsymbol{dX_2} A_2] \boldsymbol{dX_3}[\boldsymbol{dX_1} C_2] C_1 D X_{23}[\overline{X_1} C_3] + \\ &\quad A_1 X[X_2 A_2] X_3[X_1 C_2] C_1 D \boldsymbol{dX_{23}}[\overline{\boldsymbol{dX_1}} C_3] \\ &= A_1 \boldsymbol{dX}[X_2 A_2] X_3[X_1 C_2] C_1 D X_{23}[\overline{X_1} C_3] + \\ &\quad A_1 X[X_2 A_2] X_3[\boldsymbol{dX_1} C_2] C_1 D \boldsymbol{dX_{23}}[\overline{X_1} C_3] + \\ &\quad A_1 X[\boldsymbol{dX_2} A_2] \boldsymbol{dX_3}[X_1 C_2] C_1 D X_{23}[\overline{\boldsymbol{dX_1}} C_3].\end{aligned}$$

As you can see, we can select any portions of X we want as long as any portion of X in B is selected *exactly once* and each addend contains *exactly one* dX (as a whole or split into portions).

This can be proved very easily by noticing that we can *take out* the portions we're interested in. For instance:

$$\begin{aligned}
dB &= d(A_1 X[X_2 A_2]X_3[X_1 C_2]C_1 D X_{23}[\overline{X_1} C_3]) \\
&= d(X \,\overline{\odot}^{123}\, \odot^{10,3,4} A_1 X A_2 1[X_1 C_2]C_1 D X_{23} C_3) \\
&= dX \,\overline{\odot}^{123}\, \odot^{10,3,4} A_1 X A_2 1[X_1 C_2]C_1 D X_{23} C_3 + \\
&\quad X \,\overline{\odot}^{123}\, \odot^{10,3,4} d(A_1 X A_2 1[X_1 C_2]C_1 D X_{23} C_3) \\
&= A_1 X[\boldsymbol{dX_2} A_2]\boldsymbol{dX_3}[X_1 C_2]C_1 D X_{23}[\overline{\boldsymbol{dX_1}} C_3] + \\
&\quad d(A_1 X[\cancel{X_2} A_2]\cancel{X_3}[X_1 C_2]C_1 D X_{23}[\overline{\cancel{X_1}} C_3]),
\end{aligned}$$

where terms *stricken through obliquely* (e.g. $\cancel{X_2}$) are still present but can't be reused and must be regarded as *constants*.

4.3 Function application

4.3.1 $f : \mathbb{R} \to \mathbb{R}$ case

Sometimes we want to apply functions element-wise to tensors. For instance, if f is an $\mathbb{R} \to \mathbb{R}$ function and A a matrix, then $[f(A)]_{ij} = f(A_{ij})$. In general, for $f : \mathbb{R} \to \mathbb{R}$ and $A \in \mathbb{T}_n$,

$$f(A)_I = f(A_I). \tag{16}$$

We can easily prove that if f is an $\mathbb{R} \to \mathbb{R}$ function and X a tensor, then

$$df(X) = f'(X) \odot dX.$$

Let's prove it:

$$\begin{aligned}
df(X; e_J)_I &= \frac{\partial f(X)_I}{\partial X_J} && \text{(for 12)} \\
&= \frac{\partial f(X_I)}{\partial X_J} && \text{(for 16)} \\
&= \frac{\partial f(X_I)}{\partial X_I}\frac{\partial X_I}{\partial X_J} \\
&= f'(X_I)\frac{\partial X_I}{\partial X_J} \\
&= f'(X_I)\delta_{J=I} \\
&= [f'(X) \odot e_J]_I \implies \\
&\quad df(X; dX) = f'(X) \odot dX. && (17)
\end{aligned}$$

Note that identity 17 can be rewritten as

$$[df(X; dX)]_I = f'(X_I) dX_I. \tag{18}$$

Of course, this is only valid in this specific case, i.e. for $f : \mathbb{R} \to \mathbb{R}$.

4.3.2 Partial and Dot Indices

To be able to reason about tensors we'll need to introduce the concepts of (what I call) *partial indices* and *dot indices*. They're useful because they let us index *subtensors* of tensors instead of just single *scalars*.

For instance, if M is a matrix, the rows and columns of M are subtensors of M indexed by dot indices of the form $i\cdot$ and $\cdot j$, respectively. The positions with a *dot* are called *dot* positions.

Indices which correspond to single dimensions are written in *lowercase* and are called *scalar indices*. For instance, if M is a matrix, then i and j in M_{ij} are scalar indices. A *multidimensional index* is a list of one or more scalar indices. For instance, $I = ijk$ is a multidimensional index and A_I is equivalent to "A_{ijk}". Multidimensional indices are especially useful in definitions and proofs where we often want to index the elements or subtensors of generic tensors.

A scalar index has dimension 1 while a multidimensional index has dimension equal to the *length* of the list of scalar indices it represents. We can use *subscripts* to indicate the dimension of an index. For instance, I_n and J_m are indices of dimension n and m, respectively.

Let $P = p_1 \cdots p_k$ be a list of positions in $\{1, \ldots, n\}$ (as always, *without repetitions*). Let $I_n = i_1 \cdots i_n$, $J_k = j_1 \cdots j_k$, and so on. Then:

- We can *extract* sub-indices from other indices:

$$I_n[P] = i_{p_1} \cdots i_{p_k}$$

- We can *spread out* indices:

$$\begin{aligned} \dim(I_k^P) &= n \\ I_k^P[P] &= I_k \\ I_k^P[-P] &= \epsilon_{n-k} \end{aligned}$$

  where ϵ_{n-k} is a list of $n-k$ "no element". This means that I_k^P is I_k "spread out" over the positions in P (not necessarily increasing) and all the positions $-P$ (*always* increasing) are holes, i.e. they have no elements. I_k^P is called a *partial index* because it has holes. A *full index* is an index without any holes. **Do not confuse *holes* with dots!**

- We can *fill* missing positions. If $J = J_{n-k}J_m$ is a full index, then

$$\begin{aligned} \dim(I_k^P J) &= n + m \\ (I_k^P J)[P] &= I_k && (19) \\ (I_k^P J)[-P] &= J_{n-k} && (20) \\ (I_k^P J)[(n+1) \cdots (n+m)] &= J_m, \qquad \text{for } m > 0 && (21) \end{aligned}$$

  If $m = 0$ then $J = J_{n-k}$ and condition 21 doesn't apply. Note that $I_k^P J$ is a full index for any (non negative) m.

If $J = J_h$, $h < n - k$, then

$$\begin{aligned}\dim(I_k^P J) &= n \\ (I_k^P J)[P] &= I_k \\ (I_k^P J)[-P] &= J_h \epsilon_{n-k-h}\end{aligned}$$

which means that $I_k^P J$ is a partial index with $n - k - h$ holes.
Note that we can fill missing positions with dots as in $I_k^P \cdot_{n-k}$ or, more concisely, "$I_k^P \cdot$". We often use a single (vector) dot to mean that all the missing positions are filled with dots.

Now we can define what dot indices mean more formally:

$$(X_{I_k^P \cdot_{n-k}})_{I_{n-k}} = X_{I_k^P I_{n-k}}$$

Note that I stopped adding punctuation at the end of the formulas because things might get confusing now that we use dot indices.

4.3.3 Useful manipulations

Here are some useful *manipulations* we'll need soon:

- We can *split* (or *join*) indices:

$$X_{I_n I_k I_m} = (X_{I_n \cdot_k I_m})_{I_k} \tag{22}$$
$$X_{\cdot_n I_k I_m} = (X_{\cdot_n \cdot_k I_m})_{\cdot_n I_k} \tag{23}$$

  where the indices can also be partial. Note that I_k takes the place of the dots. Identity (23) is trickier because we need to replace the dots with $\cdot_n I_k$ instead of just I_k, otherwise we'd get $I_k \cdot_n I_m$ instead of "$\cdot_n I_k I_m$". Here's a simple example where M is a matrix:

$$M_{24} = (M_{2\cdot})_4 = (M_{\cdot 4})_2$$

- We can replace δ with e and vice versa:

$$\begin{aligned}X_{I_n I_k I_m} \delta_{I_k = J_k} &= X_{I_n I_k I_m}[e_{J_k}]_{I_k} \\ &= [X_{I_n \cdot_k I_m} \odot e_{J_k}]_{I_k} \\ X_{I_n I_k I_m} \delta_{I_k = J_k} &= X_{I_n I_k I_m}[e_{I_k}]_{J_k} \\ &= [X_{I_n \cdot_k I_m} \odot e_{I_k}]_{J_k}\end{aligned} \tag{24}$$

- We can replace an index with an e:

$$\begin{aligned} X_{I_n I_k I_m} &= [X_{I_n \cdot_k I_m}]_{I_k} \\ &= \sum_{J_k} [X_{I_n \cdot_k I_m}]_{J_k} \delta_{J_k = I_k} \\ &= \sum_{J_k} [X_{I_n \cdot_k I_m}]_{J_k} [e_{I_k}]_{J_k} \\ &= \sum_{J_k} [X_{I_n \cdot_k I_m} \odot e_{I_k}]_{J_k} \\ &= X_{I_n \cdot_k I_m} \cdot e_{I_k} \end{aligned} \tag{25}$$

4.3.4 $f : \mathbb{T}_n \to \mathbb{T}_m$ case

We can generalize the simple result of subsection 4.3.1 to cases when we want to apply a function to every subtensor (in a tensor) indexed by a *dot index*.

Let X be an n-dimensional tensor and $P = p_1 \cdots p_k$ a list of positions. We now want to determine the differential of $f^P(X)$ for $f : \mathbb{T}_{n-k} \to \mathbb{T}_m$ where $f^P(X)$ is the result of applying f to all the subtensors of X indexed by an index of the form "$I_k^P \cdot$". Here's a natural definition of this operation:

$$f^P(X)_{I_k^P I_m} = f(X_{I_k^P \cdot})_{I_m} \qquad \text{(not used!)} \tag{26}$$

This definition is particular because f^P takes subtensors from positions $-P$ and puts the resulting tensors back into the same positions. For instance, if M is a matrix and f is $\mathbb{T}_1 \to \mathbb{T}_1$, then $f^2(M)$ modifies the columns of M one by one without moving them.

Unfortunately, as we'll see, this definition would complicate our final formula, so we'll use this definition instead:

$$f^P(X)_{I_k I_m} = f(X_{I_k^P \cdot})_{I_m} \tag{27}$$

Now we need to define a sort of *generalized Jacobian* or *generalized gradient*. Here's the generalized Jacobian:

$$\mathbf{J} f(X)_{I_m I_n} = \frac{\partial f(X)_{I_m}}{\partial X_{I_n}}.$$

Note that

$$g(X) = \mathbf{J} f(X) \iff df(X; dX) = g(X) \cdot_> dX$$

where $A \cdot_> B = A \cdot_{\dim(B)} B$ (useful when $\dim(A) > \dim(B)$) and $A \cdot_< B = A \cdot_{\dim(A)} B$ (useful when $\dim(A) < \dim(B)$).

Since the gradient is the *transpose* of the Jacobian (for *scalar functions of vectors*) then the generalized gradient is the "transpose" of the generalized Jacobian:

$$\nabla f(X)_{I_n I_m} = \frac{\partial f(X)_{I_m}}{\partial X_{I_n}}. \tag{28}$$

Analogously to the case of the generalized Jacobian, $\nabla f(X)$ can be easily found by exploiting the following fact:

$$g(X) = \nabla f(X) \iff df(X; dX) = dX \cdot_< g(X) \tag{29}$$

As before, let's assume that f is $\mathbb{T}_{n-k} \to \mathbb{T}_m$, X is an n-dimensional tensor and $P = p_1 \cdots p_k$ is a list of positions in $\{1, \ldots, n\}$ (as always, *without repetitions*). In what follows:

- indices of dimension k index the subtensors of X which f is applied to;
- indices of dimension $n-k$ index the partial derivatives of f;
- indices of dimension m index the elements of the tensor returned by f.

Let's try to derive a formula for the general case by using definition 27 and the generalized gradient:

$$\begin{aligned}
&df^P(X; e_{J_k^P J_{n-k}})_{I_k I_m} = \\
&\frac{\partial f^P(X)_{I_k I_m}}{\partial X_{J_k^P J_{n-k}}} = && \text{(for 12)} \quad (30)\\
&\frac{\partial f(X_{I_k^P \cdot})_{I_m}}{\partial X_{J_k^P J_{n-k}}} = && \text{(for 27)}\\
&\frac{\partial f(X_{I_k^P \cdot})_{I_m}}{\partial X_{I_k^P J_{n-k}}} \frac{\partial X_{I_k^P J_{n-k}}}{\partial X_{J_k^P J_{n-k}}} = \\
&\frac{\partial f(X_{I_k^P \cdot})_{I_m}}{\partial (X_{I_k^P \cdot})_{J_{n-k}}} \frac{\partial X_{I_k^P J_{n-k}}}{\partial X_{J_k^P J_{n-k}}} = && \text{(for 22)}\\
&\nabla f(X_{I_k^P \cdot})_{J_{n-k} I_m} \delta_{I_k = J_k} = && \text{(for 28)}\\
&\nabla f^P(X)_{I_k J_{n-k} I_m} \delta_{I_k = J_k} = && \text{(for 27)} \quad (31)\\
&\left[\nabla f^P(X)_{\cdot_k J_{n-k} I_m}{}^{1 \cdots k} \odot^{1 \cdots k} e_{J_k}\right]_{I_k} = && \text{(for 24)}\\
&\left[\left(\nabla f^P(X)_{\cdot_k \cdot_{n-k} I_m}{}^{1 \cdots k} \odot^{1 \cdots k} e_{J_k}\right)^{(k+1) \cdots n}{}_{\cdot (k+1) \cdots n} e_{J_{n-k}}\right]_{I_k} = && \text{(for 25)}\\
&\left[\nabla f^P(X)_{\cdot_k \cdot_{n-k} I_m}{}^{1 \cdots k (k+1) \cdots n} \odot^{1 \cdots k \overline{(k+1) \cdots n}} e_{J_k J_{n-k}}\right]_{I_k} = \\
&\left[\left[\nabla f^P(X)_{\cdot_k \cdot_{n-k} \cdot_m}{}^{1 \cdots k (k+1) \cdots n} \odot^{1 \cdots k \overline{(k+1) \cdots n}} e_{J_k J_{n-k}}\right]_{\cdot_k I_m}\right]_{I_k} = && \text{(for 23)}\\
&\left[\nabla f^P(X)_{\cdot_k \cdot_{n-k} \cdot_m}{}^{1 \cdots k (k+1) \cdots n} \odot^{1 \cdots k \overline{(k+1) \cdots n}} e_{J_k J_{n-k}}\right]_{I_k I_m} = && \text{(for 22)}\\
&\left[\nabla f^P(X)^{1 \cdots n} \odot^{1 \cdots k \overline{(k+1) \cdots n}} e_{J_k J_{n-k}}\right]_{I_k I_m} = \\
&\left[\nabla f^P(X)^{1 \cdots n} \odot^{P \overline{(-P)}} e_{J_k^P J_{n-k}}\right]_{I_k I_m} \implies
\end{aligned}$$

$$df^P(X; dX) = \nabla f^P(X)^{1 \cdots n} \odot^{P \overline{(-P)}} dX \tag{32}$$

The final formula is pretty enough, isn't it? In case you skipped the proof, note that $\nabla f^P(X)$ is to be interpreted as $(\nabla f)^P(X)$, so we can compute ∇f separately.

If we had used the generalized Jacobian instead of the generalized gradient, expression 31 would have been

$$\mathbf{J}f^P(X)_{I_k I_m J_{n-k}} \delta_{I_k = J_k}$$

which is annoying because I_k and J_{n-k} are separated by I_m. This means that we wouldn't have get that nice $1 \cdots n$ in the final formula.

Also, if we had used definition 26 instead of definition 27, we would have ended up with

$$df^P(X; e_{J_k^P J_{n-k}})_{I_k^P I_m} = [\cdots]_{I_k I_m}$$

which can't be directly simplified. Of course, we could have manipulated the expression to get

$$df^P(X; e_{J_k^P J_{n-k}})_{I_k^P I_m} = [\cdots]_{I_k^P I_m}$$

but that would have made the final formula less pleasing to the eyes.

4.3.5 A simple example

Let $s : \mathbb{T}_1 \to \mathbb{T}_1$ be the *softmax* function defined as

$$s(a) = \frac{e^a}{\mathbf{1}^T e^a}.$$

We want to compute the differential of

$$Z = s^1(A)$$

where A is a matrix and s is applied to the rows of A.

By identity 32,

$$dZ = \nabla s^1(A) \, {}^{12}\!\odot^{1\bar{2}} dA$$

Now, since s and e^a are clearly two column vectors, we get:

$$\begin{aligned}
e^a &= (\mathbf{1}^T e^a) s \\
&= s \mathbf{1}^T e^a \implies \\
e^a \odot da &= ds \mathbf{1}^T e^a + s \mathbf{1}^T (e^a \odot da) \implies \\
ds \mathbf{1}^T e^a &= e^a \odot da - s \mathbf{1}^T (e^a \odot da) \\
&= (I - s \mathbf{1}^T)(e^a \odot da) \\
&= M(e^a \odot da) \implies \\
ds &= M(s \odot da) \\
&= M_1 [\overline{M_2} s da] \\
&= [\overline{da} s M_2] M_1
\end{aligned}$$

$$
\begin{aligned}
&= da \cdot_1 [sM_2]M_1 \\
&= da \cdot_1 (s \odot_1 M^T) \implies \\
\nabla s &= s \odot_1 M^T && \text{(for 29)} \\
&= s \odot_1 (I - s\mathbf{1}^T)^T \\
&= s \odot_1 (I - \mathbf{1}s^T) \\
&= s\mathbf{1}^T \odot (I - \mathbf{1}s^T) \\
&= s\mathbf{1}^T \odot I - s\mathbf{1}^T \odot \mathbf{1}s^T \\
&= \mathrm{diag}(s) - ss^T
\end{aligned}
$$

4.4 Common results about matrices

Here are a few results specific to matrices:

- $d(\mathrm{tr}(X)) = d(X \cdot I) = dX \cdot I = \mathrm{tr}(dX)$
- $d(X^{-1}) = -X^{-1}(dX)X^{-1}$
  Proof:

$$
\begin{aligned}
0 = d(I) = d(XX^{-1}) = dXX^{-1} + Xd(X^{-1}) &\implies \\
Xd(X^{-1}) = -dXX^{-1} &\implies \\
d(X^{-1}) = -X^{-1}dXX^{-1}.&
\end{aligned}
$$

- $d|X| = |X|X^{-T} \cdot dX$
  Proof:
  - Let C be the *cofactor matrix* of X, i.e. $C_{ij} = (-1)^{i+j}|X_{-i,-j}|$, where $X_{-i,-j}$ is the matrix obtained from X by deleting the i-th row and the j-th column.
  - We know from *Linear Algebra* that $X^{-1} = |X|^{-1}C^T$ and that $|X| = \sum_k X_{ik}C_{ik}$.
  - Therefore,

$$
\begin{aligned}
\frac{\partial |X|}{\partial X_{ij}} = \frac{\partial \sum_k X_{ik}C_{ik}}{\partial X_{ij}} = C_{ij} &\implies \\
d|X| = C \cdot dX = |X|X^{-T} \cdot dX.&
\end{aligned}
$$

To verify the last result, remember that we said that the coefficient of dX_{ij} in $d|X|$ must be equal to $\frac{\partial |X|}{\partial X_{ij}}$. Indeed,

$$
|X|X^{-T} \cdot e_{ij} = C \cdot e_{ij} = C_{ij} = \frac{\partial |X|}{\partial X_{ij}}.
$$

In the literature, this result is written as

$$
d|X| = |X|\mathrm{tr}(X^{-1}dX).
$$

Note that the two results are equivalent:

$$\begin{aligned}
|X|\mathrm{tr}(X^{-1}dX) &= |X|\sum_i (X^{-1}dX)_{ii} \\
&= |X|\sum_i \sum_k (X^{-1})_{ik} dX_{ki} \\
&= |X|\sum_k \sum_i (X^{-T})_{ki} dX_{ki} \\
&= |X| X^{-T} \cdot dX.
\end{aligned}$$

Part II
Practice

We can finally put into practice what we've learned so far, where that "finally" implies that you've read and understood Part I.

As before, terms in bracket notation will be written in blue. Note that we'll only use bracket notation when necessary.

5 Basic Examples

Unless otherwise specified, x (or X) is always the variable we take the differential with respect to. Remember that $d\cdot$ binds stronger than any other operator.

- $f(x) = a^T x$

$$\begin{aligned}
d(a^T x) &= a^T dx \\
&= a \cdot dx \implies
\end{aligned} \tag{33}$$

$$\nabla_x f = a$$

Note that $a^T dx$ becomes $a \cdot dx$ and not $a^T \cdot dx$. The latter would be a mistake because $a^T = \iota a \in \mathbb{T}_2$, while $dx = dx \in \mathbb{T}_1$. The operator "$\cdot$" (in this simple form) always multiplies tensors of the same dimension. See subsection 2.3.1 for more information.

- $f(x) = x^T A x$

$$\begin{aligned}
d(x^T Ax) &= dx^T Ax + x^T d(Ax) \\
&= dx^T Ax + x^T A dx \\
&= x^T A^T dx + x^T A dx \\
&= (x^T A^T + x^T A) dx \\
&= (x^T (A^T + A)) dx && \text{(same as 33)} \\
&= ((A + A^T)x) \cdot dx \implies
\end{aligned}$$

$$\nabla_x f = (A + A^T)x$$

- $f(X) = a^T X b$

$$\begin{aligned} d(a^T X b) &= a^T dX b \\ &= a \cdot_1 dX \cdot_1 b \\ &= [a\overline{dX_1}][\overline{dX_2}b] \\ &= ab \cdot dX \\ &= (ab^T) \cdot dX \implies \end{aligned}$$

$$\nabla_x f = ab^T$$

Note the simplicity and regularity of the derivation above! For comparison, this is the classic derivation:

$$\begin{aligned} d(a^T X b) &= d\mathrm{tr}(a^T X b) \\ &= \mathrm{tr}(d(a^T X b)) \\ &= \mathrm{tr}(a^T dX b) \\ &= \mathrm{tr}(ba^T dX) \implies \end{aligned}$$

$$Df = ba^T \implies \nabla_X f = ab^T$$

This might not seem more complex than our derivation above, but the problem is that the introduction of the *trace* operator seems an *ad hoc* trick. Also we must accept that the derivative of $\mathrm{tr}(AdX)$ is A. Now we know why, but to the average practitioner this may look like some *magical rule* one needs to remember. Our generalization sidesteps all this "ad-hociness" completely and, above all, never conceals what's going on because bracket notation is quite *explicit*, *general* and yet *easy* to reason with.

- $f(X) = a^T X X^T a$

$$\begin{aligned} d(a^T X X^T a) &= a^T dX X^T a + a^T X dX^T a \\ &= 2a^T dX X^T a \\ &= 2[a\overline{dX_1}][\overline{dX_2}X_2][X_1\overline{a}] \\ &= 2aX_2[X_1\overline{a}] \cdot dX \\ &= 2(a \otimes X^T a) \cdot dX \\ &= 2(aa^T X) \cdot dX \implies \end{aligned}$$

$$\nabla_X f = 2aa^T X$$

First of all, note that $[X_1\overline{a}]$ acts as a scalar, so $aX_2[X_1\overline{a}]$ has just 2 dimensions. I know most of you know this, but someone might have skimmed

over the more technical sections and jumped right to this more practical section (***big mistake***!).
Second, note that if a, b are two column vectors, then $ab = a \otimes b = ab^T$.

- $f(X) = \text{tr}(AX^{-1}B)$
  This time, tr is part of the original function, so we could just deal with it or rewrite it using our operators. Let's follow the second option. To do this, note that $\text{tr}(M) = M \cdot I$:

$$\begin{aligned} d\text{tr}(AX^{-1}B) &= d((AX^{-1}B) \cdot I) \\ &= -(AX^{-1}dXX^{-1}B) \cdot I \\ &= -(UdXV) \cdot I \\ &= -U_1[U_2\overline{dX_1}][\overline{dX_2}V_1]V_2 \cdot I \\ &= -[\overline{I_1}U_1][U_2\overline{dX_1}][\overline{dX_2}V_1][\overline{I_2}V_2] \\ &= -[\overline{I_1}U_1]U_2V_1[\overline{I_2}V_2] \cdot dX \\ &= -U_2[U_1\overline{I_1}][\overline{I_2}V_2]V_1 \cdot dX \\ &= -(U^TIV^T) \cdot dX \\ &= -(U^TV^T) \cdot dX \implies \end{aligned} \tag{34}$$

$$\nabla_X f = -U^TV^T = -X^{-T}A^TB^TX^{-T}$$

Note that we simplified expression 34 by grouping terms whose "*internal interactions*" we don't care about.

- $f(x) = xx^T$
  This one is quite interesting:

$$\begin{aligned} d(xx^T) &= dxx^T + xdx^T \\ &= Idxx^T + xdx^TI \\ &= I_1[I_2\overline{dx}]x + x[\overline{dx}I_1]I_2 \\ &= I_1x[I_2\overline{dx}] + xI_2[\overline{dx}I_1] \\ &= (I_1xI_2 + xI_2I_1) \cdot_1 dx \end{aligned} \tag{35}$$

Note how by multiplying both addends by I we can, in the end, bring dx in the same position in both addends.
Expression 35 can be rewritten as

$$((I \otimes x)_{132} + x \otimes I) \cdot_1 dx.$$

(Note that in our notation $I \otimes x$ is a 3-dimensional tensor, not a matrix.)

It's not difficult to see that this is correct. Indeed,

$$\frac{\partial (xx^T)_{ij}}{\partial x_p} = \frac{\partial x_i x_j}{\partial x_p} = \begin{cases} 2x_i & i = j = p \\ x_j & i = p \neq j \\ x_i & i \neq p = j \\ 0 & i \neq p \neq j \end{cases} \tag{36}$$

which makes sense. Let's rewrite the interesting part of expression 35 abusing notation *a lot* to see the correspondence between terms and indices:

$$I_i x_j I_p + x_i I_j I_p$$

Now it should be clear that the first addend is $x_j \delta_{i=p}$ and the second $x_i \delta_{j=p}$.

6 Deep Learning

I don't think you need to be familiar with *Deep Learning* to understand this section, but you should have a thorough understanding of *backpropagation*. If you need an introduction to it, or just a refresh, you can have a look at my tutorial about *Calculus & Backpropagation*[4].

Note that in this section we don't use any Jacobians. In particular, J is always the *loss function*.

6.1 Single-Layer Fully-Connected Neural Network (Regression)

First, let's consider a very small *single-layer fully-connected neural network* (FCNN) whose equations are:

$$h = f(XW + b^T)$$
$$J(W, b) = \frac{1}{2}||h - Y||^2$$

where:

- X is the *input matrix*, i.e. the matrix whose rows are the *samples* in the *mini-batch*;
- W is the matrix of the *weights* for the single layer;
- b^T is the *row vector* representing the *bias* for the single layer (note the *broadcasting*);
- Y is the *output matrix*, i.e. the matrix whose rows are the "correct predictions" for the corresponding samples;
- f is an *activation function* which is applied *element-wise* to its input.

First of all, let's rewrite the two equations more explicitly:

$$h = f(XW + \mathbf{1}b^T)$$
$$J(W, b) = \frac{1}{2}(h - Y) \cdot (h - Y).$$

Now we can compute the gradient of $J(W, b)$ with respect to the parameters W and b. Let's start from h:

$$\begin{aligned} dh &= f'(XW + \mathbf{1}b^T) \odot d(XW + \mathbf{1}b^T) \\ &= C \odot d(XW + \mathbf{1}b^T) \\ &= C \odot (X dW + \mathbf{1}db^T) \end{aligned}$$

Note that we differentiated h with respect to both W and b at once for brevity.

Now let's consider $J(W, b)$:

$$\begin{aligned} 2dJ &= dh \cdot (h - Y) + (h - Y) \cdot dh \\ &= 2(h - Y) \cdot dh \implies \\ dJ &= (h - Y) \cdot dh \\ &= Z \cdot dh \\ &= Z \cdot (C \odot (X dW)) + Z \cdot (C \odot (\mathbf{1}db^T)) \\ &= [\overline{Z}C(X_1[X_2\overline{dW_1}]dW_2)] + [\overline{Z}C(1db)] \\ &= [X_2\overline{dW_1}][ZC(\overline{X_1 dW_2})] + [ZC(\overline{1db})] \\ &= X_2[ZC(\overline{X_1}1)] \cdot dW + [ZC(\overline{1}1)] \cdot db \\ &= (X^T[ZC(11)]) \cdot dW + ([ZC(11)]^T\mathbf{1}) \cdot db \qquad \text{(see subsec. 2.3.1)} \\ &= (X^T(Z \odot C)) \cdot dW + ((Z \odot C)^T\mathbf{1}) \cdot db \end{aligned}$$

The gradients are thus

$$\nabla_W J(W, b) = X^T(Z \odot C)$$
$$\nabla_b J(W, b) = (Z \odot C)^T\mathbf{1}$$

where

$$Z \odot C = (h - Y) \odot f'(XW + \mathbf{1}b^T)$$

6.2 Multi-Layer Fully-Connected Neural Network (Regression)

A *multi-layer* FCNN with L layers is described by the following equations:

$$\begin{aligned} Z_0 &= X \\ A_j &= Z_{j-1}W_j + \mathbf{1}b_j^T \qquad j = 1, \ldots, L \\ Z_j &= f_j(A_j) \qquad\qquad\quad j = 1, \ldots, L \end{aligned}$$

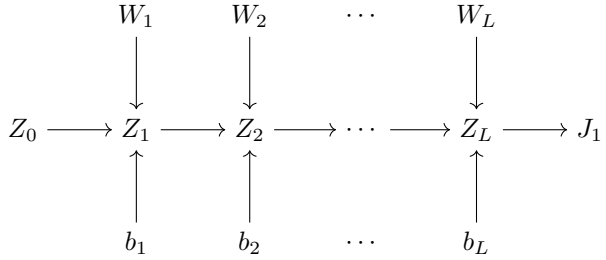
where the matrices are defined similarly to the single-layer case. Let's assume the *mini-batch* has exactly N samples.

In case of *regression* the *loss* is

$$J(W,b) = \frac{1}{2N}(Z_L - Y)\cdot(Z_L - Y) + \frac{1}{2}\lambda\sum_k W_k \cdot W_k,$$

where W and b can be seen as two tensors. This time we added an *L2 regularization* term. Let J_1 be J *without* the regularization term.

The following graph suggests that we should compute $d_{Z_j}J_1$ recursively and then compute $\nabla_{W_j}J_1$ and $\nabla_{b_j}J_1$ from $d_{Z_j}J_1$:



By $d_X f$ we mean df when X is the only variable and the other terms are regarded as constants. Of course, if Y depends on X then we have

$$d_X f = G(dY) = G(H(dX)).$$

Remember that if f is a scalar function and Y depends on X, then

$$d_X f = \nabla_Y f \cdot dY = \nabla_X f \cdot dX.$$

Again, $d_X f$ doesn't mean that df is necessarily written directly as a function of dX.

Let's proceed:

$$\begin{aligned}
d_{Z_L}J_1 &= \frac{1}{2N}(dZ_L\cdot(Z_L-Y) + (Z_L-Y)\cdot dZ_L)\\
&= \frac{1}{N}(Z_L-Y)\cdot dZ_L\\
d_{Z_{j-1}}J_1 &= \nabla_{Z_j}J_1 \cdot dZ_j\\
&= \nabla_{Z_j}J_1 \cdot (f_j'(A_j)\odot dA_j)\\
&= \nabla_{Z_j}J_1 \cdot f_j'(A_j)\odot (dZ_{j-1}W_j)\\
&= P\cdot Q\odot(RS)\\
&= [\overline{P}Q(R_1[\overline{R_2}S_1]S_2)]\\
&= [PQ(\overline{R_1S_2})][\overline{R_2}S_1]\\
&= [PQ(1\overline{S_2})]S_1\cdot R\\
&= ((P\odot Q)S^T)\cdot R
\end{aligned}$$

$$
\begin{aligned}
&= ((\nabla_{Z_j} J_1 \odot f_j'(A_j))W_j^T) \cdot dZ_{j-1} \\
d_{W_j} J_1 &= \nabla_{Z_j} J_1 \cdot dZ_j \\
&= \nabla_{Z_j} J_1 \cdot (f_j'(A_j) \odot dA_j) \\
&= \nabla_{Z_j} J_1 \cdot f_j'(A_j) \odot (Z_{j-1} dW_j) \\
&= P \cdot Q \odot (RS) \\
&= [\overline{P} Q(R_1[R_2 \overline{S_1}] S_2)] \\
&= [R_2 \overline{S_1}][PQ(\overline{R_1 S_2})] \\
&= R_2[PQ(\overline{R_1 1})] \cdot S \\
&= (R^T(P \odot Q)) \cdot S \\
&= (Z_{J-1}^T(\nabla_{Z_j} J_1 \odot f_j'(A_j))) \cdot dW_j \\
d_{b_j} J_1 &= \nabla_{Z_j} J_1 \cdot dZ_j \\
&= \nabla_{Z_j} J_1 \cdot (f_j'(A_j) \odot dA_j) \\
&= \nabla_{Z_j} J_1 \cdot f_j'(A_j) \odot (\mathbf{1} db_j^T) \\
&= P \cdot Q \odot (\mathbf{1} d^T) \\
&= [PQ(\overline{1} d)] \\
&= [PQ(\overline{1} 1)] \cdot d \\
&= ((P \odot Q)^T \mathbf{1}) \cdot d \\
&= ((\nabla_{Z_j} J_1 \odot f_j'(A_j))^T \mathbf{1}) \cdot db_j
\end{aligned}
$$

Here are the final formulas:

$$
\begin{aligned}
\nabla_{Z_L} J_1 &= \frac{1}{N}(Z_L - Y) \\
\nabla_{Z_j} J_1 &= S_{j+1} W_{j+1}^T \qquad && j = 1, \ldots, L-1 \\
\nabla_{W_j} J_1 &= Z_{j-1}^T S_j \qquad && j = 1, \ldots, L \\
\nabla_{b_j} J_1 &= S_j^T \mathbf{1}, \qquad && j = 1, \ldots, L
\end{aligned}
$$

where

$$
S_j = \nabla_{Z_j} J_1 \odot f_j'(A_j) \qquad j = 1, \ldots, L
$$

Almost forgot:

$$
\begin{aligned}
d_{W_j} J &= d_{W_j} J_1 + d_{W_j} \left(\frac{1}{2} \lambda \sum_k W_k \cdot W_k \right) \\
&= d_{W_j} J_1 + \frac{1}{2} \lambda \sum_k d_{W_j} (W_k \cdot W_k) \\
&= d_{W_j} J_1 + \frac{1}{2} \lambda (dW_j \cdot W_j + W_j \cdot dW_j) \\
&= \nabla_{W_j} J_1 \cdot dW_j + \lambda W_j \cdot dW_j
\end{aligned}
$$

$$= (\nabla_{W_j} J_1 + \lambda W_j) \cdot dW_j,$$

so the final equations are:

$$\begin{aligned}
\nabla_{Z_L} J &= \frac{1}{N}(Z_L - Y) \\
\nabla_{Z_j} J &= S_{j+1} W_{j+1}^T && j = 1, \ldots, L-1 \\
\nabla_{W_j} J &= Z_{j-1}^T S_j + \lambda W_j && j = 1, \ldots, L \\
\nabla_{b_j} J &= S_j^T \mathbf{1}, && j = 1, \ldots, L
\end{aligned}$$

where

$$S_j = \nabla_{Z_j} J \odot f_j'(A_j) \qquad j = 1, \ldots, L$$

Note that only the gradients with respect to W_j can be influenced by the regularization.

6.3 Multi-Layer Fully-Connected Neural Network (Classification)

This case is similar to the previous one, but we'll substitute the activation of the last layer (i.e. layer L) with the *softmax function*. For the loss, we'll use the *Negative Log Likelihood*, i.e. the *Cross Entropy*.

For convenience, we'll assume that Y is a matrix whose rows are *one-hot* vectors specifying the *labels* for the corresponding *samples* in the *mini-batch*.

The previous section's equations for Z_j and A_j apply to this section too, with the exception of the one for Z_L:

$$Z_L = s^1(A_L), \tag{37}$$

where

$$s(a) = \frac{e^a}{\mathbf{1}^T e^a}.$$

Note that s, the softmax function, is applied to each row of A_L.

The equation for the loss becomes

$$J(W, b) = -\frac{1}{N} \ln(Z_L) \cdot Y + \sum_k W_k \cdot W_k.$$

Let's compute $d_{W_L, b_L, Z_{L-1}} J_1$, where "$\oslash$" is the element-wise division:

$$\begin{aligned}
dJ_1 &= -\frac{1}{N} d(\ln(Z_L) \cdot Y) \\
&= -\frac{1}{N} (d \ln(Z_L)) \cdot Y \\
&= -\frac{1}{N} ((1 \oslash Z_L) \odot dZ_L) \cdot Y
\end{aligned}$$

$$
\begin{aligned}
&= -\frac{1}{N}((1 \oslash Z_L) \odot (\nabla s^1(A_L)\ {}^{1\overline{2}}\odot^{12} dA_L)) \cdot Y \\
&= (P \odot Q) \cdot Y = [PQ\overline{Y}] = [PY] \cdot Q = (P \odot Y) \cdot Q \\
&= -\frac{1}{N}((1 \oslash Z_L) \odot Y) \cdot (\nabla s^1(A_L)\ {}^{1\overline{2}}\odot^{12} dA_L) \\
&= -\frac{1}{N}(Y \oslash Z_L) \cdot (\nabla s^1(A_L)\ {}^{1\overline{2}}\odot^{12} dA_L) \\
&= -\frac{1}{N}(Y \oslash Z_L) \cdot (\nabla s^1(A_L)\ {}^{1\overline{2}}\odot^{12} d(Z_{L-1}W_L + \mathbf{1}b_L^T)) \\
&= B \cdot (C\ {}^{1\overline{2}}\odot^{12} D) && (C \in \mathbb{T}_3) \\
&= B \cdot [C_1 D_1][C_2 \overline{D_2}]C_3 \\
&= [B_1 C_1 \overline{D_1}][C_2 \overline{D_2}][\overline{B_2} C_3] \\
&= [B_1 C_1]C_2[\overline{B_2} C_3] \cdot D \\
&= (B\ {}^{1\overline{2}}\odot^{13} C) \cdot D \\
&= E \cdot d(Z_{L-1}W_L + \mathbf{1}b_L^T) && (38) \\
&= E \cdot (dZ_{L-1}W_L + Z_{L-1}dW_L + \mathbf{1}db_L^T) \\
&= E \cdot (dZ_{L-1}W_L) + E \cdot (Z_{L-1}dW_L) + E \cdot (\mathbf{1}db_L^T) \\
&= E \cdot (dZW) + E \cdot (ZdW) + E \cdot (\mathbf{1}db^T) \\
&= E \cdot dZ_1[\overline{dZ_2}W_1]W_2 + E \cdot Z_1[Z_2\overline{dW_1}]dW_2 + E \cdot [1db] \\
&= [E_1\overline{dZ_1}][\overline{dZ_2}W_1][\overline{E_2}W_2] + [\overline{E_1}Z_1][Z_2\overline{dW_1}][E_2\overline{dW_2}] + [\overline{E_1}1][E_2\overline{db}] \\
&= E_1 W_1[\overline{E_2}W_2] \cdot dZ_{L-1} + [\overline{E_1}Z_1]Z_2E_2 \cdot dW_L + [\overline{E_1}1]E_2 \cdot db_L \\
&= E_1[\overline{E_2}W_2]W_1 \cdot dZ_{L-1} + Z_2[Z_1\overline{E_1}]E_2 \cdot dW_L + E_2[\overline{E_1}1] \cdot db_L \\
&= (EW_L^T) \cdot dZ_{L-1} + (Z_{L-1}^T E) \cdot dW_L + (E^T\mathbf{1}) \cdot db_L && (39)
\end{aligned}
$$

The final formulas are thus

$$
\begin{aligned}
\nabla_{W_L} J &= Z_{L-1}^T E + \lambda W_L \\
\nabla_{b_L} J &= E^T \mathbf{1} \\
\nabla_{Z_{L-1}} J &= E W_L^T \\
\nabla_{Z_j} J &= S_{j+1} W_{j+1}^T && j = 1, \ldots, L-2 \\
\nabla_{W_j} J &= Z_{j-1}^T S_j + \lambda W_j && j = 1, \ldots, L-1 \\
\nabla_{b_j} J_1 &= S_j^T \mathbf{1}, && j = 1, \ldots, L-1
\end{aligned}
$$

where

$$
\begin{aligned}
E &= -\frac{1}{N}(Y \oslash Z_L)\ {}^{1\overline{2}}\odot^{13} \nabla s^1(A_L) && (40) \\
S_j &= \nabla_{Z_j} J \odot f_j'(A_j) && j = 1, \ldots, L-1
\end{aligned}
$$

We've already computed ∇s in subsection 4.3.5.

6.3.1 An easier way

This is one of the cases where generalization makes things harder rather than simpler. Since we know s and s is applied to each row of A_L, we can simplify things by rewriting J_1 as follows:

$$J_1 = -\frac{1}{N}\sum_i \ln(s(a_i)) \cdot y_i \tag{41}$$

where a_i^T and y_i^T are the i-th row of A_L and Y, respectively. We can first compute $d_{a_i} J_1$ and then determine $d_{A_L} J_1$. Let's proceed:

$$\begin{aligned}
dJ_1 &= -\frac{1}{N}\sum_i d\ln\left(\frac{e^{a_i}}{\mathbf{1}^T e^{a_i}}\right)\cdot y_i \\
&= -\frac{1}{N}\sum_i d(a_i \ominus \ln(\mathbf{1}^T e^{a_i}))\cdot y_i \\
&= -\frac{1}{N}\sum_i d(a_i - \mathbf{1}\ln(\mathbf{1}^T e^{a_i}))\cdot y_i \\
&= -\frac{1}{N}\sum_i (da_i - \frac{\mathbf{1}}{\mathbf{1}^T e^{a_i}}\mathbf{1}^T(e^{a_i}\odot da_i))\cdot y_i \\
&= -\frac{1}{N}\sum_i (da_i - \mathbf{1}\mathbf{1}^T(s(a_i)\odot da_i))\cdot y_i \\
&= -\frac{1}{N}\sum_i (da_i - \mathbf{1}(s(a_i)\cdot da_i))\cdot y_i \\
&= -\frac{1}{N}\sum_i (da_i\cdot y_i - s(a_i)\cdot da_i) \qquad (\sum_j y_{ij} = 1) \\
&= -\frac{1}{N}\sum_i (y_i - s(a_i))\cdot da_i \\
&= -\frac{1}{N}\sum_i [Y - s^1(A_L)]_{i\cdot}\cdot [dA_L]_{i\cdot} \\
&= -\frac{1}{N}\sum_i [(Y - s^1(A_L))\,{}^{1\overline{2}}\odot^{12} dA_L]_i \qquad (42) \\
&= -\frac{1}{N}(Y - Z_L)\cdot dA_L \qquad (43) \\
&= E\cdot dA_L \qquad (44)
\end{aligned}$$

To understand expression 42, note that, in general, $U\,{}^{1\overline{2}}\odot^{12} V$ is the column vector whose i-th "row" is the dot product between the i-th rows of U and V. Intuitively, we walk along the two dimensions of U and V in parallel and multiply element-wise, but, in addition, we *sum* along the second dimension (i.e. along the rows).

Now note that expression 44 is equivalent to expression 38, so we ended up just simplifying (*considerably*!) E.

Let's try to verify that the two E are indeed equal, i.e. that

$$-\frac{1}{N}(Y - Z_L) = -\frac{1}{N}(Y \oslash Z_L)\,{}^{1\overline{2}}\odot^{13}\nabla s^1(A_L).$$

We can simplify things by noting that $dA_L = \sum_i e_i da_i^T$, where a_i^T is the i-th row of A_L. For convenience, we'll surround "classic notation" terms inside bracket notation terms with *curly braces*. Let's proceed:

$$\begin{aligned}
&((Y \oslash Z_L)\,{}^{1\overline{2}}\odot^{13}\nabla s^1(A_L)) \cdot dA_L = \\
&((Y \oslash Z_L)\,{}^{1\overline{2}}\odot^{13}\nabla s^1(A_L)) \cdot \sum_i e_i da_i^T = \\
&\sum_i ((Y \oslash Z_L)\,{}^{1\overline{2}}\odot^{13}\nabla s^1(A_L)) \cdot (e_i da_i^T) = \\
&\sum_i (B\,{}^{1\overline{2}}\odot^{13} C) \cdot e_i da_i^T = \\
&\sum_i (e_i \cdot_1 (B\,{}^{1\overline{2}}\odot^{13} C)) \cdot da_i = \\
&\sum_i (B_{i\cdot}\,{}^{\overline{1}}\odot^{2} C_{i\cdot\cdot}) \cdot da_i = && (C \in \mathbb{T}_3) \\
&\sum_i (C_{i\cdot\cdot} B_{i\cdot}) \cdot da_i = \\
&\sum_i ([\nabla s^1(A_L)]_{i\cdot\cdot} [Y \oslash Z_L]_{i\cdot}) \cdot da_i = \\
&\sum_i (\nabla s(a_i)(y_i \oslash s(a_i))) \cdot da_i = \\
&\sum_i ([s\mathbf{1}^T \odot I - ss^T](y_i \oslash s)) \cdot da_i = && \text{(see subsec. 4.3.5)} \\
&\sum_i (([sI_1]I_2 - ss)(y_i \oslash s)) \cdot da_i = \\
&\sum_i ([sI_1][\overline{I_2}\{y_i \oslash s\}] - s[\overline{s}\{y_i \oslash s\}]) \cdot da_i = \\
&\sum_i ([sI_1][\overline{I_2}\{y_i \oslash s\}] - s[\overline{1} y_i]) \cdot da_i = \\
&\sum_i (s \odot I_1[\overline{I_2}\{y_i \oslash s\}] - s\mathbf{1}^T y_i) \cdot da_i = \\
&\sum_i (s \odot (I(y_i \oslash s)) - s) \cdot da_i = && (\sum_j y_{ij} = 1) \\
&\sum_i (y_i - s(a_i)) \cdot da_i =
\end{aligned}$$

$$\sum_i [Y - s^1(A_L)]_{i\cdot} \cdot [dA_L]_{i\cdot} =$$
$$\sum_i [(Y - s^1(A_L))\, {}^{1\bar{2}}\odot {}^{12} dA_L]_i =$$
$$(Y - Z_L) \cdot dA_L$$

6.4 Recurrent Neural Network (RNN)

In this subsection we'll consider an RNN described by the following equations:

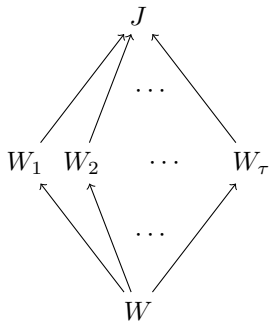
$$\begin{aligned}
A_t &= Z_{t-1}W + X_t U + \mathbf{1}b^T && t = 1, \ldots, \tau \\
Z_t &= \tanh(A_t) && t = 1, \ldots, \tau \\
O_t &= Z_t V + \mathbf{1}c^T && t = 1, \ldots, \tau \\
\hat{Y}_t &= s^1(O_t) && t = 1, \ldots, \tau \\
J_t &= -\frac{1}{N} \ln(\hat{Y}_t) \cdot Y_t && t = 1, \ldots, \tau \\
J &= \sum_{t=1}^{\tau} J_t
\end{aligned}$$

Z_0 is an initial value and, as before, we assume that the mini-batch has size N. This means that $X_1, \ldots, X_\tau$ represents N sequences. In particular, the i-th sequence is $X_{1i\cdot}, \ldots, X_{\tau i\cdot}$.

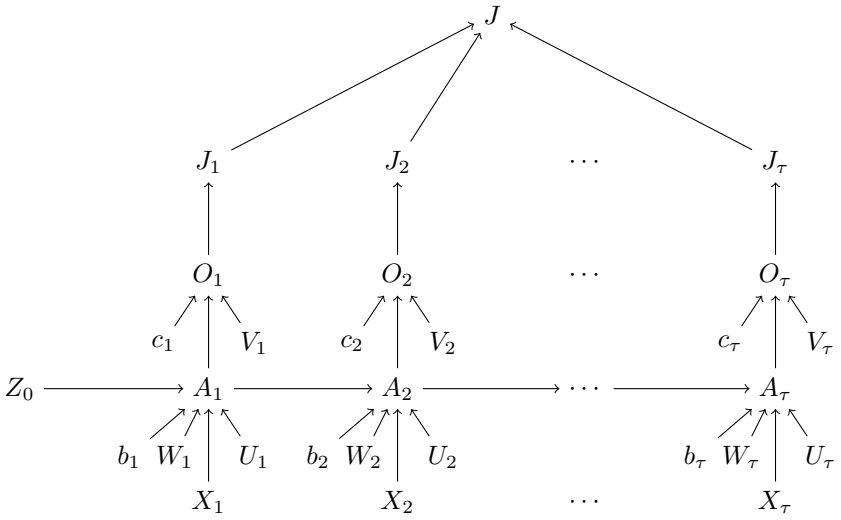
To simplify things, we'll rewrite J_t as in subsection 6.3.1 and we'll pretend that we're dealing with different parameters at each time step:

$$\begin{aligned}
A_t &= Z_{t-1}W_t + X_t U_t + \mathbf{1}b_t^T && t = 1, \ldots, \tau \\
Z_t &= \tanh(A_t) && t = 1, \ldots, \tau \\
O_t &= Z_t V_t + \mathbf{1}c_t^T && t = 1, \ldots, \tau \\
J_t &= -\frac{1}{N} \sum_i \ln(s(O_{ti\cdot})) \cdot Y_{ti\cdot} && t = 1, \ldots, \tau \\
J &= \sum_{t=1}^{\tau} J_t && (45)
\end{aligned}$$

Then, it's a matter of realizing that, for instance, $\nabla_W J = \sum_{t=1}^{\tau} \nabla_{W_t} J$. To see this, consider the following *backpropagation graph*, where $W_t = \mathrm{id}(W) = W$:



To see how we should compute the gradients of our parameters, let's have a look at the computation graph for the RNN:



Note that we omitted Z_t (except for Z_0) in the graph, which is equivalent to replacing Z_t with its definition ($\tanh(A_t)$) in every other definition.

To avoid unnecessary computations, we'll compute $\nabla_{A_t} J$ from $\nabla_{A_{t+1}} J$ and $\nabla_{O_t} J$. Then computing the gradients with respect to the parameters won't be a problem. Let's do it:

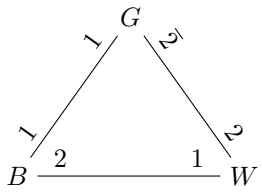
$$d_{A_t} J = \nabla_{A_{t+1}} J \cdot dA_{t+1} + \nabla_{O_t} J \cdot dO_t \tag{46}$$

Before proceeding, note that equation 46 is exactly what we'd find if we computed $d_{A_t} J$ starting from equation 45, ignoring irrelevant variables and stopping at dA_{t+1} and dO_t.

We can consider the two addends of equation 46 separately. Let's write $J^{(1)}$ to indicate that we're only considering the first term of equation 46:

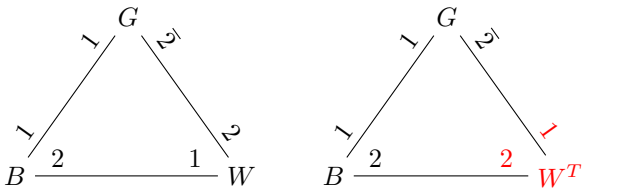
$$
\begin{aligned}
d_{A_t} J^{(1)} &= \nabla_{A_{t+1}} J \cdot dA_{t+1} \\
&= G \cdot dA_{t+1} \\
&= G \cdot d(Z_t W_{t+1} + X_{t+1} U_{t+1} + \mathbf{1} b_{t+1}^T) \\
&= G \cdot (dZ_t W_{t+1}) \qquad (47) \\
&= G \cdot (d \tanh(A_t) W_{t+1}) \\
&= G \cdot ((\tanh'(A_t) \odot dA_t) W_{t+1}) \\
&= G \cdot (((\mathbf{1}\mathbf{1}^T - \tanh^2(A_t)) \odot dA_t) W_{t+1}) \\
&= G \cdot ((B \odot D) W) \\
&= G \cdot [B_1 D_1][B_2 \overline{D_2} W_1] W_2 \\
&= [G_1 B_1 \overline{D_1}][B_2 \overline{D_2} W_1][\overline{G_2} W_2] \\
&= [G_1 B_1][B_2 W_1][\overline{G_2} W_2] \cdot D \qquad (48)
\end{aligned}
$$

The bracket notation term in expression 48 may seem a little tricky, so maybe a graph will help:



Edges indicate the product of dimensions and the labels indicate which dimensions are being multiplied and, optionally, summed over (each label refers to the nearest node). Note that there's an information missing in the graph above: the fact that $[G_1 B_1]$ comes before $[B_2 W_1]$. I imagine we could come up with some smart way to represent this in the graph (maybe with arrows between edges?) and even define a full-fledged *graphical tensor differential calculus*, but that's for another time!

We can find a neat expression for our bracket term by manipulating the graph above:



In retrospect, we should have seen it, but I thought this was a good opportunity to see how bracket notation might look like in graphical form. Indeed,

we can take B out right from the start:

$$
\begin{aligned}
&[G_1 B_1][B_2 W_1][\overline{G_2} W_2] = \\
&B \odot G_1 W_1 [\overline{G_2} W_2] = \\
&B \odot G_1 [\overline{G_2} W_2] W_1 = \\
&B \odot (G W^T)
\end{aligned}
$$

Now we can continue from where we left off:

$$
\begin{aligned}
d_{A_t} J^{(1)} &= \nabla_{A_{t+1}} J \cdot dA_{t+1} \\
&= G \cdot dA_{t+1} \\
&= \cdots \\
&= G \cdot (((\mathbf{1}\mathbf{1}^T - \tanh^2(A_t)) \odot dA_t) W_{t+1}) \\
&= G \cdot ((B \odot D) W) \\
&= \cdots \\
&= [G_1 B_1][B_2 W_1][\overline{G_2} W_2] \cdot D \\
&= (B \odot (G W^T)) \cdot D \\
&= ((\mathbf{1}\mathbf{1}^T - \tanh^2(A_t)) \odot (\nabla_{A_{t+1}} J W_{t+1}^T)) \cdot dA_t
\end{aligned}
$$

Let's do the same with the second addend of equation 46:

$$
\begin{aligned}
d_{A_t} J^{(2)} &= \nabla_{O_t} J \cdot dO_t \\
&= H \cdot dO_t \\
&= H \cdot d(Z_t V_t + \mathbf{1} c_t^T) \\
&= H \cdot (dZ_t V_t)
\end{aligned} \tag{49}
$$

We stopped because expression 49 has the same form of expression 47, so we can conclude that

$$
\begin{aligned}
d_{A_t} J^{(2)} &= (B \odot (H V^T)) \cdot D \\
&= ((\mathbf{1}\mathbf{1}^T - \tanh^2(A_t)) \odot (\nabla_{O_t} J V_t^T)) \cdot dA_t
\end{aligned}
$$

In conclusion, we get

$$
\begin{aligned}
d_{A_t} J &= [B \odot (G W^T) + B \odot (H V^T)] \cdot D \\
&= [B \odot (G W^T + H V^T)] \cdot D \\
&= [(\mathbf{1}\mathbf{1}^T - \tanh^2(A_t)) \odot (\nabla_{A_{t+1}} J W_{t+1}^T + \nabla_{O_t} J V_t^T)] \cdot dA_t
\end{aligned}
$$

Now let's determine $d_{O_t} J$:

$$
d_{O_t} J = d_{O_t} \sum_{t=1}^{\tau} J_t = d_{O_t} J_t
$$

Note that J_t has the same form of J_1 as by definition 41:

$$J_t = -\frac{1}{N}\sum_i \ln(s(O_{ti\cdot})) \cdot Y_{ti\cdot}$$
$$J_1 = -\frac{1}{N}\sum_i \ln(s(a_i)) \cdot y_i$$

where O_t and Y_t correspond to A_L and Y, respectively. Remember that a_i^T and y_i^T are the i-th rows of A_L and Y, respectively.

This means that from

$$\begin{aligned} dJ_1 &= -\frac{1}{N}(Y - Z_L) \cdot dA_L && \text{(see expr. 43)} \\ &= -\frac{1}{N}(Y - s^1(A_L)) \cdot dA_L && \text{(for def. 37)} \end{aligned}$$

we get

$$d_{O_t} J = d_{O_t} J_t = -\frac{1}{N}(Y_t - s^1(O_t)) \cdot dO_t$$

All is left to do is to compute the gradients of the parameters:

$$\begin{aligned} d_{W_t,U_t,b_t} J &= \nabla_{A_t} J \cdot dA_t \\ &= G \cdot dA_t \\ &= G \cdot d(Z_{t-1}W_t + X_tU_t + \mathbf{1}b_t^T) \\ &= G \cdot (Z_{t-1}dW_t) + G \cdot (X_t dU_t) + G \cdot (\mathbf{1}db_t^T) \\ d_{V_t,c_t} J &= \nabla_{O_t} J \cdot dO_t \\ &= H \cdot dO_t \\ &= H \cdot d(Z_tV_t + \mathbf{1}c_t^T) \\ &= H \cdot (Z_t dV_t) + H \cdot (\mathbf{1}dc_t^T) \end{aligned}$$

We have already done these derivations before (with different terms), but let's do them again for the exercise. Here's the derivation for the first form:

$$\begin{aligned} &A \cdot (BD) = \\ &A \cdot B_1[B_2\overline{D_1}]D_2 = \\ &[\overline{A_1}B_1][B_2\overline{D_1}][A_2\overline{D_2}] = \\ &[\overline{A_1}B_1]B_2A_2 \cdot D = \\ &B_2[B_1\overline{A_1}]A_2 \cdot D = \\ &(B^TA) \cdot D \end{aligned}$$

And here's the derivation for the second form:

$$
\begin{aligned}
&A \cdot (\mathbf{1}d^T) = \\
&A \cdot 1d = \\
&[\overline{A_1}1][A_2\overline{d}] = \\
&\overline{A_1}[A_2\overline{d}] = \\
&\overline{A_1}A_2 \cdot d = \\
&A_2[\overline{A_1}1] \cdot d = \\
&(A^T\mathbf{1}) \cdot d
\end{aligned}
$$

I'm being very thorough here for didactic reasons, but feel free to skip any steps you consider unnecessary in your derivations.

Now we can complete our derivations:

$$
\begin{aligned}
d_{W_t,U_t,b_t}J &= (Z_{t-1}^T \nabla_{A_t} J) \cdot dW_t + (X_t^T \nabla_{A_t} J) \cdot dU_t + (\nabla_{A_t} J^T \mathbf{1}) \cdot db_t \\
d_{V_t,c_t}J &= (Z_t^T \nabla_{O_t} J) \cdot dV_t + (\nabla_{O_t} J^T \mathbf{1}) \cdot dc_t
\end{aligned}
$$

The final equations are thus:

$$
\begin{aligned}
&\nabla_{A_{\tau+1}} J = 0 \\
&\nabla_{O_t} J = -\frac{1}{N}(Y_t - s^1(O_t)) && t = 1, \ldots, \tau \\
&\nabla_{A_t} J = (\mathbf{1}\mathbf{1}^T - \tanh^2(A_t)) \odot (\nabla_{A_{t+1}} J W_{t+1}^T + \nabla_{O_t} J V_t^T) && t = 1, \ldots, \tau \\
&\nabla_W J = \sum_{t=1}^{\tau} Z_{t-1}^T \nabla_{A_t} J \\
&\nabla_U J = \sum_{t=1}^{\tau} X_t^T \nabla_{A_t} J \\
&\nabla_b J = \sum_{t=1}^{\tau} \nabla_{A_t} J^T \mathbf{1} \\
&\nabla_V J = \sum_{t=1}^{\tau} Z_t^T \nabla_{O_t} J \\
&\nabla_c J = \sum_{t=1}^{\tau} \nabla_{O_t} J^T \mathbf{1}
\end{aligned}
$$

References

[1] Jan R. Magnus and Heinz Neudecker. *Matrix Differential Calculus with Applications in Statistics and Econometrics, 2nd Edition.* Wiley, 2 edition, March 1999. ISBN 047198633X. URL `http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20&path=ASIN/047198633X`.

[2] David Pollock. On kronecker products, tensor products and matrix differential calculus. Discussion Papers in Economics 14/02, Department of Economics, University of Leicester, 2014. URL http://EconPapers.repec.org/RePEc:lec:leecon:14/02.

[3] Massimiliano Tomassoli. Matrix calculus. 2013. URL https://github.com/mtomassoli/papers/blob/master/matrixcalculus.pdf.

[4] Massimiliano Tomassoli. Calculus and backpropagation. 2016. URL https://github.com/mtomassoli/papers/blob/master/backprop.pdf.

[5] Wikipedia. Einstein notation — wikipedia, the free encyclopedia, 2016. URL https://en.wikipedia.org/w/index.php?title=Einstein_notation&oldid=701542869. [Online; accessed 12-August-2016].

[6] Wikipedia. Hadamard product (matrices) — wikipedia, the free encyclopedia, 2016. URL https://en.wikipedia.org/w/index.php?title=Hadamard_product_(matrices)&oldid=730570149. [Online; accessed 12-August-2016].