Tensor Differentiation

Ran Wang

January 26, 2021

Chapter 1

Tensor and Differentials

1.1 Basics

We use lower case letters, such as a, b, c to denote the vectors/scalers. It should be clear from the contexts a letter refers to vector or scaler. For matrices, we use upper-case letters, such as A, B, C. In addition, we sometimes make use of "tensors", which are nothing but high-dimensional arrays. More specifically, we call vectors 1-d arrays, matrices 2-d arrays, and tensors 3-d or 4-d arrays. We also use upper-case letters to represent tensors.

Given a vector a, we call it N-dimensional, if it contains exactly N elements, and we use $a_i$to denote its i'th element, $1 \leq i \leq N$. Similarly, for a matrix A, we call it $M \times N$ dimensional, if it contains M rows and N columns, and we use A_{ij} or $A_{i,j}$to denote its (i, j)'th element, $1 \leq i \leq M, 1 \leq j \leq N$. We use similar notations for tensors.

We follow the conventions in literature when it comes to the sum/difference of vectors/matrices. Similarly, we follow the conventions in the literature in representing the product between scalers/vectors/matrices. We omit the details here, and we refer the readers to Schott (2016).

In addition, we also need the following vector/matrix operations.

1.1.1 Vectorization of matrices

Let A be an $M \times N$ matrix. We use $\text{vec}(A)$ to denote the vector obtained by stacking the columns of A. More specifically, let a_i denote the i'th column of A, then $\text{vec}(A) = [a_1^t, a_2^t, \ldots, a_n^t]^t$.

1.1.2 Inner product

Let a, b be N-dimensional vector. Their inner product $a \cdot b := a^t b = \sum_{i=1}^N a_i b_i$. Similarly, let A, B be $M \times N$-dimensional matrices, then their inner product $A \cdot B := \sum_{i=1}^M \sum_{j=1}^N A_{ij} B_{ij}$.

We have the following identity:

$$A \cdot (BC) = (B^t A) \cdot C = (AC^t) \cdot B.$$

1.1.3 Hadamard product

Let A, B be $M \times N$-dimensional matrices. Their Hadamard product, $A \bigodot B$ is defined as an $M \times N$-dimensional matrix which satisfies,

$$(A \bigodot B)_{ij} = A_{ij} B_{ij}.$$

We have the following identity

$$A \cdot (B \bigodot C) = (A \bigodot B) \cdot C.$$

1.2 Vector/matrix calculus

In this section, we derive the identities of vector/matrix calculus.

1.2.1 Directives

Let Q be an $R^M \to R$ function, and we use $Q'(x)$ or $\frac{\partial Q}{\partial x}$ to denote the following M-dimensional vector,

$$Q'(x)_i = \frac{\partial Q}{\partial x_i}.$$

Similarly, let Q be a $R^{M \times N} \to R$ functional, we use $Q'(X)$ or $\frac{\partial Q}{\partial X}$to denote the following $M \times N$-dimensional matrix,

$$Q'(X)_{ij} = \frac{\partial Q}{\partial X_{ij}}.$$

Now let Q be a function from $R^M \to R^N$, s.t.,

$$Q(x) = \begin{bmatrix} Q_1(x) \\ Q_2(x) \\ \vdots \\ Q_N(x) \end{bmatrix}.$$

Then its Jacobian $\nabla Q(x)$ is defined as an $M \times N$-dimensional matrix that satisfies,

$$\nabla f(x)_{ij} = \frac{\partial Q_i(x)}{\partial x_j}.$$

For functions that maps matrices/tensors to matrices/tensors, we define the derivatives as the corresponding vectorized version. To illustrate, let Q be a function that maps $R^{M_1 \times N_1}$ to $R^{M_2 \times N_2}$, then its derivatives are defined as $\frac{\partial \mathrm{vec}(Q)}{\partial \mathrm{vec}^T(X)}$.

We have the following formulae,

$$\frac{\partial \mathrm{vec} Q(F(X))}{\partial \mathrm{vec}^t(X)} = \frac{\partial \mathrm{vec}(Q(F(X))}{\partial \mathrm{vec}^t(F(X))} \frac{\partial \mathrm{vec}(F(X))}{\partial \mathrm{vec}^t(X)},$$

and

$$\frac{\partial Mx}{\partial x^t} = M.$$

1.2.2 Differentials

Let Q be a function that maps R^M to R^N, and assume its derivative exists. It can be proved that there exists a matrix A_x s.t.,

$$Q(x + dx) = Q(x) + A_x dx + o(dx).$$

In the display above, we use $o(dx)$ to denote an N-dimensional vector that satisfies,

$$\lim_{\|dx\| \downarrow 0} \frac{o(dx)}{\|dx\|} = 0.$$

We denote the differential of Q, $dQ(x; dx) : R^M \to R^N$ s.t. $dQ(x; dx) = A_x dx$. We also write $dQ_x.dx$ in place of $dQ(x; dx)$.

Similarly, let Q be a function that maps $R^{M_1 \times N_1}$ to $R^{M_2 \times N_2}$, and assume it is element-wise differentiable. Then there exists matrix A_X s.t.

$$\mathrm{vec}(Q(X + dX)) = \mathrm{vec}(Q(X)) + A_X \mathrm{vec}(dX) + o(dX).$$

We define the differential of Q, $dQ(X; dX)$ to be an $M_2 \times N_2$ matrix s.t.

$$\mathrm{vec}(dQ(X; dX)) = A_X \mathrm{vec}(dX).$$

We also use $dQ_X.dX$ to denote $dQ(X; dX)$.

The derivatives and differentials are related through the following implications:

$$dQ_X.dX = M \cdot dX$$

is equivalent to

$$\frac{\partial Q(X)}{\partial X} = M$$

We have the following chain rules for differentials,

$$dQ_X.dX = dG_F.dF_X.dX.$$

In view of this we write can omit dX and the subscript to write

$$dQ = dG \cdot dF$$

Finally, we have the following identities,

$$\begin{aligned} d(\alpha X + Y) &= \alpha dX + dY \\ d(XY) &= dXY + XdY \\ d(X \cdot Y) &= dX \cdot Y + X \cdot dY \\ d(X \bigodot Y) &= dX \bigodot Y + X \bigodot dY \end{aligned}$$

1.3 Applications

Artificial neural networks are nothing but stacked models. More specifically, assume we have N samples, with the input information for sample i being X_i, which is either a vector, a matrix or a tensor. In addition, we assume the output for this sample is $\hat{Y}_i$, which is either a vector, matrix or a tensor, then X_i and Y_i are linked through a series of functions, commonly dubbed as "layers", such that

$$\hat{Y}_i = F_n^{\theta_n} \circ F_{n-1}^{\theta_{n-1}} \circ \ldots \circ F_1^{\theta_1}(X_i)$$

where $F_1, F_2, \ldots, F_n$are functions, with parameters $\theta_1, \theta_2, \ldots, \theta_n$, to be estimated empirically. Commonly we denote $Z_i^0 = X_i$, $Z_i^j = F_{j-1}^{\theta_{j-1}}(Z_i^{j-1})$, $1 \leq j \leq n$, and we say the input for layer j is $Z_i^{j-1}$and the output for layer j is Z_i^j. It is often customary to omit the subscript i as long as no confusion arises.

The goal of "training" a neural network, is to estimate $\theta_1, \theta_2, \ldots, \theta_n$ such that the following quantity is minimized,

$$L(Y, \hat{Y}) = \frac{1}{N} \sum_{i=1}^{N} L_i(Y_i, \hat{Y}_i).$$

where $L(\cdot,\cdot)$is a loss function, and Y_i is the actual value of the dependent variable.

There is typically no closed-form solutions to the problems above. Instead, iterative methods based on gradients are often used. To fully specify this methods, it is enough to,

1. Specify how to compute the gradient, i.e. $\frac{\partial \text{vec}(L)}{\partial \text{vec}^t(\theta_j)}$, $1 \leq j \leq n$.

2. Specify how to update the parameters once the gradient is obtained.

For the previous point, note that since $L_i(Y_i, F_n^{\theta_n} \circ F_{n-1}^{\theta_{n-1}} \circ \ldots \circ F_1^{\theta_1}(X_i)) = L(Y_i, F_n^{\theta_n}(Z_i^{n-1}))$, we can easily calculate $\frac{\partial L}{\partial \text{vec}^t(\theta_n)}$ and $\frac{\partial L_i}{\partial \text{vec}^t(Z_i^{n-1})}$. Now apply the chain rule, we have,

$$\frac{\partial L_i}{\partial \text{vec}^t(\theta_{n-1})} = \frac{\partial L_i}{\partial \text{vec}^t(Z_i^n)} \frac{\partial \text{vec}(Z_i^n)}{\partial \text{vec}^t(\theta_{n-1})},$$

and

$$\frac{\partial L_i}{\partial \text{vec}^t(Z_i^{n-1})} = \frac{\partial L_i}{\partial \text{vec}^t(Z_i^n)} \frac{\partial \text{vec}(Z_i^n)}{\partial \text{vec}^t(Z_i^{n-1})}.$$

Since $Z_i^n = F_{n-1}^{\theta_{n-1}}(Z_i^{n-1})$,$\frac{\partial \text{vec}(Z_i^n)}{\partial \text{vec}^t(Z_i^{n-1})}$ can be derived. Continue with layer $n-2$ till 1, we obtain the gradients for all layers. We will return to this point in later chapters when we introduce the layers.

Now assume the gradient has been obtained, let us specify how to update the parameters. There are several popular strategies, mostly based on stochastic gradient descent. It is essential the steepest descent algorithm, such that in each iteration,

$$\theta_j \leftarrow \theta_j - \eta \frac{\hat{\partial L}}{\partial \theta_j}, 1 \leq j \leq n$$

where $\frac{\hat{\partial L}}{\partial \theta_j}$ is computed not from the whole sample, but rather from a small batch, typically consists of 16 to 64 samples. There are some problems with this algorithm, including,

1. The estimates of the gradients might be unstable.

2. The algorithms might stop at local minima or saddle point.

3. In the ill-conditioned problem, where the object functions react differently to changes in different directions, the algorithms may "over-shoot" or "under-update" since we require the step size to be always η.

1.4 Layers

1.4.1 Densely connected layers

Let the input x be a K-dimensional vector, and the output y be a M-dimensional vector, then a densely connected layer can be represented as $y = F(Wx + b)$, where $W$is a $M \times K$-dimensional matrix, b is a K dimensional vector. Typically, the function $F : R^K \to R^K$ is either an element-wise sigmoid function,

$$F_i(x) = \frac{1}{1 + \exp(-x)},$$

or an element-wise tanh function,

$$F_i(x) = \frac{1 - \exp(-2x)}{1 + \exp(-2x)}.$$

To derive the gradient, note,

$$\begin{aligned} dL &= \frac{\partial L}{\partial y^t}\frac{\partial y}{\partial z^t}(dWx + Wdx + db) \\ &= \frac{\partial y}{\partial z}\frac{\partial L}{\partial y} \cdot (dWx) + \frac{\partial L}{\partial y} \cdot (\frac{\partial y}{\partial z^t}Wdx) + \frac{\partial L}{\partial y} \cdot (\frac{\partial y}{\partial z^t}db) \\ &= \frac{\partial y}{\partial z}\frac{\partial L}{\partial y}x^t \cdot dW + W^t\frac{\partial y}{\partial z}\frac{\partial L}{\partial y} \cdot dx + \frac{\partial y}{\partial z}\frac{\partial L}{\partial y} \cdot db \end{aligned}$$

Therefore we have,

$$\frac{\partial L}{\partial W} = \frac{\partial y}{\partial z}\frac{\partial L}{\partial y}x^t,$$

$$\frac{\partial L}{\partial b} = \frac{\partial y}{\partial z}\frac{\partial L}{\partial y},$$

and

$$\frac{\partial L}{\partial x} = W^t\frac{\partial y}{\partial z}\frac{\partial L}{\partial y}.$$

In practice, it is very not very common to stack many densely connected layers. This is due to the simple fact that regardless of whether the activation function tends to set the gradient to zero.

1.4.2 Convolutional layer

We start with the introduction of the convolution. Let $\mathbf{X}$ be a $H \times W \times D$-dimensional tensor, and $\mathbf{K}$, the "convolution kernels", be a $H^K \times W^K \times D \times D^K$-dimensional tensor, where $H^K < H$ and $W^K < W$. The convolution, $\mathbf{X} * \mathbf{K}$, is defined as a $(H-H^K+1)\times(W-W^K+1)\times D^K$-dimensional tensor, such that,

$$(\mathbf{X}*\mathbf{K})_{i^*,j^*,d^*}=\sum_{i=1}^{H^K}\sum_{j=1}^{W^K}\sum_{d=1}^{D}\mathbf{X}_{i+i^*,j+j^*,d}\mathbf{K}_{i,j,d,d^*}$$

for $1\le i^*\le(H-H^K+1), 1\le j^*\le(W-W^K+1), 1\le d^*\le D^K$.

Intuitively, the convolution is a weighted average of an area. It can be proved that there exists function ϕ s.t. $\mathrm{vec}(\mathbf{X}*\mathbf{K})=\mathrm{vec}(\phi(\mathbf{X})\tilde{K})$. Here, $\tilde{K}$ is a $(H^KW^KD)\times D^K$-dimensional matrix obtained by stacking the first three dimensional of $\mathbf{K}$ together. In addition, there is M s.t. $\mathrm{vec}(\phi(\mathbf{X}))=M\mathrm{vec}(\mathbf{X})$.

With these in minds, we can define a convolution layer, which output the following $(H-H^K+1)\times(W-W^K+1)\times D^K$-dimensional tensor $\mathbf{Y}$ s.t.

$$\mathbf{Y}=F(\mathbf{X}*\mathbf{K}+\mathbf{B}),$$

where F is some function, $\mathbf{K}$ and $\mathbf{B}$ are tensors of appropriate dimension.

To derive the gradients, let B be the $(H-H^K+1)(W-W^K+1)\times D^K$ corresponding to $\mathbf{B}$, let X_K be a $(H-H^K+1)(W-W^K+1)\times D^K$-matrix corresponding to $\mathbf{X}*\mathbf{B}$, and $Z=X_K+B$. We can derive,

$$\begin{aligned}dL&=\frac{\partial L}{\partial Z}\cdot(\phi(\mathbf{X})dW+d\phi(\mathbf{X})W+dB)\\&=\phi^t(\mathbf{X})\frac{\partial L}{\partial Z}\cdot dW+\frac{\partial L}{\partial Z}W^t\cdot d\phi(\mathbf{X})+\frac{\partial L}{\partial Z}\cdot dB\end{aligned}$$

Therefore

$$\frac{\partial L}{\partial W}=\phi^t(\mathbf{X})\frac{\partial L}{\partial Z},$$

$$\frac{\partial L}{\partial\phi(\mathbf{X})}=\frac{\partial L}{\partial Z}W^t,$$

and

$$\frac{\partial L}{\partial B}=\frac{\partial L}{\partial Z}.$$

In addition, we can also derive

$$\frac{\partial L}{\partial\mathrm{vec}^t(X)}=\frac{\partial L}{\partial\mathrm{vec}^t(\phi(\mathbf{X}))}\frac{\partial\mathrm{vec}(\phi(\mathbf{X}))}{\partial\mathrm{vec}^t(X)}=\mathrm{vec}^t(\frac{\partial L}{\partial Z}W^t)M,$$

and

$$\frac{\partial L}{\partial Z}=\frac{\partial L}{\partial Y}\bigodot F'(Z).$$

Now it remains to specify the function F. In fact, any reasonable choices will do here.

1.4.3 Recurrent layers

Typically, the output of a recurrent layer/network is a sequence. To introduce the recurrent layers, let us start with the simple RNN. In this case, assume that the output for a sample is $\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_T$, and the actual value is $y_1, y_2, \ldots, y_T$ (note we have omitted the subscript for samples). The loss for *this sample* is

$$L(y, \hat{y}) = \sum_{t=1}^{T} L_t(y_t, \hat{y}_t).$$

$$\begin{aligned} y_t &= \sigma(W_y h_t), \\ h_t &= f(W_h X_t + V_h h_{t-1} + b_h), 1 \leq t \leq T \end{aligned}$$

where h_t is the output of the hidden layer, and σ and f are functions.

Let us derive the back propagation for simple RNN network, which is usually called BPTT method. Let $z_t = W_y h_t$ and $o_t = W_h x_t + V_h h_{t-1} + b_h$. Note that we have

$$\begin{aligned} dL_t &= \frac{\partial L_t}{\partial \hat{y}_t^t} d\hat{y}_t \\ &= \frac{\partial L_t}{\partial \hat{y}_t^t} \frac{\partial \hat{y}_t}{\partial z_t^t} dW_y h_t \\ &= \frac{\partial \hat{y}_t}{\partial z_t} \frac{\partial L_t}{\partial \hat{y}_t} \cdot (dW_y h_t + W_y dh_t) \end{aligned}$$

Now for h_t, we have

$$\begin{aligned} dh_t &= \frac{\partial f}{\partial o_t^t}(dW_h x_t + dV_h h_{t-1} + V_h dh_{t-1} + db_h) \\ &= \frac{\partial f}{\partial o_t^t}(dW_h x_t + dV_h h_{t-1} + V_h \frac{\partial f}{\partial o_{t-1}^t}(dW_h x_{t-1} + dV_h h_{t-2} + V_h dh_{t-2} + db_{h-1}) + db_h) \\ &= \sum_{i=0}^{t-1}(\prod_{j=0}^{i} V_h^j \frac{\partial f}{\partial o_{t-j}^t}) dW_h x_{t-i} + \sum_{i=0}^{t-1}(\prod_{j=0}^{i} V_h^j \frac{\partial f}{\partial o_{t-j}^t}) dV_h h_{t-i-1} + \sum_{i=0}^{t-1}(\prod_{j=0}^{i} V_h^j \frac{\partial f}{\partial o_{t-j}^t}) db_h \end{aligned}$$

Therefore we have

$$\begin{aligned} dL_t = \frac{\partial \hat{y}_t}{\partial z_t} \frac{\partial L_t}{\partial \hat{y}_t} \cdot (dW_y h_t + W_y \sum_{i=0}^{t-1}(\prod_{j=0}^{i} V_h^j \frac{\partial f}{\partial o_{t-j}^t}) dW_h x_{t-i} \\ + W_y \sum_{i=0}^{t-2}(\prod_{j=0}^{i} V_h^j \frac{\partial f}{\partial o_{t-j}^t}) dV_h h_{t-i-1} + W_y \sum_{i=0}^{t-1}(\prod_{j=0}^{i} V_h^j \frac{\partial f}{\partial o_{t-j}^t}) db_h) \end{aligned}$$

In addition, since

$$\frac{\partial \hat{y}_t}{\partial z_t}\frac{\partial L_t}{\partial \hat{y}_t}\cdot dW_y h_t = \frac{\partial \hat{y}_t}{\partial z_t}\frac{\partial L_t}{\partial \hat{y}_t}h_t^t \cdot dW_y$$

Therefore we have

$$\frac{\partial L_t}{\partial W_y} = \frac{\partial \hat{y}_t}{\partial z_t}\frac{\partial L_t}{\partial \hat{y}_t}h_t^t$$

Furthermore,

$$\begin{aligned}\frac{\partial \hat{y}_t}{\partial z_t}\frac{\partial L_t}{\partial \hat{y}_t}\cdot W_y \sum_{i=0}^{t-1}(\prod_{j=0}^{i} V_h^j \frac{\partial f}{\partial o_{t-j}^t}) dW_h x_{t-i} &= \sum_{i=0}^{t-1}\left(W_y \prod_{j=0}^{i} V_h^j \frac{\partial f}{\partial o_{t-j}^t}\right)^t \frac{\partial \hat{y}_t}{\partial z_t}\frac{\partial L_t}{\partial \hat{y}_t}\cdot dW_h x_{t-i} \\ &= \sum_{i=0}^{t-1}\left(W_y \prod_{j=0}^{i} V_h^j \frac{\partial f}{\partial o_{t-j}^t}\right)^t \frac{\partial \hat{y}_t}{\partial z_t}\frac{\partial L_t}{\partial \hat{y}_t} x_{t-i}^t \cdot dW_h\end{aligned}$$

Therefore,

$$\frac{\partial L_t}{\partial W_h} = \sum_{i=0}^{t-1}\left(W_y \prod_{j=0}^{i} V_h^j \frac{\partial f}{\partial o_{t-j}^t}\right)^t \frac{\partial \hat{y}_t}{\partial z_t}\frac{\partial L_t}{\partial \hat{y}_t} x_{t-i}^t$$

In a similar fashion, we can prove

$$\frac{\partial L_t}{\partial V_h} = \sum_{i=0}^{t-2}\left(W_y \prod_{j=0}^{i} V_h^j \frac{\partial f}{\partial o_{t-j}^t}\right)^t \frac{\partial \hat{y}_t}{\partial z_t}\frac{\partial L_t}{\partial \hat{y}_t} h_{t-i-1}^t$$

and

$$\frac{\partial L_t}{\partial b_h} = \sum_{i=0}^{t-1}\left(W_y \prod_{j=0}^{i} V_h^j \frac{\partial f}{\partial o_{t-j}^t}\right)^t \frac{\partial \hat{y}_t}{\partial z_t}\frac{\partial L_t}{\partial \hat{y}_t}.$$

Due to the term $\prod_{j=0}^{i} V_h^j \frac{\partial f}{\partial o_{t-j}^t}$, simple RNN suffers from vanishing gradient and exploding gradient problems. Some popular alternatives to simple RNN include

LSTM (Hochreiter and Schmidhuber 1997):

$$
\begin{aligned}
i_t &= f(W_{xi}x_t + W_{hi}h_{t-1} + b_i)\\
f_t &= f(W_{xf}x_t + W_{hf}h_{t-1} + b_f)\\
o_t &= f(W_{xo}x_t + W_{ho}h_{t-1} + b_o)\\
g_t &= g(W_{xg}x_t + W_{hg}h_{t-1} + b_g)\\
c_t &= f_t \bigodot c_{t-1} + i_t \bigodot g_t\\
h_t &= o_t \bigodot g(c_t)\\
\hat{y}_t &= h(W_h h_t)
\end{aligned}
$$

and GRU(Cho et al. 2014)

$$
\begin{aligned}
z_t &= f(U_z x_t + W_z s_{t-1} + b_z)\\
r_t &= f(U_r x_t + W_r s_{t-1} + b_r)\\
h_t &= g(U_h x_t + W_h(s_{t-1} \bigodot r_t) + b_h)\\
s_t &= (1 - z_t) \bigodot h_t + z_t \bigodot s_{t-1}\\
\hat{h}_t &= h(W_s s_t)
\end{aligned}
$$

Bibliography

[Cho+14] Kyunghyun Cho et al. "On the properties of neural machine translation: Encoder-decoder approaches". In: *arXiv preprint arXiv:1409.1259* (2014).

[HS97] Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory". In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[Sch16] James R Schott. *Matrix analysis for statistics*. John Wiley & Sons, 2016.