CNN for Dummies

Jianxin Wu

LAMDA Group
National Key Lab for Novel Software Technology
Nanjing University, China
wujx2001@gmail.com

June 8, 2015

Abstract

This is a note that describes how a Convolutional Neural Network (CNN) operates from a mathematical perspective. This note is self-contained, and the focus is to make it comprehensible to beginners in the CNN field.

Contents

1 Motivation

The Convolutional Neural Network (CNN) has shown excellent performance in many computer vision and machine learning problems. Many solid papers have been published in this topic, and quite some high quality open source CNN software packages have been made available.

There are also well-written CNN tutorials or CNN software manuals. However, I believe that an introductory CNN material specifically prepared for beginners is still needed. Research papers are usually very terse and lack details. It might be difficult for beginners to read such papers. A tutorial may not cover all the necessary details to understand exactly how a CNN runs (i.e., learning and predicting).

This note tries to present beginners with a document that[1]

- Is self-contained. It is expected that all required mathematical background knowledge are introduced in this note itself;
- Has details for all the derivations. As the title ("for dummies") suggests, this note tries to explain all the necessary math in details. We try not to ignore any step in a derivation. Thus, it should be easy for a beginner to follow (although an expert may feel this note as tautological.)
- Ignores implementation details. The purpose is for a reader to understand how a CNN runs at the mathematical level. We will ignore those implementation details. In CNN, making correct choices for various details is one of the keys to its high accuracy. However, we intentionally left this part out, in order for the reader to focus on the mathematics. After understanding the mathematical principles and details, it is more advantageous to learn these implementation and design details with hands-on experience by running real CNN codes.

This note is modeled after [2].

2 Preliminaries

We start by a discussion of some background knowledge that are necessary in order to understand how a CNN runs. One can ignore this section if he/she is familiar with these basics.

2.1 Tensor and vectorization

Everybody is familiar with vectors and matrices. We use a symbol shown in boldface to represent a vector, e.g., $\boldsymbol{x} \in \mathbb{R}^D$ is a column vector with D components. We use a capital letter to denote a matrix, e.g., $X \in \mathbb{R}^{H \times W}$ is a matrix

[1] My original plan is to write this note in the Chinese language, because good and up-to-date CNN introductory materials written in Chinese are particularly missing, to the best of my knowledge. Unfortunately, I failed in learning writing my first Chinese LaTeX document in a reasonable amount of time, and have to give up on this original plan.

with H rows and W columns. The vector $\boldsymbol{x}$ can also be viewed as a matrix with 1 column and D rows.

These concepts can be generalized to higher-order matrices, i.e., tensors. For example, $\boldsymbol{x} \in \mathbb{R}^{H \times W \times D}$ is a 3D tensor. It has HWD numbers, and each of them can be indexed by an index triplet (i, j, d), with $0 \leq i < H$, $0 \leq j < W$, and $0 \leq d < D$. Another way to view a 3D tensor is to treat it as containing D slices of matrices. Every slice is a matrix with size $H \times W$. The first slice contains all the numbers in the tensor that are indexed by $(i, j, 0)$. When $D = 1$, a 3D tensor reduces to a matrix.

Higher order matrices also exist. For example, we will soon see that the filter bank in a convolution layer in a CNN is a 4D tensor.

Given a tensor, we can arrange all the numbers inside it into a long vector, following a pre-specified order. For example, in Matlab, the `(:)` operator converts a matrix into a column vector in the column-first order. An example is:

$$A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}, \quad A(:) = (1, 3, 2, 4)^T = \begin{pmatrix} 1 \\ 3 \\ 2 \\ 4 \end{pmatrix}. \tag{1}$$

In mathematics, we use the notation "vec" to represent this vectorization operator. That is, $\text{vec}(A) = (1, 3, 2, 4)^T$ in the example in Equation 1. In order to vectorize a 3D tensor, we could vectorize its first slice (which is a matrix), then the second slice, ..., till all slices are vectorized. The vectorization of the 3D tensor is then the concatenation of all the slices. This recursive process can be applied to vectorize a 4D (or even higher order) tensor.

2.2 Vector calculus and chain rule

The CNN learning process depends on vector calculus and chain rule. Suppose z is a scalar (i.e., $z \in \mathbb{R}$) and $\boldsymbol{y} \in \mathbb{R}^H$ is a vector. If z is a function of $\boldsymbol{y}$, then the partial derivative of z with respect to $\boldsymbol{y}$ is a vector, defined as

$$\left(\frac{\partial z}{\partial \boldsymbol{y}}\right)_i = \frac{\partial z}{\partial y_i}. \tag{2}$$

In other words, $\frac{\partial z}{\partial \boldsymbol{y}}$ is a vector having *the same size* as $\boldsymbol{y}$, and its i-th element is $\frac{\partial z}{\partial y_i}$. Also note that $\frac{\partial z}{\partial \boldsymbol{y}^T} = \left(\frac{\partial z}{\partial \boldsymbol{y}}\right)^T$.

Furthermore, suppose $\boldsymbol{x} \in \mathbb{R}^W$ is another vector, and $\boldsymbol{y}$ is a function of $\boldsymbol{x}$. Then, the partial derivative of $\boldsymbol{y}$ with respect to $\boldsymbol{x}$ is defined as

$$\left(\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}^T}\right)_{ij} = \frac{\partial y_i}{\partial x_j}. \tag{3}$$

This partial derivative is a $H \times W$ matrix, whose entry at the intersection of the i-th row and j-th column is $\frac{\partial y_i}{\partial x_j}$.

It is easy to see that z is a function of $\boldsymbol{x}$ in a chain-like argument: a function maps $\boldsymbol{x}$ to $\boldsymbol{y}$, and another function maps $\boldsymbol{y}$ to z. A chain rule can be used to compute $\frac{\partial z}{\partial \boldsymbol{x}^T}$, as

$$\frac{\partial z}{\partial \boldsymbol{x}^T} = \frac{\partial z}{\partial \boldsymbol{y}^T} \frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}^T}. \tag{4}$$

A sanity check for Equation 4 is to check the matrix / vector dimensions. Note that $\frac{\partial z}{\partial \boldsymbol{y}^T}$ is a row vector with H elements, or a $1 \times H$ matrix. (Be reminded that $\frac{\partial z}{\partial \boldsymbol{y}}$ is a column vector). Since $\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}^T}$ is an $H \times W$ matrix, the vector / matrix multiplication between them is valid, and the result should be a row vector with W elements, which matches the dimensionality of $\frac{\partial z}{\partial \boldsymbol{x}^T}$.

For specific rules to calculate partial derivatives of vectors and matrices, please refer to the Matrix Cookbook [1].

3 CNN in a nutshell

In this section, we will see how a CNN trains and predicts in an abstract level, with the details leaved out for later sections.

3.1 The architecture

A CNN usually takes a 3D tensor as its input, e.g., an image with H rows, W columns, and 3 slices or channels (R, G, B color channels). The input then sequentially goes through a series of processing. One processing step is usually called a layer, which could be a convolution layer, a pooling layer, a normalization layer, a fully connected layer, a loss layer, etc. We will introduce the details of these layers later in this note.[2]

For now, let us give an abstract description of the CNN structure first.

$$\boldsymbol{x}^1 \longrightarrow \boxed{\boldsymbol{w}^1} \longrightarrow \boldsymbol{x}^2 \longrightarrow \ldots \longrightarrow \boldsymbol{x}^{L-1} \longrightarrow \boxed{\boldsymbol{w}^{L-1}} \longrightarrow \boldsymbol{x}^L \longrightarrow \boxed{\boldsymbol{w}^L} \longrightarrow z \quad (5)$$

The above Equation 5 illustrates how a CNN runs layer by layer forward. The input is $\boldsymbol{x}^1$, usually an image (3D tensor). It goes through the processing in the first layer, which is the first box. We denote the parameters involved in the first layer's processing collectively as a tensor $\boldsymbol{w}^1$. The output of the first layer is $\boldsymbol{x}^2$, which also acts as the input to the second layer processing.

This processing proceeds till all layers in the CNN has been finished, which outputs $\boldsymbol{x}^L$. One additional layer, however, is added for backward error propagation. Let's suppose the problem at hand is an image classification problem with C classes. A commonly used strategy is to output $\boldsymbol{x}^L$ as a C dimensional vector, whose i-th entry encodes the prediction $P(c_i|\boldsymbol{x}^1)$. That is, given the input $\boldsymbol{x}^1$, $\boldsymbol{x}^L$ outputs a vector of CNN-estimated posterior probabilities.

Let's suppose $\boldsymbol{t}$ is the corresponding target (ground-truth) value for the input $\boldsymbol{x}^1$, then a cost or loss function can be used to measure the discrepancy between the CNN prediction $\boldsymbol{x}^L$ and the target $\boldsymbol{t}$. For example, the simplest loss function could be

$$z = \|\boldsymbol{t} - \boldsymbol{x}^L\|^2 , \quad (6)$$

although more complex loss functions are usually used.

For simplicity, Equation 5 explicitly models the loss function as a loss layer, whose processing is modeled as a box with parameters $\boldsymbol{w}^L$.

Note that some layers may not have any parameters, that is, $\boldsymbol{w}^i$ may be empty for some i.

[2] We will give detailed introduction to three types of layers: convolution, pooling, and ReLU. The normalization layer does not seem very important in a CNN's accuracy, while the loss layer is usually easy to understand.

3.2 The forward run

Suppose all the parameters of a CNN model $\boldsymbol{w}^1, \ldots, \boldsymbol{w}^{L-1}$ has been learned, then we are ready to use this model for prediction.

Prediction only involves running the CNN model forward, i.e., in the direction of the arrows in Equation 5.

Let's take the image classification problem as an example. Starting from the input $\boldsymbol{x}^1$, we make it pass the processing of the first layer (the box with parameters $\boldsymbol{w}^1$), and get $\boldsymbol{x}^2$. In turn, $\boldsymbol{x}^2$ is passed into the second layer, etc. Finally, we achieve $\boldsymbol{x}^L \in \mathbb{R}^C$, which estimates the posterior probabilities of $\boldsymbol{x}^1$ belonging to the C categories. We can output the CNN prediction as

$$\arg\max_i x_i^L \,. \tag{7}$$

The problem is: how do we learn the model parameters?

3.3 Stochastic gradient descent (SGD)

Stochastic gradient descent (SGD) is the mainstream method to learn a CNN's parameters.

Let's suppose one training example $\boldsymbol{x}^1$ is given for training such parameters. The training process involves running the CNN network in both directions. We first run the forward network to get $\boldsymbol{x}^L$ to achieve a prediction using the current CNN model. Instead of outputting a prediction, we need to compare the prediction with the target $\boldsymbol{t}$ corresponding to $\boldsymbol{x}^1$, that is, continue running the forward pass till the last loss layer. Finally, we achieve a loss z.

The loss z is then a supervision signal, guiding how the parameters of the model should be modified. And the SGD way of modifying the parameters are

$$\boldsymbol{w}^i \longleftarrow \boldsymbol{w}^i - \eta \frac{\partial z}{\partial \boldsymbol{w}^i} \,. \tag{8}$$

In Equation 8, η is a proper learning rate.

A cautious note about the notation. In most CNN materials, a superscript indicates the "time" (e.g., training epochs). But in this note, we use the superscript to denote the layer index. Please do not get confused. We do not use an additional index variable t to represent time. In Equation 8, the $\longleftarrow$ sign implicitly indicates that the parameters $\boldsymbol{w}^i$ (of the i-layer) are updated from time t to $t+1$. If an time index t is explicit used, this equation will look like

$$\left(\boldsymbol{w}^i\right)^{t+1} = \left(\boldsymbol{w}^i\right)^t - \eta \frac{\partial z}{\partial \left(\boldsymbol{w}^i\right)^t} \,. \tag{9}$$

A new problem now becomes apparent: how to compute the (partial) derivatives, which seem very complex?

3.4 Error back propagation

The last layer partial derivatives are easy to compute. Because $\boldsymbol{x}^L$ is connected to z directly under the control of parameters $\boldsymbol{w}^L$, it is easy to compute $\frac{\partial z}{\partial \boldsymbol{w}^L}$. This step is only needed when $\boldsymbol{w}^L$ is not empty. In the same spirit, it is also easy to compute $\frac{\partial z}{\partial \boldsymbol{x}^L}$.

In fact, for every layer, we compute two sets of results: the partial derivatives of z with respect to the layer parameters $\boldsymbol{w}^i$, and that layer's input $\boldsymbol{x}^i$.

- The term $\frac{\partial z}{\partial \boldsymbol{w}^i}$, as seen in Equation 8, can be used to update the current (i-th) layer's parameters;
- The term $\frac{\partial z}{\partial \boldsymbol{x}^i}$ can be used to update parameters backwards, e.g., to the $(i-1)$-th layer. An intuitive explanation is that $\frac{\partial z}{\partial \boldsymbol{x}^i}$ is the part of the "error" supervision information propagated from z backwards till the current layer, in a layer by layer fashion. Thus, we can continue the back propagation process, and use $\frac{\partial z}{\partial \boldsymbol{x}^i}$ to both guide the updating of parameters and propagate the errors backwards to the $(i-1)$-th layer.

Thus, at each layer i, we need to compute two sets of derivatives: $\frac{\partial z}{\partial \boldsymbol{w}^i}$ and $\frac{\partial z}{\partial \boldsymbol{x}^i}$. This layer-by-layer backward updating procedure makes learning a CNN much easier.

Let's take the i-th layer as an example. When we are updating the i-th layer, the back propagation process for the $(i+1)$-th layer must have been finished. That is, we already computed the terms $\frac{\partial z}{\partial \boldsymbol{w}^{i+1}}$ and $\frac{\partial z}{\partial \boldsymbol{x}^{i+1}}$. Both are stored in memory and ready for use.

Now our task is to compute $\frac{\partial z}{\partial \boldsymbol{w}^i}$ and $\frac{\partial z}{\partial \boldsymbol{x}^i}$. Using the chain rule, we have

$$\frac{\partial z}{\partial (\mathrm{vec}(\boldsymbol{w}^i)^T)} = \frac{\partial z}{\partial (\mathrm{vec}(\boldsymbol{x}^{i+1})^T)} \frac{\partial \mathrm{vec}(\boldsymbol{x}^{i+1})}{\partial (\mathrm{vec}(\boldsymbol{w}^i)^T)}, \tag{10}$$

$$\frac{\partial z}{\partial (\mathrm{vec}(\boldsymbol{x}^i)^T)} = \frac{\partial z}{\partial (\mathrm{vec}(\boldsymbol{x}^{i+1})^T)} \frac{\partial \mathrm{vec}(\boldsymbol{x}^{i+1})}{\partial (\mathrm{vec}(\boldsymbol{x}^i)^T)}. \tag{11}$$

Now that $\frac{\partial z}{\partial \boldsymbol{x}^{i+1}}$ is already computed and stored in memory. It requires just a matrix reshaping operation (vec) and an additional transpose operation to get $\frac{\partial z}{\partial (\mathrm{vec}(\boldsymbol{x}^{i+1})^T)}$, which is the first term in the right hand side (RHS) of both equations. So long as we can compute $\frac{\partial \mathrm{vec}(\boldsymbol{x}^{i+1})}{\partial (\mathrm{vec}(\boldsymbol{w}^i)^T)}$ and $\frac{\partial \mathrm{vec}(\boldsymbol{x}^{i+1})}{\partial (\mathrm{vec}(\boldsymbol{x}^i)^T)}$, we can easily get what we want (the left hand side of both equations).

The computation of $\frac{\partial \mathrm{vec}(\boldsymbol{x}^{i+1})}{\partial (\mathrm{vec}(\boldsymbol{w}^i)^T)}$ and $\frac{\partial \mathrm{vec}(\boldsymbol{x}^{i+1})}{\partial (\mathrm{vec}(\boldsymbol{x}^i)^T)}$ is not difficult in most cases, because $\boldsymbol{x}^i$ is directly related to $\boldsymbol{x}^{i+1}$, through a function with parameters $\boldsymbol{w}^i$. The details of these partial derivatives will be discussed in the following sections.

4 The convolution layer

Now that the CNN architecture is clear, we will discuss in details the different types of layers, starting from the convolution layer.

4.1 Input, output, filters, and notations

Suppose we are considering the l-th layer, whose inputs form a 3D tensor $\boldsymbol{x}^l$ with $\boldsymbol{x}^l \in \mathbb{R}^{H^l \times W^l \times D^l}$. Thus, we need a triplet index set (i^l, j^l, d^l) to pointing to any specific element in $\boldsymbol{x}^l$. That is, the triplet (i^l, j^l, d^l) refers to one element in $\boldsymbol{x}^l$, which is in the d^l-th slice / channel, and at spatial location (i^l, j^l) (at the i^l-th row, and j^l-th column). In actual CNN learning, a mini-batch strategy is

usually used. In that case, $\boldsymbol{x}^l$ becomes a 4D tensor in $\mathbb{R}^{H^l \times W^l \times D^l \times N}$ where N is the mini-batch size. For simplicity we assume that $N = 1$ in this note.

In order to simplify the notations which will appear later, we follow the zero-based indexing convention, which specifies that $0 \leq i^l < H^l$, $0 \leq j^l < W^l$, and $0 \leq d^l < D^l$.

In the l-th layer, a set of convolution filters will transform the input $\boldsymbol{x}^l$ to an output $\boldsymbol{y}$, which is also the input to the next layer. Thus, we notice that $\boldsymbol{y}$ and $\boldsymbol{x}^{l+1}$ in fact refers to the same object, and it is very helpful to keep this point in mind.

Now we turn to the filter bank $\boldsymbol{f}$. Assuming D filters are used and each filter is of spatial span $H \times W$, we note that one filter needs to process all the D^l slices in $\boldsymbol{x}^l$. Thus, $\boldsymbol{f}$ is in $\mathbb{R}^{H \times W \times D^l \times D}$. Similarly, we use index variables $0 \leq i < H$, $0 \leq j < W$, $0 \leq d^l < D^l$ and $0 \leq d < D$ to pointing to a specific element in the filter bank. Also note that this filter bank $\boldsymbol{f}$ refers to the same object as the notation $\boldsymbol{w}^l$ in Equation 5. We changed the notation a bit to make the derivation a little bit simpler in its notations.

These notations will be used throughout the rest of this note.

4.2 The (forward) convolution

We consider the simplest case of convolution: the stride is 1 and no padding is used. Thus, the convolution output will have a spatial size $(H^l - H + 1) \times (W^l - W + 1)$ and it will have D slices. That is, we have $\boldsymbol{y}$ (or $\boldsymbol{x}^{l+1}$) in $\mathbb{R}^{H^{l+1} \times W^{l+1} \times D^{l+1}}$, with $H^{l+1} = H^l - H + 1$, $W^{l+1} = W^l - W + 1$, and $D^{l+1} = D$.

Specifically, the convolution output for the slice d is computed as follows.

- This one filter has size $H \times W \times D^l$;
- Considering a spatial location (i^{l+1}, j^{l+1}), so long as $0 \leq i^{l+1} < H^{l+1} = H^l - H + 1$ and $0 \leq j^{l+1} < W^{l+1} = W^l - W + 1$, we can extract a subvolume from $\boldsymbol{x}^l$ with the same size as this filter. Note that there will be $H^{l+1}W^{l+1}$ such subvolumes;
- Multiply the two 3D tensor (the d-th filter and the subvolume from $\boldsymbol{x}^l$), and sum all the per-element product into a single number.

This number is the convolution result for this subvolume, or equivalently, at the spatial location (i^{l+1}, j^{l+1}) for the slide d. If we repeat the same procedure for all slices $0 \leq d < D$, we get a vector with D elements, which is the complete convolution result for the subvolume. In the output $\boldsymbol{x}^{l+1}$, this vector is $\mathtt{x}(i^{l+1}, j^{l+1}, :)$ in the Matlab notation, where $\mathtt{x}$ is the Matlab counterpart for $\boldsymbol{x}^{l+1}$.

In precise mathematics, the convolution procedure can be expressed as an equation:

$$y_{i^{l+1},j^{l+1},d} = \sum_{i=0}^{H} \sum_{j=0}^{W} \sum_{d=0}^{D} f_{i,j,d} \times x^l_{i^{l+1}+i,j^{l+1}+j,d} \,. \tag{12}$$

Equation 12 is repeated for all $0 \leq d \leq D = D^{l+1}$, and satisfies $0 \leq i^{l+1} < H^l - H + 1 = H^{l+1}, 0 \leq j^{l+1} < W^l - W + 1 = W^{l+1}$. In the equation, $x^l_{i^{l+1}+i,j^{l+1}+j,d}$ refers to the element of $\boldsymbol{x}^l$ indexed by the triplet $(i^{l+1} + i, j^{l+1} + j, d)$.

A bias term b_d is usually added to $y_{i^{l+1},j^{l+1},d}$, we omit this term.

4.3 Expanding the convolution

Equation 12 seems pretty complex. There is a way to expand $\boldsymbol{x}^l$ and simplify the convolution.

Let's consider a special case with $D^l = D = 1$, $H = W = 2$, and $H^l = 3$, $W^l = 4$. That is, we consider convolving a small 3×4 matrix (or image) with a single 2×2 filter. This example is illustrated as

$$\begin{pmatrix} 1 & 2 & 3 & 1 \\ 4 & 5 & 6 & 1 \\ 7 & 8 & 9 & 1 \end{pmatrix} * \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix} = \begin{pmatrix} 12 & 16 & 11 \\ 24 & 28 & 17 \end{pmatrix}, \tag{13}$$

where the first matrix is denoted as A, and the filter simply adds the elements in each subvolume together; $*$ is the convolution operator.

Now let's run a Matlab command `B=im2col(A,[2 2])`, we arrive at a B matrix that is an expanded version of A:

$$B = \begin{pmatrix} 1 & 4 & 2 & 5 & 3 & 6 \\ 4 & 7 & 5 & 8 & 6 & 9 \\ 2 & 5 & 3 & 6 & 1 & 1 \\ 5 & 8 & 6 & 9 & 1 & 1 \end{pmatrix}. \tag{14}$$

It is obvious that now the first column of B corresponds to the first 2×2 subvolume in A, in a column-first order, corresponding to $(i^{l+1}, j^{l+1}) = (1, 1)$. Similarly, the second to last column in B correspond to subvolumes in A with (i^{l+1}, j^{l+1}) being $(2, 1)$, $(1, 2)$, $(2, 2)$, $(3, 1)$ and $(3, 2)$, respectively. That is, the Matlab `im2col` function explicitly expands the required elements for performing each individual convolution into a column in the matrix B. For convenience of (later) notations, we transpose B to get a matrix C, with

$$C = \begin{pmatrix} 1 & 4 & 2 & 5 \\ 4 & 7 & 5 & 8 \\ 2 & 5 & 3 & 6 \\ 5 & 8 & 6 & 9 \\ 3 & 6 & 1 & 1 \\ 6 & 9 & 1 & 1 \end{pmatrix}. \tag{15}$$

Now, if we vectorize the filter itself into a vector (in the same column-first ordering) $(1, 1, 1, 1)^T$, we find that

$$C \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 12 \\ 24 \\ 16 \\ 28 \\ 11 \\ 17 \end{pmatrix}. \tag{16}$$

It is obvious that if we reshape this resulting vector in Equation 16 properly, we get the exact convolution result matrix in Equation 13. That is, the convolution operator is a linear one. We can multiply the expanded input matrix and the vectorized filter to get a result vector, and by reshaping this vector properly we get the correct convolution results.

4.4 Now let's make it formal

Now we can generalize this idea to other situations and formalize them. If $D^l > 1$ (that is, the input $\boldsymbol{x}^l$ has more than 1 slice), the expansion operator could first expand the first slice of $\boldsymbol{x}^l$, then the second, ..., till all D^l slices are expanded. The expanded slices will be stacked together, that is, one column will have $H \times W \times D^l$ elements, rather than $H \times W$.

More formally, suppose $\boldsymbol{x}^l$ is a 3D tensor in $\mathbb{R}^{H^l \times W^l \times D^l}$, with one element in $\boldsymbol{x}^l$ being indexed by a triplet (i^l, j^l, d^l). We also consider a filter bank $\boldsymbol{f}$, whose spatial extent are all $H \times W$. Then, the expansion operator converts $\boldsymbol{x}^l$ into a matrix $\phi(\boldsymbol{x}^l)$. We use two indexes (p, q) to pointing to an element in this matrix. Then, the expansion operator assigns the element (i^l, j^l, d^l) in $\boldsymbol{x}^l$ to the (p, q) entry in $\phi(\boldsymbol{x}^l)$.

From the description of the expansion process, it is clear that given a fixed (p, q), we can calculate its corresponding (i^l, j^l, d^l) triplet, because obviously

$$p = i^{l+1} + (H^l - H + 1) \times j^{l+1}\,, \tag{17}$$

$$q = i + H \times j + H \times W \times d^l\,, \tag{18}$$

$$i^l = i^{l+1} + i\,, \tag{19}$$

$$j^l = j^{l+1} + j\,. \tag{20}$$

In Equation 18, dividing q by HW and take the integer part of the quotient, we can determine which slice (d^l) does it belong to in the subvolume. Similarly, we can get the offsets inside the subvolume as (i, j), where $0 \le i < H$ and $0 \le j < W$. In other words, q completely determines one specific location in a subvolume.

Note that the filtering result is $\boldsymbol{x}^{l+1}$, whose spatial extent is $H^{l+1} = H^l - H + 1$ and $W^{l+1} = W^l - W + 1$. Thus, in Equation 17, the remainder and quotient of dividing p by $H^{l+1} = H^l - H + 1$ will give us the offset in the convolved result (i^{l+1}, j^{l+1}), or, which subvolume is it.

Based on the definition of convolution, it is clear that we can use Equations 19 and 20 find the offset in the input $\boldsymbol{x}^l$ as $i^l = i^{l+1} + i$ and $j^l = j^{l+1} + j$. That is, the mapping from (p, q) to (i^l, j^l, d^l) is one-to-one. However, we want to emphasize that the reverse mapping from (i^l, j^l, d^l) to (p, q) is one-to-many.

Now we use the standard vec operator to convert the filter bank $\boldsymbol{f}$ into a vector. Let's start from one slice / channel of filter, which can be vectorized into a vector in $\mathbb{R}^{HWD^l}$. Thus, the entire filter bank can be reshaped into a matrix with HWD^l rows and D columns (remember that $D^{l+1} = D$.) Let's call this matrix F.

Finally, with all these notations, we have a beautiful equation to calculate convolution results:

$$\mathrm{vec}(\boldsymbol{y}) = \mathrm{vec}(\boldsymbol{x}^{l+1}) = \mathrm{vec}\left(\phi(\boldsymbol{x}^l)F\right)\,. \tag{21}$$

Note that $\mathrm{vec}(\boldsymbol{y}) \in \mathbb{R}^{H^{l+1}W^{l+1}D}$, $\phi(\boldsymbol{x}^l) \in \mathbb{R}^{(H^{l+1}W^{l+1}) \times (HWD^l)}$, and $F \in \mathbb{R}^{(HWD^l) \times D}$. The matrix multiplication $\phi(\boldsymbol{x}^l)F$ results in a matrix of size $(H^{l+1}W^{l+1}) \times D$. The vectorization of this resultant matrix generates a vector in $\mathbb{R}^{H^{l+1}W^{l+1}D}$, which matches the dimensionality of $\mathrm{vec}(\boldsymbol{y})$.

4.5 The Kronecker product

A short detour to the Kronecker product is needed to compute the derivatives.

Given two matrices $A \in \mathbb{R}^{m \times n}$ and $B \in \mathbb{R}^{p \times q}$, the Kronecker product $A \otimes B$ is a $mp \times nq$ block matrix, defined as

$$A \otimes B = \begin{pmatrix} a_{11}B & \cdots & a_{1n}B \\ \vdots & \ddots & \vdots \\ a_{m1}B & \cdots & a_{mn}B \end{pmatrix} . \tag{22}$$

The Kronecker product has the following properties that will be useful for us:

$$(A \otimes B)^T = A^T \otimes B^T , \tag{23}$$

$$\operatorname{vec}(AXB) = (B^T \otimes A) \operatorname{vec}(X) , \tag{24}$$

for matrices A, X, and B with proper dimensions (e.g., when the matrix multiplication AXB is defined.) Note that Equation 24 can be utilized from both directions.

With the help of $\otimes$, we can write down

$$\operatorname{vec}(\boldsymbol{y}) = \operatorname{vec}\left(\phi(\boldsymbol{x}^l)FI\right) = (I \otimes \phi(\boldsymbol{x}^l)) \operatorname{vec}(F) , \tag{25}$$

$$\operatorname{vec}(\boldsymbol{y}) = \operatorname{vec}\left(I\phi(\boldsymbol{x}^l)F\right) = (F^T \otimes I) \operatorname{vec}(\phi(\boldsymbol{x}^l)) , \tag{26}$$

where I is an identity matrix of proper sizes. In Equation 25, the size of I is determined the number of columns in F, hence $I \in \mathbb{R}^{D \times D}$ in Equation 25. Similarly, in Equation 26, $I \in \mathbb{R}^{(H^{l+1}W^{l+1}) \times (H^{l+1}W^{l+1})}$.

4.6 Backward propagation: the parameters

As previously mentioned, we need to compute two derivatives: $\frac{\partial z}{\partial \operatorname{vec}(\boldsymbol{x}^l)}$ and $\frac{\partial z}{\partial \operatorname{vec}(F)}$, where the first term $\frac{\partial z}{\partial \operatorname{vec}(\boldsymbol{x}^l)}$ will be used for backward propagation to the previous ($(l-1)$-th) layer, and the second term will determine how the parameters of the current (l-th) layer will be updated. A friendly reminder is to remember that $\boldsymbol{f}$, F and $\boldsymbol{w}^i$ refers to the same thing (modulo reshaping of the vector or matrix or tensor). Similarly, we can reshape $\boldsymbol{y}$ into a matrix $Y \in \mathbb{R}^{(H^{l+1}W^{l+1}) \times D}$, then $\boldsymbol{y}$, Y and $\boldsymbol{x}^{l+1}$ refers to the same object (again modulo reshaping).

From the chain rule (Equation 10), it is easy to compute $\frac{\partial z}{\partial \operatorname{vec}(F)}$ as

$$\frac{\partial z}{\partial (\operatorname{vec}(F))^T} = \frac{\partial z}{\partial (\operatorname{vec}(Y)^T)} \frac{\partial \operatorname{vec}(\boldsymbol{y})}{\partial (\operatorname{vec}(F)^T)} . \tag{27}$$

The first term in the RHS is already computed in the $(l+1)$-th layer as (equivalently) $\frac{\partial z}{\partial (\operatorname{vec}(\boldsymbol{x}^{l+1}))^T}$. The second term, based on Equation 25, is pretty straightforward:

$$\frac{\partial \operatorname{vec}(\boldsymbol{y})}{\partial (\operatorname{vec}(F)^T)} = \frac{\partial \left(\left(I \otimes \phi(\boldsymbol{x})^T\right) \operatorname{vec}(F) \right)}{\partial (\operatorname{vec}(F)^T)} = I \otimes \phi(\boldsymbol{x}^l) . \tag{28}$$

Note that we have used the fact $\frac{\partial X\boldsymbol{a}^T}{\partial \boldsymbol{a}} = X$ or $\frac{\partial X\boldsymbol{a}}{\partial \boldsymbol{a}^T} = X$ so long as the matrix multiplications are well defined. This equation leads to

$$\frac{\partial z}{\partial (\mathrm{vec}(F))^T} = \frac{\partial z}{\partial (\mathrm{vec}(\boldsymbol{y})^T)} (I \otimes \phi(\boldsymbol{x}^l)) \,. \tag{29}$$

Making a transpose, we get

$$\frac{\partial z}{\partial \,\mathrm{vec}(F)} = \left(I \otimes \phi(\boldsymbol{x}^l)\right)^T \frac{\partial z}{\partial \,\mathrm{vec}(\boldsymbol{y})} = \left(I \otimes \phi(\boldsymbol{x}^l)\right)^T \mathrm{vec}\left(\frac{\partial z}{\partial Y}\right) \tag{30}$$

$$= \left(I \otimes \phi(\boldsymbol{x}^l)^T\right) \mathrm{vec}\left(\frac{\partial z}{\partial Y}\right) \tag{31}$$

$$= \mathrm{vec}\left(\phi(\boldsymbol{x}^l)^T \frac{\partial z}{\partial Y} I^T\right) \tag{32}$$

$$= \mathrm{vec}\left(\phi(\boldsymbol{x}^l)^T \frac{\partial z}{\partial Y}\right) \,. \tag{33}$$

Note that both Equation 24 (from RHS to LHS) and Equation 23 are used in the above derivation.

Thus, we conclude that

$$\frac{\partial z}{\partial F} = \phi(\boldsymbol{x}^l)^T \frac{\partial z}{\partial Y} \,, \tag{34}$$

which is enough to update the parameters in the l-th layer.

4.7 Even higher dimensional indicator matrices

The function $\phi(\cdot)$ has been very useful in our analysis. It is pretty high dimensional, e.g., $\phi(\boldsymbol{x}^l)$ has $H^{l+1}W^{l+1}HWD^l$ elements. From the above, we know that an element in $\boldsymbol{\phi(x)}^l$ is indexed by a pair p and q.

From q we can determine d^l, which slice of the subvolume is used, and also i and j, the spatial offsets inside a subvolume. From p we can determine i^{l+1} and j^{l+1}, the spatial offsets inside the convolved result $\boldsymbol{x}^{l+1}$. The spatial offsets in the input $\boldsymbol{x}^l$ can be determined as $i^l = i^{l+1} + i$ and $j^l = j^{l+1} + j$.

That is, the mapping $t : (p, q) \mapsto (i^l, j^l, d^l)$ is one-to-one, and thus a valid function. The inverse mapping, however, is one-to-many (thus not a valid function). If we use t^{-1} to represent the inverse mapping, we know that $t^{-1}(i^l, j^l, d^l)$ is a set S, where each $(p, q) \in S$ satisfies that $t(p, q) = (i^l, j^l, d^l)$.

Now we take a look at $\phi(\boldsymbol{x}^l)$ from a different perspective. In order to fully specify $\phi(\boldsymbol{x}^l)$, what information are required? It is obvious that the following three types of information are needed (and only those). For *every* element of $\phi(\boldsymbol{x}^l)$, we need to know

(A) Which subvolume does it belong to, i.e., what is the value of p ($0 \le p < H^{l+1}W^{l+1}$)?

(B) Which element is it inside the subvolume, i.e., what is the value of q ($0 \le q < HWD^l$)?

The above two types of information determines a spatial location (p, q) inside $\phi(\boldsymbol{x}^l)$. The only missing information is

(C) What is the value in that position, i.e., $\left[\phi(\boldsymbol{x}^l)\right]_{pq}$?

Since every element in $\phi(\boldsymbol{x}^l)$ is a verbatim copy of one element from $\boldsymbol{x}^l$, we can turn [C] into a different but equivalent one:

(C.1) For $\left[\phi(\boldsymbol{x}^l)\right]_{pq}$, what is its original location inside $\boldsymbol{x}^l$, i.e., an index u that satisfies $0 \leq u < H^l W^l D^l$? And,

(C.2) The entire $\boldsymbol{x}^l$.

It is easy to see that the collective information in both [A, B, C.1] for the entire range of p, q and u, and [C.2] ($\boldsymbol{x}^l$) contains exactly the same amount of information as $\phi(\boldsymbol{x}^l)$.

Since $0 \leq p < H^{l+1}W^{l+1}$, $0 \leq q < HWD^l$, and $0 \leq u < H^l W^l D^l$, we can use a a matrix $H \in \mathbb{R}^{(H^{l+1}W^{l+1}HWD^l)\times(H^lW^lD^l)}$ to encode the information in [A, B, C.1]. One row of this matrix corresponds to one element inside $\phi(\boldsymbol{x}^l)$ (i.e., a (p, q) pair). One row has $H^lW^lD^l$ elements, and each element can be indexed by (i^l, j^l, d^l). Thus, each element in this matrix is indexed by a 5-tuple: (p, q, i^l, j^l, d^l).

Then, we can use the "indicator" method to encode the function $t(p, q) = (i^l, j^l, d^l)$ into H. That is, for any possible element in H, its row index determines a (p, q) pair, and its column index determines a (i^l, j^l, d^l) triplet, then H is defined as

$$H(x, y) = \begin{cases} 1 & \text{if } t(p, q) = (i^l, j^l, d^l) \\ 0 & \text{otherwise} \end{cases} . \tag{35}$$

The H matrix has the following properties:

- It is very high dimensional;
- But it is also very sparse: there is only 1 non-zero entry in the $H^lW^lD^l$ elements in one row;
- H, which uses information [A, B, C.1], only encodes the one-to-one correspondence between any element in $\phi(\boldsymbol{x}^l)$ and any element in $\boldsymbol{x}^l$, it does not encode any specific value in $\boldsymbol{x}^l$;
- Most importantly, putting together the one-to-one correspondence information in H and the value information in $\boldsymbol{x}^l$, obviously we have

$$\operatorname{vec}(\phi(\boldsymbol{x}^l)) = H \operatorname{vec}(\boldsymbol{x}^l) . \tag{36}$$

A natural consequence is that

$$\frac{\partial \operatorname{vec}(\phi(\boldsymbol{x}^l))}{\partial (\operatorname{vec}(\boldsymbol{x}^l))^T} = H , \tag{37}$$

a fact we will use soon.

4.8 Backward propagation: the supervision signal

In the l-th layer, we still need to compute $\frac{\partial z}{\partial \operatorname{vec}(\boldsymbol{x}^l)}$. For this purpose, we want to reshape $\boldsymbol{x}^l$ into a matrix $X \in \mathbb{R}^{(H^l W^l) \times D^l}$, and use these two equivalent forms (modulo reshaping) interchangeably.

The chain rule states that $\frac{\partial z}{\partial (\operatorname{vec}(\boldsymbol{x}^l)^T)} = \frac{\partial z}{\partial (\operatorname{vec}(\boldsymbol{y})^T)} \frac{\partial \operatorname{vec}(\boldsymbol{y})}{\partial (\operatorname{vec}(\boldsymbol{x}^l)^T)}$ (cf. Equation 11). We will start by studying the second term in the RHS (utilizing equations 26 and 37):

$$\frac{\partial \operatorname{vec}(\boldsymbol{y})}{\partial (\operatorname{vec}(\boldsymbol{x}^l)^T)} = \frac{\partial (F^T \otimes I) \operatorname{vec}(\phi(\boldsymbol{x}^l))}{\partial (\operatorname{vec}(\boldsymbol{x}^l)^T)} = (F^T \otimes I) H \,. \tag{38}$$

Thus,

$$\frac{\partial z}{\partial (\operatorname{vec}(\boldsymbol{x}^l)^T)} = \frac{\partial z}{\partial (\operatorname{vec}(\boldsymbol{y})^T)} (F^T \otimes I) H \,. \tag{39}$$

Since (using Equation 24 from right to left)

$$\begin{aligned}
\frac{\partial z}{\partial (\operatorname{vec}(\boldsymbol{y})^T)} (F^T \otimes I) &= \left((F \otimes I) \frac{\partial z}{\partial \operatorname{vec}(\boldsymbol{y})} \right)^T && (40) \\
&= \left((F \otimes I) \operatorname{vec}\left(\frac{\partial z}{\partial Y} \right) \right)^T && (41) \\
&= \operatorname{vec}\left(I \frac{\partial z}{\partial Y} F^T \right)^T && (42) \\
&= \operatorname{vec}\left(\frac{\partial z}{\partial Y} F^T \right)^T , && (43)
\end{aligned}$$

we have

$$\frac{\partial z}{\partial (\operatorname{vec}(\boldsymbol{x}^l)^T)} = \operatorname{vec}\left(\frac{\partial z}{\partial Y} F^T \right)^T H \,, \tag{44}$$

or equivalently

$$\frac{\partial z}{\partial (\operatorname{vec}(\boldsymbol{x}^l))} = H^T \operatorname{vec}\left(\frac{\partial z}{\partial Y} F^T \right) . \tag{45}$$

Let's have a closer look at the RHS. $\frac{\partial z}{\partial Y} F^T \in \mathbb{R}^{(H^{l+1} W^{l+1}) \times (H W D^l)}$, and $\operatorname{vec}\left(\frac{\partial z}{\partial Y} F^T\right)$ is a vector in $\mathbb{R}^{H^{l+1} W^{l+1} H W D^l}$. On the other hand, H^T is an indicator matrix in $\mathbb{R}^{(H^l W^l D^l) \times (H^{l+1} W^{l+1} H W D^l)}$.

In order to pinpoint one element in $\operatorname{vec}(\boldsymbol{x}^l)$ or one row in H^T, we need an index triplet (i^l, j^l, d^l), with $0 \le i^l < H^l$, $0 \le j^l < W^l$, and $0 \le d^l < D^l$. Similarly, to locate a column in H^T or an element in $\frac{\partial z}{\partial Y} F^T$, we need an index pair (p, q), with $0 \le p < H^{l+1} W^{l+1}$ and $0 \le q < H W D^l$.

Thus, the (i^l, j^l, d^l)-th entry of $\frac{\partial z}{\partial (\operatorname{vec}(\boldsymbol{x}^l))}$ equals the multiplication of two vectors: the row in H^T that is indexed by (i^l, j^l, d^l), and $\operatorname{vec}\left(\frac{\partial z}{\partial Y} F^T\right)$.

Furthermore, since H^T is an indicator matrix, in the row vector indexed by (i^l, j^l, d^l), only those entries whose index (p, q) satisfies $t(p, q) = (i^l, j^l, d^l)$ have a value 1, all other entries are 0. Thus, the (i^l, j^l, d^l)-th entry of $\frac{\partial z}{\partial (\operatorname{vec}(\boldsymbol{x}^l))}$ equals the sum of these corresponding entries in $\operatorname{vec}\left(\frac{\partial z}{\partial Y} F^T\right)$.

Transferring the above textual description into precise mathematical form, we get the following succinct equation:

$$\left[\frac{\partial z}{\partial(\mathrm{vec}(\boldsymbol{x}^l))}\right]_{(i^l,j^l,d^l)} = \sum_{(p,q)\in t^{-1}(i^l,j^l,d^l)} \left[\mathrm{vec}\left(\frac{\partial z}{\partial Y}F^T\right)\right]_{(p,q)} . \tag{46}$$

In other words, to compute $\frac{\partial z}{\partial(\mathrm{vec}(\boldsymbol{x}^l))}$, we do not need to explicitly use the extremely high dimensional vector H^T. Instead, Equation 46 and equations 17 to 20 can be used to efficiently find $\frac{\partial z}{\partial(\mathrm{vec}(\boldsymbol{x}^l))}$.

5 The pooling layer

We will use the same notation inherited from the convolution layer. Let $\boldsymbol{x}^l \in \mathbb{R}^{H^l\times W^l\times D^l}$ be the input to the l-th layer, which is now a pooling layer. The pooling operation requires no parameters (i.e., $\boldsymbol{w}^i$ is null hence parameter learning is not needed for this layer), but a spatial extent of the pooling ($H \times W$) is specified in the design of the CNN structure. Assume that H divides H^l and W divides W^l, the output of pooling ($\boldsymbol{y}$ or equivalently $\boldsymbol{x}^{l+1}$) will be a 3D tensor of size $H^{l+1} \times W^{l+1} \times D^{l+1}$, with

$$H^{l+1} = \frac{H^l}{H}, \quad W^{l+1} = \frac{W^l}{W}, \quad D^{l+1} = D^l . \tag{47}$$

A pooling layer operates upon $\boldsymbol{x}^l$ slice by slice (i.e., channel by channel) independently. Within each slice, the matrix with $H^l \times W^l$ elements are spatially divided into $H^{l+1} \times W^{l+1}$ subregions, each subregion being $H \times W$ in size. The pooling operator then maps a subregion into a single number.

Two types of pooling operators are widely used: max pooling and average pooling. In max pooling, the pooling operator maps a subregion to its maximum value, while the average pooling maps a subregion to its average value. In precise mathematics,

$$\text{max}: \qquad y_{i^{l+1},j^{l+1},d} = \max_{0\le i<H, 0\le j<W} x^l_{i^{l+1}\times H+i, j^{l+1}\times W+j,d} , \tag{48}$$

$$\text{average}: \qquad y_{i^{l+1},j^{l+1},d} = \frac{1}{HW}\sum_{0\le i<H, 0\le j<W} x^l_{i^{l+1}\times H+i, j^{l+1}\times W+j,d} , \tag{49}$$

where $0 \le i^{l+1} < H^{l+1}$, $0 \le j^{l+1} < W^{l+1}$, and $0 \le d < D^{l+1}$.

Pooling is a local operator, and its forward computation is pretty straightforward. Now we focus on the back propagation. Only max pooling is discussed and we can resort to the indicator matrix again.[3] All we need to encode in this indicator matrix is: for every element in $\boldsymbol{y}$, where does it come from in $\boldsymbol{x}^l$?

We need a triplet (i^l, j^l, d^l) to pinpoint one element in the input $\boldsymbol{x}^l$, and another triplet $(i^{l+1}, j^{l+1}, d^{l+1})$ to locate one element in $\boldsymbol{y}$. The pooling output $y_{i^{l+1},j^{l+1},d^{l+1}}$ comes from $x^l_{i^l,j^l,d^l}$, if and only if the following conditions are met:

- They are in the same slice;
- The (i^l, j^l)-th spatial entry belongs to the (i^{l+1}, j^{l+1})-th subregion;

[3] Average pooling can be dealt with using a similar idea.

- The (i^l, j^l)-th spatial entry is the largest one in that subregion.

Translating these conditions into equations, we get

$$d^{l+1} = d^l \,, \tag{50}$$

$$\left\lfloor \frac{i^l}{H} \right\rfloor = i^{l+1}, \left\lfloor \frac{j^l}{W} \right\rfloor = j^{l+1} \,, \tag{51}$$

$$x^l_{i^l, j^l, d^l} \geq y_{i+i^{l+1} \times H, j+j^{l+1} \times W, d^l}, \forall\, 0 \leq i < H, 0 \leq j < W \,, \tag{52}$$

where $\lfloor \cdot \rfloor$ is the floor function.

Given a $(i^{l+1}, j^{l+1}, d^{l+1})$ triplet, there is only one (i^l, j^l, d^l) triplet that satisfies all these conditions. Thus, we define an indicator matrix

$$S(\boldsymbol{x}^l) \in \mathbb{R}^{(H^{l+1}W^{l+1}D^{l+1}) \times (H^l W^l D^l)} \,. \tag{53}$$

One triplet of indexes $(i^{l+1}, j^{l+1}, d^{l+1})$ specifies a row in S, while (i^l, j^l, d^l) specifies a column. These two triplets together pinpoint one element in $S(\boldsymbol{x}^l)$. We set that element to 1 if the equations 50 to 52 are simultaneously satisfied, and 0 otherwise.

With the help of this indicator matrix, we have

$$\operatorname{vec}(\boldsymbol{y}) = S(\boldsymbol{x}^l) \operatorname{vec}(\boldsymbol{x}^l) \,. \tag{54}$$

Then, it is obvious that

$$\frac{\partial \operatorname{vec}(\boldsymbol{y})}{\partial (\operatorname{vec}(\boldsymbol{x}^l)^T)} = S(\boldsymbol{x}^l), \quad \frac{\partial z}{\partial (\operatorname{vec}(\boldsymbol{x}^l)^T)} = \frac{\partial z}{\partial (\operatorname{vec}(\boldsymbol{y})^T)} S(\boldsymbol{x}^l) \,, \tag{55}$$

and consequently

$$\frac{\partial z}{\partial \operatorname{vec}(\boldsymbol{x}^l)} = S(\boldsymbol{x}^l)^T \frac{\partial z}{\partial \operatorname{vec}(\boldsymbol{y})} \,. \tag{56}$$

$S(\boldsymbol{x}^l)$ is very sparse. It has exactly one nonzero entry in every row, and at most one nonzero entry in every column. Thus, we do not need to use the entire matrix in the computation. Instead, we just need to record the locations of those nonzero entries—there are only $H^{l+1}W^{l+1}D^{l+1}$ such entries in $S(\boldsymbol{x}^l)$.

6 The ReLU layer

A ReLU layer does not change the size of the input, that is, $\boldsymbol{x}^l$ and $\boldsymbol{y}$ share the same size. In fact, the Rectified Linear Unit (hence the name ReLU) can be regarded as a truncation performed individually for every element in the input:

$$y_{i,j,d} = \max\{0, x^l_{i,j,d}\} \,, \tag{57}$$

with $0 \leq i < H^l = H^{l+1}$, $0 \leq j < W^l = W^{l+1}$, and $0 \leq d < D^l = D^{l+1}$.

There is no parameter inside a ReLU layer, hence no need for parameter learning.

Based on Equation 57, it is obvious that

$$\frac{\mathrm{d}y_{i,j,d}}{\mathrm{d}x^l_{i,j,d}} = \mathbf{1}\left[x^l_{i,j,d} > 0\right] \,, \tag{58}$$

where $\mathbf{1}[x]$ is the indicator function, being 1 if x is true, and 0 otherwise.

Hence, we have

$$\left[\frac{\partial z}{\partial \boldsymbol{x}^l}\right]_{i,j,d} = \begin{cases} \left[\frac{\partial z}{\partial \boldsymbol{y}}\right]_{i,j,d} & \text{if } \boldsymbol{x}^l_{i,j,d} > 0 \\ 0 & \text{otherwise} \end{cases}. \tag{59}$$

Note that $\boldsymbol{y}$ is an alias for $\boldsymbol{x}^{l+1}$.

Strictly speaking, the function $\max(0, x)$ is not differentiable at $x = 0$, hence Equation 58 is a little bit problematic in theory. In practice, it is not an issue and are safe to use, though.

7 Conclusions

We hope this introductory note on CNN is clear, self-contained, and easy to understand to our readers.

Once a reader is confident in his/her understanding of CNN at the mathematical level, in the next step (aka, future work) it is very helpful to get some hands on CNN experience. For example, one can validate what has been talked about in this note using the MatConvNet software package [2].

References

[1] Kaare Brandt Petersen and Michael Syskind Pedersen. *The Matrix Cookbook*, 2012. 4

[2] Andrea Vedaldi and Karel Lenc. MatConvNet - convolutional neural networks for matlab. *arXiv:1412.4564*, 2014. 2, 16