47.25 CONVOLUTIONAL NEURAL NETWORKS

Convolutional neural networks (CNNs) are prevalent in computer vision, image, speech, and language processing applications, where they have been successfully applied to perform classification tasks at high accuracy rates. One of their attractive properties is the ability to operate directly on raw input signals, such as images, and to extract salient features automatically from the raw data. The designer does not need to worry about which features to select to drive the classification process. The extraction of features is achieved by the convolutional network through a succession of correlation layers. Each layer will be characterized by a filtering operation and by a small number of training parameters. These networks will therefore involve many more connections than tuning parameters. As such, it is generally easier to train deep convolutional networks than deep feedforward networks.

47.25.1 Correlation Masks

We start by explaining how the concept of "convolution" is involved in the operation of convolutional neural networks. We illustrate this operation by considering an image processing example.

One common step in image analysis is that of correlating patches of an image with a mask, also called a filter or a kernel. Consider a patch of size $(2K+1)\times(2K+1)$ from a 2D grayscale image. We denote the patch by the matrix $\mathcal{H}$. Each of its entries represents a pixel, and these pixels generally assume values in the range $[0, 255]$. The values in this range represent different shades of gray, varying from black to white. We choose the letter $\mathcal{H}$ to represent the patch because the entries of the patch will end up playing the role of feature vectors (and, by analogy, we have been using the small letter h to denote features). Consider further a 2D mask or kernel, $\mathcal{W}$, represented by a matrix of size $(2K+1)\times(2K+1)$ as well. Each entry in the convolution mask is real-valued. We choose the letter $\mathcal{W}$ because the entries of this mask will end up playing the role of combination weights (which we denoted by the small letter w before). We write schematically:

$$\mathcal{H} = \begin{bmatrix} \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \end{bmatrix}, \quad \mathcal{W} = \begin{bmatrix} \times & \times & \times \\ \times & \boxed{\times} & \times \\ \times & \times & \times \end{bmatrix} \tag{47.1124}$$

where the box inside $\mathcal{W}$ is used to highlight the location of its central entry. We index the rows and columns of $\mathcal{W}$ and $\mathcal{H}$ by the letters k and ℓ, respectively. These indices assume values in the range

$$k, \ell = -K, \ldots, -2, -1, 0, 1, 2, \ldots, K \tag{47.1125}$$

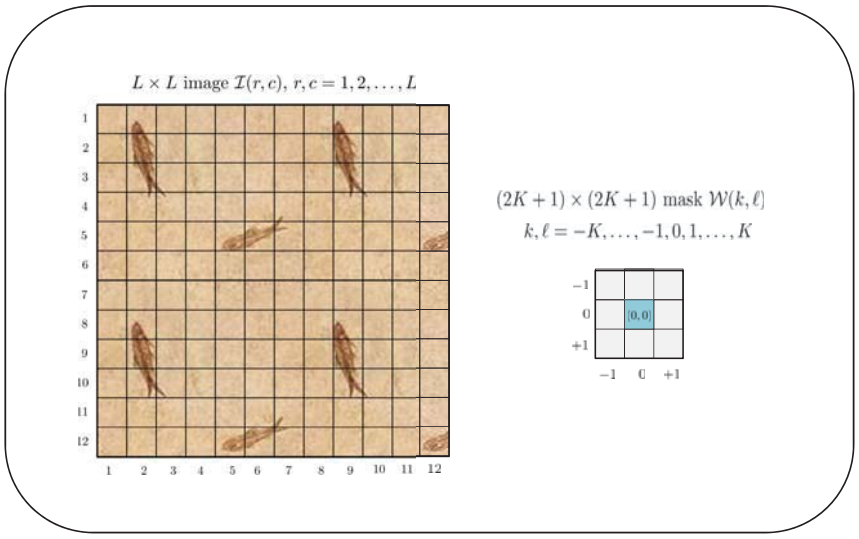

FIGURE 47.81 (*Left*) An image of size $L \times L$ with rows and columns indexed by the letters r and c, respectively. These indices assume values in the range $r, c = 1, 2, \ldots, L$. (*Right*) A mask of size $(2K + 1) \times (2K + 1)$ with rows and columns indexed by the letters k and ℓ. These indices assume values in the range $k, \ell = -K, \ldots, 0, \ldots, K$. The center location in the mask corresponds to coordinates $(0, 0)$.

with the central location in the mask corresponding to coordinates $(0, 0)$. This situation is illustrated in the plot on the right of Fig. 47.81.

The 2D correlation of $\mathcal{H}$ with $\mathcal{W}$ is defined as the transformation that maps $\mathcal{H}$ to the following scalar value:

$$\begin{aligned} \text{corr}(\mathcal{H}, \mathcal{W}) &\triangleq \sum_{k=-K}^{K} \sum_{\ell=-K}^{K} \mathcal{H}(k,\ell)\, \mathcal{W}(k,\ell) && \text{(47.1126a)} \\ &= (\text{vec}(\mathcal{H}))^{\mathsf{T}}\, \text{vec}(\mathcal{W}) && \text{(47.1126b)} \\ &\triangleq h^{\mathsf{T}} w && \text{(47.1126c)} \end{aligned}$$

In other words, we compute a weighted sum of the entries of $\mathcal{H}$ with the weights given by the entries of $\mathcal{W}$. The result of the correlation operation can be written in another equivalent form, as the inner product between the vector representations for $\mathcal{H}$ and $\mathcal{W}$. We denote these vectors by:

$$h \triangleq \text{vec}(\mathcal{H}) \in \mathbb{R}^{(2K+1)^2}, \quad \text{(patch)} \qquad \text{(47.1127a)}$$

$$w \triangleq \text{vec}(\mathcal{W}) \in \mathbb{R}^{(2K+1)^2}, \quad \text{(mask)} \qquad \text{(47.1127b)}$$

where the vec operation for a matrix stacks the columns of the matrix on top of each other. This operation is illustrated in Fig. 47.82, where a matrix of size $A \times A$ is transformed into a vector of size $A^2 \times 1$. Although we are considering square patches and masks, the same discussion is equally applicable to rectangular patches and masks, in which case the vec operation will be applied to non-square matrices.

Now, consider more fully a 2D image, denoted by the symbol $\mathcal{I}$, consisting of a grid of $L \times L$ pixels — see the plot on the left side in Fig. 47.81. The operation of filtering the

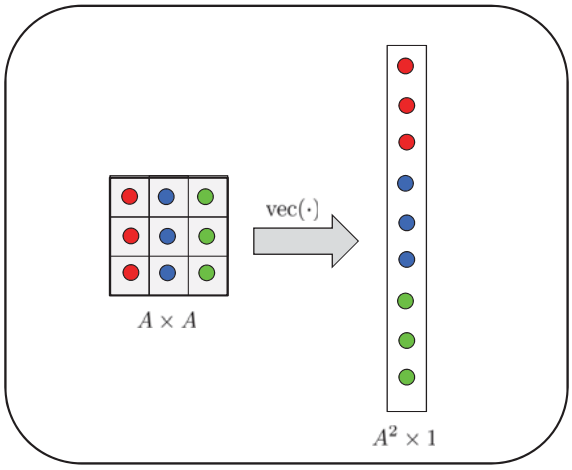

FIGURE 47.82 The vec operation stacks the columns of an $A \times A$ matrix on top of each other and generates a vector of size $A^2 \times 1$.

image by the mask is the process of sliding the mask across the image, say, one column or one row at a time, and evaluating correlations with the covered patch at each step. The collection of all correlations will constitute the entries of a new filtered image, denoted by $\mathcal{F}$:

$$\text{input image, } \mathcal{I} \quad \xrightarrow{\mathcal{W}} \quad \text{output image, } \mathcal{F} \tag{47.1128}$$

More specifically, we start by placing the center location of the mask on top of location $(1,1)$ in the image. Since this pixel lies on the boundary of the image, several other entries in the mask will lie outside the boundaries of the image — see the plot on the left side in Fig. 47.83. To handle this situation, we can assume that the boundaries of the image have been extended with fictitious zero values and carry out the calculations in this manner; other options are possible to handle boundary conditions but our choice is common and is sufficient to convey the main concepts. The value of the $(1,1)$ entry in the filtered image is therefore given by

$$\begin{aligned} \mathcal{F}(1,1) &= \sum_{k=-K}^{K} \sum_{\ell=-K}^{K} \mathcal{I}(1+k, 1+\ell)\, \mathcal{W}(k,\ell) \\ &= h_{1,1}^{\mathsf{T}} w \end{aligned} \tag{47.1129}$$

where the pixels in the image assume zero values whenever the indices $(1+k, 1+\ell)$ fall outside the valid ranges for the image. In (47.1129), we are denoting the vector representation of the patch of the image involved in the evaluation of $\mathcal{F}(1,1)$ by

$$h_{1,1} \triangleq \text{vec}\left([\, \mathcal{I}(1+k, 1+\ell)\,]_{k,\ell=-K}^{K} \right) \in \mathbb{R}^{(2K+1)^2} \tag{47.1130}$$

The vector representation for the mask is w. Next, we shift the mask by one column to the right. Its central entry will now be located on top of entry $(1,2)$ in the image. The result of

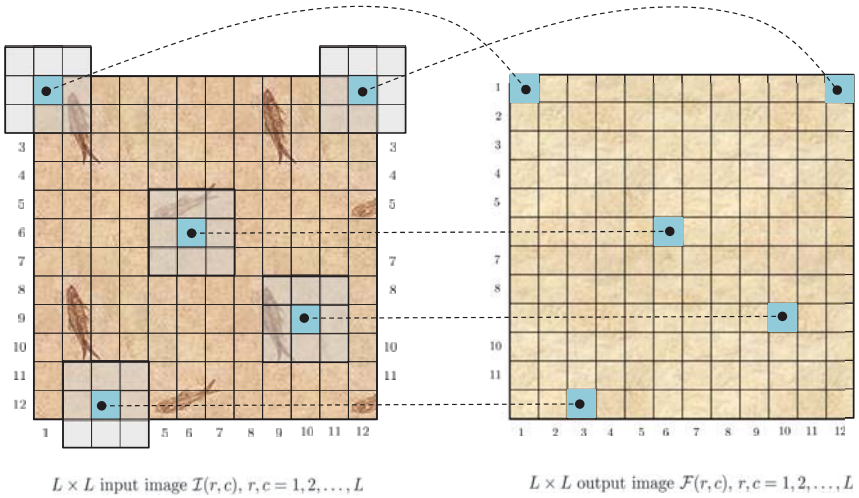

FIGURE 47.83 (*Left*) The mask is moved across the image and correlations are computed. (*Right*) The values obtained from the correlation calculations lead to a new transformed image.

the correlation with the underlying patch is assigned to entry $(1,2)$ of the resulting image:

$$\begin{aligned}\mathcal{F}(1,2) &= \sum_{k=-K}^{K}\sum_{\ell=-K}^{K} \mathcal{I}(1+k,2+\ell)\,\mathcal{W}(k,\ell) \\ &= h_{1,2}^{\mathsf{T}} w \end{aligned} \tag{47.1131a}$$

where we are, similarly, denoting the vector representation of the patch of the image involved in the evaluation of $\mathcal{F}(1,2)$ by

$$h_{1,2} \triangleq \text{vec}\Big(\big[\,\mathcal{I}(1+k,2+\ell)\,\big]_{k,\ell=-K}^{K}\Big) \in \mathbb{R}^{(2K+1)^2} \tag{47.1132}$$

Observe that the successive vectors $\{h_{1,1}, h_{1,2}\}$ share several entries in common. More generally, the $(r,c)-$th entry of $\mathcal{F}$ is given by

$$\boxed{\mathcal{F}(r,c) = \sum_{k=-K}^{K}\sum_{\ell-K}^{K} \mathcal{I}(r+k,c+\ell)\,\mathcal{W}(k,\ell) \;=\; h_{r,c}^{\mathsf{T}} w} \tag{47.1133}$$

where

$$h_{r,c} \triangleq \text{vec}\Big(\big[\,\mathcal{I}(r+k,c+\ell\,\big]_{k,\ell=-K}^{K}\Big) \in \mathbb{R}^{(2K+1)^2} \tag{47.1134}$$

In this description we are assuming that the mask is shifted by one column or by one row at a time. We say that the *stride* value is equal to one in both the horizontal and vertical directions. As a result, the filtered image $\mathcal{F}$ will have the same $L \times L$ size as the input image, $\mathcal{I}$. We can also consider higher values for the stride variable, and these values can differ along the horizontal and vertical dimensions. In that case, the dimensions of the filtered image, $\mathcal{F}$, will end up being smaller than the input image.

In future Example 47.70 on edge detection, we will provide examples for masks $\mathcal{W}$, e.g., the Sobel and Prewitt masks, and show how masks can be used to reveal useful features in the images. Another popular choice is the Gaussian mask, whose entries are defined by:

$$\mathcal{W}(k,\ell) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{k^2+\ell^2}{2\sigma^2}}, \quad k,\ell = -K,\ldots,0,\ldots,K \tag{47.1135}$$

where $\sigma^2 > 0$ controls the width of the Gaussian shape. In the context of convolutional networks, the masks will not be fixed, as is the case with the Sobel, Prewitt, and Gaussian masks, but their entries will instead be learned and adapted by a training algorithm as we explain in later sections.

Example 47.69 (Convolution masks)

The reason the correlation mask $\mathcal{W}$ is also called a convolution mask is because operation (47.1133) can be interpreted as a convolution operation in 2D. To see this, we introduce a flipped mask, denoted by the notation $\mathcal{W}^{\#}$ and whose entries are given by:

$$\mathcal{W}^{\#}(k,\ell) \triangleq \mathcal{W}(-k,-\ell) \tag{47.1136}$$

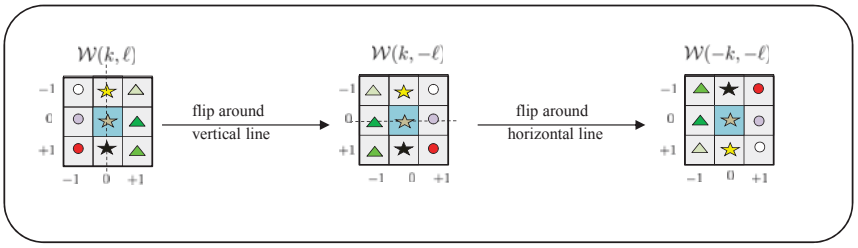

FIGURE 47.84 The flipped mask $\mathcal{W}(-k,-\ell)$ is obtained by reflecting the entries of $\mathcal{W}$ around a vertical line cutting through the center of the mask, and subsequently around a horizontal line also cutting through the center of the mask. The order of the reflections can be reversed.

Moving from $\mathcal{W}$ to $\mathcal{W}^{\#}$ amounts to reflecting the entries of $\mathcal{W}$ around a vertical line cutting through the center of the mask, and subsequently around a horizontal line also cutting through the center of the mask. This construction is illustrated in Fig. 47.84. The order of the reflections can be reversed: we may reflect first around the horizontal line, followed by the reflection around the vertical line. If we introduce the reversed identity matrix with ones of the anti-diagonal — recall definition (36.100):

$$\widetilde{I} \triangleq \begin{bmatrix} & & & 1 \\ & & 1 & \\ & \cdot & & \\ 1 & & & \end{bmatrix} \tag{47.1137}$$

then, since $\widetilde{I}$ is orthogonal, we can relate the matrices $\mathcal{W}^{\#}$ and $\mathcal{W}$, and their vector representations, directly to each other as follows:

$$\boxed{\begin{aligned} &\mathcal{W}^{\#} = \widetilde{I}\mathcal{W}\widetilde{I}, \quad \mathcal{W} = \widetilde{I}\mathcal{W}^{\#}\widetilde{I} \\ &w^{\#} = \widetilde{I}w, \qquad w = \widetilde{I}w^{\#} \end{aligned}} \tag{47.1138}$$

Now, returning to (47.1133), if we introduce the change of variables $k' = r + k$ and $\ell' = c + \ell$, then it also holds that:

$$\begin{aligned}
\mathcal{F}(r,c) &= \sum_{k'=r-K}^{r+K} \sum_{\ell'=c-K}^{c+K} \mathcal{I}(k',\ell')\,\mathcal{W}(k'-r,\ell'-c) \\
&= \sum_{k'=r-K}^{r+K} \sum_{\ell'=c-K}^{c+K} \mathcal{I}(k',\ell')\,\mathcal{W}^{\#}(r-k',c-\ell') \\
&= \sum_{k=r-K}^{r+K} \sum_{\ell=c-K}^{c+K} \mathcal{I}(k,\ell)\,\mathcal{W}^{\#}(r-k,c-\ell), \quad \text{(redefine } (k,\ell) \leftarrow (k',\ell')) \\
&= \sum_{k=-K}^{K} \sum_{\ell=-K}^{K} \mathcal{W}^{\#}(k,\ell)\mathcal{I}(r-k,c-\ell) \\
&\triangleq \mathcal{I} \star \mathcal{W}^{\#} \qquad (47.1139)
\end{aligned}$$

where the next to last expression shows that the calculation of $\mathcal{F}(r,c)$ amounts to determining the value of the 2D convolution of $\mathcal{I}(r,c)$ with the flipped mask, $\mathcal{W}^{\#}$.

$\diamond$

Example 47.70 (Edge detection)

Before moving further, we illustrate one useful application of correlation masks. A problem of fundamental importance in image analysis is edge detection. Edges characterize object boundaries and are therefore useful in many applications requiring segmentation, registration, and identification of objects. One typical application where edge detection techniques are involved is automatic character recognition.

Edges in an image are determined by pixel locations where abrupt changes in the gray levels occur. If we could assume that the intensity of the pixels varies continuously in the (x,y) coordinate variables, where x, y are now real-valued, then the derivative of the image $\mathcal{I}(x,y)$, at an edge location (x,y), will assume a local maximum value. This property motivates one technique for detecting edges by examining the gradient of $\mathcal{I}(x,y)$. This task can be accomplished by using gradient operators along the vertical and horizontal directions. These operators, also called masks, provide finite-difference approximations for the gradient vectors $\partial\mathcal{I}(x,y)/\partial x$ and $\partial\mathcal{I}(x,y)/\partial y$. Two common gradient operators are the Sobel and Prewitt operators, defined by

$$\text{(Sobel)}: \quad \mathcal{W}_x \triangleq \begin{bmatrix} -1 & 0 & 1 \\ -2 & \boxed{0} & 2 \\ -1 & 0 & 1 \end{bmatrix}, \quad \mathcal{W}_y \triangleq \begin{bmatrix} -1 & -2 & -1 \\ 0 & \boxed{0} & 0 \\ 1 & 2 & 1 \end{bmatrix} \qquad (47.1140a)$$

$$\text{(Prewitt)}: \quad \mathcal{W}_x \triangleq \begin{bmatrix} -1 & 0 & 1 \\ -1 & \boxed{0} & 1 \\ -1 & 0 & 1 \end{bmatrix}, \quad \mathcal{W}_y \triangleq \begin{bmatrix} -1 & -1 & -1 \\ 0 & \boxed{0} & 0 \\ 1 & 1 & 1 \end{bmatrix} \qquad (47.1140b)$$

These gradient operators can be used to estimate the gradients of the image at every point by evaluating the correlation of the image with the gradient operators:

$$\mathcal{I} \xrightarrow{\mathcal{W}_x} \mathcal{F}_x \qquad (47.1141a)$$

$$\mathcal{I} \xrightarrow{\mathcal{W}_y} \mathcal{F}_y \qquad (47.1141b)$$

where the notation $(\mathcal{F}_x, \mathcal{F}_y)$ denote the result of computing the gradients along the x (horizontal) and y (vertical) directions. Note that the Prewitt and Sobel operators compute local horizontal and vertical differences or sums. This helps reduce the effect of noise in the data. Note further that these

gradient operators have the desirable property of yielding zeros for uniform regions in the image. Once we have computed the gradient values, we can combine the gradient vector components along the row and column directions and determine the resulting gradient vector magnitudes:

$$\mathcal{F}(r,c) \;=\; \sqrt{\mathcal{F}_x^2(r,c) + \mathcal{F}_y^2(r,c)} \tag{47.1142}$$

We then decide that a pixel (r,c) is an edge location if the gradient vector magnitude at this location is sufficiently large (i.e., larger than a threshold value) — see Fig. 47.85. If we are interested in detecting separately the presence of horizontal or vertical edges, then the decision is based solely on the values of $\mathcal{F}_x(r,c)$ or $\mathcal{F}_y(r,c)$ exceeding appropriate thresholds. This example is meant to illustrate that different choices for the mask can be used to highlight different properties of the underlying image.

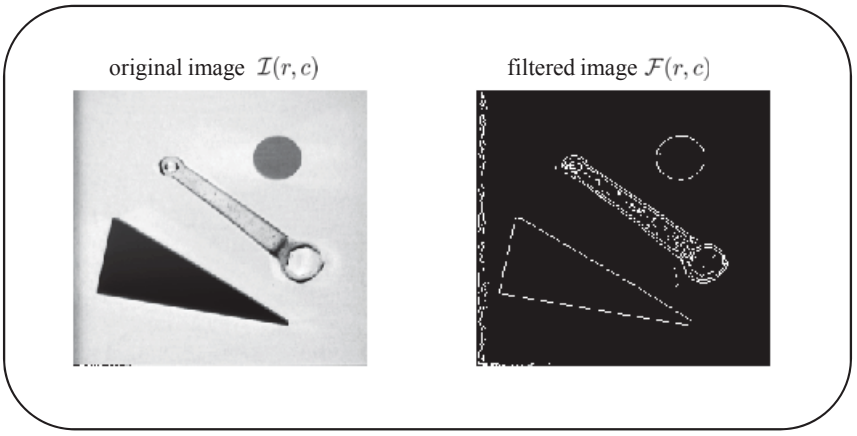

FIGURE 47.85 Illustration of the edge detection procedure using the Sobel operator.

$\diamond$

47.25.2 Correlation Layers

Returning to expression (47.1133), we recall that the successive entries of the filtered image, $\mathcal{F}(r,c)$, are obtained by computing inner products between the corresponding patch vectors, $h_{r,c}$, and the mask vector, w. If we vectorize the input image, $\mathcal{I}$, and denote its vector form by

$$h \;\triangleq\; \text{vec}(\mathcal{I}) \in \mathbb{R}^{L^2} \tag{47.1143}$$

then, the successive patch vectors $\{h_{r,c}\}$ that are using during the correlation process correspond to a partitioning of h into subvectors of size $(2K+1)^2$ each; many of these subvectors will have overlapping entries. If we further represent the filtered image, $\mathcal{F}$, by its vector form, say,

$$y \;\triangleq\; \text{vec}(\mathcal{F}) \in \mathbb{R}^{L^2} \tag{47.1144}$$

then the operation of correlating $\mathcal{I}$ with $\mathcal{W}$, i.e.,

$$\mathcal{I} \;\xrightarrow{\mathcal{W}}\; \mathcal{F} \tag{47.1145}$$

can be equivalently interpreted as transforming the vector h into the vector y:

$$h \xrightarrow{w} y \tag{47.1146}$$

The vector notation will be useful in our subsequent discussions.

Partitioning

Moving forward, we assume from now on that we are dealing with a learning problem that involves high-dimensional input vectors, denoted by $h_n \in \mathbb{R}^M$, where n is the sample index and M is the dimension. For example, if we have a sequence of input images, $\mathcal{I}_n$, of size $L \times L$, then each one of them would be represented by its vector form $\{h_n\}$ of size $M = L^2$. We use the letter M to denote the dimension of the input space for generality and in order to decouple the input vectors from the image processing context. This is because we want to allow for broader applications where h_n need not correspond to the vectorization of images but could arise from some other types of signals (such as speech signals, financial data signals, etc).

We next assume that each long vector h_n is partitioned (with or without overlaps, again for generality) into a collection of P subvectors, say, of size S each and which we index sequentially as $\{h_{n,1}, h_{n,2}, \ldots, h_{n,P}\}$. In the image processing example described before, the partitioning was obtained by sliding a mask across the image, say, with unit stride in the horizontal and vertical directions, and by reading out the entries of the corresponding patches into overlapping subvectors, which we denoted there by $\{h_{r,c}\}$. For the same image processing example, we have

$$S = (2K+1)^2, \quad \text{(size of each subvector)} \tag{47.1147a}$$

$$P = L^2, \quad \text{(number of subvectors)} \tag{47.1147b}$$

$$M = L^2, \quad \text{(size of image vector, } h\text{)} \tag{47.1147c}$$

$$K = \text{square dimension of the mask} \tag{47.1147d}$$

We are denoting the partitioning in the general case by $\{h_{n,p}\}$, with two indices, where n denotes the sample index and p denotes the subvector index. We are also using the notation (M, S, P) for generality.

There are many ways by which an input vector h_n can be partitioned into subvectors $\{h_{n,p}\}$. The image processing example showed one particular way resulting from a sliding mask. Figure 47.86 shows three additional possibilities for dividing h_n into subvectors with and without overlaps (other constructions are of course possible). The rightmost case illustrates a situation in which the entries of the subvectors are selected from across h_n and not necessarily in a structured manner as in the first two cases. Regardless of how the partitioning is performed, what matters for our subsequent presentation is that starting from h_n, there is some well-defined process to obtain subvectors from it. In the image processing example, this process was achieved by sliding a mask across the image and reading out the patches. But other situations are possible for other types of signals, leading to other forms of partitioning. Since we would like our presentation to apply more broadly than the imaging context, we shall denote the process of transforming an input vector h_n into a collection of subvectors $\{h_{n,p}\}$ generically by the notation:

$$\boxed{\{h_{n,p}\}_{p=1}^{P} = \text{partition}(h_n, P)} \tag{47.1148}$$

where the term "partition" refers to any of the possibilities that can be used to partition h_n into a total of P subvectors.

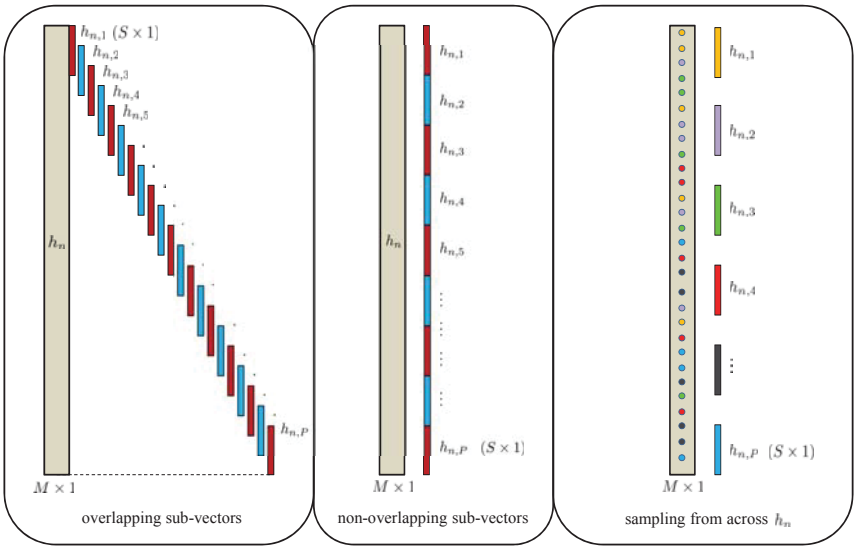

FIGURE 47.86 Illustration of three possibilities for dividing a long feature vector into subvectors with and without overlaps. The rightmost case illustrates a situation in which the entries of the subvectors are selected from across h and not necessarily in a structured manner as in the first two cases.

The partitioning operation (47.1148) can be expressed in matrix form as follows. Figure 47.87 shows one example involving an input vector h_n of size $M = 6$ and $P = 4$ subvectors, $\{h_{n,p}\}$, of size $S = 3$ each. The entries of the input vector are indexed $m = 1, \ldots, 6$. The subvector locations are colored for ease of identification and the numbers next to each entry correspond to their indices in the original input vector. We can generate the order of the indices shown on the right side of the figure by performing the following transformation:

$$\begin{bmatrix} 1 \\ 4 \\ 6 \\ \hline 2 \\ 5 \\ 4 \\ \hline 3 \\ 6 \\ 1 \\ \hline 2 \\ 4 \\ 1 \end{bmatrix} = \underbrace{\left[\begin{array}{cccccc} \mathbf{1} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \mathbf{1} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \mathbf{1} \\ \hline 0 & \mathbf{1} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \mathbf{1} & 0 \\ 0 & 0 & 0 & \mathbf{1} & 0 & 0 \\ \hline 0 & 0 & \mathbf{1} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \mathbf{1} \\ \mathbf{1} & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & \mathbf{1} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \mathbf{1} & 0 & 0 \\ \mathbf{1} & 0 & 0 & 0 & 0 & 0 \end{array}\right]}_{\text{partition matrix}, V(PS\times M)} \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{bmatrix} \tag{47.1149}$$

in terms of a partition matrix, V, whose rows are basis vectors in $\mathbb{R}^M$.

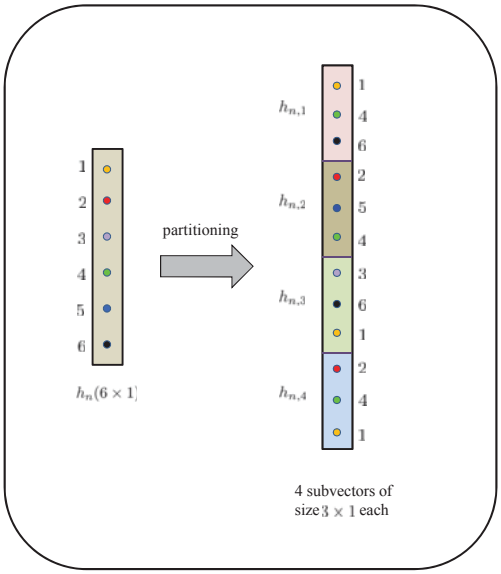

FIGURE 47.87 The entries of the input vector, h_n, are partitioned into subvectors, $\{h_{n,p}\}$. In this example, there are four overlapping vectors with $P = 4$, $S = 3$, and $M = 6$.

Note further that if we introduce the Kronecker product

$$I_P \otimes \mathbb{1}_S^{\mathsf{T}} = \begin{bmatrix} \mathbb{1}^{\mathsf{T}} & & & \\ & \mathbb{1}^{\mathsf{T}} & & \\ & & \mathbb{1}^{\mathsf{T}} & \\ & & & \mathbb{1}^{\mathsf{T}} \end{bmatrix}, \quad (P \times PS) \tag{47.1150}$$

and multiply V from the left by it, we obtain

$$(I_P \otimes \mathbb{1}_S^{\mathsf{T}})V = \begin{bmatrix} \mathbf{1} & 0 & 0 & \mathbf{1} & 0 & \mathbf{1} \\ 0 & \mathbf{1} & 0 & \mathbf{1} & \mathbf{1} & 0 \\ \mathbf{1} & 0 & \mathbf{1} & 0 & 0 & \mathbf{1} \\ \mathbf{1} & \mathbf{1} & 0 & \mathbf{1} & 0 & 0 \end{bmatrix} \tag{47.1151}$$

This is a useful matrix representation. The unit entries on each of the rows on the right-hand side of (47.1151) indicate which entries of h_n belong to the subvector associated with that row. For example, for the first row, we observe that unit entries appear in columns $\{1, 4, 6\}$; from the figure we see that these indices correspond to the entries that define subvector $h_{n,1}$.

Therefore, we can associate with every partitioning operation a uniquely-defined partition matrix, V, of size $PS \times M$. The matrix generates the indices of the entries within each subvector, $h_{n,p}$, for $p = 1, 2, \ldots, P$. In example (47.1149), the first $S = 3$ rows of V determine the indices for subvector $h_{n,1}$, the next 3 rows determine the indices for the second subvector, $h_{n,2}$, and so on. If we denote the transformation matrix by V then, for any vector h_n, we write

$$\boxed{\text{col}\{h_{n,p}\} = \text{partition}(h_n) \iff \text{col}\{h_{n,p}\} = Vh_n} \tag{47.1152}$$

to represent the operation that maps the entries of $h_n \in \mathbb{R}^M$ into subvectors of size $S \times 1$ each, stacked on top of each other. Observe that the rows of V are formed by stacking the row basis vectors $\{e_k^{\mathsf{T}}\}$ in the same order of the indices in the subvectors $\{h_p\}$:

$$V = \left[\begin{array}{c} e_1^{\mathsf{T}} \\ e_4^{\mathsf{T}} \\ e_6^{\mathsf{T}} \\ \hline e_2^{\mathsf{T}} \\ e_5^{\mathsf{T}} \\ e_4^{\mathsf{T}} \\ \hline e_3^{\mathsf{T}} \\ e_6^{\mathsf{T}} \\ e_1^{\mathsf{T}} \\ \hline e_2^{\mathsf{T}} \\ e_4^{\mathsf{T}} \\ e_1^{\mathsf{T}} \end{array}\right], \quad (PS \times M) \tag{47.1153}$$

Feature Maps

Now, given a partitioning $\{h_{n,p} \in \mathbb{R}^S\}$, for $p = 1, 2, \ldots, P$, the first step in a convolutional neural network is to construct a correlation layer consisting of P identical neural units, one for each subvector. Each correlation layer is characterized by a correlation vector mask, $w \in \mathbb{R}^S$, and a bias coefficient, θ, and each subvector, $h_{n,p}$, is applied to the neural unit defined by weight w and bias θ. This situation is illustrated in the left plot in Fig. 47.88. The activation function for the unit can be any of the possibilities listed earlier in Table 47.11, although the tanh and rectifier functions have been observed to provide enhanced performance in the case of convolutional networks:

$$f(x) = \max(0, x), \quad f(x) = \tanh(x) \tag{47.1154}$$

We denote the scalar outputs of the neurons in this initial correlation layer by $\{y_n(p)\}$. If we return to the image filtering example, and if we ignore for now the activation functions and the bias, then each neuron is filtering a patch of the image (represented by the vector $h_{n,p}$) and generating one entry in the filtered image, $\mathcal{F}$, (represented by $y_n(p)$). Now, however, we are allowing for the transformation from the input image to the filtered image to involve a nonlinearity represented by the activation function, in addition to a bias factor represented by θ. We will continue to designate the resulting filtered image by the letter $\mathcal{F}$. Therefore, the correlation layer can be interpreted as transforming an input image, represented by the patches $\{h_{n,p}\}$, into a transformed image, represented by the entries $\{y_n(p)\}$ where

$$y_n(p) = f\left(h_{n,p}^{\mathsf{T}} w - \theta\right), \quad p = 1, 2, \ldots, P \tag{47.1155}$$

Observe that the parameters (w, θ) are the *same* across all neural units, i.e., for all $p = 1, 2, \ldots, P$. This is a useful observation because it means that this correlation layer is defined by a total of $S + 1$ parameters, even though it has $P(S + 1)$ connections.

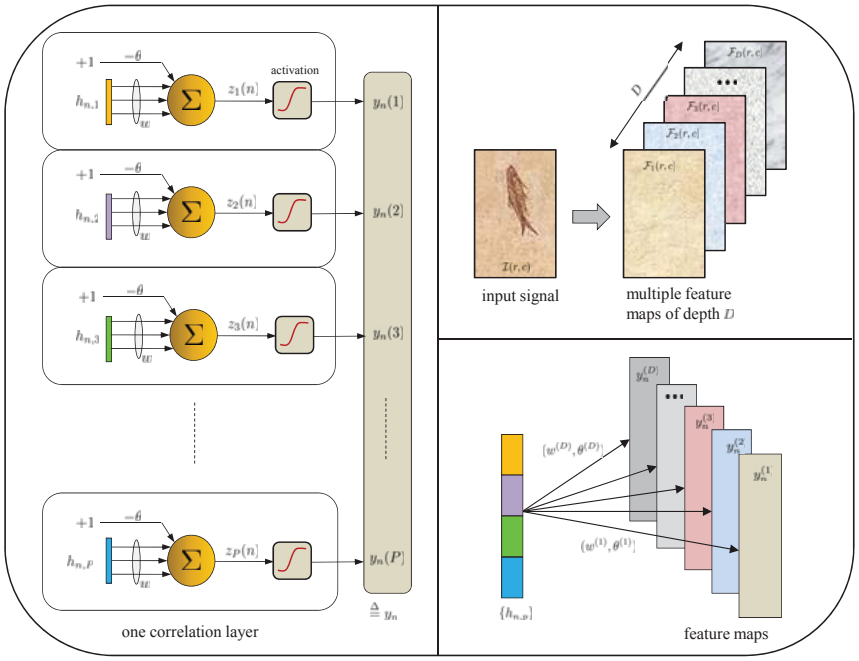

FIGURE 47.88 (*Left*) Construction of one correlation layer. The subvectors $\{h_{n,p}\}$ are fed into neurons with identical weight vector, w, and bias coefficient, θ. (*Bottom right*) Construction of multiple feature maps, of depth D. Each map d is generated by a correlation layer with its own weight vector $w^{(d)} \in \mathbb{R}^S$ and scalar bias coefficient $\theta^{(d)}$. (*Top right*) Analogy with the image processing example. The input signal is the image $\mathcal{I}(r,c)$ and the feature maps correspond to filtered images, $\mathcal{F}_d(r,c)$.

We can describe the mapping from the input space to the output space more compactly in vector form as follows. We collect the output signals of the neurons into a $P \times 1$ vector:

$$y_n \triangleq \begin{bmatrix} y_n(1) \\ y_n(2) \\ \vdots \\ y_n(P) \end{bmatrix}, \quad (P \times 1) \tag{47.1156}$$

and collect the input subvectors into an $S \times P$ matrix:

$$H_n \triangleq \begin{bmatrix} h_{n,1} & h_{n,2} & \ldots & h_{n,P} \end{bmatrix}, \quad (S \times P) \tag{47.1157}$$

Then, it holds that

$$y_n = f\left(H_n^{\mathsf{T}} w - \theta \mathbb{1}\right) \tag{47.1158}$$

where the notation $\mathbb{1}$ denotes the $P \times 1$ vector with all entries equal to one, and the activation function is applied individually to the entries of its vector argument.

We refer to the output vector y_n a *feature map*. Again, using the analogy with the image processing example, if the mask w were chosen to correspond to an edge detection filter, then this initial feature map will be reflecting the location of edges (vertical or horizontal) in the input image. Other choices for w would reveal other useful information that may

be present in the input signal. For this reason, it is customary to construct *multiple* feature maps during this initial step and not only a single map, with different masks w used for the different maps.

Multiple Feature Maps

To describe this flexibility, we now assume that a total of D feature maps are being evaluated during this initial stage by means of D correlation masks. That is, we assume that the initial correlation layer has depth D. We extend the notation by attaching a superscript d to the variables that relate to layer d. Thus, we denote the weights and biases for the $d-$th mask

$$\{w^{(d)}, \theta^{(d)}\}, \quad d = 1, 2, \ldots, D \tag{47.1159}$$

Each of these masks operates on the same subvectors $\{h_{n,p}\}$ and generates a feature map denoted by

$$y_n^{(d)} \triangleq \begin{bmatrix} y_n^{(d)}(1) \\ y_n^{(d)}(2) \\ \vdots \\ y_n^{(d)}(P) \end{bmatrix}, \quad (P \times 1) \tag{47.1160}$$

according to the relation:

$$y_n^{(d)} = f\left(H_n^{\mathsf{T}} w^{(d)} - \theta^{(d)} \mathbb{1}\right), \quad d = 1, 2, \ldots, D \tag{47.1161}$$

The result of this first layer of processing is shown on the right-hand side of Fig. 47.88. The plot in the bottom right corner shows multiple feature maps, of depth D. Each map is generated by a correlation layer with its own weight vector $w^{(d)}$ and bias coefficient $\theta^{(d)}$, applied to the same subvectors $\{h_{n,p}\}$. The top right plot presents the analogy with the image processing example to illustrate the construction. The input signal is the image $\mathcal{I}(r, c)$ and the feature maps correspond to the various filtered images, denoted by $\mathcal{F}_d(r, c)$.

Example 47.71 (RGB color images)

We have motivated our presentation so far by focusing on grayscale images. A similar construction can be applied to color images as well.

Recall that a grayscale image is described by a single matrix of size $L \times L$, and which we denoted by $\mathcal{I}(r, c)$. The entries of this matrix contain pixel values in the range $[0, 255]$. Color images, on the other hand, are described by three matrices, one for each of the RGB color channels: red, green, and blue. In this case, the image is described by a three-dimensional array of size $L \times L \times 3$, which is an example of a *tensor*. The array consists of three matrices, each containing the pixel intensities for one of the fundamental colors (red, green, and blue).

We indicated earlier in (47.1133) that, for grayscale images, the $(r, c)-$th entry of the feature map $\mathcal{F}$ is obtained by computing the inner product between a mask vector w and the relevant patch vector, $h_{r,c}$. Incorporating the activation function and the bias coefficient, the $(r, c)-$th entry in the feature map for grayscale images is then given by

$$\mathcal{F}(r, c) = f\left(h_{r,c}^{\mathsf{T}} w - \theta\right) \tag{47.1162}$$

More generally, for the case of color images, we now have three patches associated with location (r, c): one patch for each of the red, green, and blue arrays. Let us denote the three patch vectors by $\{h_{r,c}^{(1)}, h_{r,c}^{(2)}, h_{r,c}^{(3)}\}$, which are indexed by the subscripts $1, 2$ and 3. This is illustrated in Fig. 47.89.

Each of the patches will be correlated with a mask, say, mask $(w_j, \theta(j))$ for the $j-$th input patch. Then, the $(r, c)-$th entry of the feature map will be obtained by combining the inner products between the patches and the masks, namely,

$$\mathcal{F}(r,c) = f\left(\sum_{j=1}^{3}\left(\left[h_{r,c}^{(j)}\right]^{\mathsf{T}} w_j - \theta(j)\right)\right) \tag{47.1163}$$

where w_j is a vector and $\theta(j)$ is a scalar. The right plot in Fig. 47.89 provides a schematic representation for the construction of the feature map from multiple layers of input subvectors, with one mask $(w_j, \theta(j))$ for each input layer. The input vectors are denoted by $\{h_n^{(j)}, j = 1, 2, 3\}$ with subvectors $\{h_{n,p}^{(j)}\}$, and the output vector is denoted by y_n with entries $\{y_n(p)\}$, i.e.,

$$y_n(p) \;=\; f\left(\sum_{j=1}^{3}\left(\left[h_{n,p}^{(j)}\right]^{\mathsf{T}} w_j - \theta(j)\right)\right) \tag{47.1164}$$

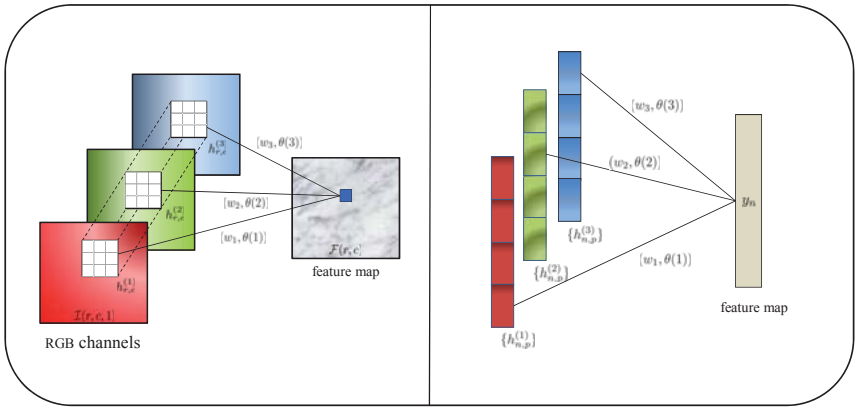

FIGURE 47.89 (*Left*) Each pixel in the feature map (i.e., the filtered image) is obtained by combining the result of the filtering operations across the three RGB arrays. (*Right*) Representation in terms of subvectors across three input layers, denoted by $\{h_{n,p}^{(j)}, j = 1, 2, 3\}$. The output vector is denoted by y_n.

$\diamond$

Parameter Savings

We already noted that the feature maps generated by convolutional layers help reveal salient features in the input space through filtering by the respective masks. There is another important reason for using a convolutional layer, which relates to significant savings in complexity in comparison to feedforward neural networks. To illustrate this point, consider again an image processing example. Assume the image has dimensions 64×64, i.e., $L = 64$. Then, the size of its vector representation, h, will be

$$M = (64)^2 = 4096 \tag{47.1165}$$

If we were to employ a neural unit to process this input vector, then the unit will need to have $M + 1 = 4097$ coefficients (its weights and the bias coefficient). This number of coefficients will be compounded for a larger number of neural units in a feedforward neural network implementation. On the other hand, consider a convolutional layer with

depth $D = 30$ and mask size $2K + 1 = 9$. Then, each neural unit will be employing $2K + 2$ coefficients (the weights of the mask and the bias coefficient). The total number of coefficients that need to be trained for the first convolutional layer will therefore be:

$$30 \times (9 + 1) = 300 \text{ coefficients!} \tag{47.1166}$$

This amounts to a significant reduction in parameter savings. The size of each feature map (each vector $y_{n,d}$) will nevertheless be large and given by

$$P = L^2 = (64)^2 = 4096 \tag{47.1167}$$

For this reason, it is customary to perform a subsampling step to reduce the size of the feature map by means of pooling before constructing subsequent convolutional layers. This step is not only useful in reducing complexity but it also helps smooth out the effect of noise.

47.25.3 Pooling

Pooling is a technique that helps reduce each feature map, $y_n^{(d)}$, of size $P \times 1$ to a smaller vector representation of size $P' \times 1$, and which we shall denote by $t_n^{(d)}$:

$$y_n^{(d)} \ (P \times 1) \ \xrightarrow{\text{pooling}} \ t_n^{(d)} \ (P' \times 1) \tag{47.1168}$$

Pooling can be implemented in a variety of ways. We first illustrate the procedure in the context of image analysis before describing it more generically.

Image Processing Context

Figure 47.90 shows a feature map on the left. A mask of size 3×3 is shown in color placed at several locations in the feature map. In each location, the mask covers a patch of size 3×3. During pooling, the mask locations do not overlap. In pooling, each of these patches are replaced by a single entry computed, for example, as the average of the patch entries or the maximum of the patch entries. For example, if we represent the patch region that is used to generate entry (r, c) in the reduced map by the matrix $\mathcal{H}_{r,c}$ of some size $(2B+1) \times (2B+1)$, and if we denote the sample resulting from the pooling operation by $t(r, c)$, then mean-pooling amounts to the operation

$$t(r, c) \ = \ \frac{1}{(2B+1)^2} \sum_{k=-B}^{B} \sum_{\ell=-B}^{B} \mathcal{H}_{r,c}(k, \ell) \tag{47.1169}$$

so that $t(r, c)$ is the average value over the patch region. We can re-express this pooling operation as correlating the patch $\mathcal{H}_{r,c}$ with a particular mask, which we denote generically by the letter $\mathcal{M}$ and is given by (e.g., for the case of 3×3 patches):

$$\mathcal{M}_{\text{mean}} \ \triangleq \ \begin{bmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{bmatrix} \ = \ \frac{1}{3^2} \mathbb{1}\mathbb{1}^{\mathsf{T}} \tag{47.1170}$$

or, more generally, for patches of size $(2B + 1) \times (2B + 1)$:

$$\mathcal{M}_{\text{mean}} \ \triangleq \ \frac{1}{(2B+1)^2} \mathbb{1}\mathbb{1}^{\mathsf{T}} \tag{47.1171}$$

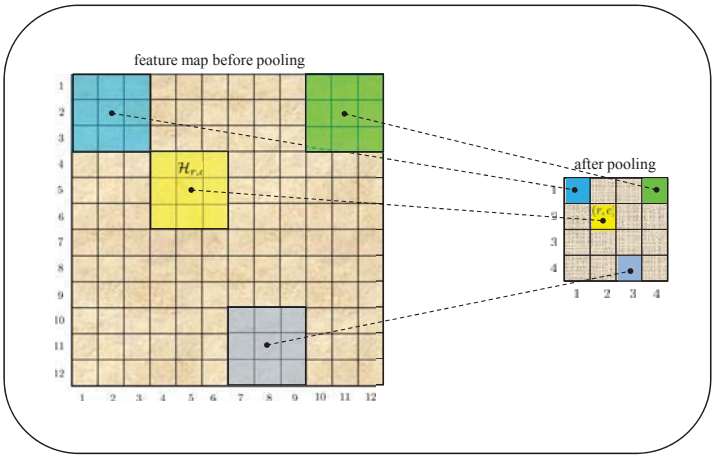

FIGURE 47.90 Patches of the feature map are processed and replaced by a single point. The green windows represent patches of size 3×3. Each patch is replaced, for example, by the average of its entries or the maximum of its entries. The result is the smaller feature map on the right.

There are other forms of pooling. For example, max-pooling would amount to the operation

$$t(r,c) \;=\; \max_{-B\leq k,\ell\leq B} \mathcal{H}_{r,c}(k,\ell) \tag{47.1172}$$

This computation searches for the largest entry in $\mathcal{H}_{r,c}$ and assigns it to $t(r,c)$. We also save the coordinates (k^o,ℓ^o) for this maximum location:

$$(k^o,\ell^o) \;\triangleq\; \operatorname*{argmax}_{-B\leq k,\ell\leq B} \mathcal{H}_{r,c}(k,\ell) \tag{47.1173}$$

These coordinates will be needed during the training procedure when we propagate signals backwards through the network. We can also re-express the max-pooling operation as correlating the patch $\mathcal{H}_{r,c}$ with a particular mask (e.g.,, for the case of 3×3 patches)

$$\mathcal{M}_{\max} \;\triangleq\; \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \;=\; \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \begin{bmatrix} 0 & 1 & 0 \end{bmatrix} \;\triangleq\; e_1 e_2^{\mathsf{T}} \tag{47.1174}$$

with one appearing at the location $(1,2)$ corresponding to the maximum value in $\mathcal{H}_{r,c}$; in the above examples, the vectors $\{e_1, e_2\}$ denote basis vectors with unit entries at locations 1 and 2, respectively, More generally, for patches of size $(2B+1)\times(2B+1)$:

$$\mathcal{M}_{\max} \;\triangleq\; e_{k^o} e_{\ell^o}^{\mathsf{T}} \tag{47.1175}$$

with one appearing at the location (k^o,ℓ^o).

A third form of pooling is ℓ_2-pooling defined as follows (ℓ_p-pooling is considered in Prob. 47.163):

$$t(r,c) \;=\; \left(\sum_{k=-B}^{B} \sum_{\ell=-B}^{B} \mathcal{H}_{r,c}^2(k,\ell) \right)^{1/2} \;=\; \|\mathcal{H}_{r,c}\|_{\mathrm{F}} \tag{47.1176}$$

which amounts to computing the Euclidean norm of the patch entries (or the Frobenius norm of the patch matrix).

Pooling can also be achieved by subsampling the feature map by some factor. For example, if the sub-sampling factor is chosen to be 3 for each of the vertical and horizontal directions in an image, then every third row entry and every third column entry in the feature map is retained while the remaining entries are discarded. Schematically, the transformation takes the following form

$$\begin{bmatrix} \boxed{a} & \times & \times & \boxed{b} & \times & \times & \boxed{c} & \times & \times \\ \times & \times & \times & \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times & \times & \times & \times \\ \boxed{d} & \times & \times & \boxed{e} & \times & \times & \boxed{f} & \times & \times \\ \times & \times & \times & \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times & \times & \times & \times \\ \boxed{g} & \times & \times & \boxed{h} & \times & \times & \boxed{i} & \times & \times \end{bmatrix} \longrightarrow \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \tag{47.1177}$$

We can represent the sub-sampling operation as correlating the patch $\mathcal{H}_{r,c}$ with a particular mask (e.g., for the case of 3×3 patches)

$$\mathcal{M}_{\text{sub}} \triangleq \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \triangleq e_1 e_1^{\mathsf{T}} \tag{47.1178}$$

with one appearing at the leftmost corner. More generally, we can model subsampling as the process of retaining one entry from $\mathcal{H}_{r,c}$ and discarding all other entries, so that the process can be represented by a mask of the form:

$$\mathcal{M}_{\text{sub}} \triangleq e_{k^o} e_{\ell^o}^{\mathsf{T}} \tag{47.1179}$$

where (k^o, ℓ^o) denote the location of the entry that is retained.

One other form of pooling we may consider is to select some arbitrary mask, $\mathcal{M}$, and to compute $t(r,c)$ by correlating the patch region with the mask; this construction allows us to perform pooling by giving different weights to the entries in a patch:

$$t(r,c) = \sum_{k=-B}^{B} \sum_{\ell=-B}^{B} \mathcal{H}_{r,c}(k,\ell) \mathcal{M}(k,\ell) = \mathcal{H}_{r,c} \odot \mathcal{M} \tag{47.1180}$$

where the notation $\odot$ refers to the Hadamard product of two matrices (their elementwise product).

Observe that in all these cases, pooling corresponds to placing *non-overlapping* masks on top of the feature map in order to evaluate the reduced map. For each output location (r,c), the entries in the patch $\mathcal{H}_{r,c}$ that are used to evaluate $t(r,c)$ do not overlap with the entries of the patch $\mathcal{H}_{r',c'}$ used to evaluate $t(r',c')$ at some other location (r',c'). If desired, we may consider using overlapping masks for pooling. However, it is customary to avoid overlaps at this stage.

Gradient Matrices and Upsampling

One important fact to note in these various examples is that pooling is a function that maps a collection of entries, corresponding to the entries of the patch region $\mathcal{H}_{r,c}$, into a single

scalar value, $t(r,c)$. For the moment, let us denote the entries of $\mathcal{H}_{r,c}$ by the generic notation $\{x_1, x_2, x_3, \ldots\}$, e.g.,

$$\mathcal{H}_{r,c} = \begin{bmatrix} x_1 & x_2 & x_3 \\ x_4 & x_5 & x_6 \\ x_7 & x_8 & x_9 \end{bmatrix} \tag{47.1181}$$

and write

$$t(r,c) \;=\; \text{pool}\,(x_1, x_2, x_3, \ldots) \tag{47.1182}$$

In a future section, when we discuss the procedure for training convolutional neural networks, it will become necessary to evaluate the derivatives of this pooling function relative to its arguments, i.e., to evaluate terms of the form

$$\frac{\partial t(r,c)}{\partial x_k}, \;\; k = 1, 2, 3, \ldots \tag{47.1183}$$

If we collect these partial derivatives in matrix form, we obtain a matrix that has the same dimensions as $\mathcal{H}_{r,c}$. We denote this matrix generically by the notation:

$$\frac{\partial t(r,c)}{\partial \mathcal{H}_{r,c}} = \begin{bmatrix} \partial t(r,c)/\partial x_1 & \partial t(r,c)/\partial x_2 & \partial t(r,c)/\partial x_3 \\ \partial t(r,c)/\partial x_4 & \partial t(r,c)/\partial x_5 & \partial t(r,c)/\partial x_6 \\ \partial t(r,c)/\partial x_7 & \partial t(r,c)/\partial x_8 & \partial t(r,c)/\partial x_9 \end{bmatrix} \tag{47.1184}$$

It is straightforward to evaluate these gradients for the various operations of pooling that we presented:

$$(\text{mean pooling}): \qquad \frac{\partial t(r,c)}{\partial \mathcal{H}_{r,c}} = \frac{1}{(2B+1)^2}\mathbb{1}\mathbb{1}^{\mathsf{T}} \;\triangleq\; \mathcal{M}_{\text{mean}} \tag{47.1185a}$$

$$(\text{max pooling}): \qquad \frac{\partial t(r,c)}{\partial \mathcal{H}_{r,c}} = e_{k^o} e_{\ell^o}^{\mathsf{T}} \;\triangleq\; \mathcal{M}_{\text{max}} \tag{47.1185b}$$

$$(\text{subsampling}): \qquad \frac{\partial t(r,c)}{\partial \mathcal{H}_{r,c}} = e_{k^o} e_{\ell^o}^{\mathsf{T}} \;\triangleq\; \mathcal{M}_{\text{sub}} \tag{47.1185c}$$

$$(\ell_2\text{−pooling}): \qquad \frac{\partial t(r,c)}{\partial \mathcal{H}_{r,c}} = \frac{1}{t(r,c)}\mathcal{H}_{r,c} \tag{47.1185d}$$

$$(\text{masking/correlation}): \qquad \frac{\partial t(r,c)}{\partial \mathcal{H}_{r,c}} \;\triangleq\; \mathcal{M} \tag{47.1185e}$$

We denote any of these gradient matrices generically by the letter $\mathcal{G}$ and collect them into Table 47.12. Observe that, in general, the value of $\mathcal{G}$ is a function of (r,c).

We further introduce a new operation, called "upsampling". For any scalar α, this operation transforms α a matrix quantity as follows:

$$\boxed{\text{up}(\alpha) \;=\; \alpha\mathcal{G}, \quad \alpha \in \mathbb{R}} \tag{47.1186}$$

where we scale the gradient matrix, $\mathcal{G}$, by α. Therefore, pooling and upsampling operate in opposite directions: one reduces the dimension of a patch down to a scalar while the other recovers the matrix dimension. This construction is illustrated in Fig. 47.91.

TABLE 47.12 Gradient matrices for some common pooling strategies in 2D.

type	definition	gradient matrix, $\mathcal{G}$
mean pooling	$\frac{1}{(2B+1)^2}\sum_{k=-B}^{B}\sum_{\ell=-B}^{B}\mathcal{H}_{r,c}(k,\ell)$	$\frac{1}{(2B+1)^2}\mathbb{1}\mathbb{1}^{\mathsf{T}}$
max pooling	$\max_{-B\leq k,\ell\leq B}\ \mathcal{H}_{r,c}(k,\ell)$	$e_{k^o}e_{\ell^o}^{\mathsf{T}}$
ℓ_2-pooling	$\left(\sum_{k=-B}^{B}\sum_{\ell=-B}^{B}\mathcal{H}_{r,c}^2(k,\ell)\right)^{1/2}$	$\frac{1}{t(r,c)}\mathcal{H}_{r,c}$
subsampling	$\underset{-B\leq k,\ell\leq B}{\text{subsample}}\,\mathcal{H}_{r,c}$	$e_{k^o}e_{\ell^o}^{\mathsf{T}}$
correlation	$\sum_{k=-B}^{B}\sum_{\ell=-B}^{B}\mathcal{H}_{r,c}(k,\ell)\mathcal{M}(k,\ell)$	$\mathcal{M}$

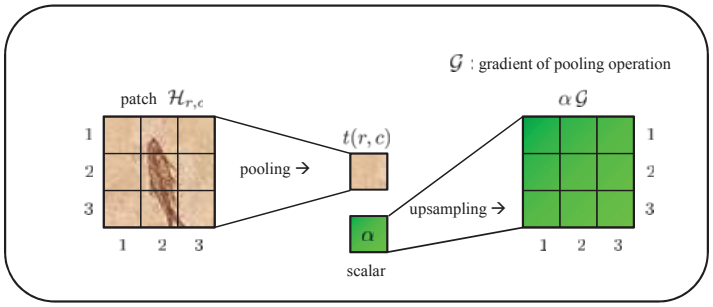

FIGURE 47.91 Upsampling transforms a scalar entry into a patch region by scaling by the gradient matrix of the pooling operation.

General Pooling Formulation

We described pooling in the context of image analysis. The same constructions apply more generally to feature map vectors $y_n^{(d)}$ of size $P\times 1$ in order to reduce them to smaller vector representations, $t_n^{(d)}$ of size $P'\times 1$. Each entry of $t_n^{(d)}$ is obtained by pooling a region of $y_n^{(d)}$. We shall use the letter $p=1,2,\ldots,P$ to index the entries of $y_n^{(d)}$ and the letter $p'=1,2,\ldots,P'$ to index the entries of $t_n^{(d)}$. In pooling, the $p'-$th entry of $t_n^{(d)}$ is obtained by operating on a region of $y_n^{(d)}$, denoted by $\Omega_{p'}$ — see Fig. 47.92. The entries in this region need not be consecutive entries from $y_n^{(d)}$. If we consider again the analogy with the image processing context and refer back to Fig. 47.90, the symbol Ω_1 would correspond to the region in the feature map that is used to construct the $(1,1)$ entry of the reduced map; this region is the patch colored in blue on the left, and which we refereed to earlier by the notation $\mathcal{H}_{1,1}$ used in (47.1169). We denote the pooling operation generically by writing

$$t_n^{(d)}(p')\ =\ \underset{p\in\Omega_{p'}}{\text{pool}}\left(\{y_n^{(d)}(p)\}\right) \tag{47.1187}$$

where the term "pool" refers to the operation that is performed on the entries of $\Omega_{p'}$ to obtain $t_n^{(d)}(p')$.

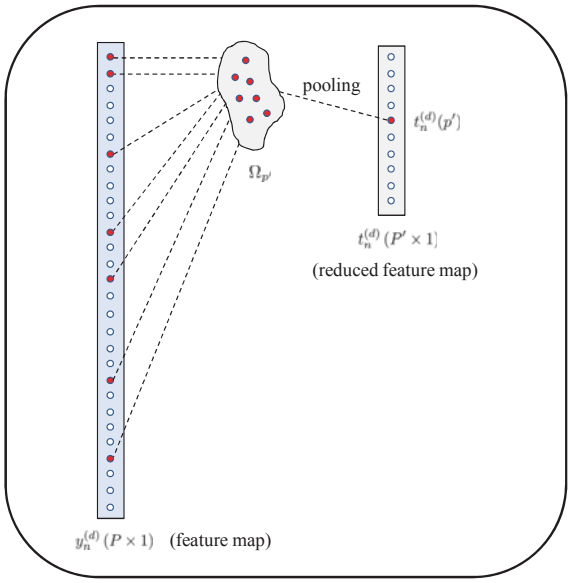

FIGURE 47.92 Entry of index p' in the reduced feature map, $t_n^{(d)}$, is obtained by pooling entries from the region Ω_p' from the original feature map, $y_n^{(d)}$.

TABLE 47.13 Some common choices for the pooling function used in (47.1187).

type	definition	comment
mean pooling	$\dfrac{1}{\lvert\Omega_p'\rvert}\displaystyle\sum_{p\in\Omega_{p'}} y_n^{(d)}(p)$	
max pooling	$\displaystyle\max_{p\in\Omega_{p'}}\left\{y_n^{(d)}(p)\right\}$	index of maximum entry is denoted by p^o
ℓ_2-pooling	$\left(\displaystyle\sum_{p\in\Omega_{p'}}\left(y_n^{(d)}(p)\right)^2\right)^{1/2}$	
correlation	$\displaystyle\sum_{p\in\Omega_{p'}} y_n^{(d)}(p)m^{(d)}(p)$	filter coefficients are denoted by $\{m^{(d)}(p)\}$
subsampling	$\underset{p\in\Omega_{p'}}{\text{subsample}}\left\{y_n^{(d)}\right\}$	index of retained sample is denoted by p^o

Table 47.13 lists common choices for the pooling function, motivated by our earlier discussions in the imaging context. In the table, the notation $|\Omega_{p'}|$ denotes the cardinality of set $\Omega_{p'}$ (i.e., number of elements in it). In the row corresponding to "correlation", the filter coefficients are denoted by $\{m^{(d)}(p)\}$, with the letter "m" used in analogy to "masking", and with the filter allowed to depend on the index, d, of the feature map. In

the row corresponding to "subsampling", say, of order R, the set $\Omega_{p'}$ will consist of R elements with one of them being retained and the others discarded (this description also allows for the possibility in which the retained sample is selected randomly). In all cases, the result of the pooling operation is a transformation from the feature map $y_n^{(d)}$, of size $P \times 1$, into a reduced feature map, $t_n^{(d)}$, of size $P' \times 1$. For compactness, we also rewrite (47.1187) as a mapping between two feature *vectors* rather than individually for each entry of $t_n^{(d)}$, namely,

$$\boxed{t_n^{(d)} = \text{pool}\left(y_n^{(d)}\right), \quad d = 1, 2, \ldots, D} \tag{47.1188}$$

with the understanding that this transformation is computed elementwise according to (47.1187).

We also assume here, as explained earlier after (47.1180) in the context of image processing, that the regions $\Omega_{p'}$ do not overlap with each other. That is, the entries in a region $\Omega_{p'}$ that are used to evaluate $t_n^{(d)}(p')$ do not overlap with the entries in a region $\Omega_{q'}$ used to evaluate some other point $t_n^{(d)}(q')$. This condition implies that every entry in the feature vector $y_n^{(d)}$ influences *only* one entry in the reduced feature vector $t_n^{(d)}$. This condition will help simplify the derivation of the backward step in the backpropagation algorithm — see (47.1238).

The pooling procedure is applied to every feature map of index d and, accordingly, a total of D reduced feature maps are generated. This construction is illustrated in Fig. 47.93. For generality, the figure shows the possibility of multiple layers in the input signal as well (as happens with the case of RGB images); we denote the feature vectors for these input layers by $\{h_n^{(1)}, h_n^{(2)}, h_n^{(3)}\}$ with subvectors $\{h_{n,p}^{(1)}, h_{n,p}^{(2)}, h_{n,p}^{(3)}\}$. Then, in view of (47.1163), the entries $\{y_n^{(d)}(p)\}$ of each feature map $y_n^{(d)}$ are generated by a collection of three weight vectors $\{w_1^{(d)}, w_2^{(d)}, w_3^{(d)}\}$ and three biases $\{\theta^{(d)}(1), \theta^{(d)}(2), \theta^{(d)}(3)\}$ as follows:

$$y_n^{(d)}(p) \;=\; f\left(\sum_{j=1}^{3}\left(\left[h_{n,p}^{(j)}\right]^{\mathsf{T}} w_j^{(d)} - \theta^{(d)}(j)\mathbb{1}\right)\right), \quad d = 1, 2, \ldots, D \tag{47.1189}$$

If we collect the subvectors across the three input layers into $S \times P$ matrices (cf. (47.1157)):

$$H_n^{(1)} \;=\; \left[\begin{array}{cccc} h_{n,1}^{(1)} & h_{n,2}^{(1)} & \ldots & h_{n,P}^{(1)} \end{array}\right], \quad \text{(input layer 1)} \tag{47.1190a}$$

$$H_n^{(2)} \;=\; \left[\begin{array}{cccc} h_{n,1}^{(2)} & h_{n,2}^{(2)} & \ldots & h_{n,P}^{(2)} \end{array}\right], \quad \text{(input layer 2)} \tag{47.1190b}$$

$$H_n^{(3)} \;=\; \left[\begin{array}{cccc} h_{n,1}^{(3)} & h_{n,2}^{(3)} & \ldots & h_{n,P}^{(3)} \end{array}\right], \quad \text{(input layer 3)} \tag{47.1190c}$$

then, in view of (47.1188) and (47.1163), the feature maps in the figure are related to each other as follows:

$$y_n^{(d)} \;=\; f\left(\sum_{j=1}^{3}\left(\left[H_n^{(j)}\right]^{\mathsf{T}} w_j^{(d)} - \theta^{(d)}(j)\mathbb{1}\right)\right), \quad d = 1, 2, \ldots, D \tag{47.1191a}$$

$$t_n^{(d)} \;=\; \text{pool}\left(y_n^{(d)}\right) \tag{47.1191b}$$

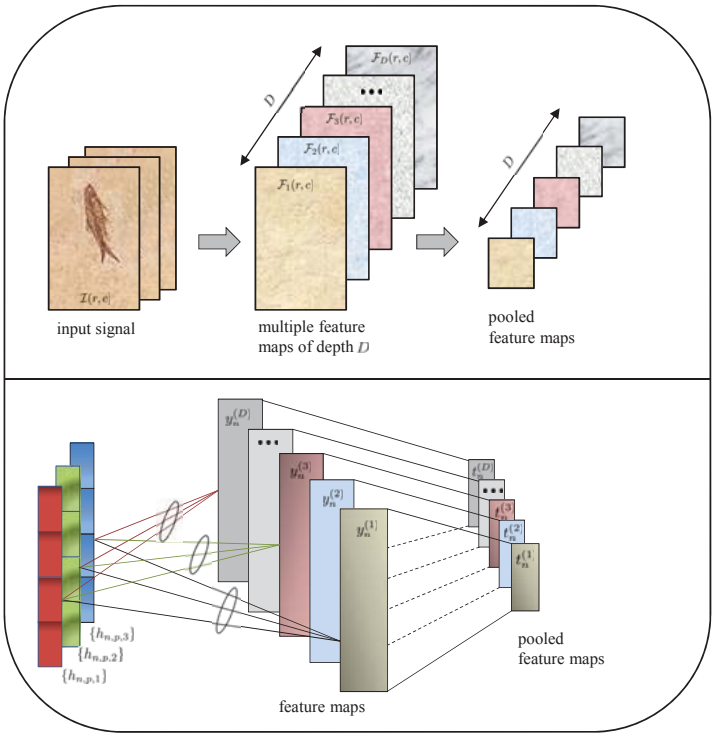

FIGURE 47.93 The feature map vectors $\{y_n^{(d)}, d = 1, 2, \ldots, D\}$ are reduced in size to feature map vectors $\{t_n^{(d)}\}$. The top plot illustrates the operation using the analogy with images, while the bottom plot illustrates the operations in the vector domain.

Gradient Vectors and Upsampling

The pooling operation in the general setting is also a function that maps a collection of entries in $y_n^{(d)}$, corresponding to the entries in the region $\Omega_{p'}$, into a single scalar value, $t_n^{(d)}(p')$. If we denote the entries of the region $y_n^{(d)}(\Omega_{p'})$ by the generic notation $\{x_1, x_2, x_3, \ldots\}$, e.g.,

$$y_n^{(d)}(\Omega_{p'}) \triangleq \{x_1, x_2, x_3, \ldots\} \tag{47.1192}$$

and write

$$t_n^{(d)}(p') = \text{pool}\,(x_1, x_2, x_3, \ldots) \tag{47.1193}$$

then we can once more consider the problem of evaluating derivatives of this function relative to its arguments, namely,

$$\frac{\partial t_n^{(d)}(p')}{\partial x_k}, \quad k = 1, 2, 3, \ldots \tag{47.1194}$$

which can be collected into the column vector notation:

$$\frac{\partial t_n^{(d)}(p')}{\partial y_n^{(d)}(\Omega_{p'})} = \begin{bmatrix} \partial t_n^{(d)}(p')/\partial x_1 \\ \partial t_n^{(d)}(p')/\partial x_2 \\ \partial t_n^{(d)}(p')/\partial x_3 \\ \vdots \\ \partial t_n^{(d)}(p')/\partial x_{|\Omega_{p'}|} \end{bmatrix} \tag{47.1195}$$

It is straightforward to evaluate these gradient vectors for the various operations of pooling shown in Table 47.13 :

$$(\text{mean pooling}): \quad \frac{\partial t_n^{(d)}(p')}{\partial y_n^{(d)}(\Omega_{p'})} = \frac{1}{|\Omega_{p'}|}\mathbb{1} \tag{47.1196a}$$

$$(\text{max pooling}): \quad \frac{\partial t_n^{(d)}(p')}{\partial y_n^{(d)}(\Omega_{p'})} = e_{p^o} \tag{47.1196b}$$

$$(\text{subsampling}): \quad \frac{\partial t_n^{(d)}(p')}{\partial y_n^{(d)}(\Omega_{p'})} = e_{p^o} \tag{47.1196c}$$

$$(\ell_2-\text{pooling}): \quad \frac{\partial t_n^{(d)}(p')}{\partial y_n^{(d)}(\Omega_{p'})} = \frac{1}{t_n^{(d)}(p')} y_n^{(d)}(\Omega_{p'}) \tag{47.1196d}$$

$$(\text{masking/correlation}): \quad \frac{\partial t_n^{(d)}(p')}{\partial y_n^{(d)}(\Omega_{p'})} = m_d = \text{col}\{m_d(p)\} \tag{47.1196e}$$

We shall denote the gradient vector generically by the letter g; observe that, in general, the value of g is a function of n, d, and p' and we can also use the more explicit notation $g_{n,p'}^{(d)}$ to highlight this fact. Table 47.14 lists the pooling functions and their respective gradient vectors for ease of reference.

TABLE 47.14 Pooling functions and their gradient vectors at location p'.

type	**definition**	**gradient vector,** g
mean pooling	$\frac{1}{\|\Omega_p'\|}\sum_{p\in\Omega_{p'}} y_n^{(d)}(p)$	$\frac{1}{\|\Omega_{p'}\|}\mathbb{1}$
max pooling	$\max_{p\in\Omega_{p'}}\left\{y_n^{(d)}(p)\right\}$	e_{p^o}
ℓ_2-pooling	$\left(\sum_{p\in\Omega_{p'}}\left(y_n^{(d)}(p)\right)^2\right)^{1/2}$	$\frac{1}{t_n^{(d)}(p')} y_n^{(d)}(\Omega_{p'})$
correlation	$\sum_{p\in\Omega_{p'}} y_n^{(d)}(p) m^{(d)}(p)$	$m^{(d)}$
subsampling	$\underset{p\in\Omega_{p'}}{\text{subsample}}\,\{y_n^{(d)}\}$	e_{p^o}

We further define the operation of "upsampling", which will play an important role during the training of convolutional neural networks — see, e.g., expression (47.1240)

where the gradient vectors and the upsampling operation appear. For any scalar α, its upsampling amounts to transforming it to a vector as follows:

$$\boxed{\text{up}(\alpha) \;=\; \alpha\, g, \quad \alpha \in \mathbb{R}} \tag{47.1197}$$

That is, we scale g by α. Therefore, pooling and upsampling operate in opposite directions: one reduces the region $\Omega_{p'}$ to a scalar while the other recovers the original dimension. This construction is illustrated schematically in Fig. 47.94. We also need to define a *vector* upsampling operation. For example, if we write $\text{up}(\alpha_1, g_1; \alpha_2, g_2; \alpha_3, g_3)$ that would amount to upsampling the three scalar entries $\alpha_1, \alpha_2, \alpha_3$ by the vectors g_1, g_2, g_3, respectively,

$$\begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{bmatrix} \quad \underset{\longrightarrow}{\text{upsampling}} \quad \begin{bmatrix} \alpha_1 g_1 \\ \alpha_2 g_2 \\ \alpha_3 g_3 \end{bmatrix} \tag{47.1198}$$

This situation will arise as follows. Consider, for illustration purposes, the case in which the vector $t_n^{(d)}$ has 3 entries, i.e., $p' = 1, 2, 3$. Then, there are three gradient vectors associated with $t_n^{(d)}$, namely, $g_{n,1}^{(d)}$, $g_{n,2}^{(d)}$, and $g_{n,3}^{(d)}$; one for each location p'. Then, upsampling relative to these vectors amounts to performing the expansion:

$$\begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{bmatrix} \quad \underset{\longrightarrow}{\text{upsampling}} \quad \begin{bmatrix} \alpha_1\, g_{n,1}^{(d)} \\ \alpha_2\, g_{n,2}^{(d)} \\ \alpha_3\, g_{n,3}^{(d)} \end{bmatrix} \tag{47.1199}$$

where each scalar entry α is expanded according to the gradient vector at the corresponding location p'. It is this form of upsampling that will arise in our derivations, whereby each entry in a vector is upsampled by the gradient vector of the corresponding entry in $t_n^{(d)}$. We shall denote this operation, for any vector v with entries $\{\alpha'_p\}$, by the notation $\text{upv}(\cdot)$ where

$$\boxed{\text{upv}(v) \;\triangleq\; \text{col}\left\{\alpha'_p\, g_{n,p'}^{(d)}\right\}} \tag{47.1200}$$

with an additional letter "v".

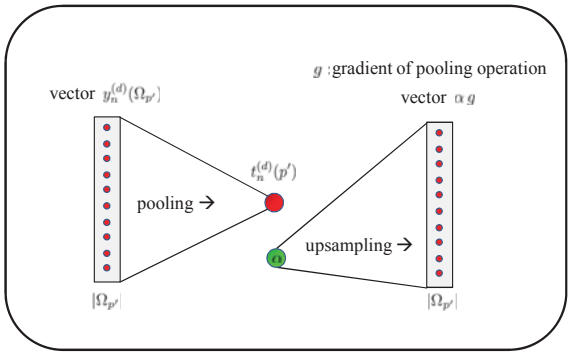

FIGURE 47.94 Upsampling transforms a scalar entry into a vector by scaling by the gradient vector of the pooling operation.

Ordering and Permutations

We need to introduce one final operation before moving on to describe the convolutional network more fully, along with an algorithm for training its weights.

Consider again a $P \times 1$ feature vector $y_n^{(d)}$. We explained under pooling how regions, $\Omega_{p'}$, of $y_n^{(d)}$ are pooled to generate the entries $t_n^{(d)}(p')$ in the reduced feature domain of size $P' \times 1$. We also explained that these regions are non-overlapping and, therefore, each entry of $y_n^{(d)}$ contributes towards generating a single entry of $t_n^{(d)}$. We continue to index the entries of $y_n^{(d)}$ by $p = 1, 2, \ldots, P$ and the entries of $t_n^{(d)}$ by $p' = 1, 2, \ldots, P'$.

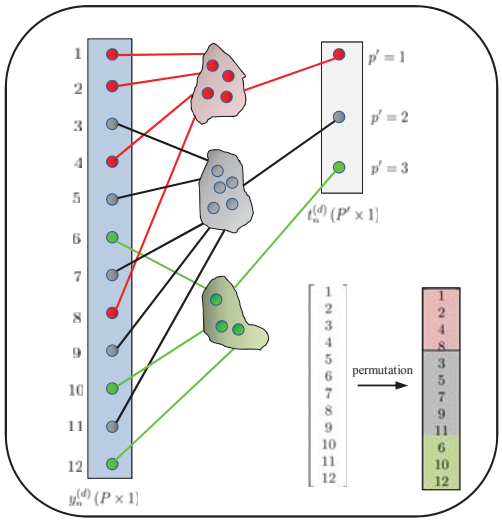

FIGURE 47.95 The entries of the original feature vector, $y_n^{(d)}$, can be re-ordered by means of a permutation matrix to group together the entries of the regions $\{\Omega_{p'}\}$ that generate the reduced feature vector, $t_n^{(d)}$.

Figure 47.95 shows one example with $P = 12$, $P' = 3$ and 3 partitions in $y_n^{(d)}$. The vectors shown in the lower right corner of the figure are meant to illustrate that the entries of the original feature vector, $y_n^{(d)}$, can be re-ordered by means of a permutation matrix to group together the entries of the regions $\{\Omega_{p'}\}$ that generate the reduced feature vector, $t_n^{(d)}$. Thus, observe, for this example that:

$$\begin{bmatrix} 1 \\ 2 \\ 4 \\ 8 \\ \hline 3 \\ 5 \\ 7 \\ 9 \\ 11 \\ \hline 6 \\ 10 \\ 12 \end{bmatrix} = \underbrace{\left[\begin{array}{cccccccccccc} \mathbf{1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \mathbf{1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & \mathbf{1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{1} & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{1} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{1} \end{array}\right]}_{\text{square permutation matrix}, T} \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \\ 10 \\ 11 \\ 12 \end{bmatrix} \tag{47.1201}$$

Therefore, we can associate with every pooling operation a uniquely-defined permutation matrix, which groups the indices $p = 1, 2, \ldots, P$ into the regions $\Omega_{p'}$, for $p' = 1, 2, \ldots, P'$. In the above expression, the first four rows of the permutation matrix extract the indices for region Ω_1, the next five rows extract the indices for region Ω_2, and the last three rows extract the indices for region Ω_3. If we denote the permutation matrix by T then, for any generic vector y, we can write

$$\boxed{y_o = \text{permute}(y) \iff y_o = Ty} \tag{47.1202}$$

to represent the operation that permutes the entries of y and generates a re-ordered vector, y_o. Observe that the rows of T are formed by stacking the row basis vectors $\{e_k^{\mathsf{T}}\}$ in the same order of the indices in the groups $\{\Omega_{p'}\}$:

$$T = \left[\begin{array}{c} e_1^{\mathsf{T}} \\ e_2^{\mathsf{T}} \\ e_4^{\mathsf{T}} \\ e_8^{\mathsf{T}} \\ \hline e_3^{\mathsf{T}} \\ e_5^{\mathsf{T}} \\ e_7^{\mathsf{T}} \\ e_9^{\mathsf{T}} \\ e_{11}^{\mathsf{T}} \\ \hline e_6^{\mathsf{T}} \\ e_{10}^{\mathsf{T}} \\ e_{12}^{\mathsf{T}} \end{array}\right], \quad (P \times P) \tag{47.1203}$$

It is easy to verify that $TT^{\mathsf{T}} = I$ so that $y = T^{\mathsf{T}} y_o$. We shall refer to the reverse operation by writing

$$\boxed{y = \text{permute}^{\#}(y_o) \iff y = T^{\mathsf{T}} y_o} \tag{47.1204}$$

47.25.4 Full Network

The network so far consists of an input layer, a correlation layer, and a pooling layer. Each layer has depth. For example, in the RGB model, the input layer has depth 3. The correlation and pooling layers have each depth D. We now proceed and treat the pooling layers as input maps to a new correlation layer, followed by its own pooling layer, and so forth.

Specifically, let us assume that we have a total of C such paired combinations of correlation/pooling layers in the network. We index the layers by the letter $c = 1, 2, \ldots, C$. Since we are now considering a multiplicity of correlation and pooling layers, we need to adjust our notation in order to incorporate a label for the layer. For example, the first correlation layer was assumed earlier to have depth D and its feature maps were denoted by the vectors $\{y_n^{(1)}, y_n^{(2)}, \ldots, y_n^{(D)}\}$. We now replace D by D_1 and denote these same feature maps by

$$\left\{y_n^{(1,1)}, y_n^{(2,1)}, \ldots, y_n^{(D_1,1)}\right\} \tag{47.1205}$$

with two superscripts: the first superscript continues to indicate the map index, running from $d = 1$ up to depth $d = D_1$, while the second superscript indicates the layer index for these maps; they are all part of the first layer $c = 1$. Likewise, the size P of each of these feature maps is replaced by P_1. More generally, for a layer of index c, we denote P by P_c, D by D_c, and the feature maps by

$$\left\{y_n^{(1,c)}, y_n^{(2,c)}, \ldots, y_n^{(D_c,c)}\right\}, \quad \text{(each of size } P_c \times 1) \tag{47.1206}$$

Likewise, we denote the reduced feature maps that are generated from these features by

$$\left\{t_n^{(1,c)}, t_n^{(2,c)}, \ldots, t_n^{(D_c,c)}\right\}, \quad \text{(each of size } P_c' \times 1) \tag{47.1207}$$

Each of these maps represents a vector of size $P_c' \times 1$ and is generated by using

$$t_n^{(d,c)} \;=\; \text{pool}\left(y_n^{(d,c)}\right), \quad d = 1, 2, \ldots, D_c \tag{47.1208}$$

These reduced features vectors can now be used to generate the input subvectors for the next correlation layer of index, $c + 1$. We denote these subvectors by

$$\left\{h_{n,p}^{(d,c)}\right\}_{p=1}^{P_{c+1}} \;=\; \text{partition}\left(t_n^{(d,c)}, P_{c+1}\right), \quad d = 1, 2, \ldots, D_c \tag{47.1209}$$

and there will be a total of P_{c+1} of them, each of dimensions $S_{c+1} \times 1$ — see Fig. 47.96:

$$h_{n,p}^{(d,c)} \in \mathbb{R}^{S_{c+1}} \tag{47.1210}$$

Thus, note that the number of size of these partitions are already prepared for the next layer, $c+1$. In the above expression, we are using the notation introduced in (47.1148) for partitioning an input vector into a collection of subvectors.

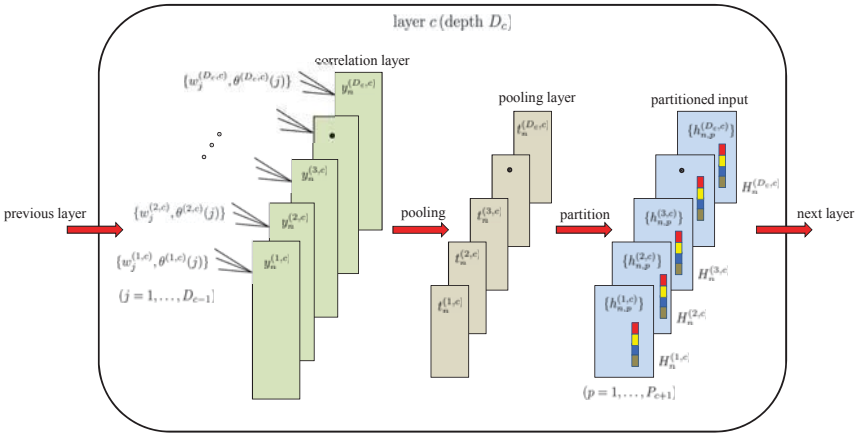

FIGURE 47.96 Each layer consists of D_c correlation layers, with feature maps $\{y_n^{(d,c)}\}$, D_c pooling layers with pooled vectors $\{t_n^{(d,c)}\}$, and D_c partitioned output vectors $\{h_{n,p}^{(d,c)}\}$ that will feed into the next layer; there are P_{c+1} partitions in each of these collections. Each feature map d operates on the partitioned vectors from the previous layer with weight vectors $\{w_j^{(d,c)} \in \mathbb{R}^{S_c}\}$ and scalar bias coefficients $\{\theta^{(d,c)}(j)\}$, where $j = 1, 2, \ldots, D_{c-1}$ is defined by the depth of the previous layer.

The feature maps for layer $c+1$ are subsequently generated as follows. For each map d, we collect the subvectors into matrices:

$$\begin{aligned} H_n^{(d,c)} &\triangleq \left[\begin{array}{cccc} h_{n,1}^{(d,c)} & h_{n,2}^{(d,c)} & \ldots & h_{n,P_{c+1}}^{(d,c)} \end{array}\right], \ (S_{c+1} \times P_{c+1}) \quad (47.1211) \\ &= d = 1, 2, \ldots, D_c \end{aligned}$$

and we denote the mask vectors and bias coefficients involved in the computation of feature map d in layer $c+1$ by $\{w_j^{(d,c+1)}, \theta^{(d,c+1)}(j)\}$. Then, in a manner similar to (47.1191a), the feature maps for layer $c+1$ are constructed s follows:

$$y_n^{(d,c+1)} \ = \ f\left(\sum_{j=1}^{D_c}\left(\left[H_n^{(j,c)}\right]^{\mathsf{T}} w_j^{(d,c+1)} - \theta^{(d,c+1)}(j)\mathbb{1}\right)\right), \ \ d = 1, 2, \ldots, D_{c+1} \quad (47.1212)$$

We may also consider limiting the sum of the layers to a subset of these layers, say, denoted by $\mathcal{D}_c$:

$$y_n^{(d,c+1)} \ = \ f\left(\sum_{j\in\mathcal{D}_c}\left(\left[H_n^{(j,c)}\right]^{\mathsf{T}} w_j^{(d,c+1)} - \theta^{(d,c+1)}(j)\mathbb{1}\right)\right), \ \ d = 1, 2, \ldots, D_c \quad (47.1213)$$

After a total of C pairs of correlation and pooling layers (e.g., $C = 3$), we will end up with the reduced feature maps

$$\left\{t_n^{(1,C)}, t_n^{(1,C)}, \ldots, t_n^{(D_C,C)}\right\}, \ \ \text{(each of size } P_C' \times 1) \quad (47.1214)$$

where D_C is the depth of the last layer. At this stage, the entries of these maps are fed into a fully connected feedforward neural network, say, consisting of one hidden layer and one output layer. This is achieved as follows. First, we concatenate the entries of the reduced maps (47.1214) into a feature vector h_n':

$$h_n' \ = \ \text{col}\left\{t_n^{(1,C)}, t_n^{(2,C)}, \ldots, t_n^{(D_C,C)}\right\} \quad (47.1215)$$

and then apply the entries of this vector as input signals into a feedforward network, in a manner similar to what we described earlier in Fig. 47.72. Adopting the same notation from that figure, we denote this input vector by $y_1 = h_n'$. Figure 47.97 then illustrates the resulting structure for a convolutional neural network involving $C = 3$ pairs of correlation and pooling layers, and a fully-connected feedforward network with two output nodes; more correlation/pooling layers can be used, as well as more output nodes. The structure in the figure is meant to serve as an example. It follows from this representation that the cascade of correlation/pooling layers can be interpreted as extracting pertinent features from the input data on the far left, and then feeding the resulting features, denoted by h_n', into the neural network at the far right for classification purposes.

47.25.5 Forward Propagation

Combining the analysis from the previous sections with the forward propagation algorithm (47.968) for neural networks, we arrive at the following description for the flow of signals through a convolutional neural network with C convolutional layers, each with depth D_c

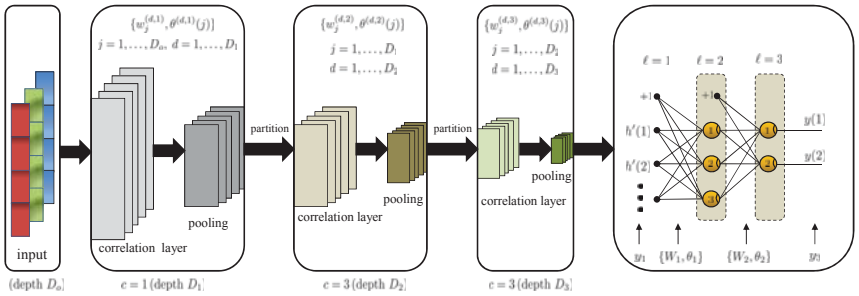

FIGURE 47.97 Example of a convolutional neural network consisting of 3 convolutional and pooling layers, and a fully-connected neural network with two output nodes.

and parameters $\{w_j^{(d,c)}, \theta^{(d,c)}(j)\}$, for $c = 1, 2, \ldots, C$. The input layer is assumed to have depth D_o with subvectors $\{h_{n,p}^{(d)}\}$ for $d = 1, \ldots, D_o$ and $p = 1 = 2, \ldots, P_o$. The feedforward network at the end of the structure has L layers (one input layer, $L-2$ hidden layers, and one output layer); its weight and bias vectors denoted by $\{W_\ell, \theta_\ell\}$, for $\ell = 1, 2, \ldots, L-1$. The activation function for the feedforward network need not be the same as the activation function used by the correlation layers.

Propagation through a convolutional neural network with C convolutional layers

start with $H_n^{(d,0)} = \begin{bmatrix} h_{n,1}^{(d)} & h_{n,2}^{(d)} & \ldots, h_{n,P_o}^{(d)} \end{bmatrix}$, $d = 1, 2, \ldots, D_0$

for $c = 1, \ldots, C$ (convolutional network) :

  for $d = 1, \ldots, D_c$:

$$z_n^{(d,c)} = \sum_{j=1}^{D_{c-1}} \left(\left[H_n^{(j,c-1)} \right]^{\mathsf{T}} w_j^{(d,c)} - \theta^{(d,c)}(j) \mathbb{1} \right)$$

$$y_n^{(d,c)} = f\left(z_n^{(d,c)} \right)$$

$$t_n^{(d,c)} = \text{pool}\left(y_n^{(d,c)} \right)$$

$$\left\{ h_{n,p}^{(d,c)} \right\}_{p=1}^{P_{c+1}} = \text{partition}\left(t_n^{(d,c)}, P_{c+1} \right) \tag{47.1216}$$

$$H_n^{(d,c)} = \begin{bmatrix} h_{n,1}^{(d,c)} & h_{n,2}^{(d,c)} & \ldots, h_{n,P_{c+1}}^{(d,c)} \end{bmatrix}, \quad d = 1, 2, \ldots, D_c$$

  end

end

$$y_{1,n} = \text{col}\left\{ t_n^{(1,C)}, t_n^{(2,C)}, \ldots, t_n^{(D_C,C)} \right\}$$

for $\ell = 1, \ldots, L-1$ (feedforward network) :

$$z_{\ell+1,n} = W_\ell y_{\ell,n} - \theta_\ell$$

$$y_{\ell+1,n} = f\left(z_{\ell+1,n} \right)$$

end

$$y = y_L$$

Mapping of Indices

In the next section we will derive a training algorithm for convolutional networks. First, however, we introduce the following notation to facilitate the presentation. Observe in the listing above that the pooling vectors $\{t_n^{(d,C)}\}$ at the last convolutional layer are concate-

nated into a vector $y_{1,n}$, which is then fed into the first layer of feedforward network. We now highlight two facts.

First, since each vector $t_n^{(d,C)}$ has dimensions $P'_C \times 1$, if we pick an arbitrary entry of index p' from $t_n^{(d,C)}$, its location within $y_{1,n}$ will be indexed by

$$i'_p \;=\; (d-1)P'_C + p' \tag{47.1217}$$

Obviously, the size of $y_{1,n}$ is $D_C P'_C \times 1$. We will need to use both indices (p', i'_p) in the derivation that follows — see, e.g., (47.1237) and the expressions afterwards. What is important to note is that these indices are referring to the *same* variable, with one index computed within $t_n^{(d,C)}$ while the other index is computed within $y_{1,n}$.

Second, the weighting matrix W_1 at the entry of the feedforward network contains the weight vectors that scale the entries of $y_{1,n}$ and transform this vector into the pre-activation vector $z_{2,n}$ at the input of the second layer. From the notation used in Sec. 47.23, the columns of W_1 are denoted by

$$W_1 = \left[\begin{array}{ccccc} w_1^{(1)} & w_2^{(1)} & w_3^{(1)} & \ldots & w_{D_C P'_C}^{(1)} \end{array} \right] \tag{47.1218}$$

where each column $w_i^{(\ell)}$ has the following interpretation:

$$w_i^{(1)} = \text{weight vector emanating from node } i \text{ in } y_{1,n} \tag{47.1219}$$

In this way, the first column of W_1 is the weight vector that emanates from the first entry of $y_{1,n}$, which is the first entry of $t_n^{(1,C)}$. Thus, the first P'_C columns of W_1 define the column vectors that emanate from the entries of $t_n^{(1,C)}$. The next P'_C columns of W_1 define the column vectors that emanate from the entries of $t_n^{(2,C)}$, and so on. In the derivation that follows, we will need to refer to these weight vectors that emanate from the separate pooling vectors. For this purpose, we introduce the notation $W_1^{(1)}$ to refer to the first P'_C vectors of W_1, which relate to $t_n^{(1,C)}$. Likewise for $W_2^{(2)}$ and, more generally, for $t_n^{(d,C)}$:

$$W_1^{(d)} \;\triangleq\; [W_1]_{:,(d-1)P'_C+1:dP'_C} \tag{47.1220}$$

where the notation means that $W_1^{(d)}$ will consist of all columns of indices $(d-1)P'_C + 1$ through dP'_C from W_1.

47.25.6 Training Algorithm

We now turn to the problem of training the convolutional neural network. It turns out that the same backpropagation structure from Sec. 47.23 is applicable with proper adjustments.

Thus, consider again a convolutional neural network consisting of C pairs of correlation and pooling layers followed by a fully-connected feedforward neural network with L layers (input layer, hidden layers, and output layer). Some typical values are $C = 6$ and $L = 3$. We index the correlation layers by $c = 1, \ldots, C$ and the feedforward layers by $\ell = 1, \ldots, L$. We recall the notation used in Sec. 47.23 to denote the variables associated with

the feedforward network:

$$
\begin{aligned}
w_{ij}^{(\ell)} &= \text{weight from node } i \text{ in layer } \ell \text{ to node } j \text{ in layer } \ell+1 && \text{(47.1221a)}\\
w_{i}^{(\ell)} &= \text{weight vector emanating from node } i \text{ in layer } \ell && \text{(47.1221b)}\\
-\theta_\ell(j) &= \text{weight from bias source in layer } \ell \text{ to node } j \text{ in layer } \ell+1 && \text{(47.1221c)}\\
y_\ell(j) &= \text{output of node } j \text{ at layer } \ell && \text{(47.1221d)}\\
z_\ell(n) &= \text{output of node } j \text{ at layer } \ell \text{ prior to the activation function} && \text{(47.1221e)}\\
n_\ell &= \text{number of nodes in layer } \ell, \text{ excluding the bias source} && \text{(47.1221f)}\\
W_\ell &= \left[\; w_1^{(\ell)} \;\middle|\; w_2^{(\ell)} \;\middle|\; \ldots \;\middle|\; w_{n_{\ell+1}}^{(\ell)} \;\right], \quad (n_{\ell+1}\times n_\ell) && \text{(47.1221g)}\\
\theta_\ell &= \text{col}\left\{\theta_\ell(1),\theta_\ell(2),\ldots,\theta_\ell(n_{\ell+1})\right\}, \quad (n_{\ell+1}\times 1) && \text{(47.1221h)}
\end{aligned}
$$

where (W_ℓ,θ_ℓ) denote the weight combination matrix and the bias vector used to generate the output signals for layer $\ell+1$ in the feedforward network. Similarly, for the convolutional network we recall that:

$$
\begin{aligned}
w_j^{(d,c)} &= j-\text{th weight vector to generate map } d \text{ in layer } c && \text{(47.1222a)}\\
\theta^{(d,c)}(j) &= j-\text{th bias coefficient to generate map } d \text{ in layer } c && \text{(47.1222b)}\\
z_n^{(d,c)} &= \text{feature map } d \text{ prior to activation in layer } c && \text{(47.1222c)}\\
y_n^{(d,c)} &= \text{feature map } d \text{ after activation in layer } c && \text{(47.1222d)}\\
t_n^{(d,c)} &= \text{reduced feature map } d \text{ in layer } c && \text{(47.1222e)}\\
H_n^{(d,c)} &= \text{feature matrix for map } d \text{ in layer } c && \text{(47.1222f)}
\end{aligned}
$$

For ease of reference, we collect the weight vectors and bias coefficients associated with each map d in layer c into the quantities

$$
\begin{aligned}
W^{(d,c)} &\triangleq \text{col}\left\{w_1^{(d,c)}, w_2^{(d,c)},\ldots,w_{D_{c-1}}^{(d,c)}\right\}, \quad (1\times S_cD_{c-1}) && \text{(47.1223a)}\\
\theta^{(d,c)} &\triangleq \text{col}\left\{\theta^{(d,c)}(1),\theta^{(d,c)}(2),\ldots,\theta^{(d,c)}(D_{c-1})\right\}, \quad (D_{c-1}\times 1) && \text{(47.1223b)}
\end{aligned}
$$

so that all weight vectors and bias coefficients that are associated with layer c are represented by the quantities

$$
\begin{aligned}
W_c &\triangleq \left[\; W^{(1,c)} \quad W^{(2,c)} \quad \ldots \quad W^{(D_c,c)} \;\right], \quad (S_cD_{c-1}\times D_c) && \text{(47.1224a)}\\
\Theta_c &\triangleq \left[\; \theta^{(1,c)} \quad \theta^{(2,c)} \quad \ldots \quad \theta^{(D_c,c)} \;\right], \quad (D_{c-1}\times D_c) && \text{(47.1224b)}
\end{aligned}
$$

Likewise, we collect the feature maps that are associated with layer c at time n into the matrix:

$$
H_{n,c} \triangleq \left[\begin{array}{c} H_n^{(1,c)} \\ H_n^{(2,c)} \\ \vdots \\ H_n^{(D_c,c)} \end{array}\right], \quad (D_cS_{c+1}\times P_{c+1}) \tag{47.1225}
$$

We also denote the individual entries of the vector maps $\{z_{n,d}^{(c)}, y_{n,d}^{(c)}, t_{n,d}^{(c)}\}$ by

$$z_n^{(d,c)} = \mathrm{col}\left\{z_{n,p}^{(d,c)}\right\}, \quad 1 \le p \le P_c \tag{47.1226a}$$

$$y_n^{(d,c)} = \mathrm{col}\left\{y_{n,p}^{(d,c)}\right\}, \quad 1 \le p \le P_c \tag{47.1226b}$$

$$t_n^{(d,c)} = \mathrm{col}\left\{t_{n,p}^{(d,c)}\right\}, \quad 1 \le p \le P_c' \tag{47.1226c}$$

where we are denoting the sizes of $\{z_n^{(d,c)}, y_n^{(d,c)}\}$ by P_c in layer c, and similarly for $t_n^{(d,c)}$ using P_c'.

Empirical Risk

We now refer to the setting described in Sec. 47.23 and consider a collection of N training pairs $\{\gamma_n, h_n\}$, where $\gamma_n \in \mathbb{R}^Q$ and $h_n \in \mathbb{R}^M$. The objective is to train the network, i.e., to select its parameters $\{W_\ell, \theta_\ell, W_c, \theta_c\}$, in order for the mapping from the input space, $h \in \mathbb{R}^M$, to the output space, $y \in \mathbb{R}^Q$, to approximate well the mapping from h to γ that is reflected in the training data. To do so, we consider an empirical risk optimization problem similar to (47.976), namely,

$$\left\{ \begin{array}{c} W_\ell^\circ, \theta_\ell^\circ \\ W_c^\circ, \theta_c^\circ \end{array} \right\} \triangleq \begin{array}{c} \mathrm{argmin} \\ \left(\begin{array}{c} W_\ell, \theta_\ell \\ W_c, \Theta_c \end{array} \right) \end{array} \frac{1}{N} \left[\sum_{c=1}^{C} \rho_1 \|W_c\|_{\mathrm{F}}^2 + \sum_{\ell=1}^{L-1} \rho_2 \|W_\ell\|_{\mathrm{F}}^2 + \sum_{n=0}^{N-1} \|\gamma_n - y_n\|^2 \right] \tag{47.1227}$$

where the first two terms apply regularization to the sum of the squared Frobenius norms of the weight matrices in the correlation and feedforward layers. Other forms of regularization are possible, including ℓ_1-regularization, as well as other choices for the risk function — see, e.g., Probs. 47.169–47.174. Observe from (47.1227) that regularization is not applied to the bias vectors $\{\theta_\ell, \Theta_c\}$; these entries are embedded in the output signals $\{y_n\}$.

We may also consider an alternative to (47.1227) where the scaling by $1/N$ appears multiplying the last term only, namely,

$$\left\{ \begin{array}{c} W_\ell^\circ, \theta_\ell^\circ \\ W_c^\circ, \theta_c^\circ \end{array} \right\} \triangleq \begin{array}{c} \mathrm{argmin} \\ \left(\begin{array}{c} W_\ell, \theta_\ell \\ W_c, \Theta_c \end{array} \right) \end{array} \sum_{c=1}^{C} \rho_1 \|W_c\|_{\mathrm{F}}^2 + \sum_{\ell=1}^{L-1} \rho_2 \|W_\ell\|_{\mathrm{F}}^2 + \frac{1}{N} \sum_{n=0}^{N-1} \|\gamma_n - y_n\|^2 \tag{47.1228}$$

The analysis that follows is equally applicable to this case with minimal adjustments, especially since we will be dealing with stochastic-gradient and batch algorithms. For this reason, it is sufficient to continue with the empirical risk

$$\boxed{J_{\mathrm{emp}}(W, \theta) \triangleq \sum_{c=1}^{C} \rho_1 \|W_c\|_{\mathrm{F}}^2 + \sum_{\ell=1}^{L-1} \rho_2 \|W_\ell\|_{\mathrm{F}}^2 + \frac{1}{N} \sum_{n=0}^{N-1} \|\gamma_n - y_n\|^2} \tag{47.1229}$$

Although this cost is dependent on all unknowns $\{W_\ell, \theta_\ell, W_c, \Theta_c\}$, we are denoting the dependency generically by using the letters W and θ as arguments for $J_{\mathrm{emp}}(\cdot)$. In order to implement iterative procedures for minimizing $J_{\mathrm{emp}}(W, \theta)$, e.g., either in batch gradient-descent form or in stochastic gradient-descent form, we need to know how to evaluate the

gradients of $J_{\rm emp}(W,\theta)$ relative to the individual entries of $\{W_\ell, \theta_\ell, W_c, \Theta_c\}$, namely, we need to evaluate:

$$\frac{\partial J_{\rm emp}(W,\theta)}{\partial w_{ij}^{(\ell)}} \quad \text{and} \quad \frac{\partial J_{\rm emp}(W,\theta)}{\partial \theta_\ell(j)} \tag{47.1230a}$$

$$\frac{\partial J_{\rm emp}(W,\theta)}{\partial w_j^{(d,c)}(s)} \quad \text{and} \quad \frac{\partial J_{\rm emp}(W,\theta)}{\partial \theta^{(d,c)}(j)} \tag{47.1230b}$$

where $w_j^{(d,c)}(s)$ denotes the $s-$th entry of vector $w_j^{(d,c)}$. The backpropagation algorithm is the procedure that enables us to compute these gradients in an effective manner, as we proceed to explain.

Sensitivity Factors over Feedforward Network

We denote the output vectors for the $\ell-$th layer in the feedforward network, before and after the activation function, by $\{z_{\ell,n}, y_{\ell,n}\}$. We denote the overall output vectors for the feedforward network by $\{z_n, y_n\}$, without a subscript L, and refer to their individual entries by

$$y_n = \begin{bmatrix} y_n(1) \\ y_n(2) \\ \vdots \\ y_n(Q) \end{bmatrix}, \quad z_n = \begin{bmatrix} z_n(1) \\ z_n(2) \\ \vdots \\ z_n(Q) \end{bmatrix} \tag{47.1231}$$

In a manner similar to (47.1004), we introduce the scalar sensitivity factor:

$$\delta_{\ell,n}(j) \quad \triangleq \quad \frac{\partial \|\gamma_n - y_n\|^2}{\partial z_{\ell,n}(j)}, \quad \ell = 2,3,\ldots,L \tag{47.1232}$$

which measures how the (unregularized portion of the) risk varies in response to changes in the pre-activation signal, $z_{\ell,n}(j)$. We derive a recursive update for this sensitivity factor as follows.

Let $\delta_{L,n}$ denote the vector that collects the sensitivities relative to the output layer, $\ell = L$. Then, we already know from (47.1006) that this vector is given by

$$\boxed{\delta_{L,n} \;=\; 2(y_n - \gamma_n) \odot f'(z_n)} \tag{47.1233}$$

Likewise, let $\delta_{\ell,n}$ denote the vector that collects the sensitivities $\{\delta_{\ell,n}(j)\}$ in layer ℓ of the feedforward network. Again, the same argument that led to (47.1010) holds so that this vector satisfies:

$$\boxed{\delta_{\ell,n} = f'\left(z_{\ell,n}\right) \odot \left(W_\ell^{\mathsf{T}} \delta_{\ell+1,n}\right)} \tag{47.1234}$$

This recursion runs backwards from $\ell = L-1$ down to $\ell = 2$. Now we transition to propagating the sensitivity factors over the convolutional network.

Sensitivity Factor for Last Correlation Layer

First, and in a manner similar to (47.1004), we introduce the scalar sensitivity factor:

$$\delta_n^{(d,c)}(j) \quad \triangleq \quad \frac{\partial \|\gamma_n - y_n\|^2}{\partial z_n^{(d,c)}(j)}, \quad c = 1,2,\ldots,C \tag{47.1235}$$

which measures how the (unregularized portion of the) risk varies in response to changes in the pre-activation signal, $z_n^{(d,c)}(j)$. We derive a recursive update for this factor as follows.

Now, at the boundary layer $c = C$, which links the convolutional and feedforward networks, we have:

$$\begin{aligned}
\delta_n^{(d,C)}(j) &\triangleq \frac{\partial \|\gamma_n - y_n\|^2}{\partial z_n^{(d,C)}(j)} \\
&= \sum_{k=1}^{n_2} \frac{\partial \|\gamma_n - y_n\|^2}{\partial z_{2,n}(k)} \frac{\partial z_{2,n}(k)}{\partial z_n^{(d,C)}(j)} \\
&= \sum_{k=1}^{n_2} \delta_{2,n}(k) \frac{\partial z_{2,n}(k)}{\partial z_n^{(d,C)}(j)} \qquad (47.1236)
\end{aligned}$$

where the right-most term involves differentiating the pre-activation output of node k in layer 2 in the feedforward network relative to the $j-$th pre-activation output in feature map d of the last correlational layer, C. The last term in (47.1236) can be evaluated as follows:

$$\begin{aligned}
\frac{\partial z_{2,n}(k)}{\partial z_n^{(d,C)}(j)} &= \sum_{p'=1}^{P_c'} \frac{\partial z_{2,n}(k)}{\partial t_n^{(d,C)}(p')} \frac{\partial t_n^{(d,C)}(p')}{\partial z_n^{(d,C)}(j)} \\
&= \sum_{p'=1}^{P_c'} \frac{\partial z_{2,n}(k)}{\partial t_n^{(d,C)}(p')} \left(\sum_{p \in \Omega_{p'}} \frac{\partial t_n^{(d,C)}(p')}{\partial y_n^{(d,C)}(p)} \frac{\partial y_n^{(d,C)}(p)}{\partial z_n^{(d,C)}(j)} \right) \\
&\overset{(a)}{=} \sum_{p' | j \in \Omega_{p'}} \frac{\partial z_{2,n}(k)}{\partial t_n^{(d,C)}(p')} \frac{\partial t_n^{(d,C)}(p')}{\partial y_n^{(d,C)}(j)} f'\left(z_n^{(d,C)}(j)\right) \\
&\overset{(b)}{=} \sum_{p' | j \in \Omega_{p'}} w_{i_{p'},k}^{(1)} \frac{\partial t_n^{(d,C)}(p')}{\partial y_n^{(d,C)}(j)} f'\left(z_n^{(d,C)}(j)\right) \qquad (47.1237)
\end{aligned}$$

where step (b) uses (47.1217) and the scalar $w_{i_{p'},k}^{(1)}$ refers to the value of the weight at the input of the feedforward network that maps entry i_p' to the $k-$th node. Moreover, step (a) is because only $y_n^{(d,C)}(j)$ depends on $z_n^{(d,C)}(j)$, and the notation $\{p' | j \in \Omega_{p'}\}$ means that the sum is over all values p' for which $j \in \Omega_{p'}$. This condition is including in the summation all signals $t_n^{(d,C)}(p')$ that are influenced by $z_n^{(d,C)}(j)$. However, we assumed earlier that the regions $\Omega_{p'}$ do not overlap with each other. This means that each $y_n^{(d,C)}(j)$ and, consequently, each $z_n^{(d,C)}(j)$ influence only one signal $t_n^{(d,C)}(p')$. Let p'' denote the index of the signal that is influenced by $z_n^{(d,C)}(j)$, and let i_p'' be the corresponding mapping within h_n' according to (47.1217). Then, we can remove the sum over p' from the last expression and write it instead in the form:

$$\frac{\partial z_{2,n}(k)}{\partial z_n^{(d,C)}(j)} = w_{i_{p''},k}^{(1)} \frac{\partial t_n^{(d,C)}(p'')}{\partial y_n^{(d,C)}(j)} f'\left(z_n^{(d,C)}(j)\right) \qquad (47.1238)$$

Consequently, substituting into (47.1236) gives

$$\begin{aligned}
\delta_n^{(d,C)}(j) &= \sum_{k=1}^{n_2} \delta_{2,n}(k) \left(w_{i_p'',k}^{(1)} \, \frac{\partial t_n^{(d,C)}(p'')}{\partial y_n^{(d,C)}(j)} \, f'\left(z_n^{(d,C)}(j)\right) \right) \\
&= f'\left(z_n^{(d,C)}(j)\right) \left(\sum_{k=1}^{n_2} \delta_{2,n}(k) w_{i_p'',k}^{(1)} \right) \frac{\partial t_n^{(d,C)}(p'')}{\partial y_n^{(d,C)}(j)} \\
&= f'\left(z_n^{(d,C)}(j)\right) \left(\left(w_{i_p''}^{(1)}\right)^{\mathsf{T}} \delta_{2,n} \right) \frac{\partial t_n^{(d,C)}(p'')}{\partial y_n^{(d,C)}(j)} \qquad (47.1239)
\end{aligned}$$

in terms of the inner product between the last sensitivity vector $\delta_{2,n}$ originating from the feedforward network and the weight vector, available at time $n-1$, and originating from node i_p'' in the input layer to the feedforward network (as defined by (47.1221b)). In order to rewrite the above result in a more compact vector form, let us illustrate its meaning by referring back to the situation shown in Fig. 47.95 and expression (47.1201). In the context of that example, we have that the index j runs over $j = 1, 2, \ldots, 12$. Moreover, the index p'' that corresponds to each j is given by the transformation (47.1201). For example, from that relation, we know that entry $j = 3$ influences $p'' = 2$. Therefore, if we stack the entries $\delta_n^{(d,C)}(j)$ on top of each other, for $j = 1, 2, \ldots, 12$ and use (47.1201) we would obtain:

$$\underbrace{\begin{bmatrix} \delta_n^{(d,C)}(1) \\ \delta_n^{(d,C)}(2) \\ \delta_n^{(d,C)}(3) \\ \delta_n^{(d,C)}(4) \\ \delta_n^{(d,C)}(5) \\ \delta_n^{(d,C)}(6) \\ \delta_n^{(d,C)}(7) \\ \delta_n^{(d,C)}(8) \\ \delta_n^{(d,C)}(9) \\ \delta_n^{(d,C)}(10) \\ \delta_n^{(d,C)}(11) \\ \delta_n^{(d,C)}(12) \end{bmatrix}}_{\triangleq \, \delta_n^{(d,C)}} = \underbrace{\begin{bmatrix} f'\left(z_n^{(d,C)}(1)\right) \\ f'\left(z_n^{(d,C)}(2)\right) \\ f'\left(z_n^{(d,C)}(3)\right) \\ f'\left(z_n^{(d,C)}(4)\right) \\ f'\left(z_n^{(d,C)}(5)\right) \\ f'\left(z_n^{(d,C)}(6)\right) \\ f'\left(z_n^{(d,C)}(7)\right) \\ f'\left(z_n^{(d,C)}(8)\right) \\ f'\left(z_n^{(d,C)}(9)\right) \\ f'\left(z_n^{(d,C)}(10)\right) \\ f'\left(z_n^{(d,C)}(11)\right) \\ f'\left(z_n^{(d,C)}(12)\right) \end{bmatrix}}_{\triangleq \, f'\left(z_n^{(d,C)}\right)} \odot \underbrace{\begin{bmatrix} \left(w_{i_1''}^{(1)}\right)^{\mathsf{T}} \delta_{2,n} \\ \left(w_{i_1''}^{(1)}\right)^{\mathsf{T}} \delta_{2,n} \\ \left(w_{i_2''}^{(1)}\right)^{\mathsf{T}} \delta_{2,n} \\ \left(w_{i_1''}^{(1)}\right)^{\mathsf{T}} \delta_{2,n} \\ \left(w_{i_2''}^{(1)}\right)^{\mathsf{T}} \delta_{2,n} \\ \left(w_{i_3''}^{(1)}\right)^{\mathsf{T}} \delta_{2,n} \\ \left(w_{i_2''}^{(1)}\right)^{\mathsf{T}} \delta_{2,n} \\ \left(w_{i_1''}^{(1)}\right)^{\mathsf{T}} \delta_{2,n} \\ \left(w_{i_2''}^{(1)}\right)^{\mathsf{T}} \delta_{2,n} \\ \left(w_{i_3''}^{(1)}\right)^{\mathsf{T}} \delta_{2,n} \\ \left(w_{i_2''}^{(1)}\right)^{\mathsf{T}} \delta_{2,n} \\ \left(w_{i_3''}^{(1)}\right)^{\mathsf{T}} \delta_{2,n} \end{bmatrix}}_{\triangleq \, x} \odot \begin{bmatrix} \frac{\partial t_n^{(d,C)}(1)}{\partial y_n^{(d,C)}(1)} \\ \frac{\partial t_n^{(d,C)}(1)}{\partial y_n^{(d,C)}(2)} \\ \frac{\partial t_n^{(d,C)}(2)}{\partial y_n^{(d,C)}(3)} \\ \frac{\partial t_n^{(d,C)}(1)}{\partial y_n^{(d,C)}(4)} \\ \frac{\partial t_n^{(d,C)}(2)}{\partial y_n^{(d,C)}(5)} \\ \frac{\partial t_n^{(d,C)}(3)}{\partial y_n^{(d,C)}(6)} \\ \frac{\partial t_n^{(d,C)}(2)}{\partial y_n^{(d,C)}(7)} \\ \frac{\partial t_n^{(d,C)}(1)}{\partial y_n^{(d,C)}(8)} \\ \frac{\partial t_n^{(d,C)}(2)}{\partial y_n^{(d,C)}(9)} \\ \frac{\partial t_n^{(d,C)}(3)}{\partial y_n^{(d,C)}(10)} \\ \frac{\partial t_n^{(d,C)}(2)}{\partial y_n^{(d,C)}(11)} \\ \frac{\partial t_n^{(d,C)}(3)}{\partial y_n^{(d,C)}(12)} \end{bmatrix} \qquad (47.1240)$$

We are denoting the Hadamard product of the last two vectors by the generic letter x, which we now identify. Following (47.1220), let $W_1^{(d)}$ collect the columns in W_1 that correspond

to $t_n^{(d,C)}$ and consider the vector

$$v \triangleq \left(W_1^{(d)}\right)^{\mathsf{T}} \delta_{2,n} \tag{47.1241}$$

We next upsample this vector using construction (47.1200), i.e., compute

$$\text{upv}\left(\left(W_1^{(d)}\right)^{\mathsf{T}} \delta_{2,n}\right) \tag{47.1242}$$

where the notation $\text{upv}(\cdot)$ is now upsampling the individual entries of its vector argument, denoted by $v = \text{col}\{\alpha_j\}$, according to

$$\boxed{\text{upv}(v) \;=\; \text{col}\left\{\alpha_j \, g_{n,j}^{(d,C)}\right\}} \tag{47.1243}$$

by using the gradient vectors of the entries of $t_n^{(d,C)}$. It can then be verified that the term x in (47.1240) is given by — see Prob. 47.164:

$$x \;=\; \text{permute}^{\#}\left(\text{upv}\left(\left(W_1^{(d)}\right)^{\mathsf{T}} \delta_{2,n}\right)\right) \tag{47.1244}$$

so that

$$\boxed{\delta_n^{(d,C)} = f'\left(z_n^{(d,C)}\right) \odot \text{permute}^{\#}\left(\text{upv}\left(\left(W_1^{(d)}\right)^{\mathsf{T}} \delta_{2,n}\right)\right), \; d = 1, 2, \ldots, D_C} \tag{47.1245}$$

where the operations of upsampling and permutation were defined earlier in (47.1243) and (47.1204), respectively.

Sensitivity Factor for Earlier Correlation Layers

We now continue to the earlier correlation layers by noting first that

$$\begin{aligned}
\delta_n^{(d,c)}(j) \;&\triangleq\; \frac{\partial \|\gamma_n - y_n\|^2}{\partial z_n^{(d,c)}(j)} \\
&=\; \sum_{i=1}^{P_{c+1}} \sum_{d'=1}^{D_{c+1}} \frac{\partial \|\gamma_n - y_n\|^2}{\partial z_n^{(d',c+1)}(i)} \, \frac{\partial z_n^{(d',c+1)}(i)}{\partial z_n^{(d,c)}(j)} \\
&=\; \sum_{i=1}^{P_{c+1}} \sum_{d'=1}^{D_{c+1}} \delta_n^{(d',c+1)}(i) \frac{\partial z_n^{(d',c+1)}(i)}{\partial z_n^{(d,c)}(j)} \\
&=\; \sum_{i=1}^{P_{c+1}} \sum_{d'=1}^{D_{c+1}} \delta_n^{(d',c+1)}(i) \frac{\partial z_n^{(d',c+1)}(i)}{\partial y_n^{(d,c)}(j)} \frac{\partial y_n^{(d,c)}(j)}{\partial z_n^{(d,c)}(j)} \\
&=\; \sum_{i=1}^{P_{c+1}} \sum_{d'=1}^{D_{c+1}} \delta_n^{(d',c+1)}(i) \frac{\partial z_n^{(d',c+1)}(i)}{\partial y_n^{(d,c)}(j)} f'\left(z_n^{(d,c)}(j)\right)
\end{aligned} \tag{47.1246}$$

Let us now examine the partial derivative that remains in the above expression. Using the chain rule for differentiation, it holds that:

$$\begin{aligned}\frac{\partial z_n^{(d',c+1)}(i)}{\partial y_n^{(d,c)}(j)} &= \sum_{d''=1}^{D_c}\sum_{p'=1}^{P_c'}\frac{\partial z_n^{(d',c+1)}(i)}{\partial t_n^{(d'',c)}(p')}\,\frac{\partial t_n^{(d'',c)}(p')}{\partial y_n^{(d,c)}(j)} \\ &\overset{(a)}{=} \sum_{p'=1}^{P_c'}\frac{\partial z_n^{(d',c+1)}(i)}{\partial t_n^{(d,c)}(p')}\,\frac{\partial t_n^{(d,c)}(p')}{\partial y_n^{(d,c)}(j)} \end{aligned} \tag{47.1247}$$

where (a) is because only $t_n^{(d,c)}(p')$ depends on $y_n^{(d,c)}(j)$ since pooling is assumed to be applied on each layer d separately, and we assumed earlier that the regions $\Omega_{p'}$ do not overlap with each other. This latter condition means that each $y_n^{(d,c)}(j)$ influences only one signal $t_n^{(d,c)}(p')$. Let p'' denote the index of the signal that is influenced by $y_n^{(d,c)}(j)$. Then, we can remove the sum over p' from the last expression and write it instead in the form (this is the same argument we used before to write (47.1238)):

$$\frac{\partial z_n^{(d',c+1)}(i)}{\partial y_n^{(d,c)}(j)} = \frac{\partial z_n^{(d',c+1)}(i)}{\partial t_n^{(d,c)}(p'')}\,\frac{\partial t_n^{(d,c)}(p'')}{\partial y_n^{(d,c)}(j)} \tag{47.1248}$$

The last term on the right-hand side is related to the gradient of the pooling operation, which we already know how to evaluate. Let us examine the term preceding it. Recall that the mapping from $t_n^{(d,c)}(p'')$ in layer c to $z_n^{(d',c+1)}(i)$ in layer $c+1$ involves the steps shown in Fig. 47.98. The $j-$th entry in $y_n^{(d,c)}$ influences a single entry, of index p'', in the pooled feature vector, $t_n^{(d,c)}$. This entry can in turn be mapped into several of the P_{c+1} subvectors, $h_{n,p}^{(d,c)}$ in the $d-$th feature map during partitioning, as exemplified by the bright green entries. The subvectors $\{h_{n,p}^{(d,c)}\}$ are subsequently processed by the *same* filter $w_d^{(d',c+1)}$ to contribute to the generation of $z_n^{(d',c+1)}(i)$. Observe that this filter is employed P_{c+1} times. Observe also that, while the other feature maps in layer $c+1$ contribute to the generation of the same signals $z_n^{(d',c+1)}(i)$ through their weight vectors, these vectors are not necessary for the current argument and, therefore, they are not shown in the figure.

Returning to (47.1246) we now have

$$\delta_n^{(d,c)}(j) = \sum_{d'=1}^{D_{c+1}} f'\left(z_n^{(d,c)}(j)\right)\left(\sum_{i=1}^{P_{c+1}}\delta_n^{(d',c+1)}(i)\,\frac{\partial z_n^{(d',c+1)}(i)}{\partial t_n^{(d,c)}(p'')}\right)\frac{\partial t_n^{(d,c)}(p'')}{\partial y_n^{(d,c)}(j)} \tag{47.1249}$$

Let us examine the inner sum over the index i. This sum depends on j, i.e., it can assume different values for different $j = 1, 2, \ldots, P_c$. This is because the sum depends on j through the variable p''; if desired, we can indicate the dependence of p'' explicitly on j by writing p''_j. Let

$$b_n^{(d,d',c)}(p'') \triangleq \sum_{i=1}^{P_{c+1}}\delta_n^{(d',c+1)}(i)\frac{\partial z_n^{(d',c+1)}(i)}{\partial t_n^{(d,c)}(p'')}, \quad p'' = 1, 2, \ldots, P_c' \tag{47.1250}$$

where the value of $b_n^{(d,d',c)}(p'')$ depends on both d and d'; hence, the addition of d to the superscript list. Once we determine the value of $b_n^{(d,d',c)}(p'')$, for any p'', that value would be the same for all indices j that contribute to the same p''. These indices are determined

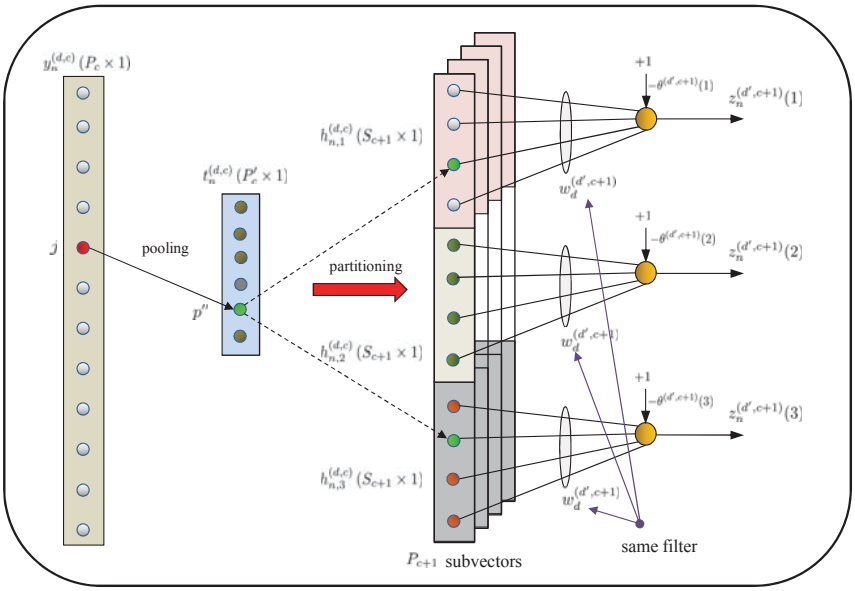

FIGURE 47.98 The $j-$th entry in $y_n^{(d,c)}$ influences a single entry, of index p'', in the pooled feature vector, $t_n^{(d,c)}$. This entry can in turn be map into several subvectors, $h_{n,p}^{(d,c)}$ in the $d-$th feature map during partitioning, as exemplified by the bright green entries. The subvectors $\{h_{n,p}^{(d,c)}\}$ are subsequently processed by the *same* filter $w_d^{(d',c+1)}$ to contribute to the generation of $z_n^{(d',c+1)}(i)$.

by the mapping defined by the pooling operation. For example, if we refer to the situation described by (47.1201) and Fig. 47.95, we know from the partitioning on the left-hand side of (47.1201) that the first entry in $t_n^{(d,c)}$ corresponding to $p''=1$ is determined by the indices $j \in \{1,2,4,8\}$ in the original vector $y_n^{(d,c)}$ prior to pooling. In other words, the mapping between the variables j and p'' is known. We now illustrate the computation of the variables $b_n^{(d,d',c)}(p'')$ by considering the simplified example shown in Fig. 47.99.

In the example, we have $P_{c+1}=4$ partitions and weight vector, $w_d^{(d',c+1)}$, with two scalar entries denoted simply by (w_1, w_2). The variable p'' runs from $p''=1$ to $p''=6$. For this situation, we can easily conclude from the figure by inspection that

$$\underbrace{\begin{bmatrix} b_n^{(d,d',c)}(1) \\ b_n^{(d,d',c)}(2) \\ b_n^{(d,d',c)}(3) \\ b_n^{(d,d',c)}(4) \\ b_n^{(d,d',c)}(5) \\ b_n^{(d,d',c)}(6) \end{bmatrix}}_{\triangleq\, b_n^{(d,d',c)}} = \underbrace{\begin{bmatrix} w_1 & 0 & w_2 & 0 \\ 0 & w_2 & 0 & w_2 \\ w_2 & 0 & 0 & 0 \\ 0 & w_1 & 0 & 0 \\ 0 & 0 & w_1 & 0 \\ 0 & 0 & 0 & w_1 \end{bmatrix}}_{6\times 4} \underbrace{\begin{bmatrix} \delta_n^{(d',c+1)}(1) \\ \delta_n^{(d',c+1)}(2) \\ \delta_n^{(d',c+1)}(3) \\ \delta_n^{(d',c+1)}(4) \end{bmatrix}}_{=\,\delta_n^{(d',c+1)}} \tag{47.1251}$$

Observe that each column of the matrix multiplying $\delta_n^{(d',c+1)}$ contains the coefficients for the weight vector $w_d^{(d',c+1)}$. The locations where these coefficients appear are dependent on the partitioning operation, which maps the entries of $t_n^{(d,c)}$ into the four subvectors. We explained earlier in (47.1149) how this mapping occurs, in terms of a transformation

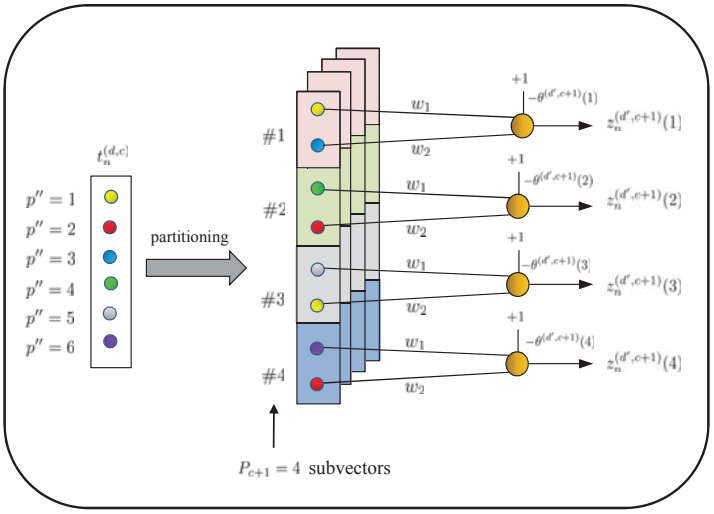

FIGURE 47.99 An example with $P_{c+1} = 4$ partitions and weight vector, $w_d^{(d',c+1)}$ for the $d-$th feature map, with two entries denoted simply by (w_1, w_2).

matrix for layer c, which we now denote by V_c of size $P_{c+1}S_c \times P_c'$; here S_c is the size of the subvectors $h_{n,p}^{(d,c)}$ for layer c. The matrix V_c for this example is given by

$$V_c = \left[\begin{array}{cccccc} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{array}\right], \quad (8 \times 6) \tag{47.1252}$$

That is,

$$\left[\begin{array}{c} 1 \\ 3 \\ \hline 4 \\ 2 \\ \hline 5 \\ 1 \\ \hline 6 \\ 2 \end{array}\right] = V_c \left[\begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array}\right] \tag{47.1253}$$

We also know from (47.1151) that

$$(I_4 \otimes \mathbb{1}_2)^{\mathsf{T}} V_c = \left[\begin{array}{cccccc} \mathbf{1} & 0 & \mathbf{1} & 0 & 0 & 0 \\ 0 & \mathbf{1} & 0 & \mathbf{1} & 0 & 0 \\ \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & 0 \\ 0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} \end{array}\right] \tag{47.1254}$$

with the unit entries on each row indicating which entries of $t_n^{(d,c)}$ belong to the subvector associated with that row after partitioning. For example, for the first row, the unit entries

appear in columns $\{1,3\}$; from the figure we see that these indices correspond to the entries in the first subvector, and so forth. An important fact to note here is that if we transpose the above matrix we obtain

$$V_c^{\mathsf{T}}(I_4 \otimes \mathbb{1}_2) \;=\; \begin{bmatrix} \mathbf{1} & 0 & \mathbf{1} & 0 \\ 0 & \mathbf{1} & 0 & \mathbf{1} \\ \mathbf{1} & 0 & 0 & 0 \\ 0 & \mathbf{1} & 0 & 0 \\ 0 & 0 & \mathbf{1} & 0 \\ 0 & 0 & 0 & \mathbf{1} \end{bmatrix} \tag{47.1255}$$

This matrix is the "skeleton" of the matrix that appears multiplying $\delta_n^{(d',c+1)}$ in (47.1251); its unit entries appear in the same locations where the coefficients (w_1, w_2) appear. Motivated by this result, it is easy to verify that

$$\left(V_c^{\mathsf{T}} \odot \left(\mathbb{1}_6\mathbb{1}_4^{\mathsf{T}} \otimes \left(w_d^{(d',c+1)}\right)^{\mathsf{T}}\right)\right)(I_4 \otimes \mathbb{1}_2) \;=\; \begin{bmatrix} w_1 & 0 & w_2 & 0 \\ 0 & w_2 & 0 & w_2 \\ w_2 & 0 & 0 & 0 \\ 0 & w_1 & 0 & 0 \\ 0 & 0 & w_1 & 0 \\ 0 & 0 & 0 & w_1 \end{bmatrix} \tag{47.1256}$$

The result in (47.1256) is the same matrix that appears multiplying $\delta_n^{(d',c+1)}$ in (47.1251). The computation (47.1256) amounts to transforming the weight column vector, $w_d^{(d',c+1)}$, into a matrix quantity whose dimensions match the mapping from $t_n^{(d,c)}$ (of size 6×1 in this example) to $z_n^{(d',c)}$ (of size 4×1). For ease of reference, we denote the transformation by the following matrix (where we are now restoring the general dimensions):

$$\boxed{V^{(d,d',c+1)} \;\triangleq\; \left(V_c^{\mathsf{T}} \odot \left(\mathbb{1}_{P_c'}\mathbb{1}_{P_{c+1}}^{\mathsf{T}} \otimes \left(w_d^{(d',c+1)}\right)^{\mathsf{T}}\right)\right)(I_{P_{c+1}} \otimes \mathbb{1}_{S_c}) \quad (P_c' \times P_{c+1})} \tag{47.1257}$$

We therefore arrived at the relation

$$b_n^{(d,d',c)} \;=\; V^{(d,d',c+1)}\delta_n^{(d',c+1)} \tag{47.1258}$$

where the entries of $b_n^{(d,d',c)}$ run over $p'' = 1, 2, \ldots, P_c'$. However, from (47.1249), we are interested in evaluating the quantities $\delta_n^{(d,c)}(j)$ for $j = 1, 2, \ldots, P_c$. We explained earlier that once we determine the value of $b_n^{(d,d',c)}(p'')$, for any p'', then this value would be the same for all indices j that contribute to p''. If we stack the values of $\delta_n^{(d,c)}(j)$ on top of

each other, say, for an example with $j = 1, 2, \ldots, 12$, and use (47.1249) we obtain

$$\underbrace{\begin{bmatrix} \delta_n^{(d,c)}(1) \\ \delta_n^{(d,c)}(2) \\ \delta_n^{(d,c)}(3) \\ \delta_n^{(d,c)}(4) \\ \delta_n^{(d,c)}(5) \\ \delta_n^{(d,c)}(6) \\ \delta_n^{(d,c)}(7) \\ \delta_n^{(d,c)}(8) \\ \delta_n^{(d,c)}(9) \\ \delta_n^{(d,c)}(10) \\ \delta_n^{(d,c)}(11) \\ \delta_n^{(d,c)}(12) \end{bmatrix}}_{\triangleq \delta_n^{(d,c)}} = \sum_{d'=1}^{D_{c+1}} \underbrace{\begin{bmatrix} f'\left(z_n^{(d,c)}(1)\right) \\ f'\left(z_n^{(d,c)}(2)\right) \\ f'\left(z_n^{(d,c)}(3)\right) \\ f'\left(z_n^{(d,c)}(4)\right) \\ f'\left(z_n^{(d,c)}(5)\right) \\ f'\left(z_n^{(d,c)}(6)\right) \\ f'\left(z_n^{(d,c)}(7)\right) \\ f'\left(z_n^{(d,c)}(8)\right) \\ f'\left(z_n^{(d,c)}(9)\right) \\ f'\left(z_n^{(d,c)}(10)\right) \\ f'\left(z_n^{(d,c)}(11)\right) \\ f'\left(z_n^{(d,c)}(12)\right) \end{bmatrix}}_{\triangleq f'\left(z_n^{(d,c)}\right)} \odot \underbrace{\begin{bmatrix} b_n^{(d,d',c)}(1) \\ b_n^{(d,d',c)}(1) \\ b_n^{(d,d',c)}(2) \\ b_n^{(d,d',c)}(1) \\ b_n^{(d,d',c)}(2) \\ b_n^{(d,d',c)}(3) \\ b_n^{(d,d',c)}(2) \\ b_n^{(d,d',c)}(1) \\ b_n^{(d,d',c)}(2) \\ b_n^{(d,d',c)}(3) \\ b_n^{(d,d',c)}(2) \\ b_n^{(d,d',c)}(3) \end{bmatrix}}_{} \odot \underbrace{\begin{bmatrix} \frac{\partial t_n^{(d,c)}(1)}{\partial y_n^{(d,c)}(1)} \\ \frac{\partial t_n^{(d,c)}(1)}{\partial y_n^{(d,c)}(2)} \\ \frac{\partial t_n^{(d,c)}(2)}{\partial y_n^{(d,c)}(3)} \\ \frac{\partial t_n^{(d,c)}(1)}{\partial y_n^{(d,c)}(4)} \\ \frac{\partial t_n^{(d,c)}(2)}{\partial y_n^{(d,c)}(5)} \\ \frac{\partial t_n^{(d,c)}(3)}{\partial y_n^{(d,c)}(6)} \\ \frac{\partial t_n^{(d,c)}(2)}{\partial y_n^{(d,c)}(7)} \\ \frac{\partial t_n^{(d,c)}(1)}{\partial y_n^{(d,c)}(8)} \\ \frac{\partial t_n^{(d,c)}(2)}{\partial y_n^{(d,c)}(9)} \\ \frac{\partial t_n^{(d,c)}(3)}{\partial y_n^{(d,c)}(10)} \\ \frac{\partial t_n^{(d,c)}(2)}{\partial y_n^{(d,c)}(11)} \\ \frac{\partial t_n^{(d,c)}(3)}{\partial y_n^{(d,c)}(12)} \end{bmatrix}}_{\triangleq x} \tag{47.1259}$$

where we are denoting the Hadamard product of the last two vectors by the generic letter x. It can be verified that — see Prob. 47.166:

$$x = \text{permute}^{\#}\left(\text{upv}\left(b_n^{(d,d',c)}\right)\right) \tag{47.1260}$$

where the notation upv$(\cdot)$ is now upsampling the individual entries of its vector argument, denoted by $v = \text{col}\{\alpha_j\}$, according to

$$\boxed{\text{upv}(v) \;=\; \text{col}\left\{\alpha_j \, g_{n,j}^{(d,c)}\right\}} \tag{47.1261}$$

by using the gradient vectors of the entries of $t_n^{(d,c)}$. Therefore, using (47.1258), we arrive at the relation:

$$\boxed{\delta_n^{(d,c)} = \sum_{d'=1}^{D_{c+1}} f'\left(z_n^{(d,c)}\right) \odot \text{permute}^{\#}\left(\text{upv}\left(V^{(d,d',c+1)}\delta_n^{(d',c+1)}\right)\right), \; d = 1, 2, \ldots, D_c} \tag{47.1262}$$

where the operations of upsampling and permutation were defined earlier in (47.1261) and (47.1204), respectively.

Desired Partial Derivatives

We are now in a position to evaluate partial derivatives of the unregularized risk terms, $\|\gamma_n - y_n\|^2$, relative to the combination weights and bias vectors themselves. For the layers in the feedforward network, results (47.1013) and (47.1019) continue to hold, namely,

$$\frac{\partial \|\gamma_n - y_n\|^2}{\partial W_\ell} = \delta_{\ell+1,n} \, (y_{\ell,n})^{\mathsf{T}} \tag{47.1263}$$

$$\frac{\partial \|\gamma_n - y_n\|^2}{\partial \theta_\ell} = -\delta_{\ell+1,}n \tag{47.1264}$$

We now consider the correlation layers. Differentiating with respect to the $s-$th coefficient of weight vector $w_j^{(d,c)}$ for map d in layer c we get:

$$\begin{aligned}
\frac{\partial \|\gamma_n - y_n\|^2}{\partial w_j^{(d,c)}(s)} &= \sum_{d'=1}^{D_c} \sum_{p=1}^{P_c} \frac{\partial \|\gamma_n - y_n\|^2}{\partial z_{n,}^{(d',c)}(p)} \, \frac{\partial z_{n,}^{(d',c)}(p)}{\partial w_j^{(d,c)}(s)} \\
&\overset{(a)}{=} \sum_{p=1}^{P_c} \frac{\partial \|\gamma_n - y_n\|^2}{\partial z_n^{(d,c)}(p)} \, \frac{\partial z_n^{(d,c)}(p)}{\partial w_j^{(d,c)}(s)} \\
&= \sum_{p=1}^{P_c} \delta_n^{(d,c)}(p) \, \frac{\partial z_n^{(d,c)}(p)}{\partial w_j^{(d,c)}(s)}
\end{aligned} \tag{47.1265}$$

where step (a) is because only $z_n^{(d,c)}(p)$ depends on $w_j^{(d,c)}(s)$. In particular, we recall from (47.1216) that the vector $z_n^{(d,c)}$ is given by

$$z_n^{(d,c)} = \sum_{j=1}^{D_{c-1}} \left(\left[H_n^{(j,c-1)} \right]^{\mathsf{T}} w_j^{(d,c)} - \theta^{(d,c)}(j) \mathbb{1} \right) \tag{47.1266}$$

and, hence,

$$\frac{\partial z_n^{(d,c)}(p)}{\partial w_j^{(d,c)}(s)} = \left[\left(H_n^{(j,c-1)} \right)^{\mathsf{T}} \right]_{p,s} = \left[H_n^{(j,c-1)} \right]_{s,p} \tag{47.1267}$$

where the generic notation $[A]_{p,s}$ denotes the $(p,s)-$th entry of matrix A. We conclude that

$$\begin{aligned}
\frac{\partial \|\gamma_n - y_n\|^2}{\partial w_j^{(d,c)}(s)} &= \sum_{p=1}^{P_c} \delta_n^{(d,c)}(p) \, \left[H_n^{(j,c-1)} \right]_{s,p} \\
&= \left[H_n^{(j,c-1)} \right]_{s,:} \delta_n^{(d,c)}
\end{aligned} \tag{47.1268}$$

where the notation $[A]_{s,:}$ refers to the $s-$th row of A. Therefore, expression (47.1268) amounts to an inner product between the $s-$th row of $H_n^{(j,c-1)}$ and $\delta_n^{(d,c)}$. Using the notation (47.1223a) and (47.1225) we conclude that — see Prob. 47.167

$$\frac{\partial \|\gamma_n - y_n\|^2}{\partial W^{(d,c)}} = H_{n,c-1} \delta_n^{(d,c)} \tag{47.1269}$$

If we introduce the block diagonal matrix

$$\Delta_{n,c} \triangleq \begin{bmatrix} \delta_n^{(1,c)} & & & \\ & \delta_n^{(2,c)} & & \\ & & \ddots & \\ & & & \delta_n^{(Dc,c)} \end{bmatrix} \tag{47.1270}$$

then we can verify that

$$\boxed{\frac{\partial \|\gamma_n - y_n\|^2}{\partial W_c} = \left(\mathbb{1}_{D_c}^{\mathsf{T}} \otimes H_{n,c-1}\right) \Delta_{n,c}} \tag{47.1271}$$

Likewise,

$$\begin{aligned} \frac{\partial \|\gamma_n - y_n\|^2}{\partial \theta^{(d,c)}(j)} &= \sum_{d'=1}^{D_c} \sum_{p=1}^{P_c} \frac{\partial \|\gamma_n - y_n\|^2}{\partial z_n^{(d',c)}(p)} \frac{\partial z_n^{(d',c)}(p)}{\partial \theta^{(d,c)}(j)} \\ &= \sum_{p=1}^{P_c} \frac{\partial \|\gamma_n - y_n\|^2}{\partial z_n^{(d,c)}(p)} \frac{\partial z_n^{(d,c)}(p)}{\partial \theta^{(d,c)}(j)} \\ &= -\sum_{p=1}^{P_c} \delta_n^{(d,c)}(p) \\ &= -\mathbb{1}_{P_c}^{\mathsf{T}} \delta_n^{(d,c)} \end{aligned} \tag{47.1272}$$

so that the gradient in relation to the vector $\theta^{(d,c)}$ is given by

$$\frac{\partial \|\gamma_n - y_n\|^2}{\partial \theta^{(d,c)}} = -\mathbb{1}_{D_{c-1}} \mathbb{1}_{P_c}^{\mathsf{T}} \delta_n^{(d,c)} \tag{47.1273}$$

and, consequently,

$$\boxed{\frac{\partial \|\gamma_n - y_n\|^2}{\partial \Theta_c} = -\left(\mathbb{1}_{D_c}^{\mathsf{T}} \otimes \mathbb{1}_{D_{c-1}} \mathbb{1}_{P_c}^{\mathsf{T}}\right) \Delta_{n,c}} \tag{47.1274}$$

In summary, we arrive at the following listing for the training algorithm, which enables us to evaluate the gradients of the risk function, $J_{\text{emp}}(W, \theta)$, relative to all combination matrices and all bias vectors in a convolutional neural network consisting of C convolutional layers and L feedforward layers. It is assumed that each layer c in the convolutional network has depth D_c, receives subvectors of size $S_c \times 1$, has a total of P_c neural units, and uses D_{c-1} weight vectors $\{w_j^{(d,c)}\}$ for each map d. Each layer c also has a matrix V_c associated with it, which describes the partitioning operation as was explained earlier in (47.1152). Moreover, the "permute" operation that appears in the listing of the algorithm amounts to multiplying its vector argument by the permutation matrix that is associated with the pooling operation, as was explained earlier in (47.1202). Each pooling layer has its own permutation matrix. Finally, the upsampling operation, denoted by "upv", amounts to multiplying each entry of its vector argument by the corresponding gradient of the pooling operation.

Training algorithm for convolutional neural networks

start with training data $\{\gamma_n, h_n\}, n = 0, 1, \ldots, N-1.$

(forward pass) repeat for $n = 0, 1, \ldots, N-1$:

feed h_n into the network and compute $\{z_{\ell,n}, y_{\ell,n}, z_n^{(d,c)}, y_n^{(d,c)}, H_n^{(d,c)}\}$ using (47.1216).

$\delta_{L,n} = 2\,(y_{L,n} - \gamma_n) \odot f'\,(z_{L,n})\,.$

end

(backward pass) repeat for $\ell = L-1, \ldots, 3, 2$ (feedforward network) :

$\delta_{\ell,n} = f'\,(z_{\ell,n}) \odot \left(W_\ell^{\mathsf{T}} \delta_{\ell+1}\right)$

end

set $\delta_n^{(d,C)} = f'\left(z_n^{(d,C)}\right) \odot \text{permute}^{\#}\left(\text{upv}\left(W_1^{\mathsf{T}}\delta_{2,n}\right)\right)$

(backward pass) repeat for $c = C-1, \ldots, 2, 1$ (convolutional network) :

$$V^{(d,d',c+1)} \;\triangleq\; \left(V_c^{\mathsf{T}} \odot \left(\mathbb{1}_{P_c'}\mathbb{1}_{P_{c+1}}^{\mathsf{T}} \otimes \left(w_d^{(d',c+1)}\right)^{\mathsf{T}}\right)\right)\left(I_{P_{c+1}} \otimes \mathbb{1}_{S_c}\right)$$

$$\delta_n^{(d,c)} = \sum_{d'=1}^{D_{c+1}} f'\left(z_n^{(d,c)}\right) \odot \text{permute}^{\#}\left(\text{upv}\left(V^{(d,d',c+1)}\delta_n^{(d',c+1)}\right)\right)$$

$$\Delta_{n,c} = \text{diag}\left\{\delta_n^{(1,c)}, \delta_n^{(2,c)}, \ldots, \delta_n^{(D_c,c)}\right\}$$

end

(derivatives) compute for $\ell = 1, 2, \ldots, L-1,\ c = 1, 2, \ldots, C$:

$$\frac{\partial J_{\text{emp}}}{\partial W_\ell} = 2\rho_2 W_\ell \;+\; \sum_{n=0}^{N-1} \delta_{\ell+1,n}\,(y_{\ell,n})^{\mathsf{T}}$$

$$\frac{\partial J_{\text{emp}}}{\partial \theta_\ell} = -\sum_{n=0}^{N-1} \delta_{\ell+1,n}$$

$$\frac{\partial J_{\text{emp}}}{\partial W_c} = 2\rho_1 W_c \;+\; \sum_{n=0}^{N-1} \left(\mathbb{1}_{D_c}^{\mathsf{T}} \otimes H_{n,c-1}\right)\Delta_{n,c}$$

$$\frac{\partial J_{\text{emp}}}{\partial \Theta_c} = -\sum_{n=0}^{N-1} \left(\mathbb{1}_{D_c}^{\mathsf{T}} \otimes \mathbb{1}_{D_{c-1}}\mathbb{1}_{P_c}^{\mathsf{T}}\right)\Delta_{n,c}$$

end

(47.1275)

Stochastic-Gradient Algorithm

We can now employ the algorithm to train a convolutional neural network by employing a stochastic-gradient implementation with step-size $\mu > 0$. In this implementation, one data point (γ_n, h_n) is used per iteration. We attach a subscript n to the combination matrices and bias vectors, which will now be adjusted, and denote them by $\{W_{\ell,n}, W_{c,n}\}$ and $\{\theta_{\ell,n}, \Theta_{c,n}\}$ at iteration n. We can also train the network by employing a mini-batch implementation, where a collection of B samples $\{\gamma_m, h_m\}$ are used at each iteration n to perform the gradient updates — recall (47.1031). If desired, different step-size values, μ_{ab} can be used for different entries of the weight matrices; likewise, for the bias coefficients.

Stochastic-gradient algorithm for training convolution networks

for each training point (γ_n, h_n) :

feed h_n into network $\{W_{\ell,n-1}, W_{c,n-1}, \theta_{\ell,n-1}, \Theta_{c,n-1}\}$ and

compute $\{z_{\ell,n}, y_{\ell,n}, z_n^{(d,c)}, y_n^{(d,c)}, H_n^{(d,c)}\}$ for all layers using (47.1216).

compute $\delta_{L,n} = 2(y_{L,n} - \gamma_n) \odot f'(z_{L,n})$.

(backward pass) repeat for $\ell = L-1, \ldots, 3, 2$ (feedforward network) :

$$\delta_{\ell,n} = f'(z_{\ell,n}) \odot \left(W_{\ell,n-1}^{\mathsf{T}} \delta_{\ell+1,n}\right)$$

end

set $\delta_n^{(d,C)} = f'\left(z_n^{(d,C)}\right) \odot \text{permute}^{\#}\left(\text{upv}\left(W_{1,n-1}^{\mathsf{T}} \delta_{2,n}\right)\right)$

(backward pass) repeat for $c = C-1, \ldots, 2, 1$ (convolutional network) :

$$\begin{aligned}
V_{n-1}^{(d,d',c+1)} &\triangleq \left(V_c^{\mathsf{T}} \odot \left(\mathbb{1}_{P_c'} \mathbb{1}_{P_{c+1}}^{\mathsf{T}} \otimes \left(w_{d,n-1}^{(d',c+1)}\right)^{\mathsf{T}}\right)\right)(I_{P_{c+1}} \otimes \mathbb{1}_{S_c}) \\
\delta_n^{(d,c)} &= \sum_{d'=1}^{D_{c+1}} f'\left(z_n^{(d,c)}\right) \odot \text{permute}^{\#}\left(\text{upv}\left(V_{n-1}^{(d,d',c+1)} \delta_n^{(d',c+1)}\right)\right) \\
\Delta_{n,c} &= \text{diag}\left\{\delta_n^{(1,c)}, \delta_n^{(2,c)}, \ldots, \delta_n^{(D_c,c)}\right\}
\end{aligned} \tag{47.1276}$$

end

repeat for $\ell = 1, 2, \ldots, L-1$, $c = 1, 2, \ldots, C$:

$$\begin{aligned}
W_{\ell,n} &= (1 - 2\mu\rho_2) W_{\ell,n-1} \;-\; \mu\mu\delta_{\ell+1,n} (y_{\ell,n})^{\mathsf{T}} \\
\theta_{\ell,n} &= \theta_{\ell,n-1} - \mu\delta_{\ell+1,n} \\
W_{c,n} &= (1 - 2\mu\rho_1) W_{c,n-1} \;-\; \mu \left(\mathbb{1}_{D_c}^{\mathsf{T}} \otimes H_{n,c-1}\right) \Delta_{n,c} \\
\Theta_{c,n} &= \Theta_{c,n-1} + \mu \left(\mathbb{1}_{D_c}^{\mathsf{T}} \otimes \mathbb{1}_{D_{c-1}} \mathbb{1}_{P_c}^{\mathsf{T}}\right) \Delta_{n,c}
\end{aligned}$$

end