## Sequence Decimation Network

Many sequences of practical interest exhibit a hierarchy of structure. For example, speech samples exhibit a hierarchy of periodic and noisy structures which give rise to phonemes, words, sentences and extended speech of arbitrary scale. Similarly, the character sequence comprising an article or news feed is rich with the hierarchy of characters, words, sentences, and so on. Internet usage exhibits structures at the millisecond level due to protocol operation and typical network traffic flows, at larger time scales due to transient news events, usage shocks, and due to periodicities in the calendar.

Consider a sequence classification task in which the classifier input is a sequence of samples from some ongoing process and the output is a classification decision. It is reasonable to assume that we could break the input sequence into small chunks and analyze those chunks to identify and summarize the small scale structures such as phonemes or low level protocol steps. We could then analyze small sequences of these summaries to identify structures spanning a larger portion of the input sequence, and we could continue this approach to find structures of arbitrary scale.

As a concrete example consider the sequence of six samples illustrated Figure 1. We first analyze groups of three samples resulting in two summary samples. It is important to note that the analysis performed on each group of three samples is identical. As the network grows in size, this sharing of analyzers limits complexity growth in the model and allows the lower layer of the network to be trained more efficiently than it otherwise would. We then analyze groups of two summary samples resulting in a single summary sample. In principle, the final summary could be a classification decision.

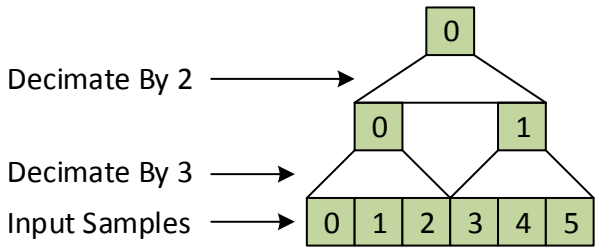


*Figure 1: This example decimates a sequence of six samples by three to yield two summary samples and then decimates the resulting sequence by two to yield a single summary sample.*

Thus far we have considered the input sequence to be one dimensional (having one feature per sample), and we have not described a particular transformation for summarizing the samples of one layer for use by the next. We now consider using a neural network to form the summaries and generalizing the approach to support multidimensional data. Figure 2 illustrates a case in which the input layer samples have three features each. The neural network first transforms groups of three samples having three features into summary samples having four features. It then transforms groups of two samples having four features into summary samples having two features. At each level of the hierarchy, there is a dual interpretation of the data. The dashed blue boxes enclose a summary sample from the layer below. The solid blue boxes aggregate multiple summary samples to form an input sample for the succeeding layer.

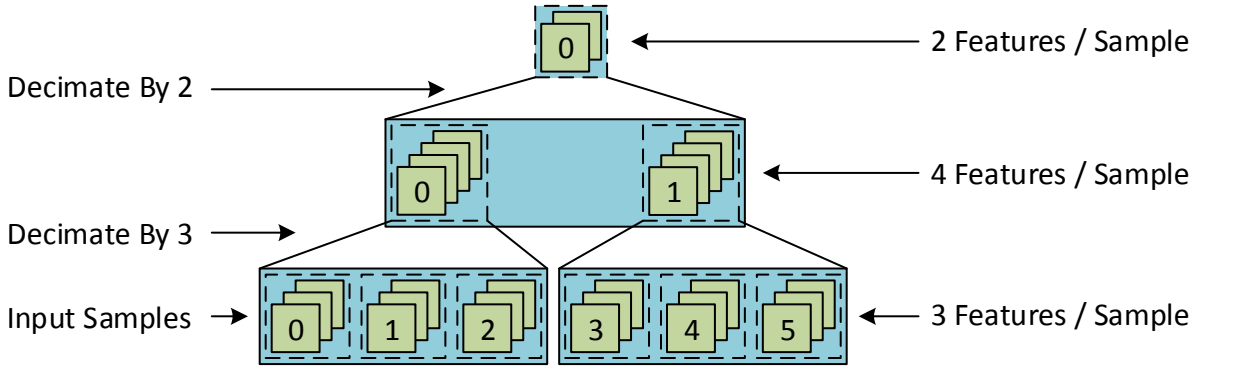


*Figure 2: This example decimates a sequence of six three dimensional samples by three to form two four dimensional summaries and then decimates a sequence of two four dimensional summaries to form a single two dimensional summary. The particular decimation ratios and feature dimensionality are arbitrary and chosen to illustrate the fact that the transformations need not be homogeneous in either feature space or decimation space.*

Consider the particular case of a neural network for which $f$ is an element-wise activation function, $W_n$ is the weight matrix connecting layer $n$ to layer $n+1$, and $B_n$ is the corresponding bias vector. Suppose that the content of a solid blue box from Figure 2 is flattened into a row vector $x$ as illustrated in Figure 3. The transformation of content from a solid blue box on layer $n$ to a dashed blue box on the layer $n+1$ is then described as follows:

$$x_{n+1} = f(x_n W_n + B_n)$$

In this case, the two 1x9 vectors at the input layer are each multiplied by a 9x4 weight matrix and added to a 1x4 bias matrix to which $f$ is applied to form summaries for the next layer. These summaries are interpreted as a 1x8 vector which is multiplied by an 8x2 weight matrix and added to a 1x2 weight matrix to which $f$ is applied to form the summary for the top layer. One nice property of this network is that there is no need to move the data to transform between the summary interpretation for a lower layer and the sample interpretation for an upper layer.

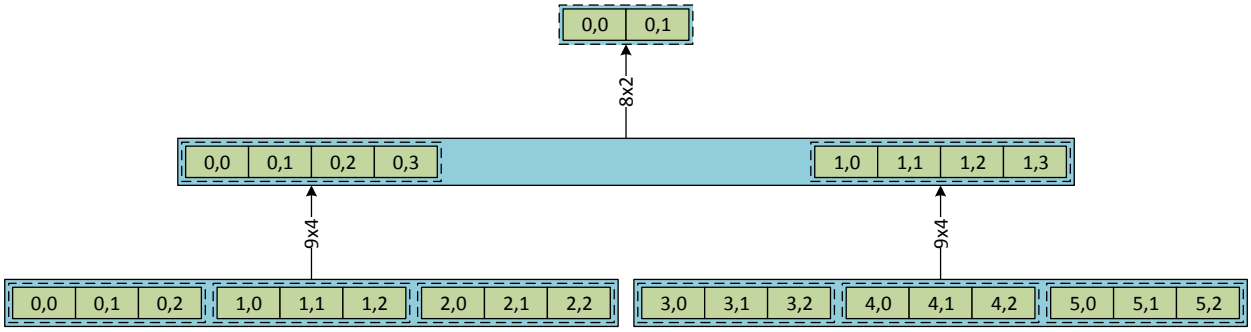


*Figure 3: When the data illustrated in figure 2 above is stored with C language ordering, it can easily be transformed from samples at one layer to summaries at the next. Decimation of the summaries to form new samples can be accomplished by reshaping the array (simply changing the interpretation) without moving any data.*

For optimal efficiency, we would like to be able to be able to process many input patterns at once. This can be accomplished using much the same strategy described above. Consider a case where we have 10 input patterns, each of which is six samples having three features $x[10][6][3]$. We reshape the array as $x_{0d}[20][9]$, where the $d$ subscript denotes the decimated interpretation of the data, and use $W_0[9][4]$ and $B_0[4]$ to compute $x_1[20][4] = f(x_{0d}W_0 + B_0)$. We then reshape as $x_{1d}[10][8]$ and use $W_1[8][2]$

and $B_1[2]$ to compute $x_2[10][2] = f(x_{1d}W_1 + B_1)$. This approach is quite general and can be used to efficiently process any number of input patterns through any number of layers with arbitrary decimation factors and feature dimensions at each layer.

*(Explain that the output error measure must correspond to the network activation function for back-propagation to work as described below)*

We now turn our attention to computing the gradient of error in this network with respect to its weights and biases using back-propagation. During the forward propagation phase described above we can also compute and save the derivatives of the activation function:

$$d_1[20][4] = f'(x_0W_0 + B_0)$$

$$d_2[10][2] = f'(x_1W_1 + B_1)$$

Knowing the desired output $t_2[10][2]$, we can compute the output layer error:

$$e_2[10][2] = y_2[10][2] - t_2[10][2]$$

We now compute the gradients of error with respect to weight and bias as:

$$g_1[8][2] = \frac{x_{1d}^T[10][8]e_2[10][2]}{10}$$

$$h_1[1][2] = \frac{\sum_{i=0}^{9} e_2[i][2]}{10}$$

We back-propagate the error to the preceding layer as:

$$e_{1d}[10][8] = d_1[10][8] \odot (e_2[10][2]W_1^T[8][2])$$

We now compute the gradients of error with respect to weight and bias as:

$$g_0[9][4] = \frac{x_{0d}^T[20][9]e_1[20][4]}{20}$$

$$h_0[1][4] = \frac{\sum_{i=0}^{19} e_1[i][4]}{20}$$

Having thus determined the gradients, we can update the network parameters using stochastic gradient descent with momentum $\alpha$ and learning rate $\lambda$ as:

$$\Delta_{W_n} \rightarrow \alpha\Delta_{W_n} + \lambda g_n$$

$$W_n \rightarrow W_n - \Delta_{W_n}$$

$$\Delta_{B_n} \rightarrow \alpha\Delta_{B_n} + \lambda h_n$$

$$B_n \rightarrow B_n - \Delta_{B_n}$$