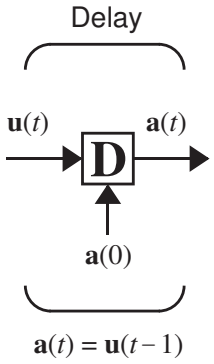
Theory and Examples

Dynamic networks were discussed in Chapters 10 and 14 of NND2. They are networks that are designed to operate on sequences of inputs and have the capability of remembering previous inputs and outputs.

Delays and Tapped Delay Lines

A fundamental component of a dynamic network is the delay, which we first introduced in Chapter 2 of NND2. The delay output is computed from its input according to

$$\mathbf{a}(t) = \mathbf{u}(t-1) \tag{6.1}$$



A delay enables a network to have memory. Note that in the diagram in the right margin the initial condition for the delay, $\mathbf{a}(0)$, is indicated by the vertical arrow going into the delay block.

As described in Chapter 10 of NND2, it is often useful to stack delays together to obtain a *tapped delay line* (TDL). This produces a history of previous values of the input, as shown in Figure 6.1

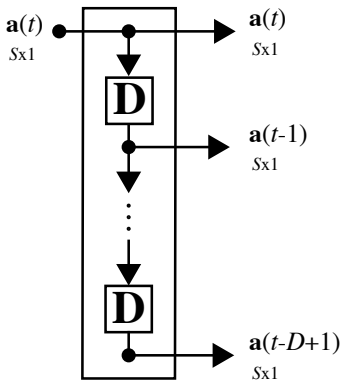


Figure 6.1: Tapped Delay Line

When using the output of the TDL, it can be convenient to stack, or concatenate, the outputs of the individual delays. Figure 6.2 shows a horizontal stacking (hstack), in which the vectors are stacked as columns. There is one $S \times 1$ vector coming in to the TDL_hstack, and a $S \times D$ matrix going out, where $D-1$ is the number of delays in the TDL. In other words,

$$\mathbf{A}(t) = \begin{bmatrix} \mathbf{a}(t) & \vdots & \mathbf{a}(t-1) & \vdots & \cdots & \vdots & \mathbf{a}(t-D+1) \end{bmatrix} \tag{6.2}$$

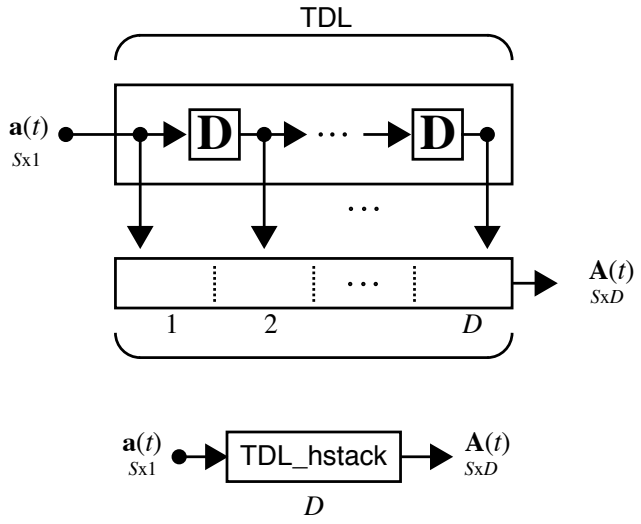


Figure 6.2: Horizontal Stacked Tapped Delay Line

Recurrent Networks

A recurrent network is a network that has feedback connections. The most basic recurrent network is shown in Figure 6.3. Equations 6.3 and 6.4 define the network operation.

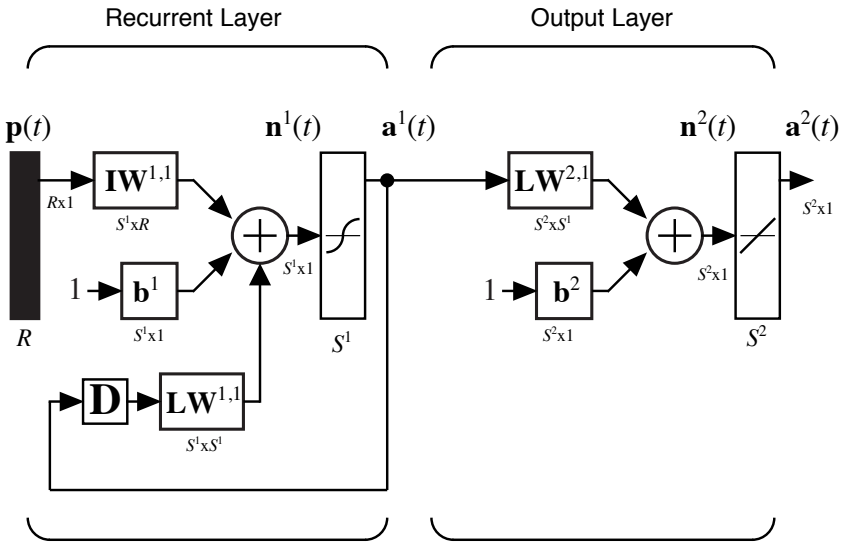


Figure 6.3: Basic Recurrent Network

$$\mathbf{a}^1(t) = \mathbf{tansig}\left(\mathbf{IW}^{1,1}\mathbf{p}(t) + \mathbf{LW}^{1,1}\mathbf{a}^1(t-1) + \mathbf{b}^1\right) \tag{6.3}$$

$$\mathbf{a}^2(t) = \mathbf{LW}^{2,1}\mathbf{a}^1(t) + \mathbf{b}^2 \tag{6.4}$$

Sequence to Sequence Networks

An interesting network that uses two RNNs as components is the *sequence-to-sequence network* (seq2seq). which is shown in Figure 6.4. The first RNN is called the *Encoder*, and the second RNN is called the *decoder*. This network has been used for such tasks as machine translation. For example, the $\mathbf{u}(t)$ input sequence might represent English words and the output sequence $\mathbf{y}(t)$ might represent French words.

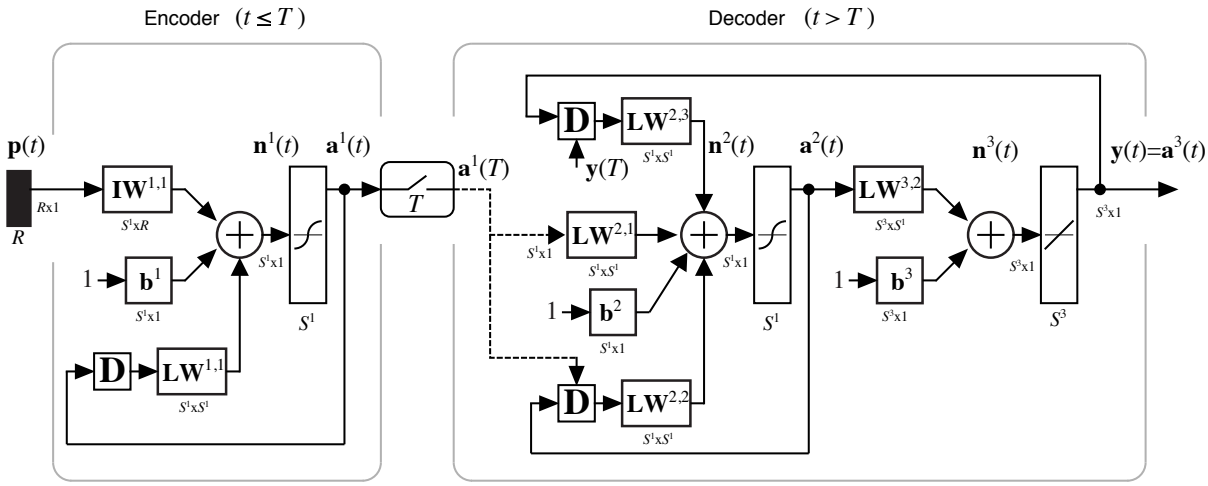
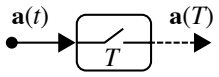


Figure 6.4: Sequence to Sequence Network

The encoder takes the input sequence, which has T time points (words, or word fragments), and produces a layer output $\mathbf{a}^1(t)$. This output is sampled at the final time point $t = T$. This is represented by the sampler block, shown in the right margin. The dotted line at the output of the sampler indicates that this value is fixed, and does not change with time. The operation of the encoder is defined by Eq. 6.5.



$$\mathbf{a}^1(t) = \mathbf{tansig}\left(\mathbf{IW}^{1,1}\mathbf{p}(t) + \mathbf{LW}^{1,1}\mathbf{a}^1(t-1) + \mathbf{b}^1\right) \qquad (6.5)$$

The decoder initializes its state with the final state of the encoder. The final state of the encoder also becomes a constant input to the first layer of the decoder. (There are many different formulations of the seq2seq model. In some cases the final encoder state is only used to initialize the state of the decoder.) This final encoder state is sometimes referred to as the *context*. The context is a summary of the input sequence. The operation of the decoder is defined by

Equations 6.6 and 6.7.

$$\mathbf{a}^2(t) = \mathbf{tansig}\left(\mathbf{LW}^{2,3}\mathbf{a}^3(t-1) + \mathbf{LW}^{2,1}\mathbf{a}^1(T) + \mathbf{LW}^{2,2}\mathbf{a}^2(t-1) + \mathbf{b}^1\right) \quad (6.6)$$

$$\mathbf{a}^3(t) = \mathbf{LW}^{3,2}\mathbf{a}^1 + \mathbf{b}^3 \quad (6.7)$$

Attention

The recurrent layer of the encoder is able to store information from the beginning of the input sequence, but it may not be able to store it in the most efficient way. To provide more flexibility, we can add a tapped delay line of previous values of the encoder state, rather than using just the final value. This process is called *attention*.

The first step is to use a TDL_hstack to form a horizontal concatenation of previous encoder states, as in

$$\mathbf{A}^1(t) = \begin{bmatrix}\mathbf{a}^1(t) & \mathbf{a}^1(t-1) & \cdots & \mathbf{a}^1(t-D+1)\end{bmatrix} \quad (6.8)$$

We want to combine these vectors together in some way to form the context vector to send to the decoder (in place of just the final encoder state). The question is how much should each previous encoder state contribute to the context. Also, we may want the context to change at each time step of the decoder operation. For example, when translating a given English sentence to French, as we get to a given French word in the translation, it may depend on different combinations of English words in the original sentence. One approach could be to find out how correlated the current decoder state is to each previous encoder state. This could be done by finding the inner product between the previous decoder state and all previous encoder states, as in the following equation.

$$\mathbf{n}^4(t) = \left[\mathbf{A}^1(T)\right]^T \mathbf{a}^2(t-1) \quad (6.9)$$

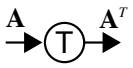
We should then normalize these correlations by using the `softmax` activation function, as in

$$\mathbf{a}^4(t) = \mathbf{softmax}\left(\mathbf{n}^4(t)\right) \quad (6.10)$$

Finally, we combine the previous encoder state vectors according to the relative amount of correlation with the current decoder state.

$$\mathbf{a}^5(t) = \mathbf{A}^1(T)\mathbf{a}^4(t) \quad (6.11)$$

The resulting $\mathbf{a}^5(t)$ is the context vector that will be passed to the decoder. A diagram of this dot product attention is shown in Figure 6.5, where the T inside the circle is the transpose operator (see margin).



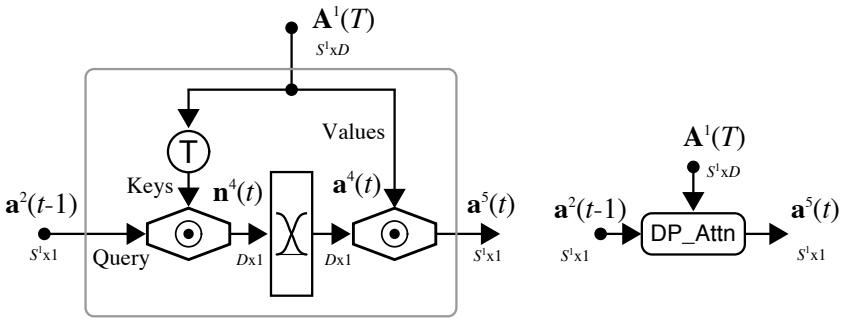


Figure 6.5: Dot Product Attention

The idea of attention is sometimes generalized as operations between a query vector and sets of key and value pairs. The query is compared to each of the keys to determine how much of each corresponding value is included in the result. In our case, the query is the previous decoder state $\mathbf{a}^2(t-1)$, the keys are all previous encoder states, and the values are also the previous encoder states.

Figure 6.6 shows the seq2seq network with dot product attention added. The difference between the standard seq2seq model and the one with attention is that $\mathbf{a}^5(t)$ is the context vector that is input to Layer 2, instead of $\mathbf{a}^1(T)$. The $\mathbf{a}^5(t)$ context will change with each time step, whereas $\mathbf{a}^1(T)$ is fixed.The $\mathbf{a}^4(t)$ vector (whose elements are sometimes referred to as the attention weights) will determine how much each $\mathbf{a}^1(t)$ will contribute to the context at each step of the decoder. If the first element of $\mathbf{a}^4(t)$ is equal 1 (and the other elements are consequently equal to 0), then the context will again be $\mathbf{a}^1(T)$, as in the standard seq2seq model. Otherwise, the context will include additional time points of the encoder state. This means that the decoder will be paying *attention* to different encoder states at different steps of the decoder operation.

In addition to dot product attention, there are other forms available. The dot product between the query vector $\mathbf{a}^2(t-1)$ (for $t > T$) and the key vectors $\mathbf{a}^1(t)$ (for $t \leq T$) is a measure of the similarity between the query and each of the keys. This use of inner product for measuring similarity is discussed in Chapter 5 of NND2. As that chapter indicated, there are many forms of inner product.

If we have two vectors, $\mathbf{x}$ and $\mathbf{y}$, then the standard inner prod-

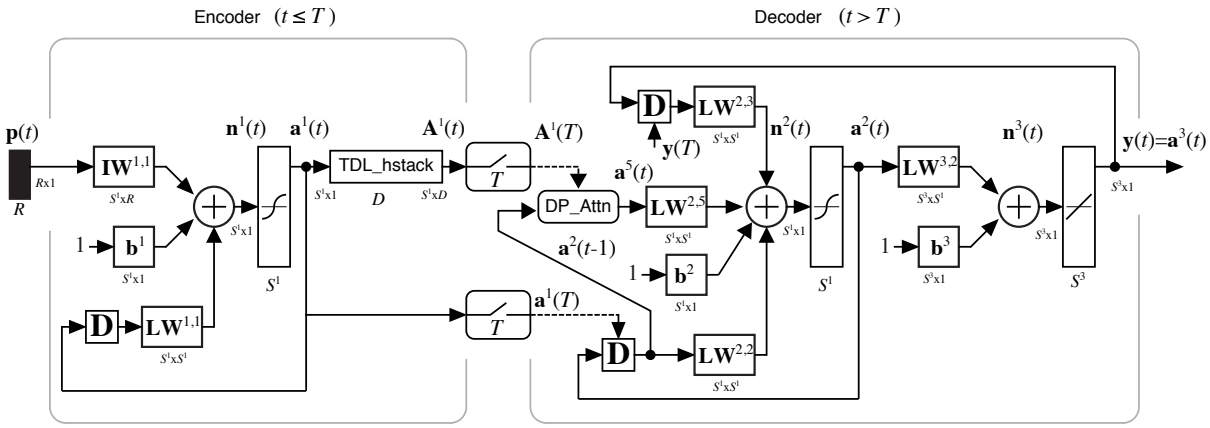


Figure 6.6: Sequence to Sequence Network With Attention

uct is the dot product $\mathbf{x}^T\mathbf{y}$. Another common inner product is the weighted dot product $\mathbf{x}^T\mathbf{W}\mathbf{y}$, where the positive definite weighting matrix $\mathbf{W}$ is used to emphasize certain components of the dot product. This concept can be easily incorporated into the attention mechanism. We just need to replace Eq. 6.9 with

$$\mathbf{n}^4(t) = \left[\mathbf{A}^1(T)\right]^T \mathbf{W}_a \mathbf{a}^2(t-1) \tag{6.12}$$

where $\mathbf{W}_a$ is the attention weighting matrix. This type of attention has been referred to as general attention, although it might be more precise to call it general dot product attention or weighted dot product attention. Figure 6.7 is a representation of this general dot product attention.

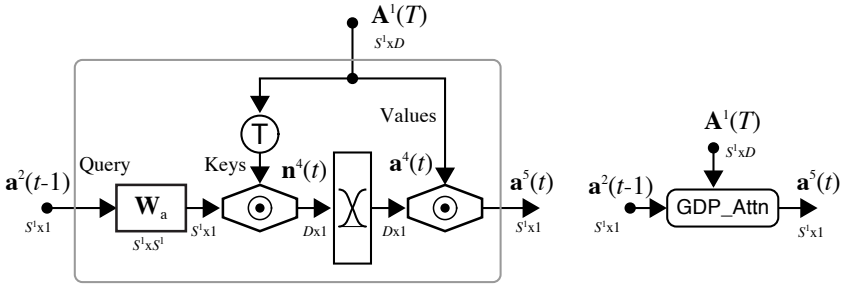


Figure 6.7: General Dot Product Attention

The idea of attention is to determine how much the encoder state at each time point (for $t \le T$) should contribute to the context vector, given the query vector $\mathbf{a}^2(t-1)$. Instead of performing an inner product between the query and the keys (encoder states), we could just use a general functional relationship between them. This could

be done by stacking (concatenating) the query with each of the keys and passing the result through a multilayer subnetwork, which could be trained as part of the overall seq2seq network.

The first step is to vertically stack, or concatenate the query $\mathbf{a}^2(t-1)$ with the previous encoder states $\mathbf{A}^1(T)$.

The vertical stacking operation uses the NumPy broadcasting rules. If one of the arrays has a single column, that column is repeated until the number of columns matches the larger array.

$$\bar{\mathbf{A}}^{1,2}(t) = \begin{bmatrix} \mathbf{A}^1(T) \\ \begin{bmatrix} \mathbf{a}^2(t-1) & \mathbf{a}^2(t-1) & \cdots & \mathbf{a}^2(t-1) \end{bmatrix} \end{bmatrix} \tag{6.13}$$

$$= \begin{bmatrix} \begin{bmatrix} \mathbf{a}^1(T) \\ \mathbf{a}^2(t-1) \end{bmatrix} & \begin{bmatrix} \mathbf{a}^1(T-1) \\ \mathbf{a}^2(t-1) \end{bmatrix} & \cdots & \begin{bmatrix} \mathbf{a}^1(T-D+1) \\ \mathbf{a}^2(t-1) \end{bmatrix} \end{bmatrix} \tag{6.14}$$

This matrix is then passed into a two layer network.

$$\mathbf{A}^4(t) = \mathbf{tansig}\left(\mathbf{LW}^{4,2}\bar{\mathbf{A}}^{1,2}(t) + \mathbf{b}^4\right) \tag{6.15}$$

$$\mathbf{N}^5(t) = \mathbf{LW}^{5,4}\mathbf{A}^4(t) + \mathbf{b}^5 \tag{6.16}$$

$$\mathbf{a}^5(t) = \mathbf{softmax}\left(\left[\mathbf{N}^5(t)\right]^T\right) \tag{6.17}$$

Finally, to get the attention weights, we multiply by the values $\mathbf{A}^1(T)$.

$$\mathbf{a}^6(t) = \mathbf{A}^1(T)\mathbf{a}^6(t) \tag{6.18}$$

Figure 6.8 shows the implementation of concatenation attention.

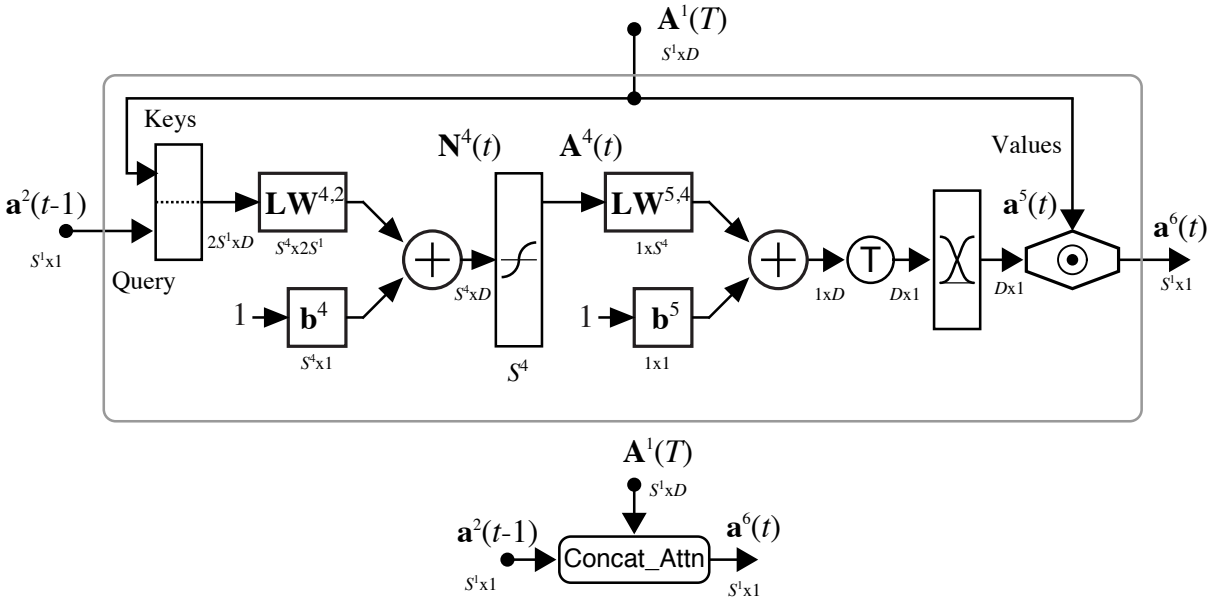


Figure 6.8: Concatenation Attention