7 General Modular Network Notation

Objective

As we describe more and more complex types of neural networks, it is very important to have clear, consistent and efficient notation and diagrams. The notation is the mathematical language that is used to precisely express the network operation and training. Good diagrams enable us to use visual reasoning, which is complementary to verbal and mathematical reasoning. Deep networks can be very complex, and we need a range of tools to develop a deep understanding. In this chapter we will introduce some general notation and diagrams that we can used to describe the deep networks that will be presented in later chapters. This notation is an extension of the notation from NND2.

Theory and Examples

A variety of types of diagrams and notation have been used to represent neural networks in the years since their first mathematical formulations in the 1940's by McCulloch and Pitts. One difficulty in finding a unified notation for neural networks is that their development has been so interdisciplinary. Contributions have come from many different research and application areas, each with their own notational tendencies. In addition, developments in the neural network field have come in fits and starts over the years, usually with new starts coming from researchers in new disciplines, often with new notation to propose. The result is that notation and diagrams will often change from one paper to another, making it difficult to see the affinity among seemingly different network types.

For this reason, it is important to find a consistent notation to use for all the networks we present in this text. A useful notation should encompass as much of the neural network field as possible, without becoming overly complex and unwieldy. (Of course, beauty is in the eye of the beholder.) In addition, if possible, a diagram of a neural network should provide all the information we need to implement the network.

In the development of the first edition of NND2 in 1993, we made a significant effort to find a notation and associated diagrams that would be suitable for the neural network field at that time, and that would be able to expand in the future. Since we were developing an extensive software framework at that time (and extending it through the next 25 years), we sought to make the diagrams/notation consistent with the code. This encouraged the practicality of the notation. Over the years the software has adapted to accommodate new network components and architectures. Some of the notation supporting the new architectures was presented in NND2.

The following sections will describe an update of the notation from NND2. This updated notation maintains the modularity of previous versions and is based on the concept of a layer.

Definition of a Layer

The general notation that we use in this text revolves around layers and the connections between layers. Chapters 2 and 14 of NND2 in-

troduced these ideas, but here we make extensions to allow more general types of weight and net input operations. The first step is to define what we mean by a layer. Over the years, the meaning of the term "layer" in the neural network community has changed multiple times. In order to develop a structure for which we can efficiently perform gradient calculations, we need a careful and consistent definition of *layer*.

For our purposes in this text, a layer consists of the following parts:

- A set of weight matrices, and associated weight functions, that come into that layer (which may connect from other layers or from external inputs),
- Any tapped delay lines that appear at the input of a weight matrix
- A bias vector,
- A net input function (e.g., a summing junction), and
- An activation function.

Multilayer Perceptron

The simplest example of a network of layers is the multilayer network, which we discussed in Chapter 2 and is shown in Figure 7.1 for a three layer network. In this case the weight functions are matrix multiplications between the weight matrices and the inputs, and the net input functions are summations.

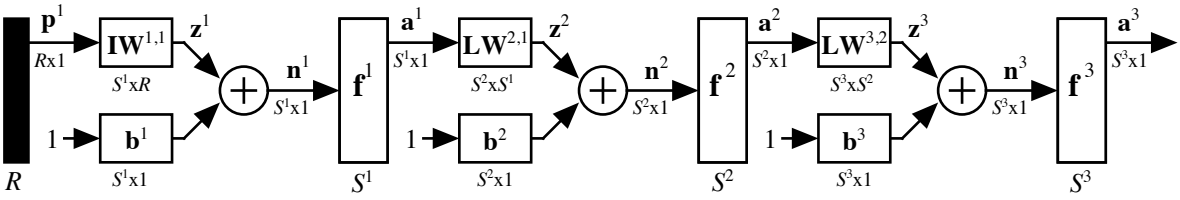


Figure 7.1: Three Layer Network

The equations of operation for the multilayer network are

$$\mathbf{a}^0 = \mathbf{p} \tag{7.1}$$

$$\mathbf{n}^{m+1} = \mathbf{W}^{m+1}\mathbf{a}^m + \mathbf{b}^{m+1}, m = 0, 1, ..., M-1 \tag{7.2}$$

$$\mathbf{a}^m = \mathbf{f}^m(\mathbf{n}^m) \tag{7.3}$$

Layered Feedforward Network

We can generalize the multilayer network by allowing each layer to connect forward to an arbitrary number of other layers. In addition, we can have multiple input vectors, each one of which can be connected to any layer. We call the resulting class of networks *Layered Feedforward Neural Networks* (LFNN). An example is shown in Figure 7.2.

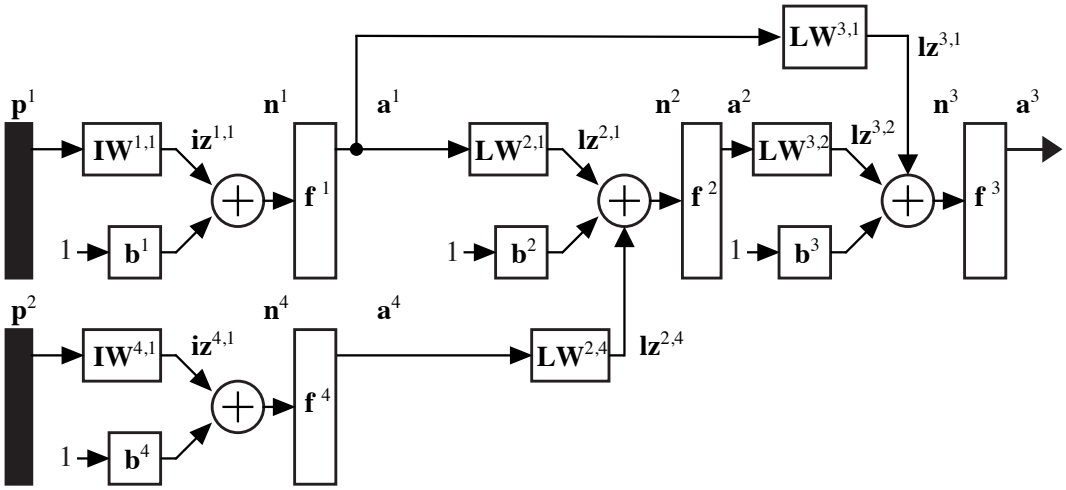


Figure 7.2: Layered Feedforward Network Example

The equations of operation of an LFNN can be written as

$$\mathbf{n}^m = \sum_{i \in I_m} \mathbf{IW}^{m,l}\mathbf{p}^l + \sum_{i \in L_m^f} \mathbf{LW}^{m,l}\mathbf{a}^l + \mathbf{b}^m \tag{7.4}$$

$$\mathbf{a}^m = \mathbf{f}^m(\mathbf{n}^m) \tag{7.5}$$

where $\mathbf{p}^l$ is the l^{th} input vector to the network, $\mathbf{IW}^{m,l}$ is the *input weight* between input l and layer m, $\mathbf{LW}^{m,l}$ is the *layer weight* between layer l and layer m, $\mathbf{b}^m$ is the bias vector for layer m, I_m is the set of indices of input vectors that connect to layer m, and L_m^f is the set of indices of layers that directly connect forward to layer m.

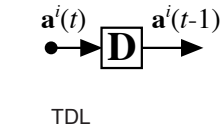
To compute the output of a multilayer network, we start at Layer 1, and proceed in order to Layer M. When we allow general feedforward connections, we need to know what order to compute the layer outputs in Eq. 7.4, so that all of the terms on the right side of the equation are available. We call this order the *Simulation Order*. The simulation order may not be unique; for the network in Figure 7.2, the simulation order could be 1-4-2-3 or 4-1-2-3.

It should be noted that LFNNs, since they are purely feedforward networks, cannot have any feedback loops. If feedback loops existed in a static network, it would not be possible to compute the network output, since a layer output would be needed before it was computed. Feedback can only occur in a dynamic network – a network that contains delays.

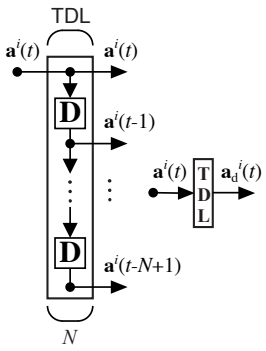
Dynamic Networks

The networks that we have discussed to this point in this chapter are static networks. The current outputs of the network can be computed from the current inputs. Such networks have no memory. However, there are many problems, like machine translation and control of dynamic systems, for which it is important for a network to have access to previous inputs, outputs or states. We call such networks *dynamic networks*. (We will devote a later chapter to dynamic networks, but we will describe the notation here.)

Dynamic networks use delays (for discrete-time, or *digital networks*) or integrators (for continuous-time networks) to actualize memory. In this book we will consider only digital networks. A single delay is shown in the margin. The output of the delay is the input to the delay from the previous time step. When operating a dynamic network, all delays need to be initialized first.



In the literature on dynamic networks, which are often discussed in the context of recurrent neural networks (RNNs), you will most often see single delays in the network diagrams. However, it is useful on many occasions to stack these delays into a *tapped delay line* (TDL). In this way, a network can have direct access to a history of a given variable. A diagram of a TDL is shown in the margin.



If we generalize the LFNN to include TDLs, which then allows the network to have feedback (recurrent) connections, we have a class of network that we refer to as *Layered Digital Dynamic Networks* (LDDN). (These networks were covered extensively in Chapter 14 of

NND2.) The equations of operation for these networks are given by

$$\mathbf{n}^m(t) = \sum_{l \in L_m^f} \sum_{d \in DL_{m,l}} \mathbf{LW}^{m,l}(d)\mathbf{a}^l(t-d) \tag{7.6}$$

$$+ \sum_{l \in I_m} \sum_{d \in DI_{m,l}} \mathbf{IW}^{m,l}(d)\mathbf{p}^l(t-d) + \mathbf{b}^m \tag{7.7}$$

$$\mathbf{a}^m(t) = \mathbf{f}^m(\mathbf{n}^m(t)) \tag{7.8}$$

where most of these expressions have the same meaning as in the LFNN equations, with the addition of $DI_{m,l}$ and $DL_{m,l}$, which represent the sets of the delays to Layer m from Input or Layer l, respectively. Usually, these delays consist of a sequential sequence of numbers from 0 or 1 up to the maximum number of delays. However, the equations allow for any combination of delays.

Unlike the LFNN, an LDDN can have arbitrary connections between layers, but every feedback loop must contain at least one delay. Forward computations are performed forward in time and forward in the simulation order for layers. An example LDDN is shown in Figure 7.3.

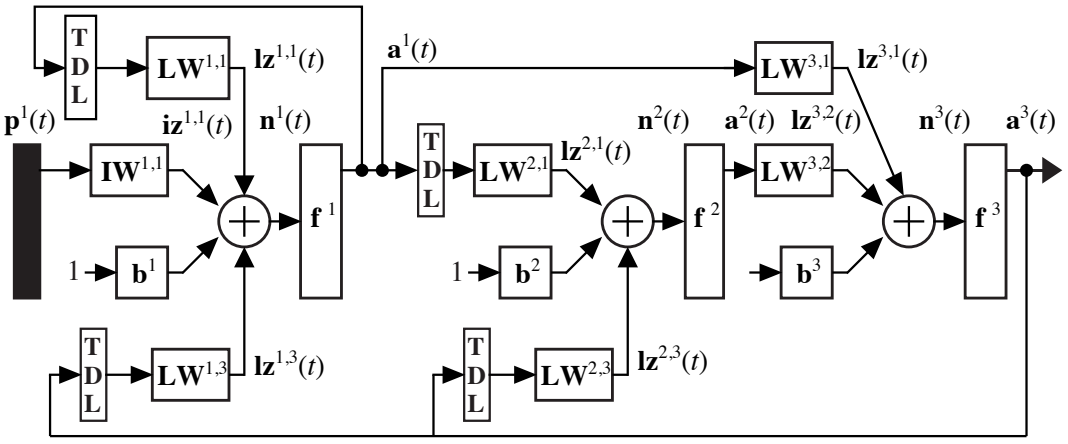


Figure 7.3: Layered Digital Dynamic Network Example

The LFNN and the simpler multilayer network are subsets of the more general LDDN, since we can create these networks within the LDDN framework by setting the delays equal to zero. We want to make one more step of generalization by allowing the weight functions and net input functions to be arbitrary operations (as long as they are differentiable), instead of being restricted to be matrix multiplication and addition, respectively. We will refer to this most general formulation as the *Generalized LDDN* (GLDDN).

For the GLDDN, the output of the weight functions (indicated by the letter **z**) at each layer are computed as follows:

$$\mathbf{iz}^{m,l}(t,d) = \mathbf{ih}^{m,l}(\mathbf{IW}^{m,l}(d), \mathbf{p}^l(t-d)) \tag{7.9}$$

$$\mathbf{lz}^{m,l}(t,d) = \mathbf{lh}^{m,l}(\mathbf{LW}^{m,l}(d), \mathbf{a}^l(t-d)) \tag{7.10}$$

where $\mathbf{ih}^{m,l}()$ and $\mathbf{lh}^{m,l}()$ are input and layer weight functions. They are functions of the corresponding weight matrices $\mathbf{IW}^{m,l}(d)$ and $\mathbf{LW}^{m,l}(d)$, as well as the input $\mathbf{p}^l(t-d))$ or layer output $\mathbf{a}^l(t-d)$. For example, if the weight functions were standard matrix multiplication, then we would have

$$\mathbf{ih}^{m,l}(\mathbf{IW}^{m,l}(d), \mathbf{p}^l(t-d)) = \mathbf{IW}^{m,l}(d)\mathbf{p}^l(t-d) \tag{7.11}$$

The net input for the GLDDN is computed as follows:

$$\mathbf{n}^m(t) = \mathbf{o}^m(\mathbf{iz}^{m,l}(t,d)|_{d\in DI_{m,l}}^{l\in I_m}, \mathbf{lz}^{m,l}(t,d)|_{d\in DL_{m,l}}^{l\in L_m^f}, \mathbf{b}^m) \tag{7.12}$$

where $\mathbf{o}^m()$ is the net input function at Layer m. For the LDDN the net input is simply computed by summing all of the weight function outputs plus the bias. For the GLDDN, the net input can combine the weight function outputs and the bias in an arbitrary way.

The layer output for the GLDDN is computed in the same way as all of the other formulations:

$$\mathbf{a}^m(t) = \mathbf{f}^m(\mathbf{n}^m(t)) \tag{7.13}$$

The GLDDN layer outputs are computed forward in time and forward in the simulation order for layers, as in the LDDN. An example GLDDN is shown in Figure 7.4.

There are no restrictions on the dimensions of the weight matrices in the GLDDN relative to the dimensions of the input of the weight function and no restriction on the dimension of the bias. Their sizes just need to be consistent with the defined weight and net input functions.

Equations 7.9 through 7.13 for the GLDDN show vector variables. (Our standard notation for a vector is the lowercase bold letter.) However, with a substitution of notation we can interpret these equations as operating on multi-dimensional matrices (tensors). We will take advantage of this in the next chapter on convolution networks.

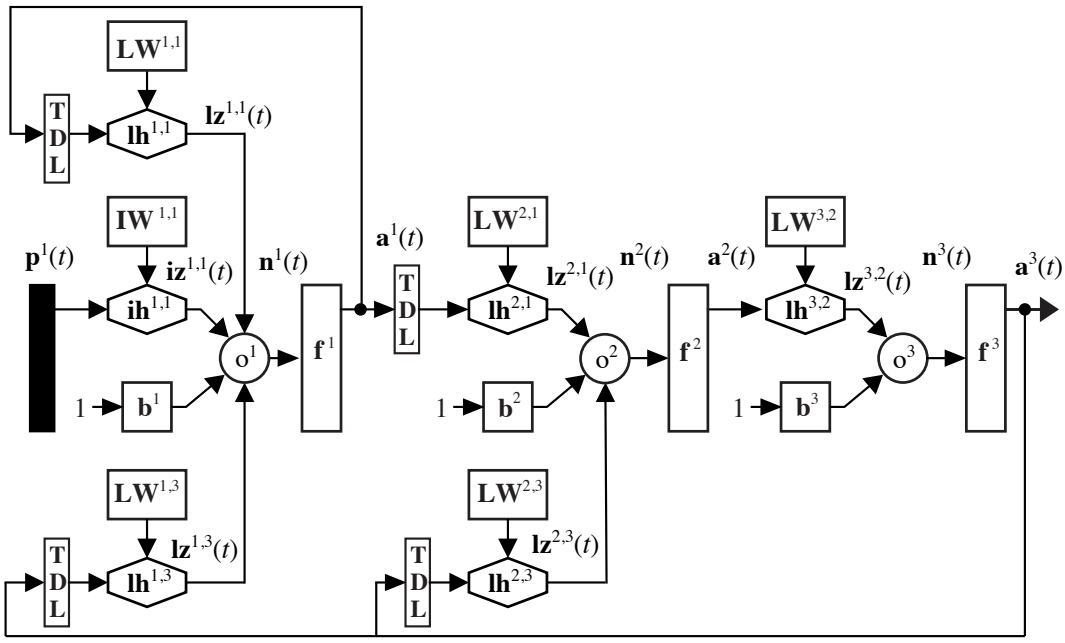


Figure 7.4: General Layered Digital Dynamic Network Example

Gradient Calculation for GLFNN

We are going to investigate GLDDNs in some detail in a later chapter on dynamic networks, including the gradient calculation, which is more complex than for static networks. In this section we want to describe how to compute the gradient for the static version of this network, which would be the Generalized LFNN, or GLFNN. This will be needed in the next chapter on convolution networks.

The gradient for the GLFNN is computed using the standard backpropagation process, except that there may be several backward paths. As we noted in Chapter 3, for multilayer networks the backpropagation process starts with the derivative of the performance (or loss) function $F(\mathbf{x})$ with respect to the net input in the final (output) layer of the network. (See Eq. 3.50.) It starts at the final layer, because the output of that layer is directly compared to the targets in the performance function. In the GLFNN, however, it is possible to have multiple output layers, just as the network can have multiple inputs. We will indicate the set of output layer indices by U. (For the GLFNN networks, a layer is an input layer if it has an input weight. A layer is an output layer if its output will used in the performance function calculation.)

The elements of the gradient are the derivatives of the perfor-

mance function with respect to the weights and biases of the network. These derivatives are computed using the chain rule, as in

$$\frac{\partial F(\mathbf{x})}{\partial iw_{i,j}^{m,l}} = \left(\frac{\partial F(\mathbf{x})}{\partial \mathbf{n}^m}\right)^T \frac{\partial \mathbf{n}^m}{\partial iw_{i,j}^{m,l}} = (\mathbf{s}^m)^T \frac{\partial \mathbf{n}^m}{\partial iw_{i,j}^{m,l}} \tag{7.14}$$

$$\mathbf{s}^m \triangleq \frac{\partial F(\mathbf{x})}{\partial \mathbf{n}^m} \tag{7.15}$$

where we call $\mathbf{s}^m$ the sensitivity, as in Chapter 11 of NND2.

To compute the final term in Eq. 7.14, we need to compute the derivative of the weight function in the following chain rule:

$$\frac{\partial \mathbf{n}^m}{\partial iw_{i,j}^{m,l}} = \frac{\partial \mathbf{n}^m}{\partial (\mathbf{iz}^{m,l})^T} \frac{\partial \mathbf{iz}^{m,l}}{\partial iw_{i,j}^{m,l}} \tag{7.16}$$

$$\frac{\partial \mathbf{n}^m}{\partial lw_{i,j}^{m,l}} = \frac{\partial \mathbf{n}^m}{\partial (\mathbf{lz}^{m,l})^T} \frac{\partial \mathbf{lz}^{m,l}}{\partial lw_{i,j}^{m,l}} \tag{7.17}$$

$$\frac{\partial \mathbf{n}^m}{\partial b_i^m} = \frac{\partial \mathbf{n}^m}{\partial b_i^m} \tag{7.18}$$

The previous equations simplify for the standard multilayer network.

$$\frac{\partial \mathbf{n}^m}{\partial (\mathbf{iz}^{m,l})^T} = \mathbf{I}, \qquad \frac{\partial \mathbf{n}^m}{\partial (\mathbf{lz}^{m,l})^T} = \mathbf{I} \tag{7.19}$$

$$\frac{\partial \mathbf{iz}^{m,l}}{\partial iw_{i,j}^{m,l}} = p_j^l \mathbf{e}_i, \qquad \frac{\partial \mathbf{lz}^{m,l}}{\partial lw_{i,j}^{m,l}} = a_j^l \mathbf{e}_i \tag{7.20}$$

$$\frac{\partial \mathbf{n}^m}{\partial b_i^m} = \mathbf{e}_i \tag{7.21}$$

$$\tag{7.22}$$

where $\mathbf{e}_i$ is a vector whose i^{th} element is 1, and the rest of the elements are zero.

The sensistivity $\mathbf{s}^m$ is computed by starting at all $\mathbf{s}^u$, $u \in U$. The performance function $F(\mathbf{x})$ will be an explicit function of these output layers.

$$\mathbf{s}^u = \frac{\partial F(\mathbf{x})}{\partial \mathbf{n}^u} = \left(\frac{\partial \mathbf{a}^u}{\partial (\mathbf{n}^u)^T}\right)^T \frac{\partial F(\mathbf{x})}{\partial \mathbf{a}^u} = \dot{\mathbf{F}}^u(\mathbf{n}^u)^T \frac{\partial F(\mathbf{x})}{\partial \mathbf{a}^u} \tag{7.23}$$

The remaining sensitivities $\mathbf{s}^m$ for $m \notin U$ are computed by following the backpropagation order (inverse of the simulation order), where L_m^b contains the indices of layers directly connected backward to layer m.

$$\mathbf{s}^m = \left(\frac{\partial \mathbf{a}^m}{\partial \left(\mathbf{n}^m\right)^T}\right)^T \sum_{l \in L_m^b} \left(\frac{\partial \mathbf{lz}^{l,m}}{\partial \left(\mathbf{a}^m\right)^T}\right)^T \left(\frac{\partial \mathbf{n}^l}{\partial \left(\mathbf{lz}^{l,m}\right)^T}\right)^T \mathbf{s}^l \tag{7.24}$$

The previous equations simplify for the standard multilayer network:

$$\frac{\partial \mathbf{n}^l}{\partial (\mathbf{lz}^{l,m})^T} = \mathbf{I} \tag{7.25}$$

$$\frac{\partial \mathbf{lz}^{l,m}}{\partial (\mathbf{a}^m)^T} = \mathbf{LW}^{l,m} \tag{7.26}$$

$$\frac{\partial \mathbf{a}^m}{\partial \left(\mathbf{n}^m\right)^T} = \dot{\mathbf{F}}^m(\mathbf{n}^m) \tag{7.27}$$

$$\mathbf{s}^m = \dot{\mathbf{F}}^m(\mathbf{n}^m)^T \left(\mathbf{LW}^{m+1,m}\right)^T \mathbf{s}^{m+1} \tag{7.28}$$

which is the same as Eq. 3.49.

We can summarize the gradient calculations for the GLFNN as follows. First, we compute the sensitivities at all of the output layers $u \in U$

$$\mathbf{s}^u = \dot{\mathbf{F}}^u(\mathbf{n}^u)^T \frac{\partial F(\mathbf{x})}{\partial \mathbf{a}^u} \tag{7.29}$$

Then we compute the remaining sensitivities, $\mathbf{s}^m$, as m follows the backpropagation order (reverse of the simulation order).

$$\mathbf{s}^m = \dot{\mathbf{F}}^m(\mathbf{n}^m)^T \sum_{l \in L_m^b} \left(\frac{\partial \mathbf{lz}^{l,m}}{\partial \left(\mathbf{a}^m\right)^T}\right)^T \left(\frac{\partial \mathbf{n}^l}{\partial \left(\mathbf{lz}^{l,m}\right)^T}\right)^T \mathbf{s}^l \tag{7.30}$$

Finally, we compute the elements of the gradient:

$$\frac{\partial F(\mathbf{x})}{\partial iw_{i,j}^{m,l}} = (\mathbf{s}^m)^T \frac{\partial \mathbf{n}^m}{\partial (\mathbf{iz}^{m,l})^T} \frac{\partial \mathbf{iz}^{m,l}}{\partial iw_{i,j}^{m,l}} \tag{7.31}$$

$$\frac{\partial F(\mathbf{x})}{\partial lw_{i,j}^{m,l}} = (\mathbf{s}^m)^T \frac{\partial \mathbf{n}^m}{\partial (\mathbf{lz}^{m,l})^T} \frac{\partial \mathbf{lz}^{m,l}}{\partial lw_{i,j}^{m,l}} \tag{7.32}$$

$$\frac{\partial F(\mathbf{x})}{\partial b_i^m} = (\mathbf{s}^m)^T \frac{\partial \mathbf{n}^m}{\partial b_i^m} \tag{7.33}$$

If we have an LFNN and we use mean square error as the performance function, then the equations simplify to the following:

$$\mathbf{s}^u = -2\dot{\mathbf{F}}^u(\mathbf{n}^u)^T (\mathbf{t}^u - \mathbf{a}^u), u \in U \tag{7.34}$$

$$\mathbf{s}^m = \sum_{l \in L_m^b} \left(\dot{\mathbf{F}}^u(\mathbf{n}^m)\right)^T \left(\mathbf{LW}^{l,m}\right)^T \mathbf{s}^l \tag{7.35}$$

$$\frac{\partial F(\mathbf{x})}{\partial iw_{i,j}^{m,l}} = (\mathbf{s}^m)^T p_j^l \mathbf{e}_i = s_i^m p_j^l \tag{7.36}$$

$$\frac{\partial F(\mathbf{x})}{\partial lw_{i,j}^{m,l}} = (\mathbf{s}^m)^T a_j^l \mathbf{e}_i = s_i^m a_j^l \tag{7.37}$$

$$\frac{\partial F(\mathbf{x})}{\partial b_i^m} = (\mathbf{s}^m)^T \mathbf{e}_i = s_i^m \tag{7.38}$$

It is instructive to compare Eq. 7.35 for the LFNN with Eq. 3.49 for the standard multilayer network. Because the LFNN layers can connect forward to several other layers, the backpropagation path needs to sum over all of these connections. In the standard multilayer network each layer only connects forward to one layer, so there is only one term in the backpropagation equation.

Epilogue

The key concept in this chapter is *modularity*. Network outputs are computed one layer at a time in the simulation order. The computation in each layer follows the same order: weight function – net input function – transfer function. The same modularity continues for the computation of the gradient of performance for static networks. It is computed one layer at a time in the backpropagation order. To compute the gradient, for each layer, you need only

- $\dot{\mathbf{F}}^m(\mathbf{n}^m)$ – the derivative of the transfer function
- $\frac{\partial \mathbf{lz}^{l,m}}{\partial(\mathbf{a}^m)}$ – the derivative of the weight function with respect to the layer input.
- $\frac{\partial \mathbf{lz}^{m,l}}{\partial lw_{i,j}^{m,l}}$ – the derivative of the weight function with respect to the weight.
- $\frac{\partial \mathbf{n}^m}{\partial(\mathbf{lz}^{m,l})^T}$ – the derivative of the net function with respect to the weight output.
- $\frac{\partial \mathbf{n}^m}{\partial b_i^m}$ – the derivative of the net function with respect to the bias.

In the next chapter we will use the concepts developed in this chapter to introduce the convolution neural network, which takes the form of a GLFNN.

Further Reading

[Hagan et al., 2014] Martin T Hagan, Howard B Demuth, Mark Beale, and Orlando De Jesús. *Neural Network Design (2nd Edition)*. Martin Hagan, 2014

Chapter 14 of this text covers layered digital dynamic networks in detail, including backpropagation through time and real-time recurrent learning algorithms for computing the gradient.

[De Jesús and Hagan, 2007] Orlando De Jesús and Martin T Hagan. Backpropagation algorithms for a broad class of dynamic networks. *IEEE Transactions on Neural Networks*, 18(1):14–27, 2007

This paper covers layered digital dynamic networks. It presents both gradient and Jacobian calculations using both backpropagation through time and real-time recurrent learning. It investigates the complexities of each algorithm as a function of the size of the network and the length of the time sequence.

[Hagan and De Jesús, 2007] Martin T Hagan and Orlando De Jesús. A general framework for dynamic networks. In *Research in Computing Science*, volume 15, pages 3–12, 2007

This paper generalizes the layered digital dynamic network and derives the real-time recurrent learning algorithm for gradient calculation.

[De Jesús and Hagan, 2008] Orlando De Jesús and Martin T Hagan. Backpropagation through time for general dynamic networks. In *IC-AI*, pages 45–51, 2008

This paper generalizes the layered digital dynamic network and derives the back-propagation through time algorithm for gradient calculation.

Summary of Results

Definition of a Layer

- A set of weight matrices, and associated weight functions, that come into that layer (which may connect from other layers or from external inputs),
- Any tapped delay lines that appear at the input of a weight matrix
- A bias vector,
- A net input function (e.g., a summing junction), and
- An activation function.

Multilayer Network Equations

$$\begin{aligned}\mathbf{a}^0 &= \mathbf{p}\\ \mathbf{n}^{m+1} &= \mathbf{W}^{m+1}\mathbf{a}^m + \mathbf{b}^{m+1}, m = 0, 1, ..., M-1\\ \mathbf{a}^m &= \mathbf{f}^m(\mathbf{n}^m)\end{aligned}$$

Layered Feedforward Network Equations

$$\begin{aligned}\mathbf{n}^m &= \sum_{i \in I_m} \mathbf{IW}^{m,l}\mathbf{p}^l + \sum_{i \in L_m^f} \mathbf{LW}^{m,l}\mathbf{a}^l + \mathbf{b}^m\\ \mathbf{a}^m &= \mathbf{f}^m(\mathbf{n}^m)\end{aligned}$$

where $\mathbf{p}^l$ is the l^{th} input vector to the network, $\mathbf{IW}^{m,l}$ is the input weight between input l and layer m, $\mathbf{LW}^{m,l}$ is the layer weight between layer l and layer m, $\mathbf{b}^m$ is the bias vector for layer m, I_m is the set of indices of input vectors that connect to layer m, and L_m^f is the set of indices of layers that directly connect forward to layer m.

Layered Digital Dynamic Network Equations

$$\mathbf{n}^m(t) = \sum_{l \in L_m^f} \sum_{d \in DL_{m,l}} \mathbf{LW}^{m,l}(d)\mathbf{a}^l(t-d) \tag{7.39}$$

$$+ \sum_{l \in I_m} \sum_{d \in DI_{m,l}} \mathbf{IW}^{m,l}(d)\mathbf{p}^l(t-d) + \mathbf{b}^m \tag{7.40}$$

$$\mathbf{a}^m(t) = \mathbf{f}^m(\mathbf{n}^m(t)) \tag{7.41}$$

where $DI_{m,l}$ and $DL_{m,l}$ represent the sets of the delays to Layer m from Input or Layer l, respectively.

Generalized Layered Digital Dynamic Network Equations

$$\mathbf{iz}^{m,l}(t,d) = \mathbf{ih}^{m,l}(\mathbf{IW}^{m,l}(d), \mathbf{p}^l(t-d))$$

$$\mathbf{lz}^{m,l}(t,d) = \mathbf{lh}^{m,l}(\mathbf{LW}^{m,l}(d), \mathbf{a}^l(t-d))$$

$$\mathbf{n}^m(t) = \mathbf{o}^m(\mathbf{iz}^{m,l}(t,d)|_{d \in DI_{m,l}}^{l \in I_m}, \mathbf{lz}^{m,l}(t,d)|_{d \in DL_{m,l}}^{l \in L_m^f}, \mathbf{b}^m)$$

$$\mathbf{a}^m(t) = \mathbf{f}^m(\mathbf{n}^m(t))$$

where $\mathbf{ih}^{m,l}()$ and $\mathbf{lh}^{m,l}()$ are input and layer weight functions, and $\mathbf{o}^m()$ is the net input function at Layer m.

Generalized LFNN Gradient Calculations

$$\mathbf{s}^u = \dot{\mathbf{F}}^u(\mathbf{n}^u)^T \frac{\partial F(\mathbf{x})}{\partial \mathbf{a}^u}$$

$$\mathbf{s}^m = \dot{\mathbf{F}}^m(\mathbf{n}^m)^T \sum_{l \in L_m^b} \left(\frac{\partial \mathbf{lz}^{l,m}}{\partial (\mathbf{a}^m)^T} \right)^T \left(\frac{\partial \mathbf{n}^l}{\partial \left(\mathbf{lz}^{l,m}\right)^T} \right)^T \mathbf{s}^l$$

$$\frac{\partial F(\mathbf{x})}{\partial iw_{i,j}^{m,l}} = (\mathbf{s}^m)^T \frac{\partial \mathbf{n}^m}{\partial (\mathbf{iz}^{m,l})^T} \frac{\partial \mathbf{iz}^{m,l}}{\partial iw_{i,j}^{m,l}}$$

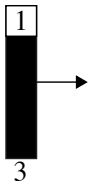
$$\frac{\partial F(\mathbf{x})}{\partial lw_{i,j}^{m,l}} = (\mathbf{s}^m)^T \frac{\partial \mathbf{n}^m}{\partial (\mathbf{lz}^{m,l})^T} \frac{\partial \mathbf{lz}^{m,l}}{\partial lw_{i,j}^{m,l}}$$

$$\frac{\partial F(\mathbf{x})}{\partial b_i^m} = (\mathbf{s}^m)^T \frac{\partial \mathbf{n}^m}{\partial b_i^m}$$

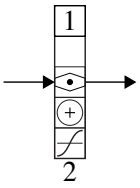
Solved Problems

P7.1 **Before stating the problem, it will be useful to introduce some shorthand notation, based on the notation in Solved Problem P14.1 in NND2.**

Input #1 with 3 elements.



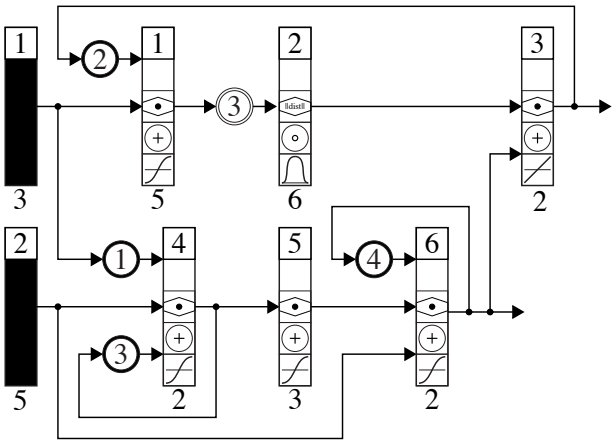
Layer #1 with 2 tansig neurons, dot product weight function and summation net input function.



Tapped delay from 1 to 9.

Tapped delay from 0 to 9.

Using this notation, consider the following network.



i. **Find the sets I_m.**

ii. **Find the sets L_m^f.**

iii. **Find a Simulation Order.**

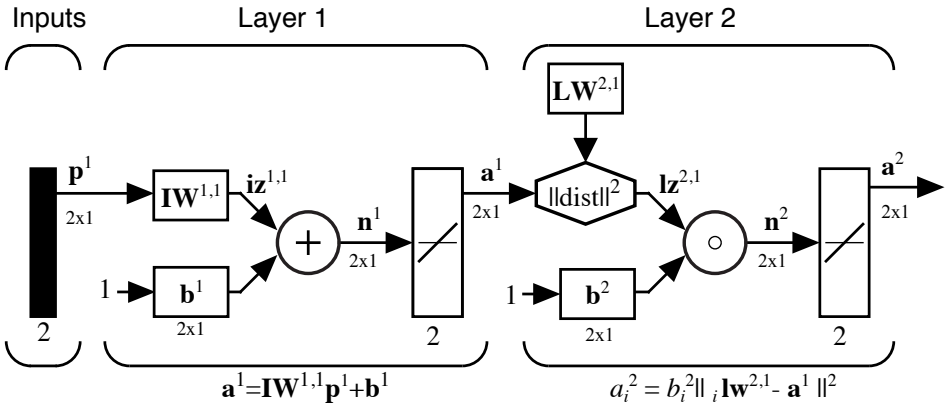
i. Neither of the inputs connect to Layers 2, 3 and 5, so I_m will be the null set for $m = 2, 3, 5$. Layer 1 is connected to the first input

only, Layer 4 is connected to both inputs, and Layer 6 is connected to the second input, therefore $I_1 = \{1\}$, $I_4 = \{1,2\}$ and $I_6 = \{2\}$.

ii. L_m^f is the set of indices of layers that directly connect forward to Layer m. Therefore $L_1^f = \{3\}$, $L_2^f = \{1\}$, $L_3^f = \{2,6\}$, $L_4^f = \{4\}$, $L_5^f = \{4\}$, $L_6^f = \{5,6\}$.

iii. The simulation order for this network is not unique, but one possibility is $\{1,4,2,5,6,3\}$

P7.2 **Consider the following network, and assume that the performance function is sum squared error. The second layer weight function output is computed as $lz_i^{2,1} = \left\| {}_i\mathbf{lw}^{2,1} - \mathbf{a}^1 \right\|^2$. The net input function output is Hadamard product, and is computed as $n_i^2 = lz_i^{2,1} b_i^2$.**



i. Find the sensitivities $\mathbf{s}^1$ and $\mathbf{s}^2$.

ii. Find the gradient equations.

i. This network has the general form of a GLFNN, since the weight function in the second layer is not matrix multiplication. The sensitivities can be computed using Equations 7.29 and 7.30. We start with the last layer and Eq. 7.29.

$$\mathbf{s}^2 = \dot{\mathbf{F}}^2(\mathbf{n}^2)^T \frac{\partial F(\mathbf{x})}{\partial \mathbf{a}^2}$$

Since the transfer function in the last layer is linear, and there are two neurons in the last layer, we have:

$$\dot{\mathbf{F}}^2(\mathbf{n}^2) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

The performance function is sum square error:

$$F(\mathbf{x}) = \sum_{j=1}^{S^2} \left(t_j - a_j^2\right)^2$$

Therefore

$$\frac{\partial F(\mathbf{x})}{\partial \mathbf{a}^2} = -2\left(\mathbf{t} - \mathbf{a}^2\right)$$

Combining the last two equations, we have

$$\mathbf{s}^2 = -2\left(\mathbf{t} - \mathbf{a}^2\right)$$

For $\mathbf{s}^1$ we use Eq. 7.30.

$$\mathbf{s}^1 = \dot{\mathbf{F}}^1(\mathbf{n}^1)^T \left(\frac{\partial \mathbf{lz}^{2,1}}{\partial \left(\mathbf{a}^1\right)^T}\right)^T \left(\frac{\partial \mathbf{n}^2}{\partial \left(\mathbf{lz}^{2,1}\right)^T}\right)^T \mathbf{s}^2$$

For this network

$$lz_i^{2,1} = \left\| {}_i\mathbf{lw}^{2,1} - \mathbf{a}^1 \right\|^2 = \sum_{j=1}^{S^1} \left(lw_{i,j}^{2,1} - a_j^1\right)^2$$

Therefore

$$\frac{\partial lz_i^{2,1}}{\partial a_j^1} = -2\left(lw_{i,j}^{2,1} - a_j^1\right)$$

In matrix form, this gives us

$$\frac{\partial \mathbf{lz}^{2,1}}{\partial \left(\mathbf{a}^1\right)^T} = \begin{bmatrix} -2\left(lw_{1,1}^{2,1} - a_1^1\right) & -2\left(lw_{1,2}^{2,1} - a_2^1\right) \\ -2\left(lw_{2,1}^{2,1} - a_1^1\right) & -2\left(lw_{2,2}^{2,1} - a_2^1\right) \end{bmatrix}$$

For the net input $\mathbf{n}^2$, we can write

$$\mathbf{n}^2 = \mathbf{lz}^{2,1} \circ \mathbf{b}^2$$

Therefore,

$$\frac{\partial \mathbf{n}^2}{\partial \left(\mathbf{lz}^{2,1}\right)^T} = \begin{bmatrix} b_1^2 & 0 \\ 0 & b_2^2 \end{bmatrix}$$

Finally, because we have a linear activation function in the first layer,

$$\dot{\mathbf{F}}^1(\mathbf{n}^1) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Putting this together, we find

$$\mathbf{s}^1 = \begin{bmatrix} -2\left(lw_{1,1}^{2,1} - a_1^1\right) & -2\left(lw_{1,2}^{2,1} - a_2^1\right) \\ -2\left(lw_{2,1}^{2,1} - a_1^1\right) & -2\left(lw_{2,2}^{2,1} - a_2^1\right) \end{bmatrix}^T \times \begin{bmatrix} b_1^2 & 0 \\ 0 & b_2^2 \end{bmatrix}^T \times \mathbf{s}^2$$

Multiplying out the first two terms, we have

$$\mathbf{s}^1 = -2 \begin{bmatrix} b_1^2\left(lw_{1,1}^{2,1} - a_1^1\right) & b_2^2\left(lw_{2,1}^{2,1} - a_1^1\right) \\ b_1^2\left(lw_{1,2}^{2,1} - a_2^1\right) & b_2^2\left(lw_{2,2}^{2,1} - a_2^1\right) \end{bmatrix} \begin{bmatrix} s_1^2 \\ s_2^2 \end{bmatrix}$$

If we write this in terms of individual components, we have

$$s_j^1 = -2\sum_{i=1}^{2} b_i^2\left(lw_{i,j}^{2,1} - a_j^1\right) s_i^2 \tag{7.42}$$

ii. For the gradient calculations, we use Equations 7.31 through 7.33:

$$\frac{\partial F(\mathbf{x})}{\partial iw_{i,j}^{m,l}} = (\mathbf{s}^m)^T \frac{\partial \mathbf{n}^m}{\partial (\mathbf{iz}^{m,l})^T} \frac{\partial \mathbf{iz}^{m,l}}{\partial iw_{i,j}^{m,l}}$$

$$\frac{\partial F(\mathbf{x})}{\partial lw_{i,j}^{m,l}} = (\mathbf{s}^m)^T \frac{\partial \mathbf{n}^m}{\partial (\mathbf{lz}^{m,l})^T} \frac{\partial \mathbf{lz}^{m,l}}{\partial lw_{i,j}^{m,l}}$$

$$\frac{\partial F(\mathbf{x})}{\partial b_i^m} = (\mathbf{s}^m)^T \frac{\partial \mathbf{n}^m}{\partial b_i^m}$$

For Layer 1, we have

$$\frac{\partial F(\mathbf{x})}{\partial iw_{i,j}^{1,1}} = \left(\mathbf{s}^1\right)^T \frac{\partial \mathbf{n}^1}{\partial (\mathbf{iz}^{1,1})^T} \frac{\partial \mathbf{iz}^{1,1}}{\partial iw_{i,j}^{1,1}}$$

$$\frac{\partial F(\mathbf{x})}{\partial b_i^1} = \left(\mathbf{s}^1\right)^T \frac{\partial \mathbf{n}^1}{\partial b_i^1}$$

Since the net input function in the first layer is summation, and $\mathbf{iz}^{1,1}$ is matrix multiplication,

$$\frac{\partial F(\mathbf{x})}{\partial iw_{i,j}^{1,1}} = \left(\mathbf{s}^1\right)^T p_j^1 \mathbf{e}_i = s_i^1 p_j^1$$
$$\frac{\partial F(\mathbf{x})}{\partial b_i^1} = \left(\mathbf{s}^1\right)^T \mathbf{e}_i = s_i^1$$

For Layer 2, we have

$$\frac{\partial F(\mathbf{x})}{\partial lw_{i,j}^{2,1}} = \left(\mathbf{s}^2\right)^T \frac{\partial \mathbf{n}^2}{\partial (\mathbf{lz}^{2,1})^T} \frac{\partial \mathbf{lz}^{2,1}}{\partial lw_{i,j}^{2,1}}$$
$$\frac{\partial F(\mathbf{x})}{\partial b_i^2} = \left(\mathbf{s}^2\right)^T \frac{\partial \mathbf{n}^2}{\partial b_i^2}$$

The net input function is Hadamard product, and

$$lz_i^{2,1} = \left\| {}_i\mathbf{lw}^{2,1} - \mathbf{a}^1 \right\|^2 = \sum_{j=1}^{S^1} \left(lw_{i,j}^{2,1} - a_j^1\right)^2$$

Therefore

$$\frac{\partial lz_i^{2,1}}{\partial lw_{i,j}^{2,1}} = 2\left(lw_{i,j}^{2,1} - a_j^1\right)$$

and from earlier we know

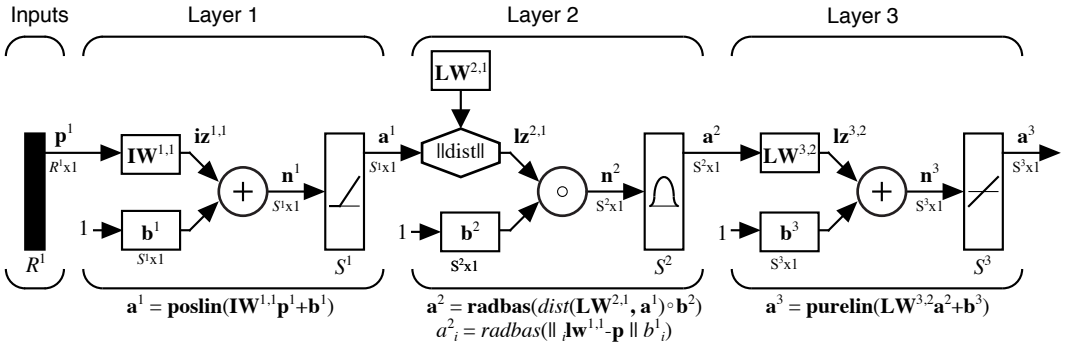
$$\frac{\partial \mathbf{n}^2}{\partial \left(\mathbf{lz}^{2,1}\right)^T} = \begin{bmatrix} b_1^2 & 0 \\ 0 & b_2^2 \end{bmatrix}$$

The gradients for the second layer are therefore

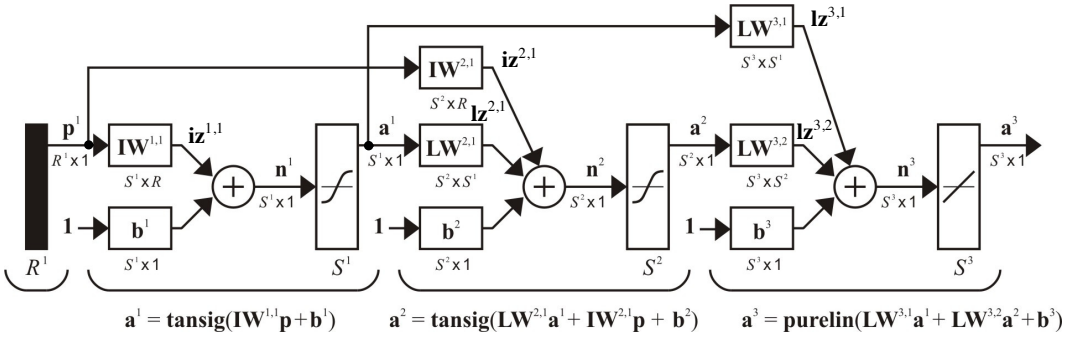
$$\frac{\partial F(\mathbf{x})}{\partial lw_{i,j}^{2,1}} = 2s_i^2 b_i^2 \left(lw_{i,j}^{2,1} - a_j^1\right)$$
$$\frac{\partial F(\mathbf{x})}{\partial b_i^2} = 2s_i^2 b_i^2 \sum_{j=1}^{2} \left(lw_{i,j}^{2,1} - a_j^1\right)^2$$

Exercises

E7.1 Consider the three layer network shown below. Write out the complete backpropagation equations for computing the gradient for this network. The radbas transfer function is defined as $radbas(n) = e^{-n^2}$. Assume the performance function is mean square error.



E7.2 Consider the following network.



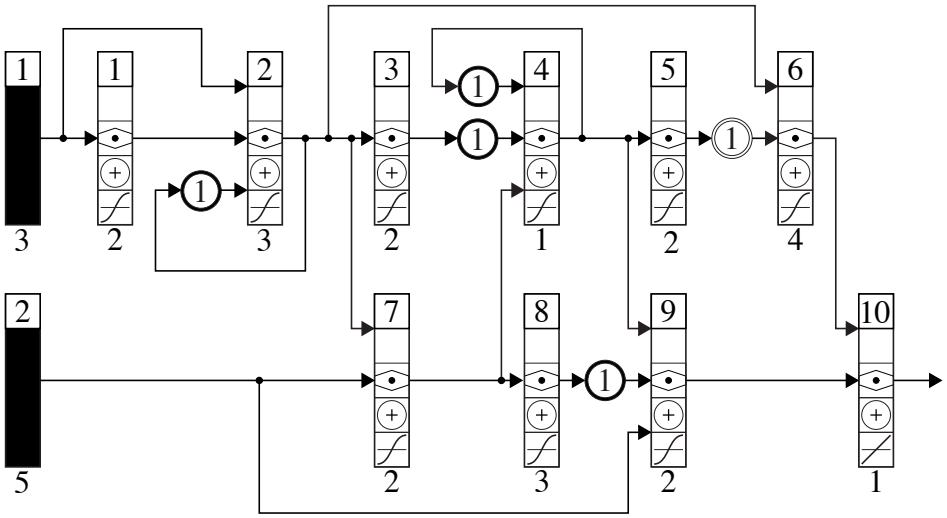
i. Find the sets I_m, L_m^f and L_m^b.

ii. Write out the equations for computing the sensitivities $\mathbf{s}^m$, assuming that $U = \{3\}$.

iii. Write out the equations for computing the gradient.

General Modular Network Notation

E7.3 Consider the following GLDDN network.



i. Find the sets I_m.

ii. Find the sets L_m^f.

iii. Find a Simulation Order.

E7.4 Suppose that the net input function for a certain layer is the Hadamard product. This means that all of the weight functions and the bias will be multiplied element by element, maintaining the same dimensions.

$$n_i^m = b_i^m \prod_{l \in L_m^f} lz_i^{m,l} \prod_{l \in I_m} iz_i^{m,l}$$

In order to train a network with this layer, we need to know the derivatives of the net input function.

i. Find $\frac{\partial \mathbf{n}^m}{\partial (\mathbf{iz}^{m,l})^T}$.

ii. Find $\frac{\partial \mathbf{n}^m}{\partial (\mathbf{lz}^{m,l})^T}$.

iii. Find $\frac{\partial \mathbf{n}^m}{\partial b_i^m}$.