

General Notation

Deep Learning



- Neural network notation is widely divergent.
- It was developed in fits and starts over many years.
- New architectures are often developed by different disciplines.
- This makes it difficult to see connections between models.
- We will extend the notation of NND2, and use the same notation to represent all architectures.
- The notation is based on the concept of a layer.



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 (MLP) Network

MLP Layer Equations

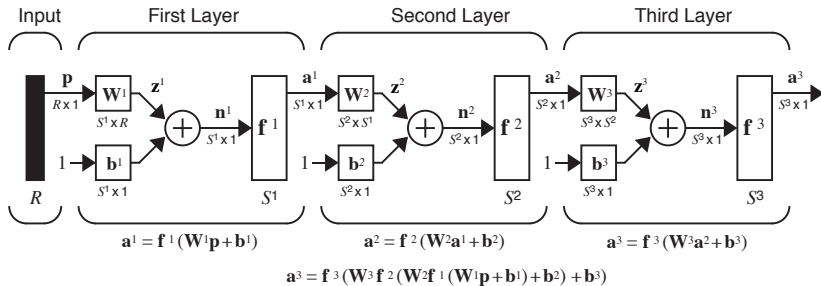
$$\mathbf{a}^0 = \mathbf{p}$$

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

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



Three layer example



Layered Feedforward Neural Network (LFNN)

LFNN Layer Equations

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

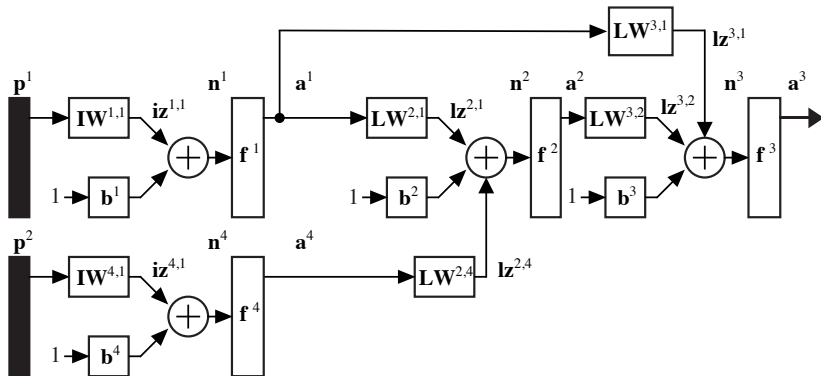
- \mathbf{p}^l - l th input to the network
- $\mathbf{IW}^{m,l}$ - input weight between input l and layer m
- $\mathbf{LW}^{m,l}$ - layer weight between layer l and layer m
- I_m - indices of input vectors that connect to layer m
- L_f^m - layers connecting directly forward to layer m



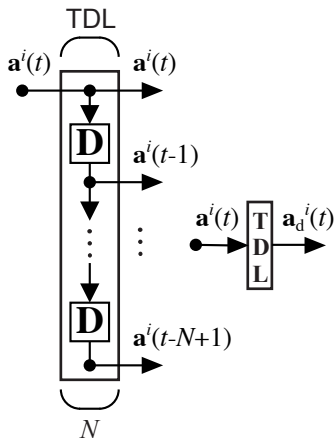
- For an MLP the output is computed starting at Layer 1 and proceeding in order to Layer M .
- For an LFNN, Layer 1 is not necessarily connected to Layer 2.
- No loops are allowed in an LFNN.
- We need to proceed in the proper layer order, so that the necessary inputs at each layer will be available.
- This ordering (which need not be unique) is called the **Simulation Order**.
- In the network on the next slide, one possible Simulation Order is 1-2-4-3.



Example LFNN



- Feedforward networks have no memory
- Outputs are computed only from the current inputs
- To include memory we add tapped delay lines to the input of layers



Layered Digital Dynamic Network (LDDN)

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

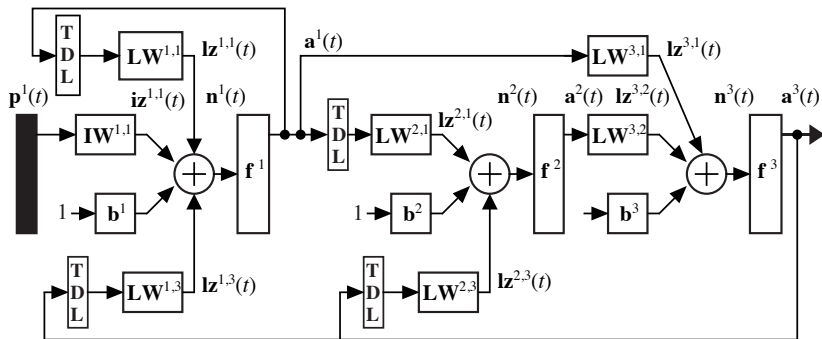
- $\mathbf{p}^l(t)$ - l th input to the network at time t
- $\mathbf{IW}^{m,l}(d)$ - weight between input l and layer m at delay d
- $\mathbf{LW}^{m,l}(d)$ - weight between layer l and layer m at delay d
- $DL_{m,l}$ - delays between layers l and m
- $DI_{m,l}$ - delays between input l and layer m



- An LDDN can have arbitrary connections between layers, but every feedback loop must contain at least one delay.
- Forward computations must be performed forward in time and forward in the simulation order for layers.
- LDDNs can have multiple input vectors and multiple output layers.



Example LDDN



Generalized LDDN (GLDDN)

- For the LDDN (and LFNN) the weight function is a standard matrix multiplication (dot product) between the weight matrix and the input to the layer.
- The net input function for an LDDN (or an LFNN) is a sum of the weight function outputs and the bias.
- The GLDDN (and GLFNN) can have arbitrary weight functions and net input functions.
- Forward computations for a GLDDN are performed in the same order as those for an LDDN with the same structure.



Weight Functions

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

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

Net Input Function

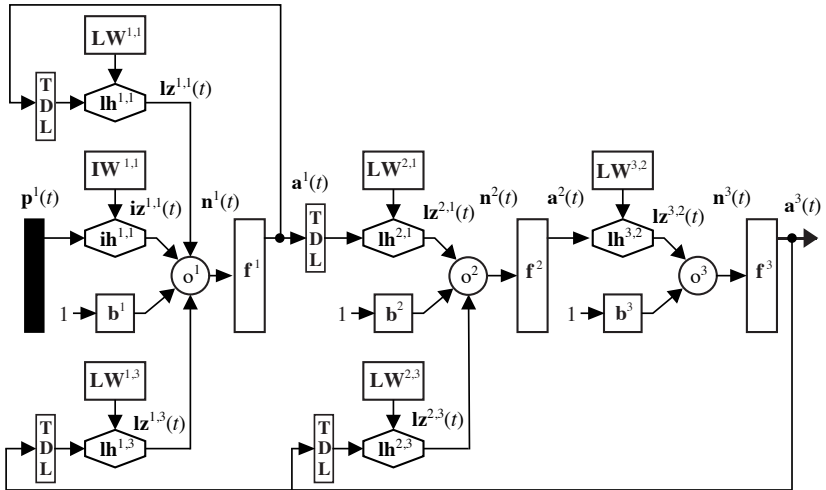
$$\mathbf{n}^m(t) = \mathbf{o}^m(\mathbf{iz}^{m,l}(t, d)|_{d \in DI_{m,l}}^{l \in I_m}, \mathbf{lz}^{m,l}(t, d)|_{d \in DL_{m,l}}^{l \in L_m^f}, \mathbf{b}^m)$$

Transfer Function

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



Example GLDNN



Gradient Calculation for a GLFNN

- A generalized LFNN can have multiple input vectors and multiple output layers.
- Let U be the set of output layer indices. (Output layers will be compared to targets.)
- The gradient of the performance function with respect to a weight would be

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



Weight and bias derivatives

$$\begin{aligned}\frac{\partial \mathbf{n}^m}{\partial i w_{i,j}^{m,l}} &= \frac{\partial \mathbf{n}^m}{\partial (\mathbf{i} \mathbf{z}^{m,l})^T} \frac{\partial \mathbf{i} \mathbf{z}^{m,l}}{\partial i w_{i,j}^{m,l}} \\ \frac{\partial \mathbf{n}^m}{\partial l w_{i,j}^{m,l}} &= \frac{\partial \mathbf{n}^m}{\partial (\mathbf{l} \mathbf{z}^{m,l})^T} \frac{\partial \mathbf{l} \mathbf{z}^{m,l}}{\partial l w_{i,j}^{m,l}} \\ \frac{\partial \mathbf{n}^m}{\partial b_i^m} &= \frac{\partial \mathbf{n}^m}{\partial b_i^m}\end{aligned}$$



The previous equations simplify for the standard MLP network.

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

$$\frac{\partial \mathbf{iz}^{m,l}}{\partial w_{i,j}^{m,l}} = p_j^l \mathbf{e}_i,$$

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

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

$$\frac{\partial \mathbf{lz}^{m,l}}{\partial w_{i,j}^{m,l}} = a_j^l \mathbf{e}_i$$

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



Backpropagation for a GLFNN

The sensitivity \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.

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

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 .

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



The previous equations simplify for the standard MLP network.

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

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

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

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



Summary GLFNN gradient calculations

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

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

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

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

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



$$\mathbf{s}^u = -2\dot{\mathbf{F}}^u(\mathbf{n}^u)^T (\mathbf{t}^u - \mathbf{a}^u)$$

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

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

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

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



Key concept – modularity

- Network outputs are computed one layer at a time in the simulation order (modular).
- Layer: weight function \rightarrow net input function \rightarrow transfer function.
- Gradient of performance is computed one layer at a time in the backpropagation order (modular).
- To compute the gradient, for each layer, you need only
 - $\dot{\mathbf{F}}^m(\mathbf{n}^m)$ – derivative of transfer function
 - $\frac{\partial \mathbf{z}^{l,m}}{\partial (\mathbf{a}^m)}$ – derivative of weight function w.r.t. layer input
 - $\frac{\partial \mathbf{z}^{m,l}}{\partial w_{i,j}^{m,l}}$ – derivative of weight function w.r.t. the weight
 - $\frac{\partial \mathbf{n}^m}{\partial (\mathbf{z}^{m,l})^T}$ – derivative of net function w.r.t. weight output
 - $\frac{\partial \mathbf{n}^m}{\partial b_i^m}$ – derivative of net function w.r.t. bias

