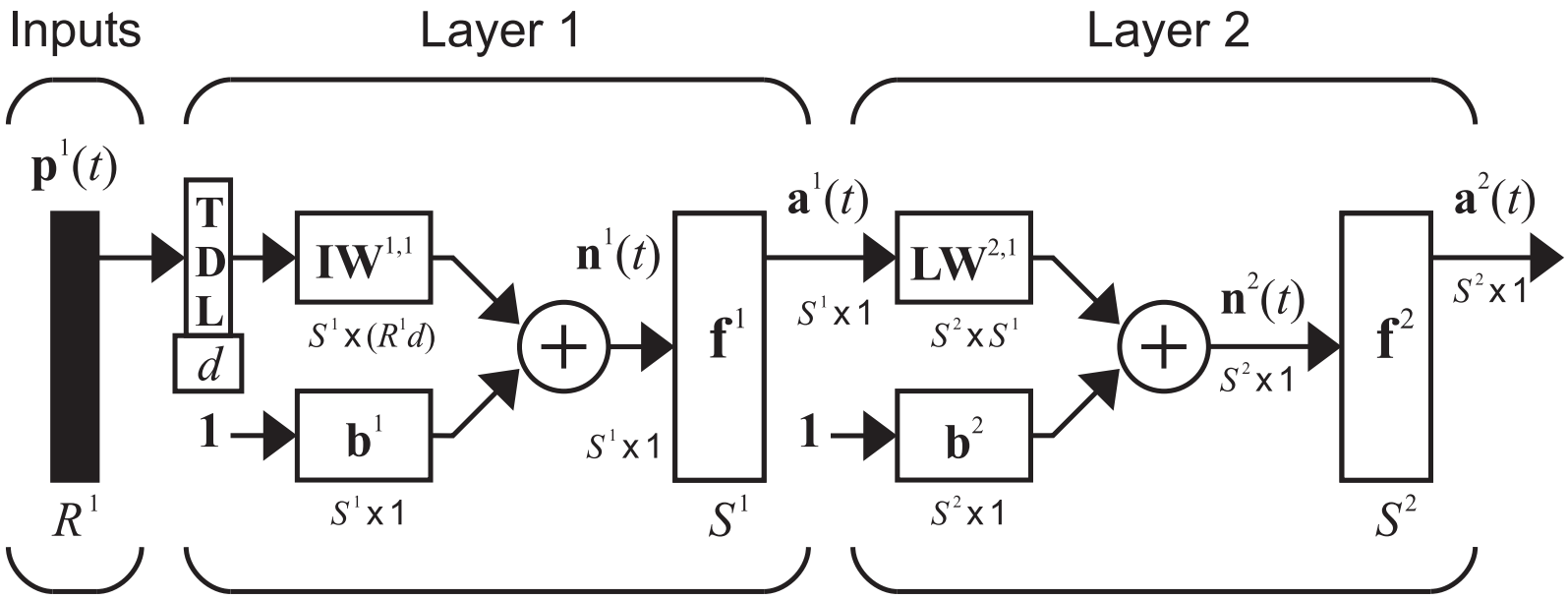
Nonlinear Sequence Processing

Deep Learning
Lecture

Objective

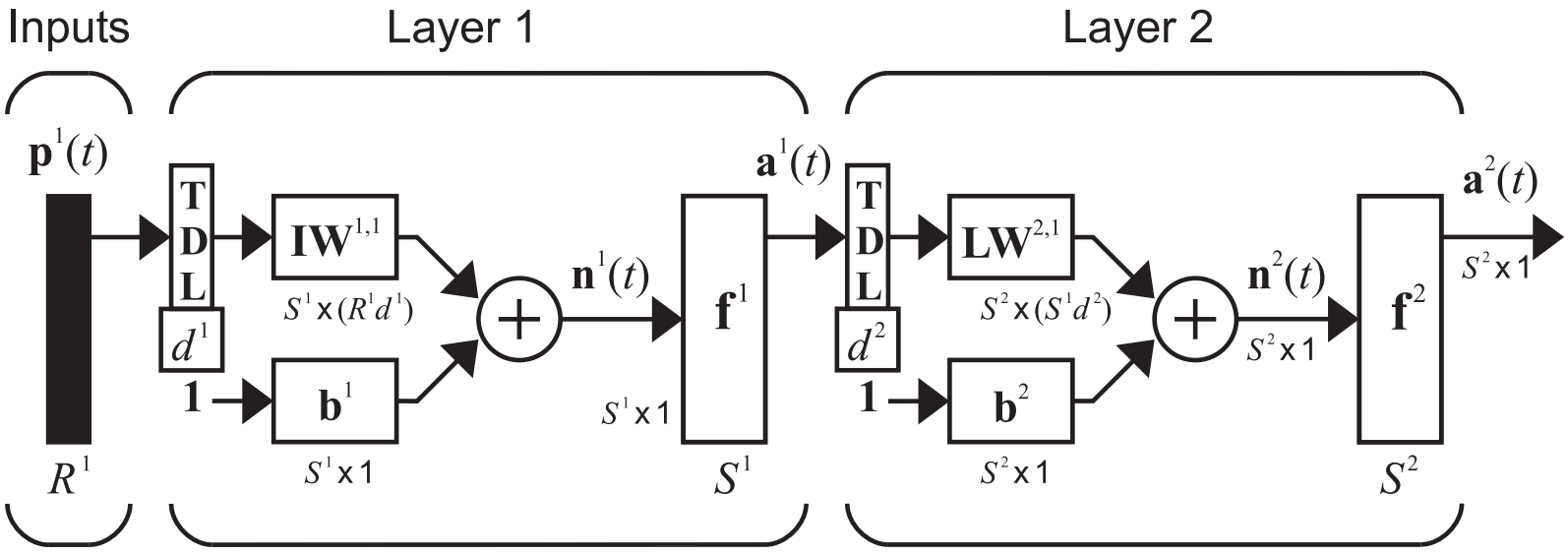
- The linear sequence processing concepts from the previous chapter lead into the nonlinear networks of this chapter.
- These networks are also dynamic – they have memory in terms of tapped delay lines.
- Some of these networks are strictly feedforward, and some have feedback – recurrent networks.
- Some networks are of the input-output type and use tapped delay lines.
- Other networks are of the state space type, and use single delays.

Focused time delay neural network



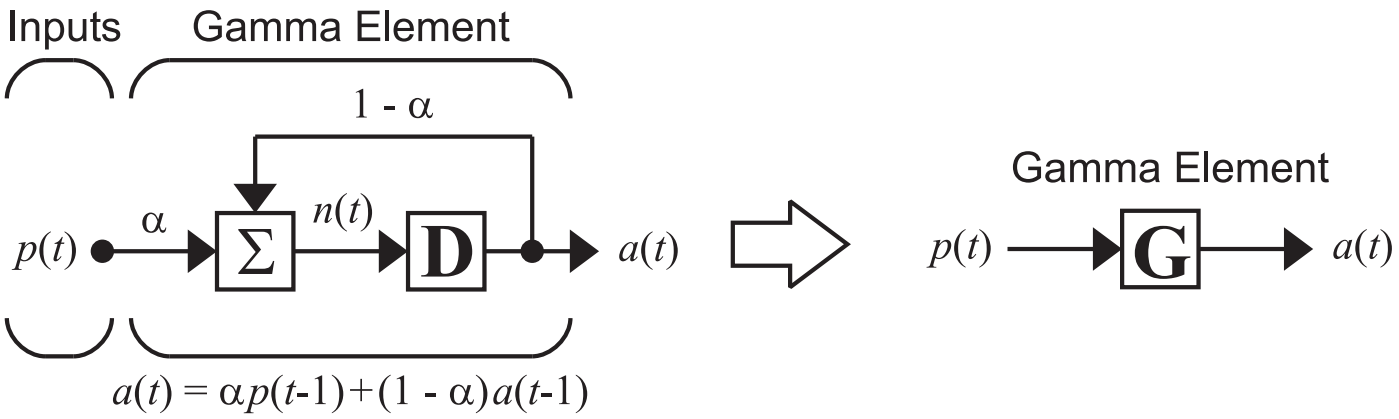
- The TDL is focused at the input to the network.
- An advantage of this dynamic network, because there is no feedback, is that computations can be done in parallel.
- Also, standard backpropagation can be used to compute the gradient.

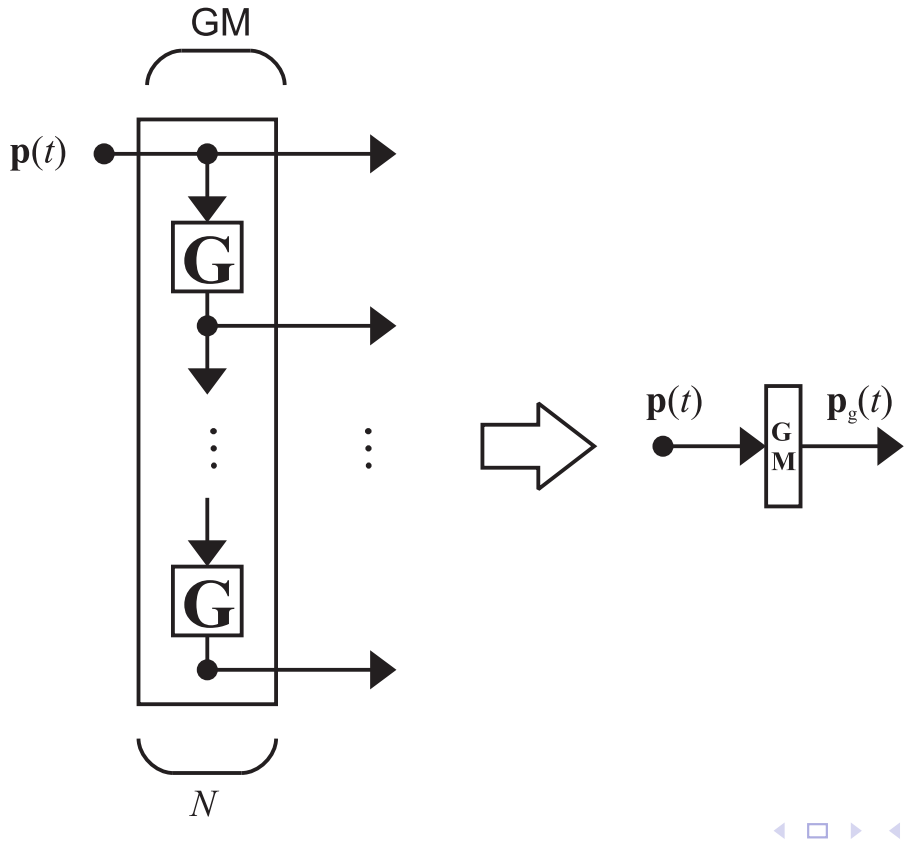
Distributed time delay neural network



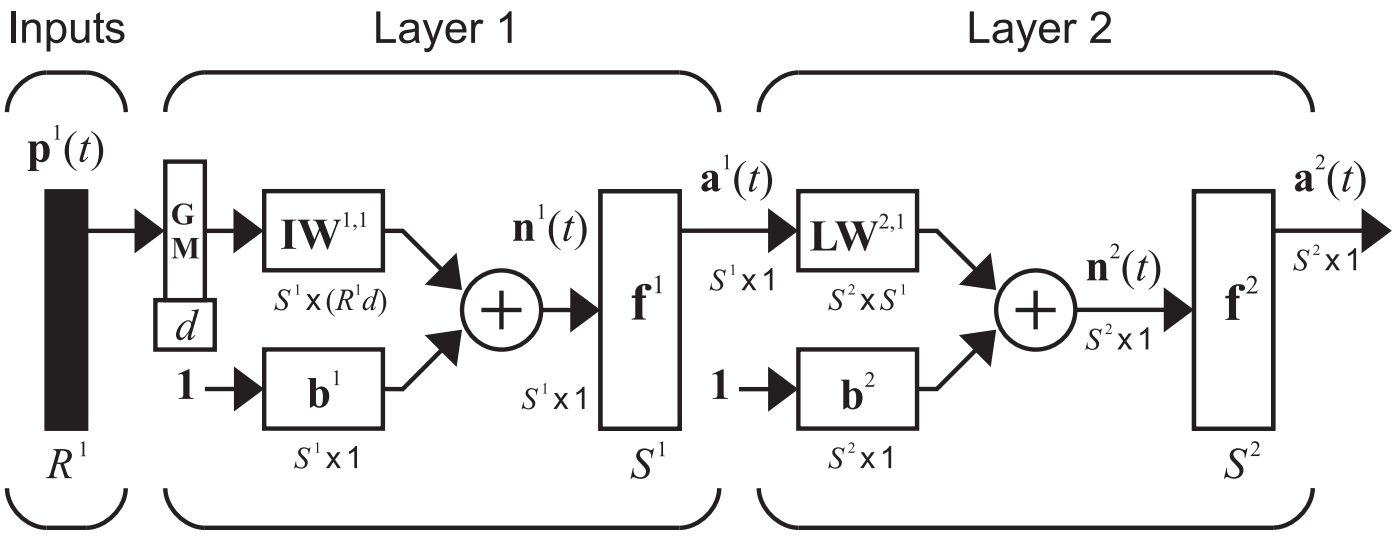
- Because tapped delay lines are distributed throughout the network, the backpropagation algorithm must be adjusted.

Gamma memory elements



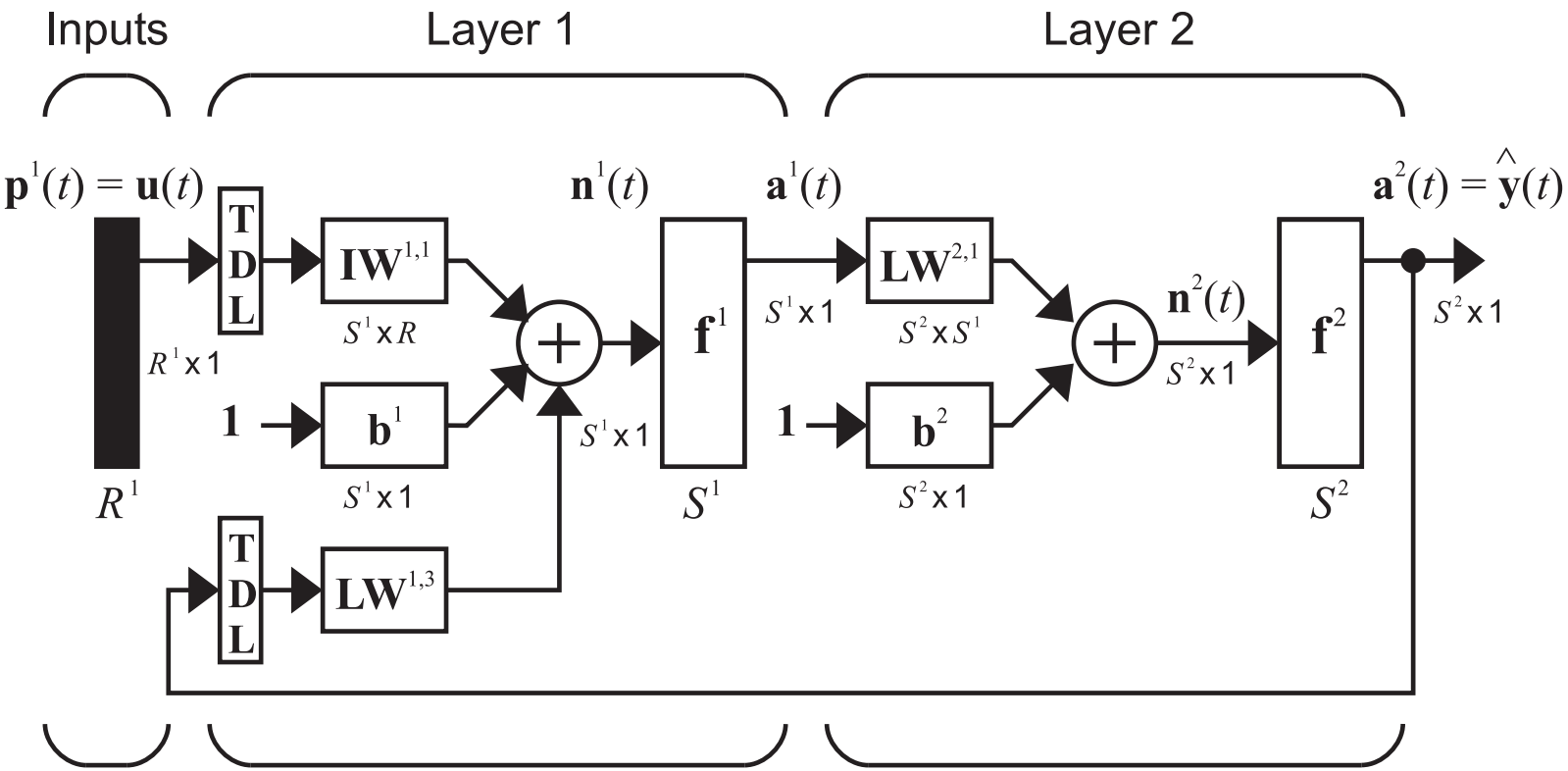


Focused gamma memory network

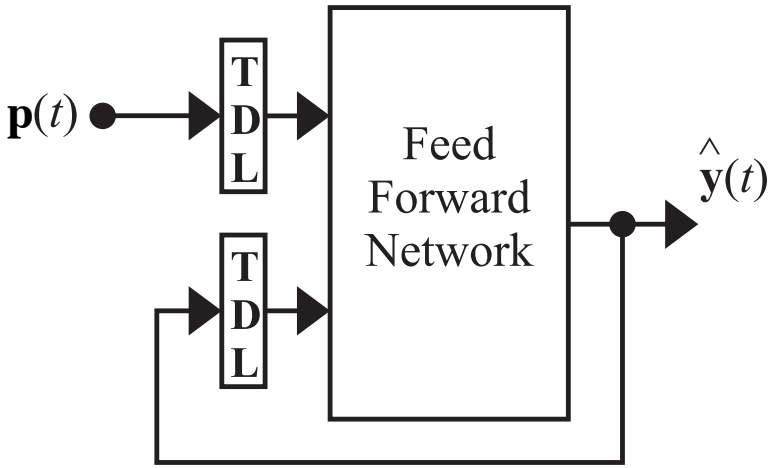


- This network does have some limited feedback, which gives it longer memory.
- Although the feedback is limited, the backpropagation algorithm does have to be somewhat modified.

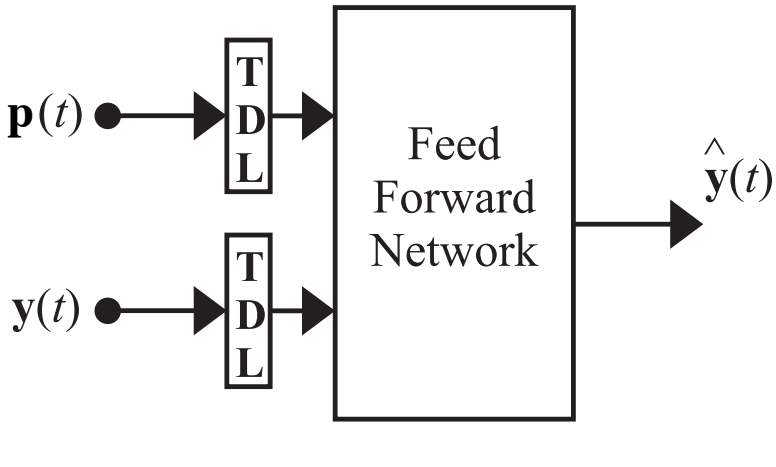
Nonlinear Autoregressive Network with Exogenous Input



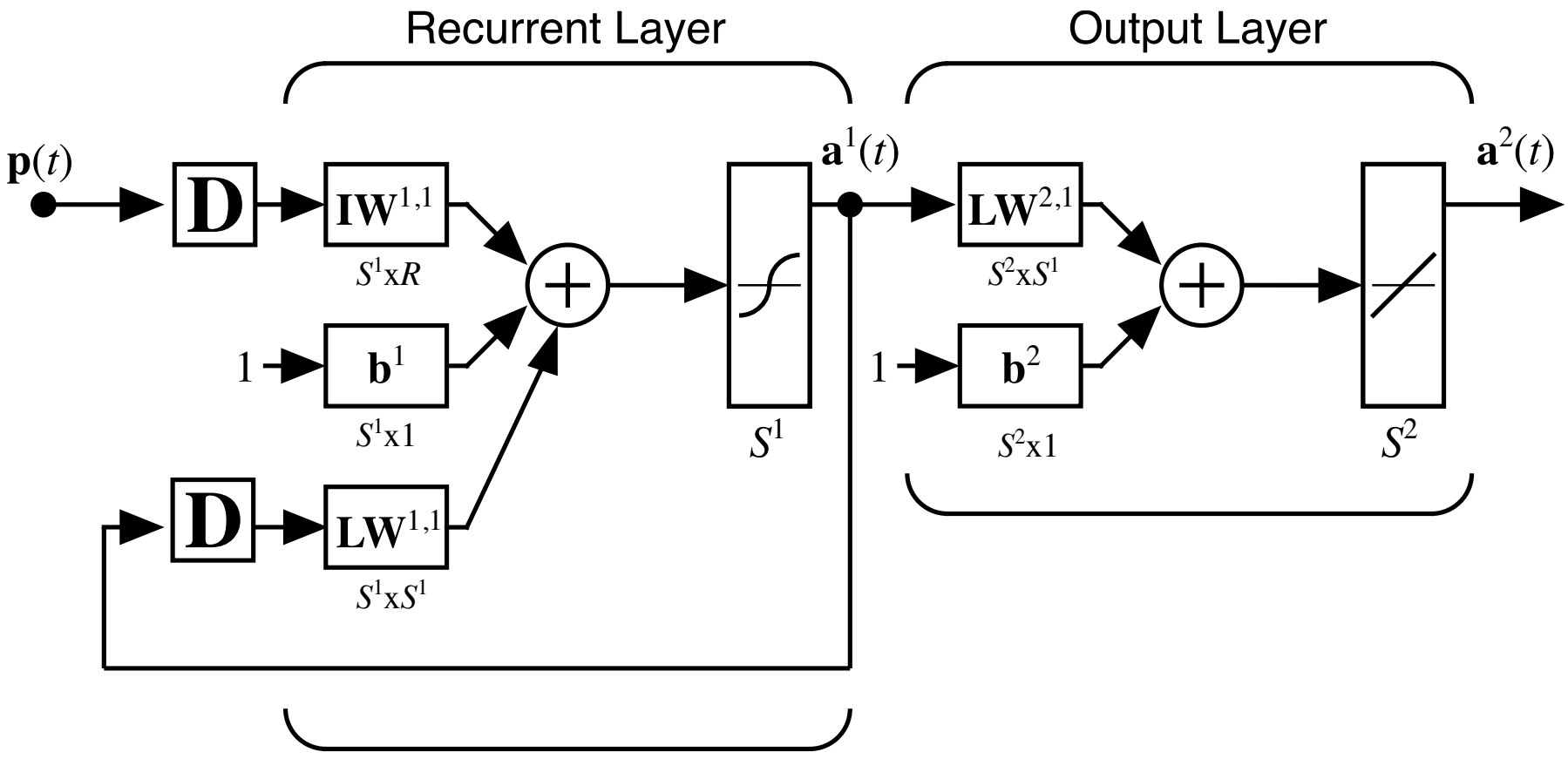
NARX can be trained with standard backprop



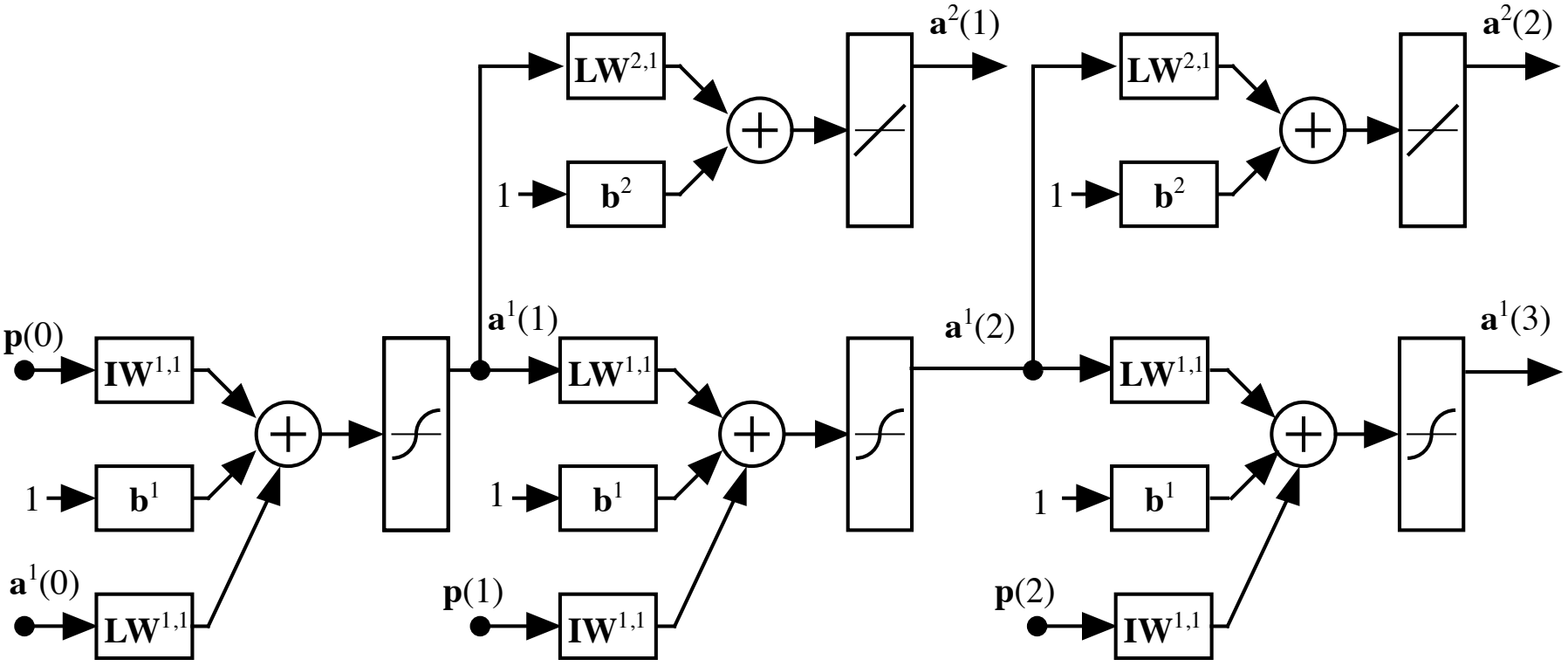
Parallel Architecture



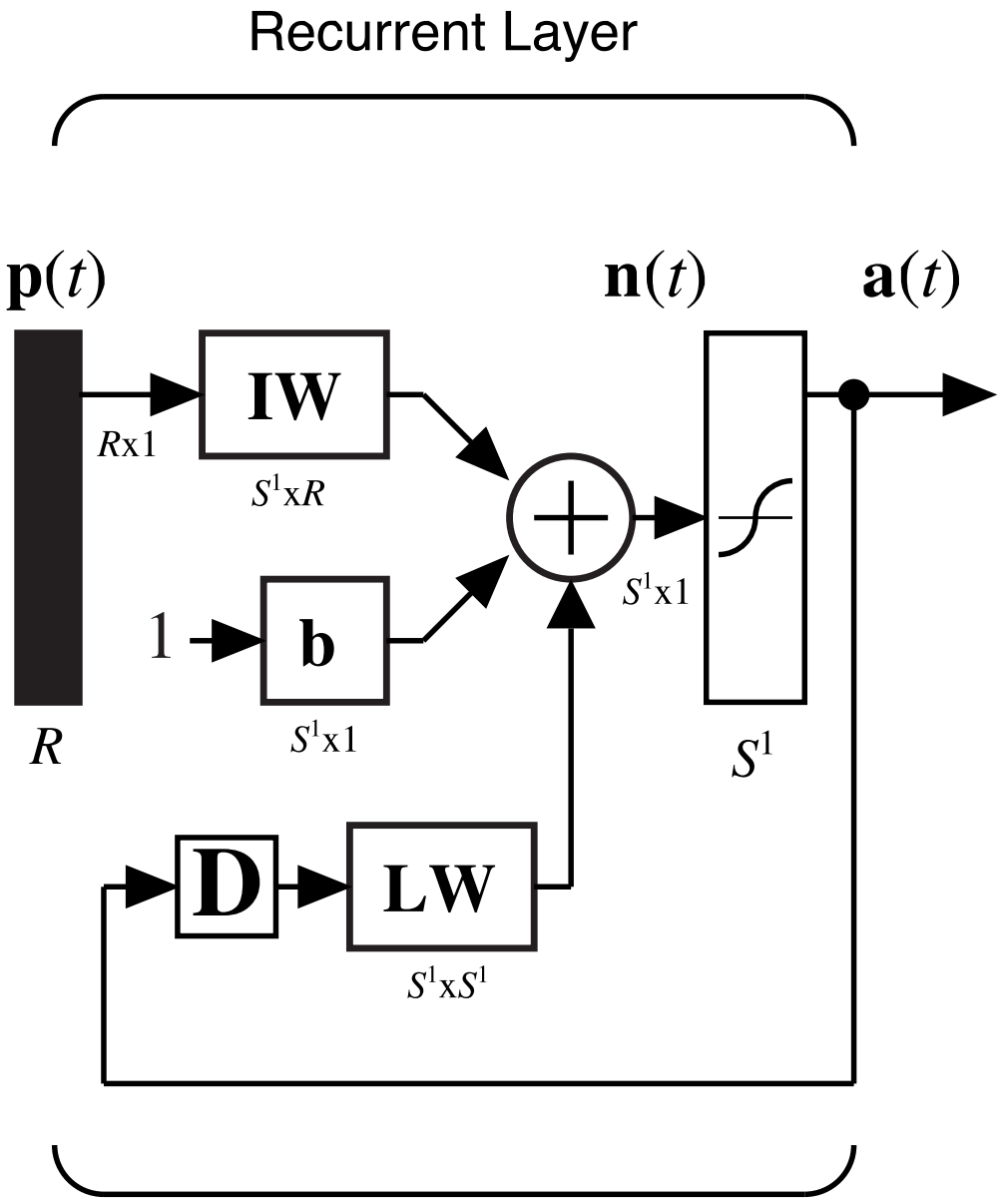
Series-Parallel Architecture


Recurrent Layer
Output Layer
p(t)
D
IW1,1
S1xR
1
b1
S1x1
D
LW1,1
S1xS1
S1
a1(t)
LW2,1
S2xS1
1
b2
S2x1
S2
a2(t)

Unrolled RNN



Basic RNN for illustration



Real Time Recurrent Learning (RTRL)

$$\frac{\partial F}{\partial \mathbf{x}} = \sum_{t=1}^{Q} \left[\frac{\partial \mathbf{a}(t)}{\partial \mathbf{x}^T} \right]^T \times \frac{\partial^e F}{\partial \mathbf{a}(t)}$$

$$\frac{\partial \mathbf{a}(t)}{\partial \mathbf{x}^T} = \frac{\partial^e \mathbf{a}(t)}{\partial \mathbf{x}^T} + \frac{\partial^e \mathbf{a}(t)}{\partial \mathbf{a}(t-1)^T} \frac{\partial \mathbf{a}(t-1)}{\partial \mathbf{x}^T}$$

Backpropagation Through Time (BTT)

$$\frac{\partial F}{\partial \mathbf{x}} = \sum_{t=1}^{Q} \left[\frac{\partial^e \mathbf{a}(t)}{\partial \mathbf{x}^T} \right]^T \times \frac{\partial F}{\partial \mathbf{a}(t)}$$

$$\frac{\partial F}{\partial \mathbf{a}(t)} = \frac{\partial^e F}{\partial \mathbf{a}(t)} + \frac{\partial^e \mathbf{a}(t+1)}{\partial \mathbf{a}(t)^T} \times \frac{\partial F}{\partial \mathbf{a}(t+1)}$$

- Let $Q = 2$, and let $F(\mathbf{x}) = (\mathbf{t} - \mathbf{a}(2))^T(\mathbf{t} - \mathbf{a}(2)) = \mathbf{e}^T\mathbf{e}$

$$\frac{\partial F}{\partial \mathbf{a}(t)} = \frac{\partial^e F}{\partial \mathbf{a}(t)} + \frac{\partial^e \mathbf{a}(t+1)}{\partial \mathbf{a}(t)^T} \times \frac{\partial F}{\partial \mathbf{a}(t+1)}$$

$$\frac{\partial F}{\partial \mathbf{a}(t)} = \mathbf{0}, t > Q$$

$$\frac{\partial F}{\partial \mathbf{a}(2)} = \frac{\partial^e F}{\partial \mathbf{a}(2)}^{-2\mathbf{e}} + \frac{\partial^e \mathbf{a}(3)}{\partial \mathbf{a}(2)^T} \times \frac{\partial F}{\partial \mathbf{a}(3)}^{0} = -2\mathbf{e}$$

$$\frac{\partial F}{\partial \mathbf{a}(1)} = \frac{\partial^e F}{\partial \mathbf{a}(1)}^{0} + \frac{\partial^e \mathbf{a}(2)}{\partial \mathbf{a}(1)^T} \times \frac{\partial F}{\partial \mathbf{a}(2)}^{-2\mathbf{e}} = [\mathbf{LW}]^T \dot{\mathbf{F}}(\mathbf{n}(2))^T [-2\mathbf{e}]$$

$$\frac{\partial F}{\partial \mathbf{x}} = \sum_{t=1}^{Q} \left[\frac{\partial^e \mathbf{a}(t)}{\partial \mathbf{x}^T} \right]^T \times \frac{\partial F}{\partial \mathbf{a}(t)}$$

$$\frac{\partial F}{\partial lw_{i,j}} = \sum_{t=1}^{2} \left[\frac{\partial^e \mathbf{a}(t)}{\partial lw_{i,j}} \right]^T \times \frac{\partial F}{\partial \mathbf{a}(t)}$$

$$\frac{\partial^e \mathbf{a}^T(t)}{\partial lw_{i,j}} \frac{\partial F}{\partial \mathbf{a}(t)} = \frac{\partial^e \mathbf{n}^T(t)}{\partial lw_{i,j}} \frac{\partial^e \mathbf{a}^T(t)}{\partial \mathbf{n}(t)} \frac{\partial F}{\partial \mathbf{a}(t)}$$

$$\frac{\partial^e \mathbf{n}(t)}{\partial lw_{i,j}} = \epsilon_i a_j(t-1)$$

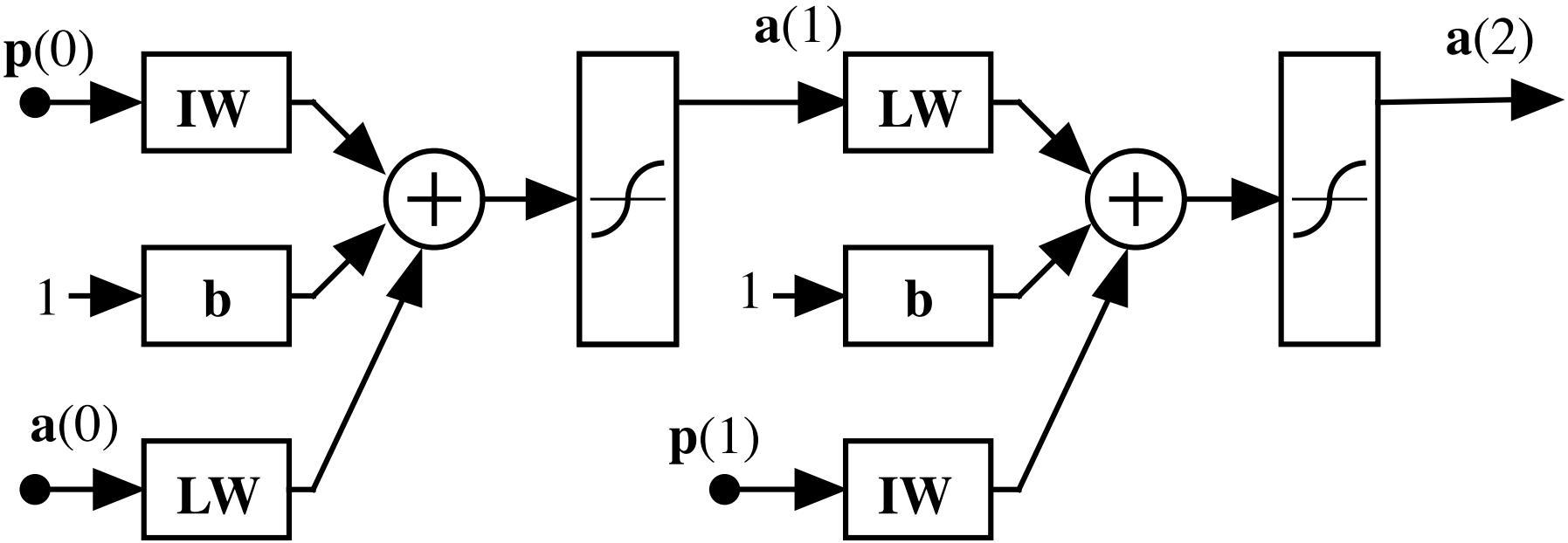
$$\frac{\partial^e \mathbf{a}^T(t)}{\partial \mathbf{n}(t)} = \dot{\mathbf{F}}^T(\mathbf{n}(t))$$

Gradient calculation 2

$$\frac{\partial^e \mathbf{a}^T(t)}{\partial lw_{i,j}} \frac{\partial F}{\partial \mathbf{a}(t)} = a_j(t-1)\dot{f}(n_i(t)) \frac{\partial F}{\partial a_i(t)}$$

$$\begin{aligned}
\frac{\partial F}{\partial \mathbf{LW}} &= \sum_{t=1}^{2} \dot{\mathbf{F}}^T(\mathbf{n}(t)) \frac{\partial F}{\partial \mathbf{a}(t)} \mathbf{a}^T(t-1) \\
&= \dot{\mathbf{F}}^T(\mathbf{n}(1)) \frac{\partial F}{\partial \mathbf{a}(1)} \mathbf{a}^T(0) + \dot{\mathbf{F}}^T(\mathbf{n}(2)) \frac{\partial F}{\partial \mathbf{a}(2)} \mathbf{a}^T(1) \\
&= \dot{\mathbf{F}}^T(\mathbf{n}(1)) \mathbf{LW}^T \dot{\mathbf{F}}^T(\mathbf{n}(2)) \left[-2\mathbf{e}\right] \mathbf{a}^T(0) \\
&\quad + \dot{\mathbf{F}}^T(\mathbf{n}(2)) \left[-2\mathbf{e}\right] \mathbf{a}^T(1)
\end{aligned}$$

Unrolled basic RNN



Standard Backpropagation for Multilayer Networks

$$\mathbf{s}^M = \dot{\mathbf{F}}^{\mathbf{M}}\left(\mathbf{n}^M\right)^T[-2\mathbf{e}]$$

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

$$\frac{\partial \hat{F}(\mathbf{x})}{\partial w_{i,j}^m} = s_i^m a_j^{m-1}$$

$$\frac{\partial F(\mathbf{x})}{\partial \mathbf{W}^m} = \mathbf{s}^m\left(\mathbf{a}^{m-1}\right)^T$$

Static backpropagation for unrolled network

$$\mathbf{s}(2) = \dot{\mathbf{F}}\left(\mathbf{n}(2)\right)^T \left[-2\mathbf{e}\right]$$

$$\mathbf{s}(1) = \dot{\mathbf{F}}(\mathbf{n}(1))^T \mathbf{L}\mathbf{W}^T \mathbf{s}(2)$$

$$= \dot{\mathbf{F}}(\mathbf{n}(1))^T \mathbf{L}\mathbf{W}^T \dot{\mathbf{F}}\left(\mathbf{n}(2)\right)^T \left[-2\mathbf{e}\right]$$

$$\left.\frac{\partial F(\mathbf{x})}{\partial \mathbf{LW}}\right|_{t=2} = \mathbf{s}(2)\mathbf{a}(1)^T = \dot{\mathbf{F}}\left(\mathbf{n}(2)\right)^T \left[-2\mathbf{e}\right] \mathbf{a}(1)^T$$

$$\left.\frac{\partial F(\mathbf{x})}{\partial \mathbf{LW}}\right|_{t=1} = \mathbf{s}(1)\mathbf{a}(0)^T = \dot{\mathbf{F}}(\mathbf{n}(1))^T \mathbf{L}\mathbf{W}^T \dot{\mathbf{F}}\left(\mathbf{n}(2)\right)^T \left[-2\mathbf{e}\right] \mathbf{a}(0)^T$$

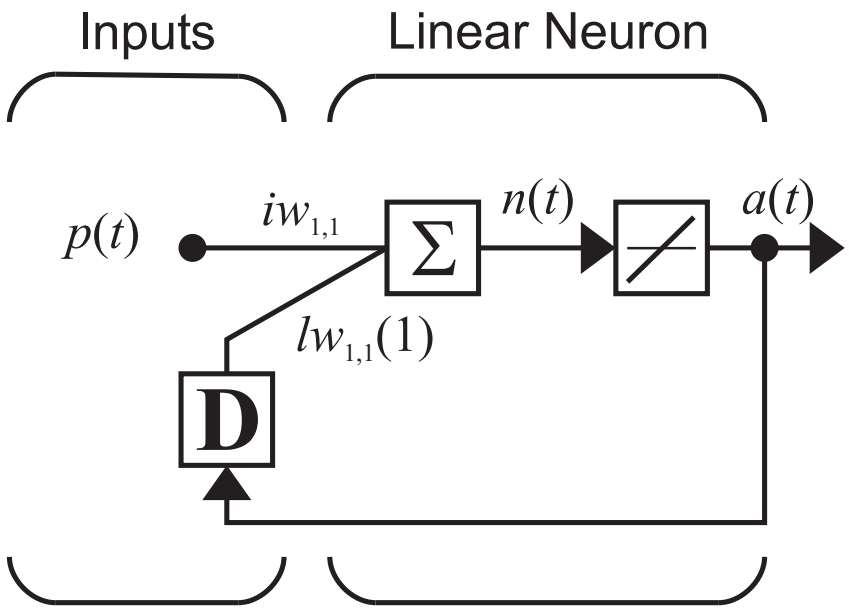
$$\left.\frac{\partial F(\mathbf{x})}{\partial \mathbf{LW}}\right|_{total} = \dot{\mathbf{F}}(\mathbf{n}(1))^T \mathbf{L}\mathbf{W}^T \dot{\mathbf{F}}\left(\mathbf{n}(2)\right)^T \left[-2\mathbf{e}\right] \mathbf{a}(0)^T$$

$$+ \dot{\mathbf{F}}\left(\mathbf{n}(2)\right)^T \left[-2\mathbf{e}\right] \mathbf{a}(1)^T$$

Learning long term dependencies

- For some recurrent network problems, we would like to predict responses that may be significantly delayed from the corresponding stimulus.
  - A chess game can be lost 12 moves after a mistake.
  - Words in a previous paragraph can provide context for a translation.
- A network structure must enable this possibility (have long term memory).
- Even within a structure that allows long term memory, the actual memory depends on the weights.
- Long memory systems can flirt with instability.

Simple recurrent network

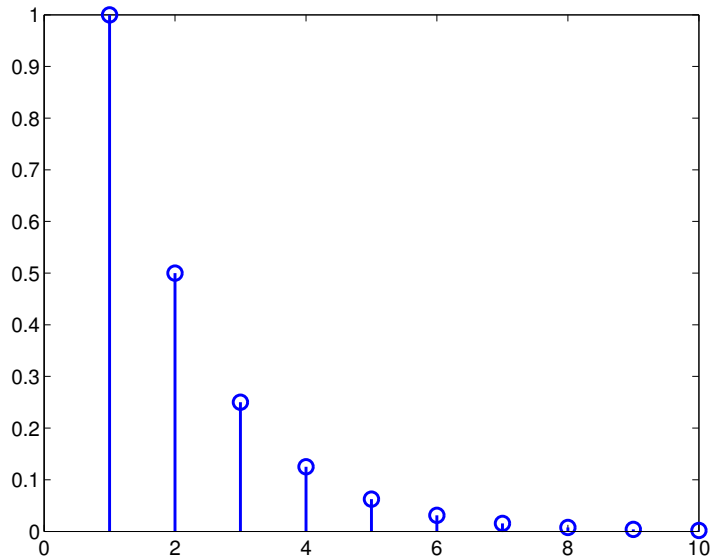


$$a(t) = iw_{1,1}p(t) + lw_{1,1}(1)a(t\text{-}1)$$

Network response

Let $lw_{1,1} = 0.5$ and $iw_{1,1} = 1$. Apply the input sequence of a one followed by 9 zeros, with zero initial condition (impulse response).

$$a(t) = iw_{1,1}p(t) + lw_{1,1}a(t-1) = p(t) + 0.5a(t-1)$$
$$a(1) = p(1) = 1$$
$$a(2) = 0.5a(1) = 0.5$$
$$a(t) = 0.5^{(t-1)}$$



Gradient also decays quickly

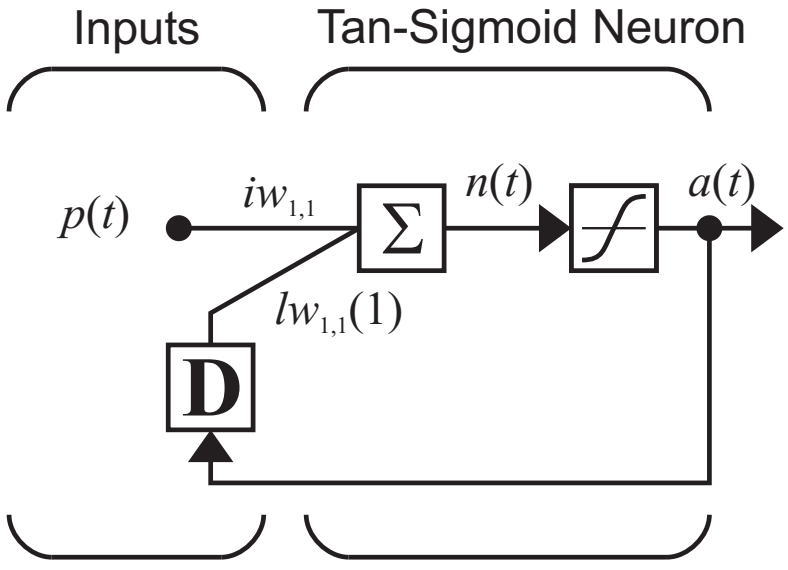
$$\frac{\partial F}{\partial a(t)} = \frac{\partial^e F}{\partial a(t)} + lw_{1,1} \frac{\partial F}{\partial a(t+1)}$$

If the error only occurs at the 10th time point, and $lw_{1,1} = 0.5$, we have

$$\frac{\partial F}{\partial a(t)} = 0.5^{(10-t)} \frac{\partial^e F}{\partial a(10)}$$

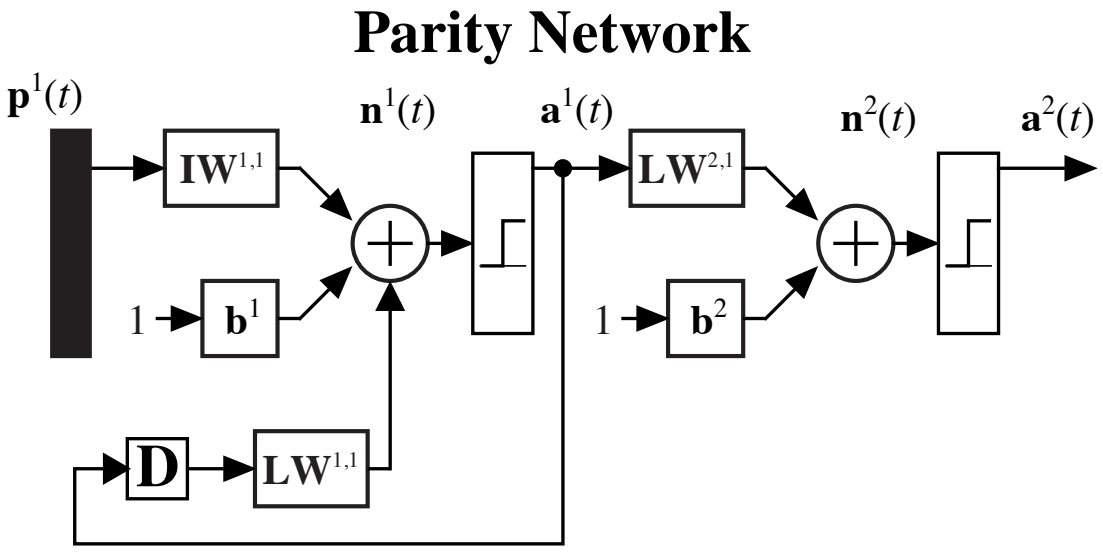
- If the network is stable, derivatives decay as you go back in time.
- Because the derivatives are small, it will take a long time to learn long term dependencies.

Nonlinear transfer function increases decay



$$\frac{\partial F}{\partial a(t)} = \frac{\partial^e F}{\partial a(t)} + \dot{f}(n(t)) lw_{1,1} \frac{\partial F}{\partial a(t+1)}, \left|\dot{f}(n(t))\right| \leq 1$$

Example network with long memory



$$\mathbf{IW}^{1,1} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \mathbf{LW}^{1,1} = \begin{bmatrix} 1 & -1 \\ 1 & -1 \end{bmatrix}, \mathbf{LW}^{2,1} = \begin{bmatrix} 1 & -1 \end{bmatrix}, \mathbf{b}^1 = \begin{bmatrix} -0.5 \\ -1.5 \end{bmatrix}, \mathbf{b}^2 = [-0.5]$$

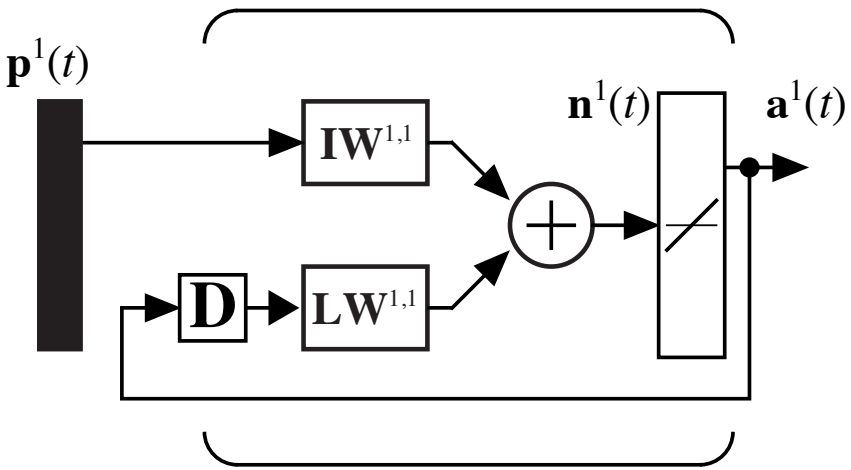
- This network computes the parity of a sequence of bits.
- It can remember back as many bits as have been presented.

Difficulties in learning long term dependencies

- Long term memories are network weights – short term memories are layer outputs.
- We need a network with long short term memory
- In recurrent networks, as the weights change during training, the length of the short term memory will change.
- If the initial weights produce a network without long short term memory, it will be difficult to increase it.
- This is because the gradient will be small for short short term memory networks.
- If the initial weights produce long short term memory, instabilities can easily occur.

Constant error carousel (CEC)

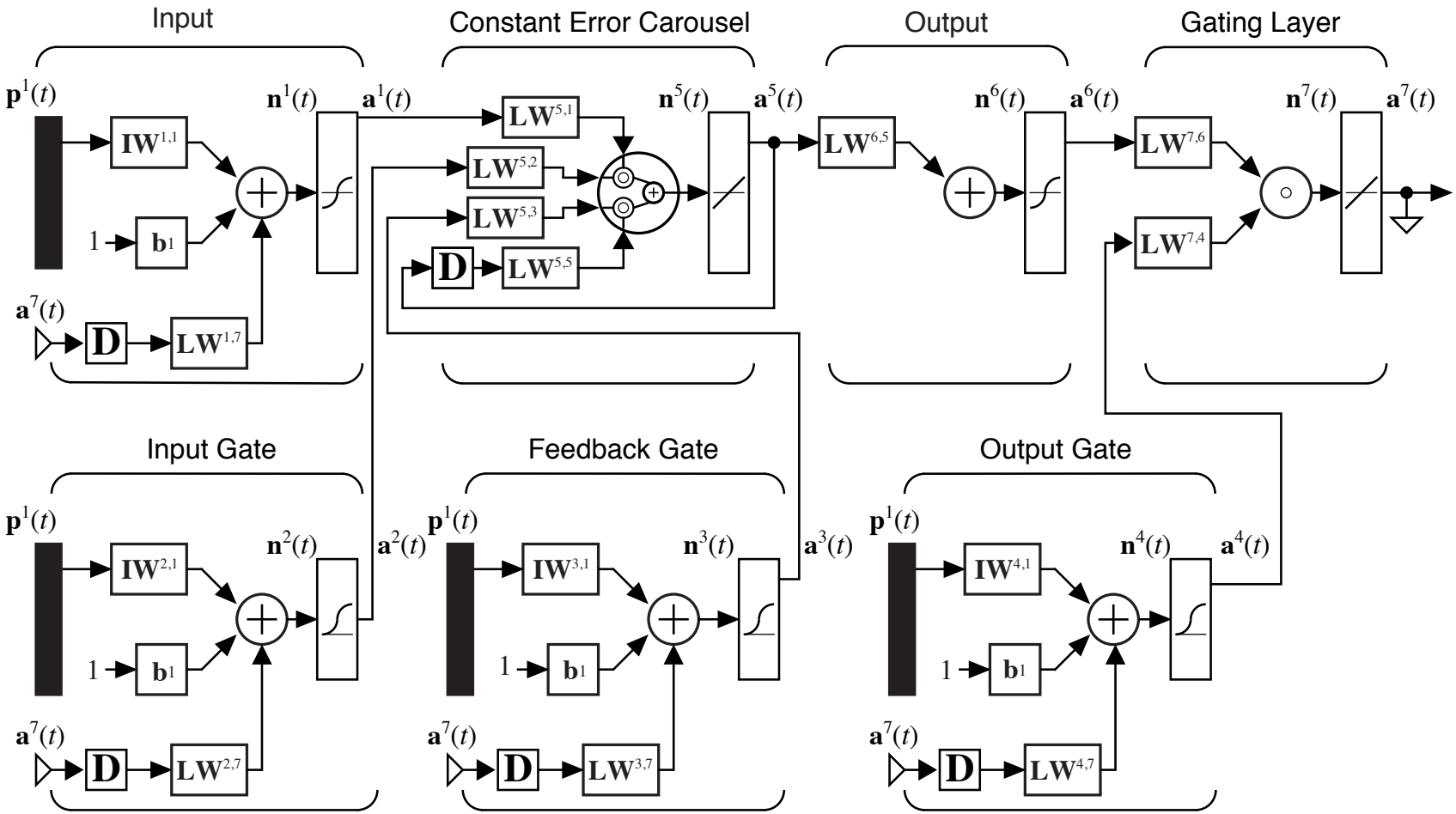
- To maintain a long memory, we would like the feedback matrix $\mathbf{LW}^{1,1}$ to have some eigenvalues very close to one.
- This has to be maintained during training, or the gradients will vanish.
- In addition, to ensure long memories, the derivative of the transfer function should be constant.
- Eigenvalues greater than one $\rightarrow$ unstable.
- Solution: Set $\mathbf{LW}^{1,1} = \mathbf{I}$ and use linear transfer function.



Gating

- We don't want to indiscriminately remember everything.
- To remember selectively, we introduce several gates (switches).
  - An input gate will allow selective inputs into the CEC.
  - A feedback (or forget) gate will clear the CEC.
  - An output gate will allow selective outputs from the CEC.
- Each gate will be a layer with inputs from the gated ouputs and the network inputs.
- The result is called Long Short Term Memory (LSTM).
- With the CEC, short term memories last longer.

Long Short Term Memory Network



LSTM Details

- The $\circ$ operator is the Hadamard product, which is an element by element matrix multiplication.
- The weights in the CEC are all fixed to the identity matrix. They are not trained.
- The output and gating layer weights are also fixed to the identity matrix.
- It has been shown that the best results are obtained when initializing the feedback (forget) gate bias, $\mathbf{b}^3$, to all ones or larger values. This turns the gate on initially.
- Other weights and biases are set to small random numbers.
- The output of the gating layer generally connects to another layer or a multilayer network with softmax transfer function.
- Multiple LSTMs can be cascaded together.

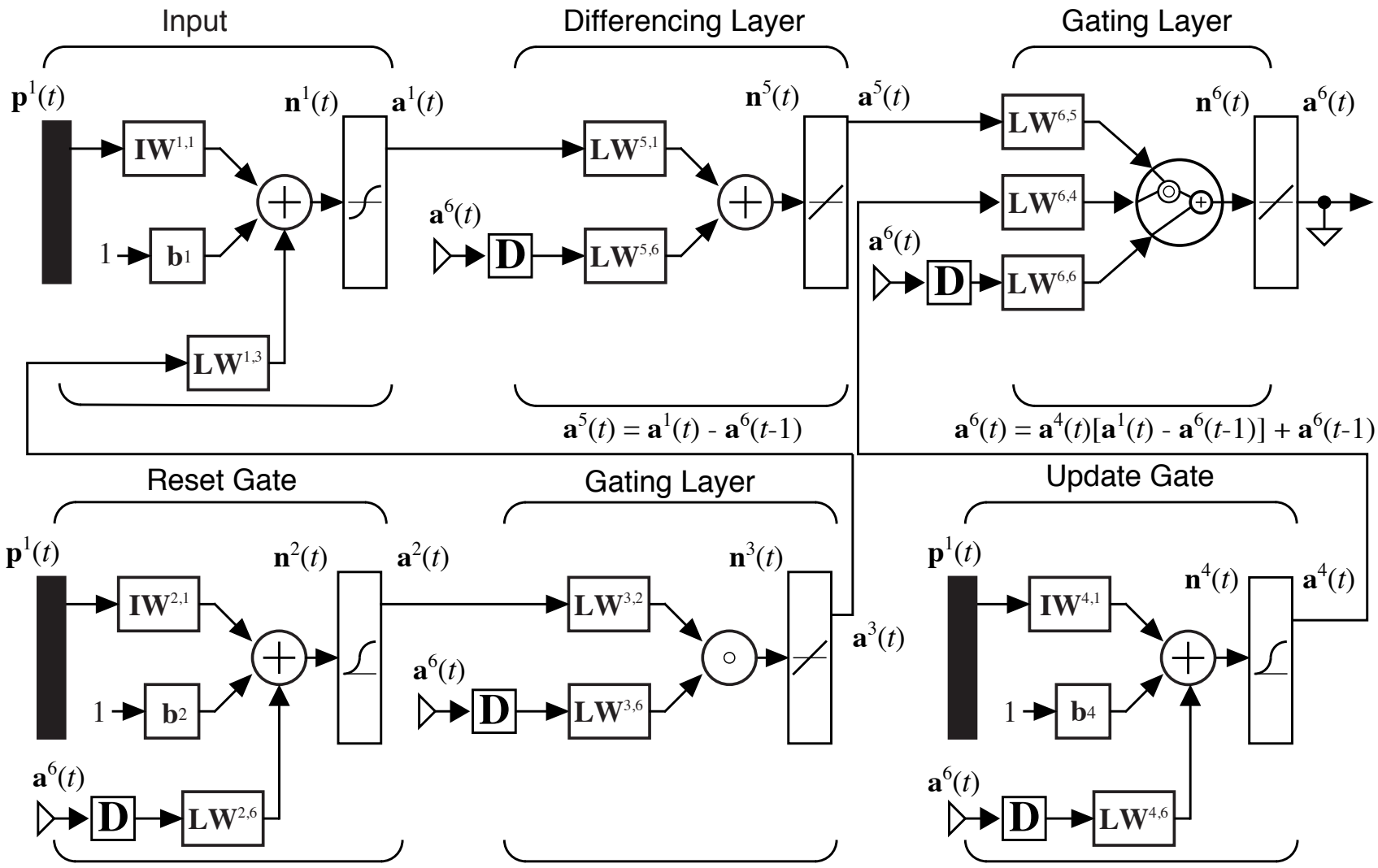
LSTM Training

- LSTM is trained with the standard gradient-based algorithms as with other deep networks.
- The original LSTM paper used RTRL for computing the gradient.
- An approximate gradient is used, in which the derivatives are only propagated in time through the CEC delays. Only static derivatives are computed for the remaining terms.

Variations of the LSTM

- There have been many variations of the LSTM
- In the original LSTM, there was no feedback (forget) gate.
- Some variations feeback the CEC output $\mathbf{a}^5(t)$ to the gate layers. These are called "peephole" connections.
- See http://jmlr.org/proceedings/papers/v37/jozefowicz15.pdf for an experimental comparison of various alternatives.
- One of the popular alternatives is the Gated Recurrent Unit (GRU), shown on the following page.

GRU Network


Input
Differencing Layer
Gating Layer
Reset Gate
Gating Layer
Update Gate
$\mathbf{p}^1(t)$
$\mathbf{n}^1(t)$
$\mathbf{a}^1(t)$
$\mathbf{n}^5(t)$
$\mathbf{a}^5(t)$
$\mathbf{n}^6(t)$
$\mathbf{a}^6(t)$
$\mathbf{n}^2(t)$
$\mathbf{a}^2(t)$
$\mathbf{n}^3(t)$
$\mathbf{a}^3(t)$
$\mathbf{n}^4(t)$
$\mathbf{a}^4(t)$
IW1,1
b1
LW1,3
LW5,1
LW5,6
LW6,5
LW6,4
LW6,6
IW2,1
b2
LW2,6
LW3,2
LW3,6
IW4,1
b4
LW4,6
D
1
$\mathbf{a}^5(t) = \mathbf{a}^1(t) - \mathbf{a}^6(t\text{-}1)$
$\mathbf{a}^6(t) = \mathbf{a}^4(t)[\mathbf{a}^1(t) - \mathbf{a}^6(t\text{-}1)] + \mathbf{a}^6(t\text{-}1)$

- The differencing layer subtracts the delayed network output $\mathbf{a}^6(t-1)$ from the processed input $\mathbf{a}^1(t)$.
- $\mathbf{LW}^{5,1} = \mathbf{I}$ and $\mathbf{LW}^{5,6} = -\mathbf{I}$. They are not trained.
- The gating layer weights are fixed to the identity matrix.
- The delayed network output is gated by the reset gate, before it is fed into the input layer. This resets the memory.
- The overall output can be considered a weighted average of a candidate potential output $\mathbf{a}^1(t)$ and the previous output $\mathbf{a}^6(t-1)$.

$$\begin{aligned}\mathbf{a}^6(t) &= \mathbf{a}^4(t) \times [\mathbf{a}^1(t) - \mathbf{a}^6(t-1)] + \mathbf{a}^6(t-1) \\ &= \mathbf{a}^4(t) \times \mathbf{a}^1(t) + [1 - \mathbf{a}^4(t)] \times \mathbf{a}^6(t-1)\end{aligned}$$