Chapter 8: Dynamically Driven Recurrent Networks

Neural Networks CSC372

Kiran Bagale

St. Xavier's College

September, 2025

Outline

Introduction

Key Statement

Global feedback is a facilitator of computational intelligence.

- This fundamental principle guides our understanding of recurrent networks
- Well illustrated through the study of recurrent networks as **associative memories**
- Global feedback in recurrent networks enables several useful computational tasks:

1. **Content-addressable memory**
   - Exemplified by the Hopfield network
2. **Autoassociation**
   - Exemplified by Anderson's brain-state-in-a-box model
3. **Dynamic reconstruction of chaotic processes**
   - Using feedback built around a regularized one-step predictor

RECURRENT NETWORK ARCHITECTURES

Definition

Dynamically driven recurrent networks respond *temporally* to externally applied input signals.

- They all incorporate a static multilayer perceptron or parts thereof.
- They all exploit the nonlinear mapping capability of the multilayer perceptron.

Input–Output Recurrent Model

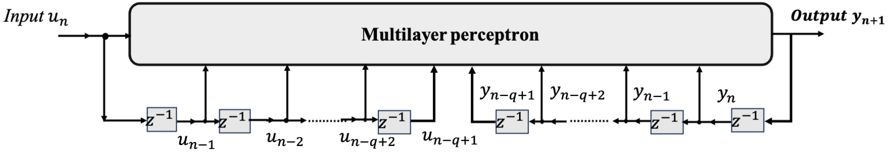


FIGURE 8.1 Nonlinear autoregressive with exogenous inputs (NARX) model; the feedback part of the network is shown in red.

State-Space Representation

The notion of **state** plays a vital role in the mathematical formulation of a dynamic system.

- State: A set of quantities that summarize all the information about the past behavior of the system, needed to describe its future behavior.
- Let $\mathbf{x}_n$ = state vector $(q \times 1)$, $\mathbf{u}_n$ = input vector $(m \times 1)$, $\mathbf{y}_n$ = output vector $(p \times 1)$.

The recurrent system is described by:

$$\mathbf{x}_{n+1} = \phi\left(W_a \mathbf{x}_n + W_b \mathbf{u}_n\right)$$

$$\mathbf{y}_n = W_c \mathbf{x}_n$$

where W_a, W_b, W_c are weight matrices and $\phi(\cdot)$ is a nonlinear activation function.

State-Space Model

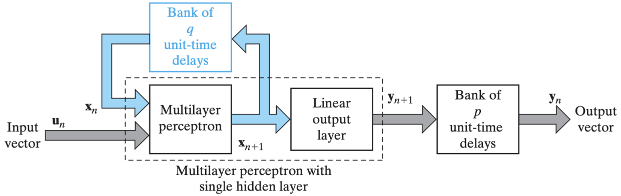


Figure: State-space model; the feedback part of the model is shown in Blue.

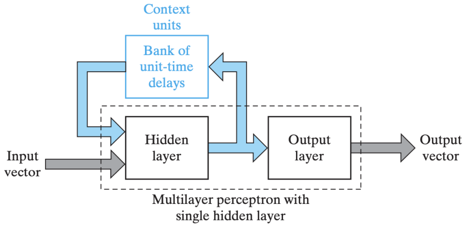


Figure: Simple recurrent network (SRN); the feedback part of the network is

Recurrent Multilayer Perceptron (RMLP)

- Extension of static multilayer perceptrons with feedback connections
- Each computational layer has feedback loops (unit-time delays)
- More effective and parsimonious than single hidden layer networks

Key Features

- Multiple hidden layers with recurrent connections
- Subsumes both Elman networks and state-space models
- No constraints on activation functions

RMLP Architecture

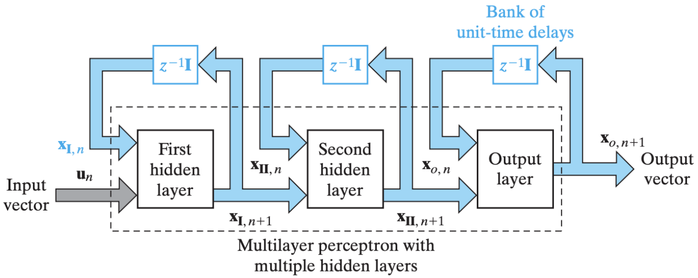


Figure: Recurrent multilayer perceptron with feedback paths (shown in Blue)

- Input vector: $\mathbf{u}_n$
- Hidden layer outputs: $\mathbf{x}_{I,n}, \mathbf{x}_{II,n}, \dots$
- Final output: $\mathbf{x}_{o,n+1}$
- Unit-time delays: z^{-1} blocks

RMLP Mathematical Formulation

The dynamic behavior is described by a system of coupled equations:

$$\mathbf{x}_{I,n+1} = \phi_I(\mathbf{x}_{I,n}, \mathbf{u}_n) \tag{1}$$
$$\mathbf{x}_{II,n+1} = \phi_{II}(\mathbf{x}_{II,n}, \mathbf{x}_{I,n+1}) \tag{2}$$
$$\vdots \tag{3}$$
$$\mathbf{x}_{o,n+1} = \phi_o(\mathbf{x}_{o,n}, \mathbf{x}_{K,n+1}) \tag{4}$$

- $\phi_I(\cdot,\cdot), \phi_{II}(\cdot,\cdot), \ldots, \phi_o(\cdot,\cdot)$: activation functions
- K: number of hidden layers (K=2 in the example)
- No specific constraints on activation function forms

Second-Order Recurrent Network Architecture

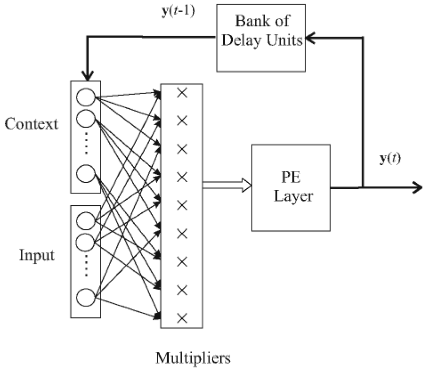


Figure: Second-order recurrent network with multipliers and unit-time delays

- Network with 2 inputs and 3 state neurons
- Requires $3 \times 2 = 6$ multipliers
- Feedback links shown in red emphasize global role
- Bias connections omitted for clarity

Second-Order Network Dynamics

The network dynamics are governed by:

$$v_{k,n} = b_k + \sum_i \sum_j w_{kij} x_{i,n} u_{j,n} \tag{5}$$

$$x_{k,n+1} = \varphi(v_{k,n}) = \frac{1}{1 + \exp(-v_{k,n})} \tag{6}$$

Where:

- $v_{k,n}$: induced local field of neuron k
- b_k: bias term for neuron k
- $x_{k,n}$: state (output) of neuron k at time n
- $u_{j,n}$: input applied to source node j
- w_{kij}: second-order weight

State Transitions and Applications

State Transition Interpretation

Product $x_i u_j$ represents the pair {state, input}

- Positive weight w_{kij}: presence of transition {state, input} $\rightarrow$ {next state}
- Negative weight: absence of the transition

State Transition Function

$$\delta(x_i, u_j) = x_k \tag{7}$$

Applications

Second-order networks are readily used for:

- Representing and learning **deterministic finite-state automata (DFA)**
- Information-processing systems with finite number of states

First-Order vs Second-Order Neurons

First-Order Neuron

Induced local field combines inputs **additively**:

$$v_k = \sum_j w_{a,kj} x_j + \sum_i w_{b,ki} u_i \tag{8}$$

Second-Order Neuron

Induced local field combines inputs **multiplicatively**:

$$v_k = \sum_i \sum_j w_{kij} x_i u_j \tag{9}$$

- x_j: feedback signal from hidden neuron j
- u_i: source signal from input node i
- w_{kij}: single weight connecting neuron k to input nodes i and j

Rich Architectural Repertoire

- Recurrent networks have a **very rich repertoire** of architectural layouts
- Multiple possible feedback configurations
- Various neural network configurations as building blocks
- This diversity makes them **powerful in computational terms**

Key Insight

The architectural flexibility of recurrent networks contributes significantly to their computational power.

Temporal Response Characteristics

Input-Output Mapping

The input space of a mapping network is mapped onto an output space.

Diverse Applications

The ability to acquire state representations makes dynamically driven recurrent networks suitable for:

- **Nonlinear prediction and modeling**
- **Adaptive equalization of communication channels**
- **Speech processing**
- **Plant control**
- Many other applications...

Common Thread

All these applications benefit from the temporal processing capabilities and state representation features of recurrent networks.

Recurrent Neural Network State-Space Model

- The **state** of a dynamic system summarizes all information about past behavior needed to uniquely describe future behavior
- For a nonlinear discrete-time system with:
  - $\mathbf{x}_n$: q-by-1 state vector
  - $\mathbf{u}_n$: m-by-1 input vector
  - $\mathbf{y}_n$: p-by-1 output vector

State-Space Equations

$$\mathbf{x}_{n+1} = \phi(\mathbf{W}_a\mathbf{x}_n + \mathbf{W}_b\mathbf{u}_n) \tag{10}$$

$$\mathbf{y}_n = \mathbf{W}_c\mathbf{x}_n \tag{11}$$

Network Architecture Components

- $\mathbf{W}_a$: q-by-q matrix representing synaptic weights of hidden neurons connected to feedback nodes
- $\mathbf{W}_b$: q-by-m matrix representing synaptic weights of hidden neurons connected to input nodes
- $\mathbf{W}_c$: p-by-q matrix representing synaptic weights of output neurons connected to hidden neurons
- $\phi : \mathbb{R}^q \to \mathbb{R}^q$: diagonal nonlinear activation function

Diagonal Activation Function

$$\phi : \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_q \end{bmatrix} \to \begin{bmatrix} \varphi(x_1) \\ \varphi(x_2) \\ \vdots \\ \varphi(x_q) \end{bmatrix}$$

Activation Functions

Common choices for the componentwise nonlinearity $\varphi(\cdot)$:

Hyperbolic Tangent

$$\varphi(x) = \tanh(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}}$$

Logistic Function

$$\varphi(x) = \frac{1}{1 + e^{-x}}$$

Key Property: Only neurons in the hidden layer that feed back their outputs to the input layer via delays define the state of the recurrent network.

Universal Approximation Theorem

Main Result (Lo, 1993)

Any nonlinear dynamic system may be approximated by a recurrent neural network to any desired degree of accuracy and with no restrictions imposed on the compactness of the state space, provided that the network is equipped with an adequate number of hidden neurons.

- This theorem demonstrates the **computing power** of recurrent neural networks
- Applications in signal processing and control systems
- No limitations on state space compactness
- Only requirement: sufficient number of hidden neurons

Example: Fully Connected Recurrent Network

Consider a network with $m = 2$, $q = 3$, and $p = 1$:

Weight Matrices

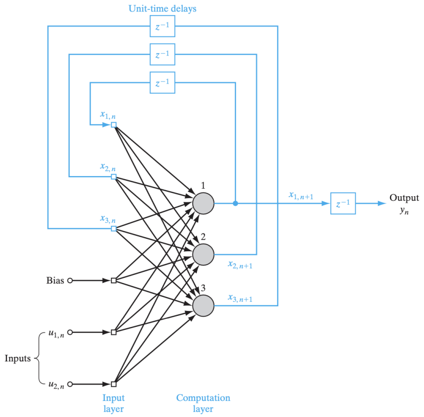
$$\mathbf{W}_a = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{bmatrix}$$

$$\mathbf{W}_b = \begin{bmatrix} b_1 & w_{14} & w_{15} \\ b_2 & w_{24} & w_{25} \\ b_3 & w_{34} & w_{35} \end{bmatrix}$$

$$\mathbf{W}_c = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}$$

- First column of $\mathbf{W}_b$ contains bias terms
- Global feedback connections shown in network diagram
- Unit-time delays implement memory
- Single output from first hidden neuron

Network Architecture Diagram



- Blue connections emphasize global feedback role
- Unit-time delays (z^{-1}) provide memory capability
- Input layer processes current inputs
- Computation layer implements nonlinear dynamics
- Feedback connections define system state

Controllability and Observability: Key Concepts

Motivation

For recurrent networks with state-space representation, we need to understand:

- Can we **control** the dynamic behavior?
- Can we **observe** the results of our control?

Definition (Controllability)

A dynamic system is **controllable** if any initial state can be steered to any desired state within a finite number of time-steps.

Definition (Observability)

A system is **observable** if the state can be determined from a finite set of input-output measurements.

Local Controllability and Observability

Local Analysis Approach

We focus on **local** controllability and observability around an equilibrium state:

- Both properties apply in the neighborhood of equilibrium

Equilibrium State Condition

A state $\mathbf{x}$ is an **equilibrium state** if:

$$\mathbf{x} = \mathbf{A}_1\mathbf{x} \tag{12}$$

For simplification, the equilibrium point is at the origin: $(\mathbf{0}, \mathbf{0})$

$$\mathbf{0} = \phi(\mathbf{0}) \quad \text{for } \mathbf{x} = \mathbf{0} \tag{13}$$

Single-Input Single-Output (SISO) System

Simplified System Equations

For a SISO system, the recurrent network equations become:

$$\mathbf{x}_{n+1} = \phi(\mathbf{W}_v \mathbf{x}_n + \mathbf{w}_b u_n) \tag{14}$$

$$y_n = \mathbf{w}_c^T \mathbf{x}_n \tag{15}$$

Linearization Process

Since ϕ is continuously differentiable (sigmoid function), we can linearize around equilibrium:

$$\delta \mathbf{x}_{n+1} = \mathbf{\Phi}(\mathbf{0}) \mathbf{W}_v \delta \mathbf{x}_n + \mathbf{\Phi}(\mathbf{0}) \mathbf{w}_b \delta u_n \tag{16}$$

Leading to the linearized system:

$$\delta \mathbf{x}_{n+1} = \mathbf{A}_1 \delta \mathbf{x}_n + \mathbf{a}_2 \delta u_n \tag{17a}$$

$$\delta y_n = \mathbf{w}_c^T \delta \mathbf{x}_n \tag{17b}$$

System Matrices

Key Matrix Definitions

The linearized system matrices are defined as:

$$\mathbf{A}_1 = \mathbf{\Phi}(\mathbf{0})\mathbf{W}_v \tag{18}$$

$$\mathbf{a}_2 = \mathbf{\Phi}(\mathbf{0})\mathbf{w}_b \tag{19}$$

Standard Linear Form

The state equations (17a) and (17b) are now in **standard linear form**, allowing us to use well-established results from mathematical control theory.

Connection to Classical Control

This linearization bridges recurrent neural networks with classical linear systems theory, enabling the application of:

- Controllability analysis tools
- Observability analysis tools and Linear control design methods

Local Observability Theory

Input-Output Sequence

Consider a sequence of inputs and outputs:

$$\mathbf{u}_{q-1,n} = [u_n, u_{n+1}, \ldots, u_{n+q-2}]^T \tag{20}$$

$$\mathbf{y}_{q,n} = [y_n, y_{n+1}, \ldots, y_{n+q-1}]^T \tag{21}$$

Nonlinear Mapping

The system defines a mapping:

$$\mathbf{H}(\mathbf{u}_{q-1,n}, \mathbf{x}_n) = (\mathbf{u}_{q-1,n}, \mathbf{y}_{q,n}) \tag{22}$$

where $\mathbf{H} : \mathbb{R}^{2q-1} \to \mathbb{R}^{2q-1}$

Jacobian Analysis

The Jacobian of $\mathbf{y}_{q,n}$ with respect to $\mathbf{x}_n$ at the origin equals the observability matrix $\mathbf{M}_o$.

Observability Matrix Structure

Jacobian Matrix Form

The Jacobian of $\mathbf{H}$ at the origin $(0,0)$ has the structure:

$$\mathbf{J}^{(c)}_{(0,0)} = \begin{bmatrix} \left(\frac{\partial \mathbf{u}_{q-1,n}}{\partial \mathbf{u}_{q-1,n}}\right)_{(0,0)} & \left(\frac{\partial \mathbf{y}_{q,n}}{\partial \mathbf{u}_{q-1,n}}\right)_{(0,0)} \\ \left(\frac{\partial \mathbf{u}_{q-1,n}}{\partial \mathbf{x}_{n}}\right)_{(0,0)} & \left(\frac{\partial \mathbf{y}_{q,n}}{\partial \mathbf{x}_{n}}\right)_{(0,0)} \end{bmatrix} \quad (23)$$

$$= \begin{bmatrix} \mathbf{I} & \mathbf{X} \\ \mathbf{0} & \mathbf{M}_o \end{bmatrix} \quad (24)$$

Key Result

The determinant of $\mathbf{J}^{(c)}_{(0,0)}$ equals the product of:

- Determinant of identity matrix $\mathbf{I}$ (equals 1)
- Determinant of observability matrix $\mathbf{M}_o$

If $\mathbf{M}_o$ is full rank, then $\mathbf{J}^{(c)}_{(0,0)}$ is full rank.

Local Observability Theorem

Inverse Function Theorem Application

If the observability matrix $\mathbf{M}_o$ is full rank, then locally there exists an inverse mapping:

$$(\mathbf{u}_{q-1,n}, \mathbf{x}_n) = \mathbf{H}^{-1}(\mathbf{u}_{q-1,n}, \mathbf{y}_{q,n}) \tag{25}$$

Theorem (Local Observability Theorem (Levin and Narendra, 1993))

Let a recurrent network be defined by Eqs. (14) and (15), and let its linearized version around the origin be defined by Eqs. (17a) and (17b). **If the linearized system is observable, then the recurrent network is locally observable around the origin.**

Practical Implication

In the local neighborhood of the origin, $\mathbf{x}_n$ is some nonlinear function of both $\mathbf{u}_{q-1,n}$ and $\mathbf{y}_{q,n}$, providing an observer for the recurrent network.

Local Controllability Theory

Inverse Function Theorem for Control

Consider the mapping $\mathbf{f} : \mathcal{U} \to \mathcal{V}$. If:

1. $\mathbf{f}(\mathcal{U}) = \mathcal{V}$
2. The mapping $\mathbf{f} : \mathcal{U} \to \mathcal{V}$ is one-to-one (invertible)
3. Each component of $\mathbf{f}^{-1} : \mathcal{V} \to \mathcal{U}$ is continuously differentiable

Then $\mathbf{f}$ is a **smooth diffeomorphism**.

Controllability Application

If the controllability matrix $\mathbf{M}_c$ is of rank q, then locally there exists an inverse mapping:

$$(\mathbf{x}_n, \mathbf{x}_{n+q}) = \mathbf{G}^{-1}(\mathbf{x}_n, \mathbf{u}_{q,n}) \tag{26}$$

This means there exists an input sequence $(\mathbf{u}_{q,n})$ that can drive the network from state $\mathbf{x}_n$ to state $\mathbf{x}_{n+q}$ in q time-steps.

Local Controllability Theorem

Controllability Matrix Construction

Using the linearized equation repeatedly:

$$\delta \mathbf{x}_{n+1} = \mathbf{A}_1 \delta \mathbf{x}_n + \mathbf{a}_2 \delta u_n$$
$$\delta \mathbf{x}_{n+2} = \mathbf{A}_1^2 \delta \mathbf{x}_n + \mathbf{A}_1 \mathbf{a}_2 \delta u_n + \mathbf{a}_2 \delta u_{n+1}$$
$$\vdots$$
$$\delta \mathbf{x}_{n+q} = \mathbf{A}_1^q \delta \mathbf{x}_n + \mathbf{A}_1^{q-1} \mathbf{a}_2 \delta u_n + \cdots + \mathbf{a}_2 \delta u_{n+q-1}$$

System Analysis Matrices

Controllability Matrix

The linearized system is controllable if the matrix:

$$\mathbf{M}_c = [\mathbf{A}_1^{q-1}\mathbf{a}_2, \dots, \mathbf{A}_1\mathbf{a}_2, \mathbf{a}_2] \tag{27}$$

is of rank q (full rank).

Observability Matrix

The linearized system is observable if the matrix:

$$\mathbf{M}_o = [\mathbf{w}_c, \mathbf{w}_c\mathbf{A}_1^T, \dots, \mathbf{w}_c(\mathbf{A}_1^T)^{q-1}] \tag{28}$$

is of rank q (full rank).

Rank Condition

Both controllability and observability require their respective matrices to have **full rank** for the properties to hold locally around the equilibrium.

Example: Simple State-Space Model Analysis

Example (Controllability and Observability Analysis)

Consider a state-space model with matrix $\mathbf{A}_1 = a\mathbf{I}$, where a is a scalar and $\mathbf{I}$ is the identity matrix.

Controllability Analysis

The controllability matrix reduces to:

$$\mathbf{M}_c = a[\mathbf{a}_2, \ldots, \mathbf{a}_2, \mathbf{a}_2] \tag{29}$$

- The rank of this matrix is 1
- Hence, the linearized system with this value of $\mathbf{A}_1$ is **not controllable**

Observability Analysis

Similarly, putting $\mathbf{A}_1 = a\mathbf{I}$ in the observability matrix:

$$\mathbf{M}_o = a[\mathbf{w}_c, \mathbf{w}_c, \ldots, \mathbf{w}_c] \tag{30}$$

Summary: Controllability and Observability of RNNs

Key Theoretical Results

1. **Linearization Approach**: Analysis around equilibrium points using Taylor series expansion
2. **Local Controllability Theorem**: Full-rank controllability matrix $\Rightarrow$ local controllability
3. **Local Observability Theorem**: Full-rank observability matrix $\Rightarrow$ local observability
4. **Classical Control Connection**: Standard linear system analysis applies to linearized RNNs

Practical Implications

- Design guidelines for RNN architectures
- Understanding limitations of simple models (e.g., $\mathbf{A}_1 = a\mathbf{I}$)
- Foundation for RNN-based control system design
- Observer design for state estimation in RNNs

Computational Power of Recurrent Networks

- Recurrent networks can simulate finite-state automata
- Automata represent abstractions of information-processing systems
- Neural networks and automata share a long history
- Early work used hard threshold logic instead of soft sigmoid functions

Historical Context

Minsky (1967): *"Every finite-state machine is equivalent to, and can be 'simulated' by, some neural net. That is, given any finite-state machine $\mathcal{M}$, we can build a certain neural net $\mathcal{N}$ which, regarded as a black-box machine, will behave precisely like $\mathcal{M}$."*

Early Experimental Work

- **Cleeremans et al. (1989)**: First experimental demonstration
- Simple recurrent network learned contingencies of finite-state grammar
- Task: Predict next letter in context-dependent strings
- Network developed internal representations corresponding to automaton states

Key Finding

The network was able to develop internal representations in its hidden neurons that correspond to the states of the finite-state machine.

Kremer (1995): Formally proved that simple recurrent networks have computational power equal to any finite-state machine.

Fundamental Theorems

Theorem (Siegelmann and Sontag, 1991)

All Turing machines may be simulated by fully connected recurrent networks built on neurons with sigmoidal activation functions.

Turing Machine Components

A Turing machine consists of three functional blocks:

1. A **control unit** with a finite number of possible states
2. A **linear tape**, infinitely long in both directions, divided into discrete squares for storing symbols
3. A **read-write head** that moves along the tape and transmits information

NARX Networks and Enhanced Results

Theorem (Siegelmann et al., 1997)

NARX networks with one layer of hidden neurons with bounded, one-sided saturated activation functions and a linear output neuron can simulate fully connected recurrent networks with bounded, one-sided saturated activation functions, except for a linear slowdown.

BOSS Functions

A function $\varphi(\cdot)$ is **bounded, one-sided saturated** (BOSS) if:

1. Bounded range: $a \leq \varphi(x) \leq b$, $a \neq b$
2. Left-side saturated: $\varphi(x) = S$ for all $x \leq s$
3. Nonconstant: $\varphi(x_1) \neq \varphi(x_2)$ for some x_1, x_2

Learning Algorithms

Training Modes for Recurrent Networks

1. Epochwise Training

- Uses temporal sequence of input-target pairs
- Runs from initial state to new state
- Network reset between epochs
- Good for emulating finite-state machines

2. Continuous Training

- No reset states available
- Network learns while performing
- Suitable for nonstationary processes
- Example: speech signal modeling

Learning Algorithms

- **BPTT** (Back-Propagation Through Time): Better for off-line training
- **RTRL** (Real-Time Recurrent Learning): More suitable for on-line training

Training Heuristics for RNNs

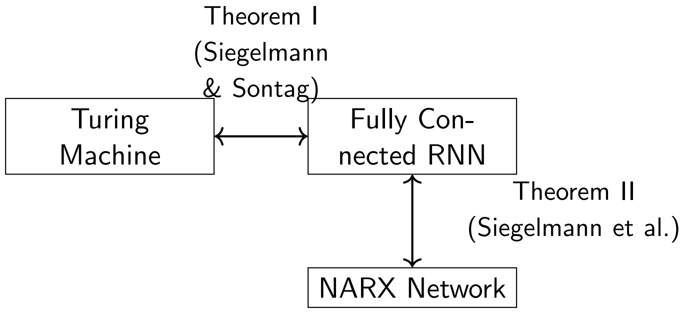
Recommended Heuristics (Giles, 1996)

1. **Lexicographic order**: Present shortest symbol strings first
2. **Incremental training**: Start with small samples, gradually increase size
3. **Conditional updates**: Update weights only if absolute error exceeds threshold
4. **Weight decay**: Use complexity regularization during training

Vanishing Gradients

The first heuristic is particularly important as it may help alleviate the vanishing gradients problem in recurrent networks trained with gradient-descent methods.

Computational Equivalence Summary



Key Insight

Recurrent networks possess universal computational power, making them theoretically capable of learning any computable function, though practical limitations may apply.

Thank You!

Questions?