ARTIFICIAL INTELLIGENCE AND NEURAL NETWORKS

Lecture 6b – Recurrent Neural Networks and Sequence Modeling

Chizhi Chris ZHANG

zhangchizhi@ciomp.ac.cn
Advanced Computing and Digital Technology Research Center
University of Chinese Academy of Sciences

Spring 2026

Today's Question

What we are trying to answer

How should a neural network handle information that arrives one step after another, when earlier words or signals can change the meaning of what comes next?

Why this lecture matters

If we only say that sequence models read things in order, the topic stays vague. We need to understand what memory problem they are trying to solve.

What changes from NN5

NN5 focused on spatial structure in images. NN6 shifts from space to order: the challenge is no longer location in an image, but information moving through time.

From NN5 to NN6

Last time

We saw how CNNs respect the geometry of images by using local filters and shared weights.

Today

We ask what a network should do when data arrives as a sequence, where order and memory matter.

One sentence

NN5 asked how to see structure in space. NN6 asks how to carry structure through time.

Why Order Matters

A simple fact

"dog bites man" and "man bites dog" contain the same words, but not the same meaning.

The broader lesson

Speech, music, finance, sensor streams, and language all depend on order. The same element can mean something different depending on what came before.

Once order matters, one static snapshot is no longer enough.

What Counts as a Sequence Task

Several common patterns

- Read many steps, then output one answer
- Output something at every step
- Read a sequence and then generate another sequence

Examples

Sentiment analysis, speech tagging, translation, captioning, next-word prediction, and time-series forecasting all fit somewhere in this family.

Why Static Networks Struggle

The weakness

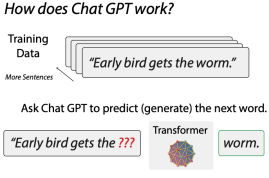
A plain feedforward network treats the input as one fixed package. It does not naturally maintain a running memory of what happened several steps ago.

Why sequence tasks ask for more

The model needs to update its internal summary as new pieces arrive instead of starting from scratch at every step.

This is why sequence modeling became its own major topic rather than a small extension of ordinary networks.

Why Modern AI Cares So Much



Generative

We are asking Chat GPT to generate the next word!

Pretrained

Trained on a simple task, but can be applied to many others!

Transformer

The ML model that guesses the word.

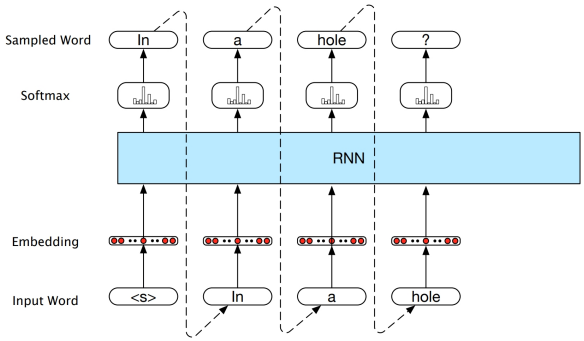
Why this picture belongs here

Modern language systems work only because sequence modeling became strong enough to connect a token with many earlier tokens.

Why this lecture still begins with RNNs

Even though transformers dominate today, the core memory questions became visible much earlier in recurrent models.

The Core Idea of an RNN



RNN idea

Read one step, update an internal state, then use that updated state when reading the next step.

Why this matters

The hidden state acts like a running memory of what the network has seen so far.

The Smallest Useful Formula Set

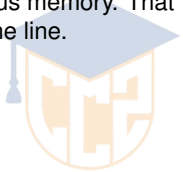
Minimal recurrence

$$h_t = f(W_x x_t + W_h h_{t-1} + b)$$

Output step

$$y_t = g(W_y h_t)$$

Read it plainly: the new memory h_t depends on the current input and the previous memory. That is the whole recurrent idea in one line.



Reading a Sequence One Step at a Time

What repeats at each moment

Take the current input, combine it with the previous hidden state, and produce a new hidden state that carries the story forward.

Why this is elegant

The same small computation repeats across the whole sequence, so one model can process sequences of different lengths.

This is the neural version of reading one word after another while updating your understanding as you go.

Three Output Styles

Several common patterns

- One output at the end for tasks like classification
- One output at each step for tasks like tagging
- A new sequence as output for tasks like translation or generation

Why this helps

It shows that recurrence is not one narrow trick. It is a general way to process ordered information.

Why Words Must Become Vectors

Word Coordinates

	Gender	Age
man	1	7
woman	9	7
boy	1	2
girl	9	2

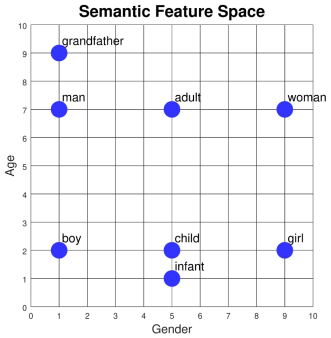
Why symbols are not enough

The network cannot compute directly on plain word names. It needs numeric representations that can be compared, combined, and transformed.

Embedding intuition

Each word is mapped into a vector so the model can work with meaning in a continuous space.

From Word Vectors to Meaning



Word Coordinates

	Gender	Age
grandfather	1	9
man	1	7
adult	5	7
woman	9	7
boy	1	2
child	5	2
girl	9	2
infant	5	1

Why embeddings matter

Words with related roles or meanings can end up near one another, so the model can reuse what it learns across similar terms.

What the RNN is remembering

It is not only remembering raw word identities. It is carrying forward transformed semantic signals.

Unrolling Through Time

What unrolling means

We draw the same recurrent cell again and again across time steps so we can see the full chain of influence.

Why this matters

The picture reveals how one early piece of information has to survive through many updates if it is going to matter later.

The weights are shared across moments, but the hidden state keeps changing.

A Prediction Story

A language example

The model reads words one by one and uses the evolving hidden state to guess what kind of word is likely to come next.

Why this page matters

Meaning does not live in one token. It lives in the changing state built across many tokens.

That is why sequence models are really about memory, not just about reading in order.

What the Training Loop Really Does

Three steps

- Move forward through the sequence and compute hidden states
- Compare outputs with targets and measure the error
- Send correction signals backward through the unrolled chain

Why this sounds harder than CNNs

The model is not only learning across layers. It is also learning across many time steps.

What the State Should Remember

The hidden challenge

The state has to keep useful earlier information while letting unimportant detail fade away.

Why this is difficult

Too much forgetting destroys meaning. Too much memory fills the state with noise.

The central design problem

Good sequence models are really good memory managers.

The Long Memory Problem

Short dependencies are easier

If the answer depends mostly on the last few steps, a basic RNN may manage well enough.

Long dependencies are harder

If meaning depends on something far back in the sequence, the information can fade before it becomes useful.

This is why sequence modeling quickly became a question about memory range, not only about recurrence.

Why Gradients Vanish

A simple picture

The training signal has to pass through many repeated steps. If those repeated multiplications shrink the signal, early steps receive almost no useful correction.

Why students should care

This is the reason a model can in principle see the whole past but still fail to learn the important long-range dependency.

Why RNN Training Felt Fragile

The practical problem

Long chains, repeated reuse of the same weights, and heavy pressure on the hidden state made optimization more delicate than in many feedforward settings.

What the field learned

Plain recurrence was an important idea, but it was not yet a comfortable answer for long and complicated sequences.

This practical frustration is what pushed the field toward gated models and later toward attention.

Where RNN-Style Models Worked Well

Typical strengths

Speech, early language modeling, tagging, sequence labeling, and time-series forecasting all benefited from recurrent thinking.

Why that makes sense

These are exactly the settings where order is central and where each new step should update a running interpretation.

Why LSTM Was Invented

The motivation

Plain RNNs often forgot useful earlier information too quickly.

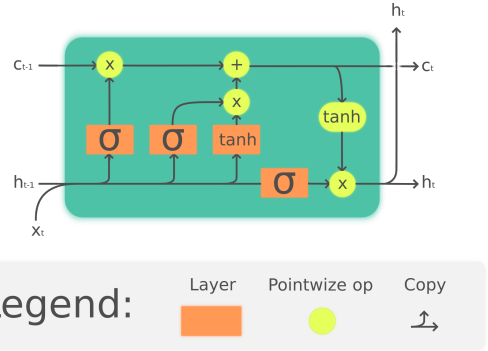
The LSTM answer

Give the network more deliberate control over what to keep, what to write, and what to expose from memory.

The teaching sentence

LSTM was a memory-management upgrade, not a completely different species of network.

Inside an LSTM



What to notice

There is a more protected path for longer-term information, plus gates that control updates and exposure.

What not to do

Do not panic over every symbol. First understand the purpose: more careful control of memory flow.

What the Gates Mean

Forget gate

Decide what older information can fade away.

Input gate

Decide what new information is worth writing into memory.

Output gate

Decide what part of the memory should influence the current visible state.

GRU as a Simpler Memory Fix

GRU idea

Keep the same general goal as LSTM, but use a simpler gating structure.

Why that mattered

Sometimes a slightly simpler memory mechanism is easier to train or easier to deploy while still handling long-range information better than a plain RNN.

Once the field recognized memory control as the bottleneck, several related architectures tried to improve it.

A Common Misunderstanding

The myth

"If an RNN has access to the full past, it automatically remembers everything important."

The better statement

An RNN is exposed to the past, but whether it can preserve the right information strongly enough is a different question.

Exposure to information is not the same as effective memory.

Why Recurrence Still Had Limits

Sequential bottleneck

RNNs process one step after another, which makes long sequences harder to parallelize and harder to connect across distant positions.

Bigger issue

Even gated memory improves the situation without completely removing the difficulty of very long dependencies.

The field needed a new way to let distant tokens influence each other more directly.

Why Transformers Took Over

The main shift

Instead of forcing information to pass step by step through one recurrent chain, attention lets tokens connect more directly across distance.

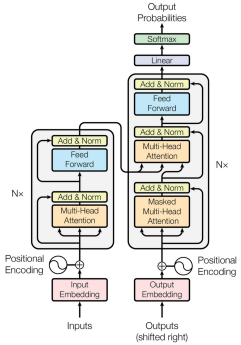
Why that mattered

Long-range relationships became easier to model, and training could be scaled much more effectively.

The balanced view

Transformers solved several major limitations of recurrence, but they make more sense if we first understand what recurrence struggled with.

A First Look at the Transformer



What to notice now

The architecture no longer depends on one simple hidden state passing from step to step through the whole sequence.

Why this is enough for today

The details come later. For now, the important point is that the memory problem was attacked from a new angle.

Why RNNs Still Matter

Historical value

RNNs made the memory problem visible and pushed the field to think seriously about sequence structure.

Conceptual value

If students understand RNNs, they understand what later sequence models were trying to improve.

This lecture turns “sequence modeling” from a buzzword into a concrete technical question about memory.

Why AI6 and NN6 Belong Together

What AI6 showed

Sequential decision making depends on how consequences travel through time.

What NN6 showed

Sequence modeling depends on how information travels through time.

Summary

- RNNs process sequences by updating a hidden state over time, which lets earlier information influence later outputs.
- The main technical difficulty is memory: useful information can fade as it passes through many steps.
- Embeddings, recurrent updates, and backpropagation through time together form the basic RNN story.
- LSTM and GRU were designed to control memory more carefully and reduce the weaknesses of plain recurrence.
- Transformers became dominant because they model long-range relationships more directly and scale more effectively.

Next Time

Where this story goes next

Later neural lectures keep following the evolution of sequence and language models beyond classical recurrence.

What to keep in mind

NN5 taught us to respect structure in images. NN6 taught us to respect structure in time. Later architectures keep building on that same lesson.

Thank You