CS 486/686
Neural Networks I

Yuntian Deng

Lecture 19

RN 21.1 · PM 7.5

Learning goals

- Describe the **simple mathematical model** of a neuron.
- Describe desirable properties of an **activation function**.
- Distinguish **feedforward** from **recurrent** networks.
- Learn a **perceptron** for a simple logical function.
- Determine the logical function represented by a perceptron.
- Explain why a perceptron **cannot represent XOR**.
- Understand **recurrent neural networks**.

Learning complex non-linear relationships

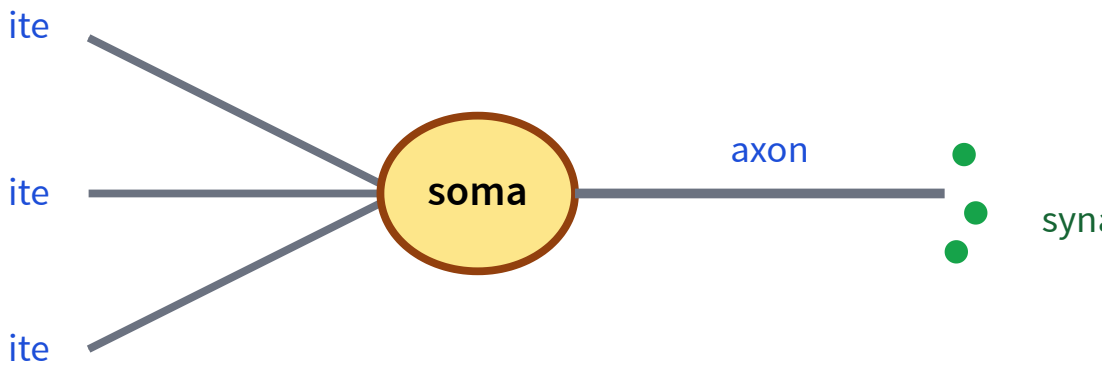
In vision, speech, translation, and beyond, the relationship between inputs and outputs is too complex to hand-program.

> We need a model that can:
>
> - Learn complex relationships.
> - Be trained efficiently on lots of data.
> - Generalize without overfitting.

Loose inspiration: the human brain — many simple components, densely connected.

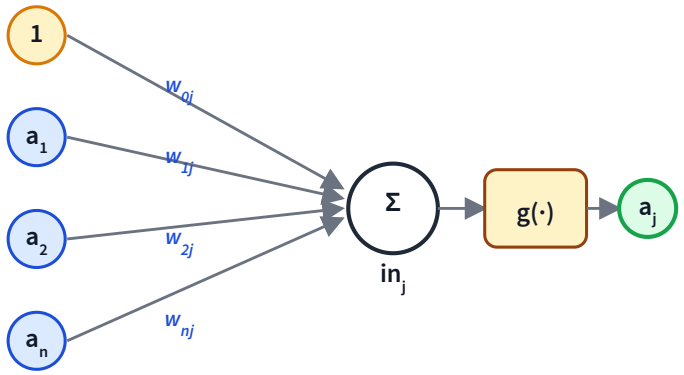
From brain neurons to math neurons

- **Dendrites** receive input signals.
- **Soma** processes the signals.
- **Axon** sends an output signal.
- **Synapses** connect neurons together.
- Neurons fire (or don't) based on the inputs.



The math model abstracts this into **weighted sum + non-linear activation**.

The math model (McCulloch & Pitts, 1943)



$$in_j = \sum_{i=0}^{n} w_{ij}\, a_i$$

$$a_j = g(in_j)$$

$a_0 = 1$ is the bias; w_{0j} acts as a threshold. g is a non-linear **activation function**.

Desirable properties of the activation function

Non-linear

Complex relationships are non-linear; stacking linear layers stays linear.

Mimics real neurons

Fires (≈ 1) when input is big; quiet (≈ 0) otherwise.

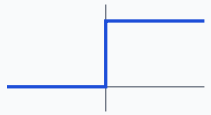
Differentiable

So we can train via gradient-based optimization (next lecture!).

Four common activation functions

Step

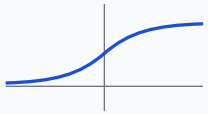
$$g(x) = 1 \text{ if } x > 0, \text{ else } 0.$$



Simple but *not differentiable*; rarely used in practice.

Sigmoid

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



Smooth, but *vanishing-gradient* in saturation regions.

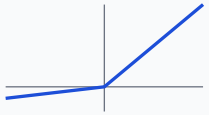
ReLU

$$g(x) = \max(0, x)$$

Fast, sparse; *dying-ReLU* when input stays negative.

Leaky ReLU

$$g(x) = \max(0, x) + k \cdot \min(0, x)$$



Small negative slope keeps gradient alive on the left.

Networks: feedforward vs recurrent

Feedforward

Connections form a **DAG** (no loops). The network is just a function of its inputs.

Recurrent

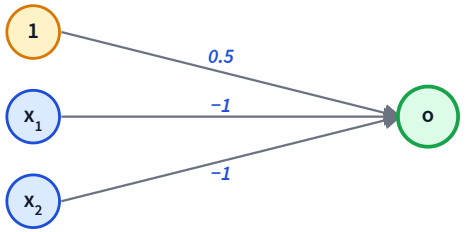
Outputs feed back as inputs. Supports **short-term memory** — behavior depends on history.

Perceptron

A **single-layer feedforward** network: inputs connect directly to outputs. Can represent AND, OR, NOT and many other logical functions.

Identify a perceptron

Q1. The perceptron below uses step activation with weights $w_0 = 0.5, w_1 = -1,$ $w_2 = -1$. What logical function does it compute?



A. $x_1 \wedge x_2$

B. $\neg(x_1 \wedge x_2)$

C. $x_1 \vee x_2$

D. $\neg(x_1 \vee x_2)$ (NOR)

D — NOR. The output is $g(0.5 - x_1 - x_2)$, which is 1 only when $0.5 - x_1 - x_2 > 0$, i.e. both x_1 and x_2 are 0.

Learn a perceptron: AND

Q2. Find weights (w_0, w_1, w_2) so the perceptron $o = g(w_0 + w_1 x_1 + w_2 x_2)$ computes $x_1 \wedge x_2$. The AND truth table outputs 1 *only* on $(1, 1)$.

A. $w_0 = -1, \;\; w_1 = 0.5, \;\; w_2 = 0.5$

B. $w_0 = 0.5, \;\; w_1 = -1, \;\; w_2 = 1$

C. $w_0 = 1.5, \;\; w_1 = -1, \;\; w_2 = -1$

D. $w_0 = -1.5, \;\; w_1 = 1, \;\; w_2 = 1$

D. Check: $-1.5 + x_1 + x_2 > 0$ iff $x_1 + x_2 > 1.5$, i.e. both equal 1. The other options either misclassify $(1, 1)$ or fire on $(0, 0)$.

More perceptron arithmetic

Step activation: $g(x) = 1$ if $x > 0$, else 0.

Q3. What does $h_1 = g(x_1 + x_2 - 0.5)$ compute?

A. $x_1 \vee x_2$ (OR)

B. $x_1 \wedge x_2$

C. $\neg(x_1 \vee x_2)$

D. $\neg(x_1 \wedge x_2)$

A — OR. Outputs 1 if at least one input is 1.

Q4. What does $h_2 = g(-x_1 - x_2 + 1.5)$ compute?

A. $x_1 \vee x_2$

B. $x_1 \wedge x_2$

C. $\neg(x_1 \vee x_2)$

D. $\neg(x_1 \wedge x_2)$ (NAND)

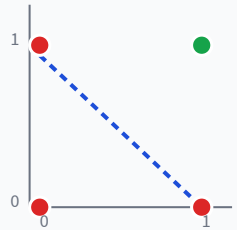
D — NAND. Outputs 0 only when both inputs are 1.

Remember h_1 and h_2 — we'll combine them to build XOR.

Why a perceptron can't represent XOR

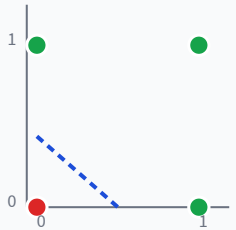
A perceptron is a **linear classifier**. Its decision boundary is a hyperplane in the input space.

AND



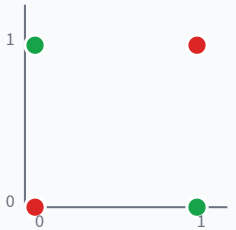
Separable

OR



Separable

XOR

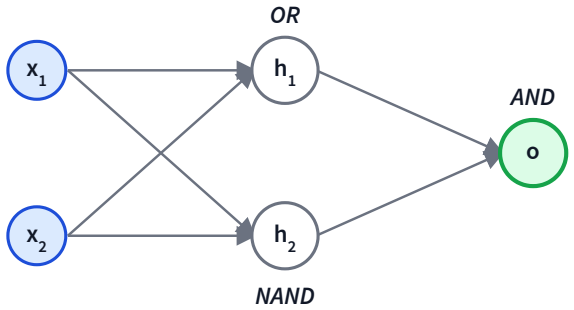


Not separable

Minsky & Papert (1969) showed this — and it triggered the first AI winter.

XOR as a 2-layer network

Rewrite XOR using gates a perceptron *can* handle: $x_1 \oplus x_2 = (x_1 \vee x_2) \wedge \neg(x_1 \wedge x_2) = h_1 \wedge h_2$.



Truth-table check:

x_1	x_2	h_1 (OR)	h_2 (NAND)	$h_1 \wedge h_2$
0	0	0	1	0
0	1	1	1	1
1	0	1	1	1
1	1	1	0	0

Matches XOR exactly. One hidden layer is enough.

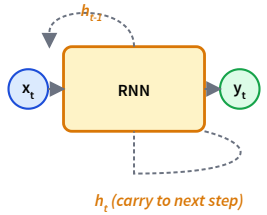
Recurrent networks: process a sequence

When inputs come as a sequence (text, audio, sensor data), the same network reads one element at a time and carries a **state** across steps.

Recurrence

$$h_t = f_W(h_{t-1}, x_t), \quad y_t = f_Y(h_t)$$

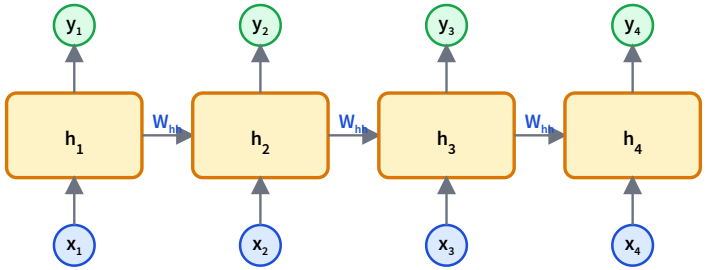
The state h_t summarizes everything seen so far.



Same weights at every step — the model can handle *any-length* sequence with a fixed parameter count.

Vanilla RNN: equations + unrolled

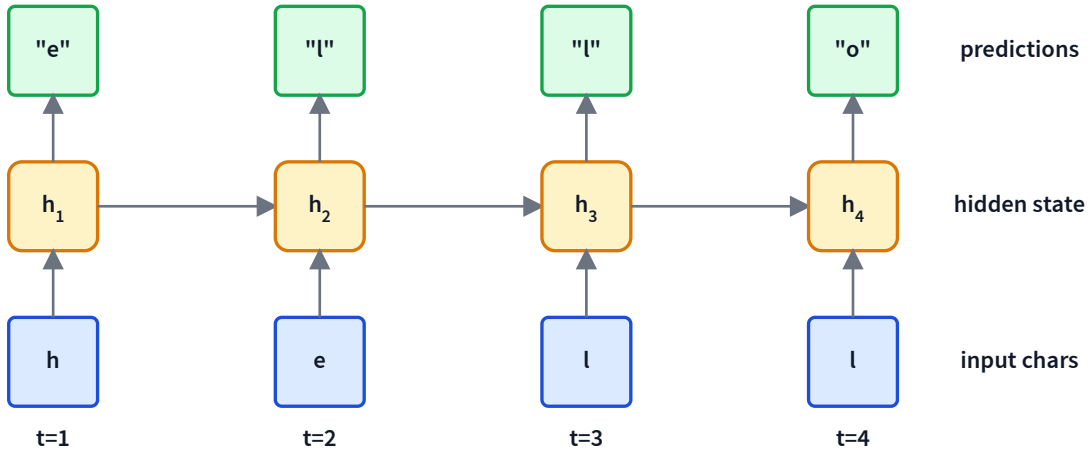
$$h_t = \tanh\big(W_{hh}\, h_{t-1} + W_{xh}\, x_t\big), \qquad y_t = W_{hy}\, h_t$$



The **same** W_{hh}, W_{xh}, W_{hy} are reused at every time step.

Application: language modeling

Feed characters one at a time; predict the next character at each step.



Trained to predict the next character; sample autoregressively to generate text.

Learning goals (recap) — Next: training

✓ Describe the **math neuron**.

✓ Describe desirable properties of an **activation function**.

✓ Distinguish **feedforward** from **recurrent** networks.

✓ Learn / identify a **perceptron** for a logical function.

✓ Explain why a perceptron **cannot represent XOR**.

✓ Understand **RNNs** and how they process sequences.

L20: how do we actually *train* these networks? — gradient descent + **backpropagation**.