

UNIT 5

Neural Networks and Natural Language Processing

(7 Hours) — Artificial Intelligence (CACS410)

⚠ Please Note

These notes were prepared for quick revision and **may contain errors or omissions**. Please take care: cross-check important definitions, formulas, algorithms and worked examples against your prescribed textbook, class notes and teacher's instructions before relying on them (especially for exams).

5.1 Artificial Neural Networks

Introduction

An **Artificial Neural Network (ANN)** is a computing model inspired by the human brain: a network of simple, highly interconnected processing units (**artificial neurons**) that learns from examples by adjusting the strengths (**weights**) of the connections. The human brain has roughly 86 billion neurons, each connected to thousands of others.

Biological neuron	Artificial neuron
Dendrites receive signals	Inputs $x_1 \dots x_n$
Synapse strength	Weights $w_1 \dots w_n$
Cell body (soma) sums the signals	Summation unit $\sum w_i x_i + b$
Firing threshold	Activation function f
Axon carries the output	Output y

- **Characteristics:** learning from data, parallel processing, fault tolerance, ability to model non-linear relationships, generalization to unseen inputs.
- **Applications:** pattern and image recognition, speech recognition, forecasting, medical diagnosis, control systems, language models.

Mathematical Model of a Neuron

The McCulloch–Pitts / perceptron model computes a weighted sum of the inputs plus a bias and passes it through an activation function:

$$\text{net} = w_1x_1 + w_2x_2 + \dots + w_nx_n + b, y = f(\text{net})$$

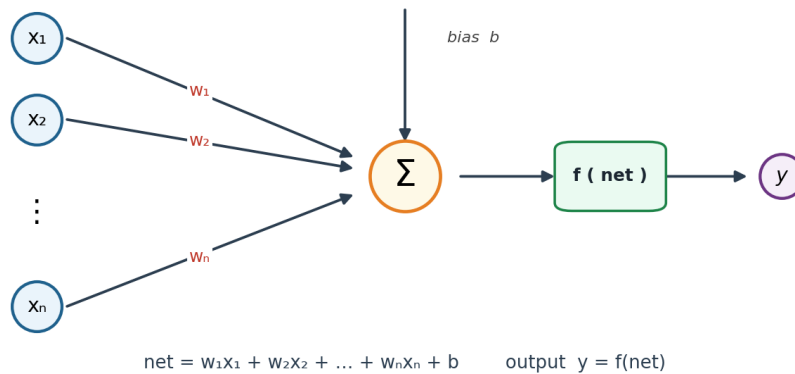


Figure 5.1 — Mathematical model of an artificial neuron

Activation Functions

Function	Formula	Output range	Use
Step (threshold)	$f(x) = 1$ if $x \geq 0$ else 0	$\{0, 1\}$	McCulloch–Pitts neuron, perceptron
Sigmoid	$f(x) = 1 / (1 + e^{(-x)})$	$(0, 1)$	Probabilities, back-propagation; $f' = f(1 - f)$
Tanh	$f(x) = (e^x - e^{-x}) / (e^x + e^{-x})$	$(-1, 1)$	Hidden layers, zero-centred
ReLU	$f(x) = \max(0, x)$	$[0, \infty)$	Deep networks, fast training
Linear	$f(x) = x$	$(-\infty, \infty)$	Regression output layer

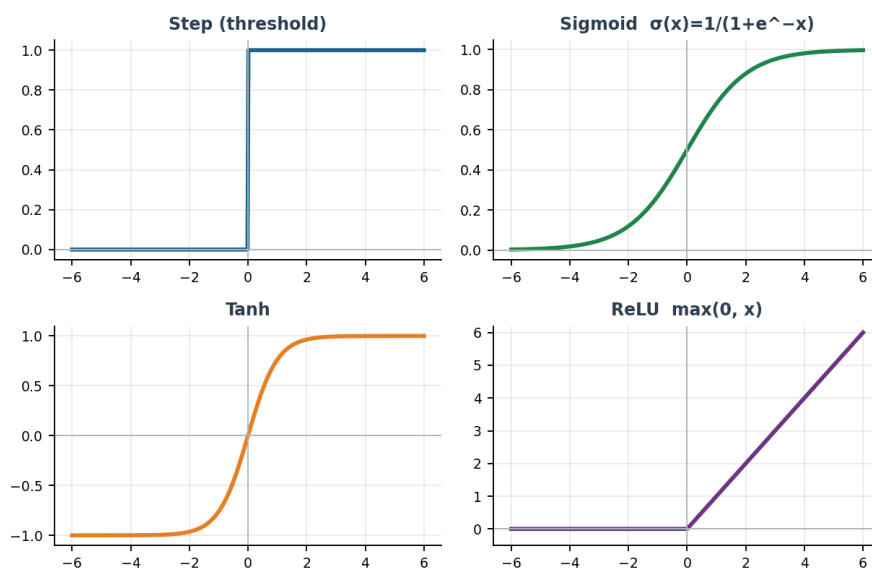


Figure 5.2 — Common activation functions

Types of Neural Networks

- **Feed-forward networks:** signals flow in **one direction** (input → hidden → output) with no cycles. Examples: single-layer perceptron, multilayer perceptron (MLP). Used for classification and regression.
- **Feed-back (recurrent) networks:** contain **loops/feedback connections**, so the output depends on previous states and the network has memory. Examples: Hopfield network, Boltzmann machine, Elman/RNN.

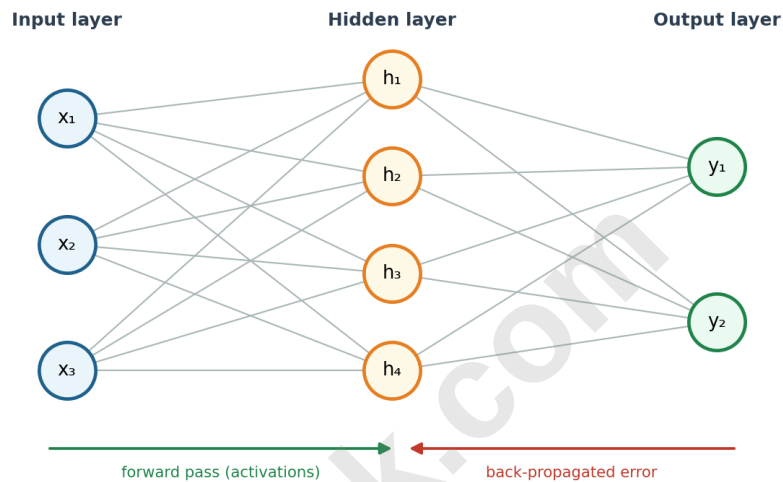


Figure 5.3 — A feed-forward network with one hidden layer

Point	Feed-forward	Feed-back (recurrent)
Connections	Acyclic, forward only	Contains cycles / feedback
Memory	None (output depends only on current input)	Remembers previous states
Behavior	Static mapping	Dynamic; converges to stable states or processes sequences
Typical use	Classification, regression	Associative memory, optimization, sequences

Gate Realization using Neural Networks

A McCulloch–Pitts neuron outputs **1** if $\sum w_i x_i \geq \theta$ (threshold θ), else 0. Choosing weights and threshold realizes logic gates:

Gate	w_1	w_2	Threshold θ	Check ($x_1 x_2 \rightarrow y$)
AND	1	1	2	00→0, 01→0, 10→0, 11→1 (sum = 2 only for 11)
OR	1	1	1	00→0, 01→1, 10→1, 11→1

Gate	w_1	w_2	Threshold θ	Check ($x_1 x_2 \rightarrow y$)
NOT (one input)	-1	—	0	$x = 0: 0 \geq 0 \rightarrow 1; x = 1: -1 < 0 \rightarrow 0$
NAND	-1	-1	-1	$00 \rightarrow 1, 01 \rightarrow 1 (-1 \geq -1), 10 \rightarrow 1, 11 \rightarrow 0 (-2 < -1)$
NOR	-1	-1	0	$00 \rightarrow 1, 01 \rightarrow 0, 10 \rightarrow 0, 11 \rightarrow 0$

XOR cannot be realized by a single neuron because it is not *linearly separable* (no straight line separates its 1s from its 0s). It needs a hidden layer: **XOR = (x_1 OR x_2) AND (x_1 NAND x_2)**.

Example — XOR with two layers

Hidden neuron $h_1 = \text{OR}(x_1, x_2)$ (weights 1, 1; $\theta = 1$). Hidden neuron $h_2 = \text{NAND}(x_1, x_2)$ (weights -1, -1; $\theta = -1$). Output $y = \text{AND}(h_1, h_2)$ (weights 1, 1; $\theta = 2$).

Truth check

(0,0): $h_1 = 0, h_2 = 1 \rightarrow y = 0$. (0,1): $h_1 = 1, h_2 = 1 \rightarrow y = 1$. (1,0): $h_1 = 1, h_2 = 1 \rightarrow y = 1$. (1,1): $h_1 = 1, h_2 = 0 \rightarrow y = 0$. This is XOR.

Learning in Neural Networks

Learning means adjusting weights so that the network's outputs match the desired targets. In **supervised learning** the network is shown input–target pairs, the **error** is measured and the weights are changed to reduce it (gradient descent).

Perceptron learning rule (single neuron): $w_i \leftarrow w_i + \eta (t - y) x_i$, $b \leftarrow b + \eta (t - y)$, where t = target, y = actual output and η = learning rate.

Example — Perceptron learns AND

Start with $w = (0, 0)$, $b = 0$, $\eta = 1$, output = 1 if net > 0. Presenting the four patterns repeatedly, the weights change only when an output is wrong. After 5 passes the weights stabilize at **$w = (2, 1)$, $b = -2$** , which classifies all four patterns correctly (for 1,1: $2 + 1 - 2 = 1 > 0 \rightarrow 1$; for 1,0: $2 - 2 = 0 \rightarrow 0$; for 0,1: $1 - 2 = -1 \rightarrow 0$; for 0,0: $-2 \rightarrow 0$).

Back-Propagation Algorithm

A multilayer network is trained by **back-propagation of error**: the output error is propagated backward through the layers to compute how much each weight contributed, and weights are changed by **gradient descent** on the error $E = \frac{1}{2} \sum (t_k - o_k)^2$.

- **Step 1:** initialize all weights with small random values.
- **Step 2 — Forward pass:** compute hidden outputs $o_j = \sigma(\sum w_{ij} x_i)$ and output values $o_k = \sigma(\sum w_{jk} o_j)$.
- **Step 3 — Output error term:** $\delta_k = (t_k - o_k) \cdot o_k (1 - o_k)$.
- **Step 4 — Hidden error term:** $\delta_j = o_j (1 - o_j) \cdot \sum_k \delta_k w_{jk}$.
- **Step 5 — Weight update:** $\Delta w = \eta \cdot \delta(\text{receiving node}) \cdot \text{output}(\text{sending node})$, then $w \leftarrow w + \Delta w$.
- **Step 6:** repeat for all training patterns (one pass = one *epoch*) until the error is acceptably small.

Example — One training step ($\eta = 1$, sigmoid, no bias)

Network 2-2-1. Inputs $x_1 = 0.35$, $x_2 = 0.9$; target $t = 0.5$. Weights: input $\rightarrow$ h₁: $w_{13} = 0.1$, $w_{23} = 0.8$; input $\rightarrow$ h₂: $w_{14} = 0.4$, $w_{24} = 0.6$; hidden $\rightarrow$ output: $w_{35} = 0.3$, $w_{45} = 0.9$.

Forward pass

$\text{net}(h_1) = 0.35 \times 0.1 + 0.9 \times 0.8 = 0.755 \rightarrow o_3 = \sigma(0.755) = \mathbf{0.6803}$. $\text{net}(h_2) = 0.35 \times 0.4 + 0.9 \times 0.6 = 0.68 \rightarrow o_4 = \sigma(0.68) = \mathbf{0.6637}$.

$\text{net}(\text{out}) = 0.6803 \times 0.3 + 0.6637 \times 0.9 = 0.8014 \rightarrow o_5 = \sigma(0.8014) = \mathbf{0.6903}$. Error $E = \frac{1}{2}(0.5 - 0.6903)^2 = \mathbf{0.0181}$.

Backward pass and update

$\delta_5 = (0.5 - 0.6903) \times 0.6903 \times (1 - 0.6903) = \mathbf{-0.0407}$.

$\delta_3 = 0.6803 \times (1 - 0.6803) \times \delta_5 \times w_{35} = \mathbf{-0.00265}$; $\delta_4 = 0.6637 \times (1 - 0.6637) \times \delta_5 \times w_{45} = \mathbf{-0.00817}$.

New weights: $w_{35} = 0.3 + (-0.0407)(0.6803) = \mathbf{0.2723}$; $w_{45} = 0.9 + (-0.0407)(0.6637) = \mathbf{0.8730}$; $w_{13} = 0.0991$; $w_{23} = 0.7976$; $w_{14} = 0.3971$; $w_{24} = 0.5926$.

After the update the output is **0.6820** (closer to 0.5) and E falls to **0.0166**.

Note: Practical issues: choice of learning rate η , local minima, vanishing gradients with sigmoid in deep networks, overfitting (use validation data, regularization, early stopping). A momentum term is often added to speed up training.

Hopfield Network

A Hopfield network (John Hopfield, 1982) is a **fully connected recurrent network** of binary neurons (± 1) with **symmetric weights** ($w_{ij} = w_{ji}$) and **no self-connections** ($w_{ii} = 0$). It works as a **content-addressable (associative) memory**: given a noisy or partial pattern it recalls the closest stored pattern.

- **Storage (Hebbian rule):** for patterns $x^{(p)}$, $w_{ij} = (1/N) \sum_p x_i^{(p)} x_j^{(p)}$.
- **Recall:** neurons are updated (asynchronously) as $s_i \leftarrow \text{sign}(\sum_j w_{ij} s_j)$.
- **Energy:** $E = -\frac{1}{2} \sum_i \sum_j w_{ij} s_i s_j$ never increases during updates, so the network settles in a **stable state** (a local energy minimum = a stored pattern).
- **Capacity** is limited, about 0.14 N patterns for N neurons; beyond that, spurious states appear.

Example — Storing and recalling one pattern

Store $p = (1, -1, 1)$. Using $w_{ij} = x_i x_j$ (ignoring the $1/N$ scale): $w_{12} = -1$, $w_{13} = +1$, $w_{23} = -1$.

Present the noisy input $(1, 1, 1)$ (energy +1). Update neuron 2: $\text{net} = w_{21}s_1 + w_{23}s_3 = (-1)(1) + (-1)(1) = -2 \rightarrow s_2 = -1$. The state becomes $(1, -1, 1)$ — the stored pattern (energy -3).

Boltzmann Machines

A Boltzmann machine (Hinton and Sejnowski, 1985) is a **stochastic recurrent network** with symmetric weights, having **visible** units (data) and **hidden** units. Neurons are binary and update **probabilistically**: $P(s_i = 1) = 1 / (1 + e^{-(\Delta E_i/T)})$, $\Delta E_i = \sum_j w_{ij} s_j + b_i$, with temperature T . Using the idea of **simulated annealing** (T lowered gradually), it can escape local energy minima and find low-energy (good) states. The probability of a network state is $P(s) \propto e^{-(E(s)/T)}$.

- **Learning:** $\Delta w_{ij} = \eta (\langle s_i s_j \rangle_{\text{data}} - \langle s_i s_j \rangle_{\text{model}})$: strengthen connections between units active together in the data and weaken those active together in the model's own “dreaming”. This is accurate but very slow.
- A **Restricted Boltzmann Machine (RBM)** has no connections within a layer, making training practical; RBMs are the building blocks of deep belief networks.
- **Uses:** pattern completion, optimization (e.g., travelling salesman), feature learning.

Point	Hopfield network	Boltzmann machine
Neuron update	Deterministic	Stochastic (probabilistic)
Hidden units	None	Yes
Local minima	Can get trapped	Escapes using temperature (annealing)
Main use	Associative memory	Learning distributions, optimization
Training	Hebbian, one-shot	Slow, sampling-based

5.2 Natural Language Processing (NLP)

Concepts of Natural Language Understanding and Generation

Natural Language Processing is the field of AI concerned with enabling computers to understand, interpret and produce human (natural) languages such as English or Nepali. It has two major parts:

Point	Natural Language Understanding (NLU)	Natural Language Generation (NLG)
Direction	Text/speech → meaning (internal representation)	Meaning / data → text/speech
Main task	Interpret input and resolve ambiguity	Decide what to say and how to say it
Stages	Lexical, syntactic, semantic, discourse, pragmatic analysis	Content determination, document structuring, sentence aggregation, lexical choice, referring expressions, surface realization
Difficulty	Harder because of ambiguity	Easier to control, but must sound natural
Examples	Voice assistants, question answering, sentiment analysis	Report writing, chatbot replies, summarization, machine translation output

Why Language is Hard — Ambiguity

Type	Example
Lexical	“bank” — river bank or financial institution
Syntactic (structural)	“I saw the man with the telescope” — who has the telescope?
Semantic	“The chicken is ready to eat” — the chicken eats, or is eaten?
Referential (anaphoric)	“Ram told Shyam that he won” — who won?
Pragmatic	“Can you pass the salt?” — a request, not a question about ability

Steps in Natural Language Processing

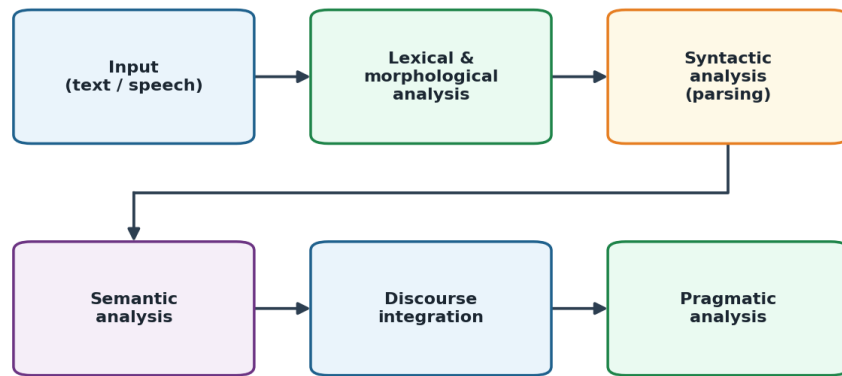


Figure 5.4 — Stages of natural language processing

- **Lexical / morphological analysis:** split text into words (tokens), find word roots and affixes (stemming, lemmatization), assign parts of speech.
- **Syntactic analysis (parsing):** check grammar and build the structure of the sentence.
- **Semantic analysis:** assign meaning to the structure.
- **Discourse integration:** interpret a sentence using the sentences before it (e.g., resolving pronouns).
- **Pragmatic analysis:** interpret the intended meaning in context.

Syntax Analysis

Syntax analysis checks whether a sentence is well formed according to a **grammar** and produces a **parse tree**. A context-free grammar (CFG) is typical:

S	-> NP VP	NP	-> Det N Name
VP	-> V NP V	Det	-> the a
N	-> boy ball	V	-> hit saw

Example — Parse “The boy hit the ball”

Parse tree (bracket form): [S [NP [Det The] [N boy]] [VP [V hit] [NP [Det the] [N ball]]]]

Parsing may be **top-down** (start from S and expand rules) or **bottom-up** (start from words and combine). The ill-formed sentence “Boy the hit ball the” cannot be parsed and is rejected.

Syntactic ambiguity yields more than one parse tree.

Semantic Analysis

Semantic analysis maps the syntactic structure to a **meaning representation** (for example, logic or a frame), determines **word senses** and checks that the meaning is sensible. “The boy hit the ball” becomes **hit(boy, ball)** or $\exists b, \exists l (\text{Boy}(b) \wedge \text{Ball}(l) \wedge \text{Hit}(b, l))$. A sentence can be grammatical but meaningless, e.g., Chomsky's “Colorless green ideas sleep furiously”, or “Hot ice-cream” — semantic analysis rejects these using selectional restrictions.

Pragmatic Analysis

Pragmatics deals with **meaning in context**: the speaker's intention, world knowledge and the situation. Literal meaning is not enough:

- “Can you pass the salt?” — literally a question about ability, pragmatically a polite request.
- “It's cold in here.” — may be a request to close the window.
- “Do you know what time it is?” — a request for the time, not a yes/no question.

Pragmatic analysis uses **speech acts**, shared knowledge and the dialogue history. Together with **discourse integration** (resolving, e.g., “Ram took the book from Shyam. *He* was angry.”) it is the hardest stage to automate.

Applications of NLP: machine translation, speech recognition and assistants, chatbots, question answering, information retrieval and extraction, sentiment analysis, text summarization, spelling and grammar checking.

Key Takeaways

- ✓ An artificial neuron computes $y = f(\sum w_i x_i + b)$; common activations are step, sigmoid, tanh and ReLU.
- ✓ Feed-forward networks have no cycles; feed-back (recurrent) networks have loops and memory.
- ✓ Single neurons realize AND, OR, NOT, NAND, NOR; XOR needs two layers (not linearly separable).
- ✓ Back-propagation: forward pass, compute error, propagate δ backward, update weights by gradient descent.
- ✓ Hopfield network = deterministic associative memory with energy minimization; Boltzmann machine = stochastic version with hidden units and annealing.
- ✓ NLP pipeline: lexical $\rightarrow$ syntactic $\rightarrow$ semantic $\rightarrow$ discourse $\rightarrow$ pragmatic analysis; NLU maps language to meaning and NLG maps meaning to language.

Review Questions

- Explain the mathematical model of an artificial neuron and the common activation functions.
- Differentiate feed-forward and feed-back networks.
- Realize AND, OR and NOT gates with McCulloch–Pitts neurons. Why can't XOR be realized by a single neuron?
- Explain the back-propagation algorithm with an example.
- Explain the Hopfield network and the Boltzmann machine; list their differences.
- Explain NLU and NLG. Describe the steps of NLP with examples of syntax, semantic and pragmatic analysis.

— *End of Unit 5* —

Reminder: these notes may contain errors — verify with your textbook and teacher.

bcalink.com