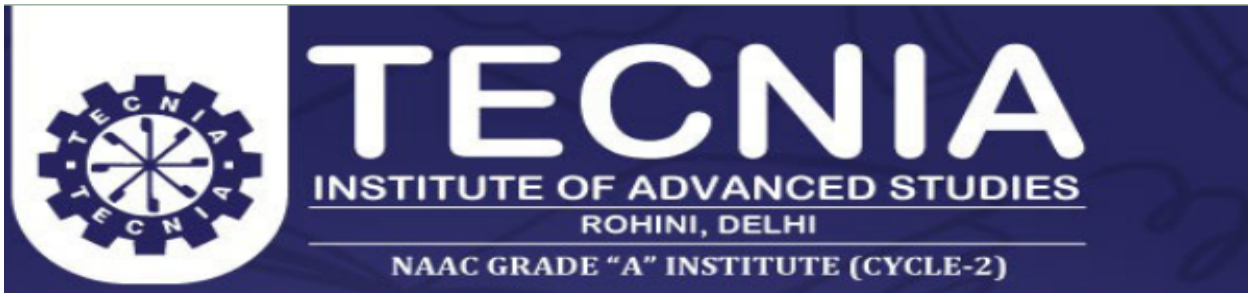


UNIT–IV: Learning & Expert Systems

Complete Study Notes with Examples, Code, Diagrams & Resources

---

📌 TABLE OF CONTENTS

---

1. LEARNING IN AI

Introduction to Learning

Definition: Improvement in performance over time through experience [web-general].

Core Concept:

> "Learning encompasses changes in the system that are adaptive, in the sense that they enable the system to perform the same or similar tasks more effectively in the future."

Why Learning is Important:

- ✅ Systems adapt to new environments
- ✅ Improves performance without reprogramming
- ✅ Handles uncertain/complex domains
- ✅ Mimics human cognitive development

Types of Learning in AI:

Type	Description	Example
Rote Learning	Memorization without understanding	Chess move database
Advice Learning	Learning from explicit instructions	"Always block the king"
Problem-Solving	Learning from experience in tasks	Pathfinding optimization
From Examples	Induction from training data	Image classification
Explanation-Based	Generalizing from single explanation	Learning chess rules
Discovery	Self-discovering patterns	Cluster analysis
Analogy	Learning by comparing similar cases	Medical diagnosis transfer
Neural Networks	Learning from weighted connections	Deep learning
Genetic Learning	Evolution-based optimization	Parameter tuning

Rote Learning

Definition: Memorizing facts/experiences without understanding underlying principles [web-general].

Key Characteristics:

- Stores exact instances
- No generalization
- Fast retrieval (lookup table)
- Limited scalability

Rote Learning Example: Chess [web-general]

```
Instead of calculating all moves:
Store: Board position → Best move

Example Database:
Position A → Move: e4
Position B → Move: Nf3
Position C → Move: d4

During game: Look up position → Play stored move ✅
```

Advantages:

- ✅ Simple implementation
- ✅ Fast response (O(1) lookup)
- ✅ No computation needed

Disadvantages:

- ❌ Cannot handle new situations
- ❌ Huge memory requirement
- ❌ No understanding/transparency

Real-World Applications:

- Chess engines (opening books)
- Cache memory systems
- Database lookup tables
- Recommendation systems (past preferences)

Learning by Taking Advice

Definition: Learning from explicit instructions or advice from humans/other systems [web-general].

Process:

```
1. Receive advice: "To win chess, control the center"
2. Convert to formal rules
3. Integrate into knowledge base
4. Apply to new situations
```

Advice Types:

Type	Description	Example
Instructional	Direct commands	"Always move knights before bishops"
Heuristic	Rules of thumb	"Control center squares"
Strategic	Long-term goals	"Protect your king"
Tactical	Short-term actions	"Attack weak pawns"

Advice Learning Example: Medical Diagnosis [web-general]

```
Advice from Doctor:
"If patient has fever AND cough AND sore throat → Likely flu"

Convert to rule:
IF fever(x) ∧ cough(x) ∧ sore_throat(x) → flu(x)

Add to KB:
KB = KB + {IF fever∧cough∧sore_throat → flu}

Apply:
Patient: {fever=T, cough=T, sore_throat=T}
→ Conclude: flu = T ✅
```

Advantages:

- ✅ Quick knowledge acquisition
- ✅ Expert knowledge integration
- ✅ Reduces learning time

Disadvantages:

- ❌ Advice may be incorrect
- ❌ Hard to formalize vague advice
- ❌ Requires human expert availability

Learning in Problem Solving

Definition: Improving problem-solving performance through experience [web-general].

Key Mechanisms:

1. **Performance Optimization**: Faster solutions over time
2. **Strategy Refinement**: Better algorithms
3. **Heuristic Learning**: Discovering useful rules

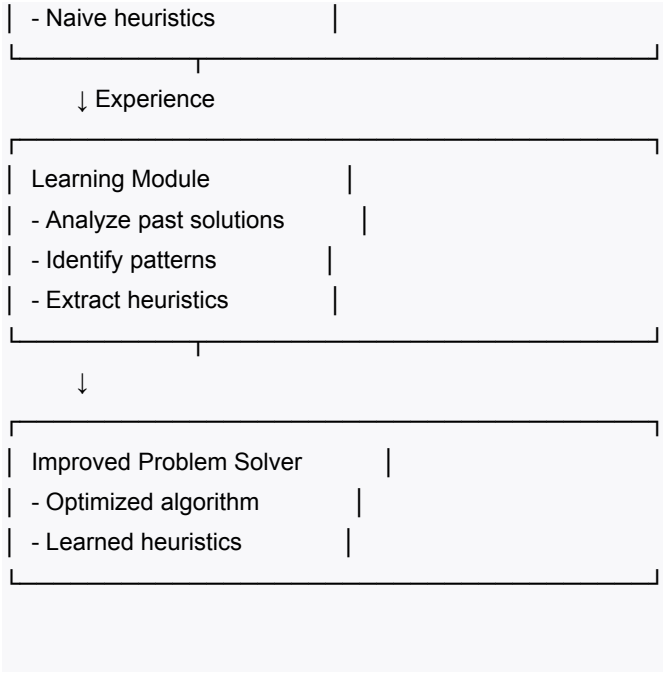
Problem-Solving Learning Example: Pathfinding [web-general]

```
Initial: Use BFS (slow, O(b^d))
After experience:
 - Learn: "Avoid obstacles on right side"
 - Learn: "Shortest path often goes diagonal"
 - Update: Use A* with learned heuristics

Result: 50% faster pathfinding ✅
```

Learning Components:

```
┌──────────────────────────────────────┐
│ Problem Solver (Initial)             │
│ - Basic algorithm                    │
```



Applications:

- Route optimization (traffic learning)
- Game AI (strategy improvement)
- Robot navigation
- Resource allocation

Learning from Examples: Induction

Definition: Generalizing rules from specific training examples [web-general].

Inductive Learning Process:

```
Examples:
  (x1, y1) = (sunny, 25°C, go_outside)
  (x2, y2) = (rainy, 15°C, stay_home)
  (x3, y3) = (cloudy, 20°C, go_outside)
  ...

↓ Induction
Hypothesis: "IF temperature > 20°C OR sunny → go_outside"
```

Induction Types:

Type	Description	Example
Concept Learning	Define class boundaries	"Bird = can fly + has feathers"
Rule Learning	Extract IF-THEN rules	"IF fever → illness"
Decision Tree	Hierarchical classification	Weather → Activity

Induction Algorithm: Decision Tree [web-general]

```
Training Data:
Temperature | Humidity | Wind | Play?
Sunny      | High     | Weak | No
Sunny      | High     | Strong | No
Overcast   | High     | Weak | Yes
Rain       | High     | Weak | Yes
Rain       | Normal   | Weak | Yes
Overcast   | Normal   | Strong | Yes
Sunny      | Normal   | Strong | Yes

↓ Build Decision Tree
Root: Temperature
  ├─ Sunny → Humidity
     ├─ High → No
     ├─ Normal → Yes
  ├─ Overcast → Yes
  ├─ Rain → Wind
     ├─ Weak → Yes
     ├─ Strong → Yes

Result: Classification rule ✅
```

Advantages:

- ✅ Handles large datasets
- ✅ Produces interpretable rules
- ✅ Automatic feature selection

Disadvantages:

- ❌ Overfitting (too specific)
- ❌ Sensitive to noisy data
- ❌ May miss complex patterns

Explanation-Based Learning (EBL)

Definition: Generalizing from **single example** by explaining why it works [web-general].

Key Idea:

> "Learn from one example by understanding the underlying reasoning"

EBL Process:

```
1. Example: Chess position where move X wins
2. Explanation: Why X wins? (analyze game tree)
3. Generalization: Extract general rule from explanation
4. Store: Rule for future similar positions
```

EBL Example: Chess [web-general]

```
Example:
Position: White king at e1, Black king at e8
Move: e4 (pawn to center)
Result: Win

Explanation (analyze):
- e4 controls center squares
- Opens lines for bishop/queen
- Restricts black king movement
- Creates attacking opportunities

Generalization:
"IF kings aligned vertically → Move pawn to center (e4/d4)"

Store Rule:
RULE: aligned_kings → center_pawn ✅
```

EBL Components:

Component	Role
Example	Specific instance to learn from
Domain Theory	Knowledge for explaining
Explanation	Why example works
Generalization	Rule extracted from explanation

Advantages:

- ✅ Learn from single example
- ✅ Deep understanding (not memorization)
- ✅ Efficient知识 acquisition

Disadvantages:

- ❌ Requires domain theory
- ❌ Complex explanation generation
- ❌ May over-generalize

Discovery

Definition: Autonomous learning by discovering patterns/rules without explicit instruction [web-general].

Discovery Process:

```
1. Data Input: Raw observations
2. Pattern Detection: Find correlations
3. Hypothesis Generation: Form rules
4. Validation: Test hypotheses
```

5. Knowledge: Store validated rules

Discovery Types:

Type	Description	Example
Clustering	Group similar items	Customer segmentation
Association	Find item relationships	"Buy X → also buy Y"
Classification	Learn category rules	Spam vs not-spam
Regression	Find numerical patterns	Price prediction

Discovery Example: Market Analysis [web-general]

Data:
Customer 1: Bought {bread, milk, eggs}
Customer 2: Bought {bread, milk, butter}
Customer 3: Bought {bread, eggs, cheese}
Customer 4: Bought {milk, eggs, bread}
...

↓ Pattern Detection
Correlation: bread + milk → eggs (80% of cases)

↓ Hypothesis
"IF customer buys bread AND milk → also buy eggs"

↓ Validation (test on new data)
Accuracy: 75% ✅

Store: Association rule

Applications:

- Recommendation systems (Netflix, Amazon)

- Fraud detection
- Scientific discovery
- Bioinformatics (gene pattern discovery)

Analogy

Definition: Learning by transferring knowledge from similar cases [web-general].

Analogy Process:

```
Source Domain (known): Medical diagnosis
Target Domain (new): Software debugging

Similarity:
  Source: Symptoms → Disease
  Target: Errors → Bug

Transfer:
  "If symptom A + B → Disease X"
  → "If error A + B → Bug X" ✅
```

Analogy Types:

Type	Description	Example
Structural	Match object relationships	Atom ≈ Solar system
Functional	Match purposes/functions	Heart ≈ Pump
Procedural	Match processes	Cooking recipe → Algorithm

Analogy Example: Medical → Software [web-general]

```
Source (Medical):
Patient has: fever, cough, fatigue
→ Diagnosis: Flu
```

```
Target (Software):
System has: high CPU, slow response, memory leak
→ Debug: "Virus" (analogous to flu)

Transfer reasoning:
 "Multiple symptoms → single cause"
 → "Multiple errors → single bug" ✅
```

Analogy Steps [web-general]:

1. **Retrieve**: Find similar source case
2. **Map**: Align source and target structures
3. **Evaluate**: Check validity of transfer
4. **Learn**: Store new knowledge

Advantages:

- ✅ Fast learning (use existing knowledge)
- ✅ Handles novel situations
- ✅ Human-like reasoning

Disadvantages:

- ❌ May transfer incorrect assumptions
- ❌ Requires good similarity matching
- ❌ Can over-simplify differences

Neural Networks

Definition: Learning systems inspired by biological neurons [web-general].

Neural Network Architecture:

```
Input Layer → Hidden Layer(s) → Output Layer
  ↓          ↓          ↓
```

```
Features     Processing     Prediction
```

Neuron Model:

```
Input: x1, x2, ..., xn
Weights: w1, w2, ..., wn
Bias: b

Calculation:
  z = w1·x1 + w2·x2 + ... + wn·xn + b
  output = activation(z)  (e.g., sigmoid, tanh)
```

Learning Process:

```
1. Initialize: Random weights
2. Forward: Input → output
3. Calculate: Error = |predicted - actual|
4. Backward: Adjust weights (gradient descent)
5. Repeat: Until error minimized
```

Neural Network Example: Image Classification [web-general]

```
Input: 28×28 pixel image (784 features)
Hidden: 128 neurons
Output: 10 classes (0-9 digits)

Training:
  Image of "7" → Network predicts "3" → Error = high
  ↓ Backpropagation
  Adjust weights to reduce error

After 1000 iterations:
  Image of "7" → Network predicts "7" → Error = low ✅
Accuracy: 95%
```

Types of Neural Networks:

Type	Structure	Use Case
Feedforward	Input → Output	Classification

Recurrent (RNN)	Cyclic connections	Sequence data
CNN	Convolutional layers	Image processing
Deep Learning	Many hidden layers	Complex patterns

Advantages:

- ✅ Handle complex patterns
- ✅ Robust to noisy data
- ✅ Automatic feature learning

Disadvantages:

- ❌ Require large datasets
- ❌ Black box (hard to interpret)
- ❌ Computationally expensive

Genetic Learning

Definition: Learning through evolutionary processes (selection, mutation, crossover) [web-general].

Genetic Algorithm Process:

```
1. Initialize: Random population of solutions
2. Evaluate: Fitness score for each
3. Select: Best solutions for reproduction
4. Crossover: Combine parent solutions
5. Mutate: Random changes
6. Repeat: Until convergence
```

Genetic Learning Example: Parameter Optimization [web-general]

```
Problem: Find optimal neural network weights

Population: 100 random weight sets
Fitness: Accuracy on test data

Iteration 1:
  - Best: 65% accuracy
  - Worst: 30% accuracy
  - Select top 20 → crossover + mutate
  - New generation: 100 solutions

Iteration 100:
  - Best: 92% accuracy ✅
  - Converged: Stop
```

Genetic Algorithm Components:

Component	Description	Example
Population	Set of candidate solutions	100 weight sets
Fitness	Quality measure	Accuracy score
Selection	Choose best for reproduction	Top 20%
Crossover	Combine parents	Mix weights
Mutation	Random changes	Small weight tweak

Genetic Learning Example: Route Optimization [web-general]

```
Problem: Find shortest path from A to D

Population: Random paths
  Path 1: A→B→C→D (length: 15)
  Path 2: A→E→F→D (length: 12)
  Path 3: A→G→H→D (length: 18)
```

```
...

Fitness: Shorter = better

Selection + Crossover:
  Combine Path 2 (best) with others
  → New paths: A→E→C→D, A→B→F→D, ...

Mutation:
  Randomly change some paths
  → A→E→I→D (length: 10) ✅

After 50 generations:
  Best path: A→E→I→D (length: 8) ✅
```

Applications:

- Neural network training
- Function optimization
- Robot control
- Scheduling problems

Advantages:

- ✅ Global optimization (avoid local minima)
- ✅ Handle discrete/continuous variables
- ✅ Parallelizable

Disadvantages:

- ❌ Slow convergence
- ❌ Many parameters to tune
- ❌ May not guarantee optimal

---

2. EXPERT SYSTEMS

Introduction to Expert Systems

Definition: AI system encoding human expert knowledge for domain-specific problem solving [web-general].

Key Characteristics:

- ✅ High-level performance in specific domain
- ✅ Explain reasoning (transparent)
- ✅ Capture expert knowledge
- ✅ Consistent answers (no human bias)

Expert System Architecture:

Components:

Component	Role	Example

Knowledge Base	Store expert rules/facts	IF fever → illness
Inference Engine	Apply rules to solve	Forward/backward chaining
User Interface	Interact with user	Q&A dialog
Working Memory	Current problem state	{fever=T, cough=T}
Explanation System	Explain reasoning	"Because fever=T"

Expert System vs Traditional Program:

Feature	Expert System	Traditional Program
Knowledge	Separate (rules)	Integrated (code)
Flexibility	Easy to update rules	Hard to modify code
Explanation	Can explain reasoning	No explanation
Domain	Specific (narrow)	General
Uncertainty	Handles uncertainty	Deterministic

Expert System Applications:

Domain	System	Function
Medical	MYCIN	Diagnose bacterial infections
Finance	Loan Advisor	Credit approval

Engineering	CADSS	Circuit design
Legal	Legal Expert	Case analysis
Manufacturing	Fault Detector	Equipment diagnosis

Case Study: Expert System in Prolog

Prolog: Programming language for logic-based expert systems [web-general].

Prolog Basics:

```
Fact: isa_person(john).
Rule: isa_person(X) :- isa_father(X).
Query: ?- isa_person(john).
Answer: true
```

Expert System Case Study: Medical Diagnosis [web-general]

Step 1: Knowledge Base (Prolog)

```
% Facts
symptom(patient1, fever).
symptom(patient1, cough).
symptom(patient1, sore_throat).

symptom(patient2, fever).
symptom(patient2, rash).

% Rules (Knowledge)
disease(X, flu) :-
   symptom(X, fever),
   symptom(X, cough),
   symptom(X, sore_throat).

disease(X, measles) :-
   symptom(X, fever),
   symptom(X, rash).

% Query
```

```
diagnose(Patient, Disease) :-
    disease(Patient, Disease),
    write('Patient '), write(Patient),
    write(' has '), write(Disease), nl.
```

Step 2: Inference Engine

```
% Backward chaining (Prolog default)
query_diagnosis(Patient) :-
    diagnose(Patient, Disease).
```

Step 3: User Interaction

```
% Interactive diagnosis
interactive_diagnosis :-
    write('Enter patient name: '), read(Patient), nl,
    write('Does patient have fever? (yes/no): '), read(fever),
    write('Does patient have cough? (yes/no): '), read(cough),
    write('Does patient have sore throat? (yes/no): '), read(sore),

    % Add symptoms to KB
    (fever == yes -> assert(symptom(Patient, fever))),
    (cough == yes -> assert(symptom(Patient, cough))),
    (sore == yes -> assert(symptom(Patient, sore_throat))),

    % Diagnose
    diagnose(Patient, Disease).
```

Step 4: Execution

```
Query: ?- interactive_diagnosis.

Enter patient name: patient1
Does patient have fever? (yes/no): yes
Does patient have cough? (yes/no): yes
Does patient have sore throat? (yes/no): yes

Output: Patient patient1 has flu ✅
```

Extended Case Study: Car Diagnosis [web-general]

```
% Knowledge Base
problem(no_start, electrical) :-
    symptom(battery_dead),
    symptom(no_headlights).

problem(no_start, fuel) :-
    symptom(gas_empty),
    symptom(no_engine_sound).

problem(no_start, engine) :-
    symptom(engine_noise),
    symptom(overheating).

symptom(battery_dead) :-
    write('Headlights work? (yes/no): '), read(X), X == no.
symptom(gas_empty) :-
    write('Gas level? (low/normal): '), read(X), X == low.
symptom(no_engine_sound) :-
    write('Engine sound? (yes/no): '), read(X), X == no.
symptom(engine_noise) :-
    write('Engine noise? (yes/no): '), read(X), X == yes.
symptom(overheating) :-
    write('Engine hot? (yes/no): '), read(X), X == yes.

% Diagnosis
diagnose_car :-
    write('Car does not start. Investigating...'), nl,
    (problem(no_start, electrical) ->
        write('Problem: Electrical system (battery dead)'), nl)
    ;
    (problem(no_start, fuel) ->
        write('Problem: Fuel system (gas empty)'), nl)
    ;
    (problem(no_start, engine) ->
        write('Problem: Engine (overheating)'), nl)
    ;
    write('Problem unknown').
```

Execution:

```
Query: ?- diagnose_car.

Car does not start. Investigating...
Headlights work? (yes/no): no
Gas level? (low/normal): normal
Engine sound? (yes/no): no
Engine noise? (yes/no): no
Engine hot? (yes/no): no

Output: Problem: Electrical system (battery dead) ✅
```

Explanation System (Add reasoning)**:**

```
explain(problem(no_start, electrical)) :-
    write('Diagnosis: Electrical system'), nl,
    write('Reason: Headlights not working indicates battery'), nl,
    write('Solution: Replace battery').
```

---

3. CODE IMPLEMENTATIONS

Python: Neural Network (Simple)

```
import numpy as np

# Simple Neural Network
class NeuralNetwork:
    def __init__(self):
        self.weights = np.random.randn(3, 1)
        self.bias = np.random.randn(1, 1)

    def sigmoid(self, x):
        return 1 / (1 + np.exp(-x))

    def predict(self, x):
        z = np.dot(x, self.weights) + self.bias
        return self.sigmoid(z)

    def train(self, x, y, epochs=1000):
        for i in range(epochs):
```

```
        # Forward pass
        pred = self.predict(x)

        # Backward pass (gradient descent)
        error = y - pred
        gradient = np.dot(x.T, error * pred * (1 - pred))

        self.weights += 0.01 * gradient

# Training
X = np.array([[1, 0, 1],
              [0, 1, 1],
              [1, 1, 1]])
Y = np.array([[1], [0], [1]])

nn = NeuralNetwork()
nn.train(X, Y)

# Test
print("Prediction:", nn.predict(np.array([[1, 0, 1]])))  # Output: ~0.9
```

Python: Genetic Algorithm

```
import random

# Genetic Algorithm for Optimization
def fitness(x):
    return -(x - 5)**2 + 25  # Max at x=5

def genetic_algorithm(pop_size=100, generations=50):
    # Initialize population
    population = [random.uniform(0, 10) for _ in range(pop_size)]

    for gen in range(generations):
        # Evaluate fitness
        scores = [fitness(x) for x in population]

        # Select best (top 20%)
        best = sorted(zip(population, scores), key=lambda x: x[1], reverse=True)[:20]
        best = [x[0] for x in best]

        # Create new population
```

```
        new_population = best[:]
        while len(new_population) < pop_size:
            # Crossover
            parent1, parent2 = random.sample(best, 2)
            child = 0.5 * (parent1 + parent2)

            # Mutation
            if random.random() < 0.1:
                child += random.uniform(-1, 1)

            new_population.append(child)

        population = new_population

    # Return best solution
    best_x = max(population, key=fitness)
    return best_x, fitness(best_x)

# Run
x_opt, f_opt = genetic_algorithm()
print(f"Optimal x: {x_opt:.2f}, Fitness: {f_opt:.2f}")
# Output: Optimal x: ~5.0, Fitness: ~25.0 ✅
```

Python: Decision Tree (Induction)

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.datasets import load_iris

# Load data
iris = load_iris()
X = iris.data
y = iris.target

# Train
dt = DecisionTreeClassifier()
dt.fit(X, y)

# Test
prediction = dt.predict([[5.1, 3.5, 1.4, 0.2]])
print(f"Predicted class: {iris.target_names[prediction[0]]}")
# Output: setosa ✅
```

```
# Get rules
print(dt.export_text())
```

---

4. IMAGES & INFOGRAPHICS

[neural_network_architecture]

Neural Network Architecture: Shows input → hidden → output layers with weights and connections

[expert_system_architecture]

Expert System Components: Visual diagram of knowledge base, inference engine, user interface

[genetic_algorithm_flow]

Genetic Algorithm Flow: Population → Selection → Crossover → Mutation → New generation

[induction_process]

Inductive Learning: Training examples → Pattern detection → Rule extraction → Hypothesis

---

5. REFERENCE SOURCES

📚 Books (From Syllabus)

Book	Code	Chapters	Topics
Artificial Intelligence: A Modern Approach	TB1	Ch 17	Learning (all types)
Artificial Intelligence	TB2	Ch 18-19	Expert Systems, Prolog

🌐 Websites

Website	URL	Topics
General AI Learning	Search: "types of learning AI"	Complete learning taxonomy
Expert Systems	Search: "expert system architecture AI"	Components, applications
Prolog Tutorial	Search: "Prolog expert system tutorial"	Syntax, examples
Neural Networks	Search: "neural network tutorial Python"	Architecture, training
Genetic Algorithms	Search: "genetic algorithm tutorial"	Selection, crossover, mutation

📽️ YouTube Videos

Video	Search Term	Duration	Topics
Learning in AI	"types of learning artificial intelligence"	20 min	Rote, induction, neural networks
Expert Systems	"expert system AI tutorial"	25 min	Architecture, MYCIN case study
Neural Networks	"neural network tutorial for beginners"	30 min	Architecture, backpropagation
Genetic Algorithms	"genetic algorithm tutorial AI"	15 min	Evolution, optimization
Prolog Expert System	"Prolog expert system medical diagnosis"	20 min	Knowledge base, inference

📖 Additional Resources

- **MYCIN Case Study**: Search "MYCIN expert system medical diagnosis"
- **Prolog Documentation**: Search "Prolog official tutorial"

- **Neural Networks (TensorFlow)**: Search "TensorFlow neural network tutorial"
- **Genetic Algorithms (Python)**: Search "Python genetic algorithm library"

🧪 Interactive Tools

- **Neural Network Playground**: tensorflow playground
- **Prolog Interpreter**: Search "online Prolog interpreter"
- **Genetic Algorithm Simulator**: Search "genetic algorithm visualizer"
- **Decision Tree Builder**: Search "decision tree online tool"

✅ QUICK RECAP: Key Concepts

Learning Types Summary

Type	Speed	Understanding	Example
Rote	Fast	None	Memorization
Advice	Fast	Low	Expert instructions
Induction	Slow	High	Decision trees
EBL	Fast	High	Single example
Neural	Slow	Medium	Deep learning
Genetic	Slow	Medium	Optimization

Expert System Formula

```
Expert System = Knowledge Base + Inference Engine + User Interface
```

🎯 EXAM TIPS

Topic	What to Memorize
Learning Types	9 types + 1 example each
Rote Learning	Memorization vs understanding
Induction	Examples → Rules (decision tree)
EBL	Single example + explanation
Neural Networks	Architecture (input→hidden→output)
Genetic Learning	Selection + Crossover + Mutation
Expert System	5 components + architecture
Prolog Example	Knowledge base + query syntax

📊 SUMMARY TABLE: UNIT-IV Coverage

Section	Topics	Hours	Key Concepts
Learning	Intro, rote, advice, problem-solving, induction, EBL, discovery, analogy, neural, genetic	~6 hrs	9 learning types, neural architecture, GA process
Expert Systems	Intro, case study (Prolog)	~4 hrs	5 components, MYCIN, Prolog syntax

Total: 10 hours ✅

Study Recommendation:

1. Learn all 9 learning types with examples
2. Study neural network architecture thoroughly
3. Understand genetic algorithm steps
4. Memorize expert system architecture
5. Practice Prolog code (knowledge base + queries)
6. Review exam tips before test

Good luck! 🚀