national center for ontological research
ncor

University at Buffalo
The State University of New York

Machine Learning with Ontologies

John Beverley

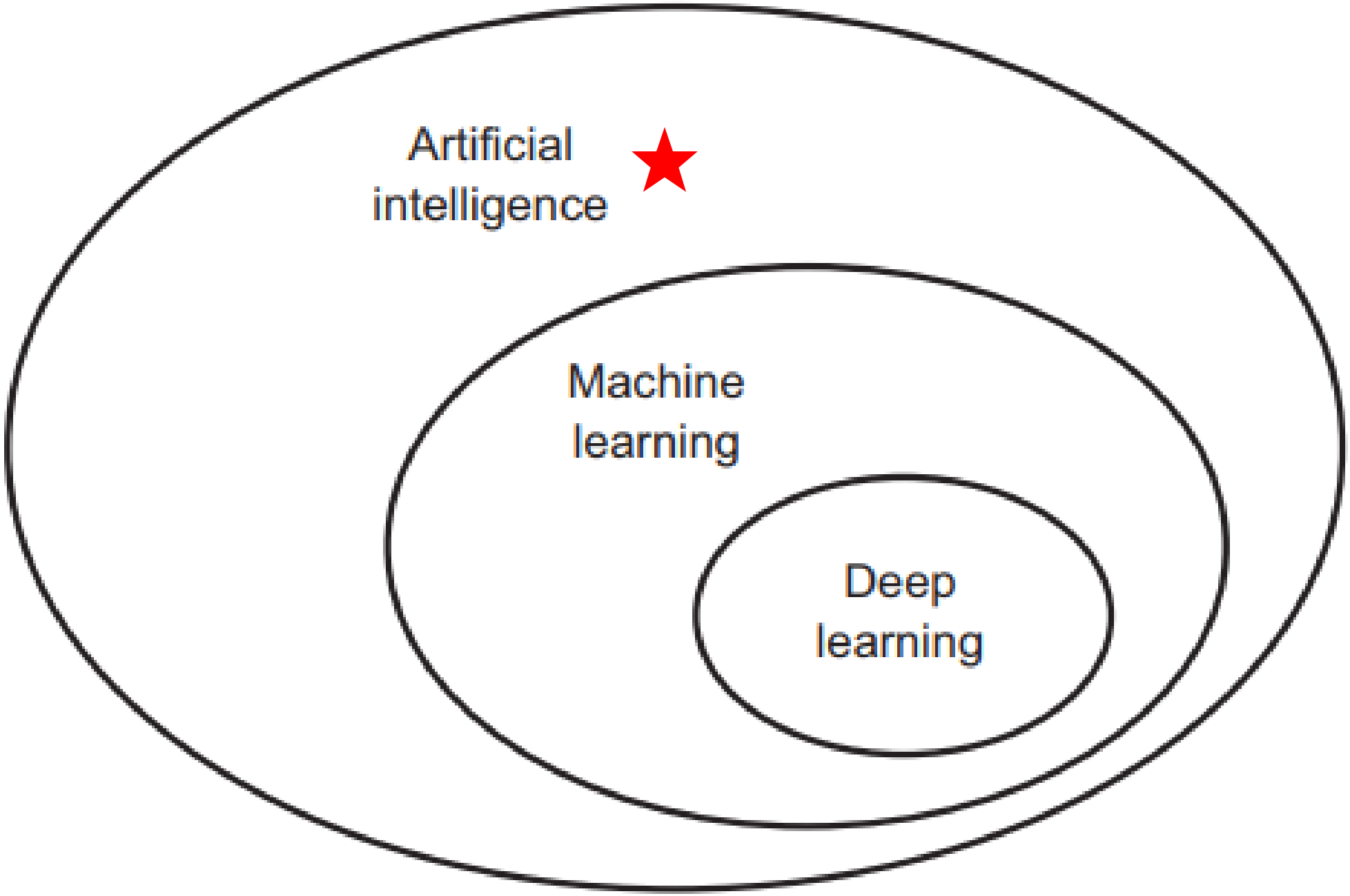
Assistant Professor, *University at Buffalo*
Co-Director, *National Center for Ontological Research*
Affiliate Faculty, *Institute of Artificial Intelligence and Data Science*

Outline

- Foundations of Modern AI
- Machine Learning
- Deep Learning
- MOWL

Outline

- Foundations of Modern AI
- Machine Learning
- Deep Learning
- MOWL


Artificial
intelligence
Machine
learning
Deep
learning

WHAT IS INTELLIGENCE?

Spearman

- Spearman popularized the modern definition in *General Intelligence Objectively Determined and Measured*

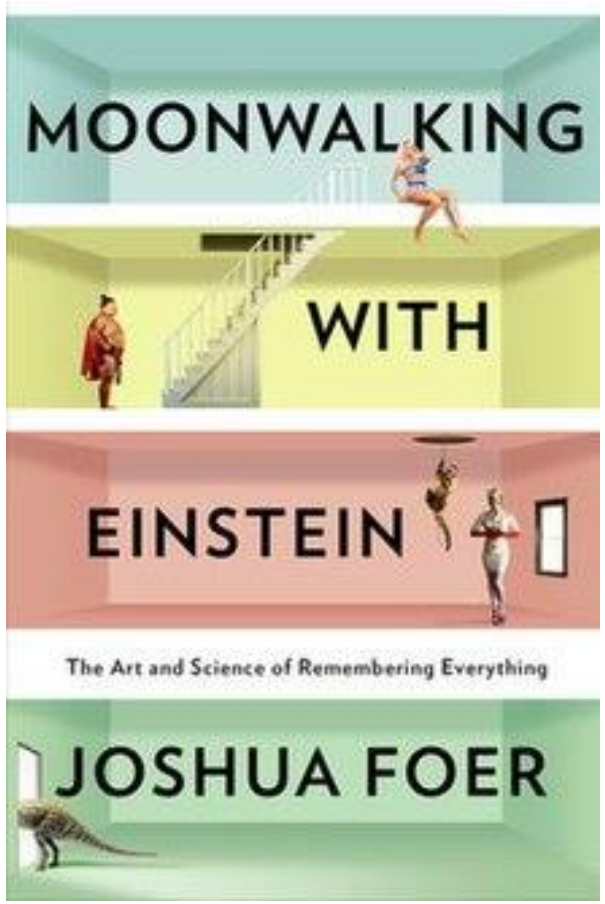
- He showed test scores are correlated across many subjects

- Spearman proposed "general intelligence" as a unifying faculty

Spearman's correlation matrix for six measures of school performance. All the correlations are positive, the *positive manifold* phenomenon. The bottom row shows the *g* loadings of each performance measure.[7]

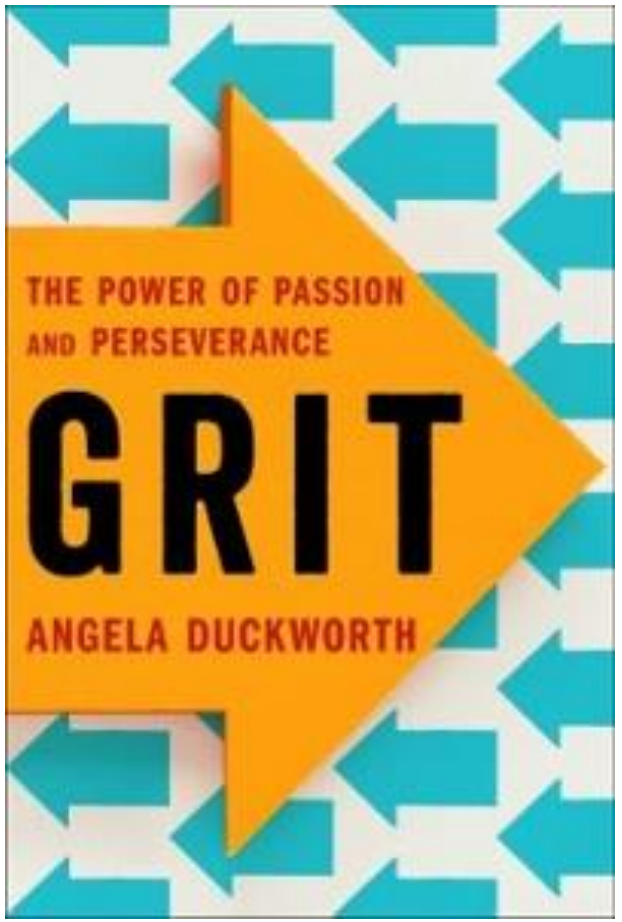
	Classics	French	English	Math	Pitch	Music
Classics	–					
French	.83	–				
English	.78	.67	–			
Math	.70	.67	.64	–		
Pitch discrimination	.66	.65	.54	.45	–	
Music	.63	.57	.51	.51	.40	–
g	.958	.882	.803	.750	.673	.646

Trained Intelligence



It is possible to **increase your score** on tests of general intelligence

Perseverance for long-term goals is not correlated with general intelligence **but is a better predictor of long-term success**



WHAT IS ARTIFICIAL INTELLIGENCE?

Defining AI

"[The automation of] activities that we associate with human thinking, activities such as decision making, problem solving, learning" **Bellman, 1978**

"The study of mental faculties through the use of computational models"
Charniak & McDermott, 1985

"The exciting new effort to make computers think *machines with minds*, in the full and literal sense" **Haugeland, 1985**

"A field of study that seeks to explain and emulate intelligent behavior in terms of computational processes"
Schalkoff, 1990

Defining AI

"The art of creating machines that perform functions that require intelligence when performed by people" **Kurzweil, 1990**

"The study of the computations that make it possible to perceive, reason, and act" **Winston, 1992**

"The branch of computer science that is concerned with the automation of intelligent behavior" **Luger & Stubblefield, 1993**

"The design and study of computer programs that behave intelligently. These programs are constructed to perform as would a human or an animal whose behavior we consider intelligent" **Dean et al., 1995**

What is "Artificial Intelligence"

- AI is a multifaceted phenomenon, much like intelligence

- Appears to include:
  - Ability to learn
  - Ability to plan/reason
  - Ability to communicate
  - Ability to reason probabilistically

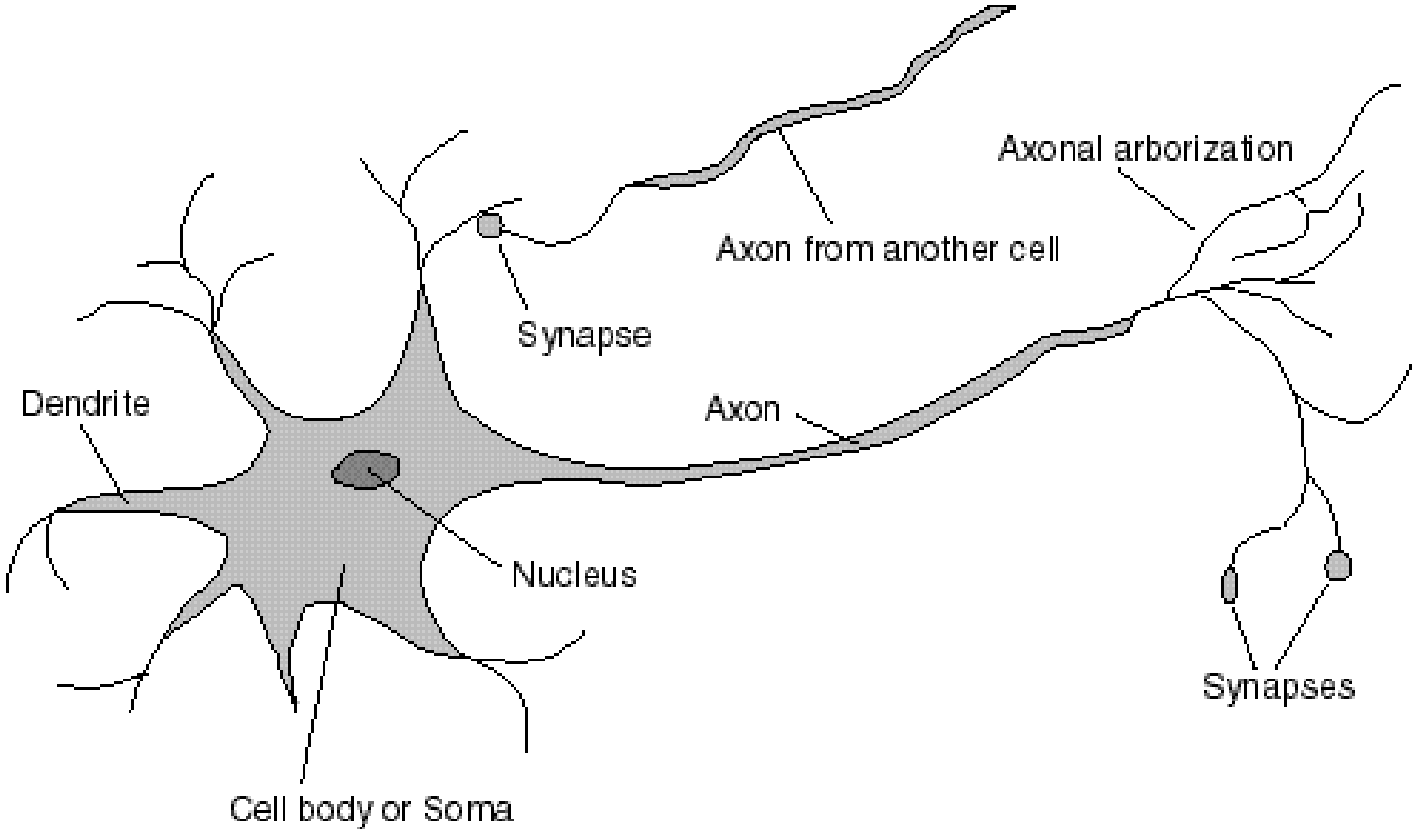
What is "Artificial Intelligence"

- AI is a multifaceted phenomenon, much like intelligence

- Appears to include:
  - **Ability to learn**
  - Ability to plan/reason
  - Ability to communicate
  - Ability to reason probabilistically

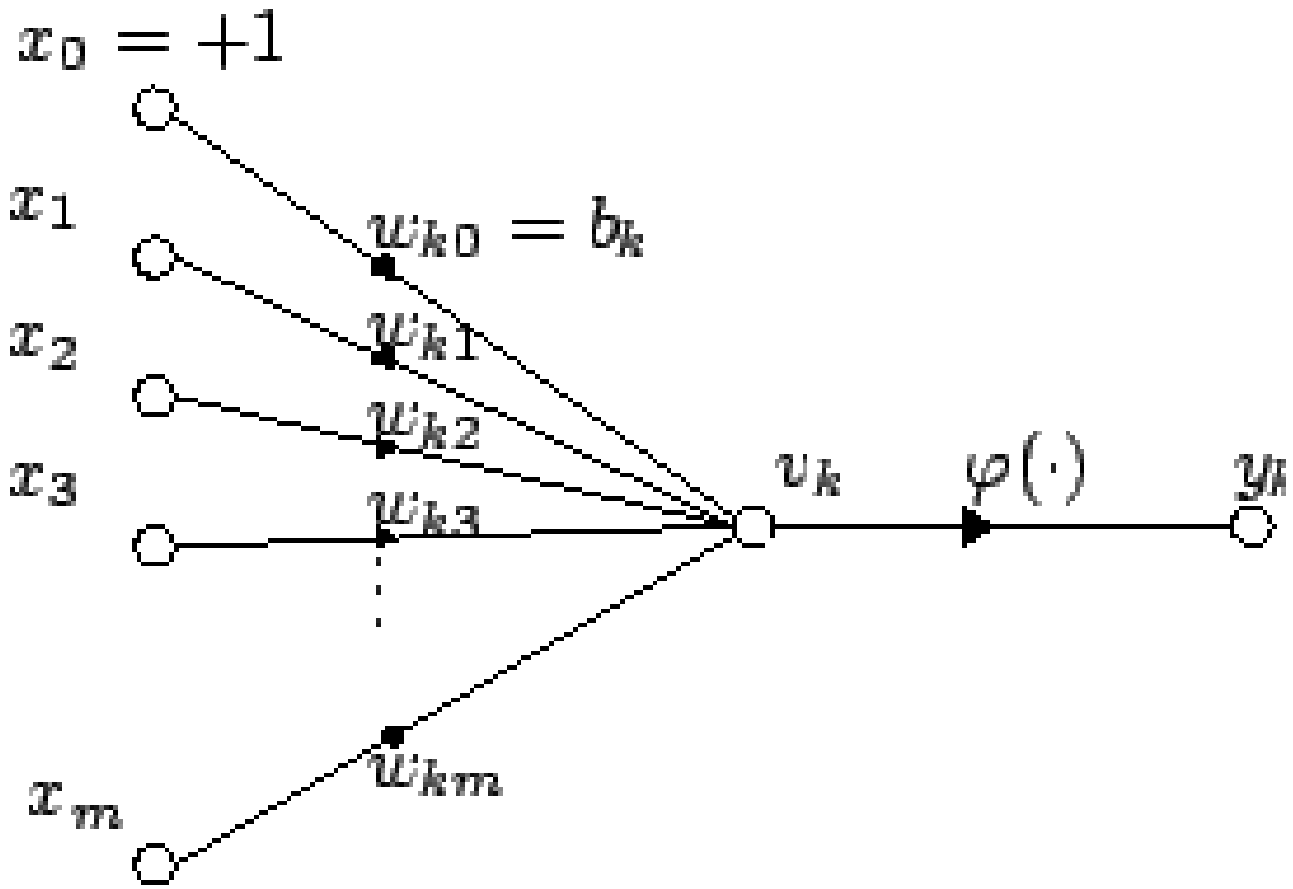
Ability to Learn

- AI **imitates neuroscience**
  - Basis is a neuron 10^{11}
  - Connected by snapses 10^{14}
  - Cycle time 10^{-3} second

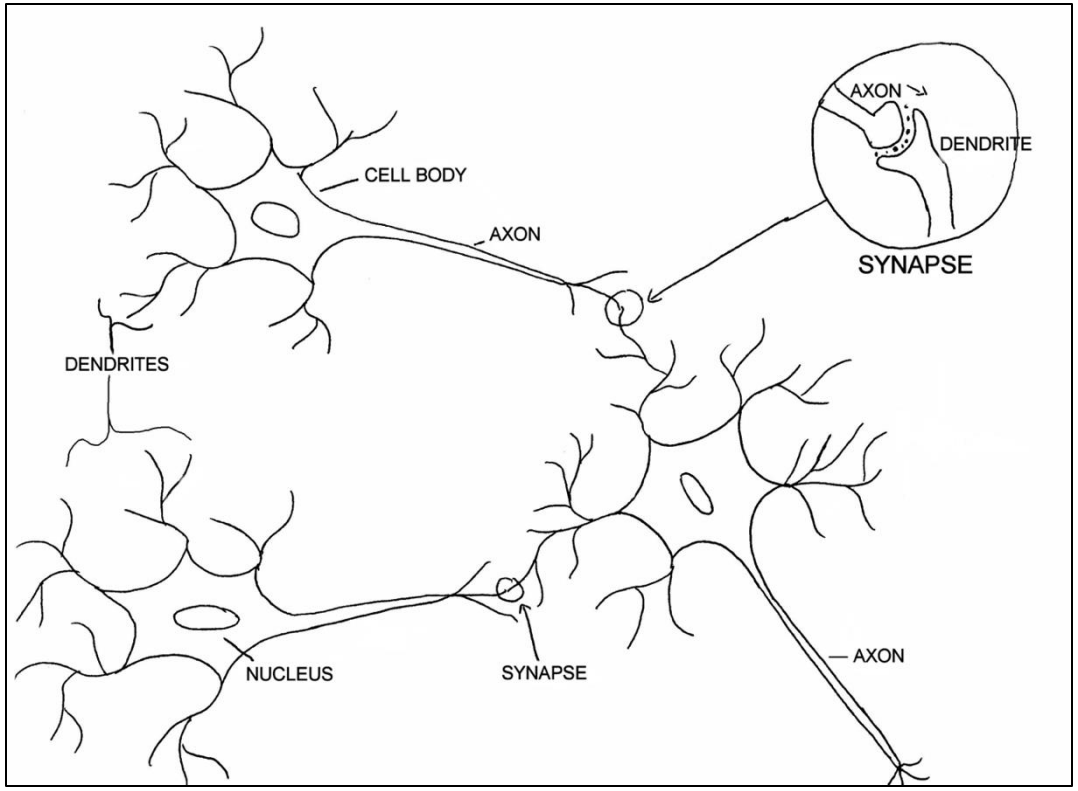


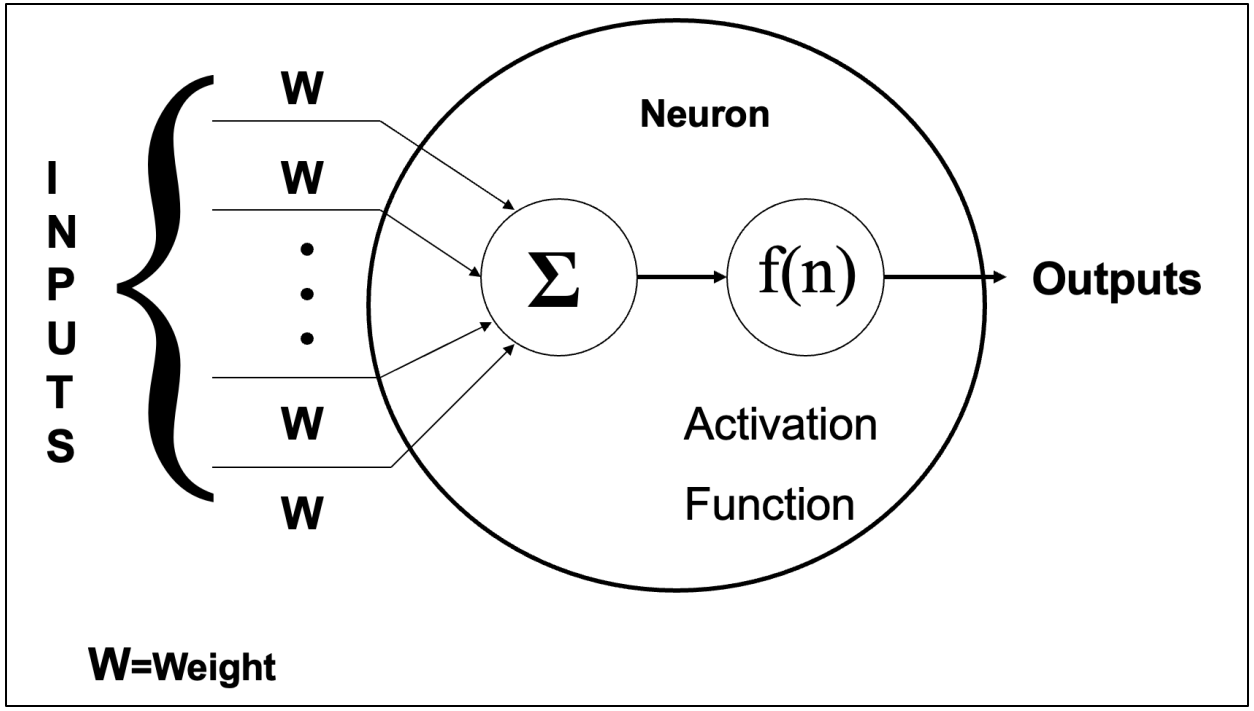
Ability to Learn

- AI **imitates neuroscience**
- **McCulloch-Pitts Neuron Model**
- "Deep Learning" is more modern strategy



Neurons


AXON
DENDRITE
CELL BODY
AXON
SYNAPSE
DENDRITES
NUCLEUS
SYNAPSE
AXON


INPUTS
W
W
W
W
Neuron
Σ
f(n)
Outputs
Activation
Function
W=Weight

Biological Neuron

- Each neuron has a **body**, an **axon**, and many **dendrites**
  - Can be in one of the two states: firing and rest
  - Neuron fires if the total incoming stimulus exceeds the threshold
- **Synapses** are thin gaps between axons and dendrites
  - They facilite signal exchange
  - They exhibit varying degrees of strength/efficiency

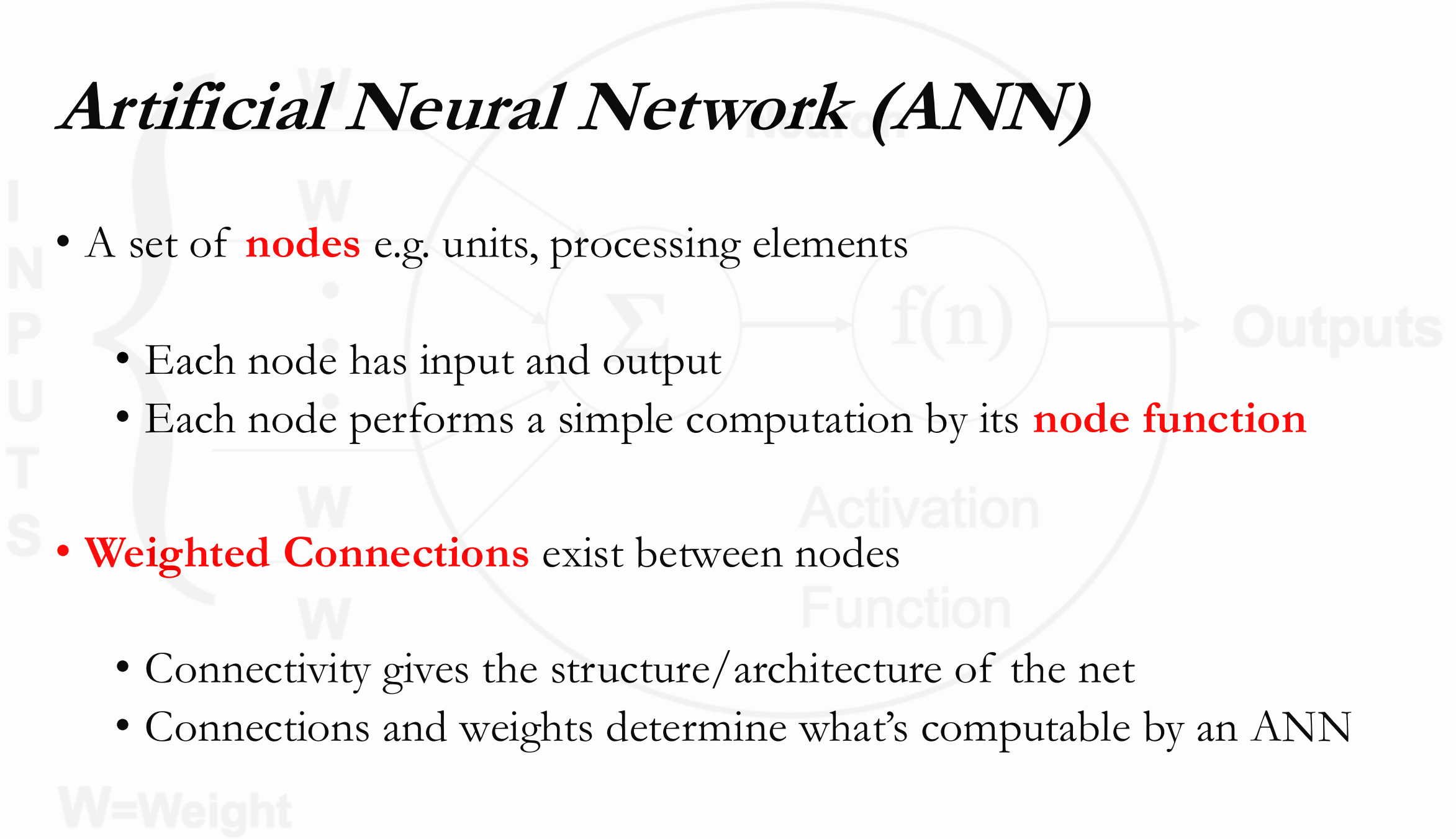
Computational Learning

- The **perceptron** was an early **artificial neuron**

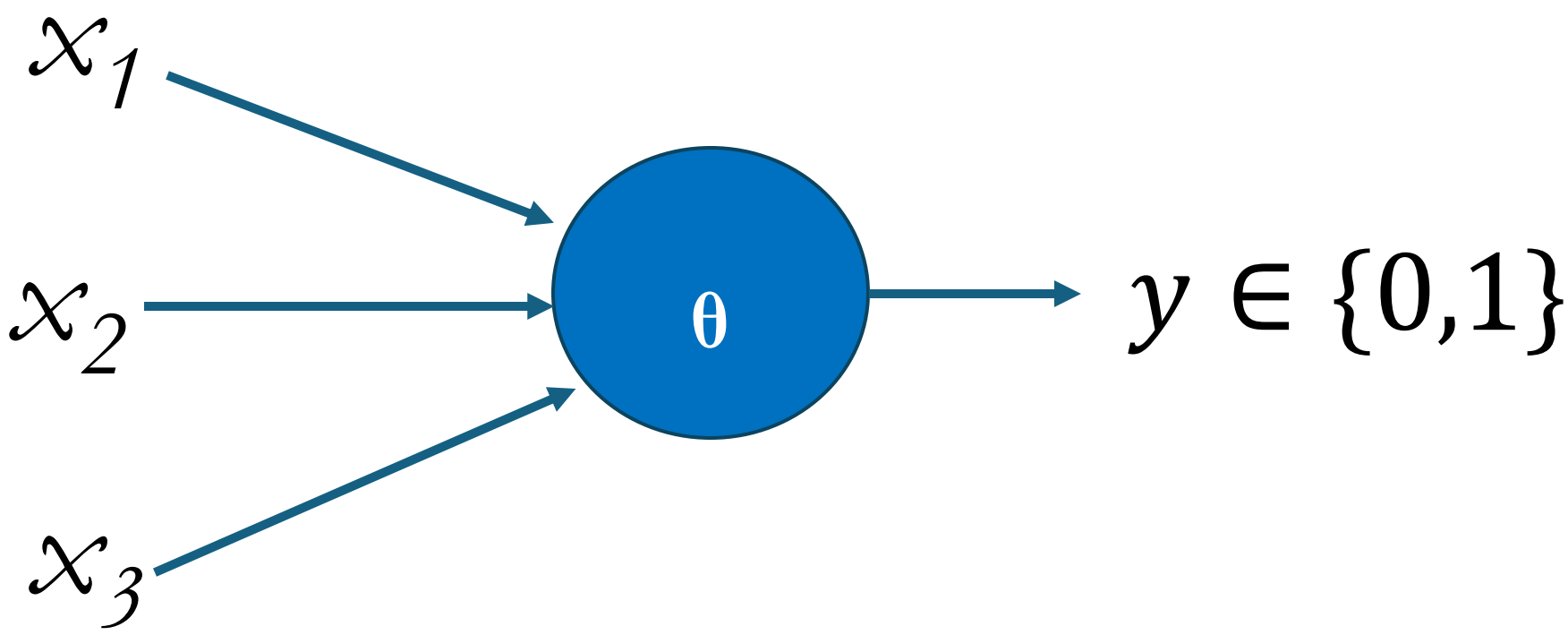
- This was inspired by **connectionism**—the idea that intelligence emerges from networks of simple connected units rather than by symbolic manipulation

Artificial Neural Network (ANN)

- A set of **nodes** e.g. units, processing elements
  - Each node has input and output
  - Each node performs a simple computation by its **node function**
- **Weighted Connections** exist between nodes
  - Connectivity gives the structure/architecture of the net
  - Connections and weights determine what's computable by an ANN

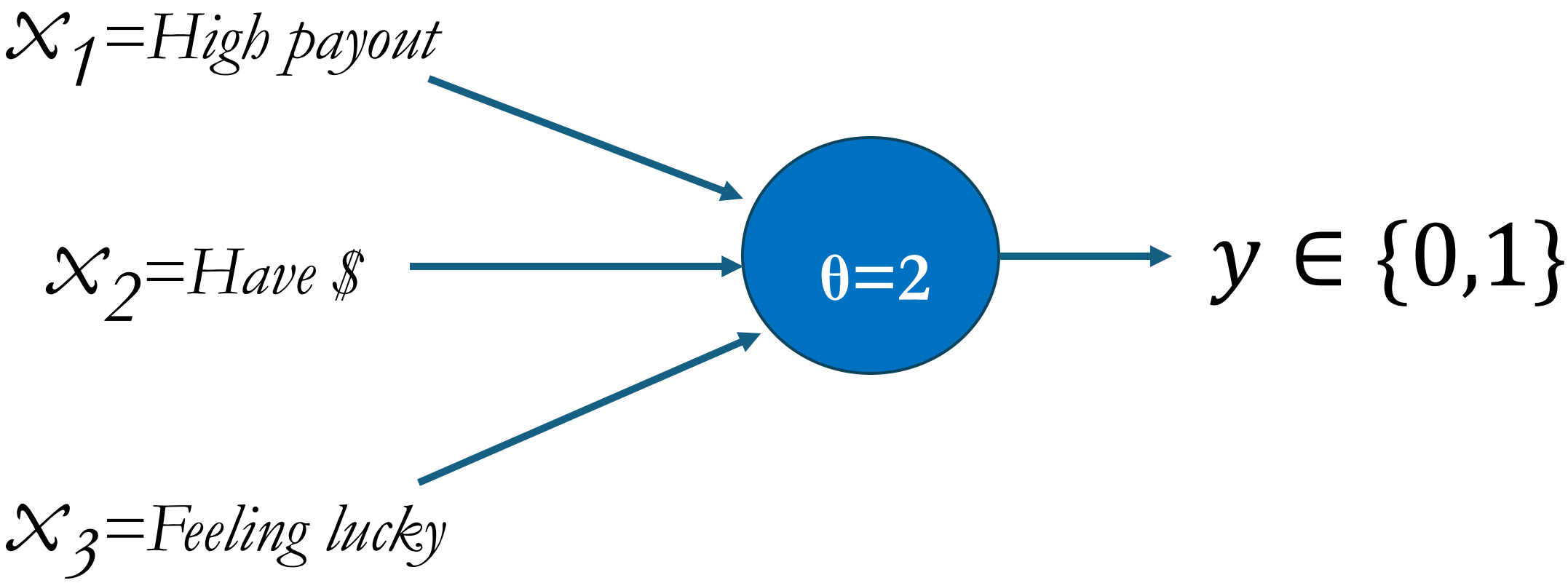


Inputs x_1, x_2, x_3 can be 0 or 1 and are inputs to the neuron



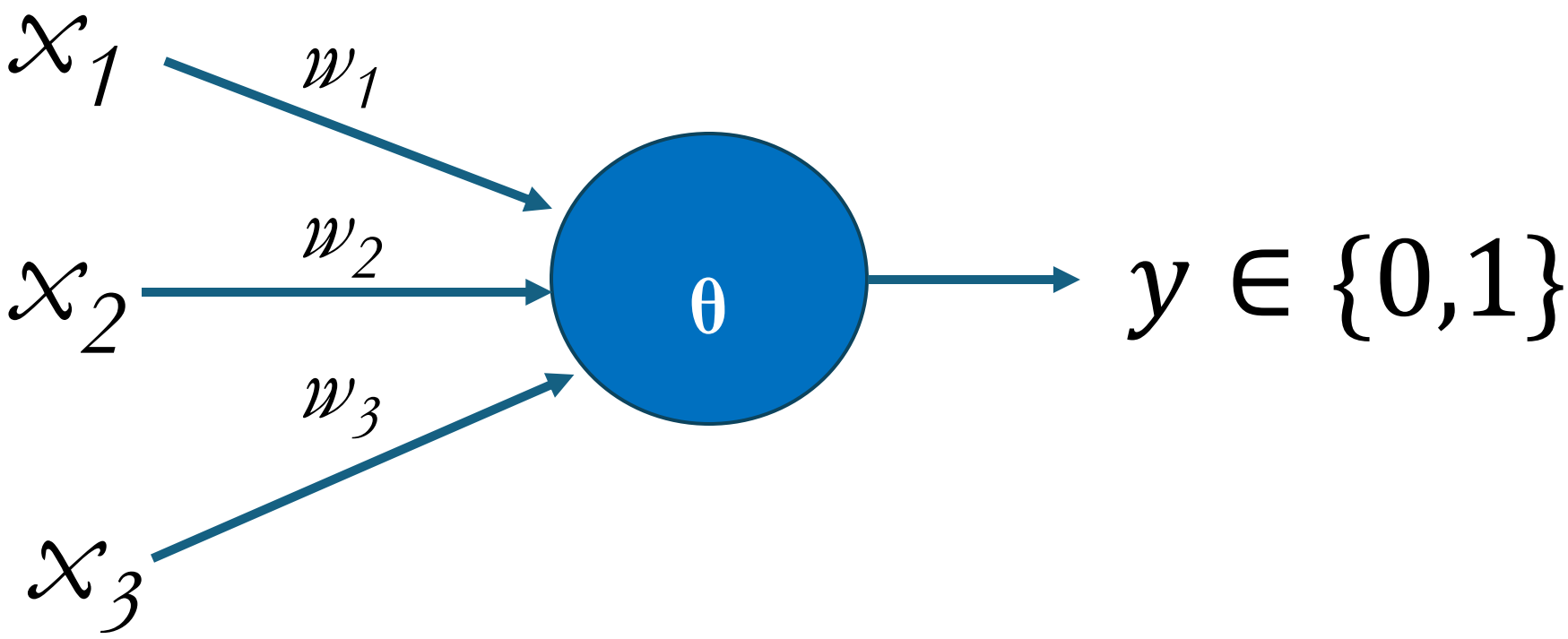
Neuron will "fire" a 1 if the sum of the three inputs > the threshold θ

Will I buy a lottery ticket?



Neuron fires "1" if the sum of inputs > 2, i.e. if any 2 conditions are met, I will buy a ticket

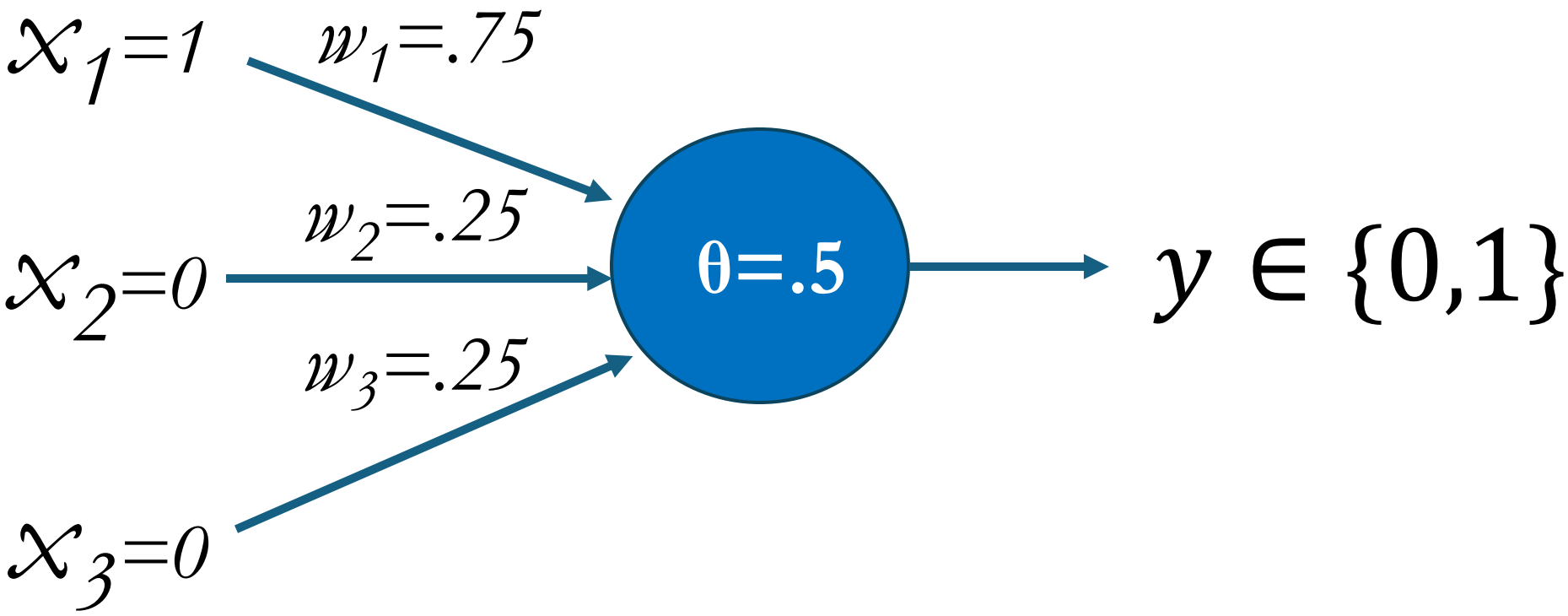
Will I buy a lottery ticket?



Because some decisions are worth more than others use the weighted sum:

$$w_1x_1 + w_2x_2 \geq \theta$$

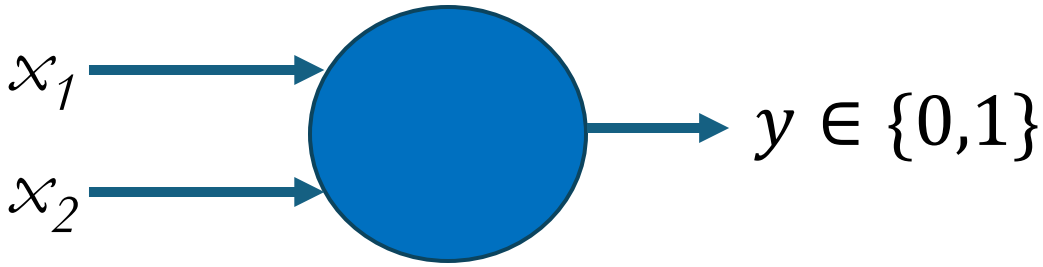
Will I buy a lottery ticket?



$.75*1+.25*0+.25*0=0.75 \geq .5$

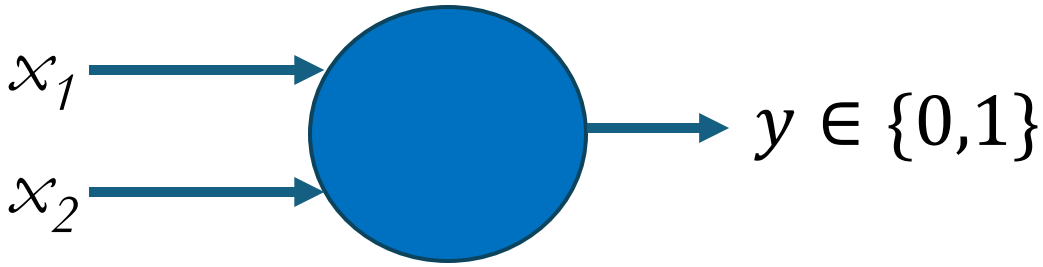
Hence, $y=1$

OR GATE



OR function: Output y=1 if x_1=1 or x_2=1

OR GATE

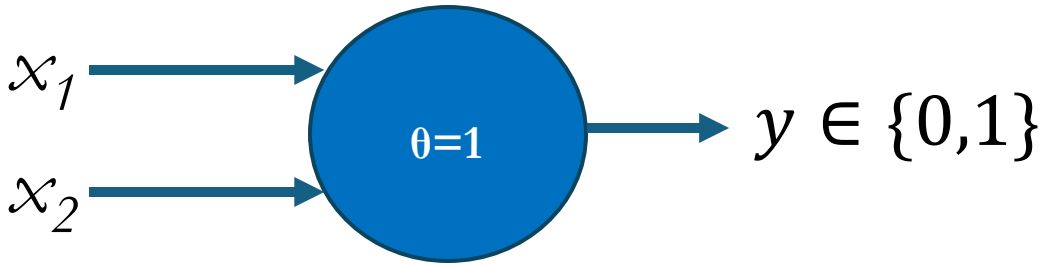


X_1	X_2	$X_1 + X_2$	y
0	0	0	0
1	0	1	1
1	1	2	1
0	1	1	1

OR GATE

Rows 2-4 result in an output of 1.

Sum column suggests we should set the threshold θ=1 since values of 1 or 2 make the OR function true.



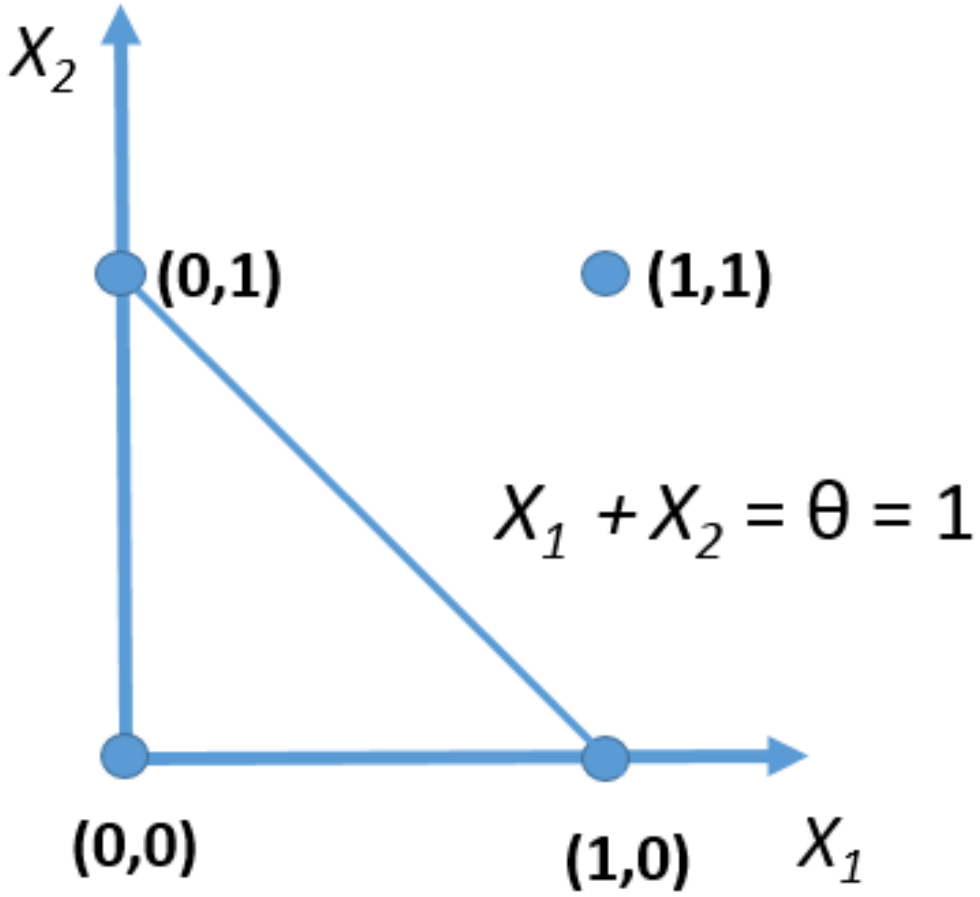
X_1	X_2	$X_1 + X_2$	y
0	0	0	0
1	0	1	1
1	1	2	1
0	1	1	1

OR GATE

Each combination of inputs can be plotted

Points on or above the boundary line produce output of 1

X_1	X_2	$X_1 + X_2$	y
0	0	0	0
1	0	1	1
1	1	2	1
0	1	1	1

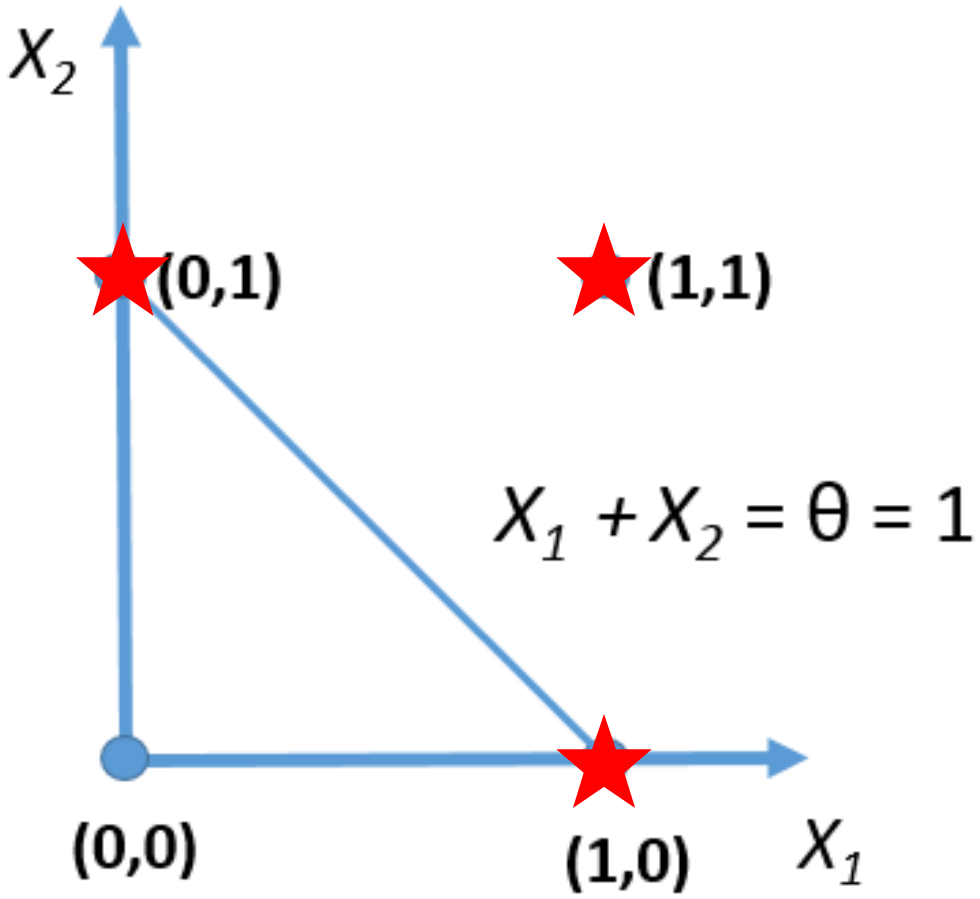


OR GATE

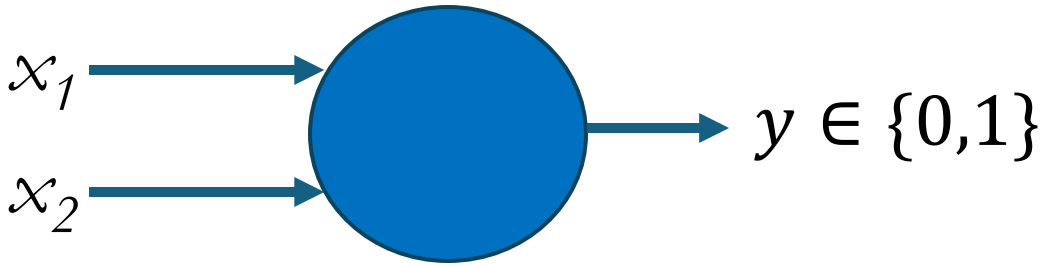
Each combination of inputs can be plotted

Points on or above the boundary line produce output of 1

X_1	X_2	$X_1 + X_2$	y
0	0	0	0
1	0	1	1
1	1	2	1
0	1	1	1

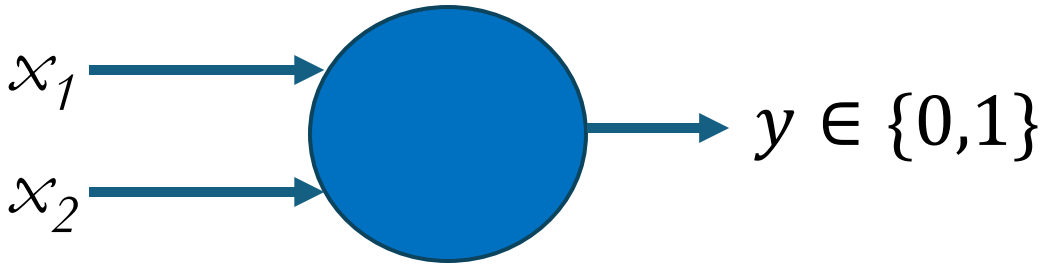


AND GATE



OR function: Output y=1 if x_1=1 and x_2=1

AND GATE

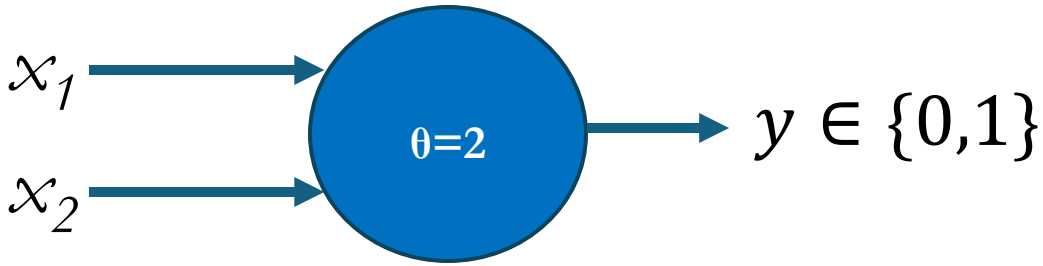


X_1	X_2	$X_1 + X_2$	y
0	0	0	0
1	0	1	0
1	1	2	1
0	1	1	0

AND GATE

True only when both inputs equal 1

Suggests threshold should be θ=2



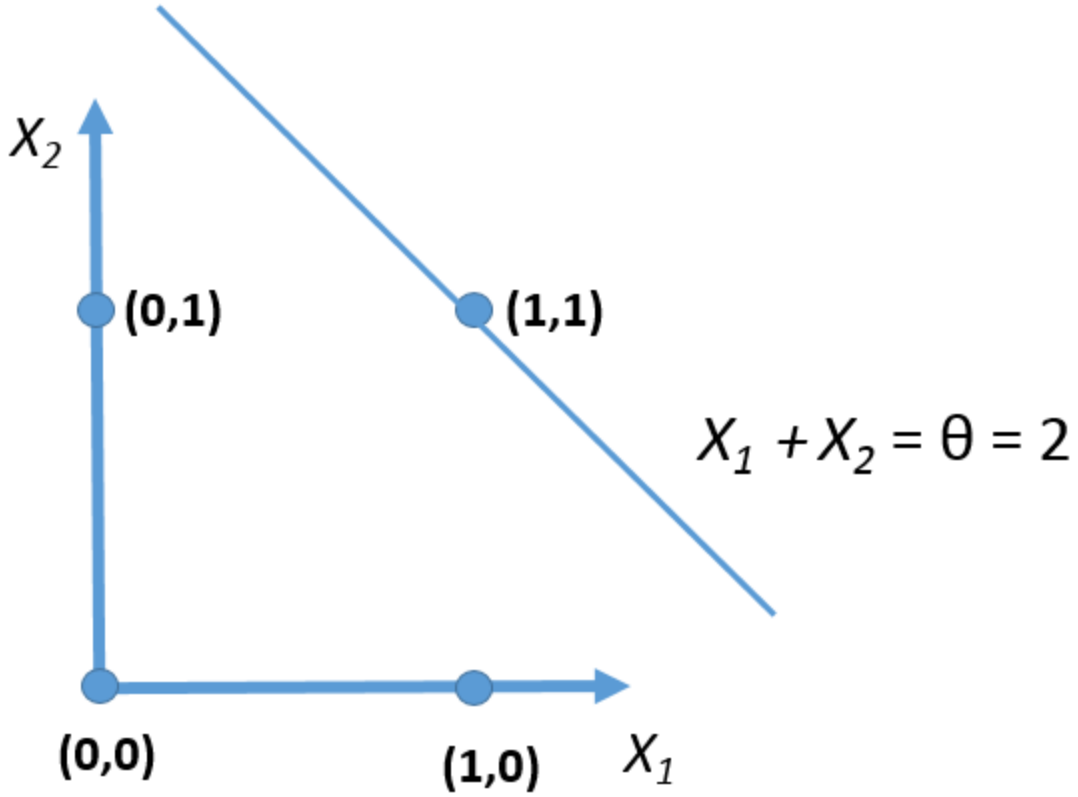
X_1	X_2	X_1+X_2	y
0	0	0	0
1	0	1	0
1	1	2	1
0	1	1	0

AND GATE

Each combination of inputs can be plotted

Notice the threshold is 2 in this case

X_1	X_2	X_1+X_2	y
0	0	0	0
1	0	1	0
1	1	2	1
0	1	1	0

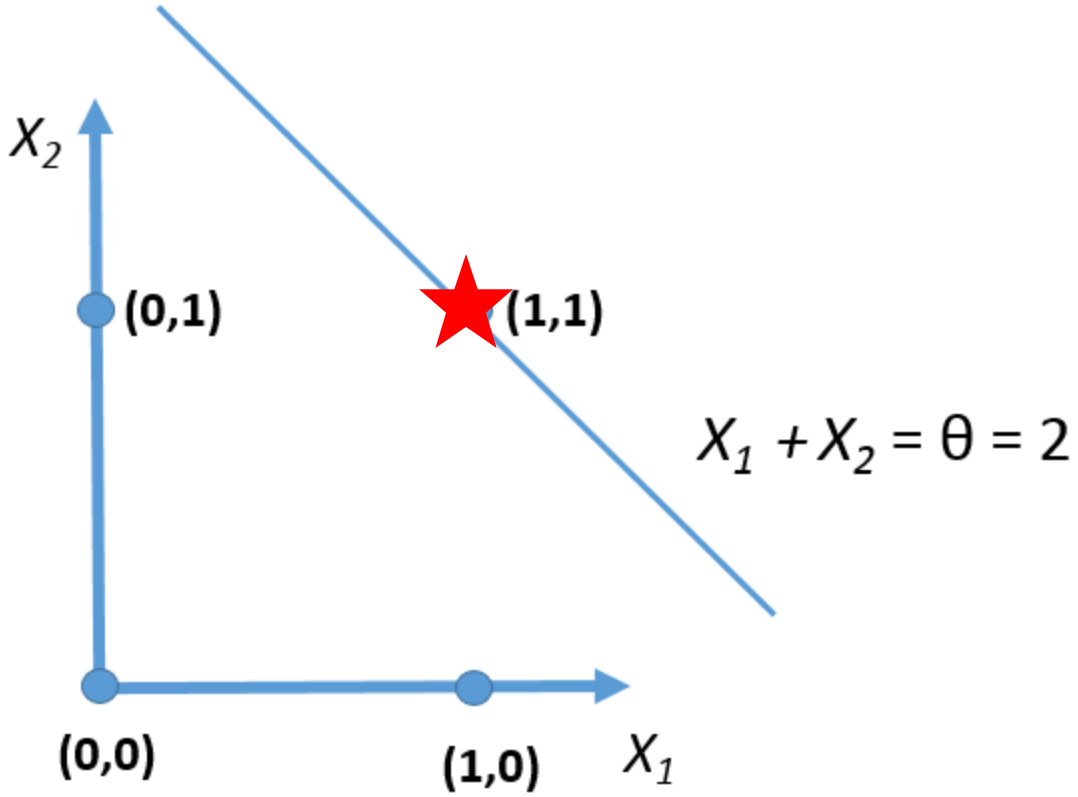


AND GATE

Each combination of inputs can be plotted

Notice the threshold is 2 in this case

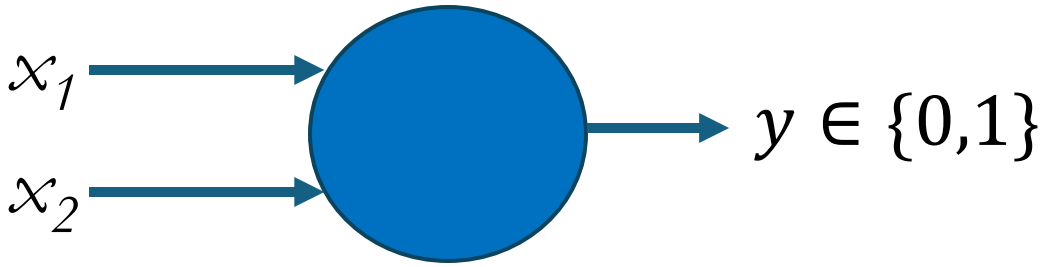
X_1	X_2	X_1+X_2	y
0	0	0	0
1	0	1	0
1	1	2	1
0	1	1	0



Minsky's Critique

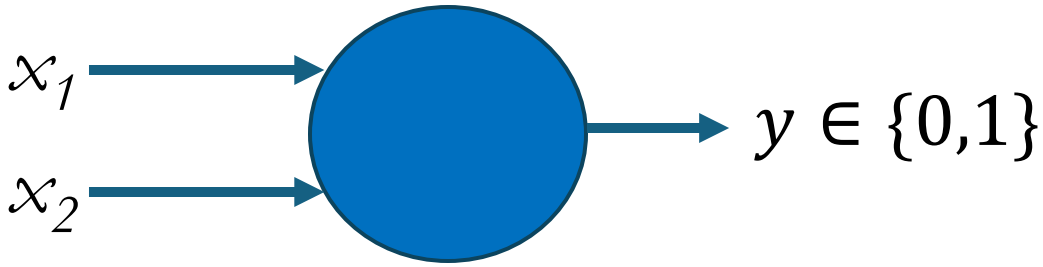
SINGLE LAYER PERCEPTRONS CANNOT LEARN SIMPLE FUNCTIONS SUCH AS XOR

XOR GATE



OR function: Output y=1 if x_1=1 or x_2=1 but not both

XOR GATE

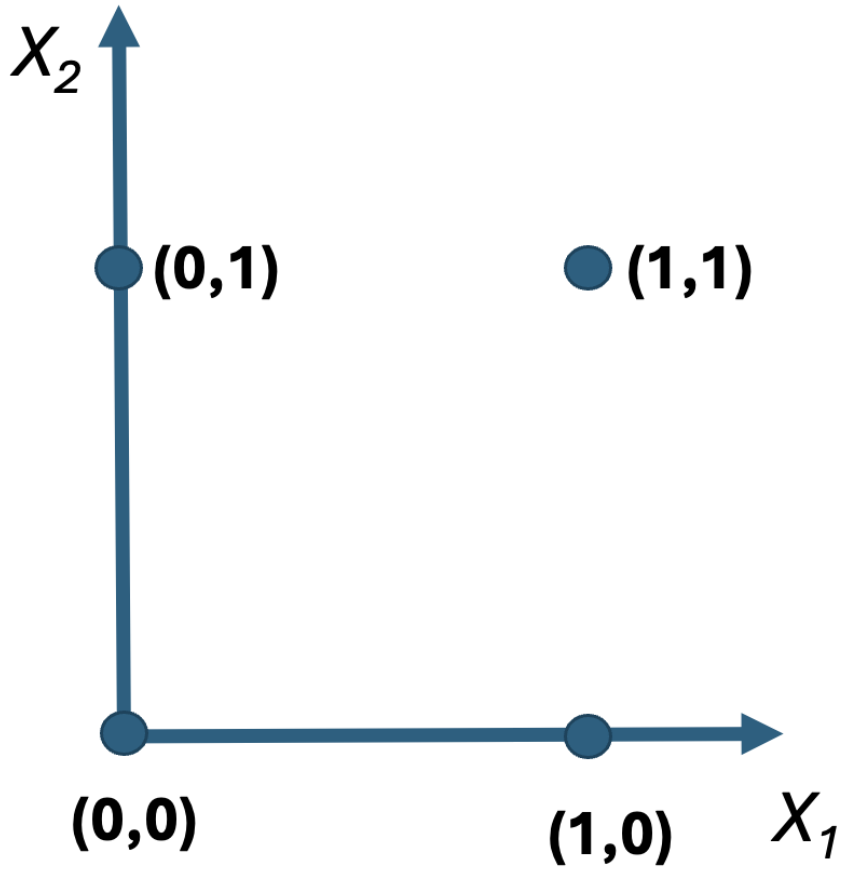


X_1	X_2	X_1+X_2	y
0	0	0	0
1	0	1	1
1	1	2	0
0	1	1	1

XOR GATE

Can one draw a single line which separates 1 and 0 outputs?

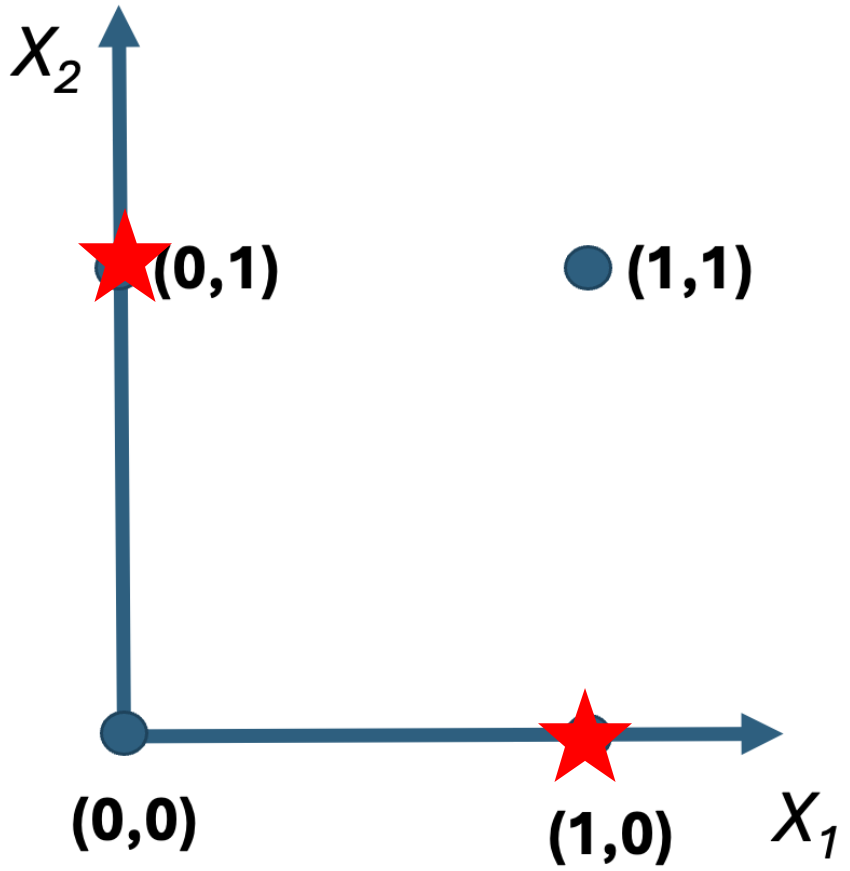
X_1	X_2	$X_1 + X_2$	y
0	0	0	0
1	0	1	1
1	1	2	0
0	1	1	1



XOR GATE

Can one draw a single line which separates 1 and 0 outputs?

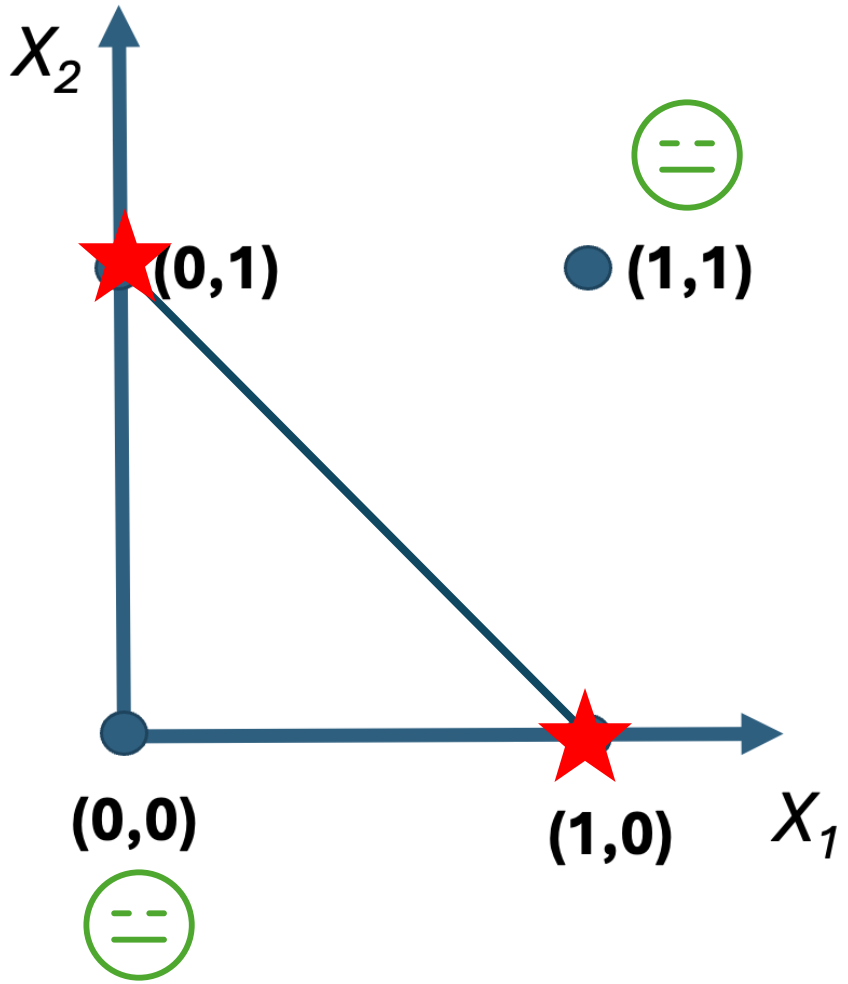
X_1	X_2	$X_1 + X_2$	y
0	0	0	0
1	0	1	1
1	1	2	0
0	1	1	1



XOR GATE

Can one draw a single line which separates 1 and 0 outputs?

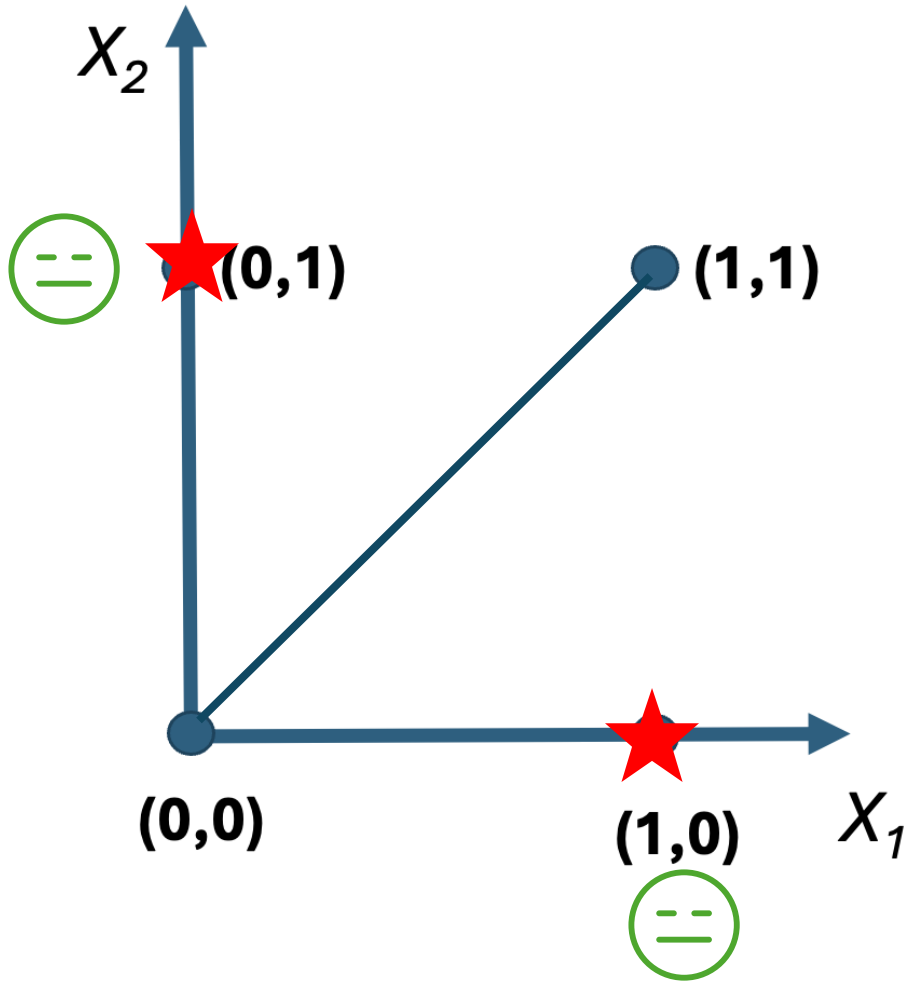
X_1	X_2	$X_1 + X_2$	y
0	0	0	0
1	0	1	1
1	1	2	0
0	1	1	1



XOR GATE

Can one draw a single line which separates 1 and 0 outputs?

X_1	X_2	$X_1 + X_2$	y
0	0	0	0
1	0	1	1
1	1	2	0
0	1	1	1



What is "Artificial Intelligence"

- AI is a multifaceted phenomenon, much like intelligence

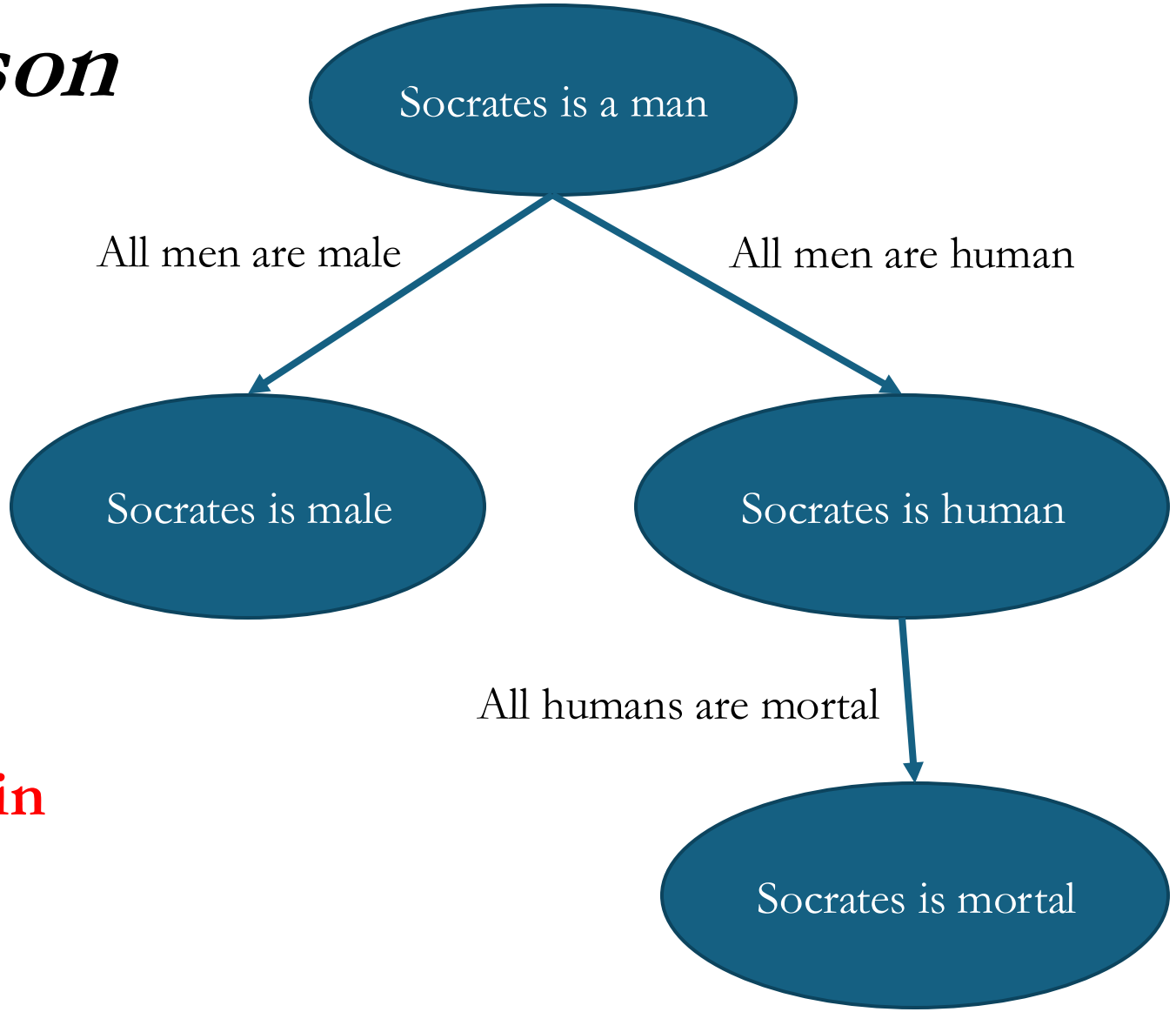
- Appears to include:
  - Ability to learn
  - **Ability to plan/reason**
  - Ability to communicate
  - Ability to reason probabilistically

Ability to Plan/Reason

Logic Theorist

1. Initial hypotheses are root of a search tree
2. Branches are logical deduction
3. Theorem to be proven is goal

Proved 38 of the first 52 theorems in Chapter 2 of the *Principia Mathematica*



Styles of Automated Reasoning

- Logicians have for decades leveraged automated theorem provers and model checkers to explore logical space

- Most run by leveraging **resolution**, or **unification**, or **tableau algorithms**

Prover9

```
============================== PROOF =========== 

% -------- Comments from original proof -------
% Proof 1 at 0.00 (+ 0.03) seconds.
% ngth of proof is 7.
% el of proof is 3.
% imum clause weight is 2.
% en clauses 2.

> Q # label(non_clause).  [assumption].
 label(non_clause) # label(goal).  [goal].
3 -P | Q.  [clausify(1)].
4 P.  [assumption].
5 -Q.  [deny(2)].
6 Q.  [ur(3,a,4,a)].
7 $F.  [resolve(6,a,5,a)].

============================== end of proof ===
```

Practical Reasoning for Expressive Description Logics

Ian Horrocks[1], Ulrike Sattler[2], and Stephan Tobies[2]

[1] Department of Computer Science, University of Manchester†
[2] LuFG Theoretical Computer Science, RWTH Aachen‡

Abstract. Description Logics (DLs) are a family of knowledge representation formalisms mainly characterised by constructors to build complex concepts and roles from atomic ones. Expressive role constructors are important in many applications, but can be computationally problematical. We present an algorithm that decides satisfiability of the DL $\mathcal{ALC}$ extended with transitive and inverse roles, role hierarchies, and qualify-

HermiT OWL Reasoner

The New Kid on the OWL Block

Reasoning Algorithms

Davis-Putnam Algorithm – Recursively assign truth-values to variables of a proposition, if reach a contradiction then backtrack and try new assignment, terminate when assignment is found for all variables or all possibilities are exhausted

Analytic Tableau - Construct a tree where each branch represents an interpretation of a proposition, decompose proposition by extended branches with new interpretations on each, close branch if reach a contradiction, terminate when all branches closed or one full branch open

Reasoning Algorithms

Davis-Putnam Algorithm – Recursively assign truth-values to variables of a proposition, if reach a contradiction then backtrack and try new assignment, terminate when assignment is found for all variables or all possibilities are exhausted

Analytic Tableau - Construct a tree where each branch represents an interpretation of a proposition, decompose proposition by extended branches with new interpretations on each, close branch if reach a contradiction, terminate when all branches closed or one full branch open

Reasoner	Algorithm	Strengths	Weaknesses	Use Cases
HermiT	Hypertableau	Supports complex DL reasoning	High memory usage; slower on large ontologies	Highly expressive ontologies with complex DL features
Pellet	Tableau	SWRL support, datatype reasoning	Slower on large ontologies; performance can degrade	Reasoning with rules (SWRL), expressive DL ontologies
Fact++	Optimized Tableau	Fast for moderately complex ontologies	Not ideal for very large ontologies	OWL DL reasoning with moderate complexity
ELK	OWL 2 EL optimized	Extremely fast for large OWL 2 EL ontologies	Limited to OWL 2 EL; no inverse roles	Large-scale biomedical ontologies (e.g., SNOMED CT)

Tableau

The $\rightarrow_{\sqcap}$-rule

***Condition*:** $\mathcal{A}$ contains $(C_1 \sqcap C_2)(x)$, but not both $C_1(x)$ and $C_2(x)$.

***Action*:** $\mathcal{A}' := \mathcal{A} \cup \{C_1(x), C_2(x)\}$.

The $\rightarrow_{\sqcup}$-rule

***Condition*:** $\mathcal{A}$ contains $(C_1 \sqcup C_2)(x)$, but neither $C_1(x)$ nor $C_2(x)$.

***Action*:** $\mathcal{A}' := \mathcal{A} \cup \{C_1(x)\}$, $\mathcal{A}'' := \mathcal{A} \cup \{C_2(x)\}$.

The $\rightarrow_{\exists}$-rule

***Condition*:** $\mathcal{A}$ contains $(\exists r.C)(x)$, but there is no individual name z such that $C(z)$ and $r(x, z)$ are in $\mathcal{A}$.

***Action*:** $\mathcal{A}' := \mathcal{A} \cup \{C(y), r(x, y)\}$ where y is an individual name not occurring in $\mathcal{A}$.

The $\rightarrow_{\forall}$-rule

***Condition*:** $\mathcal{A}$ contains $(\forall r.C)(x)$ and $r(x, y)$, but it does not contain $C(y)$.

***Action*:** $\mathcal{A}' := \mathcal{A} \cup \{C(y)\}$.

Tableau

- **Conjunction Rule**: If given (C_1 & C_2) on branch b then add C_1 and C_2 to b
- **Disjunction Rule**: If given (C_1 v C_2) on branch b_1 then add C_1 to new branch b_1 and C_2 to new branch b_2
- **Existential Rule**: If ∀x∃y r(x,y) & Cy on branch b_1 then add r(x,a) and Ca to a new branch, as long as "a" fresh
- **Universal Rule**: ∀x∀y r(x,y) → Cy & r(x,y) on b then add Cy to b

Tableau

⊓-rule
- **Conjunction Rule**: If given (C_1 & C_2) on branch b then add C_1 and C_2 to b

⊔-rule
- **Disjunction Rule**: If given (C_1 v C_2) on branch b_1 then add C_1 to new branch b_1 and C_2 to new branch b_2

∃-rule
- **Existential Rule**: If ∀x∃y r(x,y) & Cy on branch b_1 then add r(x,a) and Ca to a new branch, as long as "a" fresh

∀-rule
- **Universal Rule**: ∀x∀y r(x,y) → Cy & r(x,y) on b then add Cy to b

Consistency Checking

- Is the following set of propositions **consistent**?

$$\exists y\ s(a,x)\ \&\ F(x)$$

$$s(a,b)$$

$$\forall y\ s(a,y) \rightarrow \neg F(y) v \neg B(y)$$

$$B(b)$$

Davis-Putnam Algorithm

- Is the following set of propositions **consistent**?

$$\exists y\ s(a,x)\ \&\ F(x)$$

$$s(a,b)$$

$$\forall y\ s(a,y) \rightarrow \neg F(y) \vee \neg B(y)$$

$$B(b)$$

Davis-Putnam Algorithm

- If so, propositions in each subset can be assigned true without contradiction

Method of Analytic Tableau

- Is the following set of propositions **consistent**?

$$\exists y\ s(a,x) \ \&\ F(x)$$

$$s(a,b)$$

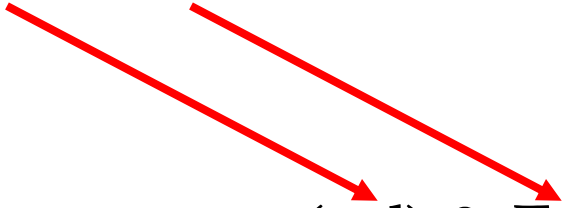
$$\forall y\ s(a,y) \rightarrow \neg F(y) \vee \neg B(y)$$

$$B(b)$$

Tableau Algorithm

- If so, each can be decomposed into a tree in which one branch is open

$\exists y(s(a,x) \,\&\, F(x))$, $s(a,b)$, $\forall y\, s(a,y) \rightarrow \neg F(y) \vee \neg B(y)$, $B(b)$



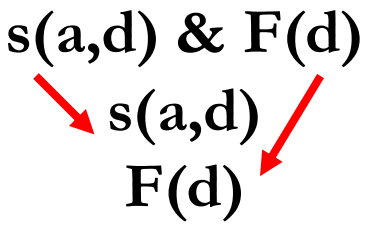
$\exists$-rule

$s(a,d) \,\&\, F(d)$

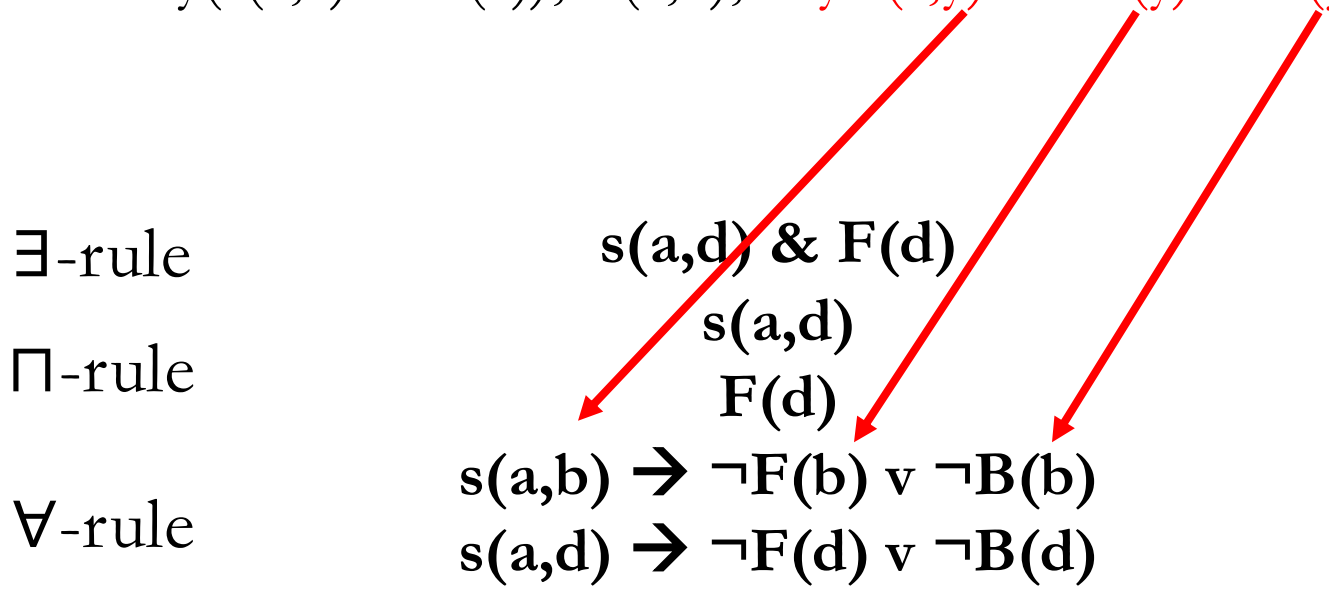
$\exists y(s(a,x) \,\&\, F(x)),\ \ s(a,b),\ \ \forall y\ s(a,y) \rightarrow \neg F(y) v \neg B(y),\ B(b)$

$\exists$-rule

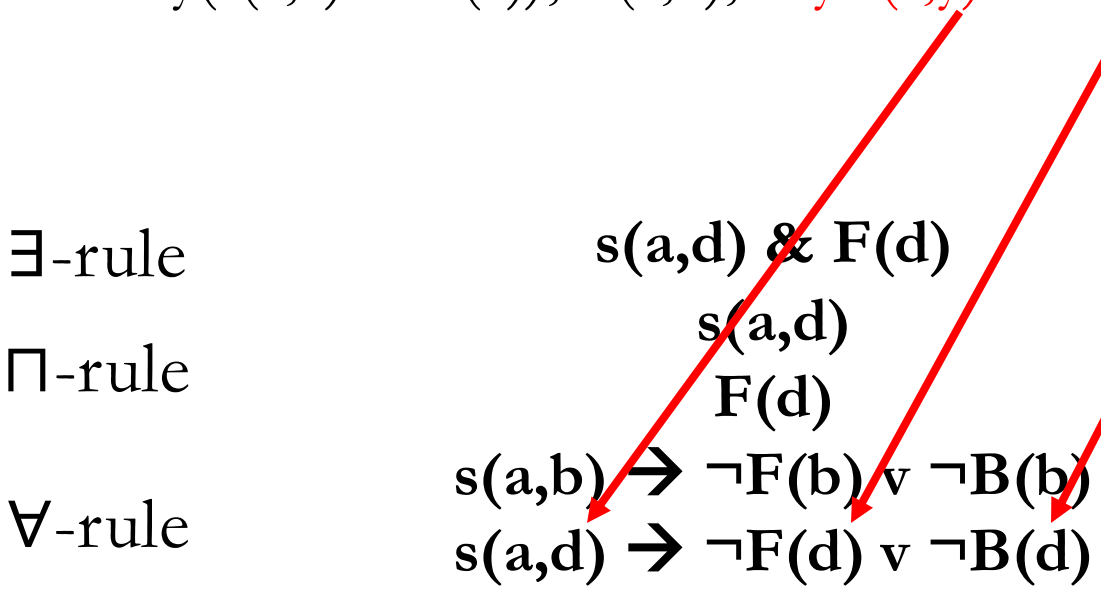
$\sqcap$-rule



$\exists y(s(a,x) \,\&\, F(x))$, $s(a,b)$, $\forall y\, s(a,y) \rightarrow \neg F(y) \vee \neg B(y)$, $B(b)$



$\exists y(s(a,x) \,\&\, F(x))$, $s(a,b)$, $\forall y\, s(a,y) \rightarrow \neg F(y) \vee \neg B(y)$, $B(b)$



$\exists y(s(a,x) \,\&\, F(x)),\ s(a,b),\ \forall y\ s(a,y) \rightarrow \neg F(y) v \neg B(y),\ B(b)$

∃-rule

s(a,d) & F(d)

⊓-rule

s(a,d)

F(d)

∀-rule

s(a,b) → ¬F(b) v ¬B(b)

s(a,d) → ¬F(d) v ¬B(d)

¬F(b) v ¬B(b)

$\exists y(s(a,x) \,\&\, F(x)),\ \ s(a,b),\ \ \forall y\ s(a,y) \rightarrow \neg F(y) v \neg B(y),\ B(b)$

∃-rule

s(a,d) & F(d)

⊓-rule

s(a,d)

F(d)

∀-rule

s(a,b) → ¬F(b) v ¬B(b)

s(a,d) → ¬F(d) v ¬B(d)

¬F(b) v ¬B(b)

¬F(d) v ¬B(d)

$\exists y(s(a,x) \,\&\, F(x)),\ \ s(a,b),\ \ \forall y\ s(a,y) \rightarrow \neg F(y) v \neg B(y),\ B(b)$

∃-rule	**s(a,d) & F(d)**
⊓-rule	**s(a,d)** **F(d)**
∀-rule	**s(a,b) → ¬F(b) v ¬B(b)** **s(a,d) → ¬F(d) v ¬B(d)**
	¬F(b) v ¬B(b)
	¬F(d) v ¬B(d)
⊔-rule	**¬F(b)** **¬B(b)**

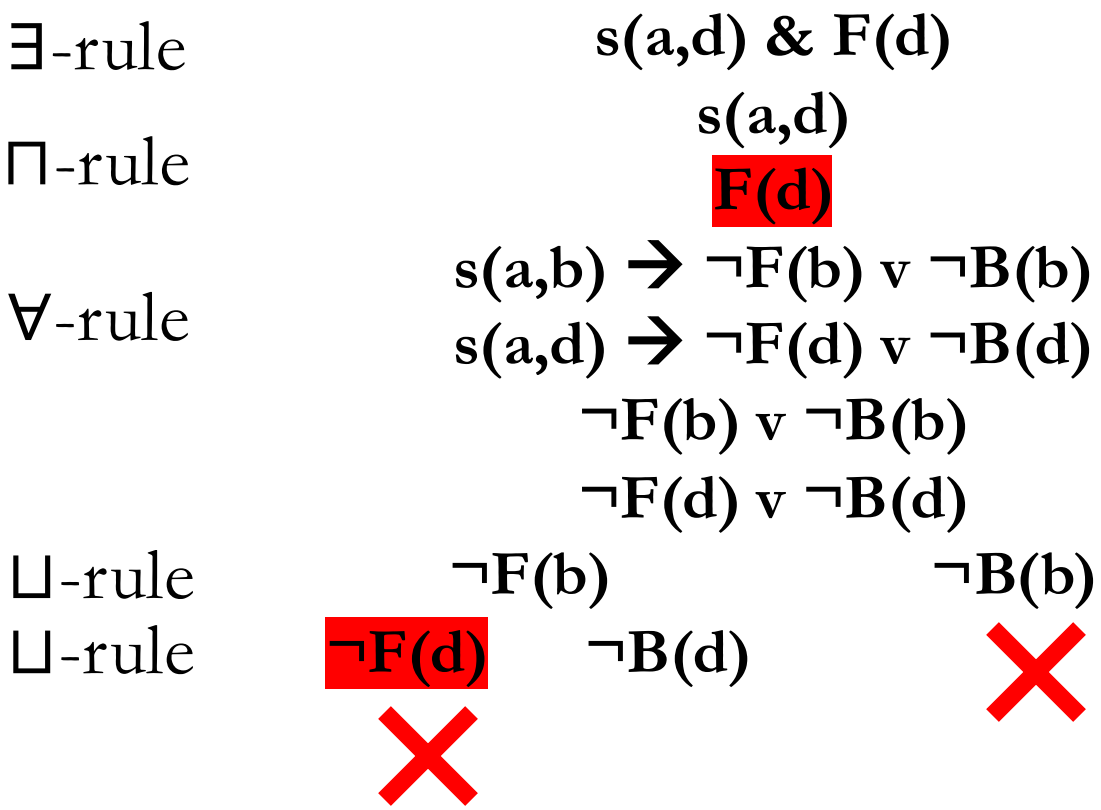
$\exists y(s(a,x) \,\&\, F(x)),\ \ s(a,b),\ \ \forall y\ s(a,y) \rightarrow \neg F(y) v \neg B(y),\ B(b)$

∃-rule	**s(a,d) & F(d)**
⊓-rule	**s(a,d)** **F(d)**
∀-rule	**s(a,b) → ¬F(b) v ¬B(b)** **s(a,d) → ¬F(d) v ¬B(d)**
	¬F(b) v ¬B(b) **¬F(d) v ¬B(d)**
⊔-rule	**¬F(b)** **¬B(b)**

$\exists y(s(a,x) \,\&\, F(x)),\ \ s(a,b),\ \ \forall y\ s(a,y) \rightarrow \neg F(y) v \neg B(y),\ B(b)$

∃-rule	**s(a,d) & F(d)**
⊓-rule	**s(a,d)** **F(d)**
∀-rule	**s(a,b) → ¬F(b) v ¬B(b)** **s(a,d) → ¬F(d) v ¬B(d)**
	¬F(b) v ¬B(b)
	¬F(d) v ¬B(d)
⊔-rule	**¬F(b)** **¬B(b)**
⊔-rule	**¬F(d)** **¬B(d)**

$\exists y(s(a,x) \,\&\, F(x)),\ \ s(a,b),\ \ \forall y\ s(a,y) \rightarrow \neg F(y) v \neg B(y),\ B(b)$



$\exists y(s(a,x) \,\&\, F(x)),\ \ s(a,b),\ \ \forall y\ s(a,y) \rightarrow \neg F(y) v \neg B(y),\ B(b)$

∃-rule **s(a,d) & F(d)**

⊓-rule **s(a,d)**
F(d)

∀-rule **s(a,b) → ¬F(b) v ¬B(b)**
s(a,d) → ¬F(d) v ¬B(d)

¬F(b) v ¬B(b)
¬F(d) v ¬B(d)

⊔-rule **¬F(b)** **¬B(b)**

⊔-rule **¬F(d)** **¬B(d)**

What is "Artificial Intelligence"

- AI is a multifaceted phenomenon, much like intelligence

- Appears to include:
  - Ability to learn
  - Ability to plan/reason
  - **Ability to communicate**
  - Ability to reason probabilistically

Ability to Communicate

- The ALPAC Report of 1966 concluded machine translation was more expensive, less accurate and slower than human translation

- Dragon Dictate 1.0 Speech Recognizer minimized the probability of error using a new technology called a **hidden Markov model**

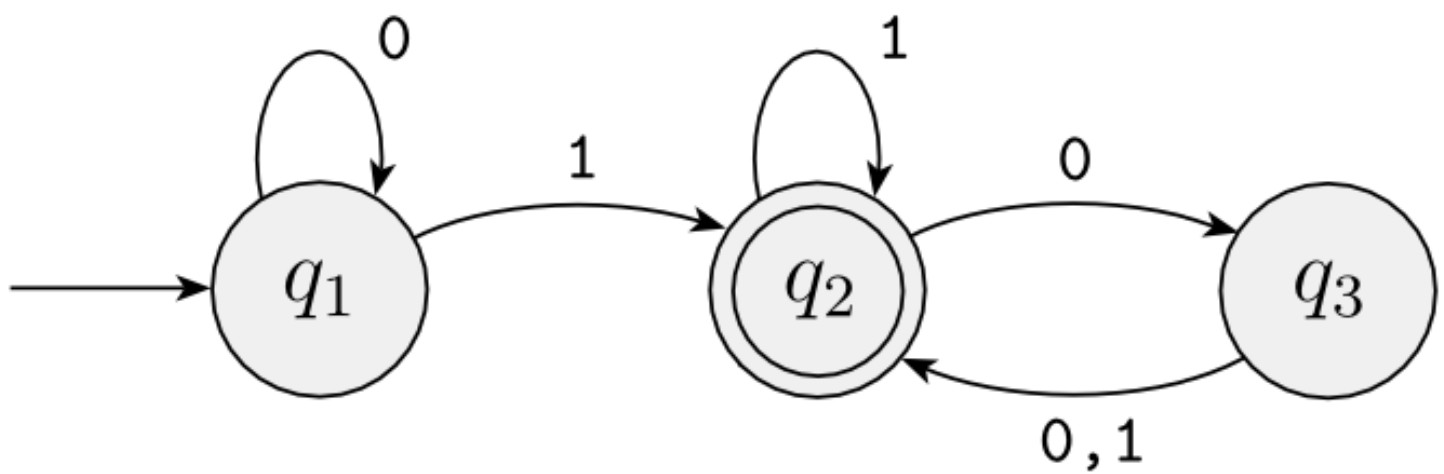
A ***finite automaton*** is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

1. Q is a finite set called the ***states***,
2. Σ is a finite set called the ***alphabet***,
3. $\delta : Q \times \Sigma \longrightarrow Q$ is the ***transition function***,
4. $q_0 \in Q$ is the ***start state***, and
5. $F \subseteq Q$ is the ***set of accept states***.

1. $Q = \{q_1, q_2, q_3\}$,
2. $\Sigma = \{0,1\}$,
3. δ is described as

	0	1
q_1	q_1	q_2
q_2	q_3	q_2
q_3	q_2	q_2,

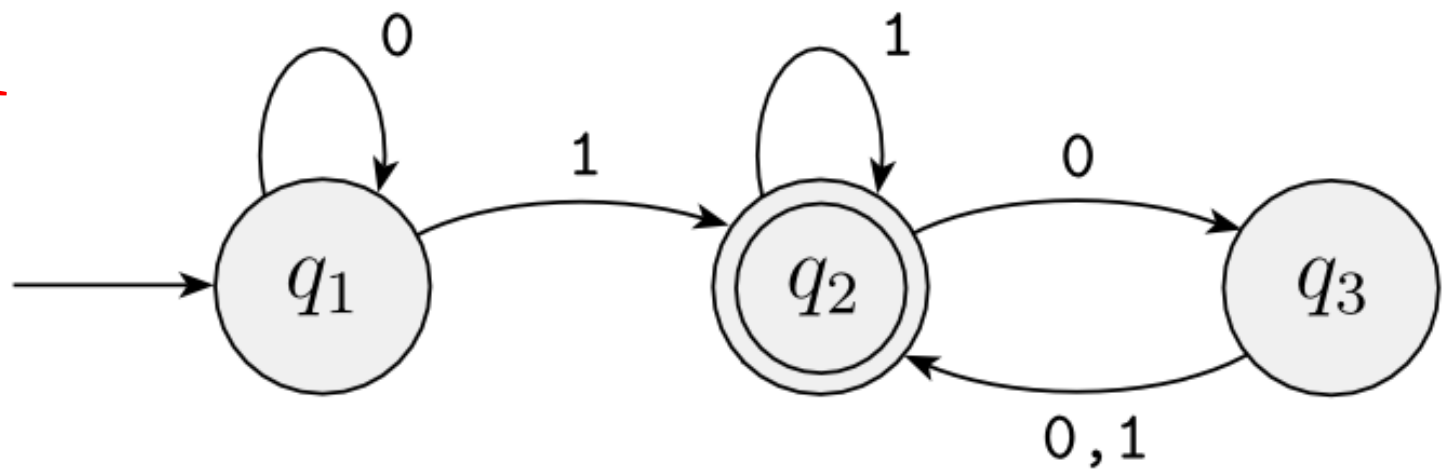
4. q_1 is the start state, and
5. $F = \{q_2\}$.



1. $Q = \{q_1, q_2, q_3\}$,
2. $\Sigma = \{0,1\}$,
3. δ is described as
4. q_1 is the start state, and
5. $F = \{q_2\}$.

	0	1
q_1	q_1	q_2
q_2	q_3	q_2
q_3	q_2	q_2,

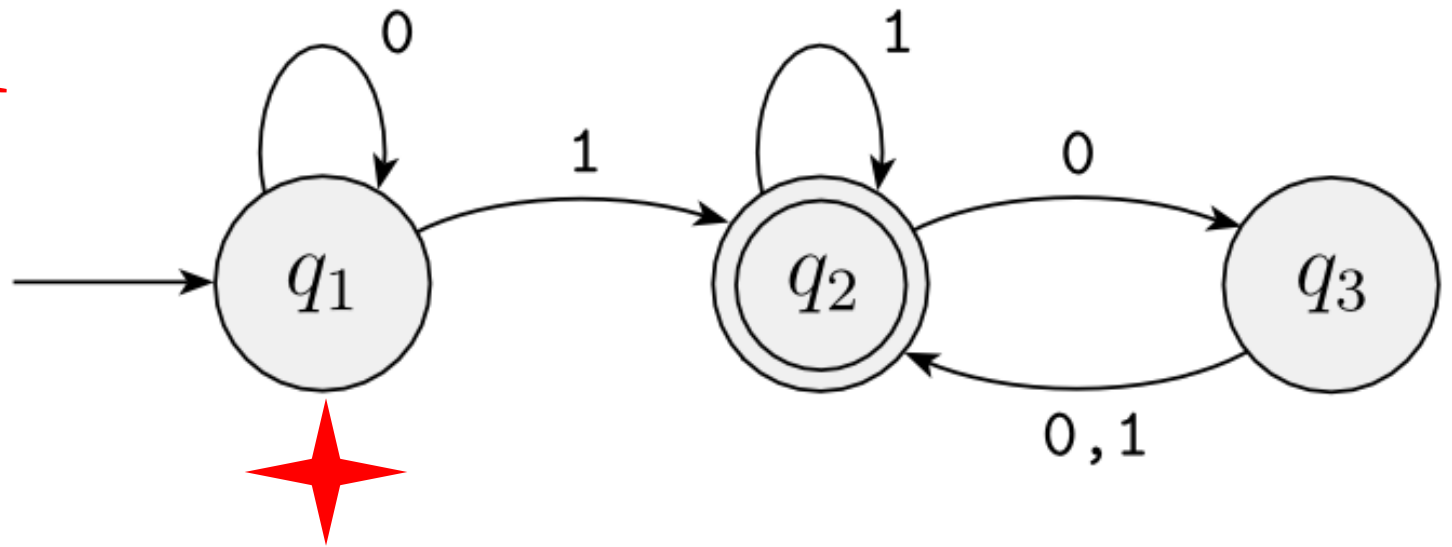
01011



1. $Q = \{q_1, q_2, q_3\}$,
2. $\Sigma = \{0,1\}$,
3. δ is described as
4. q_1 is the start state, and
5. $F = \{q_2\}$.

	0	1
q_1	q_1	q_2
q_2	q_3	q_2
q_3	q_2	q_2,

01011

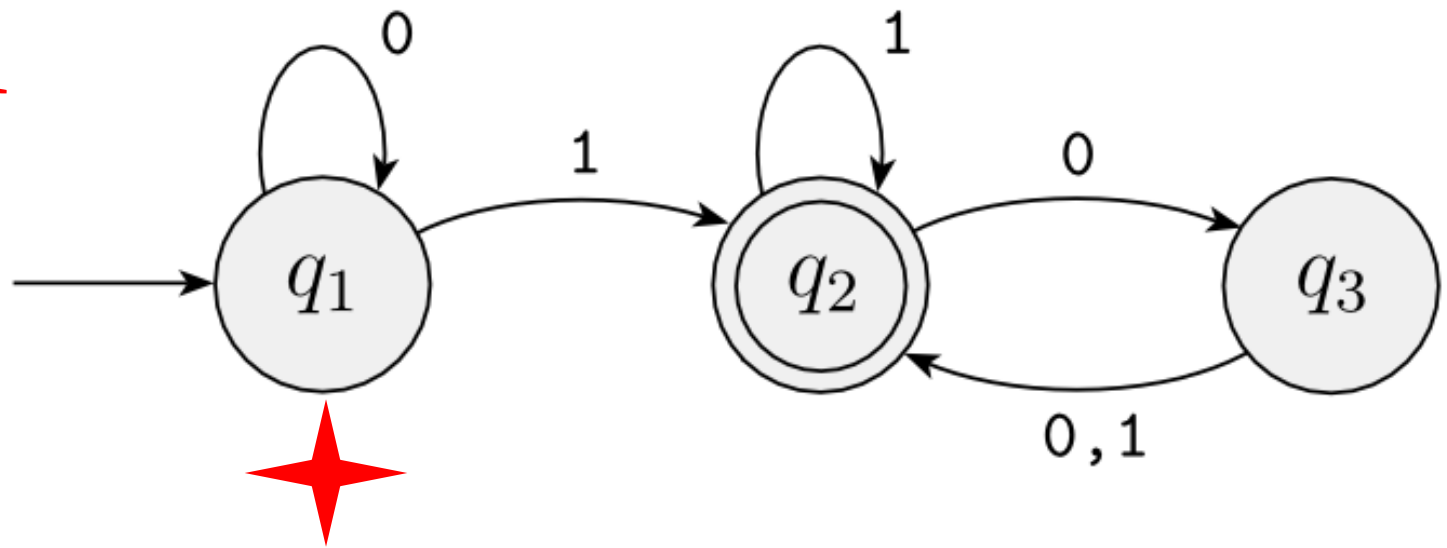


1. $Q = \{q_1, q_2, q_3\}$,
2. $\Sigma = \{0,1\}$,
3. δ is described as

	0	1
q_1	q_1	q_2
q_2	q_3	q_2
q_3	q_2	q_2,

4. q_1 is the start state, and
5. $F = \{q_2\}$.

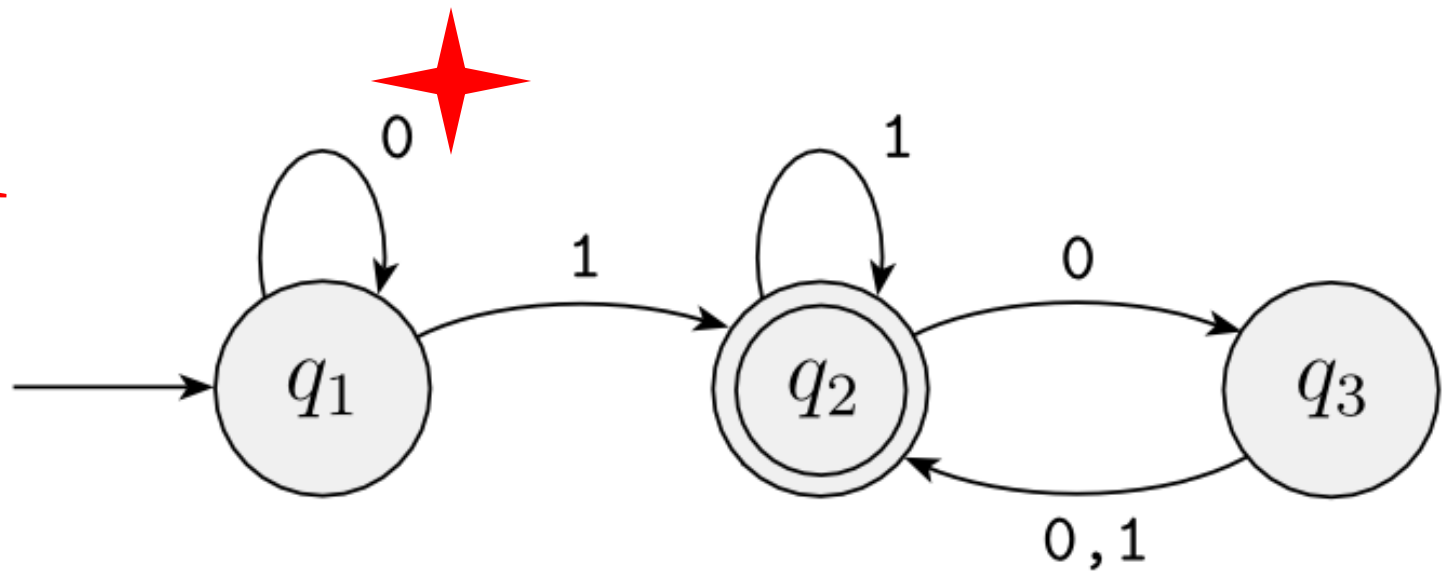
01011



1. $Q = \{q_1, q_2, q_3\}$,
2. $\Sigma = \{0,1\}$,
3. δ is described as
4. q_1 is the start state, and
5. $F = \{q_2\}$.

	0	1
q_1	q_1	q_2
q_2	q_3	q_2
q_3	q_2	q_2,

01011

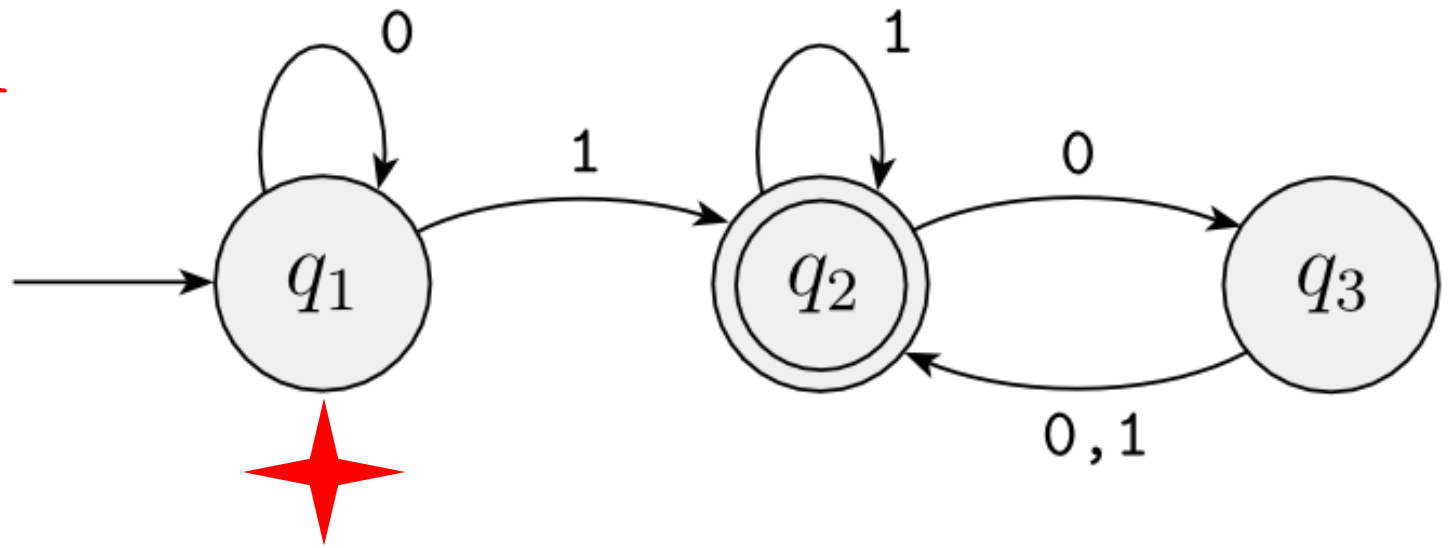


1. $Q = \{q_1, q_2, q_3\}$,
2. $\Sigma = \{0,1\}$,
3. δ is described as

	0	1
q_1	q_1	q_2
q_2	q_3	q_2
q_3	q_2	q_2,

4. q_1 is the start state, and
5. $F = \{q_2\}$.

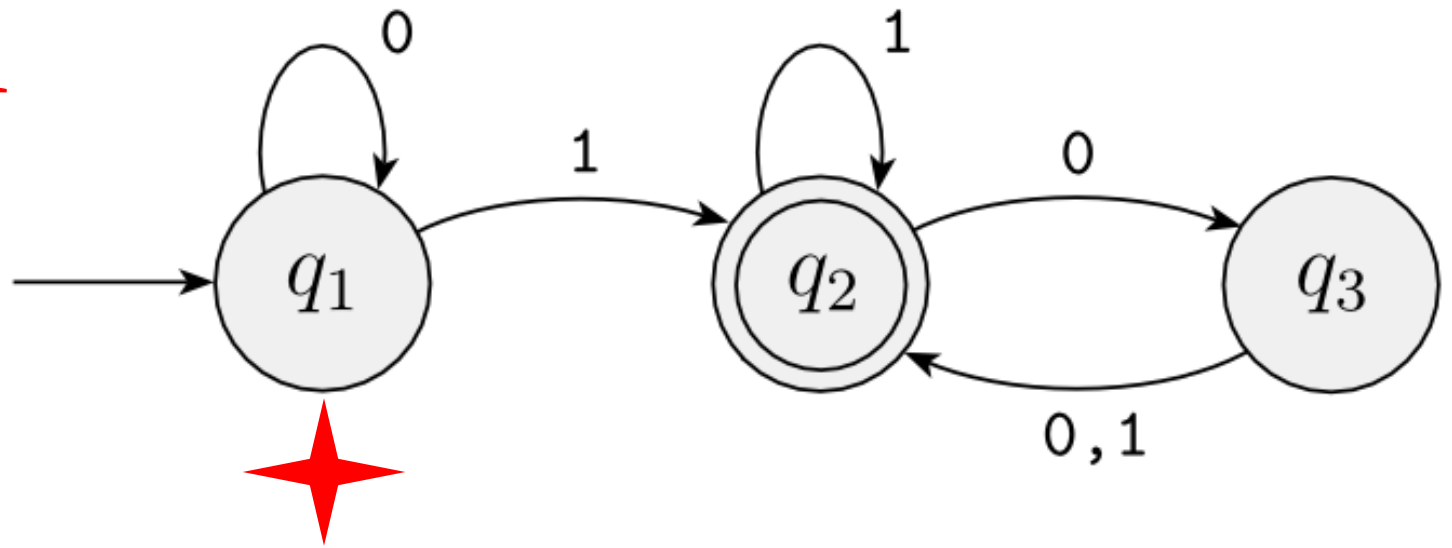
01011



1. $Q = \{q_1, q_2, q_3\}$,
2. $\Sigma = \{0,1\}$,
3. δ is described as
4. q_1 is the start state, and
5. $F = \{q_2\}$.

	0	1
q_1	q_1	q_2
q_2	q_3	q_2
q_3	q_2	q_2,

01011

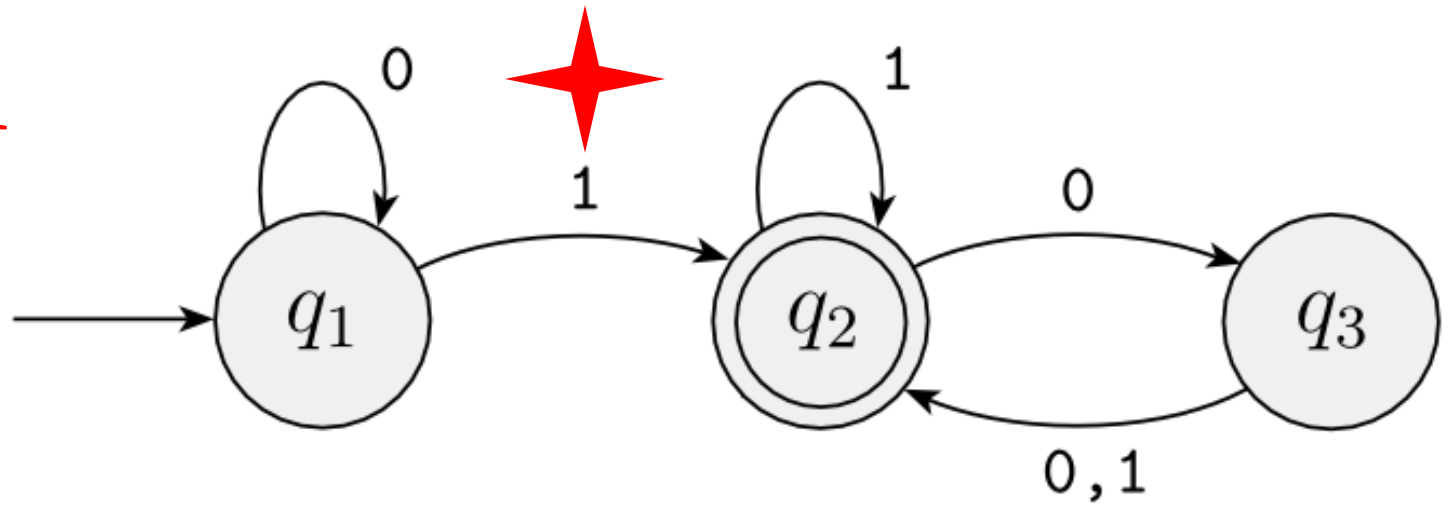


1. $Q = \{q_1, q_2, q_3\}$,
2. $\Sigma = \{0,1\}$,
3. δ is described as

	0	1
q_1	q_1	q_2
q_2	q_3	q_2
q_3	q_2	q_2,

4. q_1 is the start state, and
5. $F = \{q_2\}$.

01011

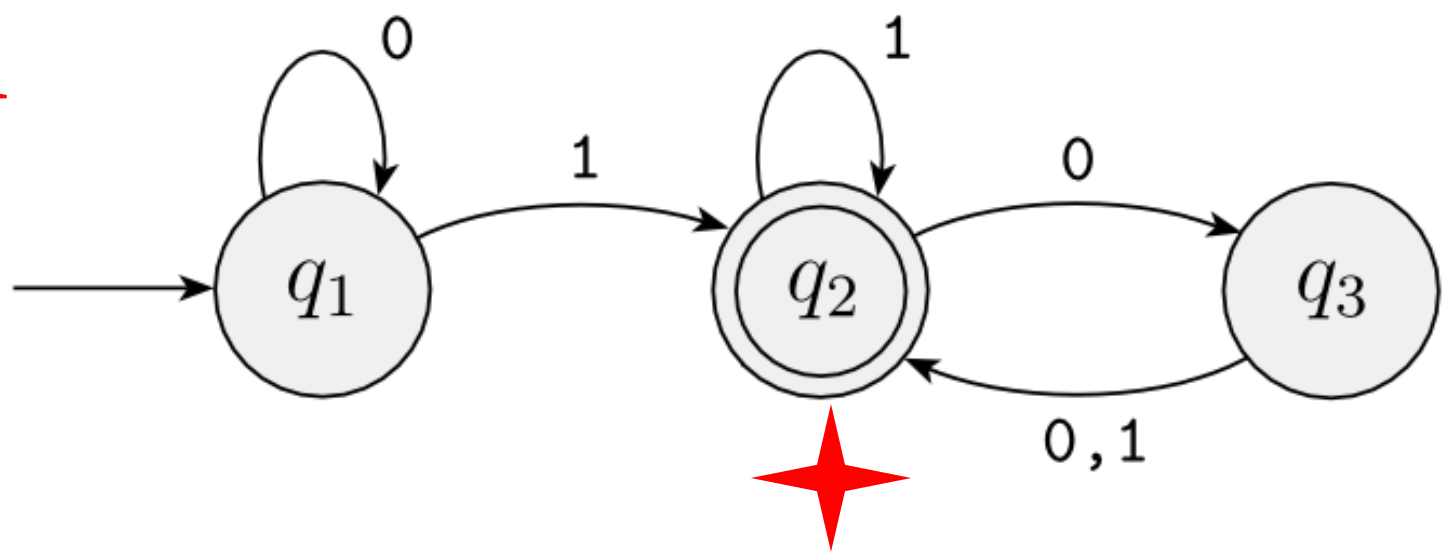


1. $Q = \{q_1, q_2, q_3\}$,
2. $\Sigma = \{0,1\}$,
3. δ is described as

	0	1
q_1	q_1	q_2
q_2	q_3	q_2
q_3	q_2	q_2,

4. q_1 is the start state, and
5. $F = \{q_2\}$.

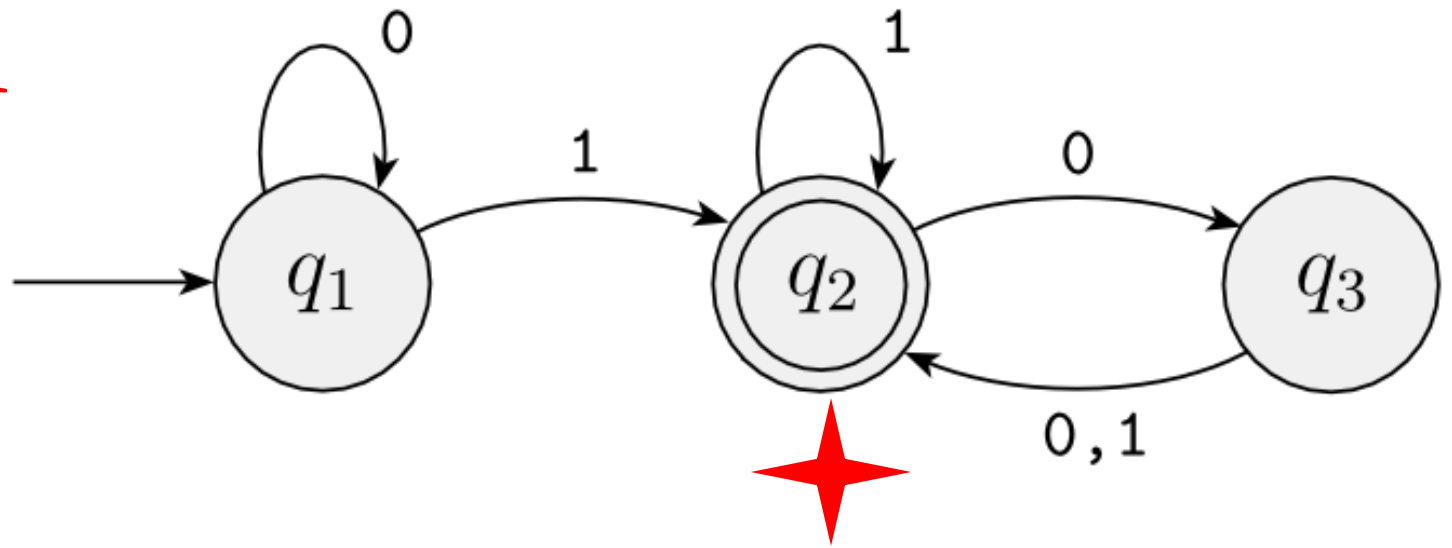
01011



1. $Q = \{q_1, q_2, q_3\}$,
2. $\Sigma = \{0,1\}$,
3. δ is described as
4. q_1 is the start state, and
5. $F = \{q_2\}$.

	0	1
q_1	q_1	q_2
q_2	q_3	q_2
q_3	q_2	q_2,

01011

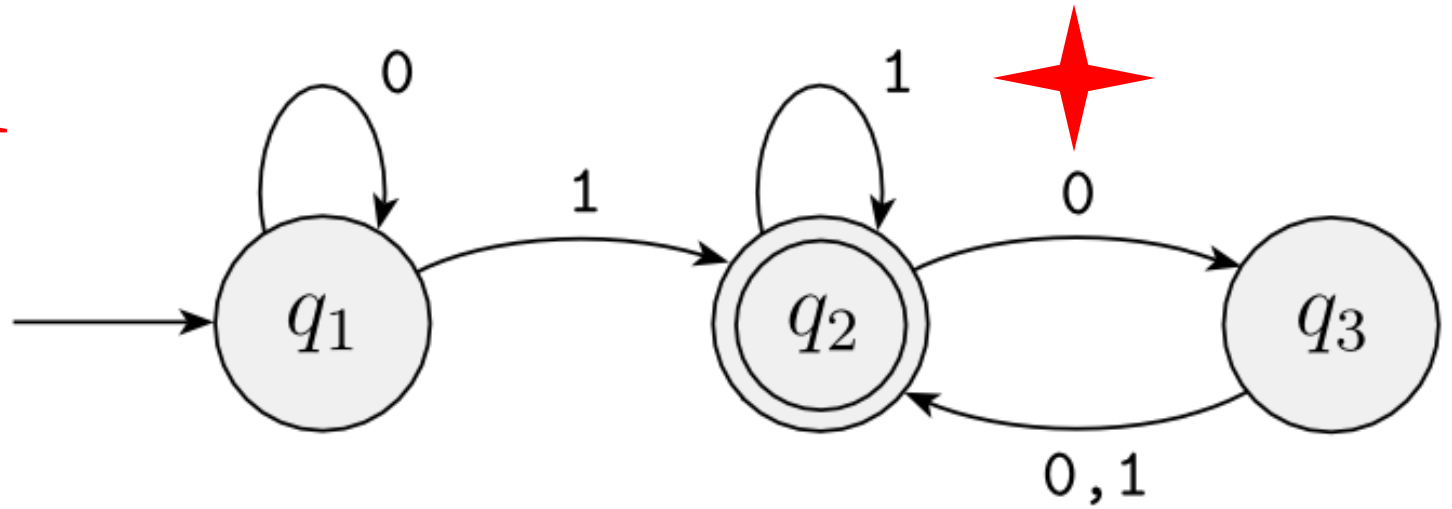


1. $Q = \{q_1, q_2, q_3\}$,
2. $\Sigma = \{0,1\}$,
3. δ is described as

	0	1
q_1	q_1	q_2
q_2	q_3	q_2
q_3	q_2	q_2,

4. q_1 is the start state, and
5. $F = \{q_2\}$.

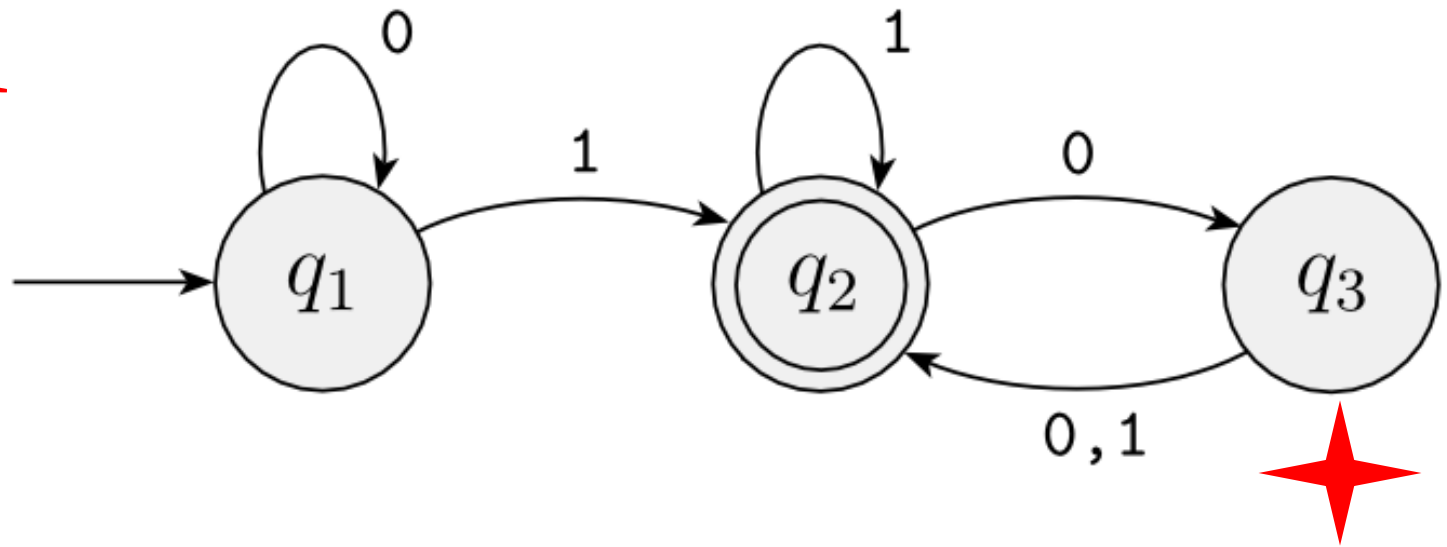
01011



1. $Q = \{q_1, q_2, q_3\}$,
2. $\Sigma = \{0,1\}$,
3. δ is described as
4. q_1 is the start state, and
5. $F = \{q_2\}$.

	0	1
q_1	q_1	q_2
q_2	q_3	q_2
q_3	q_2	q_2,

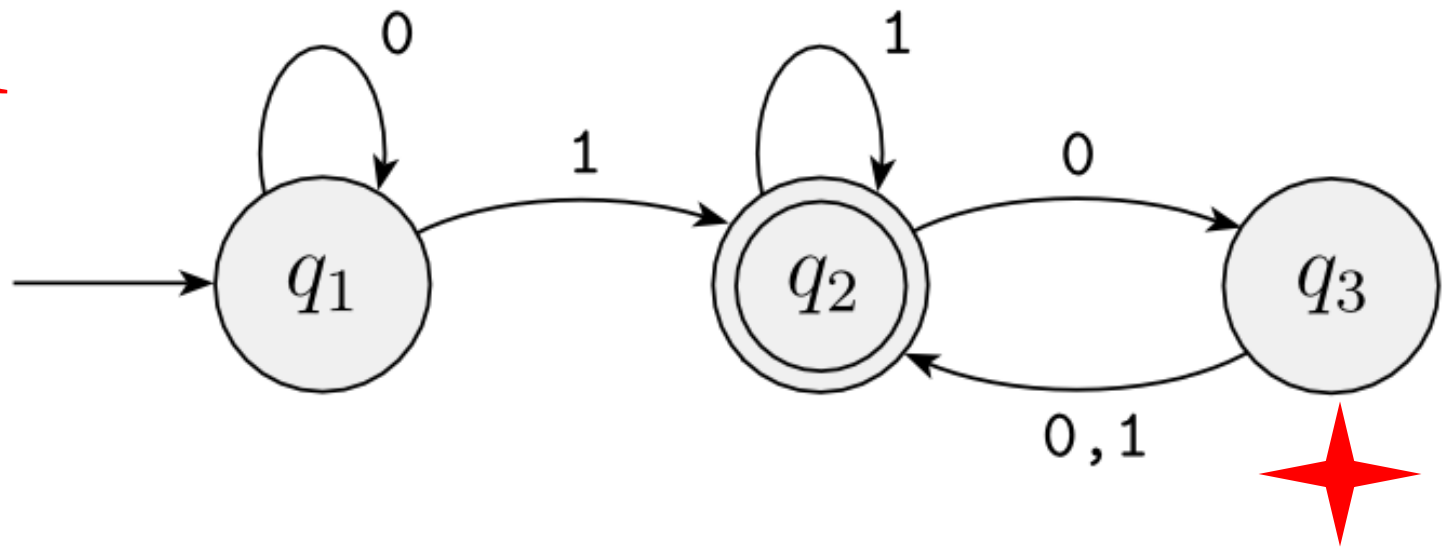
01011



1. $Q = \{q_1, q_2, q_3\}$,
2. $\Sigma = \{0,1\}$,
3. δ is described as
4. q_1 is the start state, and
5. $F = \{q_2\}$.

	0	1
q_1	q_1	q_2
q_2	q_3	q_2
q_3	q_2	q_2,

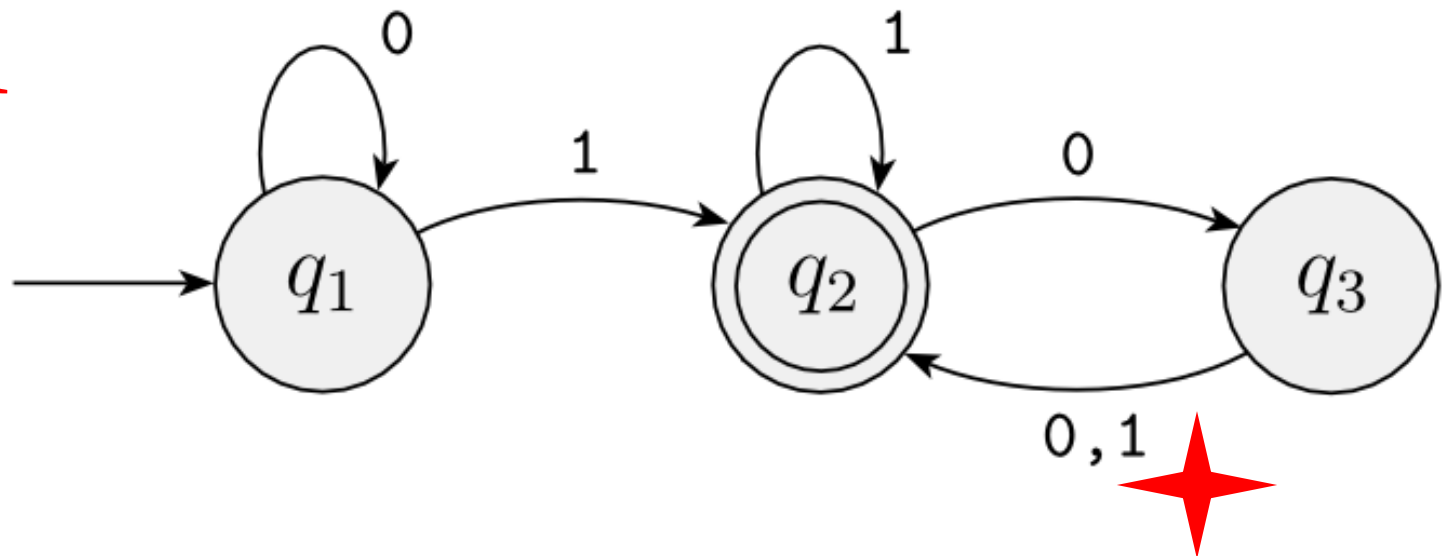
01011



1. $Q = \{q_1, q_2, q_3\}$,
2. $\Sigma = \{0,1\}$,
3. δ is described as
4. q_1 is the start state, and
5. $F = \{q_2\}$.

	0	1
q_1	q_1	q_2
q_2	q_3	q_2
q_3	q_2	q_2,

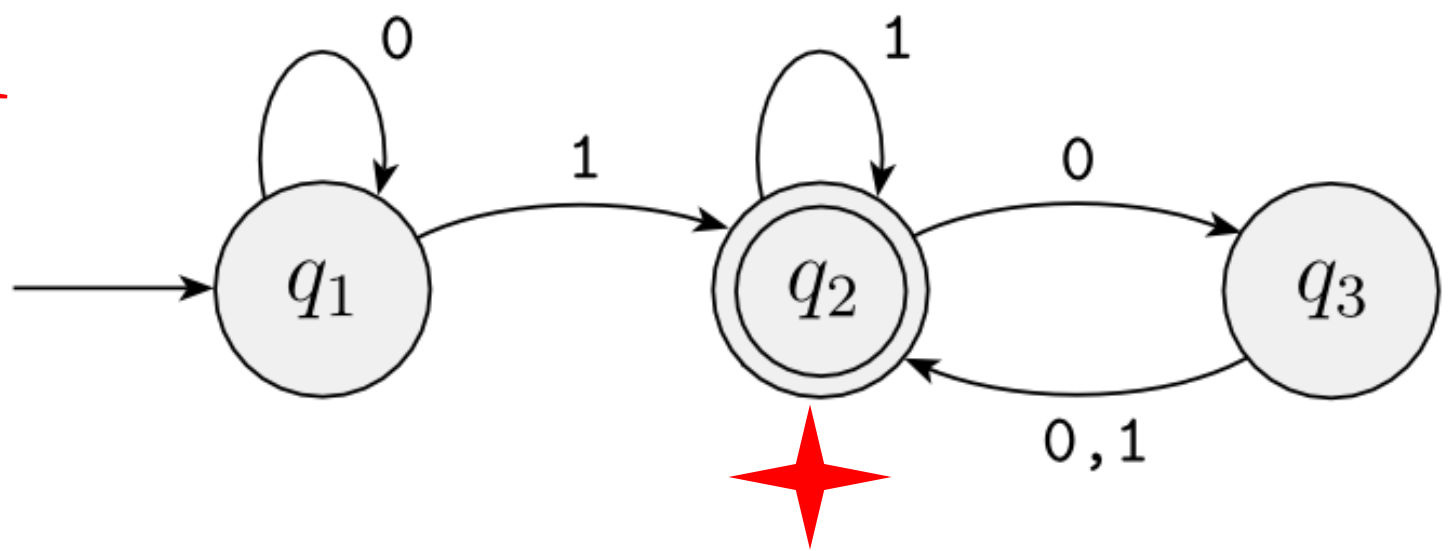
01011



1. $Q = \{q_1, q_2, q_3\}$,
2. $\Sigma = \{0,1\}$,
3. δ is described as
4. q_1 is the start state, and
5. $F = \{q_2\}$.

	0	1
q_1	q_1	q_2
q_2	q_3	q_2
q_3	q_2	q_2,

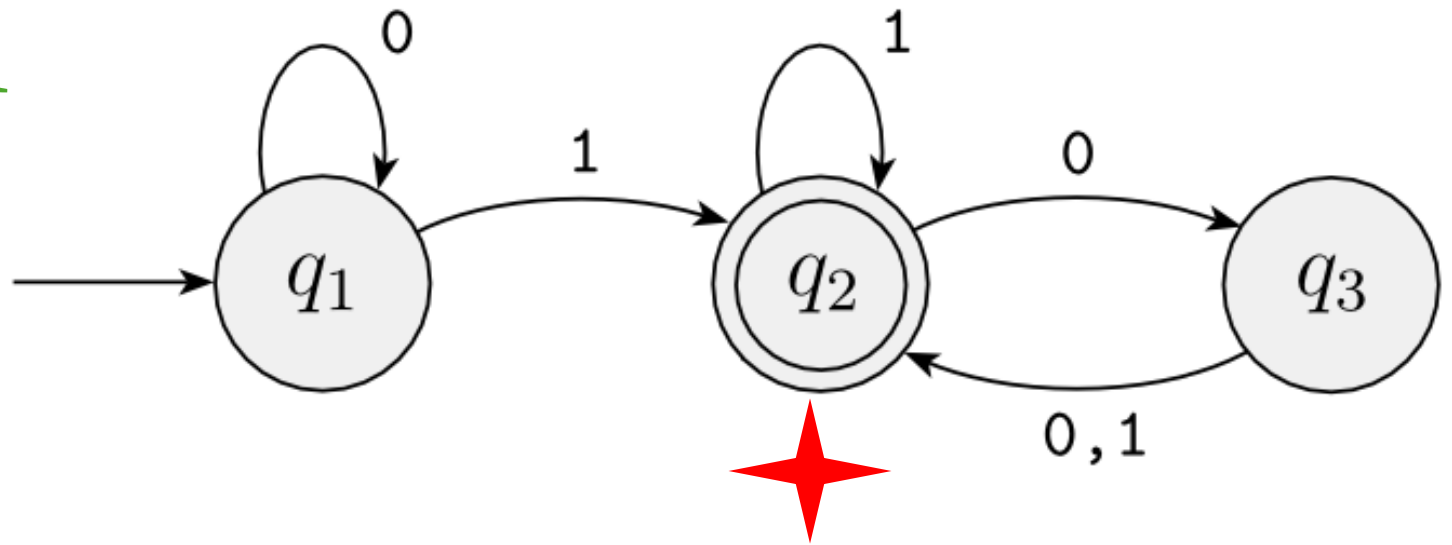
01011



1. $Q = \{q_1, q_2, q_3\}$,
2. $\Sigma = \{0,1\}$,
3. δ is described as
4. q_1 is the start state, and
5. $F = \{q_2\}$.

	0	1
q_1	q_1	q_2
q_2	q_3	q_2
q_3	q_2	q_2,

01011

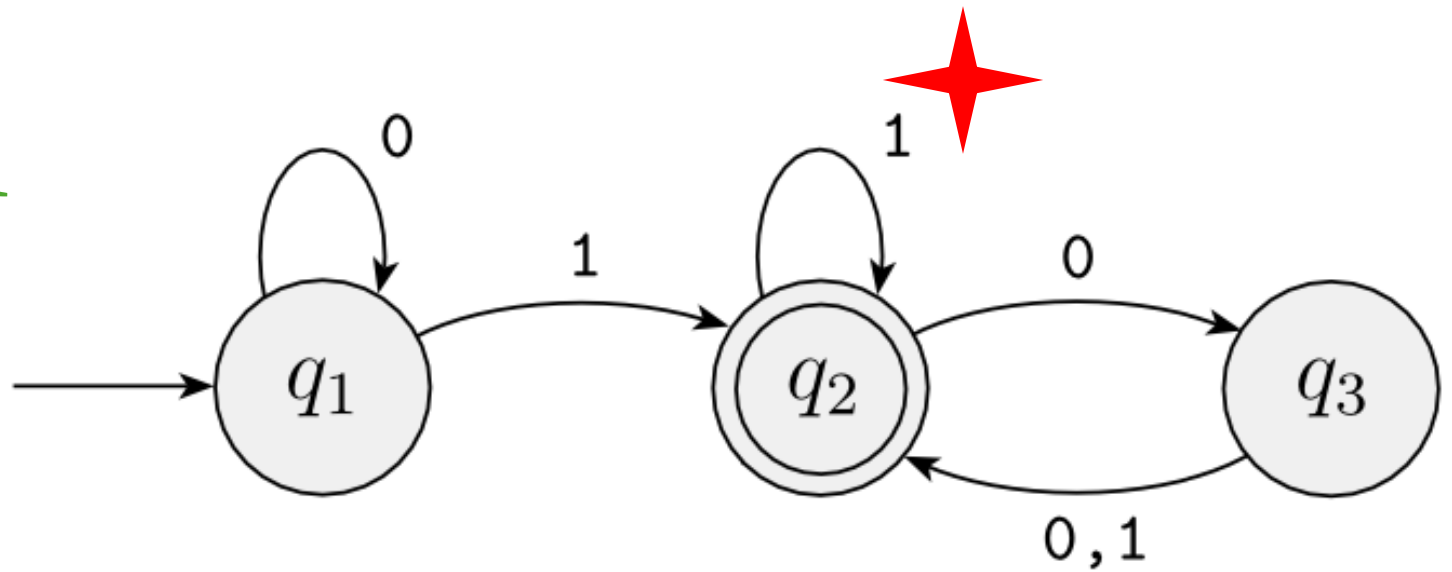


1. $Q = \{q_1, q_2, q_3\}$,
2. $\Sigma = \{0,1\}$,
3. δ is described as

	0	1
q_1	q_1	q_2
q_2	q_3	q_2
q_3	q_2	q_2,

4. q_1 is the start state, and
5. $F = \{q_2\}$.

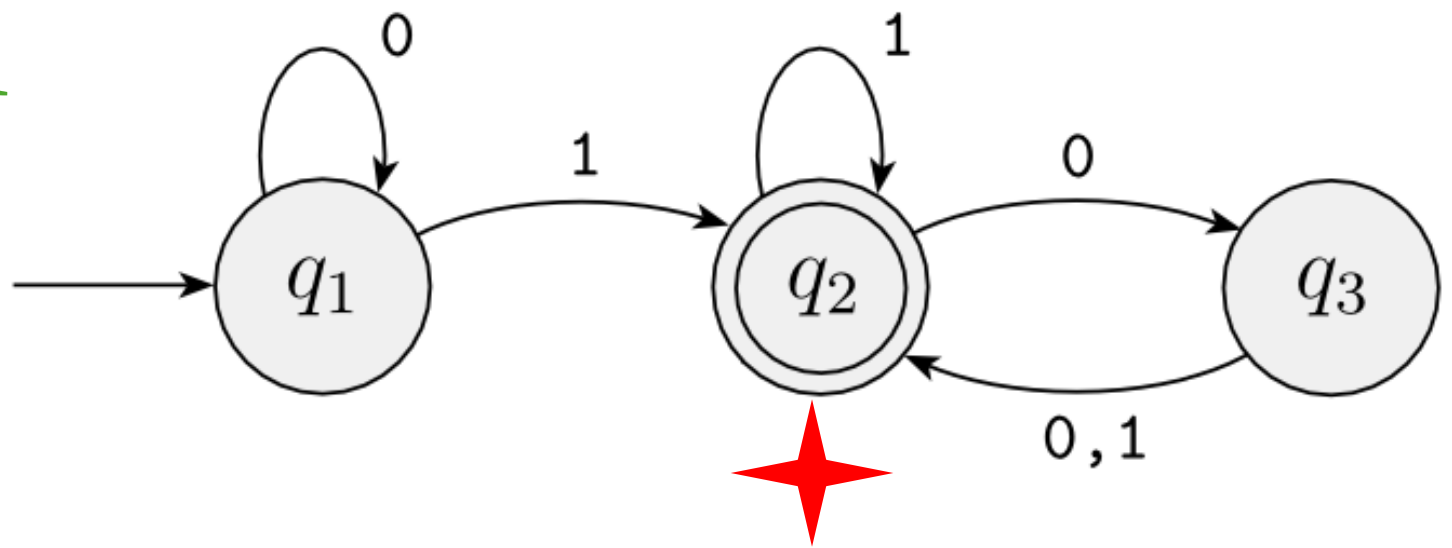
01011



1. $Q = \{q_1, q_2, q_3\}$,
2. $\Sigma = \{0,1\}$,
3. δ is described as
4. q_1 is the start state, and
5. $F = \{q_2\}$.

	0	1
q_1	q_1	q_2
q_2	q_3	q_2
q_3	q_2	q_2,

01011

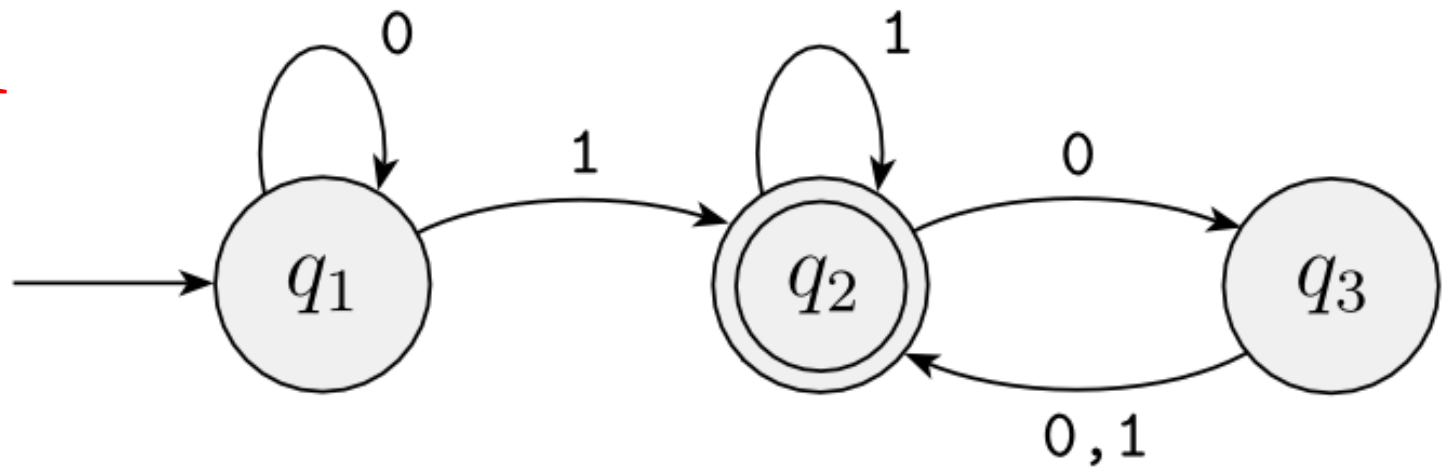


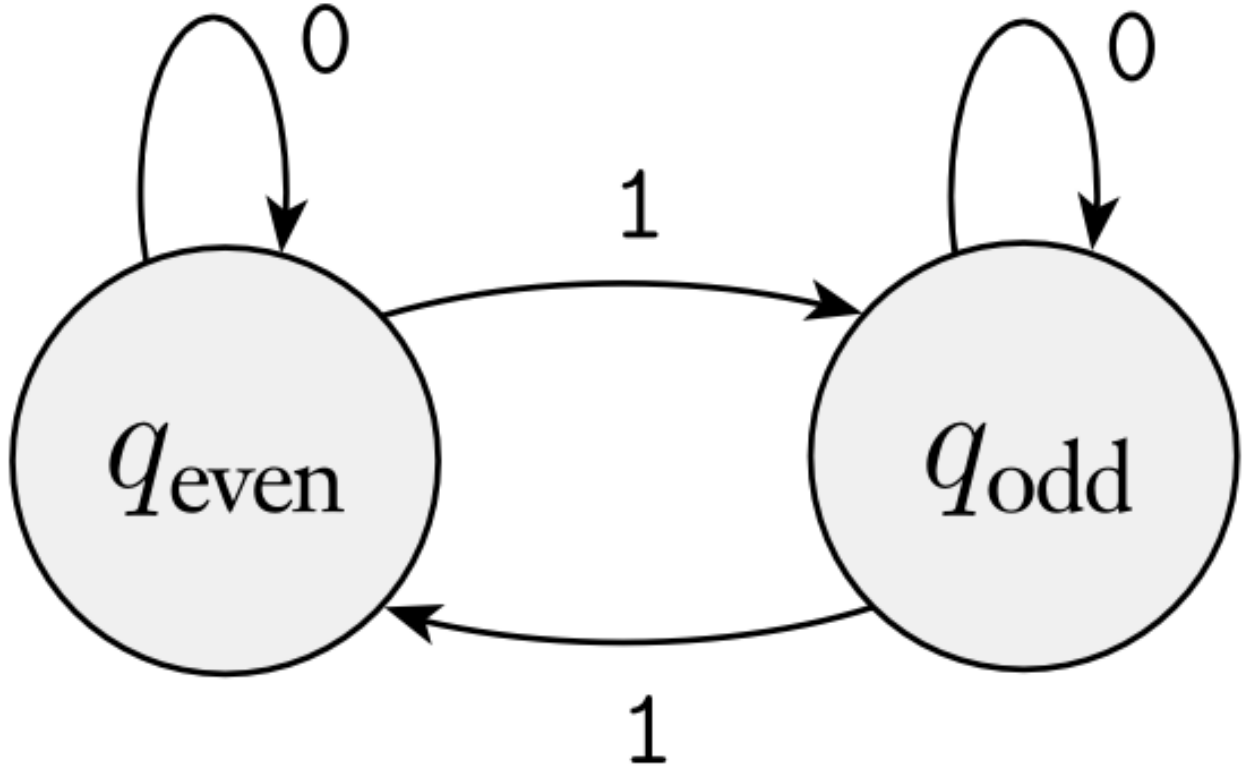
1. $Q = \{q_1, q_2, q_3\}$,
2. $\Sigma = \{0,1\}$,
3. δ is described as

	0	1
q_1	q_1	q_2
q_2	q_3	q_2
q_3	q_2	q_2,

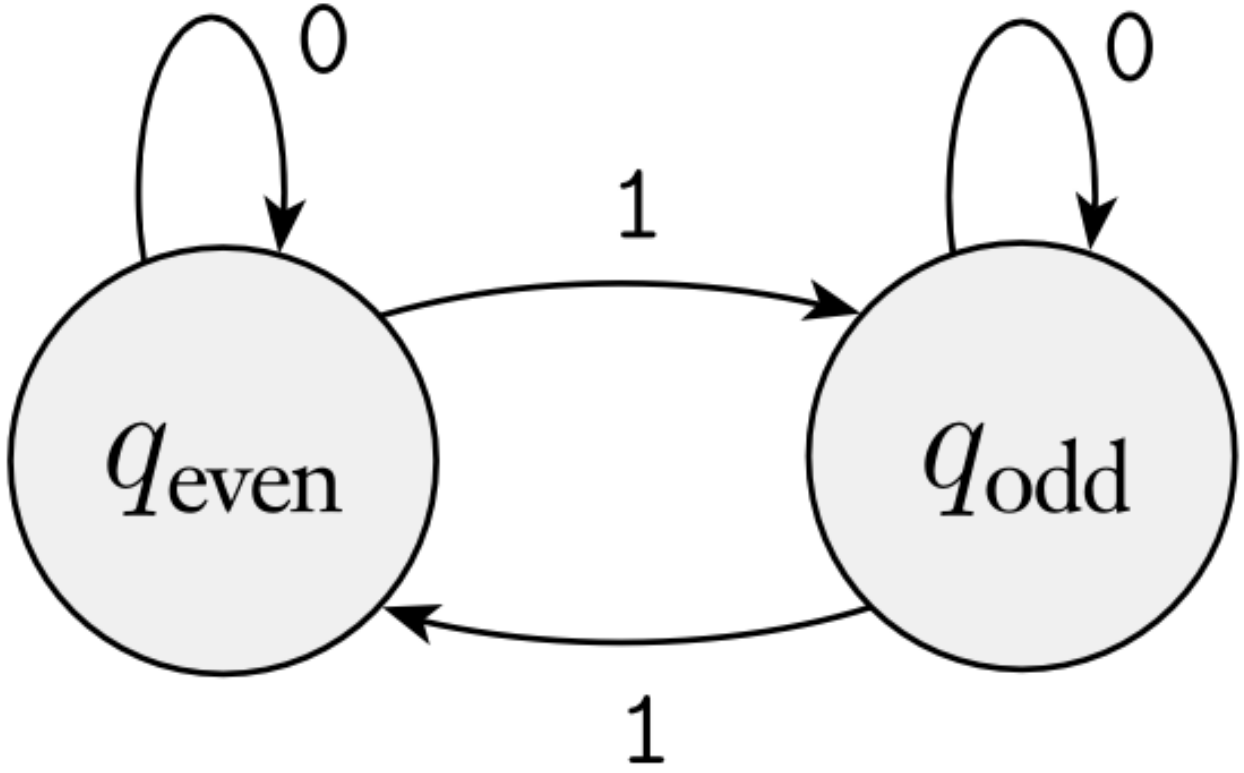
4. q_1 is the start state, and
5. $F = \{q_2\}$.

01011

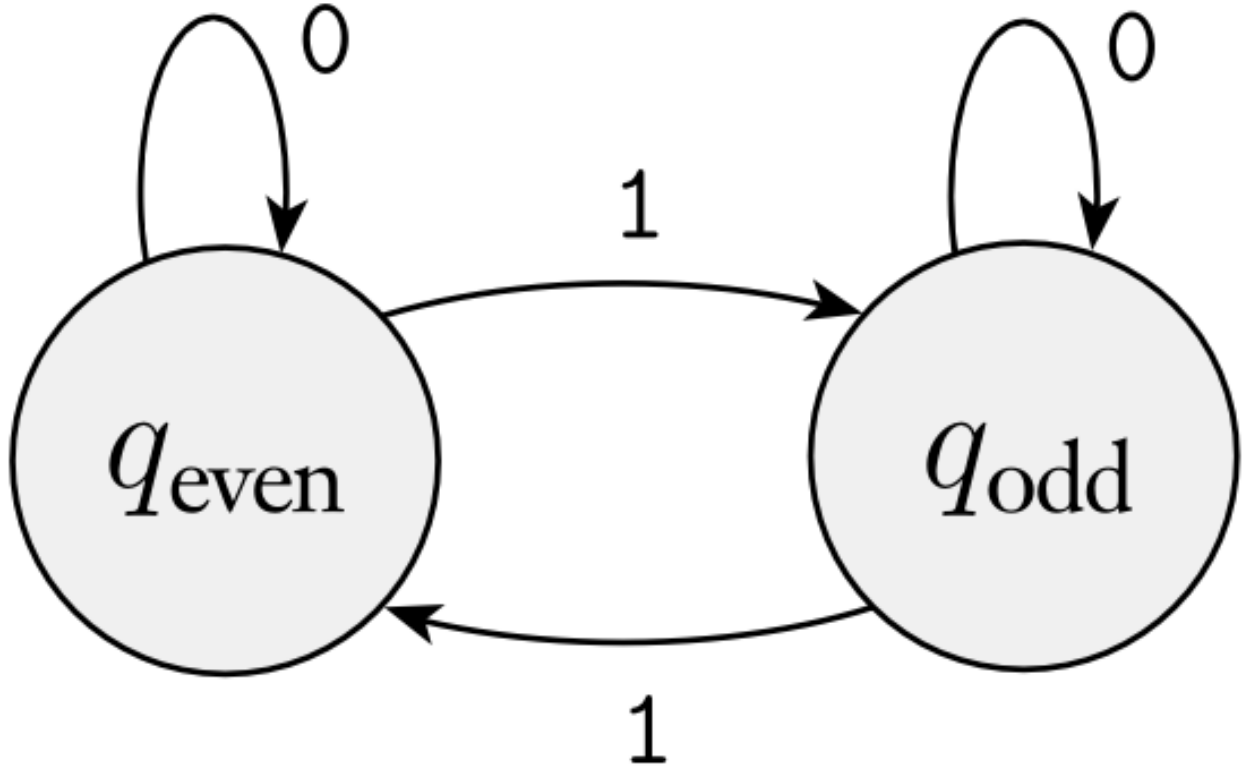



0
0
1
q_{even}
q_{odd}
1

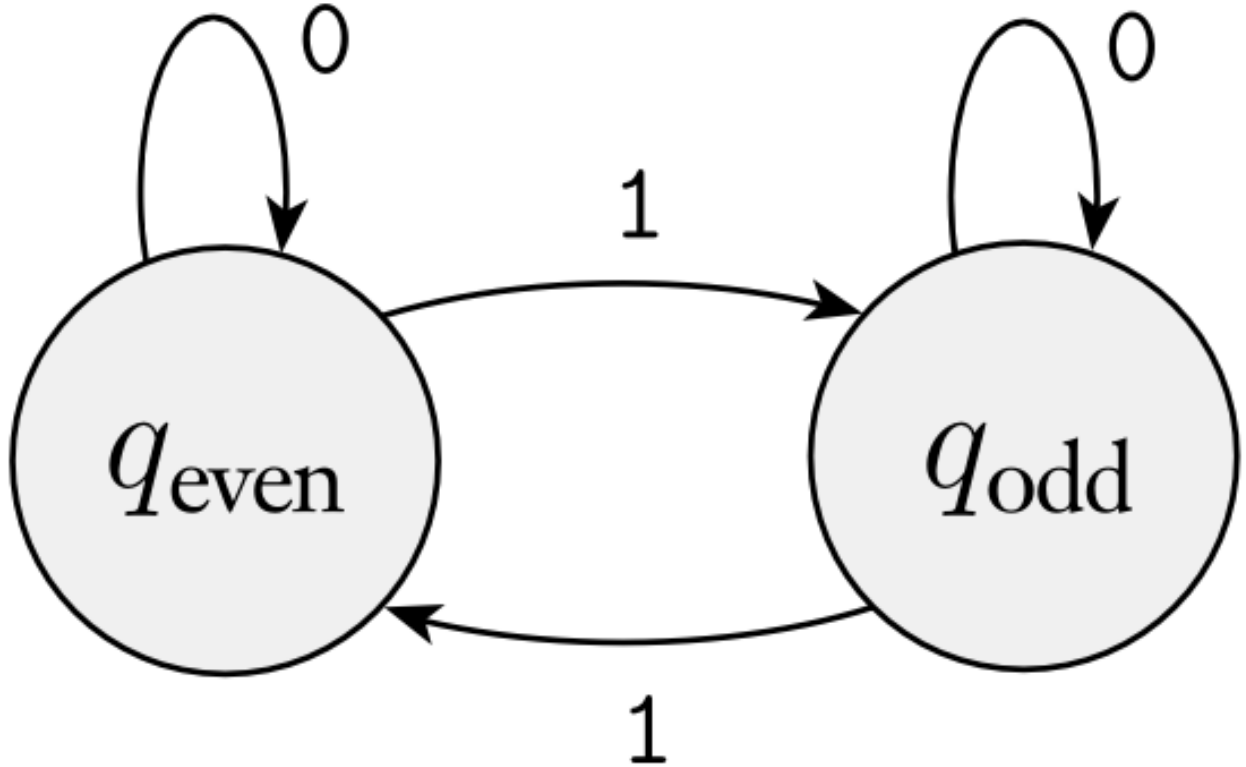
2+0=Even



2+1=Odd



2+1+0=Odd



Markov Model

- A Markov model (or more specifically, a Markov chain) can be seen as a probabilistic generalization of a finite automaton

- Instead of saying "if I'm in state A and see symbol x, I go to state B," say "if I'm in state A, then I have a probability $P(B|A)$ of going to B."

- In other words, the deterministic transition function δ becomes a transition probability distribution

*Markov Model**

A ~~***finite automaton***~~ is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

1. Q is a finite set called the ***states***,
2. Σ is a finite set called the ***alphabet***,
3. $\delta : Q \times \Sigma \longrightarrow Q$ is the ~~***transition function***~~,
4. $q_0 \in Q$ is the ***start state***, and
5. $F \subseteq Q$ is the ***set of accept states***.

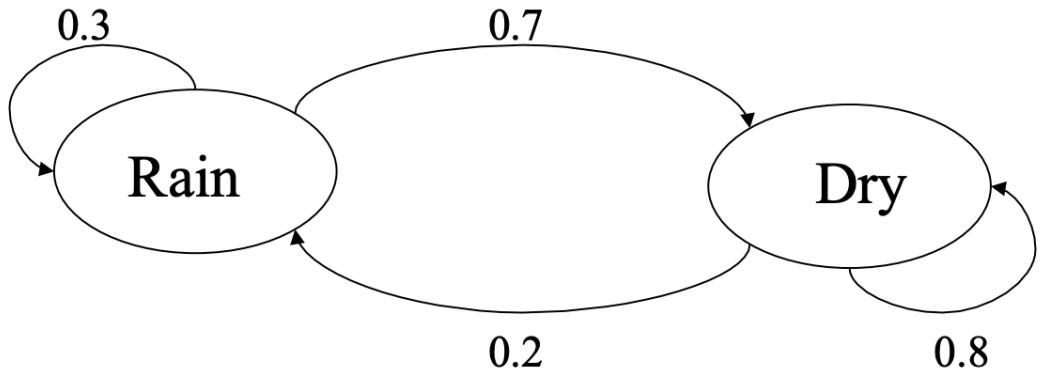
Transition Probabilistic Function

**Roughly...*

Markov Model

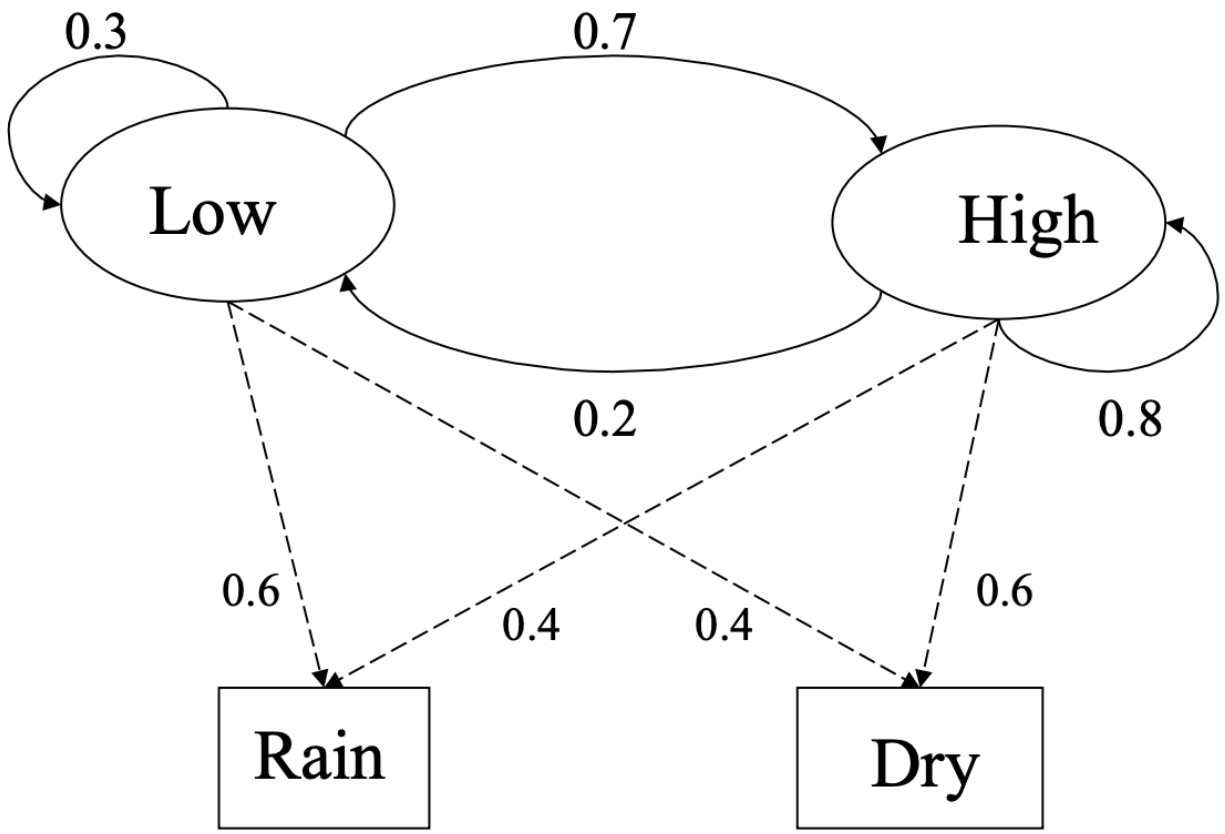
- Nodes describing states connected by edges reflecting the likelihood of a transition from one state to the next

- **States**: Dry, Rain
- **Transition Probabilities**:
  - P(Rain|Rain)=0.3
  - P(Rain|Dry)=0.2



Hidden Markov Model

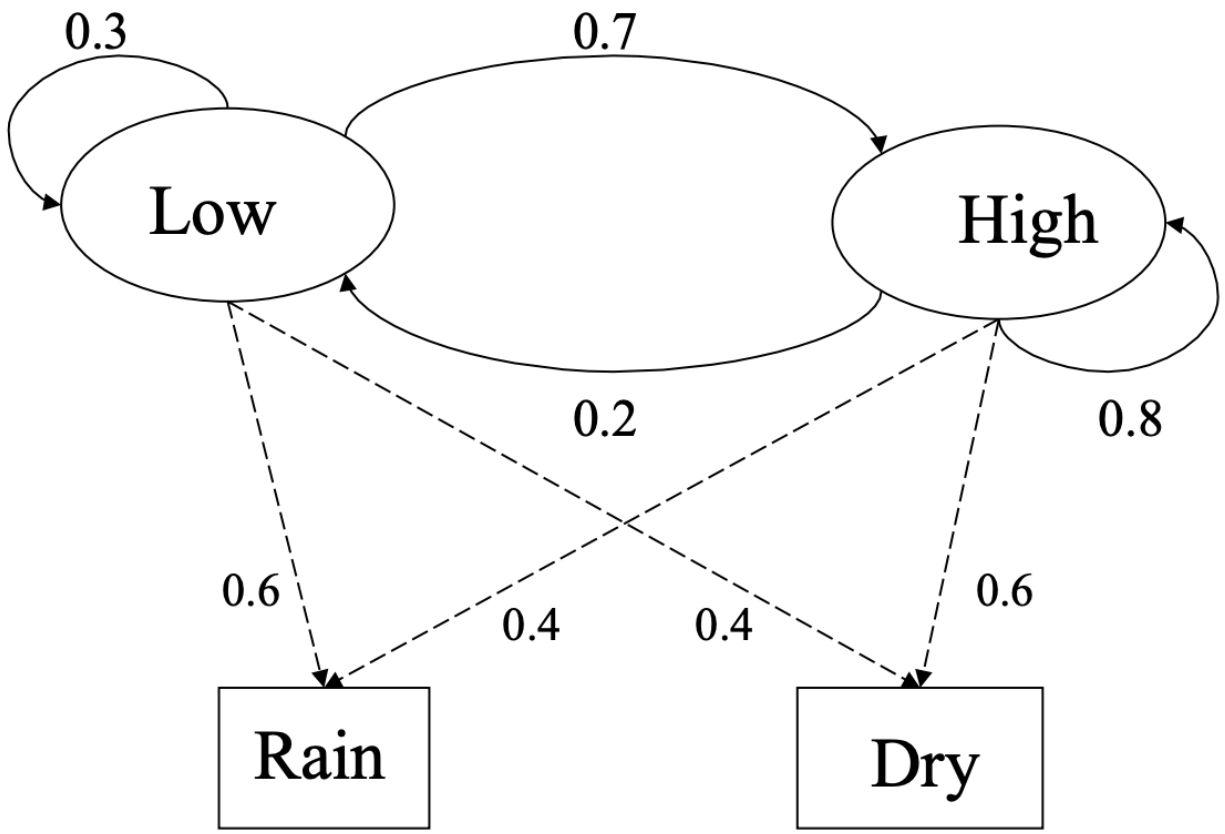
- In a plain Markov chain, you can directly see the state you're in at each time step
- In a Hidden Markov Model, the true state of the system is not directly observable
- What you do see are observations (outputs) that are caused by the hidden state



Hidden Markov Model

- Markov Model in which states randomly generate visible state

- **States**: Low, High Pressure
- **Observations**: Rain, Dry
- **Transition Probabilities**:
  - P(High | Low)=0.7
- **Observation Probabilities**:
  - P(Dry | Low)=0.4



What is "Artificial Intelligence"

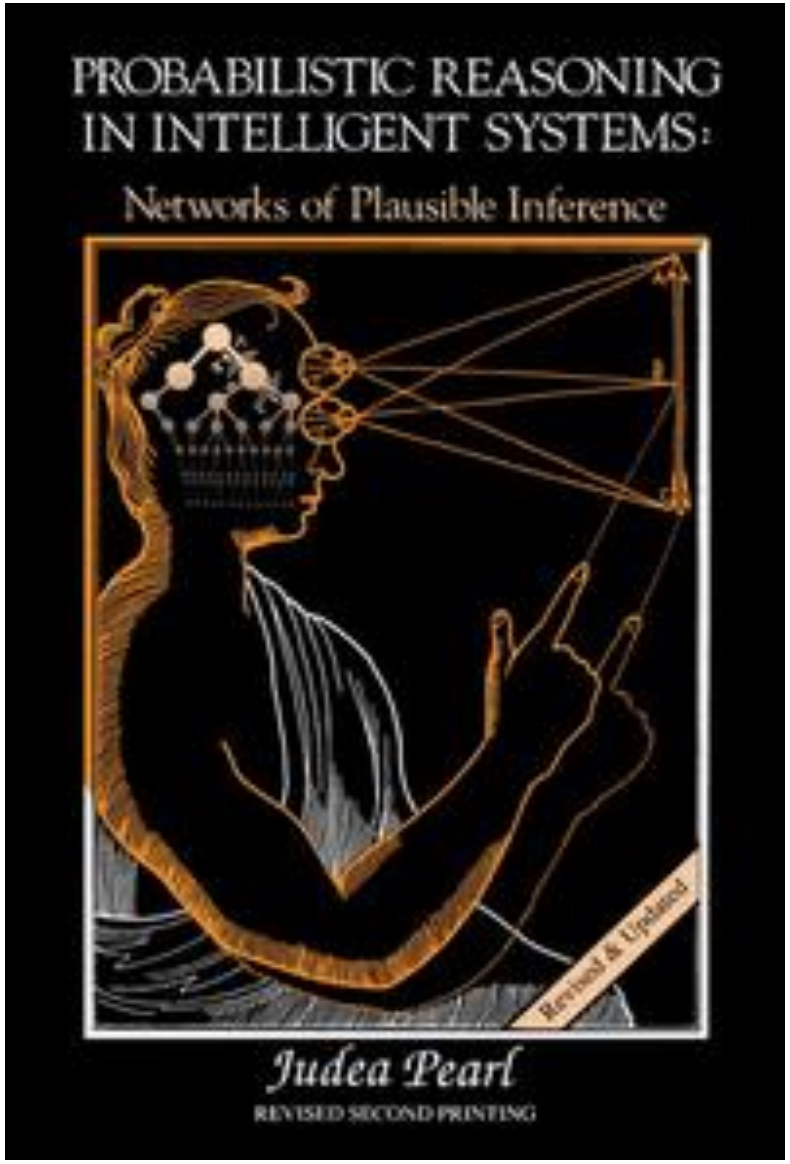
- AI is a multifaceted phenomenon, much like intelligence

- Appears to include:
  - Ability to learn
  - Ability to plan/reason
  - Ability to communicate
  - **Ability to reason probabilistically**

Ability to Probabilize

- Dragon Dictate 1.0 Speech Recognizer minimized the probability of error using a new technology called a **hidden Markov model**

- Pearl's **Bayesian Networks** showed how to extend this idea to every problem in AI; introduced math for causal reasoning

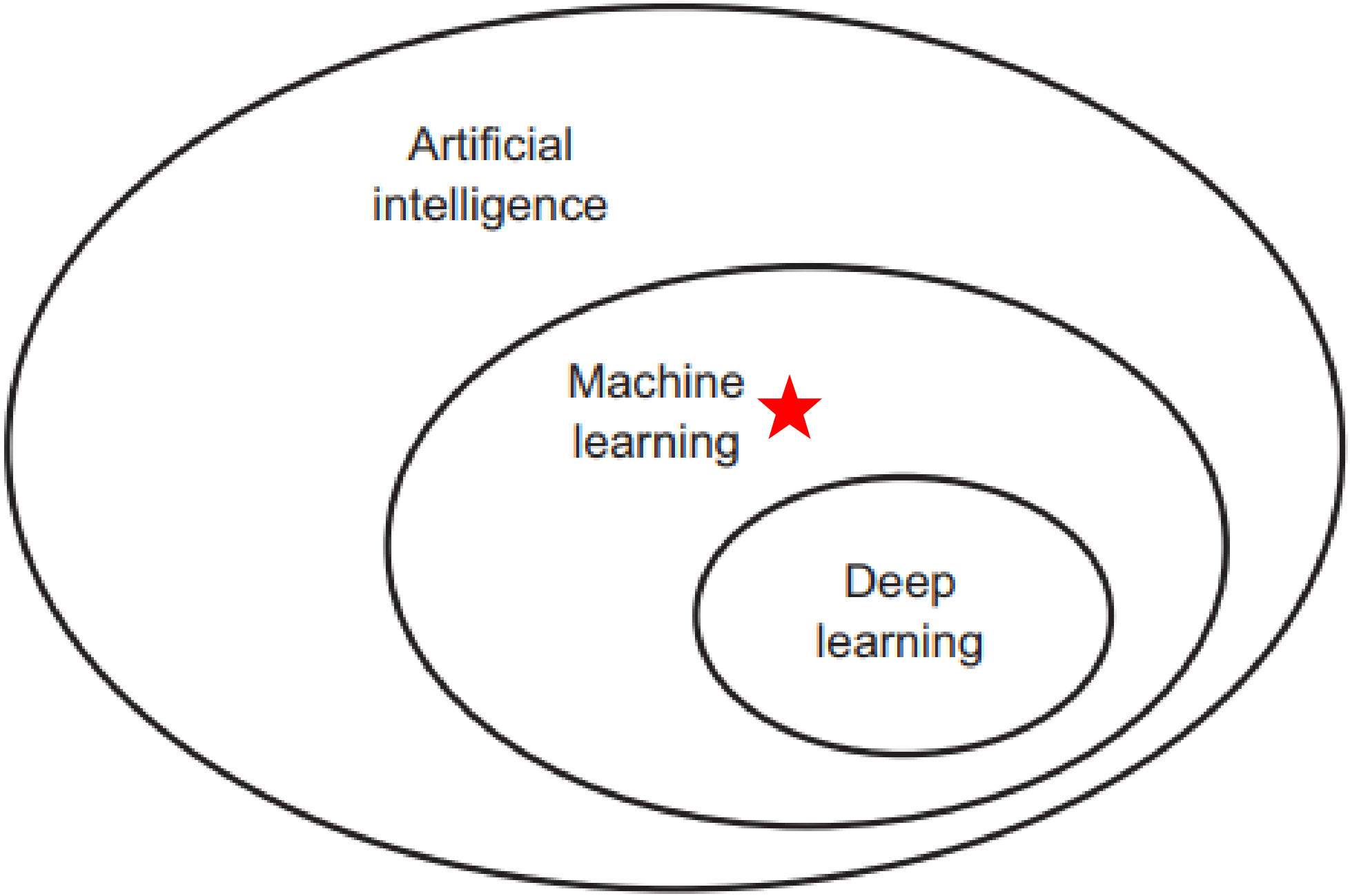


Key Research Areas of Modern AI

- **Planning and Search** – Problem solving based on cognitive science
- **Knowledge Representation** – Store and manipulate information
- **Automated Reasoning** – Use stored information to draw conclusions
- **Machine Learning** – Extrapolate patterns from data
- **Natural Language Processing** – Communicate with the machine
- **Computer Vision** – Processing visual information
- **Robotics** – Autonomy, manipulation, full integration of AI capabilities

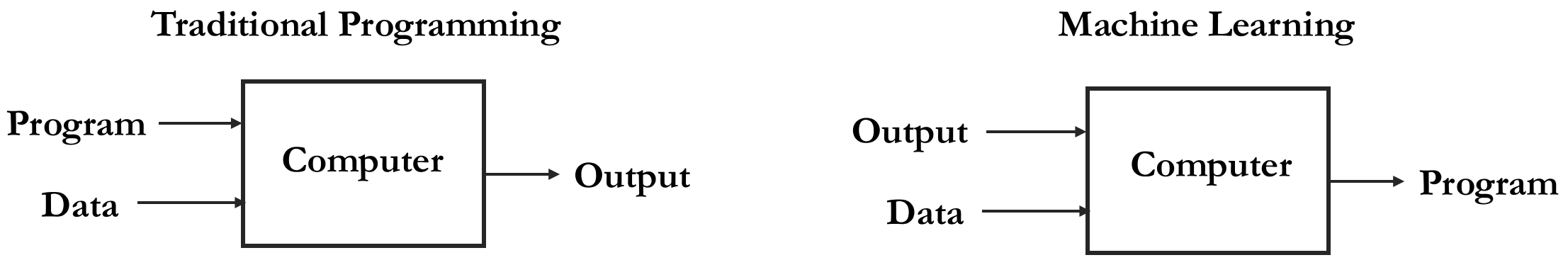
Outline

- Foundations of Modern AI
- Machine Learning
- Deep Learning
- MOWL


Artificial
intelligence
Machine
learning
Deep
learning

Machine Learning

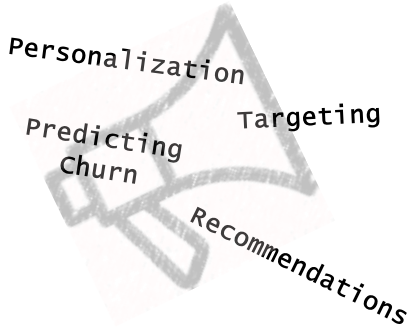
- **Machine learning** is a branch of artificial intelligence based on the idea that systems can learn from data, identify patterns and make decisions with minimal human intervention



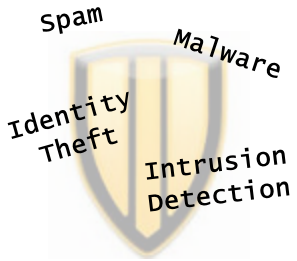
Machine Learning Successes

Web Search

Finance

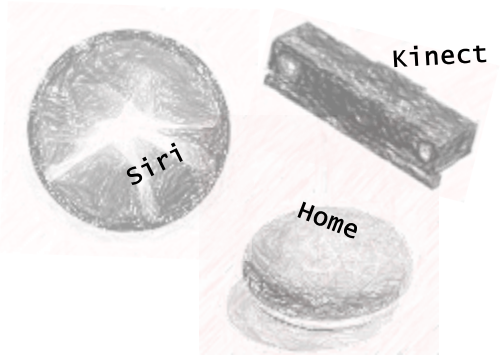


Marketing &
E-commerce



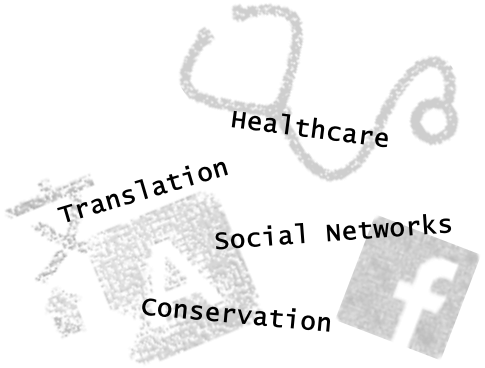
Abuse / Security

Robotics



Digital Assistants

Games



Many Others

Major Events

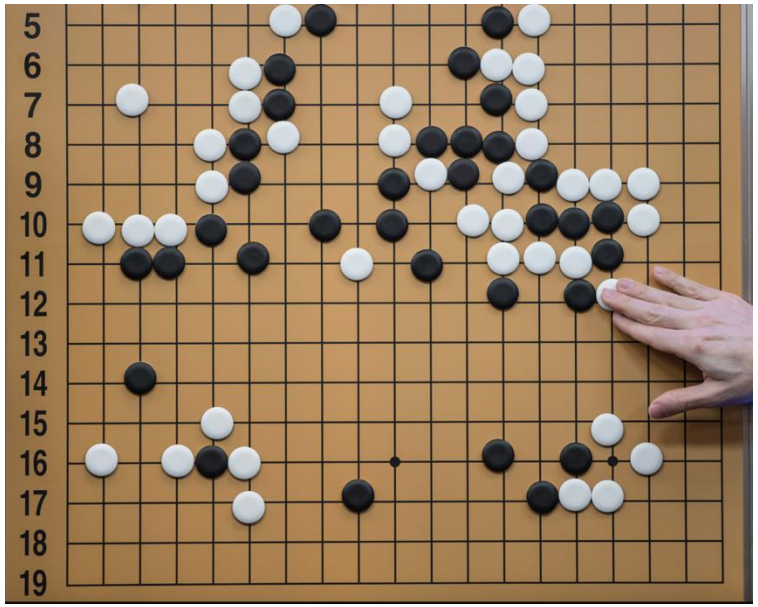
- 1997: IBM's **Deep Blue** became the first computer chess-playing system to defeat a reigning world chess champion when it beat Garry Kasparov

- 2011: **Watson** competed on the game show Jeopardy! against two human competitors and won.

Major Events

- 2014: Facebook developed **DeepFace**, software algorithm that can recognize individuals on photos with an accuracy of 97.35%

- 2016: **AlphaGo** defeated one of the best Go players, Lee Sedol



Major Events

- 2020: OpenAI creates **GPT-3** which is able generate human-like text in zero-prompt contexts

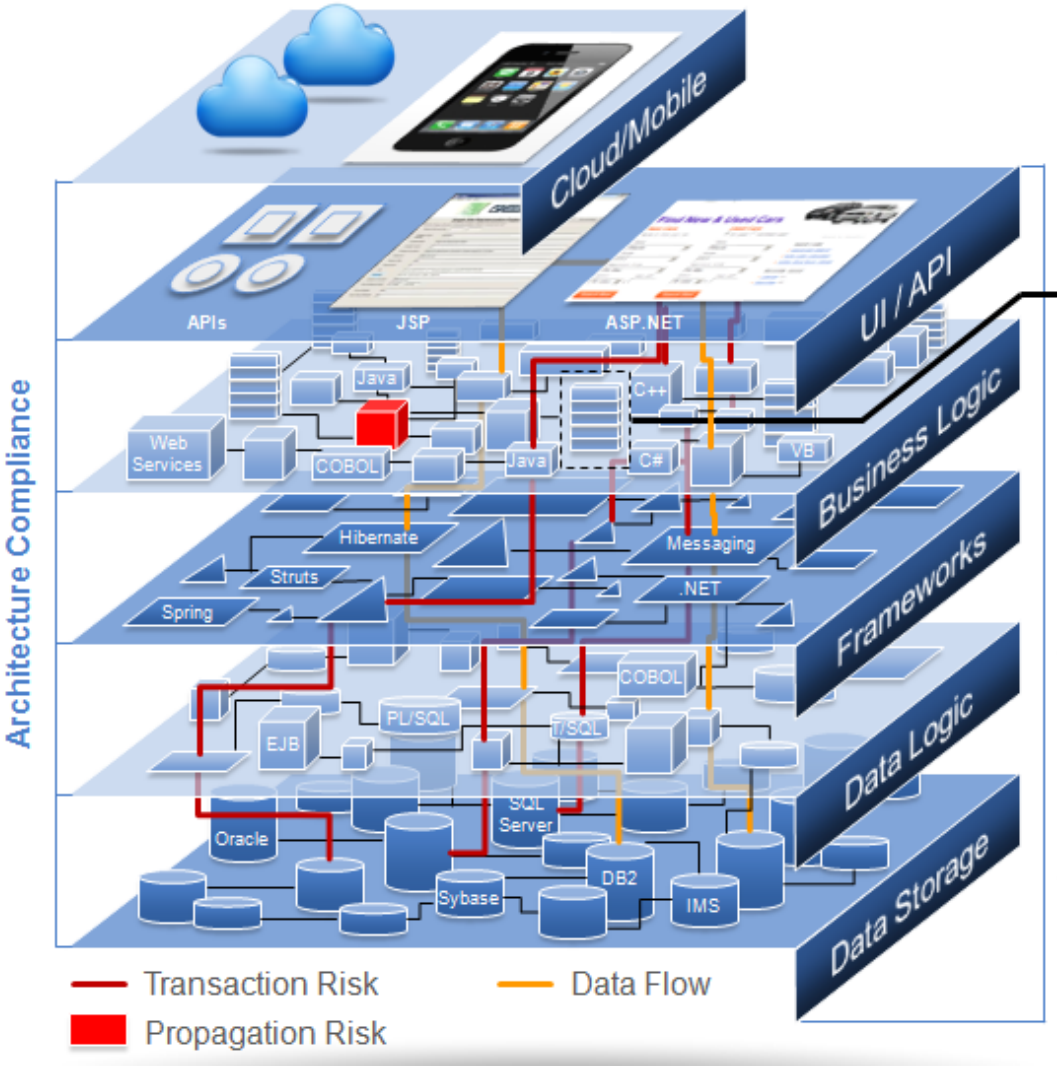
- 2021: OpenAI releases **DALL-E**, a variant of GPT-3 focused on image processing and production

Major Events

- 2022: **ChatGPT**

- 2023: **GPT-4**, **Gemini**, and Computer Vision

Role of Machine Learning



Program Unit Level

- Code style & layout
- Expression complexity
- Code documentation
- Class or program design
- Basic coding standards

Technology Level

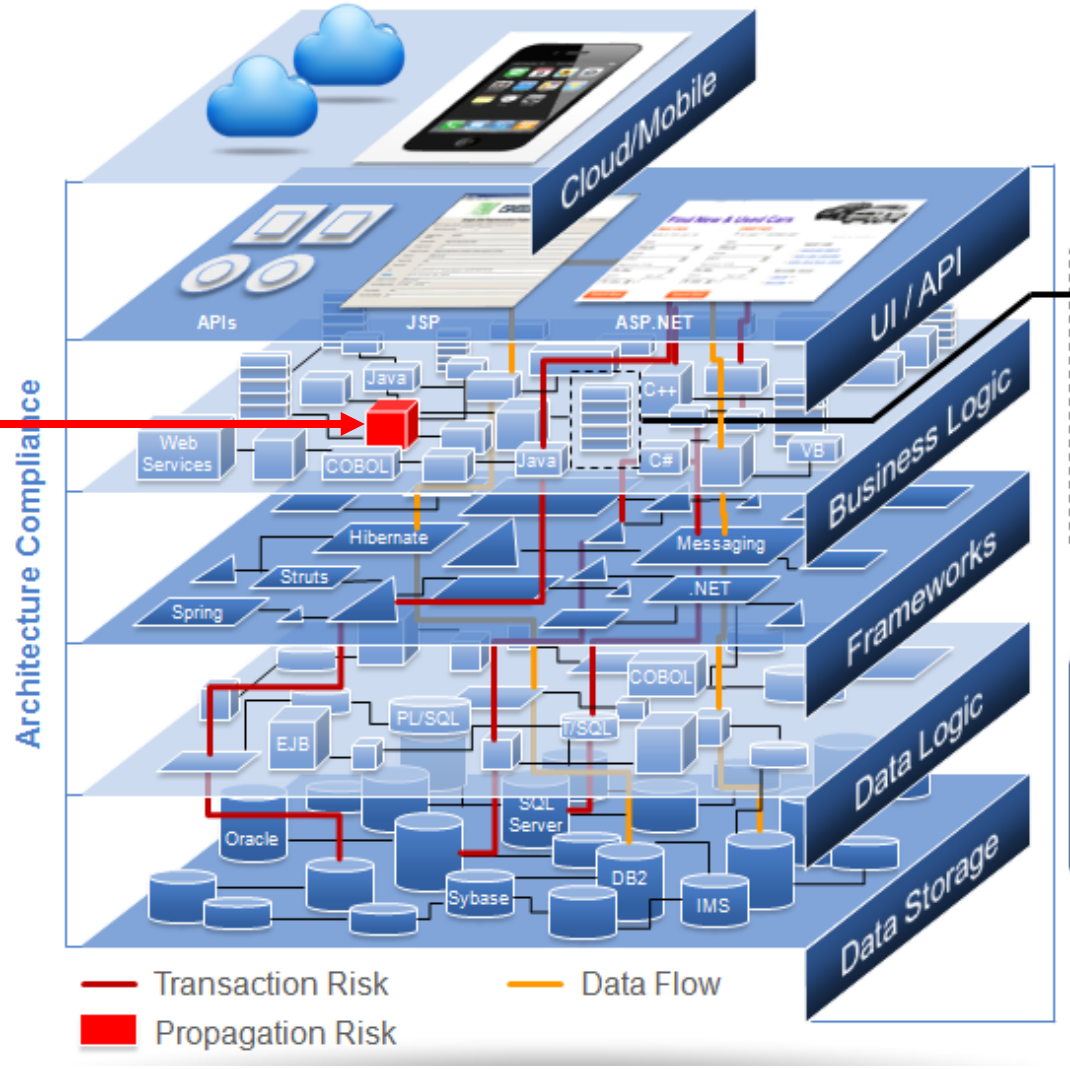
- Intra-technology architecture
- Intra-layer dependencies
- Intra-module communication
- Module complexity & cohesion
- Design & structure
- Inter-program invocation
- Security Vulnerabilities

System Level

- Integration quality
- Architectural compliance
- Risk propagation simulation
- Application security
- Resiliency checks
- Transaction integrity
- Function point & EFP measurement
- Effort estimation
- Data access control
- SDK versioning
- Calibration across technologies

Role of Machine Learning

Machine Learning



Program Unit Level

- Code style & layout
- Expression complexity
- Code documentation
- Class or program design
- Basic coding standards

Technology Level

- Intra-technology architecture
- Intra-layer dependencies
- Intra-module communication
- Module complexity & cohesion
- Design & structure
- Inter-program invocation
- Security Vulnerabilities

System Level

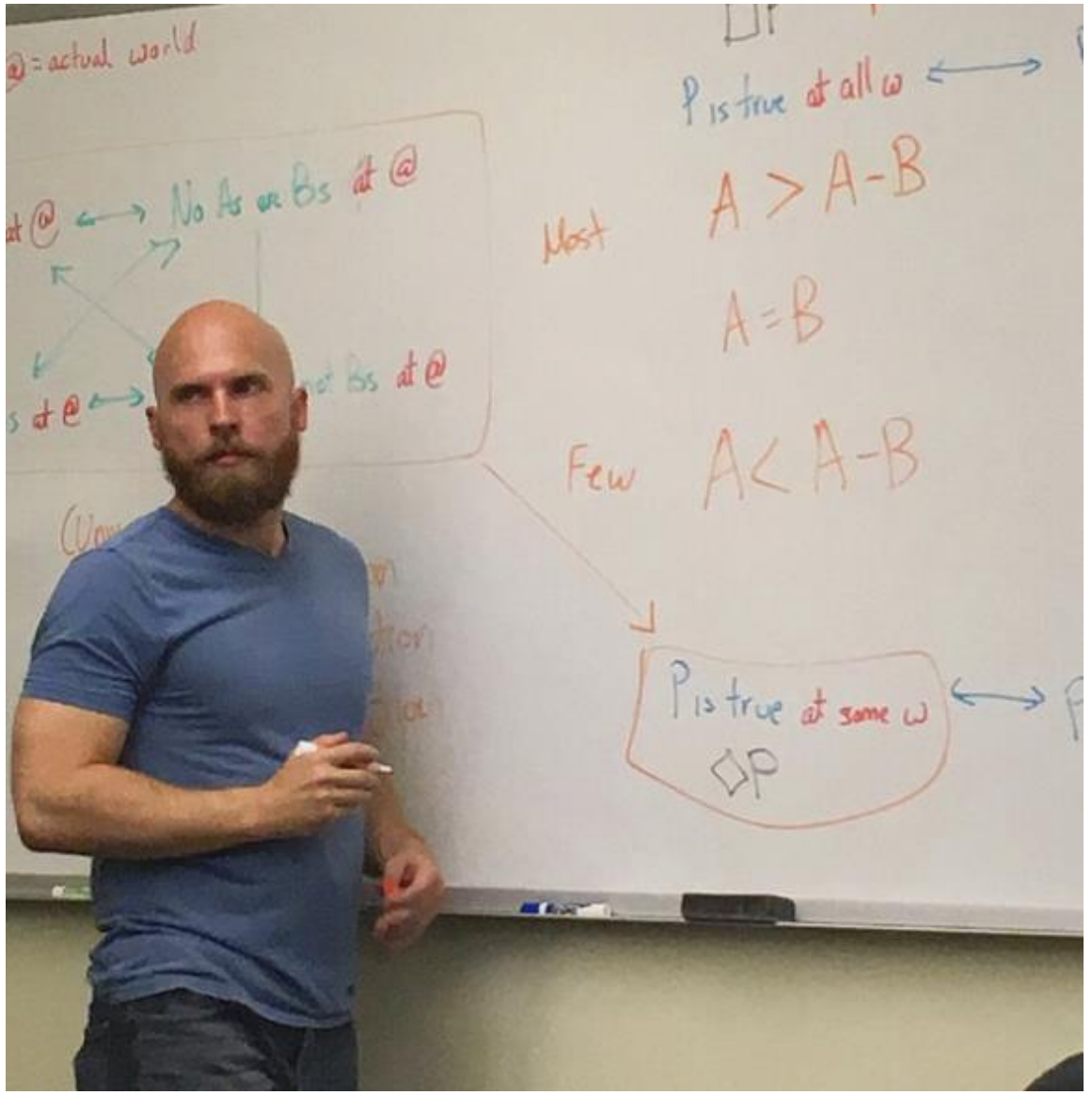
- Integration quality
- Architectural compliance
- Risk propagation simulation
- Application security
- Resiliency checks
- Transaction integrity
- Function point & EFP measurement
- Effort estimation
- Data access control
- SDK versioning
- Calibration across technologies

Learning & Adaptation

- "Modification of a behavioral tendency by expertise." (Webster 1984)
- "A learning machine, broadly defined is any device whose actions are influenced by past experiences." (Nilsson 1965)
- "Any change in a system that allows it to perform better the second time on repetition of the same task or on another task drawn from the same population." (Simon 1983)
- "An improvement in information processing ability that results from information processing activity." (Tanimoto 1990)

How Do We Learn?

- Engaging with environment
- Mimicking, reading, reporting
- Tutoring
- Reinforcement when correct
- Correction when incorrect
- Analogy
- Applying knowledge to new problems
- Self-reflection
- Deduction, induction, abduction



Learning from Data

- A machine-learning model transforms its input data into meaningful outputs, i.e. it "learns" from exposure to known examples

- The central problem in machine learning and deep learning is to to learn **useful representations** of the input data

- Which get us closer to expected output

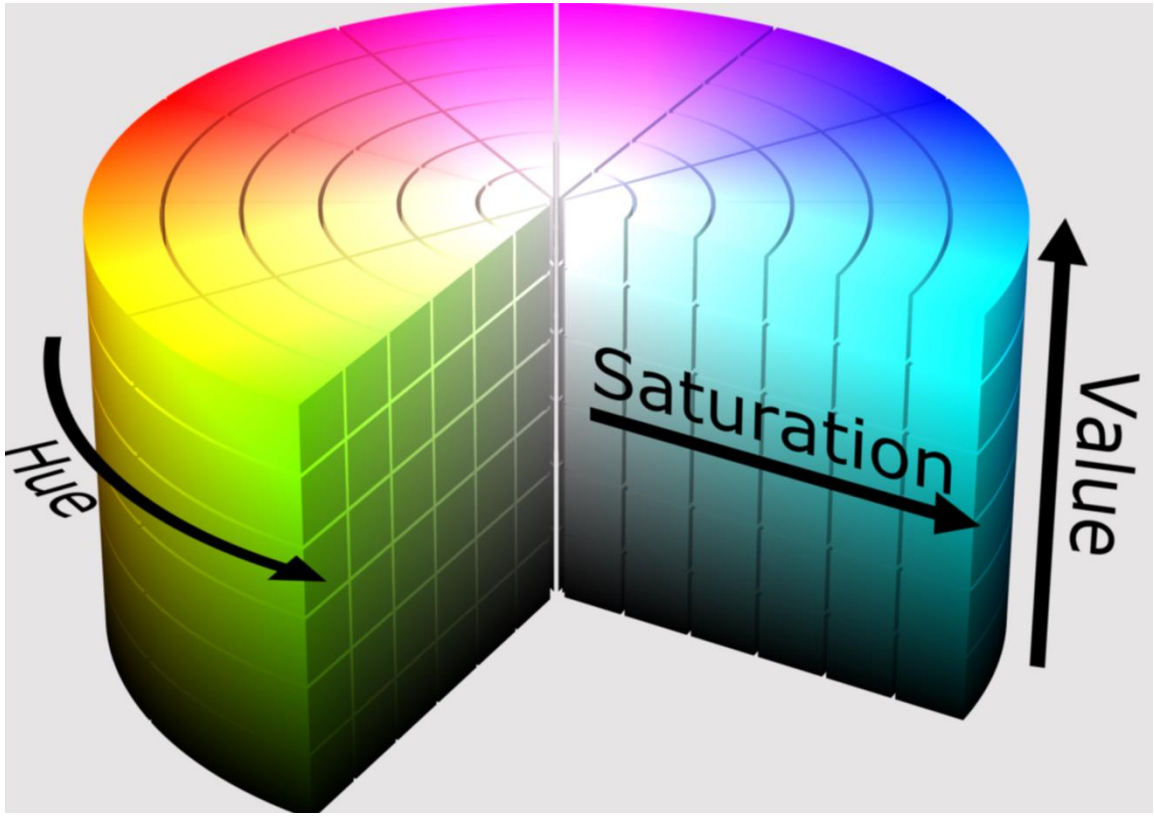
Data Representations

- An **encoding of data**, e.g. a color image encoded in the RGB (**red**-**green**-**blue**) format or in the HSV (hue-saturation-value) format

- Tasks difficult with one representation can be easy with another

- The task "select all **red** pixels in the image" is simpler in RGB, whereas "make the image less **saturated**" is simpler in HSV

Data Representations



"select all **red** pixels in the image"

"make the image less **saturated**"

Learning from Data

- Machine learning involves searching for useful representations of some input data, within a **predefined space of possibilities**, using guidance from a **feedback signal**

- This **simple idea** allows for approaching potential automated solutions a broad range of intellectual tasks, from speech recognition to autonomous car driving

Learning Paradigms

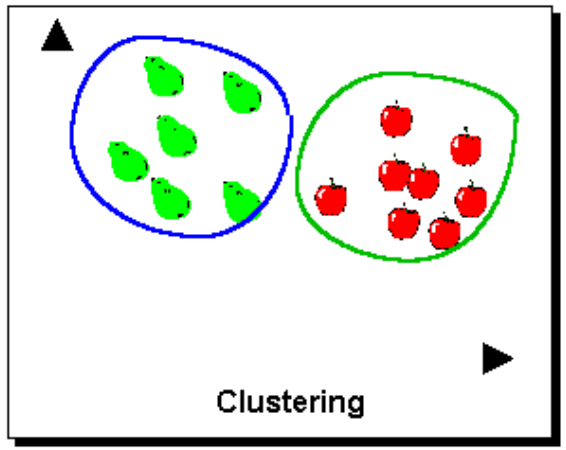
- **Supervised Learning** - inputs and correct outputs are provided by an agent, often in the form of labeled data and training data

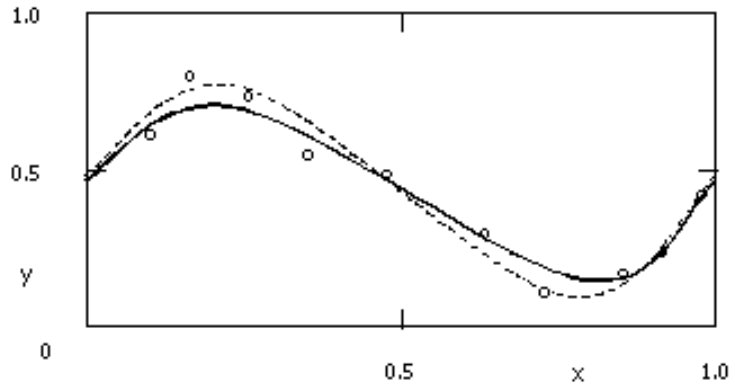
- **Unsupervised Learning** - no guidance and no hint as to what counts as correct output is given

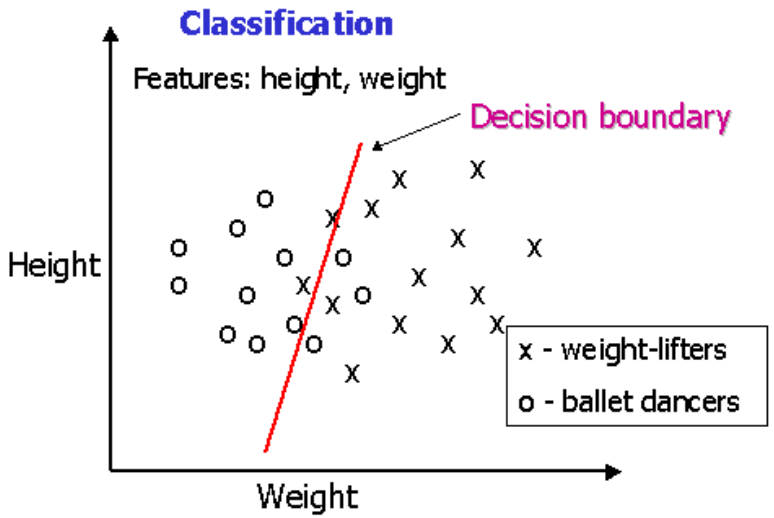
- **Reinforced Learning** – characteristically involves aspects of reward or punishment for correct of incorrect outputs, respectively

ML Models

- Classification
- Regression
- Clustering
- Time series analysis
- Q-Learning



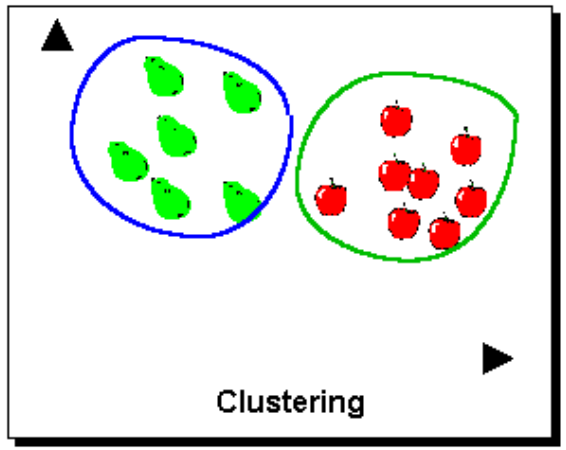


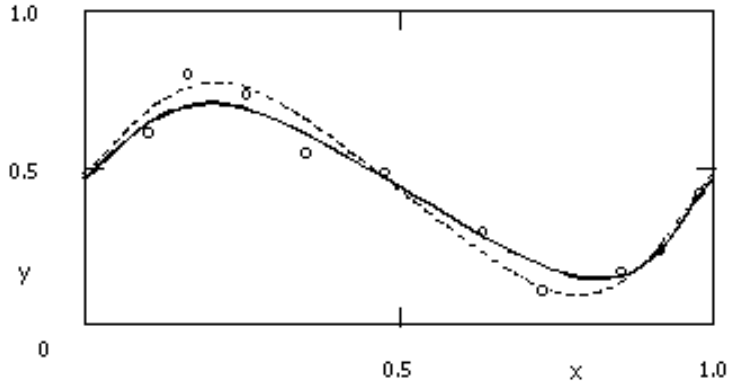


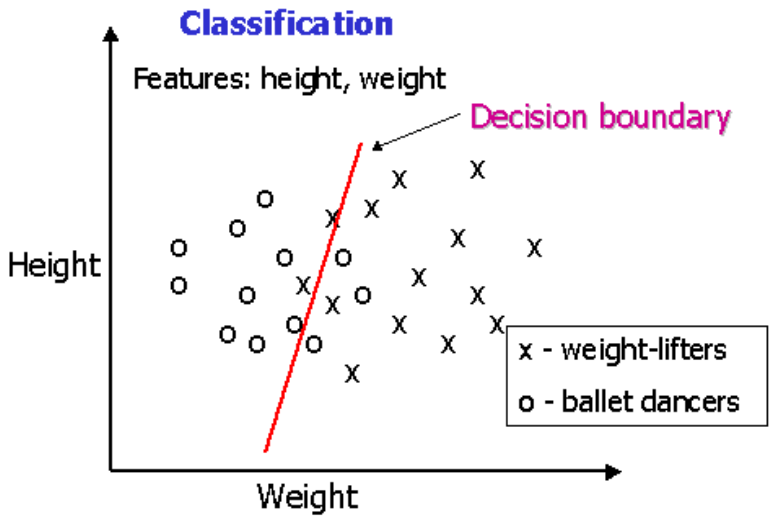
ML Models

- **Classification**
- Regression
- Clustering
- Time series analysis
- Q-Learning

SUPERVISED

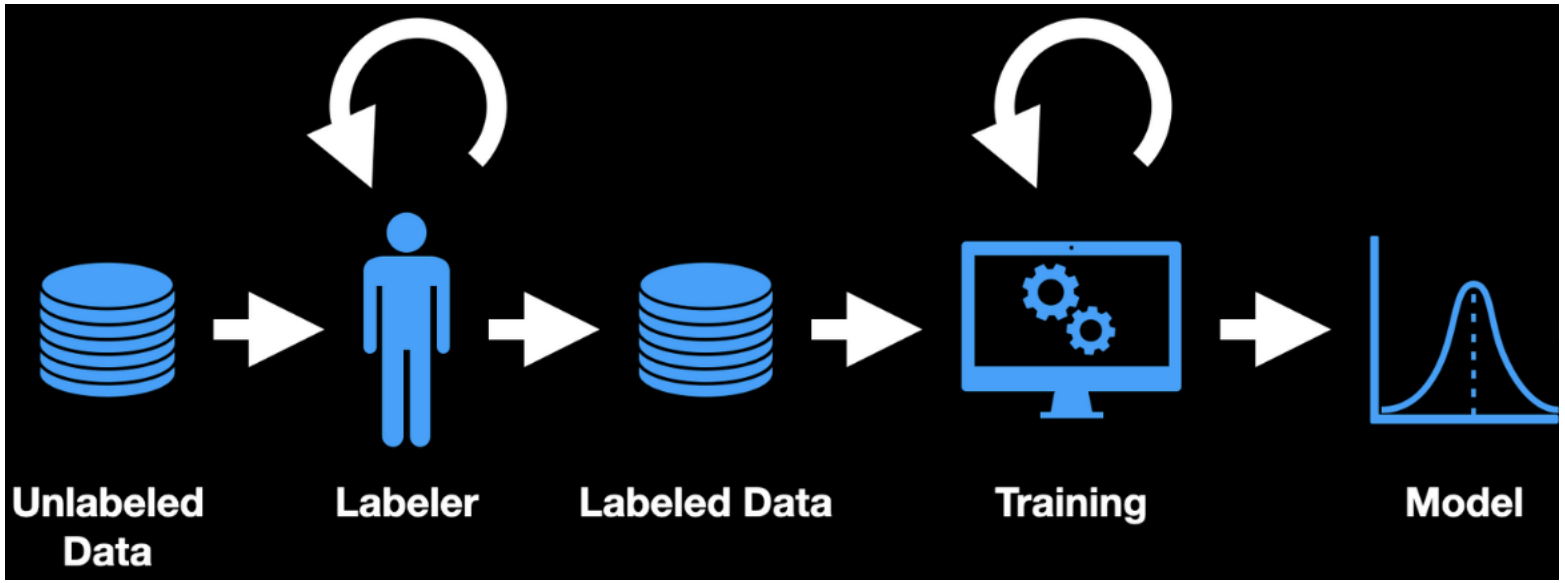






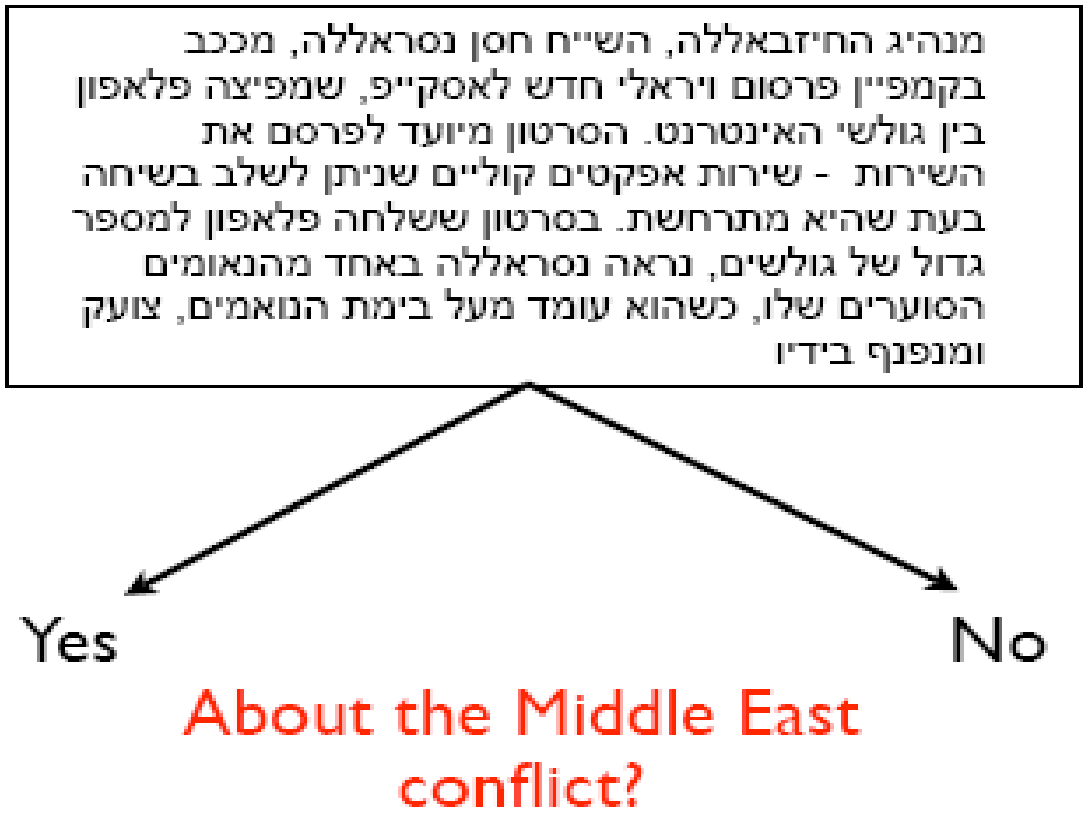
Supervised Learning

- Use pre-defined labels on data and rules to predict the output for future inputs



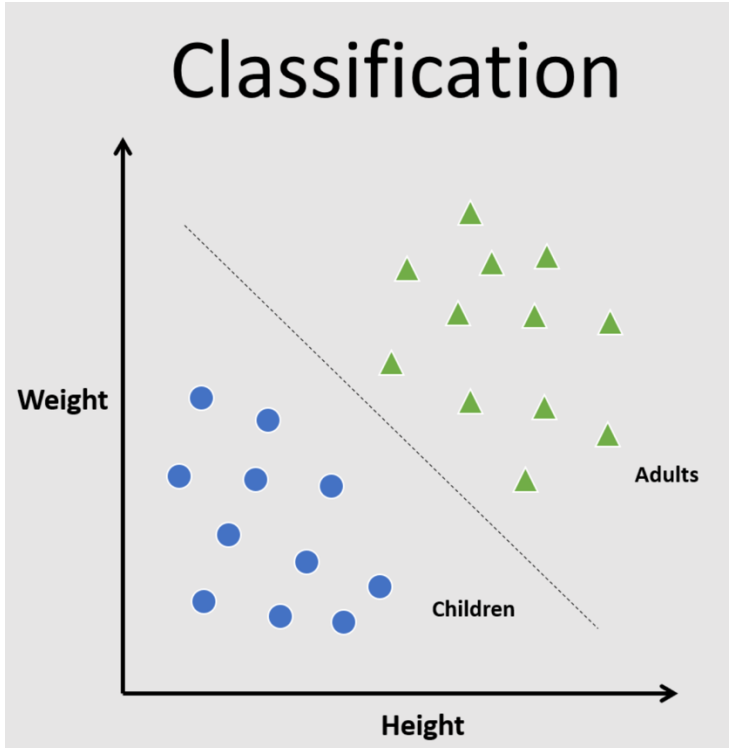
Classification

- Assign items to one of a set of predefined classes of objects based on a set of observed features



Classification

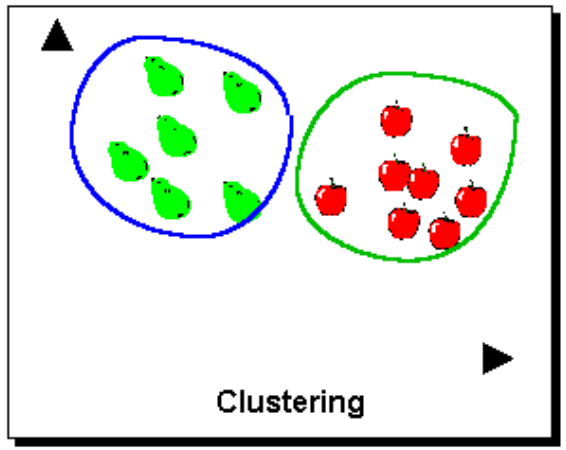
- Assign items to one of a set of predefined classes of objects based on a set of observed features

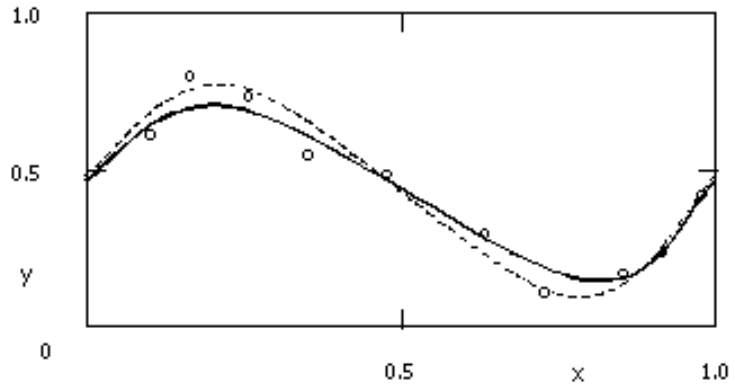


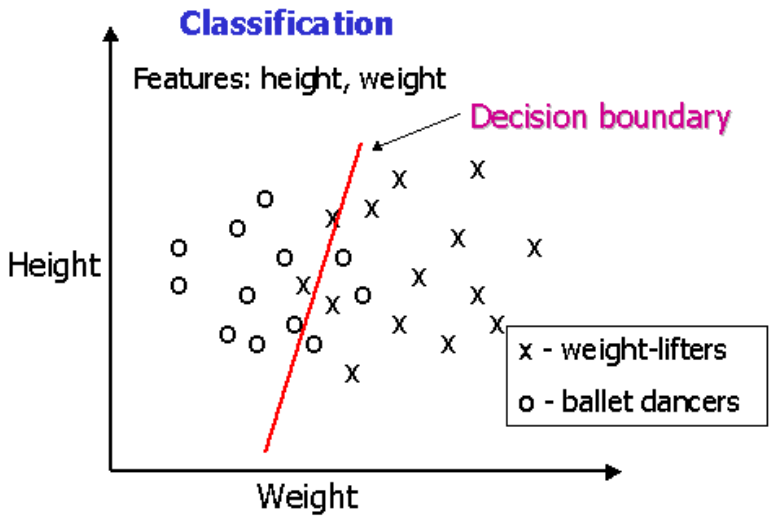
ML Models

- Classification
- **Regression**
- Clustering
- Time series analysis
- Q-Learning

SUPERVISED



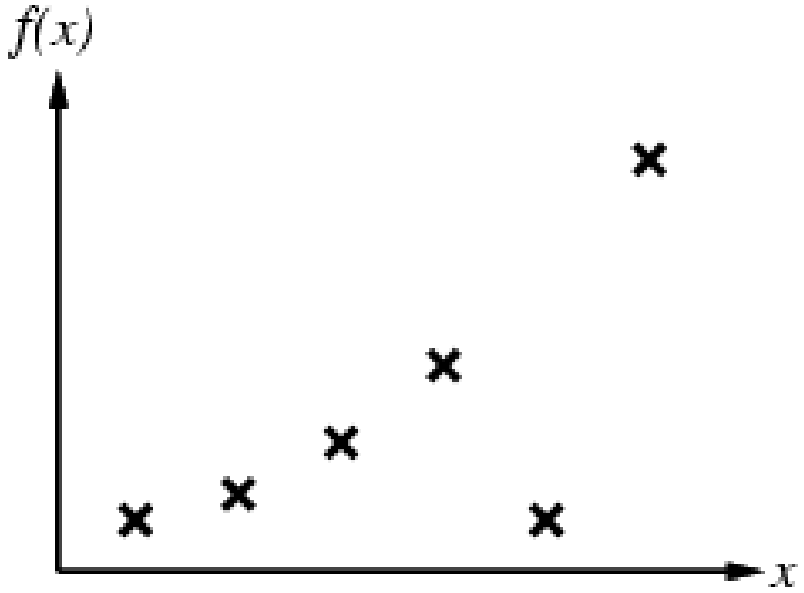




Inductive Learning

- Simplest form is to learn a function from examples

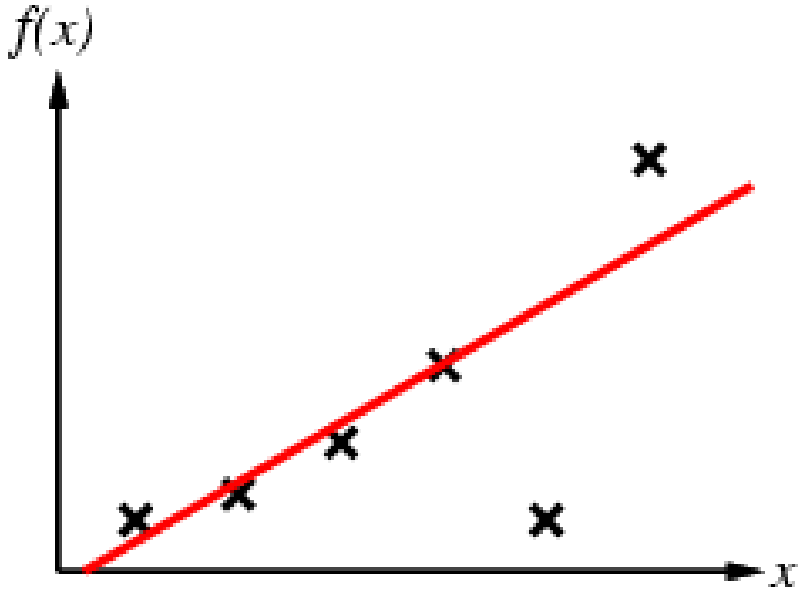
- Construct/adjust h to agree with f on training set e.g. curve fitting:



Inductive Learning

- Simplest form is to learn a function from examples

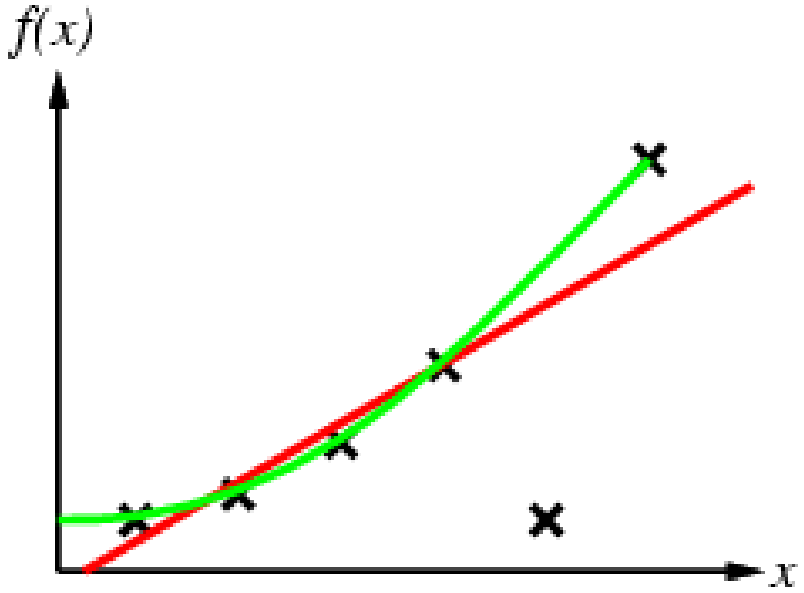
- Construct/adjust h to agree with f on training set e.g. curve fitting:



Inductive Learning

- Simplest form is to learn a function from examples

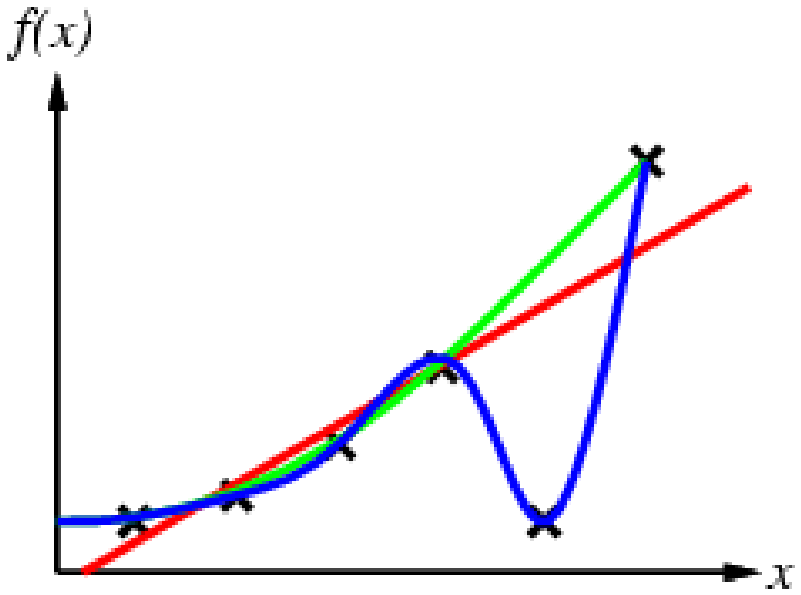
- Construct/adjust h to agree with f on training set e.g. curve fitting:



Inductive Learning

- Simplest form is to learn a function from examples

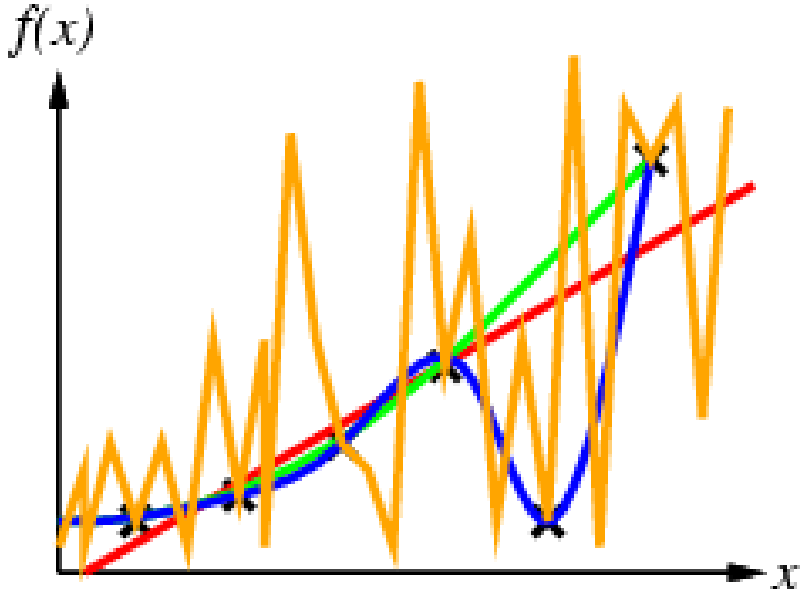
- Construct/adjust h to agree with f on training set e.g. curve fitting:



Inductive Learning

- Simplest form is to learn a function from examples

- Construct/adjust h to agree with f on training set e.g. curve fitting:

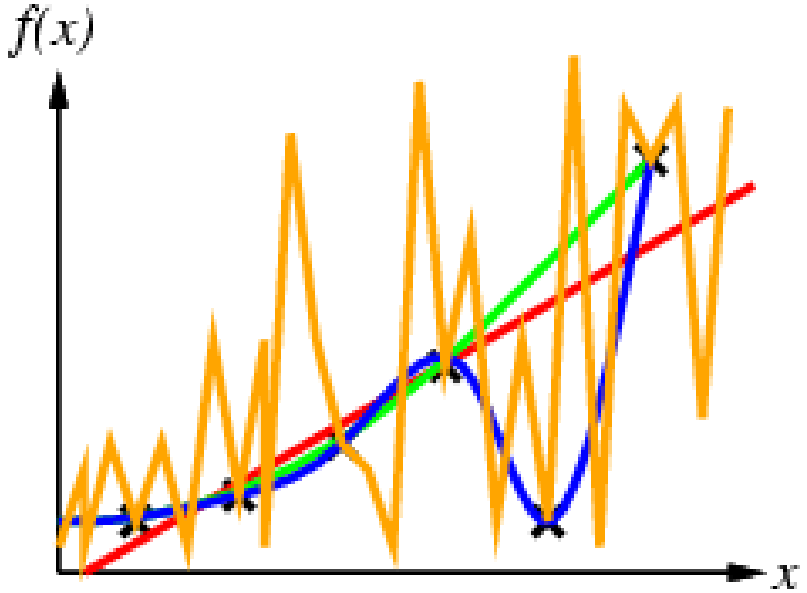


Inductive Learning

- Simplest form is to learn a function from examples

- Construct/adjust h to agree with f on training set e.g. curve fitting:

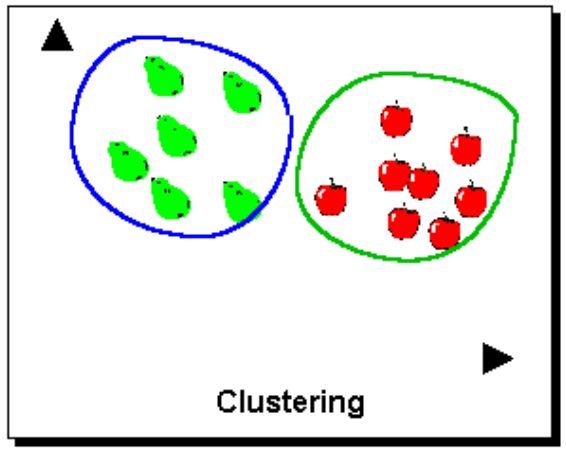
Ockham's Razor: prefer the simplest hypothesis consistent with data

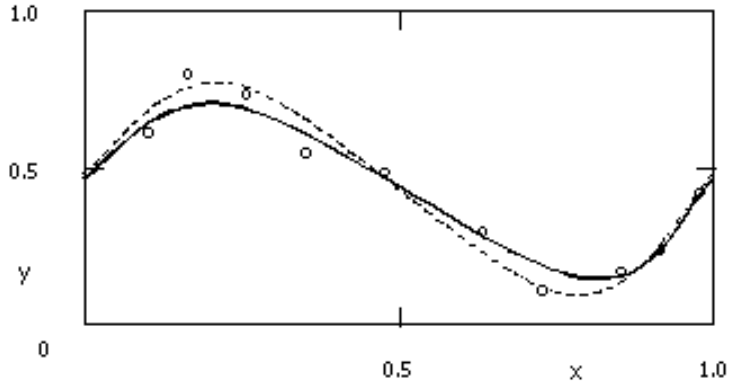


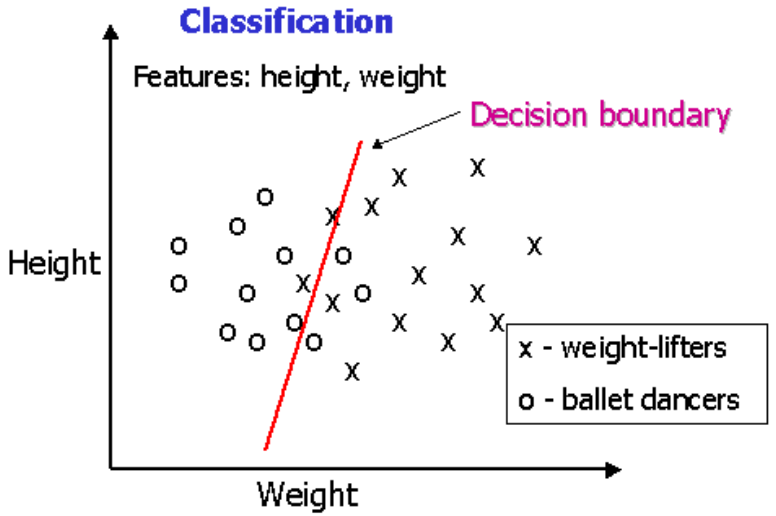
ML Models

- Classification
- Regression
- **Clustering**
- Time series analysis
- Q-Learning

UNSUPERVISED

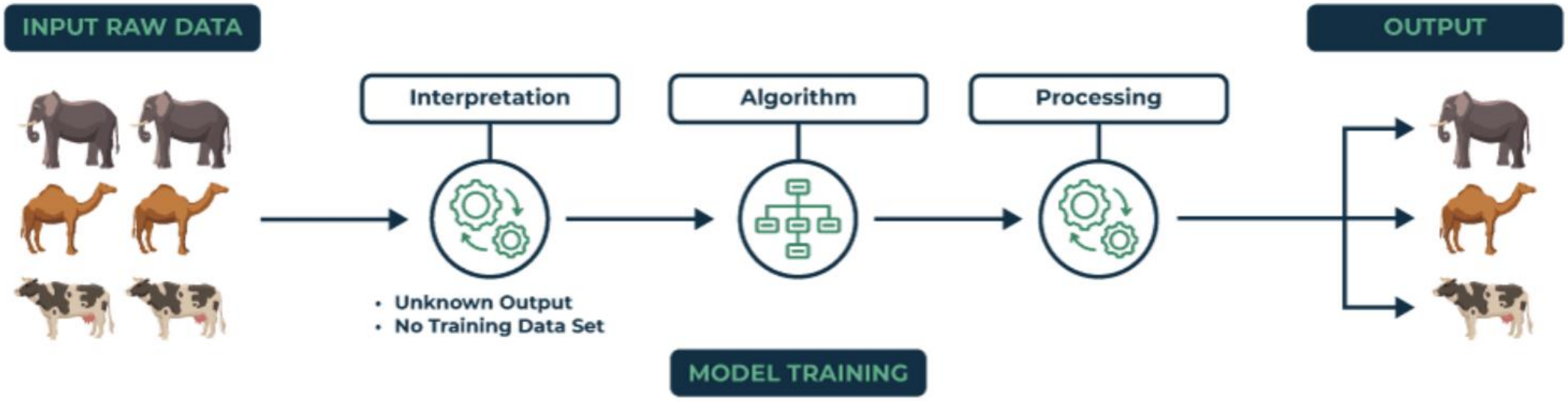






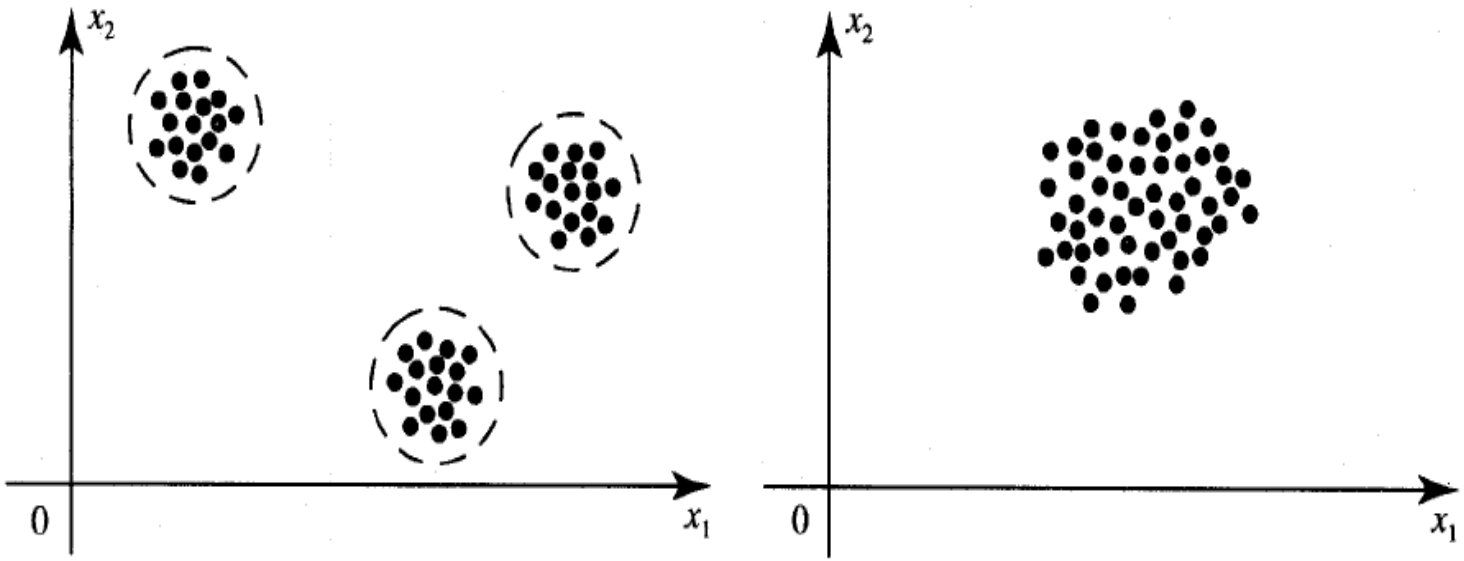
Unsupervised Learning

- Learning that involves no guidance and no hint about correct output is given; data is raw, i.e. no labels



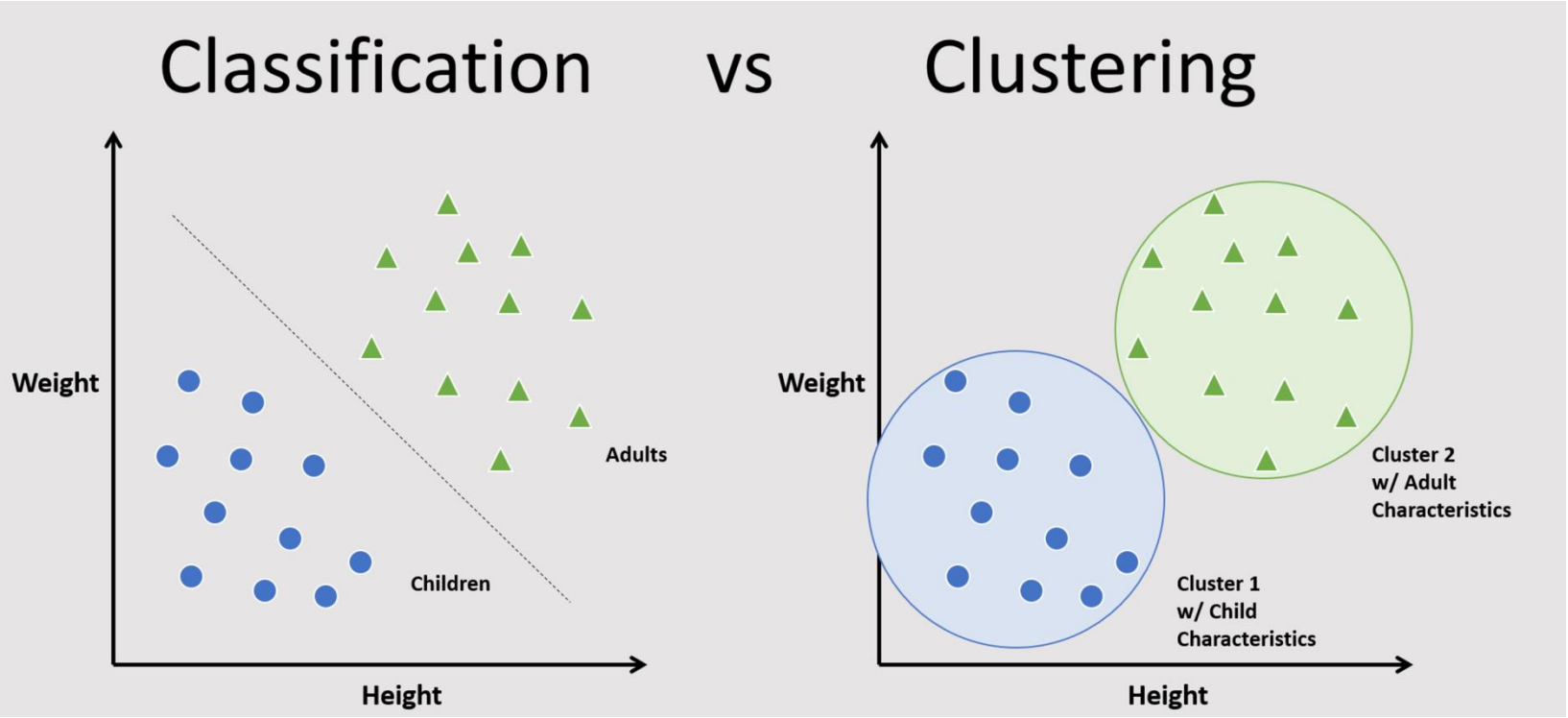
Unsupervised Learning

- How a fish or tadpole learns
- All similar input patterns are grouped together as clusters
- If a matching input pattern is not found a new cluster is formed



Clustering

- Place objects into meaningful groups based on similarity; aim to discover useful but unknown groups



Group Exercise

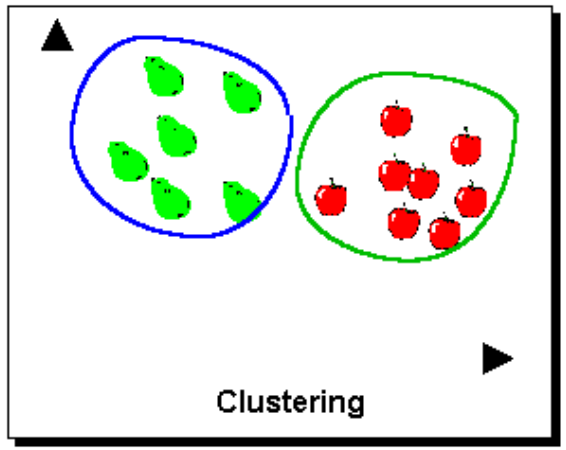
NAVIGATE TO THE FOLLOWING SITE AND EXPLORE CLUSTERING

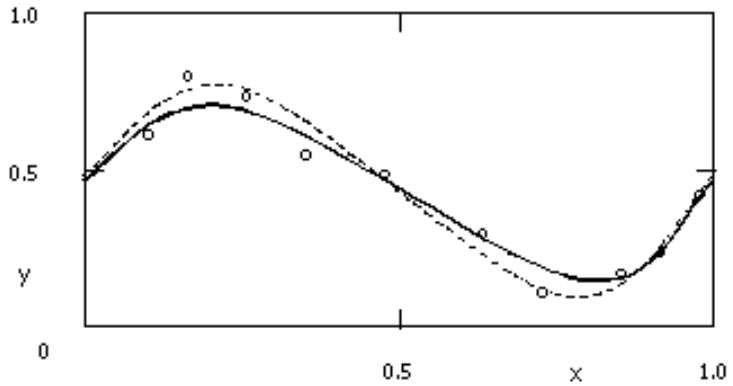
https://clustering-visualizer.web.app/

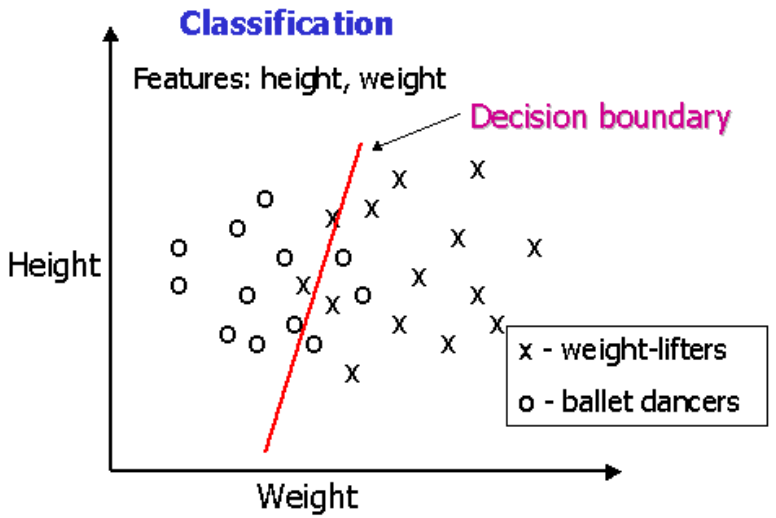
ML Models

- Classification
- Regression
- Clustering
- **Time series analysis**
- Q-Learning

(UN)SUPERVISED

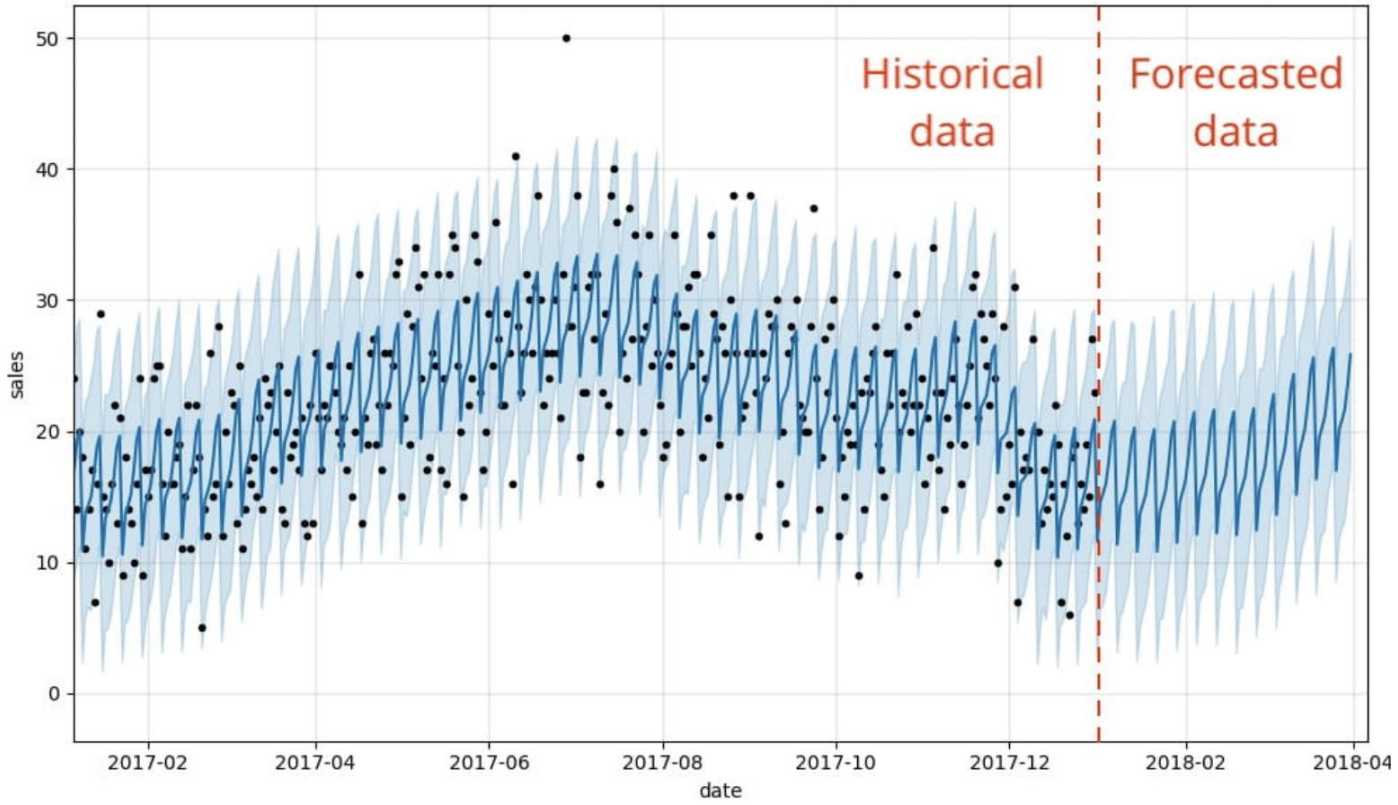






Time Series Analysis

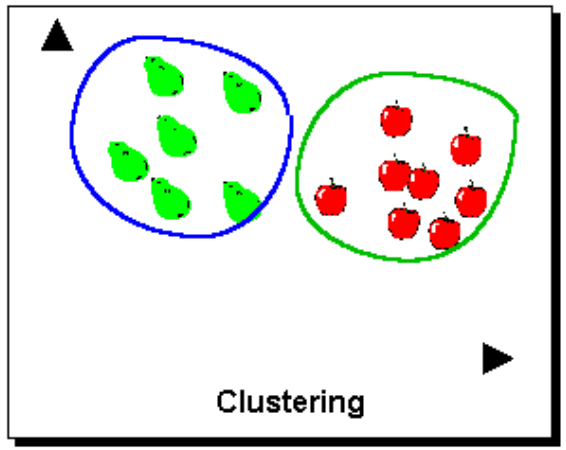
- Analyze data points collected over time to identify trends, patterns, and make forecasts or predictions

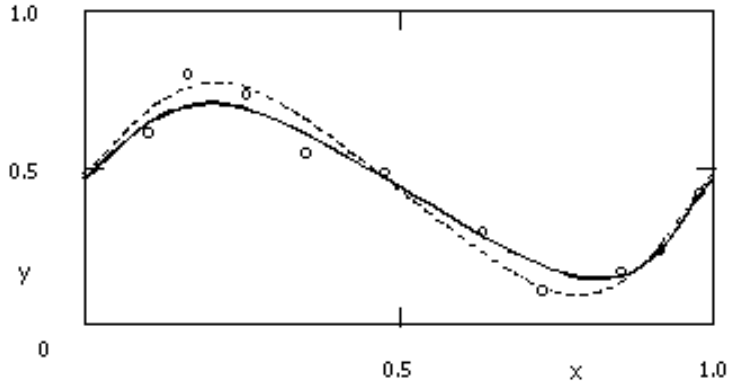


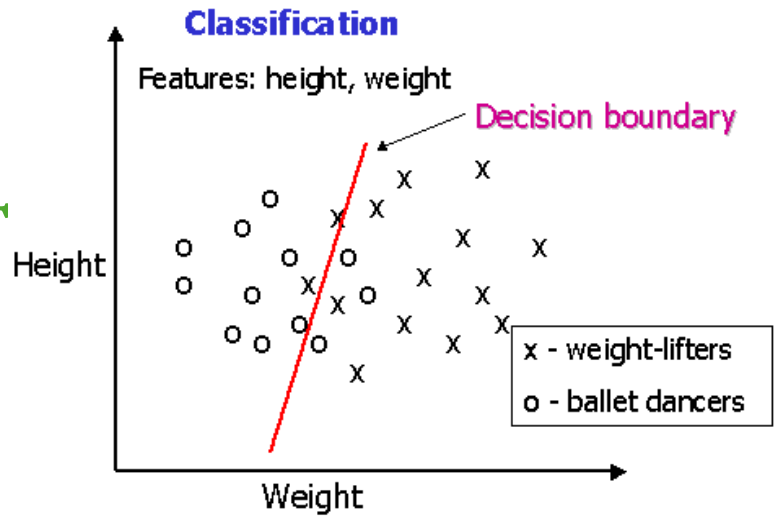
ML Models

- Classification
- Regression
- Clustering
- Time series analysis
- **Q-Learning**

REINFORCEMENT

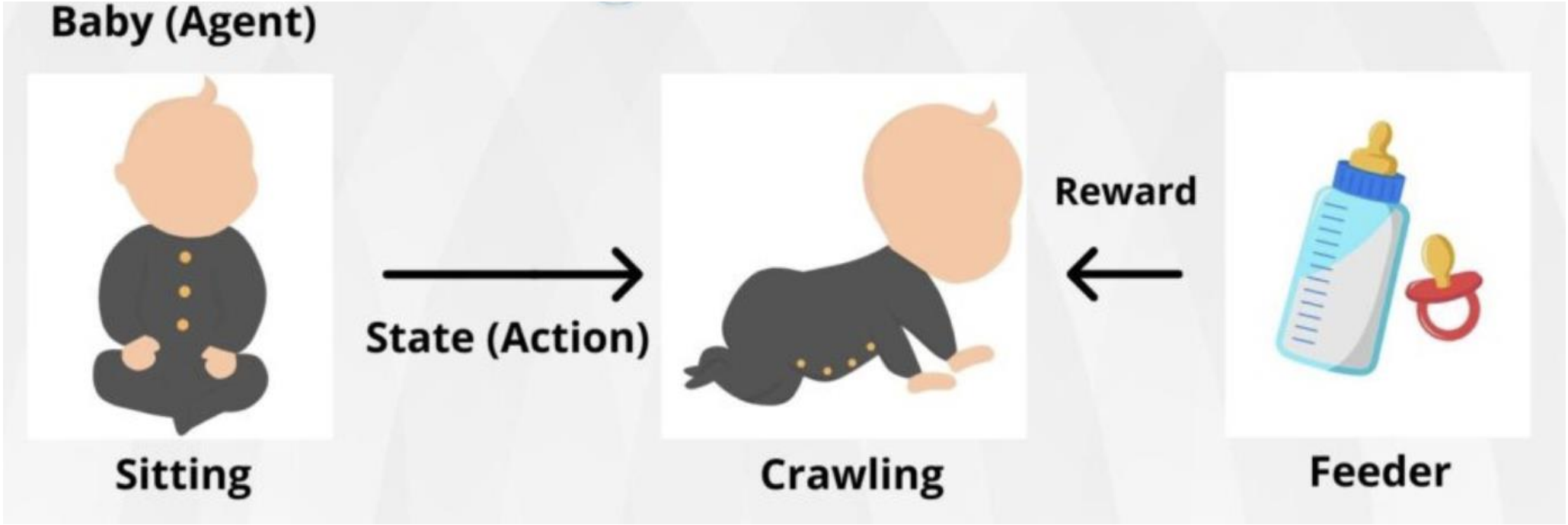






Reinforcement Learning

- Agent takes actions, receives feedback (rewards/penalties), and adjusts its strategy to improve performance over time



Q-Learning

- Teaches an agent to assign values to each action it might take, conditioned on the agent being in a particular state

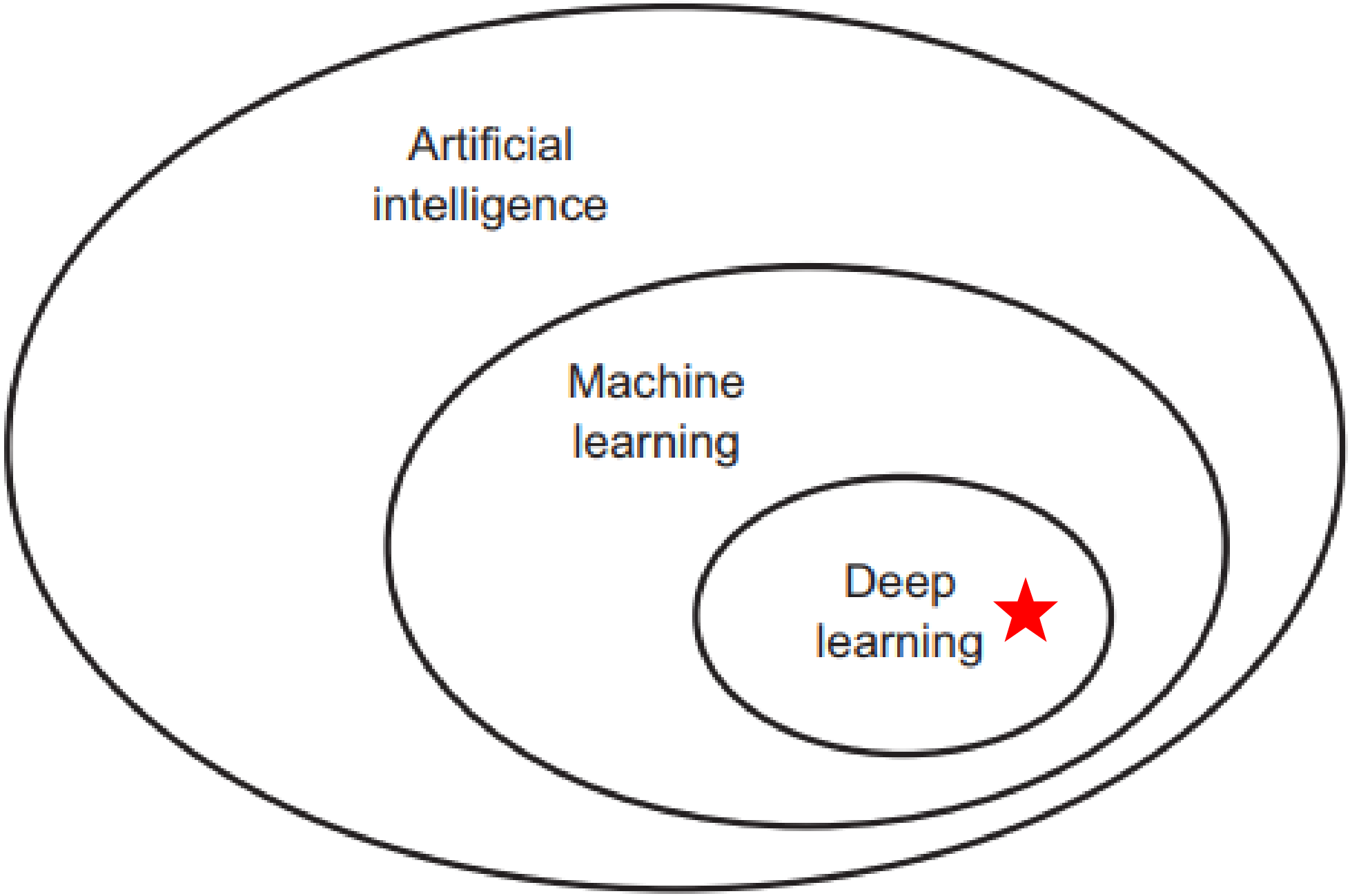
Group Exercise

NAVIGATE TO THE FOLLOWING LOCATION AND PLAY ARTBOT:

https://art-bot.net/

Outline

- Foundations of Modern AI
- Machine Learning
- Deep Learning
- MOWL


Artificial
intelligence
Machine
learning
Deep
learning

"Deep"

- **Deep learning** emphasizes learning successive layers of increasingly meaningful representations

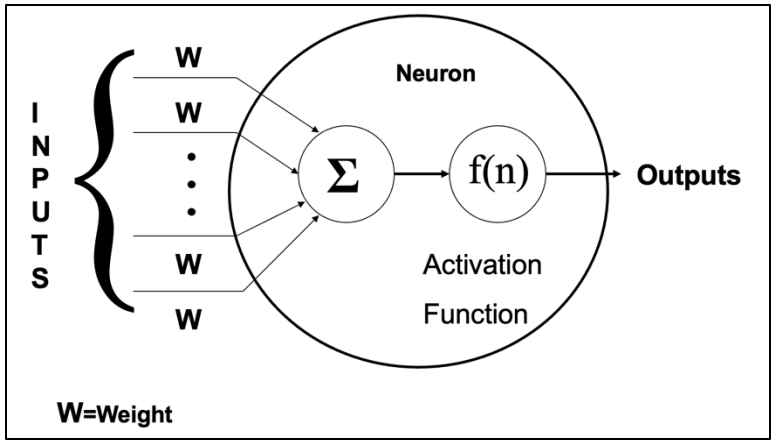
- It is "deep" not because some deeper understanding is achieved

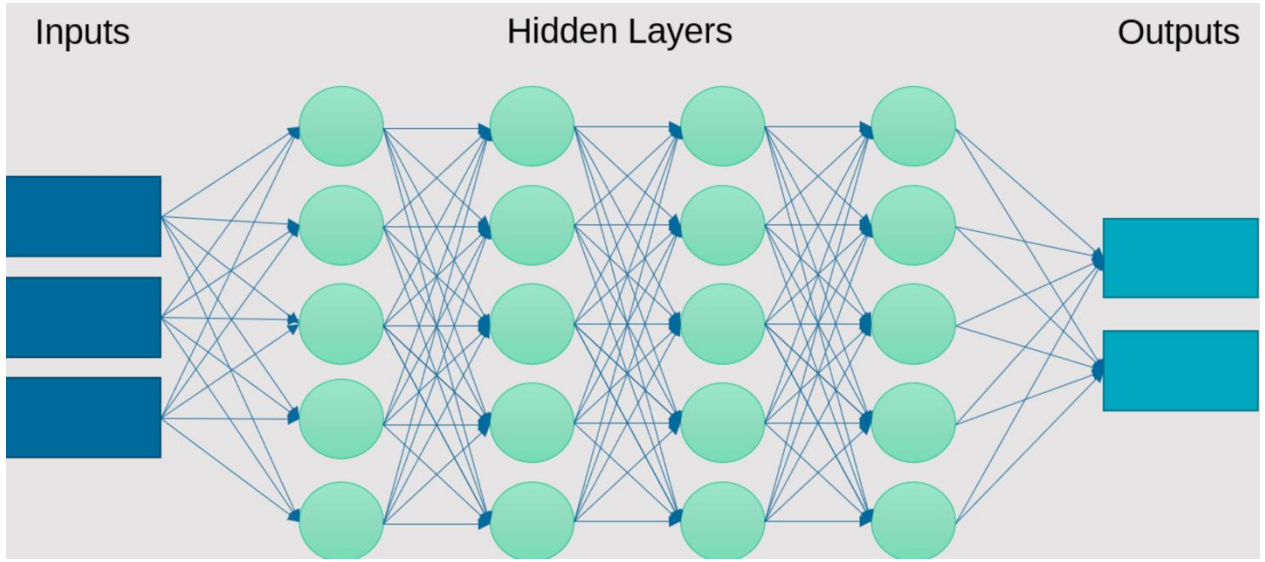
- Rather, the "depth" is in terms of successive layers of representations

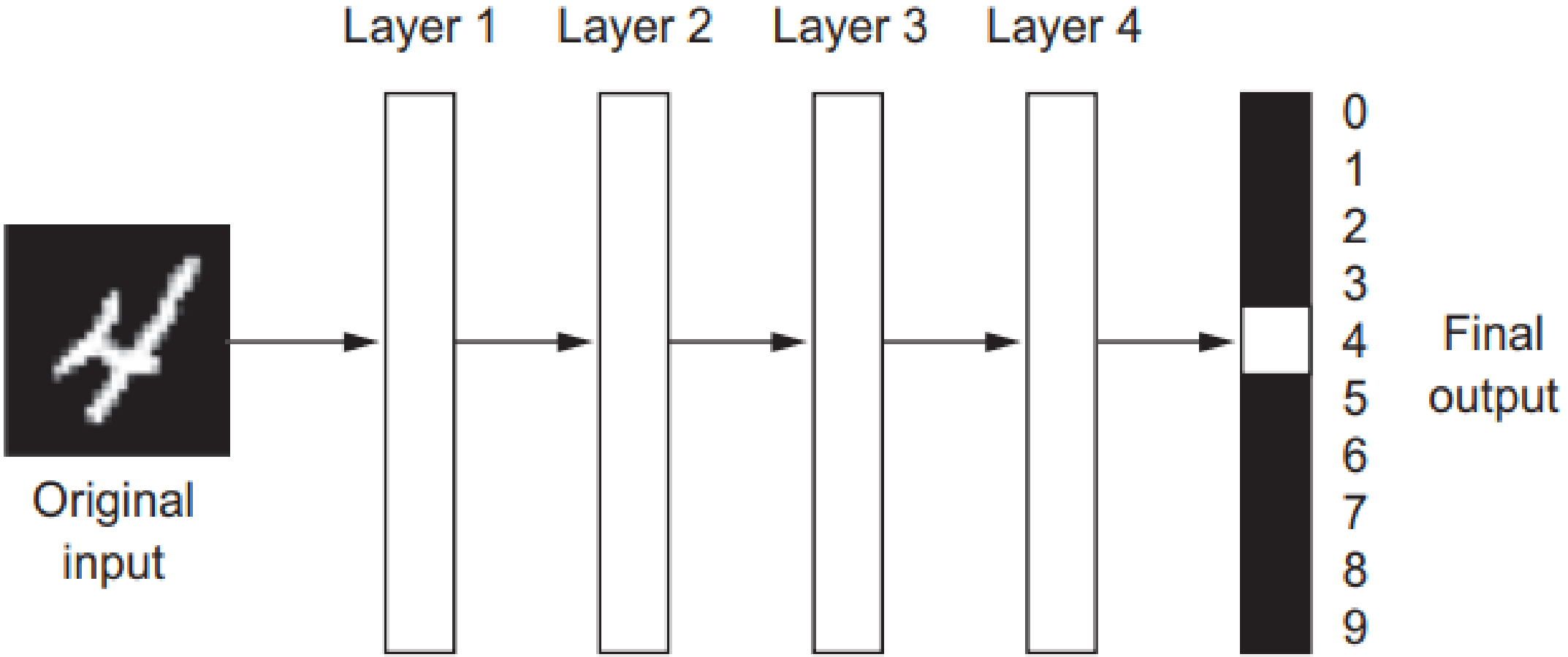
Neural Networks Again

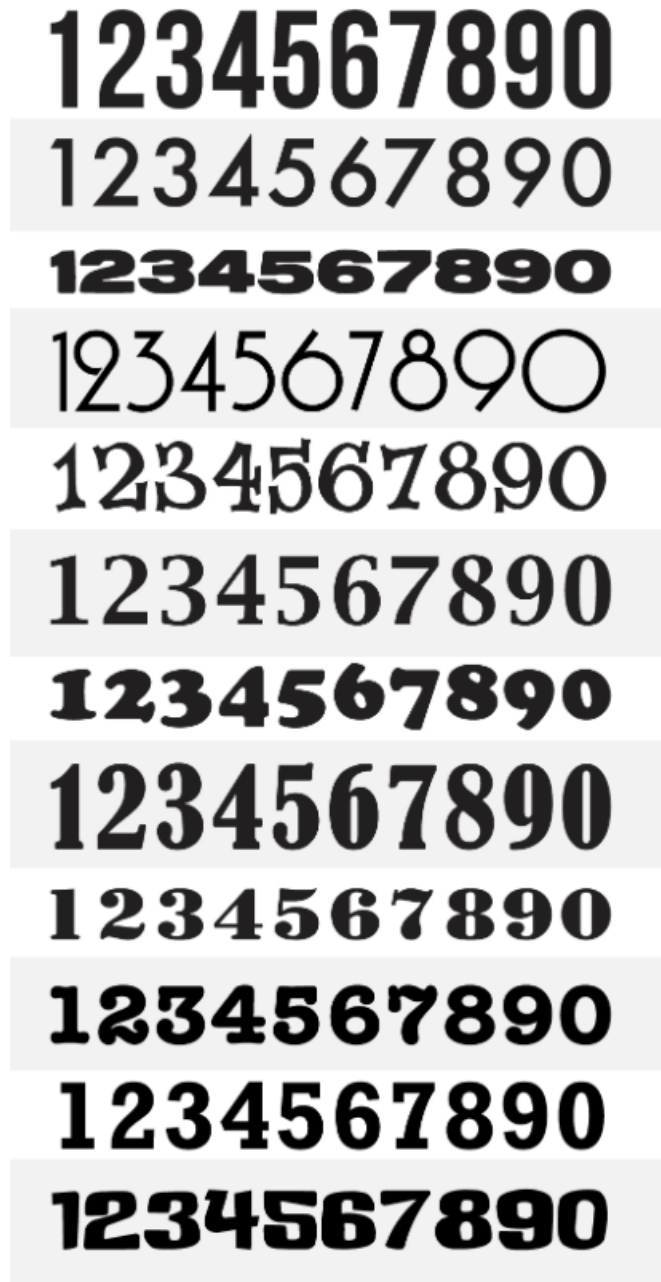
- Other ML approaches focus on learning a few layers of data representation

- Deep learning involves structured layers stacked on top of each other

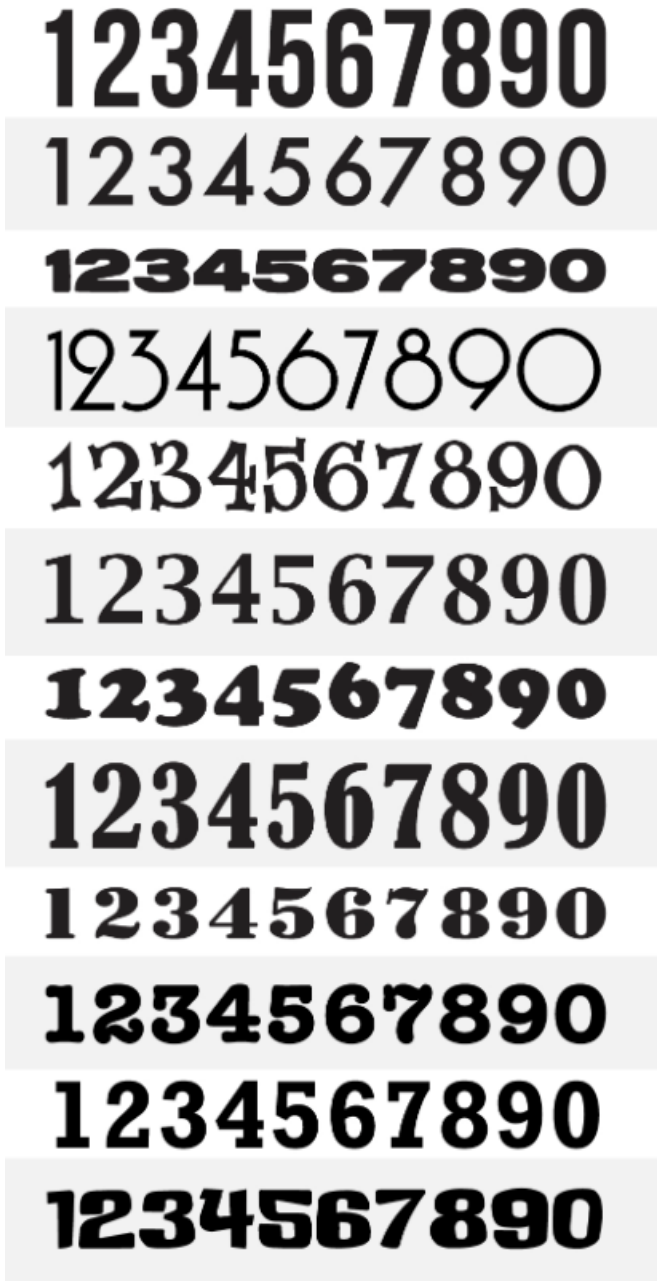





Layer 1
Layer 2
Layer 3
Layer 4
Original input
0
1
2
3
4
5
6
7
8
9
Final output


1234567890
1234567890
1234567890
1234567890
1234567890
1234567890
1234567890
1234567890
1234567890
1234567890
1234567890
1234567890

HOW IS IT THAT YOU RECOGNIZE THESE NUMBERS DESPITE DIFFERENCES IN SHAPE?

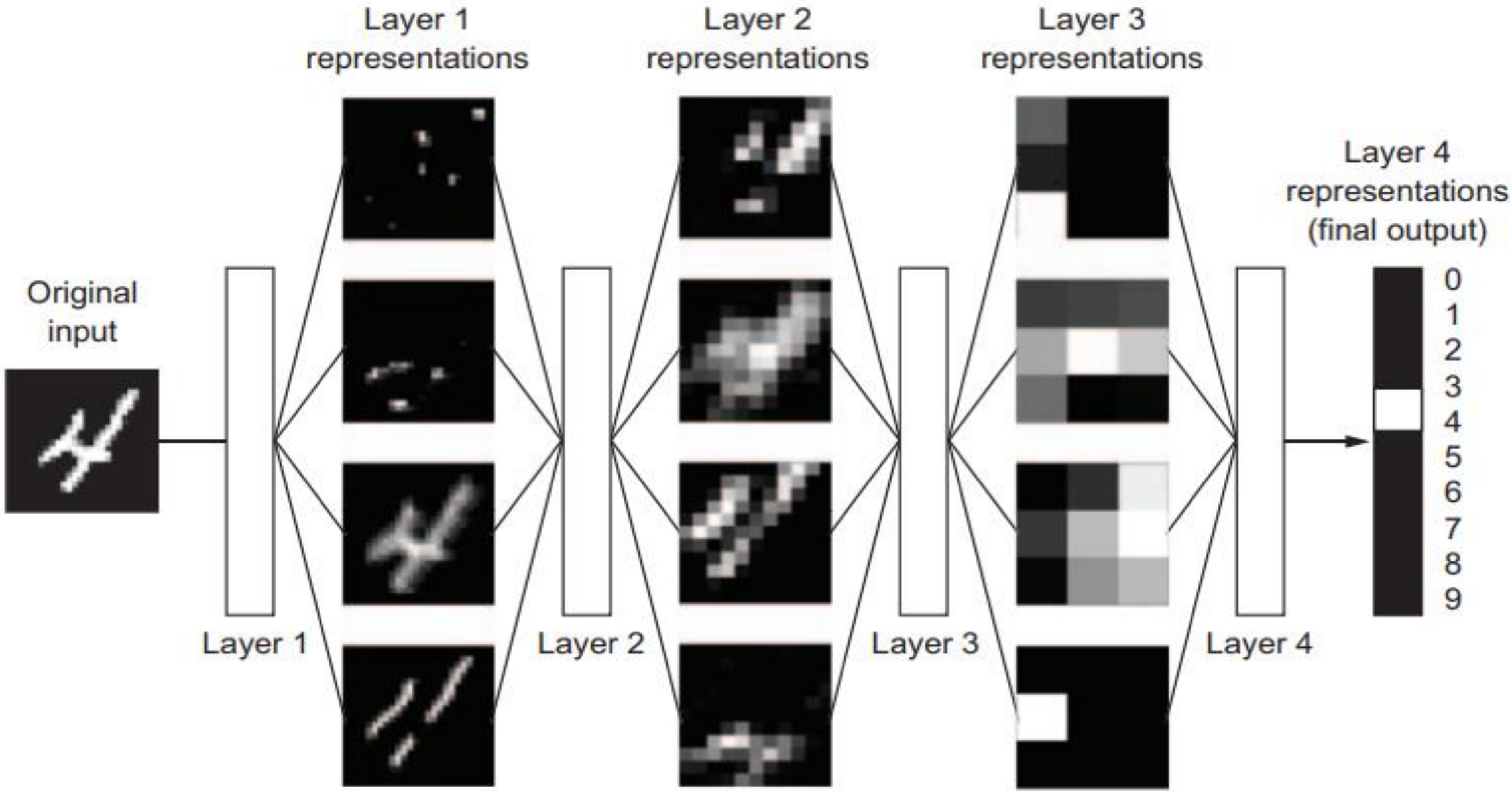


Digital Transformation

- The neural network transforms the image into representations increasingly different from the original but more informative of the output

- Deep neural networks are like information distillation operations, where information goes through filters and comes out increasingly purified

- When sufficiently scaled, the output can look like **magic**

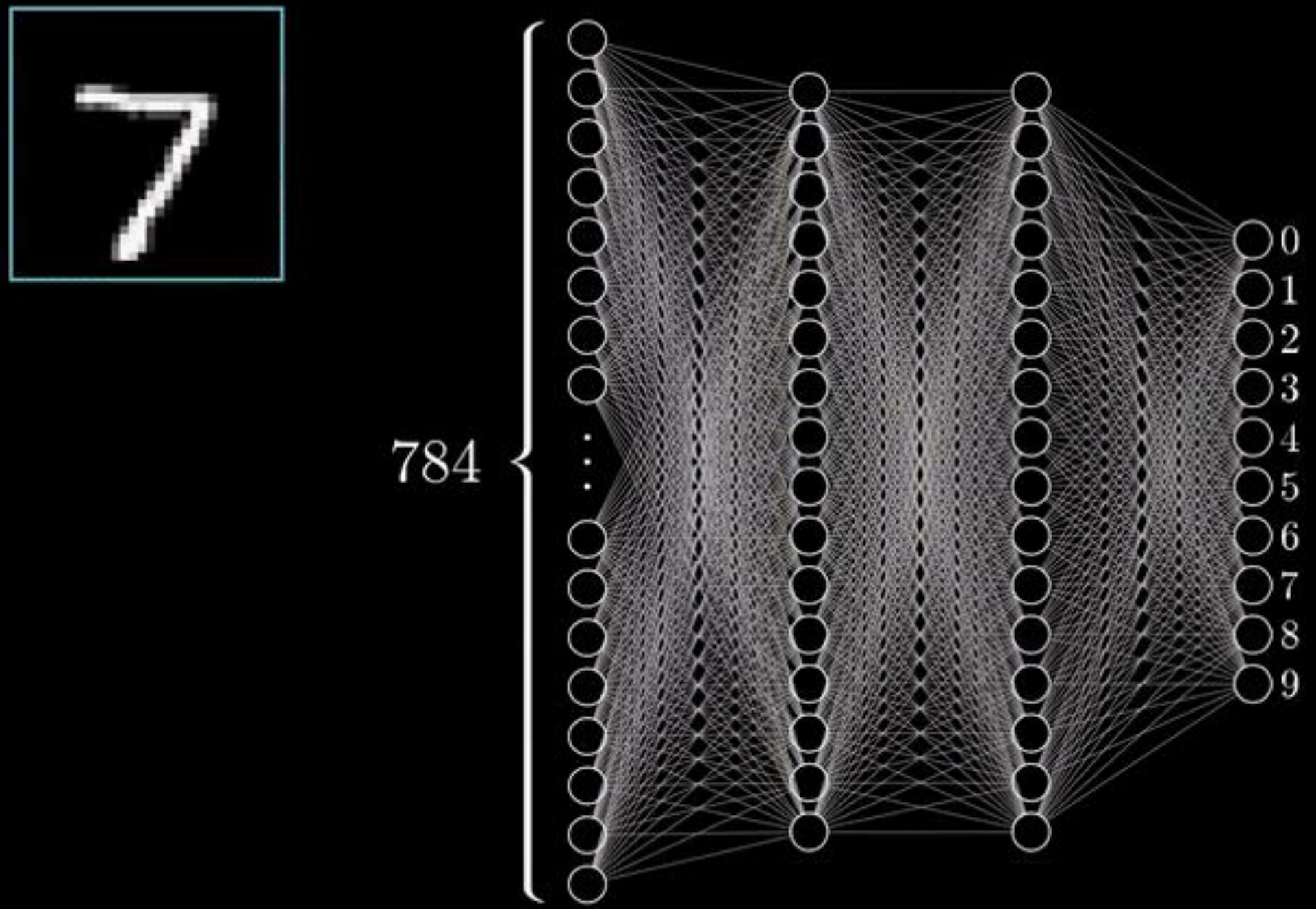

Layer 1 representations
Layer 2 representations
Layer 3 representations
Layer 4 representations (final output)
Original input
Layer 1
Layer 2
Layer 3
Layer 4
0
1
2
3
4
5
6
7
8
9

How Does Deep Learning Work?

- Machine learning involves mapping inputs (such as images) to targets (the label "cat"), having observed many examples of input and targets

- Deep neural networks do this **input-to-target mapping** via a deep sequence of data transformations (layers)

- Where **data transformations** are learned by exposure to examples


784
0
1
2
3
4
5
6
7
8
9

How Does Deep Learning Work?

- The specification of what a layer does to its input data is stored in the **layer's weights**

- In technical terms, we'd say that the transformation implemented by a layer is **parameterized** by its weights

- **Weights** are also sometimes called the **parameters of a layer**

How Does Deep Learning Work?

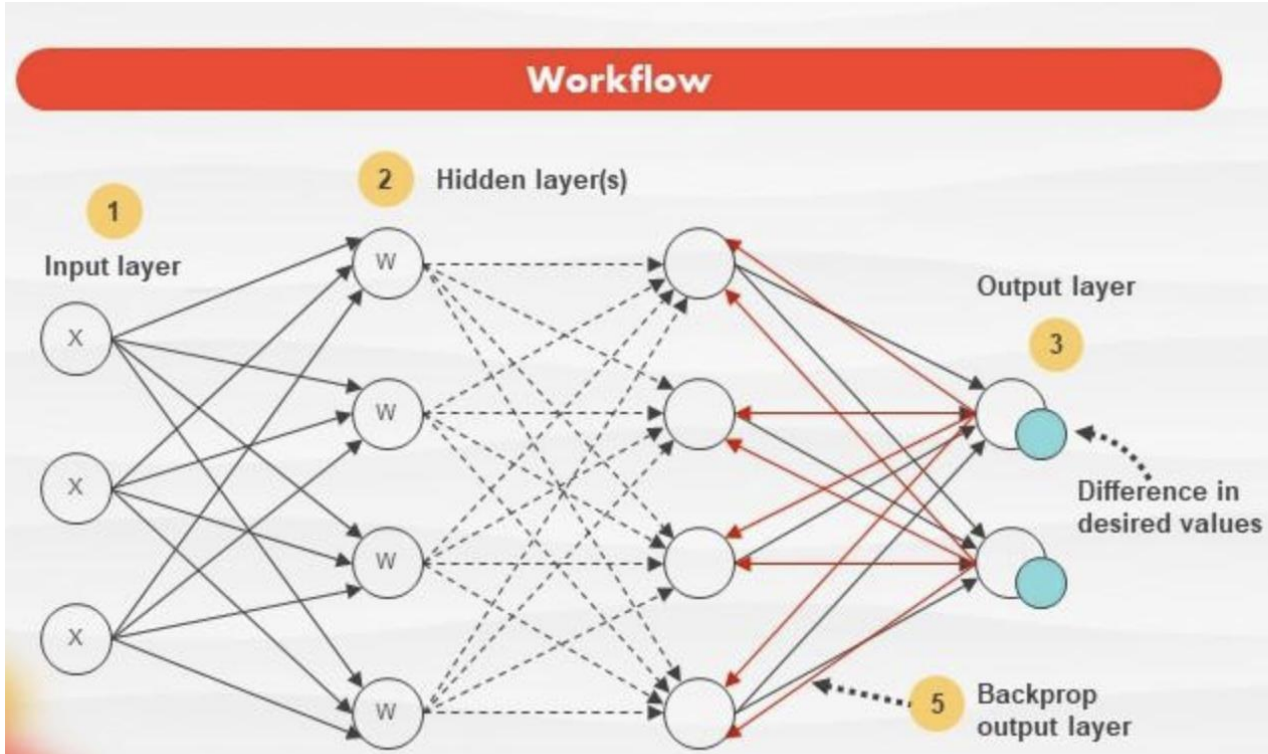
- Learning means finding weights of all layers in a network, so the network will correctly map example inputs to target outputs
- A deep neural network can contain **tens of millions** of parameters
- Modifying the value of one parameter may affect the behavior of all the others

How Does Deep Learning Work?

- The fundamental trick in deep learning is to use a feedback signal to adjust the value of the weights a little in a direction that will provide more accurate output

BACKPROPAGATION ALGORITHM

Backpropagation Algorithm



Algorithm

1. Inputs, represented by X, are fed into the neural network through a pre-connected path
2. Input is modeled using real weights, which are typically chosen randomly
3. Output for each neuron in the input layer, hidden layers, and output layer is computed
4. Error in the output is calculated using the formula
   ErrorB = Actual Output - Desired Output
5. Neural network travels back from the output layer to the hidden layer to adjust the weights to reduce the error

Analogy

- Imagine you have a group of art students trying to learn to recognize the number "7" in different styles

- Each student specializes in looking for different features - one might focus on straight lines, another on angles, another on overall shape

Analogy

- Our art students would initially guess **wildly**

- One student might say "I think it's a 7 if I see any diagonal line!" while another might say "It's only a 7 if it looks exactly like Times New Roman"

Master Guidance

- Backpropagation is like having a master artist who looks at their collective guesses and gives feedback
    - No, that wasn't a 7, that was a 1
    - Student A, you were too excited about that vertical line
    - Student B, you need to pay more attention to the overall proportions
- The students adjust their criteria based on this feedback

Backpropagation

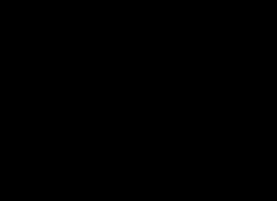
- Students adjust based on the feedback. Over time:
  - The "line specialist" learns which lines matter for 7s
  - The "angle specialist" learns the characteristic top angle
  - The "shape specialist" learns the overall proportions

- The master artist's feedback flows backward through all the students - if the final answer was wrong, each student learns how they contributed

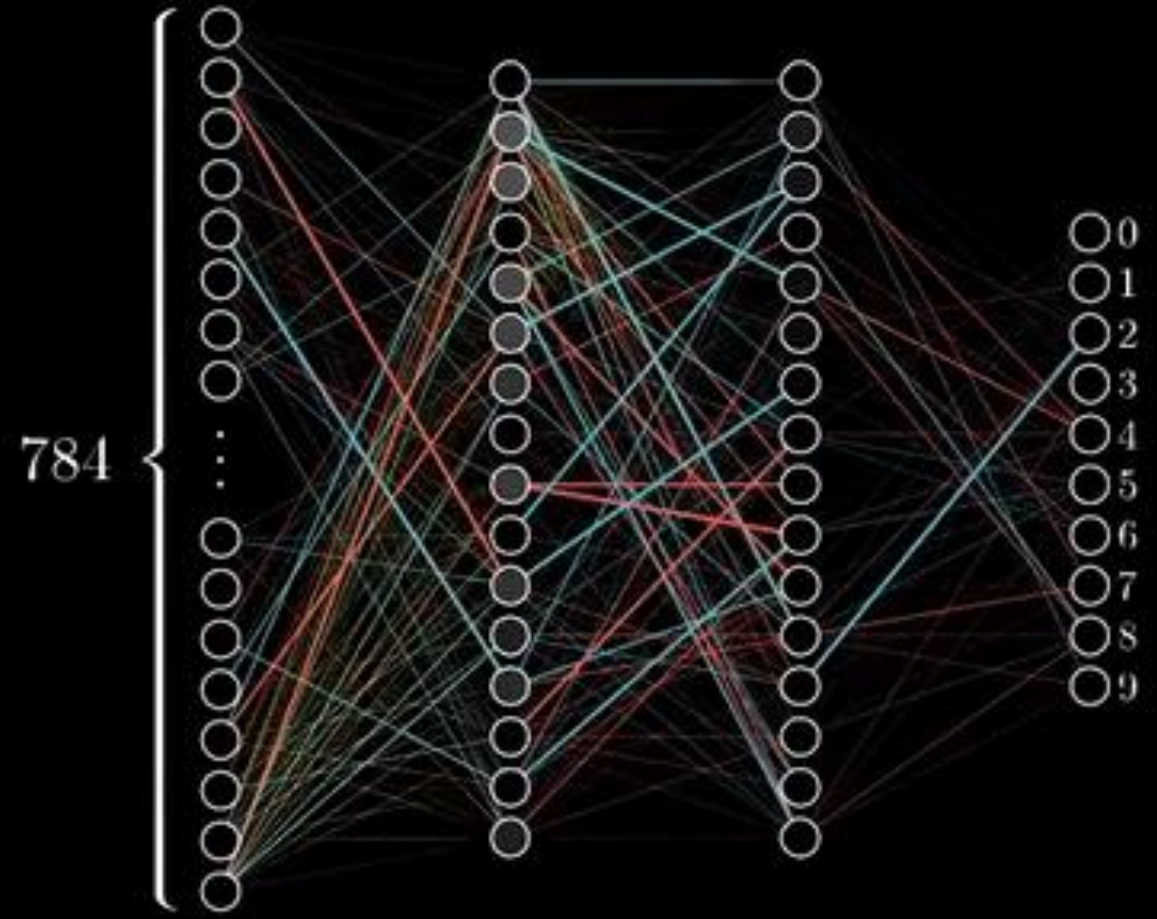
If node X encounters a strong diagonal line, AND node Y spots a horizontal top piece, AND the proportions are confirmed to be correct by node Z

THEN we're probably looking at a 7

Training in progress...



9



Outline

- Foundations of Modern AI
- Machine Learning
- Deep Learning
- MOWL

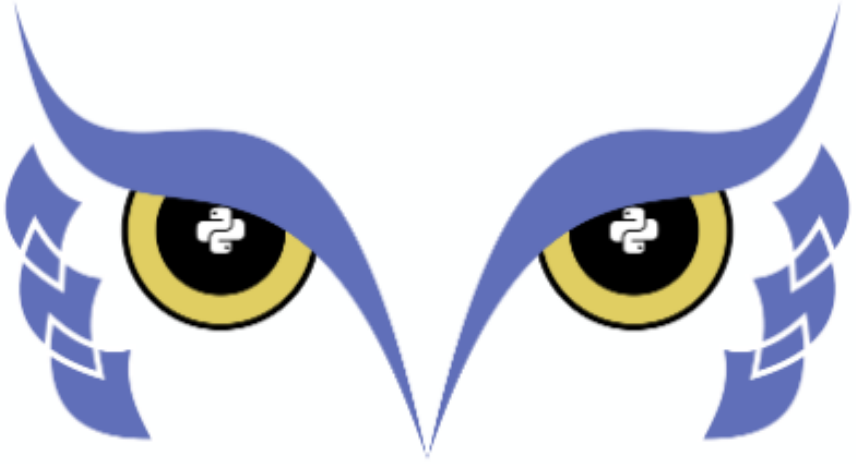
MOWL

- MOWL (Machine Learning with Ontologies) is a Python library for:
  - Converting OWL ontologies into graph data usable in deep learning
  - Training knowledge graph embedding models under logical constraints
  - Evaluating models for axiom prediction, link prediction, or ontology completion

Practically Uses…

- Use design patterns to generate embeddings that can then be used for classification, clustering, etc.
- If missing axioms, train embeddings so the model suggests missing links
- Compute vector distances between entities to find ontologically "near" items for mapping
- Predict novel instances of classes based on ontological relationships, e.g. sensor instance → predict class/units with ontology embedding

Install and Setup



mOWL

Python library for
Machine Learning with Ontologies

https://mowl.readthedocs.io/en/latest/install/index.html

Venv

- We want a reproducible, isolated space so package versions don't clash with anything else on your machine (or each other).

```
python -m venv env
source env/bin/activate      # (Mac/Linux)
env\Scripts\activate         # (Windows)
```

- After activation, your terminal prompt usually shows "env" at the beginning, i.e. you're "inside" the environment

Venv

- We want a reproducible, isolated space so package versions don't clash with anything else on your machine (or each other).

???

```
python -m venv env
source env/bin/activate      # (Mac/Linux)
env\Scripts\activate         # (Windows)
```

- After activation, your terminal prompt usually shows "env" at the beginning, i.e. you're "inside" the environment

Virtual Environment

- **venv** stands for "Virtual Environment", which is a self-contained folder that holds its own installation of Python and its own packages; the command:

```
python -m venv env
```

- Creates a private workspace named "env" that has its own copy of Python libraries and dependencies

Venv

- We want a reproducible, isolated space so package versions don't clash with anything else on your machine (or each other).

???

```
python -m venv env
source env/bin/activate      # (Mac/Linux)
env\Scripts\activate         # (Windows)
```

- If you open a new terminal later, you'll need to **activate it again**

Activate

- When you activate the environment:

```
source env/bin/activate      # Mac/Linux
env\Scripts\activate         # Windows
```

- Your terminal prompt changes to show (env) at the beginning, and now every python installation or python command refers only to that folder environment

Requirements

- MOWL builds neural models over ontologies; to do that we need:
  - torch for deep learning (PyTorch)
  - mowl for the ontology-ML bridge
  - owlready2 for reading OWL easily in Python
- Within your env, use the following command:

```
pip install torch mowl owlready2
```

Check

- Before we write any code, we should confirm the libraries were imported

- Try:

  python -c "import mowl; print('MOWL version:', mowl.__version__)”

Python

- Before we write any code, we should confirm the libraries were imported

- Try:

  python -c "import mowl; print('MOWL version:', mowl.__version__)”

Tells your computer to use the python interpreter

If you’re in env, then it will use the interpreter in that folder

Command

- Before we write any code, we should confirm the libraries were imported

- Try:

  python -c "import mowl; print('MOWL version:', mowl.__version__)”

  “command”

  Means, run the code I’m about to give you as a string

  Everything in the quotes will be executed as if it were inside a .py file

Import

- Before we write any code, we should confirm the libraries were imported

- Try:

  python -c "import mowl; print('MOWL version:', mowl.__version__)”

Loads the MOWL library into memory.

Print Version

- Before we write any code, we should confirm the libraries were imported

- Try:

  python -c "import mowl; print('MOWL version:', mowl.__version__)”

Prints a message along with MOWL’s internal version number

Uses the special “__version__” attribute from Python

Check

- Before we write any code, we should confirm the libraries were imported

- Try:

  python -c "import mowl; print('MOWL version:', mowl.__version__)”

- If **successful**, you should see a printed version number, e.g. MOWL version: “0.x.y”; if you get an **ImportError**, you’re probably not in the env or the install failed

MOWL Datasets

- MOWL is designed to handle input in OWL format

- A **MOWL dataset** contains 3 ontologies:
  - Training Ontology
  - Validation Ontology
  - Testing Ontology

- Each plays a role in how the model **learns**, **tunes**, and **evaluates** its understanding of ontological structure

MOWL Datasets

- MOWL is designed to handle input in OWL format

- A **MOWL dataset** contains 3 ontologies:
  - **Training Ontology**
  - Validation Ontology
  - Testing Ontology

- Each plays a role in how the model **learns**, **tunes**, and **evaluates** its understanding of ontological structure

Training Ontology

- The ontology MOWL uses to train the embedding model; contains axioms that define the semantic structure you want the model to learn
  - Dog $\sqsubseteq$ Mammal
  - Mammal $\sqsubseteq$ Animal
- What MOWL:
  - Converts all logical axioms into a set of triples (subject–predicate–object).
  - Uses these triples to train neural embeddings, so related entities (like Dog and Mammal) become geometrically close
- This is the information the model sees and learns from

Toy Training Ontology

@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .

@prefix owl: <http://www.w3.org/2002/07/owl#> .

@prefix ex: <http://example.org#> .

ex:Animal a owl:Class .

ex:Mammal a owl:Class ; rdfs:subClassOf ex:Animal .

ex:Dog a owl:Class ; rdfs:subClassOf ex:Mammal .

MOWL will turn such logical statements into training constraints

MOWL Datasets

- MOWL is designed to handle input in OWL format

- A **MOWL dataset** contains 3 ontologies:
  - Training Ontology
  - **Validation Ontology**
  - Testing Ontology

- Each plays a role in how the model **learns**, **tunes**, and **evaluates** its understanding of ontological structure

Validation Ontology

- Ontology used to check whether model is learning effectively; might contain a subset of axioms from training, e.g. subclass or property assertions not used in the training ontology

- What MOWL does:
  - After each training session, MOWL tests the model's predictions against the ontology
  - Validation score guides adjustments, e.g. learning rate, embedding size, etc.
  - Helps determine when to stop training, i.e. validation stop improving

- Used for fine-tuning the model, not for training it directly

Toy Validation Ontology

@prefix ex: <http://example.org#> .

@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .

@prefix owl: <http://www.w3.org/2002/07/owl#> .

ex:Cat a owl:Class ; rdfs:subClassOf ex:Mammal .

Axiom held back to check learning

MOWL Datasets

- MOWL is designed to handle input in OWL format

- A **MOWL dataset** contains 3 ontologies:
  - Training Ontology
  - Validation Ontology
  - **Testing Ontology**

- Each plays a role in how the model **learns**, **tunes**, and **evaluates** its understanding of ontological structure

Testing Ontology

- The testing ontology is used after training to measure how well the model generalizes to unseen data:

  If the model learned Dog $\sqsubseteq$ Mammal and Mammal $\sqsubseteq$ Animal, then the test ontology might include Dog $\sqsubseteq$ Animal as a **logically implied axiom**

- What MOWL does:
  - Evaluates how well the trained embeddings predict withheld axioms
  - Produces metrics, e.g. how highly the correct entity is ranked; how often the correct relation is among the top predictions

- Test data the model has never seen, used to report final performance

Toy Testing Ontology

@prefix ex: <http://example.org#> .

@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .

@prefix owl: <http://www.w3.org/2002/07/owl#> .

ex:Dog a owl:Class ; rdfs:subClassOf ex:Animal .

This logical relationship should be inferred, not seen during training

```
import os, mowl, torch
from mowl.datasets import PathDataset
from mowl.models import ELEmbeddings

mowl.init_jvm("2g")                                  # Start the JVM once per process

ds = PathDataset("toy_train.ttl")                    # Load the training ontology

os.makedirs("checkpoints", exist_ok=True)            # Configure checkpoint (avoids temp-file errors)

model = ELEmbeddings(ds, embed_dim=16, epochs=6)
model.save_path = "checkpoints"
model.model_filepath = "checkpoints/elembeddings_best.pt"

model.train(epochs=6)                                # Training loop

emb = model.get_embeddings()                         #Get embeddings
cos = torch.nn.functional.cosine_similarity          # Compute similarities

dog    = "http://example.org#Dog"
mammal = "http://example.org#Mammal"
animal = "http://example.org#Animal"

print("sim(Dog, Mammal):", float(cos(emb[dog],  emb[mammal], dim=0)))
print("sim(Mammal, Animal):", float(cos(emb[mammal], emb[animal], dim=0)))
print("sim(Dog, Animal):", float(cos(emb[dog],    emb[animal], dim=0)))

print("\n[TEST] Held-out axiom: Dog ⊑ Animal")
print("Plausibility (cosine):", float(cos(emb[dog], emb[animal], dim=0)))
```

Reads toy_train.ttl and parses axioms into internal triples for learning.

```
import os, mowl, torch
from mowl.datasets import PathDataset
from mowl.models import ELEmbeddings

mowl.init_jvm("2g")                                  # Start the JVM once per process

ds = PathDataset("toy_train.ttl")                    # Load the training ontology

os.makedirs("checkpoints", exist_ok=True)            # Configure checkpoint (avoids temp-file errors)

model = ELEmbeddings(ds, embed_dim=16, epochs=6)
model.save_path = "checkpoints"
model.model_filepath = "checkpoints/elembeddings_best.pt"

model.train(epochs=6)                                # Training loop

emb = model.get_embeddings()                         #Get embeddings
cos = torch.nn.functional.cosine_similarity          # Compute similarities

dog    = "http://example.org#Dog"
mammal = "http://example.org#Mammal"
animal = "http://example.org#Animal"

print("sim(Dog, Mammal):", float(cos(emb[dog],  emb[mammal], dim=0)))
print("sim(Mammal, Animal):", float(cos(emb[mammal], emb[animal], dim=0)))
print("sim(Dog, Animal):", float(cos(emb[dog],    emb[animal], dim=0)))

print("\n[TEST] Held-out axiom: Dog ⊑ Animal")
print("Plausibility (cosine):", float(cos(emb[dog], emb[animal], dim=0)))
```

```
import os, mowl, torch
from mowl.datasets import PathDataset
from mowl.models import ELEmbeddings

mowl.init_jvm("2g")                                  # Start the JVM once per process

ds = PathDataset("toy_train.ttl")                    # Load the training ontology

os.makedirs("checkpoints", exist_ok=True)            # Configure checkpoint (avoids temp-file errors)

model = ELEmbeddings(ds, embed_dim=16, epochs=6)
model.save_path = "checkpoints"
model.model_filepath = "checkpoints/elembeddings_best.pt"

model.train(epochs=6)                                # Training loop

emb = model.get_embeddings()                         #Get embeddings
cos = torch.nn.functional.cosine_similarity          # Compute similarities

dog    = "http://example.org#Dog"
mammal = "http://example.org#Mammal"
animal = "http://example.org#Animal"

print("sim(Dog, Mammal):", float(cos(emb[dog],  emb[mammal], dim=0)))
print("sim(Mammal, Animal):", float(cos(emb[mammal], emb[animal], dim=0)))
print("sim(Dog, Animal):", float(cos(emb[dog],    emb[animal], dim=0)))

print("\n[TEST] Held-out axiom: Dog ⊑ Animal")
print("Plausibility (cosine):", float(cos(emb[dog], emb[animal], dim=0)))
```

```
import os, mowl, torch
from mowl.datasets import PathDataset
from mowl.models import ELEmbeddings

mowl.init_jvm("2g")                                  # Start the JVM once per process

ds = PathDataset("toy_train.ttl")                    # Load the training ontology

os.makedirs("checkpoints", exist_ok=True)            # Configure checkpoint (avoids temp-file errors)

model = ELEmbeddings(ds, embed_dim=16, epochs=6)
model.save_path = "checkpoints"
model.model_filepath = "checkpoints/elembeddings_best.pt"

model.train(epochs=6)                                # Training loop

emb = model.get_embeddings()                         #Get embeddings
cos = torch.nn.functional.cosine_similarity          # Compute similarities

dog    = "http://example.org#Dog"
mammal = "http://example.org#Mammal"
animal = "http://example.org#Animal"

print("sim(Dog, Mammal):", float(cos(emb[dog],  emb[mammal], dim=0)))
print("sim(Mammal, Animal):", float(cos(emb[mammal], emb[animal], dim=0)))
print("sim(Dog, Animal):", float(cos(emb[dog],    emb[animal], dim=0)))

print("\n[TEST] Held-out axiom: Dog ⊑ Animal")
print("Plausibility (cosine):", float(cos(emb[dog], emb[animal], dim=0)))
```

Embeddings

- Recall, an **embedding** is a way of representing symbolic things (like words, images, or classes in an ontology) as vectors of numbers, e.g. points in a geometric space

- For example:
  - The word "king" might be represented as [0.12, -0.3, 0.8, ...]
  - The class ex:Dog might be represented as [0.5, 0.7, -0.2, ...]

- Each of these numbers corresponds to a dimension in a geometric space

- If the embedding size is 16, then each class is represented as a 16-D vector

```
import os, mowl, torch
from mowl.datasets import PathDataset
from mowl.models import ELEmbeddings

mowl.init_jvm("2g")                                  # Start the JVM once per process

ds = PathDataset("toy_train.ttl")                    # Load the training ontology

os.makedirs("checkpoints", exist_ok=True)            # Configure checkpoint (avoids temp-file errors)

model = ELEmbeddings(ds, embed_dim=16, epochs=6)
model.save_path = "checkpoints"
model.model_filepath = "checkpoints/elembeddings_best.pt"

model.train(epochs=6)                                # Training loop

emb = model.get_embeddings()                         #Get embeddings
cos = torch.nn.functional.cosine_similarity          # Compute similarities

dog    = "http://example.org#Dog"
mammal = "http://example.org#Mammal"
animal = "http://example.org#Animal"

print("sim(Dog, Mammal):", float(cos(emb[dog],  emb[mammal], dim=0)))
print("sim(Mammal, Animal):", float(cos(emb[mammal], emb[animal], dim=0)))
print("sim(Dog, Animal):", float(cos(emb[dog],    emb[animal], dim=0)))

print("\n[TEST] Held-out axiom: Dog ⊑ Animal")
print("Plausibility (cosine):", float(cos(emb[dog], emb[animal], dim=0)))
```

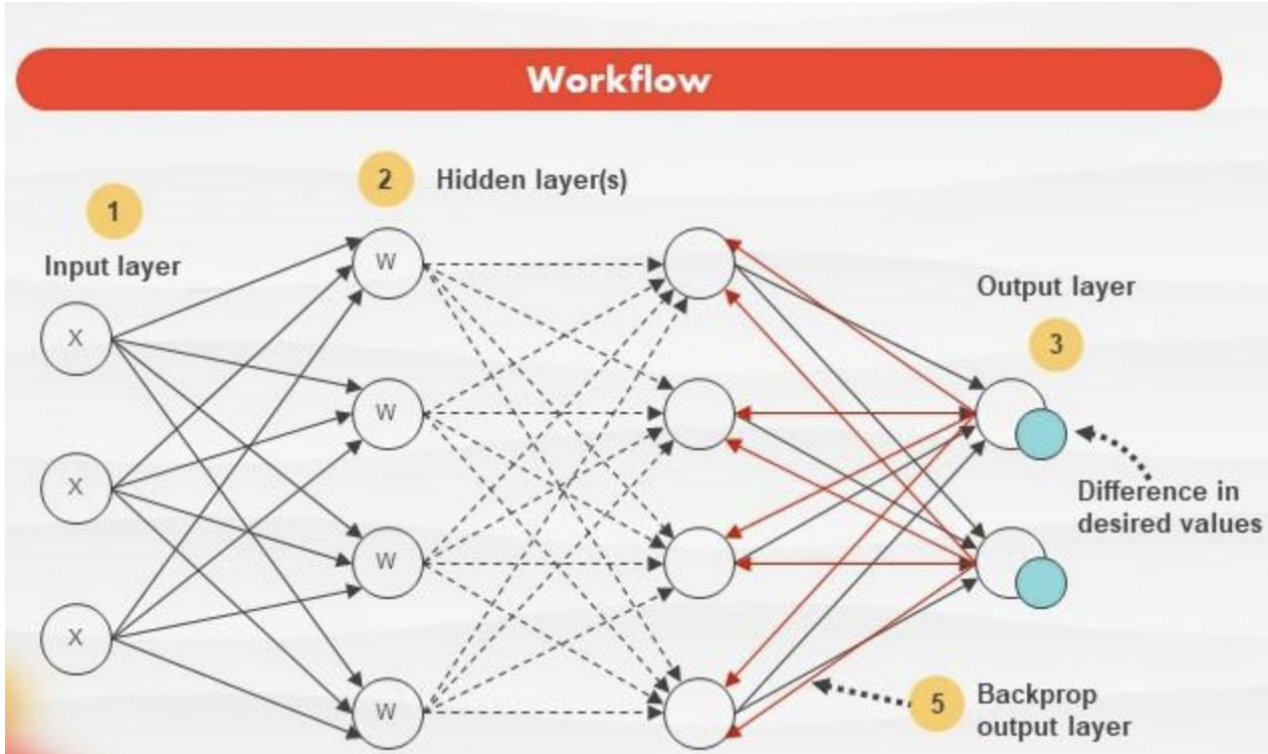
Training Loop

- The training loop is where MOWL **learns the embeddings** by iteratively adjusting them ; “learner.train(epochs=6)” runs through this process 6 times

- What MOWL does:
  - Every class (e.g., Dog, Mammal, Animal) starts with a random vector
  - Each axiom becomes a **training pair**, a constraint the model must satisfy
  - A scoring function estimates how well each pair fits, e.g. how close Dog is to Mammal
  - Model updates vectors to minimize loss, e.g. make Dog closer to Mammal
  - With each pass, vectors settle into positions that best represent the logical structure

- Learned embeddings encode semantic structure that mirrors ontology logic

Backpropagation Algorithm



Algorithm

1. Inputs, represented by X, are fed into the neural network through a pre-connected path
2. Input is modeled using real weights, which are typically chosen randomly
3. Output for each neuron in the input layer, hidden layers, and output layer is computed
4. Error in the output is calculated using the formula
   ErrorB = Actual Output - Desired Output
5. Neural network travels back from the output layer to the hidden layer to adjust the weights to reduce the error

```
import os, mowl, torch
from mowl.datasets import PathDataset
from mowl.models import ELEmbeddings

mowl.init_jvm("2g")                                  # Start the JVM once per process

ds = PathDataset("toy_train.ttl")                    # Load the training ontology

os.makedirs("checkpoints", exist_ok=True)            # Configure checkpoint (avoids temp-file errors)

model = ELEmbeddings(ds, embed_dim=16, epochs=6)
model.save_path = "checkpoints"
model.model_filepath = "checkpoints/elembeddings_best.pt"

model.train(epochs=6)                                # Training loop

emb = model.get_embeddings()                         #Get embeddings
cos = torch.nn.functional.cosine_similarity          # Compute similarities

dog    = "http://example.org#Dog"
mammal = "http://example.org#Mammal"
animal = "http://example.org#Animal"

print("sim(Dog, Mammal):", float(cos(emb[dog],  emb[mammal], dim=0)))
print("sim(Mammal, Animal):", float(cos(emb[mammal], emb[animal], dim=0)))
print("sim(Dog, Animal):", float(cos(emb[dog],    emb[animal], dim=0)))

print("\n[TEST] Held-out axiom: Dog ⊑ Animal")
print("Plausibility (cosine):", float(cos(emb[dog], emb[animal], dim=0)))
```

Returns a mapping {IRI: torch.Tensor} e.g., emb["http://example.org#Dog"] → a 16-dim vector.

```
import os, mowl, torch
from mowl.datasets import PathDataset
from mowl.models import ELEmbeddings

mowl.init_jvm("2g")                                  # Start the JVM once per process

ds = PathDataset("toy_train.ttl")                    # Load the training ontology

os.makedirs("checkpoints", exist_ok=True)            # Configure checkpoint (avoids temp-file errors)

model = ELEmbeddings(ds, embed_dim=16, epochs=6)
model.save_path = "checkpoints"
model.model_filepath = "checkpoints/elembeddings_best.pt"

model.train(epochs=6)                                # Training loop

emb = model.get_embeddings()                         #Get embeddings
cos = torch.nn.functional.cosine_similarity          # Compute similarities

dog    = "http://example.org#Dog"
mammal = "http://example.org#Mammal"
animal = "http://example.org#Animal"

print("sim(Dog, Mammal):", float(cos(emb[dog],  emb[mammal], dim=0)))
print("sim(Mammal, Animal):", float(cos(emb[mammal], emb[animal], dim=0)))
print("sim(Dog, Animal):", float(cos(emb[dog],    emb[animal], dim=0)))

print("\n[TEST] Held-out axiom: Dog ⊑ Animal")
print("Plausibility (cosine):", float(cos(emb[dog], emb[animal], dim=0)))
```

After training, expect:
sim(Dog, Mammal) high
sim(Mammal, Animal) high,
sim(Dog, Animal) also positive

Reflect Structure Geometrically

- "Every Dog is a kind of Mammal" is expressed by placing the vector for Dog close to or inside the vector for Mammal in space

- "cos(emb[dog], emb[mammal])" checks how **geometrically close** the two classes are

Logical Relation	Geometric Analogy
Subclass ($A \sqsubseteq B$)	The vector/region of A is contained within or near B
Disjointness	Vectors are far apart or non-overlapping
Equivalence ($A \equiv B$)	Vectors are nearly identical
Relation ($A \rightarrow B$)	The vector of A + some offset ≈ vector of B

Exercise

Setup your MOWL environment

Navigate to the course GitHub page, download the training ontology, and the test script

Ensure you can run the script and can return a test score as described

Training datasets:
    gci0: 2
    gci1: 0
    gci2: 0
    gci3: 0
    gci0_bot: 0
    gci1_bot: 0
    gci3_bot: 0

Epoch 1: Train loss: 1.21
Epoch 2: Train loss: 1.20
Epoch 3: Train loss: 1.19
Epoch 4: Train loss: 1.18
Epoch 5: Train loss: 1.17
Epoch 6: Train loss: 1.16

Similarity scores:
Dog–Mammal: 0.92
Mammal–Animal: 0.89
Dog–Animal: 0.83

✅ Done! Model trained and embeddings computed.

Training datasets:

gci0: 2
gci1: 0
gci2: 0
gci3: 0
gci0_bot: 0
gci1_bot: 0
gci3_bot: 0

These numbers summarize how many logical axioms MOWL extracted from your ontology

Epoch 1: Train loss: 1.21
Epoch 2: Train loss: 1.20
Epoch 3: Train loss: 1.19
Epoch 4: Train loss: 1.18
Epoch 5: Train loss: 1.17
Epoch 6: Train loss: 1.16

Similarity scores:
Dog–Mammal: 0.92
Mammal–Animal: 0.89
Dog–Animal: 0.83

✅ Done! Model trained and embeddings computed.

Code	Logical Pattern	Meaning
gci0	$(A \sqsubseteq B)$	Simple subclass inclusion
gci1	$(A \sqsubseteq \exists R.B)$	Class with existential restriction on the right
gci2	$(\exists R.A \sqsubseteq B)$	Restriction on the left
gci3	$(A \sqcap B \sqsubseteq C)$	Intersection inclusion
gci*_bot	$\bot$	Inconsistent

Training datasets:

gci0: 2
gci1: 0
gci2: 0
gci3: 0
gci0_bot: 0
gci1_bot: 0
gci3_bot: 0

Epoch 1: Train loss: 1.21
Epoch 2: Train loss: 1.20
Epoch 3: Train loss: 1.19
Epoch 4: Train loss: 1.18
Epoch 5: Train loss: 1.17
Epoch 6: Train loss: 1.16

Similarity scores:
Dog–Mammal: 0.92
Mammal–Animal: 0.89
Dog–Animal: 0.83

✅ Done! Model trained and embeddings computed.

An epoch means one full pass through the training data, e.g. the 2 subclass axioms
Each epoch computes a loss, which measures how far off the model's current embeddings are from what the logic requires

The ELEmbeddings model represents each class as a 16x16 vector, e.g. each vector is adjusted so subclasses sit inside their superclasses in that space

Training datasets:

gci0: 2
gci1: 0
gci2: 0
gci3: 0
gci0_bot: 0
gci1_bot: 0
gci3_bot: 0

Epoch 1: Train loss: 1.21
Epoch 2: Train loss: 1.20
Epoch 3: Train loss: 1.19
Epoch 4: Train loss: 1.18
Epoch 5: Train loss: 1.17
Epoch 6: Train loss: 1.16

Similarity scores:
Dog–Mammal: 0.92
Mammal–Animal: 0.89
Dog–Animal: 0.83

Each number is the cosine similarity between two vectors in the embedding space

1.0 = identical vectors
0.0 = completely unrelated
–1.0 = opposite direction

✅ Done! Model trained and embeddings computed.

Initial Run

- Our first exercise here used only the **training ontology**
- This was to emphasize how learning works
- During the next session, we'll add **validation** and **test ontologies** to measure how well the model generalizes
- Though you'll get a head start on that in **project 5**...

Summary

- create env → install → load ontology → train → inspect embeddings
- MOWL turns OWL axioms into a trainable representation whose geometry reflects semantic structure
- Everyone should now be able to run MOWL locally and is primed to plug it into your **semantic pipelines**