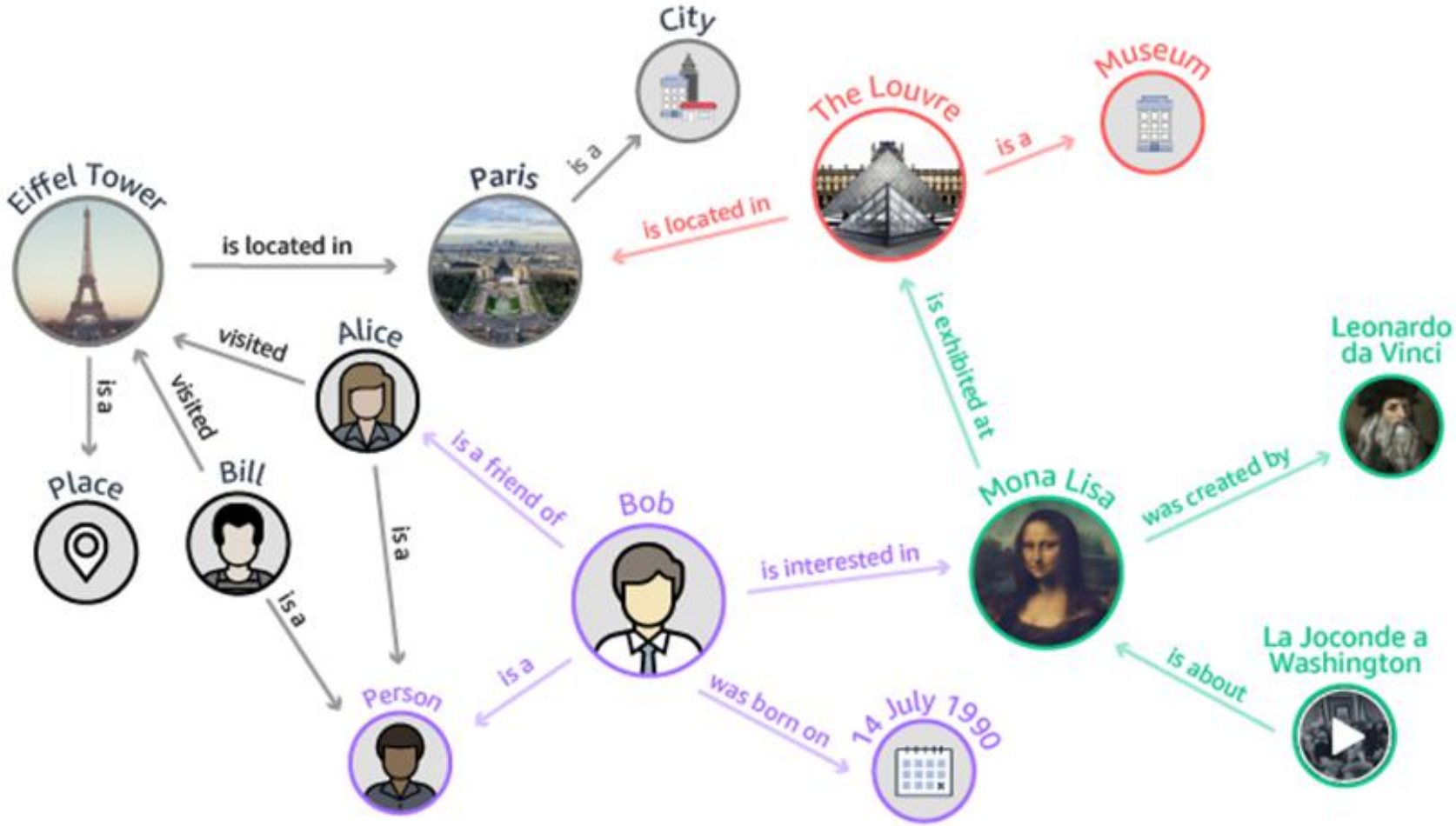
# Artificial Intelligence

## CS4365 --- Fall 2022

## Knowledge Representation and Reasoning

Instructor: Yunhui Guo

# Knowledge and Reasoning



# Knowledge and Reasoning

- Knowledge:
  - the fact or condition of knowing something with familiarity gained through experience or association

- Reasoning:
  - the drawing of inferences or conclusions through the use of reason

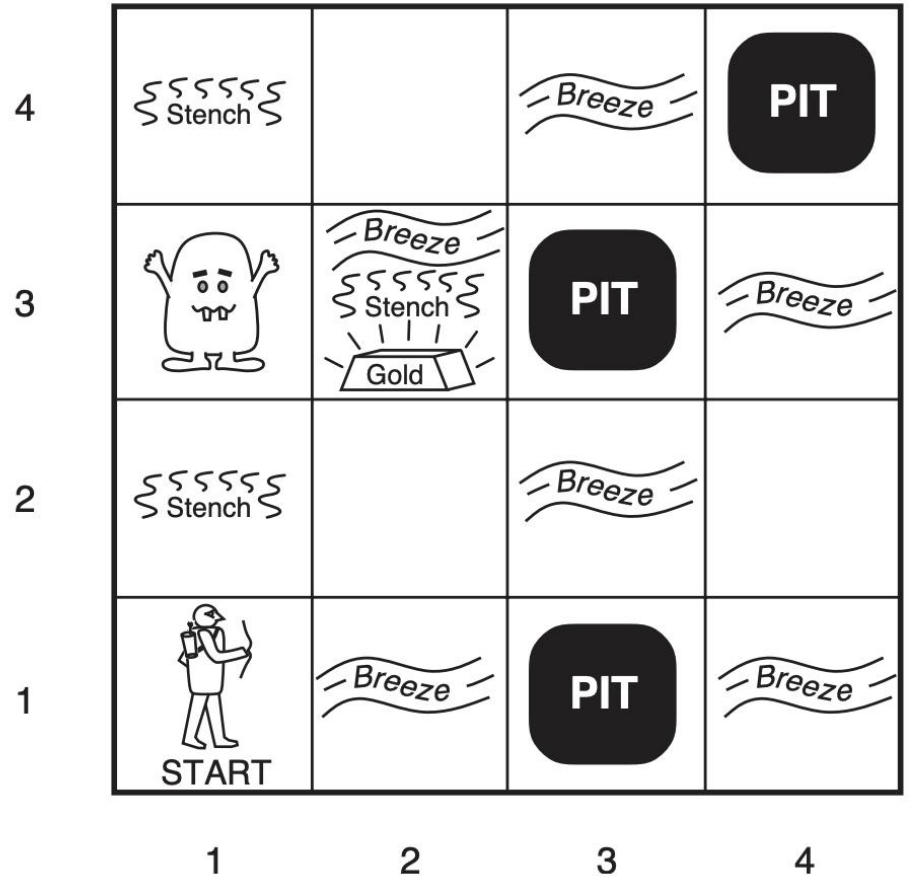
Knowledge + Reasoning → New Knowledge

# The Wumpus World

- A decision-maker needs to represent knowledge of the world and reason with it in order to safely explore this world.

E.g. In the squares directly adjacent to a pit, the agent will perceive a Breeze

→ There a pit in [2, 2] or [3, 1] or both



# Knowledge Representation

- Human intelligence relies on a lot of background knowledge (the more you know, the easier many tasks become / “knowledge is power”)

  E.g. SEND + MORE = MONEY puzzle.

- Natural language understanding
  - Time flies like an arrow.
  - Fruit flies like bananas.

  - The spirit is willing but the flesh is weak. (English)
  - The vodka is good but the meat is rotten. (Russian)
- Or: Plan a trip to L.A.

# Knowledge Representation

Q. How did we encode (domain) knowledge so far?

For search problems?

Fine for limited amounts of knowledge / well-defined domains.

Otherwise: knowledge-based systems approach

# Knowledge-Based Systems / Agents

Key components

- knowledge base: a set of sentences expressed in some knowledge representation language
- Inference / reasoning mechanisms to query what is known and to derive new information or make decisions

# Knowledge-Based Systems / Agents

- Natural candidate: logical language (propositional / first-order) combined with a logical inference mechanism
- How close to human thought?
- In any case, appears reasonable strategy for machines

# Logic

- Logic:
  - defines a formal language for logical reasoning
- It gives us a tool that helps us to understand how to construct a valid argument
- Logic defines:
  - the meaning of statements
  - the rules of logical inference

# Logic as a Knowledge Representation

Three components:

- syntax: specifies which sentences can be constructed in a given formal logic
  - E.g. $x + y = 4$
- semantics: specifies what a sentence means
  - $x + y = 4$ is True if $x = 2$ and $y = 2$
- proof theory: a set of general purpose rules that allow efficient derivation of new information from the sentences in the knowledge base

# Logic as a Knowledge Representation

Model:  a truth assignment to every propositonal symbol

Logic entailment:

A sentence follows logically from another sentence:

$$\alpha \models \beta$$

In every model in which α is true, β is also true.

# Logic as a Knowledge Representation

- Logic inference:
  - Given a knowledge base KB and a sentence α
  - Does a KB  semantically entail α?  KB $\models$ α

One possible approach:

Model Checking:  enumerate all the possible models to check if α is true in all models in which KB is true

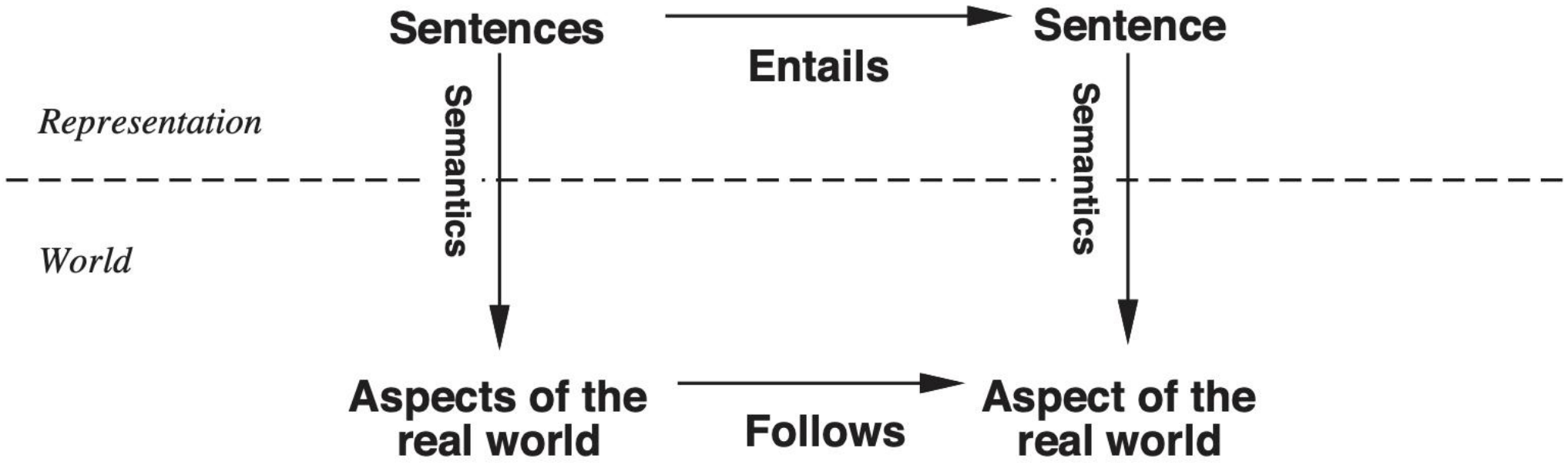
# Logic as a Knowledge Representation

Proof theory:

Sound: An inference algorithm that derives only entailed sentences

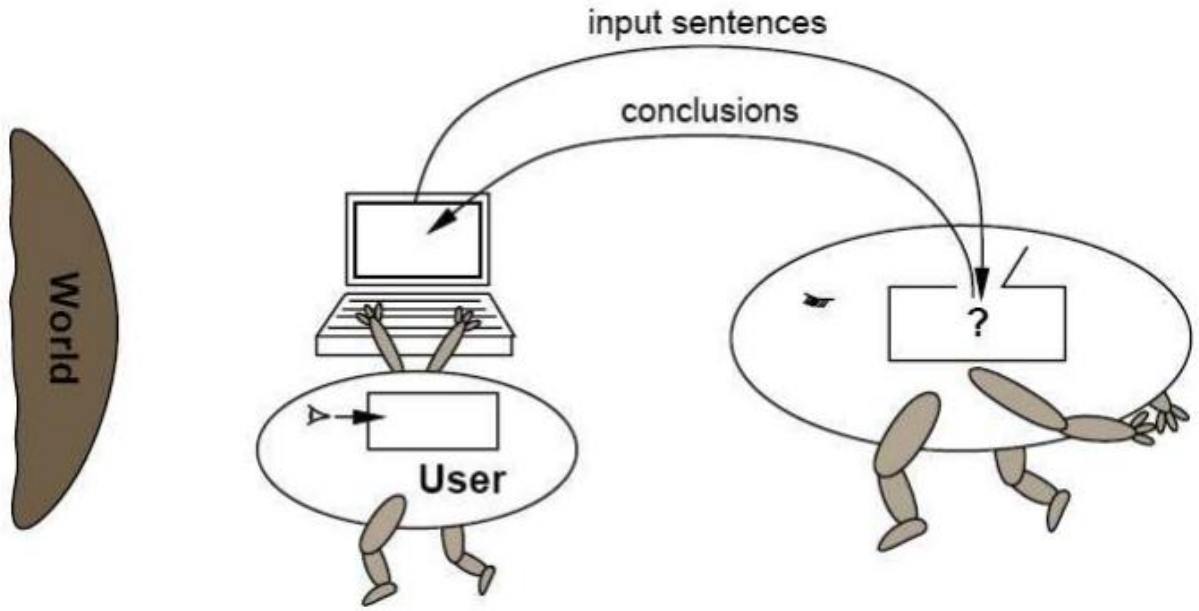
Complete: an inference algorithm is complete if it can derive any sentence that is entailed

# Connecting Sentences to the Real World




- Logical reasoning should ensure that the new configurations represent aspects of the world that actually follow from the aspects that the old configurations represent.

# Tenuous Link to Real World




- All computer has are sentences (hopefully about the world).

# KR Language: Propositional Logic

- Go back to 3rd century B.C. studied by Stoic school of philosophy
- Real development began in the mid-19th century and was initiated by the English mathematician G. Boole
- The classical propositional calculus was first formulated as a formal axiomatic system by the eminent German logician G. Frege in 1879.

# KR Language: Propositional Logic

- The simplest logic
- Definition
  - A proposition is a statement that is either true or false.

- Example:
  - 5 + 2 = 8 (F)
  - It is raining today
    - (either T or F)

# KR Language: Propositional Logic

- Literal: an atomic formula or its negation
  - Positive literal: P, Q
  - Negative literal: ¬P, ¬Q

- Syntax: build sentences from atomic propositions, using connectives:
  - $\wedge$ : and
  - $\vee$ : or
  - ¬ : not
  - $\Rightarrow$ : implies
  - $\Leftrightarrow$ : equivalence (biconditional)

# KR Language: Propositional Logic

Syntax: build sentences from atomic propositions, using connectives $\wedge$ , $\vee$ , $\neg$ , $\Rightarrow$, $\Leftrightarrow$

(and / or / not / implies / equivalence (biconditional))

E.g.: $\neg P$

$Q \wedge R$

$( \neg P \vee (Q \wedge R)) \Rightarrow S$

# KR Language: Propositional Logic

- Clause: a disjunction of literals

  E.g.: $Q \vee R$

- Conjunctive normal form (CNF): a conjunction of clauses

  E.g.: $(Q \vee R) \wedge (P \vee R)$

- Every formula can be equivalently written as a formula in conjunctive normal form

  $(Q \wedge R) \vee P \quad \rightarrow \quad (Q \vee P) \wedge (R \vee P)$

# Semantics

Semantics specifies what something means.

In propositional logic, the semantics (i.e., meaning) of a sentence is the set of interpretations (i.e., truth assignments) in which the sentence evaluates to True.

Example:

The semantics of the sentence $P \vee Q \Rightarrow R$ is

- P is True, Q is True, R is True
- P is True , Q is False, R is True
- P is False , Q is True , R is True
- P is False , Q is False , R is True
- P is False , Q is False , R is False

# Interpretations: The Key to Semantics

An interpretation is a logician's word for "truth assignment"

- Given 3 propositional symbols P, Q, R, there are 8 interpretations.
- Given n propositional symbols $P_1$, $P_2$,... $P_n$, there are $2^n$ interpretations

In propositional logic:

- an interpretation is a mapping from propositional symbols to truth values.
- the meaning of a sentence is the set of interpretations in which the sentence evaluates to True

How to evaluate a sentence under a given interpretation?

# Evaluating a sentence under interpretation I

We can evaluate a sentence using a truth table

| $P$ | $Q$ |
|---|---|
| $false$ | $false$ |
| $false$ | $true$ |
| $true$ | $false$ |
| $true$ | $true$ |

# Evaluating a sentence under interpretation I

We can evaluate a sentence using a truth table

| $P$ | $Q$ | $\neg P$ |
|---|---|---|
| $false$ | $false$ | $true$ |
| $false$ | $true$ | $true$ |
| $true$ | $false$ | $false$ |
| $true$ | $true$ | $false$ |

# Evaluating a sentence under interpretation I

We can evaluate a sentence using a truth table

| $P$ | $Q$ | $\neg P$ | $P \wedge Q$ |
|---|---|---|---|
| $false$ | $false$ | $true$ | $false$ |
| $false$ | $true$ | $true$ | $false$ |
| $true$ | $false$ | $false$ | $false$ |
| $true$ | $true$ | $false$ | $true$ |

# Evaluating a sentence under interpretation I

We can evaluate a sentence using a truth table

| $P$ | $Q$ | $\neg P$ | $P \wedge Q$ | $P \vee Q$ |
|---|---|---|---|---|
| $false$ | $false$ | $true$ | $false$ | $false$ |
| $false$ | $true$ | $true$ | $false$ | $true$ |
| $true$ | $false$ | $false$ | $false$ | $true$ |
| $true$ | $true$ | $false$ | $true$ | $true$ |

# Evaluating a sentence under interpretation I

We can evaluate a sentence using a truth table

| $P$ | $Q$ | $\neg P$ | $P \wedge Q$ | $P \vee Q$ | $P \Rightarrow Q$ |
|---|---|---|---|---|---|
| $false$ | $false$ | $true$ | $false$ | $false$ | $true$ |
| $false$ | $true$ | $true$ | $false$ | $true$ | $true$ |
| $true$ | $false$ | $false$ | $false$ | $true$ | $false$ |
| $true$ | $true$ | $false$ | $true$ | $true$ | $true$ |

# Evaluating a sentence under interpretation I

We can evaluate a sentence using a truth table

| $P$ | $Q$ | $\neg P$ | $P \wedge Q$ | $P \vee Q$ | $P \Rightarrow Q$ | $P \Leftrightarrow Q$ |
|---|---|---|---|---|---|---|
| $false$ | $false$ | $true$ | $false$ | $false$ | $true$ | $true$ |
| $false$ | $true$ | $true$ | $false$ | $true$ | $true$ | $false$ |
| $true$ | $false$ | $false$ | $false$ | $true$ | $false$ | $false$ |
| $true$ | $true$ | $false$ | $true$ | $true$ | $true$ | $true$ |

# Evaluating a sentence under interpretation I

We can evaluate a sentence using a truth table

| $P$ | $Q$ | $\neg P$ | $P \wedge Q$ | $P \vee Q$ | $P \Rightarrow Q$ | $P \Leftrightarrow Q$ |
|---|---|---|---|---|---|---|
| *false* | *false* | *true* | *false* | *false* | *true* | *true* |
| *false* | *true* | *true* | *false* | *true* | *true* | *false* |
| *true* | *false* | *false* | *false* | *true* | *false* | *false* |
| *true* | *true* | *false* | *true* | *true* | *true* | *true* |

Note: $\Rightarrow$ is somewhat counterintuitive

What's the true value of “5 is even implies Sam is smart”

If P is True, then I claim Q is True

# Three Important Concepts

- Logic Equivalence
- Validity
- Satisfiability

# Logic Equivalence

- Two sentences are equivalent if they are true in the same set of models.

- We write this as $\alpha \equiv \beta$. $\alpha \equiv \beta$ if and only if $\alpha \models \beta$ and $\beta \models \alpha$

For example:

I. If Lisa is in Denmark, then she is in Europe

II. If Lisa is not in Europe, then she is not in Denmark

# Logic Equivalence

- $(\alpha \wedge \beta) \equiv (\beta \wedge \alpha)$ commutativity of $\wedge$
- $(\alpha \vee \beta) \equiv (\beta \vee \alpha)$ commutativity of $\vee$
- $((\alpha \wedge \beta) \wedge \gamma) \equiv (\alpha \wedge (\beta \wedge \gamma))$ associativity of $\wedge$
- $((\alpha \vee \beta) \vee \gamma) \equiv (\alpha \vee (\beta \vee \gamma))$ associativity of $\vee$
- $\neg(\neg \alpha) = \alpha$ double-negation
- $(\alpha \Rightarrow \beta) \equiv (\neg\beta \Rightarrow \neg\alpha)$ contraposition
- $(\alpha \Rightarrow \beta) \equiv (\neg\alpha \vee \beta)$ implication elimination
- $(\alpha \Leftrightarrow \beta) \equiv ((\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha))$ biconditional elimination

# Logic Equivalence

- $\neg(\alpha \wedge \beta) \equiv (\neg\alpha \vee \neg\beta)$ De Morgan
- $\neg(\alpha \vee \beta) \equiv (\neg\alpha \wedge \neg\beta)$ De Morgan
- $(\alpha \wedge (\beta \vee \gamma)) \equiv ((\alpha \wedge \beta) \vee (\alpha \wedge \gamma))$ distributivity of $\wedge$ over $\vee$
- $(\alpha \vee (\beta \wedge \gamma)) \equiv ((\alpha \vee \beta) \wedge (\alpha \vee \gamma))$ distributivity of $\vee$ over $\wedge$

These equivalences play much the same role in logic as arithmetic identities do in ordinary mathematics.

# Validity

- Some sentences are very true! For example

1) True 2) $P \Rightarrow P$ 3) $(P \wedge Q) \Rightarrow Q$

A valid sentence is one whose meaning includes every possible interpretation.

$$((P \vee H) \wedge (\neg H)) \Rightarrow P$$

| $P$ | $H$ | $P \vee H$ | $(P \vee H) \wedge \neg H$ | $((P \vee H) \wedge \neg H) \Rightarrow P$ |
|---|---|---|---|---|
| *False* | *False* | *False* | *False* | *True* |
| *False* | *True* | *True* | *False* | *True* |
| *True* | *False* | *True* | *True* | *True* |
| *True* | *True* | *True* | *False* | *True* |

The truth table shows that $((P \vee H) \wedge (\neg H)) \Rightarrow P$ is valid

We write $\models ((P \vee H) \wedge (\neg H)) \Rightarrow P$

# Satisfiability

- An unsatisfiable sentence is one whose meaning has no interpretation (e.g., $P \wedge \neg P$ )
- A satisfiable sentence is one whose meaning has at least one interpretation.
- A sentence must be either satisfiable or unsatisfiable but it can't be both.
- If a sentence is valid then it's satisfiable.
- If a sentence is satisfiable then it may or may not be valid.

# Satisfiability

- The SAT problem is to determine the satisfiability of sentences

- Conection to validity:
  - α is valid iff ¬α is unsatisfiable
  - α is satisfiable iff ¬α is not valid

- Proving by checking the unsatisfiability:

  $\alpha \models \beta$ if and only if the sentence $(\alpha \wedge \neg\beta)$ is unsatisfiable

# Knowledge Base and Models

- Knowledge base: a set of sentences. Each sentence represents some assertation about the world.
- A model of a set of sentences (KB) is a truth assignment in which each of the KB sentences evaluates to True.
- With more and more sentences, the models of KB start looking more and more like the “real-world”.

# Models

If a sentence α holds (is True) in all models of a KB, we say that α is entailed by the KB.

α is of interest, because whenever KB is true in a world α will also be True.

We write $KB \models \alpha$

# Entailment Examples

KB

R1: CS4365Lectures ⇒ (TodayIsTuesday ∨ TodayIsThursday)

R2: ¬TodayIsThursday

R3: TodayIsSaturday ⇒ SleepLate

R4: Rainy ⇒ GrassIsWet

R5: CS4365Lectures ∨ TodayIsSaturday

R6: ¬ SleepLate

# Entailment Examples

KB

R1: CS4365Lectures ⇒ (TodayIsTuesday ∨ TodayIsThursday)

R2: ¬TodayIsThursday

R3: TodayIsSaturday ⇒ SleepLate

R4: Rainy ⇒ GrassIsWet

R5: CS4365Lectures ∨ TodayIsSaturday

R6: ¬ SleepLate

Then which of these are correct entailments?

$KB \models \neg \text{SleepLate}$

$KB \models \neg \text{SleepLate} \vee \text{GrassIsWet}$

$KB \models \text{GrassIsWet}$

$KB \models \text{TodayIsTuesday}$

# Entailment Examples

- KB
  - Propositional symbols:
    - CS4365Lectures, TodayIsTuesday, TodayIsThursday, TodayIsSaturday, SleepLate, Rainy, GrassIsWet
  - Model checking:
    - Enumerate all the possible models to check if α is true in all models is in all models in which KB is true

# Entailment Examples

KB

R1: CS4365Lectures ⇒ (TodayIsThursday ∨ TodayIsThusday)

R2: ¬TodayIsTuesday

R3: TodayIsSaturday ⇒ SleepLate

R4: Rainy ⇒ GrassIsWet

R5: CS4365Lectures ∨ TodayIsSaturday

R6: ¬ SleepLate

CS4365Lectures: T TodayIsThursday: T TodayIsTuesday: F

TodayIsSaturday: F SleepLate: F Rainy: F/T GrassIsWet: T/F

# Entailment Examples

- KB is True when
  - CS4365Lectures: T
  - TodayIsThursday: T
  - TodayIsTuesday: F
  - TodayIsSaturday: F
  - SleepLate: F
  - Rainy: F/T
  - GrassIsWet: T/F

$KB \models \neg \text{SleepLate}$ T

$KB \models \neg \text{SleepLate} \vee \text{GrassIsWet}$ T

$KB \models \text{GrassIsWet}$ F

$KB \models \text{TodayIsTuesday}$ F

- Complexity: $O(2^N)$

# Logical Inference

- Problem definition:
  - The computer has a knowledge base KB.
  - The user inputs a sentence.
  - The computer tells the user whether the sentence is entailed by the knowledge base.

Humans who are doing proofs almost never use this brute-force approach. Then how to do logical inference efficiently?

# Proof Theory

- A set of purely syntactic rules for efficiently determining entailment

- We write: KB $\vdash \alpha$, i.e., $\alpha$ can be deduced from KB or $\alpha$ is provable from KB.

Key property:

Both in propositional and in first-order logic we have a proof theory ("calculus") such that:

$\models$ and $\vdash$ are equivalent

# Proof Theory (cont.)

If KB $\vdash \alpha$ imples KB $\models \alpha$, we say the proof theory is sound

If KB $\models \alpha$ implies KB $\vdash \alpha$ , we say the proof theory is complete.

Why so important?

Allow computer to ignore semantics and “just push symbols”!

# Example Proof Theory

One rule of inference: Modus Ponens

From $\alpha$ and $\alpha \Rightarrow \beta$ it follows that $\beta$.

Semantic soundness can easily be verified (using truth table).

Another rule of inference: **And-Elimination**

From $\alpha \wedge \beta$, it follows that $\alpha$ and $\beta$.

# Example Proof Theory

Axiom schemas:

(Ax. I) $\alpha \Rightarrow (\beta \Rightarrow \alpha)$

(Ax. II) $((\alpha \Rightarrow (\beta \Rightarrow \gamma)) \Rightarrow ((\alpha \Rightarrow \beta) \Rightarrow ((\alpha \Rightarrow \gamma)))$

(Ax. III) $(\neg\alpha \Rightarrow \beta) \Rightarrow ((\neg\alpha \Rightarrow \neg\beta) \Rightarrow \alpha)$

Note: $\alpha$, $\beta$, $\gamma$ stand for arbitrary sentences. So, we have an infinite collection of axioms.

# Example Proof

- Now, α can be **deduced** from a set of sentences φ iff there exists a sequence of applications of modus ponens that leads from φ to α (possibly using axioms).
- One can prove that:
  - Modus ponens with the above axioms will generate exactly all (and only those) statements logically entailed by φ.

So, we have a way of generating entailed statements in a purely syntactic manner!

(Sequence is called a proof. Finding it can be hard …)

# Example Proof

Lemma. 1) For any $\alpha$, we have $\vdash (\alpha \Rightarrow \alpha)$.

Proof.

$(\alpha \Rightarrow ((\alpha \Rightarrow \alpha) \Rightarrow \alpha)) \Rightarrow ((\alpha \Rightarrow (\alpha \Rightarrow \alpha)) \Rightarrow (\alpha \Rightarrow \alpha)), \text{Ax. II}$

$\alpha \Rightarrow ((\alpha \Rightarrow \alpha) \Rightarrow \alpha), \text{Ax. I}$

$(\alpha \Rightarrow (\alpha \Rightarrow \alpha)) \Rightarrow (\alpha \Rightarrow \alpha); \text{Modus Ponens}$

$\alpha \Rightarrow (\alpha \Rightarrow \alpha), \text{Ax. I}$

$\alpha \Rightarrow \alpha, \text{Modus Ponens}$

# Another Example Proof

Lemma. 2) For any $\alpha$ and $\beta$, we have $\beta, \neg\beta \vdash \alpha$

Proof.

$(\neg\alpha \Rightarrow \beta) \Rightarrow ((\neg\alpha \Rightarrow \neg\beta) \Rightarrow \alpha)$, (Ax. III)

$\beta$, (hyp.)

$\beta \Rightarrow (\neg\alpha \Rightarrow \beta)$, (Ax. I)

$\neg\alpha \Rightarrow \beta$, (Modus Ponens)

$(\neg\alpha \Rightarrow \neg\beta) \Rightarrow \alpha$, (Modus Ponens)

$\neg\beta$, (hyp.)

$\neg\beta \Rightarrow (\neg\alpha \Rightarrow \neg\beta)$, (Ax. I)

$\neg\alpha \Rightarrow \neg\beta$, (Modus Ponens)

$\alpha$, (Modus Ponens)

# Another Example Proof

Why are lemma 1 and lemma 2 true semantically?

I.e., $\models \alpha \Rightarrow \alpha$ and $\beta, \neg\beta \models \alpha$

- Logic entailment: $\varphi \models \gamma$
  - In every model in which $\varphi$ is **true**, $\gamma$ is also **true**.
  - No model $\omega$ in which $\varphi$ is **true** and $\gamma$ is **false**.

- False entails anything (Principle of explosion)

# Another Example Proof

Why are lemma 1 and lemma 2 true semantically?

I.e., $\models \alpha \Rightarrow \alpha$ and $\beta, \neg\beta \models \alpha$

Note: proofs are purely syntactic --- machines does not need to know anything about the meaning of the sentences!

Whatever is **syntactically** derived will be **semantically** true, and we can derive everything syntactically that is semantically true.

How hard is it to find proofs?

# Sound rules of inference

| Rule | Premise | Conclusion |
|---|---|---|
| Modus Ponens | $A, A \rightarrow B$ | $B$ |
| And Introduction | $A, B$ | $A \wedge B$ |
| And Elimination | $A \wedge B$ | $A$ |
| Double Negation | $\neg\neg A$ | $A$ |

# Monotonicity

- The set of entailed sentences can only increase as information is added to the knowledge base.

- For any sentence $\alpha$ and $\beta$

  if $KB \models \alpha$ then $KB \wedge \beta \models \alpha$

- Propositional logic is monotonic

# Key Properties

We have the following properties (also for first-order logic):

For a sound and complete proof theory, the following three conditions are equivalent:

(I) $\varphi \models \alpha$

(II) $\varphi \vdash \alpha$

(III) $\varphi, \neg\alpha$ is inconsistent (i.e., can be refuted)

(I) is semantic; (II) syntactic; (III) at high-level semantic but we have a nice syntactic automatic procedure: resolution.

What common proof technique does III represent?

# Resolution

First need canonical form: CNF

Conjunction of disjunctions / CNF (conjunctive normal form)

Example: $\neg(P \Rightarrow Q) \vee (R \Rightarrow P)$

$\neg(\neg P \vee Q) \vee (\neg R \vee P)$

$(P \wedge \neg Q) \vee (\neg R \vee P)(\text{de Morgan's law})$

$(P \vee \neg R \vee P) \wedge (\neg Q \vee \neg R \vee P)(\text{assoc. and distri. laws})$

$(P \vee \neg R) \wedge (\neg Q \vee \neg R \vee P)$

$\{(P \vee \neg R), (\neg Q \vee \neg R \vee P)\}$

$\{(P, \neg R), (\neg Q, \neg R, P)\}(\text{just notation})$

# Resolution

Given a CNF, a single inference rule (and no axioms) will allow us to determine inconsistency.

So, using property III (above) and resolution, we have a sound and complete proof procedure for propositional logic (which can be extended to first-order logic).

# The Resolution Rule (clausal form)

From $\alpha \vee p$ and $\neg p \vee \beta$ , we can derive:

$\alpha \vee \beta$ ($\alpha$ and $\beta$ are disjunctions of literals, where a literal is a propositional variable or its negation).

- Reason:
  - If p is true, $\beta$ must be true
  - If p is false, $\alpha$ must be true
  - So $\beta$ or $\alpha$ holds

# The Resolution Rule (clausal form)

From $\alpha \vee p$ and $\neg p \vee \beta$ , we can derive:

$\alpha \vee \beta$ (resolvent)

($\alpha$ and $\beta$ are disjunctions of literals, where a literal is a propositional variable or its negation).

Note: $\neg\alpha \Rightarrow p$ and $p \Rightarrow \beta$ gives $\neg\alpha \Rightarrow \beta$

It is a “chaining rule”.

# General Resolution Rule

If $(L_1 \vee L_2 \vee ...L_k)$ is True,

and $(\neg L_k \vee L_{k+1} \vee ...L_m)$ True,

then we can conclude that

$(L_1 \vee L_2 \vee ...L_{k-1} \vee L_{k+1} \vee ... \vee L_m)$ is True.

A resolution-based theorem prover can, for any sentences α and β in propositional logic, decide whether α ⊨ β

# General Resolution Rule

- We can derive the empty clause via resolution iff the set of clauses is inconsistent. ($\neg p \wedge p$)
- Method relies on property III. It's refutation complete. Note that method does not generate theorems from scratch

E.g., we have $p \wedge r \models p \vee r$ , but we can't get $p \vee r$ from {{p}, {r}}. (Why not??)

But, given {{p}, {r}} and the negation of $p \vee r$, we get the set {{p}, {r}, {¬p}, {¬r}}. Resolving on this set gives the empty clause. Thus contradiction. Thus proof.

# Algorithm: Resolution Proof

- Negate the theorem to be proved, and add the result to the list of sentences in the KB. We show that (KB $\wedge$ $\neg\alpha$) is unsatisfiable
- Put the list of sentences into conjunctive normal form.

- Until there is no resolvable pair of clauses,
  - Find resolvable clauses and resolve them.
  - Add the results of resolution to the list of clauses.
  - If NIL (empty clause) is produced, stop and report that the (original) theorem is true. (empty clause represents contradiction)
- Report that the (original) theorem is false

# Algorithm: Resolution Proof

- May seem cumbersome but note that can be easily automated. Just “smash” clauses till empty clause or no more new clauses

- Guaranteed sound and (refutation) complete.

- Q. Why is method with axioms more difficult to implement?

# Example

1) ARM-OK
2) ¬MOVES
3) ARM-OK $\wedge$ LIFTABLE $\Rightarrow$ MOVES
4) ¬ARM-OK $\vee$ ¬LIFTABLE $\vee$ MOVES
Prove: ¬LIFTABLE

5) LIFTABLE (assert)
6) ¬ARM-OK $\vee$ MOVES (resolving 5 and 4)
7) ¬ARM-OK ( from 6 and 2)
8) Nil (empty clause / contradiction, from 7 and 1).

# Another Example

$KB = (B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})) \wedge \neg B_{1,1}$

$\alpha: \neg P_{1,2}$

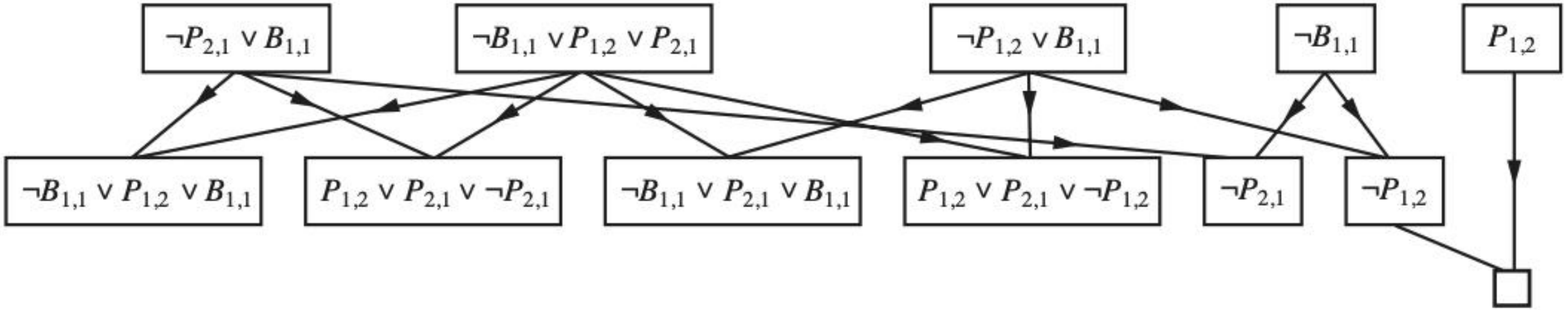
- First convert $(KB \wedge \neg\alpha)$ into CNF

# Another Example

- First convert $(KB \wedge \neg\alpha)$ into CNF
- $(B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})) \wedge \neg B_{1,1} \wedge P_{1,2}$
- $((B_{1,1} \Rightarrow (P_{1,2} \vee P_{2,1})) \wedge ((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1})) \wedge \neg B_{1,1} \wedge P_{1,2}$ (biconditional)
- $((\neg B_{1,1} \vee (P_{1,2} \vee P_{2,1})) \wedge (\neg(P_{1,2} \vee P_{2,1}) \vee B_{1,1}) \wedge \neg B_{1,1} \wedge P_{1,2}$ (Implication elimination)
- $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge ((\neg P_{1,2} \wedge \neg P_{2,1}) \vee B_{1,1}) \wedge \neg B_{1,1} \wedge P_{1,2}$ (De Morgan)
- $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg P_{1,2} \vee B_{1,1}) \wedge (\neg P_{2,1} \vee B_{1,1}) \wedge \neg B_{1,1} \wedge P_{1,2}$ (Distri.)

# Another Example

- Resolution proof



# Completeness of Resolution

- Rosolution closure (RC): the set of all clauses derivable by repeated application of the resolution rule to clauses in S or their derivatives

- Ground resolution theorem: If a set of clauses is unsatisfiable, then the resolution closure of those clauses contains the empty clause

- Proof by contrapositive:

  if the closure RC(S) does not contain the empty clause, then S is satisfiable

# Completeness of Resolution

- Construct a model for S with suitable truth values for $P_1$, ..., $P_k$

For i from 1 to k,

- If a clause in RC(S) contains the literal $\neg P_i$ and all its other literals are false under the assignment chosen for $P_1$, ..., $P_{i-1}$, then assign false to $P_i$.

- Otherwise, assign true to $P_i$.

This assignment to $P_1$,...,$P_k$ is a model of S.

# Completeness of resolution

- Assume at some stage i in the construction, assigning $P_i$ causes some C to be false.
- C must look like (false $\vee$ false ..., $\vee P_i$) or (false $\vee$ false ..., $\vee \neg P_i$)
- C can only be falsified if both of these clauses are in RC(S)
- RC(S) will contain resolvent of these two clauses (already falsified)
- Contradiction

# Horn clauses and Definite Clauses

- Definite clauses: a disjunction of literals of which exactly one is positive
  - $\neg L_{1,1} \vee \neg P_{1,2} \vee B_{1,1,}$
- Horn clauses: a disjunction of literals of which at most one is positive
- Horn clauses are closed under resolution

# Horn clauses and Definite Clauses

- Every definite clause can be written as an implication whose premise is a conjunction of positive literals and whose conclusion is a single positive literal.
  - $(L_{1,1} \wedge P_{1,2}) \Rightarrow B_{1,1,}$
- Inference with Horn clauses can be done through the forward-chaining and backward-chaining algorithms
- Deciding entailment with Horn clauses can be done in time that is linear in the size of the knowledge base

# Forward Chaining

- From facts to conclusions

- It begins from known facts (positive literals) in the knowledge base
- If all the premises of an implication are known, then its conclusion is added to the set of known facts

- Given $(L_{1,1} \wedge P_{1,2}) \Rightarrow B_{1,1}$, If $L_{1,1}$ and $P_{1,2}$, then $B_{1,1}$ can be added to the knowledge base

# Forward Chaining

- It begins from known facts (positive literals) in the knowledge base
- If all the premises of an implication are known, then its conclusion is added to the set of known facts

$$P \Rightarrow Q$$

$$L \wedge M \Rightarrow P$$

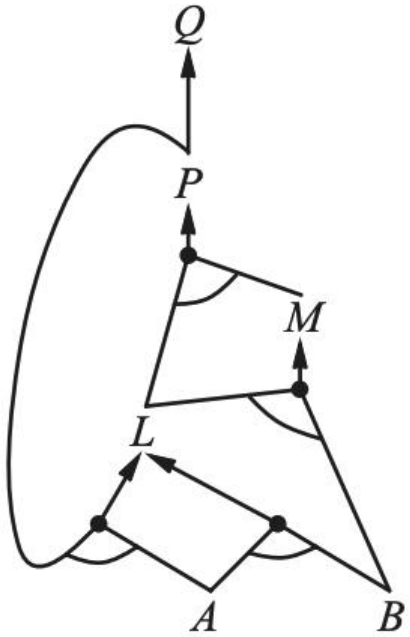
$$B \wedge L \Rightarrow M$$

$$A \wedge P \Rightarrow L$$

$$A \wedge B \Rightarrow L$$

$$A$$

$$B$$



# Forward Chaining

- Sound: every inference is essentially an application of Modus Ponens

- Complete: every entailed atomic sentence will be derived.

# Backward Chaining

- From goal to data
- Assume the query q is true and works backward
- If all the premises of one of those implications can be proved true (by backward chaining), then q is true

# Propositional logic is a weak language

- Hard to identify “individuals” (e.g., Mary, 3)
- Can’t directly talk about properties of individuals or relations between individuals (e.g., “Bill is tall”)
- Generalizations, patterns, regularities can’t easily be represented (e.g., “all triangles have 3 sides”)

# Length of Resolution Proof

Consider Pigeon-Hole (PH) problem: Formula encodes that you cannot place n+1 pigeons in n holes (one per hole).

PH takes exponentially many steps (no matter in what order)!

PH hidden in many practical problems. Makes theorem proving expensive.

# Pigeon-Hole Principle

$P_{i,j}$ for Pigeon i in hole j.

$$P_{1,1} \vee P_{1,2} \vee P_{1,3} \ldots P_{1,n}$$

$$P_{2,1} \vee P_{2,2} \vee P_{2,3} \ldots P_{2,n}$$

....

$$P_{(n+1),1} \vee P_{(n+1),2} \vee P_{(n+1),3} \cdots P_{(n+1),n}$$

and ??

# Pigeon-Hole Principle

$(\neg P_{1,1} \vee \neg P_{1,2}), (\neg P_{1,1} \vee \neg P_{1,3}), (\neg P_{1,1} \vee \neg P_{1,4}), ...$

$(\neg P_{1,(n-1)} \vee \neg P_{1,n}),$

$(\neg P_{2,1} \vee \neg P_{2,2})...(\neg P_{2,(n-1)} \vee \neg P_{2,n})$

etc.

$(\neg P_{1,1} \vee \neg P_{2,1}), (\neg P_{1,1)} \vee \neg P_{3,1}), ...$

$(\neg P_{1,2} \vee \neg P_{2,2}), (\neg P_{1,2} \vee \neg P_{3,2}),$ etc.

# Pigeon-Hole Principle

Resolution proof of inconsistency requires at least an exponential number of clauses, no matter in what order you resolve things!

# A More Concise Formulation

$\forall x\ \exists y\ (x \in \text{Pigeons})\ (\ y \in \text{Holes})\ IN(x, y)$

$\forall x \forall x' \forall y (IN(x,y) \wedge IN(x', y)$ ... ??

$\forall x \forall y \forall y' (IN(x,y) \wedge IN(x, y')$ ... ??

Pigeons = $\{p_1, p_2, ..., p_{n+1}\}$,

Holes = $\{h_1, h_2, ..., h_n\}$

We have first-order logic with some set-theory notation.

# Midterm review

- Uninformed search
  - Breadth first search
  - Depth first search
  - Uniform cost search
  - Iterative deepening search
  - bidirectional search

# Midterm review
# Generic Tree-Search Algorithm

Add initial state to the frontier

Loop

node = remove-frontier( )  -- and save in order to return as part of path to goal

if goal-test(node) = true return path to goal

S = successors(node)

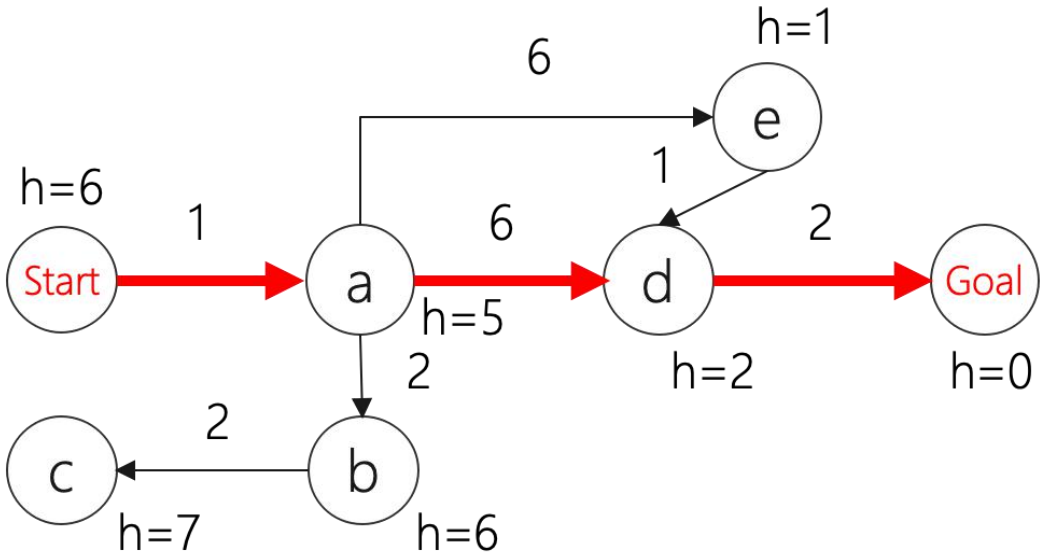
Add S to frontier

until frontier is empty
return failure

# Midterm review

- Informed search
  - Greedy Best-First Search
    - Expands the node that is “closest” to the goal as measured by h(n)
  - A* search
    - Combining Uniform-Cost Search and Greedy Best-First Search
    - f(n) = g(n) + h(n)
      - g(n): the path cost from the start node to node n
      - h(n): the estimated cost of the cheapest path from node n to the goal node

# Midterm review
# A* Search




- Uniform-cost orders by path cost g(n)
- Greedy best first search orders by estimated goal proximity h(n)
- A* search combines g(n) and h(n)

# Midterm review

- If h is admissible, then the A* tree search is optimal
- If h is consistent, then the A* graph search is optimal

# Midterm review
# Hill Climbing Search

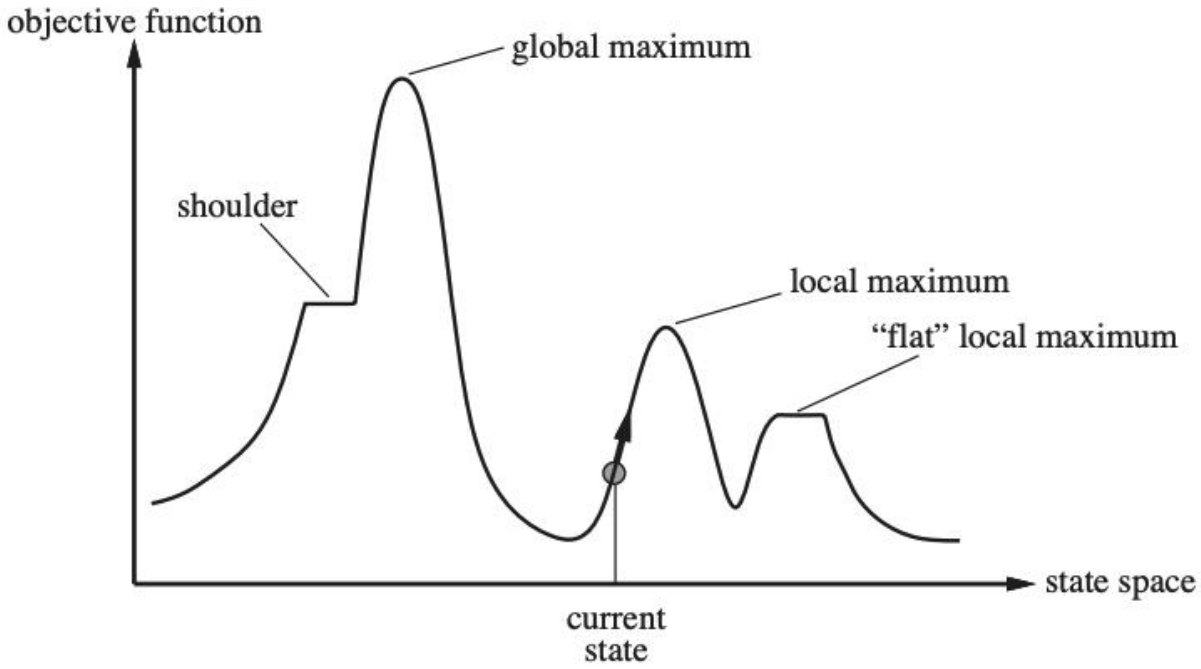
- Move in the direction of increasing value
- Terminate when it reaches a “peak” where no neighbor has a higher value.

```
function HILL-CLIMBING(problem) returns a state that is a local maximum

    current ← MAKE-NODE(problem.INITIAL-STATE)
    loop do
        neighbor ← a highest-valued successor of current
        if neighbor.VALUE ≤ current.VALUE then return current.STATE
        current ← neighbor
```

# Midterm review
# State-Space Landscape

- Local Search Methods rely on an objective function or a cost function to compute the state-space landscape



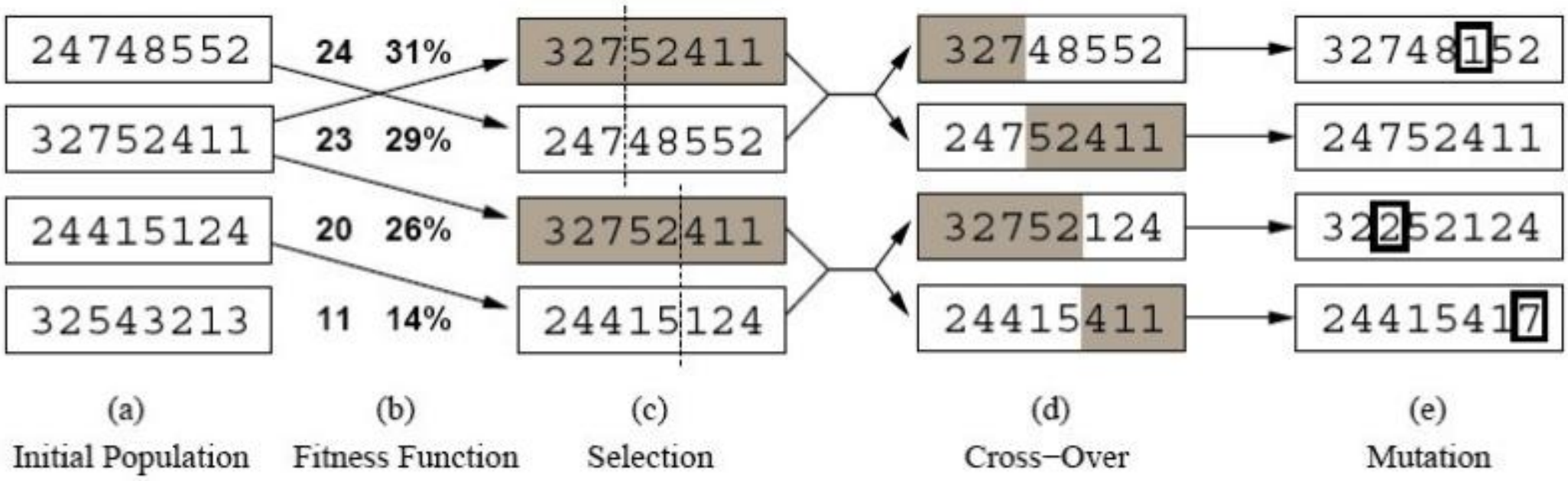

- Local search algorithms explore this landscape

# Midterm review
# Improvements to Basic Local Search

- Issue: How to move more quickly to successively higher plateaus and avoid getting “stuck” / local minima

- Idea: Introduce uphill moves (“noises”) to escape from long plateaus (or true local minima).
- Strategies:
  - Simulated Annealing
  - Random-restart hill-climbing
  - Tabu search
  - Local beam search
  - Genetic Algorithms

# Midterm review
# Genetic Algorithm



# Midterm review
# Crossover Operators

- Single-point crossover:

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Parent A: | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 1 |
| Parent B: | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 |
| | | | | | | | | | | |
| Child AB: | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 |
| Child BA: | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 |

# Midterm review
# Mutation

- Mutation: randomly flip one bit

Individual A: 1 0 0 1 0 1 1 1 0 1

Individual A': 1 1 0 0 0 1 1 1 0 1

# Machine Learning

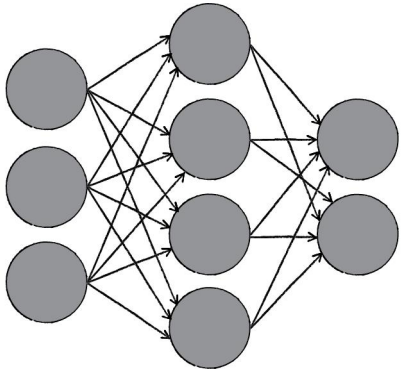
Task: classify a given image

2

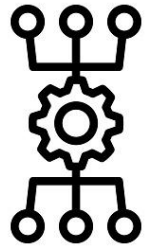
---

Data

Model



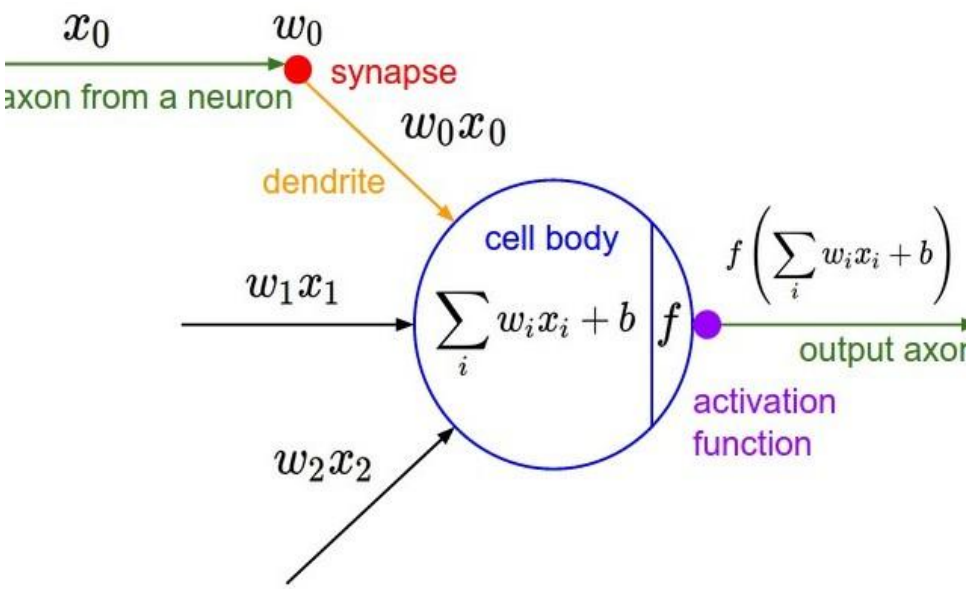
Algorithm

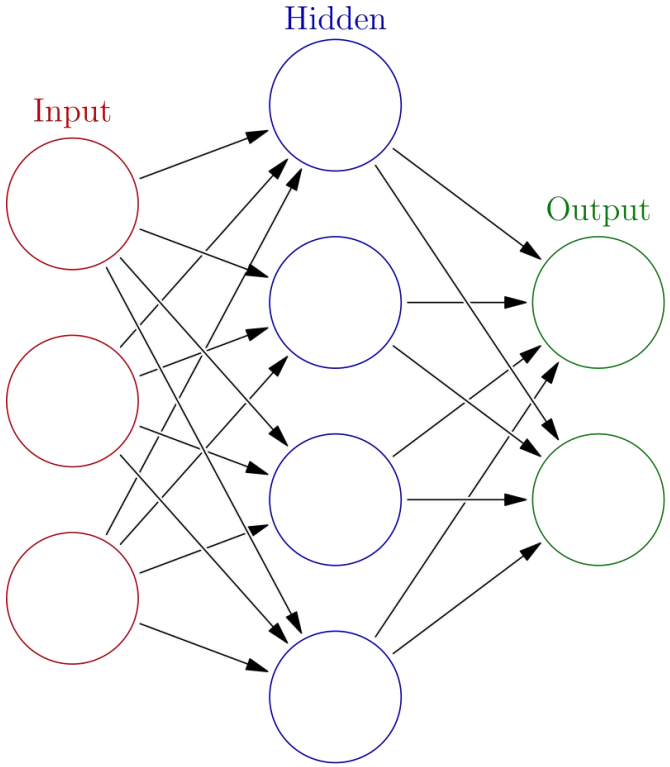


# Neural Networks

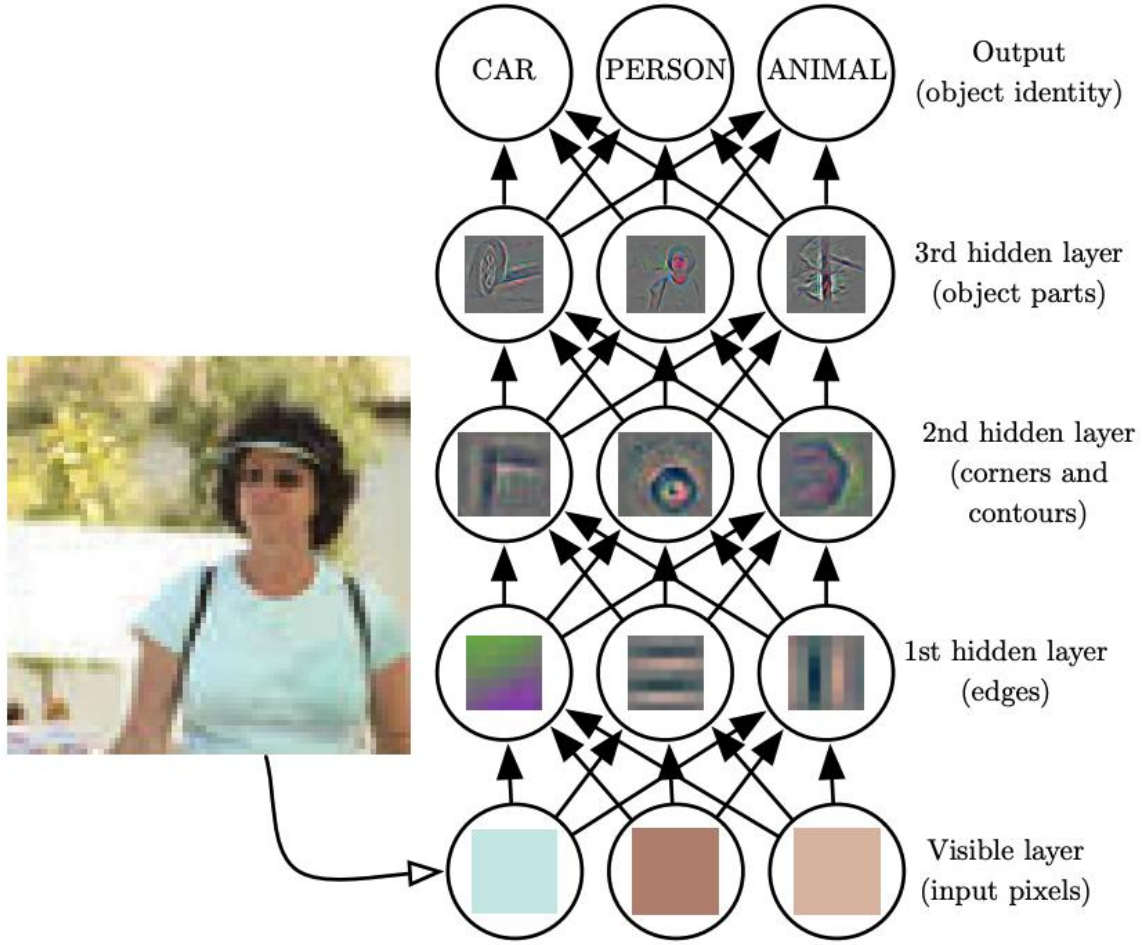
- Neural networks consist of a collection of nodes which model biological neurons in human brain.

- Neural networks have one input layer, one or multiple hidden layers and one output layer



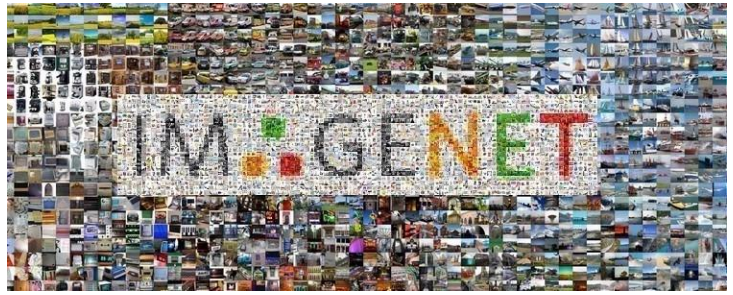



# What are learnt by neural networks

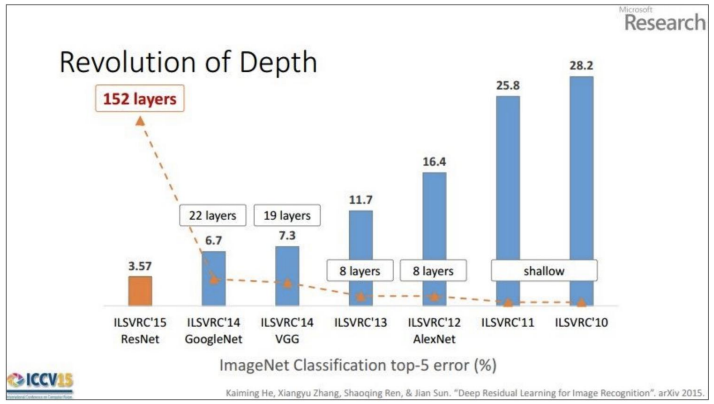


# Deep Neural Networks: The Success

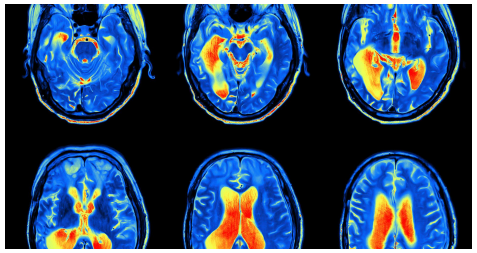
## Large-Scale Dataset




## Top-5 Error on ImageNet



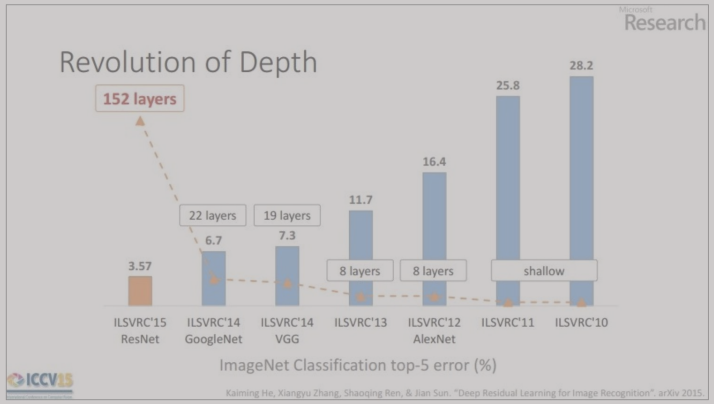




# Practical Deployment

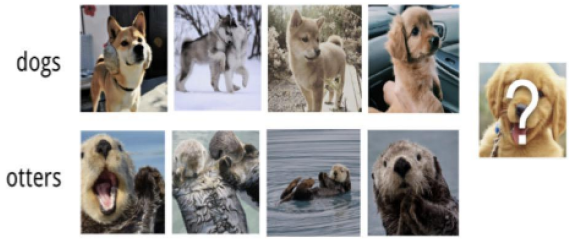
## Large-Scale Dataset


## Top-5 Error on ImageNet

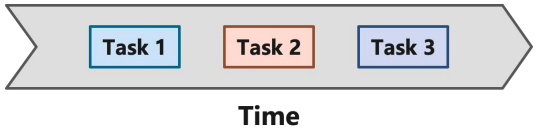



## Limited Resources

## Limited Examples

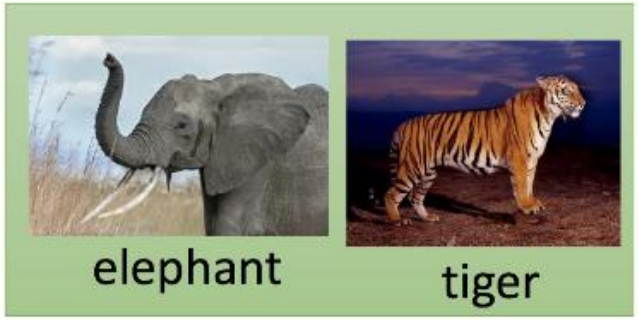



## A Sequence of Tasks



# Tranfer Learning

Dataset 1



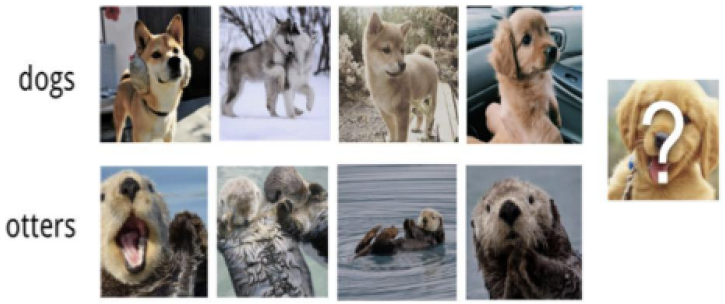

Knowledge extracted from dataset 1 can be reused for learning on dataset 2

Dataset 2

# Few-shot Learning

N-way K-shot few-shot learning task:

- Each task has a total of N classes.
- Each class has K examples.



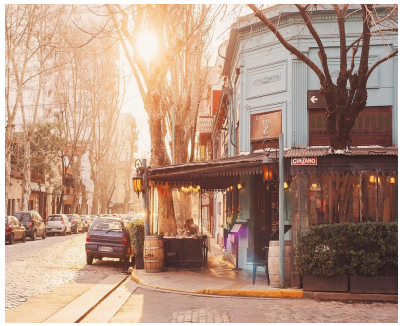

2- way 4-shot

# Sequential Tasks

Rainy

Cloudy

Sunny



Dog or Cat

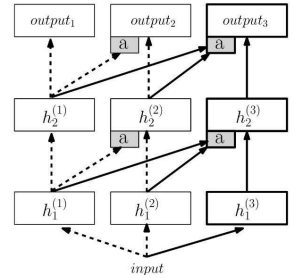
Tiger or Lion

Dolphin or Shark

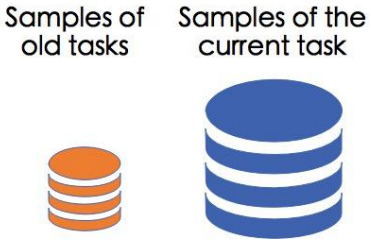
# Lifelong Learning

$$\mathcal{L}(\theta) = \mathcal{L}_B(\theta) + \sum_i \frac{\lambda}{2} F_i(\theta_i - \theta^*_{A,i})^2$$

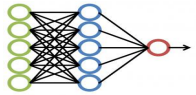
- Regularization based approach
  - Prevent important weights for old tasks from changing drastically.




- Knowledge transfer based method
  - Build connections between the models for the new task and old tasks.




- Episodic memory based approach
  - Store a small subset of examples from old tasks



# Semantic Segmentation