ECE408/CS483/CSE408 Spring 2025

Applied Parallel Programming

Lecture 10:

Machine Learning and Deep Learning

Course Reminders

- Lab 4 is due this week
- Midterm 1 is on Tuesday, March 4th
  - 7-10pm, in-person, in ECEB
  - Includes materials from lectures 1-12 and labs 1-4
- Project Milestone 1: Baseline CPU/GPU implementation
  - Released, see GitHub

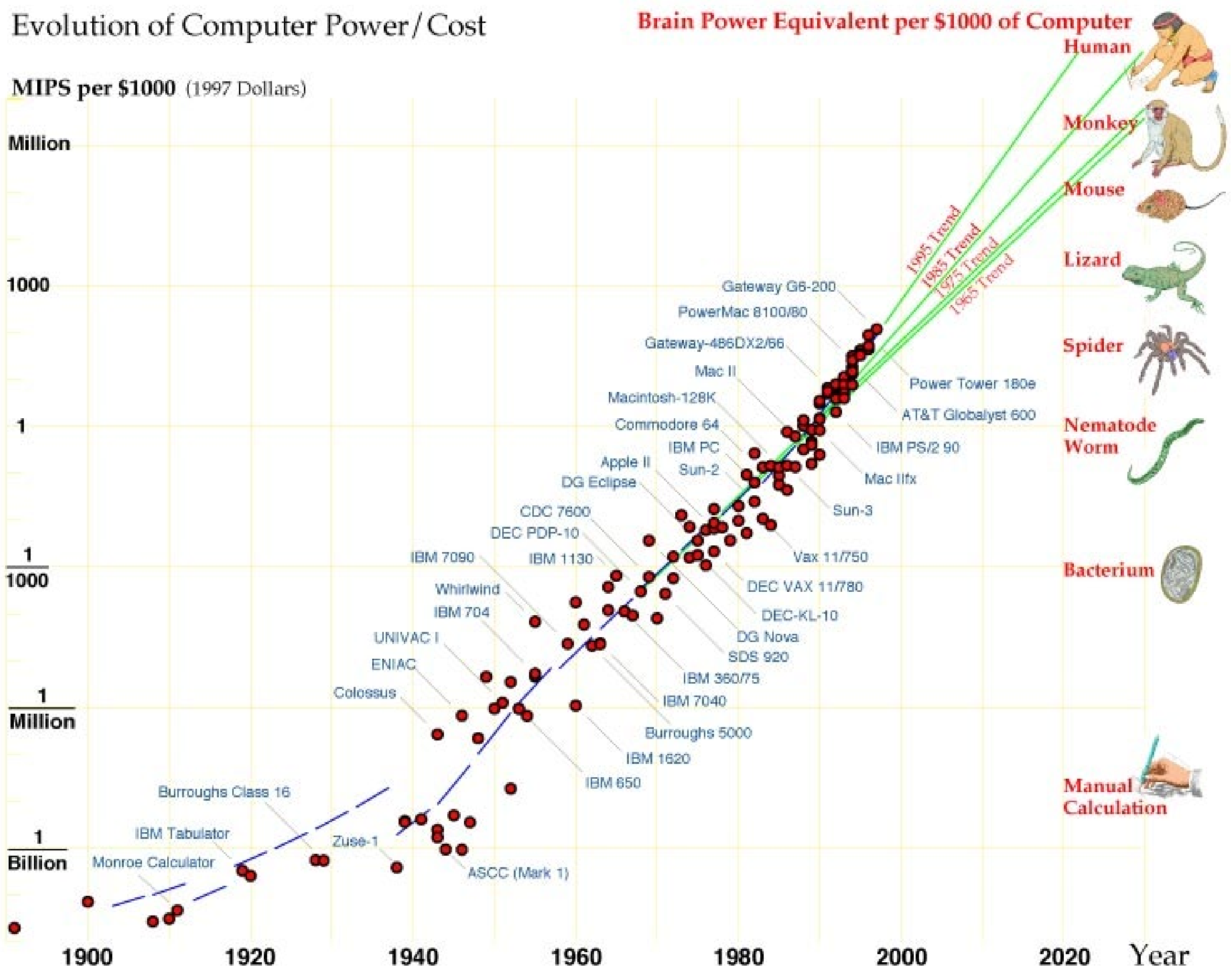
Objective

- To understand the application areas for machine learning.
- To learn the basic strategy for machine learning applications.
- To understand the extension to deep learning (mostly a research pitch).
- To learn about a Multi-Layer Perceptron.

Perspective is Important

- **Chips are cheaper than ever**
- Unlike humans, digital systems offer
  - **high-speed computation**,
  - **low capital investment**
    (purchase vs. training a human), and
  - **negligible operations cost** (no salary!)
- **If computer outperforms** (or even matches) a **human, use a computer**
- **Industry has done so for 40-50 years now**

Evolution of Computer Power/Cost



Hans Moravec, 1997

Computing has evolved under the premise that some day, computing machines will be able to mimic general human intelligence.

From a computing power perspective, Moore's Law has fueled the idea of the intelligent machine. Hardware has gotten 2x faster every 18 months.

The software, though, has been a vexing open question.

https://jetpress.org/volume1/moravec.htm

What is Machine Learning?

- **Machine learning**: important method of building applications whose logic is not fully understood
- Typically **by example**:
  - **use labeled data** (matched input-output pairs)
  - **to represent** desired **relationship**
- **Iteratively adjust program logic** to produce desired/approximate answers (called **training**)

Types of Learning Tasks

- classification
  - Map each input to a category
  - Ex: object recognition, chip defect detection
- regression
  - Numerical prediction from a sequence
  - Ex: predict tomorrow's temperature
- transcription
  - Unstructured data into textual form
  - Ex: optical character recognition

More Advanced Learning Tasks

- translation
  – Convert a sequence of symbols in one language to a sequence of symbols in another
- structured output
  – Convert an input to a vector with important relationships between elements
  – Ex: natural language sentence into grammatical structure
- others
  – Anomaly detection, synthesis, sampling, imputation, denoising, density estimation, genetic variant calling

Why Machine Learning Now?

- **Computing Power**
  - GPU computing hardware and programming interfaces such as CUDA has enabled very fast research cycle of deep neural net training
- **Data**
  - Lots of cheap sensors, cloud storage, IoT, photo sharing, etc.
- **Needs**
  - Autonomous Vehicles, Smart Devices, Security, Societal Comfort with Tech, Health Care

Test Cycle Time is Important

You've all written code…

- code, test, code, test, code, test
- integrate, test, test, test
- and test again!

But how long is the code, test cycle?

Depends what you're building.

What's your longest?

Your Cycle Times are Probably Small

- In college, **10k lines** took **½ hour** to compile on my PC.
- In grad. school, **100k lines** took
  - **½ hour** to compile on my workstation, or
  - **2 minutes** on our cluster (research platform)
- In ECE 435 (networking lab), students needed
  - **½ hour** to reinstall Linux after a bad bug
  - (Ever had a good bug?)
- Gene sequencing / applications can take **two weeks**

We're all a little spoiled…

Why Machine Learning Again?

- In 2007, **programmable GPUs accelerated the training cycle**
- Today, **new chip designs** for learning applications **have further accelerated**
- Led to a resurgence of interest
  - in Computer Vision, Speech Recognition, Document Translation, Self Driving Cars, Data Science…
  - all tasks that **human brains solve regularly**, **but** for which **we** have **struggled to express solutions** systematically.

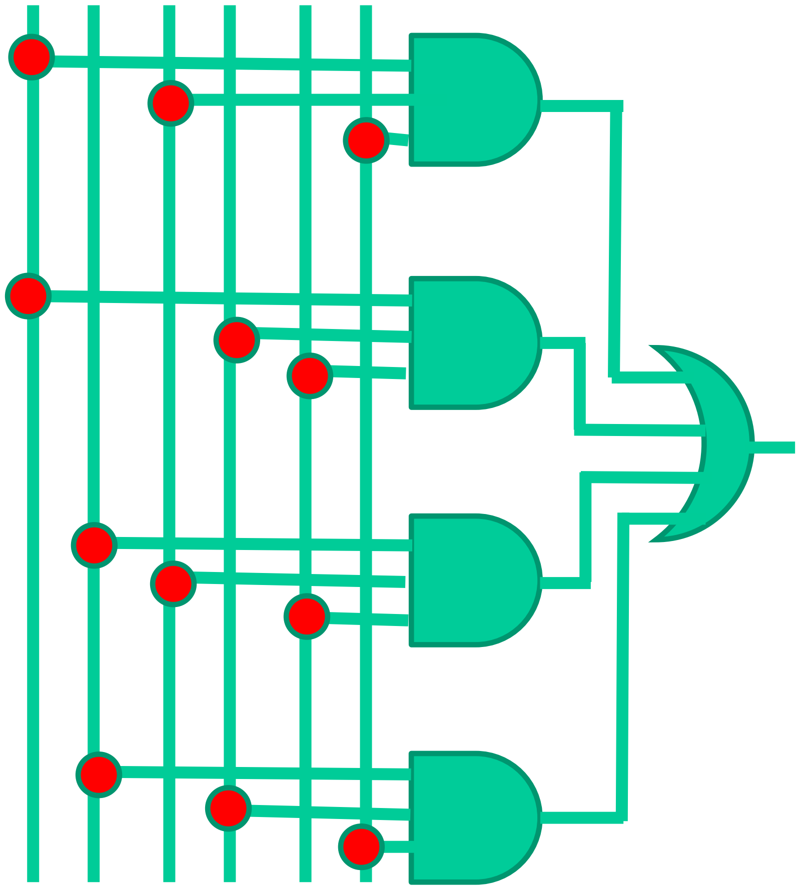
Many Problems are Still Hard

- **Speed is not a panacea**
- Many **tasks still require human insight**
  - for network structure and feature selection
  - for effective input and output formats, and
  - for production of high-quality labeled data.
- Other trends sometimes help: ubiquitous computing enables crowdsourcing, for example.

Many Problems Have Systematic Solutions

Example: building a Boolean function from a truth table

Input			output
a	b	c	
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1



What if We Lack a Truth Table?

- Make enough observations to construct a rule
  - 000 → 0
  - 011 → 0
  - 100 → 1
  - 110 → 0

- If we cover all input patterns, we can construct a truth table!

Many Problems are Too Large

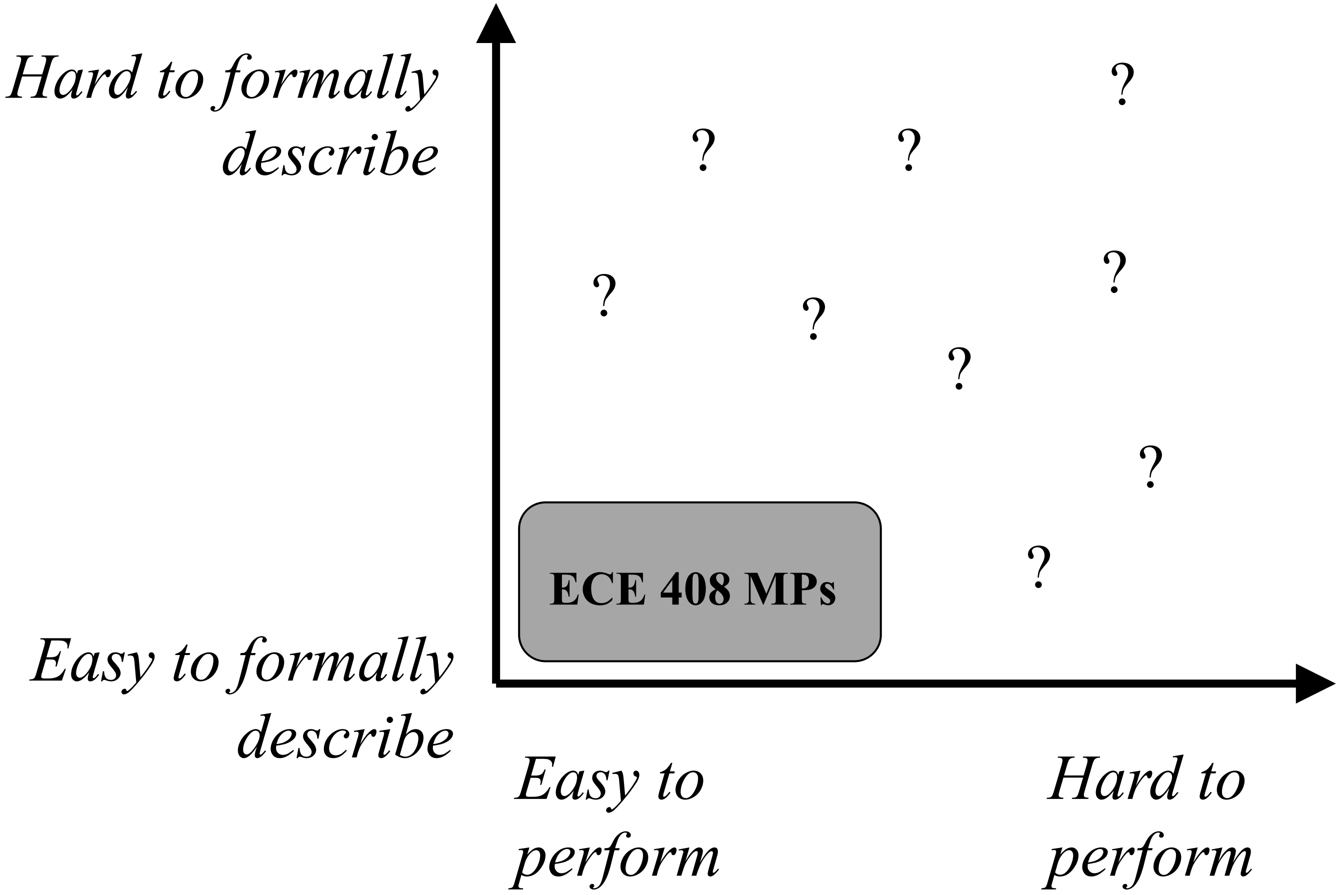
- The logic formulation of a 32x32-pixel (small) image recognition problem involves
  - 1024*8 bit input,
  - which will have a truth table of 2^{8196} entries
- If we managed to collect and label 1 billion ($\sim 2^{32}$) images as training data
  - We cover only $2^{32} / 2^{8196} = 1/ 2^{8164}$ of the truth table
  - Solution - learning processes that exploits features

Features in our logic example

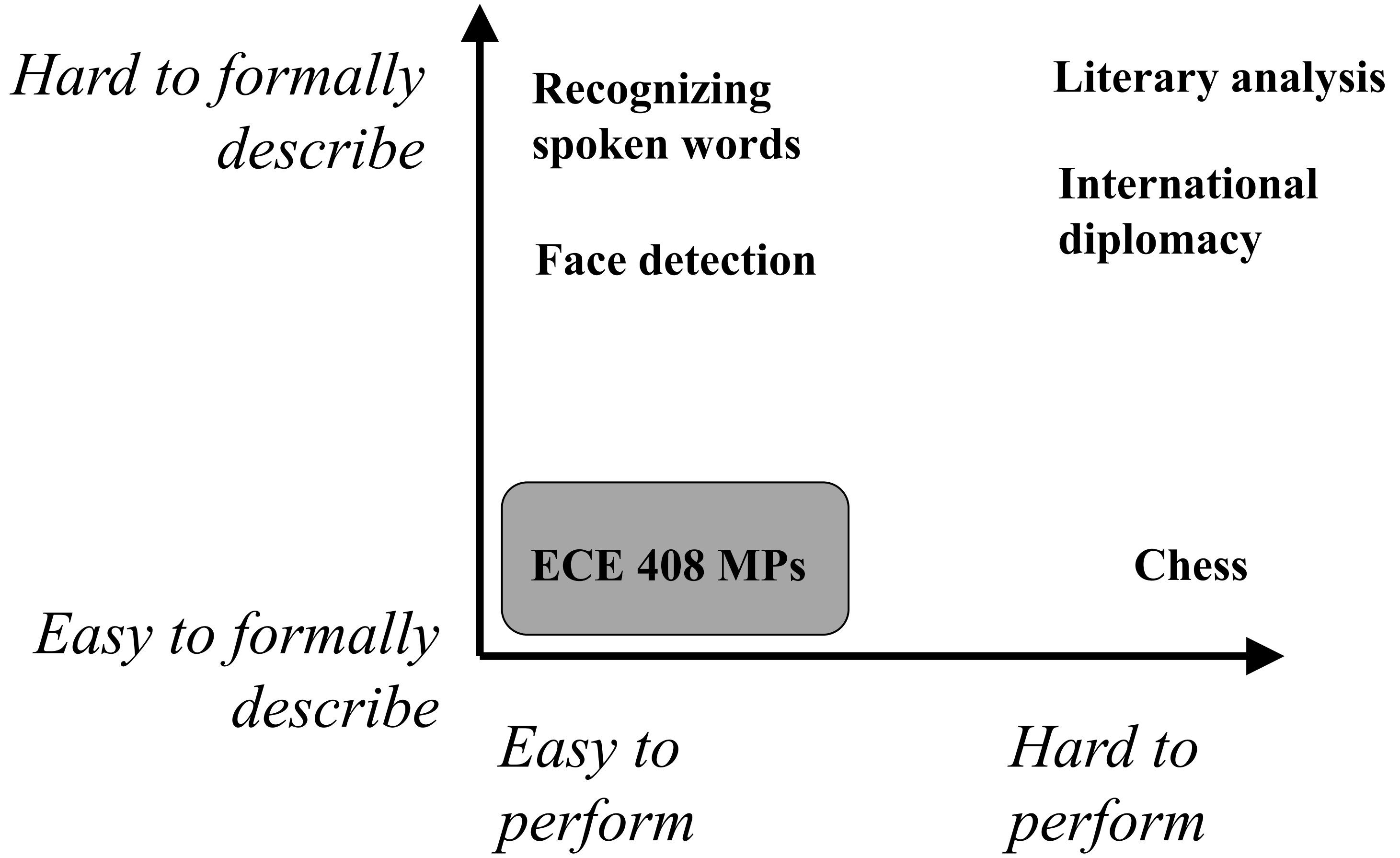
Input			output
a	b	c	
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

Feature 1: bit patterns with odd number of 1's result in output 1
Feature 2: bit patterns with even number of 1's result in output 0

Types of Problems



Types of Problems



(Algorithm complexity, parallelism, and data bandwidth)

Chess as an AI Success (1)

- Easy to formalize
  - 64 locations, 32 pieces
  - Well-defined, allowable moves
- Score each leaf in a tree of possible board positions
- Proceed down path that results in best position

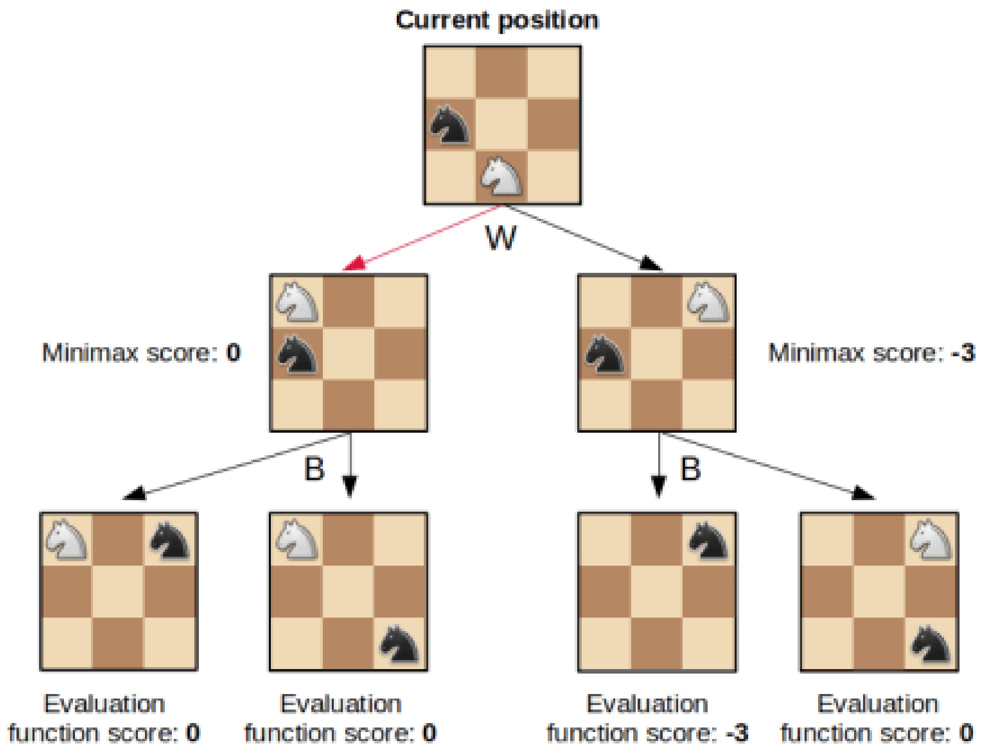


Image taken from https://www.r-bloggers.com/2022/07/programming-a-simple-minimax-chess-engine-in-r/

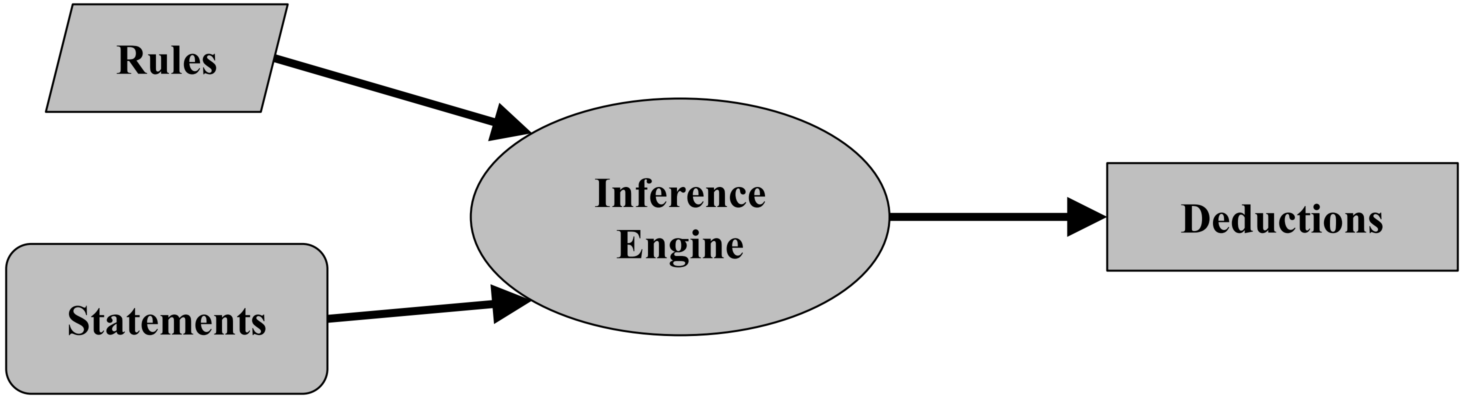
Chess as an AI Success (2)

Deep Blue defeated Gary Kasparov
in 1997

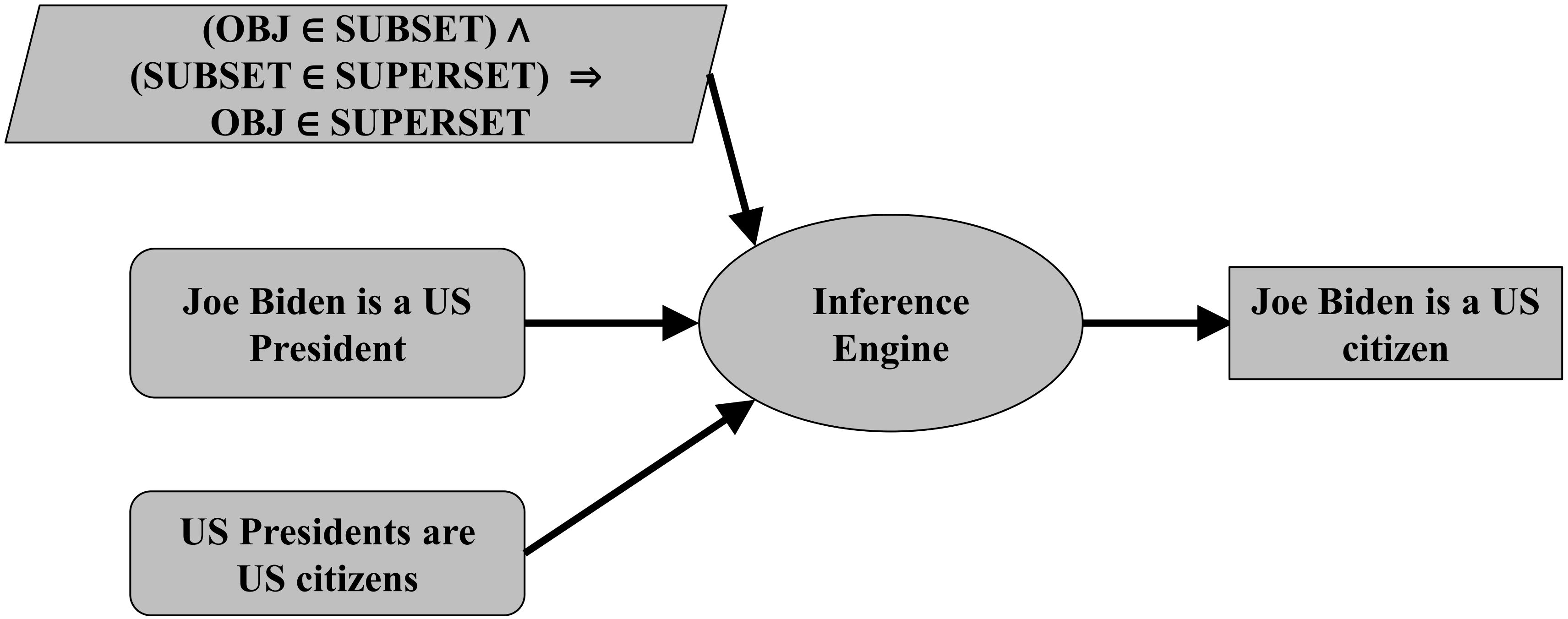
- Hard to perform
  - ~30 legal moves per position
  - 1,015 moves for 10-ply lookahead
  - 30 years of compute at 1M positions/sec
- Heuristics, pruning, parallel search, fast computers

Cyc: Extending Rule-based Systems to the Real World

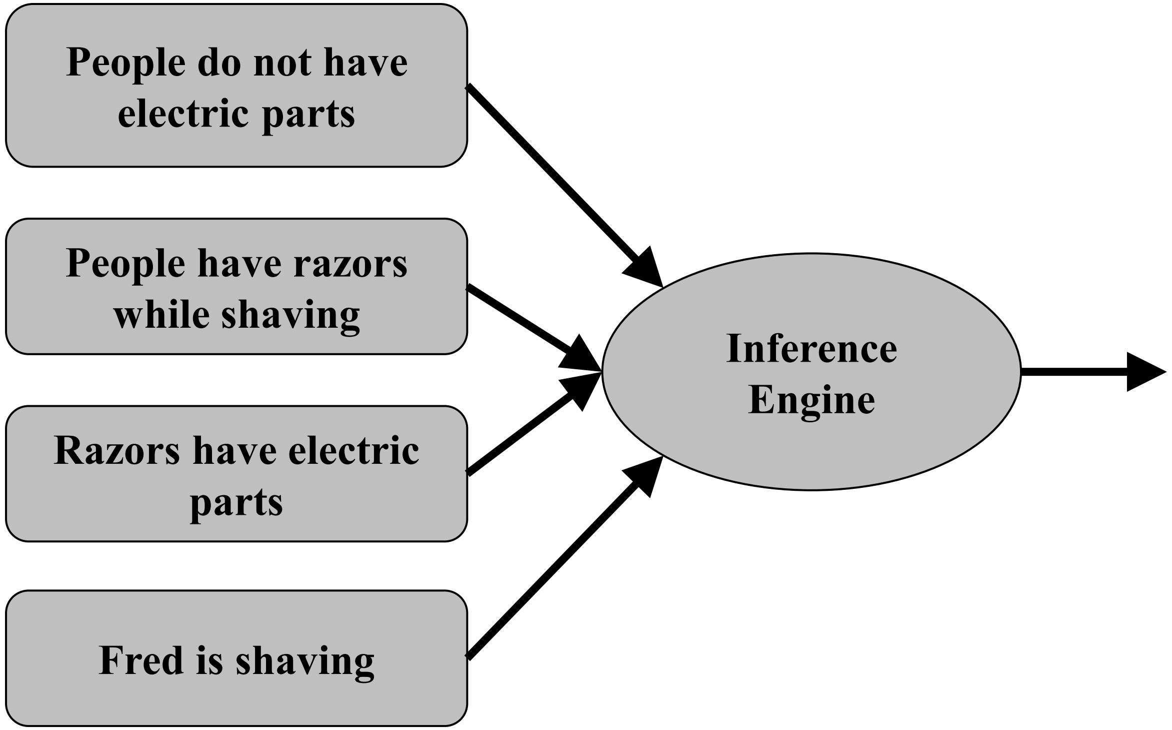
- Comprehensive ontology and knowledge base of common sense
- Cyc reasons about formal statements about the world



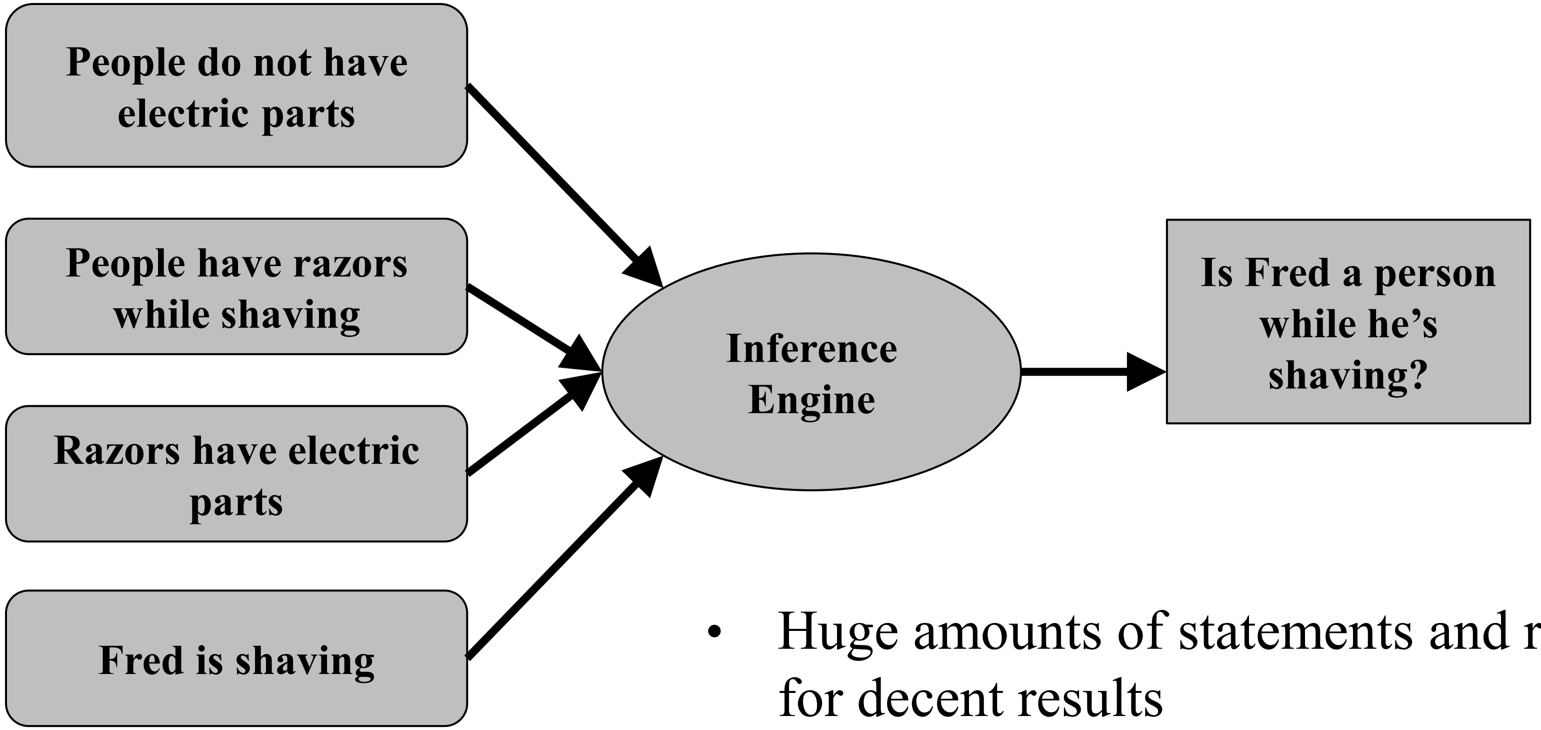
Cyc: A Simple Example



Cyc: FredWhileShaving

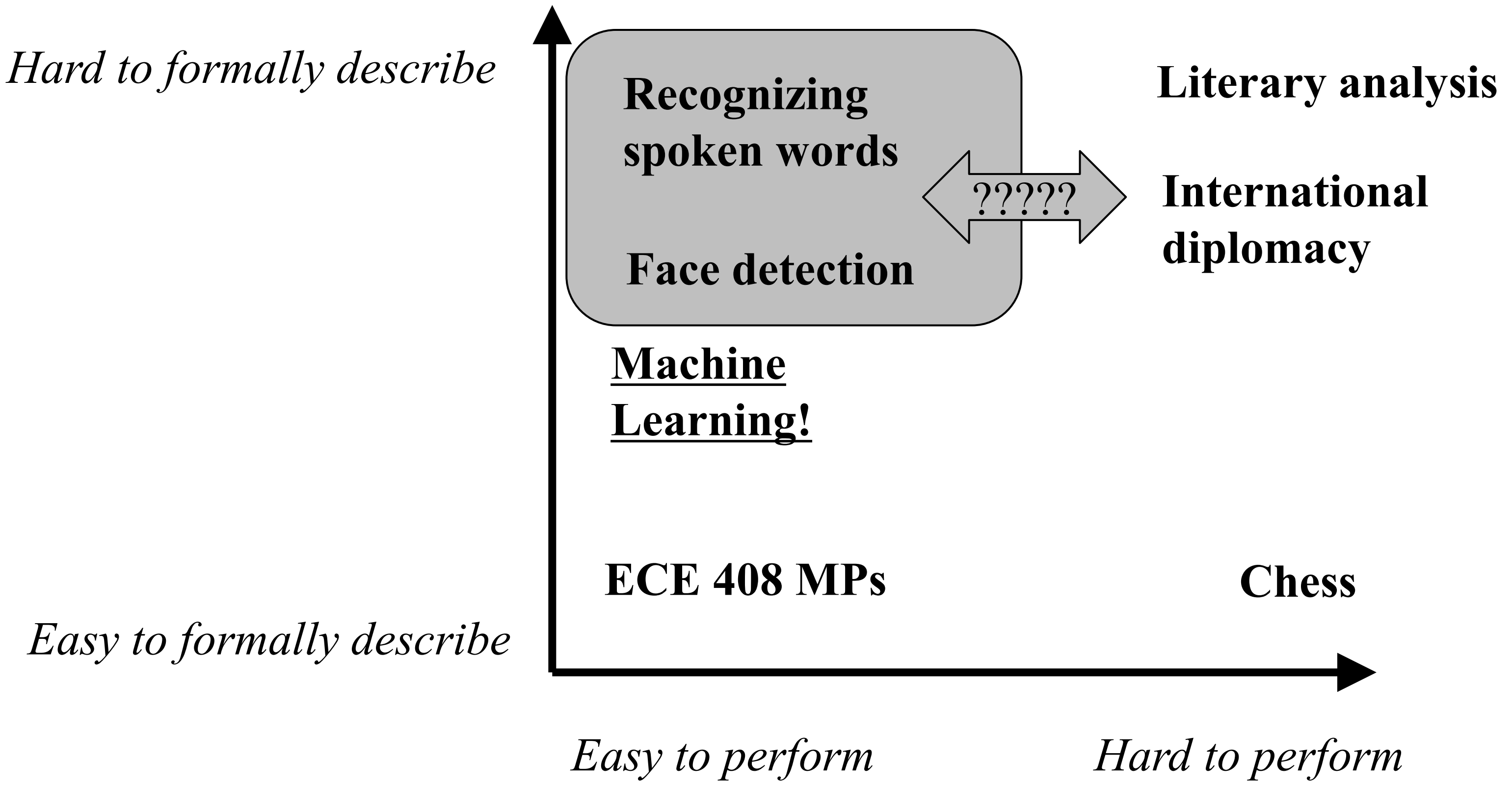


Cyc: FredWhileShaving



- Huge amounts of statements and rules for decent results
- Cannot learn new rules or statements on its own

Types of Problems



The “Machine Learning” Approach

Challenge

Hard to formalize the problem.

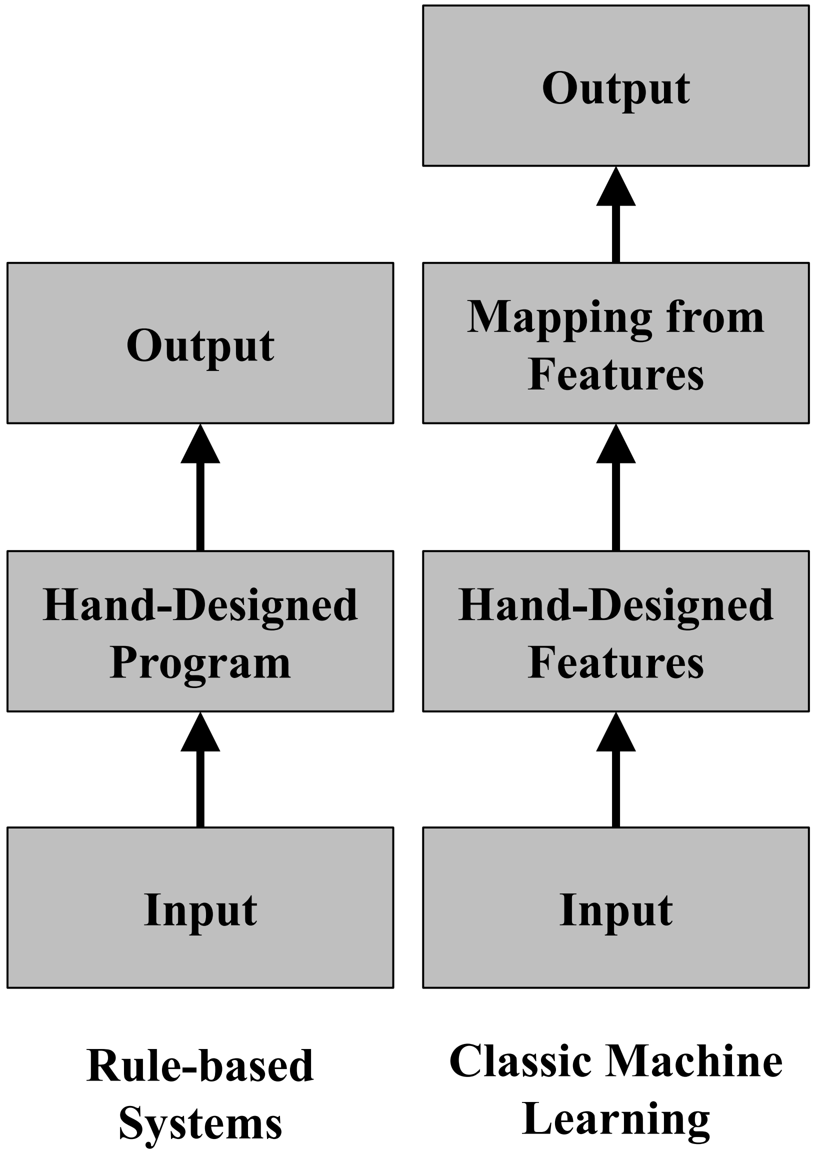
Solution

Don’t formalize the problem.

Let the machine learn from experience.

Classic Machine Learning

- Humans choose features
- Learn how features are associated with outputs

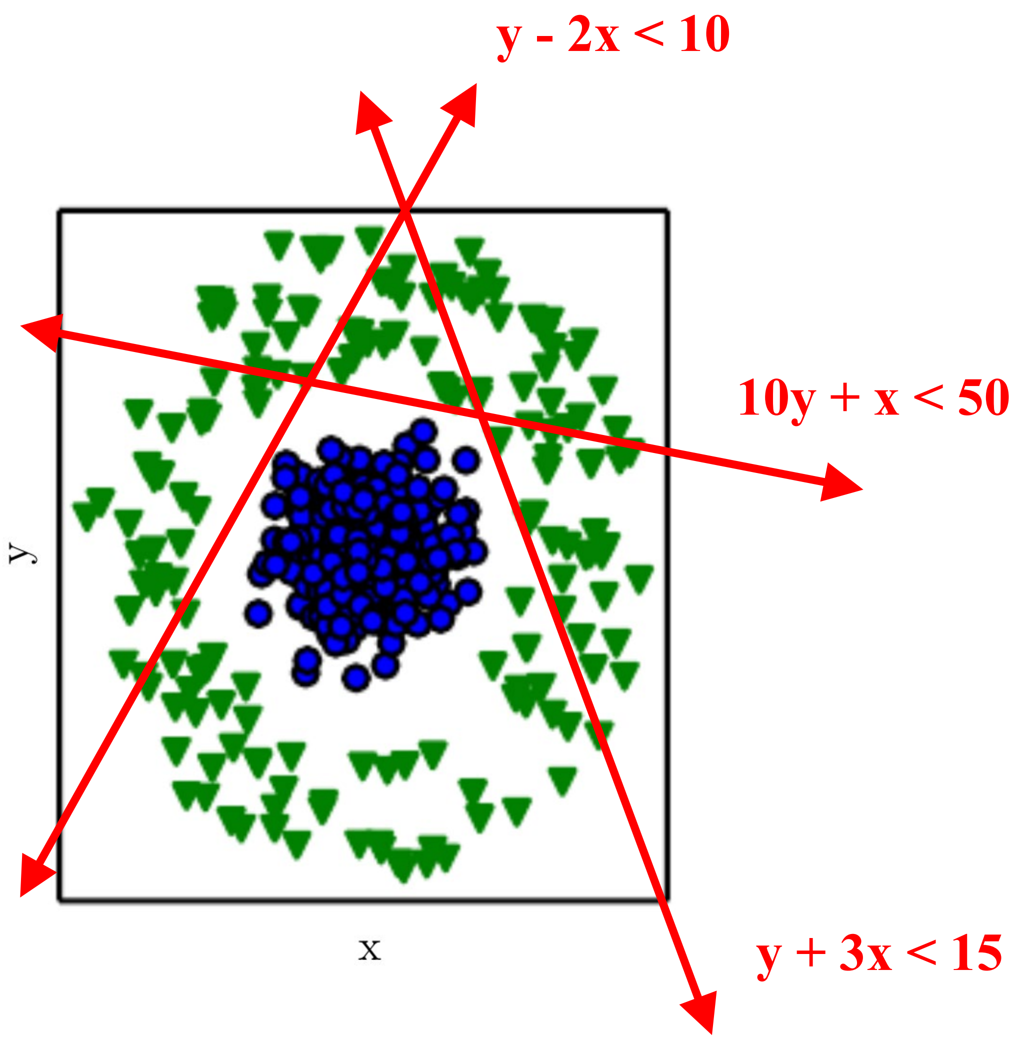


You may have heard of…

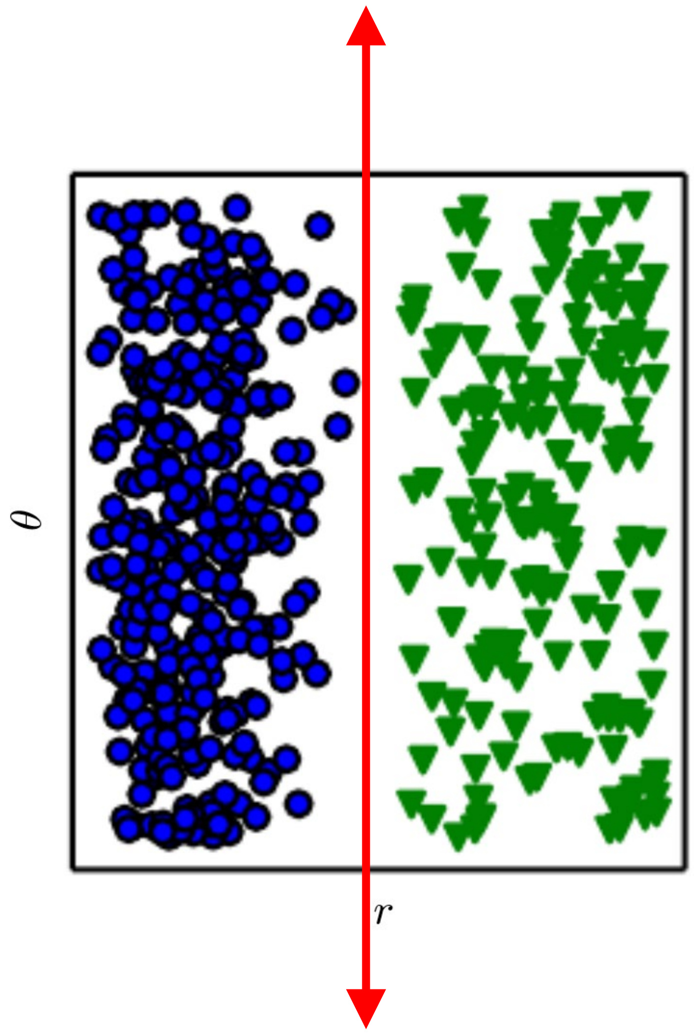
- Naïve Bayes:
  features as independent contributors to output
- Logistic Regression:
  - learn how to weight each feature's contribution to output,
  - usually through gradient descent*

*more on this topic later in these slides

Data Representation is important!



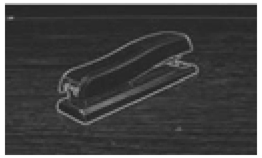
Data Representation is important!



Θ = arctan(y / x)
r = sqrt($x^2 + y^2$)

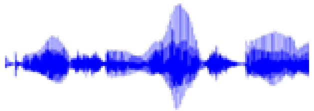
Different Features for Different Tasks

Image

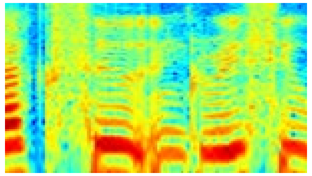


Vision Features

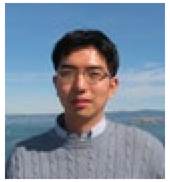
Detection



Audio

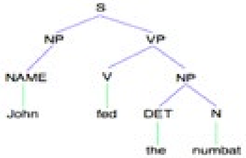


Audio Features



Identify
Speaker

Text



Text Features

Text classification, machine translation, information retrieval

Which Data Features are Relevant

- Detecting a car in an image
- Cars have wheels ➡ presence of a wheel?
- Can we describe pixel values that make up a wheel?
  - Circle-shaped?
  - Dark around perimeter?
- But what about?
  - Occlusion, perspective, shadows, white-walled tires, …

Identify Factors of Variation that Explain Data

- Unobserved objects or forces that affect observed quantities
- Mental constructs that provide simplifying explanations or inferred causes
- Ex: speech
  – Age, sex, accent, words being spoken
- Ex: car
  – Position, color, angle of sun
- Many factors influence each piece of observed data

Representation Learning Approach

Challenge

Which data features are relevant?

Solution

Learn the features too!

(Looking ahead)

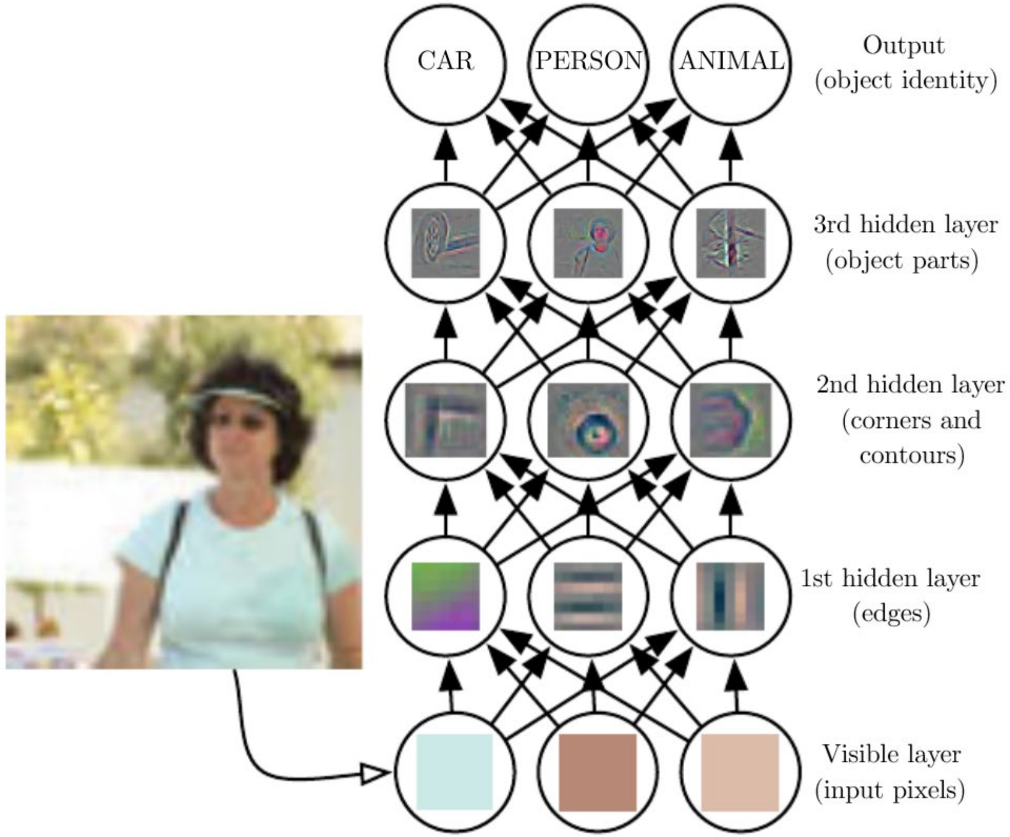
Deep Learning: a deep hierarchy of features

Machine Learning

- Ability to acquire knowledge by extracting patterns from data

Deep Learning

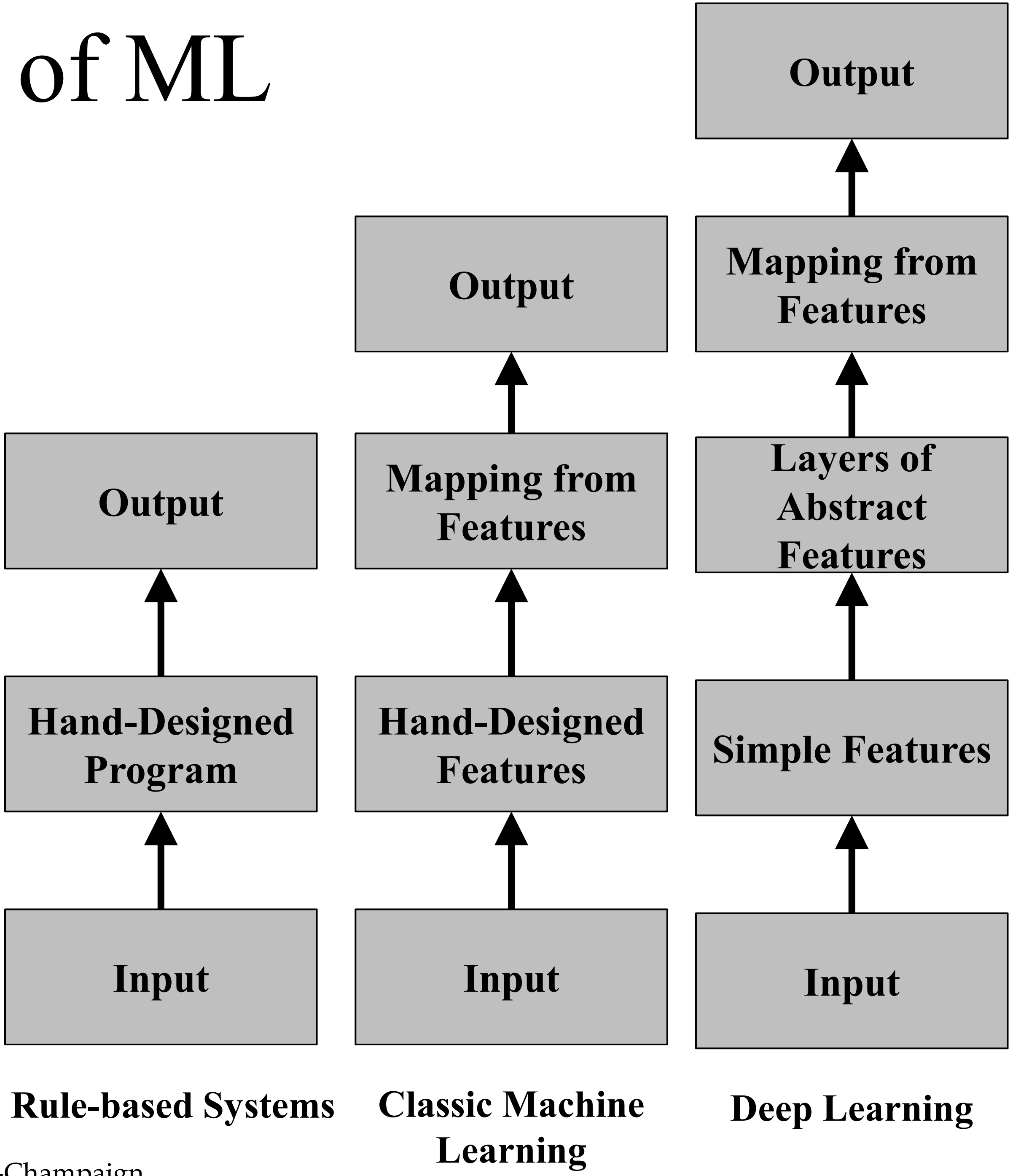
- A type of representation learning
- Representations expressed in terms of other representations



Deep Learning Approach

Challenge	Solution
Hard to formalize the problem?	Don't formalize the problem
	Let the machine learn from experience
Which data features are relevant?	Hierarchy of concepts to capture simple and complicated features
	Learn the hierarchy too!

Evolution of ML



Let's Look at Classification

In a **classification problem**, we model

- a function mapping an input vector to a set of C categories: $F : \mathbb{R}^N \rightarrow \{1, \ldots, C\}$,
- where the function F **is unknown**.

We **approximate** F **using a set of functions** f

- parametrized by a (large) set of weights, θ
- that map from a vector of N real values* to an integer value representing a category:
- for category i, **prob**$(i) = f(x, \theta)$

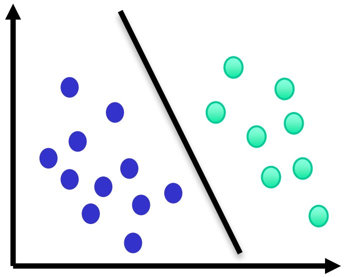
*floating-point values

Linear Classifier (Perceptron)

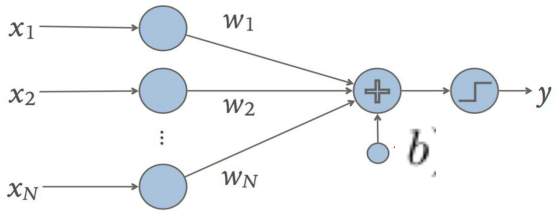
- our formulation: $y = f(\boldsymbol{x}, \Theta)$

$$\Theta = \{W, b\}$$

$$y = sign(W \bullet \boldsymbol{x} + b)$$

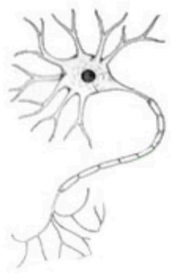


The perceptron



$\approx$

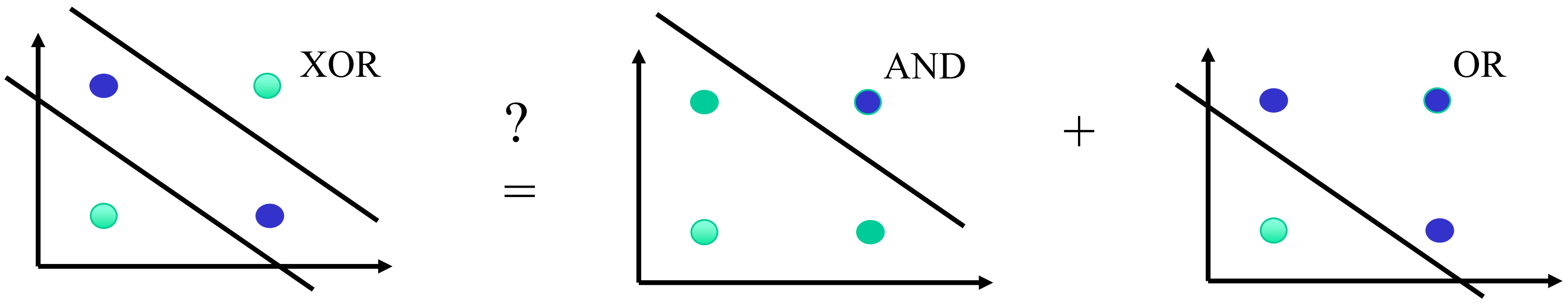
The neuron



- Dot product + Scalar addition: $y = W \bullet \boldsymbol{x} + b$

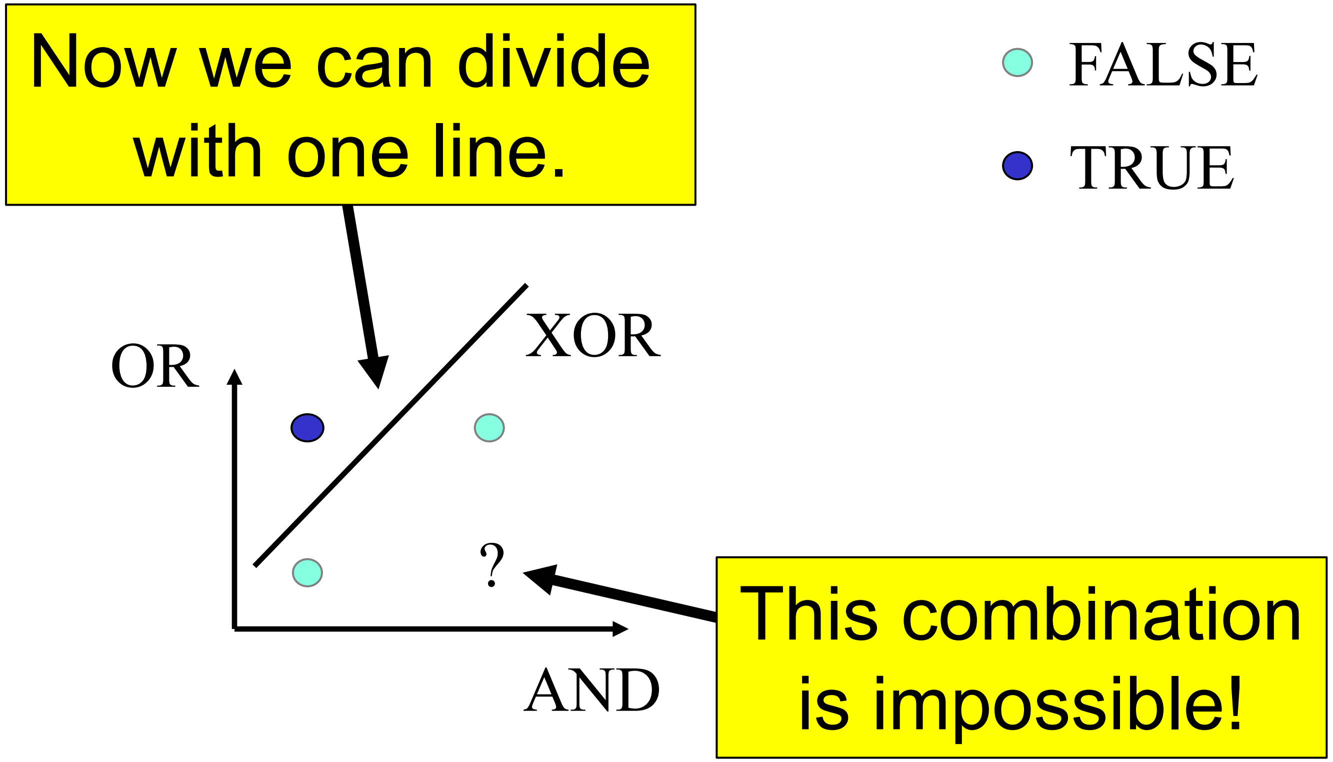
output
bias
weight
input

Can we learn XOR with a Perceptron?

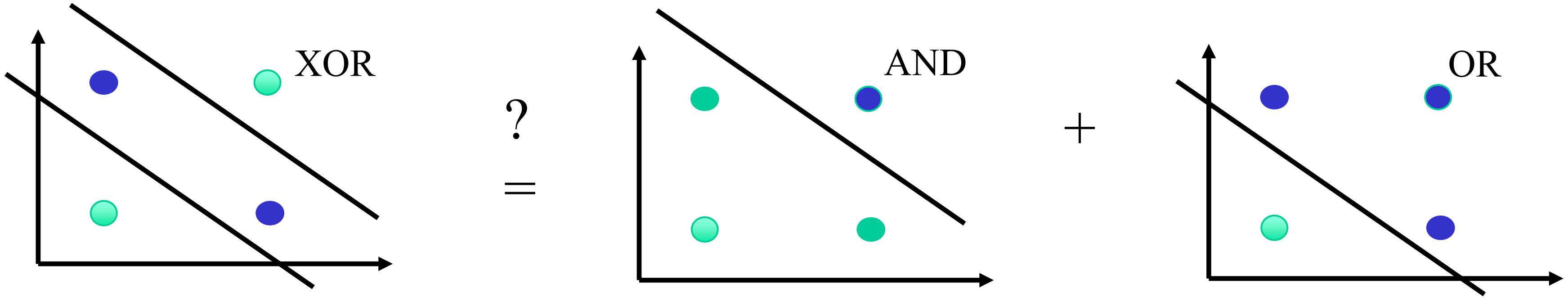


Multiple Layers Solve More Problems

What if input dimensions are AND and OR?



Perceptron



$x[0] + x[1] - 1.5 > 0$ $x[0] + x[1] - 0.5 > 0$

x[1]	x[0]	AND	OR	XOR
0	0	-1 (-1.5 < 0)	-1 (-0.5 < 0)	-1 (-2.0 < 0)
0	1	-1 (-0.5 < 0)	1 (0.5 > 0)	?
1	0	-1 (-0.5 < 0)	1 (0.5 > 0)	?
1	1	1 (0.5 > 0)	1 (1.5 > 0)	1 (2.0 > 0)

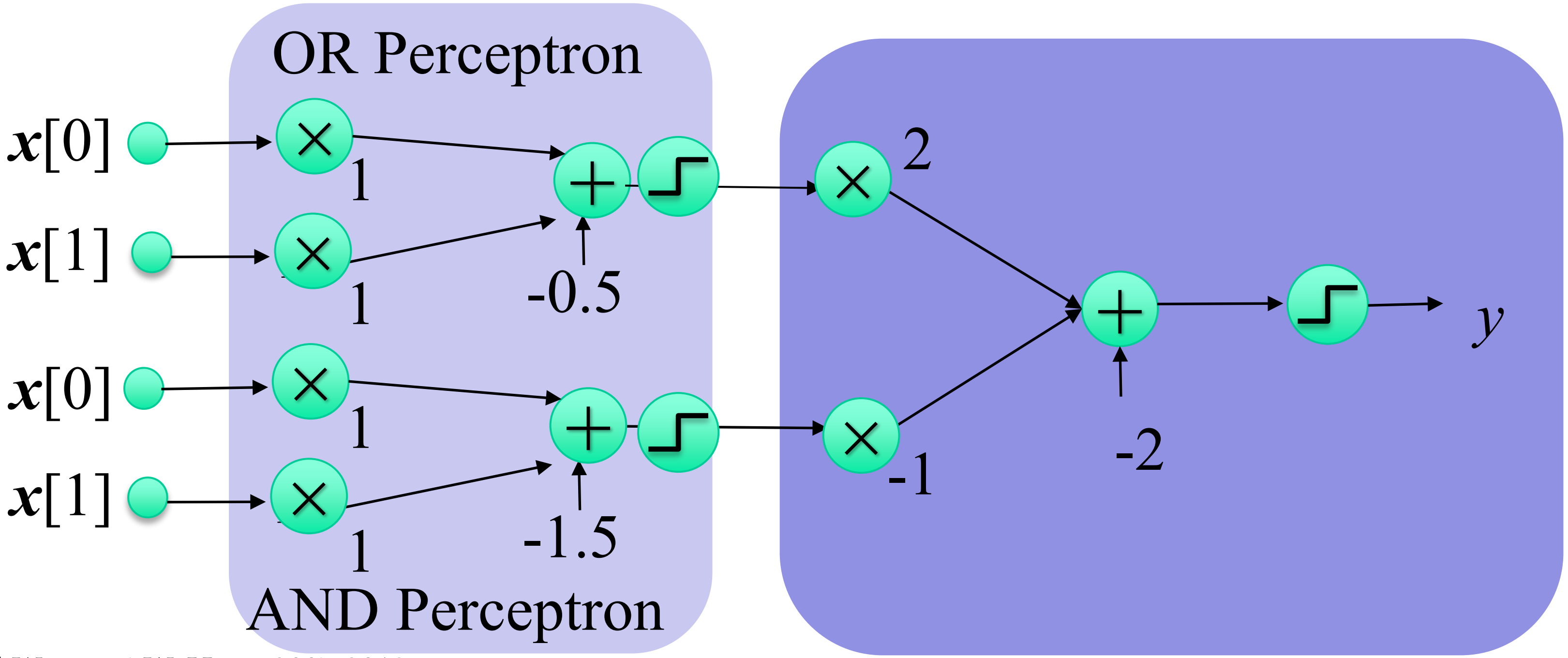
XOR is not a linear combination of AND and OR functions.

x[1]	x[0]	AND	OR	XOR
0	0	-1	-1	-1 (-3 < 0)
0	1	-1	+1	1 (1 > 0)
1	0	-1	+1	1 (1 > 0)
1	1	+1	+1	-1 (-1 < 0)

$$\text{OR} = \text{sign}(\boldsymbol{x}[0] + \boldsymbol{x}[1] - 0.5)$$

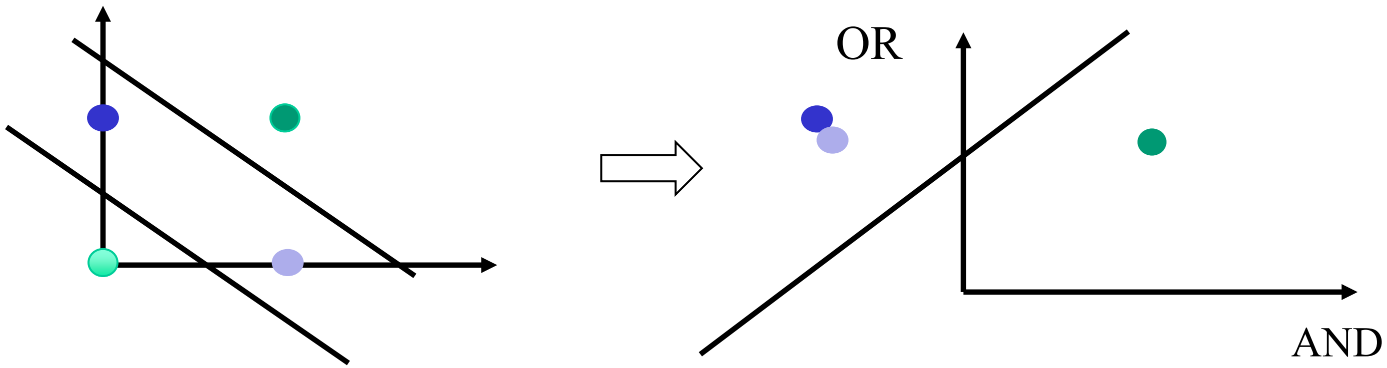
$$\text{AND} = \text{sign}(\boldsymbol{x}[0] + \boldsymbol{x}[1] - 1.5)$$

sign() function adds non-linearity to "reposition" data points for the next layer.



$$\text{XOR} = \text{sign}(2*\text{OR} + -1*\text{AND} - 2)$$

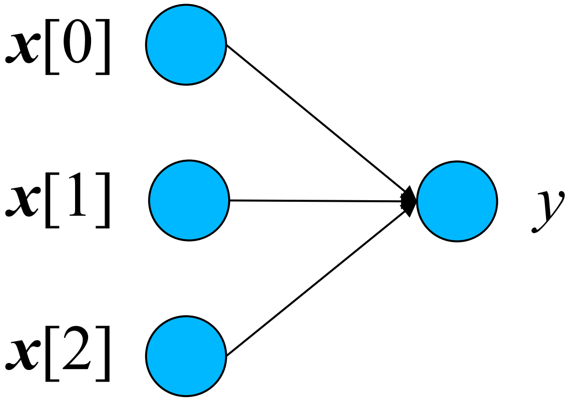
Multi-Layer Perceptron – data repositioning

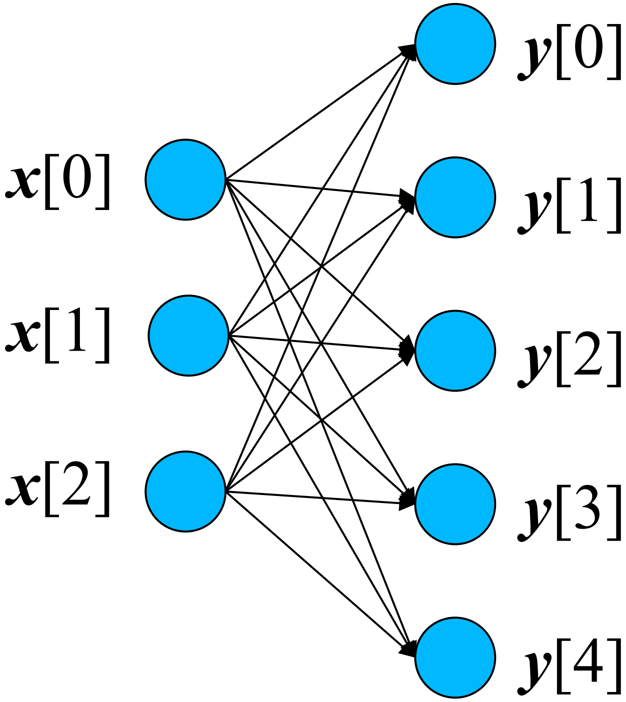


XOR = sign(2*OR + -1*AND - 2)

x[1]	x[0]	AND	OR	XOR
0	0	-1	-1	-1 (-3 < 0)
0	1	-1	+1	1 (1 > 0)
1	0	-1	+1	1 (1 > 0)
1	1	+1	+1	-1 (-1 < 0)

Generalize to Fully-Connected Layer

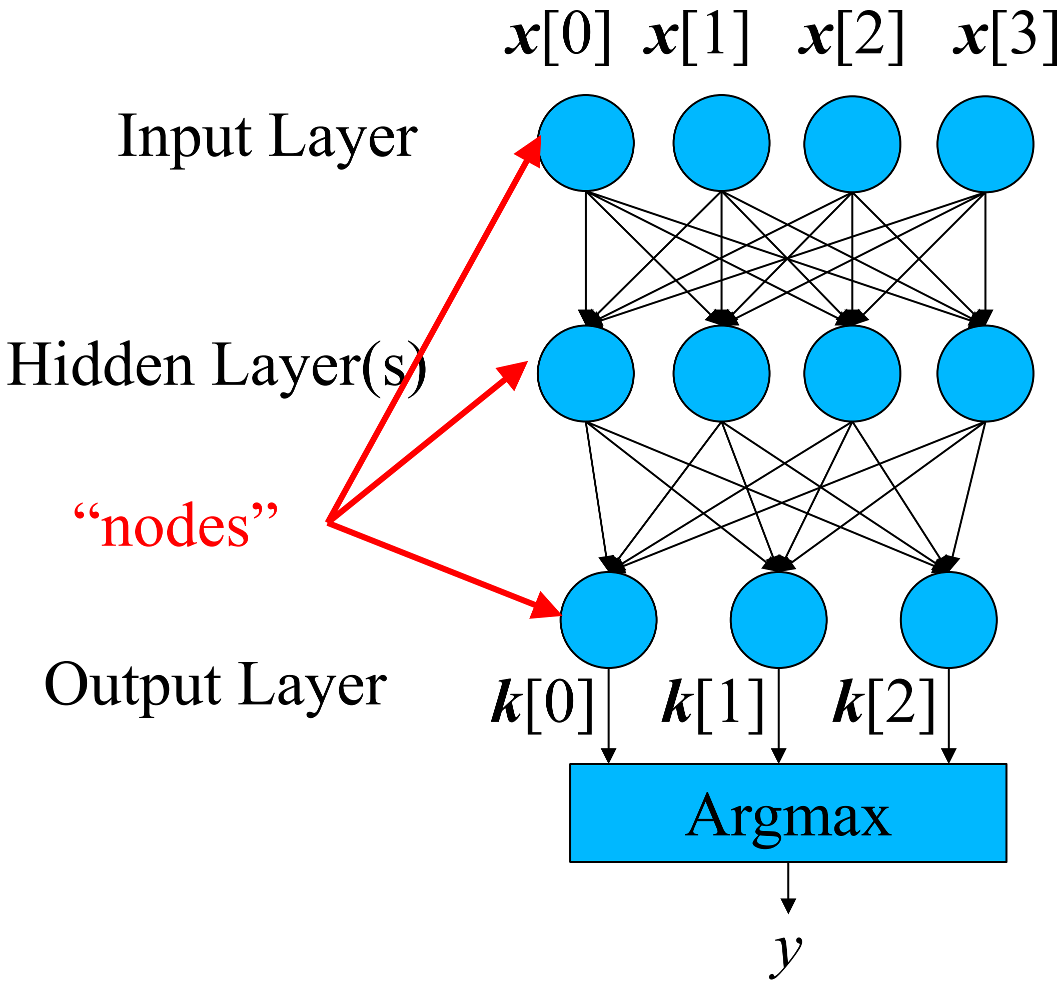




Linear Classifier:
Input vector $\boldsymbol{x} \times$ weight vector $\boldsymbol{w}$ to produce scalar output y

Fully-connected:
Input vector $\boldsymbol{x} \times$ weight <u>matrix</u> $\boldsymbol{w}$ to produce <u>vector</u> output y

Multilayer Terminology



W_k[i, j] : weight between i^{th} input and j^{th} output of the k^{th} layer

W_1 is [4x4], b_1 is [4x1]

W_2 is [4x3], b_2 is [3x1]

Probability that input is class $\boldsymbol{k}$[i]

How to determine the weights?

- Look at observational data to determine the weights?
- Pick some random values?
- Start with something that partially works?

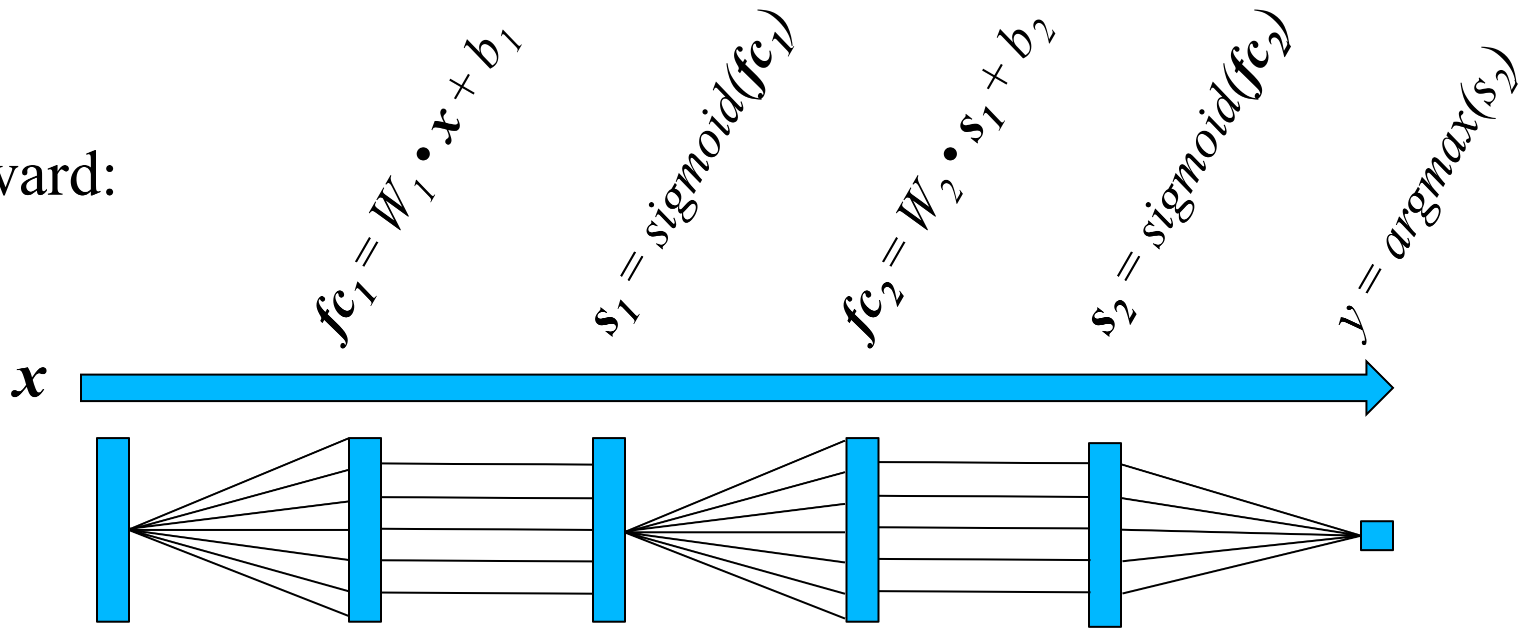
- With enough *labeled* data, we can automatically *encode* the relationship between inputs and outputs.

Forward and Backward Propagation

- Forward (inference)
  - Given parameters Θ and input $\boldsymbol{x}$, produce label y
- Backward (training)
  - Need a way to assess correctness (loss function)
  - Example: $(x - y)^2$
  - Find Θ , such that loss is minimized over all input data

Forward Propagation (Inference)

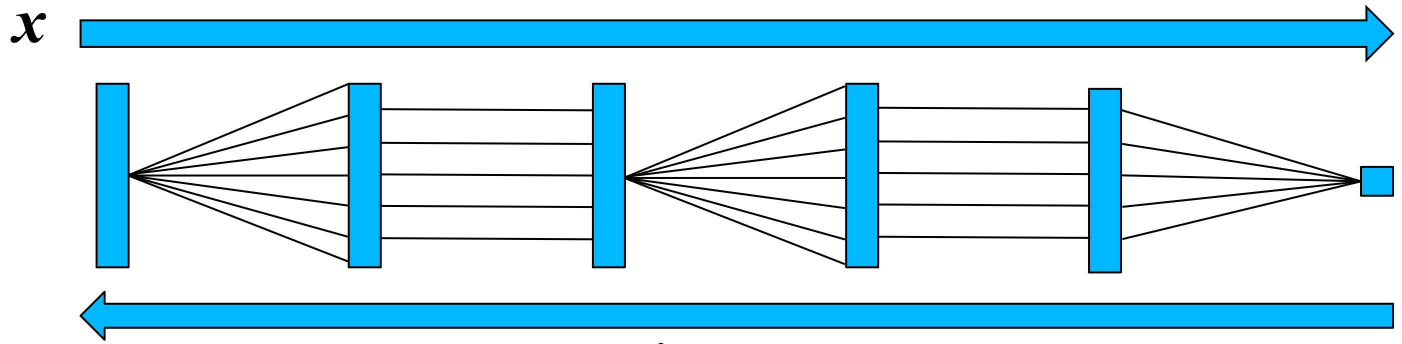
Forward:



Backward Propagation (Training)

Forward:

$fc_1 = W_1 \bullet \boldsymbol{x} + b_1$ $\quad s_1 = sigmoid(fc_1)$ $\quad fc_2 = W_2 \bullet s_1 + b_2$ $\quad s_2 = sigmoid(fc_2)$ $\quad y = argmax(s_2)$



Minimize loss over entire training set:

Need $\frac{dE}{dW}$

Backward:

$\frac{dE}{dfc_1} = \frac{dE}{ds_1}\frac{ds_1}{dfc_1}$ $\quad \frac{dE}{ds_1} = \frac{dE}{dfc_2}\frac{dfc_2}{ds_1}$ $\quad \frac{dE}{dfc_2} = \frac{dE}{ds_2}\frac{ds_2}{dfc_2}$ $\quad \frac{dE}{ds_2} = \frac{dE}{dy}\frac{dy}{ds_2}$ $\quad \frac{dE}{dy}$

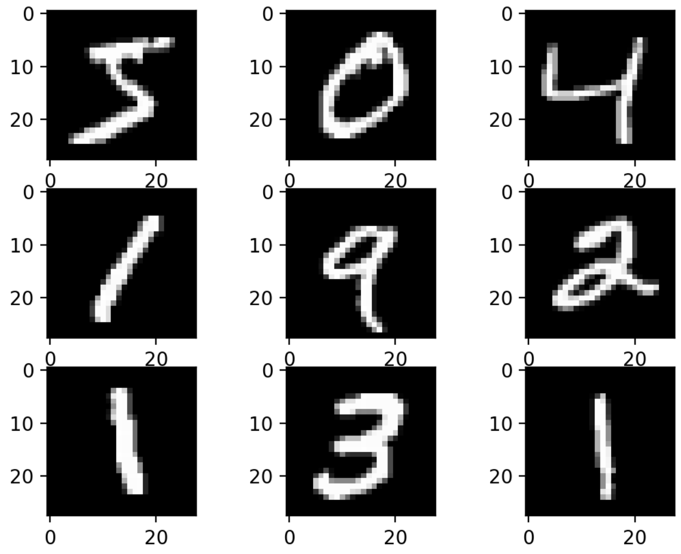
Example: Digit Recognition

Let's consider an example.

- **handwritten digit recognition**:
- given a **28 × 28 grayscale image**,
- produce a **number from 0 to 9**.

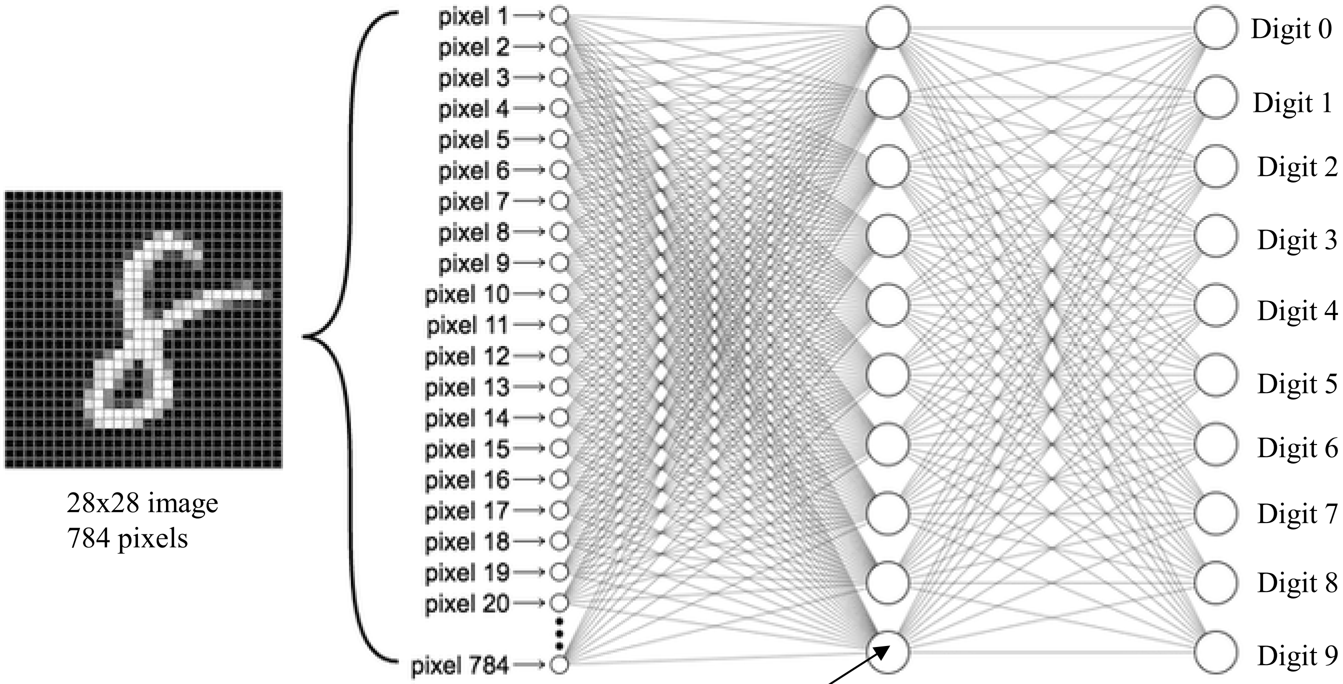
Input dataset

- **60,000** images
- Each labeled by a human with correct answer.



MNIST dataset

MultiLayer Perceptron (MLP) for Digit Recognition



$$n_k^1 = \text{activation}(\boldsymbol{w}_k \boldsymbol{x} + b_k)$$

This network has
- 784 nodes on input layer (L0)
- 10 nodes on hidden layer (L1)
- 10 nodes on output layer (L2)

784*10 weights + 10 biases for L1
10*10 weights + 10 biases for L2

A total of 7,960 parameters

Each node represents a function, based on a linear combination of inputs + bias

Activation function "repositions" output value.

Sigmoid, sign, ReLU are common…

How Do We Determine the Weights?

First layer of perceptron:

- **784** (28^2) inputs, **1024** outputs, **fully connected**
- **[1024 × 784]** weight matrix ***W***
- **[1024 x 1]** bias vector ***b***

Use labeled training data to pick weights.

Idea:

- given enough labeled input data,
- we can **approximate the input-output function**.

Forward and Backward Propagation

Forward (**inference**):

- given input x (for example, an image),
- **use parameters θ** (W and b for each layer)
- **to compute probabilities $k[i]$** (ex: for each digit i).

Backward (**training**):

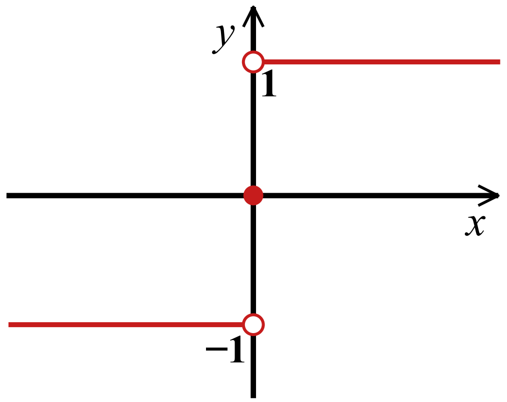
- given input x, parameters θ, and outputs $k[i]$,
- **compute error E** based on target label t,
- then **adjust θ** proportional to E to reduce error.

Neural Functions Impact Training

Recall perceptron function: **$y = \mathrm{sign}\,(W \cdot x + b)$**

To propagate error backwards,

- **use chain rule** from calculus.
- **Smooth functions are useful.**



Sign is not a smooth function.

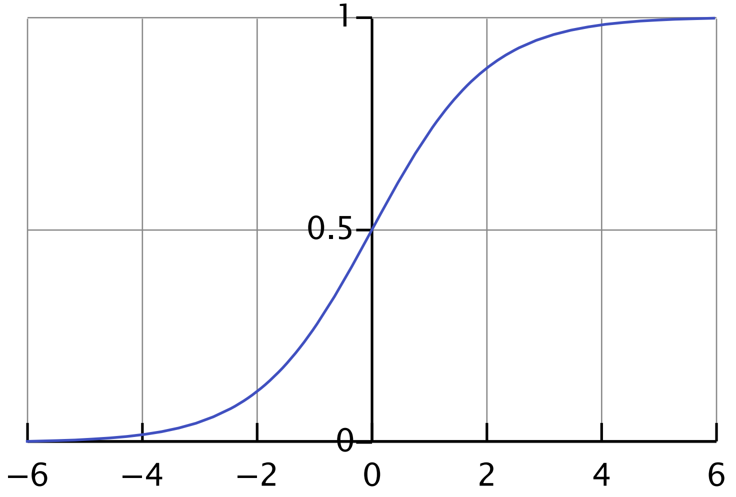
One Choice: Sigmoid/Logistic Function

Until about 2017,

- **sigmoid / logistic function** most popular

$$f(x) = \frac{1}{1+e^{-x}} \qquad (f: \mathbb{R} \rightarrow (0,1))$$

  for replacing sign.



- Once we have *f(x)*, finding *df/dx* is easy:

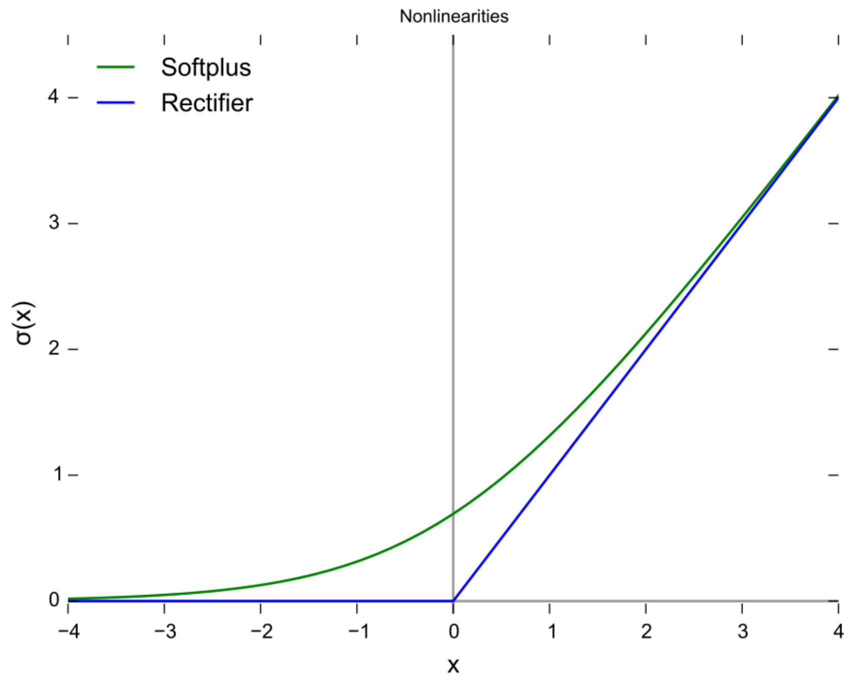
$$\frac{df(x)}{dx} = \frac{e^{-x}}{(1+e^{-x})^2} = f(x)\frac{e^{-x}}{(1+e^{-x})} = f(x)(1 - f(x))$$

(Our example used this function.)

Today's Choice: ReLU

In 2017, most common choice became

- **rectified linear unit / ReLU / ramp function**
  $$\boldsymbol{f(x) = \max(0, x)} \quad (\mathrm{f}: \mathbb{R} \rightarrow \mathbb{R}^{+})$$
  which is much faster (no exponent required).
- A smooth approximation is **softplus/SmoothReLU**
  $$\boldsymbol{f(x) = \ln(1 + e^{x})} \quad (\mathrm{f}: \mathbb{R} \rightarrow \mathbb{R}^{+})$$
  which is the integral of the logistic function.
- Lots of variations exist. See for example Wikipedia for an overview and discussion of tradeoffs.



Use Softmax to Produce Probabilities

How can sigmoid / ReLU produce probabilities?

They can't.

- Instead, given output vector **Z = (z[0], …, z[C-1])***,
- we produce a second vector **K = (k[0], …, k[C-1])**
- using the **softmax function**

$$\boldsymbol{k[i]} = \frac{\boldsymbol{e^{z[i]}}}{\boldsymbol{\sum_{j=0}^{C-1} e^{z[j]}}}$$

Notice that **the k[i] sum to 1.**

*Remember that we classify into one of C categories.

Softmax Derivatives Needed to Train

We also need the **derivatives of softmax**,

$$\frac{dk[i]}{dz[m]} = k[i](\delta_{i,m} - k[m]),$$

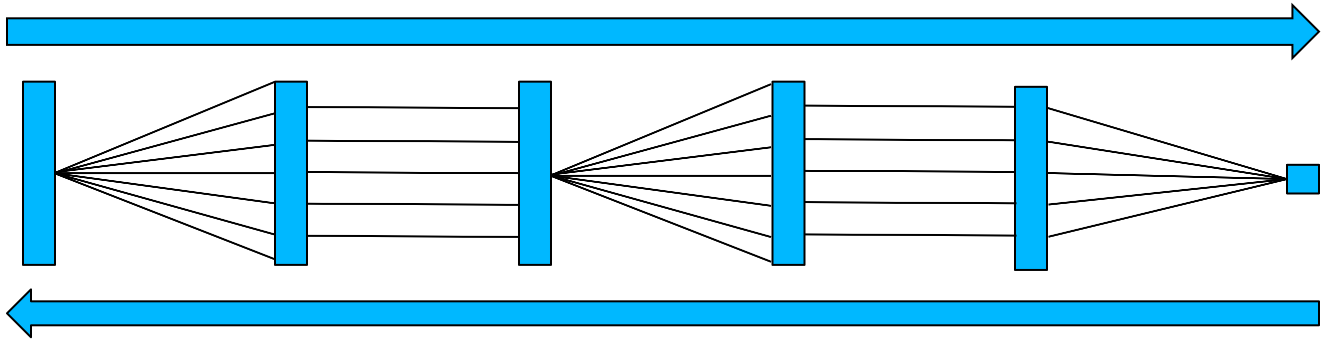
where $\delta_{i,m}$ is the Kronecker delta (1 if i = m, and 0 otherwise).

Forward and Backward Propagation

Forward:

$fc_1 = W_1 \bullet x + b_1$ $\quad y = sigmoid(fc_1)$ $\quad fc_2 = W_2 \bullet y + b_2$ $\quad z = sigmoid(fc_2)$ $\quad k = softmax(z)$

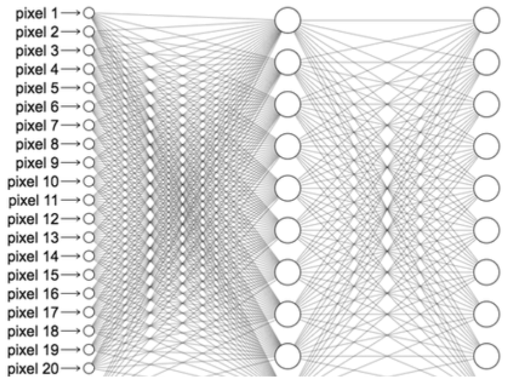
x



dE/dk

Backward:

$$\frac{dE}{dfc_1} = \frac{dE}{dy}\frac{dy}{dfc_1} \qquad \frac{dE}{dy} = \frac{dE}{dfc_2}\frac{dfc_2}{dy} \qquad \frac{dE}{dfc_2} = \frac{dE}{dz}\frac{dz}{dfc_2} \qquad \frac{dE}{dz} = \frac{dE}{dk}\frac{dk}{dz}$$



ANY MORE QUESTIONS?