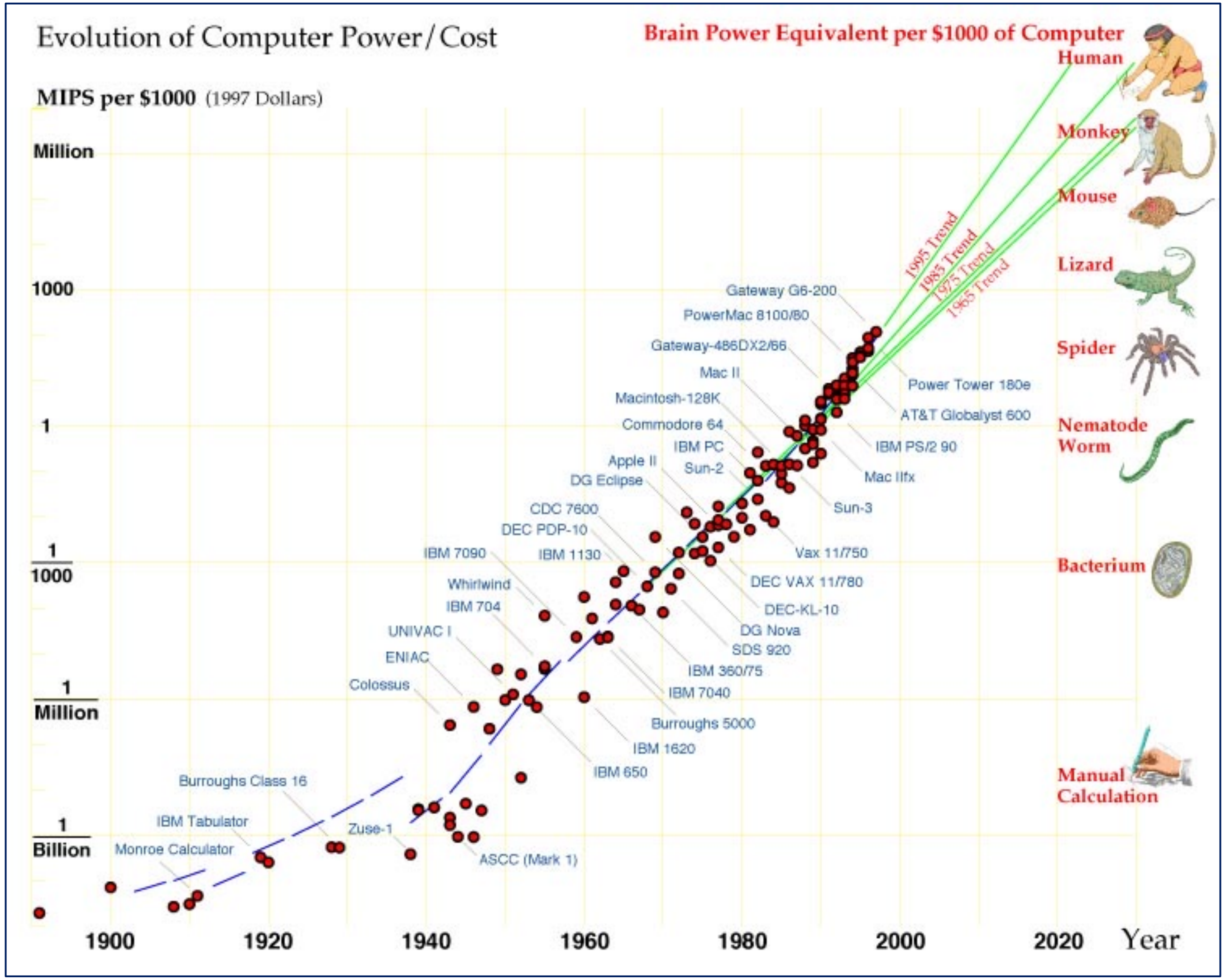
ECE408 / CS483 / CSE 408
Applied Parallel Programming

Introduction to Machine Learning

© D. Kirk, W W. Hwu, and Others

Course Reminders

- Lab 3 is due on Friday
- Midterm 1 is on Tuesday, October 10th
- Project Milestone 1: Baseline CPU implementation is due Friday October 13th
  - Project details to be posted next week on Canvas



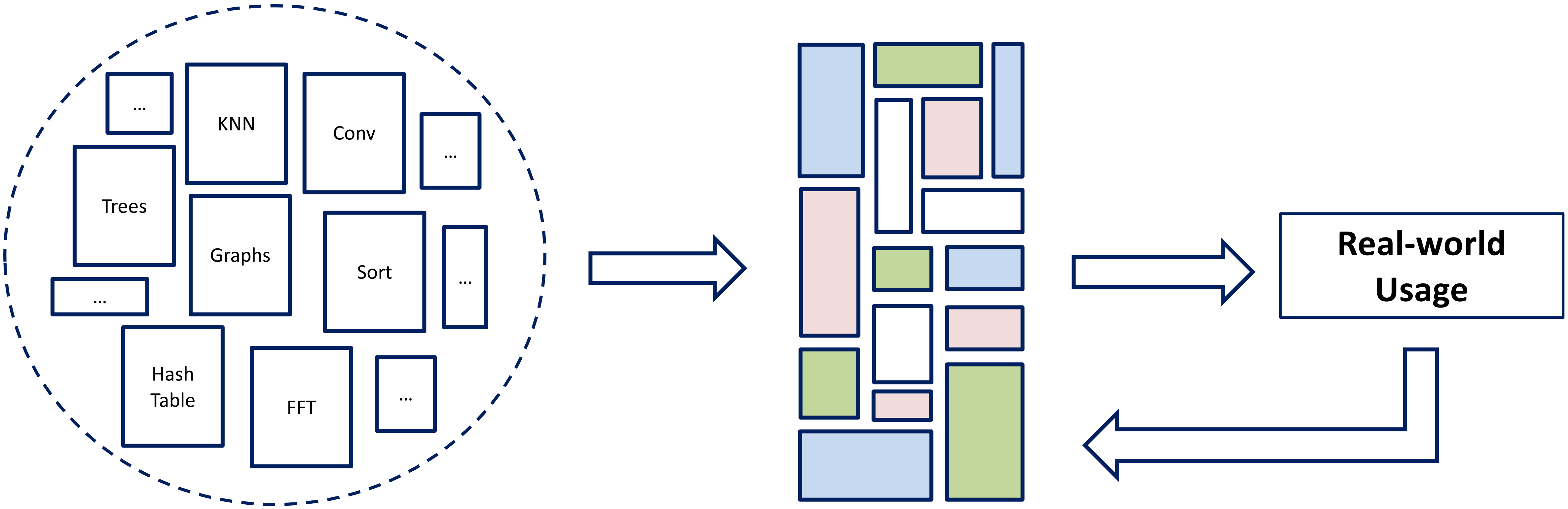
Hans Moravec, 1997

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.

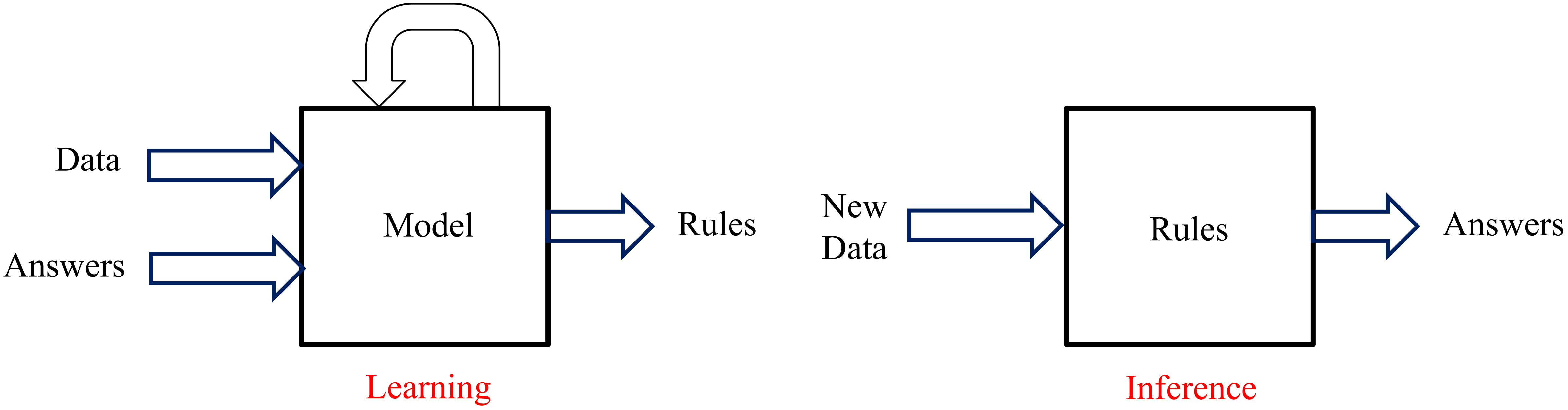
Solving Hard Problems with Software (the established way)

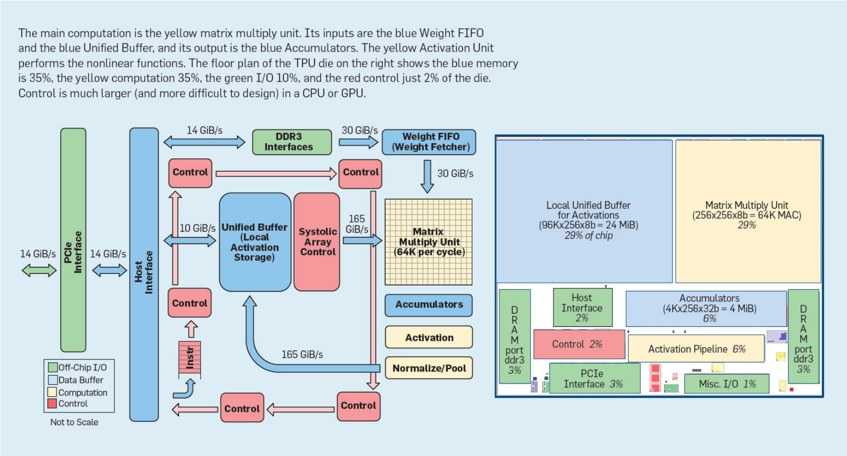


Algorithmic Building Blocks   Software   Refinement

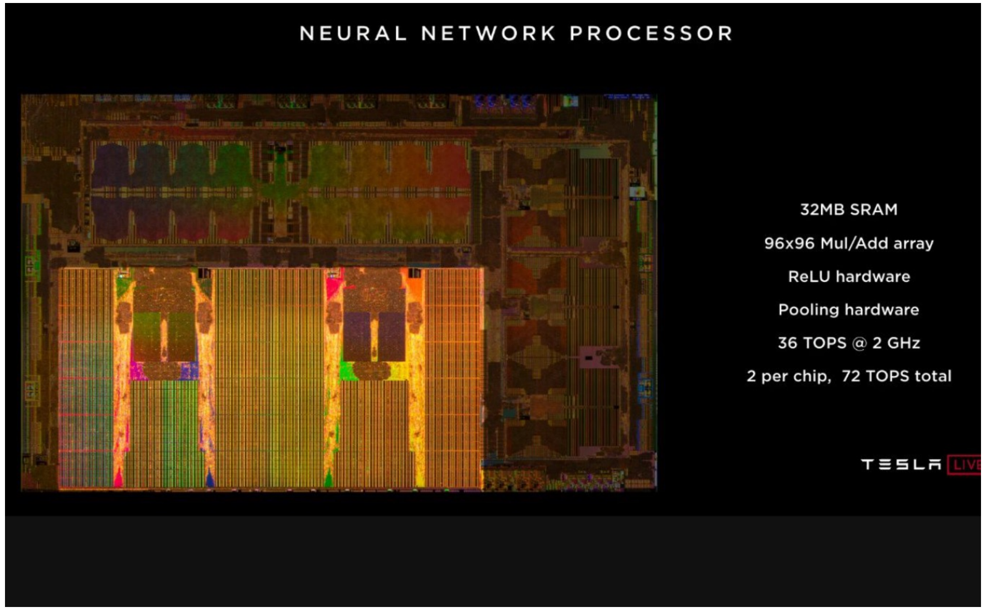
Machine Learning

- Building applications whose logic is not fully understood.
  - Use data to learn what the logic should be

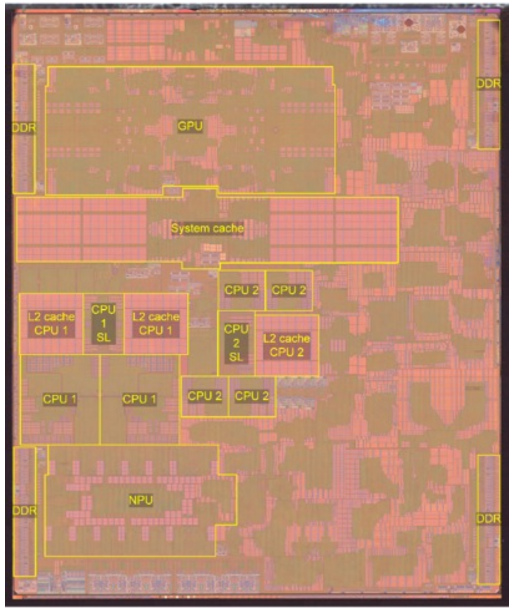




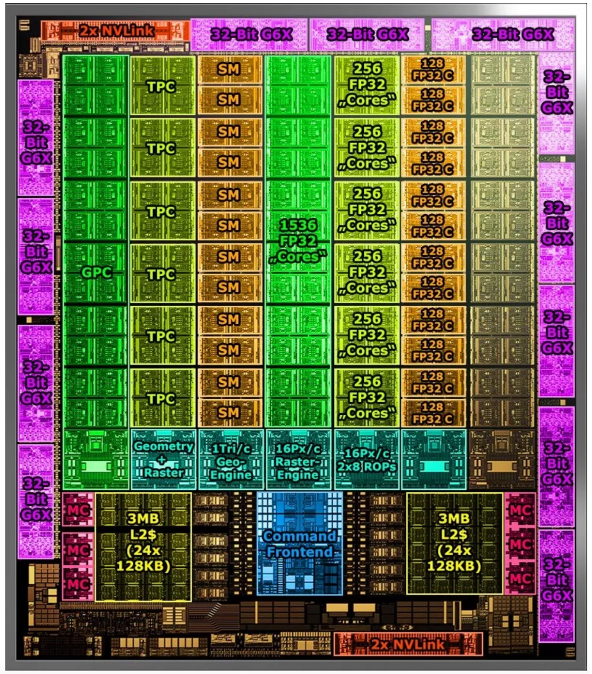
Google Tensor Processing Unit



Tesla ASIC

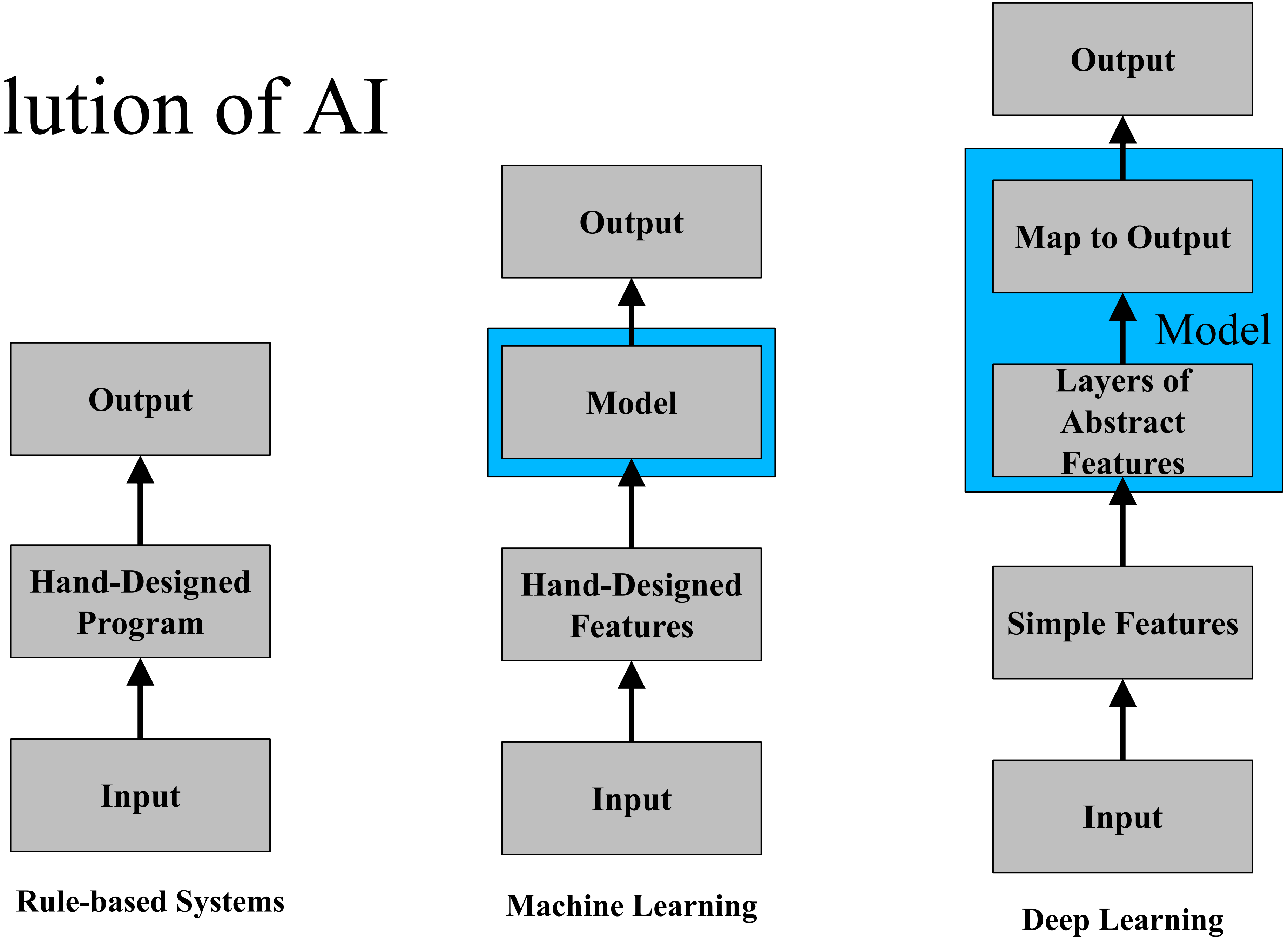


Apple A14



Nvidia Ampere

Evolution of AI



Machine Learning Tasks (1)

- Classification
  - Which of *k* categories an input belongs to
  - Ex: object recognition
- Regression
  - Predict a numerical value given some input
  - Ex: predict tomorrow's temperature
- Transcription
  - Unstructured data into textual form
  - Ex: optical character recognition

Machine Learning Tasks (2)

- 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 a tree of grammatical structure
- Others
  - Forecasting, Anomaly detection, recommendation, synthesis, sampling, imputation, denoising, density estimation

Why Deep Learning Now?

- **Deep Learning:** Advances in DL concepts, frameworks, toolkits have made DL very accessible to all

- **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, SmartDevices, Security, Societal Comfort with Tech, Health Care, Digital Agriculture, Digital Manufacturing

Machine Learning Building Blocks

- **Naïve Bayes**
  - Independent features combined using Bayesian prediction model
- **Perceptrons**
  - "Bio/Neural" inspired method
- **Linear / Logistic Regression**
  - Feature contribution learned to perform prediction / classification
- **Support Vector Machines**
  - Large margin method
- **Decision Trees / Random Forests**
  - Space-splitting methods
- **K-Means Clustering**
  - Unsupervised technique for data analysis

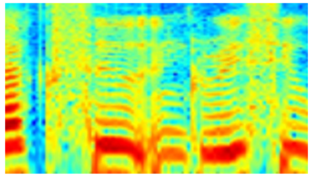
Different Features for Different Tasks

Image

Vision Features

Detection

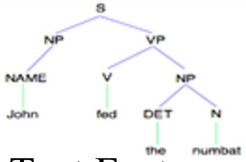
Audio



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?
  - Black/dark rubber around metal rim?
- But what about?
  - Occlusion, perspective, shadows, different colored tires, …
- Need to treat variations in a consistent and comprehensive manner

Classification

- Formally: a function that maps an input to *k* categories

  $f : \mathrm{R}^n \rightarrow \{1, \ldots, k\}$,

- Our formulation: a function *f* parameterized by Θ that maps input vector $\boldsymbol{x}$ to numeric code y

  $y = f(\boldsymbol{x}, \Theta)$

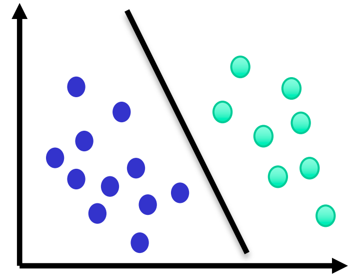
- Θ encapsulates the parameters in our network

Linear Classifier (Perceptron)

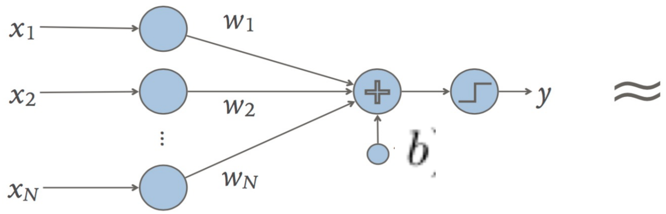
- Our formulation: $y = f(\boldsymbol{x}, \Theta)$

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

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



The perceptron



The neuron

- Dot product + Scalar addition:

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

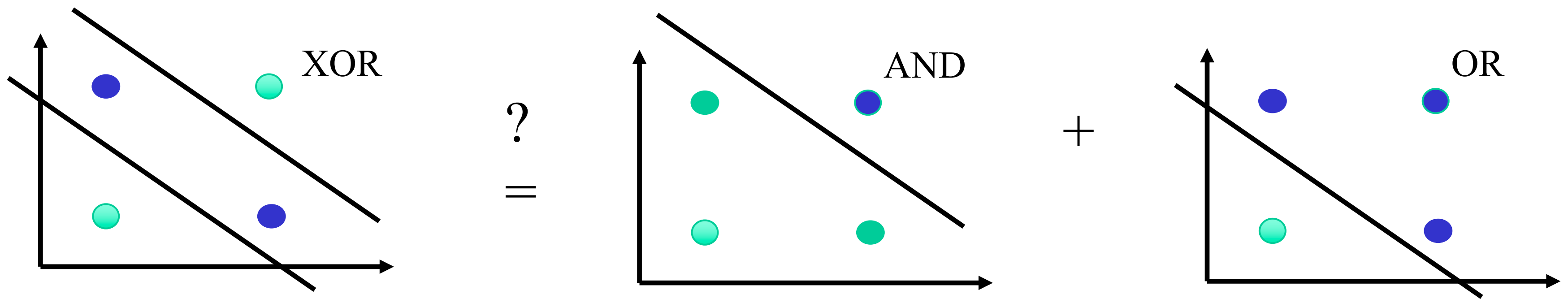
output

bias

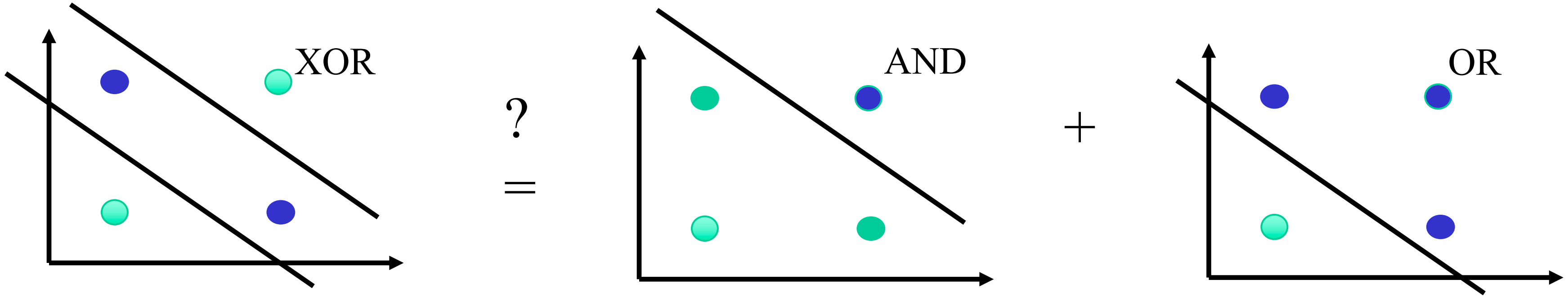
weight

input

Can we learn XOR with a Perceptron?



Perceptron



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

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)

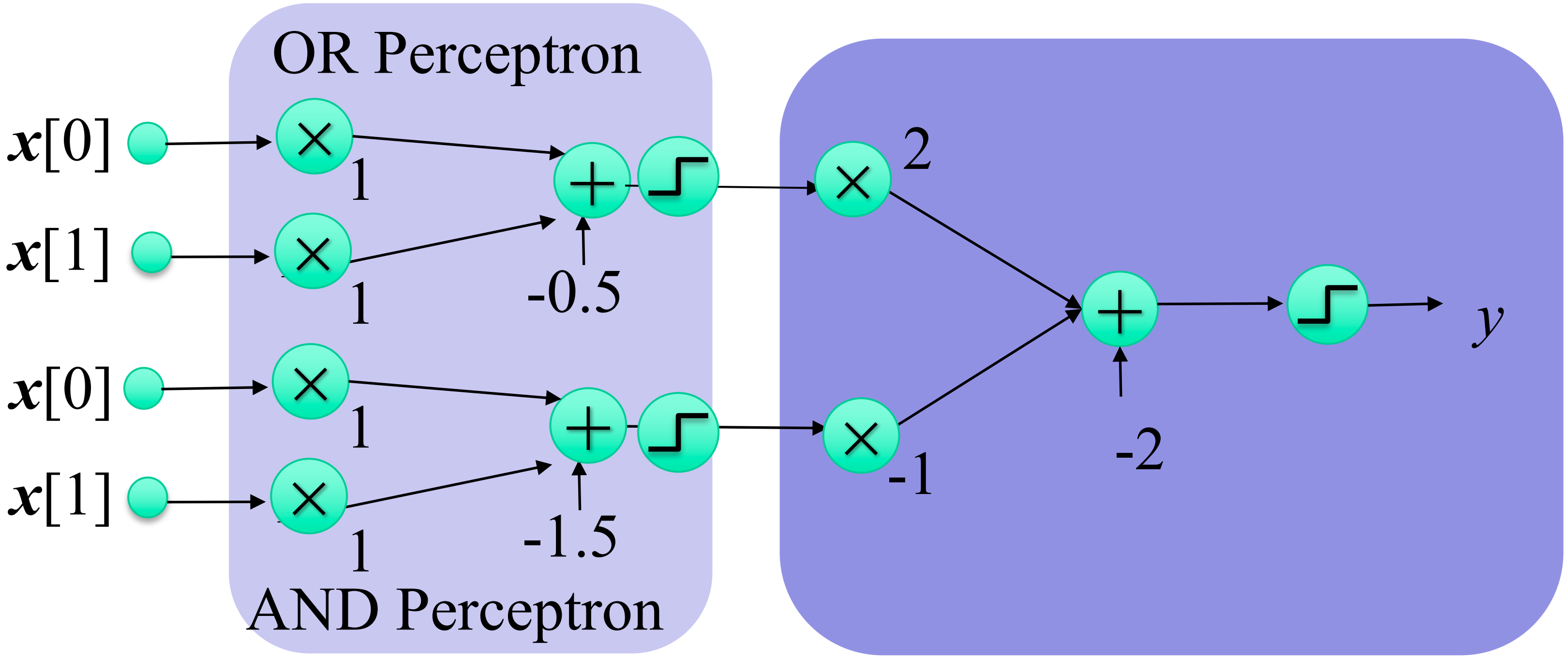
XOR is not a linear combination of AND and OR functions.

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)

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

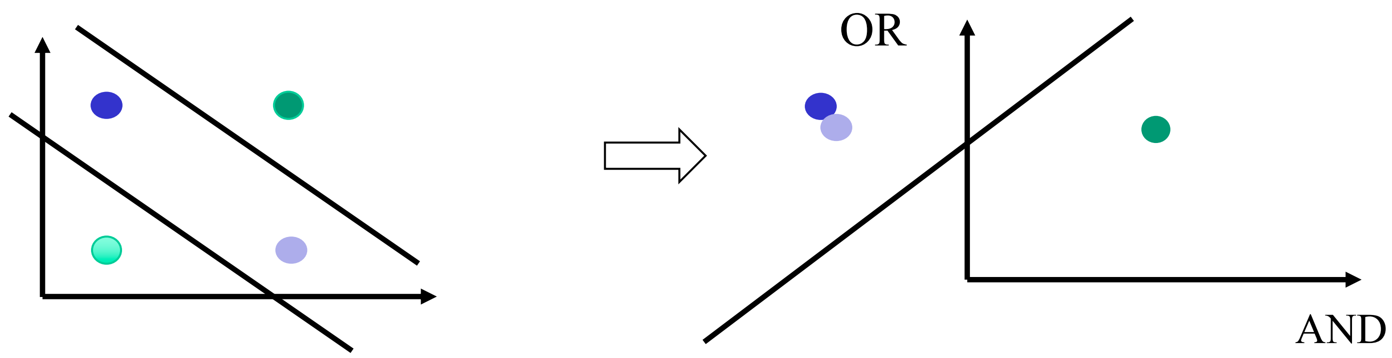
$$\text{AND} = \text{sign}(\boldsymbol{x}[0] + \boldsymbol{x}[1] - 1.5)$$

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



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

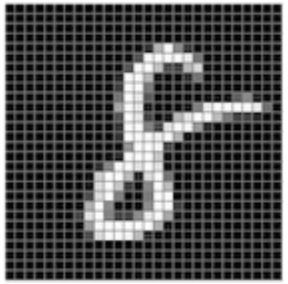
Multi-Layer Perceptron



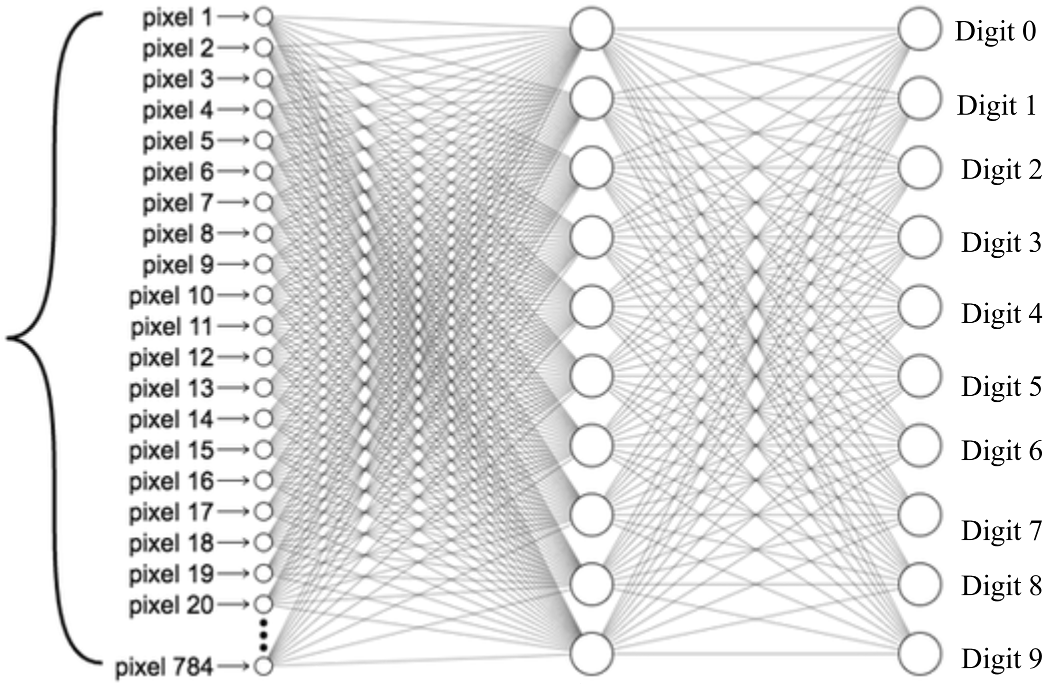
XOR = sign(2*OR + -1*AND - 2)

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)

MultiLayer Perceptron (MLP) for Digit Recognition



28x28 image
784 pixels



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

This network would 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

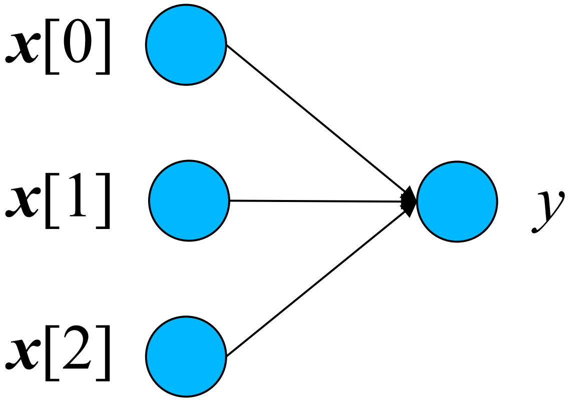
A total of 7,960 parameters

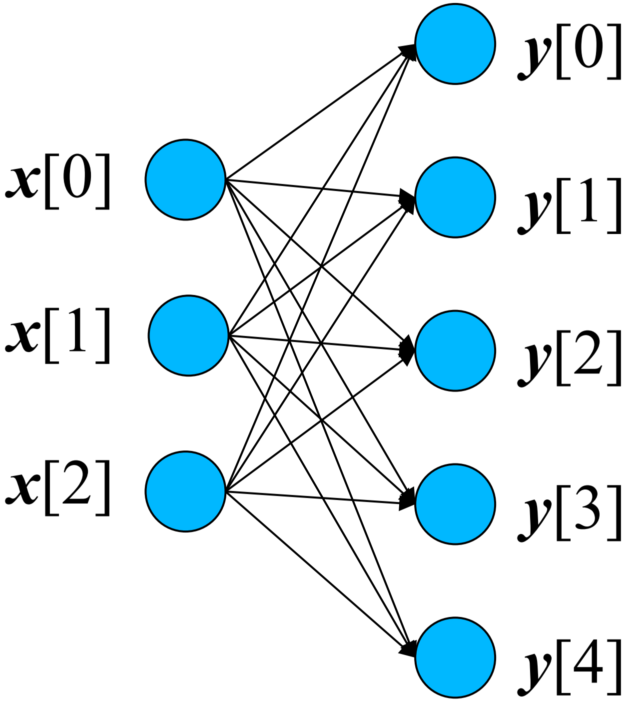
Each node represents a function, based on a linear combination of inputs + bias

Activation function “repositions” output value.

Sigmoid, sign, ReLU are common…

Generalize to Fully-Connected Layer

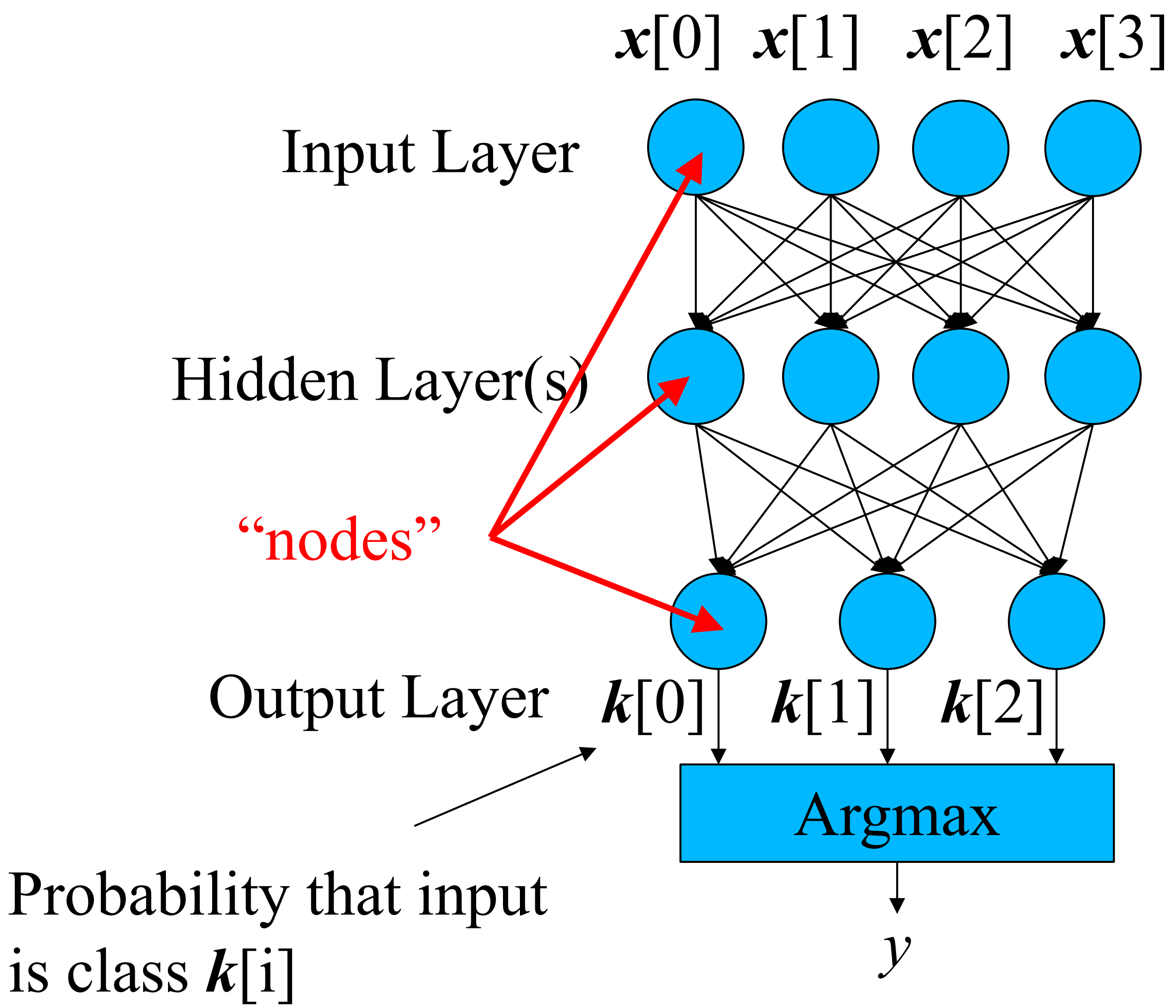




Linear Classifier:
Input vector $\boldsymbol{x} \times$ weight vector $\boldsymbol{w}$ to produce scalar output y

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

Multilayer Terminology



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

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

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

How Do We Determine the Weights?

First layer of perceptrons

- **784** (28^2) inputs, **10** outputs, **fully connected**
- **[10 × 784]** weight matrix ***W***
- **[10 x 1]** bias vector ***b***

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

Forward and Backward Propagation

Forward (**inference**):

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

Backward (**training**) [Rumelhart, Hinton, Williams 1986]:

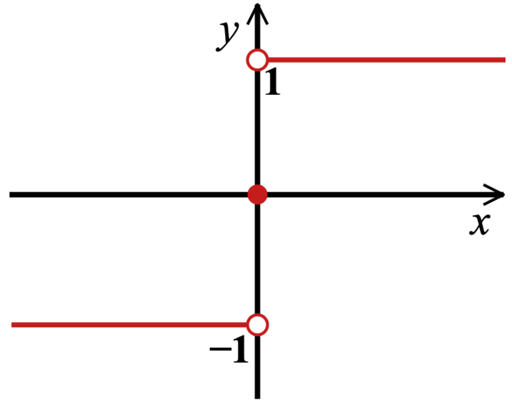
- given input $\boldsymbol{x}$, parameters $\boldsymbol{\theta}$, and outputs ***k[i]***,
- **compute error $\boldsymbol{E}$** based on target label $\boldsymbol{t}$,
- then **adjust $\boldsymbol{\theta}$** proportional to $\boldsymbol{E}$ to reduce error.

Neural Functions Impact Training

Recall perceptron function: **y = sign (W·x + b)**

To propagate error backwards,

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



Sign is not a smooth function, and has regions of 0 slope.

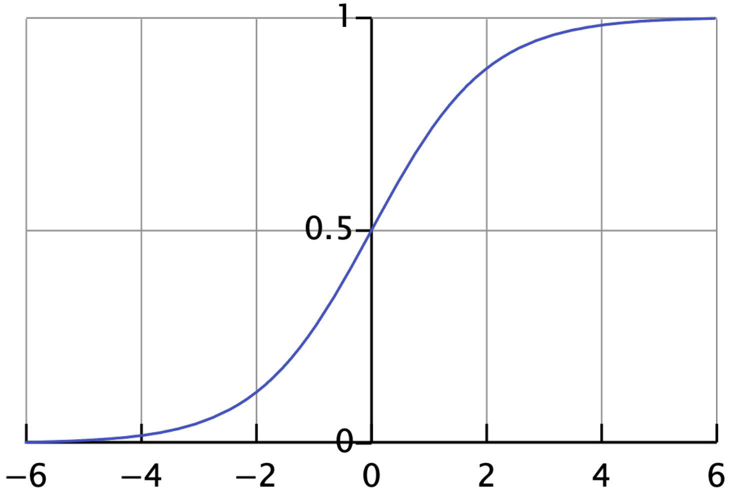
One Choice: Sigmoid/Logistic Function

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**

  $$\boldsymbol{f(x) = \max(0, x)} \quad (\mathrm{f}: \mathbb{R} \to \mathbb{R}^{+})$$

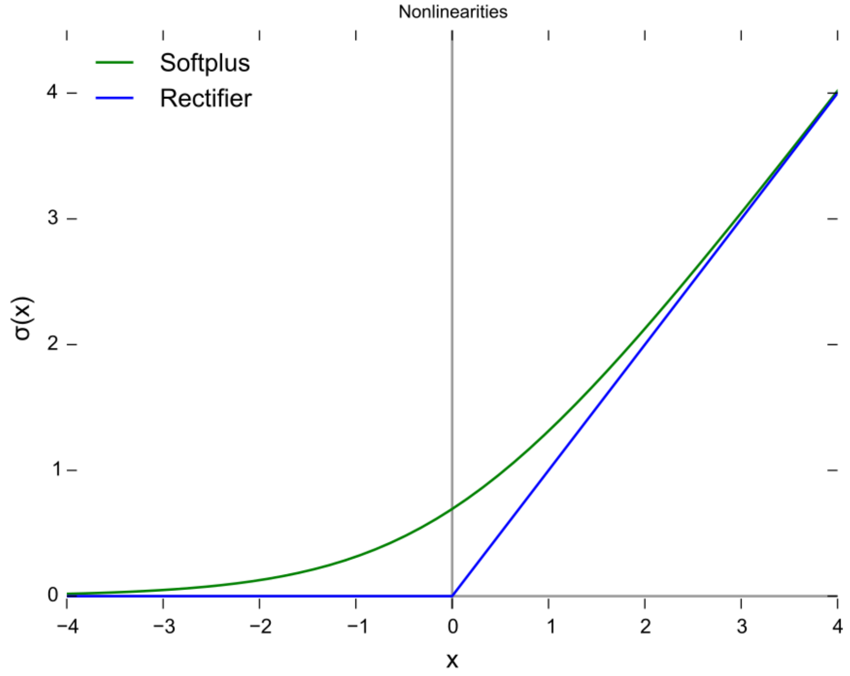
  which is much faster (no exponent required).

- A smooth approximation is **softplus/SmoothReLU**

  $$\boldsymbol{f(x) = \ln(1 + e^{x})} \quad (\mathrm{f}: \mathbb{R} \to \mathbb{R}^{+})$$

  which is the integral of the logistic function.

- Lots of variations exist. See 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**

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

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

*Remember that we classify into one of C categories.

Choosing a Loss (or Error) Function

Many error functions are possible.

For example, **given label *T*** (digit ***T***),

- ***E = 1 – k[T]***, the **probability of not classifying as *t***.

Alternatively, since our categories are numeric,
we can **penalize quadratically**:

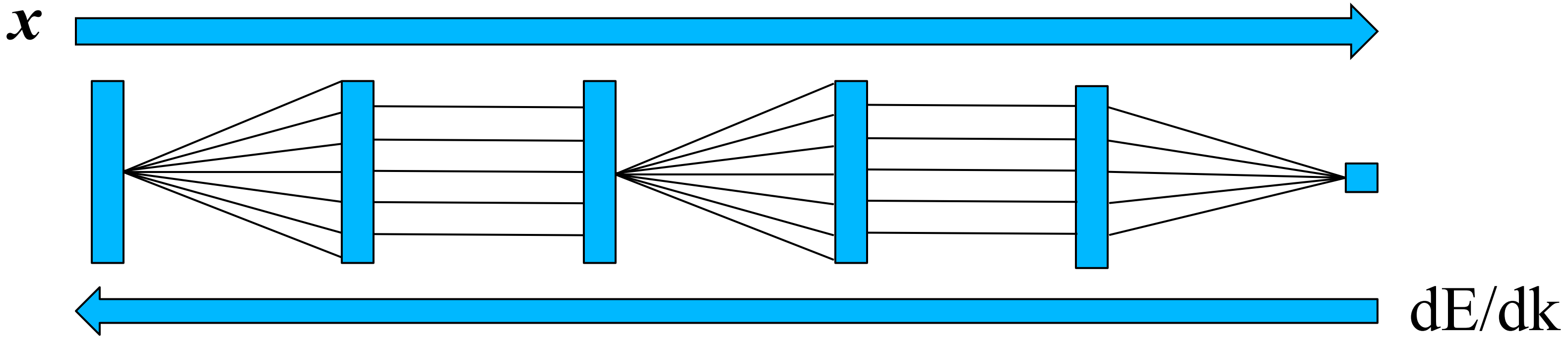
$$E = \sum_{j=0}^{C-1} k[j](j - T)^2$$

Let's **go with the latter**.

Forward and Backward Propagation

Forward:

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



Backward:

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