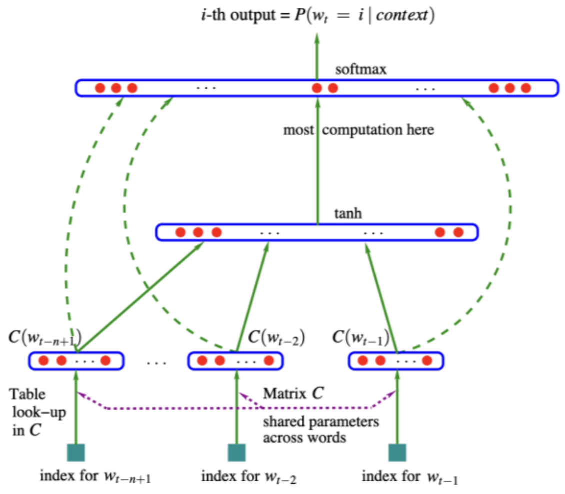
Neural computation models

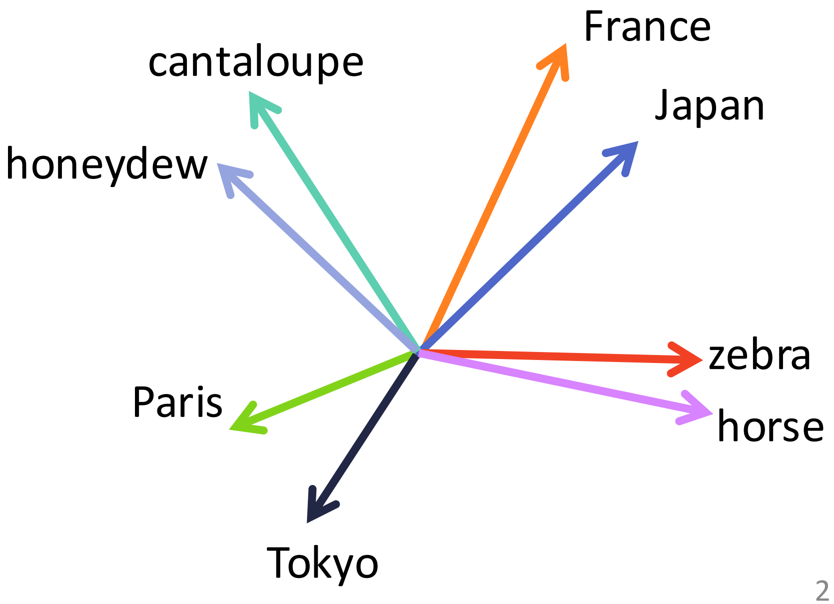
Daniel Hsu

COMS 6998-7 Spring 2025

Recap: neural language models

- Shannon 1948, 1951: Compute
$$P(x_t | x_1, \dots, x_{t-1})$$
- Bengio et al, 2003 (and others): Maybe it can be done using a neural network
  - Maybe just the n-gram approximation, i.e., only a function of $x_{t-n+1}, \dots, x_t$
- Neural language models:
  - Embed discrete tokens into vector space
  - Consider function of vectors in this space
  - There are interesting computations to be done in this vector space!





Computation

Computational task

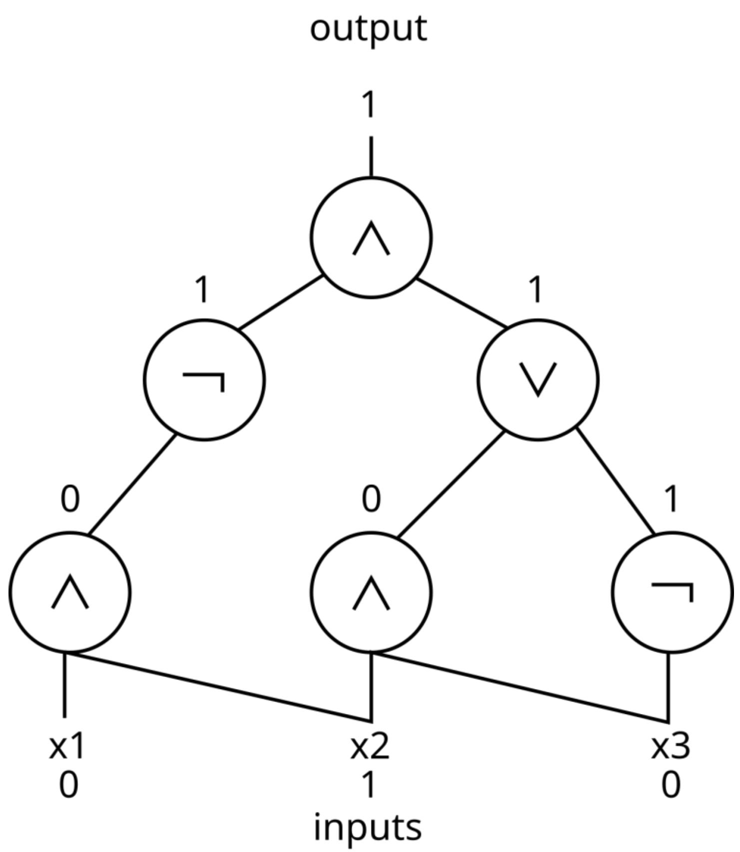
- Evaluate functions
- Execute algorithms
- …

Models of computation

- Circuits
- Automata (i.e., "machines")
  - E.g., finite state automata, push-down automata, Turing machines
- …

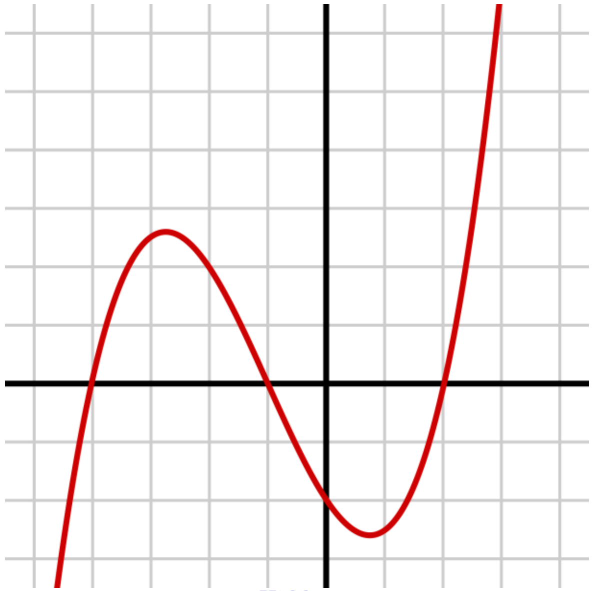
Circuit model

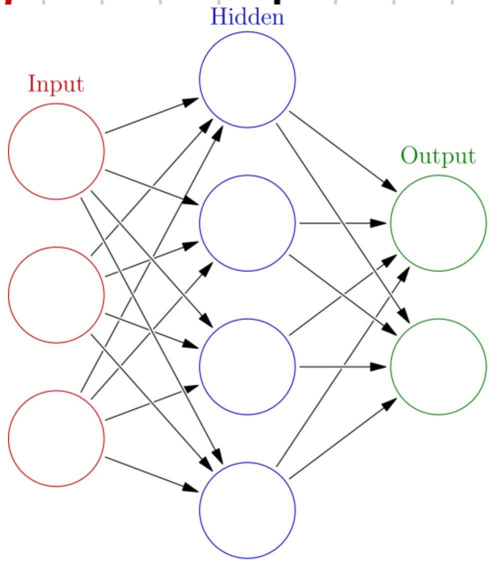
- Boolean functions $f: \{0,1\}^n \to \{0,1\}$
  - There are 2^{2^n} such functions
- How many Boolean circuits (using AND, OR, NOT gates) of size m?
  $\leq (3 \cdot 2^{m+n})^m$
- So cannot compute all Boolean functions using $\mathrm{poly}(n)$-size circuits
- <u>Circuit complexity</u>: Which functions have small circuits?



Function approximation

- Consider arbitrary continuous $f: [0,1]^n \to [0,1]$
- How can you (approximately) compute it?
  - Weierstrass approximation theorem: for any $\epsilon > 0$, there is a (multivariate) polynomial p such that
$$\max_{x \in [0,1]^n} |f(x) - p(x)| \leq \epsilon$$
    - Caveat: degree of p might be large!
  - Cybenko; Hornik-Stinchcombe-White; Barron; …: Can also achieve this approximation using a neural network with a single layer of "hidden units"
    - Caveat: number of "hidden units" might be large!
    - Saving grace?: for some "nice" functions, number of "hidden units" can be relatively small





Decidability of languages

- A subset of strings $L \subseteq \{0,1\}^*$ is called a *language*
  - E.g., $L =$ valid encodings of 3-CNF formulas that are satisfiable
- A machine M *decides* the language L if, on input $x \in \{0,1\}^*$, the machine M halts and returns $1_{\{x \in L\}}$
- Church-Turing thesis: the following are equivalent
  - There is an algorithm for deciding L
  - There is a *Turing machine* that decides L
- Many other types of "machines" are equivalent to Turing machines in computational power (at least up to polynomial factors in "runtime")
  - E.g., multi-tape Turing machines, Random Access Machines
  - Siegelmann and Sontag: also Recurrent Neural Networks (over $\mathbb{Q}$)

Things to think about

- How do neural nets perform computation?
- Does it "simulate" a more "classical" type of machine?
  - How are the components of the classical machine being simulated?
  - Why might a neural net be preferable to the classical machine?