1 Introduction

The goal of this document is to better understand what types of functions exist, with the intuition that *functions are knowledge*, and so a taxonomy of functions is a taxonomy of the ways to represent knowledge. Descriptive categorization is helpful for:

- Finding gaps and continuations: what's missing, and why? What classes of functions can *not* be easily represented by other classes?
- Improving the capacity and generalization properties of models - to better match the 'true' function that generated the data.[1]
- Enhancing the learning & optimization algorithms that build them, e.g. for learning the right function with less data or fewer interactions. Function inverses, which map data back to generating funcions, are *themselves* functions / algorithms that need to be derived or learned[2].

[1] See previous post's discussion of equivalence classes

[2] DL, of course, derives them with the chain rule, but this is only one means of propagating information from data to a compressed model. Inverses might well range from LUTs to heuristics to algorithms - they are in general not in the same class as the forward function!

Most of the functions levels below are, at the limit, interchangeable - you can approximate any input-output mapping (or algorithm) with a giant look-up table. Functional equivalence means that extensional properties are the same, while intensional properties may differ. LUTs can be tremendously *inefficient* in terms of learning and storage; they have effectively zero compression, hence ~ 0 generalization. As you get higher and higher levels of compression you *generally* need more sophisticated algorithms for inducing the function from data, as not only may one element of the function affect many or all outputs, but frequently relationships between functional elements (pairwise and higher-order) affect the outputs. Indeed, a key element of compression is to form representations from pairwise and higher-order interactions; this is critical for compositionality. Another key element is re-use, which enables the few-to-many expansions during function application.

At the limit of this, I argue elsewhere that the induction algorithm must be a strange loop. This is a loop (as in iteration or recurrence) in which state variables index over abstraction itself, e.g. whether functions are treated as the result of computation (data) or the cause of computation (code). There are a wide variety of strange loops, from the trivial oscillators of like "this statement is false" to the knowledge and abstraction generating strange loops we embody. Open-ended induction / compression requires repeated, recurrent abstraction-generation to circumvent exponential hypothesis space.

$\sim$

The remainder of this document was created with the help Gemini and Claude; I tried to edit down into reasonable coherence. About half the examples (on folding and ordering computation, XOR problem, others) are my own, the rest are filled in by AI. I think more work can be done wrt describing the learning and generalization properties of each class - properties which are uncomputable and hence only heuristic / approximate - but that's a matter of continuing effort.

A Taxonomy of Function Representations and Computational Expressivity

Abstract

A function is a mathematical mapping $f : X \to Y$.[3] The properties of this mapping can be categorized by the mechanism of representation, the geometry of the resulting hypothesis class, and the computational architecture used to calculate it. The following taxonomy attempts to segment functions by their structural or compressional efficiency, progressing from unstructured memorization to the theoretical limits of computation. In the real world, functions are in one or several classes, or are compositions of different classes, leading to composite attributes, costs, and complexity.

[3] Elements of X and Y could be scalars or vectors; the non-function case that they are sets is not covered here, though it's important for random processes...

Level 0: Explicit Tabular Representations (Pure Memorization)

The domain X and codomain Y are treated as unstructured, discrete & unordered sets.[4]

[4] Technically: "Equipped solely with the discrete topology."

- **Concept:** The function is represented explicitly as an exhaustive set of tuples (x, y).
- **Topological Mechanism:** The concept of "nearness" does not exist. Because the discrete topology isolates every point, $f(x)$ has zero mutual information to $f(x')$, $\forall x, x' : x' \neq x$.
- **Mathematical Grounding (Information Theory):** The information required to specify the mapping is the Shannon entropy of the full Cartesian product. For a discrete domain of size $|X| = 2^{b_{in}}$ and codomain size $|Y| = 2^{b_{out}}$, specification requires $\mathcal{O}(2^{b_{in}} \times b_{out})$ bits.
- **Capacity:** The Vapnik-Chervonenkis (VC) dimension is strictly $|X|$ (or infinite for infinite domains). Every possible mapping can be represented, provided sufficient memory.
- **Generalization:** None. Out-of-distribution (OOD) generalization is impossible without any inductive bias or structural priors.
- **Learnability & Optimization (Triviality):** Learning is an $\mathcal{O}(1)$ memory allocation and storage operation per sample. There is no optimization landscape to navigate, no loss surface, and no credit assignment. The computational cost of training and querying can be constant in the case of a hashmap, or can scale gracefully with other datastructures.

Level 1: Metric & Topological Interpolators (Non-Parametric Smoothness)

The domain X is endowed with a metric $d_X(x, x')$ or a specified topological structure, allowing the representation to leverage the native geometry of the input space.

- **Concept:** The "ordered look-up table." Outputs for novel inputs are interpolated based on proximity to known inputs, bounded by a smoothness constraint (e.g., Lipschitz or Hölder continuity). Examples include k-Nearest Neighbors, Splines, and Kernel Methods.
- **Topological Mechanism:** The function constructs a non-parametric cover of the space. It relies entirely on the ambient metric d_X to define local neighborhoods, meaning it cannot discover new geometries; it merely smooths over the existing one.
- **Mathematical Grounding (Approximation Theory):** The complexity of this class is measured by Metric Entropy (or Kolmogorov ϵ-entropy). To approximate a function with smoothness s in a d-dimensional space to an error ϵ, the required number of interpolation points scales as $\mathcal{O}(\epsilon^{-d/s})$.
- **Capacity:** The capacity scales strictly with the size of the observed data, theoretically capable of approximating any continuous function given infinite samples.
- **Generalization:** Can be robust; inductive bias is obviously toward in-distribution interpolation. OOD generalization (extrapolation) fails gracefully; the degradation is deterministic, governed directly by the chosen metric. In-domain generalization is inherently bottlenecked by the **Curse of Dimensionality** (evident in the d/s exponent), as the volume of the space grows exponentially faster than it can be populated by data points.
- **Learnability & Optimization (Convex Bottlenecks):** Many methods (kNN, splines) do not have optimization and hence do not have a loss landscape. For kernel methods (e.g., Kernel Ridge Regression, SVMs), the optimization landscape is strictly convex, guaranteeing convergence to a global optimum without the risk of local minima. However, learning is heavily bottlenecked by computational complexity rather than optimization difficulty. Constructing and inverting the kernel Gram matrix requires $\mathcal{O}(N^3)$ time and $\mathcal{O}(N^2)$ memory. This can be remedied with random features, as discussed in the next section.

Level 2: Shallow Parameterized Maps (Fixed-Basis Global Approximation)

The function takes the form $f(x) = \phi(x; \theta)$, decoupling the representation from the ambient metric d_X by embedding the input into a

learned or fixed parameterized space.[1]

- **Concept:** The mapping relies on a finite set of parameters decoupled from the size of the training data. This encompasses *single*-hidden-layer neural networks and fixed global basis expansions.
- **Topological Mechanism:** The map $x \mapsto \phi(x; \theta)$ pushes the data forward into a fixed-dimensional latent manifold, substituting the extrinsic metric of the input space with an intrinsic, learned metric. Rather than covering a space with small local metric balls (Level 1), it defines global level sets. For example, a hidden layer $\sigma(Wx + b)$ partitions the space globally; rather than defining a local region of interest, a single hyperplane slices the entire domain space in half.
- **Mathematical Grounding (Statistical Learning Theory):** Complexity is bounded by the size and numerical resolution of the parameter space $\mathcal{O}(|\theta|)$, formally measured by VC Dimension or Rademacher Complexity. This parameter compression enables a degree of geometric generalization.
- **Capacity:** By the Universal Approximation Theorem (UAT), a single hidden layer can approximate any continuous function on a compact subset of $\mathbb{R}^n$. However, achieving sufficient representational capacity may require the width of the basis (the dimensionality of θ) to grow exponentially with the input dimension.
- **Generalization:** Capable of global generalization, including limited OOD extrapolation, as always *if and only if* the true generative function lies within the span of the fixed hypothesis class. Empirically, has weaker inductive bias and sample efficiency than Level 3, though with better training dynamics.
- **Learnability & Optimization (Fixed-Basis Convexity):** Highly efficient. If the parameterized basis is fixed and randomized (e.g., Random Fourier Features, Reservoir Computing), learning is restricted to a final linear readout layer. This reduces the problem to convex Ordinary Least Squares, solvable in $\mathcal{O}(N|\theta|^2)$ time. If the parameters are actively trained (e.g., a shallow neural network), the landscape becomes non-convex but generally avoids the severe vanishing gradient pathologies of deeper architectures.

[1] **Ambient Topologies vs. Parameterized Latent Spaces:** A common point of confusion exists between kernel methods (Level 1) and fixed global bases like Reservoir Computing or Liquid State Machines (Level 2). The "Kernel Trick" evaluates $f(x) = \sum \alpha_i K(x, x_i)$. Because it explicitly references the training data x_i, it defines local neighborhoods based on the pairwise distances in the data, keeping it in Level 1. Conversely, fixed random basis expansions (such as LSMs for static inputs or Random Fourier Features) evaluate $f(x) = \sum \theta_i \phi_i(x)$. This maps the data into a fixed parameterized space entirely independent of the dataset size N, placing it in Level 2.

Level 3: Deep Compositional Functions (Hierarchical Architectures)

The introduction of sequential algebraic composition and intermediate representations: $f(x) = (h_L \circ h_{L-1} \circ \cdots \circ h_1)(x)$.

- **Concept:** The function permits exponential compression of specific representational classes by computing intermediate states, reusing sub-functions to build hierarchical features.
- **Topological Mechanism:** Each layer h_i is a piecewise affine map that acts to fold and quotient the space. Geometrically, a shallow network (Level 2) requires k distinct hyperplanes to polygonize a circular decision boundary; a deep network can use absolute-value folds (e.g., ReLUs) to fold the space such that after 4 folds a *single* linear hyperplane cut creates a 16-sided polygon. Furthermore, intermediate layers act as quotient maps, identifying structurally equivalent points (symmetries and invariances) that may be highly distant under the ambient metric d_X.
- **Mathematical Grounding (Circuit & Approximation Theory):** Deep networks bypass the bottlenecks of shallow networks via **Depth Separation Theorems**.
  - *Discrete space:* In Boolean circuit complexity, calculating the parity of N bits requires $\mathcal{O}(2^N)$ depth-2 gates. However, allowing intermediate states (a deeper circuit of XOR gates, NC^1 complexity) solves it with $\mathcal{O}(N)$ gates; parity $\notin \mathsf{AC}^0$ for any constant depth; a log-depthNC^1 XOR-tree does it in $O(N)$.
  - *Continuous space:* Telgarsky (2016) and Eldan & Shamir (2016) demonstrated that there exist functions computable by deep neural networks of polynomial width that require strictly *exponential* width to be approximated by shallow networks.
- **Capacity:** Depth allows the construction of decision boundaries with exponentially complex Betti numbers[5] (topological holes and connected components) using $\sim$ linear number of parameters.
  - Bianchini and Scarselli (2014) showed that the sum of Betti numbers of a deep network's expressible regions can grow exponentially with its depth $\mathcal{O}(2^L)$, while only requiring parameters proportional to $\mathcal{O}(LW^2)$ where L is depth and W is width. A shallow network would require an exponentially wider layer to achieve the same topological complexity.
- **Generalization:** *If* training succeeds, enables **compositional generalization**. Because intermediate representations capture structural invariances (e.g. quotienting via nonlinearities), the function can extrapolate accurately to OOD data. However, as the folds that capture the invariances and symmetries are (almost always) static, in struggles to represent higher-order invariances, like parameterized manifolds.

[5] In topology, Betti numbers simply count the number of "holes" of different dimensions in a shape. The 0-th Betti number is the number of distinct connected pieces, the 1st is the number of circular holes (like a donut), and the 2nd is the number of hollow voids (like a balloon). In the context of classification, higher Betti numbers mean the decision boundary can separate highly fragmented or "swiss-cheese-like" clusters of data.

- **Learnability & Optimization (Non-Convex Credit Assignment):** The optimization landscape is highly non-convex, riddled with saddle points, and computationally demanding. Learning requires backpropagation to solve complex credit assignment across hierarchical layers. The compute cost scales linearly with depth and quadratically with width, necessitating massively parallel hardware. Crucially, learning here relies heavily on the geometry of *overparameterization*, which acts to smooth the loss landscape and make otherwise unnavigable local minima into saddles traversable via stochastic gradient descent.

Level 4: State-Bound Sequential Machines (Dynamical Systems and Automata)

The domain shifts from fixed-length vectors to trajectories and sequences: $f : X^* \to Y^*$. This entails temporal recursion and/or internal state. *Levels 0–3 concern approximation of fixed-arity maps; 4–6 concern computation over unbounded inputs, where the relevant complexity measure changes from hypothesis-class capacity to computational model power.*

- **Concept:** Computations are defined via the recurrence $s_t = \psi(s_{t-1}, x_t)$, processing inputs along a strictly 1-dimensional metric progression (time) via a fixed-size internal memory.[6]
- **Topological Mechanism:** This level bridges the continuous and discrete domains. These functions are dynamical systems, so evaluation shifts from pointwise distances to trajectory distances (e.g., dynamic time warping). If the function map contains distinct attractors, it stores information; if it is a contraction, it exhibits fading memory. If the state is topologically discrete then it is a type of finite automata.
- **Mathematical Grounding (Automata Theory):** Governed formally by the Chomsky Hierarchy and the Myhill-Nerode Theorem. Fixed-state sequential machines map perfectly to Finite State Machines (bounded $\mathcal{O}(1)$ memory, so cannot solve $a^n b^n$) grammars; with augmented fixed-mechanism memory, Push-Down Automata (can solve $a^n b^n$ but not $a^n b^n c^n$) etc.
- **Capacity:** Memory and state-transitions are bounded by the internal capacity of the transition function ψ, but the temporal operations scale linearly with the sequence length.
- **Generalization:** Introduces **systematic length extrapolation**. The model can be trained on trajectories or sequences of length N and generalize to length $N + M$ (OOD), provided the underlying generative grammar or dynamic transition rule ψ was correctly identified. In practice, this is quite hard: semi-continuous[7] transformers and RNNs struggle with length generalization.

[6] With unbounded precision and time, RNNs are Turing-complete (Siegelmann–Sontag) - here we assume finite precision.

[7] Floats are rationals, not reals

- **Learnability & Optimization (Temporal Pathologies):** Dominated by the extreme difficulty of temporal credit assignment. Because the mapping $s_t = \psi(s_{t-1}, x_t)$ is iterated recursively, the Jacobian of the transition function is multiplied sequentially over time. Depending on the spectral radius of this Jacobian, gradients will deterministically vanish or explode (Backpropagation Through Time). Learning long-term dependencies requires heavy architectural interventions—such as gating mechanisms (LSTMs) or structured linear state-spaces—to explicitly force the memory dynamics to remain stable and differentiable.

Level 4.5: Expanding state sequential machines (Transformers)

Polynomially bounded computation with linearly growing state. This state can be appended to, but cannot be modified, which enables parallel training.

- **Concept:** Rather than compressing history into a fixed-capacity vector s_t (Level 4), computation acts over an explicitly growing memory buffer (e.g., the Context Window / KV Cache). This class is inherently composite: it evaluates sequences by embedding a Level 1 kernel smoother (softmax attention) parameterized by Level 2 affine maps (FFN layers), stacked into a Level 3 deep hierarchy.
- **Topological Mechanism (Dynamic computation graphs):** Replaces the strictly 1-dimensional progression of time with a dynamically weighted, fully connected graph. The topological distance between the 1st and N-th input token is forcibly collapsed to a single $\mathcal{O}(1)$ hop.[8] The "metric" determining interactions is not the ambient space, but a dynamically computed data-dependent affinity (query-key dot products).
- **Mathematical Grounding (Circuit Complexity):** A bounded-depth Transformer operating in a single forward pass without recurrence is formally restricted to the complexity class TC^0. It is theoretically incapable of computing inherently sequential problems (e.g., evaluating arbitrary deterministic finite automata or calculating parity over long strings) purely in the forward pass. However, when combined with chain-of-thought, the output tape itself serves as an externally expanding state, effectively mimicking Level 5 algorithmic indirection and pushing the model's expressivity toward P-completeness (Merrill & Sabharwal).
- **Capacity:** State capacity grows linearly $\mathcal{O}(N)$ with the sequence, completely eliminating the information bottleneck and fading memory limits of Level 4 recurrent systems. This perfect recall comes at the cost of computational complexity, which (in standard dense attention) scales quadratically $\mathcal{O}(N^2)$, bounding the maximum sequence length by available physical memory bandwidth rather than theoretical representational limits.

[8] This dense routing topology effectively simulates the "multiple heads" of a Level 5 Turing Machine, allowing distant structural dependencies to interact directly without physical temporal traversal.

- **Generalization:** Exhibits deeply bifurcated generalization properties. Systematic length extrapolation (OOD length) is notoriously fragile; because the attention manifold is calibrated for a specific maximum N during training, longer sequences fundamentally distort the softmax entropy and positional geometries. Conversely, the explicitly preserved state enables profound **in-context learning**: the architecture can act as a meta-learner, essentially performing non-parametric Level 1 interpolation over the novel token state provided at inference time, achieving algorithmic generalization without weight updates.
- **Learnability & Optimization (Parallel Credit Assignment):** Exceedingly favorable for gradient descent. By unrolling the temporal recurrence into a spatial graph (causal masking over a simultaneous batch), it completely bypasses the sequential Jacobian multiplications that cause vanishing/exploding gradients in Level 4 (BPTT). The error signal propagates directly across all tokens in $\mathcal{O}(1)$ depth per layer. Optimization is therefore bottlenecked not by the fragility of sequential temporal dynamics, but by the memory limits of large batch, high-dimensional non-convex geometry (Level 3).

Level 5: Turing-Complete Computation

Unbounded computation with unbounded memory. The function is instantiated by an algorithm that dynamically manipulates arbitrary state.

- **Concept:** Functions where the mapping from input to output is achieved via algorithms capable of navigating an arbitrary amount of state over an arbitrary number of steps.
- **Topological Mechanism (The Geometry of Indirection):** Computation in Levels 4 and 4.5 are bound to the fixed 1D topology of Time. Level 5 escapes this: ***the order of computation is itself computed***.
  - This in turn leads to the natural architectural hierarchy within Turing-complete computation. If computation is fundamentally the traversal of a directed acyclic dependency graph, then a single-head, single-tape Turing Machine forces a strict 1-dimensional embedding of this graph. If causally linked variables are distant, the machine must physically traverse the intervening tape, costing time.
  - Introducing *multiple heads* on a single tape allows concurrent access to distant regions, effectively acting as dynamic "pointers" that bridge disparate parts of the graph without requiring continuous physical traversal.
  - A *multi-tape* Turing Machine extends this further by providing parallel, independent 1-dimensional workspaces, allowing complex data routing and decoupling previously entangled computations.

  - Taking this to the limit, Random Access Memory (RAM) induces absolute *new locality*.
  - Memory indirection[9] acts as a dynamically constructed shortcut edge in the dependency graph, collapsing the metric distance between any two causally linked states strictly to $\mathcal{O}(1)$.[10]
  - Example: palindrome recognition requires $O(N^2)$ on a single-head single-tape TM; can be done in $O(N)$ with two heads or two tapes.

[9] An old adage in computer science says that all problems can be solved with another level of indirection...

[10] In reality, this is $O(\log n)$ cost due to the cost of communication (speed of light, energy) as bits must be in physically different locations. (Compare this with the quadratic cost of graph attention in a transformer.)

- **Mathematical Grounding (Computability Theory):** The complexity of a function is no longer measured by VC dimension, but by **Kolmogorov Complexity** $K(x)$ (the length of the shortest program that computes the output).
- **Capacity (Time/Space Complexity):** Driven by the Time Hierarchy Theorem. A multi-tape Turing Machine can solve certain problems in $\mathcal{O}(N \log N)$ time that a single-tape TM requires $\mathcal{O}(N^2)$ time to solve, purely by overcoming the topological constraints of a 1D tape.
- **Generalization:** Algorithmic equivalence. This represents the ultimate theoretical ceiling of OOD generalization: if the exact algorithmic generator of the data is discovered, generalization is perfect.
- **Learnability & Optimization (Combinatorial Discontinuity):** Exceptionally hostile and fundamentally non-differentiable. The search space of discrete algorithms and logic is vast, discontinuous, and lacks the smooth gradient geometry required for continuous optimization. Learning at this level (e.g., Program Synthesis, Inductive Logic Programming) requires navigating combinatorial spaces via discrete search algorithms (Monte Carlo Tree Search, Evolutionary Algorithms, or Reinforcement Learning). Finding the optimal algorithm is theoretically uncomputable in the limit, forcing practical learning to rely on heuristics.

Level 6: Hypercomputation (The Theoretical Ceiling)

Functions that represent valid mathematical mappings $f : X \to Y$ but cannot be instantiated, evaluated, or learned by any finite physical or algorithmic computational device.

- **Concept:** Mappings that inherently require a Turing Oracle or infinite computational steps to resolve. They exist strictly as abstract mathematical realities rather than executable procedures.
- **Topological Mechanism (The Limit of Reachability):** Not very meaningful. In the geometric interpretation of computation as the traversal of a dependency graph, these functions are analogous to disconnected components or paths of infinite length.

- **Mathematical Grounding (Computability & Set Theory):** Governed by the strict limits of uncomputability. Canonical examples include the Halting Function (determining if an arbitrary Turing machine halts), the Busy Beaver function, and the exact calculation of Kolmogorov Complexity itself.

- **Capacity:** Complete. This level encompasses the entirety of the mathematical universe, including the uncountable infinities of mappings that transcend Turing computability (demonstrable via Cantor's diagonal argument).

- **Generalization:** Empirically moot. Because these functions cannot be represented by computable architectures or approximated to arbitrary precision by finite algorithms, the concept of generalization does not apply. Its inclusion in this taxonomy serves strictly to establish the absolute upper bound, delineating the universe of pure mathematical mappings from the infinitely smaller subset of computable representations.

- **Learnability & Optimization :** Impossible.