Introduction to "No Free Lunch" in Optimization and Learning

Thomas M. English

The Tom English Project
Lubbock, Texas USA

Tom.English@ieee.org

http://members.door.net/tmenglish

What is Conservation?

- Generalized conservation (statistical perspective)

  Performance is said to be conserved for a class of algorithms if all members of the class have identically distributed performance under a given distribution of inputs.

- Alternatively, there is *no free lunch* (NFL).
- Wolpert proved NFL assuming uniform problem distributions
  - for learning in 1992 and 1994
  - for optimization in 1995 (with Macready)

Scope of the Tutorial

- Unify *optimization* and *active category learning* under the rubric of *function exploration*
- NFL in function exploration from statistical perspective
- Distribution-free results from algorithmic information theory
  - more fundamental than the statistical perspective
  - almost all functions have little or no exploitable structure
  - length of exploration program bounds information gain
- Some remarks on practical significance

A Realistic Representation of Functions

- Ordered domain $D = \{d_1, \ldots, d_M\}$ and codomain $\Sigma = \{0,1\}^L$

  e.g., $D = \{1,2\}$ and $\Sigma = \{0,1\}^2 = \{00, 01, 10, 11\}$

- Set of all functions, $\mathcal{F} = \{f \mid f : D \to \Sigma\}$

- Natural *description* of $f \in \mathcal{F}$ as string $f(d_1) \ldots f(d_M) \in \Sigma^M$

  e.g., for $f(1) = 01$ and $f(2) = 11$, description is 0111

- Alternative treatment as string of $N = LM$ bits

  e.g., $0111 \in \{0,1\}^4$

- Functions and descriptions are in 1-to-1 correspondence

Introduction to Function Exploration

- Restrict learning to *active category learning*
  - select *unvisited* points in domain of function
  - guess their values
- Allows unification under rubric of *function exploration*
  - "evaluate" points by reading bit fields from description
  - write *trace* of values in order read
- Learning requires that values be guessed before they are read
- Performance evaluation is quite different
  - learning: percentage of correct guesses
  - optimization: any function of the trace

Example of Function Exploration

- $|D| = 4$, $\Sigma = \{0,1\}^2$ interpreted as unsigned base-2 integers
- (*description*, *trace*) with asterisk (*) for ‘‘unknown’’

  (********, λ) [guess 00, visit d_2]

  (**10****, 10) [guess 10, visit d_4]

  (**10**11, 1011) [guess 11, visit d_1]

  (0010**11, 101100) [guess 00, visit d_3]

  (00100011, 10110000)
- Guessing accuracy of 25%
- Max. value found with 2 evaluations, min. with 3
- Optimization omits the guesses

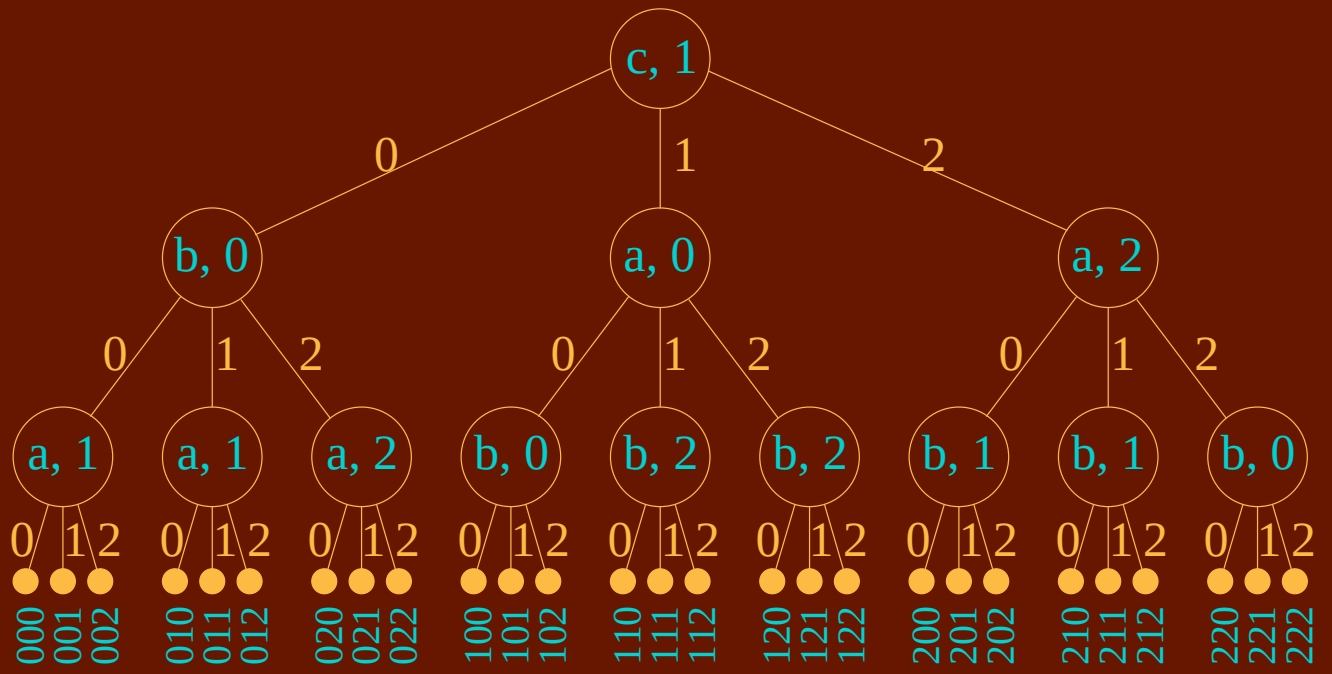
Function Exploration: Assumed Properties

- *Nonredundant*–no domain point is evaluated twice
- *Exhaustive*–every domain point is evaluated in finite time
- *Algorithmic*–the procedure halts after a finite number of steps
- *Deterministic*–e.g., pseudorandom with constant seed
- *Honest*
  - trace lists values in order observed
  - for parallel evaluation, convert partial ordering of values to a compatible total ordering without referring to the values

Generic Algorithm for Sequential Exploration

```
Algorithm Explore
   π ← y ← g ← a ← λ
   While |π| < M Loop
      Select p ∈ {1, ..., M} Not In π
      Include if learning: Guess a ∈ Σ
      Read x_p into b
      π ← πp
      g ← ga
      y ← yb
   EndLoop
   Write yg
EndAlgorithm Explore
```

Equivalence to Decision Trees



- Point to evaluate and guess of its value label branch node
- Observed values label edges, each function has a distinct path
- Bijection between descriptions and traces (leaf labels)

Elementary Conservation in Exploration

- Consider processing each function description exactly once.
- Every trace occurs exactly once.
- Thus all exploration algorithms have identically distributed traces when functions are equiprobable.
- For any measure of performance that is a function of the trace, performance is conserved.
- At each branch node, each outgoing edge is traversed equally often, and guessing accuracy is 1 in 2^L for any exploration algorithms.

Necessary and Sufficient Conditions for NFL

- The random component of an r.v. distributed on $\mathcal{F}$ is equivalent to jointly distributed r.v.'s, $Y = \{Y_1, \ldots, Y_M\}$
- Y_i is the random value of domain point d_i
- Optimization performance as any function of the trace

  For $n = 1, \ldots, M$, all size-n subsets of Y are identically distributed

- Learning performance as accuracy in guessing values of unvisited points

  Random values Y_i are i.i.d. uniform on Σ^L, $1 \leq i \leq M$

NFL Example: Needle-in-a-Haystack Functions

- Let test functions be uniformly distributed on the set of all functions $f : D \to \{0, 1\}$ s.t. $f(x) = 1$ for exactly one $x \in D$
- Minimization is very easy, so NFL does not mean "hard"
- Unlike the case of uniformly distributed functions, observed values provide information about unobserved values
- Before 1 is observed, each observation of 0 increases probability of 1 equally at all unvisited points
- When 1 is observed, values of all unvisited points are surely 0
- No observation differentiates one unvisited point from another

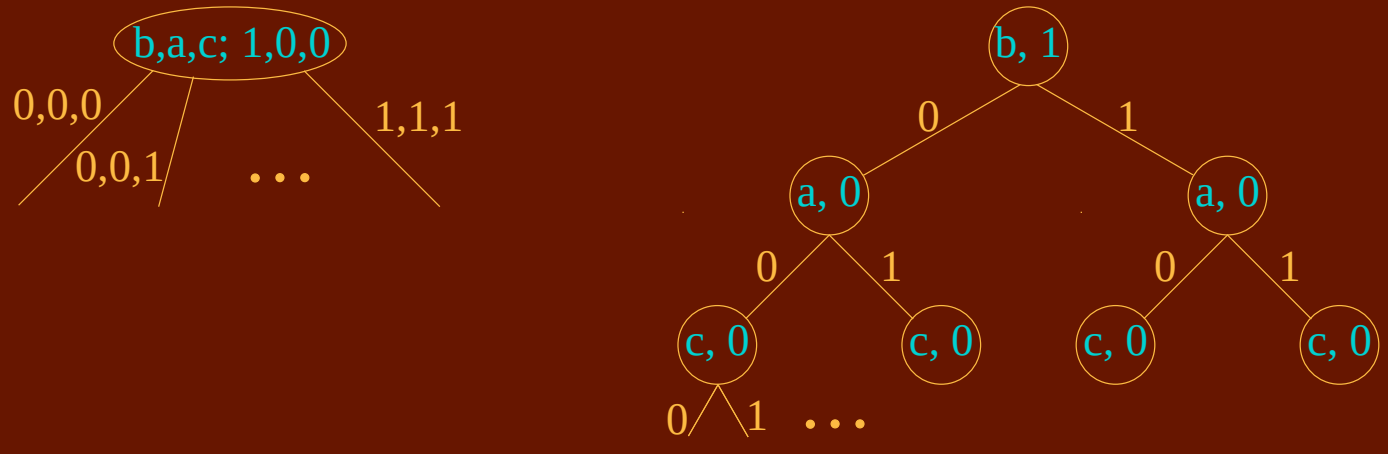
Needle-in-a-Haystack Exploration

- Use asterisk (*) to represent unobserved bit
- With *****, probability of 1 at each unvisited point is 1/5
- With *0***, probability of 1 at each unvisited point is 1/4
- With *0*0*, probability of 1 at each unvisited point is 1/3
- With *010*, 0 is certain at each unvisited point

Note: For uniform distribution on $\mathcal{F}$, probabilities are constant at 1/2

Notes: Parallel and Randomized Algorithms

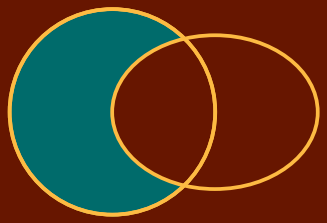
- A randomly seeded pseudorandom exploration algorithm implements a randomly selected decision tree on each run.
- Conservation results are preserved if the seed and function distributions are independent.
- Simple conversion of parallel decisions to sequential:



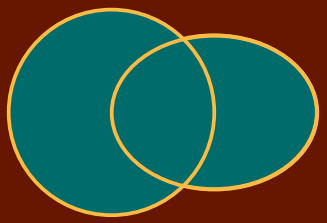
Shift in Paradigm: Algorithmic Complexity

- The information of a function description . . .
  - has been its extrinsic "surprise" as the realization of r.v.
  - is now the intrinsic *complexity* of computing it.
- Pick any universal computer, describe strings with programs
  - programs are *self-delimiting* binary strings
- The *algorithmic complexity* of $x \in \{0,1\}^*$, $h(x)$, is the length of the shortest program that outputs x on null input and halts.
  - think of length of self-extracting, compressed data archive
  - changing universal computer changes complexity by $O(1)$

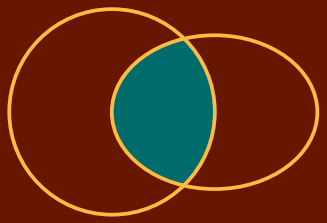
Fundamental Relations on Complexity



$h(x \mid y)$



$h(x, y)$



$h(x : y)$

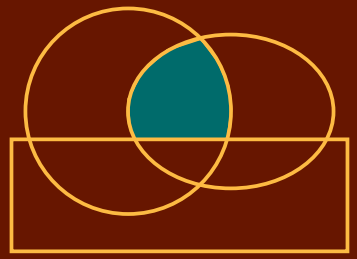
- For x, $y \in \{0,1\}^*$, circle represents $h(x)$, ellipse $h(y)$
- Complexity of x relative to y, $h(x \mid y)$, is the minimum length of a program to compute x given code to compute y "for free"
- Joint complexity of x and y is

  $$h(x, y) = h(x \mid y) + h(y) + O(1)$$

- Mutual complexity of x and y is

  $$h(x : y) = h(x, y) - h(x \mid y) - h(y \mid x)$$

Mutual Relative Complexity



$h(x : y \mid p)$

- Box represents $p \in \{0,1\}^*$
- Shaded region is mutual complexity of x and y relative to p

How Many Programs Have Fewer than n Bits?

- There are at most $|\{0,1\}^k| = 2^k$ halting programs of length k
- Fewer than 2^n halting programs are of length less than n, i.e.,

$$\sum_{k=0}^{n-1} 2^k = 2^n - 1$$

- Not all programs halt, and many halt with no output
- If programs halt with identical output, all but one is discarded
- Hence 2^n is a loose bound on the number of programs of length less than n
- It follows immediately that fewer than 2^n function descriptions have complexity less than n

Algorithmic Randomness

- $x \in \{0,1\}^N$ is *algorithmically random* iff $h(x \mid N) \geq N$
  - i.e., randomness is algorithmic incompressibility
  - entails all computable tests for randomness
  - holds for more than half of all descriptions in $\{0,1\}^N$
- If $h(x \mid N) \approx N$, then x is *highly random*.
- **Almost all function descriptions are highly random**
  - fewer than 2^{N-k} descriptions x with $h(x \mid N) < N - k$
  - i.e., fewer than 1 in 2^k descriptions can be compressed by more than k bits
  - justification: $2^{N-k}/2^N = 2^{-k}$

Realistic Example: Functions on 32-bit Integers

- Let $D = \Sigma$ be the set of 32-bit integers of some computer.

  $N = 2^{32} \cdot 32 = 2^{37}$

  Let $k = N/1024 = 2^{27}$, about 0.1% of length

- Fraction of descriptions x such that $h(x \mid N) < N - k$ is less than $2^{2^{-27}} = 2^{-134217728}$

- For almost all descriptions x, $h(x \mid N) > 0.999\, N$.

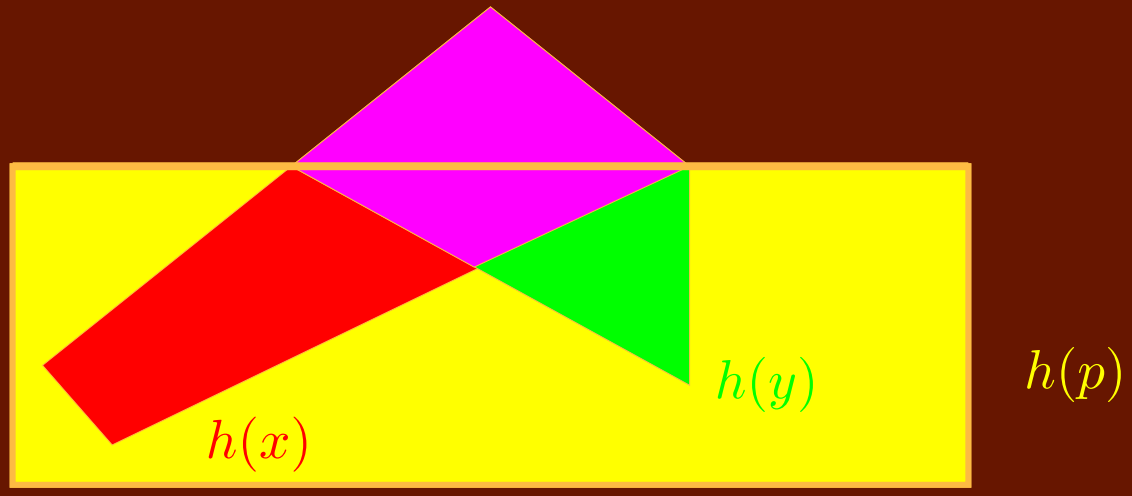
  – i.e., almost all descriptions are highly random

Can Exploration Reduce Complexity?

- Summing over all functions, no.
  - The description-to-trace mapping is a bijection on $\{0,1\}^N$.
  - The set of all traces is the set of all descriptions.
  - Description and trace complexity difference is zero-mean.
- For a particular function, complexity gain is possible:
  - i.e., $|h(y \mid N) - h(x \mid N)| > 0$
  - negative gain is possible
- If p is the exploration program, $h(p)$ bounds the magnitude of complexity gain.

Conservation of Algorithmic Information

- Let y be the trace generated by p on input of description x
- Complexity expressions are implicitly relative to N
- Key identity: $h(x : y \mid p) = h(y \mid p) + O(1) = h(x \mid p) + O(1)$
- Conservation: $|h(x) - h(y)| = |h(x : p) - h(y : p)| + O(1) \leq h(p)$



Optimization Is Almost Always Easy

- Let k be number of unsatisfactory x_i in x, with $h(x \mid N) \approx N$.
  - satisfactory values are randomly located
- Random sampling is not only optimal, but efficient.
  - probability of unsuccessful m-sample is less than $(k/M)^{-m}$
- Example: bijections on 32-bit integers
  - objective is to find a value better than 99.999% of all others
  - probability of failure for sample size of 1 million is less than 5 in 100 thousand

Learning Is Almost Always Hard

- For $h(x \mid N) \approx N$, all short programs guess poorly.
- If n is the number of incorrect guesses made by p on input x,

$$\lg M + \lg \binom{M}{n} + n\,L \geq h(x \mid N) - h(p \mid N) + O(1)\ .$$

- "The error complexity is not much less than the difference in complexity of the description and the program."
- There appears to be no analogous bound for optimization
  - Learning performance is always a matter of information
  - Uninformed optimizers usually perform as well as any other

Informal Derivation of the Learning Bound

- Modify p to obtain p^* that writes x on null input
  - encode the guessing errors of p internally
  - use correct guesses and stored values in place of inputs
- The inequality follows from simple encoding of errors in p^*:
  - $\lg M$ bits encode the number of errors, n
  - $\lg \binom{M}{n}$ bits encode the subset of n points with incorrectly guessed values
  - simple listing of the correct values requires nL bits
- $h(x|N) \leq h(p^*|N) = h(p|N) + \lg M + \lg \binom{M}{n} + nL + O(1)$

Practical Considerations: Socratic Introduction

True or False (and why):

- The best performance on benchmark problems is paid for with the worst performance on all other problems.
- Adaptive optimizers perform better than non-adaptive.
- Performance is not conserved for practical distributions.
- A long program may guess a low-complexity function poorly.
- Sometimes short programs guess highly random functions well.
- Complexity determines expected time to locate a good value.

Discussion

- Near-ubiquity of highly random functions underlies conservation, making the function distribution a non-issue.
- Why does randomness not seem pervasive to us?
  - We work with functions of our own contrivance.
  - In mathematical culture, the complex is culled.
  - A function with a short code is low in complexity.
- To perform well . . .
  - a learner needs information about most of the description.
  - an optimizer needs no information unrelated to good points.
- Conservation results for partial exploration are similar.

Discussion (cont.)

- Almost all functions are highly random, and we have unconsciously focused upon a tiny pocket of order.

  "We don't know what we don't know."

- "Just random search" is almost always an optimal optimizer.
- A learner's accuracy is close to the chance rate almost always.
- Optimization is not complexity reduction, but learning is.
- Complexity reduction is bounded by the length of the exploration program.

Ramifications for Performance Studies

- **Note**: Real programs don't have the hypothesized properties
  - may revisit points too often
  - may spend too much time avoiding revisiting points
  - may be generally inferior, not generally superior
- Performance is not conserved for some problem distributions
  - quadratic optimization is a simple example
- In light of NFL, the primary goal should be to determine *when* a program is the best, not *which* program is the best
  - a good engineer chooses the right tool for the job
  - our community has not yet reached that level of maturity

Algorithm Design for Applications

- Algorithms almost always require tuning for particular applications
  - application-specific information enters the algorithm
  - typically, parameters are tweaked over a series of runs
  - i.e., the algorithm is evolved with an informal strategy
- This is no mystery in the context of foregoing analysis
  - algorithms do not "self-adapt" to problem instances
  - practitioners fit algorithms to problems

Within-Run vs. Run-to-Run Adaptation

- Within a run, a program cannot learn the distribution of unobserved values
- Only run-to-run learning of the value distribution is possible
- Why does self-adaptation of parameters work sometimes?
  - self-similarity of the function is generally assumed
  - i.e., what brings good results in one context brings good results in others
  - self-adaptation succeeds to degree that assumption holds
- Run-to-run learning is sensible if what works well for one instance of the problem should work well for other instances

Conclusion

- "Optimization is Easy and Learning is Hard in the Typical Function"
- We work with functions that are far from typical

  e.g., their descriptions fit in real digital computers
- But the statistical criterion for NFL may hold for optimization
- Also, conservation of algorithmic information in learning is of practical significance

  i.e., a small learning program cannot achieve high accuracy on a complex function description

Bibliography

- English, T. M. 1999. "Some information theoretic results on evolutionary optimization," in *Proc. 1999 Congress on Evolutionary Computation: CEC99*, Piscataway, New Jersey: IEEE Service Center, pp. 788-795.
- English, T. M. 2000. "Optimization is easy and learning is hard in the typical function," in *Proc. 2000 Congress on Evolutionary Computation: CEC00*, Piscataway, New Jersey: IEEE Service Center, pp. 924-931.
- Wolpert, D. H. 1992. "On the connection between in-sample testing and generalization error," *Complex Systems*, vol. 6, pp. 47-94.
- Wolpert, D. H. 1995. "Off-training set error and *a priori* distinctions between learning algorithms." Santa Fe Institute technical report 95-01-003, available at http://www.santafe.edu.
- Wolpert, D. H., and W. G. Macready. 1997. "No free lunch theorems for optimization," *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 67-82.