TTIC 31250
An Introduction to the Theory of Machine Learning

Computational Hardness of Learning

Avrim Blum

05/11/20

•1

Computational Hardness of Learning

- We know efficient algorithms for various problems:
  - Given a dataset S, find a consistent decision list if one exists.
  - Given a dataset S, find a consistent linear threshold function if one exists.
  - Given a dataset S, find a consistent disjunction if one exists.

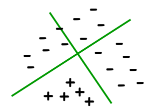
•2

Computational Hardness of Learning

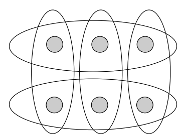
- But what about the following?
  - Given a dataset S, find a consistent AND of 2 linear threshold functions (intersection of two halfspaces) if one exists.
  - Given a dataset S, find a consistent AND of 2 disjunctions (2-clause CNF formula) if one exists.
  - Given a dataset S, find a linear threshold function with the fewest mistakes on S.
- Can we get algorithms that are guaranteed to solve these in polynomial-time without assumptions on the distribution, etc.?
- Answer: don't expect to because they are NP-hard

•3

Hardness for intersection of 2 halfspaces



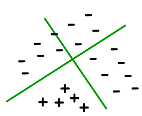
- Reduction from NP-hard hypergraph 2-coloring problem.



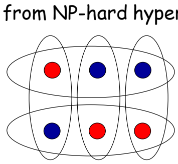
Given m subsets $S_1, \ldots, S_m$ of n nodes, color each node Red or Blue such that each S_i has at least one red node and at least one blue node.

•4

Hardness for intersection of 2 halfspaces



- Reduction from NP-hard hypergraph 2-coloring problem.



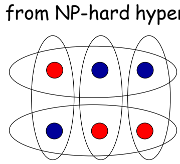
Given m subsets $S_1, \ldots, S_m$ of n nodes, color each node Red or Blue such that each S_i has at least one red node and at least one blue node.

•5

Hardness for intersection of 2 halfspaces

Given an instance of hypergraph 2-coloring, we want to create a set S of positive and negative examples such that: S is consistent with an intersection of 2 halfspaces iff the given instance was 2-colorable.

- Reduction from NP-hard hypergraph 2-coloring problem.

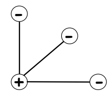


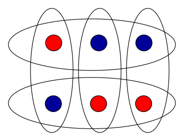
Given m subsets $S_1, \ldots, S_m$ of n nodes, color each node Red or Blue such that each S_i has at least one red node and at least one blue node.

•6

Hardness for intersection of 2 halfspaces

- Points in $\mathbb{R}^n$. Label origin positive, each coord vector $\hat{x}_j$ negative.
- For each set S_i, label indicator vector for that set as positive.



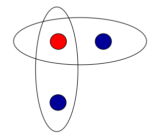


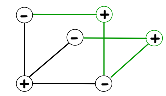
Given m subsets $S_1, \ldots, S_m$ of n nodes, color each node Red or Blue such that each S_i has at least one red node and at least one blue node.

•7

Hardness for intersection of 2 halfspaces

- Points in $\mathbb{R}^n$. Label origin positive, each coord vector $\hat{x}_j$ negative.
- For each set S_i, label indicator vector for that set as positive.



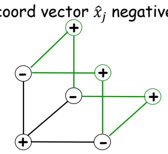


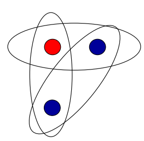
- Claim: data is consistent with an intersection of 2 halfspaces iff given instance was 2-colorable.

•8

Hardness for intersection of 2 halfspaces

- Points in $\mathbb{R}^n$. Label origin positive, each coord vector $\hat{x}_j$ negative.
- For each set S_i, label indicator vector for that set as positive.



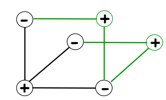


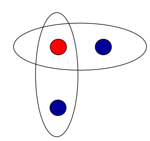
- Claim: data is consistent with an intersection of 2 halfspaces iff given instance was 2-colorable.

•9

Hardness for intersection of 2 halfspaces

- Points in $\mathbb{R}^n$. Label origin positive, each coord vector $\hat{x}_j$ negative.
- For each set S_i, label indicator vector for that set as positive.





- Claim: data is consistent with an intersection of 2 halfspaces iff given instance was 2-colorable.

Proof: Given a 2-coloring, just define halfspaces:

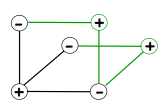
$w_1 x_1 + \ldots + w_n x_n \leq 1/2$, where $w_j = 1$ if j is Red and $w_j = -n$ if j is Blue

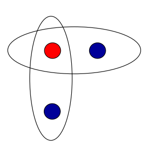
$w'_1 x_1 + \ldots + w'_n x_n \leq 1/2$, where $w'_j = 1$ if j is Blue and $w'_j = -n$ if j is Red

•10

Hardness for intersection of 2 halfspaces

- Points in $\mathbb{R}^n$. Label origin positive, each coord vector $\hat{x}_j$ negative.
- For each set S_i, label indicator vector for that set as positive.





- Claim: data is consistent with an intersection of 2 halfspaces iff given instance was 2-colorable.

Proof: Given halfspaces, call one "Red" and the other "Blue". If $\hat{x}_j$ is separated from orgin by Red halfspace, then color j Red; if separated by Blue halfspace then color j Blue; if separated by both, pick arbitrarily.

No 1-color set because convex hulls of $\{\hat{x}_j \in S_i\}$ and $\{\vec{0}, \vec{S}_i\}$ overlap.

•11

Hardness for learning 2-clause CNF

- A 2-clause CNF formula is an AND of two OR functions, e.g.,

  $(x_1 \vee x_2 \vee x_3)(x_4 \vee x_5 \vee x_6)$

- Also NP-hard to find a consistent 2-clause CNF if one exists.
- But, you can learn a 2-clause CNF using a 2-DNF representation:
  - Any 2-clause CNF can be multiplied out to a 2-DNF, e.g.,

    $x_1x_4 \vee x_1x_5 \vee x_1x_6 \vee x_2x_5 \vee x_2x_6 \vee x_3x_4 \vee \cdots$

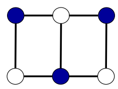
  - Learn using "list-and-cross-off" algorithm. But VC-dimension is now $\Theta(n^2)$.

Note: DNF is an OR of ANDs. Each AND is called a "term".

•12

How about finding the LTF with lowest empirical error?

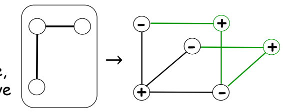
- We know how to find a consistent LTF when one exists. Can also minimize total hinge loss. But what about minimizing the training error?
- Turns out this is NP-hard.
- Reduction from the Maximum Independent Set problem in graphs



•13

How about finding the LTF with lowest empirical error?

- Similar reduction from before.
- Origin is positive, each coordinate vector $\hat{x}_i$ is negative, for each edge (i, j) put a positive at $\hat{x}_i + \hat{x}_j$.



- Notice that maximum independent set corresponds to largest set of negatives that can be linearly separated from the positives.
- This shows the problem "Find the LTF that correctly classifies all positives and makes fewest mistakes on negatives" is NP-hard.
- To finish off the argument, just replicate each positive example $n + 1$ times (so min error will only make mistakes on negatives)

•14

Representation-Independent Hardness

- These results show hardness for PAC learning using a particular hypothesis class H.
- (We gave hardness for consistency problem, but can set uniform distribution on the output of the reduction and set $\epsilon < 1/n_{points}$)
- Representation-independent: hardness based on complexity of target, allowing learner to use any representation it wants.

•15

Representation-Independent Hardness

Examples from last time:

- Parity functions require $2^{\Omega(n)}$ SQs of tolerance 1/poly(n) to learn in SQ model.
- Decision trees, DNF formulas require $n^{O(\log n)}$ SQs of tolerance 1/poly(n) to learn in SQ model.
- Hardness was even for doing slightly better than random guessing.

These results don't restrict representation used by learning algorithm, but they do restrict the way the algorithm can interact with the data.

What if you don't want to restrict either one?

•16

Representation-Independent Hardness

In this case, can use cryptographic assumptions.

A function $f: \{0,1\}^n \to \{0,1\}^m$ where $m > n$ is a pseudorandom generator if for any poly-time algorithm A, any constant c,

$$\left| \Pr_{v \sim \{0,1\}^m}(A(v) = 1) - \Pr_{x \sim \{0,1\}^n}(A(f(x) = 1) \right| = o\left(\frac{1}{n^c}\right).$$

In other words, no poly-time algo A can distinguish pseudorandom strings of length m (the result of running f on random input of length n) and truly random strings of length m.

Classic result: can construct generator f such that breaking f would give a poly-time algorithm for factoring. So, f is a PRG if factoring is hard. (Also known for some other hard problems too).

•17

Representation-Independent Hardness

Classic use for hardness of learning.

Any algorithm that can even weak-learn arbitrary $O(\log n)$-depth AND/OR networks over uniform random examples in polynomial time would give a poly-time algorithm for factoring.

High-level idea:

- Think of PRG with significant stretch: $\{0,1\}^n \to \{0,1\}^{poly(n)}$.
- Network has PRG input I built in, computes $f(I)$, and outputs jth bit, where j is given by low-order $\lg(n^2)$ bits of example x.
- If algo can learn, then can distinguish PRG output from true random.

•18

Representation-Independent Hardness

Classic use for hardness of learning.

Any algorithm that can even weak-learn arbitrary $O(\log n)$-depth AND/OR networks over uniform random examples in polynomial time would give a poly-time algorithm for factoring.

Can even extend to pseudo-random functions. These are indistinguishable from truly random even with query access.

•19

Representation-Independent Hardness

More recent results [Daniely 2016]:

Under a stronger assumption (next slide), for any constant c, there is no polynomial-time algorithm that given any sample S of n^c points in $\{-1,+1\}^n$ can whp distinguish the case (a) that labels are just uniform random coin flips, versus (b) there exists a linear separator of error at most 10%. (Can replace "10%" with any constant)

So this is pretty sad. Luckily, real-world problems are much nicer and simple local update algorithms have been very successful.

•20

Representation-Independent Hardness

More recent results [Daniely 2016]:

Assumption is that "refuting random k-XOR formulas" is hard.

This is very similar to the assumption that the thing we wish to prove is true for parity functions.

Specifically, assumption is that given $m < n^{\sqrt{k}\log k}$ examples over $\{0,1\}^n$ with k bits set to 1 in each, it is hard to distinguish the case that

(a) The examples and labels are random, from
(b) There is a parity function with error $\leq 10\%$.

•21

Overall

One major challenge of learning theory is to reconcile the power of deep learning (and the ability to use simple local updates to learn complex representations in general) with these worst-case hardness results. Clearly, the problems where deep learning succeeds are not worst-case, and understanding what makes them easier is a major research area.

•22