Representation Classes

- In our definition of efficient PAC learning, the algorithm $\mathcal{A}$, having no access to the target concept $c \in \mathcal{C}$, must work in polynomial time **independently** on c.

Representation Classes

- In our definition of efficient PAC learning, the algorithm $\mathcal{A}$, having no access to the target concept $c \in \mathcal{C}$, must work in polynomial time **independently** on c.
  - We assume concepts in $\mathcal{C}$ can be represented by way of binary strings, and each concept $e \in \mathcal{C}$ requires $size(e)$ bits. We talk of a **representation class**.

Representation Classes

- In our definition of efficient PAC learning, the algorithm $\mathcal{A}$, having no access to the target concept $c \in \mathcal{C}$, must work in polynomial time **independently** on c.
  - We assume concepts in $\mathcal{C}$ can be represented by way of binary strings, and each concept $e \in \mathcal{C}$ requires $size(e)$ bits. We talk of a **representation class**.
- Examples
  - X_n could be $\{0, 1\}^n$, the set of **boolean vectors** of of (fixed!) length n, and $\mathcal{C}_n$ is the set of all subsets of $\{0, 1\}^n$ *represented by CNFs.*
  - X_n could rather be $\mathbb{R}^n$, the set of **vectors of real numbers** of length n, while $\mathcal{C}_n$ are say, the subsets of $\mathbb{R}^n$ *represented by some form of neural network* with n inputs and 1 output.

Representation Classes

- In our definition of efficient PAC learning, the algorithm $\mathcal{A}$, having no access to the target concept $c \in \mathcal{C}$, must work in polynomial time **independently** on c.
  - We assume concepts in $\mathcal{C}$ can be represented by way of binary strings, and each concept $e \in \mathcal{C}$ requires $size(e)$ bits. We talk of a **representation class**.
- Examples
  - X_n could be $\{0,1\}^n$, the set of **boolean vectors** of of (fixed!) length n, and $\mathcal{C}_n$ is the set of all subsets of $\{0,1\}^n$ *represented by CNFs*.
  - X_n could rather be $\mathbb{R}^n$, the set of **vectors of real numbers** of length n, while $\mathcal{C}_n$ are say, the subsets of $\mathbb{R}^n$ *represented by some form of neural network* with n inputs and 1 output.
- In many cases (e.g. SGD), one has a *single* learning algorithm that work for every value of n. In that case, we allow (in the definition of efficient PAC learning) the algorithm $\mathcal{A}$ to take time polynomial in n, $size(c)$, $\frac{1}{\varepsilon}$ and $\frac{1}{\delta}$.

Boolean Functions as a Representation Class

- Suppose your instance class is $X = \cup_{n \in \mathbb{N}} X_n$ where $X_n = \{0, 1\}^n$.

Boolean Functions as a Representation Class

- Suppose your instance class is $X = \cup_{n \in \mathbb{N}} X_n$ where $X_n = \{0, 1\}^n$.
- One **first example** of a representation class for X_n is the class $\mathbf{CL}_n$ of all *conjunctions of literals* on the variables $x_1, \ldots, x_n$.
  - As an example, the conjunction

    $$x_1 \wedge \neg x_2 \wedge x_4,$$

    defines a subset of $\{0, 1\}^4$.
  - *Not all* subsets of $\{0, 1\}^n$ can be captured.

Boolean Functions as a Representation Class

- Suppose your instance class is $X = \cup_{n \in \mathbb{N}} X_n$ where $X_n = \{0,1\}^n$.
- One **first example** of a representation class for X_n is the class $\mathbf{CL}_n$ of all *conjunctions of literals* on the variables $x_1, \ldots, x_n$.
  - As an example, the conjunction

    $$x_1 \wedge \neg x_2 \wedge x_4,$$

    defines a subset of $\{0,1\}^4$.
  - *Not all* subsets of $\{0,1\}^n$ can be captured.
- A **second example** of a representation class for X is a class we know, namely the class $\mathbf{CNF}_n$ of CNFs over $x_1, \ldots, x_n$, which are conjunction *of disjunctions* of literals.
  - CNFs are normal forms of any boolean functions.
  - *All* subsets of $\{0,1\}^*$ can be captured this way.
  - We could even consider $k\mathbf{CNF}_n$ rather than arbitrary one, but this way we would lose universality.

Learning Conjuctions of Literals

- Suppose your target concept is a conjunction of literals c on n variables $x_1, \ldots, x_n$. How could a learning algorithm proceed?

Learning Conjuctions of Literals

- Suppose your target concept is a conjunction of literals c on n variables $x_1, \ldots, x_n$. How could a learning algorithm proceed?
- Data are in the form (s, b) where $s \in \{0,1\}^n$ and $b \in \{0,1\}$. The latter is a label telling us whether $s \in c$ or $s \notin c$.
- A learning algorithm could proceed by keeping a conjunction of literals h as its state, initially set to

  $$x_1 \wedge \neg x_1 \wedge x_2 \wedge \neg x_2 \wedge \cdots \wedge x_n \wedge \neg x_n.$$

  and updating it according to positive data (while negative data are discarded).
  - If $n = 3$, the current state of h is $x_1 \wedge x_2 \wedge \neg x_2 \wedge \neg x_3$ and we receive $(101, 1)$, the hypothesis h is updated as $x_1 \wedge \neg x_2$.

Learning Conjuctions of Literals

- Suppose your target concept is a conjunction of literals c on n variables $x_1, \ldots, x_n$. How could a learning algorithm proceed?
- Data are in the form (s, b) where $s \in \{0,1\}^n$ and $b \in \{0,1\}$. The latter is a label telling us whether $s \in c$ or $s \notin c$.
- A learning algorithm could proceed by keeping a conjunction of literals h as its state, initially set to

  $$x_1 \wedge \neg x_1 \wedge x_2 \wedge \neg x_2 \wedge \cdots \wedge x_n \wedge \neg x_n.$$

  and updating it according to positive data (while negative data are discarded).
  - If $n = 3$, the current state of h is $x_1 \wedge x_2 \wedge \neg x_2 \wedge \neg x_3$ and we receive $(101, 1)$, the hypothesis h is updated as $x_1 \wedge \neg x_2$.

Theorem

The representation class of boolean conjuctions of literals is efficiently PAC-learnable.

Intractability of Learning DNFs

- We know that conjunctions of literals are efficiently learnable. But they are highly incomplete as a way to represent boolean functions.

Intractability of Learning DNFs

- We know that conjunctions of literals are efficiently learnable. But they are highly incomplete as a way to represent boolean functions.
- Let us take a look at a *slight generalization* of conjunctions of literals as a representation class.
  - A **3-term DNF formula** over n bits is a propositional formula in the form $T_1 \vee T_2 \vee T_3$, where each T_i is a conjunction of literals over $x_1, \ldots, x_n$.
  - In a sense, this class is the *dual* to 3CNFs!
  - As such, it is more expressive than conjunctions of literals, but still not universal.

Intractability of Learning DNFs

- We know that conjunctions of literals are efficiently learnable. But they are highly incomplete as a way to represent boolean functions.
- Let us take a look at a *slight generalization* of conjunctions of literals as a representation class.
  - A **3-term DNF formula** over n bits is a propositional formula in the form $T_1 \vee T_2 \vee T_3$, where each T_i is a conjunction of literals over $x_1, \ldots, x_n$.
  - In a sense, this class is the *dual* to 3CNFs!
  - As such, it is more expressive than conjunctions of literals, but still not universal.

Theorem

If $\mathbf{NP} \neq \mathbf{RP}$, *then the representation class of 3-term DNF formulas is not efficiently PAC learnable.*

Is This the End of the Story?

Is This the End of the Story?

- **Definitely No!** Actually, we have just *scratched the surface* of computational learning theory.

Is This the End of the Story?

- **Definitely No!** Actually, we have just *scratched the surface* of computational learning theory.
- Models and results we did not have time to talk about include:
  - The VC Dimension.
  - The Fundamental Theorem of Learning.
  - The No-Free-Lunch Theorem.
  - Occam's Razor.
  - Positive and negative results about neural networks.
  - ...
- More information can be found in of the many excellent books on CLT, e.g.

  - Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar *Foundations of Machine Learning* Second Edition. The MIT Press. 2018
  - Shai Shalev-Shwartz and Shai Ben-David. *Understanding Machine Learning: from Theory to Algorithms* Cambridge University Press. 2014.
  - Michael Kearns and Umesh Vazirani. *An Introduction to Computational Learning Theory* The MIT Press. 1994.

Example Results about Neural Networks (from Kearns and Vazirani's Book)

Theorem 3.7 *Let G be any directed acyclic graph, and let C_G be the class of neural networks on an architecture G with indegree r and s internal nodes. Then the number of examples required to learn C_G is*

$$O\left(\frac{1}{\epsilon}\log\frac{1}{\delta}+\frac{(rs+s)\log s}{\epsilon}\log\frac{1}{\epsilon}\right).$$

Example Results about Neural Networks (from Kearns and Vazirani's Book)

Theorem 3.7 *Let G be any directed acyclic graph, and let C_G be the class of neural networks on an architecture G with indegree r and s internal nodes. Then the number of examples required to learn C_G is*

$$O\left(\frac{1}{\epsilon}\log\frac{1}{\delta}+\frac{(rs+s)\log s}{\epsilon}\log\frac{1}{\epsilon}\right).$$

Theorem 6.6 *Under the Discrete Cube Root Assumption, there is fixed polynomial $p(\cdot)$ and an infinite family of directed acyclic graphs (architectures) $G=\{G_{n^2}\}_{n\geq 1}$ such that each G_{n^2} has n^2 boolean inputs and at most $p(n)$ nodes, the depth of G_{n^2} is a fixed constant independent of n, but the representation class $C_G=\cup_{n\geq 1}C_{G_{n^2}}$ (where $C_{G_{n^2}}$ is the class of all neural networks over $\Re^{n^2}$ with underlying architecture G_{n^2}) is not efficiently PAC learnable (using any polynomially evaluatable hypothesis class). This holds even if we restrict the networks in $C_{G_{n^2}}$ to have only binary weights.*

Thank You!

Questions?