A THEORY OF THE LEARNABLE (L.G. VALIANT)

Theory Lunch Presentation

Claire Le Goues

05/20/10

HOW DO YOU KNOW THAT?

HOW DO YOU KNOW THAT?

```
bool is_a_duck() {
    return (walks_like_a_duck() &&
                   quacks_like_a_duck());
}
```

~~HOW DO YOU KNOW THAT?~~

```
bool is_a_duck() {
    return (walks_like_a_duck() &&
                          quacks_like_a_duck());
}
```

HOW DID YOU LEARN THAT?

"A program for performing a task [like recognizing ducks – Ed.] has been acquired by learning if it has been acquired by any means other than explicit programming."

Why This Paper Matters

Why This Paper Matters

- Present a general framework for reasoning about what is learnable as allowed by algorithmic complexity.

Why This Paper Matters

- Present a general framework for reasoning about what is learnable as allowed by algorithmic complexity.
- Introduce the idea of Probably Approximately Learnable (PAL) problems, or problems that are learnable in polynomial time, with high correctness.

Why This Paper Matters

- Present a general framework for reasoning about what is learnable as allowed by algorithmic complexity.
- Introduce the idea of Probably Approximately Learnable (PAL) problems, or problems that are learnable in polynomial time, with high correctness.
- Prove 3 classes of programs to be PAL.

Outline

1. General framework for defining Learning Machines, or programs that can learn/write/produce other programs of a particular type.
   - A Learning Machine for animal recognition, for example, might learn to write a program that recognizes whether a given animal is a duck.
2. Definition of a particular learning protocol.
3. Definition of when a program class is reasonably-learnable.
4. Definition/proofs of reasonably-learnable program classes.

A Restriction

A Restriction

- Focus on learning skills that involve recognizing whether a concept (boolean predicate) is true for given (boolean) data.

A Restriction

- Focus on learning skills that involve recognizing whether a concept (boolean predicate) is true for given (boolean) data.
- Learn to answer the question: is this animal a duck?

A Restriction

- Focus on learning skills that involve recognizing whether a concept (boolean predicate) is true for given (boolean) data.
- Learn to answer the question: is this animal a duck?

walks like a duck = true
purple = false
fluffy = true
yellow = true
beak = true
big = false
quacks like a duck=true
angry = false
...

Low-level Definitions

Low-level Definitions

- Given t boolean variables $p_1, \ldots, p_t$:

Low-level Definitions

- Given t boolean variables $p_1, \ldots, p_t$:
- A vector is an assignment to each of the t variables one of $\{0, 1, *\}$.
  - $*$ means "undetermined."

Low-level Definitions

- Given t boolean variables $p_1, \ldots, p_t$:
- A vector is an assignment to each of the t variables one of $\{0, 1, *\}$.
  - * means "undetermined."
- Function (**concept**) $\mathcal{F}$ maps all vectors to $\{0, 1\}$.
  - Learning machine is learning concepts.
  - The variables used in $\mathcal{F}$ are *determined* in $\mathcal{F}$.

Low-level Definitions

- Given t boolean variables $p_1, \ldots, p_t$:
- A vector is an assignment to each of the t variables one of {0,1,*}.
  - * means "undetermined."
- Function (**concept**) $\mathcal{F}$ maps all vectors to {0,1}.
  - Learning machine is learning concepts.
  - The variables used in $\mathcal{F}$ are *determined* in $\mathcal{F}$.

Low-… …tions

- Given t boo… $p_1, \ldots, p_t$:
- A vector is a… …o each of the t variables one of $\{0,1,*\}$.
  - $*$ means “undetermined.”
- Function (**concept**) $\mathcal{F}$ maps all vectors to $\{0,1\}$.
  - Learning machine is learning concepts.
  - The variables used in $\mathcal{F}$ are *determined* in $\mathcal{F}$.

Low-… …tions

- Given t boo… $p_1, \dots, p_t$:
- A **vector** is a… assignment to each of the t v…
  - •
- Fu… s to {0,1}.
  - Learning machine is learning concepts.
  - The variables used in $\mathcal{F}$ are *determined* in $\mathcal{F}$.

- Variables: {walks like a duck, beak, purple, …}
- Vector ***v***: {walks_like_a_duck=0, beak=1, purple=*, …}
- $\mathcal{F}(v)$ = ***is_a_duck(v)*** = ***false***

Low-level Definitions

- Given t boolean variables $p_1, \ldots, p_t$:
- A vector is an assignment to each of the t variables one of $\{0, 1, *\}$.
  - Variables: {purple, walks like a duck, beak, ...}
  - Vector v: {purple=*, walks_like_a_duck=0, beak=1 ...}

Variables determined in v: {walks_like_a_duck, beak}

to $\{0, 1\}$.

  - Learning machine is learning concepts.
  - The variables used in $\mathcal{F}$ are *determined* in $\mathcal{F}$.

Low-level Definitions

- Given t boolean variables $p_1, \ldots, p_t$:
- A *vector* is an assignment to each of the t v…

- Variables: {purple, walks like a duck, beak, …}

Variables determined in is_a_duck: {walks_like_a_duck, quacks_like_a_duck}

- $\mathcal{F}$(v) = is_a_duck(v) = false

- Fu… s to {0, 1}.
  - Learning machine is learning concepts.
  - The variables used in $\mathcal{F}$ are *determined* in $\mathcal{F}$.

Low-level Definitions

- Given t boolean variables $p_1, \dots, p_t$:
- A vector is an assignment to each of the t v

- t boolean variables $p_1, \dots, p_t$
- Vectors assign variables to one of {0,1,*}.
- Concept $\mathcal{F}$ maps vectors to {0,1}.

- Fu… s to {0,1}.
  - Learning machine is learning concepts.
  - The variables used in $\mathcal{F}$ are *determined* in $\mathcal{F}$.

- t boolean variables $p_1, \ldots, p_t$:
- Vectors assign variables to one of $\{0, 1, *\}$.
- Concept $\mathcal{F}$ maps vectors to $\{0, 1\}$.

- t boolean variables $p_1, \ldots, p_t$:
- Vectors assign variables to one of $\{0, 1, *\}$.
- Concept $\mathcal{F}$ maps vectors to $\{0, 1\}$.

- Assume $\mathcal{D}$, a probability distribution over all vectors $\boldsymbol{v}$ which $\mathcal{F}$ evaluates to 1.

- t boolean variables $p_1, \ldots, p_t$:
- Vectors assign variables to one of $\{0, 1, *\}$.
- Concept $\mathcal{F}$ maps vectors to $\{0, 1\}$.

- Assume $\mathcal{D}$, a probability distribution over all vectors $\boldsymbol{v}$ which $\mathcal{F}$ evaluates to 1.
- $\mathcal{D}$ is meant to describe the relative natural frequency of positive examples of whatever we're trying to learn.

- t boolean variables $p_1, \ldots, p_t$:
- Vectors assign variables to one of $\{0, 1, *\}$.
- Concept $\mathcal{F}$ maps vectors to $\{0, 1\}$.

- Assume $\mathcal{D}$, a probability distribution over all vectors $\boldsymbol{v}$ which $\mathcal{F}$ evaluates to 1.
- $\mathcal{D}$ is meant to describe the relative natural frequency of positive examples of whatever we're trying to learn.
- If we have a vector $\boldsymbol{v}$ that describes a mallard, then $\mathcal{D}(\boldsymbol{v})$ = relative frequency of mallards in the duck population.

- t boolean variables $p_1, \ldots, p_t$:
- Vectors assign variables to one of $\{0, 1, *\}$.
- Concept $\mathcal{F}$ maps vectors to $\{0, 1\}$.

- Assume $\mathcal{D}$, a probability distribution over all vectors $\boldsymbol{v}$ which $\mathcal{F}$ evaluates to 1.
- $\mathcal{D}$ is meant to describe the relative natural frequency of positive examples of whatever we're trying to learn.

- Probability distribution $\mathcal{D}$ over all true vectors v.

- If we have a vector $\boldsymbol{v}$ that describes a mallard, then $\mathcal{D}(\boldsymbol{v})$ = relative frequency of mallards in the universe.

- t boolean variables $p_1, \dots, p_t$:
- Vectors assign variables to one of $\{0, 1, *\}$.
- Concept $\mathcal{F}$ mapping vectors to $\{0, 1\}$.
- Probability distribution $\mathcal{D}$ over all true v.

- t boolean variables $p_1, \ldots, p_t$:
- Vectors assign variables to one of $\{0, 1, *\}$.
- Concept $\mathcal{F}$ mapping vectors to $\{0, 1\}$.
- Probability distribution $\mathcal{D}$ over all true v.

High-level Definitions.

High-level Definitions.

- A learning machine has two components:

High-level Definitions.

- A learning machine has two components:
  - A learning protocol, or the method by which information is gathered from the world.

High-level Definitions.

- A learning machine has two components:
  - A learning protocol, or the method by which information is gathered from the world.
  - A deduction procedure, or the mechanism for learning new concepts from gathered information.

VALIANT’S LEARNING PROTOCOL

- t boolean variables $p_1, \ldots, p_t$:
- Vectors assign variables to one of {0,1,*}.
- Concept $\mathcal{F}$ mapping vectors to {0,1}.
- Probability distribution $\mathcal{D}$ over all true v.

VALIANT'S LEARNING PROTOCOL

- t boolean variables $p_1, \ldots, p_t$:
- Vectors assign variables to one of $\{0, 1, *\}$.
- Concept $\mathcal{F}$ mapping vectors to $\{0, 1\}$.
- Probability distribution $\mathcal{D}$ over all true v.

- t boolean variables $p_1, \ldots, p_t$:
- Vectors assign variables to one of $\{0,1,*\}$.
- Concept $\mathcal{F}$ mapping vectors to $\{0,1\}$.
- Probability distribution $\mathcal{D}$ over all true v.

- Learner has access to two routines (or teachers):

- t boolean variables $p_1, \dots, p_t$:
- Vectors assign variables to one of $\{0, 1, *\}$.
- Concept $\mathcal{F}$ mapping vectors to $\{0, 1\}$.
- Probability distribution $\mathcal{D}$ over all true v.

- Learner has access to two routines (or teachers):

1. EXAMPLE: takes no input, returns a vector v such that $\mathcal{F}(v) = 1$.
   - Probability that EXAMPLE returns any particular v is $\mathcal{D}(v)$.

- t boolean variables $p_1, \ldots, p_t$:
- Vectors assign variables to one of $\{0, 1, *\}$.
- Concept $\mathcal{F}$ mapping vectors to $\{0, 1\}$.
- Probability distribution $\mathcal{D}$ over all true v.

- Learner has access to two routines (or teachers):

1. EXAMPLE: takes no input, returns a vector v such that $\mathcal{F}(v) = 1$.
   - Probability that EXAMPLE returns any particular v is $\mathcal{D}(v)$.
2. ORACLE: takes as input a vector v, returns $\mathcal{F}(v)$.

Duck Example of Protocol Functions

Duck Example of Protocol Functions

$$\mathcal{F}(\boldsymbol{v}) = \texttt{is\_a\_duck}(\boldsymbol{v})$$

Duck Example of Protocol Functions

$$\mathcal{F}(\boldsymbol{v}) = \texttt{is\_a\_duck}(\boldsymbol{v})$$

EXAMPLE()

24

Duck Example of Protocol Functions

$\mathcal{F}(\boldsymbol{v}) = \texttt{is\_a\_duck}(\boldsymbol{v})$	
EXAMPLE()	

24

Duck Example of Protocol Functions

$$\mathcal{F}(\boldsymbol{v}) = \texttt{is\_a\_duck}(\boldsymbol{v})$$

EXAMPLE()	
ORACLE()	

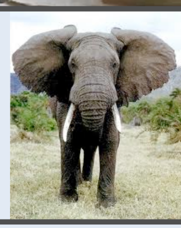
24

Duck Example of Protocol Functions

$\mathcal{F}(\boldsymbol{v}) = \texttt{is\_a\_duck}(\boldsymbol{v})$	
EXAMPLE()	
ORACLE()	TRUE

24

Duck Example of Protocol Functions

$\mathcal{F}(v) = \texttt{is\_a\_duck}(v)$	
EXAMPLE()	
ORACLE()	TRUE
ORACLE()	

24

Duck Example of Protocol Functions

$\mathcal{F}(v) = \texttt{is\_a\_duck}(v)$		
EXAMPLE()		
ORACLE()		TRUE
ORACLE()		FALSE

Realistic Example of Protocol Functions

Realistic Example of Protocol Functions

$$\mathcal{F} = (\mathtt{a1} \lor \mathtt{a2}) \land (\mathtt{a4} \lor \mathtt{a1})$$

Realistic Example of Protocol Functions

$$\mathcal{F} = \texttt{(a1} \lor \texttt{a2)} \land \texttt{(a4} \lor \texttt{a1)}$$

EXAMPLE()

25

Realistic Example of Protocol Functions

$\mathcal{F}$ = (a1 ∨ a2) ∧ (a4 ∨ a1)	
EXAMPLE()	{a1=1, a2=0, a3=*}

Realistic Example of Protocol Functions

$\mathcal{F}$ = (a1 ∨ a2) ∧ (a4 ∨ a1)	
EXAMPLE()	{a1=1, a2=0, a3=*}
EXAMPLE()	

25

Realistic Example of Protocol Functions

$\mathcal{F} = (a1 \lor a2) \land (a4 \lor a1)$	
EXAMPLE()	`{a1=1, a2=0, a3=*}`
EXAMPLE()	`{a1=0, a2=1, a3=*, a4=1}`

Realistic Example of Protocol Functions

$\mathcal{F}$ = `(a1 ∨ a2) ∧ (a4 ∨ a1)`	
EXAMPLE()	`{a1=1, a2=0, a3=*}`
EXAMPLE()	`{a1=0, a2=1, a3=*, a4=1}`
ORACLE(`{a1=0,a2=0,a3=*,a4=1}`)	

Realistic Example of Protocol Functions

$\mathcal{F}$ = (a1 ∨ a2) ∧ (a4 ∨ a1)	
EXAMPLE()	{a1=1, a2=0, a3=*}
EXAMPLE()	{a1=0, a2=1, a3=*, a4=1}
ORACLE({a1=0,a2=0,a3=*,a4=1})	FALSE

25

Probably Approximately Learnable

Probably Approximately Learnable

- A class of problems is **Probably Approximately Learnable** if instances of the problem can be learned by a deduction algorithm that:

Probably Approximately Learnable

- A class of problems is **Probably Approximately Learnable** if instances of the problem can be learned by a deduction algorithm that:
  - Uses this protocol.

Probably Approximately Learnable

- A class of problems is **Probably Approximately Learnable** if instances of the problem can be learned by a deduction algorithm that:
  - Uses this protocol.
  - Runs in reasonable time: polynomial by adjustable parameter h, size of learned program, and number of variables determined in the learned formula.

Probably Approximately Learnable

- A class of problems is **Probably Approximately Learnable** if instances of the problem can be learned by a deduction algorithm that:
  - Uses this protocol.
  - Runs in reasonable time: polynomial by adjustable parameter h, size of learned program, and number of variables determined in the learned formula.
  - Produces a program that says something is false when it's true with probability no greater than $(1-h^{-1})$; never says that something is true when it's false.

A Summary, in English

A Summary, in English

- We are trying to make a program (learning machine) that can learn, in polynomial time, another program (the learned program) that recognizes whether a boolean formula (concept) is true for any set of boolean data.

A Summary, in English

- We are trying to make a program (learning machine) that can learn, in polynomial time, another program (the learned program) that recognizes whether a boolean formula (concept) is true for any set of boolean data.
- The learning program has access to a function that will give it a bunch of examples, as well as a function that will check its work.

A Summary, in English

- We are trying to make a program (learning machine) that can learn, in polynomial time, another program (the learned program) that recognizes whether a boolean formula (concept) is true for any set of boolean data.
- The learning program has access to a function that will give it a bunch of examples, as well as a function that will check its work.
- The learning machine can learn a program that is sometimes wrong, so long as the probability that the learned program is ever wrong is adjustable.

Outline

1. General framework for defining Learning Machines, or programs that can learn/write/produce other programs of a particular type.
   - A Learning Machine for animal recognition, for example, might learn to write a program that recognizes whether a given animal is a duck.
2. Definition of a particular learning protocol.
3. Definition of when a program class is reasonably-learnable.
4. Definition/proofs of reasonably-learnable program classes.

Outline

1. General framework for defining Learning Machines, or programs that can learn/write/produce other programs of a particular type.
   - A Learning Machine for animal recognition, for example, might learn to write a program that recognizes whether a given animal is a duck.
2. Definition of a particular learning protocol.
3. Definition of when a program class is reasonably-learnable.
4. Definition/proofs of reasonably-learnable program classes.

Outline

1. General framework for defining Learning Machines,

- The paper proves three different program classes probably-approximately-learnable.

2. D
3. D
4. D

Outline

1. General framework for defining Learning Machines,

- The paper proves three different program classes probably-approximately-learnable.
- I am not going to walk through the proofs; they are by construction of deduction algorithms that can learn the given programs and proofs of their bounds.

2. D
3. D
4. D

Outline

1. General framework for defining Learning Machines,

- The paper proves three different program classes probably-approximately-learnable.
- I am not going to walk through the proofs; they are by construction of deduction algorithms that can learn the given programs and proofs of their bounds.
- I am going to give the upper bounds of the algorithms. This requires a definition of a function.

29

Outline

1. General framework for defining Learning Machines,

- The paper proves three different program classes probably-approximately-learnable.
- I am not going to walk through the proofs; they are by construction of deduction algorithms that can learn the given programs and proofs of their bounds.
- I am going to give the upper bounds of the algorithms. This requires a definition of a function.
- The proof of that function's upper bound is the major lemma in all three proofs, so I will outline it.

Outline

1. General framework for defining Learning Machines,

- The paper proves three different program classes probably-approximately-learnable.
- I am not going to walk through the proofs; they are by construction of deduction algorithms that can learn the given programs and proofs of their bounds.
- I am going to give the upper bounds of the algorithms. This requires a definition of a function.
- The proof of that function's upper bound is the major lemma in all three proofs, so I will outline it.
- This means the next 3 slides are mathy.

A Combinatorial Bound

A Combinatorial Bound

- $L(h, S)$ is a function defined for all real numbers $h > 1$ and integers $S > 1$.

A Combinatorial Bound

- $L(h,S)$ is a function defined for all real numbers $h > 1$ and integers $S > 1$.
- Returns smallest integer n such that in n independent Bernoulli trials, each with probability at least h^{-1} of success, P(< S successes) < h^{-1}
  - Bernoulli trial: an experiment whose outcomes are either "success" or "failure"; randomly distributed by some probability function.

Upper Bound on `L`(h, S)

Upper Bound on $\mathtt{L}(h,S)$

$$\mathtt{L}(h,S) \le 2h(S + \mathtt{log}_e h)$$

Upper Bound on $\mathtt{L}(h, S)$

$$\mathtt{L}(h, S) \le 2h(S + \log_e h)$$

Proof by algebraic substitution of well-known inequalities:

Upper Bound on $\mathtt{L}(h, S)$

$$\mathtt{L}(h, S) \le 2h(S + \log_e h)$$

Proof by algebraic substitution of well-known inequalities:

1. $\forall x > 0,\ (1 + x^{-1})^{x} < e$

Upper Bound on $\mathtt{L}(h,S)$

$$\mathtt{L}(h,S) \leq 2h(S + \log_e h)$$

Proof by algebraic substitution of well-known inequalities:

1. $\forall x > 0,\ (1 + x^{-1})^x < e$
2. $\forall x > 0,\ (1 - x^{-1})^x < e^{-1}$

Upper Bound on $L(h, S)$

$$L(h,S) \leq 2h(S + \log_e h)$$

Proof by algebraic substitution of well-known inequalities:

1. $\forall x > 0, \ (1 + x^{-1})^{x} < e$
2. $\forall x > 0, \ (1 - x^{-1})^{x} < e^{-1}$
3. In m independent trials, each with success probability $\geq p$:

$P(\text{successes at most } k) \leq \left(\frac{m-mp}{m-k}\right)^{m-k} \left(\frac{mp}{k}\right)^{k}$

So?

So?

- `L(`h`,`S`)` is basically linear in both h and S.

So?

- $\mathtt{L}(h, S)$ is basically linear in both h and S.
- Applies to using EXAMPLEs and ORACLE to determine vectors.

So?

- $\mathbf{L}(\boldsymbol{h}, \boldsymbol{S})$ is basically linear in both $\boldsymbol{h}$ and $\boldsymbol{S}$.
- Applies to using EXAMPLEs and ORACLE to determine vectors.
- An algorithm can approximate the set of determined variables in natural EXAMPLEs of $\mathcal{F}$ in runtime independent of *total* number of variables in the world.

So?

- $\mathbf{L}(\boldsymbol{h}, \boldsymbol{S})$ is basically linear in both $\boldsymbol{h}$ and $\boldsymbol{S}$.
- Applies to using EXAMPLEs and ORACLE to determine vectors.
- An algorithm can approximate the set of determined variables in natural EXAMPLEs of $\mathcal{F}$ in runtime independent of *total* number of variables in the world.
  - Dependent only the number of variables that are determined in $\mathcal{F}$.

Remaining Question

Given that learning protocol, what classes of tasks are learnable in polynomial time?

Answer: At Least 3 Classes of Programs

Answer: At Least 3 Classes of Programs

1. *k*-CNF expressions

Answer: At Least 3 Classes of Programs

1. *k*-CNF expressions
2. Monotone DNF expressions

Answer: At Least 3 Classes of Programs

1. *k*-CNF expressions
2. Monotone DNF expressions
3. μ-expressions

k–CNF Expressions

k–CNF Expressions

- Conjunctive Normal form (CNF):

  $(a_1 \lor a_2 \lor a_3) \land (a_4 \lor a_1) \ldots$

k–CNF Expressions

- Conjunctive Normal form (CNF):

  $(a_1 \lor a_2 \lor a_3) \land (a_4 \lor a_1) \ldots$

- *k*-CNF expression: a CNF expression where each internal clause is composed of $\leq$ ***k*** literals.

k–CNF Expressions

- Conjunctive Normal form (CNF):

  $(a_1 \lor a_2 \lor a_3) \land (a_4 \lor a_1) \ldots$

- *k*-CNF expression: a CNF expression where each internal clause is composed of $\leq k$ literals.
- Learnable with an algorithm that does not call ORACLE, and calls EXAMPLE $\leq L(h, 2t^{k+1})$ times. *(t is the number of variables)*

Monotone DNF Expressions

Monotone DNF Expressions

- Disjunctive Normal Form (DNF):

  $(a_1 \wedge a_2 \wedge a_3) \vee (a_1 \wedge a_4) \ldots$

Monotone DNF Expressions

- Disjunctive Normal Form (DNF):

  $(a_1 \land a_2 \land a_3) \lor (a_1 \land a_4) \ldots$

- An expression is monotone if it contains no negated literals.

Monotone DNF Expressions

- Disjunctive Normal Form (DNF):

  $(a_1 \wedge a_2 \wedge a_3) \vee (a_1 \wedge a_4) \ldots$

- An expression is monotone if it contains no negated literals.
- Learnable with an algorithm that calls EXAMPLEs $L = L(h,d)$ times and ORACLEs $d*t$ times, where d is the degree of the expression and t is the number of variables.

μ-expressions

μ-expressions

- General expression over $\{p_1, \ldots, p_t\}$ defined recursively $(1 \le i \le t)$:

$$f := p_i \mid \sim p_i \mid f_1 \wedge f_2 \mid f_1 \vee f_2$$

μ-expressions

- General expression over $\{p_1, \ldots, p_t\}$ defined recursively $(1 \leq i \leq t)$:

$$f := p_i \mid \sim p_i \mid f_1 \wedge f_2 \mid f_1 \vee f_2$$

- A μ-expression is an expression in which each p appears at most once.

μ-expressions

- General expression over $\{p_1,\dots,p_t\}$ defined recursively $(1 \le i \le t)$:

$$f := p_i \mid \sim p_i \mid f_1 \wedge f_2 \mid f_1 \vee f_2$$

- A μ-expression is an expression in which each p appears at most once.
- Learnable with an exactly correct algorithm that calls two slightly more powerful ORACLE functions $O(t^3)$ times total.

Executive Summary

Executive Summary

- Learnability theory is concerned with what programs can be learned automatically.

Executive Summary

- Learnability theory is concerned with what programs can be learned automatically.
- We should reason about what is programmatically learnable in the same way we reason about what is computable.

Executive Summary

- Learnability theory is concerned with what programs can be learned automatically.
- We should reason about what is programmatically learnable in the same way we reason about what is computable.
- A class of programs is Probably Approximately Learnable when, using a particular type of teacher, a given algorithm can learn a program that can recognize instances of that class with a certain probability.

Executive Summary

- Learnability theory is concerned with what programs can be learned automatically.
- We should reason about what is programmatically learnable in the same way we reason about what is computable.
- A class of programs is Probably Approximately Learnable when, using a particular type of teacher, a given algorithm can learn a program that can recognize instances of that class with a certain probability.
- 3 examples of such learnable program types are k-CNF expressions, monotone DNF expressions, and μ-expressions.

Interesting Concluding Questions

- What else is learnable by these definitions?
- Is the definition of “learnable” reasonable?
  - How powerful should the teachers be?
  - What about if we use negative in addition to positive examples?
- How do humans learn?

- Assume we have urn that contains many marbles of s different types. We want to “learn” the different types of marbles by taking a small random sample x, of size sufficient that, with high probability, it contains at least 99% of s marble type representatives.

- Assume we have urn that contains many marbles of ***s*** different types. We want to "learn" the different types of marbles by taking a small random sample ***x***, of size sufficient that, with high probability, it contains at least 99% of ***s*** marble type representatives.
- Definition of `L(h,S)` implies:

`|X| = L(100,S) ⇒ P(succeeded overall) > 99%`

- Assume we have urn that contains many marbles of ***s*** different types. We want to "learn" the different types of marbles by taking a small random sample ***X***, of size sufficient that, with high probability, it contains at least 99% of ***s*** marble type representatives.
- Definition of `L(h,S)` implies:

  `|X| = L(100,S) ⇒ P(succeeded overall) > 99%`

- "Success" for each trial is defined as picking a marble we haven't picked before. Success clearly depends on previous choices, but the probability of each success will always be at least 1%, **independent of previous choices.**