Probabilistic Inference by Hashing and Optimization

Focusing on Approximate Model Counting

Stefano Ermon
slides by Dor Cohen

June 13, 2017

Introduction

- Many problems in ML and Statistics involve computation of high-dimensional integrals (e.g. evaluate posterior probabilities)
- Computing expectations $w.r.t$ high dimensional probability distributions known to be intractable in worst-case (Roth, 2016)
- Difficulty arises as number of possible states scales exponentially - *curse of dimensionality* (Belmman, 1996)

Standard approaches

Focusing on *discrete* probability distributions and *approximately* computing expectations, there are two main standard approaches:

- **Monte Carlo** sampling techniques (1950s): Approximate complex distributions with small number of representative samples
- **Variational** methods: Approximate complex models using families of tractable distributions (e.g., estimate $P(Z|X) \approx Q(Z)$ and compare distributions using $KL - divergence$)

These don't provide tight guarantees on accuracy

MCMC - Monte Carlo Markov Chain

Goal: Sample from a *discrete* distributions vector $\pi = (\pi_1, ...\pi_n)$
Idea:

- Build a Markov chain with n states such that it will be *ergodic*
- Choose weights for graph edges (probabilities) such that the chain will be *invertible* $w.r.t$ to the desirable distribution π
- *ergodic* and *invertible* chain imply that the "last" state of long random walk on the chain is distributed like π

MCMC - Monte Carlo Markov Chain

Key difficulty

is to to draw proper samples

- Needs to set up a Markov Chain over entire state space
- Has to reach equilibrium distribution - requires exponential time (Madras, 2002)
- In practice will only give approximate answer. Chain may "trap" in less relevant areas of the state space

Convergence of *Metropolis – Hastings*

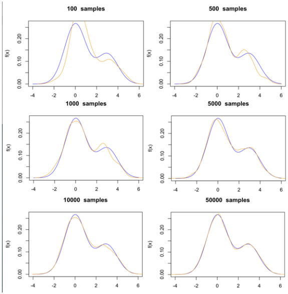


Figure: MCMC attempts to approximate blue distribution with orange one [Wiki]

A new approach

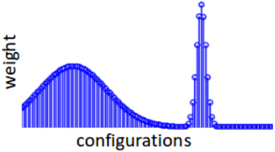
Approximate inference/counting algorithms based on randomized hashing and optimizations:

- *provably accurate results* assuming access to an **optimization oracle**
- Can compute parity functions, marginal probabilities providing an approximately correct answer
- Specifically, within a factor of $1 + \epsilon$ of the true value for any desired ϵ , with high probability (at least $1 - \lambda$ for any desired $\lambda > 0$)

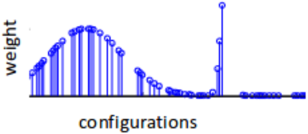
Hashing and optimization

Statistics are still computed using small subset of samples. However these samples aren't drawn randomly from the distribution using an MC, but:

- Samples are obtained by random projection of original high-dimensional states to lower-dimensional ones using *Universal hash functions*
- Then look for "Most likely" configuration in the projected subspace (current optimization tools can handle millions of variables)



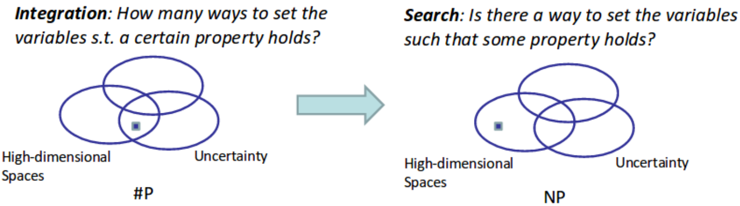
$\longrightarrow$



Complexity perspective

Inference or counting problems we consider are in $\#P$, believed to be significantly harder than *NP* (Valiant, 1979)

- **If we allow small failure probability, and small error** we can *reduce* the $\#P$ problem to smaller number of instances of a *NP-equivalent* (combinatorial) optimization problem



Problem statement - setup

- Let $\sum$ be a large but finite set (e.g. the set of all possible assignments in a model) and let $w : \sum \to \mathbb{R}^+$ be a non-negative function that assigns a weight for each configuration (an element of $\sum$)
- We are given 2^n configurations, and non-negative weights w (Bayes net, Factor graph, weighted CNF)
- **Goal:** (approximately) compute the total weight of the set, defined as:

$$W = \sum_{\sigma \in \sum} w(\sigma)$$

- **Example:** $n = 100$ variables, sum over 2^{100} configurations

Problem statement - setup

Assumption: We assume that we have access to an *optimization oracle* that can solve the following optimization problem:

$$\max_{\sigma \in \sum} w(\sigma)\mathbb{1}_{\{o\}}(\sigma)$$

Where $\mathbb{1}_{\{o\}} : \sum \to \{0,1\}$ is an indicator function for compactly represented subset $o \subseteq \sum$, i.e., $\mathbb{1}_{\{o\}} = 1$ if $\sigma \in o$ and 0 otherwise.

- o may be compactly represented using a smaller subset of constraints

SAT

- A boolean formula φ is said to be in *CNF* if its a logical conjunction of a set of clauses $C_1, \ldots C_n$, where each clause C is a logical disjunction of a set of literals. e.g., $(x_1 \vee \neg x_2)$
- *SAT*: deciding if there exists an assignment that **satisfies** φ

Example: $\varphi = (x_1 \vee \neg x_2 \vee x_3) \wedge (x_4 \vee \neg x_1) \wedge (x_2 \vee \neg x_3)$

Satisfying assignment: $\{x_1 = 1, x_2 = 1, x_3 = 1, x_4 = 1\}$

$$\implies \varphi \text{ is SATISFIABLE}$$

Model counting

- Let V be the set of boolean variables of φ , $|V| = n$, and let $\sigma = \{0,1\}^n$ be the set of all possible assignments to these variables
- An assignment $\sigma \in \sum$ is a mapping that assigns a value in $\{0,1\}$ to each variable in V
- Define the weight $w(\sigma)$ to be 1 if φ is satisfied by σ and 0 otherwise

In this context:

$$W = \sum_{\sigma \in \sum} w(\sigma) = \#(\varphi)$$

Approximate model counting

- Problem of *approximately* counting the number of solutions, assuming access to an *NP* oracle (e.g. SAT solver) was first considered by Stockmeyer (1985) - Important result he established: $\#P$ can be approximated in BPP^{NP} [1]

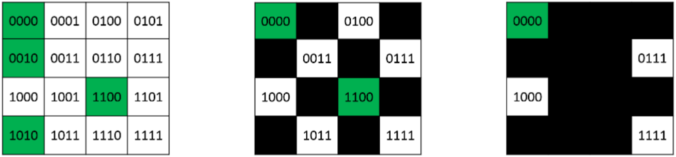
Intuition behind his algorithm:

- Let $S \subseteq \sum$ be the set of solutions to φ
- Reliably estimate $|S|$ by **repeating** the following process
  1. Randomly partition $\sum$ into 2^m cells
  2. Select one of the cells
  3. Compute if S has at least one element in this cell (invoke a SAT solver)

1

[1] algorithms that have bounded-error probabilistic polynomial time and access to an *NP* oracle

Approximate model counting - Example



- **Example:** $\sum = \{0,1\}^4$, vars: x_1, x_2, x_3, x_4 .
  $Sols = \{(0,0,0,0),(0,0,1,0),(1,1,0,0),(1,0,1,0)\}$.
- **Middle:** matrix partitioned into 2 cells, based on the parity of $x_3 \oplus x_4$ - two solutions are left, corresponding to the assignments satisfying $x_3 \oplus x_4 = 0$.
- **Right:** after partitioning again based on the parity of $x_2 \oplus x_3$, only solution $(0,0,0,0)$ is left.

Approximate model counting

To summarize:

- Estimate $|S|$, by defining progressively smaller cells, until no element of S can be found inside a randomly chosen cell
- The larger $|S|$ is, the smaller the cells have to be
- **Correctness** of the algorithm relies crucially on how the space is randomly partitioned into cells
- To achieve strong worst-case (probabilistic) guarantees, algorithm relies on *universal hash functions* (Vadhan, 2011; Goldreich, 2011)

Universal hash functions

Definition: A family of functions $\mathcal{H} = \{h : \{0,1\}^n \mapsto \{0,1\}^m\}$ is *strongly universal* (pairwise independent) if the following two conditions hold when h is chosen uniformly at random from $\mathcal{H}$:

1. $\forall x \in \{0,1\}^n$, the random variable $H(x)$ is uniformly distributed in $\{0,1\}^m$
2. $\forall x_1, x_2 \in \{0,1\}^n | \ x_1 \neq x_2$, the random variables $H(x_1)$ and $H(x_2)$ are independent

Universal hash functions

- Considering the set of all possible functions from $\{0,1\}^n$ to $\{0,1\}^m$ we establish statistically optimal functions. Easy to verify this is a family of *fully* independent functions
- However, functions like these require $m2^n$ bits for specifying $\rightarrow$ not useful for large n
- *pairwise independent* hash functions can be specified compactly, with number of bits linear in n

Pairwise independent hash functions

- Generally, these functions are based on modular arithmetic constraints of the form $Ax = b \bmod 2$. Implies that Ax is congruent to b modulo 2

Proposition: Let $A \in \{0,1\}^{mxn}$, $b \in \{0,1\}^m$. The family $\mathcal{H} = \{h_{A,b}(x) : \{0,1\}^n \mapsto \{0,1\}^m\}$ where $h_{A,b}(x) = Ax + b \bmod 2$ is a family of pairwise independent hash functions.

Algorithm: ApproxModelCount

Algorithm 9.1 ApproxModelCount $(F, \mathcal{O}_S, \Delta)$

Let S denote the set of solutions to the input formula F
$T \leftarrow \left\lceil \frac{\log(1/\Delta)}{\alpha} \log n \right\rceil$
$i = 0$
while $i \leq n$ **do**
    for $t = 1, \cdots, T$ **do**
        Sample hash function $h^i_{A,b} : \Sigma \to \{0,1\}^i$, i.e.
            sample uniformly $A \in \{0,1\}^{i \times n}$, $b \in \{0,1\}^i$
        Let $S(h^i_{A,b}) = |\{x \in S \mid Ax \equiv b \pmod 2\}|$
        $w^t_i \leftarrow \mathbb{I}[S(h^i_{A,b}) \geq 1]$, using $\mathcal{O}_S$ to check if $\{x \in S \mid Ax \equiv b \pmod 2\}$ is empty
    end for
    if Median$(w^1_i, \cdots, w^T_i) < 1$ **then**
        break
    end if
    $i = i + 1$
end while
Return $\lfloor 2^{i-1} \rfloor$

Algorithm: ApproxModelCount

Based on pairwise independence, possible to show that ApproxModelCount provides with high probability an accurate estimate of the true value, summarized by the following Theorem:

Theorem: For any $\Delta > 0$, positive constant $a \leq 0.0042$, ApproxModelCount makes $\Theta(nlnnln1/\delta)$ queries to the NP oracle O_S and, with probability of at least $(1 - \Delta)$, outputs a 16-approximation of $|S|$, the number of solutions of a formula φ

Theorem proof sketch

By the uniformity property of *Universal hash functions*: Given any solution $x \in S$, we can see that in iteration i it holds that: $P[Ax = b(mod2)] = (\frac{1}{2})^i$

Algorithm 9.1 ApproxModelCount $(F, \mathcal{O}_S, \Delta)$

```
Let S denote the set of solutions to the input formula F
T ← ⌈ (log(1/Δ)/α) log n ⌉
i = 0
while i ≤ n do
    for t = 1, ··· , T do
        Sample hash function h^i_{A,b} : Σ → {0,1}^i, i.e.
            sample uniformly A ∈ {0,1}^{i×n}, b ∈ {0,1}^i
        Let S(h^i_{A,b}) = |{x ∈ S | Ax ≡ b (mod 2)}|
        w^t_i ← I[S(h^i_{A,b}) ≥ 1], using O_S to check if {x ∈ S | Ax ≡ b
    end for
    if Median(w^1_i, ··· , w^T_i) < 1 then
        break
    end if
    i = i + 1
end while
Return ⌊2^{i−1}⌋
```

Theorem proof sketch

By linearity of expectation: $E[S(h^i_{A,b})] = \frac{|S|}{2^i}$. It implies that *in expectation* the while loop should break for $i \approx log|S|$ i.e., when the corresponding expected number of "surviving" solutions is less than 1. When this happens, the algorithm provides an accurate estimate for $|S|$

Algorithm 9.1 ApproxModelCount $(F, \mathcal{O}_S, \Delta)$

Let S denote the set of solutions to the input formula F
$T \leftarrow \left\lceil \frac{\log(1/\Delta)}{\alpha} \log n \right\rceil$
$i = 0$
while $i \leq n$ **do**
  for $t = 1, \cdots, T$ **do**
    Sample hash function $h^i_{A,b} : \Sigma \rightarrow \{0,1\}^i$, i.e.
      sample uniformly $A \in \{0,1\}^{i \times n}$, $b \in \{0,1\}^i$
    Let $S(h^i_{A,b}) = |\{x \in S \mid Ax \equiv b \pmod 2\}|$
    $w^t_i \leftarrow \mathbb{I}[S(h^i_{A,b}) \geq 1]$, using $\mathcal{O}_S$ to check if $\{x \in S \mid Ax \equiv b$
  end for
  if Median$(w^1_i, \cdots, w^T_i) < 1$ **then**
    break
  end if
  $i = i + 1$
end while
Return $\lfloor 2^{i-1} \rfloor$

Theorem proof sketch

Because the hash functions family is pairwise independent, $S(h^i_{A,b})$ is the sum of pairwise independent random variables (one for each element, corresponding to whether that solution satisfies the random constraints)

Algorithm 9.1 ApproxModelCount $(F, \mathcal{O}_S, \Delta)$

Let S denote the set of solutions to the input formula F
$T \leftarrow \left\lceil \frac{\log(1/\Delta)}{\alpha} \log n \right\rceil$
$i = 0$
while $i \leq n$ **do**
  for $t = 1, \cdots, T$ **do**
    Sample hash function $h^i_{A,b} : \Sigma \rightarrow \{0,1\}^i$, i.e.
      sample uniformly $A \in \{0,1\}^{i \times n}$, $b \in \{0,1\}^i$
    Let $S(h^i_{A,b}) = |\{x \in S \mid Ax \equiv b \pmod 2\}|$
    $w^t_i \leftarrow \mathbb{I}[S(h^i_{A,b}) \geq 1]$, using $\mathcal{O}_S$ to check if $\{x \in S \mid Ax \equiv b$
  end for
  if Median$(w^1_i, \cdots, w^T_i) < 1$ **then**
    break
  end if
  $i = i + 1$
end while
Return $\lfloor 2^{i-1} \rfloor$

Theorem proof sketch

Therefore, the variance of $S(h^i_{A,b})$ is the sum of **individual** variances. And can be shown to be "small" compared to the mean.

By standard concentration inequalities, $S(h^i_{A,b})$ takes values close to it's mean reasonably often.

Computing median over T runs, guarantees an accurate estimate with high probability.

Algorithm 9.1 ApproxModelCount $(F, \mathcal{O}_S, \Delta)$

```
Let S denote the set of solutions to the input formula F
T ← ⌈ (log(1/Δ)/α) log n ⌉
i = 0
while i ≤ n do
    for t = 1, ⋯ , T do
        Sample hash function h^i_{A,b} : Σ → {0,1}^i, i.e.
            sample uniformly A ∈ {0,1}^{i×n}, b ∈ {0,1}^i
        Let S(h^i_{A,b}) = |{x ∈ S | Ax ≡ b (mod 2)}|
        w^t_i ← 𝕀[S(h^i_{A,b}) ≥ 1], using 𝒪_S to check if {x ∈ S | Ax ≡ b
    end for
    if Median(w^1_i, ⋯ , w^T_i) < 1 then
        break
    end if
    i = i + 1
end while
Return ⌊2^{i−1}⌋
```

Improving the approximation factor

Given the k-approximation algorithm and any $\epsilon > 0$, it's possible to design a $(1+\epsilon)$-approximation algorithm, with the following construction:

Let $l = log_{1+\epsilon}k$, define new set of configurations $\sum^l = \sum x \sum ... x \sum$, and a new weight function $w' : \sum^l \mapsto \mathcal{R}$ as $w'(\sigma_1, ..., \sigma_l) = w(\sigma_1)w(\sigma_2)...w(\sigma_l)$

Idea is to estimate the size of $S(h^i_{A,b})$ with multiple calls to an NP-oracle , e.g. check if $S(h^i_{A,b})$ contains at least k elements $\rightarrow$ Reduces the variance, can be used to improve accuracy, but requires more calls to the NP-oracle

Compactly representing constraints

- In model counting, the calls to an NP-oracle are implemented by using a SAT-solver. This is accomplished by adding to the formula i parity constraints, and checking for satisfiability.
- Naive encoding of a parity constraint over p variables requires 2^{p-1} clauses of length p - ruling out 2^p possible assignments (ones with wrong parity)
- Constraints can be compactly represented (introducing $O(p)$ extra variables) using standard Tseitin transformation (Tseitin, 1983).

Example: $x_1 \oplus x_2 \oplus x_3 \oplus x_4 = 0$ can be written as
$\{x_1 \oplus x_2 = z_1, x_2 \oplus x_3 = z_2, z_1 \oplus z_2 = 0\}$

Practical implementations

- First practical implementation by Gomes et al. (2006) who used a SAT solver as an *NP*-oracle.
- Their algorithm leverages decades of research and engineering in SAT solving techniques for approximate model counting, resulted in huge improvements
- Recently, Chakraborty et al. (2013); Ivrii et al. (2015) provided several practical improvements
- Specifically, the former introduced the use of *pivots*, where an *NP*-oracle is used to check the existence of at least $k > 1$ solutions ($k = 1$ corresponds to earlier discussed algorithm), in order to improve the accuracy of the estimated count

Probabilistic Models and Approx Inference: WISH Algorithm

- Generally, when the weight function is "close" for being constant on a subset of states, and zero elsewhere, then the hashing-based algorithm of Chakraborty et al. (2013) can be used - as in the model counting problem.
- Typical models in ML are unlikely to satisfy this restriction (e.g., weight function that is log-linear can have large variability)
- An alternative algorithm (WISH) based on universal hashing and combinatorial optimization which can handle general weight function, was introduced by Ermon et al. (2013)

WISH Algorithm

Algorithm 9.2 WISH $(w : \Sigma \to \mathbb{R}^+, n = \log_2 |\Sigma|, \Delta, \alpha)$

1: $T \leftarrow \left\lceil \frac{\ln(n/\delta)}{\alpha} \right\rceil$
2: **for** $i = 0, \cdots, n$ **do**
3:     **for** $t = 1, \cdots, T$ **do**
4:         Sample hash function $h^i_{A,b} : \Sigma \to \{0,1\}^i$, i.e.
5:             sample uniformly $A \in \{0,1\}^{i \times n}$, $b \in \{0,1\}^i$
6:         $w^t_i \leftarrow \max_\sigma w(\sigma)$ subject to $A\sigma \equiv b \pmod 2$
7:     **end for**
8:     $M_i \leftarrow \text{Median}(w^1_i, \cdots, w^T_i)$
9: **end for**
10: Return $M_0 + \sum_{i=0}^{n-1} M_{i+1} 2^i$

Thank you for listening!

Any questions?