COMP 480/580 — Probabilistic Algorithms and Data Structure Mar 7, 2018

Lecture 16

Lecturer: Anshumali Shrivastava *Scribe By: Wei Lu*

1 Importance Sampling via Locality Sensitive Hashing(LSH)

1.1 Motivating Problem: Stochastic Gradient Descent

Consider a dataset $\mathcal{D}$ with elements $x_1, x_2, ...x_i, ...x_N$ and a function $F(x)$ that is parameterized by a parameter vector θ. We wish to find an optimal θ^* such that $F(x)$ is either maximized or minimized. We can use a process known as gradient descent to find the optimal θ^* that minimizes $F(\theta)$. In gradient descent, we compute the gradient of $F(x, \theta)$ with respect to θ and we update θ using $\theta_t = \theta_{t-1} - \eta^t \nabla F(x_i, \theta_{t-1})$. To do this, we need to compute the full gradient ∇F, which is often slow.

Stochastic gradient descent improves the time (but not the number of iterations) needed to converge by replacing the full gradient ∇F with a stochastic gradient ∇f. Even though we require more iterations, the stochastic gradient is substantially cheaper to compute. One way to obtain a stochastic gradient is to iteratively pick x_i at random from the dataset and update θ only according to the gradient for that x_i. This is known as a sub-gradient. Using SGD at iteration t, $\theta_t = \theta_{t-1} - \eta^t \nabla f(x_i, \theta_{t-1})$, where $\nabla f(x_i, \theta_{t-1})$ is the sub-gradient of F with respect to θ at x_i and η is the "learning rate" - a value that determines how quickly and with what precision θ converges to θ^*.

We want a method that converges faster than the standard SGD method. Instead of uniformly picking x_i, the proposed method chooses x_i with probability proportional to w_i where the optimal variance weight $w_i = ||\nabla f(x_i, \theta_{t-1})||_2$ (Alain et. al 2015). The idea behind this procedure is that if the full gradient ∇F is influenced by only a few of the sub-gradients, we can converge faster by choosing to descend in the direction of the largest sub-gradients at each iteration. However, this creates a chicken-and-egg problem. In order to sample with weight w_i, we need to compute $\nabla f(x_i, \theta_{t-1})$. Therefore, we need to know all of the sub-gradients! Since computing all of the sub-gradients has the same complexity as computing the full gradient, this method is not practical without improvements.

1.2 Detour: Probabilistic Hashing

1.2.1 Probabilistic Fingerprinting(Hashing)

We have seen that hash functions give each input a discrete fingerprint (hash signature or value). If a locality-sensitive hash (LSH) is used, then the probability that two vectors $\mathbf{x}$ and $\mathbf{y}$ have the same fingerprint is high if $\mathbf{x}$ and $\mathbf{y}$ have high similarity. Popular similarity measures are: Jaccard similarity (MinHash), cosine similarity, Euclidian distance, earth mover distance, and Hamming distance. We have already seen an LSH for the Jaccard similarity.SimHash, sometimes referred to as signed random projections (SRP), is an LSH family for the cosine similarity. To compute the SimHash of a vector, we randomly generate a Gaussian vector $\mathbf{r} \sim N(\mathbf{0}, \mathbf{I})$ and compute

$$h_{\mathbf{r}}(\mathbf{x}) = \text{sign}(\mathbf{x}^T \mathbf{r})$$

The collision probability under SRP is

$$P(h(\mathbf{x}) = h(\mathbf{y})) = 1 - \frac{\cos^{-1}(\theta)}{\pi}$$

where θ is the angle between $\mathbf{x}$ and $\mathbf{y}$. θ is also known as the angular distance, and therefore the collision probability is monotone increasing with cosine similarity. By this, we mean that SimHash has the property

$$P(h(\mathbf{x}) = h(\mathbf{y})) = f(sim(x, y))$$

where f is monotonic. In the following sections, we will abbreviate $P(h(\mathbf{x}) = h(\mathbf{y}))$ as $p(x, y)$.

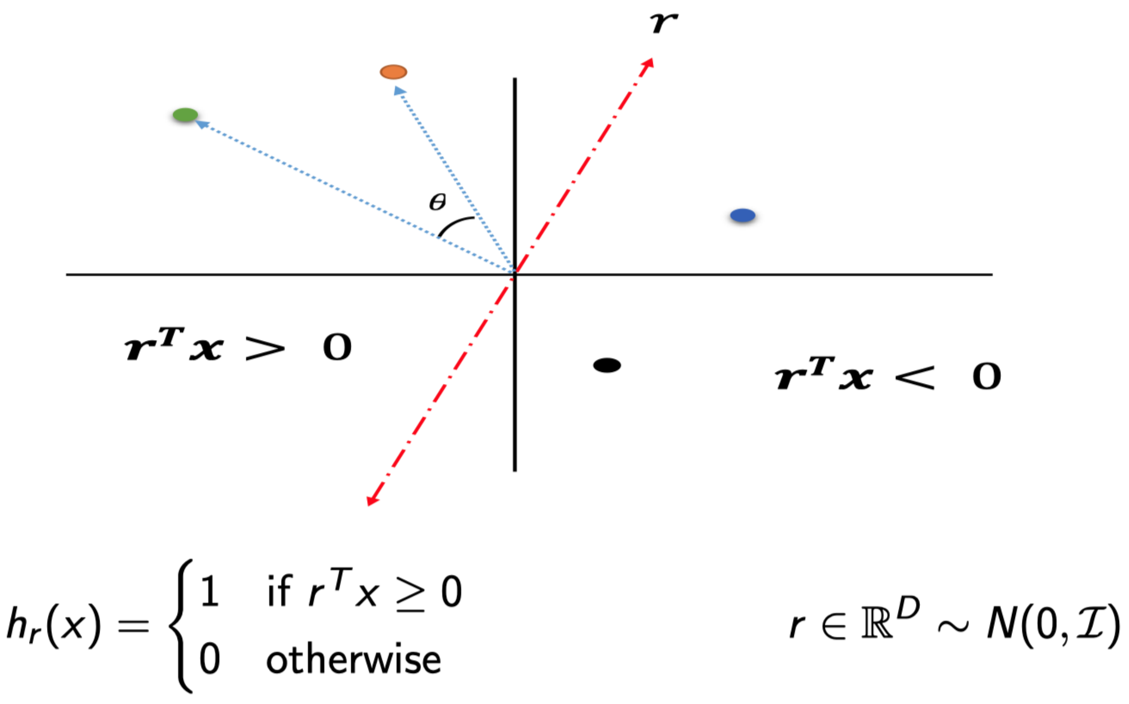
r
θ
$r^T x > 0$
$r^T x < 0$
$h_r(x) = \begin{cases} 1 & \text{if } r^T x \geq 0 \\ 0 & \text{otherwise} \end{cases}$
$r \in \mathbb{R}^D \sim N(0, \mathcal{I})$

Figure 1: The signed random projection hash family

1.2.2 Application of SimHash: Sub-linear time Near-Neighbor Search

If we want to find the data point $x \in \mathcal{D}$ that has the highest similarity with respect to a query q, $q = \arg\max_q \{\text{sim}(q, x)\}$. The naive method is to check every possible $x \in \mathcal{D}$, which requires $O(N)$ distance computations. Using LSH, we can create a hash table with K fingerprints and L tables that retrieves elements y for a query x with probability $p(y) = 1 - (1 - p(x, y)^K)^L$

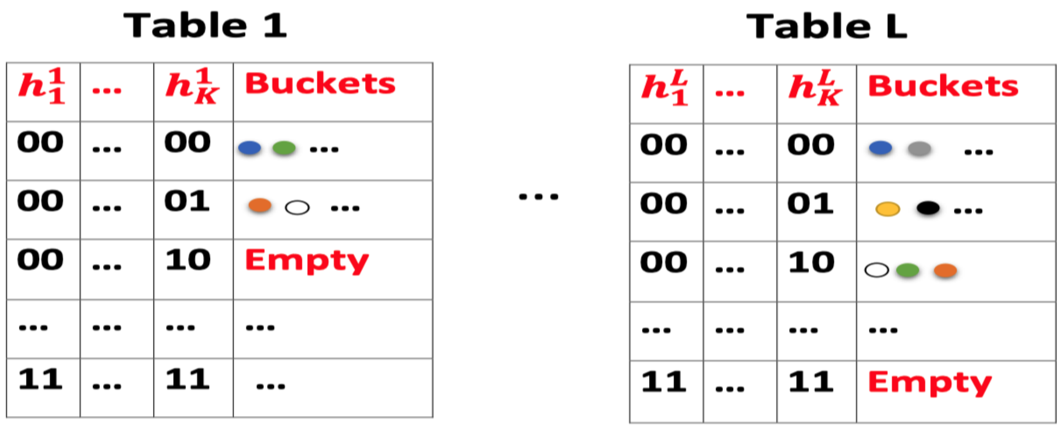

Figure 2: Two knobs K and L

1.3 Key observation: Hashing is an efficient adaptive sampling in disguise

1.3.1 Application: Partition Function in Log-Linear Models

Log-linear models are a common class of models for machine learning. The most computational demanding step in log-linear models is the computation of the partition function $Z_\theta = \sum_{y \in Y} e^{\theta_y x}$. One reasonable approach is to use importance sampling to estimate Z_θ. To do this, we would draw N samples y_i $g(y)$ for $i = 1...N$, then compute $\hat{Z}_\theta = 1/N \sum \frac{f(y_i)}{g(y_i)}$. However, this leads to a chicken-and-egg problem that is similar to the one involved with weighted SGD. We need $g(y)$ that is close to $f(y)$, but it is not easy to see how we can do this without explicitly finding the partition.

Our solution is to use LSH to efficiently generate $g(y)$ that close to $f(y)$. Here, we let

$$g(y) = p(y) = 1 - (1 - p(x, \theta_y)^K)^L$$

The distribution we really wish to draw from $(f(y))$ is

$$f(y) = e^{\theta_y x}$$

Upon closer examination, we see that $g(y)$ and $f(y)$ are both monotonic with respect to θ_y! This implies that, roughly speaking, $g(y)$ and $f(y)$ are close. In fact, they are sufficiently close that we can use importance sampling with $g(y)$. The breakthrough is that we can sample from $g(y)$ much more efficiently than from $f(y)$. To sample from $g(y)$, we simply have to construct our near-neighbor hash tables, then query x and get many y.

The final estimator $\hat{Z}_\theta = \sum_i \frac{e^{\theta_y x}}{1-(1-p(x,\theta_y)^K)^L}$. It should be noted that this formulation of importance sampling is not normalized and the samples are correlated, but this turns out to be of little importance in practice.

1.3.2 Application: Back to Adaptive SGD

We have just seen that LSH can improve the efficiency of log-linear model partitioning by providing an efficient distribution for importance sampling. Can we apply a similar technique to our SGD problem?

Recall from the first section that

$$w_i = ||\nabla f(x_i, \theta_t)||_2$$

This expression is equivalent to

$$|2(\theta_t x_i - y_i)||x_i||_2| = 2|\langle \theta_t, -1\rangle \langle x_i||x_i||_2, y_i||x_i||_2\rangle|$$

Note that θ is no longer the angle between x and a query but instead is our parameter vector, as before. Using LSH, we can efficiently sample $\langle x_i||x_i||_2, y_i||x_i||_2\rangle$ with probability equal to $p_i = 1-(1-p(\langle \theta_t, -1\rangle, \langle x_i||x_i||_2, y_i||x_i||_2\rangle)^K)^L$. The key observation here is that p_i is monotonic in w_i. Therefore, we can also use LSH and importance sampling to solve this chicken-and-egg problem! The per-iteration cost of this method is 1.5 times that of SGD, but the convergence rate is substantially better.