Case Study 1: Estimating Click Probabilities

Tackling an Unknown Number of Features with Sketching

Motivating AdaGrad (Duchi, Hazan, Singer 2011)

• Assuming $\mathbf{w} \in \mathbb{R}^d$, standard stochastic (sub)gradient descent updates are of the form:

$$w_i^{(t+1)} \leftarrow w_i^{(t)} - \eta g_{t,i}^{(i)}$$

• Should all features share the same learning rate?

• Often have high-dimensional feature spaces
  – Many features are irrelevant
  – Rare features are often very informative

• Adagrad provides a feature-specific adaptive learning rate by incorporating knowledge of the geometry of past observations
AdaGrad Algorithm

- At time $t$, estimate optimal (sub)gradient modification $A_t$ by
  \[ g_t = \nabla f(x_t), \quad A_t = \left( \sum_{\tau=1}^{t} g_\tau g_\tau^T \right)^{\frac{1}{2}} \]
  \[ \text{use proj 6.2 with } A_t \text{ at time } t. \]

- For $d$ large, $A_t$ is computationally intensive to compute. Instead,
  \[ \text{diag}(A_t) = \begin{pmatrix} A_{1,1} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & A_{d,d} \end{pmatrix}, \quad \text{where } A_{\tau,\tau} \approx \sum_{\tau=1}^{t} g_\tau^2. \]

- Then, algorithm is a simple modification of normal updates:
  \[ w^{(t+1)} = \arg \min_{w \in \mathcal{W}} \left\| w - (w^{(t)} - \eta \text{diag}(A_t)^{-1} g_t) \right\|_2^2 \text{ diag}(A_t) \]
  \[ w^{(t+1)} = w^{(t)} - \eta A_t^{-\frac{1}{2}} g_t. \]

AdaGrad in Euclidean Space

- For $\mathcal{W} = \mathbb{R}^d$,

- For each feature dimension,
  \[ w_{i}^{(t+1)} = w_{i}^{(t)} - \eta_{t,i} g_{t,i}, \]
  where
  \[ \eta_{t,i} = n \sqrt{A_{t,i}}. \]

- That is,
  \[ w_{i}^{(t+1)} = w_{i}^{(t)} - \frac{\eta}{\sqrt{\sum_{\tau=1}^{t} g_{\tau,i}^2}} g_{t,i}. \]

- Each feature dimension has its own learning rate!
  - Adapts with $t$
  - Takes geometry of the past observations into account
  - Primary role of $\eta$ is determining rate the first time a feature is encountered

©Sham Kakade 2016
AdaGrad Theoretical Guarantees

- AdaGrad regret bound:
  \[ R_\infty := \max_t ||w^{(t)} - w^*||_\infty \]
  \[ \sum_{t=1}^{\infty} \ell_t(w^{(t)}) - \ell_t(w^*) \leq 2R_\infty \sum_{t=1}^{\infty} g_{1:T,i}||_2 \]
  - In stochastic setting:
  \[ \mathbb{E} \left[ \ell \left( \frac{1}{T} \sum_{t=1}^{T} w^{(t)} \right) \right] - \ell(w^*) \leq \frac{2R_\infty}{T} \sum_{i=1}^{d} \mathbb{E} \left[ ||g_{1:T,i}||_2 \right] \]

- This really is used in practice!
- Many cool examples. Let’s just examine one...

AdaGrad Theoretical Example

- Expect to out-perform when gradient vectors are sparse
- SVM hinge loss example:
  \[ \ell_t(w) = [1 - y^t \langle x^t, w \rangle]^+ \]
  \[ x^t \in \{-1, 0, 1\}^d \]
- If \( x^t_i \neq 0 \) with probability \( \propto j^{-\alpha}, \quad \alpha > 1 \)
  \[ \mathbb{E} \left[ \ell \left( \frac{1}{T} \sum_{t=1}^{T} w^{(t)} \right) \right] - \ell(w^*) = O \left( \frac{||w^*||_\infty}{\sqrt{T}} \cdot \max \{ \log d, d^{1-\alpha/2} \} \right) \]
- Previously best known method:
  \[ \mathbb{E} \left[ \ell \left( \frac{1}{T} \sum_{t=1}^{T} w^{(t)} \right) \right] - \ell(w^*) = O \left( \frac{||w^*||_\infty}{\sqrt{T}} \cdot \sqrt{d} \right) \]
Neural Network Learning

- Very non-convex problem, but use SGD methods anyway

\[ \ell(w, x) = \log(1 + \exp(\langle p(\langle w_1, x_1 \rangle) \cdots p(\langle w_k, x_k \rangle) \rangle, x_0))) \]

\[ p(\alpha) = \frac{1}{1 + \exp(\alpha)} \]

(Dean et al. 2012)

Distributed, \( d = 1.7 \cdot 10^9 \) parameters. SGD and AdaGrad use 80 machines (1000 cores), L-BFGS uses 809 (10000 cores). Images from Duchi et al. ISMP 2012 slides

Related Ideas for Adversary

1. Newton’s Method
2. Conjugate gradient
3. BFGS
4. Natural gradient (loss)

\[ w^{t+1} = w^t - \alpha \nabla_l \log p(z^{(t+1)}) \nabla_l \]
What you should know about Logistic Regression (LR) and Click Prediction

• Click prediction problem:
  – Estimate probability of clicking
  – Can be modeled as logistic regression
• Logistic regression model: Linear model
• Gradient ascent to optimize conditional likelihood
• Overfitting + regularization
• Regularized optimization
  – Convergence rates and stopping criterion
• Stochastic gradient ascent for large/streaming data
  – Convergence rates of SGD
• AdaGrad motivation, derivation, and algorithm

Problem 1: Complexity of LR Updates

• Logistic regression update:
  \[ w_i^{(t+1)} \leftarrow w_i^{(t)} + \eta_t \left\{ -\lambda w_i^{(t)} + x_i^{(t)} [y^{(t)} - P(Y = 1| x^{(t)}, w^{(t)})] \right\} \]

• Complexity of updates:
  – Constant in number of data points
  – In number of features?
    • Problem both in terms of computational complexity and sample complexity
    \[ \text{\underline{1 \% f e r t h e s a m p l e}} \]
• What can we with very high dimensional feature spaces?
  – Kernels not always appropriate, or scalable
  – What else?
Problem 2: Unknown Number of Features

- For example, bag-of-words features for text data:
  - “Mary had a little lamb, little lamb…”
  - What’s the dimensionality of $x$?
  - What if we see new word that was not in our vocabulary?
    - Obamacare
      - Theoretically, just keep going in your learning, and initialize $w_{\text{Obamacare}} = 0$
      - In practice, need to re-allocate memory, fix indices,… A big problem for Big Data

What Next?

- Hashing & Sketching!
  - Addresses both dimensionality issues and new features in one approach!

- Let’s start with a much simpler problem: Is a string in our vocabulary?
  - Membership query

- How do we keep track?
  - Explicit list of strings
    - Very slow
  - Fancy Trees and Tries
    - Hard to implement and maintain
  - Hash tables?
### Hash Functions and Hash Tables

- Hash functions map **keys** to integers (bins):
  - Keys can be integers, strings, objects,…

- Simple example: \( \text{mod} \)
  - \( h(i) = (a \cdot i + b) \mod m \)
  - \( a = 7 \), \( b = 11 \), \( m = 32 \)
    - \( i = 4 \), \( h(i) = 39 \mod 32 = 7 \)
  - Random choice of \((a,b)\) (usually primes)
  - If inputs are uniform, bins are uniformly used
  - From two results can recover \((a,b)\), so not pairwise independent -> Typically use fancier hash functions

- Hash table:
  - Store list of objects in each bin
  - Exact, but storage still linear in size of object ids, which can be very long
    - E.g., hashing very long strings, entire documents

### Hash Bit-Vector Table-Based Membership Query

- Approximate queries with one-sided error: Accept false positives only
  - If we say no, element is not in set
  - If we say yes, element is very likely to be in set

- Given hash function, keep binary bit vector \( v \) of length \( m \):
  - Query \( Q(i) \): Element \( i \) in set?
  - \( v(h(i)) = 0 \) \( \Rightarrow \) \( Q(i) = 0 \)
  - \( v(h(i)) = 1 \) \( \Rightarrow \) \( Q(i) = \text{prob. yes} \)

- Collisions:
  - \( h(\text{obama\textunderscore name}) = 8 \)
  - \( h(\text{man\textunderscore name}) = 8 \)
    - \( \text{prob. at collision} \)

- Guarantee: One-sided errors, but may make many mistakes
  - How can we improve probability of correct answer?
Bloom Filter: Multiple Hash Tables

- Single hash table → Many false positives
- Multiple hash tables with independent hash functions
  - Apply $h_1(i), \ldots, h_p(i)$, set all bits to 1
- Query $Q(i)$?
  \[
  \text{if } \forall j \quad V_j(h_j(i)) = 1 \quad \text{set } Q(i) = \text{Yes}
  \]
  \[
  \text{else } Q(i) = \text{No}
  \]
- Significantly decrease probability of false positives

Analysis of Bloom Filter

- Want to keep track of $n$ elements with false positive probability of $\delta > 0$... how large $m$ & $p$?
  \[
  \dim_{\text{of each table}} \leq \# \text{ of hash functions}
  \]
- Simple analysis yields:
  \[
  m = \frac{n \log_2 \frac{1}{\delta}}{\ln 2} \approx 1.5n \log_2 \frac{1}{\delta}
  \]
  \[
  p = \log_2 \frac{1}{\delta} \quad \text{prob of error dec. exp. quickly with } p
  \]
Sketching Counts

• Bloom Filter is super cool, but not what we need...
  – We don’t just care about whether a feature existed before, but to keep track of counts of occurrences of features! (assuming $x_i$ integer)

• Recall the LR update:
  \[ w_i^{(t+1)} \leftarrow w_i^{(t)} + \eta \{ -\lambda w_i^{(t)} + x_i^{(t)}[y^{(t)}] - P(Y = 1|x^{(t)}, w^{(t)}) \} \]

• Must keep track of (weighted) counts of each feature:
  – E.g., with sparse data, for each non-zero dimension $i$ in $x^{(t)}$:

    \[
    \text{for } i \in \{ \text{ s.t. } x_i^{(t)} \neq 0 \}
    \text{ w } + x_i^{(t)} \cdot \text{ const }\]

• Can we generalize the Bloom Filter?

Count-Min Sketch: single vector

• Simpler problem: Count how many times you see each string

• Single hash function:
  – Keep Count vector of length $m$
  – every time see string $i$:

    \[
    \text{Count}[h(i)] \leftarrow \text{Count}[h(i)] + 1
    \]

  – Again, collisions could be a problem:
    • $a_i$ is the count of element $i$:

    \[
    \text{count}(i) = \sum_{i: h(i) = j} a_i
    \]
Count-Min Sketch: general case

- Keep \( p \) by \( m \) Count matrix

- \( p \) hash functions:
  - Just like in Bloom Filter, decrease errors with multiple hashes
  - Every time see string \( i \):
    \[
    \forall j \in \{1, \ldots, p\} : \text{Count}[j, h_j(i)] \leftarrow \text{Count}[j, h_j(i)] + 1
    \]

Querying the Count-Min Sketch

- \( \forall j \in \{1, \ldots, p\} : \text{Count}[j, h_j(i)] \leftarrow \text{Count}[j, h_j(i)] + 1 \)
- Query \( Q(i) \)?
  - What is in \( \text{Count}[j,k] \)?
    \[
    \text{count}[(i,k)] = \sum_{i : h_j(i) = k} a_i \geq a_i
    \]
  - Thus:
  
  - Return:
    \[
    a_i \leq \min \text{count}(j, h_j(i)) \geq a_i
    \]
Analysis of Count-Min Sketch

\[ \hat{a}_i = \min_j \text{Count}[j, h(i)] \geq a_i \]

- Set:
  \[ m = \left\lceil \frac{e}{\epsilon} \right\rceil, \quad p = \left\lceil \ln \frac{1}{\delta} \right\rceil \]
  \[ O\left(\frac{m}{\epsilon^2}\right) \]

- Then, after seeing \( n \) elements:
  \[ \hat{a}_i \leq a_i + \epsilon n \]

- With probability at least \( 1-\delta \)

---

Proof of Count-Min for Point Query with Positive Counts: Part 1 – Expected Bound

- \( I_{i,j,k} \) = indicator that \( i \) & \( k \) collide on hash \( j \):

- Bounding expected value:

- \( X_{ij} \) = total colliding mass on estimate of count of \( i \) in hash \( j \):

- Bounding colliding mass:

- Thus, estimate from each hash function is close in expectation
Proof of Count-Min for Point Query with Positive Counts: Part 2 – High Probability Bounds

- What we know: \( \text{Count}[j, h_j(i)] = a_i + X_{i,j} \quad E[X_{i,j}] \leq \frac{\epsilon n}{e} \)

- Markov inequality: For \( z_1, \ldots, z_k \) positive iid random variables
  \[ P(\forall z_i : z_i > \alpha E[z_i]) < \alpha^{-k} \]

- Applying to the Count-Min sketch:

But updates may be positive or negative

\[ w_i^{(t+1)} \leftarrow w_i^{(t)} + \eta_t \left\{ -\lambda w_i^{(t)} + x_i^{(t)} [y^{(t)} - P(Y = 1|\mathbf{x}^{(t)}, \mathbf{w}^{(t)})] \right\} \]

- Count-Min sketch for positive & negative case
  - \( a_i \) no longer necessarily positive

- Update the same: Observe change \( \Delta_i \) to element \( i \):
  \[ \forall j \in \{1, \ldots, p\} : \text{Count}[j, h_j(i)] \leftarrow \text{Count}[j, h_j(i)] + \Delta_i \]
  - Each \( \text{Count}[j, h(j)] \) no longer an upper bound on \( a_i \)

- How do we make a prediction?

- Bound: \[ |\hat{a}_i - a_i| \leq 3\epsilon ||\mathbf{a}||_1 \]
  - With probability at least \( 1-\delta^{1/4} \), where \( ||\mathbf{a}|| = \sum_i |a_i| \)
Finally, Sketching for LR

\[ w_i^{(t+1)} \leftarrow w_i^{(t)} + \eta_t \left\{ -\lambda w_i^{(t)} + x_i^{(t)} [y^{(t)} - P(Y = 1|\mathbf{x}^{(t)}, \mathbf{w}^{(t)})] \right\} \]

- Never need to know size of vocabulary!
- At every iteration, update Count-Min matrix:

- Making a prediction:

- Scales to huge problems, great practical implications...

Hash Kernels

- Count-Min sketch not designed for negative updates
- Biased estimates of dot products

- Hash Kernels: Very simple, but powerful idea to remove bias
- Pick 2 hash functions:
  - \( h \): Just like in Count-Min hashing
  - \( \xi \): Sign hash function
    - Removes the bias found in Count-Min hashing (see homework)

- Define a “kernel”, a projection \( \phi \) for \( \mathbf{x} \):
Hash Kernels Preserve Dot Products

\[ \phi_i(x) = \sum_{j: h(j) = i} \xi(j)x_j \]

- Hash kernels provide unbiased estimate of dot-products!
- Variance decreases as \( O(1/m) \)
- Choosing \( m \)? For \( \varepsilon > 0 \), if
  \[
  m = \mathcal{O} \left( \frac{\log N}{\varepsilon^2} \right)
  \]
  - Under certain conditions...
  - Then, with probability at least 1-\( \delta \):
  \[
  (1 - \varepsilon)\|x - x'\|_2^2 \leq \|\phi(x) - \phi(x')\|_2^2 \leq (1 + \varepsilon)\|x - x'\|_2^2
  \]

Learning With Hash Kernels

- Given hash kernel of dimension \( m \), specified by \( h \) and \( \xi \)
  - Learn \( m \) dimensional weight vector
- Observe data point \( x \)
  - Dimension does not need to be specified a priori!
- Compute \( \phi(x) \):
  - Initialize \( \phi(x) \)
  - For non-zero entries \( j \) of \( x_j \):
- Use normal update as if observation were \( \phi(x) \), e.g., for LR using SGD:
  \[
  w_i^{(t+1)} \leftarrow w_i^{(t)} + \eta_t \left\{ -\lambda w_i^{(t)} + \phi_i(x^{(t)})y^{(t)} - P(Y = 1|\phi(x^{(t)}), w^{(t)}) \right\}
  \]
Interesting Application of Hash Kernels: Multi-Task Learning

• Personalized click estimation for many users:
  – One global click prediction vector \( w \):
    • But...
    – A click prediction vector \( w_u \) per user \( u \):
  • But...

• Multi-task learning: Simultaneously solve multiple learning related problems:
  – Use information from one learning problem to inform the others

• In our simple example, learn both a global \( w \) and one \( w_u \) per user:
  – Prediction for user \( u \):
    • If we know little about user \( u \):
    • After a lot of data from user \( u \):

Problems with Simple Multi-Task Learning

• Dealing with new user is annoying, just like dealing with new words in vocabulary

• Dimensionality of joint parameter space is HUGE, e.g. personalized email spam classification from Weinberger et al.:
  – 3.2M emails
  – 40M unique tokens in vocabulary
  – 430K users
  – 16T parameters needed for personalized classification!
Hash Kernels for Multi-Task Learning

• Simple, pretty solution with hash kernels:
  – Very multi-task learning as (sparse) learning problem with (huge) joint data point \( z \) for point \( x \) and user \( u \):

• Estimating click probability as desired:

• Address huge dimensionality, new words, and new users using hash kernels:

Simple Trick for Forming Projection \( \phi(x,u) \)

• Observe data point \( x \) for user \( u \)
  – Dimension does not need to be specified a priori and user can be new!

• Compute \( \phi(x,u) \):
  – Initialize \( \phi(x,u) \)
  – For non-zero entries \( j \) of \( x_j \):
    • E.g., \( j='\text{Obamacare}' \)
    • Need two contributions to \( \phi \):
      – Global contribution
      – Personalized Contribution
    • Simply:

• Learn as usual using \( \phi(x,u) \) instead of \( \phi(x) \) in update function
Results from Weinberger et al. on Spam Classification: Effect of $m$

Figure 2. The decrease of uncaught spam over the baseline classifier averaged over all users. The classification threshold was chosen to keep the not-spam misclassification fixed at 1%. The hashed global classifier (global-hashed) converges relatively soon, showing that the distortion error $\epsilon_d$ vanishes. The personalized classifier results in an average improvement of up to 30%.

Results from Weinberger et al. on Spam Classification: Multi-Task Effect

Figure 3. Results for users clustered by training emails. For example, the bucket [8, 15] consists of all users with eight to fifteen training emails. Although users in buckets with large amounts of training data do benefit more from the personalized classifier (up to 65% reduction in spam), even users that did not contribute to the training corpus at all obtain almost 20% spam-reduction.
What you need to know

• Hash functions
• Bloom filter
  – Test membership with some false positives, but very small number of bits per element
• Count-Min sketch
  – Positive counts: upper bound with nice rates of convergence
  – General case
• Application to logistic regression
• Hash kernels:
  – Sparse representation for feature vectors
  – Very simple, use two hash function (Can use one hash function...take least significant bit to define ξ)
  – Quickly generate projection ϕ(x)
  – Learn in projected space
• Multi-task learning:
  – Solve many related learning problems simultaneously
  – Very easy to implement with hash kernels
  – Significantly improve accuracy in some problems *(if there is enough data from individual users)*