Stochastic Gradient Descent

CSE *97B

Big Data

Empirical Risk

- Data $= \left\{(\vec{X}_1, t_1), (\vec{X}_2, t_2), \dots, (\vec{X}_n, t_n)\right\}$
  - $\vec{X}_i$ is the feature vector (set of measurements) for the i$^{\text{th}}$ record: $\vec{X}_i = (X_i[0], X_i[1], \dots, X_i[m])$.
  - t_i is the target for the i$^{\text{th}}$ record
- Models have parameters (e.g, $\theta, \vec{w}, b$)
  - Linear regression: $\widehat{t} = \vec{w} \cdot \vec{X} + b$
  - Logistic regression: $\widehat{P}(t) = \frac{1}{1+\exp(-t(\vec{w}\cdot\vec{x}+b))}$
    - $t = \pm 1$ (e.g. $t = 1$ if rain, $t = -1$ if no rain).
    - Logistic regression models the probability of an event.
- Parameters set by optimization problem (minimize empirical risk).
  - Linear regression: $\vec{w}, b = \text{argmin} \sum_{i=1}^{n} (\vec{w} \cdot \vec{X}_i + b - t_i)^2$
  - Logistic regression: $\vec{w}, b = \text{argmin} \sum_{i=1}^{n} \log\left(1 + \exp(-t_i(\vec{w} \cdot \vec{X}_i + b))\right)$
  - Common pattern: $\vec{w}, b = \text{argmin} \sum_{i=1}^{n} f(\vec{X}_i, t_i, \vec{w}, b)$

Minimizing Empirical Risk

- Linear regression: $\vec{w}, b = \text{argmin} \sum_{i=1}^{n} (\vec{w} \cdot \vec{X}_i + b - t_i)^2$
  - Empirical risk $R(\vec{w}, b) = \sum_{i=1}^{n} (\vec{w} \cdot \vec{X}_i + b - t_i)^2$
  - Can be solved in closed form (i.e. formula).
  - Compute $\frac{\partial R}{\partial w[0]}, \frac{\partial R}{\partial w[1]}, \dots, \frac{\partial R}{\partial w[m]}, \frac{\partial R}{\partial b}$
  - Set partial derivatives to 0 and solve.
- Logistic regression: $\vec{w}, b = \text{argmin} \sum_{i=1}^{n} \log \Big(1 + \exp(-t_i(\vec{w} \cdot \vec{X}_i + b))\Big)$
  - No closed form solution, need algorithm.
- Common pattern: $\vec{w}, b = \text{argmin} \sum_{i=1}^{n} f(\vec{X}_i, t_i, \vec{w}, b)$
  - Generaly no closed form solution, need algorithm.
- mini-batch Stochastic gradient descent: simple and fast.

Mini-Batch Stochastic Gradient Descent

Goal: find $\vec{w}, b$ to minimize $\sum_{i=1}^{n} f(\vec{X}_i, t_i, \vec{w}, b)$

parameters: Batch size k; learning rate η `// Typically` $k = 10$

Initialize b and $\vec{w}$

while *stopping criterion not met* **do**

  Select next k records $(\vec{X}_j, t_j), (\vec{X}_{j+1}, t_{j+1}), \ldots, (\vec{X}_{j+k-1}, t_{j+k-1})$

    `// Error contribution is` $\sum_{i=j}^{j+k-1} f(\vec{X}_i, t_i, \vec{w}, b)$

  $b \leftarrow b - \eta \sum_{i=j}^{j+k-1} \frac{\partial f(\vec{X}_i, t_i, \vec{w}, b)}{\partial b}$

  for $\ell = 0..m$ **do**

    $w[\ell] \leftarrow w[\ell] - \eta \sum_{i=j}^{j+k-1} \frac{\partial f(\vec{X}_i, t_i, \vec{w}, b)}{\partial w[\ell]}$

  end

  if *at end of dataset* **then**

    Permute dataset;

  end

end

In MapReduce

- Algorithm is inherently sequential.
- Cannot be parallelized as is.
- Can be approximated in parallel mode.
  - Zinkevich et al. "Parallelized Stochastic Gradient Descent" NIPS 2010.
- Main idea:
  - Each mapper independently performs stochastic gradient descent.
    - Same initial $\vec{w}$ and b
  - A reducer averages the $\vec{w}$ from all mappers (similarly for b)
  - This is the common starting point for the next iteration.
- We use sort and shuffle phase to permute the data.

Controller

```
// For simplicity, batch size k = 1
```

Set learning rate η for each mapper

Initialize $\vec{w}$ and b, distribute to all mappers

while *convergence criterion not met* **do**

  Set up mapreduce job

  Obtain $\vec{w}$ and b from Reducer 0 `// via file or Hive`

  Distribute $\vec{w}$ and b to each mapper

  Modify η if necessary, set for each mapper

end

Mapper

def *setup()*:
    Get parameters η
    Read initial $\vec{w}$ and b

def *map(key, value)*:
    $(\vec{X}, t) =$parse(value)
    emit(RandomPositiveNumber, value)`// for permuting data`
    $b \leftarrow b - \eta \frac{\partial f(\vec{X}, t, \vec{w}, b)}{\partial b}$
    for $\ell = 0..m$ **do**
        $w[\ell] \leftarrow w[\ell] - \eta \frac{\partial f(\vec{X}, t, \vec{w}, b)}{\partial w[\ell]}$
    end

def *cleanup()*:
    emit($\vec{w}$ and b to Reducer 0)

Reducer

def *reduce(key, value-list)*:

- **if** *value-list is the set of vectors:* $\vec{w}_1, \vec{w}_2, \ldots, \vec{w}_\ell$ **then**
  - $\vec{w} = \frac{1}{\ell}\sum_{i=1}^{\ell} \vec{w}_i$
- **else if** *value-list is the set of b values:* $b_1, b_2, \ldots, b_\ell$ **then**
  - $b = \frac{1}{\ell}\sum_{i=1}^{\ell} b_i$
- **else**
  - `// Ignore key (which is random number)`
  - `// value is data record, output it`
  - For every value in value-list, emit(value)
- **end**

def *cleanup()*:

- Output $\vec{w}$ and b (e.g. to HDFS or Hive)