CS535 Big Data

3/26/2019  Week 10-A  Sangmi Lee Pallickara

PART B. GEAR WORKSHOP II
SCALABLE COMPUTING MODELS

Sangmi Lee Pallickara
Computer Science, Colorado State University
http://www.cs.colostate.edu/~cs535

FAQs
- Programming Assignment 2
  - 5:00PM March 29 (Adjusted)
  - Team submission
- Scores of the workshop critical reviews are available
- Scores of the TP1 are available

Today’s topics
- Optimization for ML
  - Distributed optimization for ML

Scalable computing challenges in Analytics
- Effective data access
  - CD-ROM drive, 200 milliseconds;
  - Hard drive, 15 milliseconds
  - Random Access Memory (RAM), 70 nanosecond
- Effective computing
  - Optimization of the analytics algorithms

http://www.cs.colostate.edu/~cs535
What is "optimization"?

- Finding one or more minimizer of a function subject to constraints
- Most of machine learning problems are optimization problems in computing
- For example, in k-Means clustering
  - Looks for k-clusters in which each observation belongs to the cluster with the nearest mean
  - In this case, “optimization” is the process to find:

\[ \text{arg min}_{j_1, j_2, \ldots, j_k} \| \sum_{i \in j_l} x_i - \mu_l \| \]

Sometimes, optimization is NOT straightforward

- Minimize \( f(x) \)

Convex optimization

- Convex function
- Definition
  - A function \( f: \mathbb{R}^n \to \mathbb{R} \) is convex if for \( x, y \in \text{dom } f \) and any \( a \in [0,1] \),
  \[ f(ax + (1 - a)y) \leq af(x) + (1 - a)f(y) \]

Convexity optimization

- Theorem
  - If \( x \) is a local minimizer of a convex optimization problem, it is a global minimizer

Optimizations in Spark

- Spark supports
  - Gradient descent
  - Stochastic gradient descent (SGD)
  - Limited-memory BFGS (L-BFGS)

http://www.cs.colostate.edu/~cs535
Gradient Descent

- The simplest method to solve optimization problems
- Achieve $\min_{\theta \in \mathbb{R}} f(\theta)$
- Suitable for large scale and distributed computation
- Finds a local minimum of a function by iteratively taking steps in the direction of steepest descent
- Negative of the derivative (gradient) of the function at the current point

Objective function (Cost function)

- For a given training set, how do we pick, or learn the parameter $\theta$?
  - Make $h(x)$ close to $y$
  - Make your prediction close to the real observation
- We define the objective (cost) function
  - Using Mean Squared Error and multiplying $\frac{1}{2}$ for convenience
  $$J(\theta) = \frac{1}{2m} \sum_{i=0}^{m} (h_\theta(x^i) - y^i)^2$$

Minimization problem

- We have a function $f(\theta_0, \theta_1)$
- We want to find $\min_{\theta_0, \theta_1} f(\theta_0, \theta_1)$
- Goal: Find parameters to minimize the cost (output of the objective function)
- Outline of our approach:
  - Start with some $\theta_0, \theta_1$
  - Keep changing $\theta_0, \theta_1$ to reduce $J(\theta_0, \theta_1)$ until we end up at a minimum

Gradient descent algorithm

Repeat until convergence

$$\theta_j \leftarrow \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$$

(for $j=0$ and $j=1$)

$$\begin{align*}
\theta_0 &\leftarrow \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) \\
\theta_1 &\leftarrow \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)
\end{align*}$$
Decreasing $\theta_1$

- Positive slope

\[ \theta_1 = \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1) \]

Increasing $\theta_1$

- Negative slope

\[ \theta_1 = \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1) \]

Learning rates

- If $\alpha$ is too small, gradient descent can be slow

- If $\alpha$ is too large, gradient descent can overshoot the minimum

\[ \frac{1}{m} \sum_{i=1}^{m} (h_{\theta}(x(i)) - y(i))^2 \]

Case 1, $i = 0$:

\[ \frac{1}{m} \sum_{i=1}^{m} (h_{\theta}(x(i)) - y(i))^2 \]

Case 2, $i = 1$:

\[ \frac{1}{m} \sum_{i=1}^{m} (h_{\theta}(x(i)) - y(i))^2 \]

GEAR Workshop II | Scalable Computing Models

Optimization Algorithms

2. Stochastic Gradient Descent

Stochastic Gradient Descent (SGD)

- Batch methods
  - Full training set to compute the next update to parameters at each iteration tend to converge very well

- Advantage
  - Straight forward to get working provided a good off the shelf implementation
  - Very few hyper-parameters to tune

- Disadvantages
  - Computing the cost and gradient for the entire training set can be very slow
  - Intractable on a single machine if the dataset is too big to fit in main memory
  - No easy way to incorporate new data in an ‘online’

http://www.cs.colostate.edu/~cs535
Stochastic Gradient Descent (SGD)

- Stochastic Gradient Descent (SGC)
  - Follows the negative gradient of the objective after seeing only a single or a few training examples
  - The use of SGD in the neural network setting is motivated by the high cost of running back propagation over the full training set.
  - Fast convergence

SGD used in supervised machine learning with Spark

- The loss is written as an average of the individual losses coming from each data point
- A stochastic subgradient is a randomized choice of a vector
  - Selects one datapoint \( i \in \{1, \ldots, n\} \) uniformly at random, to obtain a stochastic subgradient of \( f(w) \) with respect to \( w \) as follows:
    \[
    S_i^w = G_i^w + \lambda \nabla L_i(w, x_i, y_i),
    \]
  - Where \( S_i^w \) is a sub-gradient of the part of the loss function determined by \( i \)-th data point
  - \( S_i^w \) is a sub-gradient of the regularizer \( R(w) \)

SGD used in supervised machine learning with Spark: Update schemes for Distributed SGD

- SGD uses a simple (distributed) sampling of the data examples
  - Recall the SGD optimization problem (1):
    \[
    f(w) = J(w) + \frac{1}{n} \sum_{i=1}^{n} L(w; x_i, y_i), \quad \cdots (1)
    \]
  - Here, the loss part of the optimization problem
    \[
    \frac{1}{n} \sum_{i=1}^{n} L(w; x_i, y_i)
    \]
  - Therefore, the true sub-gradient:
    \[
    \frac{1}{n} \sum_{i=1}^{n} S_i^w
    \]
  - This will require access to the full dataset

Stochastic Gradient Descent

- The standard gradient descent algorithm updates the parameters \( \theta \) of the objective \( J(\theta) \) as:
  \[
  \theta = \theta - \alpha J(\theta)
  \]
  - where the function evaluates the cost and gradient over the full training set
  - Stochastic Gradient Descent (SGD) uses only a single or a few training examples
    \[
    \theta = \theta - \alpha J(\theta; x_i, y_i)
    \]
  - with a pair \((x_i, y_i)\) from the training set

SGD used in supervised machine learning with Spark

- Running SGD is now walking in the direction of the negative stochastic sub-gradient
  \[
  \theta^{t+1} = \theta^t - y S_i^w
  \]
  - \( y \) is the step size
  - Default implementation is decreasing with the square root of the iteration counter

SGD used in supervised machine learning with Spark: Update schemes for Distributed SGD

- In Apache Spark, the parameter miniBatchFraction specifies which fraction of the full data
  - The average of the gradients within this subset
    \[
    \frac{1}{|S_i|} \sum_{i \in S} L_i(w_i)
    \]
  - Will be an actual stochastic gradient
  - Here, \(|S|\) is the sample subset size
  - In each iteration, Spark performs sampling in its RDDs

http://www.cs.colostate.edu/~cs535

Spring 2019 Colorado State University, page 5
SGD used in supervised machine learning with Spark
• Update schemes for Distributed SGD
  - \(|S|\): size of the sampled subset
  - \(|S| = miniBatchFraction \times n\)
  - If \(|S| = 1\), it is equivalent to the standard SGD
    - In that case, the step direction depends from the uniformly random sampling of the point

Limited-memory BFGS (L-BFGS)
• BFGS (Broyden–Fletcher–Goldfarb–Shanno algorithm)
  - Iterative method for solving unconstrained nonlinear optimization problems
  - Objective functions are non-linear
  - A type of quasi-Newton methods

Choosing an optimization method
• Linear methods use optimization internally
  - Linear SVM, logistic regression, regressions (Linear least squares, Lasso)
  - Some linear methods in `spark.mllib` support both SGD and L-BFGS
• Different optimization methods can have different convergence guarantees
  - depending on the properties of the objective function
  - In general, when L-BFGS is available, we recommend using it instead of SGD since L-BFGS tends to converge faster (in fewer iterations)
GD and SGD: Implementation in MLlib

- Gradient descent methods including stochastic sub-gradient descent (SGD) as included as a low-level primitive in MLlib

```scala
Vector.optimize(RDD<scala.Tuple2<Object, Vector>>, Vector initialWeights)
```

- The SGD class GradientDescent sets the following parameters:

  - `Gradient` - A class that computes the stochastic gradient of the function being optimized, i.e., with respect to a single training example, at the current parameter value.
  - `MLlib` includes gradient classes for common loss functions, e.g., hinge, logistic, least-squares.
  - The gradient class takes as input a training example, its label, and the current parameter value.

- `Updater` - A class that performs the actual gradient descent step, i.e., updating the weights in each iteration, for a given gradient of the loss part.
  - `MLlib` includes updaters for cases without regularization, as well as L1 and L2 regularizers.

  ```scala
  stepSize
  ```

  A scalar value denoting the initial step size for gradient descent. All updaters in MLlib use a step size at the t-th step equal to \( \frac{1}{\text{stepSize}} \).

  ```scala
  numIterations
  ```

  The number of iterations to run.

GD and SGD: Implementation in MLlib

- `regParam` - The regularization parameter when using L1 or L2 regularization.

- `miniBatchFraction` - The fraction of the total data that is sampled in each iteration, to compute the gradient direction. Sampling still requires a pass over the entire RDD, so decreasing `miniBatchFraction` may not speed up optimization much. Users will see the greatest speedup when the gradient is expensive to compute, for only the chosen samples are used for computing the gradient.

GEAR Workshop II | Scalable Computing Models
Distributed Optimization Algorithms

Asynchronized Parallel Optimization

This material is built based on

- Jeffrey Dean and Greg S. Corrado and Rajat Monga and Kai Chen and Matthieu Devin and Quoc V. Le and Mark Z. Mao and Marc'Aurelio Ranzato and Andrew Senior and Paul Tucker and Ke Yang and Andrew Y. Ng, Large Scale Distributed Deep Networks, 2012, NIPS

- Martin Abadi and Paul Barham and Jianmin Chen and Zhifeng Chen and Andy Davis and Jeffrey Dean and Matthieu Devin and Sanjay Ghemawat and Geoffrey Irving and Michael Isard and Martin Jaggi and Li-Jia Li and Yoshua Bengio and Roman Patro and Ava Pendar and Sai Srinivas and Markéta Zabicka and Vincent�, TensorFlow: A system for large-scale machine learning, 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16), 2016

http://www.cs.colostate.edu/~cs535

Spring 2019 Colorado State University, page 7
Distributed optimization and Synchronization schemes

- Large dataset for training
  - Not only training within a single instance of model
  - Distribute training across multiple model instances

- E.g. SGD is one of the most popular optimization procedure for training deep neural network
  - The traditional formulation of SGD is inherently sequential
  - The time required to move through the data in an entirely serial fashion is prohibitive

Overview of DistBelief

- Large dataset for training
  - Not only training within a single instance of model
  - Distribute training across multiple model instances

- Downpour SGD
  - Sandblaster L-BFGS

  Take advantage of the distributed computation within each individual replica
  - Tolerate variance in the processing speed of different model replicas
  - Even the failure of model replicas which may be taken offline or restarted at random

Downpour SGD

- Objectives of Downpour SGD
  - Applying SGD to extremely large datasets

  - A variant of asynchronous stochastic gradient descent
    - Uses multiple replicas of a single DistBelief model

  - This approach is asynchronous in two distinct aspects
    - The model replicas run independently of each other
    - The parameter server shards also run independently of one another

- Divide the training data into a number of subsets and run a copy of the model on each of these subsets

  - The models communicate updates through a centralized parameter server
    - Keeps the current state of all parameters for the model
    - Shards parameters across many machines
    - E.g., if we have 10 parameter server shards, each shard is responsible for storing and applying updates to 1/10th of the model parameters.

- Before processing each mini-batch, a model replica asks the parameter server service for an updated copy of its model parameters
  - Each machine needs to communicate with just the subset of parameter server shards that hold the model parameters relevant to its partition

  - After receiving an updated copy of its parameters, the DistBelief model replica processes a mini-batch of data to compute a parameter gradient

  - Sends the gradient to the parameter server

  - Then, applies the gradient to the current value of the model parameters
Reducing the communication overhead of Downpour SGD

- Limit each model replica to request updated parameters only every $n_{\text{fetch}}$ steps and send updated gradient values only every $n_{\text{push}}$ steps
- where $n_{\text{fetch}}$ might not be equal to $n_{\text{push}}$

- Traditional distributed SGD
  - $n_{\text{fetch}} = n_{\text{push}} = 1$

Fault tolerance with Downpour SGD

- More robust to machines failures
- Synchronous SGD with failures
  - If one machine fails, entire training process is delayed
- Asynchronous SGD with failures
  - If one machine fails, other model replica will continue processing training and updating model parameters

Failures vs. Stochasticity

- No guarantee that:
  - There were the same number of updates in the each shard of parameters
  - There were updates applied in the same order
- There are subtle inconsistencies in the timestamps of parameters
- Relaxing consistency requirement is effective to enhance stochasticity

Improving Downpour SGD with Adagrad

- Adagrad
  - Adaptive learning rate procedure
  - Let $\eta_i, K$ be the learning rate of the $i$-th parameter at iteration $K$ and $\Delta w_i$ its gradient
  - We set:
    $$\eta_i, K = \frac{\gamma}{\sqrt{\sum_{j=1}^{K} \Delta w_i^2}}$$

- These learning rates are computed only from the summed squared gradients of each parameter
- Adagrad can be implemented locally within each parameter server shard

Improving Downpour SGD with Adagrad

- $\eta_i, K = \frac{\gamma}{\sqrt{\sum_{j=1}^{K} \Delta w_i^2}}$
- The value of $\gamma$
  - The constant scaling factor for all learning rates
  - Generally larger than the best fixed learning rate used without Adagrad
- The use of Adagrad
  - Extends the maximum number of model replicas that can productively work simultaneously
  - Eliminates stability concerns
  - Combined with a practice of "warmstarting" model training with only a single model replica before unleashing the other replicas

http://www.cs.colostate.edu/~cs535