# Accelerating SGD using Predictive Variance Reduction
# Parallelized SGD
# Discussion

Zhuoyan Xu

March 5, 2019

# Gradient Descent and SGD

- Traditional gradient descent:

$$\omega^{(t)} = \omega^{(t-1)} - \frac{\eta_t}{n} \sum_{i=1}^{n} \nabla \psi_i(\omega^{(t-1)})$$

  Gradient descent requires evaluation of n derivatives, which brings in SGD.

- SGD: randomly select $i_t$ from samples:

$$\omega^{(t)} = \omega^{(t-1)} - \eta_t \nabla \psi_{i_t}(\omega^{(t-1)})$$

- The conditional expectations are same, where SGD's computation cost is 1/n of gradient descent. However, randomness also introduce variance.

# SVRG:Stochastic variance reduced gradient

- The tug-of-war between large computation per iteration and fast convergence for gradient descent and small computation per iteration and slow convergence for SGD. This dilemma brings new method of SGD.
- The goal is to increase the convergence rate of SGD. it needs to reduce variance in order to use larger learning rate $\eta_t$.
- SAG(stochastic average gradient)[Le Roux 2012] and SDCA(stochastic dual coordinate ascent)[Shalev-Schwarz Zhang 2012] were proposed, which were suitable for smooth and strong convex condition. And both methods need to store all gradients.

# SVRG:Stochastic variance reduced gradient

- SVDG conduct a variance reduction without the storage of intermediate gradients. It can obtain the same convergence rate as SAG and SDCA when dealing with strong convex and smooth problems. When deal with nonconvex problems(such as neural networks), the asymptotically variance of SGD goes to zero under some assumptions.
- Take a snapshot of $\widetilde{\omega}$ after every m iterations and take average:

$$\widetilde{\mu} = \frac{1}{n} \sum_{i=1}^{n} \nabla \psi_i(\widetilde{\omega})$$

Then we have update rule:

$$\omega^{(t)} = \omega^{(t-1)} - \eta_t(\nabla \psi_i(\omega^{(t-1)}) - \nabla \psi_{i_t}(\widetilde{\omega}) + \widetilde{\mu})$$

# The practical usage

- When under strong convex and smooth condition, SVRG has better performance than SGD.
- When dealing with non-convex problem such as neural networks, it is useful to use SGD find a general location quickly, then use SVRG to accelerating the local convergence rate to minimum.

# Parallel SGD

**Algorithm 1** $\mathrm{SGD}(\{c^1, \ldots, c^m\}, T, \eta, w_0)$

**for** $t = 1$ **to** $T$ **do**
  Draw $j \in \{1 \ldots m\}$ uniformly at random.
  $w_t \leftarrow w_{t-1} - \eta \partial_w c^j(w_{t-1})$.
**end for**
**return** $w_T$.

**Algorithm 2** $\mathrm{ParallelSGD}(\{c^1, \ldots c^m\}, T, \eta, w_0, k)$

**for all** $i \in \{1, \ldots k\}$ **parallel do**
  $v_i = \mathrm{SGD}(\{c^1, \ldots c^m\}, T, \eta, w_0)$ on client
**end for**
Aggregate from all computers $v = \frac{1}{k} \sum_{i=1}^{k} v_i$ and **return** $v$

- A direct implementation of the algorithms above would place every example on every machine: however, if T is much less than m, then it is only necessary for a machine to have access to the data it actually touches.

# Parallel SGD

**Algorithm 3** SimuParallelSGD(Examples $\{c^1, \dots c^m\}$, Learning Rate $\eta$, Machines $k$)

Define $T = \lfloor m/k \rfloor$
Randomly partition the examples, giving $T$ examples to each machine.
**for all** $i \in \{1, \dots k\}$ **parallel do**
  Randomly shuffle the data on machine $i$.
  Initialize $w_{i,0} = 0$.
  **for all** $t \in \{1, \dots T\}$: **do**
    Get the $t$th example on the $i$th machine (this machine), $c^{i,t}$
    $w_{i,t} \leftarrow w_{i,t-1} - \eta \partial_w c^i(w_{i,t-1})$
  **end for**
**end for**
Aggregate from all computers $v = \frac{1}{k} \sum_{i=1}^{k} w_{i,t}$ and **return** $v$.

One can consider the actual data in the real dataset to be a subset of a virtually infinite set. And drawing with replacement on actual dataset and drawing without replacement on the infinite dataset can both be simulated by shuffling the real data and accessing it sequentially.