Large-scale Stochastic Optimization

11-741/641/441 (Spring 2016)

Hanxiao Liu
hanxiaol@cs.cmu.edu

March 24, 2016

Outline

1. Gradient Descent (GD)
2. Stochastic Gradient Descent (SGD)
   - Formulation
   - Comparisons with GD
3. Useful large-scale SGD solvers
   - Support Vector Machines
   - Matrix Factorization
4. Random topics
   - Variance reduction
   - Implementation trick
   - Other variants

Outline

1. Gradient Descent (GD)
2. Stochastic Gradient Descent (SGD)
   - Formulation
   - Comparisons with GD
3. Useful large-scale SGD solvers
   - Support Vector Machines
   - Matrix Factorization
4. Random topics
   - Variance reduction
   - Implementation trick
   - Other variants

Outline

1. Gradient Descent (GD)
2. Stochastic Gradient Descent (SGD)
   - Formulation
   - Comparisons with GD
3. Useful large-scale SGD solvers
   - Support Vector Machines
   - Matrix Factorization
4. Random topics
   - Variance reduction
   - Implementation trick
   - Other variants

Outline

1. Gradient Descent (GD)
2. Stochastic Gradient Descent (SGD)
    - Formulation
    - Comparisons with GD
3. Useful large-scale SGD solvers
    - Support Vector Machines
    - Matrix Factorization
4. Random topics
    - Variance reduction
    - Implementation trick
    - Other variants

Risk Minimization

$\{(x_i, y_i)\}_{i=1}^n$: training data $\overset{i.i.d.}{\sim} \mathcal{D}$.

$$\min_f \underbrace{\mathbb{E}_{(x,y)\sim\mathcal{D}} \ell\left(f\left(x\right), y\right)}_{\text{True risk}} \implies \min_f \underbrace{\frac{1}{n}\sum_{i=1}^{n} \ell\left(f\left(x_i\right), y_i\right)}_{\text{Empirical risk}} \quad (1)$$

$$\implies \min_w \frac{1}{n}\sum_{i=1}^{n} \ell\left(f_w\left(x_i\right), y_i\right) \quad (2)$$

Algorithm	$\ell\left(f_w(x_i), y_i\right)$
Logistic Regression	$\ln\left(1 + e^{-y_i w^\top x_i}\right) + \frac{\lambda}{2}\|w\|^2$
SVMs	$\max\left(0, 1 - y_i w^\top x_i\right) + \frac{\lambda}{2}\|w\|^2$

Risk Minimization

$\{(x_i, y_i)\}_{i=1}^n$: training data $\overset{i.i.d.}{\sim} \mathcal{D}$.

$$\min_f \underbrace{\mathbb{E}_{(x,y)\sim\mathcal{D}} \ell\left(f\left(x\right), y\right)}_{\text{True risk}} \implies \min_f \underbrace{\frac{1}{n}\sum_{i=1}^{n} \ell\left(f\left(x_i\right), y_i\right)}_{\text{Empirical risk}} \quad (1)$$

$$\implies \min_w \frac{1}{n}\sum_{i=1}^{n} \ell\left(f_w\left(x_i\right), y_i\right) \quad (2)$$

Algorithm	$\ell\left(f_w(x_i), y_i\right)$
Logistic Regression	$\ln\left(1 + e^{-y_i w^\top x_i}\right) + \frac{\lambda}{2}\|w\|^2$
SVMs	$\max\left(0, 1 - y_i w^\top x_i\right) + \frac{\lambda}{2}\|w\|^2$

Gradient Descent (GD)

$\ell\left(f_w\left(x_i\right), y_i\right) \overset{def}{=} \ell_i\left(w\right)$, $\ell\left(w\right) \overset{def}{=} \frac{1}{n}\sum_{i=1}^{n} \ell_i\left(w\right)$

Training objective:

$$\min_w \quad \ell(w) \tag{3}$$

Gradient update: $w^{(k)} = w^{(k-1)} - \eta_k \nabla \ell\left(w^{(k-1)}\right)$

- η_k: pre-specified or determined via backtracking
- Can be easily generalized to handle nonsmooth loss
  1. ~~Gradient~~ Subgradient
  2. Proximal gradient method (for structured $\ell\left(w\right)$)

Question of interest: How fast does GD converge?

Gradient Descent (GD)

$\ell\left(f_w\left(x_i\right), y_i\right) \overset{def}{=} \ell_i\left(w\right)$, $\ell\left(w\right) \overset{def}{=} \frac{1}{n}\sum_{i=1}^{n} \ell_i\left(w\right)$

Training objective:

$$\min_{w} \ \ell(w) \tag{3}$$

Gradient update: $w^{(k)} = w^{(k-1)} - \eta_k \nabla \ell\left(w^{(k-1)}\right)$

- η_k: pre-specified or determined via backtracking
- Can be easily generalized to handle nonsmooth loss
  1. ~~Gradient~~ Subgradient
  2. Proximal gradient method (for structured $\ell\left(w\right)$)

Question of interest: How fast does GD converge?

Gradient Descent (GD)

$\ell\left(f_w\left(x_i\right), y_i\right) \stackrel{def}{=} \ell_i\left(w\right)$, $\ell\left(w\right) \stackrel{def}{=} \frac{1}{n}\sum_{i=1}^{n} \ell_i\left(w\right)$

Training objective:

$$\min_{w} \ \ell(w) \tag{3}$$

Gradient update: $w^{(k)} = w^{(k-1)} - \eta_k \nabla \ell\left(w^{(k-1)}\right)$

- η_k: pre-specified or determined via backtracking
- Can be easily generalized to handle nonsmooth loss
    1. ~~Gradient~~ Subgradient
    2. Proximal gradient method (for structured $\ell\left(w\right)$)

Question of interest: How fast does GD converge?

Convergence

Theorem (GD convergence)

If ℓ is both convex and differentiable [1]

$$\ell\left(w^{(k)}\right) - \ell\left(w^*\right) \le \begin{cases} \frac{\|w^{(0)} - w^*\|_2^2}{2\eta k} = O\left(\frac{1}{k}\right) & \textit{in general} \\ \frac{c^k L \|w^{(0)} - w^*\|_2^2}{2} = O\left(c^k\right) & \ell \textit{ is strongly convex} \end{cases} \tag{4}$$

– To achieve $\ell\left(x^{(k)}\right) - \ell\left(x^*\right) \le \rho$, GD needs $O\left(\frac{1}{\rho}\right)$ iterations in general, and $O\left(\log\left(\frac{1}{\rho}\right)\right)$ iterations with strong convexity.

[1] the step size η must be no larger than $\frac{1}{L}$, where L is the Lipschitz constant satisfying $\|\nabla \ell(a) - \nabla \ell(b)\|_2 \le L\|a - b\|_2 \;\; \forall a, b$

Convergence

Theorem (GD convergence)

If ℓ is both convex and differentiable [1]

$$\ell\left(w^{(k)}\right)-\ell\left(w^{*}\right) \leq \begin{cases} \frac{\|w^{(0)}-w^{*}\|_2^2}{2\eta k} = O\left(\frac{1}{k}\right) & \textit{in general} \\ \frac{c^k L\|w^{(0)}-w^{*}\|_2^2}{2} = O\left(c^k\right) & \ell \textit{ is strongly convex} \end{cases} \tag{4}$$

– To achieve $\ell\left(x^{(k)}\right) - \ell\left(x^{*}\right) \leq \rho$, GD needs $O\left(\frac{1}{\rho}\right)$ iterations in general, and $O\left(\log\left(\frac{1}{\rho}\right)\right)$ iterations with strong convexity.

[1] the step size η must be no larger than $\frac{1}{L}$, where L is the Lipschitz constant satisfying $\|\nabla\ell\left(a\right) - \nabla\ell\left(b\right)\|_2 \leq L\|a-b\|_2 \ \forall a, b$

GD Efficiency

Why not happy with GD?

- Fast convergence $\neq$ high efficiency.

$$w^{(k)} = w^{(k-1)} - \eta_k \nabla \ell \left(w^{(k-1)} \right) \tag{5}$$

$$= w^{(k-1)} - \eta_k \nabla \left[\frac{1}{n} \sum_{i=1}^{n} \ell_i \left(w^{(k-1)} \right) \right] \tag{6}$$

- Per-iteration complexity $= O(n)$ (extremely large)
  - A single cycle of all the data may take forever.
- Cheaper GD? - SGD

GD Efficiency

Why not happy with GD?

- Fast convergence $\neq$ high efficiency.

$$w^{(k)} = w^{(k-1)} - \eta_k \nabla \ell \left(w^{(k-1)} \right) \tag{5}$$

$$= w^{(k-1)} - \eta_k \nabla \left[\frac{1}{n} \sum_{i=1}^{n} \ell_i \left(w^{(k-1)} \right) \right] \tag{6}$$

- Per-iteration complexity $= O(n)$ (extremely large)
  - A single cycle of all the data may take forever.
- Cheaper GD? - SGD

Stochastic Gradient Descent

Approximate the full gradient via an unbiased estimator

$$w^{(k)} = w^{(k-1)} - \eta_k \nabla \Big(\frac{1}{n} \sum_{i=1}^{n} \ell_i \left(w^{(k-1)} \right) \Big) \tag{7}$$

$$\approx \underbrace{w^{(k-1)} - \eta_k \nabla \Big(\frac{1}{|B|} \sum_{i \in B} \ell_i \left(w^{(k-1)} \right) \Big)}_{\text{mini-batch SGD } ^2} \quad B \overset{unif}{\sim} \{1, 2, \dots n\} \tag{8}$$

$$\approx \underbrace{w^{(k-1)} - \eta_k \nabla \ell_i \left(w^{(k-1)} \right)}_{\text{pure SGD}} \quad i \overset{unif}{\sim} \{1, 2, \dots n\} \tag{9}$$

Trade-off: lower computation cost v.s. larger variance

[2] When using GPU, $|B|$ usually depends on the memory budget.

GD v.s. SGD

For strongly convex $\ell(w)$, according to [Bottou, 2012]

Optimizer	GD	SGD	Winner
Time per-iteration	$O(n)$	$O(1)$	SGD
Iterations to accuracy ρ	$O\left(\log\left(\frac{1}{\rho}\right)\right)$	$\tilde{O}\left(\frac{1}{\rho}\right)$	GD
Time to accuracy ρ	$O\left(n\log\frac{1}{\rho}\right)$	$\tilde{O}\left(\frac{1}{\rho}\right)$	Depends
Time to "generalization error" ϵ	$O\left(\frac{1}{\epsilon^{1/\alpha}}\log\frac{1}{\epsilon}\right)$	$\tilde{O}\left(\frac{1}{\epsilon}\right)$	SGD

where $\frac{1}{2} \leq \alpha \leq 1$

SVMs Solver: Pegasos

[Shalev-Shwartz et al., 2011]

Recall

$$\ell_i(w) = \max\left(0, 1 - y_i w^\top x_i\right) + \frac{\lambda}{2}\|w\|^2 \tag{10}$$

$$= \begin{cases} \frac{\lambda}{2}\|w\|^2 & y_i w^\top x_i \geq 1 \\ 1 - y_i w^\top x_i + \frac{\lambda}{2}\|w\|^2 & y_i w^\top x_i < 1 \end{cases} \tag{11}$$

Therefore

$$\nabla \ell_i(w) = \begin{cases} \lambda w & y_i w^\top x_i \geq 1 \\ \lambda w - y_i x_i & y_i w^\top x_i < 1 \end{cases} \tag{12}$$

SVMs Solver: Pegasos

[Shalev-Shwartz et al., 2011]

Recall

$$\ell_i(w) = \max\left(0, 1 - y_i w^\top x_i\right) + \frac{\lambda}{2}\|w\|^2 \tag{10}$$

$$= \begin{cases} \frac{\lambda}{2}\|w\|^2 & y_i w^\top x_i \geq 1 \\ 1 - y_i w^\top x_i + \frac{\lambda}{2}\|w\|^2 & y_i w^\top x_i < 1 \end{cases} \tag{11}$$

Therefore

$$\nabla \ell_i(w) = \begin{cases} \lambda w & y_i w^\top x_i \geq 1 \\ \lambda w - y_i x_i & y_i w^\top x_i < 1 \end{cases} \tag{12}$$

SVMs in 10 Lines

Algorithm 1: Pegasos: SGD solver for SVMs

Input: n, λ, T;
Initialization: $w \leftarrow 0$;
for $k = 1, 2, \ldots, T$ **do**
  $i \overset{uni}{\sim} \{1, 2, \ldots n\}$;
  $\eta_k \leftarrow \frac{1}{\lambda k}$;
  if $y_i {w^{(k)}}^\top x_i < 1$ **then**
    $w^{(k+1)} \leftarrow w^{(k)} - \eta_k \left(\lambda w^{(k)} - y_i x_i\right)$
  else
    $w^{(k+1)} \leftarrow w^{(k)} - \eta_k \lambda w^{(k)}$
  end
end
Output: $w^{(T+1)}$

Empirical Comparisons

SGD v.s. batch solvers[3] on RCV1

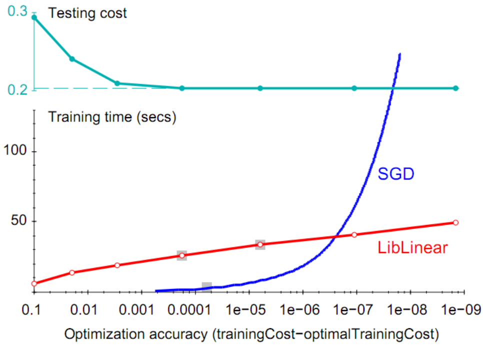
#Features	#Training examples
47, 152	781, 265

Algorithm	Time (secs)	Primal Obj	Test Error
SMO (SVM^{light})	$\approx 16,000$	0.2275	6.02%
Cutting Plane (SVM^{perf})	≈ 45	0.2275	6.02%
SGD	< 1	0.2275	6.02%

Where is the magic?

[3] http://leon.bottou.org/projects/sgd

The Magic



- SGD takes a long time to accurately solve the problem.
- There's no need to solve the problem super accurately in order to get good generalization ability.

[3] http://leon.bottou.org/slides/largescale/lstut.pdf

SGD for Matrix Factorization

The idea of SGD can be trivially extended to MF [4]

$$\ell(U,V) = \frac{1}{|\mathcal{O}|} \sum_{(a,b)\in\mathcal{O}} \underbrace{\ell_{a,b}(u_a, v_b)}_{\text{e.g. } \left(r_{ab} - u_a^\top v_b\right)^2} \tag{13}$$

SGD updating rule: for each user-item pair $(a, b) \sim \mathcal{O}$

$$u_a^{(k)} = u_a^{(k-1)} - \eta_k \nabla \ell_{a,b}\left(u_a^{(k-1)}\right) \tag{14}$$

$$v_b^{(k)} = v_b^{(k-1)} - \eta_k \nabla \ell_{a,b}\left(v_b^{(k-1)}\right) \tag{15}$$

Buildingblock for distributed SGD for MF

[4] We omit the regularization term for simplicity

SGD for Matrix Factorization

The idea of SGD can be trivially extended to MF [4]

$$\ell(U,V) = \frac{1}{|\mathcal{O}|} \sum_{(a,b)\in\mathcal{O}} \underbrace{\ell_{a,b}(u_a, v_b)}_{\text{e.g. } \left(r_{ab} - u_a^\top v_b\right)^2} \tag{13}$$

SGD updating rule: for each user-item pair $(a,b) \sim \mathcal{O}$

$$u_a^{(k)} = u_a^{(k-1)} - \eta_k \nabla \ell_{a,b}\left(u_a^{(k-1)}\right) \tag{14}$$

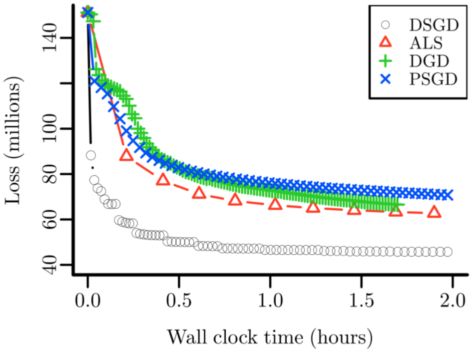
$$v_b^{(k)} = v_b^{(k-1)} - \eta_k \nabla \ell_{a,b}\left(v_b^{(k-1)}\right) \tag{15}$$

Buildingblock for distributed SGD for MF

[4] We omit the regularization term for simplicity

Empirical Comparisons

On Netflix [Gemulla et al., 2011]



DSGD Distributed SGD

ALS Alternating least squares

- one of the state-of-the-art batch solvers

DGD Distributed GD

SGD Revisited

Can we even do better?

Bottleneck of SGD: high variance in $\nabla \ell_i(w)$

- Less effective gradient steps
- The existence of variance $\implies \lim_{k\to\infty} \eta_k = 0$ for convergence $\implies$ slower progress

Variance reduction—SVRG [Johnson and Zhang, 2013], SAG, SDCA ...

Stochastic Variance Reduced Gradient

$\tilde{w}$ - a snapshot of w (to be updated every few cycles)

$\tilde{\mu}$ - $\frac{1}{n}\sum_{i=1}^{n} \nabla \ell_i(\tilde{w})$

Key idea - use $\nabla \ell_i(\tilde{w})$ to "cancel" the volatility in $\nabla \ell_i(w)$

$$\frac{1}{n}\sum_{i=1}^{n} \nabla \ell_i(w) = \frac{1}{n}\sum_{i=1}^{n} \nabla \ell_i(w) - \tilde{\mu} + \tilde{\mu} \tag{16}$$

$$= \frac{1}{n}\sum_{i=1}^{n} \nabla \ell_i(w) - \frac{1}{n}\sum_{i=1}^{n} \nabla \ell_i(\tilde{w}) + \tilde{\mu} \tag{17}$$

$$\approx \nabla \ell_i(w) - \nabla \ell_i(\tilde{w}) + \tilde{\mu} \quad i \sim \{1, 2, \ldots, n\} \tag{18}$$

A desirable property: $\nabla \ell_i\left(w^{(k)}\right) - \nabla \ell_i(\tilde{w}) + \tilde{\mu} \to 0$

Stochastic Variance Reduced Gradient

$\tilde{w}$ - a snapshot of w (to be updated every few cycles)

$\tilde{\mu}$ - $\frac{1}{n}\sum_{i=1}^{n} \nabla \ell_i(\tilde{w})$

Key idea - use $\nabla \ell_i(\tilde{w})$ to "cancel" the volatility in $\nabla \ell_i(w)$

$$\frac{1}{n}\sum_{i=1}^{n} \nabla \ell_i(w) = \frac{1}{n}\sum_{i=1}^{n} \nabla \ell_i(w) - \tilde{\mu} + \tilde{\mu} \tag{16}$$

$$= \frac{1}{n}\sum_{i=1}^{n} \nabla \ell_i(w) - \frac{1}{n}\sum_{i=1}^{n} \nabla \ell_i(\tilde{w}) + \tilde{\mu} \tag{17}$$

$$\approx \nabla \ell_i(w) - \nabla \ell_i(\tilde{w}) + \tilde{\mu} \quad i \sim \{1, 2, \ldots, n\} \tag{18}$$

A desirable property: $\nabla \ell_i\left(w^{(k)}\right) - \nabla \ell_i(\tilde{w}) + \tilde{\mu} \to 0$

Results

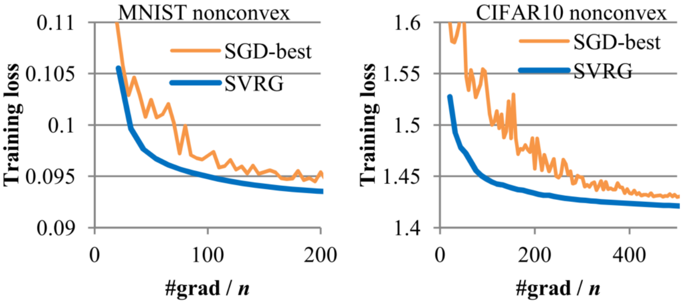


Image classification using neural networks
[Johnson and Zhang, 2013]

Implementation trick

For $\ell_i(w) = \psi_i(w) + \frac{\lambda}{2}\|w\|^2$

$$w^{(k+1)} \leftarrow \underbrace{(1-\eta\lambda)\, w^{(k)}}_{\text{shrink } w} - \eta \underbrace{\nabla\psi_i\left(w^{(k)}\right)}_{\text{highly sparse}} \tag{19}$$

The shrinking operations takes $O(p)$ – not happy

Trick [5]: recast w as $w = c \cdot w'$

$$c^{(k+1)} \cdot w'^{(k+1)} \leftarrow \underbrace{(1-\eta\lambda)\, c^{(k)}}_{\text{scalar update}} \cdot \underbrace{\left[w'^{(k)} - \frac{\eta\psi_i\left(c^{(k)} w'^{(k)}\right)}{(1-\eta\lambda)\, c^{(k)}} \right]}_{\text{sparse update}} \tag{20}$$

More SGD tricks can be found at [Bottou, 2012]

[5] http://blog.smola.org/post/940672544/fast-quadratic-regularization-for-online-learning

Implementation trick

For $\ell_i(w) = \psi_i(w) + \frac{\lambda}{2}\|w\|^2$

$$w^{(k+1)} \leftarrow \underbrace{(1-\eta\lambda)\,w^{(k)}}_{\text{shrink } w} - \eta \underbrace{\nabla\psi_i\left(w^{(k)}\right)}_{\text{highly sparse}} \tag{19}$$

The shrinking operations takes $O(p)$ – not happy
Trick [5]: recast w as $w = c \cdot w'$

$$c^{(k+1)} \cdot {w'}^{(k+1)} \leftarrow \underbrace{(1-\eta\lambda)\,c^{(k)}}_{\text{scalar update}} \cdot \underbrace{\left[{w'}^{(k)} - \frac{\eta\psi_i\left(c^{(k)}{w'}^{(k)}\right)}{(1-\eta\lambda)\,c^{(k)}}\right]}_{\text{sparse update}} \tag{20}$$

More SGD tricks can be found at [Bottou, 2012]

[5] http://blog.smola.org/post/940672544/fast-quadratic-regularization-for-online-learning

Popular SGD Variants

A non-exhaustive list

1. AdaGrad [Duchi et al., 2011]
2. Momentum [Rumelhart et al., 1988]
3. Nesterov's method [Nesterov et al., 1994]
4. AdaDelta: AdaGrad refined [Zeiler, 2012]
5. Rprop & Rmsprop [Tieleman and Hinton, 2012]: Ignoring the magnitude of gradient

All are empirically found effective in solving nonconvex problems (e.g., deep neural nets).

Demos [6]: Animation 0, 1, 2, 3

[6] https://www.reddit.com/r/MachineLearning/comments/2gopfa/visualizing_gradient_optimization_techniques/cklhott

Popular SGD Variants

A non-exhaustive list

1. AdaGrad [Duchi et al., 2011]
2. Momentum [Rumelhart et al., 1988]
3. Nesterov's method [Nesterov et al., 1994]
4. AdaDelta: AdaGrad refined [Zeiler, 2012]
5. Rprop & Rmsprop [Tieleman and Hinton, 2012]: Ignoring the magnitude of gradient

All are empirically found effective in solving nonconvex problems (e.g., deep neural nets).

Demos [6]: Animation 0, 1, 2, 3

[6] https://www.reddit.com/r/MachineLearning/comments/2gopfa/visualizing_gradient_optimization_techniques/cklhott

Summary

Today's talk

1. GD - expensive, accurate gradient evaluation
2. SGD - cheap, noisy gradient evaluation
3. SGD-based solvers (SVMs, MF)
4. Variance reduction techniques

Remarks about SGD

- extremely handy for large problems
- only one of many handy tools
  - alternatives: quasi-Newton (BFGS), Coordinate descent, ADMM, CG, etc.
  - depending on the problem structure

Summary

Today's talk

1. GD - expensive, accurate gradient evaluation
2. SGD - cheap, noisy gradient evaluation
3. SGD-based solvers (SVMs, MF)
4. Variance reduction techniques

Remarks about SGD

- extremely handy for large problems
- only one of many handy tools
    - alternatives: quasi-Newton (BFGS), Coordinate descent, ADMM, CG, etc.
    - depending on the problem structure

Reference I

Bordes, A., Bottou, L., and Gallinari, P. (2009).
Sgd-qn: Careful quasi-newton stochastic gradient descent.
The Journal of Machine Learning Research, 10:1737–1754.

Bottou, L. (2012).
Stochastic gradient descent tricks.
In *Neural Networks: Tricks of the Trade*, pages 421–436. Springer.

Duchi, J., Hazan, E., and Singer, Y. (2011).
Adaptive subgradient methods for online learning and stochastic optimization.
The Journal of Machine Learning Research, 12:2121–2159.

Gemulla, R., Nijkamp, E., Haas, P. J., and Sismanis, Y. (2011).
Large-scale matrix factorization with distributed stochastic gradient descent.
In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 69–77. ACM.

Johnson, R. and Zhang, T. (2013).
Accelerating stochastic gradient descent using predictive variance reduction.
In *Advances in Neural Information Processing Systems*, pages 315–323.

Reference II

- Nesterov, Y., Nemirovskii, A., and Ye, Y. (1994).
  Interior-point polynomial algorithms in convex programming, volume 13.
  SIAM.
- Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1988).
  Learning representations by back-propagating errors.
  Cognitive modeling, 5:3.
- Shalev-Shwartz, S., Singer, Y., Srebro, N., and Cotter, A. (2011).
  Pegasos: Primal estimated sub-gradient solver for svm.
  Mathematical programming, 127(1):3–30.
- Tieleman, T. and Hinton, G. (2012).
  Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude.
  COURSERA: Neural Networks for Machine Learning, 4.
- Zeiler, M. D. (2012).
  Adadelta: An adaptive learning rate method.
  arXiv preprint arXiv:1212.5701.
- Zhang, T.
  Modern optimization techniques for big data machine learning.