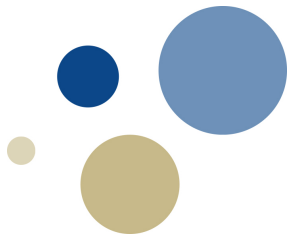




Norwegian University of
Science and Technology



Large-scale machine learning with stochastic gradient descent

Bjarne Grimstad

Department of Engineering Cybernetics, NTNU

October 5, 2019

Outline

1. Optimization in machine learning (ML)
2. Gradient descent
3. Stochastic gradient descent (SGD)
4. Implementations and variations of SGD



Optimization in machine learning



Most ML problems can be stated as a finite sum optimization:

$$\min_{\theta} f(\theta), \text{ where } f(\theta) = \frac{1}{n} \sum_{i=1}^n f_i(\theta)$$

For state-of-the-art ML, this is a high-dimensional and non-convex problem (\mathcal{NP} -hard). Local minima can be found by gradient descent, Newton methods, subgradient methods, and other methods!

Big datasets



Training data: $\{(x_1, y_1), \dots, (x_n, y_n)\} \subset \mathbb{R}^d \times \mathcal{Y}$

Large-scale ML: d and/or n are large!

- d : number of input/explanatory variables
- n : number of training data points / samples

For later: we compactly write (X, y) , where $X \in \mathbb{R}^{n \times d}$ and $y \in \mathcal{Y}^n$.

Supervised learning (classification and regression)

Consider a function $m(\cdot; \theta) : \mathbb{R}^d \rightarrow \mathcal{Y}$, parameterized by $\theta \in \mathbb{R}^p$.

And a loss function $\ell : \mathcal{Y} \times \mathcal{Y} \rightarrow \mathbb{R}$.

The prototypical supervised learning problem is to minimize the **empirical risk**:

$$\min_{\theta} \frac{1}{n} \sum_{i=1}^n \ell(y_i, m(x_i; \theta)) = \min_{\theta} \frac{1}{n} \sum_{i=1}^n f_i(\theta). \quad (1)$$

Examples of ML problems

Least-squares

$$\frac{1}{n} \|y - X\theta\|_2^2 = \frac{1}{n} \sum_{i=1}^n (y_i - x_i^\top \theta)^2 = \frac{1}{n} \sum_{i=1}^n f_i(\theta)$$

Lasso / L_1 -regularized least-squares

$$\frac{1}{n} \|y - X\theta\|_2^2 + \lambda \|\theta\|_1 = \frac{1}{n} \sum_{i=1}^n (y_i - x_i^\top \theta)^2 + \lambda \sum_{j=1}^p |\theta_j|$$

Logistic regression

$$-\frac{1}{n} \sum_{i=1}^n y_i \log(\sigma(x_i^\top \theta)) + (1 - y_i) \log(1 - \sigma(x_i^\top \theta))$$

Examples of ML problems

Support vector machine (SVM)

$$\frac{1}{n} \sum_{i=1}^n \max \left(0, 1 - y_i (x_i^\top \theta) \right) + \lambda \|\theta\|_2^2$$

Deep neural network with L_2 -regularization

$$\frac{1}{n} \sum_{i=1}^n (y_i - \text{DNN}(x_i; \theta))^2 + \lambda \|\theta\|_2^2$$

Applications of ML

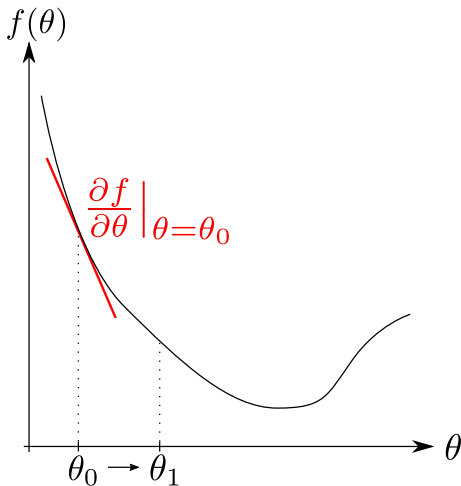
SGD is used to train models that drive many important applications and services in today's society. It is important that we understand these algorithms.

- **Prediction:** credit scoring, housing value, branch prediction in CPUs, ...
- **Computer vision:** recognition, image reconstruction, self-driving cars, ...
- **Recommender systems:** media, products, ...
- **Clustering analysis:** spam filtering, fake news detection, ...
- **Natural language processing:** speech recognition, machine translation, personal assistants, ...



Hi, how can I help?

Gradient descent



We search for a minimum of $f : \mathbb{R}^p \rightarrow \mathbb{R}$ by moving iteratively in the direction of the negative gradient:

$$\theta_{k+1} = \theta_k - \alpha \nabla_{\theta} f(\theta_k)$$

The step length α is called **learning rate** in machine learning.

Gradient descent



Require: θ_0, α, K

1: $k \leftarrow 0$

2: **for** $k < K$ **do**

3: $\mathbf{g}_k \leftarrow \nabla_{\theta} f(\theta_k)$

▷ Compute gradient

4: $\theta_{k+1} = \theta_k - \alpha \mathbf{g}_k$

▷ Update parameters

5: $k \leftarrow k + 1$

6: **end for**

7: **return** θ_K

Note: θ_0 is typically randomly initialized and K is the number of iterations.

Gradient descent

Require: θ_0, α, K

1: $k \leftarrow 0$

2: **for** $k < K$ **do**

3: $g_k \leftarrow \nabla_{\theta} f(\theta_k)$

4: $\theta_{k+1} = \theta_k - \alpha g_k$

5: $k \leftarrow k + 1$

6: **end for**

7: **return** θ_K

- ▷ Compute gradient
- ▷ Update parameters



Large-scale optimization

The basic gradient descent method is often called **batch gradient descent** since it computes $\nabla_{\theta} f(\theta)$ from all sample points. This requires $O(np)$ operations, which is expensive when n or p is large.

$$\nabla_{\theta} f(\theta) = \nabla_{\theta} \left(\frac{1}{n} \sum_{i=1}^n f_i(\theta) \right) = \frac{1}{n} \sum_{i=1}^n \nabla_{\theta} f_i(\theta)$$

To reduce computational load we can approximate the gradient:

$$\frac{1}{n} \sum_{i=1}^n \nabla_{\theta} f_i(\theta) \approx \nabla_{\theta} f_j(\theta),$$

where $j \in \{1, \dots, n\}$. The approximation (stochastic gradient) requires $O(p)$ operations.

Stochastic gradients

Two rules for choosing index i_k at iteration k :

- Randomized rule: choose $i_k \in \{1, \dots, n\}$ uniformly at random
- Cyclic rule: choose $i_k \in \{1, 2, \dots, n, 1, 2, \dots, n, \dots\}$

For the randomized rule, the stochastic gradient is an **unbiased estimate** of the gradient:

$$\mathbb{E}_{j \sim \mathcal{U}(1, n)} [\nabla_{\theta} f_j(\theta)] = \sum_{i=1}^n \nabla_{\theta} f_i(\theta) \frac{1}{n} = \nabla_{\theta} f(\theta).$$

Main appeal of **stochastic gradient descent**:

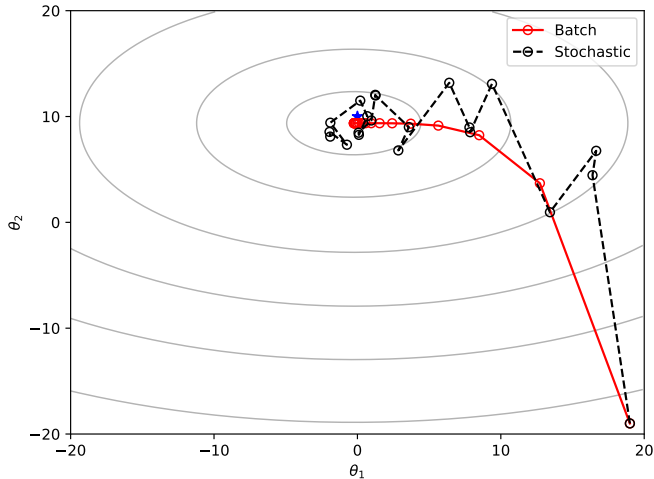
- Iteration cost is independent on n (number of samples)
- Can give big savings in terms of memory usage

Stochastic gradient descent

Require: θ_0, α, K

- 1: $k \leftarrow 0$
- 2: **for** $k < K$ **do**
- 3: $i_k \sim \mathcal{U}(1, n)$ ▷ Draw an index uniformly at random
- 4: $g_k \leftarrow \nabla_{\theta} f_{i_k}(\theta_k)$ ▷ Compute stochastic gradient
- 5: $\theta_{k+1} = \theta_k - \alpha g_k$ ▷ Update parameters
- 6: $k \leftarrow k + 1$
- 7: **end for**
- 8: **return** θ_K

Batch versus stochastic gradient descent



Linear regression with
 $n = 10$ and $p = 2$.

Rule of thumb for
stochastic methods:

- generally thrive
far from optimum
- generally struggle
close to optimum

Variance of stochastic gradients

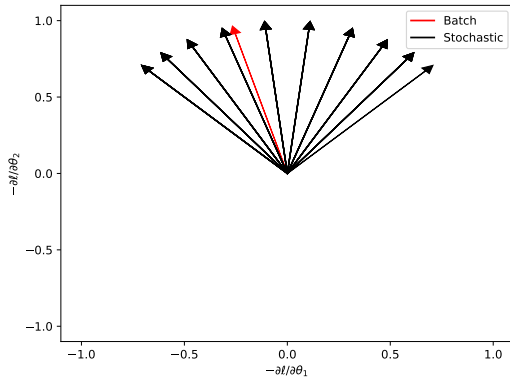
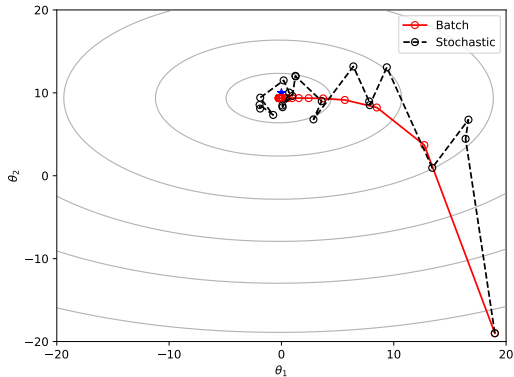


Figure: Normalized gradients at iteration 1 of SGD sequence

Variance of stochastic gradients

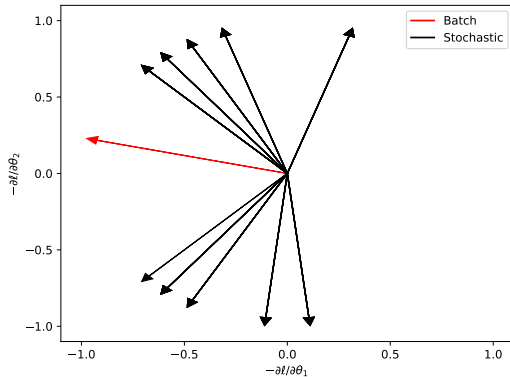
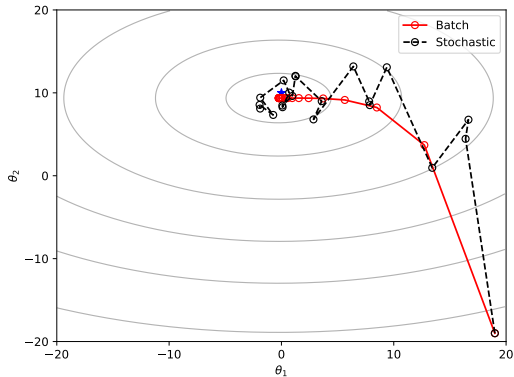


Figure: Normalized gradients at iteration 10 of SGD sequence

Region of confusion

For a strongly convex f , fixed step length α , and as $k \rightarrow \infty$:

$$\mathbb{E}[f(\theta_k) - f(\theta^*)] \leq \alpha M,$$

where M is a positive constant. SGD converges to a “region of confusion”.

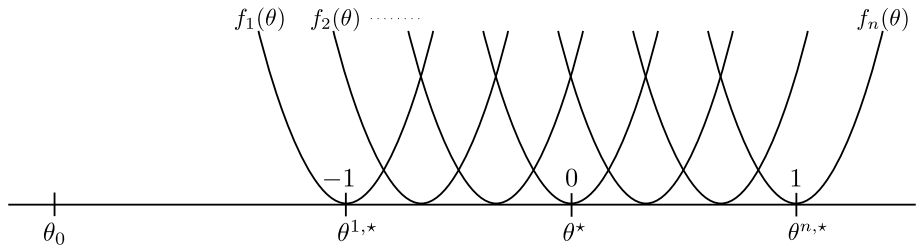


Figure: Region of confusion

Convergence of SGD for strongly convex functions

What happens if we use a diminishing step size? For example $\alpha_k = \alpha/k$?
It turns out that this ensures **sublinear convergence** for SGD [1]:

$$\mathbb{E}[f(\theta_k) - f(\theta^*)] = O(1/k).$$

Batch gradient descent enjoys **linear convergence**:

$$f(\theta_k) - f(\theta^*) \leq O(\rho^k), \quad \rho \in (0, 1).$$

Batch gradient descent converges faster, but what is the computational cost?

- Batch: total work for ϵ -optimality is $O(n \log(1/\epsilon))$
- SGD: total work for ϵ -optimality is $O(1/\epsilon)$

Mini-batch (stochastic) gradient descent

A **mini-batch** is a randomly chosen subset $I_k \subseteq \{1, \dots, n\}$ of size $|I_k| = b \ll n$.
The update rule for mini-batch SGD is:

$$\theta_{k+1} = \theta_k - \alpha \frac{1}{b} \sum_{i \in I_k} \nabla_{\theta} f_i(\theta_k)$$

The approximation is an unbiased estimate of the full gradient:

$$\mathbb{E} \left[\frac{1}{b} \sum_{i \in I_k} \nabla_{\theta} f_j(\theta) \right] = \nabla_{\theta} f(\theta).$$

Using mini-batches reduces the **variance** of our gradient estimate by a factor of $1/b$, but is also b times more expensive. Note: we can parallelize.

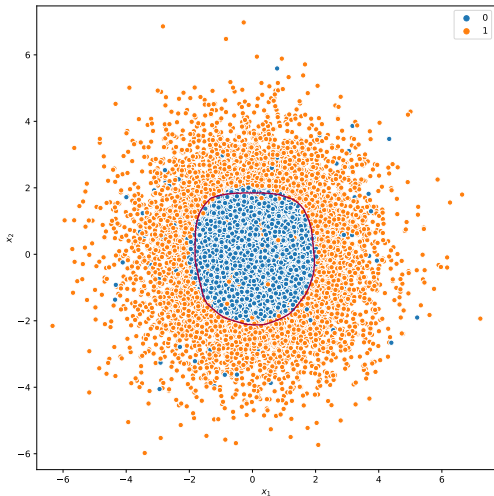
Example – binary classification



Binary classification problem with $n = 10,000$, $p = 401$ and $d = 2$.

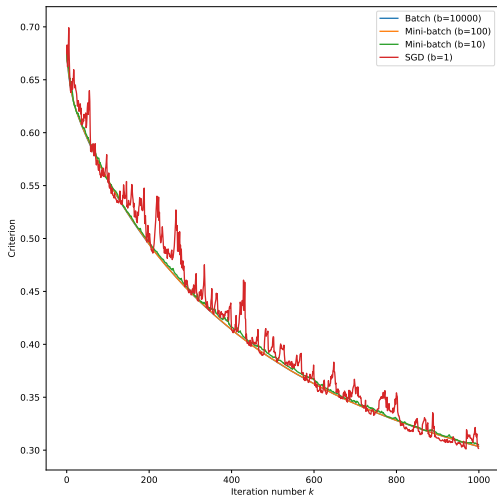
$$f(\theta) = -\frac{1}{n} \sum_{i=1}^n y_i \log(\sigma(\text{DNN}(x_i; \theta))) + (1 - y_i) \log(1 - \sigma(\text{DNN}(x_i; \theta)))$$

Example – binary classification



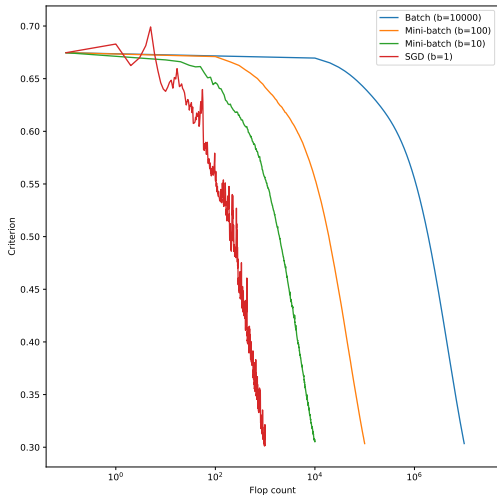
- 1,000 iterations with mini-batch gradient descent ($b=10$)
- The DNN maps (x_1, x_2) to a nonlinear decision boundary

Binary classification example – loss



Increasing batch size reduces the noise.

Binary classification example – flop count (log-scale)



Flop count for 1,000 iterations:

- One batch update costs $O(np)$
- One mini-batch update costs $O(bp)$
- One stochastic update costs $O(p)$

Intuitions – SGD for deep neural networks

- The **non-convex loss function** is poorly understood. As network size increase, so does the number of minima.
- For **over-parameterized networks** most stationary points are saddle points.
- SGD can be viewed as GD with an additive anisotropic noise term:
 - helps to escape saddle points [2]
 - avoids sharp (and poor) minima and tends to flat (and stable) minima [3]
 - has a **high probability of ϵ -convergence** [4]
- In many ML problems we don't care about optimizing to high accuracy, it doesn't pay off in terms of statistical performance.

Variations of SGD

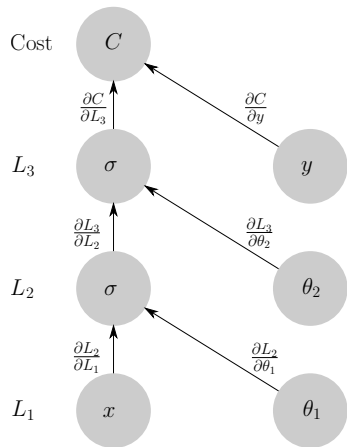


Variants of SGD can provide better stability and convergence properties:

- Adaptive learning rates (one per parameter)
- Momentum (exponentially decaying average of past gradients)
- Learning rate scheduling/decay
- Early stopping
- See AdaGrad, Adam, AdaMax, AdaDelta, SVRG, SAG, SAGA, etc.

See some cool visualizations at <http://ruder.io/optimizing-gradient-descent/index.html#visualizationofalgorithms>.

ML frameworks simplify implementation



- Computational graphs (flexible modeling)
- Tensor computations (parallel computing/GPU)
- Automatic differentiation (back-propagation)
- Optimization algorithms (SGD methods)

All ML frameworks implement SGD*

 TensorFlow

 PyTorch

 Keras

theano

 Microsoft
CNTK


Chainer

 scikit
learn

Caffe

 mxnet

 DL4J

 fast.ai

*Disclaimer: At least the ones listed here

Summary



- Most optimization problems in ML can be formulated as finite sum problems
- SGD can be very efficient in terms of computational cost and memory usage
- Diminishing step size and mini-batches can reduce gradient noise
- Over-parameterization and stochastic gradients help to optimize DNNs
- ML frameworks make implementation easy

References and further reading

- [1] Léon Bottou, Frank E. Curtis, and Jorge Nocedal. Optimization methods for large-scale machine learning. *SIAM Review*, 60(2):223–311, 2018.
- [2] Hadi Daneshmand, Jonas Kohler, Aurélien Lucchi, and Thomas Hofmann. Escaping saddles with stochastic gradients. In *35th International Conference on Machine Learning, ICML 2018*, volume 3, pages 1859–1884, 2018. ISBN 9781510867963.
- [3] Zhanxing Zhu, Jingfeng Wu, Bing Yu, Lei Wu, and Jinwen Ma. The Anisotropic Noise in Stochastic Gradient Descent: Its Behavior of Escaping from Sharp Minima and Regularization Effects. In *36th International Conference on Machine Learning*, pages 7654–7663, 2019.
- [4] Difan Zou, Yuan Cao, Dongruo Zhou, and Quanquan Gu. Stochastic Gradient Descent Optimizes Over-parameterized Deep ReLU Networks, 2018. URL <http://arxiv.org/abs/1811.08888>.

You can download this presentation at <https://github.com/bgrimstad/opt-courseware>