ECE 598: Machine Learning in Silicon
Fall 2017

Training via the Stochastic Gradient Descent Algorithm (SGD)

Naresh Shanbhag
Professor
Department of Electrical and Computer Engineering
University of Illinois at Urbana-Champaign
Optimization Methods for Large-Scale Machine Learning

Léon Bottou* Frank E. Curtis† Jorge Nocedal‡
June 16, 2016

https://arxiv.org/abs/1606.04838

• most widely used training algorithm in machine learning
• robust to data statistics and highly flexible, e.g., LMS-like algorithm for SVM training, backprop for deep neural networks
• recall: set-up for training
• assigning a class label $\hat{y} \in \{C_0, C_1, ..., C_{M-1}\}$ to data $x$
• prediction function $\rightarrow h(x; w) = \hat{y}$ (w is classifier parameter)
• class labels are discrete – binary or multi-class
• data $x$ can be discrete or continuous, scalar or vector
• first $\rightarrow$ need to train classifier to obtain $w$
Example - Channel Equalizer

- data: \( \mathbf{x}_k = [y[k], y[k - 1], \ldots, y[k - M + 1]]^T \)
- predicted label: \( \hat{c}[k] \in \{\pm 1\} \) (BPSK)
- prediction function: \( \hat{c}[k] = h(\mathbf{x}_k; \mathbf{w}) = \text{sign}(\mathbf{w}^T \mathbf{x}_k) \)
- training sample: \( (\mathbf{x}_k, c[k]) \)
- LMS algorithm was used for training
• **sample**: \( z = (x, y) = (\text{data, label}); \)
• **training sample/example**: samples used in training
• **training set** → obtain candidate \( h(x; w) \)s
• **validation set** → select \( h(x; w) \) with best generalization
• **test set** → evaluate best \( h(x; w) \)'s accuracy
• **how to train?** → need an objective function
Expected Risk

- expected risk of prediction function $h(x; w)$

$$R(h(x; w)) = R(w) = \Pr\{y \neq \hat{y}\}$$

- evaluated over $P(x, y)$
- same as the true misclassification error rate
- ultimate metric for classification
Example

\[ \hat{y} = \text{sgn}(x - 1.5) \]

- **misclassification error rate:**
  \[ p_e = \text{Pr}\{\hat{y} = C_1|C_0\} \pi_0 + \text{Pr}\{\hat{y} = C_0|C_1\} \pi_1 \]

- \( \text{Pr}\{\hat{y} = C_1|C_0\} = 0.3 \rightarrow \text{sum non-underlined entries in } C_0 \text{ row} \)
- \( \text{Pr}\{\hat{y} = C_0|C_1\} = 0.1 \rightarrow \text{sum non-underlined entries in } C_1 \text{ row} \)
- As \( \pi_0 = 0.8 \), \( p_e = 0.26 \) or 26% of the cases will be misdiagnosed

<table>
<thead>
<tr>
<th>( x \rightarrow )</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>( C_1 )</td>
<td>0.0</td>
<td>0.1</td>
<td>0.3</td>
<td>0.6</td>
</tr>
<tr>
<td>( C_0 )</td>
<td>0.4</td>
<td>0.3</td>
<td>0.2</td>
<td>0.1</td>
</tr>
</tbody>
</table>

- \( R(h(x; w)) = p_e ; y \in \{0,1\} \) (class index)
- \( h(x; w) = h(x; \theta) = \text{sign}(x - 1.5) \)
- requires the knowledge of joint distribution \( P(x, y) \)

→ the data generation model
Empirical Risk

- empirical risk of prediction function $h(x; w)$

$$R_n(h(x; w)) = R_n(w) = \frac{1}{n} \sum_{i=1}^{n} \mathbb{I}(h(x_i; w) \neq y_i)$$

where the $\mathbb{I}(A)$ is the indicator function

$$\mathbb{I}(A) = \begin{cases} 
1 & \text{if } A \text{ is true} \\
0 & \text{otherwise}
\end{cases}$$

- misclassification error rate obtained by error counting → no need to know the joint distribution $P(x, y)$
Loss Function

• expected risk $R(h(x; w))$ is a discontinuous function of $w$, e.g., $\text{sign}()$
• hard to optimize $\rightarrow$ cannot take derivatives

• need a surrogate function that is a continuous approximation of risk $\rightarrow$

  loss function $l(h(x; w), y)$

• evaluated per example $(x_i, y_i)$
• equalizer example:

\[ l(h(x; w), c[k]) = (c[k] - w^T x)^2 = (c[k] - y_R[k])^2 \]

→ squared error across the slicer
• redefine expected risk via the loss function

\[ R(h(x; w)) = R(w) = E[l(h(x; w), y)] \]

where expectation is over \( P(x, y) \)

• equalizer example \( \rightarrow R(w) = E[(c[k] - y_R[k])^2] = J(w) \rightarrow \)

mean squared error (MSE)

• similarly redefine empirical risk

\[ R_n(h(x; w)) = R_h(w) = \frac{1}{n} \sum_{1}^{n} l(h(x_i; w), y_i) \]

• instantaneous risk \( (n = 1) \rightarrow \) loss function
• can take gradient of $R(\mathbf{w}) = E[l(h(\mathbf{x}; \mathbf{w}), y)]$ wrt $\mathbf{w}$ and set it to zero to obtain $\mathbf{w}_{opt}$

• example: in LMS, $l(h(\mathbf{x}_n; \mathbf{w}), d_n) = (d_n - \mathbf{w}^T \mathbf{x}_n)^2$

$$R(\mathbf{w}) = E[l(h(\mathbf{x}_n; \mathbf{w}), d_n)] = \sigma_y^2 - 2 \mathbf{p}^T \mathbf{w} + \mathbf{w}^T \mathbf{R} \mathbf{w}$$

leads to Wiener-Hopf solution $\mathbf{w}_{opt} = \mathbf{R}^{-1} \mathbf{p}$

• alternatively, use iterative methods such as
  – gradient descent
  – stochastic gradient descent (SGD)
Gradient Descent

Gradient

\[ \nabla_n = \frac{\partial R(w)}{\partial w} \bigg|_{w=w_n} \]

\[ w_{n+1} = w_n + \mu(-\nabla_n) \]

- \( \nabla_n \) \( \Rightarrow \) is the gradient of \( R(w) \) wrt \( w \) evaluated at \( w = w_n \)
- **LMS example**: \( \nabla_n = -2p + 2Rw_n \) \( \rightarrow \) needs data statistics (\( R \) and \( p \)) but does not need to compute matrix inverse
- can get stuck in a local minima
- large \( \mu \) may help jump over local minima but cause oscillations around the optimum (misadjustment)
Secthastic Gradient Descent

$\hat{v}_n = \frac{\partial l(w)}{\partial w} \bigg|_{w=w_n}$

$w_{n+1} = w_n + \mu(-\hat{v}_n)$

- $\hat{v}_n$ is the gradient of the loss function wrt $w$ evaluated at $w = w_n$
- replaces true gradient $\nabla_n$ with instantaneous gradient $\hat{v}_n$
- **example**: LMS update

\[ e_n = d_n - w_n^T x_n \]

\[ w_{n+1} = w_n + \mu e_n x_n \]

- falls out when SGD is applied with

\[ l(h(x_n; w), d_n) = (d_n - w^T x_n)^2 = e_n^2 \]
\[ w_{n+1} = w_n + \mu(-\hat{\nabla}_n) \]  

(SGD equation)

\[ \hat{\nabla}_n = \frac{\partial e_n^2}{\partial w} \bigg|_{w=w_n} = 2e_n \frac{\partial e_n}{\partial w} \bigg|_{w=w_n} \]

\[ = 2e_n \frac{\partial (d_n - w^T x_n)}{\partial w} \bigg|_{w=w_n} \]

\[ = -2e_n x_n \]

- substitute in SGD equation to obtain LMS update:

\[ w_{n+1} = w_n + 2\mu e_n x_n \]
Example – Sign-LMS

\[ w_{n+1} = w_n + \mu(-\hat{\nabla}_n) \]

(SGD equation)

- \( l(h(x_n; w), d_n) = |e_n| = |d_n - w^T x_n| \)

\[ \hat{\nabla}_n = \frac{\partial |e_n|}{\partial w} \bigg|_{w=w_n} = \text{sign}(e_n) \frac{\partial e_n}{\partial w} \bigg|_{w=w_n} = -\text{sign}(e_n)x_n \]

- derivative of \(|x|\) is \(\text{sign}(x)\)
- substitute in SGD equation to obtain sign-LMS update:

\[ w_{n+1} = w_n + \mu \times \text{sign}(e_n)x_n \]
SVM-SGD

\[ w_{n+1} = w_n - \gamma \nabla \]

- SVM-SGD minimizes instantaneous loss function:
  \[ l(w_n) = \frac{1}{2} \lambda \|w_n\|^2 + \max(0, 1 - y_n w_n^T x_n) \]

- instantaneous gradient:
  \[ \nabla_n = \frac{\partial l(w_n)}{\partial w_n} = \lambda w_n + \begin{cases} 0 & \text{if } y_n w_n^T x_n > 1 \\ -y_n x_n & \text{otherwise} \end{cases} \]

- to give the SVM
  \[ w_{n+1} = w_n - \gamma \begin{cases} \lambda w_n & \text{if } y_n w_n^T x_n > 1 \\ \lambda w_n - y_n x_n & \text{otherwise} \end{cases} \]
Weight Update Block

• the SVM-SGD update equation:

\[
\mathbf{w}_{n+1} = \mathbf{w}_n - \gamma \begin{cases} 
\lambda \mathbf{w}_n & \text{if } y_n \mathbf{w}_n^T \mathbf{x}_n > 1 \\
\lambda \mathbf{w}_n - y_n \mathbf{x}_n & \text{otherwise}
\end{cases}
\]

• can be re-written as:

\[
\mathbf{w}_{n+1} = (1 - \gamma \lambda) \mathbf{w}_n + \gamma \begin{cases} 
0 & \text{if } y_n \mathbf{w}_n^T \mathbf{x}_n > 1 \\
y_n \mathbf{x}_n & \text{otherwise}
\end{cases}
\]
SVM-SGD Data-flow Graph & Architecture

• inner loop has a multiplier → can choose \((1 - \gamma \lambda)\) to be power of 2
Synthetic Dataset
Linear SVM

Linear SVM: Separating Hyperplane after 10 streamed samples

\[ p_{det} = 85.4\% \text{ (SVM-SGD)} \]
\[ p_{det} = 93.4\% \text{ (LIBSVM)} \]
Linear SVM: Separating Hyperplane after 20 streamed samples

\[ p_{det} = 88.7\% \text{ (SVM-SGD)} \]
\[ p_{det} = 93.4\% \text{ (LIBSVM)} \]
Linear SVM: Separating Hyperplane after 30 streamed samples

\[ p_{det} = 87.6\% \text{ (SVM-SGD)} \]
\[ p_{det} = 93.4\% \text{ (LIBSVM)} \]
Linear SVM: Separating Hyperplane after 40 streamed samples

\[ p_{det} = 90.7\% \text{ (SVM-SGD)} \]
\[ p_{det} = 93.4\% \text{ (LIBSVM)} \]
Linear SVM: Separating Hyperplane after 50 streamed samples

\[ p_{det} = 91.1\% \text{ (SVM-SGD)} \]
\[ p_{det} = 93.4\% \text{ (LIBSVM)} \]
Fixed-point SVM-SGD

Understanding the Energy and Precision Requirements for Online Learning

Charbel Sakr  Ameya Patil  Sai Zhang  Yongjune Kim  Naresh Shanbhag
Department of Electrical and Computer Engineering
University of Illinois at Urbana-Champaign
Urbana, IL 61801


• previously, bounds on $B_X$ and $B_F$ were obtained (geometric bound)

$$B_X > \log_2 \left( \frac{\sqrt{N}||w||}{1 - 2^{-B_F} - \sqrt{N}2^{-B_F}||x||} \right)$$

• next \(\rightarrow\) bounds on $B_W$
• based on the stopping criterion used in LMS
Bounds on WUD Precision

\[ w_{n+1} = (1 - \gamma \lambda)w_n + \gamma \begin{cases} 
0 & \text{if } y_n w_n^T x_n > 1 \\
y_n x_n & \text{otherwise}
\end{cases} \]

- update term is non-zero if:

\[ B_w \geq B_x - \log_2(\gamma) \]
Example

- **dataset**: Breast Cancer UCI Machine Learning repository; \( N = 10 \) (dimension) \( \rightarrow \|x\| \)
- **run floating-point simulations** \( \rightarrow \|w\|, \gamma = 2^{-5} \)
- **substitute into the geometric bound**

\[
B_X > \log_2 \left( \frac{\sqrt{N} \|w\|}{1-2^{-B_F} - \sqrt{N}2^{-B_F} \|x\|} \right)
\]

\( \rightarrow B_F = 6; B_X = 6 \)

- **obtain weight update precision**

\[
B_w \geq B_X - \log_2(\gamma) = 6 + 5 = 11
\]
• calculate $R_h(\mathbf{w}_n)$ for each $n$ (over the data set)
• average $R_h(\mathbf{w}_n)$ over $M$ independent runs
• fixed-point convergence curve approaches floating-point curve as $B_W \rightarrow 11$ bits
Non-Linear SVM-SGD
• second order polynomial
• SVM-SGD update equation via NLIM

\[ w_{n+1} = (1 - \gamma \lambda) w_n + \gamma \begin{cases} 0 & \text{if } y_n w_n^T \phi(x_n) > 1 \\ y_n \phi(x_n) & \text{otherwise} \end{cases} \]

\[ \Phi(x) = \begin{bmatrix} x_1 \\ x_2 \\ x_1 x_2 \\ x_1^2 \\ x_2^2 \end{bmatrix} \]
Poly SVM

Polynomial SVM: Separating Hyperplane after 10 streamed samples

\[ p_{det} = 89.0\% \text{ (SVM-SGD)} \]
\[ p_{det} = 97.6\% \text{ (LIBSVM)} \]
Polynomial SVM: Separating Hyperplane after 20 streamed samples

\[ p_{det} = 84.8\% \text{ (SVM-SGD)} \]
\[ p_{det} = 97.6\% \text{ (LIBSVM)} \]
Polynomial SVM: Separating Hyperplane after 30 streamed samples

\[ p_{det} = 87.5\% \text{ (SVM-SGD)} \]
\[ p_{det} = 97.6\% \text{ (LIBSVM)} \]
Polynomial SVM: Separating Hyperplane after 40 streamed samples

\[ p_{det} = 91.4\% \text{ (SVM-SGD)} \]
\[ p_{det} = 97.6\% \text{ (LIBSVM)} \]
 Polynomial SVM: Separating Hyperplane after 50 streamed samples

\[ p_{det} = 92.5\% \text{ (SVM-SGD)} \]
\[ p_{det} = 97.6\% \text{ (LIBSVM)} \]
https://courses.engr.illinois.edu/ece598ns/fa2017

http://shanbhag.ece.uiuc.edu
clear all; close all; clc;
load('synth_data.mat');

run_size=2001;
run_numbers = 100;
batch_size = 50;

gamma=2^-10;
length_x = floor(run_size/batch_size)+1;

fig1 = figure;
hold on;
set(findall(fig1,'-property','FontSize'),'FontSize',20);

ylabel('Loss Function');
xlabel('Streamed Samples');

one_minus_gamma_lambda = 1-gamma;

for run=1:run_numbers
    so_far=1;
    W_current = rand(1,2);
    b_current = rand;
    for streamed=1:run_size
        % test so far
        if((rem(streamed,batch_size)==1) || (streamed==1))
            for i=1:length(testing_labels)
                x = testing_instances(i,:);'
                y=testing_labels(i);
                inside_term = dot(x,W_current) + b_current;
                loss_function_online(run,so_far) = loss_function_online(run,so_far) + max(0,1-y*inside_term);
            end
            loss_function_online(run,so_far)=loss_function_online(run,so_far)/(size(testing_labels,1));
            loss_function_online(run,so_far)= loss_function_online(run,so_far) + dot(W_current,W_current);
            so_far=so_far+1;
        end
        % train one
        rand_index = randi(length(training_labels));
        x_streamed = training_instances(rand_index,:);'
        y_true = training_labels(rand_index);
        [W_current,b_current] = online_update(W_current,b_current,...
        x_streamed,y_true,gamma,one_minus_gamma_lambda);
    end
    end
    tracking_loss_function = sum(loss_function_online,1)/run_numbers;
    plot(1:batch_size:batch_size*length(tracking_loss_function),tracking_loss_function,'k-x','Linewidth',2,'...
    MarkerSize',12,'DisplayName','Floating Point: SGD');