Machine Learning (CSE 446):
Variations on the Theme of Gradient Descent

Sham M Kakade
© 2018

University of Washington
cse446-staff@cs.washington.edu
Learning as Loss Minimization

\[ z^* = \arg\min_z \frac{1}{N} \sum_{n=1}^{N} L(x_n, y_n, z) + R(z) \]

For our hyperplane/neuron-inspired classifier, \( z = (w, b) \).
Subderivatives and Subgradients

A **subderivative** of $F$ at $x_0$ is any $c$ such that, for all $x$:

$$F(x) - F(x_0) \geq c(x - x_0)$$

This is a generalization of derivatives (for differentiable functions, there is only one subderivative at $x_0$, and it’s the derivative).

Vector of subderivatives in all dimensions: **subgradient**.
The set of subderivatives for the function at a point $x_0$ consists of the slopes of all tangent lines fully below the function.
The set of subderivatives for the function at a point $x_0$ consists of the slopes of all tangent lines fully below the function.
The set of subderivatives for the function at a point $x_0$ consists of the slopes of all tangent lines fully below the function.
The set of **subderivatives** for the function at a point $x_0$ consists of the slopes of all tangent lines fully below the function.
Variation 1

Data: function $F : \mathbb{R}^d \to \mathbb{R}$, number of iterations $K$, step sizes $\langle \eta^{(1)}, \ldots, \eta^{(K)} \rangle$

Result: $z \in \mathbb{R}^d$

initialize: $z^{(0)} = 0$;

for $k \in \{1, \ldots, K\}$ do

# choose a subgradient; doesn't matter which one;
$g^{(k)} = \nabla_z F(z^{(k-1)})$;
$z^{(k)} = z^{(k-1)} - \eta^{(k)} \cdot g^{(k)}$;

end

return $z^{(K)}$;

Algorithm 1: SUBGRADIENT DESCENT
Variation 2

**Data**: loss functions $L_1, \ldots, L_N$, regularization function $R$, number of iterations $K$, step sizes $\langle \eta^{(1)}, \ldots, \eta^{(K)} \rangle$

**Result**: parameters $z \in \mathbb{R}^d$

initialize: $z^{(0)} = 0$

for $k \in \{1, \ldots, K\}$ do

\[ i \sim \text{Uniform}(\{1, \ldots, N\}); \]

\[ g^{(k)} = \nabla_z L_i(z^{(k-1)}) + \nabla_z R(z^{(k-1)}); \]

\[ z^{(k)} = z^{(k-1)} - \eta^{(k)} \cdot g^{(k)}; \]

end

return $z^{(K)}$;

**Algorithm 2**: Stochastic(Sub)GradientDescent for minimizing $\frac{1}{N} \sum_{n=1}^{N} L_n(z) + R(z)$. 
Observation

If you let $L$ be the perceptron loss and don’t regularize, and run stochastic subgradient descent with all $\eta = 1$, you have recovered the perceptron algorithm.
Variation 3

**Data:** loss functions $L_1, \ldots, L_N$, regularization function $R$, number of iterations $K$, step sizes $\langle \eta^{(1)}, \ldots, \eta^{(K)} \rangle$, minibatch size $B$

**Result:** parameters $z \in \mathbb{R}^d$

initialize: $z^{(0)} = 0;$

for $k \in \{1, \ldots, K\}$ do

$I \sim \text{Uniform}(\{1, \ldots, N\}^B);$

$g^{(k)} = \frac{1}{B} \sum_{i \in I} \nabla z L_i(z^{(k-1)}) + \nabla z R(z^{(k-1)});$  

$z^{(k)} = z^{(k-1)} - \eta^{(k)} \cdot g^{(k)};$

end

return $z^{(K)}$;

**Algorithm 3:** MinibatchStochastic(Sub)GradientDescent for minimizing

$$\frac{1}{N} \sum_{n=1}^{N} L_n(z) + R(z).$$
General-Purpose Optimization Algorithms

\{\text{batch, minibatch, stochastic}\} \times (\text{sub})\text{gradient descent}
General-Purpose Optimization Algorithms

\{batch, minibatch, stochastic\} × (sub)gradient descent

Ninja: treat minibatch size \( B \in \{1, \ldots, N\} \) as a hyperparameter!
Choose your loss function \( L \). To fit the training data:

\[
\min_{w,b} \frac{1}{N} \sum_{n=1}^{N} L \left( y_n \cdot (w \cdot x_n + b) \right) + R(w, b)
\]

**Regularization**: add a penalty to the objective function to encourage generalization.

Most common: \( R(w, b) = \lambda \|w\|^2 \).

- Note that this term is convex and differentiable.

This is called (**squared**) \( L_2 \) **regularization** or **ridge regularization**.
Some Regularization Functions

ridge or (squared) $L_2$ \[ \lambda \|w\|_2^2 = \lambda \sum_d w[d]^2 \]

"$L_0$" \[ \lambda \|w\|_0 = \lambda \sum_d [w[d] \neq 0] \]

lasso or $L_1$ \[ \lambda \|w\|_1 = \lambda \sum_d |w[d]| \]

Inductive bias for ridge: small change in $x[d]$ should have a small effect on prediction. Penalizing $\|w\|_2$ is the same as penalizing $\|w\|_2^2$, but to get the same effect you’ll need a different $\lambda$. 
Some Regularization Functions

ridge or (squared) $L_2$ \[ \lambda \| \mathbf{w} \|_2^2 = \lambda \sum_d \mathbf{w}[d]^2 \]

“$L_0$” \[ \lambda \| \mathbf{w} \|_0 = \lambda \sum_d [\mathbf{w}[d] \neq 0] \]

lasso or $L_1$ \[ \lambda \| \mathbf{w} \|_1 = \lambda \sum_d |\mathbf{w}[d]| \]

Inductive bias for $L_0$: use fewer features.
Some Regularization Functions

ridge or (squared) $L_2$ \[ \lambda \| \mathbf{w} \|_2^2 = \lambda \sum_d \mathbf{w}[d]^2 \]

"$L_0$" \[ \lambda \| \mathbf{w} \|_0 = \lambda \sum_d \left[ \mathbf{w}[d] \neq 0 \right] \]

lasso or $L_1$ \[ \lambda \| \mathbf{w} \|_1 = \lambda \sum_d |\mathbf{w}[d]| \]

Inductive bias for $L_0$ and lasso: use fewer features.
A Constrained View of the Regularized Loss Minimization Problem

Tikhonov regularization:

\[ z^* = \arg\min_z \frac{1}{N} \sum_{n=1}^{N} L_n(z) + \lambda \|z\|_p \]

Ivanov regularization:

\[ z^* = \arg\min_z \frac{1}{N} \sum_{n=1}^{N} L_n(z) \]

s.t. \( \|z\|_p \leq \tau \)