Optimization for Machine Learning

Elman Mansimov[1]

September 24, 2015

[1] Modified based on Shenlong Wang's and Jake Snell's tutorials, with additional contents borrowed from Kevin Swersky and Jasper Snoek

Contents

An informal definition of optimization

Minimize (or maximize) some quantity.

Applications

- Engineering: Minimize fuel consumption of an automobile
- Economics: Maximize returns on an investment
- Supply Chain Logistics: Minimize time taken to fulfill an order
- Life: Maximize happiness

Recap: Linear Regression

Want to predict house price based on some information about the house (location, number of rooms, etc.) Assume that you have a data for n houses.

- x_i is a d dimensional vector of observations for house i $x_i = (x_i^1, x_i^2, ..., x_i^d)$; X is a $d \times n$ matrix of all houses.
- y is a vector of the price of each house $y = (y_1, y_2, ..., y_n)$
- $y = \mathbf{WX}$, W is a $1 \times d$ dimensional matrix
- W that would result in a lowest error, i.e. smallest difference between predicted price and real price.

More formally

Goal: find $\theta^* = \text{argmin}_\theta f(\theta)$, (possibly subject to constraints on θ).

- $\theta \in \mathbb{R}^n$: *optimization variable*
- $f : \mathbb{R}^n \to \mathbb{R}$: *objective function*

Maximizing $f(\theta)$ is equivalent to minimizing $-f(\theta)$, so we can treat everything as a minimization problem.

Optimization is a large area of research

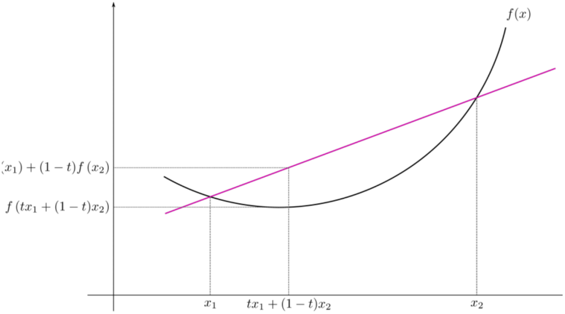
The best method for solving the optimization problem depends on which assumptions we want to make:

- Is θ discrete or continuous?
- What form do constraints on θ take? (if any)
- Is f "well-behaved"? (linear, differentiable, convex, submodular, etc.)

Convex Functions

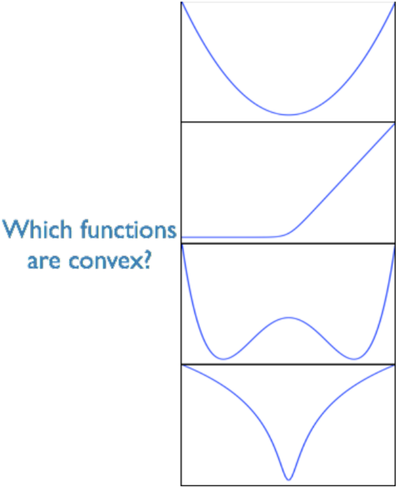
A function f is **convex** if for any two points θ_1 and θ_2 and any $t \in [0, 1]$,

$$f(t\theta_1 + (1-t)\theta_2) \leq tf(\theta_1) + (1-t)f(\theta_2)$$



Use the line test.

Convex Functions



Why do we care about convexity?

- Any local minimum is a global minimum.
- Which means that whatever solution we find would be the best solution.
- This makes optimization a lot easier because we don't have to worry about getting stuck in a local minimum.

Overview of Optimization for Machine Learning

Often in machine learning we are interested in learning the parameters θ of a model.
Goal: minimize some loss function

- For example, if we have some data (x, y), we may want to maximize $P(y|x,\theta)$.
- Equivalently, we can minimize $-\log P(y|x,\theta)$.
- We can also minimize other sorts of loss functions

log can help for numerical reasons

Naive Optimization Algorithm

- Try all posssible combinations of $\mathbf{W}$ until you find one that has the lowest error (brute force).
- Doesn't scale as you grow number of parameters and dimensions.
- Need help from calculus.

Gradient Descent: Motivation

From calculus, we know that the minimum of f must lie at a point where $\frac{\partial f(\theta^*)}{\partial \theta} = 0$.

- Sometimes, we can solve this equation analytically for θ.
- Most of the time, we are not so lucky and must resort to iterative methods.

Review

- Gradient: $\nabla_\theta f = (\frac{\partial f}{\partial \theta_1}, \frac{\partial f}{\partial \theta_2}, ..., \frac{\partial f}{\partial \theta_k})$

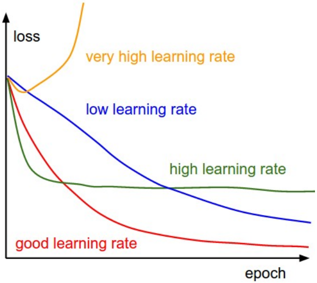
Outline of Gradient Descent Algorithm

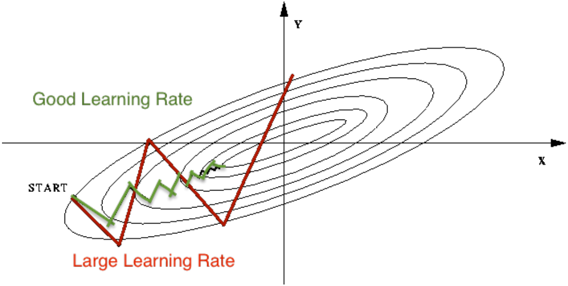
Where η is the learning rate and T is the number of iterations:

- Initialize θ_0 randomly
- for $t = 1 : T$:
    - $\delta_t \leftarrow -\eta \nabla_{\theta_{t-1}} f$ (i.e. calculate the change for the θ_t)
    - $\theta_t \leftarrow \theta_{t-1} + \delta_t$

The learning rate shouldn't be too big (objective function will blow up) or too small (will take a long time to converge)

Illustration of Learning Rates[2]





[2] First image taken from Andrej Karpathy's Stanford Lectures, second image taken from Wikipedia

Gradient Descent with Line-Search

Where η is the learning rate and T is the number of iterations:

- Initialize θ_0 randomly
- for $t = 1 : T$:
  - Finding a step size η_t such that $f(\theta_t - \eta_t \nabla_{\theta_{t-1}}) < f(\theta_t)$
  - $\delta_t \leftarrow -\eta_t \nabla_{\theta_{t-1}} f$
  - $\theta_t \leftarrow \theta_{t-1} + \delta_t$

Require a line-search step in each iteration.

Gradient Descent with Momentum

We can introduce a momentum coefficient $\alpha \in [0, 1)$ so that the updates have "memory":

- Initialize θ_0 randomly
- Initialize δ_0 to the zero vector
- for $t = 1 : T$:
  - $\delta_t \leftarrow -\eta \nabla_{\theta_{t-1}} f + \alpha \delta_{t-1}$
  - $\theta_t \leftarrow \theta_{t-1} + \delta_t$

Momentum is a nice trick that can help speed up convergence. Generally we choose α between 0.8 and 0.95, but this is problem dependent

Outline of Gradient Descent Algorithm

Where η is the learning rate and T is the number of iterations:

- Initialize θ_0 randomly
- Do:
  - $\delta_t \leftarrow -\eta \nabla_{\theta_{t-1}} f$
  - $\theta_t \leftarrow \theta_{t-1} + \delta_t$
- Until convergence

Setting a convergence criteria.

Some convergence criteria

- Change in objective function value is close to zero: $|f(\theta_{t+1}) - f(\theta_t)| < \epsilon$
- Gradient norm is close to zero: $\|\nabla_\theta f\| < \epsilon$
- Validation error starts to increase (this is called *early stopping*)

Checkgrad

- When implementing the gradient computation for machine learning models, it's often difficult to know if our implementation of f and ∇f is correct.
- We can use finite-differences approximation to the gradient to help:

$$\frac{\partial f}{\partial \theta_i} \approx \frac{f((\theta_1, \ldots, \theta_i + \epsilon, \ldots, \theta_n)) - f((\theta_1, \ldots, \theta_i - \epsilon, \ldots, \theta_n))}{2\epsilon}$$

Why don't we always just use the finite differences approximation?

- slow: we need to recompute f twice for each parameter in our model.
- numerical issues

Stochastic Gradient Descent

- Any iteration of gradient descent method requires that we sum over the entire dataset to compute gradient.
- SGD idea: at each iteraton, sub-sample a small amount of data (even just 1 point can work) and use that to estimate the gradient. (typically use around 100 samples)
- Each update is noisy, but very fast!
- This is the basis of optimizing ML algorithms with huge datasets (e.g., deep learning).

More on optimization

Convex Optimization by Boyd & Vandenberghe
Book available for free online at
http://www.stanford.edu/~boyd/cvxbook/
Numerical Optimization by Nocedal & Wright
Electronic version available from UofT Library

Resources for MATLAB

- Tutorials are available on the course website at http://www.cs.toronto.edu/~zemel/inquiry/matlab.php

Resources for Python

- Official tutorial: http://docs.python.org/2/tutorial/
- Google's Python class: https://developers.google.com/edu/python/
- Zed Shaw's *Learn Python the Hard Way*: http://learnpythonthehardway.org/book/

NumPy/SciPy/Matplotlib

- Scientific Python bootcamp (with video!): http://register.pythonbootcamp.info/agenda
- SciPy lectures: http://scipy-lectures.github.io/index.html

Questions?