First-Order Optimization Algorithms for Machine Learning

Gradient Descent Progress Bound

Mark Schmidt

University of British Columbia

Summer 2020

Current Hot Topics in Machine Learning

- Graph of most common keywords among ICML papers in 2015:

- Why is there so much focus on deep learning and optimization?

Why Study Optimization for Machine Learning?

- In machine learning, training is typically written as an optimization problem:
  - We optimize parameters w of model, given data.

- There are some exceptions:
  1. Methods based on counting and distances (KNN, random forests).
     - See CPSC 340.
  2. Methods based on averaging and integration (Bayesian learning).
     - See CPSC 540.

  But even these models have parameters to optimize.

- But why study optimization? Can't I just use optimization libraries?
  - "\", linprog, quadprog, CVX, MOSEK, NumPy, and so.

The Effect of Big Data and Big Models

- Datasets are getting huge, we might want to train on:
  - Entire medical image databases.
  - Every webpage on the internet.
  - Every product on Amazon.
  - Every rating on Netflix.
  - All flight data in history.

- With bigger datasets, we can build bigger models:
  - Complicated models can address complicated problems.
  - Regularized linear models on huge datasets are standard industry tool.
  - Deep learning allows us to learn features from huge datasets.

The Effect of Big Data and Big Models

- But optimization becomes a bottleneck because of time/memory.
  - We huge number of variables d, we can't afford $O(d^2)$ memory or $O(d^2)$ operations.
  - Going through huge datasets hundreds of times is too slow.
  - Evaluating huge models many times may be too slow.

- We are going to focus on first-order methods (that only use gradients).
  - Subgradient method, proximal-gradient, stochastic gradient, variance reduction.
    - Biased by own research interests/viewpoints towards what I think is important.
  - Key advantage: $O(d)$ iteration cost and memory (usually).

- Next time: how many iterations of gradient descent do we need?

Assumed Background Knowledge

- I am assuming that you already know the material from CPSC 340.
  - UBC's introductory machine learning class.
  - Least squares, logistic regression, and regularization.
  - Machine learning notation (X, y, w, λ, and so on).
  - Multivariate calculus in matrix notation, including gradients and Hessians.
  - Convex sets and functions, and how to show something is convex.
  - Gradient descent and stochastic gradient descent.
  - Topics from 340's prereqs (Taylor's theorem, eigenvalues, expectation function).
- The material from the last time I taught 340 is here:
  `https://www.cs.ubc.ca/~schmidtm/Courses/340-F19`

- You may get lost quickly if you aren't familiar with the above.
  - It might be better to go through the above first, and come back to this stuff later.
- I am also assuming you are familiar with proofs involving inequalities.
  - We will formally show many things, although this is not a "proofs" course.

Lectures

- Webpage: `www.cs.ubc.ca/~schmidtm/Courses/5XX-S20`
  - All slides will be posted online (before lecture, possibly with modifications after).

- Please ask questions: you probably have similar questions to others.
  - I may deflect to the next lecture or Piazza for certain questions.
  - You can also ask questions on Piazza.

- Be warned that the course will move fast and cover a lot of topics:
  - I recommend reviewing slides from previous lecture before each lecture.
    - Increases chances of you remembering things long-term.
  - Big ideas will be covered slowly and carefully.
  - But a bunch of other topics won't be covered in a lot of detail.
- Isn't it wrong to have only have shallow knowledge?
  - I think that each person should know some things really well ("be an expert").
  - But you should have some knowledge of many topics outside your expertise.
    - Know why many things are important, with the ability to fill details later.

Bonus Slides

- I will include a lot of "bonus slides".
  - May mention advanced variations of methods from lecture.
  - May overview big topics that we don't have time for.
  - May go over technical details that would derail class.
- You can safely "skip" this material and still follow the main "story".
  - But you may find them interesting or useful in the future.
- I'll use this colour of background on bonus slides.

Textbooks

- Some classic textbooks containing related material:
  - Boyd & Vandenberghe's "Convex Optimization".
  - Nesterov's "Introductory Lectures on Convex Optimization".
  - Any of Bertesekas' optimization-related textbooks.
- Some more-recent textbooks with an ML focus:
  - Bubeck's "Convex Optimization: Algorithms and Complexity".
  - Hazan's "Lecture notes: Optimization for Machine Learning".

Outline

Motivating Problem: Depth Estimation from Images

- We want to build system that predicts "distance to car" for each pixel in an image:

https://www.gadzooki.com/gadgets/5-ways-technology-is-going-to-make-driving-safer

- For example, pixel $(59, 108)$ has distance 30.4 meters.
- One way to build such a system:
  1. Collect a large number of images and label their pixels with the true depth.
  2. Use supervised learning to build a model that can predict depth of any pixel.

Supervised Learning Notation

- Supervised learning input is a set of n training examples.
- Each training example i consists of:
  - A set of features $x^i \in \mathbb{R}^d$
  - A label y^i
- For depth estimation:
  - Features could be a bunch of convolutions centered around the pixel.
  - Label would be the actual distance to the object in the pixel.
  - Supervised learning is a crucial tool used in self-driving cars.
- Supervised learning output is a model:
  - With linear models, summarized by a d-dimensioanl parameter vector w.
  - Given a new input $\tilde{x}^i$, model makes a prediction $\hat{y}^i$.
  - Goal is to maximize accuracy on new examples (test error).

Supervised Learning Notation (MEMORIZE)

- We'll assume that all vectors are column-vectors,

$$w = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_d \end{bmatrix}, \quad y = \begin{bmatrix} y^1 \\ y^2 \\ \vdots \\ y^n \end{bmatrix}, \quad x^i = \begin{bmatrix} x_1^i \\ x_2^i \\ \vdots \\ x_d^i \end{bmatrix}.$$

- I'm using w_j as the scalar parameter j.
- I'm using y^i as the label of example i (currently a scalar).
- I'm using x^i as the list of features for example i (column-vector of lenght d).
- I'm using x_j^i to denote feature j in training example i.
- I'll use x_j to denote feature j in a generic training example.

Supervised Learning Notation (MEMORIZE)

- We'll use X to denote the data matrix containing the x^i in the rows:

$$X = \begin{bmatrix} \text{—} & (x^1)^\top & \text{—} \\ \text{—} & (x^2)^\top & \text{—} \\ & \vdots & \\ \text{—} & (x^n)^\top & \text{—} \end{bmatrix}, \quad y = \begin{bmatrix} y^1 \\ y^2 \\ \vdots \\ y^t \end{bmatrix},$$

- We'll use $\tilde{X}$ and $\tilde{y}$ to denote test data:

$$\tilde{X} = \begin{bmatrix} \text{—} & (\tilde{x}^1)^\top & \text{—} \\ \text{—} & (\tilde{x}^2)^\top & \text{—} \\ & \vdots & \\ \text{—} & (\tilde{x}^n)^\top & \text{—} \end{bmatrix}, \quad \tilde{y} = \begin{bmatrix} \tilde{y}^1 \\ \tilde{y}^2 \\ \vdots \\ \tilde{y}^t \end{bmatrix},$$

  and $\hat{y}$ to denote a vector of predictions.
- Our prediction in linear models is $\hat{y}^i = w^\top x^i$ (train) or $\hat{y}^i = w^\top \tilde{x}^i$ (test).
  - Notation alert: I use $\hat{y}^i$ whether it's a prediction on training or test data.

Loss Plus Regularizer Framework

- A classic model for supervised learning is L2-regularized least squares,

$$f(w) = \underbrace{\frac{1}{2}\sum_{i=1}^{n}(w^\top x^i - y^i)^2}_{\text{squared error}} + \underbrace{\frac{\lambda}{2}\sum_{j=1}^{d} w_j^2}_{\text{regularizer}}.$$

- Squred error measures how well we fit example i with parameters w.
- Regularizer measures how complicated the model is with parameters w.
- Regularization parameter $\lambda > 0$ controls strength of regularization:
  - Controls complexity of model, with large λ leading to less overfitting.
  - Usually set by optimizing error on a validation set or with cross-validation.

Other Loss Functions and Regularizers

- Many machine learning models be viewed in a "loss plus regularizer" framework:

$$f(w) = \underbrace{\sum_{i=1}^{n} f_i(w)}_{\text{data-fitting term}} + \underbrace{\lambda g(w)}_{\text{regularizer}}.$$

- Alternative loss functions to squared error:
  - Absolute error $|w^\top x^i - y^i|$ is more robust to outliers.
  - Hinge loss $\max\{0, 1 - y^i w^\top x^i\}$ is better for binary y^i.
  - Logistic loss $\log(1 + \exp(-y^i w^\top x^i))$ is better for binary y^i and is smooth.
  - Softmax loss $-w_{y^i}^\top x^i + \log(\sum_{c=1}^{k} \exp(w_c^\top x^i))$ for discrete y^i ("cross entropy").
- Another common regularizer is L1-regularizer,

$$g(w) = \sum_{j=1}^{d} |w_j|,$$

which encourages sparsity in w (many w_j are set to zero for large λ).

Solution of L2-Regularized Least Squares

- Our L2-regularized least squares objective function was

$$f(w) = \frac{1}{2}\sum_{i=1}^{n}(w^\top x^i - y^i)^2 + \frac{\lambda}{2}\sum_{j=1}^{d} w_j^2,$$

which is convex and can be writen in matrix and norm notation as

$$f(w) = \frac{1}{2}\|Xw - y\|^2 + \frac{\lambda}{2}\|w\|^2.$$

- After some work, the gradient of this quadratic objective can be written

$$\nabla f(w) = X^\top(Xw - y) + \lambda w,$$

and setting the gradient to zero and solving for w gives

$$w = (X^\top X + \lambda I)^{-1}(X^\top y),$$

where we've used that $(X^\top X + \lambda I)$ is invertible (because it is positive-definite).

- We could alternately minimize f by using gradient descent ("first-order method").

Outline

Gradient Descent for Finding a Local Minimum

- A typical gradient descent algorithm (first proposed by Cauchy in 1847):
  - Start with some initial guess, w^0.
  - Generate new guess w^1 by moving in the negative gradient direction:

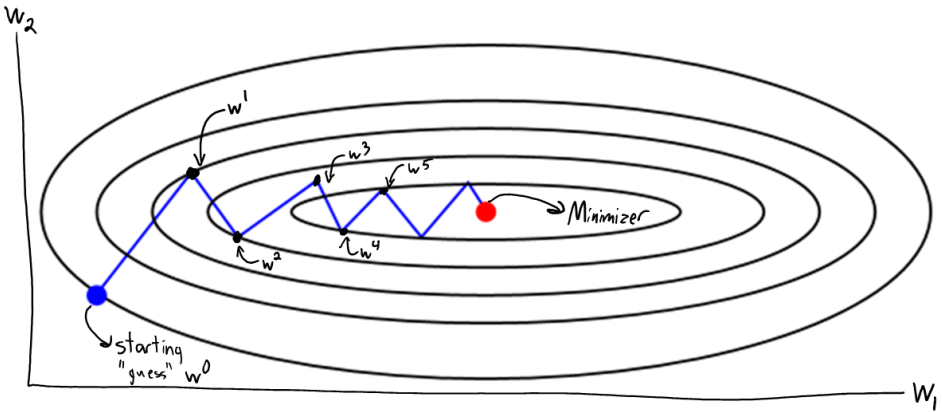
$$w^1 = w^0 - \alpha_0 \nabla f(w^0),$$

    where α_0 is the step size.
  - Repeat to successively refine the guess:

$$w^{k+1} = w^k - \alpha_k \nabla f(w^k), \quad \text{for } k = 1, 2, 3, \ldots$$

    where we might use a different step-size α_k on each iteration.
- Stop if $\|\nabla f(w^k)\| \leq \epsilon$.
  - In practice, you also stop if you detect that you aren't making progress.

Gradient Descent in 2D



Lipschitz Contuity of the Gradient

- Let's first show a basic property:
  - If the step-size α_t is small enough, then gradient descent decreases f.

- We'll analyze gradient descent assuming gradient of f is Lipschitz continuous.
  - There exists an L such that for *all* w and v we have

$$\|\nabla f(w) - \nabla f(v)\| \leq L\|w - v\|.$$

  - "Gradient can't change arbitrarily fast".

- This is a fairly weak assumption: it's true in most ML models.
  - Least squares, logistic regression, neural networks with sigmoid activations, etc.

- Assumption is not necessary to show gradient descent decrease f for small α_t.
  - You only need continuity of the gradient, but that isn't enough to prove rates.

Lipschitz Contuity of the Gradient

- For C^2 functions, Lipschitz continuity of the gradient is equivalent to

$$\nabla^2 f(w) \preceq LI,$$

  for all w.
- Equivalently: "singular values of the Hessian are bounded above by L".
  - For least squares, minimum L is the maximum eigenvalue of $X^T X$.

- This means we can bound quadratic forms involving the Hessian using

$$\begin{aligned} d^T \nabla^2 f(u) d &\leq d^T (LI) d \\ &= L d^T d \\ &= L \|d\|^2. \end{aligned}$$

Descent Lemma

- For a C^2 function, a variation on the multivariate Taylor expansion is that

$$f(v) = \underbrace{f(w) + \nabla f(w)^T(v-w)}_{\text{tangent hyper-plane}} + \underbrace{\frac{1}{2}(v-w)^T\nabla^2 f(u)(v-w)}_{\text{quadratic form}},$$

  for any w and v (with u being some point between w and v).

- Lipschitz continuity implies the green term is at most $L\|v-w\|^2$,

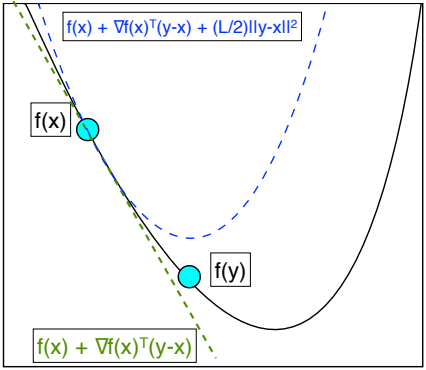
$$f(v) \leq f(w) + \nabla f(w)^T(v-w) + \frac{L}{2}\|v-w\|^2,$$

  which is called the descent lemma.

- The descent lemma also holds for C^1 functions (bonus slide).

Descent Lemma

- The descent lemma gives us a convex quadratic upper bound on f:



- This bound is minimized by a gradient descent step from w with $\alpha_k = 1/L$.

Gradient Descent decreases f for $\alpha_k = 1/L$

- So let's consider doing gradient descent with a step-size of $\alpha_k = 1/L$,

$$w^{k+1} = w^k - \frac{1}{L}\nabla f(w^k).$$

- If we substitle w^{k+1} and w^k into the descent lemma we get

$$f(w^{k+1}) \leq f(w^k) + \nabla f(w^k)^T(w^{k+1} - w^k) + \frac{L}{2}\|w^{k+1} - w^k\|^2.$$

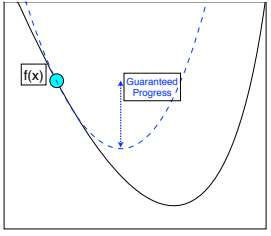
- Now if we use that $(w^{k+1} - w^k) = -\frac{1}{L}\nabla f(w^k)$ in gradient descent,

$$\begin{aligned} f(w^{k+1}) &\leq f(w^k) - \frac{1}{L}\nabla f(w^k)^T\nabla f(w^k) + \frac{L}{2}\|\frac{1}{L}\nabla f(w^k)\|^2 \\ &= f(w^k) - \frac{1}{L}\|\nabla f(w^k)\|^2 + \frac{1}{2L}\|\nabla f(w^k)\|^2 \\ &= f(w^k) - \frac{1}{2L}\|\nabla f(w^k)\|^2. \end{aligned}$$

Implication of Lipschitz Continuity

- We've derived a bound on guaranteed progress when using $\alpha_k = 1/L$.

$$f(w^{k+1}) \leq f(w^k) - \frac{1}{2L}\|\nabla f(w^k)\|^2.$$



- If gradient is non-zero, $\alpha_k = 1/L$ is guaranteed to decrease objective.
- Amount we decrease grows with the size of the gradient.

General Step-Size

- Let's consider doing using generic step-size of α_k,

$$w^{k+1} = w^k - \alpha_k \nabla f(w^k).$$

- We can derive this step as minimizing a quadratic approximation of f,

$$w^{k+1} \in \underset{w}{\text{argmin}} \left\{ f(w^k) + \nabla f(w^k)^T (w - w^k) + \frac{1}{2\alpha_k} \|w - w^k\|^2 \right\},$$

  which is not necessarily an upper bound (since we could have $1/\alpha_k << L$).
- Plugging in a generic step-size into the descent lemma and simplifying gives

$$f(w^{k+1}) \leq f(w^k) - \alpha_k \underbrace{\left(1 - \frac{\alpha_k L}{2}\right)}_{> 0 \text{ if } \alpha_k < 2/L} \|\nabla f(w^k)\|^2,$$

  which shows that any $\alpha_k < 2/L$ is guaranteed to decrease f.

Choosing the Step-Size in Practice

- In practice, you should never use $\alpha_k = 1/L$.
  - L is usually expensive to compute, and this step-size can be really small.
    - You only need a step-size this small in the worst case.

- One practical option is to approximate L:
  - Start with a small guess for $\hat{L}$ (like $\hat{L} = 1$).
  - Before you take your step, check if the progress bound is satisfied:

$$f(\underbrace{w^k - (1/\hat{L})\nabla f(w^k)}_{\text{potential } w^{k+1}}) \leq f(w^k) - \frac{1}{2\hat{L}}\|\nabla f(w^k)\|^2.$$

  - Double $\hat{L}$ if it's not satisfied, and test the inequality again (don't ever decrease $\hat{L}$).
- Worst case: eventually have $L \leq \hat{L} < 2L$ and you decrease f at every iteration.
- Good case: $\hat{L} << L$ and you are making way more progress than using $1/L$.

Choosing the Step-Size in Practice

- An approach that usually works better is a backtracking line-search:
  - Start each iteration with a large step-size α.
    - So even if we took small steps in the past, be optimistic that we're not in worst case.
  - Decrease α until Armijo condition is satisfied (this is what *findMin* does),

$$f(\underbrace{w^k - \alpha\nabla f(w^k)}_{\text{potential } w^{k+1}}) \leq f(w^k) - \alpha\gamma\|\nabla f(w^k)\|^2 \quad \text{for} \quad \gamma \in (0, 1/2],$$

    often we choose γ to be very small like $\gamma = 10^{-4}$.
    - We would rather take a small decrease instead of trying many α values.

- Good codes use clever tricks to initialize and decrease the α values.
  - Usually only try 1 value per iteration.
- Even more fancy line-search: Wolfe conditions (makes sure α is not too small).
  - Good reference on these tricks: Nocedal and Wright's Numerical Optimization book.

Summary

- First-order methods: gradient-based iterative continous optimization algorithms.
  - Key property is low iteration cost.
- Supervised learning notation: X, y, w, $\hat{y}$, $\tilde{y}$, λ, and so on.
- "Loss plus regularizer" framework can be used to describe most ML models.
- Gradient descent: "repeatedly follow the negative gradient direction".
- Guaranteed progress bound if gradient is Lipschitz, based on norm of gradient.
- Practical step size strategies based on the progress bound.

- Post-lecture slides: Cover various related issues.
  - Checking derivative code, why use gradient direction, descent lemma for C^1.

- Next time: how many iterations of gradient descent do we need?

Checking Derivative Code

- Gradient descent codes require you to write objective/gradient code.
  - This tends to be error-prone, although automatic differentiation codes are helping.

- Make sure to check your derivative code:
  - Numerical approximation to partial derivative i:

$$\nabla_i f(x) \approx \frac{f(x + \delta e_i) - f(x)}{\delta},$$

    where δ is small e_i has a 1 at position i and zero elsewhere.
  - For large-scale problems you can check a random direction d:

$$\nabla f(x)^T d \approx \frac{f(x + \delta d) - f(x)}{\delta}$$

  - If the left side coming from your code is very different from the right side, there is likely a bug.

Why the gradient descent iteration?

- For a C^2 function, a variation on the multivariate Taylor expansion is that

$$f(v) = f(w) + \nabla f(w)^T(v - w) + \frac{1}{2}(v - w)^T \nabla^2 f(u)(v - w),$$

for any w and v (with u being some convex combination of w and v).

- If w and v are very close to each other, then we have

$$f(v) = f(w) + \nabla f(w)^T(v - w) + O(\|v - w\|^2),$$

and the last term becomes negligible.

- Ignoring the last term, for a fixed $\|v - w\|$ I can minimize $f(v)$ by choosing $(v - w) \propto -\nabla f(w)$.
  - So if we're moving a small amount the optimal choice is gradient descent.

Descent Lemma for C^1 Functions

- Let ∇f be L-Lipschitz continuous, and define $g(\alpha) = f(x + \alpha z)$ for a scalar α.

$$f(y) = f(x) + \int_0^1 \nabla f(x + \alpha(y - x))^T (y - x) d\alpha \quad \text{(fund. thm. calc.)}$$

$$(\pm \text{ const.}) = f(x) + \nabla f(x)^T (y - x) + \int_0^1 \left(\nabla f(x + \alpha(y - x)) - \nabla f(x)\right)^T (y - x) d\alpha$$

$$(\text{CS ineq.}) \leq f(x) + \nabla f(x)^T (y - x) + \int_0^1 \|\nabla f(x + \alpha(y - x)) - \nabla f(x)\| \|y - x\| d\alpha$$

$$(\text{Lipschitz}) \leq f(x) + \nabla f(x)^T (y - x) + \int_0^1 L\|x + \alpha(y - x) - x\| \|y - x\| d\alpha$$

$$(\text{homog.}) = f(x) + \nabla f(x)^T (y - x) + \int_0^1 L\alpha \|y - x\|^2 d\alpha$$

$$(\int_0^1 \alpha = \frac{1}{2}) = f(x) + \nabla f(x)^T (y - x) + \frac{L}{2} \|y - x\|^2.$$

Equivalent Conditions to Lipschitz Continuity of Gradient

- We said that Lipschitz continuity of the gradient

$$\|\nabla f(w) - \nabla f(v)\| \leq L\|w - v\|,$$

  is equivalent for C^2 functions to having

$$\nabla^2 f(w) \preceq LI.$$

- There are a lot of other equivalent definitions, see here:
  - `http://xingyuzhou.org/blog/notes/Lipschitz-gradient`.

Functions that don't have Lipschitz-continuous Gradient

- A simple example of a function which does not have a Lipschitz-continuous gradient is $f(x) = x^3$: $f''(x) = 6x$ which is not bounded as we vary x.
- Regarding ML applications: any non-smooth function would not have a Lipschitz-continuous gradient, such as the L1-regularizer $g(x) = \lambda\|x\|_1$ or neural networks with ReLU activations. We will analyze non-smooth functions later. In this case you usually assume that the function is Lipschitz-continuous.
- The other common type of functions arising in ML that are not Lipschitz-continuous are entropy-like functions. For example, $f(x) = x\log x$ for $x > 0$ is smooth (differentiable) on its domain, but it has $f''(x) = 1/x$ which is not bounded as x approaches 0.
- However, note that $f(x) = x^3$ and $f(x) = x\log x$ (for $x > 0$) have a Lipschitz-continuous gradient over any compact set. So if your iterations stay in a closed and bounded set, then they effectively have a Lipschitz-continuous gradient.

Lipschitz-Continuos Function vs. Lipschitz-Continuous Gradient

- Function f is Lipschitz-cont. if $|f(w) - f(v)| \leq L\|w - v\|$ for some L.
- Gradient ∇f is Lipschitz-cont. if $\|\nabla f(w) - \nabla f(v)\| \leq L\|w - v\|$ for some L.
  - Some people say that f is "Lipschitz-smooth" here. I avoid to prevent confusing.

- Don't get these confused, and neither implies the other:
  - $f(w) = \|w\|_1$ is Lipschitz-cont. but does not have a Lipschitz-cont. gradient.
  - $f(w) = \frac{1}{2}\|Xw - y\|^2$ is not Lipshitz-cont. but does have a Lipschitz-cont. gradient.
  - $f(w) = \sum_{i=1}^{n} \log(1 + \exp(-y^i w^\top x^i))$ is Lipschitz-cont. with ∇f Lipschitz-cont.