CPSC 340: Machine Learning and Data Mining

Gradient Descent

Fall 2017
• **We will have tutorials** on non-holiday days this week.

• **Assignment 2** is due Friday.
  – 1 late day to hand it in on Monday, 2 for Wednesday.
  – The “imread” function is in PyPlot (not Images.jl), weird error in findMin.jl (fixed in a2.zip).

• **Assignment 1** marks are up.
  – If you have questions, see “Assignment 1 Marking Thread” on Piazza.

• **Extra office hours:**
  – 2 TAs on the Thursday 2-3pm office hours when assignments are due.
  – Extra office hours this Friday at 1-2 (Siyuan at Table 2).
  – Extra instructor office hours on October 19th 4pm (ICICS 246).

• **Midterm details:**
  – In class October 20th (55 minutes).
  – 1 page double-sided cheat sheet.
  – Previous midterms posted on Piazza.
  – Short-answer questions on “non-bonus” (white) slides.
  – Calculation questions will focus on assignment topics.
  – Topics only appearing in L14 will treated as “bonus”.
Last Week: Linear Regression

- We discussed linear models:
  \[ \hat{y}_i = w_1 x_{i1} + w_2 x_{i2} + \cdots + w_d x_{id} \]
  \[ = \sum_{j=1}^{d} w_j x_{ij} = w^T x_i \]

- “Multiply feature \( x_{ij} \) by weight \( w_j \), add them to get \( \hat{y}_i \).”

- We discussed squared error function:
  \[ f(w) = \frac{1}{2} \sum_{i=1}^{n} (w^T x_i - y_i)^2 \]

- Interactive demo:

Motivation: Large-Scale Least Squares

• Normal equations find ‘w’ with \( \nabla f(w) = 0 \) in \( O(nd^2 + d^3) \) time.

\[
(\chi^\top \chi)w = \chi^\top y
\]

\( O(nd^2) \) \hspace{1cm} \( O(nd) \)
(matrix multiply) \hspace{1cm} (matrix-vector)

→ Solving a \( d \times d \) system is \( O(d^3) \)

– Very slow if ‘d’ is large.

• Alternative when ‘d’ is large is gradient descent methods.
  – Probably the most important class of algorithms in machine learning.
Gradient Descent for Finding a Local Minimum

• **Gradient descent** is an *iterative optimization* algorithm:
  – It starts with a “guess” $w^0$.
  – It uses the gradient $\nabla f(w^0)$ to generate a better guess $w^1$.
  – It uses the gradient $\nabla f(w^1)$ to generate a better guess $w^2$.
  – It uses the gradient $\nabla f(w^2)$ to generate a better guess $w^3$.
  …
  – The limit of $w^t$ as ‘$t$’ goes to $\infty$ has $\nabla f(w^t) = 0$.

• It converges to the global optimum if ‘$f$’ is convex.
Gradient Descent for Finding a Local Minimum

• Gradient descent is based on a simple observation:
  – Give parameters ‘w’, the direction of largest decrease is $-\nabla f(w)$.
• Gradient descent is based on a simple observation:
  – Give parameters ‘w’, the direction of largest decrease is $-\nabla f(w)$.
Gradient Descent for Finding a Local Minimum

- Gradient descent is based on a simple observation:
  - Give parameters ‘w’, the direction of largest decrease is $-\nabla f(w)$. 

![Diagram showing gradient descent](image)
Gradient Descent for Finding a Local Minimum

- **Gradient descent** is based on a simple observation:
  - Give parameters ‘w’, the direction of largest decrease is $-\nabla f(w)$.
Gradient Descent for Finding a Local Minimum

- **Gradient descent** is based on a simple observation:
  - Give parameters ‘w’, the direction of largest decrease is $-\nabla f(w)$.

  ![Graph showing gradient descent](image)

  Now the slope $\nabla f(w^4)$ is positive, so we move in the negative direction.
Gradient Descent for Finding a Local Minimum

– We start with some initial guess, $w^0$.
– Generate new guess by moving in the negative gradient direction:

$$w^{l+1} = w^l - \alpha^l \nabla f(w^l)$$

  • This decreases ‘$f$’ if the “step size” $\alpha^0$ is small enough.
  • Usually, we decrease $\alpha^0$ if it increases ‘$f$’ (see “findMin.jl”).

– Repeat to successively refine the guess:

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

– Stop if not making progress or

$$\|\nabla f(w^t)\| \leq \varepsilon$$

  $\varepsilon$ Some small scalar.

  Approximate local minimum
• Under weak conditions, algorithm converges to a ‘w’ with $\nabla f(w) = 0$.
  – ‘f’ is bounded below, $\nabla f$ doesn’t change arbitrarily fast, small and constant $\alpha^t$. 
Gradient Descent

• Least squares via normal equations vs. gradient descent:
  – Normal equations cost $O(nd^2 + d^3)$.
  – Gradient descent costs $O(ndt)$ to run for ‘t’ iterations.
    \[
    \text{Computing } \nabla f(w) = X^T X w - X^T y \text{ only costs } O(nd).
    \]
    – Gradient descent can be faster when ‘d’ is very large:
      • If solution is “good enough” for a ‘t’ less than minimum(d, d^2/n).
      • CPSC 540: ‘t’ proportional to “condition number” of $X^TX$ (no direct ‘d’ dependence).
  – Normal equations only solve linear least squares problems.
    • Gradient descent solves many other problems.
Beyond Gradient Descent

• There are many variations on gradient descent.
  – Methods employing a “line search” to choose the step-size.
  – “Conjugate” gradient and “accelerated” gradient methods.
  – Newton’s method (which uses second derivatives).
  – Quasi-Newton and Hessian-free Newton methods.
  – Stochastic gradient (later in course).

• This course focuses on gradient descent and stochastic gradient:
  – They’re simple and give reasonable solutions to most ML problems.
  – But the above can be faster for some applications.
(pause)
Least Squares with Outliers

- Consider least squares problem with outliers:
  \[ x \leftarrow \text{"outlier" that doesn't follow trend} \]

http://setosa.io/ev/ordinary-least-squares-regression
Least Squares with Outliers

• Consider least squares problem with outliers:

This is what least squares will actually do.

• Least squares is very sensitive to outliers.
Least Squares with Outliers

• Squaring error shrinks small errors, and magnifies large errors:

• Outliers (large error) influence ‘w’ much more than other points.

http://students.brown.edu/seeing-theory/regression/index.html
Least Squares with Outliers

• Squaring error shrinks small errors, and magnifies large errors:

  ![Graph showing absolute and squared errors]

• Outliers (large error) influence ‘w’ much more than other points.
  – Good if outlier means ‘plane crashes’, bad if it means ‘data entry error’.
Robust Regression

- **Robust regression** objectives put less focus on large errors (outliers).
- For example, use **absolute error** instead of squared error:

$$f(w) = \sum_{i=1}^{n} |w^T x_i - y_i|$$

- Now decreasing ‘small’ and ‘large’ errors is equally important.
- Instead of minimizing L2-norm, minimizes **L1-norm** of residuals:

  Least squares:
  $$f(w) = \frac{1}{2} ||Xw - y||^2$$

  Least absolute error:
  $$f(w) = ||Xw - y||_1$$
Least Squares with Outliers

• Least squares is very sensitive to outliers.

Linear model \( \mathbf{w} \) minimizing

\[
J(w) = \frac{1}{2} \| \mathbf{Xw} - \mathbf{y} \|^2
\]
Least Squares with Outliers

- Absolute error is more robust to outliers:

Linear model \( w \) minimizing \( f(w) = \| Xw - y \|_1 = \sum_{i=1}^{n} |w^T x_i - y_i| \)
Regression with the L1-Norm

• Unfortunately, minimizing the absolute error is harder.
  – We don’t have “normal equations” for minimizing the L1-norm.
  – Absolute value is non-differentiable at 0.

  - Generally, harder to minimize non-smooth than smooth functions.
    • Unlike smooth functions, the gradient may not get smaller near a minimizer.
  – We’re going to use a smooth approximation, then apply gradient descent.
Smooth Approximations to the L1-Norm

• There are differentiable approximations to absolute value.
  – Common example is Huber loss:

\[
f(w) = \sum_{i=1}^{n} h(w^T x_i - y_i)
\]

\[
h(r_i) = \begin{cases} 
\frac{1}{2} r_i^2 & \text{for } |r_i| \leq \varepsilon \\
\varepsilon (|r_i| - \frac{1}{2} \varepsilon) & \text{otherwise}
\end{cases}
\]

  – Note that ‘h’ is differentiable: \( h'(\varepsilon) = \varepsilon \) and \( h'(-\varepsilon) = -\varepsilon \).
  – This ‘f’ is convex but setting \( \nabla f(x) = 0 \) does not give a linear system.
  – But we can minimize the Huber loss using gradient descent.
Motivation for Considering Worst Case

https://xkcd.com/937/
“Brittle” Regression

• What if you really care about getting the outliers right?
  – You want best performance on worst training example.
  – For example, if in worst case the plane can crash.

• In this case you could use something like the infinity-norm:

\[
\ell(w) = \| X_w - y \|_\infty
\]

where

\[
\| r \|_\infty = \max_i \sum |r_i|
\]

• Very sensitive to outliers (“brittle”), but worst case will be better.
Log-Sum-Exp Function

• As with the $L_1$-norm, the $L_\infty$-norm is convex but non-smooth:
  – We can again use a smooth approximation and fit it with gradient descent.

• Convex and smooth approximation to max function is log-sum-exp function:

\[
\max_i \left\{ z_i \right\} \approx \log \left( \sum_i \exp(z_i) \right)
\]

  – We’ll use this several times in the course.
  – Notation alert: when I write “log” I always mean “natural” logarithm: $\log(e) = 1$.

• Intuition behind log-sum-exp:
  – $\sum_i \exp(z_i) \approx \max_i \exp(z_i)$, as largest element is magnified exponentially (if no ties).
    • While $\log(\exp(z_i)) = z_i$. 

Summary

• **Gradient descent** finds stationary point of differentiable function.
  – Finds global optimum if function is convex.
• **Robust regression** using L1-norm is less sensitive to outliers.
• **Brittle regression** using Linf-norm is more sensitive to outliers.
• **Smooth approximations:**
  – Let us apply gradient descent to non-smooth functions.
  – **Huber loss** is a smooth approximation to absolute value.
  – **Log-Sum-Exp** is a smooth approximation to maximum.

• **Next time:**
  – We start our quest to automatically find the right features...
Why use the negative gradient direction?

• For a twice-differentiable ‘f’, multivariable Taylor expansion gives:
  \[ f(w^{t+1}) = f(w^t) + \nabla f(w^t)^T (w^{t+1} - w^t) + \frac{1}{2} (w^{t+1} - w^t)^T \nabla^2 f(v)(w^{t+1} - w^t) \]
  for some ’v’ between \( w^{t+1} \) and \( w^t \).

• If gradient can’t change arbitrarily quickly, Hessian is bounded and:
  \[ f(w^{t+1}) = f(w^t) + \nabla f(w^t)^T (w^{t+1} - w^t) + O(||w^{t+1} - w^t||^2) \]
  becomes negligible as \( w^{t+1} \) gets close to \( w^t \).

  – But which choice of \( w^{t+1} \) decreases ‘f’ the most?
    • As \( ||w^{t+1} - w^t|| \) gets close to zero, the value of \( w^{t+1} \) minimizing \( f(w^{t+1}) \) in this formula converges to \( (w^{t+1} - w^t) = -\alpha^t \nabla f(w^t) \) for some scalar \( \alpha^t \).
    • So if we’re moving a small amount, the optimal \( w^{t+1} \) is:
      \[ w^{t+1} = w^t - \alpha_t \nabla f(w^t) \] for some scalar \( \alpha_t \).
Question from class: "Can we use $w^{t+1} = w^t - \frac{1}{\|\nabla f(w^t)\|} \nabla f(w^t)$?"

This will work for a while, but notice that

$$\|w^{t+1} - w^t\| = \| \frac{1}{\|\nabla f(w^t)\|} \nabla f(w^t) \|$$

$$= \frac{1}{\|\nabla f(w^t)\|} \| \nabla f(w^t) \|$$

$$= 1$$

So the algorithm never converges.
Log-Sum-Exp for Brittle Regression

- To use log-sum-exp for brittle regression:

\[ \|Xw - y\|_\infty = \max_i \frac{1}{2} \max_j \{ w^T x_i - y_i, y_j - w^T x_i \} \]

\[ = \max_i \frac{1}{2} \max \{ w^T x_i - y_i, y_j - w^T x_i \} \] since \( |z| = \max \{ z, -z \} \)

\[ = \log \left( \sum_{i=1}^{n} \exp (w^T x_i - y_i) + \sum_{i=1}^{n} \exp (y_i - w^T x_i) \right) \] using log-sum-exp to approximate "max" over \( 2n \) terms.
Log-Sum-Exp Numerical Trick

- Numerical problem with log-sum-exp is that $\exp(z_i)$ might overflow.
  - For example, $\exp(100)$ has more than 40 digits.

- Implementation ‘trick’: Let $\beta = \max_i \sum z_i$,

\[
\log \left( \sum_i \exp(z_i) \right) = \log \left( \sum_i \exp(z_i - \beta + \beta) \right) \\
= \log \left( \sum_i \exp(z_i - \beta) \exp(\beta) \right) \\
= \log(\exp(\beta)) + \log \left( \sum_i \exp(z_i - \beta) \right) \\
= \beta + \log \left( \sum_i \exp(z_i - \beta) \right) \leq 1 \text{ so no overflow}
\]
Gradient Descent for Non-Smooth?

• “You are unlikely to land on a non-smooth point, so gradient descent should work for non-smooth problems?”
  – Consider just trying to minimize the absolute value function:

    ![Graph of absolute value function](image)

    – Norm(gradient) is constant when not at 0, so unless you are lucky enough to hit exactly 0, you will just bounce back and forth forever.
    – We didn’t have this problem for smooth functions, since the gradient gets smaller as you approach a minimizer.
    – You could fix this problem by making the step-size slowly go to zero, but you need to do this carefully to make it work, and the algorithm gets much slower.
Gradient Descent for Non-Smooth?

• Counter-example from Bertsekas’ “Nonlinear Programming” where gradient descent for a non-smooth convex problem does not converge to a minimum.
Random Sample Consensus (RANSAC)

- In computer vision, a widely-used generic framework for robust fitting is random sample consensus (RANSAC).
- This is designed for the scenario where:
  - You have a large number of outliers.
  - Majority of points are “inliers”: it’s really easy to get low error on them.
Random Sample Consensus (RANSAC)

- RANSAC:
  - Sample a small number of training examples.
    - Minimum number needed to fit the model.
    - For linear regression with 1 feature, just 2 examples.
  - Fit the model based on the samples.
    - Fit a line to these 2 points.
    - With ‘d’ features, you’ll need ‘d’ examples.
  - Test how many points are fit well based on the model.
  - Repeat until we find a model that fits at least the expected number of “inliers”.
- You might then re-fit based on the estimated “inliers”.