Gradient-based Learning

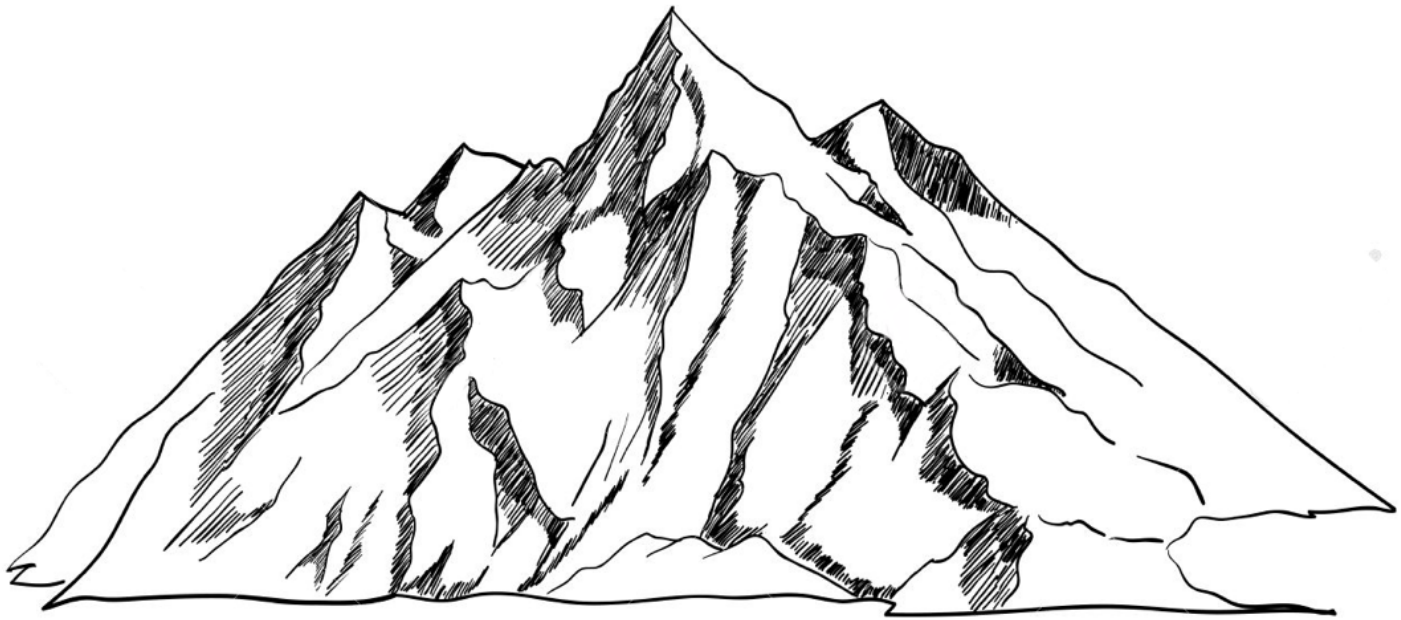
Nan Ye

School of Mathematics and Physics
The University of Queensland

Learning as Optimization

- Many learning problems are directly formulated as an optimization problem.
  - e.g. learning a Bernoulli/Gaussian distribution
  - e.g. OLS, ridge regression
  - e.g. naive Bayes classifier, logistic regression, SVM
- Some of the optimization problems have closed-form solutions, but many do not.
  - e.g. logistic regression and SVM do not have closed-form solutions
- We often need to use numerical methods to solve optimization problems in machine learning.

The Hill Climber Analogy



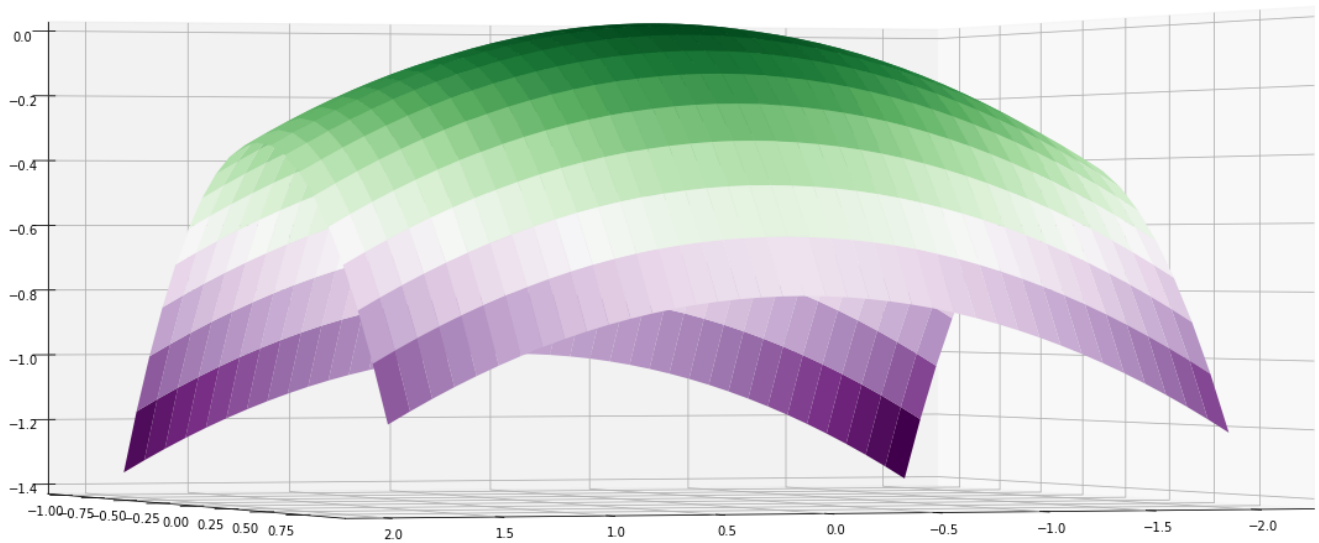
- You want to climb to the peak of a hill, but you have very limited visibility due to a heavy fog.
- You have a tool that allows you to measure the steepness of the hill at your location along any direction.
- Using the tool takes a lot of time, and you do not want to use it too frequently.

- We can choose to climb along the steepest uphill direction.
- However, we need to determine how much we climb along that direction each time after we measure steepness so that we do not go off the track and then possibly go downhill instead.
- Similarly, if we want to go downhill, we can choose the steepest downhill direction.

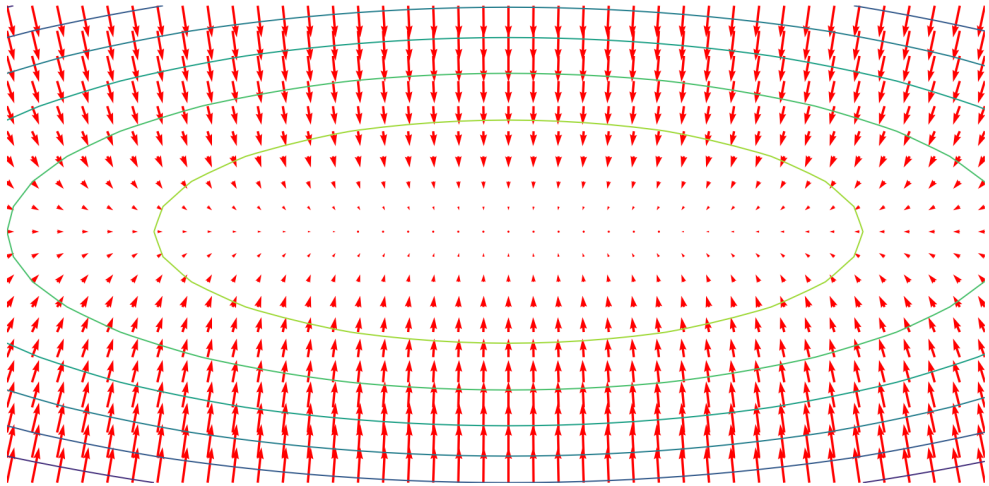
- It is illustrative to look at how this translates to maximizing the function

$$f(x, y) = -0.1x^2 - y^2$$

- The surface plot of the function is shown below



- The gradient field of the function is shown below.



- The red arrows indicate the directions and magnitudes of gradients.
- We can see that the gradients generally point towards the maximizer: in fact these are the steepest ascent directions, and the negative gradients are the steepest descent directions.

Gradient Descent

- Now we focus on the minimization problem, and consider minimizing a function $f(\mathbf{w})$.
- Gradient descent is the most basic gradient-based method for minimizing a function.
- Assume that we start from some $\mathbf{w}_0$, then at iteration $t \geq 0$, we compute the next iterate $\mathbf{w}_{t+1}$ using

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta_t \nabla f(\mathbf{w}_t),$$

  where $\eta_t \geq 0$ is a step size to be chosen.
- The step size is often called the learning rate when $f(\mathbf{w})$ is an objective function in machine learning.

Why does gradient descent work

- Using the first order Taylor series expansion, for any small vector d,

$$f(\mathbf{w} + d) \approx f(\mathbf{w}) + d^\top \nabla f(\mathbf{w}).$$

- Hence for small η, we have

$$\begin{aligned} f(\mathbf{w}_{t+1}) &= f(\mathbf{w}_t - \eta_t \nabla f(\mathbf{w}_t)) \\ &\approx f(\mathbf{w}_t) + (-\eta_t \nabla f(\mathbf{w}_t))^\top \nabla f(\mathbf{w}_t) \\ &= f(\mathbf{w}_t) - \eta_t \|\nabla f(\mathbf{w}_t)\|_2^2 \\ &< f(\mathbf{w}_t), \end{aligned}$$

if $\nabla f(\mathbf{w}) \neq 0$. That is, the function value decreases if we move along the negative gradient direction by a small step.

Examples

- Consider the following minimization problem

$$\min_{x \in \mathbf{R}} f(x, y) = x^2 + y^2$$

- Clearly, the minimizer is $(0, 0)$.
- We consider how gradient descent works starting from $(2, 3)$ using different step sizes.

- Constant step size $\eta_s = 0.25$ for all s.
  - $(x_1, y_1) = (2, 3)$.
  - $(x_2, y_2) = (x_1, y_1) - 0.25 \cdot 2(x_1, y_1) = 0.5(x_1, y_1)$.
  - $(x_3, y_3) = (x_2, y_2) - 0.25 \cdot 2(x_2, y_2) = 0.5(x_2, y_2) = 0.5^2(x_1, y_1)$.
  - $\ldots$
  - $(x_s, y_s) = 0.5^{s-1}(x_1, y_1) \to (0, 0)$ as $s \to \infty$.
- In this case, we never find the minimizer in finitely many iterations, but we can find a solution that is arbitrarily close to the minimizer after sufficiently many iterations.

- Constant step size $\eta_s = 1$ for all s.
  - $(x_1, y_1) = (2, 3)$.
  - $(x_2, y_2) = (x_1, y_1) - 1 \cdot 2(x_1, y_1) = -(x_1, y_1)$.
  - $(x_3, y_3) = (x_2, y_2) - 1 \cdot 2(x_2, y_2) = -(x_2, y_2) = (x_1, y_1)$.
  - $\ldots$
  - $(x_s, y_s) = (-1)^{s-1}(x_1, y_1)$ does not converge to the minimizer.
- We never encounter a solution that is close to the minimizer in this case!

- As another example, consider minimizing the MSE for OLS (see Lecture 2)

$$R_n(\beta) = \frac{1}{n}\sum_i (\mathbf{x}_i^\top \beta - y_i)^2 = \frac{1}{n}\|\mathbf{X}\beta - \mathbf{y}\|_2^2.$$

- Then we have $\nabla R_n(\beta) = \frac{2}{n}\mathbf{X}^\top(\mathbf{X}\beta - \mathbf{y})$.
- Gradient descent is given by

$$\beta_{t+1} = \beta_t - \frac{2\eta_t}{n}\mathbf{X}^\top(\mathbf{X}\beta_t - \mathbf{y}).$$

Step Size Rules

- As can be seen from previous slides, the step size is important.
- It determines whether gradient descent converges or not, and how fast it converges. To make gradient descent converge fast,
  - The step size need to be small enough to guarantee function value is decreasing.
  - The step size need to be large enough so that there is a sufficient decrease.

  It is often not easy to choose a good step size.
- There are some commonly used rules for choosing the step sizes.

Fixed step sizes

- We can choose the step sizes in advance, such as setting η_t to a constant or $\eta_t = \frac{1}{\sqrt{t+1}}$.
- For some problems, we know how to choose a constant step size so that gradient descent converges efficiently.
- In general, we need to try diminishing step sizes though.

Line search (optional)

- A natural idea is to choose η_t to minimize $f(\mathbf{w}_t - \eta_t \nabla f(\mathbf{w}_t))$.
- This is called line search, and is itself a minimization problem that is hard to solve exactly.
- A useful rule to approximately find η_t is the Goldstein-Armijo rule: Find $\mathbf{w}_{t+1} = \mathbf{w}_t - \eta_t \nabla f(\mathbf{w}_t)$ such that

$$\alpha \nabla f(\mathbf{w}_t)^\top (\mathbf{w}_t - \mathbf{w}_{t+1}) \leq f(\mathbf{w}_t) - f(\mathbf{w}_{t+1}),$$
$$\beta \nabla f(\mathbf{w}_t)^\top (\mathbf{w}_t - \mathbf{w}_{t+1}) \geq f(\mathbf{w}_t) - f(\mathbf{w}_{t+1}),$$

  where $0 < \alpha < \beta < 1$.
  That is, the decrease in function value is in $[c\alpha, c\beta]$, where $c = \eta_t \| \nabla f(x) \|^2)$.

Gradient Checking

- Implementating gradient descent can be tricky, because for machine learning problems, the gradient can be difficult to compute and implement.
- Assuming that we have implemented the objective function $f(\mathbf{w})$ and the exact gradient $\nabla f(\mathbf{w})$, then we can use the numerical gradient to check the gradient implementation.
- That is, we choose a small δ and check whether

$$\frac{\partial f(\mathbf{w})}{\partial w_i} \approx \frac{f(\mathbf{w} + \delta \mathbf{e}_i) - f(\mathbf{w})}{\delta},$$

  where $\mathbf{e}_i$ is the i-th standard unit vector.
- Numerical gradients are easy to implement but slow, while exact gradients are hard to implement but efficient.

Stochastic Gradient Descent (SGD)

- In machine learning, the objective function $f(\mathbf{w})$ often has the form

$$f(\mathbf{w}) = \frac{1}{n} \sum_i f_i(\mathbf{w}),$$

  where $f_i(\mathbf{w})$ measures how the model $\mathbf{w}$ fits example i.
  - e.g., in OLS, $f_i(\mathbf{w}) = (\mathbf{w}^\top \mathbf{x}_i - y_i)^2$.
- In SGD, instead of using full gradients computed on the whole dataset, we use stochastic gradients computed on random selected examples.

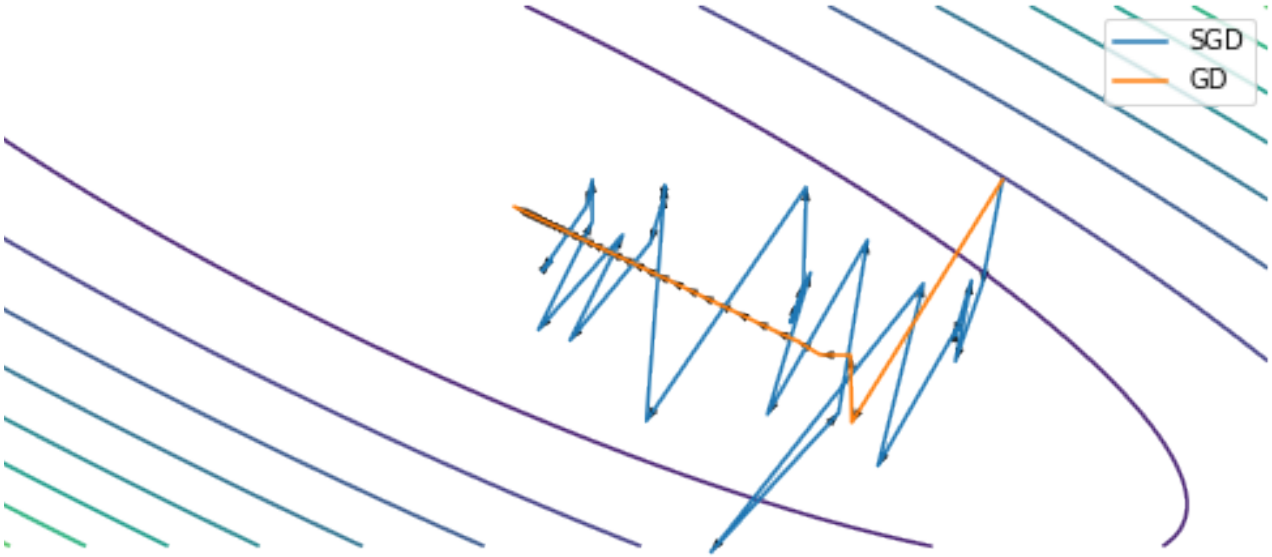
- That is, we update $\mathbf{w}_t$ to

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta_t \tilde{g}_t,$$

  where $\tilde{g}_t = \nabla f_i(\mathbf{w})$ with i randomly sampled from $1, \ldots, n$.
- We often use the mini-batch version of SGD, in which we set

$$\tilde{g}_t = \frac{1}{|S|} \sum_{i \in S} \nabla f_i(\mathbf{w}),$$

  where S is a random subset of $1, \ldots, n$.
- While SGD can jump around in the solution space, the mini-batch version is much more stable.



- Gradient descent has a smooth trajectory, while SGD has a zigzagging trajectory.

- SGD is often computationally attractive when the dataset is very large.
- In practice, if the dataset has a lot of redundancy, SGD is able to find a good solution quickly even though it uses only a subset of the examples (typically the case for large datasets).

Revisiting the Perceptron

- The Perceptron algorithm can be seen as a special case of SGD.
- Consider the loss function

$$L((\mathbf{x}, y), \mathbf{w}) = (-y\mathbf{w}^\top \mathbf{x})_+,$$

  for the classifier $h_{\mathbf{w}}(\mathbf{x}) = \text{sgn}(\mathbf{w}^\top \mathbf{x})$. Here $(x)_+ = \max(x, 0)$ is the positive part function.
- The loss is 0 for correct classification, and $-y\mathbf{w}^\top x$ for incorrect classification.
- The empirical risk is

$$R_n(\mathbf{w}) = \frac{1}{n} \sum_{i=1}^{n} (-y_i \mathbf{w}^\top \mathbf{x}_i)_+,$$

- The Perceptron algorithm is SGD applied to $R_n(\mathbf{w})$.

Revisiting Adaline

- The objective function is

$$R_n(\mathbf{w}) = \frac{1}{n} \sum_i (\mathbf{w}^\top \mathbf{x}_i - y_i)^2.$$

- Adaline is SGD applied to MSE above.

More on Gradient-based Learning

- Gradient descent is the simplest algorithm to use gradients for optimization.
- We will see many more sophisticated algorithms in later lectures.
  - momentum, AdaDelta, Adam...

Your turn

Which of the following statement is correct? (Multiple choice)

(a) Gradient descent minimizes a function by moving along the steepest descent direction.

(b) The best step size for gradient descent is 1.

(c) Several early neural net training algorithms are special cases of SGD.

What You Need to Know

- Gradient descent
  - Intuition, update formula, justification
  - Choice of step size
- Stochastic gradient descent
  - Capable of exploiting the redundancy in large datasets
  - Special cases: Perceptron, Adaline