11.3. Gradient Descent

Lecture based on "Dive into Deep Learning" **http://D2L.AI** (Zhang et al., 2020)

Prof. Dr. Christoph Lippert

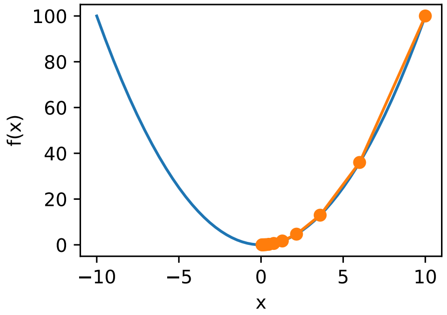
Digital Health & Machine Learning

Gradient descent in one dimension:

- For continuously differentiable real-valued function $f : \mathbb{R} \to \mathbb{R}$, we obtain the 1^{st} order Taylor expansion
$$f(x+\epsilon) = f(x) + \epsilon f'(x) + O(\epsilon^2).$$
- Moving a small ϵ in the direction of the **negative gradient** will decrease f.
- We pick a **fixed step size** $\eta > 0$ and choose $\epsilon = -\eta f'(x)$.
- Plugging this into the Taylor expansion above we get
$$f(x - \eta f'(x)) = f(x) - \eta f'^2(x) + O(\eta^2 f'^2(x)).$$
- If the derivative $f'(x) \neq 0$ does not vanish we make progress since $\eta f'^2(x) > 0$.
- For sufficiently small η the higher order terms to become irrelevant. Hence,
$$f(x - \eta f'(x)) \lessapprox f(x).$$
- Thus, if we use
$$x \leftarrow x - \eta f'(x)$$
to iterate x, the value of function $f(x)$ might decline.

gradient descent

1. choose an initial value x
2. choose constant $\eta > 0$
3. iterate x until the **stop condition** is reached
    - magnitude of the gradient $|f'(x)|$ is small enough
    - number of iterations has exceeded a certain value



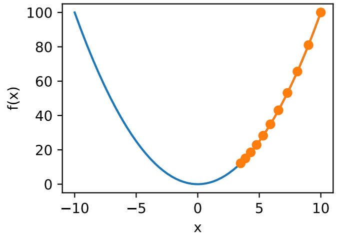
Example (objective function $f(x) = x^2$)

- initial value $x = 10$
- learning rate $\eta = 0.2$
- 10 iterations

A learning rate that is **too small**, will cause x to update very slowly

- requires more iterations to get a better solution.

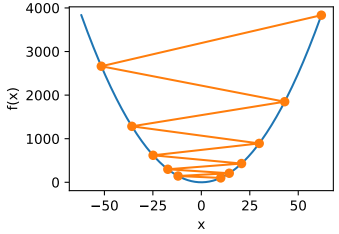
$\eta = 0.05:$



For an **excessively high** learning rate, the algorithm overshoots and diverges

- Large $|\eta f'(x)|$ might be too large for the first-order Taylor expansion.
- That is, the term $O(\eta^2 f'^2(x))$ might become significant.
- We cannot guarantee that the iteration of x will be able to lower the value of $f(x)$.

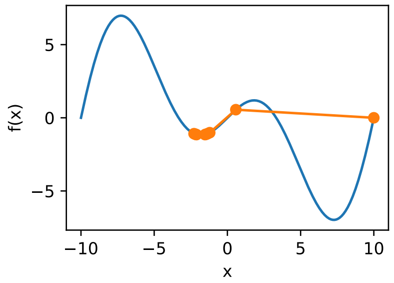
$\eta = 1.1:$



For **non-convex** functions we may end up with one of many solutions.

- depending on our choice of learning rate
- depending on how well conditioned the problem is

$$f(x) = x \cdot \cos cx$$



For **multivariate** functions $f : \mathbb{R}^d \to \mathbb{R}$, $\mathbf{x} \in \mathbb{R}^d$

- The gradient is a vector consisting of d partial derivatives:

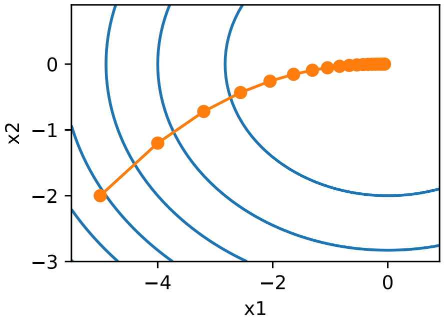
$$\nabla f(\mathbf{x}) = \left[\frac{\partial f(\mathbf{x})}{\partial x_1}, \frac{\partial f(\mathbf{x})}{\partial x_2}, \ldots, \frac{\partial f(\mathbf{x})}{\partial x_d}\right]^\top.$$

- Each $\partial f(\mathbf{x})/\partial x_i$ indicates the rate of change of f at $\mathbf{x}$ with respect to x_i.
- The 1^{st} order Taylor approximation is

$$f(\mathbf{x} + \epsilon) = f(\mathbf{x}) + \epsilon^\top \nabla f(\mathbf{x}) + O(\|\epsilon\|^2).$$

- The direction of steepest descent in terms of ϵ is given by $-\nabla f(x)$.
- Choosing learning rate $\eta > 0$ yields

$$\mathbf{x} \leftarrow \mathbf{x} - \eta \nabla f(\mathbf{x})$$



Example ($f(\mathbf{x}) = x_1^2 + 2x_2^2$)

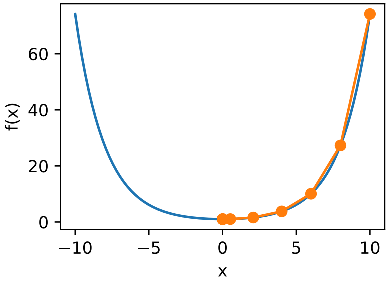
- $\nabla f(\mathbf{x}) = [2x_1, 4x_2]^\top$
- $\mathbf{x}^0 = [-5, -2]^\top$
- $\eta = 0.1$
- 20 iterations

We can also take the 2^{nd} order Taylor expansion of f

$$f(\mathbf{x}+\epsilon) = f(\mathbf{x}) + \epsilon^\top \nabla f(\mathbf{x}) + \frac{1}{2}\epsilon^\top \nabla^2 f(\mathbf{x})\epsilon + O(\|\epsilon\|^3)$$

- $\nabla^2 f(\mathbf{x}) =: H_f$ is a $d \times d$ matrix called the **Hessian** of f.
- Taking derivatives of the Taylor expansion with regard to ϵ and ignoring higher order terms we arrive at

$$\nabla f(\mathbf{x}) + H_f \epsilon = \vec{0} \quad \text{and hence} \quad \epsilon = -H_f^{-1} \nabla f(\mathbf{x}).$$



Example ($f(x) = \cosh(0.5 \cdot x)$)

- $\nabla f(x) = 0.5 \cdot \sinh(0.5 \cdot x)$
- $H_f = 0.5^2 \cdot \cosh(0.5 \cdot x$

Example ($f(x) = \frac{1}{2}x^2$)

- $\nabla f(x) = x$ and $H_f = 1$
- $x \leftarrow x - x$ (converges in 1 iteration)

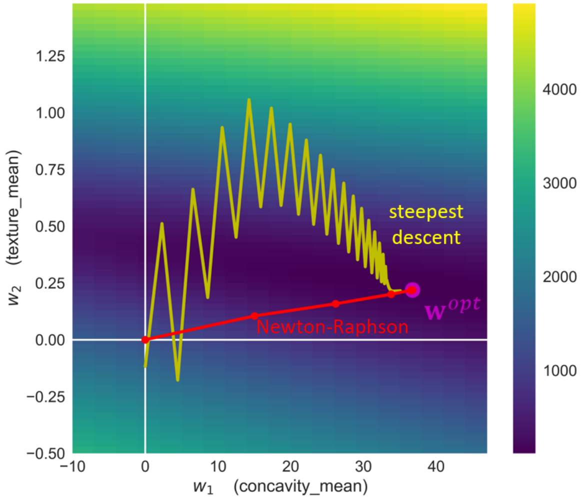
Logistic Regression

Logistic regression is a convex optimization problem.

$$L(\mathbf{w}) = \underbrace{-\sum_{n\in c_1} \ln(\pi(\mathbf{x}_n\mathbf{w})) - \sum_{n'\in c_2} \ln(1-\pi(\mathbf{x}_{n'}\mathbf{w}))}_{loss} + \underbrace{\lambda \cdot 0.5 \cdot \sum_{d=1}^{D} {w_d}^2}_{regularizer}$$

Gradient:

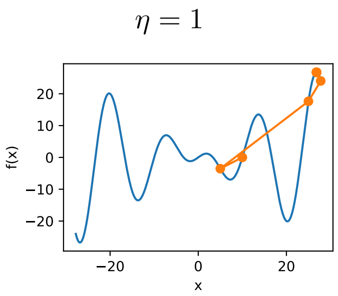
$$\nabla L\left(\mathbf{w}^t\right) = \begin{bmatrix} \frac{\partial L}{\partial w_1^t} \\ \vdots \\ \frac{\partial L}{\partial w_D^t} \end{bmatrix} = \underbrace{\mathbf{X}^T\left(\pi\left(\mathbf{X}\mathbf{w}^t\right) - I\left(\mathbf{y} == c_1\right)\right)}_{\nabla \text{loss}(\mathbf{w}^t)} + \underbrace{\lambda \cdot \mathbf{w}^t}_{\nabla \text{regularizer}(\mathbf{w}^t)}$$

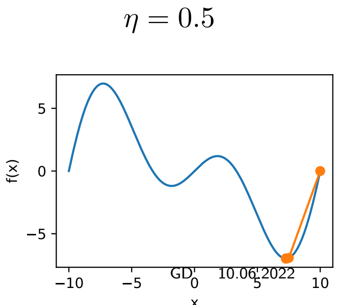


Now let's see what happens when we have a *non-convex* function.

- in Newton's method divide by the second derivative (multiply by Hessian^{-1}).
- This means that if the second derivative is *negative* we would walk into the direction of *increasing* f.

Example ($f(x) = x\cos(cx)$)





Preconditioning

- For small d and simple problems ∇_f^2 is easy to compute.
- For deep networks, ∇_f^2 may be prohibitively large, due to the cost of storing $O(d^2)$ entries.
- To compute ∇_f^2 via backprop we would need to apply backprop to the backpropagation call graph. (expensive)

An alternative way is by avoiding to compute the Hessian in its entirety but only compute the **diagonal** entries.

$$\mathbf{x} \leftarrow \mathbf{x} - \eta \mathrm{diag}(H_f)^{-1} \nabla \mathbf{x}.$$

- Not as good as the full Newton method, but much better than not using it.

  For example: 1 feature in meters, 1 feature in millimeters
- Estimates for the main diagonal elements are central to stochastic gradient descent optimization algorithms.
- Effectively **preconditioning** with gradient descent amounts to selecting a different learning rate for each coordinate.

line search in conjunction with gradient descent avoids overshooting and insufficient progress.

- In each iteration, we use the direction given by $\nabla f(\mathbf{x})$ and then perform (binary) search as to which steplength η minimizes $f(x - \eta \nabla f(\mathbf{x}))$.
- This algorithm converges rapidly.
- However, each step of the line search requires evaluation of the objective function on the entire dataset.
- Typically, this is too costly.

Summary

- Learning rates matter.
  - Too large and we diverge
  - Too small and we don't make progress.
- Gradient descent can get stuck in local minima.
- In high dimensions adjusting learning the learning rate is complicated.
- Preconditioning can help with scale adjustment.
- Newton's method is a lot faster *once* it has started working properly in convex problems.
- Beware of using Newton's method without any adjustments for non-convex problems.