EECS 189 Introduction to Machine Learning
Fall 2020

Note 12

1 Nonlinear Least Squares

Up to this point, we've restricted ourselves to *linear* regression models. That is, our prediction $\hat{y} = \mathbf{w}^\top \mathbf{x}$ is a linear function of the input $\mathbf{x}$. This holds even in the case of least-squares polynomial regression — while the predicted value is not a linear function of the raw input $\mathbf{x}$, it is still a linear function of the augmented polynomial feature input $\phi(\mathbf{x})$.

Effectively, we have been able to form nonlinear models by manually augmenting features to the input. Now what if instead of using a linear function of the augmented input, we could use an arbitrary nonlinear function $f(\mathbf{x}; \mathbf{w})$ *directly* of the raw input $\mathbf{x}$? This approach is often more expressive and robust, because it removes the burden of augmenting expressive features to the input. As a motivating example, consider the problem of estimating the 2D position $\mathbf{w} = (w_1, w_2)$ of a robot. We are given noisy distance estimates $Y_i \in \mathbb{R}$ from n sensors whose positions $\mathbf{x}_i \in \mathbb{R}^2$ are fixed and known. Since we are predicting *distance*, it is reasonable to use the model $f(\mathbf{x}; \mathbf{w}) = \|\mathbf{x} - \mathbf{w}\|_2$. This model is clearly more appropriate than restricting ourselves to a linear model with augmented features — in that case, what exactly would the augmented features be?

Note however that for most problems, we are not given the form or structure of the model. Consider the following example: we are trying to predict a user's income based on their occupation, age, education, etc... It is not exactly clear what model we should use. Rather than specifying a specific family of nonlinear functions, we are instead interested in a universal function appropriator $f(\mathbf{x}; \mathbf{w})$ which can approximate any function $f(\mathbf{x})$ with appropriate parameters $\mathbf{w}$. This will be the basis for **neural networks**, which we will study in detail later.

For the purposes of our discussion, let us assume that we are given a model f, an arbitrary (nonlinear) differentiable function parameterized by $\mathbf{w}$:

$$Y_i = f(\mathbf{x}_i; \mathbf{w}) + Z_i, \quad Z_i \overset{\text{iid}}{\sim} \mathcal{N}(0, \sigma^2), \quad i = 1, \ldots, n$$

which can equivalently be expressed as $Y_i \mid \mathbf{x}_i \sim \mathcal{N}(f(\mathbf{x}_i; \mathbf{w}), \sigma^2)$. We are interested in finding the parameters $\hat{\mathbf{w}}_{\text{MLE}}$ that maximize the likelihood of the data:

$$\begin{aligned}
\hat{\mathbf{w}}_{\text{MLE}} &= \arg\max_{\mathbf{w}} \ \ell(\mathbf{w}; \mathbf{X}, \mathbf{y}) \\
&= \arg\max_{\mathbf{w}} \ \sum_{i=1}^{n} \log p(y_i \mid \mathbf{x}_i, \mathbf{w}) \\
&= \arg\max_{\mathbf{w}} \ \sum_{i=1}^{n} \log \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y_i - f(\mathbf{x}_i; \mathbf{w}))^2}{2\sigma^2}\right) \\
&= \arg\max_{\mathbf{w}} \ \sum_{i=1}^{n} \left[-\frac{1}{2}\log\left(2\pi\sigma^2\right) - \frac{1}{2\sigma^2}(y_i - f(\mathbf{x}_i; \mathbf{w}))^2\right]
\end{aligned}$$

$$= \arg\min_{\mathbf{w}} \sum_{i=1}^{n} (y_i - f(\mathbf{x}_i; \mathbf{w}))^2$$

Observe that the objective function is a sum of squared residuals as we've seen before, but now the function f is nonlinear. For this reason this method is called **nonlinear least squares**.

Motivated by the MLE formulation above, our goal is to solve the following optimization problem:

$$\min_{\mathbf{w}} L(\mathbf{w}) = \min_{\mathbf{w}} \frac{1}{2} \sum_{i=1}^{n} (y_i - f(\mathbf{x}_i; \mathbf{w}))^2$$

One way to solve this optimization problem is to find all of its critical points and choose the point that minimizes the objective. From **first-order optimality conditions**, the gradient of the objective function at any minimum must be zero:

$$\nabla_{\mathbf{w}} L(\mathbf{w}) = \sum_{i=1}^{n} (y_i - f(\mathbf{x}_i; \mathbf{w})) \nabla_{\mathbf{w}} f(\mathbf{x}_i; \mathbf{w}) = \mathbf{0}$$

In compact matrix notation:

$$\nabla_{\mathbf{w}} L(\mathbf{w}) = J(\mathbf{w})^\top (\mathbf{y} - F(\mathbf{w})) = \mathbf{0}$$

where

$$F(\mathbf{w}) = \begin{bmatrix} f(\mathbf{x}_1; \mathbf{w}) \\ \vdots \\ f(\mathbf{x}_n; \mathbf{w}) \end{bmatrix}, \quad J(\mathbf{w}) = \begin{bmatrix} \nabla_{\mathbf{w}} f(\mathbf{x}_1; \mathbf{w})^\top \\ \vdots \\ \nabla_{\mathbf{w}} f(\mathbf{x}_n; \mathbf{w})^\top \end{bmatrix}$$

J is also referred to as the **Jacobian** of F. Observe that in the special case when f is linear in $\mathbf{w}$ (i.e. $f(\mathbf{x}_i; \mathbf{w}) = \mathbf{w}^\top \mathbf{x}_i$), the gradient $\nabla_{\mathbf{w}} L(\mathbf{w})$ will only depend $\mathbf{w}$ in $F(\mathbf{w})$ because the term $\nabla_{\mathbf{w}} f(\mathbf{x}_i; \mathbf{w})$ will only depend on $\mathbf{x}_i$:

$$\nabla_{\mathbf{w}} L(\mathbf{w}) = \sum_{i=1}^{n} (y_i - \mathbf{w}^\top \mathbf{x}_i) \nabla_{\mathbf{w}} (\mathbf{w}^\top \mathbf{x}_i) = \sum_{i=1}^{n} (y_i - \mathbf{w}^\top \mathbf{x}_i) \mathbf{x}_i = \mathbf{X}^\top (\mathbf{y} - \mathbf{X}\mathbf{w})$$

and we can derive a **closed-form** solution for $\mathbf{w}$, arriving at the OLS solution:

$$\begin{aligned} \mathbf{X}^\top (\mathbf{y} - \mathbf{X}\mathbf{w}) &= \mathbf{0} \\ \mathbf{X}^\top \mathbf{y} - \mathbf{X}^\top \mathbf{X}\mathbf{w} &= \mathbf{0} \\ \mathbf{X}^\top \mathbf{y} &= \mathbf{X}^\top \mathbf{X}\mathbf{w} \\ \mathbf{w} &= (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y} \end{aligned}$$

In the general case where f is nonlinear in $\mathbf{w}$, it is not necessarily possible to derive a closed-form solution for $\mathbf{w}$, for a few reasons. First of all, without additional assumptions on f, the NLS objective may not be convex. Therefore there may exist values of $\mathbf{w}$ that are not global minima, but nonetheless $\nabla_{\mathbf{w}} L(\mathbf{w}) = \mathbf{0}$ — they could be local minima, saddle points, or worse, local maxima! Second of all, even if the objective is convex, we may not be able to solve the equation $J(\mathbf{w})^\top (\mathbf{y} - F(\mathbf{w})) = \mathbf{0}$ for $\mathbf{w}$. Given the challenges that nonlinear least squares introduces over linear least squares, we need a principled approach to solve problems that have no closed-form solution, preferably agnostic of the specific objective itself.

2 Optimization

In the specific case of nonlinear least squares the objective is

$$\min_{\mathbf{w}} \frac{1}{2} \sum_{i=1}^{n} (y_i - f(\mathbf{x}_i; \mathbf{w}))^2$$

but more generally, as we move into the realm of neural networks and beyond, we will be solving arbitrary problems of the form

$$\min_{\mathbf{w} \in \mathcal{X}} f(\mathbf{w})$$

over an arbitrary *continuous* objective function $f : \mathbb{R}^d \to \mathbb{R}$ and arbitrary domain $\mathcal{X}$. If we are able to solve this more general class of problems, then we can solve nonlinear least squares as a specific instance of the problem. Solving such problems is the focus of **optimization**, an extensive field that has applications in control theory, finance, and machine learning.

In optimization we are interested in finding the **global minimum** of a function. In the pursuit of finding the global minimum, we may encounter **local minima** along the way, which are suboptimal but may actually be close enough to the global minimum. More broadly, such points belong to the class of **critical points**, the "interesting" points of deflection that we may want to consider when finding minima:

(i) **local minimum**: a differentiable point $\mathbf{w} \in \mathcal{X}$ such that there exists a *neighborhood* around $\mathbf{w}$ where $f(\mathbf{w})$ attains the minimum value

(ii) **local maximum**: a differentiable point $\mathbf{w} \in \mathcal{X}$ such that there exists a *neighborhood* around $\mathbf{w}$ where $f(\mathbf{w})$ attains the maximum value

(iii) **saddle point**: a differentiable point $\mathbf{w} \in \mathcal{X}$ such that for all *neighborhoods* around $\mathbf{w}$, there exists $\mathbf{u}, \mathbf{v}$ such that $f(\mathbf{u}) \leq f(\mathbf{w}) \leq f(\mathbf{v})$

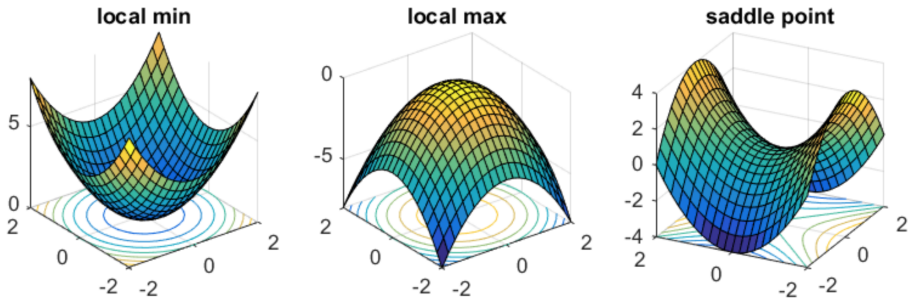

Figure 1: Source: Off the Convex Path

Technically, local minima must exist within a *neighborhood* of the domain and be *differentiable*, and our analysis of minima isn't complete without also considering the following as potential minima:

(i) **boundary points**: points $\mathbf{w} \in \mathcal{X}$ that can be approached from both $\mathcal{X}$ and outside $\mathcal{X}$, or more intuitively, points that lie of the "boundary" of $\mathcal{X}$

(ii) **non-differentiable points**: points at which the derivative is undefined, such as points with "kinks"

For the remainder of our discussion, we will assume that we are solving unconstrained optimization problems with differentiable functions, so that we will only have to consider critical points in our analysis.

The categorization in the previous section is helpful, but how exactly can we determine which points in the domain are critical points? As it turns out, the set of all critical points is simply the set of points at which the gradient is zero. Given that f is continuously differentiable, the **gradient** is defined as the vector of partial derivatives of f, denoted by

$$\nabla f = \begin{bmatrix} \frac{\partial f}{\partial w_1} \\ \frac{\partial f}{\partial w_2} \\ \vdots \\ \frac{\partial f}{\partial w_d} \end{bmatrix}$$

Since the set of points for which the gradient is zero in turn define the set of critical points, the gradient being zero is a necessary condition for local minima.

Proposition 1. *If $\mathbf{w}^*$ is a local minimum of f and f is continuously differentiable in a neighborhood of $\mathbf{w}^*$, then $\nabla f(\mathbf{w}^*) = \mathbf{0}$.*

Proof. See math4ml. □

This justifies the technique we have been using on numerous occasions so far to solve least squares problems: setting the gradient of the objective function to zero and solving the corresponding equation. Note however, that while setting the gradient to zero is a necessary condition for local minima, it is not a *sufficient* condition. In many circumstances, the function that we are optimizing may not have a local minima, and generally setting the gradient to zero could yield local maxima or saddle points. Even if all critical points were minima, we would still have to solve the corresponding equation $\nabla f(\mathbf{w}) = \mathbf{0}$, which is not always trivial. In the cases when solving this equation is intractable, we say that no **closed-form solution** exists, and therefore an *iterative algorithm* is needed to solve the optimization problem. Even if a problem does have a closed form solution that we can directly find, it may still be much more computationally efficient to solve the problem with iterative algorithms.

3 Gradient Descent

Rather than using gradients to find the closed-form solution, we can use gradients to "creep toward" a local minimum in an iterative fashion. **Gradient Descent** is an algorithm that iteratively takes

small steps in the direction of *steepest descent* of the objective f. Intuitively, we can view gradient descent as a ball rolling down a hill. If we place the ball somewhere at the top of the hill, it will naturally roll down the direction of steepest descent until it reaches the bottom of the hill, at which point it may oscillate around until it eventually comes to a stop at the bottom.

Gradient descent is a simple, intuitive method that works remarkably well in practice. One question that remains is: how exactly do we determine the direction of steepest descent of a multivariate function, and what does this method have to do with gradients? Given that we are currently at a point $\mathbf{w}^{(t)}$ in the domain of the function, the direction of steepest descent is the negative of the gradient at that point, $-\nabla f(\mathbf{w}^{(t)})$. To see why, recall that the directional derivative in a unit direction $\mathbf{u}$ at $\mathbf{w}^{(t)}$ is defined as the inner product of the gradient and the direction:

$$D_{\mathbf{u}} f(\mathbf{w}^{(t)}) = \angle \nabla f(\mathbf{w}^{(t)}), \mathbf{u} = \|\nabla f(\mathbf{w}^{(t)})\| \cdot \|\mathbf{u}\| \cdot \cos(\theta)$$

where θ is the angle between $\nabla f(\mathbf{w}^{(t)})$ and $\mathbf{u}$. Finding the direction of steepest descent entails finding the direction that minimizes the directional derivative. We can minimize the directional derivative by setting $\theta = -\pi$, which will mean that the direction $\mathbf{u}$ and $\nabla f(\mathbf{w}^{(t)})$ are opposite to each other, and thus the direction of steepest descent $\mathbf{u}^*$ is $-\nabla f(\mathbf{w}^{(t)})$ (similarly the direction of steepest *ascent* is $\nabla f(\mathbf{w}^{(t)})$). The gradient descent algorithm will take an arbitrary step in this direction, scaling the gradient by a scalar α_t.

vlined 1 Gradient Descent

Initialize $\mathbf{w}^{(0)}$ to a random point

while $f(\mathbf{w}^{(t)})$ not converged **do** $\mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} - \alpha_t \nabla f(\mathbf{w}^{(t)})$

Determining this scaling α_t is dependent on the attributes of the function f. Sometimes we can set the scaling to a constant value and converge to the optimum value, whereas in other instances we need to determine an *adaptive stepsize*. A scaling that is too high may cause the algorithm to diverge from the optimal solution, whereas a scaling that is too low may cause the algorithm to converge too slowly. For certain classes of functions, there are theoretical guarantees that establish convergence, which we will state later.

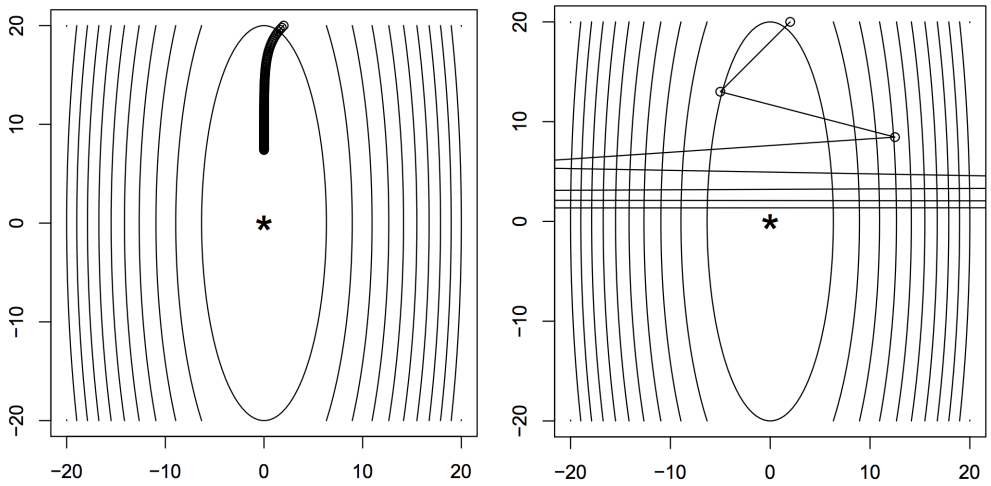

Figure 2: In gradient descent, stepsize matters. A small stepsize (left) will never converge to the optimal point, and a large stepsize (right) will lead to divergence. Source: CMU 10-725

3.1 Stochastic Gradient Descent

In many machine learning applications, the loss f that we want to minimize can be decomposed into a sum of functions, that is $f(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n} f_i(\mathbf{w})$. This holds in particular for problems involving an average over the training data, which is for example the case when we want to find the maximum likelihood estimator given i.i.d. data in a generative probabilistic model. In the standard gradient descent update, computing the gradient effectively entails computing and adding n separate gradients:

$$\nabla f(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n} \nabla f_i(\mathbf{w})$$

This standard form of gradient descent is commonly referred to as **batch gradient descent**, because it computes a full "batch" of gradients in each update. Assuming that the objective function f is deterministic, and given a fixed initial iterate $\mathbf{w}^{(0)}$, batch gradient descent is a deterministic algorithm.

One major issue with batch gradient descent is that it can be computationally expensive, because it requires computing and adding n separate gradients. In addition, due to the deterministic nature of the algorithm, it can easily get stuck at local minima and saddle points. We can mitigate these issues by deploying **stochastic gradients**. Given a fixed $\mathbf{w}$, the stochastic gradient $G(\mathbf{w})$ is a random vector-valued function which is equal to the gradient $\nabla f(\mathbf{w})$ *in expectation*, i.e. $\mathbb{E}_G[G(\mathbf{w})] = \nabla f(\mathbf{w})$, where the expectation is over the stochasticity of the gradient. We can therefore say that the stochastic gradient is an *unbiased estimate* of the true gradient. The stochastic gradient is used in place of the true gradient in the update rule:

$$\mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} - \alpha_t \nabla G(\mathbf{w}^{(t)})$$

Mini-batch gradient descent is a stochastic variant of batch gradient descent, that instead of summing an entire "batch" of n gradients, samples and adds a random "mini-batch" of gradients

over $k < n$ indices drawn from $\{1, \ldots, n\}$:

$$G(\mathbf{w}) = \frac{1}{k} \sum_{i=1}^{k} \nabla f_i(\mathbf{w})$$

A major advantage of mini-batch gradient descent is that each iteration is now more computationally efficient, leading to greater progress and allowing us to monitor the performance of the algorithm faster. In addition, mini-batch gradient descent can escape local minima with more ease compared to batch gradient descent, due the noisy nature of its gradients. However note that this can also lead to instability if the stochastic gradients have high variance. For this reason, mini-batch gradient descent generally requires a higher number of overall iterations to match the performance of batch gradient descent, which can lead to expensive computational overhead. Given the appropriate choice of k, mini-batch gradient descent can be significantly more computationally efficient overall than batch gradient descent.

The special case of mini-batch gradient descent with $k = 1$ is called **stochastic gradient descent (SGD)**. In this case, we can define the stochastic gradient by just drawing an index i uniformly at random from $\{1, \ldots, n\}$ and setting

$$G(\mathbf{w}) = \nabla f_i(\mathbf{w})$$

We can verify that the stochastic gradient is indeed an unbiased estimate of the true gradient:

$$\begin{aligned} \mathbb{E}_i[G(\mathbf{w})] = \mathbb{E}_i[\nabla f_i(\mathbf{w})] &= \sum_{j=1}^{n} P(i = j) \nabla f_j(\mathbf{w}) \\ &= \frac{1}{n} \sum_{j=1}^{n} \nabla f_j(\mathbf{w}) = \nabla f(\mathbf{w}) \end{aligned}$$

vlined 2 Stochastic Gradient Descent

Initialize $\mathbf{w}^{(0)}$ to a random point

while $f(\mathbf{w}^{(t)})$ not converged **do** Sample a random index i_t from $\{1, \ldots, n\}$
$\mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} - \alpha_t \nabla f_{i_t}(\mathbf{w}^{(t)})$

Compared to batch gradient descent, the gradient updates in SGD are significantly faster, but SGD often requires a significantly higher number of updates. In practice, mini-batch gradient descent is more effective than SGD and batch gradient descent, capturing the stability of batch gradient descent while at the same time injecting enough stochasticity to escape local minima and saddle points.

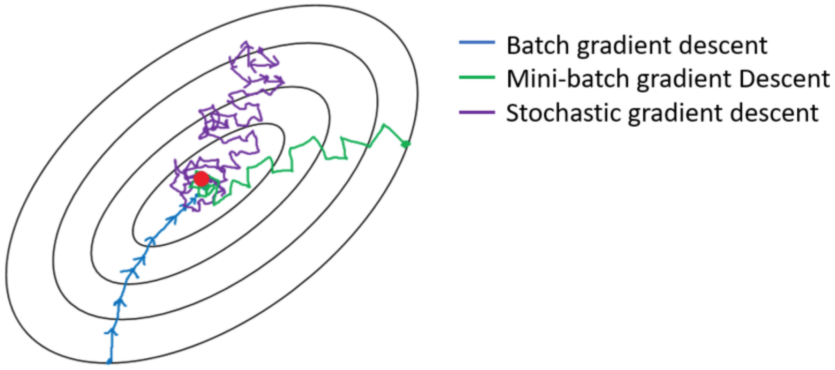

Figure 3: Increasing the batch size will lead to more stability at the cost of higher computational costs. Source: Towards Data Science

A fair metric that we can use to compare batch, mini-batch, and stochastic gradient descent is through the concept of **epochs**. An epoch is a measure of time — it is defined as the number of iterations in order to traverse the training data once. In the case of batch gradient descent, since all n examples comprising the training data are used to compute the gradient at each iteration, an epoch is simply equivalent to one iteration. In the case of SGD, since we only sample one example at each iteration, an epoch is equivalent to n iterations. In the case of mini-batch gradient descent, as epoch comprises of $\frac{n}{k}$ iterations. In practice, given the same number of epochs, mini-batch gradient descent tends to perform the best.

3.2 Momentum

Just as mini-batch gradient descent can lead us to escape local minima and saddle points, the stochastic nature of the algorithm can often lead to oscillations that cause instability and slow convergence. These issues are not just unique to stochastic gradients and can arise in the deterministic case, for example when the objective function is disproportionately scaled — ie. the function is elongated along one axis while being contracted along along another, giving the illusion of "ravines" in the function landscape. The disproportionate scalings cause the algorithm to make large leaps in contracted directions, while making very slow progress along elongated directions. The resulting behavior is a series of oscillations that may reach the optimal point very slowly.

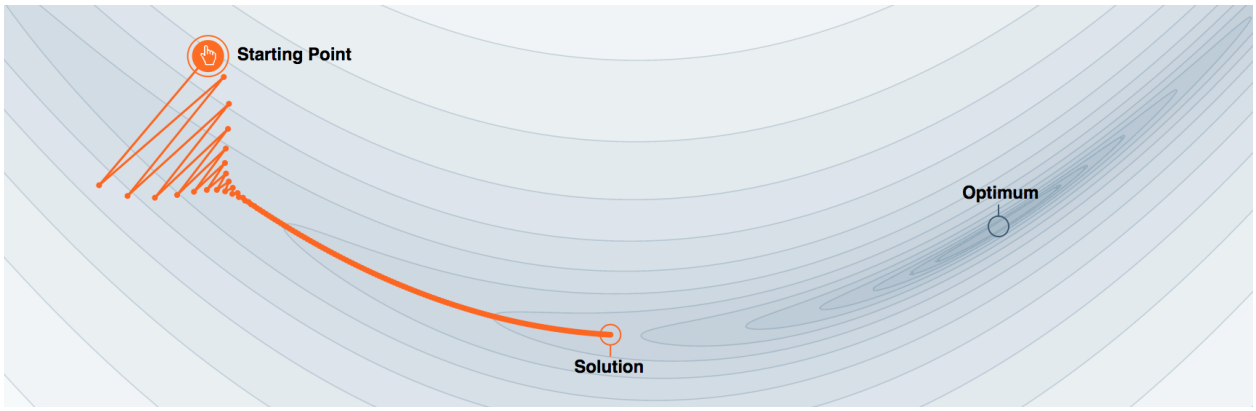

Figure 4: Standard gradient descent cannot converge to the optimum when the objective function is disproportionately scaled. Source: distill.pub

Polyak's heavy ball method addresses these issues by introducing a *momentum* term that adds inertia to the iterates and prevents them from deviating from the overall direction of the updates. Rather than updating the iterate $\mathbf{w}^{(t)}$ using the gradient $\nabla f(\mathbf{w}^{(t)})$, Polyak's heavy ball method uses $\nabla f(\mathbf{w}^{(t)})$ along with a history of all the gradients from the iterates seen so far. Specifically, it updates the iterates via a velocity term $\mathbf{v}^{(t)}$ that represents an exponential moving average of all of the gradients seen so far.

vlined 3 Polyak's Heavy Ball Method

Initialize $\mathbf{w}^{(0)}$ to a random point
Initialize $\mathbf{v}^{(0)}$ to $-\alpha_0 \nabla f(\mathbf{w}^{(0)})$

while $f(\mathbf{w}^{(t)})$ not converged **do** $\mathbf{v}^{(t)} \leftarrow \beta_t \mathbf{v}^{(t-1)} - \alpha_t \nabla f(\mathbf{w}^{(t)})$
$\mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} + \mathbf{v}^{(t)}$

The velocity term is updated in the following recursive fashion:

$$\mathbf{v}^{(t)} \leftarrow \beta_t \mathbf{v}^{(t-1)} - \alpha_t \nabla f(\mathbf{w}^{(t)})$$

which when unrolled, is equivalent to

$$\begin{aligned}\mathbf{v}^{(t)} \leftarrow & -\beta_t \cdot \beta_{t-1} \ldots \beta_1 \alpha_0 \nabla f(\mathbf{w}^{(0)}) \\ & -\beta_t \cdot \beta_{t-1} \ldots \beta_2 \alpha_1 \nabla f(\mathbf{w}^{(1)}) \\ & \ldots \\ & -\alpha_t \nabla f(\mathbf{w}^{(t)})\end{aligned}$$

which in the case when the β's and α's are constant is equivalent to

$$\mathbf{v}^{(t)} \leftarrow -\beta^t \alpha \nabla f(\mathbf{w}^{(0)}) - \beta^{t-1} \alpha \nabla f(\mathbf{w}^{(1)}) - \ldots - \alpha \nabla f(\mathbf{w}^{(t)})$$

The motivation for using a moving average of the gradients is as follows: we want to downplay the directions for which the gradient is oscillating over time and boost the directions for which

the gradient is constant over time. When we are in a "ravine" this has the effect of "killing" the gradient in constricted directions whose derivatives oscillate over time, accelerating convergence.

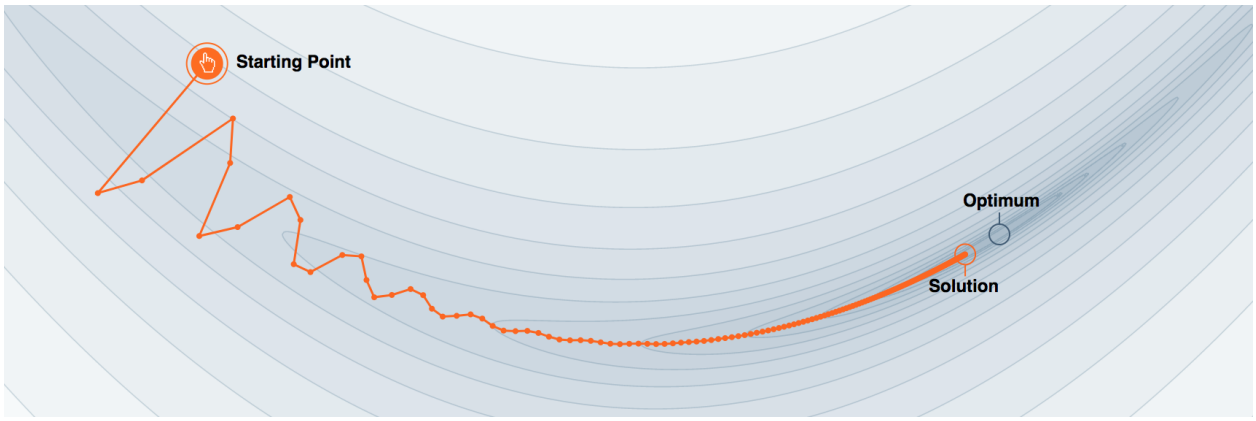

Figure 5: Polyak's heavy ball method uses momentum to dampen oscillations, accelerating convergence to the optimum point. Source: distill.pub

There is an alternative interpretation of Polyak's heavy ball method that is condensed to just one line:

$$\mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} - \alpha_t \nabla f(\mathbf{w}^{(t)}) + \beta_t(\mathbf{w}^{(t)} - \mathbf{w}^{(t-1)})$$

We can establish equivalence through the following manipulations:

$$\begin{aligned}\mathbf{w}^{(t+1)} &= \mathbf{w}^{(t)} + \mathbf{v}^{(t)} \\ &= \mathbf{w}^{(t)} + (-\alpha_t \nabla f(\mathbf{w}^{(t)}) + \beta_t \mathbf{v}^{(t-1)}) \\ &= \mathbf{w}^{(t)} - \alpha_t \nabla f(\mathbf{w}^{(t)}) + \beta_t(\mathbf{w}^{(t)} - \mathbf{w}^{(t-1)})\end{aligned}$$

Polyak's heavy ball method uses information about past iterates to determine the descent direction. **Nesterov's accelerated gradient descent** improves on this reasoning, incorporating information about potential future iterates as well. The only difference in Nesterov's accelerated gradient descent is that it computes a "lookahead gradient" $\nabla f(\mathbf{w}^{(t)} + \beta_t \mathbf{v}^{(t-1)})$ instead of the gradient at the current iterate $\nabla f(\mathbf{w}^{(t)})$. Effectively, we are performing a one step "look ahead" of the gradient and moving in that direction, potentially correcting for oscillations ahead of us.

vlined 4 Nesterov's Accelerated Gradient Descent

Initialize $\mathbf{w}^{(0)}$ to a random point
Initialize $\mathbf{v}^{(0)}$ to $-\alpha_0 \nabla f(\mathbf{w}^{(0)})$

while $f(\mathbf{w}^{(t)})$ not converged **do** $\mathbf{v}^{(t)} \leftarrow \beta_t \mathbf{v}^{(t-1)} - \alpha_t \nabla f(\mathbf{w}^{(t)} + \beta_t \mathbf{v}^{(t-1)})$
$\mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} + \mathbf{v}^{(t)}$

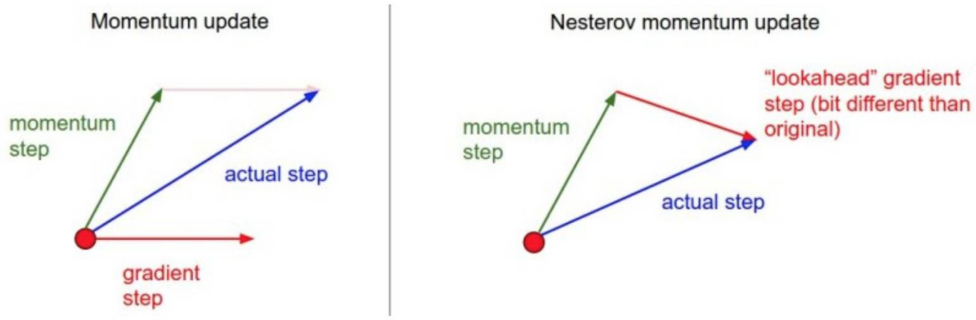

Figure 6: Polyak's heavy ball method applies gradient before update, while Nesterov's accelerated gradient descent applies gradient after update. Source: Stanford CS 231n

Through the same manipulations that we showed for Polyak's heavy ball method, we derive the one-line update for Nesterov's accelerated gradient descent:

$$\mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} - \alpha_t \nabla f(\mathbf{w}^{(t)} + \beta(\mathbf{w}^{(t)} - \mathbf{w}^{(t-1)})) + \beta_t(\mathbf{w}^{(t)} - \mathbf{w}^{(t-1)})$$

4 Line Search

Line search is another iterative optimization algorithm that, instead of taking small gradient steps, repeatedly slices the function across a 1 dimensional line and finds the minimum. Normally, finding the (global) minimum of d functions is an extremely difficult problem, but in this case we are only doing so for 1 dimensional "sliced" functions, which is a much more trivial task. Each iteration of line search entails three steps: (1) choosing a promising descent direction (or sometimes a random direction), (2) looking ahead in that direction and (roughly) finding the minimum, and (3) going to that minimum.

vlined 5 Line Search

Initialize $\mathbf{w}^{(0)}$ to a random point

while $f(\mathbf{w}^{(t)})$ not converged **do** Find a descent direction $\mathbf{u}^{(t)}$
Find $\alpha_t \in \mathbb{R}_+$ to minimize $h(\alpha) = f(\mathbf{w}^{(t)} + \alpha \mathbf{u}^{(t)})$
$\mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} - \alpha_t \mathbf{u}^{(t)}$

There are several options for the direction, such as the negative gradient (which is used in gradient descent). Broadly, we can pick any descent direction — a direction which entails to a negative directional derivate:

$$D_{\mathbf{u}} f(\mathbf{w}^{(t)}) = \angle \nabla f(\mathbf{w}^{(t)}), \mathbf{u} < 0$$

Another common choice is to choose an arbitrary coordinate (for example, the x/y/z coordinate in 3D), a line search variant called **coordinate descent**. Note that the minimization in the second step does not necessarily need to be exact. A simple approach is to sample several points across the line and choose the minimum, in a grid search fashion.

Line search methods offer a few advantages over gradient descent methods. For one, they do not necessarily require gradients, which can be particularly helpful in non-differentiable domains.

Also, they are potentially more robust to local minima, because they find the global minima of the 1D functions ahead of them.

5 Convex Optimization

A critical issue with the methods we have presented so far is that they can get stuck in local minima. With gradient descent for example, moving in the direction of steepest descent is a *greedy* choice that can cause convergence to a poor local minimum, depending on the initial starting point of the algorithm.

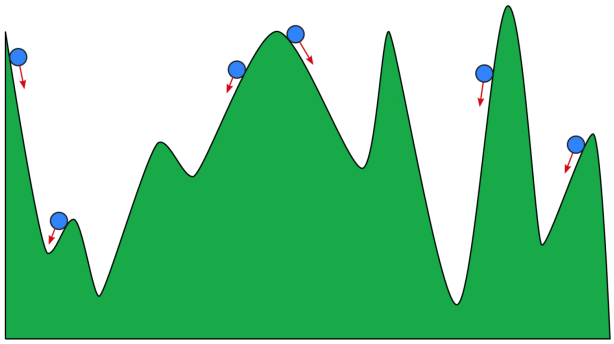

Figure 7: Depending on the initialization of gradient descent, the algorithm will converge to different local minima. Source: Towards Data Science

Convex functions conveniently eliminate this problem due to their "bowl shape," which ensures that all local minima are global minima.

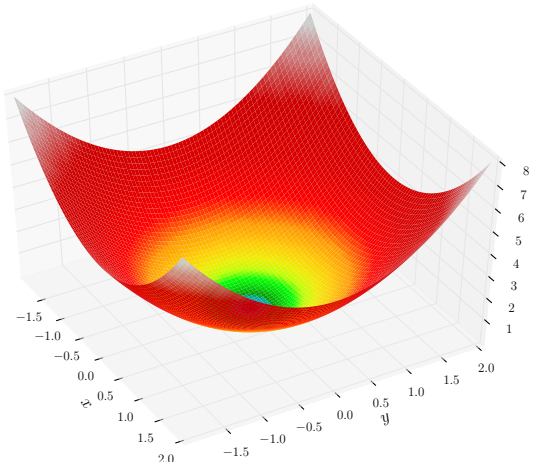

Figure 8: Source: Wikipedia

Due to this property, optimizing convex functions entails nice theoretical convergence rates that are otherwise not guaranteed for non-convex functions. For these reasons, there is a dedicated subfield of optimization called **convex optimization** that focuses on optimization problems with convex functions and convex constraints.

Given that $f : \mathbb{R}^n \to \mathbb{R}$ is twice continuously differentiable, the following are equivalent conditions of convexity:

(i) $f(t\mathbf{w}_1 + (1-t)\mathbf{w}_2) \leq tf(\mathbf{w}_1) + (1-t)f(\mathbf{w}_2), \quad \forall \mathbf{w}_1, \mathbf{w}_2, t \in [0,1]$

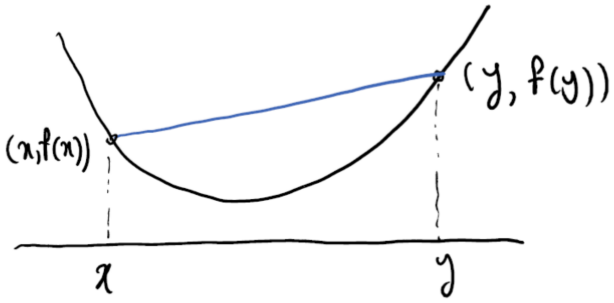

Figure 9: Any line segment connecting two points of a convex function must lie above the function. Source: Princeton University ORF 523

(ii) $f(\mathbf{w}_2) \geq f(\mathbf{w}_1) + \nabla f(\mathbf{w}_1)^\top (\mathbf{w}_2 - \mathbf{w}_1), \quad \forall \mathbf{w}_1, \mathbf{w}_2$

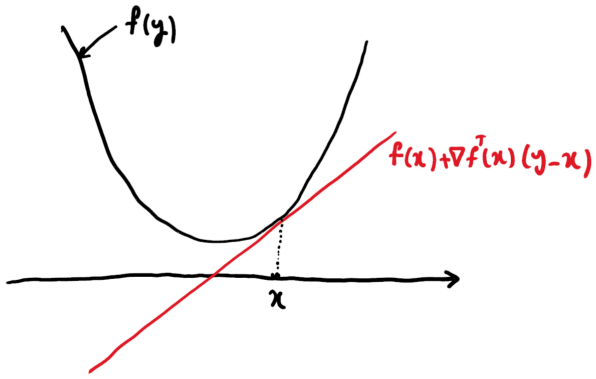

Figure 10: Any line tangent to a convex function must lie below the function. Source: Princeton University ORF 523

(iii) $(\nabla f(\mathbf{w}_2) - \nabla f(\mathbf{w}_1))^\top (\mathbf{w}_2 - \mathbf{w}_1) \geq 0, \quad \forall \mathbf{w}_1, \mathbf{w}_2$

(iv) $\nabla^2 f(\mathbf{w}) \succeq \mathbf{0}, \quad \forall \mathbf{w}$

Let's study these properties closely. The first condition states that for any two points $\mathbf{w}_1, \mathbf{w}_2$, the function lies below the line segment connecting $\mathbf{w}_1$ and $\mathbf{w}_2$. The next condition states that any tangent line to f must lie below the entire function. The third condition intuitively states that if $\mathbf{w}_2$ is greater than $\mathbf{w}_1$, then the derivative of $\mathbf{w}_2$ is also greater than the derivative of $\mathbf{w}_1$.

Finally, the last condition states that the "second derivative" of f is always non-negative. More rigorously, we can generalize the concept of second derivatives in higher dimensions with the Hessian. Given that f is twice continuously differentiable, we define the **Hessian** as the matrix of

second partial derivatives of f, denoted by

$$\nabla^2 f = \begin{bmatrix} \frac{\partial^2 f}{\partial w_1^2} & \cdots & \frac{\partial^2 f}{\partial w_1 \partial w_d} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial w_d \partial w_1} & \cdots & \frac{\partial^2 f}{\partial w_d^2} \end{bmatrix}$$

The Hessian being PSD is a necessary condition for local minima:

Proposition 2. *If $\mathbf{w}^*$ is a local minimum of f and f is twice continuously differentiable in a neighborhood of $\mathbf{w}^*$, then $\nabla^2 f(\mathbf{w}^*)$ is positive semi-definite and $\nabla f(\mathbf{w}^*) = \mathbf{0}$.*

Proof. See math4ml. □

Unfortunately, the gradient being zero and the Hessian being PSD together are necessary but not sufficient conditions local minima (consider the function $f(w) = w^3$ or $f(w) = -w^4$). However, the gradient being zero and the Hessian being PSD in a *neighborhood* are sufficient conditions.

Proposition 3. *Suppose f is twice continuously differentiable with $\nabla^2 f$ positive semi-definite in a neighborhood of $\mathbf{w}^*$, and that $\nabla f(\mathbf{w}^*) = \mathbf{0}$. Then $\mathbf{w}^*$ is a local minimum of f.* [1]

Proof. See math4ml. □

Since for convex functions the Hessian is PSD at all points in the domain, any critical point is a local minimum. In fact, any local minimum is also a global minimum, so any point for which the gradient is zero must be the global minimum.

Proposition 4. *Let $\mathcal{X}$ be a convex set. If f is convex, then any local minimum of f in $\mathcal{X}$ is also a global minimum.*

Proof. See math4ml. □

Consequently we can find any point for which the gradient is zero and guarantee that it is the global minimum (this is exactly the case in OLS and Ridge Regression since the objective function is PSD and therefore convex). Note however, that this does not imply that the global minimum is unique — there could be several different points which achieve the global minimum.

5.1 Strong Convexity

While convex functions guarantee that all local minima are global minima, they do not guarantee that the global minimum is satisfied uniquely. Strongly convexity is an extension that guarantees this property. For a strictly positive $m \in \mathbb{R}$, a function is **m-strongly convex** if the following equivalent conditions hold:

[1] A subtle point: if $\nabla^2 f(\mathbf{w}^*)$ is *positive definite* and $\nabla f(\mathbf{w}^*) = \mathbf{0}$, then $\mathbf{w}^*$ is a strict local minimum. We do not have to check that the Hessian is PSD in a neighborhood of $\mathbf{w}^*$, as this condition is implied from the fact that f is twice continuously differentiable.

(i) $f(t\mathbf{w}_1 + (1-t)\mathbf{w}_2) \leq tf(\mathbf{w}_1) + (1-t)f(\mathbf{w}_2) - \frac{t(1-t)m}{2}\|\mathbf{w}_2 - \mathbf{w}_1\|^2, \quad \forall \mathbf{w}_1, \mathbf{w}_2, t \in [0,1]$

(ii) $g(\mathbf{w}) = f(\mathbf{w}) - \frac{m}{2}\|\mathbf{w}\|^2$ is convex

(iii) $f(\mathbf{w}_2) \geq f(\mathbf{w}_1) + \nabla f(\mathbf{w}_1)^\top(\mathbf{w}_2 - \mathbf{w}_1) + \frac{m}{2}\|\mathbf{w}_2 - \mathbf{w}_1\|^2, \quad \forall \mathbf{w}_1, \mathbf{w}_2$

(iv) $(\nabla f(\mathbf{w}_2) - \nabla f(\mathbf{w}_1))^\top(\mathbf{w}_2 - \mathbf{w}_1) \geq m\|\mathbf{w}_2 - \mathbf{w}_1\|^2, \quad \forall \mathbf{w}_1, \mathbf{w}_2$

(v) $\nabla^2 f(\mathbf{w}) \succeq m\mathbf{I}, \quad \forall \mathbf{w}$

The conditions for strong convexity are identical to those for convex functions, but with an additional term involving m. Strongly convex functions provide several advantages over general convex functions. From the third condition, we see that strongly convex functions can be lower bounded by a quadratic function, which establishes the *uniqueness* of a global minimum.

Proposition 5. *Let $\mathcal{X}$ be a convex set. If f is strongly convex, then there exists at exactly one local minimum of f in $\mathcal{X}$. Consequently, it is the unique global minimum of f in $\mathcal{X}$.*

Proof. See math4ml. □

If the Hessian of $\nabla^2 f$ has eigenvalues that are all strictly positive at all points, then f is m-strongly convex with m equal to the the smallest eigenvalue of $\nabla^2 f$ (over all points $\mathbf{w}$). Recall from our discussion of OLS vs. Ridge Regression that while OLS may have several solutions, Ridge Regression has a unique solution. This is because the Ridge Regression formulation is positive definite and thus strongly convex, while OLS is positive semi-definite and not necessarily strongly convex.

5.2 Smoothness

While strongly convex functions are *lower bounded* by a quadratic function, smooth functions are *upper bounded* by a quadratic function. [2]

An M**-smooth** (or more formally **Lipschitz continuous gradient**) function is one for which there exists a strictly positive $M \in \mathbb{R}$ such that

$$\|\nabla f(\mathbf{w}_2) - \nabla f(\mathbf{w}_1)\| \leq M\|\mathbf{w}_2 - \mathbf{w}_1\|, \quad \forall \mathbf{w}_1, \mathbf{w}_2 \tag{1}$$

This definition does not assume that f is convex. **??** implies all of the following equivalent conditions:

(i) $f(t\mathbf{w}_1 + (1-t)\mathbf{w}_2) \geq tf(\mathbf{w}_1) + (1-t)f(\mathbf{w}_2) - \frac{t(1-t)M}{2}\|\mathbf{w}_2 - \mathbf{w}_1\|^2, \quad \forall \mathbf{w}_1, \mathbf{w}_2, t \in [0,1]$

(ii) $f(\mathbf{w}_2) \leq f(\mathbf{w}_1) + \nabla f(\mathbf{w}_1)^\top(\mathbf{w}_2 - \mathbf{w}_1) + \frac{M}{2}\|\mathbf{w}_2 - \mathbf{w}_1\|^2, \quad \forall \mathbf{w}_1, \mathbf{w}_2$

(iii) $(\nabla f(\mathbf{w}_2) - \nabla f(\mathbf{w}_1))^\top(\mathbf{w}_2 - \mathbf{w}_1) \leq M\|\mathbf{w}_2 - \mathbf{w}_1\|^2, \quad \forall \mathbf{w}_1, \mathbf{w}_2$

(iv) $\nabla^2 f(\mathbf{w}) \preceq M\mathbf{I}, \quad \forall \mathbf{w}$

[2] Not to be confused with smooth functions in the context of real analysis

When f is convex, then the above conditions also imply **??**, establishing equivalence among all of the conditions. Roughly speaking, smoothness is the counterpart to strong convexity, with the inequality signs flipped. If the Hessian of $\nabla^2 f$ has eigenvalues that are bounded from above, f is M-smooth with M equal to the the maximum eigenvalue of $\nabla^2 f$ (over all points $\mathbf{x}$).

5.3 Gradient Descent Convergence under Convexity

While gradient descent does not have convergence guarantees in general, we can make theoretical guarantees when the function is convex. Furthermore, strong convexity and smoothness will provide lower and upper bounds for f respectively, allowing us to achieve a significantly faster convergence rate. Assuming that the distance from the initial point $\mathbf{w}^{(0)}$ and the optimal point $\mathbf{w}^*$ is R, we have the following:

properties of f	stepsize α_t	convergence rate to $f(\mathbf{w}^*)$
convex, L-Lipschitz	$\frac{R}{L\sqrt{t}}$	$O(\frac{1}{\sqrt{t}})$
m-strongly convex, L-Lipschitz	$\frac{2}{m(t+1)}$	$O(\frac{1}{t})$
convex, M-smooth	$\frac{1}{M}$	$O(\frac{1}{t})$
m-strongly convex, M-smooth	$\frac{1}{M}$	$O(\exp\left(-t\frac{m}{M}\right))$

For detailed proofs of rates above, refer to the EE 227C lecture notes. Individually, strong convexity and smoothness will allow us to accelerate the rate of convergence from $O(\frac{1}{\sqrt{t}})$ to $O(\frac{1}{t})$. Put together, they allow us to achieve an exponential convergence rate — a significant acceleration! The quantity $\kappa = \frac{M}{m}$ is known as the **condition number** — the ratio of the largest over smallest singular value of the Hessian of f. Recall from our discussion of OLS vs. Ridge Regression that Ridge Regression adds a small penalty term $\lambda\|\mathbf{w}\|^2$ to the objective, effectively making the problem strongly convex. Since the OLS is already smooth as well, then gradient descent can achieve an exponential rate of convergence to the optimal value. The higher the value of λ, the lower the value of the condition number κ, which leads to an even faster convergence rate. This of course, comes at the costs of regularization.

6 Newton's Method

Up until this point, we have only considered first-order methods to optimize functions. Now, we will present **Newton's method**, an iterative method that utilizes second-order information to achieve a faster rate of convergence than existing first-order methods. Given an arbitrary twice continuously differentiable objective function f, Newton's Method iteratively minimizes the second-order Taylor expansion of the objective function. Given the current iterate $\mathbf{w}^{(t)}$, it minimizes the following objective:

$$\min_{\mathbf{w}} \bar{f}(\mathbf{w}) = f(\mathbf{w}^{(t)}) + \nabla f(\mathbf{w}^{(t)})^\top(\mathbf{w} - \mathbf{w}^{(t)}) + \frac{1}{2}(\mathbf{w} - \mathbf{w}^{(t)})^\top \nabla^2 f(\mathbf{w}^{(t)})(\mathbf{w} - \mathbf{w}^{(t)})$$

We can minimize $\bar{f}(\mathbf{w})$ by setting its gradient to zero:

$$\nabla \bar{f}(\mathbf{w}) = \nabla f(\mathbf{w}^{(t)}) + \nabla^2 f(\mathbf{w}^{(t)})(\mathbf{w} - \mathbf{w}^{(t)}) = \mathbf{0}$$

Which leads to the update rule (otherwise known as *Newton step*)

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \nabla^2 f(\mathbf{w}^{(t)})^{-1} \nabla f(\mathbf{w}^{(t)})$$

The updates for Newton's method and gradient descent are nearly identical:

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \alpha_t \nabla f(\mathbf{w}^{(t)}) \quad \text{(Gradient descent)}$$
$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \nabla^2 f(\mathbf{w}^{(t)})^{-1} \nabla f(\mathbf{w}^{(t)}) \quad \text{(Newton's method)}$$

We can think of gradient descent as a Newton update in which we approximate $\nabla^2 f(\mathbf{w}^{(t)})^{-1}$ by a scaled version of the identity. That is, gradient descent is equivalent to Newton's method when $\nabla^2 f(\mathbf{w}^{(t)})^{-1} = \alpha_t \mathbf{I}$ where $\mathbf{I}$ is the identity matrix.

The algorithm is as follows:

vlined 6 Newton's Method

Initialize $\mathbf{w}^{(0)}$ to a random point

while $f(\mathbf{w}^{(t)})$ not converged **do** $\mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} - \nabla^2 f(\mathbf{w}^{(t)})^{-1} \nabla f(\mathbf{w}^{(t)})$

6.1 Alternative Interpretation

Newton's method can equivalently be viewed as a a *root-finding algorithm* — specifically it finds the "roots" of the gradient by iteratively approximating the gradient and finding the root of the approximation. Newton's method is agnostic to the type of function that it optimizes — whether it is the gradient function, or just the objective function. At its simplest form, Newton's method can be used to find the roots of a single variable function $\varphi \colon \mathbb{R} \to \mathbb{R}$. Our goal is to find a root of the non-linear equation $\varphi(w) = 0$. Suppose we have a current estimate of the root $\varphi(w^{(t)})$. From Taylor's theorem, we can express the first-order form of $\varphi(w)$ with respect to $\varphi(w^{(t)})$ as

$$\varphi(w) = \varphi(w^{(t)}) + \varphi'(w) \cdot (w - w^{(t)}) + o(\left|w - w^{(t)}\right|)$$

given $\delta = w - w^{(t)}$ we equivalently have that

$$\varphi(w^{(t)} + \delta) = \varphi(w^{(t)}) + \varphi'(w) \cdot \delta + o(|\delta|)$$

Disregarding the $o(|\delta|)$ term, we solve (over δ) the following objective:

$$\varphi(w^{(t)}) + \varphi'(w^{(t)})\delta = 0$$

Then, $\delta = -\frac{\varphi(w^{(t)})}{\varphi'(w^{(t)})}$, leading to the iteration $w^{(t+1)} = w^{(t)} - \frac{\varphi(w^{(t)})}{\varphi'(w^{(t)})}$. We can similarly make an argument for a multivariate function $F : \mathbb{R}^d \to \mathbb{R}^d$. Our goal is to solve $F(\mathbf{w}) = \mathbf{0}$. Again, from Taylor's theorem we have that

$$F(\mathbf{w} + \Delta) = F(\mathbf{w}) + J_F(\mathbf{w})\Delta + o(\|\Delta\|)$$

where J_F is the Jacobian. This gives us $\Delta = -J_F^{-1}(\mathbf{w})F(\mathbf{w})$, and the iteration

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - J_F^{-1}(\mathbf{w}^{(t)})F(\mathbf{w}^{(t)})$$

In the context of optimization, Newton's method is a special application of this root-finding method, applied to the gradient function. That is, given that we are minimizing $f\colon \mathbb{R}^d \to \mathbb{R}$, Newton's method finds the roots of the gradient function $\nabla f\colon \mathbb{R}^d \to \mathbb{R}^d$. It uses the update rule

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \nabla^2 f(\mathbf{w}^{(t)})^{-1} \nabla f(\mathbf{w}^{(t)})$$

as the Hessian $\nabla^2 f(\mathbf{w}^{(t)})$ of the objective function corresponds to the Jacobian $J_F^{-1}(\mathbf{w})$ of the gradient. Let's understand the motivation of Newton's method in close detail. Our goal is to find local minima for f, points for which it is necessarily true that $\nabla f(\mathbf{w}) = \mathbf{0}$. Consequently, we wish to find points for which $\nabla f(\mathbf{w}) = \mathbf{0}$. The gradient $\nabla f(\mathbf{w})$ can be difficult or even intractable to work with, so instead we work with a first-order Taylor approximation of the gradient with respect to our current iterate $\mathbf{w}^{(t)}$. We solve for the roots of the first-order gradient, update our iterate, and repeat the process. Note that while solving $\nabla f(\mathbf{w}) = \mathbf{0}$ may yield local maxima or even saddle points, we are finding the roots of the linearized gradient, which is convex — therefore any point for which the first-order approximation of the gradient is zero yields a global minimum for the approximation.

6.2 Issues with Newton's Method

There are a few issues with Newton's method that we glossed over in our analysis. In general, there are no guarantees that Newton's method can converge, and even more concerning, the algorithm may get stuck as the Hessian $\nabla^2 f(\mathbf{w}^{(t)})$ may not be invertible. Placing invertibility issues aside, the most concerning issue is that Newton's method may not even be attempting to minimize the objective function. To see why, recall that the goal of each Newton step is to minimize the second-order approximation, which we do so by setting the gradient of the approximation to zero. This is not a sound step, as it may yield saddle points or maxima. This can happen when the Hessian $\nabla^2 f(\mathbf{w}^{(t)})$ has non-positive eigenvalues. In order to ensure that the second order approximation $\bar{f}(\mathbf{w})$ yields a unique global minimum, we must ensure that it is strongly convex. We can do so by regularizing the objective $f(\mathbf{w})$ with an additional $\lambda\|\mathbf{w}\|^2$ term, with an appropriately chosen λ that shifts all of the eigenvalues of the objective to be positive.

Even when the objective is strongly convex, Newton's method can be quite unpredictable. For example, consider the function

$$f(w) = \sqrt{w^2 + 1}$$

essentially a smoothed version of the absolute value $|x|$. Clearly, the function is minimized at $w^* = 0$. Calculating the necessary derivatives for Newton's method, we find

$$f'(w) = \frac{w}{\sqrt{w^2 + 1}}$$
$$f''(w) = (1 + w^2)^{-3/2}\,.$$

Note that $f(w)$ is strongly convex since its second derivative strictly positive and 1-smooth ($|f'(w)| < 1$). The Newton step for minimizing $f(w)$ is

$$w^{(t+1)} = w^{(t)} - \frac{f'(w^{(t)})}{f''(w^{(t)})} = -{w^{(t)}}^3\,.$$

The behavior of this algorithm depends on the magnitude of $w^{(t)}$. In particular, we have the following three regimes

$$\begin{cases} |w^{(t)}| < 1 & \text{Algorithm converges } \textit{cubically} \\ |w^{(t)}| = 1 & \text{Algorithm oscillates between } -1 \text{ and } 1 \\ |w^{(t)}| > 1 & \text{Algorithm diverges} \end{cases}$$

This example shows that even for strongly convex functions with Lipschitz gradients that Newton's method is only guaranteed to converge locally. To avoid divergence, a popular technique is to use a *damped* step–size:

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \alpha_t \nabla^2 f(\mathbf{w}^{(t)})^{-1} \nabla f(\mathbf{w}^{(t)})$$

6.3 Convergence Analysis

We can ensure that Newton's method converges, if all of the following conditions are met:

1. $\nabla^2 f(\mathbf{w})$ is Lipschitz: $\|\nabla^2 f(\mathbf{w}) - \nabla^2 f(\mathbf{w}')\| \leq \|\mathbf{w} - \mathbf{w}'\|$
2. $\exists \mathbf{w}^*$ s.t. $\nabla f(\mathbf{w}^*) = \mathbf{0}$ and $\nabla^2 f(\mathbf{w}^*) \succeq \alpha \mathbf{I}$ and $\|\mathbf{w}^{(0)} - \mathbf{w}^*\| \leq \frac{\alpha}{2}$

These conditions combined establish *local convergence* of Newton's method to a local minimum. That is, given that the initial point $\mathbf{w}^{(0)}$ is sufficiently close to the local minimum, the Hessian is positive definite at the local minimum, and the Hessian is Lipschitz (meaning that its rate of change can be bounded), we can ensure a quadratic convergence rate of $O(e^{-e^t})$, which is significantly faster than the fastest rate for gradient descent that we have seen, $O(e^{-t})$. Note however, that each Newton step will involve inverting the Hessian, which itself is an expensive $O(d^3)$ operation that becomes impractical for high dimensional functions.

7 Gauss-Newton Algorithm

Let's revisit the nonlinear least squares problem. We can try to apply all of the techniques and approaches we have covered so far to solve this problem, but there is a specialized algorithm for solving the nonlinear least squares problem, called **Gauss-Newton**. The Gauss-Newton algorithm has parallels to Newton's method, as they both repeatedly make linearly approximations of an objective and solve that approximation. At each iteration, this method linearly approximates the function F about the current iterate and solves a least-squares problem involving the linearization in order to compute the next iterate.

Let's say that we have a "guess" for $\mathbf{w}$ at iteration k, which we denote $\mathbf{w}^{(k)}$. We consider the first-order approximation of $F(\mathbf{w})$ about $\mathbf{w}^{(k)}$:

$$\begin{aligned} F(\mathbf{w}) \approx \tilde{F}(\mathbf{w}) &= F(\mathbf{w}^{(k)}) + \frac{\partial}{\partial \mathbf{w}} F(\mathbf{w}^{(k)})(\mathbf{w} - \mathbf{w}^{(k)}) \\ &= F(\mathbf{w}^{(k)}) + J(\mathbf{w}^{(k)}) \Delta \mathbf{w} \end{aligned}$$

where $\Delta \mathbf{w} := \mathbf{w} - \mathbf{w}^{(k)}$.

Now that $\tilde{F}$ is linear in $\Delta\mathbf{w}$ (the Jacobian and F are just constants: functions evaluated at $\mathbf{w}^{(k)}$), our objective is convex and we can perform linear least squares to form the closed form solution for $\Delta\mathbf{w}$. Applying the first-order optimality condition to the objective $\tilde{F}$ yields the following equation:

$$\mathbf{0} = J_{\tilde{F}}(\mathbf{w})^\top(\mathbf{y} - \tilde{F}(\mathbf{w})) = J(\mathbf{w}^{(k)})^\top\Big(\mathbf{y} - \Big(F(\mathbf{w}^{(k)}) + J(\mathbf{w}^{(k)})\Delta\mathbf{w}\Big)\Big)$$

Note that the Jacobian of the linearized function $\tilde{F}$, evaluated at any $\mathbf{w}$, is precisely $J(\mathbf{w}^{(k)})$. Denoting $\mathbf{J} = J(\mathbf{w}^{(k)})$ and $\Delta\mathbf{y} := \mathbf{y} - F(\mathbf{w}^{(k)})$ for brevity, we have

$$\begin{aligned}
\mathbf{J}^\top(\Delta\mathbf{y} - \mathbf{J}\Delta\mathbf{w}) &= \mathbf{0} \\
\mathbf{J}^\top\Delta\mathbf{y} &= \mathbf{J}^\top\mathbf{J}\Delta\mathbf{w} \\
\Delta\mathbf{w} &= (\mathbf{J}^\top\mathbf{J})^{-1}\mathbf{J}^\top\Delta\mathbf{y}
\end{aligned}$$

Comparing this solution to OLS, we see that it is effectively solving

$$\Delta\mathbf{w} = \arg\min_{\delta\mathbf{w}} \|\mathbf{J}\delta\mathbf{w} - \Delta\mathbf{y}\|^2$$

where $\mathbf{J}$ represents $\mathbf{X}$ in OLS, $\Delta\mathbf{y}$ represents $\mathbf{y}$ in OLS, and $\delta\mathbf{w}$ represents $\mathbf{w}$ in OLS. At each iteration we are effectively minimizing the objective with respect to the linearization of F at the current iterate $\mathbf{w}^{(k)}$. Since $\delta F \approx \mathbf{J}\delta\mathbf{w}$, we can expect that the minimization with respect to $\tilde{F}$ is also optimal with respect to F in the local region around $\mathbf{w}^{(k)}$. Recalling that $\Delta\mathbf{w} = \mathbf{w} - \mathbf{w}^{(k)}$, we can improve upon our current guess $\mathbf{w}^{(k)}$ with the update

$$\begin{aligned}
\mathbf{w}^{(k+1)} &= \mathbf{w}^{(k)} + \Delta\mathbf{w} \\
&= \mathbf{w}^{(k)} + (\mathbf{J}^\top\mathbf{J})^{-1}\mathbf{J}^\top\Delta\mathbf{y}
\end{aligned}$$

---

vlined 7 Gauss-Newton

Initialize $\mathbf{w}^{(0)}$ with some guess

while $\mathbf{w}^{(k)}$ has not converged **do** Compute Jacobian with respect to the current iterate: $\mathbf{J} = J(\mathbf{w}^{(k)})$
Compute $\Delta\mathbf{y} = \mathbf{y} - F(\mathbf{w}^{(k)})$
Update: $\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} + (\mathbf{J}^\top\mathbf{J})^{-1}\mathbf{J}^\top\Delta\mathbf{y}$

Note that the solution will depend on the initial value $\mathbf{w}^{(0)}$ in general. There are several choices for measuring convergence. Some common choices include testing changes in the objective value:

$$\left|\frac{L^{(k+1)} - L^{(k)}}{L^{(k)}}\right| \leq \text{threshold}$$

or in the iterates themselves:

$$\max_j \left|\frac{\Delta w_j}{w_j^{(k)}}\right| \leq \text{threshold}$$