Optimization with Taylor Expansions

Gradient Descent, Newton's Method, XGBoost

SYS 6018 | Spring 2025

taylor-expansion.pdf

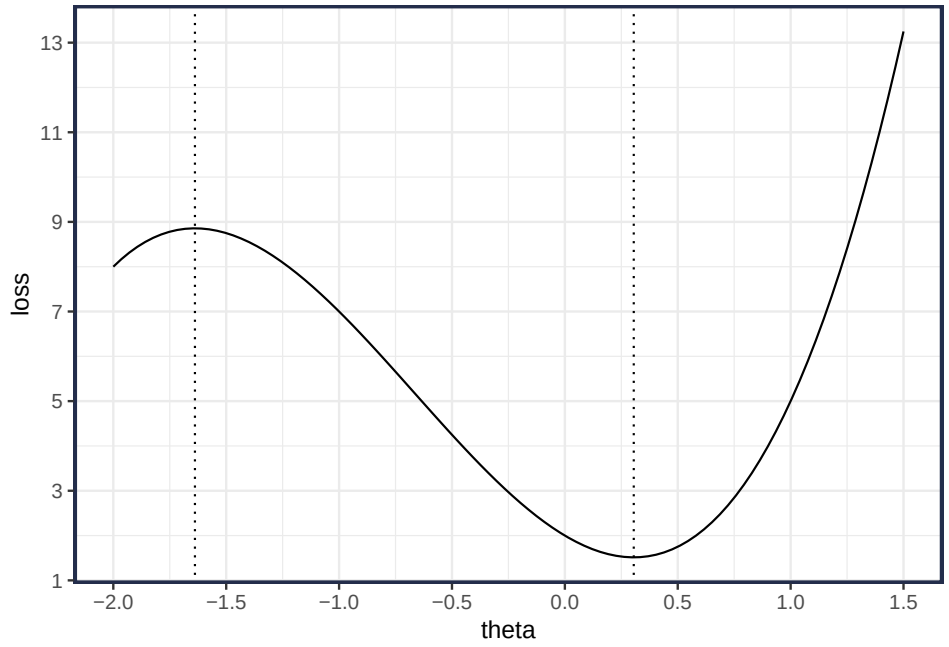
Contents

1 Optimization with Taylor

Consider the problem of finding the parameter that minimizes the function

$$l(\theta) = 2 - 3\theta + 4\theta^2 + 2\theta^3$$

in the range of $\theta \in [-2, 1.5]$.



There are several options including:

- Grid search
  a. Evaluate at a grid of θ values.
  b. Choose the one that minimizes the loss.
- Sequential Optimization / Design of Experiments / Response Surface / Surrogate Model / Bayesian Optimization
  a. Select initial evaluation points according to an experimental design.
  b. Fit a model to the loss (e.g., quadratic, Gaussian Process).
  c. Use the model to suggest a new evaluation points.
  d. Repeat until desired convergence.
- Gradient Descent
  – sequential optimization using 1st order approximations.
- Newton's Method
  – sequential optimization using 2nd order approximations.

We will specifically focus on the basics of the last two.

2 Taylor Series

A Taylor Series (or Taylor Expansion) of a function approximates the function around a given input value. For example, the n^{th} order Taylor approximation to our function l around a specific input θ

$$\begin{aligned} l(\theta+\epsilon) &\approx l(\theta) + \frac{l^{(1)}(\theta)}{1!}\epsilon + \frac{l^{(2)}(\theta)}{2!}\epsilon^2 + \ldots + \frac{l^{(n)}(\theta)}{n!}\epsilon^n \\ &= \sum_{j=0}^{n} \frac{l^{(j)}(\theta)}{j!}\epsilon^j \end{aligned}$$

Where $\epsilon \in \mathbb{R}$ is a step (direction and size) and $l^{(j)}(\theta) = d^j l(\theta)/(d\theta)^j$ is the jth derivative of the function evaluated at input θ.

For our particular function we have

$$\begin{aligned} l(\theta) &= 2 - 3\theta + 4\theta^2 + 2\theta^3 \\ l^{(1)}(\theta) &= g(\theta) = -3 + 8\theta + 6\theta^2 \\ l^{(2)}(\theta) &= h(\theta) = 8 + 12\theta \\ l^{(3)}(\theta) &= 12 \\ l^{(4)}(\theta) &= 0 \end{aligned}$$

2.1 Optimization with Taylor

Let's take a guess and start with $\theta_1 = 1$. We evaluate the function and find $L(\theta = 1) = 5$. Where should we try next?

Let's consider the Taylor approximations.

- The 1st order (linear) approximation is:

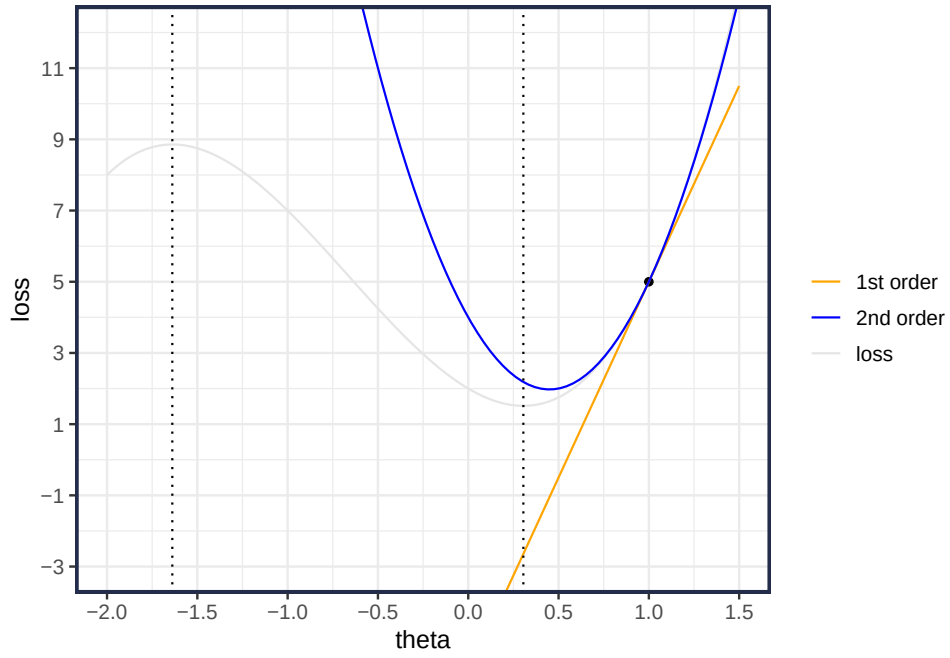
$$\begin{aligned} l(1+\epsilon) &\approx l(1) + g(1)\epsilon \\ &= 5 + 11\epsilon \end{aligned}$$

- The 2nd order (quadratic) approximation is:

$$\begin{aligned} l(1+\epsilon) &\approx l(1) + g(1)\epsilon + (h(1)/2)\epsilon^2 \\ &= 5 + 11\epsilon + 10\epsilon^2 \end{aligned}$$

Note

Notice that we evaluate the derivatives at $\theta = 1$, not at $1 + \epsilon$! These are constants, not a function of ϵ.



Which input should we try next?

2.1.1 Gradient Descent

Gradient descent is an optimization algorithm based on the 1st order (linear) approximation. The basic idea is to take a step in the direction of the negative gradient. That is, the next input to try is:

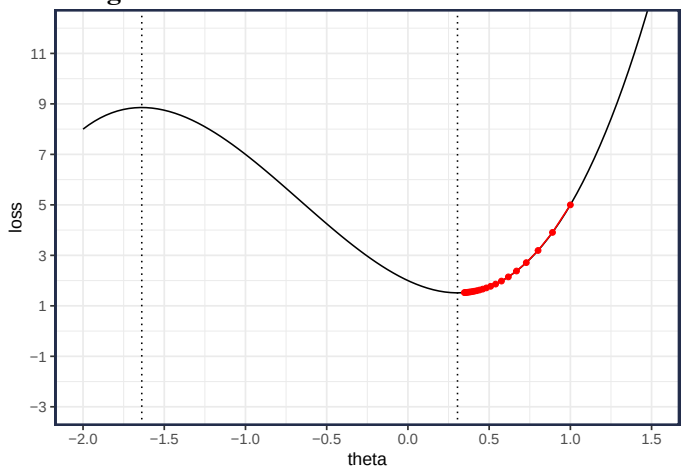
$$\theta_2 = \theta_1 - \nu g(\theta_1)$$

where $g(\theta_1) = l^{(1)}(\theta_1)$ is the *gradient* and ν is the *step size*. Recall our notation $\theta_2 = \theta_1 + \epsilon$ implies that gradient descent suggests the step $\epsilon^* = -\nu g(\theta_1)$.

The step size parameter ν is important. If it's too large, you can overshoot the minimum; if it's too small, it will take you too many iterations to converge. There are lots of approaches at optimizing the step size (most make step size smaller over the iterations), but these issues are beyond what we can cram into our class.

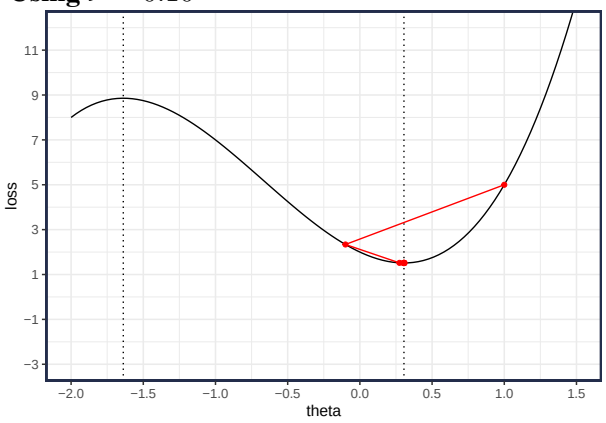
Gradient descent is an iterative approach; keep taking steps proportional to the negative gradient until the loss doesn't decrease very much (note danger of over-stepping) or the maximum number of iterations are reached.

2.1.1.1 Using $\nu = 0.01$



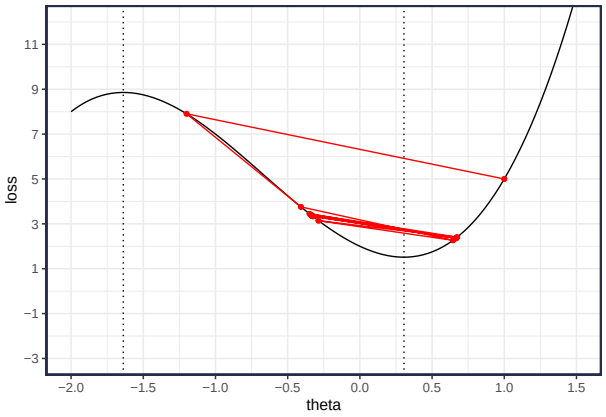
iter	nu	theta	gradient	epsilon	loss	delta
0	0.01	1.00	11.00	NA	5.00	NA
1	0.01	0.89	8.87	-0.11	3.91	1.09
2	0.01	0.80	7.26	-0.09	3.19	0.72
3	0.01	0.73	6.01	-0.07	2.71	0.48
4	0.01	0.67	5.03	-0.06	2.38	0.33
5	0.01	0.62	4.24	-0.05	2.15	0.23
6	0.01	0.58	3.60	-0.04	1.98	0.17
7	0.01	0.54	3.07	-0.04	1.86	0.12
8	0.01	0.51	2.63	-0.03	1.77	0.09
9	0.01	0.48	2.26	-0.03	1.71	0.06
10	0.01	0.46	1.95	-0.02	1.66	0.05
11	0.01	0.44	1.69	-0.02	1.63	0.04
12	0.01	0.42	1.47	-0.02	1.60	0.03
13	0.01	0.41	1.28	-0.01	1.58	0.02
14	0.01	0.40	1.11	-0.01	1.56	0.02
15	0.01	0.39	0.97	-0.01	1.55	0.01
16	0.01	0.38	0.85	-0.01	1.54	0.01
17	0.01	0.37	0.74	-0.01	1.54	0.01
18	0.01	0.36	0.65	-0.01	1.53	0.01
19	0.01	0.35	0.57	-0.01	1.53	0.00
20	0.01	0.35	0.50	-0.01	1.52	0.00

2.1.1.2 Using $\nu = 0.10$



iter	nu	theta	gradient	epsilon	loss	delta
0	0.1	1.00	11.00	NA	5.00	NA
1	0.1	-0.10	-3.74	-1.10	2.34	2.66
2	0.1	0.27	-0.36	0.37	1.52	0.82
3	0.1	0.31	0.05	0.04	1.51	0.01
4	0.1	0.30	-0.01	-0.01	1.51	0.00
5	0.1	0.31	0.00	0.00	1.51	0.00
6	0.1	0.31	0.00	0.00	1.51	0.00
7	0.1	0.31	0.00	0.00	1.51	0.00
8	0.1	0.31	0.00	0.00	1.51	0.00
9	0.1	0.31	0.00	0.00	1.51	0.00
10	0.1	0.31	0.00	0.00	1.51	0.00
11	0.1	0.31	0.00	0.00	1.51	0.00
12	0.1	0.31	0.00	0.00	1.51	0.00
13	0.1	0.31	0.00	0.00	1.51	0.00
14	0.1	0.31	0.00	0.00	1.51	0.00
15	0.1	0.31	0.00	0.00	1.51	0.00
16	0.1	0.31	0.00	0.00	1.51	0.00
17	0.1	0.31	0.00	0.00	1.51	0.00
18	0.1	0.31	0.00	0.00	1.51	0.00
19	0.1	0.31	0.00	0.00	1.51	0.00
20	0.1	0.31	0.00	0.00	1.51	0.00

2.1.1.3 Using $\nu = 0.20$



iter	nu	theta	gradient	epsilon	loss	delta
0	0.2	1.00	11.00	NA	5.00	NA
1	0.2	-1.20	-3.96	-2.20	7.90	-2.90
2	0.2	-0.41	-5.27	0.79	3.75	4.15
3	0.2	0.65	4.66	1.05	2.27	1.49
4	0.2	-0.29	-4.80	-0.93	3.14	-0.87
5	0.2	0.67	5.11	0.96	2.40	0.74
6	0.2	-0.35	-5.06	-1.02	3.45	-1.04
7	0.2	0.66	4.95	1.01	2.35	1.09
8	0.2	-0.33	-4.97	-0.99	3.33	-0.98
9	0.2	0.67	5.02	0.99	2.38	0.96
10	0.2	-0.34	-5.01	-1.00	3.39	-1.01
11	0.2	0.67	4.99	1.00	2.37	1.02
12	0.2	-0.33	-4.99	-1.00	3.36	-1.00
13	0.2	0.67	5.00	1.00	2.37	0.99
14	0.2	-0.33	-5.00	-1.00	3.37	-1.00
15	0.2	0.67	5.00	1.00	2.37	1.00
16	0.2	-0.33	-5.00	-1.00	3.37	-1.00
17	0.2	0.67	5.00	1.00	2.37	1.00
18	0.2	-0.33	-5.00	-1.00	3.37	-1.00
19	0.2	0.67	5.00	1.00	2.37	1.00
20	0.2	-0.33	-5.00	-1.00	3.37	-1.00

2.1.2 Newton's Method

Newton's method is an optimization algorithm based on the 2nd order (quadratic) approximation. Recall the Taylor's second order approximation of the loss function:

$$l(\theta + \epsilon) = l(\theta) + g(\theta)\epsilon + (h(\theta)/2)\epsilon^2$$

where $g(\theta)$ is the *gradient* and $h(\theta)$ is the *hessian*.

If we want to minimize the loss, we can write

$$\begin{aligned}\epsilon^* &= \arg\min_{\epsilon} \; l(\theta + \epsilon) \\ &\approx \arg\min_{\epsilon} \; l(\theta) + g(\theta)\epsilon + (h(\theta)/2)\epsilon^2\end{aligned}$$

Take the derivative with respect to ϵ, set to 0 and solve for ϵ:

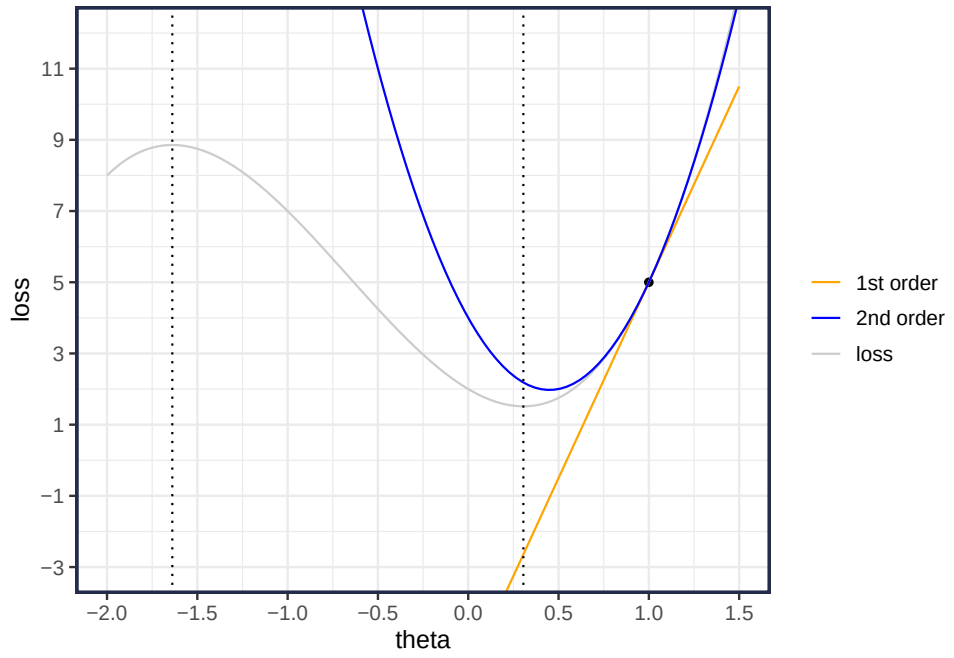
$$\begin{aligned}\frac{dl(\theta + \epsilon)}{d\epsilon} &= g(\theta) + h(\theta)\epsilon = 0 \\ \epsilon^* &= -\frac{g(\theta)}{h(\theta)}\end{aligned}$$

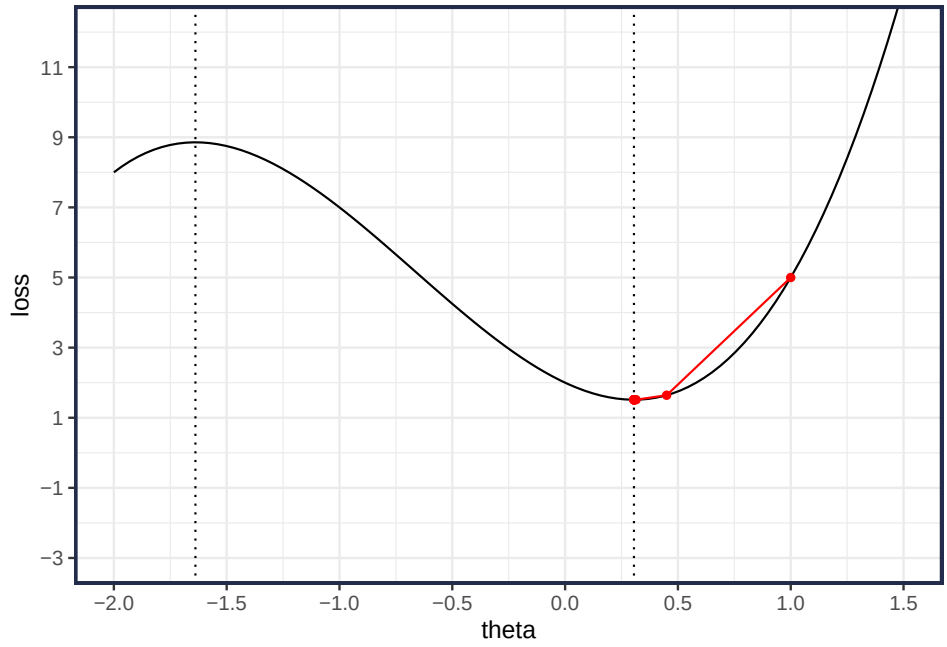
This suggests using

$$\begin{aligned}\theta_2 &= \theta_1 - \frac{g(\theta_1)}{h(\theta_1)} \\ &= \theta_1 - \nu g(\theta_1)\end{aligned}$$

where the step size $\nu = \frac{1}{h(\theta_1)}$ is given by one over the hessian.

Newton's method is also an iterative approach; keep iterating until the loss doesn't decrease very much (note danger of over-stepping) or the maximum number of iterations are reached. Note, it is still possible to overshoot, so adding shrinkage (i.e., force ν closer to 0) is sometimes done with Newton's method.





iter	nu	theta	gradient	hessian	epsilon	loss	delta
0	NA	1.00	11.00	20.00	NA	5.00	NA
1	0.05	0.45	1.81	13.40	-0.55	1.64	3.36
2	0.07	0.31	0.11	11.77	-0.14	1.51	0.13
3	0.08	0.31	0.00	11.66	-0.01	1.51	0.00
4	0.09	0.31	0.00	11.66	0.00	1.51	0.00
5	0.09	0.31	0.00	11.66	0.00	1.51	0.00
6	0.09	0.31	0.00	11.66	0.00	1.51	0.00
7	0.09	0.31	0.00	11.66	0.00	1.51	0.00
8	0.09	0.31	0.00	11.66	0.00	1.51	0.00
9	0.09	0.31	0.00	11.66	0.00	1.51	0.00
10	0.09	0.31	0.00	11.66	0.00	1.51	0.00

3 Newton Boosting and XGBoost

A boosting model can be written:

$$\hat{F}_M(x) = \sum_{m=1}^{M} \hat{a}_m \hat{f}_m(x)$$

where $\hat{a}_m$ is the weight assigned to model $\hat{f}_m(x)$. They are fit sequentially such that at iteration m the goal is to choose $\hat{f}_m(x)$ to minimize the loss:

$$\hat{f}_m(x) = \arg\min_f \sum_{i=1}^{n} l(y_i, \hat{F}_{m-1}(x_i) + f(x_i))$$

Let's re-write the loss, at iteration m, for observation i

$$l(F_i + f_i) = l(y_i, \hat{F}_{m-1}(x_i) + f(x_i))$$

and approximate it with a 2nd order (quadratic) Taylor expansion:

$$\begin{aligned} l(F_i + f_i) &\approx l(F_i) + g_i f(x_i) + \frac{1}{2} h_i f^2(x_i) \\ &= \frac{1}{2} h_i \left(-g_i/h_i - f(x_i)\right)^2 - \left[g_i^2/(2h_i) + l(F_i)\right] \end{aligned}$$

where $g_i = \frac{\partial l(F_i)}{\partial F_i}$ is the *gradient* and $h_i = \frac{\partial^2 l(F_i)}{\partial F_i^2}$ is the *hessian*.

Gradient and Hessian for common loss functions

- **Squared Error Loss**

$$\begin{aligned} \ell(y_i, F_i) &= \frac{1}{2}(y_i - F_i)^2 \\ g_i &= -(y_i - F_i) \\ h_i &= 1 \end{aligned}$$

- **Log Loss (negative Bernoulli log-likelihood, cross-entropy)**

$$\begin{aligned} y_i &\in \{0, 1\} \\ p_i &= \frac{e^{F_i}}{1 + e^{F_i}} \\ \ell(y_i, F_i) &= -\left(y_i \log p_i + (1 - y_i) \log(1 - p_i)\right) \\ &= -\left(y_i F_i - \log(1 + e^{F_i})\right) \\ g_i &= -(y_i - p_i) \\ h_i &= p_i(1 - p_i) \end{aligned}$$

Using this approximation, we can consider our optimization problem at iteration m as finding the model such

that

$$\begin{aligned}
\hat{f}_m(x) &= \arg\min_f \sum_{i=1}^n l(y_i, \hat{F}_{m-1}(x_i) + f(x_i)) \\
&\approx \arg\min_f \sum_{i=1}^n \left(g_i f(x_i) + \frac{1}{2} h_i f^2(x_i) \right) \\
&= \arg\min_f \sum_{i=1}^n \frac{h_i}{2} \left(-g_i/h_i - f(x_i) \right)^2 \\
&= \arg\min_f \sum_{i=1}^n w_i \left(z_i - f(x_i) \right)^2
\end{aligned}$$

where the last line shows that this is equivalent to *weighted least squares* with observation weights $w_i = h_i/2$ and outcome variable $z_i = -g_i/h_i$.

XGBoost uses this approximate loss to build its trees in a straightforward weighted least squares manner.