Bonus Lecture: Nonlinear Optimization

Chris Conlon

Thursday 25th September, 2025

Grad IO

Basic Setup

Often we are interested in solving a problem like this:

Root Finding $f(x) = 0$

Optimization $\arg\min_x f(x)$.

These problems are related because we find the minimum by setting: $f'(x) = 0$

Nonlinear Optimization

Newton's Method for Root Finding

Consider the Taylor series for $f(x)$ approximated around $f(x_0)$:

$$f(x) \approx f(x_0) + f'(x_0) \cdot (x - x_0) + f''(x_0) \cdot (x - x_0)^2 + o_p(3)$$

Suppose we wanted to find a root of the equation where $f(x^*) = 0$ and solve for x:

$$0 = f(x_0) + f'(x_0) \cdot (x - x_0)$$

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$$

This gives us an iterative scheme to find x^*:

1. Start with some x_k. Calculate $f(x_k), f'(x_k)$
2. Update using $x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$
3. Stop when $|x_{k+1} - x_k| < \epsilon_{tol}$.

Newton-Raphson for Minimization

We can re-write optimization as root finding;

- We want to know $\hat{\theta} = \arg\max_\theta \ell(\theta)$.
- Construct the FOCs $\frac{\partial \ell}{\partial \theta} = 0 \rightarrow$ and find the zeros.
- How? using Newton's method! Set $f(\theta) = \frac{\partial \ell}{\partial \theta}$

$$\theta_{k+1} = \theta_k - \left[\frac{\partial^2 \ell}{\partial \theta^2}(\theta_k)\right]^{-1} \cdot \frac{\partial \ell}{\partial \theta}(\theta_k)$$

The SOC is that $\frac{\partial^2 \ell}{\partial \theta^2} > 0$. Ideally at all θ_k.
This is all for a single variable but the multivariate version is basically the same.

Newton's Method: Multivariate

Start with the objective $Q(\theta) = -l(\theta)$:

- Approximate $Q(\theta)$ around some initial guess θ_0 with a quadratic function
- Minimize the quadratic function (because that is easy) call that θ_1
- Update the approximation and repeat.

$$\theta_{k+1} = \theta_k - \left[\frac{\partial^2 Q}{\partial \theta \partial \theta'}\right]^{-1} \frac{\partial Q}{\partial \theta}(\theta_k)$$

- The equivalent SOC is that the Hessian Matrix is positive semi-definite (ideally at all θ).
- In that case the problem is globally convex and has a unique maximum that is easy to find.

Newton's Method

We can generalize to Quasi-Newton methods:

$$\theta_{k+1} = \theta_k - \lambda_k \underbrace{\left[\frac{\partial^2 Q}{\partial\theta\partial\theta'}\right]^{-1}}_{A_k} \frac{\partial Q}{\partial\theta}(\theta_k)$$

Two Choices:

- Step length λ_k
- Step direction $d_k = A_k \frac{\partial Q}{\partial\theta}(\theta_k)$
- Often rescale the direction to be unit length $\frac{d_k}{\|d_k\|}$.
- If we use A_k as the true Hessian and $\lambda_k = 1$ this is a full Newton step.

Newton's Method: Alternatives

Choices for A_k

- $A_k = I_k$ (Identity) is known as gradient descent or steepest descent
- BHHH. Specific to MLE. Exploits the Fisher Information.

$$A_k = \left[\frac{1}{N} \sum_{i=1}^{N} \frac{\partial \ln f}{\partial \theta} (\theta_k) \frac{\partial \ln f}{\partial \theta'} (\theta_k) \right]^{-1}$$
$$= -\mathbb{E} \left[\frac{\partial^2 \ln f}{\partial \theta \partial \theta'} (Z, \theta^*) \right] = \mathbb{E} \left[\frac{\partial \ln f}{\partial \theta} (Z, \theta^*) \frac{\partial \ln f}{\partial \theta'} (Z, \theta^*) \right]$$

- Alternatives SR1 and DFP rely on an initial estimate of the Hessian matrix and then approximate an update to A_k.
- Usually updating the Hessian is the costly step.
- Non invertible Hessians are bad news.
- Optimizers Differ in: (1) Hessian update; (2) Step-size calculation; (3) Handling of Constraints (which I ignored!)

What about ML Models?

How do people fit ML models?

- Inverting the Hessian for models with large numbers of parameters becomes infeasible.
- Many ML models (not all) can be highly non-convex meaning that Hessian may be not invertible, and there may be many potential local minima.
- Most models do some form of gradient descent (setting Hessian $A_k = \mathbb{I}$)
- Many models use early stopping, terminating at a point before we reach the minimum (!)

$$\theta_{k+1} = \theta_k - \lambda_k \cdot \frac{\partial Q}{\partial \theta}(\theta_k)$$

With key modificiations:

- Set the learning rate / step size to a small value $\lambda_k < .001$.
- Use batched gradients to approximate the gradient on a random subset of observations.

$$g_k = \frac{1}{|\mathcal{B}_t|} \sum_{i \in \mathcal{B}_k} \nabla_\theta \ell_i \left(\theta_t\right).$$

This typically means many more iterations than quasi-Newton but often cheaper ones.

AdamW: Objective and Moment Estimates

We aim to minimize the loss function $\ell(\theta), \quad \theta \in \mathbb{R}^d$.

Gradient and moment estimates (using a batched gradient):

$$g_k = \nabla_\theta \ell(\theta_k), \tag{1}$$

$$m_k = \beta_1 m_{k-1} + (1-\beta_1) g_k \qquad \text{(momentum)}, \tag{2}$$

$$v_k = \beta_2 v_{k-1} + (1-\beta_2) g_k^2 \qquad \text{(adaptive scaling)}. \tag{3}$$

Bias correction:

$$\hat{m}_k = \frac{m_k}{1-\beta_1^k}, \tag{4}$$

$$\hat{v}_k = \frac{v_k}{1-\beta_2^k}. \tag{5}$$

AdamW: Update Rule and Properties

Parameter update:

$$\theta_{k+1} = \theta_k - \eta \frac{\hat{k}_k}{\sqrt{\hat{v}_k} + \epsilon} - \eta\lambda\theta_k. \tag{6}$$

Key features:

- **Momentum:** $\hat{m}_t$ accumulates past gradients, accelerating learning.
- **Parameter-specific learning rates:** $\hat{v}_t$ rescales each coordinate of θ.
  - This captures how g_t, g_t^2 changes from the moving average m_{k-1}, v_{k-1} which gives us some estimate of curvature that is parameter specific without trying to approximate A_k
- **Decoupled weight decay:** $-\eta\lambda\theta_k$ is applied separately from gradient update.

Applications of AdamW

`AdamW` is now the standard optimizer in many deep learning frameworks (`PyTorch`, `TensorFlow`) and implementations exist for others like Google's `Jax`. It is often used on very large models

- **Deep learning:** training deep neural networks with millions or billions of parameters.
- **Transformer models:** BERT, GPT, and other large language models often use AdamW by default.

But it has some disadvantages

- Even on simple problems it may take tens of thousands of iterations. (Quasi-Newton approaches might take a dozen or fewer on convex MLE problems).
- The idea is that with Automatic Differentiation and batched gradients those iterations might be much cheaper.