Math 6880/7875: Advanced Optimization
Examples, Part 1

Akil Narayan[1]

[1]Department of Mathematics, and Scientific Computing and Imaging (SCI) Institute
University of Utah

January 27, 2022

Examples in optimization

We'll take a short tour of some examples in optimization:

- Machine learning
  - Model training to deep learning: Training of model parameters
  - Classification: Building of classification models
- ~~Statisics~~ Statistics
  - Bayesian inference: Updating beliefs with data
  - Gaussian Processes: Hyperparameter optimization
- PDE-constrained optimization: Optimization with PDE constraints
- Sparse approximation: Compressed sensing and matrix completion
- Stochastic programming: Optimization under uncertainty

Training models

Training models is one of the simplest examples of optimization.

Let $\mathcal{X}$ be an input space, and $\mathcal{Y}$ an output space.
Given data $\mathcal{D}$ from an unknown function $f : \mathcal{X} \to \mathcal{Y}$,

$$\mathcal{D} = \{(x_j, f(x_j))\}_{j=1}^{M},$$

the wish to build a model that approximates f.

Linear models define a model space spanned by fixed functions,

$$f_N(x) = \sum_{j=1}^{N} c_j v_j(x), \qquad\qquad x \in \mathcal{X},$$

where the model is trained by computing the c_j coefficients,

$$\{c_1, \ldots, c_N\} = \underset{c_1, \ldots, c_n}{\arg\min}\, \mathcal{L}(c_1, \ldots, c_n; \mathcal{D}).$$

The function $\mathcal{L}$ is sometimes called a loss function.

Training models

Training models is one of the simplest examples of optimization.

Let $\mathcal{X}$ be an input space, and $\mathcal{Y}$ an output space.
Given data $\mathcal{D}$ from an unknown function $f : \mathcal{X} \to \mathcal{Y}$,

$$\mathcal{D} = \{(x_j, f(x_j))\}_{j=1}^M ,$$

the wish to build a model that approximates f.

Linear models define a model space spanned by fixed functions,

$$f_N(x) = \sum_{j=1}^N c_j v_j(x), \qquad x \in \mathcal{X},$$

where the model is trained by computing the c_j coefficients,

$$\{c_1, \ldots, c_N\} = \underset{c_1,\ldots,c_n}{\arg\min}\, \mathcal{L}(c_1, \ldots, c_n; \mathcal{D}).$$

The function $\mathcal{L}$ is sometimes called a loss function. $\longrightarrow$ measure of data discrepancy + model complexity.

Examples of training

Such a paradigm is used everywhere, sometimes just as one ingredient:

- data/curve fitting
- statistical models
- clustering (supervised or unsupervised)
- simulations of differential equations
- deep learning

Linear models

$$\mathcal{D} = \{(x_j, f(x_j))\}_{j=1}^M, \qquad f_N(x) = \sum_{j=1}^N c_j v_j(x).$$

$$\{c_1, \ldots, c_N\} = \underset{c_1,\ldots,c_n}{\arg\min}\, \mathcal{L}(c_1, \ldots, c_n; \mathcal{D}).$$

Note: "linear approximation" $\neq$ "linear model"

A frequently used loss function is the quadratic loss, or mean square error:

$$\mathcal{L}(c; \mathcal{D}) = \sum_{j=1}^M \left(f(x_j) - f_N(x_j)\right)^2.$$

This is a quadratic function of the coefficients c_j with positive semi-definite Hessian.

There is a unique solution if and only if the design matrix,

$$A \in \mathbb{R}^{M \times N}, \qquad (A)_{i,j} = v_j(x_i),$$

has full column rank.

No particularly special optimization is needed here, just some linear algebra.

Linear models

$$\mathcal{D} = \{(x_j, f(x_j))\}_{j=1}^{M}, \qquad f_N(x) = \sum_{j=1}^{N} c_j v_j(x).$$

$$\{c_1, \ldots, c_N\} = \operatorname*{arg\,min}_{c_1,\ldots,c_n} \mathcal{L}(c_1, \ldots, c_n; \mathcal{D}).$$

Note: "linear approximation" $\neq$ "linear model"

A frequently used loss function is the quadratic loss, or mean square error:

$$\mathcal{L}(c; \mathcal{D}) = \sum_{j=1}^{M} \left(f(x_j) - f_N(x_j)\right)^2.$$

This is a quadratic function of the coefficients c_j with positive semi-definite Hessian. $= A^T A$

There is a unique solution if and only if the design matrix,

$$A \in \mathbb{R}^{M \times N}, \qquad (A)_{i,j} = v_j(x_i),$$

has full column rank.

No particularly special optimization is needed here, just some linear algebra.

Linear models

$$\mathcal{D} = \{(x_j, f(x_j))\}_{j=1}^{M}, \qquad f_N(x) = \sum_{j=1}^{N} c_j v_j(x).$$

$$\{c_1, \ldots, c_N\} = \underset{c_1,\ldots,c_n}{\arg\min}\, \mathcal{L}(c_1, \ldots, c_n; \mathcal{D}).$$

Note: "linear approximation" $\neq$ "linear model"

A frequently used loss function is the quadratic loss, or mean square error:

$$\mathcal{L}(c; \mathcal{D}) = \sum_{j=1}^{M} \left(f(x_j) - f_N(x_j)\right)^2.$$

This is a quadratic function of the coefficients c_j with positive semi-definite Hessian.

There is a unique solution if and only if the design matrix,

$$A \in \mathbb{R}^{M \times N}, \qquad (A)_{i,j} = v_j(x_i),$$

has full column rank.

No particularly special optimization is needed here, just some linear algebra.

Constrained linear models

Constraints on model behavior are often feasible to implement for linear models.

For example, enforcing defines a feasible set:

$$S = \{c_1, \ldots, c_N \mid f_N(x) \geqslant 0 \text{ for all } x \in \mathcal{X}\}$$

Nontrivial constraints are still "nice": S is convex.

$\min_{c} \mathcal{L}(c; D)$ s.t. $c \in S$

Explicit solution: ????

Deep learning models

In principle, any kind of parametric model can be used, e.g., a neural network.

A neural network with L layers ($L-2$ are "hidden") is given by,

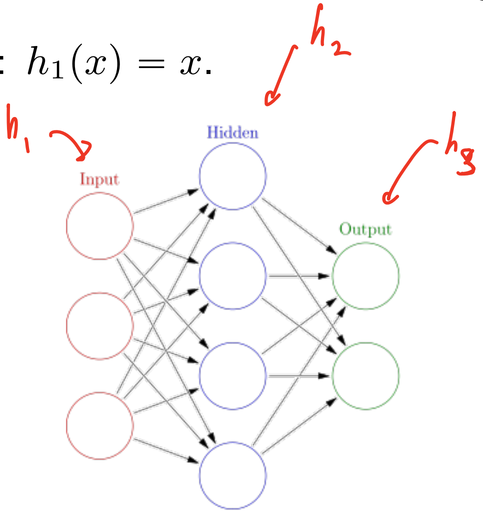
$$f_N(x) = h_L(x), \qquad\qquad h_\ell = \phi\left(W_\ell h_{\ell-1} + b_\ell\right),$$

for some weight matrices and bias vectors $\theta = \{(W_\ell, b_\ell)\}_{\ell=2}^{L}$, with $\theta \in \mathbb{R}^N$.

ϕ is an "activation function", common choices emulating on/off switches, such as

$$\phi_{\text{RELU}}(x) = x\mathbb{1}_{x\geqslant 0}, \qquad\qquad \phi_{sig} = \frac{e^x}{e^x+1}.$$

The initial layer is the input: $h_1(x) = x$.



($L \geq 3$ is the interesting case)

Image: Glosser.ca, CC BY-SA 3.0 <https://creativecommons.org/licenses/by-sa/3.0>, via Wikimedia Commons

Training deep learning models

Training from data, even with a quadratic loss, is less pleasant than before.

$$\theta = \arg\min_{\theta} \mathcal{L}(\theta; \mathcal{D}) = \arg\min_{\theta} \sum_{j=1}^{M} \left(f(x_j) - f_N(x_j)\right)^2 .$$

This is complicated since $\theta \mapsto f_N$ is not linear in θ anymore.

- The optimization is non-convex
- Analytically computing stationary points is unrealistic
- How much data is required to achieve good training is unclear
- Little theory or intuition about feasible sets for constraints

Training deep learning models

Training from data, even with a quadratic loss, is less pleasant than before.

$$\theta = \arg\min_{\theta} \mathcal{L}(\theta; \mathcal{D}) = \arg\min_{\theta} \sum_{j=1}^{M} \left(f(x_j) - f_N(x_j)\right)^2 .$$

This is complicated since $\theta \mapsto f_N$ is not linear in θ anymore.

- The optimization is non-convex
- Analytically computing stationary points is unrealistic
- How much data is required to achieve good training is unclear
- Little theory or intuition about feasible sets for constraints

Classification

Some (naive) supervised classification models just perform model fitting:

$$\{(x_j, c_j)\}_{j=1}^M,$$

where x_j are features and c_j are discrete classification labels.

One strategy quantizes classification labels as points on the unit simplex:

$$\text{Labels } \ell_1, \ldots, \ell_J \rightarrow \begin{array}{rcl} e_1 = (1,0,0,\ldots) & \leftrightarrow & \ell_1 \\ e_2 = (0,1,0,\ldots) & \leftrightarrow & \ell_2 \\ e_3 = (0,0,1,\ldots) & \leftrightarrow & \ell_3 \\ \vdots & & \end{array}$$

Training proceeds by emulating the model with labels c_j replaced by appropriate vector e_k.

- Any general model may be used for prediction
- Predictions often lie in Δ^{J+1}, giving quantitative information about label likelihoods

Classification

Some (naive) supervised classification models just perform model fitting:

$$\{(x_j, c_j)\}_{j=1}^{M},$$

where x_j are features and c_j are discrete classification labels.

One strategy quantizes classification labels as points on the unit simplex:

$$\text{Labels } \ell_1, \ldots, \ell_J \rightarrow \begin{array}{lcl} e_1 = (1, 0, 0, \ldots) & \leftrightarrow & \ell_1 \\ e_2 = (0, 1, 0, \ldots) & \leftrightarrow & \ell_2 \\ e_3 = (0, 0, 1, \ldots) & \leftrightarrow & \ell_3 \\ \vdots & & \end{array}$$

Training proceeds by emulating the model with labels c_j replaced by appropriate vector e_k.

- Any general model may be used for prediction
- Predictions often lie in Δ^{J+1}, giving quantitative information about label likelihoods

$$\{(x_j, c_j)\} \longrightarrow \{(x_j, e_{k(j)})\}_{j=1}^{M}$$

Statistical inference

Many statistical inference problems are cast in the Bayesian paradigm.

- *frequentist paradigm*: Probabilities encode actual randomness. If an experiment were repeated several times, the outcomes would reflect modeled probabilities.
- *Bayesian paradigm*: Probabilities encode *belief*, not necessarily randomness. There might be an underlying deterministic reality, but we model our uncertainty about it as randomness.

Inference is the task of learning about unobservables via data (observables).

Statistical inference makes quantitative predictions about probabilities from data.

$\wedge$
Bayesian

Bayesian inference

Data should inform beliefs

$$P\left(\text{unobservable} \,\middle|\, \text{data}\right) = \frac{P\left(\text{data} \,\middle|\, \text{unobservable}\right) P\left(\text{unobservable}\right)}{P(\text{data})}$$

- $P(\text{unobservable})$: the *prior* distribution, encodes belief about unobservable without data
- $P(\text{data} \mid \text{unobservable})$: the *likelihood*, knowledge about how unobservable values would affect observed data
- $P(\text{data})$: the *evidence*, normalization of the prior and likelihood
- $P\left(\text{unobservable} \,\middle|\, \text{data}\right)$: the *posterior* distribution, the object of interest

Bayes' Rule: $P(A \cap B) = P(A|B)\,P(B)$

$= P(B|A)\,P(A)$

Bayesian inference

Data should inform beliefs

$$P\left(\text{unobservable} \,\middle|\, \text{data}\right) = \frac{P\left(\text{data} \,\middle|\, \text{unobservable}\right) P\left(\text{unobservable}\right)}{P(\text{data})}$$

- $P(\text{unobservable})$: the *prior* distribution, encodes belief about unobservable without data
- $P(\text{data} \,|\, \text{unobservable})$: the *likelihood*, knowledge about how unobservable values would affect observed data
- $P(\text{data})$: the *evidence*, normalization of the prior and likelihood
- $P\left(\text{unobservable} \,\middle|\, \text{data}\right)$: the *posterior* distribution, the object of interest

Posterior distribution analysis

$$P\left(\overset{\text{typically high-dimensional}}{\overset{\downarrow}{\text{unobservable}}} \,\middle|\, \text{data}\right) = \frac{P\left(\text{data} \,\middle|\, \text{unobservable}\right) P\left(\text{unobservable}\right)}{P(\text{data})}$$

Types of analysis using the posterior:

- computing statistics
- generating samples
- computing mode of the distribution – the *maximum a posteriori* (MAP) estimate

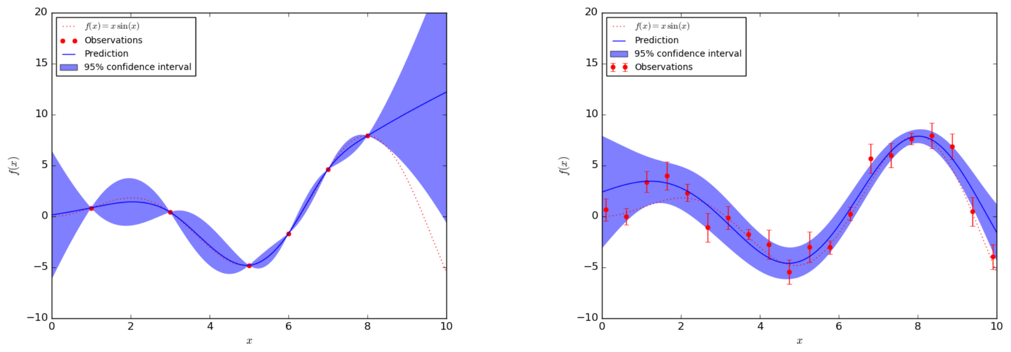
Computing the MAP is explicitly an optimization problem:

$$\text{MAP} = \arg\max_{u} P(d|u)\ P(u).$$

(Sometimes maximizing just the likelihood is desired.)

Example: Gaussian Processes

Gaussian processes (GPs) are *stochastic* models that predict data.



Images: scikit-learn documentation

The ingredients:

- A covariance/correlation function $K(x, x'; \theta)$
- "Hyperparmeters" θ that affect the type of covariance function (e.g., correlation length)

A GP is trained from data polluted with a variance-σ^2 Gaussian noise,

$$c = (A + \sigma I)^{-1} f, \qquad (A)_{i,j} = K(x_i, x_j; \theta)$$

This defines the mean:

$$f_N(x; \theta) = \sum_{i=1}^{N} c_i K(x, x_i; \theta).$$

(a linear model!)

Hyperparameter training

The hyperparameters θ can significantly affect the quality of the model.

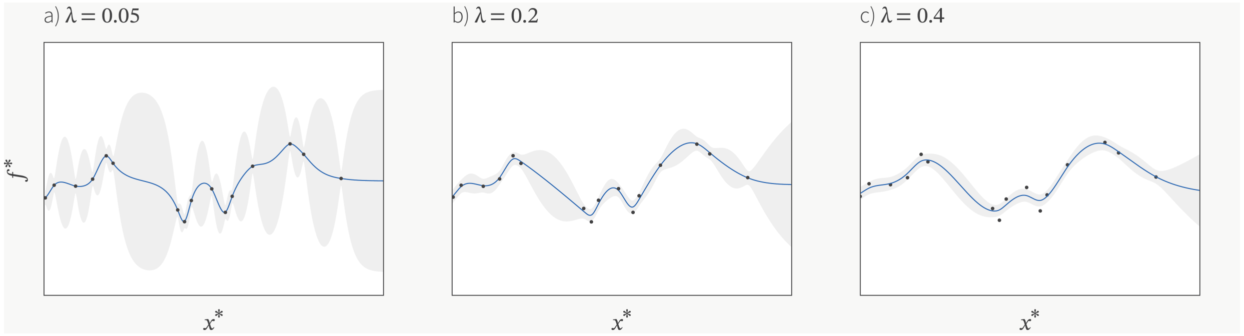


Image: `https://www.borealisai.com/en/blog/tutorial-8-bayesian-optimization`

One step in training GP's is hyperparameter optimization:
The hyperparameters θ are chosen to maximize the (log-)likelihood:

$$\theta = \arg\max_{\theta} -\frac{1}{2} f^T A(\theta)^{-1} f - \frac{1}{2} \log\det A(\theta).$$

Hyperparameter training

The hyperparameters θ can significantly affect the quality of the model.

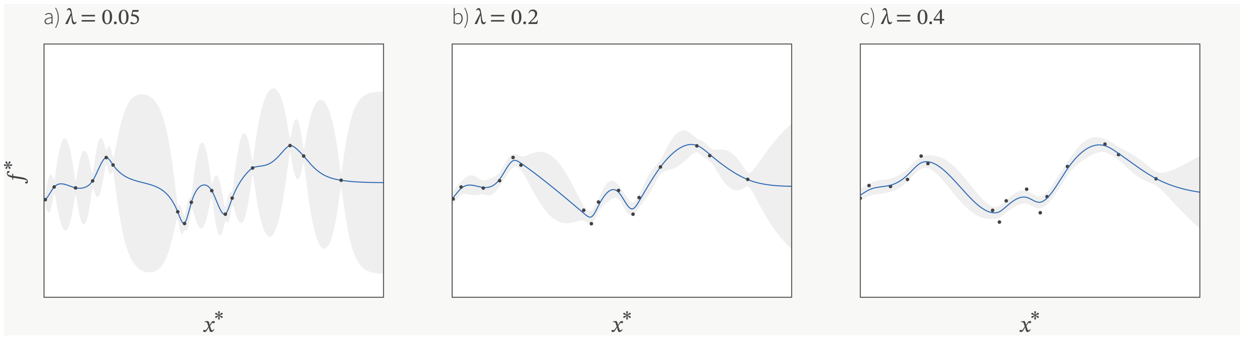


Image: `https://www.borealisai.com/en/blog/tutorial-8-bayesian-optimization`

One step in training GP's is hyperparameter optimization:
The hyperparameters θ are chosen to maximize the (log-)likelihood:

$$\theta = \arg\max_{\theta} -\frac{1}{2} f^T A(\theta)^{-1} f - \frac{1}{2} \log\det A(\theta).$$

Ex: $K(x,y;\theta) = \exp\left(\frac{-\|x-y\|^2}{\theta}\right)$

References I

- Ian Goodfellow, Yoshua Bengio, and Aaron Courville, *Deep Learning*, MIT Press, 2016.
- Trevor Hastie, Robert Tibshirani, and Jerome Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, Springer Science & Business Media, 2013.
- Carl Edward Rasmussen, Christopher K. I. Williams, and Francis Bach, *Gaussian Processes for Machine Learning*, MIT Press, 2006 (en).