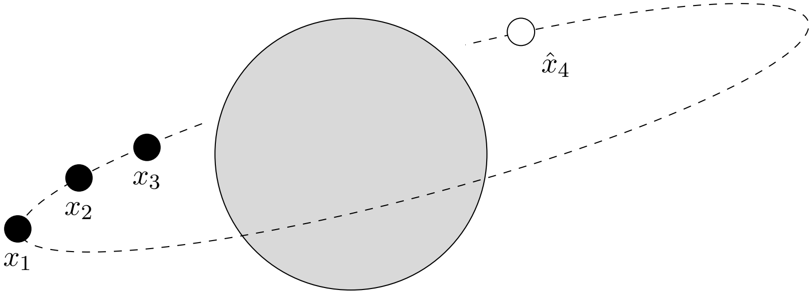
Differentiable Optimisation in Deep Learning

Stephen Gould
stephen.gould@anu.edu.au

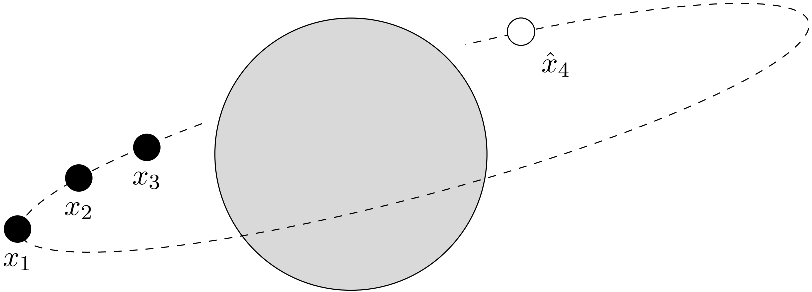
Australian National University

15 December, 2022

Discovery of Ceres



Discovery of Ceres



Optimisation is Everywhere

- **financial mathematics:** maximise profits or minimise costs subject to constraints on resources and budgets

Optimisation is Everywhere

- **financial mathematics:** maximise profits or minimise costs subject to constraints on resources and budgets
- **mechanical engineering:** maximise the span of a bridge subject to load constraints

Optimisation is Everywhere

- **financial mathematics:** maximise profits or minimise costs subject to constraints on resources and budgets
- **mechanical engineering:** maximise the span of a bridge subject to load constraints
- **electrical engineering:** minimise the size of a transistor in a circuit subject to power and timing constraints

Optimisation is Everywhere

- **financial mathematics:** maximise profits or minimise costs subject to constraints on resources and budgets
- **mechanical engineering:** maximise the span of a bridge subject to load constraints
- **electrical engineering:** minimise the size of a transistor in a circuit subject to power and timing constraints
- **logistics and planning:** find the cheapest way to distribute goods from suppliers to consumers across a network

Optimisation is Everywhere

- **financial mathematics:** maximise profits or minimise costs subject to constraints on resources and budgets
- **mechanical engineering:** maximise the span of a bridge subject to load constraints
- **electrical engineering:** minimise the size of a transistor in a circuit subject to power and timing constraints
- **logistics and planning:** find the cheapest way to distribute goods from suppliers to consumers across a network
- **statistics/data science:** curve fitting and data visualisation

Optimisation is Everywhere

- **financial mathematics:** maximise profits or minimise costs subject to constraints on resources and budgets
- **mechanical engineering:** maximise the span of a bridge subject to load constraints
- **electrical engineering:** minimise the size of a transistor in a circuit subject to power and timing constraints
- **logistics and planning:** find the cheapest way to distribute goods from suppliers to consumers across a network
- **statistics/data science:** curve fitting and data visualisation
- **machine learning and deep learning:** minimise loss functions with respect to the parameters of our model

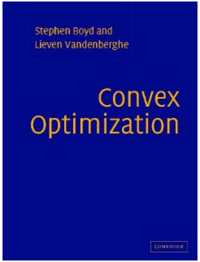
Overview

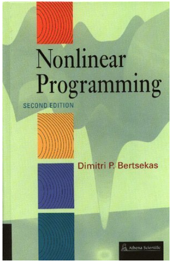
- **Introduction to Optimisation**
  - Formal definition
  - Least squares
  - Convex sets and functions
  - Convex optimisation problems
  - Lagrangian
  - Optimality conditions
  - Algorithms
- **Differentiable Optimisation and Deep Learning**
  - Machine learning from 10,000ft
  - Automatic differentiation
  - Forward and backward passes
  - Imperative and declarative nodes
  - Bi-level optimisation
  - Implicit function theorem
  - Differentiable optimisation results
- **Examples and Applications**
  - Least squares
  - Optimal transport
  - Blind perspective-n-point

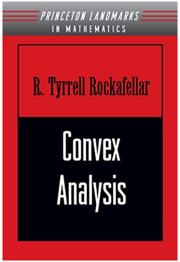
accompanying lecture notes available at
`https://users.cecs.anu.edu.au/~sgould`

lecture 1

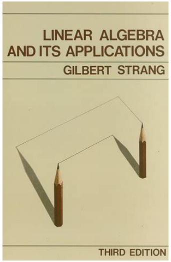
Lecture 1: Introduction to Optimisation

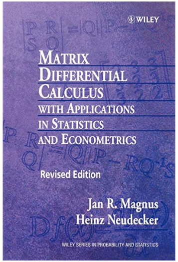


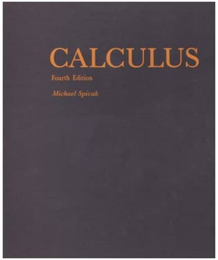




Assumed Background







Optimisation Problems

find the assignment to variables that minimises a measure of cost subject to some constraints[1]

[1] In these lectures we will be concerned with continuous-valued variables

Optimisation Problems

minimize (over x)	objective(x)
subject to	constraints(x)

Optimisation Problems

$$\begin{array}{ll} \text{minimize} & f_0(x) \\ \text{subject to} & f_i(x) \leq 0, \quad i = 1, \ldots, p \\ & h_i(x) = 0, \quad i = 1, \ldots, q \end{array}$$

- $x = (x_1, \ldots, x_n) \in \mathbb{R}^n$ — optimisation variables
- $f_0 : \mathbb{R}^n \to \mathbb{R}$ — objective (or cost or loss) function
- $f_i : \mathbb{R}^n \to \mathbb{R}$, $i = 1, \ldots, p$ — inequality constraint functions
- $h_i : \mathbb{R}^n \to \mathbb{R}$, $i = 1, \ldots, q$ — equality constraint functions

Solution and Optimal Value

A point x is **feasible** if $x \in \mathbf{dom}\,(f_0)$ and it satisfies the constraints.

A **solution**, or optimal point, $x^\star$ has the smallest value of f_0 among all feasible x.

[1] Warning: notation clash between p and $p^\star$!

Solution and Optimal Value

A point x is **feasible** if $x \in \mathbf{dom}\,(f_0)$ and it satisfies the constraints.

A **solution**, or optimal point, $x^\star$ has the smallest value of f_0 among all feasible x.

The **optimal value** is[1]

$$p^\star = \inf_{x \in \mathcal{D}} \left\{ f_0(x) \;\middle|\; \begin{array}{ll} f_i(x) \le 0, & i = 1, \dots, p \\ h_i(x) = 0, & i = 1, \dots, q \end{array} \right\}.$$

- $p^\star$ and is equal to $f_0(x^\star)$ when $x^\star$ exists
- $p^\star = \infty$ if the problem is infeasible (no x satisfies the constraints)
- $p^\star = -\infty$ if the problem is unbounded below

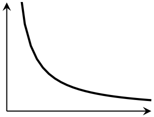
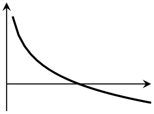
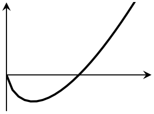
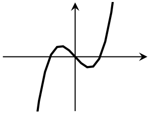
[1] Warning: notation clash between p and $p^\star$!

Locally Optimal Points

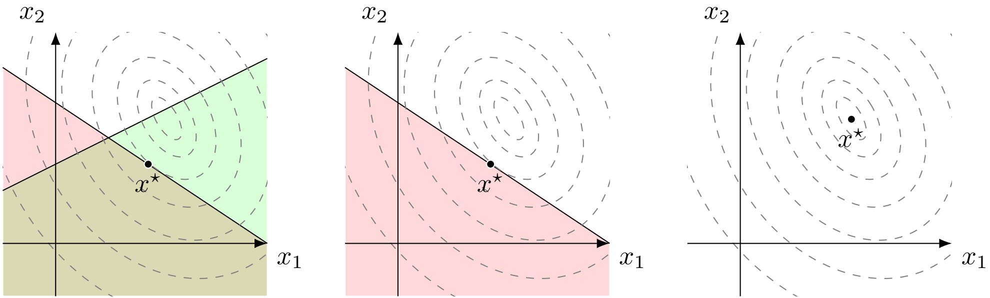
A point x is **locally optimal** if there is an $R > 0$ such that $z = x$ is optimal for

$$\begin{array}{lll} \text{minimize (over } z) & f_0(z) & \\ \text{subject to} & f_i(z) \leq 0 & i = 1, \ldots, p \\ & h_i(z) = 0 & i = 1, \ldots, q \\ & \|z - x\|_2 \leq R. & \end{array}$$

Examples (1D)

	$1/x$	$-\log x$	$x \log x$	$x^3 - 3x$
f_0:				
dom (f_0):	$\mathbb{R}_{++}$	$\mathbb{R}_{++}$	$\mathbb{R}_{++}$	$\mathbb{R}$
$p^\star$:	0	$-\infty$	$-1/e$	$-\infty$
$x^\star$:	none	none	$1/e$	$x = 1$ locally

Examples (2D)



Least Squares

$$\text{minimize} \quad \|Ax - b\|_2^2$$

Least Squares

$$\text{minimize} \quad \|Ax - b\|_2^2$$

- unique solution if A^TA is invertible, $x^\star = \left(A^TA\right)^{-1}A^Tb$
- solution via SVD, $A = U\Sigma V^T$, if A^TA not invertible, $x^\star = V\Sigma^{-1}U^Tb$
  - in fact, $x^\star + w$ for any $w \in \mathcal{N}(A)$ also a solution
- solution via QR factorisation, $x^\star = R^{-1}Q^Tb$
- solved in $O(n^2m)$ time, less if structured
- typically use iterative solver

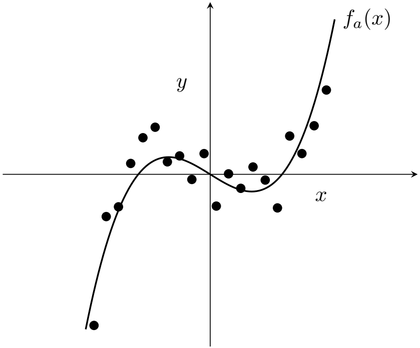
Example: Polynomial Curve Fitting

fit n-th order polynomial $f_a(x) = \sum_{k=0}^{n} a_k x^k$ to set of noisy points $\{(x_i, y_i)\}_{i=1}^{m}$

$$\text{minimize (over } a\text{)} \quad \sum_{i=1}^{m} \left(f_a(x_i) - y_i\right)^2$$

$$\text{minimize} \quad \left\| \begin{bmatrix} 1 & x_1 & x_1^2 & \dots & x_1^n \\ 1 & x_2 & x_2^2 & \dots & x_2^n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_m & x_m^2 & \dots & x_m^n \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{bmatrix} - \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix} \right\|_2^2$$

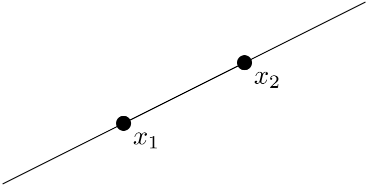
- special case of convex optimisation



Lines and Line Segments

- a **line** through two points x_1 and x_2

$$x = \theta x_1 + (1 - \theta) x_2, \quad (\theta \in \mathbb{R})$$

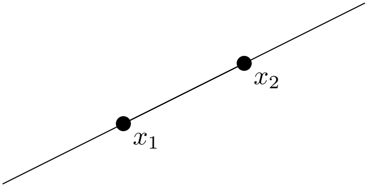


- an **affine set** contains the line through any two distinct points in the set
- an **affine hull** the set formed by taking all lines through points in a set

Lines and Line Segments

- a **line** through two points x_1 and x_2

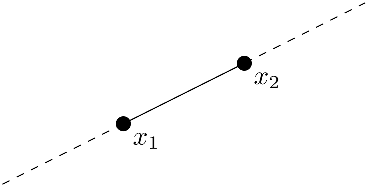
$$x = \theta x_1 + (1 - \theta)x_2, \quad (\theta \in \mathbb{R})$$



- an **affine set** contains the line through any two distinct points in the set
- an **affine hull** the set formed by taking all lines through points in a set

- a **line segment** between x_1 and x_2

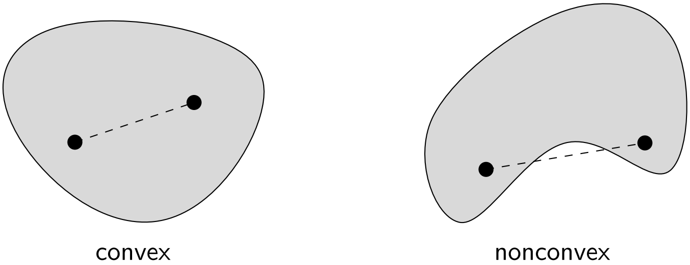
$$x = \theta x_1 + (1 - \theta)x_2, \quad (0 \leq \theta \leq 1)$$



- a **convex set** contains the line segment between any two distinct points in the set
- an **convex hull** the set formed by taking all line segments between points in a set

Convex Sets

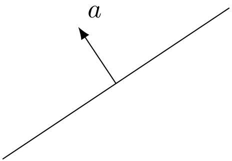
$$x_1, x_2 \in \text{convex set } C \quad \Longrightarrow \quad \theta x_1 + (1-\theta)x_2 \in C \text{ for all } 0 \le \theta \le 1$$



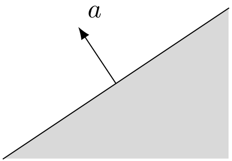
common examples in machine learning:

- nonnegative orthant, $\mathbb{R}^n_+ = \{x \mid x_i \ge 0, i = 1, \ldots, n\}$
- positive semindefinite matrices, $\mathbb{S}^n_+ = \{X \mid z^T X z \ge 0, z \in \mathbb{R}^n\}$

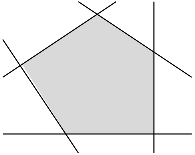
More Examples



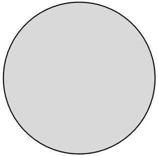
hyperplane,
$\{x \mid a^T x = b\}$



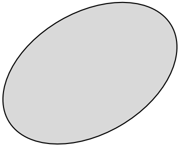
halfspace,
$\{x \mid a^T x \leq b\}$



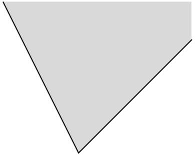
polyhedron,
$\{x \mid Ax \preceq b, Cx = d\}$



norm ball,
$\{x \mid \|x - x_c\|_p \leq r\}$



ellipsoid,
$\{Ax + b \mid \|x\|_2 \leq 1\}$



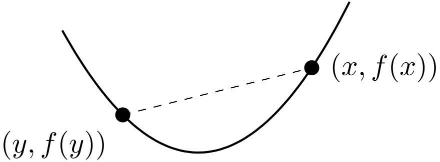
Lorentz cone,
$\{(x, t) \mid \|x\| \leq t\}$

Convex Functions

A function $f : \mathbb{R}^n \to \mathbb{R}$ is convex if **dom** (f) is a convex set and

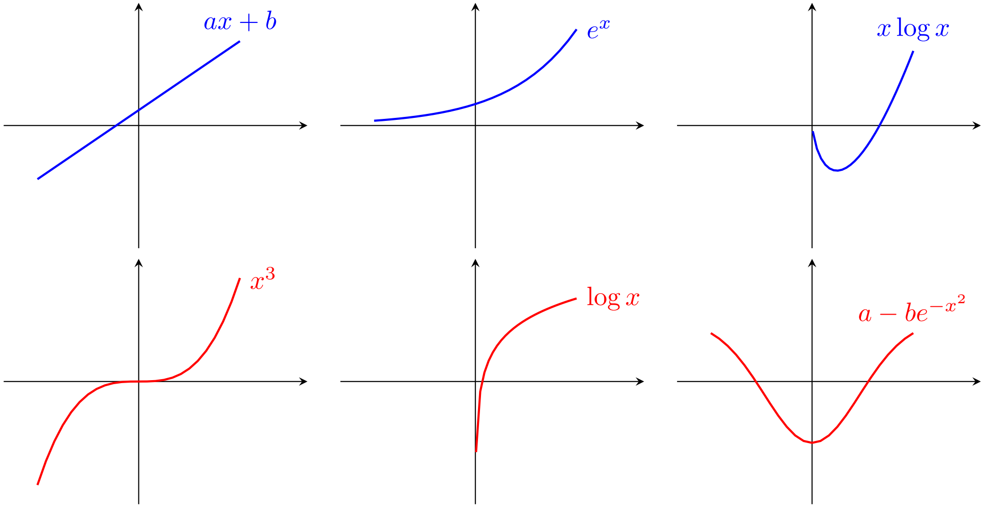
$$f(\theta x + (1-\theta)y) \leq \theta f(x) + (1-\theta) f(y)$$

for all $x, y \in$ **dom** $(f), 0 \leq \theta \leq 1$.

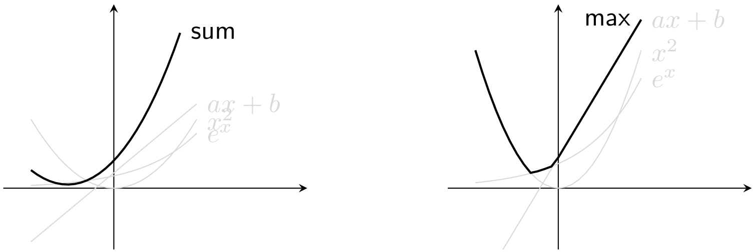


- f is concave if $-f$ is convex

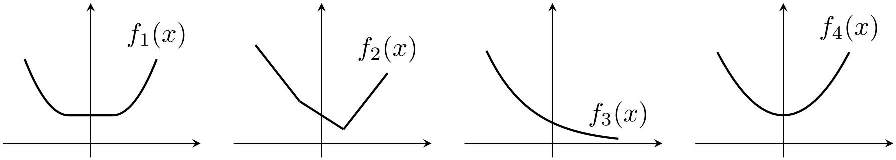
Examples



Weighted Sum and Pointwise Maximum Preserve Convexity



Convex, Strictly Convex, and Strongly Convex

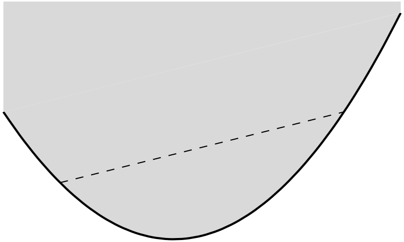


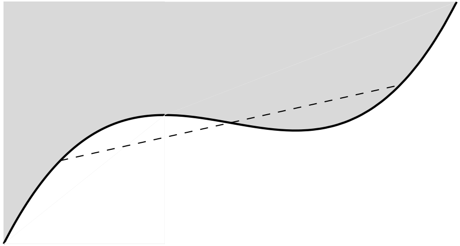
- f_1 is smooth and convex: $f(\theta x + (1-\theta)y) \leq \theta f(x) + (1-\theta)y$
- f_2 is non-differentiable and convex: $f(\theta x + (1-\theta)y) \leq \theta f(x) + (1-\theta)y$
- f_3 is strictly convex: $f(\theta x + (1-\theta)y) < \theta f(x) + (1-\theta)y$
- f_4 is strongly convex: $\exists m$ s.t. $m(y-x)^2 \leq f(y) - f(x)$

Epigraph

The epigraph of function $f : \mathbb{R}^n \to \mathbb{R}$ is the set

$$\textbf{epi}(f) = \{(x, t) \in \mathbb{R}^{n+1} \mid x \in \textbf{dom}\,(f)\,, f(x) \leq t\}.$$



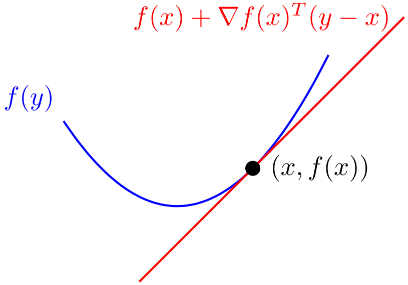


- f is a convex function if and only if $\textbf{epi}(f)$ is a convex set

First-order Condition

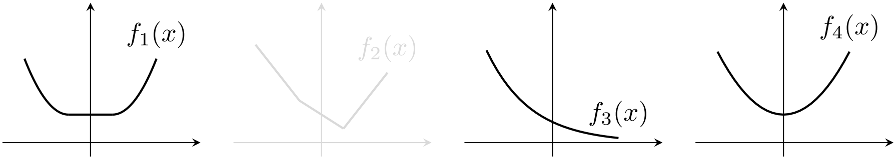
differentiable f with convex domain is convex iff

$$f(y) \geq f(x) + \nabla f(x)^T (y - x) \quad \text{for all } x, y \in \textbf{dom}\,(f)$$



- first-order approximation of (convex) f is a global under estimator

Second-order Condition



twice differentiable f with convex domain is convex iff

$$\nabla^2 f(x) \succeq 0 \quad \text{for all } x \in \textbf{dom}\,(f)$$

- if $\nabla^2 f(x) \succ 0$ for all $x \in \textbf{dom}\,(f)$, then f is strictly convex
- if $\nabla^2 f(x) \succeq mI$ for some $m > 0$ and all $x \in \textbf{dom}\,(f)$, then f is strongly convex
- strongly convex functions have a unique minimum

Worked Example: *log-sum-exp* is Convex

$$f(x) = \log \sum_{k=1}^{n} \exp x_k$$

Worked Example: *log-sum-exp* is Convex

$$f(x) = \log \sum_{k=1}^{n} \exp x_k$$

Proof. Start by computing the gradient and Hessian,

$$\frac{\partial f(x)}{\partial x_i} = \frac{\exp x_i}{\sum_{k=1}^{n} \exp x_k} \qquad \text{(derivative of } \log(z),\ z'/z\text{)}$$

$$\frac{\partial^2 f(x)}{\partial x_i \partial x_j} = \frac{\left(\sum_{k=1}^{n} \exp x_k\right) [\![i = j]\!] \exp x_i - \exp x_i \exp x_j}{\left(\sum_{k=1}^{n} \exp x_k\right)^2} \qquad \text{(quotient rule, } \frac{v \cdot \mathrm{d}u - u \cdot \mathrm{d}v}{v^2}\text{)}$$

Worked Example: *log-sum-exp* is Convex

$$f(x) = \log \sum_{k=1}^{n} \exp x_k$$

Proof. Start by computing the gradient and Hessian,

$$\frac{\partial f(x)}{\partial x_i} = \frac{z_i}{\mathbf{1}^T z} \qquad (z_k = \exp x_k)$$

$$\frac{\partial^2 f(x)}{\partial x_i \partial x_j} = \frac{\left(\mathbf{1}^T z\right) [\![i = j]\!] z_i - z_i z_j}{\left(\mathbf{1}^T z\right)^2}$$

Worked Example: *log-sum-exp* is Convex

$$f(x) = \log \sum_{k=1}^{n} \exp x_k$$

Proof. Start by computing the gradient and Hessian,

$$\nabla f(x) = \frac{1}{\mathbf{1}^T z} z \qquad (z_k = \exp x_k)$$

$$\nabla^2 f(x) = \frac{1}{\left(\mathbf{1}^T z\right)^2} \left((\mathbf{1}^T z)\,\mathbf{diag}(z) - z z^T \right)$$

Worked Example: *log-sum-exp* is Convex

$$f(x) = \log \sum_{k=1}^{n} \exp x_k$$

Proof. Start by computing the gradient and Hessian,

$$\nabla^2 f(x) = \frac{1}{\left(\mathbf{1}^T z\right)^2} \left((\mathbf{1}^T z)\mathbf{diag}(z) - zz^T \right) \qquad (z_k = \exp x_k)$$

To show that $\nabla^2 f(x) \succeq 0$, we must verify that $v^T \nabla^2 f(x) v \geq 0$ for all v.

Worked Example: *log-sum-exp* is Convex

$$f(x) = \log \sum_{k=1}^{n} \exp x_k$$

Proof. Start by computing the gradient and Hessian,

$$\nabla^2 f(x) = \frac{1}{\left(\mathbf{1}^T z\right)^2} \left((\mathbf{1}^T z)\textbf{diag}(z) - zz^T \right) \qquad (z_k = \exp x_k)$$

To show that $\nabla^2 f(x) \succeq 0$, we must verify that $v^T \nabla^2 f(x) v \geq 0$ for all v.

$$\begin{aligned} v^T \nabla^2 f(x) v &= \frac{1}{\left(\mathbf{1}^T z\right)^2} v^T \left((\mathbf{1}^T z)\textbf{diag}(z) - zz^T \right) v \\ &= \frac{1}{\left(\mathbf{1}^T z\right)^2} \left((\mathbf{1}^T z) v^T \textbf{diag}(z) v - v^T zz^T v \right) \end{aligned}$$

Worked Example: *log-sum-exp* is Convex

$$f(x) = \log \sum_{k=1}^{n} \exp x_k$$

Proof. Start by computing the gradient and Hessian,

$$\nabla^2 f(x) = \frac{1}{(\mathbf{1}^T z)^2} \left((\mathbf{1}^T z)\textbf{diag}(z) - zz^T \right) \qquad (z_k = \exp x_k)$$

$$\begin{aligned} v^T \nabla^2 f(x) v &= \frac{1}{(\mathbf{1}^T z)^2} v^T \left((\mathbf{1}^T z)\textbf{diag}(z) - zz^T \right) v \\ &= \frac{1}{(\mathbf{1}^T z)^2} \left((\mathbf{1}^T z) v^T \textbf{diag}(z) v - v^T z z^T v \right) \end{aligned}$$

Therefore we need to show that $(\mathbf{1}^T z) v^T \textbf{diag}(z) v \geq (v^T z)^2$ for all v.

Worked Example: *log-sum-exp* is Convex

$$f(x) = \log \sum_{k=1}^{n} \exp x_k$$

Proof. Start by computing the gradient and Hessian,

$$\nabla^2 f(x) = \frac{1}{(\mathbf{1}^T z)^2} \left((\mathbf{1}^T z)\mathbf{diag}(z) - zz^T \right) \qquad (z_k = \exp x_k)$$

Therefore we need to show that $(\mathbf{1}^T z) v^T \mathbf{diag}(z) v \geq (v^T z)^2$ for all v.

Worked Example: *log-sum-exp* is Convex

$$f(x) = \log \sum_{k=1}^{n} \exp x_k$$

Proof. Start by computing the gradient and Hessian,

$$\nabla^2 f(x) = \frac{1}{(\mathbf{1}^T z)^2} \left((\mathbf{1}^T z)\mathbf{diag}(z) - zz^T \right) \qquad (z_k = \exp x_k)$$

Therefore we need to show that $(\mathbf{1}^T z) v^T \mathbf{diag}(z) v \geq (v^T z)^2$ for all v. That is, we need to show

$$\left(\sum_{k=1}^{n} z_k \right) \left(\sum_{k=1}^{n} z_k v_k^2 \right) \geq \left(\sum_{k=1}^{n} v_k z_k \right)^2$$

Worked Example: *log-sum-exp* is Convex

$$f(x) = \log \sum_{k=1}^{n} \exp x_k$$

Proof. Start by computing the gradient and Hessian,

$$\nabla^2 f(x) = \frac{1}{(\mathbf{1}^T z)^2}\left((\mathbf{1}^T z)\,\mathbf{diag}(z) - zz^T\right) \qquad (z_k = \exp x_k)$$

Therefore we need to show that $(\mathbf{1}^T z) v^T \mathbf{diag}(z) v \geq (v^T z)^2$ for all v. That is, we need to show

$$\left(\sum_{k=1}^{n} z_k\right)\left(\sum_{k=1}^{n} z_k v_k^2\right) \geq \left(\sum_{k=1}^{n} v_k z_k\right)^2$$

which is true by the Cauchy-Schwarz inequality, $\|a\|_2^2 \|b\|_2^2 \geq (a^T b)^2$, with $a = (\sqrt{z_1}, \ldots, \sqrt{z_n})$ and $b = (\sqrt{z_1} v_1, \ldots, \sqrt{z_n} v_n)$.

Convex Optimisation

$$
\begin{array}{ll}
\text{minimize} & f_0(x) \\
\text{subject to} & f_i(x) \le 0, \quad i = 1, \dots, p \\
& a_i^T x = b_i, \quad i = 1, \dots, q
\end{array}
$$

- $f_0, f_1, \dots, f_p$ are convex
- $h_i(x) \triangleq a_i^T x - b_i$ are affine, often written as $Ax = b$

minimise a convex objective over a convex feasible set

Local Optima are Global Optima

any local minimum of a convex problem is (globally) optimal

Local Optima are Global Optima

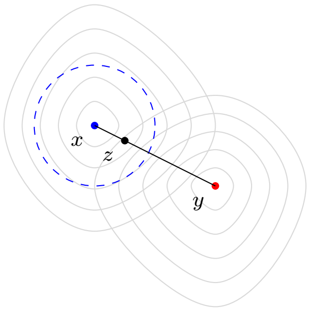
any local minimum of a convex problem is (globally) optimal

Proof Sketch.

- towards contradiction, suppose x is locally optimal, but there exists a feasible y with lower objective
- since x is locally optimally there exists a radius R such that no other point within R of x has lower objective
- (so y must be further than R from x)
- pick a point z on the line segment between x and y and within R of x
- so z must be feasible and have objective no lower than x
- but, by the basic inequality of convex functions,

$$f_0(\theta x + (1-\theta)y) \leq \theta f_0(x) + (1-\theta) f_0(y),$$

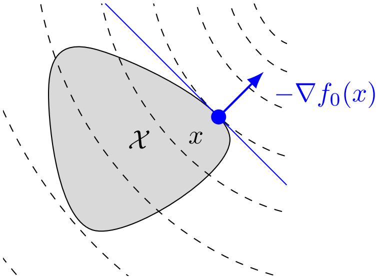
  the objective value at z must be between that at x and y, i.e., lower than $f_0(x)$
- we have a contradiction



full proof

Optimality Criterion for Differentiable f_0

x is optimal if and only if it is feasible and $\nabla f_0(x)^T(y - x) \geq 0$ for all feasible y



if nonzero,

- $\nabla f_0(x)$ defines a supporting hyperplane to feasible set $\mathcal{X}$ at x
- f_0 cannot be improved by moving in a direction where x stays feasible

Lagrangian

Standard form problem (not necessarily convex),

$$\begin{array}{ll} \text{minimize} & f_0(x) \\ \text{subject to} & f_i(x) \leq 0, \quad i = 1, \ldots, p \\ & h_i(x) = 0, \quad i = 1, \ldots, q \end{array}$$

variable $x \in \mathbb{R}^n$, domain $\mathcal{D}$, optimal value $p^\star$

Lagrangian

Standard form problem (not necessarily convex),

$$\begin{array}{ll} \text{minimize} & f_0(x) \\ \text{subject to} & f_i(x) \leq 0, \quad i = 1, \ldots, p \\ & h_i(x) = 0, \quad i = 1, \ldots, q \end{array}$$

variable $x \in \mathbb{R}^n$, domain $\mathcal{D}$, optimal value $p^\star$

Lagrangian: $\mathcal{L} : \mathbb{R}^n \times \mathbb{R}^p \times \mathbb{R}^q \to \mathbb{R}$, with **dom** $(\mathcal{L}) = \mathcal{D} \times \mathbb{R}^p \times \mathbb{R}^q$,

$$\mathcal{L}(x, \lambda, \nu) = f_0(x) + \sum_{i=1}^{p} \lambda_i f_i(x) + \sum_{i=1}^{q} \nu_i h_i(x)$$

- weighted sum of objective and constraint functions
- λ_i is the Lagrange multiplier (dual variable) associated with $f_i(x) \leq 0$
- ν_i is the Lagrange multiplier (dual variable) associated with $h_i(x) = 0$

duality

Karush-Kuhn-Tucker (KKT) Conditions

The following four conditions are called KKT conditions (for differentiable f_i, h_i):

- primal feasible: $$\begin{aligned} f_i(x) \leq 0, &\quad i = 1, \ldots, p \\ h_i(x) = 0, &\quad i = 1, \ldots, q \end{aligned}$$
- dual feasible: $\lambda \succeq 0$
- complementary slackness: $\lambda_i f_i(x) = 0$ for $i = 1, \ldots, p$
- gradient of Lagrangian with respect to x vanishes,

$$\nabla f_0(x) + \sum_{i=1}^{p} \lambda_i \nabla f_i(x) + \sum_{i=1}^{q} \nu_i \nabla h_i(x) = 0$$

Generalizes optimality condition $\nabla f_0(x) = 0$ for unconstrained problems.

Gradient Descent

$$\text{minimize} \quad f_0(x)$$

- f_0 convex, twice continuously differentiable
- we assume optimal value $p^\star = \inf_x f_0(x)$ is attained (and finite)

Gradient Descent

$$\text{minimize} \quad f_0(x)$$

- f_0 convex, twice continuously differentiable
- we assume optimal value $p^\star = \inf_x f_0(x)$ is attained (and finite)

Gradient descent:

1. **given** a starting point $x \in \textbf{dom}\,(f_0)$
2. **repeat** $x := x - t\nabla f_0(x)$. (choose step size, t)
3. **until** stopping criterion satisfied, e.g., $\|\nabla f_0(x)\|_2 \leq \epsilon$.

- variants of gradient descent define step direction Δx different to $-\nabla f_0(x)$

Choosing Step Size

fixed schedule: set t to a small constant or decay with each iteration

exact line search: $t = \text{argmin}_{t>0} f_0(x + t\Delta x)$

backtracking line search (with parameters $\alpha \in (0, 1/2), \beta \in (0, 1)$)

- starting at $t = 1$ with search direction Δx, repeat $t := \beta t$ until

$$f_0(x + t\Delta x) < f_0(x) + \alpha t \nabla f_0(x)^T \Delta x$$

Choosing Step Size

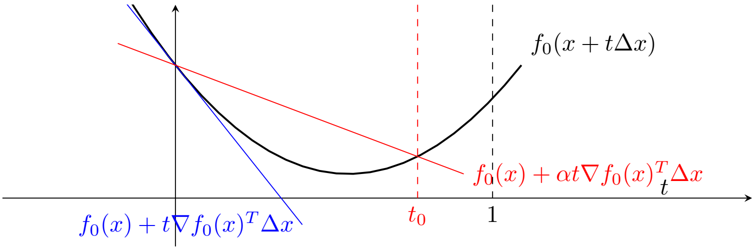
fixed schedule: set t to a small constant or decay with each iteration

exact line search: $t = \text{argmin}_{t>0} f_0(x + t\Delta x)$

backtracking line search (with parameters $\alpha \in (0, 1/2), \beta \in (0, 1)$)

- starting at $t = 1$ with search direction Δx, repeat $t := \beta t$ until

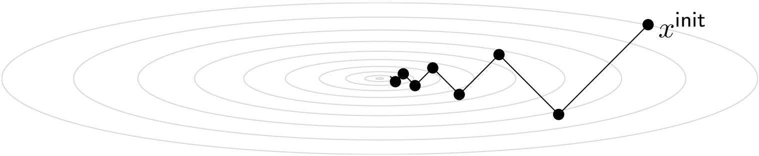
$$f_0(x + t\Delta x) < f_0(x) + \alpha t \nabla f_0(x)^T \Delta x$$



Example

Gradient descent (even with exact line search) can be slow. E.g.,

$$f_0(x) = x_1^2 + \gamma x_2^2, \quad \gamma \gg 1$$



Newton's Method

$$\Delta x_{\mathsf{nt}} = -\nabla^2 f_0(x)^{-1} \nabla f_0(x)$$

- $x + \Delta x_{\mathsf{nt}}$ minimizes the second-order approximation of f_0 at x,

$$\hat{f}(x+v) = f_0(x) + \nabla f_0(x)^T v + \frac{1}{2} v^T \nabla^2 f_0(x) v$$

Newton's method:

1. **given** a starting point $x \in \textbf{dom}\,(f_0)$.
2. **repeat** $x := x + t\Delta x_{\mathsf{nt}}$. (choose step size, t)
3. **until** stopping criterion satisfied.

Equality Constrained Methods

$$\begin{array}{ll} \text{minimize} & f_0(x) \\ \text{subject to} & Ax = b \end{array}$$

- f_0 convex, twice continuously differentiable
- $A \in \mathbb{R}^{q \times n}$ with $\textbf{rank}(A) = q$ (and $b \in \textbf{range}(A)$)
- we assume $p^\star$ is finite and attained

Equality Constrained Methods

$$\begin{array}{ll} \text{minimize} & f_0(x) \\ \text{subject to} & Ax = b \end{array}$$

- f_0 convex, twice continuously differentiable
- $A \in \mathbb{R}^{q \times n}$ with $\textbf{rank}(A) = q$ (and $b \in \textbf{range}(A)$)
- we assume $p^\star$ is finite and attained

optimality condition: $x^\star$ is optimal iff there exists a $\nu^\star$ such that

$$\nabla f_0(x^\star) + A^T \nu^\star = 0, \quad Ax^\star = b$$

Newton Step for Equality Constrained Optimisation

Newton step Δx_{nt} of f_0 at feasible x is given by solution v of

$$\begin{bmatrix} \nabla^2 f_0(x) & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} v \\ w \end{bmatrix} = \begin{bmatrix} -\nabla f_0(x) \\ 0 \end{bmatrix}$$

- second row ensures that x iterates stay feasible
- solves quadratic approximation of optimisation problem

$$\begin{array}{ll} \text{minimize} & \hat{f}(x+v) \triangleq f_0(x) + \nabla f_0(x)^T v + \frac{1}{2} v^T \nabla^2 f_0(x) v \\ \text{subject to} & A(x+v) = b \end{array}$$

- solves linear approximation of optimality condition

The Barrier Method

For inequality constrained problems,

$$\begin{array}{ll} \text{minimize} & f_0(x) \\ \text{subject to} & f_i(x) \leq 0, \quad i = 1, \ldots, p \\ & Ax = b \end{array}$$

The Barrier Method

For inequality constrained problems,

$$\begin{array}{ll} \text{minimize} & f_0(x) \\ \text{subject to} & f_i(x) \leq 0, \quad i = 1, \ldots, p \\ & Ax = b \end{array}$$

we reformulate using an indicator function,

$$\begin{array}{ll} \text{minimize} & f_0(x) + \sum_{i=1}^{p} I_{\mathbb{R}_-}(f_i(x)) \\ \text{subject to} & Ax = b \end{array}$$

where $I_{\mathbb{R}_-}(u) = 0$ if $u \leq 0$ and $I_{\mathbb{R}_-}(u) = \infty$ otherwise,

The Barrier Method

For inequality constrained problems,

$$\begin{array}{ll} \text{minimize} & f_0(x) \\ \text{subject to} & f_i(x) \leq 0, \quad i = 1, \ldots, p \\ & Ax = b \end{array}$$

we reformulate using an indicator function,

$$\begin{array}{ll} \text{minimize} & f_0(x) + \sum_{i=1}^{p} I_{\mathbb{R}_-}(f_i(x)) \\ \text{subject to} & Ax = b \end{array}$$

where $I_{\mathbb{R}_-}(u) = 0$ if $u \leq 0$ and $I_{\mathbb{R}_-}(u) = \infty$ otherwise, which we approximate with a logarithmic barrier

$$\begin{array}{ll} \text{minimize} & f_0(x) - \frac{1}{t} \sum_{i=1}^{p} \log(-f_i(x)) \\ \text{subject to} & Ax = b \end{array}$$

to get an equality constrained approximation.

The Barrier Method

For inequality constrained problems,

$$\begin{array}{ll} \text{minimize} & f_0(x) \\ \text{subject to} & f_i(x) \leq 0, \quad i = 1, \ldots, p \\ & Ax = b \end{array}$$

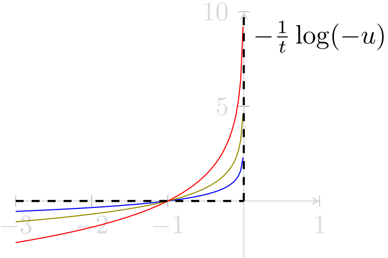
we reformulate using an indicator function,

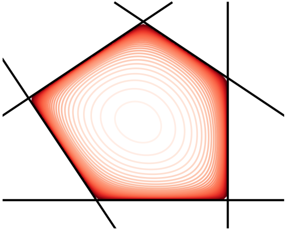
$$\begin{array}{ll} \text{minimize} & f_0(x) + \sum_{i=1}^{p} I_{\mathbb{R}_-}(f_i(x)) \\ \text{subject to} & Ax = b \end{array}$$

where $I_{\mathbb{R}_-}(u) = 0$ if $u \leq 0$ and $I_{\mathbb{R}_-}(u) = \infty$ otherwise, which we approximate with a logarithmic barrier

$$\begin{array}{ll} \text{minimize} & f_0(x) - \frac{1}{t} \sum_{i=1}^{p} \log(-f_i(x)) \\ \text{subject to} & Ax = b \end{array}$$

to get an equality constrained approximation.





Algorithms for Large Scale Problems

- for large scale problems, e.g., deep learning, Newton's method is too expensive
- even computing the true gradient may be too expensive
- many loss functions in machine learning decompose over train data $\{(x_i, y_i)\}_{i=1}^m$,

$$L(\theta) = \sum_{i=1}^{m} \ell(f(x_i; \theta), y_i)$$

- SGD approximates the gradient on mini-batches $\mathcal{I} \subseteq \{1, \ldots, m\}$

$$\widehat{\nabla_\theta L} = \sum_{i \in \mathcal{I}} \nabla_\theta \ell(f(x_i; \theta), y_i)$$

- under mild assumptions $E\left[\widehat{\nabla_\theta L}\right] = \nabla_\theta L$
- for constrained problems can project back onto feasible set

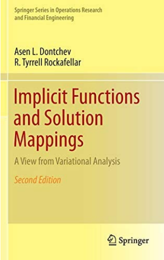
Many, many other schemes and variations!

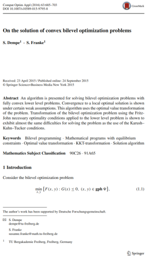
lecture 2

Lecture 2: Differentiable Optimisation and Deep Learning


Deep Declarative Networks

Stephen Gould, Member, IEEE, Richard Hartley, Fellow, IEEE, Dylan Campbell, Member, IEEE




On the solution of convex bilevel optimization problems

S. Dempe · S. Franke


On Differentiating Parameterized Argmin and Argmax Problems with Application to Bi-level Optimization

Stephen Gould, Basura Fernando, Anoop Cherian, Peter Anderson, Rodrigo Santa Cruz, and Edison Guo

July 21, 2016

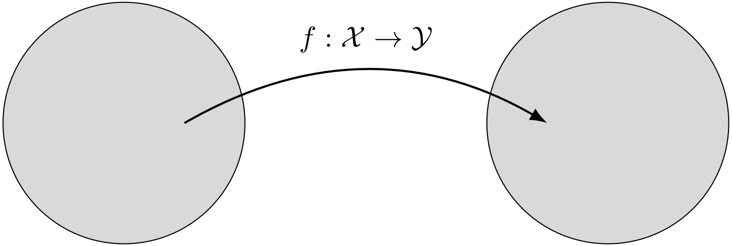

Differentiable Optimization-Based Modeling for Machine Learning

Brandon Amos

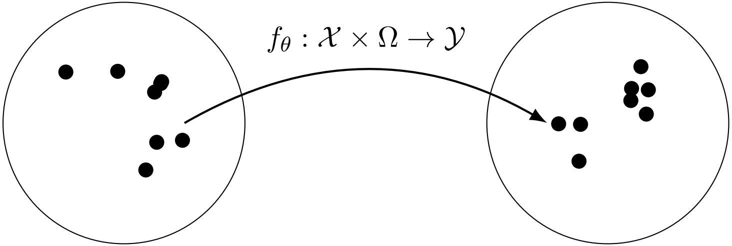
CMU-CS-19-109

May 2019

Machine Learning from 10,000ft



Machine Learning from 10,000ft

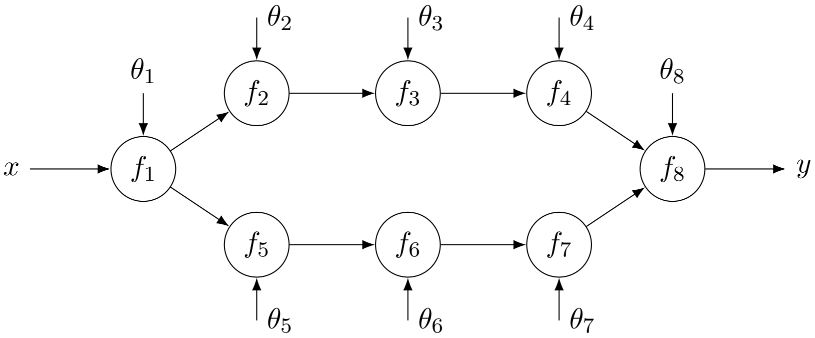


minimize (over θ) $\sum_{(x,y)\sim\mathcal{X}\times\mathcal{Y}} L(f_\theta(x), y)$

- loss L — what to do
- model f_θ — how to do it
- optimised by gradient descent

Deep Learning as an End-to-end Computation Graph

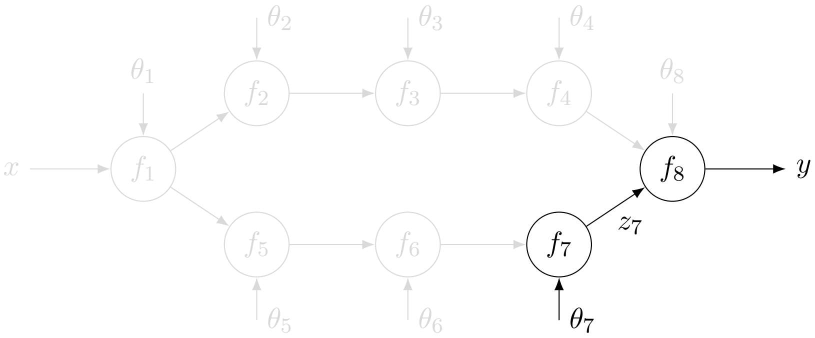
Deep learning does this by defining a function (equiv. computation graph) composed of many simple parametrized functions (equiv. computation nodes).



$$y = f_8(f_4(f_3(f_2(f_1(x)))), f_7(f_6(f_5(f_1(x)))))$$

(parameters θ_i omitted for brevity)

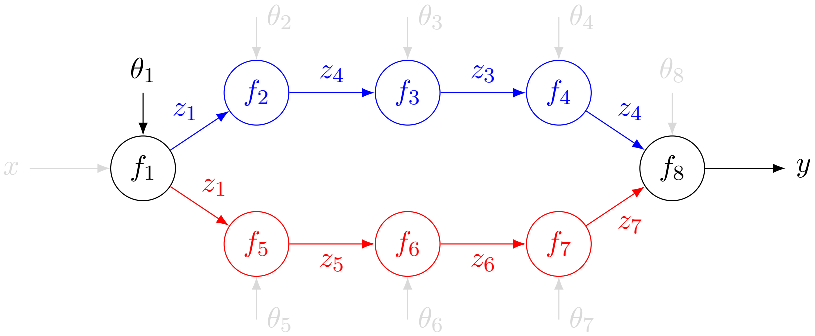
Backward Pass



Example 1.

$$\frac{\partial L}{\partial \theta_7} = \frac{\partial L}{\partial y} \frac{\partial y}{\partial z_7} \frac{\partial z_7}{\partial \theta_7}$$

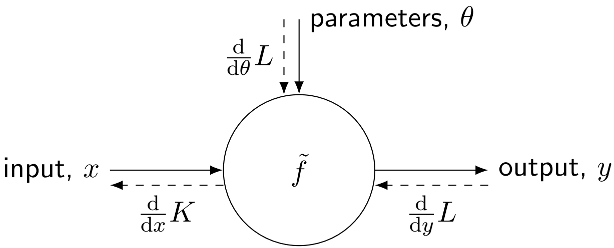
Backward Pass



Example 2.

$$\frac{\partial L}{\partial \theta_1} = \frac{\partial L}{\partial y} \left(\frac{\partial y}{\partial z_4} \frac{\partial z_4}{\partial z_3} \frac{\partial z_3}{\partial z_2} \frac{\partial z_2}{\partial z_1} + \frac{\partial y}{\partial z_7} \frac{\partial z_7}{\partial z_6} \frac{\partial z_6}{\partial z_5} \frac{\partial z_5}{\partial z_4} \right) \frac{\partial z_1}{\partial \theta_1}$$

Deep Learning Node



- **Forward pass:** compute output y as a function of the input x (and model parameters θ).
- **Backward pass:** compute the derivative of the loss with respect to the input x (and model parameters θ) given the derivative of the loss with respect to the output y.

Notational Aside (Often Sloppy)

For scalar-valued functions:

total derivative: $\dfrac{\mathrm{d}f}{\mathrm{d}x}$ partial derivative: $\dfrac{\partial f}{\partial x}$

Notational Aside (Often Sloppy)

For scalar-valued functions:

total derivative: $\frac{\mathrm{d}f}{\mathrm{d}x}$ partial derivative: $\frac{\partial f}{\partial x}$

For multi-dimensional scalar-valued functions, $f : \mathbb{R}^n \to \mathbb{R}$:

$$\nabla f(x) = \left(\frac{\mathrm{d}f}{\mathrm{d}x_1}, \ldots, \frac{\mathrm{d}f}{\mathrm{d}x_n} \right) \in \mathbb{R}^n$$

Notational Aside (Often Sloppy)

For scalar-valued functions:

total derivative: $\frac{\mathrm{d}f}{\mathrm{d}x}$ partial derivative: $\frac{\partial f}{\partial x}$

For multi-dimensional scalar-valued functions, $f : \mathbb{R}^n \to \mathbb{R}$:

$$\nabla f(x) = \left(\frac{\mathrm{d}f}{\mathrm{d}x_1}, \dots, \frac{\mathrm{d}f}{\mathrm{d}x_n} \right) \in \mathbb{R}^n$$

For multi-dimensional vector-valued functions, $f : \mathbb{R}^n \to \mathbb{R}^m$:

$$\frac{\mathrm{d}}{\mathrm{d}x} f(x) = \begin{bmatrix} \frac{\mathrm{d}f_1}{\mathrm{d}x_1} & \cdots & \frac{\mathrm{d}f_1}{\mathrm{d}x_n} \\ \vdots & \ddots & \vdots \\ \frac{\mathrm{d}f_m}{\mathrm{d}x_1} & \cdots & \frac{\mathrm{d}f_m}{\mathrm{d}x_n} \end{bmatrix} \in \mathbb{R}^{m \times n}$$

($\frac{\partial}{\partial x} f(x, y)$ for partial)

Sometimes D and D_X for $\frac{\mathrm{d}}{\mathrm{d}x}$ and $\frac{\partial}{\partial x}$, respectively.

Automatic Differentiation (AD)

- algorithmic procedure that produces code for computing exact derivatives
- assumes numeric computations are composed of a small set of elementary operations that we know how to differentiate
  - arithmetic, exp, log, trigonometric
- workhorse of modern machine learning that greatly reduces development effort

Automatic Differentiation (AD)

- algorithmic procedure that produces code for computing exact derivatives
- assumes numeric computations are composed of a small set of elementary operations that we know how to differentiate
  - arithmetic, exp, log, trigonometric
- workhorse of modern machine learning that greatly reduces development effort
- two flavours
  - (forward mode) propagates results on the first-order approximation $x + \Delta x$ forward through the computations
  - (reverse mode) builds a program to compute derivative based on the chain rule re-using computation where applicable

$$\frac{\mathrm{d}L}{\mathrm{d}x} = \frac{\mathrm{d}L}{\mathrm{d}y}\frac{\mathrm{d}y}{\mathrm{d}x}$$

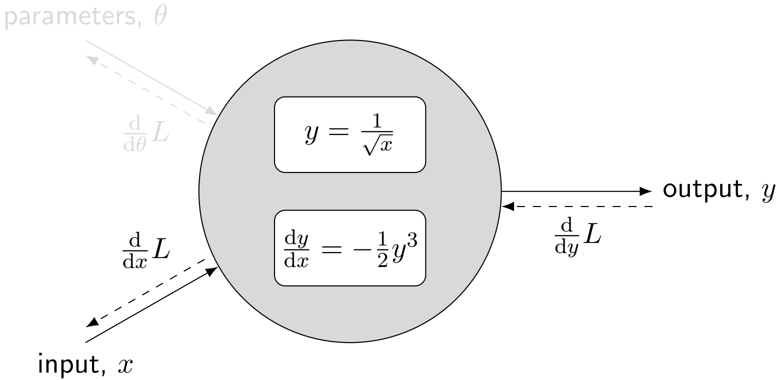
- different deep learning frameworks use slightly different approaches (explicit graph construction versus implicit operator tracking)

example

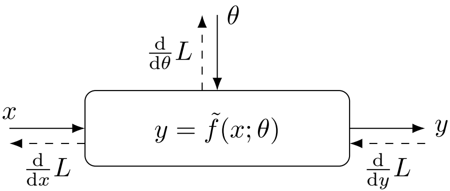
Computing $1/\sqrt{x}$

```c
float Q_rsqrt( float number )
{
    long i;
    float x2, y;
    const float threehalfs = 1.5F;

    x2 = number * 0.5F;
    y = number;
    i = * ( long * ) &y;                  // evil floating point bit level hacking
    i = 0x5f3759df - ( i >> 1 );          // what the f**k?
    y = * ( float * ) &i;
    y = y * ( threehalfs - ( x2 * y * y ) );     // 1st iter
    //  y = y * ( threehalfs - ( x2 * y * y ) ); // 2nd iter, can be removed

    return y;
}
```

Separate Forward and Backward Operations



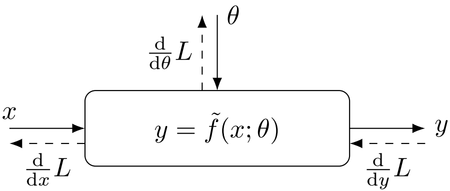
Imperative vs Declarative Nodes

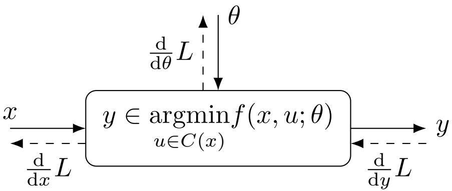


- imperative node
- input-output relationship explicit,

$$y = \tilde{f}(x;\theta)$$

Imperative vs Declarative Nodes





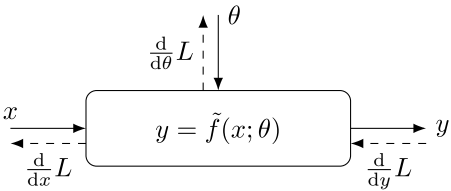
- imperative node
- input-output relationship explicit,

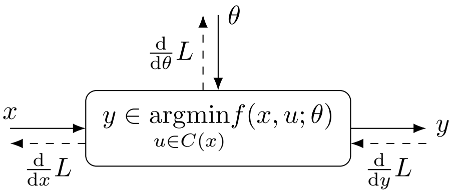
$$y = \tilde{f}(x;\theta)$$

- declarative node
- input-output relationship specified as solution to an optimisation problem,

$$y \in \arg\min_{u \in C(x)} f(x, u; \theta)$$

Imperative vs Declarative Nodes





- imperative node
- input-output relationship explicit,

$$y = \tilde{f}(x;\theta)$$

- declarative node
- input-output relationship specified as solution to an optimisation problem,

$$y \in \arg\min_{u \in C(x)} f(x, u; \theta)$$

can co-exist in the same computation graph (network)

Average Pooling Example

$$\{x_i \in \mathbb{R}^m \mid i = 1, \ldots, n\} \to \mathbb{R}^m$$

- imperative specification

$$y = \frac{1}{n} \sum_{i=1}^{n} x_i$$

- declarative specification

$$y = \text{argmin}_{u \in \mathbb{R}^m} \sum_{i=1}^{n} \|u - x_i\|^2$$

Average Pooling Example

$$\{x_i \in \mathbb{R}^m \mid i = 1, \ldots, n\} \to \mathbb{R}^m$$

- imperative specification

$$y = \frac{1}{n} \sum_{i=1}^{n} x_i$$

- declarative specification

$$y = \text{argmin}_{u \in \mathbb{R}^m} \sum_{i=1}^{n} \|u - x_i\|^2$$

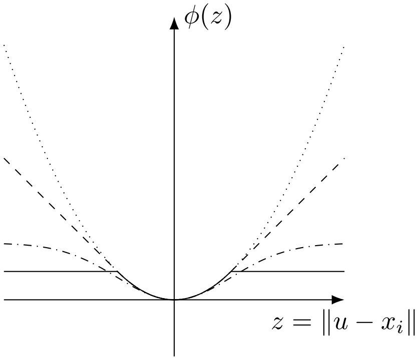
- can be easily varied, e.g., made robust

$$y = \text{argmin}_{u \in \mathbb{R}^m} \sum_{i=1}^{n} \phi(u - x_i)$$

for some penalty function ϕ

Average Pooling Example

$$\{x_i \in \mathbb{R}^m \mid i = 1, \ldots, n\} \to \mathbb{R}^m$$



- declarative specification

$$y = \text{argmin}_{u \in \mathbb{R}^m} \sum_{i=1}^{n} \|u - x_i\|^2$$

- can be easily varied, e.g., made robust

$$y = \text{argmin}_{u \in \mathbb{R}^m} \sum_{i=1}^{n} \phi(u - x_i)$$

for some penalty function ϕ

Bi-level Optimisation: Stackelberg Games

Consider two players, a **leader** and a **follower**

- the market dictates the price it's willing to pay for some goods based on supply, i.e., quantity produced by both players, $P(q_1 + q_2)$
- each player has a cost structure associated with producing goods, $C_i(q_i)$ and wants to maximize profits, $q_i P(q_1 + q_2) - C_i(q_i)$
- the leader picks a quantity of goods to produce knowing that the follower will respond optimally. In other words, the leader solves

$$\begin{array}{ll} \text{maximize (over } q_1) & q_1 P(q_1 + q_2) - C_1(q_1) \\ \text{subject to} & q_2 \in \operatorname{argmax}_q qP(q_1 + q) - C_2(q) \end{array}$$

Solving Bi-level Optimisation Problems

$$\begin{array}{ll} \text{minimize (over } x\text{)} & L(x, y) \\ \text{subject to} & y \in \operatorname{argmin}_{u \in C(x)} f(x, u) \end{array}$$

Solving Bi-level Optimisation Problems

$$\begin{array}{ll} \text{minimize (over } x) & L(x, y) \\ \text{subject to} & y \in \text{argmin}_{u \in C(x)} f(x, u) \end{array}$$

- **closed-form solution:** substitute for y in upper-level problem (if possible)

$$\text{minimize (over } x) \quad L(x, y(x))$$

Solving Bi-level Optimisation Problems

$$\begin{array}{ll} \text{minimize (over } x\text{)} & L(x, y) \\ \text{subject to} & y \in \text{argmin}_{u \in C(x)} f(x, u) \end{array}$$

- **closed-form solution:** substitute for y in upper-level problem (if possible)

$$\text{minimize (over } x\text{)} \quad L(x, y(x))$$

- **convex lower-level problem:** replace lower-level problem with sufficient optimality conditions (e.g., KKT conditions),

$$\begin{array}{ll} \text{minimize (over } x, y\text{)} & L(x, y) \\ \text{subject to} & h(x, y) = 0 \end{array}$$

Solving Bi-level Optimisation Problems

$$\begin{array}{ll} \text{minimize (over } x) & L(x, y) \\ \text{subject to} & y \in \operatorname{argmin}_{u \in C(x)} f(x, u) \end{array}$$

- **closed-form solution:** substitute for y in upper-level problem (if possible)

$$\text{minimize (over } x) \quad L(x, y(x))$$

- **convex lower-level problem:** replace lower-level problem with sufficient optimality conditions (e.g., KKT conditions),

$$\begin{array}{ll} \text{minimize (over } x, y) & L(x, y) \\ \text{subject to} & h(x, y) = 0 \end{array}$$

- **gradient descent:** compute gradient of lower-level solution y with respect to x, and use the chain rule to get the total derivative,

$$x \leftarrow x - \eta \left(\frac{\partial L(x, y)}{\partial x} + \frac{\partial L(x, y)}{\partial y} \frac{\mathrm{d}y}{\mathrm{d}x} \right)$$

Solving Bi-level Optimisation Problems

$$\begin{array}{ll} \text{minimize (over } x) & L(x, y) \\ \text{subject to} & y \in \text{argmin}_{u \in C(x)} f(x, u) \end{array}$$

- **closed-form solution:** substitute for y in upper-level problem (if possible)

$$\text{minimize (over } x) \quad L(x, y(x))$$

- **convex lower-level problem:** replace lower-level problem with sufficient optimality conditions (e.g., KKT conditions),

$$\begin{array}{ll} \text{minimize (over } x, y) & L(x, y) \\ \text{subject to} & h(x, y) = 0 \end{array}$$

- **gradient descent:** compute gradient of lower-level solution y with respect to x, and use the chain rule to get the total derivative,

$$x \leftarrow x - \eta \left(\frac{\partial L(x, y)}{\partial x} + \frac{\partial L(x, y)}{\partial y} \frac{\mathrm{d}y}{\mathrm{d}x} \right)$$

  - by back-propagating through optimisation procedure or implicit differentiation

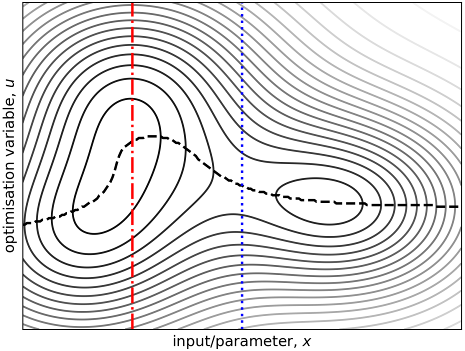
Parametrized Optimisation

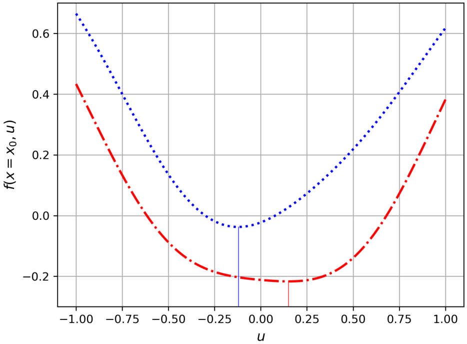
In the context of deep learning the upper-level Stackelberg problem is the **learning problem** and the lower-level Stackelberg problem is the **inference problem**.

A declarative node defines a family of problems indexed by continuous variable $x \in \mathbb{R}^n$,

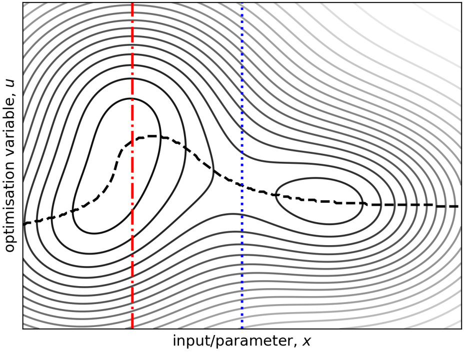
$$\left\{ \begin{array}{ll} \text{minimize (over } u \in \mathbb{R}^m) & f_0(x, u) \\ \text{subject to} & f_i(x, u) \leq 0, \quad i = 1, \ldots, p \\ & h_i(x, u) = 0, \quad i = 1, \ldots, q \end{array} \right\}_{x \in \mathbb{R}^n}$$

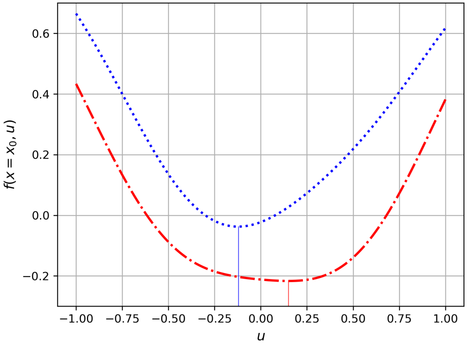
Parametrized Optimisation Example





Parametrized Optimisation Example





Main question: How do we compute $\frac{\mathrm{d}}{\mathrm{d}x}\operatorname{argmin}_u f(x,u)$?

Dini's Implicit Function Theorem

Consider the solution mapping associated with the equation $f(x, u) = 0$,

$$Y : x \mapsto \{u \in \mathbb{R}^m \mid f(x, u) = 0\} \text{ for } x \in \mathbb{R}^n.$$

We are interested in how elements of $Y(x)$ change as a function of x.

Dini's Implicit Function Theorem

Consider the solution mapping associated with the equation $f(x, u) = 0$,

$$Y : x \mapsto \{u \in \mathbb{R}^m \mid f(x, u) = 0\} \text{ for } x \in \mathbb{R}^n.$$

We are interested in how elements of $Y(x)$ change as a function of x.

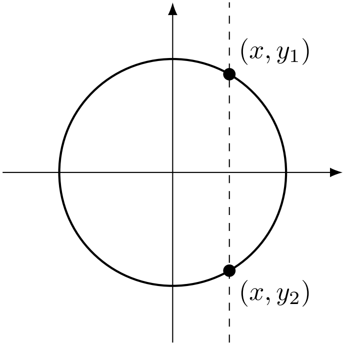
Theorem

Let $f : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^m$ be differentiable in a neighbourhood of (x, u) and such that $f(x, u) = 0$, and let $\frac{\partial}{\partial u} f(x, u)$ be nonsingular. Then the solution mapping Y has a single-valued localization y around x for u which is differentiable in a neighbourhood $\mathcal{X}$ of x with Jacobian satisfying

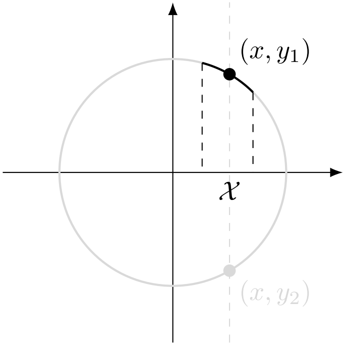
$$\frac{dy(x)}{dx} = -\left(\frac{\partial f(x, y(x))}{\partial y}\right)^{-1} \frac{\partial f(x, y(x))}{\partial x}$$

for every $x \in \mathcal{X}$.

Unit Circle Example



$$y = \pm\sqrt{1 - x^2}$$
$$\frac{\mathrm{d}y}{\mathrm{d}x} = \frac{\mp 2x}{2\sqrt{1 - x^2}} = -\frac{x}{y}$$



$$f(x, y) = x^2 + y^2 - 1$$
$$\frac{\mathrm{d}y}{\mathrm{d}x} = -\left(\frac{\partial f}{\partial y}\right)^{-1}\left(\frac{\partial f}{\partial x}\right)$$
$$= -\left(\frac{1}{2y}\right)(2x) = -\frac{x}{y}$$

Differentiating Unconstrained Optimisation Problems

Let $f : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ be twice differentiable and let

$$y(x) \in \text{argmin}_u f(x, u)$$

then for non-zero Hessian

$$\frac{\mathrm{d}y(x)}{\mathrm{d}x} = -\left(\frac{\partial^2 f}{\partial y^2}\right)^{-1} \frac{\partial^2 f}{\partial x \partial y}.$$

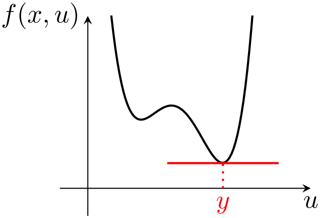
Differentiating Unconstrained Optimisation Problems

Let $f : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ be twice differentiable and let

$$y(x) \in \text{argmin}_u f(x, u)$$

then for non-zero Hessian

$$\frac{\mathrm{d}y(x)}{\mathrm{d}x} = -\left(\frac{\partial^2 f}{\partial y^2}\right)^{-1} \frac{\partial^2 f}{\partial x \partial y}.$$



Proof. The derivative of f vanishes at (x, y), i.e., $y \in \text{argmin}_u f(x, u) \implies \frac{\partial f(x,y)}{\partial y} = 0$.

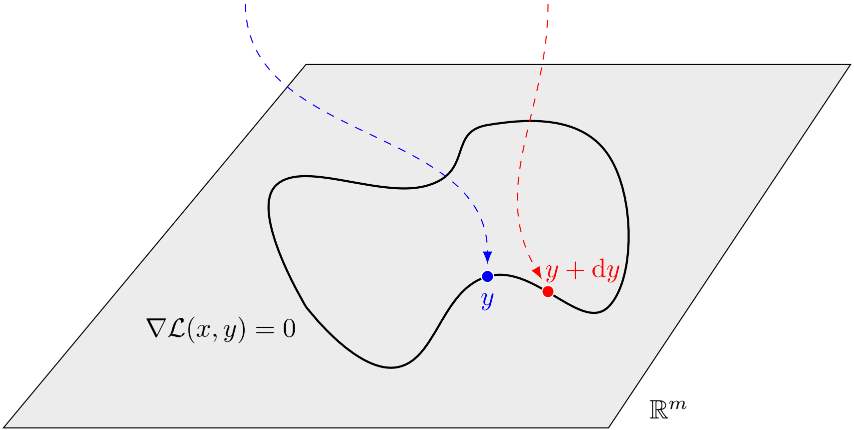
$$\text{LHS}: \quad \frac{\mathrm{d}}{\mathrm{d}x}\frac{\partial f(x,y)}{\partial y} = \frac{\partial^2 f(x,y)}{\partial x \partial y} + \frac{\partial^2 f(x,y)}{\partial y^2}\frac{\mathrm{d}y}{\mathrm{d}x}$$

$$\text{RHS}: \quad \frac{\mathrm{d}}{\mathrm{d}x} 0 = 0$$

Equating and rearranging gives the result.

Differentiable Optimisation: Big Picture Idea

$$\begin{array}{rl} \text{min.} & f_0(x, u) \\ \text{s.t.} & u \in C(x) \end{array} \qquad \begin{array}{rl} \text{min.} & f_0(x + \mathrm{d}x, u) \\ \text{s.t.} & u \in C(x + \mathrm{d}x) \end{array}$$



Differentiating Equality Constrained Optimisation Problems

Consider functions $f : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$ and $h : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^q$. Let

$$\begin{array}{rll} y(x) \in & \arg\min_{u \in \mathbb{R}^m} & f(x, u) \\ & \text{subject to} & h(x, u) = 0_q \end{array}$$

Assume that $y(x)$ exists, that f and h are twice differentiable in the neighbourhood of $(x, y(x))$, and that **rank**$(\frac{\partial h(x,y)}{\partial y}) = q$.

Differentiating Equality Constrained Optimisation Problems

Consider functions $f : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$ and $h : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^q$. Let

$$\begin{array}{rl} y(x) \in \arg\min_{u \in \mathbb{R}^m} & f(x, u) \\ \text{subject to} & h(x, u) = 0_q \end{array}$$

Assume that $y(x)$ exists, that f and h are twice differentiable in the neighbourhood of $(x, y(x))$, and that **rank**$(\frac{\partial h(x,y)}{\partial y}) = q$. Then for H non-singular

$$\frac{\mathrm{d}y(x)}{\mathrm{d}x} = H^{-1}A^T\left(AH^{-1}A^T\right)^{-1}\left(AH^{-1}B - C\right) - H^{-1}B$$

where

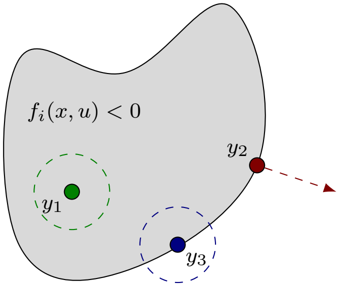
$$\begin{array}{ll} A = \frac{\partial h(x,y)}{\partial y} \in \mathbb{R}^{q \times m} & B = \frac{\partial^2 f(x,y)}{\partial x \partial y} - \sum_{i=1}^q \nu_i \frac{\partial^2 h_i(x,y)}{\partial x \partial y} \in \mathbb{R}^{m \times n} \\ C = \frac{\partial h(x,y)}{\partial x} \in \mathbb{R}^{q \times n} & H = \frac{\partial^2 f(x,y)}{\partial y^2} - \sum_{i=1}^q \nu_i \frac{\partial^2 h_i(x,y)}{\partial y^2} \in \mathbb{R}^{m \times m} \end{array}$$

and $\nu \in \mathbb{R}^q$ satisfies $\nu^T A = \frac{\partial f(x,y)}{\partial y}$.

derivation

Dealing with Inequality Constraints

$$
\begin{array}{rl}
y(x) \in \operatorname{arg\,min}_{u \in \mathbb{R}^m} & f_0(x, u) \\
\text{subject to} & h_i(x, u) = 0, \;\; i = 1, \ldots, q \\
& f_i(x, u) \leq 0, \;\; i = 1, \ldots, p.
\end{array}
$$



- Replace inequality constraints with log-barrier approximation (see last lecture)
- Treat as equality constraints if active (y_2 or y_3) and ignore otherwise (y_1 or y_3)
  - may lead to one-sided gradients since $\lambda \succeq 0$

Automatic Differentiation for Differentiable Optimisation

- At one extreme we can try back propagate through the optimisation algorithm (i.e., unrolling the optimisation procedure using automatic differentiation)
- At the other extreme we can use the implicit differentiation result to hand-craft efficient backward pass code
- There are two options in between:
  - Use automatic differentiation to obtain quantities A, B, C and H from software implementations of the objective and (active) constraint functions
  - Implement the optimality condition $\nabla \mathcal{L} = 0$ in software and automatically differentiate that

(in the next lecture we will see examples of the first two)

Vector-Jacobian Product

For brevity consider the unconstrained optimisation case. The backward pass computes

$$\frac{\mathrm{d}L}{\mathrm{d}x} = \frac{\mathrm{d}L}{\mathrm{d}y}\frac{\mathrm{d}y}{\mathrm{d}x}$$
$$= \underbrace{\left(v^T\right)}_{\mathbb{R}^{1\times m}}\underbrace{\left(-H^{-1}B\right)}_{\mathbb{R}^{m\times n}}$$

evaluation order:	$-v^T\left(H^{-1}B\right)$	$\left(-v^TH^{-1}\right)B$
cost[†]:	$O(m^2n + mn)$	$O(m^2 + mn)$

[†] assumes H^{-1} is already factored (in $O(m^3)$ if unstructured, less if structured)

Summary and Open Questions

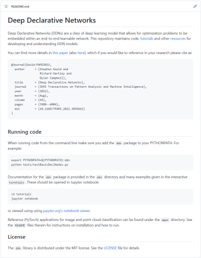
- optimisation problems can be embedded *inside* deep learning models
- back-propagation by either unrolling the optimisation algorithm or implicit differentiation of the optimality conditions
  - the former is easy to implement using automatic differentiation but memory intensive
  - the latter requires that solution be strongly convex locally (i.e., invertible H)
  - but does not need to know how the problem was solved, nor store intermediate forward-pass calculations
  - computing H^{-1} may be costly

Summary and Open Questions

- optimisation problems can be embedded *inside* deep learning models
- back-propagation by either unrolling the optimisation algorithm or implicit differentiation of the optimality conditions
  - the former is easy to implement using automatic differentiation but memory intensive
  - the latter requires that solution be strongly convex locally (i.e., invertible H)
  - but does not need to know how the problem was solved, nor store intermediate forward-pass calculations
  - computing H^{-1} may be costly
- active area of research and many open questions
  - Are declarative nodes slower?
  - Do declarative nodes give theoretical guarantees?
  - How best to handle non-smooth or discrete optimization problems?
  - What about problems with multiple solutions?
  - What if the forward pass solution is suboptimal?
  - Can problems become infeasible during learning?
  - ...

lecture 3

Lecture 3: Examples and Applications



README.md

Deep Declarative Networks

Running code

License

Exploiting Problem Structure in Deep Declarative Networks: Two Case Studies

Stephen Gould[1], Dylan Campbell[2], Itzik Ben-Shabat[1,3], Chamin Hewa Koneputugodage[1], Zhiwei Xu[1]

[1] Australian National University, Canberra, Australia
[2] University of Oxford, Oxford, United Kingdom
[3] Technion, Haifa, Israel

Abstract

Introduction

Solving the Blind Perspective-n-Point Problem End-To-End With Robust Differentiable Geometric Optimization

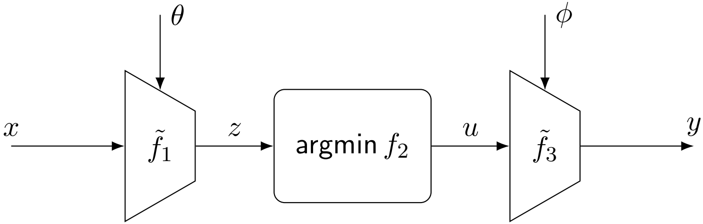
Dylan Campbell*, Liu Liu*, and Stephen Gould

Australian National University, Australian Centre for Robotic Vision**

1 Introduction

https://deepdeclarativenetworks.com

Common Theme



Differentiable Least Squares

Consider our old friend, the least-squares problem,

$$\text{minimize} \quad \|Ax - b\|_2^2$$

parameterized by A and b and with closed-form solution $x^\star = \left(A^TA\right)^{-1}A^Tb$.

Differentiable Least Squares

Consider our old friend, the least-squares problem,

$$\text{minimize} \quad \|Ax - b\|_2^2$$

parameterized by A and b and with closed-form solution $x^\star = \left(A^TA\right)^{-1}A^Tb$.

We are interested in derivatives of the solution with respect to the elements of A,

$$\frac{\mathrm{d}x^\star}{\mathrm{d}A_{ij}} = \frac{\mathrm{d}}{\mathrm{d}A_{ij}}\left(A^TA\right)^{-1}A^Tb \quad \in \mathbb{R}^n$$

We could also compute derivatives with respect to elements of b (but not here).

Least Squares Backward Pass

The backward pass combines $\frac{\mathrm{d}x^\star}{\mathrm{d}A_{ij}}$ with $v^T = \frac{\mathrm{d}L}{\mathrm{d}x^\star}$ via the vector-Jacobian product. After some algebraic manipulation (see lecture notes) we get

$$\left(\frac{\mathrm{d}L}{\mathrm{d}A}\right)^T = wr^T - x^\star(Aw)^T \quad \in \mathbb{R}^{m\times n}$$

where $w^T = v^T(A^TA)^{-1}$.

Least Squares Backward Pass

The backward pass combines $\frac{\mathrm{d}x^\star}{\mathrm{d}A_{ij}}$ with $v^T = \frac{\mathrm{d}L}{\mathrm{d}x^\star}$ via the vector-Jacobian product. After some algebraic manipulation (see lecture notes) we get

$$\left(\frac{\mathrm{d}L}{\mathrm{d}A}\right)^T = wr^T - x^\star(Aw)^T \quad \in \mathbb{R}^{m \times n}$$

where $w^T = v^T(A^TA)^{-1}$.

- $\left(A^TA\right)^{-1}$ is used in both the forward and backward pass
- factored once to solve for x, e.g., into $A = QR$
- cache R and re-use when computing gradients

derivation

Aside: PyTorch and Batched Data

Deep learning frameworks process data in batches, passed as tensors, for stochastic gradient descent. The first dimension of the tensor is the batch dimension.

Example. For the operation $y = Ax + b$ we might have

$$X = \{x^{(1)}, \ldots, x^{(K)}\} \qquad \text{(input)}$$

$$Y = \{Ax^{(1)} + b, \ldots, Ax^{(K)} + b\} \qquad \text{(output)}$$

Many PyTorch functions are batch-aware, e.g., `torch.bmm`. For many operations the `einsum` function and broadcasting are particularly useful, e.g.,

```
y = torch.einsum("ij,kj->ki", A, x) + b
```

computes $y = Ax^{(k)} + b$ on each element $k = 1, \ldots, K$ of the batch.

PyTorch Implementation: Forward Pass

```python
class LeastSquaresFcn(torch.autograd.Function):
    """PyTorch autograd function for least squares."""

    @staticmethod
    def forward(ctx, A, b):
        B, M, N = A.shape
        assert b.shape == (B, M, 1)

        with torch.no_grad():
            Q, R = torch.linalg.qr(A, mode='reduced')
            x = torch.linalg.solve_triangular(R,
                torch.bmm(b.view(B, 1, M), Q).view(B, N, 1), upper=True)

        # save state for backward pass
        ctx.save_for_backward(A, b, x, R)

        # return solution
        return x
```

$$A = QR$$

$$x = R^{-1} \left(Q^T b\right)$$

(solves $Rx = Q^T b$)

PyTorch Implementation: Backward Pass

```python
    @staticmethod
    def backward(ctx, dx):
        # check for None tensors
        if dx is None:
            return None, None

        # unpack cached tensors
        A, b, x, R = ctx.saved_tensors
        B, M, N = A.shape

        dA, db = None, None

        w = torch.linalg.solve_triangular(R,
            torch.linalg.solve_triangular(torch.transpose(R, 2, 1),
            dx, upper=False), upper=True)
        Aw = torch.bmm(A, w)

        if ctx.needs_input_grad[0]:
            r = b - torch.bmm(A, x)
            dA = torch.einsum("bi,bj->bij", r.view(B,M), w.view(B,N)) - \
                torch.einsum("bi,bj->bij", Aw.view(B,M), x.view(B,N))
        if ctx.needs_input_grad[1]:
            db = Aw

        # return gradients
        return dA, db
```

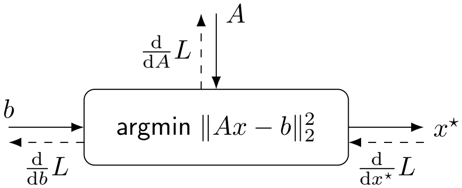
$$w = \left(A^T A\right)^{-1} v$$
$$= R^{-1} \left(R^{-T} v\right)$$
$$r = b - Ax$$
$$\left(\frac{\mathrm{d}L}{\mathrm{d}A}\right)^T = wr^T - x(Aw)^T$$
$$\left(\frac{\mathrm{d}L}{\mathrm{d}b}\right)^T = Aw$$

Example

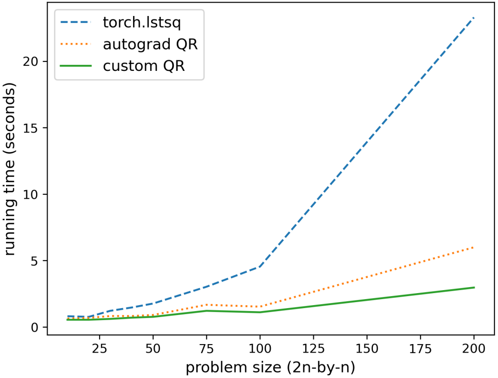
Bi-level optimisation problem with lower-level least squares:

$$\begin{array}{ll} \text{minimize} & \frac{1}{2}\|x^\star - x^{\text{target}}\|_2^2 \\ \text{subject to} & x^\star = \operatorname{argmin}_x \|Ax - b\|_2^2 \end{array}$$

with upper-level variable $A \in \mathbb{R}^{m \times n}$.

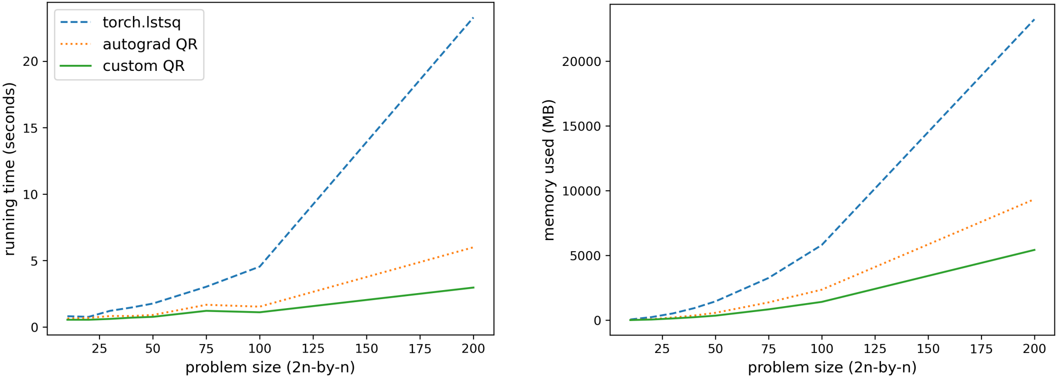


Profiling



(problems with $m = 2n$; run for 1000 iterations on CPU using PyTorch 1.13.0)

Profiling



(problems with $m = 2n$; run for 1000 iterations on CPU using PyTorch 1.13.0)

Optimal Transport

One view of optimal transport is as a matching problem

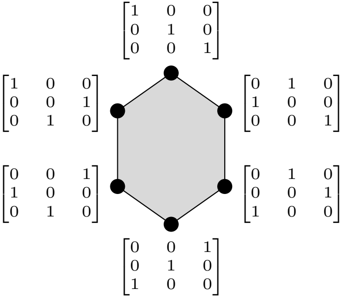
- from an m-by-n cost matrix M
- to an m-by-n probability matrix P,

often formulated with an entropic regularisation term,

$$\begin{aligned} \text{minimize} \quad & \langle M, P \rangle + \frac{1}{\gamma} \langle P, \log P \rangle \\ \text{subject to} \quad & P\mathbf{1} = r \\ & P^T \mathbf{1} = c \end{aligned}$$

with $\mathbf{1}^T r = \mathbf{1}^T c = 1$.

The row and column sum constraints ensure that P is a doubly stochastic matrix (lies within the convex hull of permutation matrices).



Solving Entropic Optimal Transport

Solution takes the form

$$P_{ij} = \alpha_i \beta_j e^{-\gamma M_{ij}}$$

and can be found using the Sinkhorn algorithm,

- Set $K_{ij} = e^{-\gamma M_{ij}}$ and $\alpha, \beta \in \mathbb{R}^n_{++}$
- Iterate until convergence,

$$\alpha \leftarrow r \oslash K\beta$$
$$\beta \leftarrow c \oslash K^T\alpha$$

  where $\oslash$ denotes componentwise division
- Return $P = \textbf{diag}(\alpha) K \textbf{diag}(\beta)$

Differentiable Optimal Transport

- Option 1: back-propagate through Sinkhorn algorithm

Differentiable Optimal Transport

- Option 1: back-propagate through Sinkhorn algorithm
- Option 2: use the implicit differentiation result

$$\underbrace{\frac{\mathrm{d}L}{\mathrm{d}M}}_{m\text{-by-}n} = \underbrace{\frac{\mathrm{d}L}{\mathrm{d}P}}_{m\text{-by-}n} \overbrace{\frac{\mathrm{d}P}{\mathrm{d}M}}^{m\text{-by-}n\text{-by-}m\text{-by-}n}$$

Differentiable Optimal Transport

- Option 1: back-propagate through Sinkhorn algorithm
- Option 2: use the implicit differentiation result

$$\underbrace{\frac{\mathrm{d}L}{\mathrm{d}M}}_{\text{1-by-}mn} = \underbrace{\frac{\mathrm{d}L}{\mathrm{d}P}}_{\text{1-by-}mn} \overbrace{\frac{\mathrm{d}P}{\mathrm{d}M}}^{mn\text{-by-}mn}$$

(think of vectorising M and P)

Optimal Transport Gradient

Derivation of the optimal transport gradient is quite tedious (see notes). The result:

$$\frac{\mathrm{d}L}{\mathrm{d}M} = \frac{\mathrm{d}L}{\mathrm{d}P}\left(H^{-1}A^T\left(AH^{-1}A^T\right)^{-1}AH^{-1} - H^{-1}\right)B$$
$$= \gamma\frac{\mathrm{d}L}{\mathrm{d}P}\textbf{diag}(P)\begin{bmatrix}A_1\\A_2\end{bmatrix}^T\begin{bmatrix}\Lambda_{11} & \Lambda_{12}\\\Lambda_{12}^T & \Lambda_{22}\end{bmatrix}\begin{bmatrix}A_1\\A_2\end{bmatrix}\textbf{diag}(P) - \gamma\frac{\mathrm{d}L}{\mathrm{d}P}\textbf{diag}(P)$$

where

$$\begin{bmatrix}A_1\\A_2\end{bmatrix} = \begin{bmatrix}\mathbf{0}_n^T & \mathbf{1}_n^T & \cdots & \mathbf{0}_n^T\\\vdots & \vdots & \ddots & \vdots\\\mathbf{0}_n^T & \mathbf{0}_n^T & \cdots & \mathbf{1}_n^T\\I_{n\times n} & I_{n\times n} & \cdots & I_{n\times n}\end{bmatrix}$$

$$\left(AH^{-1}A^T\right)^{-1} = \frac{1}{\gamma}\begin{bmatrix}\Lambda_{11} & \Lambda_{12}\\\Lambda_{12}^T & \Lambda_{22}\end{bmatrix} = \frac{1}{\gamma}\begin{bmatrix}\textbf{diag}(r_{2:m}) & P_{2:m,1:n}\\P_{2:m,1:n}^T & \textbf{diag}(c)\end{bmatrix}^{-1}$$

derivation

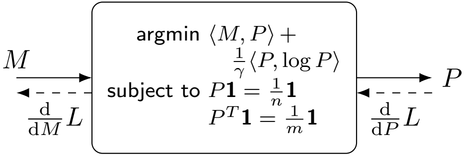
Implementation

```
@staticmethod
def backward(ctx, dJdP)
    # unpacked cached tensors
    M, r, c, P = ctx.saved_tensors
    batches, m, n = P.shape

    # initialize backward gradients (-v^T H^{-1} B)
    dLdM = -1.0 * gamma * P * dLdP

    # compute [vHAt1, vHAt2] = -v^T H^{-1} A^T
    vHAt1, vHAt2 = sum(dJdM[:, 1:m, 0:n], dim=2), sum(dJdM, dim=1)

    # compute [v1, v2] = -v^T H^{-1} A^T (A H^{-1] A^T)^{-1}
    P_over_c = P[:, 1:m, 0:n] / c.view(batches, 1, n)
    lmd_11 = cholesky(diag_embed(r[:, 1:m]) - einsum("bij,bkj->bik", P[:, 1:m, 0:n], P_over_c))
    lmd_12 = cholesky_solve(P_over_c, lmd_11)
    lmd_22 = diag_embed(1.0 / c) + einsum("bji,bjk->bik", lmd_12, P_over_c)

    v1 = cholesky_solve(vHAt1.view(batches, m-1, 1), lmd_11).view(batches, m-1) -
        einsum("bi,bji->bj", vHAt2, lmd_12)
    v2 = einsum("bi,bij->bj", vHAt2, lmd_22) - einsum("bi,bij->bj", vHAt1, lmd_12)

    # compute v^T H^{-1} A^T (A H^{-1] A^T)^{-1} A H^{-1} B - v^T H^{-1} B
    dLdM[:, 1:m, 0:n] -= v1.view(batches, m-1, 1) * P[:, 1:m, 0:n]
    dJdM -= v2.view(batches, 1, n) * P

    # return gradients
    return dJdM
```

Experiment

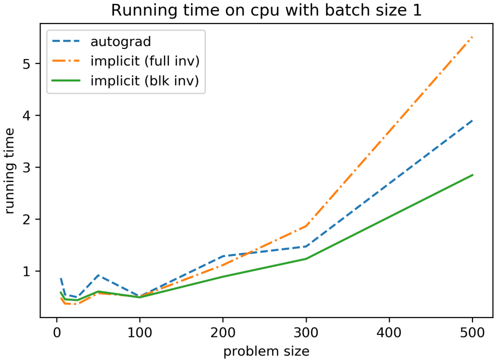
Bi-level optimisation problem with
lower-level optimal transport problem:

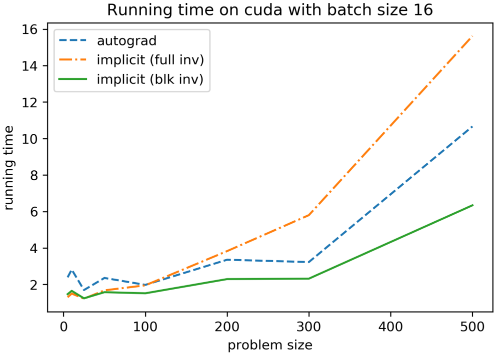
$$
\begin{aligned}
\text{minimize} \quad & \tfrac{1}{2}\|P - P^{\text{target}}\|_F^2 \\
\text{subject to} \quad & \text{minimize } \langle M, P\rangle + \tfrac{1}{\gamma}\langle P, \log P\rangle \\
& \text{subject to} \quad P\mathbf{1} = \tfrac{1}{n}\mathbf{1} \\
& \qquad\qquad\quad\;\; P^T\mathbf{1} = \tfrac{1}{m}\mathbf{1}
\end{aligned}
$$



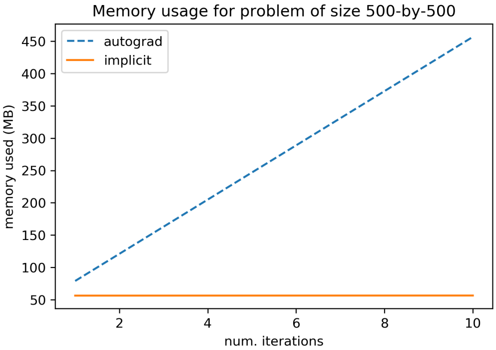
with upper-level variable $M \in \mathbb{R}^{m\times n}$.

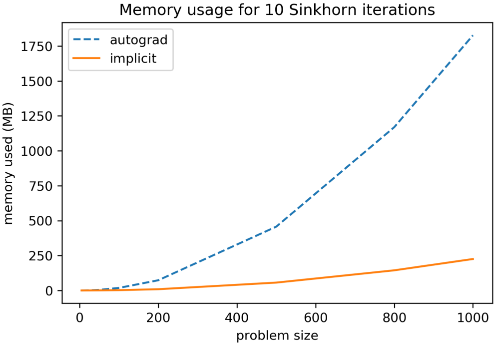
Results: Running Time





Results: Memory Usage





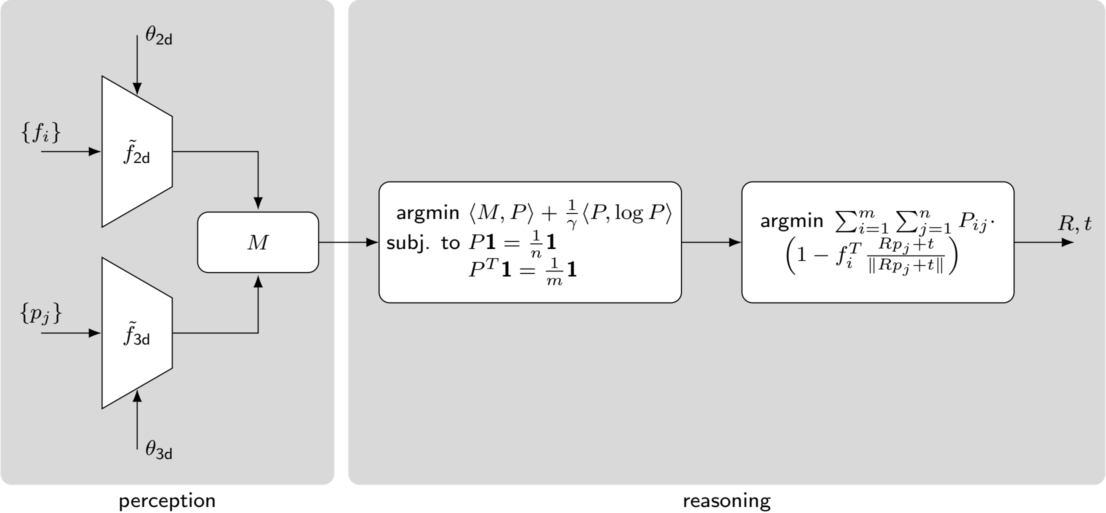
Application to Blind Perspective-n-Point

find the location where the photograph was taken

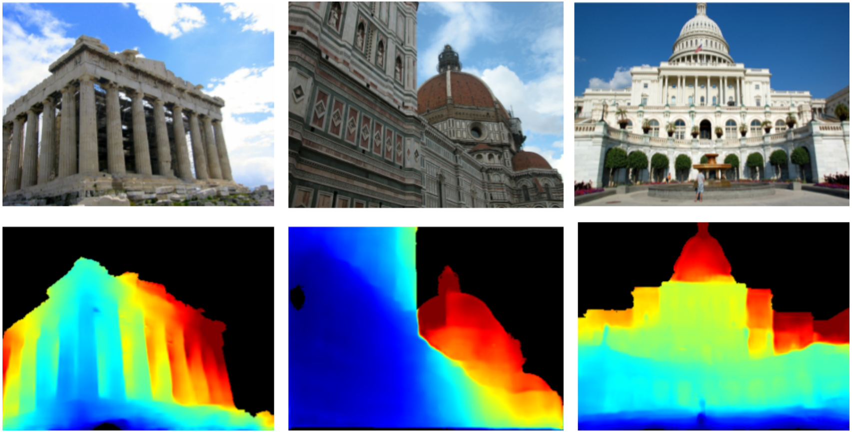
Coupled Problem

- if we knew **correspondences** then determining **camera pose** would be easy
- if we knew **camera pose** then determining **correspondences** would be easy

Blind Perspective-n-Point Network Architecture



Blind Perspective-n-Point Results



Further Resources

Where to from here?

- Deep declarative networks (`http://deepdeclarativenetworks.com`)
  - lots of small code examples and tutorials
- CVXPyLayers (`https://github.com/cvxgrp/cvxpylayers`)
- Theseus (`https://sites.google.com/view/theseus-ai`)
- JAXopt (`https://github.com/google/jaxopt`)

lecture notes available at `https://users.cecs.anu.edu.au/~sgould`

break-out

Local Optima are Global Optima Proof

back

any local minimum of a convex problem is (globally) optimal

Proof. Suppose that x is locally optimal, but there exists a feasible y with lower objective, i.e., $f_0(y) < f_0(x)$. Local optimality of x means there must be an $R > 0$ such that

$$z \text{ feasible and } \|z - x\|_2 \leq R \implies f_0(z) \geq f_0(x)$$

Consider $z = \theta y + (1-\theta)x$ with $\theta = \frac{R}{2\|y-x\|_2}$. We have that $\|y - x\|_2 > R$ since we assumed $f_0(y) < f_0(x)$, so $0 < \theta < 1/2 < 1$. Therefore z is a convex combination of two feasible points, hence also feasible. Moreover, $\|z - x\|_2 = R/2$ (from our choice of θ) and therefore $f_0(z) \geq f_0(x)$ by our assumption that x is locally optimal. But

$$\begin{aligned} f_0(z) &\leq \theta f_0(y) + (1-\theta) f_0(x) \\ &< \theta f_0(x) + (1-\theta) f_0(x) \\ &= f_0(x) \end{aligned}$$

where the first inequality is by the definition of convex function and the second inequality is from our assumption that $f_0(y) < f_0(x)$. We have a contradiction. Therefore every locally optimal point is globally optimal.

automatic differentiation

Toy Example: Babylonian Algorithm

back

Consider the following implementation for a forward operation:

```
procedure FwdFcn(x)
    y_0 ← (1/2) x
    for t = 1, ..., T do
        y_t ← (1/2) (y_{t-1} + x / y_{t-1})
    end for
    return y_T
end procedure
```

Toy Example: Babylonian Algorithm

back

Consider the following implementation for a forward operation:

procedure $\textsc{FwdFcn}(x)$
  $y_0 \leftarrow \frac{1}{2}x$
  for $t = 1, \ldots, T$ **do**
    $y_t \leftarrow \frac{1}{2}\left(y_{t-1} + \frac{x}{y_{t-1}}\right)$
  end for
  return y_T
end procedure

Automatic differentiation algorithmically generates the backward code:

procedure $\textsc{BckFcn}(x, y_T, \frac{\mathrm{d}L}{\mathrm{d}y_T})$
  $\frac{\mathrm{d}L}{\mathrm{d}x} \leftarrow 0$
  for $t = T, \ldots, 1$ **do**
    $\frac{\mathrm{d}L}{\mathrm{d}x} \leftarrow \frac{\mathrm{d}L}{\mathrm{d}x} + \frac{\mathrm{d}L}{\mathrm{d}y_t}\overbrace{\left(\frac{1}{2y_{t-1}}\right)}^{\partial y_t / \partial x}$
    $\frac{\mathrm{d}L}{\mathrm{d}y_{t-1}} \leftarrow \frac{\mathrm{d}L}{\mathrm{d}y_t}\underbrace{\left(\frac{1}{2} - \frac{x}{2y_{t-1}^2}\right)}_{\partial y_t / \partial y_{t-1}}$
  end for
  $\frac{\mathrm{d}L}{\mathrm{d}x} \leftarrow \frac{\mathrm{d}L}{\mathrm{d}x} + \frac{\mathrm{d}L}{\mathrm{d}y_0}\frac{1}{2}$
  return $\frac{\mathrm{d}L}{\mathrm{d}x}$
end procedure

Toy Example: Babylonian Algorithm back

Consider the following implementation for a forward operation:

```
procedure FwdFcn(x)
    y_0 ← (1/2) x
    for t = 1, ..., T do
        y_t ← (1/2) (y_{t-1} + x / y_{t-1})
    end for
    return y_T
end procedure
```

$$
\begin{array}{l}
\textbf{procedure } \textsc{FwdFcn}(x) \\
\quad y_0 \leftarrow \frac{1}{2} x \\
\quad \textbf{for } t = 1, \ldots, T \textbf{ do} \\
\quad\quad y_t \leftarrow \frac{1}{2}\left(y_{t-1} + \frac{x}{y_{t-1}}\right) \\
\quad \textbf{end for} \\
\quad \textbf{return } y_T \\
\textbf{end procedure}
\end{array}
$$

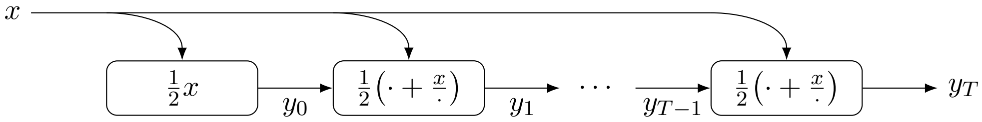
- computes $y = \sqrt{x}$
- derivative computed directly is $\frac{\mathrm{d}y}{\mathrm{d}x} = \frac{1}{2\sqrt{x}} = \frac{1}{2y}$

Automatic differentiation algorithmically generates the backward code:

$$
\begin{array}{l}
\textbf{procedure } \textsc{BckFcn}\left(x, y_T, \frac{\mathrm{d}L}{\mathrm{d}y_T}\right) \\
\quad \frac{\mathrm{d}L}{\mathrm{d}x} \leftarrow 0 \\
\quad \textbf{for } t = T, \ldots, 1 \textbf{ do} \\
\quad\quad \frac{\mathrm{d}L}{\mathrm{d}x} \leftarrow \frac{\mathrm{d}L}{\mathrm{d}x} + \frac{\mathrm{d}L}{\mathrm{d}y_t} \overbrace{\left(\frac{1}{2y_{t-1}}\right)}^{\partial y_t / \partial x} \\
\quad\quad \frac{\mathrm{d}L}{\mathrm{d}y_{t-1}} \leftarrow \frac{\mathrm{d}L}{\mathrm{d}y_t} \underbrace{\left(\frac{1}{2} - \frac{x}{2y_{t-1}^2}\right)}_{\partial y_t / \partial y_{t-1}} \\
\quad \textbf{end for} \\
\quad \frac{\mathrm{d}L}{\mathrm{d}x} \leftarrow \frac{\mathrm{d}L}{\mathrm{d}x} + \frac{\mathrm{d}L}{\mathrm{d}y_0} \frac{1}{2} \\
\quad \textbf{return } \frac{\mathrm{d}L}{\mathrm{d}x} \\
\textbf{end procedure}
\end{array}
$$

Computation Graph for Babylonian Algorithm

back



$$y_T = f(x, f(x, f(x, \ldots f(x, \tfrac{1}{2}x)))) \text{ with } f(x,y) = \frac{1}{2}\left(y + \frac{x}{y}\right)$$

duality

Lagrange Dual Function ⏩ back

Define Lagrange dual function, $g : \mathbb{R}^p \times \mathbb{R}^q \to \mathbb{R}$, as

$$g(\lambda, \nu) = \inf_{x \in \mathcal{D}} \mathcal{L}(x, \lambda, \nu)$$
$$= \inf_{x \in \mathcal{D}} \left(f_0(x) + \sum_{i=1}^{p} \lambda_i f_i(x) + \sum_{i=1}^{q} \nu_i h_i(x) \right)$$

- g is concave (always), can be $-\infty$ for some λ, ν
- **lower bound property:** if $\lambda \succeq 0$, then $g(\lambda, \nu) \leq p^\star$
  (since for feasible x we have $f_i(x) \leq 0$ and $h_i(x) = 0$)

The Dual Problem ⏩ back

The Lagrange dual problem is to maximise the dual function

$$\begin{array}{ll} \text{maximize} & g(\lambda, \nu) \\ \text{subject to} & \lambda \succeq 0 \end{array}$$

- finds the best lower bound on $p^\star$, obtained from Lagrange dual function
- a convex optimisation problem with optimal value denoted by $d^\star$
- λ, ν are dual feasible if $\lambda \succeq 0$ and $(\lambda, \nu) \in$ **dom** (g)
- original problem is known as the **primal problem**

Weak and Strong Duality

back

weak duality: $d^\star \leq p^\star$

- always holds (for convex and nonconvex problems)
- can be used to find nontrivial lower bounds for difficult problems

strong duality: $d^\star = p^\star$

- does not hold in general
- (usually) holds for convex problems
- conditions that guarantee strong duality on convex problems are called **constraint qualifications**

differentiating equality constrained problems

Abridged Derivation

back

Forming the Lagrangian at optimal y for fixed x we have

$$\mathcal{L}(x, y, \nu) = f(x, y) - \sum_{i=1}^{q} \nu_i h_i(x, y).$$

Since $\frac{\partial h(x,y)}{\partial y}$ is full rank we have that y is a regular point. Then there exists a ν such that the Lagrangian is stationary at the point (y, ν). Thus

$$\begin{bmatrix} \frac{\partial \mathcal{L}}{\partial Y}^T \\ \frac{\partial \mathcal{L}}{\partial \nu}^T \end{bmatrix} = \begin{bmatrix} \left(\frac{\partial f(x,y)}{\partial y} - \sum_{i=1}^{q} \nu_i \frac{\partial h_i(x,y)}{\partial y} \right)^T \\ h(x,y) \end{bmatrix} = \mathbf{0}_{m+q}$$

which we can differentiate with respect to x,

$$\frac{\mathrm{d}}{\mathrm{d}x} \begin{bmatrix} (\frac{\partial f(x,y)}{\partial y})^T - \sum_{i=1}^{q} \nu_i (\frac{\partial h_i(x,y)}{\partial y})^T \\ h(x,y) \end{bmatrix} = \mathbf{0}_{(m+q)\times n}$$

to get (after some re-arranging in matrix form)

$$\begin{bmatrix} \frac{\partial^2 f(x,y)}{\partial y^2} - \sum_{i=1}^{q} \nu_i \frac{\partial^2 h_i(x,y)}{\partial y^2} & -(\frac{\partial h(x,y)}{\partial y})^T \\ \frac{\partial h(x,y)}{\partial y} & \mathbf{0}_{q\times q} \end{bmatrix} \begin{bmatrix} \frac{\mathrm{d}y(x)}{\mathrm{d}x} \\ \frac{\mathrm{d}\nu(x)}{\mathrm{d}x} \end{bmatrix} = - \begin{bmatrix} \frac{\partial^2 f(x,y)}{\partial x \partial y} - \sum_{i=1}^{q} \nu_i \frac{\partial^2 h_i(x,y)}{\partial x \partial y} \\ \frac{\partial}{\partial x} h(x,y) \end{bmatrix}.$$

Abridged Derivation ⏩ back

Forming the Lagrangian at optimal y for fixed x we have

$$\mathcal{L}(x, y, \nu) = f(x, y) - \sum_{i=1}^{q} \nu_i h_i(x, y).$$

Since $\frac{\partial h(x,y)}{\partial y}$ is full rank we have that y is a regular point. Then there exists a ν such that the Lagrangian is stationary at the point (y, ν). Thus

$$\begin{bmatrix} \frac{\partial \mathcal{L}}{\partial Y}^T \\ \frac{\partial \mathcal{L}}{\partial \nu}^T \end{bmatrix} = \begin{bmatrix} \left(\frac{\partial f(x,y)}{\partial y} - \sum_{i=1}^{q} \nu_i \frac{\partial h_i(x,y)}{\partial y}\right)^T \\ h(x, y) \end{bmatrix} = \mathbf{0}_{m+q}$$

which we can differentiate with respect to x,

$$\frac{\mathrm{d}}{\mathrm{d}x} \begin{bmatrix} (\frac{\partial f(x,y)}{\partial y})^T - \sum_{i=1}^{q} \nu_i (\frac{\partial h_i(x,y)}{\partial y})^T \\ h(x, y) \end{bmatrix} = \mathbf{0}_{(m+q)\times n}$$

to get (after some re-arranging in matrix form)

$$\begin{bmatrix} H & -A^T \\ A & \mathbf{0}_{q\times q} \end{bmatrix} \begin{bmatrix} \frac{\mathrm{d}y(x)}{\mathrm{d}x} \\ \frac{\mathrm{d}\nu(x)}{\mathrm{d}x} \end{bmatrix} = -\begin{bmatrix} B \\ C \end{bmatrix}.$$

Abridged Derivation (cont.) ⏩ back

(from last slide:)

$$\begin{bmatrix} H & -A^T \\ A & \mathbf{0}_{q\times q} \end{bmatrix} \begin{bmatrix} \frac{\mathrm{d}y(x)}{\mathrm{d}x} \\ \frac{\mathrm{d}\nu(x)}{\mathrm{d}x} \end{bmatrix} = - \begin{bmatrix} B \\ C \end{bmatrix}$$

We can solve this system of equations directly or solve by variable elimination. Multiplying out we have

$$H\frac{\mathrm{d}y(x)}{\mathrm{d}x} - A^T\frac{\mathrm{d}\nu(x)}{\mathrm{d}x} = -B \qquad (1)$$

$$A\frac{\mathrm{d}y(x)}{\mathrm{d}x} = -C \qquad (2)$$

Substituting $\frac{\mathrm{d}y(x)}{\mathrm{d}x}$ from (1) into (2) gives,

$$A\overbrace{H^{-1}(A^T\frac{\mathrm{d}\nu(x)}{\mathrm{d}x} - B)}^{\frac{\mathrm{d}y(x)}{\mathrm{d}x}} = -C$$

$$\therefore \quad \frac{\mathrm{d}\nu(x)}{\mathrm{d}x} = \left(AH^{-1}A^T\right)^{-1}\left(AH^{-1}B - C\right)$$

Then substituting back into (1) we get the result

$$\frac{\mathrm{d}y(x)}{\mathrm{d}x} = H^{-1}A^T\underbrace{\left(AH^{-1}A^T\right)^{-1}\left(AH^{-1}B - C\right)}_{\frac{\mathrm{d}\nu(x)}{\mathrm{d}x}} - H^{-1}B$$

least squares

Least Squares Backward Pass Derivation back

Differentiating $x^\star$ with respect to single element A_{ij}, we have

$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}A_{ij}} x^\star &= \frac{\mathrm{d}}{\mathrm{d}A_{ij}} \left(A^T A\right)^{-1} A^T b \\ &= \left(\frac{\mathrm{d}}{\mathrm{d}A_{ij}} \left(A^T A\right)^{-1}\right) A^T b + \left(A^T A\right)^{-1} \left(\frac{\mathrm{d}}{\mathrm{d}A_{ij}} A^T b\right)\end{aligned}$$

Using the identity $\frac{\mathrm{d}}{\mathrm{d}z} Z^{-1} = -Z^{-1} \left(\frac{\mathrm{d}}{\mathrm{d}z} Z\right) Z^{-1}$ we get, for the first term,

$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}A_{ij}} \left(A^T A\right)^{-1} &= -\left(A^T A\right)^{-1} \left(\frac{\mathrm{d}}{\mathrm{d}A_{ij}} \left(A^T A\right)\right) \left(A^T A\right)^{-1} \\ &= -\left(A^T A\right)^{-1} \left(E_{ij}^T A + A^T E_{ij}\right) \left(A^T A\right)^{-1}\end{aligned}$$

where E_{ij} is a matrix with one in the (i, j)-th element and zeros elsewhere. Furthermore, for the second term,

$$\frac{\mathrm{d}}{\mathrm{d}A_{ij}} A^T b = E_{ij}^T b$$

Least Squares Backward Pass Derivation (cont.) ⏩ back

Plugging these back into parent equation we have

$$\begin{aligned}
\frac{\mathrm{d}}{\mathrm{d}A_{ij}} x^\star &= -\left(A^T A\right)^{-1} \left(E_{ij}^T A + A^T E_{ij}\right) \left(A^T A\right)^{-1} A^T b + \left(A^T A\right)^{-1} E_{ij}^T b \\
&= -\left(A^T A\right)^{-1} \left(E_{ij}^T A + A^T E_{ij}\right) x^\star + \left(A^T A\right)^{-1} E_{ij}^T b \\
&= -\left(A^T A\right)^{-1} \left(E_{ij}^T (A x^\star - b) + A^T E_{ij} x^\star\right) \\
&= -\left(A^T A\right)^{-1} \left((a_i^T x^\star - b_i) e_j + x_j^\star a_i\right)
\end{aligned}$$

where $e_j = (0, 0, \ldots, 1, 0, \ldots) \in \mathbb{R}^n$ is the j-th canonical vector, i.e., vector with a one in the j-th component and zeros everywhere else, and $a_i^T \in \mathbb{R}^{1 \times n}$ is the i-th row of matrix A.

Least Squares Backward Pass Derivation (cont.) back

Let $r = b - Ax^\star$ and let v^T denote the backward coming gradient $\frac{\mathrm{d}}{\mathrm{d}x^\star} L$. Then

$$\begin{aligned}\frac{\mathrm{d}L}{\mathrm{d}A_{ij}} &= v^T \frac{\mathrm{d}x^\star}{\mathrm{d}A_{ij}} \\ &= v^T \left(A^T A\right)^{-1} \left(r_i e_j - x_j^\star a_i\right) \\ &= w^T \left(r_i e_j - x_j^\star a_i\right) \\ &= r_i w_j - w^T a_i x_j^\star\end{aligned}$$

where $w = \left(A^T A\right)^{-1} v$. We can compute the entire matrix of $m \times n$ derivatives efficiently as the sum of outer products

$$\left(\frac{\mathrm{d}L}{\mathrm{d}A}\right)^T = \left[\frac{\mathrm{d}L}{\mathrm{d}A_{ij}}\right]_{\substack{i=1,\ldots,m \\ j=1,\ldots,n}} = w r^T - x^\star (Aw)^T$$

optimal transport

Objective and Constraint Functions

back

$$f(M,P) = \sum_{i=1}^{m}\sum_{j=1}^{n} M_{ij}P_{ij} + \frac{1}{\gamma}\sum_{i=1}^{m}\sum_{j=1}^{n} P_{ij}\log P_{ij}$$

$$h(M,P) = \begin{bmatrix} \mathbf{1}_n^T & \mathbf{0}_n^T & \cdots & \mathbf{0}_n^T \\ \mathbf{0}_n^T & \mathbf{1}_n^T & \cdots & \mathbf{0}_n^T \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0}_n^T & \mathbf{0}_n^T & \cdots & \mathbf{1}_n^T \\ I_{n\times n} & I_{n\times n} & \cdots & I_{n\times n} \end{bmatrix} \begin{bmatrix} P_{11} \\ P_{12} \\ \vdots \\ P_{1n} \\ P_{21} \\ \vdots \\ P_{mn} \end{bmatrix} - \begin{bmatrix} r_1 \\ r_2 \\ \vdots \\ r_m \\ c_1 \\ \vdots \\ c_n \end{bmatrix}$$

(one constraint is redundant—a linear combination of the others—and removed to ensure **rank**$(A) = q$)

Deriving the Gradient ⏩ back

$$f(M,P) = \sum_{i=1}^{m}\sum_{j=1}^{n} M_{ij}P_{ij} + \frac{1}{\gamma}\sum_{i=1}^{m}\sum_{j=1}^{n} P_{ij}\log P_{ij}$$

$$h(M,P) = \begin{bmatrix} \mathbf{0}_n^T & \mathbf{1}_n^T & \dots & \mathbf{0}_n^T \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0}_n^T & \mathbf{0}_n^T & \dots & \mathbf{1}_n^T \\ I_{n\times n} & I_{n\times n} & \dots & I_{n\times n} \end{bmatrix} \vec{P} - \begin{bmatrix} r_2 \\ \vdots \\ r_m \\ c \end{bmatrix}$$

Deriving the Gradient ⏩ back

$$f(M,P) = \sum_{i=1}^{m}\sum_{j=1}^{n} M_{ij}P_{ij} + \frac{1}{\gamma}\sum_{i=1}^{m}\sum_{j=1}^{n} P_{ij}\log P_{ij}$$

$$h(M,P) = \begin{bmatrix} \mathbf{0}_n^T & \mathbf{1}_n^T & \dots & \mathbf{0}_n^T \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0}_n^T & \mathbf{0}_n^T & \dots & \mathbf{1}_n^T \\ I_{n\times n} & I_{n\times n} & \dots & I_{n\times n} \end{bmatrix} \vec{P} - \begin{bmatrix} r_2 \\ \vdots \\ r_m \\ c \end{bmatrix}$$

$$\frac{\mathrm{d}P}{\mathrm{d}M} = \left(H^{-1}A^T\left(AH^{-1}A^T\right)^{-1}AH^{-1} - H^{-1}\right)B$$

$$A = \frac{\mathrm{d}}{\mathrm{d}P}h \in \mathbb{R}^{(m+n-1)\times mn} \qquad B = \frac{\mathrm{d}^2}{\mathrm{d}M\partial P}f \in \mathbb{R}^{mn\times nn} \qquad H = \frac{\mathrm{d}^2}{\mathrm{d}P^2}f \in \mathbb{R}^{mn\times mn}$$

Deriving the Gradient

back

$$f(M,P) = \sum_{i=1}^{m}\sum_{j=1}^{n} M_{ij}P_{ij} + \frac{1}{\gamma}\sum_{i=1}^{m}\sum_{j=1}^{n} P_{ij}\log P_{ij}$$

$$h(M,P) = \begin{bmatrix} \mathbf{0}_n^T & \mathbf{1}_n^T & \dots & \mathbf{0}_n^T \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0}_n^T & \mathbf{0}_n^T & \dots & \mathbf{1}_n^T \\ I_{n\times n} & I_{n\times n} & \dots & I_{n\times n} \end{bmatrix} \vec{P} - \begin{bmatrix} r_2 \\ \vdots \\ r_m \\ c \end{bmatrix}$$

$$\frac{\mathrm{d}P}{\mathrm{d}M} = \left(H^{-1}A^T\left(AH^{-1}A^T\right)^{-1}AH^{-1} - H^{-1}\right)B$$

$$A = \frac{\mathrm{d}}{\mathrm{d}P}h \in \mathbb{R}^{(m+n-1)\times mn} = \begin{bmatrix} \mathbf{0}_n^T & \mathbf{1}_n^T & \dots & \mathbf{0}_n^T \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0}_n^T & \mathbf{0}_n^T & \dots & \mathbf{1}_n^T \\ I_{n\times n} & I_{n\times n} & \dots & I_{n\times n} \end{bmatrix}$$

$$B = \frac{\mathrm{d}^2}{\mathrm{d}M\partial P}f \in \mathbb{R}^{mn\times nn} \qquad H = \frac{\mathrm{d}^2}{\mathrm{d}P^2}f \in \mathbb{R}^{mn\times mn}$$

Deriving the Gradient ⏩ back

$$f(M,P) = \sum_{i=1}^{m}\sum_{j=1}^{n} M_{ij}P_{ij} + \frac{1}{\gamma}\sum_{i=1}^{m}\sum_{j=1}^{n} P_{ij}\log P_{ij} \qquad h(M,P) = \begin{bmatrix} \mathbf{0}_n^T & \mathbf{1}_n^T & \dots & \mathbf{0}_n^T \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0}_n^T & \mathbf{0}_n^T & \dots & \mathbf{1}_n^T \\ I_{n\times n} & I_{n\times n} & \dots & I_{n\times n} \end{bmatrix} \vec{P} - \begin{bmatrix} r_2 \\ \vdots \\ r_m \\ c \end{bmatrix}$$

$$\frac{\mathrm{d}P}{\mathrm{d}M} = \left(H^{-1}A^T\left(AH^{-1}A^T\right)^{-1}AH^{-1} - H^{-1}\right)B$$

$$A = \frac{\mathrm{d}}{\mathrm{d}P}h \in \mathbb{R}^{(m+n-1)\times mn} \qquad B = \frac{\mathrm{d}^2}{\mathrm{d}M\partial P}f \in \mathbb{R}^{mn\times nn} \qquad H = \frac{\mathrm{d}^2}{\mathrm{d}P^2}f \in \mathbb{R}^{mn\times mn}$$

$$= \begin{bmatrix} \mathbf{0}_n^T & \mathbf{1}_n^T & \dots & \mathbf{0}_n^T \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0}_n^T & \mathbf{0}_n^T & \dots & \mathbf{1}_n^T \\ I_{n\times n} & I_{n\times n} & \dots & I_{n\times n} \end{bmatrix} \qquad B_{ij,kl} = \begin{cases} 1 & \text{if } ij = kl \\ 0 & \text{otherwise} \end{cases}$$

Deriving the Gradient back

$$f(M,P) = \sum_{i=1}^{m}\sum_{j=1}^{n} M_{ij}P_{ij} + \frac{1}{\gamma}\sum_{i=1}^{m}\sum_{j=1}^{n} P_{ij}\log P_{ij} \qquad h(M,P) = \begin{bmatrix} \mathbf{0}_n^T & \mathbf{1}_n^T & \dots & \mathbf{0}_n^T \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0}_n^T & \mathbf{0}_n^T & \dots & \mathbf{1}_n^T \\ I_{n\times n} & I_{n\times n} & \dots & I_{n\times n} \end{bmatrix} \vec{P} - \begin{bmatrix} r_2 \\ \vdots \\ r_m \\ c \end{bmatrix}$$

$$\frac{\mathrm{d}P}{\mathrm{d}M} = \left(H^{-1}A^T\left(AH^{-1}A^T\right)^{-1}AH^{-1} - H^{-1}\right)B$$

$$A = \frac{\mathrm{d}}{\mathrm{d}P}h \in \mathbb{R}^{(m+n-1)\times mn} \qquad B = \frac{\mathrm{d}^2}{\mathrm{d}M\partial P}f \in \mathbb{R}^{mn\times nn} \qquad H = \frac{\mathrm{d}^2}{\mathrm{d}P^2}f \in \mathbb{R}^{mn\times mn}$$

$$= \begin{bmatrix} \mathbf{0}_n^T & \mathbf{1}_n^T & \dots & \mathbf{0}_n^T \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0}_n^T & \mathbf{0}_n^T & \dots & \mathbf{1}_n^T \\ I_{n\times n} & I_{n\times n} & \dots & I_{n\times n} \end{bmatrix} \qquad = I_{mn\times mn}$$

Deriving the Gradient ⏩ back

$$f(M,P) = \sum_{i=1}^{m}\sum_{j=1}^{n} M_{ij}P_{ij} + \frac{1}{\gamma}\sum_{i=1}^{m}\sum_{j=1}^{n} P_{ij}\log P_{ij}$$

$$h(M,P) = \begin{bmatrix} \mathbf{0}_n^T & \mathbf{1}_n^T & \cdots & \mathbf{0}_n^T \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0}_n^T & \mathbf{0}_n^T & \cdots & \mathbf{1}_n^T \\ I_{n\times n} & I_{n\times n} & \cdots & I_{n\times n} \end{bmatrix} \vec{P} - \begin{bmatrix} r_2 \\ \vdots \\ r_m \\ c \end{bmatrix}$$

$$\frac{\mathrm{d}P}{\mathrm{d}M} = \left(H^{-1}A^T\left(AH^{-1}A^T\right)^{-1}AH^{-1} - H^{-1}\right)B$$

$$A = \frac{\mathrm{d}}{\mathrm{d}P}h \in \mathbb{R}^{(m+n-1)\times mn} = \begin{bmatrix} \mathbf{0}_n^T & \mathbf{1}_n^T & \cdots & \mathbf{0}_n^T \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0}_n^T & \mathbf{0}_n^T & \cdots & \mathbf{1}_n^T \\ I_{n\times n} & I_{n\times n} & \cdots & I_{n\times n} \end{bmatrix}$$

$$B = \frac{\mathrm{d}^2}{\mathrm{d}M\partial P}f \in \mathbb{R}^{mn\times nn} = I_{mn\times mn}$$

$$H = \frac{\mathrm{d}^2}{\mathrm{d}P^2}f \in \mathbb{R}^{mn\times mn} \qquad H_{ij,kl} = \begin{cases} \frac{1}{\gamma P_{ij}} & \text{if } ij = kl \\ 0 & \text{otherwise} \end{cases}$$

Deriving the Gradient ⏩ back

$$f(M,P) = \sum_{i=1}^{m}\sum_{j=1}^{n} M_{ij}P_{ij} + \frac{1}{\gamma}\sum_{i=1}^{m}\sum_{j=1}^{n} P_{ij}\log P_{ij}$$

$$h(M,P) = \begin{bmatrix} \mathbf{0}_n^T & \mathbf{1}_n^T & \cdots & \mathbf{0}_n^T \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0}_n^T & \mathbf{0}_n^T & \cdots & \mathbf{1}_n^T \\ I_{n\times n} & I_{n\times n} & \cdots & I_{n\times n} \end{bmatrix} \vec{P} - \begin{bmatrix} r_2 \\ \vdots \\ r_m \\ c \end{bmatrix}$$

$$\frac{\mathrm{d}P}{\mathrm{d}M} = \left(H^{-1}A^T\left(AH^{-1}A^T\right)^{-1}AH^{-1} - H^{-1}\right)B$$

$$A = \frac{\mathrm{d}}{\mathrm{d}P}h \in \mathbb{R}^{(m+n-1)\times mn}$$

$$= \begin{bmatrix} \mathbf{0}_n^T & \mathbf{1}_n^T & \cdots & \mathbf{0}_n^T \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0}_n^T & \mathbf{0}_n^T & \cdots & \mathbf{1}_n^T \\ I_{n\times n} & I_{n\times n} & \cdots & I_{n\times n} \end{bmatrix}$$

$$B = \frac{\mathrm{d}^2}{\mathrm{d}M\partial P}f \in \mathbb{R}^{mn\times nn}$$

$$= I_{mn\times mn}$$

$$H = \frac{\mathrm{d}^2}{\mathrm{d}P^2}f \in \mathbb{R}^{mn\times mn}$$

$$H^{-1} = \gamma\,\mathbf{diag}\left(\vec{P}\right)$$

Computing $(AH^{-1}A^T)^{-1}$ back

$$H^{-1} = \gamma \mathbf{diag}\left(\vec{P}\right) \qquad A = \begin{bmatrix} \mathbf{0}_n^T & \mathbf{1}_n^T & \cdots & \mathbf{0}_n^T \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0}_n^T & \mathbf{0}_n^T & \cdots & \mathbf{1}_n^T \\ I_{n\times n} & I_{n\times n} & \cdots & I_{n\times n} \end{bmatrix}$$

$$\frac{\mathrm{d}P}{\mathrm{d}M} = \left(H^{-1}A^T\left(AH^{-1}A^T\right)^{-1}AH^{-1} - H^{-1}\right)B$$

Computing $(AH^{-1}A^T)^{-1}$ back

$$H^{-1} = \gamma \mathbf{diag}\left(\vec{P}\right) \qquad A = \begin{bmatrix} \mathbf{0}_n^T & \mathbf{1}_n^T & \cdots & \mathbf{0}_n^T \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0}_n^T & \mathbf{0}_n^T & \cdots & \mathbf{1}_n^T \\ I_{n\times n} & I_{n\times n} & \cdots & I_{n\times n} \end{bmatrix}$$

$$\frac{\mathrm{d}P}{\mathrm{d}M} = \left(H^{-1}A^T\left(AH^{-1}A^T\right)^{-1}AH^{-1} - H^{-1}\right)B$$

The (k,l)-th entry of $AH^{-1}A^T$ for $k,l \in 1,\ldots,m+n-1$ is

$$(AH^{-1}A^T)_{kl} = \sum_{i=1}^{m}\sum_{j=1}^{n}\frac{A_{k,ij}A_{l,ij}}{H_{ij,ij}} = \gamma\sum_{i=1}^{m}\sum_{j=1}^{n}A_{k,ij}A_{l,ij}P_{ij}$$

Interpreting $A_{k,ij}A_{l,ij}$ back

$$
\begin{array}{c}
k \\ \\ l \\ \\
\end{array}
\begin{bmatrix}
\mathbf{0}_n^T & \mathbf{1}_n^T & \cdots & \mathbf{0}_n^T \\
\vdots & \vdots & \ddots & \vdots \\
\mathbf{0}_n^T & \mathbf{0}_n^T & \cdots & \mathbf{1}_n^T \\
I_{n\times n} & I_{n\times n} & \cdots & I_{n\times n}
\end{bmatrix}
\qquad
\begin{array}{c}
k \\ \\ \\ l
\end{array}
\begin{bmatrix}
\mathbf{0}_n^T & \mathbf{1}_n^T & \cdots & \mathbf{0}_n^T \\
\vdots & \vdots & \ddots & \vdots \\
\mathbf{0}_n^T & \mathbf{0}_n^T & \cdots & \mathbf{1}_n^T \\
I_{n\times n} & I_{n\times n} & \cdots & I_{n\times n}
\end{bmatrix}
\left.\begin{matrix} \uparrow \\ m-1 \\ \downarrow \\ \uparrow \\ n \\ \downarrow \end{matrix}\right\}
$$

$$
\underbrace{
\begin{array}{c}
l \\ \\ \\ k
\end{array}
\begin{bmatrix}
\mathbf{0}_n^T & \mathbf{1}_n^T & \cdots & \mathbf{0}_n^T \\
\vdots & \vdots & \ddots & \vdots \\
\mathbf{0}_n^T & \mathbf{0}_n^T & \cdots & \mathbf{1}_n^T \\
I_{n\times n} & I_{n\times n} & \cdots & I_{n\times n}
\end{bmatrix}
}_{\longleftarrow \quad mn \quad \longrightarrow}
\qquad
\underbrace{
\begin{array}{c}
\\ \\ \\ k \\ l
\end{array}
\begin{bmatrix}
\mathbf{0}_n^T & \mathbf{1}_n^T & \cdots & \mathbf{0}_n^T \\
\vdots & \vdots & \ddots & \vdots \\
\mathbf{0}_n^T & \mathbf{0}_n^T & \cdots & \mathbf{1}_n^T \\
I_{n\times n} & I_{n\times n} & \cdots & I_{n\times n}
\end{bmatrix}
}_{\longleftarrow \quad mn \quad \longrightarrow}
$$

Evaluating $(AH^{-1}A^T)_{kl} = \gamma \sum_{i=1}^{m} \sum_{j=1}^{n} A_{k,ij} A_{l,ij} P_{ij}$ back

	$0 \leq l \leq m-1$	$m \leq l \leq m+n-1$
$0 \leq k \leq m-1$	$\begin{cases} \gamma \sum_{j=1}^{n} P_{k+1,j} & \text{if } k = l \\ 0 & \text{otherwise} \end{cases}$	$\gamma P_{k+1,l-m+1}$
$m \leq k \leq m+n-1$	$\gamma P_{l+1,k-m+1}$	$\begin{cases} \gamma \sum_{i=1}^{m} P_{i,k-m+1} & \text{if } k = l \\ 0 & \text{otherwise} \end{cases}$

Computing $(AH^{-1}A^T)^{-1}$ back

$$H^{-1} = \gamma \mathbf{diag}\left(\vec{P}\right) \qquad A = \begin{bmatrix} \mathbf{0}_n^T & \mathbf{1}_n^T & \cdots & \mathbf{0}_n^T \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0}_n^T & \mathbf{0}_n^T & \cdots & \mathbf{1}_n^T \\ I_{n\times n} & I_{n\times n} & \cdots & I_{n\times n} \end{bmatrix}$$

$$\frac{\mathrm{d}P}{\mathrm{d}M} = \left(H^{-1}A^T \left(AH^{-1}A^T\right)^{-1} AH^{-1} - H^{-1}\right) B$$

$$AH^{-1}A^T = \gamma \begin{bmatrix} \mathbf{diag}(r_{2:m}) & P_{2:m,1:n} \\ P_{2:m,1:n}^T & \mathbf{diag}(c) \end{bmatrix} \qquad \left(AH^{-1}A^T\right)^{-1} = \frac{1}{\gamma} \begin{bmatrix} \Lambda_{11} & \Lambda_{12} \\ \Lambda_{12}^T & \Lambda_{22} \end{bmatrix}$$

$$\Lambda_{11} = \left(\mathbf{diag}\left(r_{2:m} - P_{2:m,1:n}\mathbf{diag}(c)^{-1} P_{2:m,1:n}^T\right)\right)^{-1}$$

$$\Lambda_{12} = -\Lambda_{11} P_{2:m,1:n}\mathbf{diag}(c)^{-1}$$

$$\Lambda_{22} = \mathbf{diag}(c)^{-1} - \mathbf{diag}(c)^{-1} P_{2:m,1:n}^T \Lambda_{12}$$

end