Machine Learning Basics

Instructors: Babak n. Arabi, Mohammadreza A. Dehaqani, Mostafa Tavassolipour

Parsa Daghig, Samar Nikfarjad

Fall 2025

Homework 2

Question 1: Line Search Optimization Problem

(a) Write the general form of the optimization problem that line-search algorithms solve, and briefly explain the roles of the **search direction** and the **step length**.

(b) Define a **descent direction** in the context of line search. State the necessary condition for a vector p to be a descent direction at x.

(c) List at least three common methods to determine the step length α in line-search algorithms. For each method give a short explanation (e.g., exact line search, Armijo/backtracking, Wolfe conditions, fixed step, Barzilai–Borwein).

(d) Consider the quadratic function

$$f(x_1, x_2) = 2x_1^2 + x_1x_2 + 4x_2^2.$$

At iteration t, the current point and search direction are

$$x^{(t)} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \qquad p^{(t)} = \begin{pmatrix} 1 \\ -2 \end{pmatrix}.$$

1. Show that $p^{(t)}$ is a descent direction at $x^{(t)}$, i.e. show that $\nabla f(x^{(t)})^\top p^{(t)} < 0$.
2. Form the one-dimensional function $\phi(\alpha) = f\big(x^{(t)} + \alpha p^{(t)}\big)$ and simplify it to exhibit its quadratic form.
3. Compute $\alpha^* > 0$ that minimizes $\phi(\alpha)$ (exact line search). State whether $\alpha^* > 0$.

Question 2: Support Vector Machine (SVM) — Primal, Lagrangian, and Dual (Optional)

Consider a binary classification dataset $\{(x_i, y_i)\}_{i=1}^n$ with $x_i \in \mathbb{R}^d$ and $y_i \in \{+1, -1\}$.

(a) Write the *hard-margin SVM* primal optimization problem.

(b) (Lagrangian & stationarity) Form the Lagrangian with multipliers $\alpha_i \geq 0$, derive $\nabla_w \mathcal{L} = 0$ and $\partial \mathcal{L}/\partial b = 0$.

(c) (Dual) Derive the dual problem (objective and constraints in α only).

(d) (KKT & support vectors) State KKT complementary slackness and explain support vectors. Explain how to recover (w, b) from $\alpha^\star$.

(e) (Soft margin) Give the *soft-margin* primal with slack variables and $C > 0$, state its dual, and the bounds on α_i.

(f) (Kernelization) Explain how the dual enables kernels and write the kernelized decision function.

Question 3: Bilevel Optimization

Introduction (short). Bilevel optimization models a leader choosing variables anticipating the follower's optimal response. Below is a simple scalar example solvable analytically:

The leader chooses $x \in \mathbb{R}$ to

$$\min_x \; F(x,y) = (x-2)^2 + (y-3)^2 \quad \text{s.t. } x \geq 0,$$

while the follower, given x, chooses $y \in \mathbb{R}$ to

$$\min_y \; f(x,y) = (y-x^2)^2 + y \quad \text{s.t. } y \geq 0.$$

(a) Write the bilevel formulation (upper and lower).

(b) For fixed x, solve the follower and obtain $y^*(x)$.

(c) Substitute $y^*(x)$ into the leader objective and reduce to one variable.

(d) Find (x^*, y^*).

(e) Discuss regularity (convexity of follower, uniqueness, validity of substitution).

Question 4: Multi-Objective Optimization — Pareto Front (Optional)

Consider $(x,y) \in \mathbb{R}^2$ with objectives

$$f_1(x,y) = (x-2)^2 + y^2, \qquad f_2(x,y) = x^2 + (y-3)^2.$$

(a) Define Pareto optimality for (x^*, y^*).

(b) Weighted-sum scalarization: for $w \in [0,1]$ set

$$F_w(x,y) = w f_1(x,y) + (1-w) f_2(x,y).$$

1. Derive $(x(w), y(w))$ minimizing F_w.
2. Show that as w varies, these trace the Pareto set.

(c) Give a parametric expression for the Pareto front $\mathcal{P} = \{(f_1(x(w),y(w)),\ f_2(x(w),y(w))) : w \in [0,1]\}$ and a closed relation between $u = f_1$ and $v = f_2$.

(d) (Epsilon-constraint and KKT) Minimize f_1 subject to $f_2 \leq t$; relate the multiplier λ to w and conclude how this maps to a Pareto point.

(e) Compute $(x(w), y(w))$ and (f_1, f_2) for $w \in \{0, 0.25, 0.5, 0.75, 1\}$ (rounded to 4 decimals).

Question 5: Genetic Algorithm — Jigsaw Puzzle Solver

Create a program that automatically solves a jigsaw puzzle using a Genetic Algorithm (GA).

(a) **Define a fit score.** Design a function that numerically measures how well two puzzle pieces fit together along their edges.

(b) **Evaluate a whole puzzle.** Develop a method to compute an overall fitness score for any complete arrangement of all pieces, assigning higher scores to better assemblies.

(c) **Survival of the fittest.** Implement a selection mechanism so that higher-fitness arrangements are more likely to be chosen to produce the next generation.

(d) **Evolutionary engine.** Combine initialization, selection, crossover/mutation (as appropriate), and replacement into a GA that starts from random arrangements and evolves over many generations toward the correct solution.

Project repository: github.com/nemanja-m/gaps.

Question 6: Simulated Annealing for the Travelling Salesman Problem (TSP) (Optional)

Use Simulated Annealing (SA) to find short tours for the TSP and study how design choices (neighbour operator, cooling schedule, initial solution, stopping rule) affect solution quality and runtime.

Problem statement: Given N cities with Euclidean coordinates $c_1, \dots, c_N \in \mathbb{R}^2$, find a permutation (tour) π visiting each city exactly once and returning to the start that minimizes the total length

$$L(\pi) \;=\; \sum_{k=1}^{N} d\big(c_{\pi(k)},\, c_{\pi(k+1)}\big), \qquad \text{with } \; c_{\pi(N+1)} \equiv c_{\pi(1)},$$

where $d(\cdot,\cdot)$ is the Euclidean distance. Students will implement SA from scratch to search the space of permutations and produce a near-optimal tour.

Representation & objective:

- **State:** a permutation of city indices (an array of length N).
- **Objective:** minimize total tour length $L(\pi)$. (Treat this as the energy $E(\pi) = L(\pi)$.)

Neighbourhood moves (choose at least two and compare).

- **Swap:** pick two indices i, j and swap the cities at positions i and j.
- **2-opt (segment reversal):** choose indices $i < j$ and reverse the sub-sequence $[i..j]$ (very effective for TSP).
- **Insert:** remove the city at position i and reinsert it immediately after position j.
- **3-opt (optional/advanced).**

Students should implement at least *swap* and *2-opt* and compare their performance.

SA components (implement and experiment):

1. **Initial solution.**
   - Options: random permutation, greedy nearest-neighbour, or a simple heuristic (e.g., sorted by x-coordinate).
   - Compare at least two initializers.
2. **Acceptance probability.**
   - Metropolis criterion: if $E' \le E$ accept; otherwise accept with probability

$$\Pr(\text{accept}) \;=\; \exp\!\left(-\frac{E' - E}{T}\right).$$

3. **Temperature schedule (cooling).**
   - **Geometric:** $T_{k+1} = \alpha T_k$ with $\alpha \in (0,1)$ (typical $\alpha \in [0.90,\, 0.999]$).
   - **Linear:** $T_{k+1} = T_k - \beta$.
   - **Logarithmic (theoretically safe):** $T_k = \dfrac{T_0}{\log(1+k)}$.
   - Experiment with at least two schedules and various parameters.
4. **Inner loop / iterations per temperature.**
   - Either a fixed number of neighbour trials per temperature (e.g., $M = c \cdot N$) or adaptively until the acceptance rate drops below a threshold.
   - Suggested default: $M = 10 \cdot N$.
5. **Stopping criteria.**
   - Minimum temperature $T_{\min}$,

- Maximum number of iterations,
- No improvement for K consecutive temperatures,
- Time budget.

6. **Reheating / restarts (optional).**
   - After stagnation, raise T to escape local minima or perform random restarts and keep the best solution.

Tasks:

(a) Compare **neighbour operators**: swap vs. 2-opt (report best tour length, average over 5 runs, and runtime).

(b) Compare **cooling schedules**: geometric ($\alpha = 0.995$), geometric ($\alpha = 0.999$), linear; include convergence plots.

(c) Compare **initial solutions**: random vs. greedy nearest-neighbour.

(d) **Parameter sensitivity:** vary initial temperature T_0, α, and iterations per temperature; show effect on final tour length.

(e) (Optional) Compare SA to a baseline: greedy heuristic and/or pure 2-opt local search.

For each experiment, report:

- best found tour length,
- mean and standard deviation over several independent runs,
- runtime (seconds) and number of evaluations,
- convergence plot: tour length vs. iteration (or vs. temperature),
- visualization of the best tour path in the plane.

Question 7: Batch vs. Stochastic Gradient Descent

Compare Batch Gradient Descent (BGD) and Stochastic Gradient Descent (SGD). Discuss:

(a) Computational load per iteration.

(b) Noise (variance) in the gradient estimation.

(c) Stability and nature of convergence to the minimum.

Question 8: Steepest Descent Direction via First-Order Taylor

Gradient Descent uses the direction $p_k = -\nabla f(x_k)$. Using a first-order Taylor approximation, prove this is the *steepest descent* direction in a local neighborhood of x_k. (Hint: For small $\alpha > 0$, minimize $f(x_k + \alpha p)$ over unit vectors $\|p\| = 1$.)

Question 9: Pseudoinverse Solution is the Least-SQ12uares Minimizer (Optional)

For an overdetermined system $Ax = b$ with $A \in \mathbb{R}^{m \times n}$ $(m > n)$ full column rank, the candidate

$$x^* = (A^\top A)^{-1} A^\top b$$

satisfies the normal equations $A^\top A\, x^* = A^\top b$. Prove x^* is the *unique global minimizer* of $L(x) = \|Ax - b\|_2^2$.

Question 10: Newton and Quasi-Newton Methods (Optional)

Newton's method faces (i) high cost of forming the Hessian and (ii) $O(n^3)$ cost to invert it. Quasi-Newton methods (BFGS/DFP) address these. Answer:

(a) How is the Newton step computed?

(b) When can Newton's method fail to converge to a minimum (or converge to a saddle/maximum)?

(c) When is the Newton direction not a descent direction?

(d) What is the main idea of Quasi-Newton methods?

(e) How do they approximate the Hessian (or inverse) without explicit second derivatives? (Secant equation.)

Question 11: Quadratic Convergence of Newton's Method

(a) Define *quadratic convergence* using the error $e_k = x_k - x^*$, and contrast with linear convergence.

(b) Show that if x_k is sufficiently close to a local minimizer x^*, then the Newton direction p_k approximates the exact displacement $(x^* - x_k)$. (Use $\nabla f(x^*) = 0$ and a Taylor expansion of the gradient.)

Question 12: Rosenbrock Function: GD vs. Newton (Optional)

Consider the Rosenbrock function

$$f(x_1, x_2) = 100\,(x_2 - x_1^2)^2 + (1 - x_1)^2,$$

whose global minimizer is $x^* = [1, 1]^\top$ with $f(x^*) = 0$.

(a) Compute $\nabla f(x)$ and $H(x)$.

(b) Implement (1) Steepest Descent (GD) and (2) Pure Newton's Method.

(c) For GD, use a fixed step size $\alpha = 0.001$ (bonus: Backtracking Line Search). For Newton, use full steps $\alpha = 1$.

(d) Run both from $x_0 = [-1.2,\, 1]^\top$ (e.g., 50 iterations or until $\|\nabla f(x_k)\| < 10^{-6}$).

(e) Plot: (i) trajectories on a contour plot of f; (ii) $\log f(x_k)$ vs. iteration. Discuss convergence rates.

Question 13: Equality-Constrained Optimization via Lagrange Multipliers & Newton–Raphson (Optional)

Solve

$$\min_{x,y} \; f(x,y) = e^{x^2+y^2} \quad \text{s.t.} \quad g(x,y) = x + y - 1 = 0.$$

(a) Write the Lagrangian $L(x, y, \lambda)$.

(b) Find the optimal (x^*, y^*, λ^*) by solving $\nabla_{x,y,\lambda} L = 0$.

(c) (Coding) Write the 3×3 nonlinear system $F(w) = 0$ with $w = [x, y, \lambda]^\top$.

(d) Apply Newton–Raphson: $w_{k+1} = w_k - J_F(w_k)^{-1} F(w_k)$. Compute the Jacobian $J_F(w)$.

(e) Implement Newton's method and show convergence from $w_0 = [0, 0, 0]^\top$ to the analytic solution from (b).

Question 14: Step-Size Strategies in Gradient Descent

(a) Briefly explain two step-size selection methods (e.g., Backtracking/Armijo, Wolfe conditions, or adaptive methods such as AdaGrad/RMSProp).

(b) Using the Rosenbrock problem (from Q6) with $x_0 = [-1.2, 1]^\top$, implement GD with the two methods from (a).

(c) Compare results (e.g., trajectories, function values vs. iterations, convergence behavior).