Machine Learning

Assignment 2

Florian Thaler

April 15, 2025

Deadline: May 20, 2025
Submission: There are two different types of tasks: the **Pen and paper** category and the **Implementation** category. The former requires a written elaboration (calculations, explanations of results, observations, etc.), while the latter involves the implementation of specified functions and code blocks in the provided template files. Please use only these template files for the implementation. Additionally, please use only the packages included in the template files to solve the tasks.

Summarise all results of the 'Pen and paper' tasks in a PDF file named **report.pdf**. Upload this file and all your implementations to the TeachCenter. Please do not zip the files.

1. Langevin sampling (18P)

Let us discuss another sampling method. Suppose we have a parametric distribution on $\mathbb{R}^d$ with PDF $p(\cdot|\theta)$ such that

$$p(x|\theta) \propto \exp(-f(x|\theta)), x \in \mathbb{R}^d, \tag{1}$$

where $f(\cdot|\theta) : \mathbb{R}^d \to \mathbb{R}$, the so-called energy of the distribution, is a C^1 function. The main idea of Langevin-sampling is to iteratively find samples in high-density regions of the distribution. According to (1) such regions are characterised by large energies and can be reached iteratively by application of the gradient ascent method on $-f(\cdot|\theta)$. Since we are interested in iterates in the neighbourhood of the maximiser of the negative energy, and not in the maximiser itself, a noisy gradient ascent step is considered. This ensures that the algorithm explores the full distribution rather than just converging to a mode. Summarising, this leads us to the so-called unadjusted Langevin algorithm (ULA):

Data: Initial guess x_0, step size γ
for $k = 0, 1, \ldots$ **do**
  Draw a sample z_k from $N(0, 1)$
  $x_{k+1} = x_k - \gamma \nabla f(x_k) + \sqrt{2\gamma} z_k$
end

Algorithm 1: Unadjusted Langevin algorithm (ULA)

1.1. Quadratic energy

Let us consider the case of a quadratic energy, i.e. for a symmetric positive definite matrix $Q \in \mathbb{R}^{d\times d}$, a vector $\mu \in \mathbb{R}^d$ let f be defined by

$$f(x|\mu, Q) = \frac{1}{2}(x-\mu)^T Q(x-\mu), x \in \mathbb{R}^d.$$

The corresponding distribution is actually the multivariate normal distribution with mean μ and covariance matrix $\Sigma = Q^{-1}$.

1. [**Pen and paper**] Let us interpret each Langevin-iterate x_k as realisation of a corresponding random variable X_k and let $(Z_k)_{k\geq 0}$ be a sequence of independent $N_d(0, I)$-distributed random variables. Then, for the deterministic initial guess x_0, the ULA step for the sequence

$(X_k)_{k\geq 1}$ reads

$$X_{k+1} = X_k - \gamma \nabla f(X_k) + \sqrt{2\gamma} Z_k, k \geq 0.$$

By unrolling the update step prove that

$$X_{k+1} = \mu + (I - \gamma Q)^{k+1}(x_0 - \mu) + \sqrt{2\gamma} \sum_{j=0}^{k} (I - \gamma Q)^{k-j} Z_j, k \geq 0. \tag{2}$$

Infer that each ULA iterate follows a normal distribution, i.e. $X_{k+1} \sim N_d(\mu_k, \Sigma_{k+1})$ with

$$\mu_{k+1} = \mu + (I - \gamma Q)^{k+1}(x_0 - \mu),$$
$$\Sigma_{k+1} = (Q - \frac{\gamma}{2} Q^2)^{-1} - (I - \gamma Q)^{2k+2}(Q - \frac{\gamma}{2} Q^2)^{-1}.$$

Hint For the computation of $Var(X_{k+1})$ consider the identities

- $Var(AX) = AVar(X)A^T$ for a n-dimensional random vector X, and a matrix $A \in \mathbb{R}^{m \times n}$.
- $\sum_{i=0}^{l-1} A^i = (I - A)^{-1} - A^l (I - A)^{-1}$ for $l \in \mathbb{N}$, and a quadratic matrix A.

2. [**Pen and paper**] Under the assumption $\|I - \gamma Q\| < 1$ compute the limits $\mu^* = \lim_{k\to\infty} \mu_{k+1}$, and $\Sigma^* = \lim_{k\to\infty} \Sigma_{k+1}$. Do the limits coincide with the expected value and covariance of the distribution we would like to sample from?[1] What happens in the limiting case $\gamma \to 0^+$?

3. [**Pen and paper**] Let us investigate some convergence properties of ULA in the special case $Q = \frac{1}{\alpha} I$, where $\alpha \in \mathbb{R}_{>0}$ with $|1 - \gamma/\alpha| < 1$. Prove that for the distribution P_{X_k} of the ULA-iterates X_k it holds

$$\lim_{k\to\infty} D_{KL}(N_d(\mu^*, \Sigma^*), P_{X_k}) = 0,$$

where $\mu^* = \mu$, $\Sigma^* = (\alpha^2/(\alpha - \gamma/2))I$.

The above statement implies that the sequence of distributions (P_{X_k}) converges w.r.t. the KL-divergence to a normal distribution. As the previous task already indicates, (P_{X_k}) does not converge to the actual target distribution $N_d(\mu, \alpha I)$. For small values of γ, however, we "come close" to the target distribution. We will further investigate these observations in the next tasks.

Hint The KL divergence between $P_1 = N_d(\mu_1, \Sigma_1)$ and $P_2 = N_d(\mu_2, \Sigma_2)$ can be computed as follows

$$D_{KL}(P_1, P_2) = \frac{1}{2}(\log \frac{det(\Sigma_2)}{det(\Sigma_1)} - d + (\mu_1 - \mu_2)^T \Sigma_2^{-1}(\mu_1 - \mu_2) + tr(\Sigma_2^{-1}\Sigma_1))$$

4. [**Implementation**] Let us now investigate numerically the convergence of the sequence of distributions P_{X_k} for a general symmetric and positive definite matrix Q. As in the previous task we will study convergence in terms of the KL divergence. In the template file **langevin_sampling_quadratic.py** implement for this purpose the functions

- *expected_value_ula_quadratic(...)*
- *covariance_matrix_ula_quadratic(...)*
- *kullback_leibler_normals(...)*
- *ula_quadratic(...)*

5. [**Pen and paper**] Apply your code to sample from the distribution with

$$Q = \begin{bmatrix} 4 & 1 \\ 1 & 2 \end{bmatrix}, \mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix}.$$

For each $\gamma \in \{0.01, 0.1, 0.4\}$ generate $m = 10$ chains[2] of $n = 5000$ ULA-iterates. For each of these scenarios use the functions

[1] The discrepancy between target distribution and limit distribution is the reason why Algorithm 1 is said to be unadjusted, or in other words, biased.

[2] The ULA-scheme actually generates for each initial guess x_0 a (finite) sequence of iterates which correspond to a (finite) subpath of a Markov chain. By a chain we simply mean a finite sequence of ULA iterates corresponding to one initial guess. Thus, to generate m chains, apply ULA to m different initial guesses.

- *visualise_kl_divs(...)*
- *visualise_sample_quadratic(...)*

in the template file **langevin_sampling_quadratic.py** to visualise your results. What do you observe w.r.t. the convervence in terms of the Kullback-Leibler divergence? Which influence has the parameter γ on the sample generated by ULA?

6. [**Pen and paper**] Let us consider MALA[3], an approach to adjust Algorithm 1 in such a way that the limiting distribution generated by the algorithm coincides with the target distribution. For this purpose MALA uses the acceptance-rejection procedure of the Metropolis-Hastings-algorithm.

Data: Initial guess x_0, step size γ
for $k = 0, 1, \ldots$ **do**
  Draw a sample z_k from $N(0,1)$ and compute

$$\bar{x}_{k+1} = x_k - \gamma \nabla f(x_k) + \sqrt{2\gamma} z_k$$

  Compute the acceptance probability α_k
  Draw a random sample u from $U(0,1)$
  if $u \leq \alpha_k$ **then**
    $x_{k+1} = \bar{x}_{k+1}$
  else
    $x_{k+1} = x_k$
  end
end

Algorithm 2: MALA

Given an iterate x_k, MALA generates a proposal step $\bar{x}_{k+1}$ in the same manner as ULA via

$$\bar{x}_{k+1} = x_k - \gamma \nabla f(x_k) + \sqrt{2\gamma} z_k. \tag{3}$$

In contrast to ULA, $\bar{x}_{k+1}$ is accepted as new iterate x_{k+1} only with acceptance probability α_k. Let q denote the transition probability density of ULA, i.e. $q(y, x)$ reflects the probability that by the update step (3) the state x is transitioned to state y. Then the acceptance probability is defined as

$$\alpha_k = \min\{1, \frac{q(x_k, \bar{x}_{k+1}) p(\bar{x}_{k+1}|\theta)}{q(\bar{x}_{k+1}, x_k) p(x_k|\theta)}\}.$$

In terms of Markov chain theory, the choice of α_k makes sure that the Markov chain induced by MALA satifies the so-called detailed balance condition. This implies under additional mild assumptions that the Markov chain $(X_k)_{k\geq 1}$ generated by MALA has stationary distribution $p(\cdot|\theta)$ - our target distribution[4].

Show that the transition probability density q satisfies

$$q(y, x) \propto \exp(-\frac{1}{4\gamma} \|y - x + \gamma \nabla f(x)\|_2^2), \ x, y \in \mathbb{R}^d.$$

7. [**Implementation**] Implement the function *mala(...)* in the template file **langevin_sampling_quadratic.py**.

8. [**Pen and paper**] Let us now apply MALA to sample from a distribution with quadratic energy. As in Task 5 apply your implementation to sample from the distribution with quadratic energy $f(\cdot|\mu, Q)$, where

$$Q = \begin{bmatrix} 4 & 1 \\ 1 & 2 \end{bmatrix}, \mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix}.$$

For $\gamma = 0.4$ generate $m = 10$ chains of $n = 5000$ iterates. Use the function *visualise_sample_quadratic(...)* to visualise your results. What do you observe? Compare the result with the results obtained in Task 5.

[3] Metropolis-Adjusted Langevin Algorithm
[4] For a more thorough discussion of the Metropolis-Hastings algorithm see for example [2], or [5]

1.2. Double banana-shaped distribution

Let us apply MALA to sample from a fancier distribution. For parameters $a, b > 0$ consider the density $p(\cdot|a,b) \propto \exp(-f(\cdot|a,b))$, where $f(\cdot|a,b) : \mathbb{R}^2 \to \mathbb{R}$ is defined as

$$f(x_1, x_2|a,b) = \frac{1}{2b^2}(x_1^2 + x_2^2 - 2)^2 - \log(\exp(-\frac{1}{2a^2}(x_1 - 2)^2) + \exp(-\frac{1}{2a^2}(x_1 + 2)^2)).$$

1. [**Pen and paper**] Prove that the partial derivatives of the energy function are given as follows

$$\partial_1 f(x_1, x_2|a,b) = \frac{2}{b^2} x_1(x_1^2 + x_2^2 - 2) + A$$
$$\partial_2 f(x_1, x_2|a,b) = \frac{2}{b^2} x_2(x_1^2 + x_2^2 - 2),$$

   with

$$A = \frac{x_1 - 2 + (x_1 + 2)\exp(-4x_1/a^2)}{a^2(1 + \exp(-4x_1/a^2))}$$

2. [**Implementation**] In the template file **langevin_sampling_double_banana.py** implement the functions
   - *energy_func_double_banana(...)*
   - *grad_energy_func_double_banana(...)*

3. [**Pen and paper**] Use your implementation of MALA from Task to sample from $p(\cdot|a,b)$, with $a = 0.6$, $b = 1.8$. Use $\gamma = 0.1$, and generate $m = 3$ chains of $n = 5000$ MALA-iterates. Use the function *visualise(...)* in the template file **langevin_sampling_double_banana.py** to visualise your results.

2. Stochastic neighbour embedding (SNE) (12P)

Consider the problem of the visualisation of high-dimensional data, such as images. To address this, we aim to map the data into a lower dimensional space. By reducing the dimensions - typically to 2 or 3 - we can use scatter plots to visualise the transformed data.
Let us discuss here the dimensionality-reduction technique called tSNE[5]. This method aims to map data from the high-dimensional space into the low-dimensional space while preserving (pairwise) neighborhood information between data points. Neighborhood information, or similarity of data points, can be modeled in various ways. In tSNE similarities in high- and low-dimensional spaces are expressed in terms of probability distributions. Thus, the requirement of similarity preservation can then be formulated using the Kullback-Leibler divergence: Given the data $\mathcal{X} = \{x_1, \ldots, x_n\} \subset \mathbb{R}^d$ in high-dimensional space and its similarity distribution P, we seek $y_1, \ldots, y_n \in \mathbb{R}^m$, $m \ll d$ such that for its similarity distribution $Q = Q(y_1, \ldots, y_n)$ the Kullback-Leibler divergence $D_{KL}(P||Q)$ is minimal.

To measure the similarity of a pair $(x_i, x_j) \in \mathcal{X} \times \mathcal{X}$ let us introduce

$$p_{j|i} = \begin{cases} \frac{\exp(-\|x_i - x_j\|_2^2/2\sigma_i^2)}{\sum_{k=1,k\neq i}^n \exp(-\|x_i - x_k\|_2^2/2\sigma_i^2)} & \text{if } j \neq i \\ 0 & \text{else} \end{cases}$$

and $p_{i,j} = (p_{j|i} + p_{i|j})/2n$, where the parameters $\sigma_1, \ldots, \sigma_n$ are determined based on the density of the data points: In dense regions σ_i is chosen to be small, whereas in sparse regions σ_i is chosen to be larger[6]. Note that the similarity measure $p_{i,j}$ is symmetric, i.e. $p_{i,j} = p_{j,i}$, where $p_{j|i}$ it isn't.

[5] SNE is short for stochastic neighbour embedding; the 't' in tSNE refers to Student's t-distribution

[6] In [4] the authors propose to choose σ_i such that the perplexities ψ_i, $i = 1, \ldots, n$ with

$$\log_2(\psi_i) = -\sum_{j=1}^n p_{j|i} \log_2(p_{j|i})$$

meet a given target value. This can be achieved numerically by means of a binary search.

Using this kind of similarity measure we thus implicitely assume that for each pair $(x_i, x_j) \in \mathcal{X}$ the point x_i is as much similar to x_j, as x_j is similar to x_i.

Let us denote by P the distribution induced by the set $\{p_{i,j} : i, j = 1, \ldots, n\}$. On the low-dimensional space $\mathbb{R}^m$ we consider the distribution $Q^{(tSNE)}$ induced by

$$q_{i,j}^{(tSNE)} = \begin{cases} \frac{(1+\|y_i - y_j\|_2^2)^{-1}}{\sum_{k,l=1,k\neq l}^{n}(1+\|y_l - y_k\|_2^2)^{-1}} & \text{if } j \neq i \\ 0 & \text{else} \end{cases}$$

1. [**Implementation**] The gradients of $D_{KL}(P||Q^{(t-SNE)})$ w.r.t. $y_1, \ldots, y_n$ can be computed exactly. For every $\nu = 1, \ldots, n$ it holds

$$\partial_{y_\nu} D_{KL}(P||Q^{(tSNE)})(y_1, \ldots, y_n) = 4 \sum_{j=1}^{n} \frac{p_{i,j} - q_{i,j}}{1 + \|y_i - y_j\|_2^2}(y_i - y_j)$$

   We will use these formulas for the implementation of an accelerated gradient method to determine the low-dimensional representatives $y_1, \ldots, y_n$ minimising the Kullback- Leibler divergence between the similarity distributions P and Q. The gradient method we aim to use is the so-called Polyak's heavy ball method - see Section A for its update formula.

   Implement the following functions in the template file
   stochastic_neighbour_embedding.py

   - *compute_distance_matrix(...)*
   - *compute_high_dim_similarity_matrix(...)*
   - *compute_gradient_tsne(...)*
   - *train_tsne(...)*

   Do not use any loops or iterators in your implementation of *compute_distance_matrix(...)* and use the log-sum-exp trick in your implementation of *compute_high_dim_similarity_matrix(...)*.

   Note that early exxageration is applied in the training method *train_tsne(...)* to boost learning performance during initial stages[7].

2. [**Pen and paper**] Let us now apply the implementation to map a random sample of size $n = 1000$ consisting of the $0, \ldots, 4$ from the MNIST dataset[8]. Use the optimiser parameters

α	β	num_iterations
500	0.7	500

   and choose perplexity[9] 20.

   Visualise all of your results. You can use the function *visualise(...)* provided in the template file. Feel free to implement your own function for the visualisation of the low-dimensional representatives of the data. If so, please use only the packages which are already included in the template file.

[7] This parameter seems to be rather important to get nice results - feel free to experiment with it.

[8] See for example https://github.com/cvdfoundation/mnist?tab=readme-ov-file

[9] As pointed out in [4] typical values for this parameter are between 5 and 50. I experienced that the results are pretty robust w.r.t. the perplexity - choose extreme values like 2, or 750 if you would like to see its impact.

Appendix

A. Polyak's heavy ball method

For a function $f : \mathbb{R}^d \to \mathbb{R}$ consider the problem

$$\min_{x \in \mathbb{R}^d} f(x). \tag{4}$$

Starting from an initial guess x_0, the vanilla gradient descent method for the solution of (4) iteratively generates a sequence $(x_k)_{k \geq 0}$ according to the update rule

$$x_{k+1} = x_k - \alpha_k \nabla f(x_k), k \geq 0,$$

where $\alpha_k \in \mathbb{R}_{\geq 0}$ refers to the step size iteration k. There are several methods for choosing these step sizes, such as

- Constant step sizes
- Backtracking line search
- Exact line search

The choice of step sizes does significantly affect the convergence behaviour of the gradient descent algorithm. Under mild assumptions on f, ∇f one can prove that for each of the abovementioned line search strategies the sequence $(f(x_k))$ converges to the minimum of f with rate $\mathcal{O}(1/k)$. See for example Theorem 4.25 in [1].
Although, the gradient descent method often suffers from slow convergence - a feasible approach to improve its convergence behaviour is to introduce a momentum term. This leads to the so-called Polyak's heavy ball method. For non-negative momentum parameters β_k, the update rule reads

$$x_{k+1} = x_k - \alpha_k \nabla f_{x_k} + \beta_k (x_k - x_{k-1}), k \geq 0.$$

Under suitable conditions, the sequence generated by the heavy ball method converges linearly to the minimiser of f - see [3]. Note that, the momentum term in the update formula helps to prevent the method getting stuck in regions of local minima. Therefore, the method not only accelerates convergence but, in some cases, enables convergence to the global minimizer. For an illustration of the impact of the momentum see for example https://distill.pub/2017/momentum/.

B. Log-sum-exp trick

Let $x_1, \ldots, x_n \in \mathbb{R}$ and let $c \in \mathbb{R}$. Then it holds

$$\log(\sum_{j=1}^{n} \exp(x_j)) = c + \log(\sum_{j=1}^{n} \exp(x_j - c))$$

This identity is important since it can prevent numerical overflow: Defining $c = \max_{1 \leq j \leq n} x_j$, we achieve that the largest summand on the right hand side of the above formula equals 1.
Similarly,

$$\frac{\exp(x_i)}{\sum_{j=1}^{n} \exp(x_j)} = \frac{\exp(x_i - c)}{\sum_{j=1}^{n} \exp(x_j - c)}.$$

As above, this identity helps to prevent numerical issues.

References

[1] Amir Beck. *Introduction to nonlinear optimization: Theory, algorithms, and applications with MATLAB*. SIAM, 2014.

[2] Siddhartha Chib and Edward Greenberg. Understanding the metropolis-hastings algorithm. *The american statistician*, 49(4):327–335, 1995.

[3] Boris T Polyak. Some methods of speeding up the convergence of iteration methods. *Ussr computational mathematics and mathematical physics*, 4(5):1–17, 1964.

[4] Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(11), 2008.

[5] E Weinan, Tiejun Li, and Eric Vanden-Eijnden. *Applied stochastic analysis*, volume 199. American Mathematical Soc., 2021.