Machine Learning

Assignment 3

Florian Thaler

May 20, 2025

Deadline: July 01, 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.

Terminology and notation

This assignment is about the training of energy based models (EBMs) and their application as generative models - the methods to be discussed are outlined and emphasized in [2].

By an EBM we mean a parameter-dependent probability distribution P_θ with PDF $p(\cdot|\theta)$ where

$$p(x|\theta) \propto \exp(-f(x|\theta)).$$

The mapping $f(\cdot|\theta)$ is called the energy of P_θ[1].

Throughout this document we will denote a training dataset by $\mathcal{X}$, where each element $x \in \mathcal{X}$ is interpreted as an iid sample/observation from the (unknown) data distribution P_{data}. We assume that P_{data} has PDF p_{data}. Finally, we use the notation $E_{X\sim R}(g(X))$ to indicate that the expected value of $g(X)$ is taken w.r.t. the distribution R.

1 Maximum likelihood training (13 P)

The learning approach we want to investigate here is based on the idea of fitting P_θ to P_{data} by maximising the log-likelihood function. Following the standard maximum likelihood principle, our objective is to solve

$$\operatorname*{argmax}_\theta \sum_{x\in\mathcal{X}} \log(p(x|\theta)).$$

However, the log-likelihood function involves the normalising constant $z(\theta) = \int \exp(-f(x|\theta))dx$, which is in general intractable. Consequently, we cannot directly solve the above optimisation problem. It turns out that, while the log-likelihood function itself is intractable, its gradient can

[1] Recall Task 1 of assignment 2

be approximated reasonably well. Exploiting the defining property of an EBM we immediately get that

$$\nabla_\theta \log(p(x|\theta)) = -\nabla_\theta f(x|\theta) - \nabla_\theta \log(z(\theta)),\ x \in \mathbb{R}^d. \tag{1}$$

1. [**Pen and paper**] Under the assumption that $\nabla_\theta \int \exp(-f(x|\theta))dx = \int \nabla_\theta \exp(-f(x|\theta))dx$ prove that

$$\nabla_\theta \log(z(\theta)) = E_{X\sim p(\cdot|\theta)}(-\nabla_\theta f(X|\theta)). \tag{2}$$

   Consequently we obtain

$$\nabla_\theta \log(p(x|\theta)) = -\nabla_\theta f(x|\theta) - E_{X\sim p(\cdot|\theta)}(-\nabla_\theta f(X|\theta)). \tag{3}$$

   Approximating the latter term using a single Monte Carlo sample $\hat{x}$ from P_θ, we get

$$\nabla_\theta \log(p(x|\theta)) \approx -\nabla_\theta f(x|\theta) + \nabla_\theta f(\hat{x}|\theta).$$

2. [**Pen and paper**] Let $x_1, \ldots, x_n \in \mathcal{X}$ and let $\hat{x}_1, \ldots, \hat{x}_n$ be samples of $p(\cdot|\theta)$. Define the mapping l via

$$l(\theta) = -\frac{1}{n}\sum_{j=1}^{n}(f(x_j, \theta) - f(\hat{x}_j, \theta)).$$

   Explain why l is a suitable choice for the loss function in the given context.

3. [**Implementation**] Implement the functions
   - *sample(...)*
   - *max_likelihood_loss(...)*
   - *train_ebm_max_likelihood(...)*

   in the template files **SamplingUtils.py** and **maximum_likelihood_training()** respectively. Feel free to use any PyTorch optimiser in your training loop.

4. [**Pen and paper**] Apply your implementation to train an EBM on the 'swiss-roll' dataset. Provide visualisations that illustrate the behaviour of the trained model and the training process. This should include a plot of the learned energy landscape, a scatter plot of samples generated by the trained model, and a graph showing the evolution of the loss during training. Feel free to implement your own, custom[2] visualisation functions or to use the functions
   - visualise_energy_landscape(...)
   - visualise_samples(...)
   - visualise_optimisation_stats(...)

   In addition, provide in tabular form all the hyperparameters you used to generate the results.

2 Noise contrastive estimation (13 P)

The main idea of the training principle we want to investigate here is in turning the actual learning problem into a binary classification task - specifically, distinguishing real data from artificially generated noise

Let $\mathcal{Y}$ denote the set of noise samples, drawn from a well known distribution Q (e.g. multivariate normal distribution) with PDF q. Introduce the set $\mathcal{U} = \mathcal{X} \cup \mathcal{Y}$, where each element $u \in \mathcal{U}$ is trated as realisation of a random variable U. Further introduce the binary random variable C defined as $C = \mathbb{1}_{\mathcal{X}}(U)$. We assume that $C \sim Ber(1/2)$, which basically means that $|\mathcal{X}| = |\mathcal{Y}|$.

1. [**Pen and paper**] For the sake of simplicity assume that both, the data distribution P_{data} and the noise distribution Q are discrete[3].

[2] Please use only the packages which are already included in the template files.
[3] Note that noise contrastive estimation is not (!) bound to discrete distributions.

a) Prove that for all $u \in \mathcal{U}$, $c \in \{0, 1\}$ the posterior probabilities $P(C = c|U = u)$ can be computed as follows

$$P(C = c|U = u) = \begin{cases} \frac{p_{data}(u)}{p_{data}(u)+q(u)} & \text{if } c = 1 \\ \frac{q(u)}{p_{data}(u)+q(u)} & \text{if } c = 0 \end{cases}$$

Note that the posterior probabilities depend on the PDF p_{data} of the unknown data distribution P_{data}.

b) Consider the binary classification problem of distinguishing whether a sample $u \in \mathcal{U}$ stems from $\mathcal{X}$ or from $\mathcal{Y}$. Let us introduce the set $\mathcal{C} = \{\mathbb{1}_{\mathcal{X}}(u) : u \in \mathcal{U}\}$, where we assume that all elements of $\mathcal{C}$ were sampled independently.

Prove that the log-likelihood problem of this classification task reads

$$\operatorname*{argmax}_{\theta} \sum_{x \in \mathcal{X}} \log(r(x|\theta))) + \sum_{y \in \mathcal{Y}} \log(1 - r(y|\theta)), \tag{4}$$

where $r(u|\theta) = p(u|\theta)/(p(u|\theta) + q(u))$, $u \in \mathcal{U}$.

2. [**Implementation**] Solving problem (4) requires that we know or can compute the normalising constant $z(\theta)$ of P_θ[4]. In general we do not have any information on $z(\theta)$. Unlike in Task 1, where the learning objective was designed to avoid the need for evaluating $z(\theta)$, a different strategy is adopted here: We treat $\log(1/z(\theta))$ as additional trainable parameter of our model.

Implement the functions

- *nce_loss(...)*
- *train_ebm_noise_contrastive_estimation(...)*

in the template file **noise_contrastive_estimation.py** while using the PyTorch optimiser of your choice.

3. [**Pen and paper**] Apply your implementation to train an EBM with noise contrastive estimation on the 'gaussians' dataset. Provide a table containing all the hyperparameters you used to train your model. Provide plots of the energy landscape of the trained model, a scatter plot of samples generated by the model and a plot depicting the evolution of training loss over time. You can write your own, custom [5] functions for this purpose, or use the already implemented functions

- visualise_energy_landscape(...)
- visualise_samples(...)
- visualise_optimisation_stats(...)

to visualise your results.

3 Score matching (14 P)

The objective of score matching is to learn the score function $\log(p(\cdot|\theta))$ of an EBM P_θ. For this purpose, following [1], we aim to solve

$$\operatorname*{argmin}_{\theta} E_{X \sim P_{data}}(\frac{1}{2}\|\nabla_x \log p_{data}(X) - \nabla_x \log p(X|\theta)\|_2^2). \tag{5}$$

[4] Recall Task 1

[5] Please use only packages which are already included in the template files.

3.1 Implicit score matching

In contrast to the learning principles discussed in the preceding sections, solving (5) does not require knowledge of the normalising constant $z(\theta)$. However, the formulation depends on the score function of the data distribution P_θ, which is in general intractable or difficult to approximate. Fortunately, as shown in Theorem 1 of [1], and under mild regularity assumptions, problem (5) can be equivalently reformulated as

$$\operatorname*{argmin}_\theta E_{X\sim P_{data}}(\frac{1}{2}\|\nabla_x f(X|\theta)\|_2^2 + \operatorname{tr}(\nabla_x^2 f(X|\theta))).$$

Note that this formulation involves derivatives of second order, whose numerical computation is usually very expansive - particularly in high-dimensional settings. An effective strategy to circumvent this issue is to directly approximate the score function rather than the energy function itself. As a result, only first-order derivatives are needed. This approach is adopted in the provided template files - see the implementation of *EnergyGradientModel* in the template file **EnergyModel.py**.

1. [**Implementation**] Implement the function
   - *implicit_score_matching_loss(...)*

   in the template file **implicit_score_matching.py**. Do not use more than one for-loop for the computation of the trace.

   Hint Compute gradients using *torch.autograd.grad(..., grad_outputs=...)*.

2. [**Pen and paper**] Apply your code to train an energy based model on the 'rings' dataset. Provide a plot illustrating the score of the trained EBM, a scatter plot of samples generated by the trained model and plot showing the evolution of the loss over training. You can implement custom visualisation routines or use the methods
   - *visualise_optimisation_stats(...)*
   - *visualise_gradient_field(...)*
   - *visualise_samples(...)*

   Please note that, provided your implementation of the score matching loss is correct, there is no need for hyperparameter tuning to get decent results.

3.2 Denoising score matching

The variant of score matching we will discuss in this subsection, renounces on the need to compute derivatives for the loss function. To motivate this approach recall the original score matching objective (5). Unlike the original formulation we consider the noise-perturbed score matching objective

$$\frac{1}{2}E(\|\nabla_y \log p_{data}(Y) - \nabla_y \log p(Y|\theta)\|_2^2),$$

where $X \sim P_{data}$ and Y with $Y = X + \sigma Z, Z \sim N_d(0, I)$, $\sigma > 0$ is a noisy version of X. As proved by Vincent in [3], minimising this perturbed score matching objective is equivalent to the problem

$$\operatorname*{argmin}_\theta E_{(X,Y)\sim P_{data,\sigma}}(\frac{1}{2}\|\nabla_y \log q(Y|X) - \nabla_y \log p(Y|\theta)\|_2^2). \tag{6}$$

Here, $P_{data,\sigma}$ refers to the joint distribution of X and Y, and $q(y|x)$ represents the conditional density of Y given $X = x$.

1. [**Pen and paper**] Let $\mathcal{Y}$ denote the perturbed dataset , i.e.

   $$\mathcal{Y} = \{x + \sigma z : z \text{ is a realisation of } N_d(0, I)\}.$$

   Using the fact that $\nabla_y \log q(y|x) = (x - y)/\sigma^2$ express the loss function corresponding to (6).

2. [**Implementation**] For the implementation of denoising score matching, by filling in the logic for the functions
   - *denoising_score_matching_loss(...)*

- *train_ebm_denoising_score_matching(...)*

in the template file **denoising_score_matching.py**. Use a PyTorch optimiser of your choice.

As for the implementation of implicit score matching in the previous subsection, it is convenient to use a model approximating the score of the unknown data distribution.

3. [**Pen and paper**] Apply your code to train an EBM on the 'rings' dataset. As for Task 2 in the previous section, provide plots illustrating your results[6]. In addition provide a table containing all the hyperparameters you used to generate the results.

[6] Reuse previous implementations.

References

[1] Aapo Hyvärinen and Peter Dayan. Estimation of non-normalized statistical models by score matching. *Journal of Machine Learning Research*, 6(4), 2005.

[2] Yang Song and Diederik P Kingma. How to train your energy-based models. *arXiv preprint arXiv:2101.03288*, 2021.

[3] Pascal Vincent. A connection between score matching and denoising autoencoders. *Neural computation*, 23(7):1661–1674, 2011.