Computational Statistics & Machine Learning

Lecture 14

Energy-based and Score-based Modelling

Mark Girolami
mag92@cam.ac.uk

Department of Engineering
University of Cambridge

October 15, 2025

Overview

- Generative modelling
- Energy-based models
- Score-based models
- Sampling

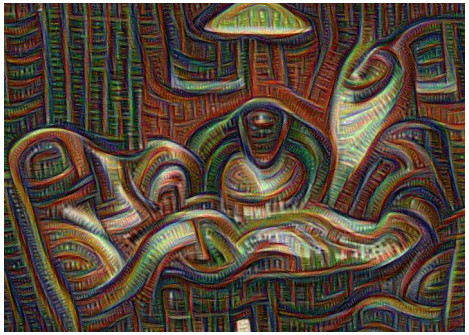
Generative modelling

- Learning models of the data distribution is a significant problem in data science and machine learning
- Given a dataset $\mathcal{D} = \{x_i\}_{i=1}^N$
- Assumed generated according to some *unknown* data distribution $x_i \sim p_{\text{data}}(x)$
- Aim to approximate $p_{\text{data}}(x)$ with parameterised $p_\theta(x)$
- Learn parameters θ from the data $\mathcal{D}$

We will cover different methods for learning $p_\theta(x)$, and generating samples $x' \sim p_\theta(x)$

Generative modelling of images

DeepDream (2014)



GANs (2015)

Stable Diffusion, DALL-E, Midjourney (Current)

Energy-based models

- Inspired by statistical physics, where probability of a system state is modelled via an energy function $E(x)$
- The probability of a state x, is determined by the energy function $E(\cdot)$, and the (inverse) temperature β

$$p(x) = \frac{1}{Z_\beta} \exp\left(-\beta E(x)\right)$$

- Known as the Boltzmann or Gibbs distribution
- Normalisation constant

$$Z_\beta = \int \exp(-\beta E(x)) \mathrm{d}x$$

  is the 'partition function'
- Partition function generally difficult to compute - requires integration over all states

Energy-based models

- EBMs directly parameterise the energy function:

$$p_\theta(x) \propto \exp\left(-E_\theta(x)\right)$$

- Boltzmann machines were introduced by Geoffrey Hinton and Terry Sejnowski in 1985
- Early example of neural networks modelling complex distributions
- Trained via maximum-likelihood or constrastive divergence

Training energy-based models

$$p_\theta(x) = \frac{1}{Z_\theta} \exp\left(-E_\theta(x)\right), \quad Z_\theta = \int \exp\left(-E_\theta(x)\right)\, \mathrm{d}x$$

- Maximum-likelihood

$$\mathcal{L}(\theta) \coloneqq \mathbb{E}_{x\sim p_{\text{data}}(x)}\left[\log p_\theta(x)\right], \quad \theta^\star = \underset{\theta}{\text{argmax}}\left\{\mathcal{L}(\theta)\right\}$$

- Parameters optimised via gradient methods

$$\nabla_\theta \log p_\theta(x) = -\nabla_\theta E_\theta(x) - \nabla_\theta \log Z_\theta$$

- Second term involving partition function Z_θ can be shown to be equivalent to

$$\nabla_\theta \log Z_\theta = -\mathbb{E}_{x\sim p_\theta(x)}\left[\nabla_\theta E_\theta(x)\right]$$

- Giving the gradient of the objective as

$$\nabla_\theta \mathcal{L}(\theta) = -\mathbb{E}_{x\sim p_{\text{data}}(x)}\left[\nabla_\theta E_\theta(x)\right] + \mathbb{E}_{x\sim p_\theta(x)}\left[\nabla_\theta E_\theta(x)\right]$$

Contrastive Divergence

The maximum likelihood gradient requires sampling from the model p_θ (second term).

- Generally requires using expensive MCMC methods
- Contrastive divergence uses practical (but biased) approximation:

1. For each x_i, initialise MCMC chain (p_θ) at $x_i^{(0)} := x_i$
2. Run for short sequence of k steps, giving $x_i^{(0)}, \ldots x_i^{(k)}$
3. Compute gradient estimate

$$\hat{\mathcal{L}} = -\frac{1}{N}\sum_{i=1}^{N} \nabla_\theta E_\theta(x_i^{(0)}) + \frac{1}{N}\sum_{i=1}^{N} \nabla_\theta E_\theta(x_i^{(k)})$$

4. Update parameters $\theta \leftarrow \theta + \eta\hat{\mathcal{L}}$
   (or other gradient-based method)

- Downside is CD is biased, and MCMC can be very slow. Due to computing this partition function

Score-based models

Score-based models, instead of directly modelling the energy function, aim to match the **score** of the data-distribution

- Denoting the score function $s_\theta(x) := \nabla_x \log p_\theta(x)$
- Working with the score means we don't have to calculate the partition function Z_θ:

$$s_\theta(x) = -\nabla_x E_\theta(x) - \underbrace{\nabla_x \log Z_\theta}_{=0} = -\nabla_x E_\theta(x) \qquad (1)$$

- Score matching, introduced by Hyvärinen (2005), can be used to train these score models
- Aim to minimise the Fisher divergence between the true and approximate score:

$$J(\theta) = \frac{1}{2}\mathbb{E}_{x\sim p_{\text{data}}(x)}\left[\|\nabla_x \log p_{\text{data}}(x) - s_\theta(x)\|^2\right]$$

as we are only given samples from p_{data}, this is intractable in its current form - can be rearranged

Score matching

Fisher divergence can be shown to be equivalent to the following:

$$J_{\mathrm{SM}}(\theta) = \mathbb{E}_{x \sim p_{\mathrm{data}}(x)} \left[\mathrm{tr}(\nabla_x s_\theta(x)) + \frac{1}{2} \left\| s_\theta(x) \right\|^2 \right]$$

which can be estimated using data samples via MC.

- However... the trace of the Jacobian, $\mathrm{tr}(\nabla_x s_\theta(x))$, is expensive for high-dimensions.

Sliced score matching uses random projections to approximate this trace.

Sliced Score matching

- Random projections, denoted v are used to approximate the trace of the Jacobian.
- The following equality can be proved for standard normal projections, i.e. $p(v) = \mathcal{N}(0, \mathrm{I})$

$$\mathrm{tr}(\nabla_x s_\theta(x)) = \mathbb{E}_{v \sim p(v)} \left[v^\top \nabla_x s_\theta(x) v \right]$$

- Applying this gives the objective function for sliced score matching:

$$J_{\mathrm{SSM}}(\theta) = \mathbb{E}_{v \sim p(v)} \mathbb{E}_{x \sim p_{\mathrm{data}}} \left[v^\top \nabla_x s_\theta(x) v + \frac{1}{2} \left\| s_\theta(x) \right\|^2 \right]$$

- The term $v^\top \nabla_x s_\theta(x) v$ can be computed efficiently in high dimensions using Jacobian-vector products

how are these trained in practice?

Training Score-based Models

For standard score matching, we use the data samples to update parameters using any gradient descent scheme.

- Initialise θ_0, then iterate $t = 1, \ldots$

$$\hat{J}_{\mathrm{SM}}(\theta_t) = \frac{1}{N} \sum_{i=1}^{N} \mathrm{tr}(\nabla_x s_{\theta_t}(x_i)) + \frac{1}{2} \left\| s_{\theta_t}(x_i) \right\|^2$$

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta \hat{J}_{\mathrm{SM}}(\theta_t)$$

- or stochastic gradient descent, using batch $B_t \subset \{1, \ldots, N\}$, size $m = |B_t|$

$$\tilde{J}_{\mathrm{SM}}(\theta_t) = \frac{1}{m} \sum_{i \in B_t} \mathrm{tr}(\nabla_x s_{\theta_t}(x_i)) + \frac{1}{2} \left\| s_{\theta_t}(x_i) \right\|^2$$

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta \tilde{J}_{\mathrm{SM}}(\theta_t)$$

Training Score-based Models

- Sliced score matching requires sampling from $p(v)$ to compute the random projection.
- In practice, typically one sample is used for computing this estimate

$$\hat{J}_{\text{SSM}}(\theta_t) = \frac{1}{N}\sum_{i=1}^{N} v_i^\top \nabla_x s_{\theta_t}(x_i) v_i + \frac{1}{2}\left\|s_{\theta_t}(x_i)\right\|^2$$

  where $v_i \sim p(v)$.
- Once we have estimated parameters $\theta^\star$, we can evaluate the estimated score as $s_{\theta^\star}(x)$.
- How can we generate new samples using the estimated score?
- Can't use MCMC like for EBMs, as we can't evaluate $p_{\theta^\star}(x)$, we can only evaluate $s_{\theta^\star}(x)$
- Langevin dynamics

Sampling

- Initial sample $x_0 \sim p_0(x)$ from some initial distribution
- Recall Langevin SDE of the form

$$\mathrm{d}x_t = \nabla_x U(x)\mathrm{d}t + \sqrt{2}\,\mathrm{d}B_t$$

has invariant density $p(x) \propto \exp(-U(x))$.

- Setting $\nabla_x U(x) = s_{\theta^\star}(x)$ gives us $p_{\theta^\star}(x)$ as invariant density
- Simulate Langevin dynamics, e.g. ULA, with step-size $\gamma > 0$

$$x^{(k+1)} = x^{(k)} + \gamma s_{\theta^\star}(x^{(k)}) + \sqrt{2\gamma} z^{(k)}$$

where $z^{(k)} \sim \mathcal{N}(0, \mathrm{I})$.