Contents

Todo list

Section 1: Learned Convex Prior Sampling

1.1 Core papers and ideas

- Exponential convergence of Langevin distributions and their discrete approximations [4]
  - Propose ULA
    - Thm 2.1: if our convex regularizer is smaller than $a\|x\|^2 + b$ for large $\|x\|$ we can expect convergence. We don't know the rate at this point
    - Thm 2.3 does give a criterion for geometric convergence
    - Thm 2.4 gives criterion for when we don't get the rate (or convergence whatsoever?)
  - And propose MALA and give similar convergence results with rates
  - At the end of this paper it is not very clear whether MALA is much better than ULA: both seem to converge geometrically under the same conditions
    - According to [5]: the difference between ULA and MALA seems to be a bias for the former
      - so we don't converge to the correct distribution
      - only for very small step sizes, but then the method becomes infeasible
- Proximal markov chain monte carlo algorithms [3]
  - Instead of doing a forward Euler gradient step, we can also do backward Euler, i.e., a proximal step
  - Propose P-ULA and P-MALA
    - Thm 3.1: we get similar result as Thm 2.3 in [4] for P-ULA
    - Thm 3.4: geometric convergence or P-MALA
  - Use FBS to get a decent expression for the prox of the whole functional
    - No proof here how accurate the FBS approximation is, but emperically very accurate (also for high dimensional models)
    - Later looked into again [**?**] and is now known as MYULA (and MYMALA?)
  - Applying the models still relies on having a good approximation of the proximal map
    - Here they use TV and use [1] for the TV algorithm
    - Duality based and not very generalizable to other functionals
- Accelerating proximal Markov chain Monte Carlo by using an explicit stabilised method [5]
  - Speed-up, but no theoretical contributions

- Learned convex regularizers for inverse problems [2]
  - learning regularizers

1.2 Ideas for the project

Main three tasks

- Set up MYULA
- Set up Regulariser learner
- Set up Prox solver

Further to start with

- Get MYMALA/MYULA to work on something we know the distribution of
  - 2D Gaussian and tikhonov regularization
    * also implement ULA/MALA for comparison
  -

Section 2: Proximal Map Learning

2.1 Main Idea

We want to be able to do the Markov step

$$X_{n+1} = X_n - \delta \nabla f(X_n) - \frac{\delta}{\lambda}\left(X_n - \operatorname{prox}_g^\lambda(X_n)\right) + \sqrt{2\delta} Z_{n+1} \tag{1}$$

where f our data fidelity term and $g = \mathcal{R}_\theta$ our learned regularizer.

Note that in the construction of the Markov chain we used

$$\begin{aligned} \nabla \log \pi^\lambda &= -\nabla f(x) - \nabla g^\lambda(x) \\ &= -\nabla f(x) - \frac{1}{\lambda}\left(x - \operatorname{prox}_g^\lambda(x)\right) \end{aligned} \tag{2}$$

Instead of learning the prox directly, we can also try to learn g^λ. The immediate pro here is:

- the function has scalar (1D) output: dimensionality-wise efficient representation
- the function is differentiable
- the function is convex!

So we can use the input-convex neural networks (ICNNs) framework

Then for the resulting NN $\mathcal{H}_\phi$ we can either plug in its gradient right away or we can compute the proximal map explicitly through

$$\operatorname{prox}_g^\lambda(x) = x - \lambda \nabla \mathcal{H}_\phi(x) \tag{3}$$

2.2 First Steps: a Toy Example

We will first focus on an example of something we already know the answer of: $g(x) = \alpha\|x\|_1$. There we have for $x \in \mathbb{R}^N$

$$g^\lambda(x) = \inf_y \left\{\frac{1}{2}\|x-y\|_2^2 + \lambda\alpha\|y\|_1\right\} = \sum_i^N S_{\lambda\alpha}(x_i) \tag{4}$$

where

$$S_{\lambda\alpha}(z) = \begin{cases} \frac{1}{2\lambda\alpha} z^2, & |z| \leq \lambda\alpha \\ |z| - \frac{\lambda\alpha}{2}, & |z| > \lambda\alpha \end{cases} \tag{5}$$

works coordinate wise, i.e., $x = (x_i)_i$.

2.3 Approach 1: supervised learning

Data generation

So we want to train a network $\mathcal{H}_\phi$ that approximates g^λ. We will initially use a supervised approach with data $\{x_j, y_j\}$ where $x_j \sim \pi_X$ and $y_j = g^\lambda(x_j)$[1]

- Fix λ (we can also do without this assumption, but initially this will be easier for data generation and training)
- At this point it is not entirely clear how to sample from π_X in order to generate our x_j
- I guess normally, we do have some x_j's already which we used to train our $\mathcal{R}_\theta$. We should reuse them then
- In that sense, it would make sense to sample from the *prior distribution*.
  - Then, for our toy example we will use $\pi_X = \text{Laplace}(0, \frac{2}{\alpha^2})^N$

    $$x_j \sim \frac{1}{2\alpha} e^{-\alpha \|x\|_1} \mathrm{d}x \tag{6}$$

    corresponding to our prior.
  - We should note that we have typically trained a regularizer $\mathcal{R}_\theta$ and not $\alpha \mathcal{R}_\theta$.
    * So maybe we shouldn't use α in the distribution for the toy example either
    * Or we should just set $\alpha = 1$ and assume that our regularizer is just fine as it is.

The Learning problem

We need to formulate a loss function to define the learning problem. We will initially choose plain ℓ^2 loss, i.e.,

$$\phi^* = \arg\min_\phi \left(\mathbb{E}_{\pi_X} \left[\left(\mathcal{H}_\phi(X) - g^\lambda(X) \right)^2 \right] \right) \tag{7}$$

and in order to do this, we will minimize the emperical risk

$$\phi^* = \arg\min_\phi \frac{1}{N} \sum_j^N \|\mathcal{H}_\phi(x_j) - y_j\|^2. \tag{8}$$

Additionally we can potentially use regularisation:

- ℓ^2
- $\|\nabla \mathcal{H}_\phi(\cdot)\|^2$
- $\|1 - \lambda \nabla \mathcal{H}_\phi(x_j)\|^2$

[1] For a general $\mathcal{R}_\theta$, we just need to solve the problem here a couple of times to generate data

2.4 Approach 2: unsupervised learning

Data generation
So we want to train a network $\mathcal{H}_\phi$ that approximates g^λ. We will initially use a supervised approach with data $\{x_j, y_j\}$ where $x_j \sim \pi_X$ and $y_j = g^\lambda(x_j)$[2]

- Fix λ (we can also do without this assumption, but initially this will be easier for data generation and training)
- At this point it is not entirely clear how to sample from π_X in order to generate our x_j
- I guess normally, we do have some x_j's already which we used to train our $\mathcal{R}_\theta$. We should reuse them then
- In that sense, it would make sense to sample from the *prior distribution.*
  - Then, for our toy example we will use $\pi_X = \text{Laplace}(0, \frac{2}{\alpha^2})^N$

$$x_j \sim \frac{1}{2\alpha} e^{-\alpha\|x\|_1} \mathrm{d}x \tag{9}$$

    corresponding to our prior.
  - We should note that we have typically trained a regularizer $\mathcal{R}_\theta$ and not $\alpha\mathcal{R}_\theta$.
    * So maybe we shouldn't use α in the distribution for the toy example either
    * Or we should just set $\alpha = 1$ and assume that our regularizer is just fine as it is.

The Learning problem
We need to formulate a loss function to define the learning problem. We know that the poximal map $y = \text{prox}_g^\lambda(x)$ solves

$$g^\lambda(x) = \inf_y \left\{ \frac{1}{2}\|x - y\|_2^2 + \lambda\alpha\mathcal{R}_\theta(y) \right\} \tag{10}$$

Using eq. (3) we define the learning problem

$$\phi^* = \arg\min_\phi \left\{ \mathbb{E}_{\pi_X} \left[\frac{\lambda^2}{2} \|\nabla\mathcal{H}_\phi(x)\|^2 + \lambda\alpha\mathcal{R}_\theta(x + \lambda\nabla\mathcal{H}_\phi(x)) \right] \right\} \tag{11}$$

and in order to do this, we will minimize the emperical risk

$$\phi^* = \arg\min_\phi \left\{ \frac{1}{N} \sum_j^N \frac{\lambda^2}{2} \|\nabla\mathcal{H}_\phi(x_j)\|^2 + \lambda\alpha\mathcal{R}_\theta(x_j + \lambda\nabla\mathcal{H}_\phi(x_j)) \right\} \tag{12}$$

Additionally we can potentially use regularisation:

[2] For a general $\mathcal{R}_\theta$, we just need to solve the problem here a couple of times to generate data

- ℓ^2 on the weights

and we might even extend our data to the distribution $A_\sharp^\dagger \pi_Y$, i.e.,

$$\phi^* = \arg\min_\phi \left\{ \mathbb{E}_{\pi_X} \left[F^\lambda(x, \nabla \mathcal{H}_\phi(x)) \right] + \mathbb{E}_{A_\sharp^\dagger \pi_Y} \left[F^\lambda(x, \nabla \mathcal{H}_\phi(x)) \right] \right\} \tag{13}$$

where

$$F^\lambda(x, y) = \frac{\lambda^2}{2} \|y\|^2 + \lambda \alpha \mathcal{R}_\theta(x + \lambda y) \tag{14}$$

in order to make sure that the typical inputs of the markov chain will be mapped to the correct proximal mapping

Network Architecture
ICNN

References

[1] Antonin Chambolle. An algorithm for total variation minimization and applications. *Journal of Mathematical imaging and vision*, 20(1):89–97, 2004.

[2] Subhadip Mukherjee, Sören Dittmer, Zakhar Shumaylov, Sebastian Lunz, Ozan Öktem, and Carola-Bibiane Schönlieb. Learned convex regularizers for inverse problems. *arXiv preprint arXiv:2008.02839*, 2020.

[3] Marcelo Pereyra. Proximal markov chain monte carlo algorithms. *Statistics and Computing*, 26(4):745–760, 2016.

[4] Gareth O Roberts, Richard L Tweedie, et al. Exponential convergence of langevin distributions and their discrete approximations. *Bernoulli*, 2(4):341–363, 1996.

[5] Luis Vargas, Marcelo Pereyra, and Konstantinos C Zygalakis. Accelerating proximal markov chain monte carlo by using an explicit stabilised method. *arXiv preprint arXiv:1908.08845*, 2019.