Wasserstein gradient flow project

Victor Mercklé
April 17, 2024

Contents

1 Gradient Flow vs Wasserstein GF **1**

2 Simple example 2D setting, grid jko **2**

2.1 JKO stepping with Dykstra's algorithm 3

3 Classic setup **3**

4 Infinite width, using a measure: mean-field **4**

4.1 Algorithm, discretize the measure's space 5

4.2 JKO 5

4.3 Papers 5

4.3.1 Grid problems 5

1 Gradient Flow vs Wasserstein GF

Take a two layer ReLU network with m neurons. Each neuron has a trainable parameter $w_i \in \mathbb{R}^d$ and and a fixed output sign $\alpha_i \in \{-1, 1\}$. Each of the n data points of dimension $d-1$ are augmented with a 1, so each sample is of dimension n. Each sample x_j is stored as a row $X \in \mathbb{R}^{n \times d}$ and is associated with a scalar label $y_j \in \mathbb{R}$.

The output of one neuron is: $x \in \mathbb{R}^d \to \max(0, \langle w_i, x \rangle)\alpha_i$, shorthand $\langle w_i, x \rangle_+ \ \alpha_i$.

The output of a network of m neurons on one data point is $f(x) = \sum_{i=1}^{m} \max(0, \langle w_i, x \rangle) \ \alpha_i$

We store the neuron trainable parameter w_i as the columns of $W \in \mathbb{R}^{d \times m}$. The loss for n data points

$$F(W) = \frac{1}{n} \sum_{j=1}^{n} \Big(f(x_j) - y_j \Big)^2$$

Discretized, unregularized gradient descent with $\lambda \in \mathbb{R}^+$ stepsize: (resolving the non differentiable points $\max(0, 0)$ with 0 as gradient)

$$W^{t+1} = W^t - \lambda \nabla F(W^t)$$

Taking $\lambda \to 0$, we get the gradient flow.

Explicit discret gradient:

$$\frac{\partial F}{\partial \boldsymbol{w}_i^t} = \frac{\alpha_i^t}{n} \sum_{j=1}^{n} e_j^t \, s_{i,j}^t \, \boldsymbol{x}_j$$

With real $e_j^t = f(W^t) - y_j$ the "signed error on input j"

With boolean $s_{i,j}^t = \mathbb{1}_{\langle \boldsymbol{w}_i^t, \boldsymbol{x}_j \rangle > 0}$ the "is neuron i activating on datapoint j. The vector $\boldsymbol{s}_i^t \in \{0,1\}^n$ would be the activation pattern of neuron i at time t.

Remark: the gradient of a neuron is a linear combination of the data points it activates.

Discretized wasserstein prox step:

$$W^{t+1} = \underset{W \in \mathbb{R}^{d \times m}}{\arg\min} F(W) + \frac{1}{2\gamma} W_p(W; W^t))$$

EMD($p = 1$):

$$\begin{aligned} W_1(W; W^t) &= \min_{\gamma} \langle \gamma, M \rangle_F \\ \text{s.t. } \gamma \, \mathbf{1} &= W \\ \gamma^\top \, \mathbf{1} &= W^t \\ \gamma &\geq 0 \\ M_{i,j} &= \|w_i - w_j\|_2^2 \end{aligned}$$

2 Simple example 2D setting, grid jko

From one dimensional data, we add a dimension filled with ones to act as a bias for the first layer. The output of one ReLU neuron for one data point $(x, 1) \in \mathbb{R}^2$:

$$w, b, \alpha \in \mathbb{R} \to \max(0, wx + b)\alpha$$

The loss against labels $y_j \in \mathbb{R}$ using squared loss of the whole network of neurons is the double sum:

$$\mathcal{L} = \sum_{j=1}^{n} \left(\left(\sum_{i=1}^{m} \max(0, w_i x_j + b_i)\alpha_i \right) - y_j \right)^2$$

The mean-field limit of this network requires taking an infinite-width ReLU network where parameters are described by a measure μ, and its output by an integral:

$$\int_{\mathbb{R}^2} m((w, b); x) \, \mathrm{d}\mu((w, b))$$

To simplify things, we restrict α_i to $\{-1, 1\}$ and to not be a trainable parameter anymore. We keep the same expressivity(as long as we provide both a positive($\alpha_i = 1$) and negative($\alpha_i = -1$) version of the neuron) but this change will slightly alter the training dynamic in some cases. For example , we can match the output of one neuron (of the original network) by simply scaling the first layer by the seconder layer (α):

$$\max(0, w_i x + b_i)\alpha_i = \max(0, |\alpha_i|(w_i x + b_i)) \ \operatorname{sign}(\alpha_i)$$

Our network with restricted α_i would describe this neuron using only two trainable parameters: ($|alpha_i|w_i, |\alpha_i|b_i$) and fix its sign in the output.

The measure is on the parameter space. In order to do simulations we discretize the parameter space, by taking a uniform grid in $\mathbb{R}^2$ centered on $(0,0)$: $(w_i, b_i)_{i=1,...m}$

We can see that we have the same output and expressivity as the regular ReLU network by taking a measure $\mu = \sum_{i=1}^{m} p_i \delta_{\theta_i = w_i}$ with $(\sum_i p_i = 1)$ and $m((w_i, b_i); x) = \max(0, w_i x + b_i)\alpha_i$, we have this equality:

$$\int_{\mathbb{R}^2} m((w,b);x) \mathrm{d}\mu((w,b)) = \sum_{i=1}^{m} \max(0, w_i x_j + b_i)\alpha_i p_i$$

In this case, the first layer is fixed: the change of direction($\frac{-b_i}{w_i}$) and slope (w_i) of a neuron is described by a mass displacement from point A to point B.

The movemement is described by a PDE and simulated on a grid. Each point i of the grid has a weight $p_i \in \mathbb{R}$, and as a whole $p \in \mathbb{R}^m$ is the discretized distribution.

The same wasserstein gradient flow can be computed by this step:

$$\mu(t+1) = \underset{\mu \in \mathcal{M}(\Theta)}{\arg\min}\, F(\mu) + \frac{1}{2\gamma} W_2(\mu; \mu(t))$$

We tried different ways of computing the Wasserstein Gradient Flow.

- JKO stepping: entropic approximation on a fixed grid. Pros: not very dependant on dimension d. Cons: add another loop and more parameters to fine tune, introduce diffusion.
- Sliced Wasserstein: Pros: midly dependant on d without diffusion. Differentiable with pytorch. Cons: Parameters to tune, distance to true WS distance has to be studied
- Direct EMD distance from POT library. Pros: differentiable with pytorch. Cons: Might be slow with d

Preliminary results using the EMD distance indicate no particular differences between the gradient flow and the wasserstein gradient flow.

2.1 JKO stepping with Dykstra's algorithm

$$\begin{aligned} p_{t+1} &:= \mathrm{Prox}_{\tau f}^{W_\gamma}(p_t) \\ &= \underset{p \in \text{simplex}}{\arg\min}\, W_\gamma(p,q) + \tau f(p) \\ &= \underset{p \in \text{simplex}}{\arg\min} \left(\min_{\pi \in \Pi(p,q)} \langle c, \pi \rangle + \gamma E(\pi) \right) + \tau f(p) \end{aligned}$$

Where π is a mapping, c the ground cost for every point on the grid. When the ground cost between two points in the euclidian space is $c_{i,j} = \|x_i - x_j\|^2$, (and $\gamma = 0$, f smooth...), this scheme formaly discretize the above mentionned PDE.

To do the step above, we'll use a bregman splitting approach that replace the single implicit W_γ proximal step by many iterative KL implicit proximal steps. Specifically(?) Dykstra's algorithm for JKO stepping. This involve using the gibbs kernel:$\xi = e^{-\frac{c}{\gamma}} \in \mathbb{R}_{+,*}^{N \times N}$

3 Classic setup

Data $x_j \in \mathbb{R}^d$ and labels $y_j \in \mathbb{R}$, $j = 1, .., n$

Algorithm 1 JKOstep

$p \leftarrow p_0 \in \mathbb{R}^m$
$q_{\text{norm}} \leftarrow \|p\|^2$
$a, b \leftarrow \mathbf{1}, \mathbf{1} \in \mathbb{R}^m$ ▷ Initialize vectors with ones
for $i \leftarrow 1$ **to** T **do**
    $p \leftarrow \text{prox}^{\text{KL}}_{\tau/\gamma}(\xi b)$
    $a \leftarrow p/(\xi b)$
    ConstrEven $\leftarrow \frac{\|b\cdot(\xi a)-q\|}{q_{\text{norm}}}$
    $b \leftarrow q/(\xi a)$
    ConstrOdd $\leftarrow \frac{\|a\cdot(\xi b)-p\|}{q_{\text{norm}}}$
    if ConstrOdd < tol **and** ConstrEven < tol **then**
        break
    end if
end for

First layer $w_i \in \mathbb{R}^d$, second layer $\alpha_i \in \mathbb{R}$, $i = 1, .., m$

$\gamma > 0$ step-size, β regularization

$$\mathcal{L}(W, \alpha) = \sum_{j=1}^{n} \Bigg(\underbrace{\sum_{i=1}^{m} \max(0, w_i^\top x_j)\alpha_i}_{\text{Network's Output}} - y_j \Bigg)^2 + \underbrace{\lambda \sum_{i=1}^{m} \|w_i\|_2^2 + \alpha_i^2}_{\text{Weight Decay}}$$

Discret time.

Full-batch gradient descent

$$(W, \alpha)_{t+1} = (W, \alpha)_t - \gamma \nabla \mathcal{L}((W, \alpha)_t)$$

Implicit

$$\theta_{t+1} = \arg\min_{\theta} \mathcal{L}(\theta) + \frac{1}{2\gamma}\|\theta - \theta_t\|$$

Continuous time.

Taking $\gamma \to 0$, we get the gradient flow: $\frac{\mathrm{d}\theta_t}{\mathrm{d}t} = -\nabla \mathcal{L}(\theta_t)$. We make ReLU differentiable with $\sigma'(0) = 0$ as justified in (Boursier et al.).

4 Infinite width, using a measure: mean-field

Mean-field limit(Chizat & Bach): For a sufficiently large width, the training dynamics of a NN can be coupled with the evolution of a probability distribution described by a PDE.

If [...] converges, with $m \to \infty$ (many-particle limit), our particles of interest converges to a Wasserstein gradient flow of F:

$$\partial \mu_t = -\text{div}(v_t \mu_t) \text{ where } v_t \in -\partial F'(\mu_t)$$

$$\int_\Theta m(\theta; x) \mathrm{d}\mu(\theta) = \frac{1}{m} \sum_{i=1}^{m} \langle \boldsymbol{w}_i, \boldsymbol{x}_j \rangle_+ \alpha_i$$

Different ways to use a measure to represent the neurons of a two layer network:

- $\Theta = \mathbb{R}^d \times \mathbb{R}$, measure $\mu = \frac{1}{m}\sum_{i=1}^m \delta_{\theta_i=(w_i,\alpha_i)}$, output of one neuron $m(\theta = (\boldsymbol{w}, \alpha); \boldsymbol{x}) = \langle \boldsymbol{x}, \boldsymbol{w}\rangle_+ \alpha$: (works, output matches discrete)
- $\Theta = \mathbb{R}^d$, measure $\mu = \frac{1}{m}\sum_{i=1}^m \alpha_i \delta_{\theta_i=\boldsymbol{w}_i}$ output of one neuron $m(\theta = \boldsymbol{w}; \boldsymbol{x}) = \langle \boldsymbol{x}, \boldsymbol{w}\rangle_+$ (works)
- $\Theta = \mathbb{R}^d \times \mathbb{R}^d$, output of one neuron $m(\tilde{\boldsymbol{w}}_+, \tilde{\boldsymbol{w}}_-, \boldsymbol{x}) = \langle \tilde{\boldsymbol{w}}_+, \boldsymbol{x}\rangle - \langle \tilde{\boldsymbol{w}}_-, \boldsymbol{x}\rangle$ (works, separate neg and positive)
- $\Theta = (S^{d-1} \times \mathbb{R})$, output of one neuron $m((\boldsymbol{d}, \tilde{\alpha}); \boldsymbol{x}) = \tilde{\alpha}\langle \boldsymbol{d}, \boldsymbol{x}\rangle = \tilde{\alpha}\ \mathbb{1}_{\langle \boldsymbol{d}, \boldsymbol{x}\rangle > 0}$ (works), mapping: $\boldsymbol{d} = \frac{\boldsymbol{w}}{\|\boldsymbol{w}\|}$ and $\tilde{\alpha} = \|\boldsymbol{w}\|\alpha$. Gradient are not equal to discrete.

4.1 Algorithm, discretize the measure's space

Take a grid of N points in Θ, we can match the notation above by taking a neuron for each point of the grid $m = N$.

$$\mu(t+1) = \arg\min_{\mu \in \mathcal{M}(\Theta)} F(\mu) + \frac{1}{2\gamma} W_2(\mu; \mu(t))$$

4.2 JKO

What we compute by using the entropic JKO flow iterations.

$$\begin{aligned}
\forall t > 0, p_{t+1} &:= \mathrm{Prox}_{\tau f}^{W_\gamma}(p_t) \\
&= \arg\min_{p \in \text{simplex}} W_\gamma(p, q) + \tau f(p) \\
&= \arg\min_{p \in \text{simplex}} \left(\min_{\pi \in \Pi(p,q)} \langle c, \pi\rangle + \gamma E(\pi) \right) + \tau f(p)
\end{aligned}$$

- Meta Optimal Transport (paper) and (code git): InputConvexNN to predict solution of OT problem
- JKOnet (paper) and (code git):
  - /models -> sinkhorn loss defined in loss.py, differentiable loop in fixed point.py
  - next step: trying to create the right Geometry object from OTT library, which is what's used for sinkhorn

4.3 Papers

The algo we try to implement

Paper with a specific case that doesn't match ours:

In the future, large-scale waserstein gradient flows

4.3.1 Grid problems

The grid currently dictate the neuron's scale, giving multiple choices. One solution: duplicate each neuron, make one with a small scale and one with a very big scale.

References

Etienne Boursier, Loucas Pillaud-Vivien, and Nicolas Flammarion. Gradient flow dynamics of shallow relu networks for square loss and orthogonal inputs. URL http://arxiv.org/abs/2206.00939.

Lénaïc Chizat and Francis R. Bach. On the global convergence of gradient descent for over-parameterized models using optimal transport. In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett (eds.), *Adv. Neural Inf. Process. Syst. 31 Annu. Conf. Neural Inf. Process. Syst. 2018 NeurIPS 2018 Dec. 3-8 2018 Montr. Can.*, pp. 3040–3050. URL https://proceedings.neurips.cc/paper/2018/hash/a1afc58c6ca9540d057299ec3016d726-Abstract.html.