Wasserstein gradient flow of two-layers ReLU networks

Victor Mercklé
June 21, 2024

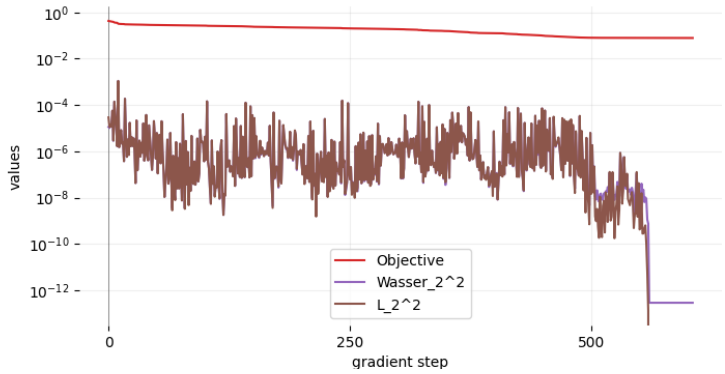
Contents

1 Objective value vs iterates norm

1.1 Early experiments

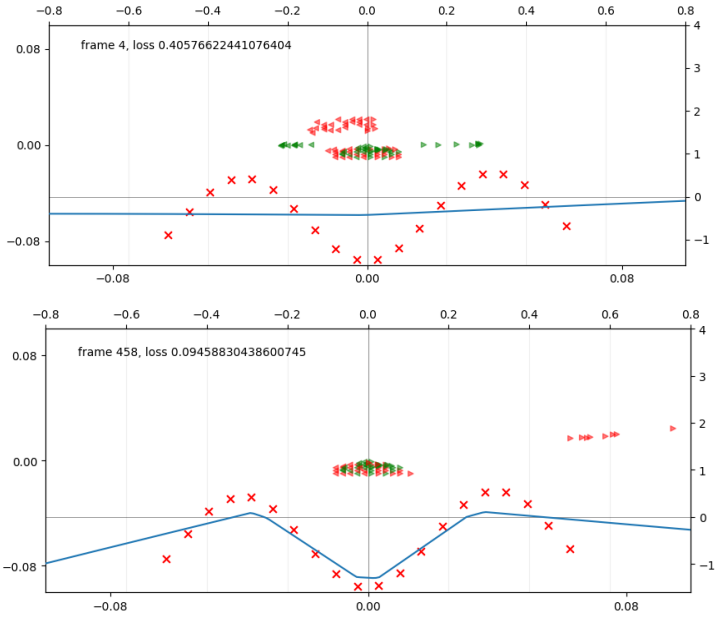
1.1.1 proximal point

We run proximal point which optimize the objective $F(W^{t+1})$ and the distance between two iterates $d(W^{t+1}, W^{(t)})$. We plot those two values for $\gamma = 10$, and for Frobenius distance and Wasserstein distance as defined later.



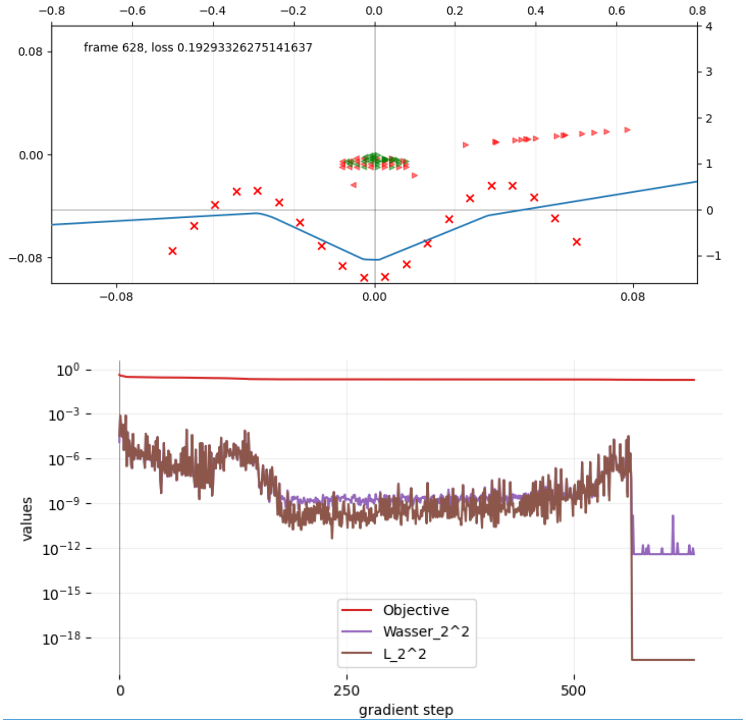
It looks like they are exactly the same. We also plot the difference between the two distances to confirm.

Start and 'end' of convergence for proximal point:



It's a relatively large stepsize but not an easy problem. The triangle of neurons that do not move are simply neurons that do not activate any data. Initialization is a random grid, with an output norm close to zero.

1.1.2 WASSERSTEIN DESCENT



The early trajectory is very similar, but then it looks stuck for a while.

Caveat: prox solution is not accurate due to small number of inner iterations.

2 Accurate prox simulations

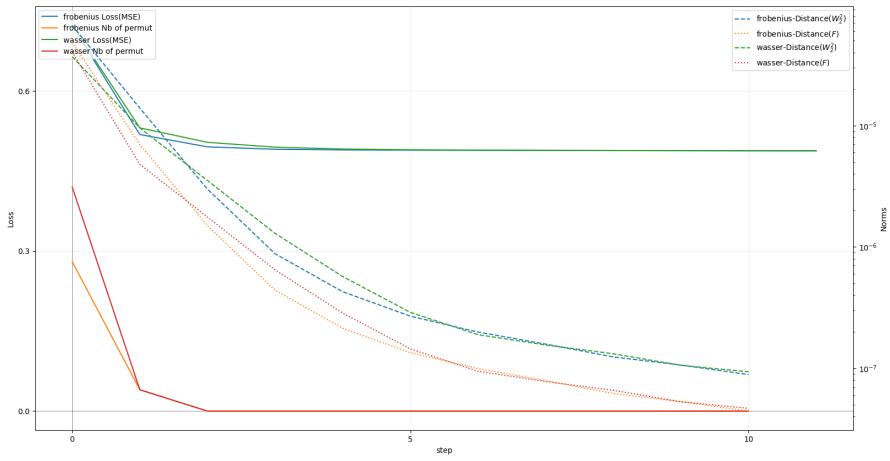
$m = 50$(scale 10^{-2}), $n = 5$(some usual $[-1,1]$ sinus data), we compute 10^6 prox inner iterations (the bigger the stepsize the larger the step size might need to be) for both the frobenius and the wasserstein distance.

In blue/green filled lines we plot the normal MSE loss of the network (for frobenius proxpoint and wasserstein proxpoint)

In dotted/dashed lines we compute $d(W^t, W^{(t+1)})$ (distance between iterates, which is one of the two objectives minimized at each step) for d =wasserstein and d =frobenius.

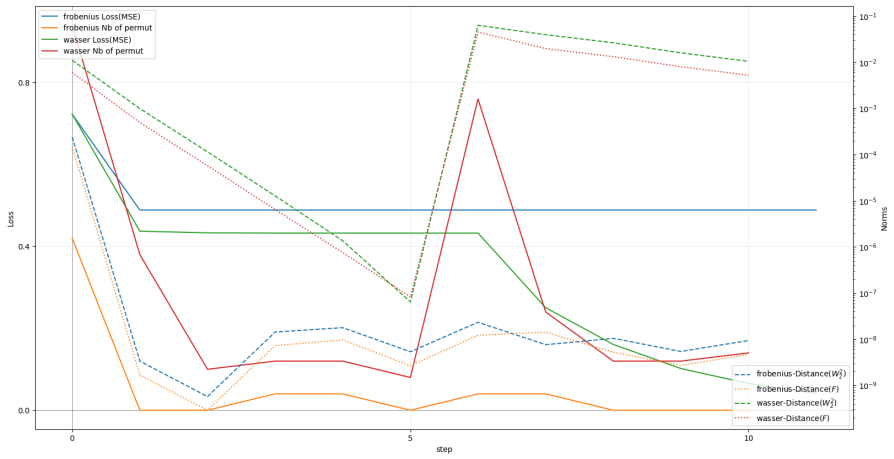
Because the only difference between wasserstein and frobenius is the assignement (which neuron/particle is assigned which particle to compute the distance), we also plot in thick green and red lines the number of permutations of the optimal mapping between two iterates. (1 = every particles has been permuted, 0 = each particle is associated to itself)

For $\gamma = 10^{-3}$ (small stepsize):



The 3 lines associated with wasserstein descent are very close to the 3 lines of frobenius descent, as expected.

For $\gamma = 10^0$ (big stepsize). This is one example where the convergence is very different. Wasserstein descent converges to 0 while Frobenius descent gets stuck on a bad local minima. We do the same plot as before.

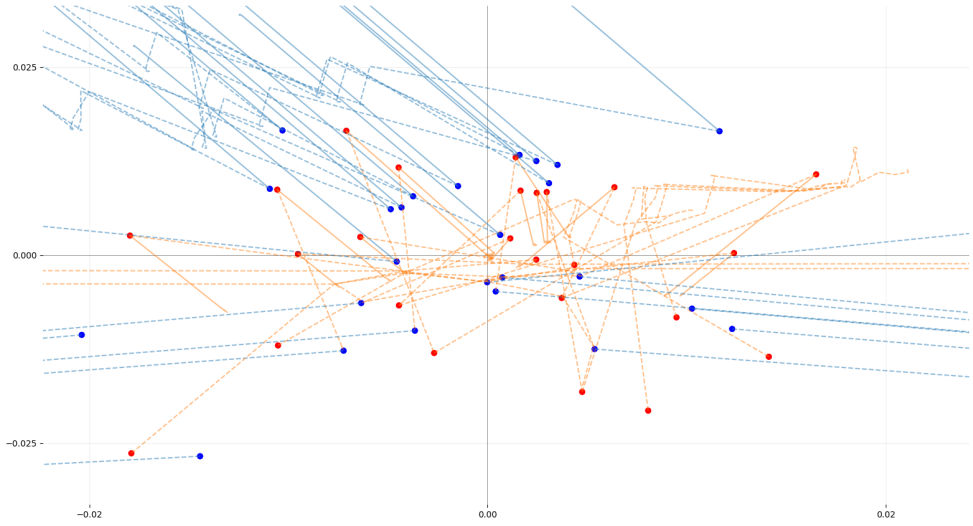


We can see the loss is completely different after 10 steps (it's normal to converge very quickly as the stepsize of the prox is large).

One thing to note, if we can focus on wasserstein descent lines (dashed green and dotted red), the distance between iterates in frobenius or wasserstein is very close. However we can see in filled red that the number of permutations is close to 100% at some steps.

This indicates that the actual distance value is not as important as the actual trajectories of the neurones which are decided by the gradient of the wasserstein distance used in the prox.

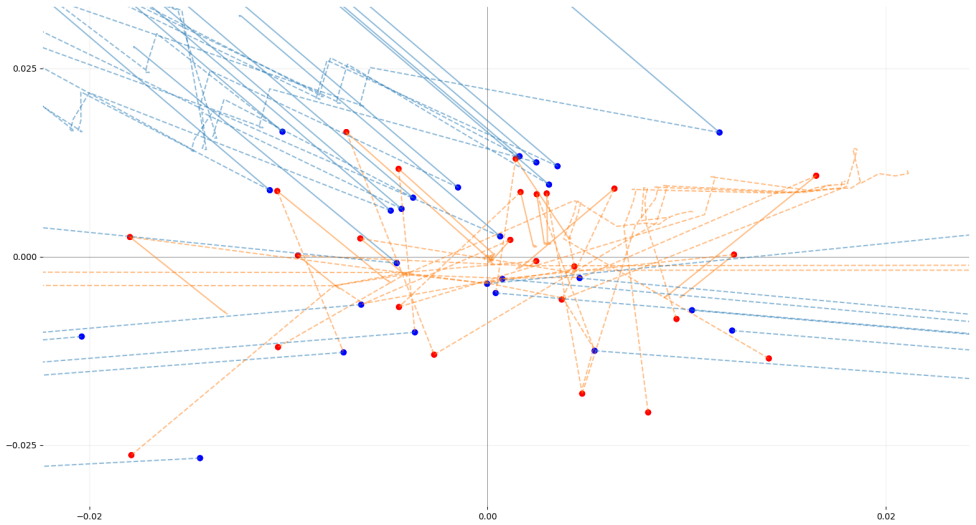
We plot the trajectories of each neuron (dashed lines for wasserstein descent and filled lines for frobenius descent) for the 10 prox steps. Circles denotes the initial positions of the neurons, blue for negative neurons and red for positives neurons.



The trajectories are indeed different between the two descents. One major difference: neurons in the bottom quadrants are not moved at all by the frobenius descent, but in the wasserstein descent they are moved, they exchange positions (we can see two (dashed) trajectories very close going to and from the same initial neuron).

Those neurons (bottom/center quadrants) do not activate any datapoint and as such their differential is 0, so they cannot possibly move under prox descent. If weight decay is used, they will converge to 0. This completely differ from wasserstein descent, so the differential of the wasserstein distance must be different.

We'll focus on one proximal step by showing the first few *inner* iterations for the two descents. We take two particles with different gradients. (this could happen near a non-differentiability, or as we see in the simulations if we can switch positive and negative neurons)



The inner iterations are dictated by the gradient of the objective $\frac{\partial F}{\partial w_i}$ to minimize the loss, and by gradient of the distance to the first iterate ($W^{(0)}$). The frobenius proximal point goes as usual, the distance gradient pulling the particle back to its initial position. However the wasserstein distance ends up pulling the particles to the other neuron's initial positions.

Remark(not 100% sure): in the mean field paper, the only results on ReLU are obtained by computing the wasserstein distance on two separate sets (i.e. you cannot assign a positive neuron to a negative neuron)

Implementations details: Frobenius is ran on one GPU(15min per run) while wasserstein is ran on a single CPU core (30min per run) as that was found to be the most efficient to run multiple experiments. Results for $\gamma = 10^a$, $a = [0, -1, -2, -3, -4, -5]$ are available but do not differ much.

3 Implemented descents

We only update the first layer $W^{(0)} \in \Theta = \mathbb{R}^{d \times m}$ for simplicity. $F : \Theta \to \mathbb{R}^+$ is the objective to minimize.

$$\min_{W \in \Theta} F(W)$$

3.1 Gradient descent

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

3.2 Proximal point

$d : \Theta \times \Theta \to \mathbb{R}^+$ a distance between two parameters

$$W^{(t+1)} = \underset{V \in \Theta}{\arg\min} \;\; F(V) + \frac{1}{2\,\gamma}\, d(V, W^{(t)})$$

To solve each arg min, we use a gradient descent on V. $V^{(0)} = W^{(t)}$

$$V^{(k+1)} = V^{(k)} - \beta\left(\nabla F(V^{(k)}) - \frac{1}{\gamma}\nabla d(V^{(k)}, W^{(t)})\right)$$

3.2.1 DIST = L2 SQ, FROBENIUS

Frobenius norm ($L_{2,2}$): $\|W\|_{2,2}^2 = \|W\|_F^2 = \sum_{i=1}^m \sum_{j=1}^d W_{i,j}^2 = \sum_{i=1}^m \|W_i\|_2^2$

$$d : (V, W) \in \Theta \times \Theta \to \frac{1}{m}\sum_{i=1}^{m} \|V_i - W_i\|_2^2$$

To sum up, we're solving this

$$W^{(t+1)} = \underset{V \in \Theta}{\arg\min} \;\; F(V) + \frac{1}{2\,\gamma}\,\frac{1}{m}\sum_{i=1}^{m} \|V_i - W_i\|_2^2$$

Using gradient descent for arg min, one inner step (with inner step size β) on V with $V^{(0)} = W^{(t)}$ is

$$V^{(k+1)} = V^{(k)} - \beta\left(\nabla F(V^{(k)}) + \frac{1}{m\,\gamma}(V^k - W^{(t)})\right)$$

$$V^{(k+1)} = V^{(k)}\left(1 - \frac{\beta}{m\,\gamma}\right) + \frac{\beta}{m\,\gamma} W^{(t)} - \beta\nabla F(V^{(k)})$$

which is kinda the same thing as this(only for $\gamma \to 0$) (trust region problem equivalence with proximal problem) $V^{(0)} = W^{(t)}$

$$V^{(k+1)} = W^{(t)} - \gamma\nabla F(V^{(k)})$$

3.2.2 DIST = WASSERSTEIN2 SQ

distance definition.

$$W^{(t+1)} = \underset{V \in \Theta}{\arg\min} \;\; F(V) + \frac{1}{2\,\gamma}\, d(V, W^{(t)})$$

With $V, W \in \mathbb{R}^d$ parameters.

p-Wasserstein distance between two measures $\mu, \nu \in \mathcal{P}(\mathbb{R}^d)$:

$$W_p(\mu, \nu)^p = \inf_{\gamma \in \Pi(\mu,\nu)} \int |y - x|^p \mathrm{d}\gamma(x, y)$$

Π set of probability measures on $\mathbb{R}^d$ with marginals μ and ν.

We take two sum of diracs measures $\mu = \frac{1}{m}\sum_{i=1}^m \delta_{V_i}$ and $\nu = \frac{1}{m}\sum_{i=1}^m \delta_{W_i}$. Then in that case W_2 is

$$W_2(V,W)^2 = \min_{\sigma \in S_n} \frac{1}{m} \sum_{i=1}^{m} \|V_i - W_{\sigma(i)}\|_2^2$$

With σ a permutation. This is the linear assignment problem which can be solved by the hungarian algorithm. It always has a solution so inf=min

This is equivalent to what `OT.emd(d=squareeuclidian)` solves:

$$\begin{aligned} W_2^2(W; W^t) &= \min_{\gamma} \langle \gamma, M \rangle_F \\ \text{s.t. } \gamma\, \mathbf{1} &= W \\ \gamma^\top\, \mathbf{1} &= W^t \\ \gamma &\geq 0 \\ M_{i,j} &= \|w_i - w_j\|_2^2 \end{aligned}$$

However, we only need the gradient of the distance. I haven't yet read what it is exactly.

Small stepsize.

To sum up, for wasserstein descent we have this step

$$W^{(t+1)} = \operatorname*{arg\,min}_{V \in \Theta} \; F(V) + \frac{1}{2\,\gamma\,m} \min_{\sigma \in S_n} \sum_{i=1}^{m} \|V_i - W_{\sigma(i)}\|_2^2$$

If γ is small, the candidate V will be close to $W^{(t)}$, and each V_i will be close to $W_i^{(t)}$. Therefore the optimal assignment is the identity.

$$W^{(t+1)} = \operatorname*{arg\,min}_{V \in \Theta} \; F(V) + \frac{1}{2\,\gamma\,m} \sum_{i=1}^{m} \|V_i - W_i\|_2^2$$

Which is exactly the proximal point. Additionnaly, as the step size gets smaller, proximal point converges to the gradient flow (just like gradient descent).

4 Small vs Big initialization in ReLU

Why is it that as the scale gets smaller, the training dynamic consist of an alignment phase and then a convergence phase?

Notation: Take a neuron $w \in \mathbb{R}^d$. The norm of the neuron is $\|w\|_2 = \sqrt{\sum_{i=1}^d w_i^2}$, the direction of a neuron is a vector of norm 1: $\frac{w}{\|w\|_2}$.

A training step is approximately $w^{(t+1)} = w^{(t)} + \gamma\, \mathbf{v}$ with γ a coefficient correlated with the current (t) output, labels, error and step size. However, it is not directly correlated to the norm of the neuron.

$\mathbf{v} \in \mathbb{R}^d$ is a partial sum of training examples. It has its own norm and direction.

If $\|w^{(0)}\|$ is large, $w^{(1)}$ will be close to $w^{(0)} + \gamma\, \mathbf{v}$.

If $\|w^{(0)}\|$ is close to 0, $w^{(1)}$ will be close to equal to $\gamma\, \mathbf{v}$ as every coefficient of w is negligible compared to the update. Therefore, its direction will (after some updates) be dominated by $\mathbf{v}$. The same for every neuron that activate the same data points. (Since activation pattern is entirely decided by direction, activation pattern will all converge to extremal vectors..)

Refs: (expés, some results) (Maennel et al., 2018), (the orthogonal paper) (Boursier et al., 2022). (incremental learning) (Berthier, 2022), (scaling path) (Neumayer et al., 2023).

5 Mean field

5.1 direct result

Minimization of a linear combination of ('neurons' $\{\phi(\theta)\}_{\theta\in\Theta}$) through an unknown measure μ: $J^* = \min_{\mu\in\mathcal{M}(\Theta)} J(\mu) = R(\int \phi \mathrm{d}\mu) + G(\mu)$

With R a convex loss function and G a convex regularizer, $\mathcal{M}(\Theta)$ the set of signed measures on the parameter space Θ.

Discretize the measure into m particles: $\min_{w\in\mathbb{R}^m\ \theta\in\Theta^m} J_m(w,\theta) = J(\frac{1}{m}\sum_{i=1}^m w_i \delta_{\theta_i})$

Proved: if WGF cvg, it cvg to global minimizer. If $(w^{(m)}(t), \theta^{(m)}(t))_{t>0}$ are gradient flows for J_m then, with the corresponding measure $\mu_{m,t} = \frac{1}{m}\sum_{i=1}^m w_i^{(m)}(t)\sigma_{\theta_i^{(m)}(t)}$ (a WGF), $J(\mu_{m,t})$ cvg (with $m, t \to \inf$) to global minimizer of J.

p-Wasserstein distance between two measures $\mu, \nu \in \mathcal{P}(\mathbb{R}^d)$: $W_p(\mu,\nu)^p = \min_{\gamma\in\Pi(\mu,\nu)} \int |y - x|^p \mathrm{d}\gamma(x,y)$

5.2 simple result

Cast the parameters $w \in \mathcal{W}^{d+1}$ into $r \in \mathbb{R}, \eta \in \mathcal{S}^d = w \in \mathbb{R}^{d+1}, \|w\|_2 = 1$ with $w = r\ \eta$. The two flows will have exactly the same dynamics.

Then measure $\nu = \frac{1}{m}\sum_{j=1}^m r_j^2 \sigma_\nu$ satisfy this PDE : $\partial_t \nu_t(\eta) = -4J(\eta|\nu_t) + \mathrm{div}(\nu_t(\eta)\nabla J(\eta|\nu_t))$ (so it's a WGF)

Prediction functions : $h = \frac{1}{m}\sum_{j=1}^m \psi(w_j) = \frac{1}{m}\sum_{j=1}^m r_j^2 \psi(\nu_j)$

Theorem 1: if by taking $m \leftarrow \inf$ at $t = 0$, ν converges to ν_0, then for any t, $\nu_{m,t}$ cvg to the unique WGF ν_t.

Theorem 2: Take a ν_0 with a support that includes all directions at initialization, if WGF ν_t cvg, it's to a global optimum.

6 Math introduction

6.1 Gradient Flow

Given a smooth function $a \to F(a)$, the gradient flow is gradient descent algorithm

$a^{l+1} = a^l - \gamma \nabla F(a^l)$

with a small enough γ. If F is not smooth, the gradient flow is the proximal-point algorithm

$a^{l+1} = \mathrm{Prox}_{\gamma F}^{\|\cdot\|}(a^{(l)} = \arg\min_a \frac{1}{2}\|a - a^{(l)}\|^2 + \gamma F(a)$

with a small enough γ.

If F is defined on histograms, it makes sense to use the wasserstein distance W^p

7 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(POT library), d=squareeuclidian

$$\begin{aligned} W_2^2(W; W^t) &= \min_{\gamma} \langle \gamma, M \rangle_F \\ \text{s.t. } \gamma\ \mathbf{1} &= W \\ \gamma^\top\ \mathbf{1} &= W^t \\ \gamma &\geq 0 \\ M_{i,j} &= \|w_i - w_j\|_2^2 \end{aligned}$$

8 3 points example

- Data: slope : $a = 2$
- Data: $\begin{pmatrix} X_1 \\ X_2 \\ X_3 \end{pmatrix} = \begin{pmatrix} x_1 & 1 \\ x_2 & 1 \\ x_3 & 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \\ 4 \end{pmatrix}$, $Y = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} ax_1^1 \\ ax_2^1 \\ ax_3^1 \end{pmatrix}$
- Loss: $F(W) = \frac{1}{2n} \sum_{j=1}^{n} \Big(\max(0, \langle w_1, X_j \rangle \alpha_1 - y_j \Big)^2$
- Gradient: $\nabla F(W) = \Big(\frac{\partial F}{\partial w_1} \Big) = \frac{\alpha_1}{n} \sum_{j=1}^{n} e_j s_{1,j} X_j$
- Algo: $W^{t+1} = W^t - \nabla F(W^t)$
- Initialization: $W^0 = (w_1) = (1 \quad 0)$, $\alpha = (\alpha_1) = (1)$

Run gradient descent:

- Iteration 0:
  - $W^0 = (1 \quad 0)$
  - $s_1 = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$
  - $e = \begin{pmatrix} \langle w_1, x_1 \rangle \alpha_1 - y_1 \\ \langle w_1, x_2 \rangle \alpha_1 - y_2 \\ \langle w_1, x_3 \rangle \alpha_1 - y_3 \end{pmatrix} = \begin{pmatrix} \langle (1 \quad 0), (1 \quad 1) \rangle - 2 \\ \langle (1 \quad 0), (2 \quad 1) \rangle - 4 \\ \langle (1 \quad 0), (4 \quad 1) \rangle - 8 \end{pmatrix} = \begin{pmatrix} 1-2 \\ 2-4 \\ 4-8 \end{pmatrix} = \begin{pmatrix} -1 \\ -2 \\ -4 \end{pmatrix}$
  - $e_j = \langle w_1, X_j \rangle \alpha_1 - y_j = w_1^1 x_j + w_1^2 - a x_j = x_j(w_1^1 - a) + w_1^2$
  - $W^1 = \begin{pmatrix} w_1^1 & w_1^2 \end{pmatrix} - \frac{1}{n} \sum_{j=1}^3 (x_j(w_1^1 - a) + w_1^2) X_j$
  - $W^1 = \begin{pmatrix} w_1^1 & w_1^2 \end{pmatrix} - \frac{1}{n} \sum_{j=1}^3 \begin{pmatrix} x_j^2(w_1^1 - a) + x_j w_1^2 & x_j(w_1^1 - a) + w_1^2 \end{pmatrix}$
  - $W^1 = \begin{pmatrix} w_1^1 - \frac{w_1^2}{n}(x_1 + x_2 + x_3) - \frac{(w_1^1 - a)}{n}(x_1^2 + x_2^2 + x_3^2) & ok \end{pmatrix}$
  - $b = \frac{x_1 + x_2 + x_3}{n} = 7/3, c = \frac{x_1^2 + x_2^2 + x_3^2}{n} = 21/3 = 7$
  - $W^1 = \begin{pmatrix} w_1^1 - w_1^2 b - (w_1^1 - a)c & w_1^2 - (w_1^1 - a)b + w_1^2 \end{pmatrix}$
  - $W^1 = \begin{pmatrix} w_1^1 - w_1^2 b - w_1^1 c + ac & w_1^2 - w_1^1 b + w_1^2 + ab \end{pmatrix}$
  - $W^1 = (1 - 0 - 7 + 14 \quad 0 - 7/3 + 0 + 14/3) = (8 \quad 7/3)$
  - $W^1 = W^0 + 7 \begin{pmatrix} 1 & \frac{1}{3} \end{pmatrix}$
  - $X_1 + 2X_2 + 4X_3 = (1 + 4 + 16 \quad 1 + 2 + 4) = (21 \quad 7) = 7 \times 3 \begin{pmatrix} 1 & \frac{1}{3} \end{pmatrix}$

9 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 $(0, 0)$: $(w_i, b_i)_{i=1,\dots m}$

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.

9.1 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:$\xi = e^{-\frac{c}{\gamma}} \in \mathbb{R}_{+,*}^{N \times N}$

10 Classic setup

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

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

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

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

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

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., 2022).

11 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.

11.1 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 $m = N$.

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

11.2 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

11.3 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

11.3.1 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

Raphael Berthier. Incremental learning in diagonal linear networks. *arXiv preprint arXiv:1803.08367*, 2022. URL https://jmlr.org/papers/volume24/22-1395/22-1395.pdf.

Etienne Boursier, Loucas Pillaud-Vivien, and Nicolas Flammarion. Gradient flow dynamics of shallow relu networks for square loss and orthogonal inputs. *arXiv preprint arXiv:1803.08367*, 2022. doi: 10.48550/arXiv.2206.00939. 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.

Hartmut Maennel, Olivier Bousquet, and Sylvain Gelly. Gradient descent quantizes relu network features. *arXiv preprint arXiv:1803.08367*, 2018.

Sebastian Neumayer, Lénaïc Chizat, and Michael Unser. On the effect of initialization: The scaling path of 2-layer neural networks. *arXiv preprint arXiv:1803.08367*, 2023. URL http://arxiv.org/abs/2303.17805.