Fine-tuning STRUCTURES25 model with implicit loss

Academic Year: 2024–2025
Master's Program: M2 Mathématiques, Modélisation et Apprentissage
Name: Łukasz Adamowicz

Internship Supervisor: Professor Fred Hamprecht
Email: fred.hamprecht@iwr.uni-heidelberg.de

Hosting institution:
Heidelberg University,
Interdisciplinary Center for Scientific Computing
UFR Contact:
Alexis Glaunes
alexis.glaunes@parisdescartes.fr

August 28, 2025

Abstract

During my internship at Hamprecht Lab, an investigation was conducted on fine-tuning deep learning models using a loss function defined in an implicit way.

The goal of the internship was to adapt and implement the approach from the preprint [8] to a setting used in Hamprecht lab.

Two approaches to gradient computation were implemented: equilibrium propagation, and Jacobian approach. A stability issue arose, but was subsequently resolved. The results indicated that the equilibrium propagation approach could be viable, while the Jacobian approach proved ineffective.

Contents

1 Lab presentation

ScientificAI group is located at Interdisciplinary Center for Scientific Computing (IWR), which is an interdisciplinary research center for Scientific Computing. The group is currently focusing on applying deep learning to orbital-free density functional theory (OF-DFT).

To accurately predict molecular properties, quantum mechanics formalism can be employed to solve for the electronic structure, but for big molecules this approach is not tractable.

However, a theorem by Hohenberg-Kohn has shown that all that's required to know the electronic structure is the ground state electron density ρ_{gs} and that there exists an energy functional such that

$$\rho_{gs} = \arg\min_{\rho} E[\rho].$$

Exact form of the energy functional $E[\rho]$ remains unknown. The goal of project is to approximate the energy functional $E[\rho]$ using a deep learning model (a surrogate functional) denoted as $E(\theta, \rho)$, which is then used to obtain ground state density by gradient descent.

If successful, this would enable the computation of the ground-state electron density with a complexity of $O(n)$, thereby allowing for efficient simulations of large molecules. Since the exact ground state is not always known, the model must be capable of converging from a wide range of initial starting points.

The electron density $\rho(\vec{r})$ is approximated as a linear combination of atom-centered smooth functions ω_i, that is:

$$\rho(\vec{r}) = \sum_{i=1}^{n} p_i \omega_i(\vec{r})$$

The specific number and type of functions ω_i depend on the atoms in the system.

The integral of the electron density yields the total number of electrons in the system:

$$N = \int_{\mathbb{R}^3} \rho(\vec{r}) d\vec{r} = \sum_{i=1}^{n} p_i \int_{\mathbb{R}^3} \omega_i(\vec{r}) d\vec{r}.$$

By defining a vector $w = (w_1, \dots, w_n)$, where each component is:

$$w_i = \int_{\mathbb{R}^3} \omega_i(\vec{r}) d\vec{r},$$

we can express the electron number as:

$$\langle w, p \rangle = N.$$

2 Problem statement

Notation: Let $E(\theta, M, p)$ denote the energy model, with $\theta \in \mathbb{R}^k$ being model parameters, M the molecule information, and $p = (p_1, \dots, p_n) \in \mathbb{R}^n$ electron density coefficients. We assume that $E(\theta, M, p)$ is smooth with respect to θ and p. By $\frac{d}{d\theta}$ we mean total derivative, while $\frac{\partial}{\partial\theta}$ denotes partial derivative θ, when all other arguments are held constant. Then $\frac{\partial E}{\partial p}$ is a **row vector** of size n, hessian $\frac{\partial^2 E}{\partial p^2}$ is a matrix of size $n \times n$, and $\frac{\partial^2 E}{\partial\theta\partial p}$ is a matrix of size $n \times k$.

2.1 Goal of the project

The goal of the project was to investigate and minimize the following loss

$$\min_\theta \sum_i \mathcal{L}(p_\theta^{M_i}), \tag{1}$$

where

$$p_\theta^{M_i} := \operatorname*{arg\,min}_{p:\langle w,p\rangle = N_i} E(\theta, M_i, p) \tag{2}$$

and $\mathcal{L}_{M_i}(p) = \frac{1}{2}\|p - p_{M_i}\|^2$ is the standard L_2 loss.

In the rest of the report I will drop index M_i. The reason is it will simplify the notation significantly, while not impacting the gradient derivation in any notable way. Therefore I will write $E(\theta, p)$ instead of $E(\theta, M, p)$ and the same for p_θ, $\mathcal{L}(p)$ and so on. The quantity p_θ is a **fixed point**. This optimization problem falls under the domain of **bilevel optimization**.

The main problem lies in computing the gradient of $\mathcal{L}(p_\theta)$ and it will be the main subject of this report.

Remark 1 (Motivating the loss function)**.** *Finding a ground state electron density of a molecule is a fundamental problem in quantum chemistry. Usual methods are too computationally expensive for big systems. Surrogate functionals are used to approximate true energy functional and through that approximate the ground state by computing fixed point of the surrogate functional. In that light, minimizing the error between ground state density and fixed point of the model makes a certain amount of sense.*

One molecule

Let w be a non-zero vector and $N > 0$ a positive number. We consider the problem of minimizing

$$\min_\theta \mathcal{L}(p_\theta), \tag{3}$$

with

$$p_\theta = \operatorname*{arg\,min}_{\substack{p\in\mathbb{R}^n \\ p:\langle p,w\rangle = N}} E(\theta, p) \tag{4}$$

and

$$\mathcal{L}(p) = \frac{1}{2}\|p - p_{gs}\|^2, \tag{5}$$

where p_{gs} are ground state density coefficients given by the data and satisfy the constraint $\langle p_{gs}, w\rangle = N$.

Direct approach to gradient computation

The most direct approach would be to backpropagate through the trajectory of the fixed point. More precisely, we run gradient descent and obtain a trajectory $p_1, \dots, p_T$, where $p_T \approx p_\theta$. We can store this trajectory in memory and use backpropagation on the loss

$$\mathcal{L}(p_T(\theta)) \approx \mathcal{L}(p_\theta) \tag{6}$$

Although straightforward there are a few issues with this approach.

- backpropagation through the whole trajectory is very memory intensive, therefore fixed point search can be run for very low number of steps.
- the gradient depends on the starting point
- can't apply warm-starting to speed up the training. In theory, for Jacobian and equilibrium propagation approach we can find the fixed point once and reuse it in subsequent runs as a starting point, therefore making training much faster. However, due to stability issues and difficulties of tracking loss, I ended up not using warm-starting.

3 Implicit function theorem

Gradient calculation can be accomplished by using implicit function theorem. In the rest of the document I will refer to computation of the gradient using this method as **Jacobian approach**.

I use the version of implicit function theorem from [10], which in turn is taken from [4].

Theorem 1 (Implicit Function Theorem, [10])**.** *Let $G(\theta, p) : \mathbb{R}^k \times \mathbb{R}^n \to \mathbb{R}^n$ be C^1 and $(\bar{\theta}, \bar{p})$ such that $G(\bar{\theta}, \bar{p}) = 0$. If the Jacobian $\frac{\partial G}{\partial p}(\bar{\theta}, \bar{p})$ is non-singular, then there exists an open neighbourhood U of $\bar{\theta}$ and a function $p(\theta) : U \to \mathbb{R}^n$, such that $G(\theta, p(\theta)) = 0$ and $p(\bar{\theta}) = \bar{p}$. Moreover, $p(\theta)$ is also differentiable with its derivative given by*

$$\frac{\partial p(\theta)}{\partial \theta} = -\left(\frac{\partial G}{\partial p}(\theta, p(\theta))\right)^{-1} \frac{\partial^2 G}{\partial \theta \partial p}(\theta, p(\theta)). \tag{7}$$

In our case $G = \frac{\partial E}{\partial p}$. Since our model is smooth with respect to its arguments and the hessian of E is assumed to be positive definite, the conditions of the theorem are satisfied.

Basis-dependent approach

First, let's consider the problem in an orthogonal basis containing w. Let the basis be $(e_1, \dots, e_{n-1}, w)$. Define $\widetilde{E}(\theta, p) = E(\theta, p_{gs} + p)$ and $p = \sum_{i=1}^{n-1} p_i e_i$. Then $\widetilde{E}(\theta, \cdot)$ is just E restricted to the affine space $p_{gs} + \text{span}(w)^\perp$, and the problem becomes

$$\min_\theta \mathcal{L}(p_\theta), \tag{8}$$

$$p_\theta = \underset{p\in\mathbb{R}^{n-1}}{\arg\min}\, \widetilde{E}(\theta, p) \tag{9}$$

We will find the gradient with the help of the implicit function theorem.

Statement 1. *The gradient of $\mathcal{L}(p_\theta)$ is equal to*

$$\frac{\partial \mathcal{L}(p_\theta)}{\partial\theta} = -(p_\theta - p_{gs})\left(\frac{\partial^2 \widetilde{E}}{\partial p^2}\right)^{-1}\frac{\partial^2 \widetilde{E}}{\partial\theta\partial p} \tag{10}$$

To obtain the gradient of the loss, we start from the chain rule

$$\frac{\partial \mathcal{L}(p_\theta)}{\partial\theta} = \frac{\partial \mathcal{L}}{\partial p}\frac{\partial p_\theta}{\partial\theta}. \tag{11}$$

Since $\frac{\partial \mathcal{L}}{\partial p}(p_\theta) = p_\theta - p_{gs}$, all we need to find is $\frac{\partial p_\theta}{\partial\theta}$.

Statement 2. *The gradient of p_θ is equal to*

$$\frac{\partial p_\theta}{\partial\theta} = -\left(\frac{\partial^2 \widetilde{E}}{\partial p^2}\right)^{-1}\frac{\partial^2 \widetilde{E}}{\partial\theta\partial p}(\theta, p_\theta) \tag{12}$$

Proof. By definition, $\widetilde{E}(\theta, \cdot)$ attains its minimum at p_θ, so it follows that

$$\frac{\partial \widetilde{E}}{\partial p}(\theta, p_\theta) = 0. \tag{13}$$

By taking derivative with respect to θ, we get

$$0 = \frac{\partial}{\partial\theta}\left(\frac{\partial \widetilde{E}}{\partial p}(\theta, p_\theta)\right) = \frac{\partial^2 \widetilde{E}}{\partial\theta\partial p} + \frac{\partial^2 \widetilde{E}}{\partial p^2}\frac{\partial p_\theta}{\partial\theta}. \tag{14}$$

Therefore

$$\frac{\partial p_\theta}{\partial\theta} = -\left(\frac{\partial^2 \widetilde{E}}{\partial p^2}\right)^{-1}\frac{\partial^2 \widetilde{E}}{\partial\theta\partial p}(\theta, p_\theta). \tag{15}$$

Let's denote the Hessian of $\widetilde{E}$ as

$$H := \frac{\partial^2 \widetilde{E}}{\partial p^2}(\theta, p_\theta) \tag{16}$$

□

This gives us the final formula

$$\frac{\partial}{\partial\theta}\mathcal{L}(p_\theta) = -(p_\theta - p_{gs})H^{-1}\left.\frac{\partial^2 \widetilde{E}}{\partial\theta\partial p}\right|_{p=p_\theta,\theta=\theta}. \tag{17}$$

Basis-free approach

Let's reformulate the problem without relying on a specific basis. We want to find

$$\min_{\theta} \mathcal{L}(p_\theta), \tag{18}$$

with

$$p_\theta = \underset{\substack{p \in \mathbb{R}^n \\ p : \langle p, w \rangle = N}}{\arg\min} \; E(\theta, p). \tag{19}$$

To this end we make use of the following lemma.

Lemma 1. *Let $P := I - \frac{ww^T}{w^T w}$ be the projection operator onto subspace $V = span(w)^\perp$. Then the following equality is true*

$$P\left(\frac{\partial E}{\partial p}(\theta, p_\theta)\right) = 0 \tag{20}$$

Proof. Since p_θ is the minimum of $E(\theta, p)$ restricted to $V + p_{gs}$ we have

$$\langle \frac{\partial E}{\partial p}(\theta, p_\theta), u \rangle = 0$$

for any $u \in V$. Taking $u = P\big(\frac{\partial E}{\partial p}(\theta, p_\theta)\big)$ we obtain

$$\langle \frac{\partial E}{\partial p}(\theta, p_\theta), P\big(\frac{\partial E}{\partial p}(\theta, p_\theta)\big) \rangle = \left\| P\big(\frac{\partial E}{\partial p}(\theta, p_\theta)\big) \right\|^2 = 0,$$

which concludes the proof. □

Let's denote $F(\theta, p) := P\left(\frac{\partial E}{\partial p}(\theta, p)\right)$. Then the previous condition can be written as

$$F(\theta, p_\theta) = 0. \tag{21}$$

We take the derivative with respect to θ

$$0 = \frac{dF(\theta, p_\theta)}{d\theta} = \frac{\partial F}{\partial \theta}(\theta, p_\theta) + \frac{\partial F}{\partial p}(\theta, p_\theta)\frac{dp_\theta}{d\theta}. \tag{22}$$

Rearranging we obtain

$$\frac{dp_\theta}{d\theta} = -\left(\frac{\partial F}{\partial p}(\theta, p_\theta)\right)^{-1} \frac{\partial F}{\partial \theta}(\theta, p_\theta). \tag{23}$$

Therefore the full gradient of $\mathcal{L}(p_\theta)$ is equal to

$$\frac{d\mathcal{L}(p_\theta)}{d\theta} = -(p_\theta - p_{gs})\left(\frac{\partial F}{\partial p}(\theta, p_\theta)\right)^{-1} \frac{\partial F}{\partial \theta}(\theta, p_\theta). \tag{24}$$

Remark 2. *The Jacobian $\frac{\partial F}{\partial p}$ is not invertible, since F is a projection.*

$$F(\theta, p) = \sum_{i=1}^{n-1} \langle \frac{\partial E}{\partial p}(\theta, p), e_i \rangle e_i + 0 \cdot w \tag{25}$$

The Jacobian matrix in the basis $(e_0, ..., e_{n-1}, w)$ *can be written as*

$$\left.\frac{\partial F}{\partial p}\right|_{p=p_\theta} = \begin{bmatrix} H & 0 \\ 0 & 0 \end{bmatrix}. \tag{26}$$

and solution to linear regression

$$y = \underset{u \in \mathbb{R}^n}{\operatorname{argmin}} \left\| u \begin{bmatrix} H & 0 \\ 0 & 0 \end{bmatrix} + (p_\theta - p_{true}) \right\|^2 \tag{27}$$

is satisfied for by y *of the form*

$$y = P\big((p_\theta - p_{gs})\big)H^{-1} + \alpha w. \tag{28}$$

Since F *is a projection onto plane perpendicular to* w*, it shouldn't matter which* y *we choose. That is to say*

$$\langle y, \frac{\partial F}{\partial \theta} \rangle = \langle Py, \frac{\partial F}{\partial \theta} \rangle. \tag{29}$$

Therefore, the formula doesn't change, however some algorithms for solving linear systems might not work, if they rely on invertibility.

In the rest of the report I will reference the procedure of finding y vector as **solving the linear equation**.

We can rewrite the formula for the gradient as

$$\frac{d\mathcal{L}(p_\theta)}{d\theta} = y \cdot \left.\frac{\partial F}{\partial \theta}\right|_{p=p_\theta, \theta=\theta} \tag{30}$$

treat y as constant vector and pull it inside partial derivative to obtain

$$\frac{d\mathcal{L}(p_\theta)}{d\theta} = \left.\frac{\partial(\langle y, F \rangle)}{\partial \theta}\right|_{p=p_\theta, \theta=\theta}. \tag{31}$$

The whole procedure for gradient calculation is as below

Algorithm 1 Jacobian approach gradient calculation

```
Require: Data $p_{gs}$, model parameters $\theta$
1: Solve for $p_\theta = \arg\min_{\substack{p \in \mathbb{R}^n \\ p:\langle p, w\rangle = N}} E(\theta, p)$.
2: Solve for $y = \underset{u \in \mathbb{R}^n}{\operatorname{argmin}} \left\| u \left.\frac{\partial F}{\partial p}\right|_{p=p_\theta} + (p_\theta - p_{gs}) \right\|^2$
3: Compute gradient $\frac{d\mathcal{L}(p_\theta)}{d\theta} = \left.\frac{\partial(\langle y, F\rangle)}{\partial \theta}\right|_{\substack{p=p_\theta \\ \theta=\theta}}$
4: return $\frac{d\mathcal{L}(p_\theta)}{d\theta}$
```

The quantity $\left.\frac{\partial(\langle y, F\rangle)}{\partial \theta}\right|_{\substack{p=p_\theta \\ \theta=\theta}}$ can be obtained using automatic differentation. Similarly $u\left.\frac{\partial F}{\partial p}\right|_{p=p_\theta}$ can be computed with automatic differentation and without storing the full Jacobian matrix in memory. This matters for bigger batch sizes and molecules, since the memory cost scales quadratically. Matrix-free methods were used to solve for y vector.

3.1 Jacobian approach experiments and results

Convergence thresholds

I tested the algorithm with various tolerance threshold and training learning rate on both fixed point search and linear equation solver. I found out that strict thresholds on fixed point search hindered training completely. The loss increased during training and varying both learning rate and the tolerance on linear equation solving did not change this fact. Even when the loss decreased at first it could increase later as shown in Figure 1.

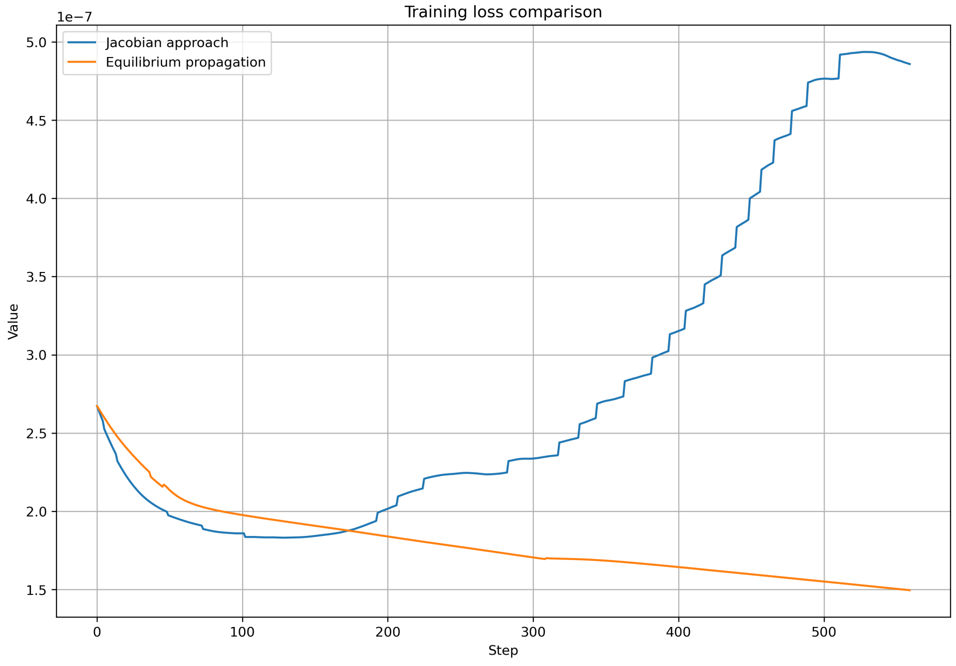

Figure 1: Comparison of training loss for Jacobian and equilibrium propagation approaches. Jacobian approach decreases loss at first, but starts diverging later. Equilibrium propagation seems to be more stable.

Conjugate gradient

I implemented and tested conjugate gradient method for solving linear system. However, the algorithm rapidly diverged and proved to be unsuitable.

Summary of Jacobian appraoch

I have not been able to make Jacobian approach perform for the purpose of fine-tuning the model. Despite loss function decreasing during course of the training, when started from a different starting point the loss did not decrease. This suggests, that Jacobian approach did not in fact move the fixed point p_θ closer to real ground state p_{gs} or did it while making energy landscape much more difficult to navigate. I include more details in section 1.5.

4 Equilibrium propagation

Base version

During my internship I also implemented and tested optimization using Equilibrium Propagation (or EQ,EqProp, see [7], [10]), which is an alternative method of computing the gradient with respect to model parameters. Let's denote

$$T(\theta, p, \beta) := \beta\mathcal{L}(p) + E(\theta, p), \tag{32}$$

where $\beta \in \mathbb{R}$. This is called **total energy** in [7]. Let's also denote

$$p_\theta^\beta := \arg\min_p T(\theta, p, \beta) \tag{33}$$

Remark 3. *For $\beta > 0$ the function $p \mapsto \beta\mathcal{L}(p)$ acts as a regularizer.*

Remark 4. *Since $T(\theta, p, 0) = E(\theta, p)$ we have $p_\theta^0 = p_\theta$.*

Statement 3 (Equilibrium propagation formula)**.**

$$\frac{d}{d\theta}\mathcal{L}(p_\theta) = \lim_{\beta\to 0}\frac{\frac{\partial T}{\partial\theta}(\theta, p_\theta^\beta, \beta) - \frac{\partial T}{\partial\theta}(\theta, p_\theta^0, 0)}{\beta} = \lim_{\beta\to 0}\frac{1}{\beta}\frac{\partial}{\partial\theta}\Big(T(\theta, p_\theta^\beta, \beta) - T(\theta, p_\theta^0, 0)\Big). \tag{34}$$

Remark 5. *This formula can be used to numerically approximate the gradient of the loss function. Since in our case $\mathcal{L}(p)$ doesn't depend on θ, $\frac{\partial T}{\partial\theta}$ simplifies to $\frac{\partial E}{\partial\theta}$, although the formula remains true, when $\mathcal{L}$ is also a function of θ. For example there might be regularization term added to $\mathcal{L}(p)$.*

Proof. Let's denote

$$G(\theta, \beta) := T(\theta, p_\theta^\beta, \beta). \tag{35}$$

Because p_θ^β is C^2, G is C^2 and so we have symmetry of second derivatives

$$\frac{d}{d\theta}\frac{d}{d\beta}G(\theta, \beta)\big|_{\beta=0,\theta=\theta} = \frac{d}{d\beta}\frac{d}{d\theta}G(\theta, \beta)\big|_{\beta=0,\theta=\theta} \tag{36}$$

We have

$$\frac{dG}{d\beta} = \frac{\partial T}{\partial\beta} + \frac{\partial T}{\partial p}(\theta, p_\theta^\beta, \beta)\frac{\partial p_\theta^\beta}{\partial\beta}. \tag{37}$$

Since T attains minimum at p_θ^β we have

$$\frac{\partial T}{\partial p}(\theta, p_\theta^\beta, \beta) = 0. \tag{38}$$

This leaves us with

$$\frac{dG}{d\beta}\big|_{\beta=0} = \frac{\partial T}{\partial\beta}(\theta, p_\theta, 0) = \mathcal{L}(p_\theta). \tag{39}$$

Analogically

$$\frac{dG}{d\theta} = \frac{\partial T}{\partial\theta}(\theta, p_\theta^\beta, \beta). \tag{40}$$

In the end we obtain

$$\frac{d}{d\theta}\mathcal{L}(p_\theta) = \frac{d}{d\theta}\frac{d}{d\beta}G(\theta,\beta)\big|_{\beta=0} = \frac{d}{d\beta}\frac{d}{d\theta}G(\theta,\beta)\big|_{\beta=0} = \tag{41}$$

$$= \lim_{\beta\to 0}\frac{1}{\beta}\frac{\partial}{\partial\theta}\Big(T(\theta,p_\theta^\beta,\beta) - T(\theta,p_\theta^0,0)\Big) \tag{42}$$

□

Remark 6. *It turns out the formula remains the same if we change the fixed point definition to*

$$p_\theta^\beta = \underset{\substack{p\in\mathbb{R}^n \\ p:\langle p,w\rangle = N}}{\arg\min}\ T(\theta,p,\beta) \tag{43}$$

In this case equilibrium condition 37 changes to

$$P\bigg(\frac{\partial T}{\partial p}(\theta,p_\theta^\beta,\beta)\bigg) = 0,$$

where, as before, P is a projection operator. Since $\langle w,p_\theta^\beta\rangle$ is does not depend on β nor θ both 39 and 40 also stay the same and so the final formula remains unchanged.

Algorithm 2 Equilibrium propagation algorithm

```
Require: Data $p_{gs}$, model parameters $\theta$, $\beta > 0$.
1: Solve for $p_\theta^0 = \arg\min_{p\in\mathbb{R}^n,\ p:\langle p,w\rangle=N} T(\theta,p,0)$.
2: Solve for $p_\theta^\beta = \arg\min_{p\in\mathbb{R}^n,\ p:\langle p,w\rangle=N} T(\theta,p,\beta)$.
3: Compute gradient $\frac{d\mathcal{L}(p_\theta)}{d\theta} = \frac{1}{\beta}\frac{\partial}{\partial\theta}\Big(T(\theta,p_\theta^\beta,\beta) - T(\theta,p_\theta^0,0)\Big)$
4: return $\frac{d\mathcal{L}(p_\theta)}{d\theta}$
```

4.1 Equilibrium propagation experiments

In contrast to Jacobian approach I managed to fine-tune the model on single molecule. The training loss decreased. What is more telling is that loss also decreased, when fixed point search was run from a different starting point, which wasn't the case for Jacobian approach.

Varying β parameter

I tested equilibrium propagation for different values of parameter β. As shown in 2, choosing too small of a β can negatively affect the performance.

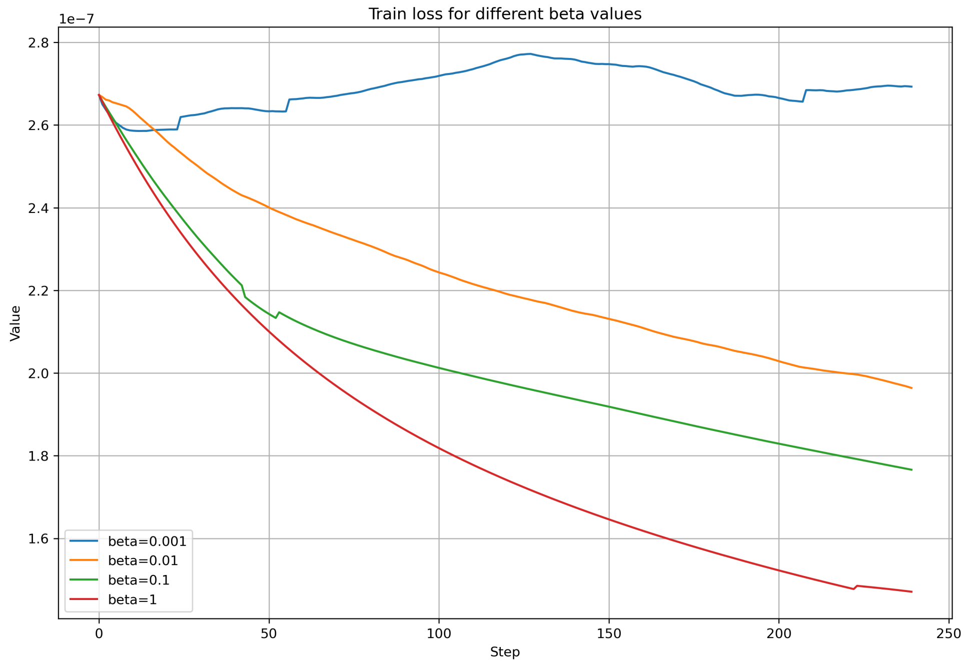

Figure 2: Training loss for different valus of β parameter. The curves correspond to $\beta = 0.001, 0.01, 0.1, 1$ starting from the top.

Training on more than one molecule

After resolving stability issue I tested equilibrium propagation approach on a sample containing 180 molecules. I split the dataset into 80:10:10 ratios for training, validation, and testing; however, the test dataset ultimately went unused. Unfortunately, despite decrease of training loss, there was no improvement in validation loss and therefore no generalization.

Remark 7. *The problem 1 falls under* ***bilevel optimization****. However, due to memory and computational constraints, the gradient is computed in mini-batches. It's possible that an adaptation is required to make it work, which would turn it into a problem of stochastic bilevel optimization. Due to time constraints I did not manage to resolve the problem.*

Summary of equilibrium propagation approach

Equilibrium propagation proved to be more successful than Jacobian approach. I managed to decrease the implicit loss on one molecule, but more work needs to be done to make training on full dataset possible.

5 Other results

5.1 Stability issues

Implicit loss training falls under the framework of deep equilibrium model (DEQ). One of the issues plaguing DEQ models is training stability. As training goes on, fixed point search takes longer and longer. This is reported in [1], [2], [3], and [5]. This was also the case for me when training multiple molecules; however, I did not encounter it when training on a single molecule.

To alleviate the issue, I adapted a technique introduced in [1] called **fixed point correction** (FPC). I am not aware of any explanation why the technique works. It stabilizes the training, although it can hurt performance; therefore, more work is required to make it suitable for fine-tuning.

Implementation and details of the fixed point correction technique are described in section 1.3.

It would be beneficial to make the technique compatible with warm-starting, since that would allow for dramatic improvements in stability and training speed.

Figure 3 shows a plot of the number of unconverged runs plotted against the total number of runs. The fixed point search parameters were chosen such that every molecule converged before fine-tuning. Results show, that fixed point correction is an effective measure to stabilize training.

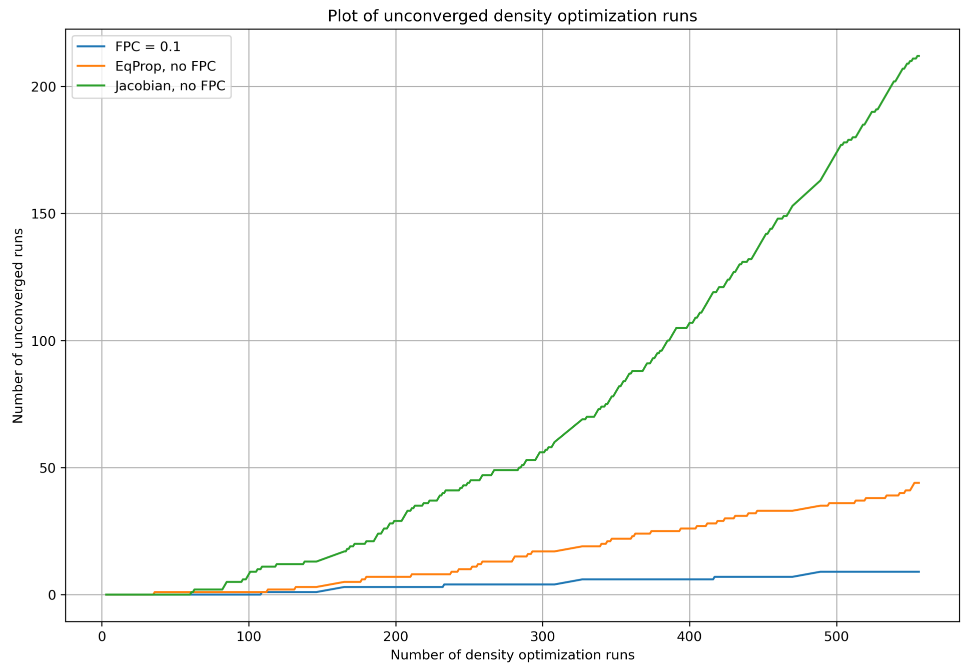

Figure 3: Number of unconverged density optimization runs for EqProp approach with Jacobian approach with no fixed point correction (top curve),no fixed point correction (middle curve) and fixed point correction $p = 0.1$ (bottom curve).

6 Conclusions

During my internship at Hamprecht Lab, I implemented and tested two approaches to implicit loss calculations. I have found that Jacobian approach was not successful for fine-tuning and did not result in improvement of key metrics.

In contrast, the equilibrium propagation approach successfully decreased targeted metrics. However, I encountered significant model training stability issues. After conducting an extensive literature search, I found the technique for alleviating the issues and adapted it to my setting.

Therefore, equilibrium propagation was shown to be an alternative method for optimizing losses defined implicitly, even when the Jacobian approach does not work.

On a personal level, this experience taught me the importance of working with version control within a group and conducting thorough literature searches.

Future work would involve investigating Jacobian approach further. Since both methods should yield the exact same gradient the issue lies either in faulty implementation or some unforeseen complications. Since fine-tuning on multiple molecules currently faces significant challenges, it also requires additional work, both regarding training stability and model generalization. Lastly, another issue to mention is training speed. Currently, this approach is much too slow, to train on the whole dataset.

Bibliography

[1] Shaojie Bai, Zhengyang Geng, Yash Savani, and J Zico Kolter. Deep equilibrium optical flow estimation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 620–630, 2022.

[2] Shaojie Bai, Vladlen Koltun, and J Zico Kolter. Stabilizing equilibrium models by jacobian regularization. *arXiv preprint arXiv:2106.14342*, 2021.

[3] Andreas Burger, Luca Thiede, Alan Aspuru-Guzik, and Nandita Vijaykumar. DEQuify your force field: More efficient simulations using deep equilibrium models. In *AI for Accelerated Materials Design - ICLR 2025*, 2025.

[4] Asen L Dontchev and R Tyrrell Rockafellar. *Implicit functions and solution mappings*, volume 543. Springer, 2009.

[5] Zhengyang Geng and J Zico Kolter. Torchdeq: A library for deep equilibrium models. *arXiv preprint arXiv:2310.18605*, 2023.

[6] Roman Remme, Tobias Kaczun, Tim Ebert, Christof A Gehrig, Dominik Geng, Gerrit Gerhartz, Marc K Ickler, Manuel V Klockow, Peter Lippmann, Johannes S Schmidt, et al. Stable and accurate orbital-free dft powered by machine learning. *arXiv preprint arXiv:2503.00443*, 2025.

[7] Benjamin Scellier and Yoshua Bengio. Equilibrium propagation: Bridging the gap between energy-based models and backpropagation. *Frontiers in computational neuroscience*, 11:24, 2017.

[8] Feitong Song and Ji Feng. Neuralscf: Neural network self-consistent fields for density functional theory. *arXiv preprint arXiv:2406.15873*, 2024.

[9] He Zhang, Siyuan Liu, Jiacheng You, Chang Liu, Shuxin Zheng, Ziheng Lu, Tong Wang, Nanning Zheng, and Bin Shao. Overcoming the barrier of orbital-free density functional theory for molecular systems using deep learning. *Nature Computational Science*, 4(3):210–223, 2024.

[10] Nicolas Zucchet and Joao Sacramento. Beyond backpropagation: bilevel optimization through implicit differentiation and equilibrium propagation. *Neural Computation*, 34(12):2309–2346, 2022.

1 Implementation details

Model architecture

I used a standard Graphormer model, which was trained on combination of QMUGS and QM9 datasets for 90 epochs. The model architecture was based on article [9].

1.1 Fixed point search

To obtain the fixed I used projected gradient descent with momentum with learning rate $\alpha = 5 \cdot 10^{-3}$ and momentum $\beta = 0.9$. I ran the algorithm until gradient norm fell below tolerance threshold, which in most cases was set as $\epsilon = 10^{-4}$. The starting point was set to p_{gs} and the maximum number of gradient descent steps was set to 100.

1.2 Fine-tuning on single molecule

I trained the model with ADAM algorithm with learning rate equal to 10^{-7} and weight decay set to 0 for 4 epochs. Each epoch consisted of 80 weight updates.

1.3 Fixed point correction

Fixed point correction is a technique for stabilizing DEQ training. The technique is described in [1], [5] and [3]. It is as follows.

Given fixed point trajectory $p_0 = p_{gs}, p_1, \ldots, p_T \approx p_\theta$ we uniformly select N intermediate points $p_{i_1}, p_{i_2}, \ldots, p_{i_N} = p_T$ and compute the loss

$$\mathcal{L}_{FPC}(\theta) = \sum_{k=1}^{n} \gamma^{n-k} \mathcal{L}(p_{i_k}), \tag{44}$$

where gradient for each individual loss $\mathcal{L}(p_{i_k})$ is computed by treating as if p_{i_k} was a fixed point. Typical value of γ parameter is 0.8 and it was also the one I used.

I modified this technique in a stochastic way to account for varying number of fixed point steps for different molecules.

Let's fix $p \in (0, 1)$. Then, for each point in fixed point trajectory, we decide select the intermediate point with probability p. Finally, we compute the gradient based on the loss function

$$\mathcal{L}_{FPC}(\theta) = \sum_{k=1}^{n} \gamma^{n-k+1} \mathcal{L}(p_{i_k}). \tag{45}$$

1.4 Solving linear equation

To solve linear equation I used gradient descent algorithm, since obtaining the matrix involved in the problem is computationally expensive and scales quadratically with n. The derivative of function

$$u \mapsto \left\| u \frac{\partial F}{\partial p}\bigg|_{p=p_\theta} + (p_\theta - p_{gs}) \right\|^2$$

is equal to

$$2\left(u\frac{\partial F}{\partial p}\bigg|_{p=p_\theta} + (p_\theta - p_{gs})\right) \cdot \left(\frac{\partial F}{\partial p}\right)^T.$$

Therefore it's possible to calculate the derivative without materializing the full matrix using vector-jacobian and jacobian-vector product in PyTorch.

I also encountered divergence problems while using plain gradient descent and had to switch to ADAM optimizer to achieve convergence. This might indicate, that jacobian possesses big eigenvalues, which is further corroborated direct eigenvalues computation.

1.5 Jacobian matrix

The jacobian matrix $\frac{\partial F}{\partial p}$ for the model both before and after training has one zero eigenvalue, both before and after training. All other eigenvalues are positive. Table below shows s the minimal and maximal eigenvalues of jacobian matrix.

Model	**Smallest zero eigenvalue**	**Biggest eigenvalue**	**Condition Number**
Pre-trained	$4.55 \cdot 10^{-2}$	$8.94 \cdot 10^{2}$	≈ 20000
Fine-tuned	$6.67 \cdot 16^{-2}$	$11.49 \cdot 10^{2}$	≈ 17000

Table 1: Comparison of eigenvalues across models. The condition number is the ratio of the maximum eigenvalue to the minimum non-zero eigenvalue.

The big condition number might explain failure of the jacobian approach and relative success of equilibrium propagation.