Fast Training of Implicit Networks with Applications in Inverse Problems

Linghai Liu ¹, Allen Tong ², Lisa Zhao ³
Mentor: Samy Wu Fung ⁴

Emory University REU/RET Computational Mathematics for Data Science

June 30, 2022

¹Brown University
²University of California, Los Angeles
³University of California, Berkeley
⁴Department of Applied Mathematics and Statistics, Colorado School of Mines
Acknowledgements

• We sincerely thank the guidance of our mentor, Dr. Samy Wu Fung, and other mentors at Emory University for the opportunity.

• Our work is supported by the US National Science Foundation awards DMS-2051019 and DMS-1751636.
What are Inverse Problems?

Inverse problems consist of recovering a signal $x^*$ (e.g. an image, a parameter of a PDE, etc.) from indirect, noisy measurements $d$. This measurement process is usually modeled as an operator $\mathcal{A}$, satisfying the following:

$$d = \mathcal{A}x^* + \varepsilon,$$
What are Inverse Problems?

Inverse problems consist of recovering a signal \( x^* \) (e.g. an image, a parameter of a PDE, etc.) from indirect, noisy measurements \( d \). This measurement process is usually modeled as an operator \( A \), satisfying the following:

\[ d = Ax^* + \varepsilon, \]

Our task deals with image deblurring, i.e.,

- \( d \in \mathbb{R}^{n \times n} \): blurred image with noise
- \( x^* \in \mathbb{R}^{n \times n} \): original image
- \( \varepsilon \in \mathbb{R}^{n \times n} \): random noise (unknown) in \( \mathbb{R}^{n \times n} \)
From a Classical Approach

Direct Inverse:

\[ d = A x^* + \varepsilon \iff x^* = A^{-1} d - A^{-1} \varepsilon \]
From a Classical Approach

Direct Inverse:

\[ d = Ax^* + \varepsilon \implies x^* = A^{-1}d - A^{-1}\varepsilon \]
Classical Approach Cont.

**Optimization**: Formulate an optimization problem as follows:

\[
x^* = \arg \min_{x \in \mathbb{R}^{n \times n}} \frac{1}{2} \| Ax - d \|_2^2 + \lambda R(x)
\]

where \( R(x) \) is chosen based on prior knowledge of your data, \( \lambda > 0 \) is a tunable parameter.
Classical Approach Cont.

**Optimization:** Formulate an optimization problem as follows:

\[ x^* = \arg \min_{x \in \mathbb{R}^{n \times n}} \frac{1}{2} \| Ax - d \|_{L^2}^2 + \lambda R(x) \]

where \( R(x) \) is chosen based on prior knowledge of your data, \( \lambda > 0 \) is a tunable parameter.

E.g. \( R(x) = 0 \) \( \implies x^* = A^{-1}d \) when \( A \) invertible
Classical Approach Cont.

**Optimization**: Formulate an optimization problem as follows:

\[
x^* = \arg \min_{x \in \mathbb{R}^{n \times n}} \frac{1}{2} ||Ax - d||^2_{L^2} + \lambda R(x)
\]

where \( R(x) \) is chosen based on prior knowledge of your data, \( \lambda > 0 \) is a tunable parameter.

E.g. \( R(x) = 0 \implies x^* = A^{-1}d \) when \( A \) invertible

E.g. Use gradient descent where \( R(x) = \frac{\lambda}{2} ||x||^2_{L^2} \):

Original Image  Blurred Noisy Image  Apply Gradient Descent
Implicit Deep Learning

• Use dataset \( \{(d_i, x_i^*)\}_{i=1}^m \) and physics (namely \( \mathcal{A} \))

---

Implicit Deep Learning

- Use dataset $\{(d_i, x_i^*)\}_{i=1}^m$ and physics (namely $A$)
- Mimic gradient descent $^5$, but replace $\lambda \nabla_x R$ with a trainable network $S_\Theta$: $\forall i$ and $0 \leq k \leq K - 1,$

$$x_i^{k+1} = x_i^k - \eta \left( \nabla_x ||Ax_i^k - d_i||^2_{L^2} + S_\Theta(x_i^k) \right)$$

$:= T_\Theta(x_i^k)$

where:
- $\eta > 0$ is step size
- $T_\Theta(\cdot)$ is a layer of our neural network $N_\Theta(\cdot)$
- $K$ is the number of layers

---

Implicit Deep Learning

- Use dataset \( \{(d_i, x_i^*)\}_{i=1}^m \) and physics (namely \( A \))
- Mimic gradient descent \(^5\), but replace \( \lambda \nabla_x R \) with a trainable network \( S_\Theta \): \( \forall i \) and \( 0 \leq k \leq K - 1 \),

\[
x_{i}^{k+1} = x_{i}^{k} - \eta \left( \nabla_x ||Ax_{i}^{k} - d_i||^2_{L2} + S_\Theta(x_{i}^{k}) \right)
\]

:= \( T_\Theta(x_{i}^{k}) \)

where:

- \( \eta > 0 \) is step size
- \( T_\Theta(\cdot) \) is a layer of our neural network \( N_\Theta(\cdot) \)
- \( K \) is the number of layers

- Problems: memory, choice of \( K \)

---

Implicit Deep Learning

- Implicit Deep Learning: Send $K \to \infty$ until a fixed point of $T_\Theta(\cdot)$ is found, i.e. $x_i^* = T_\Theta(x_i^*)$
Implicit Deep Learning

- Implicit Deep Learning: Send $K \to \infty$ until a fixed point of $T_{\Theta}(\cdot)$ is found, i.e. $x_i^* = T_{\Theta}(x_i^*)$
- Output: given the image $d_i$, $N_{\Theta}(d_i) := x_i^*$
Implicit Deep Learning

• Implicit Deep Learning: Send $K \to \infty$ until a fixed point of $T_{\Theta}(\cdot)$ is found, i.e. $x_i^* = T_{\Theta}(x_i^*)$

• Output: given the image $d_i$, $\mathcal{N}_\Theta(d_i) := x_i^*$

• Question: why convergence?
Implicit Deep Learning

- Implicit Deep Learning: Send $K \to \infty$ until a fixed point of $T_\Theta(\cdot)$ is found, i.e. $x_i^* = T_\Theta(x_i^*)$
- Output: given the image $d_i$, $N_\Theta(d_i) := x_i^*$
- Question: why convergence?
- Convergent if $T_\Theta(\cdot)$ is a contraction mapping with Lipschitz constant $\gamma \in [0, 1)$, i.e,
  \[
  \forall y_1, y_2 \in \mathbb{R}^{n^2}, \| T_\Theta(y_1) - T_\Theta(y_2) \|_{L^2} \leq \gamma \| y_1 - y_2 \|_{L^2}
  \]
Implicit Deep Learning

• Implicit Deep Learning: Send $K \to \infty$ until a fixed point of $T_\Theta(\cdot)$ is found, i.e. $x_i^* = T_\Theta(x_i^*)$

• Output: given the image $d_i$, $\mathcal{N}_\Theta(d_i) := x_i^*$

• Question: why convergence?

• Convergent if $T_\Theta(\cdot)$ is a contraction mapping with Lipschitz constant $\gamma \in [0, 1)$, i.e.,

$\forall y_1, y_2 \in \mathbb{R}^{n^2}, \| T_\Theta(y_1) - T_\Theta(y_2) \|_{L^2} \leq \gamma \| y_1 - y_2 \|_{L^2}$

Then, by Banach fixed-point theorem, there exists $y^* \in \mathbb{R}^{n^2}$ s.t. $T_\Theta(y^*) = y^*$
Implicit Backpropagation

Suppose we find a fixed point $x^*$ for the previous update, i.e.,

$$x^* = T_\Theta(x^*)$$

With implicit differentiation,

$$\frac{dx^*}{d\Theta} = T'\Theta(x^*) \frac{dx^*}{d\Theta} + \frac{\partial T_\Theta}{\partial \Theta}(x^*)$$

So the update rule of trainable parameters becomes:

$$\Theta \leftarrow \Theta - \alpha \frac{d\ell}{dx^*} I - T'\Theta(x^*) \frac{dx^*}{d\Theta} - \frac{1}{\alpha} \frac{\partial T_\Theta}{\partial \Theta}(x^*) \frac{dx^*}{d\Theta},$$

where $\alpha > 0$ is the learning rate.

Potential problem: solving (1) is highly nontrivial.
Implicit Backpropagation

Suppose we find a fixed point $x^*$ for the previous update, i.e.,

$$x^* = T_\Theta(x^*)$$

With implicit differentiation,

$$\frac{dx^*}{d\Theta} = \frac{dT_\Theta(x^*)}{dx^*} \frac{dx^*}{d\Theta} + \frac{\partial T_\Theta(x^*)}{\partial \Theta}$$

$$\Rightarrow \left( I - \frac{dT_\Theta(x^*)}{dx^*} \right) \frac{dx^*}{d\Theta} = \frac{\partial T_\Theta(x^*)}{\partial \Theta} \quad (1)$$

So the update rule of trainable parameters becomes:
Implicit Backpropagation

Suppose we find a fixed point $x^*$ for the previous update, i.e.,

$$x^* = T_\Theta(x^*)$$

With implicit differentiation,

$$\frac{dx^*}{d\Theta} = \frac{dT_\Theta(x^*)}{dx^*} \frac{dx^*}{d\Theta} + \frac{\partial T_\Theta(x^*)}{\partial \Theta}$$

$$\Rightarrow \left(1 - \frac{dT_\Theta(x^*)}{dx^*}\right) \frac{dx^*}{d\Theta} = \frac{\partial T_\Theta(x^*)}{\partial \Theta}$$  \hspace{1cm} (1)

So the update rule of trainable parameters becomes:

$$\Theta \leftarrow \Theta - \alpha \frac{d\ell}{dx^*} \left(1 - \frac{dT_\Theta(x^*)}{dx^*}\right)^{-1} \frac{\partial T_\Theta(x^*)}{\partial \Theta},$$

where $\alpha > 0$ is the learning rate.
Implicit Backpropagation

Suppose we find a fixed point $x^*$ for the previous update, i.e.,

$$x^* = T_\Theta(x^*)$$

With implicit differentiation,

$$\frac{dx^*}{d\Theta} = \frac{dT_\Theta(x^*)}{dx^*} \frac{dx^*}{d\Theta} + \frac{\partial T_\Theta(x^*)}{\partial \Theta}$$

$$\Rightarrow \left( I - \frac{dT_\Theta(x^*)}{dx^*} \right) \frac{dx^*}{d\Theta} = \frac{\partial T_\Theta(x^*)}{\partial \Theta}$$

(1)

So the update rule of trainable parameters becomes:

$$\Theta \leftarrow \Theta - \alpha \frac{d\ell}{dx^*} \left( I - \frac{dT_\Theta(x^*)}{dx^*} \right)^{-1} \frac{\partial T_\Theta(x^*)}{\partial \Theta},$$

where $\alpha > 0$ is the learning rate.

Potential problem: solving (1) is highly nontrivial
Jacobian-Free Backpropagation (JFB)

- Goal: alleviate memory requirement and avoid high computational cost.
Jacobian-Free Backpropagation (JFB)

- **Goal**: alleviate memory requirement and avoid high computational cost.
- **Key idea**: replace the Jacobian \( \left( I - \frac{dT_\Theta(x^*)}{dx^*} \right) \) with \( I \).
Jacobian-Free Backpropagation (JFB)

- **Goal:** alleviate memory requirement and avoid high computational cost.
- **Key idea:** replace the Jacobian \( \left( I - \frac{dT_{\Theta}(x^*)}{dx^*} \right) \) with \( I \)
- **Implicit Networks** calculate the true gradient:
  \[
  \nabla_\Theta \ell = \frac{d\ell}{dx^*} \left( I - \frac{dT_{\Theta}(x^*)}{dx^*} \right)^{-1} \frac{\partial T_{\Theta}(x^*)}{\partial \Theta}
  \]

---

Jfb: Jacobian-free back-propagation for implicit networks.
Jacobian-Free Backpropagation (JFB)

- **Goal:** alleviate memory requirement and avoid high computational cost.
- **Key idea:** replace the Jacobian \( I - \frac{dT_\Theta(x^*)}{dx^*} \) with \( I \)
- **Implicit Networks** calculate the true gradient:
  \[
  \nabla \Theta \ell = \frac{d\ell}{dx^*} \left( I - \frac{dT_\Theta(x^*)}{dx^*} \right)^{-1} \frac{\partial T_\Theta(x^*)}{\partial \Theta}
  \]
- **JFB** approximates the gradient: \( p_\Theta = \frac{d\ell}{dx^*} \frac{\partial T_\Theta(x^*)}{\partial \Theta} \) which is a **descent direction** for \( \ell \) if the following conditions hold (next slide) \(^6\):

\(^6\)S. W. Fung, H. Heaton, Q. Li, D. McKenzie, S. Osher, and W. Yin (2021)
**Jfb:** Jacobian-free back-propagation for implicit networks.
JFB Conditions

If:

i. $T_\Theta$ is contraction mapping with Lipschitz constant $\gamma$

ii. $T_\Theta$ is continuously differentiable w.r.t. $\Theta$

iii. $M := \frac{\partial T_\Theta}{\partial \Theta}$ has full column rank

iv. $M$ is well-conditioned, i.e., $\kappa(M^T M) < \frac{1}{\gamma}$

Then

$$p_\Theta = \frac{d\ell}{dx^*} \frac{\partial T_\Theta}{\partial \Theta}$$

is a descent direction for loss function $\ell$. 
Numerical Experiments

- **Dataset**: CelebA\(^7\) (annotated celebrity faces)

Numerical Experiments

- Generate blurred noisy images:
  
<table>
<thead>
<tr>
<th>Original image</th>
<th>Blurred noisy image</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="Original Image" /></td>
<td><img src="image2" alt="Blurred Noisy Image" /></td>
</tr>
</tbody>
</table>

  PSNR = 21.57, SSIM=0.80

- Train with JFB

  
<table>
<thead>
<tr>
<th>Loss vs number of SGD iterations</th>
<th>Reconstructed image</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image3" alt="Loss vs Iterations" /></td>
<td><img src="image4" alt="Reconstructed Image" /></td>
</tr>
</tbody>
</table>

  PSNR = 25.69, SSIM=0.86
Numerical Experiments

- Generate blurred noisy images:
  - original image
  - blurred noisy image
  - PSNR = 21.57, SSIM=0.80

- Train with JFB

- Preliminary results:
  - Loss v.s. number of SGD iterations
  - Reconstructed image
  - PSNR = 25.69, SSIM=0.86
Numerical Experiments

• Generate blurred noisy images:
  original image  blurred noisy image
  ![Original Image](image1.jpg)  ![Blurred Noisy Image](image2.jpg)  
  PSNR = 21.57, SSIM=0.80

• Train with JFB

• Preliminary results:
  Loss v.s. number of SGD iterations
  ![Loss vs. SGD Iterations](image3.png)
  Reconstructed image
  ![Reconstructed Image](image4.jpg)  
  PSNR = 25.69, SSIM=0.86
Future Work

- Train models using different learned optimization algorithms, e.g. \textit{Proximal Gradient Descent} and \textit{Alternating Directions Method of Multipliers (ADMM)}
- Experiment with fastMRI data \(^8\)
- Compare training speeds and accuracy with Jacobian-based algorithms

---

Future Work

• Train models using different learned optimization algorithms, e.g. *Proximal Gradient Descent* and *Alternating Directions Method of Multipliers (ADMM)*
• Experiment with fastMRI data \(^8\)
• Compare training speeds and accuracy with Jacobian-based algorithms

Thank you!

---

Distributed optimization and statistical learning via the alternating
direction method of multipliers

Convex Optimization
*Cambridge University Press*

Deep learning face attributes in the wild
*Proceedings of the IEEE international conference on computer vision*
3730–3738, 2015

C. Vogel (2002)
Computational Methods for Inverse Problems
*Frontiers in Applied Mathematics. Society for Industrial and Applied Mathematics*

IEEE Transactions on Computational Imaging 7:1123–1133

IEEE Journal on Selected Areas in Information Theory 1(1):39–56

Advances in Neural Information Processing Systems 32

Banach Fixed-Point Theorem

We demonstrate it in $\mathbb{R}^d$:

Suppose $T : \mathbb{R}^d \rightarrow \mathbb{R}^d$ is a contraction map with Lipschitz constant $\gamma \in [0, 1)$.

$\forall x_0 \in \mathbb{R}^d$, iterate as follows:

$$x_1 = T(x_0)$$
$$x_2 = T(x_1)$$
$$\vdots$$
$$x_{i+1} = T(x_i)$$
$$\vdots$$

Then we obtain a sequence $\{x_m\}_{m \in \mathbb{N}}$
Observe that

\[ \| x_2 - x_1 \| = \| T(x_1) - T(x_0) \| \leq \gamma \| x_0 - x_1 \| \]
\[ \| x_3 - x_2 \| = \| T(x_2) - T(x_1) \| \leq \gamma \| x_2 - x_1 \| \leq \gamma^2 \| x_0 - x_1 \| \]
\[ \vdots \]
\[ \| x_{i+1} - x_i \| \leq \gamma^i \| x_0 - x_1 \| \]
\[ \vdots \]

So \( \lim_{m \to \infty} \| x_{m+1} - x_m \| \leq \lim_{m \to \infty} \gamma^m \| x_0 - x_1 \| = 0 \)

We also know that \( 0 \leq \lim_{m \to \infty} \| x_{m+1} - x_m \| \).

\[ \implies \lim_{m \to \infty} \| x_{m+1} - x_m \| = 0 \text{ by Squeeze Theorem} \]
Banach Fixed-Point Theorem

By Triangular Inequality,

\[ ||x_{m+k} - x_m|| \leq ||x_m - x_{m+1}|| + ||x_{m+1} - x_m + k|| \]
\[ \leq ||x_m - x_{m+1}|| + (||x_{m+1} - x_{m+2}|| + ||x_{m+2} - x_{m+k}||) \]
\[ \vdots \]
\[ \leq ||x_m - x_{m+1}|| + ||x_{m+1} - x_{m+2}|| + \cdots \]
\[ + ||x_{m+k-2} - x_{m+k-1}|| + ||x_{m+k-1} - x_{m+k}|| \]

By Squeeze Theorem again, \( \{x_m\}_{m \in \mathbb{N}} \) is a Cauchy sequence, which is equivalent to \( \lim_{m \to \infty} x_m = x^* \) exists.

\[ \implies x^* = T(x^*) \] is a fixed point. \( \square \)
Proximal Gradient Descent

With a function $h(\cdot)$, we can define a proximal operator

$$\text{prox}_h(x) = \arg \min_u \frac{1}{2} \|u - x\|_2^2 + h(u)$$

Then the updating rule becomes:

$$x^{k+1} = \text{prox}_{h,\eta} \left( x^k - \eta \nabla_x \|Ax^k - d\|_2^2 \right)$$

We can replace this \text{prox}_h with a trainable network $R_\Theta(\cdot)$:

$$x^{k+1} = R_\Theta \left( x^k - \eta \nabla_x \|Ax^k - d\|_2^2 \right)$$