Optimization and Optimal Control in Machine Learning

Ali Hamza Abidi, Syed & Andreas Mang
Department of Mathematics, University of Houston, Houston, TX, USA

UNIVERSITY of HOUSTON

Teaser: Our goal was the design and analysis of effective numerical schemes for training deep neuronal networks based on optimal control formulations.

In the present work we explore numerical methods inspired by optimal control theory to train image classifiers [1]. In a first step, we consider a prototypical formulation to develop a generic framework for solving non-linear optimization problems [2,3,4]. In a second step, we study an optimal control formulation for deep learning [1]. Here, we arrive at a large scale, non-linear optimization problem with ordinary differential equations (ODEs) as constraints. We revisit different methods to solve the associated system of ODEs considered in [1]. For our future work, we plan to derive the associated optimality conditions and devise efficient algorithms for their solution.

Non-Linear Least Squares

To develop a simple framework classification of imaging data we considered a non-linear least squares problem.

Problem Formulation and Optimality Conditions

Generally speaking, regularized **non-linear least squares problems** are of the form

$$\underset{\boldsymbol{x}\in\mathbb{R}^n}{\text{minimize}}\ f(\boldsymbol{x}), \quad \text{where } f(\boldsymbol{x}) := \frac{1}{2}\|\sigma(\boldsymbol{A}\boldsymbol{x}) - \boldsymbol{y}\|_2^2 + \frac{\alpha}{2}\|\boldsymbol{L}\boldsymbol{x}\|_2^2, \quad (1)$$

where $f : \mathbb{R}^n \to \mathbb{R}$ is the objective function, $\sigma : \mathbb{R}^m \to \mathbb{R}^m$ is a non-linear ''activation'' function, $\boldsymbol{y} \in \mathbb{R}^m$ is a given dataset, and $\|\boldsymbol{L}\boldsymbol{x}\|_2^2$ is a regularization operator [5], the contribution of which is controlled by the parameter $\alpha > 0$. The **first order optimality conditions** of (1) are given by $\nabla f(\boldsymbol{x}^\star) = 0$, i.e., the gradient $\nabla f(\boldsymbol{x}) \in \mathbb{R}^n$ of f vanishes at optimality [2,3,4]. The **gradient** of (1) is given by $\nabla f(\boldsymbol{x}) = \boldsymbol{A}^\mathsf{T} \operatorname{diag}(\sigma'(\boldsymbol{A}\boldsymbol{x}))\boldsymbol{r} + \alpha\boldsymbol{L}^\mathsf{T}\boldsymbol{L}\boldsymbol{x}$, where $\boldsymbol{r} := \sigma(\boldsymbol{A}\boldsymbol{x}) - \boldsymbol{y}$ denotes the residual and σ' is the first derivative of σ. To solve the non-linear system $\nabla f(\boldsymbol{x}^\star) = 0$ we considered second-order optimization algorithms, which requires second-order derivative information. The **Hessian matrix** $\nabla^2 f(\boldsymbol{x}) \in \mathbb{R}^{n,n}$ associated with (1) is given by $\nabla^2 f(\boldsymbol{x}) = \boldsymbol{A}^T \operatorname{diag}(\boldsymbol{r} \odot \sigma(\boldsymbol{A}\boldsymbol{x}) + \sigma'(\boldsymbol{A}\boldsymbol{x}) \odot \sigma'(\boldsymbol{A}\boldsymbol{x}))\boldsymbol{A} + \alpha\boldsymbol{L}^\mathsf{T}\boldsymbol{L}$, where $\odot : \mathbb{R}^n \to \mathbb{R}^n$ denotes the Hadamard product. Based on this non-linear extension of the classical least-squares problem, we developed methodology that can be used to ''learn'' a weight matrix $\boldsymbol{X}$ to enable the classification of imaging data. We arrive at the (unregularized) **matrix-valued, non-linear least squares problem**

$$\underset{\boldsymbol{X}\in\mathbb{R}^{n,n}}{\text{minimize}}\ f(\boldsymbol{X}), \quad \text{where } f(\boldsymbol{X}) := \frac{1}{2}\|\sigma(\boldsymbol{A}\boldsymbol{X}) - \boldsymbol{Y}\|_F^2, \quad (2)$$

where $f : \mathbb{R}^{n,n} \to \mathbb{R}$, $\|\cdot\|_F^2$ is the squared Frobenius norm, $\sigma : \mathbb{R}^{n,n} \to \mathbb{R}$ is the activation function, $\boldsymbol{A} \in \mathbb{R}^{n,n}$, $\boldsymbol{X} \in \mathbb{R}^{n,n}$ and $\boldsymbol{Y} \in \mathbb{R}^{n,n}$. The **gradient** of (2) is given by $\nabla f(\boldsymbol{X}) = \boldsymbol{A}^\mathsf{T}\sigma'(\boldsymbol{A}\boldsymbol{X} \odot \boldsymbol{R}) \in \mathbb{R}^{n,n}$, where σ' is the first derivative of σ and $\boldsymbol{R} := \sigma(\boldsymbol{A}\boldsymbol{X}) - \boldsymbol{Y}$ denotes the residual. The action of the **Hessian matrix** $\nabla^2 f(\boldsymbol{X})$ on $\tilde{\boldsymbol{X}}$ is given by $[\nabla^2 f(\boldsymbol{X})](\tilde{\boldsymbol{X}}) = \boldsymbol{A}^\mathsf{T}(\sigma'(\boldsymbol{A}\boldsymbol{X}) \odot \sigma'(\boldsymbol{A}\boldsymbol{X}) \odot \boldsymbol{A}\tilde{\boldsymbol{X}} + \boldsymbol{R} \odot \sigma''(\boldsymbol{A}\boldsymbol{X}) \odot \boldsymbol{A}\tilde{\boldsymbol{X}})$, where σ'' denotes the second derivative of σ.

Numerical Optimization

We use an **iterative line search scheme** of the form

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \mu_k \boldsymbol{B}_k \nabla f(\boldsymbol{x}_k), \quad k = 1, 2, \ldots$$

Here, $k \in \mathbb{N}$ is the iteration index and $\mu_k \in (0, 1]$ is determined using a backtracking line search [2]. The search direction is given by $\boldsymbol{s}_k := -\boldsymbol{B}_k \nabla f(\boldsymbol{x}_k)$. We consider Newton's method with $\boldsymbol{B}_k = (\nabla^2 f(\boldsymbol{x}_k))^{-1}$, where $\nabla^2 f(\boldsymbol{x}_k)$ is the Hessian matrix. We invert the Hessian matrix using a matrix-free, conjugate gradient method with a superlinear forcing sequence. As a stopping criterion, we consider the relative reduction of the norm of the gradient $\|\nabla f(\boldsymbol{x}_k)\|_2^2$.

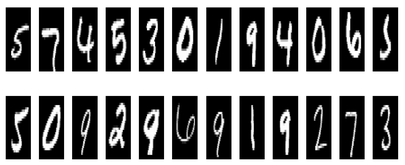

Figure 1: *The MNIST dataset consists of handrawn numbers 0 through 9. It contains 10,000 testing images and 60,000 training images of the size* 28×28. *We show 24 exemplary images from the training data.*

Results

We considered the MNIST dataset shown in Figure 1. We selected a tolerance of 1e−2 for the optimization. Our Newton method converged after 5 Newton iterations (1, 2, 17, 80, and 565 PCG iterations per Newton iteration). We reduced the norm of the gradient from 1.29e5 to 6.34e2. We obtained an accuracy of $\approx 84\%$ for the training and testing dataset.

Optimal Control for Deep Neural Networks

The **optimal control formulation** for training a deep neural network is given by [1]

$$\begin{aligned} \underset{\boldsymbol{\Phi}}{\text{minimize}}\ & \operatorname{dist}(\boldsymbol{C}_{\text{pred}}, \boldsymbol{C}) + \alpha\, \operatorname{reg}(\boldsymbol{W}, \boldsymbol{\mu}, \{\boldsymbol{K}_i\}_{i=1}^n, \{b_i\}_{i=1}^n) \\ \text{subject to}\ & \boldsymbol{Y}_{j+1} = \boldsymbol{Y}_j + h\sigma(\boldsymbol{Y}_j\boldsymbol{K}_j + \boldsymbol{b}_j), \end{aligned} \quad (3)$$

$j = 0, 1, \cdots, n-1$. Here, $\operatorname{dist} : \mathbb{R}^{s,m} \times \mathbb{R}^{s,m} \to \mathbb{R}$ measures the discrepancy between the predicted classification $\boldsymbol{C}_{\text{pred}} \in \mathbb{R}^{s,m}$ and the labels (data) $\boldsymbol{C} \in \{0, 1\}^{s,m}$. The unknowns $\boldsymbol{\Phi}$ of the optimization problem (3) are the weights $\boldsymbol{K}_i \in \mathbb{R}^{n,n}$ and biases b_i of the ''ResNET'' forward propagation and the weights $\boldsymbol{W} \in \mathbb{R}^{n,m}$ and biases $\boldsymbol{\mu} \in \mathbb{R}^m$ that parameterize the classifier. Consequently, $\boldsymbol{\Phi} := \{\boldsymbol{W}, \boldsymbol{\mu}, \{\boldsymbol{K}_i\}_{i=1}^n, \{b_i\}_{i=1}^n\}$. The prediction $\boldsymbol{C}_{\text{pred}}$ is computed according to $\boldsymbol{C}_{\text{pred}} = g(\boldsymbol{Y}_n\boldsymbol{W} + \boldsymbol{e}_s \otimes \mu)$, where $\boldsymbol{Y}_n$ is the final state computed by solving the forward propagation, $\boldsymbol{e}_s := (1, \ldots, 1)^\mathsf{T} \in \mathbb{R}^s$, and $g : \mathbb{R}^{s,m} \to \mathbb{R}^{s,m}$ is the so called hypothesis function.

Forward Propagation

The **forward propagation** in (3) is given by the constraint $\boldsymbol{Y}_{j+1} = \boldsymbol{Y}_j + h\sigma(\boldsymbol{Y}_j\boldsymbol{K}_j + \boldsymbol{b}_j)$ for $j = 1, \ldots, n$, where n denotes the number of layers and $\sigma : \mathbb{R}^{s,n} \to \mathbb{R}^{s,n}$ represents an activation function. One can interpret the forward propagation as an explicit Euler time discretization of the nonlinear ODE $\mathrm{d}_t\boldsymbol{y}(t) = \sigma(\boldsymbol{K}^\mathsf{T}(t)\boldsymbol{y}(t) + b(t))$ with initial condition $\boldsymbol{y}(0) = \boldsymbol{y}_0$ [1]. Similarly, it is possible to frame the forward propagation as a continuous differential equation that is inspired by Hamiltonian dynamics [1]. The associated coupled system of ODEs is given by $\mathrm{d}_t\boldsymbol{y}(t) = \sigma(\boldsymbol{K}(t)\boldsymbol{z}(t) + b(t))$ and $\mathrm{d}_t\boldsymbol{z}(t) = -\sigma(\boldsymbol{K}^\mathsf{T}(t)\boldsymbol{y}(t) + b(t))$, with initial conditions $\boldsymbol{y}(0) = \boldsymbol{y}_0$ and $\boldsymbol{z}(0) = 0$, respectively.

Numerical Time Integration

We considered two approaches for modeling the forward propagation. First, the explicit Euler method used in (3). To stabilize the forward propagation we used antisymmetric weight matrices [1], which lead to $\boldsymbol{Y}_{j+1} = \boldsymbol{Y}_j + h\sigma((1/2)\boldsymbol{Y}_j(\boldsymbol{K}_j + \boldsymbol{K}_j^\mathsf{T} - \gamma\boldsymbol{I}) + b_j)$. To solve the second system given above we considered a symplectic Verlet method [1], which yields the numerical scheme $\boldsymbol{z}_{j+\frac{1}{2}} = \boldsymbol{z}_{j-\frac{1}{2}} - h\sigma(\boldsymbol{K}_j^\mathsf{T}\boldsymbol{y}_j + b_j)$ and $\boldsymbol{y}_{j+1} = \boldsymbol{y}_j + h\sigma(\boldsymbol{K}_j\boldsymbol{z}_{j+\frac{1}{2}} + b_j)$.

Results

We report results for the different numerical time integration schemes for the two variants of the continuous interpretation of the forward propagation in Figures 2 and 3.

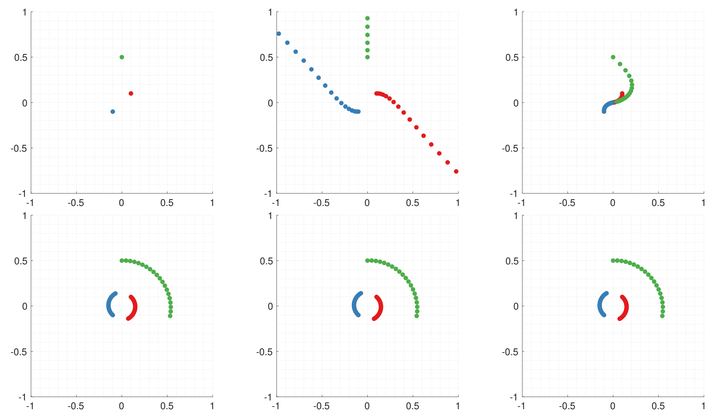

Figure 2: *Phase plane diagrams for* $n = 21$ *identical layers using an Euler time integration for the forward propagation. From top left to bottom right, we show the initial configuration of three features* $\boldsymbol{y}_1$ *(blue),* $\boldsymbol{y}_2$ *(red), and* $\boldsymbol{y}_3$ *(green). Subsequently, we show results for different test matrices matrices* $\boldsymbol{K}$ *at layer* 21 *using an unstable explicit Euler scheme (second, third, and fourth figure) and the antisymmetric discretization (last two figures). We see that the antisymmetric discretization remains stable.*

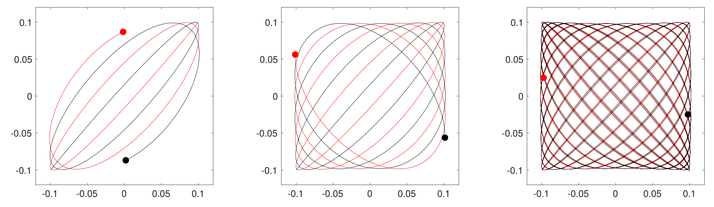

Figure 3: *Phase space diagrams for the forward propagation using the Verlet method. We show the behavior of the network for* $n = 500, 1000, 5000$ *identical layers.*

Conclusions

We have developed and studied computational methods for solving non-linear optimization problems. We have derived optimality conditions and designed a Newton-type method for their solution. In addition, we have considered an optimal control formulation for training deep neuronal networks. We have studied different numerical schemes to solve the forward propagation. For future work, we plan to derive the optimality conditions and design effective numerical methods for their solution.

References

1. E. Haber & L. Ruthotto, Stable Architectures for Deep Neural Networks, Inverse Problems 34 014004, 2017.
2. J. Nocedal & S. Wright, Numerical Optimization, Springer Science, 1999.
3. S. Boyd and L. Vandenberghe, Convex Optimization, Cambridge University Press 2004.
4. A. Beck, Introduction to Nonlinear Optimization: Theory, Algorithms, and Applications with MATLAB, SIAM, 2014.
5. H. W. Engl, M. Hanke, A. Neubauer, Regularization of Inverse Problems, Springer, 2000.