Backpropagation - A Perspective of Matrix Calculus

Yu Chen

November 20, 2023

Abstract

Backpropagation holds critical importance in machine learning, enabling various models, including deep neural networks, to learn features from data via objective optimization. This paper introduces a novel formulation for the Backpropagation algorithm in a closed-matrix form, relying solely on differential laws and matrix trace properties.

1 Introduction

Backpropagation (BP) stands as a pivotal cornerstone in the realm of machine learning and deep learning, particularly in the training of deep neural networks [6]. Operating as an iterative, recursive, and highly efficient method, BP optimizes the weights of neural networks based on error propagated from preceding iterations. Rooted in a gradient descent approach that leverages the chain rule, BP seeks to minimize the disparity between actual and desired outputs. This is achieved through the meticulous adjustment of weights, propagating errors backward through the network from the output to the input layer [10].

Diverse perspectives have arisen regarding the formulation of BP. Shun-ichi Amari characterizes BP as a stochastic gradient descent method [1]. Robert Hecht Nielsen et al. position BP as a foundational element in neural network architectures [6]. Raúl Rojas considers BP an efficient numeric algorithm for handling extensive learning problems [9]. In the realm of neuroscience, Timothy P. Lillicrap et al. explore the relationship between BP and the brain, deriving effective learning in deep networks [8]. Leonid Berlyand et al. provide a comprehensive overview, portraying BP as an effective technique for computing the gradient of an objective function through iterative application of the chain rule across each layer [2]. Saeed Damadi et al. introduce a gradient representation in matrix multiplication terms using the Jacobian operator [3]. Alan Edelman et al. present a linear algebra formulation utilizing Gaussian elimination on triangular systems of equations, represented by a ”backslash,” for calculating gradients [4].

This paper presents a reformulation of the BP algorithm, conceptualizing it as a method for deriving the gradient of a scalar objective function concerning matrix-form parameters or variables in a closed-matrix form. The novelty lies in consistently representing the gradient through a series of matrix calculations. To elucidate our new BP form, we commence with an exploration of the gradient descent (GD) algorithm in the subsequent section. Subsequently, we establish the essential mathematical foundations of matrix calculus required for the BP reformulation in Section 2. Section 3 expounds upon the closed-matrix form of BP using the introduced matrix calculus, illustrating its applicability across a broad spectrum of machine learning models. Finally, we draw conclusions and discuss future avenues based on our theoretical framework in Section 4.

Background We are confronted with an optimization problem 1 that eludes analytical solution.

$$\min_{\vec{x}} f(\vec{x}), \quad \vec{x} \in \mathbb{R}^D \quad \text{and} \quad f(\vec{x}) \in \mathbb{R}. \tag{1}$$

Hence, we address it by employing a numerical optimization method within an iterative framework.

$$\vec{x}_{t+1} := \vec{x}_t + \alpha_t \vec{d}_t \tag{2}$$

Here, $\vec{d}_t$ denotes a unit vector with $\|\vec{d}_t\|_2 = 1$, signifying a direction from the current position $\vec{x}_t$ to the subsequent position $\vec{x}_{t+1}$. The parameter α_t represents the step size the algorithm should undertake when initiated from position $\vec{x}_t$. Alternatively, we formulate the resolution of this optimization challenge as a constrained optimization problem, as presented in (3).

$$\min_{\vec{d}_t,\alpha_t} f(\vec{x}_t + \alpha_t \vec{d}_t) \quad \text{s.t.} \quad \|\vec{d}_t\|_2^2 = 1 \tag{3}$$

Assume function $f(\vec{x})$ is first-order differentiable around $\vec{x}_t$, so $f(\vec{x}_t + \alpha_t \vec{d}_t) = f(\vec{x}_t) + \alpha_t \vec{d}_t^T \nabla_{\vec{x}_t} f(\vec{x}) + o(\alpha_t)$. The GD algorithm approximate function f by formula 4.

$$f(\vec{x}_t + \alpha_t \vec{d}_t) \approx f(\vec{x}_t) + \alpha_t \vec{d}_t^T \nabla_{\vec{x}_t} f(\vec{x}) \tag{4}$$

Then, GD solves two sub-optimization problems 5 and 6 to approximately solve problem 1.

$$\vec{d}_t^* = \underset{\vec{d}_t}{argmin}\, f(\vec{x}_t) + \alpha_t \vec{d}_t^T \nabla_{\vec{x}_t} f(\vec{x}) \quad \text{s.t.} \quad \|\vec{d}_t\|_2^2 = 1 \tag{5}$$

$$\alpha_t^* = \underset{\alpha_t}{argmin}\, f(\vec{x}_t + \alpha_t \vec{d}_t^*) \tag{6}$$

Within the scope of this paper, our attention is concentrated on addressing problem (5). This choice is deliberate, given that problem (6) is amenable to a variety of line search algorithms, as detailed in the provided reference. Problem 5 can be solved by Lagrange multiplier method as following:

$$\vec{d}_t^* = argmin_{\vec{d}_t} L(\vec{d}_t, \lambda_t) := f(\vec{x}_t) + \alpha_t \vec{d}_t^T \nabla_{\vec{x}_t} f(\vec{x}) + \lambda_t (\|\vec{d}_t\|_2^2 - 1) = -\frac{\nabla_{\vec{x}_t} f(\vec{x})}{\|\nabla_{\vec{x}_t} f(\vec{x})\|_2} \tag{7}$$

Evidently, the pivotal requirement in tackling problem (5) lies in the precise computation of the gradient of the objective function, denoted as $\nabla_{\vec{x}_t} f(\vec{x})$.

2 Basics of Matrix Calculus

We start from the definition of gradients.

Definition 2.1. Suppose $f_1(x) : \mathbb{R} \to \mathbb{R}$, $f_2(\vec{x}) : \mathbb{R}^D \to \mathbb{R}$, and $f_3(X) : \mathbb{R}^{M\times N} \to \mathbb{R}$ where $x \in \mathbb{R}$, vector $\vec{x} := [x_1, \cdots, x_D]^T \in \mathbb{R}^D$, and matrix $X := [X_{ij}] \in \mathbb{R}^{M\times N}$.

- The derivative of $f_1(x)$ w.r.t. x is defined as $f_1'(x) = \frac{\partial f_1(x)}{\partial x} := \lim\limits_{\Delta x \to 0} \frac{f_1(x+\Delta x) - f_1(x)}{\Delta x}$.
- The gradient of $f_2(\vec{x})$ w.r.t. $\vec{x}$ is defined as $\nabla_{\vec{x}} f_2(\cdot) = \frac{\partial f_2}{\partial \vec{x}} := [\frac{\partial f_2}{\partial x_1}, \cdots, \frac{\partial f_2}{\partial x_D}]^T \in \mathbb{R}^D$.
- The gradient of $f_3(X)$ w.r.t. X is defined as $\nabla_X f_3(\cdot) = \frac{\partial f_3}{\partial X} := [\frac{\partial f_3}{\partial X_{ij}}] \in \mathbb{R}^{M\times N}$ where $\frac{\partial f_3}{\partial X_{ij}} := \lim\limits_{\Delta X_{ij} \to 0} \frac{f_3(X_{ij}+\Delta X_{ij}) - f_3(X_{ij})}{\Delta X_{ij}}$.

Differential is used to derive the gradient of a function w.r.t. a vector or a matrix in a closed-matrix form.

Definition 2.2. The differential of a function ∂f can be defined as the linear combination of its variables' differentials as following.

- $\partial f_1 := f_1'(x)\, \partial x$ where $f_1'(x)$ is the scope of f_1 at x and $\partial x := \lim\limits_{\Delta x \to 0} \Delta x$.
- $\partial f_2 := \sum\limits_{i=1}^{D} \frac{\partial f_2}{\partial x_i} \partial x_i = (\frac{\partial f_2}{\partial \vec{x}})^T \partial \vec{x}$ where $\partial \vec{x} := [\partial x_1, \cdots, \partial x_D]^T$.
- $\partial f_3 := \sum\limits_{i=1}^{M} \sum\limits_{j=1}^{N} \frac{\partial f_3}{\partial X_{ij}} \partial X_{ij} = Tr(\frac{\partial f_3}{\partial X}^T \partial X)$ where $\partial X = [\partial X_{ij}] \in \mathbb{R}^{M\times N}$ and $Tr()$ represents the trace operation defined as $Tr(A) := \sum\limits_{i=1}^{N} A_{ii}$ for any squared matrix $A \in \mathbb{R}^{N\times N}$.

According to Definition 2.2, we can derive the gradient of a function w.r.t. a matrix or a vector easily by formulating the differential of the function. Additionally, laws of differential and properties of Trace are useful for us to derive the differential of a function.

Theorem 2.1 (Laws of Differential)**.** Assume $A, B \in \mathbb{R}^{M\times N}$, $C \in \mathbb{R}^{N\times M}$, and $D \in \mathbb{R}^{N\times N}$ is invertible; $|D|$ is the determinant and D^* is the adjugate matrix; $F = A \odot B$ is the element-wise product with $F_{ij} = A_{ij}B_{ij}$. Then

1. $\partial(A \pm B) = \partial A \pm \partial B$
2. $\partial(AC) = (\partial A)C + A(\partial C)$
3. $\partial(A^T) = (\partial A)^T$
4. $\partial Tr(D) = Tr(\partial D)$
5. $\partial D^{-1} = -D^{-1}(\partial D)D^{-1}$ (proved by $\partial(DD^{-1}) = (\partial D)D^{-1} = D\,\partial D^{-1} = \partial I = 0$)
6. $\partial|D| = \frac{1}{N}Tr(D^*\,\partial D) = \frac{1}{N}|D|Tr(D^{-1}\,\partial D)$ (proved by $Tr(E\,\partial|D|) = Tr(D^*\,\partial D)$)
7. $\partial(A \odot B) = (\partial A) \odot B + A \odot \partial B$
8. $\partial f \odot (A) = f'(A) \odot \partial A$ where $f'(A) := [\frac{\partial f}{\partial A_{ij}}] \in \mathbb{R}^{M\times N}$

Theorem 2.2 (Properties of Trace)**.** Assume $a \in \mathbb{R}$, matrices $A, B \in \mathbb{R}^{N\times N}$, and $C, D, F \in \mathbb{R}^{M\times N}$. Then

1. $a = Tr(a)$
2. $Tr(A^T) = Tr(A)$
3. $Tr(A \pm B) = Tr(A) \pm Tr(B)$
4. $Tr(CD^T) = Tr(D^TC)$
5. $Tr(C^T(D \odot F)) = Tr((C \odot D)^TF)$

3 Backpropagation in Closed-Matrix Form

The specific manifestation of Backpropagation (BP) is contingent upon the architectural design of the model [2]. Accordingly, in this endeavor, we proceed to deduce the closed-matrix formulation of BP tailored to various canonical machine learning models.

Example 3.1 (Linear Regression)**.** Suppose $L(\vec{w}) = ||X\vec{w} - \vec{y}||^2$ where $\vec{y} \in \mathbb{R}^M$, $\vec{w} \in \mathbb{R}^N$, and $X \in \mathbb{R}^{M\times N}$. Solving $\min_{\vec{w}} L(\vec{w})$ requires $\nabla_{\vec{w}}L(\vec{w}) = 0$, so we need to derive the closed-matrix form of $\nabla_{\vec{w}}L(\vec{w})$.

1. Apply laws of matrix differential

$$\begin{aligned}\partial L &= \partial((X\vec{w} - \vec{y})^T(X\vec{w} - \vec{y}))\\ &= \partial(X\vec{w} - \vec{y})^T(X\vec{w} - \vec{y}) + (X\vec{w} - \vec{y})^T\,\partial(X\vec{w} - \vec{y})\\ &= (X\,\partial w)^T(X\vec{w} - \vec{y}) + (X\vec{w} - \vec{y})^TX\,\partial\vec{w}\end{aligned}$$

2. Add trace operation and apply properties of trace

$$\begin{aligned}Tr(\partial L) &= Tr((X\,\partial w)^T(X\vec{w} - \vec{y}) + (X\vec{w} - \vec{y})^TX\,\partial\vec{w})\\ &= Tr((X\vec{w} - \vec{y})^TX\,\partial\vec{w}) + Tr((X\vec{w} - \vec{y})^TX\,\partial\vec{w})\\ &= Tr(2(X^T(X\vec{w} - \vec{y}))^T\,\partial\vec{w})\end{aligned}$$

3. So, $\nabla_{\vec{w}}L = 2X^T(X\vec{w} - \vec{y})$.

Example 3.2 (Maximum Likelihood Estimation)**.** Suppose the data set $\{\vec{x}_i\}_{i=1}^N$ with $\vec{x}_i \in \mathbb{R}^D$ where $\vec{x}_i \sim P(\vec{x}_i|\mu, \Sigma) := (2\pi)^{-\frac{1}{2}}|\Sigma|^{-\frac{1}{2}}e^{-\frac{D}{2}(\vec{x}_i-\vec{\mu})^T\Sigma^{-1}(\vec{x}_i-\vec{\mu})}$. The Maximum Likelihood Estimation (MLE) problem is

$$\begin{aligned}\max_{\vec{\mu},\Sigma}\prod_{i=1}^{N}P(\vec{x}_i|\vec{\mu},\Sigma) &\iff \max_{\vec{\mu},\Sigma} -\frac{ND}{2}log(2\pi) - \frac{N}{2}log(|\Sigma|) - \frac{1}{2}\sum_{i=1}^{N}(\vec{x}_i - \vec{\mu})^T\Sigma^{-1}(\vec{x}_i - \vec{\mu})\\ &\iff \min_{\vec{\mu},\Sigma} L(\vec{\mu},\Sigma) = Nlog(|\Sigma|) + \sum_{i=1}^{N}(\vec{x}_i - \vec{\mu})^T\Sigma^{-1}(\vec{x}_i - \vec{\mu})\end{aligned}$$

The $\min\limits_{\vec{\mu}} L$ is independent to $\min\limits_{\Sigma} L$ and the MLE of Σ requires $\nabla_\Sigma L = 0$, so we derive $\nabla_\Sigma L$ as following.

1. Apply laws of matrix differential

$$\begin{aligned}\partial L &= N\,\partial(log(|\Sigma|)) + D\sum_{i=1}^{N}(\vec{x}_i - \vec{\mu})^T\,\partial(\Sigma^{-1})(\vec{x}_i - \vec{\mu}) \\ &= N(|\Sigma|)^{-1}\,\partial|\Sigma| - \sum_{i=1}^{N}(\vec{x}_i - \vec{\mu})^T\Sigma^{-1}\,\partial(\Sigma)\Sigma^{-1}(\vec{x}_i - \vec{\mu})\end{aligned}$$

2. Add trace operation and apply properties of trace

$$\begin{aligned}Tr(\partial L) &= Tr(N(|\Sigma|)^{-1}\,\partial|\Sigma|) - Tr(\sum_{i=1}^{N}(\vec{x}_i - \vec{\mu})^T\Sigma^{-1}\,\partial(\Sigma)\Sigma^{-1}(\vec{x}_i - \vec{\mu})) \\ &= \frac{N}{D}|\Sigma|^{-1}|\Sigma|Tr(\Sigma^{-1}\,\partial\Sigma) - Tr(\sum_{i=1}^{N}\Sigma^{-1}(\vec{x}_i - \vec{\mu})(\vec{x}_i - \vec{\mu})^T\Sigma^{-1}\,\partial\Sigma) \\ &= Tr(\frac{N}{D}\Sigma^{-1}\,\partial\Sigma - \sum_{i=1}^{N}\Sigma^{-1}(\vec{x}_i - \vec{\mu})(\vec{x}_i - \vec{\mu})^T\Sigma^{-1}\,\partial\Sigma)\end{aligned}$$

3. So, $\nabla_\Sigma L = \frac{N}{D}\Sigma^{-1} - \sum\limits_{i=1}^{N}\Sigma^{-1}(\vec{x}_i - \vec{\mu})(\vec{x}_i - \vec{\mu})^T\Sigma^{-1}$

Example 3.3 (Multi-variable Logistics Regression)**.** Suppose $\vec{h} = W\vec{x}$, $L = -\vec{y}^T log \odot (\sigma(\vec{h}))$ where $\vec{x} \in \mathbb{R}^N$, $W \in \mathbb{R}^{M\times N}$, $\vec{y} \in \{0,1\}^M : \sum\limits_{i=1}^{M} y_i = 1$, $\sigma(\vec{h}) = \frac{exp\odot(\vec{h})}{\vec{1}^T exp\odot(\vec{h})}$ with $\vec{1} = [1,1,\cdots,1]^T$ and $dim(\vec{1}) = M$. Formulate the backpropagation process $L \to \nabla_{\vec{h}}L \to \nabla_W L$ in a closed-matrix form.

1. $log \odot (\sigma(\vec{h})) = log \odot (\frac{exp\odot(\vec{h})}{\vec{1}^T exp\odot(\vec{h})}) = \vec{h} - log(\vec{1}^T exp \odot (\vec{h}))\cdot\vec{1}$

2. So, $L = -\vec{y}^T\,\underbrace{(\vec{h} - log(\vec{1}^T exp\odot(\vec{h}))\cdot\vec{1})}_{\vec{z}} = -\vec{y}^T\vec{z}$

3. $\partial L = Tr(-\vec{y}^T\,\partial\vec{z})$

   3.1. Apply laws of matrix differential $\partial(\vec{z}) = \partial\vec{h} - \partial(log(\vec{1}^T exp\odot(\vec{h}))\cdot\vec{1}) = \partial\vec{h} - \frac{1}{\vec{1}^T exp\odot(\vec{h})}\vec{1}^T\,\partial(exp\odot(\vec{h})\cdot\vec{1}) = \partial\vec{h} - \vec{1}\cdot\frac{1}{\vec{1}^T exp\odot(\vec{h})}\vec{1}^T\,\partial(exp\odot(\vec{h})) = \partial\vec{h} - \vec{1}\cdot\frac{1}{\vec{1}^T exp\odot(\vec{h})}\vec{1}^T(exp'(\vec{h})\odot(\partial\vec{h}))$

   3.2. Add trace and apply its properties

$$\begin{aligned}Tr(\partial L) &= -Tr(\vec{y}^T\,\partial\vec{h}) + Tr(\underbrace{\vec{y}^T\vec{1}}_{=1}\cdot\frac{1}{\vec{1}^T exp\odot(\vec{h})}\underbrace{\vec{1}^T(exp'(\vec{h})\odot(\partial\vec{h})))}_{Trace\ property\ 5} \\ &= -Tr(\vec{y}^T\,\partial\vec{h}) + \frac{1}{\vec{1}^T exp\odot(\vec{h})}Tr((\underbrace{\vec{1}\odot exp'(\vec{h})}_{=exp\odot(\vec{h})})^T\,\partial\vec{h}) \\ &= -Tr(\vec{y}^T\,\partial\vec{h}) + Tr(\frac{(exp\odot(\vec{h}))^T}{\vec{1}^T exp\odot(\vec{h})}\,\partial\vec{h}) \\ &= Tr((\sigma(\vec{h}) - \vec{y})^T\,\partial\vec{h})\end{aligned}$$

   3.3. So, $\nabla_{\vec{h}}L = \sigma(\vec{h}) - \vec{y}$

4. $\partial L = Tr(\nabla_{\vec{h}}L^T\,\partial\vec{h})$

   4.1. $\partial\vec{h} = \partial(W)\vec{x}$

4.2. $\partial L = Tr(\nabla_{\vec{h}} L^T \, \partial(W)\vec{x}) = Tr((\nabla_{\vec{h}} L \vec{x}^T)^T \, \partial W)$

4.3. So, $\nabla_W L = \nabla_{\vec{h}} L \vec{x}^T$

Example 3.4 (Fully Connected Neural Networks)**.** The loss $L = -\vec{y}^T log\odot(\sigma_2(\vec{h}_2))$ where $\sigma_2(\vec{h}_2) = \frac{exp\odot(\vec{h}_2)}{\vec{1}^T exp\odot(\vec{h}_2)}$, $\vec{1} = [1, \cdots, 1]^T \in \{1\}^{M_2}$, $\vec{y} \in \{0,1\}^{M_2} : \sum_{i=1}^{M_2} y_i = 1$; 2nd layer $\vec{h}_2 = W_2\vec{z}_1 + \vec{b}_2$ where $W_2 \in \mathbb{R}^{M_2 \times M_1}$,$\vec{b}_2 \in \mathbb{R}^{M_2}$, $\vec{z}_1 = \sigma_1 \odot (\vec{h}_1)$ and $\sigma_1 \odot (\cdot)$ is an elementwise Sigmoid; 1st layer $\vec{h}_1 = W_1\vec{x} + \vec{b}_1$ where $\vec{x} \in \mathbb{R}^{M_0}$, $W_1 \in \mathbb{R}^{M_1 \times M_0}$,$\vec{b}_1 \in \mathbb{R}^{M_1}$. Formulate the backpropagation process $L \to \nabla_{\vec{h}_2} L \to \nabla_{\vec{z}_1} L \to \nabla_{\vec{h}_1} L \to \nabla_{W_1} L$ in a closed-matrix form.

1. Example 3.3 indicates that $\nabla_{\vec{h}_2} L = \sigma_2(\vec{h}_2) - \vec{y}$, so for $\partial L = Tr(\nabla_{\vec{h}} L^T \, \partial \vec{h}_2)$ we have

   1.1. $\partial \vec{h}_2 = W_2 \, \partial(\vec{z}_1)$

   1.2. $\partial L = Tr((W_2^T \nabla_{\vec{h}} L)^T \, \partial \vec{z}_1)$

   1.3. So, $\nabla_{\vec{z}_1} L = W_2^T \nabla_{\vec{h}} L$

2. $\partial L = Tr(\nabla_{\vec{z}_1} L^T \, \partial \vec{z}_1)$

   2.1. $\partial \vec{z}_1 = \partial \sigma_1 \odot (\vec{h}_1) = \sigma_1'(\vec{h}_1) \odot \partial \vec{h}_1$ where $\sigma_1'(\vec{h}_1) = \sigma_1 \odot (\vec{h}_1) \odot (1 - \sigma_1 \odot (\vec{h}_1))$

   2.2. $\partial L = Tr(\nabla_{\vec{z}_1} L^T (\sigma_1'(\vec{h}_1) \odot \partial \vec{h}_1)) = Tr((\nabla_{\vec{z}_1} L \odot \sigma_1'(\vec{h}_1))^T \, \partial \vec{h}_1)$

   2.3. So, $\nabla_{\vec{h}_1} L = \nabla_{\vec{z}_1} L \odot \sigma_1'(\vec{h}_1)$

3. $\partial L = Tr(\nabla_{\vec{h}_1} L^T \, \partial \vec{h}_1)$

   3.1. $\partial \vec{h}_1 = (\partial W_1)\vec{x}$

   3.2. $\partial L = Tr(\nabla_{\vec{h}_1} L^T (\partial W_1)\vec{x}) = Tr((\nabla_{\vec{h}_1} L \vec{x}^T)^T \, \partial W_1)$

   3.3. So, $\nabla_{W_1} L = \nabla_{\vec{h}_1} L \vec{x}^T$

Example 3.5 (Convolutional Neural Networks)**.** $X \overset{\leftarrow}{\underset{s=1}{*}} K = H$ represents that the convolution between $X \in \mathbb{R}^{3\times 3}$ and the kernel $K \in \mathbb{R}^{2\times 2}$ with stride 1 is $H \in \mathbb{R}^{2\times 2}$.

$$\begin{bmatrix} X_{11} & X_{12} & X_{13} \\ X_{21} & X_{22} & X_{23} \\ X_{31} & X_{32} & X_{33} \end{bmatrix} \overset{\leftarrow}{\underset{s=1}{*}} \begin{bmatrix} K_{11} & K_{12} \\ K_{21} & K_{22} \end{bmatrix} = \begin{bmatrix} H_{11} & H_{12} \\ H_{21} & H_{22} \end{bmatrix}$$

where $H_{11} = X_{11}K_{11} + X_{12}K_{12} + X_{21}K_{21} + X_{22}K_{22}$, $H_{12} = X_{12}K_{11} + X_{13}K_{12} + X_{22}K_{21} + X_{23}K_{22}$, $H_{21} = X_{21}K_{11} + X_{22}K_{12} + X_{31}K_{21} + X_{32}K_{22}$, and $H_{22} = X_{22}K_{11} + X_{23}K_{12} + X_{32}K_{21} + X_{33}K_{22}$. In the convolution $X \overset{\leftarrow}{\underset{s=1}{*}} K = H$, assume $L(H) \in \mathbb{R}$ and $\nabla_H L$ are given, derive $\nabla_K L$ and $\nabla_X L$ in their closed matrix forms.

1. Write $X \overset{\leftarrow}{\underset{s=1}{*}} K = H$ to a matrix multiplication $\hat{K}\vec{x} = \vec{h}$:

$$\begin{bmatrix} K_{11} & K_{12} & 0 & K_{21} & K_{22} & 0 & 0 & 0 & 0 \\ 0 & K_{11} & K_{12} & 0 & K_{21} & K_{22} & 0 & 0 & 0 \\ 0 & 0 & 0 & K_{11} & K_{12} & 0 & K_{21} & K_{22} & 0 \\ 0 & 0 & 0 & 0 & K_{11} & K_{12} & 0 & K_{21} & K_{22} \end{bmatrix} \begin{bmatrix} X_{11} \\ X_{12} \\ X_{13} \\ X_{21} \\ X_{22} \\ X_{23} \\ X_{31} \\ X_{32} \\ X_{33} \end{bmatrix} = \begin{bmatrix} H_{11} \\ H_{12} \\ H_{13} \\ H_{21} \end{bmatrix}$$

2. $\nabla_H L \iff \nabla_{\vec{h}} L$.

3. $\partial L = Tr(\nabla_{\vec{h}} L^T \, \partial \vec{h})$.

3.1. $\partial \vec{h} = \partial(\hat{K}\vec{x}) = (\partial \hat{K})\vec{x} + \hat{K}(\partial \vec{x})$

3.2. So $\partial L = Tr(\nabla_{\vec{h}} L^T (\partial \hat{K})\vec{x}) + Tr(\nabla_{\vec{h}} L^T \hat{K}\, \partial \vec{x}) = Tr((\nabla_{\vec{h}} L \vec{x}^T)^T\, \partial \hat{K}) + Tr((\hat{K}^T \nabla_{\vec{h}} L)^T\, \partial \vec{x})$

3.3. $\nabla_{\hat{K}} L = \nabla_{\vec{h}} L \vec{x}^T$ and $\nabla_{\vec{x}} L = \hat{K}^T \nabla_{\vec{h}} L$

4 Conclusion and Discussion

This paper introduces a closed-matrix formulation for the Backpropagation algorithm applied across a spectrum of machine learning models, encompassing linear regression, maximum likelihood estimation, multi-variable logistic regression, fully connected neural networks, and convolutional neural networks. The proposed matrix calculus allows the representation of these models in a unified closed-matrix form, extending its applicability to other gradient-based models.

For forthcoming investigations, we explore novel differential laws in Example 3.5, utilizing ∂K and ∂X to express $\partial(X \underset{s=1}{\overset{\leftarrow}{*}} K)$. We leverage trace properties to transform $Tr(H^T(X \underset{s=1}{\overset{\leftarrow}{*}} K))$ into $Tr(F(H, X, K)^T X)$ and $Tr(G(H, X, K)^T K)$. Additionally, we extend our formulation to closed-matrix BP processes for other prominent deep learning models, such as Generative Adversarial Networks (GANs) and Transformers [11]. Lastly, we delve into the analysis of Batch Normalization [7] and Residual Connections [5] employing the mathematical foundations presented in this paper.

References

[1] Shun-ichi Amari. Backpropagation and stochastic gradient descent method. *Neurocomputing*, 5(4-5):185–196, 1993.

[2] Leonid Berlyand and Pierre-Emmanuel Jabin. Mathematics of deep learning. In *Mathematics of Deep Learning*. De Gruyter.

[3] Saeed Damadi, Golnaz Moharrer, Mostafa Cham, and Jinglai Shen. The backpropagation algorithm for a math student. In *2023 International Joint Conference on Neural Networks (IJCNN)*, pages 01–09. IEEE, 2023.

[4] Alan Edelman, Ekin Akyurek, and Yuyang Wang. Backpropagation through back substitution with a backslash. *arXiv preprint arXiv:2303.15449*, 2023.

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part IV 14*, pages 630–645. Springer, 2016.

[6] Robert Hecht-Nielsen. Theory of the backpropagation neural network. In *Neural networks for perception*, pages 65–93. Elsevier, 1992.

[7] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. pmlr, 2015.

[8] Timothy P Lillicrap, Adam Santoro, Luke Marris, Colin J Akerman, and Geoffrey Hinton. Backpropagation and the brain. *Nature Reviews Neuroscience*, 21(6):335–346, 2020.

[9] Raul Rojas and Raúl Rojas. The backpropagation algorithm. *Neural networks: a systematic introduction*, pages 149–182, 1996.

[10] Paul J Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

[11] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations*, pages 38–45, 2020.