Temporal Learning Capacity of Transformers in Non-Markovian Dynamical Systems

by

Benjamin Shih

Submitted to the Department of Applied Mathematics
in partial fulfillment of the requirements for honors in the degree of

Bachelor of Science in Applied Mathematics and Computer Science

at

BROWN UNIVERSITY

April 2024

Author . . . . . .
Department of Applied Mathematics
April 2024

Certified by . . . . . .
George Em Karniadakis, Thesis Supervisor
Professor of Applied Mathematics

Certified by . . . . . .
Khemraj Shukla, Thesis Reader
Assistant Professor of Applied Mathematics

Accepted by . . . . . .
George Em Karniadakis
Professor of Applied Mathematics

Abstract

Recent advances in the scientific computing domain center around data-driven methods for approximating solutions to differential equations that arise in a variety of domains ranging from materials engineering to neuroscience. Of particular interest are those methods which provide generalized (i.e. mesh and parameter free) results such as the Deep Operator Network (DeepONet), which are able to learn *operators* to predict classes of solutions rather than solutions for particular instances of a system (e.g. based on initial conditions, etc.). DeepONets and similar models are realizations of results in approximation theory developed in the prior decades. Concurrently, the natural language processing domain saw a recent rise in popularity of the Transformer model due to its immense success in transduction tasks. Based on the attention mechanism, the Transformer model introduces a new approach to sequence to sequence modeling from its predecessor, recurrent neural networks. In this thesis, we survey some fundamental results in approximation theory culminating in the Universal Approximation Theorem for Nonlinear Operators on which DeepONets are based, then apply and provide results for the learning capacity of modern attention models in approximating tempered fractional differential systems.

Acknowledgments

I would like to thank Dr. George Em Karniadakis for his expert supervision, teaching, and guidance over the course of the past couple years. In a short period of time, Dr. Karniadakis has exponentially accelerated my academic and research maturities and helped me to develop my curiosities and the skills necessary to explore them.

I would also like to thank my co-advisor, Dr. Zhongqiang Zhang for his unwavering support and time in supervising and guiding me over the course of this work. This certainly would not have been possible without the consistent meetings and discussions with Dr. Zhang.

In addition, I would like to thank Dr. Khemraj Shukla for all the hours of mathematical discussion, practical implementation advice, and theoretical wisdom imparted over the course of the project.

This work has also greatly benefited from the time I shared with other academics; I am thankful to Dr. Wenjun Zhao who helped to further my interests in approximation, always offered support, and gave me invaluable academic advice. In addition, I am grateful to Dr. Li-San Wang at the University of Pennsylvania who kind enough to support me in my first research experience as an undergraduate. My gratitude also extends to Alexey Izmailov for the random yet very relevant late-night insights.

Of course, I give thanks to my family; to my mom who raised me and my dad who offered unwavering support regardless of what I pursued. I'm eternally indebted to have such loving parents so that I could explore my passions at every point of my life. Finally, I'd like to thank my fencing coach and father figure, Alexey Kuznetsov, for everything he has done to get me where I am today. I am thankful for the guidance, pain and happiness we shared, and patience over the years he raised me as an athlete and person. He taught me resilience, hard work, and grit on top of just about everything else that I am today. Without him, I certainly would not have been where I am to write this today. Спасибо Kuzy.

This research was conducted using computational resources and services at the Center for Computation and Visualization, Brown University.

Contents

SECTION

1 INTRODUCTION

1.1 Motivation

Following the large successes of deep learning in natural language processing [39], computer vision [13], and numerous other fields, increases in computational power and the development of modern learning models have resulted in the rapid growth of the scientific machine learning field. In this field deep learning methods are often combined with traditional numerical methods from engineering or physics for various applications such as medical imaging, weather forecasting, and discovery of solutions to physical systems. A large problem of interest in this domain is how learning models can be identified with, or at least approximate, nonlinear maps between input and output data. Physics-Informed Neural Networks (PINNs) emerged as a way of combining deep learning with a priori knowledge of a system (e.g. conservation laws, domain expertise, etc.) to result in data-driven solutions to nonlinear parameterized partial differential equations [33]. Soon after, several variations of the PINN were introduced, each specialized to their own applications. Additional research has been conducted on how best to optimize the training of these models. For example, Bayesian PINNs [41], parallel PINNs [36], adaptive PINN weights [10] each contribute to the PINN literature in particular applications. The body of work around PINNs largely aims to compete with traditional grid and mesh-based methods such as finite difference and element methods for numerical solutions to PDEs.

Building upon the work of PINNs, Deep Operator Networks (DeepONets) were introduced as a solution to approximating *operators* rather than functions. Soon afterwards, a competing framework, the Fourier Neural Operator (FNO), was introduced with the same purpose of learning complex operators. Much work has been since the introduction of these two frameworks [8, 40], and studies on the strengths and weaknesses of each have been conducted [22]. Generally, the operator learning approach aims to generalize solutions beyond singular families of a parametric system to entire systems through the use of more data and less imposition of physical laws on the model. In particular, rather than learn mappings between vector spaces as PINNs do, neural operators such as the DeepONet and FNO learn mappings between function spaces. It is worthy to note that the

work of neural operators is based upon well-established results in approximation theory extending back multiple decades. Fundamental results from the field establish approximation guarantees for continuous functions, then extend onwards to nonlinear functionals and operators. Hence, the modern work surrounding these methods are based upon realizations of these theories enabled by compute.

The ideas of fractional calculus date back to the times of Leibniz [16] but only recently has the discipline gained traction in the modeling. We now see that its applications are widespread for various desirable properties such as its versatility and ability to capture system nonlocalites through long term memory effects. For example, fractional order models make sense for neuron signals as biological evidence shows that there is residual voltage left after a neuronal activity, which is a phenomenon synonymous to long-term memory. Today, these fractional models are used for a wide variety of phenomena including the aforementioned neuron activity in the brain [31, 38], financial processes [44], and physical systems [14] employ fractional calculus. More recent efforts have seen fractional calculus being developed and studied in the context of machine learning; for example, in accelerated optimization through fractional gradients [35] and fractional PINNs [27]. In the original DeepONet work, it is shown that the fractional derivative operator can be learned to great precision using fully connected layers for both the branch and trunk nets [21]. Another layer of complexity can be incorporated to fractional system with the addition of a *tempering* term. Tempered fractional calculus extends the original fractional framework by "tempering" power laws using an exponential factor. This offers another set of features that heuristically corresponds to how large the "window of memory" is in the system. In practice, this can affect the smoothness of function trajectories, convergence, etc. as discussed in [34].

Returning to the present, perhaps one of the biggest breakthroughs in modern artificial intelligence is the attention mechanism, proposed in [39], which is the central mechanism to *Transformer* models. These are usually constructed in an encoder-decoder structure and have found wide application in natural language processing [32], multi-modal learning [17], and computer vision [7]. Heuristically, Transformers enjoy such widespread success through ability to *contextualize*; i.e. recognize importance and give the necessary attention to specific tokens in sequence to sequence learning tasks.

In this work, we are interested in tying all of the concepts previously discussed together. That is, we study the the learning capacity of the Transformer models as a neural operator in learning tempered fractional systems. In a sense, we treat the Transformer as a surrogate for the tempered fractional differential operator for which we usually do not have a closed form for. We analyze the capability of the Transformer to successfully learn these systems and, in particular, the temporal learning capacity of these models. That is, we examine the Transformer's ability to capture the aforementioned nonlocalities present in these systems; i.e. whether the model is actually contextualizing and focusing on different parts of the input as the memory in the system increases or decays.

■ 1.2 Outline of Thesis

Broadly, this thesis studies the learning capacity of the successful attention-based learning models in an operator learning context. In particular, we are interested in the ability of these models to capture the temporal features in non-Markovian systems; central to our discussion are tempered fractional initial value problems. The work is structured as follows:

1. We provide a brief survey of the evolution of the approximation theory landscape over the last several decades, culminating in the Generalized Universal Approximation Theorem for Operators upon which the DeepONet is based. Additionally, some technical preliminaries are established to contextualize the problems studied.

2. We provide and discuss the numerical results that were obtained in experiments of Transformer models learning various tempered fractional systems.

3. We analyze the ability of these models to capture the nonlocality in these problems.

SECTION

2 BACKGROUND

This section establishes the relevant background for our work; first, we formally define the operator learning problem and contextualize it within the historical advances in approximation theory. Then, we cover some recent advances in the field in related literature and provide the necessary technical foundations for our specific problem setting and methodologies employed later on.

■ 2.1 Operator Learning

Let $\Omega_i \subset \mathbb{R}^m$ and $\Omega_o \subset \mathbb{R}^n$ be two compact domains on which there are metric function spaces $(\mathcal{F}, d_{\mathcal{F}})$ and $(\mathcal{U}, d_{\mathcal{U}})$, respectively. Suppose that we have functions $f \in \mathcal{F} : x \in \Omega_i \mapsto f(x) \in \mathbb{R}$ and $u \in \mathcal{U} : y \in \Omega_o \mapsto u(y) \in \mathbb{R}$. Here, f is our input function and u is the output function, and we denote the operator mapping f to u as

$$\mathcal{G}^* : \mathcal{F} \ni f \mapsto u \in \mathcal{U}$$

In the practical computational setting, we cannot directly use infinite-dimensional data, such as the true f or u. Instead, we take use an approximation of f with discretized points $\{t_i\}_{i=0}^n \subset \Omega_i$ and observations $f_O = \{f(t_0), \ldots, f(t_n)\} \subset \mathbb{R}^n$. Operator learning is then the problem of the construction of a network $\mathcal{G}_\theta : \mathbb{R}^n \ni f_O \mapsto \mathcal{G}_\theta[f_O](\cdot) \in \mathcal{U}$ where $\theta \in \Theta$ for a parameter space Θ such that $\mathcal{G}_\theta$ approximates $\mathcal{G}^*$. Thus, we seek the solution to the following optimization problem:

$$\min_{\theta \in \Theta} \|\mathcal{G}_\theta[f](\cdot) - \mathcal{G}^*[f](\cdot)\|$$

for some appropriately defined norm.

■ 2.2 A Brief Survey of Approximation Theory

Approximation theory has been developed for many decades, originally as an extension of functional analysis, and within the space there are some extraordinary results such as

the following theorem due to Chen and Chen [4] on which modern neural operators are based.

Polynomial Approximation A classical result of approximation theory is that the continuous functions on bounded intervals can be approximated arbitrarily well by polynomials. This is the result proven by Weierstrass in the late 1800s and forms the base of much of modern approximation theory.

Theorem 2.1 (Weierstrass). *Given a function $f \in C([a,b])$ and $\epsilon > 0$, there exists an algebraic polynomial p such that $|f(x) - p(x)| < \epsilon$ for all $x \in [a,b]$, or equivalently, $\|f - p\|_\infty < \epsilon$.*

where $C(U)$ denotes the space of all continuous functions on a domain U and $\|\cdot\|_\infty$ denotes the supremum norm. There is a great body of literature surrounding polynomial approximation using well studied techniques such as splines and interpolants. For example, the Chebyshev interpolants are known to possess many desirable properties relevant to polynomial approximation such as exponential convergence rates for differentiable functions.

Approximation with Neural Networks With the rise of machine learning techniques and applications, modern approaches to approximation theory involve results related to the approximation capability of neural networks, which joins the literature on fundamental polynomial and rational approximation. A fundamental result due to Pinkus [30] states that feedforward networks with one hidden layer can approximate continuous functions of any dimension arbitrarily well with respect to the supremum norm.

Theorem 2.2 (Pinkus). *Consider the network space*

$$\mathcal{M}(\sigma) = \text{span}\{\sigma(w \cdot x + b) : w \in \mathbb{R}^d, b \in \mathbb{R}\}$$

where $\sigma \in C(\mathbb{R})$. Then $\mathcal{M}(\sigma)$ is dense in $C(\mathbb{R}^n)$ with respect to the supremum norm on compact sets if and only if σ is not a polynomial.

Although we will omit the proofs of the major theorems stated, we will state and prove the 1-dimensional case of Theorem 2.2 to show how such results may be approached and rigorously substantiated.

Lemma 2.3 (1-D Pinkus). *Denote the one dimensional counterpart of $\mathcal{M}(\sigma)$ by*

$$\mathcal{N}(\sigma) = \text{span}\{\sigma(wx + b) : w, b \in \mathbb{R}\}$$

For any nonpolynomial $\sigma \in C^\infty(\mathbb{R})$, $\mathcal{N}(\sigma)$ is dense in $C(\mathbb{R})$.

Proof. Since we have that σ is smooth, then there is some $b_0 \in \mathbb{R}$ such that

$$\frac{\sigma((w+h)x + b_0) - \sigma(wx + b_0)}{h} \in \mathcal{N}(\sigma), \quad \forall h \neq 0$$

If we take $w = 0$, we have

$$\frac{\mathrm{d}}{\mathrm{d}w}\sigma(wx + b_0)\Big|_{w=0} = x\sigma'(b_0) \in \overline{\mathcal{N}(\sigma)}$$

In general, for $k = 1, 2, \ldots$, we have

$$\frac{\mathrm{d}^k}{\mathrm{d}w^k}\sigma(wx + b_0)\Big|_{w=0} = x^k\sigma^{(k)}(b_0) \in \overline{\mathcal{N}(\sigma)}$$

where $\overline{\mathcal{N}(\sigma)}$ is the closure of $\mathcal{N}(\sigma)$. That is, each of the k-th derivatives are contained in the closure of the network space. Since $\sigma^{(k)}(b_0) \neq 0$ for every k, then $\overline{\mathcal{N}(\sigma)}$ contains all polynomials. Then we have by Theorem 2.1 density of $\overline{\mathcal{N}(\sigma)}$ in $C(\mathbb{R})$ and hence, density of $\mathcal{N}(\sigma)$ in $C(\mathbb{R})$. □

The standard procedure to extend to arbitrary dimensions from Lemma 2.3 involves the relaxation of the smoothness and introduction of *ridge* functions; the full proof can be found in [30]. Moving beyond continuous functions, Chen and Chen proved in [3] that neural networks are universal approximators for continuous functionals, and in [4], proved the extension to nonlinear operators, which we state here.

Theorem 2.4 (Universal Approximation Theorem for Nonlinear Operators)**.** *Suppose that σ is a continuous non-polynomial function, X is a Banach space, $K_1 \subset X$, $K_2 \subset \mathbb{R}^d$ are two compact sets in X and $\mathbb{R}^d$, respectively, V is a compact set in $C(K_1)$, G is a nonlinear continuous operator mapping V into $C(K_2)$. Then for any $\epsilon > 0$, there are positive integers n, p and m, constants $c_i^k, \xi_{ij}^k, \theta_i^k, \zeta_k \in \mathbb{R}, w_k \in \mathbb{R}^d, x_j \in K_1, i = 1, \ldots, n, k = 1, \ldots, p$, and $j = 1, \ldots, m$ such that*

$$\left| G(u)(y) - \sum_{k=1}^{p}\sum_{i=1}^{n} c_i^k \sigma\left(\sum_{j=1}^{m} \xi_{ij}^k u(x_j) + \theta_i^k\right)\sigma(w_k \cdot y + \zeta_k) \right| < \epsilon$$

holds for all $u \in V$ and $y \in K_2$. Here, $C(K)$ is the Banach space of all continuous functions defined on K with norm

$$\|f\|_{C(K)} = \max_{x \in K} |f(x)|$$

Theorem 2.4 is the basis upon which the neural operator DeepONet is formulated, as we will discuss in the next section.

■ 2.3 Related Works

In recent years, there has been a flurry of progress in the operator learning domain, sparked by advances in computational capability and theoretical foundations such as Theorem 2.4. In particular, the research in *neural operators* has been rapidly developing which aims to alleviate the expensive cost associated with fine discretization grids in traditional methods such as finite element and finite difference methods. In these works, neural networks are used as a surrogate for predicting solutions to operators by learning a set of parameters to represent a given operator. This data-driven method comes with many benefits, including but not limited to being mesh-free, high dimensional, and very quick to compute solutions on unseen data [1, 12, 18, 28].

Two popular forms of the neural operator are the DeepONet and FNO. DeepONet builds directly off inspiration from Chen and Chen's work. We can decompose the construction into two components: the "branch" and "trunk" nets. In view of Theorem 2.4, we have the following formulations:

$$\text{Branch: } \sum_{i=1}^{n} c_i^k \sigma \left(\sum_{j=1}^{m} \xi_{ij}^k u(x_j) + \theta_i^k \right)$$
$$\text{Trunk: } \sigma(w_k \cdot y + \zeta_k)$$

That is, the universal approximation theorem provides a network structure which is the formulation of DeepONet [21]. The trunk network takes in the spatial (or temporal) data y, while the branch network takes as input the sensor data of the input functions $u(x_1), \ldots, u(x_m)$. The trunk layer then outputs $t_1, \ldots, t_p$ while the branch layer outputs b_i for $i = 1, 2, \ldots, p$. The network output is then

$$G(u)(y) \approx \sum_{i=1}^{p} b_i(u(x_1), \ldots, u(x_m)) t_i(y)$$

which can be identified with the formulation in the universal approximation theorem when both networks are chosen as single hidden layer networks. Theorem 2.4 has been extended in the original work of the DeepONet to establish that indeed, the DeepONet is a universal operator approximator [21]. The extension is as follows:

Theorem 2.5 (Generalized Universal Approximation Theorem for Operators)**.** *Suppose that X is a Banach space, $K_1 \subset X$, $K_2 \subset \mathbb{R}^d$ are two compact sets in X and $\mathbb{R}^d$, respectively, V is a compact set in $C(K_1)$. Assume that $G : V \to C(K_2)$ is a nonlinear continuous operator. then, for any $\epsilon > 0$, there exist positive integers, m, p, continuous vector fuctions $g : \mathbb{R}^m \to \mathbb{R}^p$, $f : \mathbb{R}^d \to \mathbb{R}^p$, and, $\{x_i\}_{i=1}^m \in K_1$ such that*

$$|G(u)(y) - \langle g(u(x_1), u(x_2), \ldots, u(x_m)), f(y) \rangle| < \epsilon$$

holds for all $u \in V$ and $y \in K_2$, where $\langle \cdot, \cdot \rangle$ denotes the dot product in $\mathbb{R}^p$. Furthermore, the functions g and f can be chosen as diverse classes of neural networks, which satisfy the classical universal approximation theorem of functions; for example, (stacked/unstacked) fully connected neural networks, residual neural networks, and convolutional neural networks.

The proof for this result can be found in the original DeepONet paper [21]. Another popular neural operator is the Fourier Neural Operator (FNO) [18], which extends the work of the original neural operator [1] by replacing the kernel with the Fourier integral in order to parametrize the operator in Fourier space. The work cited superior accuracy and computational efficiency and multiple comparisons between the two frameworks have been conducted on problems varying geomtric complexity, dimensionality, etc. [22].

Overall, the current state of the art in scientific machine learning is rapidly expanding in both theoretical underpinnings as well as practical executions. This can be seen in the work of Physics-informed Neural Networks (PINNs) [33], DeepONet, FNO, as well as variations and other frameworks such as the OFormer [19] which heavily inspired the architecture in our work, FourierFormer [25], etc.

■ 2.4 Technical Preliminaries

In this section we give the definitions for the tempered fractional calculus which will be the primary mechanism involved in the operator learning problems this work investigates.

Tempered Fractional Calculus First, we briefly introduce the concepts of (tempered) fractional derivatives. Over time, various definitions extending the integer-order derivative and integral have arisen, each tailored to various purposes and applications. The classical fractional derivative is the Riemann-Louiville fractional derivative, upon which variations were developed in order to address issues in stability, discontinuity, etc. We provide the two definitions of the fractional derivative: the Riemann-Liouville [9] and Caputo [43] fractional derivatives.

Definition 2.6 (Riemann-Liouville fractional derivative)**.** *The Riemann-Liouville fractional derivative of a function $f(t)$ with fractional order $\alpha > 0$ is defined as*

$$ {}^{\text{RL}}\mathcal{D}_t^{\alpha} u(t) = \frac{1}{\Gamma(m-\alpha)} \left[\frac{d^m}{dt^m} \int_0^t (t-s)^{m-\alpha-1} u(s) ds \right] $$

where $m - 1 < \alpha \leq m$, $m \in \mathbb{Z}_+$.

Definition 2.7 (Caputo fractional derivative)**.** *The Caputo fractional derivative of a function $f(t)$ with fractional order $\alpha > 0$ is defined as*

$$ {}_a^{\text{C}}\mathcal{D}_t^{\alpha} u(t) = \frac{1}{\Gamma(n-\alpha)} \int_a^t (t-s)^{n-\alpha-1} u^{(n)}(s) ds $$

where $n-1<\alpha<n$ *and* $n\in\mathbb{Z}_+$.

Note that in both cases, it is the non-integer fractional order α of the derivative that gives rise to the non-Markovian properties exhibited by systems involving fractional derivatives. In addition to this, we have tempering of a system, which, from a heuristic perspective, controls how large of a temporal window is taken into account in the non-localities. In this work, we employ the tempered fractional Caputo derivative [24] in the systems that are studied; the definition is as follows.

Definition 2.8 (Tempered fractional derivative)**.** *The tempered fractional Caputo derivative with temper order* σ *is defined as*

$$ {}_a^{\mathrm{C}}\mathcal{D}_t^{\alpha,\sigma}u(t)=\frac{e^{-\sigma t}}{\Gamma(n-\alpha)}\int_a^t(t-s)^{n-\alpha-1}\frac{d^n}{ds^n}(e^{\sigma s}u(s))ds $$

Clearly, if $\sigma=0$*, then this reduces to the standard Caputo fractional derivative with fractional order* α.

■ 2.5 Transformers

In recent years, the landscape of artificial intelligence (AI) has undergone a profound transformation, catalyzed by the emergence of transformer-based architectures and attention mechanisms. This paradigm shift in neural network design owes its origins to the groundbreaking work [39], which introduced the Transformer model as a revolutionary architecture for sequence transduction tasks, notably excelling in machine translation. Unlike conventional recurrent neural networks (RNNs) and convolutional neural networks (CNNs), which rely on sequential processing and fixed-length receptive fields, transformers harness the power of self-attention mechanisms to capture long-range dependencies and contextual information within input sequences.

At the core of transformer models lies the attention mechanism, a pivotal component enabling dynamic weighting of input elements based on their relevance to each other. This mechanism allows each element in a sequence to attend to all other elements, facilitating the aggregation of context-rich information. The concept of attention draws inspiration from human cognition, where focus is dynamically allocated to different elements in a stream of information, enabling contextual understanding and effective decision-making. By integrating attention mechanisms into neural network architectures, transformers enable machines to mimic this cognitive process, thereby enhancing their ability to comprehend and generate complex sequences of data.

The remarkable efficacy of attention mechanisms in contextualizing information stems from their inherent flexibility and adaptability. Unlike traditional pooling operations, which aggregate information indiscriminately, attention mechanisms enable selective processing of input elements, dynamically adjusting their importance based on contextual

cues. This adaptive behavior allows transformers to capture nuanced relationships within input sequences, discerning subtle patterns and dependencies that may span long distances. As a result, transformer-based models excel in tasks requiring comprehensive understanding of context, such as natural language understanding, image understanding, and reinforcement learning.

Furthermore, attention mechanisms offer several key advantages over traditional neural network architectures. By enabling parallel computation across input sequences, transformers achieve superior scalability and efficiency, making them well-suited for processing large volumes of data. Moreover, the self-attention mechanism facilitates bidirectional information flow, allowing each element in the sequence to influence and be influenced by all other elements, thereby capturing complex interactions and dependencies. This bidirectional nature of attention enables transformers to model both local and global context, ensuring robust performance across diverse domains and tasks.

Since the inception of the original Transformer model, a plethora of variants and extensions have been proposed, further enhancing the capabilities of transformer architectures. Models such as BERT (Bidirectional Encoder Representations from Transformers) [6], GPT (Generative Pre-trained Transformer) [32], and many more have achieved remarkable success across various natural language processing (NLP) tasks, including language modeling, question answering, and text summarization. Moreover, transformer-based architectures have been successfully applied to other domains, such as computer vision, speech recognition, and graph-based learning, demonstrating their versatility and efficacy in capturing complex relationships within data.

The advent of transformers and attention mechanisms represents a paradigm shift in AI research, enabling machines to achieve unprecedented levels of contextual understanding and information processing. By leveraging attention mechanisms to dynamically contextualize information, transformer-based models have pushed the boundaries of what is achievable in tasks requiring comprehension of sequential data. As research in this field continues to evolve, transformers are expected to play an increasingly pivotal role in advancing the state-of-the-art in artificial intelligence, driving innovations across a wide range of applications and domains.

SECTION

3 METHODOLOGY

3.1 The Attention Mechanism

The standard attention mechanism used in transformers, introduced in [39] is formulated as

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d}}\right)V$$

where the three matrices Q, K, V are batches of sets of vectors: the query vectors $\{q_i\}$, key vectors $\{k_i\}$, and value vectors $\{v_i\}$. A visual for the original attention mechanism is displayed in Figure 3.1a. However, the softmax in this formulation results in a computational complexity which can be remedied by softmax-free attention, as discussed in [2]. The authors interpret attention as a form of learned basis functions, where the columns of the Q, K, V matrices contain the vector representation of basis functions of the latent representation. From this, they formulate two linearized types of attention, termed the Fourier and Galerkin attentions, which are as follow:

$$\text{Fourier type: } (z_i)_j = \frac{1}{n}\sum_{i=1}^{n}(q_i \cdot k_s)(v^j)_s \approx \int_\Omega \kappa(x_i, \xi)v_j(\xi)d\xi$$

$$\text{Galerkin type: } (z^j)_i = \sum_{l=1}^{d}\frac{(k^l \cdot v^j)}{n}(q^l)_i \approx \sum_{l=1}^{d}\left(\int_\Omega (k_l(\xi)v_j(\xi))d\xi\right)q_l(x_i)$$

The corresponding matrix formulations for these linearized attention types are:

$$\text{Fourier Type: } Z = \frac{1}{n}(\widetilde{Q}\widetilde{K}^\top)V, \quad \text{Galerkin Type: } Z = \frac{1}{n}Q(\widetilde{K}^\top\widetilde{V})$$

where $\widetilde{\cdot}$ denotes the layer normalization operation. To allow for flexibility in the query points, the cross attention mechanism is used in addition to the linear attention. Inspired by the same approximation theorem upon which DeepONet is based, this mechanism allows for independence between the input "sensor" points and query points so that the model can be queried at arbitrary locations not necessarily defined by the sensors. Cross

(a) Standard scaled dot-product attention (b) Original proposed Transformer architecture

Figure 3.1: Standard attention and Transformer architecture; images taken from [39].

attention is formulated as:

$$z_s(y_k) = \sum_{i=1}^{d} \frac{\sum_{j=1}^{n} k_i(x_j) v_s(x_j)}{n} q_i(y_k)$$

where $\{y_k\}$ are discretized points on the output domain and $\{x_j\}$ are the input grid points. Here, similar to the previous interpretation that the columns of Q, K, V matrices contain vector representation of learned bases, $\{k_i(\cdot), v_i(\cdot), q_i(\cdot)\}$ represent three sets of basis functions as discussed in [19]. We can draw a parallel between cross attention and DeepONet, where $q_i(y_k)$ is the output of the trunk net and the inner summation $\sum_{j=1}^{n} k_i(x_j) v_s(x_j)$ represents the output of the branch net.

■ 3.2 Model Architecture

Following the traditional sequence transduction models as discussed in [39], the model takes the form of an encoder-decoder structure (similar to the Transformer displayed in Figure 3.1b), where the encoder maps the input $\{x_i\}$ into a latent representation z from which an output sequence $\{y_j\}$ is generated by the decoder. For all of the experiments that follow, we employ the same Transformer structure sans input dimension (e.g. models that have temper order as a parameter have an additional input dimension relative to those which only take in fractional order). In particular, for the baseline results, a model with an encoder embedding dimension for the sequence was 96 and 4 layers of attention was used. For the decoder, a decoding depth of 3 was used.

■ 3.3 Loss

To train the model, we seek to minimize the minimize the empirical loss incurred. In general, in training operator learning models, such a loss function may take the form

$$\mathcal{L}(\theta) = \frac{1}{m}\sum_{k=1}^{m} \|\mathcal{G}_\theta[f_k] - \mathcal{G}^*[f_k]\|$$

where $\{f_k\}_{k=1}^m$ are input functions to the operator $\mathcal{G}^*$, which is typical of a DeepONet [15]. Minimizing $\mathcal{L}(\theta)$ provides a set of model parameters such that the learned operator $\mathcal{G}_\theta$ approximates the true operator $\mathcal{G}^*$ well according to the specified norm. In particular, we are interested in the L_2 norm for functions $f \in L^2(\Omega)$, which is implemented in practice as the discretized form to adjust for the discrete representation of functions as $\{f(t_i)\}_{i=1}^n$. The formulations for these norms are

$$\|f\|_{L^2} = \left(\int_\Omega |f|^2 d\mu\right)^{\frac{1}{2}} \qquad \|f\|_{\ell^2} = \left(\frac{1}{n}\sum_{i=1}^{n} |f(t_i)|^2\right)^{\frac{1}{2}}$$

The specific loss function we minimize our model on utilizes the *relative* ℓ_2 norm so that the particular form can be written:

$$\mathcal{L}(\theta) = \frac{1}{m}\sum_{k=1}^{m} \frac{\|\mathcal{G}_\theta[f_k] - \mathcal{G}^*[f_k]\|_{\ell^2}}{\|\mathcal{G}^*[f_k]\|_{\ell^2}} \tag{3.1}$$

■ 3.4 Training

Typically, training such models relies on first-order methods such as stochastic gradient descent (SGD). In our particular case, we chose to use the Adam optimizer [11] for its prevalence in problems with many parameters and computational efficiency. Further, we employ the learning rate scheduler known as "1cycle" policy, as described in [37]. To minimize the loss further, we extended model training with L-BFGS [20], an expensive second-order optimizer, after sufficient training of the model on the first order optimizer. We trained the transformer model on input sequences containing the forcing term, input discretization grid, and relevant parameters (i.e. the fractional order term α and the temper order term σ). The operator parameters were broadcasted to match the input sequence dimension for input to the encoder.

Typically, training such models relies on first-order methods such as stochastic gradient descent (SGD). In our particular case, we chose to use the Adam optimizer [11] for its prevalence in problems with many parameters and computational efficiency. Further, we employ the learning rate scheduler known as "1cycle" policy, as described in [37]. To minimize the loss further, we extended model training with L-BFGS [20], an expensive

second-order optimizer, after sufficient training of the model on the first order optimizer. We trained the transformer model on input sequences containing the forcing term, input discretization grid, and relevant parameters (i.e. the fractional order term α and the temper order term σ). The operator parameters were broadcasted to match the input sequence dimension for input to the encoder.

Recently, the Lion optimizer has emerged as a novel approach developed by researchers at Google [5], which introduces unique mechanisms for adaptive parameter updates and dynamic exploration-exploitation trade-offs. Traditionally, optimization algorithms such as the aforementioned stochastic gradient descent (SGD) and its variants have been widely used for training neural networks. However, these algorithms often face challenges in handling non-stationary gradients and complex optimization landscapes, which can hinder convergence speed and generalization performance. In response to these challenges, the Lion Optimizer offers a fresh perspective by leveraging principles from nature to address key limitations in existing optimization techniques. The Lion optimizer adapts dynamically to changing gradients and optimization landscapes and prioritizes promising regions of the parameter space while avoiding local minima and stagnation points.

In comparison to the widely-used Adam optimizer, the Lion optimizer offers several advantages, particularly in scenarios with non-stationary gradients and high-dimensional parameter spaces. While Adam demonstrates strong performance in smooth optimization landscapes, the Lion Optimizer excels in handling dynamic environments and complex optimization tasks. Additionally, the Lion optimizer's adaptive exploration-exploitation trade-offs enable it to discover high-quality solutions more efficiently, leading to improved convergence speed and generalization performance. In our work, we also conducted experiments using this optimizer combined with the polynomial learning rate scheduler to compare to the Adam model.

SECTION

4 EXPERIMENTAL RESULTS

We are primarily interested in studying the approximation capability of Transformers in tempered fractional systems. We provide and discuss the results of our model run on two different problems, the tempered fractional system with a smooth forcing term and the tempered fractional leaky integrate-and-fire (LIF) model. In both of these problems, we study multiple subcases of the problems where different operator parameters are to be approximated.

■ 4.1 Smooth Experiments

As a baseline for tempered fractional problems, we first study the approximation capability of the Transformer for the following system:

$$\begin{cases} {}_{a}^{\mathrm{C}}\mathcal{D}_{t}^{\alpha,\sigma}[u](t) = \mu u(t) + f(t) \\ u(0) = u_0 \end{cases} \tag{4.1}$$

Hence, we are interested in the solution operator

$$\mathcal{S} : (f(t), \alpha, \sigma) \mapsto u(t; \alpha, \sigma)$$

where f denotes the forcing function and u the solution function. In this problem, f is sampled from a Gaussian Process so that $f \in C^{\infty}(\Omega)$. We study the following cases for this problem:

Case 1 As the most fundamental benchmark, we study the learning ability of the Transformer model by examining its performance on data which only varies the forcing term f and the fractional order α; i.e. problems in the form of:

$${}_{a}^{\mathrm{C}}\mathcal{D}_{t}^{\alpha}[u](t) = f(t) \tag{4.2}$$

Specifically, this is an instance of Equation 4.1 in which $\sigma = \mu = 0$ so that the system is purely of fractional order.

Case 2 Next , we add some complexity to the system by setting $\mu = -1$ so that there is dependence on $u(t)$ as well as $f(t)$ so the problem is formulated as:

$$ {}_{a}^{\mathrm{C}}\mathcal{D}_{t}^{\alpha}[u](t) = -u(t) + f(t) \tag{4.3} $$

As before, we have $f \sim \mathcal{GP}$.

Case 3 Finally, we add a layer of complexity in the form of varying temper order to the system. This problem takes the particular form

$$ {}_{a}^{\mathrm{C}}\mathcal{D}_{t}^{\alpha,\sigma}[u](t) = -u(t) + f(t) \tag{4.4} $$

which matches Equation (4.1).

■ 4.2 LIF Experiments

We then turn our attention to the core system of interest, the tempered fractional LIF model. Traditionally, the LIF model is used as a neuron model simulating different parts of the brain [23] and the integer order model was formulated as:

$$ \tau \frac{dV}{dt} = -(V - V_{\text{rest}}) + RI(t) \tag{4.5} $$

where τ is known as a membrane time constant equivalent to RC_m where C_m is the membrane capacitance and R is the membrane resistance. Further, $I(t)$ is the forcing spike function and V_{rest} is the resting membrane potential. Because this model is very accurate and computationally favorable, it has become a benchmark choice for neuron modeling.

With the rise of fractional differential equations for their non-local properties, the LIF model has been updated so that the derivative in Equation 4.5 is of fractional order α. That is, the fractional model that we study is

$$ \tau\, {}_{a}^{\mathrm{C}}\mathcal{D}_{t}^{\alpha}[V](t) = -(V - V_{\text{rest}}) + RI(t) $$

Additionally, in our experiments we analyze the dependence of the derivative on a temper order σ as well, so that we aim to approximate the following operator:

$$ \mathcal{S} : (I(t), \alpha, \sigma) \mapsto V(t; \alpha, \sigma) \tag{4.6} $$

As in the smooth case, we study multiple subcases of the problem in order to gradually

build complexity in the system, as we detail below. Note that the major difference between the LIF and smooth case is that in the LIF case, the forcing term and solution term are both nonsmooth, which is much more challenging to approximate when compared to smooth functions.

Case 1 As in the smooth case, we begin our investigation by only varying the fractional order α, so that the system is of the form:

$$ {}_{a}^{\mathrm{C}}\mathcal{D}_{t}^{\alpha}[u](t) = \frac{1}{\tau}(-u(t) + RI(t)) $$

In this case, the forcing term $I(t)$ is fixed. Note that we relabel the solution V from Equation 4.5 as u to parallel the notation from the smooth case.

Case 2 We are then interested in whether our model can be capture the location and intensity of the spike, so we build upon the previous case by still varying α, but also varying the forcing term $I(t)$. Note that the particular form of the system is the same as in the previous case.

Case 3 Finally, we incorporate the temper order into the system so that the operator under investigation is exactly that described in Equation 4.6.

■ 4.3 Tempered Fractional Diffusion

We then extend our investigation to partial differential systems; in particular, we are interested in the tempered fractional inhomogenous diffusion equation as follows:

$$ \begin{cases} {}_{a}^{\mathrm{C}}\mathcal{D}_{t}^{\alpha,\sigma}[u](x,t) = \mu u_{xx}(x,t) + f(x,t) \\ u(x,0) = u_0 \\ u(0,t) = u(1,t) = 0 \end{cases} \tag{4.7} $$

where $f(x,t)$ is the source term and u_0 is the initial condition. Again, the operator for this problem is

$$ S : (f(x,t), \alpha, \sigma) \mapsto u(x,t;\alpha,\sigma) $$

For the tempered fractional diffusion, we are interested in investigating the approximation capability of our model when we vary the fractional order α and the forcing (source) term $f(x,t)$.

■ 4.4 Data Generation

Below we give a brief description of the how the data for each experiment was generated and the format of each experiment. We used MATLAB solvers for each of the tempered fractional initial value problem/tempered fractional partial differential equation systems based on the work in [42]. In each case, we sampled the fractional and temper parameter uniformly, e.g. for α we sampled equispaced points on the interval $[0.11, 0.99]$ (we did not sample for $\alpha \leq 0.1$ due to numerical instability in the solver solutions). This was done for every case of the forcing terms, so that each batch of forcing terms would have varying solutions based on differing fractional and temper orders.

Smooth To form a baseline for the following experiments, we ensured smoothness of the forcing term f in this case by sampling from a Gaussian Process. That is, we have

$$f(t) \sim \mathcal{N}(\mu, \sigma^2(-\Delta + \tau^2 I)^{\gamma}), \quad t \in [0, 1]$$

In particular, the random processes we generated used $\mu = 1, \gamma = 2.5, \tau = 7, \sigma = 7$. In this case, the input to the model was the forcing term $f(t)$ as a vector, the fractional order α, and the temper order σ (for experiment cases that were relevant, i.e. Case 1 did not include the temper order as an input). These were compared to the ground truth $u(t)$ produced by the solver, also encoded as a sequence.

LIF In Cases 1 and 3, the forcing term $I(t)$ was fixed with 3 spiking locations of the same amplitude. In Case 2 where the forcing term was varied, random vectors were generated for location (t) and spiking amplitude ($I(t)$) in order to produce varying forcing terms with different spiking location and intensity. As with the previous case, the input to the model was the forcing term $I(t)$ as a vector, the fractional order α, and the temper order σ where applicable. These were compared to the ground truth $V(t)$.

Diffusion In this experiment, we generated a fixed set of source terms and initial conditions to iterate over and the same process as the previous two experiments was followed. The significant difference arising in the diffusion experiment is that the input and output are now matrices rather than sequences as vectors. In particular, we have the input as a matrix of the source term across both spatial and temporal domains, i.e. $\{f(x_i, t_j\}_{i,j}$, the fractional order, and the temper order. The model output is compared against the ground truth solution $u(x, t)$, which is also encoded as a matrix with dimensions across the time and space axes.

In each of these cases, the input and output functions were discretized as sequences or matrices, where the n column in Table 4.1 is the resolution of the discretization. For example, input forcing terms $f(t)$ were sampled as a sequence of 201 points while the diffusion input $f(x, t)$ was sampled on a 257×257 grid.

Table 4.1: Experiment details and relative L_2 error for each case of the tempered fractional smooth, LIF, and diffusion problems for the two different models tested. The intervals given in the α and σ columns represent the interval over which the parameter was sampled, while the values given in the n columns is the number of points over which forcing terms and solutions were discretized. The loss $\mathcal{L}(\theta)$ reported is the best relative L_2 loss achieved the model in each case.

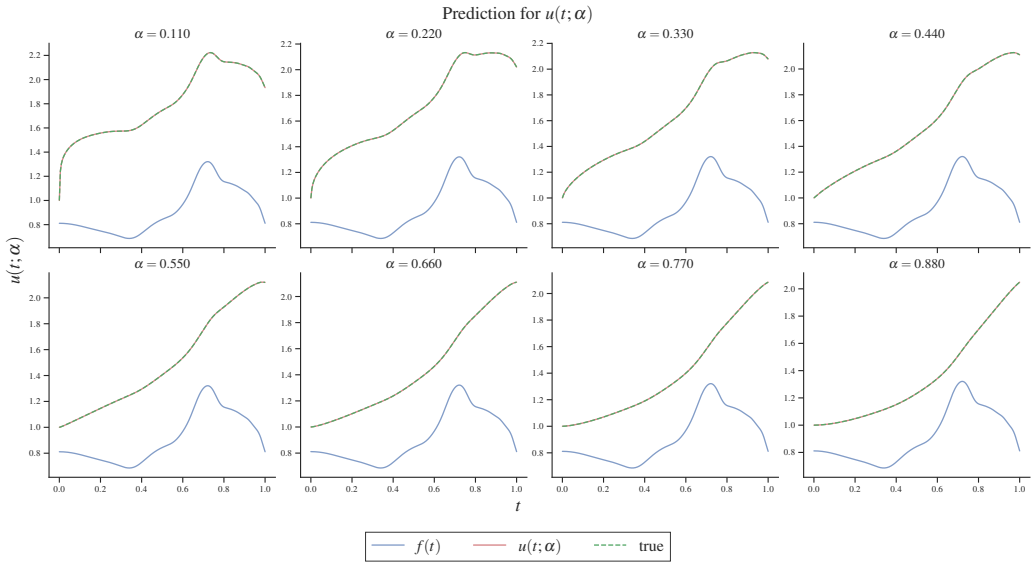
			Experiment settings						Results
Model	Experiment	Case	α	σ	n	Learning Rate	Batch Size	Iterations	$\mathcal{L}(\theta)$
Adam/1cycle	Smooth	1	$[0.11, 0.99]$	–	201	1.0×10^{-3}	32	1.0×10^6	4.7×10^{-4}
		2		–					9.2×10^{-4}
		3		$[0.11, 0.99]$					6.5×10^{-4}
	LIF	1	$[0.11, 0.99]$	–	204	1.0×10^{-4}	32	2.0×10^6	3.9×10^{-4}
		2		–	200				1.3×10^{-2}
		3		$[0.11, 2]$	314				1.3×10^{-3}
	Diffusion	1	$[0.11, 0.99]$	–	257^2	1.0×10^{-4}	32	5.0×10^4	6.1×10^{-3}
Lion/polynomial	Smooth	1	$[0.11, 0.99]$	–	201	1.0×10^{-4}	2000	5.0×10^5	5.5×10^{-5}
		2		–					7.9×10^{-5}
		3		$[0.11, 0.99]$					1.1×10^{-2}
	LIF	1	$[0.11, 0.99]$	–	204	1.0×10^{-4}	1000	2.0×10^5	3.9×10^{-5}
		2		–	200				3.0×10^{-1}
		3		$[0.11, 2]$	314				3.4×10^{-2}
	Diffusion	1	$[0.11, 0.99]$	–	257^2	1.0×10^{-4}	32	5.0×10^4	8.6×10^{-2}

■ 4.5 Results and Discussion

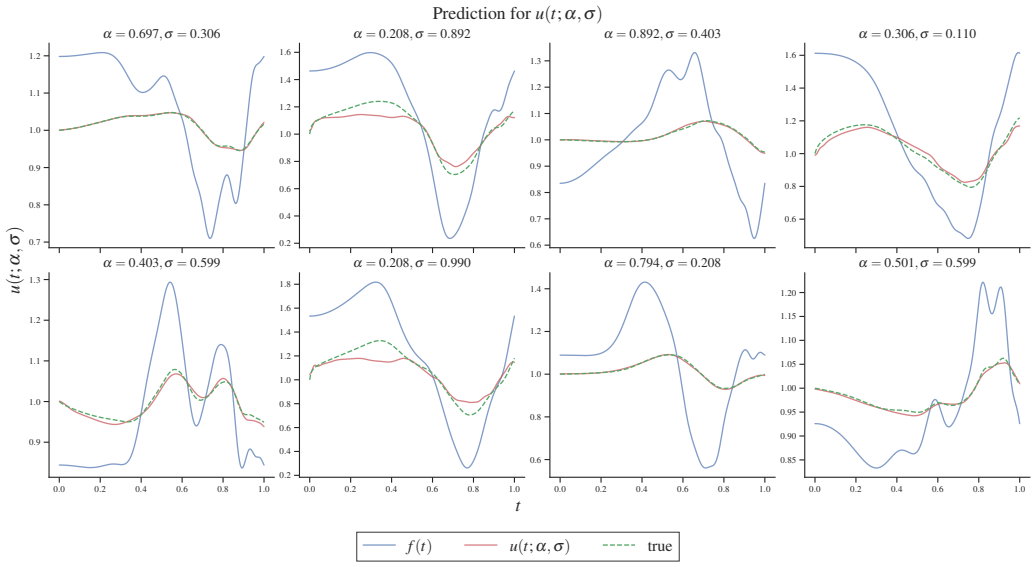
In this section, we report the particular parameter settings for each experiment and discuss the experimental results. All of our models using the Adam/1cycle optimizer and learning rate combination were trained with a batch size of 32. Based on the claims in the Lion work, we used learning rates that were about 5 to $10\times$ smaller in the experiments which employed the Lion optimizer. In addition, independent experiments have shown that better performance can be achieved with Lion on larger batch sizes, hence we used much larger batch sizes for those experiments, except for the diffusion case which required much more memory. For the smooth experiments, n is constant as the time discretization scheme allowed for a constant resolution, but due to the sampling scheme in the LIF experiment, n is not constant across experiments.

4.5.1 Smooth Experiment

We begin our discussion with the baseline result: Case 1 of the smooth system where only the forcing term $f(t)$ and α are varying while $\mu = 0$. Figure 4.1a provides a visualization of the forcing term, model prediction, and true solution for varying α and a fixed f. Evidently, the model (predictions in red, ground truth in green) produces good predictions, even in cases where the fractional order becomes quite small (e.g. $\alpha = 0.11$) and the nonlocality becomes large.



(a) Lion/polynomial predictions for the smooth experiment—Case 1 (varying α, f)

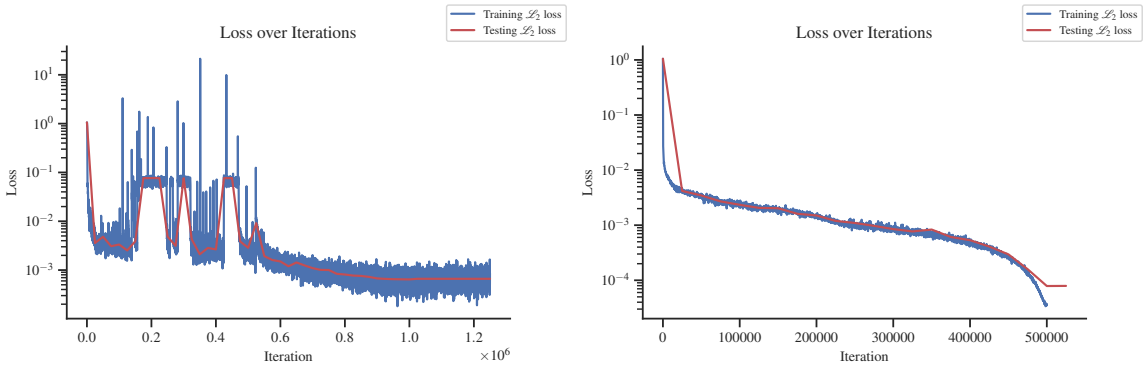


(b) Adam/1cycle predictions for the smooth experiment—Case 3 (varying α, f, σ)

Figure 4.1: Results from the smooth experiment

The model is able to achieve similar results in terms of performance in Case 2 when $\mu = -2$ is introduced to add dependence in the system, which can be seen from the similar L_2 relative errors achieved between Case 1 and Case 2 by the Lion/polynomial model (5.5×10^{-5} and 7.9×10^{-5}, respectively).

However, when we add the tempering into the system (results displayed in Figure 4.1b), there is evidently some failure of the model to converge. Although most predictions seem to be approximately correct, the model prediction is not very close to the true solution in cases where the tempering order is higher. In particular, in the trajectories for which



(a) Adam/1cycle loss curve

(b) Lion/polynomial loss curve

Figure 4.2: Comparison between the loss curves of the two models from Case 2 of the smooth experiments

$\sigma = 0.892$ and $\sigma = 0.99$ in the figure, the model is unable to capture the peaks in the solution. This was a result produced by both models and suggests the failure to capture some complexity in the system.

In terms of comparing the convergence behavior between the two models, we can look to their loss curves which exhibit some interesting differences. First, we note that due to the difference in computational expense of the two models, we trained the model which employed the Lion optimizer with polynomial learning rate decay for approximately half the iterations as the Adam model with 1cycle learning scheduler. Interestingly, it seems that the 1cycle learning rate scheduler allows the Adam model to achieve a decent loss at around 5.0×10^{-3} after a very small number of iterations compared to the Lion model. However, the loss spikes back up multiple times before converging. Notably, the second order optimizer L-BFGS which is employed after one million iterations on the Adam model does not decrease the loss much further than what Adam achieved. On the other hand, the Lion model sees a sharp and monotonically decreasing loss evolution with noticably much less oscillation in the training loss. The optima achieved by this model was also an order better than that achieved by the Adam model.

4.5.2 LIF Experiment

In the LIF experiments, we found that our model was able to converge to accurate solutions despite the difficulty of the sharpness and jumps in the forcing term and solutions. In particular, the Adam model achieved a loss on the same order as the smooth cases for Case 1 of the experiment which varied the fractional order α. However, both models struggled in the second case, which added much more complexity to the problem by varying both the location and the intensity of the spikes in the forcing term $I(t)$ in addition to the fractional order. As evidenced by the model predictions in Figure 4.3, it appears that the model was able to capture the relevant feature for location of the spike; however, the

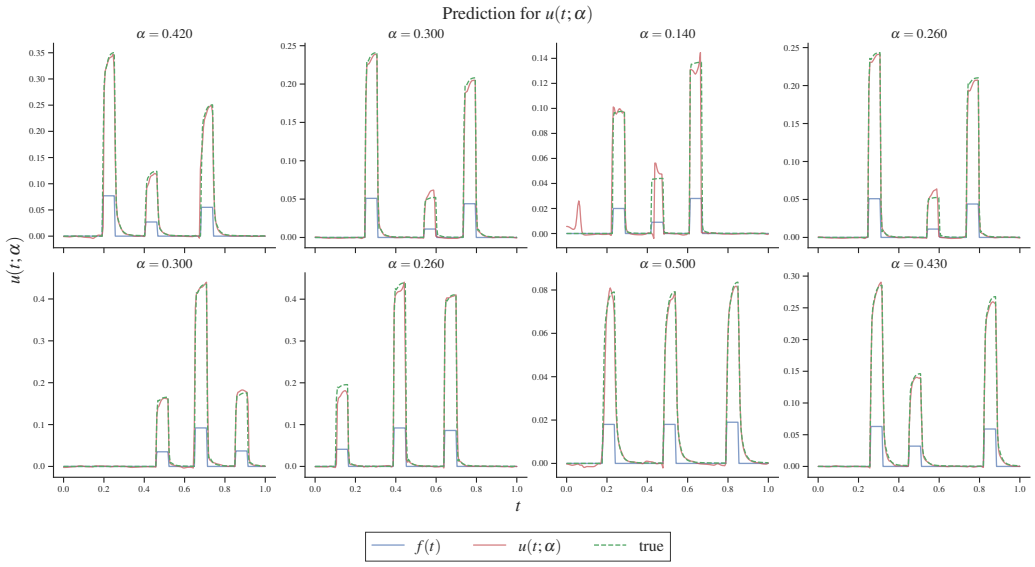


Figure 4.3: Adam/1cycle predictions for the LIF experiment—Case 2 (varying $\alpha, I(t)$)

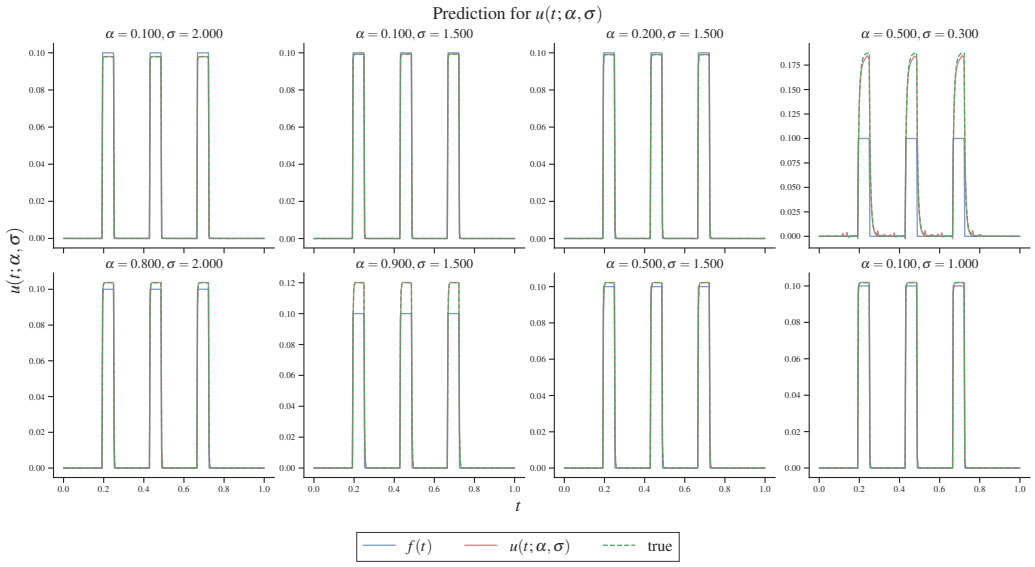


Figure 4.4: Adam/1cycle predictions for the LIF experiment—Case 3 (varying α, σ)

shape and amplitude of the solution membrane potential proved to be a difficulty.

In the final case of the LIF experiments, we tested the model's capability to capture purely the fractional and tempering parameters of the operator, and both models exhibited decent performance. That is, although the models did not achieve the same loss as they did in the smooth cases, this case appeared to be better learned that the previous one which varied the spiking term. In fact, looking at the model predictions in Figure 4.4, it appears that overall the model performs extremely well in prediction, with only one case with slightly unresolved frequency when $\alpha = 0.5$ and $\sigma = 0.3$.

Overall, the results of the LIF experiments are very good considering the difficulty

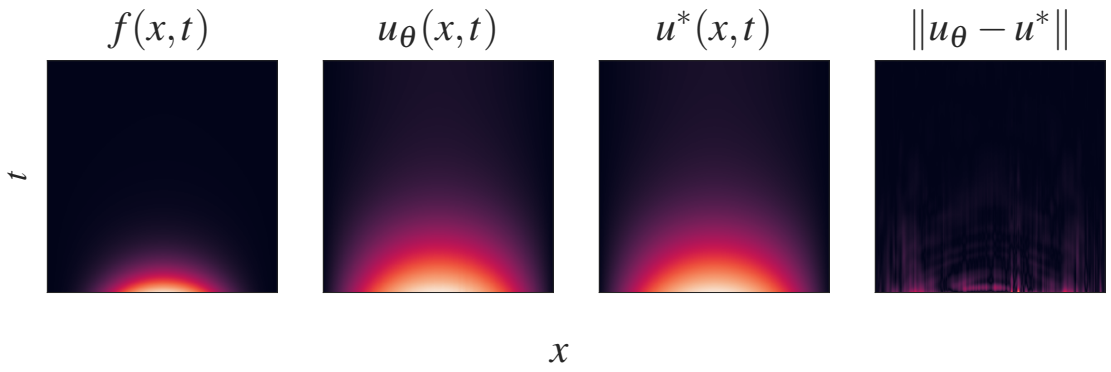


Figure 4.5: Lion/polynomial predictions for the diffusion experiment—Case 1 (varying $\alpha, f(t)$)

that the sharp spiking forcing term induces.

4.5.3 Diffusion Experiment

Finally, results for the diffusion equation are displayed in Figure 4.5. In the figure, the forcing term $f(x,t)$, model prediction $u_\theta(x,t)$, true solution $u^*(x,t)$, and pointwise absolute error $\|u_\theta - u^*\|$ are displayed. Evidently, the model also produces decent results for this system; the prediction and true solutions are visually indistinguishable and there is only some presence of residual error where the source term is the most intense. The model in this case was remarkably expensive to train; 50 thousand iterations took the same time as the million iterations for the smooth and LIF cases with the Adam model. Interestingly, we observe that the Adam/1cycle model is able to achieve a better relative error than the Lion/polynomial model under these computational constraints. Training for more iterations may result in the Lion model overtaking the Adam model in loss due to the nature of the learning rate schedulers.

Overall, from the relative L_2 errors in Table 4.1 and visual performance, the model clearly has very good approximation capability for tempered fractional operators. Each experiment demonstrated that the model is able to exhibit convergence to an optimum of some degree. However, there are still cases which result in better approximations than others, likely a product of the varying complexities in each experiment. From the behavior of the loss curves and learning performance, our experiments suggest that the Lion optimizer does i

Attention Maps To further analyze our model, we produced visualizations of the attention maps learned for each case as a heuristic for model capacity in learning appropriate bases for each operator. An example of such a visualization can be found in Figure 4.6, which displays the attention maps for each layer learned by the encoder in Case 1 of the LIF experiments. We can see that the number of relevant features is indeed increased for

the operators with lower fractional order as we might expect—there is more nonlocality present in the system. For example, in the case of $\alpha = 0.723$ (third from left) and $\alpha = 0.11$ (fourth from left, we can see that in the second attention layer, there is an entire section of features that are not relevant (blue) in the $\alpha = 0.723$ case while this same section of features is highlighted (white and red) in the $\alpha = 0.11$ case.

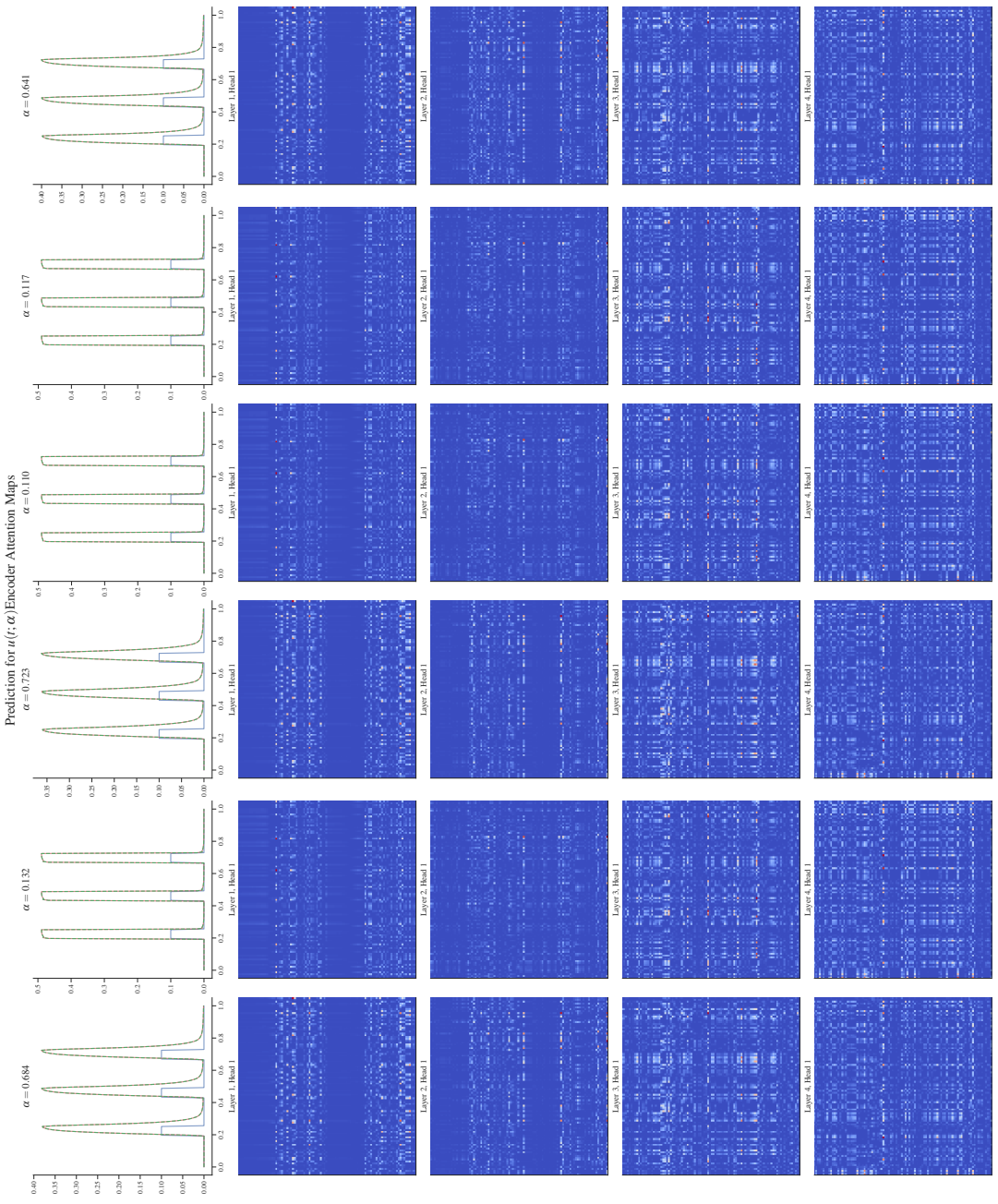


Figure 4.6: Attention maps learned by the Adam/1cycle encoder for the LIF experiments in Case 1

SECTION

5 CONCLUSION

Machine learning has revolutionized a huge spectrum of applications, and now poses itself as a significant frontier for expansion and development in the computational domain. In this work, we have shown the power of the attention mechanism in application to the operator learning problem. In particular, we studied the approximation capabilities of such models on the tempered fractional Caputo derivative operator. This was done across three systems of varying complexity: the tempered fractional initial value problem with a smooth forcing term, then the fractional LIF neuron model, and finally, the tempered fractional diffusion equation. Our experiments demonstrate the remarkable learning capacity of the attention model employed; while the vanilla DeepONet failed to learn these systems, the attention model was able to converge and exhibit excellent accuracy in each system.

■ 5.1 Future Work

In terms of building on this work, the next steps would be to run ablation experiments on various factors. For example, the discretization resolution on the inputs generated by the solvers as well as the depth of the encoders/decoders. Since the accuracy of the model is realistically bounded by the resolution of the input, we would be interested in seeing whether generating finer inputs allow for lower asymptotic error or if the models have reached optimal performance. Additionally, with respect to the couple cases of experiments where the model fails to completely converge (e.g. Case 3 of the smooth experiments and Case 2 of the LIF experiments), adding more complexity such as deepening the architecture to try to resolve these issues would be be of interest as well.

Additionally, as we previously discussed, the vanilla DeepONet was unable to learn the problems proposed in this work. However, a recent development proposes a new method of training the DeepONet that has been shown to have the capability to learn jump discontinuities in Riemann problems [29]. The core of this development is the *two-step* training of the DeepONet proposed in [15]. This is composed of first training the

trunk network according to a loss function

$$\mathcal{L}(\theta_t) = \|T(\theta_t)A - U\|$$

where $T(\theta)$ is the representation of the trunk network with parameters θ_t and A being a trainable matrix. Here U represents matrix of true solutions. After this, the matrix $T(\theta^*)$ is decomposed by Gram-Schmidt orthonormalization so that

$$Q^*R^* = \text{qr}(T(\theta^*))$$

where θ^* is the optimal learned parameter. After this, the branch network is trained to fit R^*A^*; that is, the following loss is minimized:

$$\mathcal{L}(\theta_b) = \|B(\theta_b) - R^*A^*\|$$

where θ_b are the branch network parameters. Of course, both of these losses should have some appropriate norm chosen for the problem.

We are currently interested in comparing the learning capacities of the model in our paper against that of the 2-step DeepONet as well as other neural operators, such as the Diffusion-inspired Temporal Transformer Operator (DiTTO) [26].

References

[1] Anandkumar, A., Azizzadenesheli, K., Bhattacharya, K., Kovachki, N., Li, Z., Liu, B., and Stuart, A. (2019). Neural operator: Graph kernel network for partial differential equations. In *ICLR 2020 Workshop on Integration of Deep Neural Models and Differential Equations*.

[2] Cao, S. (2021). Choose a transformer: Fourier or Galerkin. In *Advances in Neural Information Processing Systems (NeurIPS 2021)*, volume 34.

[3] Chen, T. and Chen, H. (1993). Approximations of continuous functionals by neural networks with application to dynamic systems. *IEEE Transactions on Neural Networks*, 4(6):910–918.

[4] Chen, T. and Chen, H. (1995). Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems. *IEEE Transactions on Neural Networks*, 6(4):911–917.

[5] Chen, X., Liang, C., Huang, D., Real, E., Wang, K., Liu, Y., Pham, H., Dong, X., Luong, T., Hsieh, C.-J., Lu, Y., and Le, Q. V. (2023). Symbolic discovery of optimization algorithms.

[6] Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2019). Bert: Pre-training of deep bidirectional transformers for language understanding. In *North American Chapter of the Association for Computational Linguistics*.

[7] Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., and Houlsby, N. (2021). An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*.

[8] Goswami, S., Yin, M., Yu, Y., and Karniadakis, G. E. (2022). A physics-informed variational deeponet for predicting crack path in quasi-brittle materials. *Computer Methods in Applied Mechanics and Engineering*, 391:114587.

[9] Guo, L., Zeng, F., Turner, I., Burrage, K., and Karniadakis, G. E. (2019). Efficient multistep methods for tempered fractional calculus: Algorithms and simulations. *SIAM Journal on Scientific Computing*, 41(4):A2510–A2535.

[10] Jagtap, A. D., Kawaguchi, K., and Em Karniadakis, G. (2020). Locally adaptive activation functions with slope recovery for deep and physics-informed neural networks. *Proceedings of the Royal Society A*, 476(2239):20200334.

[11] Kingma, D. P. and Ba, J. (2015). Adam: A method for stochastic optimization. In Bengio, Y. and LeCun, Y., editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

[12] Kovachki, N. B., Li, Z., Liu, B., Azizzadenesheli, K., Bhattacharya, K., Stuart, A. M., and Anandkumar, A. (2021). Neural operator: Learning maps between function spaces. *CoRR*, abs/2108.08481.

[13] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In Pereira, F., Burges, C., Bottou, L., and Weinberger, K., editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc.

[14] Kumar, D., Singh, J., and Kumar, S. (2015). A fractional model of navier–stokes equation arising in unsteady flow of a viscous fluid. *Journal of the Association of Arab Universities for Basic and Applied Sciences*, 17:14–19.

[15] Lee, S. and Shin, Y. (2023). On the training and generalization of deep operator networks. *arXiv e-prints*, page arXiv:2309.01020.

[16] Leibniz, G. W. (1962). "letter from hanover, germany to g.f.a. l'hospital, september 30, 1695". *Leibniz Mathematische Schriften*.

[17] Li, J., Li, D., Xiong, C., and Hoi, S. (2022). Blip: Bootstrapping language-image pre-training for unified vision-language understanding and generation. In *ICML*.

[18] Li, Z., Kovachki, N., Azizzadenesheli, K., Liu, B., Bhattacharya, K., Stuart, A., and Anandkumar, A. (2020). Fourier neural operator for parametric partial differential equations.

[19] Li, Z., Meidani, K., and Farimani, A. B. (2023). Transformer for partial differential equations' operator learning. *Transactions on Machine Learning Research*.

[20] Liu, D. C. and Nocedal, J. (1989). On the limited memory bfgs method for large scale optimization. *Mathematical Programming*, 45(1):503–528.

[21] Lu, L., Jin, P., Pang, G., Zhang, Z., and Karniadakis, G. E. (2021). Learning nonlinear operators via deeponet based on the universal approximation theorem of operators. *Nature Machine Intelligence*, 3(3):218–229.

[22] Lu, L., Meng, X., Cai, S., Mao, Z., Goswami, S., Zhang, Z., and Karniadakis, G. E. (2022). A comprehensive and fair comparison of two neural operators (with practical extensions) based on fair data. *Computer Methods in Applied Mechanics and Engineering*, 393:114778.

[23] Mabrouk, A., Fouda, M. E., and Eltawil, A. (2022). On numerical approximations of fractional-order spiking neuron models. *Communications in Nonlinear Science and Numerical Simulation*, 105:106078.

[24] Medveď, M. and Brestovanská, E. (2021). Differential equations with tempered phi-caputo fractional derivative. *Mathematical Modelling and Analysis*, 26:631–650.

[25] Nguyen, T. M., Pham, M., Nguyen, T. M., Nguyen, K., Osher, S., and Ho, N. (2022). Fourierformer: Transformer meets generalized fourier integral theorem. In Oh, A. H., Agarwal, A., Belgrave, D., and Cho, K., editors, *Advances in Neural Information Processing Systems*.

[26] Ovadia, O., Turkel, E., Kahana, A., and Karniadakis, G. E. (2023). Real-time inference and extrapolation via a diffusion-inspired temporal transformer operator (ditto).

[27] Pang, G., Lu, L., and Karniadakis, G. E. (2019). fpinns: Fractional physics-informed neural networks. *SIAM Journal on Scientific Computing*, 41(4):A2603–A2626.

[28] Patel, R., Trask, N., Wood, M., and Cyr, E. (2021). A physics-informed operator regression framework for extracting data-driven continuum models. *Computer Methods in Applied Mechanics and Engineering*, 373:113500.

[29] Peyvan, A., Oommen, V., Jagtap, A., and Karniadakis, G. (2024). Riemannonets: Interpretable neural operators for riemann problems.

[30] Pinkus, A. (1999). Approximation theory of the mlp model in neural networks. *Acta Numerica*, 8:143 – 195.

[31] Pirozzi, E. (2024). Some fractional stochastic models for neuronal activity with different time-scales and correlated inputs. *Fractal and Fractional*, 8(1).

[32] Radford, A. and Narasimhan, K. (2018). Improving language understanding by generative pre-training.

[33] Raissi, M., Perdikaris, P., and Karniadakis, G. E. (2019). Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707.

[34] Sabzikar, F., Meerschaert, M., and Chen, J. (2014). Tempered fractional calculus. *Journal of Computational Physics*, 293.

[35] Shin, Y., Darbon, J., and Karniadakis, G. E. (2023). Accelerating gradient descent and adam via fractional gradients. *Neural Networks*, 161:185–201.

[36] Shukla, K., Jagtap, A. D., and Karniadakis, G. E. (2021). Parallel physics-informed neural networks via domain decomposition. *Journal of Computational Physics*, 447:110683.

[37] Smith, L. N. and Topin, N. (2018). Super-convergence: Very fast training of residual networks using large learning rates.

[38] Teka, W. W., Upadhyay, R. K., and Mondal, A. (2017). Fractional-order leaky integrate-and-fire model with long-term memory and power law dynamics. *Neural Networks*, 93:110–125.

[39] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 6000–6010, Red Hook, NY, USA. Curran Associates Inc.

[40] Wen, G., Li, Z., Azizzadenesheli, K., Anandkumar, A., and Benson, S. M. (2022). U-fno–an enhanced fourier neural operator-based deep-learning model for multiphase flow. *Advances in Water Resources*, page 104180.

[41] Yang, L., Meng, X., and Karniadakis, G. E. (2021). B-pinns: Bayesian physics-informal neural networks for forward and inverse pde problems with noisy data. *Journal of Computational Physics*, 425.

[42] Yang, Z. and Zeng, F. (2023). A corrected l1 method for a time-fractional subdiffusion equation. *Journal of Scientific Computing*, 95(3):85.

[43] Zeng, F. and Li, C. (2014). Fractional differential matrices with applications.

[44] Zhang, H., Liu, F., Turner, I., and Yang, Q. (2016). Numerical solution of the time fractional black–scholes model governing european options. *Computers & Mathematics with Applications*, 71(9):1772–1783.