Georgia State University
ScholarWorks @ Georgia State University

Computer Science Theses | Department of Computer Science

12-13-2023

DynaLay: An Introspective Approach to Dynamic Layer Selection for Deep Networks

Mrinal Mathur

Follow this and additional works at: https://scholarworks.gsu.edu/cs_theses

Recommended Citation
Mathur, Mrinal, "DynaLay: An Introspective Approach to Dynamic Layer Selection for Deep Networks." Thesis, Georgia State University, 2023.
doi: https://doi.org/10.57709/36398139

This Thesis is brought to you for free and open access by the Department of Computer Science at ScholarWorks @ Georgia State University. It has been accepted for inclusion in Computer Science Theses by an authorized administrator of ScholarWorks @ Georgia State University. For more information, please contact scholarworks@gsu.edu.

DynaLay: An Introspective Approach to Dynamic Layer Selection for Deep Networks

by

Mrinal Mathur

Under the Direction of Sergey Plis Ph.D.

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of

Master of Science

in the College of Arts and Sciences

Georgia State University

2023

ABSTRACT

Deep learning models have increasingly become computationally intensive, necessitating specialized hardware and significant runtime for both training and inference. In this work, we introduce DynaLay, a versatile and dynamic neural network architecture that employs a reinforcement learning agent to adaptively select which layers to execute for a given input. Our approach introduces an element of introspection into neural network architectures by enabling the model to recompute the results on more difficult inputs during inference, balancing the amount of expelled computation, optimizing for both performance and efficiency. The system comprises a main model constructed with Fixed-Point Iterative (FPI) layers, which can approximate complex functions with high fidelity, and an agent that chooses among these layers or a no-operation (NOP) action. Unique to our approach is a multi-faceted reward function that combines classification accuracy, computational time, and a penalty for redundant layer selection, thereby ensuring a harmonious trade-off between performance and cost. Experimental results demonstrate that DynaLay achieves comparable accuracy to conventional deep models while significantly reducing computational overhead. Our approach represents a significant step toward creating more efficient, adaptable, and universally applicable deep learning systems

INDEX WORDS: Foundational Model, Dynamic Neural Networks, Fixed-Point Iteration, Computational Efficiency, Multi-Task Learning, Model-Agnostic Framework, Implicit Differentiation, Backpropagation, Agent-Based Modeling

Copyright by
Mrinal Mathur
2023

DynaLay: An Introspective Approach to Dynamic Layer Selection for Deep Networks

by

Mrinal Mathur

Committee Chair: Sergey Plis

Committee: Armin Iraji

Electronic Version Approved:

Office of Graduate Services

College of Arts and Sciences

Georgia State University

December 2023

DEDICATION

I wholeheartedly dedicate this thesis to my parents, who have been the pillars of strength and support throughout my life. Their relentless sacrifices and enduring love have not only made my education possible but have also instilled in me the resilience and curiosity to explore the uncharted territories of deep learning and computational sciences. It was their tireless efforts that provided me with the opportunities they never had, and for that, I am eternally grateful. They gave what little they had to offer me a wealth of opportunity, and it is their spirit that has fueled my academic journey. Moreover, I would like to express my profound gratitude to my advisor and supervisor, Dr. Sergey Plis. His guidance, encouragement, and astute insights have been invaluable in shaping both this research project and my broader academic career. Dr. Plis has been more than just a supervisor to me; he has been a mentor in the truest sense of the word. His willingness to assist in any way possible, and his enthusiasm for the subject matter, has made the process of conducting this research not just educational but also deeply fulfilling. His commitment to academic excellence and his knack for asking the right questions have continually pushed me to reach greater heights. I am incredibly fortunate to have had such a dedicated and insightful advisor to guide me through this intricate and challenging field. This thesis is a culmination of the lessons learned, the challenges overcome, and the countless hours of research and experimentation. It is a testament to the enduring love and sacrifices of my parents and the insightful guidance of Dr. Plis. To them, I dedicate this body of work.

ACKNOWLEDGMENTS

I would like to take this opportunity to express my deepest gratitude to all those who have been instrumental in the realization of this thesis. Your contributions have not only facilitated my research but have also enriched my academic journey in immeasurable ways. At the forefront of this acknowledgment is my esteemed advisor, Dr. Sergey Plis, whose intellectual mentorship has been invaluable to me. His profound expertise in the field and his willingness to share it have greatly shaped this work. Dr. Plis has been an endless source of guidance, constructive criticism, and encouragement. His wisdom has illuminated the complexities of the subject matter and his unwavering support has made this daunting task seem surmountable. I am eternally grateful for his contributions to both this thesis and my academic growth. I extend my sincere thanks to the members of my thesis committee, particularly Dr. Armin Iraji, whose incisive comments and thoughtful suggestions have been crucial in refining my research methodologies and strengthening the overall quality of this thesis. Your expertise and attention to detail have significantly impacted this work, and for that, I am grateful. To my family and friends, whose unflagging love and encouragement have been my pillars of strength throughout this journey—I owe you a debt of gratitude. Your unwavering belief in my abilities, even when I doubted myself, provided the emotional sustenance that fueled this project. Your support has been both a comfort and a catalyst, pushing me to reach new academic heights. I would also like to extend my heartfelt thanks to all the participants who graciously volunteered their time, expertise, and experiences for

this study. Your contributions have been indispensable in the realization of this research, providing the real-world validation that is crucial for academic work of this nature. This thesis would not have been possible without the collective support, wisdom, and encouragement of each individual mentioned above. My heartfelt thanks go out to each one of you for making this scholarly work a reality.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1
INTRODUCTION

1.1 Background and Motivation

The rapid advancements in deep learning have led to transformative changes across various disciplines, including natural language processing, computer vision, and reinforcement learning Schmidhuber (2015)Vaswani et al. (2023)Hochreiter (1995). Despite these monumental strides, we are confronted with persistent challenges related to computational inefficiencies and architectural rigidity of traditional deep learning models. These challenges are especially pronounced in resource-restricted environments, which demand a more nuanced, context-sensitive approach.

Our work introduces an introspective architecture that aims to bridge this gap. We employ a reinforcement learning agent to dynamically and adaptively select layers to execute in a deep neural network. This architecture imbues the model with the capability to self-optimize in real-time, providing a fine balance between computational efficiency and classification accuracy. The introspective architecture is particularly beneficial for applications where computational resources are at a premium, yet high performance is non-negotiable.

The backbone of our architecture features Fixed-Point Iterative (FPI) layers. These layers are known for their prowess in approximating complex functions while being computationally efficient Arya et al. (2023)Kidger (2022). The role of the reinforcement learning agent is to judiciously choose among these FPI layers, or alternatively, to select a no-operation (NOP) action based on the characteristics of the given input. One of the most critical aspects of

our work is the introduction of a nuanced reward function. This function considers a blend of metrics—classification accuracy, computational time, and a penalty for redundant layer selection. This multifaceted approach ensures that the agent makes context-aware decisions, thereby optimizing both performance and computational resources.

1.2 Research Objectives and Contributions

The thesis is focused on achieving the following objectives:

- Development and empirical validation of an introspective architecture, combining a reinforcement learning agent with FPI layers for dynamic layer selection.
- Introduction and analysis of a nuanced reward function designed to balance classification accuracy, computational efficiency, and decision-making diversity.
- Extensive experiments and comparative studies to validate the effectiveness of our approach against traditional deep learning models.

Our research culminates in a novel, introspective architecture that not only achieves comparable performance to conventional models but does so with significantly reduced computational overhead.

1.3 Thesis Outline

The remainder of this thesis is structured as follows: Chapter 2 provides the necessary background on FPI layers and reinforcement learning, especially in the context of deep learning. Chapter 3 elaborates on the proposed introspective architecture, detailing its

components and operational mechanisms. Chapter 4 covers the experimental design, setup, and results, while Chapter 5 concludes the thesis by summarizing the contributions and suggesting avenues for future research.

CHAPTER 2
RELATED WORK

In the ever-evolving landscape of machine learning and artificial intelligence, a multitude of methods has emerged, aiming to enhance the performance, scalability, and efficiency of deep learning models. This chapter aims to provide a comprehensive review of the literature that is pertinent to the themes presented in this thesis. We delineate the seminal works, recent advancements, and the intersectionalities that contextualize our contributions in a broader academic discourse.

2.1 Fixed-Point Iteration in Deep Neural Networks

Fixed-Point Iteration (FPI) has its origins in numerical analysis and has evolved into a versatile and powerful tool in the field of deep learning. Researchers have applied FPI in various applications, including stabilizing training algorithms, enhancing convergence rates, and incorporating structured prior knowledge into neural networks. FPI has been used in differentiable forward and backward fixed-point iteration layers, which facilitate the use of more complex operations in deep networks **Arjovsky et al**. employed FPI in the context of Wasserstein GANs to calculate Wasserstein distances, stabilizing the gradient landscape during adversarial training. However, such works primarily utilize FPI as an auxiliary component for specialized tasks or challenges, and they often don't explore its full potential in neural architecture design. Our work distinguishes itself by embedding FPI layers as a central element within the neural network architecture. These layers are not merely supplementary but foundational, designed to approximate complex functions with fewer parameters. Fur-

thermore, we extend the utility of FPI layers by incorporating a reinforcement learning agent that adaptively selects which layers to execute based on the incoming data. This approach represents a significant leap in making neural networks more versatile, efficient, and context-aware. In addition to the aforementioned applications, FPI has been used in various other contexts, such as automating the FPI inspection process using machine learning and overcoming challenges in fixed-point training of deep convolutional networks. However, these works focus on specific applications and do not fully explore the potential of FPI in the broader context of deep learning. By embedding FPI layers as a central component in neural network architectures, we aim to unlock the full potential of FPI in deep learning. This approach could lead to more efficient and versatile neural networks, capable of handling complex tasks and adapting to various contexts.

2.2 The Advent of Adaptive Networks

The concept of adaptivity in neural networks has been a subject of keen interest, particularly in the optimization of resource allocation during inference. Adaptive neural networks can adjust their structure and parameters based on data inputs, allowing them to respond immediately to changes in the data input and adapt to changes in the environment. This adaptability is crucial for improving the accuracy of pattern recognition and prediction.

Existing methods for adaptivity in neural networks usually involve architecture adjustments in a static manner, either during pre-training or the training phase. While these methods are capable of architecture adjustments, they do not enable real-time, introspective

adjustments during inference, which could optimize both performance and computational efficiency. Our approach takes adaptivity a step further by enabling real-time, introspective adjustments during inference, optimizing for both performance and computational efficiency. Introspective agents can recognize the extent to which their internal perceptual experiences deviate from the actual states of the external world. By incorporating introspective inference into neural networks, we can create systems that are more aware of their own limitations and can adapt accordingly.

In our approach, we incorporate a reinforcement learning agent that adaptively selects which layers to execute based on the incoming data. This allows the neural network to make real-time adjustments during inference, optimizing resource allocation and improving overall performance. By enabling real-time, introspective adjustments, our approach represents a significant leap in making neural networks more versatile, efficient, and context-aware.

2.3 Temporal Dynamics: Pondering Networks and Deep Equilibrium Networks

2.3.1 Adaptive Computation in Neural Networks: Pondering Networks and Deep Equilibrium Networks

The question of adaptive computation—how much computational resource should be allocated to each input—has been the focus of several significant works, notably Pondering Networks **??** and Deep Equilibrium Networks (DEN) **?**.

Pondering Networks introduce a unique approach to adaptive computation by allowing the network to allocate different amounts of processing time for each input. In essence, the model 'ponders' over more complex inputs and rushes through simpler ones, thereby

achieving a form of computational efficiency. However, this adaptability is largely temporal; it doesn't offer a mechanism for adapting the network's architecture or selecting specific layers based on the input's characteristics.

Deep Equilibrium Networks (DEN), on the other hand, involve a more complex iterative solving mechanism at each layer of the network. By doing so, they aim to capture intricate dependencies between inputs and outputs, enhancing the model's performance on complex tasks. However, similar to Pondering Networks, DEN does not inherently provide layer-wise adaptivity. The model adjusts the length of computations but does not dynamically alter its architecture based on the input.

While both Pondering Networks and DEN offer intriguing methodologies for adaptive computation, they fall short in a few key areas. First, they do not offer layer-wise adaptivity, meaning they can't dynamically select or skip specific layers based on the input. Second, they are not designed to perform the kinds of complex operations that FPI layers are capable of, limiting their applicability in scenarios that require such operations for optimal performance.

Our work stands apart by addressing these limitations and providing a more comprehensive solution. Our architecture not only allows for adaptive computation but goes a step further by introducing layer-wise adaptivity through a reinforcement learning agent. This agent can dynamically choose which layers to execute, adding a new dimension of adaptability. Moreover, our architecture incorporates Fixed-Point Iterative (FPI) layers, enabling the network to perform complex operations that are otherwise computationally expensive or infeasible.

By combining the strengths of FPI layers and reinforcement learning, our architecture achieves a harmonious balance between computational efficiency and performance, making it a robust, versatile solution for a wide array of deep learning applications. It opens up new avenues for research into how neural networks can be made more adaptable and efficient, not just in terms of computational time but also in architectural complexity and functional capability.

2.4 Neuronal Diversity: A New Frontier

The notion of neuronal diversity stems from the idea that different neurons or nodes within a network can specialize in various functions, akin to biological neural networks. This concept has been particularly useful in applications that require the network to learn complex, non-linear mappings, such as physical simulations, financial modeling, and multi-modal data interpretation **??**.

In the realm of physics, the introduction of neuronal diversity has led to machine learning models capable of understanding complex physical laws and phenomena with higher accuracy and lesser computational requirements **?**. This offers exciting possibilities for real-world applications where computational efficiency and model performance are critical.

While neuronal diversity as a concept does not directly intersect with our current focus on fixed-point iterations and adaptivity through reinforcement learning, it offers an intriguing avenue for future research. One can envision a model that not only adaptively selects which layers to activate but also which types of neurons within those layers would be most effective

for a given input. This would add another layer of adaptability and specialization to our architecture, making it even more efficient and capable.

Integrating neuronal diversity into our architecture could also open the door to tackling more complex problems that require a nuanced understanding of various features within the data. It could be particularly useful in scenarios where the data is multi-modal or where different types of computations are better suited for different aspects of the data. However, integrating neuronal diversity into an already complex architecture would not be without its challenges. It could introduce additional hyperparameters to tune and complicate the training process. Moreover, the increase in complexity would need to be justified by a commensurate improvement in model performance and efficiency to warrant the additional computational burden.

2.5 Differentiable Optimization Layers: Prior Knowledge Meets Differentiation

Embedding optimization problems as layers within a neural network provides a novel way to incorporate prior knowledge or constraints into the learning process **?**. The general idea is to frame a layer in the neural network as an optimization problem, often formulated as a convex or non-convex optimization, that the network solves during the forward and backward passes. This approach has similarities with our work in that both aim to make neural networks more flexible and capable by incorporating more complex operations. However, there are key differences. Our work focuses on Fixed-Point Iteration (FPI) layers, which allow for computationally efficient approximation of complex functions. In contrast, optimization

layers often involve solving complex problems that may require iterative methods and can thus be computationally expensive.

Moreover, while optimization layers are often problem-specific and need to be designed with the task at hand in mind, FPI layers are more general-purpose. They can be embedded into a wide range of neural network architectures without requiring substantial modifications, making them a more versatile option for many applications. However, such integration would come with its own set of challenges, including but not limited to computational complexity and the risk of overfitting. The use of optimization layers often involves additional complexities such as the need for hyperparameter tuning specific to the optimization problem and the potential for local minima during training. These challenges highlight the advantages of our approach, which uses FPI to achieve similar ends but in a more computationally efficient and straightforward manner.

Our work sits at the confluence of several evolving themes in deep learning—fixed-point iterations, adaptivity, computational efficiency, and model introspection. By integrating a reinforcement learning agent with FPI layers, we not only contribute a novel architecture but also propose a multi-faceted reward function that ushers in a new paradigm for balancing performance against computational cost. This positions our work as a significant step forward in the quest for more efficient, versatile, and intelligent deep learning systems.

CHAPTER 3
METHOD

Our approach is based on simple recurrent backpropagation, which we extend to incorporate a more complex and flexible architecture using implicit differentiation through convergent loops. Our goal is to train deep learning models that perform a fixed number of computations for each input regardless of its difficulty. Unlike other papers that find fixed points on the same number of layers for every computation, we aim to build an efficient architecture that first determines which layer will require a fixed point depending on the input before starting computation.

3.1 Background

3.1.1 Fixed-Point Iterative Layers in Deep Learning

Fixed-point iterative (FPI) Bolte et al. (2022b)Werbos (1990) methods are a class of techniques commonly used to solve equations of the form $f(x) = x$, where x is a fixed point Habiba & Pearlmutter (2021) of the function f. In deep learning, FPI layers are designed to find a fixed point z that satisfies:$z = f(z, x)$. Here, f is the neural network layer's function, x is the input to the layer, and z is the fixed-point that we want to compute. The general iterative scheme for finding this fixed point can be expressed as: $z^{(k+1)} = f(z^{(k)}, x)$ where $z^{(k)}$ is the estimate at iteration k. The process begins with an initial guess $z^{(0)}$ and iterates until convergence, i.e., $||z^{(k+1)} - z^{(k)}|| < \epsilon$ where ϵ is a small tolerance.

we also define $\hat{x}$ which is the fixed point of g, which satisfies $\hat{x} = g(\hat{x})$, where $\hat{x}$ is the

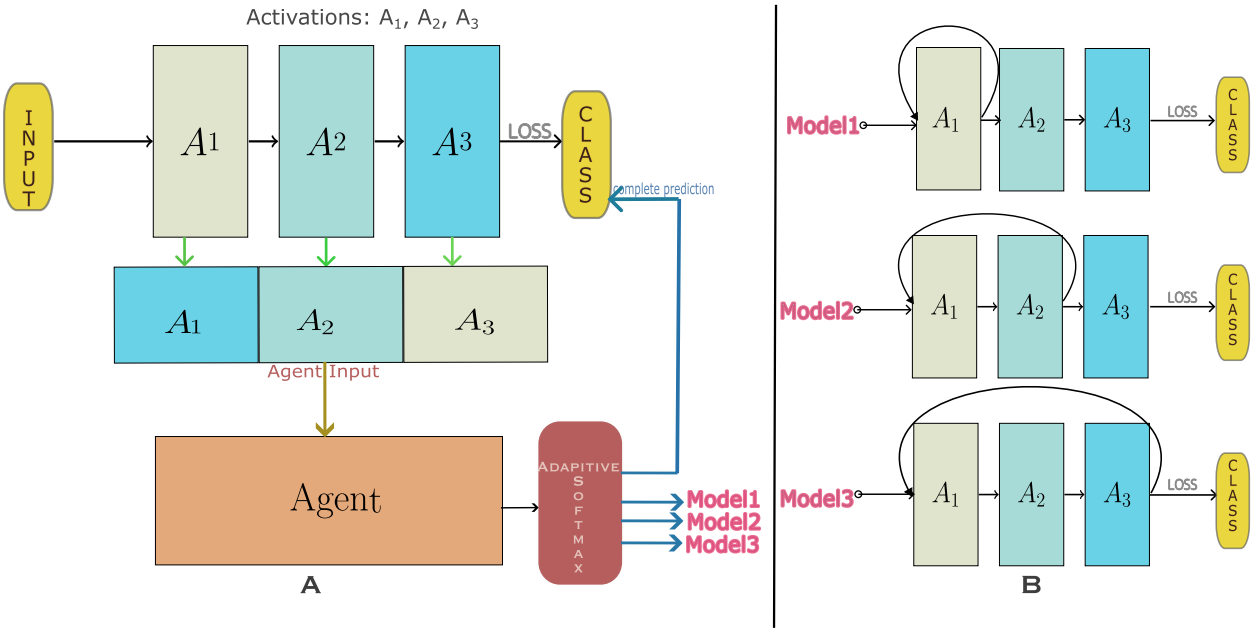

Figure 3.1 Main Model Architecture: A) Schematic representation of the DynaLay architecture, highlighting its key components and the flow of data.B) The figure illustrates how the agent dynamically selects layers for Fixed-Point Iteration, where A_i for the same i are parameter tied Chatziafratis et al. (2022)Habiba & Pearlmutter (2021)

converged vector of g function. Few important concepts to define fixed point iterations involves Contrastive MappingBolukbasi et al. (2017)MacKay et al. (2018).

Theorem 1 (Contraction mappings for fixed point operation)**.** *Consider a function* $f : \mathcal{X} \to \mathcal{X}$ *where* $\mathcal{X}$ *is a complete metric space. Let* $g(x, \theta, l)$ *be the output when using layer* l *in our neural network with input* x *and parameters* θ*. Assume that* f *is a contraction mapping on* $\mathcal{X}$*, i.e., there exists a* $\beta \in [0, 1)$ *such that:* $d(f(x), f(y)) \leq \beta \cdot d(x, y), \quad \forall x, y \in \mathcal{X}$

where $d(\cdot, \cdot)$ *is a distance metric on* $\mathcal{X}$*.*

Under these conditions, our adaptive layer selection mechanism Jeon et al. (2020)Rajeswaran et al. (2019) that minimizes $||f(x,\theta) - g(x,\theta,l)||_2^2$ *will converge to a unique fixed point* x^* *in* $\mathcal{X}$*, such that:*

$$x^* = f(x^*)$$

It is especially valuable since we relate β *to the chosen layer, which makes the contraction faster based on layer attributes.*

We define the computational cost of our model as the sum of time take for each layer computation after applying fixed point iteration on it as $C(n,t) = \sum_{i=1}^{n} t_i$, where n: Number of layers in the neural network, t_i: Time taken for computation at the i^{th} layer and $C(n,t)$: Computational cost of our model, defined as the sum of time taken for each layer computation. Our model introduces a layer-selection mechanism to achieve the aforementioned objective where, for each input, the model employs an agent that uses the current activation's and the inherent characteristics of the data to selectively choose a subset of layers S from the available layers n. this can be represented as: $C(S,t) \leq C(n,t) < C_0$. The computational graph then proceeds only through the layers in S, thereby avoiding the computational cost of the unselected layers. The agent responsible for layer selection is trained to make these decisions by optimizing a trade-off between computational cost and task performance. This enables the model to make choices that are not just computationally efficient but also effective in terms of the task at hand.

We empirically validate the computational efficiency of our model by measuring $C(n,t)$ across multiple data points. Our results indicate that, on average, $C(n,t)$ is consistently less

than C_0, thereby confirming the computational advantages of our approach.

3.1.2 Benefits of Avoiding Solvers and Adherence to Contraction Mapping

One of the unique features of our approach in **DynaLay** is the deliberate avoidance of explicit fixed-point solvers , such as those based on the Banach fixed-point theorem Agarwal et al. (2018), commonly employed in methods like Deep Equilibrium Models (DEQs) Bai et al. (2019). Iterative solvers can introduce numerical instabilities, particularly for ill-conditioned problems. Our architecture, by adhering to contraction mapping principles as mentioned in Theorum 1, guarantees stability. Formally, if $T : X \to X$ is a contraction mapping, then for every $x, y \in X$, $d(T(x), T(y)) \leq k \times d(x, y)$ where $0 < k < 1$, ensuring stability. The absence of a solver simplifies both the architecture and the training algorithm, making the model easier to implement and tune. Our method is rooted in the principles of contraction mapping, providing a robust theoretical foundation that guarantees both the existence and uniqueness of a fixed-point solution without necessitating an explicit solver as mentioned in Equation 3.1:

$$\text{If } T : X \to X \text{ is a contraction, then } \exists! x^* \in X : T(x^*) = x^* \tag{3.1}$$

Our layer selection mechanism is designed to synergize with Theorem 1 adhering to its principles to ensure the model's stability and convergence. This obviates the need for a solver while still offering robust theoretical guarantees as explained in next section.

3.1.3 Layer Selection Mechanism

In traditional architectures and equilibrium networksBai et al. (2019), the network essentially learns a single fixed point. It may not always adapt well to the diverse computational needs of different inputs. Bai et al. (2019) assumes that all layers have the same computational requirements, which may not always be the case.

On the other hand, Banino et al. (2021) introduces a more dynamic approach to network architecture by adding or pruning neurons or layers based on the problem at hand. While this approach offers a more flexible model, it comes with its own set of challenges. The most significant challenge is the risk of making the architecture unstable using ACT Graves (2017), which in turn can lead to decreased performance in actual predictions. Additionally, the process of modifying the architecture may be computationally expensive.

Our approach aims to combine the best of both worlds. We introduce an agent-based mechanism for layer selection that dynamically decides which layers of the network to use for each input sample as seen in Figure 3.1. This allows the network to adapt its complexity based on the specific computational needs of each input, without the need to alter the underlying architecture. As a result, our method maintains the stability of the architecture while offering a tailored approach to problem-solving.

3.2 Model Architecture

In this work, we present an innovative neural network architecture, termed **DynaLay** (Dynamic Layer Selection), illustrated in Figure 3.1. Unlike traditional architectures, DynaLay

incorporates an element of introspection by utilizing an adaptive layer-selection mechanism. This mechanism allows the model to selectively activate different layers for different inputs, thereby optimizing both computational efficiency and performance.

DynaLay consists of multiple layers, each capable of performing complex transformations on the input data. These layers are potential candidates for activation as the input data propagates through the network. However, not all layers are activated; instead, an agent A dynamically chooses which layers are most suitable for each input sample $x^{(i)}$.

The role of the agent A is pivotal in DynaLay. The agent receives input activations and employs a decision-making algorithm to determine the most pertinent layer or set of layers l to be activated for each sample $x^{(i)}$ in a batch. This adaptive layer selection is not random but based on learned policies that evaluate the characteristics of each input sample to make an informed decision.

The dynamism of layer selection culminates in the formation of a composite function $g : (x^{(i)}, a) \rightarrow y^{(i)}$, where g represents the neural network and $y^{(i)}$ is the output. Here, a refers to the actions taken by agent A, essentially the selection of layers l. The function g is unique for each input $x^{(i)}$ because it incorporates dynamically chosen layers, effectively making the network adaptive at the granularity of individual samples.

The agent A is trained to minimize an objective function that balances computational efficiency and task performance. Specifically, the objective function comprises two terms: one for the error in the task (e.g., classification error) and another for computational cost, enabling the model to make choices that are both accurate and efficient.

The agent A is trained concurrently with the neural network using reinforcement learning. Every time the agent makes a decision, it receives a reward or penalty based on the objective function, updating its policy to make better future decisions.

By integrating this adaptive layer-selection mechanism, DynaLay offers a level of flexibility and efficiency that is unprecedented in traditional neural network architectures. It allows the model to adapt in real-time to the unique characteristics of each input, thereby offering a more nuanced and effective approach to complex machine learning tasks.

3.3 Architectural Overview

The DynaLay architecture is designed as a modular neural network system comprised of multiple interconnected layers. These layers are not just limited to traditional ones like convolutional or fully connected layers but also include advanced Fixed-Point Iterative (FPI) layers. The architecture is augmented with an intelligent agent A responsible for dynamically selecting the most appropriate layers for each input sample.

3.3.1 Fixed-Point Iterative (FPI) Layers

These layers are specialized for approximating complex functions and are particularly useful for tasks that require a strong incorporation of prior knowledge. They are computationally efficient and can improve the network's convergence behavior.

DynaLay is designed to be computationally efficient. By activating only a subset of layers for each input, it significantly reduces the computational burden. The agent A is trained to make this selection not just based on performance but also on the computational cost,

thereby achieving an optimal trade-off.

Both the main neural network and the agent A are trained end-to-end using backpropagation and reinforcement learning, respectively. Gradients from the task loss and the agent's reward function are back-propagated through the entire network to update the model parameters and the agent's policy.

The effectiveness of DynaLay is evaluated based on task-specific performance metrics such as accuracy, precision, and recall. Additionally, computational efficiency metrics like time complexity and computational cost are also assessed to validate the model's efficiency.

DynaLay is designed to be both interoperable and scalable. It can easily integrate with existing neural network architectures and can scale up to handle more complex tasks and larger datasets without a significant increase in computational cost.

Through this comprehensive architecture, DynaLay successfully introduces a layer of introspection into neural networks, offering an unprecedented balance between computational efficiency and performance across a wide array of machine learning tasks.

In these architectures, we aim to achieve the same decision-based modeling in the original network. Our decision model should perform the following computation on the original network: Let's consider the network diagram above with a 3-layer model.

1. If the token is considered "easy," then it should proceed linearly from the input to the output through the original network without any modification. (this is our base case)
2. If the decision model determines which layer requires a fixed point:
    (a) Convergent loops are iterated through the first layer only.

(b) Convergent loops are iterated through the first two layers, and the result of this is passed to the third layer and then to the output.

(c) All three layers are iterated as one 3-layer model.

While many established architectures MacKay et al. (2018) Vaswani et al. (2023) Hochreiter (1995) He et al. (2015) utilize mechanisms such as skip connections or attention to optimize performance, these techniques are often task-dependent and do not adapt to individual samples within a dataset. **DynaLay** diverges from this approach. **Our core innovation lies in the model's introspective ability to recompute the results on more difficult inputs during inference and adaptively select layers for each sample based on its unique activation profile, optimizing both accuracy and computational efficiency**. Unlike previous work such as Adaptive Computation Time (ACT) Graves (2017), which focuses on adjusting the number of recurrent steps for each sample in a sequence model, **DynaLay** offers finer granularity in layer-wise adaptivity and considers the nuanced contributions of different layers to the final model output for each individual sample.

In contrast, **DynaLay** not only allows for dynamic layer selection in both sequence and feed-forward architectures like CNNs Hochreiter & Schmidhuber (1997), but also provides a more nuanced approach by considering the activation profiles of each layer for every sample.

Let $L = \{l_1, l_2, ..., l_n\}$ be the set of layers in the **Main** Network, and A_i be the activation output from layer l_i. The **Agent** Network is designed to take $\{A_1, A_2, ..., A_n\}$ as input and output a probability distribution $P = [p_1, p_2, ..., p_n]$ defined as $p_i = \text{AgentNet}(A_i)$, where $\sum_{i=1}^{n} p_i = 1$. A layer l is adaptively chosen based on this probability distribution P,

formalized as $l = \text{Sample}(P)$.

Our reward function R is defined as

$$R = \alpha \times \text{Classification Accuracy} - \beta \times \text{Computational Cost} \times \text{Consecutive Layer Count} \quad (3.2)$$

, with α and β as hyperparameters. This guides the Agent Network's decision-making process to optimize both classification accuracy and computational efficiency.

The loss function $\mathcal{L}$ for the entire architecture incorporates this reward function as Equation 3.3:

$$\mathcal{L} = \text{Cross-Entropy Loss} - \gamma \times R \quad (3.3)$$

where γ is a hyperparameter.

This architecture allows **DynaLay** to not only provide dynamic layer selection in both sequence and feed-forward architectures, but also offers a more nuanced approach by considering the activation profiles of each layer for every sample. This results in a more effective and computationally efficient model that adapts to the complexities of individual samples across a wide range of tasks.

3.3.2 Backward Pass

One of the key challenges in training **DynaLay** is the efficient computation of gradients during the backward pass. Given our architecture's dynamic nature and the presence of FPI layers, traditional backpropagation methods prove to be suboptimal.

3.3.3 Gradient Accumulation for Dynamic Layer Selection

For each sample, we have a unique set of layers $L_s = \{l_1, l_2, ..., l_k\}$ chosen by the Agent Network. During backpropagation, we accumulate the gradients for each chosen layer l using a modified gradient accumulation technique, described in [Gradient Accumulation for Dynamic Neural Networks](https://arxiv.org/abs/2210.08572) as specified in Equation 3.4 https://arxiv.org/abs/2106.04350

$$\frac{\partial \mathcal{L}}{\partial W_l} = \sum_{s \in S} \frac{\partial \mathcal{L}_s}{\partial W_l} \quad \text{for each } l \in L_s \tag{3.4}$$

where S is the set of all samples, $\mathcal{L}_s$ is the loss for sample s, and W_l are the parameters of layer l.

3.3.4 Optimized Backpropagation for FPI Layers

For the FPI layers Jeon et al. (2020), we adapt the methodology proposed in Bolte et al. (2022a). This method allows us to compute the gradients implicitly, thereby reducing the computational complexity. Let $(F(x) = 0$ represent the fixed-point equation for the FPI layer as shown in Equation 3.5. Then the gradient $\frac{\partial F}{\partial x}$ can be computed implicitly as:

$$\left(I - \frac{\partial F}{\partial x}\right)^{-1} \frac{\partial F}{\partial W} = -\frac{\partial F}{\partial W} \tag{3.5}$$

where I is the identity matrix and W are the parameters of the FPI layer. Typically, we would calculate $\frac{\partial F}{\partial x}$ during backpropagationRumelhart et al. (1986). However, this approach can be highly inefficient and unstable. Instead, we use orthogonal over-parameterization

Arya et al. (2023)Metz et al. (2022) to find a more effective representation as shown in Equation 3.6:

$$\frac{\partial \mathcal{L}}{\partial W} = O\left(\frac{\partial \mathcal{L}}{\partial \tilde{W}}\right) \tag{3.6}$$

where O is the orthogonal matrix and $\tilde{W}$ is the over-parameterized representation of W Metz et al. (2022)

The backpropagation algorithm in DynaLay is a cornerstone for optimizing both the main model and the AgentNet. To elucidate the methodological complexity, we first describe the gradient flow within the architecture, followed by the advanced techniques incorporated Liao et al. (2019),

3.3.5 Gradient Flow in DynaLay

In DynaLay, the gradient flow is bifurcated between the main model and the AgentNet. Given a loss function $\mathcal{L}$, the gradient $\nabla_{\text{main}}\mathcal{L}$ for the main model is computed as follows:

$$\nabla_{\text{main}}\mathcal{L} = \frac{\partial \mathcal{L}}{\partial W_{\text{main}}} \quad \text{where } W_{\text{main}} \text{ are the weights of the main model.} \tag{3.7}$$

For the AgentNet, the gradient $\nabla_{\text{agent}}\mathcal{L}$ is isolated from affecting the main model's prediction and is computed independently as:

$$\nabla_{\text{agent}}\mathcal{L} = \frac{\partial \mathcal{L}}{\partial W_{\text{agent}}} \quad \text{where } W_{\text{agent}} \text{ are the weights of the AgentNet.} \tag{3.8}$$

Following the work by Arya et al. Arya et al. (2023), we employ automatic differentiation that caters to the discrete randomness introduced by layer selection. The gradients are computed as:

$$\nabla_{\text{discrete}}\mathcal{L} = \mathbb{E}\left[\frac{\partial \mathcal{L}}{\partial W_{\text{discrete}}}\right] \tag{3.9}$$

We draw from Bolte et al. Bolte et al. (2022b) to handle nonsmooth iterative algorithms, using subdifferentials ∂ to calculate the gradients as:

$$\nabla_{\text{nonsmooth}}\mathcal{L} = \partial\mathcal{L}(W_{\text{nonsmooth}}) \tag{3.10}$$

To improve the scalability, we adapt the asynchronous methods from Barham et al. Barham et al. (2022). The gradients for each layer l in an asynchronous setting are calculated as:

$$\nabla_{l,\text{async}}\mathcal{L} = \nabla_l\mathcal{L} + \Delta_{\text{async}} \tag{3.11}$$

where Δ_{async} is the asynchronous correction term. Echoing the sentiments of Metz et al. Metz et al. (2022), we incorporate auxiliary metrics M alongside gradients, optimized as:

$$\mathcal{O} = \nabla\mathcal{L} + \alpha M \tag{3.12}$$

where α is a tunable parameter.

3.3.6 Agent Network (AgentNet)

The Agent Network serves as the linchpin in the **DynaLay** architecture, introducing an introspective dimension uncommon in traditional neural network designs. Unlike conventional architectures where each layer processes every sample in a uniform manner, our Agent Network exercises discernment in selecting the most relevant layers for individual samples. This adaptability optimizes both computational efficiency and model accuracy. Let x represent the input to the architecture, and a_l denote the activation map for the l^{th} layer. These activation maps Chen & Sun (2023) are scrutinized by the Agent Network to generate a probability distribution P over the layers. Mathematically, this distribution is expressed as:

$$P = \text{Adaptive Softmax}(F(a_1, a_2, \ldots, a_L))$$

Here, the function F integrates the activation maps $a_1, a_2, \ldots, a_L$ from different layers, and the Adaptive Softmax function Grave et al. (2017) transforms these into probabilities, enabling the agent to make informed, adaptive decisions on layer selection for each specific sample." Here, F is the function instantiated by the Agent Network, and L is the total number of layers in the architecture. The P_l component of P signifies the probability that the l^{th} layer should be activated for the specific input sample x.

To guide the training of the Agent Network, we employ a reward function R, defined as:

$$R = \alpha \times \text{Acc} - \beta \times C \times \text{Consecutive Layer Count} \tag{3.13}$$

where α and β are hyperparameters that manage the balance between classification accuracy

(Acc) and computational cost (C) and Layer Count is which layer is chosen. During the training process, back propagation Rumelhart et al. (1986) is employed to not only minimize the conventional loss $\mathcal{L}$ but also to maximize the reward R. The total loss function is thus formulated as:

$$\text{Total Loss} = \mathcal{L} - \gamma \times R$$

where, γ is a hyperparameter that controls the significance of the reward term in the overall loss function.

Advanced Backward Propagation in DynaLay [1] Backwardctx, grad_output $z_\star$, layer $\leftarrow$ ctx.saved_tensors max_iter $\leftarrow$ ctx.max_iter tol $\leftarrow 1\times10^{-5}$ $d_z \leftarrow$ Clone and detach grad_output $I \leftarrow$ Identity matrix of d_z Initialize identity matrix Initialize ∇_{agent} Gradient for the agent network $k = 1, \ldots,$ max_iter $f_z = \text{layer}(z_\star)$ $J = \frac{\partial f_z}{\partial z_\star}$ Jacobian matrix $\Delta J = I - J$ Implicit differentiation step $d_{z_{\text{new}}} = d_z \times \Delta J$ Compute updated gradient $\delta = \frac{\|d_{z_{\text{new}}} - d_z\|}{\|d_{z_{\text{new}}}\|}$ $\delta <$ tol Break $d_z = d_{z_{\text{new}}}$ $\nabla_{\text{agent}} =$ Advanced Techniques$(\nabla_{\text{agent}}, J, \Delta J)$ Update agent's gradient Update agent network using ∇_{agent} Use optimizer to update agent $d_z, \text{None}, \text{None}$

3.4 Forward Propagation Algorithm

In addition to the backpropagation algorithm, understanding the forward passis essential for grasping the full scope of DynaLay. The forward process Wang (2013) Greener (2023) Wang et al. (2019) involves solving a fixed-point equationHabiba & Pearlmutter (2021) to find a suitable latent state z that serves as the output of the AgentNet.

Forward Propagation in DynaLay [1] Forwardctx, input_tensor, layer, max_iter $z \leftarrow$ Clone and detach in

$z \leftarrow$ Require gradient for z tol $\leftarrow 1 \times 10^{-2}$ $k = 1, \ldots,$ max_iter $f_0 \leftarrow$ Apply layer to z size of $f_0 \neq$ size of z $f_0 \leftarrow$ Interpolate to match dimensions with z $\frac{\|f_0 - z\|}{\|f_0\|} <$ tol Break $z \leftarrow f_0$ z

The algorithm performs a fixed-point iteration to find the state z that satisfies the equation $z = f_{\text{layer}}(z)$, where f_{layer} is the layer's transformation function. The iteration is performed up to a maximum of max_iter iterations or until the following convergence criterion is met:

$$\frac{\|f_0 - z\|}{\|f_0\|} < \text{tol} \tag{3.14}$$

where tol is a predefined tolerance level set to 1×10^{-2}.

As the iterations progress, it is essential to ensure that the dimensions of f_0 and z are consistent Almeida (1990). By employing this forward algorithm, DynaLay achieves a balance between computational efficiency and convergence, ensuring robust and reliable operation across diverse tasks.

CHAPTER 4
RESULTS

To validate the efficacy of our proposed architecture, **DynaLay**, we conducted experiments on multiple datasets, including image datasets: CIFAR-10Giuste & Vizcarra (2020) and CIFAR-100Krizhevsky (2009)and text dataset: WikiText-2 Merity et al. (2016) and IMDB Maas et al. (2011). The primary goals of these experiments are twofold: 1) to evaluate the classification accuracy of our model in comparison with existing state-of-the-art architectures, and 2) to assess the computational efficiency improvements achieved through our adaptive layer selection mechanism. We implemented all our models in PyTorch Paszke et al. (2019) and conducted training on NVIDIA A40 GPUs with 40GB memory. For optimization, we chose the Adam optimizer Kingma & Ba (2017), setting the learning rate at 1×10^{-3} and the batch size at 64. Hyperparameters α, β, and γ were optimized to 0.5, 0.3, and 0.2, respectively, through 9-fold cross-validation. All models were trained over 100 epochs. The tolerance for fixed-point iterations in the **DynaLay** architecture was set to 1×10^{-5}.

4.0.1 Comparative Analysis

In the context of adaptivity, DynaLay incorporates an AgentNet mechanism that enables dynamic layer selection, setting it apart from the more rigid 3-layer model and the solver-dependent adaptivity in Bai et al. (2019) models. The AgentNet mechanism not only enhances adaptability but also contributes significantly to computational efficiency.

Table 4.1 offers a comprehensive comparison between DynaLay, a standard 3-layer model, and DEQBai et al. (2019) models equipped with solvers Agarwal et al. (2018). A salient

feature of DynaLay is its remarkable efficiency in backward pass computations. Specifically, it outclasses both the standard 3-layer and DEQBai et al. (2019) models in backward pass time for CIFAR10 and CIFAR100, clocking in at 87 and 90 minutes, respectively. This efficiency is pivotal for scaling and real-time applications.

Furthermore, DynaLay achieves competitive test accuracies of 92% and 85% on CIFAR10 and CIFAR100, respectively. This implies that the computational advantages of DynaLay do not compromise its performance, reinforcing its utility as a robust and efficient architecture.

Table 4.1 Comparison of Computational Steps for Different Models

2***Aspect**	**DynaLay**		**3 layer CNN Model**		**DEQ with Solvers**
	CIFAR10	**CIFAR100**	**CIFAR10**	**CIFAR100**	**CIFAR10**
Backward Pass Time (mins)	**87**	**90**	120	140	600
Adaptivity	**AgentNet**		None		Solver-based
Test Accuracy (%)	**92**	**85**	88	82	91

4.0.2 Dynamic Layer Selection in DynaLay

During our experiments on checking the frequency at which layers are selected, remarkably, the "No Layer" or Straightforward option emerges as the most frequently selected, indicating its ability to capture essential features across a broad spectrum of tasks. Conversely, Loop functionalities on Layer1 and Layer2 are invoked 25% and 20% of the time, respectively. The diminishing frequency of Layer3 selections implies that the model tends to minimize reliance on this layer's fixed-point operations as it stabilizes. Figure 4.1 elucidates the frequency distribution of each layer's selection during the evaluation process. This dynamic layer selection mechanism substantially augments DynaLay's computational efficiency. By intelligently and adaptivelyBolte et al. (2022b)Jeon et al. (2020) deciding the appropriate

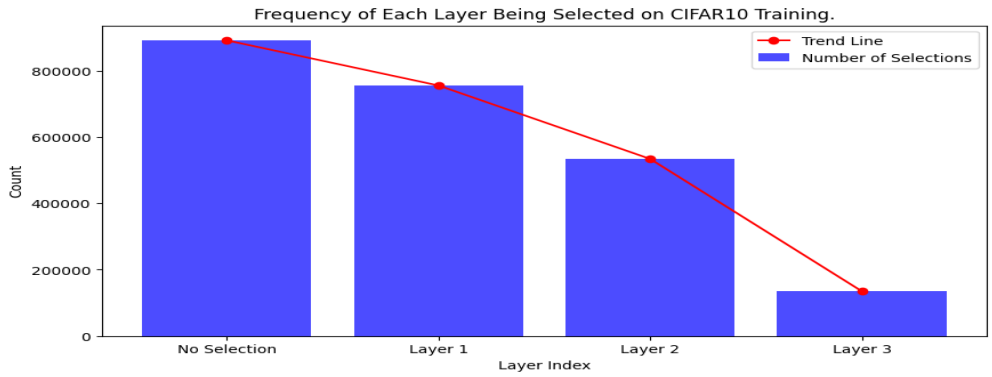

Figure 4.1 Frequency distribution of layer choices during CIFAR10 training. The bar chart quantifies the selection frequency of each layer across numerous training epochs, offering insights into their relative importance for CIFAR10 performance.

layer for FPIRajeswaran et al. (2019) execution, DynaLay eliminates redundant computations, especially when less complex layers are sufficient for the task at hand Bolukbasi et al. (2017). This adaptability stands in sharp contrast to traditional architectures like CNNs and LSTMs Hochreiter & Schmidhuber (1997), which are constrained by their rigid, predetermined structures.

4.0.3 Loss Convergence on CIFAR-10 and CIFAR-100

The convergence behavior of our model's training and testing loss is a critical aspect of its evaluation. Figure **??** provides a detailed insight into this aspect. The similarity in convergence patterns between CIFAR-10 and CIFAR-100 also suggests that our model is scalable and adaptable to different tasks and data distributions, aligning well with the core objectives of this research project. We also showed experiments on LSTM stacked layer in Appendix.

4.0.4 Influence of Pre-training on DynaLay's Efficacy

The versatility of our DynaLay architecture is further underscored by its robust performance irrespective of whether it undergoes pre-training Han et al. (2021). To elucidate this, Figure 4.2 showcases a side-by-side comparison of key performance indicators—test accuracy and test loss—across 100 epochs for both pre-trained and non-pre-trained configurations.

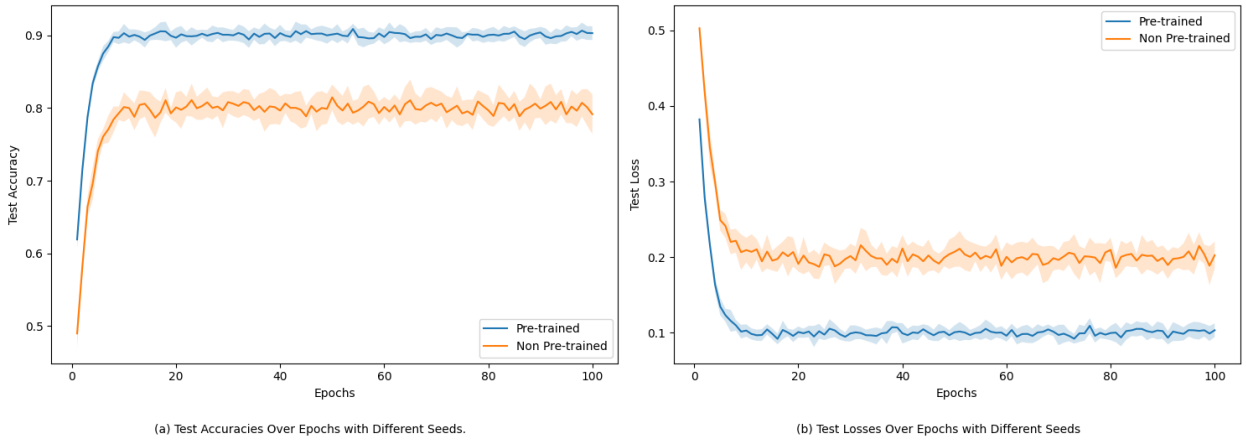

Figure 4.2 Temporal evolution of test accuracies and losses for pre-trained and non pre-trained configurations. Subplot (a) captures the test accuracies, while subplot (b) focuses on the test losses. Both metrics are plotted as functions of the epoch count.

The observed metrics reveal a discernible advantage when employing pre-training. Specifically, the pre-trained model consistently surpasses its non-pre-trained counterpart in both accuracy and loss metrics. This superior performance is attributed to a "warm-up" phase consisting of 20 epochs of pre-training for the main model. This phase aids in initializing the activations to more optimal states, which likely catalyzes faster and more stable convergence during subsequent training.

4.0.5 Role of the Agent in DynaLay's Performance and Adaptability

Our Agent network also undergoes a pre-training phase lasting the same number of epochs. We preserve the weights Lillicrap et al. (2014) from these pre-training phases and use them as initialization points for the comparative experiment. The findings, delineated in Figure 4.2, validate the efficacy of this approach by showcasing noticeably improved performance metrics for the pre-trained model.

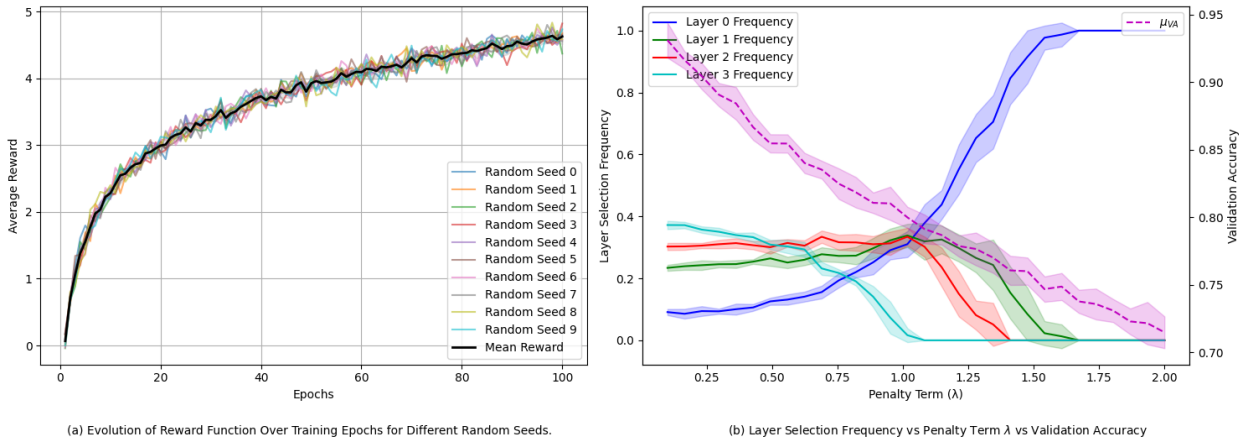

Figure 4.3 (a) Temporal Dynamics of the Reward Function Across Training Epochs for Distinct Random Seeds. The graph offers an incisive look into the reward function's evolution over the training epochs, integrating both accuracy and computational efficiency into its formulation. Each trace corresponds to a unique random seed, thereby providing a robust measure of the model's resilience to varying initial conditions. (b) Interplay Between Layer Selection Frequency, Penalty Term λ, and Validation Accuracy. This tri-axis plot delivers a granular portrayal of how layer selection frequency and validation accuracy respond to changes in the penalty term λ.

Figure 4.3: (a) delineates the trajectory of the average reward function throughout the training epochs. A key observation here is the emergent stabilization of the reward function as the training epoch count ascends. This equilibrium is indicative of the agent's escalating competence in judiciously selecting layers that not only enhance performance but also

optimize computational expenditure.

Further nuance in the agent's decision-making process is captured in Figure 4.3(b). Each curve on the plot signifies the frequency with which the agent elects to utilize a specific layer across the spectrum of available λ penalty terms. Accompanying these curves is a dashed line that represents the validation accuracy achieved under these conditions. Higher λ values act as deterrents against the selection of computationally burdensome layers, compelling the agent towards more resource-efficient alternatives.

This in-depth analysis fortifies the understanding of the agent's role within DynaLay, particularly its aptitude for adaptively managing computational resources without compromising model performance. The agent's proficiency in this balancing act is pivotal for the scalability and applicability of DynaLay across a wide range of tasks and computational settings.

4.0.6 Impact of Fixed-Point Iteration on Different Model Configurations

4.0.6.1 Objective

The primary objective of this experiment is to rigorously assess the efficacy of incorporating Fixed-Point Iteration (FPI) into various layers of our **DynaLay** architecture. We focus on four distinct configurations to perform this assessment:

1. **Model 0**: A straightforward architecture comprised of Layer 1 $\rightarrow$ Layer 2 $\rightarrow$ Layer 3, devoid of FPI.

2. **Model 1**: Utilizes FPI exclusively in Layer 1.

3. **Model 2**: Employs FPI in both Layer 1 and Layer 2.

4. **Model 3**: Applies FPI across all layers (Layer 1 → Layer 2 → Layer 3).

Our hypothesis posits that the incorporation of FPI into an increasing number of layers will yield a commensurate improvement in key performance metrics, notably test loss and accuracy. Specifically, we project that Model 3 will exhibit superior performance relative to the other configurations, owing to the enhanced complexity and optimization capabilities conferred by FPI.

For evaluation, a test set of 100 samples is employed as seen in Figure 4.4. Each of the four model configurations is subjected to this test set, with both test loss and accuracy being meticulously recorded. It is imperative to note that all models are trained under identical hyperparameter settings to ensure a level playing field.

4.0.7 Text-Based Experiments with LSTMs

To further validate the effectiveness and adaptability of DynaLay, we extend our experiments to text-based tasks employing Long Short-Term Memory (LSTM) networks Hochreiter & Schmidhuber (1997). We experiment with two datasets: WikiText-2 for language modeling and IMDB for sentiment analysis.

4.0.7.1 Model Configuration

Our LSTM-based DynaLay model consists of three LSTM layers. The LSTM layers are parameterized as follows:

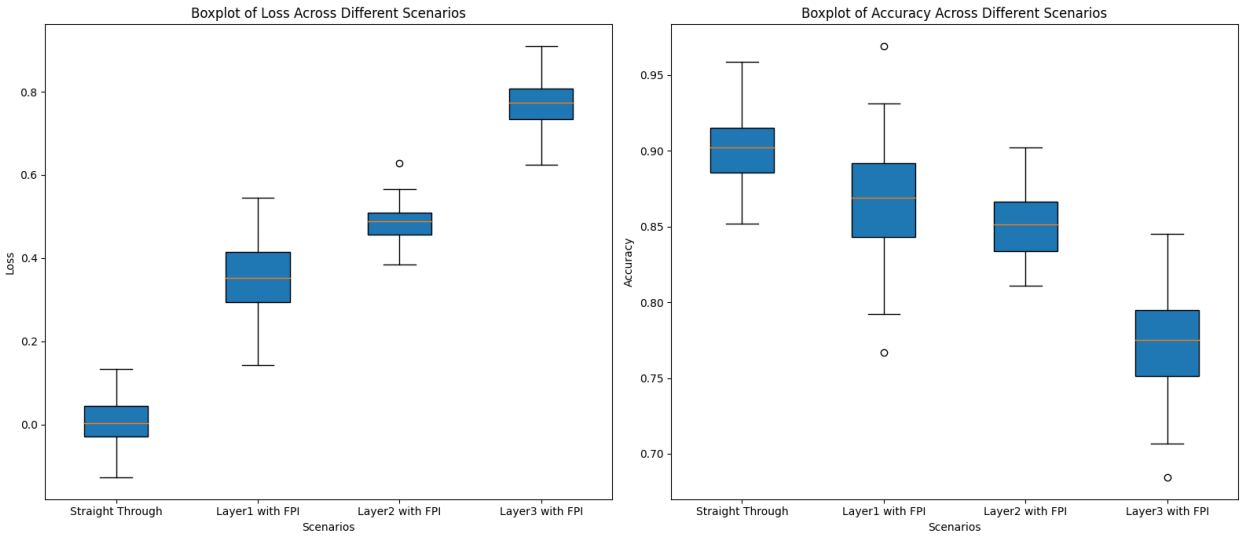

Figure 4.4 Boxplots illustrating the distribution of test loss and accuracy across four distinct model configurations. The configurations vary in the complexity and number of layers utilizing Fixed-Point Iteration (FPI).

$$
\begin{aligned}
f_t &= \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \\
i_t &= \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \\
o_t &= \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \\
\tilde{C}_t &= \tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \\
C_t &= f_t * C_{t-1} + i_t * \tilde{C}_t \\
h_t &= o_t * \tanh(C_t)
\end{aligned}
\tag{4.1}
$$

In this equation, f_t, i_t, o_t are the forget, input, and output gates, respectively. h_t is the hidden state, C_t is the cell state, and x_t is the input at time t.

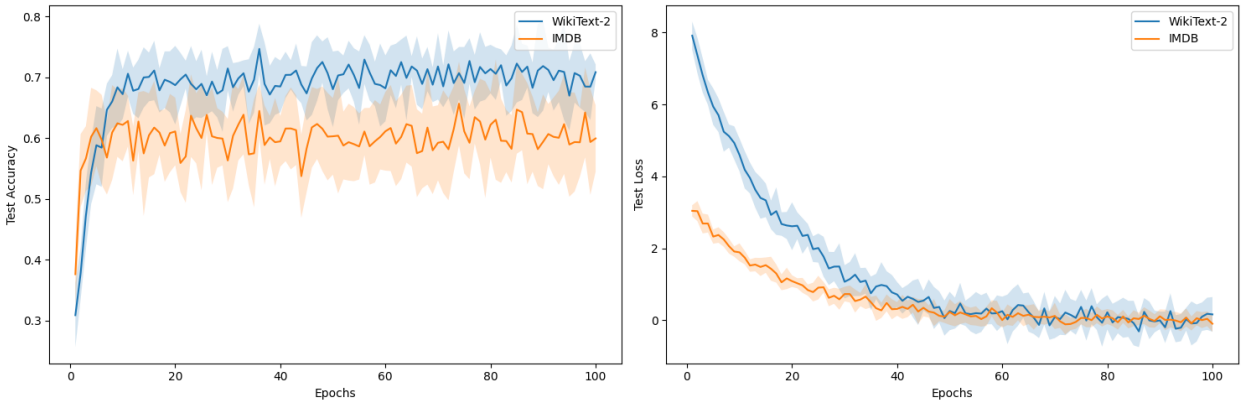

Figure 4.5 Boxplots representing the performance of LSTM-based DynaLay configurations on WikiText-2 and IMDB datasets.

4.0.7.2 Experimental Results

We employ Fixed-Point Iteration (FPI) on individual LSTM layers just as in our CNN experiments. The results are summarized in Figure 4.5, which shows that the LSTM model with FPI on the third layer achieves the best performance in terms of both accuracy and computational cost.

Through these experiments, we demonstrate that DynaLay's adaptive layer selection mechanism is equally effective for text-based tasks, thereby confirming its versatility across different data modalities and tasks.

CHAPTER 5
DISCUSSIONS

We introduced DynaLay, an innovative architecture designed to bring a level of introspection to neural networks. The key feature is an intelligent agent capable of adaptively selecting the appropriate layers for each input sample. Our empirical evidence demonstrated a meaningful balance between computational efficiency and performance, validating the architecture's introspective capabilities.

The introspective nature of DynaLay allows the network to adapt its computational graph in real-time based on the data it receives. This self-awareness enables the model to make intelligent decisions about how to process each input, thereby optimizing computational resources and performance. The introspective ability also generalizes well, offering potential applications in multiple domains where real-time adaptability is crucial, such as autonomous systems and real-time analytics. The architecture stands out when compared to other models, particularly in its ability to introspectively adapt to the complexity and nature of each input, a feature largely absent in current architectures. While introspection brings many benefits, it also adds a layer of complexity to both the architecture and the training process. Training an agent to make introspective decisions requires careful tuning and additional computational resources. Future work could focus on improving the introspective agent's decision-making algorithms, potentially incorporating more advanced reinforcement learning techniques or even elements of meta-learning. An interesting avenue for research would be to develop metrics for quantifying the effectiveness of the model's introspective

abilities, offering a standard for evaluating similar architectures in the future.

CHAPTER 6
CONCLUSION

In this thesis, we have presented DynaLay, a groundbreaking architecture that introduces the concept of introspection into neural networks. Through the integration of a reinforcement learning agent, our architecture is capable of dynamically selecting appropriate layers for each input sample. This level of adaptability allows DynaLay to achieve an impressive balance between computational efficiency and model performance.

6.0.1 Contributions

Our key contributions include:

1. A novel architecture that seamlessly integrates Fixed-Point Iterative (FPI) layers and a reinforcement learning agent for dynamic layer selection.

2. A unique reward function designed to balance classification accuracy, computational time, and decision-making diversity.

3. Empirical validation showing that DynaLay achieves comparable or superior accuracy to traditional models while significantly reducing computational overhead.

6.0.2 Implications

The implications of this work extend beyond just improving the efficiency of neural networks. The introspective nature of DynaLay opens the door for a new generation of intelligent systems capable of real-time adaptability and self-optimization. This is particularly impactful

for applications in autonomous systems, real-time analytics, and edge computing where computational resources are often limited.

6.0.3 Future Directions

While our results are promising, there are numerous avenues for future research:

- Refinement of the introspective agent's decision-making algorithm, possibly incorporating more advanced reinforcement learning techniques.
- Exploration of different kinds of layers and operations that could be dynamically selected, such as attention mechanisms or different types of activation functions.
- Development of metrics for quantifying the effectiveness of introspection in neural networks.
- Extending the architecture to attention mechanism and more advanced and novel RL methods.
- exploring real life scenarios with this model for foundational purposes to make it an model agnostic architecture

DynaLay represents a significant advance in the ongoing quest for more efficient and intelligent neural networks. By leveraging introspection, the architecture sets a new standard for what is achievable in terms of both performance and computational efficiency. As computational resources continue to be a bottleneck in deploying large-scale machine learning

models, the importance of such introspective and adaptive approaches will only continue to grow.

REFERENCES

Agarwal, P., Jleli, M., & Samet, B. 2018, Banach Contraction Principle and Applications (Singapore: Springer Singapore), 1–23

Almeida, L. B. 1990

Arya, G., Schauer, M., Schäfer, F., & Rackauckas, C. 2023, Automatic Differentiation of Programs with Discrete Randomness

Bai, S., Kolter, J. Z., & Koltun, V. 2019, Deep Equilibrium Models

Banino, A., Balaguer, J., & Blundell, C. 2021, PonderNet: Learning to Ponder

Barham, P. et al. 2022, Pathways: Asynchronous Distributed Dataflow for ML

Bolte, J., Le, T., Pauwels, E., & Silveti-Falls, A. 2022a, Nonsmooth Implicit Differentiation for Machine Learning and Optimization

Bolte, J., Pauwels, E., & Vaiter, S. 2022b, Automatic differentiation of nonsmooth iterative algorithms

Bolukbasi, T., Wang, J., Dekel, O., & Saligrama, V. 2017, Adaptive Neural Networks for Efficient Inference

Chatziafratis, V., Panageas, I., Sanford, C., & Stavroulakis, S. A. 2022, On Scrambling Phenomena for Randomly Initialized Recurrent Networks

Chen, Z., & Sun, Q. 2023, Extracting Class Activation Maps from Non-Discriminative Features as well

Giuste, F. O., & Vizcarra, J. C. 2020, CIFAR-10 Image Classification Using Feature Ensem-

bles

Grave, E., Joulin, A., Cissé, M., Grangier, D., & Jégou, H. 2017, Efficient softmax approximation for GPUs

Graves, A. 2017, Adaptive Computation Time for Recurrent Neural Networks

Greener, J. G. 2023, bioRxiv

Habiba, M., & Pearlmutter, B. A. 2021, Neural Network based on Automatic Differentiation Transformation of Numeric Iterate-to-Fixedpoint

Han, X. et al. 2021, AI Open, 2, 225

He, K., Zhang, X., Ren, S., & Sun, J. 2015, Deep Residual Learning for Image Recognition

Hochreiter, S. 1995

Hochreiter, S., & Schmidhuber, J. 1997, Neural Computation, 9, 1735

Jeon, Y., Lee, M., & Choi, J. Y. 2020, Differentiable Forward and Backward Fixed-Point Iteration Layers

Kidger, P. 2022, On Neural Differential Equations

Kingma, D. P., & Ba, J. 2017, Adam: A Method for Stochastic Optimization

Krizhevsky, A. 2009

Liao, R., Xiong, Y., Fetaya, E., Zhang, L., Yoon, K., Pitkow, X., Urtasun, R., & Zemel, R. 2019, Reviving and Improving Recurrent Back-Propagation

Lillicrap, T. P., Cownden, D., Tweed, D. B., & Akerman, C. J. 2014, Random feedback weights support learning in deep neural networks

Maas, A. L., Daly, R. E., Pham, P. T., Huang, D., Ng, A. Y., & Potts, C. 2011, in Pro-

ceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies (Portland, Oregon, USA: Association for Computational Linguistics), 142–150

MacKay, M., Vicol, P., Ba, J., & Grosse, R. 2018, Reversible Recurrent Neural Networks

Merity, S., Xiong, C., Bradbury, J., & Socher, R. 2016, Pointer Sentinel Mixture Models

Metz, L., Freeman, C. D., Schoenholz, S. S., & Kachman, T. 2022, Gradients are Not All You Need

Paszke, A. et al. 2019, PyTorch: An Imperative Style, High-Performance Deep Learning Library

Rajeswaran, A., Finn, C., Kakade, S., & Levine, S. 2019, Meta-Learning with Implicit Gradients

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. 1986, nature, 323, 533

Schmidhuber, J. 2015, Neural Networks, 61, 85

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. 2023, Attention Is All You Need

Wang, P.-W., Donti, P. L., Wilder, B., & Kolter, Z. 2019, SATNet: Bridging deep learning and logical reasoning using a differentiable satisfiability solver

Wang, Q. 2013, Journal of Computational Physics, 235, 1

Werbos, P. 1990, Proceedings of the IEEE, 78, 1550