POLYTECHNIC UNIVERSITY OF BUCHAREST
FACULTY OF AUTOMATIC CONTROL AND COMPUTERS
COMPUTER SCIENCE AND ENGINEERING DEPARTMENT

BACHELOR THESIS

A comprehensive study on Integrating Quantum Physics in Deep Learning Networks for Optimizations in Computer Vision and Natural Language Processing Fields

QRKT-GAN: Neural Ordinary Differential Equation-Inspired Generative Adversarial Network with Numerical Runge-Kutta Methods for Quantum Visual Transformer-Based Generator and Discriminator

Cătălin-Alexandru Rîpanu

Thesis advisor:

Șl. dr. ing. Dumitru-Clementin Cercel

BUCHAREST
2024

ACKNOWLEDGEMENTS

First and foremost, I would like to express my deepest gratitude to my advisor, Prof. Dumitru-Clementin Cercel, for his unwavering support and insightful guidance throughout my research journey. His expertise and encouragement have been essential in exploring the intricate intersection of Machine Learning and Quantum Mechanics, widely referred to in the literature as Quantum Machine Learning. This interdisciplinary field, though initially daunting, is at the cutting edge of technological advancement and innovation in both areas.

I am also profoundly thankful to Prof. Pantelimon-George Popescu for his advice, recommendations, and exceptional lectures on Quantum Computing. His mentorship has been instrumental in building a robust understanding of this groundbreaking paradigm, which promises new solutions to problems previously deemed insurmountable by classical methods.

Furthermore, I wish to acknowledge the significant contributions of the academic teaching assistants and collaborators encountered during my bachelor's studies. Their guidance during challenging times has been crucial to my academic development. Last but not least, I am also grateful for the computational resources provided by the Computer Science and Engineering Department, which were essential for the design and implementation of this project.

In addition, I extend my appreciation to my friends for their continuous support, ideas, and advice, especially during moments when inspiration was scarce. Lastly, my heartfelt thanks go to my parents for their enduring sacrifices and unwavering support throughout this journey into the unknown. Their belief in me has been a constant source of motivation.

ABSTRACT

Deep Learning models, such as Generative Adversarial Networks (GANs) [1] and Visual Transformers (ViTs) [2, 3], have demonstrated remarkable results across various domains in Machine Learning and Artificial Intelligence, including Object Classification, Image Segmentation, Sentiment Analysis, and Synthetic Data Generation. These neural networks are pivotal in advancing systems that require high precision and a deep understanding of complex data across a variety of tasks in both Natural Language Processing (NLP) and Computer Vision (CV).

However, the effectiveness of Deep Learning models comes with significant challenges: they require an extensive [4] number of parameters to learn and extract meaningful features from real-world data. Additionally, these models need vast amounts of information to achieve desired performance levels. Obtaining such large sets can be difficult as real-world data is often not publicly available and can be challenging to collect and curate. This results in substantial computational resource requirements for both training and hyperparameter optimization, often achieved through exhaustive techniques such as grid search [5].

To address these challenges, this thesis proposes a novel hybrid Generative Adversarial Network architecture that employs Quantum Visual Transformers (QViTs) [6] as both the Generator and Discriminator. Visual Transformers [2, 3] are selected for their superior ability to manage intricate data representations. A key innovation in this architecture is the integration of Ordinary Differential Equation (ODE) [7] solvers as Encoders [8], enhancing the model capability to capture temporal dynamics and complex data structures, and improving the residual connections within the transformer architecture to mitigate the vanishing gradients problem [9] even more.

Moreover, this architecture incorporates Variational Quantum Circuits [10] within both the Self-Attention Mechanisms [11] and the Multi-Layer Perceptrons (MLPs) [12] of the Visual Transformers [2, 3]. By leveraging the principles of Quantum Mechanics, these Quantum circuits [13] can perform complex algebraic operations more efficiently than classical methods, offering a significant computational advantage.

The performance of this hybrid approach is benchmarked against a purely classical neural network baseline from the literature using datasets from both CV and NLP areas. Specifically, the Quantum ODE configurations are tested on CIFAR-10 [14], CIFAR-100 [14], MNIST [15], IMDb Reviews [16] and ILSVRC 2012 [17] datasets. These neural network architectures are effectively trained and tested using numerical simulations tailored to Quantum circuits [18]. The obtained results indicate that these hybrid approaches achieves comparable classification and generation performance to the classical baseline, while requiring fewer trainable parameters.

Furthermore, the reduced parameter count in these hybrid models opens up the possibility of running it on Noisy Intermediate-Scale (NISQ) [19] quantum devices for both training and inference. This feasibility is a significant breakthrough, as it implies that quantum-enhanced models can be trained and deployed on actual quantum computers, which are currently limited in terms of the number of qubits and operational fidelity [19].

This thesis demonstrates the potential of integrating Quantum Computing [20], especially Quantum Mechanics, with advanced Deep Learning architectures to create more efficient and powerful networks which can significantly reduce computational costs while maintaining high performance, paving the way for more scalable and effective AI applications [21].

CONTENTS

1 Introduction

1.1 Context

Artificial Intelligence models, particularly Deep Learning ones, have made significant contributions to solving real-world tasks, greatly improving human lives in various fields, such as Medical Image Recognition [22]. Despite their impressive capabilities, Deep Learning models come with substantial drawbacks regarding computational resources [4] and effort. Achieving high performance with these models necessitates learning millions to billions of parameters, also called weights or artificial neurons, which demands considerable resources and preparation time. This limitation also has negative environmental impacts due to high power consumption.

Over the years, researchers have developed numerous solutions to mitigate the problem of minimizing the number of parameters using interesting classical algorithms and techniques. These include, for example, specialized activation functions for neural layers like Rectified Linear Unit [23], Leaky ReLU [24], Gaussian Error Linear Unit [25], Softmax [26], Sigmoid [27] and Hyperbolic Tangent [28]. Additionally, effective optimizers such as Adam [29] and Stochastic Gradient Descent [30] have been utilized to combat this limitation, along with methods like Learning Rate Schedulers [31], Weight Decay [32], and Dropout [33] to reduce also the chances of model overfitting. Over the years, various architectural designs and network combinations have emerged to address these challenges. However, the tradeoff between high computational resource consumption and performance remains a difficult issue, especially when scaling up the dimensions of available noisy real-world datasets used for training. It is unlikely[1] to be efficiently resolved in the near future using solely classical approaches.

A segment of Computer Science researchers, in collaboration with physicists, have taken a bold and innovative approach to these challenges by exploring solutions from a physical perspective, leading to the emergence of Quantum Computation and Quantum Information [20]. Leveraging the principles of Quantum Mechanics, AI research has begun to explore Deep Learning optimizations using quantum facilities, such as Variational Quantum Circuits [13] or Quantum Algorithms [10], to develop hybrid Deep Learning models. By combining classical and quantum methodologies, new experiments can be conducted to address performance and resource demands more efficiently, thus utilizing the unique capabilities of quantum hardware. This involves training these models on it, also known as Parameterized Quantum Circuits [13], where the parameters are referred to as the qurons of the quantum model.

Quantum Computing leverages properties like superposition [34] and entanglement [35], enabling the execution of complex algebraic operations that are infeasible for classical computers. Quantum Algorithms [10], such as Grover's Algorithm [36], Quantum Phase Estimation [37], Quantum Fourier Transform [38] and Deutsch-Jozsa Quantum Parallelism [39] provide exponential speedups for certain tasks. Integrating these Quantum principles with Deep Learning can significantly reduce the number of parameters and computational resources required while maintaining or even enhancing model performance. This interdisciplinary distinctive approach, also known as Quantum Machine Learning, opens up exciting possibilities for the future of AI, promising advancements that could transform how complex problems are approached and solved in ways that classical computers, with their current architecture, could never achieve.

[1] https://www.theverge.com/24066646/ai-electricity-energy-watts-generative-consumption

1.2 Problem

Over the years, numerous algorithms, techniques, network architectures, and methodologies have been proposed in the field of Deep Learning to tackle a wide array of tasks, such as Object Classification, Synthetic Data Generation and Detection, across various domains including Natural Language Processing, Computer Vision, Domain Adaptation [40], and Knowledge Distillation [41]. These approaches have achieved high performance metrics like Accuracy, Area Under the Curve (AUC), Receiver Operating Characteristic (ROC), and F1 scores. The advent of High Performance Computing (HPC) facilities, particularly classical parallelism, has enabled Deep Learning models to process billions of examples from noisy real-world datasets effectively, thus yielding impressive results.

Despite these advancements, a significant drawback persists: the enormous number of neurons and the corresponding computational resources required by these models. Traditional solutions in classical Deep Learning, such as Cross Validation [42], Hyperparameter Tuning techniques like Grid Search [5] or Random Search [5], Keras Tuner [43], and Layer Augmentation until model performance plateaus or overfits, are commonly employed. To mitigate overfitting, regularization techniques such as Ridge [44], Lasso [44], Elastic-Net [44] or Dropout [33] can be applied. However, these methods often involve exhaustive hyperparameter searches, which are extremely time-consuming, especially when dealing with validation sets comprising billions of data points for complex tasks. Furthermore, group sparsity regularizers can be applied to network parameters, where each group acts on a single neuron, thus reducing the number of parameters by up to 80% while maintaining or even enhancing the network accuracy score [45].

Another method employed to address these challenges is Transfer Learning [46], which involves finding and loading a pretrained model as a starting point and then fine-tuning it on a specific dataset. This approach can help achieve good performance with less effort on architecture design and expedite the training process. However, the availability of suitable pretrained models is limited, and fine-tuning may yield only decent, if not disappointing, results, due to differences in the data distributions between the pretraining and fine-tuning datasets.

The primary challenge lies in balancing time, memory and power consumption on classical systems to make robust and valid neural architectural choices. Furthermore, reducing the number of neurons too drastically is not a solution as it can lead to underfitting and degraded performance, making it difficult to solve complex and challenging tasks efficiently. The iterative process of tuning hyperparameters and optimizing model architectures is computationally intensive and often results in a significant trade-off between resource consumption and model performance [5].

Moreover, the reliance on vast amounts of training data for achieving desired performance further exacerbates these issues. Real-world datasets are often difficult to obtain and may not be publicly available, which complicates the development and fine-tuning of effective Deep Learning models. The extensive time and computational costs associated with hyperparameter optimization and model training present significant obstacles to advancing AI capabilities while maintaining efficiency and sustainability. This trade-off between computational resource consumption and performance remains a persistent and challenging problem in the field of Deep Learning, highlighting the need for innovative solutions that can overcome these limitations.

1.3 Objective

In the current landscape of Deep Learning research, there is an ongoing investigation into whether integrating models into a purely quantum environment can significantly reduce power consumption and the number of trainable parameters without sacrificing performance. This exploration leverages techniques such as the Ansatz [47] and Variational Quantum Circuits [10]. As classical supercomputers advance to handle increasingly complex deep models, their power consumption grows almost exponentially. In contrast, quantum machines exhibit an exponential increase in computational power while maintaining a linear increase in power consumption. This intriguing characteristic makes a strong case for Quantum Computing: with the advent of fault-tolerant quantum computers, qubits could serve as highly efficient artificial neurons in neural networks. Quantum-inspired techniques could enable networks to operate with a vast number of neurons per layer at minimal energy cost, dramatically reducing overall energy consumption[2].

Building on the challenges outlined in the above section, the primary goal of this thesis is to explore Deep Learning models, both pure Quantum and hybrid, that leverage neural ODE-based architectures [48] inspired by non-trivial Variational Quantum Circuits [10] within the context of Visual Transformers [2, 3] and their complex Encoders [8] and Decoders [49]. The most important objective is to minimize the number of parameters used by deep network layers utilizing the idea [50] of replacing classical linear projection layers in Multi-Head Attention [11] subroutines and Multi-Layer Perceptrons [12] with Quantum Circuits [13]. Additionally, low truncation error Neural Ordinary Differential Equation [48][51] techniques will be employed to further optimize Transformer Layers, namely Runge-Kutta Methods [52] of 1st, 2nd, 3rd, and 4th order. An analysis of Runge-Kutta 4th order method optimization which is not theoretically grounded in standard RK methods, but in terms of training, will also be included [51].

To better understand the rationale and motivation behind the proposed Deep Learning model in this paper, it is necessary to compare these Quantum ODE-based Transformers configurations with their classical counterparts using the datasets mentioned in Abstract. The network utilizing the optimized Runge-Kutta 4th order method will be leveraged within the context of Vision Transformers (ViTs) [2, 3] and integrated into a Generative Adversarial Network (GAN) [1] architecture. In this configuration, both the Generator and Discriminator will be Quantum Visual Transformers (QViTs)[3], incorporating the optimized Neural ODE solver to generate realistic and informative synthetic data without being limited to specific types of distributions. This will be achieved using techniques such as Data Augmentation [53] and Image Recognition at Scale [3] in the context of ViTs [2].

The final proposed architecture in this thesis, named QRKT-GAN, will be tested on a real-world consistent dataset, namely CIFAR-10 [14], and compared using the same data with a renowned classical counterpart, TransGAN [54], which is a strong architecture built entirely free of Convolutions [55]. This comparative analysis will provide another argument for the possible advantages of integrating Quantum Computing [20] with advanced Deep Learning neural networks under the right context.

[2] https://www.eetimes.eu/how-Quantum-Computing-can-help-make-ai-greener

[3] https://openreview.net/pdf?id=p7xPXoKB0H

1.4 Paper Structure

The next chapter delves into the most relevant and important classical and Variational Quantum Algorithms [10] and techniques developed over the years to leverage both Quantum and hybrid logic in Deep Learning for model optimization. This chapter will illustrate how these both hybrid and pure Quantum Neural Networks (QNNs) [49] are designed to be compatible with Noisy Intermediate-Scale (NISQ) [19] devices provided by specialized vendors such as IBM [56], Google [57], Microsoft [58], and Amazon [59]. It will explore the advancements that make it possible to run complex Deep Learning models on quantum computers, highlighting the interplay between Quantum Computing [20] and Deep Learning for neural depth decreasing while not losing too much performance.

Section three will discuss the foundational concepts of Machine Learning, Deep Learning and Ordinary Differential Equations. It will also provide a comprehensive overview of Quantum Computation and Quantum Information [20], and will also introduce the emerging field of Quantum Machine Learning [60]. This section aims to explain how quantum principles can enhance the capabilities of Deep Learning solutions in addressing high-demanding tasks, some of which are classified as NP-Hard [61] or NP-Complete [62]. By understanding these fundamentals, the reader will gain insight into the potential synergies between Classical and Quantum Computing [20], which together can form a duality in the world of Artificial Intelligence.

Section four will present an in-depth analysis of the proposed Ordinary Differential Equation-based Quantum Neural Networks (QNNs) [63, 51] configurations. This includes their motivation, underlying concepts, mathematical and quantum frameworks, physical limitations, and future potential within their problem domain environment. The chapter will also discuss in-detail the new ODE-based GAN architecture in detail, QRKT-GAN, explaining how it incorporates the traversed knowledge to minimize the solution of the optimization problem outlined in Context section, while maintaining acceptable performance in Synthetic Image Generation.

The fifth chapter will detail the methodologies adopted for conducting the experiments. This includes a comprehensive description of the datasets used, their internal characteristics, and the reasons for their selection. It will compare various Quantum ODE-based [7] configurations with their classical ODE-based networks [48] counterparts to fully understand the benefits and limitations of quantum-enhanced models. Additionally, the chapter will provide a detailed comparison between the proposed QRKT-GAN model and its classical baseline, TransGAN [54], using the CIFAR-10 [14] dataset. This analysis will highlight the benefits and challenges of integrating Quantum Computing into Deep Learning, presenting it as a viable alternative solution for very specific optimization problems.

The final chapter will summarize the main ideas presented in the thesis, highlighting the relevance of the results and drawing key conclusions. It will also propose future work and outline potential directions for further research in the field of Quantum Deep Learning models, focusing on power consumption, computational resources and performance optimization. This chapter will emphasize the importance of continued exploration in combining Quantum Computing with Machine Learning [60] to achieve more efficient and powerful neural solutions.

2 Related Work

The relentless demand for improving human lives continues unabated, with increasingly complex and data-intensive needs emerging. This necessitates the development of sophisticated models capable of delivering accurate and reliable results. However, the complexity and depth of these models often result in significantly high computational resource consumption, which has environmental implications due to the energy required for such intensive processing.

Over the years, numerous classical, quantum, and hybrid methods have been proposed to address the optimization problem of minimizing model depth without significantly sacrificing performance. One notable approach is real-time learning of the number of neurons in deep networks using structured sparsity during training, which dynamically adjusts the network structure to optimize performance without excessive resource use [45]. Another method involves crafting specialized architectures that maintain a constant computational budget while slightly increasing depth to improve performance. This is achieved by dimensionality reduction before applying expensive Convolutions [31] with larger patch sizes [64]. Unfortunately, Convolutions [31] have been shown [55] to be less effective in capturing global feature interactions, as they primarily focus on local patterns in data processing. In contrast, the Self-Attention mechanism [11] utilized in Transformers [2] excels at capturing these global interactions.

In addition, some techniques focus on minimizing weights during training by splitting network weights into sets or hierarchies of multiple groups, each using disjoint sets of features. This allows parallelization, even in an embarrassingly manner, as each subnetwork operates independently, enhancing computational efficiency [65]. Alternatively, population-based heuristic algorithms, such as Particle Swarm Optimization (PSO), have shown promise in optimally determining the number of parameters without exhaustive searches like grid search [5], and thus saves valuable computational resources during the tuning process of these Deep Learning models [66]. Another approach involves reducing model parameters in deep neural networks via product-of-sums matrix decompositions, which decompose linear operators as a product of sums of simpler linear operators [67]. For Deep Convolutional Neural Network [31] models, kernel-sharing between multiple convolutional layers can also be applied [68]. Kernel-sharing is possible only between layers having the same kernel size, input, and output channels.

Despite their advantages, these classical methods have inherent drawbacks. Designing neural architectures from scratch to meet specific requirements is challenging and time-consuming. Moreover, methods like Particle Swarm Optimization [66] require optimization of additional parameters, such as activation functions [27, 23, 24] and the number of epochs, to be truly effective. Quantum Neural Networks (QNNs) [63] and Quantum Algorithms [10] offer new avenues for addressing these limitations, providing potential solutions for power-efficient Deep Learning. Various Quantum frameworks, technologies, GPU-based numerical simulators, and access to real hardware have been developed to facilitate the implementation of quantum phenomena in Machine Learning.

However, the quantum approach shares a common challenge with classical methods: the need for specific adaptations for each type of Deep Learning model. In the classical domain, the entire architectural framework must be meticulously analyzed to prevent performance degradation. In quantum domain, constructing valid Variational Quantum Circuits (VQCs) [13] and Algorithms [10] that efficiently mimic classical functions is crucial to avoid the same pitfalls.

3 Background

This chapter will delve into the main and most important advancements in Machine Learning, presenting key concepts as they appeared in the literature. It will discuss the motivation and necessity of Deep Learning, a fundamental subdomain of Machine Learning, to fully understand the creation and utilization of neural networks. This will include an exploration of why Generative Adversarial Networks (GANs) [1] and Visual Transformers [2, 3] were developed, along with a detailed examination of their architectural specificities and functions.

Furthermore, the chapter will provide an overview of the techniques used to identify optimal hyperparameters, which are crucial for further neural model optimization. This will cover various approaches to fine-tuning Deep Learning models to enhance their performance.

A significant portion will be dedicated to Ordinary Differential Equations (ODEs) and their most popular numerical solvers, highlighting the foundational aspects of the Runge-Kutta methods [52]. It will explain how to interpret Transformer layers from an ODE perspective [51], providing a mathematical framework for understanding these non-trivial neural network structures.

The chapter will also present a concise summary of Quantum Computing [20], establishing the prerequisites for understanding Quantum Machine Learning [60]. This section will articulate the motivation behind integrating Runge-Kutta ODE methods [52, 7] into Quantum Computing [20]. By exploring Quantum Circuits [13] and Quantum Gates [69], Variational Quantum Algorithms [10], and Variational Quantum Circuits [13], the chapter will illustrate their motivation and application in the context of Quantum Transformers [50].

By comprehensively covering these topics, the chapter aims to equip readers with a deep understanding of the theoretical and practical foundations necessary for advancing in the field of Quantum Machine Learning [60], particularly through the integration of ODE methods [48] in quantum frameworks [18].

3.1 Machine Learning

Machine Learning is the process of enabling computers to learn from data and make decisions without being explicitly programmed to do so. The essential aspect of Machine Learning is training models on data before they can autonomously make decisions. Thus, data is fundamental to Machine Learning algorithms. Without data, Machine Learning cannot function, as models rely entirely on the information provided to make future predictions. The data used must accurately reflect real-world scenarios because Machine Learning models depend on this data to learn and generate accurate predictions for new, unseen data [70].

In practice, Machine Learning involves extensive mathematical operations such as matrix multiplication, feature scaling, and normalization. Consequently, the data fed into Machine Learning algorithms must be represented as numerical vectors or numbers, as opposed to text or other non-numerical forms that machines cannot directly interpret.

For instance, textual data and alphabets are not directly understandable by machines; hence, they must be converted into numerical form before being processed by Machine Learning models. Techniques such as one-hot encoding, word embeddings, or other numerical trans-

formation methods are used to convert text into a machine-readable format, enabling models to learn from and predict based on the data provided [70].

Machine Learning algorithms, along with neural networks, have been proposed for many decades. However, during the early days, the lack of sufficient data and limited computational power hindered their practical application. Today, the scenario has drastically changed. Companies generate vast amounts of data, and computational resources have become extraordinarily powerful and accessible. Services like Google Cloud and Amazon Web Services (AWS) provide scalable computational power, allowing users to run complex Machine Learning algorithms without the need for setting up their own hardware infrastructure. This has led to an exponential growth in the demand for Machine Learning and Deep Learning applications [70].

Understanding Machine Learning and Deep Learning is crucial in this data-rich environment. The ability to harness data through these technologies enables companies to make informed predictions, optimize processes, and drive innovation. The more effectively Machine Learning models are trained and utilized, the greater the value they can bring to an organization, ultimately contributing to increased profitability and competitive advantage.

Types of Machine Learning [70]:

- **Supervised learning** involves training a Machine Learning model on a labeled dataset, where the output labels are known. The model learns to map inputs to outputs based on this labeled data. During training, the model predictions are compared with the actual labels to evaluate its performance and adjust its parameters accordingly. This iterative process continues until the model achieves satisfactory accuracy. An example of supervised learning is predicting house prices. Given a dataset with features like square footage, number of bedrooms, and location, along with corresponding house prices, the model learns to predict the price of a house based on these features. By comparing its predictions with the actual prices, the model improves over time.
- **In unsupervised learning**, the model is trained on data without labeled outputs. The goal is for the model to identify patterns, structures, or relationships within the data. One common application of unsupervised learning is customer segmentation, where customers are grouped based on their behavior, such as purchase history or browsing patterns, without pre-defined labels. This helps businesses tailor marketing strategies to different customer segments.
- **Semi-supervised learning** combines elements of both supervised and unsupervised learning. It uses a small amount of labeled data along with a larger amount of unlabeled data. The model is initially trained on the labeled data, and then it generalizes its learning to the unlabeled data. This approach is useful when obtaining labeled data is expensive or time-consuming. An example of semi-supervised learning is text document classification. A model is first trained on a subset of documents with known categories. Then, it is used to classify the remaining documents, which do not have predefined labels, based on the patterns it has learned.

3.2 Gradient Descent

Gradient Descent (GD) is a foundational first-order optimization algorithm widely used in Machine Learning (ML) and Deep Learning (DL) for minimizing cost or loss functions. This iterative method works by iteratively adjusting the parameters of a model to find the local minimum or maximum of a given function, thereby improving the model accuracy. For instance, in linear regression, Gradient Descent helps minimize the difference between predicted and actual values by updating the weights and biases. Its significance and simplicity make it a fundamental topic in nearly all introductory Machine Learning courses, as it provides a clear understanding of how models learn and improve from data [71].

The Gradient Descent algorithm is not universally applicable to all functions. For it to work effectively, the function must meet two specific criteria: it must be differentiable and convex.

Mathematically:

- A function is differentiable if it has a derivative at each point in its domain. This means the function must be smooth and continuous without any sharp corners or discontinuities. Differentiability is crucial because Gradient Descent relies on the gradient (the derivative) to determine the direction and magnitude of the steps to take towards the minimum or maximum. If the function is not differentiable, the algorithm cannot compute the gradient accurately, leading to unreliable optimization results.

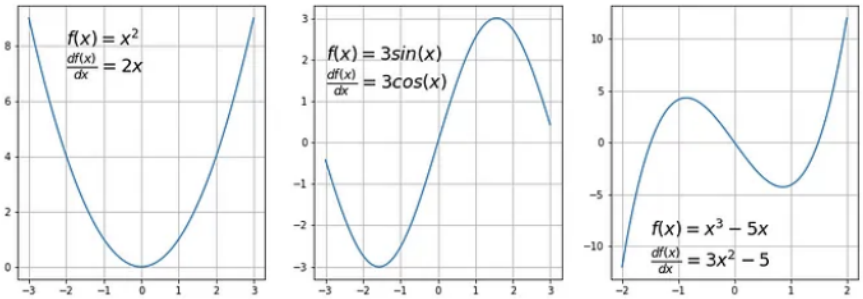

Figure 1: Examples of differentiable functions. Image from [71].

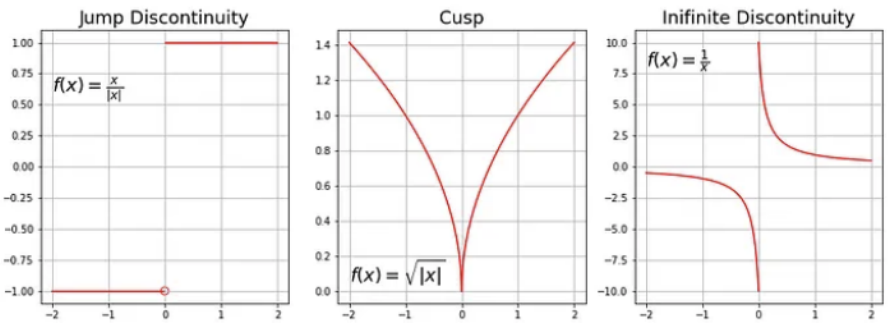

Figure 2: Examples of non-differentiable functions [71].

- A function is convex if the line segment between any two points on the function graph lies above or on the graph. Formally, a function $f(x)$ is said to be convex if for all x_1 and x_2 in its domain, and for all λ in the interval $[0, 1]$, the following inequality holds:

$$f(\lambda x_1 + (1 - \lambda)x_2) \leq \lambda f(x_1) + (1 - \lambda)f(x_2) \tag{1}$$

  Convexity ensures that any local minimum is also a global minimum, making it easier for Gradient Descent to find the optimal solution.

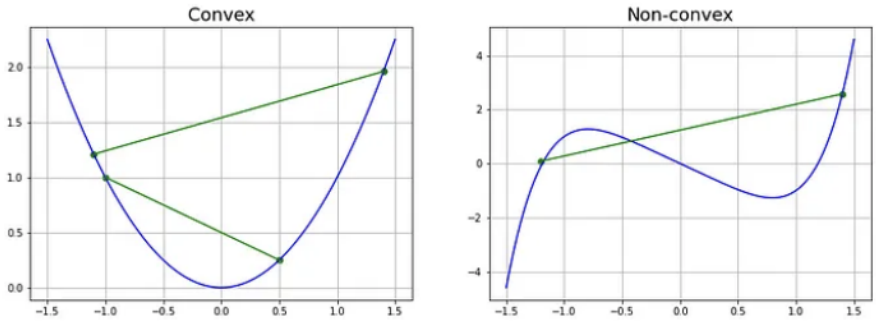

Figure 3: Examples of convex and non-convex functions [71].

Gradient Descent is an optimization algorithm used to minimize a function by iteratively moving towards the steepest descent direction. At each step, it calculates the gradient of the function at the current point, scales it by a learning rate, and updates the position by moving in the opposite direction of the gradient. This process ensures that the algorithm moves towards the function minimum. The update rule can be expressed as:

$$\mathbf{a}_n = \mathbf{a}_{n-1} - \eta \nabla f(\mathbf{a}_{n-1}) \tag{2}$$

where:

- $\mathbf{a}_{n-1}$ is the current position.
- η is the learning rate.
- $\nabla f(\mathbf{a}_{n-1})$ is the gradient of the function at the current position.

By repeatedly applying this update rule, Gradient Descent converges to the minimum of the function. If the goal were to maximize the function, the algorithm would instead add the scaled gradient [71].

The learning rate, denoted as η, is a crucial hyperparameter in gradient descent algorithms that determines the step size at each iteration while moving toward a minimum of the loss function. The choice of learning rate significantly affects the performance of the algorithm:

- **Small Learning Rate**: If the learning rate is too small, the convergence process will be slow, potentially requiring many iterations to reach the optimal point. In some cases, the algorithm may even hit the maximum number of iterations before converging.

- **Large Learning Rate**: If the learning rate is too large, the algorithm might not converge to the optimal point. Instead, it could overshoot the minimum, causing the algorithm to oscillate or even diverge entirely, failing to find a satisfactory solution.

Thus, selecting an appropriate learning rate is essential for the efficient and effective training of Machine Learning models [71].

3.2.1 AdamW Optimizer

Adam combines the strengths of both Stochastic Gradient Descent (SGD) [72] with Momentum and Root Mean Square Propagation (RMSProp). It calculates the exponential moving averages of the gradient first and second moments. Like SGD with Momentum, it helps smooth out the updates to accelerate convergence. Similar to RMSProp, Adam uses adaptive learning rates for each parameter, allowing smaller steps for steep gradients and larger steps for flatter areas. Additionally, Adam adjusts these learning rates by considering both the mean of the first moment (the gradient) and the mean of the second moment (the squared gradient). The parameter update rule in Adam is given by [1]:

$$\theta_{t+1} = \theta_t - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} \tag{3}$$

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) \nabla_\theta f(\theta_t) \tag{4}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2)(\nabla_\theta f(\theta_t))^2 \tag{5}$$

where:

- θ is the parameter to be updated
- η is the initial learning rate
- β_1 is a decay rate for the first moment, with a default value of 0.9
- β_2 is a decay rate for the second moment, with a default value of 0.999
- ϵ is a small number used to avoid division by zero
- $\nabla_\theta f(\theta)$ is the gradient of the loss function $f(\theta)$

The exponential moving average of the squared gradients v_t in the last equation is sometimes referred to as the "second moment" or "uncentered variance" (uncentered because the mean of the gradients is not subtracted). Additionally, m_t is often called the "first moment" or "mean," representing the exponential moving average of the gradients, which captures the cumulative history of gradients. In Adam, the direction of the update is determined by normalizing the first moment with respect to the second moment [73].

Initially, the moving averages m_0 and v_0 are set to zero vectors. This initialization causes the moment estimates to be biased towards zero, particularly during the initial timesteps, especially when the decay rates β_1 and β_2 are close to 1. To address this issue, "bias-corrected" estimates

[1] `https://www.linkedin.com/pulse/understanding-adam-adamw-dsaisolutions-ileof/`

$\hat{m}_t$ and $\hat{v}_t$ are calculated. This correction helps control the weights while approaching the global minimum, preventing high oscillations. The formulas for the bias corrections are as follows [73]:

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \tag{6}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t} \tag{7}$$

If no correction is applied, with $\beta_1 = 0.9$ and $\beta_2 = 0.999$, for the initial step, we have $m_1 = 0.1g_t$ and $v_1 = 0.001g_t^2$, causing the denominator in the update rule to become large. However, with bias-corrected estimates, the denominator becomes less sensitive to β_1 and β_2. The default values of β_1 and β_2 are set to 0.9 and 0.999, respectively, to retain as much historical gradient information as possible [73].

In practice, β_2 is usually set much closer to 1 than β_1, following the recommendation of the authors ($\beta_2 = 0.999$, $\beta_1 = 0.9$). Consequently, the update coefficient $1 - \beta_2 = 0.001$ is significantly smaller than $1 - \beta_1 = 0.1$. By accounting for both the first and second moments when updating parameters, Adam improves optimization efficiency, especially in complex models with numerous parameters.

However, Adam can be sensitive to the initial learning rate and other hyperparameters, which can affect its convergence and stability. Moreover, it may overfit, particularly with small datasets.

To address the generalization issue, Ilya Loshchilov and Frank Hutter introduced AdamW, an improved version of Adam, in their 2019 paper "Decoupled Weight Decay Regularization" [2]. Unlike Adam, which implicitly links weight decay to the learning rate, AdamW decouples weight decay from the optimization process.

This separation allows the learning rate and weight decay to be optimized independently. As a result, adjusting the learning rate does not necessitate recalculating the optimal weight decay, leading to more stable and effective optimization.

3.3 Deep Learning

Deep Learning is a subfield of Machine Learning that focuses on algorithms inspired by the structure and function of the brain neural networks. Unlike traditional Machine Learning algorithms that often require manual feature extraction and rely on simpler linear or non-linear models, Deep Learning employs complex architectures known as Artificial Neural Networks (ANNs). These neural networks are designed to automatically and hierarchically extract features from raw data, making them particularly powerful for tasks involving large and unstructured datasets.

[2] `https://openreview.net/forum?id=Bkg6RiCqY7`

Deep Learning models, particularly those involving deep neural networks (DNNs), consist of multiple layers of interconnected neurons. Each neuron in a layer processes input data, applies a transformation using weights and biases, and passes the output to the next layer. The hierarchical nature of these layers allows Deep Learning models to learn increasingly abstract and complex representations of the data as it moves through the network. This capability enables Deep Learning models to perform exceptionally well on tasks such as Image and Speech Recognition, Natural Language Processing, and Game Playing.

One of the key advantages of Deep Learning is its ability to learn from vast amounts of data without the need for extensive manual feature engineering. For instance, in Image Recognition tasks, traditional Machine Learning approaches might require handcrafted features such as edges, textures, and shapes. In contrast, Deep Learning models can automatically learn to detect these features directly from the pixel data [74].

Deep Learning has achieved remarkable success across various domains, driven by advancements in computational power (especially GPUs), large datasets, and novel network architectures like Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), and Transformers [2]. CNNs, for example, are particularly well-suited for image-related tasks, while RNNs and Transformers [2] excel in Sequential Data Processing, such as Language Modeling and Translation.

3.3.1 Artificial Neural Networks (ANN)

An artificial neural network (ANN) is composed of interconnected nodes, also known as artificial neurons or units, organized into layers. These layers include the input layer, one or more hidden layers, and the output layer. Because of the numerous layers, ANNs are sometimes referred to as deep neural networks when they have many hidden layers [75].

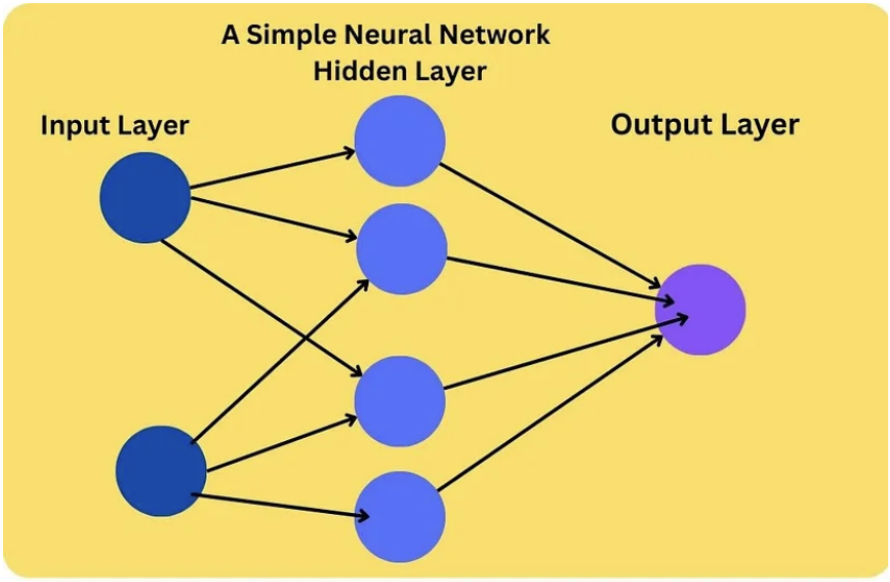

Figure 4: A simple Artificial Neural Network [75].

The three main layers in most ANNs are:

- **Input Layer**: This layer receives the raw input data, which can be in various forms such as images, text, or numerical values.
- **Hidden Layers**: Located between the input and output layers, these layers are responsible for processing the input data through weighted connections and activation functions.
- **Output Layer**: The output layer produces the final results of the neural network's processing, such as classifications or numerical predictions.

Training the neural network involves adjusting the weights of the connections to minimize the error in the output. Artificial neural networks have been successfully applied to a wide range of tasks, including Image Recognition, Natural Language Processing, Speech Recognition, Recommendation Systems, Financial Predictions, and many others. Their ability to learn and adapt from data makes them a powerful tool in Machine Learning and Artificial Intelligence.

3.3.2 Multi-Layer Perceptron (MLP)

Multilayer Perceptrons (MLPs) were initially inspired by the Perceptron, a supervised Machine Learning algorithm designed for binary classification. The original Perceptron was only capable of handling linearly separable data. To overcome this limitation, the Multilayer Perceptron was introduced, enabling the handling of both linearly and non-linearly separable data [76].

An MLP is a type of neural network that belongs to the class of feed-forward neural networks. In these networks, the neurons in one layer are connected to the neurons in the subsequent layer in a forward manner, without any loops.

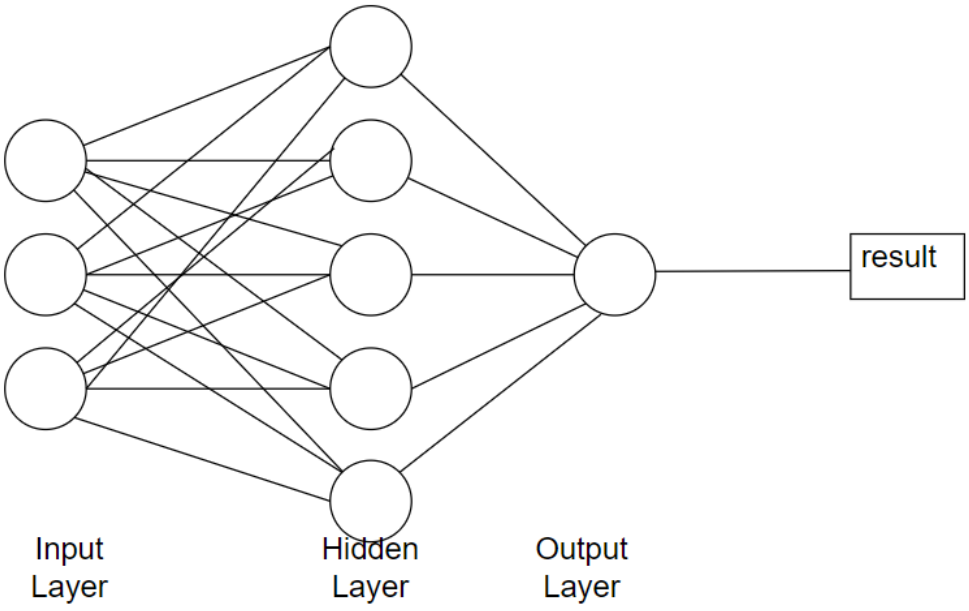

Figure 5: Multilayer Perceptron [76].

A Multilayer Perceptron (MLP) is a type of Artificial Neural Network (ANN) characterized by a layered architecture of interconnected neurons designed to process data through multiple stages. The fundamental structure of an MLP comprises an input layer, one or more hidden layers, an output layer, activation functions, and a set of weights and biases:

- **Input Layer**: This is the initial layer that receives input data, which can be numerical values, images, or other types of structured data.
- **Hidden Layers**: Positioned between the input and output layers, these layers are responsible for processing the input data through complex computations. There is no fixed limit to the number of hidden layers, though MLPs usually feature a modest number to balance complexity and computational efficiency.
- **Output Layer**: The final layer that delivers the output results derived from the data processed throughout the network.

A defining feature of MLPs is the use of backpropagation, a supervised learning technique for training neural networks. Backpropagation involves adjusting the weights in the network by propagating the error from the output back through the network. This iterative process fine-tunes the network parameters, enhancing performance and minimizing errors.

Due to their straightforward design, MLPs typically require shorter training times to learn data representations and produce outputs. However, they often necessitate more powerful computing resources than standard computers, particularly devices equipped with Graphics Processing Units (GPUs), to handle the intensive computations involved.

MLPs have proven to be versatile and efficient tools in various applications, from image recognition and natural language processing to financial forecasting and beyond. Their ability to learn from data and improve over time underscores their importance in the realm of Machine Learning and Artificial Intelligence.

3.3.3 Generative Adversarial Network (GAN)

Generative Adversarial Networks (GANs) [1] have revolutionized the field of Artificial Intelligence, capturing widespread attention for their ability to generate and enhance data in innovative ways. They are pivotal in applications such as creating photorealistic images, transforming styles in artwork, and generating lifelike human faces. Introduced by Ian Goodfellow and his team in a groundbreaking 2014 NeurIPS paper, GANs [1] are specialized Machine Learning systems designed to mimic specific data distributions [77].

At the heart of a GAN [1] are two dynamically interacting neural networks: the Generator, which creates data, and the Discriminator, which evaluates the authenticity of the generated data. This dynamic interaction forms an "adversarial" training process, conceptualized as a competitive game. Here, the Generator strives to produce data so realistic that the Discriminator cannot reliably tell it apart from genuine data, achieving a 50% deception rate.

A Generative Adversarial Network (GAN) [1] comprises two neural networks that engage in simultaneous adversarial training:

- **Generator**: This network transforms random noise into coherent data outputs, such as images. Its objective is to generate data that closely resembles real-world data.
- **Discriminator**: This network evaluates both real data and data generated by the Generator, attempting to distinguish between the two. It outputs the likelihood that the provided data is real.

Throughout training, the Generator continuously improves its ability to create data that the Discriminator cannot distinguish from real data. Concurrently, the Discriminator enhances its skill in differentiating authentic data from the generated data. This adversarial "game" leads to the Generator producing progressively more realistic data over time.

Generative Adversarial Networks (GANs) employ loss functions to train both the generator and the discriminator, ensuring they both improve over time. The loss function helps adjust the weights of these models during training to enhance their performance. Both the generator and the discriminator utilize the binary cross-entropy loss, which can be expressed as [77]:

$$L(y, p) = -(y \log(p) + (1 - y) \log(1 - p)) \tag{8}$$

where:

- $L(y, p)$ is the loss value.
- y is the true label (either 0 or 1).
- p is the predicted probability of the sample belonging to class 1.

The discriminator objective is to correctly classify real samples as real and fake samples (generated by the generator) as fake. The loss for the discriminator is typically represented as [77]:

$$L_D = -\frac{1}{2}\mathbb{E}_{x \sim p_{data}(x)}[\log(D(x))] - \frac{1}{2}\mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))] \tag{9}$$

where:

$$\mathbb{E}_{x \sim p_{data}(x)}[f(x)] \approx \frac{1}{N}\sum_{i=1}^{N} f(x_i) \tag{10}$$

$$\mathbb{E}_{z \sim p_z(z)}[f(z)] \approx \frac{1}{M}\sum_{i=1}^{M} f(z_i) \tag{11}$$

In these equations:

- x_i are samples from the real dataset.
- N is the number of samples from the real dataset.
- z_i are samples from the noise distribution.
- M is the number of samples from the noise distribution.

The first term on the right-hand side penalizes the discriminator for misclassifying real data, while the second term penalizes the discriminator for misclassifying the fake data produced by the generator.

The goal of the generator in a Generative Adversarial Network (GAN) [1] is to produce samples that the discriminator cannot distinguish from real data. The generator loss function is designed to penalize the generator when the discriminator correctly identifies its outputs as fake. Mathematically, the generator's loss can be expressed as [77]:

$$L_G = -\frac{1}{2}\mathbb{E}_{z\sim p_z(z)}\left[\log(D(G(z)))\right] \tag{12}$$

Here, z represents the noise input to the generator drawn from a prior distribution $p_z(z)$, $G(z)$ is the generated sample, and $D(G(z))$ is the discriminator probability that $G(z)$ is a real sample.

The combined GAN [1] loss, also known as the minimax loss, encapsulates the adversarial nature of GAN [1] training. In this setup, the generator and discriminator engage in a two-player minimax game where the discriminator aims to maximize its ability to classify real and fake data correctly, while the generator strives to minimize the discriminator ability by generating realistic data. The combined loss is given by [77]:

$$L_{GAN} = \min_G \max_D (L_D + L_G) \tag{13}$$

Here, L_D is the loss for the discriminator, and L_G is the loss for the generator.

Gradient penalty is a technique used to stabilize the training by penalizing the gradients if they become too steep. This can help in stabilizing the training and avoiding issues like mode collapse. The gradient penalty term is defined as [77]:

$$GP = \lambda \mathbb{E}_{\hat{x}\sim P_{\hat{x}}}\left[(\|\nabla_{\hat{x}} D(\hat{x})\|_2 - k)^2\right] \tag{14}$$

Where:

- $\mathbf{GP}$ represents the gradient penalty term.
- λ is a hyperparameter that controls the strength of the penalty.
- The gradient component $\nabla_{\hat{x}} D(\hat{x})$ is the gradient of the discriminator's output with respect to its input $\hat{x}$.
- $\mathbf{P_{\hat{x}}}$ represents the distribution of interpolated samples between real and generated data.
- $\mathbf{k}$ is a target norm for the gradient, often set to 1.

The purpose of introducing a gradient penalty is to enforce a constraint on the gradient of the discriminator output. This constraint ensures that the discriminator does not become overly confident, which can lead to sharp decision boundaries and unstable training. By keeping the gradient close to a target norm k, typically set to 1, the gradient penalty helps maintain a smooth decision boundary.

The hyperparameter λ plays a crucial role in this process. A higher value of λ results in a stronger penalty for deviations from the target gradient norm, while a lower value of λ results in a weaker penalty. Proper tuning of λ is essential for the stability and performance of the GAN.

The interpolated samples $\hat{x}$ are generated by taking convex combinations of real and generated samples. This interpolation helps in evaluating the gradient penalty across the data distribution, promoting better generalization and stability in the training process.

The discriminator loss with gradient penalty can be incorporated as follows:

$$L_D = \mathbb{E}_{x \sim p_{\text{data}}(x)} \left[\log(D(x))\right] + \mathbb{E}_{z \sim p_z(z)} \left[\log(1 - D(G(z)))\right] + GP \tag{15}$$

This loss function consists of the usual GAN discriminator loss components for real and generated data, plus the gradient penalty term to regularize the discriminator behavior.

The discriminator loss with gradient penalty in GANs [1] integrates standard loss components for real and generated data, augmented by a gradient penalty term to regulate the discriminator behavior. By penalizing large gradients, this method promotes smoother behavior in the discriminator, contributing to a more stable training process for both the generator and the discriminator. The gradient penalty helps prevent sharp changes in the discriminator output, which can otherwise lead to instability and issues like mode collapse, where the generator produces limited and repetitive outputs.

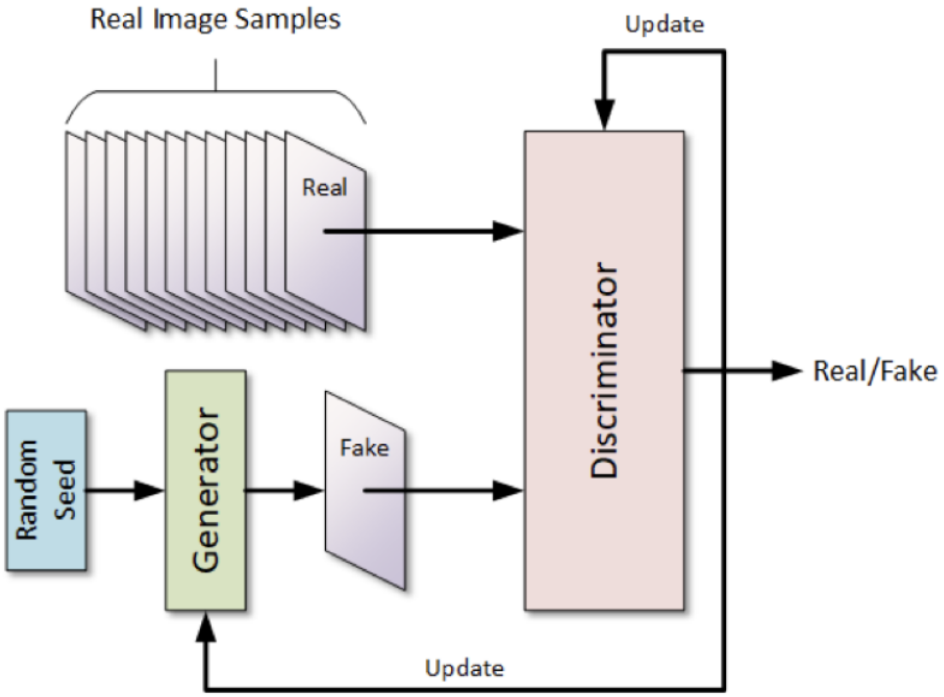

Figure 6: The Generative Adversarial Network Architecture [78].

GAN [1] are not only remarkable for their ability to create high-quality data, but also for their versatility across a range of complex tasks. Their unique adversarial training mechanism

ensures that the generated data is continually refined, pushing the boundaries of what is possible in Machine Learning and Artificial Intelligence.

3.3.4 Visual Transformer (ViTs)

Originally designed for Natural Language Processing, the Transformer [2] architecture has been successfully adapted to other domains [79].

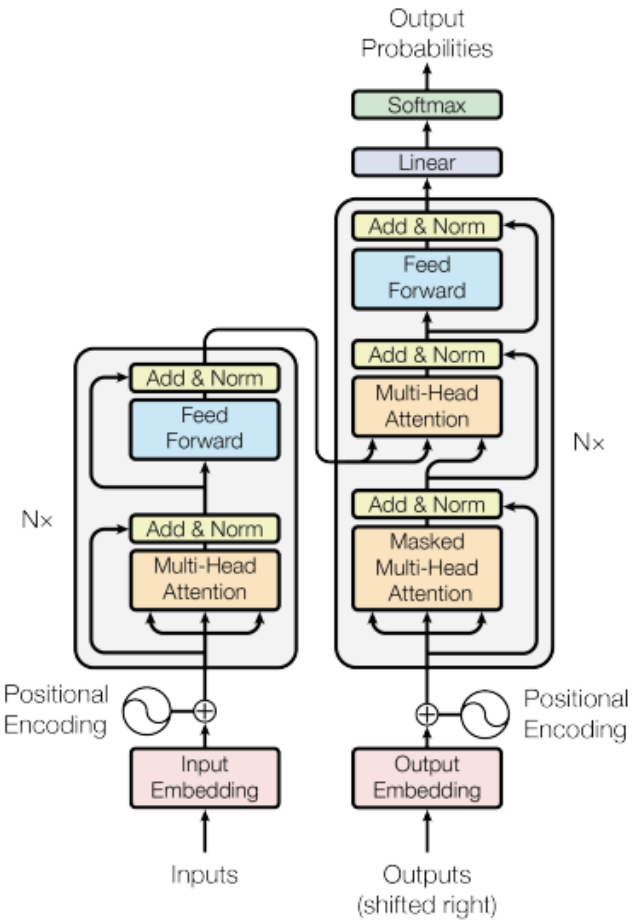

Figure 7: The Transformer Architecture [2].

A notable example is the Vision Transformer (ViT) [3], which adapts the Transformer [2] for Computer Vision tasks. In ViTs [3], an image is divided into a sequence of patches, each linearly embedded and treated as input tokens for a stack of Transformer layers, collectively referred to as the Transformer Encoder.

The ViT has achieved state-of-the-art performance on various image classification benchmarks, showcasing the Transformer [2] architecture's versatility and effectiveness across different domains.

Visual Transformer (ViT) [3] is a Deep Learning architecture that leverages the self-attention mechanism of Transformers [2] to process and understand visual data such as images [79].

By dividing an input image into smaller patches, ViT [3] allows each patch to attend to others, leading to state-of-the-art performance in various computer vision tasks

The usual workflow of Visual Transformers [3] in Image Classification task include [79]:

1. Decomposing the Image into Patches
    - **Input Image**: Consider an input image with dimensions $H \times W \times N$, where H is the height, W is the width, and N represents the number of channels (such as 3 for RGB images).
    - **Patch Flattening**: The image is divided into smaller patches, each of size 16×16 pixels. For an RGB image, this results in each patch being flattened into a vector of 768 elements (since $16 \times 16 \times 3 = 768$). This transformation converts the spatial information of the image patches into a form suitable for further processing by the Vision Transformer.
2. Patch Embedding
    - **Linear Transformation**: Each flattened patch undergoes a trainable linear transformation that maps it into a D-dimensional embedding space. This operation, which is akin to a fully connected layer, enriches the representation of each patch, making it more suitable for subsequent processing by the transformer.
    - **Spatial Information Encoding**: Since the transformer architecture does not inherently preserve the spatial arrangement of patches, additional spatial information is incorporated into the patch embeddings. This is achieved by adding positional encodings, which can be either fixed or learned parameters, to the embedded patches. These encodings ensure that the model retains information about the position of each patch within the original image.
3. Visual Transformer Encoder
    - **Self-Attention Mechanism**: At the heart of the transformer model lies the self-attention mechanism, which efficiently evaluates the relevance of different patches in relation to each other for a specific task. This mechanism determines the amount of focus each part of the image should receive when encoding a particular patch, enhancing the model's ability to understand complex relationships within the image.
    - **Multi-head Attention**: The encoder employs multi-head attention, running several self-attention mechanisms in parallel. This approach allows the model to simultaneously capture diverse aspects and patterns in the image, enriching the representation of each patch with a variety of contextual information.
    - **Feed-forward Network**: Following the attention computation, each patch's representation is processed through a feed-forward neural network (FFNN). This network is applied identically and independently to each position, further refining the encoded information and enabling the model to learn intricate features.
    - **Residual Connections and Normalization**: To ensure a stable training process and facilitate the accumulation of knowledge from each layer, the transformer encoder incorporates residual connections around each sub-layer. These are followed

by layer normalization, which helps maintain consistent gradients and improves the overall training dynamics, ensuring that each layer effectively builds upon the previous ones.

4. Creating the Output
   - **CLS Token**: To adapt transformer architecture for image classification, a dedicated classification token (CLS) is prefixed to the sequence of embedded patches. This token encapsulates comprehensive information aggregated from the entire image after undergoing transformation through the encoder. Utilized by a classifier, typically a straightforward linear layer, it facilitates accurate prediction of the image's class by leveraging enriched contextual understanding encoded within the transformer model.
5. Training
   - **Optimizing Model Parameters**: During training, the transformer model is fine-tuned end-to-end using an appropriate loss function, commonly cross-entropy for classification tasks. This function calculates the discrepancy between predicted and actual class labels, guiding the optimization of parameters across the linear projection, transformer encoder, and classifier. The objective is to systematically minimize this loss on a training dataset, ensuring the model effectively learns to classify images with high accuracy.

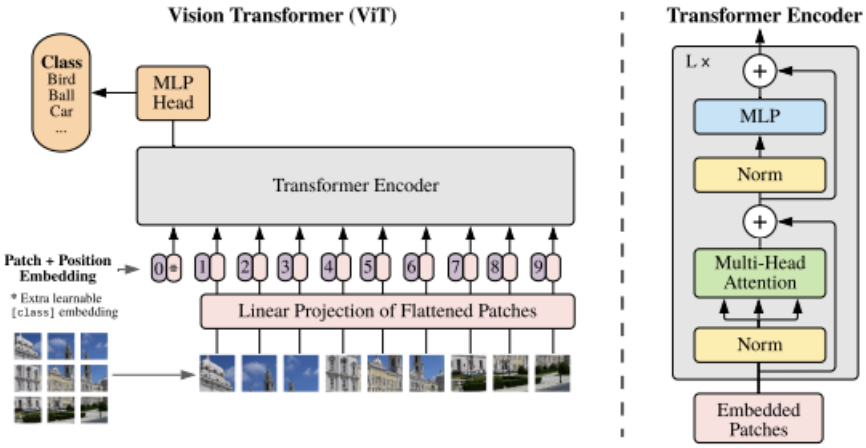

Figure 8: The Visual Transformer Neural Architecture [2, 3].

The transformation at each layer can be defined as [50]:

$$Z = X + \text{LayerNorm}(\text{MHA}(X, X, X)) \tag{16}$$

$$X' = Z + \text{LayerNorm}(\text{MLP}(Z)) \tag{17}$$

where $X \in \mathbb{R}^{N \times D}$ is a Transformer layer matrix input and $X' \in \mathbb{R}^{N \times D}$ is the output.

The attention mechanism is a vital component of the Transformer architecture, enabling the model to focus on specific parts of the input sequence when generating each output element. Given a query matrix $Q \in \mathbb{R}^{N \times D_k}$, a key matrix $K \in \mathbb{R}^{M \times D_k}$, and a value matrix $V \in \mathbb{R}^{M \times D_v}$, the attention function is defined as [50]:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{D_k}}\right)V \tag{18}$$

where D_k is the dimension of the keys, used as a scaling factor to prevent the dot products from becoming excessively large.

Self-attention is a specialized form of attention where the query, key, and value matrices are all derived from the same input matrix X. Within the Transformer, self-attention permits each position in the input sequence to attend to all other positions in the preceding layer.

Multi-head attention augments the attention mechanism, enabling the model to concurrently attend to information from diverse representation subspaces at different positions. Rather than executing a single attention function, multi-head attention projects the queries, keys, and values h times using different learned linear projections. These projections are processed in parallel, their results concatenated, and the concatenated output is then linearly projected once more. Mathematically, multi-head attention is expressed as [50]:

$$\text{MHA}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W^O \tag{19}$$

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) \tag{20}$$

Here, $W_i^Q \in \mathbb{R}^{D_x \times D_k}$, $W_i^K \in \mathbb{R}^{D_x \times D_k}$, $W_i^V \in \mathbb{R}^{D_x \times D_v}$, and $W^O \in \mathbb{R}^{hD_v \times D_x}$ are learnable parameter matrices.

3.3.5 Learning Rate Schedulers

When training deep networks, it's beneficial to gradually reduce the learning rate as the training progresses. This concept stems from the idea that a high learning rate imparts high kinetic energy to the model, causing its parameter vector to oscillate chaotically. Consequently, the model struggles to stabilize within the deeper and narrower valleys of the loss function, which represent local minima. Conversely, if the learning rate is too low, the system exhibits low kinetic energy, leading it to settle into the shallower and narrower valleys of the loss function, often getting stuck in suboptimal minima.

In essence, starting with a higher learning rate helps the model explore the loss landscape broadly, avoiding premature convergence. As training advances, lowering the learning rate allows the model to fine-tune its parameters and settle into more optimal solutions, thus achieving better performance. This dynamic adjustment helps in navigating the complex loss surfaces typical in deep learning, striking a balance between exploration and exploitation [80].

Examples of Learning Rate Scheduling [81]:

- ExponentialLR: Adjusts the learning rate by dividing it by a factor each epoch or evaluation period.
- CyclicLR: Oscillates the learning rate cyclically between specified minimum and maximum values.
- StepLR: Decreases the learning rate by multiplying it with a decay factor every fixed number of epochs or evaluation periods.
- MultiStepLR: Reduces the learning rate by a factor when the training reaches predefined milestones.
- ReduceLROnPlateau: Dynamically reduces the learning rate if a monitored metric fails to improve for a certain number of evaluations.
- CosineAnnealingLR: Starts with a high learning rate and smoothly decreases it following a cosine curve, periodically restarting the cycle.

3.3.6 Weight Decay

Weight decay, also known as L2 regularization, is an essential technique in deep learning to enhance model performance. By penalizing large weights, it offers several key benefits [82]:

- **Mitigates Overfitting:** Large weights often cause a model to memorize training data, hindering its ability to generalize to new examples. Weight decay discourages large weights, promoting the learning of smaller, more generalizable weights that capture the fundamental patterns in the data.
- **Enhances Model Stability:** Large weights can destabilize training and increase sensitivity to data noise. Weight decay stabilizes the training process by keeping weights manageable, thus enhancing the model's robustness and reducing overfitting.
- **Encourages Feature Sharing:** By promoting similar weights across different neurons, weight decay fosters feature sharing. This leads to a more efficient network where multiple neurons can utilize the same features, often resulting in fewer required parameters.
- **Boosts Generalization in Overparameterized Models:** Modern deep learning models often contain more parameters than training data, a scenario known as overparameterization. Weight decay helps manage the complexity of such models, improving their ability to generalize.

Weight decay adds a penalty to the loss function, proportional to the sum of the squared weights. This penalty encourages the model to prefer smaller weights during training. Implementation of weight decay typically follows one of two approaches:

- **L2 Regularization:** Adds a term to the loss function proportional to the sum of the squared weights.
- **Optimizer-based Weight Decay:** Alters the optimizer's update rule to include a decay factor that incrementally reduces the weights.

Choosing the right weight decay parameter involves balancing overfitting prevention with performance optimization. The optimal value varies based on model size, complexity, training data, and learning rate. Techniques such as grid search, hyperparameter optimization, and cross-validation are useful for identifying the best weight decay value.

3.3.7 Cross Validation

Cross-Validation (CV) is a cornerstone of evaluating learning models which provides the possibility to approximate model performance on unseen data not used while training. When training a model, the dataset is divided into two primary subsets: training and testing. The training set encompasses all the examples from which the model learns, while the testing set emulates real-world scenarios where the model performance is assessed [83].

Validating a learning model before deploying it in production involves ensuring it can make accurate predictions on data it has never encountered. This unseen data represents any type of information that the model hasn't been trained on. Ideally, during testing, this data flows directly into the model across numerous iterations. However, in practice, access to genuinely new data is often restricted or unavailable in a fresh environment.

A common approach is the 80-20 rule, where 80% of the data is used for training and 20% for testing. Despite its popularity, this method is prone to creating a seemingly perfect split that might artificially inflate model accuracy, while failing to replicate the same performance in real-world scenarios. The accuracy achieved in such cases can often be attributed to mere chance. It is worth noting that the 80-20 split is not a hard and fast rule; other ratios like 70-30 or 75-25 are frequently used as well.

Tuning model hyperparameters is another crucial step in optimizing an algorithm to uncover the hidden patterns within a dataset. However, performing this tuning on a simple training-testing split is generally discouraged. Model performance is highly sensitive to hyperparameters, and adjusting them based on a fixed data split can lead to overfitting, reducing the model generalizability [83].

To avoid overfitting during hyperparameter tuning, some suggest dividing the dataset into three parts: training, validation, and testing, for example, using 70% for training, 20% for validation, and 10% for testing. Yet, with small datasets, maximizing the data for training is often necessary. Moreover, such splits can introduce bias if important examples are disproportionately allocated to the training or validation sets. Cross-validation (CV) is a valuable technique to mitigate these issues by ensuring a more balanced evaluation.

The process of splitting data for training and testing can be fraught with difficulties. The testing set may not share similar properties with the training set, causing instability. Even though random sampling theoretically gives each sample an equal chance of being included in the testing set, a single split might still result in instability when the experiment is repeated with a different division.

3.3.8 Random Search

Random Search is a hyperparameter optimization technique in machine learning that samples a defined number of hyperparameter combinations from specified distributions at random. These distributions represent sets or ranges of possible parameter values [84].

This method efficiently navigates a subset of the hyperparameter space, making it faster and more resource-efficient compared to exhaustive techniques like grid search.

Although Random Search may not guarantee the discovery of the absolute optimal hyperparameters, it is highly effective at identifying good configurations, particularly in scenarios involving large or complex parameter spaces.

3.4 Ordinary Differential Equations (ODE)

TODO

3.4.1 Runge-Kutta Methods

TODO

3.4.2 ODE Interpretation of Transformer Layers

TODO

3.5 Quantum Computing

TODO

3.5.1 Quantum Gates

TODO

3.5.2 Quantum Circuits

TODO

3.5.3 Quantum Machine Learning

TODO

3.5.4 Variational Quantum Circuits (VQC)

TODO

3.5.5 Quantum Runge-Kutta Method

TODO

3.5.6 Quantum Transformer

TODO

4 Proposed Solution

5 Experiments

6 Conclusions and Future Work

7 Bibliography

[1] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[2] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[3] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[4] Xue-Wen Chen and Xiaotong Lin. Big data deep learning: challenges and perspectives. *IEEE access*, 2:514–525, 2014.

[5] Petro Liashchynskyi and Pavlo Liashchynskyi. Grid search, random search, genetic algorithm: a big comparison for nas. *arXiv preprint arXiv:1912.06059*, 2019.

[6] Riccardo Di Sipio, Jia-Hong Huang, Samuel Yen-Chi Chen, Stefano Mangini, and Marcel Worring. The dawn of quantum natural language processing. In *ICASSP 2022-2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 8612–8616. IEEE, 2022.

[7] Zipeng Fan, Jing Zhang, Peng Zhang, Qianxi Lin, and Hui Gao. Quantum-inspired neural network with runge-kutta method. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 17977–17984, 2024.

[8] Geoffrey E Hinton, Alex Krizhevsky, and Sida D Wang. Transforming auto-encoders. In *Artificial Neural Networks and Machine Learning–ICANN 2011: 21st International Conference on Artificial Neural Networks, Espoo, Finland, June 14-17, 2011, Proceedings, Part I 21*, pages 44–51. Springer, 2011.

[9] Liyuan Liu, Xiaodong Liu, Jianfeng Gao, Weizhu Chen, and Jiawei Han. Understanding the difficulty of training transformers. *arXiv preprint arXiv:2004.08249*, 2020.

[10] Marco Cerezo, Andrew Arrasmith, Ryan Babbush, Simon C Benjamin, Suguru Endo, Keisuke Fujii, Jarrod R McClean, Kosuke Mitarai, Xiao Yuan, Lukasz Cincio, et al. Variational quantum algorithms. *Nature Reviews Physics*, 3(9):625–644, 2021.

[11] Elena Voita, David Talbot, Fedor Moiseev, Rico Sennrich, and Ivan Titov. Analyzing multi-head self-attention: Specialized heads do the heavy lifting, the rest can be pruned. *arXiv preprint arXiv:1905.09418*, 2019.

[12] Marius-Constantin Popescu, Valentina E Balas, Liliana Perescu-Popescu, and Nikos Mastorakis. Multilayer perceptron and neural networks. *WSEAS Transactions on Circuits and Systems*, 8(7):579–588, 2009.

[13] Marcello Benedetti, Erika Lloyd, Stefan Sack, and Mattia Fiorentini. Parameterized quantum circuits as machine learning models. *Quantum Science and Technology*, 4(4):043001, 2019.

[14] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009.

[15] Yann LeCun, Corinna Cortes, and CJ Burges. Mnist handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, 2, 2010.

[16] Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 142–150, Portland, Oregon, USA, June 2011. Association for Computational Linguistics.

[17] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[18] Shi-Xin Zhang, Jonathan Allcock, Zhou-Quan Wan, Shuo Liu, Jiace Sun, Hao Yu, Xing-Han Yang, Jiezhong Qiu, Zhaofeng Ye, Yu-Qin Chen, Chee-Kong Lee, Yi-Cong Zheng, Shao-Kai Jian, Hong Yao, Chang-Yu Hsieh, and Shengyu Zhang. Tensorcircuit: a quantum software framework for the nisq era. *Quantum*, 7:912, February 2023.

[19] Jonathan Wei Zhong Lau, Kian Hwee Lim, Harshank Shrotriya, and Leong Chuan Kwek. Nisq computing: where are we and where do we go? *AAPPS bulletin*, 32(1):27, 2022.

[20] Michael A Nielsen and Isaac L Chuang. *Quantum computation and quantum information*, volume 2. Cambridge university press Cambridge, 2001.

[21] Hsin-Yuan Huang, Michael Broughton, Masoud Mohseni, Ryan Babbush, Sergio Boixo, Hartmut Neven, and Jarrod R McClean. Power of data in quantum machine learning. *Nature communications*, 12(1):2631, 2021.

[22] Yun He, Ziwei Zhu, Yin Zhang, Qin Chen, and James Caverlee. Infusing disease knowledge into bert for health question answering, medical inference and disease name recognition. *arXiv preprint arXiv:2010.03746*, 2020.

[23] Yuanzhi Li and Yang Yuan. Convergence analysis of two-layer neural networks with relu activation. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[24] Jin Xu, Zishan Li, Bowen Du, Miaomiao Zhang, and Jing Liu. Reluplex made more practical: Leaky relu. In *2020 IEEE Symposium on Computers and communications (ISCC)*, pages 1–7. IEEE, 2020.

[25] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016.

[26] Weiyang Liu, Yandong Wen, Zhiding Yu, and Meng Yang. Large-margin softmax loss for convolutional neural networks. *arXiv preprint arXiv:1612.02295*, 2016.

[27] Jun Han and Claudio Moraga. The influence of the sigmoid function parameters on the speed of backpropagation learning. In *International workshop on artificial neural networks*, pages 195–201. Springer, 1995.

[28] Babak Zamanlooy and Mitra Mirhassani. Efficient vlsi implementation of neural networks with hyperbolic tangent activation function. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 22(1):39–48, 2013.

[29] Zijun Zhang. Improved adam optimizer for deep neural networks. In *2018 IEEE/ACM 26th international symposium on quality of service (IWQoS)*, pages 1–2. Ieee, 2018.

[30] Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010: 19th International Conference on Computational StatisticsParis France, August 22-27, 2010 Keynote, Invited and Contributed Papers*, pages 177–186. Springer, 2010.

[31] Long Wen, Liang Gao, Xinyu Li, and Bing Zeng. Convolutional neural network with automatic learning rate scheduler for fault classification. *IEEE Transactions on Instrumentation and Measurement*, 70:1–12, 2021.

[32] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

[33] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[34] Ivan Georgiev Koprinkov. The quantum superposition principle: a reconsideration, 2023.

[35] Ryszard Horodecki, Paweł Horodecki, Michał Horodecki, and Karol Horodecki. Quantum entanglement. *Reviews of Modern Physics*, 81(2):865–942, June 2009.

[36] Hai-Long Shi, Si-Yuan Liu, Xiao-Hui Wang, Wen-Li Yang, Zhan-Ying Yang, and Heng Fan. Coherence depletion in the grover quantum search algorithm. *Physical Review A*, 95(3):032307, 2017.

[37] Thomas E O'Brien, Brian Tarasinski, and Barbara M Terhal. Quantum phase estimation of multiple eigenvalues for small-scale (noisy) experiments. *New Journal of Physics*, 21(2):023022, 2019.

[38] Yaakov S Weinstein, MA Pravia, EM Fortunato, Seth Lloyd, and David G Cory. Implementation of the quantum fourier transform. *Physical review letters*, 86(9):1889, 2001.

[39] Stephan Gulde, Mark Riebe, Gavin PT Lancaster, Christoph Becher, Jürgen Eschner, Hartmut Häffner, Ferdinand Schmidt-Kaler, Isaac L Chuang, and Rainer Blatt. Implementation of the deutsch–jozsa algorithm on an ion-trap quantum computer. *Nature*, 421(6918):48–50, 2003.

[40] Abolfazl Farahani, Sahar Voghoei, Khaled Rasheed, and Hamid R. Arabnia. A brief review of domain adaptation, 2020.

[41] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network, 2015.

[42] Daniel Berrar et al. Cross-validation., 2019.

[43] Mohamad Zaim Awang Pon and Krishna Prakash KK. Hyperparameter tuning of deep learning models in keras. *Sparklinglight Transactions on Artificial Intelligence and Quantum Computing (STAIQC)*, 1(1):36–40, 2021.

[44] Joseph O Ogutu, Torben Schulz-Streeck, and Hans-Peter Piepho. Genomic selection using regularized linear regression models: ridge regression, lasso, elastic net and their extensions. In *BMC proceedings*, volume 6, pages 1–6. Springer, 2012.

[45] Jose M Alvarez and Mathieu Salzmann. Learning the number of neurons in deep networks, 2018.

[46] Karl Weiss, Taghi M Khoshgoftaar, and DingDing Wang. A survey of transfer learning. *Journal of Big data*, 3:1–40, 2016.

[47] Stuart Hadfield, Zhihui Wang, Bryan O'gorman, Eleanor G Rieffel, Davide Venturelli, and Rupak Biswas. From the quantum approximate optimization algorithm to a quantum alternating operator ansatz. *Algorithms*, 12(2):34, 2019.

[48] Yaofeng Desmond Zhong, Tongtao Zhang, Amit Chakraborty, and Biswadip Dey. A neural ode interpretation of transformer layers. *arXiv preprint arXiv:2212.06011*, 2022.

[49] Nikolas P Breuckmann and Xiaotong Ni. Scalable neural network decoders for higher dimensional quantum codes. *Quantum*, 2:68, 2018.

[50] Marçal Comajoan Cara, Gopal Ramesh Dahale, Zhongtian Dong, Roy T. Forestano, Sergei Gleyzer, Daniel Justice, Kyoungchul Kong, Tom Magorsch, Konstantin T. Matchev, Katia Matcheva, and Eyup B. Unlu. Quantum vision transformers for quark-gluon classification. *Axioms*, 13(5):323, May 2024.

[51] Bei Li, Quan Du, Tao Zhou, Yi Jing, Shuhan Zhou, Xin Zeng, Tong Xiao, Jingbo Zhu, Xuebo Liu, and Min Zhang. Ode transformer: An ordinary differential equation-inspired model for sequence generation. *arXiv preprint arXiv:2203.09176*, 2022.

[52] John Charles Butcher. A history of runge-kutta methods. *Applied numerical mathematics*, 20(3):247–260, 1996.

[53] Shengyu Zhao, Zhijian Liu, Ji Lin, Jun-Yan Zhu, and Song Han. Differentiable augmentation for data-efficient gan training. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2020.

[54] Yifan Jiang, Shiyu Chang, and Zhangyang Wang. Transgan: Two pure transformers can make one strong gan, and that can scale up. *Advances in Neural Information Processing Systems*, 34:14745–14758, 2021.

[55] Jean-Baptiste Cordonnier, Andreas Loukas, and Martin Jaggi. On the relationship between self-attention and convolutional layers, 2020.

[56] Alan C. Santos. O computador quântico da ibm e o ibm quantum experience. *Revista Brasileira de Ensino de Física*, 39(1), September 2016.

[57] Gil Kalai, Yosef Rinott, and Tomer Shoham. Google's quantum supremacy claim: Data, documentation, and discussion, 2023.

[58] Mariia Mykhailova. Teaching quantum computing using microsoft quantum development kit and azure quantum. In *2023 IEEE International Conference on Quantum Computing and Engineering (QCE)*. IEEE, September 2023.

[59] Justin A. Reyes, Dan C. Marinescu, and Eduardo R. Mucciolo. Simulation of quantum many-body systems on amazon cloud. *Computer Physics Communications*, 261:107750, April 2021.

[60] Vedran Dunjko and Hans J Briegel. Machine learning & artificial intelligence in the quantum domain: a review of recent progress. *Reports on Progress in Physics*, 81(7):074001, 2018.

[61] Yagnik Chatterjee, Eric Bourreau, and Marko J Rančić. Solving various np-hard problems using exponentially fewer qubits on a quantum computer. *Physical Review A*, 109(5):052441, 2024.

[62] Martin Fürer. Solving np-complete problems with quantum search. In *Latin American Symposium on Theoretical Informatics*, pages 784–792. Springer, 2008.

[63] Amira Abbas, David Sutter, Christa Zoufal, Aurélien Lucchi, Alessio Figalli, and Stefan Woerner. The power of quantum neural networks. *Nature Computational Science*, 1(6):403–409, 2021.

[64] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[65] Juyong Kim, Yookoon Park, Gunhee Kim, and Sung Ju Hwang. SplitNet: Learning to semantically split deep networks for parameter reduction and model parallelization. In

Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1866–1874. PMLR, 06–11 Aug 2017.

[66] Basheer Qolomany, Majdi Maabreh, Ala Al-Fuqaha, Ajay Gupta, and Driss Benhaddou. Parameters optimization of deep learning models using particle swarm optimization. In *2017 13th International Wireless Communications and Mobile Computing Conference (IWCMC)*, pages 1285–1290, 2017.

[67] Chai Wah Wu. Prodsumnet: reducing model parameters in deep neural networks via product-of-sums matrix decompositions. *arXiv preprint arXiv:1809.02209*, 2018.

[68] Alireza Azadbakht, Saeed Reza Kheradpisheh, Ismail Khalfaoui-Hassani, and Timothée Masquelier. Drastically reducing the number of trainable parameters in deep cnns by inter-layer kernel-sharing. *arXiv preprint arXiv:2210.14151*, 2022.

[69] Adriano Barenco, Charles H Bennett, Richard Cleve, David P DiVincenzo, Norman Margolus, Peter Shor, Tycho Sleator, John A Smolin, and Harald Weinfurter. Elementary gates for quantum computation. *Physical review A*, 52(5):3457, 1995.

[70] Suhas Maddali. A brief overview of machine learning. `https://towardsdatascience.com/a-brief-overview-of-machine-learning-20abc68cbd4e`, 2022.

[71] Robert Kwiatkowski. Gradient descent algorithm. `https://towardsdatascience.com/gradient-descent-algorithm-a-deep-dive-cf04e8115f21`, 2021.

[72] Léon Bottou. Stochastic gradient descent tricks. In *Neural Networks: Tricks of the Trade: Second Edition*, pages 421–436. Springer, 2012.

[73] Fabio M. Graetz. Why adamw matters. `https://towardsdatascience.com/why-adamw-matters-736223f31b5d`, 2018.

[74] Anshu Mishra. Deep learning fundamental. `https://medium.datadriveninvestor.com/deep-learning-fundamental-important-concepts-59d7ae90901b`, 2019.

[75] Morjina akter. Artificial neural network(ann). `https://medium.com/@morjina/artificial-neural-network-ann-74eae97980ea`, 2023.

[76] baeldung. Multi-layer perceptron vs. deep neural network. `https://www.baeldung.com/cs/mlp-vs-dnn`, 2023.

[77] Marco Del Pra. Generative adversarial networks. `https://medium.com/@marcodelpra/generative-adversarial-networks-dba10e1b4424`, 2023.

[78] BRYON MOYER. Generative adversarial network (gan). `https://semiengineering.com/knowledge_centers/artificial-intelligence/neural-networks/generative-adversarial-network-gan/`, 2021.

[79] Tauseef Ahmad. Vision transformers. `https://medium.com/@tauseefahmad12/vision-transformers-4c6116f7f0ce`, 2024.

[80] Shreenidhi Sudhakar. Learning rate scheduler. https://towardsdatascience.com/learning-rate-scheduler-d8a55747dd90, 2017.

[81] Cloud Factory. Learning rate scheduler - overview. https://wiki.cloudfactory.com/docs/mp-wiki/scheduler/overview-of-learning-rate-schedulers-in-ml, 2024.

[82] Sujatha Mudadla. Weight decay in deep learning. https://medium.com/@sujathamudadla1213/weight-decay-in-deep-learning-8fb8b5dd825c, 2023.

[83] Mohammed Alhamid. What is cross-validation? https://towardsdatascience.com/what-is-cross-validation-60c01f9d9e75, 2020.

[84] M. Hammad Hassan. Random search. https://medium.com/@hammad.ai/tuning-model-hyperparameters-with-random-search-f4c1cc88f528, 2023.