BAI PROGRAMME

Introduction to AI, ML and Research 2024, Inaugural Cohort Proceedings

A programme led by
ThinkingBeyond Education

Proceedings of the BeyondAI: Introduction to AI and Research Programme

Authors:

Shaana Amarawickrama, Karuna Prakash, Osewuike Igue, Gulalayi Khankhel, Louis Vidal, Jack Woodhouse, Aditi Singh, Yonatan Yifat, Mena Zaied, Alazar Adane, Palak Kundu, Sumayah Adegbite, Adityan Srinivasan, Sadhana T. Kalidoss, Nayra Saadawy, Priyam Raul, Siddhart K. Gopal, Harini Muthurengan, Emeka Odiwe, Loulia Jaafari, Warenya K. Arachchillage, Ozodbek Eshmurodov, Arav Ajaykumar, Nafiul Haque, Shaurya Karmakar

Editors:

Filip Bár, Keisha Kwok, María Delgado Álvarez

December 2024

Preface

Welcome to the proceedings of the inaugural 2024 cohort of the *BeyondAI: Introduction to AI and Research* (BeyondAI) programme, organised by ThinkingBeyond Education from October 7th to December 6th, 2024.

This year, the programme brought together thirty exceptional young researchers, aged 14–22, from across the globe. Each participant was selected through a highly competitive process, standing out amongst 412 applicants from over 45 countries. To secure a place, candidates first had to complete the rigorous *Preparation Stage* as part of their application. Only those who demonstrated strong independent learning skills and mastery of the foundational topics progressed to the *Course Stage*, where they further honed their knowledge before embarking on their research projects under the guidance of an academic mentor during the *Research Stage*.

The *Preparation Stage*, spanning six weeks, formed the cornerstone of the selection process. Depending on their prior educational background, applicants engaged in self-directed study to build or reinforce their understanding of Linear Algebra, Calculus, Multivariable Calculus, Python programming, and LaTeXtypesetting, while also exploring fundamental AI concepts through curated resources. Only those who demonstrated sufficient proficiency in the basics across all required disciplines earned admission into the programme.

During the four-week *Course Stage*, participants explored Machine Learning from both research and engineering perspectives. The research-focused component emphasised conceptual understanding and mathematical modeling, introducing core ideas in a way accessible to newcomers. Meanwhile, the engineering component focused on the numerical implementation of these models, bridging the gap between theory and practical application.

Beyond building subject-specific knowledge and skills, the programme introduced participants to the world of academic research. A series of workshops equipped them with essential research skills, including scheduling and time management, effective learning strategies, personal knowledge management, project management, scientific writing, and research presentation. Participants worked individually and collaboratively, undertaking tasks such as writing mock research papers, designing research posters, and coding their own multilayer perceptrons (MLPs) from scratch.

Those who successfully completed the Course Stage advanced to the five-week-long *Research Stage*. Working in teams of two, participants delved into Machine Learning projects under the mentorship of experienced academics. Alongside their research, they attended expert-led talks on advanced topics and real-world applications, further broadening their understanding of the field.

Machine Learning and Artificial Intelligence are

rapidly evolving disciplines with the potential to transform industries and redefine problem-solving across domains. BeyondAI serves as a vital platform for nurturing young talent, providing them with a strong foundation in AI, exposure to academic research, and essential skills for intellectual growth. Our mission is to empower participants by fostering their expertise and curiosity, equipping them with the knowledge and experience necessary to excel as future subject experts, researchers and innovators.

We extend our deepest gratitude to the ThinkingBeyond team and the many volunteers whose dedication made this programme possible. Special thanks go to our academic mentors: Dr. Devendra Singh Dhami, Dr. Helena Bahrami, Dr. Filip Bár, Dr. Matej Cief, Adeyemi Damilare Adeoye, MSc, Emilie Gregoire, MSc, Barbora Barancikova, MSc, and Matthew Pugh, BE. Their guidance and support were invaluable in ensuring the success of our participants. We also extend our appreciation to the participants themselves, whose dedication and hard work form the foundation of these proceedings.

The research projects presented in this volume reflect a diverse range of topics and applications, from fundamental studies on Deep Learning optimisers and the phenomenon of Double Descent to explorations of advanced architectures such as Geometric Clifford Algebra Networks. These projects represent a significant leap beyond the Course Stage, posing considerable challenges—particularly for participants who were new to Machine Learning at the start of the programme.

We commend all participants for their perseverance and achievements in overcoming these challenges. As you explore these proceedings, we invite you to appreciate the depth of their intellectual efforts and contributions. We are proud to showcase their work and hope it serves as inspiration for aspiring researchers in STEM.[1] We are confident that these young minds will continue making meaningful contributions to the field and look forward to witnessing their future successes.

Chairs of BeyondAI

Dr. Filip Bár and Keisha Kwok

December 2024

[1]The code for each project can be found on https://github.com/ThinkingBeyond/BeyondAI-2024.

Contents

An Elementary Proof of the Universal Approximation Theorem for Multilayer Perceptrons

Paper

Github

Mentors: Mr. Matthew Pugh, Dr. Filip Bar

Shaana and Karuna

Introduction

The **universal approximation theorem** states that a **Multilayer Perceptron** with a finite number of neurons and a **single hidden layer** can approximate any **continuous function**. This project presents proof of a similar idea using elementary mathematics, for networks with multiple hidden layers by expanding on **Nielsen's visual proof** in his online textbook, *Neural Networks and Deep Learning*. The original proofs contain advanced mathematics that require a good deal of prerequisite knowledge to understand (Hahn-Banach theorem, Riesz Representation theorem, Fourier analysis', the Stone-Weierstrass theorem', to name a few). So, this project aims to present a more **intuitive** proof of the same using **elementary mathematics** and **visual elements**.

Breakdown of Nielsen's visual proof

Sigmoid function → Step function → Bump function → Tower function → Piecewise constant function → Approximate any continuous function!

(1) Sigmoid functions can approximate step functions

In an MLP, each neuron in each layer is connected to each neuron in the layer before and after it. Consider an **input neuron** connected to a **neuron in the hidden layer**, and call the **weight** of the connection w. We can use the $\epsilon - \delta$ definition of a **limit** and **pointwise convergence** to show that:

$$x > 0: \lim_{w\to\infty} \sigma(x) = \lim_{w\to\infty} \frac{1}{1+e^{-wx}} = 1$$

$$x < 0: \lim_{w\to\infty} \sigma(x) = \lim_{w\to\infty} \frac{1}{1+e^{-wx}} = 0$$

$$\lim_{w\to\infty} \sigma(wx) = \begin{cases} 0 & x < 0 \\ 1 & x > 0 \end{cases}$$

Fig. 1(a)

Fig. 1(b)

We can extend this idea to **n dimensions** by setting the **weights** of the other connections to **0**. The position of the step is at $s_k = -\frac{w_0}{w}$.

(2) Bump and tower functions

Bump functions can be constructed from step functions

Consider a MLP setup as Fig. 2, where $s_j(x)$ is a step function with the step at position $x = j$. The weighted output of the neuron in the second layer is:

Fig. 2

$$w_1 s_a(x) + w_2 s_b(x) = \begin{cases} w_1 + w_2 & x > b \\ w_1 & a < x < b \\ 0 & x < a \end{cases}$$

By changing the weights so each weight is the **additive inverse** of the other, like in Fig. 3, the weighted ouput of the neuron in the second hidden layer is:

Fig. 3

$$w_1 s_a(x) + w_2 s_b(x) = \begin{cases} 0 & x > b \\ h & a < x < b \\ 0 & x < a \end{cases}$$

This creates a **bump function** of height h. Similarly to the construction of step functions, we can extend this construction to **k dimensions** by using weights of zero. By adding multiple bump functions in different directions with height h, for a function with k inputs we get:

$$s(\vec{x}) = \begin{cases} 0 & x_1 > b_1 \\ h & a_1 < x_1 < b_1, a_2 < x_2 < b_2, \ldots, a_k < x_k < b_k \\ 2h & a_1 < x_1 < b_1, a_2 < x_2 < b_2, a_3 > b_3, \ldots, a_k > a_k \\ \vdots & \\ 2h & a_1 < x_1 < b_1, a_2 < x_2 < b_2, a_3 < x_3 < a_3, \ldots, a_k < a_k \\ h & a_1 < x_1 < b_1, a_2 < a_2, \ldots, a_k < a_k \\ 0 & x_1 < a_1, x_2 < a_2 \ldots, a_k < a_k \end{cases}$$

The maximum value is kh. By applying using a bias $-a_1 - (k-1)a_2 - \ldots - ka_k$ and increasing h and using a sigmoid activation function, we can create a **tower function** where there is only an non-zero output in one region of input.

(3) Proof of approximation using pointwise convergence

MLPs with sigmoid activation can approximate continuous functions

Proof for approximation

Since the constructions made from adding together bump and tower functions are **piecewise constant functions**, the next step in the proof is to show that a **piecewise constant function** can approximate a **continuous function**.

The piecewise constant function divides the input space into **equally sized regions**, and consists of a hyperplane in each region, since each input is divided into k **sections**.

In each section, the output of the **approximation function**, h, is $h(\vec{u}) = f(\vec{u})$ where $\vec{u}$ is a vector where each component is equal to the **lower bound of each input in the region**.

We need to show that the **limit** of h is f, as $k \to \infty$ which means we have to show that for $|f(\vec{x}) - h(\vec{x})| < \epsilon, \epsilon > 0$ we can find a K such that if $k > K$ then $|f(\vec{x}) - h(\vec{x})| < \epsilon$

$$f(\vec{x}) - h(\vec{x}) = f(\vec{x}) - f(\vec{u}) \implies |f(\vec{x}) - h(\vec{x})| = |f(\vec{x}) - f(\vec{u})|$$

Then, using the **definition of a continuous function**, we know that we can always make $|f(\vec{x}) - f(\vec{u})| < \epsilon$ if $|\vec{x} - \vec{u}| < \delta$.

Using the **Euclidean distance metric**, we can show that $|\vec{x} - \vec{u}| \leq \frac{h}{K}$. This tells us that we can always choose a K for which if $k > K$, $|\vec{x} - \vec{u}| < \delta$.

Therefore, we know that we can **increase the number of sections** for a **better approximation**, and have proved pointwise convergence.

Fig. 4

Fig. 5

Definition of a continuous function

Fig. 6

The $\epsilon - \delta$ definition of a continuous function is used in both proofs above. It uses the idea that as the one input approaches another, the limit of the output of the first input must equal the output of the second.

Scan for full video

MLPs with ReLU activation can approximate continuous functions

Construction of piecewise function

Affine combinations and simplices

An affine combination is a linear combination of $x_1, x_2, \ldots, x_n$

$\sum_{i=1}^{n} \alpha_i x_i = \alpha_1 x_1 + \alpha_2 x_2 + \ldots + \alpha_n x_n$ where $\sum_{i=1}^{n} \alpha_i = 1$

An affine combination of n+1 x_is where $0 \leq \alpha_i \leq 1$ gives us an n-simplex. A 0-simplex is a point, 1-simplex is a line, 2-simplex is a triangle, 3-simplex is a tetrahedron, and so on.

ReLU functions can form simplices

Fig. 8(a)

Fig. 8(b)

Made completely from triangular faces only

Fig. 8(c)

Piecewise approximation of a function $f : \mathbb{R}^n \to \mathbb{R}$

A piecewise approximation for a function can be made using affine combinations to form a function built with these simplexes.
A point x in $\mathbb{R}^n$ can then be represented as an affine combination.
The approximation function is thus given by $g(x) = \sum_{i=1}^{n+1} \alpha_i f(x_i)$

Proof for approximation

$$|f(x) - g(x)| = |\sum_{i=1}^{n} \alpha_i f(x) - \sum_{i=1}^{n} \alpha_i f(x_i)|$$
$$= |\sum_{i=1}^{n} \alpha_i (f(x) - f(x_i))|$$
$$\leq \sum_{i=1}^{n} \alpha_i |f(x) - f(x_i)|$$

With some algebra, we get:

$$|f(x) - g(x)| \leq |f(x) - c|$$

Here $c = f(x_j)$, where j is the value of i that results in the largest value of $|f(x) - f(x_i)|$.

By using the fact that f(x) is continuous, we can prove pointwise convergence.
We choose a $k < \delta$ then we have . This ensures that for all $\epsilon > 0$,

$$|f(x) - g(x)| = |f(x) - f(x_j)| < \epsilon.$$

Since each of the components in the m-dimensional output vector is a 1-dimensional output, so any m-dimensional output vector is simply m single output functions stacked. Therefore, since we can approximate each of the m components, we can approximate a function $f : \mathbb{R}^n \to \mathbb{R}^m$.

How are MLPs functions?

Weighted inputs → Activation function → Output

$$g(x) = f(w_l a_{l-1} + b)$$

Final Output of the MLP | Activation function | Weight matrix | Activation of previous layer | Bias vector

Recursive!

(4) Applications and Limitations

- The universal approximation theorem shows that for any problem where a continuous function is applied on some data, we can be **certain that there exists a neural network** that can be used.
  - There are **many problems** which use **continuous functions** so this is very valuable.
- However, the universal approximation theorem does **not provide a method** for finding the correct weights, biases and configuration for the neural network for it to approximate a function[4].
- To improve the approximation, both our proof and the original proofs show that the **number of neurons in the network must increase**. This will increase the **computational cost and memory** needed leading to the model being constrained by factors such as hardware and time.
- A limitation of our proof in particular is our method of **calculating each limit separately**, which would give us a higher error than expected. In the future, we would like to investigate this further and edit our proof accordingly.

References

1. M. Nielsen (2019) 'Neural Networks and Deep Learning'
2. G. Cybenko (1989) 'Approximation by superpositions of a sigmoidal function'
3. K. Hornik et al. (1989) 'Multilayer feedforward networks are universal approximators'
4. EITCA (2024) 'What is the universal approximation theorem, and what implications does it have for the design and capabilities of neural networks?'

Students: Osewuike Igue, Gulalaiy Khankhel

Weight Initialization for MLPs

Mentor: Barbora Barancikova

1. ABSTRACT

This project investigates how weight initialization techniques affect the **training dynamics and performance of MLPs**, focusing on **convergence**, **gradient stability**, and **accuracy**. We also explore their interaction with **activation functions**, **optimizers**, and **model depth** to determine optimal strategies.

2. METHODOLOGY

We ran **3 main experiments** to analyze the performance of Weight Initialization Methods including **Zero Initialization, Random (Gaussian) Initialization, Xavier Initialization,** and **He Initialization** ([2], [3]).

- The effect of different activations (ReLU, Tanh, Sigmoid) on the performance of different weight initialization methods.
- The effect of different optimizers (SGD, Adam) on the performance of different weight initialization methods.
- The effect of MLP depth on weight initialization performance in a shallow and deep architecture.

The **MNIST Handwritten Digit Dataset** was normalized and split into training and testing sets. Models were trained with a **learning rate** of **0.01, batch size** of **150**, and **Cross Entropy Loss**. For **Activation and Optimizer Experiments**, we used a **2-hidden-layer MLP**, and for the **MLP Depth Experiment**, we compared **1 vs 6** Hidden layers. The Main Performance metrics included **accuracy** and **convergence speed**, with results visualized through graphs **for test/training loss over epochs, test accuracy over epochs, weight distribution,** and **activation distribution.**

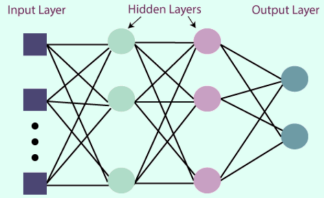
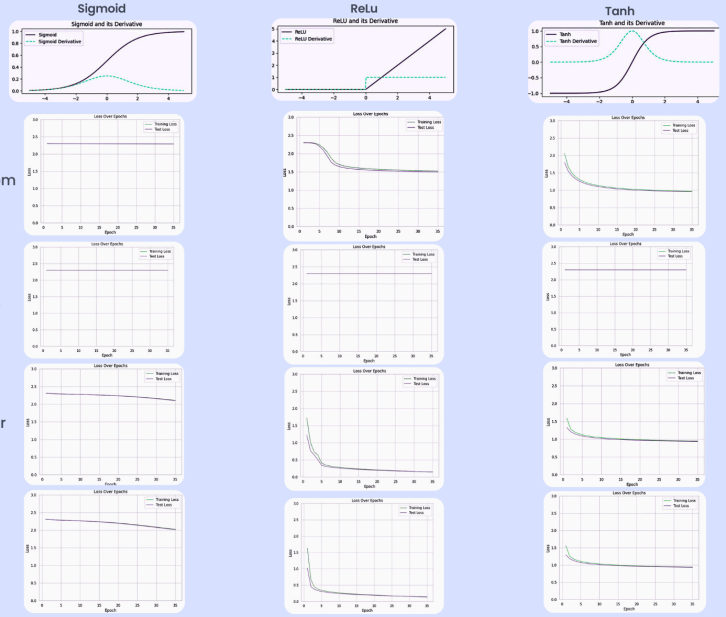


Fig. 1 - Example MLP Architecture with 2 hidden layers

$$\mathbf{h}^{(l+1)} = \sigma\left(\mathbf{W}^{(l)}\mathbf{h}^{(l)} + \mathbf{b}^{(l)}\right)$$

3. RESULTS

Activation Functions Experiment

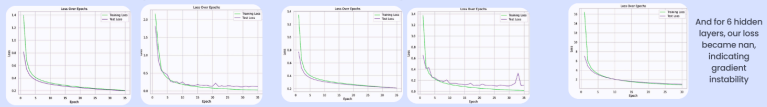


Optimizers Experiment

SGD Update Equation: $x - \eta \cdot \text{grad}$

Adam Update Equation: $x - \frac{\eta \cdot m_t}{\sqrt{v_t} + \epsilon}$

Varying MLP Depth Experiment



4. CONCLUSION

Our research shows that **Xavier** and **He Initialization** perform well across various setups: **Xavier** suits **tanh** and **sigmoid**, while **He** is ideal for **ReLU. Random Initialization** works for shallow models but risks **gradient issues** in deeper ones, while **Zero Initialization** fails entirely as it prevents learning. **He initialization** excels by maintaining gradient flow and avoiding inactive neurons hence ensuring **faster convergence with ReLU. Xavier** balances **gradient propagation** in deeper networks leading to better stability and test accuracy. In conclusion, **Xavier** and **He** are the best choices, **Random** is viable for simple models, and **Zero** should be avoided ([1], [2], [3]).

5. FUTURE RESEARCH

Future research on this topic could explore **additional weight initialization techniques**, such as **LeCun** or **Orthogonal.** Moreover, our findings could be extended to more **complex tasks and diverse datasets** to test their **generalisability** and provide more insight into **how different initializations interact with various architectures.** This could help refine best practices across a wider range of applications ([1], [3]).

Check out our code!

References:

- **[1]** Haykin, S. S. (2009). Neural networks and learning machines. Pearson Education.
- **[2]** Glorot, X., & Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks.
- **[3]** He, K., Zhang, X., Ren, S., & Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification.
- **Fig. 1** from https://www.javatpoint.com/multi-layer-perceptron-in-tensorflow.

How do Transformers compare to LSTMs with attention in performance and efficiency for text-based sentiment analysis?

Authors: Louis Vidal, Jack Woodhouse
Mentor: Matej Cief

Introduction

This study compares the performance and efficiency of Transformer networks and LSTMs with attention mechanisms on the IMDb reviews dataset, a large collection of movie reviews commonly used for sentiment analysis.

Objective

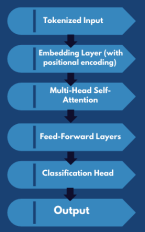
- Compare the performance of LSTMs with Attention and Transformers for sentiment analysis.
- Evaluate their efficiency regarding training time, inference speed, and resource usage.

Dataset IMDb

The IMDb reviews dataset consists of 50,000 movie reviews, with an even split of positive and negative labels. So this is perfect for training and testing binary classification with sentiment analysis.

Motivation

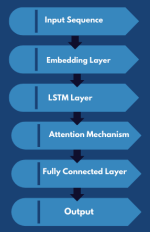
Sentiment analysis plays a critical role in analyzing consumer opinions and public sentiment. The IMDb dataset, consisting of 50,000 balanced positive and negative movie reviews, is ideal for testing binary sentiment classification models. Comparing LSTM and Transformer models helps identify the best approach for various real-world applications, considering both performance and computational constraints.



Transformer

Transformers use self-attention to process all words simultaneously, capturing relationships regardless of distance. This parallelized approach ensures faster training and inference. Multi-head self-attention identifies key sentiment phrases like 'highly recommend' or 'waste of time,' enhancing predictions.

Key Features:
- Parallel processing for scalability.
- Multi-head attention extracts diverse word relationships.



LSTM with Attention

LSTMs (Long Short-Term Memory) process sequences step-by-step, capturing context over time. Attention enhances this by focusing on key sentiment words like 'amazing' or 'disappointing,' improving predictions for long reviews.

Key Features:
- Sequential processing maintains context.
- Attention highlights crucial words.
- Ideal for smaller datasets or limited resources.

Overview

This study compares LSTMs with Attention and Transformers for sentiment analysis on the IMDb reviews dataset. Both models were trained from scratch to classify reviews as positive or negative, ensuring a fair comparison. The process included data preprocessing, model training, and performance analysis.

LSTM with attention

Transformer

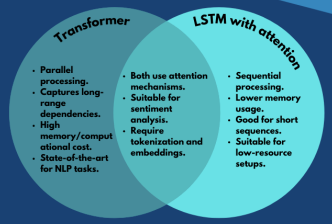
Introduction
Methodology
Results
Discussion

Abstract

This study evaluates the performance and efficiency of Transformers and LSTMs with attention mechanisms on the IMDb reviews dataset for sentiment analysis. We compare the models based on accuracy, training speed, and resource requirements, providing insights into their strengths and limitations

Conclusion

Transformers generally outperform LSTMs with Attention in sentiment analysis tasks, excelling in both performance and efficiency. This makes them ideal for larger datasets and faster processing. However, LSTMs with Attention remain a viable alternative for scenarios with limited computational resources or a need for smaller models. Future research could explore hybrid models that combine the strengths of both architectures or investigate the application of Transformers to smaller, less structured datasets.



Efficiency

- Training Time: Transformers require more GPU resources and training time due to their larger number of parameters and complex attention layers.
- Inference Speed: Despite their computational intensity, Transformers process input faster than LSTMs during inference, thanks to parallelized architecture.
- Memory Usage: LSTMs with Attention use significantly less memory, making them more suitable for deployment in resource-constrained environments.

Performance

- Overall Performance: Transformers outperform LSTMs with Attention across key metrics.
- Self-Attention Mechanism: Allows Transformers to capture long-range dependencies and subtle text patterns more effectively.
- Accuracy: Higher accuracy, especially for longer reviews where LSTMs may lose context due to sequential processing.
- Precision & Recall: Superior precision and recall, particularly for detecting subtle sentiments in complex sentences.
- F1-Score: Higher F1-scores indicate a balanced and reliable performance in real-world sentiment analysis tasks.

Limitations

- LSTM with Attention: A viable choice for smaller datasets, shorter texts, or environments with limited computational power.
- Transformer: Ideal for large-scale datasets, long-text analysis, or applications requiring high accuracy and scalability.
- However, Transformers' computational demands may limit their applicability in real-time or low-power scenarios.

Performance Metrics Comparison

The performance of both models on the IMDb dataset was evaluated using key metrics: Accuracy, Precision, Recall, F1-Score, and ROC-AUC. The Transformer outperformed the LSTM with Attention in most metrics, showcasing its superior ability to handle long-range dependencies and complex sentence structures.

Metric	Transformer	LSTM with Attention
Training Time (5 epochs)	~6 minutes	~5 minutes
Memory Usage	2.5 GB (training)	1.5 GB (training)
Inference Speed	12 ms/example	20 ms/example

Training and Efficiency Metrics

In terms of efficiency, the Transformer required approximately 6 minutes to train across 5 epochs. Its self-attention mechanism enables parallel processing, resulting in faster inference times compared to the sequential nature of LSTMs. However, the Transformer's larger architecture incurs higher computational costs.

Vaswani, A., et al. (2017). Attention Is All You Need. (Reference: https://arxiv.org/abs/1706.03762)
Bahdanau, D., et al. (2015). Neural Machine Translation by Jointly (Reference: https://arxiv.org/abs/1409.0473)

A Comparative Analysis of Optimizers for Classification with CNN

BeyondAI 2024: Project 9B
A. Singh, Y. Yifat; Mentor- Mr. Adeyemi D. Adeoye

I. INTRODUCTION

Optimization is a crucial aspect of Deep Neural Networks (DNNs). During training, an optimizer adjusts the weights and biases of the DNN to minimize a loss function.

$$\underset{\theta}{\text{minimize}}\ f(\theta) = \sum_i \ell(\hat{y}(\theta; x^i), y^i)$$

where $\boldsymbol{\theta}$ is the vectorized model parameters.

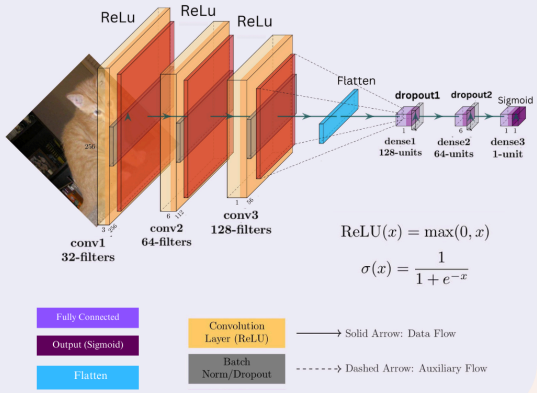
An optimization algorithm may perform better under one setting than another. Therefore, a comparative study of different algorithms is important for identifying an optimal setup for different algorithms for a particular task and model architecture. In this study, we aim to evaluate the performance of the six popular optimizers (SGD, Adam, AdaGrad, AdaDelta, RMSprop and Nadam) on a classification task using a Convolutional Neural Network (CNN) architecture. These six optimizers have been chosen because they are widely used in Deep Neural Networks. Each optimizer has its characteristic strengths and limitations, and oftentimes, one addresses the limitations of another. The choice of an optimizer directly influences the generalization properties and robustness of the final trained model, and hence it is important to choose an optimizer which is not only computationally efficient but also enhances the model performance.

II. PERFORMANCE COMPARISON & METRICS

The performance was assessed using four key evaluation metrics: **accuracy, precision, recall and F1-score**. Accuracy measured the overall correctness of predictions. Precision evaluated the proportion of true positives among predicted positives. Recall determined the proportion of true positives correctly identified. The F1-score balanced precision and recall. These metrics collectively provided a comprehensive evaluation of each optimizer's effectiveness in classification tasks. Experiments were conducted to analyze the effect of learning rate (for SGD) and batch size on model performance.

$$\text{Binary Cross-Entropy (BCE)} = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i)]$$

IV. MODEL ARCHITECTURE



III. OPTIMIZATION ALGORITHMS

Optimizer	Description	Update Formula
Adam	Combines momentum and adaptive learning rates for fast and robust convergence which is ideal for NLP, computer vision and large datasets. However, it tends to overfit smaller datasets and has higher computational costs.	$\theta_t = \theta_{t-1} - \frac{\eta}{\sqrt{\hat{v}_t}+\epsilon}\hat{m}_t, \quad \hat{m}_t = \frac{m_t}{1-\beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1-\beta_2^t}$ $m_t = \beta_1 m_{t-1} + (1-\beta_1)g_t, \quad v_t = \beta_2 v_{t-1} + (1-\beta_2)g_t^2$
Nadam	Adapts learning rates based on recent gradient magnitudes that makes it effective for non-stationary data, especially in RNNs and reinforcement learning. It is sensitive to hyperparameters.	$\theta_t = \theta_{t-1} - \frac{\eta}{\sqrt{\hat{v}_t}+\epsilon}\left(\beta_1\hat{m}_t + \frac{(1-\beta_1)g_t}{1-\beta_1^t}\right)$ $\hat{m}_t, \hat{v}_t, m_t, v_t$ as in Adam
RMSprop	Introduces Nesterov momentum for faster convergence, commonly applied in computer vision and speech recognition. Despite its speed, it requires careful hyperparameter tuning and is computationally intensive.	$\theta_t = \theta_{t-1} - \frac{\eta}{\sqrt{v_t}+\epsilon}g_t, \quad v_t = \beta v_{t-1} + (1-\beta)g_t^2$
AdaDelta	Addresses the decay issue in AdaGrad by normalizing updates with a moving average of gradients, making it suitable for RNNs and LSTMs, albeit with complexity and instability for non-convex data.	$\Delta\theta_t = -\frac{\sqrt{s_{t-1}+\epsilon}}{\sqrt{v_t+\epsilon}}g_t, \quad v_t = \gamma v_{t-1} + (1-\gamma)g_t^2,$ $s_t = \gamma s_{t-1} + (1-\gamma)(\Delta\theta_t)^2$
SGD	Uses mini-batches for gradient descent, offering simplicity and efficiency. It converges to the global minimum for convex functions but may oscillate on non-convex ones. Commonly used in linear regression and deep neural networks.	$\theta_t = \theta_{t-1} - \eta g_t$
AdaGrad	Adjusts the learning rate based on the historical sum of squared gradients, making it effective for sparse data like NLP but prone to slowing down over time.	$\theta_t = \theta_{t-1} - \frac{\eta}{\sqrt{G_t}+\epsilon}g_t, \quad G_t = G_{t-1} + g_t^2$

V. RESULTS

As shown in figure 4.2, AdaGrad achieved the highest accuracy of 79.09% and F1-score of 79.33%. The highest precision was achieved by Adam which was 84.23%. Nadam excelled in recall with 81.62%. It was found that SGD achieved highest accuracy with 0.01 learning rate. In figure 4.3, it was observed that AdaGrad demonstrated high accuracy across various batch sizes, with the best performance (78.15%) at the batch size of 32. RMSprop and Nadam performed better at higher batch sizes (512), with accuracies of 79.58% and 78.6%, respectively. Adam peaked with a batch size of 128, achieving an accuracy of 78.0% which showed its adaptability to different batch sizes. AdaDelta maintained stable but relatively lower performance across all batch sizes.

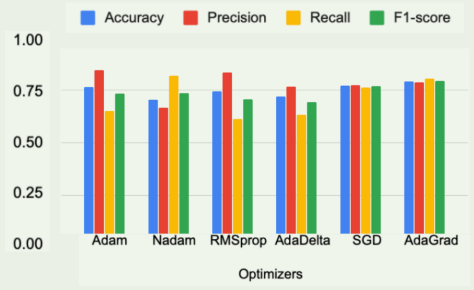


Figure 4.2: Comparison of Optimizers based on the performance metrics

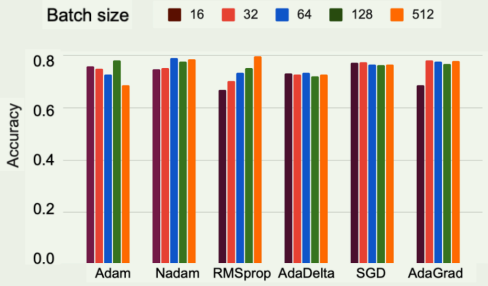


Figure 4.3: Comparison of Optimizers based on the batch sizes

Access our paper with References

View Code

Non- Linear Classifiers

Authors: Palak Kundu, Sumayah Adegbite
Mentor: Miss Barbora Barancikova

INTRODUCTION

Linear Classifiers → Struggle with Non-linear Data

Non-linear Classifiers → Evaluate Accuracy → Evaluate Complexity → Evaluate Interpretability

Compare on Different Datasets → Guide Selection of Best Method for Complex Data

METHODOLOGY

We evaluated classifier performance on three diverse datasets, each with varying complexity and feature types. The datasets were split into 70% training and 30% testing sets, ensuring that training data was used for model training, while testing data evaluated the model's generalization.

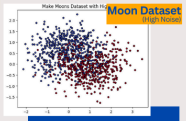
CLASSIFIERS EVALUATED

Advanced Classifier
- Support Vector Machine (SVM) with Radial Basis Function (RBF) Kernel
- Decision Tree
- Random Forest
- Gradient Boosting

Baseline Classifier
- Logistic Regression
- Naive Bayes

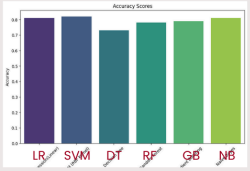
We used multiple performance metrics to assess the effectiveness of each classifier:

Accuracy | Precision | Recall | F1-Score | Training Time



synthetic dataset generated using make_moons from the sklearn.datasets library. It contains two interleaving crescent shapes with added noise.

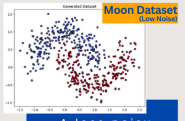
Accuracy Scores



Logistic Regression (81%) performs reasonably well but still struggles with the nature of the data. SVM(82%) is the top performer, slightly better than Logistic Regression, as it captures non-linear patterns

. Random Forest (78%) and Gradient Boosting (79%) show decent performance, but the noise reduces their ability to generalize. Decision Tree (73%) is the weakest, as it overfits to noisy data. Naive Bayes (81%) also performs similarly to Logistic Regression, but its assumptions limit its ability to handle noise effectively.

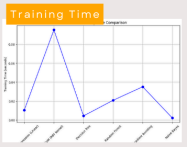
Metric	LR	SVM	DT	RF	GB	NB
P(Class 0)	0.84	0.82	0.76	0.78	0.80	0.84
P(Class 1)	0.78	0.81	0.71	0.78	0.79	0.79
R(Class 0)	0.79	0.83	0.71	0.80	0.81	0.79
R(Class 1)	0.83	0.81	0.75	0.76	0.78	0.84
F1(Class 0)	0.81	0.82	0.73	0.79	0.80	0.81
F1(Class 1)	0.81	0.81	0.73	0.77	0.78	0.81
Accuracy	0.81	0.82	0.73	0.78	0.79	0.81
Macro Avg	0.81	0.82	0.73	0.78	0.79	0.81
Weighted Avg	0.81	0.82	0.73	0.78	0.79	0.81



A less noisy variant of the Moon dataset for comparative analysis. Generated with make_moons by setting noise=0.2.

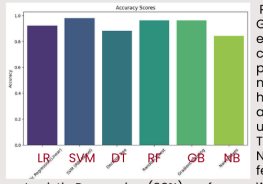
Logistic Regression (83%) struggles with non-linear patterns in the Moon dataset. SVM (95%) excels due to its ability to model complex, non-linear relationships.

Decision Tree (90%) performs well but may overfit, while Random Forest (94%) and Gradient Boosting (93%) provide strong results through ensemble learning. Naive Bayes (78%) underperforms due to its assumption of feature independence, which doesn't hold in this dataset.



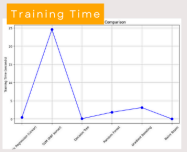
Classifier	Accuracy	F1 Score	Recall	Precision
Logistic Regression	0.89	0.88	0.89	0.88
SVM with RBF Kernel	0.94	0.94	0.94	0.94
Decision Tree	0.85	0.84	0.85	0.84
Random Forest	0.92	0.92	0.92	0.92
Gradient Boosting	0.93	0.93	0.93	0.93
Naive Bayes	0.82	0.81	0.82	0.82

A classic dataset of 28x28 grayscale handwritten digits (0-9), widely used for evaluating machine learning models



Logistic Regression (92%) performs decently but struggles with complex, non-linear patterns in MNIST data. SVM (98%) and ensemble methods like Random Forest and Gradient Boosting.

Random Forest and Gradient Boosting (96%) excel due to their ability to capture complex data patterns and can handle non-linearity . On the other hand, Decision Tree (88%) and Naive Bayes (84%) underperform, as Decision Tree tends to overfit, and Naive Bayes assumes feature independence, which doesn't hold for image data.



Classifier	Accuracy	F1 Score	Recall	Precision
Logistic Regression	0.92	0.92	0.92	0.92
SVM with RBF Kernel	0.98	0.98	0.98	0.98
Decision Tree	0.88	0.87	0.88	0.87
Random Forest	0.96	0.96	0.96	0.96
Gradient Boosting	0.96	0.96	0.96	0.96
Naive Bayes	0.84	0.84	0.84	0.84

The bar chart and line graph represent the accuracy scores and training time of the datasets shown in the image beside them, respectively. Each of the bars and points in order represents (LR)Logistic Regression (Linear), SVM (RBF kernel), (DT)Decision Tree, (RF) Random Forest, (GB) Gradient Boosting, and (NB) Naive Bayes.

NOTE: P- Precision , R- Recall, F1- F1 Score

CONCLUSION

- Logistic Regression and Naive Bayes are faster but less accurate.
- Random Forest and Gradient Boosting are effective for noisy or imbalanced datasets, balancing accuracy and robustness while reducing overfitting, though they are more computationally expensive than simpler classifiers

FUTURE WORK

- Evaluation on More Diverse Datasets
- Trying Out Other Types of Classifiers such as neural networks

REFERENCES

- Cortes, C., & Vapnik, V. (1995). Support-vector networks. Machine learning, 20(3), 273-297
- Breiman, L. (2001). Random forests. Machine learning, 45(1), 5-32.
- Murphy, K. P. (2012). Machine learning a probabilistic perspective. MIT pres

Check Our Code Here

Cooperative Control of Multi-Agent Systems: An Optimal and Robust Perspective

BeyondAI, Thinking Beyond 8B
Mr Matej Cief
Menna Zaied and Alazar Adane

Introdution

The coordination of multiple autonomous agents, such as unmanned aerial vehicles (UAVs), in dynamic and complex environments, has become a critical challenge in modern robotics. UAV swarms are increasingly being utilized for a variety of applications, including search and rescue operations, environmental monitoring, and infrastructure inspection. However, achieving effective collaboration among multiple UAVs while maintaining efficiency, avoiding collisions, and adapting to unpredictable environmental conditions remains a complex task.

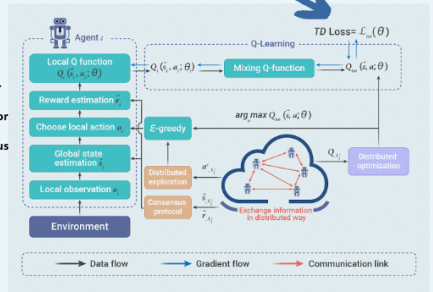
Inspired by coordinated behaviors observed in nature, such as flocking in birds and schooling in fish, the study of multi-agent systems (MAS) has grown in prominence. These natural systems exhibit remarkable levels of cooperation, where individual agents follow simple rules yet achieve globally optimal behaviors. The study of multi-agent coordination, particularly in mobile autonomous systems, has emerged as a key area of research in fields such as control theory, robotics, and optimization.

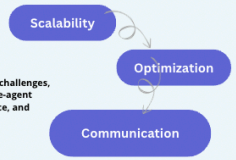
The problem statemant

In Multi-agent reinforcement learning, agents exist in different scenarios such as working collaboratively to tackle a given problem, competing with each other to get an optimal reward, or doing both works together or competing for each other like in football. In these processes of learning we have various problems in order to train how they work effectively and efficiently

When we work in real-time continuous environments such as self-driving car require immediate responses to change of environment. This require a huge amount of sensor data analysis to execute the best action in a given state.

As the number of agents in a system increases, managing their interactions grows exponentially. This results in communication network overload and loss of performance leads to suboptimal outcomes.



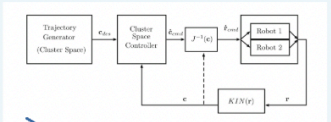


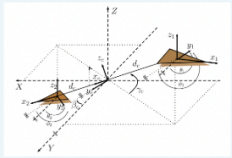
The coordination of drones in dynamic and cluttered environments presents several challenges, especially when tasks require collaboration among multiple agents. Traditional single-agent systems fail to meet the complex demands of real-time navigation, obstacle avoidance, and collaborative task execution.

Methodology

Cluster Space Control Framework

The method relies on the definition of a vector of CS variables that describes the position, orientation, and shape of a virtual kinematic mechanism representing the formation, allowing for full specification and control of the system. For a group of N robots with m degrees of freedom (DOF) each, the framework defines a set of kinematic transforms c = KIN(r) –where r is the vector of (N × m) robot space variables, (i.e., position and orientation of the robots), and c is a vector of (N × m) cluster space variables– as well as a set of inverse kinematic transforms r = INVKIN(c). Additionally, relations between velocities in both space.





Geometric Coordination & Optimal Control

Geometric Coordination

Geometric principles such as similarity transformations and manifold optimization provide a robust foundation for multi-agent coordination. These methods align individual agents' trajectories and maintain formations through precise transformations in Euclidean and Riemannian spaces. For multi-agent systems (MAS), geometric coordination ensures that each agent adjusts its position in relation to others, maintaining formation and avoiding collisions during navigation.

Key Formula:

$$T = \{R, t\}, \quad R \in SO(n), \quad t \in \mathbb{R}^n$$

Where:

- T: Transformation matrix (rotation R and translation t)
- $SO(n)$: Special orthogonal group ensuring valid rotations

Optimal Control

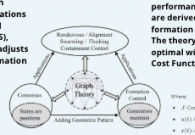
Optimal control minimizes a defined cost function, ensuring efficient resource use and high-performance task execution. In MAS, control laws are derived to balance energy consumption, formation maintenance, and collision avoidance. The theory ensures that each drone's trajectory is optimal with respect to the task requirements.

Cost Function:

$$J = \int_0^t (\|u(\tau)\|^2 + \|x(\tau) - x_d(\tau)\|^2)\, d\tau$$

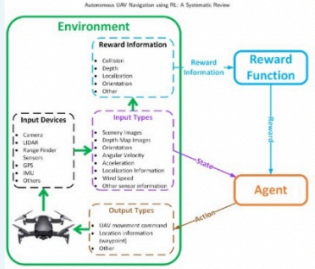
Where:

- J: Cost function
- $u(t)$: Control input
- $x(t)$: Actual trajectory
- $x_d(t)$: Desired trajectory



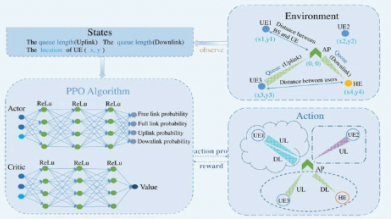
Integration of RL Framework in UAV Navigation

This diagram illustrates the overall framework for integrating Reinforcement Learning (RL) into UAV navigation, highlighting the interaction between the environment, the agent, and the reward function. It provides a structured approach to ensure that the UAV effectively learns and performs tasks such as collision avoidance, localization, and waypoint navigation.



PPO Algorithim

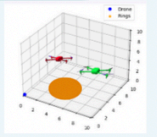
PPO is related to policy optimization where strategy determines the agent's actions in a given state through training, the policy update gradually and this ensures stability in the policy function.
One of the main advantages of using PPO is its effectiveness in continuous action space such as autonomous vehicles. PPO uses a stochastic policy, which supports exploration, while the ensures the exploitation of learned knowledge which doesn't lead to large amount of changes

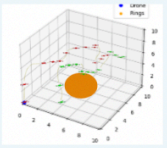


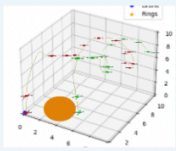
$$L^{PG}(\theta) = E_t[\log \pi_\theta(a_t|s_t) * A_t]$$

Simulation & results

This study explores the integration of reinforcement learning (RL) and geometric methods to address these challenges, focusing on enabling real-time obstacle avoidance and efficient task scheduling in UAV swarms. By leveraging the strengths of MAS and advanced mathematical techniques, we propose a framework that optimizes resource utilization and ensures seamless coordination in dynamic environments.







Our results

Drone Navigation Results Using PPO This table highlights key performance metrics from drone navigation experiments using Proximal Policy Optimization (PPO). Metrics such as FPS, timesteps, and policy optimization values demonstrate iterative improvements in navigation efficiency and stability in a dynamic 3D environment.

Scenario		Training loss		Testing loss	
Training set	Testing set	L_1	cGAN	L_1	cGAN
simulated	simulated	0.106	0.666	1.114	0.711
simulated (same training as previous case)	real-world	0.106	0.666	2.779	0.738
simulated + real-world	real-world	0.135	0.692	1.792	0.695

Reference

- Zhang, W., & Miele, A. (2020). Geometric Methods for Multi-Agent Systems: Optimal Control and Coordination. Springer.
- Shishika, R., & Choi, H. (2017). On the Optimal Control of Multi-Agent Systems: A Framework for Coordination and Communication. arXiv.
- Chen, X., & Zhang, Y. (2023). An Overview of Coordination Techniques for Multi-Agent Systems. Electronics, 6(9), 227.
- Sutton, R. S., & Barto, A. G. (2018). Reinforcement learning: An introduction. MIT Press.
- Leng, P., Fan, T., Liao, X., Liu, W., Zhang, H., & Pan, J. (2020). Towards optimally decentralized multi-robot collision avoidance via deep reinforcement learning. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (pp. 6252-6261).

If you are interested check our githup

GNNs for Anomaly Detection in Satellite Trajectories

Abstract

We implement a Spatio-Temporal Graph Neural Network (SpatioTemporalGNN) to understand spatio-temporal relationships between nodes in a dynamic graph. To realise DGNNs, we utilise satellite trajectories and the prediction of potential collisions as edge features. By utilising Two-Line Element (TLE) data, we model satellites as nodes in a dynamic graph and encode their relative spatial interactions as edge features. Our approach demonstrates the capability of graph-based learning to capture spatiotemporal dependencies, enabling the prediction of anomalies or in our application, impending satellite collisions.

Dynamic Graph Neural Networks

Graph Neural Networks (GNNs) are a powerful tool for learning from graph-structured data, where nodes represent entities and edges capture relationships between them. They are widely used across various domains, such as social networks, molecular biology, and recommendation systems, due to their ability to effectively model complex relationships and interactions. This is particularly important in real-world scenarios where graphs are not static but change as interactions and relationships evolve.

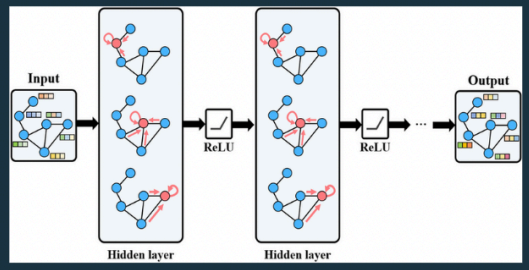
DGNNs represent entities as nodes with feature embeddings and encode evolving relationships through dynamically updated edges. These dynamic edges can reflect changes in connections, interactions, or weights over time, allowing the model to adapt to varying graph structures. We use GCNConv layers, which aggregate information from neighboring nodes and edges, to effectively learn and propagate both spatial and temporal patterns across the graph. By leveraging this approach, DGNNs can capture the interplay between the structural and temporal aspects of the data, making them capable of extracting meaningful insights from time-evolving datasets and adapting to the dynamics of complex systems.

Data

We utilized the NORAD dataset's TLE data of active satellites, which provides detailed two-line element (TLE) information for orbital modeling. Working with the SGP4 propagation algorithm, we converted TLE data into Cartesian coordinates to represent satellite positions in three-dimensional space. Using this ground truth, we generated a temporal sequence of graphs, where edges represent relative distances between satellites, capturing their spatial relationships over time.
Each snapshot of the dynamic graph was taken daily, enabling the model to process evolving spatial and temporal patterns effectively. We normalized features to ensure stable training and split our data into testing and training sets for robust evaluation.

Methodology

We approach this problem by working with a spatio-temporal graph neural network model that utilizes Graph Convolution via the GCNConv function in the PyTorch-Geometric library. This model is designed to capture both spatial and temporal dependencies in dynamic graph data. The first GCN layer transforms the input node features into an intermediate representation by aggregating information from neighboring nodes, effectively encoding the local graph structure.
The second GCN layer further refines this representation, capturing deeper spatial dependencies and enhancing the model's ability to discern intricate patterns. By normalizing the feature distribution, we maintain training stability, improve convergence, and ensure consistent performance across varying graph structures. This approach allows us to effectively handle the complexities of evolving graph data and extract meaningful patterns.



Model Architecture [4]

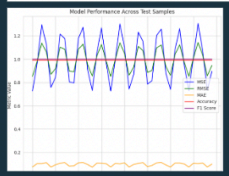
We calculate our loss using Mean Squared Error (MSE), a widely used metric for regression tasks that measures the average squared difference between predicted and actual values.
Our testing metrics include Mean Average Error (MAE) for evaluating prediction accuracy, along with F1-score, Recall, and Precision, which provide a comprehensive assessment of model performnce.

Results

Here's how our model performed against SGP4.

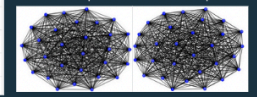
Metric	Value
Mean Squared Error (MSE)	0.9987
Mean Absolute Error (MAE)	0.0975
Accuracy	99.0000%
F1 Score	1.0000

Table 1
Metrics [5]



Graph 1
Metrics [5]

Time Step 12 Time Step 14



We observe that our model is able to accurately capture the spatio-temporal dependencies between satellites overhead and effectively predict collisions between satellites, which we have defined as anomalies. Impressively, our model achieves a 99% accuracy in identifying these anomalies, showcasing its reliability and precision in high-stakes scenarios.
This novel application of DGNNs provides an opportunity to further develop by incorporating additional features, potentially improving predictive performance and expanding the model's applicability.

Future Work

- Extend the model to integrate external factors such as space weather and satellite propulsion data.
- Improve edge representations using attention mechanisms for better and improved relational encoding.
- Apply the framework to broader datasets, including simulated collision events, for more rigorous training.

References

1. Kipf, T. N., & Welling, M. (2017). Semi-Supervised Classification with Graph Convolutional Networks.
2. Spacetrack.org. Two-Line Element (TLE) Data for Satellite Tracking.
3. Hamilton, W. L., Ying, Z., & Leskovec, J. (2017). Inductive Representation Learning on Large Graphs.
4. Kim, Minseong & Lee, Jaeseung & Kim, Jibum. (2023). GMR-Net: GCN-based mesh refinement framework for elliptic PDE problems. Engineering with Computers.

Code

5

Created By : Adityan Srinivasan & Sadhana Thirumangai
Group 7B : Graph Neural Networks for Anomaly Detection in Dynamic Graphs
Mentor : Dr Helena Bahrami

Optimizing MLPs for MNIST

by: Nayra Saadawy , Priyam Raul
mentor: Prof. Devendra Singh

Approaching Perfection...

Introduction

The main purpose of this research is to understand hindrances in order to create better performance, we study different parameters of the MLP and the effect they have on the performance of the MLP and its loss values, the parameters we focus on are the activation function, learning rate, and the step size, we experiment choices for all parameters and fix the optimal choice for each step until we reach the ideal model.

methods

To compare different parameters, we performed step-by-step tests, in each step, we measured the performance of options for the parameter tested,

-first, we tested 4 options for the activation function, Sigmoid, ReLU, Tanh, and Leaky ReLU, after gathering the loss values for each activation, Sigmoid and ReLU activation showed the lowest loss values.
-Second, the learning rate options were tested, which were: 0.0001, 0.001, and 0.01, each of these options was tested while fixing Sigmoid activation, and while fixing ReLU activation.

-For the last step, the step size was altered to 20, 30, and 40, all of which while fixing the learning rate to 0.001, and experimenting with both Sigmoid and ReLU activation.

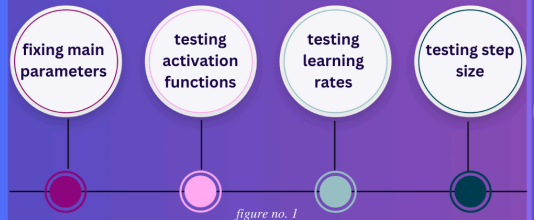
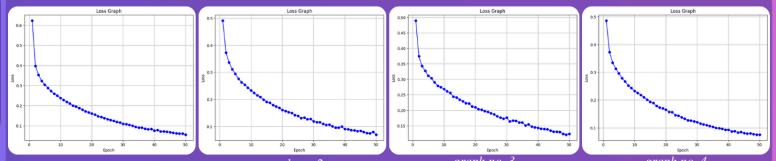


figure no. 1

after testing all the parameters the one that showed best performance was fixed as the final result of the filtering process.

results and analysis



graph no. 1 *graph no. 2* *graph no. 3* *graph no. 4*

Firstly, we tested 4 options for the activation function, Sigmoid *(graph no. 1)*, ReLU *(graph no. 2)*, Tanh *(graph no. 3)*, and Leaky ReLU *(graph no. 4)*, after gathering the loss values for each activation, Sigmoid and ReLU activation showed the lowest loss values.

Second, the learning rate options were tested, which were: 0.0001, 0.001, and 0.01, each of these options was tested while fixing Sigmoid activation, and while fixing ReLU activation.

lr = learning rate

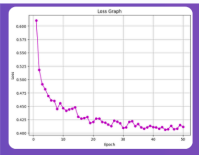
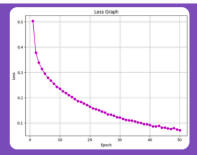
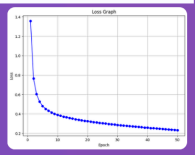
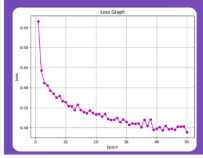
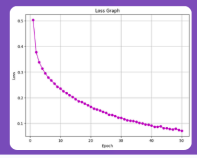
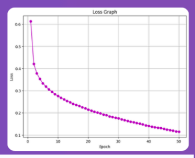
learning rate	0.01	0.001	0.0001
Sigmoid			
ReLU			

table no. 1

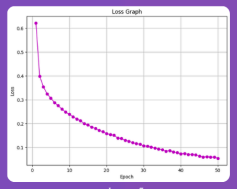
Finally, the step size was altered to 20, 30, and 40, all of which while fixing the learning rate to 0.001, and experimenting with both Sigmoid and ReLU activation.

the results optained were:

- for sigmoid:
  - 20 steps: Loss: 0.0607
  - 30 steps: Loss: 0.0552
  - 40 steps: Loss: 0.0546
- for ReLU:
  - 20 steps: Loss: 0.0718
  - 30 steps: Loss: 0.0700
  - 40 steps: Loss: 0.0753

based on all the results collected from the start and to the end, the model that showed the least loss *(graph no. 5)* value included:

- sigmoid activation
- 0.001 learning rate
- 40 step size



graph no. 5

real-life implementation

Reducing the effects of limitations as much as possible on the performance of MLPs can be translated to having more efficient models in real life, models that can recognize different objects and find the relationship between different visuals and graphics with high accuracy yet omitting noises, all these advantages can be utilized for purposes such as recognizing licenses plate numbers for cars, it can also be utilized in other fields such as historical studies, in understanding the relationship between hieroglyphs that developed as time passed, or the relationships between characters of neighboring civilizations, in addition to that, not to mention its huge role in fields like science, security, and medicine, all of which emphasizes the need of high accuracy models that aims to reach its best version.

conclusion

In this research, we systematically explored the effects of key parameters—activation function, learning rate, and step size—on the performance of a Multi-Layer Perceptron (MLP) model. By conducting experiments to optimize each parameter iteratively, we aimed to identify the ideal configuration for achieving superior performance and minimizing loss values.

The findings of our study revealed that the Sigmoid activation function, paired with a learning rate of 0.001 and a step size of 20, produced the most effective model in terms of performance and convergence. This result underscores the importance of careful parameter tuning in neural network design, as small adjustments can significantly impact model effectiveness.

resources

Glorot, X., & Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In Proceedings of the thirteenth international conference on artificial intelligence and statistics (pp. 249-256). JMLR Workshop and Conference Proceedings.

Musa, A., & Aliyu, F. (2019, October). Performance evaluation of multi-layer perceptron (mlp) and radial basis function (rbf). In 2019 2nd International Conference of the IEEE Nigeria Computer Chapter (NigeriaComputConf) (pp. 1-5). IEEE.

check out our code

MNIST Digit Classification: Comparing Classical and Quantum Approaches to Hyperparameter Tuning

Authors: Siddharth Kumar Gopal (SVM) and Harini Muthurengan (MLP) Mentor: Emilie Gregoire

1 Introduction

- Our study compares the optimal performance of Support Vector Machines (SVMs) and Multi-Layer Perceptrons (MLPs) through hyperparameter tuning.
- We employed:
  - Classical hyperparameter tuning methods for SVMs
  - Quantum approaches for MLPs
- For the SVM model, we focused on:
  - Gamma hyperparameter
  - Regularization hyperparameter (C)
- For MLPs, we utilized quantum circuits to generate hyperparameters.

2 Dataset Description - MNIST

- Training set of 60,000 examples and a test set of 10,000 examples.
- Each image is 28x28 images representing images from 0 to 9.
- Here the SVM model has used only 10,000 random images to decrease runtime and to make sure that the code was economically feasible. The model used 30% of selected images for training and the rest 70% for testing.
- Dataset was already pre-processed and balanced. Thus, it is also ideal to test both classical and quantum models.
- The dataset was taken from scikit-learn and used in our codes.

3 Implementation

```
Quantum Circuit:
0: ──RX(3.00)─┤  <Z>
1: ──RX(-1.10)─┤  <Z>
2: ──RX(-0.18)─┤  <Z>
```

Fig .1: Quantum Circuit

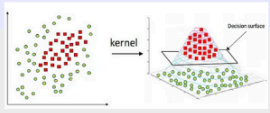


Fig .2: SVM Kernel

- We approached the problem through both classical and quantum approaches. SVM model through classical and MLP through quantum approach.
- SVMs used a Radial Basis Function (RBF) kernel and was optimized by tuning the gamma parameter and the regularization parameter (C). These were used to find a hyperplane which best separated the data points.
- MLPs used learning rate, number of neurons and regularization strength as hyperparameters and derived them from a quantum circuit employing rotation gates. These hyperparameters were used to then train and evaluate MLPs through cross-validation.

Link to our Google Colab where you can see the implementation of both our SVM and MLP models.

4 Results

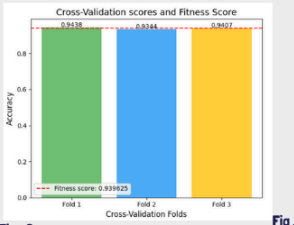


Fig . 3 : Accuracy of various folds through the hyperparameters generated through the quantum circuit

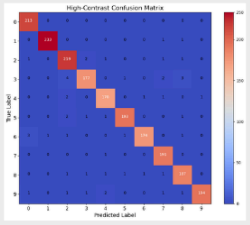
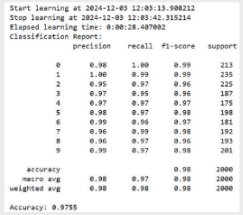


Fig . 4 :Confusion Matrix for SVM with 10,000 Images of MNIST
70% Testing
Gamma = 0.05 and C = 1.0
Accuracy = 0.9755



```
Start learning at 2024-12-03 12:03:13.908212
Stop learning at 2024-12-03 12:03:42.315214
Elapsed learning time: 0:00:28.407002
Classification Report:
              precision    recall  f1-score   support

           0       0.98      1.00      0.99       213
           1       1.00      0.99      0.99       235
           2       0.95      0.97      0.96       225
           3       0.97      0.95      0.96       187
           4       0.97      0.97      0.97       175
           5       0.98      0.97      0.98       198
           6       0.99      0.96      0.97       181
           7       0.96      0.99      0.98       192
           8       0.96      0.97      0.96       193
           9       0.99      0.97      0.98       201

    accuracy                           0.98      2000
   macro avg       0.98      0.97      0.98      2000
weighted avg       0.98      0.98      0.98      2000

Accuracy: 0.9755
```

Fig . 5 : Classification Report for SVM with 10,000 Images of MNIST
70% Testing
Gamma = 0.05 and C = 1.0
Accuracy = 0.9755

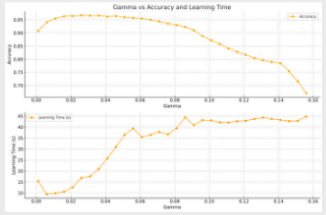
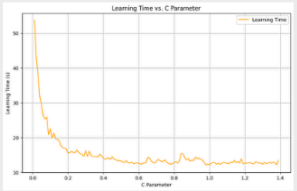


Fig . 6 : Variation of Learning Time and Accuracy with respect to Gamma.
Accuracy peaked at Gamma = 0.026.
Reached a peak accuracy of 0.967
Regularization Constant (C) set as 1.



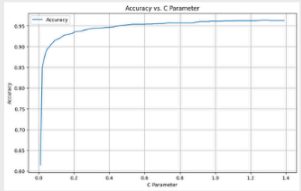


Fig . 7 & 8 : Variation of Learning Time and Accuracy with respect to C. Accuracy peaked at C = 1.26 to 1.3
Reached a peak accuracy of 0.963
Gamma = 0.026 (Peak Accuracy in Hyperparameter Tuning)

The performance of the SVM model was evaluated on the basis of two indices, the learning rate and the accuracy. We hypertuned two hyperparameters, the Gamma paramater and the regularization constant or the C parameter.

The gamma parameter in SVMs is a critical hyperparameter that controls the influence of individual training examples on the decision boundary. It is traditionally used to modify non-linear kernels, in this case, the Radial Basis Function (RBF) kernel.

5 Conclusion

SVM

- Combination of a Gamma value of 0.05 and a regularization constant of 1.0 returned the highest accuracy of 0.9755
- Accuracy rate increased with the value of Gamma parameter, plateaued, and then decreased with roughly the same slope. Peaked at Gamma = 0.926 with peak accuracy of 0.967.
- Learning time increased with the value of Gamma parameter after an initial drop and then plateaud towards the end.
- Accuracy rate initially increased with the value of C parameter very sharply and then plateaud towards the end. Peaked at C = 1.26 to 1.3 with peak accuracy of 0.963.
- Learning time initially decreased with the value of C parameter very sharply and then plateaud towards the end.

MLP

- The scores for the three cross-validation folds (0.9438, 0.9344, and 0.9407) indicates high accuracy in model performance.
- The average cross-validation accuracy of 0.939625, represented by the fitness score, indicates overall reliable model performance for the hyperparameters generated by the quantum circuit.
- These results suggest that the quantum circuit generated hyperparameters are not only valid but also capable of producing high accuracy results.

6 Reference

Jain, A. (2024). SVM Kernels and Its Type. Medium. Available at: https://medium.com/...
LeCun, Y., Cortes, C., & Burges, C. (2010). MNIST handwritten digit database. ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist
Cireşan, D. C., Meier, U., Gambardella, L. M., & Schmidhuber, J. (2010). Deep, big, simple neural nets for handwritten digit recognition. Neural computation, 22(12), 3207-3220.
Cortes, C., & Vapnik, V. (1995). Support-vector networks. *Machine learning*, *20*(3), 273-297.

Kolmogorov Arnold Network vs Multi-Layer Perceptron

Emeka Odiwe
Mentored by Dr Helena Bahrami

SCAN TO VIEW CODE ON GITHUB

1. Introduction

This study compares Kolmogorov-Arnold Networks (KANs) and Multi-Layer Perceptrons (MLPs) on classification tasks using datasets with varying sizes. Metrics such as accuracy, loss, convergence rates, and execution time are analyzed to assess the strengths and weaknesses of each model. The analysis ensures both models achieve high accuracy to facilitate a fair comparison, focusing on their ability to recall correctly. This work provides insights into the suitability of KANs and MLPs for classification tasks and offers guidance on selecting the optimal model based on dataset characteristics.

2. Brief Description and comparison

Theorem	Universal Approximation Theorem	Kolmogorov-Arnold Representation Theorem
Formula (Shallow)	$f(\mathbf{x}) \approx \sum_{i=1}^{N(\epsilon)} a_i \sigma(\mathbf{w}_i \cdot \mathbf{x} + b_i)$	$f(\mathbf{x}) = \sum_{q=1}^{2n+1} \Phi_q \left(\sum_{p=1}^{n} \phi_{q,p}(x_p) \right)$
Model (Shallow)	(a) *fixed* activation functions on *nodes*; *learnable* weights on *edges*	(b) *learnable* activation functions on *edges*; sum operation on *nodes*
Formula (Deep)	$\mathrm{MLP}(\mathbf{x}) = (\mathbf{W}_3 \circ \sigma_2 \circ \mathbf{W}_2 \circ \sigma_1 \circ \mathbf{W}_1)(\mathbf{x})$	$\mathrm{KAN}(\mathbf{x}) = (\mathbf{\Phi}_3 \circ \mathbf{\Phi}_2 \circ \mathbf{\Phi}_1)(\mathbf{x})$
Model (Deep)	(c) MLP(x): $\mathbf{W}_3$, σ_2, $\mathbf{W}_2$, σ_1, $\mathbf{W}_1$; nonlinear, fixed	(d) KAN(x): $\mathbf{\Phi}_3$, $\mathbf{\Phi}_2$, $\mathbf{\Phi}_1$; nonlinear, learnable

MLP is a neural network that works based on the universal approximation theorem. It makes use of layers having a number of neurons linked with other neurons from another layer. Each layer has a fixed activation function and each link contains learnable weight. Training is done using backpropagation to update the value of the weight .

KAN is a network that uses the Kolmogorov Arnold Representation Theorem which states that mutivariate functions can be represented as univariate functions. The learnable activation functions are controlled with Bspline. Bspline is a piecewise polynomial function of degree in a variable. It is controlled by control points and each curve between the control points has its degree

3. Dataset Analysis:

After analysis, I decided to go with Wine and Wisconsin Breast cancer Dataset. They have high data points(shape) with similar features(features), different balancing to test each model's ability for different number of classes, and classification task.

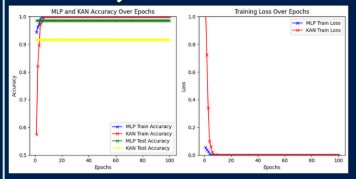
QR Code for the Dataset Analysis

References

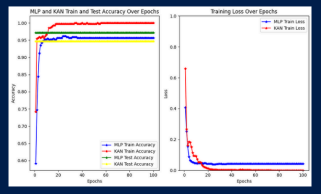
- Kolmogorov-Arnold Networks (KANs):Liu, Z., Wang, Y., Vaidya, S., Ruehle, F., Halverson, J., Soljačić, M., ... & Tegmark, M. (2024). Kan: Kolmogorov-Arnold networks. arXiv preprint arXiv:2404.19756.
- Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep learning. MIT press.
- Multilayer Perceptrons (MLPs):Nielsen, M. Neural Networks and Deep Learning. [Online]
- Haykin, S. S. (2009). Neural networks and learning machines. Pearson Education..
- Kirkpatrick, J., Pascanu, R., Rabinowitz, N., Veness, J., Desjardins, G., Rusu, A. A., ... & Hassabis, D. (2017). Overcoming catastrophic forgetting in neural networks. Proceedings of the national academy of sciences, 114(13), 3521-3526

4. Experimental Results

Wine 27,10



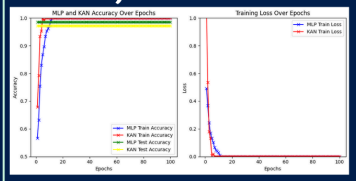
Cancer 27



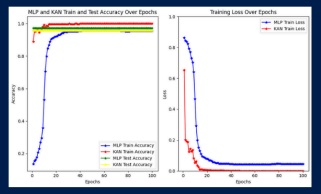
Wine Table

Model(Run number)	Hyperparameter	Execution Time(secs)	Test Accuracy(%)	Recall
MLP(0)	layer:2(27,10) Activation func: tanh Solver: adam	0.69	98.6	99
KAN(0)	Width: 13,27,10,3 Grid:5, k: 2	30.04	91.7	92
MLP(1)	layer(neurons):2(40,15) Activation func: tanh Solver: adam	0.42	98.6	99
KAN(1)	Width: 13,40,15,3 Grid:6, k: 2	53.67	97.2	97
MLP(2)	layer(neurons):2(15,7) Activation func: relu Solver: adam	0.38	98.6	100
KAN(2)	Width: 13,15,7,3 Grid:6, k: 3	15.29	98.6	99

Wine 40,15



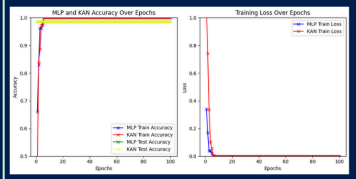
Cancer 64



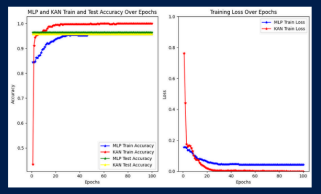
Cancer Table

Model(Run number)	Hyperparameter	Execution Time(secs)	Test Accuracy(%)	Recall
MLP(0)	layer:1(27) Activation func: relu Solver: adam	1.68	97.1	97
KAN(0)	Width: 9,27,2 Grid:5, k: 2	24.70	94.6	95
MLP(1)	layer(neurons):1(64) Activation func: relu Solver: sgd	1.07	97.1	97
KAN(1)	Width: 9,64,2 Grid:6, k: 2	57.68	95.7	96
MLP(2)	layer(neurons):1(18) Activation func: relu Solver: sgd	1.69	96.4	97
KAN(2)	Width: 9,18,2 Grid:5, k: 2	16.61	95.7	96

Wine 15,7



Cancer 18



5. Conclusion

1. MLP converges faster than KAN, as observed in the accuracy and loss graphs. This indicates that MLP reaches optimal performance in fewer epochs, making it suitable for tasks requiring quick results.
2. Both MLP and KAN achieve similar levels of final accuracy, with a difference of less than 0.03. However, MLP's accuracy was greater than KAN's accuracy. MLP also has significantly less loss values compared to KAN.
3. Both models exhibit strong generalization. Testing accuracy is comparable to training accuracy for both MLP and KAN, with no signs of overfitting. This confirms that both models are robust and reliable for unseen data.
4. For tasks requiring quick training and prediction times (e.g., real-time or large-scale systems), MLP is the preferred choice due to its faster convergence and lower computational cost.

3

GCANS

BRIDGING GEOMETRY & AI

1 Background
2 Methodology
3 Observations
4 Results+

Traditional models for robot dynamics are based on complicated kinematic equations, geometric Clifford algebra networks (GCANs) are an efficient alternative. Our research focuses on developing GCAN models for use in for predicting motion and forces in robots. Existing work on GCANs utilize symmetry group transformations through geometric algebras with the use of group action layers that combine multivector transformations with specified group actions (Ruhe, 2023).

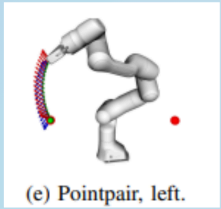


Fig 1: Trajectory Prediction (Low, n.d)

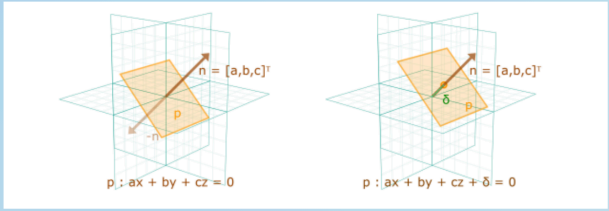


Fig 2: Planes (Ruhe, 2023)

Initially, we aimed to incorporate GCANs in robotic arms. However, GCANs could be more useful in outdoor robotic systems. Because of time restrictions for the research stage, we focused on implementing a Geometric Clifford Algebra - Multilayer Perceptron (GCA-MLP) and testing it for various tasks like projections and object recognition. In our GCA-MLP, the multivector(M) and the standard deviation(std(M)) of its components are used to normalize the multivector across the batch dimension.

The activation function MSiLU along with the loss function Mean Squared Error(MSE) is used to predict trajectories. MSE helps find the error between the target position and predicted position for each sample in the batch.

For the robot dynamic application, we used cross entropy loss along with the NAO dataset on robotic arms. This was used to observe how accurate our MLP was at classifying objects based on their position or rotation in space

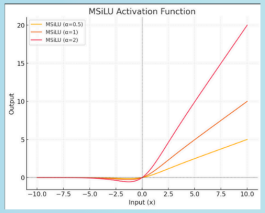


Fig 3: MSiLU Activation Function (Generated with MatPlotLib)

Even though GCANs are able to handle 3D problems, they're not great with 2D points. The loss values remained consistently high, this may be caused due to the limited dimensionalities of the 2D Euclidean space, and lack of geometric depth.

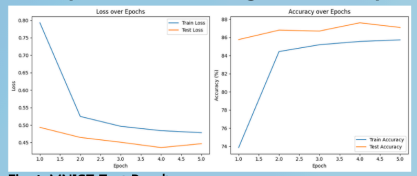


Fig 4: MNIST Test Results

Our MLP is moderately accurate at trajectory prediction, however, the predicted trajectory goes off track after a few time steps. We tried fine-tuning the hyperparameters and adding an attention layer to our code. Though we were able to improve the accuracy and minimize the loss, it wasn't highly accurate

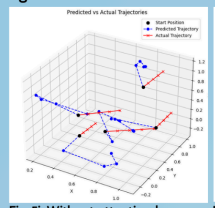


Fig 5i. Without attention layer (Generated by Matplotlib)

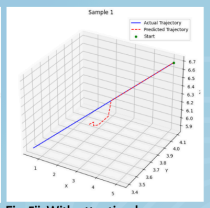


Fig 5ii. With attention layer (Generated by Matplotlib)

scan for our code!

While our MLP does not work well in 2D spaces, combining non geometric features proved to improve the accuracy and reduce the loss significantly. Though this is interesting, because of the computational overhead, we decided to not carry on with this approach.

The GCA-MLP proved to be highly accurate with extracting, combining and classifying geometric features like rotation and position in space. We plan to develop this to be used in aerial robots for pose classification and anomaly detection in the future.

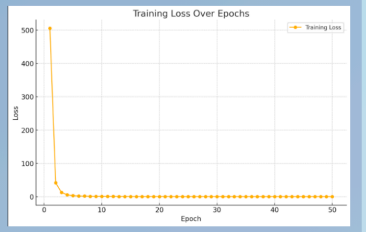


Fig 6. Loss Curve for geometric classification (Generated by Matplotlib)

References:
Löw, T. Calinon, S. (n.d.). *Geometric algebra for optimal control in robotics using gafro.*Idiap Research Institute, Martigny, Switzerland.

Ruhe, D., Gupta, J. K., de Keninck, S., Welling, M., & Brandstetter, J. (2023). Geometric Clifford Algebra Networks. arXiv. https://arxiv.org/abs/2302.06594

Loulia J. & Warenya K.

Mentors: Dr.Filip Bar, Mr. Matthew Pugh

Overfitting vs Double Descent

Authors: E.Ozodbek, A.Arav
Mentor: Emilie Gregoire

Abstract

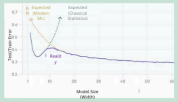
In this study, we investigate the interplay between double descent and regularization across shallow neural networks, decision trees, and ResNet architectures, using both synthetic and real-world-inspired datasets. Our experiments span multiple training regimes, including epoch-wise and model complexity-driven setups, revealing architecture-specific manifestations of double descent. Notably, dropout proves effective in mitigating overfitting in high-complexity regions, while weight decay demonstrates consistent regularization benefits across simpler models. Additionally, unexpected anomalies, such as test loss outperforming train loss, were observed in specific configurations, shedding light on the nuanced dynamics of regularization.

Introduction

Machine learning has evolved, leading to the development of overparameterized models that fit training data exactly while achieving strong generalization performance. This paradoxical behaviour challenges traditional notions of the bias–variance trade-off, which has guided our understanding of model complexity and generalization dynamics.

Double descent refers to a behaviour in model performance as complexity increases. Traditional machine learning theory suggests a U-shaped curve where test error decreases up to an optimal complexity level before rising due to overfitting. However with double descent, test error initially decreases, rises, and then decreases again.

This study addresses the following research question: How do regularization techniques (such as dropout and weight decay) influence the double descent phenomenon and mitigating overfitting across different deep learning model architectures?



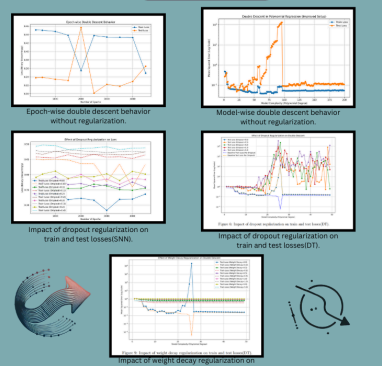
Methodology

Synthetic datasets were generated to simulate regression and binary classification tasks. The primary data generation process followed:

$$[y = \sin(2\pi x) + \epsilon, \quad \epsilon \sim \mathcal{N}(0, \sigma^2)]$$

where ϵ represents Gaussian noise. The classification tasks made a threshold on the regression outputs to create binary labels. Shallow networks were chosen to analyse dropout effects because they are simpler compared to deeper architectures. The smaller number of layers reduces the variables, making it easier to understand the role of dropout. Shallow networks are computationally efficient to train, which allowed us to explore a wide range of dropout rates and configurations across different training regimes. This choice aligns with the study's aim to explore the interaction between double descent and regularization.

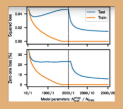
Results



Find our Code
Read more here:

Findings

Model Type	Dropout Effects	Weight Decay Effects	Combined Effects
Shallow Networks	Reduced overfitting (dropout 0.1–0.2). Instability at higher rates (dropout 0.4).	Smoothed loss curves. Reduced overfitting. High weight decay risks underfitting.	Balanced regularization. Improved generalization without instability.
Decision Trees	Unstable at higher dropout rates. Reduced accuracy with over-regularization.	Smoothed loss curves. Improved performance at moderate weight decay.	Stable performance. Balanced trade-off between overfitting and underfitting.
Polynomial Models	Reduced test loss peaks. Variability introduced at high dropout rates.	Smoothened loss curves. Enhanced generalization in overparameterized regimes.	Consistent test loss reductions. Synergistic effects in regularization.



Polynomial regression was used as it provides a well-controlled and mathematically grounded setting for observing double descent. The model complexity can be easily modulated by adjusting the polynomial degree, allowing us to capture the behaviour at and beyond the interpolation threshold. Polynomial regression is a classic example where the double descent phenomenon is theoretically understood, providing a baseline for comparison with more complex models. Its simplicity made it ideal for verifying foundational behaviours of double descent before extending the experiments to neural networks and decision trees.

Future Work

Expand experiments to larger and more diverse datasets to validate findings and uncover additional patterns in double descent behavior.

Investigate additional regularization techniques, such as batch normalization, early stopping, and data augmentation, to assess their interaction with double descent.

Extend the study to modern architectures, such as transformers, to evaluate the broader applicability of findings.

Explore adversarial training and other robust optimization methods as potential mitigators of overfitting in double descent scenarios.

Future Work Although ResNet experiments did not yield clear double descent trends, future studies could explore this phenomenon with larger computational resources, alternative datasets, or simplified ResNet variants. Investigating how modern architectures like ResNet interact with regularization techniques remains an open question with significant practical implications.

Conclusion

- Double descent was observed in both model complexity and epoch-wise training, with test loss showing characteristic peaks at the interpolation threshold.
- Dropout effectively reduced overfitting at moderate rates (0.1-0.2), but higher rates introduced instability and underfitting.
- Weight decay smoothed loss curves and improved generalization, though excessive regularization suppressed double descent, leading to underfitting.
- Combined regularization strategies provided the most consistent performance, balancing overfitting and underfitting across diverse configurations.

References

[1] Mikhail Belkin, Daniel Hsu, Siyuan Ma, and Soumik Mandal. Reconciling modern machine-learning practice and the classical bias–variance trade-off. Proceedings of the National Academy of Sciences, 116(32):15849–15854, 2019.

[2] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Deep Learning. MIT Press, 2016. Website.

[3] Jared Wilber and Brent Werness. Double descent: A visual introduction, 2021.

EARLY DETECTION OF DIABETIC RETINOPATHY USING MACHINE LEARNING

Diabetic retinopathy (DR) is a leading cause of blindness, accounting for over 1 million blindness cases and 3.28 million severe vision impairments globally. Early detection can significantly mitigate its severity and impact. This research conducts a comparative analysis of machine learning models, specifically Convolutional Neural Networks (CNNs) and Vision Transformers (ViTs), to evaluate their effectiveness in detecting DR and similar medical conditions in low-resource settings or clinical practice.

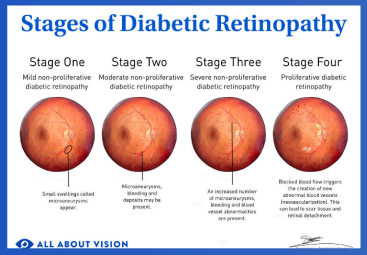


Fig 01: DR Stages

HOW DETECTION WORKS

Machine learning models like CNNs and ViTs analyze retinal images to detect key indicators of DR, such as microaneurysms, hemorrhages, and exudates:

- Stage 1 (Mild): Characterized by 1 or 2 microaneurysms.
- Stage 2 (Moderate): More than 20 microaneurysms and occasional hemorrhages.
- Stage 3 (Severe): Numerous hemorrhages in all quadrants, along with venous beading.
- Stage 4 (Proliferative DR): Growth of new, fragile blood vessels (neovascularization) prone to leakage.

These features are analyzed to classify disease. severity, aiding early intervention and treatment.

Image Preprocessing

The Kaggle DR Dataset 2019, containing 224×224 retinal images across five diagnostic categories, was augmented using the following operations:

- Rotation: Random rotations up to ±15° for varying camera angles.
- Shifting: Horizontal and vertical shifts up to 10% for alignment issues.
- Shear & Zoom: Shearing up to 10% and zooming up to 20% for distortions and scale differences.
- Flipping: Horizontal flipping to handle symmetrical features.

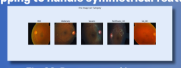


Fig 02: Preprocessed Images

CNN

This Convolutional Neural Network (CNN) classifies retinal images by DR severity:

Input Images: Processes 224x224-pixel retinal images

Feature Extraction: Convolution detects patterns like spots or textures indicative of disease.

Filters slide over the image to extract features, refined through multiple layers.

Flattening: The 3D feature maps are converted into a 1D array for further processing.

Classification: Fully connected layers analyze features, and the output layer categorizes images accrodinsg to stages.

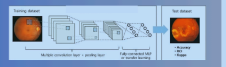


Fig 03: CNN

ViT

The Vision Transformer detects DR by the following process:

Patch Encoding: The image is divided into 16×16 patches, flattened, and embedded with positional data to retain spatial relationships.

Transformer Blocks: Four Transformer blocks use self-attention to analyze relationships between patches.

Feature Pooling and Classification: Global pooling aggregates patch features, followed by dense layers that classify the image into stages.

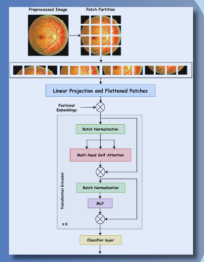


Fig 04: ViT

Binary Comparision (DR vs No DR)

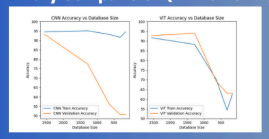


Fig 05: Accuracy over Database Size

The graph shows that for CNN, training accuracy remains high (91-95%) all across but validation accuracy drops significantly as the dataset size decreases. The ViT model, however, exhibits stable validation accuracy (~63-94%) even with smaller datasets.

MultiClass Comparision (DR Stages)

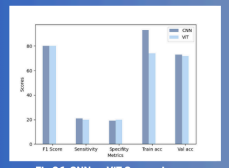


Fig 06: CNN vs ViT Comparison

Both CNN and ViT models display similar trends for F1-score, Sensitivity & Specificity (in Fig 06). However, the CNN's Training Accuracy is ever increasing while validation accuracy plateaus at 70-75%, while the ViT model oscillates around 73% for both training and validation, indicating more consistent but limited performance.

Analysis

Overall, the ViT model shows more consistent performance with training and validation accuracies around 70-73%, indicating better generalization and less overfitting. However, the confusion matrix reveals misclassification between "Moderate" and "No DR" categories. In contrast, the CNN model has higher training accuracy (~90%) but stagnant validation accuracy (70-75%), suggesting overfitting.

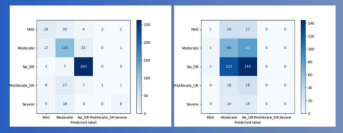


Fig 08: Confusion Matrix Comparision (CNN vs ViT)

Conclusion

In the context of detecting medical conditions like diabetic retinopathy, Vision Transformers (ViTs) demonstrate better generalization, making them potentially more robust for diverse, unseen cases. However, this advantage heavily relies on pretraining with large, related datasets. Convolutional Neural Networks (CNNs), despite a tendency to overfit, excel at extracting local features and distinguishing specific stages. For rare or underrepresented cases, CNNs can outperform ViTs if augmented datasets and regularization techniques are carefully applied. Both models demonstrate complementary strengths, and their selection should depend on the dataset size, available resources, and application requirements.

References

- Fig 1 source: All About Vision
- Vishal Awasthi a, Namita Awasthi b, Hemant Kumar c, Shubhendra Singh c, Prabal Pratap Singh d. (2021). Optimized vision transformer for diabetic retinopathy detection using Harris Hawk optimization.
- Gulshan, V., Peng, L., Coram, M., Stumpe, M. C., Wu, D., Narayanaswamy, A., & Webster, D. R. (2016). Development and validation of a deep learning algorithm for detection of diabetic retinopathy in retinal fundus photographs. Jama, 316(22), 2402-2410.
- Ting, D. S. W., Cheung, C. Y. L., Lim, G., Tan, G. S. W., Quang, N. D., Gan, A., & Wong, T. Y. (2017). Development and validation of a deep learning system for diabetic retinopathy and related eye diseases using retinal images from multiethnic populations with diabetes. Jama, 318(22), 2211-2223.
- Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems (pp. 1097-1105).

Author
MD Nafiul Haque
Mentor
Dr. Devendra Singh Dhami

Scan to access our code!

DOUBLE DESCENT VS OVERFITTING IN DEEP LEARNING

Scan here to view the code

I. Introduction

Overfitting, a classical issue in deep learning, occurs when models perform well on training data but poorly on unseen data. Double descent challenges this notion, revealing that model performance can improve beyond the interpolation threshold as capacity increases, offering new insights into the dynamics of overparameterized models.

II. Methodology

- **Model Design:** A flexible feedforward neural network (FlexibleNN) was developed to study the effects of model complexity on performance.
- **Data Preparation:** The MNIST dataset was used for training and evaluation, consisting of grayscale images of handwritten digits.
- **Training and Evaluation Pipeline:** The model was trained using cross-entropy loss and SGD optimizer, with training and testing across epochs to analyze performance trends like overfitting and double descent.
- **Visualization:** Training and test losses were plotted across epochs for each model configuration. This allowed the identification of overfitting regions and the double descent behavior as model complexity increased.

III. Findings

- **Effect of Model Complexity:**
  - Increasing the number and size of hidden layers initially improved performance, reducing both training and test loss.
  - Beyond a certain complexity, overfitting occurred, evidenced by a significant divergence between training and test loss.
- **Double Descent Phenomenon:** For highly complex models, test loss showed an initial increase (overfitting) followed by a decrease as model capacity grew further, demonstrating double descent behavior.
- **Performance Trends:**
  - Simpler models struggled to capture patterns effectively, resulting in higher training and test loss.
  - Intermediate-complexity models balanced generalization and overfitting, achieving optimal performance.
- **Loss Visualization:** Training and test loss curves provided clear visual evidence of overfitting and the transition to the double descent phase as model complexity increased.

IV. Conclusion

This study demonstrated the double descent phenomenon, where increasing model complexity initially caused overfitting but later improved generalization. These findings highlight the complex relationship between model capacity and performance.

V. Double descent graph

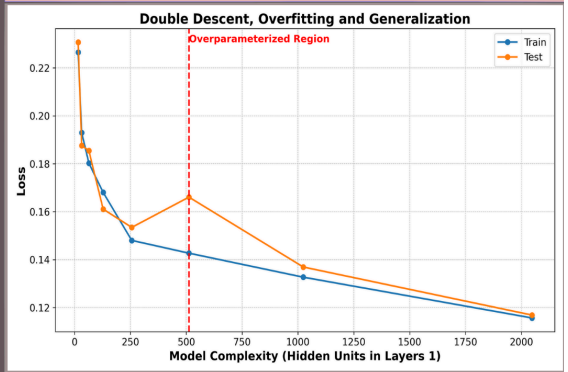


Figure 1.1

VI. References

1.Wilber J., & Werness B. (December 2021). Double Descent. MLU-Explain [Online]

2. Belkin, M., Hsu, D., Ma, S., & Mandal, S. (2019). Reconciling modern machine-learning practice and the classical bias–variance trade-off. Proceedings of the National Academy of Sciences, 116(32), 15849-15854.

3. Deep Double Descent: Where Bigger Models and More Data Hurt Preetum Nakkiran, Gal Kaplun, Yamini Bansal, Tristan Yang, Boaz Barak, Ilya Sutskever

Author: Shaurya Karmakar
Mentor: Adeyemi D. Adeoye