An-Najah National University

Faculty of Graduate Studies

USING ARTIFICIAL NEURAL NETWORK TO PREDICT PARTICLE TYPE IN HIGH-ENERGY PHYSICS

By
Iman Adnan Othman

Supervisor
Dr. Baker Abdulhaq

This Thesis is Submitted in Partial Fulfillment of the Requirements for The Degree of Master of Advanced Computing, Faculty of Graduate Studies, An Najah National University, Nablus-Palestine.

2023

USING ARTIFICIAL NEURAL NETWORK TO PREDICT PARTICLE TYPE IN HIGH-ENERGY PHYSICS

By
Iman Adnan Othman

This thesis was Defended successfully on 23/11/2023, and approved by:

Dr. Baker Abdulhaq	
Supervisor	Signature
Dr. Ahmad Hasasneh	
External Examiner	Signature
Dr. Amjad Hawash	
Internal Examiner	Signature

Dedication

To my dearest parents, my son Omar, my beloved siblings, and my husband Jihad.

Thanks for the profound impact each of you has had on my life. You are the roots that ground me and the wings that lift me up. With all my love and gratitude.

Acknowledgement

First and foremost, I would like to extend my sincere gratitude to my supervisor Dr. Baker Abdulhaq, who has been a great supervisor. His support, critical thoughts, vision and encouragement have been a focal factor in accomplishing this thesis.

I am deeply grateful to all the staff who taught me throughout the master's program, especially Dr. Amjad Hawash, Dr. Ahmad Awad, Dr. Othman Othman, and Dr. Adnan Salman.

Last but not least, I am grateful to my lovely family, especially my mother whose prayers surely had an effect in giving me the power to undertake this journey. I am also grateful to my friends Iman Mkheimer and Marwa Khaled for their infinite support during the writing of this thesis.

Declaration

I, the undersigned, declare that I submitted the thesis entitled:

USING ARTIFICIAL NEURAL NETWORK TO PREDICT PARTICLE TYPE IN HIGH-ENERGY PHYSICS

I declare that the work provided in this thesis, unless otherwise referenced, is the researcher's own work, and has not been submitted elsewhere for any other degree or qualification.

Student's Name: ایمان عدنان خفاجی

Signature: ایمان خفاجی

Date: ٢٠٢٣/١١/٢٣م

List of Contents

Dedication iii
Acknowledgement iv
Declaration v
List of Contents vi
List of Tables viii
List of Figures ix
List of Appendixes x
Abstract xii
Chapter One: Introduction and Theoretical Background 1
1.1 Large Hadron Collider (LHC) 1
1.2 Jet Energy Transverse (Jet) 3
1.2.1 Jet Definitions 3
1.2.2 Jet Algorithms 3
1.2.3 Properties and Classification of Jet 4
1.2.4 Classification of Jets into the Five Nuclear Particle Categories 5
1.3 Research Questions 6
1.4 Machine Learning 6
1.4.1 Introduction to Machine Learning 6
1.5 Classification 19
1.5.1 Classification Algorithm for Jet Classification 19
1.6 Neural Network 20
1.7 Activation Function 22
1.8 Problem Statements 33
1.9 Literature Review 34
Chapter Two: Methodology 39
2.1 The Dataset 39
2.1.1 Data Sources 39
2.1.2 Data loading and Pre-Processing 40
2.2 Neural Networks 41

2.3 Training an Artificial Neural Network 43
2.4 Optimization for Neural Networks 44
2.5 Overfitting 50
2.6 Using Feature Importance 53
2.7 Keras 54
Chapter Three: Results and Discussion 56
3.1 Single-Layer Neural Network 57
3.1.1 Different Activation Functions 57
3.1.2 Different Optimizers 60
3.1.3 Varying Epochs 63
3.2 Two-Layer Neural Network 63
3.2.1 Different Activation Functions 64
3.2.2 Different Optimizers 65
3.2.3 Change the Number of Epochs 66
3.3 Three Layer 66
3.3.1 Different Activation Functions 67
3.3.2 Different Optimizer 68
3.3.3 Change the Number of Epochs 70
Chapter Four: Conclusion and Future Work 71
4.1 Conclusion 71
4.2 Future Work 71
References 73
Appendices 78
الملخص ..... ب

List of Tables

Table *2*.1: Single-Layer Model Performance..... 51

Table *2.2*: Three-Layer Model Performance with Different Activation Functions 51

Table 3.1: Accuracy of Different activation Functions of Single-Layer Neural Network..... 58

Table 3.2: Accuracy of Different Experiments of Single-Layer Neural Network 58

Table 3.3: Anova: Single Factor of Single-Layer Neural Network 59

Table 3.4: Train Accuracy of Different Activation function of Single-Layer Neural Network..... 60

Table 3.5: Accuracy of Different Optimizer Function of Single-Layer Neural Network..... 60

Table 3.6: Train Accuracy of Different Optimizer Function of Single-Layer Neural Network..... 62

Table 3.7: Accuracy of Varying Epochs of Single-Layer Neural Network..... 63

Table 3.8: Accuracy of Different Experiments of Two-Layer Neural Network 64

List of Figures

Figure 1.1: Machine Learning Classification Techniques (Shailaja et al., 2018) 7
Figure 1.2: The Processes of Supervised Machine Learning 14
Figure 1.3: A Biological Neuron and an Artificial Neuron..... 21
Figure 1.4: Graph equation for the ReLU activation function 24
Figure 1.5: Graph derivative of equation for the ReLU activation function..... 24
Figure 1.6: Graph equation for the ELU activation function 25
Figure 1.7: Graph derivative of equation for the ELU activation function..... 26
Figure 1.8: Graph equation for the SELU activation function 27
Figure 1.9: Graph derivative of equation for the SELU activation function 27
Figure 1.10: Graph equation for the Sigmoid activation function 29

List of Appendixes

Appendix A: Tables 78

Table A.1: Accuracy of Different Activation Functions of Two-Layer Neural Network 78

Table A.2: Accuracy of Selu Activation Function of Three-Layer of Neural Network 80

Table A.3: Accuracy of ReLU Activation Function of Three-Layer of Neural Network 82

Table A.4: Accuracy of ELU Activation Function of Three-Layer of Neural Network 84

Table A.5: Accuracy of Sigmoid Activation Function of Three-Layer of Neural Network 86

Table A.6: Accuracy of Tanh Activation Function of Three-Layer of Neural Network 88

Table A.7: Accuracy of Softmax Activation Function of Three-Layer of Neural Network 90

Table A.8: Accuracy of Softplus Activation Function of Three-Layer of Neural Network 92

Table A.*9*: Anova: Single Factor of Two-Layer Neural Network 94

Table A.*10*: Accuracy of Different Optimizer Functions of Two-Layers Neural Network 94

Table A.*11*: Accuracy of different Number of Epochs of Two-Layer Neural Network 95

Table A.*12*: Accuracy of Different Activation Functions of Three-Layer 95

Table A.*13*: Anova: Single Factor of Three-Layer of Neural Network 96

Table A.*14*: Training and evaluation time of Three-layer of Neural Network models 97

Table A.*15*: Training and evaluation time of Three-layer of Neural Network models 97

Table A.*1*6: Accuracy of Different Number of Epochs of Three-Layers 97

Appendix B: Figures 98

Figure B.1: Graph Derivative of Equation for the Sigmoid Activation Function 98

Figure B.2: Graph Equation for the Tanh Activation Function 98

Figure B.3: Graph Derivative of Equation for the Tanh Activation Function ... 98

Figure B.4: Graph Equation for the Softmax Activation Function. 99

Figure B.5: Graph Equation for the Softplus Activation Function 99

Figure B.6: Distributions of the 16 High-Level Features Used in this Study Ref. 100

Figure B.7: Flowchart of a Machine Learning Model 101

Figure B.8: Chart Accuracy of Different Activation Function of Single-Layer 101

Figure B.9: Chart Accuracy of Different Experiments of Single-Layer Neural Network 102

Figure B.10: Chart Accuracy of Different Optimizer Function of Single-Layer 102

Figure B.11: Chart Varying Epochs of Single-Layer Neural Network 103

Figure B.12: Chart Average of Accuracy of Different Experiments of Two-Layers 103

Figure B.13: Chart Accuracy of Different Optimizer Function of Two-Layer 104

Figure B.14: Chart Accuracy of Different Number of Epochs of Two-Layer. 104

Figure B.15: Chart of Accuracy Comparison of Different Optimizers in Neural Network Training 105

Figure B.16: Epoch Count vs. Accuracy Chart in Three-Layer Neural Network Training 105

USING ARTIFICIAL NEURAL NETWORK TO PREDICT PARTICLE TYPE IN HIGH-ENERGY PHYSICS

By
Iman Adnan Othman
Supervisor
Dr. Baker Abdulhaq

Abstract

In the realm of high-energy physics, such as particle collision experiments in particle accelerators like the Large Hadron Collider (LHC), complex collision events occur, leading to the formation of particle jets. Precisely identifying and describing these jets is crucial for understanding fundamental particles and their interactions. However, traditional jet identification algorithms face challenges in capturing subtle features and interactions within jets, especially in dense and complex environments. Therefore, it is considered highly important to know the particle type in high-energy physics to improve scientific understanding of particles and their interactions.

Artificial intelligence is an important research field that helps in finding out perfect solutions to various scientific problems across various scientific and everyday life domains, particularly in the context of deep learning. Many models have been studied, such as deep neural networks, jet classification, the use of neural networks, and recurrent neural networks. This study explores the use of neural networks to classify jets into five different categories (light quarks (q), gluons (g), W and Z bosons, and top quarks) with the highest possible accuracy. Using a model within the TensorFlow/Keras framework, we borrowed data from the Zenodo platform consisting of 150 particles with 16 attributes used for jet classification. The methods included building various neural network architectures in depth, including single-layer networks, two-layer networks, and three-layer networks. We explored different activation functions, the number of training epochs, and optimizers. Additionally, we adopted a strategy to control for overfitting and identify prominent features to improve classification performance.

The best results were achieved by building a three-layer neural network using Softmax, Sigmoid, and Selu activation functions, with the Adamax optimizer. These results were achieved through training the model for approximately 200 epochs, achieving an

accuracy of 0.7400. This current study highlights the potential of neural networks to achieve high levels of jet classification accuracy and provides insights into improving neural network architectures for similar tasks in particle physics research.

Keywords: machine learning, neural networks, activation functions, optimizer functions, jet classification, Large Hadron Collider.

Chapter One

Introduction and Theoretical Background

This introduction chapter starts with defining the terms machine learning, classification, and neural networks. Second, it explores concepts related to Large Hadron Collider (LHC) and jets in an attempt to help the reader understand the study's focus on jet classification. Third, it discusses the Five Nuclear Particle Categories into which jets are classified. Then, a thorough representation of the research questions is provided. Finally, a number of studies are reviewed in an attempt to present the reader with an up-to-date image of the prominent applications in neural networks.

1.1 Large Hadron Collider (LHC)

The Large Hadron Collider (LHC) is a type of particle accelerator that is produced due to collisions occurring in experiments conducted in particle physics, where jets, which are collimated sprays of particles, are produced due to high-energy particle collisions. Thus, jets can be described as tiny particles that were formed due to colliding of two high-energy particles. These secondary particles move in the same direction as the main ones. The jet particles expose the characteristics of the particles involved and the properties of the collision. For example, they can help understand the nature of the collision and the characteristics of the particles during collision. This helps in designing better models of particle physics and enhancing existing ones.

The LHC in Switzerland is considered the largest and most powerful accelerator in the world. Its circumference is about 27 km. It accelerates hadrons such as protons, which then travel in opposite directions and collide at four locations where the machine's two rings cross. It helps collect data from such collisions related to their energy, angles, and other physical characteristics. Thus, such classification aims to understand these properties and classify the particles based on them [28].

Enhancement of the neural network models is achieved through analyzing the data collected after each experiment, which help train the models to make use of the relationships between the different features of the collided particles. Then, the enhanced models are used to achieve more accurate jet classification (JETS). Artificial neural networks are used to analyze the large and complex data produced by the LHC. Thus, a

better understanding of particle-level physical phenomena and the discovery of new particles become possible more accurately. The LHC contributes in a significant way to understanding important issues related to particle physics, leading to various discoveries taht have led to updates in our model of the cosmos and the particles that constitute it [28].

When was it designed?

The Large Electron-Positron (LEP) collider was conceived and constructed in the early 1980s, although CERN groups were already engaged in planning for the long term at that time. When CERN's governing body, the CERN Council, agreed to allow the construction of the LHC in December 1994, their aspirations of building such a machine finally came true after many years of work on the technical needs and physics requirements of such a machine. The project was approved on the basis that the new accelerator would be constructed within a set budget and that any contributions from non-member States would be used to further and enhance the project. Budgetary restrictions initially suggested that the LHC would be a 2-stage project. However, in 1995, the Council decided to allow the project to move on in a single phase in response to contributions from Japan, the USA, India, and other non-member States. Four experiments—ALICE, ATLAS, CMS, and LHCb—received official permission between 1996 and 1998, and building work started on the four sites. Since then, three smaller experiments (TOTEM, LHCf, and MoEDAL) have joined the search: TOTEM, situated adjacent to CMS, LHCf, installed next to ATLAS, and MoEDAL, placed in the vicinity of the intersection where the LHCb detector is located [28].

What is the Collision Energy at the LHC and What is so Special About it?

The maximum design energy of each proton beam traveling around the LHC is 7 TeV, making the collision energy between two protons 14 TeV. The maximal collision energy of lead-ion beams is 1150 TeV due to the large number of protons present in lead ions. Particle collisions are unique because of their high energy concentration. When you clap your hands, you create a "collision" at a far higher energy than protons at the LHC, but it is much less concentrated since it spreads out across the entire region of your hand. These energies are not very noteworthy in terms of their absolute magnitude when compared to the energies we encounter daily. In actuality, 1 TeV is

equivalent to the motional energy of a flying mosquito. The LHC is unique because it packs energy into a volume that is a million times smaller than a mosquito [28].

1.2 Jet Energy Transverse (Jet)

1.2.1 Jet Definitions

In particle physics, jets are streams of particles arranged in a collimated spray that arise from high-energy particle collisions. These collisions typically take place in experiments conducted at particle accelerators like the Large Hadron Collider (LHC). Particle jets can be described as tiny particles that were formed due to colliding of two high-energy particles. These secondary particles move in the same direction as the main ones. The jet particles expose the characteristics of the particles involved and the properties of the collision. For example, they explain the nature of the collision and the characteristics of the particles during collision, which in turn helps enhance the models of particle physics [31].

1.2.2 Jet Algorithms

Jet algorithms are mathematical algorithms that are used to group and analyze jets in particle physics. They mainly rely on gathering the particles generated during particle collisions in accelerators, such as the LHC. They aim to comprehend particle interactions that take place during these collisions. Main algorithms for jet clustering include [25]:

- KT Algorithm: This algorithm groups particles based on their proximity in momentum or energy. It is particularly sensitive to high-momentum particles.
- Cambridge/Aachen Algorithm: It is used for jet analysis in physical space, gathering particles based on their angles and distances from each other.
- Anti-kT Algorithm: This algorithm groups particles to form jets with a circular shape, suitable for analyzing jets in the rapidity-azimuth space.
- SISCone Algorithm (Seedless Infrared-Safe Cone): It clusters particles within independent cone-shaped regions, fitting for analysis in high-density environments.

These algorithms contribute a lot to data analysis and jet identification in physics experiments, enhancing the conclusions drawn in particle physics.

1.2.3 Properties and Classification of Jet

There are 16 characteristics that are used for jet classification on the Zenodo platform. They are known as "Jet Tagging." The characteristics are usually utilized to analyze data resulting from particle collisions in accelerators like the LHC. They help in identifying various types of jets and recognizing the particles produced from them. Here's a list of those properties:

- Energy Fraction: It portrays the energy retained in the jet compared with the total energy of the interaction.
- Path Length: Refers to the distance traveled by each particle within the jet.
- Energy Integral: Represents the sum of energy for all particles in the jet.
- Density: Represents the number of particles inside the jet relative to its volume.
- Overlap: Determines the extent of the spread and overlap of different jets.
- Minimum Energy: Defined as the minimum amount of energy that the jet must contain.
- Isolation Factor: Defined as the separation between the jet and surrounding jets.
- Total Energy: Represents the total energy of all particles within the jet.
- Outgoing Energy Fraction: Represents the ratio of outgoing energy to the total energy of the jet.
- Angular Dispersion: Reflects the spread of the jet at a specific angle.
- Number of Particles: Defined as the number of particles inside the jet.
- Total Mass: Represents the total mass of all particles within the jet.
- Extra Velocity: Determines the extent of additional momentum of the jet.
- Linear Density: Represents the number of particles relative to the distance traversed by the jet.
- Spherical Density: This is the density of particles (i.e., the thickness of the sphere around the jet.
- Sub-Angle: Determines the extent of deviation of secondary particles at a specific sub-angle.

These properties are used to differentiate and classify jets based on their physical and statistical properties in particle physics analyses.

In our system, we classify jets using neural networks. These networks are trained using data available on the Zenodo platform, where the aforementioned properties are relied upon for jet classification.

1.2.4 Classification of Jets into the Five Nuclear Particle Categories

The problem with our project lies in classifying jets into five different categories (light quarks (q), gluons (g), W and Z bosons, and top quarks). In this project, we aim to utilize neural networks to perform these tasks with high accuracy. However, before delving into the details of the work, I want to provide a general idea about these five categories.

1. Light Quarks (q):
   - Light quarks are fundamental particles composing protons, neutrons, and other particles.
   - They include "up" and "down" quarks.
   - They interact with both weak and strong nuclear forces and are essential components of nuclear structures.
   - They are particles carrying the strong nuclear force.
   - They bind quarks together within atomic nuclei by exchanging gluons between them.
   - Gluons exhibit complex interactions governed by quantum chromodynamics.
2. W and Z Bosons:
   - W and Z bosons act as intermediary particles for the weak nuclear force.
   - They facilitate transitions between different particles within atomic nuclei.
   - Their usage encompasses describing nuclear decay processes and other reactions involving particle transformations.
3. Top Quarks:
   - Top quarks are the heaviest particles in the quark family.
   - They interact with both weak and strong nuclear forces.
   - Top quarks are predominantly generated in high-energy interactions like collisions between protons and neutrons in nuclear experiments.

These categories serve as the foundation for understanding particle physics and nuclear interactions, and leveraging neural networks can aid in accurately classifying particles into these five categories.

1.3 Research Questions

The main research questions of the study are:

1. Can neural networks be used to predict the type of particles?
2. How many hidden layers in ANN will we use and what effect does it have on accuracy?
3. What are the important data and the unimportant data that can be dispensed with in the classification process, and what impact does ithave on accuracy?
4. What is the right activation function for the hidden and outputlayers of the network: ReLU, ELU, or SELU functions?
5. What is the number of neurons in the hidden layer?
6. What is the optimizer algorithm that better fits this problem?

1.4 Machine Learning

In this section, some terms which are necessary to understand the implementation phase of the study will be clarified. These terms are Machine Learning, Classification, and Supervised Machine Learning. Their importance emerges from the fact that the whole research is built on them. Therefore, understanding them will surely clarify the steps to be followed during the practical stage and make this research understandable for those who do not have a background in this field.

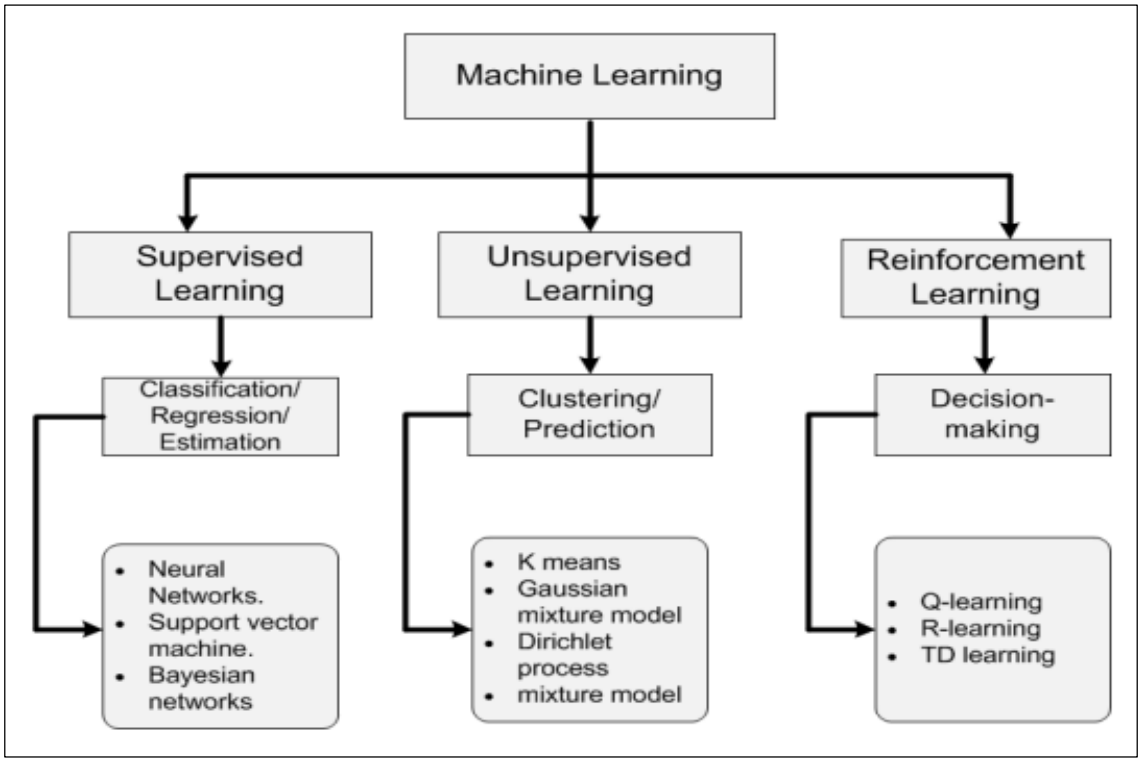
1.4.1 Introduction to Machine Learning

Since the dawn of the technological age, people have fantasized about artificial intelligence. The epitome of intelligence is when an autonomous system can derive generalizations and synthesize ideas from large amounts of complex information without explicit guidance [7]. In simple terms, learning is the acquisition of new knowledge or the improvement of someone's skills. In the first case, acquiring new knowledge is a mixture of processes like grasping significant concepts, understanding what they mean and how they relate to each other, and the specific area of interest. On the other hand, skill enhancement can be viewed in terms of biological terms as strengthening a network of neural connections to better perform a specific function [24]. One of the main branches of artificial intelligence is machine learning: it focuses on developing models and algorithms that enable systems to absorb data, acquire knowledge, and improve their performance over time. It covers many fields, such as

statistics, algebra, data collection, and data processing. Machine learning is a crucial technology in artificial intelligence, where knowledge is absorbed through data training. It's like a massive tree, with machine learning as the trunk and many branches and sub-branches spreading out. Machine learning is divided into three groups: supervised learning, unsupervised Learning, and reinforcement learning (see Figure 1.1) [35].

Figure 1.1

Machine Learning Classification Techniques (Shailaja et al., 2018) [35]



Unsupervised Machine Learning

Unsupervised machine learning involves training algorithms on data without labeled responses. The goal is to find hidden patterns or intrinsic structures in the input data. Here's a step-by-step overview of the methodology [6]:

1. Data Collection: Gather a large set of unlabeled data relevant to the problem domain.
2. Data Preprocessing: Clean and prepare the data by handling missing values, normalizing or standardizing the data, and possibly reducing dimensionality if the dataset is very large.
3. Algorithm Selection: Choose an appropriate unsupervised learning algorithm based on the type of problem (e.g., clustering, dimensionality reduction, anomaly detection).

4. Model Training**:** Train the model on the preprocessed data. The algorithm will learn from the data without supervision, identifying patterns, structures, or relationships within the data.
5. Evaluation: Evaluate the model's performance using appropriate metrics. Since there are no labels, evaluation methods might include visual inspection, silhouette scores for clustering, or reconstruction error for dimensionality reduction.
6. Hyperparameter Tuning: Adjust hyperparameters for the purpose of enhancing the model's performance. This can involve changing the number of chunks in clustering algorithms, the number of components in dimensionality reduction methods, etc.
7. Interpretation and Visualization: Interpret the results by visualizing the learned patterns or structures. This step is crucial in unsupervised learning as it helps understand the data better.
8. Application and Deployment: Apply the model to new data to uncover insights or for further use in downstream tasks like preprocessing for supervised learning models.

Common Unsupervised Machine Learning Techniques and Their Applications

1. Clustering

Clustering is the task of dividing the dataset into groups, where objects in the same group are more similar to each other than to those in other groups.

- K-Means Clustering: This algorithm partitions the data into K clusters by minimizing the sum of squared distances between data points and the centroid of their respective clusters.

Applications

- Customer segmentation for targeted marketing.
- Document clustering for topic discovery.
- Image segmentation [53].

- Hierarchical Clustering: This method builds a tree of clusters by iteratively merging or splitting existing clusters based on a distance metric.

Applications

- Gene expression data analysis in bioinformatics
- Social network analysis
- Image grouping [53].
- DBSCAN (Density-Based Spatial Clustering of Applications with Noise):

This algorithm groups points that are closely packed, marking points in low-density regions as outliers.

Applications

- Spatial data analysis.
- Anomaly detection.
- Noise filtering in datasets [53].

2. Dimensionality Reduction

Dimensionality reduction techniques simplify the data by reducing the number of variables under consideration, making it easier to visualize and analyze [53].

- Principal Component Analysis (PCA)**:** PCA transforms the data into a new coordinate system, reducing dimensions by keeping the components that explain the most variance.

Applications

- Data visualization
- Noise reduction
- Preprocessing for supervised learning
- t-Distributed Stochastic Neighbor Embedding (t-SNE): t-SNE reduces high-dimensional data to two or three dimensions for visualization, preserving local structures.

Applications

- Visualizing high-dimensional datasets
- Identifying clusters in data
- Exploring gene expression data

- Autoencoders**:** Neural networks are used for learning efficient codings of input data, often used for dimensionality reduction.

Applications

- Image compression
- Anomaly detection
- Data denoising

3. Association Rule Learning

Association rule learning discovers interesting relationships (associations) between variables in large datasets [53].

- Apriori Algorithm: Identifies frequent item sets and generates association rules based on them.

Applications

- Market basket analysis
- Recommendation systems
- Medical diagnosis

4. Anomaly Detection

Anomaly detection helps identifying specific items that are different from the rest of the data [53].

- Isolation Forest**:** An ensemble method that works through isolating observations by a random selection of features and splitting of values.

Applications

- Fraud detection
- Intrusion detection in cybersecurity
- Fault detection in manufacturing
- One-Class SVM: A version of the Support Vector Machine which is used as a method of identification of the boundary of normal data points.

Applications

- Novelty detection
- Outlier detection
- Quality control

5. Clustering-Based Anomaly Detection

This method combines grouping and anomaly detection by identifying data points that are not part of any cluster or are in small clusters [6].

Applications

- Outlier detection in datasets
- Rare event detection in time series data
- Identifying unusual patterns in network traffic

These methods help explore and understand data without the need for labeled responses. This enables insights and discovery in various applications [53].

Reinforcement Learning RL

Reinforcement Learning (RL) refers to training a machine so that it can make decisions by itself through performing actions in an environment to maximize cumulative reward. Provided below is a simplified explanation of the training methodology followed by some common RL techniques and their applications [19].

Training Methodology

1. Environment and Agent Setup: the first step includes defining the environment in which the agent operates, including the state space, action space, and reward function. The agent interacts with this environment to learn optimal behaviors.
2. Initial Exploration: The agent starts by exploring the environment, and then it tries different actions to understand their effects. This phase is crucial for collecting information about the state-action-reward dynamics.
3. Policy Learning: The agent uses the collected information to form a strategy, which defines the best action to take in each state. Policies can be deterministic (specific action for each state) or stochastic (probability distribution over actions).

4. Reward Feedback: The machine sends positive reward after each performed action. This maximizes the cumulative reward over time.
5. Value Function Estimation: The agent calculates the value function, which predicts the positive cumulated feedback from each state or state-action pair. Common methods include Q-learning and Value Iteration.
6. Policy Improvement: Based on the value function, the agent improves its policy to choose actions that yield higher rewards. This involves enhancing exploration and exploitation.
7. Convergence: The agent continues to iterate through exploration, learning, and improvement to the point it converges to an optimal policy where the expected cumulative reward is maximized.

Common Reinforcement Learning Techniques and Their Applications

1. Q-Learning

Q-Learning is a model-free RL algorithm that learns the value of actions in each state. It uses a Q-table to store Q-values representing the expected cumulative reward for each state-action pair. The policy is derived by choosing actions with the highest Q-values.

Applications [19]

- Game playing (e.g., Chess, Go)
- Robot navigation
- Autonomous driving

2. Deep Q-Networks (DQN)

DQN is an extension of Q-Learning that uses deep neural networks to approximate the Q-values instead of a Q-table. This allows it to handle large state and action spaces.

Applications

- Complex video games (e.g., Atari games)
- Robotic control tasks
- Financial trading strategies

3. Policy Gradient Methods

Policy gradient methods directly optimize the policy by adjusting the parameters of the policy network to maximize the expected reward. These methods are effective for high-dimensional action spaces and continuous actions.

Applications

- Robotics control
- Natural language processing (e.g., dialogue systems)
- Motion planning

4. Actor-Critic Methods

Actor-Critic methods use value and policy as main factors on which they base their approaches on. The actor updates the policy based on the feedback from the critic, which judges the action by estimating the value function.

Applications

- Autonomous vehicle navigation
- Energy management systems
- Real-time strategy games

5. Proximal Policy Optimization (PPO)

PPO is a type of policy gradient method that uses a surrogate objective function to ensure that updates to the policy do not deviate too much from the previous policy, improving stability and performance [19].

Applications

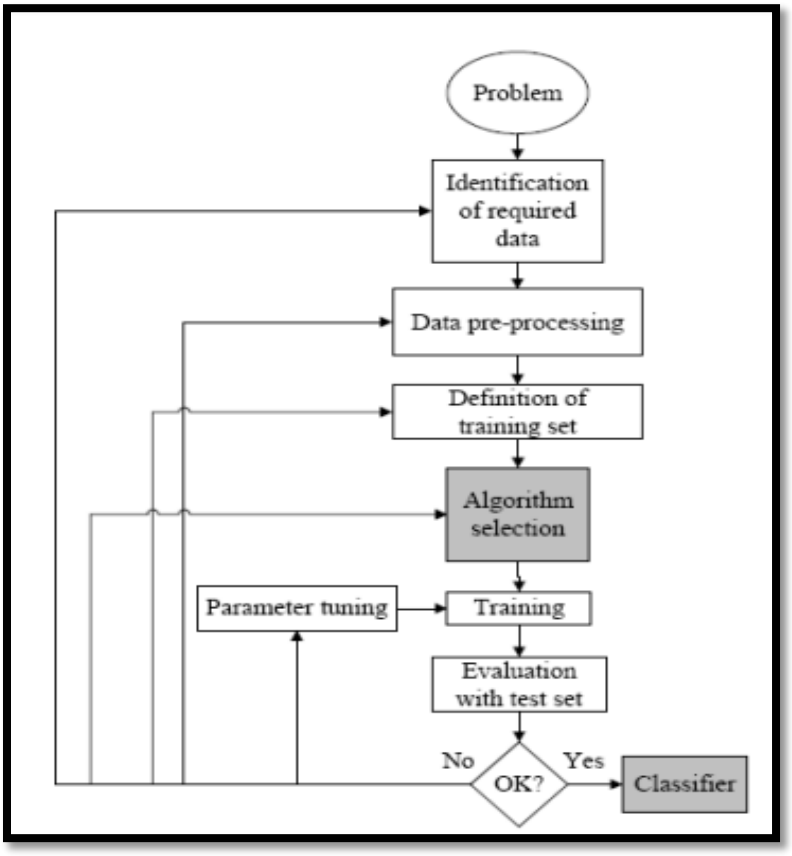
- Robotics
- Real-time decision-making
- Industrial process control

Supervised Machine Learning

Supervised machine learning involves training a model on a labeled dataset, where each training example is paired with an output label. This aims to enable the model to learn a mapping from inputs to outputs that can be generalized to new data [48].

Figure 1.2

The Processes of Supervised Machine Learning [43]



Training Methodology

1. Data Collection: Collect a large dataset which contain input-output pairs. Each item in the dataset includes input features and the corresponding labeled output.
2. Data Preprocessing: Clean and preprocess the data to ensure it is suitable for training. This may include handling missing values, normalizing or standardizing the data, and encoding categorical variables.
3. Splitting the Data: Split the data into training and test sets. The training one is used to train the model, while the test one is used to evaluate its performance.

4. Model Selection: Choose a suitable algorithm based on the problem type (e.g., regression or classification) and the nature of the data.
5. Training the Model: Use the trained data to fit the model. The algorithm adjusts the model parameters to minimize the error between the predicted and actual outputs.
6. Evaluation: Evaluate the model's performance using the test set. Common metrics for evaluation include accuracy, precision, recall, F1 score for classification problems, and mean squared error or mean absolute error for regression problems.
7. Hyperparameter Tuning: Adjust the model's hyperparameters to maximize its performance. This can be done using techniques such as grid search, random search, or more advanced methods like Bayesian optimization.
8. Model Validation: It validates the model using cross-validation to make sure it generalizes well to unseen data and is not overfitting.
9. Deployment: Deploy the trained model to make predictions on new data.
10. Monitoring and Maintenance: Continuously monitor the model's performance in production and update it as needed to handle new data or changing conditions.

Common Supervised Machine Learning Algorithms and Their Applications

1. Linear Regression

Fitting a linear equation to the observed data, so that the linear regression models the relationship between dependent and independent variables.

Applications

- Predicting housing prices
- Forecasting sales
- Analyzing the impact of marketing campaigns

2. Logistic Regression

This type of regression is used to solve binary classification problems. It calculates the likelihood of a given input belonging to a certain class.

Applications

- Spam detection
- Credit scoring
- Disease diagnosis

3. Decision Trees

Decision trees split the data into subsets based on the value of input features. Each node in the tree represents a feature, and each branch represents a decision rule.

Applications

- Customer segmentation
- Fraud detection
- Loan approval

4. Random Forest

Random forest is an ensemble learning method that combines multiple decision trees to improve accuracy and reduce overfitting.

Applications

- Predictive maintenance
- Stock market prediction
- Image classification

5. Support Vector Machines (SVM)

SVMs are used for classification and regression tasks. They help unveil the hyperplane that separates the classes in the feature space.

Applications

- Text categorization
- Face recognition
- Handwriting recognition

6. k-Nearest Neighbors (k-NN)

k-NN is a non-parametric method used for classification and regression. It classifies a data point based on the majority label of its k nearest neighbors [48].

Applications

- Recommender systems
- Image recognition
- Customer behavior analysis

7. Neural Networks

Neural networks are a set of algorithms modeled after the human brain, used to recognize patterns. They are particularly powerful for complex tasks with large datasets.

Applications

- Speech recognition
- Image and video processing
- Natural language processing

Applications of Supervised Machine Learning

1. Healthcare: Predicting patient outcomes, diagnosing diseases, personalizing treatment plans, and analyzing medical images.
2. Finance: Fraud detection, credit scoring, algorithmic trading, and risk management.
3. Marketing: Customer segmentation, churn prediction, recommendation systems, and sentiment analysis.
4. Retail: Demand forecasting, inventory management, price optimization, and personalized marketing.
5. Manufacturing: Predictive maintenance, quality control, supply chain optimization, and defect detection.
6. Transportation: Traffic prediction, route optimization, autonomous driving, and demand forecasting for ride-sharing services.

By utilizing supervised machine learning, businesses and organizations can make data-driven decisions, improve efficiency, and enhance their overall performance in various applications [48].

Why Supervised Training is used?

In high-energy physics, predicting particle types often involves using supervised training because there is labeled training data available. Here are a few reasons why supervised training is suitable for this task:

- Labeled Data

Supervised learning relies on labeled data, where each particle event is connected to a known particle type. In high-energy physics experiments, particle detectors are

specifically created to recognize and categorize particles, supplying labeled data that can be employed to train the neural network.

- Discriminative Task

Anticipating particle types involves a discriminative task, where the network learns to sort input features into distinct predefined classes (particle types). Supervised learning algorithms are specifically crafted to tackle these classification problems by understanding the relationship between input features and their corresponding labels.

- Well-defined Objectives

In supervised training, the goal is clear and measurable. The network is trained to minimize a particular loss function (like cross-entropy) that gauges the difference between predicted particle types and the actual labels. This allows direct optimization and assessment of the model's performance.

- Model Interpretability

Supervised training helps teach the network, which helps expose the characteristics of the network which are used to make predictions. By looking at the learned weights and biases, researchers can understand more about the physical patterns that make each particle unique.

- Flexibility in Network Architectures

Through supervised training, it is possible to use different network structures that work well for predicting particle types. For example, convolutional neural networks (CNNs) can grasp spatial patterns in detector data, and recurrent neural networks (RNNs) can manage sequential information, allowing for modeling intricate connections between input features and particle types.

- Continuous Improvement

Supervised training enhances a structure for making the prediction more accurate. If there's more labeled data later on or if there are improvements in the network setup and training methods, the model can be trained again to make it work better. Even though supervised training has some good points, it's crucial to remember that it needs good-quality labeled data. Being careful with how the data is prepared, managing differences

in class numbers, and dealing with possible biases in the training data are important things to think about when using supervised learning in high-energy physics.

1.5 Classification

The classification question is regarded as the main component of the supervised learning function. Through observation of multiple input-output samples of a specific function, it becomes possible to assign a vector to various classes. Inductive machine learning involves creating a classifier that can generalize to new cases by acquiring a rule set from examples within a training dataset [43].

1.5.1 Classification Algorithm for Jet Classification

In this section, we investigate the methodology used for jet classification. We shed light on the role of neural networks in classifying jets into distinct categories is imperative for various scientific fields. Traditional classification algorithms are widely used for this purpose; however more complex classification algorithms are nowadays used in this field as well. They yield impressive results.

Neural networks is an important field of machine learning and artificial intelligence. It has demonstrated miraculous capabilities in handling enormous complex data. In our study, we harness the capabilities of neural networks, specifically employing the Keras library, to build a model for jet classification.

Keras is a high-level neural networks API written in Python. It offers a user-friendly interface for building and training neural network models. Its integration with TensorFlow (defined as an open-source machine learning framework) provides access to different tools and functionalities for neural network development. Through the use of Keras, streamlining the process of constructing and fine-tuning neural network architectures can be possible to fulfill the task of jet classification.

The process starts with preparing the data after it has been collected and transfered to be into the NN model. To do so, we need to normalize the data, and then feature scale it, and finally refine the quality of the data through augmentation. After the data has been prepared, the neural network model becomes ready to be built using Keras, which is done with careful consideration given to its architecture, including the number of layers, type of activation functions, and optimization algorithms.

When the model is defined, the next thing to do is to train the neural network using labeled jet data. The neural network adjusts its internal parameters to reduce the discrepancy between predicted and real classifications. This done in a process called backpropagation. This training process continues until the model achieves satisfactory performance, ensuring its robustness.

When the training is completed successfully, the trained neural network model is deployed for jet classification. Then, it can efficiently categorize incoming jet data into predefined classes based on learned patterns and features. Using neural networks provides several advantages over traditional classification algorithms. It enhances accuracy, adaptability to complex data structures, and scalability to large datasets.

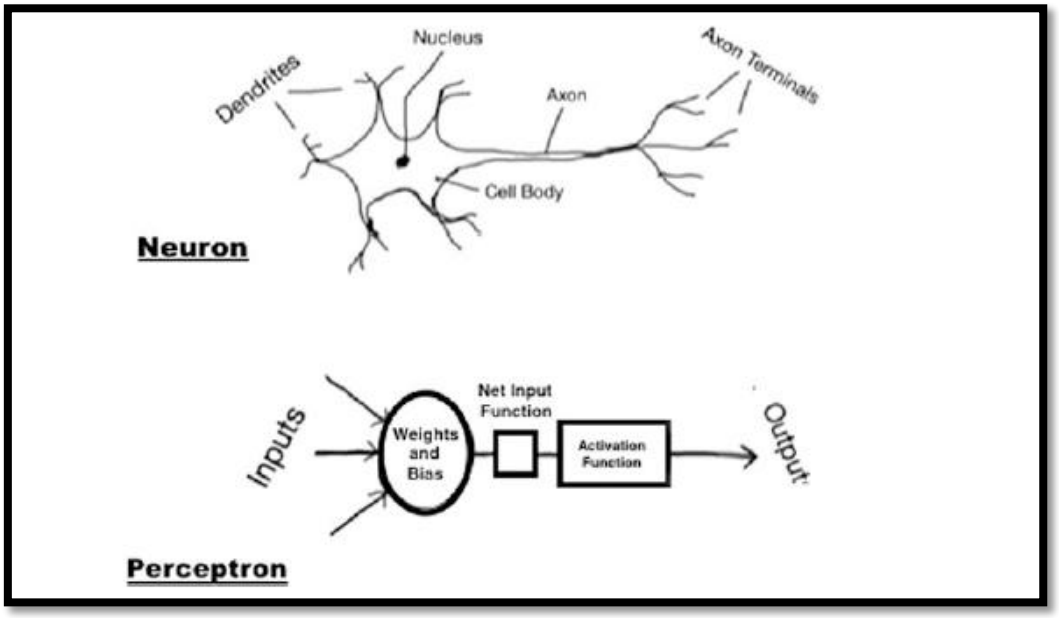
In summary, the employment of neural networks, facilitated by the Keras library, represents a new excellent approach to jet classification. By exploiting the power of artificial intelligence, we aim to advance jet classification and contribute to the broader scientific understanding of particle physics.

1.6 Neural Network

A neural network is a model formed using a certain equation that is generated through certain computations. It is designed imitating features of living neural systems. [37] It is composed of several neurons which together shape the final form of network. The artificial neuron, which is the main particle of the network. It has a mathematical function that imitates the neurons in the human brain, as illustrated in Figure 1.3 below [47].

Figure 1.3

A Biological Neuron and an Artificial Neuron [47]



In the context of classifying jets using neural networks, the 16 features are used as input variables (features used to predict the type of jets). These features are fed as input signals to the neural network and then passed through the different layers in the network to perform computations and extract patterns and useful information.

The learning and training process in neural networks occurs through a stage called training, where the training data (which includes known features and classifications) are fed into the network. The connections' weights between the neurons are adjusted and updated automatically during the training process using algorithms like backpropagation. The aim of the training process is for the neural network to learn to distinguish patterns and relationships in the data, allowing it to predict jet classifications based on the provided features.

Once the training stage is completed and an acceptable level of accuracy is achieved, the neural network can be used to classify new data that was not known or used in the training process. This happens by adding the test data into the network and monitoring the expected output. To predict particle types in high-energy physics, neural networks are repeatedly utilized for the following reasons:

1. As a result of its efficiency, neural networks are capable of capturing non-linear connections between input data and particle kinds. Neural networks are particularly

excellent at learning non-linear mappings in high-energy physics. In non-linear connections, the interactions and behaviors of particles can be very complex.

2. Neural networks can automatically identify and extract related features from the input data. Particle detectors generate enormous quantities of complex data, making it difficult to extract information manually. This result in reducing the reliance on human factor [27].
3. Using parallel processing and GPU acceleration, researchers can be able to efficiently examine analyze massive data, which may be released during high-energy physics experiments.
4. Neural networks perform well in predicting in various fields, of which is high-energy physics. When they are combined with the right structures, they give high levels of accuracy and prediction, which is important in particle identification.
5. They also can adapt to different particle physics scenarios and experiments. They are able to learn a wide array of particle kinds, energies, and detector setups.

Overall, neural networks are a useful technique for predicting particle types in high-energy physics because they can handle complex patterns, learn from vast datasets, and adapt to varied experimental setups [27]. This is what urged us to use Artificial Neural Networks to predict particle type in high-energy physics.

1.7 Activation Function

Activation functions dictate how neural network neurons are activated, a pivotal factor in governing signal propagation across different layers. They affect training speed and the capacity of models to grasp and adapt to data.

Activation functions are important for projects that require jet classification. They help categorize jets into distinct groups. Different activation functions have demonstrated efficacy in various ML contexts without one classified as the "best" one, as all can be beneficial for various objects. Some of such activation functions in high-energy physics may include:

- ReLU Function: Renowned for mitigating the vanishing gradient problem and accelerating training by activating only positive values.
- ELU Function: Similar to ReLU but surpasses it by addressing negative values more effectively, thus enhancing model performance.

- SELU Function: Offers deeper neural networks greater stability and aids in data self-normalization.
- Sigmoid Function: Primarily used in binary classification tasks, it computes the probability of an input belonging to a specific class.
- Tanh Function: It is utilized in hidden layers to refine signal distribution and bolster the model's learning capacity.
- Softmax Function: Employed in the final layers to convert outputs into probability distributions across different classes.
- Softplus Activation Functions: Elevate positive signals within the network, thereby fostering improved training and learning speed.

When experimenting and analyzing the performance of these activation functions, we can decide on the jet classification that suits each function. This endeavor holds the promise of enhancing classification accuracy and boosting the model's efficacy in handling jet classification data.

In this section, we explore the mathematical formulation, derivative, and operational principle underlying each activation function. We explore the benefits and disadvantages of each activation function and conduct practical experiments in Chapter Four to measure their influence on accuracy. Through comparative analysis, we identify the most suitable activation function to achieve optimal performance in our jet classification project.

ReLu Function

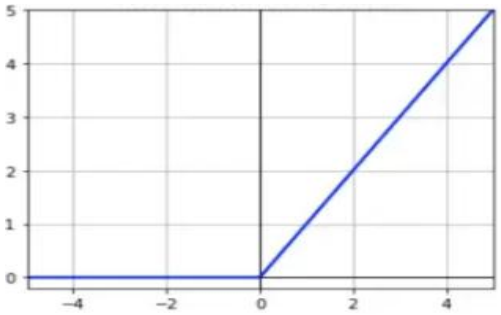
Rectified Linear Units (ReLU) are widely utilized in the hidden layers of Artificial Neural Networks (ANNs), especially in practically all Convolutional Neural Networks (CNN) or Deep Learning models (DP). It is given by [15].

Equation for the ReLU Activation Function is [15]:

$$f(x_i) = max(0, x_i) = \begin{cases} x_i\ , x_i\ > 0 \\ 0, x_i < 0 \end{cases}, \quad \text{..... (1.1)}$$

Figure 1.4

Graph equation for the ReLU activation function [15]

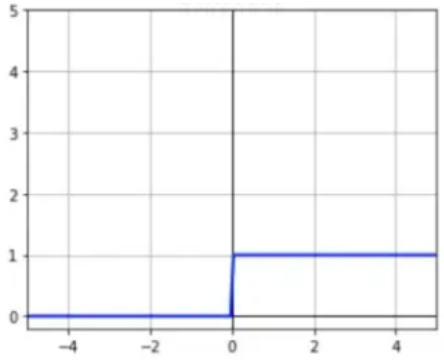


Derivative of equation for the ReLU Activation Function is [15]:

$$f\left(x_i\right) = \begin{cases} 1, x_i > 0 \\ 0, x_i < 0 \end{cases}, \ . \quad \text{..... (1.2)}$$

Figure 1.5

Graph derivative of equation for the ReLU activation function [15]



The Edited Linear Unit can be used to remove nonfunctioning values from our resulting sums, as demonstrated in the formula above. It operates by setting thresholds at 0; hence, f (x) = max (0, x). When x 0, it produces 0; when x 0, it produces a linear function (see Figure 1 for a graphic representation) [3].

We also propose to extend the current use of ReLU as an activation function in each hidden layer to the classification function at the final layer of the network [3].

Pros: [55]

1. It is measurable.
2. It provides precise results.
3. It perfectly suits complex datasets used with neural networks.

Cons:

1. The output value is unlimited, therefore issues could arise if large volumes are fed through it.
2. The neurons may "die" and cease to function if the learning rate is great.
3. As a result of the data's unbalanced processing, the results can be inconsistent.

ELU Function

The Exponential Linear Unit (ELU) activation function, in contrast to the ReLU, has negative values, leading to mean unit activations that are centered around zero. Zero means speed up learning because they bring the gradient closer to the intrinsic gradient of the unit. They found that ELU outperformed ReLu on CIFAR-10, CIFAR-100, Image Net, and other enhanced activation functions [15].

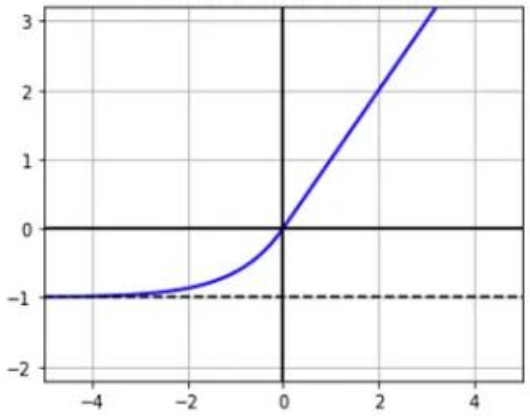
The following formula provides the ELU activation function [15]:

Equation for the ELU Activation Function [15].

$$F(x) = \begin{cases} x, & x > 0 \\ \alpha.\left(e^{x} - 1\right) & x \leq 0 \end{cases} \quad \text{..... (1.3)}$$

Figure 1.6

Graph equation for the ELU activation function [15]

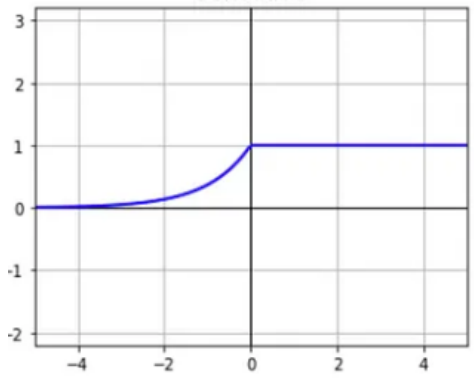


Derivative of Equation for the ELU Activation Function [44]:

$$F(x) = \begin{cases} 1, & x > 0 \\ \alpha . e^{x}, & x < 0 \end{cases}, \quad \text{..... (1.4)}$$

Figure 1.7

Graph derivative of equation for the ELU activation function [44]



ELU is composed of two parts: the identity and the exponential skewing of the negative values. The identity makes up the function's positive component. The hyperparameter controls the convergence value when the inputs are negative infinite [44].

Pros:

1. RELU smooths out abruptly, whereas ELU smooths out gradually till its output equals-α.
2. ELU is a strong replacement for ReLU.
3. Unlike ReLU, ELU can produce negative results.

Cons:

1. The activation's [0, inf] output range may burst if x is bigger than 0.

SELU Scaled Exponential Linear Units

Normalization can occur in three different stages when it comes to neural networks and the training process. The input is initialized and then normalized before being fed into the network model. In the second approach, a particular layer of the network model is used to normalize the batch of data. The last objective of SELU is internal normalization, commonly referred to as self-normalization. A quick convergence can

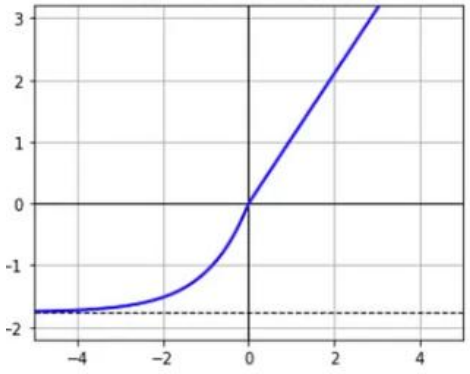
speed up the training process if the input and output of each layer in a network model have a normal distribution (x= 0, x = 1). The SELU activation function is designed with the capacity to convert output into a normal distribution based on this idea. In other words, the function operation helps map the mean and variance of the output [40].

Equations for the SELU Activation Function is [40]: $f: (x) =$

$$\begin{cases} \lambda x, & x > 0 \\ \alpha\lambda.\left(e^{x} - 1\right) & x \leq 0 \end{cases} \text{. ..... (1.5)}$$

Figure 1.8

Graph equation for the SELU activation function [40]

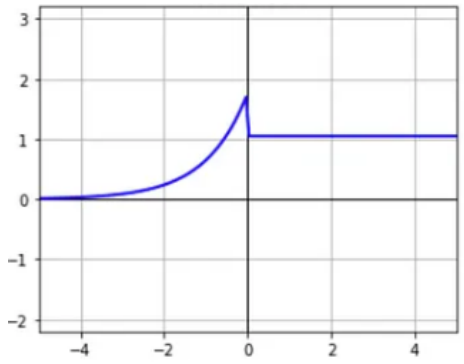


Derivative of equations for the SELU Activation Function is [40]: f(x) =

$$\begin{cases} \lambda, & x > 0 \\ \alpha\lambda\, e^{x}, & x < 0 \end{cases} \text{. ..... (1.6)}$$

Figure 1.9

Graph derivative of equation for the SELU activation function [40]



For inputs with standard scales (zero mean and one standard deviation), the values of the parameters are 1.6732 and 1.0507. The initial weights of the nodes with the SELU activation function should be chosen according to a normal distribution with a mean of zero and a variance of 1/n, where n is the quantity of the input. Moreover, a cutting-edge dropout technique known as Alpha Dropout. The proposed dropout technique randomly sets inputs to this number, which matches the network inputs' tendency to be negative infinite at this point. In order to ensure the self-normalizing property, the inputs are then converted using an affine transformation with parameters linked to the dropout rate, intended mean, and variance. It is recommended that alpha dropout rates be 0.05 or 0.1 [44].

Pros:

1. SELU, like ReLU, is used in deep neural networks because it does not suffer vanishing gradient problems.
2. Compared to other activation functions, SELUs learn faster and efficiently when no additional processing is used. Furthermore, batch normalization precludes comparison of SELUs to other activation functions.

Cons:

1. Because SELU is a relatively new activation function, it is not yet frequently used in practice. ReLU is still the suggested option.
2. Additional study is needed on architectures incorporating SELUs, including CNNs and RNNs, for widespread commercial application.

Sigmoid Function

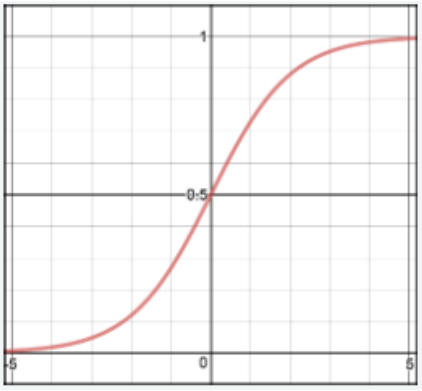
Sigmoid is one of the functions of neural networks, which is provided by equation [15].

Equations for the Sigmoid Activation Function is [15] $F(x) =$

$$\begin{cases} \frac{1}{1+e^{-x}} . \end{cases} \quad \text{..... (1.7)}$$

Figure 1.10

Graph equation for the Sigmoid activation function [15]



Derivative Equations for the Sigmoid Activation Function is [15]*:*

$$F(x) = \frac{e^{-x}}{(1+e^{x})^2}. \qquad \text{..... (1.8)}$$

According to Figure B.1 in Appendix B, the sigmoid function has an S-shaped curve, is nonlinear, and exists between [0, 1]. Although it is monotonic and has a very steep output, when the input was close to 0, small input changes would translate into large output changes. The output, however, tended to 0 or 1, responding very little to the input as the input reached either end. This resulted in "vanishing gradients," which are gradients that are extremely modest or even close to 0.

In this case, the network would reject further learning and the neuron would receive almost no input, which would prevent the weights and subsequent data points from reaching the network. In order to prevent saturation, special care should be used when initializing the weights of sigmoid neurons.

Fortunately, there are ways around this problem, and sigmoid is still frequently used in classification issues, especially in binary classification output layers where the outcome is either 0 or 1. Since the sigmoid function's value only spans the range of 0 to 1, the outcome can be predicted to be 1 if the value exceeds 0.5 and 0 if it does not [40].

Pros [55]:

1. It is monotonic and differentiable.
2. One conceivable use is binary categorization.
3. When we just need to determine the possibility, it can be quite helpful.

Cons:

1. It doesn't offer rigorous values.
2. Due to the problem of a diminishing gradient, the sigmoid function cannot be used in multi-layered networks.
3. The model could get trapped in a local minima during training.

Tanh Function

Tanh function, known as Tangent Hyperbolic Function, is employed in neural networks as the AF. It exhibits the zero centric property [10] and shares similarities with the Logistic Sigmoid function in its properties.

Equations for the Tanh Activation Function (See Figure B.2 in Appendix B) is [10]: $F(x) =$

$$\begin{cases} \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}} \end{cases} . \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad (1.9)$$

Derivative of Equations for the Tanh Activation Function (see Figure B.3 in Appendix B) is [10]:

$$f(x) = 1 - htan(x)^2 \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (1.10)$$

Tanh function squashes the inputs and is nonlinear [18], but only in the range [−1, 1]. Tanh function shares with Logistic Sigmoid function the drawbacks of vanishing gradient and computational complexity. The Tanh and Logistic Sigmoid AFs are mostly impacted by the vanishing gradient [10].

The range [-1,1] concentrates the data, makes learning much easier, and causes the mean of the outputs to be zero or very close to zero; for these reasons, it is usually used in hidden layers of ANNs [15].

Pros:

1. Tanh has a greater gradient (steeper derivatives) than sigmoid.

Cons:

1. Tanh suffers from the vanishing gradient issue.

Softmax Function

It typically appears in the last layer of a neural network. The outputs are often transformed into numbers that, when added together, equal 1. These values will therefore range from 0 to 1 [45].

Pros:

1. It is applicable to multiclass categorization.
2. The range is restricted to values occurring between 0 and 1.

Cons:

1. It is not compatible with a null class.
2. It fails to separate data in a linear manner.

Equation for the Softmax Activation Function (See Figure B.4 in Appendix B) is [47]*:*

$$f(x)_j = \begin{cases} \frac{e^{xj}}{\sum_{k=1}^{k} e^{xk}} \end{cases} for\ j = 1, \dots, k \ . \quad \text{..... (1.11)}$$

The basic purpose of it is to fit the output of neural networks between zero and one. It serves as a representation of the network output's "probability" with absolute certainty.

The normalization is calculated by dividing the output's exp value by the sum of all possible outputs' exp values [47].

Softplus Activation Functions

It was specifically for ensuring that positive premiums are obtained that this new sort of model was established. A convex relationship between an output and one of its inputs can be modeled using the softplus function, which was just recently created. A softplus

unit was added as a final transfer function after we adjusted the Neural Network architecture [45].

Equation for the Softplus Activation Function (See Figure B.5 in Appendix B) is [45]:

$$f(x) = \frac{1}{\beta} * log(1 + exp(\beta * x)) \text{ ..... (1.12)}$$

The softplus layer implements the softplus activation function Y = log (1 + eX), which guarantees correct outputs. It serves as a smooth, continuous alternative to reluLayer. This layer is important when dealing with deep neural networks for actors in reinforcement learning agents. It's particularly useful for establishing continuous Gaussian policy deep neural networks where the standard deviation output must be positive [10].

It's important to note that the choice of activation function is often combined with other design decisions, such as the architecture of the neural network, the optimization algorithm, and the specific features used as input. To determine the most effective activation function for the particle type prediction task, it is important to conduct experimentation and empirical evaluation on your specific dataset.

In the project of jet classification you need to select the appropriate activation functions within the artificial neural network to achieve perfect performance results. Activation functions are fundamental for determining how neurons respond to signals and how signals are sent and received across different layers in the network.

Experimentation with various activation functions such as Softmax, and Selu, enabled me to evaluate their performance and identify the most efficient ones in achieving the highest accuracy in gate classification. The experiments involved the use of these functions as the output layer, hidden layers, and the input layer.

It's worth noting that activation functions affect the training speed and the model's adaptability to data. For example, activation functions like Softmax is good for the output layer, while functions like Sigmoid and Selu are suitable in hidden layers.

Documentation of the experiments results is helpful for understanding performance and drawing important conclusions, reaching to an enhancement of the overall performance of the model and the development of best practices in artificial neural network.

Conclusively, the accuracy results of each experiment were documented in Chapter 3.

1.8 Problem Statements

In high-energy physics, such as particle collision experiments in particle accelerators like the LHC, complex collisions take place, leading to the formation of particle jets. Identification of these jets is important for understanding particles and their interactions. However, traditional jet identification algorithms face challenges in recognizing subtle characteristics and interactions within jets, especially in dense and complex environments.

The use of modern artificial intelligence techniques, specifically artificial neural networks (ANNs) and deep learning algorithms is essential to improving the classification of jets with accuracy and efficiency. By utilizing the Keras library to build classification models, we can develop a program that classifies jets into the specified five categories: light quarks (q), gluons (g), W and Z bosons, and top quarks (t).

Our project aims to explore the optimal structure for artificial neural networks and convolutional neural networks which help in jet classification with the highest possible accuracy. This involves identifying the optimal number of layers, selecting suitable activation functions, and choosing the optimal optimization techniques. The model also aims to identify important features and eliminate redundant features, leading to achievement of the best performance for jet classification.

Through artificial intelligence, jet classification can be enhanced, leading to a deeper understanding of fundamental particles and their interactions. This surely advances scientific progress in the field of high-energy physics.

1.9 Literature Review

This review of the literature reviews a number of studies which were written about jet classification and neural networks. It is worth mentioning here that when deciding on the articles to be reviewed we chose the ones which were written between the years 2011 and 2024.

Moreno et al. (2020) examined the performance of the JEDI-net algorithm. Their study aimed to highlight all-hadronic decays of high-momentum heavy particles generated at the LHC as well as distinguish them from regular jets formed due to quark and gluon hadronization. The algorithm Jet Extraction and Deep Interpretation Network describes the dynamics of jets as a series of interactions between their constituents. It classifies jets based on learned representations from these interactions. Unlike other approaches, JEDI-net achieves its performance without any special handling of sparse input jet representations, extensive preprocessing, particle ordering, or specific assumptions about detector geometry. The models presented yield better results with fewer parameters, offering promising prospects for applications at the LHC [38].

Fraser and Schwartz (2018) researched classifying particle jets based on their electric charge. They used machine learning techniques like convolutional, recurrent, and recursive neural networks. These new methods proved to be more efficient than traditional methods. Their achievement was due to the way they effectively used information about the distance within the jet or its clustering history. Their study also found that both fixed-size representations (like in Convolutional Neural Networks) and compact representations (like in Recurrent Neural Networks) are important in the future of machine learning in particle collider experiments [14].

In a research done by a group called the ATLAS Collaboration (2017), a recurrent neural network (RNN) was used to find a b-jet technique in the ATLAS experiment at CERN's Large Hadron Collider. By processing charged particle tracks related to jets without relying on secondary vertex detection, the RNN-based b-tagging technique can supplement secondary-vertex-based taggers. Unlike prior impact-parameter-based b-tagging systems, which assume that tracks associated with jets are disconnected, the RNN-based b-tagging methodology may exploit spatial and kinematic connections

between tracks that originate from the same b-hadron. A bigger number of input variables is also possible with this new strategy [41].

Xuewen Zeng et al. (2017) suggested the first artificial intelligence-based classification of road traffic. They have also proposed utilizing a convolutional neural network to classify malware communications in a scientific method. This method makes use of road traffic data in the form of photographs, which have been acquired in their natural state. Through eight studies, we discovered that the optimal sort of traffic representation is a session with all layers. The results were good after the procedure had been validated. As a result, it satisfies the accuracy standards [47].

Xianzhen Xu et al. (2020) conducted a study in the area of mechanical fault diagnostic technology because, despite the intricate design of rotating machinery, fault features and causes frequently exhibit ambiguity and complexity, making fault identification challenging. This essay briefly describes the development and use of intelligent technology in the field of equipment defect detection, argues for the superiority of fuzzy neural network technology in this field, and explains the fundamentals of both fuzzy theory and neural network technology. The two benefits and drawbacks of fault diagnosis are examined based on the idea, and it is explained why combining the two is necessary. An improved approach of combining fuzzy theory with neural networks is suggested, and a fuzzy neural network model appropriate for fault diagnostics is created based on prior research on the subject. The experimental findings demonstrate that, in comparison to the widely used neural network and fuzzy theory fault diagnostic approaches, this method may compensate for the drawbacks of fuzzy theory and neural networks alone. It can be applied in the field of rotating equipment defect diagnostics, and it may have a greater diagnostic accuracy [56].

According to Uzair et al. (2020), hidden layers are crucial to neural network performance, particularly for difficult tasks where accuracy and time complexity are the key limitations. It is currently unclear how to calculate the number of hidden layers and the number of neurons in each hidden layer. In this article, we examined the various effects of hidden layers on the network and gave an overview of the use of three hidden layer counts that were determined to be the best in terms of lowering the complexity of the computation process and obtaining high accuracy. If the number of hidden layer implementations does not reach three, this usually gives inaccurate results. However,

above three were shown to be less than ideal in terms of time complexity. Implementing three hidden layers typically results in the best performance in terms of accuracy and time complexity. We conducted a survey of recent work on neural networks based on empirical observations and found that the accuracy of the network is directly impacted by the number of hidden layers, as having fewer hidden layers for complex problems may result in improper network training. In contrast, time complexity rises orders of magnitude more rapidly than accuracy gain when the number of hidden layers exceeds the recommended level of three [54].

In their 2020 study, Karmakar and colleagues emphasized that Artificial Neural Networks (ANNs) are highly effective for learning functions across different types of data—whether discrete, real-valued, or vectors. They noted ANNs' robustness in fields like robotics and speech recognition, where handling imperfect training data is crucial. Their research focused on comparing various training and optimization methods to find the best approach for specific datasets, based on accuracy and loss measures. This study was the first of its kind to systematically compare multiple techniques, considering different learning rates. They concluded that stochastic gradient descent, particularly with the Adam learning rate method and with the use of three different values, proved to be the most effective for predicting breast cancer types. This conclusion followed a detailed comparison of three optimization techniques using the Duke Breast Cancer dataset [27].

Ertuğrul et al. (2018) highlighted the critical importance of identifying the optimal activation function for artificial neural networks since it directly affects the success rates achieved. Unlike analytical methods, determining the best activation function typically involves iterative testing and adjustment processes. Their study proposed a more efficient approach: training a specific activation function for each neuron using linear regression. In this approach, known as the "trained activation function," each activation function gets trained separately for each neuron by using linear regression. This training process utilizes the training dataset, which includes the sum of inputs for each neuron in the hidden layer and the desired outputs. As a result, a unique activation function is generated for every single neuron in the hidden layer. This method was applied to a random weight artificial neural network (RWN) and validated using 50 benchmark datasets. The achieved success rates by RWN using trained activation functions were

significantly higher compared to those using traditional activation functions. The results indicate that this approach is a successful, straightforward, and effective way to determine the optimal activation function, avoiding the need for iterative testing or parameter tuning, applicable to both single-layer and multilayer artificial neural networks [17].

Xinglong Luo et al. (2019) search engines, data mining, machine learning, natural language processing, multimedia learning, speech recognition, recommendation systems, and other related domains have all made extensive use of deep learning. This research examines a multilayer perceptron-based deep neural network and its optimization algorithm. To confirm the accuracy of the model and the optimization process, MNIST handwritten digital datasets were used. It is confirmed that there is a significant association between overfitting and activation functions, network architectures, training epochs, and learning rates. To lower generalization error, the research of overfitting is extremely important. The novel activation function dubbed modified-sigmoid, which is based on the well-known sigmoid function, is proposed in this study. This activation function can successfully increase the model's accuracy and prevent the overfitting issue [29].

Related work

JEDI-net algorithm for jet identification based on interaction networks was developed by Eric A. Moreno and colleagues. This algorithm focuses on improving the discrimination of full hadronic decays of high-momentum heavy particles, such as W, Z bosons, top quarks, from regular jets formed by quarks and gluons. The strength of JEDI-net lies in its ability to handle raw data without the need for complex preprocessing or specific assumptions about detector geometry, leading to improved classification performance with fewer parameters compared to traditional methods.

In the study of JEDI-net, conventional neural networks like DNN, CNN, and GRU were employed and optimized using Bayesian hyperparameter optimization. Optimal configurations were selected for each model based on a wide range of hyperparameters, resulting in enhanced classification efficiency and accuracy.

For instance, it was determined that the traditional neural network (DNN) performed exceptionally well with three hidden layers, each comprising 80 neurons activated by

ELU, along with an optimal dropout rate of 0.11 and batch size of 50. This network achieved a loss of 0.66 and a classification accuracy of 0.76.

Furthermore, other networks such as CNN and GRU were utilized to achieve similar improvements, with optimal configurations selected based on superior performance on the dataset and Bayesian hyperparameter tuning.

These results demonstrate the effectiveness of deep learning models in enhancing the discrimination of heavy hadronic decays through the collection and analysis of jet data resulting from particle collisions in accelerators [38].

Chapter Two

Methodology

This section aims to explain all about the data used in this study. First, the circumstances of the collection of the data set will be clarified. Second, the neural network techniques used in the classification process will be also defined briefly, and the specific technique used in our research, i.e., ANN, will be elaborately clarified in terms of its construction, the number of hidden layers, the activation function, optimizer function, loose function, and epochs number. Furthermore, we will introduce the framework used, Keras, for implementing our neural network model. Lastly, we will conclude this chapter with a brief discussion on the relevant literature in the field.

2.1 The Dataset

2.1.1 Data Sources

The study presents a simulation of jets with an energy of pTe 1 TeV originating from light quarks q, gluons g, W and Z bosons, and top quarks produced by proton-proton collisions. It was created using the data set provided by Zenodo platform [36].

A list of 150 particles, which consists of 16 features, computed from the particle four-momenta. The list is filled with zeros when less than 150 particles are reconstructed, the list is filled with zeros. The constituent list is computed from the coordinates of the momentum(px, py, and pz), the absolute 5 energy, and the pseudorapidity, the azimuthal angle, the distance $\Delta R = \sqrt{\Delta\eta 2 + \Delta\phi 2}$ from the jet center, the relative energy $Erel = Eparticle/Ejet$ and relative transverse momentum $pTrel = pTparticle/\ pTjet$defined as the ratio of the particle quantity and the jet quantity, the relative coordinates $\eta rel = \eta particle - \eta jet$ and $\phi rel = \phi particle - \phi jet$ defined with respect to the jet axis, $cos\ \theta$ and $cos\ \theta rel$ where $\theta rel = \theta particle - \theta jet$ is defined with respect to the jet axis, and the relative η and ϕ coordinates of the particle after applying a proper Lorentz transformation (rotation) [36]. The distributions of these features considering the 150 highest-pT particles in the jet are shown in Figure B.6 in Appendix B.

These distributions are computed for the five jet categories. This jet representation is used to train a ANN classifier.

2.1.2 Data loading and Pre-Processing

- Downloaded the data set from Zenodo platform which consisted of two files: train file and validation file.
- Import the dataset to IDE as (Jupyter).

```
import h5py
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.model_selection import train_test_split

df = 'C:/Users/hp/Desktop/train/jetImage_0_150p_20000_30000.h5'

with h5py.File(df, "r") as f:
# List all groups
print("Keys: %s" % f.keys())
a_group_key = list(f.keys())[0]
b_group_key = list(f.keys())[5]

# Get the train data jetConstituentList
x = list(f[a_group_key])
# Get the train data jets
y = list(f[b_group_key])
```

- Load H5py.Files
- Clear the dataset by replacing the missing values with zeros.
- What to do with Missing Values: Compensate the unknown values Fill of horsepower with horsepower values.

The data contains a set of lists describing the jets to be classified, which are:

['jetConstituentList', 'jetFeatureNames', 'jetImage', 'jetImageECAL', 'jetImageHCAL', 'jets', 'particleFeatureNames']

- I need to use two lists to classify jets using neural networks:
  - jetConstituentList: This is a list containing the individual constituents (particles) of a jet. Each jet can be composed of multiple particles, and this list provides detailed information about each particle.
  - jets: This term refers to a collection or dataset of jets, where each jet is characterized by its constituent particles and various features. This list contains High-Level Features (HLFs) used for jet identification in particle physics research. These features are extracted from jet events and are used in machine learning models to classify and analyze the jets.

- The jet Constituent List is a list of three dimensions (10000,150,16) and jets is a two-dimensional list (10000,59).In the context of jet classification, the last six digits in the 'jets' are reserved for the classification labels of the jet. These labels indicate the type or category of the jet, based on the features and properties extracted from the data. I combined HLFs and particle-level features by reshape the list 'jetConstituentList' to two-dimensional list (10000,2400), then I added the jet information except for the last six digits, so it became (10000,2453).

```
# Combine the two matrices
k = np.zeros((10000, 2453))  # Create a zero-filled matrix to hold the combined data
c = np.reshape(x, (10000, 2400))  # Reshape the jet constituent list to a 2D array

# Iterate over each row in the data
for i in range(10000):
    # Append jet constituent list data and jet information (except for the last six digits) to k
    k[i] = np.append(c[i], y[i][0:53])

Y = []  # Initialize an empty list to hold the classification labels

# Iterate over each row in the data
for i in range(10000):
    # Iterate over the last six digits in the 'jets' data to find the classification labels
    for j in range(6):
        if y[i][53 + j] == 1:  # Check if the classification label is 1
            Y.append(j)  # Append the index of the classification label to Y
```

- Split the dataset into two separate sets – training set and test set.

```
# Split the data into training and testing sets
x_train, x_test, y_train, y_test = train_test_split(k, Y, test_size=0.2, random_state=10)
```

I will now import train_test_split to split our datasets into training and testing datasets. Then, I will import all Machine learning models and I will be using to train and test the data.

2.2 Neural Networks

Neural networks, which are some sort of computer algorithms, are usually referred to as artificial intelligence which start working without knowledge of their task. They depend on learning and enhancing through a trial-and-error process. Therefore, the programmer isn't expected to specify all cases, solutions, and exceptions that the machine could encounter. He, instead, needs to specify the correct foundations which will enable the algorithm to complete the task correctly [37].

Neural networks are programmed in the same way human brain functions [47]. For example, in our research, we utilize the Neural Network model provided by the Keras library to analyze the relationship between jet properties. These properties include 16 attributes such as total energy and number of particles, extracted from datasets obtained from the Zenodo platform. The datasets also include corresponding categories, comprising five classes: light quarks (q), gluons (g), W and Z bosons, and top quarks. Through training data containing these attributes and classes, the neural network determines the category of each jet. This process involves leveraging the unique capabilities of neural networks to understand nonlinear relationships within high-energy physics data and to accurately predict the category of each jet based on its given properties.

The topic of neural networks, appeared in 1943, only became popular in the last decade. This is because of the development in technology, which made it easy for neural networks models to be created. Such models have been proven effective just like human brains or even better in some fields, including image recognition, face authentication, self-driving cars, generation of human-like features for cinematic special effects, speech recognition, automatic translation, medical diagnosis, game playing, and many others which was impossible to be accomplished by nonhumans [46].

Building and Training a Neural Network Model for Jet Classification: A Step-by-Step Guide:

1. Downloading the Data from Zenodo Platform:
2. Building the Model in Neural Network using Keras:
   - Utilize the Keras interface from TensorFlow to construct the neural network model capable of classifying the data into 5 categories.
3. Loading and Preparing the Data:
   - Load the data downloaded from Zenodo and preprocess it for use in training and testing.
4. Training the Model and Evaluating it:
   - First, prepare the data and use it to train the neural network model in Keras.
   - Second, adjust training parameters, such as the number of layers, neurons, activation functions, optimizer type, and number of epochs in order to achieve the

highest possible classification accuracy.

- Third, assess the model's performance on the test data to see if it is accurate for correct classification.

5. Improving Performance:

- First, use the results reached through evaluation to advance the model's performance, For example, this can be done through modifying the model architecture or training parameters.
- Second, repeat the process of training and evaluation with the changes made until a high possible accuracy is achieved.

2.3 Training an Artificial Neural Network

Developing and Training a Neural Network Model for Jet Classification using TensorFlow Environment

In this section, I explain the Python code that builds and trains a neural network model for classifying jets. The aim is to show how we use machine learning to group jets based on their characteristics. We use TensorFlow to create the neural network and teach it using our data. Each part of the process is explained clearly to help you understand and replicate it.

```
import tensorflow as tf
from tensorflow.keras import datasets, layers, models
from sklearn.model_selection import train_test_split
import numpy as np
# Load and preprocess your data (Y, k)
Y = np.array(Y)
k = np.array(k)
# Split the data into training and testing sets
x_train, x_test, y_train, y_test = train_test_split(k, Y, test_size=0.2, random_state=10)
# Build the model
model = models.Sequential([
    layers.Flatten(input_shape=(1, 2453)),
    tf.keras.layers.Dense(1638, activation='selu'),
    tf.keras.layers.Dense(920, activation='sigmoid'),
    # tf.keras.layers.Dropout(0.5),
    # tf.keras.layers.Dropout(0.3),
    tf.keras.layers.Dense(718, activation='sigmoid')
])
# Compile the model
model.compile(optimizer='Adamax',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
# Display model summary
model.summary()
# Train the model
history = model.fit(x_train, y_train, batch_size=200, epochs=200)
# Evaluate the model
test_loss, test_accuracy = model.evaluate(x_test, y_test)
print("Test Loss:", test_loss)
print("Test Accuracy:", test_accuracy)
```

1. Importing Libraries: The code imports TensorFlow and its Keras API, along with other necessary libraries such as NumPy and scikit-learn for data manipulation and model evaluation.
2. Data Preparation: The code snippet assumes that you have preprocessed your data and split it into training and testing sets using `train_test_split` from scikit-learn. It also converts the data into NumPy arrays.
3. Model Definition: The neural network model is defined using the Sequential API from Keras. It consists of layers defined using `tf.keras.layers.Dense`, which are fully connected layers. The input shape is specified in the first layer, which appears to expect input data with shape `(1, 2453)`.
4. Model Compilation: The model is compiled using `model.compile`, where you specify the optimizer, loss function, and evaluation metrics. In this case, 'Adamax' optimizer is used with 'sparse_categorical_crossentropy' loss and 'accuracy' as the metric.
5. Model Training: The `model.fit` function is called to train the model on the training data (`x_train `and `y_train`). Parameters like batch size and number of epochs are specified here. The training process iterates for a certain number of epochs, adjusting the model's weights to minimize the defined loss function.
6. Model Evaluation: Finally, `model.evaluate` is called to evaluate the trained model's performance on the testing data (`x_test` and `y_test`). It returns the loss value and the accuracy of the model on the test data.

2.4 Optimization for Neural Networks

The optimizer oversees the model improvement process by adjusting its parameters during training. Selecting the optimizer is a critical decision in model training, impacting learning speed, stability, and the model's effectiveness in reaching the optimal solution. That's why we concentrated on trying out various optimizers in our jet classification model built with Keras, aiming to identify the best optimizer for achieving high classification accuracy. First, let me provide a simplified explanation of how the optimizer function works in general:

1. Initialization:

- The optimizer is initialized with parameters such as the learning rate, momentum, etc.

- Learning rate: Determines the step size at which the parameters are updated during optimization. It's usually a small positive scalar.
- Momentum: A parameter that accelerates learning in the relevant direction and dampens oscillations.

2. Gradient Calculation:

- During each training iteration, the gradients of the loss function are calculated using techniques like backpropagation.
- Gradients represent the direction and magnitude of the steepest ascent of the loss function.

3. Update Rule:

- The optimizer applies a specific update rule to adjust the parameters based on the computed gradients.
- This update rule typically involves subtracting a fraction of the gradient from the current value of each parameter.
- The learning rate determines the step size of this update.

4. Parameter Update:

- The parameters (weights and biases) are updated according to the calculated gradients and the update rule.
- This process iterates over multiple training examples (or mini-batches) until a stopping criterion is met (e.g., a maximum number of epochs).

5. Optimization Techniques:

TensorFlow/Keras, which we used to build our model for classifying jets, provides a variety of optimizers such as Adam, SGD, RMSprop, and others, each with a unique method for updating and adjusting weights during training. I want to briefly talk about these optimizers, which were used in our jet classification model, used:

Stochastic Gradient Descent (SGD)

It relies on random updates of the learning rate, making it less stable. Despite its simplicity, it is still used in some cases where it can unexpectedly yield good results, especially when the data is small in size [20].

The update follows the following formula [50]:

$$\theta = \theta - \alpha \cdot \nabla J\big(\theta; x(i); y(i)\big) \quad \text{..... (2.1)}$$

Where *{x(i),y(i)}* are the training examples.

This formula is utilized to update the model parameters in the Stochastic Gradient Descent (SGD) algorithm. Here:

- θ represents the model parameters.
- α is the learning rate, which determines the step size taken during the update.

$\nabla J(\theta;x(i);y(i))$ is the gradient of the loss function J with respect to the parameters θ when using the training example $(x(i),y(i))$.

Using this formula, each parameter θ is updated by subtracting a scaled multiple of the learning rate α multiplied by the gradient of the loss function with respect to that parameter. This leads to the parameters being updated in a direction that gradually reduces the loss value during training.

Since this process repeats for each example in the dataset, this update is applied to the model parameters repeatedly, ultimately resulting in changes to the model parameters that reduce the loss value overall [50].

RMSprop

It updates the learning rate using a moving average ratio of squared gradients, which helps improve training stability and prevent divergence in local minima. It is particularly suitable for problems that require addressing drastic deviations in gaps between batches during training [59].

The updated equation can be performed as:

$$E[g^2](t) = \beta E[g^2](t-1) + (1-\beta)\left(\frac{\partial c}{\partial w}\right)^2 \text{ ..... (2.2)}$$

$$w_{ij}(t) = w_{ij}(t-1) - \frac{\eta}{\sqrt{E[g^2]}}\frac{\partial c}{\partial w_{ij}}[46] \text{ ..... (2.3)}$$

where E[g] is the moving average of squared gradients, $\delta c/\delta w$ is gradient of the cost function with respect to the weight, η is the learning rate and β is moving average parameter [46].

Follow The Regularized Leader (FTRL)

Relies on an estimation of positive and negative weight regularization to determine the learning rate.

Used to handle binary and multiclass classification problems.

Implemented to deal with sparse features in large-scale datasets.

FTRL weight update algorithm:

$$w^{t+1} = \text{argmin}w\left\{G^{(1:t)}.W + \lambda_1\|W\|_1 + \frac{\lambda_2}{2}\ \|W\|_2^2 + \frac{1}{2}\sum_{S=1}^{t}\sigma^{(s)}\|W - W^{(s)}\|_2^2\right\}\text{....(2.4)}$$

Where G(1:t) is the average of previous sub gradients. Another cool thing about FTRL is that the learning rate is different for different dimensions. If the training data deems that the dimension *i* needs to take a wider step than dimension *j*, the below learning rate will accommodate such updates.

Learning rate for FTRL [59].

$$\sum_{s=1}^{t}\sigma^{(s)} = \eta_i^{(t)} = \alpha/\left(\beta + \sum_{s=1}^{t}(g_i^{(s)})^2\right) \text{ ..... (2.5)}$$

Adagrad

This optimizer changes the learning rate. It changes the learning rate **'η'** for each parameter and at every time step **'t'.** It's a type second order optimization algorithm. Suitable for sparse datasets where certain features are less common [58].

.It works on the derivative of an error function.

$$g_{t,i} = \nabla_\theta J(\theta_{t,i}) \quad \text{..... (2.6)}$$

A derivative of loss function for given parameters at a given time t.

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,ii}+\epsilon}} \cdot g_{t,i} \quad \text{..... (2.7)}$$

Update parameters for given input i and at time/iteration t.

η is a learning rate that is adjusted for a given parameter $\theta(i)$ at a specific time based on previous gradients calculated for the given parameter $\theta(i)$.

We accumulate the sum of squares of the gradients with respect to θ(i) up to time step t, while ϵ is a smoothing term to prevent division by zero (typically on the order of 1e−8). Interestingly, without the square root operation, the algorithm performs significantly worse. It performs large updates for infrequent parameters and small steps for frequent parameters [58].

AdaDelta [54]

Adadelta extends AdaGrad to address its issue of diminishing learning rates. Unlike AdaGrad, which accumulates all previous squared gradients, Adadelta restricts the accumulation window to a fixed size w. Additionally, Adadelta employs an exponentially weighted moving average instead of summing all gradients.

$$E[g^2](t)=\gamma.E[g^2](t-1)+(1-\gamma).g^2(t) \quad \text{..... (2.8)}$$

We set γ to a similar value as the momentum term, around 0.9.

Update the parameters

$$E[g^2]_t = \gamma E[g^2]_{t-1} + (1-\gamma) g_t^2, \quad \text{..... (2.9)}$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t+\epsilon}} \cdot g_t. \quad \text{..... (2.10)}$$

Adam

It relies on the concept of Adaptive Learning Rate, where it updates the learning rate based on estimates of first moment (momentum) and a sequential estimate of squared gradients *M(t)*. Adam is among the most commonly used and effective optimizers in most applications due to its ability to handle various types of data and problems. *M(t)* represents the first moment, which is the mean, and V(t) represents the second moment, which is the uncentered variance, of the gradients, respectively [1].

$$\widehat{m}_t = \frac{m_t}{1-\beta_1^t} \quad \text{..... (2.11)}$$

$$\hat{v}_t = \frac{v_t}{1-\beta_1^t} \quad \text{..... (2.11)}$$

To update the parameter, we calculate the mean of *M(t)* and *V(t)* so that the expected value of *m(t)* is equal to the expected value of *g(t)*, where *E[f(x)]* represents the expected value of *f(x)*.

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t}} \widehat{m}_t \quad \text{..... (2.13)}$$

The values for $\beta 1$ is 0.9, 0.999 for $\beta 2$, and ($10 \times exp(-8)$) *for* 'ϵ '.

Nadam

NADAM, short for Nesterov-accelerated Adaptive Moment Estimation, merges the concepts of Adam and Nesterov Momentum. The update rule can be expressed as follows [50]:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t}+\epsilon}\left(\beta_1 \widehat{m}_t + \frac{(1-\beta_t)g_t}{1-\beta_1^t}\right) \text{..... (2.14)}$$

Convergence Monitoring

- The optimization process is monitored to ensure that our model's accuracy in classifying jets increases over time. We recorded the accuracy results in Chapter 4 using these different optimizers.

In short, the optimizer's role lies in iteratively adjusting parameters to minimize the loss function, thereby enhancing the model's performance in the given task. For this reason,

we experimented with several different optimizers mentioned above to determine the best ones that achieve the highest accuracy in classifying jets. Here are the programming steps we took in our project:

```
model.compile(optimizer = 'Adamax',
loss = 'sparse_categorical_crossentropy',
metrics = ['accuracy'])
model.summary()
```

1. optimizer='Adamax': In this setting, the optimizer parameter is configured as 'Adamax'. Consequently, during the training process, the Adamax optimizer is employed to adjust the weights of the neural network based on the gradients derived from the loss function. It's worth noting that we experimented with approximately six different optimizers, as explained above. The fluctuations in accuracy resulting from these various optimizers were recorded in Chapter three.
2. loss='sparse_categorical_crossentropy': This parameter defines the loss function that will be used to calculate the model's error during training. In this case, 'sparse_categorical_crossentropy' is chosen, which is suitable for multi-class classification problems where the target labels are integers.
3. metrics=['accuracy']: This parameter specifies the evaluation metric(s) that will be used to monitor the performance of the model during training and testing. Here, the accuracy metric is selected, which measures the proportion of correctly classified samples.

In summary, this line of code configures the model for training by specifying the optimizer, loss function, and evaluation metric.

2.5 Overfitting

Overfitting occurs when a machine learning model records unwanted sounds from the environment in the training phase as if they were the target data. This takes place when the model happens to be complex due to the large amount of data input. As a result, the model performs well on the training data but fails to perfectly act to new, unseen data [47].

Results with a Single-Layer Model

When building the model using only one layer, the results showed that test accuracy was significantly lower compared to training accuracy, indicating some overfitting but to a lesser degree compared to the three-layer models. For example, when using the Softmax activation, the training accuracy was 0.7346 while the test accuracy was 0.628. Similarly, with the Sigmoid activation, the training accuracy was 0.7466 while the test accuracy was 0.6295.

Table 2.1

Single-Layer Model Performance

Activation function	Train Accuracy	Test Accuracy
Softmax	0.7346	0.628
Sigmoid	0.7466	0.6295

Results with Three-Layer Models using Different Activations

When building the model using three layers with different activations, the results were as follows:

Table 2.2

Three-Layer Model Performance with Different Activation Functions

Activation function	Train Accuracy	Test Accuracy
Softmax, Softplus, Softmax	0.786	0.736
Sigmoid, Sigmoid, Sigmoid	1.0	0.702
Softplus, Tanh, Sigmoid	1.0	0.72049
ReLU, Sigmoid, Sigmoid	1.0	0.725
ELU, Tanh, Sigmoid	0.9998	0.735

Analyzing these results, it is clear that there is a significant difference between training accuracy and test accuracy in some models, indicating the presence of overfitting. For instance, with Sigmoid activations only, the training accuracy was 1.0, while the test accuracy was 0.702, meaning the model learned very well on the training data but failed to generalize to new data.

Similarly, with ReLU and Sigmoid activations, as well as ELU, Tanh, and Sigmoid activations, the models achieved near-perfect training accuracy, while the test accuracy was much lower. This reinforces the idea of overfitting, where the model learns the patterns in the training data with high accuracy but fails to apply them to unfamiliar data.

To address overfitting, various techniques can be employed [57]:

1. EarlyStopping: This technique allows controlling the model's performance on a validation set during training and stopping the training process when the performance stops improving, thereby preventing the model from unfitting to the training data excessively [9].
2. Dropout: Dropout is a regularization technique that randomly drops (sets to zero) a proportion of the neurons in a layer during training. This helps prevent the co-adaptation of neurons and encourages the network to learn more robust features.
3. L2 Regularization: This term refers to the weight decay, which adds a penalty to the loss function that penalizes large weights in the model. It controls the model disallowing it from dealing with unnecessary complex patterns.

In my project, I applied these strategies to mitigate overfitting and improve the performance of the generalizability of the machine learning model. By incorporating EarlyStopping, Dropout layers, and L2 regularization, I ensured that the model learned meaningful patterns from the data while avoiding excessive fitting to noise or irrelevant details. This approach helped enhance the model's ability to make accurate predictions on new, unseen data.

I want to clarify the steps I took to eliminate overfitting in my Gates classification program.

1. Early Stopping: In this part, I defined a callback named EarlyStopping from TensorFlow. It monitors the model's loss value on the validation set while the training process. If the loss value does not improve after training a certain number of epochs, the training process automatically stops. This helps prevent overfitting.

```
# Define EarlyStopping callback
early_stopping = EarlyStopping(monitor='val_loss', patience=10, verbose=1)
```

2. Training with Early Stopping: After the callback had been defined, it was added to the model training process using the fit() function from TensorFlow. This ensures that the training stops if there is no improvement in the loss value on the validation set after a certain number of epochs.

```
# Fit the model on the dataset with early stopping callback
history = model.fit(x_train, y_train,
                    batch_size=200,
                    epochs=200,
                    validation_data=(x_test, y_test),
                    callbacks=[early_stopping])
```

This part stops the training process when the model becomes exposed to overfitting, which helps to avoid excessive fitting to the training data.

2.6 Using Feature Importance

[Hooker, S., Erhan, D., Kindermans, P. J., & Kim, B. (2018). Evaluating feature importance estimates. *arXiv preprint arXiv:1806.10758*, *2*.]

In my classification model I used feature extraction with SelectKBest from the scikit-learn library in order to enhance the model's effectiveness. This approach allowed me to pinpoint the most important features within the training data. Employing the classification method, I identified the top 10 features that significantly contribute to the model's predictive power. Subsequently, I restructured the data to include only these selected features. Following this, I constructed the classification model using the identified important features and proceeded to the training stage. Upon completion of the training process, I assessed the model's performance using the crucial features on the test dataset. These steps highlight my efforts to enhance the model performance using important features [36].

```
# Extracting important features
selector = SelectKBest(score_func=f_classif, k=100)
# Select the top 10 features
selected_features = selector.fit_transform(x_train, y_train)

# Determining the features
num_features = selected_features.shape[1]

# Preparing the data
x_train_selected = selected_features
x_test_selected = selector.transform(x_test)
```

These comments describe the functionality of each step in the code snippet:

- selector is created using SelectKBest from scikit-learn, specifying f_classif as the scoring function and selecting the top 10 features.
- The selected features are stored in selected_features.
- num_features stores the number of selected features.
- Finally, the training and testing data are prepared using only the selected features. x_train_selected contains the selected features for training, and x_test_selected contains the selected features for testing.

2.7 Keras

Python-based, open-source Keras is a high-level neural network API. It can operate on top of Theano, Microsoft Cognitive Toolkit, or TensorFlow. Keras was created to facilitate quick experimentation and assist in the development of applications quickly. With Keras, one can move from conception to completion as quickly as possible. Due to the broad community support, Keras supports almost all current data science models involving neural networks. Layers, batch normalization, dropout, objective functions, activation functions, and optimizers are just a few examples of the building pieces that are frequently utilized. Additionally, Keras enables users to create models for the web, the Java Virtual Machine (JVM), and smartphones (Android and iOS).You may use to train your models using your GPU without any change in code [11].

Features of Keras

Deep learning is considerably simplified by a number of features and tools in Keras. The following is a list of some of these:

1. Both the CPU and GPU can run it without any issues.
2. It's simple to export models to servers, browsers, embedded devices, and other platforms.
3. Users will find research and deployment to be less challenging because it is adaptable and consistent.
4. It supports many forms of neural networks, including CNNs and RNNs
5. There is a community for users to support one another as well as rich material for additional study.

Thus, Keras offers the user trustworthy assistance and potent tools that can be incorporated into a program to design, train, test, and deploy deep learning models [17].

Applications of Keras

1. Keras is used to create deep models that can be produced on smartphones.
2. Distributed training of deep learning models is also done with Keras.
3. Organizations like Netflix, Yelp, Uber, and others employ Keras.
4. Keras is widely applied in deep learning to develop and deploy quick working models [20].

Chapter Three

Results and Discussion

In the preceding chapters, we delved into the process of constructing a neural network model for classifying jets into the five different categories. We contributed to enhancing the model's performance by adjusting various influencing factors, including the neural network structure, activation functions, optimization methods, and the number of training epochs. Additionally, we adopted the strategy of controlling for overfitting to prevent its occurrence, and identified the significant features to improve the model's performance.

In this chapter, we will document the results obtained from the model's performance across changes in the neural network structure. We will segment the chapter into three sections based on the number of layers: starting from one layer, progressing to two layers, and culminating in three layers. In each section, we will determine the optimal choices for activation functions, optimization methods, and other factors, based on experimentation and analysis. Furthermore, we will analyze and compare the achieved results to guide future steps and enhance the model's performance.

Before diving into that, I'll clarify the details of the device and environment used to build our neural network for classifying jets using TensorFlow/Keras:

Device Details

- Device Name: DESKTOP-NVRCKJD
- Processor: Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz 1.99 GHz
- Installed RAM: 8.00 GB (7.89 GB used)
- Device ID: CA54635A-A46B-417A-9673-5C1657B47539
- Product ID: 00331-10000-00001-AA202
- System Type: 64-bit operating system, x64-based processor
- Pen and Touch: Touch input or pen input not available for this display device

Environment

- Jupyter Notebook with Python 3 is used as the interface for model development and experimentation.

Libraries Used

1. TensorFlow: Used as the core library for building and training neural networks.
2. Keras: High-level API used for building neural networks, integrated with TensorFlow.
3. h5py: Library for interacting with HDF5 format data files.
4. numpy: Library for numerical computing in Python.
5. matplotlib: Library for creating still, animated, and interactive visualizations in Python.
6. pandas: Library for data manipulation and analysis.
7. scikit-learn (sklearn): Library for machine learning tasks such as classification, regression, clustering, and dimensionality reduction.

3.1 Single-Layer Neural Network

In our experiment, we constructed a Neural Network using only one layer. We found that the most optimal outcomes were achieved by employing the Adam optimization algorithm with a learning rate of 0.001, and conducting training for 300 epochs.. These values are shown in the code below.

```
import numpy as np
from sklearn.model_selection import train_test_split
import tensorflow as tf
```

3.1.1 Different Activation Functions

In the previous chapter, we discussed the topic of popular activation functions in artificial neural networks, along with the equations that define these functions and their role in the process of deep learning. Additionally, we mentioned that we constructed a neural network using Keras with a single layer and invoked various activation functions. In this section, we recorded the accuracy values for each activation function used in the Gates classification model to understand how each function impacts the network's performance as outlined in table 3.1. Subsequently, I plotted the results in the form of a graph to illustrate the differences in performance among each activation function as outlined in Figure B.8, Appendix B.

Table 3.1

Accuracy of Different activation Functions of Single-Layer Neural Network

Activation function	Accuracy
ReLu	0.1075
ELU	0.2405
SELU	0.1545
Sigmoid	0. 5775
Tanh	0.2080
Softmax	0.6140
Softplus	0.2700

As shown in the previous table, the best result was obtained when Softmax activation function was used (0.6140).

Figure B.8 in Appendix B reflects the application and accuracy of different activation functions.

Of course, in this section, our aim is to achieve the highest accuracy in classifying the jets. Based on the previous table, the best result was obtained when using the Softmax activation function (0.6140), with a model built using a single-layer neural network. Additionally, the Adam optimizer, as listed in the previous table, was employed.

Building upon this information, we will work on improving the model architecture, selecting the appropriate optimizer, and activation function to achieve the highest possible accuracy in gate classification.

The code has been executed multiple times, and we have consistently evaluated the accuracy of the highest accuracy activation functions, namely Softmax and Sigmoid, to determine which ones repeatedly yielded the best results. as explain in table 3.2.

Table 3.2

Accuracy of Different Experiments of Single-Layer Neural Network

Activation function	Exp1	Exp2	Exp3	Exp4	Exp5
Softmax	0.6140	0.5860	0.6380	0.6165	0.628
Sigmoid	0.5775	0.5280	0.6245	0.5715	0.6295

In our research, we applied Anova: Single Factor as a statistical analysis method. This allowed us to investigate and analyze the variation among two specific groups, namely Softmax and Sigmoid. Through this analysis, we aimed to determine whether there were any statistically significant differences between these groups with respect to our chosen variables.

Table 3.3

Anova: Single Factor of Single-Layer Neural Network

SUMMARY						
Groups	*Count*	*Sum*	*Average*	*Variance*		
Softmax	5	3.065	0.613	0.000343		
Sigmoid	5	2.931	0.5862	0.001755		
ANOVA						
Source of Variation	*SS*	*df*	*MS*	*F*	*P-value*	*F crit*
Between Groups	0.001796	1	0.001796	1.711256	0.22715591	5.317655072
Within Groups	0.008394	8	0.001049			
Total	0.01019	9				

The p-value and critical F-value help us determine whether the differences in accuracy between Softmax and Sigmoid are significant or if they could have happened by chance.

When the p-value (0.227) is higher than the typical threshold of 0.05, it means there isn't enough evidence to confidently say that one activation function is better than the other in terms of accuracy. This suggests that their performance is similar.

Similarly, when the calculated F-statistic (1.711) is less than the critical F-value (5.318), as in your case, it supports the idea that there's no statistically significant difference between Softmax and Sigmoid regarding accuracy.

So, to put it simply, the analysis indicates that there's no strong evidence favoring one activation function (Softmax or Sigmoid) over the other in terms of accuracy. They both to perform similarly in my study. Depending on our specific needs, either Softmax or Sigmoid could be a good choice when building a neural network.

Regarding the training accuracy when using the Sigmoid and Softmax activation functions, it is shown in the following table:

Table 3.4

Train Accuracy of Different Activation function of Single-Layer Neural Network.

Activation function	Train Accuracy
Softmax	0.60140
Sigmoid	0.5975

3.1.2 Different Optimizers

The choice of optimizer is problem-dependent, and there is no universal best optimizer. The optimal choice may vary based on the type of your problem, architecture, dataset, and other factors. We need to iterate and fine-tune your choice based on empirical results and analysis.

Based on the previously said, we experimented different optimizer functions in order to know which one of them gives more accurate results. Based on the previous results, I used the same values of variables:

Number of hidden layers: one, activation function: Softmax, number of training epochs = 300. So, as shown in the table below, I have experimented different optimizer functions and wrote the accuracy of each in the Table 3.5.

Table 3.5

Accuracy of Different Optimizer Function of Single-Layer Neural Network.

Optimizer	Accuracy
Adamax	0.6335
Adam	0.6240
Nadam	0.6145
Ftrl	0.6105
Adagrad	0.6100
RMSprop	0.6000
Adadelta	0.4805
AdaDelta	0.4845
SGD	0.4575

Based on the results shown in the table 3.5 above, it appears that the best optimizer function yielding the highest accuracy in classifying gates is Adamax with an accuracy of 0.6335, followed by the Adam optimizer with an accuracy of 0.6240. Therefore, we can conclude that using the Softmax activation function with either the Adamax or Adam optimizer can lead to the best results in constructing a single-layer artificial neural network for gate classification with the highest accuracy.

Figure B.10 as shown in Appendix B reflects the application and accuracy of different optimizer functions.

Using the Adamax optimizer function in a neural network to predict particle types in high-energy physics can yield good results due to several advantages offered by the Adamax algorithm. The following points explain the reasons why Adamax optimizer can be of great use in this context:

1. Rate of learning adaptability: Adamax is a type of algorithm that updates the learning rate as it learns, improving its learning speed based on past learning experiences. Its adaptability helps improve its performance steadily, especially when processing large chunks of information, which is the case in high-energy physics data.
2. Robustness to learning rate sensitivity: The optimizer is special in its ability to learn fast. The initial learning rate does not affect the optimizer's ability of enhancing its learning rate. It doesn’t get bothered much by the initial learning rate, which makes it unique compared to others like plain Adam. This strength point makes it perfect in choosing the right learning rate for your specific particle physics prediction task.
3. Processing of unbounded gradient norms: Adamax is excellent in dealing with individual differences in particle types. Thus, it is able to handle these differences and changes and learns and improves during processing, making it a perfect choice for predicting particle types. Unlike traditional optimizers, Adamax optimizer can handle the challenge well.
4. Rate of convergence: Its unique characteristics saves time and power. This is important and makes a difference in dealing with large complex data.
5. Requirements for minimal memory: Despite its unique characteristics and performance the optimizer does not need too much computer memory, which does not add further requirements to its users, making it more practical in the work place.

6. Wide range of uses and excellent results: Analysis of visual and written materials is one of Adams's uses. Its efficiency in such areas suggests its ability to function well predicting particle types in high-energy physics.

Several different optimizers were used during model training to achieve the best performance. The following results show the training accuracy achieved for each optimizer:

Table 3.6

Train Accuracy of Different Optimizer Function of Single-Layer Neural Network

Optimizer	Train Accuracy
Adamax	0.6011
Adam	0.5982
Nadam	0.5652
Ftrl	0.5607
Adagrad	0.5512
RMSprop	0.5400
Adadelta	0.4850
AdaDelta	0.7245
SGD	0.4012

From the table above, it can be observed that the highest training accuracy was achieved using the Adamax optimizer at 0.6011, while the lowest training accuracy was achieved using the SGD optimizer at 0.4012. These results indicate that the Adamax optimizer was the most effective in improving the model's performance on the training data.

To prevent overfitting, techniques such as Dropout and Regularization were used, along with Early Stopping to halt training when performance improvement on the validation set stopped. The learning rate was set to 0.0001 to achieve better stability during training.

3.1.3 Varying Epochs

Deciding on how many epochs needed for training a neural network is important in determining the impact of the model's performance and convergence. The term "number of epochs" refers to the times the complete training dataset cycles through the neural network during the training process.

We used the same values of variables:

Number of hidden layers: one, activation function: Softmax, Optimizer function = Adamax. So, as shown in the table below, I have experimented different values of number of epochs and wrote the accuracy of each in the table.

Table 3.7

Accuracy of Varying Epochs of Single-Layer Neural Network

Number of epochs	Accuracy
50	0.6055
150	0.6160
200	0.6265
300	0.6375
400	0.6175
700	0.6110
1000	0.6110

The best accuracy obtained when the number of epochs = 300

3.2 Two-Layer Neural Network

A neural network with two layers was established. It has been determined that the most optimal outcomes were achieved by utilizing the subsequent variable values: the Adam optimization algorithm, and a total of 300 training epochs. These specific values are visually represented in the accompanying code.

```
Y=np.array(Y)
k=np.array(k)
x_train,x_test,y_train,y_test= train_test_split(k,Y,test_size=0.2,random_state=1000)

model = tf.keras.models.Sequential([
layers.Flatten(input_shape=(1,2453)),
tf.keras.layers.Dense(120, activation='sigmoid'),
tf.keras.layers.Dropout(0.5),
tf.keras.layers.Dense(60, activation='sigmoid')
])
model.compile(optimizer = 'adamax',
loss = 'sparse_categorical_crossentropy',
metrics = ['accuracy'])
model.summary()
model.fit(x_train,y_train, epochs = 300)
```

3.2.1 Different Activation Functions

As indicated in the preceding section, there isn't a single activation function that universally performs as the 'optimal' choice for predicting particle types in high-energy physics. Typically, researchers engage in trial and error with various activation functions, network structures, and hyperparameters to identify the configuration that yields the most favorable outcomes tailored to their specific objectives. Thus, as depicted in the tabular format shown in Table A.1 in Appendix A, I conducted experiments using diverse activation functions and recorded the corresponding accuracies for each.

The activation functions that demonstrated the most favorable accuracy outcomes underwent multiple rounds of experimentation to ensure the reliability of their performance (refer to the Table 3.8 below).

Table 3.8

Accuracy of Different Experiments of Two-Layer Neural Network

Input Layer	Hidden Layers	Exp1	Exp2	Exp3	Exp4	Exp5	Exp6	Average
ELU	Softmax	0.7255	0.7345	0.7400	0.7440	0.7380	0.7430	0.7375
ELU	Sigmoid	0.7235	0.7380	0.7440	0.7370	0.7425	0.7430	0.738
Selu	Sigmoid	0.7230	0.7405	0.7435	0.7520	0.7435	0.7370	.7399
ReLu	Softmax	0.7060	0.7460	0.7350	0.7440	0.7350	0.7350	.7335
Softplus	Sigmoid	0.7060	0.7305	0.7260	0.7250	0.7370	0.7415	.7276
Sigmoid	Sigmoid	0.7020	0.7525	0.7495	0.7545	0.7515	0.7480	0.743

Similar to the information presented in the previous table, a set of results exhibited closely values. This suggests that these options can be considered the most favorable based on our observations when employing a two-layer configuration.

Table 3.8 is used to present the results of our statistical analysis using the Anova: Single Factor method. This methodology was chosen to evaluate and compare the performance of different options within the two-layer configuration shown in the previous table 3.8. It allows us to decide whether there are any significant variations between these options with respect to our chosen variables, providing valuable insights into which choices are most beneficial for this specific configuration.

Given the F statistic (1.074472) being lower than the critical F value (2.533555) at a significance level of 0.05, and the p-value (0.394177) being greater than 0.05, you should not reject the null hypothesis. This means that there are no statistically significant differences between the means of the six groups. In simpler terms, there isn't enough evidence to conclude that the group averages are different from each other.

Therefore, based on these results, you can consider this group to be the best option. This group's combination, as shown in the table 3.10, resulted in better accuracy when constructing a two-layer neural network.

3.2.2 Different Optimizers

In our experimentation process, we tested a bunch of optimizer functions to determine which one produced the most accurate results. Based on the insights gained in previous tests, we adopted the ELU (In input layer), Softmax (In hidden layer) activation function. I've maintained uniformity in the variable settings, except for one modification: setting the number of hidden layers to two. With the training duration is 300 epochs. Table A.10 in Appendix A shows the results of my experiment with the various optimizer functions, along with their corresponding accuracy values.

The highest accuracy is achieved with the "Adamax" optimization algorithm, which has an accuracy of 0.7415. On the other hand, "Adadelta" and "AdaDelta" seem to have the lowest accuracy values, both around 0.4980 and 0.4985.

In summary, in my research the accuracy is the primary metric for evaluating the performance of neural network in predicting jets, so we can rely on the "Adamax" optimization algorithm when building a neural network with two layers, as it achieved the highest accuracy among the listed algorithms. Figure B.13 as shown in Appendix B reflects the application and accuracy of different optimizer functions.

3.2.3 Change the Number of Epochs

For the importance of determining the number of epochs for a neural network on performance and convergence, I changed the number of epochs with different values in order to find out the value with the most accurate result. I used the same variable values with different number of epochs: Number of hidden layers: two, activation function: Softmax, Optimizer function = Adamax. So, as shown in the table below, I have tried different values for the number of epochs and wrote the accuracy of each of them in Table A.11 in Appendix A.

The results indicate that the neural network's accuracy steadily improves with more training epochs up to a certain point (around 300 epochs in this case). Beyond that point, the accuracy stabilizes, and additional training does not provide significant benefits. It's important to strike a balance between training for a sufficient number of epochs to learn from the data and avoiding overfitting, where the model becomes too specialized in the training data and loses generalization ability. The choice of the number of epochs should be made based on a trade-off between training time and model performance on unseen data.

We can conclude that the optimal number of epochs to use in our project is 300.

3.3 Three Layer

A three-layer neural network has been established, and it was found that the most favorable outcomes were obtained by employing specific parameter settings. This included utilizing the Adamax optimization algorithm and conducting a total of 200 training epochs. These precise configurations are visually depicted in the code that accompanies this description.

```
Y=np.array(Y)
k=np.array(k)

# Define EarlyStopping callback
early_stopping = EarlyStopping(monitor='val_loss', patience=10, verbose=1)

# Build the model
model = tf.keras.models.Sequential([ layers.Flatten(input_shape=(1,2453)),
   tf.keras.layers.Dense(1638, activation='selu', kernel_regularizer=tf.keras.regularizers.l2(0.01)),
   tf.keras.layers.Dropout(0.5),
   tf.keras.layers.Dense(920, activation='sigmoid', kernel_regularizer=tf.keras.regularizers.l2(0.01)),
   tf.keras.layers.Dropout(0.3),
   tf.keras.layers.Dense(718, activation='sigmoid', kernel_regularizer=tf.keras.regularizers.l2(0.01))
])

# Compile the model
model.compile(optimizer='Adamax',
            loss='sparse_categorical_crossentropy',
            metrics=['accuracy'])
```

3.3.1 Different Activation Functions

I designed a three-layer neural network and carried out a series of experiments with different activation functions. The accuracy results for each activation function are documented in the tables A.2, A.3, A.4, A.5, A.6, A.7, and A.8 as shown in Appendix A.

Activation functions that exhibited the most promising accuracy results underwent rigorous testing through multiple rounds of experimentation, as outlined in the table A.12 shown in Appendix A. This iterative process was employed to validate and establish the consistency and reliability of their performance.

Similar to the information provided in the preceding table 3.12، there is a cluster of results with closely aligned values. This observation implies that these options can be regarded as the most favorable choices when utilizing a three-layer configuration.

Table A.13 in Appendix A is used to present the results of our statistical analysis using the Anova: Single Factor method. This approach was chosen to evaluate and compare the performance of different options within the two-layer configuration shown in the previous Table A.12. It enables us to see if there are statistically significant differences between these options with respect to our selected variables, providing valuable insights into which options are most appropriate for this specific configuration.

Based on these results:

- The F-statistic (10.40933) is substantially greater than the critical F-value (1.766577).
- The very small p-value (2.58E-13) suggests strong evidence against the null hypothesis.

Consequently, you would conclude that there are statistically significant differences between the means of the groups. In simpler terms, you would reject the null hypothesis.

Therefore, in this analysis, you would conclude that there are significant differences between the groups in terms of their means.

Depending on the value of the mean, in my project the best activations functions when build neural network of three layers that can be used is:

- Softmax, Softplus, Softmax
- Sigmoid, Sigmoid, Sigmoid
- Softplus, Tanh, Sigmoid
- ReLU, Sigmoid, Sigmoid
- ELU, Tanh, Sigmoid

In Table A.14 in Appendix A, I provide the measured time (in seconds) required to execute the training and evaluation processes for each of the best-performing models in terms of accuracy. The timing measurements were obtained using functions like time.time() in Python to precisely measure the duration of each process.

According to the table, the evaluation time ranges between 1.303 and 1.749 seconds, while the training time ranges between 354.437 and 573.5 seconds.

3.3.2 Different Optimizer

In our experimentation process, we conducted a series of tests to evaluate different optimizer functions to identify the one that yielded the highest accuracy. Building on the insights gained from previous experiments, we chose to implement the Softmax activation function. We maintained consistency in the variable settings, with one

modification: we set the number of hidden layers to three. The training duration was fixed at 300 epochs. Table A.15 in Appendix A presents the outcomes of our experiment, displaying the performance of various optimizer functions alongside their respective accuracy values.

According to the table provided, the optimizer function that delivered the highest accuracy. This analysis explores the impact of various optimization algorithms (optimizers) on the accuracy of a neural network. Eight different optimizers were tested: Adamax, Adam, Nadam, Ftrl, Adagrad, RMSprop, Adadelta, and SGD.

Key Findings:

1. Adamax: Adamax optimizer achieved the highest accuracy at 0.7425, outperforming other optimizers in this study.
2. Adam: Adam optimizer achieved an accuracy of 0.7145, demonstrating solid performance.
3. Nadam: Nadam optimizer resulted in an accuracy of 0.6995, performing slightly below Adam and Adamax.
4. Ftrl: Ftrl optimizer achieved an accuracy of 0.7310, showing competitive performance.
5. Adagrad: Adagrad optimizer achieved an accuracy of 0.7345, demonstrating strong performance.
6. RMSprop: RMSprop optimizer achieved an accuracy of 0.7240, indicating good performance.
7. Adadelta: Adadelta optimizer had the lowest accuracy of 0.6015 among the tested optimizers.
8. SGD: Stochastic Gradient Descent (SGD) achieved an accuracy of 0.7050, showing moderate performance.

In conclusion, the choice of optimizer significantly impacts neural network accuracy. Adamax and Adagrad stand out as the top-performing optimizers in terms of accuracy, while Adadelta performed less effectively in this particular context.

3.3.3 Change the Number of Epochs

In line with my earlier discussion about the critical role of determining the optimal number of epochs for enhancing neural network performance and achieving convergence, I conducted experiments by varying the number of epochs to identify the most accurate value.

I maintained consistent parameter values, including a neural network architecture with three hidden layers, a specific activation function configuration (sigmoid for both the output and hidden layers, and sigmoid or silo for the input layer), and utilized the Adamax optimizer. Consequently, I recorded the accuracy results for different epoch values and documented them in Table A.16 in Appendix A.

Figure B.16 in Appendix B presents the accuracy of a neural network model trained at different numbers of epochs:

The model quickly learned patterns and achieved a good accuracy of 0.7510 at 50 epochs.

- Accuracy remained stable around 0.7425 between 150 and 200 epochs.
- Beyond 200 epochs, accuracy started to decline, possibly indicating overfitting or convergence to a suboptimal solution.
- Training for 400 epochs did not significantly improve accuracy.
- After 700 epochs, accuracy dropped to 0.7240, showing signs of overfitting.
- Even with extended training, accuracy improved only slightly, indicating limited gains.

In summary, the neural network reached a reasonable accuracy quickly, but further training didn't consistently improve performance. Monitoring and early stopping techniques are important to prevent overfitting and find the right balance between training time and model accuracy. The choice of the optimal number of epochs depends on the specific dataset and problem.

Chapter Four

Conclusion and Future Work

This chapter concludes the research by emphasizing the focus of the study, the main findings, main contributions, its impact, and potential future work in particle classification.

4.1 Conclusion

The study aimed to create and refine a customized Artificial Neural Network (ANN) to be used for accurate particle classification. A number of experiments and adjustments of different network settings were performed and provided in detail to improve the accuracy of particle identification. The study's main contributions are listed below:

- ANN Architecture: Artificial Neural Networks (ANN) was used to build a system for jet classification. The system built aimed to categorize jets into five distinct groups: light quarks (q), gluons (g), W and Z bosons, and top quarks.
- Parameter Optimization: The experiments conducted in this study highlighted the importance of selecting and fine-tuning parameters. It also highlighted the steps towards achieving top-notch accuracy, which include the number of layers, activation functions, optimizer functions, and training epochs.
- Insights for Researchers: Our research offers important insights for researchers experimenting in artificial intelligence in diverse fields. It stresses the importance of customized neural network architectures and careful parameter optimization to reach outstanding classification performance.

4.2 Future Work

In my future research I intend to:

- Enhance the robustness of particle classification models through incorporating new data sources.
- Explore hybrid models, such as Convolutional Neural Networks (CNNs) in an attempt to reach more accurate particle classification.
- Explore pre-trained models in applying transfer learning techniques in particle classification.

To sum up, the current study shed light on the importance of designing specific neural network structures and optimizing parameters in achieving better accuracy particle classification. Exploring the future opportunities mentioned above can bring unveil the hidden potentials of particle identification, reaching to deeper understanding of the various elements of our universe.

References

[1] Adams, C. R. (1928). On the irregular cases of the linear ordinary difference equation. Transactions of the American Mathematical Society, 30(3), 507-541.

[2] Alloghani, M., Al-Jumeily, D., Mustafina, J., Hussain, A., & Aljaaf, A. J. (2020). A systematic review on supervised and unsupervised machine learning algorithms for data science. Supervised and unsupervised learning for data science, 3-21.ISO 690.

[3] Agarap, A. F. (2018). Deep learning using rectified linear units (relu). arXiv preprint arXiv:1803.08375.

[4] Bianchini, M., Dimitri, G. M., Maggini, M., & Scarselli, F. (2018). Deep neural networks for structured data. Computational intelligence for pattern recognition, 29-51.

[5] Bordes, A., Bottou, L., & Gallinari, P. (2009). SGD-QN: Careful quasi-Newton stochastic gradient descent. Journal of Machine Learning Research, 10, 1737-1754.

[6] Chen, M., Challita, U., Saad, W., Yin, C., & Debbah, M. (2019). Artificial neural networks-based machine learning for wireless networks: A tutorial. IEEE Communications Surveys & Tutorials, 21(4), 3039-3071.

[7] Chio, C., & Freeman, D. (2018). Machine learning and security: Protecting systems with data and algorithms. " O'Reilly Media, Inc.".

[8] Choi, D., Shallue, C. J., Nado, Z., Lee, J., Maddison, C. J., & Dahl, G. E. (2019). On empirical comparisons of optimizers for deep learning. arXiv preprint arXiv:1910.05446.

[9] Dozat, T. (2016). Incorporating nesterov momentum into adam. 2016.

[10] Dubey, S. R., Singh, S. K., & Chaudhuri, B. B. (2022). Activation functions in deep learning: A comprehensive survey and benchmark. Neurocomputing.

[11] Elsheikh, A. H., Sharshir, S. W., Abd Elaziz, M., Kabeel, A. E., Guilan, W., & Haiou, Z. (2019). Modeling of solar energy systems using artificial neural network: A comprehensive review. Solar Energy, 180, 622-639.

[12] Ertuğrul, Ö. F. (2018). A novel type of activation function in artificial neural networks: Trained activation function. Neural Networks, 99, 148-157.

[13] Evans, L. R. (Ed.). (2009). The Large Hadron Collider: A marvel of technology. epfl Press.

[14] Fraser, K., & Schwartz, M. D. (2018). Jet charge and machine learning. Journal of High Energy Physics, 2018.

[15] Feng, J., & Lu, S. (2019, June). Performance analysis of various activation functions in artificial neural networks. In Journal of physics: conference series (Vol. 1237, No. 2, p. 022030). IOP Publishing.

[16] Flair, D. (2021). Advantages and disadvantages of machine learning language

[17] Gulli, A., & Pal, S. (2017). Deep learning with Keras. Packt Publishing Ltd.

[18] Ghotra, M. S., & Dua, R. (2017). Neural Network Programming with TensorFlow: Unleash the power of TensorFlow to train efficient neural networks. Packt Publishing Ltd.

[19] Harmon, M. E., & Harmon, S. S. (1996). Reinforcement learning: A tutorial. WL/AAFC, WPAFB Ohio, 45433, 237-285. ISO 690.

[20] Haykin, S. (1998). Neural networks: A comprehensive foundation. Prentice Hall PTR.

[21] Hinton, G., Srivastava, N., & Swersky, K. (2012). Neural networks for machine learning lecture 6a overview of mini-batch gradient descent. Cited on, 14(8), 2.

[22] J. Duarte et al., "Fast inference of deep neural networks in FPGAs for particle physics", JINST 13 (2018), no. 07, P07027, doi:10.1088/1748-0221/13/07/P07027, arXiv:1804.06913.

[23] Hooker, S., Erhan, D., Kindermans, P. J., & Kim, B. (2018). Evaluating feature importance estimates. arXiv preprint arXiv:1806.10758, 2

[24] Khanzode, K. C. A., & Sarode, R. D. (2020). Advantages and disadvantages of artificial intelligence and machine learning: A literature review. International Journal of Library & Information Science (IJLIS), 9(1), 3.

[25] KLAUS. RABBERTZ. (2018). JET PHYSICS AT THE LHC: The Strong Force Beyond the Tev Scale. SPRINGER.

[26] Kotsiantis, S. B. (2007). Supervised Machine Learning: A Review of Classification Techniques. Informatica 31 (2007). Pp. 249 – 268. Retrieved from IJS website: http://wen.ijs.si/ojs2.4.3/index.php/informatica/article/download/148/140.

[27] Kumar, S., Mishra, R. K., Mitra, A., Biswas, S., De, S., & Karmakar, R. (2020, September). A Relative Comparison of Training Algorithms in Artificial Neural Network. In 2020 IEEE 1st International Conference for Convergence in Engineering (ICCE) (pp. 315-319). IEEE.

[28] Lefevre, C. (2008). LHC: The guide (No. CERN-Brochure-2008-001-Eng).

[29] Li, H., Li, J., Guan, X., Liang, B., Lai, Y., & Luo, X. (2019, December). Research on overfitting of deep learning. In 2019 15th international conference on computational intelligence and security (CIS) (pp. 78-81). IEEE.

[30] Logistic Regression pp. 223 – 237. Available at: https://www.stat.cmu.edu/~cshalizi/uADA/12/lectures/ch12.pdf

[31] M arzani, S., Soyez, G., & Spannowsky, M. Looking inside jets: an introduction to jet substructure and boosted-object phenomenology [Lect. Notes Phys. 958, pp.(2019)]. arXiv preprint arXiv:1901.10342.

[32] Machine Learning ConcepEnvironment [ts with Python and the Jupyter Notebook.

[33] Malware Detection Of Portable Executable Using Machine Learning And Convolutional Neural Network. (2021). Strad Research, 8(5). doi: 10.37896/sr8.5/054

[34] McMahan, H. B., Holt, G., Sculley, D., Young, M., Ebner, D., Grady, J.,... & Kubica, J. (2013, August). Ad click prediction: a view from the trenches. In Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining (pp. 1222-1230).

[35] Mohammed, C. M., & Askar, S. (2021). Machine learning for IoT healthcare applications: A review. International Journal of Science and Business, 5(3), 42-51.

[36] Moolayil, J., Moolayil, J., & John, S. (2019). Learn Keras for deep neural networks (pp. 33-35). Berkeley, CA, USA: Apress. [10]

[37] Moreira, L., Vettor, R., & Guedes Soares, C. (2021). Neural network approach for predicting ship speed and fuel consumption. Journal of Marine Science and Engineering, 9(2), 119.

[38] Moreno, E. A., Cerri, O., Duarte, J. M., Newman, H. B., Nguyen, T. Q., Periwal, A.,... & Vlimant, J. R. (2020). JEDI-net: a jet identification algorithm based on interaction networks. The European Physical Journal C, 80, 1-15.

[39] Neocleous C. & Schizas C. (2002). Artificial Neural Network Learning: A Comparative Review. In: Vlahavas I.P., Spyropoulos C.D. (eds) Methods and Applications of Artificial Intelligence. Hellenic Conference on Artificial IntelligenceSETN 2002. Lecture Notes in Computer Science, Volume 2308. Springer, Berlin, Heidelberg, doi: 10.1007/3-540-46014-4_27 pp. 300-313. Available at: https://link.springer.com/chapter/10.1007/3-540-46014- 4_27

[40] Nguyen, A., Pham, K., Ngo, D., Ngo, T., & Pham, L. (2021, August). An analysis of state-of-the-art activation functions for supervised deep neural network. In 2021 International Conference on System Science and Engineering (ICSSE) (pp. 215-220). IEEE.

[41] Nguyen, M. (2013). b-jet Identification in PbPb Collisions with CMS. Nuclear Physics A, 904-905, 705c-708c. doi: 10.1016/j.nuclphysa.2013.

[42] Nilsson, N.J. (1965). Learning machines. New York: McGraw-Hill.Published in: Journal of IEEE Transactions on Information Theory Volume 12 Issue 3, 1966. doi: 10.1109/TIT.1966.1053912 pp. 407 – 407. Available at ACM digital library website: http://dl.acm.org/citation.cfm?id=2267404`1

[43] Osisanwo, F. Y., Akinsola, J. E. T., Awodele, O., Hinmikaiye, J. O., Olakanmi, O., & Akinjobi, J. (2017). Supervised machine learning algorithms: Classification and comparison. International Journal of Computer Trends and Technology (IJCTT), 48(3), 128-138.

[44] Sakketou, F., & Ampazis, N. (2019). On the invariance of the selu activation function on algorithm and hyperparameter selection in neural network recommenders. In Artificial Intelligence Applications and Innovations: 15th IFIP WG 12.5 International Conference, AIAI 2019, Hersonissos, Crete, Greece, May 24–26, 2019, Proceedings 15 (pp. 673-685). Springer International Publishing.

[45] Shapiro, A. F., & Jain, L. C. (Eds.). (2003). Intelligent and other computational techniques in insurance: theory and applications (Vol. 6). World Scientific.

[46] Shi, N., & Li, D. (2021, April). Rmsprop converges with proper hyperparameter. In International conference on learning representation.

[47] Silaparasetty, N. (2020). Machine learning concepts with python and the jupyter notebook environment: Using tensorflow 2.0. Apress.

[48] Singh, A., Thakur, N., & Sharma, A. (2016, March). A review of supervised machine learning algorithms. In 2016 3rd international conference on computing for sustainable global development (INDIACom) (pp. 1310-1315). Ieee. ISO 690.

[49] Smys, S., Chen, J. I. Z., & Shakya, S. (2020). Survey on neural network architectures with deep learning. Journal of Soft Computing Paradigm (JSCP), 2(03), 186-194.

[50] Sun, X., Xie, M., Zhou, F., Wu, X., Fu, J., & Liu, J. (2023). Hierarchical evolutionary construction of neural network models for an Atkinson cycle engine with double injection strategy based on the PSO-Nadam algorithm. Fuel, 333, 126531

[51] Darwis,Ong(2012). Supervised Learning Article ·DOI:10.1007/978-1-4419-1428-6_451

[52] Suzuki, K. (Ed.). (2011). Artificial neural networks: methodological advances and biomedical applications. BoD–Books on Demand.

[53] Usama, M., Qadir, J., Raza, A., Arif, H., Yau, K. L. A., Elkhatib, Y.,... & Al-Fuqaha, A. (2019). Unsupervised machine learning for networking: Techniques, applications and research challenges. IEEE access, 7, 65579-65615.

[54] Uzair, M., & Jamil, N. (2020, November). Effects of hidden layers on the efficiency of neural networks. In 2020 IEEE 23rd international multitopic conference (INMIC) (pp. 1-6). IEEE.

[55] Xu, B., Wang, N., Chen, T., & Li, M. (2015). Empirical evaluation of rectified activations in convolutional network. arXiv preprint arXiv:1505.00853.

[56] Xu, X., Cao, D., Zhou, Y., & Gao, J. (2020). Application of neural network algorithm in fault diagnosis of mechanical intelligence. Mechanical Systems and Signal Processing, 141, 106625.

[57] Ying, X. (2019, February). An overview of overfitting and its solutions. In Journal of physics: Conference series (Vol. 1168, p. 022022). IOP Publishing.

[58] Zhang, J. (2019). Gradient descent based optimization algorithms for deep learning models training. arXiv preprint arXiv:1903.03614.

[59] Zou, F., Shen, L., Jie, Z., Zhang, W., & Liu, W. (2019). A sufficient condition for convergences of adam and rmsprop. In Proceedings of the IEEE/CVF Conference on computer vision and pattern recognition (pp. 11127-11135).

Appendices

Appendix A

Tables

Table A.1

Accuracy of Different Activation Functions of Two-Layer Neural Network

Input Layer	Hidden Layers	Accuracy
Selu	ReLu	0.2080
	ELU	0.2955
	Selu	0.2080
Selu	Sigmoid	0.7230
	Tanh	0.2080
	Softmax	0.6575
	Softplus	0.2320
ELU	ReLu	0.2080
	ELU	0.2080
	Selu	0.2080
	Sigmoid	0.7235
	Tanh	0.2080
ELU	Softmax	0.7255
	Softplus	0.2295
ReLu	ReLu	0.2080
	ELU	0.2080
	Selu	0.2130
	sigmoid	0.6970
	Tanh	0.2080
ReLu	softmax	0.7060
	Softplus	
Sigmoid	ReLu	0.3495
	ELU	0.3670

Input Layer	Hidden Layers	Accuracy
	Selu	0.3260
Sigmoid	sigmoid	0.7020
	Tanh	0.4115
	softmax	0.6955
	Softplus	0.6915
Tanh	ReLu	0.1945
	ELU	0.2035
	Selu	0.2480
	sigmoid	0.6055
	Tanh	0.3695
	softmax	0.6795
	Softplus	0.6685
Softmax	ReLu	0.2075
	ELU	0.2075
	Selu	0.0000e
	sigmoid	0.2080
	Tanh	0.2070
	softmax	0.2080
	Softplus	0.2080
Softplus	ReLu	0.2080
	ELU	0.2080
	Selu	0.2080
Softplus	sigmoid	0.7060
	Tanh	0.2080
	softmax	0.6940
	Softplus	0.2110

Table A.2

Accuracy of Selu Activation Function of Three-Layer of Neural Network

Input layer	Hidden layer	Output layer	Accuracy
Selu	ELU	ReLu	0.2160
		ELU	0.2070
		Selu	0.2080
		Sigmoid	0.6960
		Tanh	0.2080
		Softmax	0.6950
		Softplus	0.2515
	ReLu	ReLu	0.2070
		ELU	0.2080
		Selu	0.2070
		Sigmoid	0.6885
		Tanh	0.2080
		Softmax	0.6775
		Softplus	0.2080
	Selu	ReLu	0.2075
		ELU	0.2080
		Selu	0.2345
		Sigmoid	0.6790
		Tanh	0.2080
		Softmax	0.6875
		Softplus	0.2150
	Sigmoid	ReLu	0.2070
		ELU	0.2080
		Selu	0.2075

Input layer	Hidden layer	Output layer	Accuracy
Selu	Sigmoid	Sigmoid	0.7420
		Tanh	0.5595
		Softmax	0.7370
		Softplus	0.2075
	Tanh	ReLu	0.2080
		ELU	0.2070
		Selu	0.2080
		Sigmoid	0.7210
		Tanh	0.1825
		Softmax	0.7335
		Softplus	0.2080
	Softmax	ReLu	0.2080
		ELU	0.3070
		Selu	0.2075
		Sigmoid	0.2080
		Tanh	0.1950
		Softmax	0.2080
		Softplus	0.1825
	Softplus	ReLu	0.2070
		ELU	0.1825
		Selu	0.1850
		Sigmoid	0.6875
		Tanh	0.2080
		Softmax	0.6955
		Softplus	0.2075

Table A.3

Accuracy of ReLU Activation Function of Three-Layer of Neural Network

Input layer	Hidden layer	Output layer	Accuracy
ReLU	ELU	ReLu	0.2475
		ELU	0.2080
		Selu	0.1935
		Sigmoid	0.6860
		Tanh	0.2080
		Softmax	0.6850
		Softplus	0.2080
	ReLu	ReLu	0.2070
		ELU	0.2070
		Selu	0.2080
		Sigmoid	0.6800
		Tanh	0.1825
		Softmax	0.6650
		Softplus	0.2075
	Selu	ReLu	0.1950
		ELU	0.2425
		Selu	0.1950
		Sigmoid	0.6855
		Tanh	0.2080
		Softmax	0.6815
		Softplus	0.1960
	Sigmoid	ReLu	0.1960
		ELU	0.1825
		Selu	0.2075

Input layer	Hidden layer	Output layer	Accuracy
ReLU	Sigmoid	Sigmoid	0.7245
		Tanh	0.5705
		Softmax	0.7235
		Softplus	0.1825
	Tanh	ReLu	0.2070
		ELU	0.1825
		Selu	0.2080
		Sigmoid	0.7220
		Tanh	0.4770
		Softmax	0.7240
		Softplus	0.2075
	Softmax	ReLu	0.2080
		ELU	0.2075
		Selu	0.2080
		Sigmoid	0.2080
		Tanh	0.3725
		Softmax	0.2080
		Softplus	0.1950
	Softplus	ReLu	0.2070
		ELU	0.2090
		Selu	0.2080
		Sigmoid	0.6875
		Tanh	0.2075
		Softmax	0.6520
		Softplus	0.1970

Table A.4

Accuracy of ELU Activation Function of Three-Layer of Neural Network

Input layer	Hidden layer	Output layer	Accuracy
ELU	ELU	ReLu	0.1950
		ELU	0.1950
		Selu	0.2070
		Sigmoid	0.6785
		Tanh	0.2080
		Softmax	0.6785
		Softplus	0.2075
	ReLu	ReLu	0.2360
		ELU	0.2075
		Selu	0.2265
		Sigmoid	0.6865
		Tanh	0.2080
		Softmax	0.6810
		Softplus	0.2030
	Selu	ReLu	0.2070
		ELU	0.2075
		Selu	0.2080
		Sigmoid	0.6790
		Tanh	0.1950
		Softmax	0.6765
		Softplus	0.2200
	Sigmoid	ReLu	0.1825
		ELU	0.2070
		Selu	0.2425

Input layer	Hidden layer	Output layer	Accuracy
ELU	Sigmoid	Sigmoid	0.7305
		Tanh	0.5070
		Softmax	0.7335
		Softplus	0.1950
	Tanh	ReLu	0.1835
		ELU	0.2495
		Selu	0.1825
	Softmax	Sigmoid	0.7285
		Tanh	0.5565
		Softmax	0.7330
		Softplus	0.1825
	Softplus	ReLu	0.2080
		ELU	0.3210
		Selu	0.2075
		Sigmoid	0.2080
		Tanh	0.2070
		Softmax	0.2080
		Softplus	0.2075
		ReLu	0.2160
		ELU	0.1950
		Selu	0.1950
		Sigmoid	0.6850
		Tanh	0.2080
		Softmax	0.6950
		Softplus	0.2080

Table A.5

Accuracy of Sigmoid Activation Function of Three-Layer of Neural Network

Input layer	Hidden layer	Output layer	Accuracy
Sigmoid	ELU	ReLu	0.1950
		ELU	0.1975
		Selu	0.2080
		Sigmoid	0.6775
		Tanh	0.2080
		Softmax	0.6840
		Softplus	0.2075
	ReLu	ReLu	2080
		ELU	0.1950
		Selu	0.2070
		Sigmoid	0.6785
		Tanh	0.1825
		Softmax	0.6810
		Softplus	. 0.2110
	Selu	ReLu	0.2070
		ELU	0.1950
		Selu	0.2070
		Sigmoid	0.6810
		Tanh	0.2080
		Softmax	0.6785
		Softplus	0.2075
	Sigmoid	ReLu	0.2070
		ELU	0.2070
		Selu	0.2320

Input layer	Hidden layer	Output layer	Accuracy
Sigmoid	Sigmoid	Sigmoid	0.7100
		Tanh	0.0075
		Softmax	0.6985
		Softplus	0.2070
	Tanh	ReLu	0.2800
		ELU	0.2070
		Selu	0.2080
		Sigmoid	0.6810
		Tanh	0.2105
		Softmax	0.6910
		Softplus	0.1825
	Softmax	ReLu	0.3785
		ELU	0.3740
		Selu	0.6240
		Sigmoid	0.2080
		Tanh	0.4565
		Softmax	0.2080
		Softplus	0.2080
	Softplus	ReLu	0. 0.1950
		ELU	0.1950
		Selu	0.1950
		Sigmoid	0.6890
		Tanh	0.2190
		Softmax	0.6805
		Softplus	0.1825

Table A.6

Accuracy of Tanh Activation Function of Three-Layer of Neural Network

Input Layer	Hidden layer	Output layer	Accuracy
Tanh	ELU	ReLu	0.1950
		ELU	0.2080
		Selu	0.1825
		Sigmoid	0.6660
		Tanh	0.2090
		Softmax	0.6700
		Softplus	0.3000
	ReLu	ReLu	0.2075
		ELU	0.2075
		Selu	0.2070
		Sigmoid	0.6720
		Tanh	0.2080
		Softmax	0.6830
		Softplus	0.1825
	Selu	ReLu	0.1825
		ELU	0.1825
		Selu	0.2075
		Sigmoid	0.6670
		Tanh	0.2080
		Softmax	0.6845
		Softplus	0.2420
	Sigmoid	ReLu	0.2415
		ELU	0.2085
		Selu	0.1950

Input Layer	Hidden layer	Output layer	Accuracy
Tanh	Sigmoid	Sigmoid	0.6925
		Tanh	5.0000e -04
		Softmax	0.6955
		Softplus	0.2075
	Tanh	ReLu	0.2070
		ELU	0.2075
		Selu	0.2220
	Softmax	Sigmoid	0.6860
		Tanh	0.2440
		Softmax	0.6760
		Softplus	0.2070
	Softplus	ReLu	0.6135
		ELU	0.5535
		Selu	0.5870
		Sigmoid	0.2630
		Tanh	0.5785
		Softmax	0.2080
		Softplus	0.2075
		ReLu	0.2230
		ELU	0.2080
		Selu	0.2115
		Sigmoid	0.6820
		Tanh	0.2080
		Softmax	0.6705
		Softplus	0.2070

Table A.7

Accuracy of Softmax Activation Function of Three-Layer of Neural Network

Input layer	Hidden layer	Output layer	Accuracy
Softmax	ELU	ReLu	0.6230
		ELU	0.5920
		Selu	0.5970
		Sigmoid	0.6130
		Tanh	0.5730
		Softmax	0.6065
		Softplus	0.2070
	ReLu	ReLu	0.6200
		ELU	0.0015
		Selu	0.1225
		Sigmoid	0.2080
		Tanh	0.5695
		Softmax	0.4355
		Softplus	0.1825
	Selu	ReLu	0.6290
		ELU	0.5425
		Selu	0.6510
		Sigmoid	0.4400
		Tanh	0.5415
Softmax	Selu	Softmax	0.7120
		Softplus	0.2080
		eLuR	0.2075
		ELU	0.1825
		Selu	0.2080

Input layer	Hidden layer	Output layer	Accuracy
		Sigmoid	0.6650
		Tanh	0.0000e
		Softmax	0.6890
		Softplus	0.2070
	Tanh	ReLu	0.5750
		ELU	0.4410
		Selu	0.6135
		Sigmoid	0.6720
		Tanh	0.6550
		Softmax	0.6905
		Softplus	0.1950
	Softmax	ReLu	0.2080
		ELU	
		Selu	0.2880
		Sigmoid	0.2080
		Tanh	
		Softmax	0.2080
		Softplus	0.2075
	Softplus	ReLu	0.2005
		ELU	0.2335
		Selu	0.0095
		Sigmoid	0.6700
		Tanh	
Softmax	Softplus	Softmax	0.7030
		Softplus	0.2075

Table A.8

Accuracy of Softplus Activation Function of Three-Layer of Neural Network

Input layer	Hidden layer	Output layer	Accuracy
Softplus	ELU	ReLu	0.2075
		ELU	0.2080
		Selu	0.1820
		Sigmoid	0.6795
		Tanh	0.2080
		Softmax	0.6775
		Softplus	0.1825
	ReLu	ReLu	0.2070
		ELU	0.2205
		Selu	0.2440
		Sigmoid	0.6810
		Tanh	0.2080
		Softmax	0.6715
		Softplus	0.2145
	Selu	ReLu	0.1825
		ELU	0.1825
		Selu	0.2070
		Sigmoid	0.6925
		Tanh	0.2080
		Softmax	0.6795
		Softplus	0.2080
	Sigmoid	ReLu	0.2070
		ELU	0.2070
		Selu	0.1950

Input layer	Hidden layer	Output layer	Accuracy
Softplus	Sigmoid	Sigmoid	0.7360
		Tanh	0.5805
		Softmax	0.7255
		Softplus	0.1950
	Tanh	ReLu	0.2070
		ELU	0.2070
		Selu	0.1825
		Sigmoid	0.7175
		Tanh	0.1950
		Softmax	0.7145
		Softplus	0.1825
	Softmax	ReLu	0.2080
		ELU	0.1950
		Selu	0.2780
		Sigmoid	0.2080
		Tanh	0.2070
		Softmax	0.2080
		Softplus	0.2080
	Softplus	ReLu	0.2085
		ELU	0.2275
		Selu	0.2570
		Sigmoid	0.6735
		Tanh	0.2080
		Softmax	0.6740
		Softplus	0.2410

Table A.9

Anova: Single Factor of Two-Layer Neural Network

SUMMARY						
Groups	*Count*	*Sum*	*Average*	*Variance*		
ELU, Softmax	6	4.425	0.7375	4.64E-05		
ELU, Sigmoid	6	4.428	0.738	5.85E-05		
Selu,Sigmoid	6	4.4395	0.739917	9.33E-05		
ReLu,softmax	6	4.401	0.7335	0.000206		
Softplus, sigmoid	6	4.366	0.727667	0.000153		
Sigmoid, sigmoid	6	4.458	0.743	0.000409		
ANOVA						
Source of Variation	*SS*	*df*	*MS*	*F*	*P-value*	*F crit*
Between Groups	0.000865	5	0.000173	1.074472	0.394177	2.533555
Within Groups	0.00483	30	0.000161			
Total	0.005694	35				

Table A.10

Accuracy of Different Optimizer Functions of Two-Layers Neural Network

Optimizer	Accuracy
Adamax	0.7415
Adam	0.7135
Nadam	0.7105
Ftrl	0.7020
Adagrad	0. 0.6275
RMSprop	0.6805
Adadelta	0.4980
AdaDelta	0.4985
SGD	0.5470

Table A.11

Accuracy of different Number of Epochs of Two-Layer Neural Network

Number of epochs	Accuracy
50	0.7230
150	0.7420
200	0.7505
300	0.7580
400	0.7575
500	0.7532
700	0.7510
1000	0.7540
1500	0.7210
3000	0.7040

Table A.12

Accuracy of Different Activation Functions of Three-Layer

Input Layer	Hidden Layers	Output Layer	Exp1	Exp2	Exp3	Exp4	Exp5
Selu	Sigmoid	Sigmoid	0.7425	0.7315	0.7320	0.7310	0.7430
Selu	Sigmoid	Softmax	0.7370	0.7360	0.7340	0.7400	0.7290
Softplus	Sigmoid	Sigmoid	0.7360	0.7230	0.7205	0.7270	0.7160
Selu	Tanh	Softmax	0.7335	0.7225	0.7345	0.7385	0.7335
ELU	Sigmoid	Softmax	0.7335	0.7275	0.7260	0.7250	0.7355
ELU	Tanh	Softmax	0.7330	0.7360	0.7240	0.7340	0.7280
ELU	Sigmoid	Sigmoid	0.7305	0.7115	0.7275	0.7285	0.7310
ELU	Tanh	Sigmoid	0.7285	0.7245	0.7340	0.7300	0.7315
ReLU	Sigmoid	Sigmoid	0.7245	0.7170	0.7235	0.7160	0.7170
ReLU	Tanh	Softmax	0.7240	0.7235	0.7225	0.7240	0.7205
ReLU	Sigmoid	Softmax	0.7235	0.7170	0.7235	0.7210	0.7200
ReLU	Tanh	Sigmoid	0.7220	0.7130	0.7220	0.7190	0.7230
Selu	Tanh	Sigmoid	0.7210	0.7400	0.7305	0.7390	0.7355
Softmax	Selu	Softmax	0.7120	0.6735	0.6980	0.7095	0.7100
Sigmoid	Sigmoid	Sigmoid	0.7100	0.7090	0.7010	0.6940	0.6880
Softmax	Softplus	Softmax	0.7030	0.7125	0.6640	0.7015	0.7155
Softplus	Tanh	Sigmoid	0.7175	0.7205	0.7230	0.7155	0.7140
Softplus	Tanh	Softmax	0.7145	0.7220	0.7175	0.7170	0.7200

Table A.13

Anova: Single Factor of Three-Layer of Neural Network

SUMMARY						
Groups	*Count*	*Sum*	*Average*	*Variance*		
Softmax, Softplus,Softmax	5	3.68	0.736	3.81E-05		
Sigmoid,Sigmoid,Sigmoid	5	3.676	0.7352	1.67E-05		
Softmax,Selu,Softmax	5	3.6225	0.7245	5.73E-05		
Softplus, Tanh, Sigmoid	5	3.6625	0.7325	3.55E-05		
Softplus, Tanh, Softmax	5	3.6475	0.7295	2.21E-05		
ReLU, Sigmoid, Sigmoid	5	3.655	0.731	0.000024		
ReLU, Tanh, Sigmoid	5	3.629	0.7258	6.6E-05		
ReLU, Sigmoid, Softmax	5	3.6485	0.7297	1.26E-05		
ReLU, Tanh, Softmax	5	3.598	0.7196	1.64E-05		
Softplus, Sigmoid, Sigmoid	5	3.6145	0.7229	2.17E-06		
ELU, Sigmoid, Sigmoid	5	3.605	0.721	7.38E-06		
ELU, Sigmoid, Softmax	5	3.599	0.7198	1.67E-05		
ELU, Tanh, Sigmoid	5	3.666	0.7332	6.03E-05		
ELU, Tanh, Softmax	5	3.503	0.7006	0.00026		
Selu, Tanh, Softmax	5	3.502	0.7004	9.03E-05		
Selu, Tanh ,Sigmoid	5	3.4965	0.6993	0.000425		
Selu, Sigmoid, Softmax	5	3.5905	0.7181	1.34E-05		
Selu, Sigmoid, Sigmoid	5	3.591	0.7182	8.32E-06		
ANOVA						
Source of Variation	*SS*	*df*	*MS*	*F*	*P-value*	*F crit*
Between Groups	0.0115247	17	0.000678	10.40933	2.58E-13	1.766577
Within Groups	0.0046891	72	6.51E-05			
Total	0.0162138	89				

Table A.14

Training and evaluation time of Three-layer of Neural Network models

Activation function	Training Time	Evaluation Time	Evaluation Accuracy	Test Accuracy
Softmax, Softplus, Softmax	442.449	1.73	0.736	0.701
Sigmoid, Sigmoid, Sigmoid	474.119	1.5885	0.702	0.692
Softplus, Tanh, Sigmoid	477.745	1.381	0.72049	0.688
ReLU, Sigmoid, Sigmoid	354.437	1.303	0.725	0.679
ELU, Tanh, Sigmoid	573.5	1.749	0.735	0.702

Table A.15

Training and evaluation time of Three-layer of Neural Network models

Optimizer	Accuracy
Adamax	0.7425
Adam	0.7145
Nadam	0.6995
Ftrl	0.7310
Adagrad	0.7345
RMSprop	0.7240
Adadelta	0.6015
SGD	0.7050

Table A.16

Accuracy of Different Number of Epochs of Three-Layers

Number of epochs	Accuracy
50	0.7510
150	0.7425
200	0.7425
300	0.7395
400	0.7395
700	0.7240
1000	0.7245
2000	0.7285
3000	0.7320
10000	0.7142

Appendix B

Figures

Figure B.1

Graph Derivative of Equation for the Sigmoid Activation Function[15]

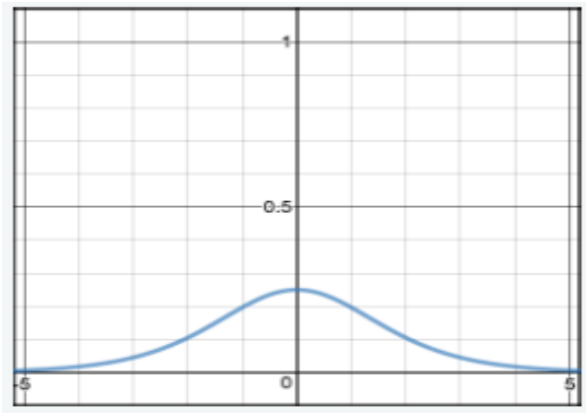


Figure B.2

Graph Equation for the Tanh Activation Function

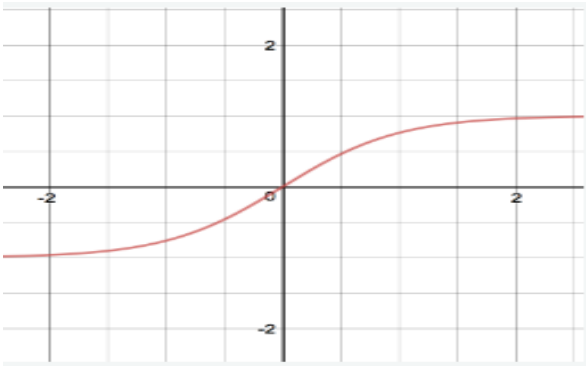


Figure B.3

Graph Derivative of Equation for the Tanh Activation Function.

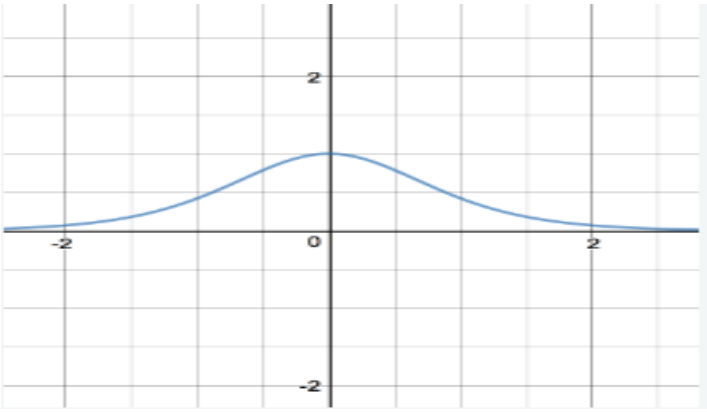


Figure B.4

Graph Equation for the Softmax Activation Function.

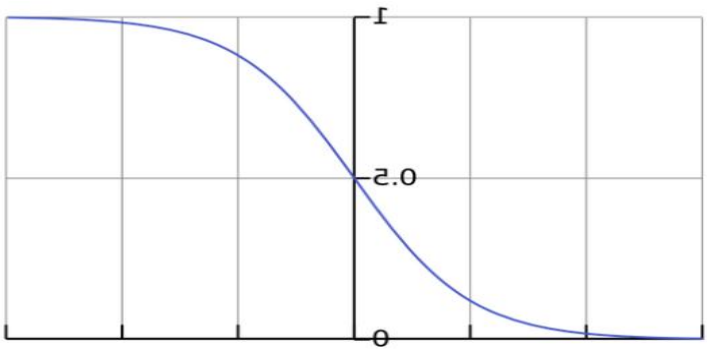


Figure B.5

Graph Equation for the Softplus Activation Function

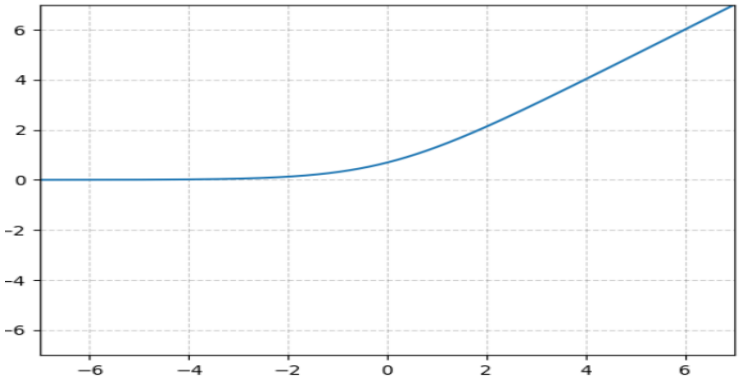


Figure B.6

Distributions of the 16 High-Level Features Used in this Study Ref.

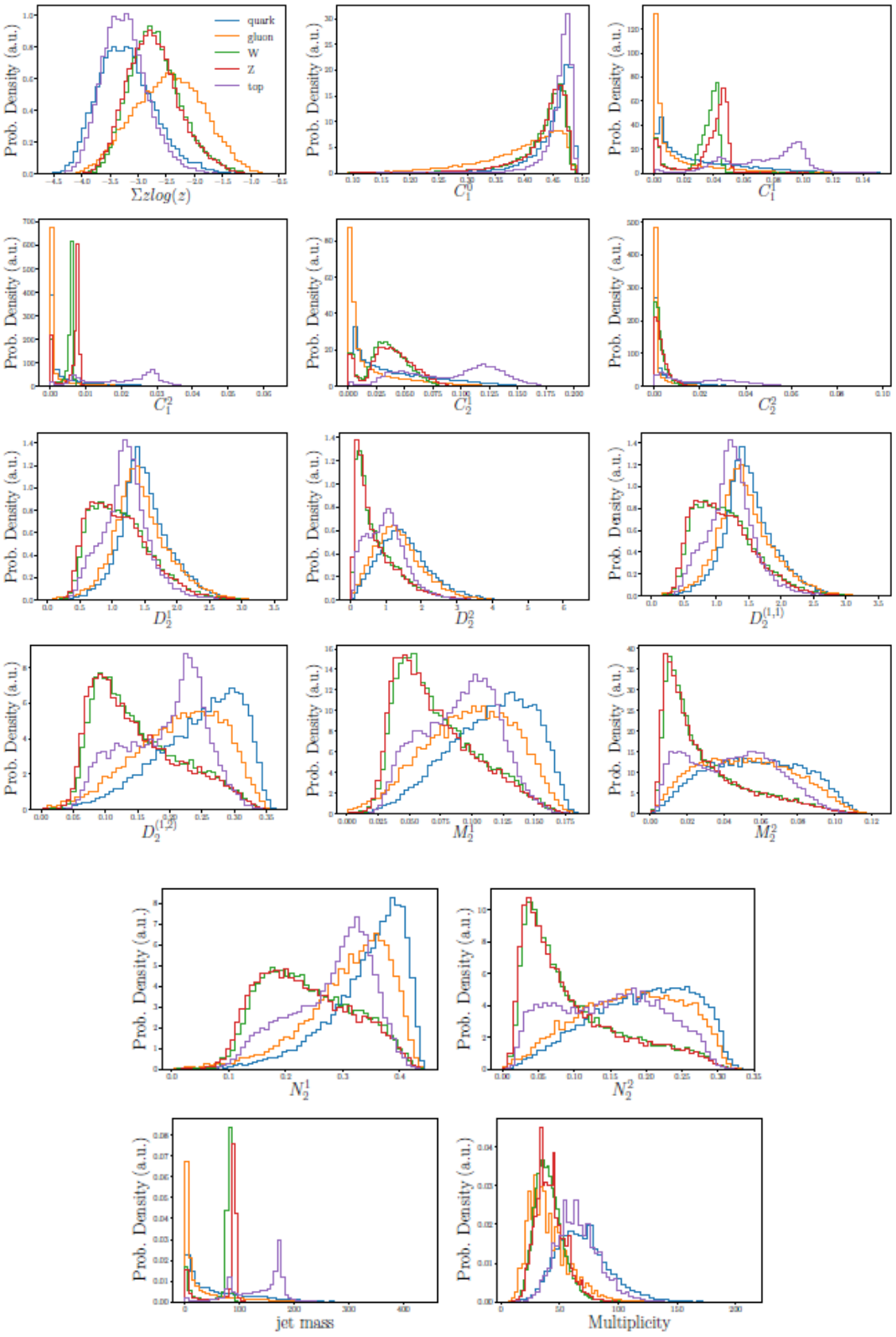


Figure B.7

Flowchart of a Machine Learning Model

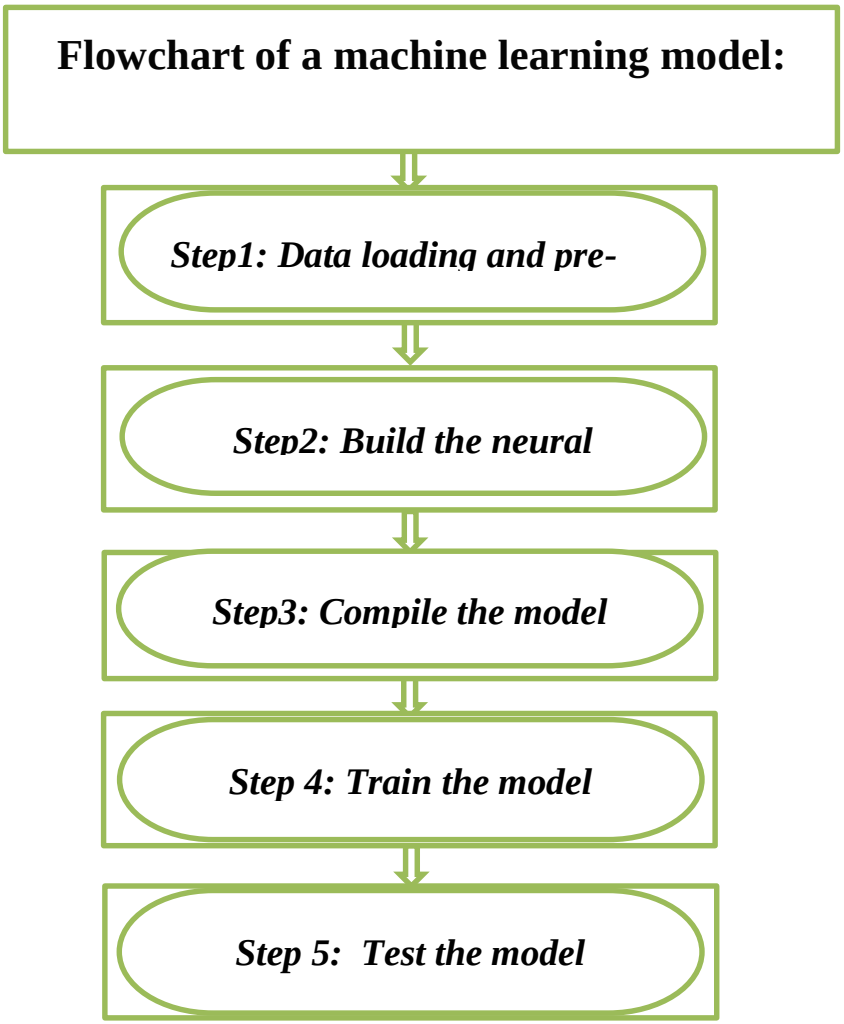


Figure B.8

Chart Accuracy of Different Activation Function of Single-Layer

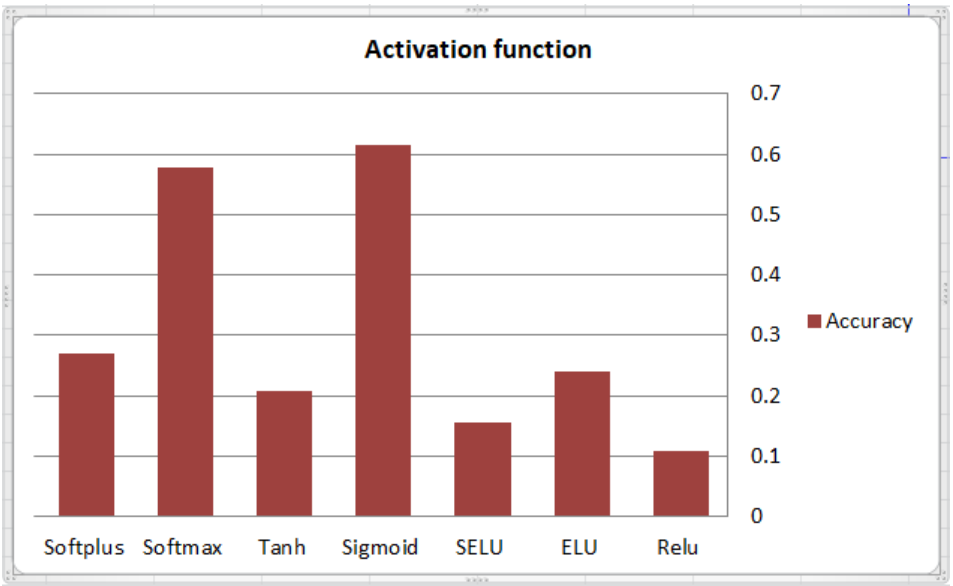


Figure B.9

Chart Accuracy of Different Experiments of Single-Layer Neural Network

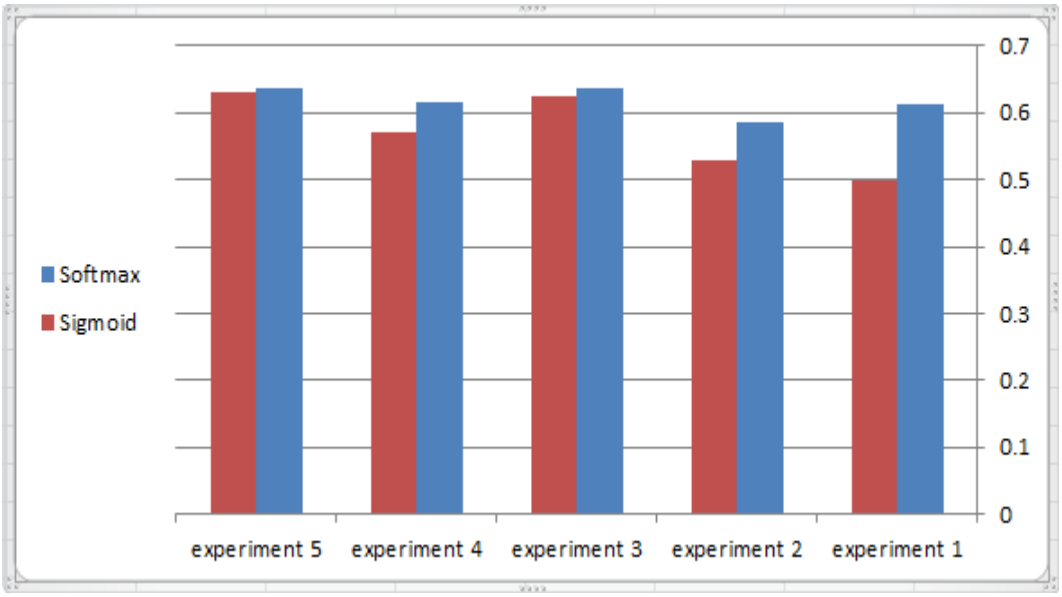


Figure B.10

Chart Accuracy of Different Optimizer Function of Single-Layer

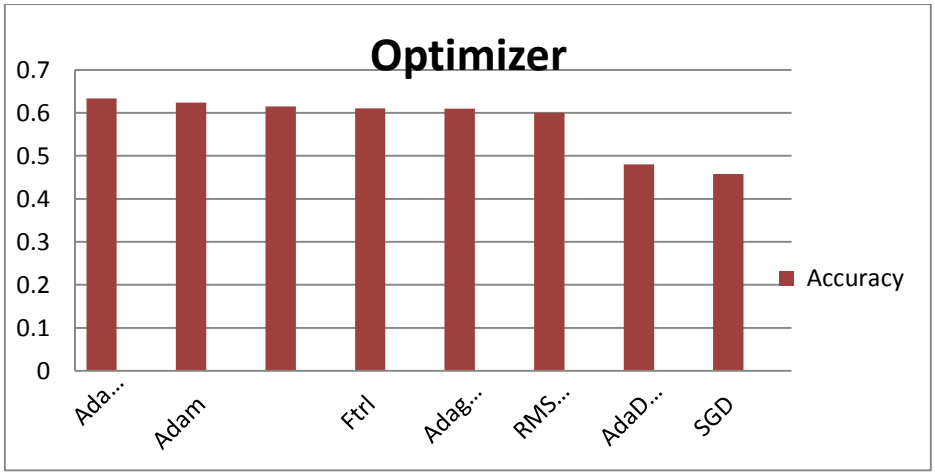


Figure B.11

Chart Varying Epochs of Single-Layer Neural Network

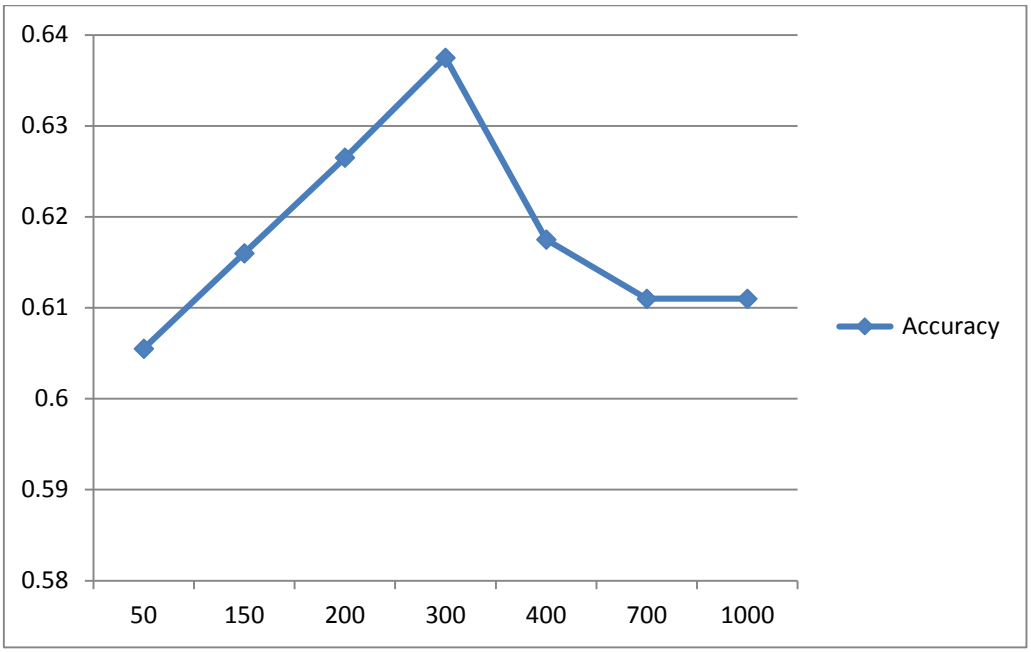


Figure B.12

Chart Average of Accuracy of Different Experiments of Two-Layers

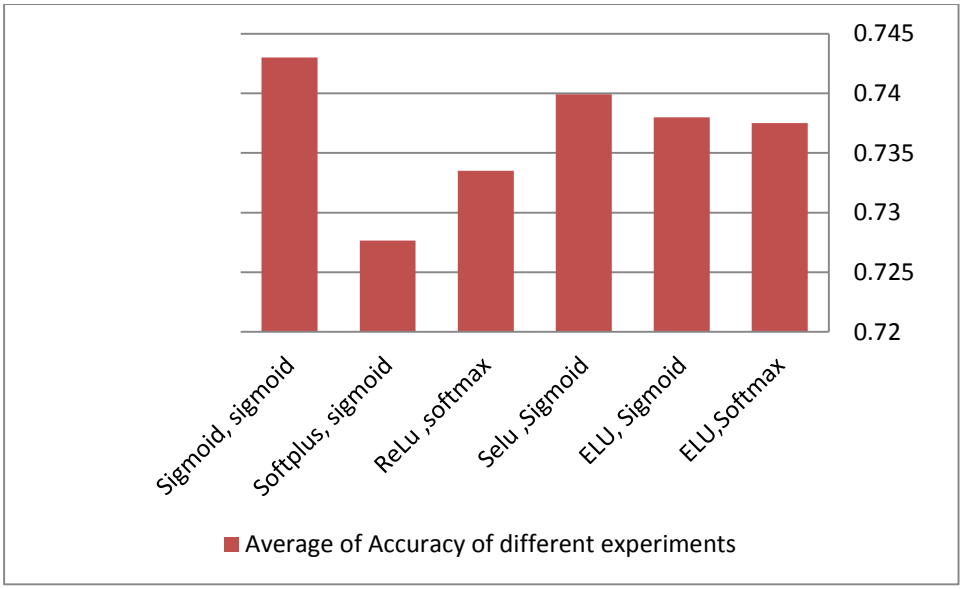


Figure B.13

Chart Accuracy of Different Optimizer Function of Two-Layer

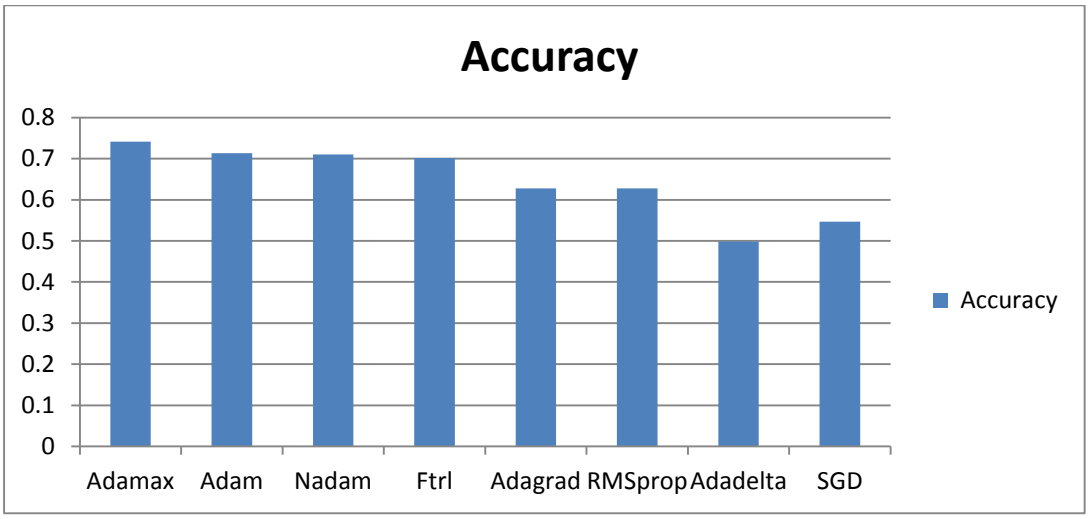


Figure B.14

Chart Accuracy of Different Number of Epochs of Two-Layer

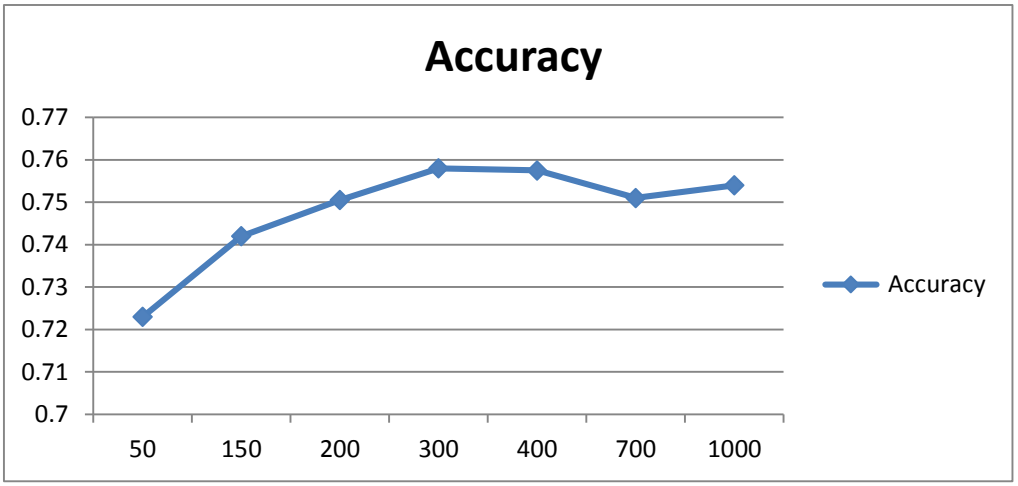


Figure B.15

Chart of Accuracy Comparison of Different Optimizers in Neural Network Training

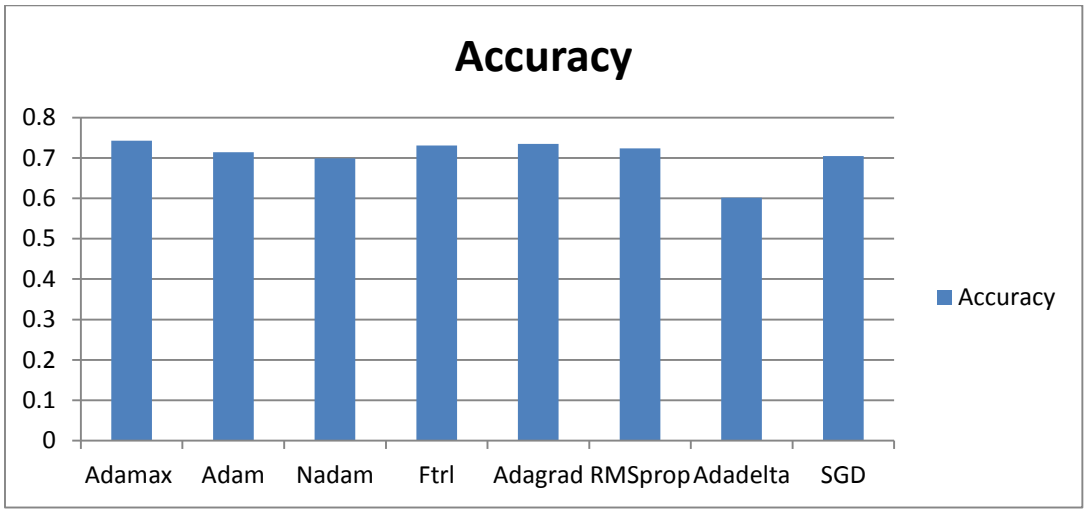
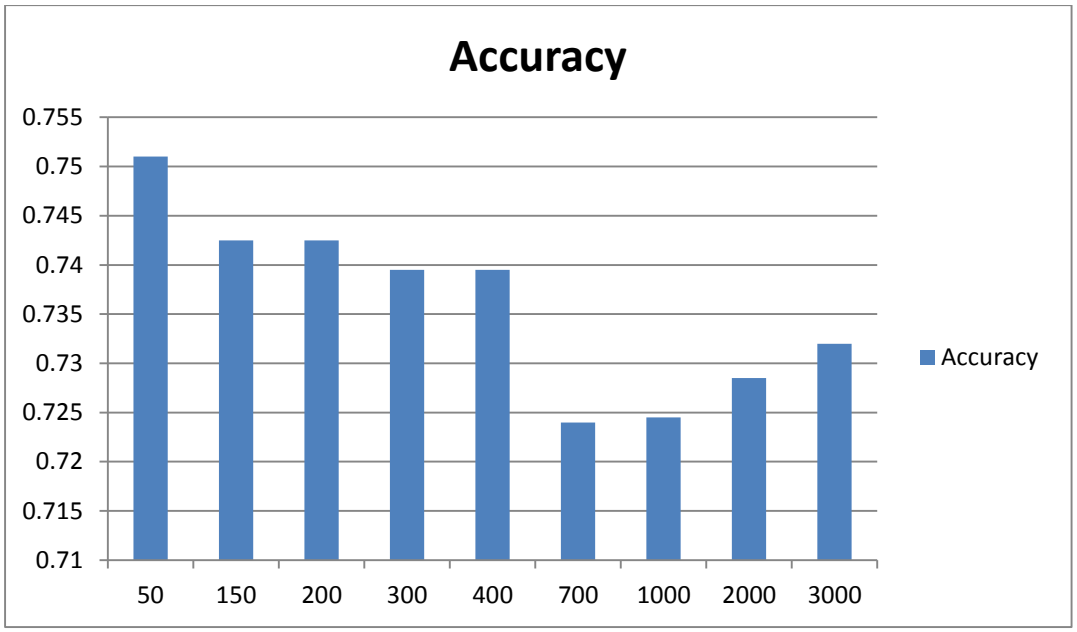


Figure B.16

Epoch Count vs. Accuracy Chart in Three-Layer Neural Network Training



جامعة النجاح الوطنية
كلية الدراسات العليا

استخدام الشبكة العصبية الاصطناعية للتنبؤ بنوع الجسيمات في فيزياء الطاقة العالية

إعداد
ايمان عدنان عثمان

إشراف
د. بكر عبدالحق

قدمت هذه الرسالة استكمالا لمتطلبات الحصول على درجة الماجستير في الحوسبة المتقدمة،
من كلية الدراسات العليا، في جامعة النجاح الوطنية، نابلس – فلسطين.

2024

استخدام الشبكة العصبية الاصطناعية للتنبؤ بنوع الجسيمات في فيزياء الطاقة العالية

إعداد

ايمان عدنان عثمان

إشراف

د. بكر عبدالحق

الملخص

لقد أصبحت مجالات تعلم الآلة أحد العناصر الأساسية في مجال البحث العلمي، حيث تقدم حلاً فعّالاً لمجموعة متنوعة من المسائل العلمية والهندسية. تبرز تقنية التعلم العميق بشكل خاص كأسلوب قوي لحل المشكلات المعقدة والمتعددة الأبعاد. ومن بين النماذج المميزة في هذا السياق تأتي الـ"Autoencoders"، وشبكات الاعتقاد العميقة "Deep Belief Networks"، وشبكات الاستدعاء التكرارية " Recurrent Neural Networks"، وشبكات التعلم التعزيزي "Reinforcement Learning".

تهدف هذه الدراسة إلى استغلال قدرات الشبكات العصبية الاصطناعية "Artificial Neural Networks" لبناء نظام تصنيف فعّال للقياسات التي تتم داخل المسارعات الذرية. والهدف الرئيسي هو تصنيف الـ"jets" إلى خمسة فئات متميزة: الكواركات الخفيفة "light quarks"، والجلونز "gluons"، "W and Z bosons"، والكواركات الأعلى "top quarks".

تشمل الدراسة استكشافاً شاملاً لهندسة الشبكات العصبية الاصطناعية بما في ذلك تحسين المعاملات المؤثرة مثل عدد الطبقات الخفية، ودوال التنشيط، ودوال التحسين، ودوال الخسارة، والعدد المثلى للحلقات التدريبية. يتم إجراء تجارب دقيقة لضبط هذه المعاملات ، بهدف تحقيق أعلى المستويات من دقة التصنيف مع تحقيق أقصى كفاءة حسابية في الوقت نفسه.

النتائج المتوقعة تشير إلى أن نموذج تصنيف الـ"jets" الخاص بنا سيكون لديه ليس فقط دقة تصنيف متفوقة ولكن أيضًا كفاءة حسابية جيدة. تسهم هذه الأبحاث بشكل كبير في تطوير منهجيات تعلم الآلة وتحمل وعداً كبيراً لتطبيقات متنوعة تمتد عبر الطيف العلمي.

الكلمات المفتاحية: تعلم الآلة ، شبكات عصبية اصطناعية ، الشبكات العصبية التصنيفية ، دوال التنشيط، ودوال التحسين،عدد الحلقات التدريبية.