

Comparative Analysis of Neural Networks: MCP vs. MLP Architectures

1. Introduction: -

1.1. The Notion and Essence of Neural Network Research

When answering the question “What are neural networks?”, we can say that in simple terms, neural networks are computer systems inspired by the way the human brain works. Just like the brain uses a network of interconnected neurons to process information, a neural network is composed of artificial "neurons" that work together to solve problems. These artificial neurons are organized in layers and use mathematical operations to learn patterns and relationships from data.

1.1.1. Why Are Neural Networks Important?

Neural networks are a cornerstone of artificial intelligence (AI), specifically machine learning. Their ability to learn and generalize from data makes them uniquely suited to solve complex problems that are difficult to program explicitly. Unlike traditional programming, where specific rules and conditions must be defined, neural networks learn directly from examples. This makes them powerful tools for tasks like pattern recognition, where neural networks can identify patterns in large datasets, such as recognizing faces in photos or detecting fraud in financial transactions.

1.1.2. Real-World Applications of Neural Networks

Neural networks are transforming various industries and improving our daily lives in many ways. Here are a few well known applications of different types of neural networks:

- **Self-Driving Cars:** Neural networks process data from sensors, cameras, and radar to understand the vehicle's surroundings. They help in tasks like object detection (e.g., identifying pedestrians or traffic signs) and decision-making (e.g., when to brake or change lanes).
- **Image Recognition:** Neural networks are the foundation of modern image recognition systems. Applications include facial recognition on smartphones, diagnosing medical conditions from X-rays or MRIs, and tagging photos on social media.
- **Natural Language Processing (NLP):** Neural networks power tools like voice assistants (e.g., Siri, Alexa), language translation (e.g., Google Translate), and text generation (e.g., chatbots or creative writing tools).
- **Healthcare:** Neural networks are used to predict diseases, design personalized treatment plans, and analyze genetic data. For instance, they can help in detecting cancers at early stages by analyzing medical images with greater accuracy than human experts.

1.1.3. The importance of Neural Network Research

Research in neural networks particularly aims to make these systems even more powerful, efficient, and adaptable. By improving their design, researchers can enable neural networks to handle larger and more diverse datasets, learn faster, and provide better predictions. This

research can drive advancements across industries, making technologies smarter and more accessible to everyone.

1.2. General Setting of Perceptron Online Learning

1.2.1. The Perceptron: A Simple Neural Network

The perceptron is one of the simplest and earliest types of neural networks. It is designed for binary classification tasks, where the goal is to categorize data points into one of two classes (e.g., "yes" or "no"). Despite its simplicity, the perceptron laid the foundation for modern neural network architectures.

A perceptron consists of a single layer of artificial neurons. Each neuron takes multiple inputs, processes them through a simple mathematical formula, and produces a single output. If the output exceeds a certain threshold, the perceptron classifies the input as one class; otherwise, it classifies it as the other.

1.2.2. Online Learning in the Perceptron

Online learning refers to a training process where the model learns incrementally. Instead of processing all data at once, the perceptron updates its weights immediately after being presented with each data point. This approach is particularly useful in scenarios where data arrives sequentially, or the dataset is too large to process all at once.

In the perceptron's online learning process:

1. Data points are fed to the model one at a time.
2. After each prediction, the model checks if it made a mistake.
3. If the prediction is incorrect, the perceptron updates its weights to correct the error.

This dynamic tuning allows the perceptron to improve continuously as it sees more data.

1.2.3. Key Components of the Perceptron

1. **Input Features:** These are the numerical values that represent the characteristics of the data point (e.g., pixel values in an image or temperature and humidity in weather data).
2. **Weights:** Each input feature is associated with a weight, which represents the importance of that feature in determining the output. Initially, weights are assigned small random values and are adjusted during the learning process.
3. **Activation Function:** We can think of the activation function as the decision-making part of the perceptron. After combining the inputs and their weights, the perceptron checks if the combined value crosses a certain threshold.

If it does, the perceptron says, "Yes, this belongs to the first class."

If it doesn't, the perceptron says, "No, this belongs to the second class."

It's like a simple test: if a number is bigger than zero, the answer is one thing; if it's zero or less, the answer is something else. This process ensures that the perceptron makes a clear and definite decision for each data point.

4. **Weight Update Rule:** We can imagine the perceptron as a student learning to classify things correctly. When it makes a mistake, it adjusts its "thinking process" to avoid making the same mistake again.

If the perceptron gets an answer wrong, it looks at how far off it was and adjusts the importance (or "weight") of each input.

For example, if it misclassified a flower based on petal length, it might decide to pay more attention to petal width next time.

The weight update is like a small correction after every mistake, helping the perceptron get better with each data point it sees.

1.3. Rationale and Ways of Hyperparameter Tuning

1.3.1. Definition and Role of Hyperparameters

Hyperparameters are the configurations external to the model that influence its performance. Examples include the **learning rate**, which determines how much to change the model in response to the estimated error each time the model weights are updated; the **number of hidden layers** in a neural network, which affects the model's capacity to learn complex patterns; and the **number of epochs**, which defines how many times the learning algorithm will work through the entire training dataset. Properly setting these hyperparameters is very important, as they determine how well the model will learn and generalize from the training data.

1.3.2. Importance of Tuning Hyperparameters

Tuning hyperparameters is essential for enhancing model performance. Proper tuning helps in achieving a balance between **overfitting** and **underfitting**. **Overfitting** occurs when a model learns the training data too well, including its noise, resulting in poor generalization to unseen data. Conversely, **underfitting** happens when a model is too simple to capture the underlying trends in the data, leading to poor performance on both training and test datasets. By fine-tuning hyperparameters, one can improve the model's ability to generalize, ultimately leading to better predictions on new data.

1.3.3. Methods for Hyperparameter Tuning

There are several methods for tuning hyperparameters:

1. **Gradient-Based Optimization:** Techniques like gradient descent can be adapted to tune hyperparameters by estimating gradients of the validation error concerning the hyperparameters.
2. **Automated Approaches:** Methods such as **Bayesian optimization** build a probabilistic model of the objective function and use it to choose the most promising

hyperparameters to evaluate. This approach can be more efficient than grid or random search, particularly in high-dimensional spaces.

Each of those methods have their own advantages and limitations, and the choice often depends on the specific problem, available computational resources, and the desired balance between exploration and exploitation.

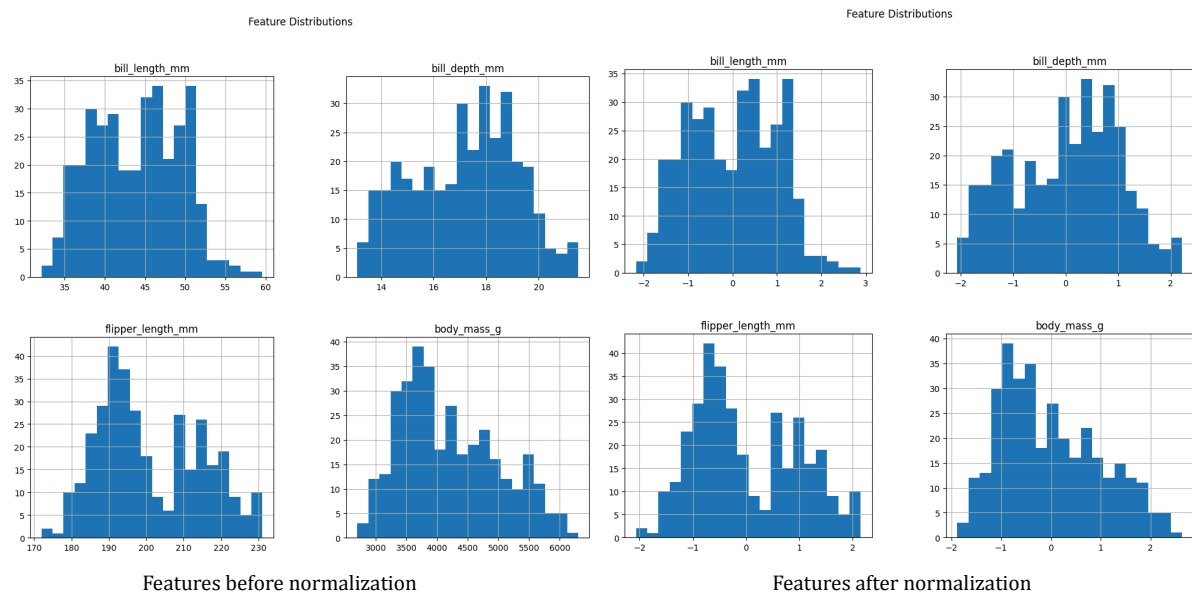
1.4. Data preparation for neural network data analysis

Cleaning and organizing your data is a critical step as the quality of the input data significantly affects model performance.

Key steps in this process include normalizing values to ensure that all features contribute equally to the learning process, converting categorical variables into numerical representations through encoding, and splitting the dataset into training, validation, and test sets.

Normalization helps to stabilize and speed up the training process by preventing features with larger ranges from dominating the learning algorithm. Encoding allows the model to interpret categorical data correctly, as neural networks require numerical inputs.

Additionally, proper data splitting ensures that the model can generalize well to unseen data, thereby reducing the risk of overfitting and improving overall accuracy. These preprocessing steps help the neural network learn effectively, minimizing mistakes that could arise from inconsistencies or biases in the data.

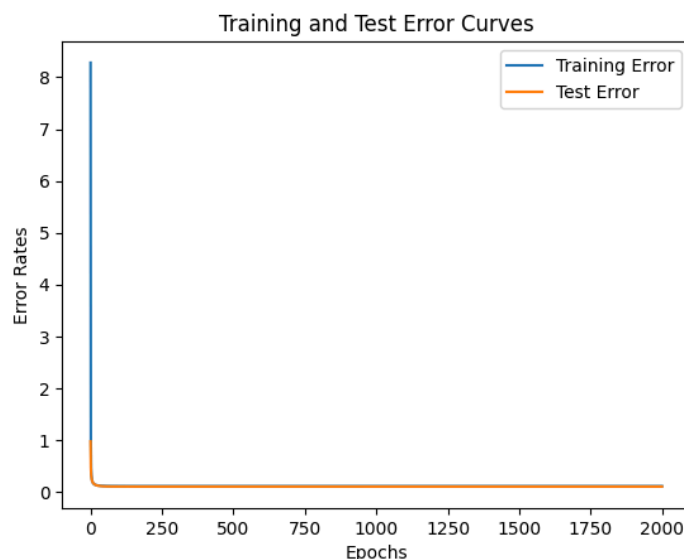


2. Experiments: -

2.1. Multiclass Perceptron Model (MCP): -

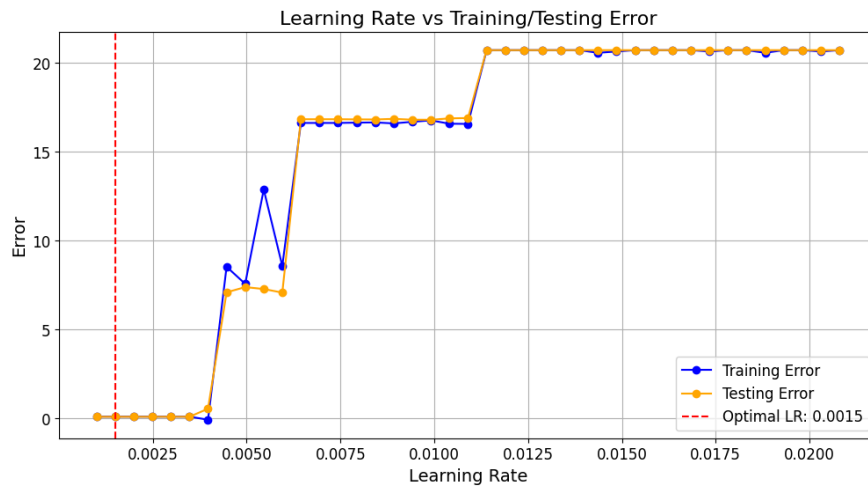
I began by splitting the dataset into training and testing sets, using an 80% training and 20% testing ratio. This ensures that the model is trained on a sufficient amount of data while retaining enough unseen data for accurate evaluation. I set a small learning rate of 0.001 to control how much the model adjusts its weights during training, allowing for stable convergence. A fixed number of epochs, specifically 100, was chosen to limit the iterations through the training data. During each epoch, I utilized the ReLU (Rectified Linear Unit) activation function, which helps to improve the training speed and effectiveness by introducing non-linearity in the model and achieve a test accuracy of 100% as depicted in the plot graph below.

To assess the model's effectiveness, I plotted the following error rates for both training and testing datasets to visualize the model's performance over time.



I conducted further experiments using the optimal epoch value determined in Part A, maintaining the same data splitting ratio as before. I varied the learning rate within a specified range, using `learning_rates = np.linspace(0.001, 0.05, 100)`, to identify the most effective learning rate for the model.

To assess the model's performance, I visualized both the training and testing error curves across this range of learning rates, similar to the methodology used in the previous section.

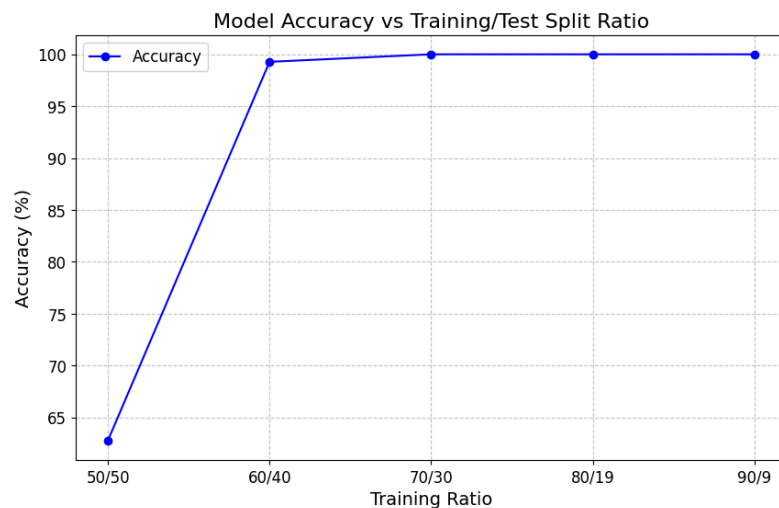


This allowed me to pinpoint the learning rate that resulted in the minimum testing error. Ultimately, I estimated the model's accuracy at this optimal learning rate and obtained the following outcomes:

Activation Function	Best Learning rate at 2000 epochs
Sigmoid	0.084 (97.1% Accuracy)
Gaussian Function	0.001 (37.68% Accuracy)
Re-Lu	0.015 (100% Accuracy)

These results demonstrate the significant impact of the activation function and learning rate on model performance. Specifically, the ReLU activation function, combined with an appropriately chosen learning rate, achieved the best results, indicating the importance of hyperparameter tuning in neural network training.

I then utilized the optimal epoch value and the best learning rate identified and systematically varied the splitting ratio of the training sample size and recorded the model's accuracy for each ratio. To achieve a comprehensive evaluation, I tested five different splitting ratios (50% to 100%), ensuring they were evenly or oddly distributed.

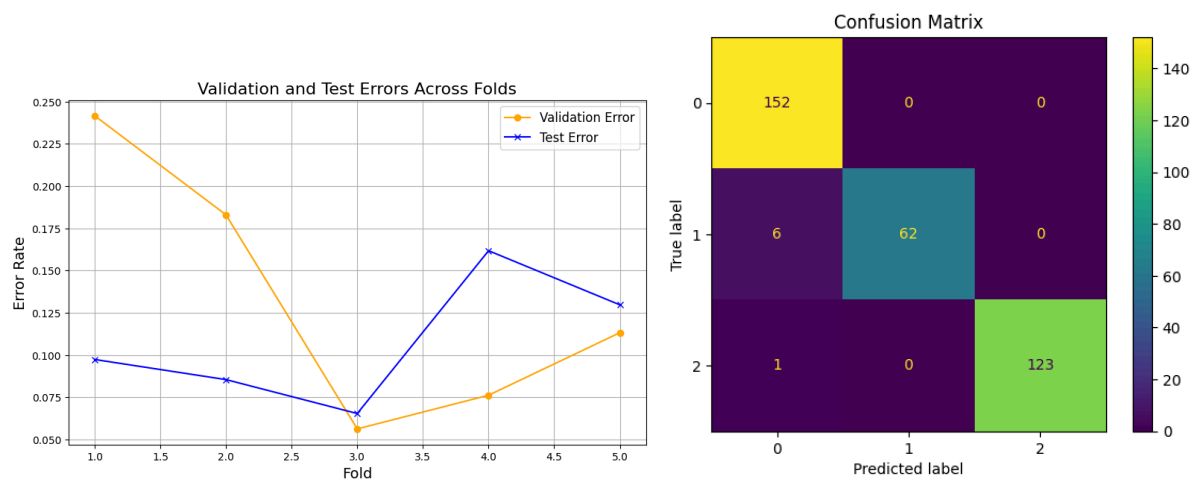


I then implemented cross-validation in my machine learning assignment by splitting the data into 5 subsets (folds). The model was trained and validated 5 times, with each fold being used for validation while the remaining folds were utilized for training.

I have achieved the following results through the implementation of cross-validation:

- **Mean Validation Error:** 0.134045
- **Mean Test Error:** 0.107918

These metrics indicate that the model has performed reasonably well, with the test error being lower than the validation error, suggesting good generalization to unseen data.

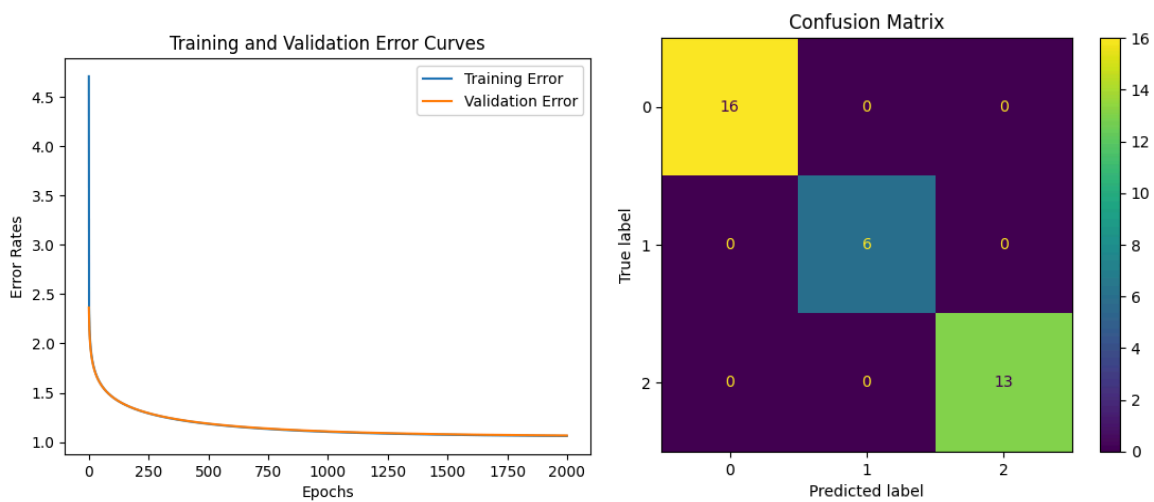


Some biases that could be investigated in this dataset. For example, a potential bias may arise from the limited geographic scope of the data collection, which focuses solely on three species of penguins found in specific regions of Antarctica. Also, a potential bias arises from the unequal representation of the three penguin species: Adelie, Gentoo, and Chinstrap.

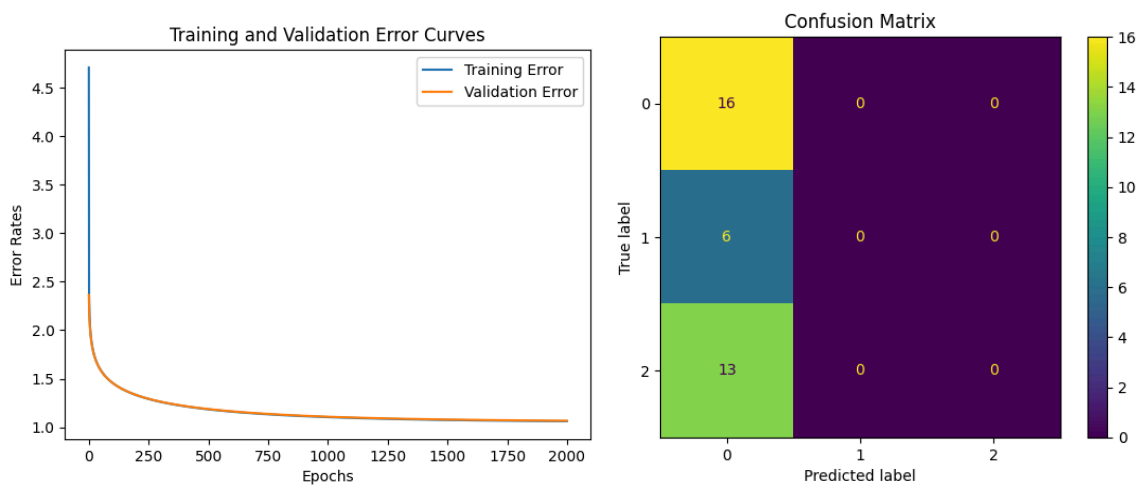
2.2. MultiLayer Perceptron Model (MLP): -

For this experiment I have expanded the model to include hidden layers between the input and output layers, starting with a single hidden layer. I chose a suitable number of perceptrons for this hidden layer, in this case between 3 and 4 as we have 4 features and 3 labels, following the guideline of selecting a number that lies between the number of input features and the number of output labels. This choice aligns with common practices in neural network design, as it can effectively capture complex patterns in the data while maintaining a balanced architecture.

For the 90-10 split ratio, I achieved 100% accuracy using the ReLU activation function in my model, which includes a hidden layer with 3 neurons, along with a 5-fold cross-validation approach. This result demonstrates the effectiveness of using ReLU in capturing the complexities of the Palmer Penguins dataset, highlighting its capability to optimize performance in neural networks.



I then extended the model to include multiple hidden layers with varying configurations, implementing different numbers of perceptrons in each layer. However, the model's performance was not great, achieving just above 45% accuracy.



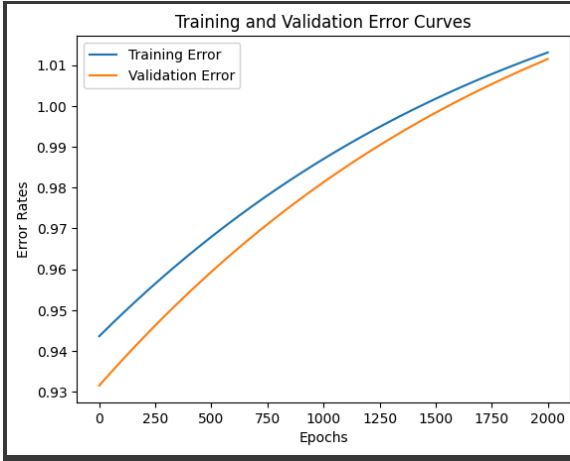
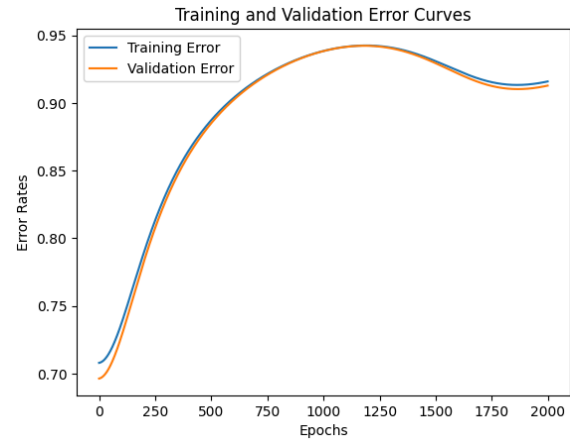
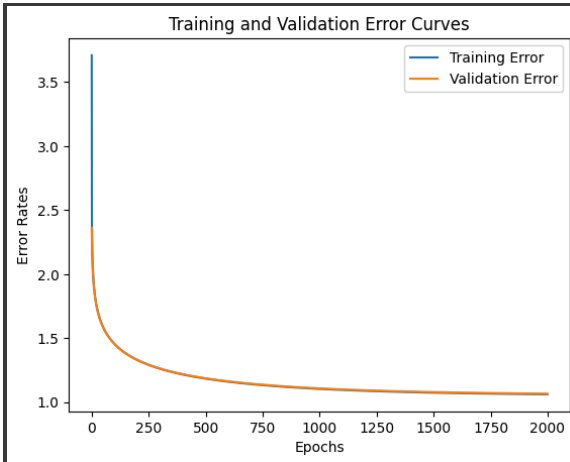
I then utilized the respective optimal learning rates for each activation function, which were determined during previous evaluations. Despite the complexity introduced by multiple hidden layers, the results did not meet expectations, indicating that the chosen configurations may not have been ideal for this dataset or task.

For reference, here are the learning rates I used for each activation function:

Sigmoid: 0.084

Gaussian Function: 0.001

Re-Lu: 0.015

Activation Function	Performance
Sigmoid	 <p>Training and Validation Error Curves for Sigmoid activation function. The graph shows Training Error (blue line) and Validation Error (orange line) over 2000 epochs. Both errors increase monotonically, with Training Error starting at ~0.945 and ending at ~1.01, and Validation Error starting at ~0.93 and ending at ~1.005.</p>
Gaussian Function	 <p>Training and Validation Error Curves for Gaussian activation function. The graph shows Training Error (blue line) and Validation Error (orange line) over 2000 epochs. Both errors follow a similar path, peaking around epoch 1250 at ~0.94 and then slightly decreasing to ~0.915 by epoch 2000.</p>
Re-Lu	 <p>Training and Validation Error Curves for Re-Lu activation function. The graph shows Training Error (blue line) and Validation Error (orange line) over 2000 epochs. Both errors decrease rapidly from epoch 0 to 250, then level off. Training Error starts at ~3.6 and ends at ~1.05, while Validation Error starts at ~2.4 and ends at ~1.05.</p>

These results suggest that while adding complexity to the model can be beneficial, it is crucial to carefully choose the architecture and parameters to ensure effective learning and generalization.

3. Conclusion: -

The analysis of the Palmer Penguins dataset using neural networks has highlighted some critical limitations alongside its potential. Despite employing a sophisticated model with multiple hidden layers, the neural network achieved just over 45% accuracy, which is significantly lower than the performance of K-Nearest Neighbors (KNN) and Naïve Bayes classifiers. On the contrary, it performs really well with just 1 hidden layer. These limitations may stem from the model's complexity. In contrast, KNN benefited from cross-validation and principal component analysis (PCA), effectively capturing the spatial relationships between features, while Naïve Bayes provided stable results without extensive tuning.

When comparing the effectiveness of these approaches, KNN demonstrated optimal performance with a training-testing split of 60:40 and tuning K values between 6 and 8, maximizing accuracy and robustness. Naïve Bayes, known for its speed and interpretability, excelled in scenarios with categorical data, providing consistent results even without significant preprocessing. The neural network's struggles with accuracy reflect the challenges of overfitting and the need for careful architecture design and hyperparameter tuning, as its complexity often requires more extensive data and fine-tuning than simpler models.

To improve the neural network's performance, hyperparameter optimization techniques like grid search could enhance model tuning, while regularization methods can help prevent overfitting. Additionally, refining input features through feature engineering and considering alternative model architectures may yield better results. Overall, this comparison underscores the importance of selecting the appropriate model for the dataset and classification task, suggesting that while KNN and Naïve Bayes are effective for the penguin dataset, neural networks hold potential for more complex applications if optimized correctly.

Source Code: -

https://colab.research.google.com/drive/1lp5k1Ui6blyHq2v9c_uV3zh-WDQVxwPE?usp=sharing

References: -

1. OpenAI (2024) *Understanding Neural Networks: The Building Blocks of AI*. Available at: <https://www.openai.com>
2. Simplilearn, 2022. What is a Perceptron? Understanding the Basics of Perceptron in Machine Learning. [online] Available at: <https://www.simplilearn.com/tutorials/deep-learning-tutorial/perceptron>
3. Snoek, J., Larochelle, H. and Adams, R.P., 2012. *Practical Bayesian Optimization of Machine Learning Algorithms*. In: Advances in Neural Information Processing Systems (NIPS), pp. 2951-2959. [online] Available at: <https://arxiv.org/abs/1206.2944>