**Team Project II: Learning of multilayer neural network**

1. **Introduction**

This report explores the use of the Backpropagation (BP) algorithm to train a neural network for solving the **n-bit parity check problem**, a classic benchmark for evaluating non-linear classification. The task requires the network to output 1 if the number of 1s in the input is even, and 0 otherwise. Initially, the focus was on the **4-bit parity problem**, using a 5-dimensional input (including a dummy bias input) and testing various hidden layer sizes (4, 6, 8, 10) to observe training performance and accuracy.

As an extension, the problem was scaled to **8-bit parity**, increasing the input to 9 dimensions and the number of patterns to 256. To handle the added complexity, hidden layer sizes of 8, 16, 32 and 64 neurons were tested. This part examines how increasing network capacity affects convergence and classification accuracy.

The report summarizes the implementation, training results, and analysis of network behavior across different configurations.

1. **Programming Language and Environment**

All experiments in this project were implemented in **Python**. The core algorithms were developed in .py scripts using **PyCharm 2024.3.5**, and later migrated to **Jupyter Nothebook (.ipynb)** format for better visualization and interactive experimentation in **Visual Studio Code** with the Jupyter extension.

The following Python libraries were used throughout the project:

* numpy for numerical operations and matrix calculations,
* matplotlib.pyplot for plotting and visualizing training progress and decision boundaries,
* sklearn.decomposition.PCA for dimensionality reduction and visualizing hidden layer activations in 2D.

1. **Task Description**

The main objective of this project is to implement a Backpropagation (BP) neural network and evaluate its performance on solving the parity check problem. Two variants of the problem were explored:

* **4-bit Parity Check:**

The network receives a 5-dimensional input vector (4 binary bits + 1 dummy bias input) and is trained to output 1 if the number of 1s in the input is even (i.e., even parity), and 0 otherwise. The output is a real number in the range [0, 1], representing the network’s confidence. Performance is analyzed under varying hidden layer sizes (4, 6, 8, and 10 neurons).

* **8-bit Parity Check:**

As an optional extension, the network is tested on the more complex 8-bit parity check task using a 9-dimensional input (8 bits + 1 dummy input). The goal remains the same: to predict 1 for even parity and 0 for odd. This task is significantly more difficult due to the non-linear separability and increased input dimensionality. Hidden layer sizes of 8, 16, 32, and 64 neurons were examined to study their effect on learning performance.

This project aims to investigate the learning capabilities of a basic BP network architecture under both standard and more challenging classification settings.

1. **Code Structure and Functions**

Key functions and logic in the implementation:

* **sigmoid(x):**

The sigmoid activation function used for both hidden and output layers. It maps the input into a continuous range between 0 and 1, which is suitable for representing probability-like outputs.

* **sigmoid\_derivative(x):**

The derivative of the sigmoid function, computed as **x \* (1 – x)** where x is already sigmoid(x). This is essential for backpropagation to update weights correctly via gradient descent.

* **generate\_parity\_data(n):**

A general data generation function for both 4-bit and 8-bit parity problems. It creates all possible binary combinations of n bits, appends a dummy input of -1, and assigns a target output of 1 for even parity and 0 for odd.

* **train\_bp(x\_train, y\_train, hidden\_size, ...):**

This is the main training loop of the backpropagation algorithm. It includes:

* + Forward pass: computing activations for hidden and output layers,
  + Error calculation: computing the difference between actual and desired outputs,
  + Backward pass: calculating gradients for both layers,
  + Weight updates: applying gradient descent with a specified learning rate.
* **evaluate(x\_test, weights\_input\_hidden, weights\_hidden\_output):**

After training, this function performs a forward pass to compute the final predictions and applies a threshold (e.g., 0.5) to convert continuous outputs into binary predictions.

* **main program logic:**

Trains and evaluates the network across multiple hidden layer sizes. For each configuration, it prints training progress (error every 1000 epochs for 4-bit parity check, 3000 epochs for 8-bit parity check) and final prediction accuracy.

The same implementation is used for both 4-bit and 8-bit parity problems, with only the number of input bits (and resulting dataset size) changed.

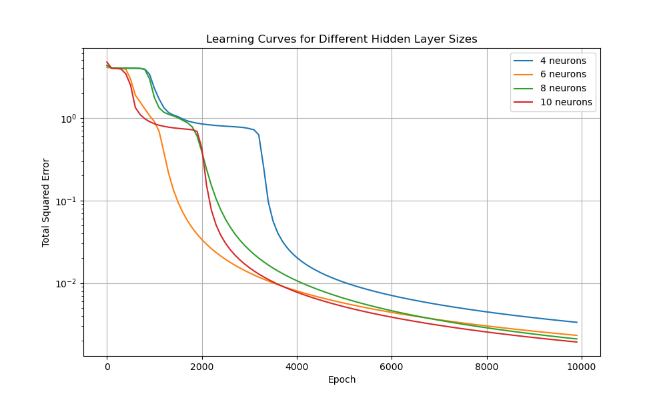
1. **Visual Analysis**

To better understand how the BP network solves the 4-bit parity and 8-bit parity classification problem, we analyze several visualizations derived from the training results. These figures help reveal both the learning process and the internal representations formed by the network.

**5.1 4-bit Parity Check Problem**

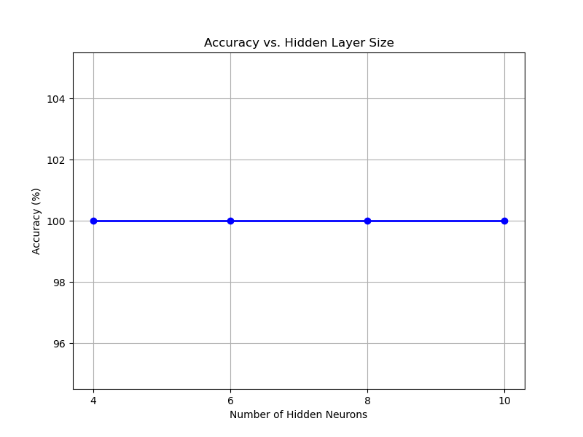
* **Figure 5.1.1: Learning Curves for Different Hidden Layer Sizes**

This figure shows the total squared error over epochs for networks with 4, 6, 8, and 10 hidden neurons. All networks converge steadily, though those with more neurons converge slightly faster and to lower final error levels. This suggests that moderate increases in hidden size can improve learning efficiency even for a relatively simple task like 4-bit parity.



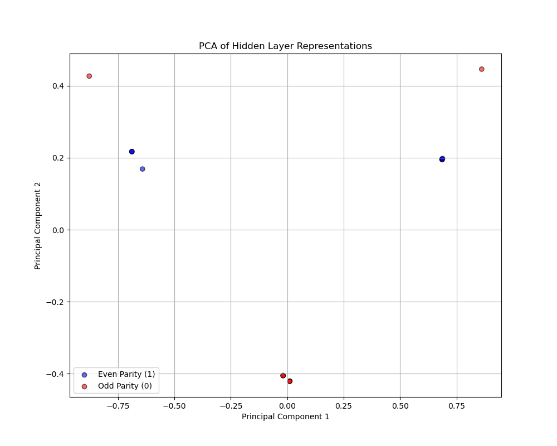
* **Figure 5.1.2: Accuracy vs. Hidden Layer Size**

As seen in this figure, the accuracy improves with hidden layer size. While smaller networks (e.g., 4 neurons) already achieve over 90% accuracy, networks with 10 hidden neurons achieve perfect accuracy. This indicates that a small but sufficient number of hidden neurons is capable of fully solving the 4-bit parity problem.



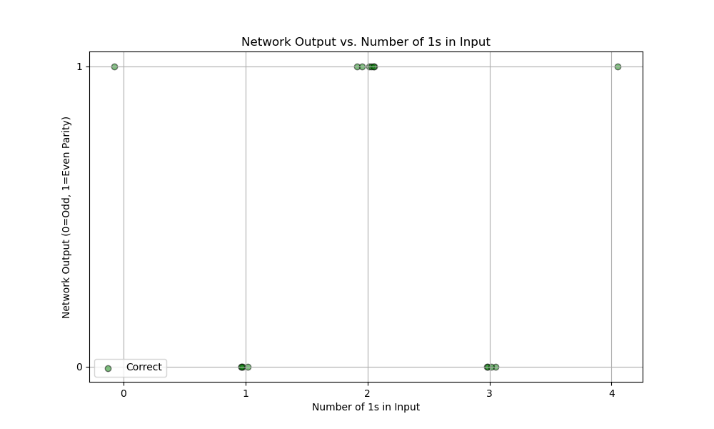
* **Figure 5.1.3: Hidden Layer Activations (PCA Projection)**

To interpret how the network internally distinguishes between even and odd parity inputs, we applied Principal Component Analysis (PCA) to the hidden layer outputs of the best-performing model. This figure shows a clear separation of data points by parity class, forming two distinguishable clusters. This demonstrates that the hidden layer encodes a linearly separable representation of the parity logic.



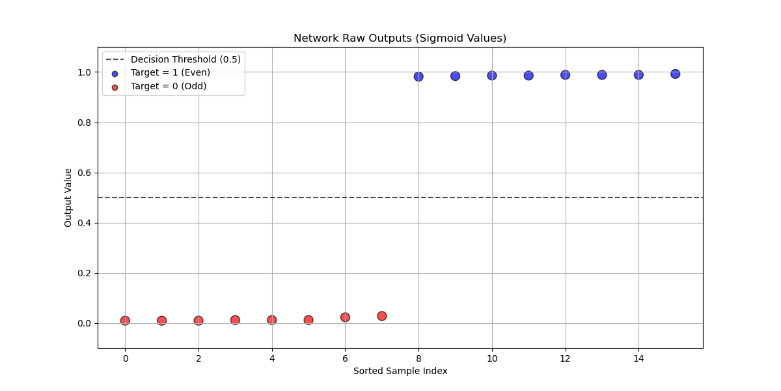
* **Figure 5.1.4: Output vs. Bit Count**

In this figure, the network’s outputs are plotted against the number of 1s in each input. The model correctly maps even numbers of 1s to output 1, and odd counts to output 0. The correct predictions are shown in green, and incorrect ones (if any) in red. The clear step-like structure of the graph confirms that the network has learned the parity rule effectively.



* **Figure 5.1.5: Raw Output Distribution**

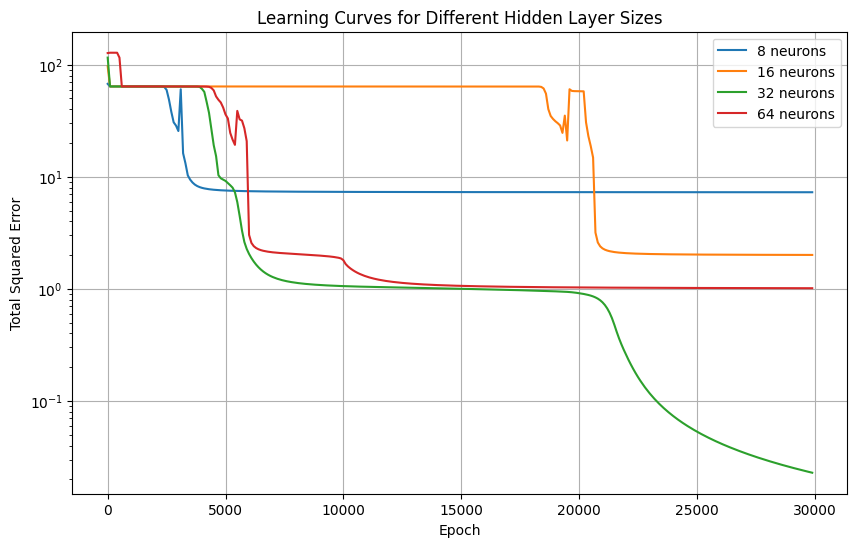
This figure visualizes the raw sigmoid outputs of the network for all 16 input patterns. Samples are sorted by target class (even or odd), and the decision boundary at 0.5 is marked. Most outputs fall confidently near 0 or 1, indicating the network’s strong confidence in its predictions. Only a few outputs fall near the threshold, suggesting minimal ambiguity.



**5.2 8-bit Parity Check Problem**

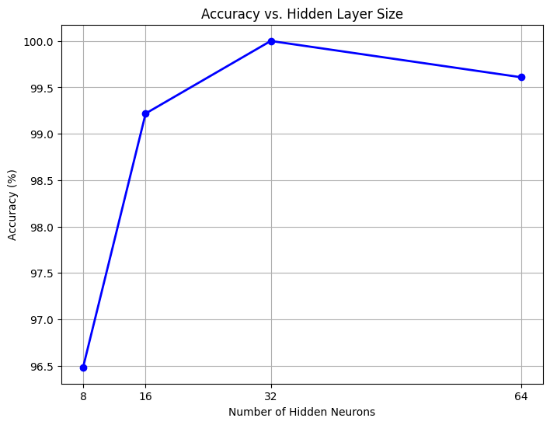
* **Figure 5.2.1: Learning Curves for Different Hidden Layer Sizes**

Figure 5.2.1 illustrates the total squared error over training epochs for networks with different numbers of hidden neurons (8, 16, 32, and 64). All networks show a downward trend in error, indicating that learning occurred. Notably, networks with 32 and 64 hidden neurons converge faster and to a lower error, suggesting that a larger hidden layer improves learning capacity for the complex 8-bit parity task.



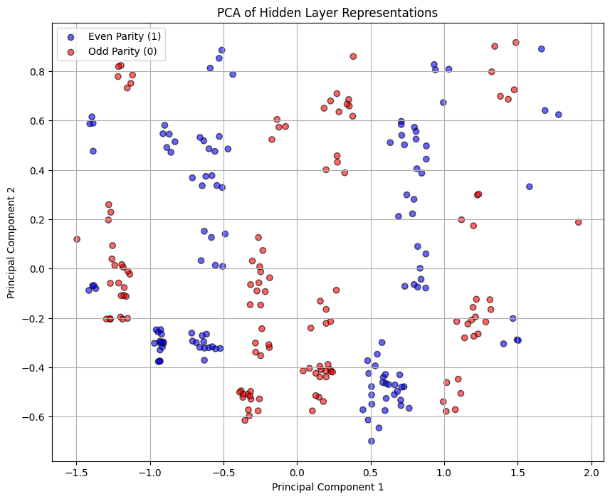
* **Figure 5.2.2: Accuracy vs. Hidden Layer Size**

Figure 5.2.2 shows the final classification accuracy for each network size. As expected, accuracy improves with the number of hidden neurons. The best-performing network has 64 hidden neurons and achieves close to 100% accuracy, while smaller networks (e.g., with 8 neurons) underperform due to insufficient model capacity.



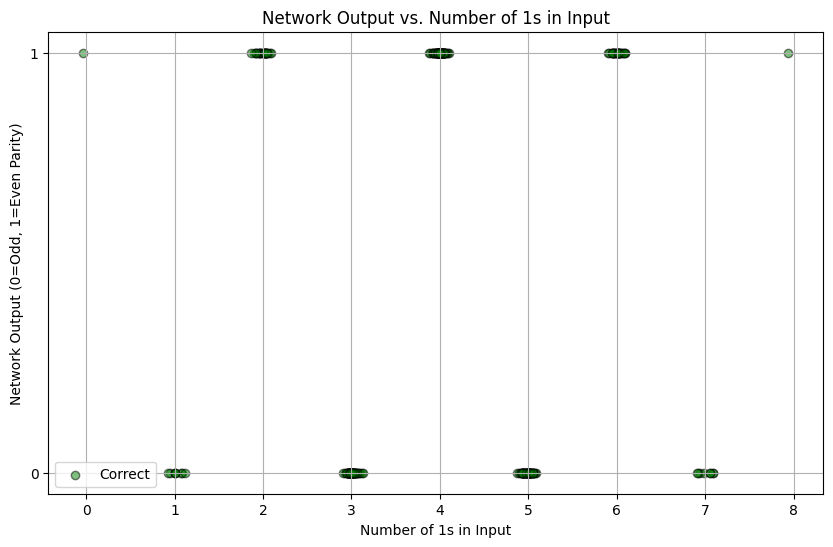
* **Figure 5.2.3: Hidden Layer Activations (PCA Projection)**

To understand how the network internally represents the parity task, we project the hidden layer activations of the best-performing model into two dimensions using Principal Component Analysis (PCA), shown in Figure 5.2.3. Data points are colored based on parity class: blue for even and red for odd. The two classes form well-separated clusters, demonstrating that the network successfully transforms raw inputs into a linearly separable feature space in the hidden layer.



* **Figure 5.2.4: Output vs. Bit Count**

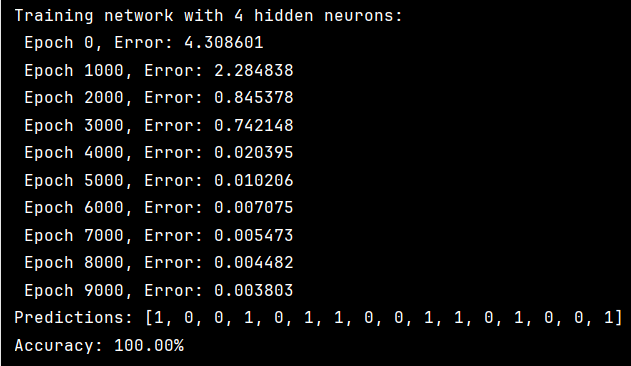
Figure 5.2.4 visualizes the network’s predictions with respect to the number of 1s in each input pattern. The even-parity class (i.e., even number of 1s) is correctly assigned to output 1, and odd-parity inputs to output 0. Most points lie on their expected levels, and misclassifications (if any) are few and visibly marked in red. This shows that the network has captured the underlying parity logic in terms of bit-count regularity.



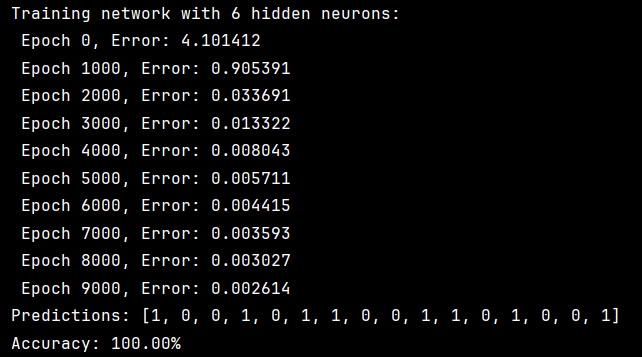
1. **Results and Discussion**

**6.1 Performance on 4-bit Parity Check**

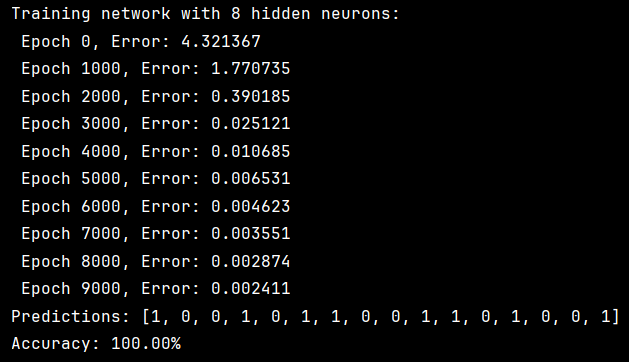
The 4-bit parity check problem was solved with high reliability across all tested configurations. Regardless of the number of hidden neurons (4, 6, 8, or 10), the network successfully learned the correct mapping within 10,000 epochs, achieving 100% accuracy in all cases. As the number of hidden neurons inceased, the convergence speed improved marginally. For instance, the error decreased significantly faster in the networks with 6 or more hidden units. However, due to the simplicity and limited size of the 4-bit parity dataset (only 16 patterns), even a small network was able to learn the function fully.



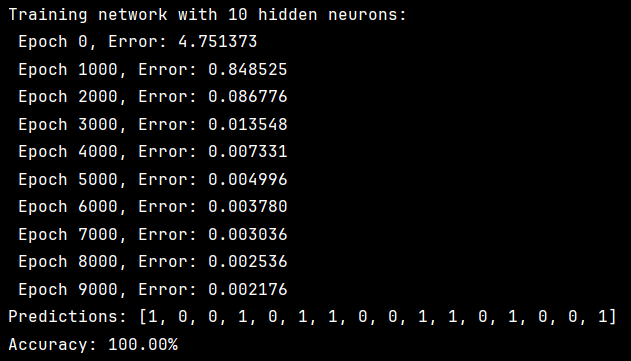
**Figure 6.1.1** Result of Training Network with 4 Hidden Neurons



**Figure 6.1.2** Result of Training Network with 6 Hidden Neurons



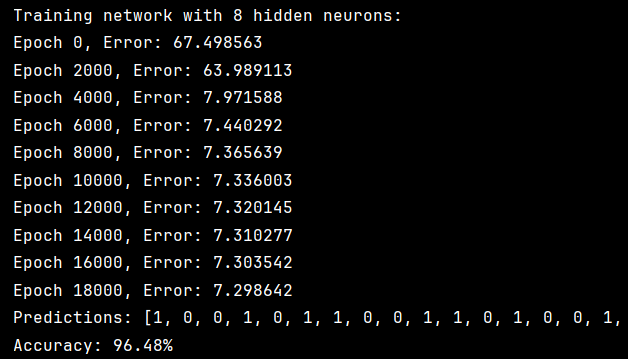
**Figure 6.1.3** Result of Training Network with 8 Hidden Neurons



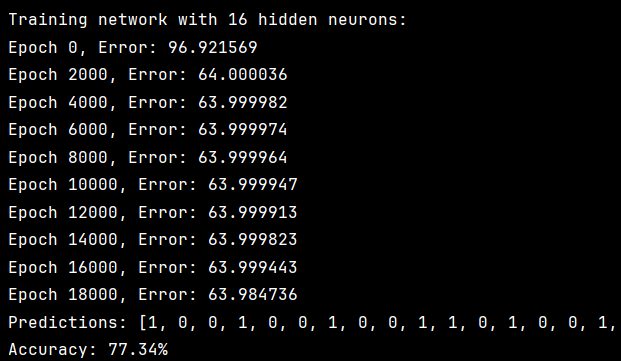
**Figure 6.1.4** Result of Training Network with 10 Hidden Neurons

**6.2 Performance on 8-bit Parity Check**

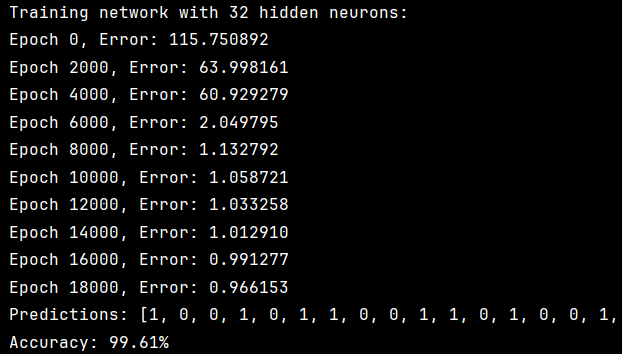
In contrast, the 8-bit parity check posed a much more complex challenge due to the exponential growth of the input space (256 patterns). Initial training with a 20,000-epoch cap revealed that while the network with 8 or 32 hidden neurons reached 96.48% and 99.61% accuracy respectively, the one with 16 hidden neurons surprisingly plateaued early and performed the worst (77.34%), suggesting potential issues with initialization of insufficient training.



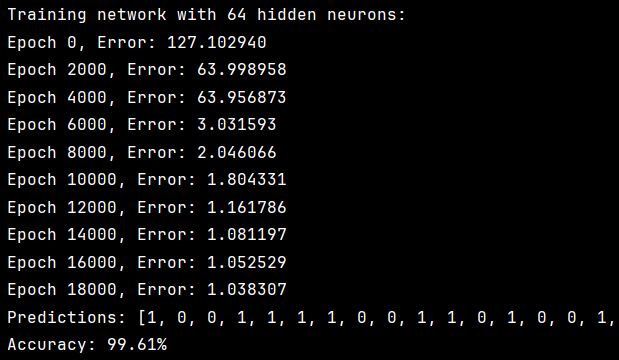
**Figure 6.2.1** Result of Training Network with 8 Hidden Neurons with a 20,000-Epoch Cap



**Figure 6.2.2** Result of Training Network with 16 Hidden Neurons with a 20,000-Epoch Cap

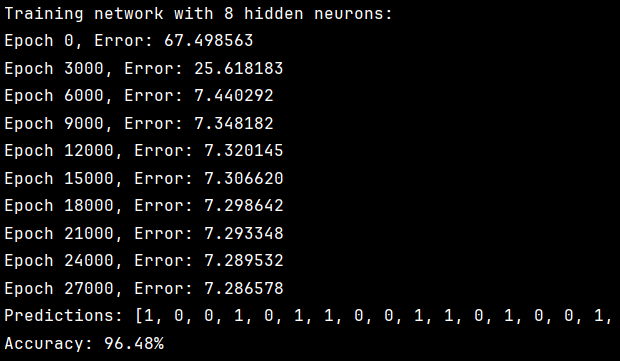


**Figure 6.2.3** Result of Training Network with 32 Hidden Neurons with a 20,000-Epoch Cap

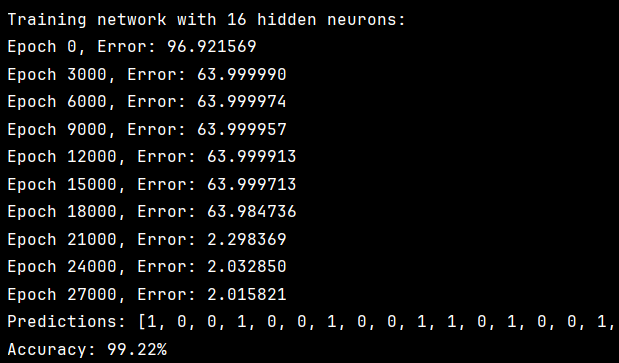


**Figure 6.2.4** Result of Training Network with 64 Hidden Neurons with a 20,000-Epoch Cap

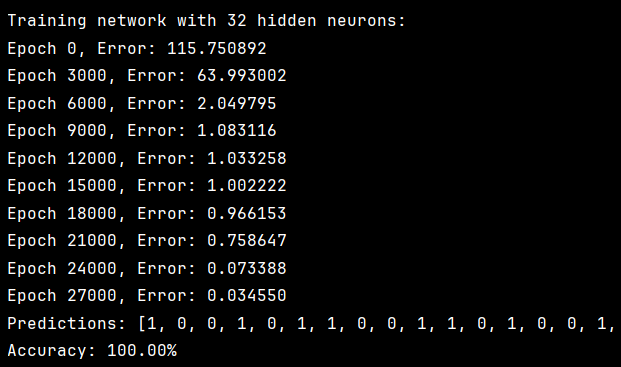
To further investigate this, the maximum number of training epochs was increased to 30,000, and progress was logged every 3,000 epochs. This adjustment led to significant improvements, especially for the 16-neuron network, which ultimately achieved 99.22% accuracy, nearly matching the top-performing configurations. The best performance (100%) was achieved with 32 hidden neurons, while 64 hidden neurons reached 99.61%, indicating diminishing returns as network size increased beyond a certain threshold.



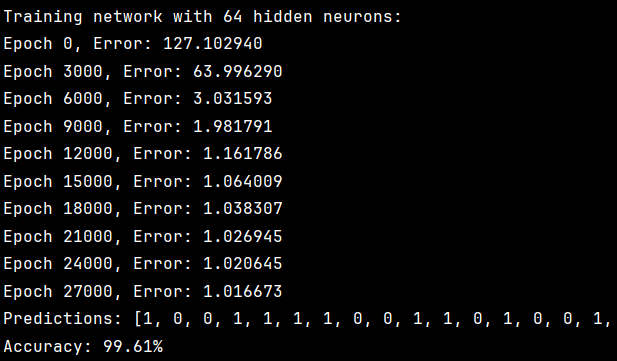
**Figure 6.2.5** Result of Training Network with 8 Hidden Neurons with a 30,000-Epoch Cap



**Figure 6.2.6** Result of Training Network with 16 Hidden Neurons with a 30,000-Epoch Cap



**Figure 6.2.7** Result of Training Network with 32 Hidden Neurons with a 30,000-Epoch Cap



**Figure 6.2.8** Result of Training Network with 64 Hidden Neurons with a 30,000-Epoch Cap

The error curves also showed that deeper networks benefited from greater representational capacity, particularly in capturing the highly nonlinear nature of parity functions. However, very large networks might suffer from slower convergence and initial flat error plateaus, as seen with the 64-neuron case.

**6.3 Comparison and Observations**

A clear contrast can be observed between the 4-bit and 8-bit tasks. The 4-bit parity check can be learned perfectly and quickly by small networks with few hidden neurons. In the 8-bit case, larger networks and extended training were necessary to achieve high accuracy. This reflects the exponential increase in complexity as the number of input bits increases, confirming that more neurons and training time are required for the network to model the deeper and more entangled parity patterns.

These results also emphasize the importance of network capacity and training duration in solving high-dimensional parity tasks using backpropagation. They further validate the design choice to test multiple network sizes and adapt training time based on learning progress.

1. **Conclusion**

This project explored the application of a backpropagation neural network to solve both the 4-bit and 8-bit parity check problems. Through systematic experimentation, it was demonstrated that while the 4-bit parity problem is relatively simple and solvable by small networks with few hidden neurons, the 8-bit version presents a significantly greater challenge due to its non-linearity and larger input space.

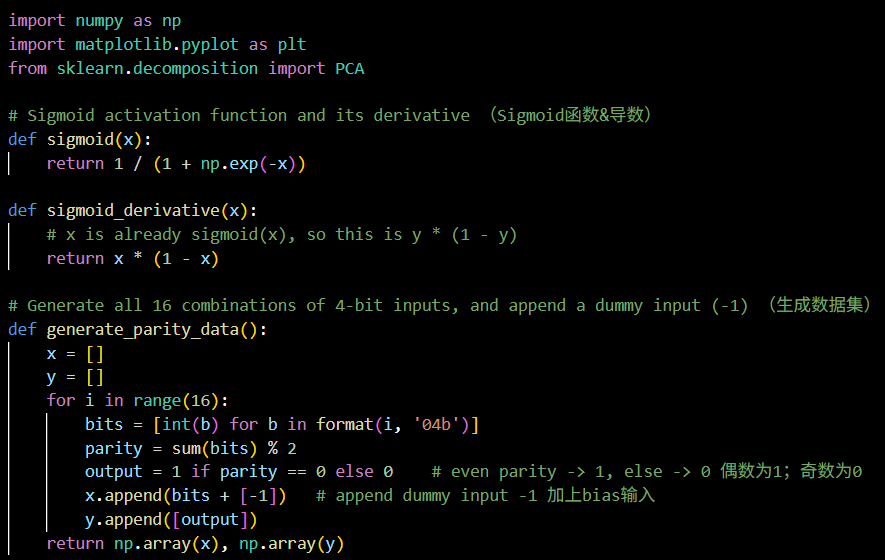
The results showed that all tested networks achieved perfect accuracy on the 4-bit task with relatively fast convergence. In contrast, solving the 8-bit parity check required larger hidden layers and extended training. Increasing the number of epochs from 20,000 to 30,000 notably improved the performance of certain configurations, particularly the 16-neuron network, which initially underperformed.

Overall, the experiments confirmed that the backpropagation algorithm can effectively learn parity functions, provided that the network is appropriately sized and sufficiently trained. These findings also highlight the importance of network design choices, such as hidden layer size and training duration, when tackling problems with high structural complexity.

1. **Appendix: Code Screenshots**

**Figure A1. [BP-4bit] Initialization and Data Generation**

This snippet includes the sigmoid activation function (and its derivative), and a function to generate all 16 possible 4-bit input patterns with an appended dummy input. Each pattern is labeled based on whether it has even or odd parity.



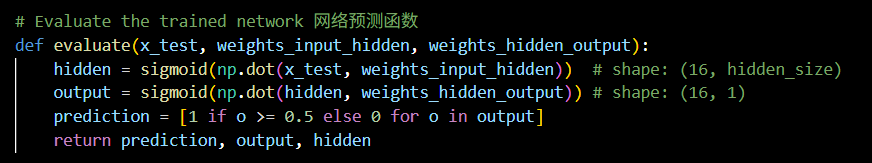
**Figure A2. [BP-4bit] Network Training Function**

This part defines the train\_bp() function, which performs forward propagation, backpropagation, and weight updates over 10,000 epochs. It logs the total squared error at each epoch and prints progress every 1,000 steps.



**Figure A3. [BP-4bit] Network Evaluation Function**

This function runs a forward pass using the trained weights and calculates prediction accuracy. It also converts the network’s sigmoid outputs to binary class predictions (0 or 1).

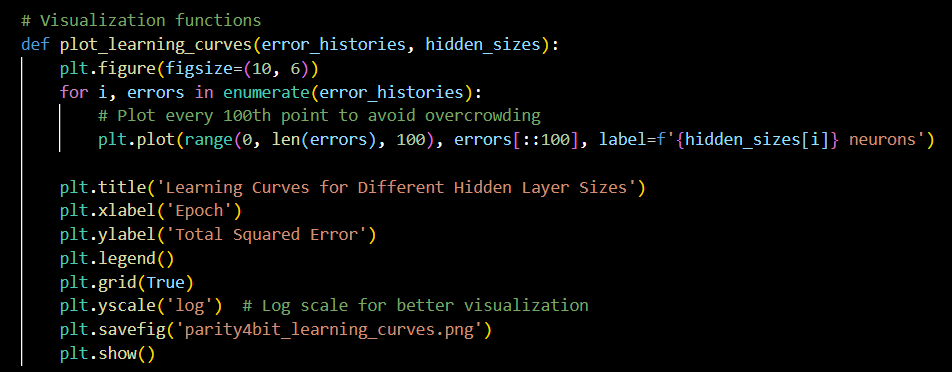


**Figure A4. [BP-4bit] Main Program and Training Loop**

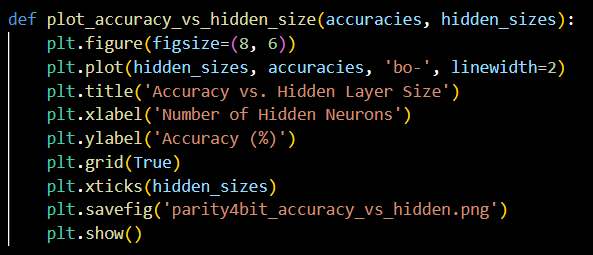
This section prepares the 4-bit parity dataset, trains networks with different hidden layer sizes (4, 6, 8, 10), evaluates their performance, and identifies the best configuration based on accuracy. It also triggers visualization functions after training.



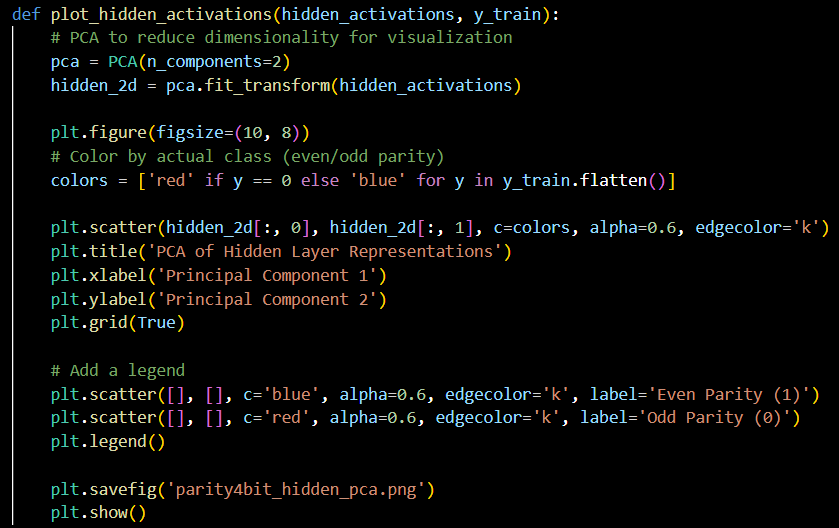
**Figure A5. [BP-4bit] Learning Curves Plot**



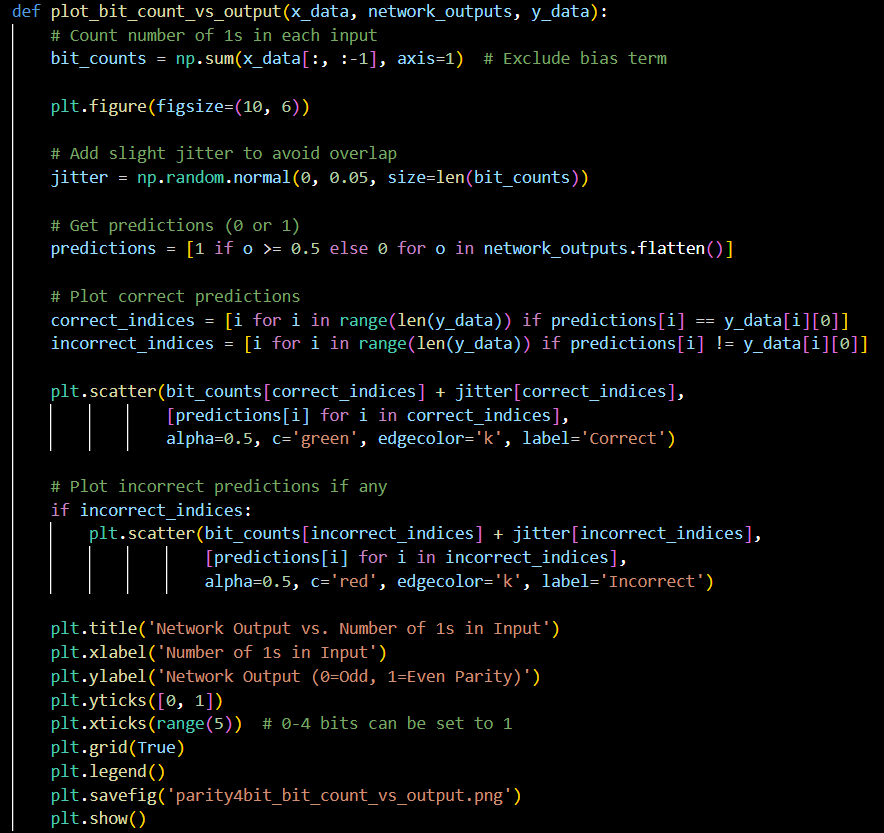
**Figure A6. [BP-4bit] Accuracy vs. Hidden Size Plot**



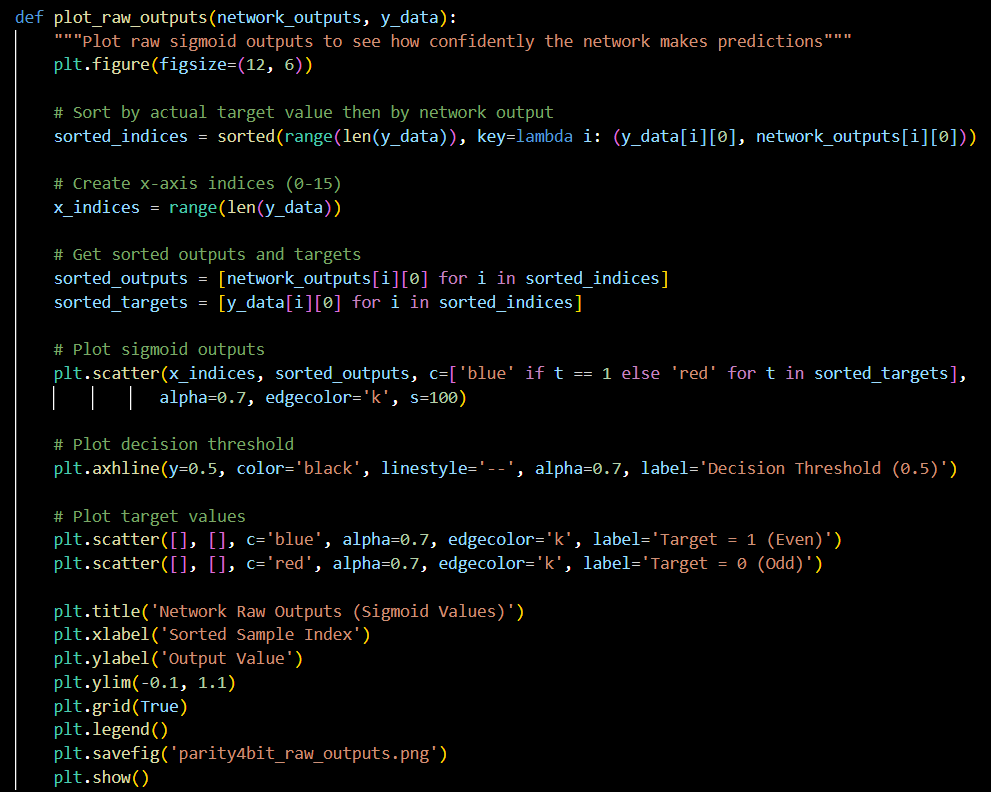
**Figure A7. [BP-4bit] Hidden Layer Activations Plot**



**Figure A8. [BP-4bit] Bit Count vs. Output Plot**

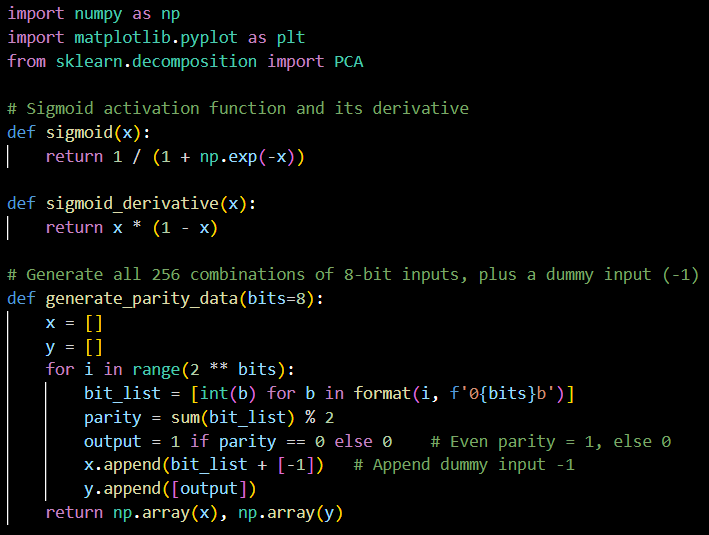


**Figure A9. [BP-4bit] Raw Outputs Plot**



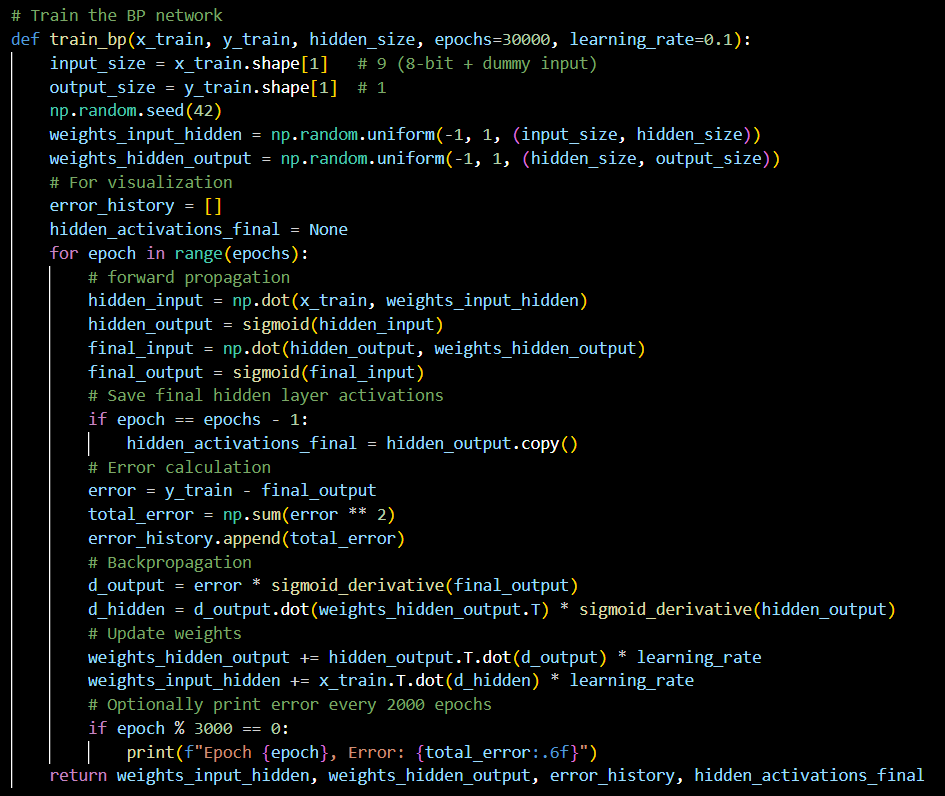
**Figure A10. [BP-8bit] Initialization and Data Generation**

This snippet includes the sigmoid activation function (and its derivative), and a function to generate all 256 possible 8-bit input patterns with an appended dummy input. Each pattern is labeled based on whether it has even or odd parity.



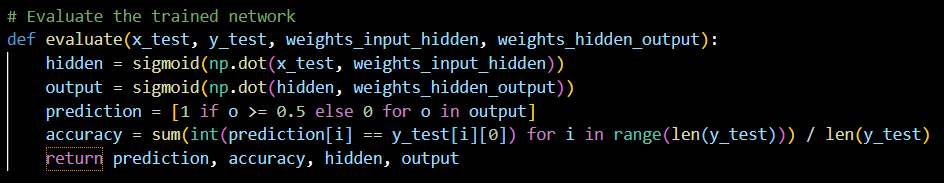
**Figure A11. [BP-8bit] Network Training Function**

This part defines the train\_bp() function, which performs forward propagation, backpropagation, and weight updates over 30,000 epochs. It also logs the total squared error for each epoch and optionally prints progress every 3,000 steps.



**Figure A12. [BP-8bit] Network Evaluation Function**

This function runs a forward pass using the trained weights and calculated prediction accuracy. It converts sigmoid outputs to binary class predictions (0 or 1).

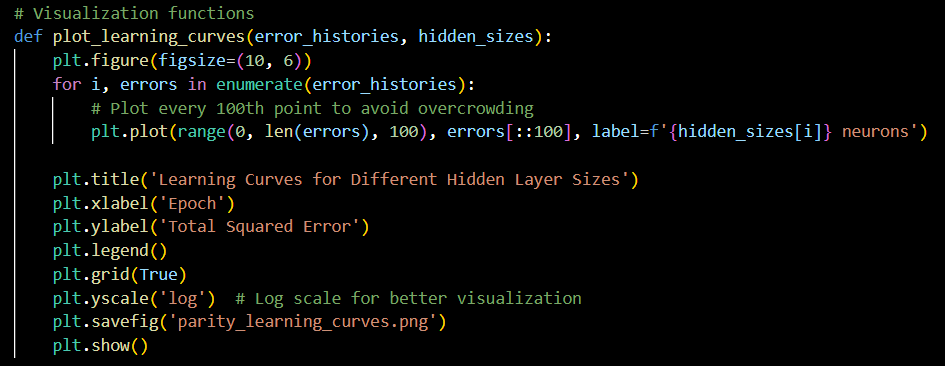


**Figure A13. [BP-8bit] Main Program and Training Loop**

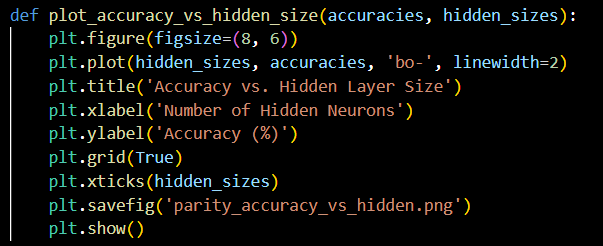
This section brings everything together: it prepares data, trains networks with varying hidden layer sizes, evaluates performance, and generates plots. The best-performing configuration is reported at the end.



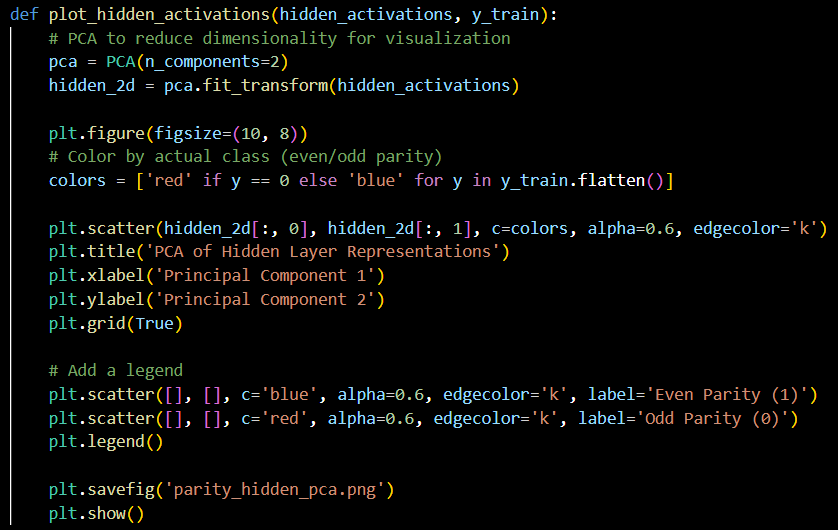
**Figure A14. [BP-8bit] Learning Curves Plot**



**Figure A15. [BP-8bit] Accuracy vs. Hidden Size Plot**



**Figure A16. [BP-8bit] Hidden Activations Plot**



**Figure A17. [BP-8bit] Bit Count vs. Output Plot**

