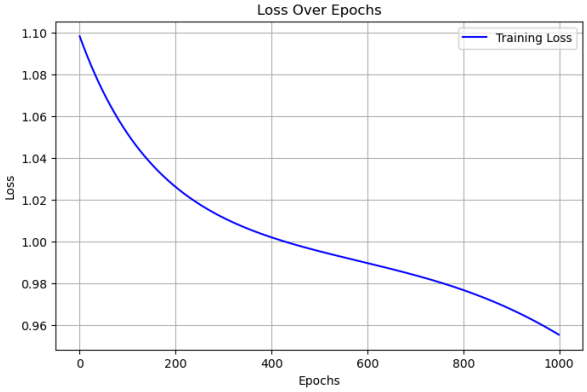
This report summarizes the training process of a neural network implemented from scratch using NumPy. The model was trained on the UCI Wine Quality dataset to classify wines based on their chemical attributes. The goal was to evaluate the model's learning process by tracking the loss over epochs and measuring the final test accuracy.

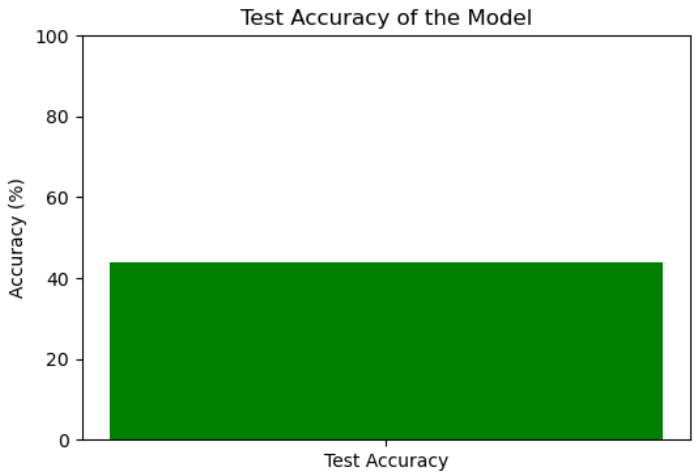
Neural networks learn by updating their parameters (weights and biases) through iterative optimization. This process involves forward propagation, where the weighted sum of inputs is computed and passed through an activation function to generate predictions. The loss calculation measures the discrepancy between the predictions and actual labels using a loss function. During backward propagation, gradients of the loss function with respect to the model parameters are computed using the chain rule. Finally, weights and biases are adjusted using an optimization algorithm such as gradient descent to minimize the loss. The model was trained using cross-entropy loss with a learning rate set to optimize convergence. The dataset was normalized to ensure efficient training.

The loss gradually decreased over time, indicating the network successfully learned patterns from the training data. The loss values at different epochs were as follows: Epoch 0, Loss: 1.0983; Epoch 100, Loss: 1.0516; Epoch 200, Loss: 1.0262; Epoch 300, Loss: 1.0114; Epoch 400, Loss: 1.0021; Epoch 500, Loss: 0.9954; Epoch 600, Loss: 0.9897; Epoch 700, Loss: 0.9839; Epoch 800, Loss: 0.9768; Epoch 900, Loss: 0.9676. The downward trend in the loss curve indicates successful learning and convergence of the model.

After training, the model was evaluated on the test set, achieving an accuracy of 44.06%. This result suggests that the model captures some structure in the data but has room for improvement. The neural network effectively reduced loss over time, demonstrating successful learning. However, the relatively low test accuracy suggests that further optimization, such as tuning hyperparameters, increasing model complexity, or employing a more advanced architecture, could enhance performance.

Future work could include experimenting with different activation functions, adding more layers, or using advanced optimization techniques to improve accuracy.





Neural networks are widely used for classification tasks, but they can also be leveraged for generating synthetic data. By adjusting the way a trained model is used, we can repurpose it for data generation, moving from a discriminative to a generative approach.

A standard feedforward neural network trained for classification can be adapted to generate new instances that resemble real data. One approach is to sample inputs from a distribution and feed them into the trained model to produce outputs that align with learned patterns. This is particularly useful in domains where new samples are needed for data augmentation, anomaly detection, or simulation. The trained network can serve as a mapping from an input space to an output space that resembles the original data distribution.

To transition from classification to generation, several modifications can be made. For instance, instead of directly predicting class labels, the network can be trained to output a probability distribution over possible features. Adding noise to inputs can enhance variability in generated samples. Another approach is to modify the final layer to generate continuous-valued outputs rather than discrete class labels, making the model more suited for synthesis tasks. By sampling different inputs, we can explore a range of outputs, effectively generating diverse synthetic data.

A classifier trained on the wine dataset could be repurposed for generative modeling by using the learned feature representations to create realistic synthetic wine quality profiles. By sampling new feature vectors and passing them through the trained network, we could generate plausible instances that mimic the original data distribution.

Beyond simple modifications, more advanced generative models extend this idea. Autoencoders, for example, compress input data into a lower-dimensional representation and then reconstruct it, allowing for controlled data generation by sampling from the latent space. Variational autoencoders (VAEs) further refine this process by enforcing a probabilistic structure on the latent space, enabling smoother interpolation between generated samples.

Generative Adversarial Networks (GANs) take a different approach by employing two neural networks—a generator and a discriminator—in a competitive setup. The generator creates synthetic samples, while the discriminator evaluates their authenticity. Through iterative training, the generator improves its ability to produce realistic data that aligns with the training

distribution. This technique has been widely used for generating images, text, and structured datasets.