Adaptive Deep Bayesian Neural Network Framework: Comprehensive Technical Documentation

Ninan sajeeth Philip
Artificial Intelligence Research and Intelligent Systems (airis4d.com)
Thelliyoor 689544

September 22, 2025

Abstract

This document provides comprehensive technical documentation for the Adaptive Deep Bayesian Neural Network (DBNN) framework. The system implements an innovative adaptive learning approach that combines Bayesian methodologies with margin-based sample selection to efficiently learn from limited data. The framework features GPU acceleration, automatic configuration management, and comprehensive statistical reporting capabilities. This documentation covers the system architecture, mathematical foundations, implementation details, and user interface specifications.

Contents

1 Introduction

The Adaptive DBNN framework represents a significant advancement in machine learning systems that can efficiently learn from limited data through intelligent sample selection. This system combines Bayesian neural networks with adaptive learning strategies to maximize learning efficiency while minimizing computational resources.

1.1 System Overview

The framework consists of two main components:

- **AdaptiveDBNN**: The main controller class that manages the adaptive learning process
- **GPUDBNN**: The core Bayesian neural network engine with GPU acceleration

1.2 Key Innovations

- Margin-based informative sample selection
- Automatic configuration persistence
- GPU-accelerated Bayesian computations
- Comprehensive statistical reporting
- Interactive user configuration

2 System Architecture

2.1 High-Level Architecture

2.2 AdaptiveDBNN Class Structure

3 Mathematical Foundations

3.1 Bayesian Framework

The system implements a Bayesian neural network framework with Gaussian likelihoods:

$$P(y|x, \theta) = \prod_{i=1}^{N} \mathcal{N}(y_i|f(x_i, \theta), \sigma^2)$$

Where feature combinations are modeled as multivariate Gaussians:

$$P(x|y = c) = \frac{1}{(2\pi)^{k/2}|\Sigma_c|^{1/2}} \exp\left(-\frac{1}{2}(x - \mu_c)^T \Sigma_c^{-1}(x - \mu_c)\right)$$

3.2 Margin-Based Sample Selection

The adaptive learning system employs a margin-based approach to select the most informative samples:

$$\text{margin} = P(y = \hat{y}|x) - P(y = y_{\text{true}}|x)$$

Where:

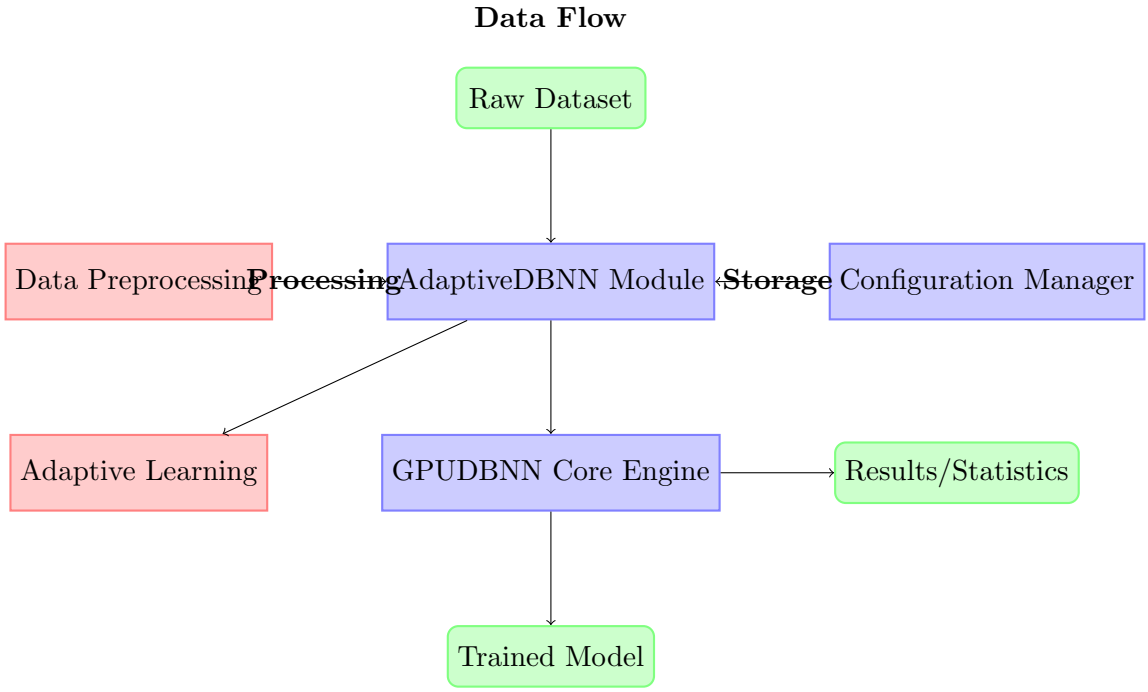

Figure 1: High-level system architecture showing main components and data flows

- $P(y = \hat{y}|x)$ is the posterior probability of the predicted class
- $P(y = y_{\text{true}}|x)$ is the posterior probability of the true class

4 Adaptive Learning Workflow

5 Configuration System

5.1 Configuration File Structure

The system uses JSON configuration files with the following structure:

```
{
    "file_path": "dataset.csv",
    "target_column": "target",
    "separator": ",",
    "has_header": true,
    "training_config": {
        "trials": 100,
        "cardinality_threshold": 0.9,
        "learning_rate": 0.1,
        "epochs": 1000,
        "test_fraction": 0.2
    },
    "likelihood_config": {
        "feature_group_size": 2,
        "max_combinations": 1000,
        "update_strategy": 3
    },
    "adaptive_learning": {
```

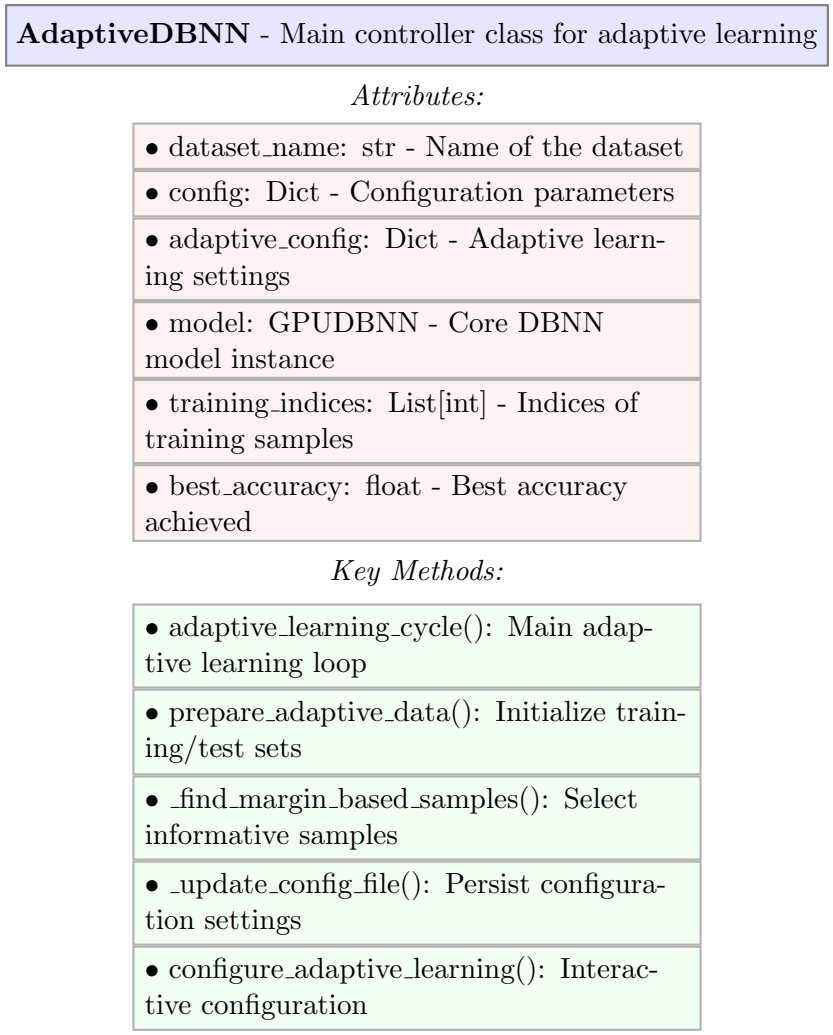

Figure 2: AdaptiveDBNN class structure with key attributes and methods

```
        "enable_adaptive": true,
        "initial_samples_per_class": 10,
        "margin": 0.1,
        "max_adaptive_rounds": 15
    },
    "statistics": {
        "enable_confusion_matrix": true,
        "enable_progress_plots": true,
        "save_plots": true
    }
}
```

5.2 Interactive Configuration

The system provides an interactive configuration interface:

```
def configure_adaptive_learning(self):
    """Interactively configure adaptive learning settings"""
    print("\nConfigure Adaptive Learning Settings")
    print("=" * 50)
```

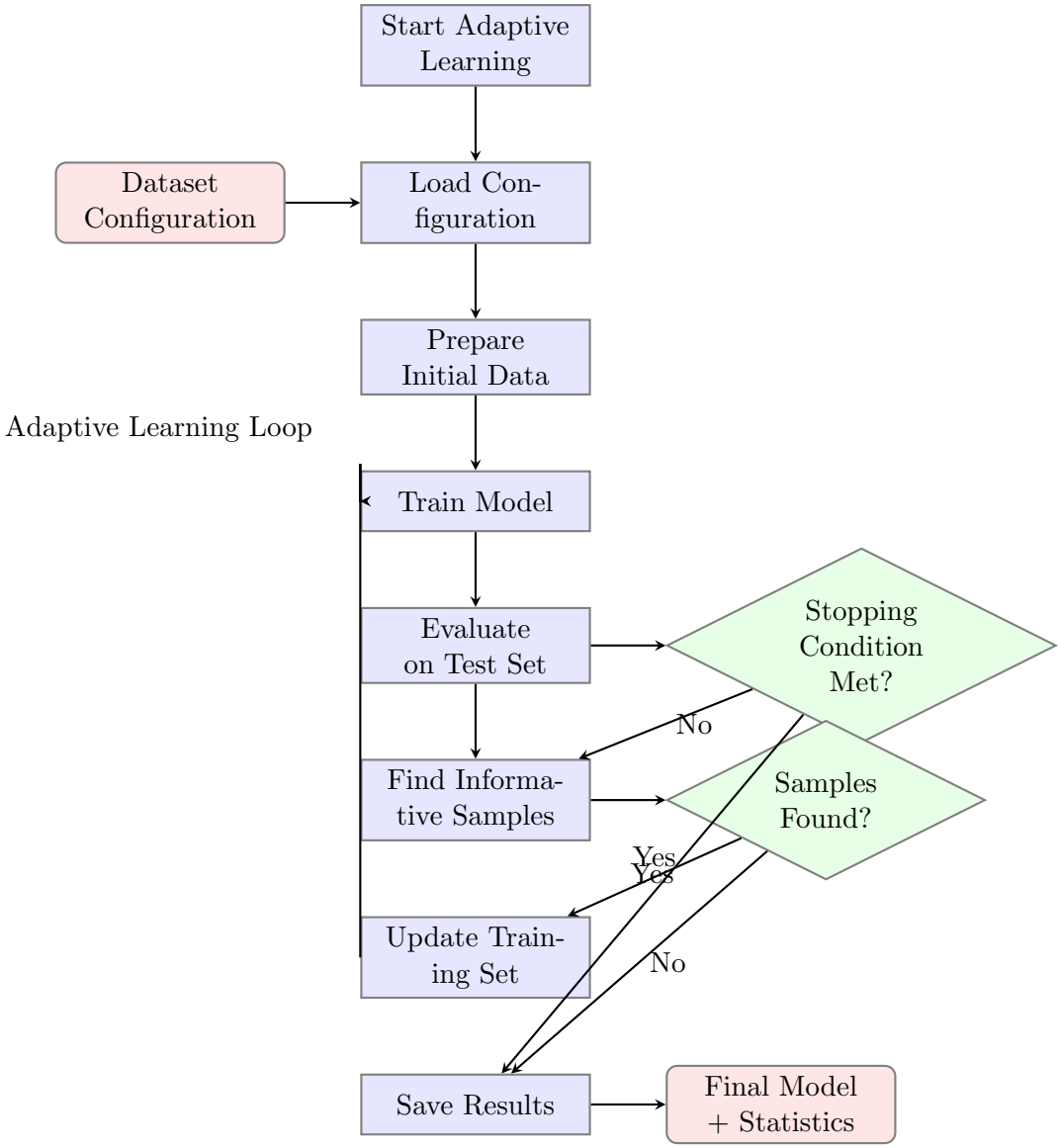

Figure 3: Complete adaptive learning workflow

```
# Get new values from user
initial_samples = int(input(f"Initial samples per class:") or 5)
margin = float(input(f"Margin threshold:") or 0.1)
max_rounds = int(input(f"Maximum adaptive rounds:") or 10)

# Update settings and config file
self.adaptive_config.update({
    'initial_samples_per_class': initial_samples,
    'margin': margin,
    'max_adaptive_rounds': max_rounds
})
self._update_config_file()
```

6 Algorithms and Implementation

6.1 Margin-Based Sample Selection Algorithm

Algorithm 1 Margin-Based Informative Sample Selection

procedure FINDINFORMATIVESAMPLES($X, y, predictions, posteriors$)
    margin ← adaptive_config[margin]
    samples_to_add ← []
    y_test ← y[test_indices]
    failed_mask ← predictions ≠ y_test
    failed_indices ← where(failed_mask)
    if len(failed_indices) = 0 **then**
        return GetDiverseFallbackSamples()
    end if
    for each idx in failed_indices **do**
        true_class ← y_test[idx]
        pred_class ← predictions[idx]
        margin_value ← posteriors[idx, pred_class] − posteriors[idx, true_class]
        Store margin information
    end for
    max_margin_sample ← argmax(margins)
    min_margin_sample ← argmin(margins)
    samples_to_add ← GetMarginBasedSamples(max_margin_sample, max)
    samples_to_add ← samples_to_add ∪ GetMarginBasedSamples(min_margin_sample, min)
    return samples_to_add
end procedure

7 Technical Specifications

7.1 System Requirements

- **Python**: 3.7 or higher
- **PyTorch**: GPU/CPU support
- **NumPy**: Numerical computations
- **Scikit-learn**: Machine learning utilities
- **Matplotlib/Seaborn**: Visualization
- **Memory**: 8GB minimum, 16GB recommended
- **Storage**: Sufficient for datasets and models

7.2 Performance Characteristics

- **Computational Complexity**: $O(n \times f \times c \times e)$
- **Memory Usage**: Optimized through pruning and efficient storage
- **GPU Acceleration**: Automatic CUDA detection and utilization
- **Scalability**: Handles large datasets through streaming

8 Usage Examples

8.1 Basic Usage

```
# Initialize adaptive learning model
adaptive_model = AdaptiveDBNN("my_dataset")

# Configure settings interactively
adaptive_model.configure_adaptive_learning()

# Prepare data and run adaptive learning
X_full, y_full = adaptive_model.prepare_full_data()
best_accuracy, best_indices = adaptive_model.adaptive_learning_cycle(X_full, y_f
```

8.2 Advanced Configuration

```
# Manual configuration
config = {
    "adaptive_learning": {
        "initial_samples_per_class": 20,
        "margin": 0.15,
        "max_adaptive_rounds": 20
    },
    "statistics": {
        "enable_confusion_matrix": True,
        "save_plots": True
    }
}

adaptive_model = AdaptiveDBNN("my_dataset", config=config)
```

9 Conclusion

The Adaptive Deep Bayesian Neural Network framework represents a significant advancement in machine learning systems that can efficiently learn from limited data. The combination of Bayesian methodologies, adaptive learning strategies, and practical implementation considerations makes this framework suitable for a wide range of applications including medical diagnostics, industrial quality control, financial fraud detection, and scientific research.

The system's key strengths include:

- Intelligent sample selection through margin-based criteria
- Comprehensive configuration management
- GPU-accelerated computations
- Detailed statistical reporting
- User-friendly interactive interface

This documentation provides a comprehensive overview of the system's capabilities, implementation details, and usage patterns. For further information, refer to the source code documentation and example implementations.