Brain-Inspired Neural Network Codebase Structure

This document provides an overview of the codebase structure for the Brain-Inspired Neural Network project.

Directory Structure

```
brain_inspired_nn/
├── config/
│   └── config.yaml          # Configuration file for the model and training
├── docs/                    # Documentation directory
├── src/                     # Source code directory
│   ├── controller/          # GRU-based controller implementation
│   │   ├── __init__.py
│   │   └── persistent_gru.py # Persistent GRU implementation
│   ├── neuromodulator/      # Neuromodulation system
│   │   ├── __init__.py
│   │   └── neuromodulator.py # Neuromodulator implementation
│   ├── utils/               # Utility functions and helper classes
│   │   ├── __init__.py
│   │   └── llm_interface.py  # Interface for LLM API integration
│   ├── __init__.py
│   ├── model.py             # Main model implementation
│   └── train.py             # Training script
├── tests/                   # Test cases
│   ├── __init__.py
│   └── test_model.py        # Test script for the model
├── CODEBASE_STRUCTURE.md    # This file
├── README.md                # Project overview and instructions
├── requirements.txt         # Project dependencies
└── run_tests.sh             # Script to run tests
```

Key Components

1. Persistent GRU Controller (`src/controller/persistent_gru.py`)

The Persistent GRU Controller is the central component of the system, responsible for processing input sequences and maintaining a persistent memory state. It consists of:

- **PersistentGRUCell**: A custom GRU cell with persistent memory capabilities
- **PersistentGRUController**: A multi-layer controller that uses PersistentGRUCell units

The persistent memory allows the controller to maintain long-term information across sequences, similar to how the brain maintains memories over time.

2. Neuromodulator System (`src/neuromodulator/neuromodulator.py`)

The Neuromodulator System modulates neural activity based on reward signals, inspired by how neurotransmitters function in the brain. It includes:

- **NeuromodulatorSystem**: Models the effects of different neurotransmitters (dopamine, serotonin, norepinephrine, acetylcholine)
- **RewardPredictor**: Predicts rewards based on the current state and action

The neuromodulator system allows the network to adapt its behavior based on reward signals, enabling reinforcement learning-like capabilities.

3. Main Model (`src/model.py`)

The `BrainInspiredNN` class integrates the Persistent GRU Controller and Neuromodulator System into a cohesive neural network. It:

- Processes input sequences through the controller
- Applies neuromodulation based on reward signals
- Generates outputs and predicted rewards

4. LLM Interface (`src/utils/llm_interface.py`)

The LLM Interface provides functionality for interacting with LLM API endpoints, allowing the system to leverage LLM capabilities for training and validation. It includes methods for:

- Sending prompts to an LLM API
- Processing responses
- Converting between LLM text and model tensor representations
- Evaluating model outputs using the LLM

5. Training Script (`src/train.py`)

The training script handles the training process, including:

- Loading configuration
- Setting up the model, optimizer, and scheduler
- Training and validation loops
- LLM integration for validation
- Checkpoint saving

Configuration (`config/config.yaml`)

The configuration file contains settings for:

- General parameters (seed, device, log directory, etc.)
- GRU Controller parameters (hidden size, number of layers, etc.)
- Neuromodulator parameters (scaling factors for different neurotransmitters)
- Training parameters (batch size, learning rate, etc.)
- LLM integration parameters (API endpoint, model name, etc.)

Tests (`tests/test_model.py`)

The test script verifies the functionality of the key components:

- PersistentGRUController
- NeuromodulatorSystem
- BrainInspiredNN

Dependencies (`requirements.txt`)

The project depends on several Python packages:

- PyTorch for neural network implementation
- NumPy for numerical operations
- Matplotlib for visualization
- Pandas for data handling
- PyYAML for configuration file parsing
- scikit-learn for machine learning utilities
- tqdm for progress bars

- pytest for testing
- OpenAI for LLM API integration