Brain-Inspired Neural Network Architecture

This document describes the architecture of the Brain-Inspired Neural Network system.

Overview

The Brain-Inspired Neural Network (BINN) is designed to mimic certain aspects of brain functionality, particularly:

1. **Persistent Memory**: The ability to maintain information over extended periods
2. **Neuromodulation**: The adjustment of neural activity based on reward signals
3. **Adaptive Learning**: Modification of learning rates and exploration based on context

Components

1. Persistent GRU Controller

The central component of the system is a GRU-based controller with persistent memory. This controller:

- Processes input sequences using GRU layers
- Maintains a persistent memory state across forward passes
- Blends current hidden states with persistent memory to enable long-term memory retention

```
Input → GRU Layers → Output Layer → Output
       ↑       ↓
       └─ Persistent Memory
```

2. Reward-Based Neuromodulator

The neuromodulation system adjusts various parameters based on reward signals:

- Computes modulator levels based on input and reward
- Applies different decay rates to each modulator
- Provides learning rate and exploration rate modifiers

```
Input + Reward → Modulator Network → Modulator Levels
                                          ↓
                        Learning Rate, Exploration Rate, etc.
```

3. Integration with LLM

The system can interface with an LLM API endpoint for:

- Training data generation
- Validation of outputs
- Knowledge integration

Data Flow

1. Input data is processed by the controller
2. Controller outputs predictions

3. Environment provides reward signals
4. Neuromodulator adjusts parameters based on reward
5. Learning process is modified by neuromodulator outputs

Implementation Details

Tensor Shape Management

The system includes utilities for handling tensor shape mismatches:

- Reshaping tensors when possible
- Padding tensors when reshaping is not possible
- Ensuring compatibility between components

Memory Optimization

Memory usage is optimized through:

- Clearing CUDA cache when appropriate
- Garbage collection
- Monitoring and reporting memory usage

Training Process

The training process involves:

1. Loading and preprocessing data
2. Initializing the model and its components
3. Training for multiple epochs with validation
4. Saving checkpoints for later use
5. Evaluating performance on test data

Testing

The system includes comprehensive tests for:

- Controller functionality
- Neuromodulator behavior
- Memory utilities
- Overall model performance