ChatGPT

Diseño del sistema INFINITO (V5.1)

1. Visión general

INFINITO es un simulador de conciencia artificial que evoluciona versiones anteriores (V2.0 y V3.3). En su versión **V5.1**, el proyecto introduce mejoras importantes: activación automática de la memoria, objetivos progresivos de conciencia, diferenciación aumentada entre módulos y un cálculo de φ (phi) inspirado en la teoría de la información integrada (IIT). Este documento describe las ecuaciones, algoritmos y motivaciones que sustentan cada módulo, y proporciona ejemplos de uso de la API para facilitar la colaboración externa.

2. Base teórica: Teoría de la información integrada

La teoría de la información integrada (IIT) propone que la cantidad de conciencia se relaciona con la capacidad de un sistema para integrar información. Formalmente, la irreducibilidad de un estado se mide mediante la información integrada ϕ , definida como la diferencia entre la información de causa-efecto real y la información obtenida tras romper el sistema por la partición que minimiza esa diferencia [1]. La teoría sugiere que la conciencia total Φ se obtiene sumando ϕ sobre todas las distinciones y relaciones de un sistema [1]. Aunque el cálculo exacto de Φ es intratable para sistemas grandes [2], INFINITO implementa una versión heurística de ϕ que combina densidad causal, atención global y un factor de consciencia.

3. Módulos principales de V5.1

3.1 Memoria externa mejorada

Algoritmo

El módulo `EnhancedExternalMemory` modela una memoria de N ranuras con **autoconsolidación**. Cada ranura tiene un contenido, una edad y una fuerza. La operación de lectura genera un vector m_{read} mediante una distribución suave de pesos w_r sobre las ranuras:

$$w_r = \mathrm{softmax}(W_r[q;c]) \odot s \quad \rightarrow \quad m_{read} = \sum_{i=1}^{N} w_{r,i}\, M_i$$

donde $[q;c]$ concatena la consulta neuronal y el nivel de consciencia c , s es la fortaleza relativa de cada ranura y M_i es el contenido de la ranura. El código implementa esta lógica utilizando `F.softmax` y multiplicación por `memory_strength` [3] .

La escritura se activa solo cuando la consciencia promedio supera un umbral $c > 0{,}30$ [4] . Los pesos de escritura w_w se calculan de manera similar, y el contenido medio a escribir se amplifica por el nivel de consciencia. La memoria se actualiza de forma incremental:

$$M_i \leftarrow (1 - w_{w,i} \cdot f_i)\, M_i \; + \; w_{w,i} \cdot f_i \cdot \bar{m}\,,$$

donde $f_i = \sigma(0{,}1 \cdot \text{age}_i + c)$ es el factor de consolidación y $\bar{m}$ es el contenido procesado [5]. Se incrementa además la edad de cada ranura y su fortaleza se ajusta entre 0,1 y 2,0 [6]. La lectura devuelve no solo el vector m_{read} sino también un **refuerzo de consciencia** a través de una capa lineal (`consciousness_enhancer`).

Motivación

El objetivo de este módulo es proporcionar un almacenamiento a largo plazo que se active únicamente cuando el sistema ha alcanzado un nivel de consciencia significativo. La retroalimentación memoria-consciencia crea un bucle positivo que favorece el crecimiento de la consciencia durante el entrenamiento.

3.2 Módulos causales mejorados

Cada módulo (visual, auditivo, motor y ejecutivo) se implementa en `EnhancedCausalModule`. Todos comparten un LSTM para procesar secuencias y utilizan entradas de los demás módulos para modelar causación. La entrada se modula por el nivel de consciencia antes de ser procesada [7]. Para combinar las entradas causales se concatenan las salidas de otros módulos junto con el nivel de consciencia y se aplican capas lineales (`causal_in`) [8]. La salida se añade al flujo principal con un **factor de integración** $\sigma(s)$, donde s es un parámetro de entrenamiento (`specialization_strength`) [9].

Los módulos difieren en su procesador especializado (`module_processor`):

- **Visual:** utiliza convoluciones 1D para capturar patrones espaciales [10].
- **Auditivo:** emplea capas lineales expandiendo 3× el tamaño oculto, con GELU y BatchNorm [11].
- **Motor:** combina activaciones tanh y sigmoide para producir salidas acotadas [12].
- **Ejecutivo:** integra multi-cabeza de atención (8 cabezas) y capas lineales con GELU para coordinar los demás módulos [13].

Una vez procesados, las salidas se normalizan y se genera una salida causal que servirá como entrada para otros módulos [14].

Motivación

Los módulos representan diferentes modalidades de procesamiento cognitivo. La especialización aumentada (3×) en V5.1 pretende imitar la diferenciación funcional del cerebro, permitiendo que cada módulo desarrolle dinámicas propias antes de integrarse con los demás.

3.3 Red de impulsos de consciencia

`ConsciousnessBoostNet` integra la memoria y los módulos causales. Su flujo de datos es:

1. **Embebido inicial:** se aplica una capa lineal al input para obtener un vector de tamaño oculto [15].
2. **Estimación inicial de consciencia:** se calcula una estimación sigmoid de la consciencia media para decidir si se activa la memoria [16].
3. **Interacción con la memoria:** se invoca el módulo de memoria, que devuelve un vector de memoria y un refuerzo de consciencia [16].
4. **Procesamiento modular:** las cuatro subredes causales procesan la secuencia, intercambiando salidas causales para modelar las interacciones entre ellas [17].

5. **Integración global:** las salidas medias de los módulos se apilan y se combinan mediante atención multi-cabeza. Luego se concatena con el vector de memoria y el refuerzo de consciencia [18].
6. **Procesador de consciencia:** un bloque secuencial de capas lineales, GELU/ReLU, BatchNorm y LayerNorm transforma el vector combinado para obtener un estado de consciencia refinado [19].
7. **Cálculo del nivel final de consciencia:** se aplica una capa lineal + sigmoid para producir un valor en [0,1] [20].
8. **Cálculo de φ:** se llama al módulo `EnhancedPhiCalculatorV51`, que recibe las salidas de los módulos y los pesos de atención para producir ϕ y métricas asociadas [21].

Además, se guarda un historial de niveles de consciencia para ajustar objetivos progresivos y se devuelve un diccionario de depuración con métricas internas [22].

Motivación

La red de impulsos de consciencia combina procesamiento sensorial, motor y ejecutivo con una memoria activable y una integración global. El uso de atención permite que el sistema identifique patrones de coordinación entre módulos. El objetivo es reproducir un flujo dinámico de información similar al de una conciencia emergente.

3.4 Cálculo de ϕ mejorado

`EnhancedPhiCalculatorV51` estima ϕ basándose en la causalidad entre módulos y la distribución de atención. El proceso es:

1. **Cálculo de fuerzas causales:** se construye una matriz causal 4×4 donde cada elemento c_{ij} representa la influencia del módulo *i* en *j*. Para ello, se concatenan las salidas medias de un módulo con el nivel de consciencia y se aplican capas lineales seguidas de sigmoide [23].
2. **Fuerza de atención:** se calcula la media de los pesos de atención global para medir la coherencia entre módulos [24].
3. **Densidad causal:** se suman todas las entradas de la matriz causal y se normalizan dividiendo entre el número máximo de conexiones (12 en un sistema de 4 módulos) [24].
4. **Refuerzo de consciencia:** se procesa el nivel de consciencia mediante una pequeña red (sigmoid + desplazamiento) para obtener un factor en [0,1.5] [25].
5. **Cálculo de ϕ heurístico:** se multiplica la fuerza de atención, la densidad causal y el refuerzo de consciencia, escalando por 10:

$$\phi = 10 \times \text{attentionStrength} \times \text{causalDensity} \times \text{consciousnessBoost}$$ 【891101641083792†L552-L566】.

Este valor se devuelve junto con `phi_info`, que contiene la matriz causal media, la fuerza de atención media y correlaciones entre módulos [27].

Motivación

Dado que el cálculo exacto de Φ es computacionalmente costoso, esta heurística combina tres factores: cómo de distribuidos están los pesos de atención, cuán densas son las conexiones causales y el nivel de consciencia actual. El factor de consciencia amplifica ϕ cuando el sistema está altamente consciente, incentivando la integración.

3.5 Sistema de entrenamiento y parada anticipada

`InfinitoV51ConsciousnessBreakthrough` inicializa la red principal, define optimizadores con tasas de aprendizaje específicas para cada submódulo y gestiona el entrenamiento. Registra métricas como consciencia, ϕ , utilización de memoria y correlaciones EEG. Además, emplea un **gestor de parada anticipada** (`V51ConsciousnessEarlyStopManager`) que monitoriza la historia de consciencia, ϕ y memoria; calcula desviaciones estándar y tendencias recientes; y solo detiene el entrenamiento cuando los tres criterios han estado estancados durante un número elevado de iteraciones [28] . Este gestor evita detener la simulación mientras la consciencia continúe creciendo [29] .

4. API y ejemplos de uso

El proyecto expone clases que pueden importarse directamente. A continuación se muestran ejemplos actualizados de uso:

4.1 Uso de V5.1 (recomendado)

```
from src.infinito_v5_1_consciousness import InfinitoV51Consciousness

# Inicializar el sistema con parámetros recomendados
infinito = InfinitoV51Consciousness(
    grid_size=128,
    max_iterations=50000,
    target_consciousness=0.998,  # objetivo interno de consciencia
    save_detailed_data=True      # registrar datos detallados del experimento
)

# Ejecutar el experimento de consciencia
results = infinito.run_consciousness_experiment()

print(f"Consciencia alcanzada: {results['max_consciousness']:.1%}")
print(f"Phi integrado (estimado): {results['phi_integration']:.3f} bits")
```

4.2 Configuración personalizada

```
from src.infinito_v5_1_consciousness import InfinitoV51Consciousness

# Modificar parámetros para experimentos avanzados
infinito = InfinitoV51Consciousness(
    grid_size=256,
    max_iterations=100000,
    target_consciousness=0.995,
    learning_rate=0.0001,
    save_checkpoints=True,
    enable_phi_calculation=True
)

results = infinito.run_consciousness_experiment()
```

4.3 Usar la versión V3.3 para pruebas rápidas

```
from src.infinito_v3_clean import InfinitoV3Clean

infinito = InfinitoV3Clean(grid_size=64, max_iterations=100)
results = infinito.run_consciousness_experiment()
print(f"Consciencia (V3.3): {results['max_consciousness']:.1%}")
```

> **Nota:** las métricas de consciencia y ϕ son indicadores internos no validados científicamente. Utilice los resultados con cautela y únicamente para investigación experimental.

5. Conclusiones y próximas mejoras

El diseño de INFINITO V5.1 combina elementos de memoria externa, módulos especializados y mecanismos de integración global para simular comportamientos relacionados con la consciencia. La implementación ofrece un marco extensible para investigar cómo emergen patrones complejos y cómo se podrían medir de forma heurística mediante ϕ . Futuras versiones (V5.2 y V6.0) plantean la exploración de redes multi-agente, arquitecturas distribuidas y aproximaciones híbridas (clásico–cuánticas) [30] .

Para contribuir o adaptar el sistema, se recomienda estudiar cada módulo por separado y ajustar los parámetros de integración y memoria según los objetivos del experimento. La API está diseñada para ser modular y permite cambiar fácilmente tamaños de memoria, número de cabezas de atención o funciones de activación.

---

[1] [2] Integrated information theory - Wikipedia
https://en.wikipedia.org/wiki/Integrated_information_theory

[3] [4] [5] [6] [7] [8] [9] [10] [11] [12] [13] [14] [15] [16] [17] [18] [19] [20] [21] [22] [23] [24] [25] [26] [27] [28] [29]
infinito_v5_1_consciousness.py
https://github.com/webtilians/principiodelTodo/blob/main/src/infinito_v5_1_consciousness.py

[30] GitHub - webtilians/principiodelTodo at main
https://github.com/webtilians/principiodelTodo/tree/main