Mercury v3+ Hyper - Manual de Usuario

📋 Tabla de Contenidos

🚀 Introducción

¿Qué es Mercury v3+ Hyper?

Mercury v3+ Hyper es una red neuronal inteligente que puede aprender a separar patrones complejos en datos tridimensionales. Imagina que tienes dos tipos de objetos mezclados de forma muy complicada en el espacio 3D, esta red puede aprender a distinguir entre ellos automáticamente.

¿Para qué sirve?

- **Clasificación automática**: Separa datos en dos categorías
- **Detección de patrones**: Encuentra estructuras ocultas en datos complejos
- **Análisis predictivo**: Predice a qué categoría pertenece un nuevo dato

¿Quién puede usarlo?

- Estudiantes e investigadores
- Profesionales en ciencia de datos
- Desarrolladores de machine learning
- Cualquier persona interesada en inteligencia artificial

🛠️ Instalación y Configuración

Paso 1: Verificar Python

Asegúrate de tener Python 3.7 o superior instalado:

```bash
python --version
```

Paso 2: Instalar Dependencias

Ejecuta el siguiente comando en tu terminal:

```bash
pip install numpy matplotlib scikit-learn scipy torch
```

Paso 3: Descargar el Código

1. Descarga el archivo `Mercury_v3+_Hyper.py`
2. Guárdalo en una carpeta de tu proyecto
3. Abre tu editor de código favorito

Paso 4: Verificar Instalación

Ejecuta este código de prueba:

```python
import numpy as np
import matplotlib.pyplot as plt
print("✅ ¡Instalación exitosa!")
```

🏃 Primeros Pasos

Ejecución Básica

1. **Abrir el archivo**: Carga `Mercury_v3+_Hyper.py` en tu editor
2. **Ejecutar completo**: Presiona F5 o ejecuta todo el script
3. **Observar resultados**: El programa mostrará gráficos y resultados automáticamente

Lo que verás al ejecutar:

1. **Gráfico 3D**: Visualización de los datos de entrenamiento

2. **Proceso de entrenamiento**: Mensajes de progreso en la consola
3. **Gráfico de MSE**: Evolución del error durante el entrenamiento
4. **Resultados de optimización**: Mejores parámetros encontrados
5. **Métricas finales**: Precisión, matriz de confusión, etc.
6. **Visualizaciones**: Curvas de aprendizaje y predicciones

📖 Guía Paso a Paso

Paso 1: Generar Datos de Prueba

python

```python
# Importar librerías necesarias
import numpy as np
import matplotlib.pyplot as plt
from Mercury_v3_Hyper import generate_3d_spiral_data

# Generar datos sintéticos
X, Y = generate_3d_spiral_data(
    n_points=1000,    # Número de puntos
    noise=0.2,        # Cantidad de ruido (0.0-1.0)
    rotations=3       # Número de vueltas de la espiral
)

print(f"Datos generados: {X.shape[1]} puntos en 3D")
```

Paso 2: Visualizar los Datos

python

```python
# Mostrar los datos en 3D
plot_3d_spiral_data(X, Y)
```

¿Qué verás?

- Puntos azules y rojos formando espirales entrelazadas
- Una estructura compleja que sería difícil de separar manualmente

Paso 3: Entrenar la Red Neuronal

```python
# Entrenar con configuración básica
W1, b1, W2, b2, W3, b3, costs, accuracies = train_deep(
    X, Y,              # Datos de entrada
    n_hidden1=16,      # Neuronas en primera capa
    n_hidden2=512,     # Neuronas en segunda capa
    learning_rate=0.01, # Velocidad de aprendizaje
    epochs=3000,       # Número de iteraciones
    verbose=True       # Mostrar progreso
)
```

¿Qué verás?

- Mensajes cada 1000 iteraciones mostrando error y precisión
- Gráfico del error cuadrático medio (MSE) al final

Paso 4: Hacer Predicciones

```python
# Predecir sobre los datos de entrenamiento
predictions = predict_deep(X, W1, b1, W2, b2, W3, b3)

# Calcular precisión
accuracy = np.mean(predictions == Y) * 100
print(f"Precisión final: {accuracy:.2f}%")
```

Paso 5: Visualizar Resultados

```python
# Ver predicciones vs realidad
plot_3d_predictions(X, Y, predictions)
```

¿Qué verás?

- Puntos correctamente clasificados en colores
- Puntos incorrectos marcados con 'X' negras

💡 Ejemplos Prácticos

Ejemplo 1: Datos con Poco Ruido

python

```python
# Datos más "limpios" - más fáciles de clasificar
X_clean, Y_clean = generate_3d_spiral_data(
    n_points=800,
    noise=0.1,        # Poco ruido
    rotations=2
)

# Entrenar con menos épocas
W1, b1, W2, b2, W3, b3, _, _ = train_deep(
    X_clean, Y_clean,
    n_hidden1=16,
    n_hidden2=256,    # Menos neuronas
    learning_rate=0.01,
    epochs=2000,      # Menos épocas
    verbose=True
)
```

Ejemplo 2: Datos más Complejos

python

```python
# Datos más difíciles de clasificar
X_complex, Y_complex = generate_3d_spiral_data(
    n_points=1500,
    noise=0.4,        # Más ruido
    rotations=4       # Más vueltas
)

# Entrenar con más potencia
W1, b1, W2, b2, W3, b3, _, _ = train_deep(
    X_complex, Y_complex,
    n_hidden1=32,     # Más neuronas
    n_hidden2=1024,   # Más neuronas
    learning_rate=0.005,  # Aprendizaje más lento
    epochs=5000,      # Más épocas
    verbose=True
)
```

Ejemplo 3: Optimización Automática

python

```python
# Buscar automáticamente los mejores parámetros
best_params, results = optimize_deep_hyperparameters(
    X, Y,
    hidden1_neurons=[16, 32],     # Opciones para capa 1
    hidden2_neurons=[256, 512],   # Opciones para capa 2
    learning_rates=[0.01, 0.05],  # Opciones de velocidad
    epochs_list=[2000, 3000],     # Opciones de épocas
    decay_rate=0.01
)

print("Mejores parámetros encontrados:")
print(best_params)
```

🎛️ Personalización Avanzada

Modificar Arquitectura de la Red

python

```python
# Red más pequeña y rápida
def train_small_network():
    return train_deep(
        X, Y,
        n_hidden1=8,      # Menos neuronas
        n_hidden2=64,     # Menos neuronas
        learning_rate=0.05,
        epochs=1500,
        verbose=True
    )

# Red más grande y potente
def train_large_network():
    return train_deep(
        X, Y,
        n_hidden1=64,     # Más neuronas
        n_hidden2=2048,   # Más neuronas
        learning_rate=0.001,  # Más lento
        epochs=8000,
        verbose=True
    )
```

Configurar Visualizaciones

python

```python
# Personalizar gráficos
plt.figure(figsize=(15, 10))  # Tamaño del gráfico
plt.style.use('seaborn-v0_8')       # Estilo elegante

# Cambiar colores
def plot_custom_3d_data(X, Y):
    fig = plt.figure(figsize=(12, 10))
    ax = fig.add_subplot(111, projection='3d')

    # Colores personalizados
    colors = ['green', 'orange']  # En lugar de azul y rojo

    ax.scatter(X[0, :], X[1, :], X[2, :],
               c=[colors[int(y)] for y in Y[0, :]],
               s=50, alpha=0.7)

    ax.set_title('Mis Datos Personalizados')
    plt.show()
```

Guardando y Cargando Modelos

python

```python
# Guardar parámetros entrenados
def save_model(W1, b1, W2, b2, W3, b3, filename='modelo_entrenado.npz'):
    np.savez(filename, W1=W1, b1=b1, W2=W2, b2=b2, W3=W3, b3=b3)
    print(f"Modelo guardado como {filename}")

# Cargar parámetros
def load_model(filename='modelo_entrenado.npz'):
    data = np.load(filename)
    return data['W1'], data['b1'], data['W2'], data['b2'], data['W3'], data['b3']

# Uso
save_model(W1, b1, W2, b2, W3, b3)
W1_loaded, b1_loaded, W2_loaded, b2_loaded, W3_loaded, b3_loaded = load_model()
```

🔧 Solución de Problemas

Problema 1: Error de Importación

Síntoma: `ModuleNotFoundError: No module named 'numpy'`

Solución:

```bash
pip install --upgrade numpy matplotlib scikit-learn scipy torch
```

Problema 2: Precisión Muy Baja

Síntoma: La precisión se queda en ~50%

Soluciones:

1. **Aumentar épocas**: Cambia `epochs=3000` por `epochs=5000`
2. **Reducir learning rate**: Cambia `learning_rate=0.01` por `learning_rate=0.005`
3. **Más neuronas**: Aumenta `n_hidden2` de 512 a 1024

Problema 3: Entrenamiento Muy Lento

Síntoma: El entrenamiento tarda mucho tiempo

Soluciones:

1. **Reducir datos**: Usa `n_points=500` en lugar de 1000
2. **Menos épocas**: Cambia a `epochs=1500`
3. **Red más pequeña**: Reduce `n_hidden2` a 256

Problema 4: Gráficos No Aparecen

Síntoma: No se muestran las visualizaciones

Soluciones:

```python
# Forzar mostrar gráficos
import matplotlib.pyplot as plt
plt.ion()  # Modo interactivo
plt.show()
```

Problema 5: Overfitting

Síntoma: Alta precisión en entrenamiento, baja en validación

Soluciones:

1. **Más regularización**: Aumenta el dropout
2. **Menos épocas**: Detén el entrenamiento antes
3. **Más datos**: Aumenta `n_points`

❓ Preguntas Frecuentes

¿Cuánto tiempo tarda en entrenar?

Respuesta: Típicamente 2-5 minutos en una computadora moderna con la configuración por defecto.

¿Puedo usar mis propios datos?

Respuesta: Sí, pero deben estar en formato 3D. Tus datos deben tener la forma:

- `X`: Matriz de 3 filas (coordenadas x, y, z) y N columnas (muestras)
- `Y`: Vector de 1 fila con valores 0 o 1

¿Qué precisión debería esperar?

Respuesta: Con los datos sintéticos por defecto, deberías obtener 85-95% de precisión.

¿Funciona con datos reales?

Respuesta: Sí, pero primero debes:

1. Normalizar tus datos
2. Convertirlos al formato correcto
3. Ajustar los hiperparámetros

¿Puedo hacer clasificación multiclase?

Respuesta: No, esta versión solo hace clasificación binaria (2 clases).

🎯 Mejores Prácticas

Antes de Entrenar

1. **Visualiza tus datos**: Siempre mira los datos antes de entrenar
2. **Normaliza**: Asegúrate de que los datos estén en rangos similares

3. **División de datos**: Separa entrenamiento, validación y prueba

Durante el Entrenamiento

1. **Monitorea el progreso**: Usa `verbose=True`
2. **Guarda checkpoints**: Guarda el modelo cada cierto tiempo
3. **Ajusta sobre la marcha**: Si no mejora, detén y ajusta parámetros

Después del Entrenamiento

1. **Evalúa con datos nuevos**: No uses los mismos datos del entrenamiento
2. **Analiza errores**: Mira qué casos clasifica mal
3. **Documenta**: Guarda los parámetros que funcionaron bien

Configuraciones Recomendadas

Para datos fáciles:

```
python

n_hidden1=16, n_hidden2=256, learning_rate=0.01, epochs=2000
```

Para datos difíciles:

```
python

n_hidden1=32, n_hidden2=1024, learning_rate=0.005, epochs=5000
```

Para pruebas rápidas:

```
python

n_hidden1=8, n_hidden2=64, learning_rate=0.05, epochs=1000
```

📊 Interpretando los Resultados

Gráfico de MSE (Error Cuadrático Medio)

- **Línea descendente**: El modelo está aprendiendo ✅
- **Línea plana**: El modelo dejó de mejorar ⚠️
- **Línea ascendente**: Posible overfitting ❌

Matriz de Confusión

```
          Predicho
Real    0    1
  0    TN   FP
  1    FN   TP
```

- **TN (True Negative)**: Correctamente clasificado como clase 0
- **TP (True Positive)**: Correctamente clasificado como clase 1
- **FP (False Positive)**: Incorrectamente clasificado como clase 1
- **FN (False Negative)**: Incorrectamente clasificado como clase 0

Métricas Clave

- **Accuracy**: % de predicciones correctas
- **Precision**: De los que predije como positivos, cuántos eran realmente positivos
- **Recall**: De todos los positivos reales, cuántos identifiqué correctamente
- **F1-Score**: Promedio armónico de precision y recall

🔄 Flujo de Trabajo Típico

1. Preparación (5 minutos)

python

```python
# Generar o cargar datos
X, Y = generate_3d_spiral_data(n_points=1000, noise=0.2, rotations=3)

# Visualizar
plot_3d_spiral_data(X, Y)
```

2. Entrenamiento Inicial (10 minutos)

python

```python
# Entrenar con configuración básica
W1, b1, W2, b2, W3, b3, costs, accuracies = train_deep(
    X, Y, n_hidden1=16, n_hidden2=512,
    learning_rate=0.01, epochs=3000, verbose=True
)
```

3. Evaluación (2 minutos)

python

```
# Predecir y evaluar
predictions = predict_deep(X, W1, b1, W2, b2, W3, b3)
accuracy = np.mean(predictions == Y) * 100
print(f"Precisión: {accuracy:.2f}%")
```

4. Optimización (15 minutos)

python

```
# Si la precisión es baja, optimizar
best_params, results = optimize_deep_hyperparameters(
    X, Y, [16, 32], [512, 1024], [0.01, 0.005], [3000, 5000], 0.01
)
```

5. Análisis Final (5 minutos)

python

```
# Visualizar resultados
plot_3d_predictions(X, Y, predictions)
evaluate_model(Y, predictions)
```

🎓 Glosario

Red Neuronal: Sistema de aprendizaje automático inspirado en el cerebro humano.

Época (Epoch): Una pasada completa por todos los datos de entrenamiento.

Learning Rate: Velocidad a la que el modelo aprende de los errores.

Dropout: Técnica para evitar que el modelo memorice los datos de entrenamiento.

Overfitting: Cuando el modelo funciona bien en entrenamiento pero mal en datos nuevos.

Regularización: Técnicas para hacer el modelo más generalizable.

Función de Activación: Función matemática que decide si una neurona debe activarse.

Backpropagation: Algoritmo para entrenar redes neuronales calculando gradientes.

Hiperparámetros: Configuraciones que se ajustan antes del entrenamiento.

Matriz de Confusión: Tabla que muestra las predicciones correctas e incorrectas.

🆘 Soporte y Ayuda

Si tienes problemas:

1. **Revisa la sección de Solución de Problemas**
2. **Verifica que todas las dependencias estén instaladas**
3. **Prueba con los ejemplos básicos primero**
4. **Consulta los mensajes de error en detalle**

Para aprender más:

- Experimenta con diferentes parámetros
- Prueba con tus propios datos
- Estudia el código fuente
- Investiga sobre redes neuronales

Recursos adicionales:

- Documentación técnica del proyecto
- Tutoriales de machine learning
- Comunidades de Python y AI

¡Feliz aprendizaje con Mercury v3+ Hyper! 🚀