Ejemplo: NN_ex1.m

Author: D.Sc. Aboud Barsekh Onji
Institution: Universidad Anáhuac México - Facultad de Ingeniería
Contact: aboud.barsekh@anahuac.mx
ORCID: 0009-0004-5440-8092

A continuación, se describen los comandos clave utilizados en el flujo de trabajo moderno de Deep Learning en MATLAB (`trainnet`), explicando las opciones abordadas en el código y alternativas disponibles, con un enfoque en aplicaciones de series temporales.

1. Particionamiento de Datos

Función: `trainingPartitions(numObservations, splits)`

Genera índices aleatorios para dividir el conjunto de datos. - **En el código:** `[0.8 0.1 0.1]` divide en 80% Entrenamiento, 10% Validación y 10% Prueba. - **Otras opciones:** - `[0.8 0.2]`: Solo partición de Entrenamiento y Prueba. - **Alternativa Clásica (`cvpartition`):** Del *Statistics and Machine Learning Toolbox* para realizar validación cruzada (*K-Fold*), útil cuando hay pocos datos. - **Alternativa Manual (`randperm`):** Permite control total si deseas mezclar y seleccionar índices manualmente.

---

2. Configuración del Entrenamiento

Función: `trainingOptions("solver", ...)`

Define la estrategia de aprendizaje de la red neuronal.

A. Solvers (Optimizadores)

- **`adam` (Usado):** Adaptive Moment Estimation. Generalmente el mejor rendimiento por defecto para series temporales y datos con ruido.
- **`sgdm`:** Descenso de gradiente estocástico con momento. Más clásico; a veces logra generalizar mejor pero requiere un ajuste fino de hiperparámetros más riguroso.
- **`rmsprop`:** Similar a Adam, históricamente útil en redes recurrentes (RNNs).

B. Opciones Críticas en Series Temporales

- **`GradientThreshold`:** Se estableció en `1`.
  - **Por qué:** Vital en LSTM/RNN. Evita el problema de *Exploding Gradients* (gradientes que crecen exponencialmente), recortando su norma a 1.
  - **Opciones relacionadas:** `GradientThresholdMethod` (puede ser `'l2norm'`, `'global-l2norm'`, o `'absolute-value'`).

- **`Shuffle`:** Se estableció en `"every-epoch"`.
  - **Por qué:** Mezcla los datos en cada época para evitar que la red memorice el orden de la secuencia y entre en ciclos.
  - **Opciones:** `"once"` (solo al inicio) o `"never"`.

C. Métricas y Visualización

- **`Metrics`:** Se usa `"accuracy"` para clasificación.
  - **Opciones:** `"rmse"` (Root Mean Squared Error) para problemas de regresión.
- **`Plots`:** `"training-progress"` muestra la gráfica en tiempo real. Es útil para detener el entrenamiento manualmente si se observa divergencia.

D. Opciones Adicionales (No usadas pero útiles)

- **`LearnRateSchedule`:** Opción `"piecewise"`. Reduce la tasa de aprendizaje cada cierto número de épocas para refinar la convergencia final.
- **`MiniBatchSize`:** Define el tamaño del lote. Por defecto suele ser 128.
  - *Menor tamaño:* Menos consumo de VRAM, gradiente más ruidoso (puede ayudar a escapar de mínimos locales).
  - *Mayor tamaño:* Entrenamiento más rápido (aprovecha GPU), gradiente más estable, mayor consumo de memoria.
- **`ExecutionEnvironment`:** `"auto"`, `"cpu"`, `"gpu"`, `"multi-gpu"`. Permite forzar el hardware.

---

3. Entrenamiento de la Red

Función: `trainnet(X, T, net, loss, options)`

Comando moderno (R2023a+) que sustituye el flujo rígido de `trainNetwork`.

- **Loss Function (`crossentropy`):**
  - **`crossentropy`:** Estándar para clasificación multiclase.
  - **`binarycrossentropy`:** Si solo hubiera dos clases.
  - **`mse` (Mean Squared Error):** Si el problema fuera predicción de series temporales (regresión).
  - **Function Handle:** Puedes pasar una función personalizada `@miFuncionDePerdida` (útil para estrategias híbridas).

---

4. Evaluación del Modelo

Funciones: `minibatchpredict` y `scores2label`

- **`minibatchpredict`:**

  - Realiza inferencia por lotes, optimizando el uso de memoria en GPU/CPU para grandes volúmenes de datos.
  - Devuelve *scores* (probabilidades brutas o logits).
- `scores2label`:
  - Toma los scores y busca el índice del valor máximo para asignar la etiqueta categórica correspondiente.
  - Si el problema fuera de regresión, este paso se omitiría.

---

5. Perspectiva de Inteligencia Computacional (Sugerencias avanzadas)

Dado tu enfoque en **Algoritmos Evolutivos (PSO, MOPSO, GA)**:

1. **Optimización de Hiperparámetros:** No dependerías solo de la intuición para elegir `InitialLearnRate` o `MaxEpochs`. Podrías usar un **Algoritmo Genético o PSO** en MATLAB donde cada "partícula" es una configuración de `options`, y la función de fitness es la *Accuracy* resultante en el conjunto de Validación.
2. **Entrenamiento Híbrido:** En lugar de usar `trainnet` con Adam, se puede escribir un bucle de entrenamiento personalizado (`dlnetwork` + `dlfeval`) donde un algoritmo **PSO actualice los pesos de la red neuronal** directamente, evitando el cálculo de gradientes (útil en funciones de coste no derivables).