

TAREA #1:

**Hankel-SVD-Entropy-Fourier
+
SNN-SGDM**

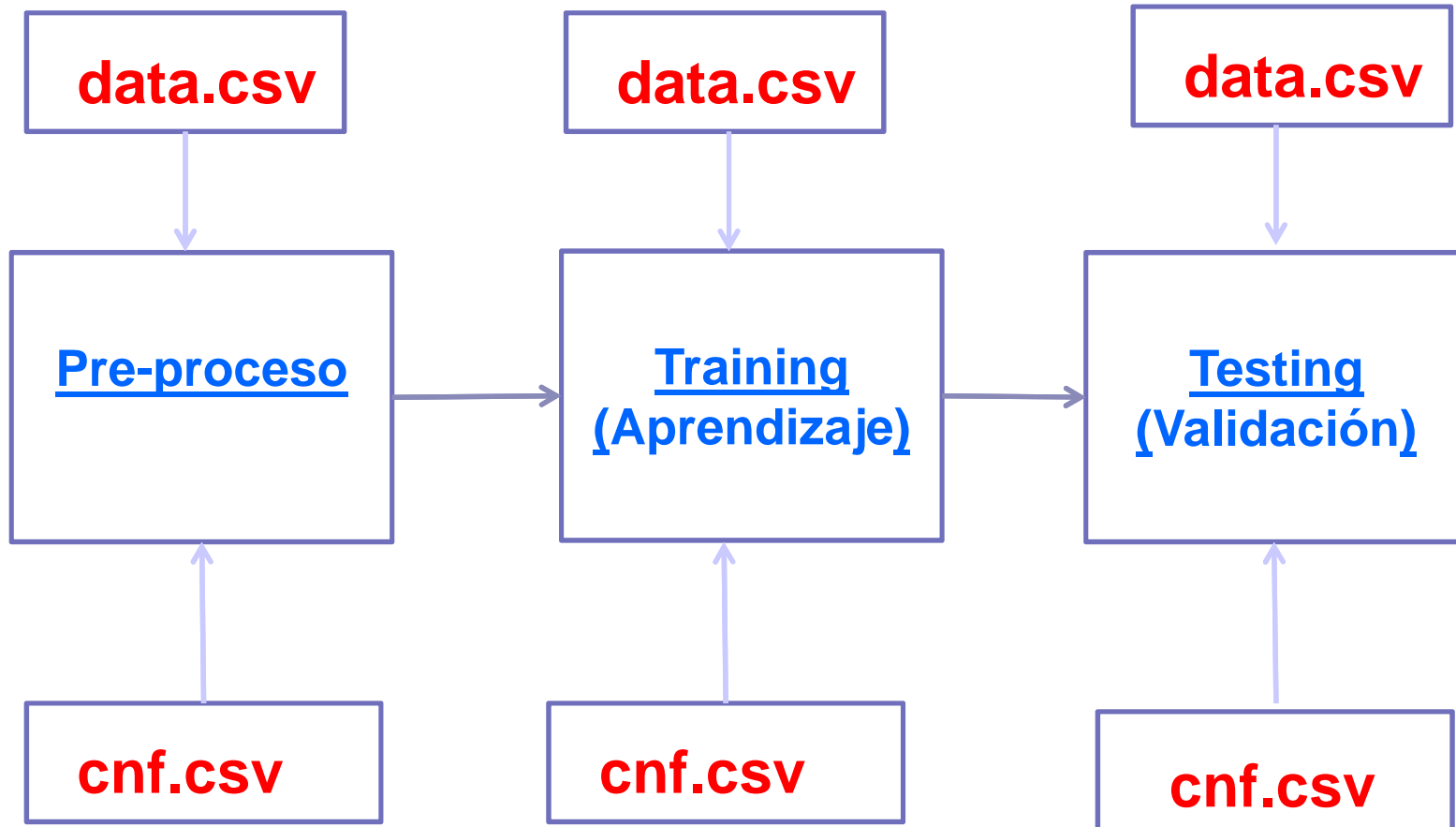
Prof. NIBALDO RODRÍGUEZ A.



OBJETIVO

- Implementar y evaluar el rendimiento de un modelo de red neuronal superficial para clasificación de datos de IoT usando matriz Hankel-SVD y Entropía Spectral de Fourier .

Etapas del Modelo:





Pre-proceso: prep.py

- Formato: Data.csv :

- ☐ M-filas.
- ☐ d-columns.

- Donde:

- ☐ **filas** : números de muestras.
- ☐ **columns**: número de variables.



Pre-proceso: `prep.py`

- **Función `Load_data()`:**
- Para cada Clase, Haga:
 - Crear una matriz de P-Filas por d-Columnas
 - P: representa el número de muestras
 - d: representa el número de variables.



Pre-proceso: Create_Features()

- For i=1 to NbrClass
 - For j=1 to NbrVariables
 - $X = \text{data_class}(\text{Dat}, j, i)$ # Retorna j-th variable de i-th class
 - $F = \text{Hankel_Features}(X, \text{nFrame}, \text{lFrame})$
 - $\text{datF} = \text{Apilar_Features}(F)$
 - EndFor
 - Label = Binary_Label(i)
 - $Y = \text{Apilar_Label}(\text{Label})$
 - $X = \text{Apilar_Features}(\text{datF})$
- EndFor
- $X = \text{norm_data}(X)$ #Usar ecuación dada en ppt actual.
- $\text{Create_dtrn_dtst}(X, Y, p)$ # p: denota porcentaje de training.

Pre-proceso: Hankel_Features()

- Para cada Frame, Haga :
 - Descomponga el n -Frame en J -niveles usando la matriz Hankel diádica y el método SVD.
 - Entropy_C:
 - Entropía de Amplitud Espectral para cada uno de los componentes de Hankel.
 - Svalues_C:
 - Valores Singulares de la Matriz de Componentes de Hankel usando el método SVD.
 - Crear el n -ésimo vector de características :
 - $F(n,:) = [\text{Entropy_C} \quad \text{Svalues_C}]$
 - La dimensión de $F(n,:) = 2^J + 2^J$.



Pre-proceso: Create_dtrn_dtst()

- **Re-ordenar aleatoriamente las posiciones de la data de entrada X y data de salida Y.**
- **Dividir la data X y data Y:**
 - ☐ Data Trainig: p (%)
 - xe, ye
 - ☐ Data Testing: (1-p) %
 - xv,yv
- **Crear archivo de training csv:**
 - ☐ dtrain.csv # para xe , ye
- **Crear archivo de testing csv:**
 - ☐ dtest.csv # para xe , ye



train.py

- Cargar datos de configuración.
- Realizar el proceso de aprendizaje de la Red Neuronal Superficial (SNN) usando :
 - **Algoritmo Descenso del Gradiente Estocástico con Momentum (SGDM).**



SNN-SGDM

Algoritmo de Aprendizaje MiniBatch:

**Stochastic Gradient Descent
with Momentum
(SGDM)**



Training de SNN con SGDM

- Considerar una base de datos de N -muestras dada como:

$$\{X_i, Y_i\}_{i=1}^N, \quad X_i \in \mathbb{R}^d, \quad Y_i \in \mathbb{R}^m$$

- \mathbf{X}_i : representa la data de training
- \mathbf{Y}_i : representa la data deseada
- \mathbf{d} : denota el número de variables de entrada
- \mathbf{m} : denota en número de clases



Algoritmo MiniBatch-SGDM

- **Paso 1:** Re-ordenar aleatoriamente la localización de cada muestra de la base de datos de training.
- **Paso 2:** Dividir las N -muestras de la base de datos en Batch de M -muestras.

$$B = \frac{N}{M}$$

B : Número de Batch

- **Paso 3:** Entrenar la SNN usando un Número Máximo de Épocas (Iteraciones).
 - Cada Épocas ajusta los pesos de la SNN B -veces vía el **SGDM**.

Ajuste Pesos de Salida de SNN con SGDM

Costo:

$$E = \frac{1}{M} \sum_{n=1}^M C_n = \frac{1}{M} \sum_{n=1}^M \frac{1}{2} \sum_{k=1}^m \left(a_{k,n}^{(L)} - Y_{k,n} \right)^2$$

} $Loss$

**Notación Matricial:
Pesos de Salida**

$$\begin{aligned} w^{(L)}(\tau+1) &= w^{(L)}(\tau) - v^{(L)}(\tau+1) \\ v^{(L)}(\tau+1) &= \beta \times v^{(L)}(\tau) + \mu \times gW^{(L)} \\ \tau &= 1, 2, \dots, \text{MaxIter} \end{aligned}$$

} Act ④

**Notación Matricial:
Gradiente**

$$\begin{aligned} gW^{(L)} &= \frac{\partial E}{\partial w^{(L)}} = \delta^{(L)} \times \left(a^{(L-1)} \right)^T \\ \delta^{(L)} &= e^{(L)} \otimes f'(z^{(L)}) \end{aligned}$$

} EW Salida ⑦

Valores Iniciales:

$$\begin{aligned} \beta, \mu &\in (0, 1), \\ w^{(L)}(0) &= \text{random} \\ v^{(L)}(0) &= w^{(L)}.shape \\ v^{(L)}(0) &= 0 \end{aligned}$$

Ajuste Pesos Ocultos de SNN con SGDM

Notación Matricial: Pesos Ocultos

$$\begin{aligned}w^{(l)}(\tau+1) &= w^{(l)}(\tau) - v^{(l)}(\tau+1) \\v^{(l)}(\tau+1) &= \beta \times v^{(l)}(\tau) + \mu \times gW^{(l)} \\l &= L-1, L-2, \dots, 2, 1 \\ \tau &= 1, 2, \dots, \text{MaxIter}\end{aligned}$$

ACT

(S)

Notación Matricial: Gradiente

$$gW^{(l)} = \frac{\partial E}{\partial w^{(l)}} = \left\{ \left(w^{(l+1)} \right)^T \times \delta^{(l+1)} \right\} \otimes f'(z^{(l)}) \times \left(a^{(l-1)} \right)^T$$

6w

Valores Iniciales:

$$\begin{aligned}w^{(l)}(0) &= \text{random} \\v^{(l)}(0) &= w^{(l)}.shape \\v^{(l)}(0) &= 0\end{aligned}$$



Función de Activación : Capas Ocultas

- 1. ReLu:

$$f(x) = \begin{cases} x, & x > 0 \\ 0, & x \leq 0 \end{cases}, x \in \mathbb{R}^d$$

- 2. L-ReLu:

$$f(x) = \begin{cases} 0.01 x, & x < 0 \\ x, & x \geq 0 \end{cases}, x \in \mathbb{R}^d$$

- 3. ELU:

$$f(x) = \begin{cases} a(e^x - 1), & x \leq 0 \\ x, & x > 0 \end{cases}, x \in \mathbb{R}^d$$

- 4. SELU:

$$f(x) = \lambda \times \begin{cases} a(e^x - 1), & x \leq 0 \\ x, & x > 0 \end{cases}$$

$x \in \mathbb{R}^d, \lambda = 1.0507, a = 1.6732$

- 5. Sigmoidal:

$$f(x) = \frac{1}{1 + e^{-z}}, x \in \mathbb{R}^d$$



Función de Activación : Capas de Salida

- Sigmoidal:

$$f(x) = \frac{1}{1 + e^{-z}}, \quad x \in \mathbb{R}^d$$

PseudoCode: train_snn()

```
Param = load_param()
W,V = iniWs(Param)  # Setup Pesos
MSE = [];
For Iter =1 to MaxIter
    X,Y = sort_data_random(X,Y)
    [Cost W V] = trn_minibatch(X,Y,W,V, Param)
    MSE(Iter) = mean(Cost)
    If mod(Iter,10)==0
        printf('\n Iterar-SGD:', Iter, MSE(Iter))
    EndIf
EndFor
```



PseudoCode: trn_minibatch(X,Y,W,V, Param)

```
#N samples del dataset X, M-samples del Batch
nBatch = N/M
For n=1 to nBatch
    Idx = get_Idx_n_Batch(n, M)
    xe = X(:,Idx)
    ye = Y(:,Idx)
    Act = forward(xe, W, Param)
    → gW, Cost = gradW(Act, ye, W, Param)
    ↵ W, V = upd_WV_sgdm(W, V, gW, Param)
EndFor
```



train.py:

- **Cargar parámetros**
- **Cargar data de training**
- **Entrenar la SNN vía BP-SGDM**
- **Crear archivo de costo (MSE):**
 - **costo.csv**
 - T-filas por 1-columna.
 - **Crear archivo de pesos:**
 - **w_snn.npz.**



test.py

- Cargar data de test.
- Cargas peso entrenados.
- Realizar proceso forward de SNN.
 - Crear archivo de Matriz de Confusión:
 - **cmatriz.csv**.
 - Crear archivo F-scores:
 - **fscores.csv**
 - $(m+1)$ -filas por 1-columa
 - Fila $(m+1)$ representa el F-scores promedio de las m -clases.



Configuración: cnf.csv

■ Parámetros :

- Línea 1: Número de Clases : 3
- Línea 2: Número de Frame : 50
- Línea 3: Tamaño del Frame : 256
- Línea 4: Nivel de Descomposición : 3
- Línea 5: Nodos Ocultos de Capa₁. : 40
- Línea 6: Nodos Ocultos de Capa₂. : 10
- Línea 7: Función Activación Oculta : 3



Configuración : cnf.csv

- Línea 8: Porcentaje de Training : 0.8
- Línea 9: Tamaño miniBatch : 32
- Línea 10: Tasa de Aprendizaje : 0.1
- Línea 11: Coeficiente Beta : 0.8
- Línea 12: Número Máximo Iteraciones : 50
-
-



Normalización de Datos:

$$x = \frac{(x - x_{\min})}{(x_{\max} - x_{\min})} \times (b - a) + a, \quad a = 0.01 \quad b = 0.99$$



ENTREGA

■ **Lunes : 17/Abril/2022**

☐ Hora : 13:00 horas

☐ Lugar : Aula Virtual del curso

■ **Lenguaje Programación:**

☐ Python version: 3.7.6 window (anaconda)

■ Numpy/Panda



OBSERVACIÓN:

- Si un Grupo no Cumple con los requerimientos funcionales y no-funcionales, entonces la nota máxima será igual a 1,0 (uno coma cero).