TAREA #3

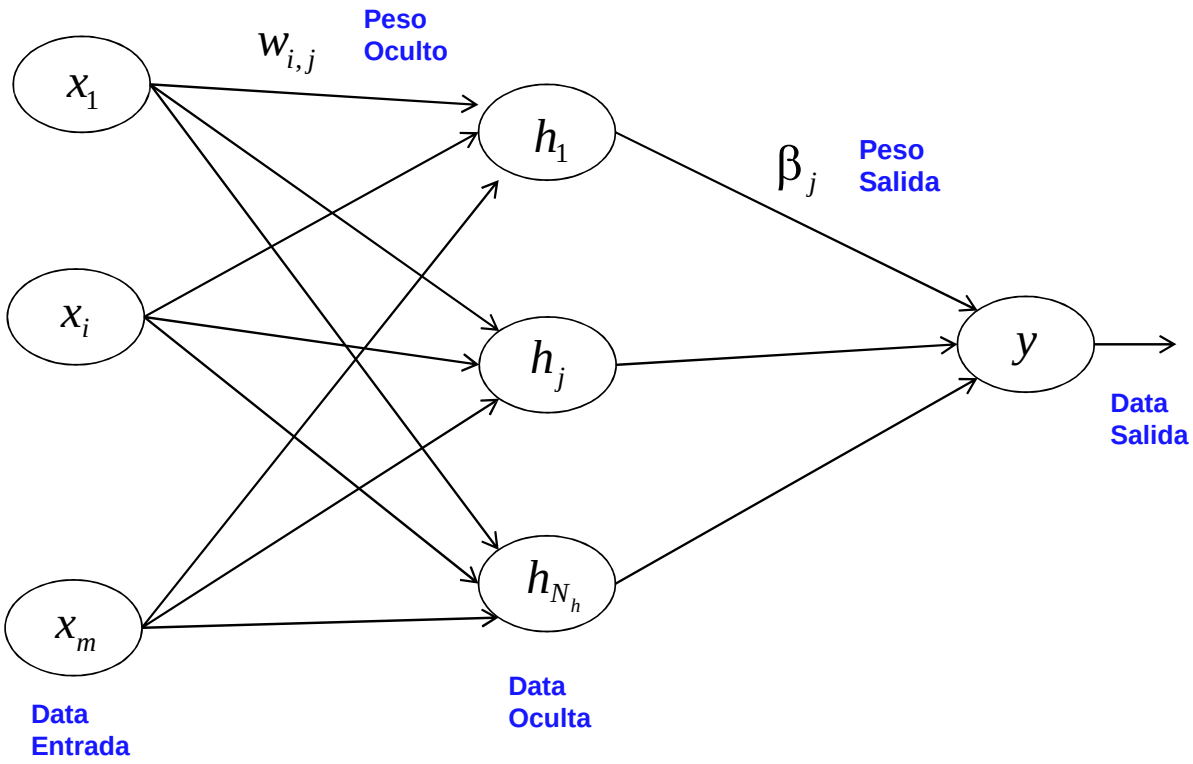
IDS

Prof. NIBALDO RODRÍGUEZ A.

OBJETIVO

- Implementar y Evaluar el rendimiento de un Sistema de Detección de Intrusos (IDS) usando Redes Neuronales Híbridas.

Red Neuronal Artificial



Red Neuronal Artificial

$$\hat{y} = \sum_{j=1}^{N_h} \beta_j \cdot h_j(x_n)$$

$$x_n \in \Re^m$$

Función de Activación

$$h_j(\cdot)$$

Red Neuronal Artificial

1.-Función de Activación

$$h_j(x_n) = \frac{1}{1+\exp(-z)}$$

$$z = \sum_{i=1}^{m} w_{j,i} x_{n,i}$$

Red Neuronal Artificial

2.-Función de Activación

$$h_j(x_n) = \frac{2}{1+\exp(-z)} - 1$$

$$z = \sum_{i=1}^{m} w_{j,i} x_{n,i}$$

Red Neuronal Artificial

3.-Función de Activación

$$h_j(x_n) = \exp\left(-0.5z^2\right)$$

$$z = \left\| x_n - w_j \right\|, \qquad x_n \in \Re^m, \; w_j \in \Re^m$$

Red Neuronal Artificial

4.-Función de Activación

$$h_j(x_n) = z \cdot \exp\left(-0.5z^2\right)$$

$$z = \left\| x_n - w_j \right\|$$

Red Neuronal Artificial

5.-Función de Activación

$$h_j(x_n) = \cos(5z)\exp\left(-0.5z^2\right)$$

$$z = \left\| x_n - w_j \right\|$$

Red Neuronal Artificial

6.-Función de Activación

$$h_j(x_n) = (1+z)^{-1/2}$$

$$z = \|x_n - w_j\|$$

OPTIMIZACIÓN DE PESOS

PESOS DE SALIDA

$$\beta = \left(H \times H^T + \frac{I}{C} \right)^{-1} \times H^T \times Y$$

OPTIMIZACIÓN DE PESOS

PESOS OCULTOS

$$w_{j,i},$$
$$j = 1, \ldots, N_h$$
$$i = 1, \ldots, m$$

Algoritmo: Particle Swarm Optimization (PSO)

ENJAMBRE

$$X, p \in \Re^{N_p \times D} \in rand(0,1)$$

$$N_p = Num.\ \text{partícula} \quad D = Dim. = N_h \times m$$

$$p_{ij}: \text{Partícula Local}$$

$$p_g \in \Re^D : \text{Mejor Partícula Global}$$

VELOCIDAD

$$V(k+1) = \alpha \cdot V(k) + c_1 r_1 \left[X(k) - p(k) \right] + c_2 r_2 \left[X(k) - p_g(k) \right]$$

$$r_1, r_2 \in rand(0,1)$$

$$c_1 = 1.05,\ c_2 = 2.95$$

Algoritmo: Particle Swarm Optimization

INERCIA

$$\alpha = \alpha_{max} - \left(\frac{\alpha_{max} - \alpha_{min}}{MaxIter} \right) \cdot IterActual$$

$$\alpha_{min} = 0.1 \quad \alpha_{max} = 0.95$$

NUEVO ENJAMBRE

$$X(k+1) = X(k) + V(k+1)$$

Algoritmo: Quantum-PSO

ENJAMBRE

$$X, p \in \Re^{N_p \times D} \in rand(0,1)$$

$$N_p = Num.\ \text{partícula} \quad D = Dim. = N_h \times m$$

$$p_{ij} = \varphi \cdot p_{ij} + (1-\varphi)\, p_{g,j}$$

$$p_g \in \Re^D : part.\ \text{global}, \quad p_{ij} = part.\ \text{local}$$

$$\varphi, \mu \in rand(0,1)$$

NUEVO ENJAMBRE

$$X_{ij} = \begin{cases} p_{ij} + \alpha \left| B_j - X_{ij} \right| \cdot Ln\left(\frac{1}{\mu}\right) & \text{Si} \quad rand \geq 0 \\ p_{ij} - \alpha \left| B_j - X_{ij} \right| \cdot Ln\left(\frac{1}{\mu}\right) & \text{o.c} \end{cases}$$

QPSO

Coeficientes

$$\alpha = (b-a)\left[\frac{MaxIter - IterActual}{MaxIter}\right] + a$$

$$a = 0.2 \qquad b = 0.95$$

$$B_j = \frac{1}{N_p}\sum_{i=1}^{N_p} p_{ij}$$

Función de Costo: PSO-QPSO

Minimizar Función de Costo: MSE (error cuadrático medio)

$$E = \frac{1}{N}\sum_{n=1}^{N} e_n^2$$

$$e_n = y_n - \hat{y}_n$$

Valor Esperado : y_n, Valor Estimado : $\hat{y}_n$

Número Muestra : N

DATA

- Training:
  - **KDDTrain.txt**
- Testing:
  - **KDDTest.txt**

DATA

- Formato :
  - **N-filas**: número de muestras.
  - **D-columnas**:
    - Las primeras 41-columnas denotas los atributos del clasificador.
    - La columna 42 representa la etiqueta de la clase:
      - Normal (N)
      - Ataque (A)

preproceso.py (kddtrain.txt)

- Transformar cada atributo (1-41) a formato numérico.
- Transformar atributo 42 a clase bi-polar. Esto es:
- Clase Normal: 1
- Clase Ataque: -1

preproceso.py (kddtrain.txt)

- Normalizar el Data set:
  - Columnas: 1-41

Norma

$$y = \frac{x - x_{\min}}{x_{\max} - x_{\min}} \qquad y = (b-a) \times y + a \qquad a = 0.1 \qquad b = 0.99$$

preproceso.py (kddtrain)

- Crear Nuevo Archivo con los Atributos transformado y normalizados:
  - **train**

preproceso.py (kddtest.txt)

- Transformar cada atributo (1-41) a formato numérico.
- Transformar atributo 42 a clase bi-polar. Esto es:
- Clase Normal: 1
- Clase Ataque: -1

preproceso.py (kddtest.txt)

- Normalizar el Data set:
  - Columnas: 1-41

Norma

$$y = \frac{x - x_{\min}}{x_{\max} - x_{\min}} \qquad y = (b-a) \times y + a \qquad a = 0.1 \qquad b = 0.99$$

preproceso.py (kddtest.txt)

- Crear Nuevo Archivo con los Atributos transformado y normalizados:
  - **test**

train.py

- **Parámetros:**
  - Nodos Ocultos :
  - Máx. Iteraciones :
  - Número de Partículas :
  - Penalidad Pseudo-inversa :
- Archivos de Salida:
  - Pesos ocultos y pesos de salida
  - Error cuadrático medio vs Iteraciones:
    - costo.csv

test.py

- Archivos de Salida:
  - metrica.csv:
    - Exactitud
    - F-score clase normal
    - F-score clase ataque

Evaluación de Rendimiento

Matriz de Confusión

		Esperado	
		P	N
PREDICHO	P	VP	FP
	N	FN	VN

$$P = \frac{VP}{VP + FP}$$

Precision (precisión)

$$R = \frac{VP}{VP + FN}$$

Recall (sensibilidad)

$$F = 2 \times \frac{P \times R}{P + R}$$

F-score

$$A = \frac{VP + VN}{VP + FP + FN + VN}$$

Accuracy (exactitud)

config.csv

- **Crear archivo de configuración:**
- **Parámetros:**
  - Línea 1: Número Nodos Ocultos : 40
  - Línea 2: Número de Partículas : 60
  - Línea 3: Máx. Iteraciones : 1000
  - Linea 4: Penalidad P-inversa : 100

GRUPOS

- Grupos: 1 al 6 con QPSO
- Grupos: 7 al 12 con PSO

ENTREGA

- Lunes 30/Noviembre/2020,
- Hora : 11:00
- Lugar : Aula Virtual del curso

- **Lenguaje Programación:**
  - Python version: 3.7.6 window (anaconda)