Día 3 - 17/11/21 (Miércoles)

Intro. a las Redes

De Neuronas Artificiales

Lic. Ronaldo Armando Canizales Turcios

UNIVERSIDAD DE GRANADA

Universidad Centroamericana José Simeón Cañas

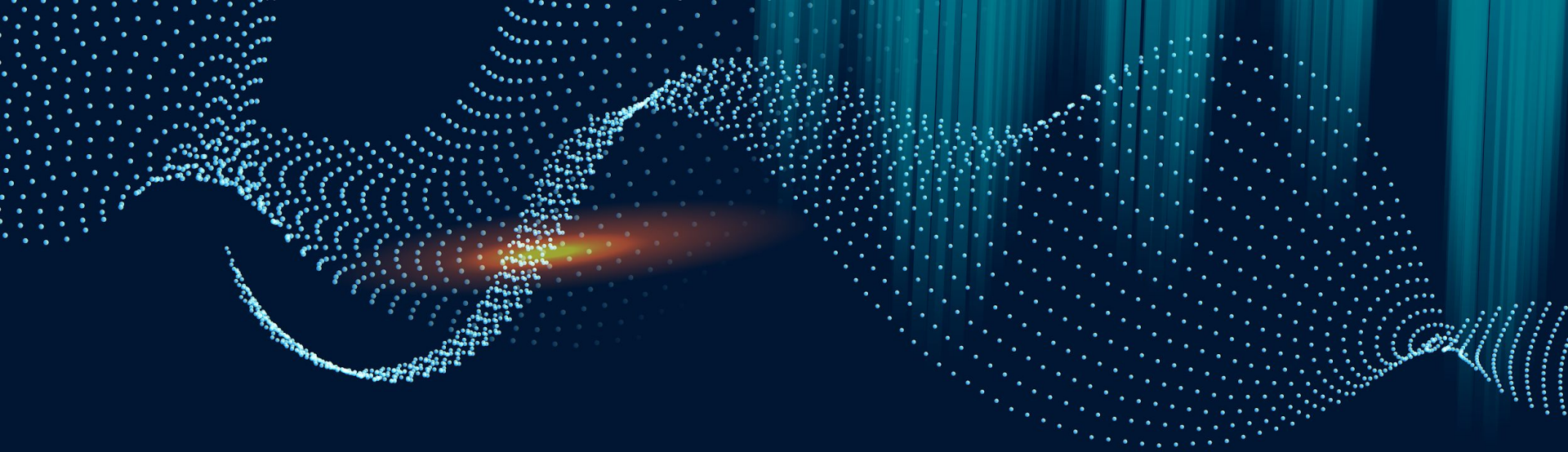
Agenda Día 3

Bloque A

- Redes de neuronas artificiales: modelos y conceptos.
- Aplicación de las RNA como aproximadores universales

Bloque B

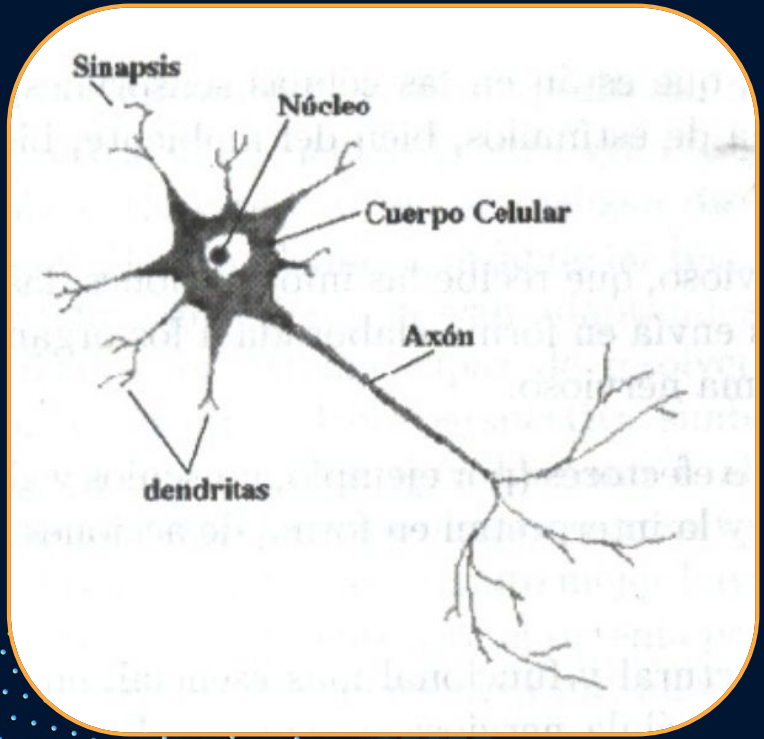
- Comprendiendo las RNA mediante el Deep Playground.
- Mi primer RNA: comprobación de autenticidad de notas de banco. **[Práctica]**



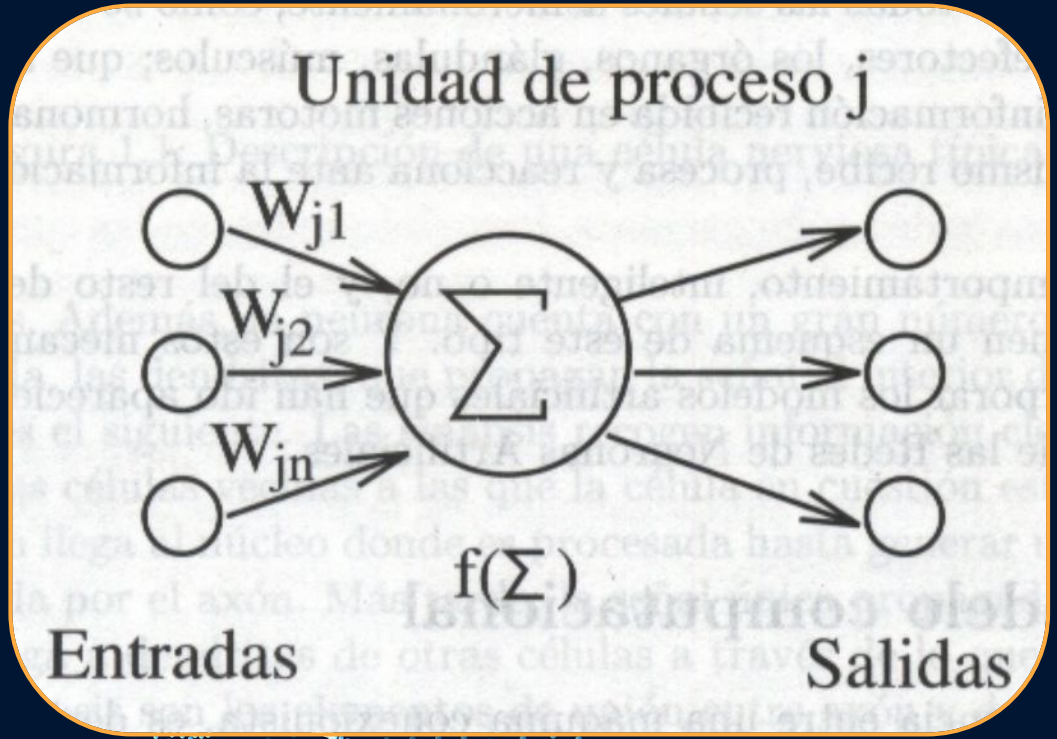
01 Modelos y conceptos

Redes de Neuronas Artificiales

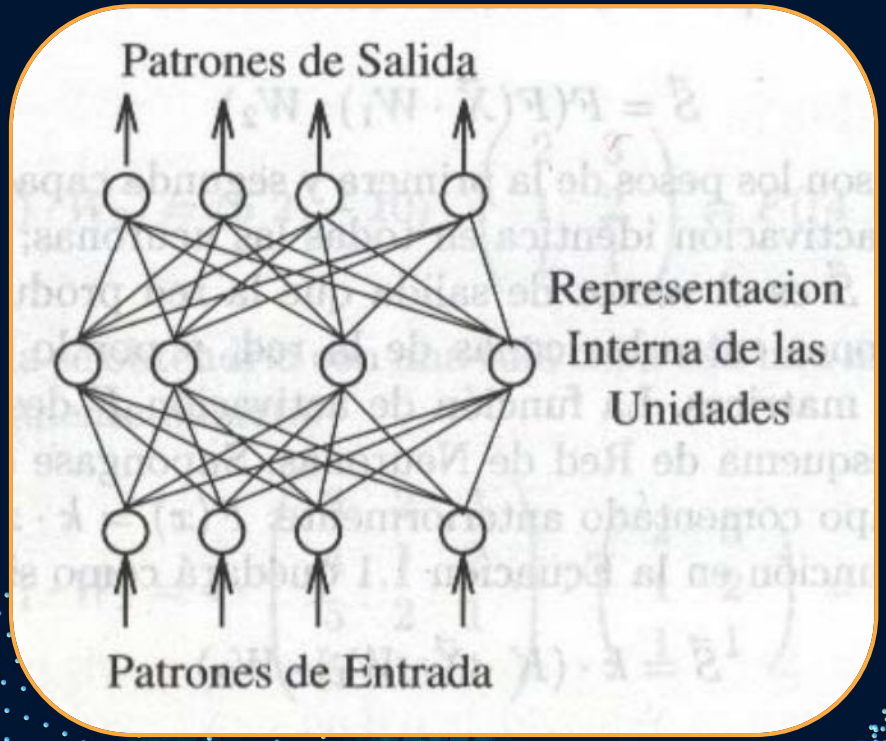
Neurona biológica típica



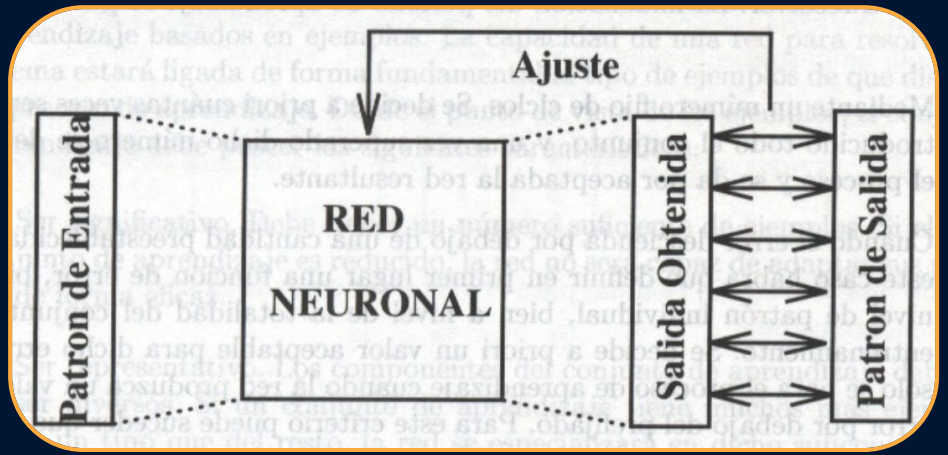
Esquema neurona artificial



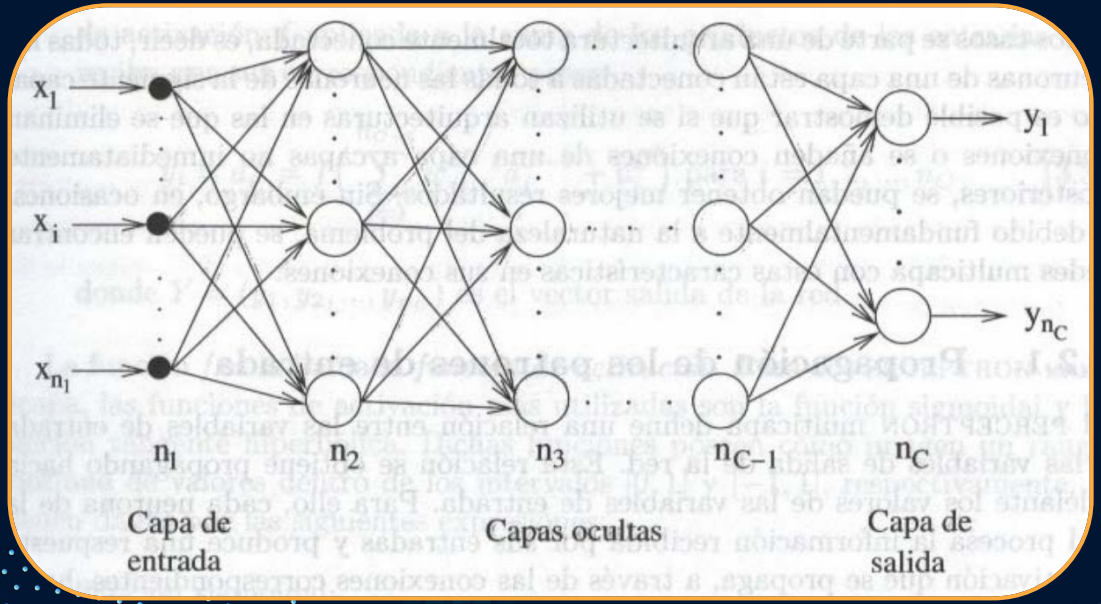
Esquema de una red de 3 capas totalmente interconectadas



Aprendizaje supervisado



Arquitectura de un perceptrón multicapa



- Capa de normalización.
- Capa de entrada.
- Capas ocultas
  (o de aprendizaje).
  - Cantidad de neuronas.
  - Función de activación.
- Capa de salida.
- Capa de desnormalización.
- Capa delimitadora (opcional).

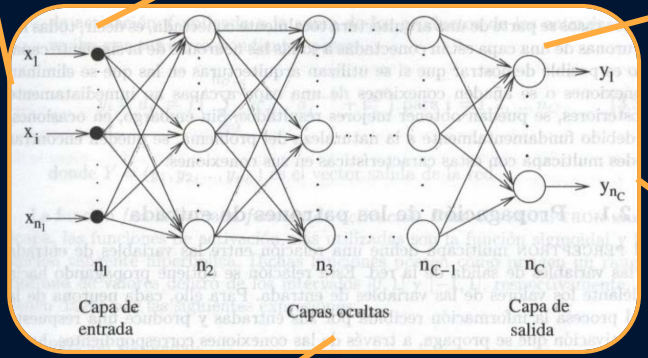
Proceso de aprendizaje supervisado

1. Inicializar pesos y umbrales con valores aleatorios.
2. Introducir una muestra “n” del conjunto de entrenamiento, obteniendo la respuesta de la red.
3. Obtener el error producido al predecir dicha muestra.
4. Aplicar el Algoritmo de RetroPropagación para modificar los pesos y umbrales de la red.
5. Repetir (2), (3) y (4) para todas las muestras del conjunto de entrenamiento (iteración / ciclo de aprendizaje).
6. Criterios de paro: error total E de la red ≈ 0, parámetros de la red estables (norma del gradiente final ≈ 0), cantidad máxima de iteraciones o tiempo máximo transcurrido.
7. Acabar el proceso de aprendizaje y dar como salida la red obtenida.

Propagación de los patrones de entrada

$$X = (x_1, x_2, ..., x_{n_1})$$

$$a_i^1 = x_i \text{ para } i = 1, 2, ..., n_1$$



$$y_i = a_i^C = f\left(\sum_{j=1}^{n_{C-1}} w_{ji}^{C-1} a_j^{C-1} + u_i^C\right) \text{ para } i = 1, 2, ..., n_C$$

$$Y = (y_1, y_2, ..., y_{n_C})$$

$$f_1(x) = \frac{1}{1 + e^{-x}}$$

$$f_2(x) = \frac{1 - e^{-x}}{1 + e^{-x}}$$

$$a_i^c = f\left(\sum_{j=1}^{n_{c-1}} w_{ji}^{c-1} a_j^{c-1} + u_i^c\right) \text{ para } i = 1, 2, ..., n_c \text{ y } c = 2, 3..., C-1$$

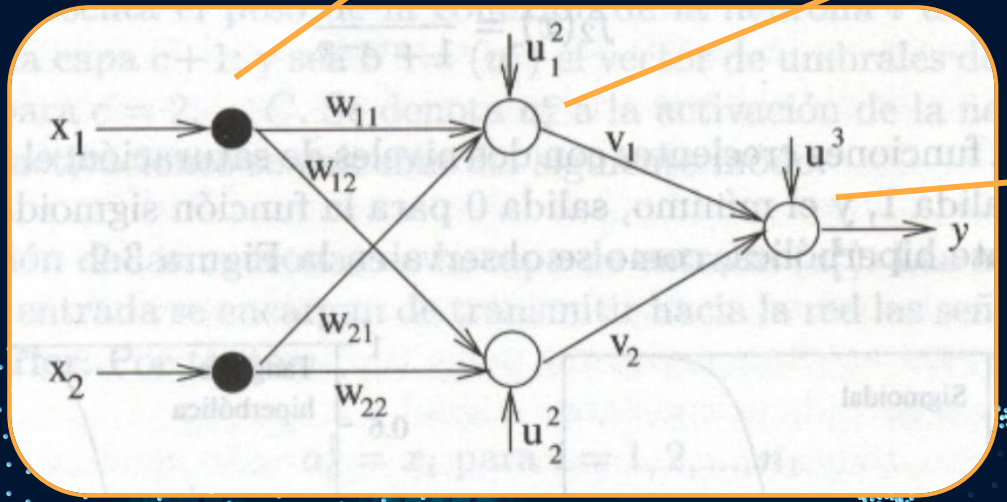
Activaciones de un Perceptrón Multicapa con 2 neuronas de entrada, dos ocultas y una salida

Activaciones de las neuronas de entrada:

$$a_1^1 = x_1 \text{ y } a_2^1 = x_2$$

Activaciones de las neuronas ocultas:

$$a_1^2 = f\left(w_{11}a_1^1 + w_{21}a_2^1 + u_1^2\right) \text{ y } a_2^2 = f\left(w_{12}a_1^1 + w_{22}a_2^1 + u_2^2\right)$$



Activación de la neurona de salida:

$$y = a_1^3 = f\left(v_1a_1^2 + v_2a_2^2 + u^3\right)$$

Algoritmo de RetroPropagación:

Encontrar (w) los pesos (v) y umbrales (u) que minimicen E

$$E = \frac{1}{N} \sum_{n=1}^{N} e(n)$$

$$w(n) = w(n-1) - \alpha \frac{\partial e(n)}{\partial w}$$

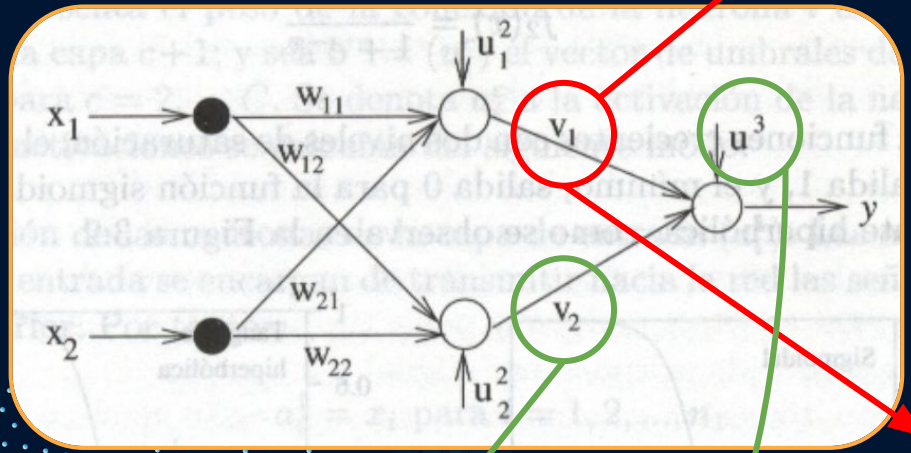
$$e(n) = \frac{1}{2} \sum_{i=1}^{n_C} (s_i(n) - y_i(n))^2$$

$$S(n) = (s_1(n), ..., s_{n_C}(n))$$

$$Y(n) = (y_1(n), ..., y_{n_C}(n))$$

Obteniendo la regla para un Perceptrón Multicapa con dos neuronas de entrada, dos ocultas y una salida

$$e(n) = \frac{1}{2}(s(n) - y(n))^2$$



$$w(n) = w(n-1) - \alpha \frac{\partial e(n)}{\partial w}$$

$$y = f\left(v_1 a_1^2 + v_2 a_2^2 + u^3\right)$$

$$\frac{\partial e(n)}{\partial v_1} = -(s(n) - y(n)) \frac{\partial y(n)}{\partial v_1}$$

$$\frac{\partial y(n)}{\partial v_1} = f'\left(v_1 a_1^2 + v_2 a_2^2 + u^3\right) a_1^2(n)$$

$$\delta^3(n) = -(s(n) - y(n)) f'\left(v_1 a_1^2 + v_2 a_2^2 + u^3\right)$$

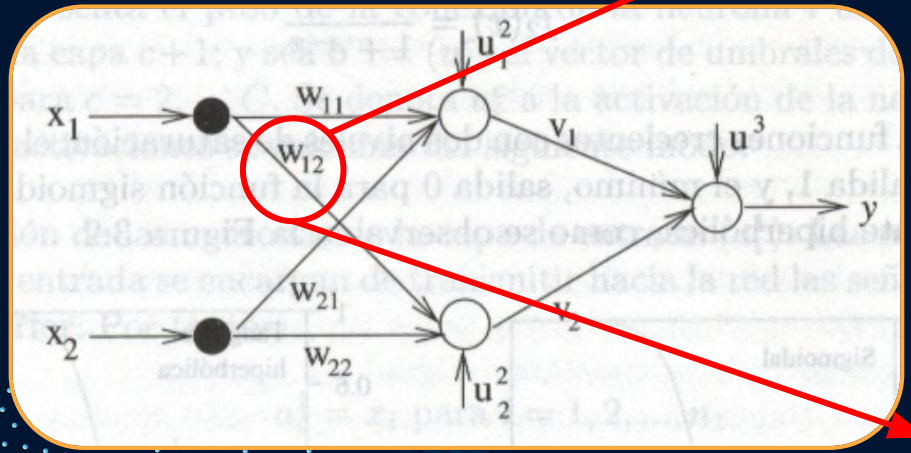
$$v_1(n) = v_1(n-1) + \alpha \delta^3(n) a_1^2(n)$$

$$v_2(n) = v_2(n-1) + \alpha \delta^3(n) a_2^2(n)$$

$$u^3(n) = u^3(n-1) + \alpha \delta^3(n)$$

Obteniendo la regla para un Perceptrón Multicapa con dos neuronas de entrada, dos ocultas y una salida

$$e(n) = \frac{1}{2}(s(n) - y(n))^2$$



$$w(n) = w(n-1) - \alpha \frac{\partial e(n)}{\partial w}$$

$$\frac{\partial e(n)}{\partial w_{12}} = -(s(n) - y(n)) \frac{\partial y(n)}{\partial w_{12}}$$

$$a_2^2 = f\left(w_{12}a_1^1 + w_{22}a_2^1 + u_2^2\right)$$

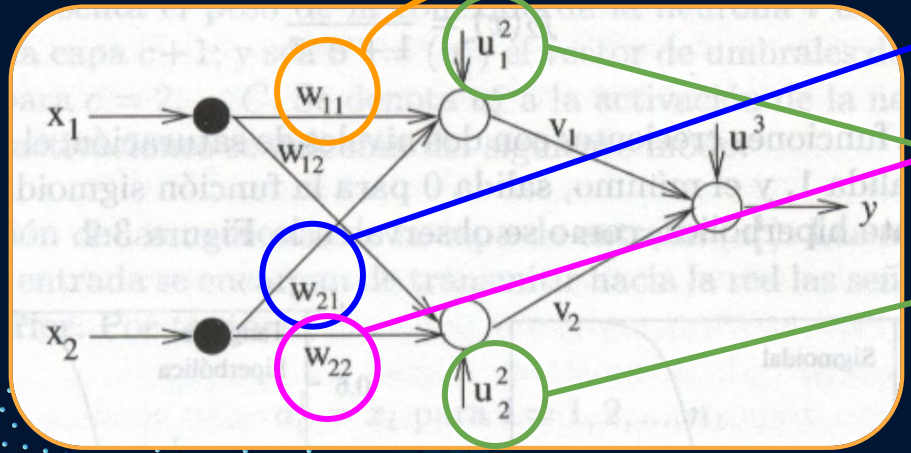
$$\frac{\partial y(n)}{\partial w_{12}} = f'\left(v_1 a_1^2 + v_2 a_2^2 + u^3\right) v_2 \frac{\partial a_2^2(n)}{\partial w_{12}}$$

$$\delta_2^2(n) = f'\left(w_{12}a_1^1 + w_{22}a_2^1 + u_2^2\right) v_2 \delta^3(n)$$

$$w_{12}(n) = w_{12}(n-1) + \alpha \delta_2^2(n) a_1^1(n)$$

$$\frac{\partial a_2^2(n)}{\partial w_{12}} = f'\left(w_{12}a_1^1 + w_{22}a_2^1 + u_2^2\right) a_1^1$$

Obteniendo la regla para un Perceptrón Multicapa con dos neuronas de entrada, dos ocultas y una salida



$$w_{11}(n) = w_{11}(n-1) + \alpha\delta_1^2(n)a_1^1(n)$$

$$w_{21}(n) = w_{12}(n-1) + \alpha\delta_1^2(n)a_2^2(n)$$

$$w_{22}(n) = w_{12}(n-1) + \alpha\delta_2^2(n)a_2^1(n)$$

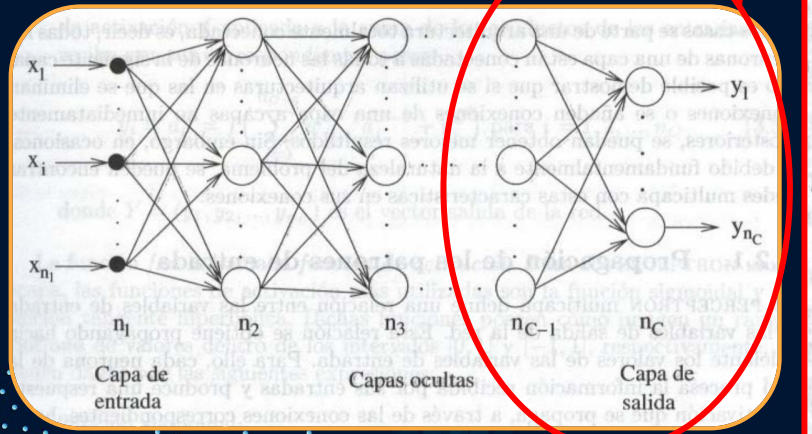
$$u_1^2(n) = u_1^2(n-1) + \alpha\delta_1^2(n)$$

$$u_2^2(n) = u_1^2(n-1) + \alpha\delta_2^2(n)$$

$$\delta_1^2(n) = f'\left(w_{11}a_1^1 + w_{21}a_2^1 + u_2^2\right)v_1\delta^3(n)$$

$$\delta_2^2(n) = f'\left(w_{12}a_1^1 + w_{22}a_2^1 + u_2^2\right)v_2\delta^3(n)$$

Resumen de la regla Delta generalizada



- **Pesos de la capa oculta $C-1$ a la capa de salida y umbrales de la capa de salida**

Pesos:

$$w_{ji}^{C-1}(n) = w_{ji}^{C-1}(n-1) + \alpha\delta_i^C(n)a_j^{C-1}(n)$$

$$\text{para } j = 1, 2, ..., n_{C-1} \text{ y } i = 1, 2, ..., n_C$$

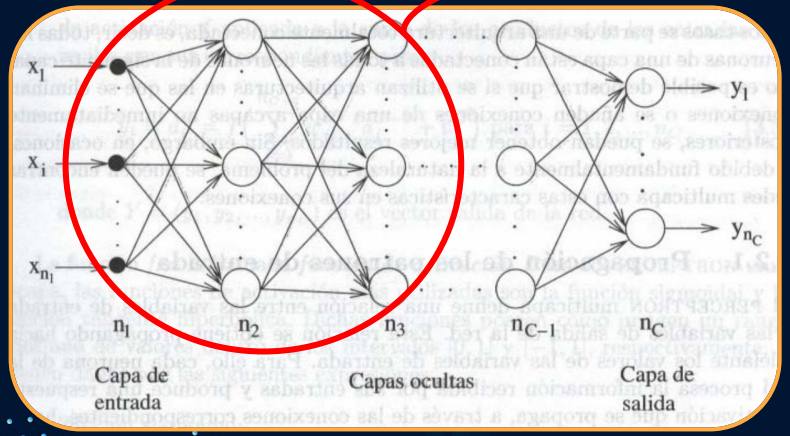
Umbrales:

$$u_i^C(n) = u_i^C(n-1) + \alpha\delta_i^C(n) \text{ para } i = 1, 2, ..., n_C$$

donde:

$$\delta_i^C(n) = -(s_i(n) - y_i(n))y_i(n)(1 - y_i(n))$$

Resumen de la regla Delta generalizada



- **Pesos de la capa c a la capa $c+1$ y umbrales de las neuronas de la capa $c+1$ para $c = 1, 2, ..., C-2$**

Pesos:

$$w_{kj}^c(n) = w_{kj}^c(n-1) + \alpha\delta_j^{c+1}(n)a_k^c(n) \quad (3.26)$$

$$\text{para } k = 1, 2, ..., n_c,\ j = 1, 2, ..., n_{c+1} \text{ y } c = 1, 2, ..., C-2$$

Umbrales:

$$u_j^{c+1}(n) = u_j^{c+1}(n-1) + \alpha\delta_j^{c+1}(n)$$

$$\text{para } j = 1, 2, ..., n_{c+1} \text{ y } c = 1, 2, ..., C-2$$

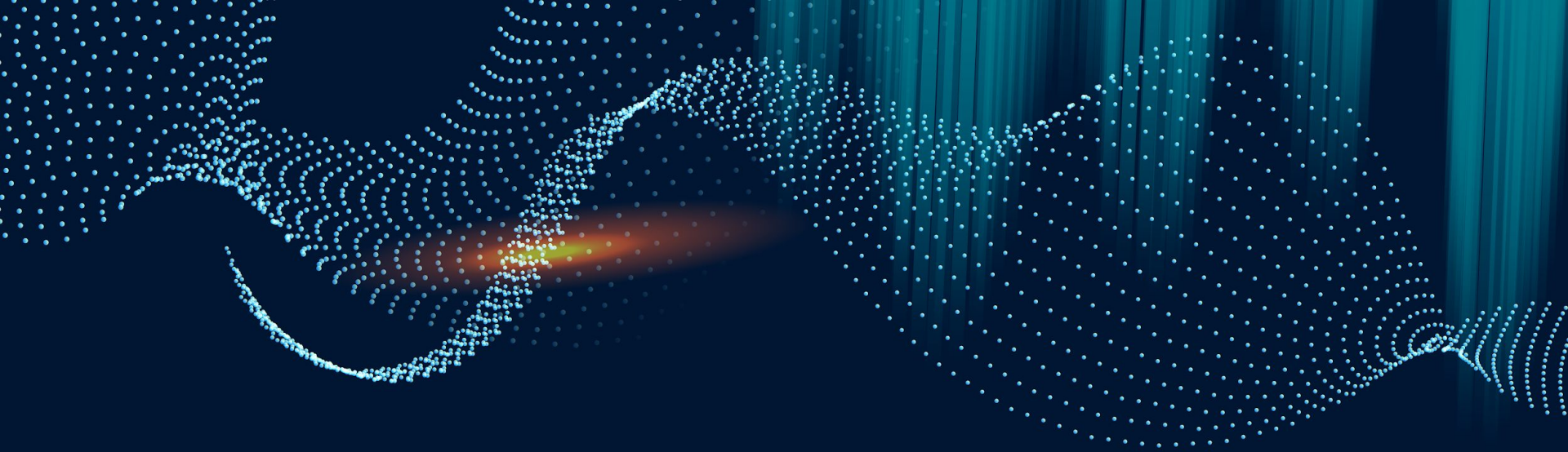
donde:

$$\delta_j^{c+1}(n) = a_j^c(n)(1 - a_j^c(n)) \sum_{i=1}^{n_{c+1}} \delta_i^{c+2}(n) w_{ji}^c$$

Una idea más intuitiva del proceso de aprendizaje del Perceptrón Multicapa puede resumirse de la siguiente forma:

Partiendo de un punto aleatorio $W(0)$ del espacio $\mathbf{R}^{n_w}$, donde n_w es el número de parámetros de la red -pesos más umbrales-, el proceso de aprendizaje desplaza el vector de parámetros $W(n-1)$ en el espacio $\mathbf{R}^{n_w}$ siguiendo la dirección negativa del gradiente del error en dicho punto, alcanzando así un nuevo punto en dicho espacio, $W(n)$, que estará más próximo al mínimo de la función error que el anterior. El proceso continúa hasta que se encuentre un mínimo de la función error E, lo cual sucede cuando $\frac{\partial E}{\partial w} \approx 0$. En este momento, y de acuerdo con la Ecuación (3.10), los parámetros dejan de sufrir cambios significativos de una iteración a otra y el proceso de aprendizaje finaliza.

$$w(n) = w(n-1) - \alpha \frac{\partial e(n)}{\partial w}$$



02 Aplicación de las RNA

Problema de regresión

Conjunto de datos

Precio de determinado activo financiero

```
dataset.head()
```

	Price	Moneyness	Time	Vol
0	0.211	0.934	0.660	0.266
1	0.038	1.011	0.065	0.207
2	0.163	1.187	0.883	0.292
3	0.091	1.245	0.648	0.271
4	0.193	0.813	0.125	0.216

En finanzas, el **grado del dinero** (moneyness) es una medida de la probabilidad de que un activo financiero tenga un valor positivo en su fecha de expiración.

Tiempo representado como fracción de un año.

Volatilidad futura del activo

Modelo de Black-Scholes

M = Moneyness
σ = Volatilidad
τ = Tiempo

Φ = Función de distribución normal acumulada

```
from scipy.stats import norm
N_d1 = norm.cdf(d1)
N_d2 = norm.cdf(d2)
```

$$P = e^{-q\tau}\Phi(d_1) - e^{-r\tau}K\Phi(d_2)$$

M = K/S

q = 0

$$d_1 = \frac{\ln(S/K) + (r - q + \sigma^2/2)\tau}{\sigma\sqrt{\tau}}$$

$$d_2 = \frac{\ln(S/K) + (r - q - \sigma^2/2)\tau}{\sigma\sqrt{\tau}}$$

r = 0.05

$$P = e^{-q\tau}\Phi\left(\frac{-\ln(M) + (r + \sigma^2/2)\tau}{\sigma\sqrt{\tau}}\right) - e^{-r\tau}M\Phi\left(\frac{-\ln(M) + (r - \sigma^2/2)\tau}{\sigma\sqrt{\tau}}\right)$$

Batería de modelos de Aprendizaje de Máquina a utilizar

```
models = []
models.append(('LR', LinearRegression()))
models.append(('LASSO', Lasso()))
models.append(('EN', ElasticNet()))
models.append(('KNN', KNeighborsRegressor()))
models.append(('CART', DecisionTreeRegressor()))
models.append(('SVR', SVR()))
```

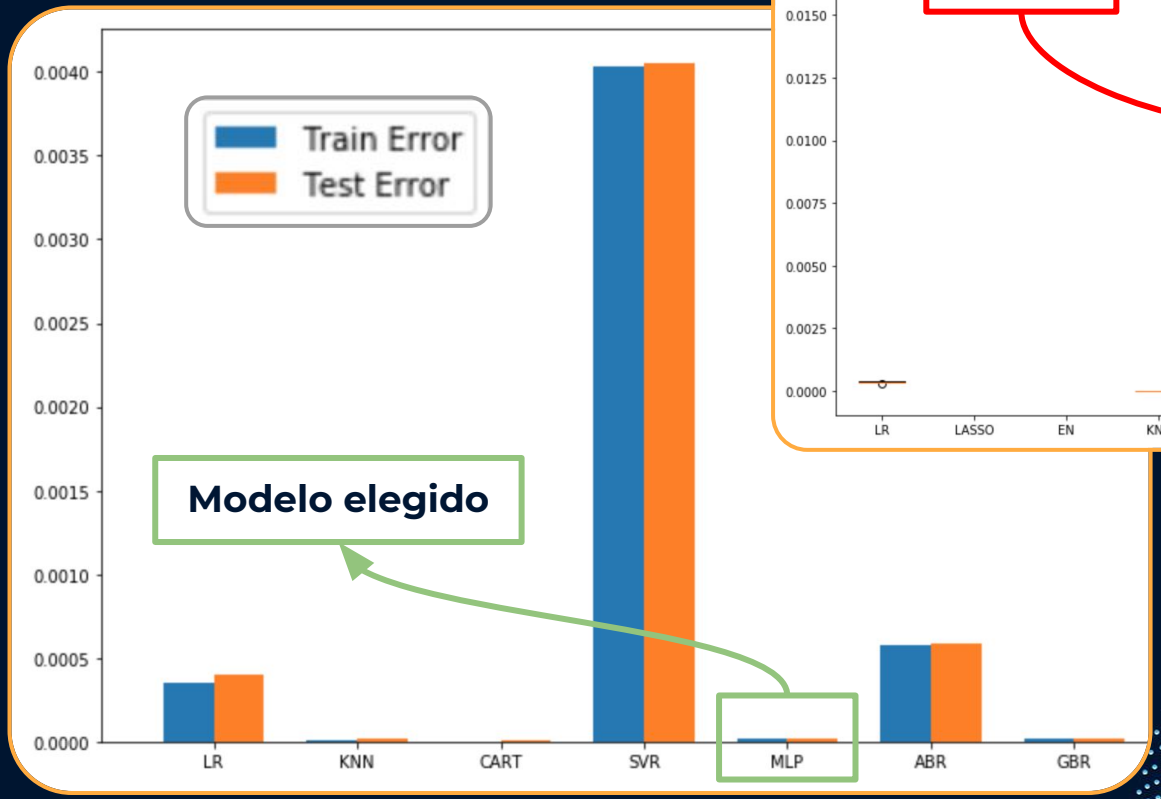
```
models.append(('MLP', MLPRegressor()))
```

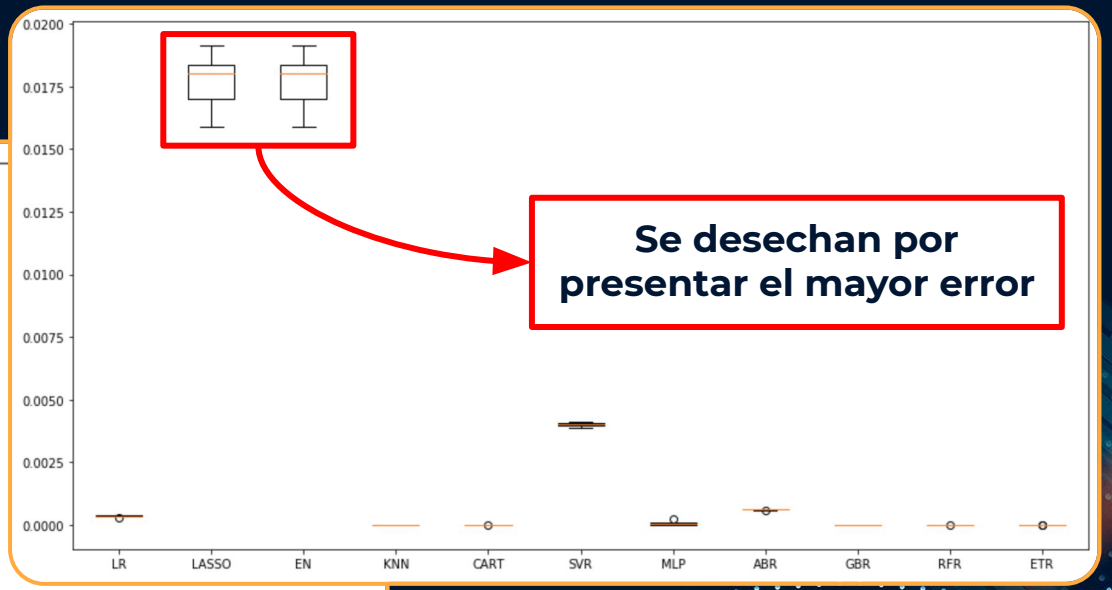
```
# Boosting methods
models.append(('ABR', AdaBoostRegressor()))
models.append(('GBR', GradientBoostingRegressor()))
# Bagging methods
models.append(('RFR', RandomForestRegressor()))
models.append(('ETR', ExtraTreesRegressor()))
```

```
names = []
kfold_results = []
test_results = []
train_results = []
for name, model in models:
    names.append(name)

    ## K Fold analysis:

    kfold = KFold(n_splits=num_folds, random_state=seed)
    #converted mean square error to positive. The lower the beter
    cv_results = -1* cross_val_score(model, X_train, Y_train, cv=kfold, scoring=scoring)
    kfold_results.append(cv_results)


    # Full Training period
    res = model.fit(X_train, Y_train)
    train_result = mean_squared_error(res.predict(X_train), Y_train)
    train_results.append(train_result)

    # Test results
    test_result = mean_squared_error(res.predict(X_test), Y_test)
    test_results.append(test_result)

    msg = "%s: %f (%f) %f %f" % (name, cv_results.mean(), cv_results.std(), train_result, test_result)
    print(msg)
```

```
validation_size = 0.2

train_size = int(len(X) * (1-validation_size))
X_train, X_test = X[0:train_size], X[train_size:len(X)]
Y_train, Y_test = Y[0:train_size], Y[train_size:len(X)]
```

Comparación de desempeño





Probando con varias arquitecturas

3 capas
20, 30 y 20 neuronas

```
param_grid={'hidden_layer_sizes': [(20,), (50,), (20,20), (20, 30, 20)]}
model = MLPRegressor()
kfold = KFold(n_splits=num_folds, random_state=seed)
grid = GridSearchCV(estimator=model, param_grid=param_grid, scoring=scoring, cv=kfold)
grid_result = grid.fit(X_train, Y_train)
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print("%f (%f) with: %r" % (mean, stdev, param))
```

1 capa y 20 neuronas
1 capa y 50 neuronas
2 capa y 20 neuronas

```
Best: -0.000024 using {'hidden_layer_sizes': (20, 30, 20)}
-0.000640 (0.000621) with: {'hidden_layer_sizes': (20,)}
-0.000169 (0.000155) with: {'hidden_layer_sizes': (50,)}
-0.000085 (0.000077) with: {'hidden_layer_sizes': (20, 20)}
-0.000024 (0.000012) with: {'hidden_layer_sizes': (20, 30, 20)}
```

```
# prepare model
model_tuned = MLPRegressor(hidden_layer_sizes=(20, 30, 20))
model_tuned.fit(X_train, Y_train)
```

```
predictions = model_tuned.predict(X_test)
print(mean_squared_error(Y_test, predictions))
```

```
3.081272766095 67e-05
```

Nuestro modelo final comete un error estimado de 3.08e-5 ¡Menos que un centavo!

https://playground.tensorflow.org/

03 Deep Playground

Imaginando las redes de neuronas artificiales

04 Mi primer RNA

Comprobación de autenticidad **[Práctica]**

Pasos a seguir

```
import tensorflow as tf
print(tf.__version__)
```

```
2.7.0
```

```
import seaborn as sns
import pandas as pd
import numpy as np
from tensorflow.keras.layers import Dense, Dropout, Activation
from tensorflow.keras.models import Model, Sequential
from tensorflow.keras.optimizers import Adam
```

Descargar archivo CSV y colocarlo en el Drive

Utilizar pd.read_csv para leer el archivo y guardarlo en la variable banknote_data

https://github.com/AbhiRoy96/Banknote-Authentication-UCI-Dataset

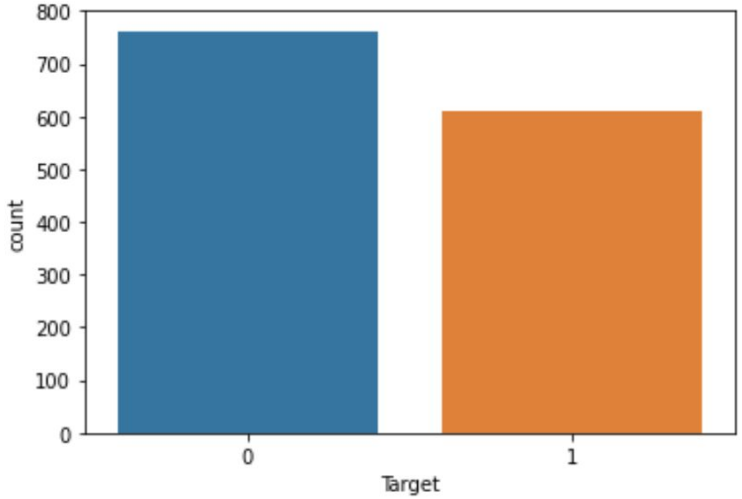
Análisis exploratorio

```
banknote_data.head()
```

	variance	skewness	curtosis	entropy	Target
0	3.62160	8.6661	-2.8073	-0.44699	0
1	4.54590	8.1674	-2.4586	-1.46210	0
2	3.86600	-2.6383	1.9242	0.10645	0
3	3.45660	9.5228	-4.0112	-3.59440	0
4	0.32924	-4.4552	4.5718	-0.98880	0

```
sns.countplot(x='Target', data=banknote_data)
```

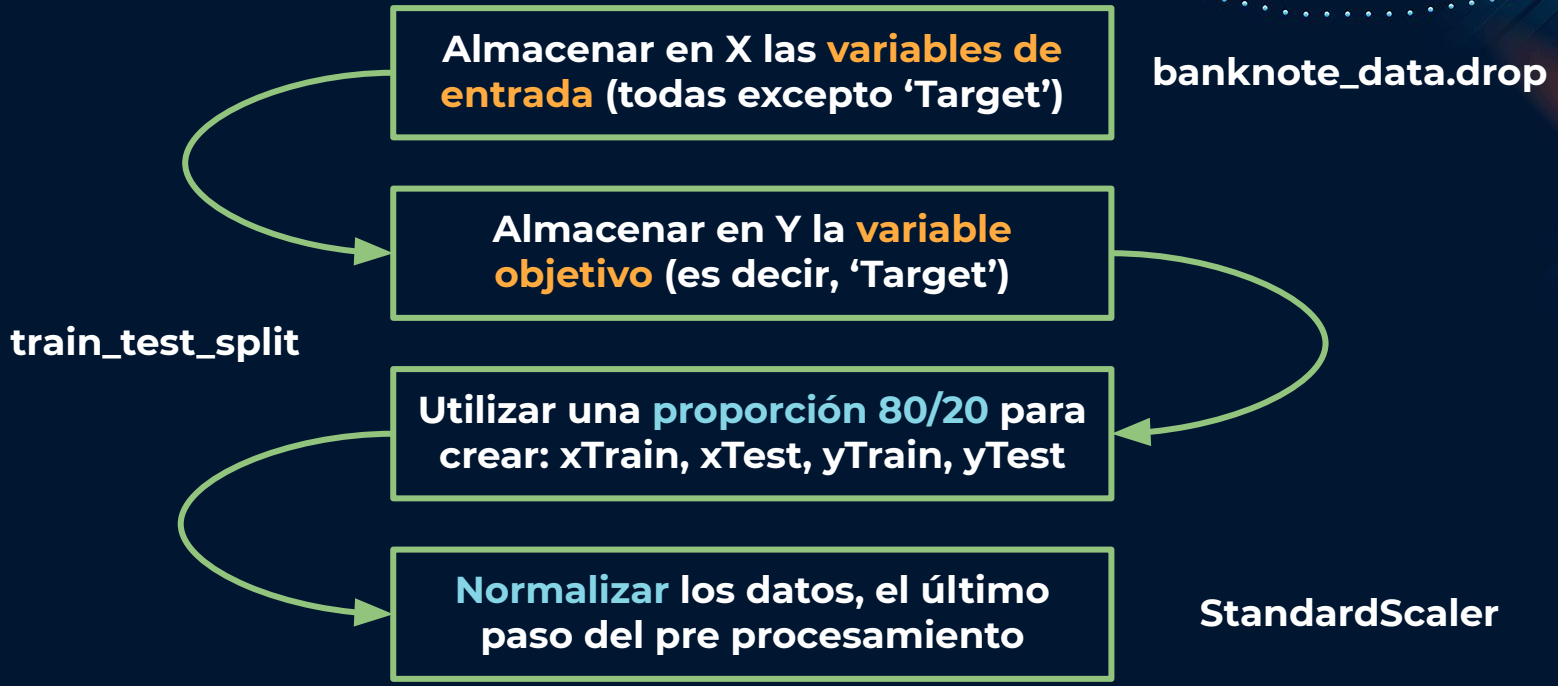
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f2af8342650>
```



No hay variables categóricas que necesiten transformación

Verificando que la variable objetivo sea significativa y representativa

Preprocesamiento de las características

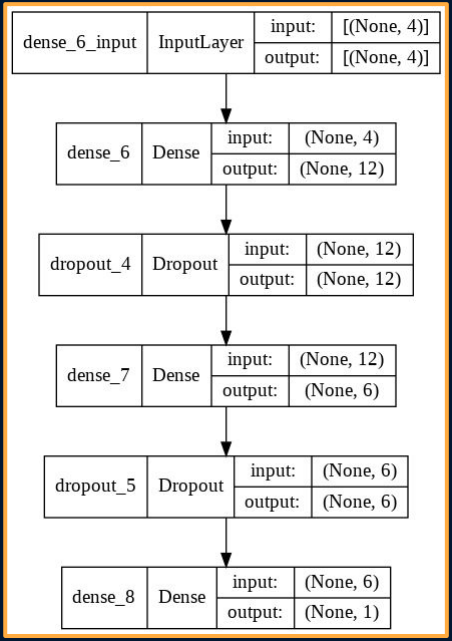


Crear la arquitectura de la RNA

```
def crear_modelo(taza_aprendizaje, taza_abandono):
  #Crear modelo
  model = Sequential()

  #Agregar capas
  model.add(Dense(12, input_dim=xTrain.shape[1], activation='relu'))
  model.add(Dropout(taza_abandono))
  model.add(Dense(6, activation='relu'))
  model.add(Dropout(taza_abandono))
  model.add(Dense(1, activation='sigmoid'))

  #Compilando el modelo
  adam = Adam(learning_rate=taza_aprendizaje)
  model.compile(optimizer=adam, metrics=['accuracy'], loss='binary_crossentropy')
  return model
```

```
model = crear_modelo(taza_aprendizaje, taza_abandono)
```

```
from tensorflow.keras.utils import plot_model
plot_model(model, to_file='diagrama.png', show_shapes=True, show_layer_names=True)
```



Visualizar las capas de la RNA

Entrenar la RNA

```
Epoch 1/20
220/220 [==============================] - 0s 2ms/step - loss: 0.3409 - accuracy: 0.8974
Epoch 2/20
220/220 [==============================] - 0s 2ms/step - loss: 0.2039 - accuracy: 0.9418
Epoch 3/20
220/220 [==============================] - 0s 2ms/step - loss: 0.1472 - accuracy: 0.9532
Epoch 4/20
220/220 [==============================] - 0s 2ms/step - loss: 0.1057 - accuracy: 0.9669
Epoch 5/20
220/220 [==============================] - 0s 2ms/step - loss: 0.1010 - accuracy: 0.9704
```

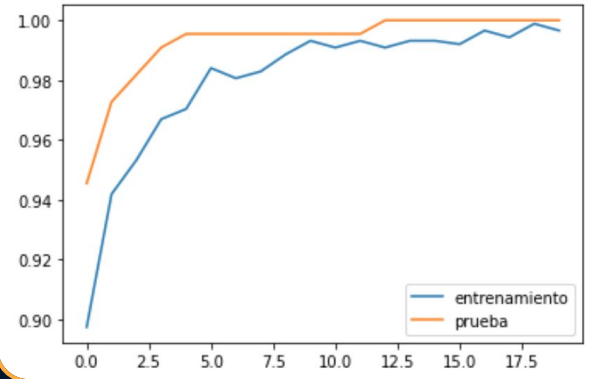
```
model_history = model.fit(xTrain, yTrain, batch_size=4, epochs=20, validation_split=0.2, verbose=1)
```

Finalmente, evaluar la RNA

```
accuracies = model.evaluate(xTest, yTest, verbose=1)
print('Test score: ', accuracies[0])
print('Test Accuracy: ', accuracies[1])
```

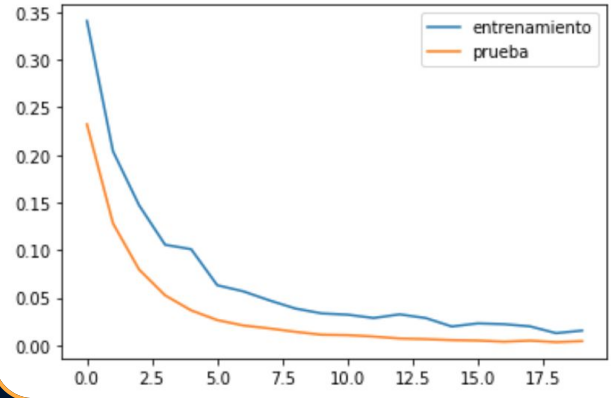
```
import matplotlib.pyplot as plt
plt.plot(model_history.history['accuracy'], label='accuracy')
plt.plot(model_history.history['val_accuracy'], label='val_accuracy')
plt.legend(['entrenamiento', 'prueba'], loc='lower right')
```

```
<matplotlib.legend.Legend at 0x7f2af1ff0a50>
```




```
plt.plot(model_history.history['loss'], label='pérdida')
plt.plot(model_history.history['val_loss'], label='val_pérdida')
plt.legend(['entrenamiento', 'prueba'], loc='upper right')
```

```
<matplotlib.legend.Legend at 0x7f2af1fd88d0>
```



¿Consultas,

dudas o comentarios?

Muchas gracias por su asistencia y atención

UNIVERSIDAD DE GRANADA

Universidad Centroamericana José Simeón Cañas