EJERCICIO 1. (3 Puntos) **Modelado conceptual**

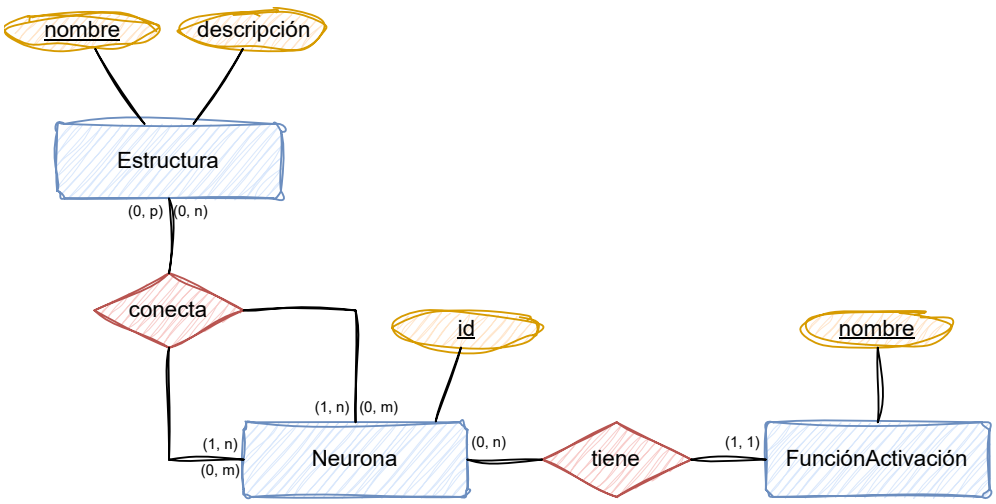
Queremos guardar información sobre estructuras cerebrales. Estas estructuras están formadas por neuronas, identificadas por un id único. Además, pueden tener asociada una única función de activación (de momento ReLU, sigmoide y tangente hiperbólica).

Dentro de una misma estructura, cada neurona tiene que estar conectada al menos a otra neurona que pertenezca a la misma estructura, estando cada una de estas conexiones caracterizada por un peso. Eso sí, dentro de una misma estructura dos neuronas solo se pueden conectar una única vez entre ellas.

Podemos tener diferentes estructuras cerebrales, cada una identificada por un nombre único y una descripción, y las neuronas pueden pertenecer a diferentes estructuras a la vez (como mínimo a una), si bien cada par conectado debe estar dentro de la misma estructura.

- Realiza un modelo conceptual de datos mediante la técnica del modelo entidad-relación de Chen que modele el sistema descrito.
- Realiza el paso a tablas del diagrama que has propuesto.

Solución propuesta:



FUNCION_ACTIVACIÓN (nombre)

NEURONA (id, f_act)

CONEXIÓN (neurona_origen, neurona_destino, estructura, peso)

ESTRUCTURA (nombre, descripción)

EJERCICIO 2. (5 Puntos) **Álgebra relacional y SQL**

Supón que estamos modelando un perceptrón multicapa. Este está formado por varias capas (una de entrada, una de salida y varias ocultas), cada capa contiene varias neuronas y las conexiones siempre van de una capa a la siguiente. Además las capas tienen un orden, que viene definido por una posición que indica su posición en la red.

`CAPA (id_c, tipo, pos) - NEURONA (id, id_c, f_act) - ENLACE (capa_from, capa_to, peso)`

Se pide:

(a) (1 Punto) Álgebra relacional

I. (½ Punto) Describe con tus propias palabras los siguientes operadores: producto cartesiano, intersección, diferencia, división y proyección.

Solución propuesta:

- Producto cartesiano ($\times$): Combina cada tupla de una relación con todas las tuplas de otra.
- Intersección ($\cap$): Devuelve las tuplas comunes a dos relaciones.
- Diferencia ($-$): Devuelve las tuplas que están en la primera relación pero no en la segunda.
- División ($\div$): Devuelve los elementos de una relación que están relacionados con todos los elementos de otra relación.
- Proyección (Π): Selecciona columnas específicas de una relación.

II. (½ Punto) Expresa en álgebra relacional la consulta que obtiene los identificadores de las neuronas que pertenecen a capas de tipo «oculta» y cuya función de activación es «ReLU».

Solución propuesta:

$$\Pi_{id_c}(\sigma_{f\_act=\text{«ReLU»}}(NEURONA) \bowtie \sigma_{tipo=\text{«oculta»}}(CAPA))$$

(b) (2 Puntos) Consultas SQL

I. (½ Punto) Obtener los id de todas aquellas capas de tipo «oculta» que no tienen ningún enlace entrante.

Solución propuesta:

```sql
SELECT id_c
FROM CAPA
WHERE tipo = 'oculta'
  AND id_c NOT IN (SELECT capa_to FROM ENLACE);
```

II. (½ Punto) Obtener el número total de neuronas que hay en la capa de salida.

Solución propuesta:

```
SELECT COUNT(*)
FROM NEURONA
WHERE id_c IN (SELECT id_c FROM CAPA WHERE tipo = 'salida'
    );
```

o

```
SELECT COUNT(DISTINCT id)
FROM NEURONA n
JOIN CAPA c ON c.id = n.ic_c
WHERE tipo = 'salida';
```

III. (½ Punto) Obtener los id de las capas cuya suma total de pesos de los enlaces salientes sea negativa.

Solución propuesta:

```
SELECT capa_from
FROM ENLACE
GROUP BY capa_from
HAVING SUM(peso) < 0;
```

IV. (½ Punto) Obtener los id de aquellas neuronas ocultas cuya función de activación sea «sigmoide».

Solución propuesta:

```
SELECT id
FROM NEURONA
WHERE f_act = 'sigmoide'
  AND id_c IN (SELECT id_c FROM CAPA WHERE tipo = 'oculta'
      );
```

o

```
SELECT n.id
FROM NEURONA n
JOIN CAPA c ON c.id = n.id_c
WHERE n.f_act = 'sigmoide'
  AND c.tipo = 'oculta';
```

(c) (1 Punto) Escribe un *trigger* que impida insertar un enlace cuya capa de origen no sea inmediatamente anterior a la capa de destino.

Solución propuesta:

```
DELIMITER $$

CREATE TRIGGER validar_enlace
BEFORE INSERT ON ENLACE
FOR EACH ROW
BEGIN
  DECLARE pos_from INT;
  DECLARE pos_to INT;

  SELECT pos INTO pos_from FROM CAPA WHERE id_c = NEW.
     capa_from;
  SELECT pos INTO pos_to FROM CAPA WHERE id_c = NEW.capa_to;

  IF pos_to <> pos_from + 1 THEN
    SIGNAL SQLSTATE '7291'
    SET MESSAGE_TEXT = 'Las capas no son consecutivas';
  END IF;
END;
$$

DELIMITER ;
```

(d) (1 Punto) Escribe un procedimiento almacenado que, dado el id de una capa oculta (comprobarlo), devuelva el número de enlaces entrantes y salientes (por separado) que tiene.

Solución propuesta:

```
DELIMITER $$

CREATE PROCEDURE contar_enlaces_oculta(
    IN id_capa INT,
    OUT num_entrantes INT,
    OUT num_salientes INT
)
BEGIN
  DECLARE tipo_capa VARCHAR(20);

  SELECT tipo INTO tipo_capa
  FROM CAPA
  WHERE id_c = id_capa;

  IF tipo_capa = 'oculta' THEN
    SELECT COUNT(*) INTO num_entrantes
    FROM ENLACE
    WHERE capa_to = id_capa;

    SELECT COUNT(*) INTO num_salientes
    FROM ENLACE
    WHERE capa_from = id_capa;
  ELSE
    SIGNAL SQLSTATE '7291'
```

```
    SET MESSAGE_TEXT = 'La capa tiene que ser oculta';
  END IF;
END;
$$

DELIMITER ;
```

EJERCICIO 3. (2 Puntos) **Programación contra bases de datos.**

Tomando como punto de partida un esquema **basado** en el modelo relacional anterior (el del bloque 2), completa los huecos (marcados con `TODO` del siguiente código fuente en Python de tal manera que se satisfaga la funcionalidad indicada en los *docstrings*.

```
import mysql.connector

conn = mysql.connector.connect(
    host="nasa.gov",
    user="romualdo",
    password="RuPauL.Dr4gR4c3",
    database="redes_neuronales"
)
cursor = conn.cursor()

def obtener_neuronas_capa(pos):
    """Obtiene las neuronas de una capa dada su posición en la red.

    :param pos: La posición de la capa dentro de la red:
    :return: Una lista con los id y las funciones de activación de
        las neuronas.
    """
    # TODO
```

Solución propuesta:

```
    consulta = """
    SELECT NEURONA.id, NEURONA.f_act
    FROM NEURONA
    JOIN CAPA ON NEURONA.id_c = CAPA.id_c
    WHERE CAPA.pos = %s;
    """

    cursor.execute(consulta, (id_capa,))
    return cursor.fetchall()

def insertar_enlace(id_capa_origen, id_capa_destino, peso):
    """Añade un nuevo enlace entre dos capas consecutivas.

    :param id_capa_origen: La capa de origen.
    :param id_capa_destino: La capa destino.
    :param El peso a asignar a toda las neuronas de las dos capas
```

```
    :return: True si las capas son consecutivas y se ha insertado el
        enlace o False en caso contrario
    """
    # TODO
```

Solución propuesta:

```
    cursor.execute("SELECT pos FROM CAPA WHERE id_c = %s", (
        id_capa_origen,))
    pos_origen = cursor.fetchone()[0]

    cursor.execute("SELECT pos FROM CAPA WHERE id_c = %s", (
        id_capa_destino,))
    pos_destino = cursor.fetchone()[0]

    if pos_destino == pos_origen + 1:
        cursor.execute(
            "INSERT INTO ENLACE (capa_from, capa_to, peso)
                        VALUES (%s, %s, %s)",
            (id_capa_origen, id_capa_destino, peso)
        )
        conn.commit()
        return True
    return False
```

```
\begin{solutionorbox}

def capas_sin_enlaces_salientes():
    """Lista todas las capas que no tienen ningun enlace saliente.

    :return: Lista con los id de las capas, ordenadas por posición.
    """
    # TODO
```

Solución propuesta:

```
    consulta = """
    SELECT id_c
    FROM CAPA
    WHERE id_c NOT IN (SELECT capa_from FROM ENLACE)
    ORDER BY pos;
    """

    cursor.execute(consulta)
    return cursor.fetchall()

conn.close()
```