UACM
Universidad Autónoma
de la Ciudad de México
NADA HUMANO ME ES AJENO

# Redes Neuronales

Sabino Miranda

# Neurona de McCulloch y Pitts

# Neurona de McCulloch y Pitts (1)

- McCulloch y Pitts presentan el primer modelo que aproxima el comportamiento de las neuronas biológicas (1943).

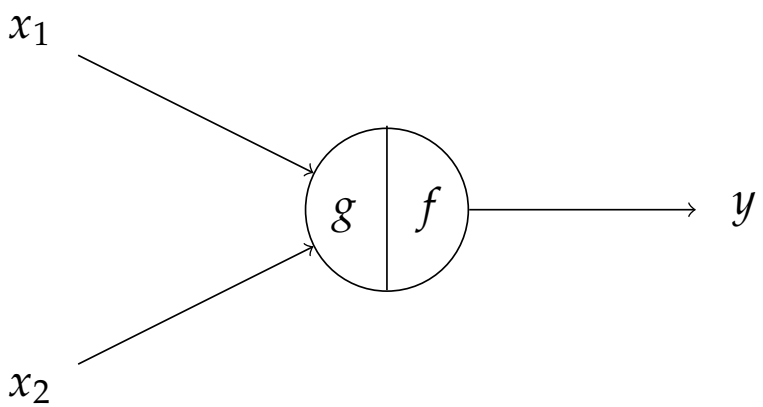



Figura 1: Neurona McCulloch-Pitts

# Neurona de McCulloch y Pitts (2)

- Entradas: La neurona recibe múltiples entradas, cada una con un valor binario (0 o 1). Las entradas pueden ser inhibidoras o exitatorias.
- Una unidad (neurona) McCulloch-Pitts puede ser inactivada por una única señal inhibidora, como sucede con algunas neuronas reales.
- Cuando no hay señales inhibidoras presentes, las unidades actúan como una puerta de umbral capaz de implementar funciones lógicas de n argumentos.

$$g(x_1, x_2, \ldots x_n) = \sum_{i=1}^{n} x_i$$

# Neurona de McCulloch y Pitts (3)

La función $y = f(g(x))$ se define como:

$$y = f(g(x)) = \begin{cases} 1 & \text{si } g(x) \geq \theta \\ 0 & \text{si } g(x) < \theta \end{cases}$$

- g(x) realiza la sumatoria; $\theta$ es el parámetro de umbral

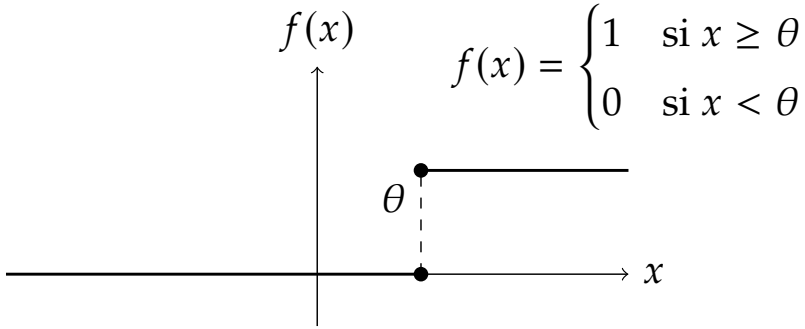



Figura 2: Función con umbral

# Neurona de McCulloch y Pitts (4)

- ¿Qué pasa con funciones como XOR? (cómo separar los puntos blancos de los puntos negros)

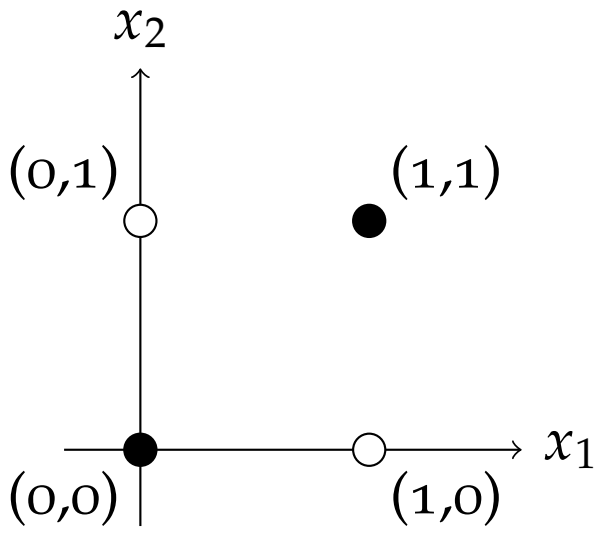
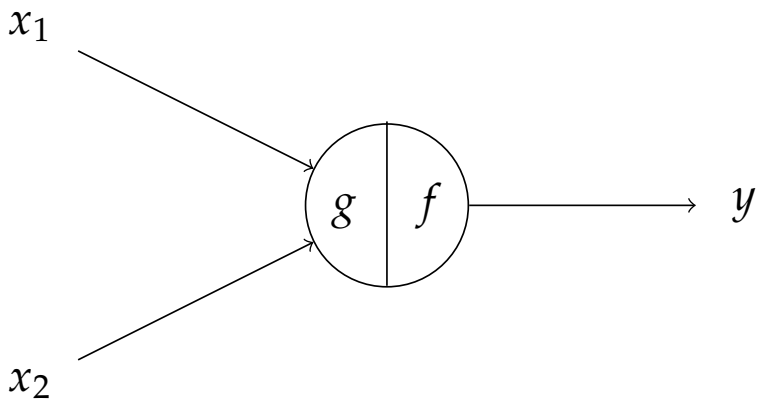



Figura 3: Función XOR

# Neurona de McCulloch y Pitts: Función AND (1)

- Una neurona de función AND solo se activaría cuando TODAS las entradas estén ENCENDIDAS, es decir, g(x) ≥ 2



# Neurona de McCulloch y Pitts: Función AND (2)

A continuación, se muestra una tabla que ilustra la salida $y$ para diferentes combinaciones de $x_1$ y $x_2$:

| $x_1$ | $x_2$ | $y = f(g(x))$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

- g(x) realiza la sumatoria
- $\theta = 2$, es el parámetro de umbral

# Neurona de McCulloch y Pitts: Función AND (1)

- En la Fig. 4 se puede trazar la frontera de decisión para la función AND de acuerdo a las entradas $x_1$ y $x_2$.

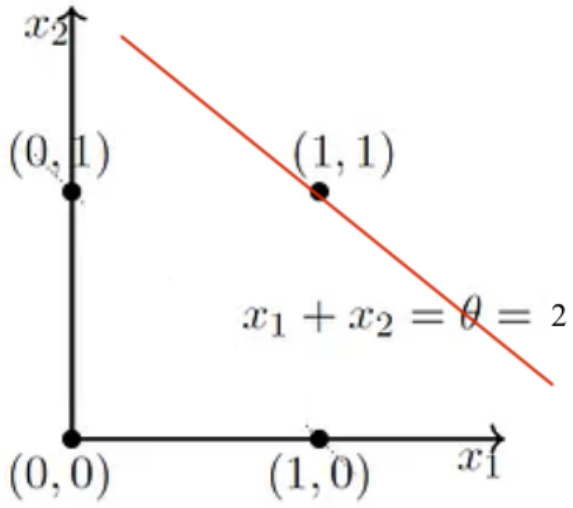



Figura 4: Frontera de decisión de la función AND

# Neurona de McCulloch y Pitts: Función AND (2)

- Se puede ver gráficamente que todas aquellas entradas cuya salida cuando pasa a través de la neurona de la función AND se encuentran EN o POR ENCIMA de esa línea y todos los puntos de entrada que se encuentran DEBAJO de esa línea tendrán una salida de 0.
- La neurona de M-P aprende fronteras de decisión lineal.
- Todas las entradas tienen la misma importancia.
- El parámetro de umbral se debe calcular manualmente.

Función OR: ¿cuál debería ser el valor del parámetro $\theta$ ?

**Consultar ejemplo de función AND:**

Notebook: 03_*McCulloch – Pitts.ipynb*

# Perceptrón

# Modelo: Perceptrón (1)

- Frank Rosenblatt propuso en 1958 el modelo del Perceptrón clásico.
- Se trata de un modelo computacional más generalizado que la neurona McCulloch-Pitts, en el que los pesos y los umbrales se pueden aprender con el tiempo.
- La innovación esencial fue la introducción de pesos numéricos y un patrón especial de interconexión.
- En el modelo original de Rosenblatt, las unidades de computación son elementos de umbral básicamente.
- El aprendizaje se lleva a cabo adaptando los pesos de la red con un algoritmo numérico.

# Modelo: Perceptrón (2)

- El modelo de Rosenblatt fue refinado y perfeccionado en la década de 1960 y sus propiedades computacionales fueron cuidadosamente analizadas por Marvin Minsky y Simmon Papert (1969).
- La interpretación geométrica del procesamiento realizado por los perceptrones es la misma que la de la neurona McCulloch-Pitts.
- Modelo del peceptrón:

# Modelo: Perceptrón (3)

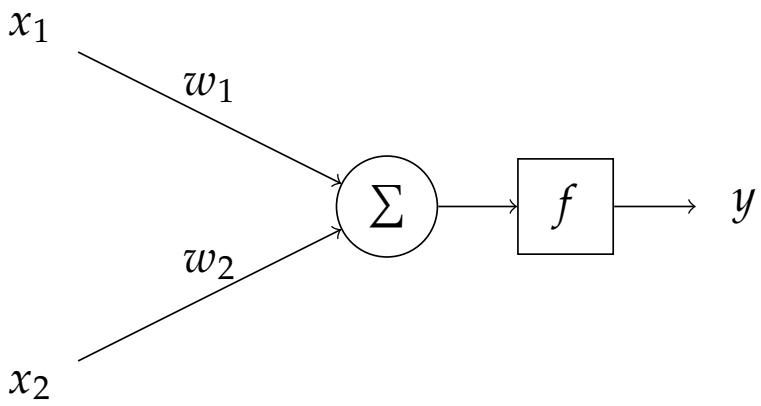



Figura 5: Modelo del perceptrón

# Perceptrón: Estructura (1)

Un **perceptrón** es un tipo de red neuronal de una sola capa (capa de entrada y salida), que es el modelo más simple de red neuronal artificial. Es fundamental para entender redes neuronales más complejas.

- **Entradas** $(x_1, x_2, \ldots, x_n)$: Estos son los valores de entrada que se pasan al perceptrón. Por ejemplo, en una compuerta lógica OR, las entradas pueden ser los valores binarios 0 o 1.
- **Pesos** $(w_1, w_2, \ldots, w_n)$: Cada entrada $x_i$ está asociada con un peso $w_i$. Los pesos determinan la importancia de cada entrada para la salida.

# Perceptrón: Estructura (2)

- **Función de Activación**: Después de calcular la suma ponderada de las entradas, la función de activación se aplica para producir la salida. En el caso más simple, se utiliza la función escalón o signo:

$$f(x) = \begin{cases} 1 & \text{si } x \geq \theta \\ 0 & \text{si } x < \theta \end{cases}$$

- $\theta$ es el umbral que debe superar el valor de $x$
- Si $\theta$ se agrega como una entrada $w_0$ más de la neurona como en la Fig. 6, entonces la función de activación sería la función escalón.

# Perceptrón: Estructura (3)

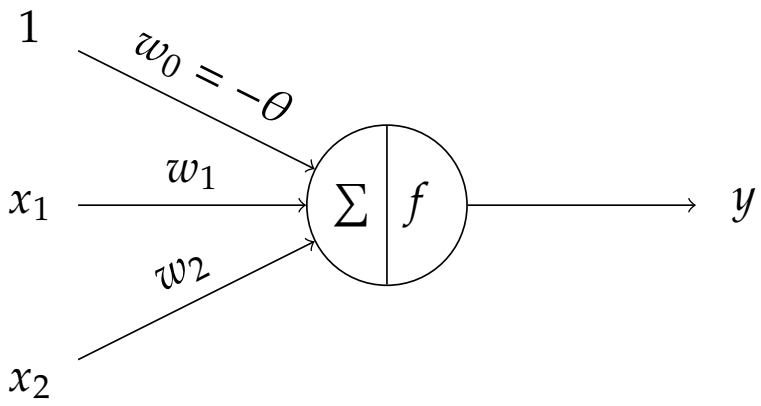



Figura 6: Neurona con bias representado por $w_0$

- Por lo que la función de activación puede ser la función escalón:

$$f(x) = \begin{cases} 1 & \text{si } x \geq 0 \\ 0 & \text{si } x < 0 \end{cases}$$

# Perceptrón: Estructura (4)

- La función que se definía por medio del umbral (Fig. 2) pasaría a ser ahora la función de la Fig. 7

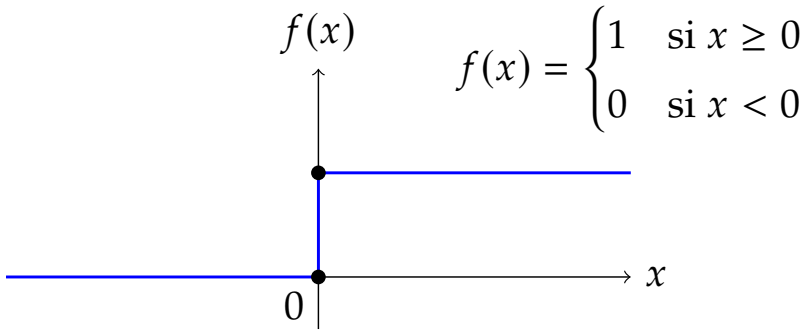



Figura 7: Función Escalón

- Con $x_0 = 1$, resultaría la salida $y$ como sigue:

$$y = f(\sum x_i * w_i) = f(X \cdot W)$$

# Perceptrón: Estructura (5)

- El $bias$ o $sesgo$ también se puede manejar de manera independiente de las entradas, $b = x_0$, la salida $y$ sería como sigue:

$$y = f(\sum x_i * w_i + b) = f(X \cdot W + b)$$

# Operación del Perceptrón

Para una entrada $\mathbf{x} = [x_1, x_2, \ldots, x_n]$, el perceptrón realiza los siguientes pasos:

1. **Calcula la Suma Ponderada**:

$$z = w_1 x_1 + w_2 x_2 + \cdots + w_n x_n + b$$

2. **Aplica la Función de Activación**:
   La salida del perceptrón es:

$$y = f(z)$$

   donde $f(z)$ es la función de activación aplicada a la suma ponderada $z$.

# Entrenamiento del Perceptrón (1)

El entrenamiento del perceptrón se realiza utilizando el algoritmo de aprendizaje del perceptrón. El objetivo es ajustar los pesos y el bias para minimizar el error en las predicciones. El proceso es el siguiente:

1. **Inicialización**: Los pesos y el bias se inicializan (generalmente a cero o a valores pequeños aleatorios).
2. **Iteración**: Para cada muestra de entrenamiento:
   - Calcular la salida del perceptrón $y$
   - Comparar la salida calculada con la etiqueta o valor verdadero $t$.

# Entrenamiento del Perceptrón (2)

- Actualizar los pesos y el bias en función del error: El error se determina calculando las diferencias entre la salida de la neurona y la salida deseada.
  Regla de aprendizaje (Regla delta):

$$\Delta w \leftarrow \eta \cdot (t - y) \cdot x_i$$

$$w_i \leftarrow w_i + \Delta w$$

$$b \leftarrow b + \eta \cdot (t - y)$$

  donde $\eta$ (eta) es la tasa de aprendizaje.

# Entrenamiento del Perceptrón (3)

3. **Repetir**: El proceso se repite durante varias épocas hasta que el perceptrón converge o se alcanza el número máximo de épocas.

# Representación de Neuronas y pesos (1)

- Para la Fig. 8, podemos representar las entradas, neuronas y bias como sigue:

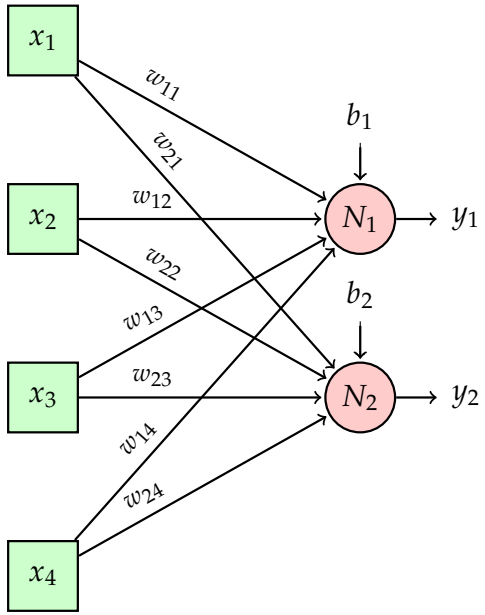



Figura 8: Red Neuronal con una sola capa (capa de salida)

# Representación de Neuronas y pesos (2)

- (Representación 1). Generalmente, el vector X se representa como un vector columna:

$$\mathbf{X} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}$$

- La neurona N1 esta representada por los pesos:

$$N1 = [w_{11} \;\; w_{12} \;\; w_{13} \;\; w_{14}]$$

- En la matriz de pesos, cada renglón representa a una neurona de entrada y sus pesos de entrada.

$$\mathbf{W} = \begin{bmatrix} w_{11} & w_{12} & w_{13} & w_{14} \\ w_{21} & w_{22} & w_{23} & w_{24} \end{bmatrix}$$

# Representación de Neuronas y pesos (3)

- Para calcular la suma pondera ($\sum w_i x_i$) se realiza la operación:

$$\mathbf{W} \cdot \mathbf{X} = \begin{bmatrix} w_{11} & w_{12} & w_{13} & w_{14} \\ w_{21} & w_{22} & w_{23} & w_{24} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}$$

- La representación del $bias$ es un vector columna: $\mathbf{B} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$
- La operación final sería:

$$\mathbf{Z} = \mathbf{W} \cdot \mathbf{X} + \mathbf{B} =$$

$$\begin{bmatrix} w_{11} & w_{12} & w_{13} & w_{14} \\ w_{21} & w_{22} & w_{23} & w_{24} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$$

# Representación de Neuronas y pesos (4)

- (Representación 2). Sin embargo, en los casos prácticos, los ejemplos de entrenamiento (el vector $\mathbf{X}$) se lee como vectores renglón al procesar los datos en un programa:
  $\mathbf{X} = \begin{bmatrix} x_1 & x_2 & x_3 & x_4 \end{bmatrix}$
- Por lo que se prefiere representar los vectores $\mathbf{X}$ en esta forma.
- En el caso de los pesos $\mathbf{W}$ se representan de la misma manera: las filas representan a las neuronas de salida y las columnas representan los pesos de entrada a la neurona.
- Para poder realizar la multiplicación matricial con el vector $\mathbf{X}$ de entrada se obtiene la transpuesta de $\mathbf{W}$ , denotada como $\mathbf{W}^\top$.
- Dada la matriz original:
  $\mathbf{W} = \begin{bmatrix} w_{11} & w_{12} & w_{13} & w_{14} \\ w_{21} & w_{22} & w_{23} & w_{24} \end{bmatrix}$

# Representación de Neuronas y pesos (5)

- La transpuesta $\mathbf{W}^\top$ será:

$$\mathbf{W}^\top = \begin{bmatrix} w_{11} & w_{21} \\ w_{12} & w_{22} \\ w_{13} & w_{23} \\ w_{14} & w_{24} \end{bmatrix}$$

# Representación de Neuronas y pesos (6)

- Para calcular la ponderación de la neurona sería:

$$\mathbf{X} \cdot \mathbf{W}^\top = \begin{bmatrix} x_1 & x_2 & x_3 & x_4 \end{bmatrix} \cdot \begin{bmatrix} w_{11} & w_{21} \\ w_{12} & w_{22} \\ w_{13} & w_{23} \\ w_{14} & w_{24} \end{bmatrix}$$

- La operación final sería:

$$\mathbf{Z} = \mathbf{X} \cdot \mathbf{W}^\top + \mathbf{B} = \begin{bmatrix} x_1 & x_2 & x_3 & x_4 \end{bmatrix} \cdot \begin{bmatrix} w_{11} & w_{21} \\ w_{12} & w_{22} \\ w_{13} & w_{23} \\ w_{14} & w_{24} \end{bmatrix}$$

$$+ \begin{bmatrix} b_1 & b_2 \end{bmatrix}$$

# Ejercicio1: Representación de neuronas y pesos (1)

- Dada la Fig. 9, realizar lo siguiente:
  - 1) Representar algebraicamente (solo indicar la notación) el vector de entrada **X** y la matriz de pesos de las neuronas **W**.
  - 2) Representar las salidas **y**, considerar que no hay función de activación. Representar el cálculo algebraico de la operación entre la matriz de pesos y el vector de entrada.
  - 3) Calcular las salidas **y**. Hacer el cálculo a mano para el vector $X = [1, 2, 5]$ y $b_1 = 1, b_2 = 2$ y considerando los pesos iniciales de la red.
  - 4) Realizar la representación numérica y cálculos en Numpy.

# Ejercicio1: Representación de neuronas y pesos (2)

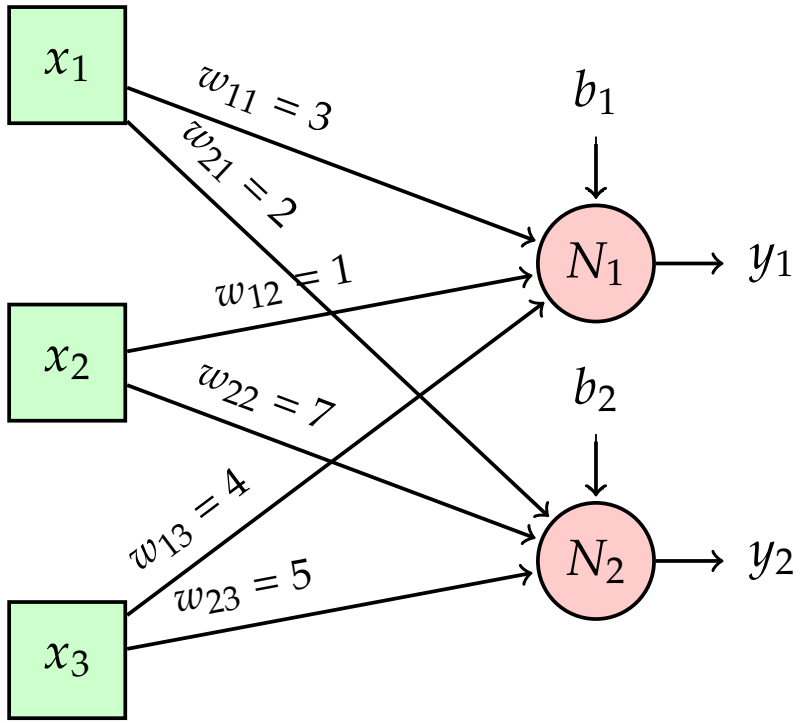



Figura 9: Red Neuronal con una sola capa y 2 neuronas de salida

# Ejercicio2: Representación de neuronas y pesos (1)

- Dada la Fig. 10, realizar lo siguiente:
  - 1) Representar algebraicamente (solo indicar la notación) la matriz de entrada de ejemplos **X** y la matriz de pesos de las neuronas **W**.En este caso la matriz de entrada es la siguiente:

$$\mathbf{X} = \begin{bmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \\ x_{31} & x_{32} \end{bmatrix}$$

  - 2) Representar las salidas **y**, considerar que no hay función de activación. Representar el cálculo algebraico de la operación entre la matriz de pesos y la matriz de entrada como se mostró previamente.

# Ejercicio2: Representación de neuronas y pesos (2)

- 3) Calcular las salidas **y**. Hacer el cálculo a mano para los datos siguientes:

  $$\mathbf{X} = \begin{bmatrix} 1 & 2 \\ 2 & 3 \\ 4 & 1 \end{bmatrix}$$

  y $b_1 = 2, b_2 = 3, b_3 = 2$ y considerando los pesos iniciales de la red.
- 4) Realizar la representación numérica y cálculos en Numpy.

# Ejercicio2: Representación de neuronas y pesos (3)

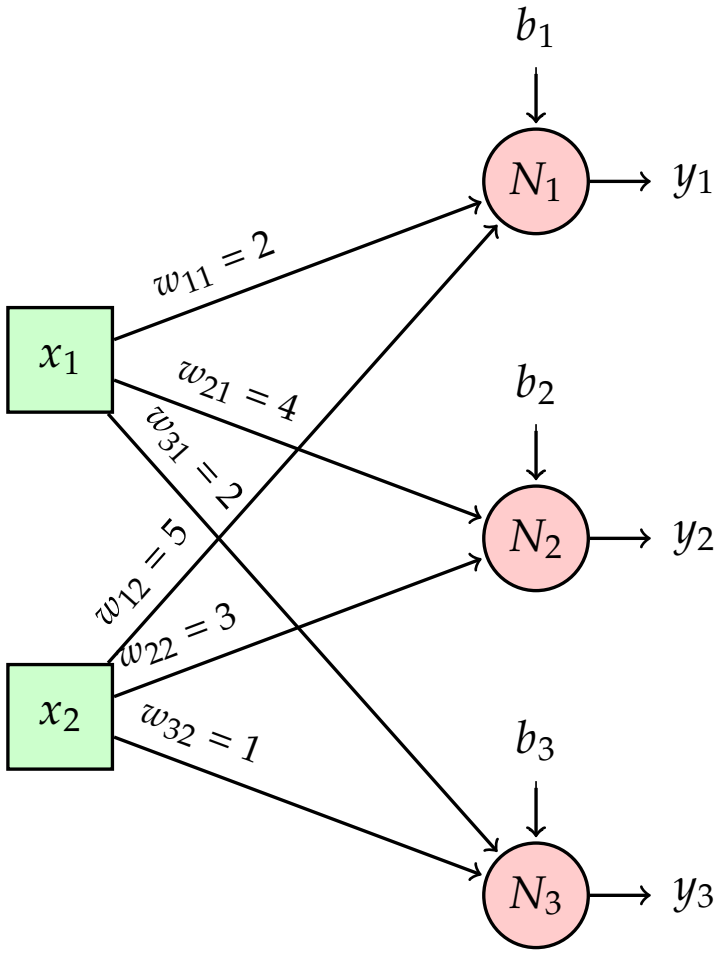



Figura 10: Red Neuronal con una sola capa y 3 neuronas de salida

# Ejercicio3: Implementar Perceptrones (1)

- Implementar en Python y Numpy la siguiente estructura de perceptrones (Fig. 12).
- Los datos de entrenamiento son números codificados en binario $X = [1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1]$, estos datos se pueden visualizar como imagen de 5x3. El número sería como el de la Fig. 11.

Figura 11: Vector X representado como imagen de 5x3 pixeles

# Ejercicio3: Implementar Perceptrones (2)

- La salida está codificada en binario que representa al número.

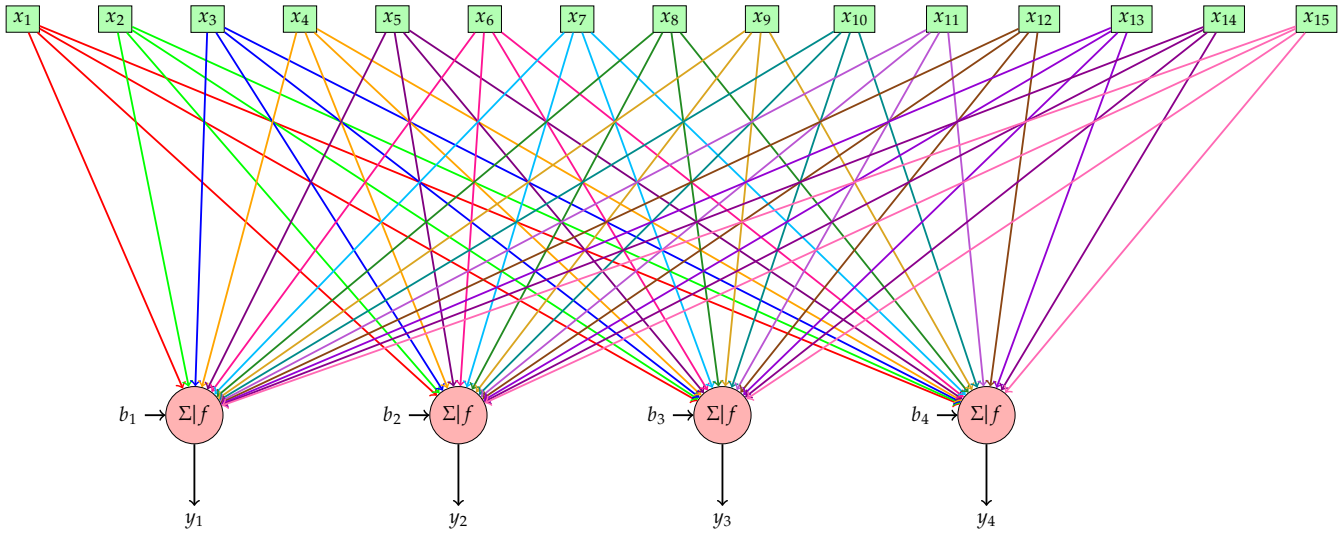



Figura 12: Red Neuronal con una sola capa: 4 neuronas de salida

## Ejercicio3: Programación de un Perceptrón

Consultar el Notebook 04_perceptronTODO.ipynb

# Ejemplo: Compuerta OR

Para una compuerta OR, el perceptrón se puede definir con los siguientes parámetros:

- **Pesos**: $w_1 = 1, w_2 = 1$
- **Bias**: $b = -0.5$

La fórmula del perceptrón para la compuerta OR es:

$$y = f(w_1 x_1 + w_2 x_2 + b)$$

Donde:

$$f(z) = \begin{cases} 1 & \text{si } z \geq 0 \\ 0 & \text{si } z < 0 \end{cases}$$

# Desventajas

- Minsky y Papert demostraron que existen problemas que no pueden resolverse mediante un único perceptrón que actúe como última unidad de decisión. Por ejemplo, como la función XOR que no es linealmente separable.

# Perceptrón para la solución de XOR (1)

- Una arquitectura de Perceptron multinivel una capa oculta con dos neuronas y la respectiva neurona de la capa de salida, como se ilustra en la Fig 13 con adecuado entrenamiento puede resolver el problema de separación no lineal planteado por la función XOR.

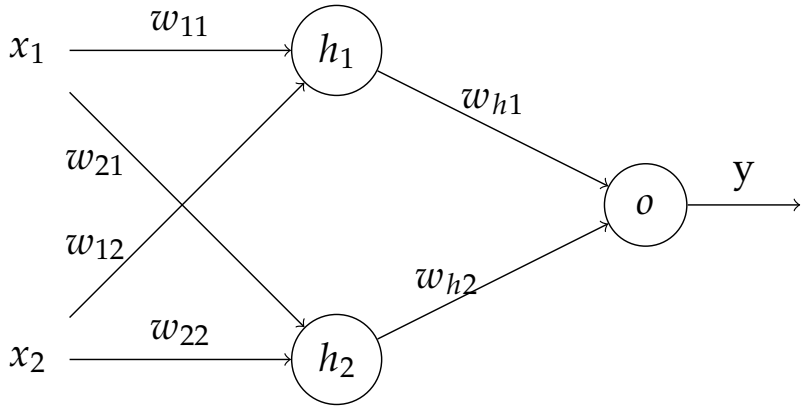



Figura 13: Modelo para la función XOR

# Perceptrón para la solución de XOR (2)

- En ese entonces se tenían algunos interrogantes sin respuesta
- ¿Cómo entrenar un Perceptrón multicapa?
- ¿Cómo evaluar el error en las capas ocultas si no hay un valor deseado conocido para las salidas de estas capas?
- Este inconveniente es inherente a la forma como opera el algoritmo de aprendizaje del Perceptrón, para variar los pesos, lo hace con base en el error de salida, se necesita conocer la salida deseada.
- En el Perceptrón sin capas ocultas, este problema no existe, el algoritmo de aprendizaje del Perceptrón del tipo supervisado, las salidas de la red están definidas.

# Perceptrón para la solución de XOR (3)

- Si el Perceptrón es multicapa no es posible conocer con anterioridad el valor de la salida de una de las capas ocultas. No se puede aplicar el algoritmo de aprendizaje del Perceptrón.
- Una solución a este problema la planteó formalmente, y por primera vez, Paul Werbos (1974), y se denominó algoritmo de backpropagation.

# Referencias (1)


1. Deep Learning. Ian Goodfellow, Yoshua Bengio, and Aaron Courville. MIT Press, 2016. http://www.deeplearningbook.org
2. Dive into Deep Learning. Aston Zhang, Zachary C. Lipton, Mu li, and Alexander J. Smola. Cambridge University Press, 2023. https://d2l.ai
3. Neural Networks and Deep Learning A Textbook (2nd Edition). Charu C. Aggarwal. Springer, 2023. https://doi.org/10.1007/978-3-031-29642-0
4. Deep Learning: Foundations and Concepts. Christopher M. Bishop and Hugh Bishop. Springer, 2024. https://doi.org/10.1007/978-3-031-45468-4

# Referencias (2)


5. PyTorch documentation.
   `https://pytorch.org`
6. Numpy documentation.
   `https://numpy.org`
7. Python documentation.
   `https://www.python.org`