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

Redes Neuronales

Sabino Miranda

Redes Multicapa

Redes Multicapa (1)

- Las redes multicapa o redes prealimentadas o *Multilayer Perceptron* (MLP) o *Feedforward Neural Networks* (FFN) usan un algoritmo de entrenamiento basado en la propagación hacia atrás del error; conocido como *backpropagation* o *backprop*.
- Las redes multicapa tienen una capa de entrada, múltiples capas ocultas y una capa de salida.
- Se pueden ver como una función f que, a partir de un vector de entrada X, obtiene un vector de salida $y = f(x)$.

Redes Multicapa (2)

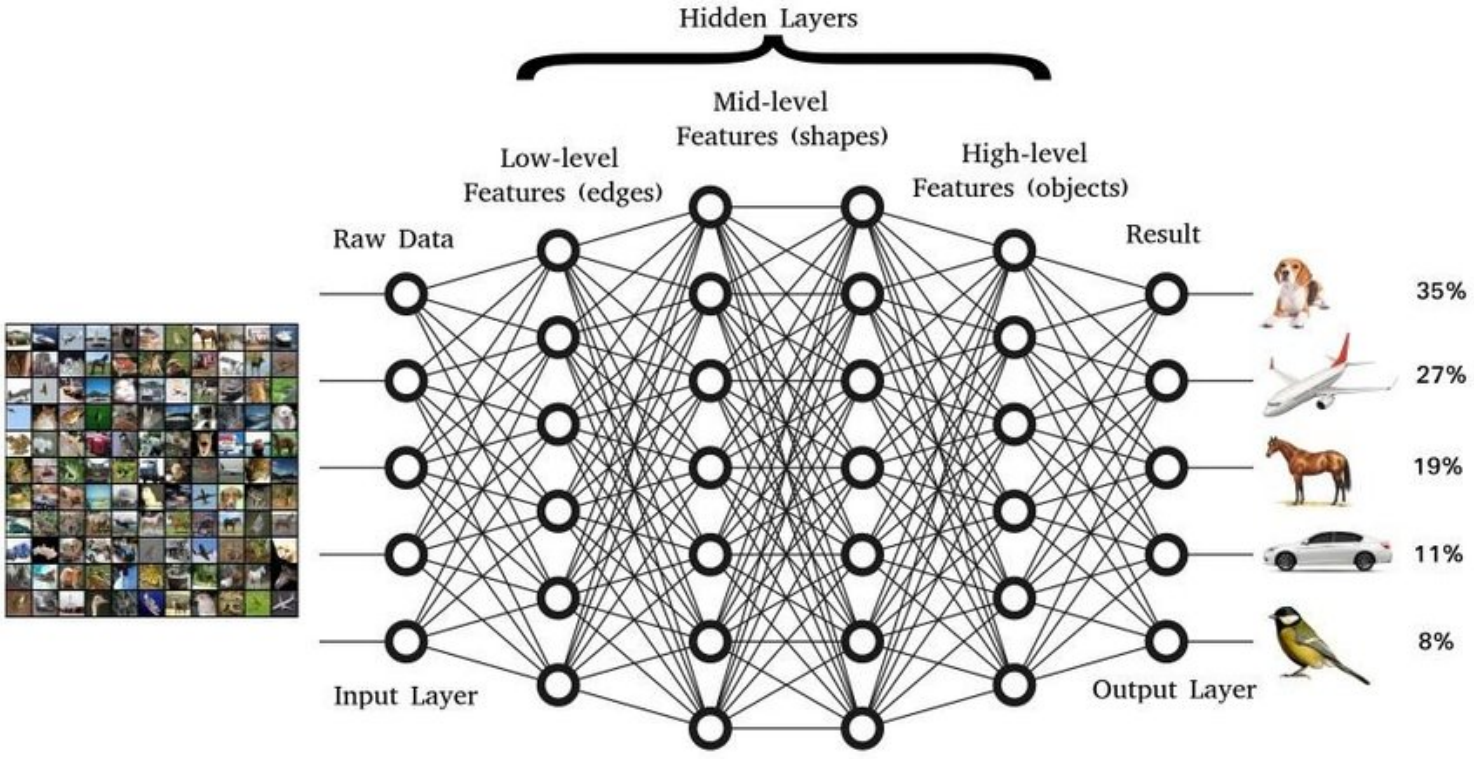

Figura 1: Red Neuronal

Fuente: https://doi.org/10.1051/matecconf/201816401015

Redes Multicapa

- Dado un conjunto de entrenamiento en forma de pares (X, T) (ejemplos, clases):
  - El objetivo del algoritmo de entrenamiento es aproximar la función f para cada posible entrada (ejemplo de entrenamiento) $x_i \in X$.
  - Obtener una salida $\hat{y} = f(x_i)$ lo más similar posible a la observada t_i en el conjunto de entrenamiento: $t_i \in T$.
  - Ajustar los parámetros de la red neuronal (hiperparámetros) para que sea capaz de generalizar correctamente, para entradas no vistas y las clasifique correctamente.
  - El objetivo es el reducir el error en el conjunto de pruebas sin aumentar el error en el entrenamiento.

Redes multicapa: entrenamiento (1)

- ¿Cómo podemos entrenar una red multicapa?
- ¿Cómo ajustar los pesos sinápticos de las neuronas de las capas ocultas de una red multicapa?
- Se requiere de un algoritmo eficiente que permita adaptar todos los pesos de una red multicapa, tanto de las capas ocultas como de la capa de salida.
- En las capas ocultas, se desconocen los valores esperados (valores correctos) de las salidas de las capas ocultas (para aplicar la regla de aprendizaje del perceptrón)
- La estrategia es fijarnos en los **cambios que se producen** en los niveles de activación de las neuronas ocultas si se modifican sus pesos.

Redes multicapa: entrenamiento (2)

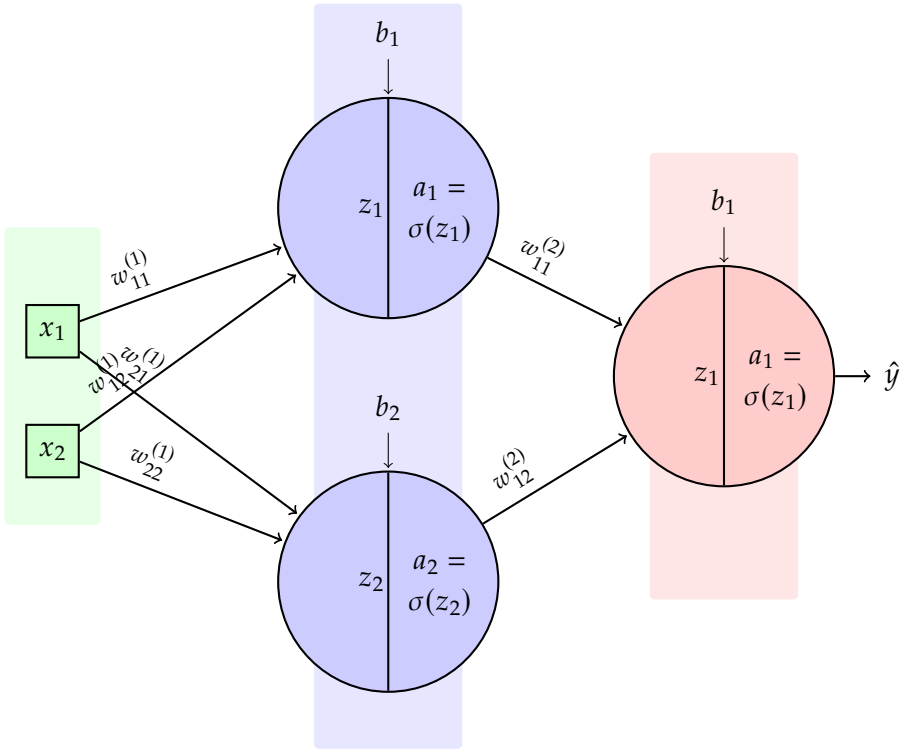

Figura 2: Modelo de una red multicapa: una capa de entrada, una capa oculta y una capa de salida.

Redes multicapa: entrenamiento (3)

- Por ejemplo, para la capa 1, el cálculo de la $z_1^{(1)}$ y $a_1^{(1)}$:

$$z_1^{(1)} = \sum_i^K w_{1_i}^{(1)} x_i + b_1^{(1)}$$

$$a_1^{(1)} = \sigma(z_1)$$

- Por ejemplo, para la capa 2:

$$z_1^{(2)} = \sum_i^K w_{1_i}^{(2)} a_i^{(1)} + b_1^{(2)}$$

$$\hat{y} = \sigma(z_1^{(2)})$$

Entrenando un MLP

- Entrenar los pesos de un MLP (y por extensión una red neuronal, Fig. 9) consiste básicamente en 4 pasos:
  1. Propagación hacia adelante(**Forward Propagation**): Calcular la salida de la red para un ejemplo de entrada.
  2. Cálculo del Error (**Error Computation**): Calcular el error de predicción entre la predicción de la red ($\hat{y}$) y la salida objetivo (t también llamada y).
  3. Propagación hacia atrás o retropropagación (**Backpropagation**): Calcular los gradientes en orden inverso de la función de pérdida con respecto a los pesos y sesgos.
  4. Actualización de parámetros (**Parameter update**): Se usa el descenso de gradiente estocástico para actualizar los pesos de la red y reducir el error para el ejemplo procesado.

1. Forward Propagation (1)

- El primer paso para entrenar un MLP es calcular la salida de la red para un ejemplo del conjunto de datos de entrenamiento.
- Existen diferente funciones de activación. Por ejemplo, la función sigmoide (Fig. 3), $\sigma(x)$, como función de activación. Se puede pensar como la función escalón suavizada.

1. Forward Propagation (2)

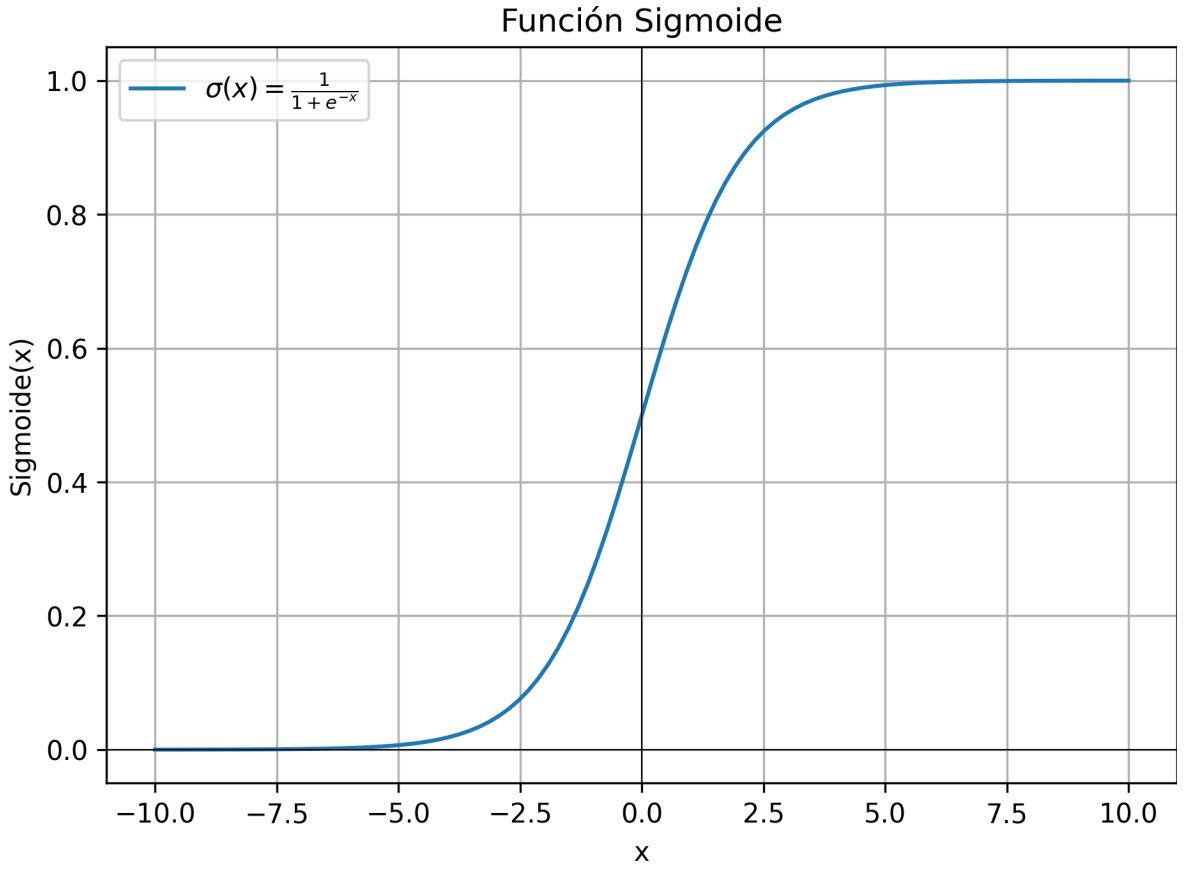

Figura 3: Función de activación Sigmoide: $\sigma(x) = \frac{1}{1+e^{-x}}$

1. Forward Propagation (3)

- Otras funciones de activación son:

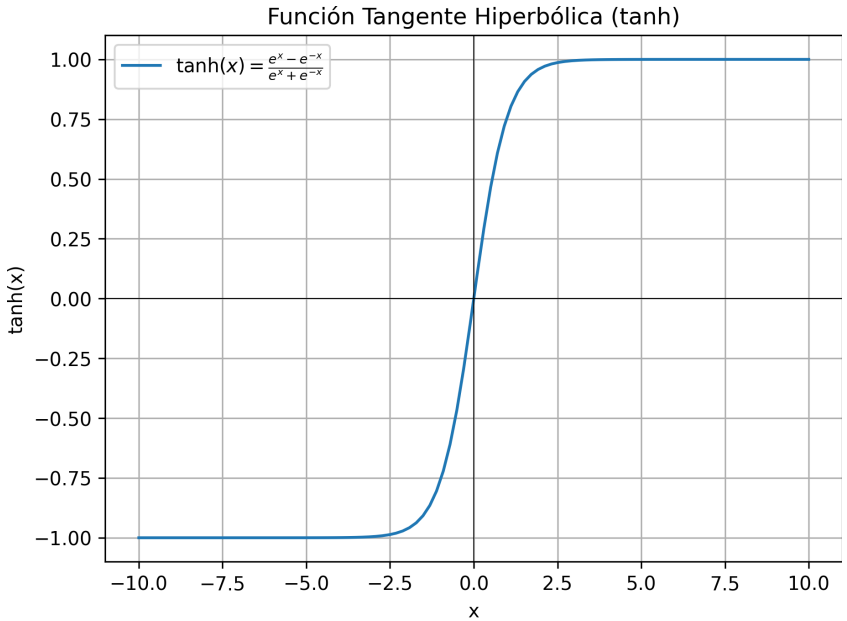

Figura 4: Función de activación tangente hiperbólica: $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$

1. Forward Propagation (4)

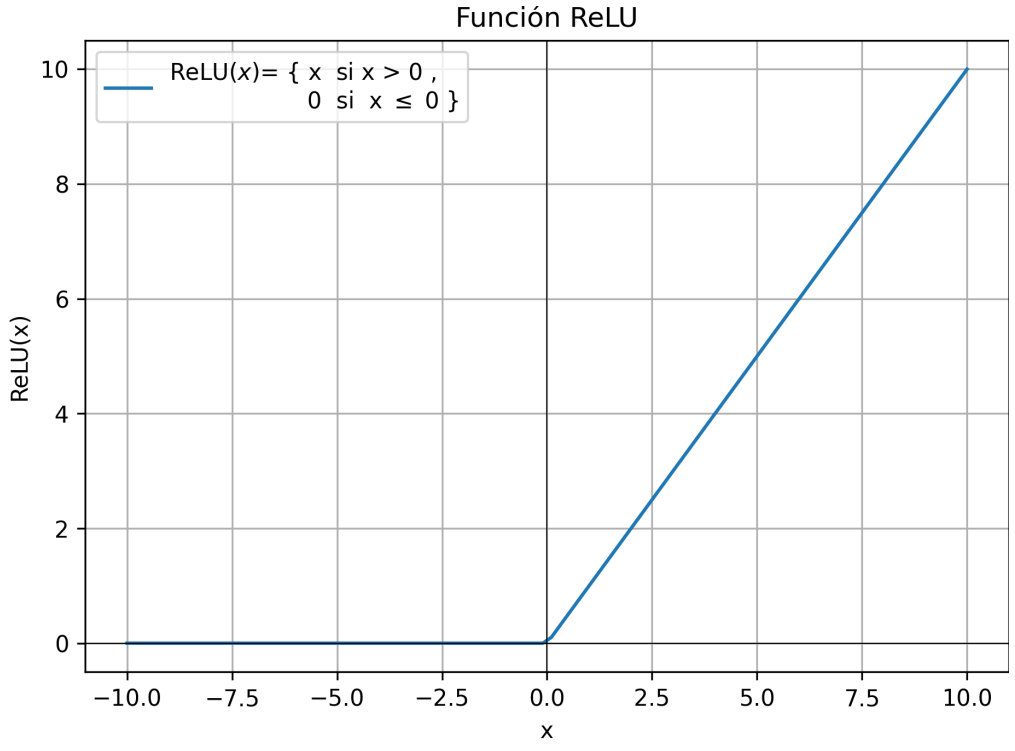

Figura 5: Función de activación Rectified Linear Unit (ReLU):

$$\text{ReLU}(x) = \begin{cases} x & \text{si } x > 0 \\ 0 & \text{si } x \leq 0 \end{cases}$$

1. Forward Propagation (5)

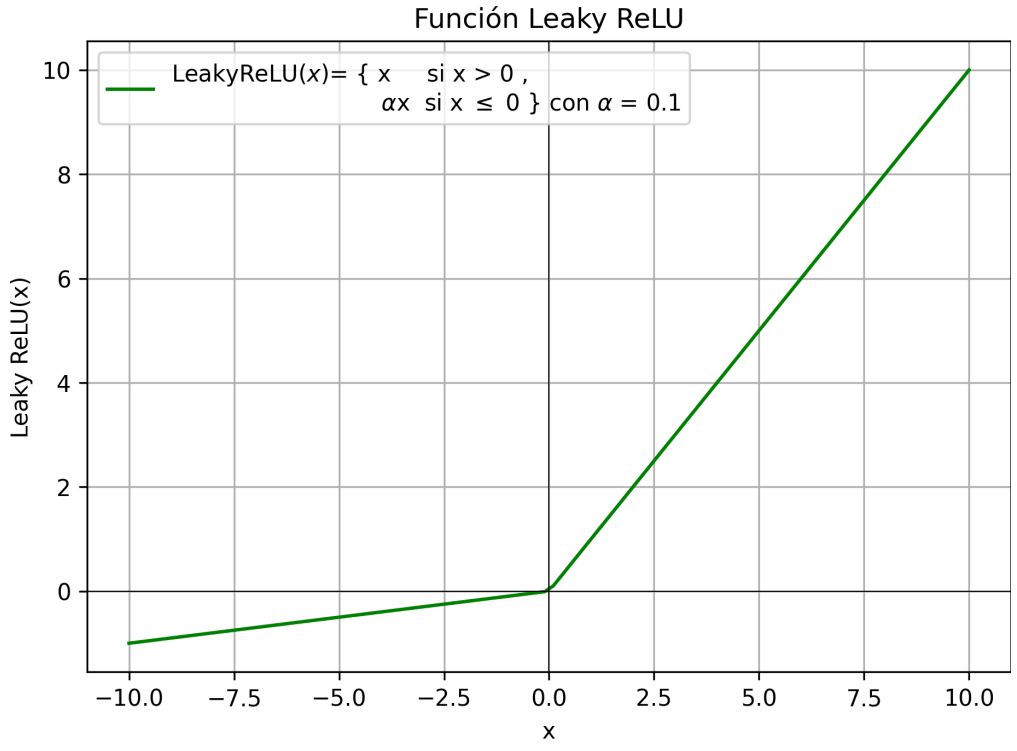

Figura 6: Función de activación Leaky ReLU:

$$\text{Leaky ReLU}(x) = \begin{cases} x & \text{si } x > 0 \\ \alpha x & \text{si } x \leq 0 \end{cases}$$

1. Forward Propagation (6)

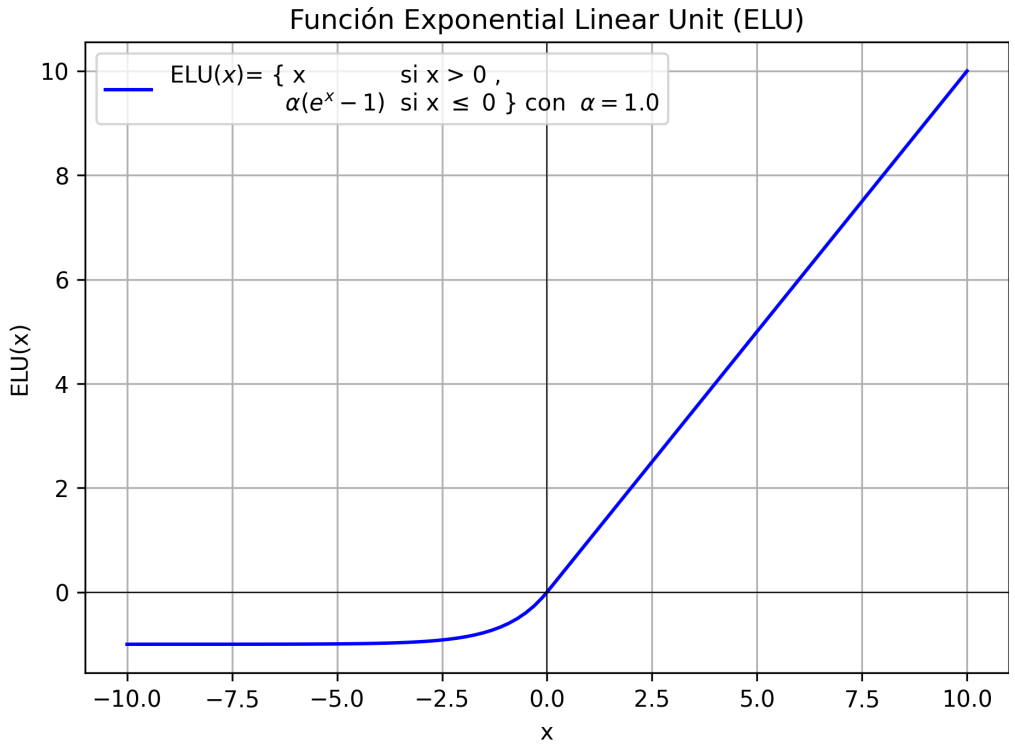

Figura 7: Función de activación Exponential Linear Unit (ELU):

$$\mathrm{ELU}(x) = \begin{cases} x & \text{si } x > 0 \\ \alpha\,(e^{x} - 1) & \text{si } x \leq 0 \end{cases}$$

1. Forward Propagation (7)

- La función sigmoide es continuamente diferenciable. Propiedad deseable para el método backpropagation basado en el gradiente.
- El objetivo de este proceso es calcular la salida de la red actual para un ejemplo particular x, con cada salida conectada como entrada de la siguiente capa de neuronas.
- La red neuronal de una capa oculta con neuronas $h1$ y $h2$ y una neurona de salida o se puede representar como sigue:

1. Forward Propagation (8)

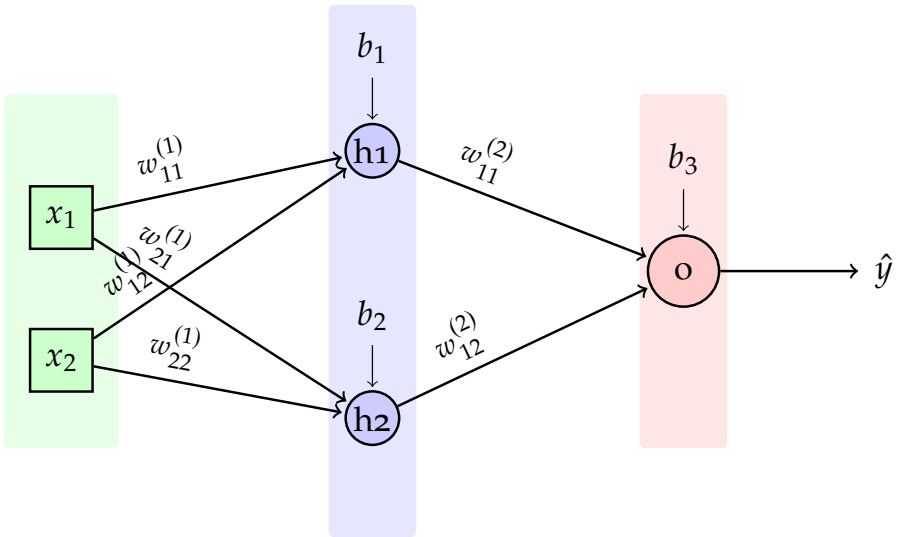

Figura 8: Modelo de una red multicapa: una capa de entrada, una capa oculta y una capa de salida.

1. Forward Propagation (9)

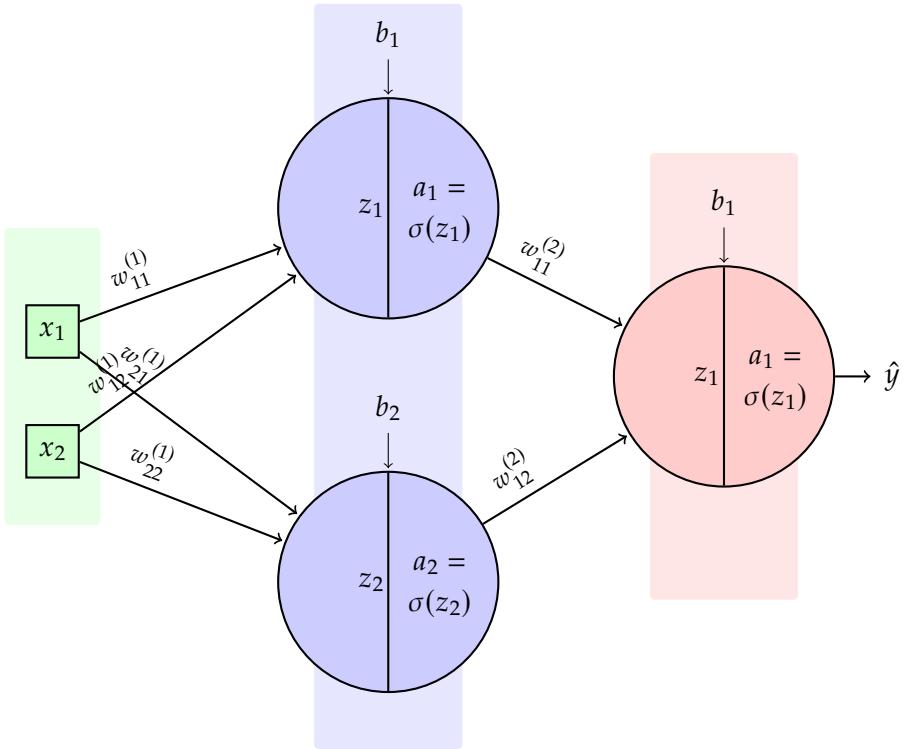

Figura 9: Modelo de una red multicapa: una capa de entrada, una capa oculta y una capa de salida.

1. Forward Propagation (10)

1. La función de activación:

$$f(v) = \sigma(v)$$

2. Cálculo de la salida de las neuronas en la capa oculta, por ejemplo h_1:

$$z_1^{(1)} = \sum w_{1_i}^{(1)} x_i + b_1$$

$$a_1^{(1)} = f(z_1)$$

$$a_1^{(1)} = f(\sum w_{1_i}^{(1)} x_i + b_1)$$

1. Forward Propagation (11)

3. Cálculo de la neurona de salida o:

$$z_1^{(2)} = \sum w_{1_i}^{(2)} a_i^{(1)} + b_1^{(2)}$$

$$\hat{y} = f(z_1^{(2)})$$

$$\hat{y} = f(\sum w_{1_i}^{(2)} a_i^{(1)} + b_1^{(2)})$$

donde:

- w_{ki} son los pesos que entran a cada neurona
- b_i son los sesgos de cada neurona
- z_i son los suma ponderada y bias
- f es la función de activación
- a_i son las salidas de las neuronas de las capas ocultas (también conocidas como activaciones)

1. Forward Propagation (12)

- Al final del paso de propagación hacia adelante, se tienen la predicción de salida de la red.

1. Forward Propagation (13)

Ejercicio. Aplicar la propagación hacia adelante al problema XOR con una arquitectura de red MLP, ver Fig. 10

Consultar el Notebook 05_MLP_XOR.ipynb

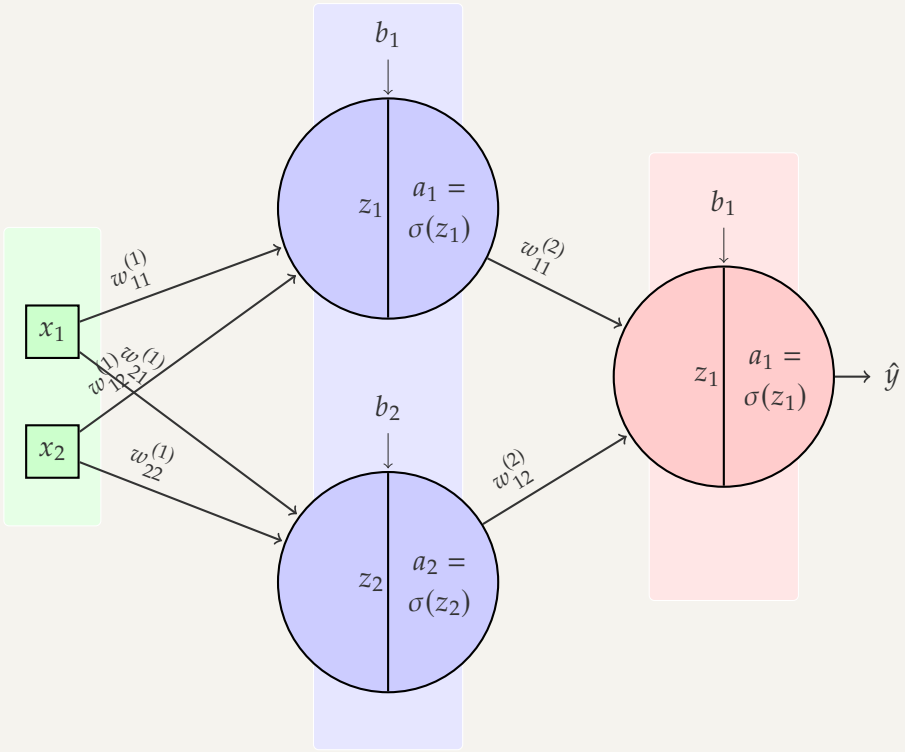

Figura 10: Modelo de red multicapa para el problema XOR.

2. Cálculo del Error (1)

- En este paso, se verifica el desempeño de la red con el ejemplo x proporcionado.
- Se usa una función de pérdida (*loss function*), función de costo (*cost fuction*) o función de error para evaluar el rendimiento del modelo. Por ejemplo, el error cuadrático medio (*mean squared error, MSE*).
- Para N ejemplos, el error cuadrático medio se calcula como:

$$\text{MSE} = \frac{1}{2N} \sum_{i=1}^{N} (\hat{y}_i - y_i)^2$$

donde:

- N es el número total de ejemplos

2. Cálculo del Error (2)

- y_i es el valor real (esperado) para el i-ésimo ejemplo
- $\hat{y}_i$ es la predicción del modelo para el i-ésimo ejemplo.

- Para un único ejemplo, el error cuadrático medio se calcula como:

$$\text{MSE}_{\text{1-ejemplo}} = \frac{1}{2}(\hat{y} - y)^2$$

- El factor $\frac{1}{2}$ se usa para simplificar el cálculo del gradiente.
- La función cuadrática obliga a que el error no sea negativo.
- Debido a la función cuadrática, los errores más pequeños tienen una penalización menos severa comparado con los errores más grandes.

2. Cálculo del Error (3)

- La función MSE penaliza más severamente los errores grandes que los errores pequeños, de manera que el error total es más sensible a los errores grandes.

2. Cálculo del Error (4)

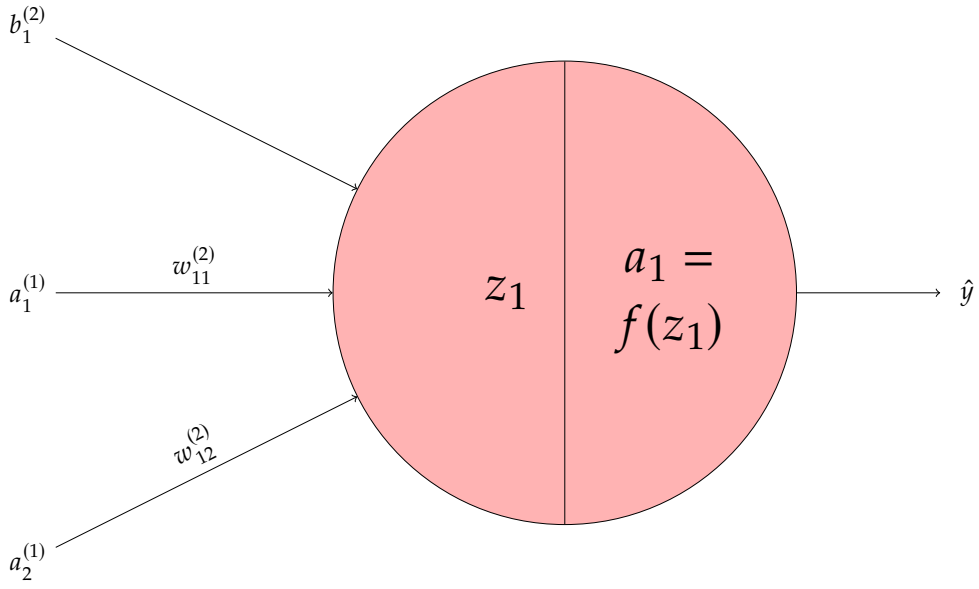

Figura 11: Neurona de salida de la Fig. 9 con el cálculo de la pre-activación z_1 y post-activación $\hat{y} = a_1$

2. Cálculo del Error (5)

- Pre-activación (z_1) y post-activación ($\hat{y} = a_1$).

$$z_1 = w_{11}^{(2)} a_1^{(1)} + w_{12}^{(2)} a_2^{(2)} + b_1^{(2)}$$

$$\hat{y} = a_1 = f(z_1) = \sigma(z_1) = \frac{1}{1 + e^{-z_1}}$$

$$\hat{y} = f(z_1) = \sigma(z_1) = \frac{1}{1 + e^{-(w_{11}^{(2)} a_1^{(1)} + w_{12}^{(2)} a_2^{(1)} + b_1^{(2)})}}$$

$$\mathrm{E}(\hat{y}, y) = \frac{1}{2}(\hat{y} - y)^2$$

$$\mathrm{E}(\hat{y}, y) = \frac{1}{2}(\frac{1}{1 + e^{-(w_{11}^{(2)} a_1^{(1)} + w_{12}^{(2)} a_2^{(1)} + b_1^{(2)})}} - y)^2$$

Donde, E representa el Error.

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`