Clase 16 - Redes Neuronales I

Aprendizaje de Máquinas - MA5204

Felipe Tobar

Department of Mathematical Engineering &
Center for Mathematical Modelling
Universidad de Chile

12 de marzo de 2021

UNIVERSIDAD
DE CHILE

Contenido

Redes Neuronales - Introducción y Arquitectura

Redes Neuronales - Introducción

Una *Red Neuronal* es un modelo computacional basado en la conexión de múltiples unidades (neuronas) cuyo objetivo es aproximar una función f^*.

Su principal ventaja radica en la posibilidad de entrenarla con datos crudos, es decir, no se presentan las características relevantes de la data y en vez, se permite que el algoritmo aprenda cuales son.

Los modelos escenciales de redes neuronales se conocen como **feedforward neural networks**, o **multilayer perceptrons** (MLPs), Una red feedforward define un mapping $\boldsymbol{y} = f(\boldsymbol{x}; \boldsymbol{\theta})$ y aprende los parámetros $\boldsymbol{\theta}$ que resultan en la mejor aproximación posible.

Estos modelos se conocen como redes ya que son típicamente el resultado de composiciones sucesivas de varios tipos de funciones $f^{(1)}$, $f^{(2)}$, $f^{(3)}, \ldots$.

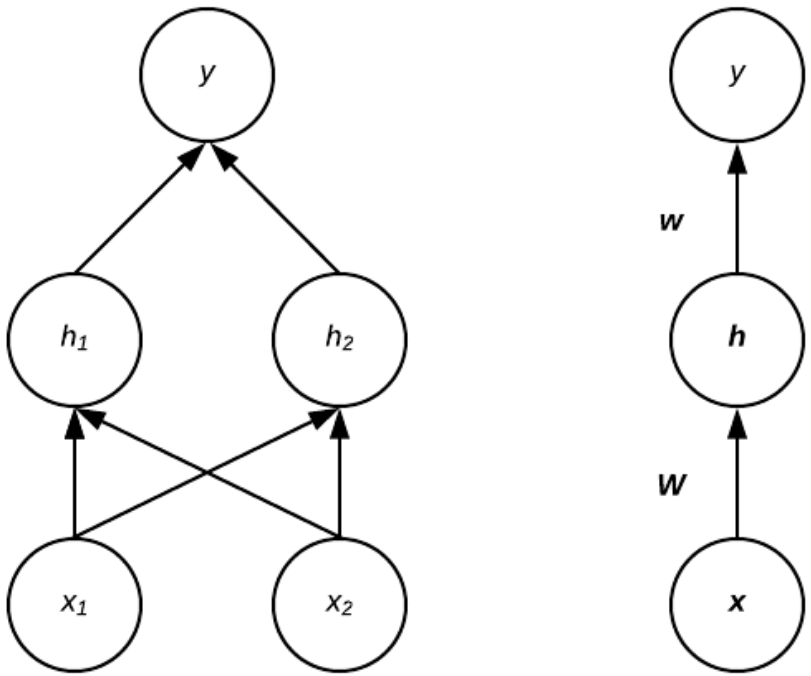

Fig.. Ejemplo de una red neuronal de 1 capa: (*izquierda*). Representación vectorial de las capas. (*derecha*)

Redes Neuronales - Introducción

Una *Red Neuronal* es un modelo computacional basado en la conexión de múltiples unidades (neuronas) cuyo objetivo es aproximar una función f^*.

Su principal ventaja radica en la posibilidad de entrenarla con datos crudos, es decir, no se presentan las características relevantes de la data y en vez, se permite que el algoritmo aprenda cuales son.

Los modelos escenciales de redes neuronales se conocen como **feedforward neural networks**, o **multilayer perceptrons** (MLPs), Una red feedforward define un mapping $\boldsymbol{y} = f(\boldsymbol{x}; \boldsymbol{\theta})$ y aprende los parámetros $\boldsymbol{\theta}$ que resultan en la mejor aproximación posible.

Estos modelos se conocen como redes ya que son típicamente el resultado de composiciones sucesivas de varios tipos de funciones $f^{(1)}$, $f^{(2)}$, $f^{(3)}, \ldots$.

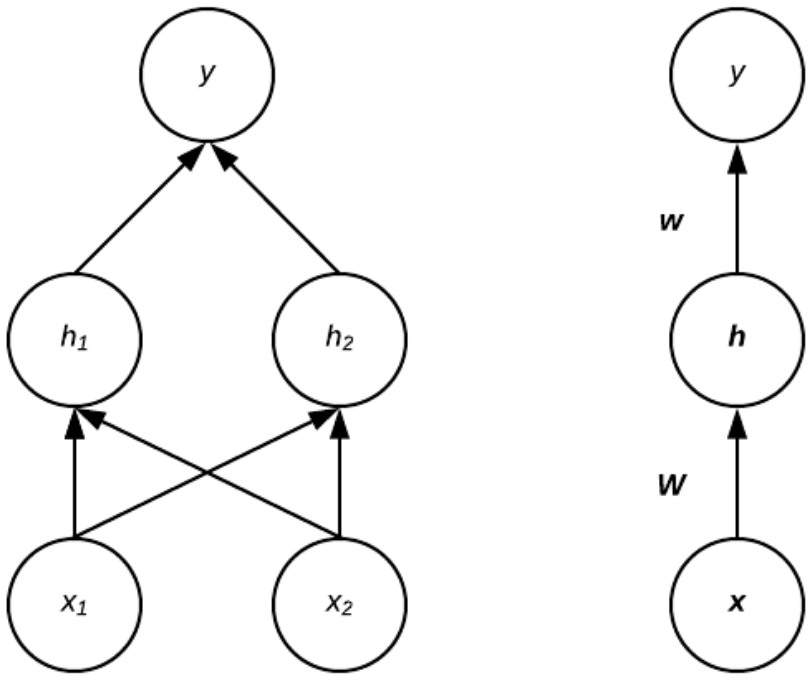

Fig.. Ejemplo de una red neuronal de 1 capa: (*izquierda*). Representación vectorial de las capas. (*derecha*)

Redes Neuronales - Introducción

Una *Red Neuronal* es un modelo computacional basado en la conexión de múltiples unidades (neuronas) cuyo objetivo es aproximar una función f^*.

Su principal ventaja radica en la posibilidad de entrenarla con datos crudos, es decir, no se presentan las características relevantes de la data y en vez, se permite que el algoritmo aprenda cuales son.

Los modelos escenciales de redes neuronales se conocen como **feedforward neural networks**, o **multilayer perceptrons** (MLPs), Una red feedforward define un mapping $\boldsymbol{y} = f(\boldsymbol{x}; \boldsymbol{\theta})$ y aprende los parámetros $\boldsymbol{\theta}$ que resultan en la mejor aproximación posible.

Estos modelos se conocen como redes ya que son típicamente el resultado de composiciones sucesivas de varios tipos de funciones $f^{(1)}$, $f^{(2)}$, $f^{(3)}, \ldots$.

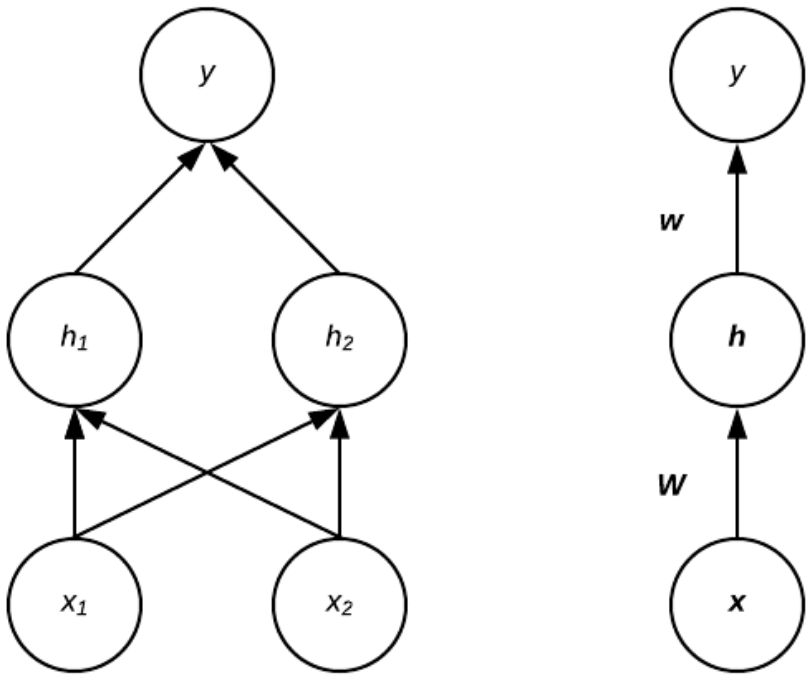

Fig.. Ejemplo de una red neuronal de 1 capa: (*izquierda*). Representación vectorial de las capas. (*derecha*)

Redes Neuronales - Introducción

Una *Red Neuronal* es un modelo computacional basado en la conexión de múltiples unidades (neuronas) cuyo objetivo es aproximar una función f^*.

Su principal ventaja radica en la posibilidad de entrenarla con datos crudos, es decir, no se presentan las características relevantes de la data y en vez, se permite que el algoritmo aprenda cuales son.

Los modelos escenciales de redes neuronales se conocen como **feedforward neural networks**, o **multilayer perceptrons** (MLPs), Una red feedforward define un mapping $\boldsymbol{y} = f(\boldsymbol{x}; \boldsymbol{\theta})$ y aprende los parámetros $\boldsymbol{\theta}$ que resultan en la mejor aproximación posible.

Estos modelos se conocen como redes ya que son típicamente el resultado de composiciones sucesivas de varios tipos de funciones $f^{(1)}$, $f^{(2)}$, $f^{(3)}, \ldots$.

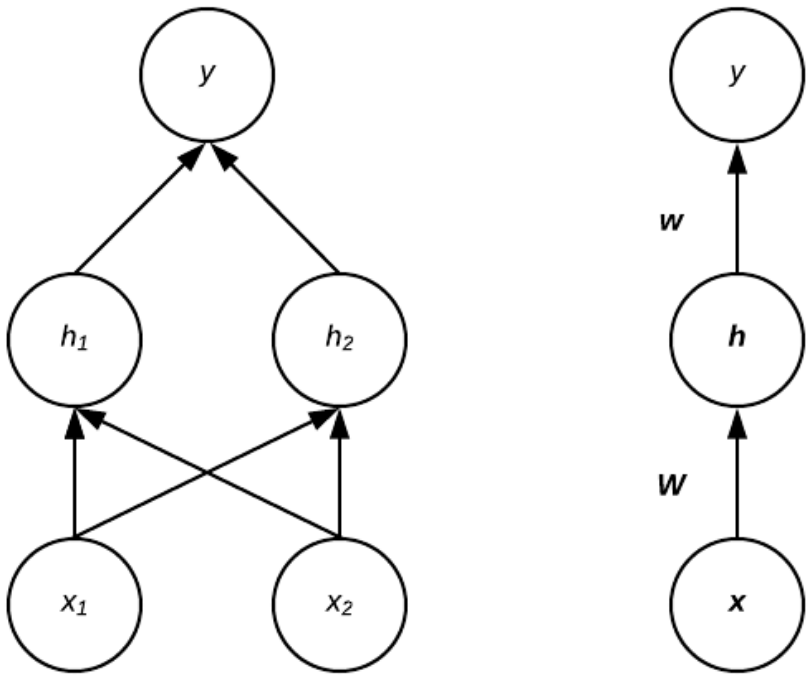

Fig.. Ejemplo de una red neuronal de 1 capa: (*izquierda*). Representación vectorial de las capas. (*derecha*)

Redes Neuronales - Introducción

Una *Red Neuronal* es un modelo computacional basado en la conexión de múltiples unidades (neuronas) cuyo objetivo es aproximar una función f^*.

Su principal ventaja radica en la posibilidad de entrenarla con datos crudos, es decir, no se presentan las características relevantes de la data y en vez, se permite que el algoritmo aprenda cuales son.

Los modelos escenciales de redes neuronales se conocen como **feedforward neural networks**, o **multilayer perceptrons** (MLPs), Una red feedforward define un mapping $\boldsymbol{y} = f(\boldsymbol{x}; \boldsymbol{\theta})$ y aprende los parámetros $\boldsymbol{\theta}$ que resultan en la mejor aproximación posible.

Estos modelos se conocen como redes ya que son típicamente el resultado de composiciones sucesivas de varios tipos de funciones $f^{(1)}$, $f^{(2)}$, $f^{(3)}, \ldots$.

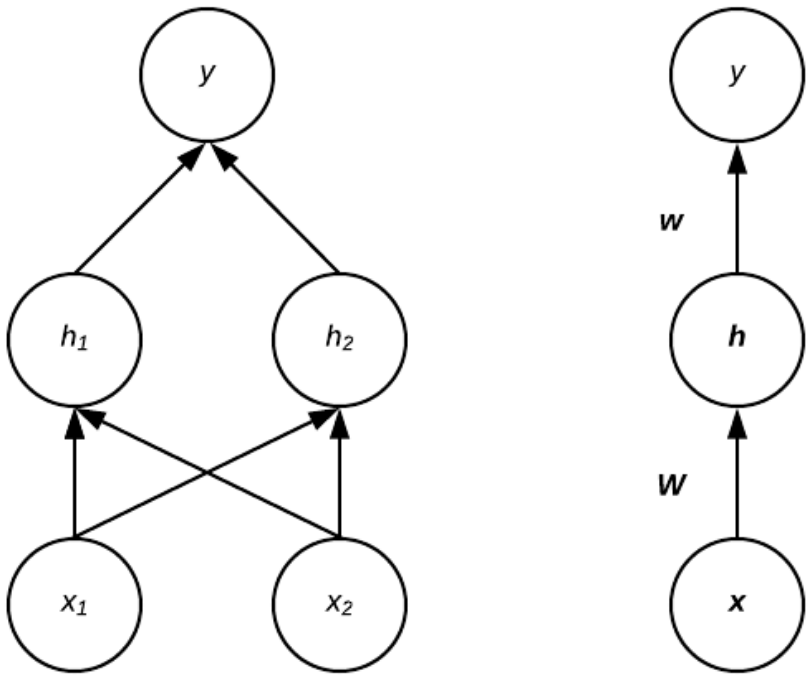

Fig.. Ejemplo de una red neuronal de 1 capa: (*izquierda*). Representación vectorial de las capas. (*derecha*)

Redes Neuronales - El perceptrón

El *perceptrón* corresponde a la forma más básica de una red neuronal, esta recibe un input numérico $x = (x_i)_{i=1}^n \in \mathbb{R}^n$ y computa la suma ponderada $u = x_1 w_1 + x_2 w_2 + \cdots + w_n x_n + b$ donde $W = (w_i)_{i=1}^n \in \mathbb{R}^n$ corresponden a los **pesos** (weights) y $b \in \mathbb{R}$ el **sesgo** (bias).

A continuación, se aplica una función de activación f y se entrega un output $h = f(u)$.

Fig.. Algunos ejemplos de funciones de activación

Redes Neuronales - El perceptrón

El *perceptrón* corresponde a la forma más básica de una red neuronal, esta recibe un input numérico $x = (x_i)_{i=1}^n \in \mathbb{R}^n$ y computa la suma ponderada $u = x_1 w_1 + x_2 w_2 + \cdots + w_n x_n + b$ donde $W = (w_i)_{i=1}^n \in \mathbb{R}^n$ corresponden a los **pesos** (weights) y $b \in \mathbb{R}$ el **sesgo** (bias).
A continuación, se aplica una función de activación f y se entrega un output $h = f(u)$.

Fig.. Algunos ejemplos de funciones de activación

Redes Neuronales - El perceptrón

El *perceptrón* corresponde a la forma más básica de una red neuronal, esta recibe un input numérico $x = (x_i)_{i=1}^n \in \mathbb{R}^n$ y computa la suma ponderada $u = x_1w_1 + x_2w_2 + \cdots + w_nx_n + b$ donde $W = (w_i)_{i=1}^n \in \mathbb{R}^n$ corresponden a los **pesos** (weights) y $b \in \mathbb{R}$ el **sesgo** (bias).
A continuación, se aplica una función de activación f y se entrega un output $h = f(u)$.

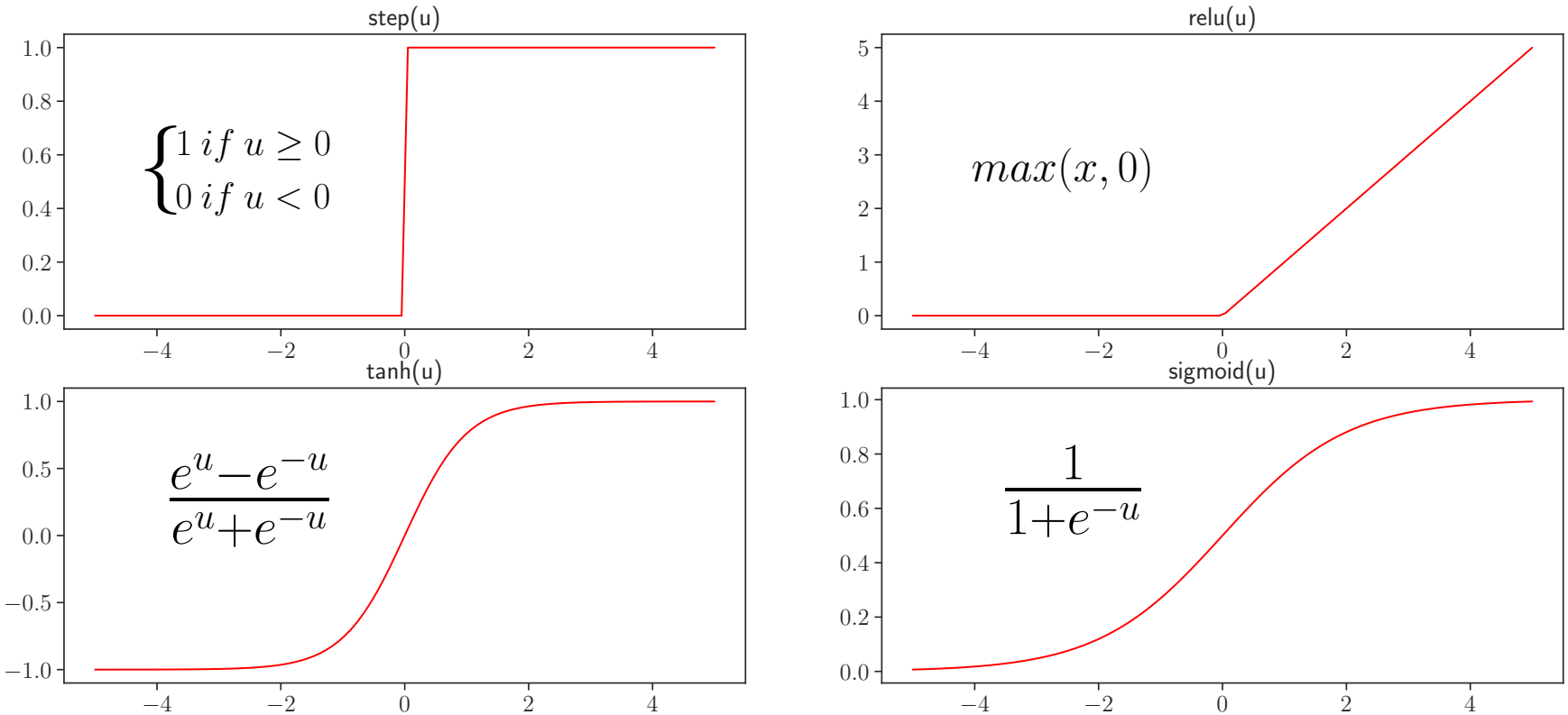

Fig.. Algunos ejemplos de funciones de activación

Redes Neuronales - El perceptrón

Veamos el siguiente ejemplo:

XOR operator

x_1	x_2	y
0	0	0
1	0	1
0	1	1
1	1	0

Fig.. Red Neuronal de 2 perceptrones para XOR

En su forma matricial

$$(h_1, h_2) = \text{Relu}\left(\begin{pmatrix} x_1 & x_2 \end{pmatrix}\begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix} + \begin{pmatrix} 0 & -1 \end{pmatrix}\right) \qquad h = \text{Relu}(xW + b)$$

$$y = \begin{pmatrix} h_1 & h_2 \end{pmatrix}\begin{pmatrix} 1 \\ -2 \end{pmatrix} + (0) \qquad y = hU + c$$

Donde U y c corresponden al peso y bias de la última capa respectivamente.

¿Cómo construir el resto de operadores lógicos?

Redes Neuronales - El perceptrón

Veamos el siguiente ejemplo:

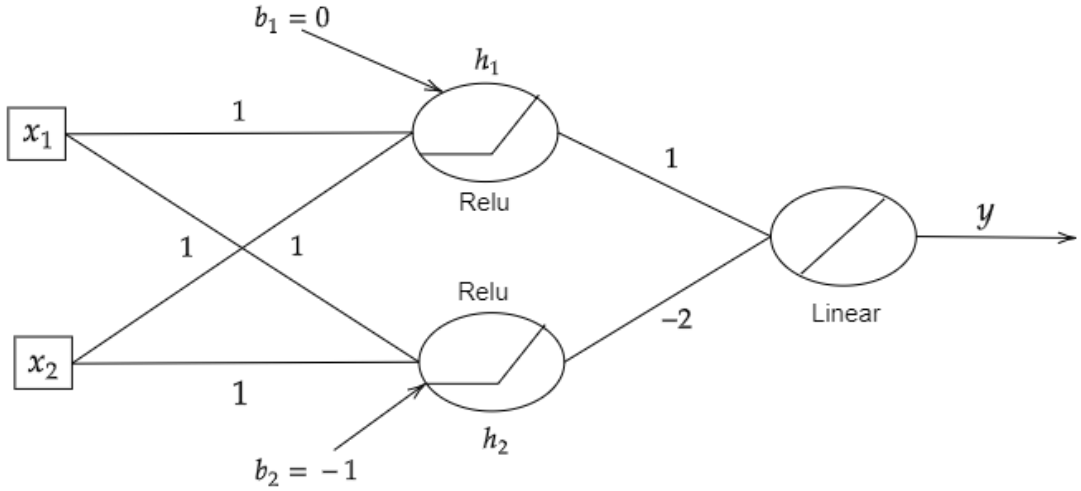

Fig.. Red Neuronal de 2 perceptrones para XOR

XOR operator

x_1	x_2	y
0	0	0
1	0	1
0	1	1
1	1	0

En su forma matricial

$$(h_1, h_2) = \text{Relu}\left(\begin{pmatrix} x_1 & x_2 \end{pmatrix} \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix} + \begin{pmatrix} 0 & -1 \end{pmatrix} \right) \qquad h = \text{Relu}(xW + b)$$

$$y = \begin{pmatrix} h_1 & h_2 \end{pmatrix} \begin{pmatrix} 1 \\ -2 \end{pmatrix} + (0) \qquad y = hU + c$$

Donde U y c corresponden al peso y bias de la última capa respectivamente.

¿Cómo construir el resto de operadores lógicos?

Redes Neuronales - El perceptrón

Veamos el siguiente ejemplo:

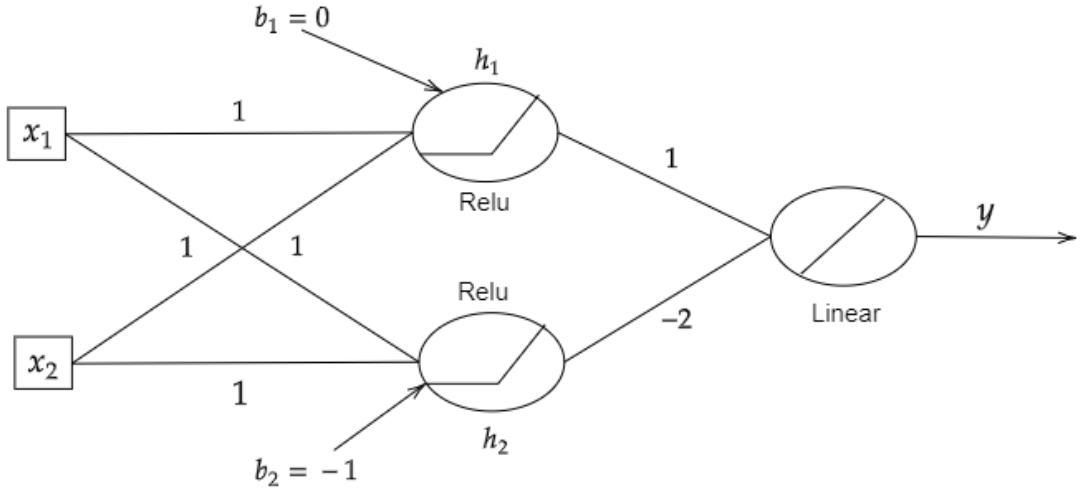

Fig.. Red Neuronal de 2 perceptrones para XOR

XOR operator

x_1	x_2	y
0	0	0
1	0	1
0	1	1
1	1	0

En su forma matricial

$$(h_1, h_2) = \text{Relu}\left(\begin{pmatrix} x_1 & x_2 \end{pmatrix}\begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix} + \begin{pmatrix} 0 & -1 \end{pmatrix}\right) \qquad h = \text{Relu}(xW + b)$$

$$y = \begin{pmatrix} h_1 & h_2 \end{pmatrix}\begin{pmatrix} 1 \\ -2 \end{pmatrix} + (0) \qquad y = hU + c$$

Donde U y c corresponden al peso y bias de la última capa respectivamente.

¿Cómo construir el resto de operadores lógicos?

Redes Neuronales - El perceptrón

Veamos el siguiente ejemplo:

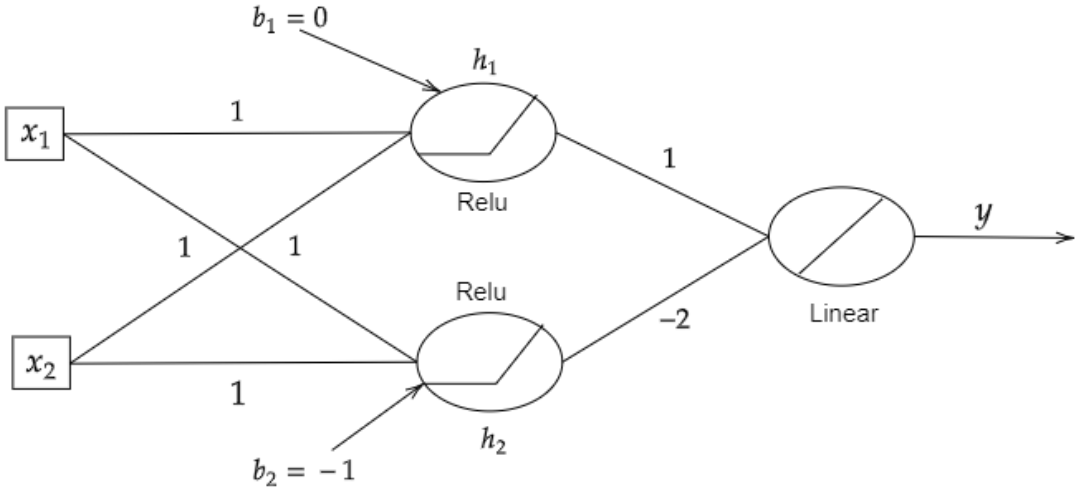

Fig.. Red Neuronal de 2 perceptrones para XOR

XOR operator

x_1	x_2	y
0	0	0
1	0	1
0	1	1
1	1	0

En su forma matricial

$$(h_1, h_2) = \text{Relu}\left(\begin{pmatrix} x_1 & x_2 \end{pmatrix}\begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix} + \begin{pmatrix} 0 & -1 \end{pmatrix}\right) \qquad h = \text{Relu}(xW + b)$$

$$y = \begin{pmatrix} h_1 & h_2 \end{pmatrix}\begin{pmatrix} 1 \\ -2 \end{pmatrix} + (0) \qquad y = hU + c$$

Donde U y c corresponden al peso y bias de la última capa respectivamente.

¿Cómo construir el resto de operadores lógicos?

Teorema de Aproximación Universal[1]

Un conjunto de perceptrones conectados unos con otros en secuencia forman arquitecturas capaces de resolver problemas más complejos.

Bajo el contexto de una red de múltiples perceptrones con una sola capa escondida, se tiene el siguiente resultado

Theorem (UAT - Ancho arbitrario)

Sea f^* una función objetivo Borel Medible (por ejemplo $f^* : [0,1]^k \to [0,1]$ continua con $k \in \mathbb{N}$) y sea f una función de activación no polinomial acotada, entonces para todo $\epsilon > 0$ existen W, b, U definidas como antes tales que

$$F(x) = f(xW + b)U, \quad |F(x) - f^*(x)| < \epsilon \quad \forall x \in Dom(f^*)$$

En palabras simples, con una cantidad suficiente de perceptrones, es posible aproximar una función objetivo razonable tan finamente como se desee.

[1] (Horniket al., 1989; Cybenko, 1989)

Teorema de Aproximación Universal[1]

Un conjunto de perceptrones conectados unos con otros en secuencia forman arquitecturas capaces de resolver problemas más complejos.
Bajo el contexto de una red de múltiples perceptrones con una sola capa escondida, se tiene el siguiente resultado

Theorem (UAT - Ancho arbitrario)

Sea f^* una función objetivo Borel Medible (por ejemplo $f^* : [0,1]^k \to [0,1]$ continua con $k \in \mathbb{N}$) y sea f una función de activación no polinomial acotada, entonces para todo $\epsilon > 0$ existen W, b, U definidas como antes tales que

$$F(x) = f(xW + b)U, \quad |F(x) - f^*(x)| < \epsilon \quad \forall x \in Dom(f^*)$$

En palabras simples, con una cantidad suficiente de perceptrones, es posible aproximar una función objetivo razonable tan finamente como se desee.

[1] (Horniket al., 1989; Cybenko, 1989)

Teorema de Aproximación Universal[1]

Un conjunto de perceptrones conectados unos con otros en secuencia forman arquitecturas capaces de resolver problemas más complejos.
Bajo el contexto de una red de múltiples perceptrones con una sola capa escondida, se tiene el siguiente resultado

Theorem (UAT - Ancho arbitrario)

Sea f^* una función objetivo Borel Medible (por ejemplo $f^* : [0,1]^k \to [0,1]$ continua con $k \in \mathbb{N}$) y sea f una función de activación no polinomial acotada, entonces para todo $\epsilon > 0$ existen W, b, U definidas como antes tales que

$$F(x) = f(xW + b)U, \quad |F(x) - f^*(x)| < \epsilon \quad \forall x \in Dom(f^*)$$

En palabras simples, con una cantidad suficiente de perceptrones, es posible aproximar una función objetivo razonable tan finamente como se desee.

[1] (Horniket al., 1989; Cybenko, 1989)

Teorema de Aproximación Universal[1]

Un conjunto de perceptrones conectados unos con otros en secuencia forman arquitecturas capaces de resolver problemas más complejos.
Bajo el contexto de una red de múltiples perceptrones con una sola capa escondida, se tiene el siguiente resultado

Theorem (UAT - Ancho arbitrario)

Sea f^* una función objetivo Borel Medible (por ejemplo $f^* : [0,1]^k \to [0,1]$ continua con $k \in \mathbb{N}$) y sea f una función de activación no polinomial acotada, entonces para todo $\epsilon > 0$ existen W, b, U definidas como antes tales que

$$F(x) = f(xW + b)U, \quad |F(x) - f^*(x)| < \epsilon \quad \forall x \in Dom(f^*)$$

En palabras simples, con una cantidad suficiente de perceptrones, es posible aproximar una función objetivo razonable tan finamente como se desee.

[1] (Horniket al., 1989; Cybenko, 1989)

Redes Neuronales - Arquitectura

Ya vimos que es posible aproximar funciones con el uso del UAT, el principal problema es que la red podría no necesariamente 'aprender' la función en si, y tampoco asegura una cota para la cantidad de neuronas necesarias. Es por esto que surge la necesidad de arquitecturas más complejas, es decir, con mayor cantidad de capas (mayor profundidad).

Fig.. Una red con mayor profundidad

Utilizaremos la siguiente notación

$$h^{(k)} = f^{(k)}(h^{(k-1)}W^{(k)} + b^{(k)}) \quad \forall k \in \{1, \ldots, l\}, \quad h^{(0)} = x \tag{1.1}$$

$$\hat{y} = g(h^{(l)}U + c) \tag{1.2}$$

Donde g es la función aplicada en la capa de output y es la que define la **unidad de output**

Redes Neuronales - Arquitectura

Ya vimos que es posible aproximar funciones con el uso del UAT, el principal problema es que la red podría no necesariamente 'aprender' la función en si, y tampoco asegura una cota para la cantidad de neuronas necesarias. Es por esto que surge la necesidad de arquitecturas más complejas, es decir, con mayor cantidad de capas (mayor profundidad).

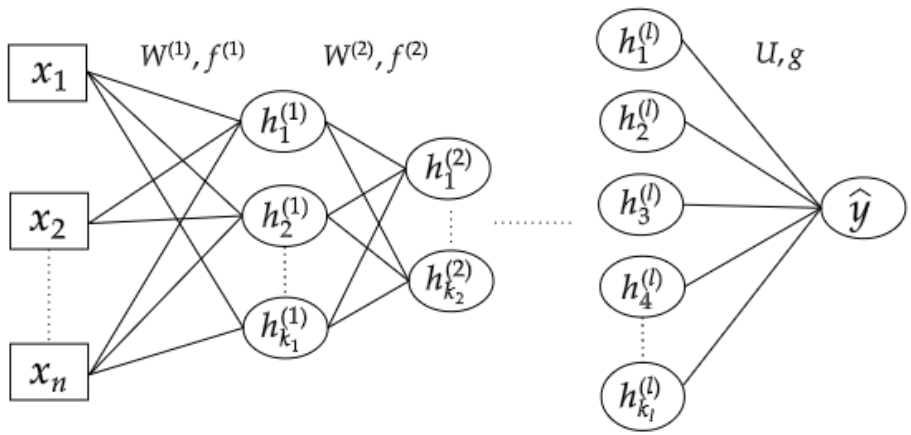

Fig.. Una red con mayor profundidad

Utilizaremos la siguiente notación

$$h^{(k)} = f^{(k)}(h^{(k-1)}W^{(k)} + b^{(k)}) \quad \forall k \in \{1, \ldots, l\}, \quad h^{(0)} = x \tag{1.1}$$

$$\hat{y} = g(h^{(l)}U + c) \tag{1.2}$$

Donde g es la función aplicada en la capa de output y es la que define la **unidad de output**

Redes Neuronales - Arquitectura

Ya vimos que es posible aproximar funciones con el uso del UAT, el principal problema es que la red podría no necesariamente 'aprender' la función en si, y tampoco asegura una cota para la cantidad de neuronas necesarias. Es por esto que surge la necesidad de arquitecturas más complejas, es decir, con mayor cantidad de capas (mayor profundidad).

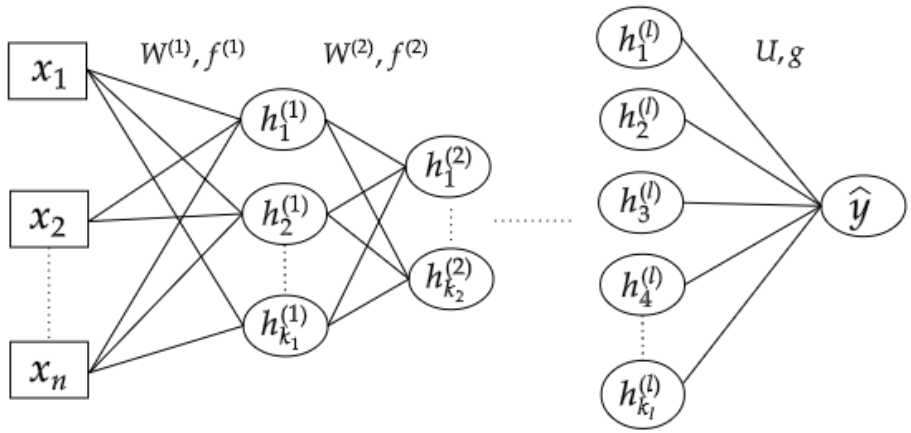

Fig.. Una red con mayor profundidad

Utilizaremos la siguiente notación

$$h^{(k)} = f^{(k)}(h^{(k-1)}W^{(k)} + b^{(k)}) \quad \forall k \in \{1, \ldots, l\}, \quad h^{(0)} = x \tag{1.1}$$

$$\hat{y} = g(h^{(l)}U + c) \tag{1.2}$$

Donde g es la función aplicada en la capa de output y es la que define la **unidad de output**

Función de costos y unidades de output

Función de costos

Una de las principales diferencias entre los modelos lineales antes vistos y una red neuronal, es que el uso de ciertas funciones de activación hacen que la función de costos **no sea convexa**, esto hace que el entrenamiento realizado en base a descenso de gradiente no entregue garantías de que se alcanzará el óptimo global, o una buena solución en términos generales, ya que el algoritmo podría estancarse en un optimo local que entregue resultados pobres.

Así, para la mayor parte de los problemas a los que se enfrenta una red, nos gustaría que el modelo paramétrico defina una distribución $p(y|\boldsymbol{x};\boldsymbol{\theta})$ y estimar los parámetros óptimos mediante máxima verosimilitud.
La **función de costos** que utilizaremos entonces será la entropía cruzada (cross-entropy) definida como

$$J(\boldsymbol{\theta}) = -\mathbb{E}_{\boldsymbol{x},\boldsymbol{y}\sim\hat{p}_{\text{data}}}(\log p_{\text{modelo}}(\boldsymbol{y}|\boldsymbol{x}))$$

Función de costos

Una de las principales diferencias entre los modelos lineales antes vistos y una red neuronal, es que el uso de ciertas funciones de activación hacen que la función de costos **no sea convexa**, esto hace que el entrenamiento realizado en base a descenso de gradiente no entregue garantías de que se alcanzará el óptimo global, o una buena solución en términos generales, ya que el algoritmo podría estancarse en un optimo local que entregue resultados pobres.

Así, para la mayor parte de los problemas a los que se enfrenta una red, nos gustaría que el modelo paramétrico defina una distribución $p(y|\boldsymbol{x};\boldsymbol{\theta})$ y estimar los parámetros óptimos mediante máxima verosimilitud.
La **función de costos** que utilizaremos entonces será la entropía cruzada (cross-entropy) definida como

$$J(\boldsymbol{\theta}) = -\mathbb{E}_{\boldsymbol{x},\boldsymbol{y}\sim\hat{p}_{\text{data}}}(\log p_{\text{modelo}}(\boldsymbol{y}|\boldsymbol{x}))$$

Función de costos

Una de las principales diferencias entre los modelos lineales antes vistos y una red neuronal, es que el uso de ciertas funciones de activación hacen que la función de costos **no sea convexa**, esto hace que el entrenamiento realizado en base a descenso de gradiente no entregue garantías de que se alcanzará el óptimo global, o una buena solución en términos generales, ya que el algoritmo podría estancarse en un optimo local que entregue resultados pobres.

Así, para la mayor parte de los problemas a los que se enfrenta una red, nos gustaría que el modelo paramétrico defina una distribución $p(y|\boldsymbol{x};\boldsymbol{\theta})$ y estimar los parámetros óptimos mediante máxima verosimilitud.
La **función de costos** que utilizaremos entonces será la entropía cruzada (cross-entropy) definida como

$$J(\boldsymbol{\theta}) = -\mathbb{E}_{\boldsymbol{x},\boldsymbol{y}\sim\hat{p}_{\text{data}}}(\log p_{\text{modelo}}(\boldsymbol{y}|\boldsymbol{x}))$$

Función de costos

Una de las principales diferencias entre los modelos lineales antes vistos y una red neuronal, es que el uso de ciertas funciones de activación hacen que la función de costos **no sea convexa**, esto hace que el entrenamiento realizado en base a descenso de gradiente no entregue garantías de que se alcanzará el óptimo global, o una buena solución en términos generales, ya que el algoritmo podría estancarse en un optimo local que entregue resultados pobres.

Así, para la mayor parte de los problemas a los que se enfrenta una red, nos gustaría que el modelo paramétrico defina una distribución $p(y|\boldsymbol{x};\boldsymbol{\theta})$ y estimar los parámetros óptimos mediante máxima verosimilitud.
La **función de costos** que utilizaremos entonces será la entropía cruzada (cross-entropy) definida como

$$J(\boldsymbol{\theta}) = -\,\mathbb{E}_{\boldsymbol{x},\boldsymbol{y}\sim\hat{p}_{\text{data}}}(\log p_{\text{modelo}}(\boldsymbol{y}|\boldsymbol{x}))$$

Unidad de output

La elección de la **Unidad de output** definirá la forma que toma la función de costos, algunos ejemplos

1. Unidad de output lineal $\hat{\boldsymbol{y}} = h^{(l)}U + c$
   Útil para cuando se busca retornar la media de una distribución Gaussiana condicional $p(\boldsymbol{y}|\boldsymbol{x}) = \mathcal{N}(\boldsymbol{y}|\hat{\boldsymbol{y}}; \boldsymbol{I})$, el problema de maximizar la log-verosimilitud será equivalente minimizar el error cuadrático medio.
2. Unidad de output sigmoidal $\hat{y} = \text{sig}(h^{(l)}U + c)$
   Este se utiliza para problemas de clasificación binaria, cuyo output es $P(y = 1|\boldsymbol{x})$ o la probabilidad de pertenecer a la clase 1.
3. Unidad de output softmax $\hat{y} = \text{softmax}(h^{(l)}U + c)$
   Es una generalización de la función sigmoidal definida como

$$\text{softmax}(z)_i = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

   y es útil para el caso de clasificación multiclase.

Unidad de output

La elección de la **Unidad de output** definirá la forma que toma la función de costos, algunos ejemplos

1. Unidad de output lineal $\hat{\boldsymbol{y}} = h^{(l)}U + c$
   Útil para cuando se busca retornar la media de una distribución Gaussiana condicional $p(\boldsymbol{y}|\boldsymbol{x}) = \mathcal{N}(\boldsymbol{y}|\hat{\boldsymbol{y}}; \boldsymbol{I})$, el problema de maximizar la log-verosimilitud será equivalente minimizar el error cuadrático medio.
2. Unidad de output sigmoidal $\hat{y} = \text{sig}(h^{(l)}U + c)$
   Este se utiliza para problemas de clasificación binaria, cuyo output es $P(y = 1|\boldsymbol{x})$ o la probabilidad de pertenecer a la clase 1.
3. Unidad de output softmax $\hat{y} = \text{softmax}(h^{(l)}U + c)$
   Es una generalización de la función sigmoidal definida como

$$\text{softmax}(z)_i = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

   y es útil para el caso de clasificación multiclase.

Unidad de output

La elección de la **Unidad de output** definirá la forma que toma la función de costos, algunos ejemplos

1. Unidad de output lineal $\hat{\boldsymbol{y}} = h^{(l)}U + c$
   Útil para cuando se busca retornar la media de una distribución Gaussiana condicional $p(\boldsymbol{y}|\boldsymbol{x}) = \mathcal{N}(\boldsymbol{y}|\hat{\boldsymbol{y}}; \boldsymbol{I})$, el problema de maximizar la log-verosimilitud será equivalente minimizar el error cuadrático medio.
2. Unidad de output sigmoidal $\hat{y} = \text{sig}(h^{(l)}U + c)$
   Este se utiliza para problemas de clasificación binaria, cuyo output es $P(y = 1|\boldsymbol{x})$ o la probabilidad de pertenecer a la clase 1.
3. Unidad de output softmax $\hat{y} = \text{softmax}(h^{(l)}U + c)$
   Es una generalización de la función sigmoidal definida como

$$\text{softmax}(z)_i = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

   y es útil para el caso de clasificación multiclase.

Unidad de output

La elección de la **Unidad de output** definirá la forma que toma la función de costos, algunos ejemplos

1. Unidad de output lineal $\hat{\boldsymbol{y}} = h^{(l)}U + c$
   Útil para cuando se busca retornar la media de una distribución Gaussiana condicional $p(\boldsymbol{y}|\boldsymbol{x}) = \mathcal{N}(\boldsymbol{y}|\hat{\boldsymbol{y}}; \boldsymbol{I})$, el problema de maximizar la log-verosimilitud será equivalente minimizar el error cuadrático medio.
2. Unidad de output sigmoidal $\hat{y} = \text{sig}(h^{(l)}U + c)$
   Este se utiliza para problemas de clasificación binaria, cuyo output es $P(y = 1|\boldsymbol{x})$ o la probabilidad de pertenecer a la clase 1.
3. Unidad de output softmax $\hat{y} = \text{softmax}(h^{(l)}U + c)$
   Es una generalización de la función sigmoidal definida como

$$\text{softmax}(z)_i = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

   y es útil para el caso de clasificación multiclase.

Unidad de output

La elección de la **Unidad de output** definirá la forma que toma la función de costos, algunos ejemplos

1. Unidad de output lineal $\hat{\boldsymbol{y}} = h^{(l)}U + c$
   Útil para cuando se busca retornar la media de una distribución Gaussiana condicional $p(\boldsymbol{y}|\boldsymbol{x}) = \mathcal{N}(\boldsymbol{y}|\hat{\boldsymbol{y}}; \boldsymbol{I})$, el problema de maximizar la log-verosimilitud será equivalente minimizar el error cuadrático medio.
2. Unidad de output sigmoidal $\hat{y} = \text{sig}(h^{(l)}U + c)$
   Este se utiliza para problemas de clasificación binaria, cuyo output es $P(y = 1|\boldsymbol{x})$ o la probabilidad de pertenecer a la clase 1.
3. Unidad de output softmax $\hat{y} = \text{softmax}(h^{(l)}U + c)$
   Es una generalización de la función sigmoidal definida como

$$\text{softmax}(z)_i = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

   y es útil para el caso de clasificación multiclase.

Unidad de output

La elección de la **Unidad de output** definirá la forma que toma la función de costos, algunos ejemplos

1. Unidad de output lineal $\hat{\boldsymbol{y}} = h^{(l)}U + c$
   Útil para cuando se busca retornar la media de una distribución Gaussiana condicional $p(\boldsymbol{y}|\boldsymbol{x}) = \mathcal{N}(\boldsymbol{y}|\hat{\boldsymbol{y}}; \boldsymbol{I})$, el problema de maximizar la log-verosimilitud será equivalente minimizar el error cuadrático medio.
2. Unidad de output sigmoidal $\hat{y} = \text{sig}(h^{(l)}U + c)$
   Este se utiliza para problemas de clasificación binaria, cuyo output es $P(y = 1|\boldsymbol{x})$ o la probabilidad de pertenecer a la clase 1.
3. Unidad de output softmax $\hat{y} = \text{softmax}(h^{(l)}U + c)$
   Es una generalización de la función sigmoidal definida como

$$\text{softmax}(z)_i = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

   y es útil para el caso de clasificación multiclase.

Clase 16 - Redes Neuronales I

Aprendizaje de Máquinas - MA5204

Felipe Tobar

Department of Mathematical Engineering &
Center for Mathematical Modelling
Universidad de Chile

12 de marzo de 2021

UNIVERSIDAD
DE CHILE