

Redes neuronales

Víctor Mijangos

Facultad de Ingeniería

Procesamiento de Lenguaje Natural



UNAM

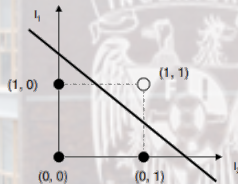
Solución de problemas no linealmente separables



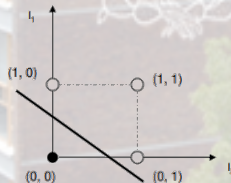
Problemas con el perceptrón

El perceptrón es un método de separación lineal. Por tanto su capacidad es limitada.

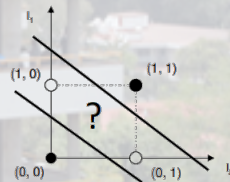
AND		
I_1	I_2	out
0	0	0
0	1	0
1	0	0
1	1	1



OR		
I_1	I_2	out
0	0	0
0	1	1
1	0	1
1	1	1



XOR		
I_1	I_2	out
0	0	0
0	1	1
1	0	1
1	1	0



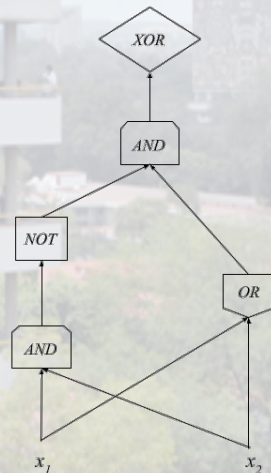
¿Cómo solucionar el problema XOR?

El problema XOR es un problema complejo, compuesto de elementos simples:

$$XOR(x_1, x_2) = AND(NOR(AND(x_1, x_2)), OR(x_1, x_2))$$

Podemos aplicar perceptrones de forma compuesta para solucionar este problema. Esto conforma una estructura gráfica más compleja.

En la última “capa” se trabaja con dos entradas: los puntos originales ahora son linealmente separables.



Otras soluciones al problema XOR

La idea es modificar los datos para que estos sean separables por un plano o hiperplano. En otras palabras, buscamos transformar los datos. Esto se puede hacer de distintas formas.



Solución al problema XOR

- ▶ El problema XOR puede resolverse por la composición de diferentes perceptrones.
- ▶ En **términos lógicos** el problema XOR es la composición de problemas lógicos básicos.
- ▶ En **términos geométricos** se transforman los datos a un espacio donde sean linealmente separables.
- ▶ Se pueden utilizar **transformaciones lineales**; sin embargo éstas **son insuficientes** en muchos casos.



¿Cómo separar un problema XOR?

Por ejemplo, se puede tomar la transformación lineal:

$$L^T = \begin{pmatrix} 1 & 0 \\ -1 & -1 \end{pmatrix}$$

Esta transformación hace que los datos sean

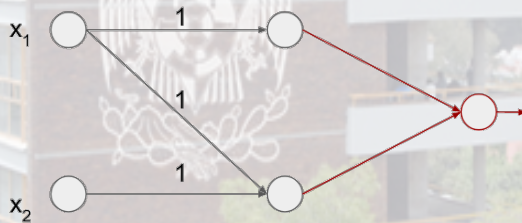
$$XL = \begin{pmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} 1 & -1 \\ 0 & -1 \end{pmatrix} = \begin{pmatrix} 0 & 0 \\ 0 & -1 \\ 1 & -1 \\ 1 & 0 \end{pmatrix}, Y = \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}$$

Que se puede resolver con un perceptrón con parámetros $w = (0 \quad -1)$, $b = -1$



Incorporación de la transformación en la red neuronal

La transformación L se puede expresar por medio de un grafo dirigido, particularmente como un perceptrón.

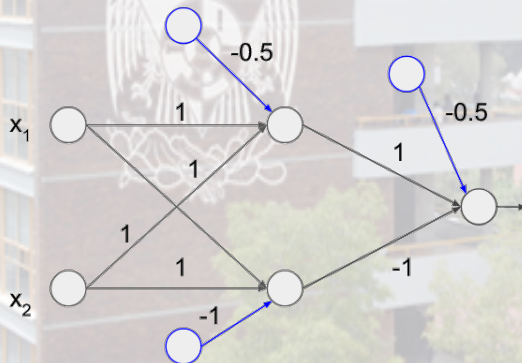


La parte roja representa el perceptrón que trabaja con los datos transformados.



Uso de capas ocultas

Más aún, se puede pensar en transformaciones que no sean meramente lineales. Así, los nodos que preseden al “perceptrón de salida” o capa de salida pueden formalizarse como un perceptrón, con bias y función de activación.



Uso de capas ocultas

Cada nodo en una capa oculta se conoce como **unidad oculta**. Cada unidad oculta puede verse como un perceptrón, que toma como entrada el vector de la capa anterior.

En el caso anterior tenemos dos unidades ocultas:

- ▶ Unidad 1, donde $w' = (1 \ 1)$ y $b' = -0,5$.
- ▶ Unidad 2, donde $w'' = (1 \ 1)$ y $b'' = -1$.

Estas unidades conforman una matriz:

$$W = \begin{pmatrix} w' \\ w'' \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$$

Y los bias conforman un vector: $b = (b' \ b'') = (-0,5 \ -1)$

Por tanto la función $Wx + b$ equivale a aplicar cada función para obtener los valores en cada unidad de la red.



Capas ocultas

Los nodos intermedios en esta estructura conforman una capa oculta. Por tanto, podemos pensar una red neuronal con una capa oculta a partir de las funciones:

$$h(x) = g(W^{(1)}x + b^{(1)}) \quad (1)$$

$$f(x) = \phi(W^{(2)}x + b^{(2)}) \quad (2)$$

Donde $W^{(1)} \in \mathbb{R}^{d \times m}$, $b^{(1)} \in \mathbb{R}^m$, $W^{(2)} \in \mathbb{R}^{m \times o}$ y $b^{(2)} \in \mathbb{R}^o$ (o es el número de salidas). Además, g y ϕ son funciones de activación (se busca que sean **derivables**).



Redes FeedForward (Pre-alimentadas)



Redes neuronales pre-alimentadas

Una red neuronal pre-alimentada (o *feedforward NN*) es una red con K capas ocultas. Cuenta con:

1. Una pre-activación para cada capa $0 < k \leq K$, dada por:

$$a^{(k)}(x) = W^{(k)}h^{(k-1)}(x) + b^{(k)}$$

2. Una función de activación para cada capa $0 < k \leq K$:

$$h^{(k)}(x) = g(a^{(k)}(x))$$

3. Una activación de salida:

$$f(x) = \phi[W^{(K+1)}h^{(K)}(x) + b^{(K+1)}]$$

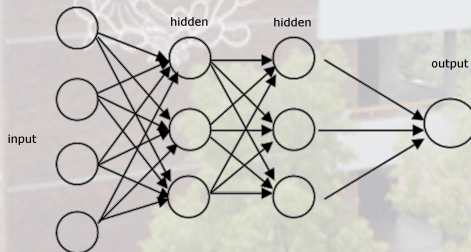


Redes neuronales pre-alimentadas

Las dimensiones de las matrices en las diferentes capas son:

1. En la capa de entrada, $W^{(1)}$ es una matriz de $d \times m_1$, con d dimensión de los vectores de entrada.
2. Para la k -ésima capa, $W^{(k)}$ es de $m_{k-1} \times m_k$.
3. En la capa de salida, $W^{(K+1)}$ es una matriz de $m_K \times o$, con o dimensiones de salida.

La **arquitectura** de una red neuronal feedforward especifica el número de capas ocultas y las unidades de cada una (K y m_1, \dots, m_K) [3].



Funciones de pre-activación y activación

Dentro de un red feedforward se hablará de dos “tipos de funciones”:

- ▶ **Funciones de pre-activación:** son funciones afines (lineales más una traslación) de la forma:

$$W^{(k)}h^{(k-1)}(x) + b^{(k)}$$

Con $0 < k \leq K$. La concatenación de funciones afines produce otra función afín.

- ▶ **Funciones de activación:** Permiten definir capas no lineales al aplicar una función no lineal $g : \mathbb{R}^{m_{k-1}} \rightarrow \mathbb{R}^{m_k}$ a la pre-activación. Para aplicar optimización por gradiente, g debe ser derivable.



Funciones de activación

Para las capas ocultas, se pueden determinar diferentes funciones de activación g :

- ▶ Sigmoides:

$$\sigma(a) = \frac{1}{1 + e^{-a}}$$

- ▶ Tangente hiperbólica:

$$\tanh(a) = \frac{e^a - e^{-a}}{e^a + e^{-a}}$$

- ▶ Softplus:

$$S(a) = \log(1 + e^a)$$

- ▶ Rectified Linear Unit:

$$\text{ReLU}(a) = \max\{0, a\}$$



Funciones de activación

Estas funciones de activación son comunes porque son fácilmente derivables. Sus **derivadas** son:

- ▶ Sigmoide:

$$\sigma'(a) = \sigma(a)(1 - \sigma(a))$$

- ▶ Tangente hiperbólica:

$$\tanh'(a) = 1 - \tanh^2(a)$$

- ▶ Softplus:

$$S(a) = \sigma(a)$$

- ▶ Rectified Linear Unit:

$$\text{ReLU}(a) = \begin{cases} 1 & \text{si } a > 0 \\ 0 & \text{si } a < 0 \end{cases}$$



Activación en la capa de salida

La **capa de salida** muestra una función de activación particular, pues es la función que determinará la solución al problema.

Podemos proponer tres funciones de activación que dependen del problema a resolver.

1. **Función sigmoide:** Si se tiene una sola neurona de salida. Determina una probabilidad $p(Y = 1) = \sigma(a)$, donde a es la única salida.
2. **Función Softmax:** Generalización de la sigmoide. Se utiliza cuando se cuentan con diferentes salidas: $\text{Softmax}(a_j)$, con $j \in 1, 2, \dots, o$.
3. **Función lineal:** Cuando se trata de un problema de regresión (por ejemplo, en redes AutoEncoders).



Función Softmax

La función Softmax es la más comúnmente utilizada como activación en las capas de salida de redes neuronales.

Supóngase que se cuenta con a_1, a_2, \dots, a_o neuronas de salida, la función Softmax para la salida a_j se define como:

$$\text{Softmax}(a_j) = \frac{e^{a_j}}{\sum_{i=1}^o e^{a_i}}$$

La función Softmax define una función de probabilidad sobre las unidades de salida.

Para la clasificación en las clases de salida, tenemos que

$$\hat{y} = \arg \max_j \{\text{Softmax}(a_j)\}$$



Derivada de la función Softmax

La función Softmax puede verse como una función de \mathbb{R}^o a \mathbb{R}^o (vector de preactivación a vector de activaciones de salida).

Para obtener la derivada parcial de la función Softmax, denotemos ψ_j a la salida j . Simplificaremos derivando sobre el logaritmo:

$$\frac{\partial \log \psi_j}{\partial a_i} = \frac{1}{\psi_j} \frac{\partial \psi_j}{\partial a_i}$$

De tal forma que:

$$\frac{\partial \psi_j}{\partial a_i} = \psi_j \frac{\partial \log \psi_j}{\partial a_i}$$

De esta forma, bastará obtener la derivada de $\log \psi_j$ para obtener la derivada de la función Softmax.



Derivada de la función Softmax

Por las propiedades del logaritmo, tenemos que:

$$\log \psi_j = \log \frac{e^{a_j}}{\sum_k e^{a_k}} = a_j - \log \sum_k e^{a_k}$$

Derivando esta expresión obtenemos que:

$$\frac{\partial \log \psi_j}{\partial a_i} = \frac{\partial a_j}{\partial a_i} - \frac{\partial \log \sum_k e^{a_k}}{\partial a_i}$$

Del segundo elemento, tenemos que:

$$\frac{\partial \log \sum_k e^{a_k}}{\partial a_i} = \frac{1}{\sum_k e^{a_k}} \frac{\partial \sum_k e^{a_k}}{\partial a_i} = \frac{e^{a_i}}{\sum_k e^{a_k}} = \psi_i$$



Derivada de la función Softmax

Falta derivar el factor $\frac{\partial a_j}{\partial a_i}$. Este caso sencillo tenemos que:

$$\frac{\partial a_j}{\partial a_i} = \delta_{i,j}$$

Donde $\delta_{i,j}$ es la delta de Kronecker definida como:

$$\delta_{i,j} = \begin{cases} 1 & \text{si } i = j \\ 0 & \text{si } i \neq j \end{cases}$$

Por tanto, la **derivada de la función Softmax** sobre la i -ésima salida es:

$$\frac{\partial \psi_j}{\partial a_i} = \psi_j(\delta_{i,j} - \psi_i)$$

Donde $\psi_j = \text{Softmax}(a_j)$ es la salida j -ésima de la red.



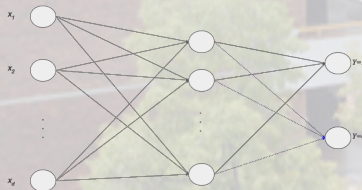
Relación de Softmax y Sigmoide

La función Softmax puede aplicarse a un problema binario, pero requerirá de dos neuronas de salida, a_1 y a_2 , tal que:

$$\text{Softmax}(a_1) = \frac{e^{a_1}}{e^{a_1} + e^{a_2}}$$

Si se reduce a una sola salida, se deberán eliminar las conexiones de una neurona de salida: $a_2 = 0$, por lo que tendremos:

$$\text{Softmax}(a_1) = \frac{e^{a_1}}{e^{a_1} + e^0} = \frac{e^{a_1}}{e^{a_1} + 1} = \frac{1}{1 + e^{-a_1}} = \sigma(a_1)$$



Resultados sobre las redes FeedForward y familia de redes



¿Por qué nos gustan tanto las redes neuronales?

Las redes neuronales FeedForward son una herramienta de gran importancia, pues permiten aproximar funciones $f : \mathbb{R}^d \rightarrow \mathbb{R}^o$.

Teorema (Aproximador universal [1])

Una red neuronal f con m unidades ocultas (una sola capa) puede aproximar tanto como se quiera cualquier función Borel-medible.

El problema al que nos enfrentamos es elegir la arquitectura adecuada para nuestro problema y, más aún, tener una muestra suficientemente rica para lograr la aproximación.



Capacidad de una red FeedForward

La capacidad, en términos de **dimensión VC**, de una red es difícil de determinar. La capacidad de una red neuronal depende de varios factores:

1. Número de capas ocultas
2. Número de unidades ocultas por capa
3. Hiperparámetros: funciones de activación, función de riesgo, rango de aprendizaje, método de optimización...

Podemos formular el siguiente resultado [2]:

Teorema

La dimensión VC de una red neuronal es $O(|\theta| \log |\theta|)$, donde θ son los parámetros de la red.



Relevancia de las redes neuronales

Algunas **ventajas** del uso de redes son:

- ▶ Una red con una sola capa puede aproximar adecuadamente cualquier función de interés.
- ▶ Son capaces de encontrar representaciones abstractas que permitan una clasificación adecuada.
- ▶ Alta flexibilidad, permiten un buen número de arquitecturas.
- ▶ Requieren poco conocimiento del problema.

Algunas **desventajas** son:

- ▶ Requieren una buena cantidad de datos para converger adecuadamente.
- ▶ Requieren poder de procesamiento alto. Muchas veces poco accesible.
- ▶ Son poco interpretables.



Flexibilidad de las redes neuronales

Las redes FeedForward son una arquitectura **simple**.

Pero éstas pueden verse como una particularidad de las **redes recurrentes**.















Asimismo, existen diferentes formas de conceptualizar una red, lo que determina diferentes algoritmos basados en redes neuronales.

Esto permite que las redes puedan configurarse de diferentes formas, permitiendo crear estructuras de redes que solucionen diferentes problemas.



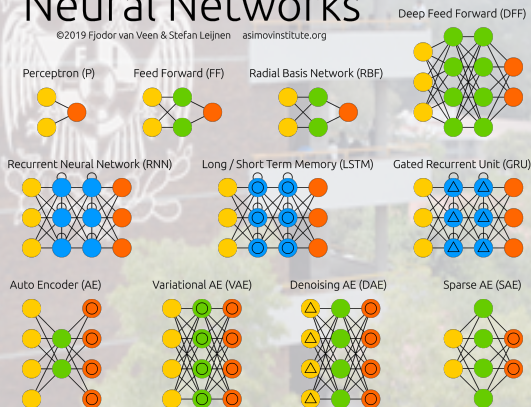
Familia de redes neuronales

Algunas arquitecturas conocidas son [4]:

-  Input Cell
-  Backfed Input Cell
-  Noisy Input Cell
-  Hidden Cell
-  Probablistic Hidden Cell
-  Spiking Hidden Cell
-  Capsule Cell
-  Output Cell
-  Match Input Output Cell
-  Recurrent Cell
-  Memory Cell
-  Gated Memory Cell
-  Kernel
-  Convolution or Pool

A mostly complete chart of Neural Networks

©2019 Fjodor van Veen & Stefan Leijnen asimovinstitute.org



Familia de redes neuronales

Otras arquitecturas son:

Markov Chain (MC)



Hopfield Network (HN)



Boltzmann Machine (BM)



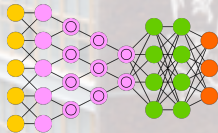
Restricted BM (RBM)



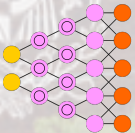
Deep Belief Network (DBN)



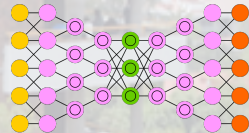
Deep Convolutional Network (DCN)



Deconvolutional Network (DN)



Deep Convolutional Inverse Graphics Network (DCIGN)



Generative Adversarial Network (GAN)



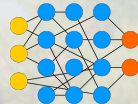
Liquid State Machine (LSM)



Extreme Learning Machine (ELM)



Echo State Network (ESN)



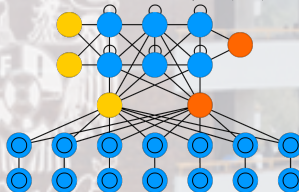
Familia de redes neuronales

Otras arquitecturas son:

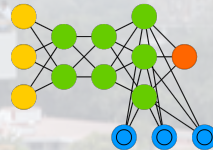
Deep Residual Network (DRN)



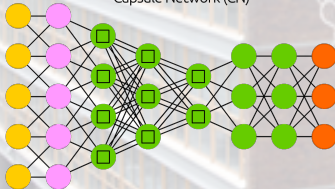
Differentiable Neural Computer (DNC)



Neural Turing Machine (NTM)



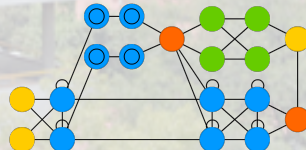
Capsule Network (CN)



Kohonen Network (KN)



Attention Network (AN)



Ejemplo de una red FeedForward



Ingredientes de una red neuronal

Para construir una red neuronal, entonces, se requiere:

- ▶ Un conjunto de datos de entrada X .
- ▶ Determinar un tipo de red neuronal que se acople al problema.
- ▶ Una función de salida ϕ (Softmax, sigmoide, escalón, ...).
- ▶ Una arquitectura $\{K, m_1, \dots, m_K\}$ (tal que $|\theta| = \sum_{k=1}^K m_{k-1} \cdot m_k + \sum_{k=1}^K m_k$).
- ▶ Funciones de activación.
- ▶ Una función de riesgo $R(\theta)$.



Ejemplo de red neuronal

Supóngase que se tiene un conjunto de datos supervisados:

$$\mathcal{S} = \{(x, y) : x \in \mathbb{R}^5, y \in \{1, 2, 3\}\}$$

Buscamos clasificar los ejemplos en 3 clases.

Suponemos que los datos **no son linealmente separables**. Por tanto, requerimos de una red **FeedForward**.



Arquitectura de la red

Definiremos nuestra red con las siguientes propiedades:

1. Una capa oculta con 10 unidades. Esto define una matriz $W^{(1)} \in \mathbb{R}^{10 \times 5}$ y un bias $b^{(1)} \in \mathbb{R}^{10}$.
2. Una función de activación en la capa oculta \tanh .
3. Una función de salida dada por Softmax (por tanto, se trata de una probabilidad de clase).

La salida de la red contará con 3 nodos (uno para cada clase) determinados por:

$$\phi_y(x) = p(Y = y|x)$$



Implementación de la red neuronal

Dado un ejemplo de entrada x , la red calculará:

- Capa oculta ($\mathbb{R}^5 \rightarrow \mathbb{R}^{10}$):

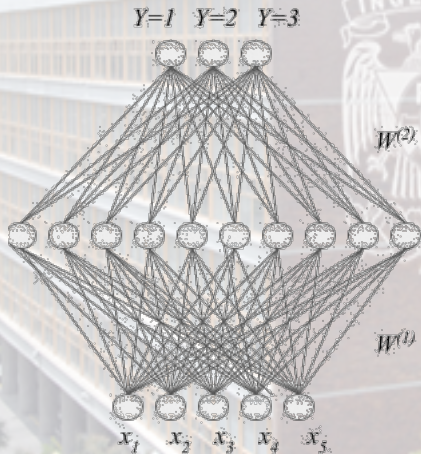
$$h = \tanh(W^{(1)}x + b^{(1)})$$

- Pre-activación de la capa de salida (para una sola clase):

$$a = W^{(2)}h + b^{(2)}$$

- Activación en la capa de salida (probabilidad de una clase):

$$\phi_y(x) = \frac{e^{a_y}}{\sum_{y'} e^{a_{y'}}}$$



Implementación de la red neuronal

Finalmente, se determinará que x pertenece a una clase a partir de la regla:

$$\hat{y} = \arg \max_y \phi_y(x)$$

A todo este paso se le conoce como **Forward**.

La red así constituida tiene un **número de parámetros**

$$|\theta| = (5 \times 10 + 10) + (10 \times 3) + 3 = (50 + 10) + (30 + 3) = 83$$





El problema de aprendizaje consiste en encontrar los parámetros óptimos:

$$\theta = \{W^{(1)}, W^{(2)}, b^{(1)}, b^{(2)}\}$$

¿Cómo se puede realizar esto en redes de este tipo?



References

-  Kurt Hornik, Maxwell Stinchcombe, and Halbert White.
Multilayer feedforward networks are universal approximators.
Neural networks, 2(5):359–366, 1989.
-  Shai Shalev-Shwartz and Shai Ben-David.
Understanding machine learning: From theory to algorithms.
Cambridge university press, 2014.
-  Hao Shen.
A differential topological view of challenges in learning with feedforward neural networks.
arXiv preprint arXiv:1811.10304, 2018.
-  F. Van Veen and S. Leijnen.
The neural network zoo.
<https://www.asimovinstitute.org/neural-network-zoo>.
Accessed: 2021-07-08.



The End

