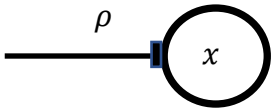
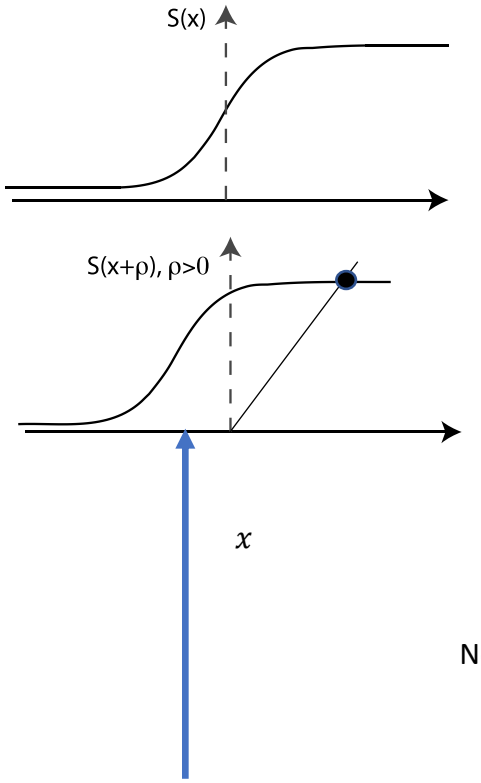
Clase II: redes neuronales

La primera unidad:



$$\frac{dx}{dt} = -x + S(\rho + cx) \quad con \quad S(x) = \frac{1}{1+e^{-x}}$$



Notar que para un rho positivo, el argumento de la sigmoidea ocurre para un x negativo

La estructura de los puntos fijos para distintos valores de los parámetros (vamos disminuyendo ρ)

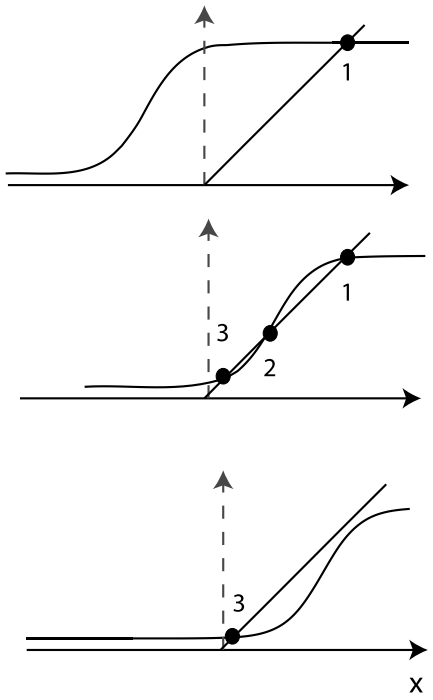
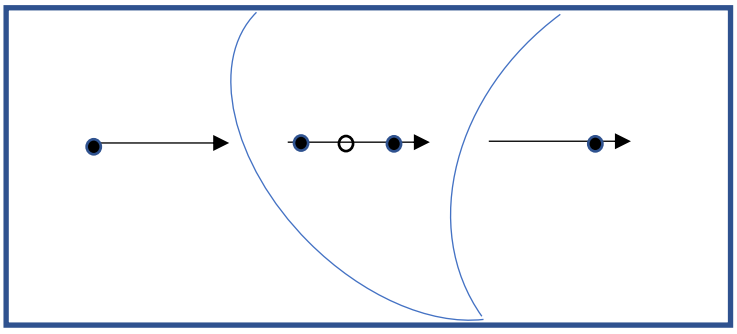


Figura: Puntos fijos de un campo vector Sigmoideo, para distintos valores del parámetro ρ.

Diagrama de bifurcaciones



$$\textbf{\textit{Notacion}}: \quad output = S_c(input) = S_c\left(\sum_{i=1}^{N} W_i x_i\right) = S\left(\sum_{i=1}^{N} W_i x_i\right)$$

Esto es: solo se hace referencia explicita a los inputs externos,

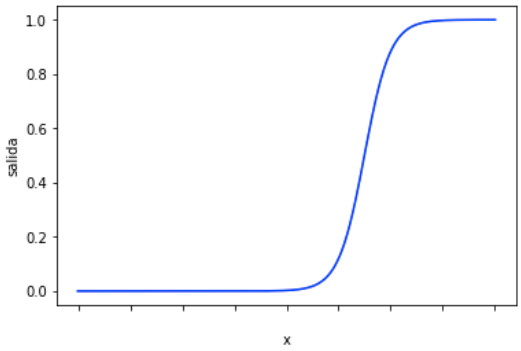
$$\text{net} = \sum_{i=1}^{N} W_i x_i$$

y se olvida la referencia a c (que queda absorbida por la forma funcional de la funcion saturada elegida)

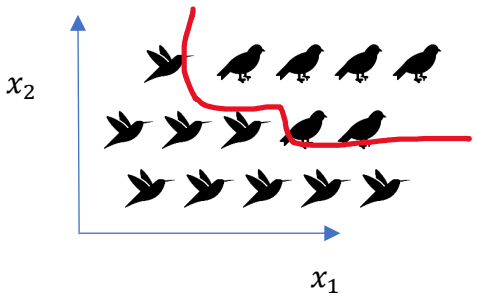
La primera observación sobre la capacidad de unidades neuronales, y su no linealidad, para llevar a cabo tareas de tipo cognitivo, es que

la sigmoidea permite "clasificar"

En la medida en que c, en el modelo dinámico sigmoideo, sea suficientemente chico, un input bajo tendrá respuesta cercana a cero, y un input alto, cercana a uno. **Una clasificación binaria**.



Una clasificación con mas “features” (coloración, latitud de hallazgo)



- Una posibilidad es asumir que entran propiedades **no lineales** $x_1x_2, x_1x_1, x_1x_2 \ldots$ y en la medida en que las “features” sean pocas, esto funciona.

$$output = S(\rho + c_1x_1 + c_2x_2 + c_{11}{x_1}^2 + \cdots)$$

(encontrar la suma adecuada de los datos, con los pesos adecuados, como para que sea posible tomar una decisión se conoce como “**representación**” de los datos).

Si Estamos hablando de una imagen, y la necesidad de clasificar mediante la misma, podemos pensar cada pixel como un escalar, y podemos estar hablando de 5000 features… y pensar en los términos cuadráticos ya escala el problema de un modo computacionalmente inviable.

Sera un chingolo?

Pero antes de hablar de las redes de unidades como soluciones pragmáticas a problemas computacionalmente difíciles, veamos una **aproximación histórica importante**, que consistió en explorar si podían implementarse **puertas lógicas** con estas unidades.

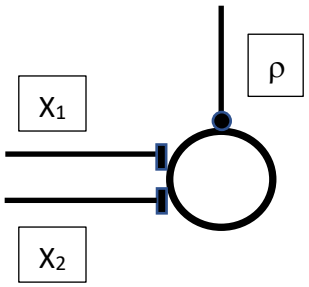
Dos visiones bien distintas sobre que hacer con estas unidades.

Una mosca con 100,000 neuronas pesa un miligramo, consume un mili watt, y vuela, navega, encuentra comida y se reproduce. Una computadora es enorme, pesada, consume un montón, no vuela, no ve, no se reproduce, y consume infinito trabajo de programadores. ¿Que esta mal en esta comparación?

La computadora calcula, ineficientemente, pero es "todo propósito", mientras que una mosca es un dispositivo con una finalidad especifica, que jamás permitirá que le instale un Excel.

<u>Parte de la fantasía en los comienzos de la inteligencia artificial fue si podía implementarse una lógica booleana con unidades tipo neuronales.</u>

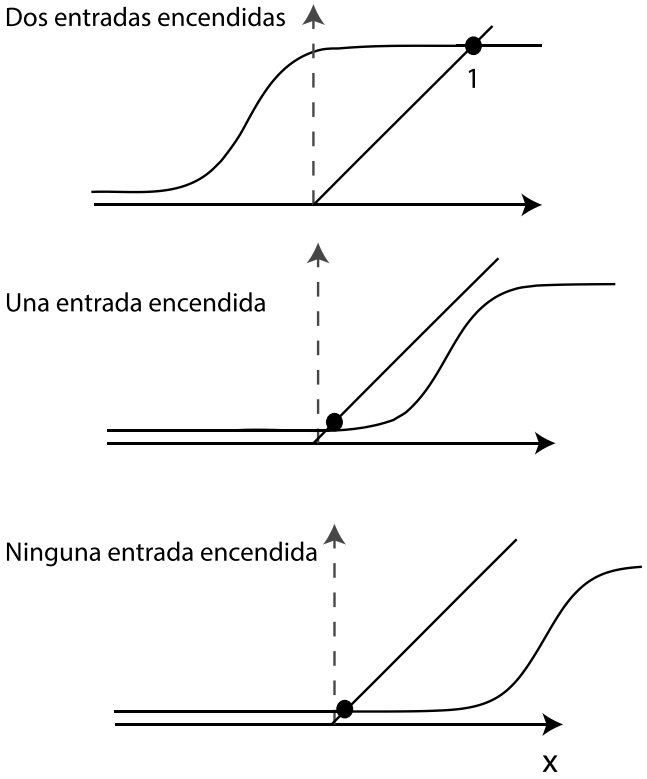
1. **McCulloch y Pitts:** armar puertas lógicas, y mostrar que con neuronas se puede computar.

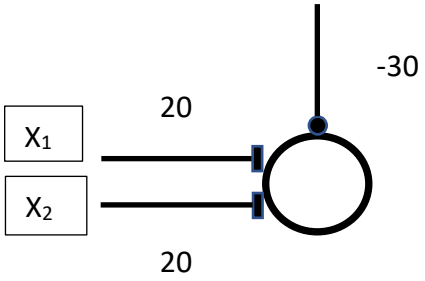


$$\frac{dx}{dt} = -x + S\left(\rho + \sum_{1}^{n} c_j x_j\right)$$

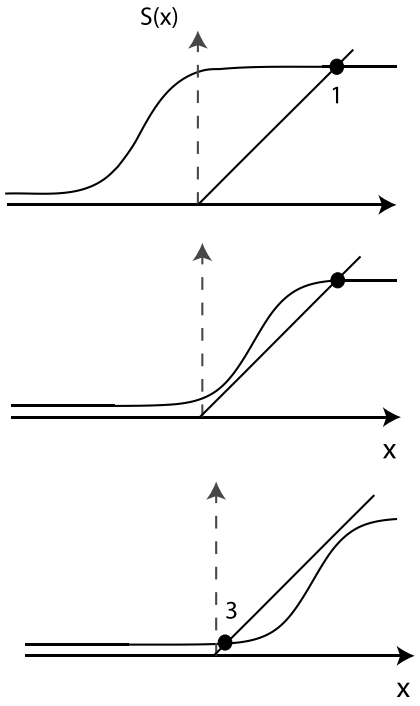
$$output = S\left(\rho + \sum_{1}^{n} c_j x_j\right).$$

Se puede, por ejemplo, armar una puerta lógica “**AND**”, que dará un estado “encendido” cuando ambas entradas estén encendidas, si la entrada ρ es lo suficientemente negativa.

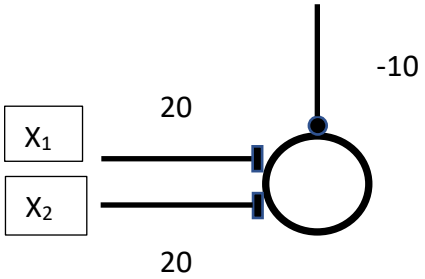




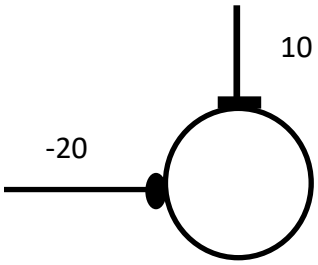
Si quisiéramos, en cambio, realizar una puerta "**OR inclusiva**", podríamos arreglar que el umbral fuera mas pequeño, de modo que con una sola entrada encendida, ya tuviésemos una salida encendida:



Es la figura vemos como basta con una entrada positiva para que ya la salida sea positiva.



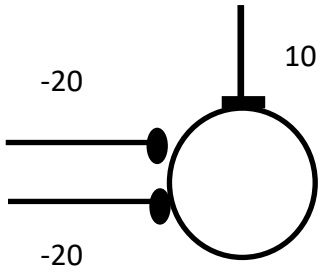
¿Como podría armarse una negación?



$$output = S\left(\rho + \sum_{1}^{1} c_j x_j\right).$$

x_1	output
1	0
0	1

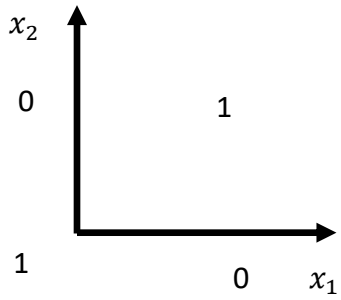
NOT x1 AND NOT x2. (1 solo si x1 y x2 son cero))

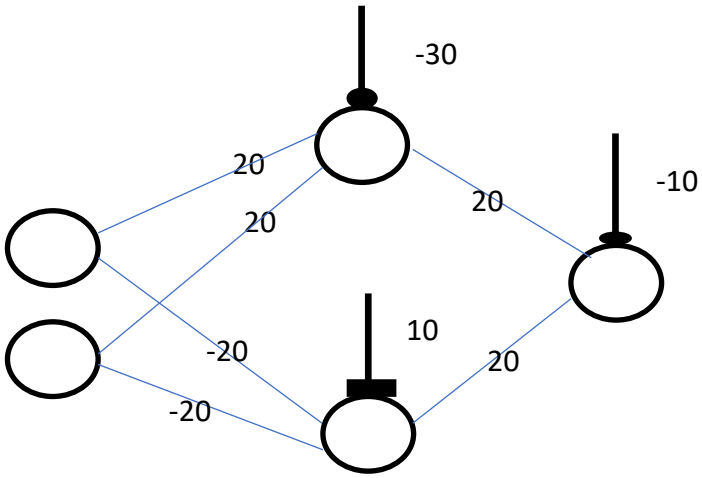


x_1	x_2	a_1
0	0	1
0	1	0
1	0	0
1	1	0

<u>Finalmente, una bastante sutil: x1 XNOR x2.</u>
Esta red da 1 si ambas entradas son uno, o ambas son cero.

Si lo pensamos como un mecanismo de regresión logística, es bastante sutil:





x_1	x_2	a_1 (AND)	a_2 (Nx1 and Nx2)	Output (x1 or x2)
0	0	0	1	1
0	1	0	0	0
1	0	0	0	0
1	1	1	0	1

Este ejemplo, además de introducir una puerta lógica, pone en evidencia como con mas capas, se logran funciones no triviales de las entradas.

Armar una red de puertas lógicas, es pensar a la red como una computadora, capaz de implementar algoritmos. **Que tal si en cambio, pensamos a una red como un conjunto de unidades, ¿y le enseñamos por medio de "ejemplos"?**

No escribimos un código que implemente un calculo para llevar a cabo una tarea: simplemente vamos acomodando las conectividades de modo tal que, ante ciertos inputs, logre determinados outputs.

2. **Rosenblatt:** el perceptron.

Con la misma matemática, pero una concepción radicalmente diferente (y mucho mas cercana al actual proceso de Deep learning), Rosenblatt propuso el perceptron: una "red" de un elemento con una capa de entradas x_i, que apropiadamente sumadas, entran a una unidad no lineal (output tipo sigmoidea):

$$out(t) = S(\sum_{i=1}^{n} W_i \, x_i)$$

Lo interesante es que no diseñamos, como en el caso de las puertas lógicas, los pesos para tener una salida dada, **si no que sometemos a la red a un aprendizaje**, cuya regla es:

$$\delta w_i = \alpha \, \delta x_i$$

$$\delta = salida - maestro$$

(0 si es correcto, y +1 o -1 si no lo es)

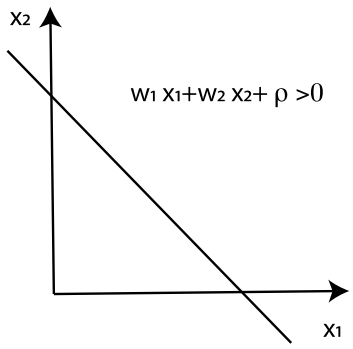


Figura: El tipo de clasificación que puede aprender a hacer un perceptron.

La limitación de esta configuración: solo puede separar lo que es linealmente separable.

Para poder clasificar distribuciones mas complejas, había que combinar los inputs.
Exactamente igual que cuando tuvimos que armar la puerta **xnor.**

Muchas capas podrían lidiar co problemas complejos,
pero el problema era **como entrenar a tal dispositivo.**

Redes de propagación hacia delante y su entrenamiento
(Rumelhart, Hinton, Williams, Nature 1986)

La red que mostramos en la Figura se conoce como una red de propagación hacia delante ("feedforward"). Está constituida por un conjunto de unidades, cada una de las cuales responde ante las entradas según alguna función no lineal, que podría ser simplemente $output = S_c\left(\rho + \sum_1^n c_j x_j\right)$. Puede pensarse que cada unidad convergió rápidamente a la solución estacionaria $\frac{dx}{dt} = -x + S\left(\rho + \sum_1^n c_j x_j + cx\right) = 0$.

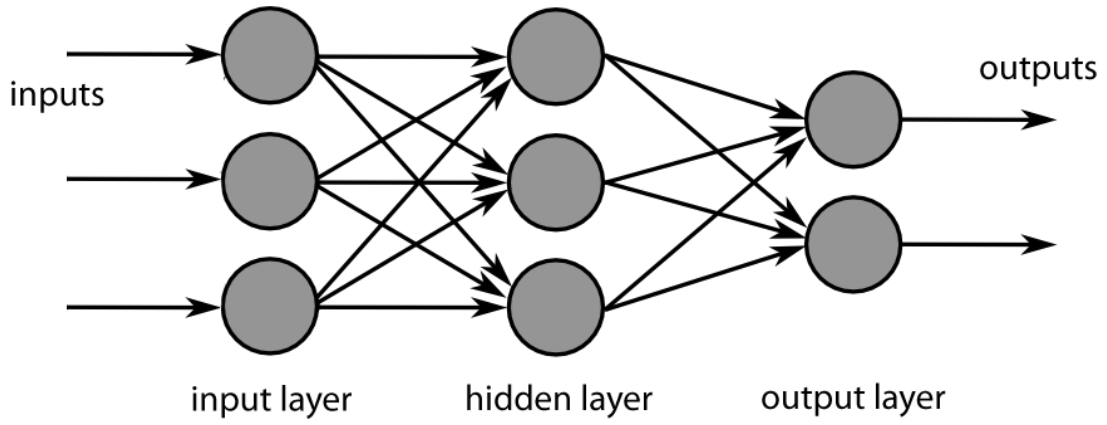
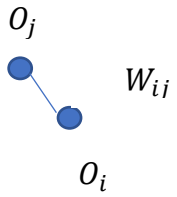


Figura 2. Una red de propagación hacia delante con una capa escondida, una entrada de 3 unidades y una salida de dos. Clasificará objetos caracterizados por tres números, entre dos clases posibles.

Vamos a ver un modo de ajustar los coeficientes que **explote la naturaleza composicional** de una red multicapa, en la cual cada nivel tiene como salida, la entrada del siguiente nivel.

El asunto central es como emplear la diferencia entre la expectativa y lo obtenido al evaluar el input por medio de la red, para guiar en la modificación de los parámetros.

$$\Delta W_{ij} = -\eta \frac{\partial E}{\partial W_{ij}},$$



$$\frac{\partial E}{\partial W_{ij}} = \frac{\partial E}{\partial O_j}\frac{\partial O_j}{\partial net_j}\frac{\partial net_j}{\partial W_{ij}}.$$

$$\frac{\partial net_j}{\partial W_{ij}} = O_i.$$

$$O_j = S(net_j) = 1/(1 + e^{-net_j}),$$

$$\frac{\partial O_j}{\partial net_j} = O_j(1 - O_j).$$

Si estamos en la ultima capa, el primer termino se escribe asi:

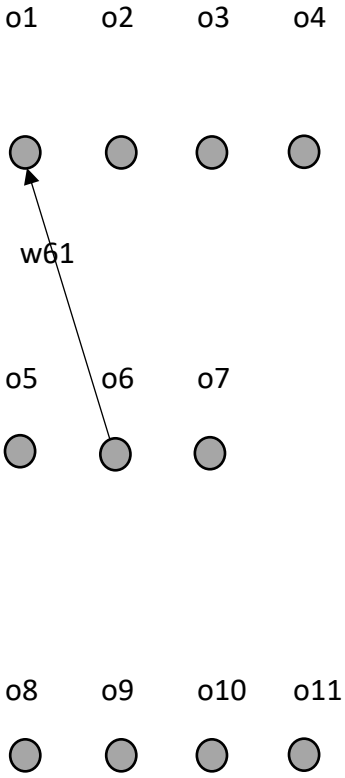
$$\frac{\partial E}{\partial O_j} = (y^j_{elemento} - O_j),$$

Anteúltima capa, el error cambia al cambiar el output de la unidad j, ya que cambian todos los outputs de la ultima capa. De ese modo:

$$\frac{\partial E}{\partial O_j} = \sum_{\substack{l, el\ índice\ de \\ todos\ los\ elementos \\ de\ la\ última\ capa}} \frac{\partial E}{\partial O_l}\frac{\partial O_l}{\partial net_l}\frac{\partial net_l}{\partial O_j}$$

Los valores de la capa anterior

Ejemplo 1.

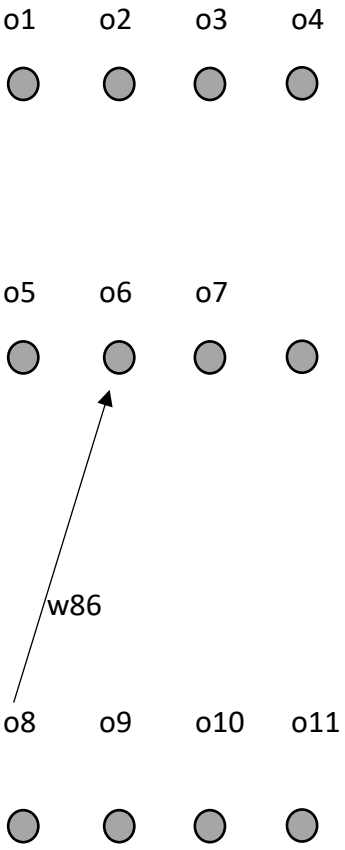


$$\frac{\partial E}{\partial W_{ij}} = \frac{\partial E}{\partial O_j}\frac{\partial O_j}{\partial net_j}\frac{\partial net_j}{\partial W_{ij}}.$$

$$\frac{\partial E}{\partial W61} = (o1 - t1)\big(o1(1 - o1)\big)o6$$

Solo depende de los "O"!!!

Ejemplo 2.



$$\frac{\partial E}{\partial W86} = \boldsymbol{\frac{\partial E}{\partial O6}} \frac{\partial O6}{\partial net6} \frac{\partial net6}{\partial W86} = \boldsymbol{\frac{\partial E}{\partial O6}} \big(o6(1-o6)\big) o8$$

$$\boldsymbol{\frac{\partial E}{\partial O6}} = \frac{\partial E}{\partial O1}\frac{\partial O1}{\partial net1}\frac{\partial net1}{\partial O6} + \frac{\partial E}{\partial O2}\frac{\partial O2}{\partial net2}\frac{\partial net2}{\partial O6} + \frac{\partial E}{\partial O3}\frac{\partial O3}{\partial net3}\frac{\partial net3}{\partial O6} + \frac{\partial E}{\partial O4}\frac{\partial O4}{\partial net4}\frac{\partial net4}{\partial O6}$$

$$\frac{\partial E}{\partial O6} = (O1 - t1)\big(O1(1-O1)\big)W16 + (O2 - t2)\big(O2(1-O2)\big)W26 + (O3 - t3)\big(O3(1-O3)\big)W36 + (O4 - t4)\big(O4(1-O4)\big)W46$$

Solo depende de los "O"!!!

Los auto-codificadores.

Un auto codificador es una red neuronal artificial diseñada para lograr una representación eficiente de los datos, de modo auto-supervisado. La Figura muestra una estructura de auto codificador.

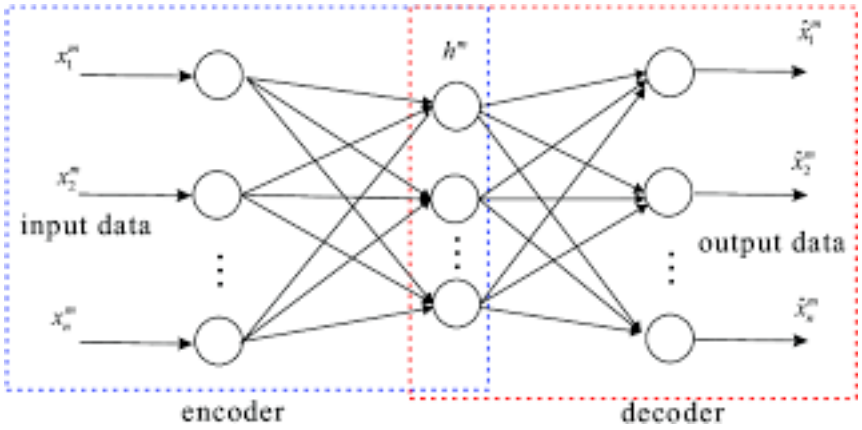
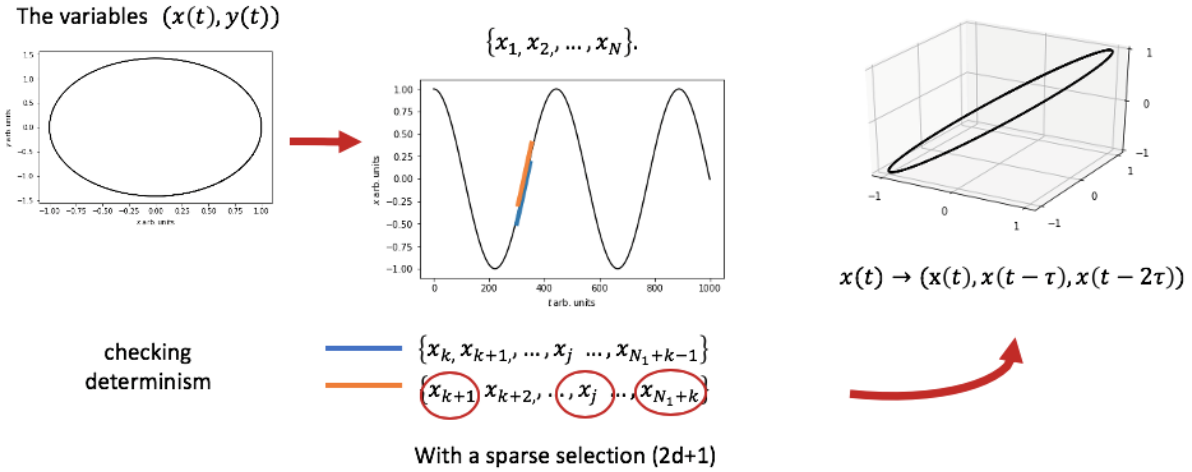


Figura . La estructura de un auto codificador. Un cuello de botella separa dos copias simétricas de una red. Se le solicita a la red que los parámetros sean tales que la salida sea igual a la entrada.

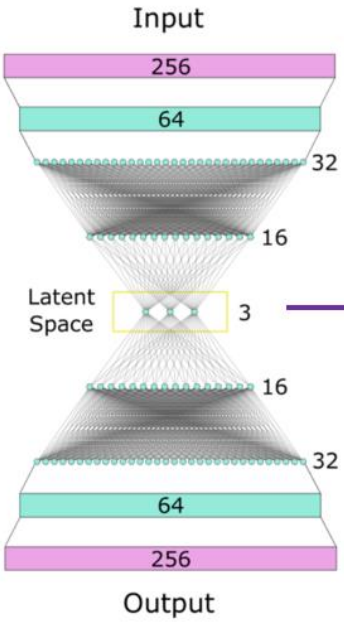
¿Por que este "compresor de información" puede reproducir los datos, pese a la dramática compresión?



An interesting idea from the tool-box of ML

You ask the system to reproduce, as output, the input.

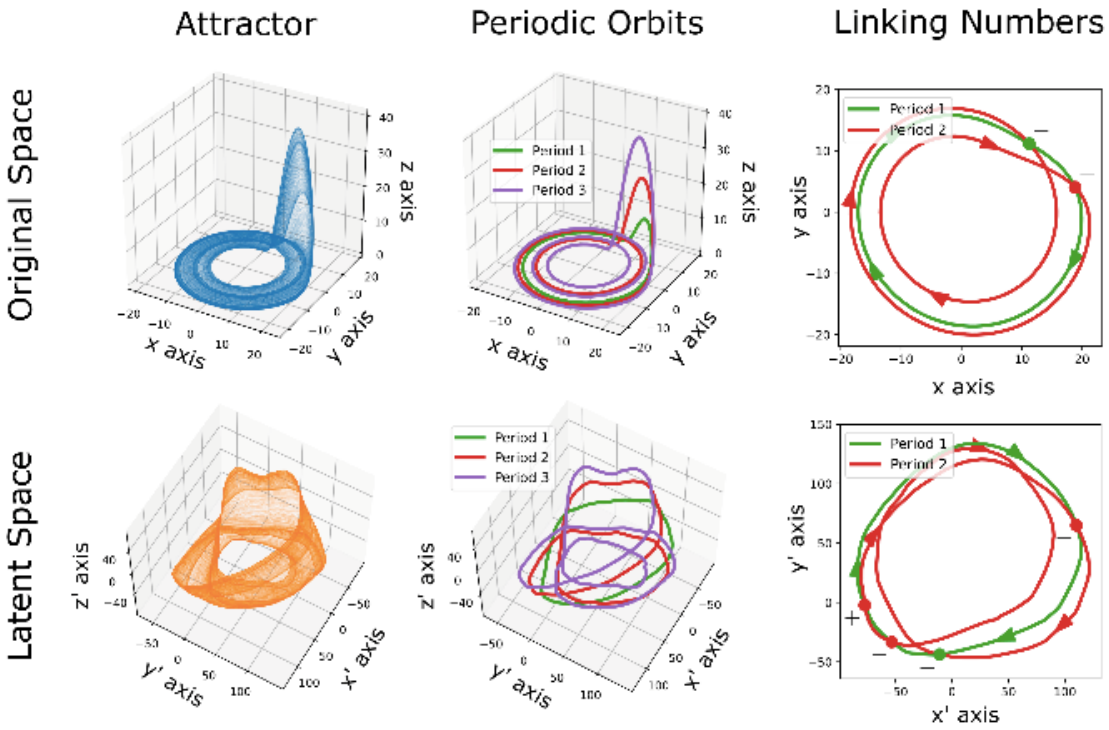
This leads to a compression in the middle layer



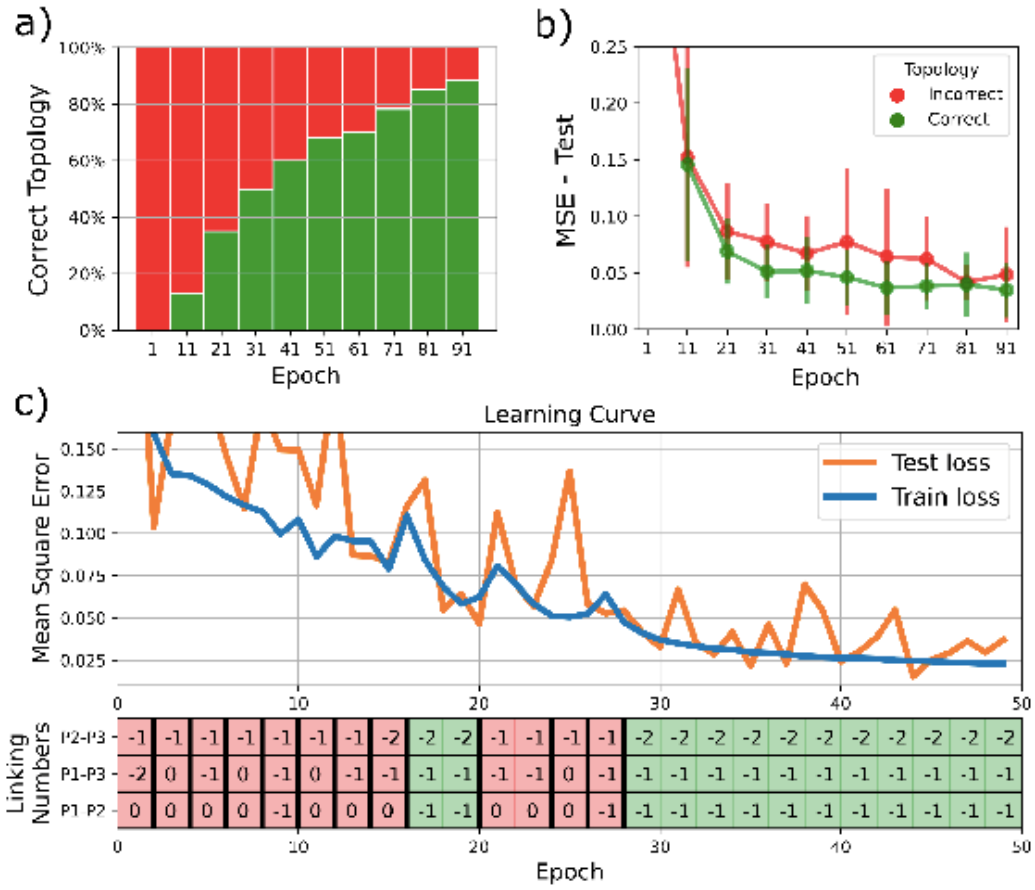
The question is whether this latent space constitutes an embedding

Does it preserve the flow's topology

(With Gonzalo Uribarri)


Attractor
Periodic Orbits
Linking Numbers
Original Space
Latent Space
Period 1
Period 2
Period 3
x axis
y axis
z axis
x' axis
y' axis
z' axis

How do the original phase space and the reconstructed latent space compare?



El éxito de la compresión se debe a la posibilidad de generar un embedding de los datos. Este resulta topológicamente equivalente al obtenido mediante el teorema de Takens.

En el ejemplo que discutimos recién, analizamos una señal temporal proveniente de una simulación numérica de las ecuaciones de Rossler. Cada segmento de la señal, de un número *n* de muestras, era mapeado por la parte "cuello de botella" del auto codificador en un punto del espacio multi-valuado en el cual cada componente es el valor que toma cada una de las unidades de la capa "cuello de botella". En lugar de analizar una señal temporal escalar, como la que constituye la evolución temporal una de las variables de un sistema dinámico en el ejemplo anterior, podríamos estar interesados en analizar una película. En este caso tenemos también una serie temporal, pero no de un escalar, si no de un arreglo de números, que representan los valores de los pixels en el eje x, en el eje y, y eventualmente, tres copias de esa información si la película está en RGB. Así, al asociar la información de la película a la primera capa del auto codificador, cada elemento a la entrada en un arreglo de n frames, cada uno de los cuales son 3 juegos de arreglos de números que expresan el valor del pixel en cada una de las *m_ancho* times *m_alto* posiciones.

En la Figura 5 a se muestran unos frames de una película que muestra parte del aparato vocal de un ave mientras canta (Figura 5.a.). La película analizada permite visualizar 3 oscilaciones completas de los labios siríngeos (esto es, es una película bastante repetitiva). El espacio de fases de esa película se muestra en la Figura 5.b. Se puede observar un ciclo límite.

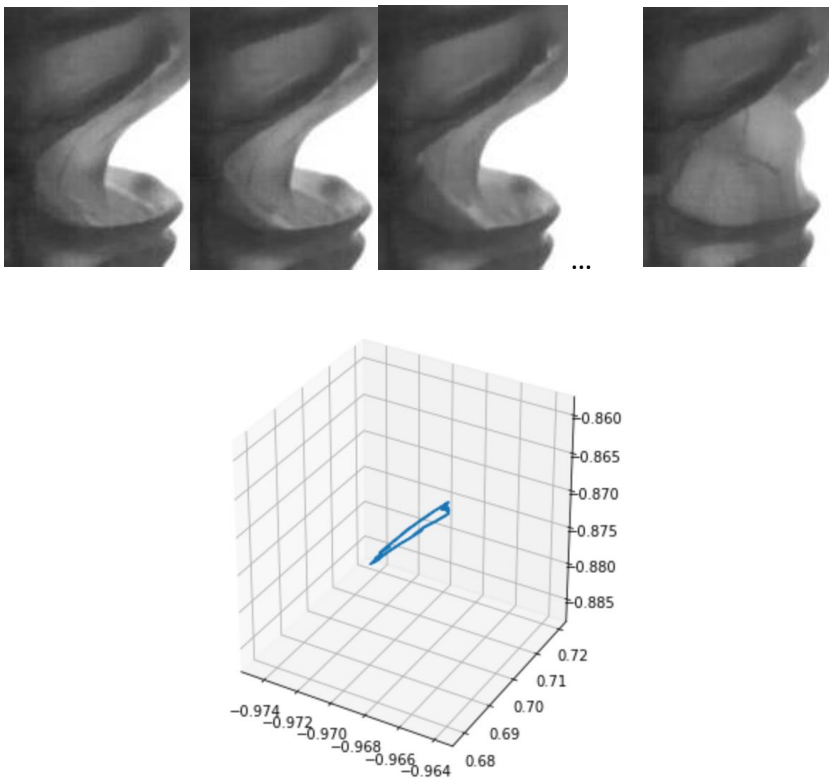


Figura 5. Una película, y el espacio de fases reconstruido con el auto codificador

En este ejemplo, 105 elementos fueron empleados para entrenar la red, y 19 para validarla. Cada elemento está constituido por 10 frames, de tres colores, y cada nivel de esos tres, correspondían a una resolución espacial de 120 por 167. El autocodificador tenia capas de 50, 30, 10, 3, 10, 30 y 50 respectivamente. El entrenamiento se lleva a cabo a lo largo de 20 épocas, y el éxito del "cuello de botella" de la dimensión empleada en esta arquitectura se basa en la convergencia de los errores cuadráticos a valores despreciables, como se ilustra en la Figura 6.

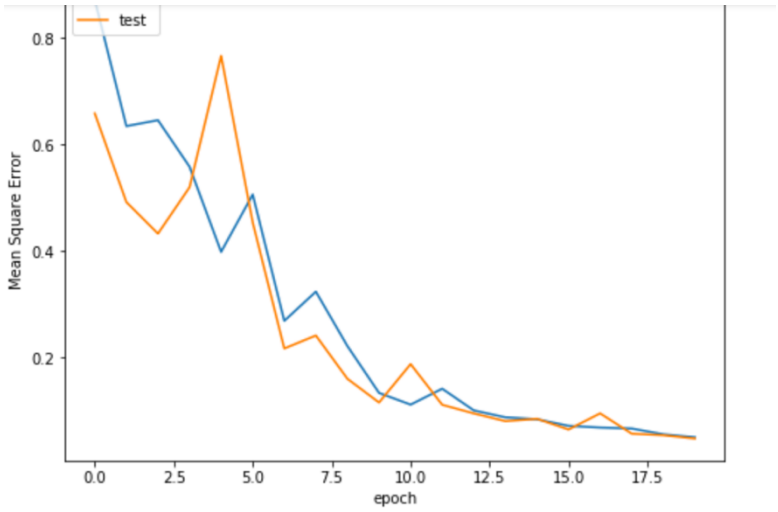


Figura 6. Los errores cuadráticos medios para el conjunto de entrenamiento y el de la validación.