

## Kolmogorov Arnold Networks

Son redes en las cuales en lugar de entrenar cambiando los pesos entre neuronas que comparten una funcion de activacion, se entrena para cambiar las funciones de activacion mismas.

MLP → KAN

---

# KAN: Kolmogorov–Arnold Networks

---

**Ziming Liu<sup>1,4\*</sup> Yixuan Wang<sup>2</sup> Sachin Vaidya<sup>1</sup> Fabian Ruehle<sup>3,4</sup>**  
**James Halverson<sup>3,4</sup> Marin Soljačić<sup>1,4</sup> Thomas Y. Hou<sup>2</sup> Max Tegmark<sup>1,4</sup>**

<sup>1</sup> Massachusetts Institute of Technology

<sup>2</sup> California Institute of Technology

<sup>3</sup> Northeastern University

<sup>4</sup> The NSF Institute for Artificial Intelligence and Fundamental Interactions

Para MLP

## Teorema de la aproximacion universal

Una red neuronal feedforward, con una capa oculta constituida por un numero finito de neuronas, puede aproximar a cualquier funcion continua en un rango finito del input, con cualquier grado de precision, si la funcion de activacion es no lineal, acotada y continua

En principio, el numero de unidades en esa capa oculta puede ser enorme...

La solucion: poner muchas capas.

Para KAN

toda funcion continua multivaluada puede, en un intervalo acotado, escribirse como una composicion finita de suma de funciones de una variable.

$$f(x_1, x_2, \dots, x_n) = \sum_{q=1}^{2n+1} \Phi_q \left( \sum_{p=1}^n \phi_{q,p}(x_p) \right)$$

Un motivo por el que no se uso esto por mucho tiempo en redes...

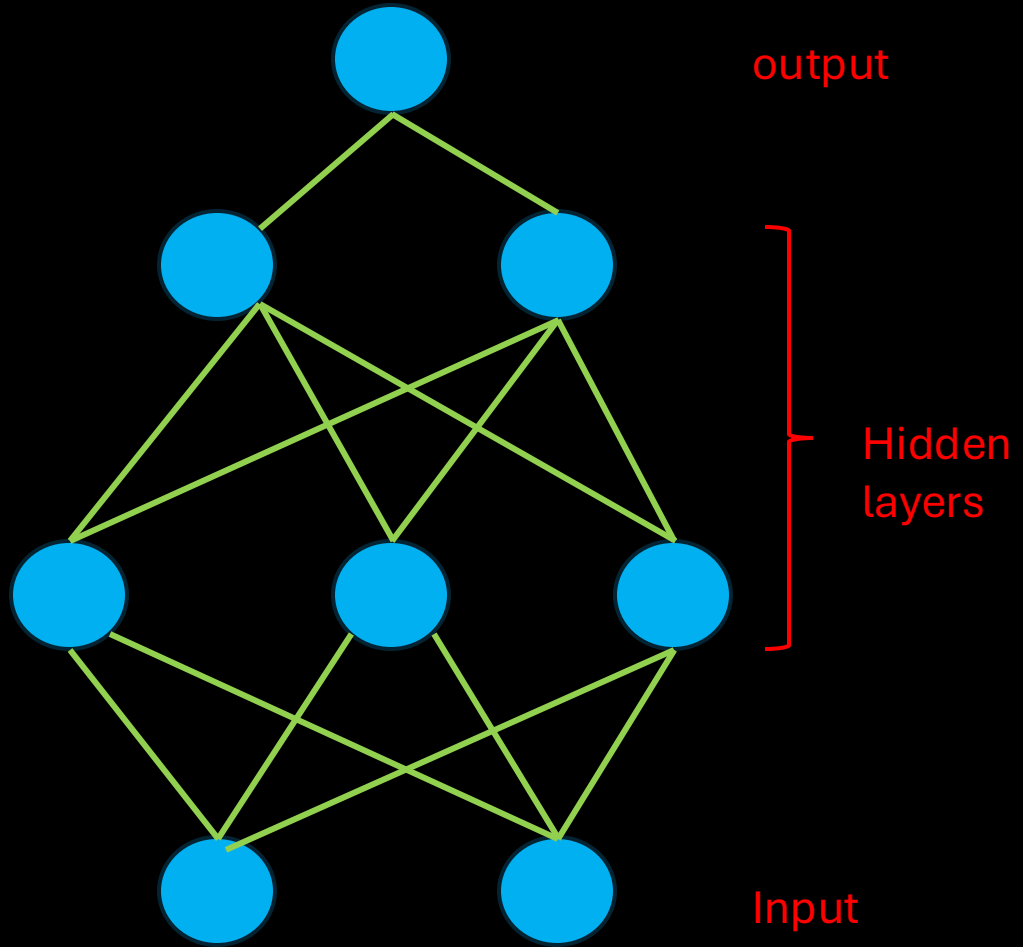
La eventual complejidad de las funciones. Solucion... muchas capas

$$f(x_1, x_2, \dots, x_n) = \sum_{q=1}^{2n+1} \Phi_q \left( \sum_{p=1}^n \phi_{q,p}(x_p) \right)$$

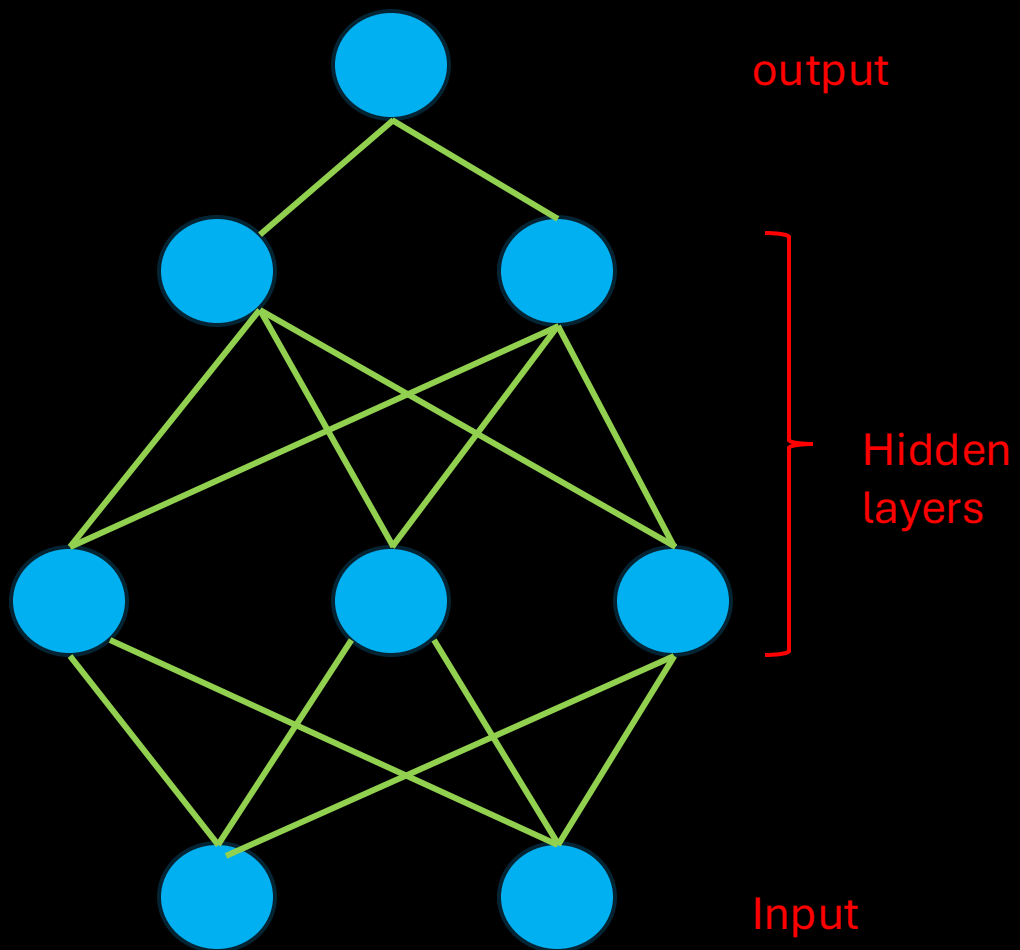
$$xy = \frac{1}{2}z^2 - \frac{1}{2}x^2 - \frac{1}{2}y^2$$

$$z=x+y$$

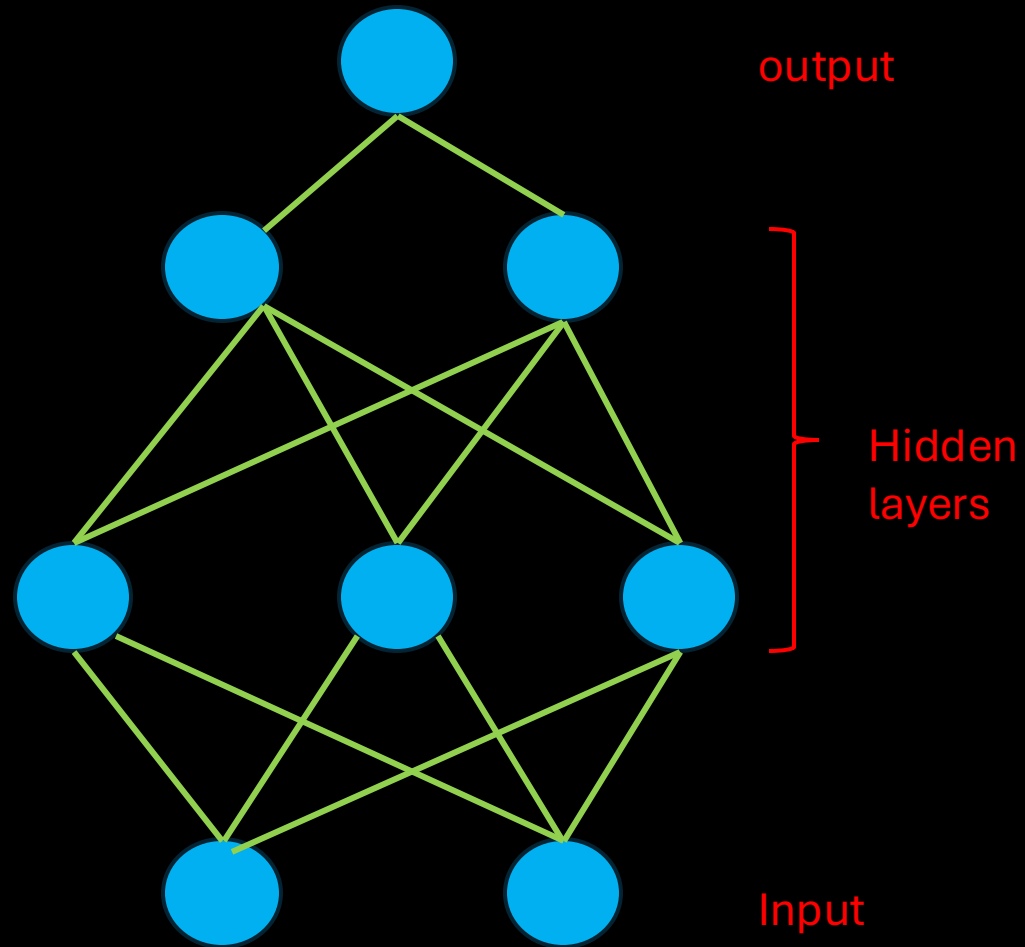
# MLP (Multi layer perceptron)



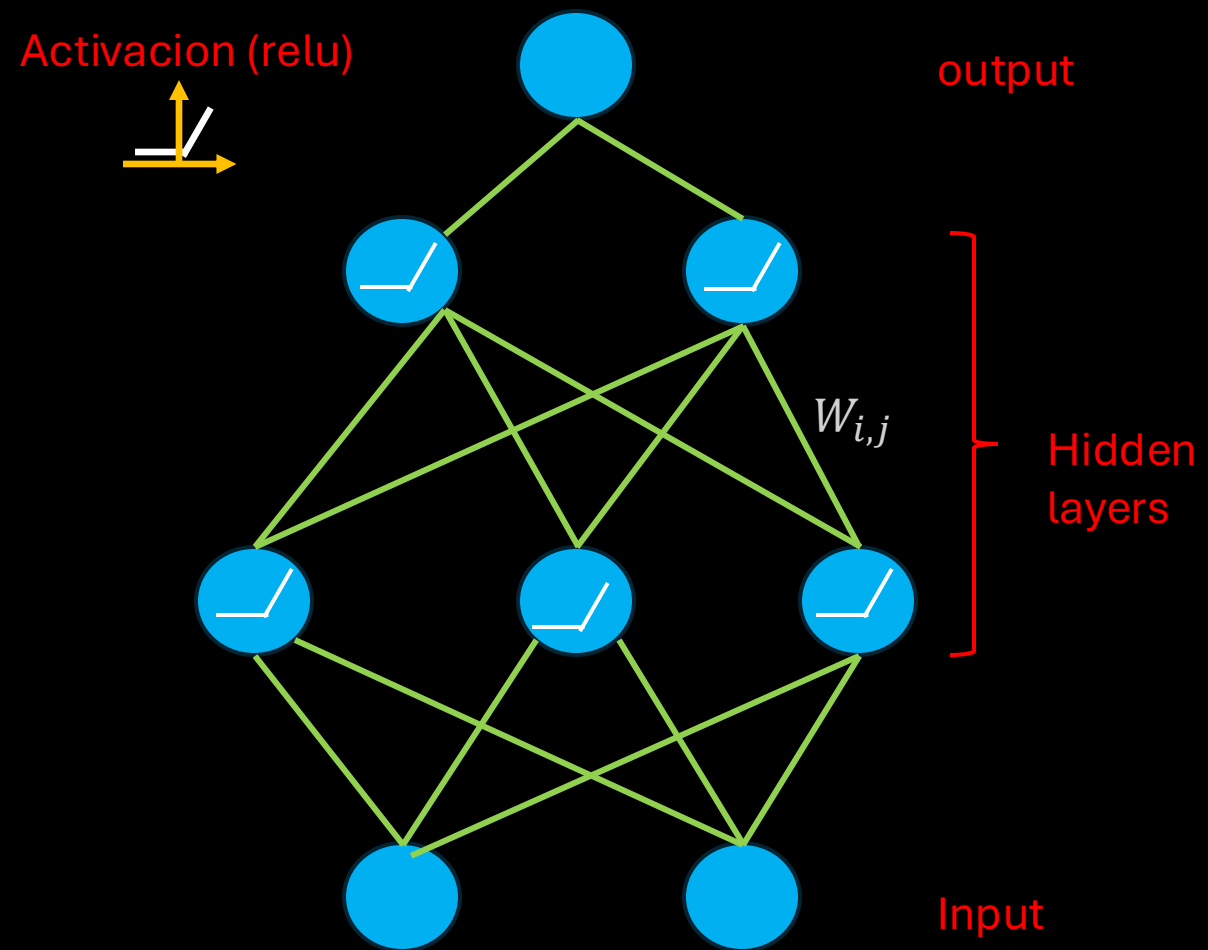
MLP (Multi layer perceptron)



KAN (Kolmogorov Arnold Network)

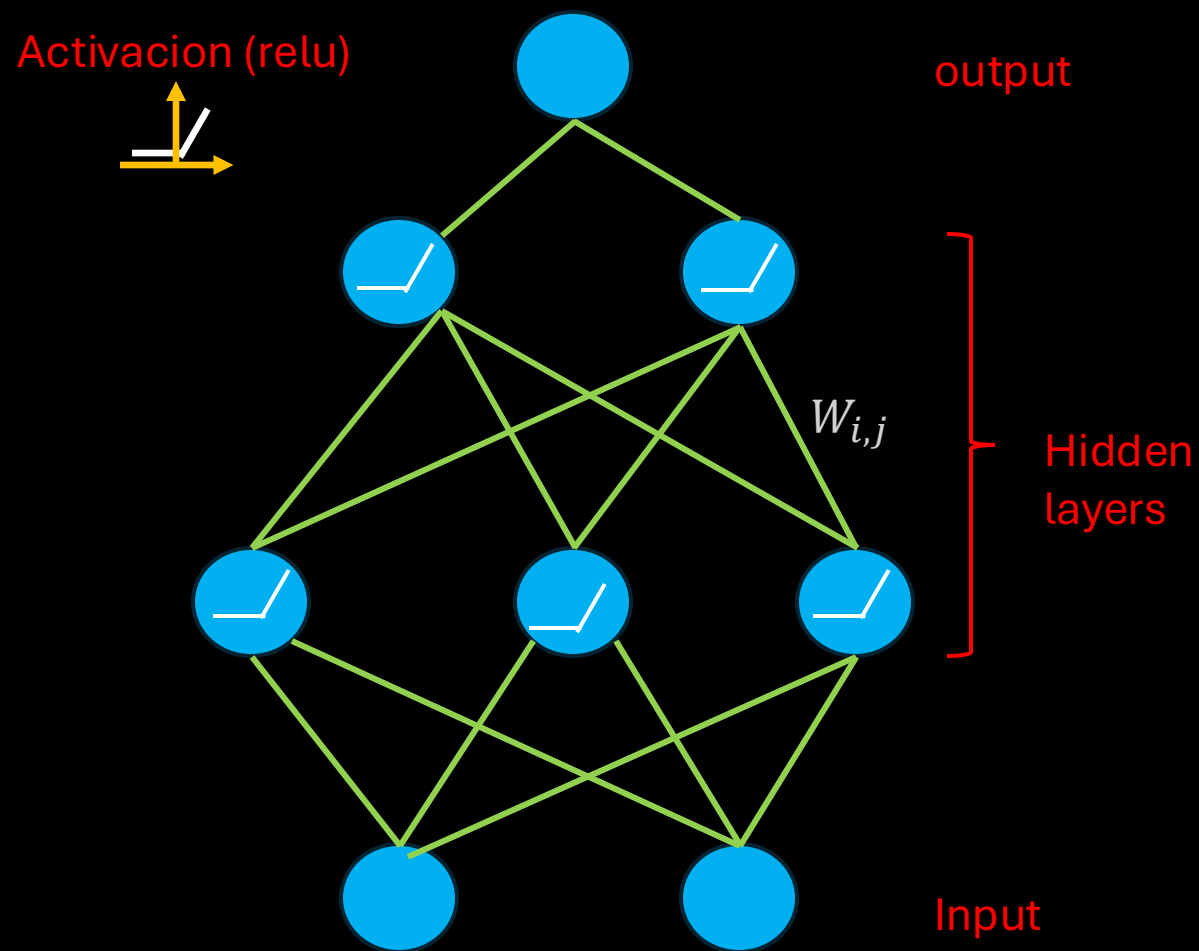


## MLP (Multi layer perceptron)



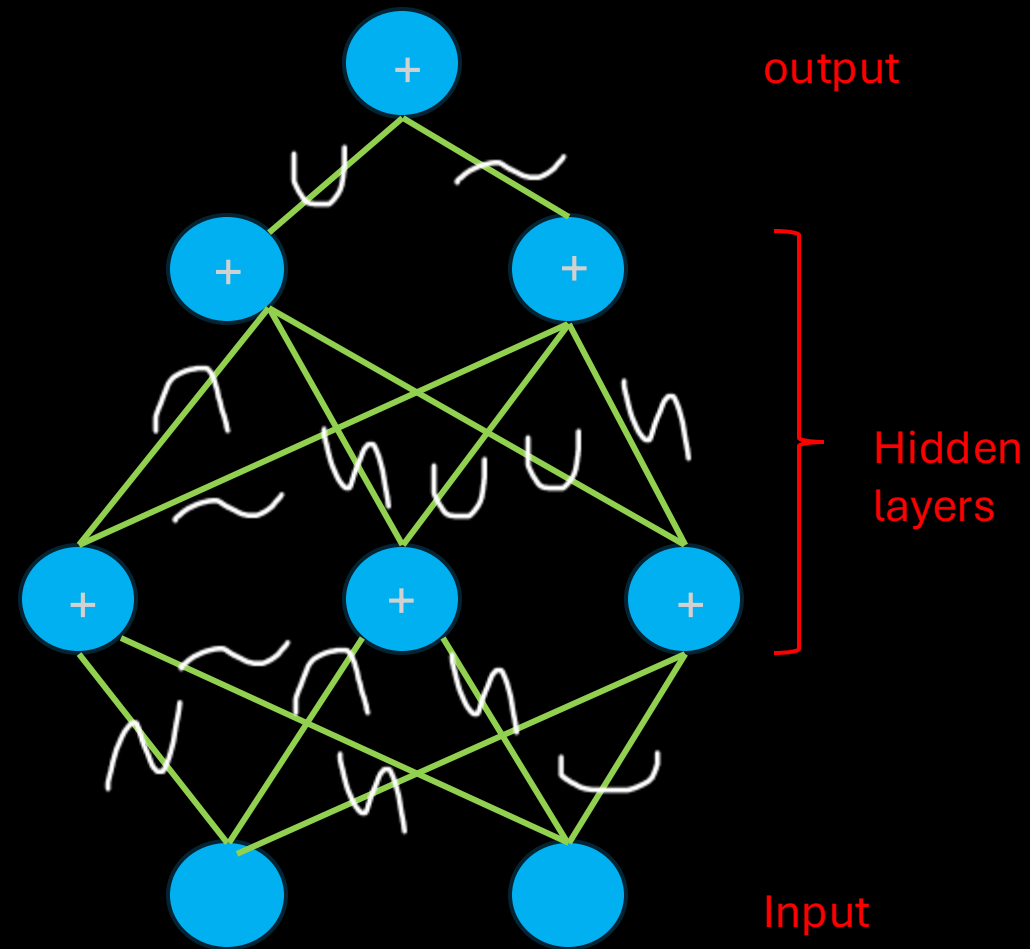


MLP (Multi layer perceptron)



Activaciones fijas, entrenamos pesos

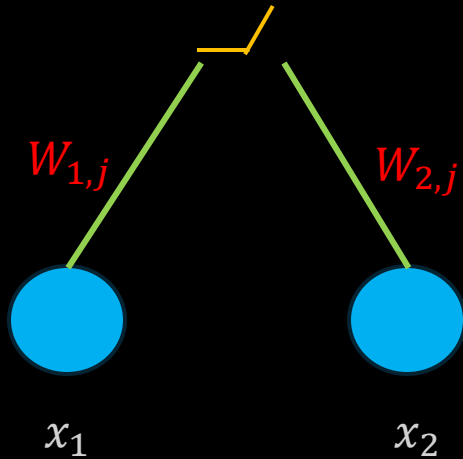
KAN (Kolmogorov Arnold Network)



Pesos fijos, entrenamos funciones

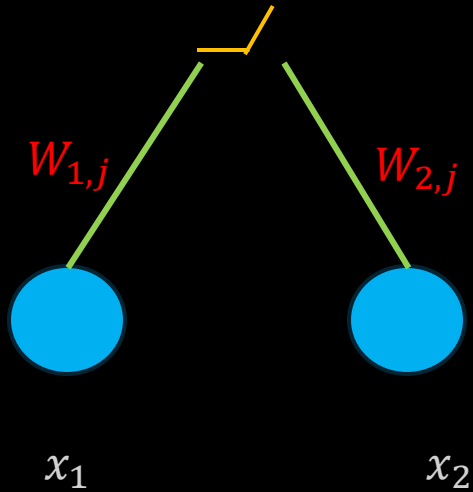
Los parametros entrenables en una MLP

$$Output = \text{ReLu}(W_{1,j}\text{ReLu}(x_1) + W_{2,j}\text{ReLu}(x_2))$$



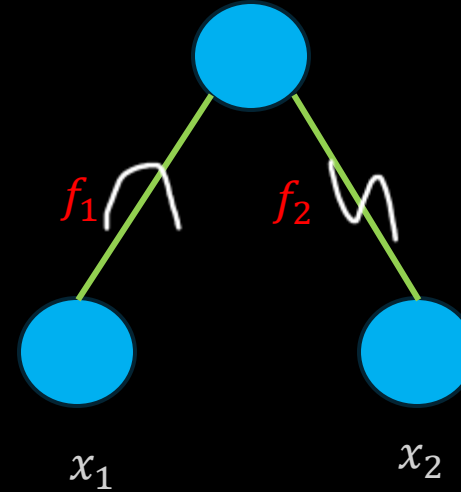
Los parametros entrenables en una MLP

$$Output = \text{ReLu}(W_{1,j}\text{ReLu}(x_1) + W_{2,j}\text{ReLu}(x_2))$$



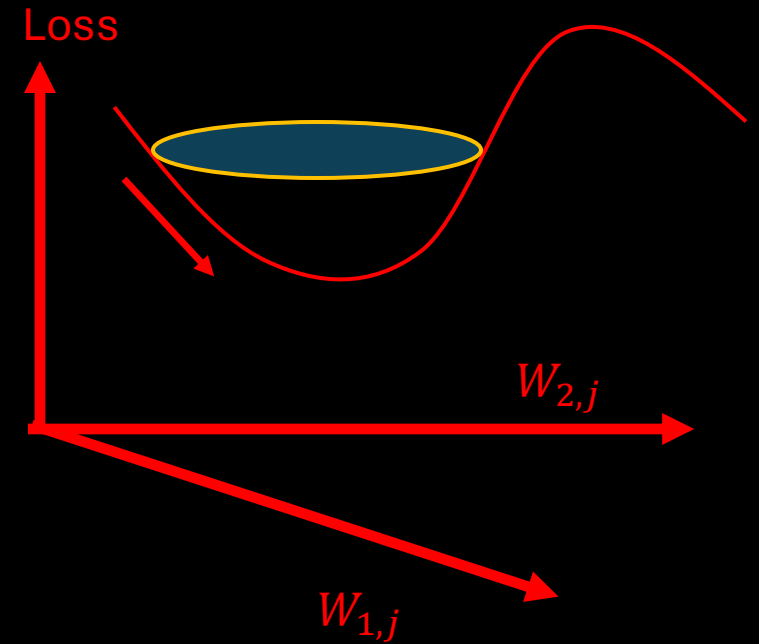
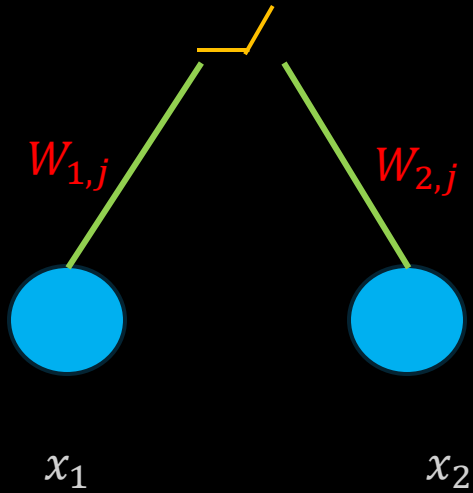
Los parametros entrenables en una MLP

$$Output = 1f_1(x_1) + 1f_2(x_2)$$



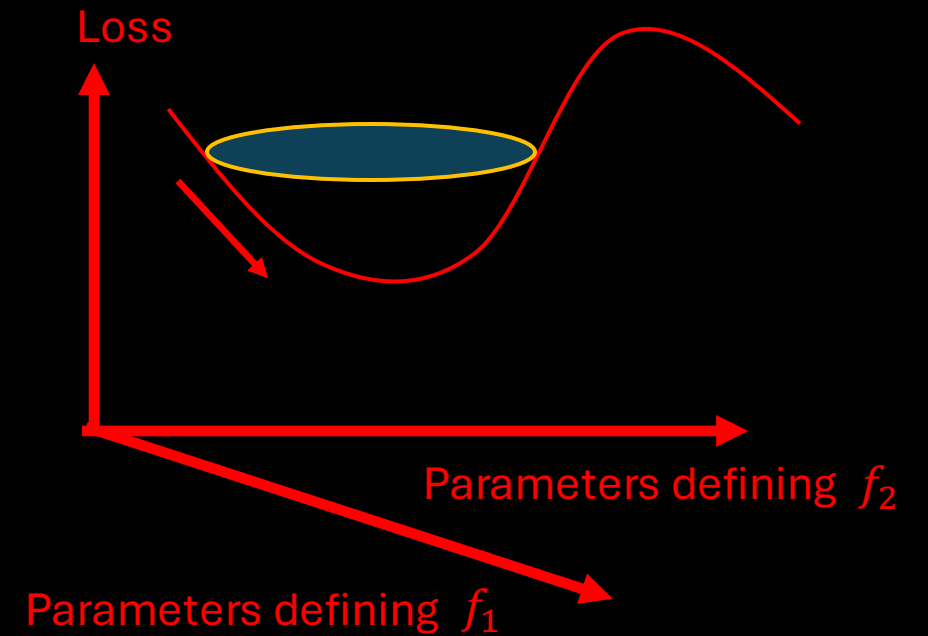
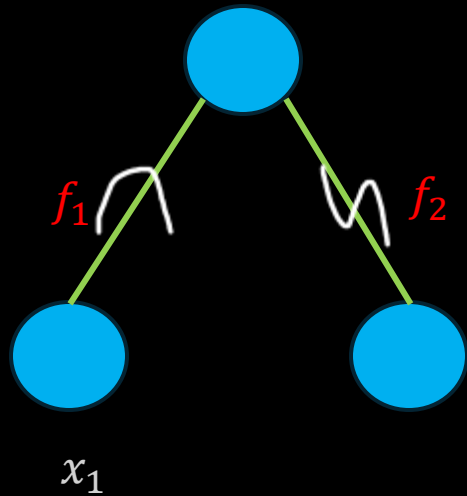
Los parametros entrenables en una MLP

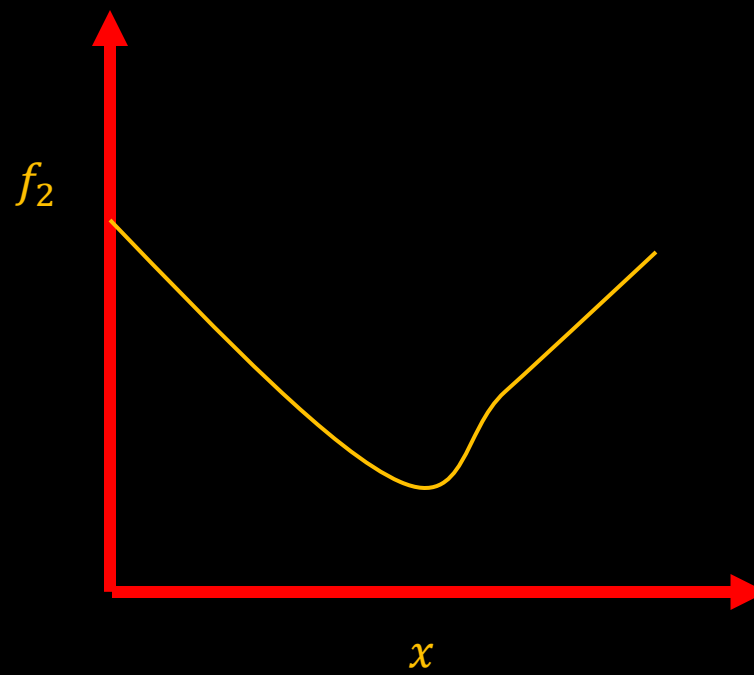
$$\text{Output} = \text{ReLU}(W_{1,j}\text{ReLU}(x_1) + W_{2,j}\text{ReLU}(x_2))$$



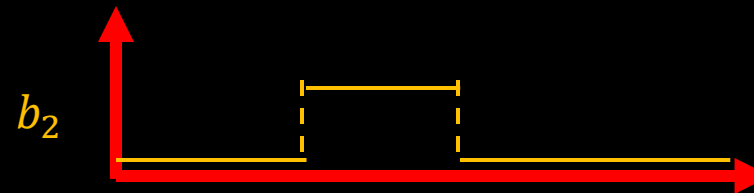
Los parametros entrenables en una MLP

$$\text{Output} = 1f_1(x_1) + 1f_2(x_2)$$

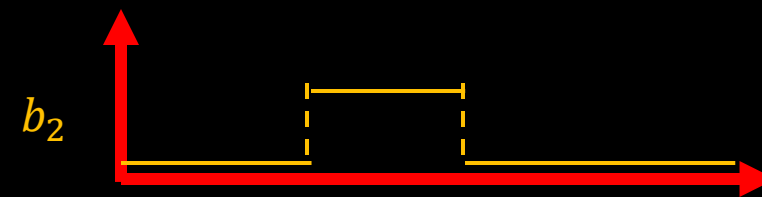
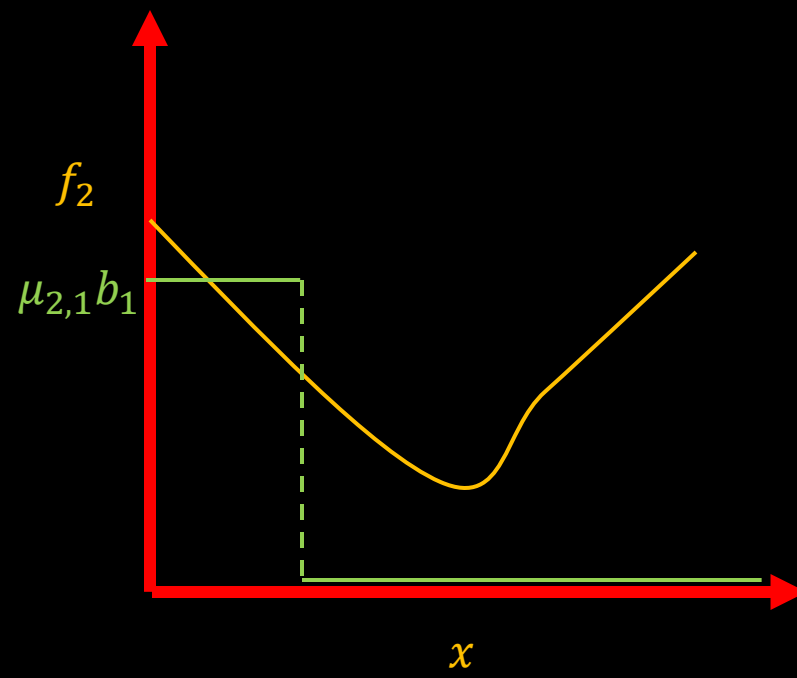


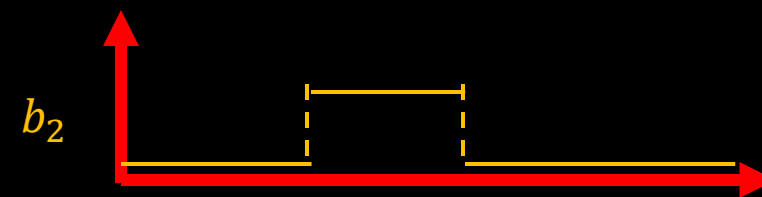
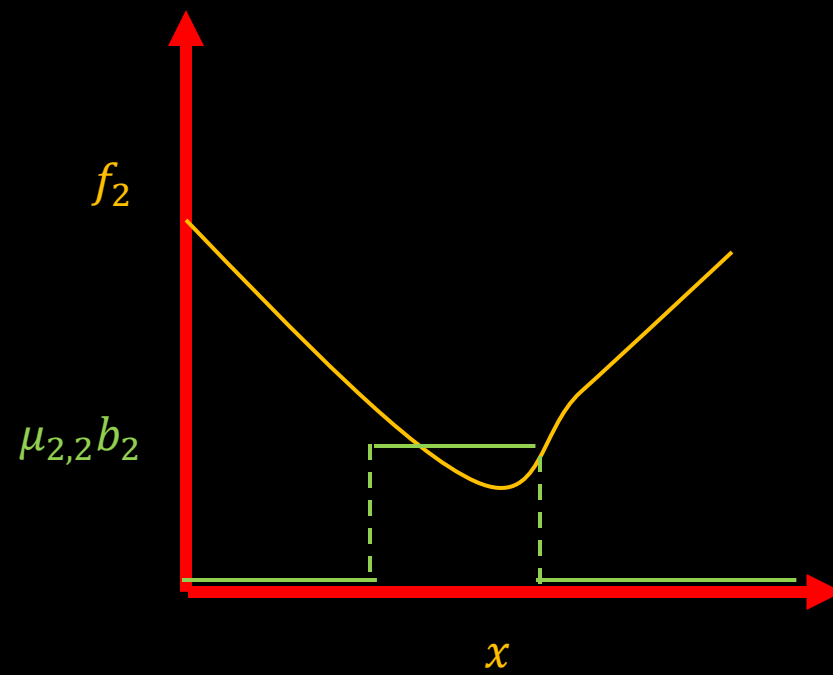


Que querría decir  
“parámetros que definen a  $f_1$ ”?

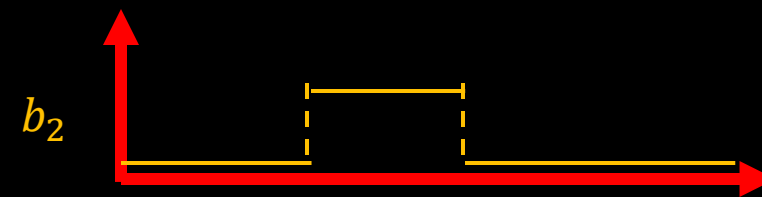
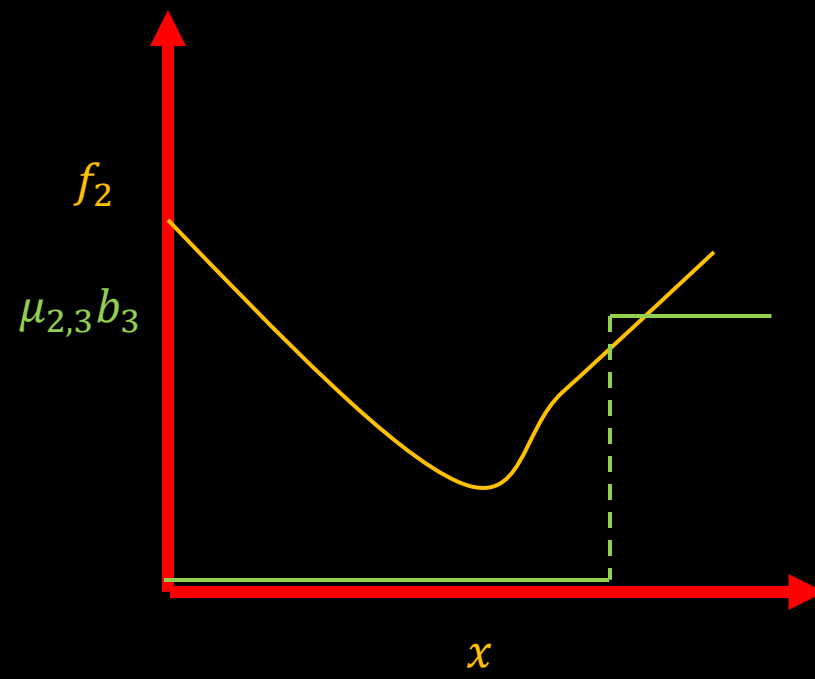


Tomemos como “base” de  
nuestra aproximacion a  
este conjunto de funciones

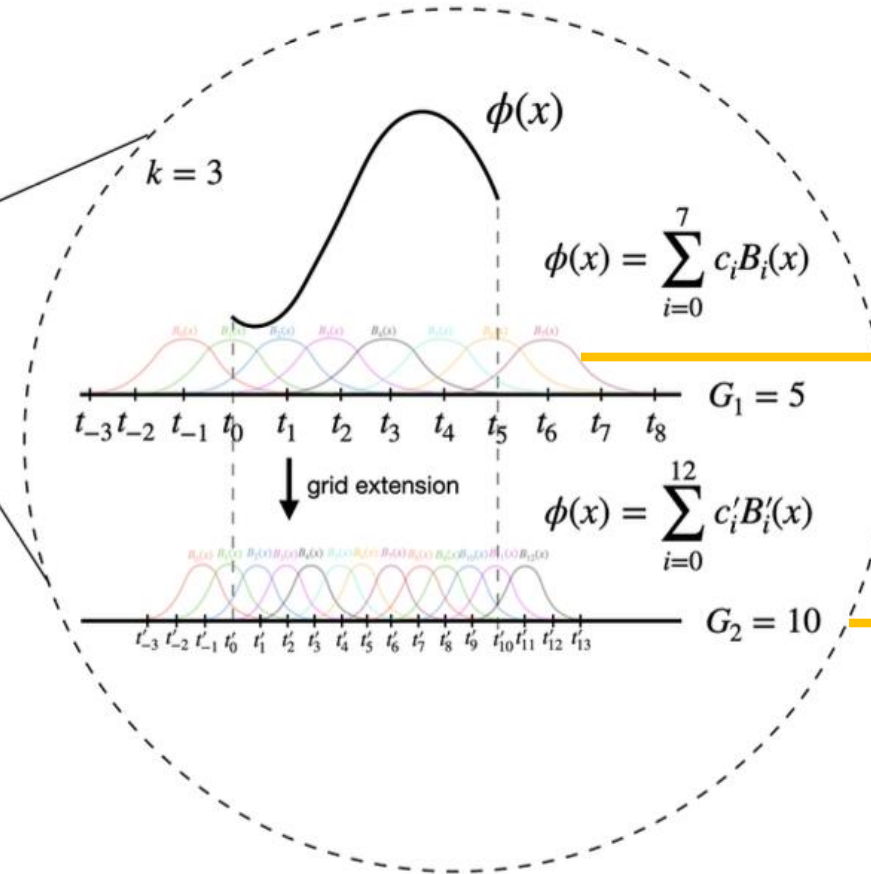
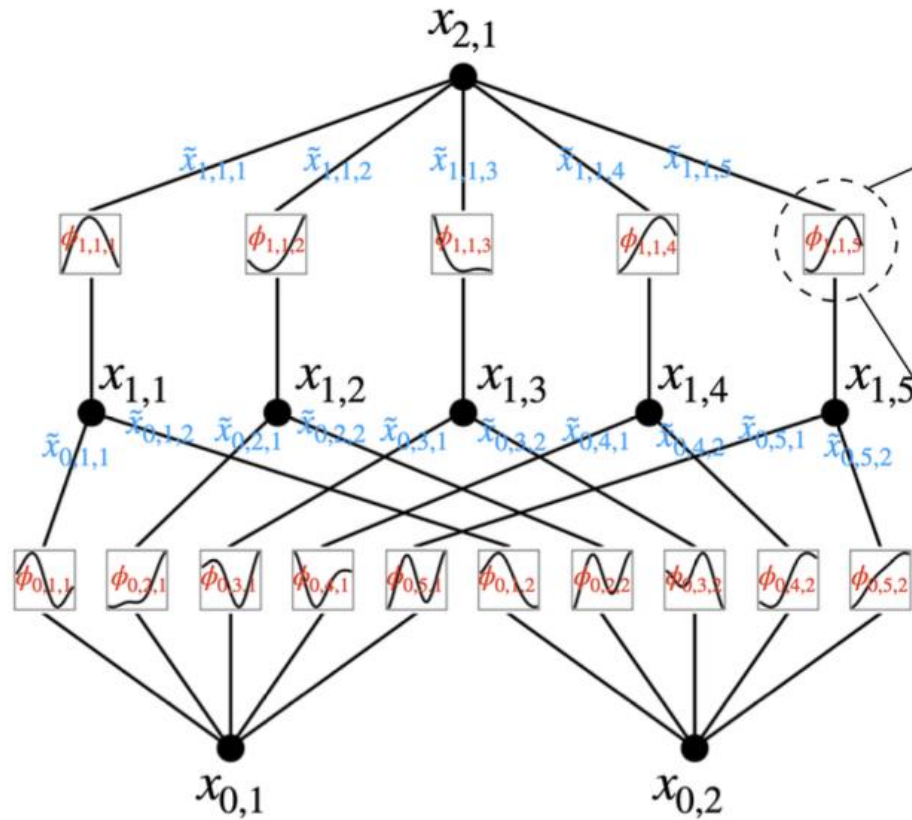












Orden  
de los  
splines

Tamaño  
de la  
grilla

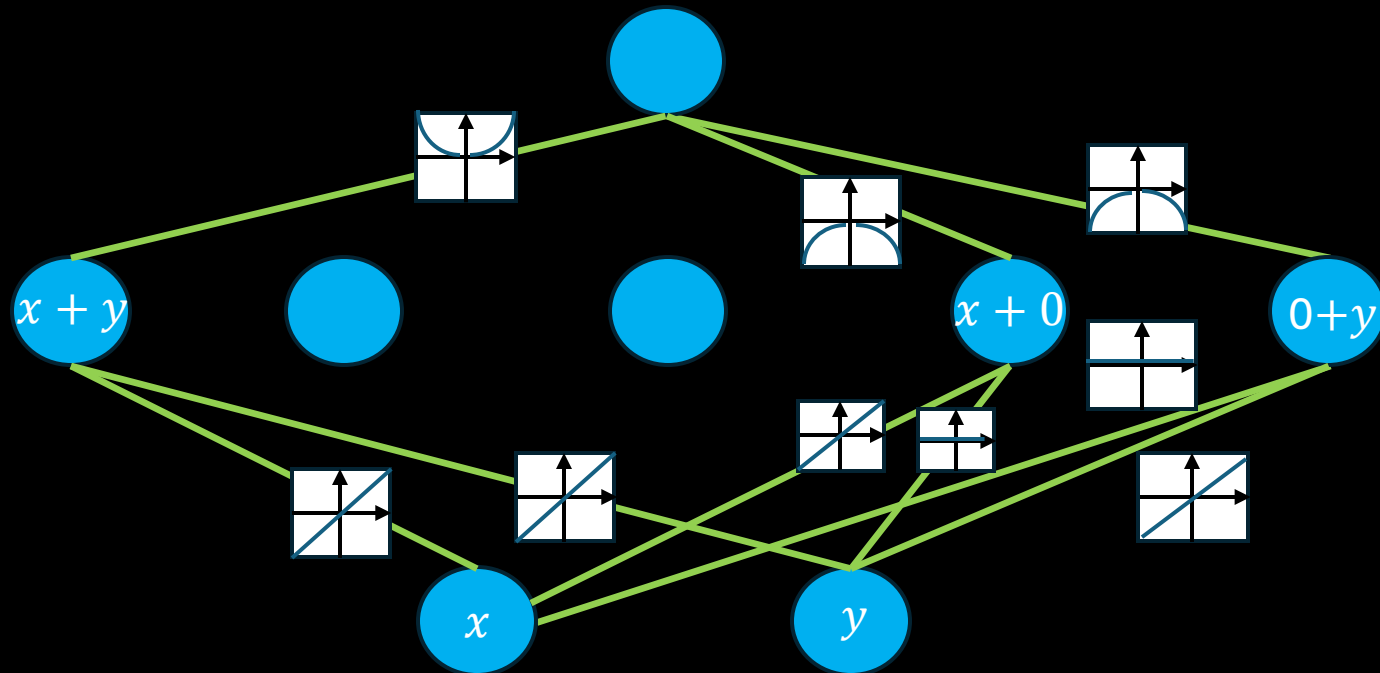
## La base teorica: el teorema de Kolmogorov Arnold

$$f(x_1, x_2, \dots, x_n) = \sum_{q=1}^{2n+1} \Phi_q \left( \sum_{p=1}^n \phi_{q,p}(x_p) \right)$$

Toda funcion continua multivaluada puede, en un intervalo acotado, escribirse como una composicion finita de suma de funciones de una variable.

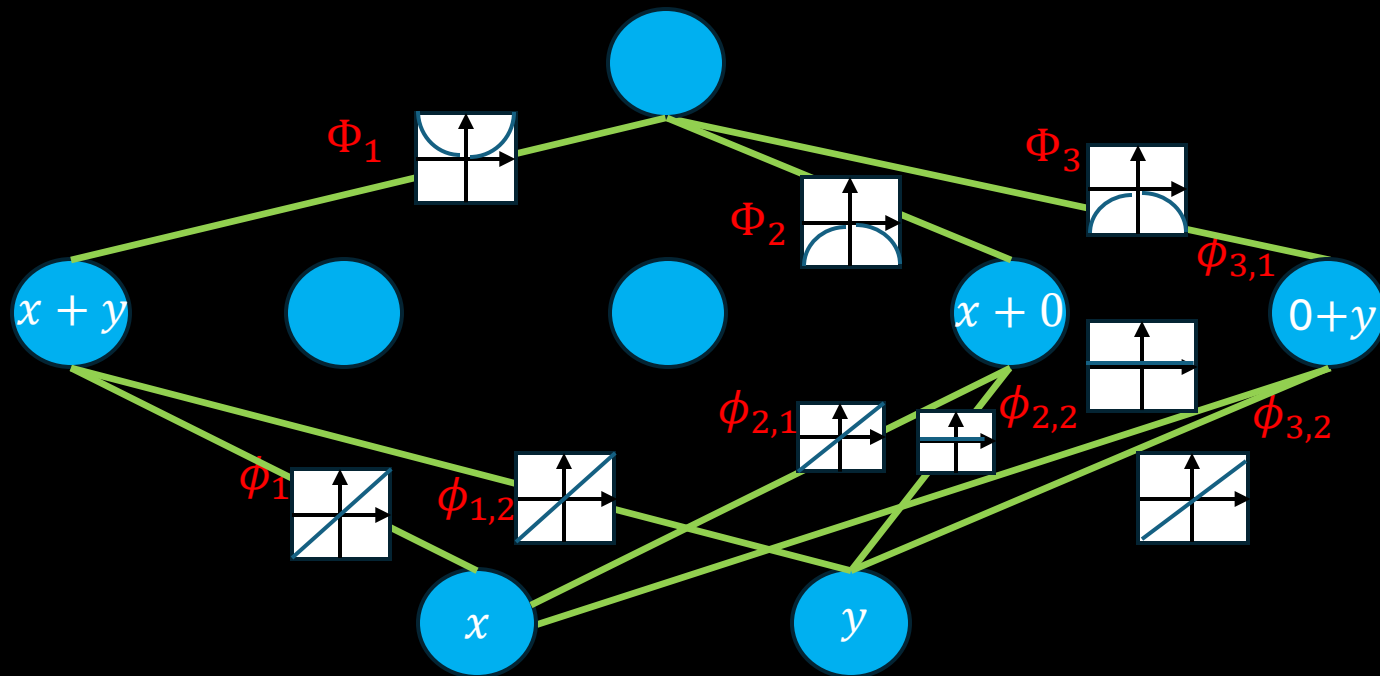
## Ejemplo

$$\frac{1}{2}(x+y)^2 - \frac{1}{2}x^2 - \frac{1}{2}y^2 = xy$$

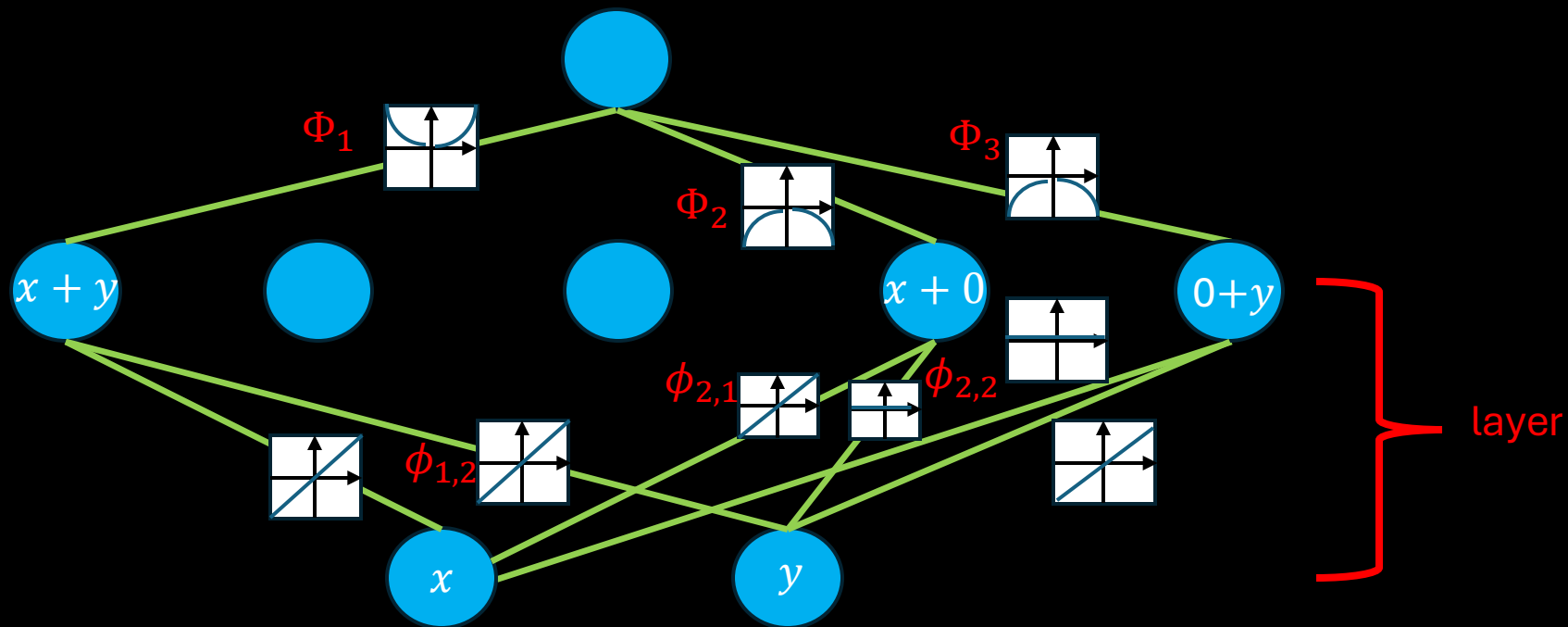


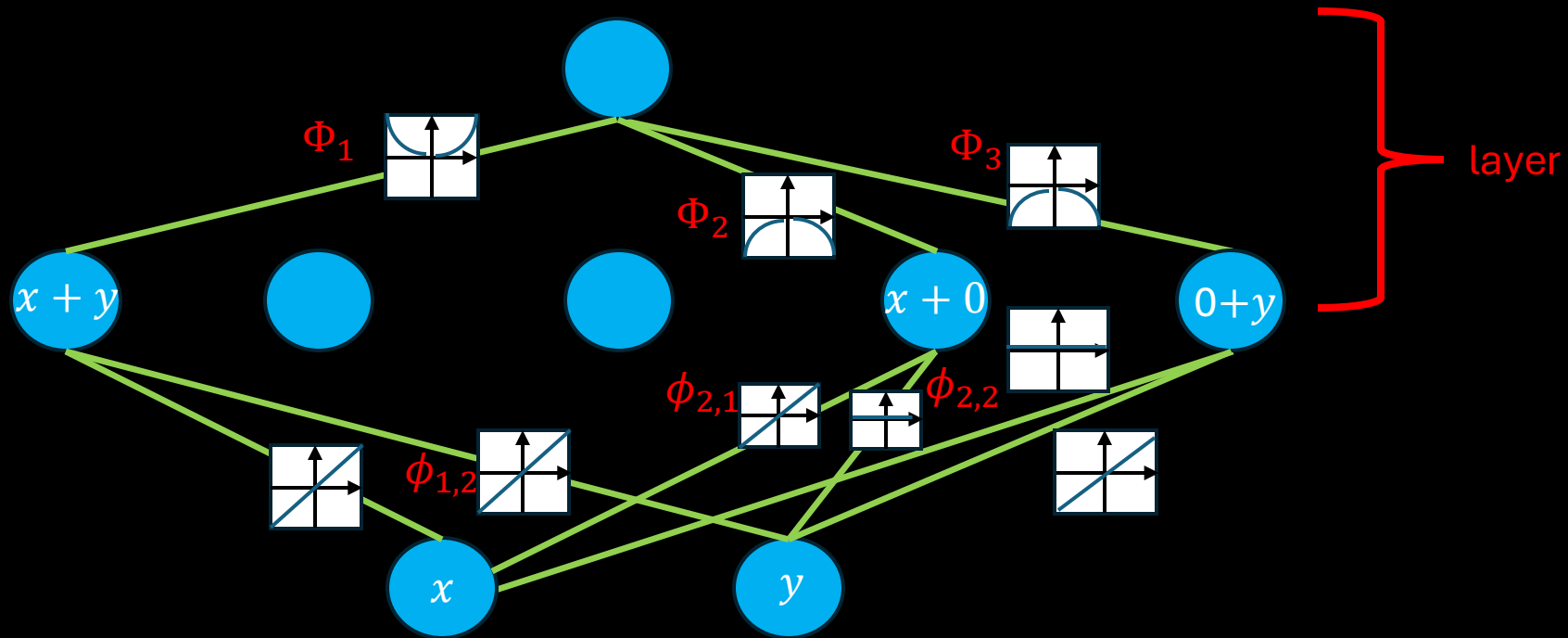
$$\sum_{q=1}^{2n+1} \Phi_q \left( \sum_{p=1}^n \phi_{q,p}(x_p) \right) = f(x_1, x_2, \dots, x_n)$$

$$\frac{1}{2}(x+y)^2 - \frac{1}{2}x^2 - \frac{1}{2}y^2 = xy$$



$$\left[ \begin{array}{l} \Phi_1(z) = \frac{1}{2}z^2 \\ \Phi_2(z) = -\frac{1}{2}z^2 \\ \Phi_3(z) = -\frac{1}{2}z^2 \end{array} \right.$$







Para MLP

## Teorema de la aproximacion universal

Una red neuronal feedforward, con una capa oculta constituida por un numero finito de neuronas, puede aproximar a cualquier funcion continua en un rango finito del input, con cualquier grado de precision, si la funcion de activacion es no lineal, acotada y continua

En principio, el numero de unidades en esa capa oculta puede ser enorme...

La solucion: poner muchas capas.

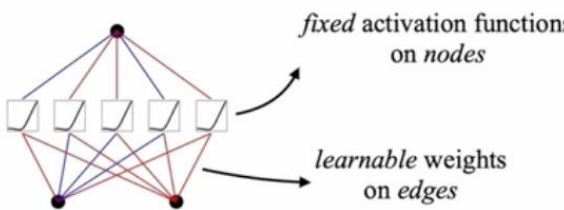
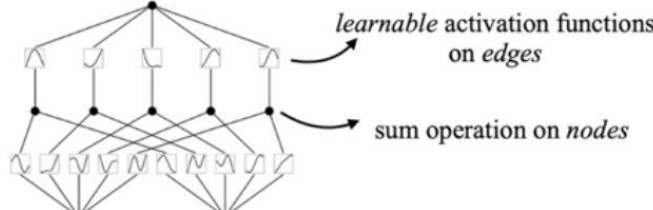
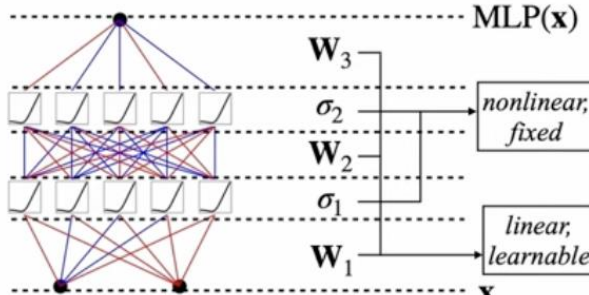
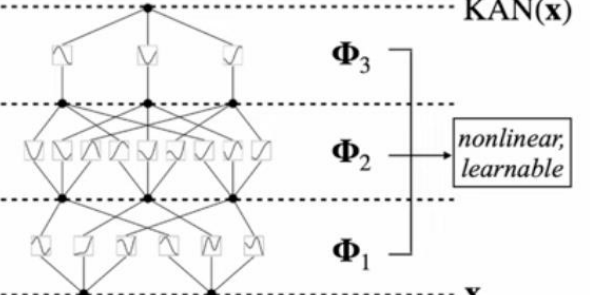
Para KAN

toda funcion continua multivaluada puede, en un intervalo acotado, escribirse como una composicion finita de suma de funciones de una variable.

$$f(x_1, x_2, \dots, x_n) = \sum_{q=1}^{2n+1} \Phi_q \left( \sum_{p=1}^n \phi_{q,p}(x_p) \right)$$

Un motivo por el que no se uso esto por mucho tiempo en redes...

La eventual complejidad de las funciones. Solucion... muchas capas

Model	Multi-Layer Perceptron (MLP)	Kolmogorov-Arnold Network (KAN)
Theorem	Universal Approximation Theorem	Kolmogorov-Arnold Representation Theorem
Formula (Shallow)	$f(\mathbf{x}) \approx \sum_{i=1}^{N(\epsilon)} a_i \sigma(\mathbf{w}_i \cdot \mathbf{x} + b_i)$	$f(\mathbf{x}) = \sum_{q=1}^{2n+1} \Phi_q \left( \sum_{p=1}^n \phi_{q,p}(x_p) \right)$
Model (Shallow)	<p>(a)</p> 	<p>(b)</p> 
Formula (Deep)	$\text{MLP}(\mathbf{x}) = (\mathbf{W}_3 \circ \sigma_2 \circ \mathbf{W}_2 \circ \sigma_1 \circ \mathbf{W}_1)(\mathbf{x})$	$\text{KAN}(\mathbf{x}) = (\Phi_3 \circ \Phi_2 \circ \Phi_1)(\mathbf{x})$
Model (Deep)	<p>(c)</p> 	<p>(d)</p> 

# Computational Complexity

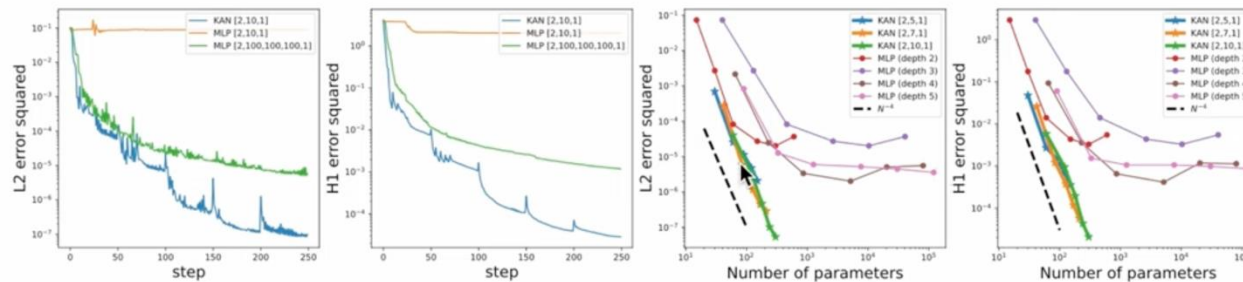


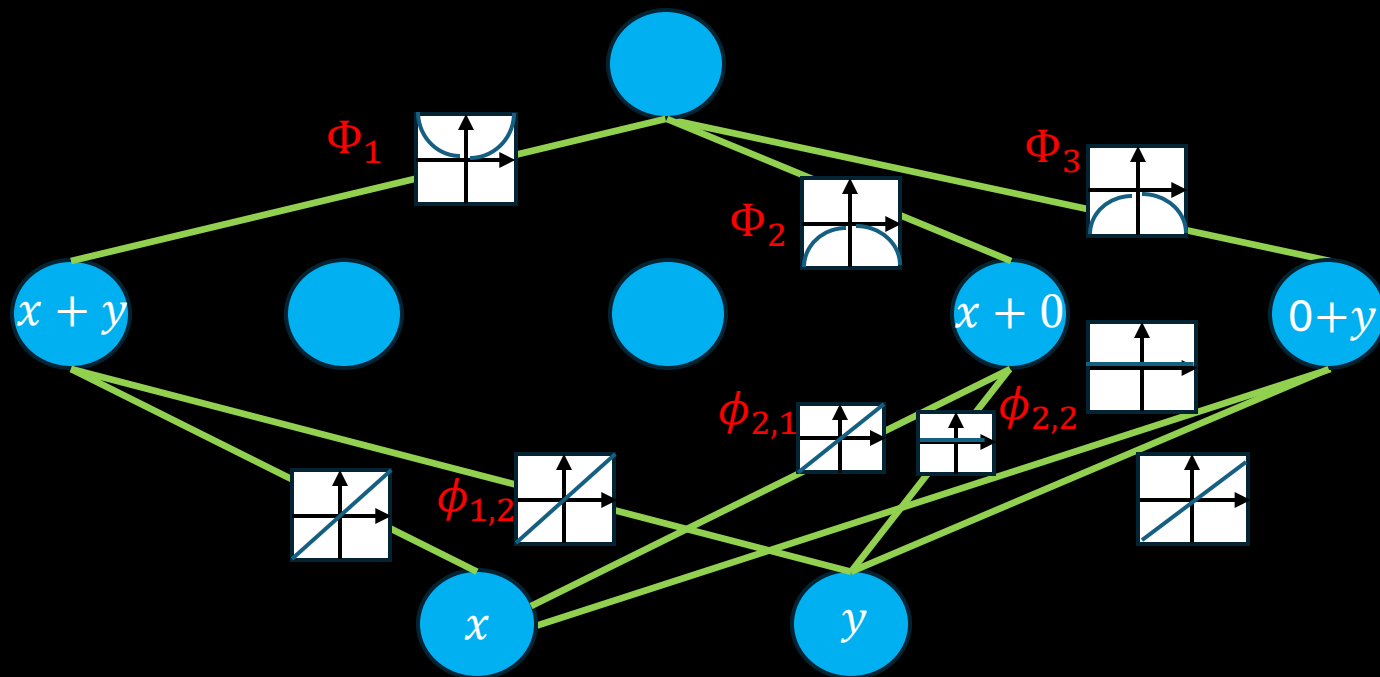
Figure 3.3: The PDE example. We plot L2 squared and H1 squared losses between the predicted solution and ground truth solution. First and second: training dynamics of losses. Third and fourth: scaling laws of losses against the number of parameters. KANs converge faster, achieve lower losses, and have steeper scaling laws than MLPs.

KAN, converge mas rapido que MLP

KAN, con menos parametros, igual loss a misma arquitectura que MLP

La gran cosa...  
interpretabilidad

$$\frac{1}{2}(x+y)^2 - \frac{1}{2}x^2 - \frac{1}{2}y^2 = xy$$



$$|\Phi|_1 \equiv \sum_{i=1}^{n_{\text{in}}} \sum_{j=1}^{n_{\text{out}}} |\phi_{i,j}|_1.$$

Norma L1

$$S(\Phi) \equiv - \sum_{i=1}^{n_{\text{in}}} \sum_{j=1}^{n_{\text{out}}} \frac{|\phi_{i,j}|_1}{|\Phi|_1} \log \left( \frac{|\phi_{i,j}|_1}{|\Phi|_1} \right).$$

“Entropia”

$$\ell_{\text{total}} = \ell_{\text{pred}} + \lambda \left( \mu_1 \sum_{l=0}^{L-1} |\Phi_l|_1 + \mu_2 \sum_{l=0}^{L-1} S(\Phi_l) \right),$$

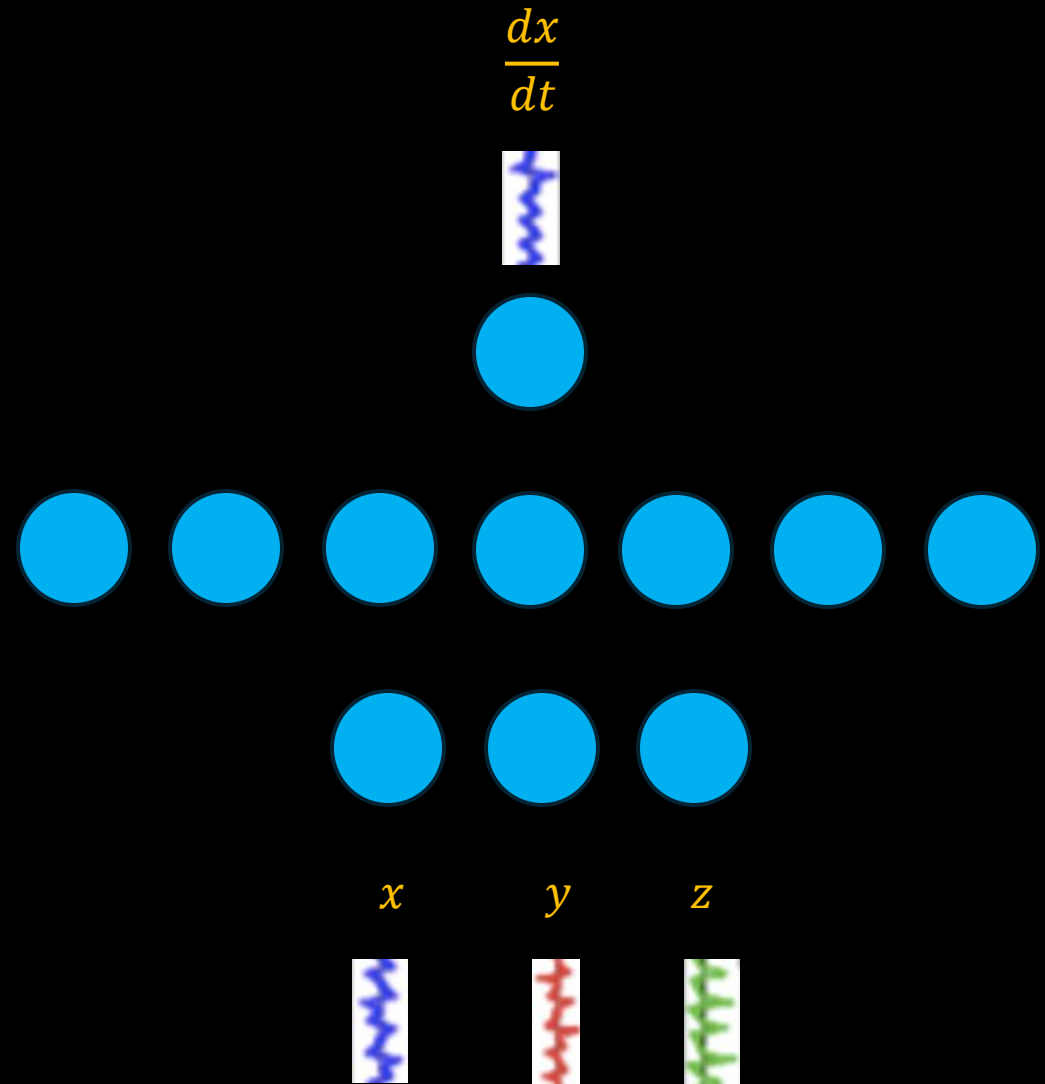
Penalizando el numero de neuronas activas, y la complejidad de las funciones, podemos simplificar

Una aplicacion de esta tecnica:  
reconstruir un campo vector a partir de un flujo

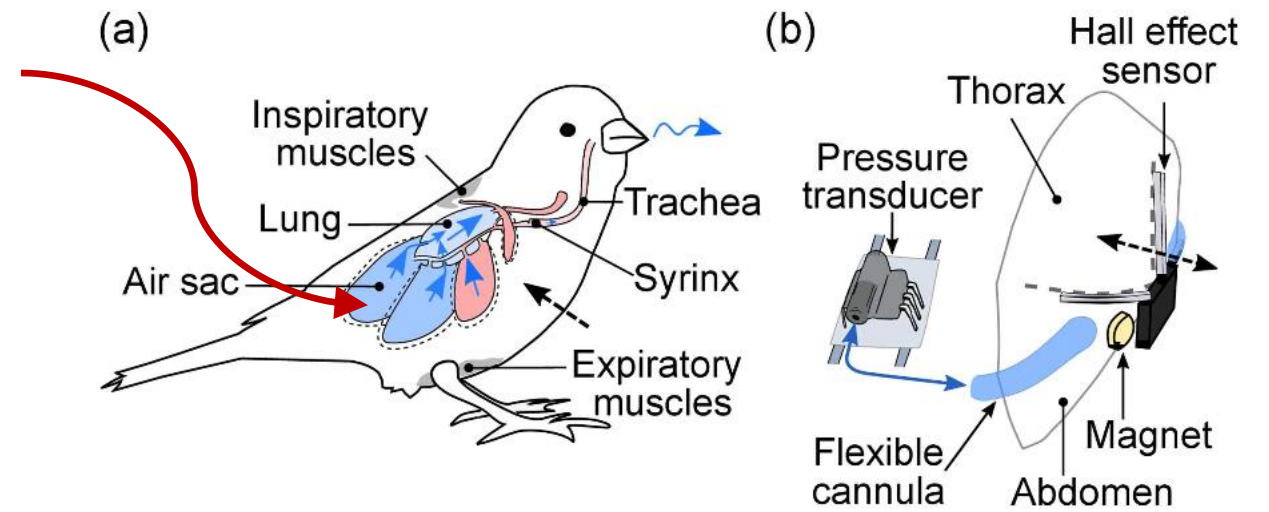
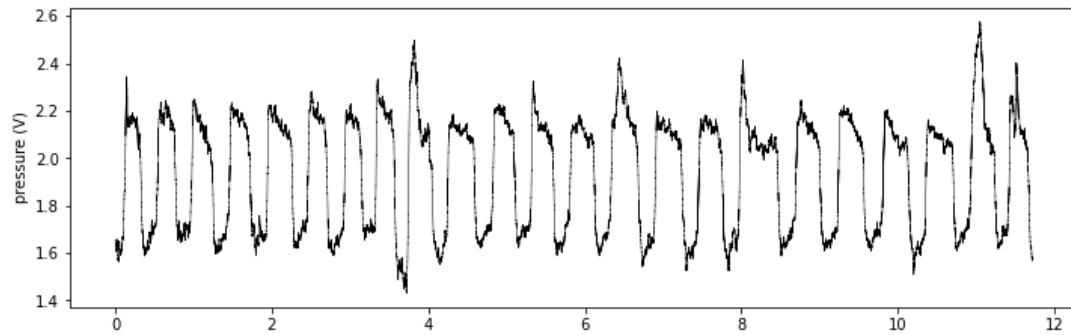
$$\frac{dx}{dt} = f_1(x, y, z)$$

$$\frac{dy}{dt} = f_2(x, y, z)$$

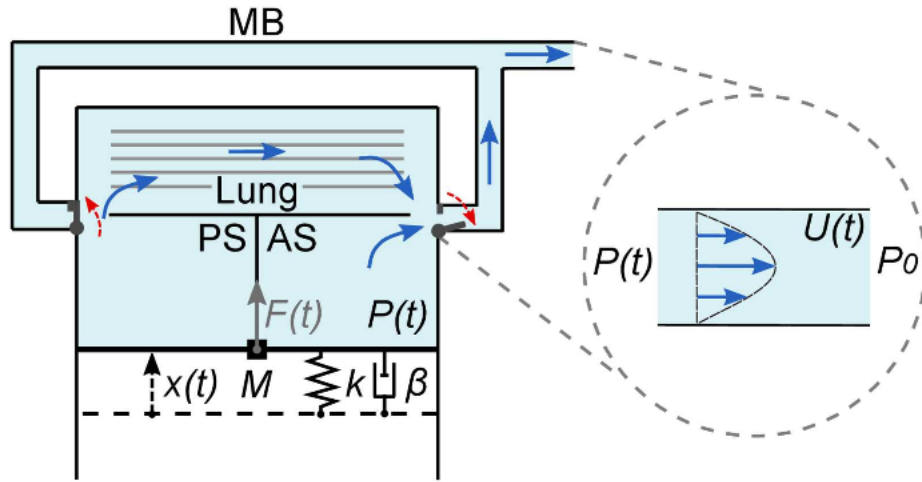
$$\frac{dz}{dt} = f_3(x, y, z)$$



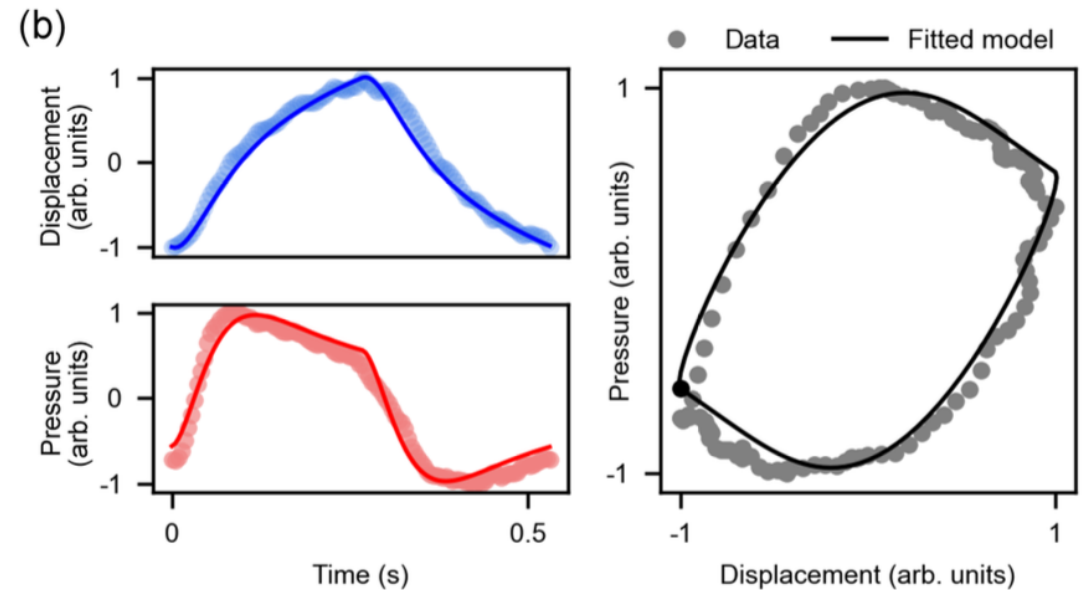
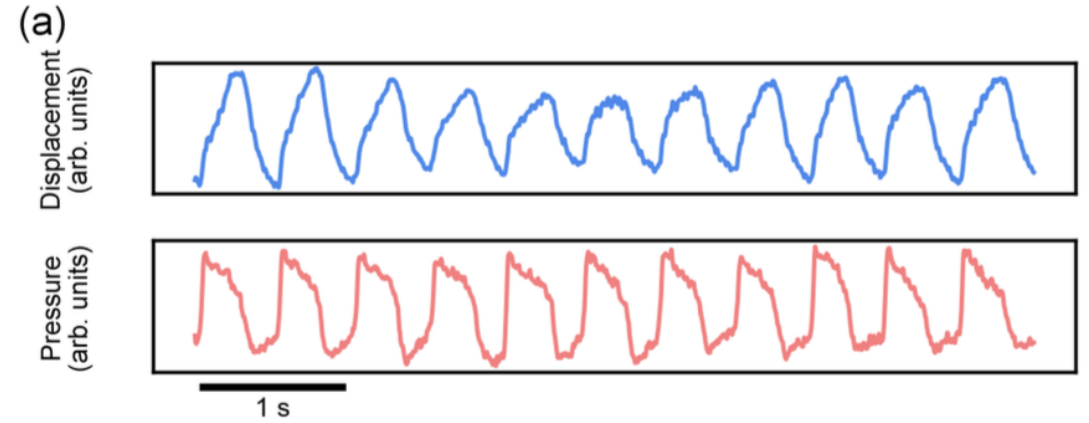
## Presion en los sacos aereos durante la respiracion normal



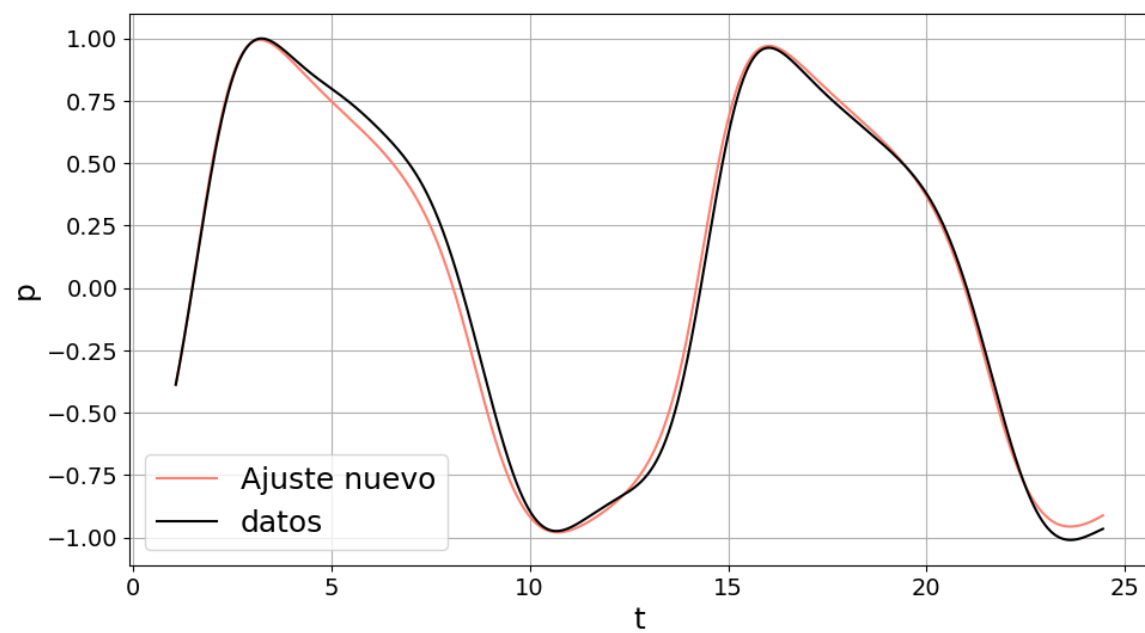
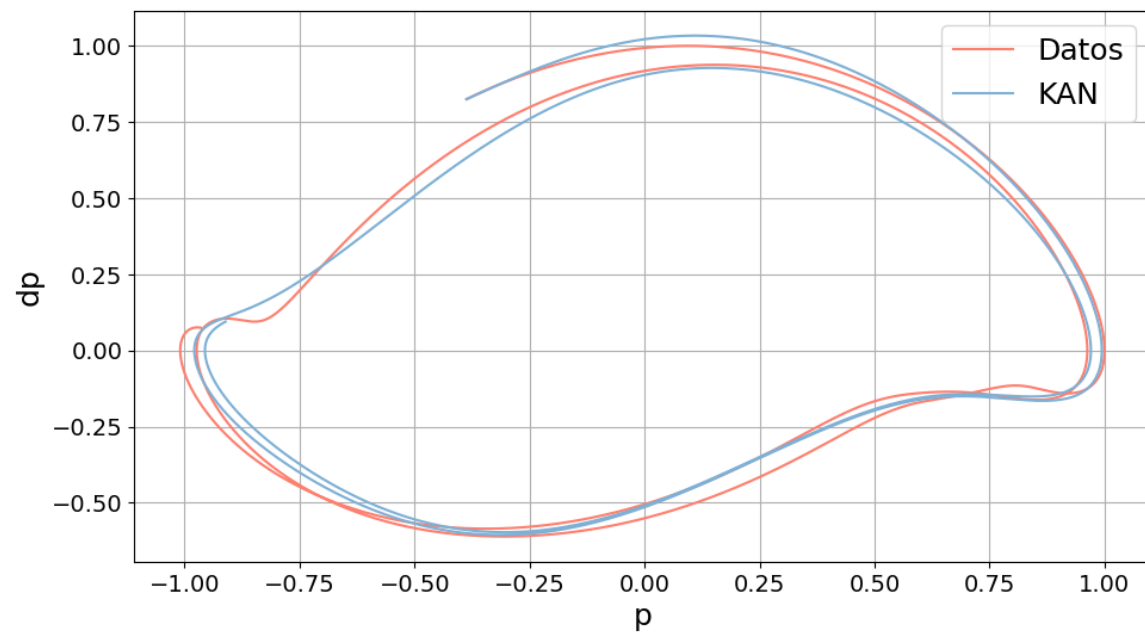




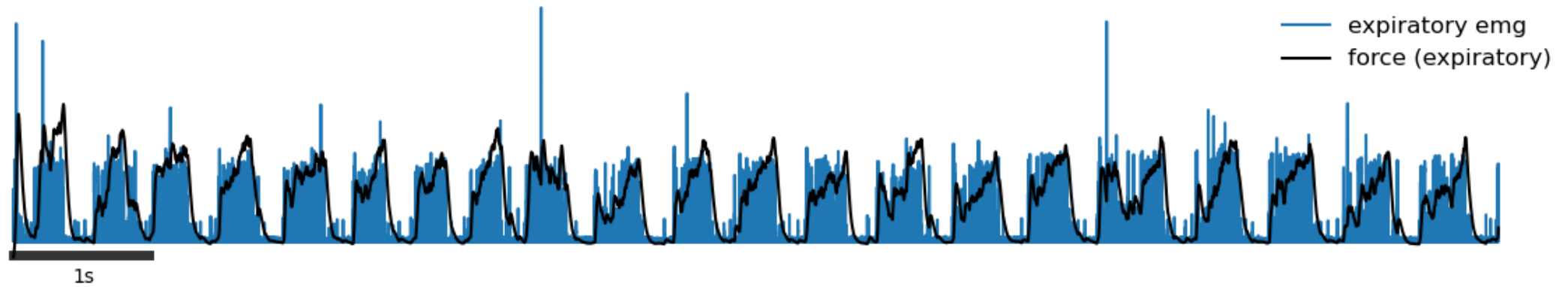
$$\ddot{p} = -\left(\frac{1}{\tau_x} + \frac{1+a}{\tau_p}\right)\dot{p} - \left(\frac{1}{\tau_x\tau_p}\right)p + \frac{F_0}{\tau_p}\frac{\partial F(t)}{\partial t}$$



Esta integracion asumio una fuerza “cuadrada”



$$\ddot{p} = \alpha \dot{p} + \beta (p + \gamma)^2 + \eta \left( \epsilon \dot{p} + \nu (p + \delta)^3 - 1 \right)^2 + h$$

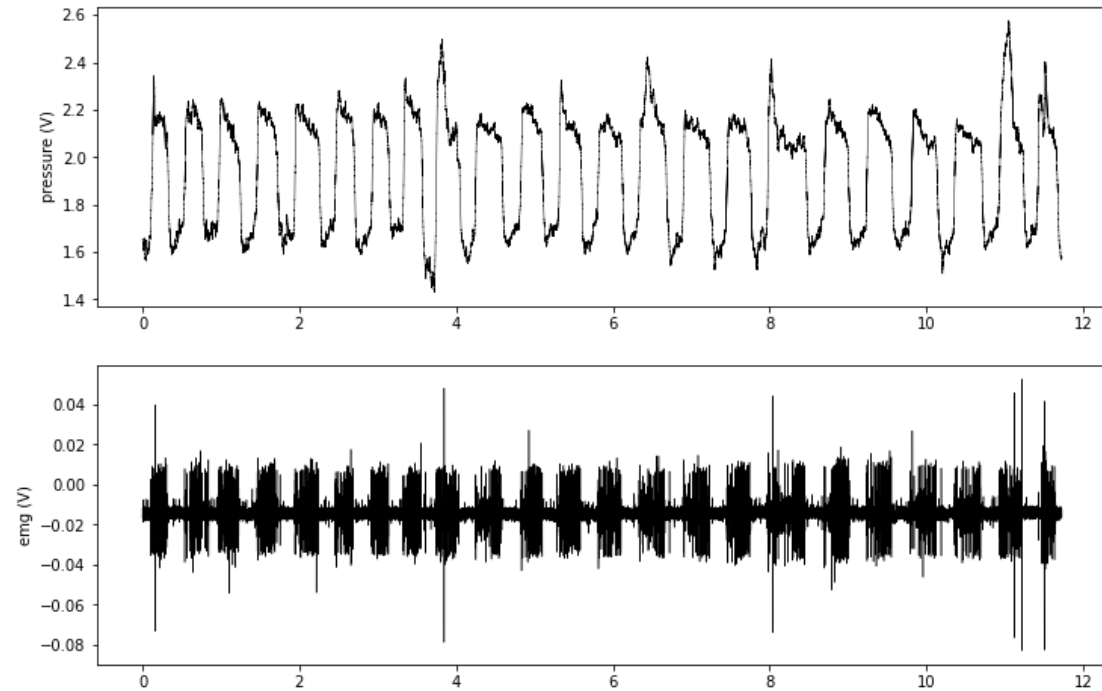


Que predice sobre la fuerza el modelo reconstruido con KAN?

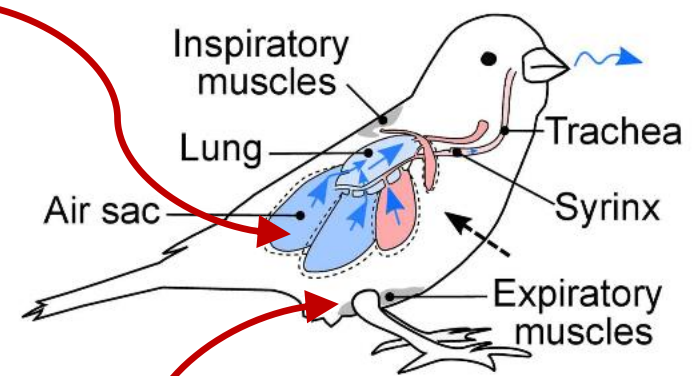
$$\ddot{p} = -\left(\frac{1}{\tau_x} + \frac{1+a}{\tau_p}\right)\dot{p} - \left(\frac{1}{\tau_x\tau_p}\right)p + \frac{F_0}{\tau_p}\frac{\partial f}{\partial t}$$

$$\ddot{p} = \alpha \dot{p} + \beta (p + \gamma)^2 + \eta \left( \epsilon \dot{p} + \nu (p + \delta)^3 - 1 \right)^2 + h$$

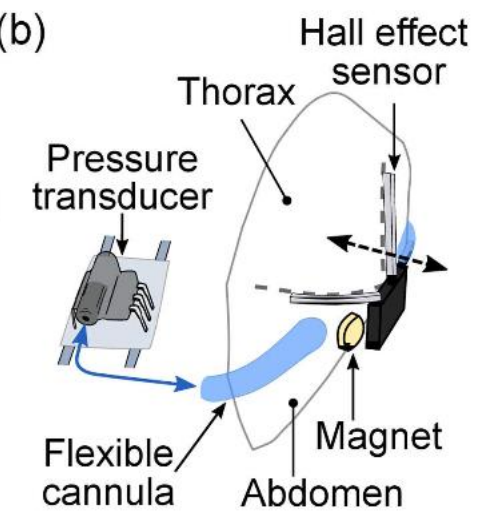
Esto permite recobrar  $f$

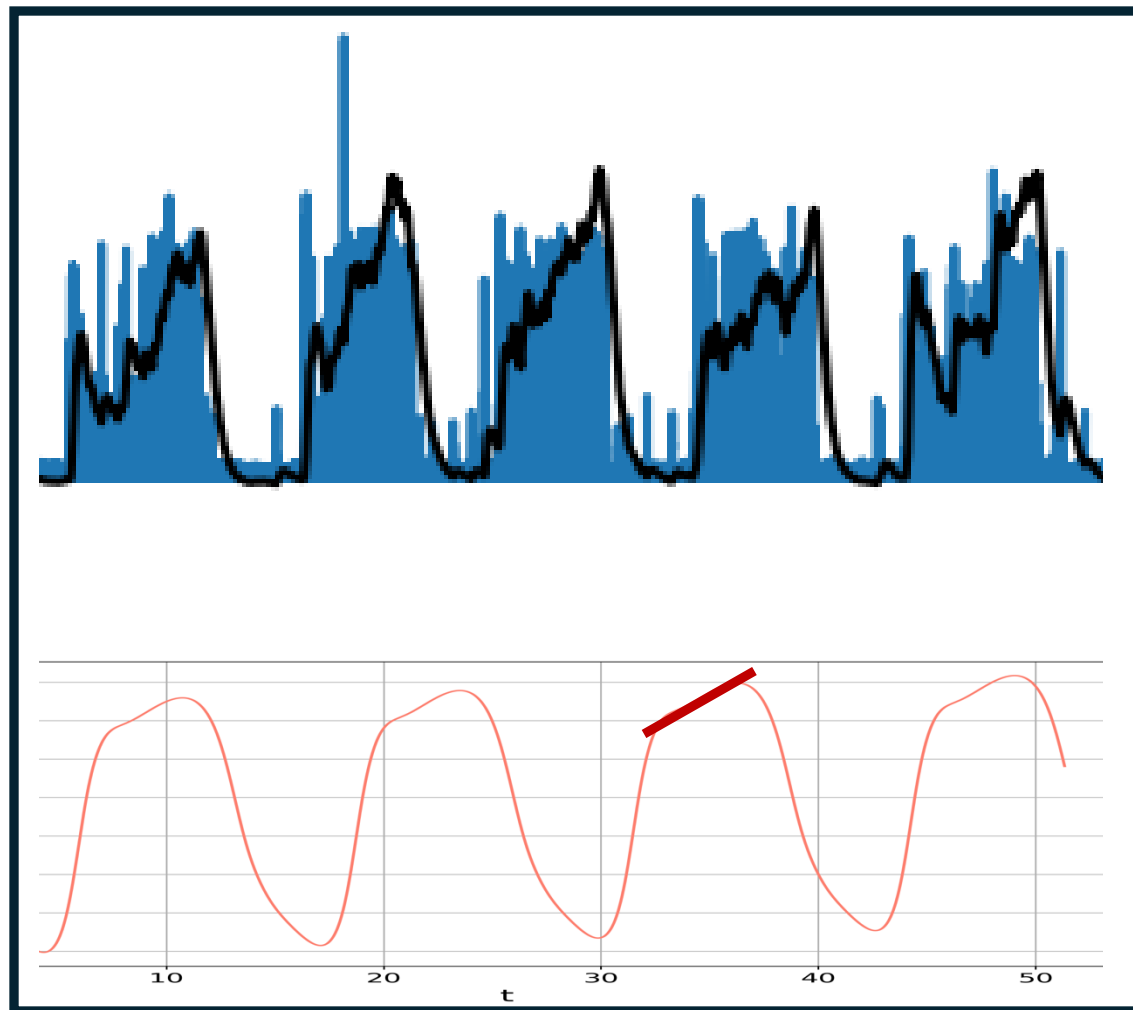


(a)



(b)





Prediccion del  
modelo KAN

# Sindy sparse identification of nonlinear dynamical systems

PNAS

ARTICLES ▾ FRONT MATTER AUTHORS ▾ TOPICS +

RESEARCH ARTICLE | APPLIED MATHEMATICS | 

## Discovering governing equations from data by sparse identification of nonlinear dynamical systems

Steven L. Brunton , Joshua L. Proctor, and J. Nathan Kutz [Authors Info & Affiliations](#)

Edited by William Bialek, Princeton University, Princeton, NJ, and approved March 1, 2016 (received for review August 31, 2015)

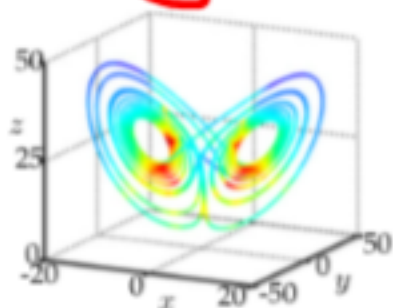
March 28, 2016 | 113 (15) 3932-3937 | <https://doi.org/10.1073/pnas.1517384113>

 267,667 | 2,931

   [PDF/EPUB](#)

### I. True Lorenz System

$$\begin{aligned}\dot{x} &= \sigma(y - x) \\ \dot{y} &= x(\rho - z) - y \\ \dot{z} &= xy - \beta z.\end{aligned}$$



Data In

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix} = \begin{bmatrix} 1 & x & y & z & x^2 & xy & xz & y^2 & z^5 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{bmatrix} \begin{bmatrix} \xi_1 \\ \xi_2 \\ \xi_3 \end{bmatrix}$$

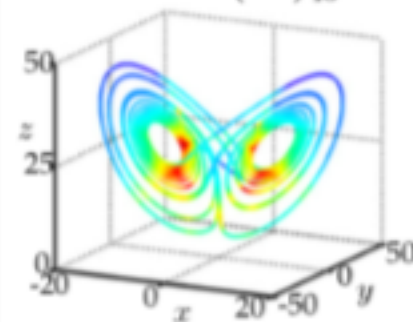
	'xi_1'	'xi_2'	'xi_3'
'1'	[ 0]	[ 0]	[ 0]
'x'	[-9.9996]	[27.9980]	[ 0]
'y'	[ 9.9998]	[-0.9997]	[ 0]
'z'	[ 0]	[ 0]	[-2.6665]
'xx'	[ 0]	[ 0]	[ 0]
'xy'	[ 0]	[ 0]	[ 1.0000]
'xz'	[ 0]	[-0.9999]	[ 0]
'yy'	[ 0]	[ 0]	[ 0]
'yz'	[ 0]	[ 0]	[ 0]
...	...	...	...
'yzzzz'	[ 0]	[ 0]	[ 0]
'zzzzz'	[ 0]	[ 0]	[ 0]

Sparse Coefficients of Dynamics

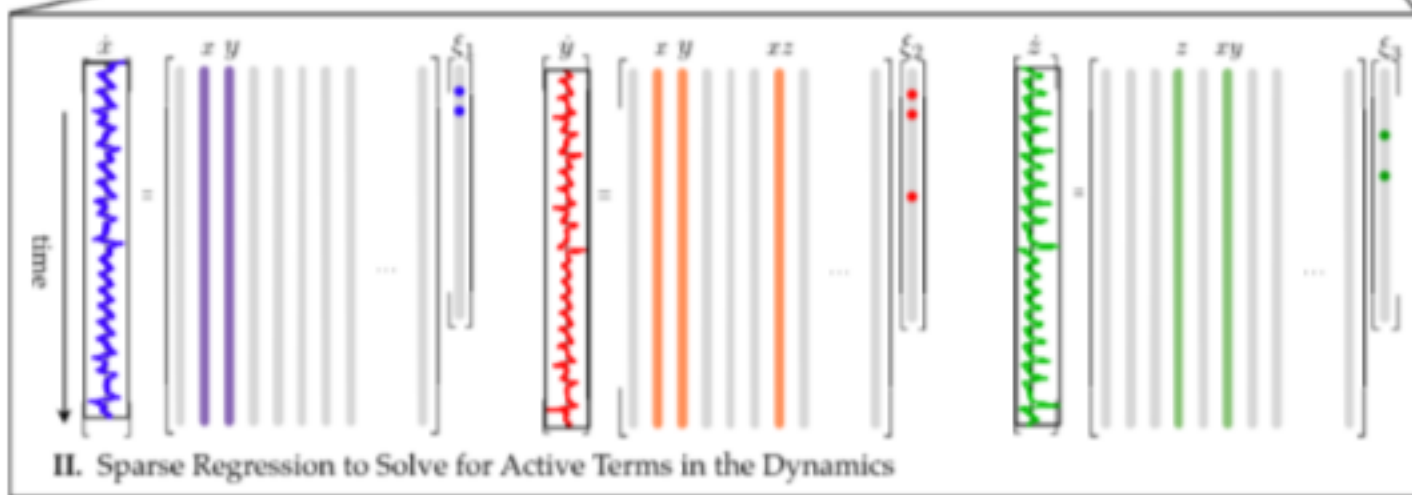
Model Out

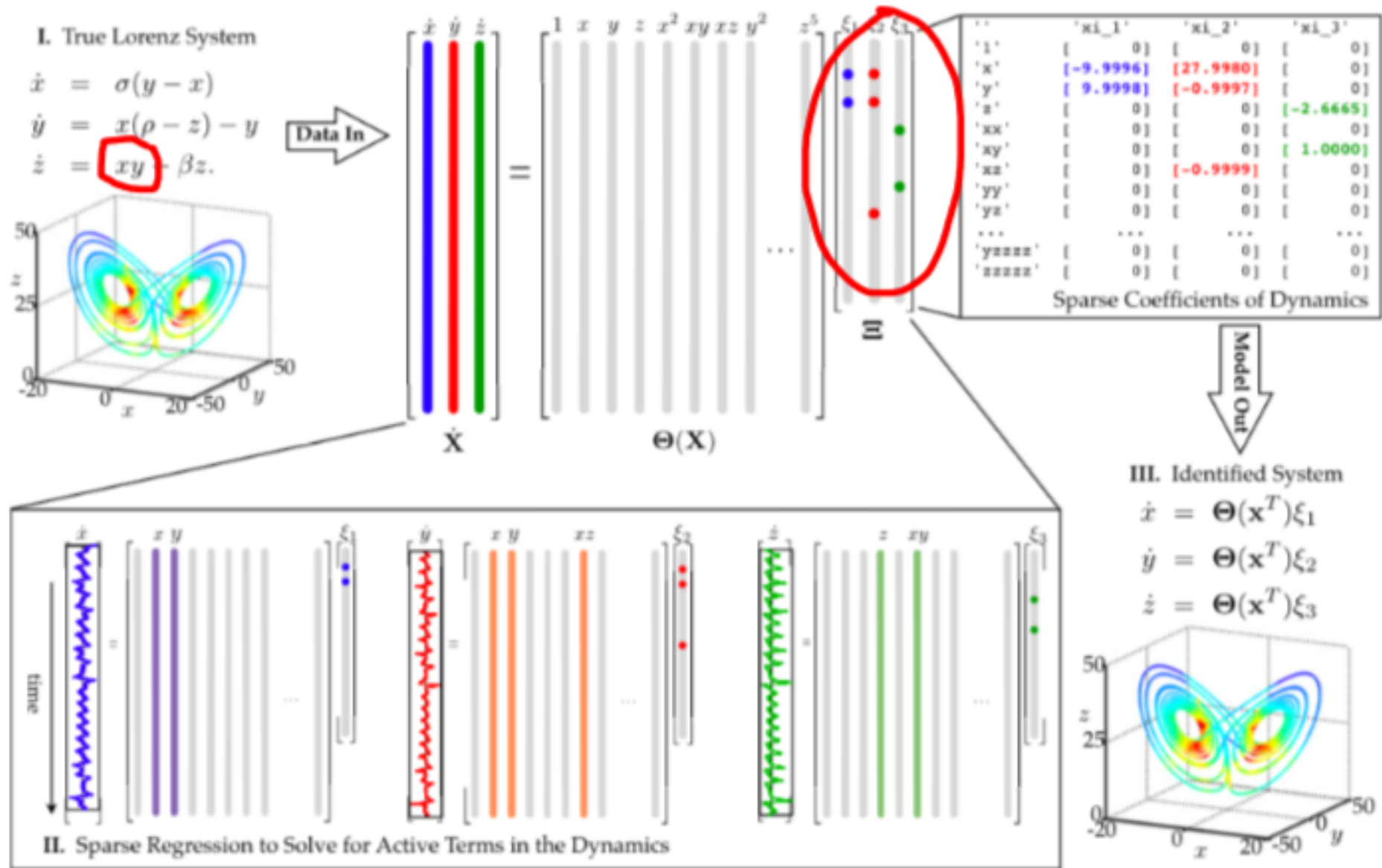
### III. Identified System

$$\begin{aligned}\dot{x} &= \Theta(\mathbf{x}^T)\xi_1 \\ \dot{y} &= \Theta(\mathbf{x}^T)\xi_2 \\ \dot{z} &= \Theta(\mathbf{x}^T)\xi_3\end{aligned}$$



### II. Sparse Regression to Solve for Active Terms in the Dynamics





A sparse fitting system, like LASSO, minimizes the difference penalizing the inclusion of terms.



Así, para recuperar un campo vectorial a partir de datos, uno comienza tomando un conjunto de series temporales  $\mathbf{x}(t) = (x_1(t), x_2(t), \dots, x_n(t))^T$ , y calculamos o medimos las derivadas de las variables también:

$$\begin{bmatrix} \mathbf{x}^T(t_1) \\ \mathbf{x}^T(t_2) \\ \vdots \\ \mathbf{x}^T(t_m) \end{bmatrix} = \begin{bmatrix} x_1(t_1) & x_2(t_1) & \dots & x_n(t_1) \\ x_1(t_2) & x_2(t_2) & \dots & x_n(t_2) \\ \vdots & \vdots & \ddots & \vdots \\ x_1(t_m) & x_2(t_m) & \dots & x_n(t_m) \end{bmatrix}$$

$$\begin{bmatrix} \dot{\mathbf{x}}^T(t_1) \\ \dot{\mathbf{x}}^T(t_2) \\ \vdots \\ \dot{\mathbf{x}}^T(t_m) \end{bmatrix} = \begin{bmatrix} \dot{x}_1(t_1) & \dot{x}_2(t_1) & \dots & \dot{x}_n(t_1) \\ \dot{x}_1(t_2) & \dot{x}_2(t_2) & \dots & \dot{x}_n(t_2) \\ \vdots & \vdots & \ddots & \vdots \\ \dot{x}_1(t_m) & \dot{x}_2(t_m) & \dots & \dot{x}_n(t_m) \end{bmatrix}$$

Luego, construimos una “librería” con todos los términos de funciones no lineales que se nos ocurra pertinente incluir en el modelo (por ejemplo, todas las funciones polinomiales hasta cierto grado, funciones periódicas...):

$$\Theta(\mathbf{X}) = [\mathbf{I} \quad \mathbf{X} \quad \mathbf{X}^{p_2} \quad \mathbf{X}^3 \dots \cos(\mathbf{X}) \dots],$$

donde

$$\mathbf{X} = [\mathbf{x}(t_1), \mathbf{x}(t_2), \dots, \mathbf{x}(t_m)]^T$$

y planteamos un algoritmo de regresión “rala”, tipo LASSO, para calcular los vectores de coeficientes del ajuste:

$$\Xi = [\xi_1 \quad \xi_2 \dots \xi_n],$$

Donde cada coeficiente dará cómo entran los términos no lineales, en cada una de las componentes del campo vector. De este modo, cada vector  $\xi_1$  tiene tantas componentes como columnas tenga la librería  $\Theta(\mathbf{X})$ .

Así, el sistema dinámico original, evaluado en los tiempos  $\{t_1, t_2, \dots, t_m\}$  da lugar al siguiente sistema (sobre determinado!) de ecuaciones algebraicas:

$$\dot{\mathbf{X}} = \mathbf{\Theta}(\mathbf{X}) \mathbf{\Xi},$$

donde cada columna  $\xi_k$  en  $\mathbf{\Xi}$  es un vector cuyos coeficientes indican cuales son los términos presentes en el campo vector en la k-ésima fila del campo vector original:

$$\frac{dx}{dt} = f(x), \quad \text{con } x \in R^n,$$

o sea, la de la componente:

$$\frac{dx_k}{dt} = f_k(x_1, x_2, \dots, x_n).$$

Con esta notación, entonces, un modelo para ajustar estos coeficientes (que según nuestra hipótesis de modelos “ralos” deberían ser pocos por componente) se logra estimando esos coeficientes mediante un algoritmo de regresión rala (l1-regularizada):

$$\xi_k = \underset{\xi_{k'}}{\operatorname{argmin}} (\|\dot{\mathbf{X}}_k - \mathbf{\Theta}(\mathbf{X})\xi_{k'}\|) + \lambda \|\xi_{k'}\|,$$

donde  $\dot{\mathbf{X}}_k$  es la k-ésima columna de

$$\dot{\mathbf{X}} = \begin{bmatrix} \dot{x}_1(t_1) & \dot{x}_2(t_1) & \dots & \dot{x}_n(t_1) \\ \dot{x}_1(t_2) & \dot{x}_2(t_2) & \dots & \dot{x}_n(t_2) \\ \vdots & \vdots & \ddots & \vdots \\ \dot{x}_1(t_m) & \dot{x}_2(t_m) & \dots & \dot{x}_n(t_m) \end{bmatrix}$$

El parámetro  $\lambda$  es el responsable de penalizar la presencia de términos en el ajuste. Es decir, uno busca el conjunto de coeficientes que constituyen  $\xi_k$  que minimizan la diferencia entre las derivadas y el campo vector, pero poniéndole un precio a cada término que agrego. El precio es precisamente  $\lambda$ .

Veamos un ejemplo. Supongamos tenemos un sistema regido por las ecuaciones de Lorenz, y supongamos, además que las tres variables del sistema dinámico pueden ser medidas. Así, el problema está regido por estas ecuaciones:

$$\begin{aligned}\frac{dx}{dt} &= \sigma(y - x) \\ \frac{dy}{dt} &= x(\rho - z) \\ \frac{dz}{dt} &= xy - \beta z.\end{aligned}$$

Medimos las tres variables, calculamos las derivadas, y tenemos:

$$\dot{\mathbf{X}} = \begin{bmatrix} \dot{x}(t_1) & \dot{y}(t_1) & \dot{z}(t_1) \\ \dot{x}(t_2) & \dot{y}(t_2) & \dot{z}(t_2) \\ \vdots & \vdots & \vdots \\ \dot{x}(t_m) & \dot{y}(t_m) & \dot{z}(t_m) \end{bmatrix}.$$

Luego, calculamos la librería sobre las mediciones, esto es:

$$\Theta(\mathbf{X}) = \begin{bmatrix} 1 & x(t_1) & y(t_1) & z(t_1) & x^2(t_1) & xy(t_1) & xz(t_1) & yz(t_1) & y^2(t_1) & z^2(t_1) \\ 1 & x(t_2) & y(t_2) & z(t_2) & x^2(t_2) & xy(t_2) & xz(t_2) & yz(t_2) & y^2(t_2) & z^2(t_2) \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x(t_m) & y(t_m) & z(t_m) & x^2(t_m) & xy(t_m) & xz(t_m) & yz(t_m) & y^2(t_m) & z^2(t_m) \end{bmatrix}$$

y lo que necesitamos calcular son los coeficientes de los vectores  $(\xi_1, \xi_2, \xi_3)$ , cada uno de los cuales tiene 10 componentes. Así, la primera ecuación del sistema que debemos resolver lucirá:

$$\begin{bmatrix} \dot{x}(t_1) \\ \dot{x}(t_2) \\ \vdots \\ \dot{x}(t_m) \end{bmatrix} = \begin{bmatrix} 1 & x(t_1) & y(t_1) & z(t_1) & xy(t_1) & \dots \\ 1 & x(t_2) & y(t_2) & z(t_2) & xy(t_2) & \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x(t_m) & y(t_m) & z(t_m) & xy(t_m) & \dots \end{bmatrix} \begin{bmatrix} \xi_1^1 \\ \xi_1^2 \\ \vdots \\ \xi_1^{10} \end{bmatrix}$$

la segunda ecuación a resolver es:

$$\begin{bmatrix} \dot{y}(t_1) \\ \dot{y}(t_2) \\ \vdots \\ \dot{y}(t_m) \end{bmatrix} = \begin{bmatrix} 1 & x(t_1) & y(t_1) & z(t_1) & xy(t_1) & \dots \\ 1 & x(t_2) & y(t_2) & z(t_2) & xy(t_2) & \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x(t_m) & y(t_m) & z(t_m) & xy(t_m) & \dots \end{bmatrix} \begin{bmatrix} \xi_2^1 \\ \xi_2^2 \\ \vdots \\ \xi_2^{10} \end{bmatrix}$$

y la tercera es:

$$\begin{bmatrix} \dot{z}(t_1) \\ \dot{z}(t_2) \\ \vdots \\ \dot{z}(t_m) \end{bmatrix} = \begin{bmatrix} 1 & x(t_1) & y(t_1) & z(t_1) & xy(t_1) & \dots \\ 1 & x(t_2) & y(t_2) & z(t_2) & xy(t_2) & \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x(t_m) & y(t_m) & z(t_m) & xy(t_m) & \dots \end{bmatrix} \begin{bmatrix} \xi_3^1 \\ \xi_3^2 \\ \vdots \\ \xi_3^{10} \end{bmatrix}$$

El resultado del algoritmo de ajuste por regresión lineal, raro, debería dar:

$$\begin{bmatrix} \xi_1^1 \\ \xi_1^2 \\ \xi_1^3 \\ \xi_1^4 \\ \xi_1^5 \\ \xi_1^6 \\ \xi_1^7 \\ \xi_1^8 \\ \xi_1^9 \\ \xi_1^{10} \end{bmatrix} = \begin{bmatrix} 0 \\ -\sigma \\ \sigma \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} \xi_2^1 \\ \xi_2^2 \\ \xi_2^3 \\ \xi_2^4 \\ \xi_2^5 \\ \xi_2^6 \\ \xi_2^7 \\ \xi_2^8 \\ \xi_2^9 \\ \xi_2^{10} \end{bmatrix} = \begin{bmatrix} 0 \\ \rho \\ 0 \\ 0 \\ 0 \\ 0 \\ -1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} \xi_3^1 \\ \xi_3^2 \\ \xi_3^3 \\ \xi_3^4 \\ \xi_3^5 \\ \xi_3^6 \\ \xi_3^7 \\ \xi_3^8 \\ \xi_3^9 \\ \xi_3^{10} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ -\beta \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

En qué consiste un ajuste de regresión “raro”, tipo LASSO? Simplemente en buscar minimizar la función diferencia entre lo que esté del lado derecho e izquierdo de nuestra ecuación, pero sumándole un término por cada coeficiente distinto de cero que se introduzca en el ajuste. De ese modo, buscar la minimización metiendo tantos términos como sea posible queda excluido por la penalización.

Estos algoritmos están eficientemente implementados en las librerías estadísticas de cualquier paquete de programación de alto nivel. En Python, por ejemplo, el proceso es tan sencillo como armar con los datos la librería y aplicar una función de ajuste via Lasso. Supongamos nuestro problema viene descrito por tres variables  $(x, y, p)$ , a partir de cuya medición calculamos las derivadas temporales  $(\dot{x}, \dot{y}, \dot{p})$ . suponemos por simplicidad que consideraremos sólo términos lineales. Entonces las instrucciones:

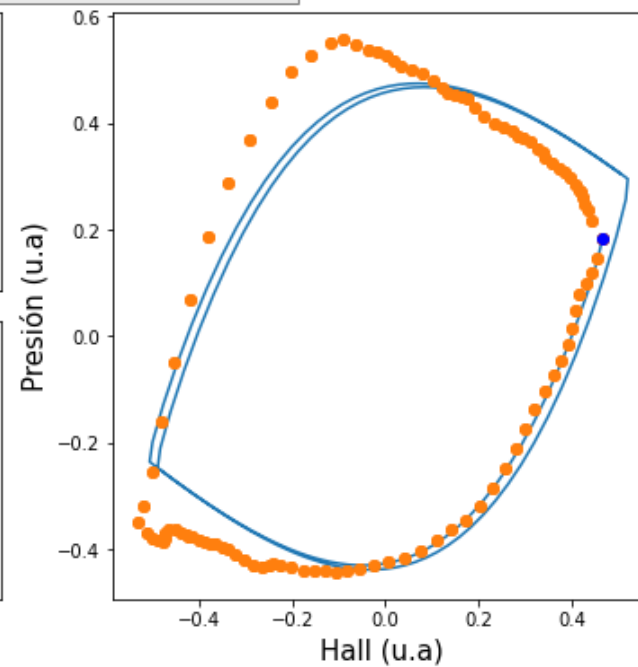
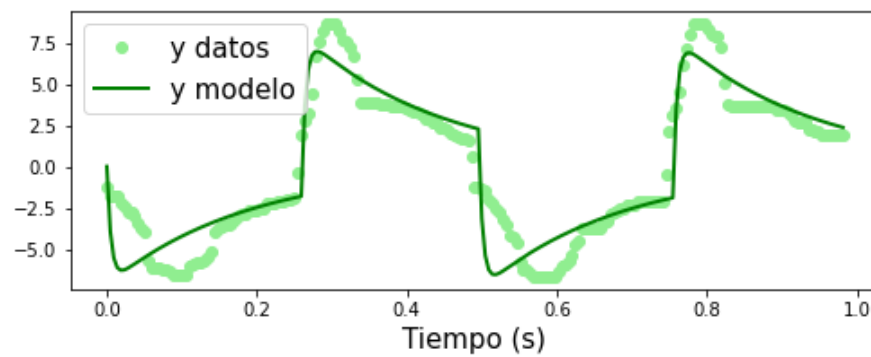
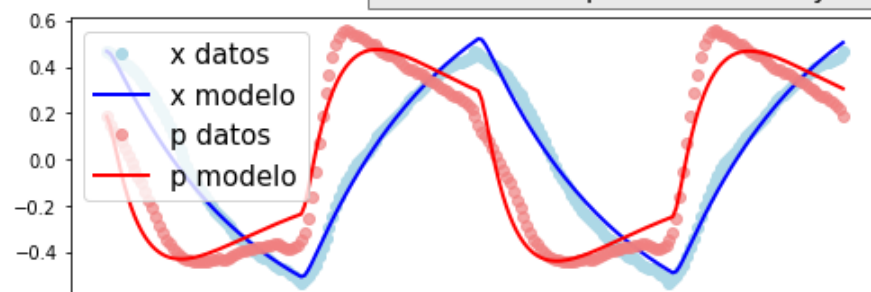
```
theta[:,0]=np.ones_like(x)
theta[:,1]=x
theta[:,2]=y
theta[:,3]=p
```

cargan la matriz  $\Theta(X)$ , y asignan un uno al primer término, que se encarga de los términos constantes en el campo vector. Ahora, el ajuste es tan sencillo como correr las dos instrucciones que escribimos a continuación,

```
clf=linear_model.Lasso(alpha=0.01,max_iter=10000000
0,fit_intercept=False,normalize=False)
clf.fit(-theta,dxdt)
```

$$dy/dt = -830. - 980. x - 190. y + 1800. f(t)$$

$$dp/dt = +2.3 y - 23. p$$



# Problemas

- No esta garantizada la estabilidad de las soluciones del modelo ajustado. Se ajustan soluciones, no necesariamente soluciones estables.
- La hipotesis de que los sistemas fisicos a modelar estan regidos por campos vectores sencillos, es poco realista para datos experimentales.
- Las variables medidas son un subconjunto de las que rigen la dinamica del problema