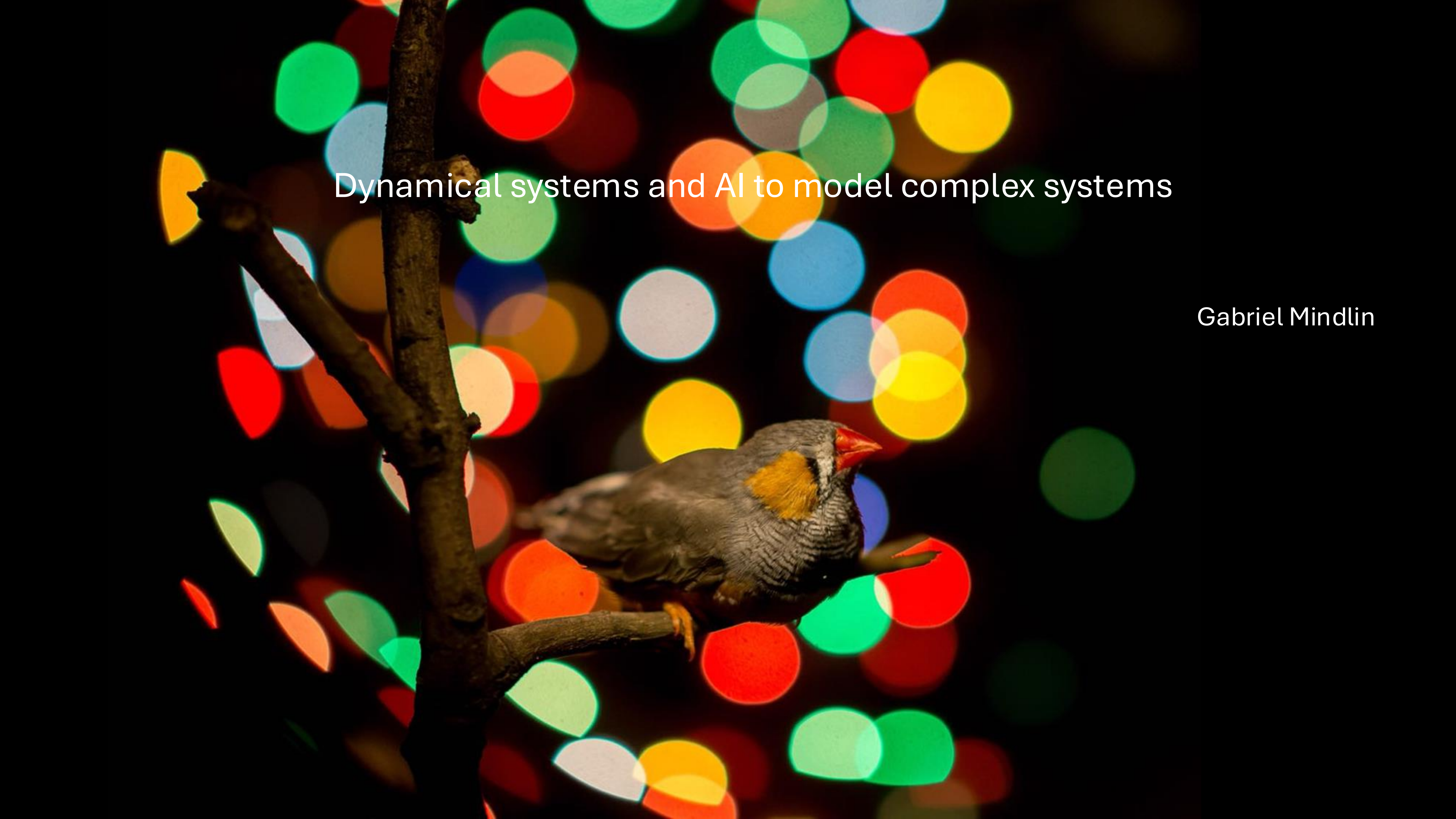
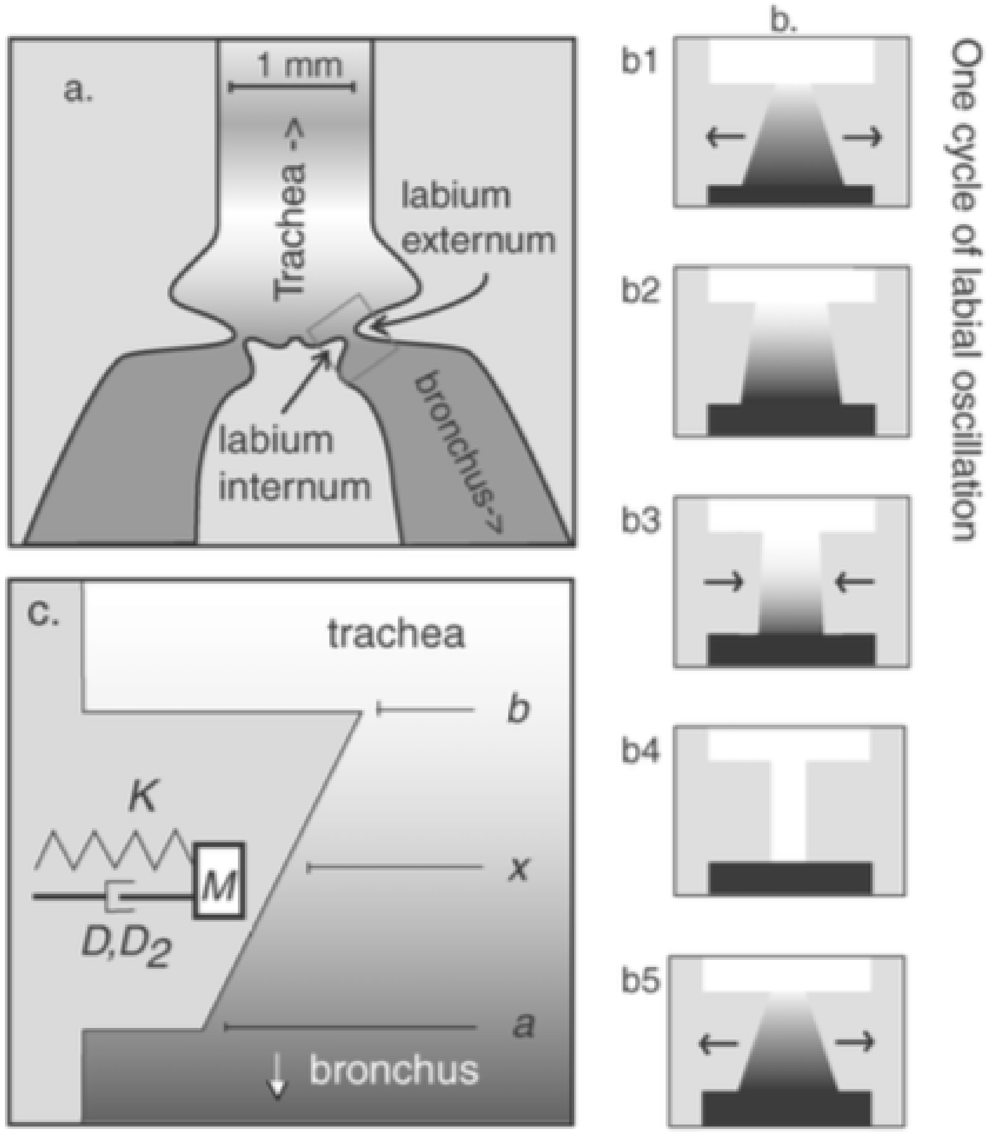
Dynamical systems and AI to model complex systems

Gabriel Mindlin



Example

¿How do birds generate their songs?

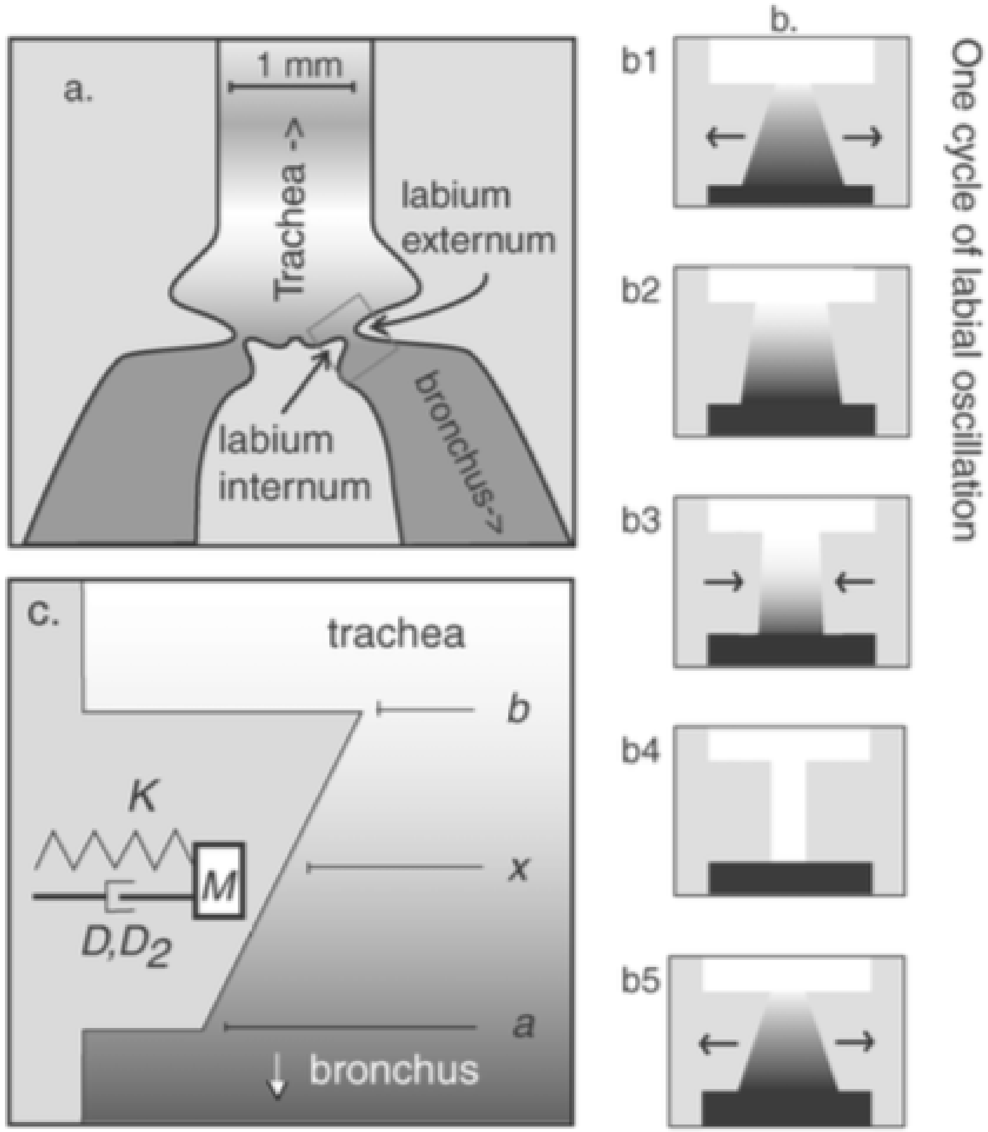


What did we conjecture in 2000?

$$\frac{dx}{dt} = y,$$

$$\frac{dy}{dt} = (1/m)\underbrace{\left[-k(x)x - b(y)y - cx^2y + a_{lab}p_s\left(\frac{\Delta a + 2\tau\tau}{a_{01} + x + \tau y}\right)\right]}.$$

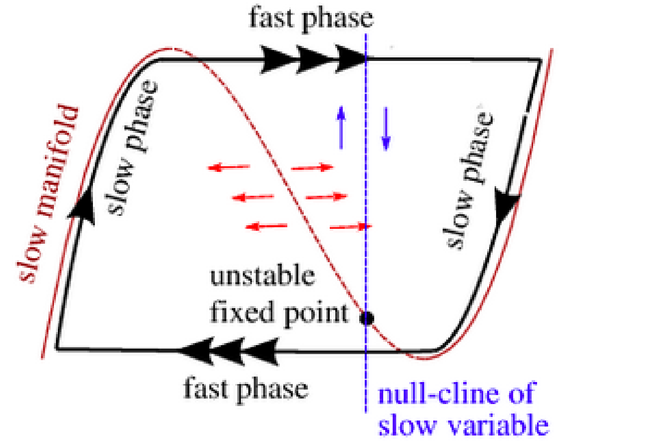
Relaxation oscillator,
where fast regions alternate with
slow regions (two of each per period)



What did we conjecture in 2000?

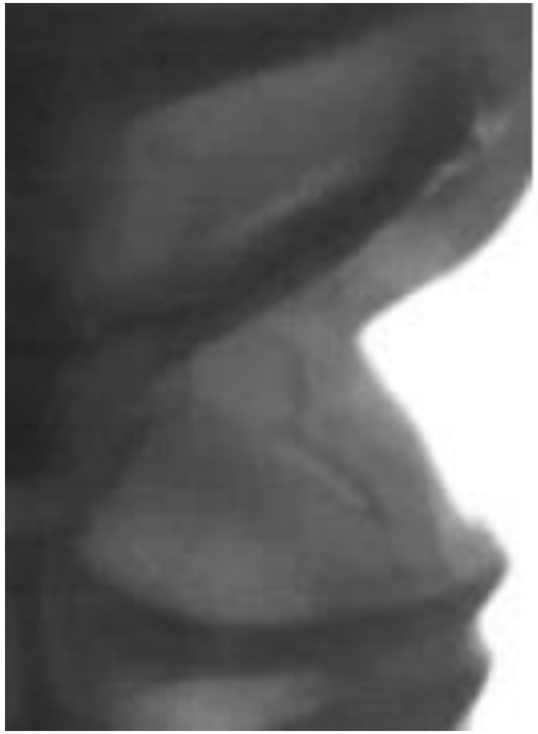
$$\frac{du}{dt} = v - \frac{c}{3}\,u^3 + bu$$

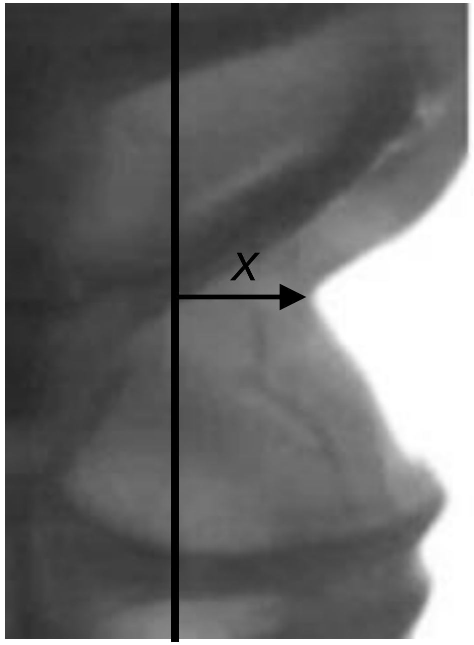
$$\frac{dv}{dt} = -k\left(u + \frac{g}{k}\right)$$



This is a movie of an avian membrane.

It is the equivalent of a vocal fold.
When airflow passes by, it induces oscillations which modulate the airflow, generating sound



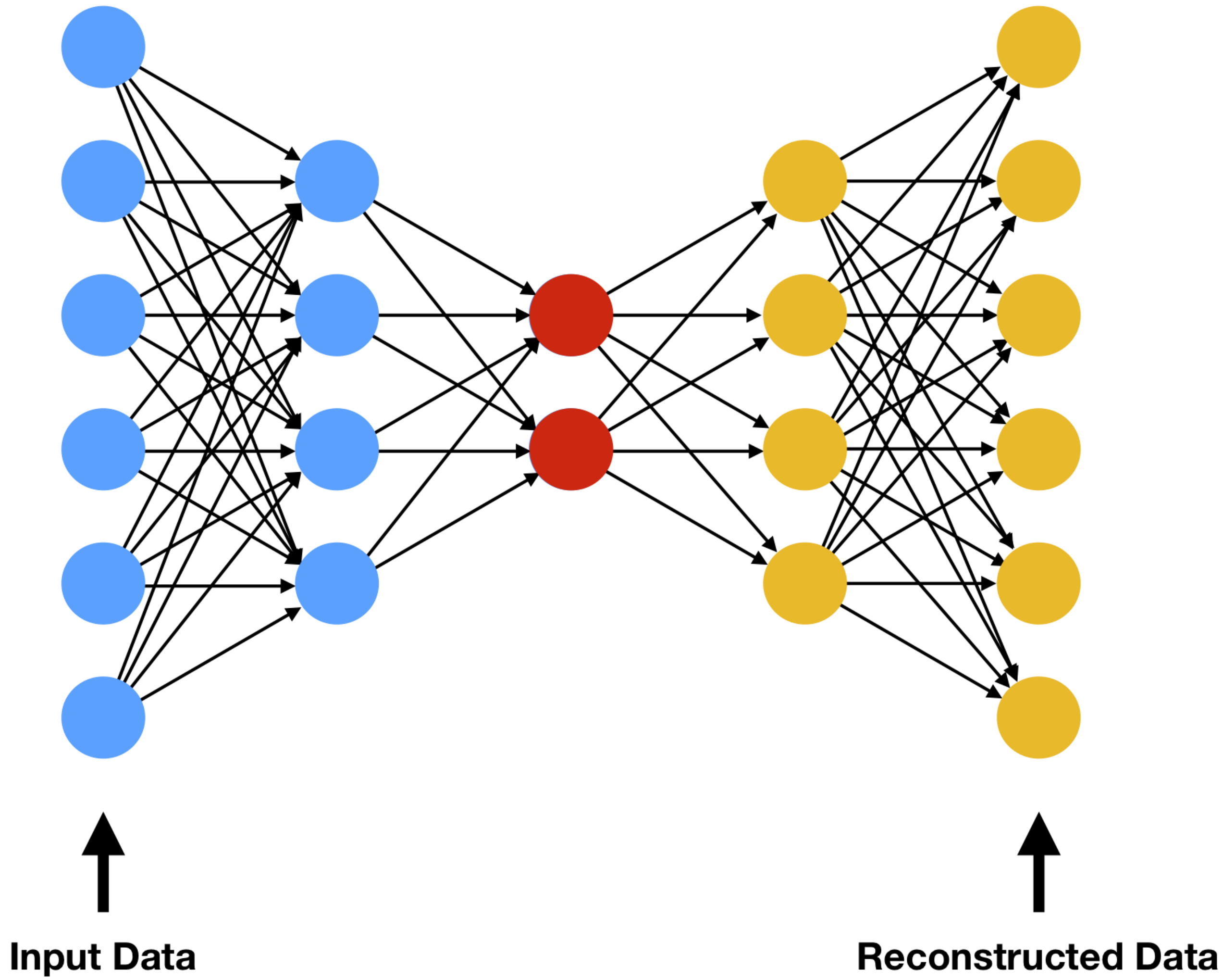


In an “interpretable” paradigm, **we** define *x*

And either phenomenologically or from first principles, we derive an equation for *x*

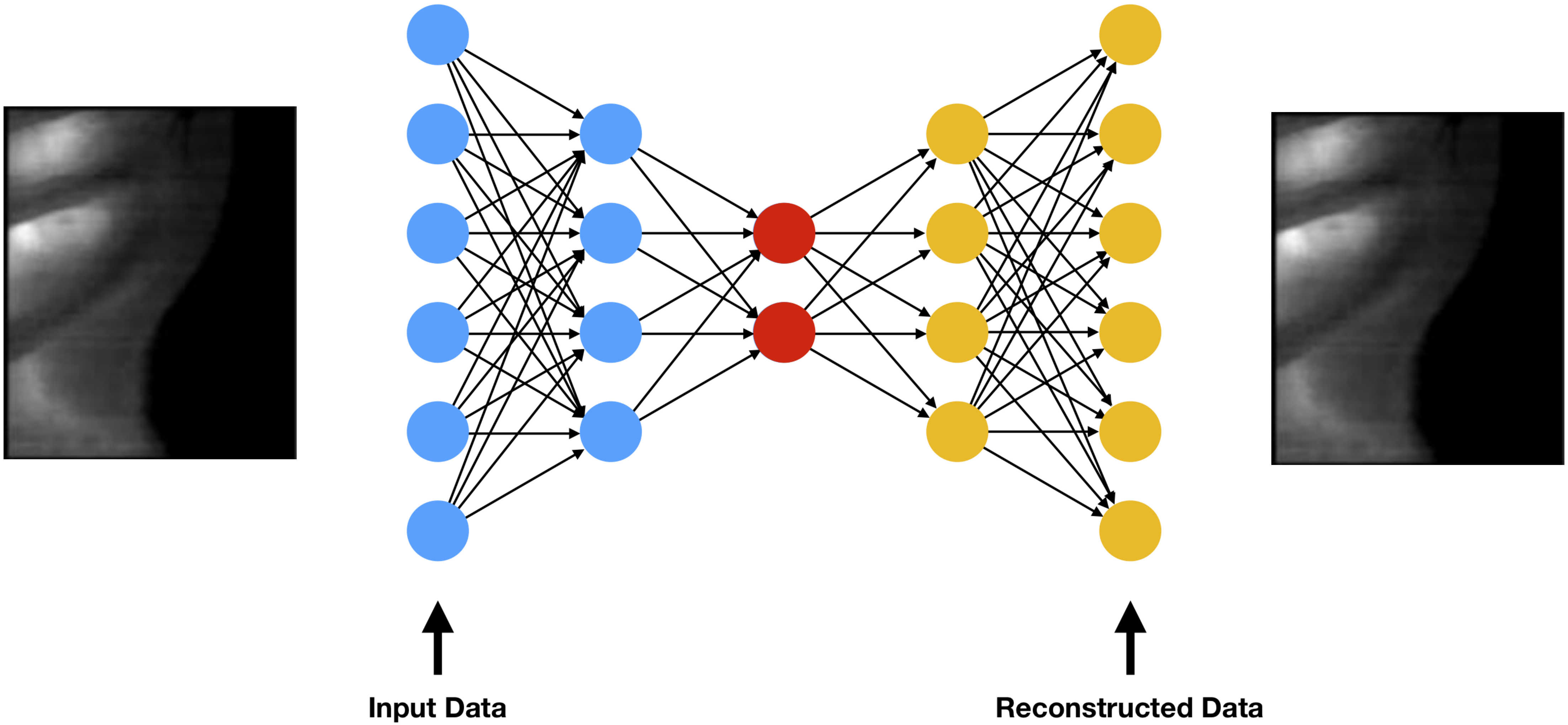
Let us try a different approach

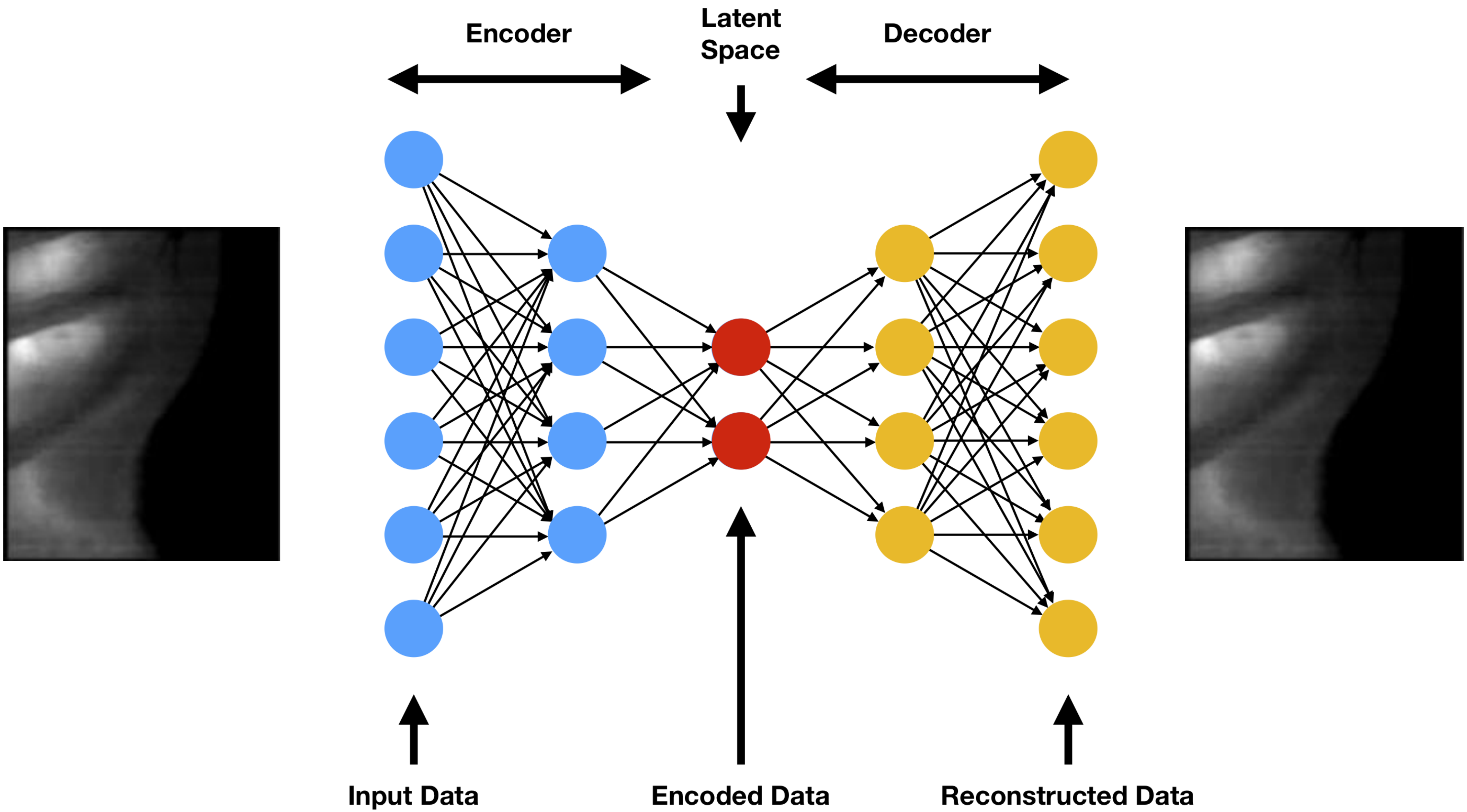
An unpretentious network

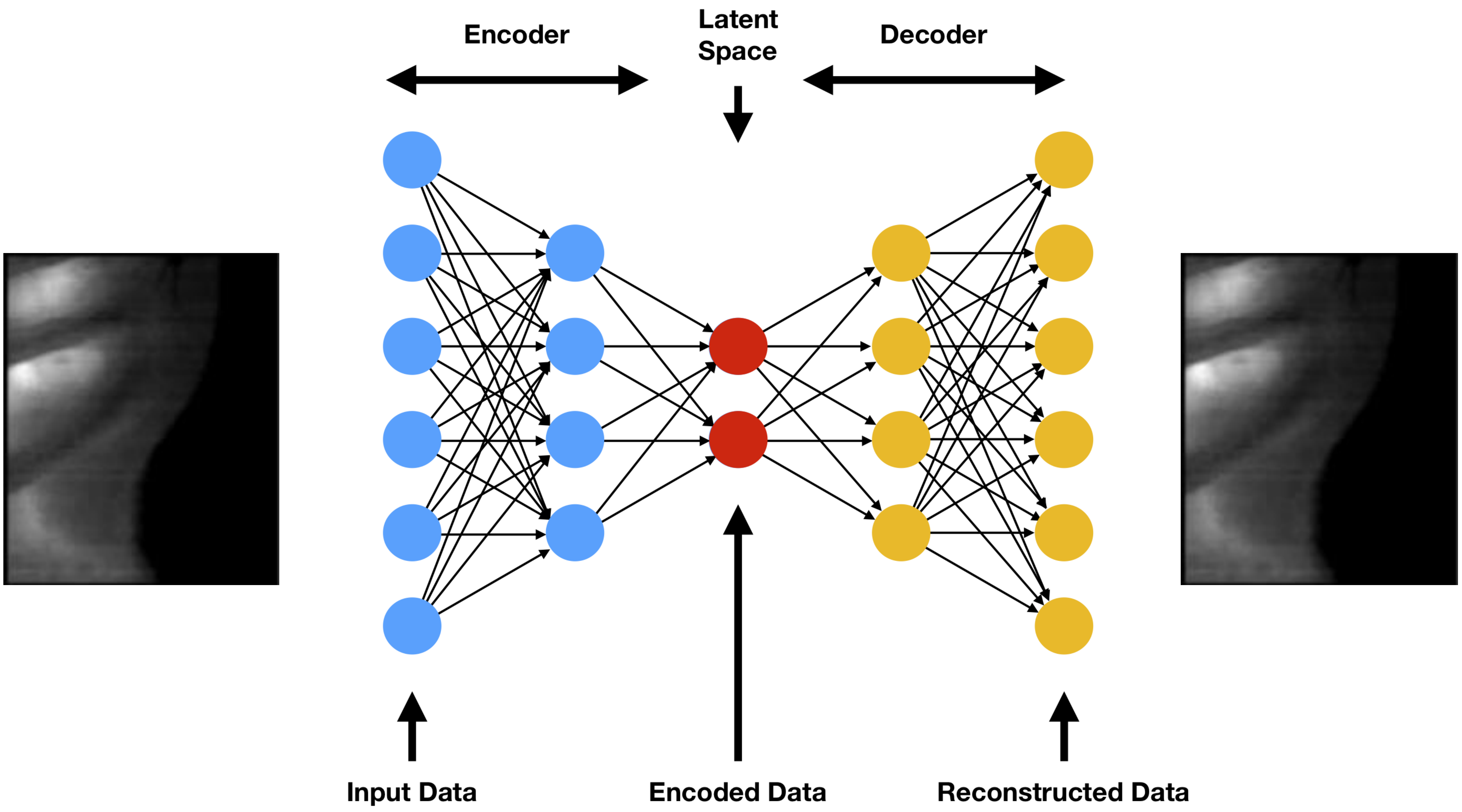


why unpretentious?

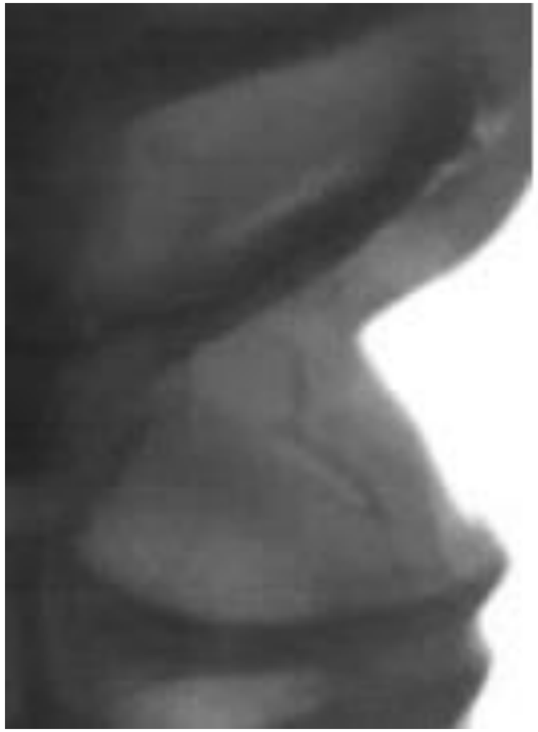
We only ask it to reproduce the input

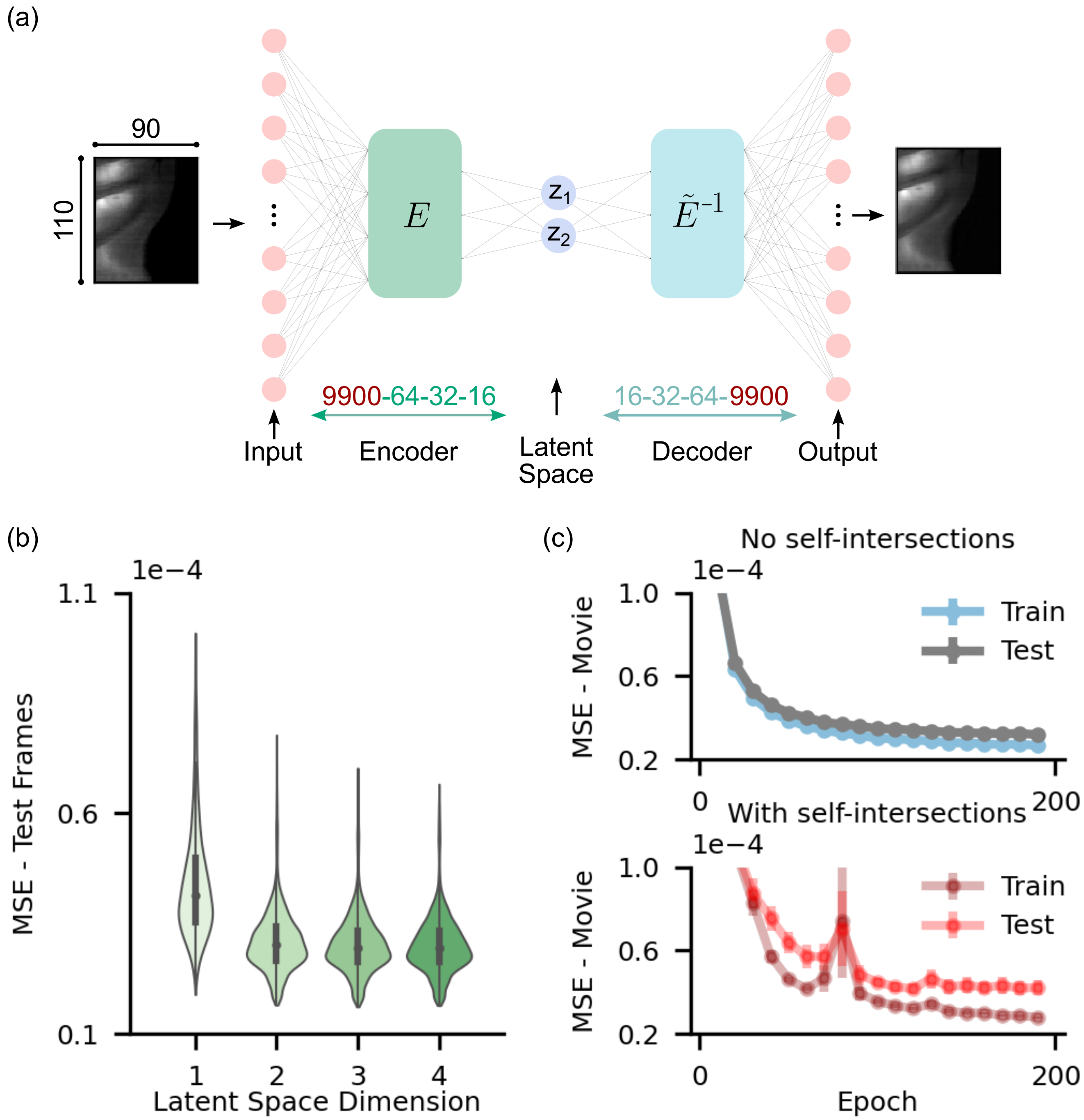



Encoder
Latent Space
Decoder
Input Data
Encoded Data
Reconstructed Data

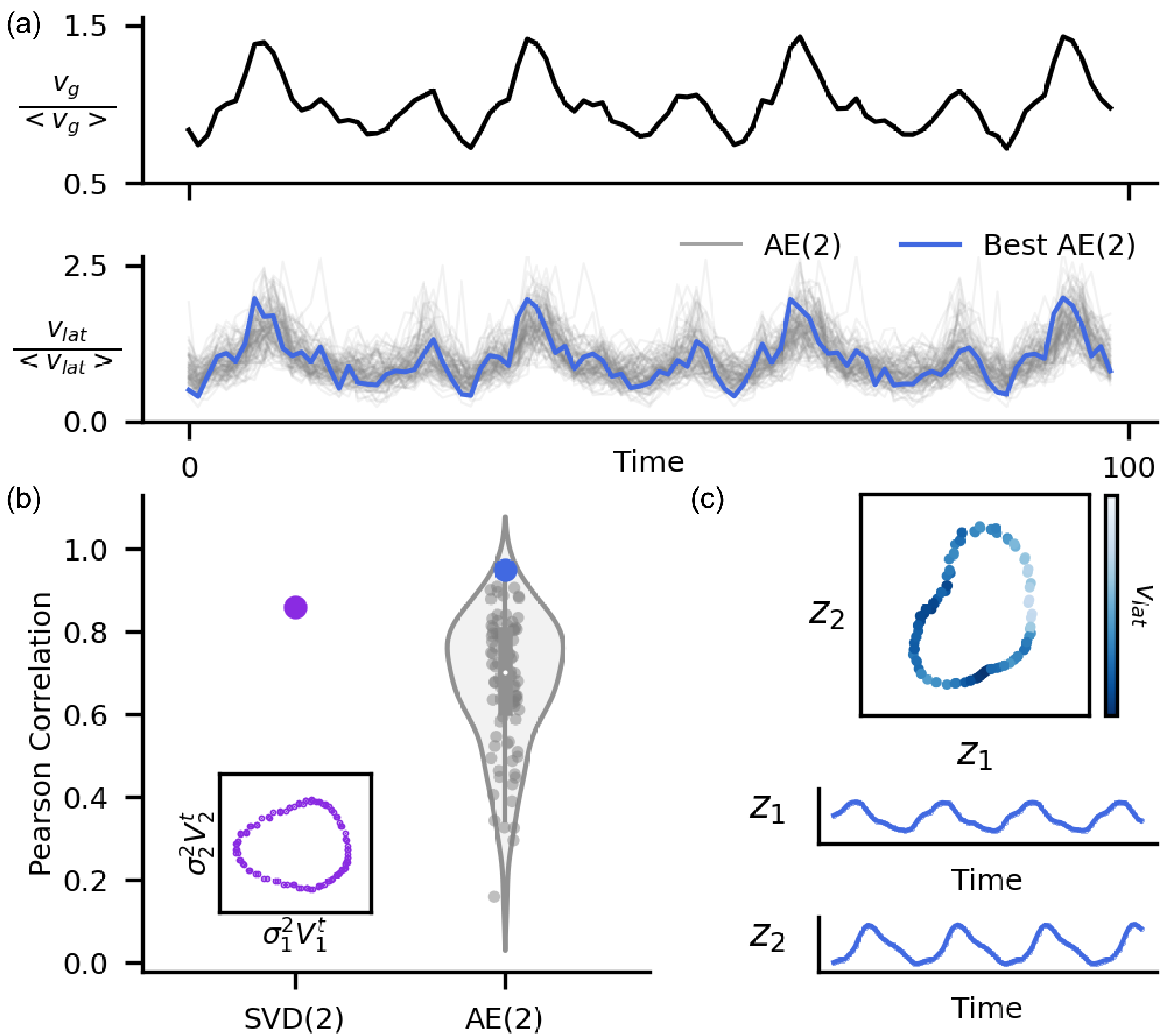


A good coding: no self-intersections in the latent space



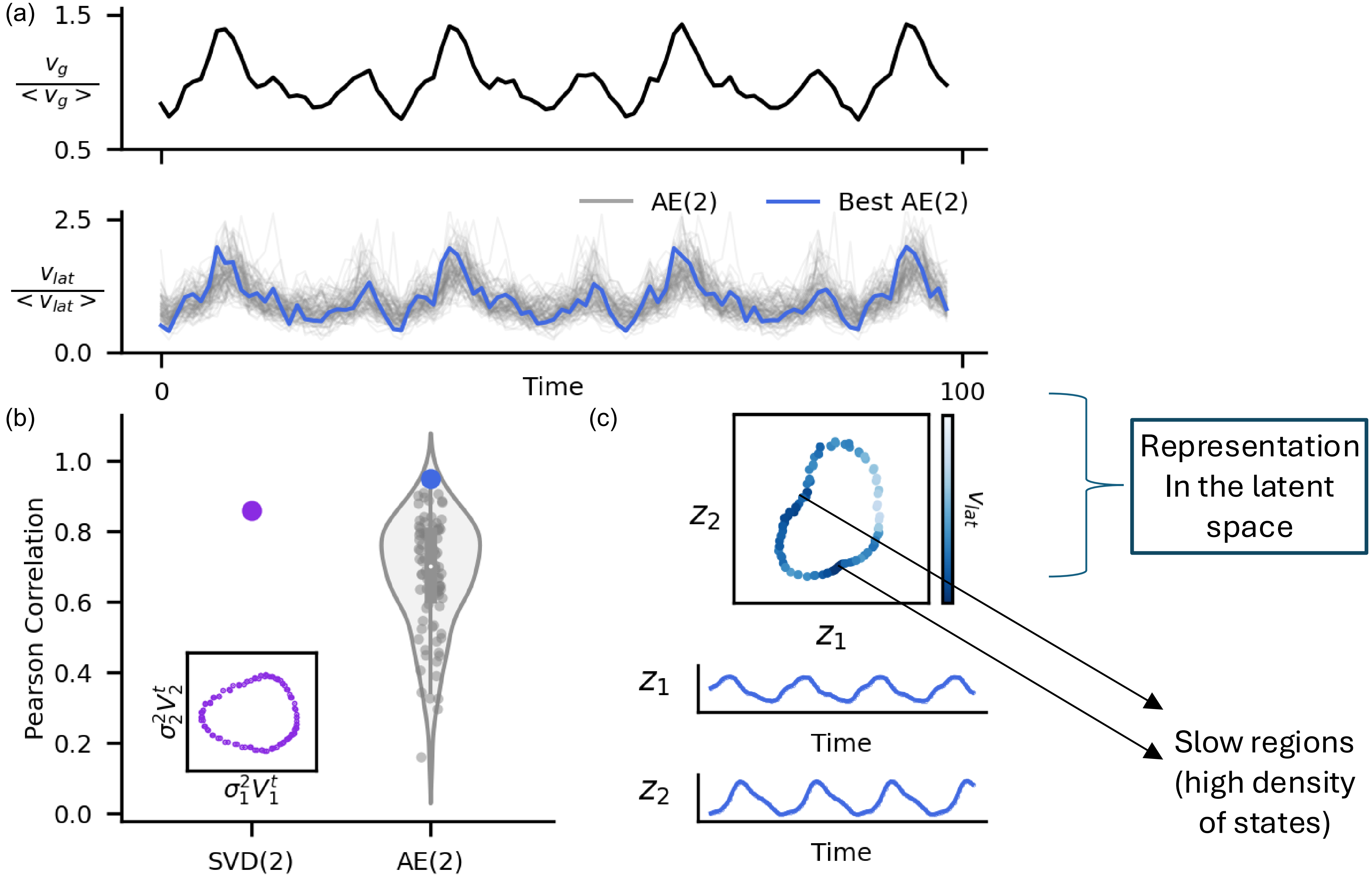

(a)
90
110
E
z_1
z_2
$\tilde{E}^{-1}$
9900-64-32-16
16-32-64-9900
Input
Encoder
Latent Space
Decoder
Output
(b)
1e−4
1.1
0.6
0.1
MSE - Test Frames
1
2
3
4
Latent Space Dimension
(c)
No self-intersections
1e−4
1.0
0.6
0.2
MSE - Movie
Train
Test
0
200
With self-intersections
1e−4
1.0
0.6
0.2
MSE - Movie
Train
Test
0
200
Epoch

What does the network say?



Representation In the latent space

What does the network say?



Without predefining the variables

Without a processing designed to obtain those variables

We reconstruct a “phase space” (a space in which each point has a unique future) in the latent space.

The structure of the reconstructed flow informs on the dynamics in the original phase space.

Without predefining the variables

Without a processing designed to obtain those variables

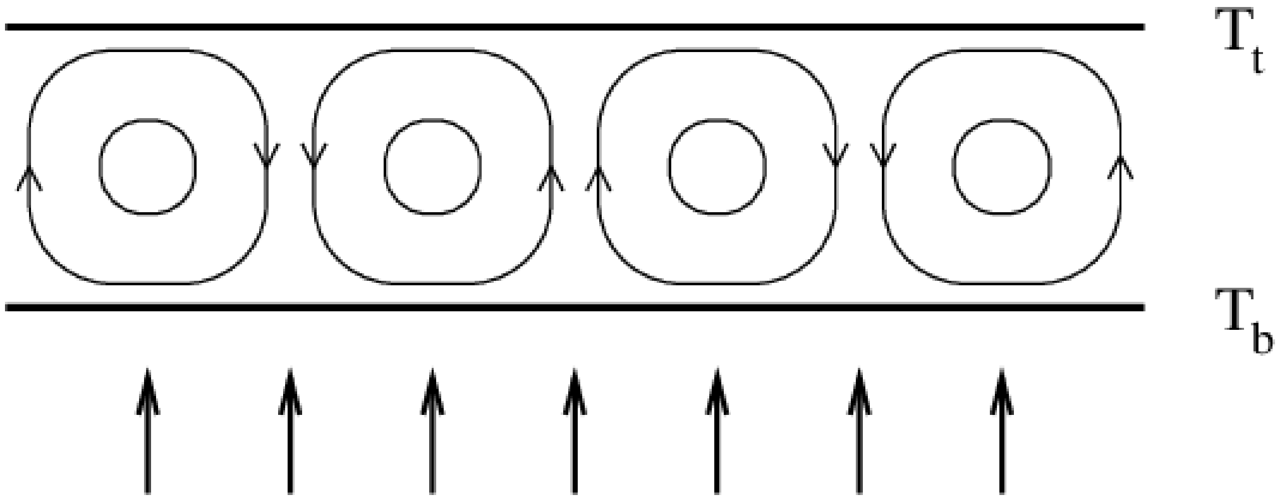
We reconstruct a "phase space" (a space in which each point has a unique future) in the latent space.

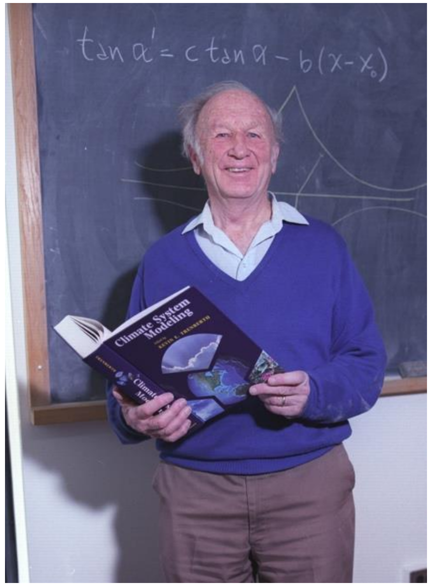
The structure of the reconstructed flow informs on the dynamics in the original phase space.

We'll be back to this

II

La vida no siempre nos da una 2d ODE





La vida no siempre nos da una ODE
(a veces hay que buscarla...)

Las ecuaciones de las que se parte con las de Navier Stokes, que analizan la dinámica de pequeños elementos de fluido, aplicándole al mismo la conservación del momento, teniendo en cuenta que al elemento de fluido lo describimos termodinámicamente (son 10^23 partículas, no una):

$$\frac{\partial \boldsymbol{v}}{\partial t} + (\boldsymbol{v}.\boldsymbol{\nabla})\boldsymbol{v} = \frac{1}{\rho}(\boldsymbol{F} - \boldsymbol{\nabla}\mathrm{p} + \boldsymbol{\mu}\boldsymbol{\nabla}^2\boldsymbol{v})$$

$$\frac{\partial T}{\partial t} + (\boldsymbol{v}.\boldsymbol{\nabla})T = k\boldsymbol{\nabla}^2 T$$

$$\frac{\partial \rho}{\partial t} + \boldsymbol{\nabla}.(\rho\boldsymbol{v}) = \boldsymbol{0}$$

Equaciones a derivadas parciales, que describen el comportamiento de los fluidos. Las variables son la **velocidad** en cada punto, la **temperatura** en cada punto, y la **densidad**.

$$\rho = \bar{\rho}\big(1 - \gamma(T - T_0)\big),$$

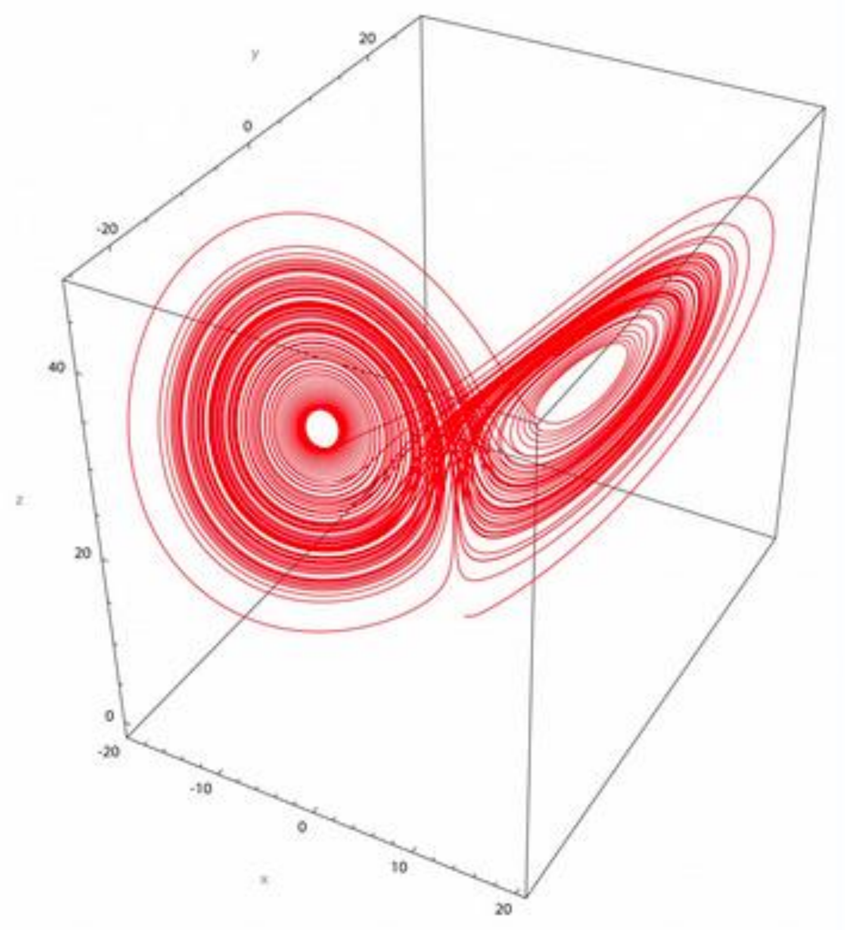
Alimentando con la expansion de los campos de temperatura a las ecuaciones diferenciales a derivadas parciales, y usando la ortogonalidad de los modos espaciales, llega a:

$$\frac{dX}{dt} = \sigma(Y - X)$$

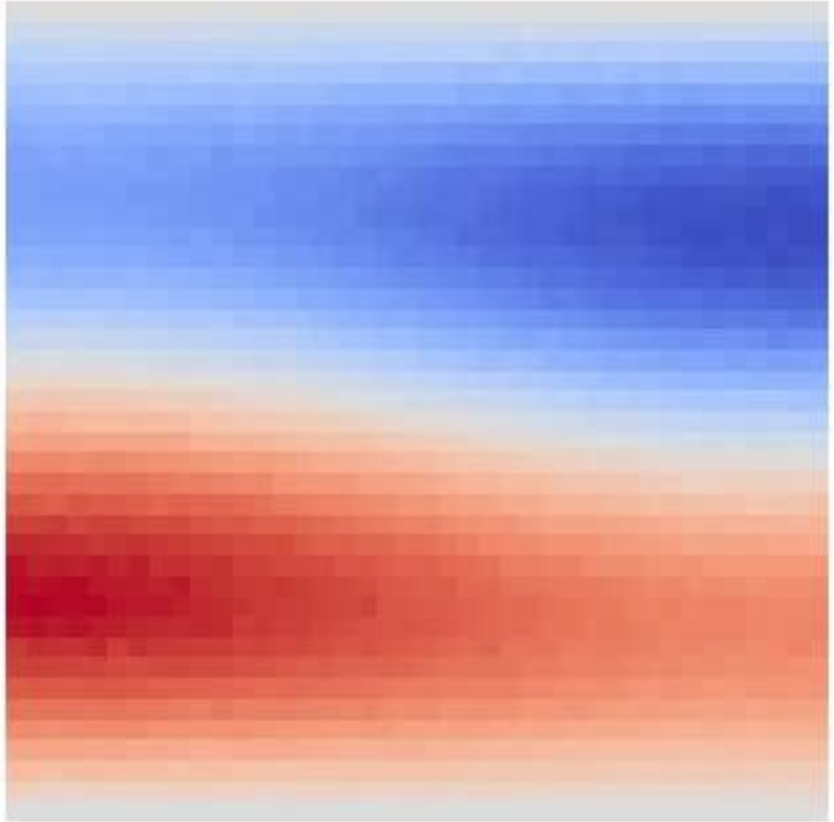
$$\frac{dY}{dt} = rX - Y - XZ$$

$$\frac{dZ}{dt} = XY - bZ$$

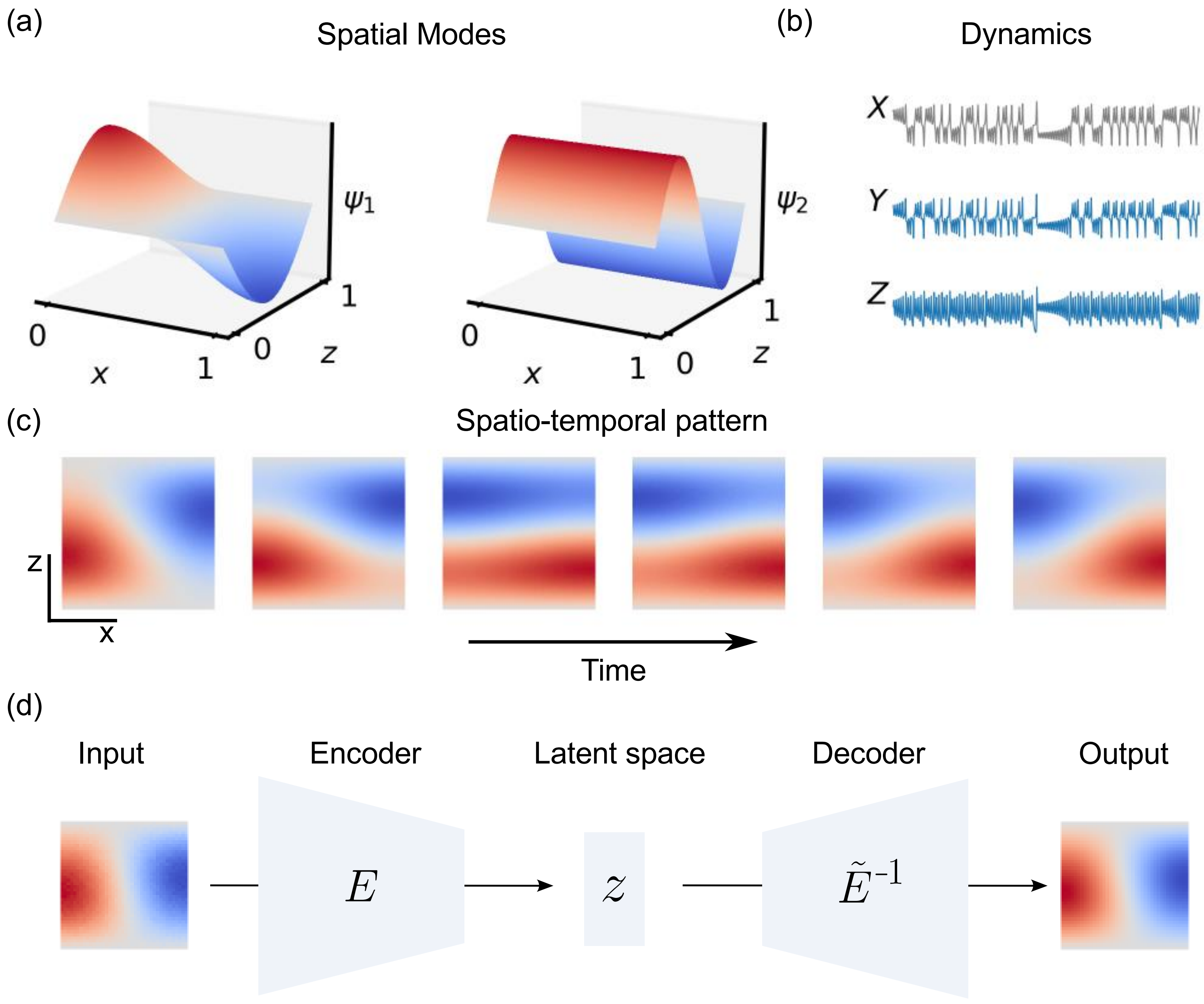
Un atractor muy famoso.
Y muy extraño



Una pelicula muy sintetica

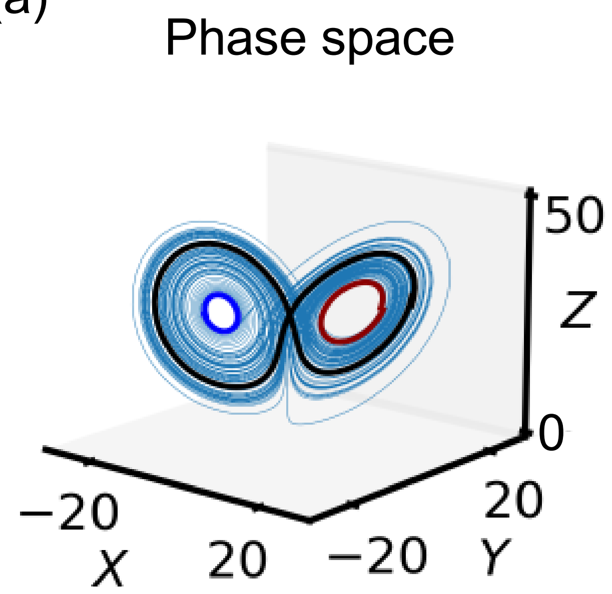


Facundo Fainstein

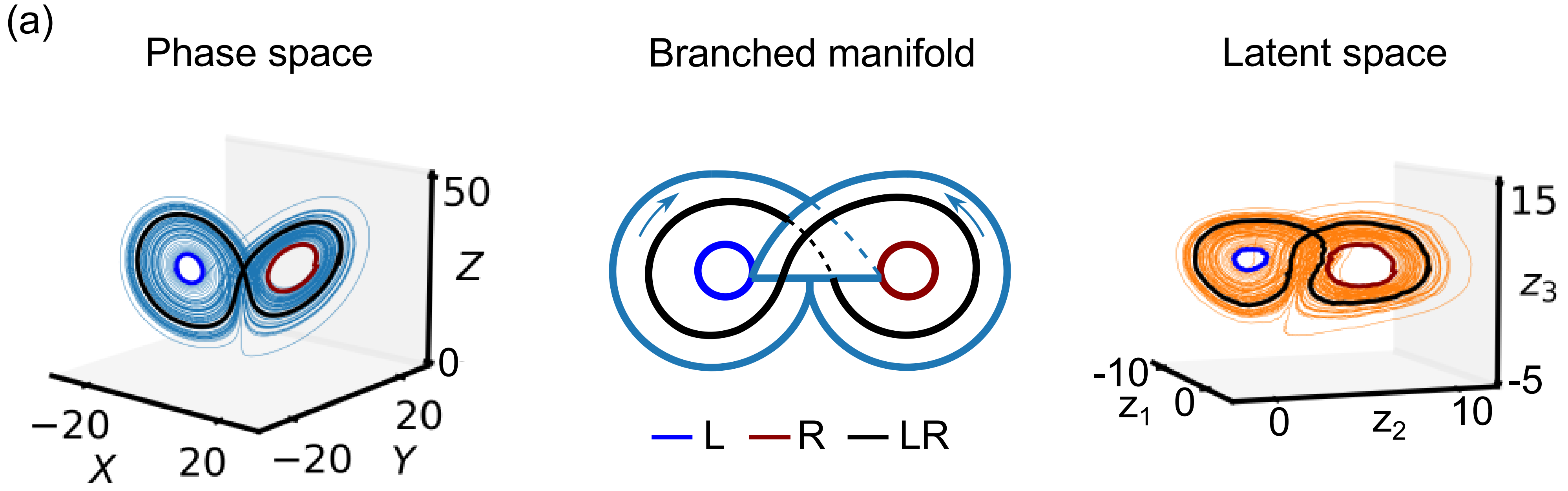


(a)
Spatial Modes
(b)
Dynamics
X
Y
Z
ψ_1
ψ_2
0
x
1
0
z
1
(c)
Spatio-temporal pattern
z
x
Time
(d)
Input
Encoder
Latent space
Decoder
Output
E
z
$\tilde{E}^{-1}$

(a)



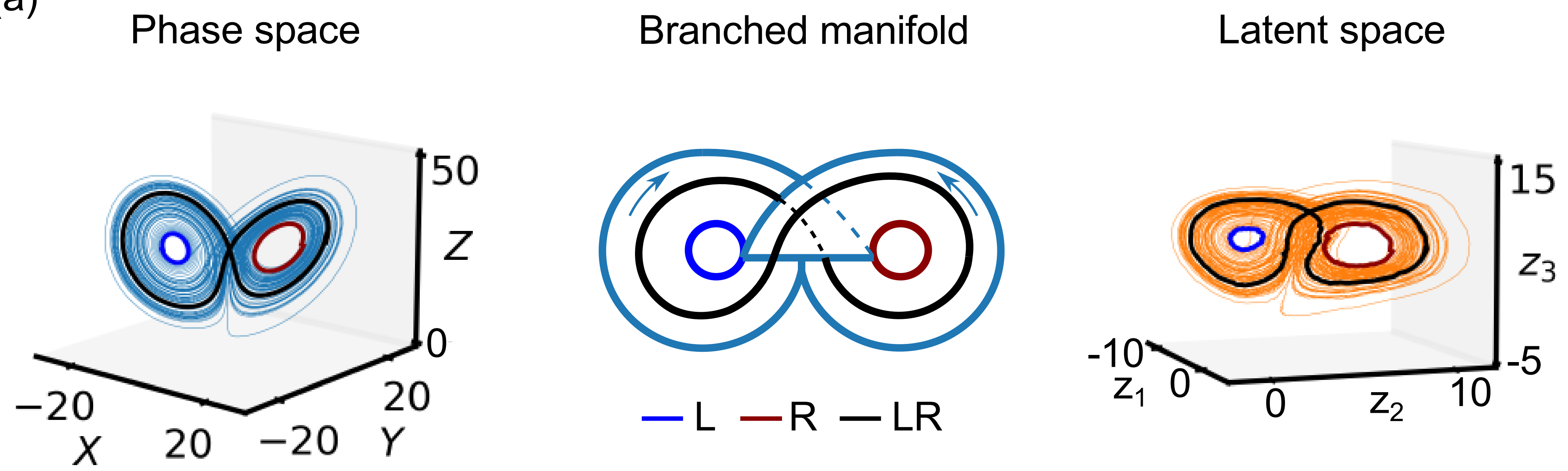
(a)
Phase space
Latent space
50
Z
0
−20
X
20
−20
20
Y
15
z_3
-5
-10
z_1
0
0
z_2
10



El flujo original y el reconstruido se asemejan morfologicamente

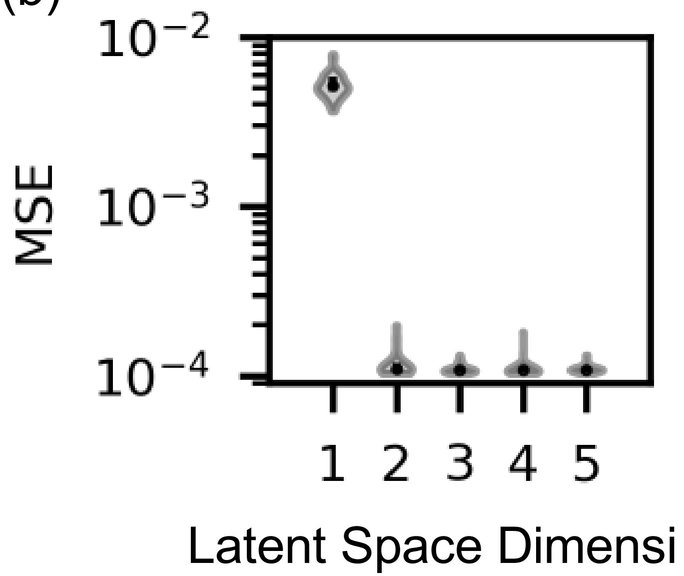
En el espacio latente, las orbitas parecen organizarse como en el **"templado"** del sistema de Lorenz (la variedad enramada en la que podemos acomodar todas las orbitas periodicas para estudiar su topologia)

(a)

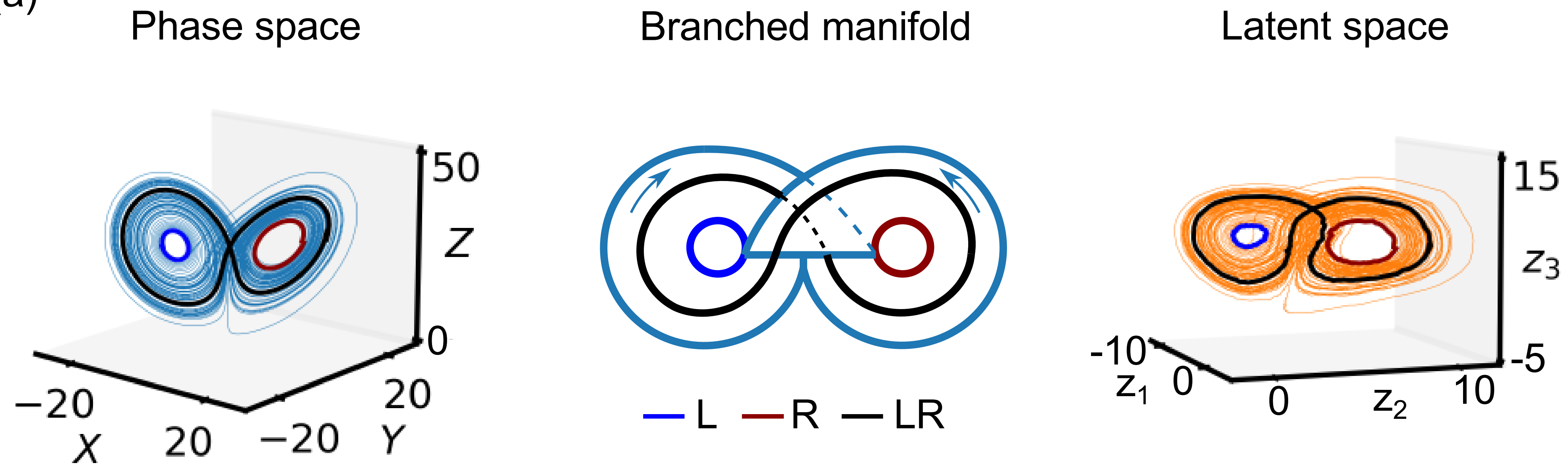

Phase space
Branched manifold
Latent space
50
Z
0
−20
X
20
−20
Y
20
L
R
LR
15
z_3
-5
-10
z_1
0
0
z_2
10

Pero...

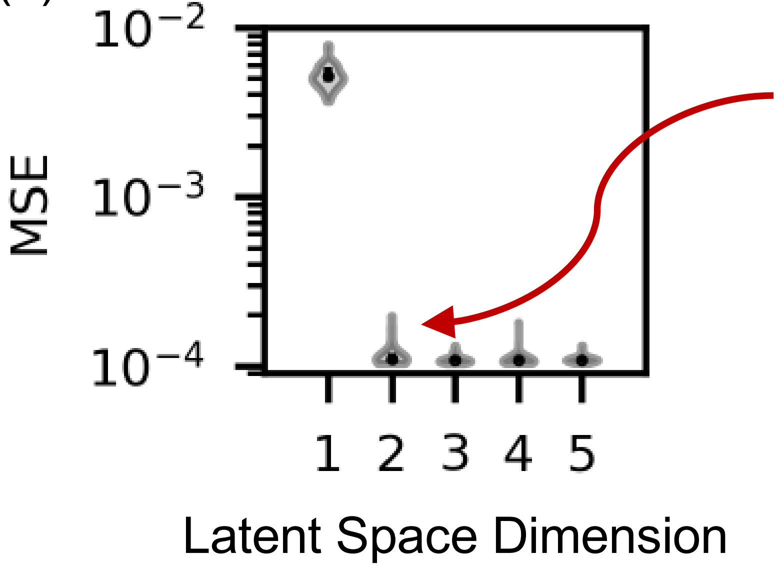
(b)


10^{-2}
10^{-3}
10^{-4}
MSE
1
2
3
4
5
Latent Space Dimensi

(a)

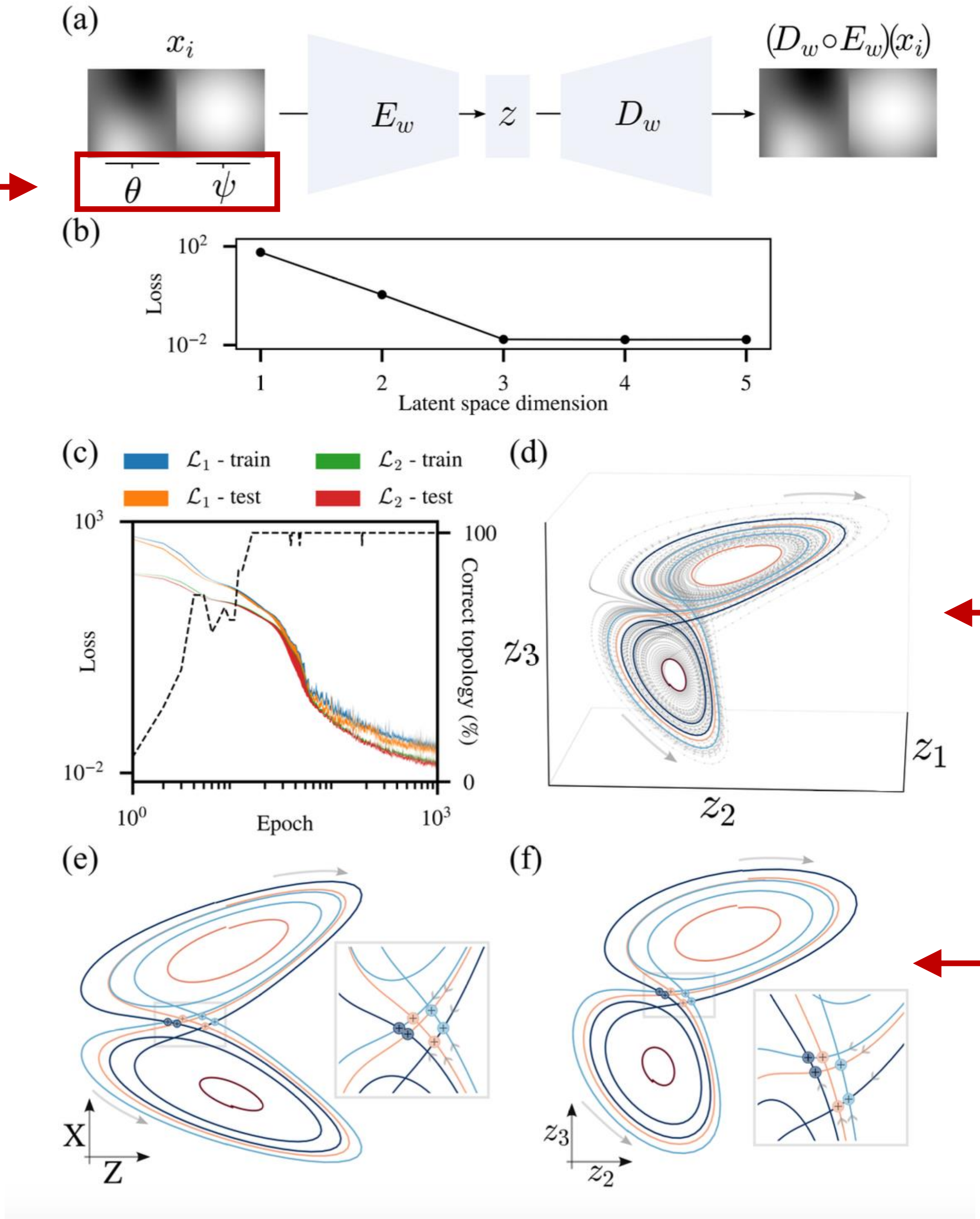


(b)



Guarda... no sos vos, son los datos!

Alimentando con toda la informacion



Ahora si el espacio de embedding es 3d

Y la topologia del sistema en el espacio de fases...

Es la misma topologia que en el espacio latente

En el tratamiento mostrado en la transparencia anterior, hubo otra novedad

Dos terminos en la perdida

$$\mathcal{L}_1(w) = \sum_{x_i} \left|(D_w \circ E_w)(x_i) - x_i\right|^2 ,$$

$$\mathscr{L}_2(w) = \sum_i \left| \frac{(D_w \circ E_w)(x_{i+1}) - (D_w \circ E_w)(x_i)}{t_{i+1} - t_i} - \frac{x_{i+1} - x_i}{t_{i+1} - t_i} \right|^2 .$$

El segundo termino de la perdida se incluyo para asegurar que tenemos un sistema dinamico en el espacio latente.
Para verlo, definimos asi un campo vector:

$$\Phi(z) \doteq \frac{\partial E_w}{\partial(\varphi(x))}(x) \approx \frac{\partial E_w}{\partial(\varphi(D_w(z)))}(D_w(z)).$$

Podemos probar que el mismo constituye el campo de velocidades para las trayectorias , ya que

$$\dot{z} = \frac{d}{dt}E_w(x(t)) = \frac{E_w(x(t+dt)) - E_w(x(t))}{dt} =$$
$$= \frac{E_w(x(t)+\dot{x}(t)dt) - E_w(x(t))}{dt} = \frac{\partial E_w}{\partial(\varphi(x))}(x) = \Phi(z).$$

Por lo tanto, el conjunto de puntos imagen de una trayectoria...
es la trayectoria de un sistema dinamico

El segundo termino de la perdida se incluyo para asegurar que tenemos un sistema dinamico en el espacio latente.
Para verlo, definimos asi un campo vector:

$$\Phi(z) \doteq \frac{\partial E_w}{\partial(\varphi(x))}(x) \approx \frac{\partial E_w}{\partial(\varphi(D_w(z)))}(D_w(z)).$$

Podemos probar que el mismo constituye el campo de velocidades para las trayectorias , ya que

$$\dot{z} = \frac{d}{dt}E_w(x(t)) = \frac{E_w(x(t+dt)) - E_w(x(t))}{dt} =$$
$$= \frac{E_w(x(t)+\dot{x}(t)dt) - E_w(x(t))}{dt} = \frac{\partial E_w}{\partial(\varphi(x))}(x) = \Phi(z).$$

Por lo tanto, el conjunto de puntos imagen de una trayectoria...
es la trayectoria de un sistema dinamico

Si tenemos un mapa continuo E_w, con inversa continua D_w, que mapea orbitas en orbitas, los flujos son topologicamente equivalentes

Notar que este termino de regularizacion

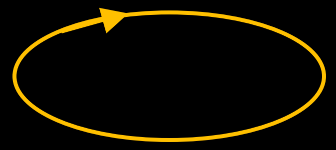
$$\mathcal{L}_2(w) = \sum_i \left| \frac{(D_w \circ E_w)(x_{i+1}) - (D_w \circ E_w)(x_i)}{t_{i+1} - t_i} - \frac{x_{i+1} - x_i}{t_{i+1} - t_i} \right|^2 .$$

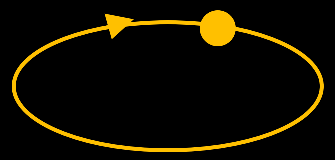
Da lugar a que se cumpla que

$$\frac{\partial \left(D_w \circ E_w\right)}{\partial(\varphi(x))}(x) = \frac{\partial D_w}{\partial E_w} \frac{\partial E_w}{\partial(\varphi(x))} = \frac{\partial D_w}{\partial E_w} \Phi(z) \approx \varphi(x) .$$

Y por lo tanto, $\Phi \neq 0$ si $\varphi \neq 0$

No mapea orbitas en arcos conectores

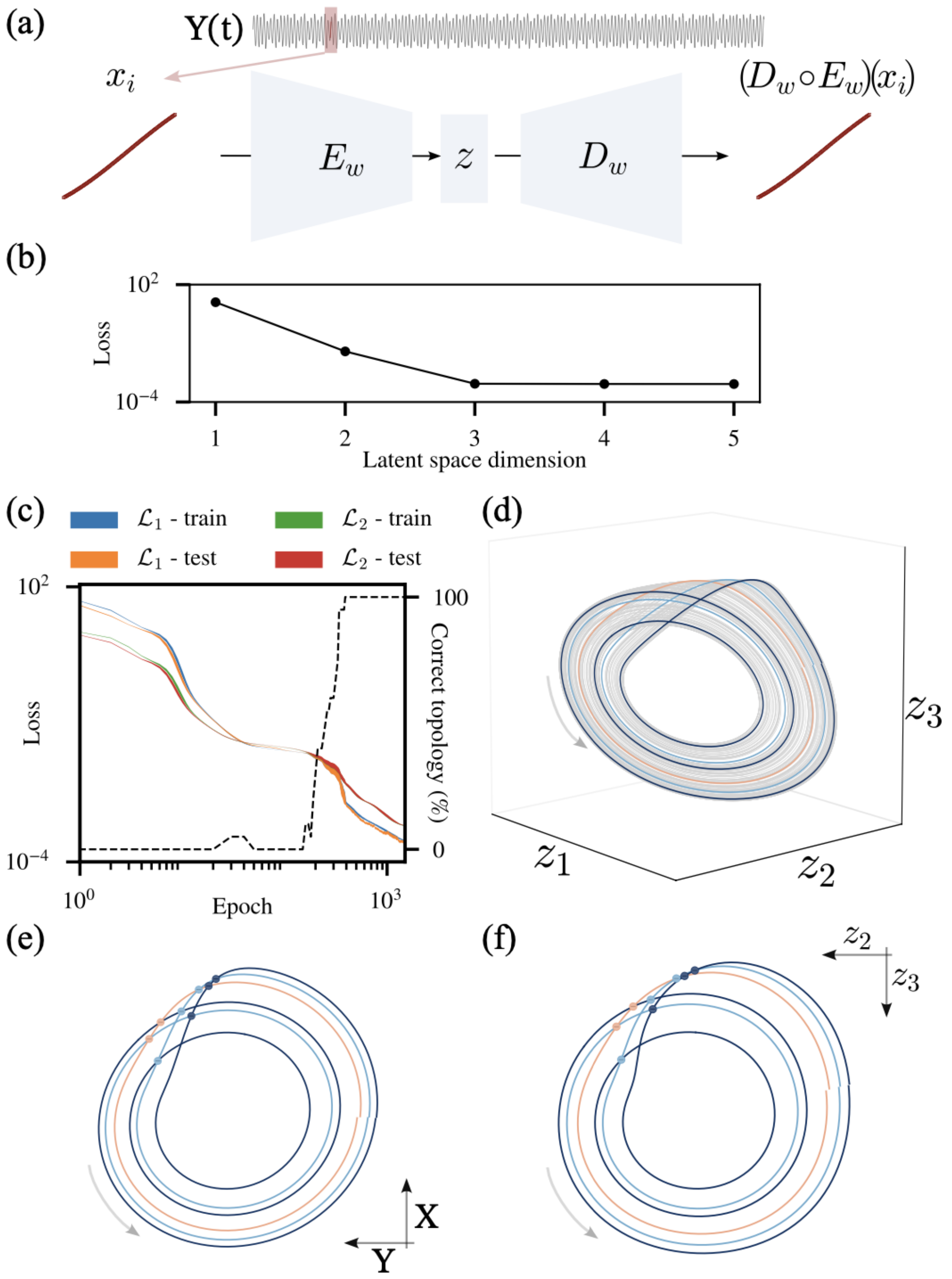




La pintura que emerge es que detras del exito de algunas tareas por parte de la redes neuronales una verdadera representacion de la dinamica del problema.
el espacio latente seria un espacio de embedding de la dinamica

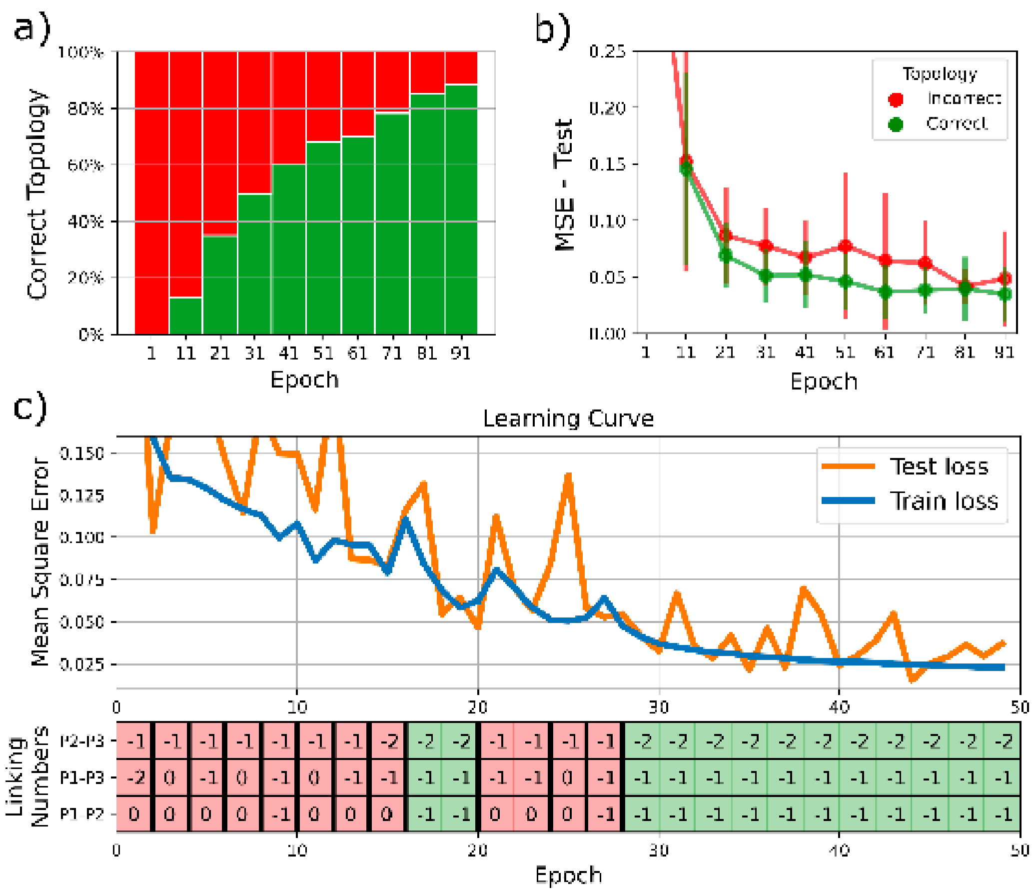
Con Facu Fainstein y Pablo Groisman

Si alimentamos a nuestra red con segmentos de señal temporal escalar



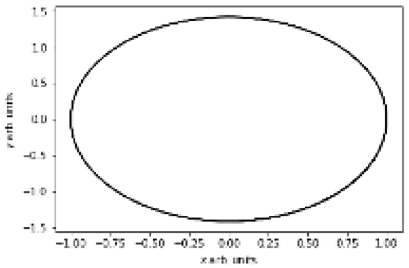
En el espacio latente encontramos soluciones con la topologia del flujo original

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

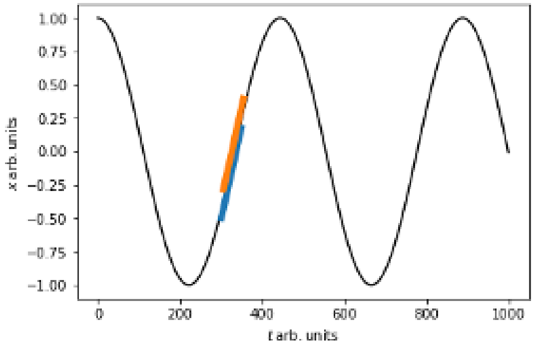


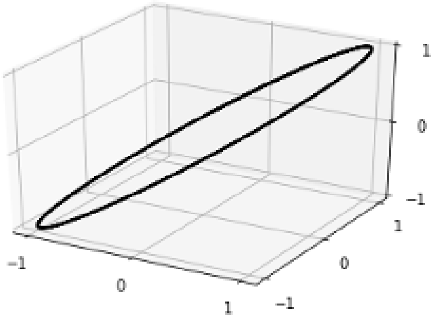
Como se hacia antes?

The variables $(x(t), y(t))$



$\{x_1, x_2, \ldots, x_N\}$.





$x(t) \rightarrow (\mathrm{x}(t), x(t-\tau), x(t-2\tau))$

checking determinism

$\{x_k, x_{k+1}, \ldots, x_j \ \ldots, x_{N_1+k-1}\}$

$\{x_{k+1}, x_{k+2}, \ldots, x_j \ \ldots, x_{N_1+k}\}$

With a sparse selection (2d+1)

Vimos en la teórica que la red se entrena calculando el **gradiente** de los pesos respecto a la función de costo mediante el proceso de **Backpropagation**.

Vimos en la teórica que la red se entrena calculando el **gradiente** de los pesos respecto a la función de costo mediante el proceso de **Backpropagation**.

Si mi set de entrenamiento está compuesto por **N** instancias.

¿Cómo calculo este gradiente?

¿Instancia a instancia?

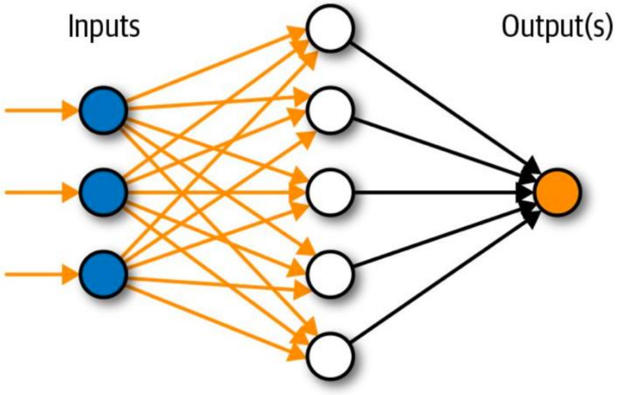
¿Promedio sobre todas?

Usando todo el dataset (Gradient Descent tradicional),
calcula los gradientes sobre todos los datos.
Esto es preciso pero **muy costoso computacionalmente**,
especialmente con datasets grandes.

Batches y Epochs

Batch

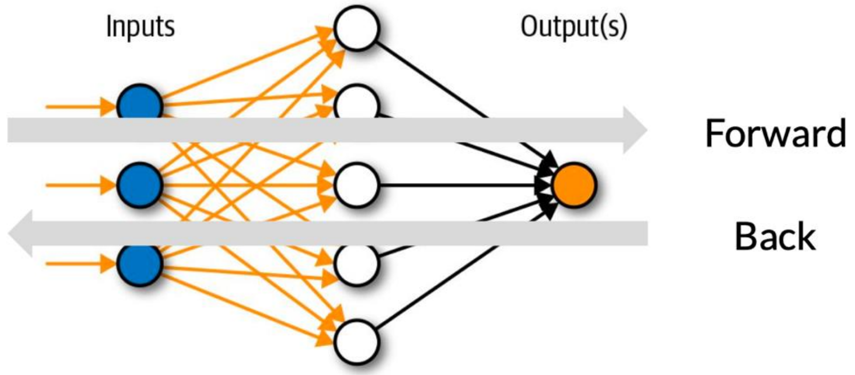
Conjunto de instancias.



Batches y Epochs

Batch

Conjunto de
instancias.



Una **Iteración** (pasada): Computo el costo J, computo sus derivadas y actualizo los pesos de la red.

Como elegir sobre cuantos datos
computar el gradiente?

Batches y Epochs

Batch

Conjunto de
instancias.

Batches y Epochs

Batch

Conjunto de instancias.

1 única instancia: **Stochastic**

m instancias (m<<N): **Mini-Batch**

N instancias (todo el training set): **Batch**

Batches y Epochs

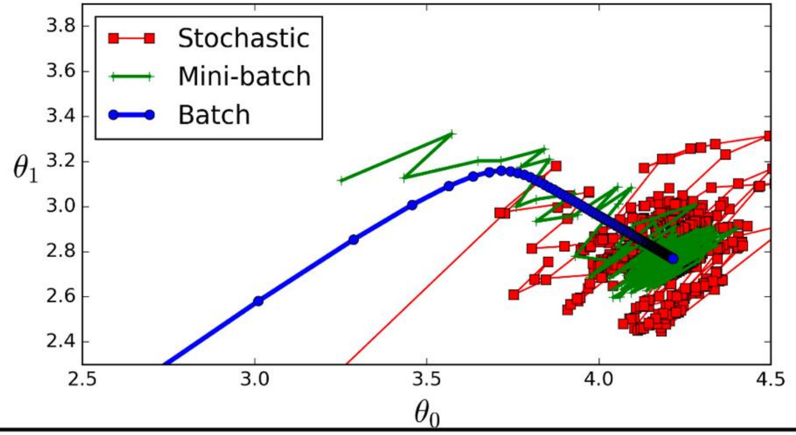
Batch

Conjunto de instancias.

1 única instancia: **Stochastic**

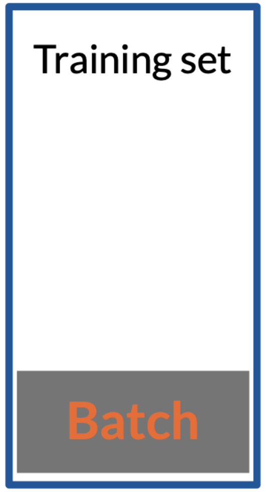
m instancias (m<<N): **Mini-Batch**

N instancias (todo el training set): **Batch**



Batches y Epochs

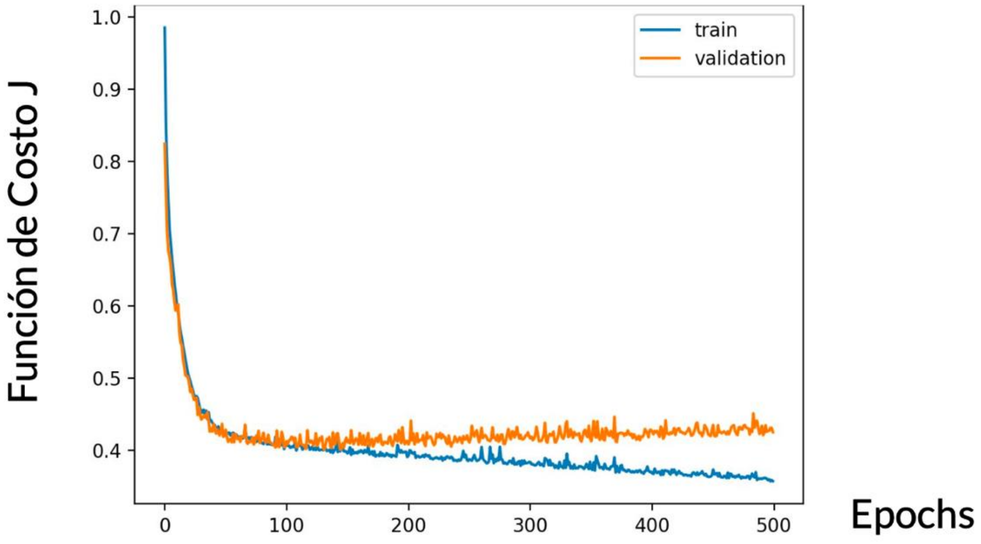
Epochs: Es la cantidad de veces que pasamos **el training set completo** por la red.



Noten que si el número de instancias **m** en el batch es mucho menor que el número de instancias **N** en todo el training set, vamos a necesitar varias **iteraciones** para completar un epoch.

Batches y Epochs

En general se precisan varios **Epochs** para entrenar la red.



Optimizadores

Vamos a comentar 3 métodos de optimización:

- Método 1: SGD
- Método 2: Momentum
- Método 3: ADAM

1. Backpropagation:

Es el **mecanismo de cálculo de gradientes** en redes neuronales. Consiste en:

1. Propagar hacia adelante (**forward pass**) la entrada a través de la red y calcular la pérdida con respecto a la salida esperada.
2. Propagar hacia atrás (**backward pass**) la derivada de la pérdida con respecto a los pesos de cada capa, utilizando la **regla de la cadena** para actualizar los parámetros de la red.

En resumen, backpropagation es el método que calcula los gradientes de la función de pérdida con respecto a los pesos de la red.

2. Optimizer:

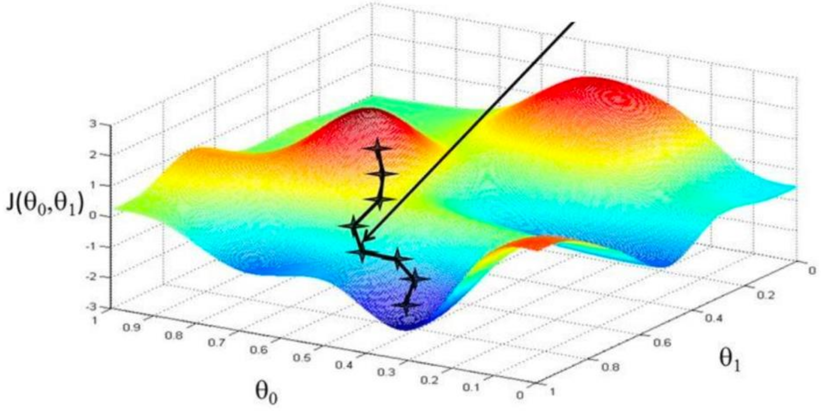
El optimizador es el algoritmo que **usa los gradientes calculados por backpropagation para actualizar los pesos de la red** de manera eficiente. Algunos ejemplos de optimizadores son:

- **SGD (Stochastic Gradient Descent):** Actualiza los pesos en la dirección opuesta al gradiente, con una tasa de aprendizaje fija.
- **Adam:** Una versión más avanzada que usa momentum y adaptación de la tasa de aprendizaje para cada peso.
- **RMSprop:** Similar a Adam, pero con una estrategia diferente de ajuste de la tasa de aprendizaje.

Optimizadores: SGD

A este proceso de actualizar los pesos a partir de calcular el gradiente en un batch (y no en todo el dataset) se lo llama **Stochastic Gradient Descent (SGD)**.

$$\theta = \theta - \alpha \nabla_\theta J(\theta)$$



Optimizadores: SGD

A este proceso de actualizar los pesos a partir de calcular el gradiente en un batch (y no en todo el dataset) se lo llama **Stochastic Gradient Descent (SGD)**.

$$\theta = \theta - \alpha \nabla_{\theta} J(\theta)$$

Keras

```
opti = tf.keras.optimizers.SGD( learning_rate=0.01)
model.compile(loss = 'mse', optimizer=opti)
```

Optimizadores: Momentum

La idea es incorporar **inercia** al término de actualización de los pesos, esto quiere decir que dependa del valor de actualización de la iteración anterior. Se busca acelerar el proceso de convergencia y ayudar a superar mínimos locales.

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta)$$
$$\theta = \theta - v_t$$

Optimizadores: Momentum

La idea es incorporar **inercia** al término de actualización de los pesos, esto quiere decir que dependa del valor de actualización de la iteración anterior. Se busca acelerar el proceso de convergencia y ayudar a superar mínimos locales.

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta)$$
$$\theta = \theta - v_t$$

Keras

```
opti = tf.keras.optimizers.SGD( learning_rate=0.01, momentum=0.9)
model.compile(loss = 'mse', optimizer=opti)
```

Optimizadores: Adam

Además de la **inercia**, el método ajusta el Learning Rate para cada parámetro teniendo en cuenta el cuadrado del gradiente correspondiente a ese parámetro.

Inercia - $m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$

2do momento - $v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$

Actualización - $\theta_{t+1} = \theta_t - \dfrac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t$

Optimizadores: Adam

Además de la **inercia,** el método ajusta el Learning Rate para cada parámetro teniendo en cuenta el cuadrado del gradiente correspondiente a ese parámetro.

Inercia - $$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$

2do momento - $$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

Actualización - $$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t$$

Corregir inicio:

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

Optimizadores: Adam

Además de la **inercia,** el método ajusta el Learning Rate para cada parámetro teniendo en cuenta el cuadrado del gradiente correspondiente a ese parámetro.

Actualización -

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t$$

Keras

```
opti = tf.keras.optimizers.Adam(learning_rate=0.001, beta_1=0.9, beta_2=0.999)
model.compile(loss = 'mse', optimizer=opti)
```