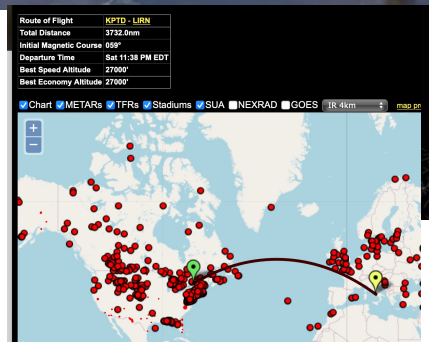


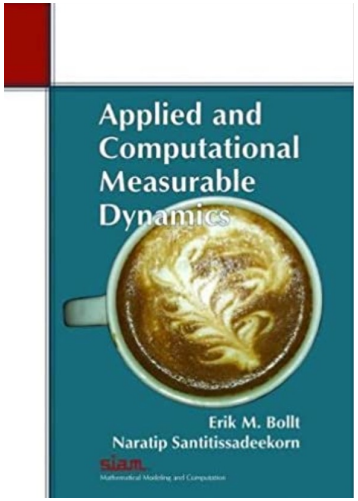
Erik Bollt

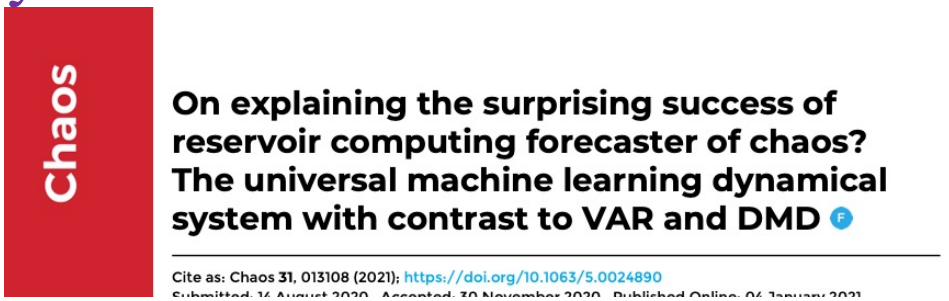
-bolltem@clarkson.edu,

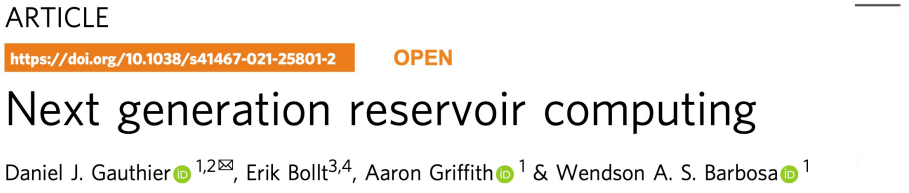
-http://www.clarkson.edu/~bolltem

How Neural Networks Work: Unraveling the Mystery of Random Projection Networks For Functions and Chaotic Dynamical Systems

-bolltem@clarkson.edu,
-http://www.clarkson.edu/~bolltem







So much great work about how well ANN works - WOW they sure do "work" we can all agree.

When I first saw "**multi-layer perceptrons**" over 25 years ago - as a student –
I thought - wow that's dumb. So much over fitting. So much computation – surely there is a better way.

Somehow along the way it emerged as a leader

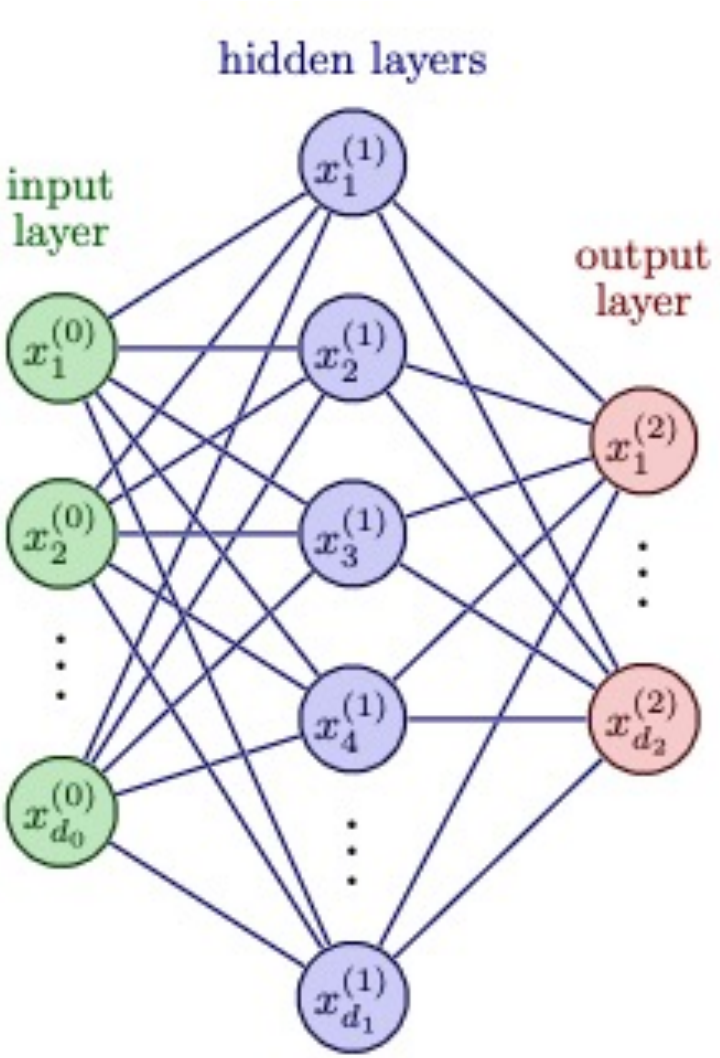
Great **Salesmanship**
– machine learning, neural nets, deep learning, AI, so many variants…

OK OK – let me in – I want to understand **why it works**

Random ANN variants, RC, and **here ELM**.

All the great things you can do with RC, ELM, Feedforward DNN, etc.

Here focused on how/why does it work – with random parameters - it still works?!!



Extreme Learning Machines – ELM
ELM-1

For supervised learning tasks like –

-classification
-regression
-incl forecasting

Here's the crazy part – all but output layer has random parameters

$$\mathcal{D} = \{(X_i, Y_i)\}_{i=1}^{N} \subset \mathbb{R}^{d_0} \times \mathbb{R}^{d_2} = D.$$

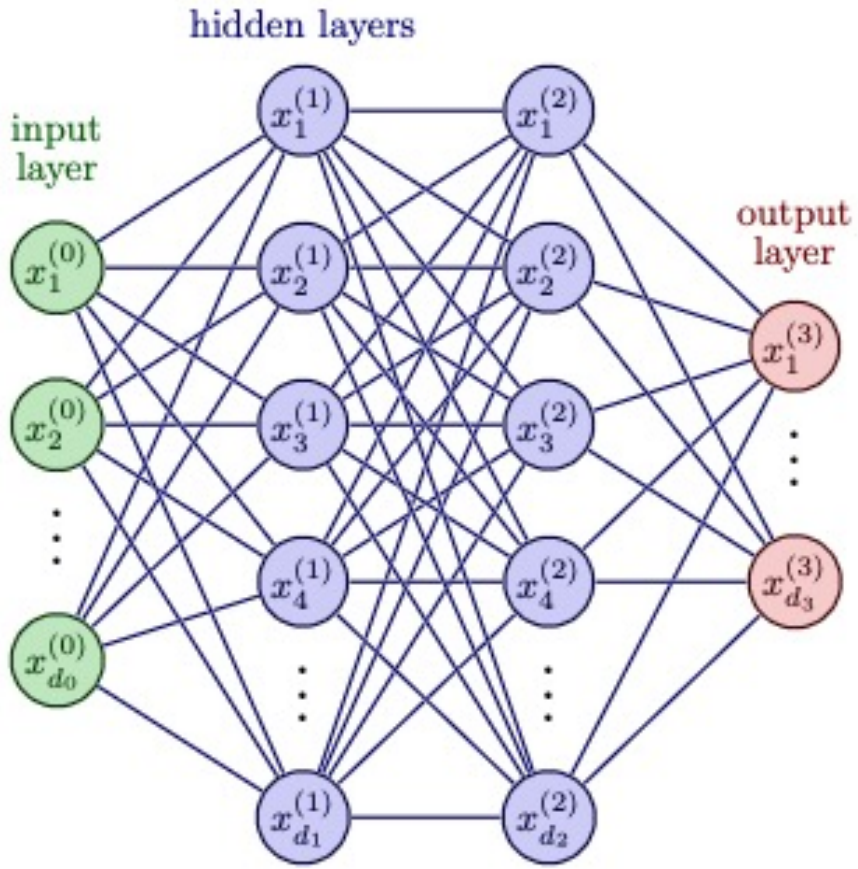
$$f : \mathbb{R}^{d_0} \to \mathbb{R}^{d_2}$$

$$X^{(1)} = [x_1^{(1)}; x_2^{(1)}; ...; x_{d_1}^{(1)}] \in \mathbb{R}^{d_1}.$$

$$X^{(2)} = [x_1^{(2)}; x_2^{(2)}; ...; x_{d_2}^{(2)}] \in \mathbb{R}^{d_2}.$$

I want to explain why it works anyway.

Deep Learning

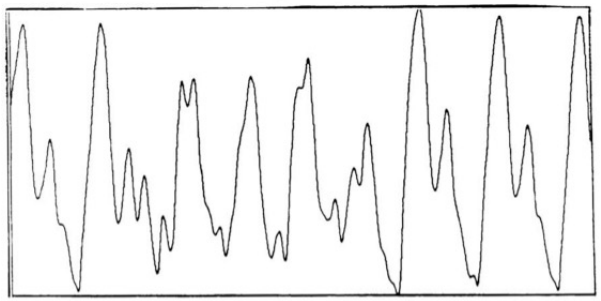


How to train? Lots of parameters, massive nonlinear optimization problem? Massively parallel computers. Stochastic gradient descent, etc – ***not what we are studying here***.

What’s old is new again

How Neural Nets Work
Alan Lapedes
Robert Farber
Theoretical Division
Los Alamos National Laboratory
Los Alamos, NM 87545

$$\dot{x} = \frac{ax(t-\tau)}{1+x^{10}(t-\tau)} - bx(t)$$ Glass-Mackey



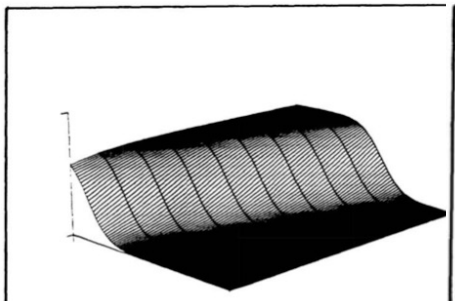


Figure 4. A sigmoidal surface.

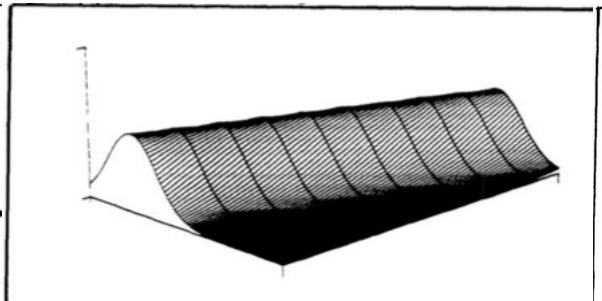


Figure 5. A ridge.

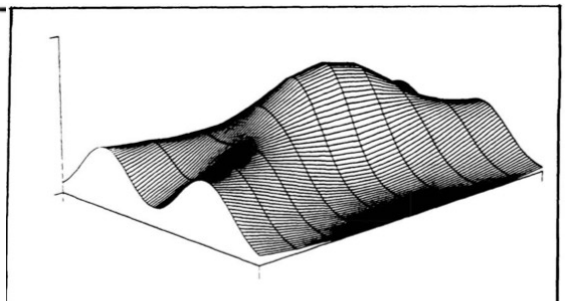


Figure 6. A pseudo-bump.

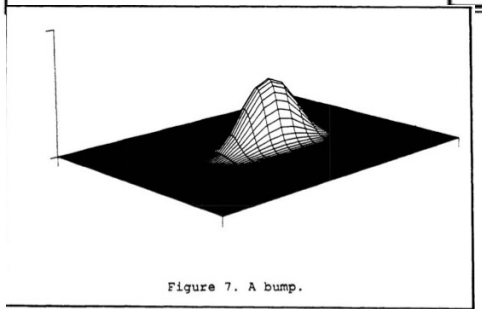


Figure 7. A bump.

Main idea:

You can make bump functions and with bump functions

you can make general functions including a flow.

Thanks Constantinos

Random Projection Neural Networks

- Schmidt et al. (1992): fixing the weights between the input and the hidden layer at random values, and by solving a linear problem for the output weights the approximation accuracy is equivalent to that obtained with back-propagation.
- Random Vector Functional-Link Networks (RVFLNs) were addressed in Pao et al. (1992) in which the input layer is directly connected also to the output layer, the internal weights are chosen randomly in $[-1, 1]$: the output weights are estimated in one step by solving a system of linear equations.
- Igelnik et al. (1995) proved that RVFLNs are universal approximators for continuous functions on bounded finite-dimensional sets.

A Reservoir Computer, RC

Is a kind of neural network - for forecasting dynamical systems
but most of the (millions of) parameters are chosen randomly.

Clearly its cheap

Surprisingly - it actually works!

And surprisingly, it works really well.

(This talk is not about neat things you can do with RC)
(This talk is about how/why/bridge-equivalent to something else)

Last year's talk….

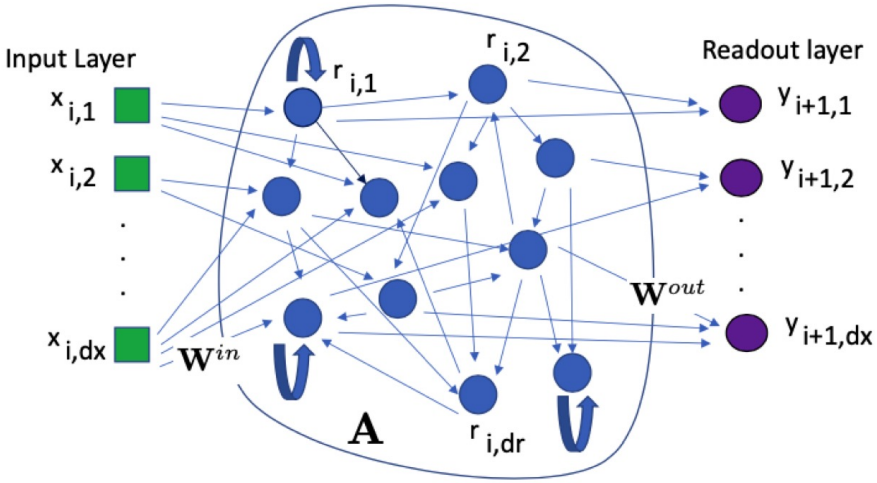
ESN-Jaeger 2001, Jaeger-Haas, 2004.

Reservoir computing – a special case of RNN
spec case ANN, Jaeger-Hass 2004, ESN-Jaeger 2001.

$$\{\mathbf{x}_i\}_{i=1}^{N} \subset \mathbb{R}^{d_x} \qquad d_r > d_x$$

$$\mathbf{u}_i = \mathbf{W}^{in}\mathbf{x}_i, \qquad \mathbf{r}_i \in \mathbb{R}^{d_r}$$

$$\begin{aligned} \mathbf{r}_{i+1} &= (1-\alpha)\mathbf{r}_i + \alpha q(\mathbf{A}\mathbf{r}_i + \mathbf{u}_i + \mathbf{b}), \\ \mathbf{y}_{i+1} &= \mathbf{W}^{out}\mathbf{r}_{i+1}. \end{aligned}$$



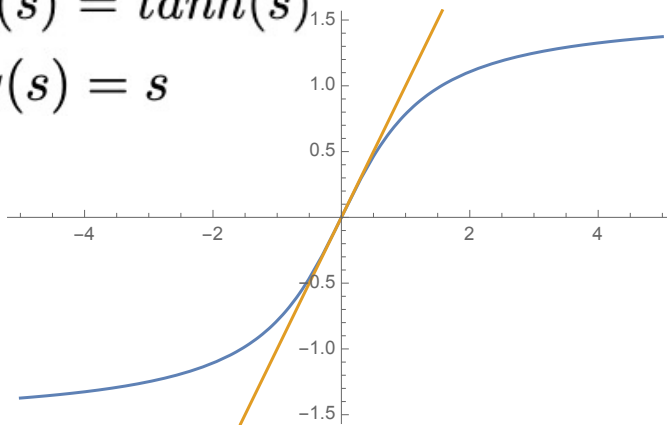
$\mathbf{W}^{in}_{i,j} \sim U(0,\gamma)$ $\quad d_r \times d_x$ read in matrix

$\mathbf{A}_{i,j} \sim U(-\beta,\beta)$, with β to scale the spectral radius

$d_x \times d_r$ *trained read-out matrix* matrix $\mathbf{W}^{out}$

$q(s) = tanh(s)$

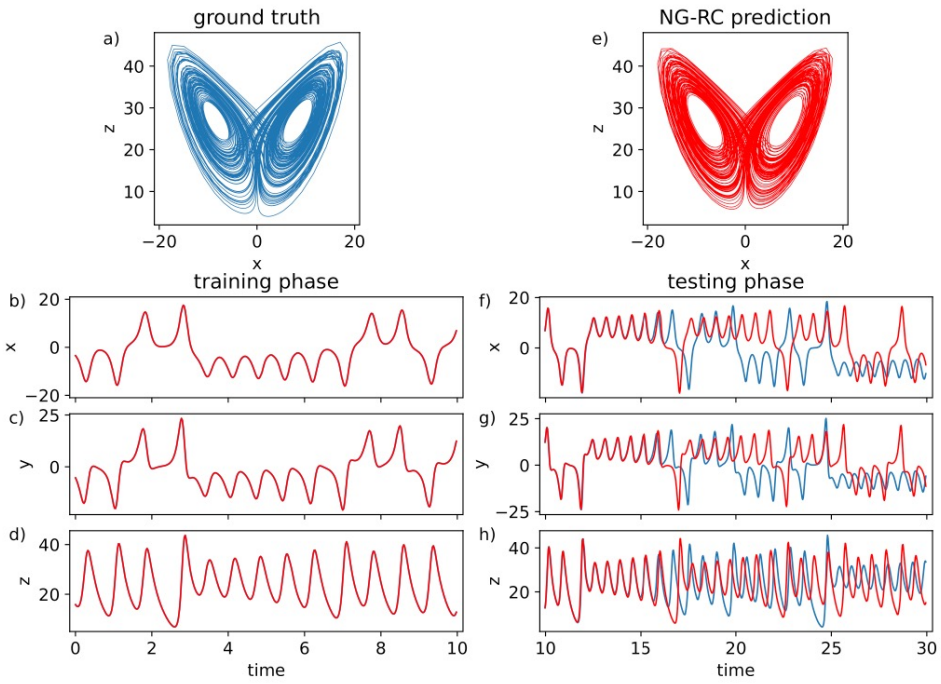
→ $q(s) = s$



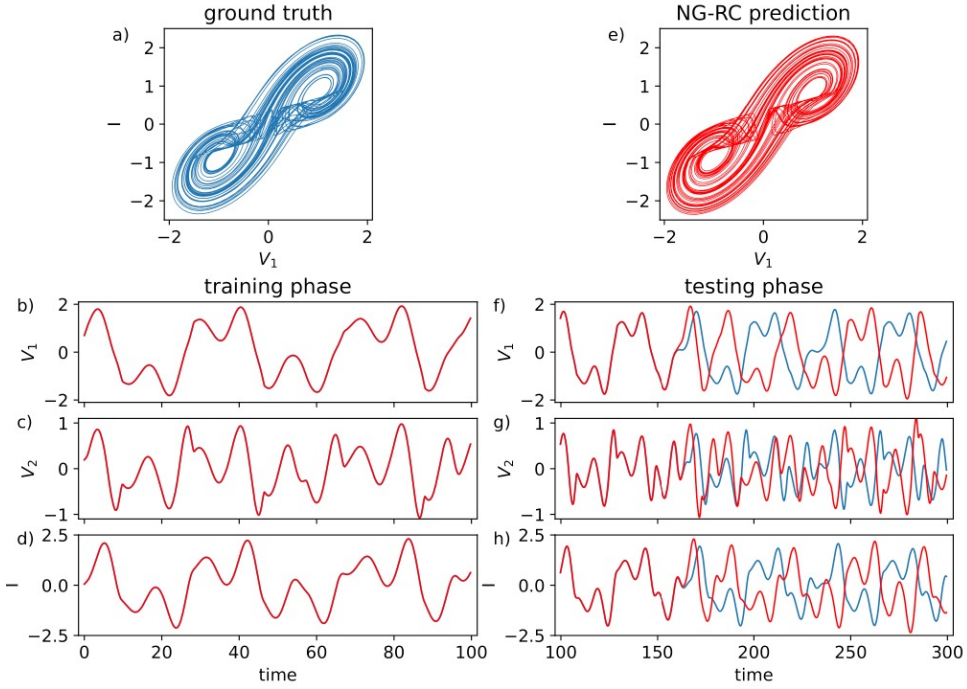
Surprise – A and $\mathbf{W}^{in}$ are random but it still works!

Notable Litt: **Gonon - Ortega 19', 20'** – RC enjoys a universal approximation theorem.
Even if linear with nonlinear readout.

NG-RC works very well, with very few points, almost no tunable parameters

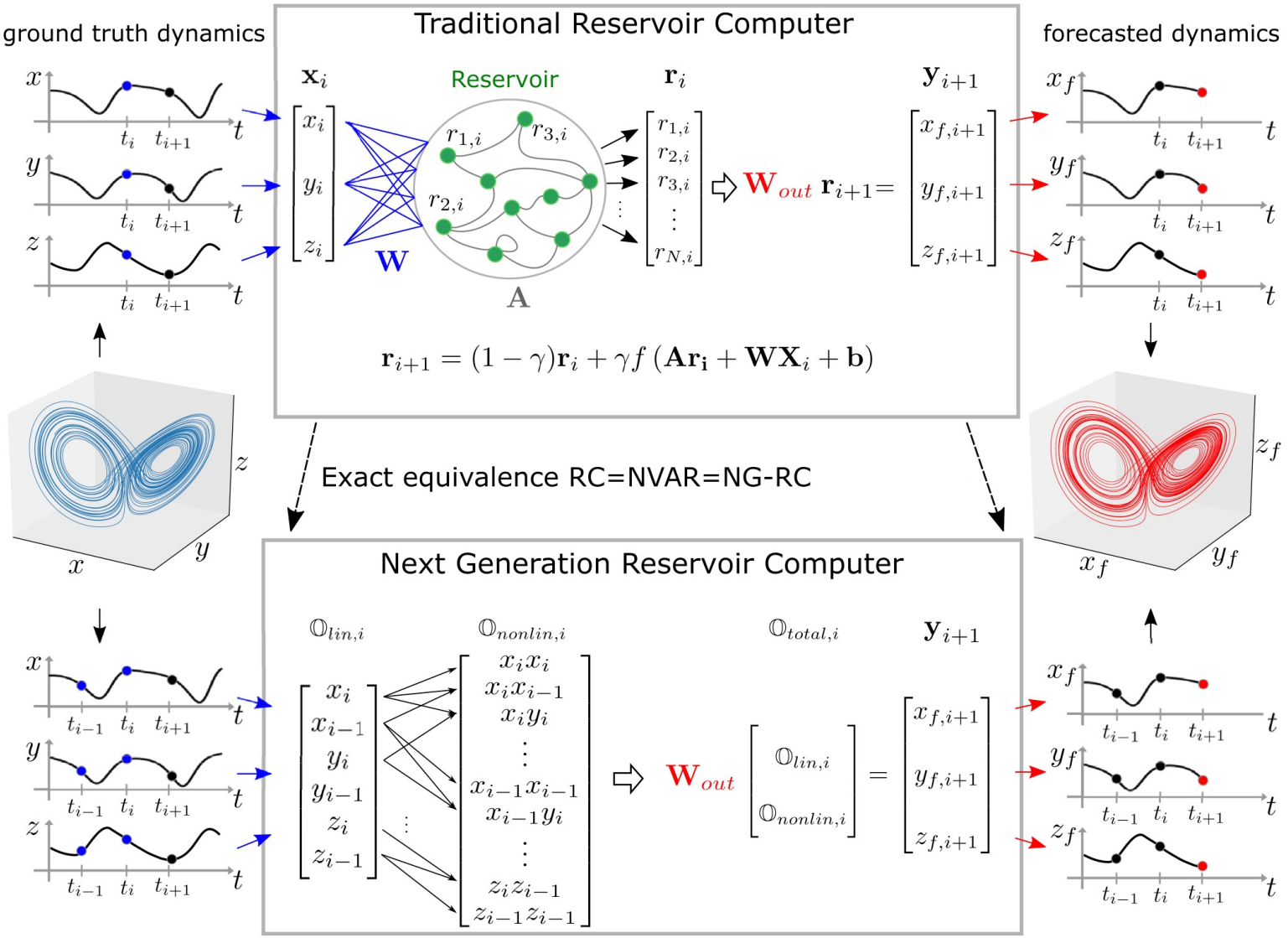


Forecasting a dynamical system using the NG-RC.
Lorenz63 strange attractors.

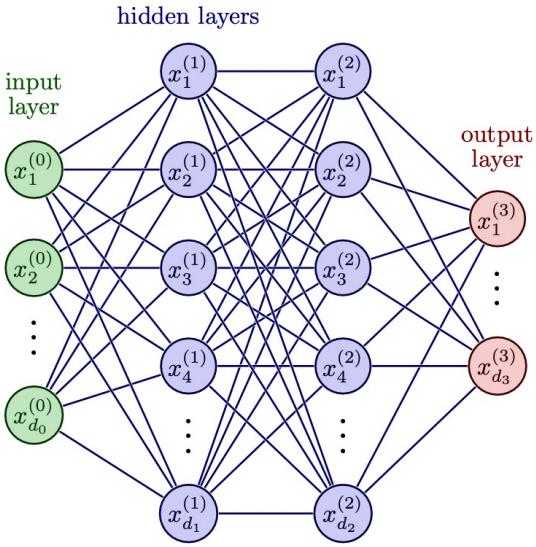


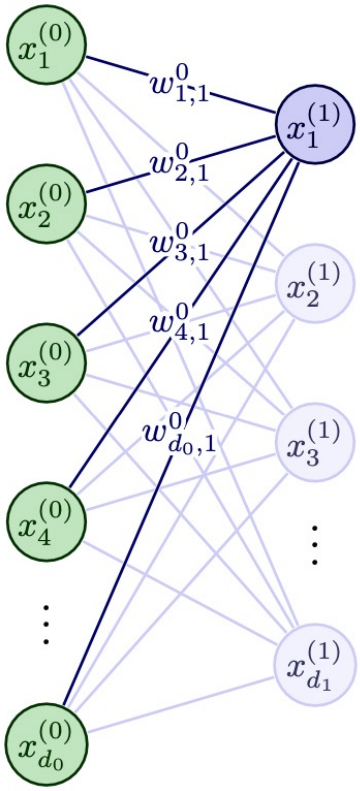
Forecasting the double-scroll system using the NG-RC

A traditional RC
is implicit
in an NG-RC



The "usual" FFNN (Feedforward Neural Network)– DNN (Deep Neural Network)
Full training vs ELM





$$= \sigma\left(w^0_{1,1}x^{(0)}_0 + w^0_{2,1}x^{(0)}_1 + \ldots + w^0_{d_0,1}x^{(0)}_n + b^1_1\right)$$

$$= \sigma\left(\sum_{i=1}^{d_0} w^0_{1,i}x^{(0)}_i + b^1_1\right)$$

$$\begin{pmatrix} x^{(1)}_1 \\ x^{(1)}_2 \\ \vdots \\ x^{(1)}_{d_1} \end{pmatrix} = \sigma\left[\begin{pmatrix} w^0_{1,1} & w^0_{2,1} & \cdots & w^0_{d_0,1} \\ w^0_{1,2} & w^0_{2,2} & \cdots & w^0_{d_0,2} \\ \vdots & \vdots & \ddots & \vdots \\ w^0_{1,d_1} & w^0_{2,d_1} & \cdots & w^0_{d_0,d_1} \end{pmatrix}\begin{pmatrix} x^{(0)}_1 \\ x^{(0)}_2 \\ \vdots \\ x^{(0)}_{d_0} \end{pmatrix} + \begin{pmatrix} b^1_1 \\ b^1_2 \\ \vdots \\ b^1_{d_1} \end{pmatrix}\right]$$

$$x^{(1)} = \sigma\left(\mathbf{W}^0 x^{(0)} + \mathbf{b}^1\right)$$

$$\mathcal{L}(\mathcal{D};\Theta) = \sum_{i=1}^{N} \|F(X_i,\Theta) - Y_i\|_2 + \lambda\mathcal{R}(\Theta). \qquad F(X,\Theta) = F_{q,\Theta_q} \circ F_{q-1,\Theta_{q-1}} \circ \ldots \circ F_{0,\Theta_0}(X),$$

$$\Theta^*_{\text{ANN-}q} = \underset{\mathcal{D};\Theta}{\operatorname{argmin}}\ \mathcal{L}(\mathcal{D};\Theta), \quad \textbf{vs} \quad \Theta^*_{\text{ELM-}q} = \underset{\mathcal{D};\Theta_q}{\operatorname{argmin}}\ \mathcal{L}(\mathcal{D};\Theta_0 \cup \Theta_1).$$

ELM-1 it's a random SLFN

– random parameters except output layer.
- Nonlinear threshold function to hidden layers
- Identify function threshold function output

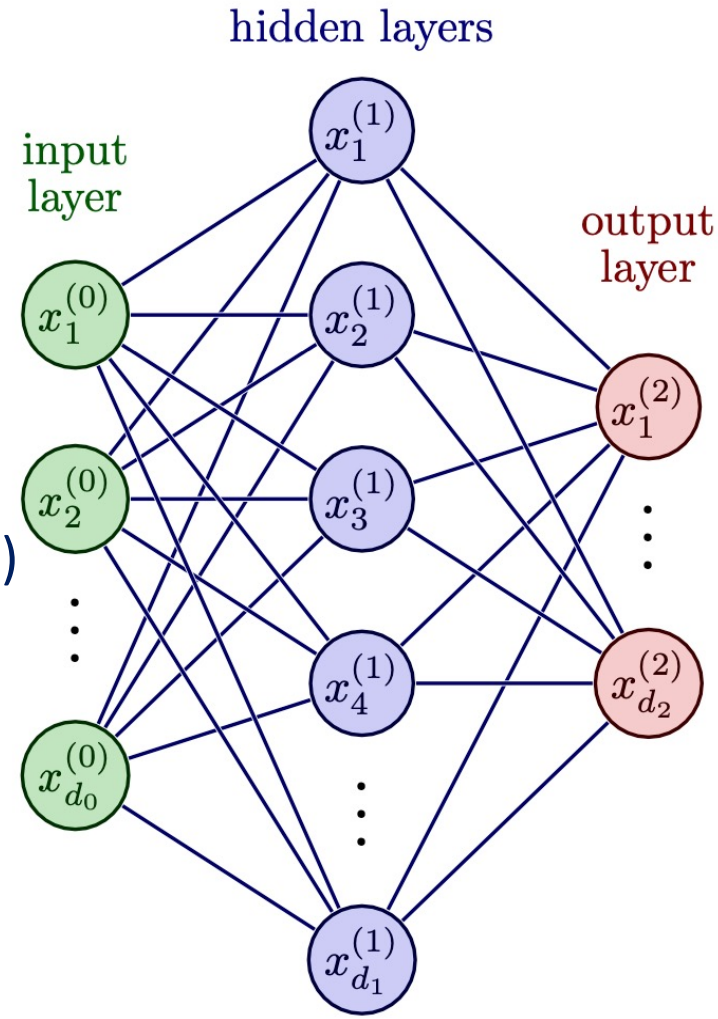
TRAIN only output layer,
Done by simple (linear – OLS regression/regularized)

$$f : \mathbb{R}^{d_0} \to \mathbb{R}^{d_2}$$

regression of a

$$\mathcal{D} = \{(X_i, Y_i)\}_{i=1}^{N} \subset \mathbb{R}^{d_0} \times \mathbb{R}^{d_2} = D$$

Based on observed data



Counting Parameters- Full Training.

Vs SLFN

q hidden layers,

$$\Theta = \cup_{r=0}^{q} \Theta_r.$$

$$\Theta_r = \{\{w_{j,k}^r\}_{j=1,k=1}^{d_r,d_{r+1}}, \{b_k^{r+1}\}_{k=1}^{d_{r+1}}\}.$$

$$|\Theta| = d_0 d_1 + d_1 d_2 + d_2 d_3 + ... + d_q d_{q+1} + d_q = d_q + \prod_{i=0}^{q} d_i d_i.$$

$$[W^r]_{j,k} = w_{j,k}^r, \text{ and, } [B^{r+1}]_k = b_k^{r+1}, j = 1, ..., d_r, k = 1, .., d_{r+1},$$

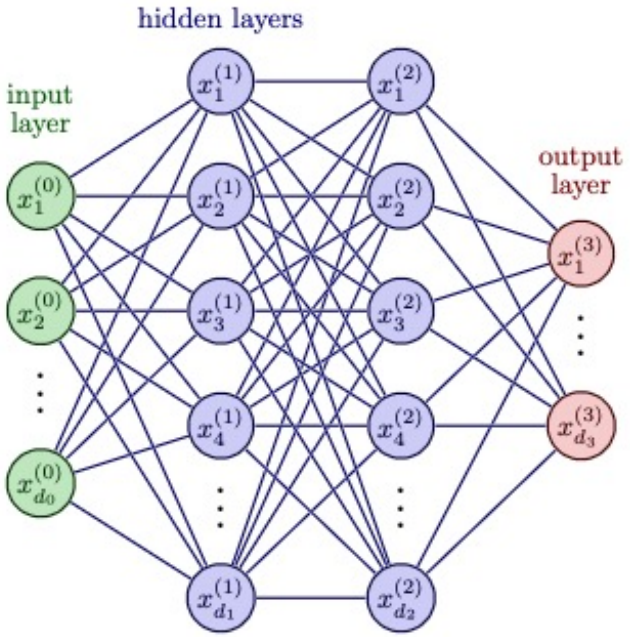
$$F_{r,\Theta_r}(X^r) = \sigma_r(W^r X^r + B^{r+1})$$

Composition

$$F(X, \Theta) = F_{q,\Theta_q} \circ F_{q-1,\Theta_{q-1}} \circ ... \circ F_{0,\Theta_0}(X)$$

$$|\Theta| = d_0 d_1 + d_1 d_2 + d_1 + d_2,$$

$$|\Theta| = d_0 d_1 + d_1 d_2 + d_1$$



With ReLu Activation Something Simple Happens –-configuration of fallen lines

$$\sigma_r(s) = ReLu(s) = max(s, 0), r < q, \sigma_q(s) = s$$

The Joys of not training

SLFN vs ELM-1

$$|\Theta_{out}| = |\Theta_1| = d_1 d_2,$$

$$|\Theta_{out}| = d_1 d_2 << |\Theta| = d_0 d_1 + d_1 d_2 + d_1 + d_2,$$

And for DFN vs ELM-q, q>1

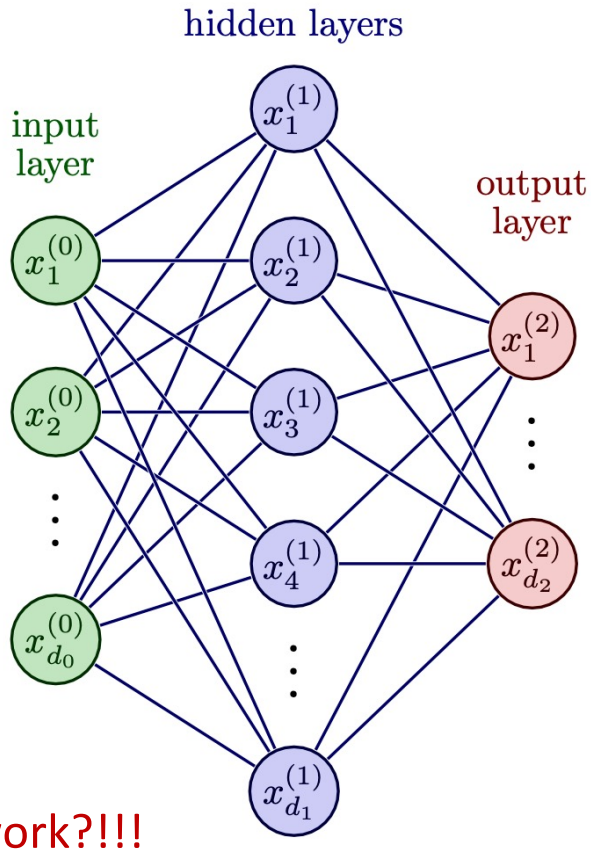
$$|\Theta_{out}| = d_q d_{q+1} << |\Theta| = d_{q+1} + \prod_{i=0}^{q} d_i d_{i+1}.$$

Less parameters to train - training easy - OLS - or - Ridge-T

$$\mathbf{X} = [X_1^q | X_2^q | ... | X_N^q], \mathbf{Y} = [Y_1 | Y_2 | ... \quad \mathbf{Y} = \beta \mathbf{X}$$

$$W^q := \underset{\beta}{\operatorname{argmin}} \|\mathbf{Y} - \beta \mathbf{X}\|_F + \lambda \|\beta\|_F,$$

$$\beta^* = \mathbf{Y}\mathbf{X}^T(\mathbf{X}\mathbf{X}^T + \lambda I)^{-1} := \mathbf{Y}\mathbf{X}_\lambda^\dagger.$$

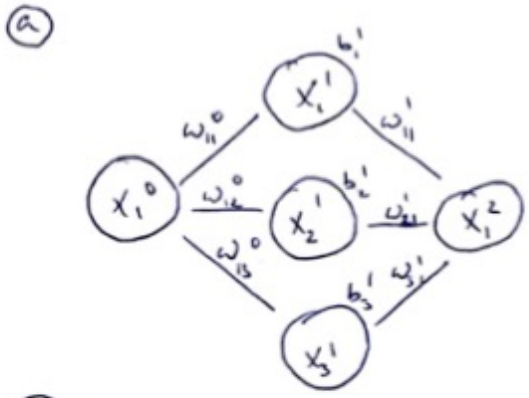


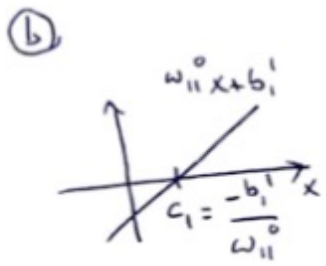
It's Cheap! - But does it work?!!!

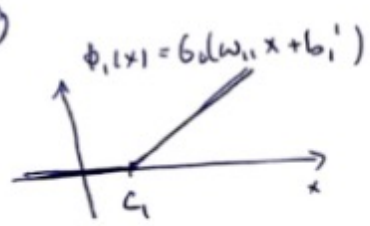
Story of configurations of randomly fallen lines in plane
And randomly fallen hyperplanes in space.

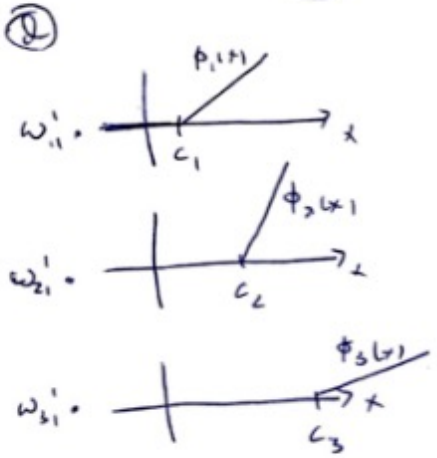
ELM-1 - single layer FNN – SLFN

- here simplified to a really small, one input dimension, one output dimension

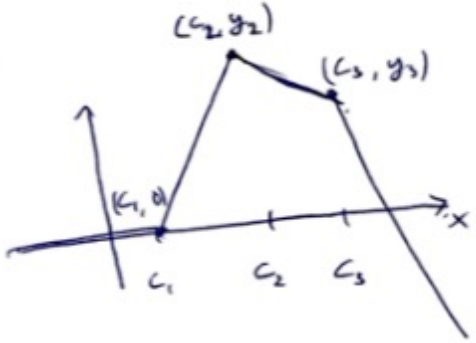








=



Untrained - Random Parameters.

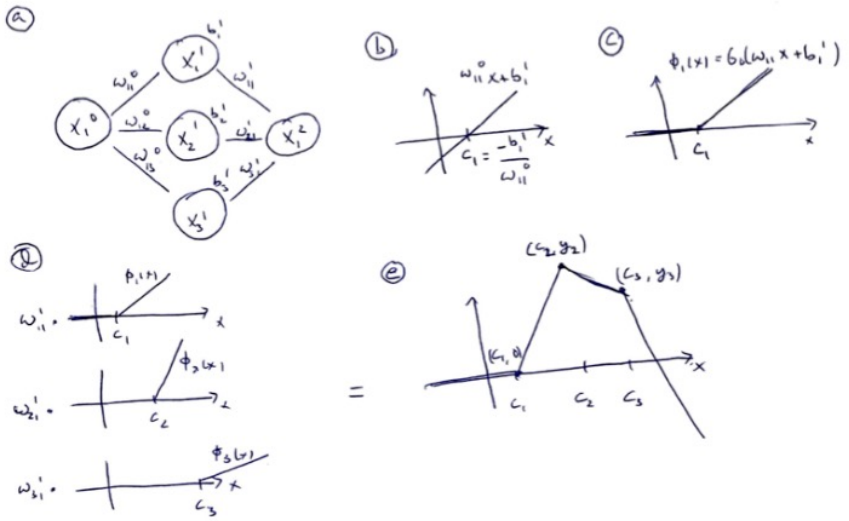
$$w_{1,i} \sim U([0,1]), b_i^{(1)} \sim U([-1,0]),$$

x-intercept

$$c_i^{(0)} = \frac{-b_i^{(1)}}{w_{1,i}^{(0)}},$$

ELM-1 - single layer FNN – SLFN

- here simplified to a really small, one input dimension, one output dimension



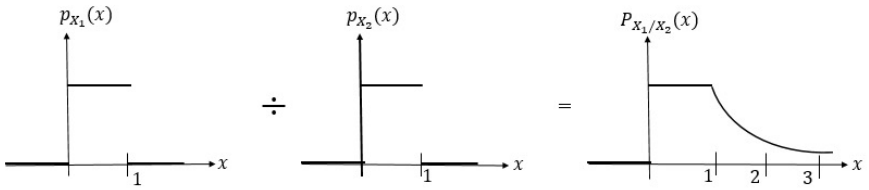
Untrained - Random Parameters.

$$w_{1,i} \sim U([0,1]), b_i^{(1)} \sim U([-1,0]),$$

x-intercept

$$c_i^{(0)} = \frac{-b_i^{(1)}}{w_{1,i}^{(0)}},$$

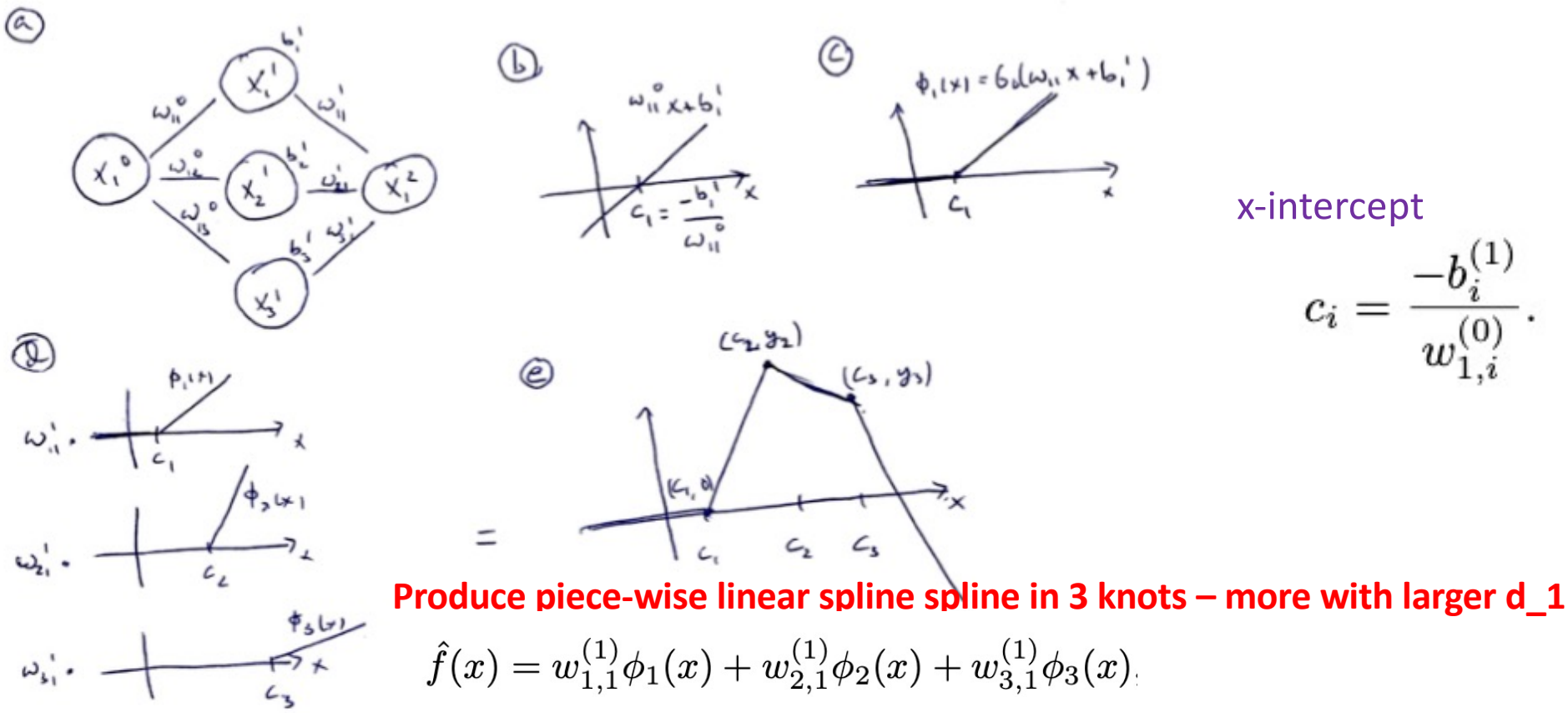
Quotient distribution



$$c_i \sim \frac{(H(1-x)) - H(x -}{2}, \text{ where } H(x) = \text{n}(0, x < 0, \frac{}{2}, x = 0, 1, x > 0),$$

ELM-1 - single layer FNN – SLFN

- here simplified to a really small, one input dimension, one output dimension



x-intercept

$$c_i = \frac{-b_i^{(1)}}{w_{1,i}^{(0)}}.$$

Produce piece-wise linear spline spline in 3 knots – more with larger d_1

$$\hat{f}(x) = w_{1,1}^{(1)}\phi_1(x) + w_{2,1}^{(1)}\phi_2(x) + w_{3,1}^{(1)}\phi_3(x)$$

$$\hat{f} \in \Delta_{d_1} = span(\{\phi_(x), \phi_2(x), \phi_3(x)\})$$

Universal Approximation Theorem ELM as a picture

Given lots of data,

$$\mathcal{D} = \{(X_i, Y_i)\}_{i=1}^{N} \subset \mathbb{R}^{d_0} \times \mathbb{R}^{d_2} = D$$

With a large hidden layer of the SLFN

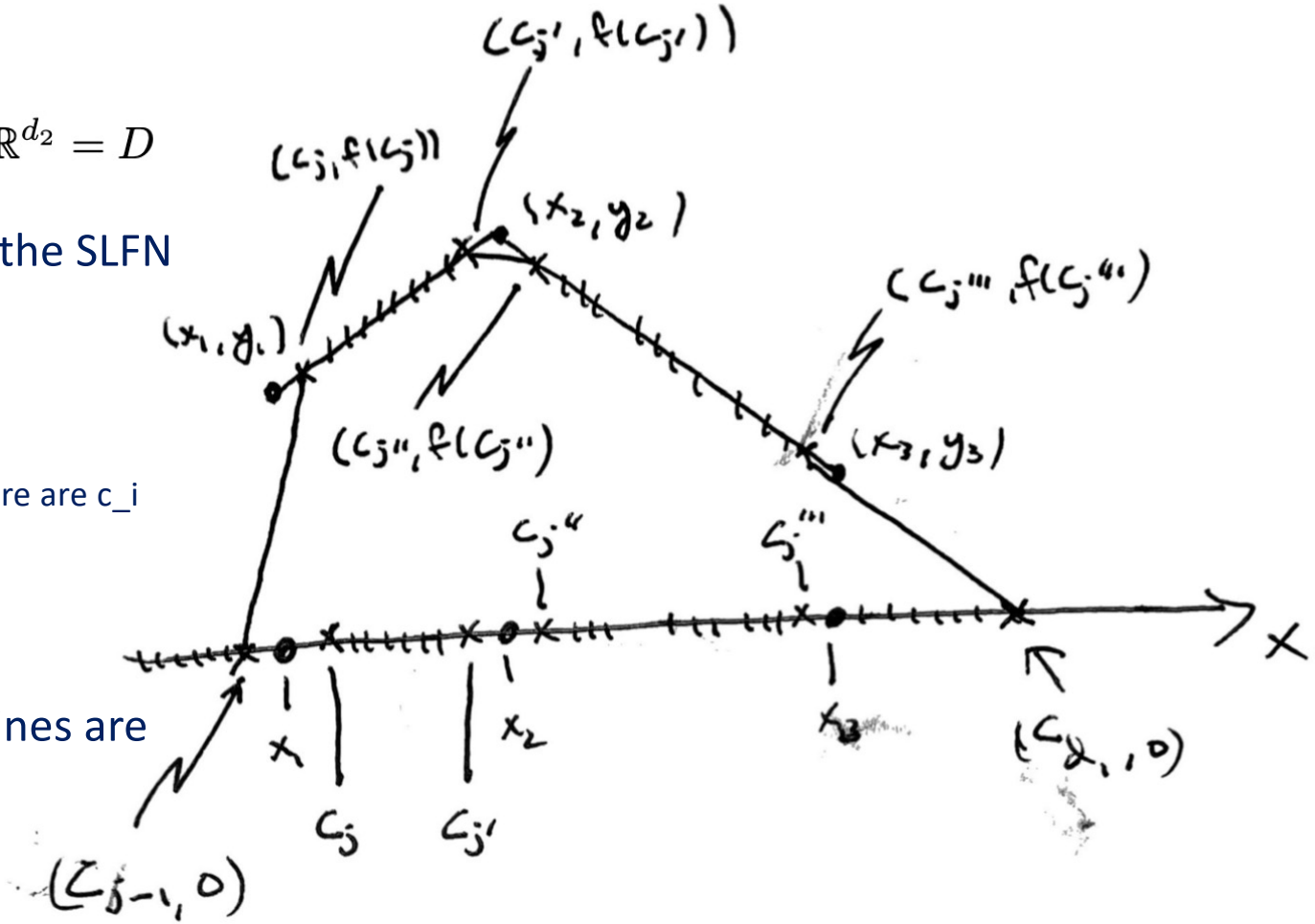
My take:

-random hidden layer

(the c_i are placed randomly)

between data points x_i there are c_i

-a linear spline can run through data.

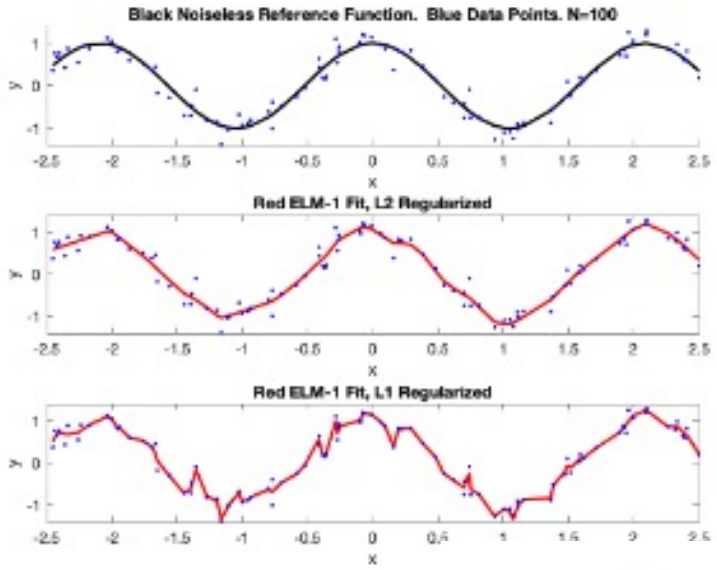


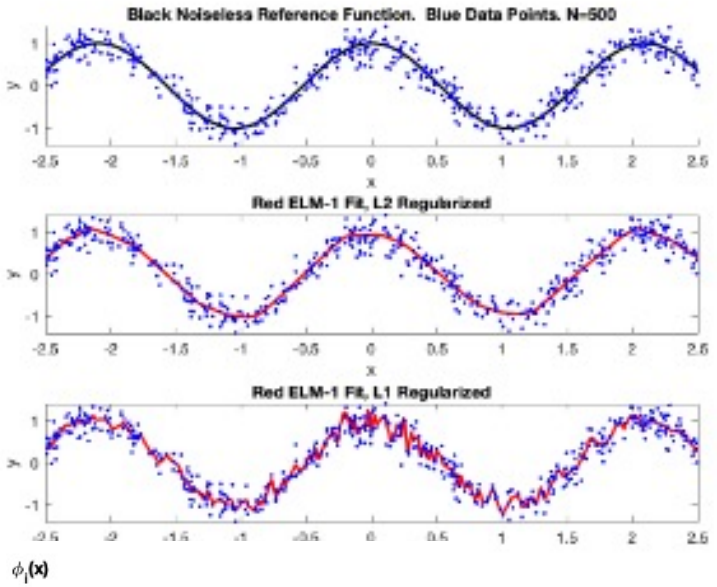
General piecewise linear splines are Dense in C^0.

Vs Stone-Weierstrass

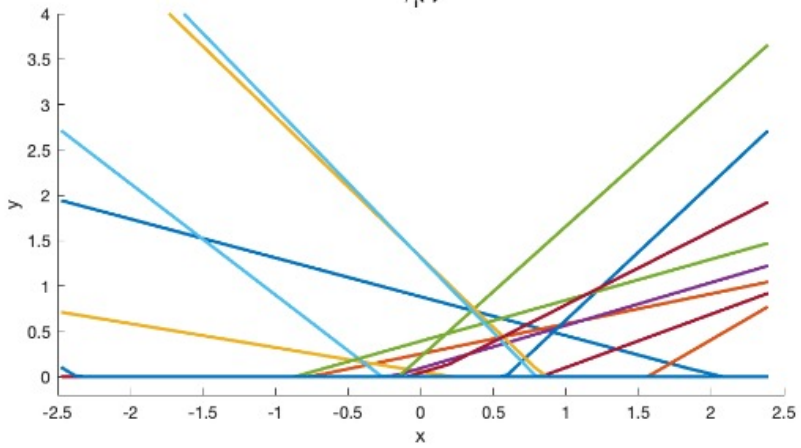
Cite Huang, ELM as universal approximation theorem.

A One-D ELM-1 experiment – with noise – and increasing data set size

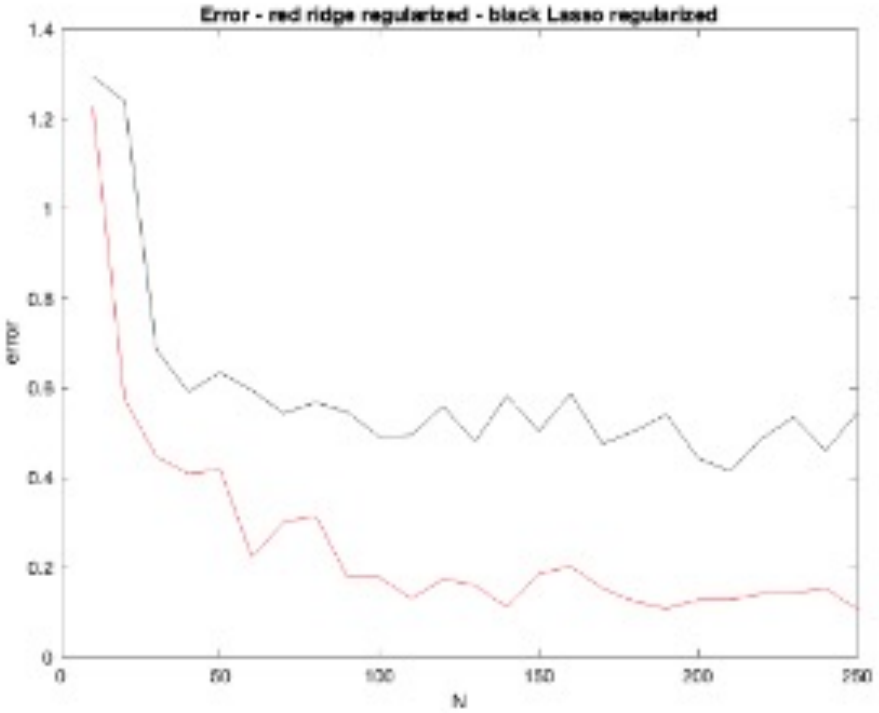


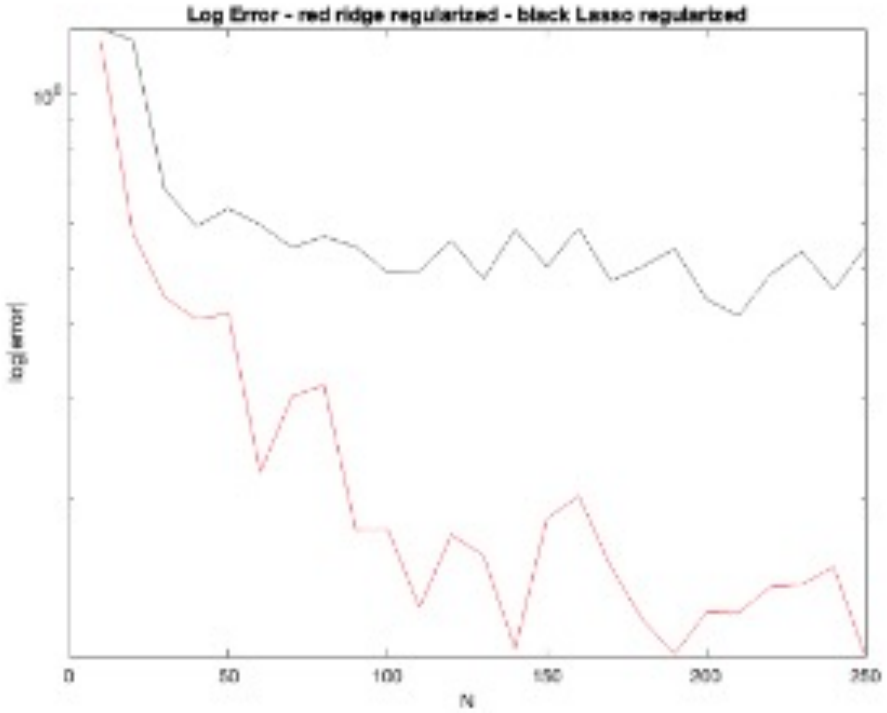


Some of the phi_i(x)



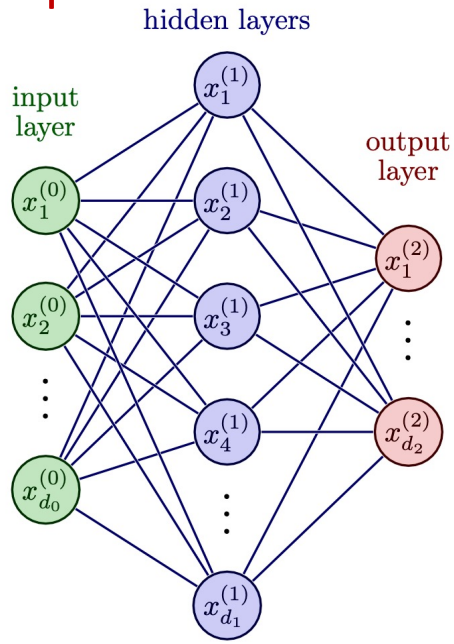
A One-D ELM-1 experiment – with noise – and increasing data set size





ELM-1 - single layer FNN – SLFN

- here simplified to a really small, TWO input dimension, one output dimension



Consider the i^{th} neuron in the hidden layer, but before the threshold is applied,

$$z = W_{i,:}^0 \cdot \mathbf{X}^{(0)} + b_i^{(1)}, i = 1, 2, ..., d_1.$$

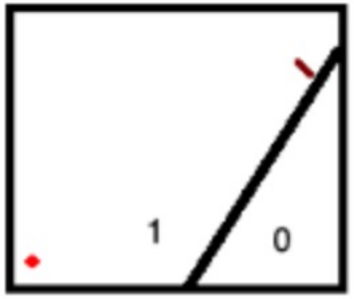
!!equation of a line in 2-d! Or a plane in 3-d, or a hyperplane in n-d.

Still ReLu activation function:

$$\sigma_r(s) = ReLu(s) = max(s, 0), r < q, \sigma_q(s) = s$$

half plane ReLu does nothing, and for other half plane, you get zero.

$$z \geq W_{i,:}^0 \cdot \mathbf{X}^{(0)} + b_i^{(1)}.$$



ELM-1 - single layer FNN – SLFN

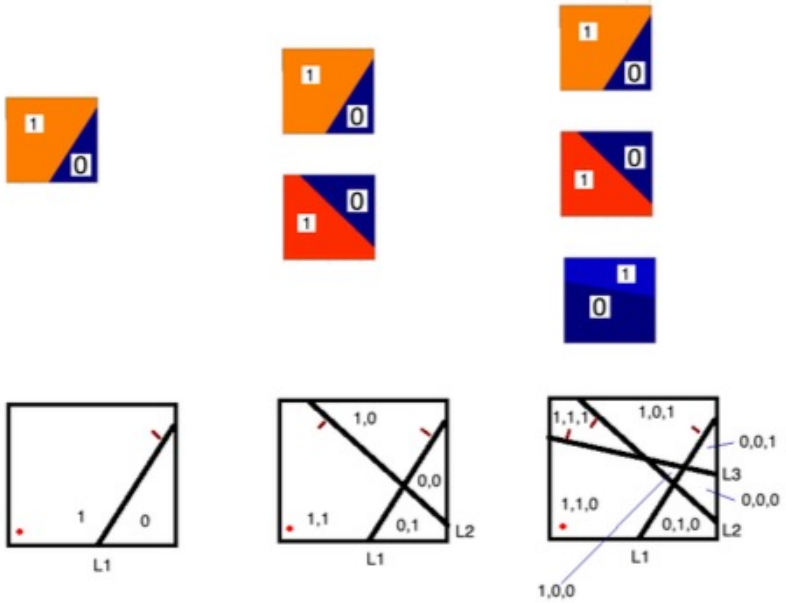
- here simplified to a really small, one input dimension, one output dimension

Consider the i^{th} neuron in the hidden layer, but before the threshold is applied,

$$z = W^0_{i,:} \cdot \mathbf{X}^{(0)} + b^{(1)}_i, i = 1, 2, ..., d_1.$$

For half plane ReLu, you get zero.

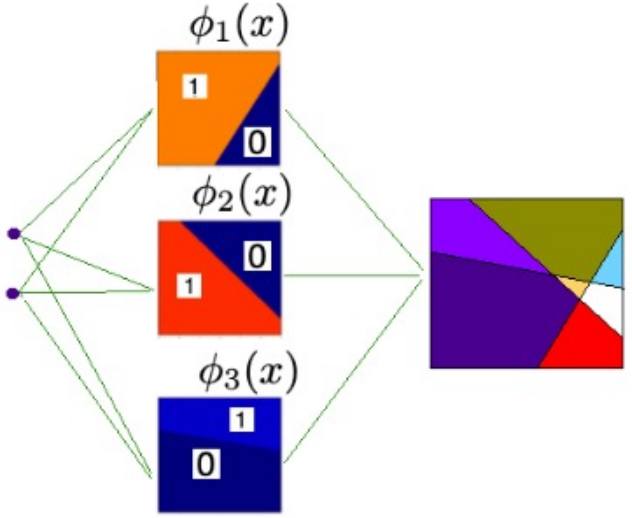
$$z \geq W^0_{i,:} \cdot \mathbf{X}^{(0)} + b^{(1)}_i.$$



a)

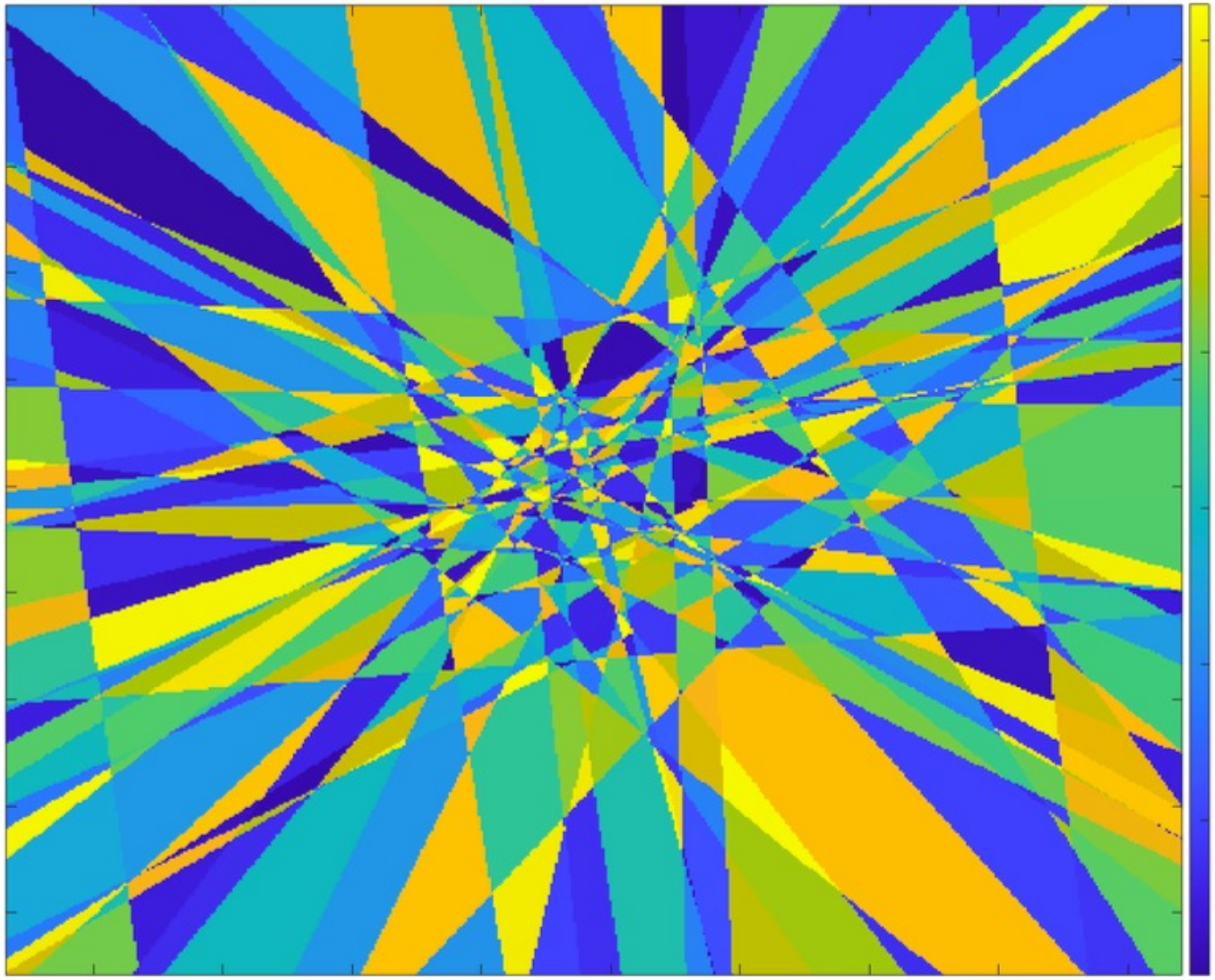
Notice Symbology

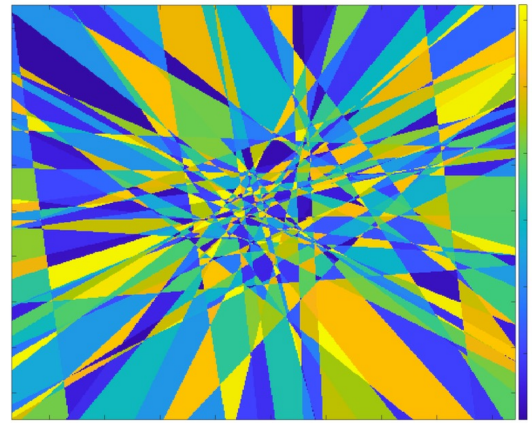
Still end up with
Linear combinations of phi_i(x)



b)

ELM-1 in
2-input dimensions
Lots of hidden layer nodes

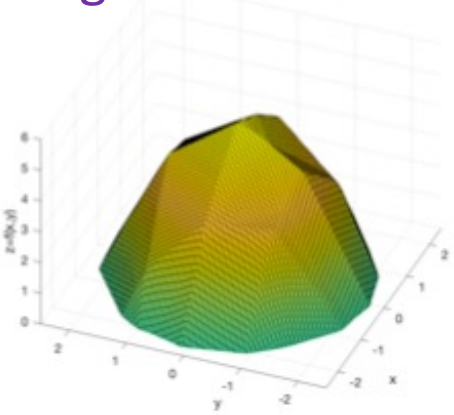
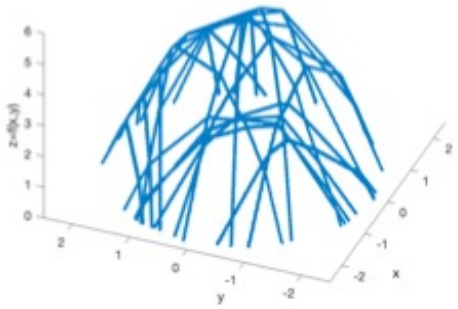
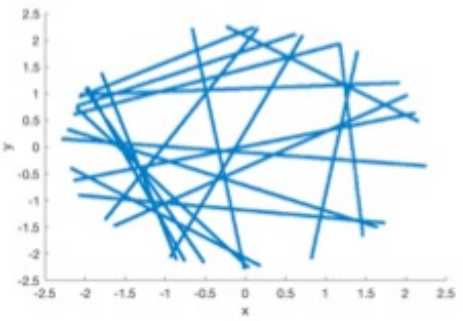
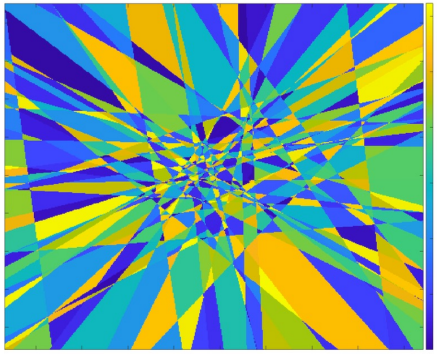
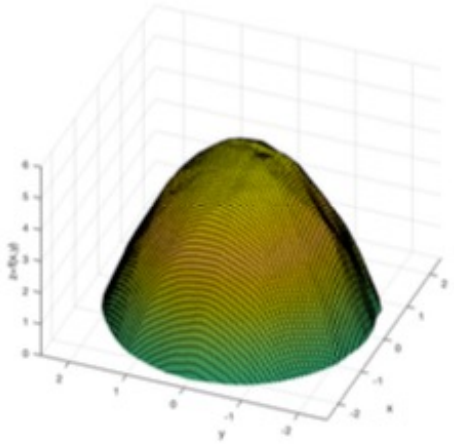
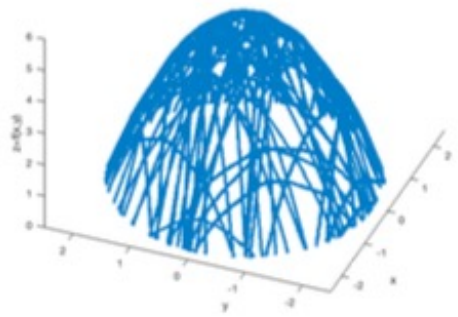
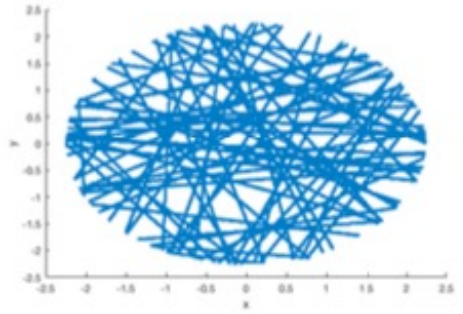




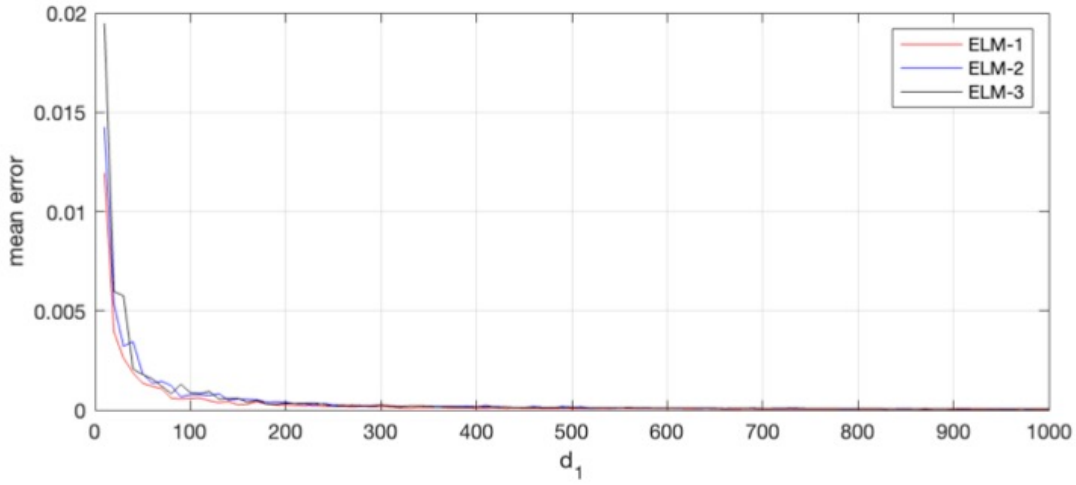
Church Windows!

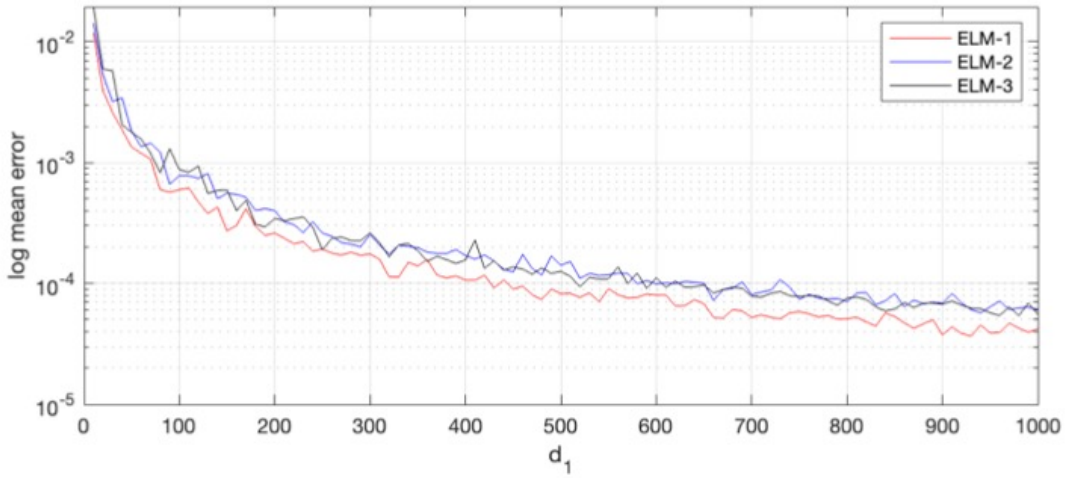
Neural Nets in
The Middle Ages!

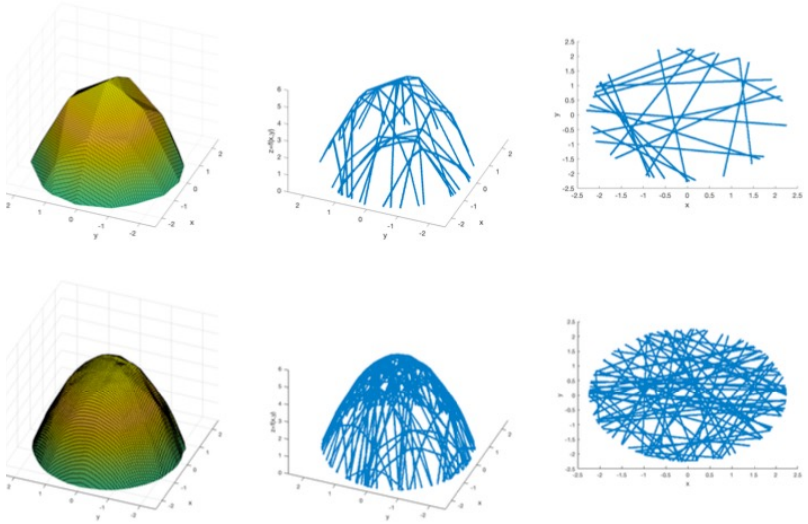
Fitting a function with an ELM-1 and increasing hidden layer size

Larger d_1 means more randomly placed lines, - finer random partition => more precision of piecewise linear cts fn.







Story summary –

ELM-1 – bigger hidden layer,

-more random lines

-finer fitting of piecewise linear fn

-piecewise linear dense in C^0

ELM-q – q>1

-even though for fully trained – deep

Is good. More expressive

-for ELM not any better

Thank you!

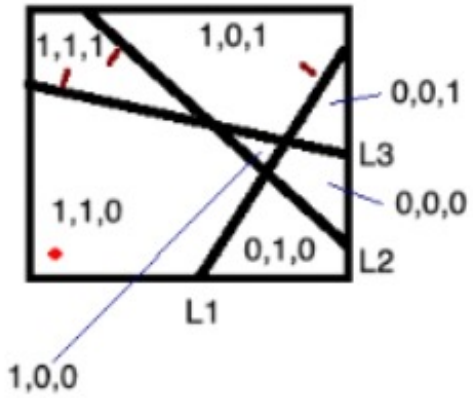
Theory of Random Configurations of Lines in the Plane

Of (hyper)Planes in Space.

$$M_n \leq \frac{n(n+1)}{2} + 1 = \#\text{ of cells in an arrangement of } n\text{-lines in 2-dimensional space.}$$

Notice:

$$M_1 = 2, M_2 = 4, \text{ but } M_3 = 7.$$



Fewer than you might guess – due to occlusion

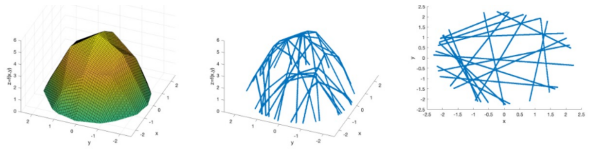
$$\frac{n(n+1)}{2} + 1 << 2^n.$$

011 is missing.

$$M_{m+1} \leq M_m + (m+1), M_1 = 2.$$

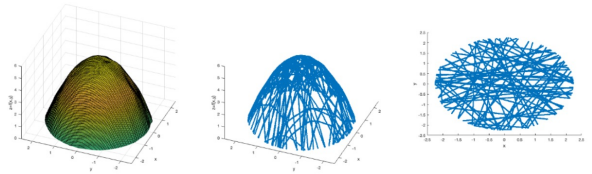
Theory of Random Configurations of Lines in the Plane Of (hyper)Planes in Space.

$$M(d_0, n) \leq \sum_{i=0}^{d_0} \binom{n}{i} = \#\text{ of cells in an arrangement of } n\text{-planes (hyperplanes) in } d_0\text{-dimensional space.}$$



recursion relationship. This time,

$$M(d_0, n) \leq M(d_0, n-1) + M(d_0, n-1).$$

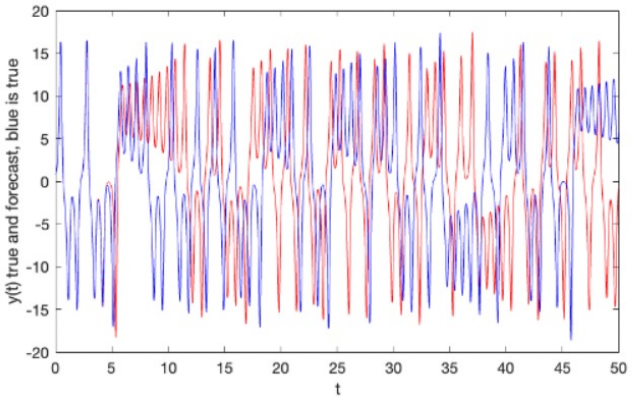


$$B(d_0, n) \leq \binom{n-1}{d_0} = \#\text{ of bounded cells in an arrangement of } n\text{-hyperplanes in } d_0\text{-dimensional space,}$$

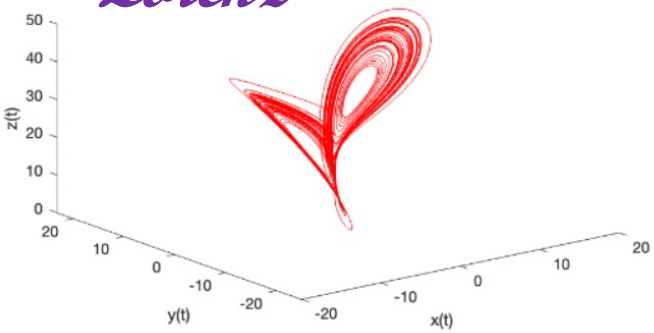
$$B(d_0, n) \leq \binom{n-1}{d_0} = \#\text{ of bounded cells in an arrangement of } n\text{-hyperplanes in } d_0\text{-dimensional space,}$$

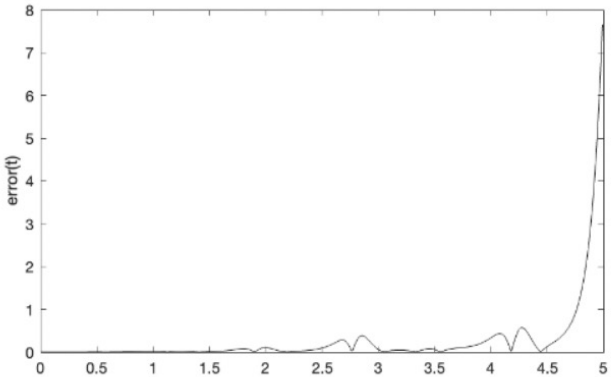
- As the size of the hidden layer increases, $d_1 \uparrow$, then the number of linear domains increases.
- As $d_1 \uparrow$, the size of these domains decreases. This is a random refinement.
- As $d_1 \uparrow$, the number of basis elements in Δ_{d_1} increases, and the fit accuracy improves.

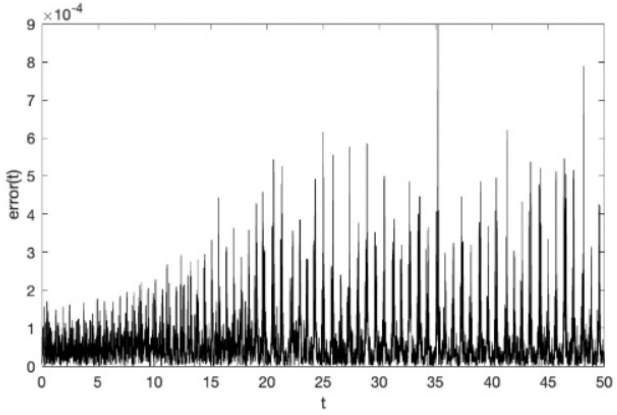
ELM for Forecasting Chaos



Lorenz







Differential Equation

$$\dot{z} = f(z), \text{ for a given initial condition in its phase space } z(t_0) = z_0 \in \mathcal{M} \subset \mathbb{R}^{d_0},$$

Flow – FUNCTION

$$\begin{aligned} \varphi : \mathcal{M} \times \mathbf{R} &\rightarrow \mathcal{M} \\ (t, z_0) &\mapsto z(t) = \varphi(t; z_0). \end{aligned}$$

Orbit DATA

$$\mathcal{D} = \{(z_k, z_{k+1}\}_{k=1}^{N}.$$