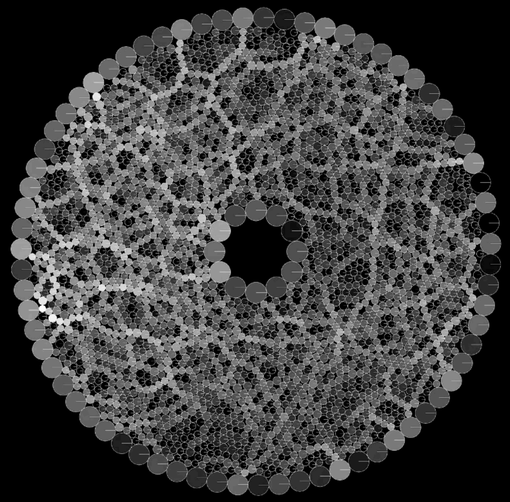
A Brief Intro to Deep Learning

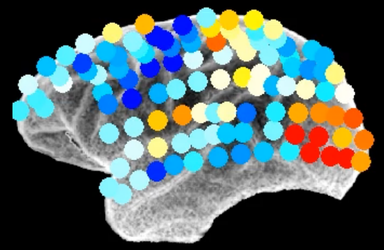
Joseph Bakarji
Olivia Thomas
Jan William

Complex systems are high dimensional



Granular Material (DEM Simulation)

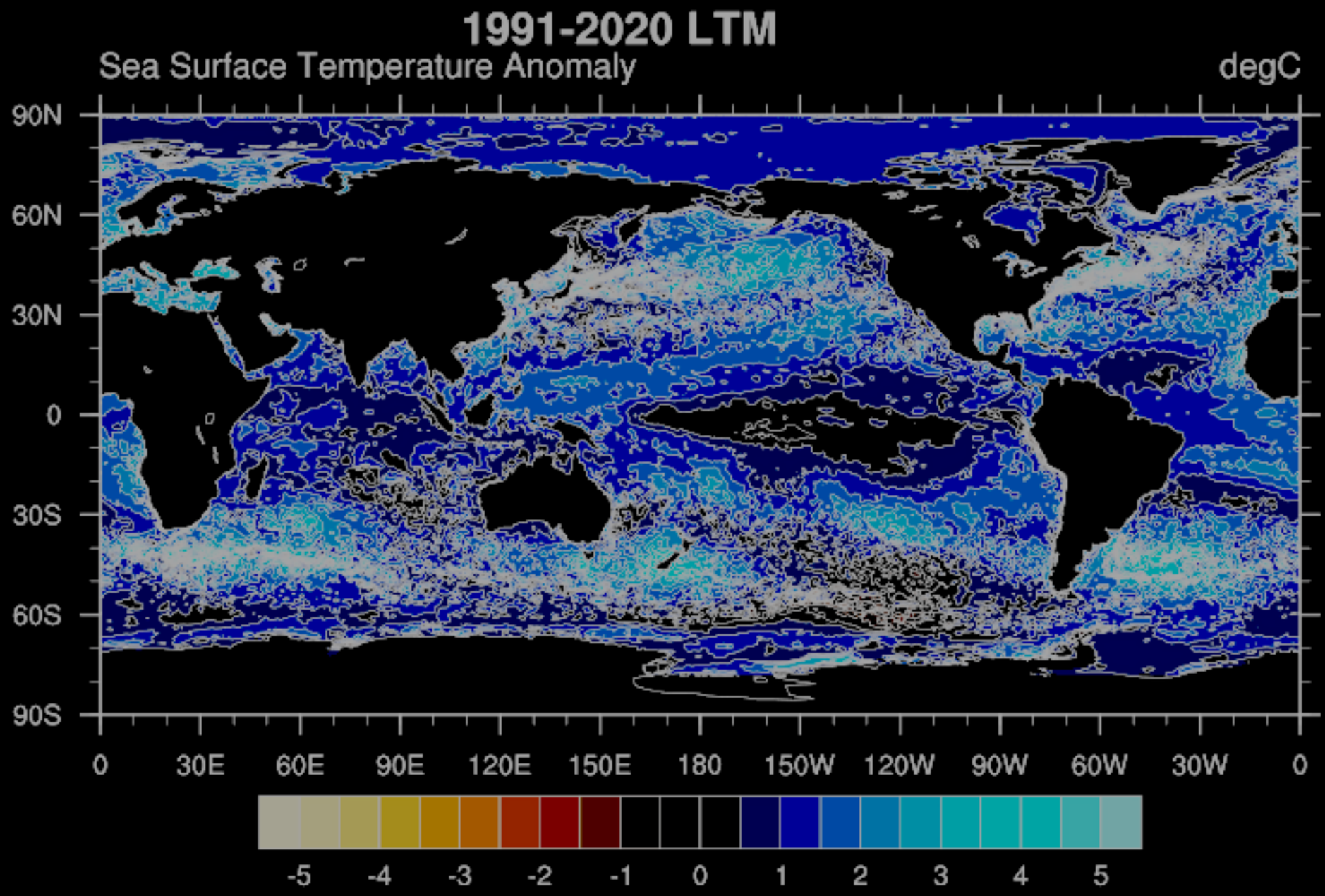
lasefist
https://youtu.be/uI-KrtV0PJA



Raut, Ryan V., et al. Science advances (2021)

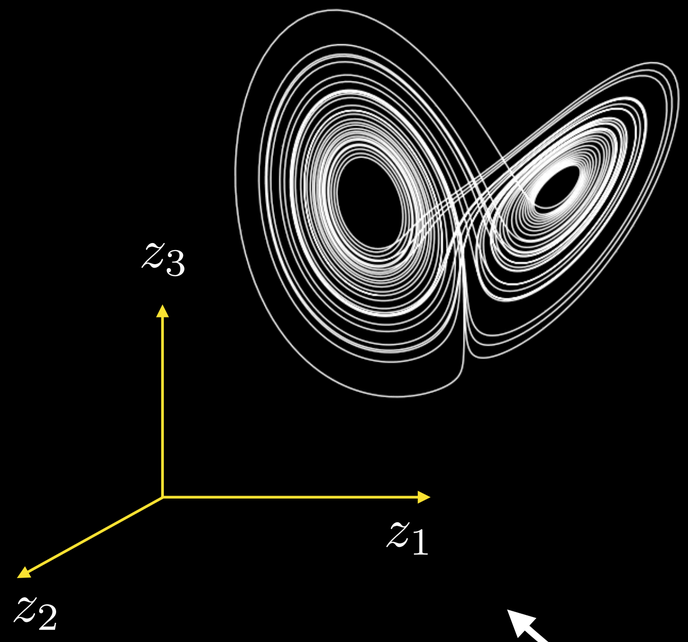
National Geographic
https://youtu.be/RtUQ_pz5wlo

We only have access to partial measurements

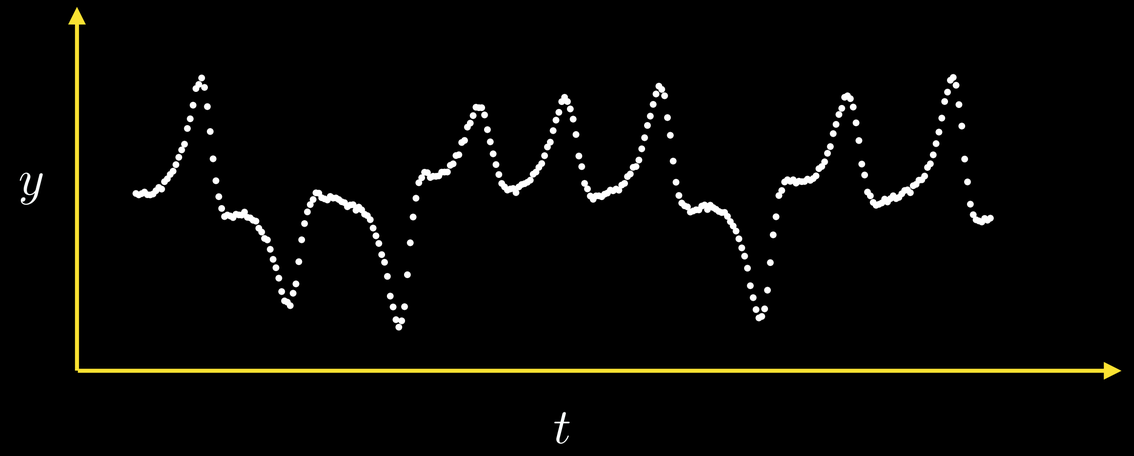

1991-2020 LTM
Sea Surface Temperature Anomaly
degC
90N
60N
30N
0
30S
60S
90S
0
30E
60E
90E
120E
150E
180
150W
120W
90W
60W
30W
0
-5
-4
-3
-2
-1
0
1
2
3
4
5

High Dimensional System

Partial Measurements



Measure

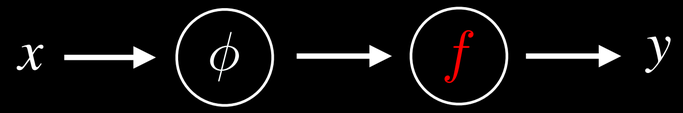


Reconstruct

?

Linear predictors

(x_1, y_1)
(x_2, y_2)
(x_3, y_3)
(x_4, y_4)
$\vdots$
(x_n, y_n)



feature vector

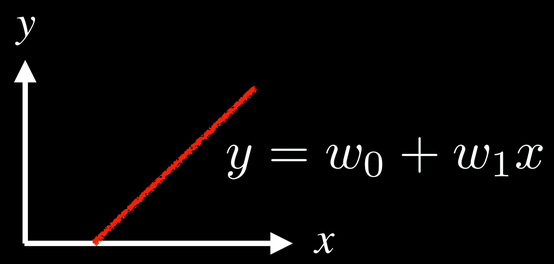
$$\phi(x) = [\phi_1(x),\ \phi_2(x),\ \ldots,\ \phi_d(x)]$$

linear predictor

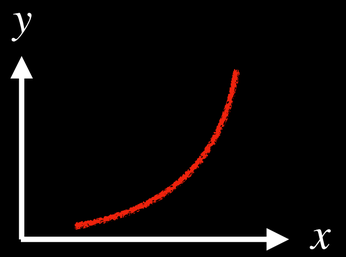
$$\begin{aligned} f_{\mathbf{w}} &= \mathbf{w} \cdot \phi(x) \\ &= w_1\phi_1(x) + w_2\phi_2(x) + \cdots + w_d\phi_d(x) \end{aligned}$$

What is linear?

Adding a bias: $\phi(x) = [1,\ x]$

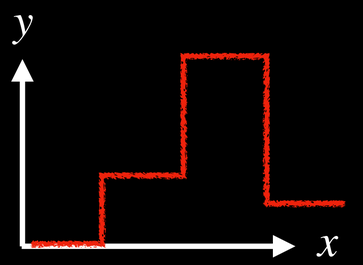


A polynomial predictor: $\phi(x) = \left[1, x, x^2, x^3\right]$



Sines and cosines: $\phi(x) = [1,\ x,\ \sin(3x)]$

Indicator functions: $\phi(x) = [\mathbf{1}[0 < x \leq 1],\ \mathbf{1}[1 < x \leq 2],\ \mathbf{1}[2 < x \leq 3]]$



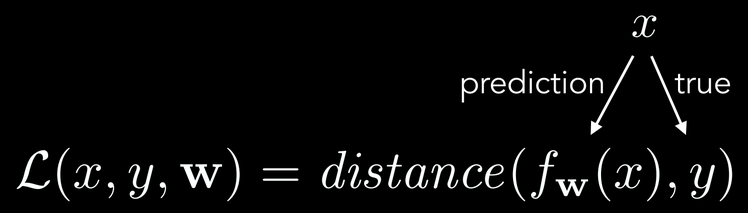
Derivatives: $\phi(x, t) = \left[x^2,\ \frac{\Delta x}{\Delta t}\right]$

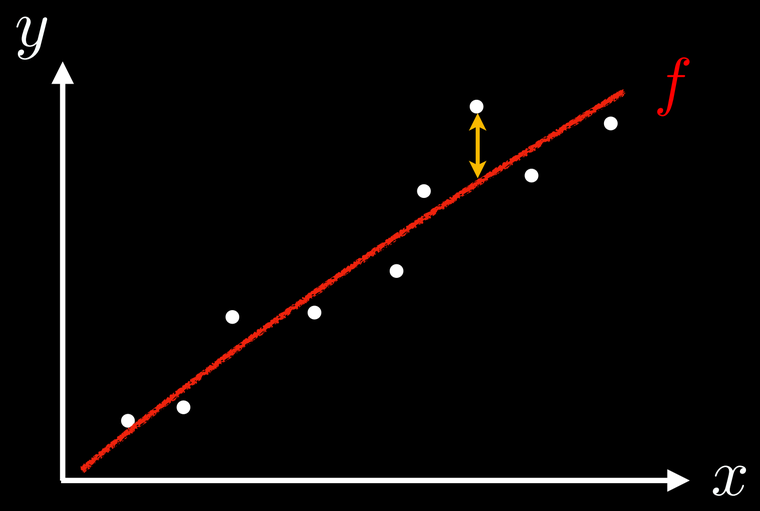
Linear in $\phi(x)$ and $\mathbf{w}$ but not in x

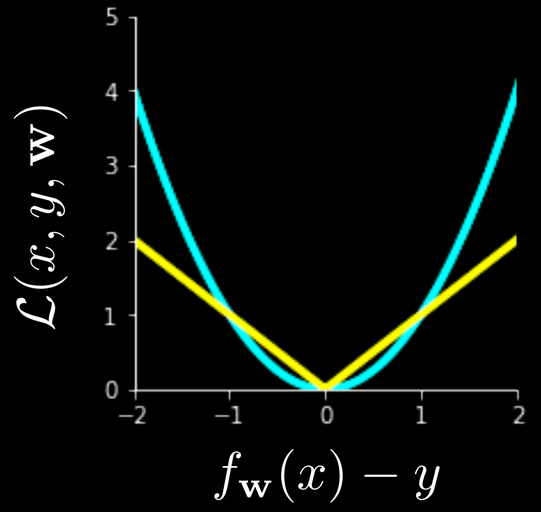
Brunton, S. L., Joshua L. P., and J. N. Kutz. "Discovering governing equations from data by sparse identification of nonlinear dynamical systems."

The loss function

A **loss function** quantifies how *bad* the predictor is







$$\mathcal{L}_{\text{sq}}(x, y, \mathbf{w}) = (f_{\mathbf{w}}(x) - y)^2$$

$$\mathcal{L}_{\text{abs}}(x, y, \mathbf{w}) = |f_{\mathbf{w}}(x) - y|$$

Minimize the loss

The **training loss** is the average loss over the data set

$$\mathcal{L}_{\text{train}}(\mathbf{w}) = \frac{1}{|\mathcal{D}_{\text{train}}|} \sum_{(x,y) \in \mathcal{D}_{\text{train}}} \mathcal{L}(x, y, \mathbf{w})$$

objective

$$\hat{\mathbf{w}} = \operatorname*{argmin}_{\mathbf{w} \in \mathbb{R}^d} \mathcal{L}_{\text{train}}(\mathbf{w})$$

optimal predictor

$$f_{\hat{\mathbf{w}}}(x) = \hat{\mathbf{w}} \cdot \phi(x)$$

Gradient Descent

Training loss gradient

$$\nabla_{\mathbf{w}} \mathcal{L}_{\text{train}}(\mathbf{w}) = \frac{1}{|\mathcal{D}_{\text{train}}|} \sum_{(x,y) \in \mathcal{D}_{\text{train}}} \overbrace{2(\mathbf{w} \cdot \phi(x) - y)\phi(x)}^{\nabla_{\mathbf{w}} \mathcal{L}_{\text{sq}}(x, y, \mathbf{w})}$$

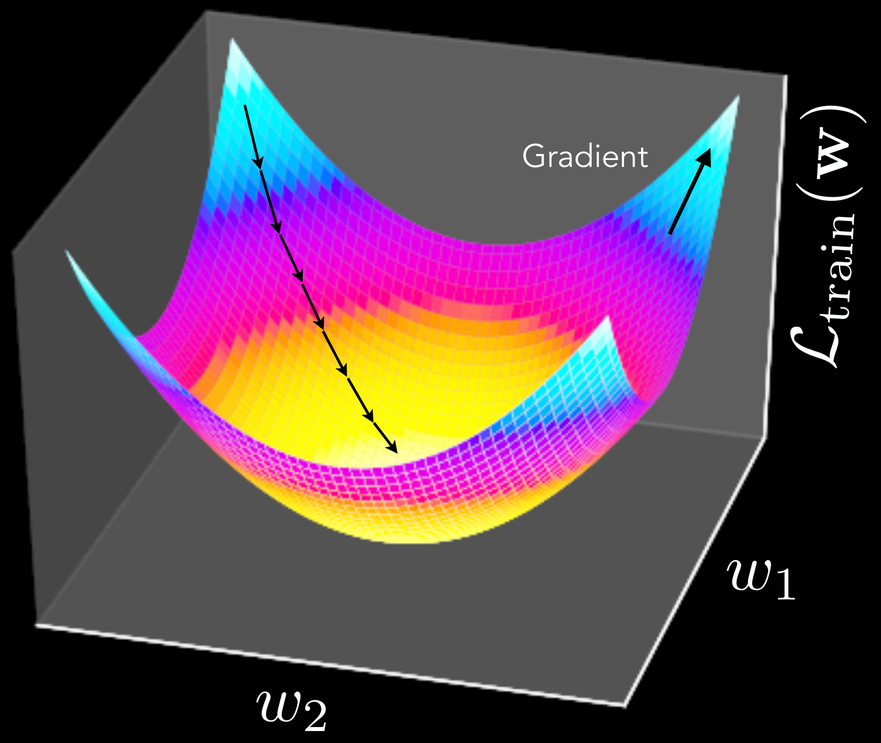
Prediction — True value

Gradient descent algorithm

epochs

initialize $\mathbf{w} = [0, \ldots, 0]$;
for $t = 1, \ldots, T$ **do**
  $\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla_{\mathbf{w}} \mathcal{L}_{\text{train}}$
end

step size



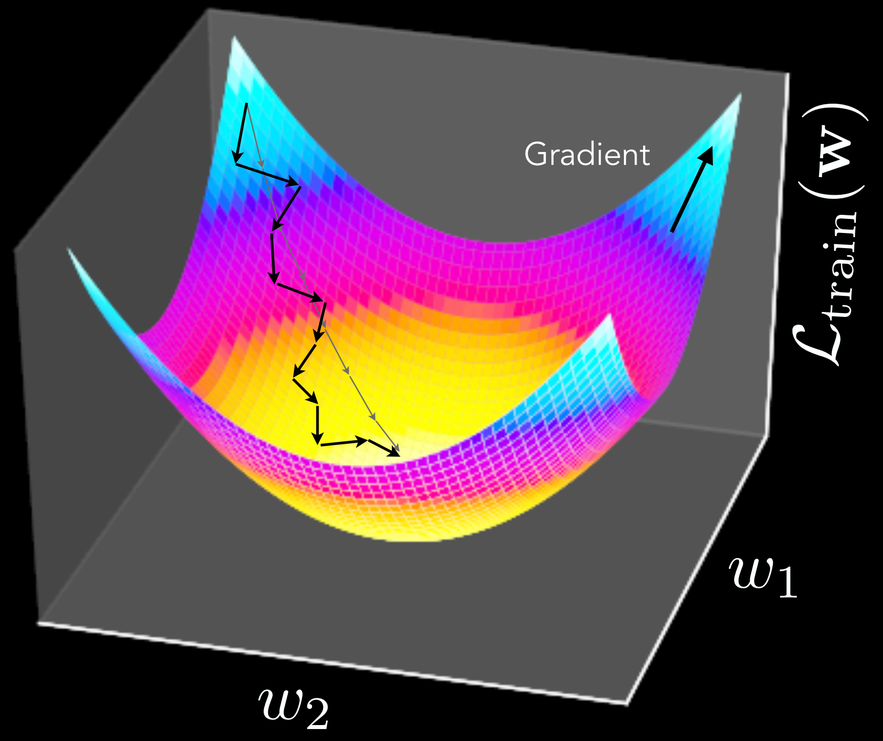
Stochastic Gradient Descent

Gradient descent is too expensive

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla_{\mathbf{w}} \mathcal{L}_{\text{train}}$$

Stochastic gradient descent

initialize $\mathbf{w} = [0, \ldots, 0]$;
for $t = 1, \ldots, T$ **do**
  for $(x, y) \in \mathcal{D}_{train}$ **do**
    $\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla_{\mathbf{w}} \mathcal{L}(x, y, \mathbf{w})$
  end
end

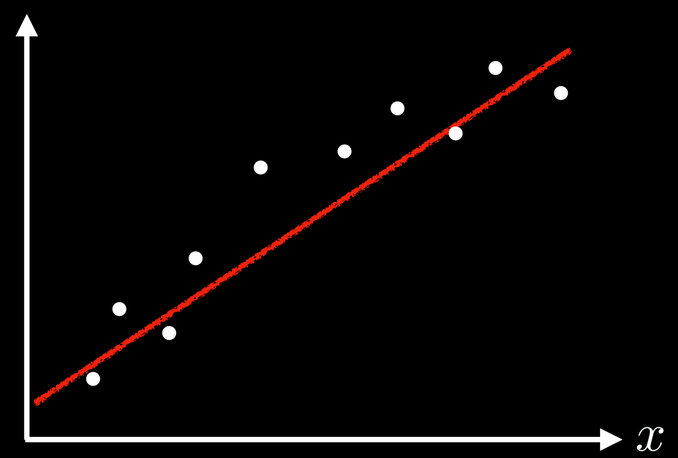


Variance vs. Bias

How do you choose $\phi(\cdot)$?

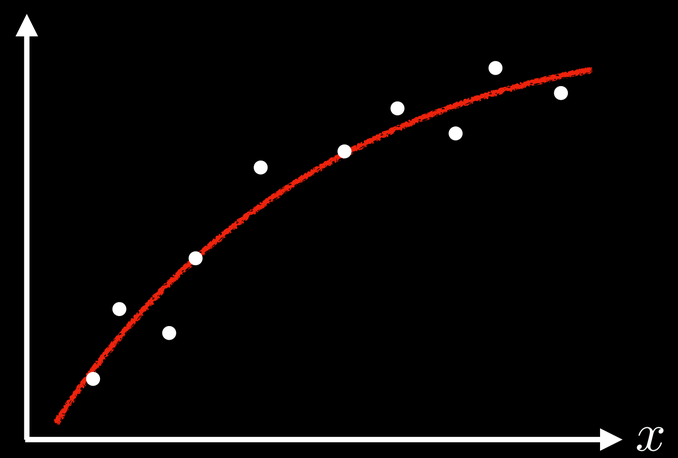
under-fitting

high bias



$\phi(x) = [1, x]$

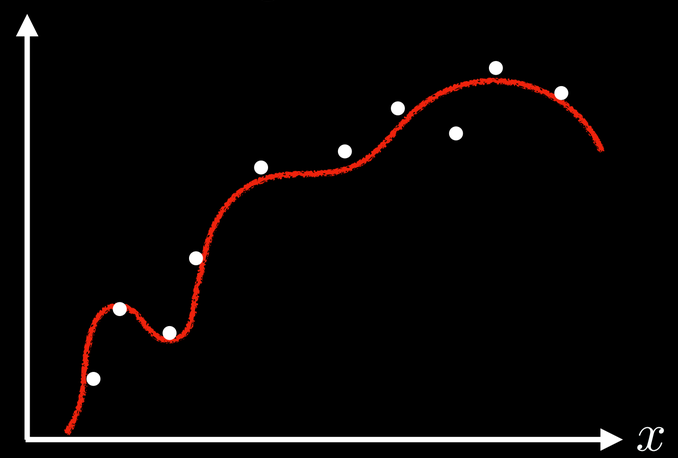
just right



$\phi(x) = [1, x, x^2]$

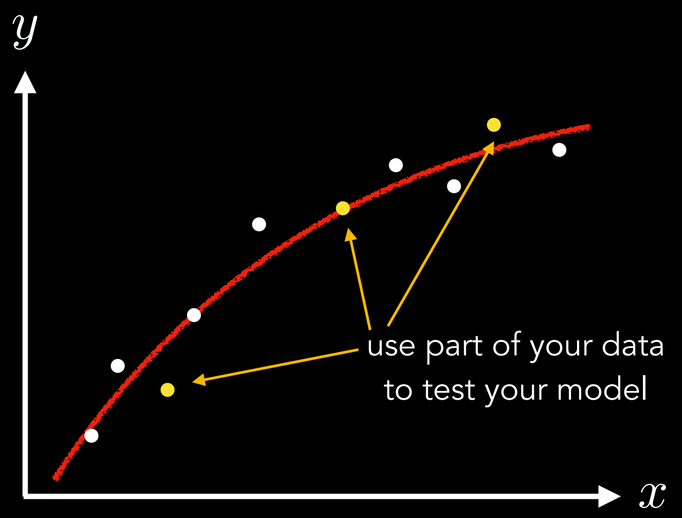
over-fitting

high variance



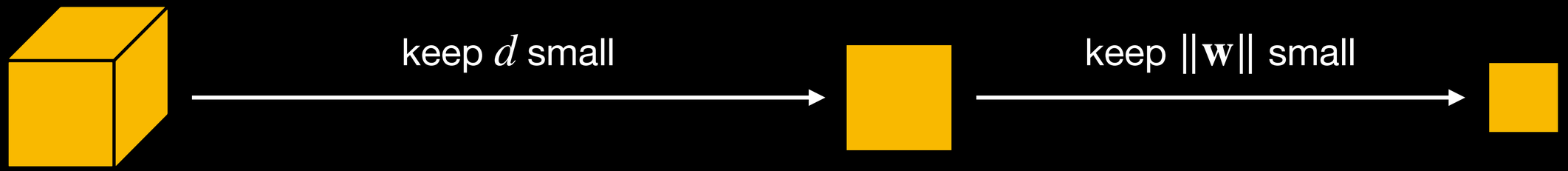
$\phi(x) = [1, x, x^2, \ldots, x^n]$

Use a test set



Designing the hypothesis class

$\mathbf{w} \in \mathbb{R}^d$



Remove features if they don't help

Use automatic feature selection

$$\mathcal{L} = \mathcal{L}_{\text{sq}} + \lambda \|\mathbf{w}\|_0$$

number of nonzero elements

Use L_2 regularization

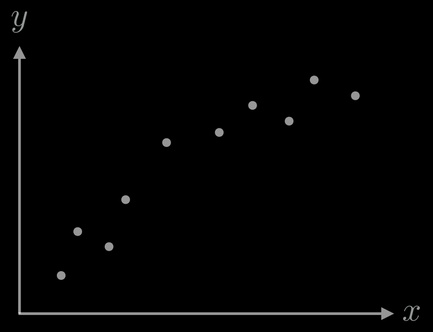
$$\mathcal{L} = \mathcal{L}_{\text{sq}} + \lambda \|\mathbf{w}\|^2$$

Hyperparameters

How do you choose the

- Regularization parameter
- Number of iterations
- Steps size
- …

The ML workflow



Can we automate feature extraction?

$$\phi(x) = [1, x, \ldots, ?]$$

not satisfied

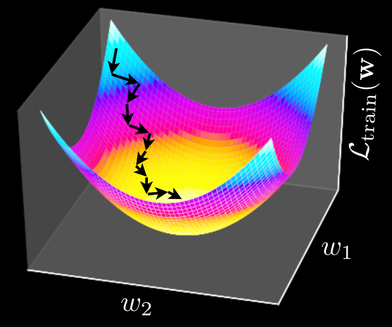
η

T λ

$\mathcal{L}_{\text{val}}$

satisfied

$\mathcal{L}_{\text{test}}$



Linear Predictor

$$f_{\mathbf{w}}(x) = \mathbf{w} \cdot \phi(x)$$

$$\phi(x) = [1,\ x]$$

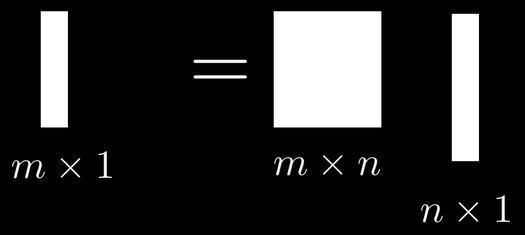
$$\phi(x) = [1, x, x^2, x^3]$$

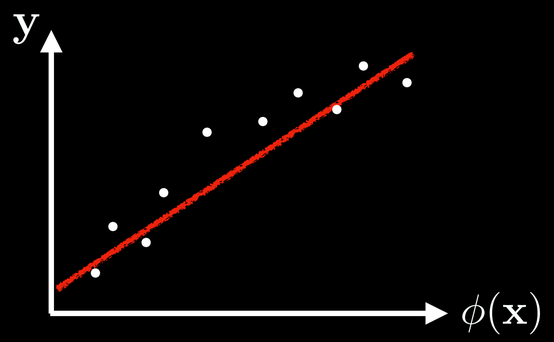
$$\phi(x) = [1,\ x,\ \sin(3x)]$$

Higher dimensions

$$\hat{\mathbf{y}} \qquad \mathbf{x}$$

$$f_{\mathbf{W}}(\mathbf{x}) = \mathbf{W}\ \phi(\mathbf{x})$$





Linear Predictor

Matrix multiplication

$$\hat{\mathbf{y}} = \mathbf{W}\mathbf{x}$$

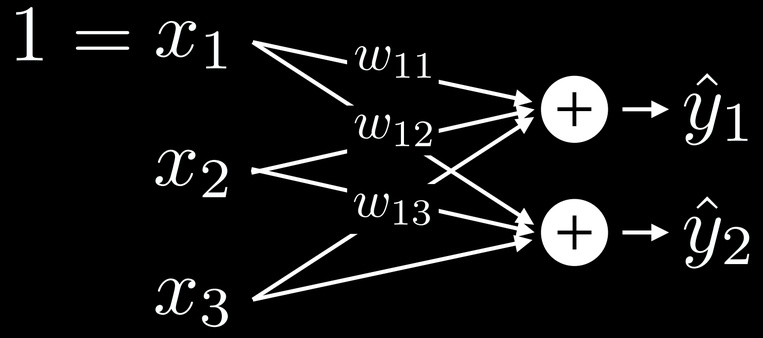
Index notation

$$\hat{y}_i = \sum_{j=1}^{n} w_{ij} x_j$$

$$\begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \end{bmatrix} = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

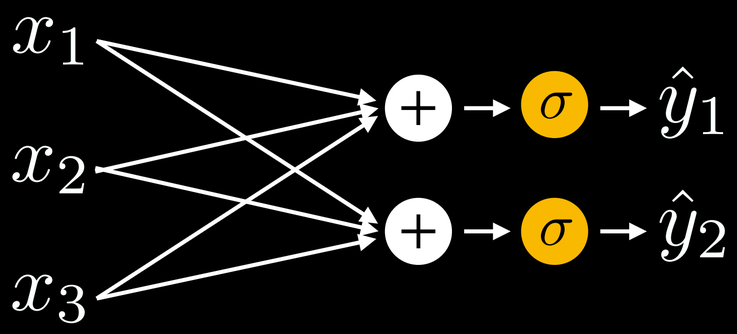
Define features such that
first component accounts for bias

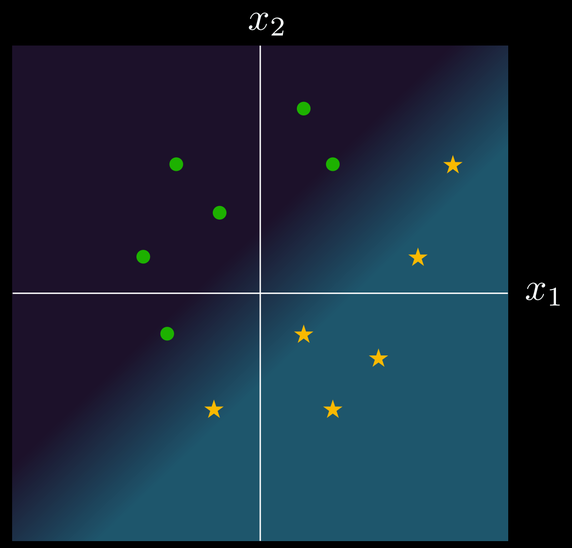
Network representation



Nonlinear Predictor

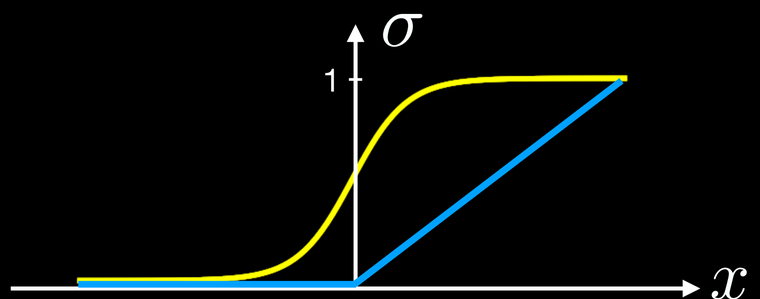
$$\hat{\mathbf{y}} = \sigma(\mathbf{W}\mathbf{x})$$





Logistic function: $\sigma(x) = \dfrac{1}{1+e^{-x}}$

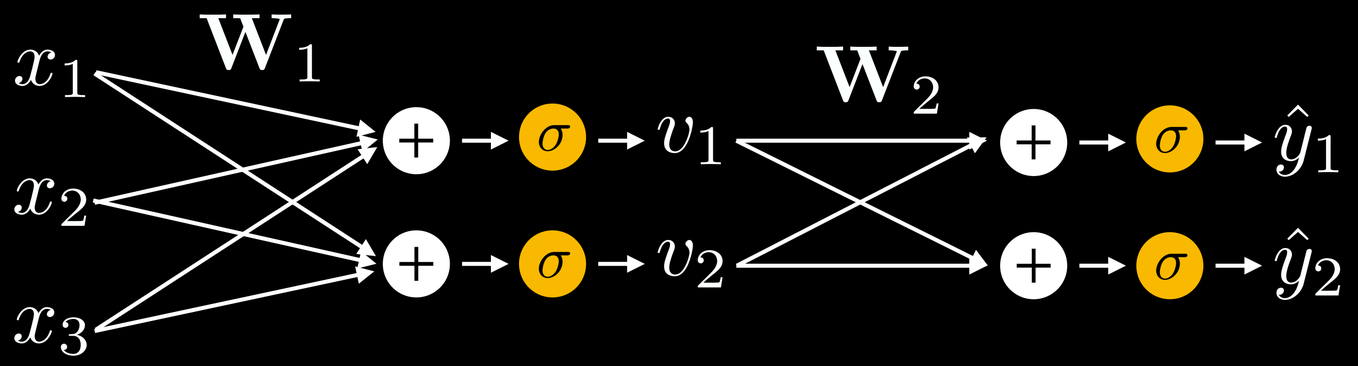
ReLU: $\sigma(x) = xH(x)$



Activation Function

Neural network

$$\hat{\mathbf{y}} = \sigma\left(\mathbf{W}_2\ \sigma\left(\mathbf{W}_1\mathbf{x}\right)\right)$$



$$\mathbf{x} \rightarrow \boxed{\sigma \circ \mathbf{W}_2 \circ \sigma \circ \mathbf{W}_1} \rightarrow \hat{\mathbf{y}}$$

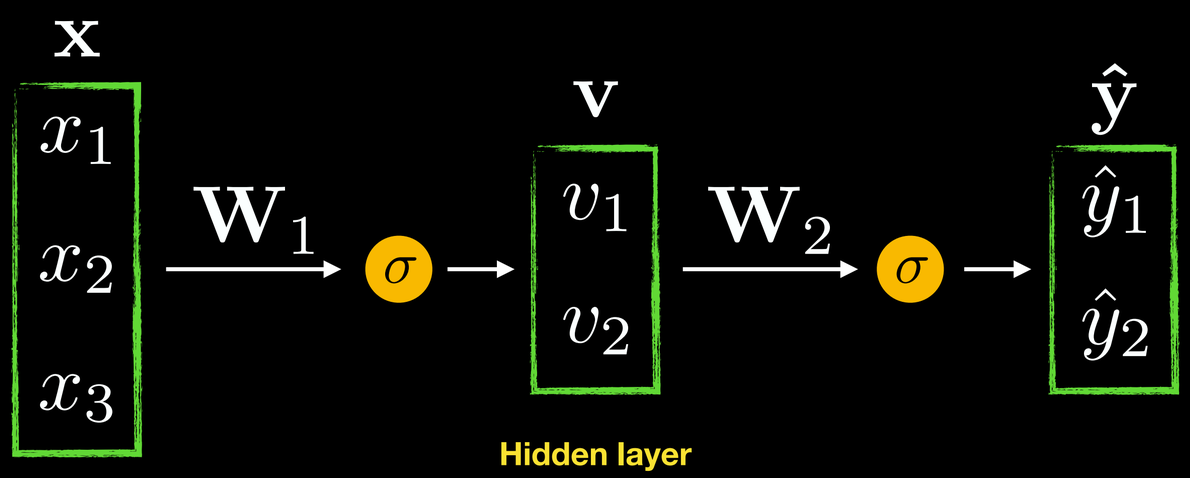
Nonlinear Linear Nonlinear Linear

Neural network

$$\hat{\mathbf{y}} = \sigma\left(\mathbf{W}_2\ \sigma\left(\mathbf{W}_1\mathbf{x}\right)\right)$$

$$\mathbf{v} = \sigma(\mathbf{W}_1\mathbf{x})$$

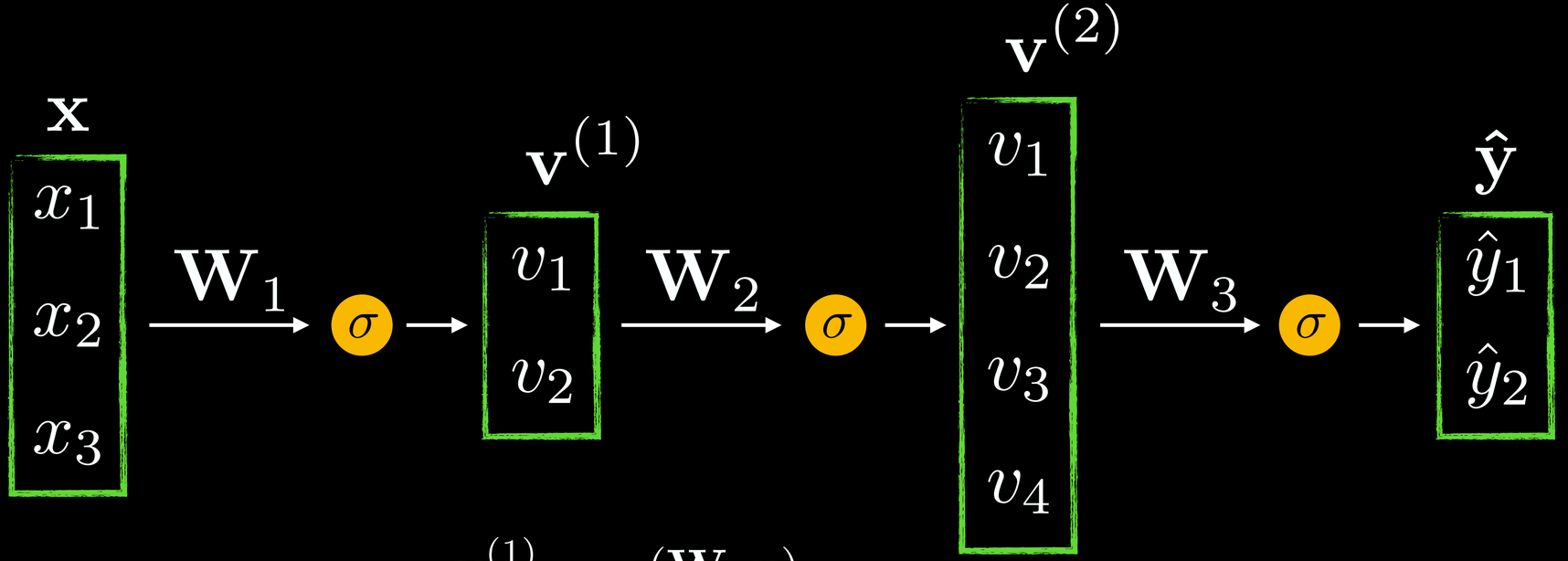
$$\hat{\mathbf{y}} = \sigma(\mathbf{W}_2\mathbf{v})$$



Can be interpreted as a learned $\phi(\mathbf{x})$

Deep network

$$\hat{\mathbf{y}} = \sigma\left(\mathbf{W}_3\ \sigma\left(\mathbf{W}_2\ \sigma\left(\mathbf{W}_1\mathbf{x}\right)\right)\right)$$

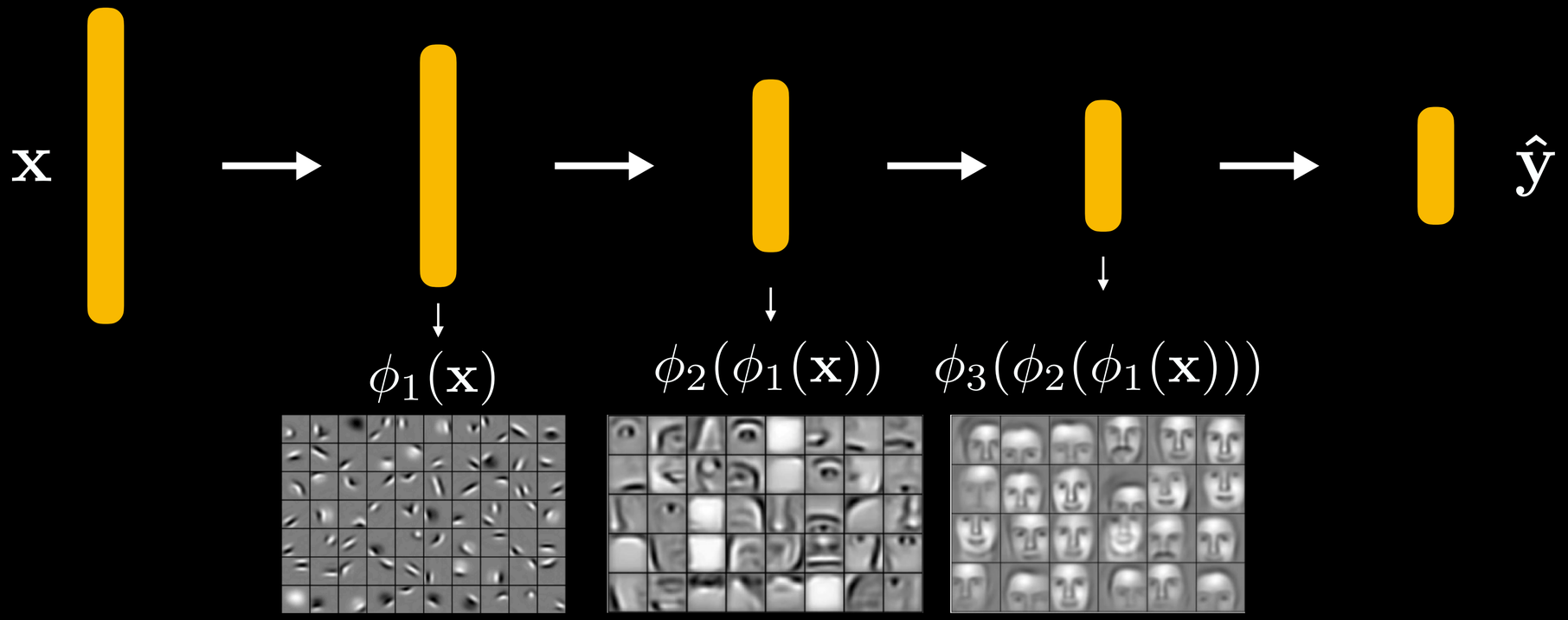


$$\mathbf{v}^{(1)} = \sigma\left(\mathbf{W}_1\mathbf{x}\right)$$

$$\mathbf{v}^{(2)} = \sigma\left(\mathbf{W}_2\mathbf{v}^{(1)}\right)$$

$$\hat{\mathbf{y}} = \sigma\left(\mathbf{W}_3\mathbf{v}^{(2)}\right)$$

Why *deep* learning?



Feature learning!

Lee, Honglak, et al. "Unsupervised learning of hierarchical representations with convolutional deep belief networks." Communications of the ACM 54.10 (2011)

Loss function

$$f_{\mathbf{W}_1 \mathbf{W}_2}(\mathbf{x}) = \sigma\left(\mathbf{W}_2\ \sigma\left(\mathbf{W}_1 \mathbf{x}\right)\right)$$

$$\mathcal{L}(\mathbf{x}, \mathbf{y}, \mathbf{W}_1, \mathbf{W}_2) = \left\| f_{\mathbf{W}_1 \mathbf{W}_2}(\mathbf{x}) - \mathbf{y} \right\|^2$$

Stochastic gradient descent update

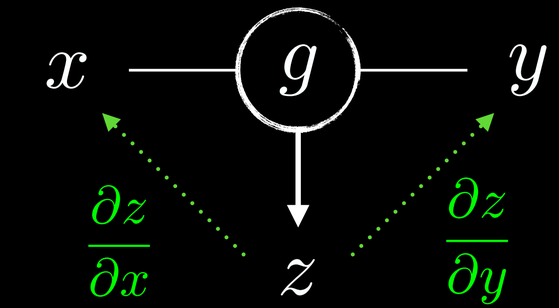
$$\mathbf{W}_1 \leftarrow \mathbf{W}_1 - \eta \nabla_{\mathbf{W}_1} \mathcal{L}(\mathbf{x}, \mathbf{y}, \mathbf{W}_1, \mathbf{W}_2)$$

$$\mathbf{W}_2 \leftarrow \mathbf{W}_2 - \eta \nabla_{\mathbf{W}_2} \mathcal{L}(\mathbf{x}, \mathbf{y}, \mathbf{W}_1, \mathbf{W}_2)$$

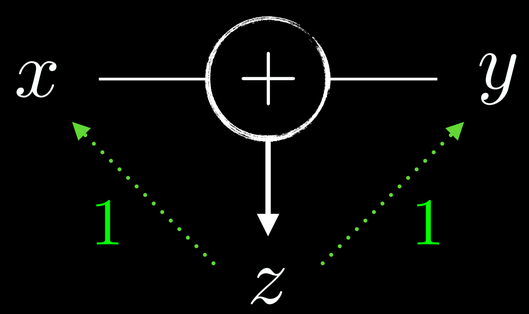
How do we calculate the gradients?

Computation graphs

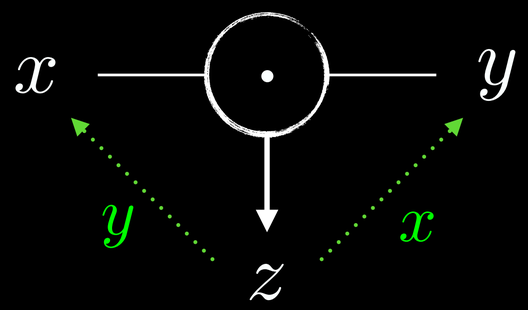
$$z = g(x, y)$$



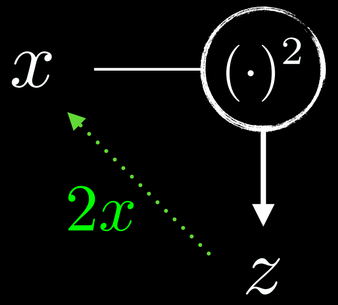
$$z = x + y$$



$$z = xy$$

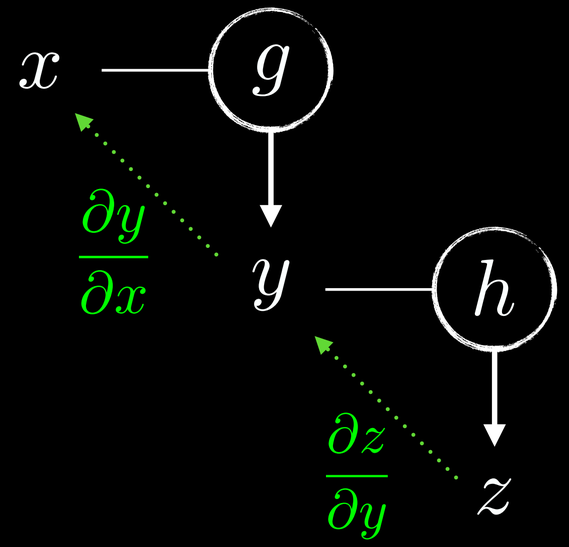


$$z = x^2$$



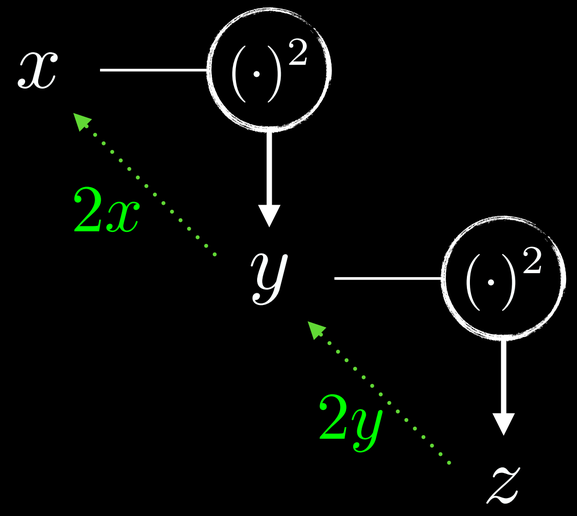
Chain rule

$$y = g(x)$$
$$z = h(y)$$



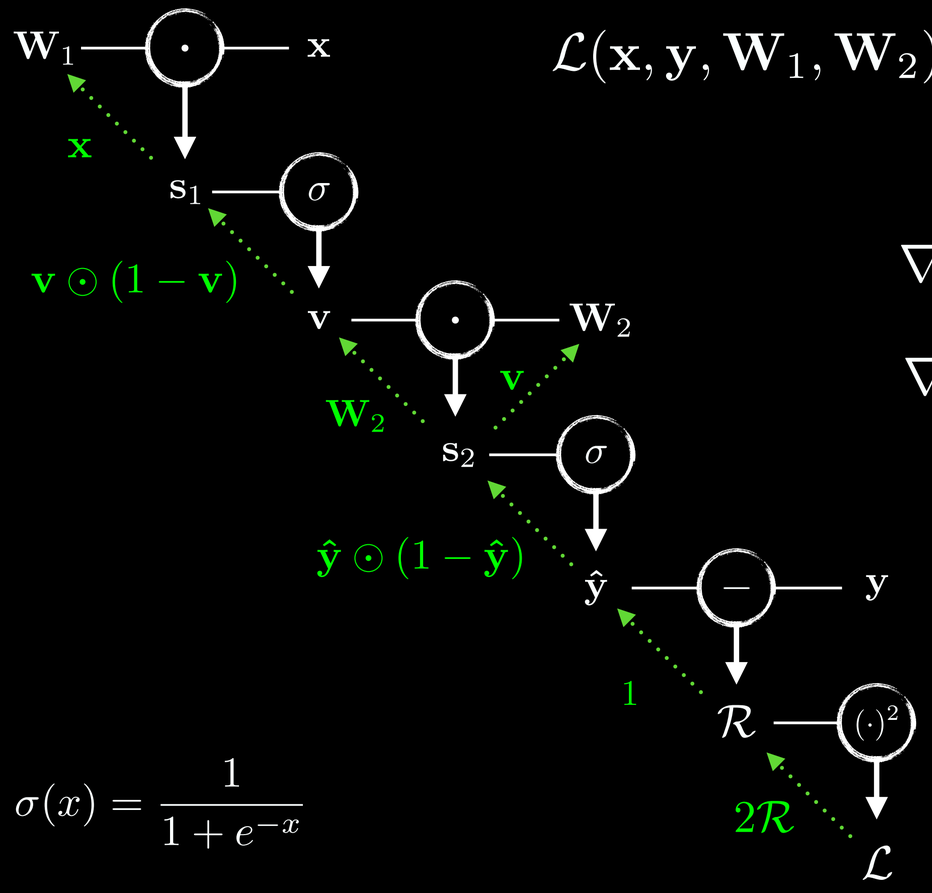
$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y}\frac{\partial y}{\partial x}$$

$$y = x^2$$
$$z = y^2$$



$$\frac{\partial z}{\partial x} = 4xy = 4x^3$$

Backpropagation



$$\mathcal{L}(\mathbf{x}, \mathbf{y}, \mathbf{W}_1, \mathbf{W}_2) = \|\sigma(\mathbf{W}_2\,\sigma(\mathbf{W}_1\mathbf{x})) - \mathbf{y}\|^2$$

$$\nabla_{\mathbf{W}_1}\mathcal{L} = 2\mathbf{W}_2^\top \mathcal{R} \odot \hat{\mathbf{y}} \odot (1 - \hat{\mathbf{y}}) \odot \mathbf{v} \odot (1 - \mathbf{v})\mathbf{x}^\top$$

$$\nabla_{\mathbf{W}_2}\mathcal{L} = 2\mathcal{R} \odot \hat{\mathbf{y}} \odot (1 - \hat{\mathbf{y}})\mathbf{v}^\top$$

assuming $\sigma(x) = \dfrac{1}{1 + e^{-x}}$

Optimization

Training loss

$$\mathcal{L}_{\text{train}}(\mathbf{W}_1, \mathbf{W}_2) = \frac{1}{|\mathcal{D}_{\text{train}}|} \sum_{(\mathbf{x},\mathbf{y}) \in \mathcal{D}_{\text{train}}} \mathcal{L}(\mathbf{x}, \mathbf{y}, \mathbf{W}_1, \mathbf{W}_2)$$

Objective

$$\hat{\mathbf{W}}_1, \hat{\mathbf{W}}_2 = \underset{\mathbf{W}_1, \mathbf{W}_2}{\operatorname{argmin}} \mathcal{L}_{\text{train}}(\mathbf{W}_1, \mathbf{W}_2)$$

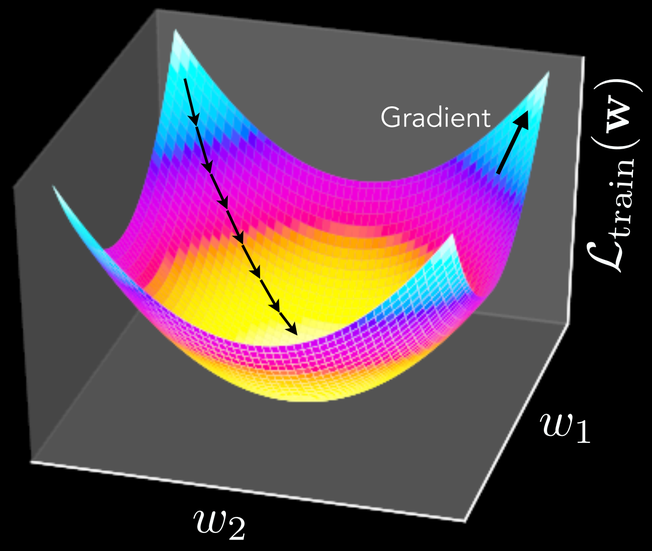
Optimal predictor

$$f_{\hat{\mathbf{W}}_1 \hat{\mathbf{W}}_2}(\mathbf{x}) = \sigma\left(\hat{\mathbf{W}}_2 \sigma\left(\hat{\mathbf{W}}_1 \mathbf{x}\right)\right)$$

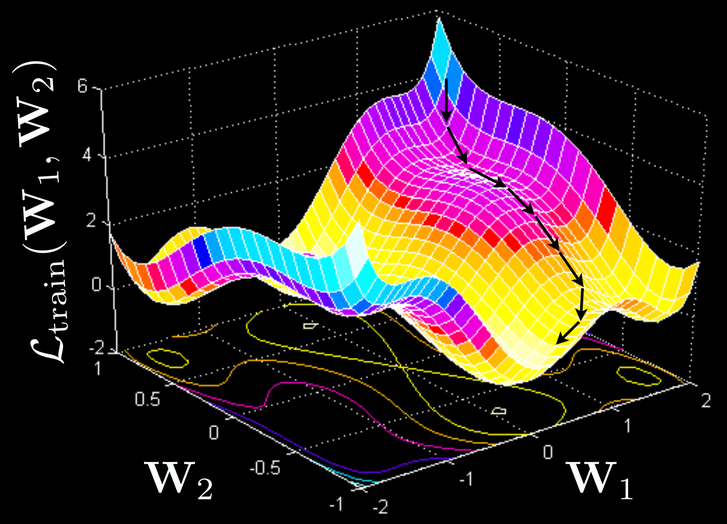
Non-convexity

$$\hat{\mathbf{W}}_1, \hat{\mathbf{W}}_2 = \underset{\mathbf{W}_1, \mathbf{W}_2}{\operatorname{argmin}} \mathcal{L}_{\text{train}}(\mathbf{W}_1, \mathbf{W}_2)$$

Linear predictor loss



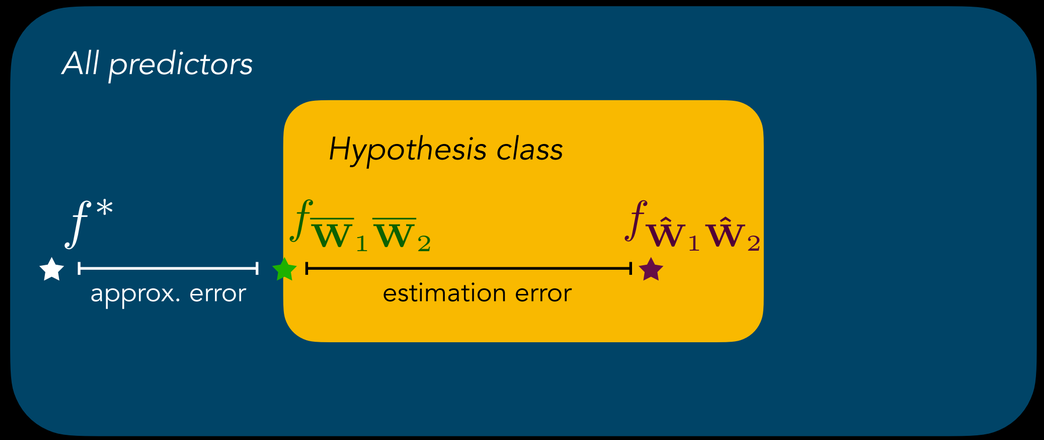
Neural network loss



Hypothesis Class

$$f_{\mathbf{W}_1 \mathbf{W}_2}(\mathbf{x}) = \sigma\left(\mathbf{W}_2\ \sigma\left(\mathbf{W}_1 \mathbf{x}\right)\right)$$

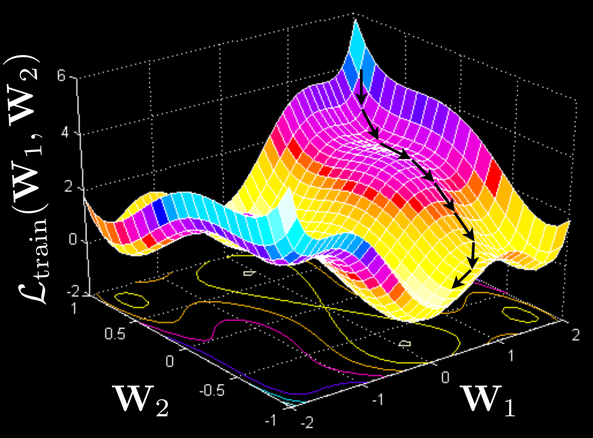
$$\mathcal{F} = \left\{ f_{\mathbf{W}_1 \mathbf{W}_2}(\mathbf{x}) \mid \mathbf{W}_1 \in \mathbb{R}^{k \times n}, \mathbf{W}_2 \in \mathbb{R}^{m \times k} \right\}$$



Hyperparameters

How do you train a deep network?

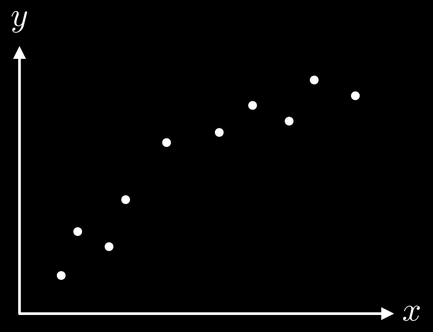
- Use many hidden layers for abstraction
- Use adaptive time steps
- Use hyper-parameter optimization



https://playground.tensorflow.org/

http://vision.stanford.edu/teaching/cs231n-demos/linear-classify/

The ML workflow



- Data cleaning
- Dimensionality reduction
- Unsupervised metrics

$\phi(\mathbf{x})$

η

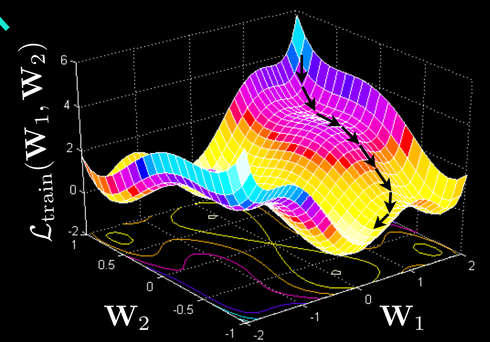
T λ

$\mathcal{L}_{\text{val}}$

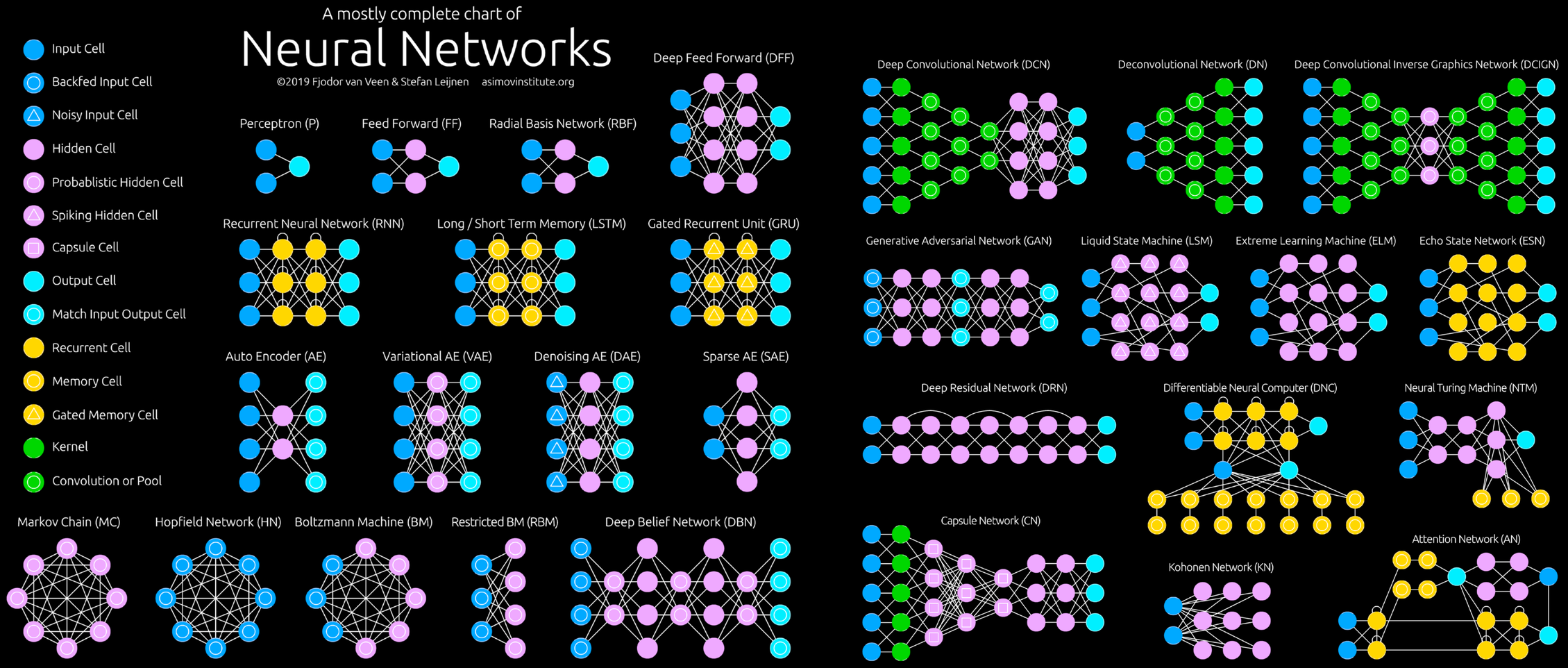
not satisfied

satisfied

$\mathcal{L}_{\text{test}}$

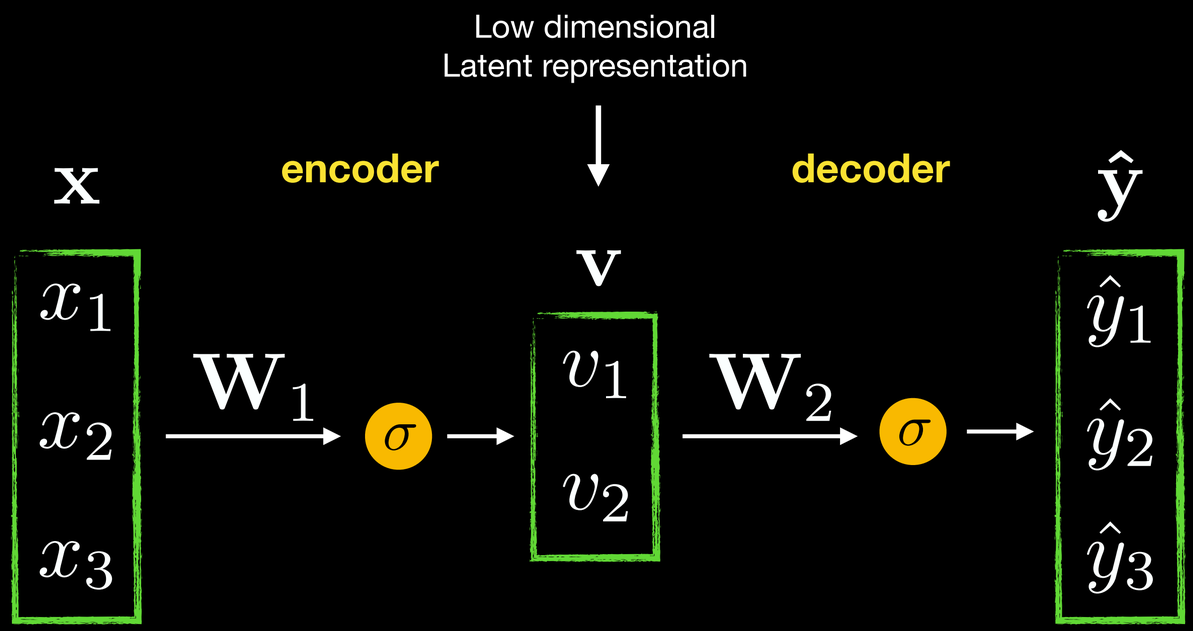


Neural network zoo



asimovinstitute.org

Auto-encoders

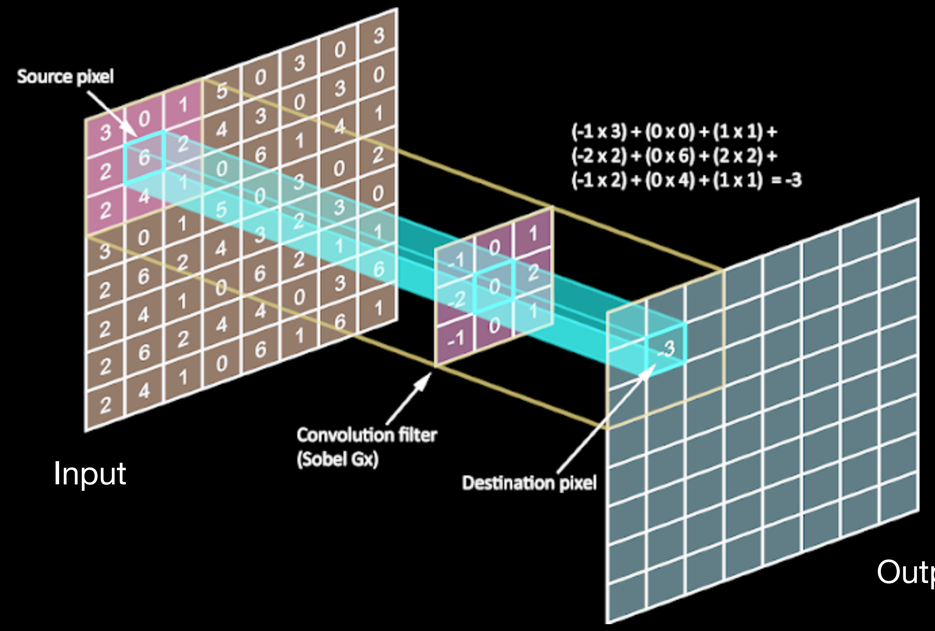


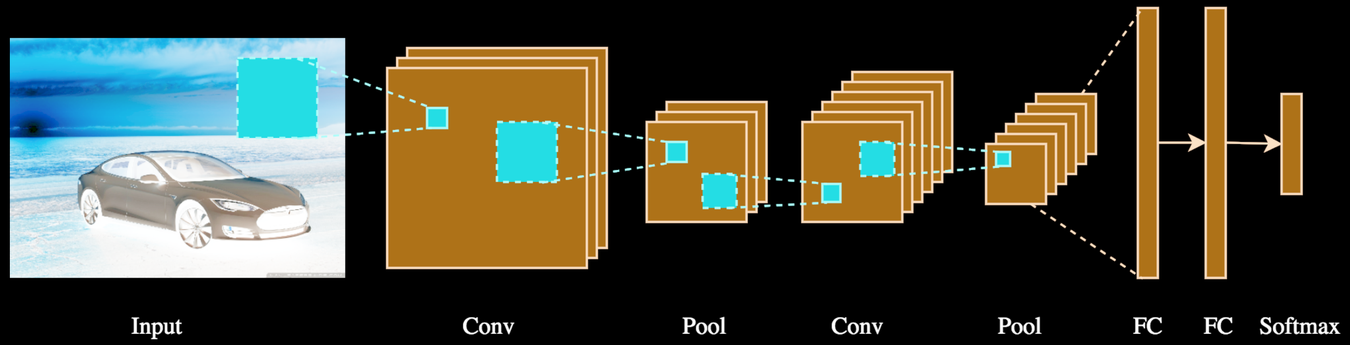
$$\mathcal{L} = \|f_{\mathbf{W}_1\mathbf{W}_2}(\mathbf{x}) - \mathbf{x}\|^2$$

if $\sigma = I$, network performs SVD decomposition

$$\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^*$$

Convolutional neural networks



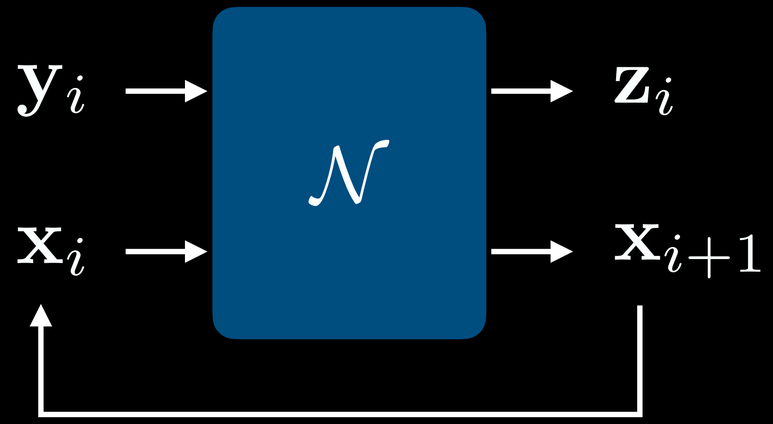


https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2

Recurrent neural networks

$$\boxed{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \ldots, \mathbf{x}_i, \mathbf{x}_{i+1}, \ldots, \mathbf{x}_n}$$

$$[\mathbf{x}_{i+1}, \mathbf{z}_i] = \mathcal{N}(\mathbf{x}_i, \mathbf{y}_i)$$

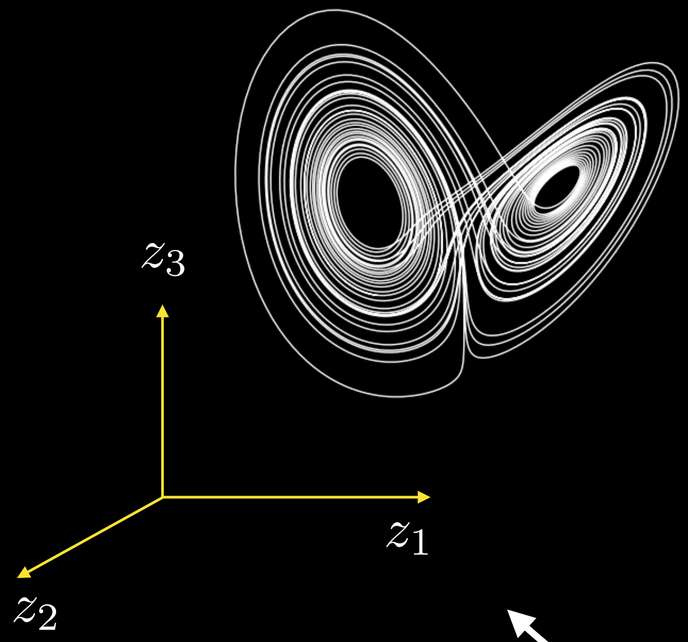


Feedback output to input

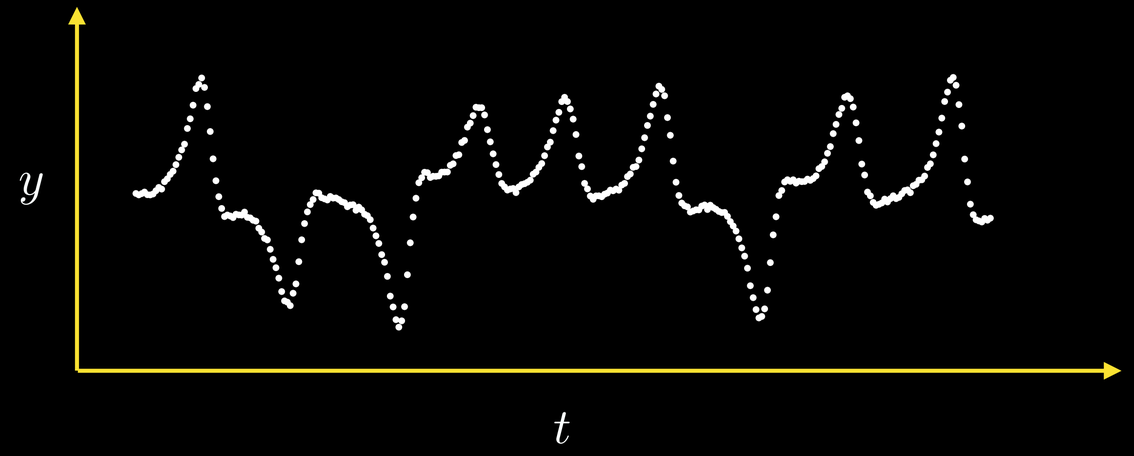
Numerical solvers are recurrence relations!

High Dimensional System

Partial Measurements



Measure

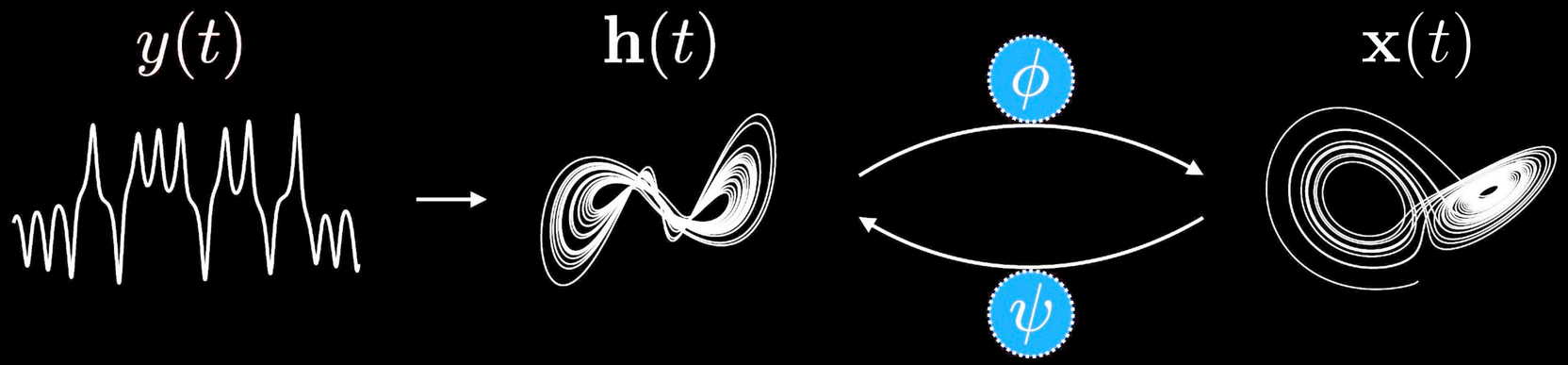


Reconstruct

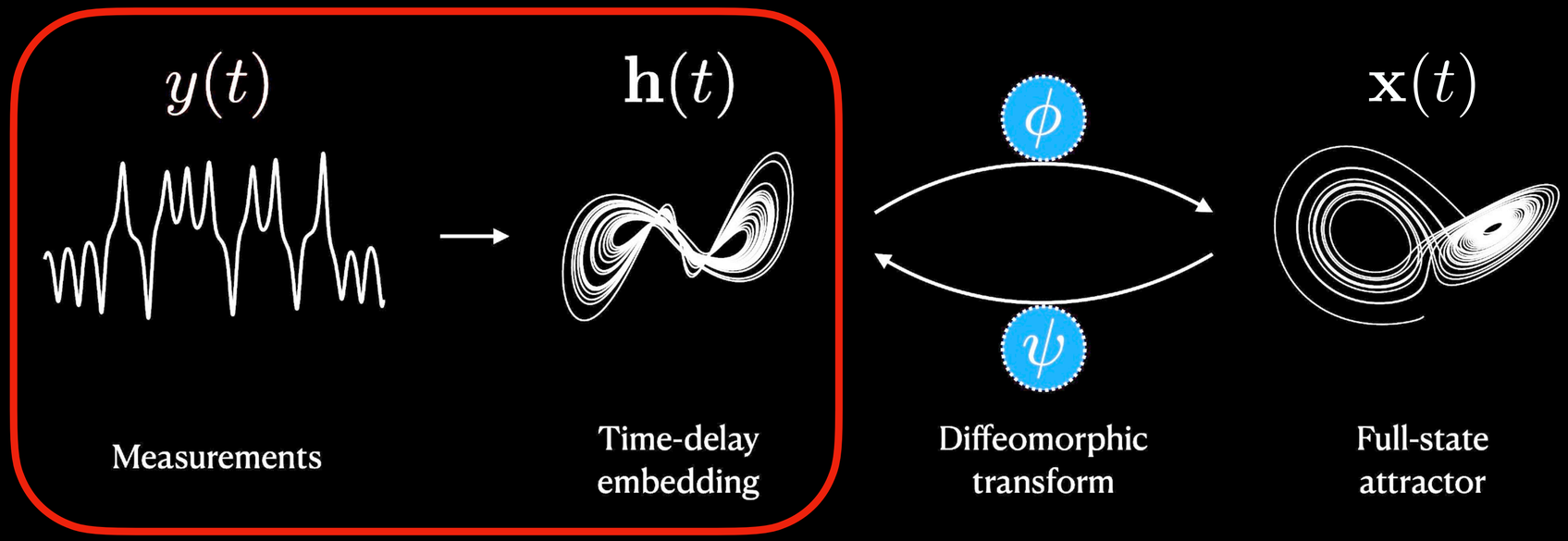
?

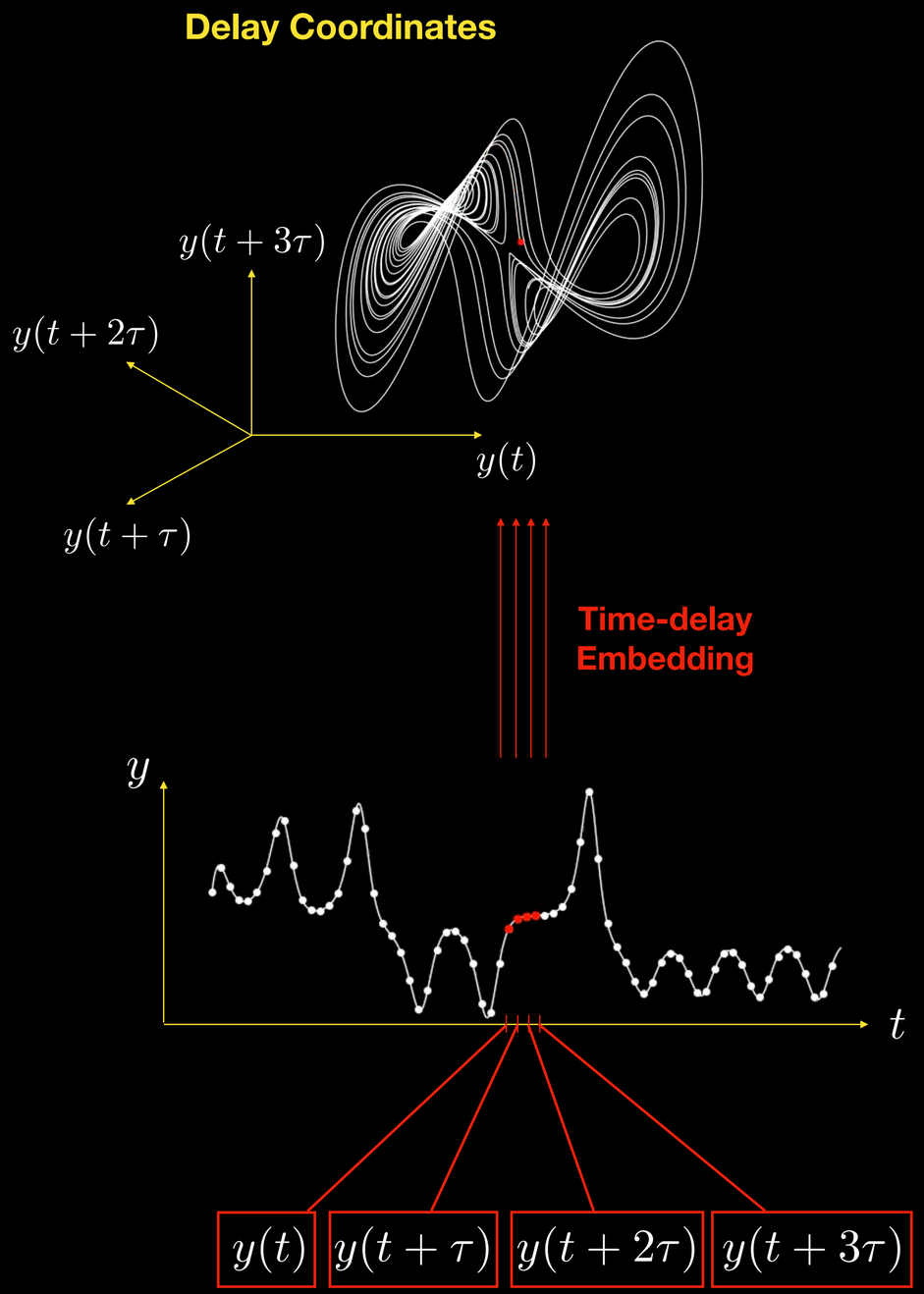
Takens' Theorem

Detecting strange attractors in turbulence (1981)



Time-delay embedding

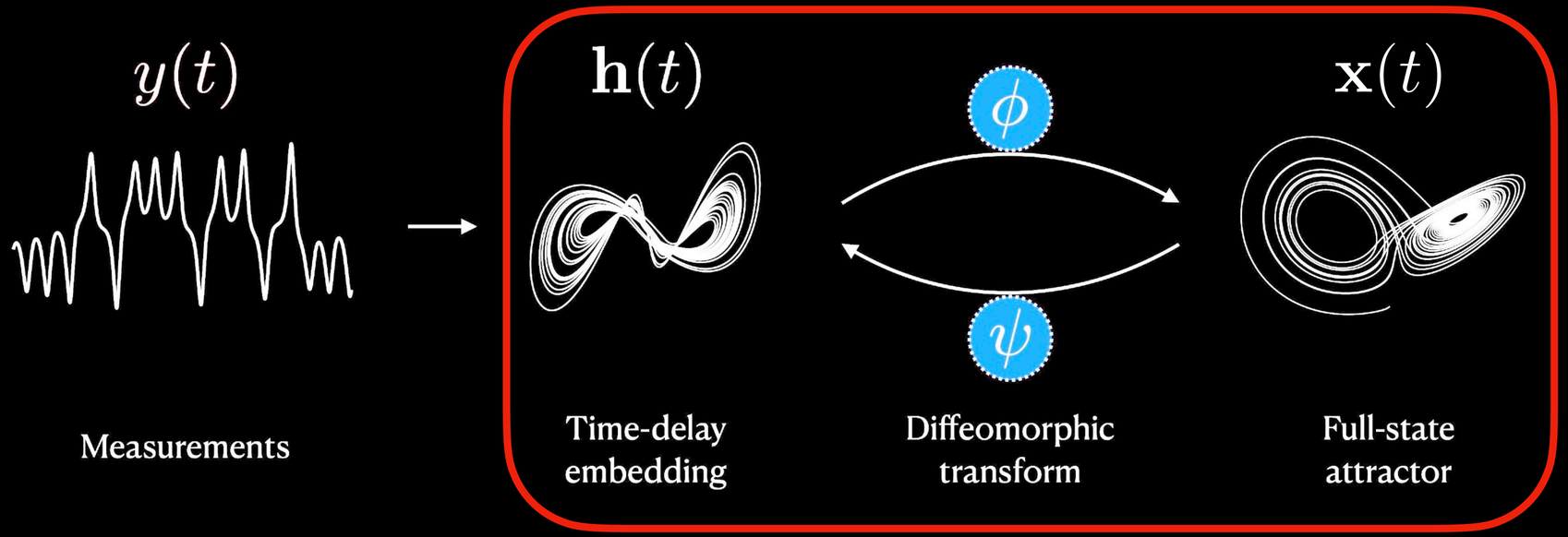




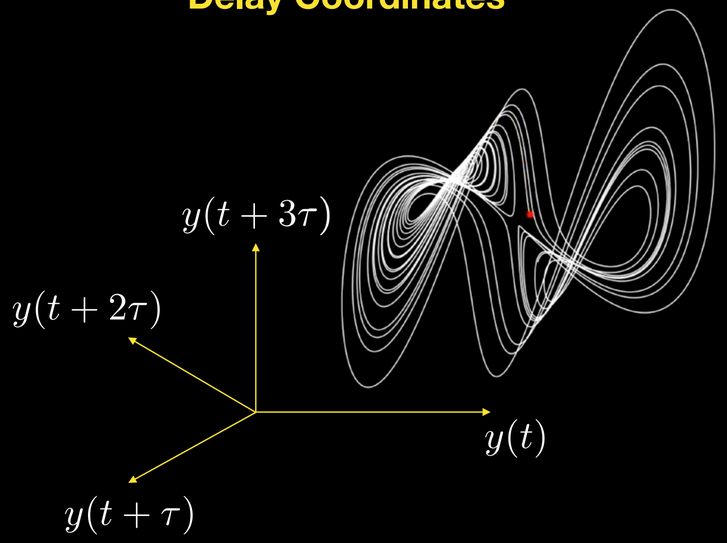
Hankel Matrix

$$\mathbf{H} = \begin{bmatrix} y(t_1) & y(t_2) & \dots & y(t_q) \\ y(t_2) & y(t_3) & \dots & y(t_{q+1}) \\ \vdots & \vdots & \ddots & \vdots \\ y(t_n) & y(t_{n+1}) & \dots & y(t_{n+q+1}) \end{bmatrix} = \left[\mathbf{h}_1, \mathbf{h}_2, \dots, \mathbf{h}_q\right].$$

Takens' Theorem

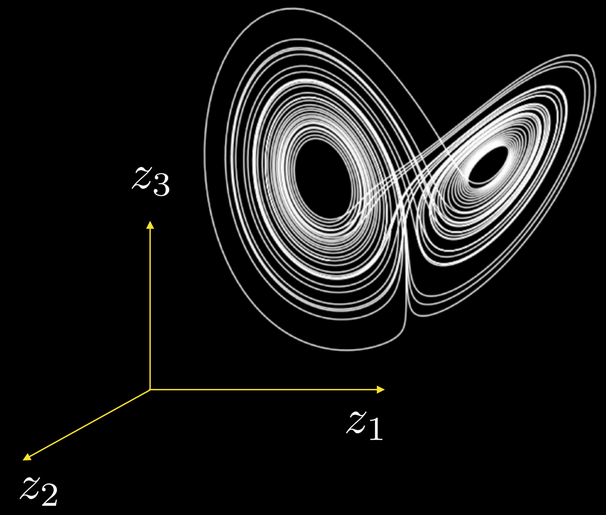


Delay Coordinates



Diffeomorphic

Full State Coordinates

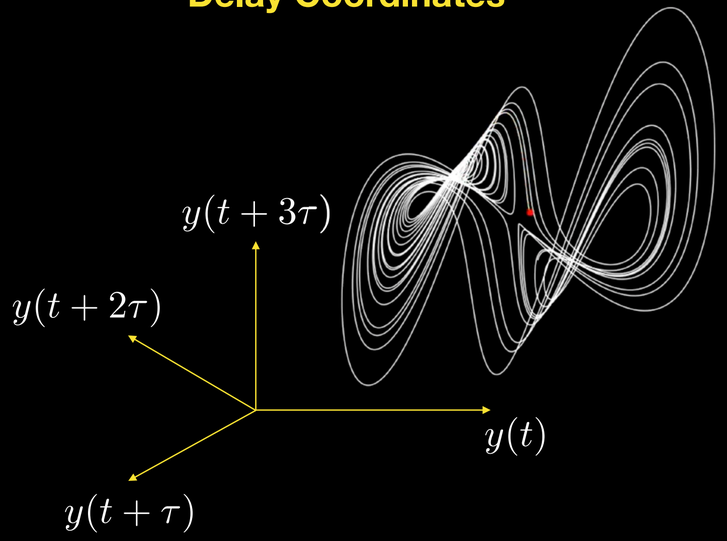


Takens' Theorem

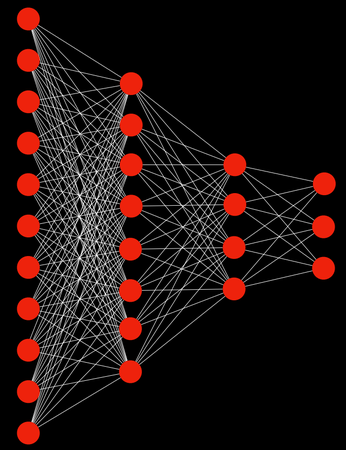
If the dynamical system and the observed quantity are generic, then the delay-coordinate map from a d-dimensional smooth compact manifold $\mathscr{M}$ to $\mathbb{R}^{2d+1}$ is a diffeomorphism on $\mathscr{M}$.

F. Takens
Detecting strange attractors in turbulence (1981)

Delay Coordinates



Diffeomorphic



Full State Coordinates