Machine Learning for Systems and Control

5SC28

Lecture 4

dr. ir. Maarten Schoukens & dr. ir. Roland Tóth

Control Systems Group

Department of Electrical Engineering

Eindhoven University of Technology

Academic Year: 2020-2021 (version 1.0)

EINDHOVEN UNIVERSITY OF TECHNOLOGY

Past Lectures

Data-Driven Modelling

RKHS / Gaussian Processes

Artificial Neural Networks

Learning Objectives

Deep Learning & Deep Neural Networks

Training a Deep Neural Network

Artificial Neural Networks for Dynamical Systems

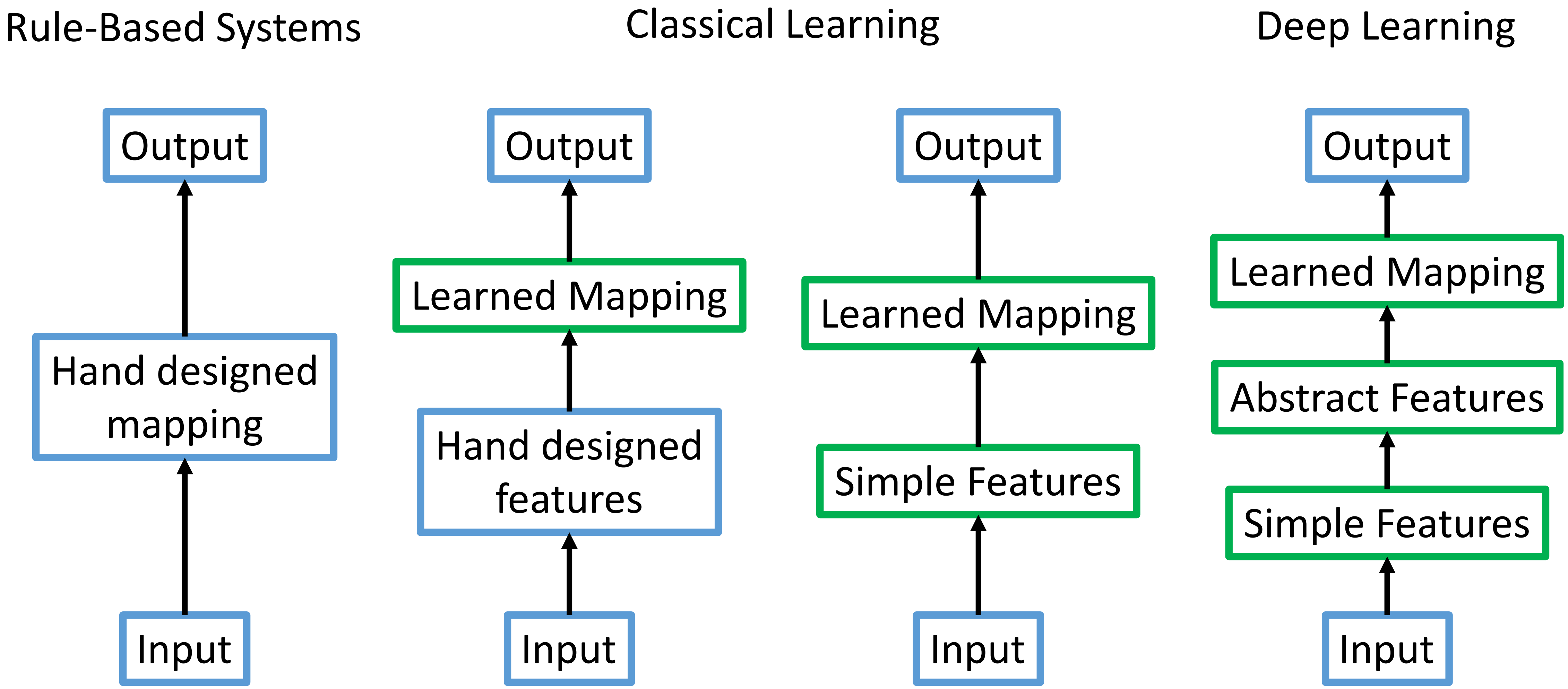
Artificial Neural Networks

Deep Learning & Deep Neural Networks

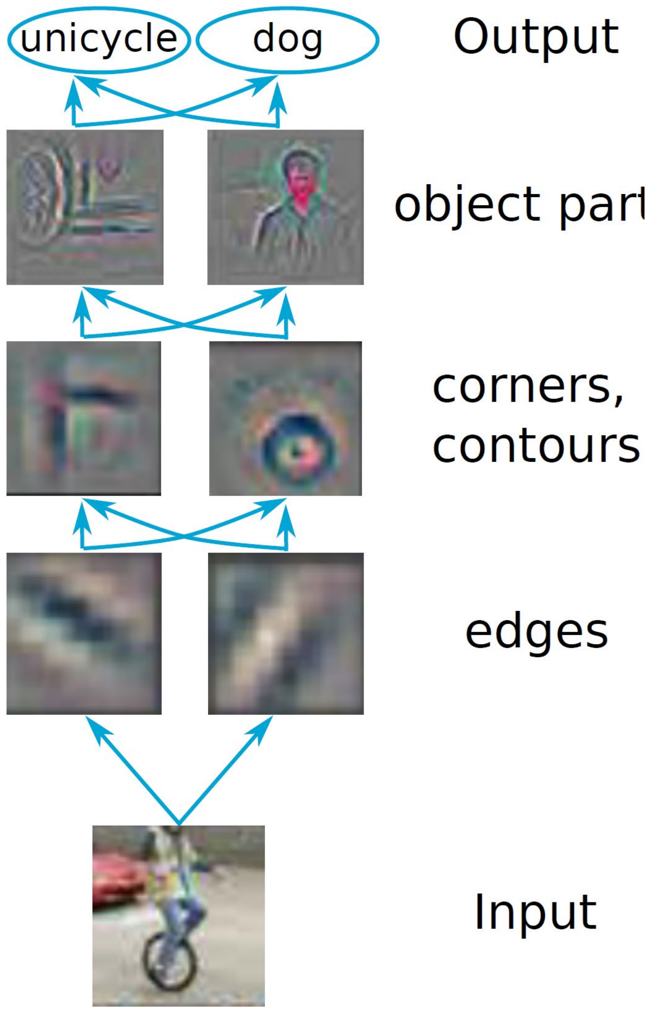
Training a Deep Neural Network

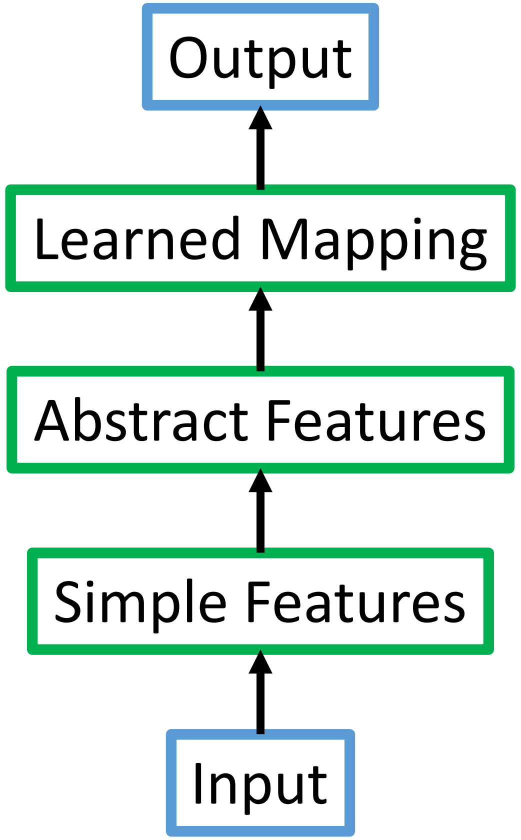
Artificial Neural Networks for Dynamical Systems

Deep Learning



Deep Learning

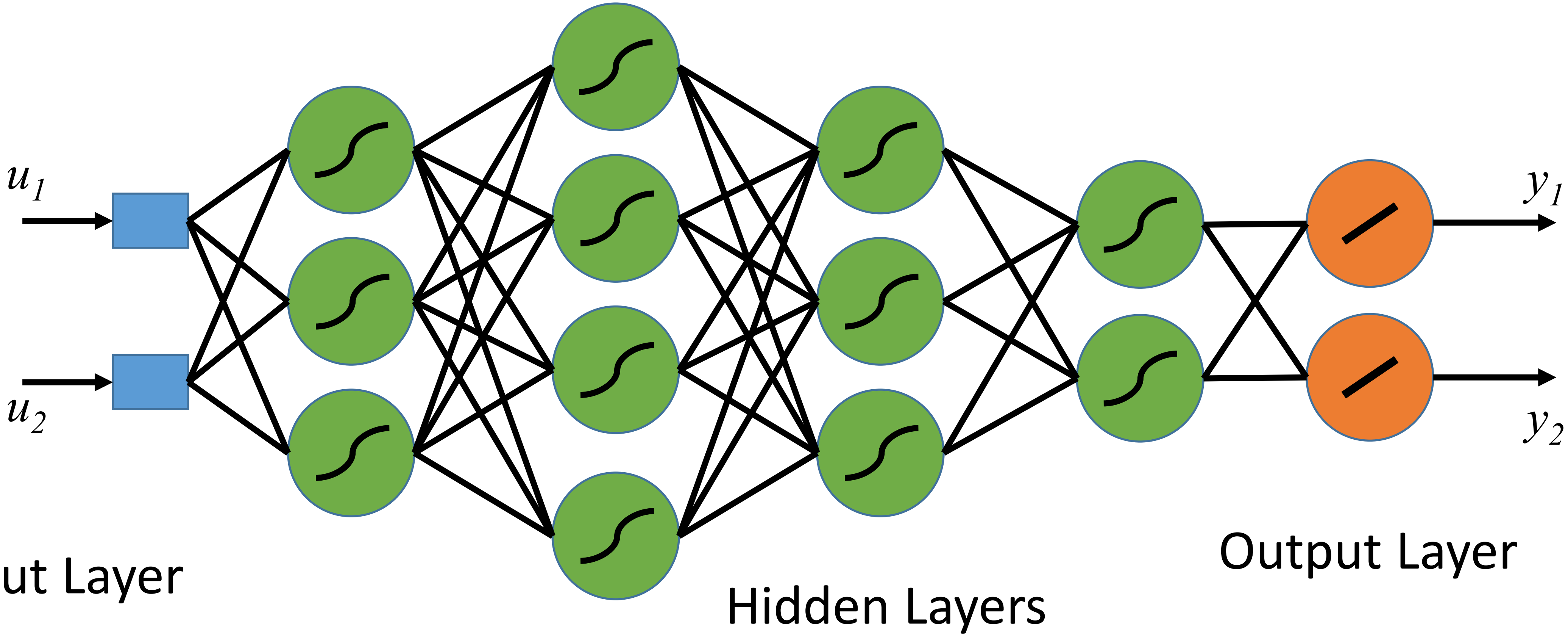




source: http://www.dcsc.tudelft.nl/~sc42050/2018/Materials/180305_Lecture_7_Artificial_neural_networks_I.pdf

Deep Neural Network

"A network with multiple hidden layers"[1]



[1] Michael A. Nielsen, "Neural Networks and Deep Learning", Determination Press, 2015

Deep Neural Networks

Deep Feedforward

Recurrent

Long-Short Term Memory

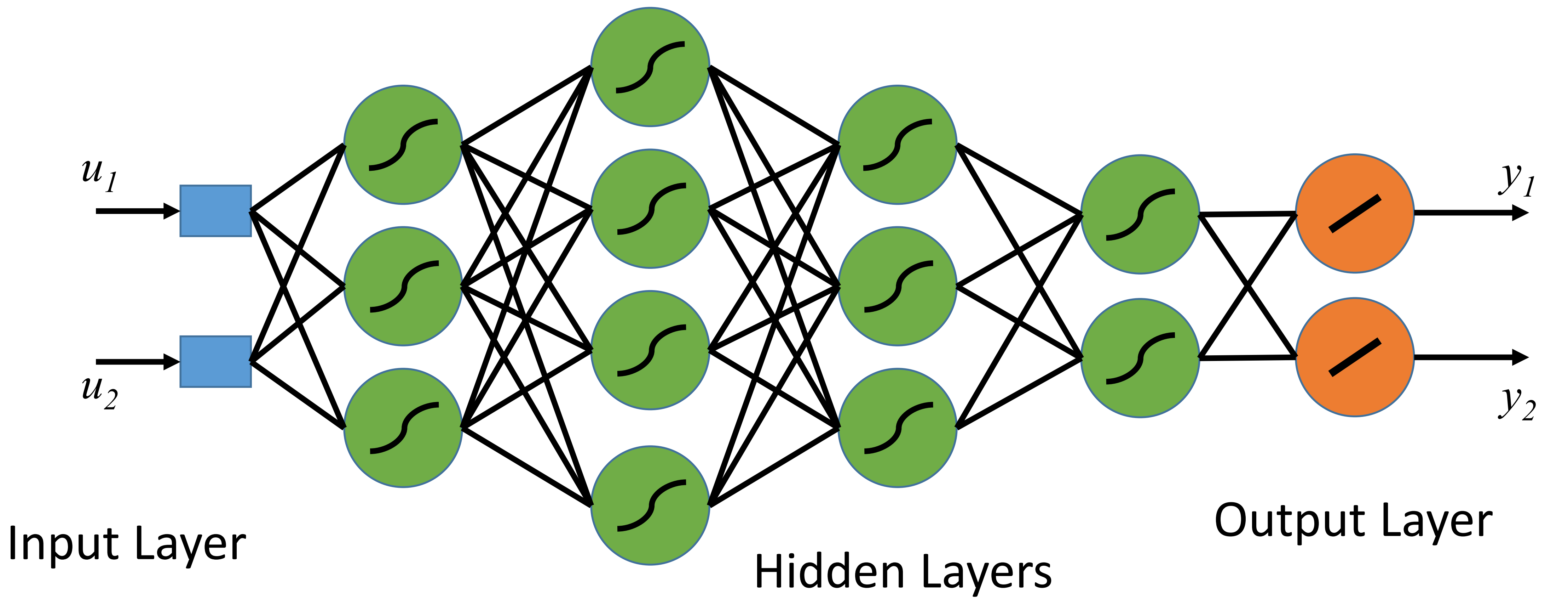
Autoencoder

Residual

Convolutional

…

Deep Feedforward Neural Network

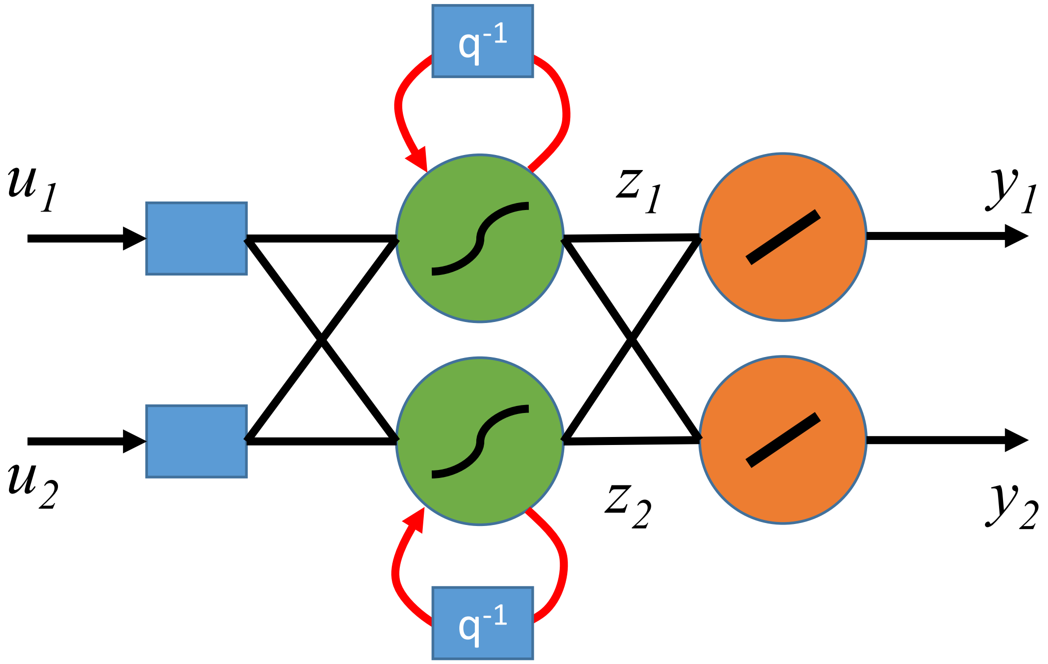


Recurrent Neural Networks

Multiple recurrence schemes possible

$$z_j(k) = g\left(\boldsymbol{w}_{1,j}^T \boldsymbol{u}(k) + v_{1,j} z_j(k-1) + b_{1,j}\right)$$

$$y_j(k) = \boldsymbol{w}_{2,j}^T \boldsymbol{z}(k) + b_{2,j}$$



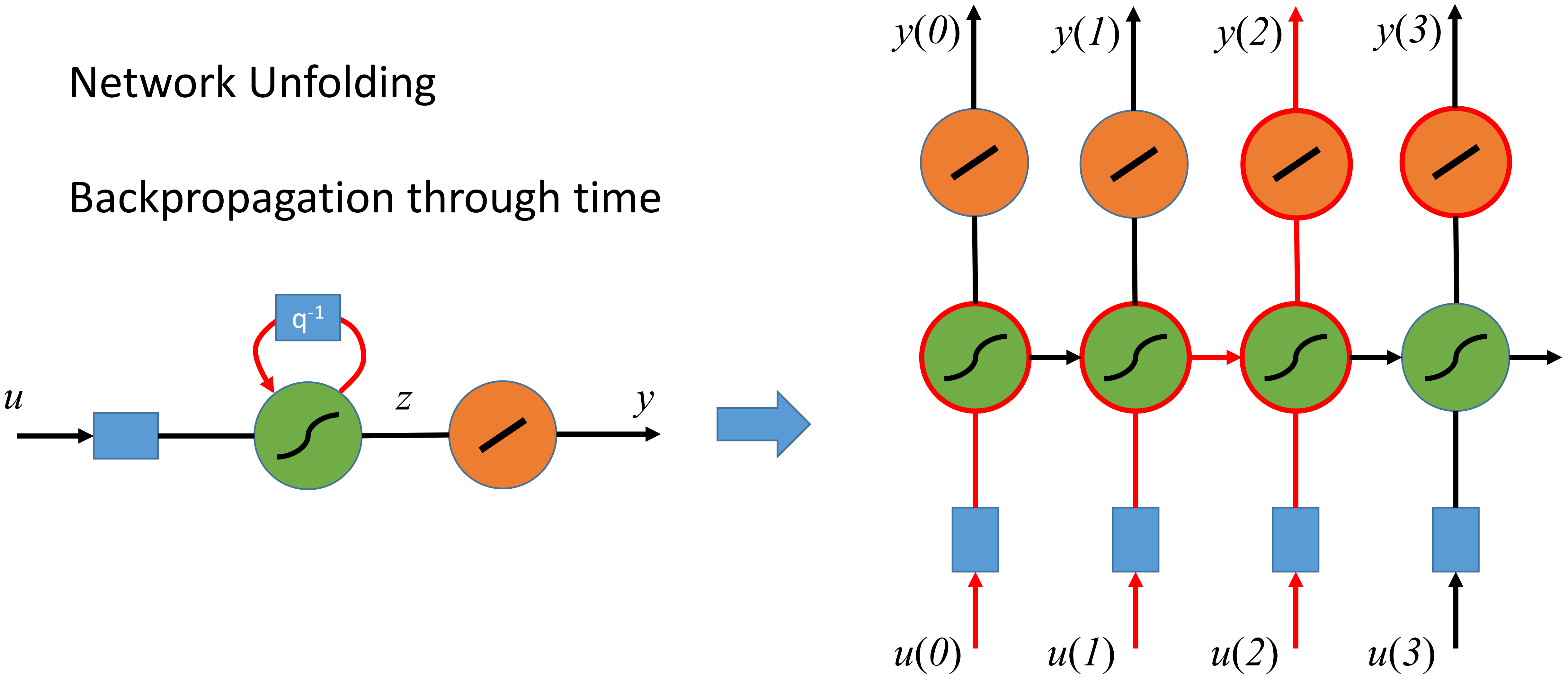
Introduces states in the network

The mapping from $\boldsymbol{u}$ to $\boldsymbol{y}$ is now dynamic

Recurrent Neural Networks - Training

Network Unfolding

Backpropagation through time



Long-Short Term Memory Networks

Extension of recurrent neural networks

For experiences with very long time lags

LSTM - neuron

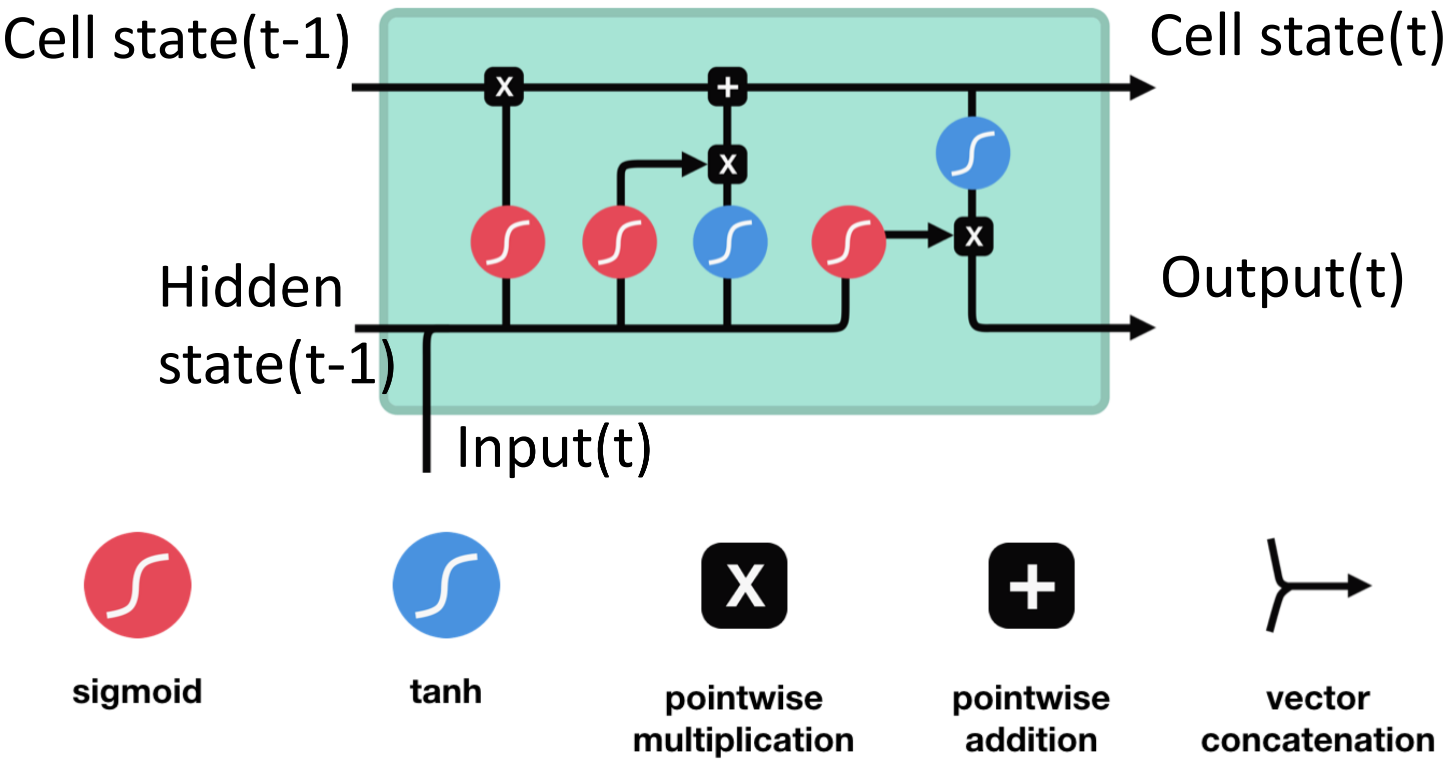


Figure Source: https://towardsdatascience.com/

Long-Short Term Memory Networks

Extension of recurrent neural networks

For experiences with very long time lags

Input gate:

Decide when to update memory

Output gate:

Decide when to output memory

Forget Gate:

Decide when to erase memory

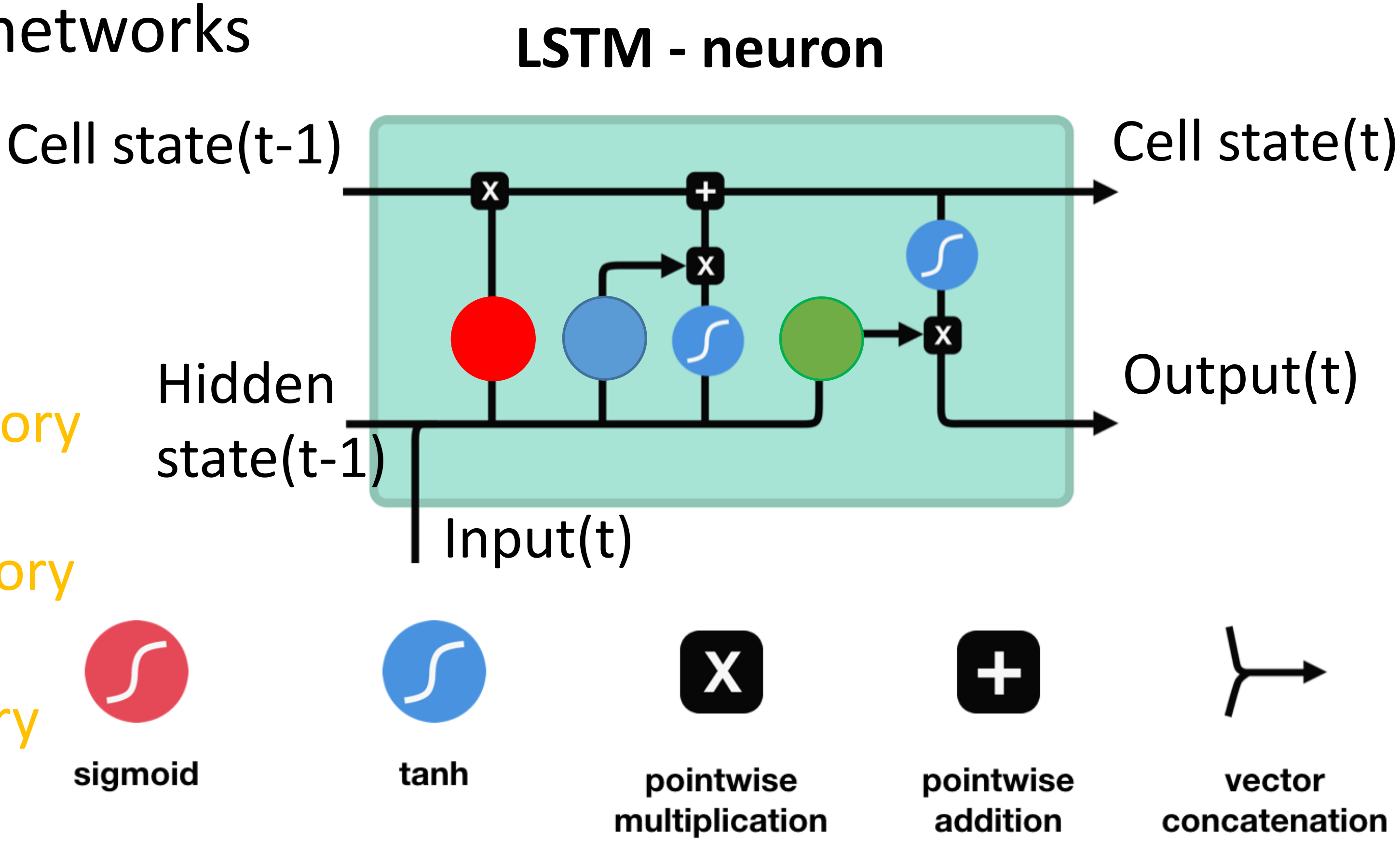


Figure Source: https://towardsdatascience.com/

Long-Short Term Memory Networks

Extension of recurrent neural networks

For experiences with very long time lags

Input gate:

Decide when to update memory

Output gate:

Decide when to output memory

Forget Gate:

Decide when to erase memory

Alternative: GRU

LSTM - neuron

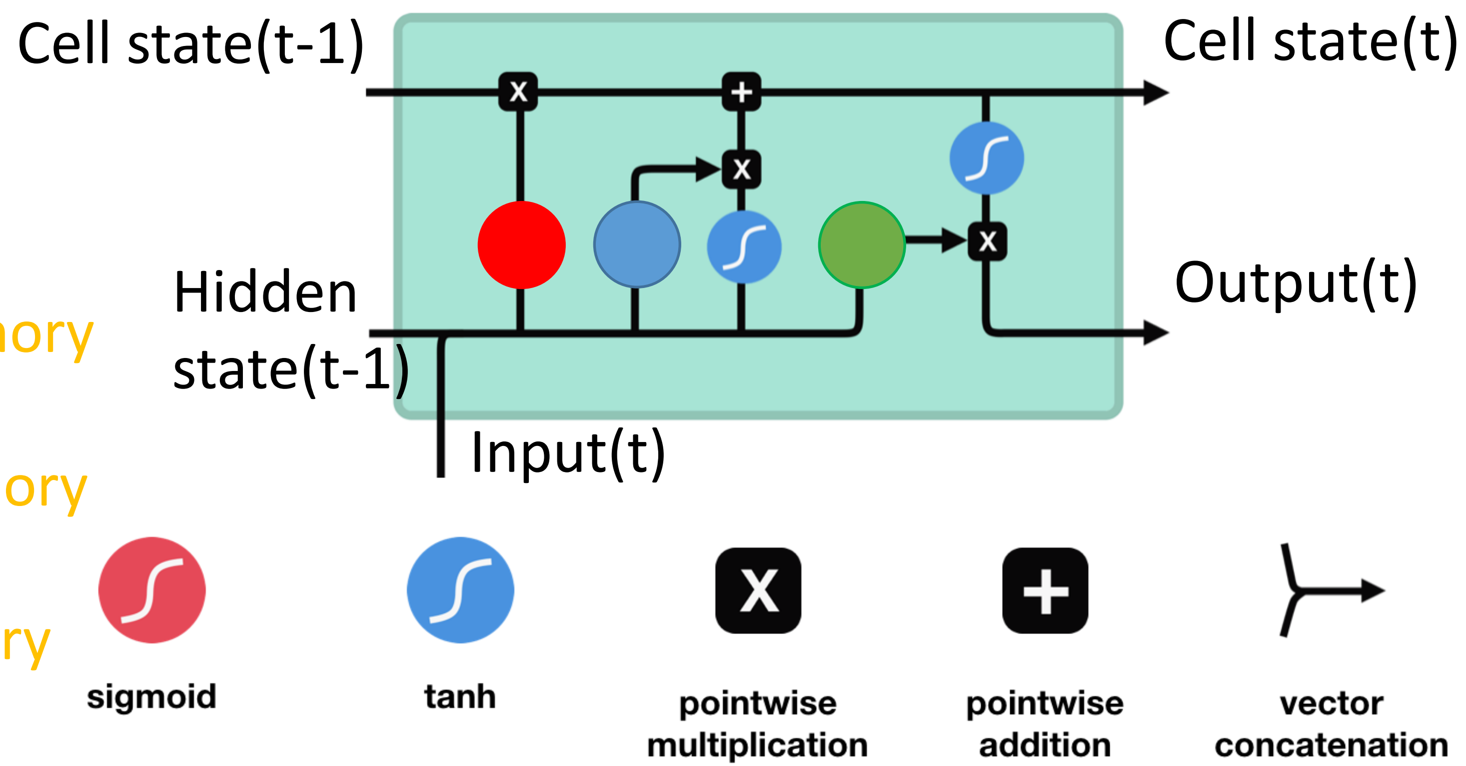
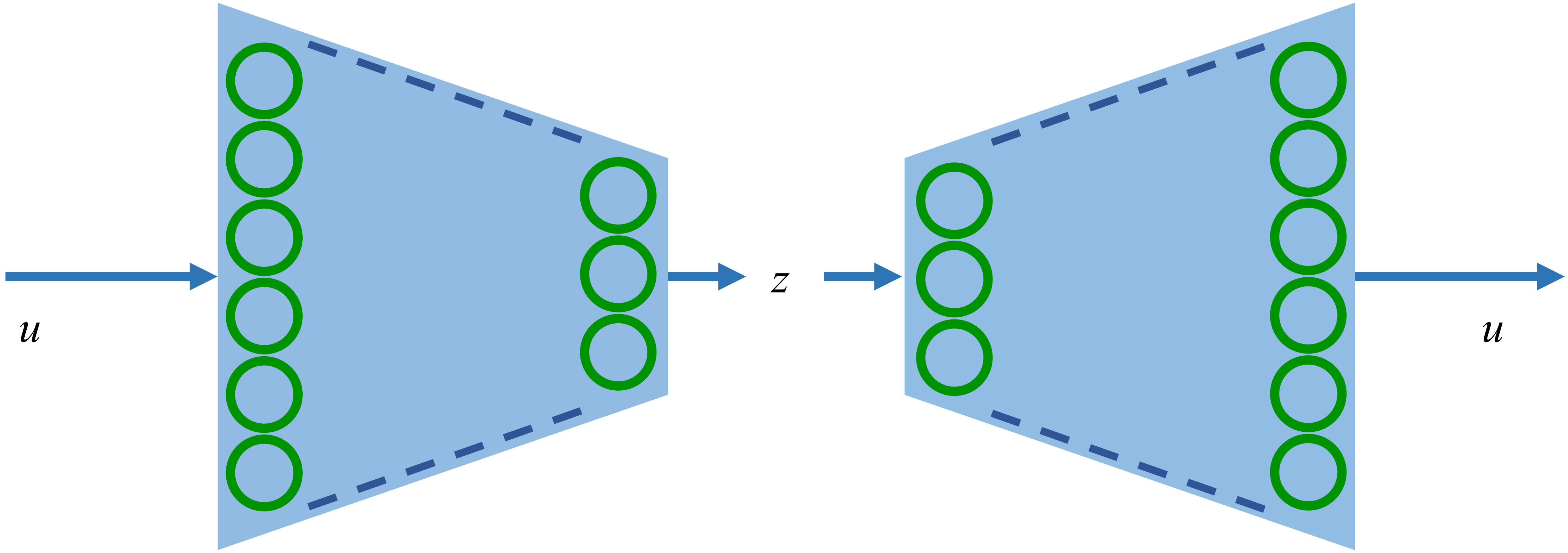


Figure Source: https://towardsdatascience.com/

Auto-Encoder Neural Networks

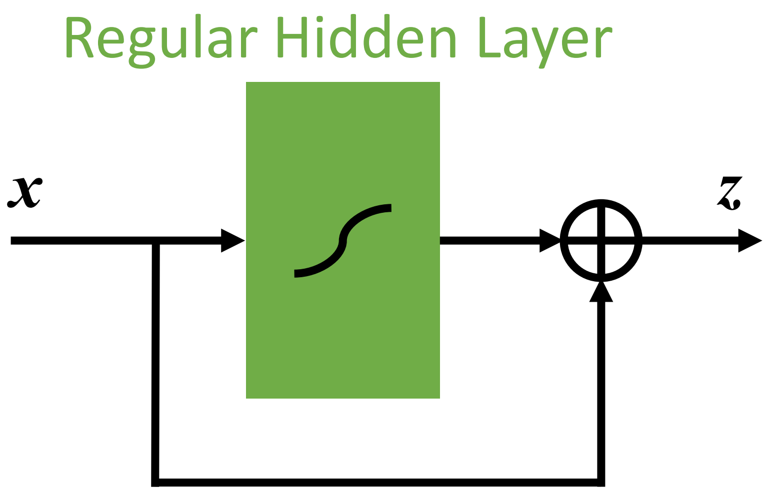


High-dimensional input space

Low-dimensional Learned features

Reconstructed input space

Residual Networks



Residual Network Layer

$$z = x + g\left(W_1 x + b_1\right)$$

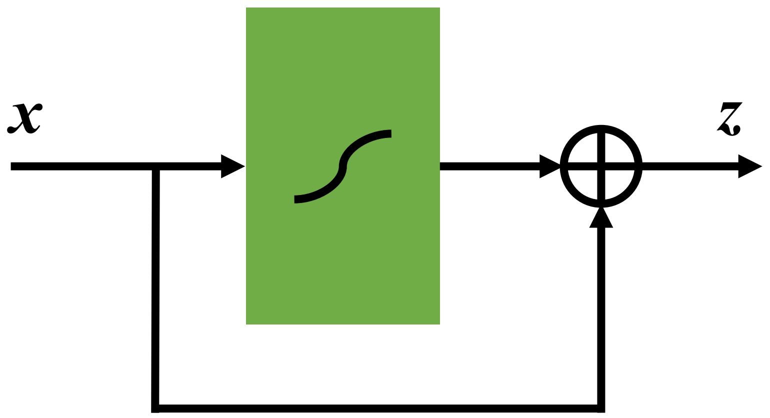
Improved training characteristics

Direct feedthrough can skip multiple layers

Dimension z = dimension x

Residual Networks: Changing Dimensions

Regular Hidden Layer



Residual Network Layer

$$\boldsymbol{z} = W_s \boldsymbol{x} + g\left(\boldsymbol{W}_1 \boldsymbol{x} + \boldsymbol{b}_1\right)$$

Improved training characteristics

Direct feedthrough can skip multiple layers

Dimension $\boldsymbol{z} \neq$ dimension $\boldsymbol{x}$

- Zero-padding for increasing dimensions
- Linear projection for changing output dimensions (e.g. by 1x1 convolution)

Convolutional Networks

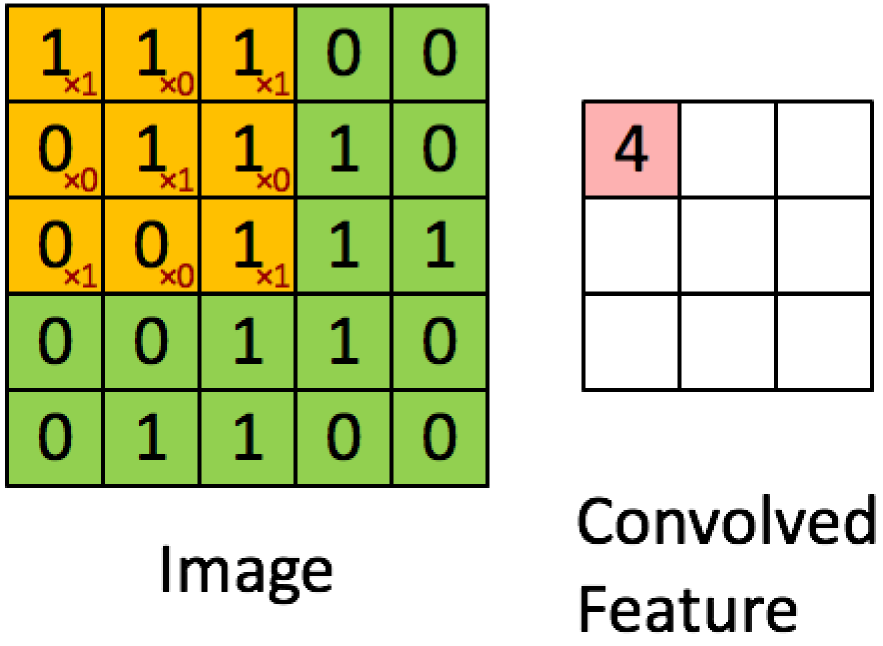
Convolve the input with a filter

Learn filter weights + bias

Spatial / Temporal relation of entries preserved
(as opposed to vectorizing the tensor / matrix)

Layer output decreases in dimension → perform padding to preserve the same dimension

Convolutional Operation



source: https://towardsdatascience.com/

Convolutional Networks – Pooling

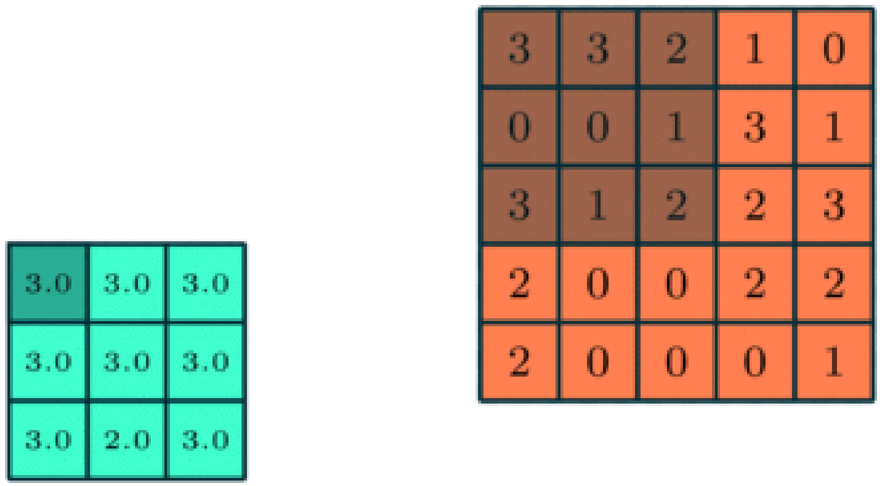
Dimension reduction

Noise suppression

Extract dominant features

Max or average pooling

Max Pooling



source: https://towardsdatascience.com/

Deep Neural Networks

Deep Feedforward	basic structure
Recurrent	time series, natural language processing
Long-Short Term Memory	time series, natural language processing (long dependencies)
Autoencoder	dimension reduction, feature learning
Residual	to go deep
Convolutional	image / video processing, spatial-temporal

…

Artificial Neural Networks

Deep Learning & Deep Neural Networks

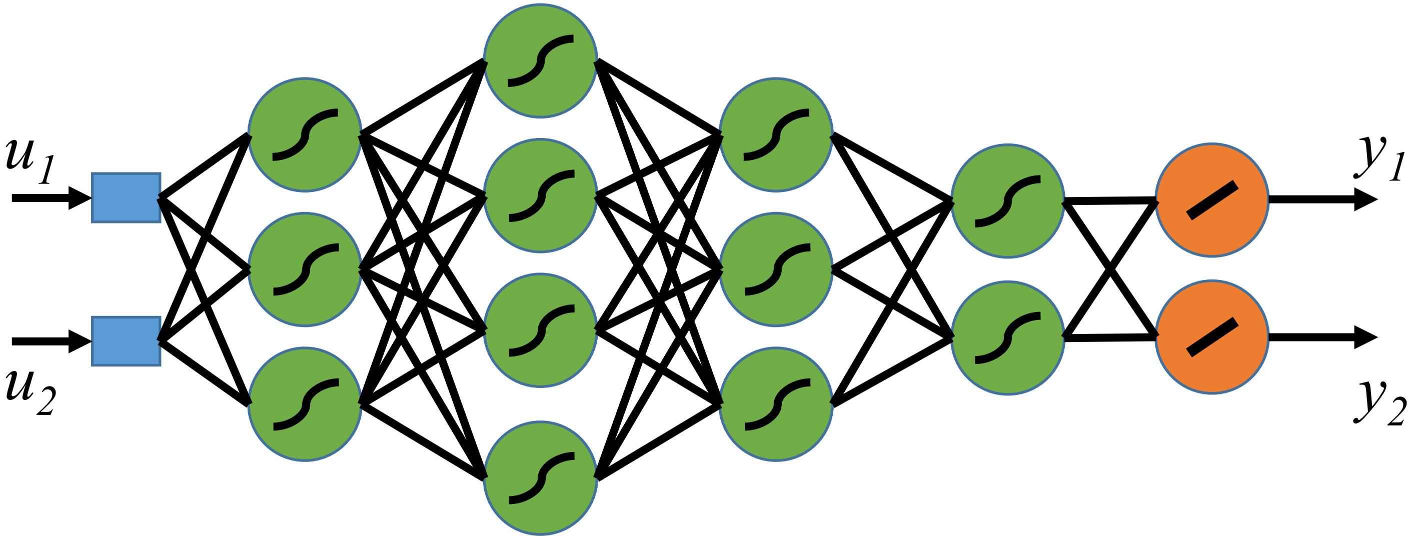
Training a Deep Neural Network

Artificial Neural Networks for Dynamical Systems

Deep Learning

Larger Networks

Big Data



➔ Difficult for Training: Computational Load
Vanishing Gradient
Overfitting

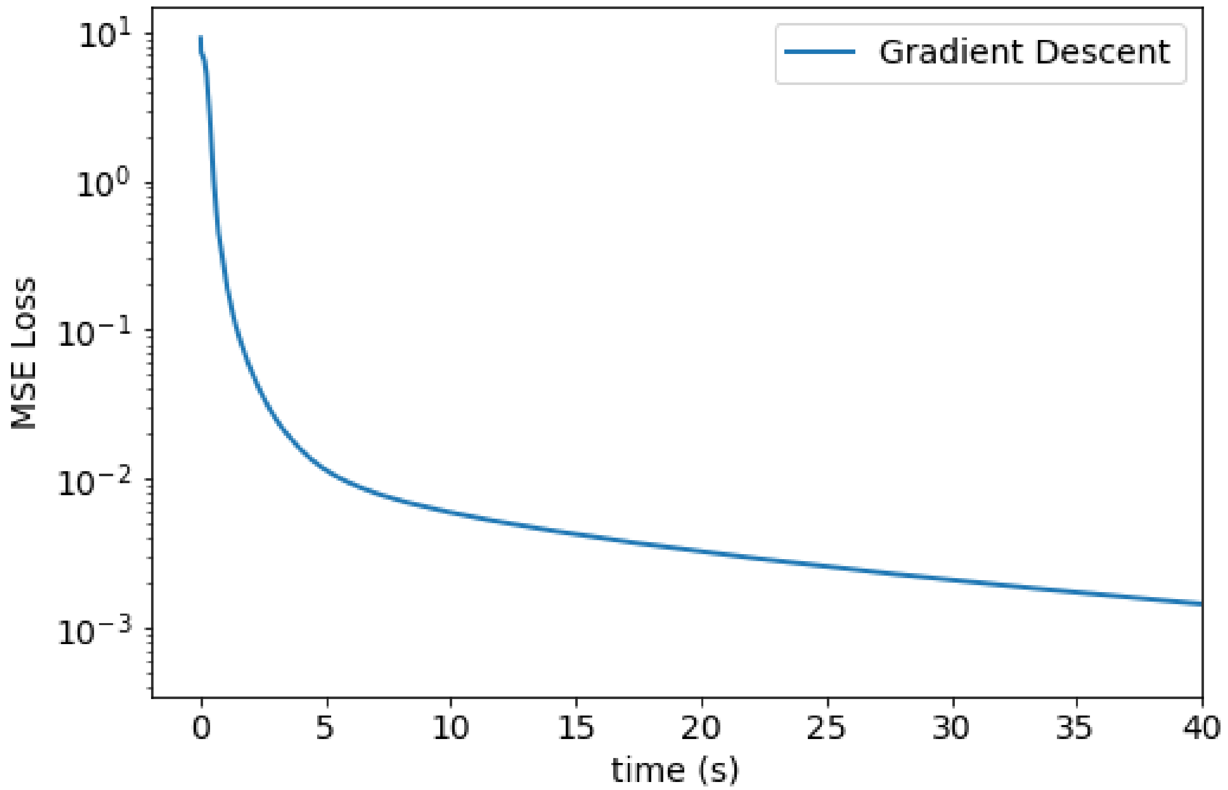
Stochastic Gradient Descent

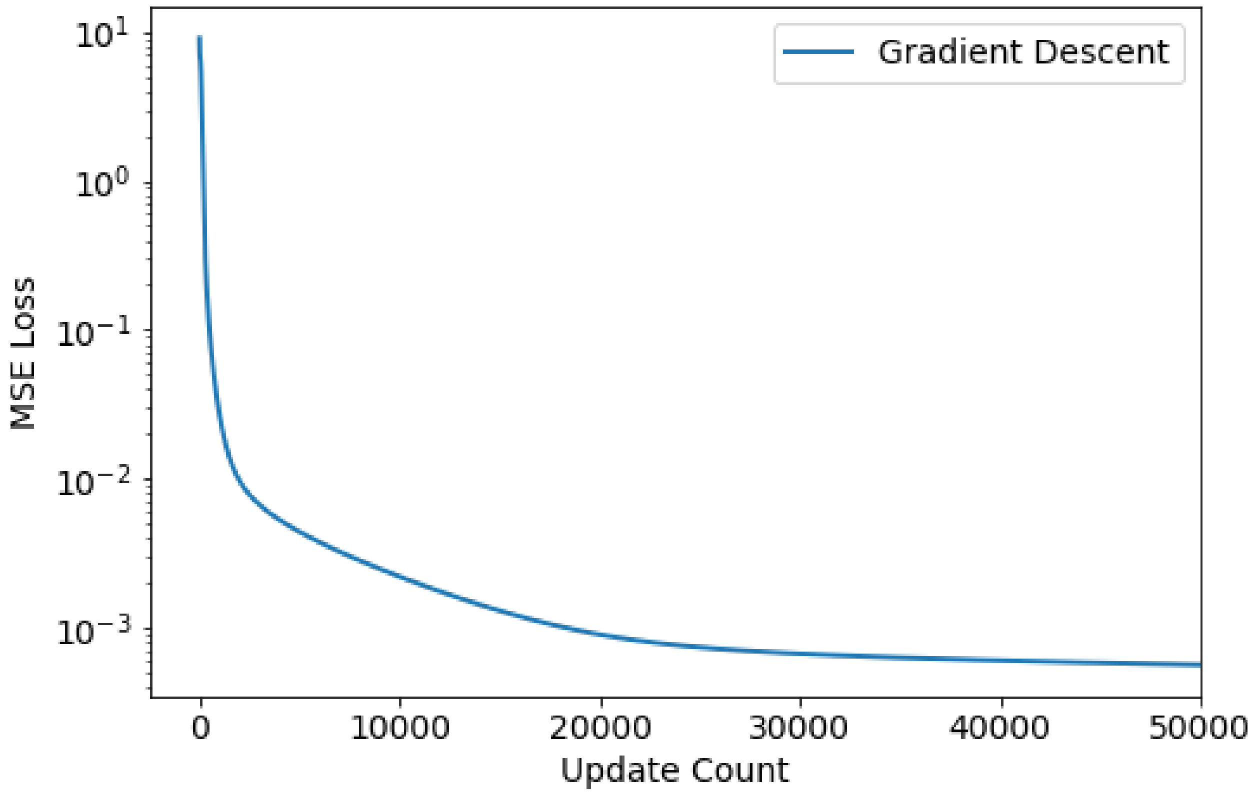
Idea: Do we need to use the full dataset to compute the gradient?

➔ Use mini-batches of data to compute the gradient

This results in stochastic behavior as the mini-batch is only and approximation of the full dataset

Stochastic Gradient Descent

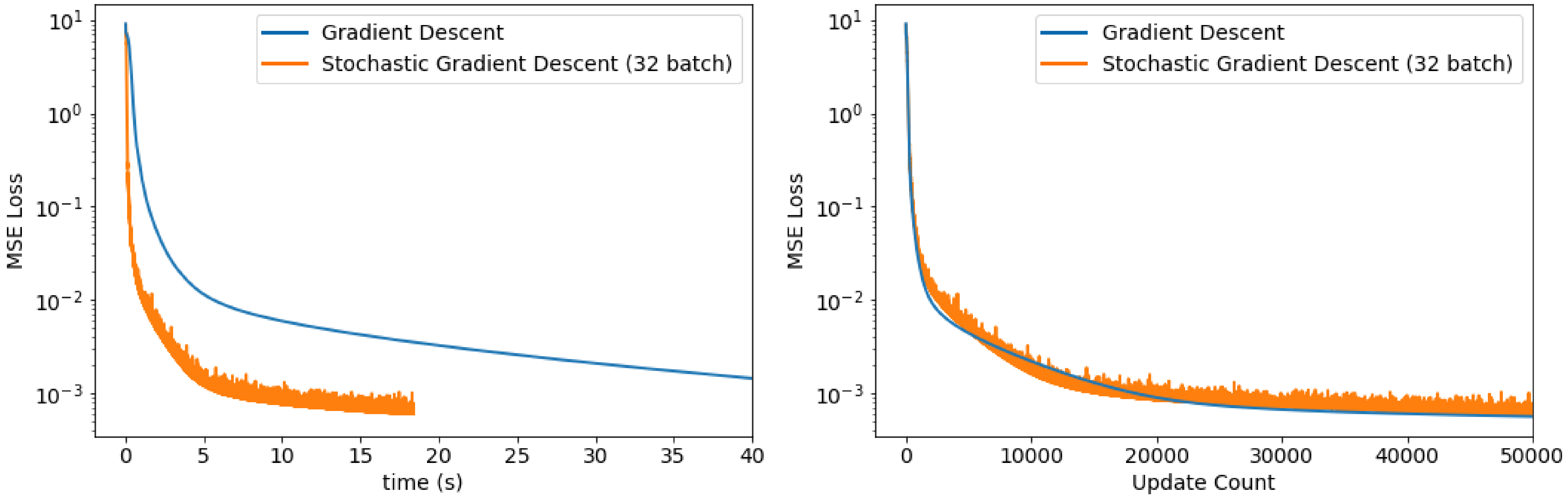




$y = u^2$

10^4 samples, 1 hidden layer, 15 neurons (sigmoid activation), 1 white Gaussian input – std = 1,
1 (noisy) output, std noise = 0.02, learning rate 0.05

Stochastic Gradient Descent



$y = u^2$

10^4 samples, 1 hidden layer, 15 neurons (sigmoid activation), 1 white Gaussian input – std = 1,
1 (noisy) output, std noise = 0.02, learning rate 0.05, mini-batch size = 32

Stochastic Gradient Descent - Convergence

If cost-function is (locally) convex, differentiable and Lipschitz-continuous gradient + diminishing learning rate

Then GD and SGD converge to the closest (local) minimum

GD: $f\left(\boldsymbol{x}^{(n)}\right) - f^\star = \mathcal{O}\left(\frac{1}{n}\right)$

SGD: $f\left(\boldsymbol{x}^{(n)}\right) - f^\star = \mathcal{O}\left(\frac{1}{\sqrt{n}}\right)$ (batch size = 1)

➔ SGD has (much) slower theoretical convergence

Stochastic Gradient Descent - Convergence

If cost-function is (locally) convex, differentiable and Lipschitz-continuous gradient + diminishing learning rate

Then GD and SGD converge to the closest (local) minimum

GD: $f\left(\boldsymbol{x}^{(n)}\right) - f^{\star} = \mathcal{O}\left(\frac{1}{n}\right)$

SGD: $f\left(\boldsymbol{x}^{(n)}\right) - f^{\star} = \mathcal{O}\left(\frac{1}{\sqrt{n}}\right)$ (batch size = 1)

In practice: batch SGD, with fixed learning rate

reduce in computational cost / iteration outweigh disadvantages

Stochastic Gradient Descent - Extensions

Momentum

Next parameter update is linear combination of current gradient and previous updates

Adaptive Gradient (AdaGrad)

Adaptive learning rate per parameter

Adaptive Moment estimation (AdaM)

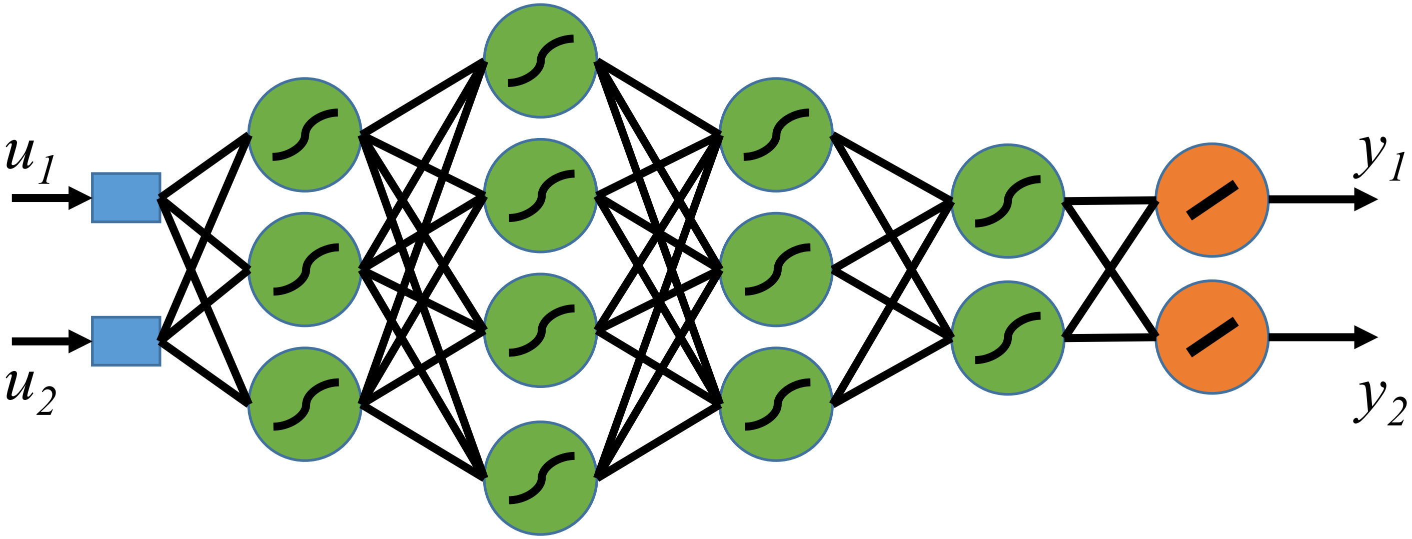
Uses a running average of the gradient and second moment of the gradient

….

Deep Learning

Larger Networks

Big Data



➔ Difficult for Training: Computational Load
Vanishing Gradient
Overfitting

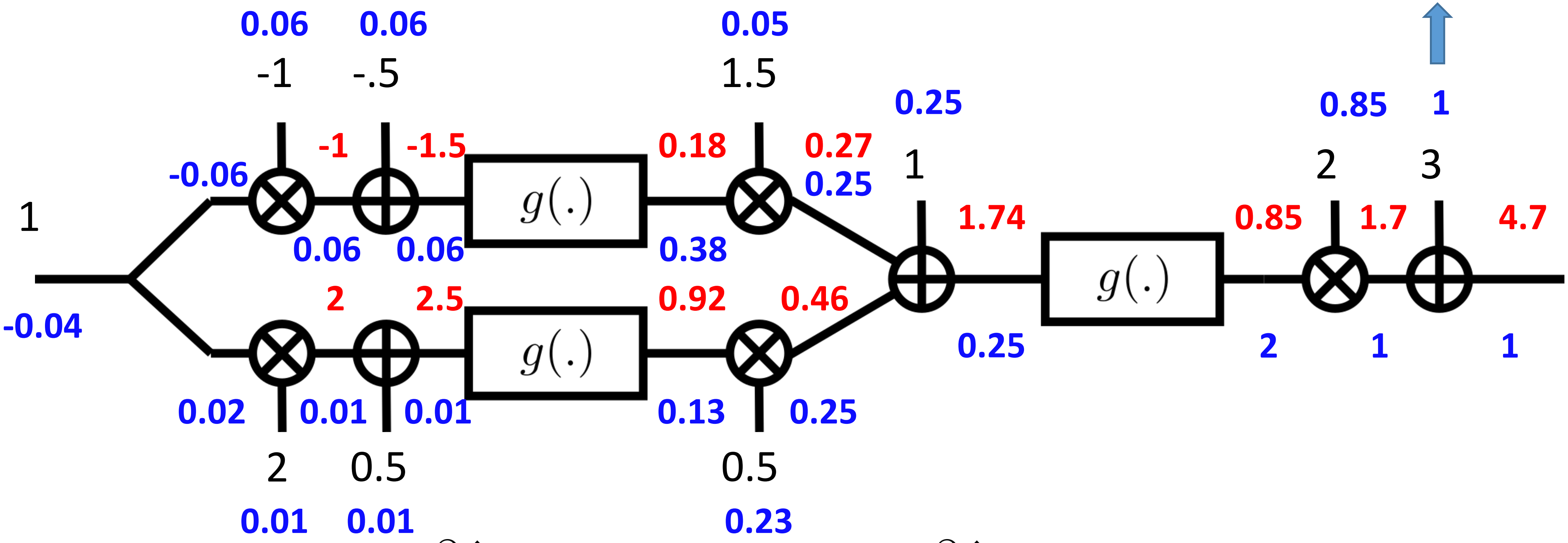
Vanishing Gradient

The gradient information becomes vanishing small when backpropagating backwards through the deep neural network

Prevents effective weight and bias training

Vanishing Gradient – Previous Lecture

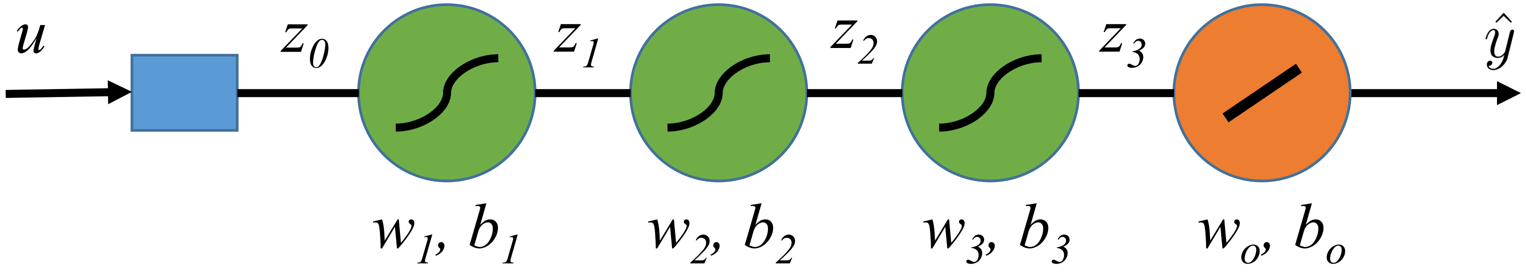
$$\frac{\partial \hat{y}_k}{\partial b_4} = 1$$



$$\frac{\partial \hat{y}_k}{\partial w_2} = 0.01$$

$$\frac{\partial \hat{y}_k}{\partial w_{32}} = 0.23$$

Vanishing Gradient

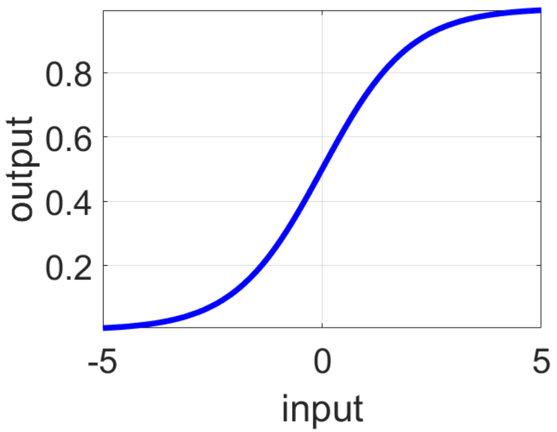


Typical activation function: sigmoid

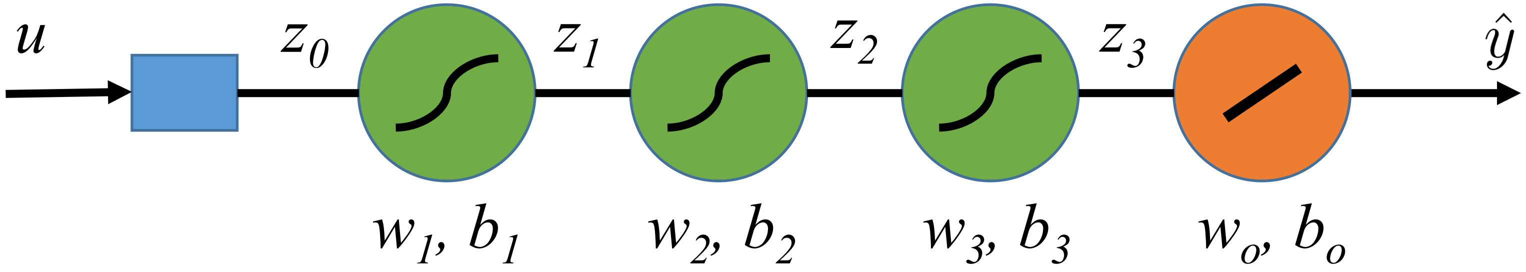
Initialization approach: (small) random weights

Sigmoid

$$g(x) = \frac{1}{1+e^{-x}}$$



Vanishing Gradient

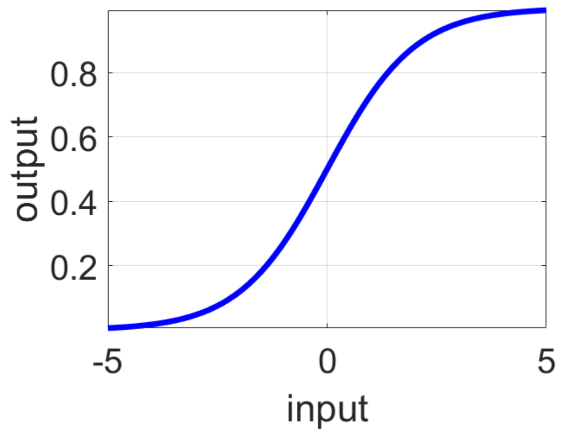


Typical activation function: sigmoid

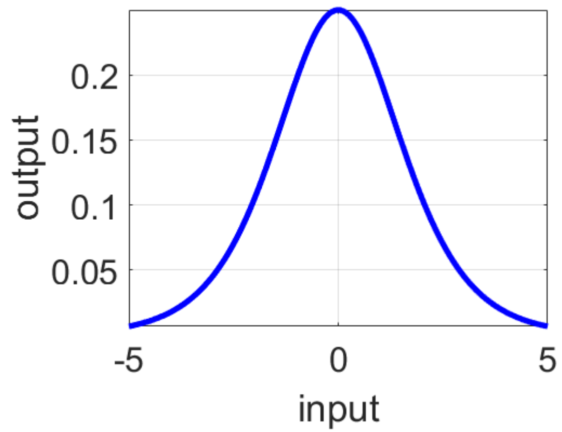
Initialization approach: (small) random weights

Sigmoid

$$g(x) = \frac{1}{1 + e^{-x}}$$

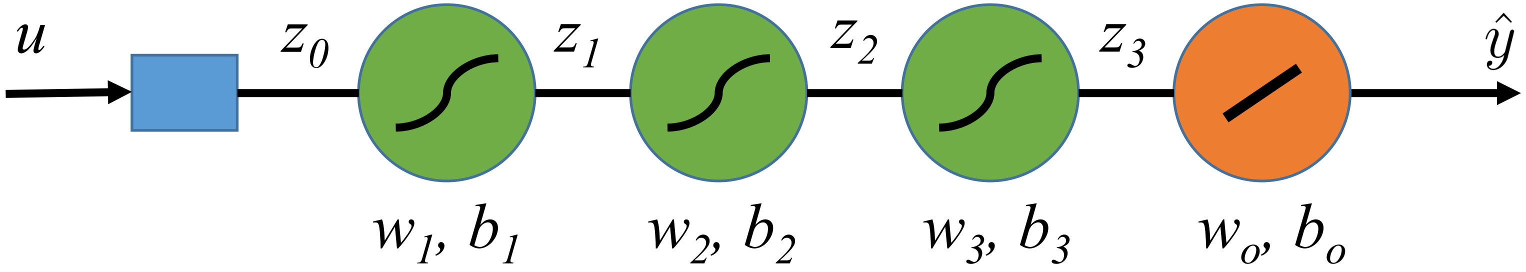


derivative



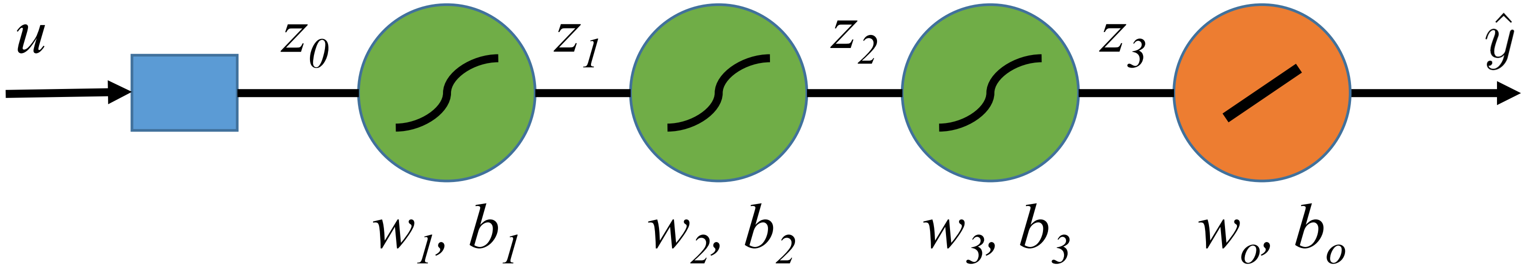
Max. of 0.25

Vanishing Gradient



$$\frac{\partial \hat{y}}{\partial b_3} = w_o \left.\frac{\partial g(x)}{\partial x}\right|_{x_3} w_3 \left.\frac{\partial g(x)}{\partial x}\right|_{x_2} w_2 \left.\frac{\partial g(x)}{\partial x}\right|_{x_1}$$

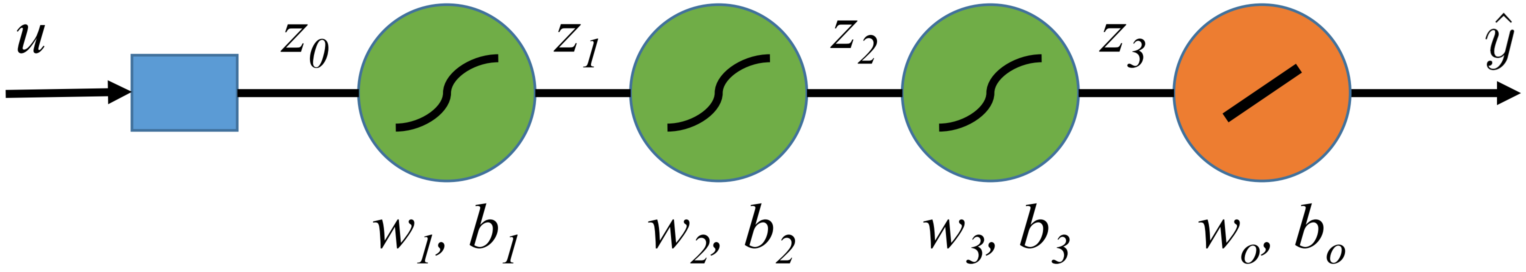
Vanishing Gradient



$$\frac{\partial \hat{y}}{\partial b_3} = w_o \underbrace{\left.\frac{\partial g(x)}{\partial x}\right|_{x_3}}_{\leq 0.25} w_3 \underbrace{\left.\frac{\partial g(x)}{\partial x}\right|_{x_2}}_{\leq 0.25} w_2 \underbrace{\left.\frac{\partial g(x)}{\partial x}\right|_{x_1}}_{\leq 0.25}$$

Initialized to small values

Vanishing Gradient

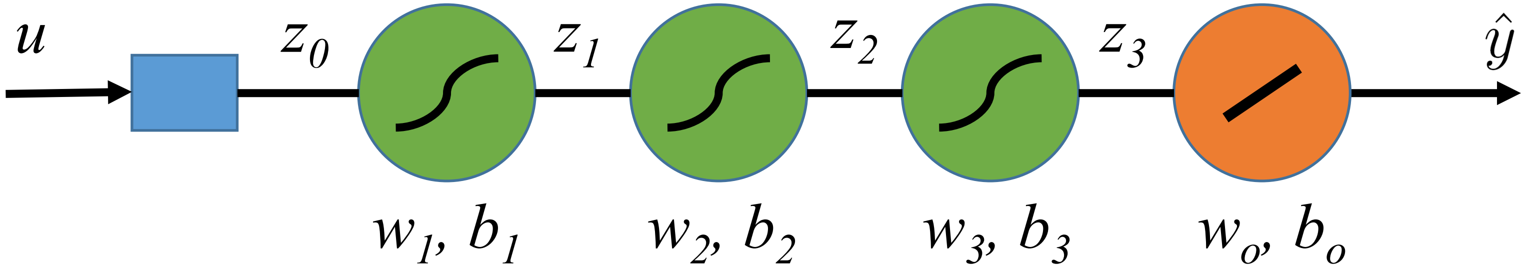


Gradient typically vanishes the further you propagate backwards

$$\frac{\partial \hat{y}}{\partial b_3} = w_o \underbrace{\left.\frac{\partial g(x)}{\partial x}\right|_{x_3}}_{\leq 0.25} w_3 \underbrace{\left.\frac{\partial g(x)}{\partial x}\right|_{x_2}}_{\leq 0.25} w_2 \underbrace{\left.\frac{\partial g(x)}{\partial x}\right|_{x_1}}_{\leq 0.25}$$

Initialized to small values

Exploding Gradient



Gradient can also explode if derivative x weights > 1

$$\frac{\partial \hat{y}}{\partial b_3} = w_o \left.\frac{\partial g(x)}{\partial x}\right|_{x_3} w_3 \left.\frac{\partial g(x)}{\partial x}\right|_{x_2} w_2 \left.\frac{\partial g(x)}{\partial x}\right|_{x_1}$$

Especially problematic for recurrent neural networks

Vanishing / Exploding Gradient: Solutions

Change activation function

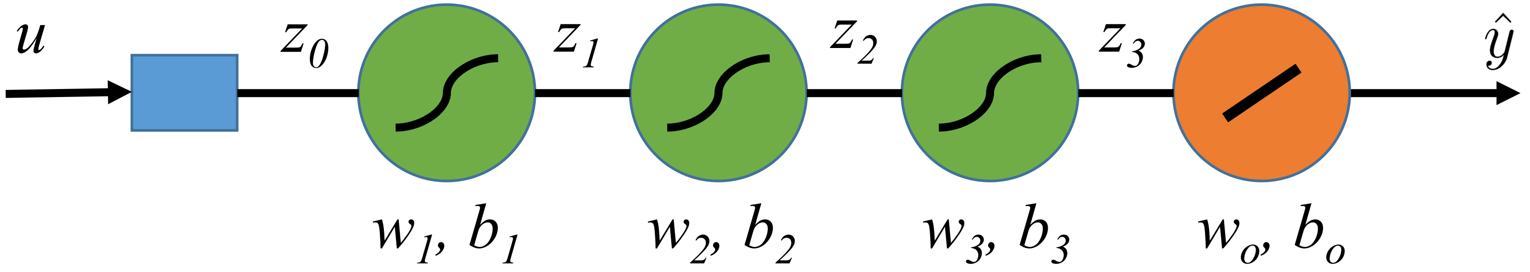
Data Normalization

Change network structure

Better Initialization

Faster Hardware

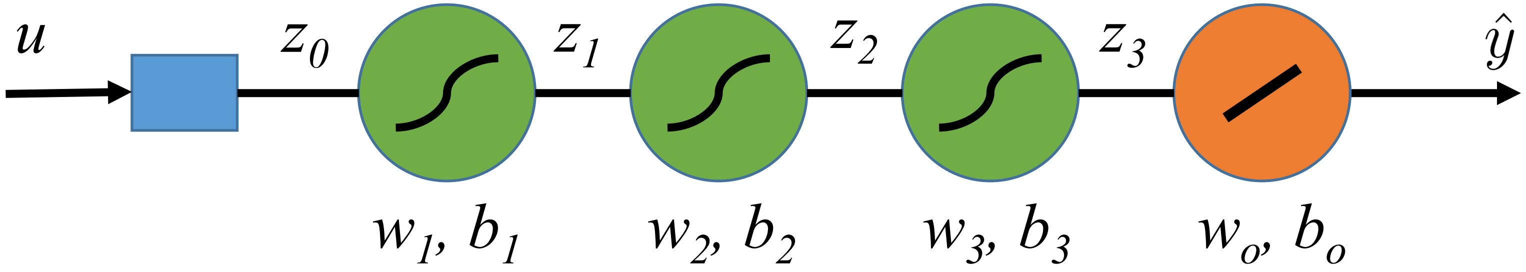
Changing the Activation Function



Gradient propagates well if derivative x weights ≈ 1

$$\frac{\partial \hat{y}}{\partial b_3} = w_o \left.\frac{\partial g(x)}{\partial x}\right|_{x_3} w_3 \left.\frac{\partial g(x)}{\partial x}\right|_{x_2} w_2 \left.\frac{\partial g(x)}{\partial x}\right|_{x_1}$$

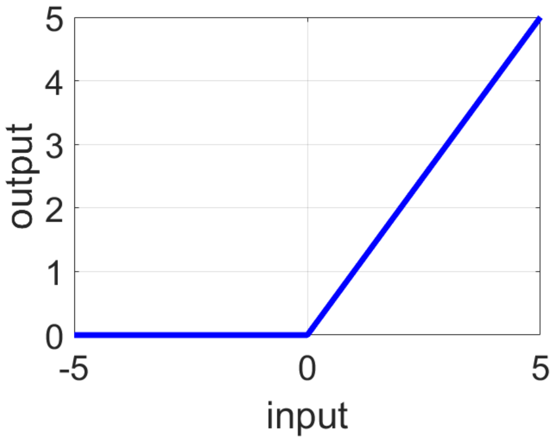
Changing the Activation Function



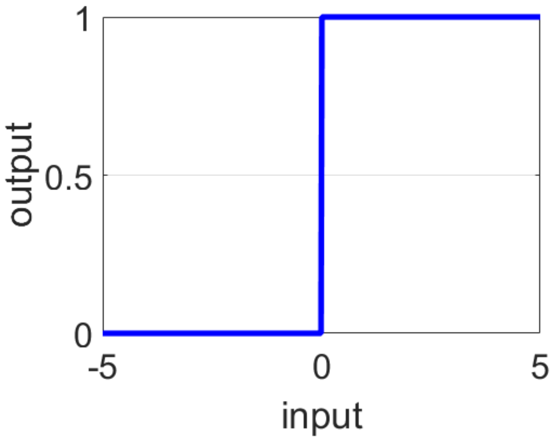
Gradient propagates well if derivative x weights ≈ 1

ReLu

$$g(x) = \begin{cases} 0 & \forall\, x < 0 \\ x & \forall\, x \geq 0 \end{cases}$$

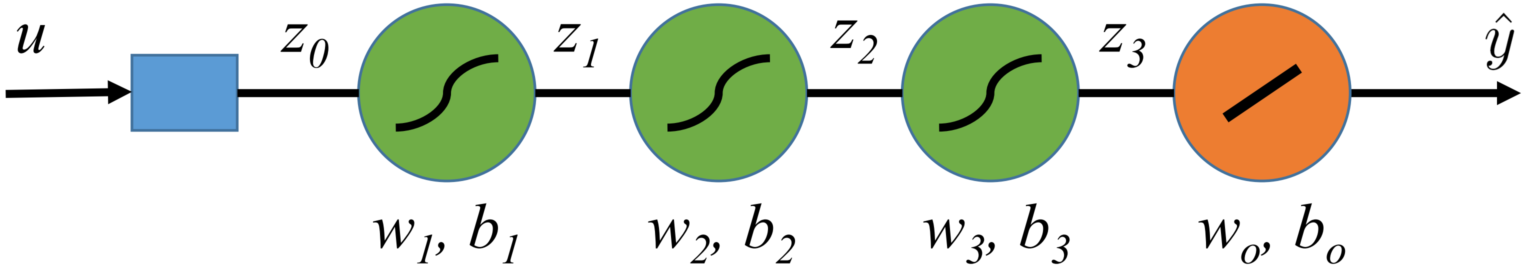


derivative



Max. of 1

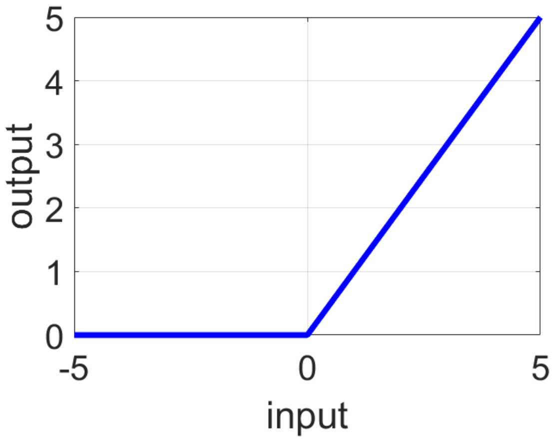
Changing the Activation Function



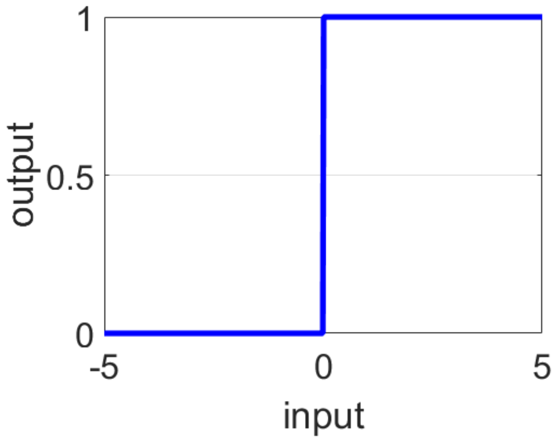
ReLu (and similar forms) became one of the most popular choices

ReLu

$$g(x) = \begin{cases} 0 & \forall\, x < 0 \\ x & \forall\, x \geq 0 \end{cases}$$



derivative



Max. of 1

Vanishing / Exploding Gradient: Solutions

Change activation function

Data Normalization

Change network structure

Better Initialization

Faster Hardware

Data Normalization

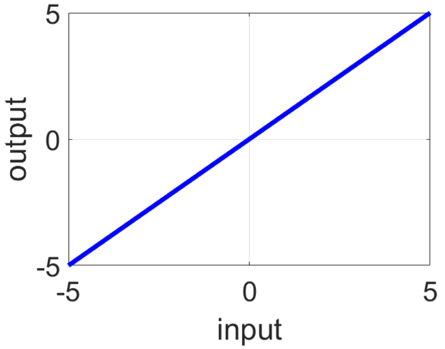
Always normalize the input-output data before training

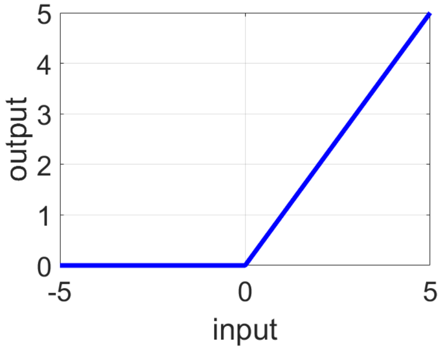
zero-mean

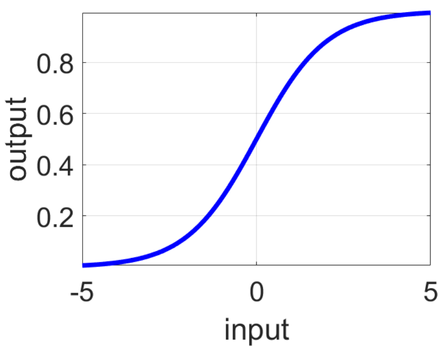
unit variance

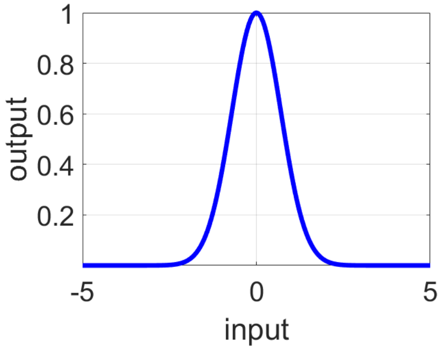
Improves the conditioning of the learning problem

Lowers risk of vanishing gradient problem since the full activation function range is used



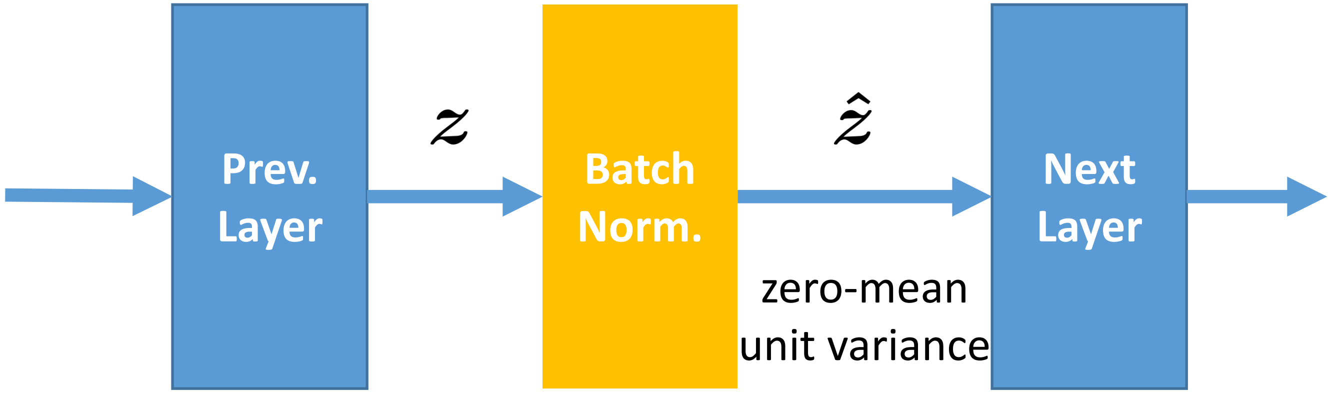






Batch Normalization Layers

Add a normalization layer in the neural network architecture



$$\tilde{\boldsymbol{z}} = \boldsymbol{z} - \frac{1}{n}\sum_{k=1}^{n} \boldsymbol{z}_k$$

n → batch size

$$\hat{\boldsymbol{z}} = \frac{\tilde{\boldsymbol{z}}}{\sqrt{\epsilon + \frac{1}{n}\sum_{k=1}^{n} \tilde{\boldsymbol{z}}_k^2}}$$

First order statistics (mean, variance) are always the same, independent from the previous layers. Previous layers only affect the higher order statistics.

Vanishing / Exploding Gradient: Solutions

Change activation function

Data Normalization

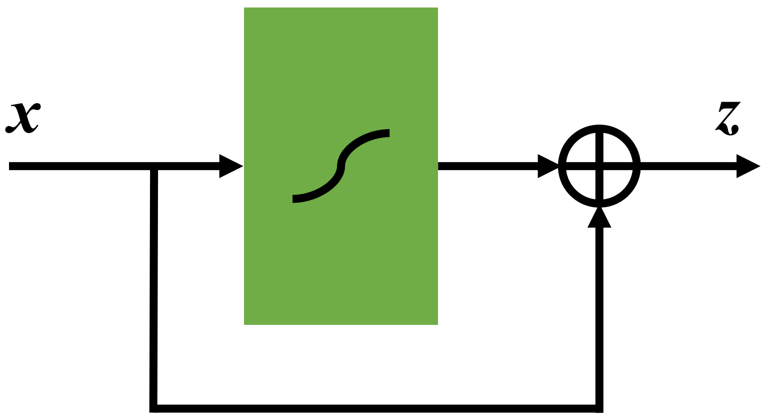
Change network structure

Better Initialization

Faster Hardware

Change Network Structure

Smart network structure allow for a better backpropagation of the gradient



Residual Network Layer

$$\boldsymbol{z} = \boldsymbol{x} + g\left(\boldsymbol{W}_1\boldsymbol{x} + \boldsymbol{b}_1\right)$$

$$\boldsymbol{z} = f(\boldsymbol{x}) = \boldsymbol{x} + g\left(\boldsymbol{W}_1\boldsymbol{x} + \boldsymbol{b}_1\right)$$

$$\frac{\partial f(\boldsymbol{x})}{\partial \boldsymbol{x}} = 1 + \frac{\partial g(\boldsymbol{x})}{\partial \boldsymbol{x}}$$

Ensures values close to 1

Vanishing / Exploding Gradient: Solutions

Change activation function

Data Normalization

Change network structure

Better Initialization

Faster Hardware

Better Initialization

Vanishing Gradients → Slower Learning

I. Better initialization leads you faster to the (local) optimum. The learning time is reduced.
II. A smart initialization can avoid regions of vanishing / exploding gradients.

This only overcomes the vanishing gradient problem; it doesn't solve the root cause of it.

Vanishing / Exploding Gradient: Solutions

Change activation function

Data Normalization

Change network structure

Better Initialization

Faster Hardware

Faster Hardware

Vanishing Gradients → Slower Learning

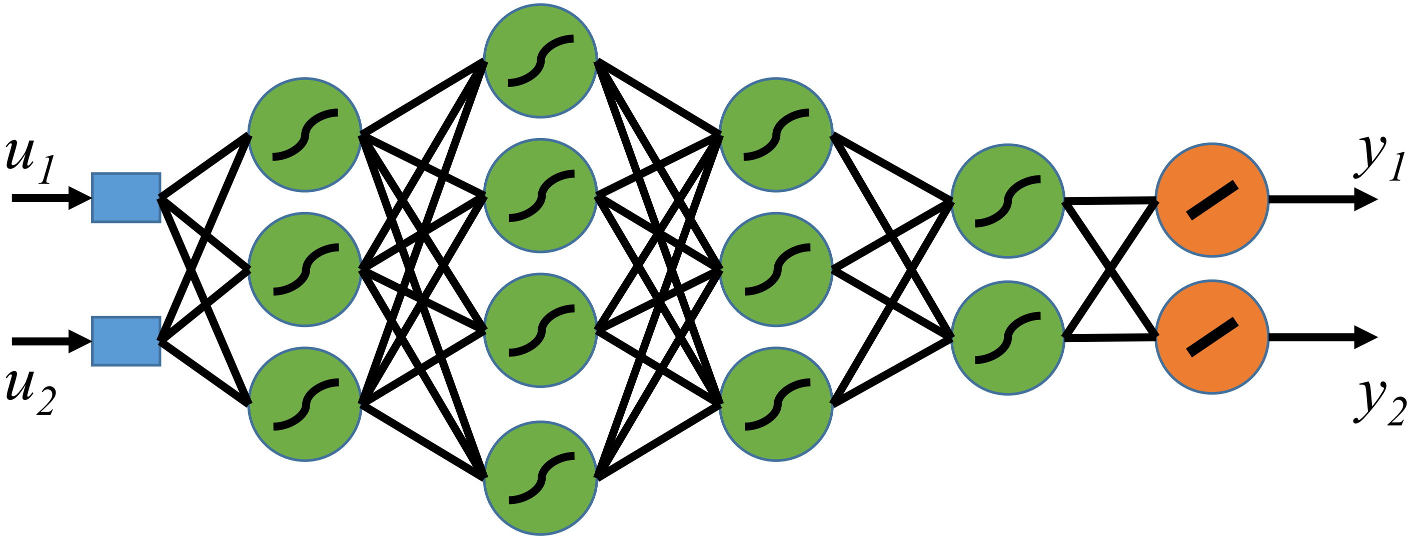
The development of faster and parallelized hardware has been one of the major drivers in the development of deep learning.

This only overcomes the vanishing gradient problem; it doesn’t solve the root cause of it.

Deep Learning

Larger Networks

Big Data



➔ Difficult for Training: Computational Load
Vanishing Gradient
Overfitting

Overfitting & Regularization

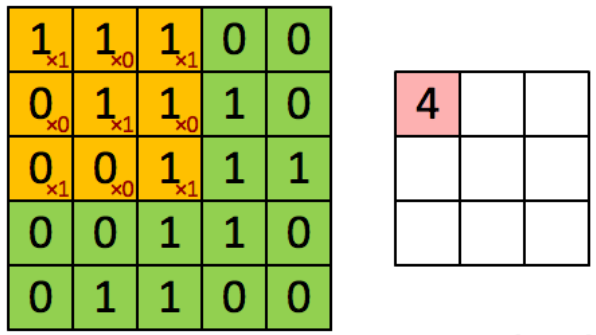
Parameter Norm Regularization	(previous lecture)
Early Stopping	(previous lecture)
Parameter Sharing	
Data Augmentation	
Noise Robustness	
Sparse Representations	

Parameter Sharing

Share parameters over multiply neurons / layers

Often used in Convolutional Networks or multi-task learning problems

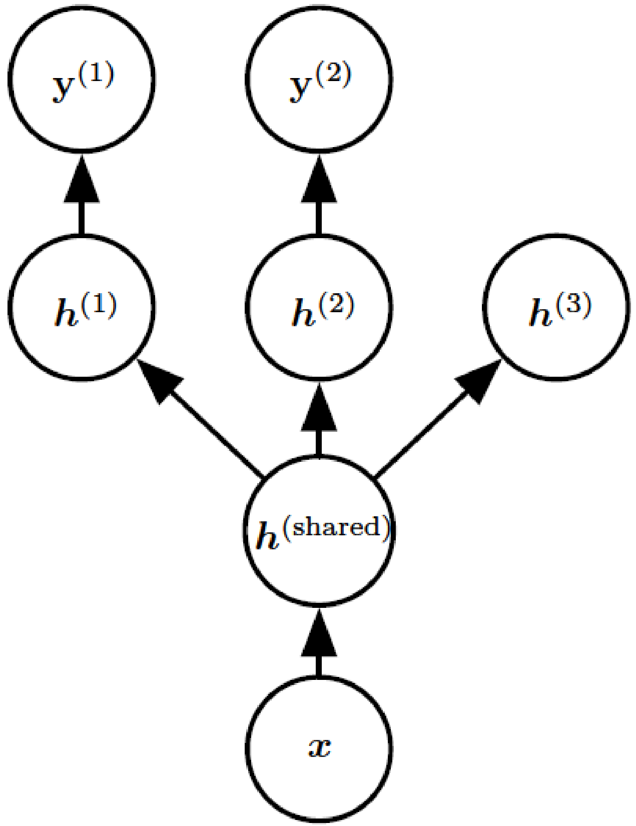
Convolutional Operation



Image

Convolved Feature

source: https://towardsdatascience.com/



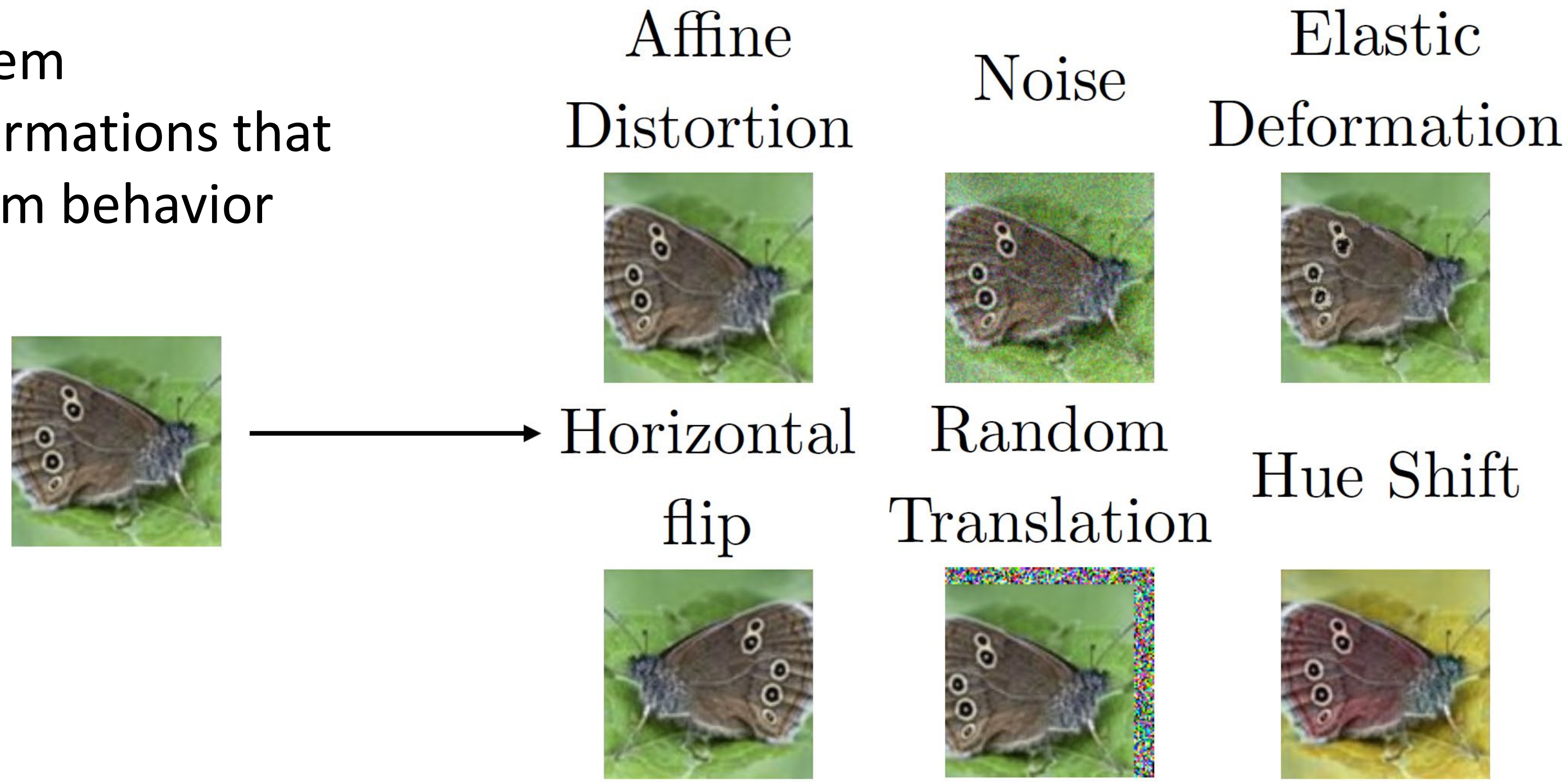
Multi-Task Learning

e.g. multiple systems / outputs with shared dynamics

source: www.deeplearningbook.org

Data Augmentation

Make use of data / system symmetries and transformations that do not impact the system behavior



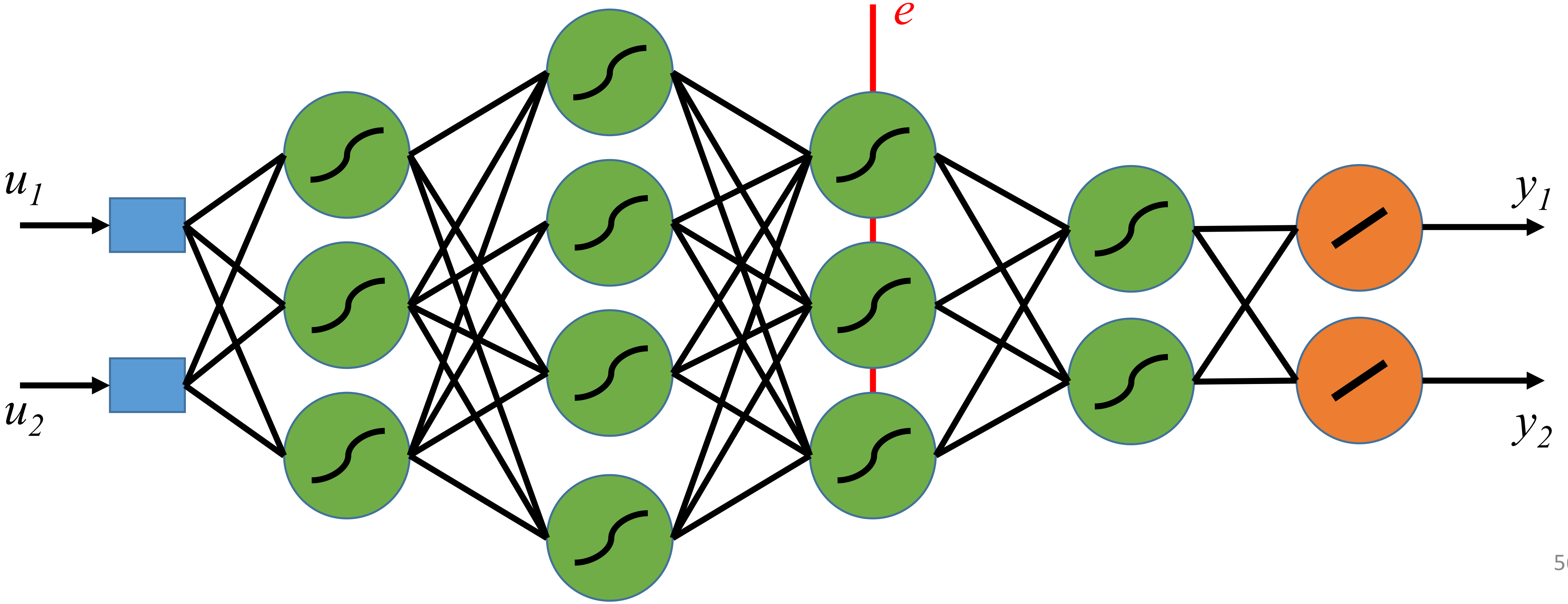
source: www.deeplearningbook.org

Noise Robustness

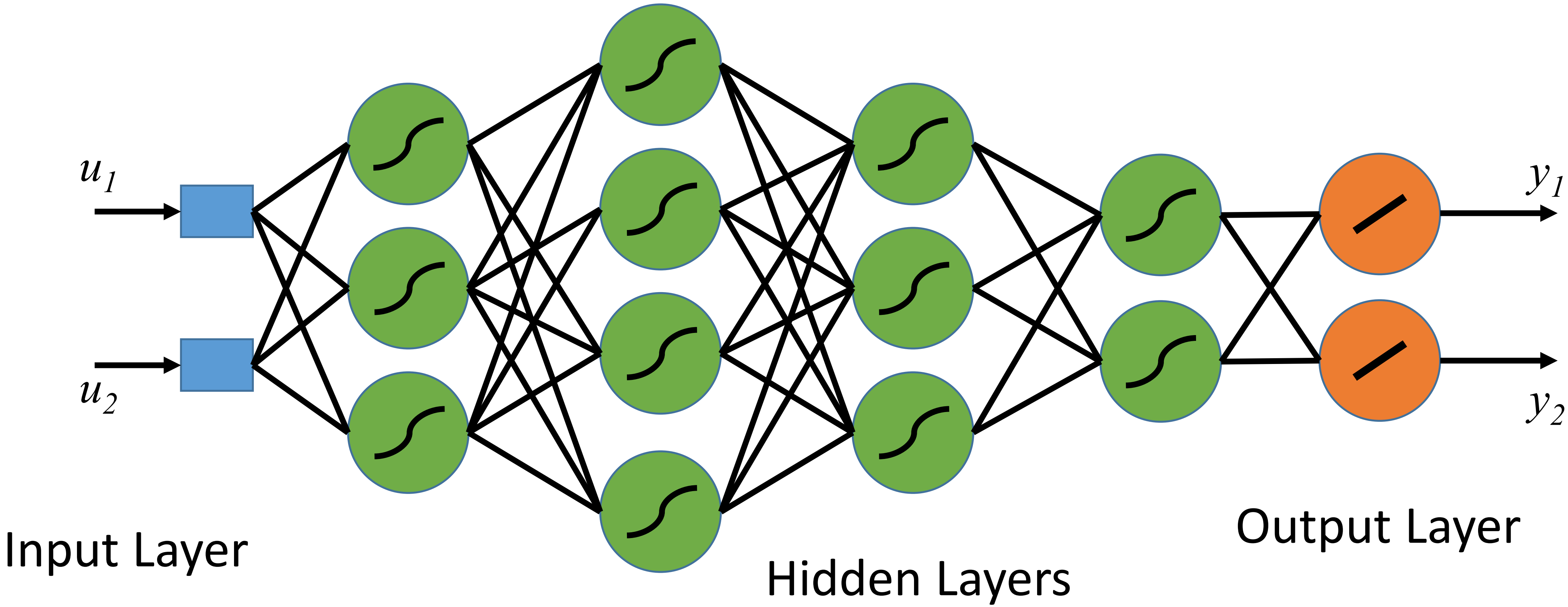
Add noise to the (hidden) layers

Increases robustness at cost of bias introduction

Links with L^2-regularization of parameters

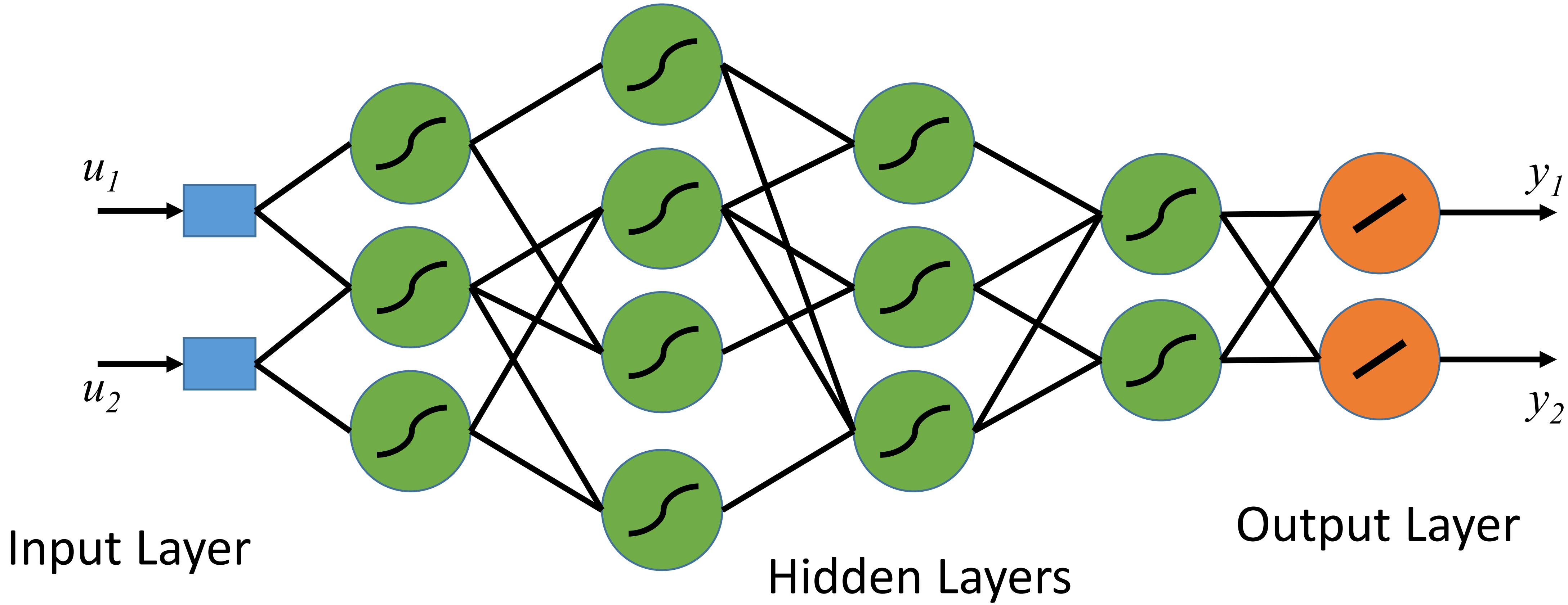


Sparse Connections



Sparse Connections

Reduces the number of weights to be trained
e.g. L^1-norm regularization of the weights



Sparse Representations

Instead of having sparse weights, obtain a sparse representation (i.e. sparse signals)
e.g. L^1-norm regularization of the representation

$$\begin{bmatrix} 18 \\ 5 \\ 15 \\ -9 \\ -3 \end{bmatrix} = \begin{bmatrix} 4 & 0 & 0 & -2 & 0 & 0 \\ 0 & 0 & -1 & 0 & 3 & 0 \\ 0 & 5 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & -1 & 0 & -4 \\ 1 & 0 & 0 & 0 & -5 & 0 \end{bmatrix} \begin{bmatrix} 2 \\ 3 \\ -2 \\ -5 \\ 1 \\ 4 \end{bmatrix}$$

$\boldsymbol{y} \in \mathbb{R}^m$ $\quad$ $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ $\quad$ $\boldsymbol{x} \in \mathbb{R}^n$

sparse weights / connections

$$\begin{bmatrix} -14 \\ 1 \\ 19 \\ 2 \\ 23 \end{bmatrix} = \begin{bmatrix} 3 & -1 & 2 & -5 & 4 & 1 \\ 4 & 2 & -3 & -1 & 1 & 3 \\ -1 & 5 & 4 & 2 & -3 & -2 \\ 3 & 1 & 2 & -3 & 0 & -3 \\ -5 & 4 & -2 & 2 & -5 & -1 \end{bmatrix} \begin{bmatrix} 0 \\ 2 \\ 0 \\ 0 \\ -3 \\ 0 \end{bmatrix}$$

$\boldsymbol{y} \in \mathbb{R}^m$ $\quad$ $\boldsymbol{B} \in \mathbb{R}^{m \times n}$ $\quad$ $\boldsymbol{h} \in \mathbb{R}^n$

sparse representation

source: www.deeplearningbook.org

Many other methods

Bagging

model output = mean of an ensemble of trained network structures

Dropout

optimize over an ensemble of network structures

Adversarial Training

training on adversarial perturbed examples from the training set

Artificial Neural Networks

Deep Learning & Deep Neural Networks

Training a Deep Neural Network

Artificial Neural Networks for Dynamical Systems

Model Structures

NARX

NOE

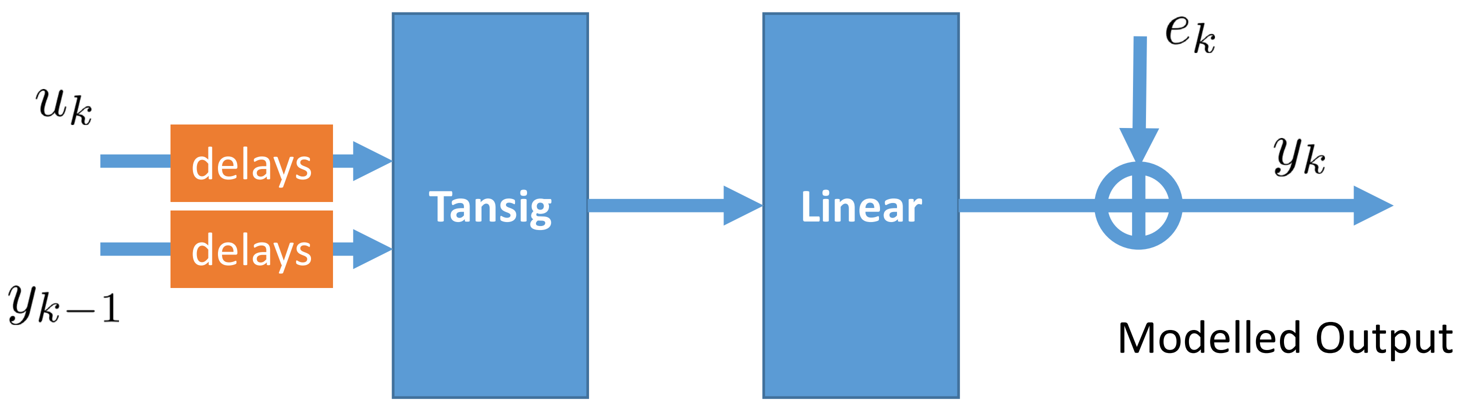
Recurrent ANN

Recursive State-Space

Unwrapped State-Space

Feedforward NN - NARX

Feedforward NN with delayed inputs and delayed measured outputs (NARX)



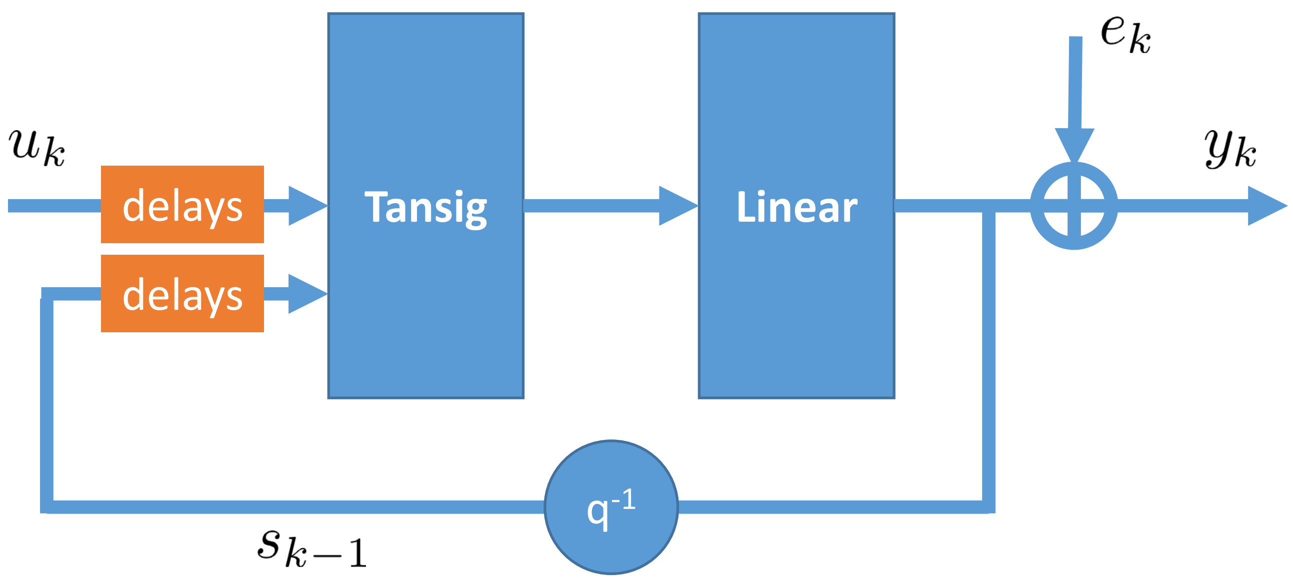
Delayed Measured Output

Nonlinear Auto Regressive with eXogeneous Input (NARX)

Very particular noise structure
Not always easy to analyze

$$y_k = f\left(u_k, u_{k-1}, \ldots, u_{k-n_b}, y_{k-1}, \ldots, y_{k-n_a}\right) + e_k$$

Recurrent NN - NOE



Loop output back

Nonlinear Output Error (NOE)

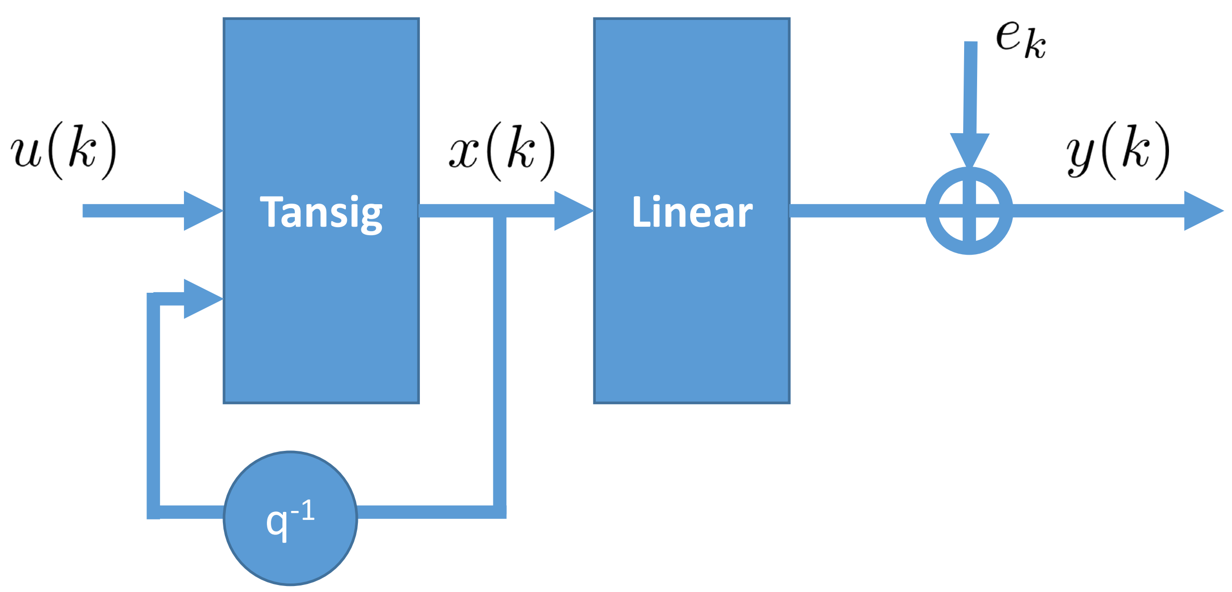
Similar to NARX, but different noise handling

Output depends on past unknown noiseless outputs
Difficult to analyze
Difficult to estimate
Straightforward noise structure

$$s_k = f\left(u_k, u_{k-1}, \ldots, u_{k-n_b}, s_{k-1}, \ldots, s_{k-n_a}\right)$$
$$y_k = s_k + e_k$$

Recurrent NN – State-Space

Recurrent NN can be interpreted as a state-space representation



Loop over one layer

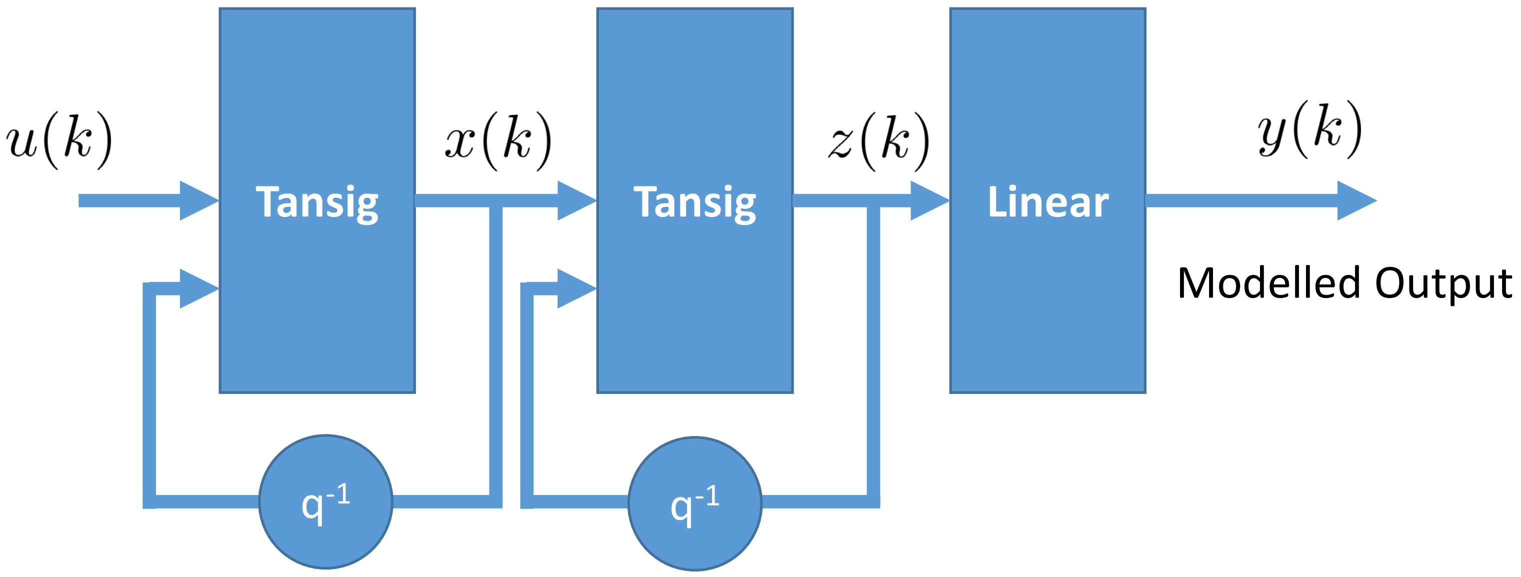
$$x_k = f(x_{k-1}, u_k)$$
$$y_k = Cx_k$$

$x(k)$ are states of the model

As many states as we have neurons in the hidden layer

Recurrent NN – State-Space

Recurrent NN can be interpreted as a state-space representation



Loop over one layer

$$x_k = f(x_{k-1}, u_k)$$
$$z_k = g(z_{k-1}, x_k)$$
$$y_k = Cz_k$$

$$x_k = f(x_{k-1}, u_k)$$
$$z_k = g(z_{k-1}, f(x_{k-1}, u_k))$$
$$y_k = Cz_k$$

$$\begin{bmatrix} x_k \\ z_k \end{bmatrix} = \tilde{f}\left(\begin{bmatrix} x_{k-1} \\ z_{k-1} \end{bmatrix}, u_k\right)$$
$$y_k = Cz_k$$

$x(k), z(k)$ are states of the model

As many states as we have neurons in the hidden layers

Recurrent NN – General Form

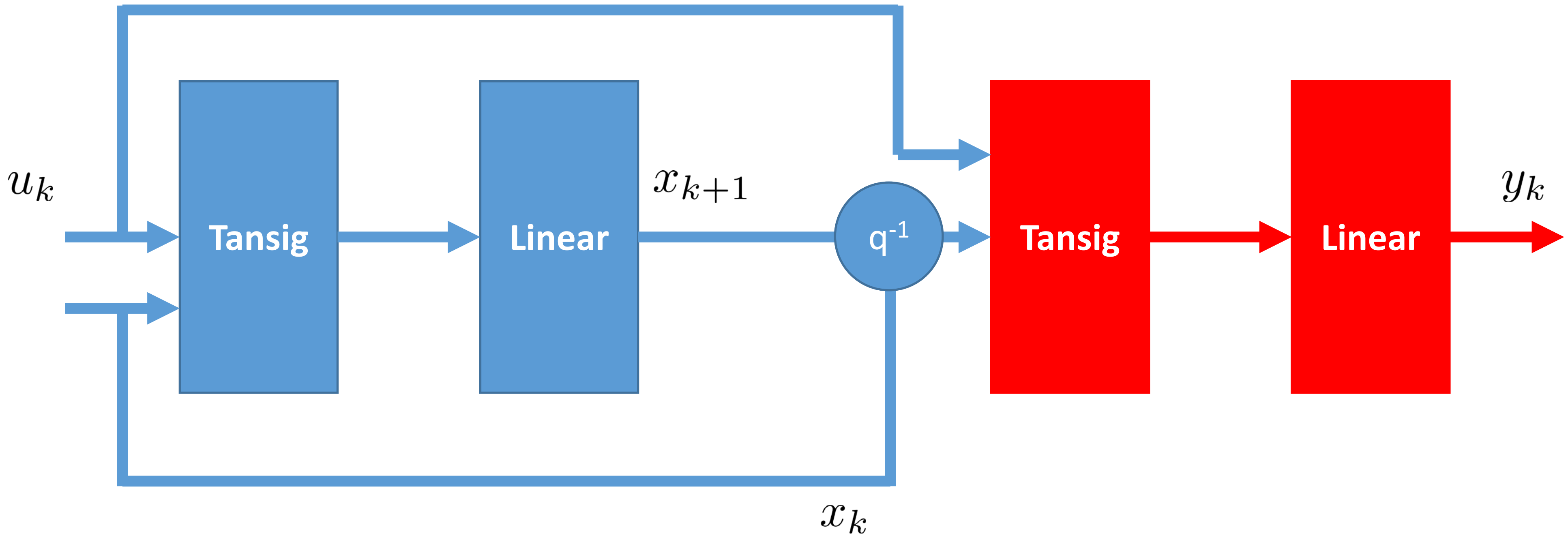
$$x_k = f\left(x_{k-1}, u_k, y_{k-1}\right)$$
$$y_k = g(x_k)$$

Comprises a huge range of recurrent NN structures, including LSTM

How to structure f, g? → Model structure selection / design

NN Training? → Initialization, training strategy, expanded training network

State-Space Neural Network (SSNN)

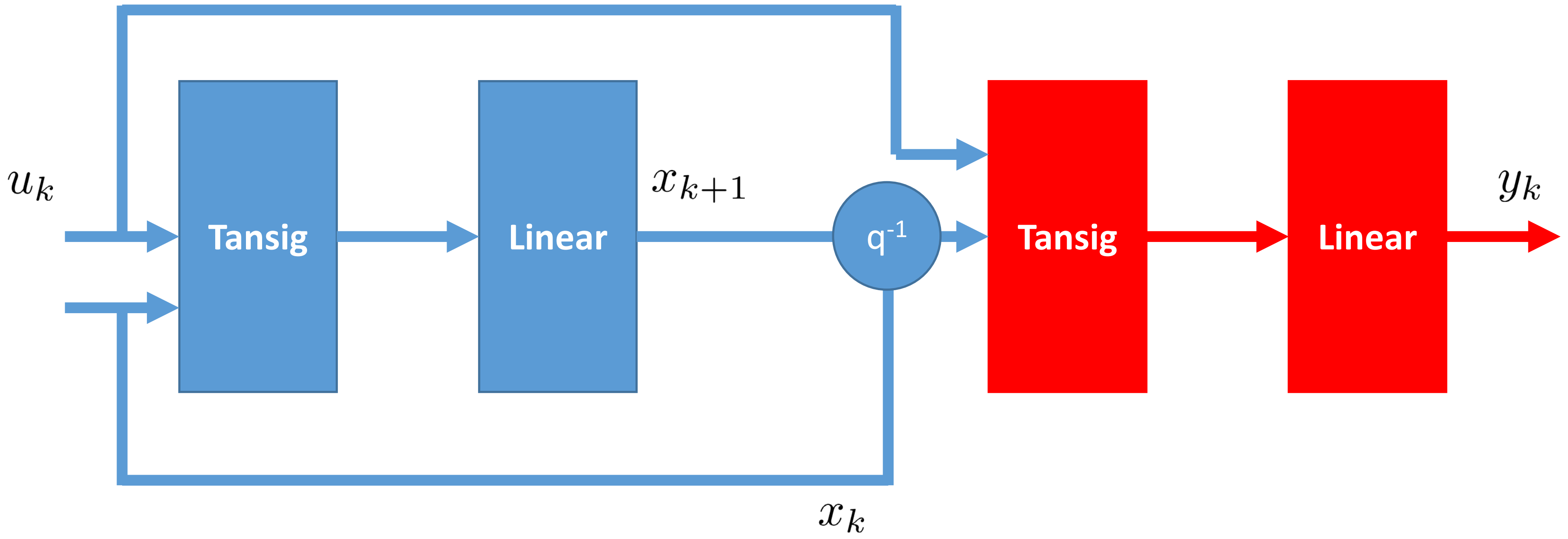


$$\boxed{x_{k+1} = f(x_k, u_k)}$$

$$\boxed{y_k = g(x_k, u_k)}$$

State dimension = # neurons in the blue linear layer

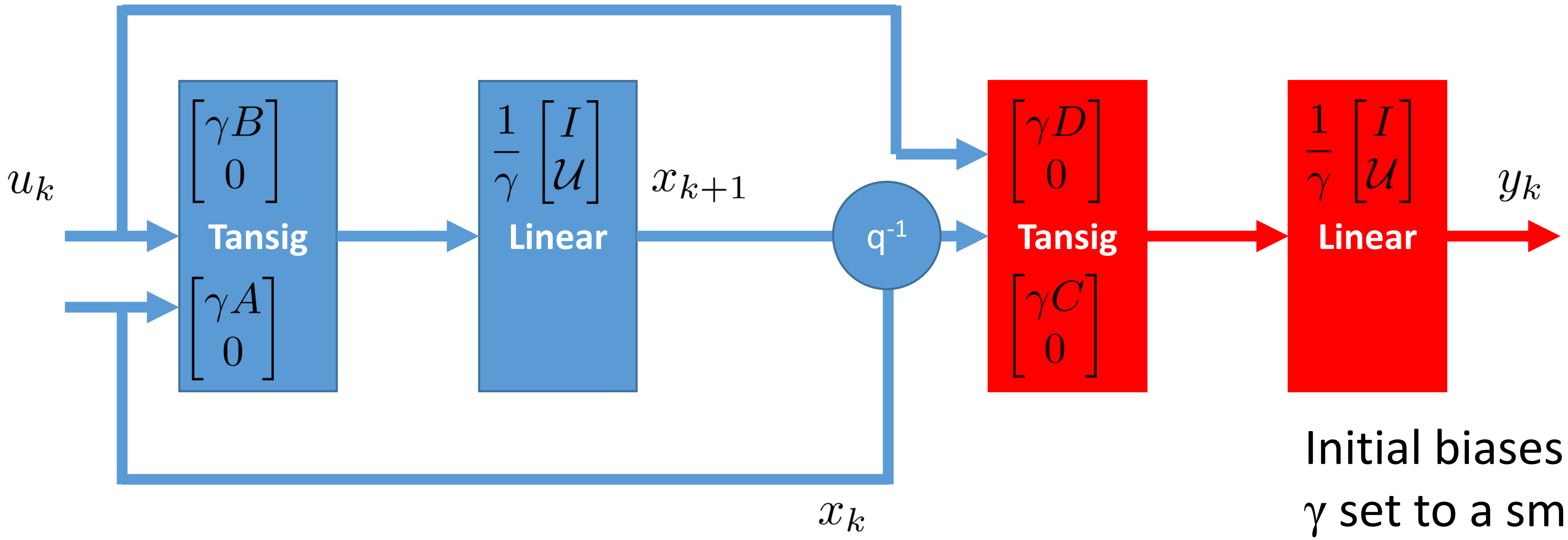
SSNN – Initialization



$$\boxed{x_{k+1} = f(x_k, u_k)}$$
$$\boxed{y_k = g(x_k, u_k)}$$

Random
Starting from a Linear Model (Suykens 1995)
Using Deep Autoencoder NN

SSNN – Initialization – Linear Approximation



Initial biases set to 0

γ set to a small value

$$x_{k+1} = f(x_k, u_k)$$
$$y_k = g(x_k, u_k)$$

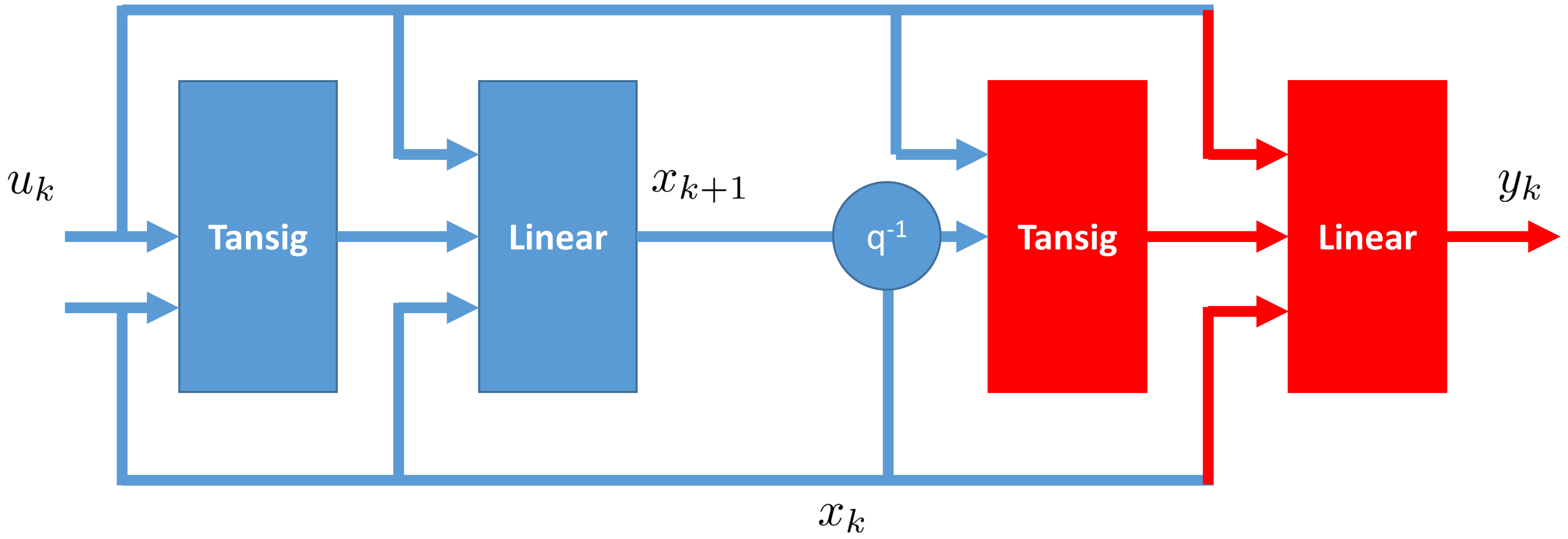
$\Rightarrow$

$$x_{k+1} = Ax_k + Bu_k$$
$$y_k = Cx_k + Du_k$$

J.A.K. Suykens et al., Nonlinear system identification using neural state space models, applicable to robust control design, *International Journal of Control*, Vol 62, pp. 129-152, 1995

SSNN – Initialization – Linear Approximation

Alternative SS-NN with hardwired Linear + Nonlinear Representation



$$x_{k+1} = f(x_k, u_k)$$
$$y_k = g(x_k, u_k)$$

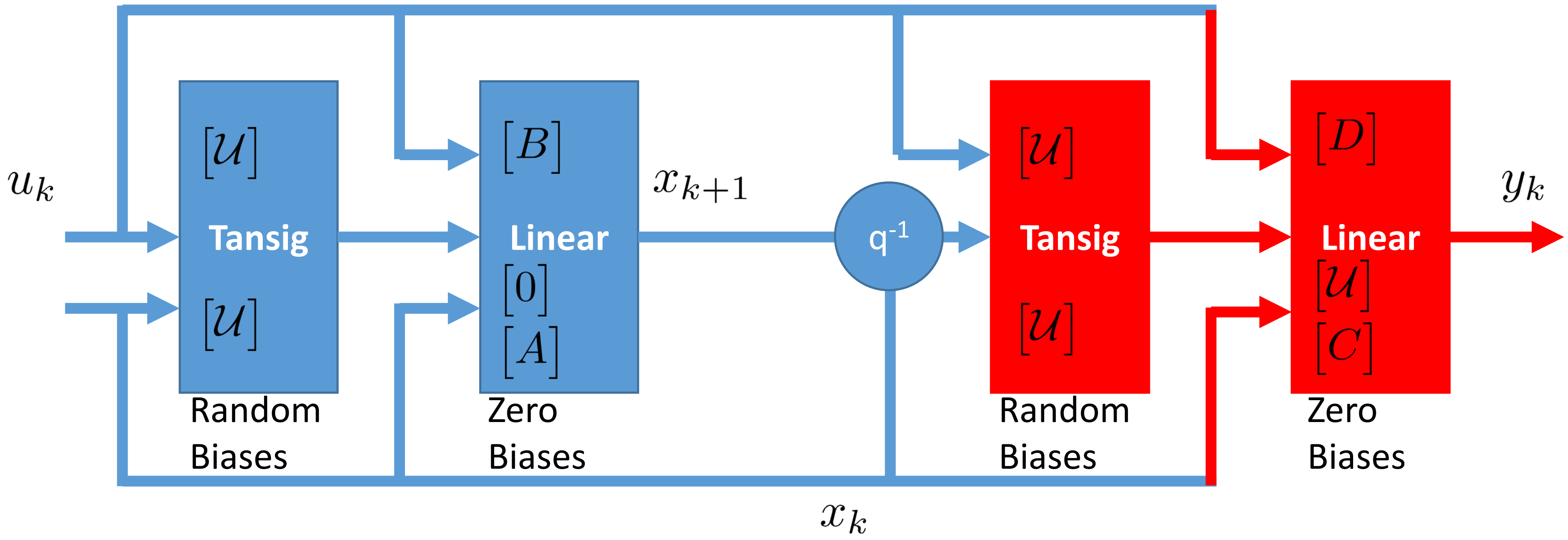
$$x_{k+1} = Ax_k + Bu_k + f(x_k, u_k)$$
$$y_k = Cx_k + Du_k + g(x_k, u_k)$$

Linear + Nonlinear

M. Schoukens. Improved Initialization of State-Space Artificial Neural Networks, European Control Conference, 2021. https://arxiv.org/pdf/2103.14516.pdf

SSNN – Initialization – Linear Approximation

Alternative SS-NN with hardwired Linear + Nonlinear Representation



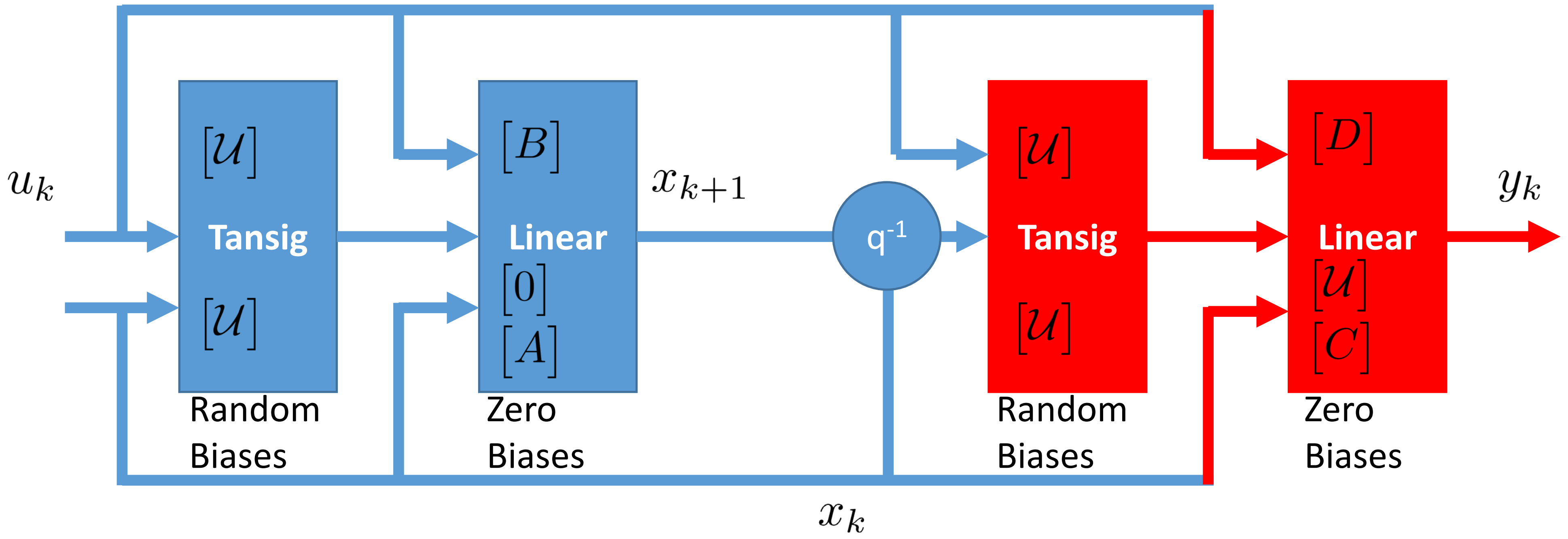
$$x_{k+1} = f(x_k, u_k)$$
$$y_k = g(x_k, u_k)$$

$$x_{k+1} = Ax_k + Bu_k + f(x_k, u_k)$$
$$y_k = Cx_k + Du_k + g(x_k, u_k)$$

Linear + Nonlinear

SSNN – Initialization – Linear Approximation

Alternative SS-NN with hardwired Linear + Nonlinear Representation



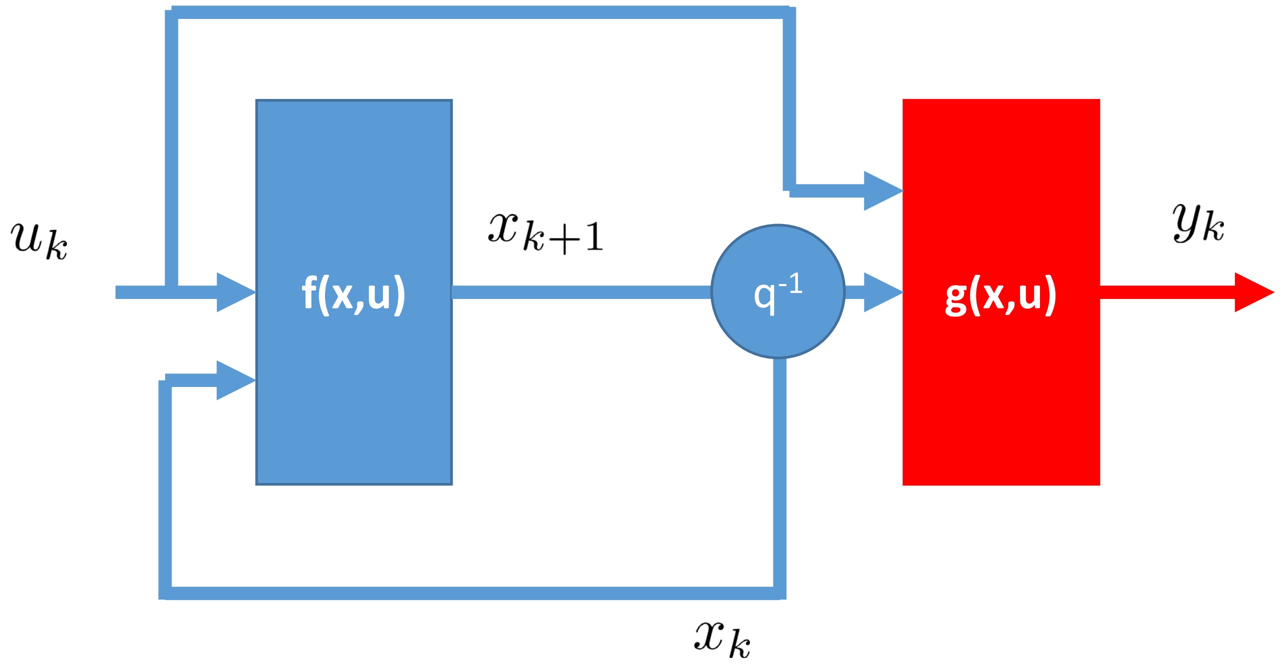
Linear part → Generalized ResNet

$$x_{k+1} = Ax_k + Bu_k + f(x_k, u_k)$$
$$y_k = Cx_k + Du_k + g(x_k, u_k)$$

Linear + Nonlinear

SSNN + Subspace Encoder

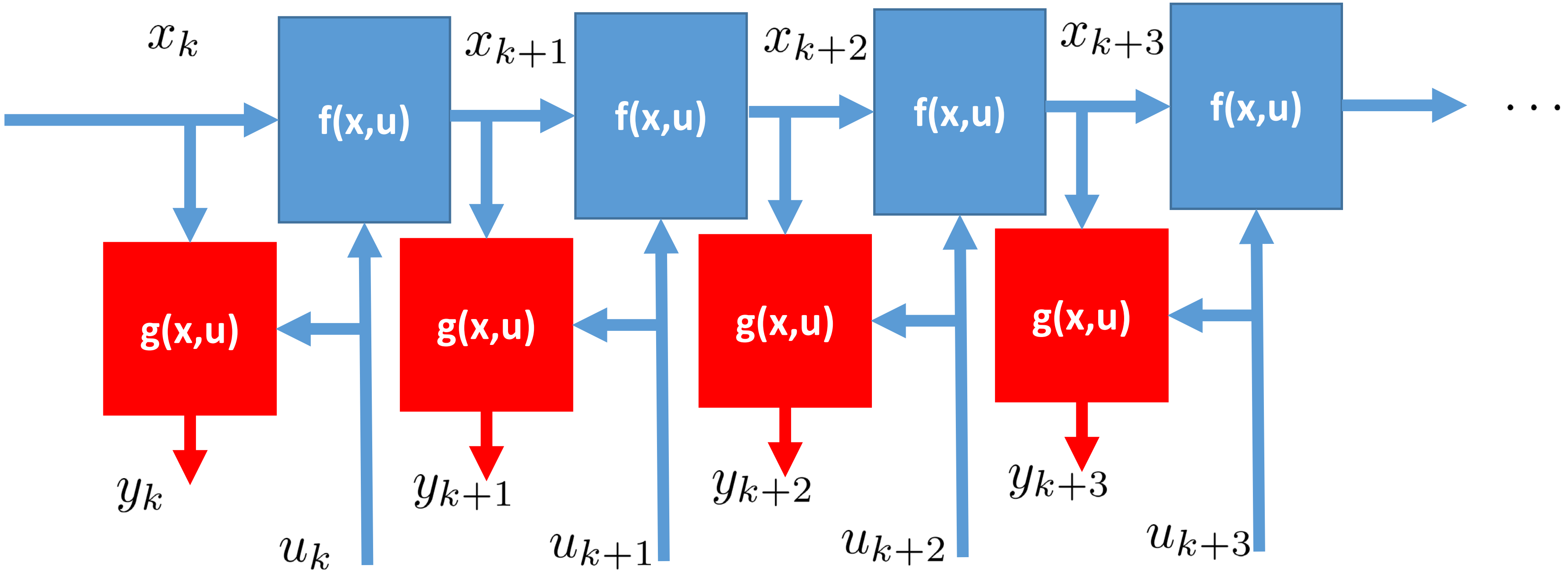
Hard problem due to
unknown state signal



Regular State-Space Expression

SSNN + Subspace Encoder

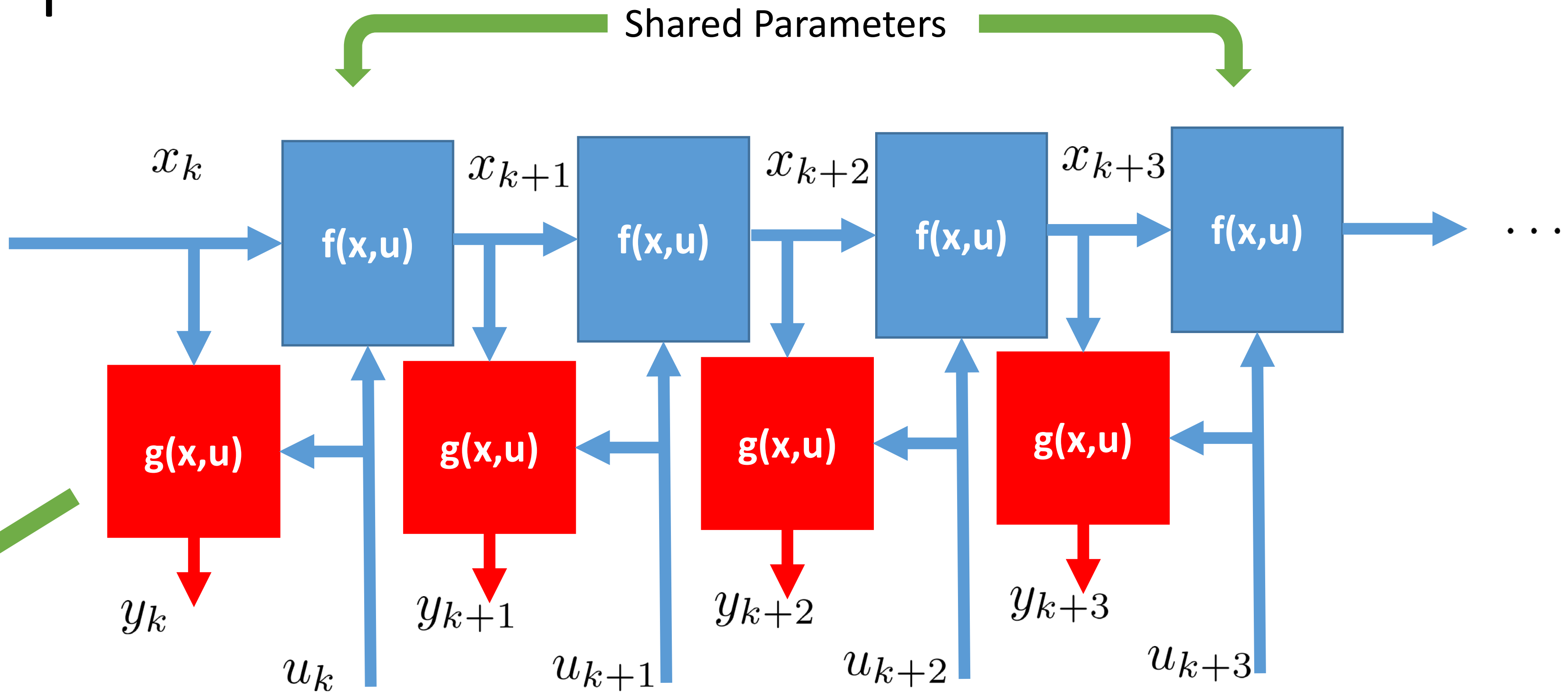
Breaking the recurrence, and only take a limited # steps forward smoothens the cost function



Unwrapped State-Space Expression

SSNN + Subspace Encoder

Breaking the recurrence, and only take a limited # steps forward smoothens the cost function



Fully connected ANN
Linear Bypass (~ ResNet)

Unwrapped State-Space Expression

Batch Optimization

Consider full dataset at every optimization step

$$V_{\text{simulation}}(\theta) = \frac{1}{N_{\text{samples}}} \sum_{t=1}^{N_{\text{samples}}} ||h_\theta(x_t, u_t) - y_t||_2^2$$

Split full dataset in smaller sections, only consider some of them at every optimization step

$$V_{\text{batch}}(\theta) = \frac{1}{2N_{\text{batch}}(T+1)} \sum_{i \in B} \sum_{k=k_0}^{T+k_0} ||\hat{y}_{t_i \to t_i+k} - y_{t_i+k}||^2,$$

$$B \subset \{1, 2, ..., N\}.$$

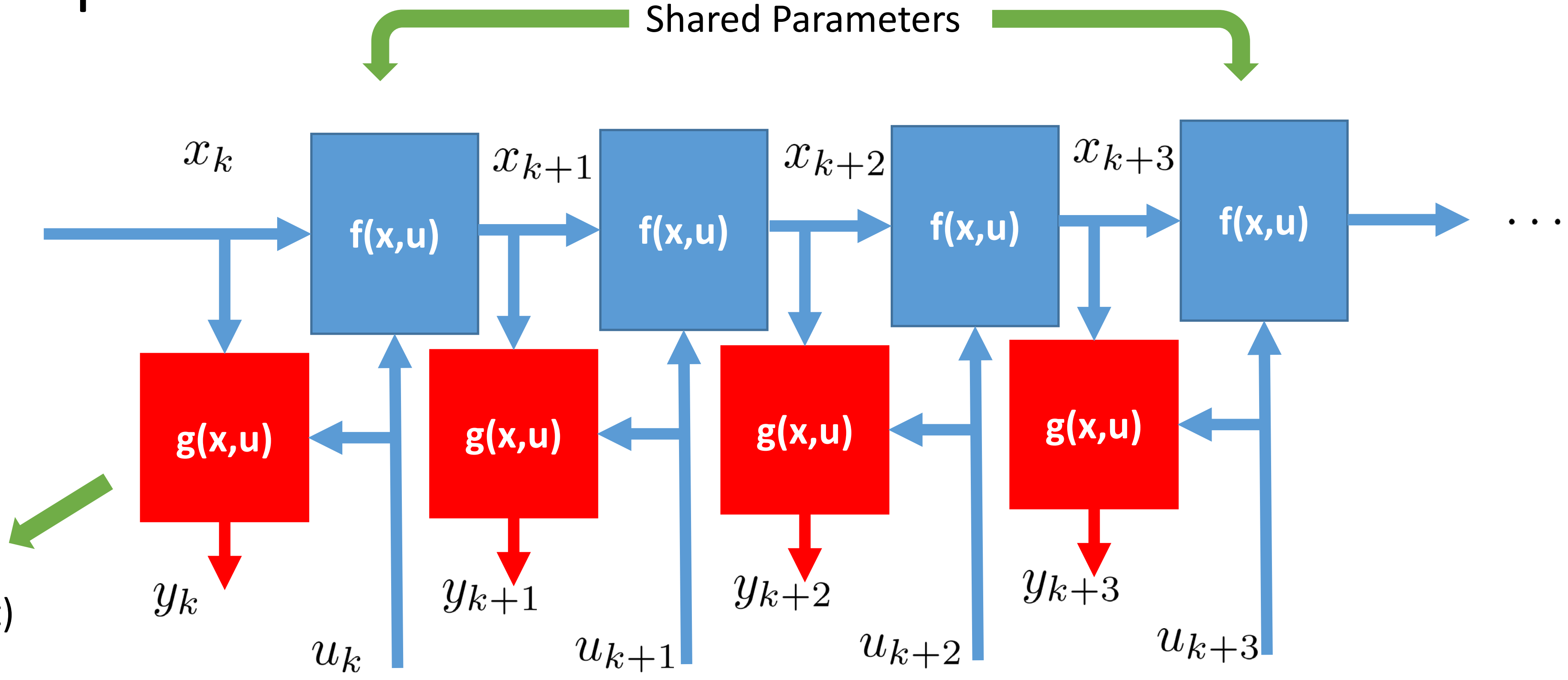
Improved computational efficiency

SSNN + Subspace Encoder

Breaking the recurrence, and only take a limited # steps forward smoothens the cost function

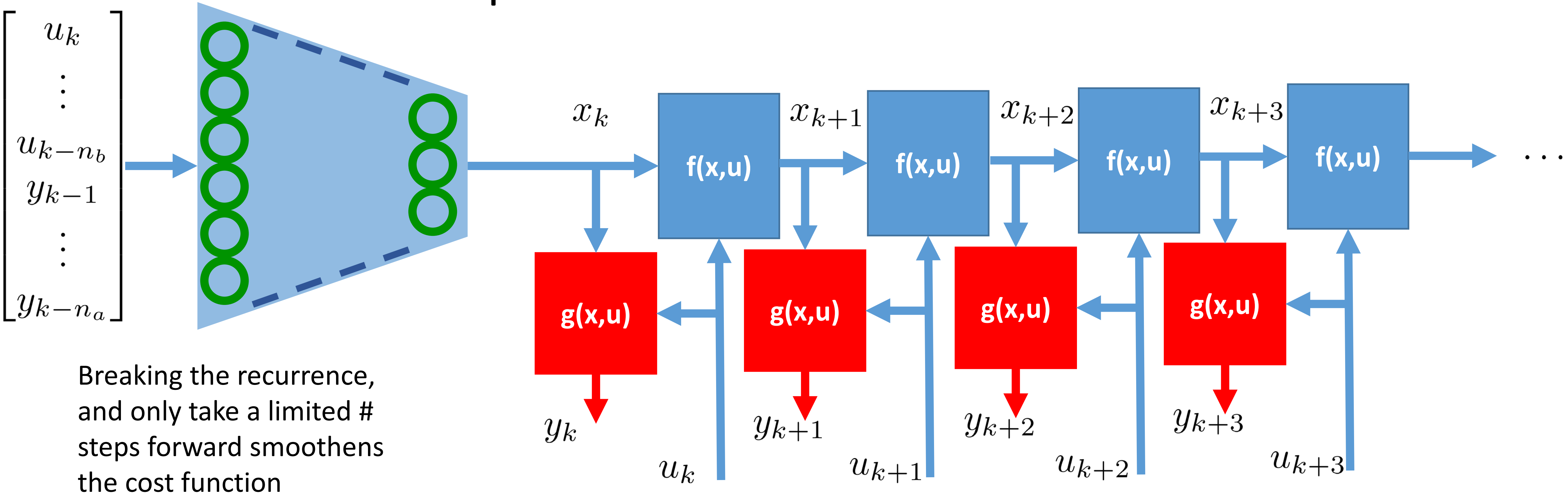
What should the starting state be?

Fully connected ANN
Linear Bypass (~ ResNet)



Unwrapped State-Space Expression

SSNN + Subspace Encoder



Breaking the recurrence, and only take a limited # steps forward smoothens the cost function

Learn initial state with an encoder → state is known at every time instance

Run for all time instances k

Unwrapped State-Space Expression

+

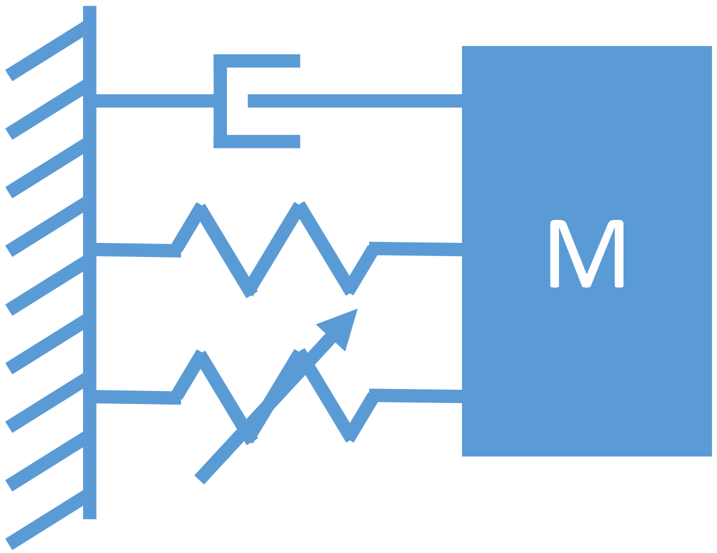
Encoder to learn starting state

Examples

Hysteretic System – Bouc-Wen MSD

Video-Encoder: Ball in a Box

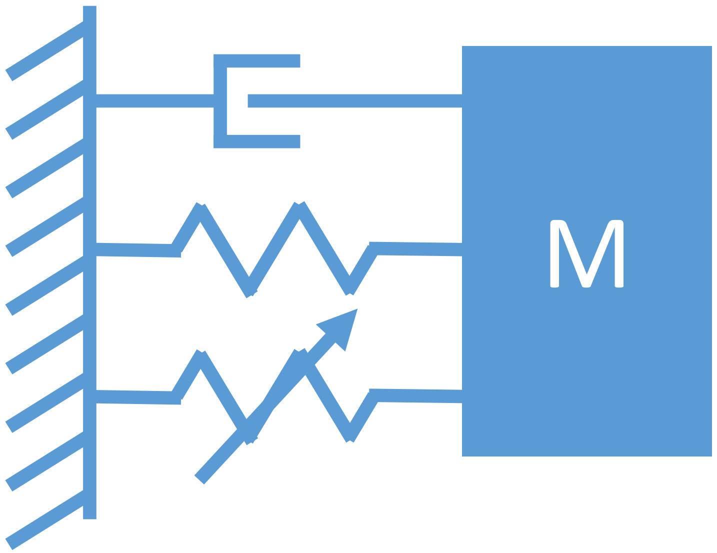
Hysteretic System: Bouc-Wen MSD (nonlinearbenchmark.org)



$$m\ddot{y}_t + c\dot{y}_t + ky_t + z(y_t, \dot{y}_t) = u_t$$

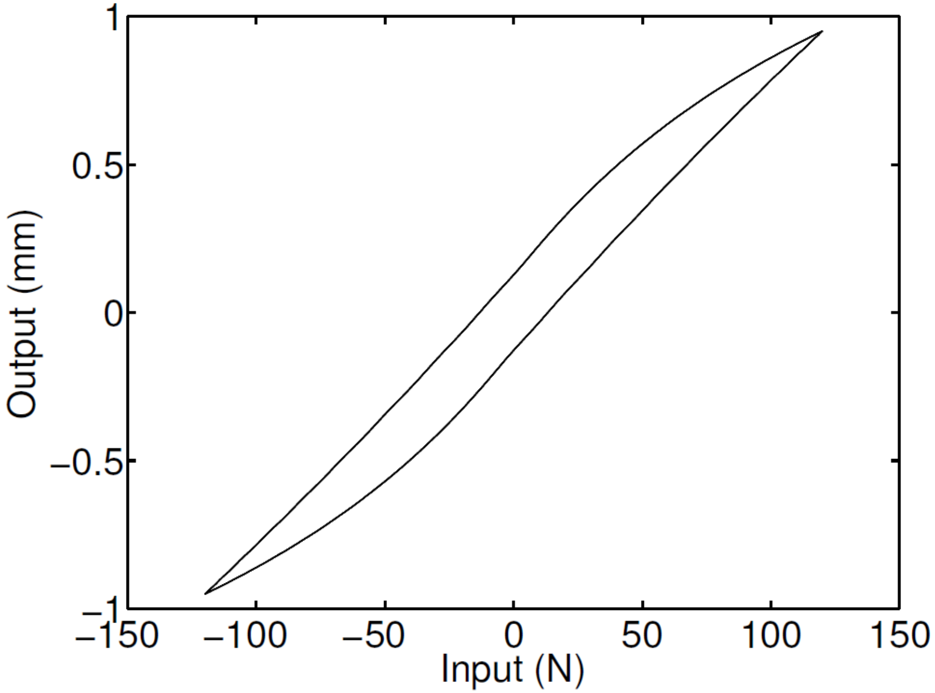
$$\dot{z}_t = \alpha\dot{y}_t - \beta\left(\gamma\left|\dot{y}_t\right| z_t + \delta\dot{y}_t\left|z_t\right|\right)$$

Hysteretic System: Bouc-Wen MSD



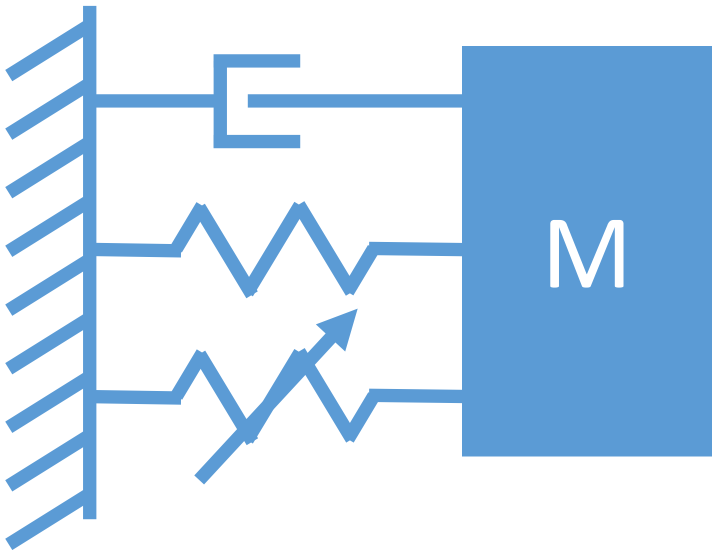
$$m\ddot{y}_t + c\dot{y}_t + ky_t + z(y_t, \dot{y}_t) = u_t$$

$$\dot{z}_t = \alpha\dot{y}_t - \beta\left(\gamma\left|\dot{y}_t\right| z_t + \delta\dot{y}_t\left|z_t\right|\right)$$



Hysteretic Loop

Hysteretic System – Linear Identification

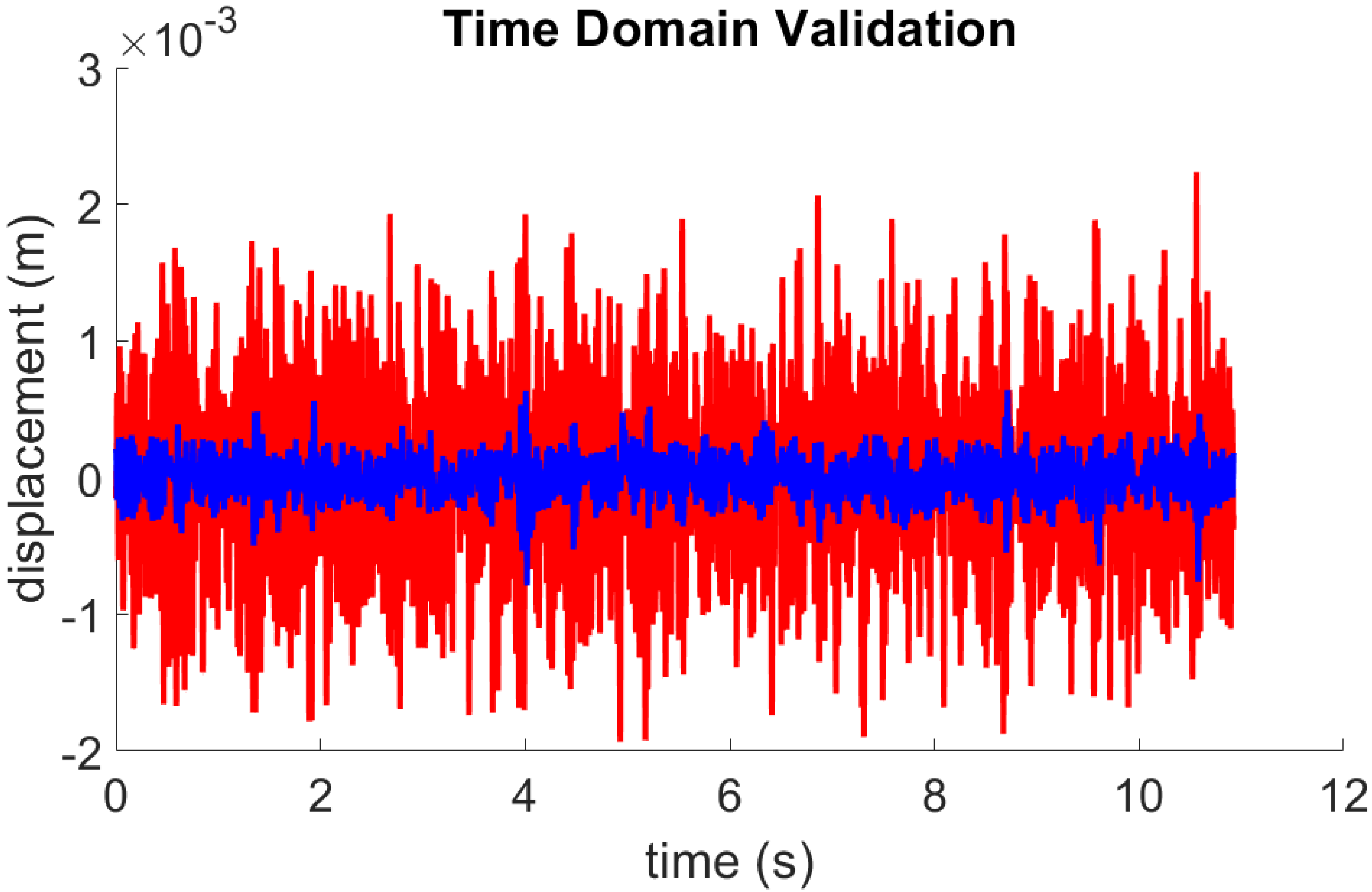


$$x_{k+1} = Ax_k + Bu_k$$
$$y_k = Cx_k + Du_k$$

3 states

Matlab function ‘ssest’

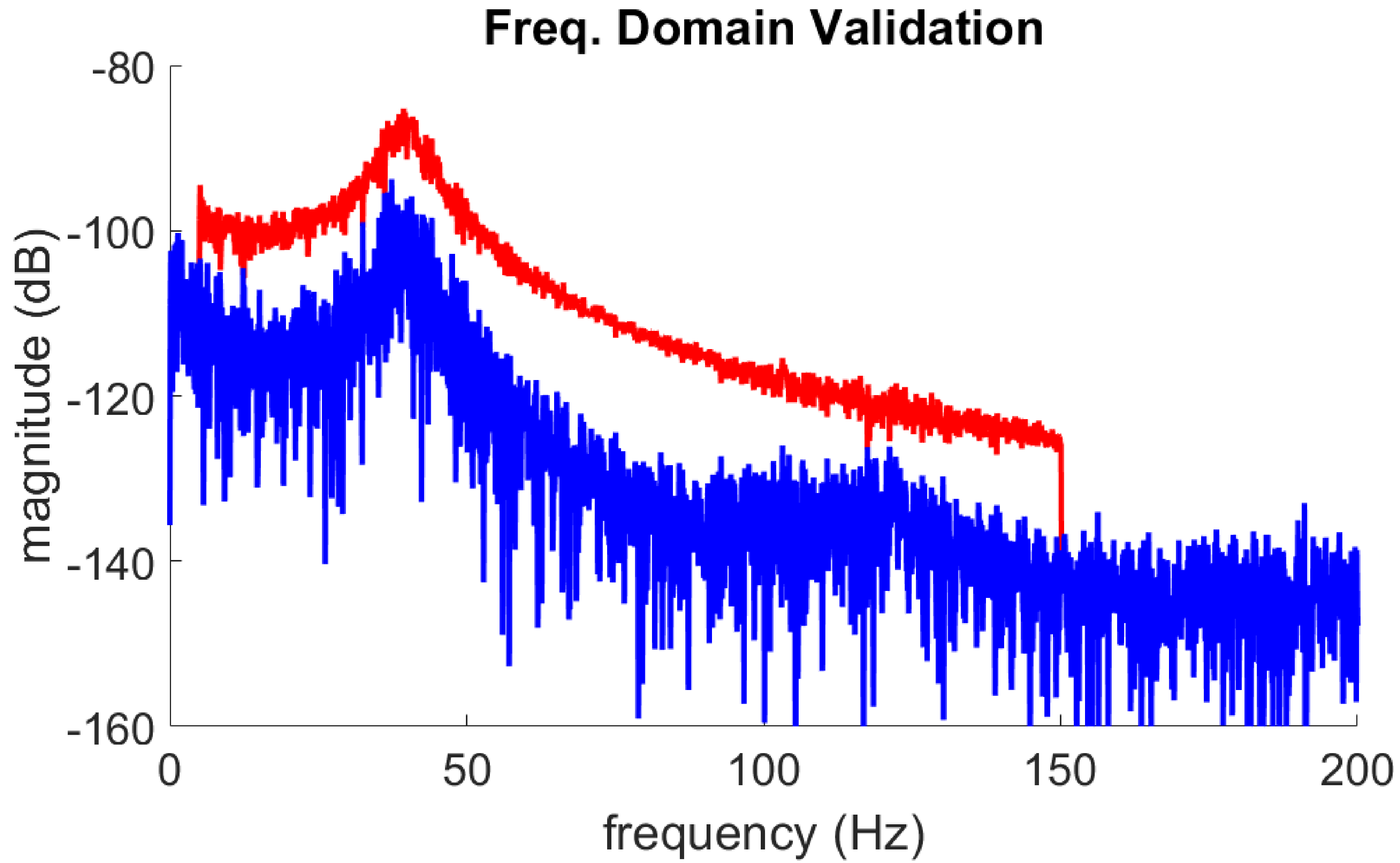
Hysteretic System – Linear Identification



Output

Linear Error

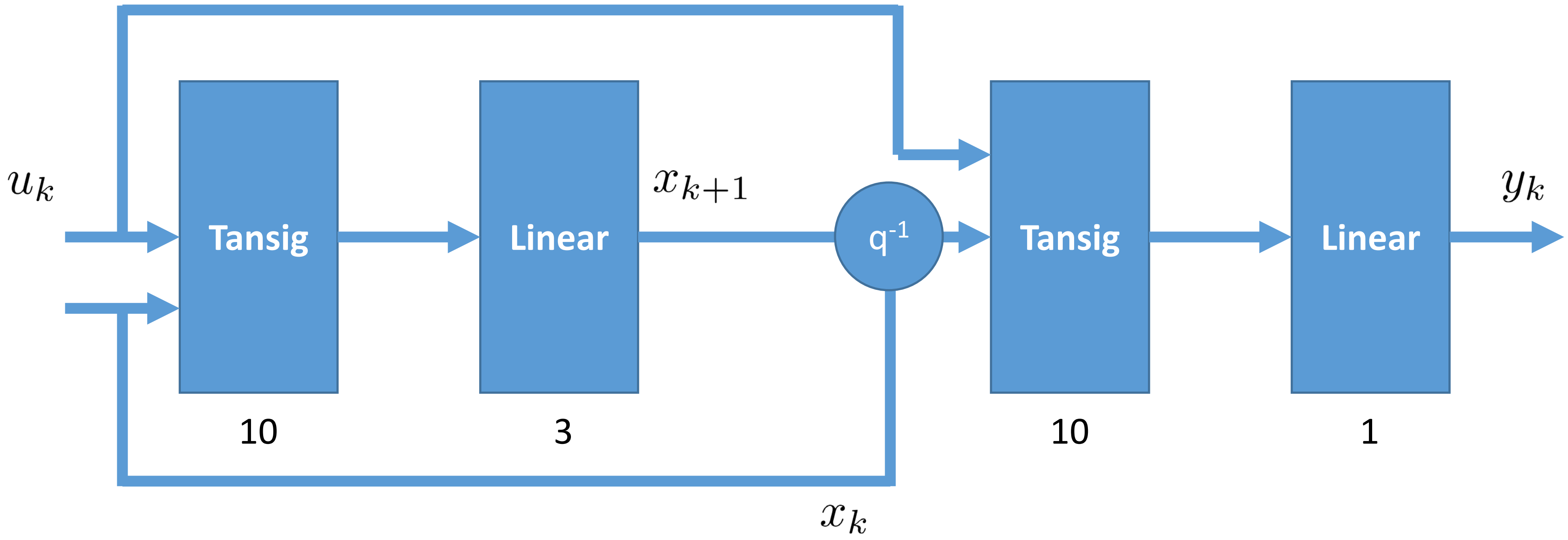
Hysteretic System – Linear Identification



Output

Linear Error

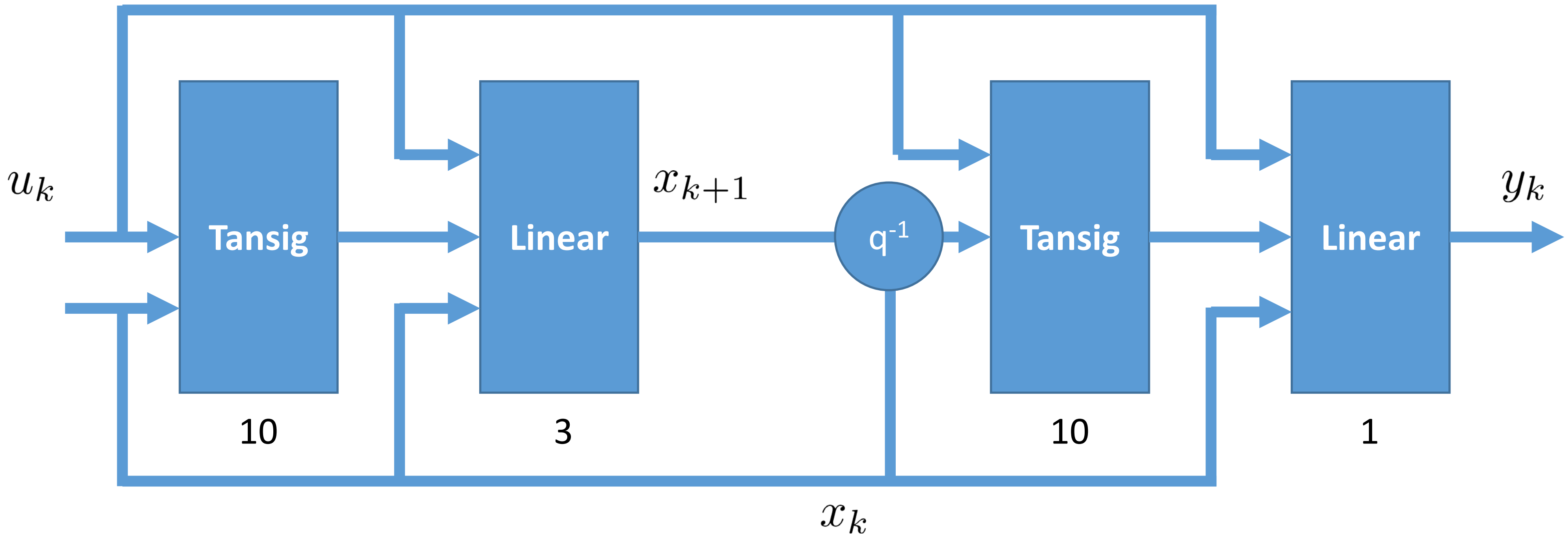
Hysteretic System – SS-NN



Random Initialization
Linear Initialization (Suykens 1995)

$$x_{k+1} = Ax_k + Bu_k$$
$$y_k = Cx_k + Du_k$$

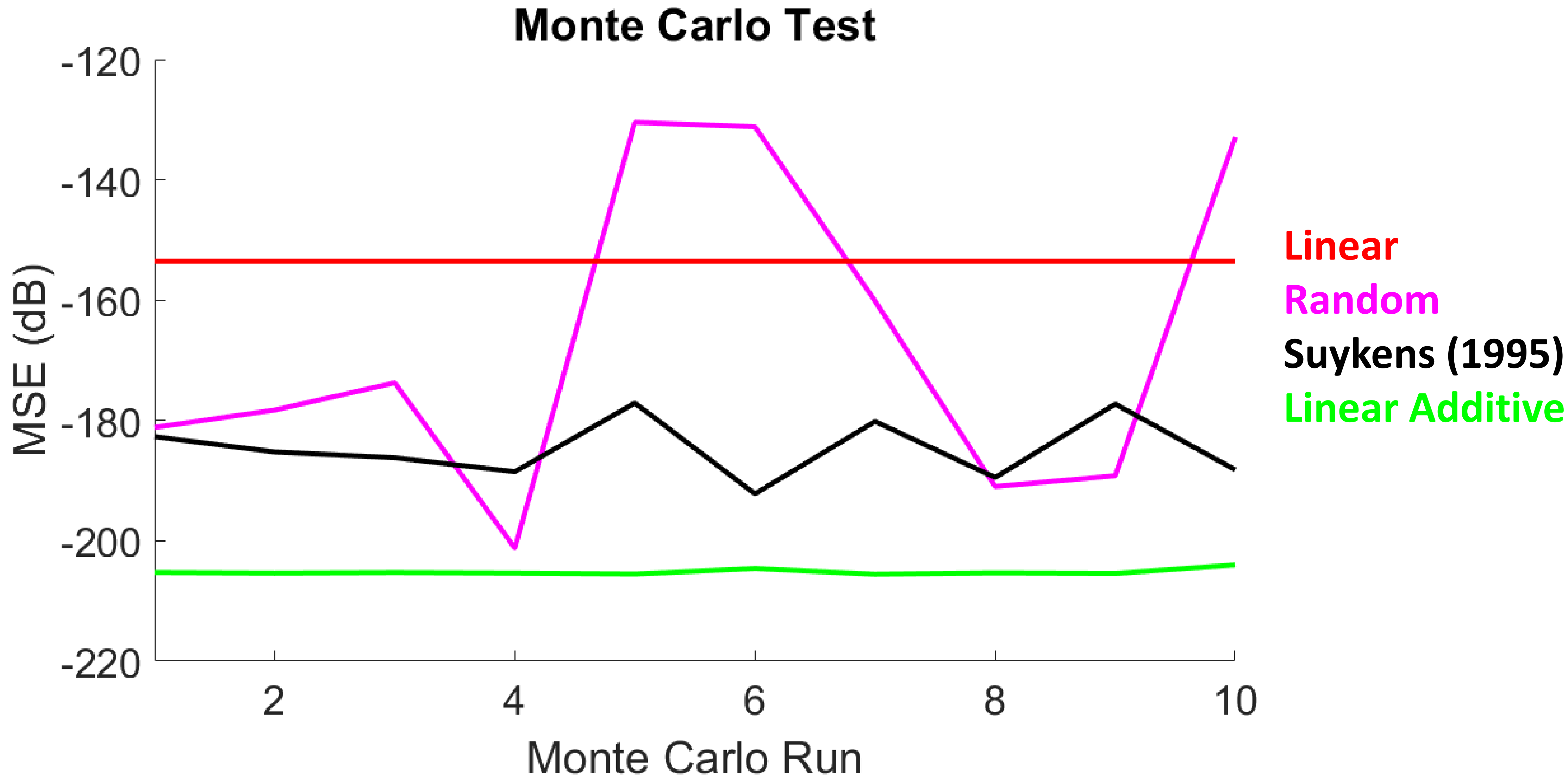
Hysteretic System – SS-NN



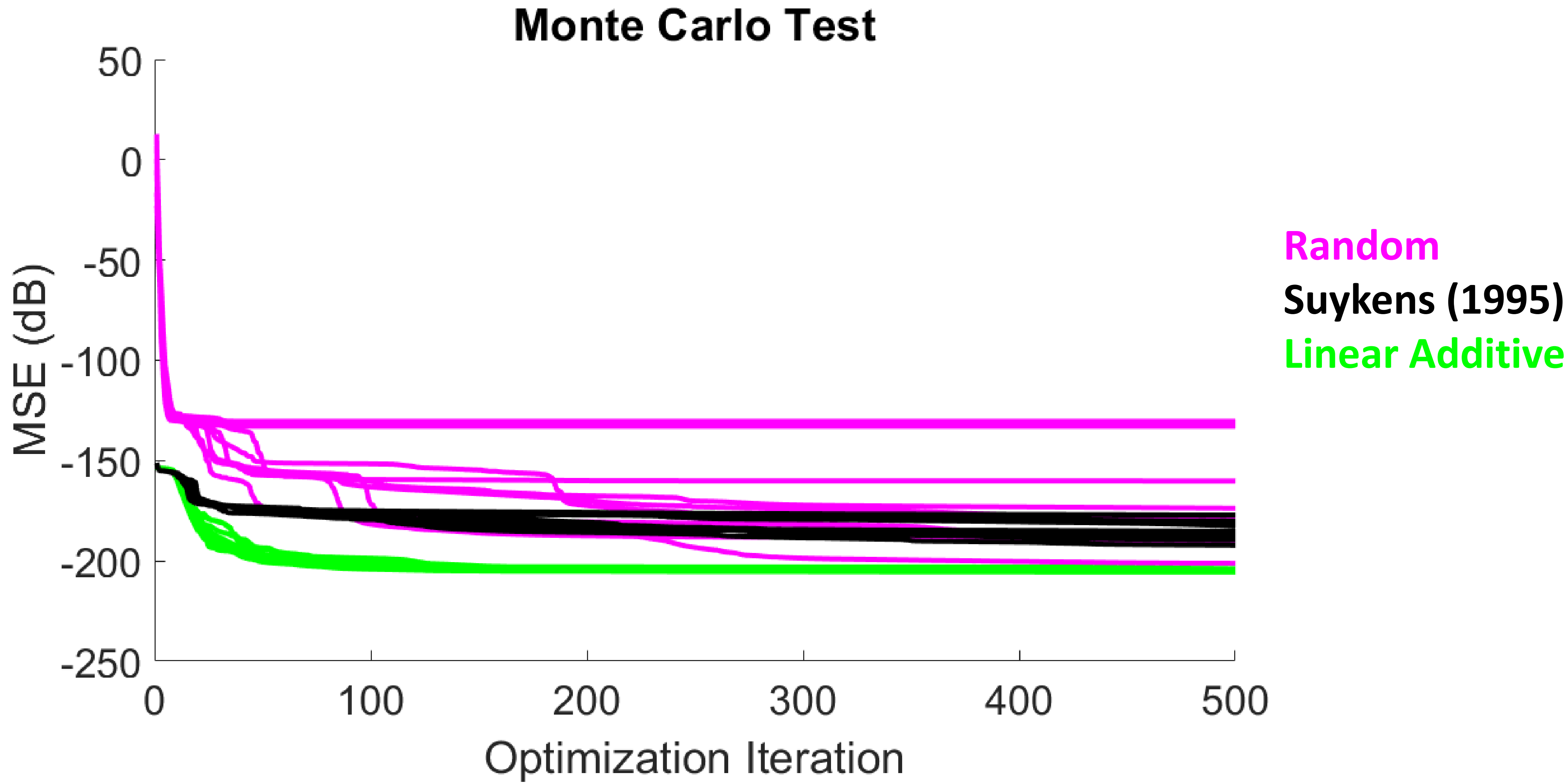
Linear + Nonlinear Initialization

$$x_{k+1} = Ax_k + Bu_k + f(x_k, u_k)$$
$$y_k = Cx_k + Du_k + g(x_k, u_k)$$

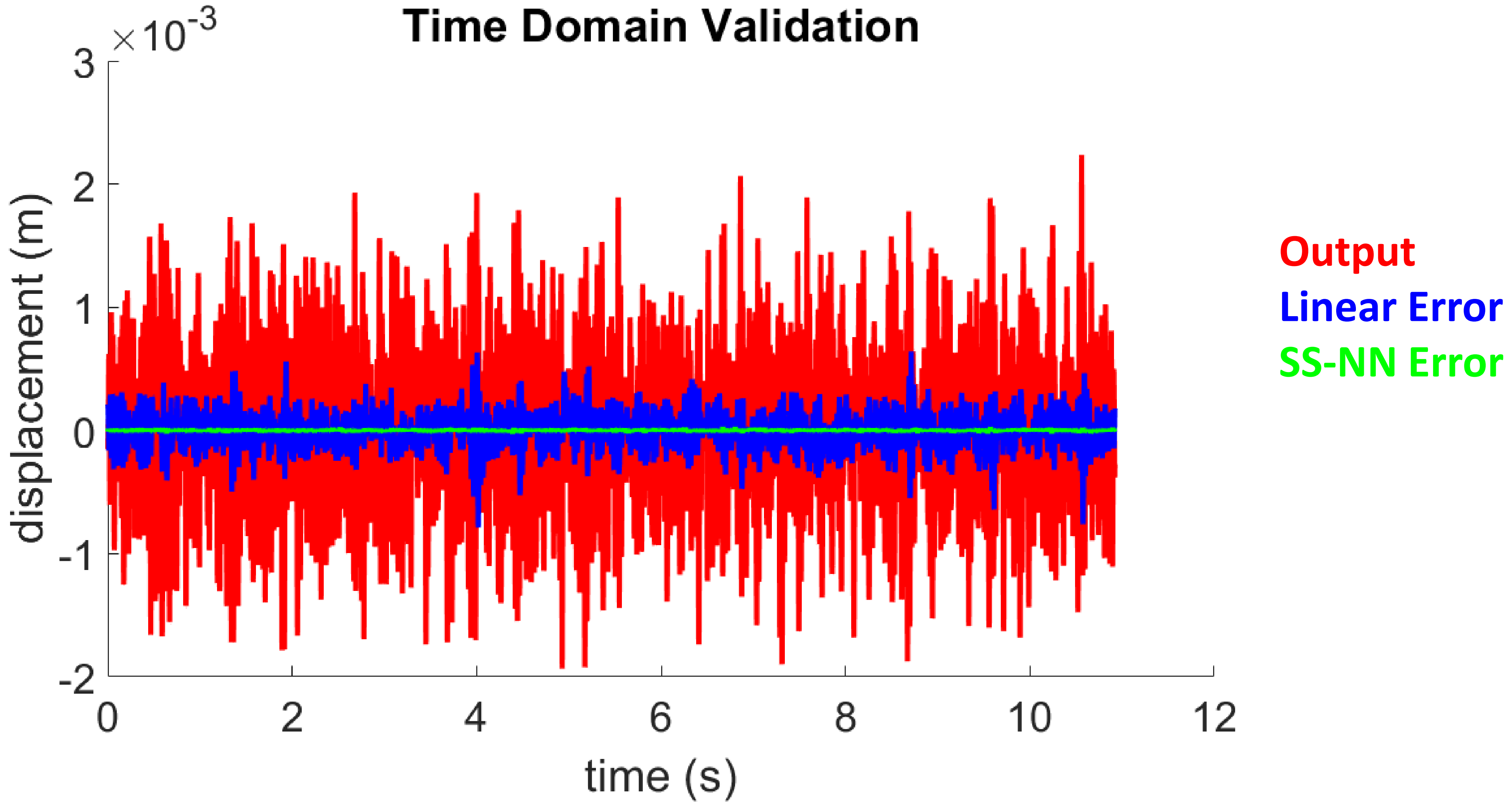
Hysteretic System – Results



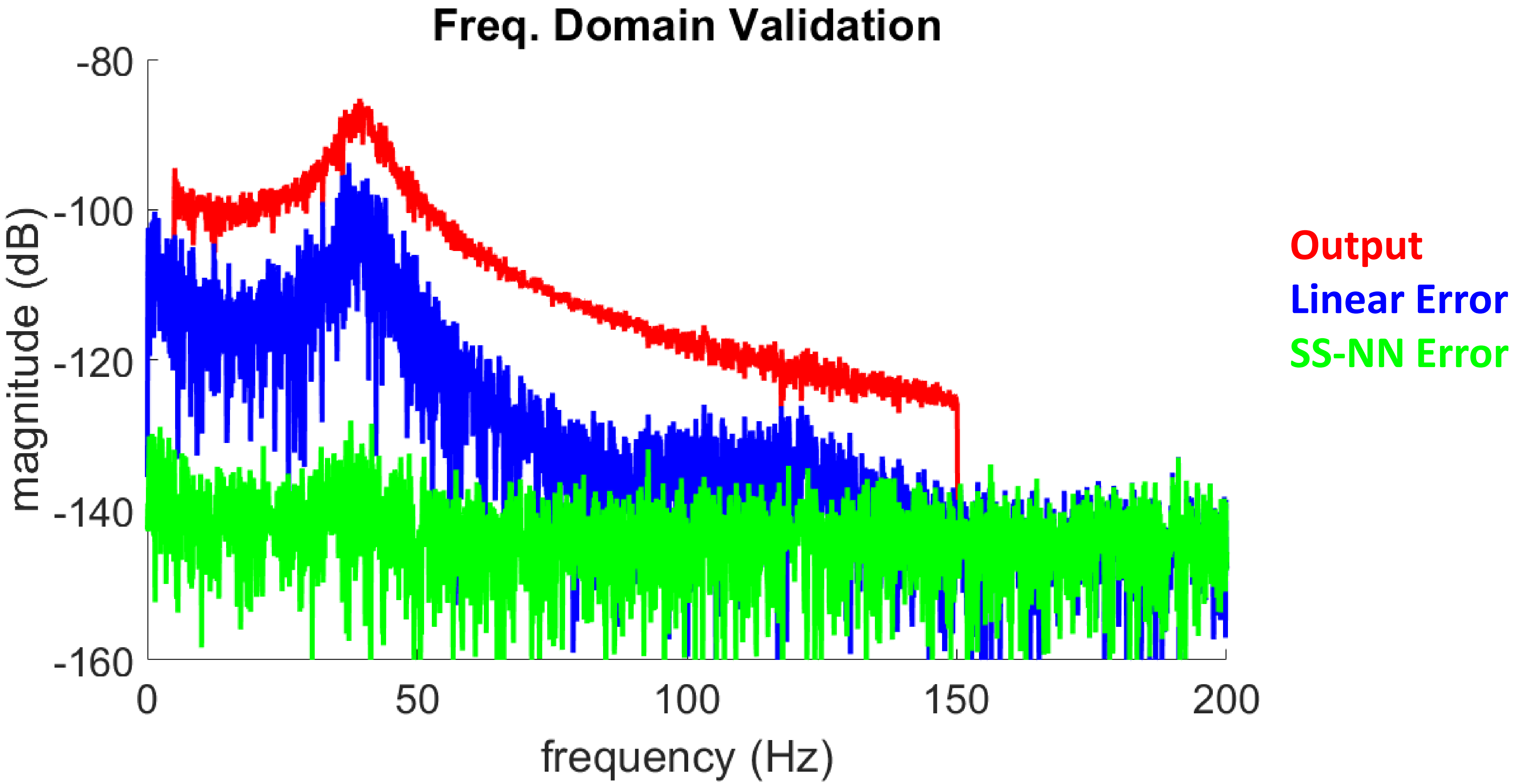
Hysteretic System – Results



Hysteretic System – Results



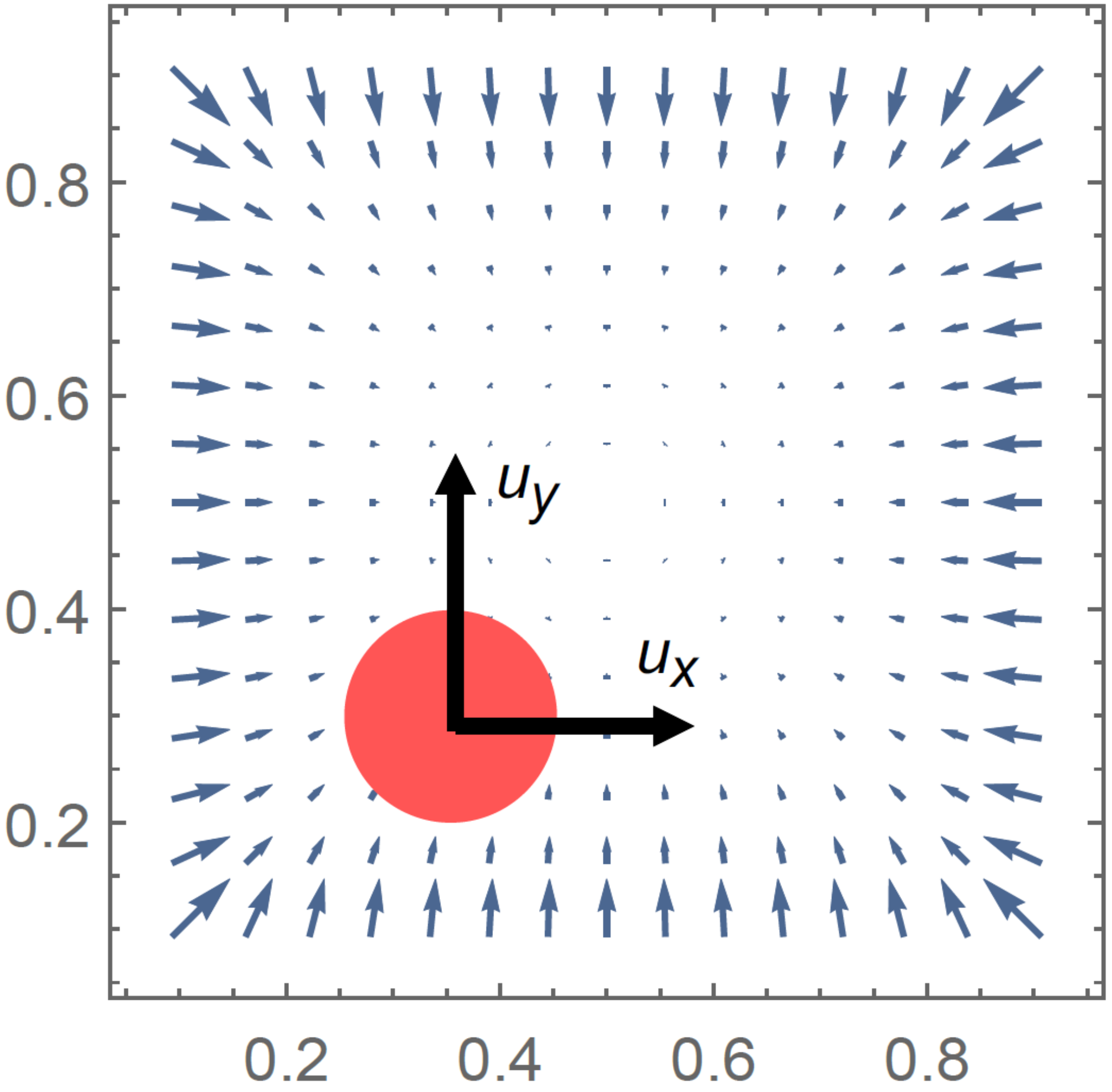
Hysteretic System – Results



Ball in a Box: System

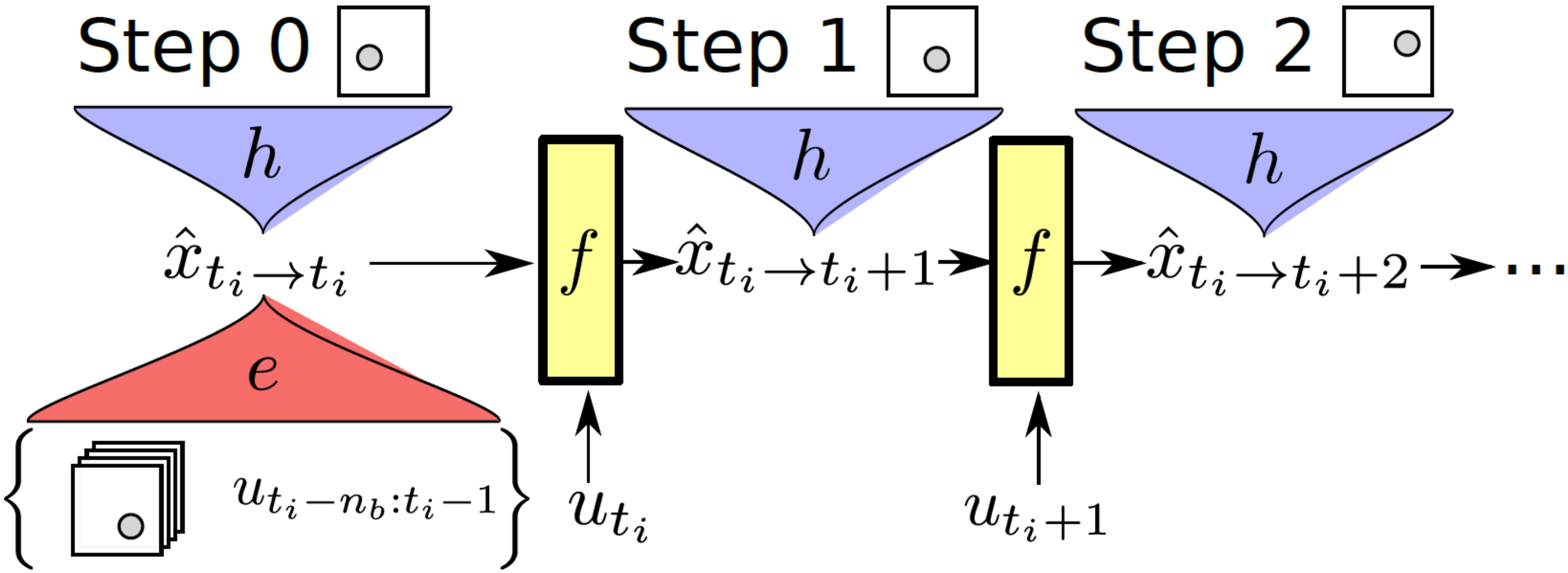
Mass in 2D forcefield

Input: external forces
Output: 25x25 video feed of box



G.I. Beintema et al., Non-linear State-space Model Identification from Video Data using Deep Encoders, IFAC Symposium on System Identification (SYSID'21), 2021. https://arxiv.org/pdf/2012.07721.pdf

Ball in a Box: State-Space with Subspace Encoder

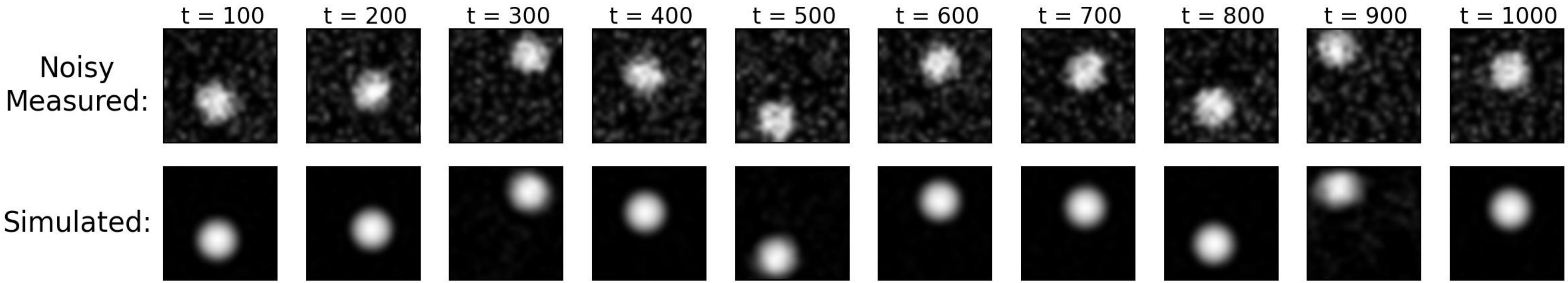


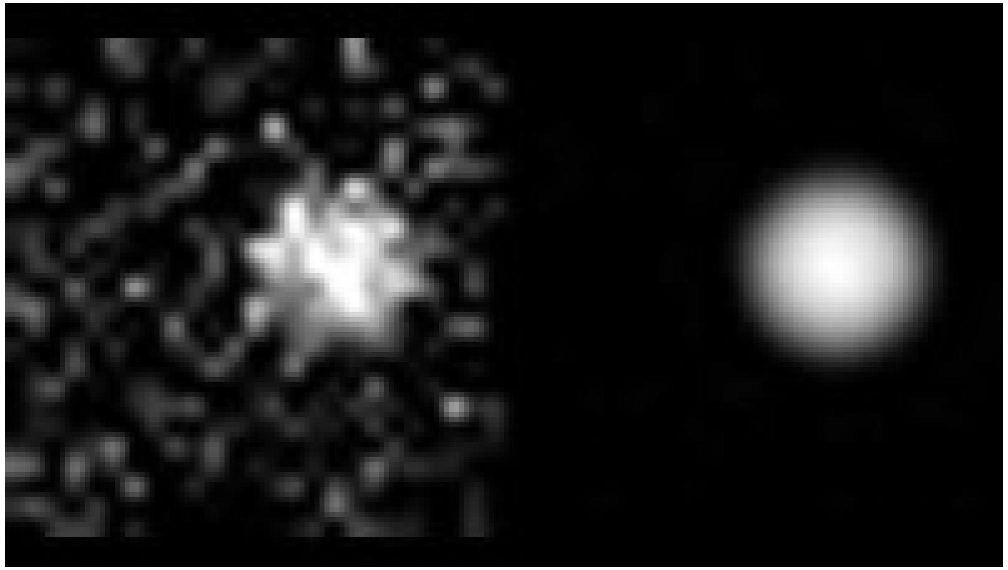
6 States 50 steps ahead n_a, n_b = 5

h, *f*, *e*: 64 neurons/layer, 2 layers, tansig activation, linear bypass

Random weight and bias initialization

Ball in a Box: Results





https://www.youtube.com/watch?v=IJzW1ma_7Wg

Ball in a Box: Future

Convolutional Layers in Encoder / Output Function

Data Management

Identification and Control of Spatio-Temporal Systems

https://www.youtube.com/watch?v=lJzW1ma_7Wg

Discussion

Embedding systems & control knowledge is advantageous, noise is important

a step towards explainable AI

Dynamic models have a wide range of use

control, system design, system validation, understanding

Model to be estimated can be

system model

feedforward controller / policy

feedback controller / policy

Artificial Neural Networks

Deep Learning & Deep Neural Networks

Training a Deep Neural Network

Artificial Neural Networks for Dynamical Systems