Machine Learning for Systems and Control

5SC28

Lecture 3A

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

Learning Outcomes

What is an artificial neural network?

Why are artificial neural networks interesting for function approximation?

How to train a neural network? / What is backpropagation?

Artificial Neural Networks

What is an artificial neural network?

Simple feedforward networks

Approximation properties

Training an artificial neural network

Artificial Neural Networks

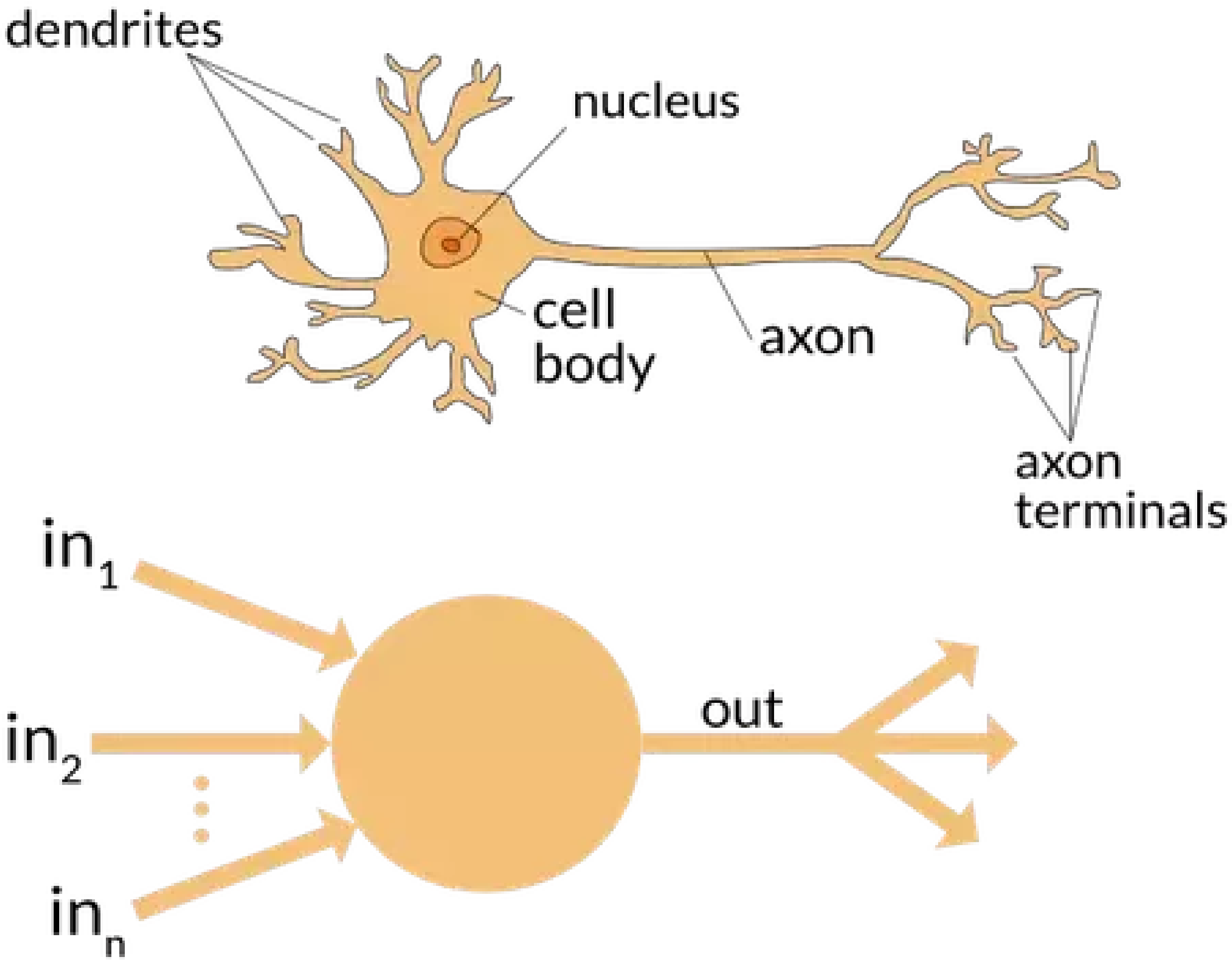
What is an artificial neural network?

Simple feedforward networks

Approximation properties

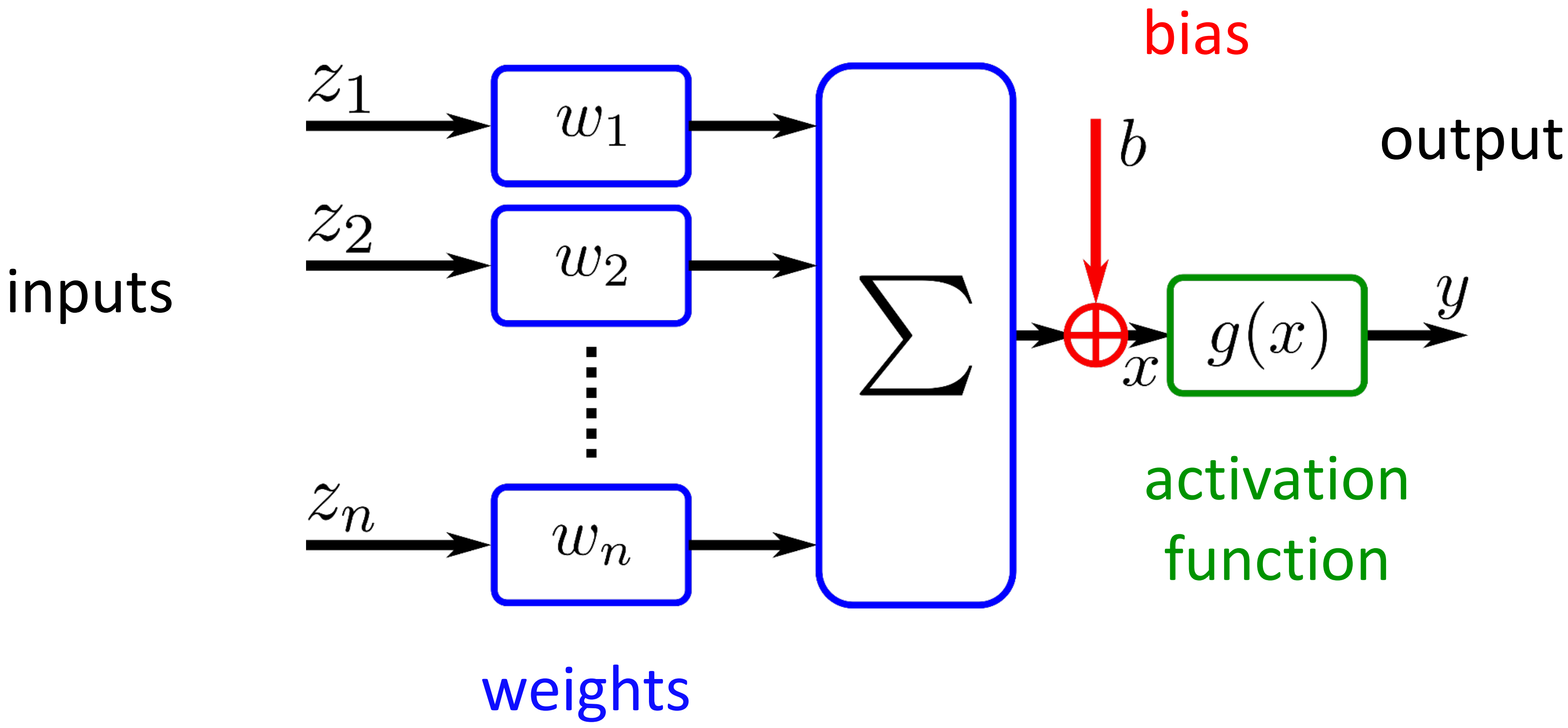
Training an artificial neural network

Biology Inspired



Source: https://www.quora.com/

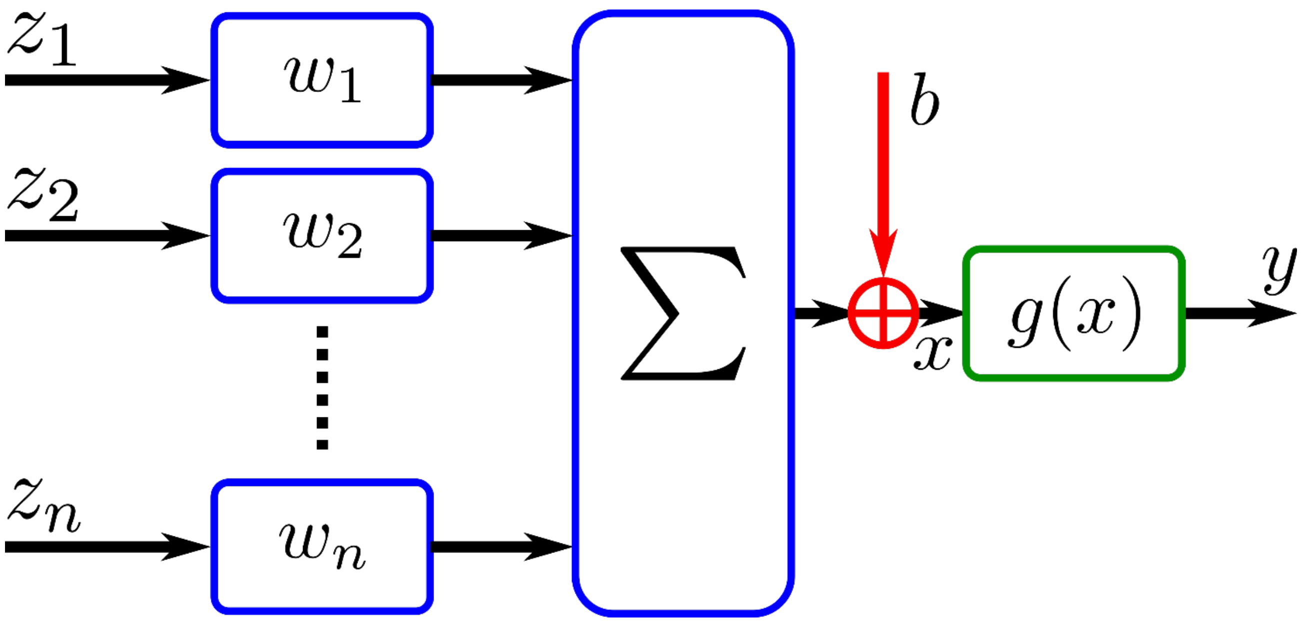
The Artificial Neuron



The Artificial Neuron

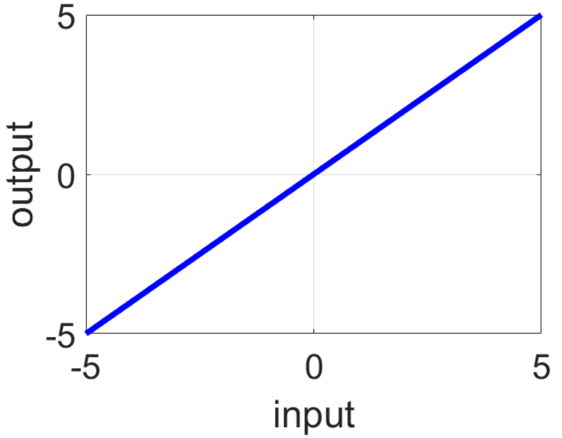
$$y = g(\boldsymbol{w}^T \boldsymbol{z} + b)$$

$$\boldsymbol{w} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix} \quad \boldsymbol{z} = \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_n \end{bmatrix}$$

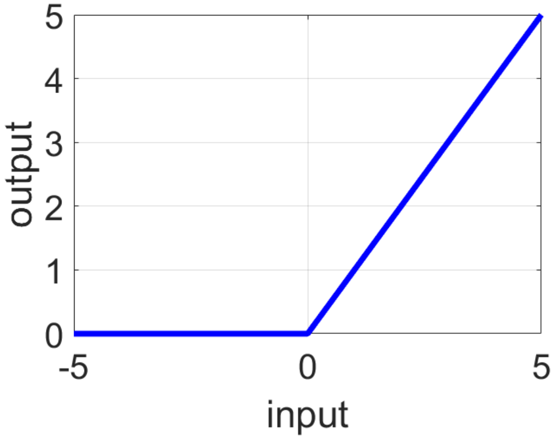


Activation Function

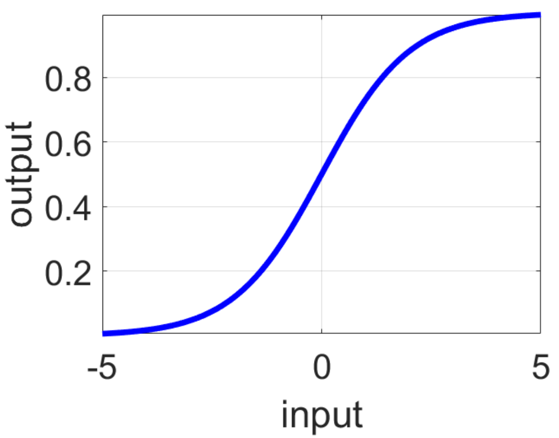
Linear



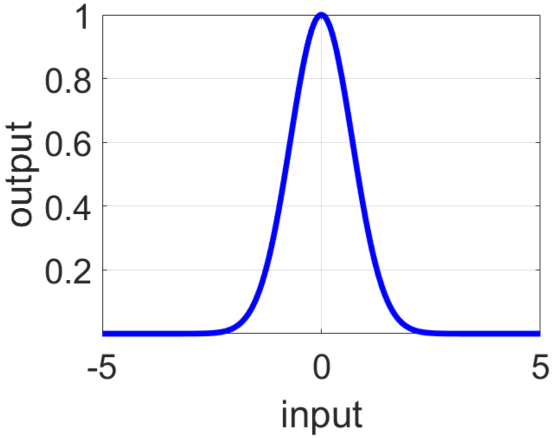
Rectified Linear Unit



Sigmoid



Radial Basis



$$g(x) = x$$

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

ReLu

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

Logistic Function

$$g(x) = e^{-x^2}$$

Artificial Neural Networks

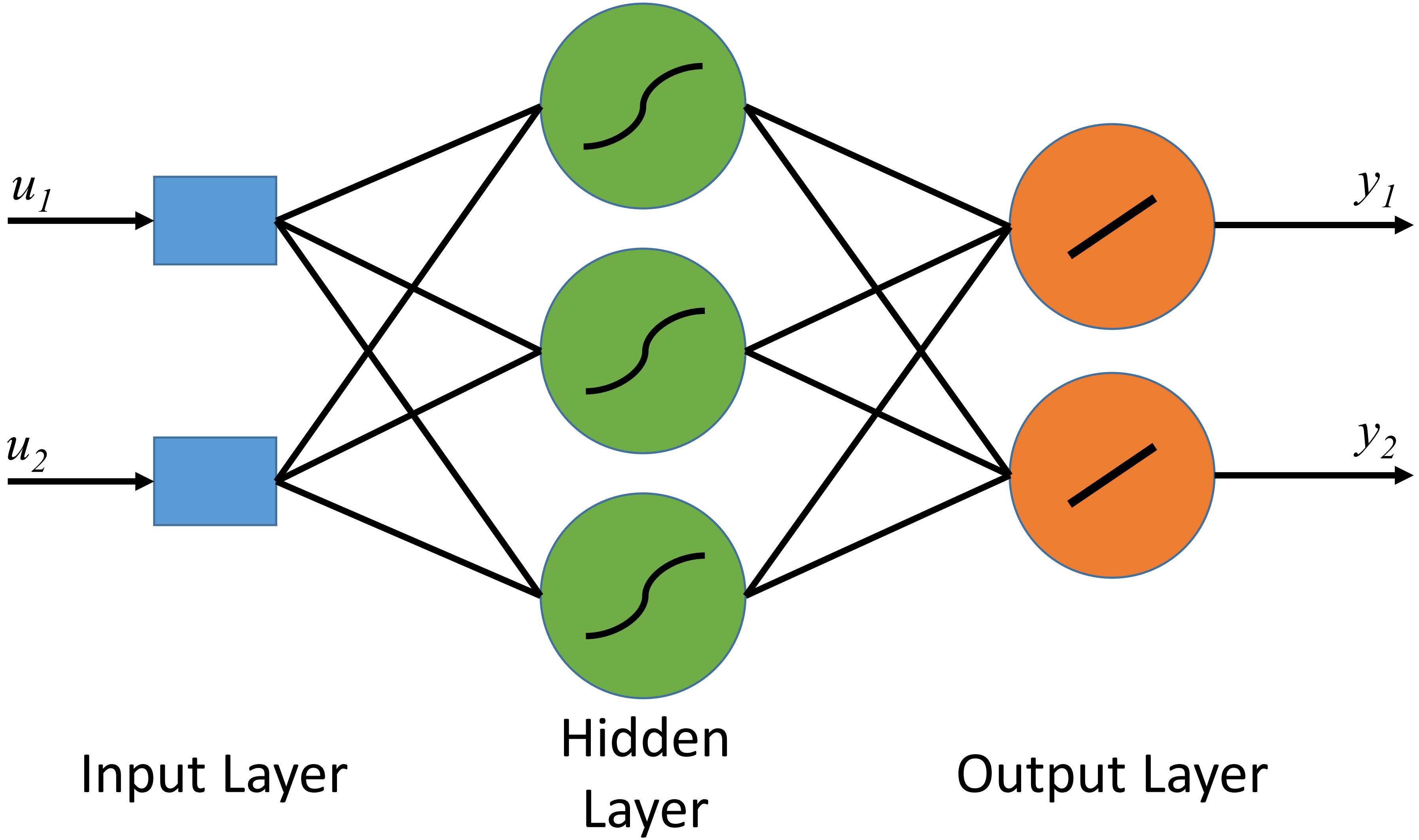
What is an artificial neural network?

Simple feedforward networks

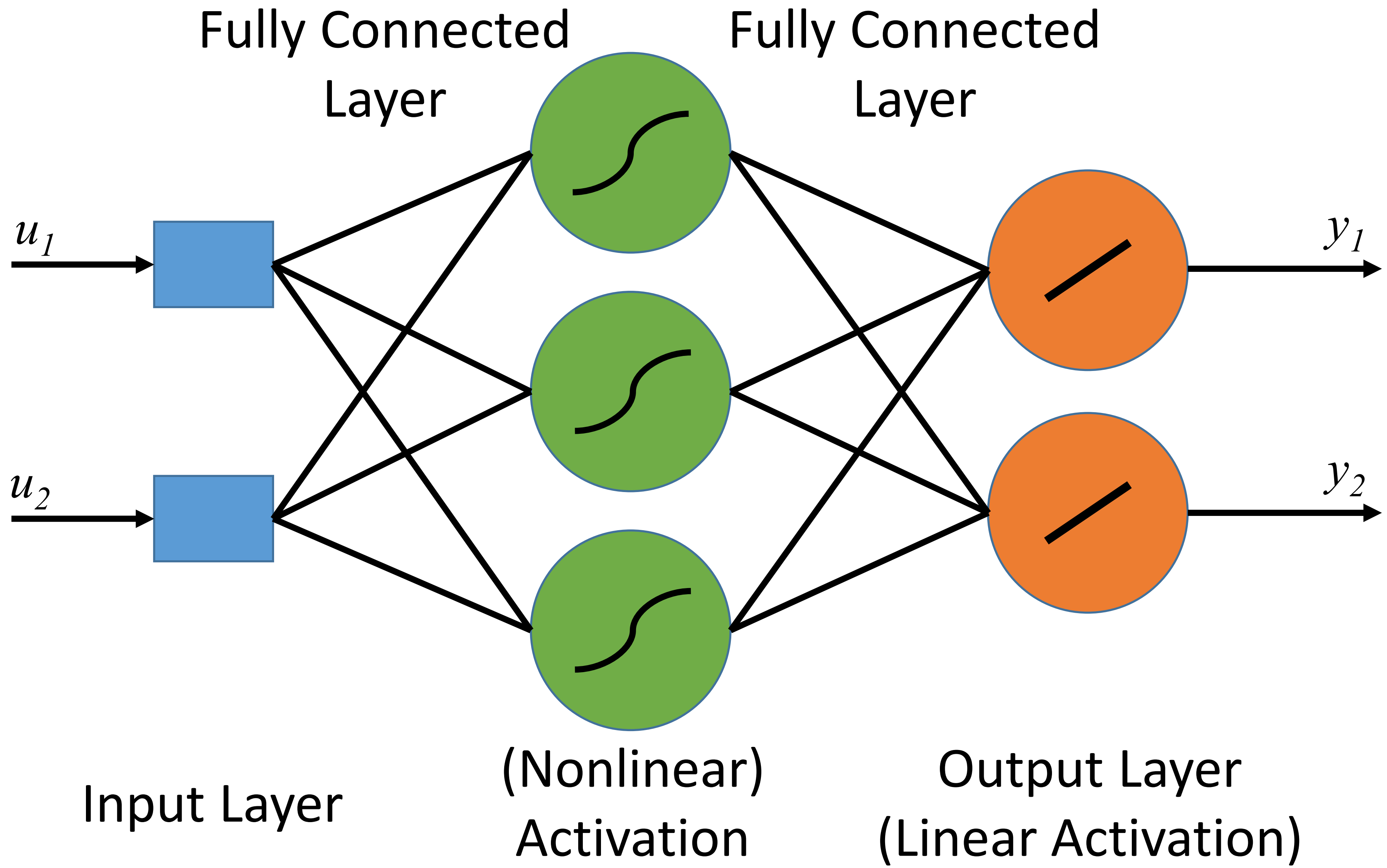
Approximation properties

Training an artificial neural network

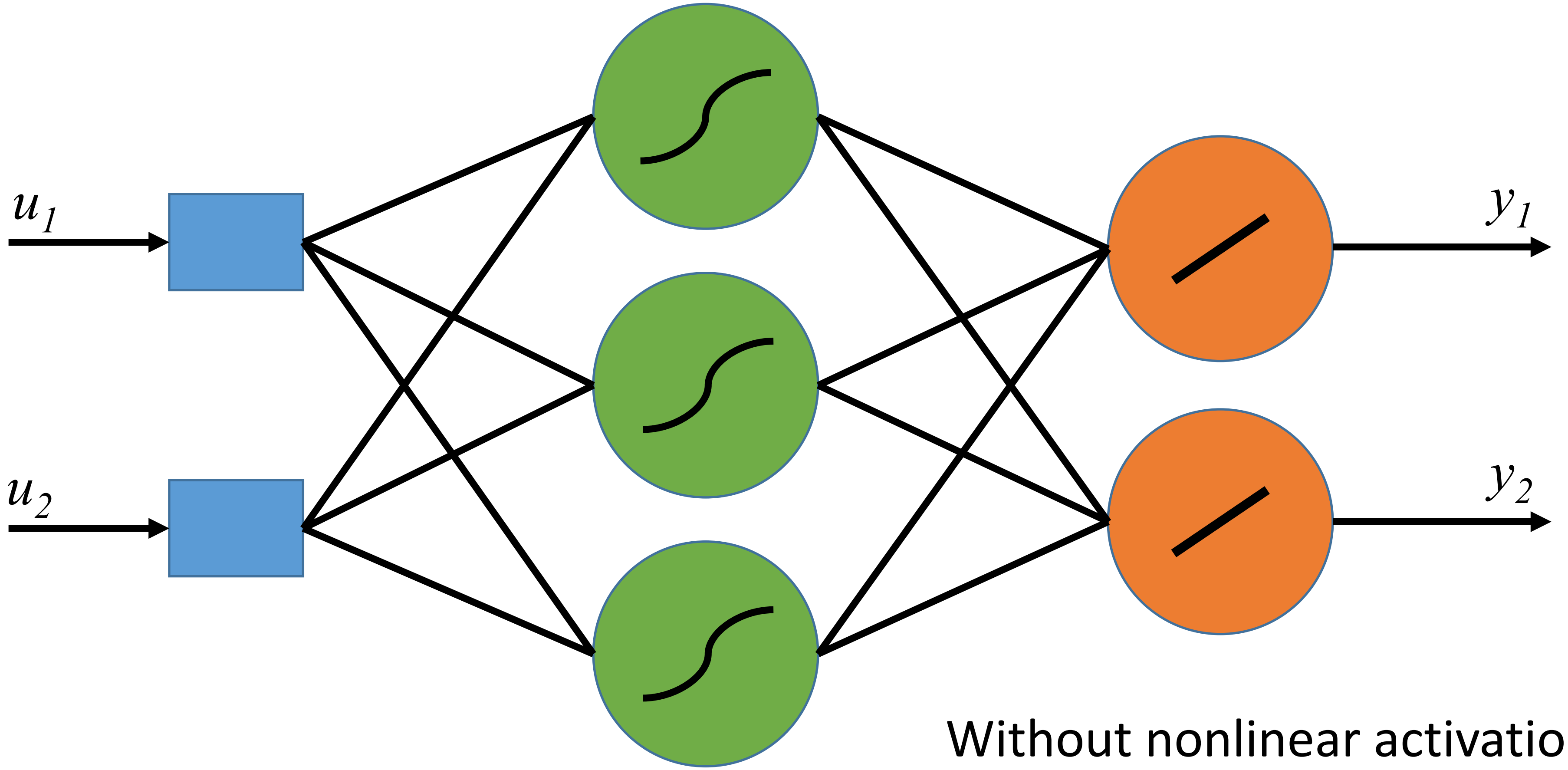
One Hidden Layer Network



One Hidden Layer Network



One Hidden Layer Network

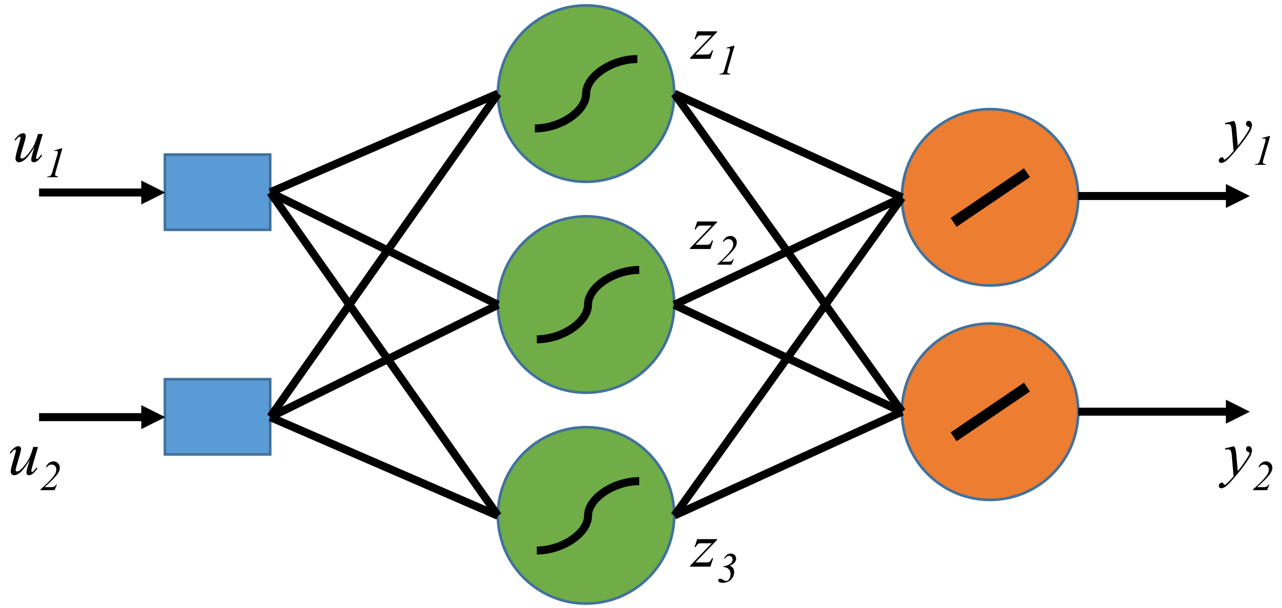


(Nonlinear) Activation

Without nonlinear activation layer this would be a simple linear/affine mapping

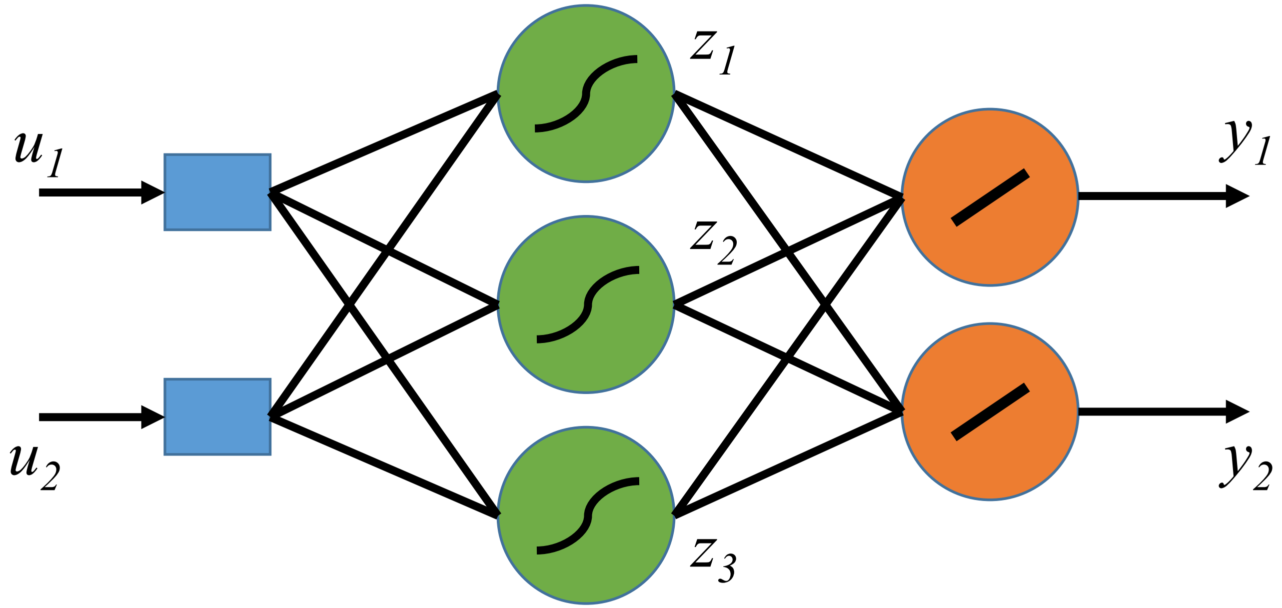
One Hidden Layer Network

Network output



One Hidden Layer Network

Network output



Output of **hidden layer** neuron j

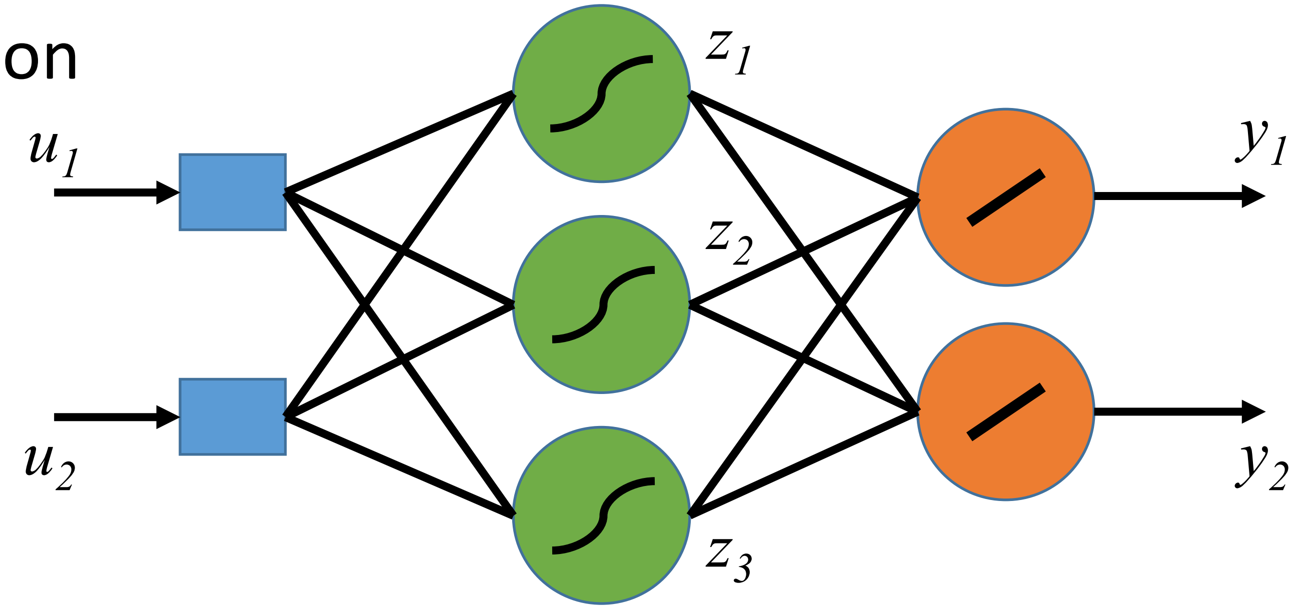
$$z_j = g\left(\boldsymbol{w}_{1,j}^T \boldsymbol{u} + b_{1,j}\right)$$

Output of linear **output layer** neuron j

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

One Hidden Layer Network

Network output – matrix notation



Output of hidden layer

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

Output of linear output layer

$$\boldsymbol{y} = \boldsymbol{W}_2 \boldsymbol{z} + \boldsymbol{b}_2$$

$$\boldsymbol{W}_i = \begin{bmatrix} \boldsymbol{w}_{i,1} & \boldsymbol{w}_{i,2} & \dots & \boldsymbol{w}_{i,n_i} \end{bmatrix}^T$$

$$\boldsymbol{b}_i = \begin{bmatrix} b_{i,1} & b_{i,2} & \dots & b_{i,n_i} \end{bmatrix}^T$$

n_i = number of neurons in layer i

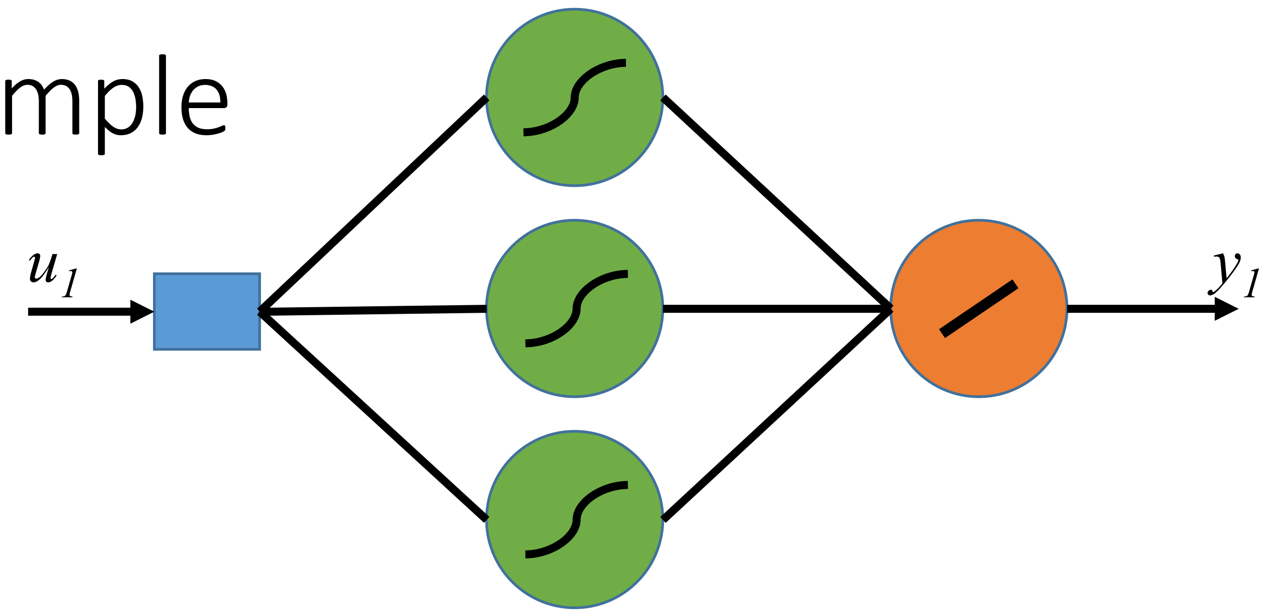
Network Output - Example

Output of hidden layer neuron j

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

Output of linear output layer neuron j

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



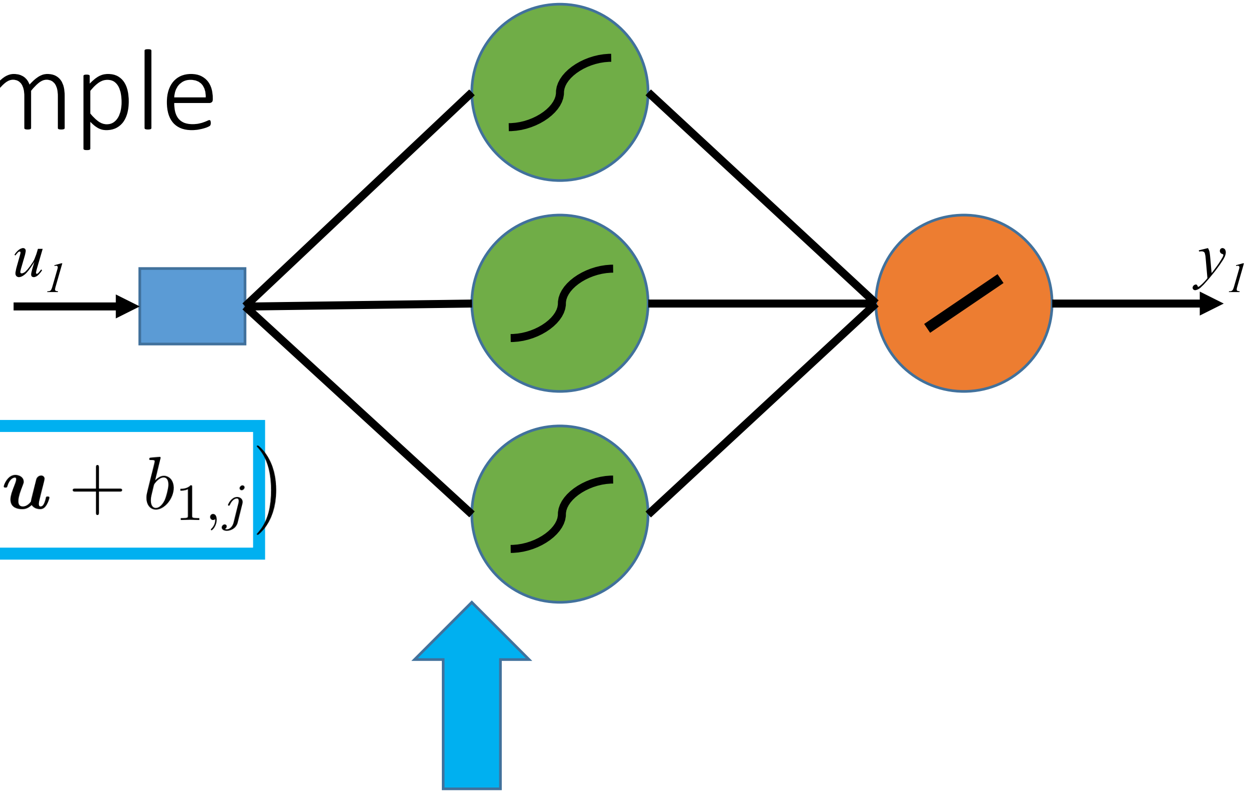
$\boldsymbol{w}_1$ = [1 -3 0.2]
$\boldsymbol{b}_1$ = [0 -1 0.4]

$\boldsymbol{w}_2$ = [1 2 0.5]
$\boldsymbol{b}_2$ = -1.5

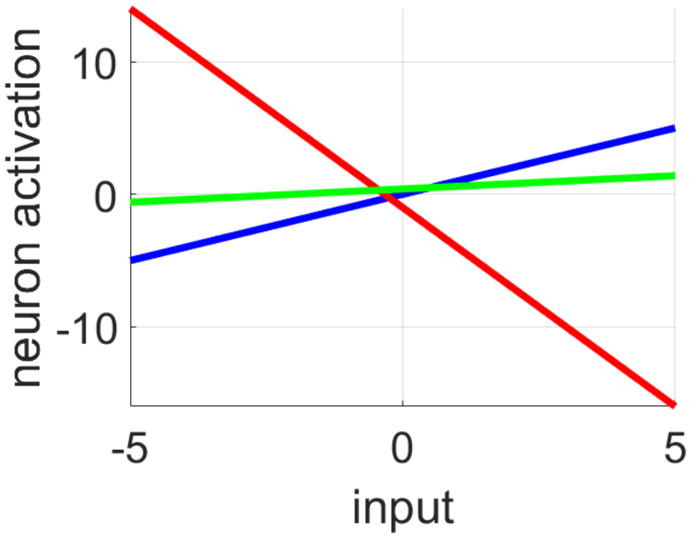
Network Output - Example

$\boldsymbol{w}_1$ = [1 -3 0.2]
$\boldsymbol{b}_1$ = [0 -1 0.4]
$\boldsymbol{w}_2$ = [1 2 0.5]
$\boldsymbol{b}_2$ = -1.5

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

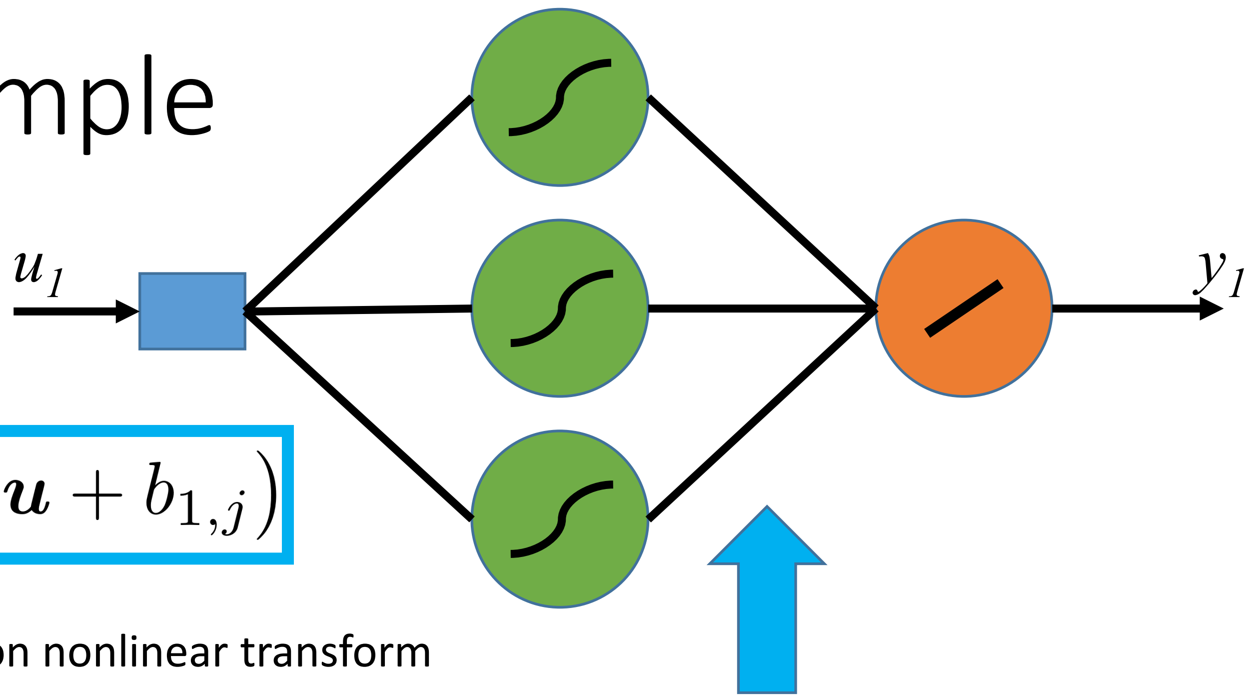


Activation input weighting



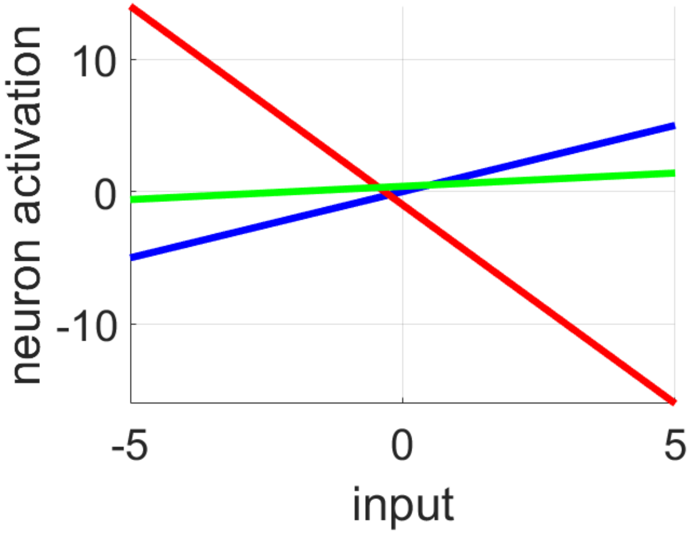
Network Output - Example

$\boldsymbol{w}_1$ = [1 -3 0.2]
$\boldsymbol{b}_1$ = [0 -1 0.4]
$\boldsymbol{w}_2$ = [1 2 0.5]
$\boldsymbol{b}_2$ = -1.5

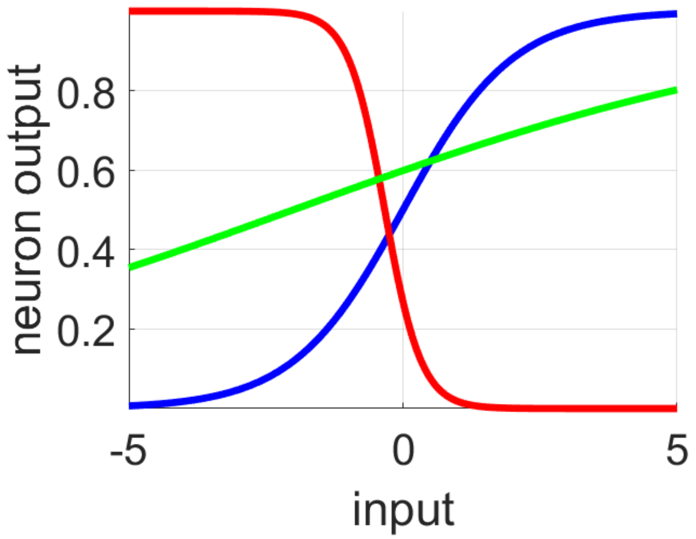


$$z_j = \boxed{g\left(\boldsymbol{w}_{1,j}^T \boldsymbol{u} + b_{1,j}\right)}$$

Activation input weighting



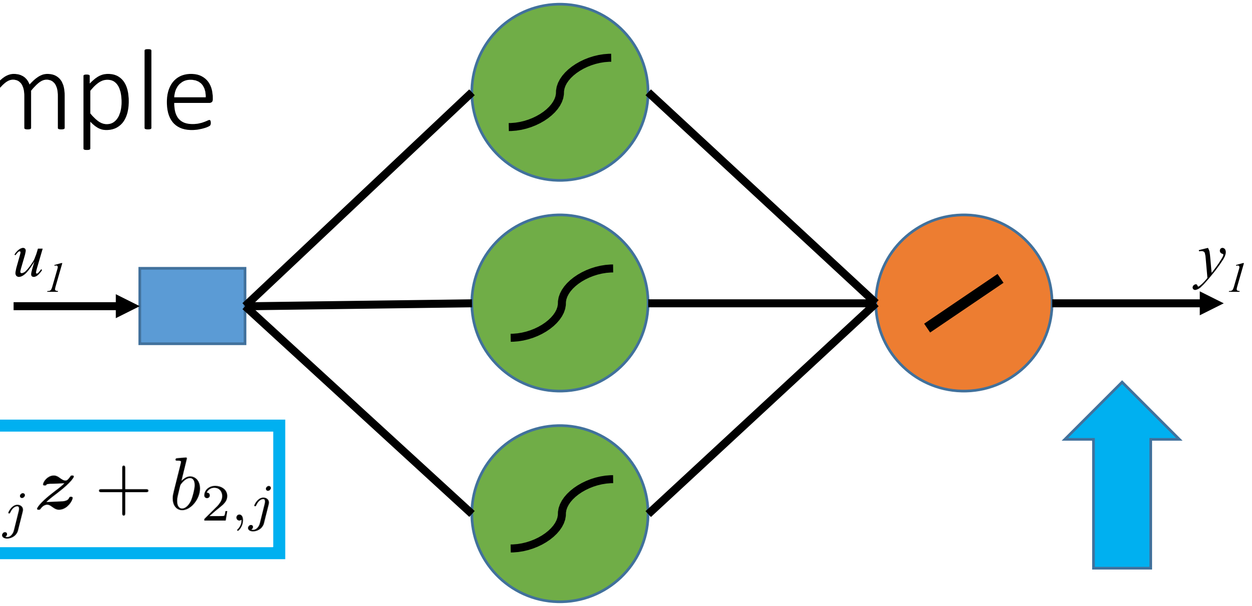
Activation nonlinear transform

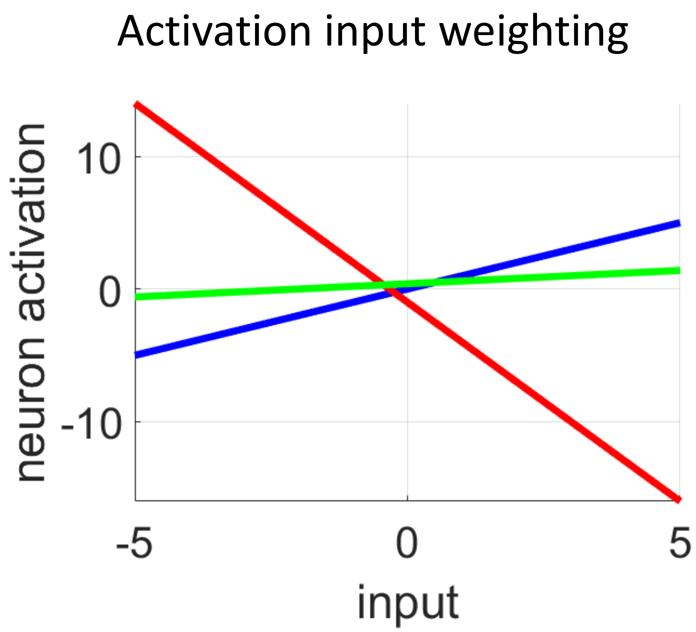


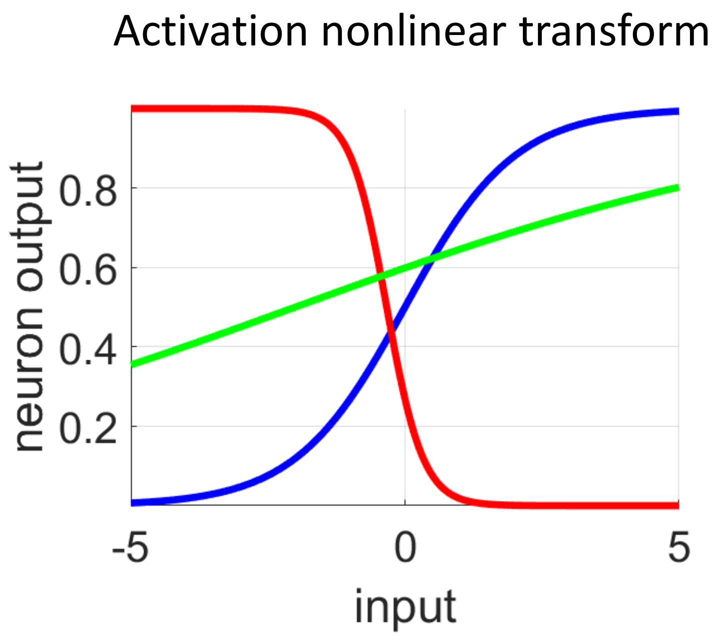
Network Output - Example

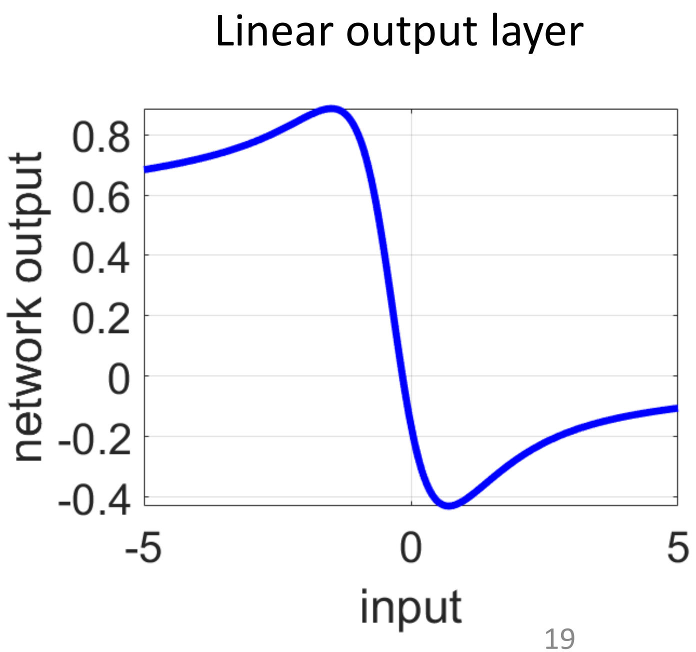
$\boldsymbol{w}_1$ = [1 -3 0.2]
$\boldsymbol{b}_1$ = [0 -1 0.4]
$\boldsymbol{w}_2$ = [1 2 0.5]
$\boldsymbol{b}_2$ = -1.5

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

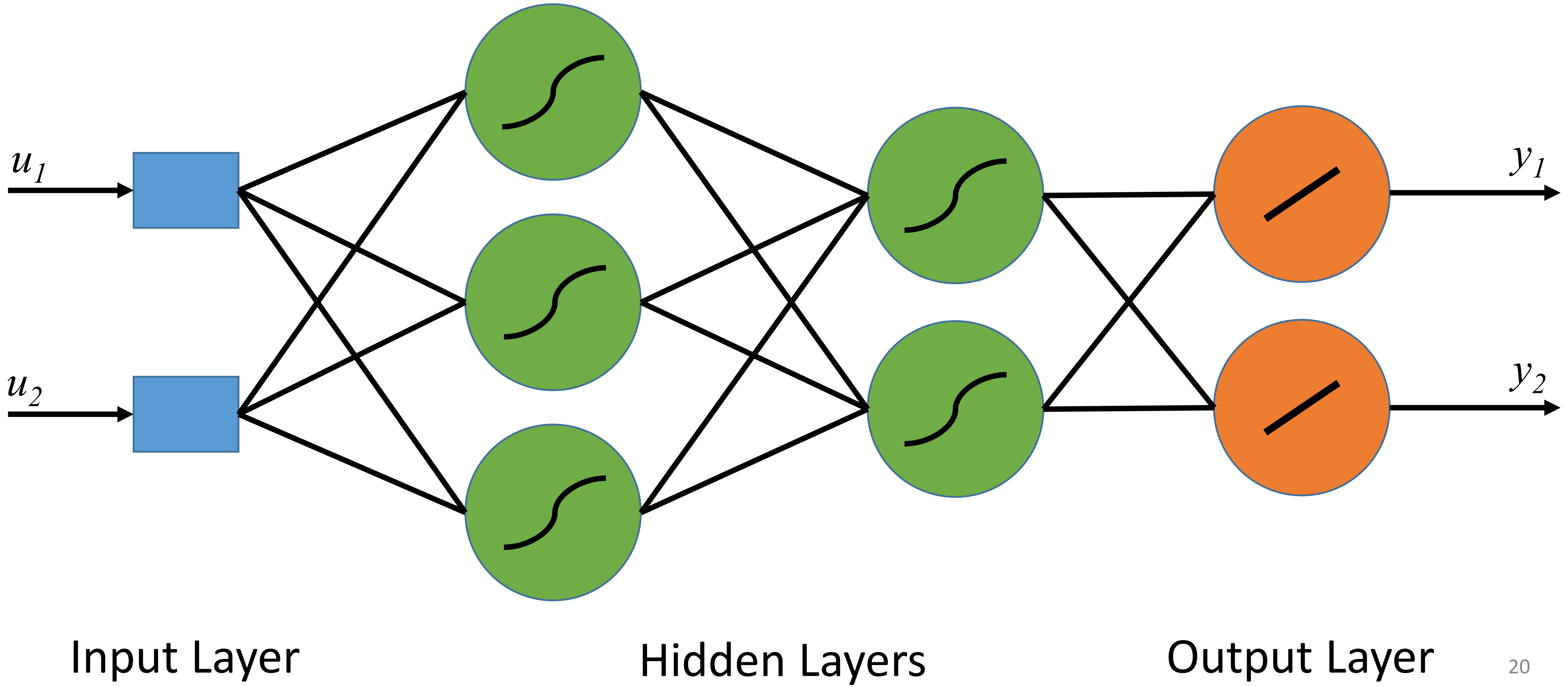








Two Hidden Layer Network



Artificial Neural Networks

What is an artificial neural network?

Simple feedforward networks

Approximation properties

Training an artificial neural network

Approximation Properties

[Cybenko, 1989]: A **feedforward sigmoidal neural net** with at least one hidden layer can approximate any continuous nonlinear function $\mathbb{R}^{n_x} \to \mathbb{R}^{n_y}$ arbitrarily well, provided that sufficient number of hidden neurons are available.

Polynomial or Neural Network?

Which one approximates best a function $f : \mathbb{R} \to \mathbb{R}$?

A: Neural Net

B: Polynomial basis functions

Polynomial or Neural Network?

Which one approximates best a function $f : \mathbb{R} \to \mathbb{R}$?

A: Neural Net B: Polynomial basis functions

Which one approximates best a function $f : \mathbb{R}^2 \to \mathbb{R}$?

A: Neural Net B: Polynomial basis functions

Polynomial or Neural Network?

Which one approximates best a function $f : \mathbb{R} \to \mathbb{R}$?

A: Neural Net B: Polynomial basis functions

Which one approximates best a function $f : \mathbb{R}^2 \to \mathbb{R}$?

A: Neural Net B: Polynomial basis functions

Which one approximates best a function $f : \mathbb{R}^4 \to \mathbb{R}$?

A: Neural Net B: Polynomial basis functions

Polynomial or Neural Network?

Which one approximates best a function $f : \mathbb{R} \to \mathbb{R}$?

B

A: Neural Net B: Polynomial basis functions

Which one approximates best a function $f : \mathbb{R}^2 \to \mathbb{R}$?

A & B

A: Neural Net B: Polynomial basis functions

Which one approximates best a function $f : \mathbb{R}^4 \to \mathbb{R}$?

A

A: Neural Net B: Polynomial basis functions

Approximation Properties

[Barron, 1993]: A **feedforward sigmoidal neural net** with one hidden layer can achieve an integrated squared error of the order

$$V = \mathcal{O}\left(\frac{1}{n}\right)$$

independently of the dimension of the input space, where n denotes the number of hidden neurons.

For a **basis function expansion** (e.g. multivariate polynomial) with n terms, in which only the parameters of the linear combination are adjusted the integrated squared error is of the order

$$V = \mathcal{O}\left(\frac{1}{n^{2/n_x}}\right)$$

where n_x denotes the number of input variables.

Example

Function of 2 variables $(n_x = 2)$

ANN: $V = \mathcal{O}\left(\frac{1}{n}\right)$

POL: $V = \mathcal{O}\left(\frac{1}{n}\right)$

Same behavior for both

Function of 10 variables $(n_x = 10)$

ANN: $V = \mathcal{O}\left(\frac{1}{n}\right)$

POL: $V = \mathcal{O}\left(\frac{1}{n^{2/10}}\right)$

Approximation error decreases much more rapidly for neural networks

Example

Approximation error decreases much more rapidly for neural networks

Function of 2 variables $(n_x = 2)$

ANN: $V = \mathcal{O}\left(\frac{1}{n}\right)$

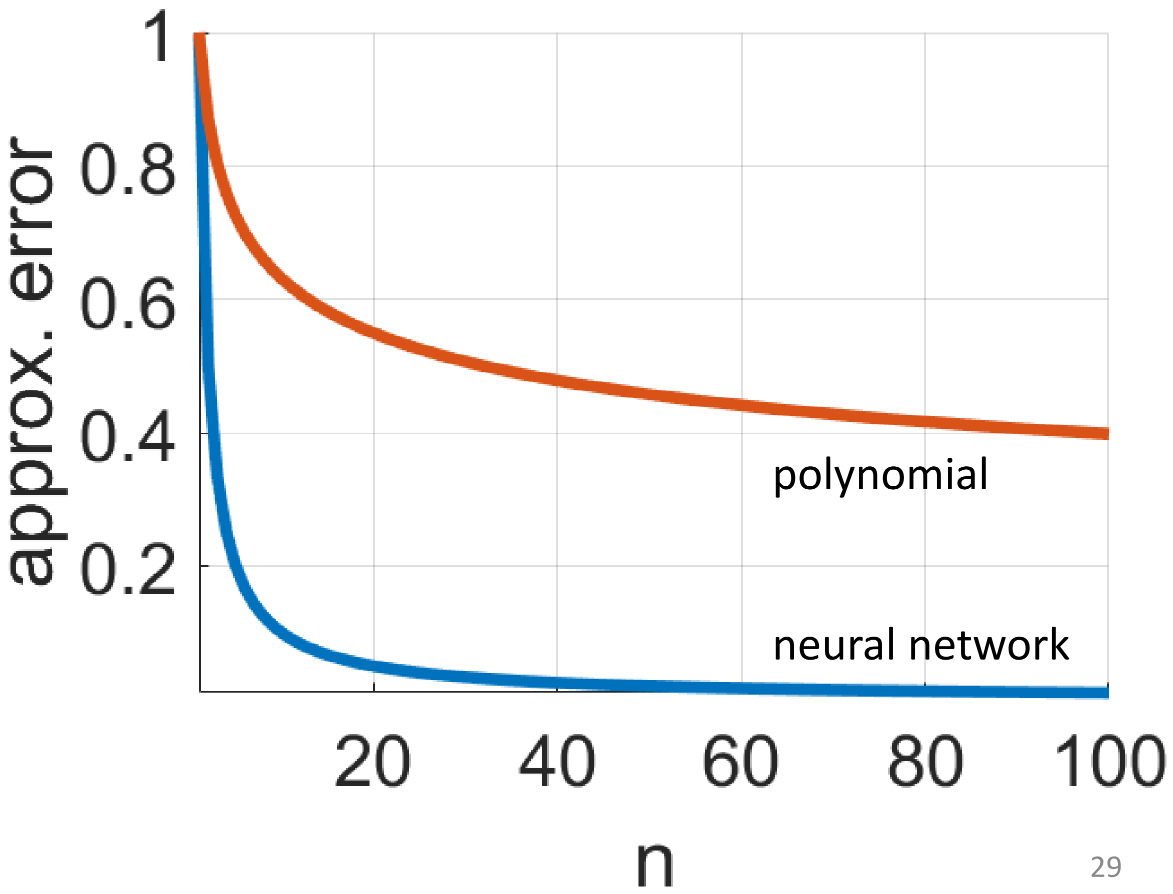
POL: $V = \mathcal{O}\left(\frac{1}{n}\right)$

Function of 10 variables $(n_x = 10)$

ANN: $V = \mathcal{O}\left(\frac{1}{n}\right)$

POL: $V = \mathcal{O}\left(\frac{1}{n^{2/10}}\right)$

10 input variables



Interpretation

Neural networks are well-suited for high-dimensional problems

Does not mean 1-hidden layer is always the best choice

Does not mean the error will always improve by adding more neurons

Does not mean a neural network is always better than basis function expansion

Artificial Neural Networks

What is an artificial neural network?

Simple feedforward networks

Approximation properties

Training an artificial neural network

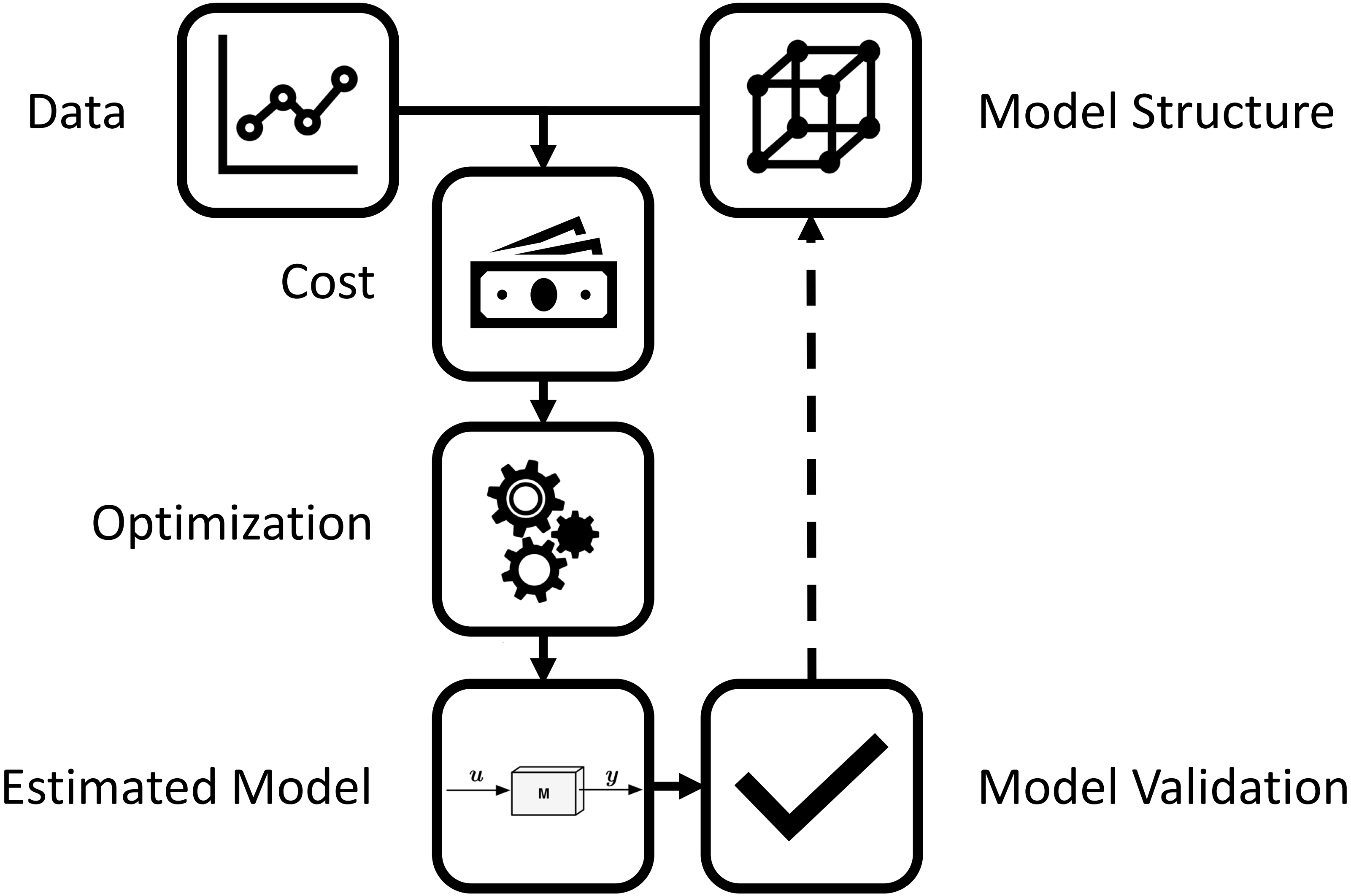
Training A Neural Network

Cost Function

Gradient Based Methods

Backpropagation

Data-Driven Modelling Process



Model Structure

Decisions to make:

network structure

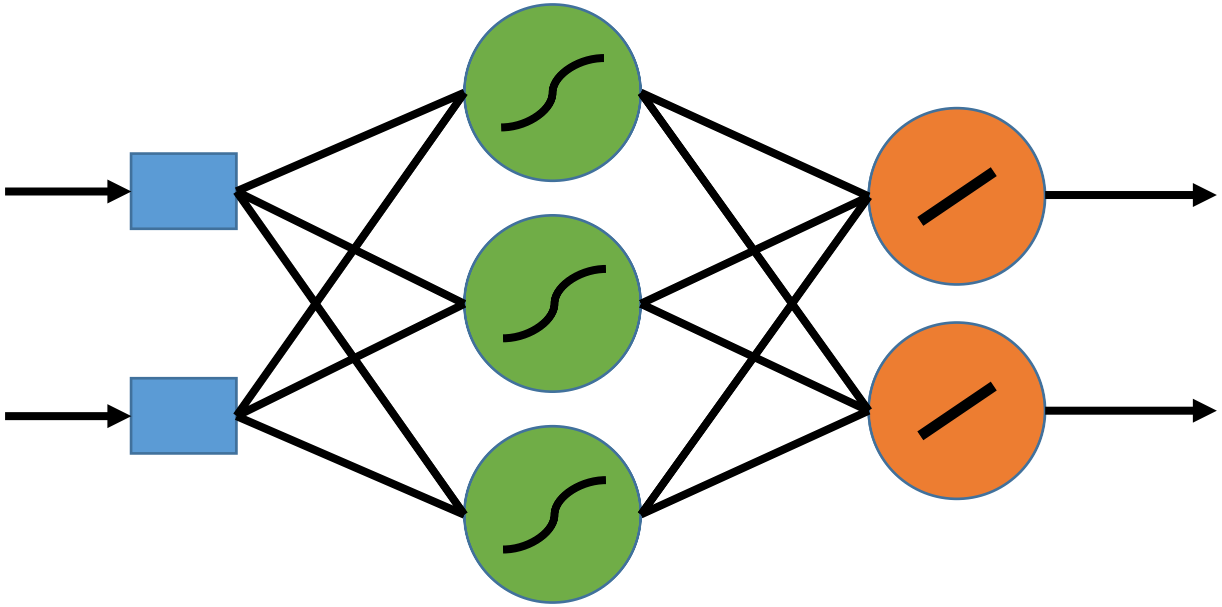
hidden layers

neurons

activation function

Parameters to estimate:

weights and biases

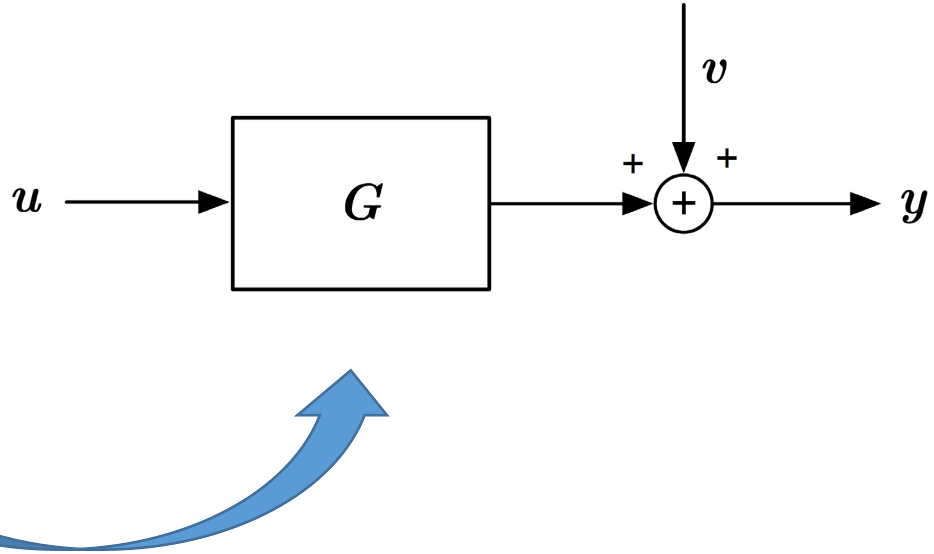


Cost Function

The cost function describes the objective we want to minimize or maximize

Describes how well the model matches the data

Can be extended with extra penalty terms (regularization)

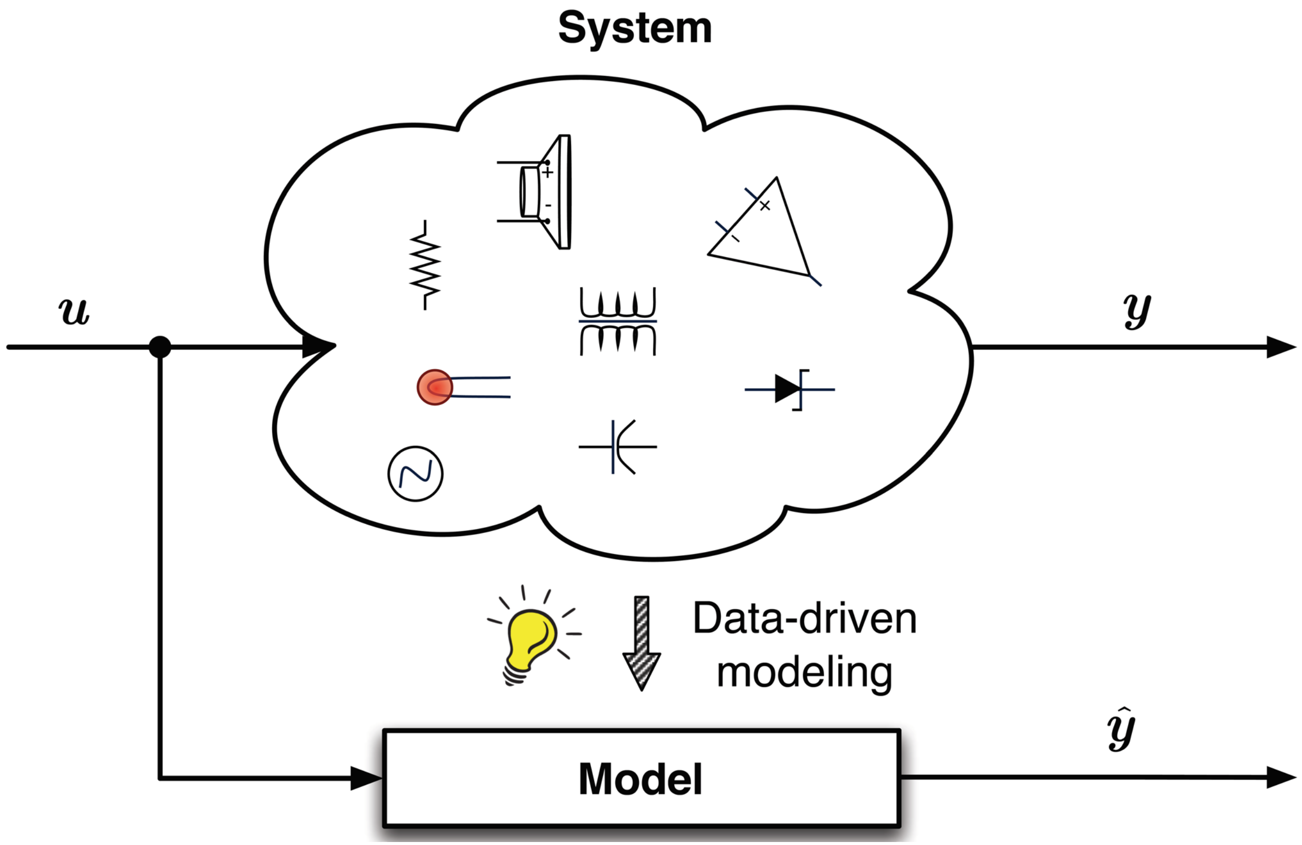


Cost Function
Objective Function
Loss Function

Cost Function

Squared L^2 Norm

$$V_N(\theta) = \frac{1}{N}\sum_{k=0}^{N-1}(y_k - \hat{y}_k)^2$$



Supervised learning is considered here (dataset with labeled input and output)

Estimator

$$\begin{aligned}\hat{\theta}_N &= \arg\min_{\theta\in\Theta} \frac{1}{N}\sum_{k=0}^{N-1}(y_k - \hat{y}_k)^2 \\ &= \arg\min_{\theta\in\Theta} V_N(\theta)\end{aligned}$$

$\hat{y}_k$	output of the artificial neural network
θ	stacked parameter vector containing all the network weights and biases

How to minimize?

Nonlinear Optimization

$$\begin{aligned}\hat{\theta}_N &= \arg\min_{\theta\in\Theta} \frac{1}{N}\sum_{k=0}^{N-1}(y_k - \hat{y}_k)^2 \\ &= \arg\min_{\theta\in\Theta} V_N(\theta)\end{aligned}$$

Nonlinear in the parameters

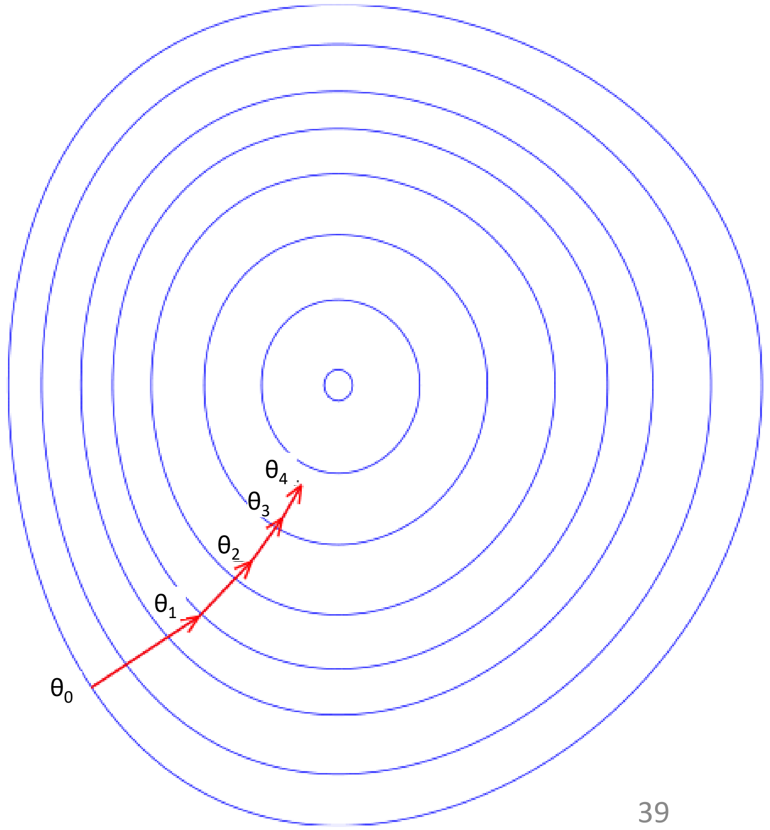
Differentiable

Steepest Descent

$$\hat{\theta}_N = \arg\min_{\theta \in \Theta} \frac{1}{N} \sum_{k=0}^{N-1} (y_k - \hat{y}_k)^2$$

$$= \arg\min_{\theta \in \Theta} V_N(\theta)$$

1. Make an initial guess
2. Compute the gradients $\nabla_{\theta_i} V_N(\theta)$
3. Parameter update $\theta_{i+1} = \theta_i - \epsilon \nabla_{\theta_i} V_N(\theta)$
4. Check stopping criterion
5. Stop



Steepest Descent

Initial guess?	(small random values)
Step size?	(line search, adaptive rules, second-order methods, …)
Stopping Criterium?	(gradient threshold, cost threshold, early stopping, …)
Gradient Calculation?	(backpropagation)

Gradient Calculation

Automatic gradient calculation for

wide range of model structures

large datasets

→ Flexible and efficient algorithm required

Automatic Differentiation

Functions are composed of

Elementary operations: + - / x …

Elementary functions: sin, log, exp, …

Automatic differentiation exploits the function structure and evaluates the derivative for a given set of function values and parameters

≠ symbolic differentiation

≠ numerical differentiation

Forward vs Reverse Automatic Differentiation

Forward Mode

1. Fix the free variable you want to know the derivative of
2. Perform chain rule
3. Repeat for all free variables

Good for functions with little free variables and many outputs

Reverse Mode

1. Select the function output wrt which you will calculate the derivative
2. Calculate all chain rule elements towards all free variables
3. Repeat for all function outputs

Good for functions with many free variables and little outputs

Backpropagation: Computational Efficiency

graph edges

reverse-mode derivation $\mathcal{O}\left(n_e n\right)$ ⟺ $\mathcal{O}\left(n_e m\right)$ forward-mode derivation

function outputs

function inputs

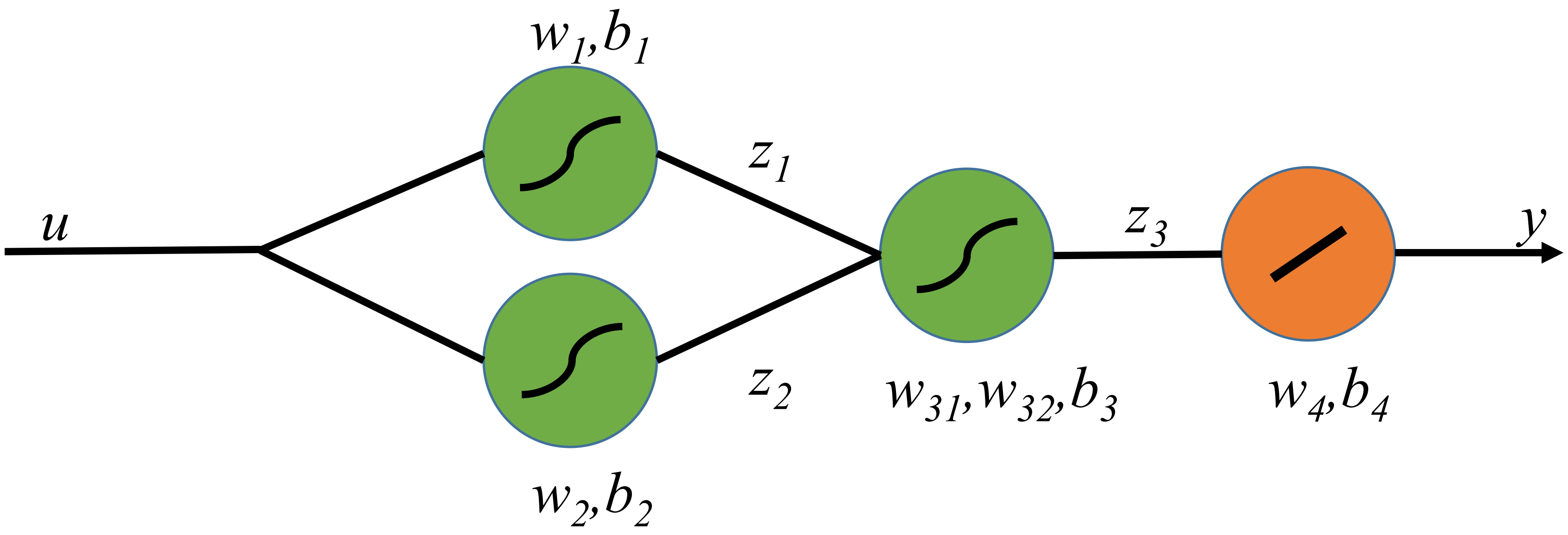
Function: cost function

Function inputs: parameters → Many!

Function outputs: cost → Few!

Graph edges: ≈ANN links

Example



2-Hidden Layer NN

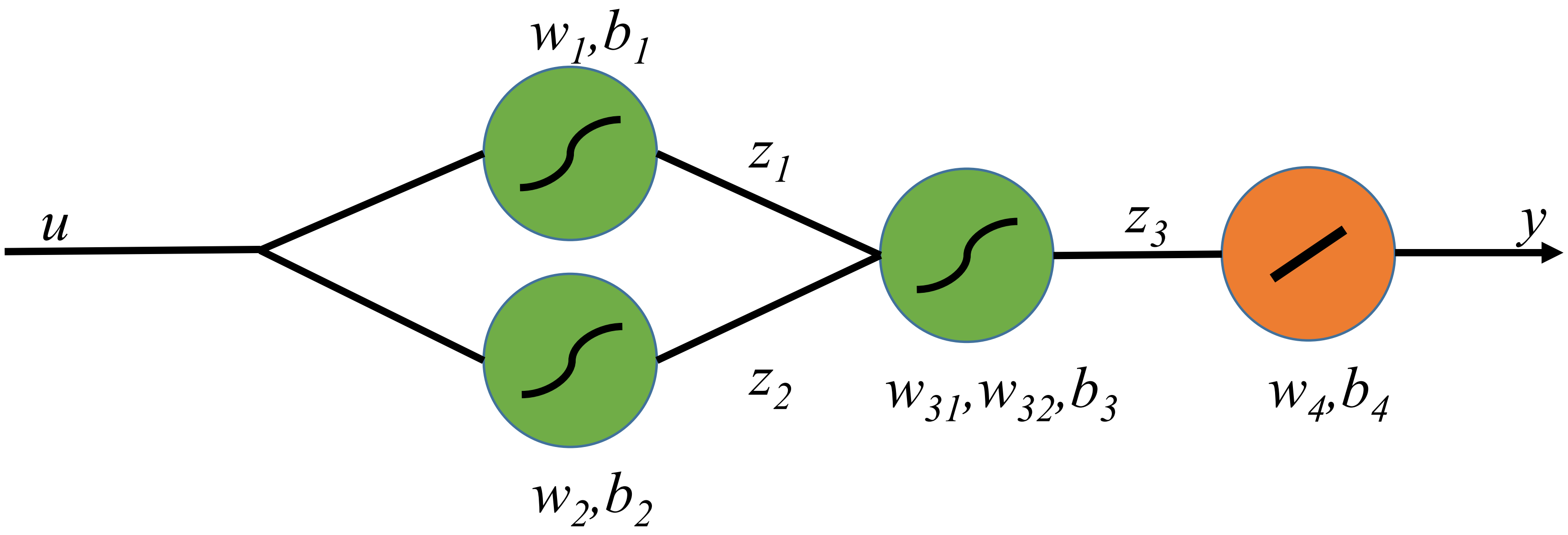
1 input, 1 output

9 parameters

Sigmoid Activation

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

Example



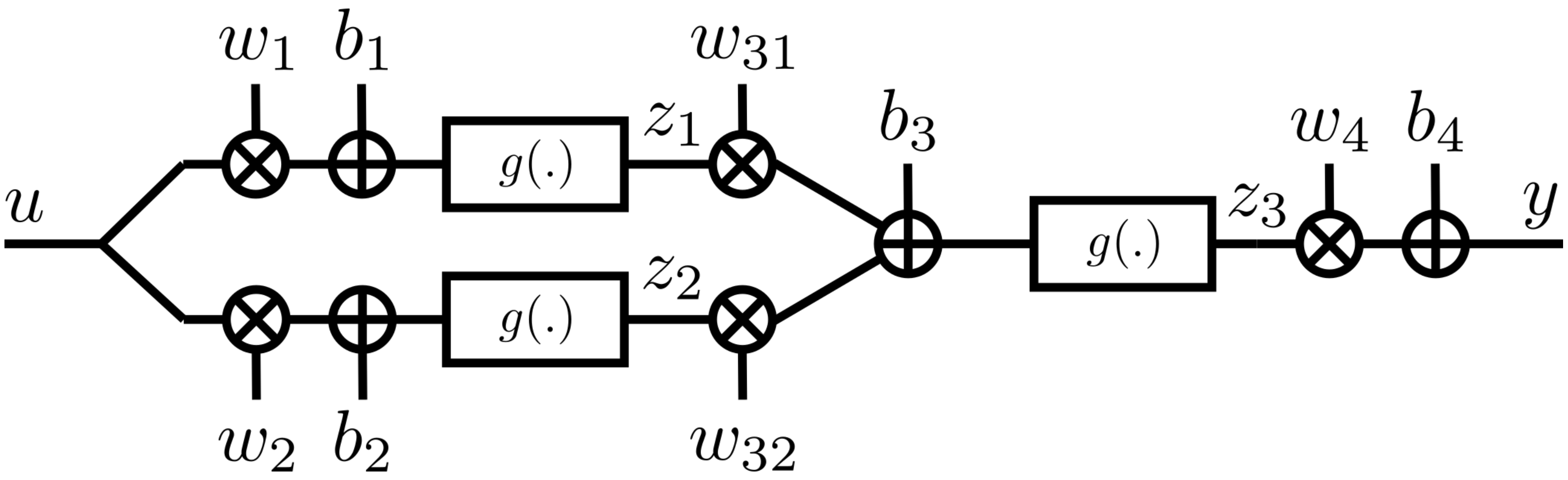
2-Hidden Layer NN

1 input, 1 output

9 parameters

Sigmoid Activation

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



Backpropagation

Automatic differentiation

Reverse mode algorithm

Forward pass

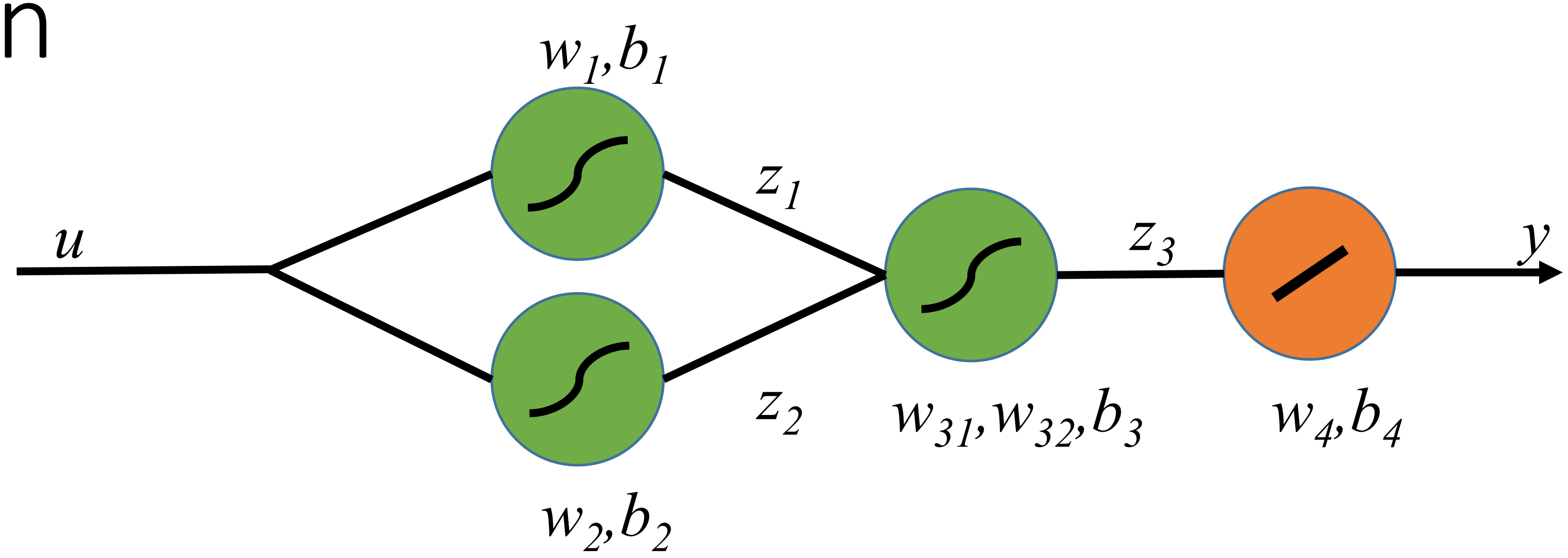
Backward pass

Applying chain rule

Reuse of computations

Backpropagation

$$\hat{\theta}_N = \arg\min_{\theta\in\Theta} \frac{1}{N}\sum_{k=0}^{N-1}(y_k - \hat{y}_k)^2$$
$$= \arg\min_{\theta\in\Theta} V_N(\theta)$$

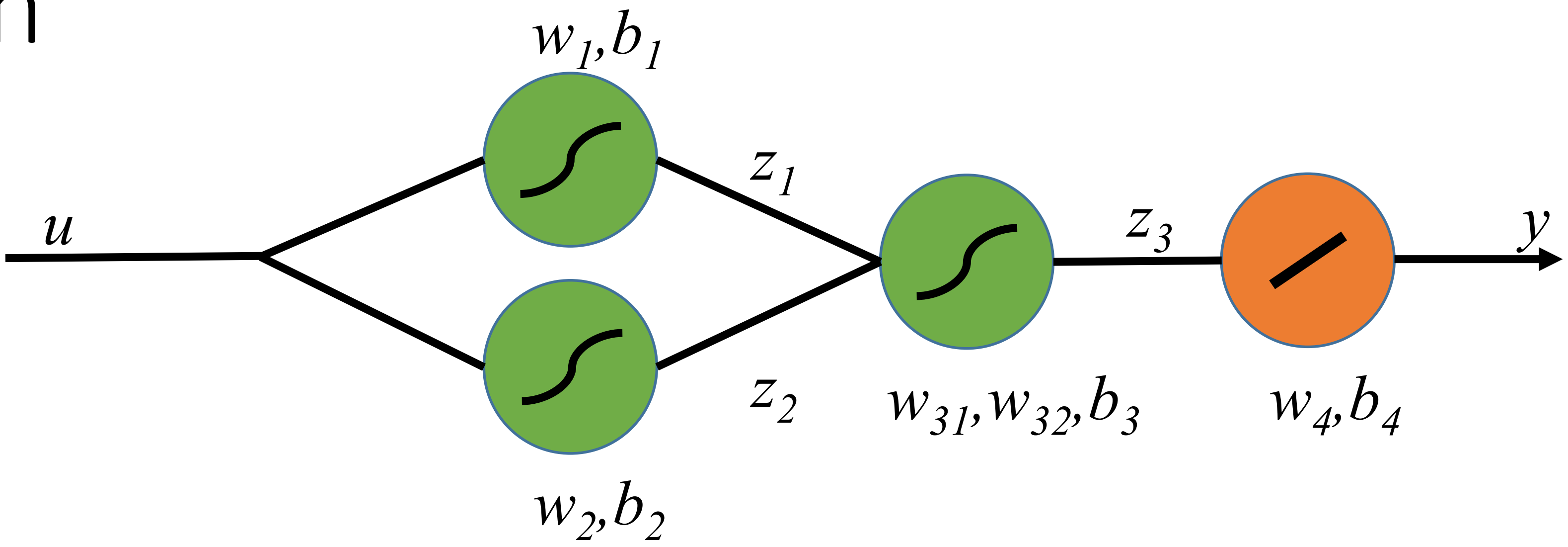


Gradients:

$$\nabla_{\theta_i} V_N(\theta) = \left(\left.\frac{\partial V_N(\theta)}{\partial w_1}\right|_{\theta_i}, \left.\frac{\partial V_N(\theta)}{\partial w_2}\right|_{\theta_i}, \left.\frac{\partial V_N(\theta)}{\partial w_{31}}\right|_{\theta_i}, \left.\frac{\partial V_N(\theta)}{\partial w_{32}}\right|_{\theta_i}, \left.\frac{\partial V_N(\theta)}{\partial w_4}\right|_{\theta_i}, \left.\frac{\partial V_N(\theta)}{\partial b_1}\right|_{\theta_i}, \left.\frac{\partial V_N(\theta)}{\partial b_2}\right|_{\theta_i}, \left.\frac{\partial V_N(\theta)}{\partial b_3}\right|_{\theta_i}, \left.\frac{\partial V_N(\theta)}{\partial b_4}\right|_{\theta_i} \right)$$

Backpropagation

$$\hat{\theta}_N = \arg\min_{\theta\in\Theta} \frac{1}{N}\sum_{k=0}^{N-1}(y_k - \hat{y}_k)^2$$
$$= \arg\min_{\theta\in\Theta} V_N(\theta)$$



$$\left.\frac{\partial V_N(\theta)}{\partial \theta}\right|_{\theta_i} = -\frac{2}{N}\sum_{k=1}^{N}(y_k - \hat{y}_k)\left.\frac{\partial \hat{y}_k}{\partial \theta}\right|_{\theta_i}$$

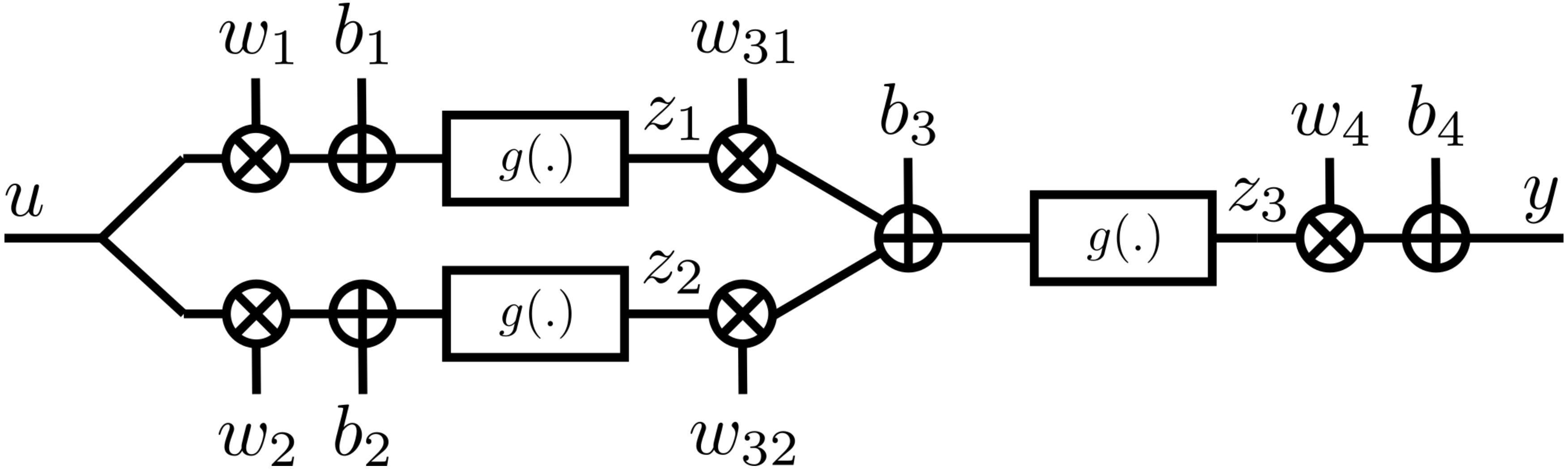
Focus on the partial derivatives of the outputs for simplicity

The notation for evaluating the partial derivative in the i-th iteration of the parameters is dropped from now on
The notation for time-indexing is also dropped

Backpropagation – Fill in Values

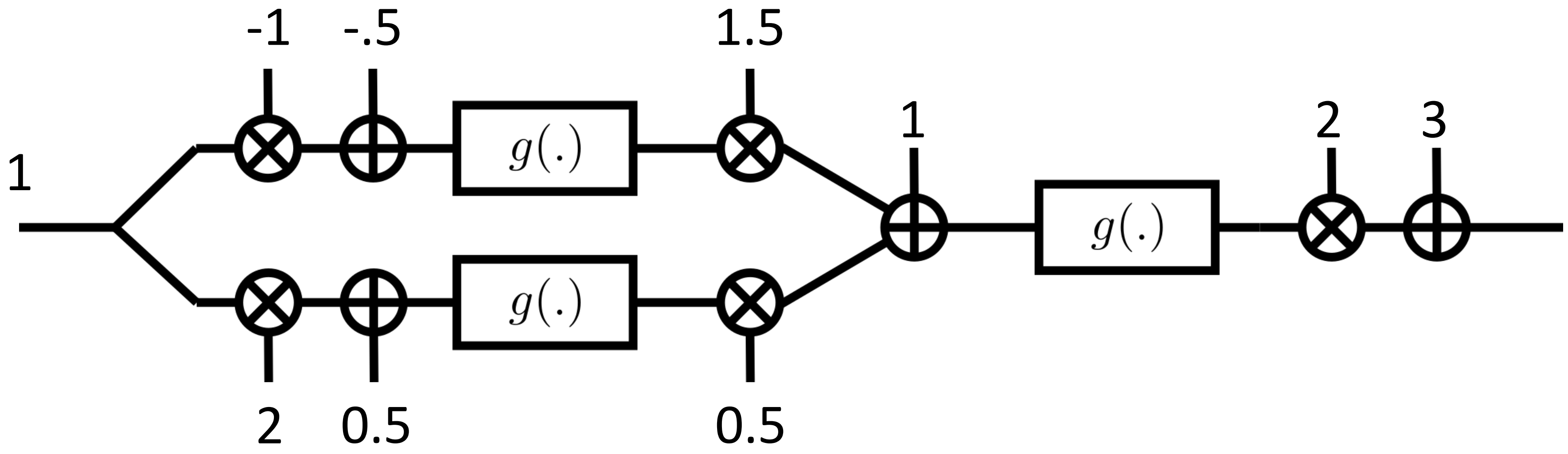
Parameter Values for Iteration i

Input-Output Values for time k



Backpropagation – Fill in Values

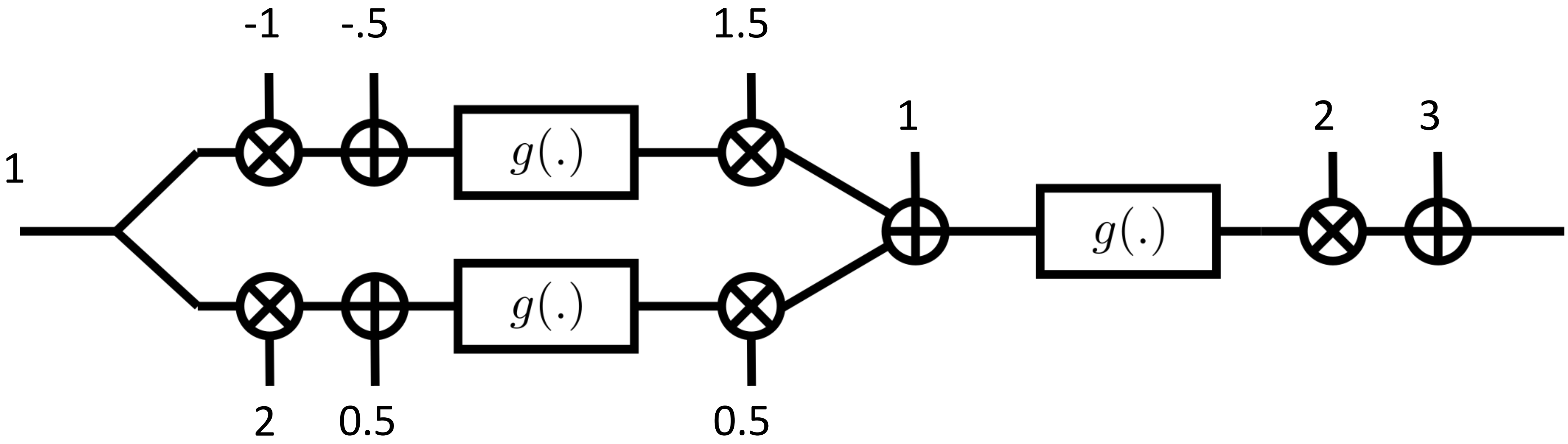
Parameter Values for Iteration i
Input-Output Values for time k



Backpropagation – Forward Pass

Propagate the values forward through the network

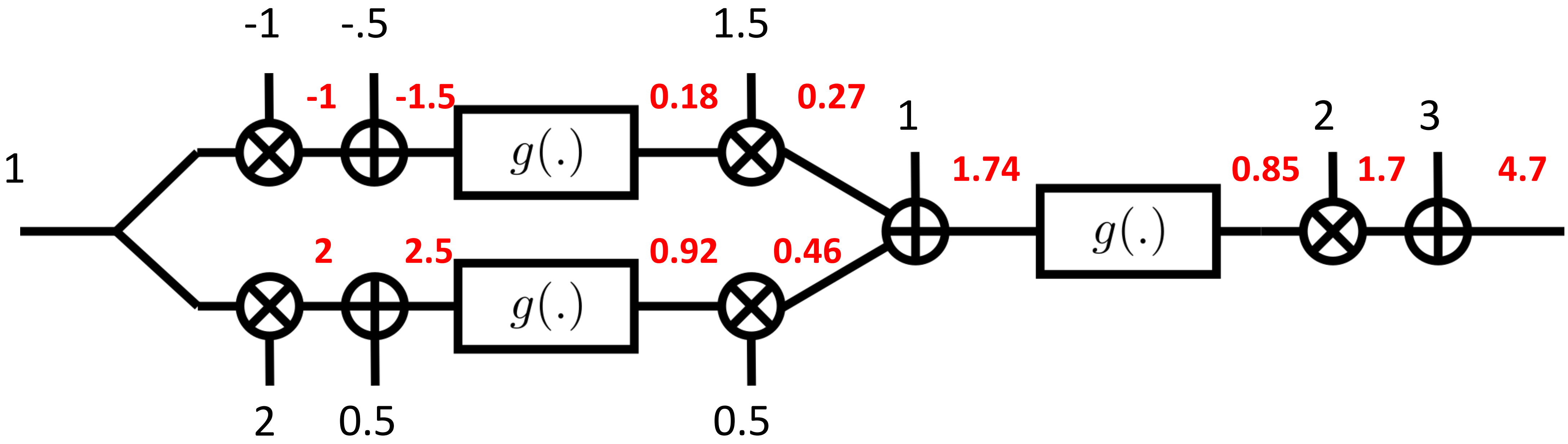
$$g(x) = \frac{1}{1 + e^{-x}}$$



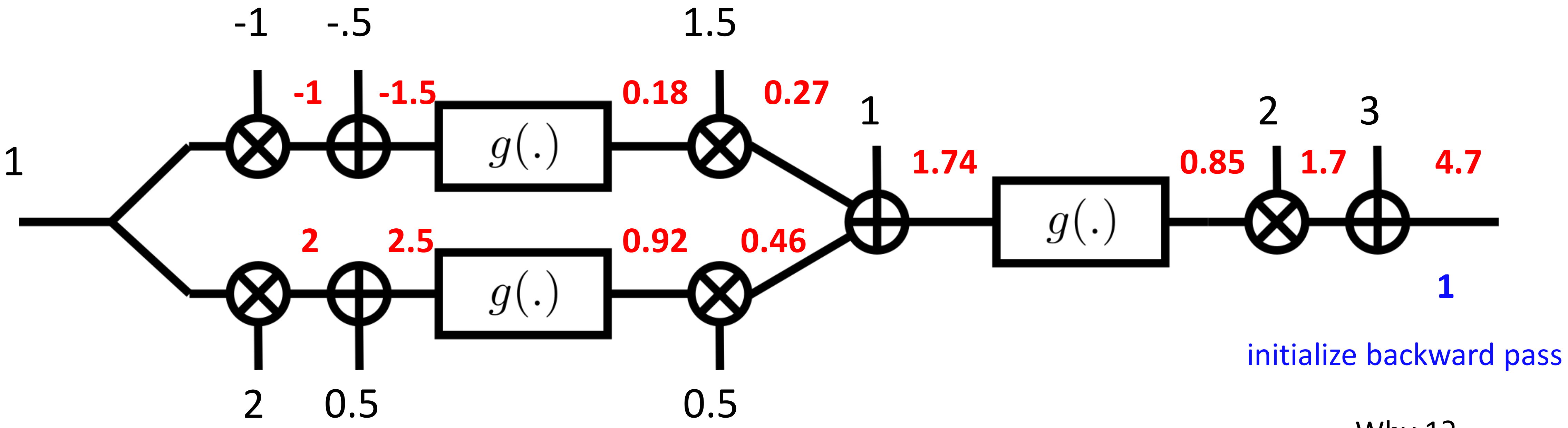
Backpropagation – Forward Pass

Propagate the values forward through the network

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

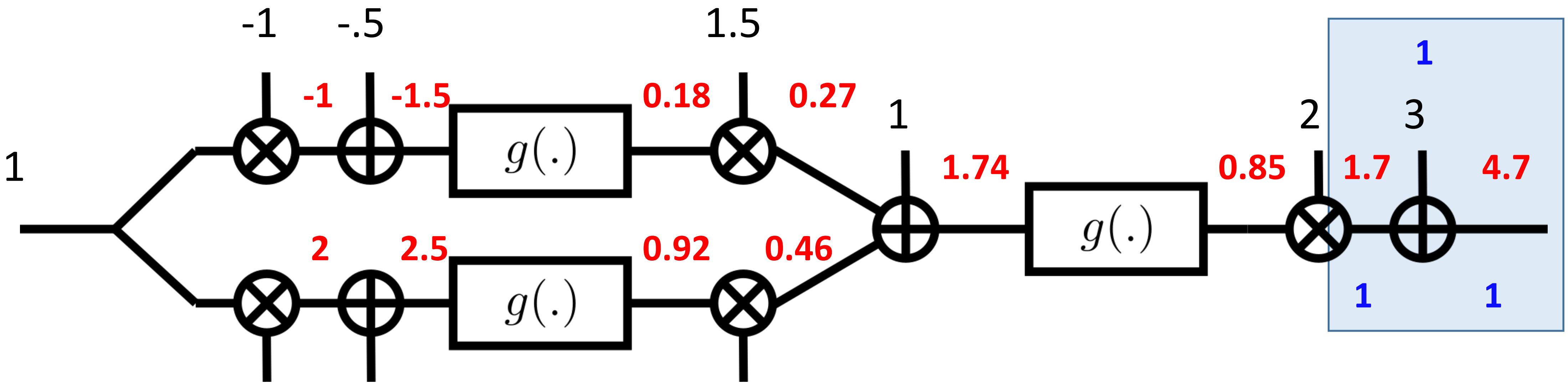


Backpropagation – Backward Pass



Why 1?

Backpropagation – Backward Pass



$$f(x) = x + b_4$$

$$x = 1.7$$

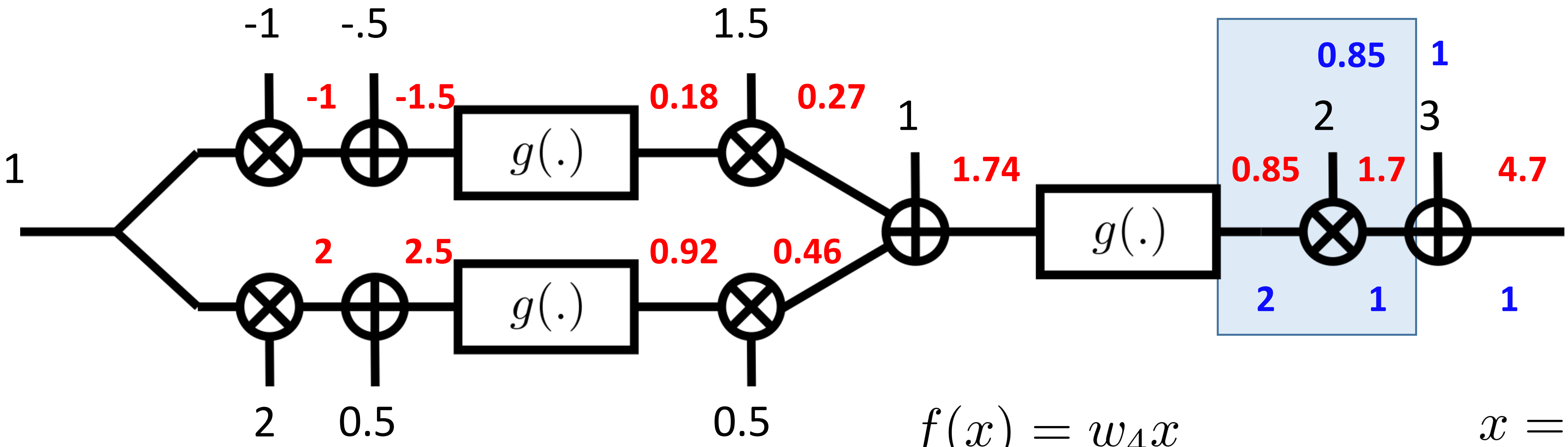
$$b_4 = 3$$

$$\frac{\partial y}{\partial x} = \frac{\partial y}{\partial f(x)} \frac{\partial f(x)}{\partial x}$$

$$\frac{\partial f}{\partial x} = 1, \quad \frac{\partial f}{\partial b_4} = 1$$

Downstream partial derivative (previously computed)

Backpropagation – Backward Pass



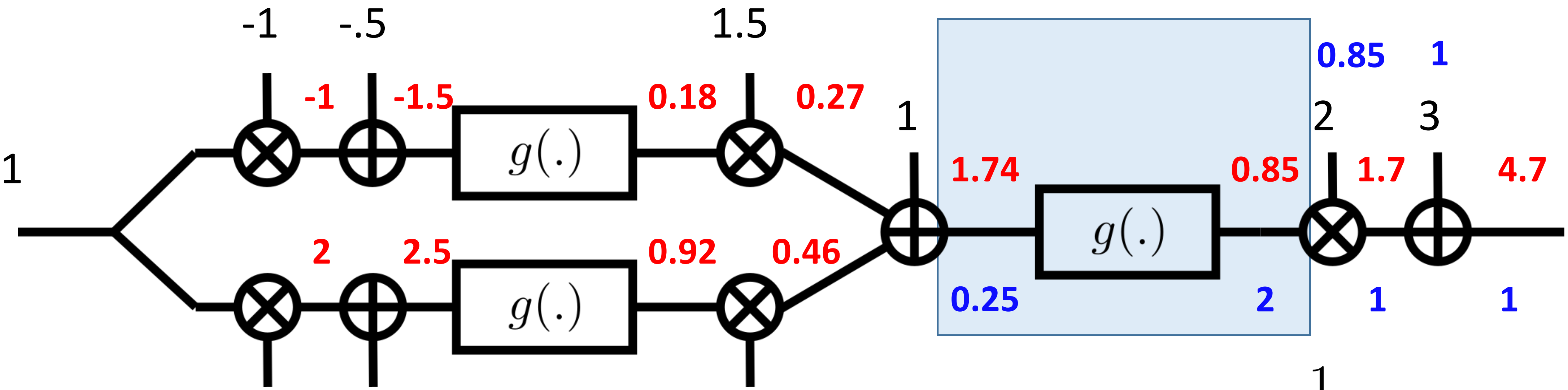
$$f(x) = w_4 x \qquad x = 0.85$$

$$\frac{\partial f}{\partial x} = w_4, \qquad \frac{\partial f}{\partial w_4} = x \qquad w_4 = 2$$

$$\frac{\partial y}{\partial x} = \frac{\partial y}{\partial f(x)} \frac{\partial f(x)}{\partial x}$$

Downstream partial derivative (previously computed)

Backpropagation – Backward Pass



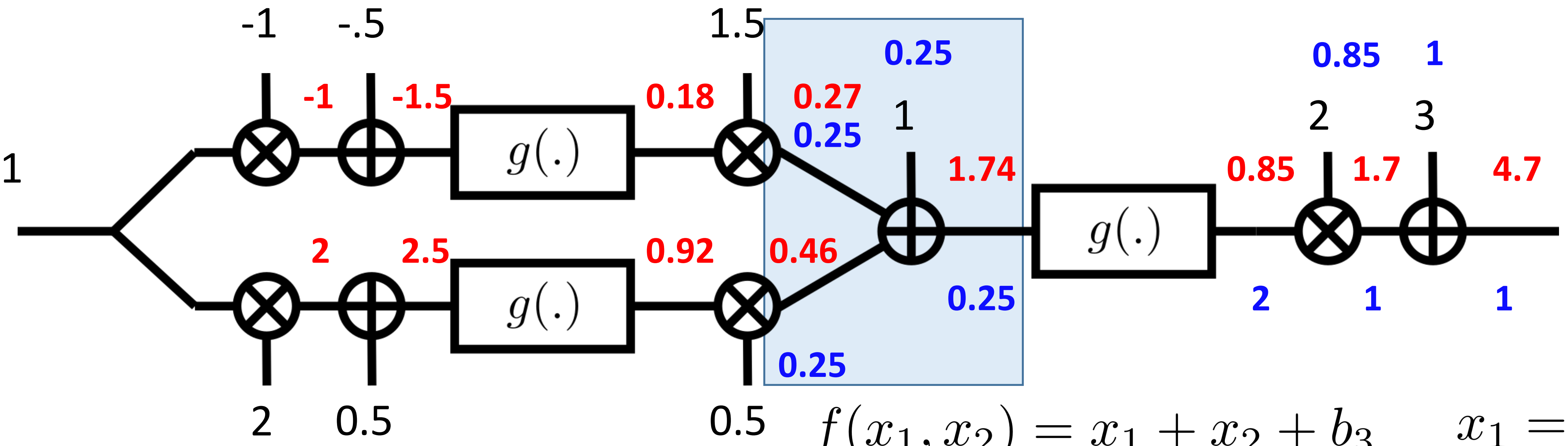
$$f(x) = g(x) = \frac{1}{1+e^{-x}} \qquad x = 1.74$$

$$\frac{\partial y}{\partial x} = \frac{\partial y}{\partial f(x)} \frac{\partial f(x)}{\partial x}$$

$$\frac{\partial f(x)}{\partial x} = \frac{e^x}{(1+e^x)^2} = 0.127$$

Downstream partial derivative (previously computed)

Backpropagation – Backward Pass



$$f(x_1, x_2) = x_1 + x_2 + b_3$$

$$x_1 = 0.27$$

$$x_2 = 0.46$$

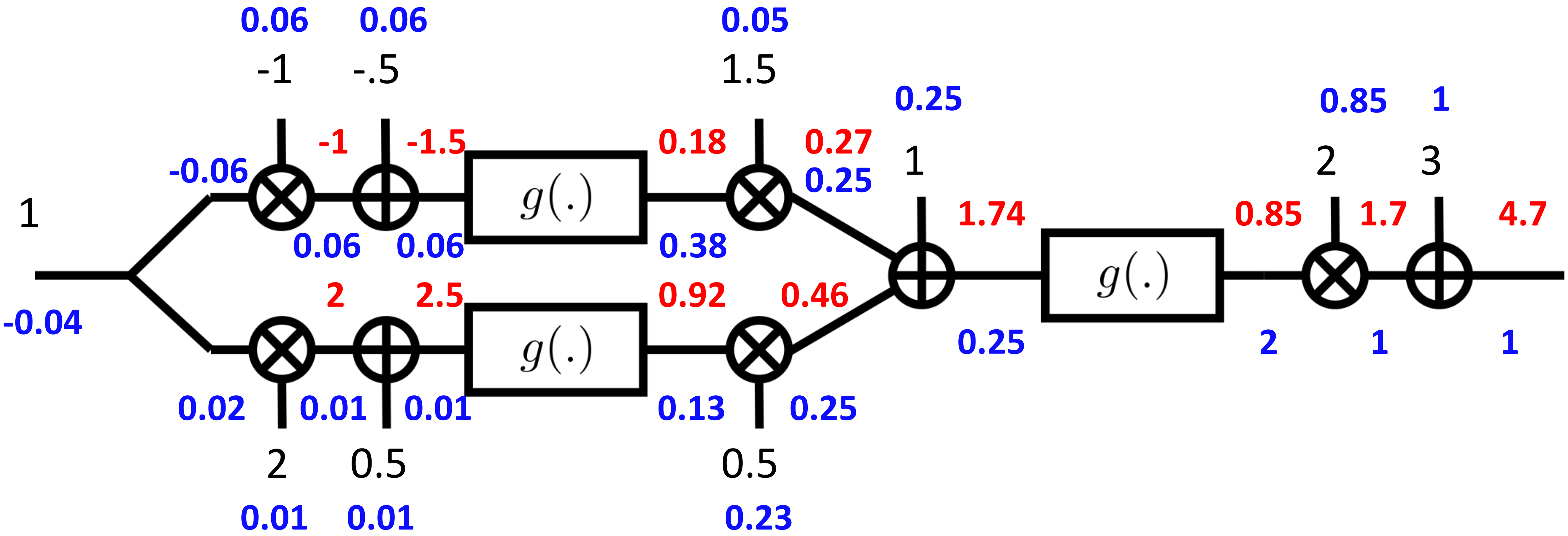
$$b_3 = 1$$

$$\frac{\partial y}{\partial x} = \frac{\partial y}{\partial f(x)} \frac{\partial f(x)}{\partial x}$$

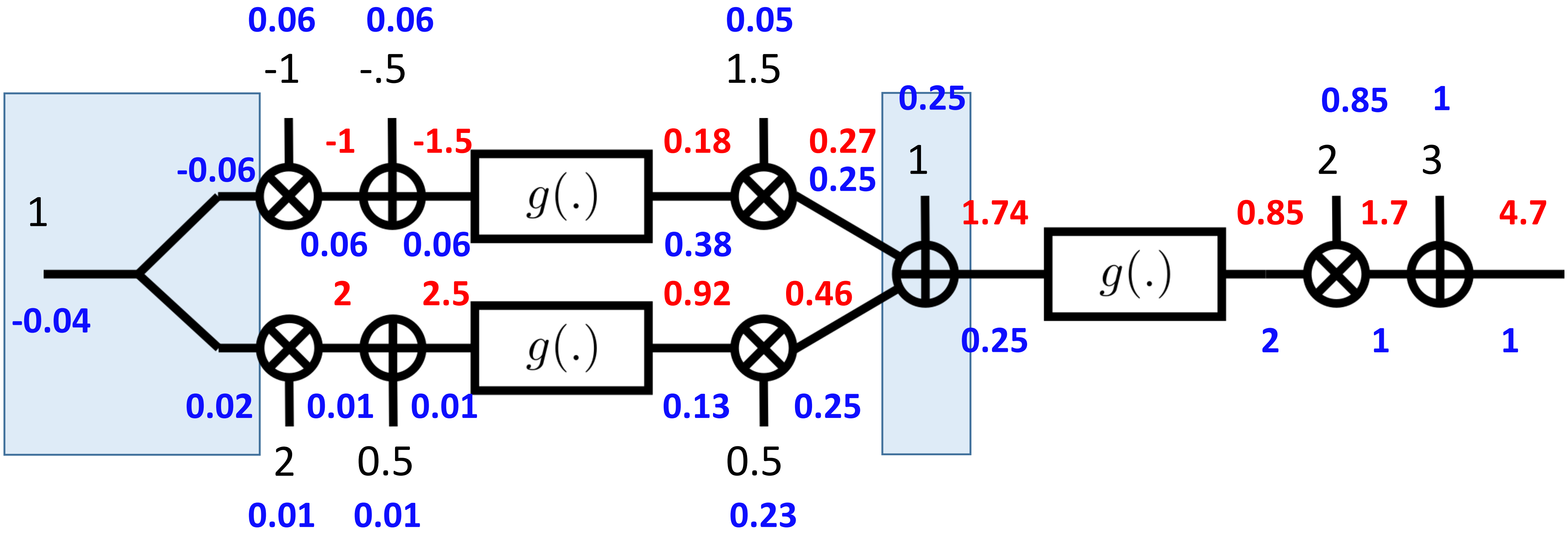
$$\frac{\partial f}{\partial x_1} = 1, \quad \frac{\partial f}{\partial x_2} = 1, \quad \frac{\partial f}{\partial b_3} = 1$$

Downstream partial derivative (previously computed)

Backpropagation – Backward Pass



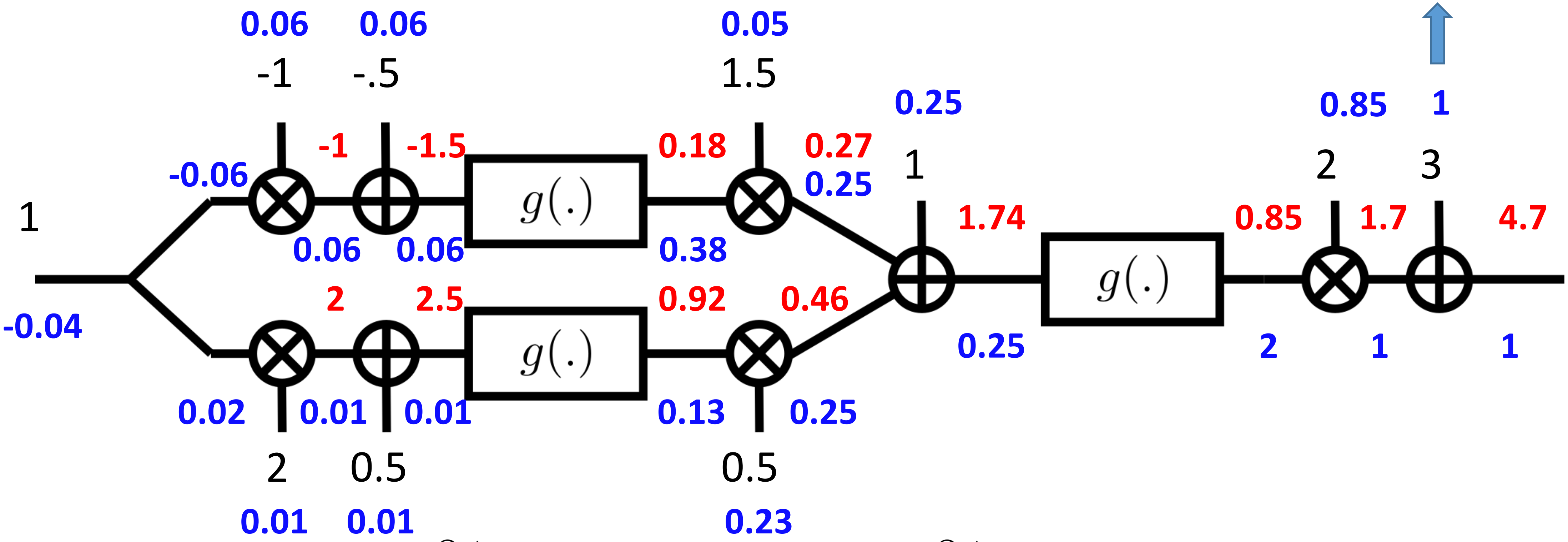
Backpropagation – Backward Pass



Derivative of a sum = sum of derivatives

Backpropagation – Backward Pass

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

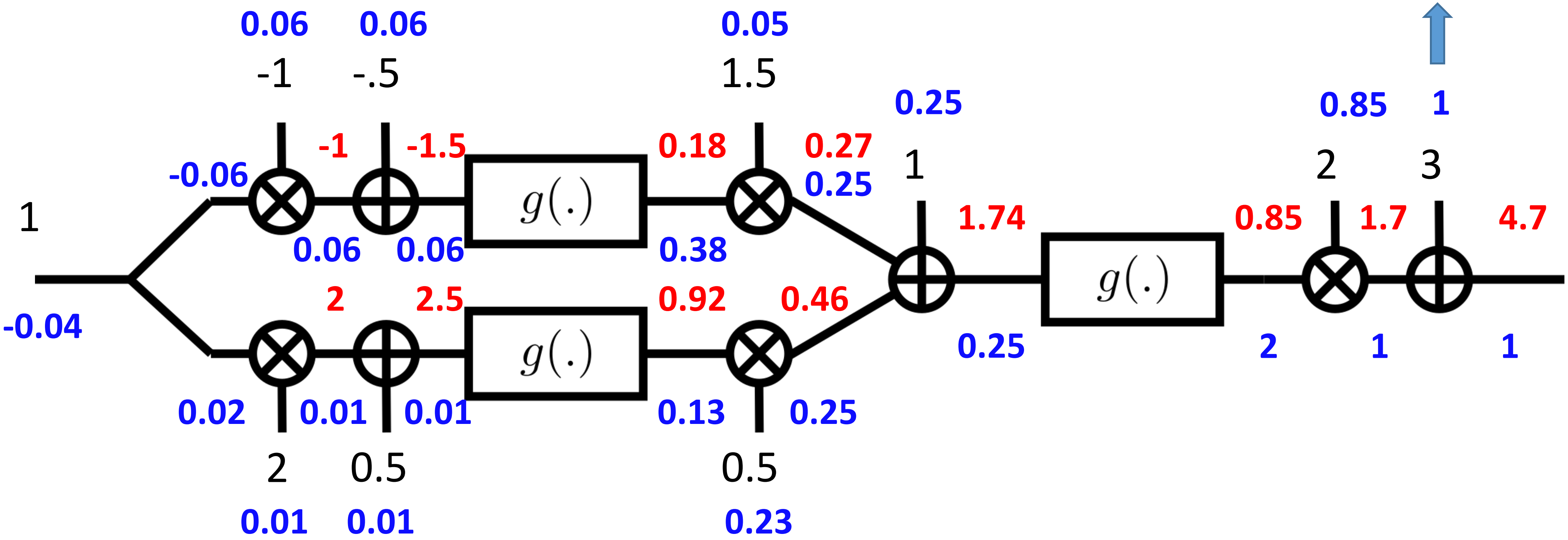


$$\frac{\partial \hat{y}_k}{\partial w_2} = 0.01$$

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

Backpropagation – Backward Pass

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



$$\left.\frac{\partial V_N(\theta)}{\partial \theta}\right|_{\theta_i} = -\frac{2}{N}\sum_{k=1}^{N}(y_k - \hat{y}_k)\left.\frac{\partial \hat{y}_k}{\partial \theta}\right|_{\theta_i}$$

Do this for all time-instances k

The cost equation should also be included in the backpropagation

Backpropagation

Automatic differentiation

Small sub-functions & Applying chain rule

Forward and backward pass

Computational efficiency $\mathcal{O}\left(\underbrace{n_e}_{\text{\# graph edges}}\underbrace{n}_{\text{\# function outputs}}\right) \Longleftrightarrow \mathcal{O}\left(\underbrace{n_e}_{\text{\# graph edges}}\underbrace{m}_{\text{\# function free variables}}\right)$ forward-mode derivation

Requires much storage

Learning Outcomes

What is an artificial neural network?

Why are artificial neural networks interesting for function approximation?

How to train a neural network? / What is backpropagation?

Artificial Neural Networks – After the Break

Training, Validation and Test (or Training a Neural Network Cont'd)

Simple Neural Networks for Modelling Dynamical Systems

Artificial Neural Networks and Gaussian Processes