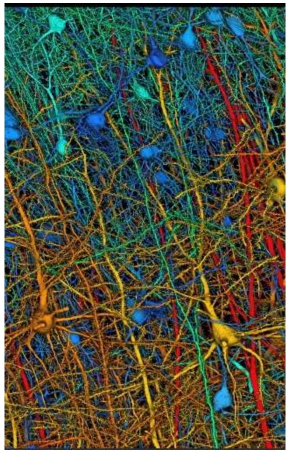
NEURAL NETWORKS



Prof. Dr. Mehmet Önder Efe
Department of Computer Engineering
Hacettepe University, Ankara, Turkey
http://web.cs.hacettepe.edu.tr/~onderefe/
e-mail : onderefe@gmail.com

My University

Daily Plan for You

- **To be announced here**

Webpages

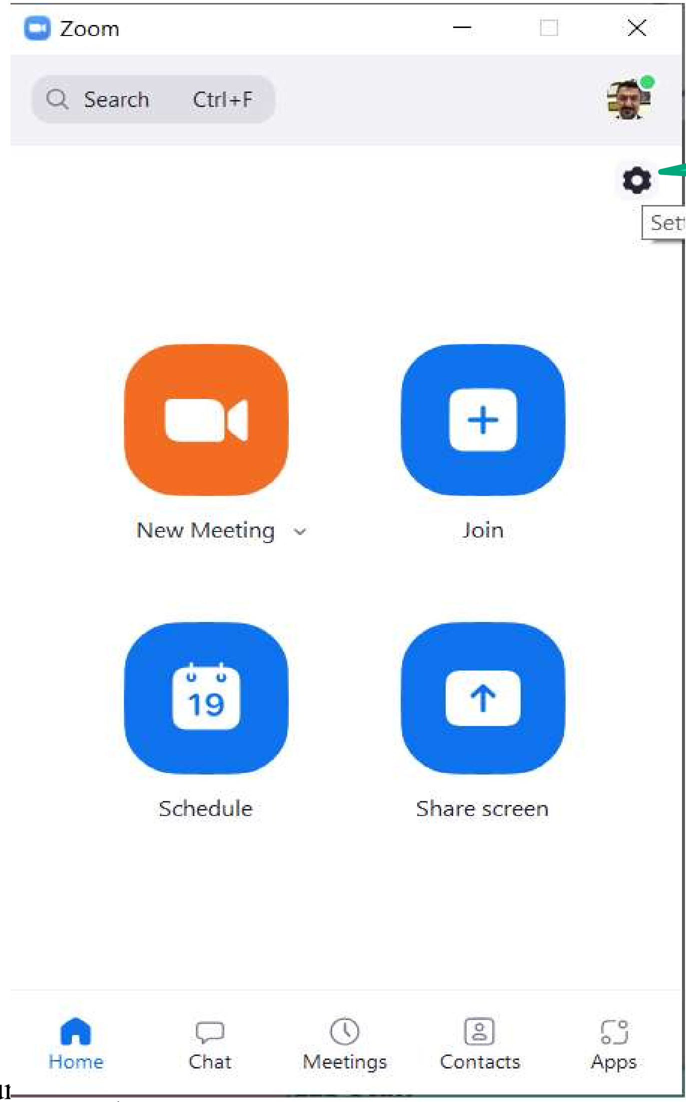
- My detailed webpage
- https://web.cs.hacettepe.edu.tr/~onderefe
- onderefe@gmail.com

- Course webpage
- https://web.cs.hacettepe.edu.tr/~onderefe/cmp684
- I will update this page continuously

Grading

- Paper reading 40%
- Paper writing 60%

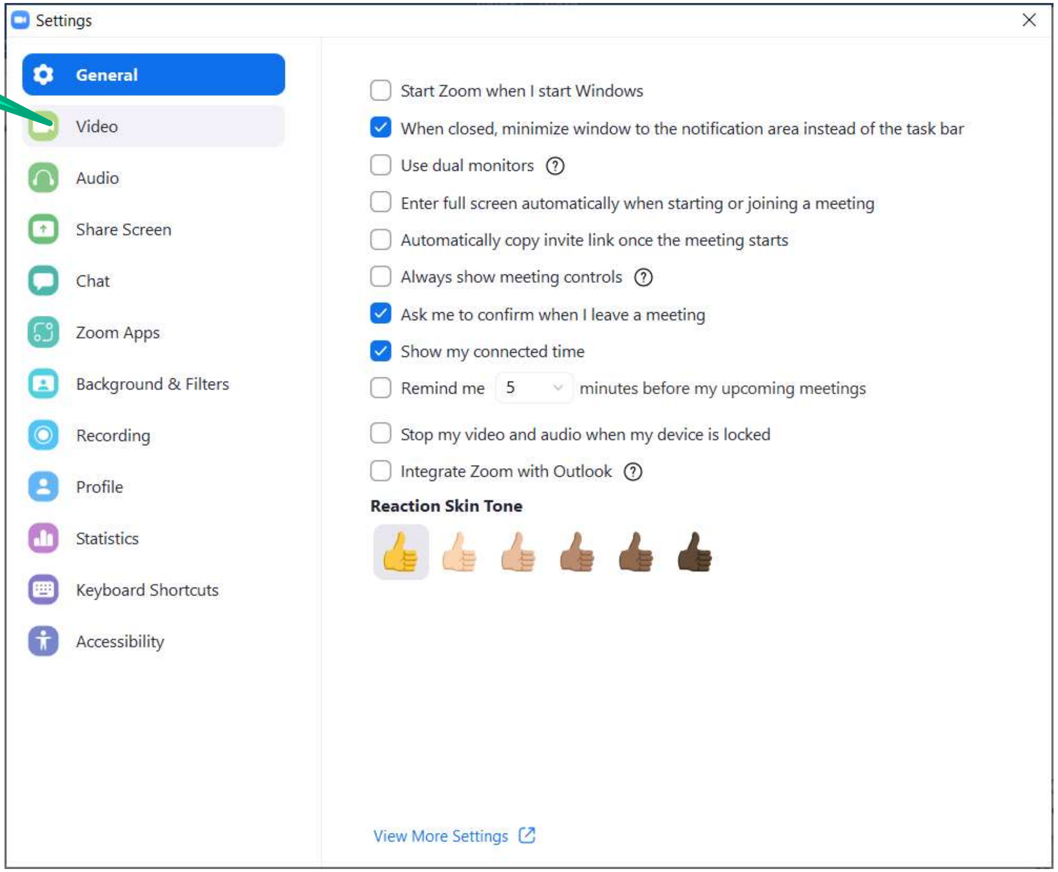
Entering a Zoom session with camera off



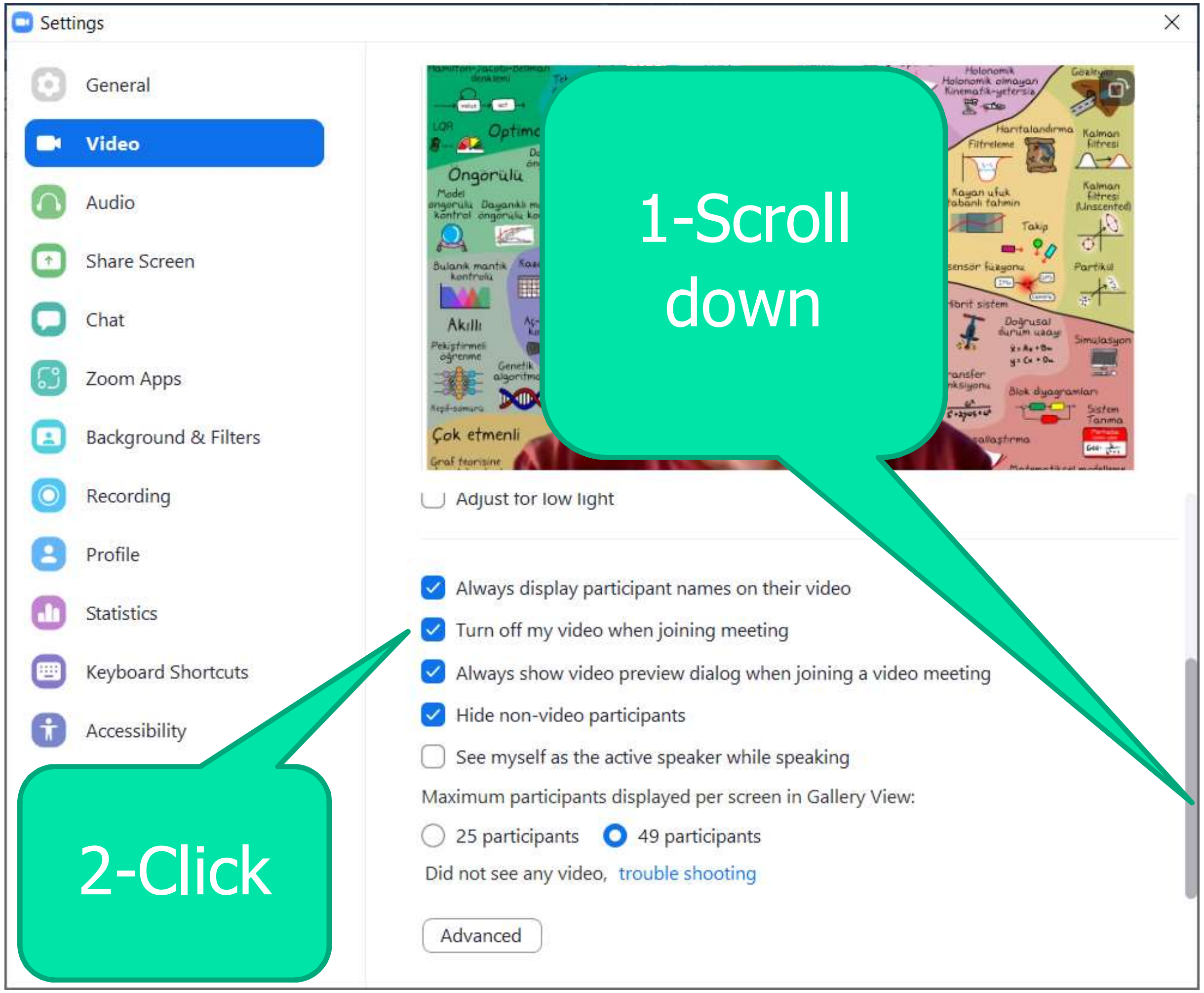
Click Settings

Entering a Zoom session with camera off

Click Video



Entering a Zoom session with camera off



- Let’s see the technical outline of the course

Course Outline

- A Historical Perspective
- Neuron and its Analytic Model
  - Inner product as a similarity measure (net sum)
  - Activation functions
  - Differentiability
  - Parameterization and computational aspects
  - Concept of learning (Tuning, Adaptation or Parameter Adjustment)
- Hopfield Neural Network

Course Outline

- Perceptron Learning Algorithms
- Multilayer Perceptron and Error Backpropagation
  - Derivation of the Learning Algorithm
  - Problems of Error Backpropagation
  - Memorization (Overfitting) and Generalization
  - Range of Variables (Normalization)
- Radial Basis Function Neural Networks
- Dynamic Neural Networks

Course Outline

- Second Order Training Schemes
    - Levenberg-Marquardt Algorithm
    - Gauss-Newton Algorithm
- Recurrent Neural Network Structures
- Several Applications of Neural Networks
    - Identification of Nonlinear Systems
    - Neurocontrol Structures
    - Noise Elimination
    - Adaptive Noise Cancellation
    - VLSI Implementation of NNs
    - NNs in Medical Diagnosis
    - NNs for Financial Applications

Course Outline

- An Open Question - Stability in Learning Systems
- Reinforcement Learning
- Unsupervised Learning

A Historical Perspective

McCulloch and Pitts (1943)

  A neuron model

Hebb (1949)

  A book: The Organization of Behavior

  First mentioning of *Synaptic Modification*

Uttley (1956)

  Classifiction of simple sets (binary patterns)

Rosenblatt (1958)

  Perceptron

Widrow and Hopf (1960)

  Least Mean Squares (LMS) for ADALINE
  (Adaptive Linear Element)

A Historical Perspective

Minsky (1961)

Credit Assignment Problem (Hidden layer issues)

Hopfield (1982)

Hopfield Networks

Rumelhart, Hinton, and Williams (1984)

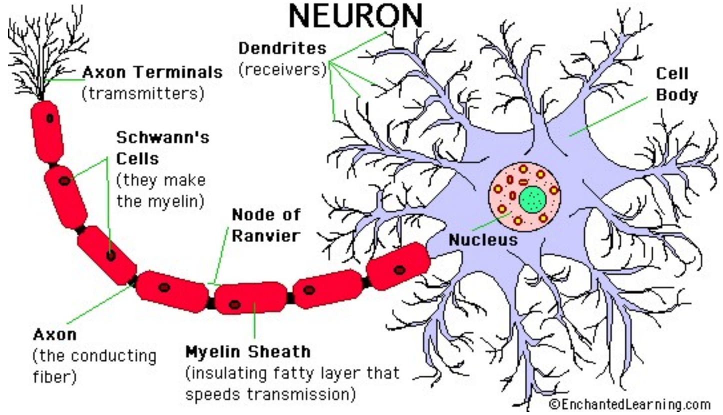
Backpropagation

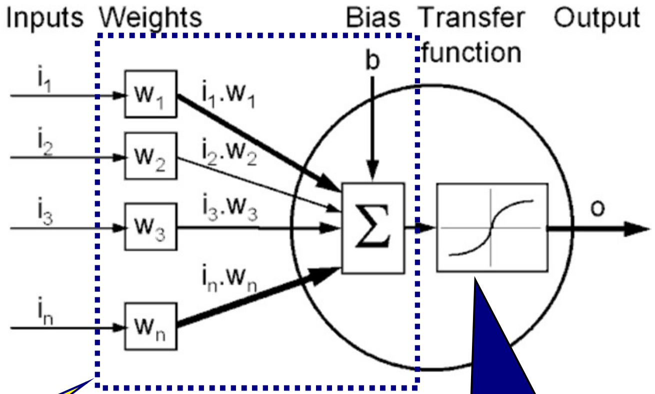
Broomhead and Lowe (1988)

Radial Basis Function Neural Networks

Deep Learning Era

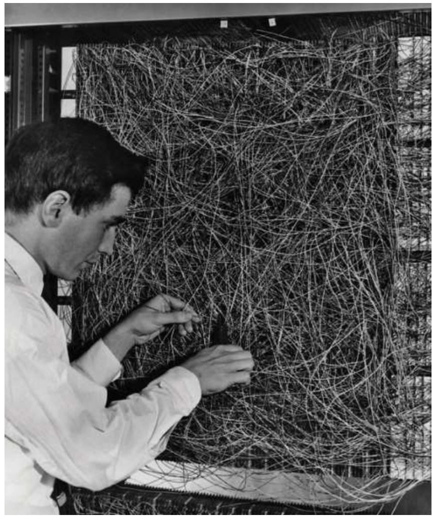
Neuron and its Analytic Model



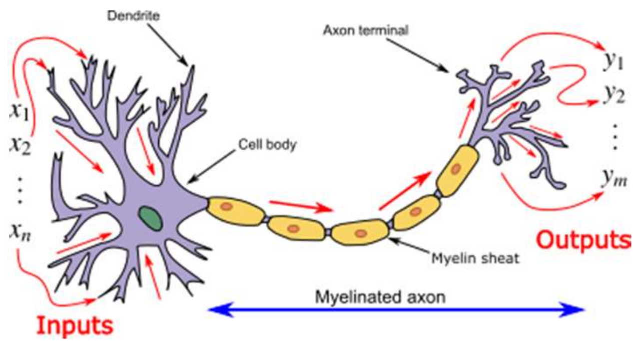


Inner Product

Activation Function



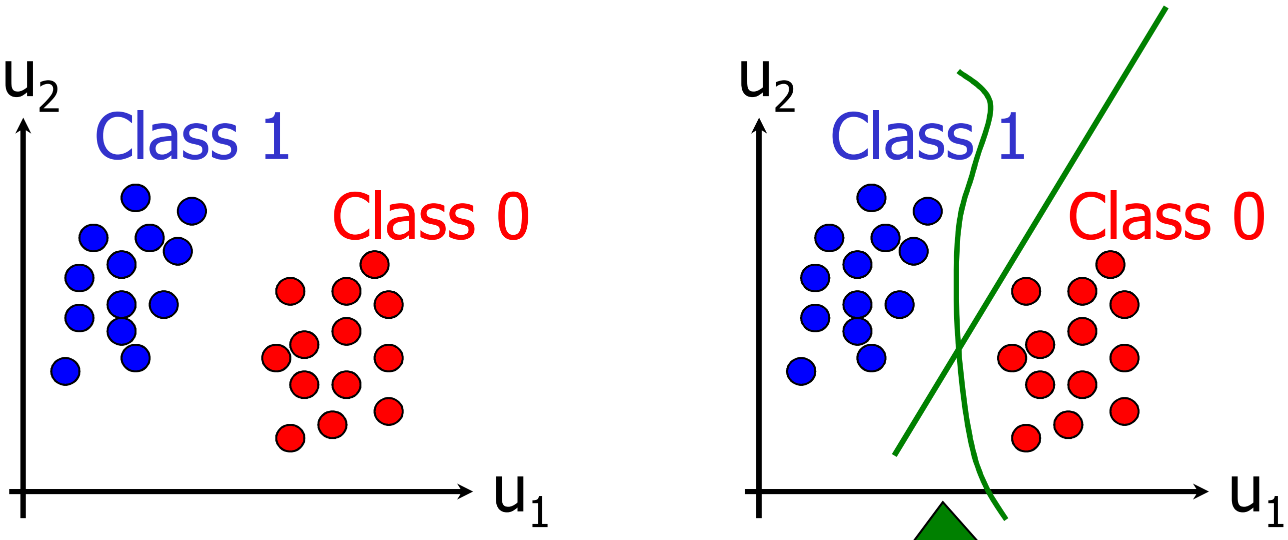
Mark-1 Perceptron



Mehmet Önder Efe, *Neural Networks*, Lecture Notes, 2022.

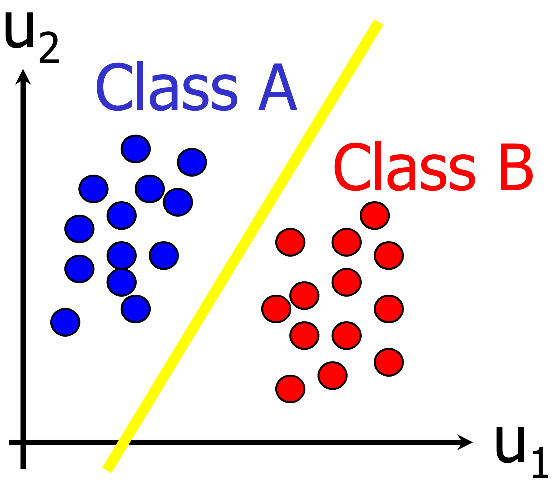
Neuron and its Analytic Model
Learning (Tuning, Adaptation, Adjustment)

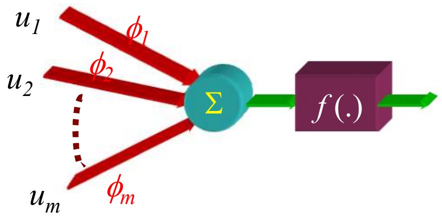
Assume you are given this data. How would you separate the two classes?



There are many decision boundaries!

Perceptron Learning Algorithm
Learning (Tuning, Adaptation, Adjustment)

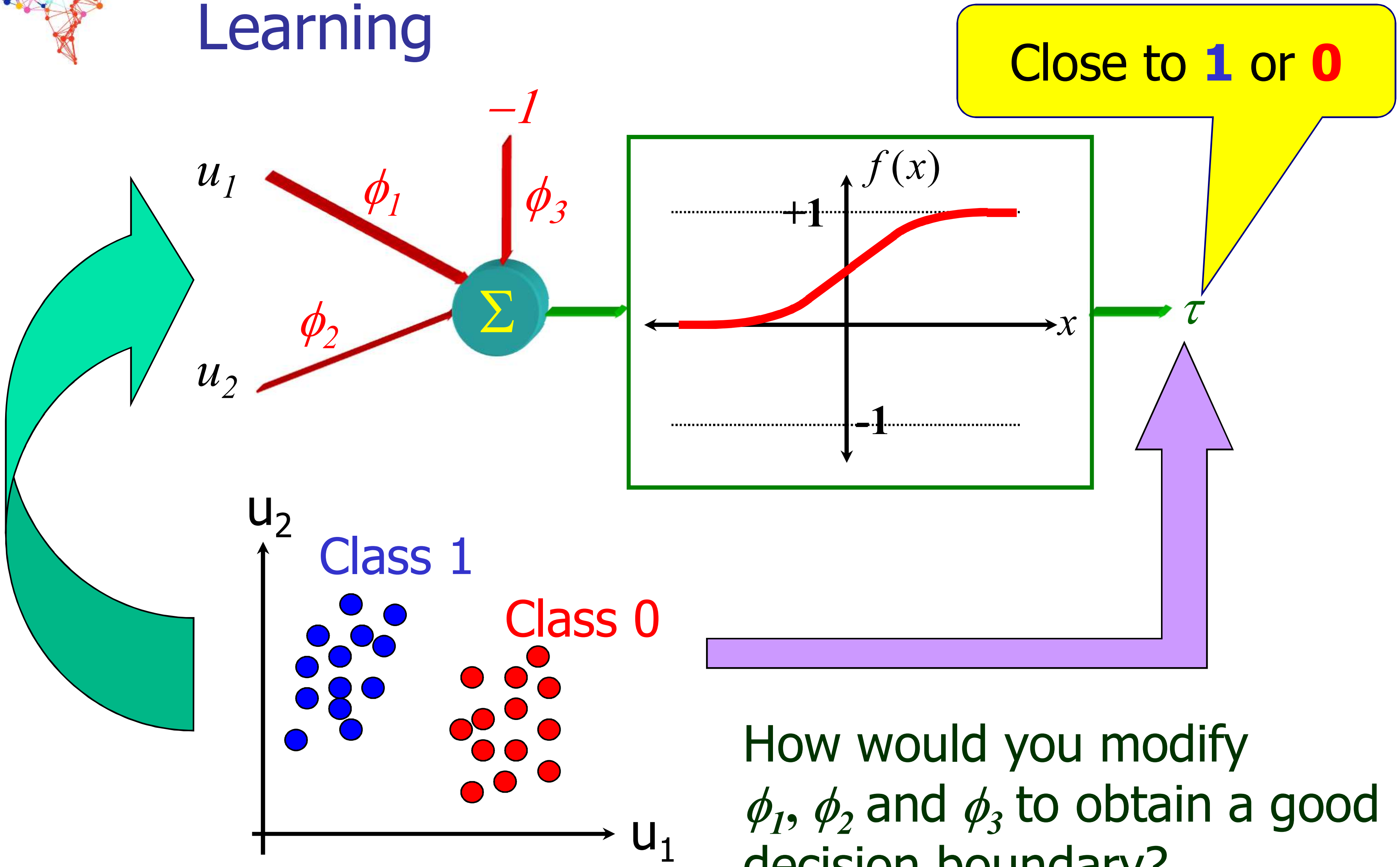




$$f(x)=\begin{cases}1 & Class\ \mathrm{A}\\ -1 & Class\ \mathrm{B}\end{cases}$$

- Find a decision boundary by modifying the adjustable parameters

Neuron and its Analytic Model Learning



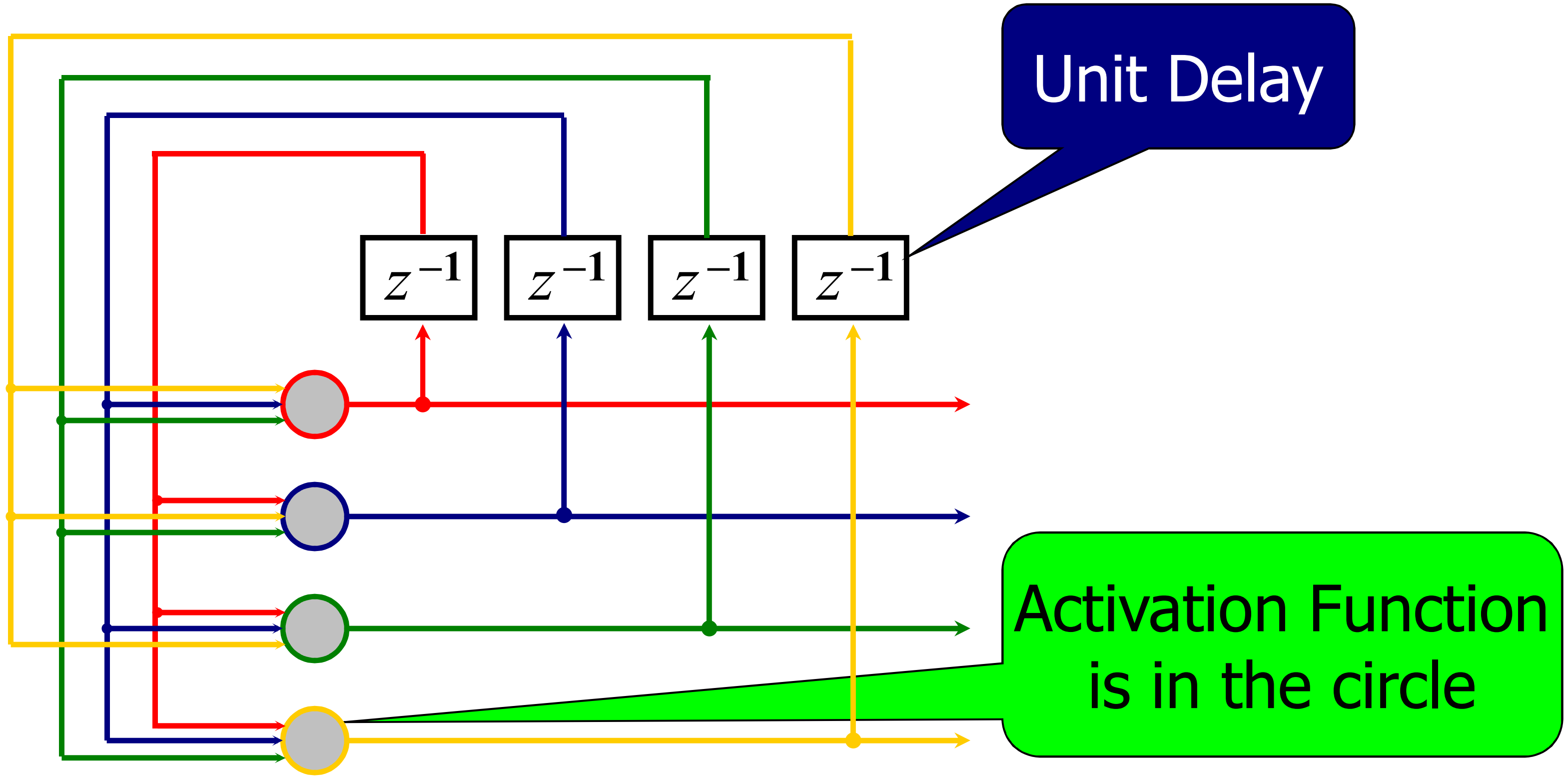
How would you modify ϕ_1, ϕ_2 and ϕ_3 to obtain a good decision boundary?

Performance of a Classifier

		Predicted Class		
		Positive	**Negative**	
Actual Class	**Positive**	True Positive (TP)	False Negative (FN) **Type II Error**	**(Recall)** **Sensitivity** $\frac{TP}{(TP+FN)}$
	Negative	False Positive (FP) **Type I Error**	True Negative (TN)	**Specificity** $\frac{TN}{(TN+FP)}$
		Precision $\frac{TP}{(TP+FP)}$	**Negative Predictive Value** $\frac{TN}{(TN+FN)}$	**Accuracy** $\frac{TP+TN}{(TP+TN+FP+FN)}$

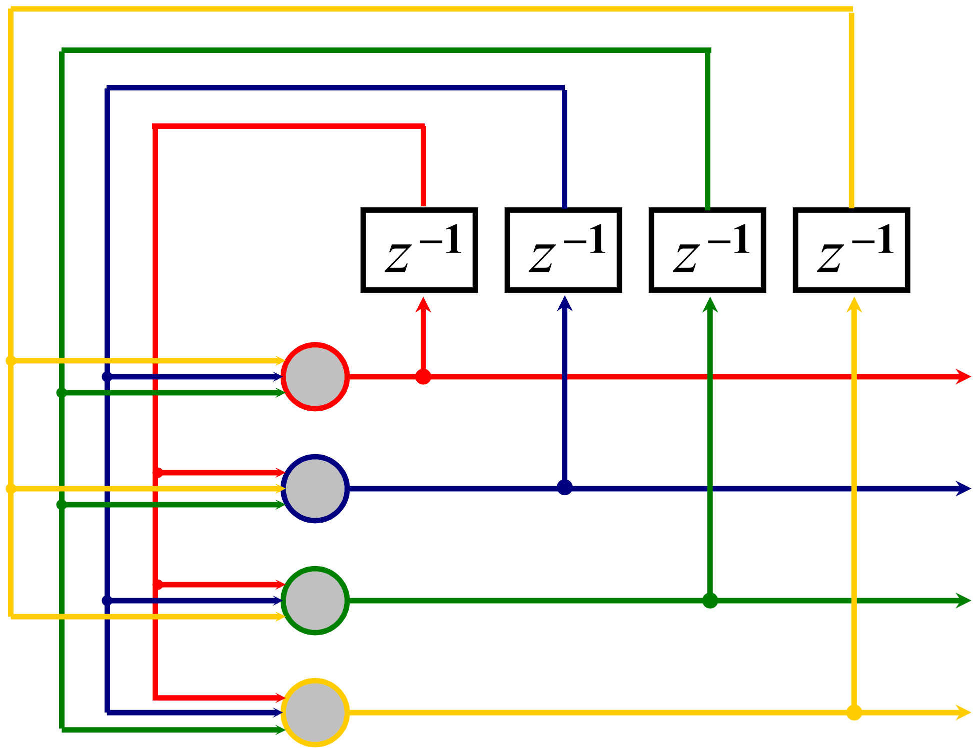
F1-Score = 2 x Precision x Recall /(Precision + Recall)

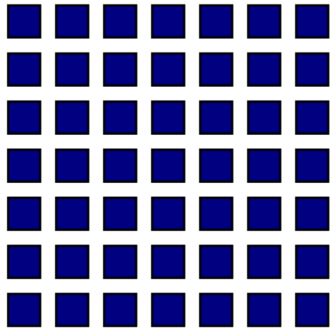
Hopfield Neural Network



- This is a 4 neuron Hopfield network, which is recurrent
- Output of a neuron is not fed back to itself

Hopfield Neural Network

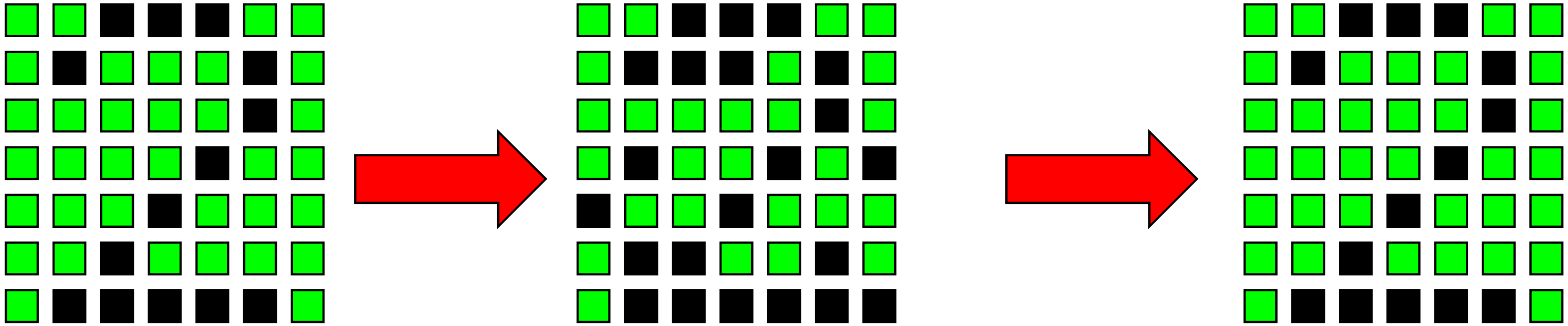




This is a canvas composed of neurons in the Hopfield Network

- Character recognition
- Content Addressable Memory

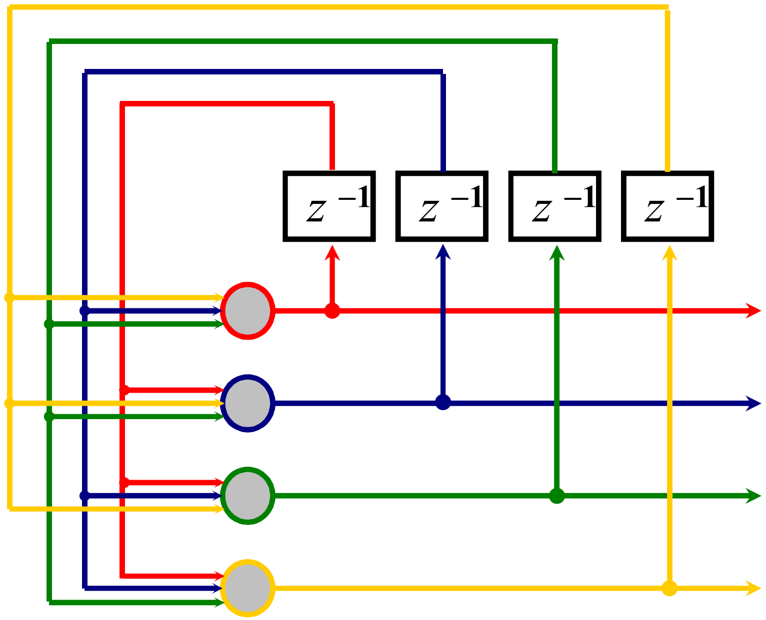
Hopfield Neural Network



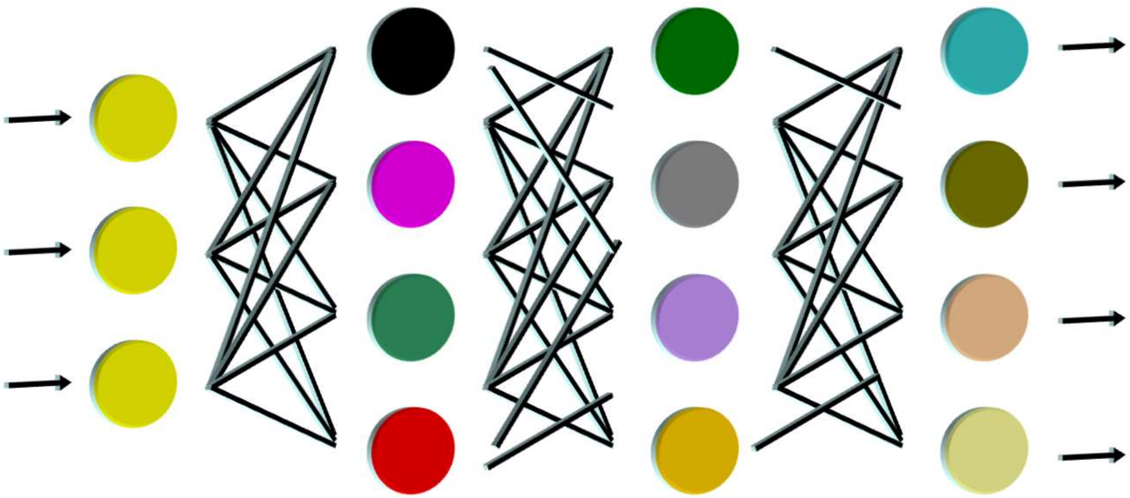
Train the net for this

Start with a noisy 2

Converges to memorized 2

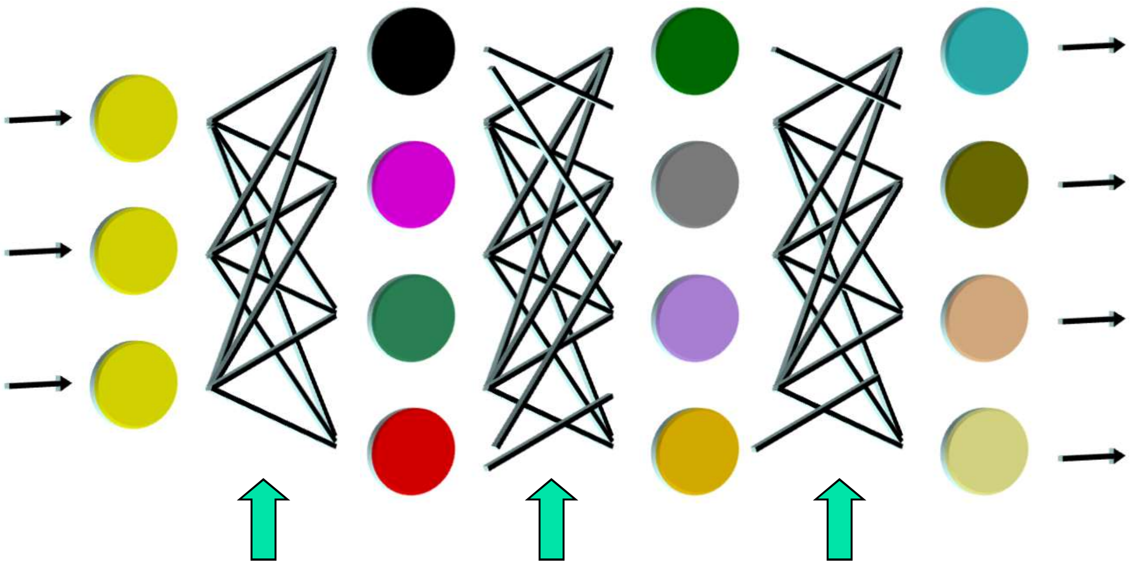


Multilayer Perceptron and Error Backpropagation (EBP)



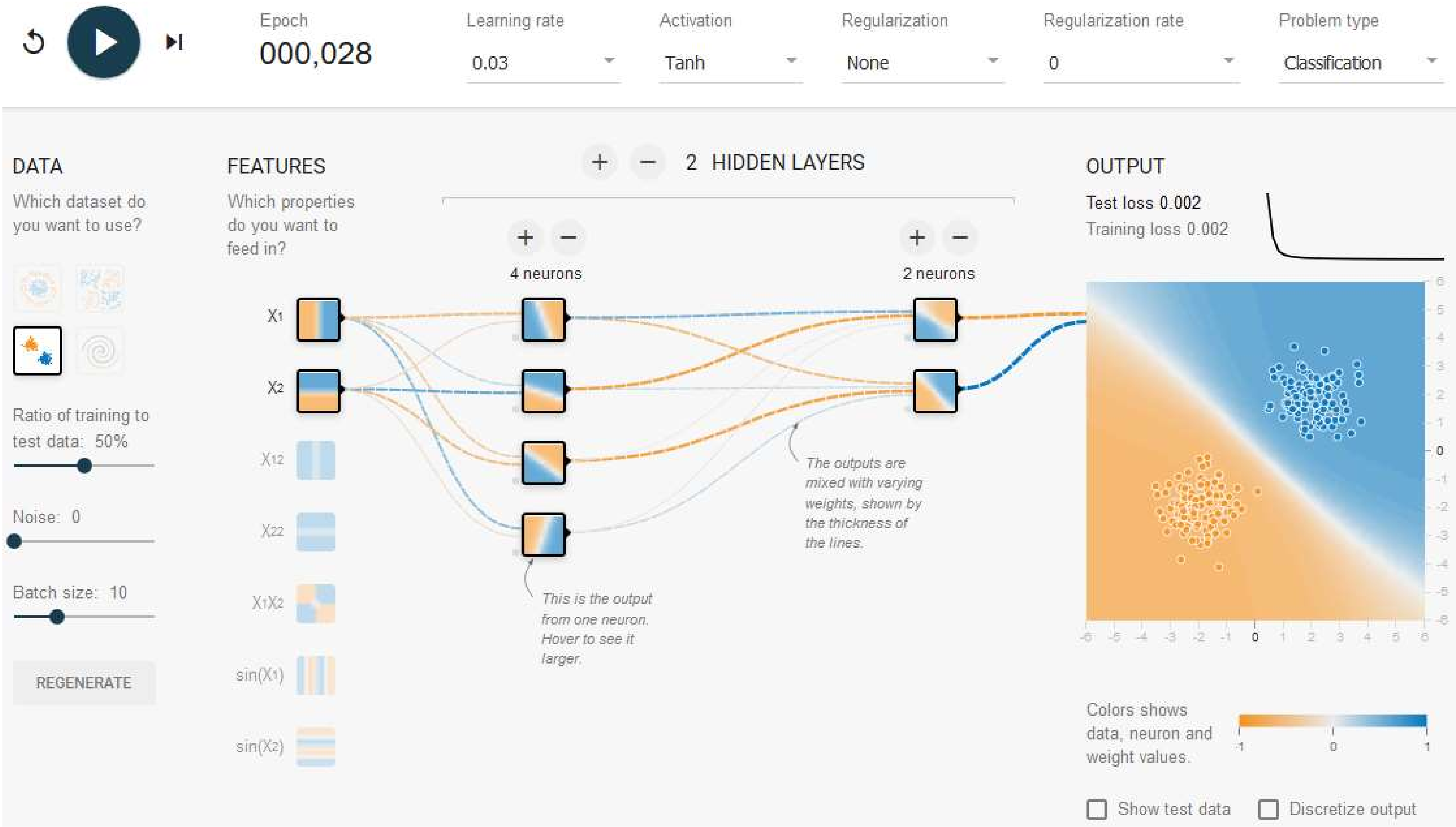
- Structure is layered, and a hierarchy is apparent in it
- Structure is composed of some sub-components, neurons
- A nonlinear map from input space to output space

Multilayer Perceptron and Error Backpropagation (EBP)



- What is adjustable?: The matrices (weights and biases) in between layers
- How is this done: EBP, CG, GN, LM etc.

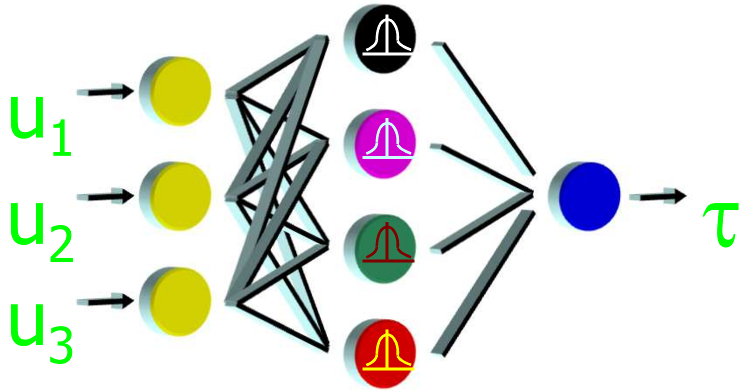
Multilayer Perceptron and Error Backpropagation (EBP)

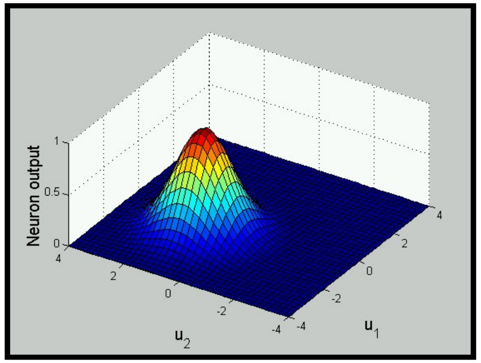


Mehmet Önder Efe, *Neural Networks*, Lecture Notes, 2022.

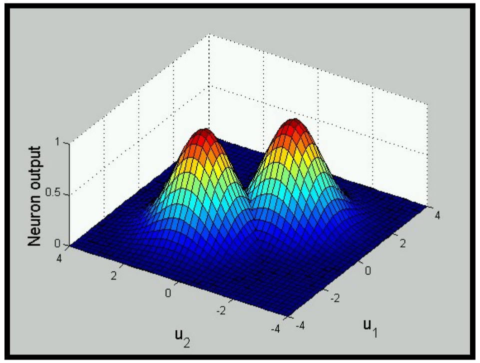
playground.tensorflow.org/

Radial Basis Function Neural Networks

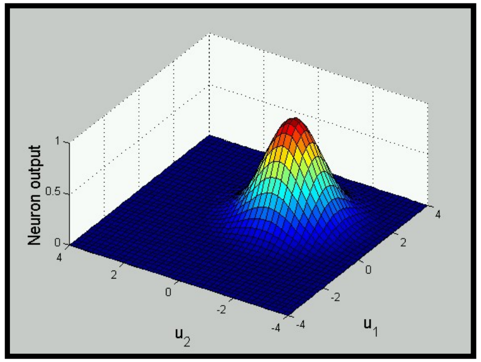




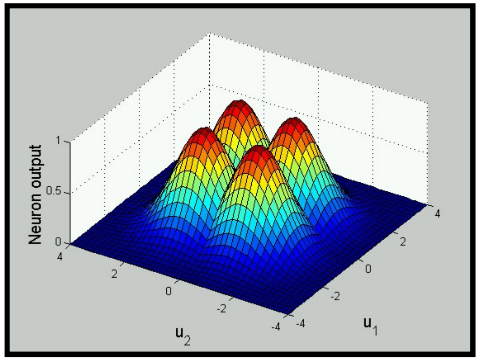
A neuron becomes active for the current input



As the input moves, another neuron starts responding

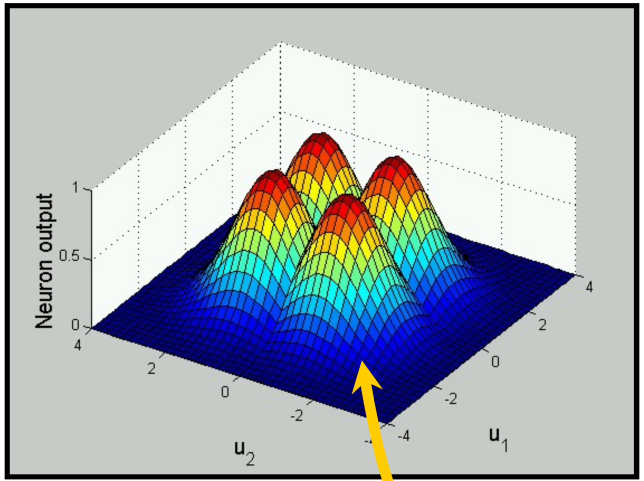


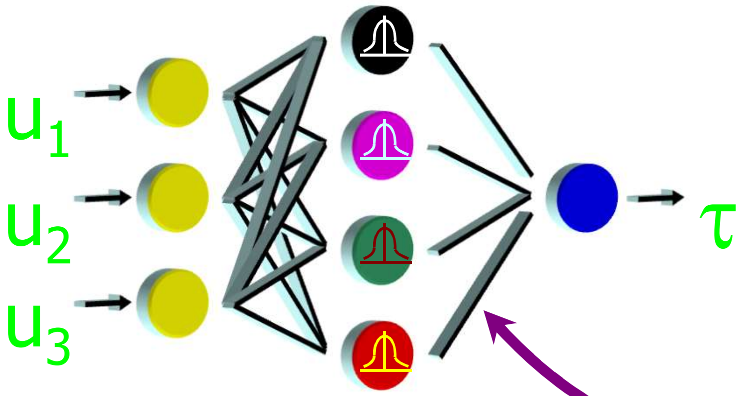
Then it becomes the dominantly excited neuron



A good coverage of the input space lets you know where you are during the course of your application

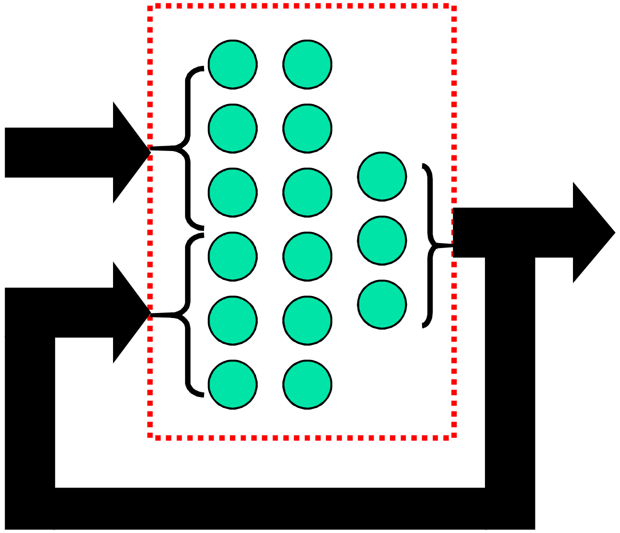
Radial Basis Function Neural Networks



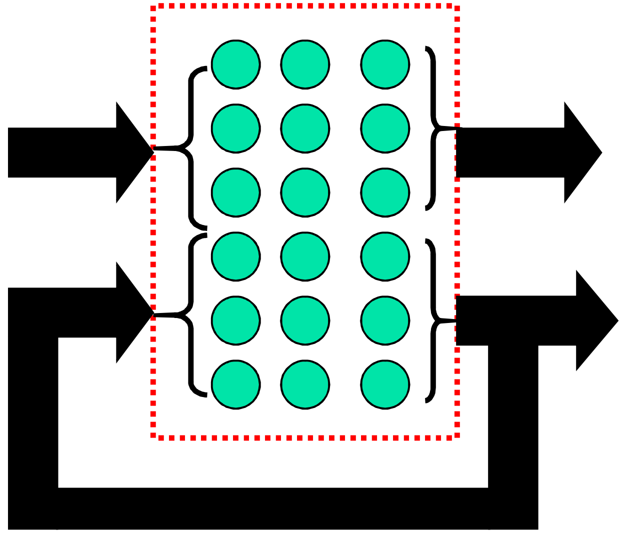


- What is adjustable?: **Centers and widths** of the basis functions, and the **output parameters**
- How is this done: EBP, CG, GN, LM etc.

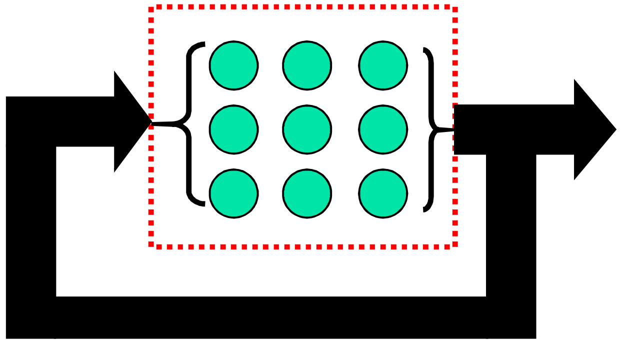
Recurrent Neural Network Structures



Real time recurrent net.



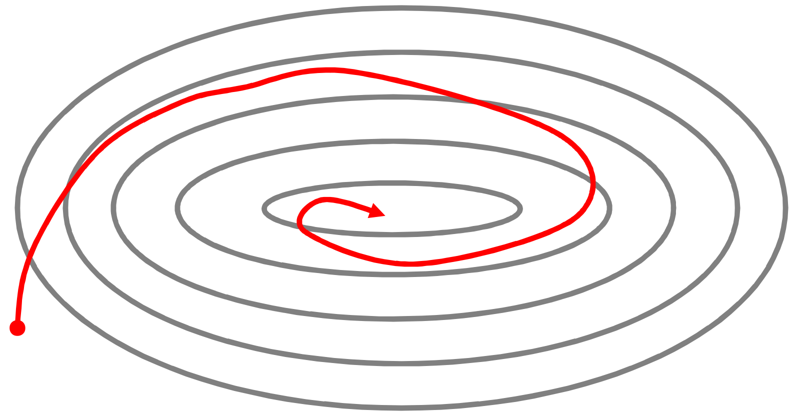
Partially recurrent net.



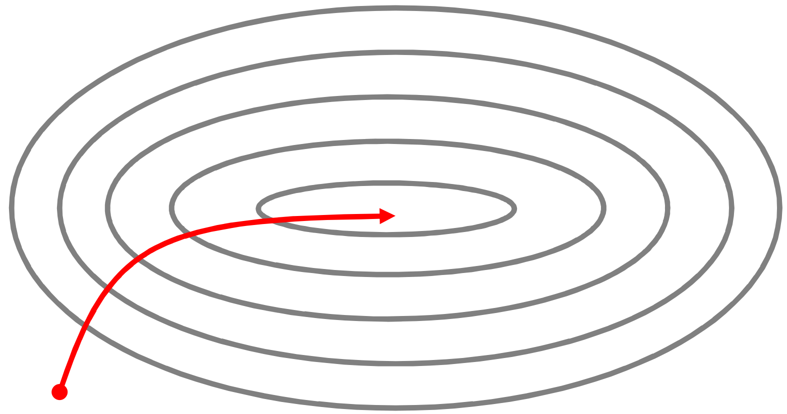
Hopfield net.

Second Order Training Schemes

Cost surface contours



Cost surface contours

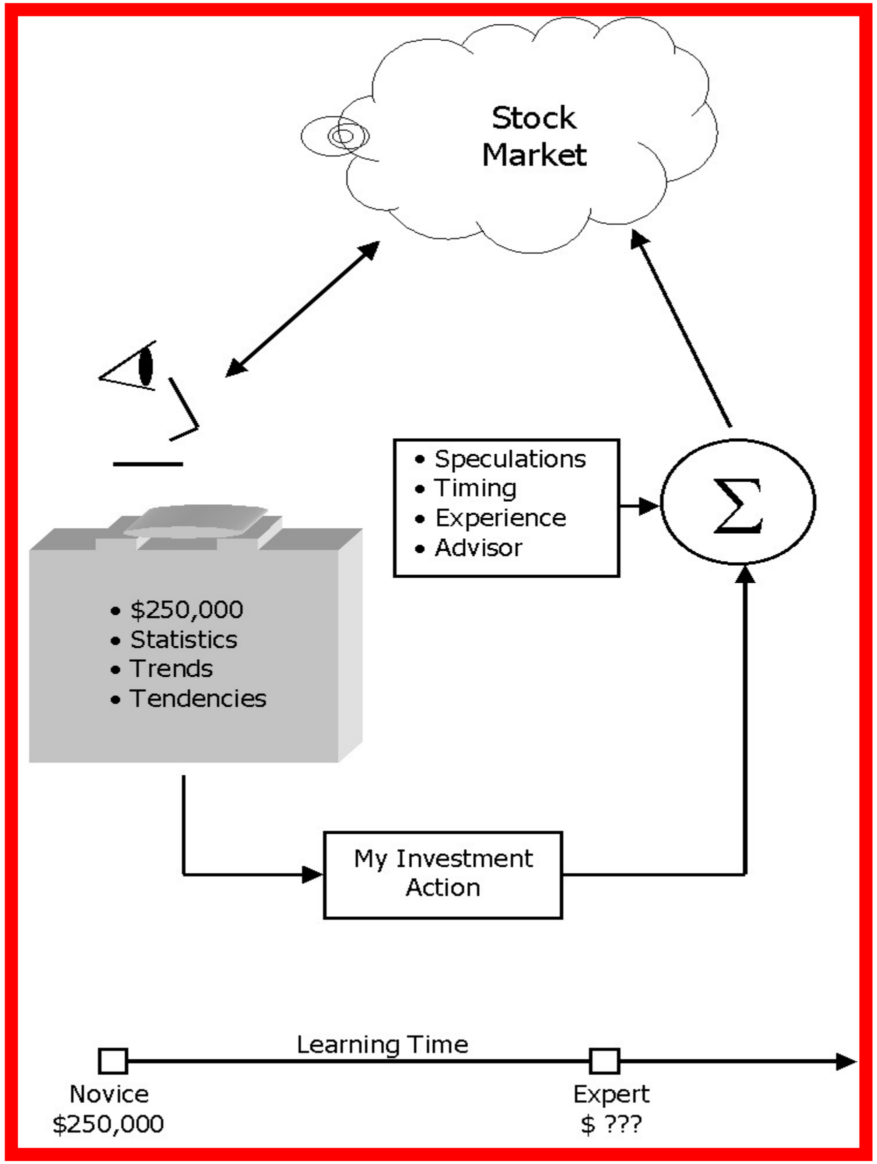


- Cost is decreasing in both of them. But in one of them it takes a long time to find the minimum.
- Levenberg-Marquardt (LM), Gauss-Newton (GN) algorithms are examples of 2nd order methods. EBP is a 1st order method

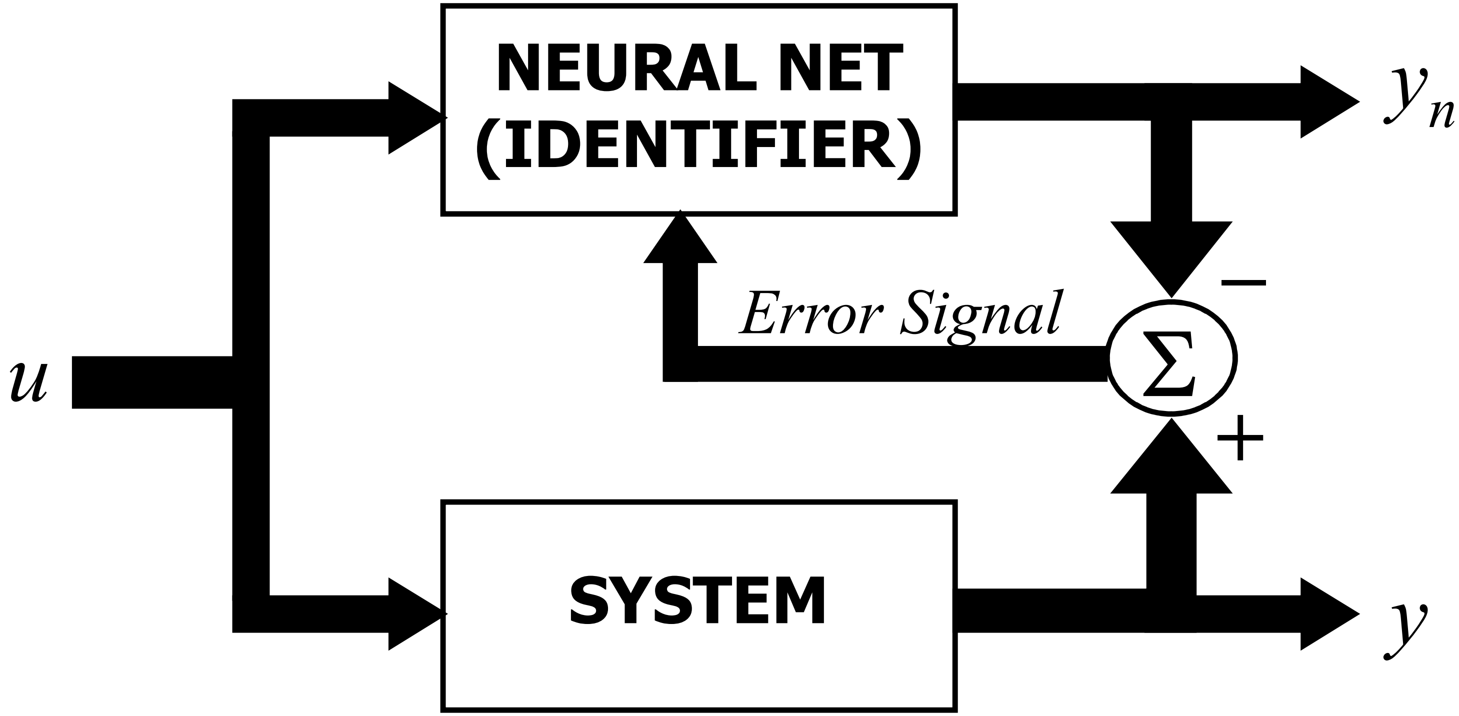
Applications of Neural Networks

An Example

Increase the profit by identifying the mechanism and appropriately making the decisions

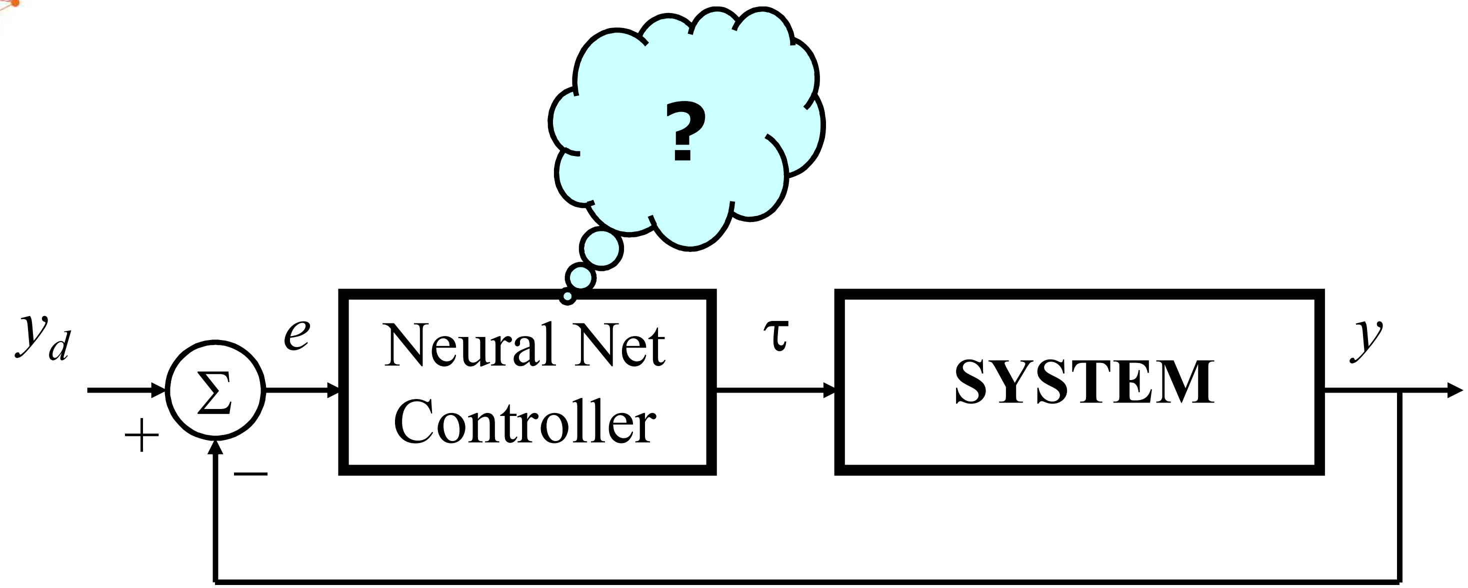


Applications of Neural Networks



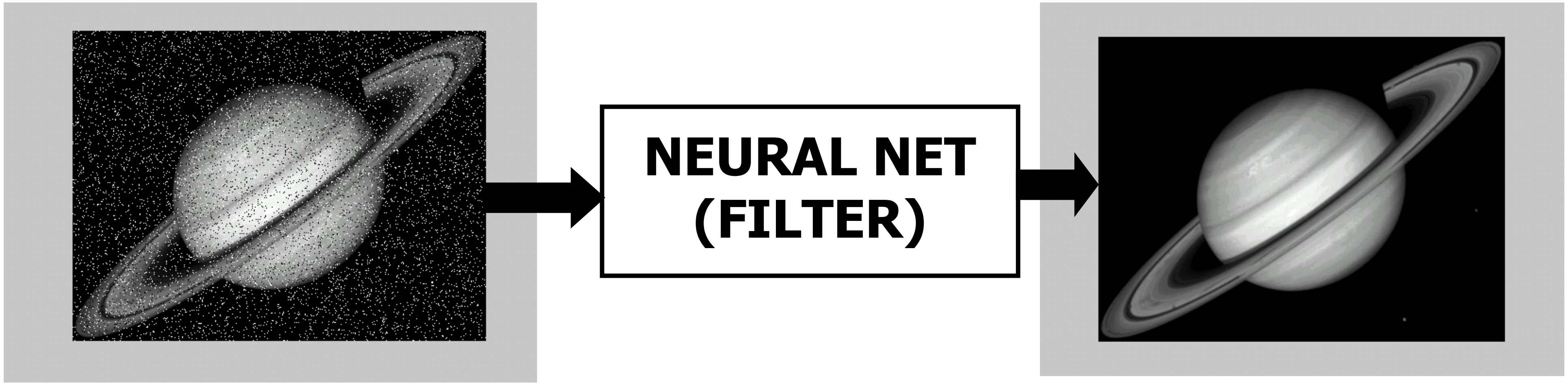
- System above may be a robot, a chemical process, an industrial process etc. We will see all these in detail...

Applications of Neural Networks



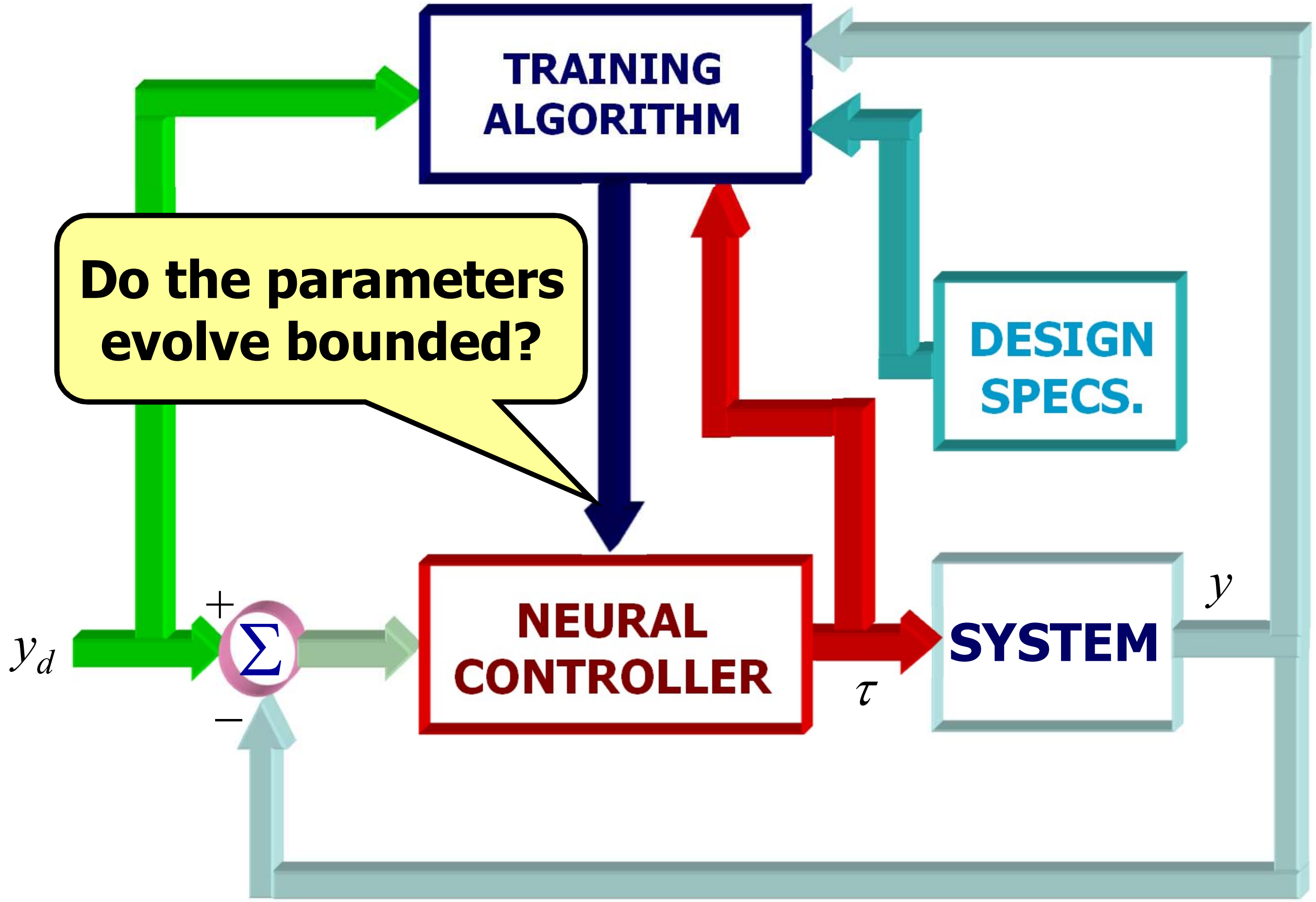
- How do we train such a neurocontroller?
- What alternatives are possible (online/offline tuning)
- What considerations are important (training robustness)
- Is this useful? Or when is neural control useful?

Applications of Neural Networks

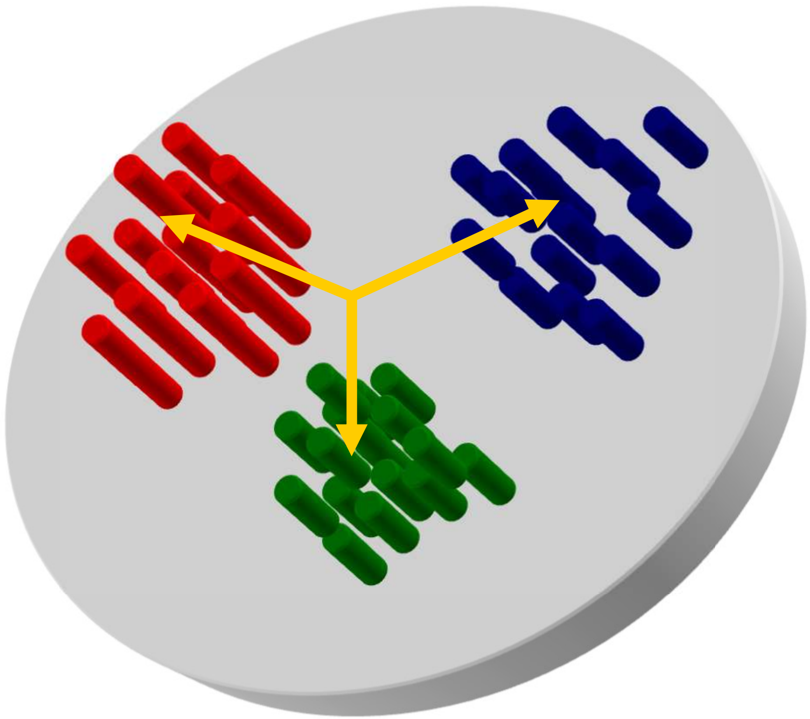


- How do we filter out the noise from the source?
- How do we teach *what to filter out* and *how to filter out* ?

Applications of Neural Networks

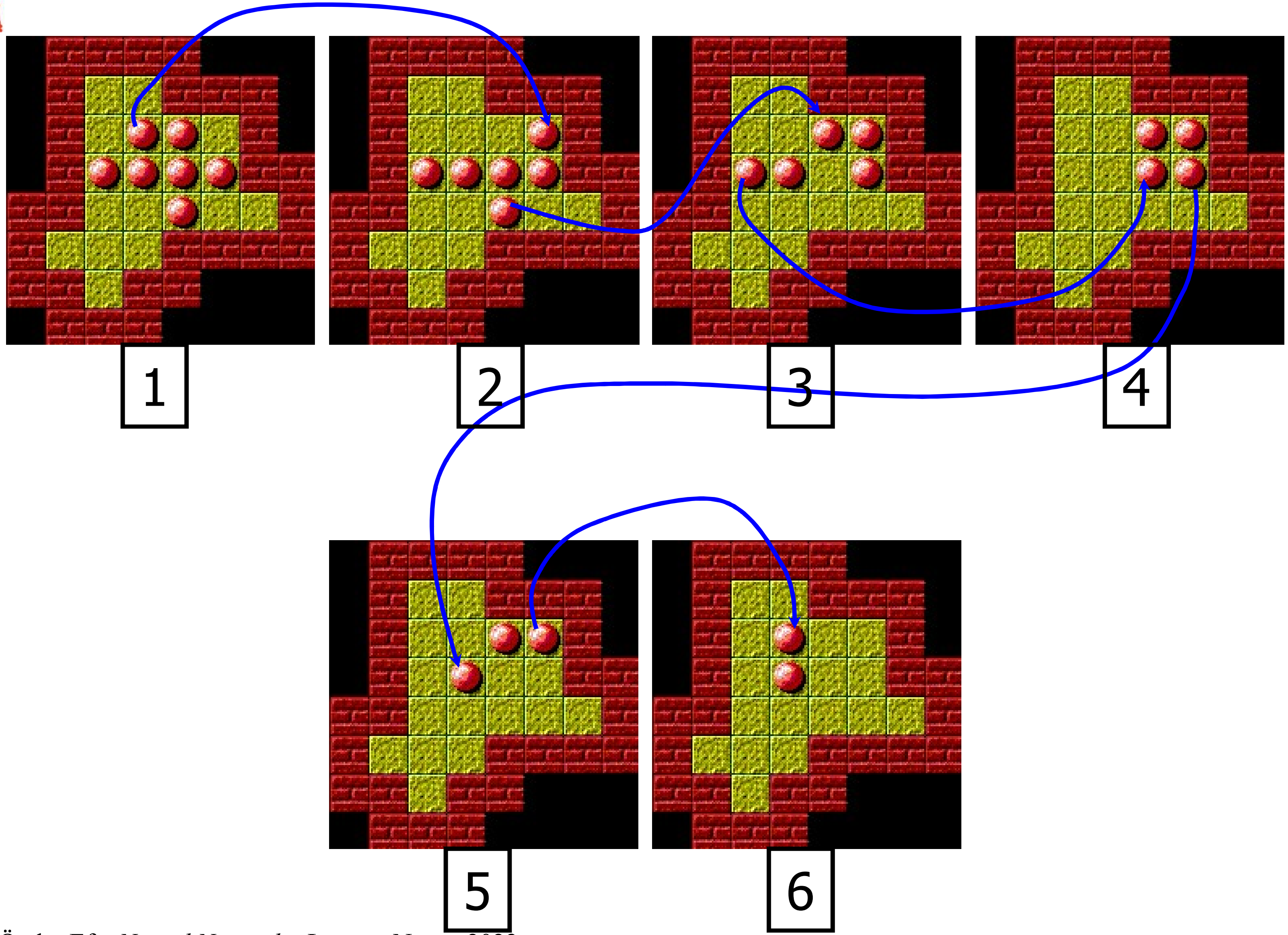


Applications of Neural Networks



Detect the persistent features of the input data without any feedback (teacher, supervisor) from the environment: Used for data clustering, feature extraction and similarity detection.

Applications of Neural Networks



Applications of Neural Networks

- How would you model this problem?
- How would you design a neural net playing the game?

Let's Start

- Neuron and its Analytic Model
  - Inner product as a similarity measure (net sum)
  - Activation functions
  - Differentiability
  - Parameterization and computational aspects
  - Concept of learning (Tuning, Adaptation or Parameter Adjustment)
- Hopfield Neural Network

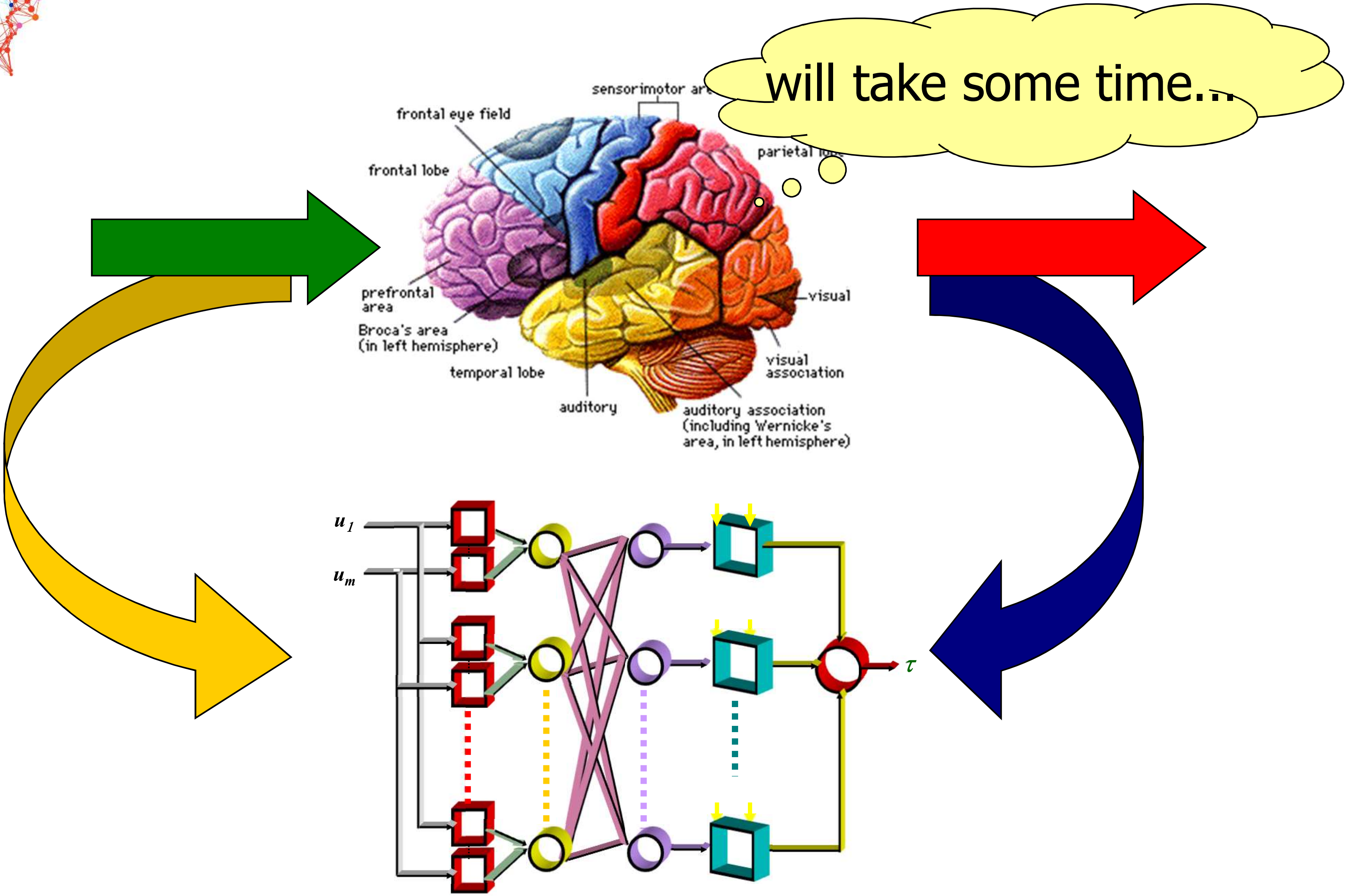
Motivation

Complexity requiring machine intelligence is everywhere…

- Industry workers
  Welding and assembly
- Unmanned Vehicles
  UAV, UGV, USV
- Medical applications
  Coronary surgery
- Military Applications
  Missile Control
- Space research
  Mars mission
- Entertainment
  Robot dog

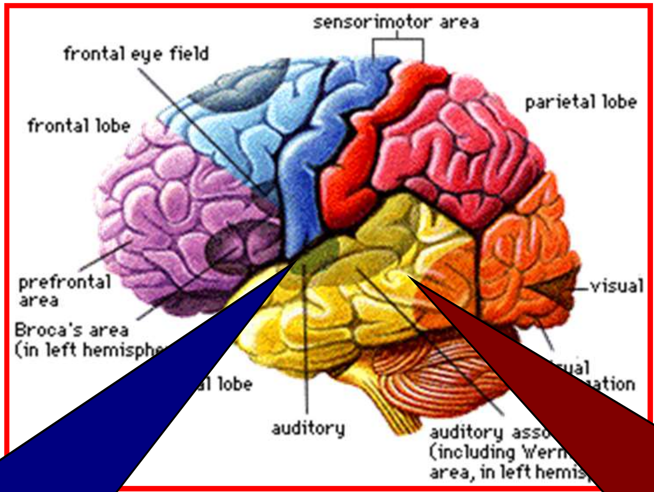
Design ‘*systems*’ operating without human intervention

Brain



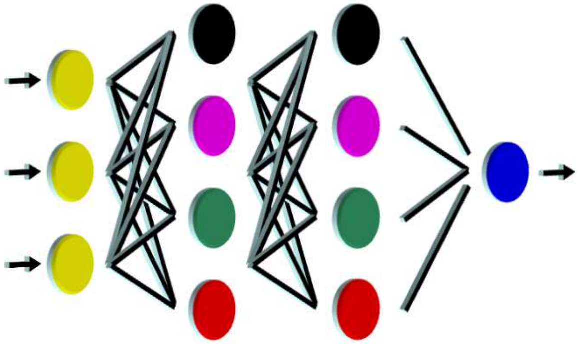
How to devise a model to imitate it?

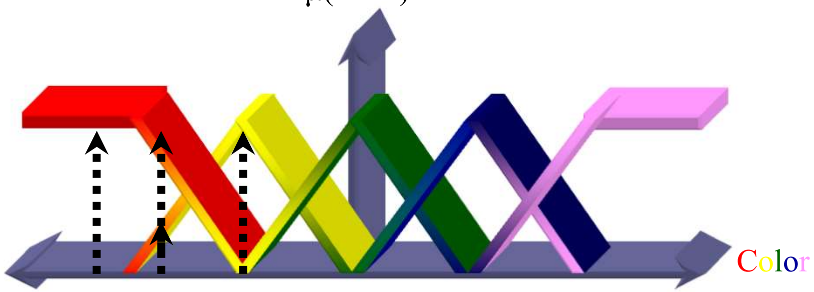
Imitate what?



Hardware
-Connectionist structures

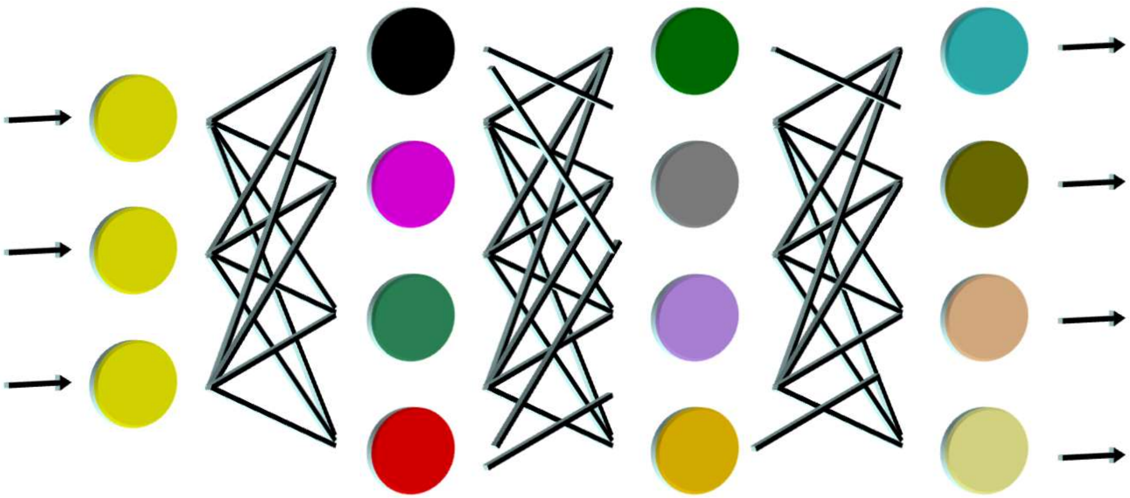
Software
-Rule based structures





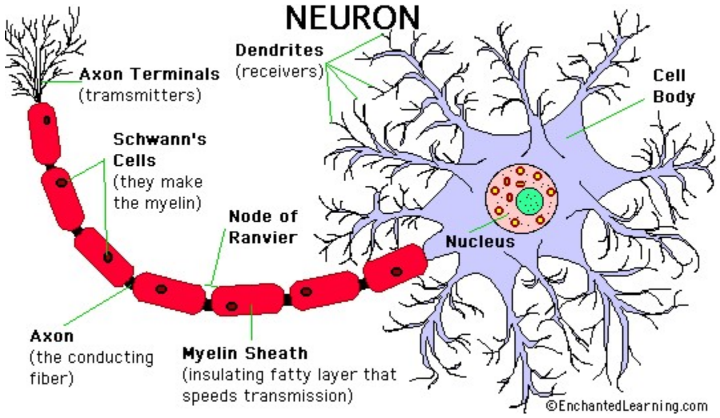
Mehmet Önder Efe, *Neural Networks*, Lecture Notes, 2022.

We will consider hardware of it

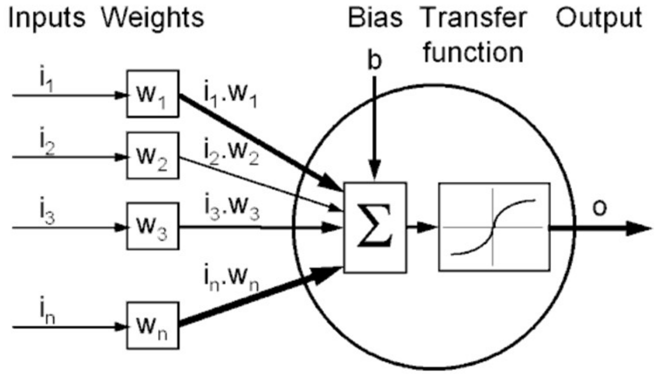


- Structure is layered, and a hierarchy is apparent in it
- Structure is composed of some sub-components, neurons

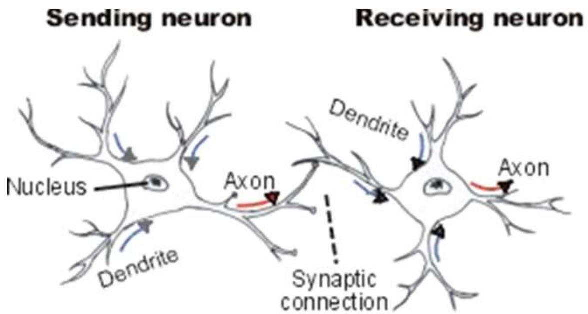
Neuron and Its Analytic Model



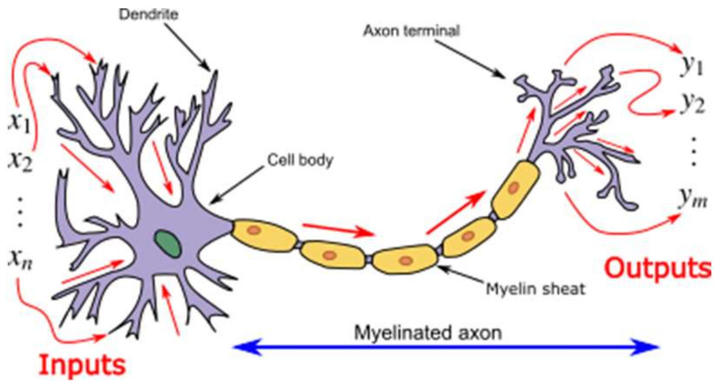
A Neuron



Math Model



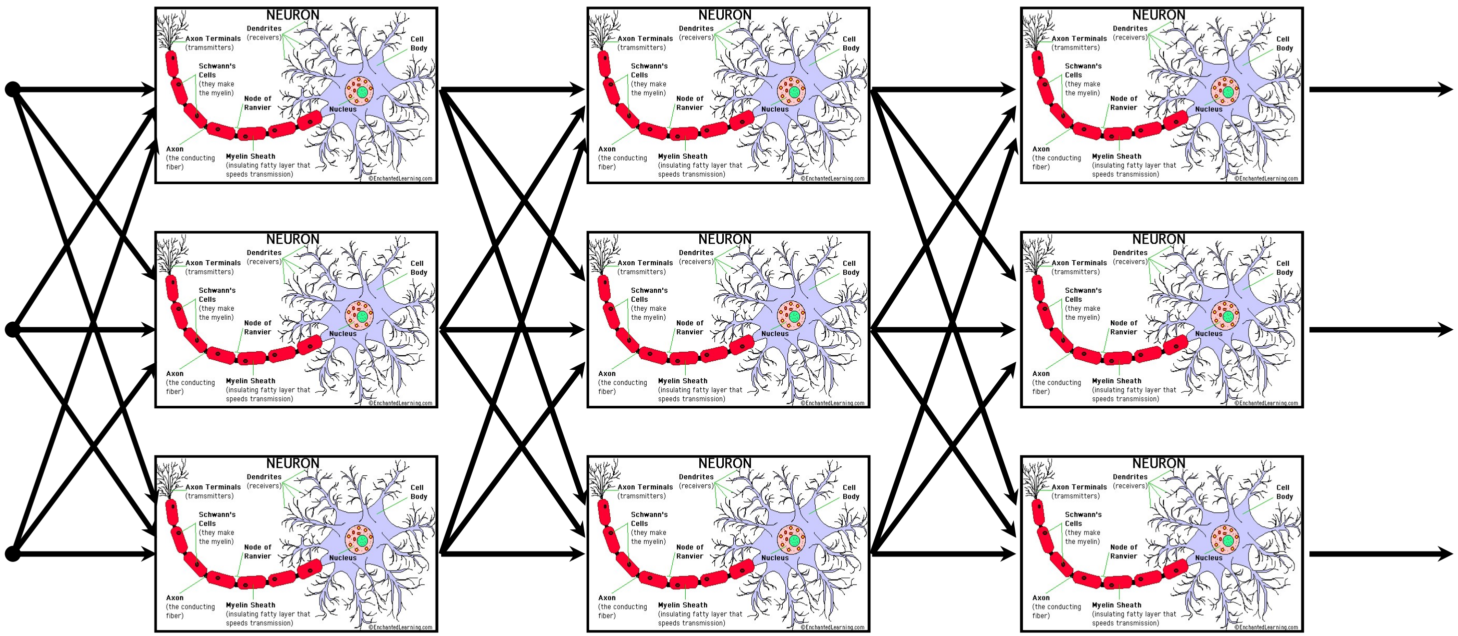
Two neurons in interaction

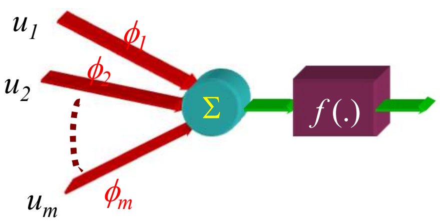


Mehmet Önder Efe, *Neural Networks*, Lecture Notes, 2022.

Neuron and Its Analytic Model

This is what we will get

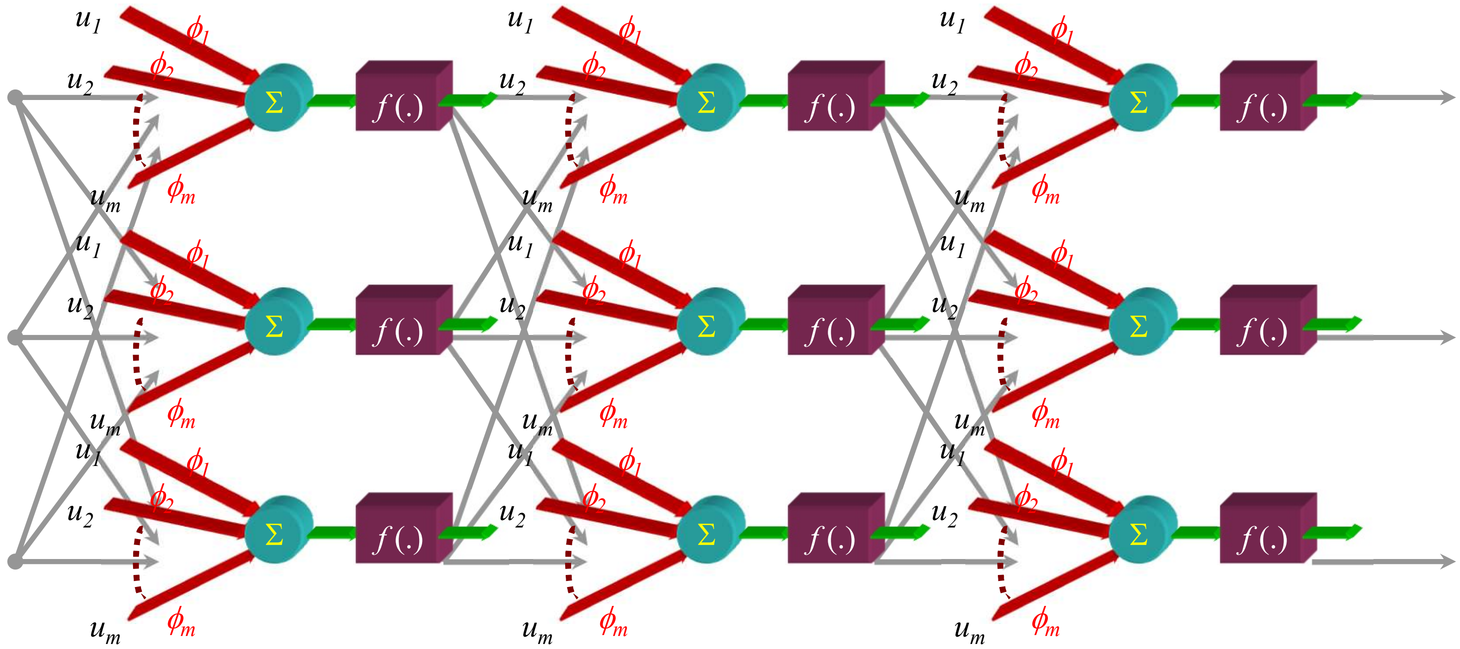




- Replace each neuron with its analytic counterpart

Neuron and Its Analytic Model

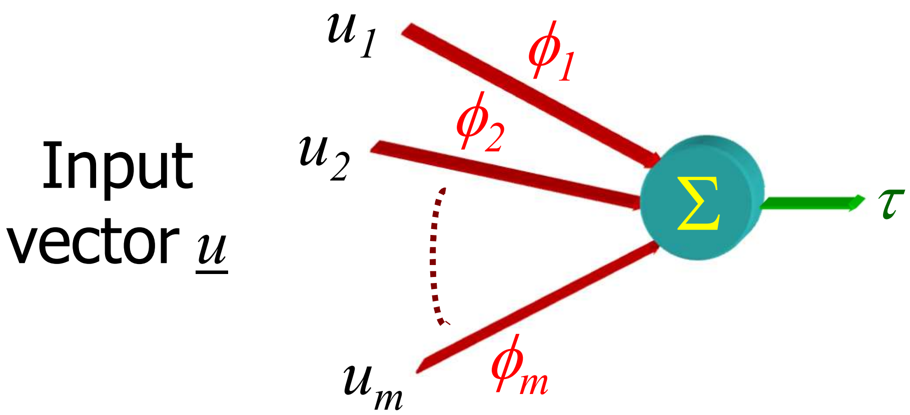
Now analyze this network



Mehmet Önder Efe, *Neural Networks*, Lecture Notes, 2022.

Neuron and Its Analytic Model

Adaptive Linear Element - ADALINE



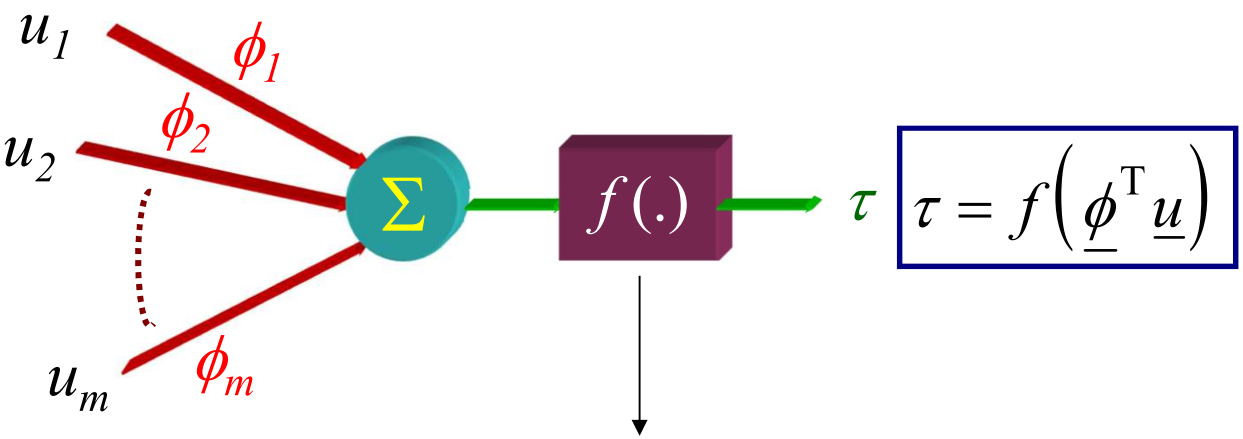
Output

$$\tau = \underline{\phi}^{\mathrm{T}} \underline{u} = \sum_{i=1}^{m} \phi_i u_i$$

Adjustable
Parameter vector $\underline{\phi}$

Neuron and Its Analytic Model

Activation Functions

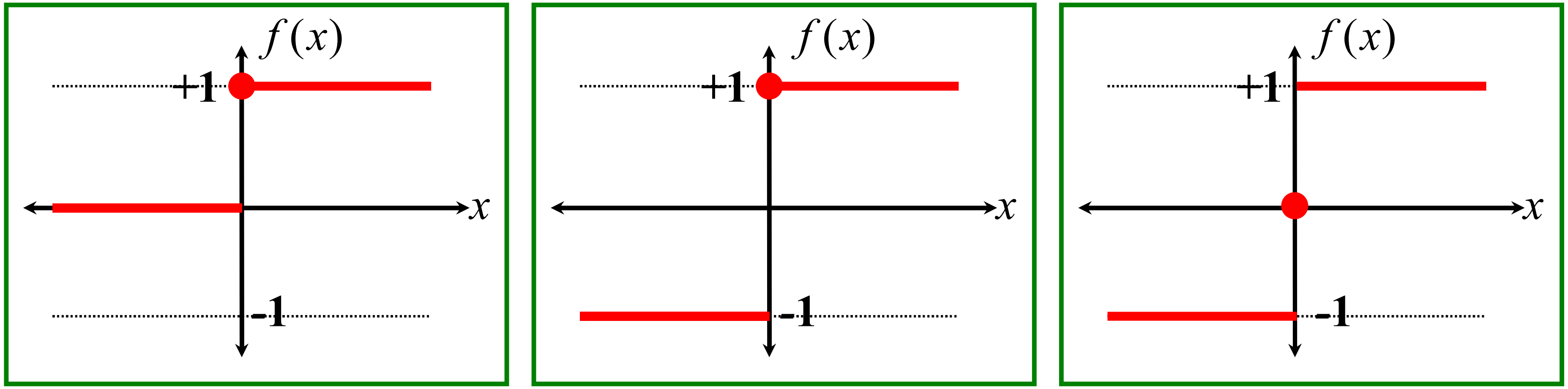


Logistic (Activation) Function

- If f(x)=x, ADALINE is obtained
- This model is a building block for interconnected networks
- Activation function is generally a hyperbolic tangent, a sigmoid, a hard limiting function or a linear expression.

Neuron and Its Analytic Model

Activation Functions



$$f(x)=\begin{cases}1 & x\geq 0\\ 0 & x<0\end{cases}$$

Unipolar

$$f(x)=\begin{cases}1 & x\geq 0\\ -1 & x<0\end{cases}$$

Bipolar

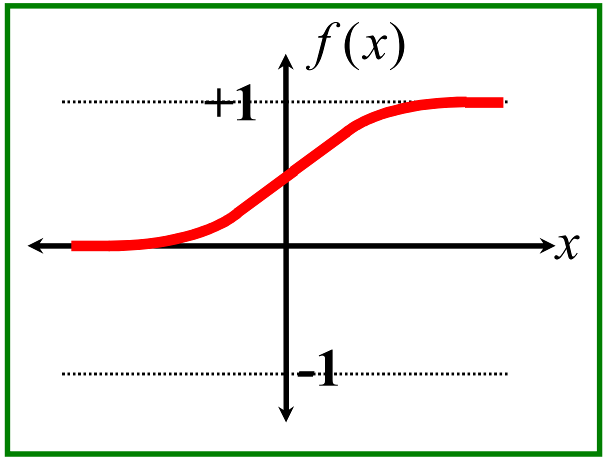
$$f(x)=\operatorname{sgn}(x)$$

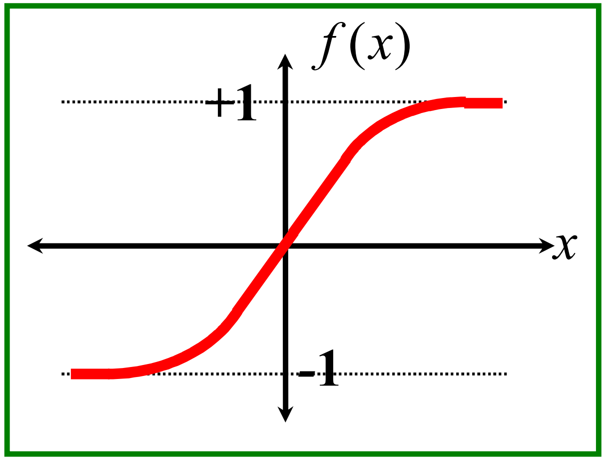
Bipolar

- None of them is differentiable with respect to x
- Note that the decision boundary at x=0 can be changed

Neuron and Its Analytic Model

Activation Functions





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

Unipolar

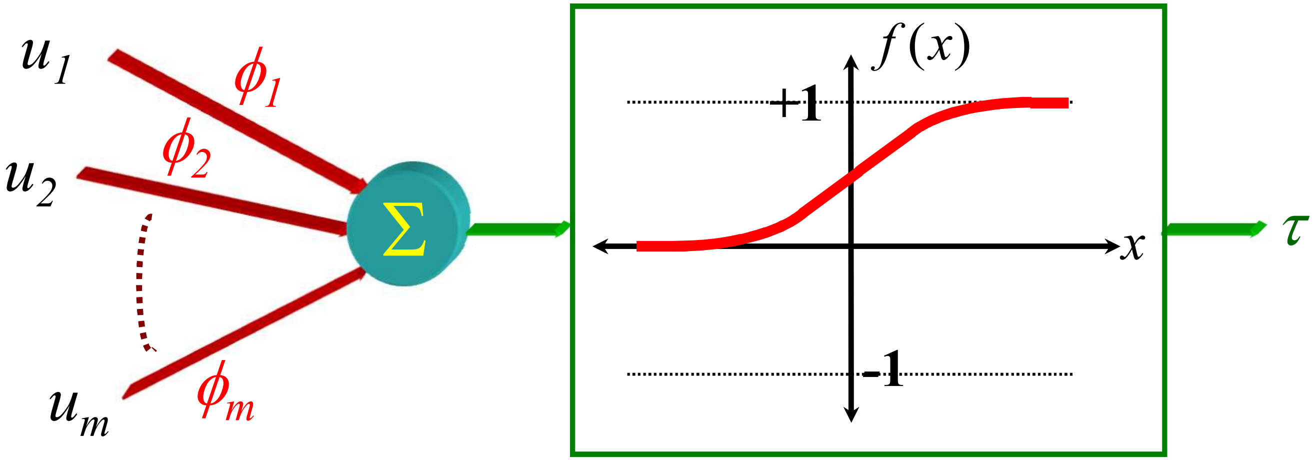
$$f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} = \tanh(x)$$

Bipolar

- Both of them are differentiable with respect to x
- Note that the decision boundary is smooth now!

Neuron and Its Analytic Model

Activation Functions



Be reasonable! Such a system cannot realize negative values, so what you can expect from it has to be nonnegative

Neuron and Its Analytic Model

Activation Functions

Table 1 Adjustable Parameters and Number of Adjustable Parameters for Each Model

Label	Activation function	Number of adjustables associated to the network	Derivative computation
tanh	$f=\tanh(S)$	$(m+2)R+1$	$\frac{\partial f(S)}{\partial S}=1-f(S)^2$
polyexp See [2]	$f=\left(aS^2+bS+c\right)\exp\left(-\lambda S^2\right)$	$(m+2)R+1+4R$	$\frac{\partial f(a,b,c,S)}{\partial a}=S^2\exp\left(-\lambda S^2\right),\ \frac{\partial f(a,b,c,S)}{\partial b}=S\exp\left(-\lambda S^2\right),\ \frac{\partial f(a,b,c,S)}{\partial c}=\exp\left(-\lambda S^2\right)$ $\frac{\partial f(a,b,c,S)}{\partial S}=\left(-2\lambda S^3-2\lambda bS^2+(2a-2\lambda c)S+b\right)\exp\left(-\lambda S^2\right)$
quan See [8]	$f=\frac{1}{2M+1}\sum_{k=-M}^{M}\tanh(S-\lambda k)$	$(m+2)R+1+R$	$\frac{\partial f(S,\lambda)}{\partial S}=\frac{-1}{2M+1}\sum_{k=-M}^{M}(\tanh(S-\lambda k))^2$ $\frac{\partial f(S,\lambda)}{\partial \lambda}=\frac{1}{2M+1}\sum_{k=-M}^{M}k(\tanh(S-\lambda k))^2$
sinc	$f=\begin{cases}\sin(\pi S)/\pi S & S\neq 0\\ 1 & S=0\end{cases}$	$(m+2)R+1$	$\frac{\partial f(S)}{\partial S}=\begin{cases}(\cos(\pi S)-\mathrm{sinc}(S))/S & S\neq 0\\ 0 & S=0\end{cases}$
sincos	$f=a\sin(pS)+b\cos(qS)$	$(m+2)R+1+4R$	$\frac{\partial f(a,b,p,q,S)}{\partial a}=\sin(pS),\ \frac{\partial f(a,b,p,q,S)}{\partial b}=\cos(qS),\ \frac{\partial f(a,b,p,q,S)}{\partial p}=aS\cos(pS)$ $\frac{\partial f(a,b,p,q,S)}{\partial q}=-bS\sin(qS),\ \frac{\partial f(a,b,p,q,S)}{\partial a}=ap\cos(pS)-bq\sin(qS)$
wave See [2]	$f=\left(1-S^2\right)\exp\left(-\lambda S^2\right)$	$(m+2)R+1+R$	$\frac{\partial f(S,\lambda)}{\partial \lambda}=-S^2 f(S,\lambda),\ \frac{\partial f(S,\lambda)}{\partial S}=2S(\lambda S^2-\lambda-1)e^{-\lambda S^2}$
atan See [1]	$f=\mathrm{atan}(S)$	$(m+2)R+1$	$\frac{\partial f(S)}{\partial S}=\frac{1}{1+S^2}$
log See [1]	$f=\begin{cases}\ln(S+1) & S\geq 0\\ -\ln(-S+1) & S<0\end{cases}$	$(m+2)R+1$	$\frac{\partial f(S)}{\partial S}=\frac{1}{1+\lvert S\rvert}$

M.Ö. Efe, "Novel Neuronal Activation Functions for Feedforward Neural Networks," *Neural Processing Letters*, v.28, no.2, pp.63-79, October 2008.

Some Preliminary Mathematics
Inner Product

$$\underline{w} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix}$$

$$\underline{o} = \begin{bmatrix} o_1 \\ o_2 \\ \vdots \\ o_n \end{bmatrix}$$

$$\underline{w}^{\mathrm{T}} \underline{o} = \sum_{i=1}^{n} w_i o_i$$

Some Preliminary Mathematics

Derivative for Inner Product

$$\underline{w} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix}$$

$$\underline{o} = \begin{bmatrix} o_1 \\ o_2 \\ \vdots \\ o_n \end{bmatrix}$$

$$\underline{w}^{\mathrm{T}} \underline{o} = \sum_{i=1}^{n} w_i o_i := S$$

$$\frac{\partial S}{\partial w_j} = o_j$$ where j=1,2,..,n

$$\frac{\partial S}{\partial o_k} = w_k$$ where k=1,2,..,n

Some Preliminary Mathematics

Matrix-Vector Multiplication

$$W = \begin{bmatrix} W_{11} & W_{12} & \cdots & W_{1m} \\ W_{21} & W_{22} & \cdots & W_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ W_{n1} & W_{n2} & \cdots & W_{nm} \end{bmatrix}$$

$$\underline{o} = \begin{bmatrix} o_1 \\ o_2 \\ \vdots \\ o_m \end{bmatrix}$$

$$W\underline{o} = \begin{bmatrix} W_{11} & W_{12} & \cdots & W_{1m} \\ W_{21} & W_{22} & \cdots & W_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ W_{n1} & W_{n2} & \cdots & W_{nm} \end{bmatrix} \begin{bmatrix} o_1 \\ o_2 \\ \vdots \\ o_m \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{m} W_{1i} o_i \\ \sum_{i=1}^{m} W_{2i} o_i \\ \vdots \\ \sum_{i=1}^{m} W_{ni} o_i \end{bmatrix} := \underline{S}$$

Some Preliminary Mathematics

Derivative

$$W\underline{o} = \begin{bmatrix} W_{11} & W_{12} & \cdots & W_{1m} \\ W_{21} & W_{22} & \cdots & W_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ W_{n1} & W_{n2} & \cdots & W_{nm} \end{bmatrix} \begin{bmatrix} o_1 \\ o_2 \\ \vdots \\ o_m \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{m} W_{1i} o_i \\ \sum_{i=1}^{m} W_{2i} o_i \\ \vdots \\ \sum_{i=1}^{m} W_{ni} o_i \end{bmatrix} := \underline{S}$$

$$\frac{\partial \underline{S}}{\partial \underline{o}} = \begin{bmatrix} \frac{\partial S_1}{\partial o_1} & \frac{\partial S_1}{\partial o_2} & \cdots & \frac{\partial S_1}{\partial o_m} \\ \frac{\partial S_2}{\partial o_1} & \frac{\partial S_2}{\partial o_2} & \cdots & \frac{\partial S_2}{\partial o_m} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial S_n}{\partial o_1} & \frac{\partial S_n}{\partial o_2} & \cdots & \frac{\partial S_n}{\partial o_m} \end{bmatrix} = W$$

Some Preliminary Mathematics

Derivative for Several Activation Functions

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

$$\frac{df(x)}{dx} = \frac{0*(1+e^{-x}) - (-e^{-x})*1}{\left(1+e^{-x}\right)^2} = \frac{1}{1+e^{-x}}\left(1 - \frac{1}{1+e^{-x}}\right) = f(x)\left(1 - f(x)\right)$$

Some Preliminary Mathematics

Derivative for Several Activation Functions

$$f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} = \tanh(x)$$

$$\frac{df(x)}{dx} = \frac{\left(e^{x} + e^{-x}\right)^{2} - \left(e^{x} - e^{-x}\right)^{2}}{\left(e^{x} + e^{-x}\right)^{2}} = 1 - \left(\frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}\right)^{2} = 1 - f\left(x\right)^{2}$$

Some Preliminary Mathematics

Derivative for Quadratic Functions

$$J(x)=\frac{1}{2}x^{2} \quad\Rightarrow\quad \frac{\partial J(x)}{\partial x}=x$$

$$J(\underline{S})=\frac{1}{2}\underline{S}^{\mathrm{T}}\underline{S} \quad\Rightarrow\quad \frac{\partial J(\underline{S})}{\partial \underline{S}}=\underline{S}^{\mathrm{T}}$$

$$J(\underline{S})=\frac{1}{2}(\underline{D}-\underline{S})^{\mathrm{T}}(\underline{D}-\underline{S}) \quad\Rightarrow\quad \frac{\partial J(\underline{S})}{\partial \underline{S}}=-(\underline{D}-\underline{S})^{\mathrm{T}}$$

where $\underline{D}$ is another vector of appropriate dimensions

Some Preliminary Mathematics

Inner Product as a Measure of Similarity

$$\underline{w} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix} \qquad \underline{o} = \begin{bmatrix} o_1 \\ o_2 \\ \vdots \\ o_n \end{bmatrix}$$

$$\underline{w}^{\mathrm{T}} \underline{o} = \sum_{i=1}^{n} w_i o_i := S$$

$$\langle \underline{w}, \underline{o} \rangle = \underline{w}^{T} \underline{o} = |\underline{w}||\underline{o}| \cos \alpha$$

For $n=1$, $\alpha=0$

For $n=2,3$ α can be found by geometric relations

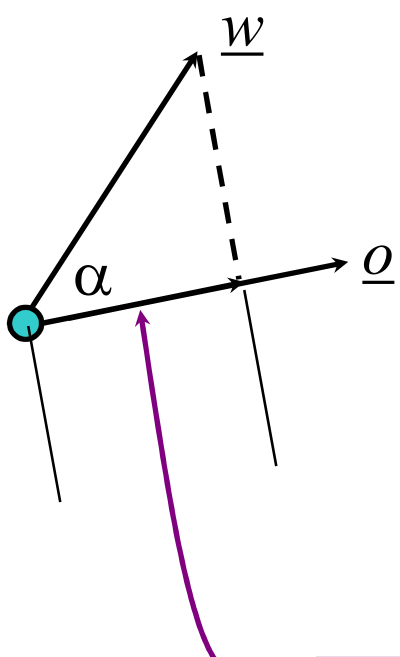
For $n\geq 4$ Finding α may be tedious

Let's see how it measures similarity for n=2

Some Preliminary Mathematics

Inner Product as a Measure of Similarity

Let's see how it measures similarity for n=2



Notice that, keeping the lengths same, they are most similar when $\alpha=0$, indeed they become identical.

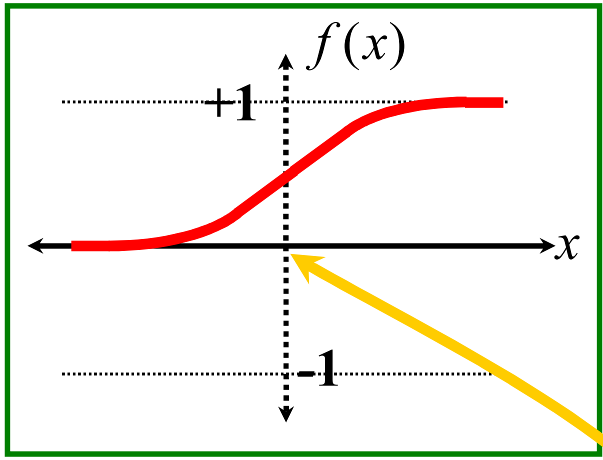
When $\alpha\neq 90$, the two vectors are dissimilar.

For $n\geq 4$, nothing changes, simply calculate $\underline{w}^T\underline{o}$. Basically, a neuron fires when the input vector is similar to its weight vector.

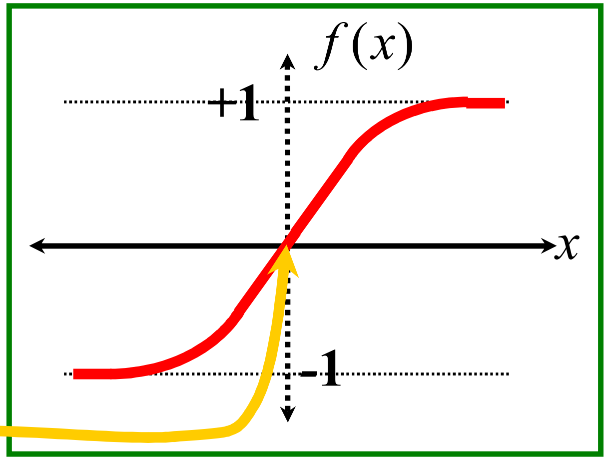
$$\langle \underline{w},\underline{o}\rangle = \underline{w}^T\underline{o} = |\underline{w}||\underline{o}|\cos\alpha$$

Neuron and Its Analytic Model

Activation Functions - Shifting the origin with a threshold θ



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



$$f(x)=\frac{e^{x-\theta}-e^{-(x-\theta)}}{e^{x-\theta}+e^{-(x-\theta)}}=\tanh(x-\theta)$$

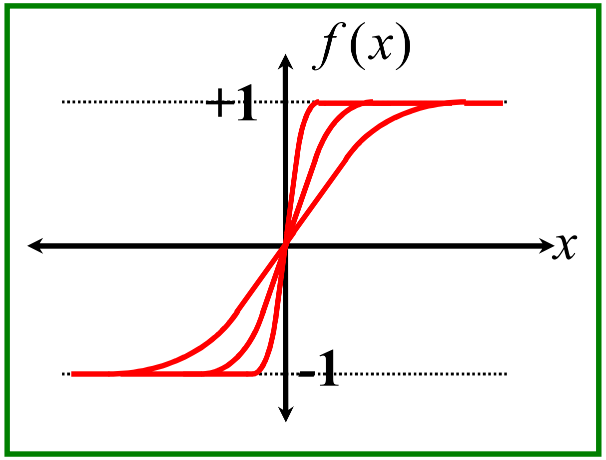
$$x=\theta$$

Mehmet Önder Efe, *Neural Networks*, Lecture Notes, 2022.

Neuron and Its Analytic Model

Activation Functions - Adding a slope parameter λ

$$f(x) = \frac{e^{\lambda x} - e^{-\lambda x}}{e^{\lambda x} + e^{-\lambda x}} = \tanh(\lambda x)$$



Notice that this changes the derivative

$$\frac{df(x)}{dx} = \lambda\left(1 - f(x)^2\right)$$

Neuron and Its Analytic Model

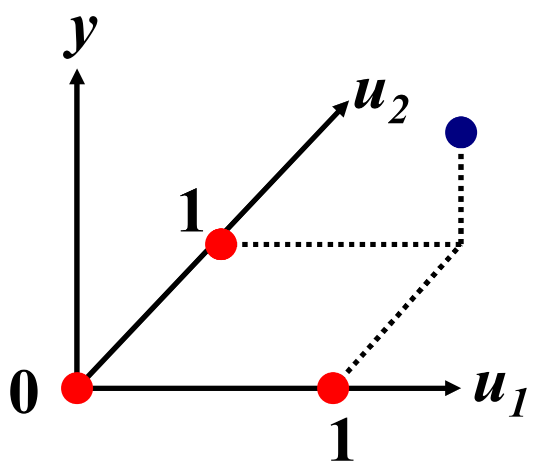
Concept of Learning (Tuning, Adaptation or Parameter Adjustment)

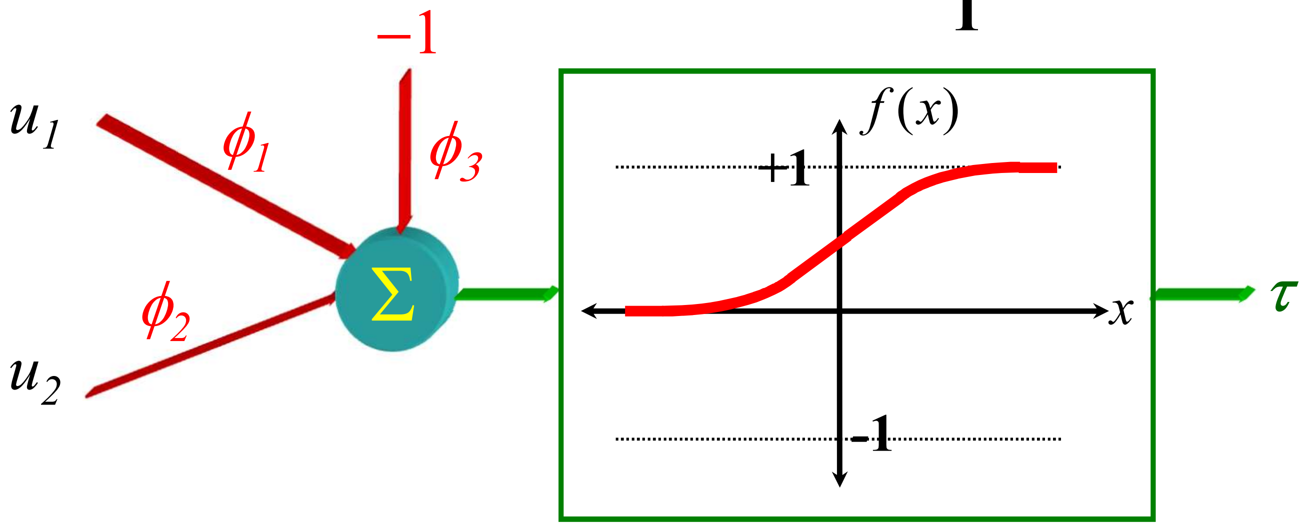
- Learning is the process of searching a parameter set.
- The goal of learning is to minimize some cost or maximize some profit function.
- For Neural Networks, learning is *to change the weights and biases appropriately*.
- This process is also called Parameter Adaptation, Parameter Tuning, Parameter Adjustment or Training.

Neuron and Its Analytic Model

Concept of Learning (Tuning, Adaptation or Parameter Adjustment)

u_1	u_2	y
0	0	0
0	1	0
1	0	0
1	1	1





Mehmet Önder Efe, *Neural Networks*, Lecture Notes, 2022.

Neuron and Its Analytic Model

Concept of Learning (Tuning, Adaptation or Parameter Adjustment)

There are 3 parameters

At each step (time instant) update them by calculating the **corrective information**

$$\phi_i^{new} = \phi_i^{current} + \Delta\phi_i$$

$$\phi_1(k+1) = \phi_1(k) + \Delta\phi_1(k)$$
$$\phi_2(k+1) = \phi_2(k) + \Delta\phi_2(k)$$
$$\phi_3(k+1) = \phi_3(k) + \Delta\phi_3(k)$$

Neuron and Its Analytic Model

Concept of Learning (Tuning, Adaptation or Parameter Adjustment)

u_1	u_2	y
0	0	0
0	1	0
1	0	0
1	1	1

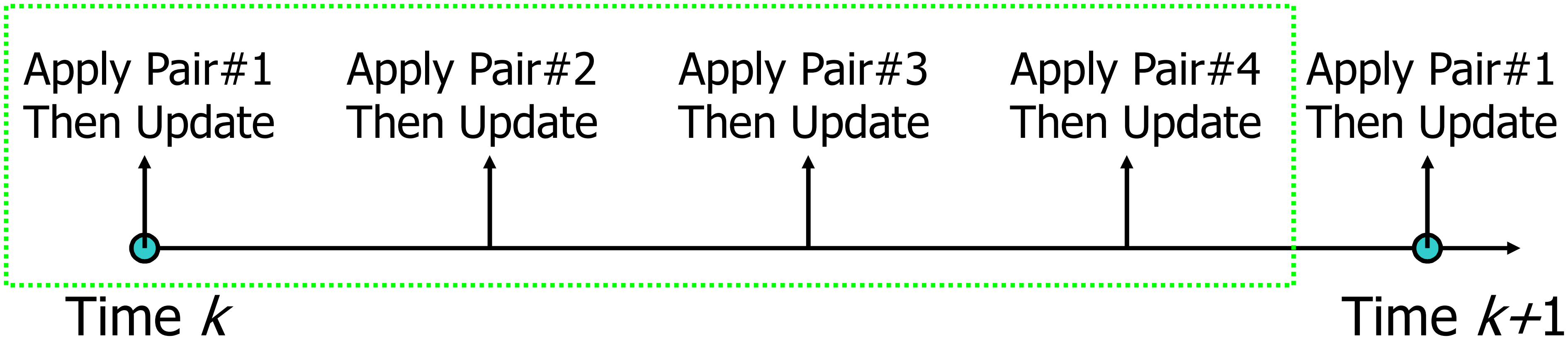
Input vector

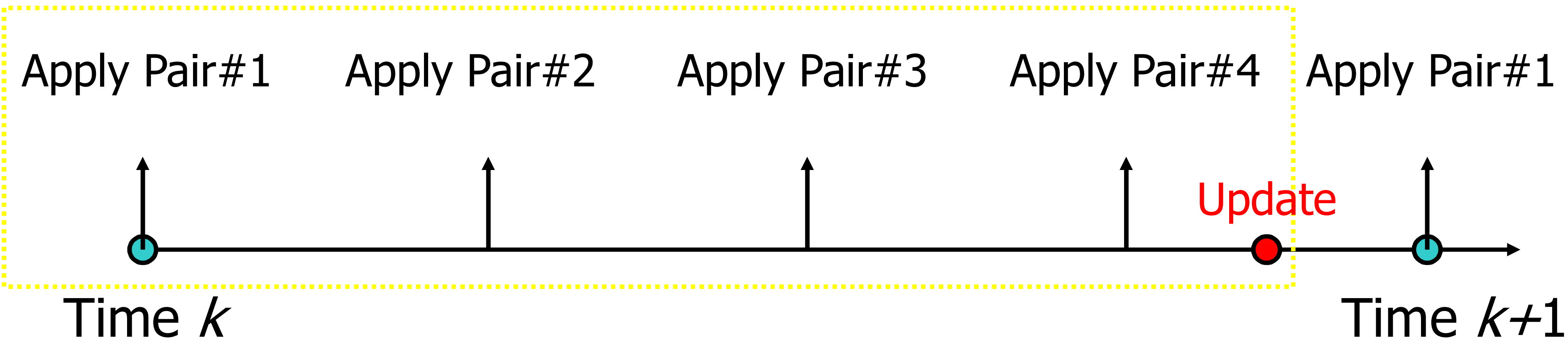
Output vector

An input/output pair
A training pair
A pair
A sample

Neuron and Its Analytic Model

Concept of Learning (Pattern and Batch)

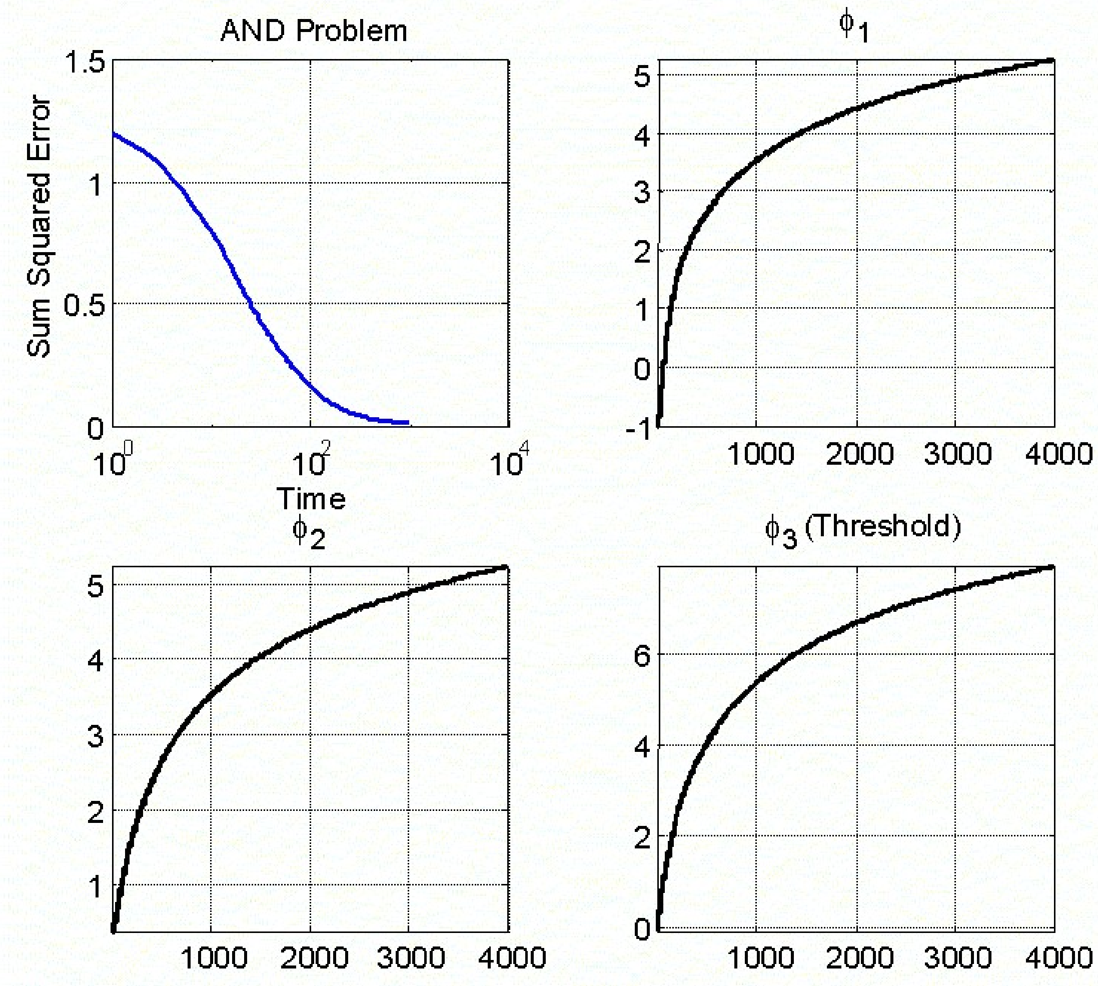




Mehmet Önder Efe, *Neural Networks*, Lecture Notes, 2022.

Neuron and Its Analytic Model

Concept of Learning (Tuning, Adaptation or Parameter Adjustment)



Mehmet Önder Efe, *Neural Networks*, Lecture Notes, 2022.

```
clear all
close all
clc

U = [0 0;0 1;1 0;1 1];
Y = [0 0 0 1]';

Phi = 2*rand(3,1)-1;
Eta = 0.8;
PHI(1,:)=Phi';

for count=1:1000
  epoche_error(count) = 0;
  for sample=1:4
    inputvector=[U(sample,:)';-1];
    Yn(sample) = 1/(1+exp(-(Phi'*inputvector)));
    errorvector = Y(sample)-Yn(sample);
    Phi=Phi+Eta*errorvector*Yn(sample)*(1-Yn(sample))*inputvector;
    PHI((count-1)*4+sample+1,:)=Phi';
    epoche_error(count) = epoche_error(count) + errorvector'*errorvector;
  end
  epoche_error(count)
end
[U,Y,Yn']
```

```
» [U,Y,Yn']

ans =

   0   0   0   0.0004
   0   1   0   0.0625
   1   0   0   0.0626
   1   1   1   0.9255
```

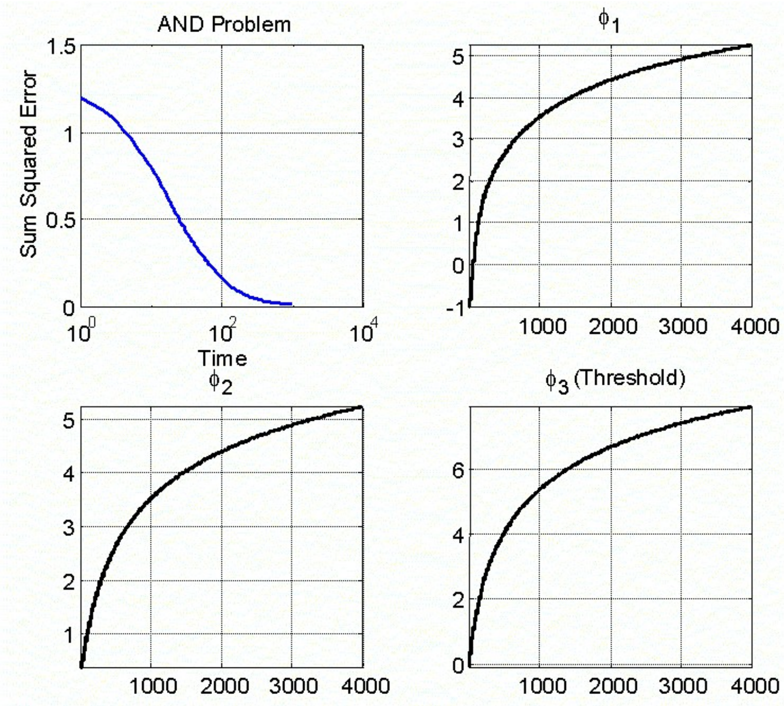
We will see how this works!

$$\phi(k+1) = \phi(k) + \eta e(k)\big(f(k)(1-f(k))\big)u(k)$$

$$\frac{1}{1+e^{-x}}$$

```
for count=1:1000
  epoche_error(count) = 0;
  for sample=1:4
    inputvector=[U(sample,:)';-1];
    Yn(sample) = 1/(1+exp(-(Phi'*inputvector)));
    errorvector = Y(sample)-Yn(sample);
    Phi=Phi+Eta*errorvector*Yn(sample)*(1-Yn(sample))*inputvector;
    PHI((count-1)*4+sample+1,:)=Phi';
    epoche_error(count) = epoche_error(count) + errorvector'*errorvector;
  end
  epoche_error(count)
end
[U,Y,Yn']
```

We will see how this works!



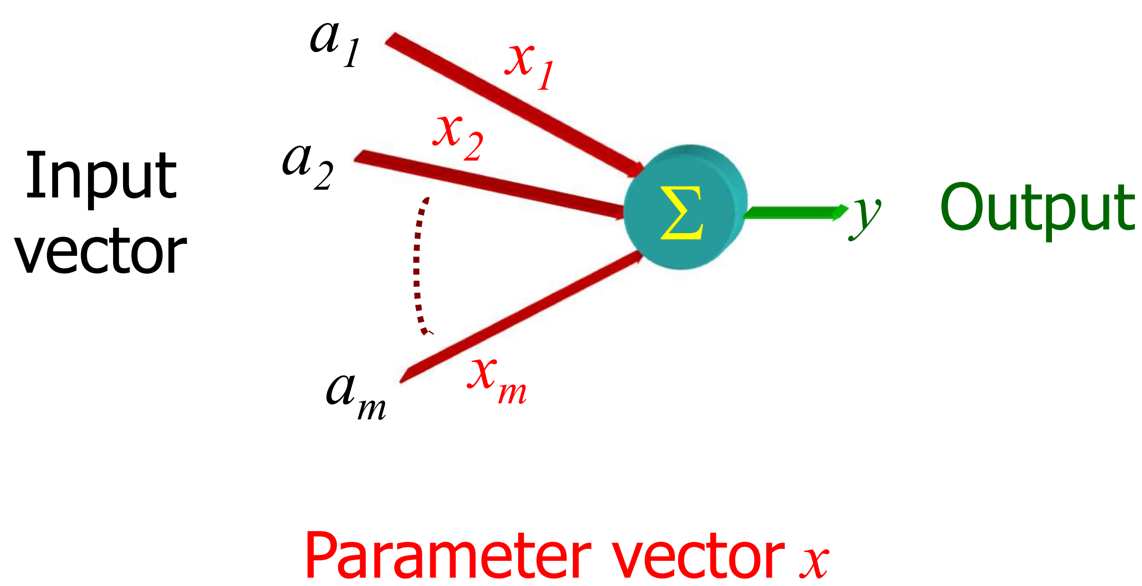
$$f = \frac{1}{1 + e^{-\phi^T u}} = 0 \Rightarrow \phi^T u = -\infty, \qquad \|\phi^T\|\|u\| = \infty$$

$$\|u\| < \infty \text{ This requires } \|\phi\| = \infty$$

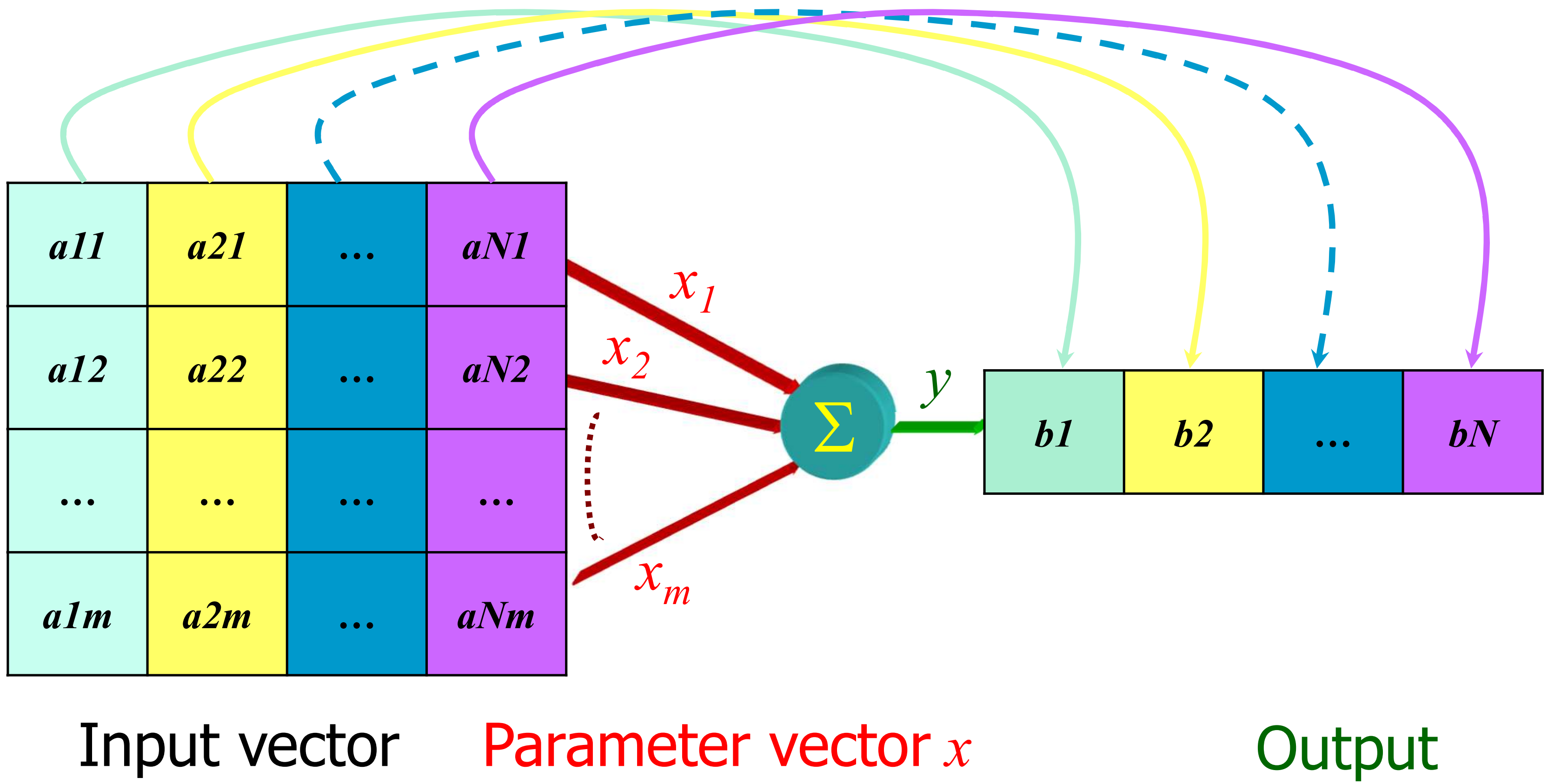
$$f = \frac{1}{1 + e^{-\phi^T u}} = 1 \Rightarrow \phi^T u = \infty, \qquad \|\phi^T\|\|u\| = \infty$$

$$\|u\| < \infty \text{ This requires } \|\phi\| = \infty$$

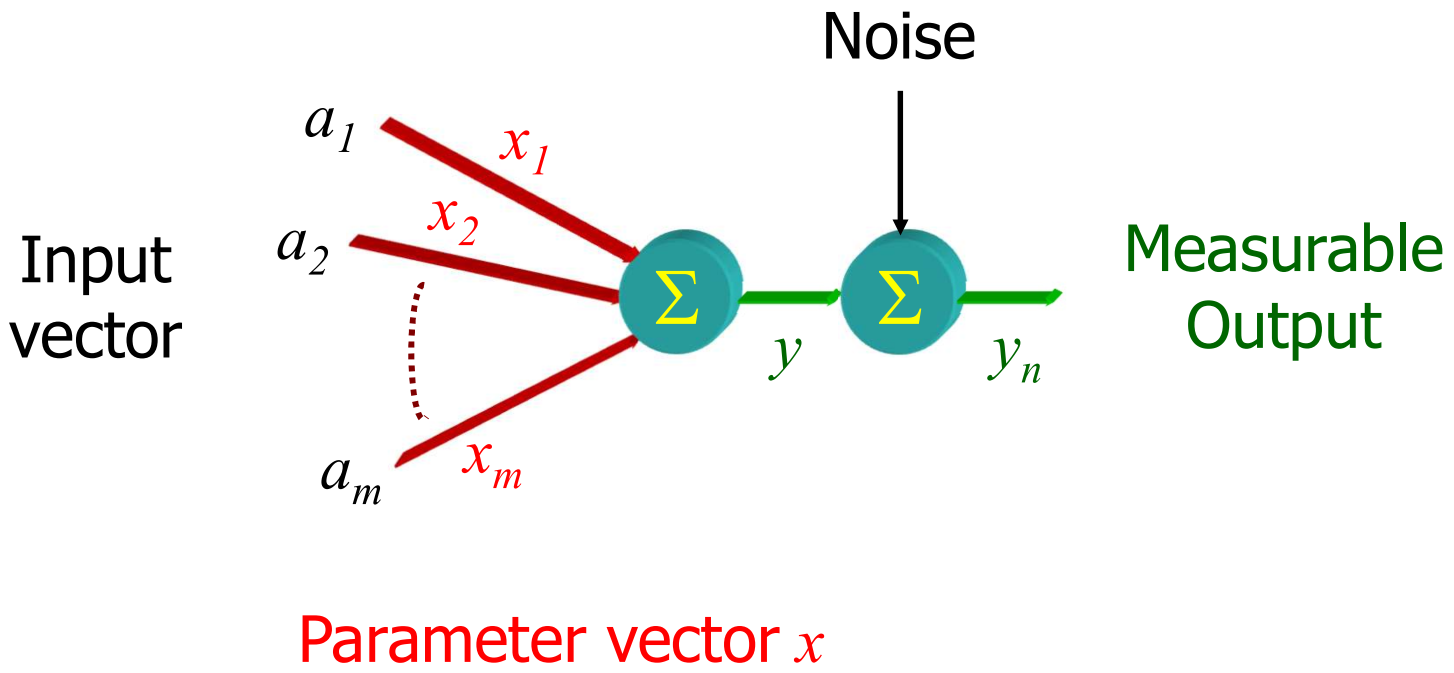
Least Squares (LS) Algorithm



Least Squares (LS) Algorithm

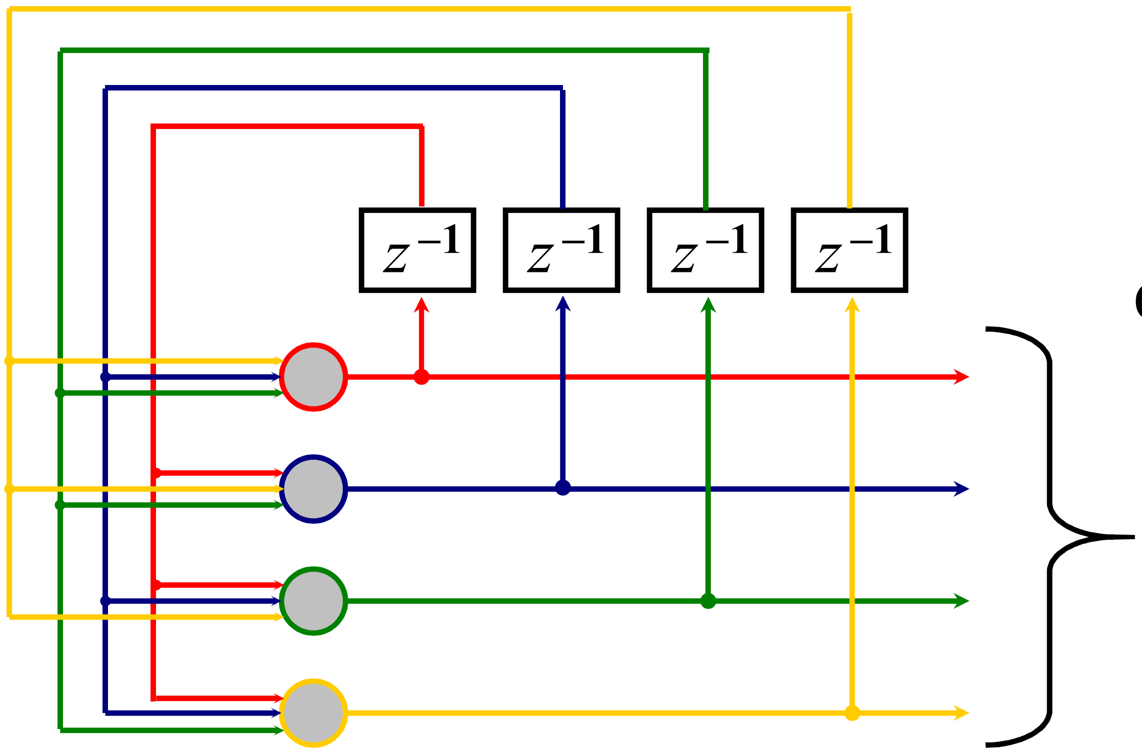


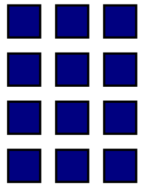
Least Squares (LS) Algorithm



Let's switch to Least Squares document

Hopfield Neural Network





An example canvas for 12 neurons

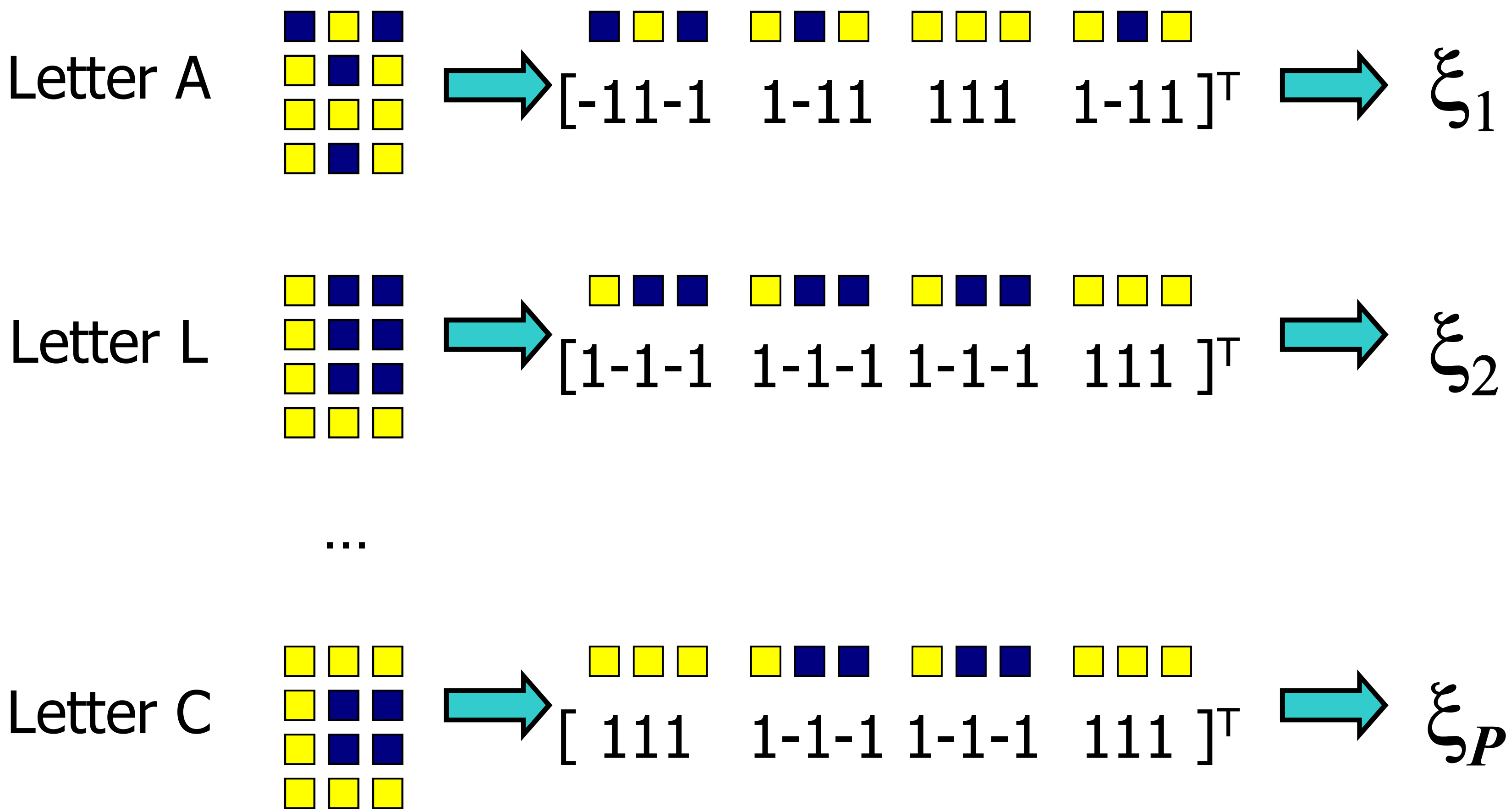
$$y_{k+1} = \mathrm{sgn}(\mathbf{W}y_k)$$

$y_{k+1} = y_k$ means no change!

- Character recognition
- Content Addressable Memory

Hopfield Neural Network

Patterns and Encoding



There are P patterns

Hopfield Neural Network
Computation of $\mathbf{W}$

Either use this one

$$w_{ji} = \begin{cases} \dfrac{1}{N} \sum_{p=1}^{P} \xi_{p,i} \xi_{p,j} & j \neq i \\ 0 & j = i \end{cases}$$

Or this one

$$\widetilde{\mathrm{W}} = \frac{1}{N} \sum_{p=1}^{P} \xi_p \xi_p^T$$

To obtain $\mathbf{W}$, set the diagonal entries of $\widetilde{\mathbf{W}}$ to zero

Hopfield Neural Network

Computation of **W**-What if a new pattern emerges?

$$\tilde{W} = \frac{1}{N}\sum_{p=1}^{P}\xi_p\xi_p^T \quad \text{then remove the diagonal to obtain } W$$

$$W_P = \frac{1}{N}\sum_{p=1}^{P}\left(\xi_p\xi_p^T - \text{diag}(\xi_p\xi_p^T)\right)$$

$$W_{P+1} = \frac{1}{N}\sum_{p=1}^{P+1}\left(\xi_p\xi_p^T - \text{diag}(\xi_p\xi_p^T)\right)$$

$$= W_P + \frac{1}{N}\left(\xi_{P+1}\xi_{P+1}^T - \text{diag}(\xi_{P+1}\xi_{P+1}^T)\right)$$

Hopfield Neural Network

An Example

Three fundamental memories are the three patterns used to determine W

CODEBOOK, N=30, P=3

```
ooooo    XoXoX    XXXXX
ooooo    XoXoX    XoooX
ooooo    XoXoX    XoooX
XXXXX    XXoXX    XoooX
XXXXX    XXoXX    XoooX
XXXXX    XXoXX    XXXXX
```

Initial state

```
ooooo  →  oXoXo  →  XoXoX  →  XoXoX
ooXoX     oXoXX     XoXoX     XoXoX
ooXoX     oXoXX     XoXoX     XoXoX
oXoXX     ooooo     XXoXX     XXoXX
oooXX     ooooo     XXoXX     XXoXX
XooXo     ooXoo     XXoXX     XXoXX
```

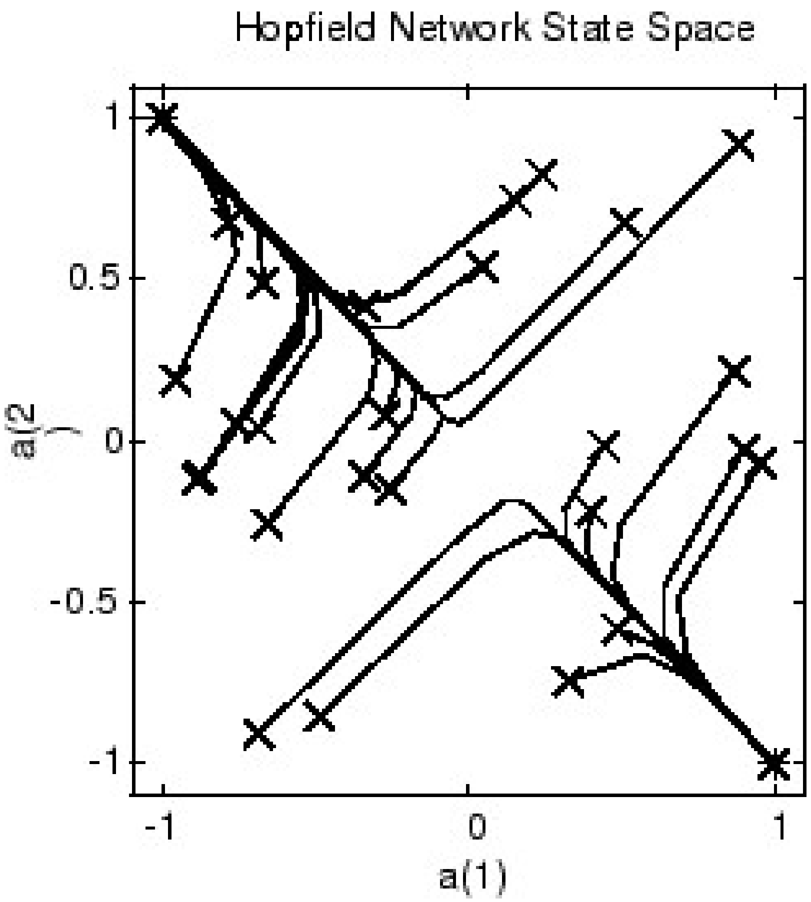
No change at all

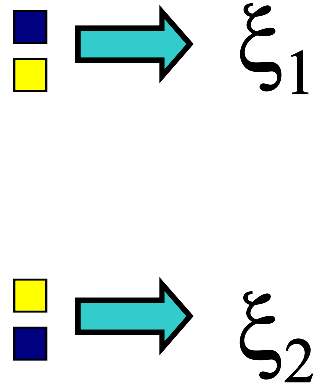
Hopfield Neural Network
An Algorithmic Summary

- Choose the patterns, ξ, which will be the fundamental memories.
- Storage: Compute $\mathbf{W}$ (Notice this is a one-shot computation, i.e. no iterations on $\mathbf{W}$).
- Initialization: Set the output vector to a N-dimensional vector, which may be a corrupted version of fundamental memories.
- Run: Iterate $\boldsymbol{y}_{k+1} = \mathrm{sgn}(\mathbf{W}\boldsymbol{y}_k)$ until convergence.

Hopfield Neural Network

State Space





Mehmet Önder Efe, *Neural Networks*, Lecture Notes, 2022.

Hopfield Neural Network

HOMEWORK #1

- Choose your canvas, N (neurons)
  Choose your P patterns and encode them
  Find $\mathbf{W}$ (Now your network is ready)

- For every pattern from your library of patterns:
  Perturb it according to the perturbation procedure
  Run your network get the result

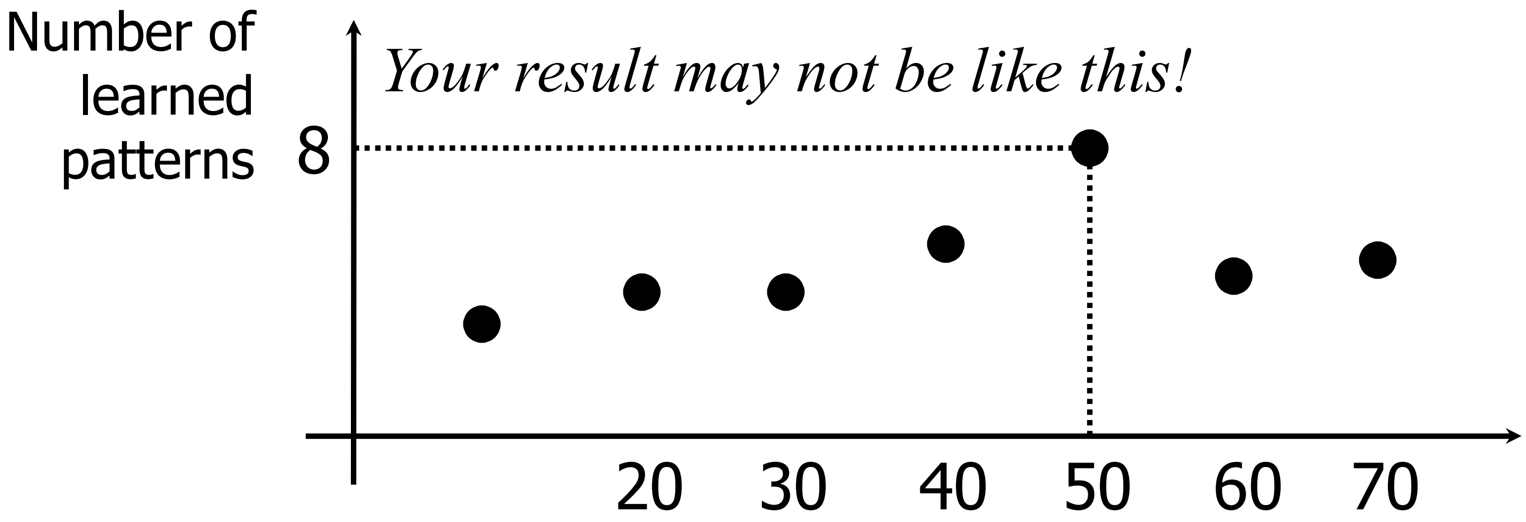
- Determine empirically the learning capacity of your network in terms of N.

- Perturbation procedure
  For every bit of the chosen pattern
  Generate a random number by using rand command
  If it is bigger than 0.3 reverse that bit
  Otherwise leave it as it is

Hopfield Neural Network

HOMEWORK #1

- If every pattern is learned, increase P, and repeat everything until you find the limit of P for that N.
- "Code" everything in Matlab, submit it.
  Insert as much comments as possible
  Give a plot like the one below
  Due date is 2-weeks from today!



Hopfield Neural Network

Remarks on Content Addressability

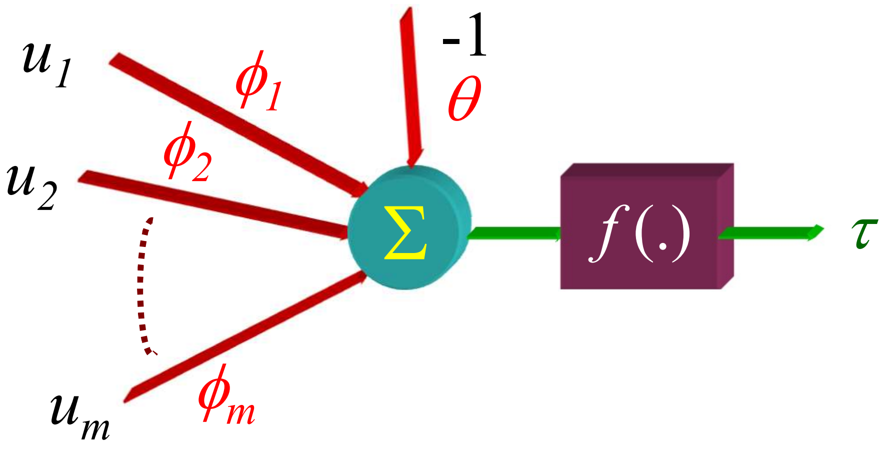
Suppose that an item stored in memory is "**H.A. Kramers & G.H. Wannier Physi Rev. 60, 242 (1941).**" A more general content-addressable memory would be capable of retrieving this entire memory item on the basis of sufficient partial information. The input "**& Wannier (1941)**" might suffice. An ideal memory could deal with errors and retrieve this reference even from the input "**Wannier, (1941).**"

Hopfield, 1982

- Perceptron Learning Algorithms
- Multilayer Perceptron (MLP) and Error Backpropagation
    - Derivation of the Learning Algorithm
    - Problems of Error Backpropagation
    - Memorization (Overfitting) and Generalization
    - Range of Variables (Normalization)

Perceptron Learning Algorithms

Perceptron

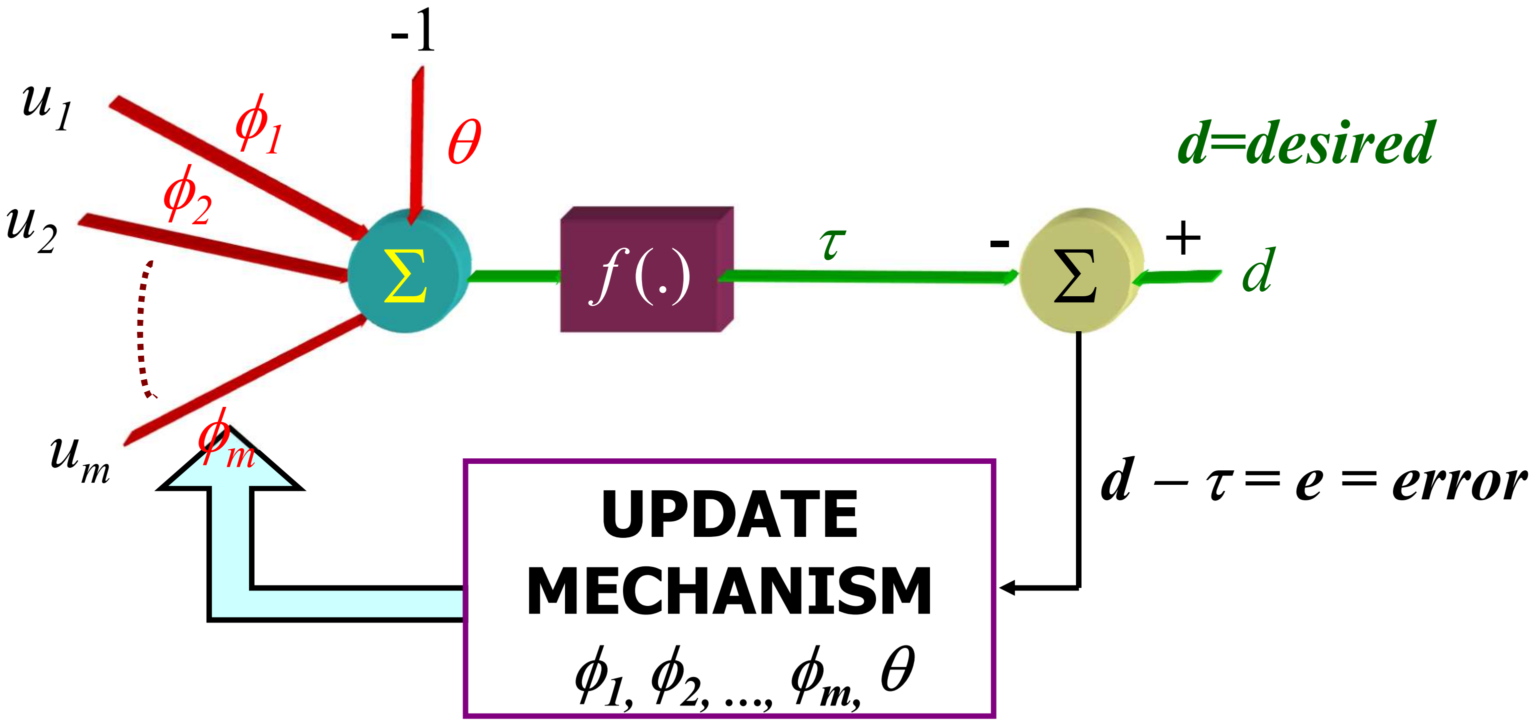


$$\tau = f\left(\underline{\phi}^{\mathrm{T}}\underline{u} - \theta\right)$$

$$\tau = f\left(\sum_{i=1}^{m}\phi_i u_i - \theta\right)$$

- We will discuss this topic for classification purposes
- This model is a building block for interconnected networks
- Several tuning laws (learning algorithms) exist

Perceptron Learning Algorithms
Perceptron with Parameter Update Loop



- A generic pair is : $[\ u_1,\ u_2,\ \ldots,\ u_m,\ d\ ]$

Perceptron Learning Algorithms

Summary for the First Algorithm

- Initialize the weights and the bias to randomly selected small numbers
- Present a pattern **[** $u_1, u_2, \ldots, u_m$ **]** obtain τ
- Calculate error $e = d - \tau$
- Adapt the weights (Choose η and tuning law)

$$\phi_i^{new} = \phi_i^{old} + \eta\, e\, u_i$$
$$\theta^{new} = \theta^{old} + \eta\, e\,(-1)$$

$$f(x) = \begin{cases} 1 & x \geq 0 \\ -1 & x < 0 \end{cases}$$

where η is the learning rate (adaptation gain) satisfying 0< η <1.

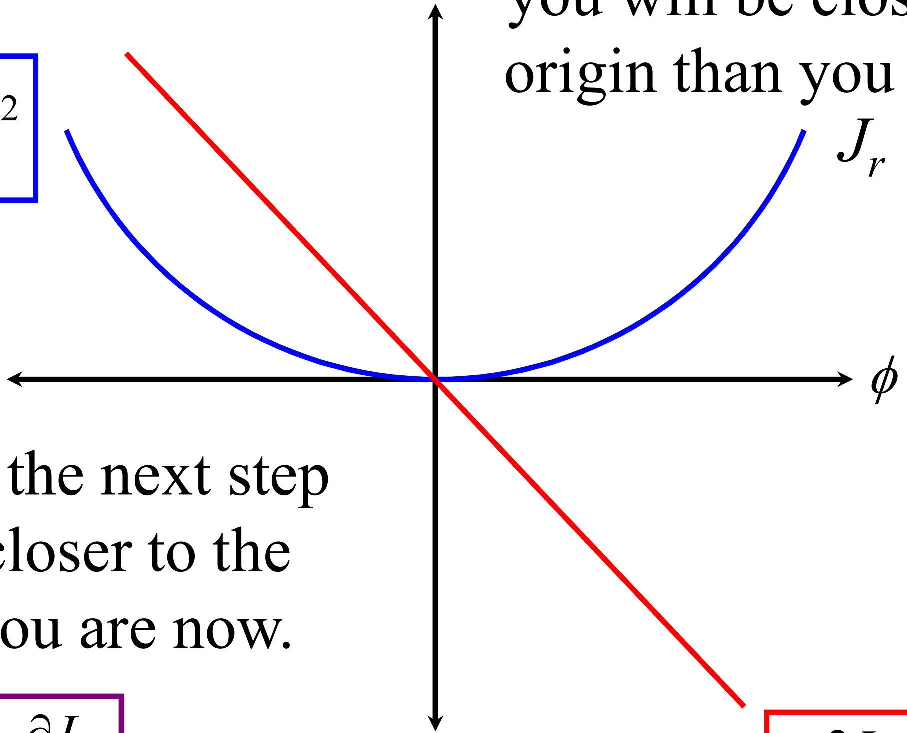
- Above tuning law is known as Hebbian Learning

Perceptron Learning Algorithms

Gradient Descent (MIT Rule)

$\Delta\phi > 0$, i.e. in the next step you will be closer to the origin than you are now.

$$J_r = \frac{1}{2}\phi^2$$



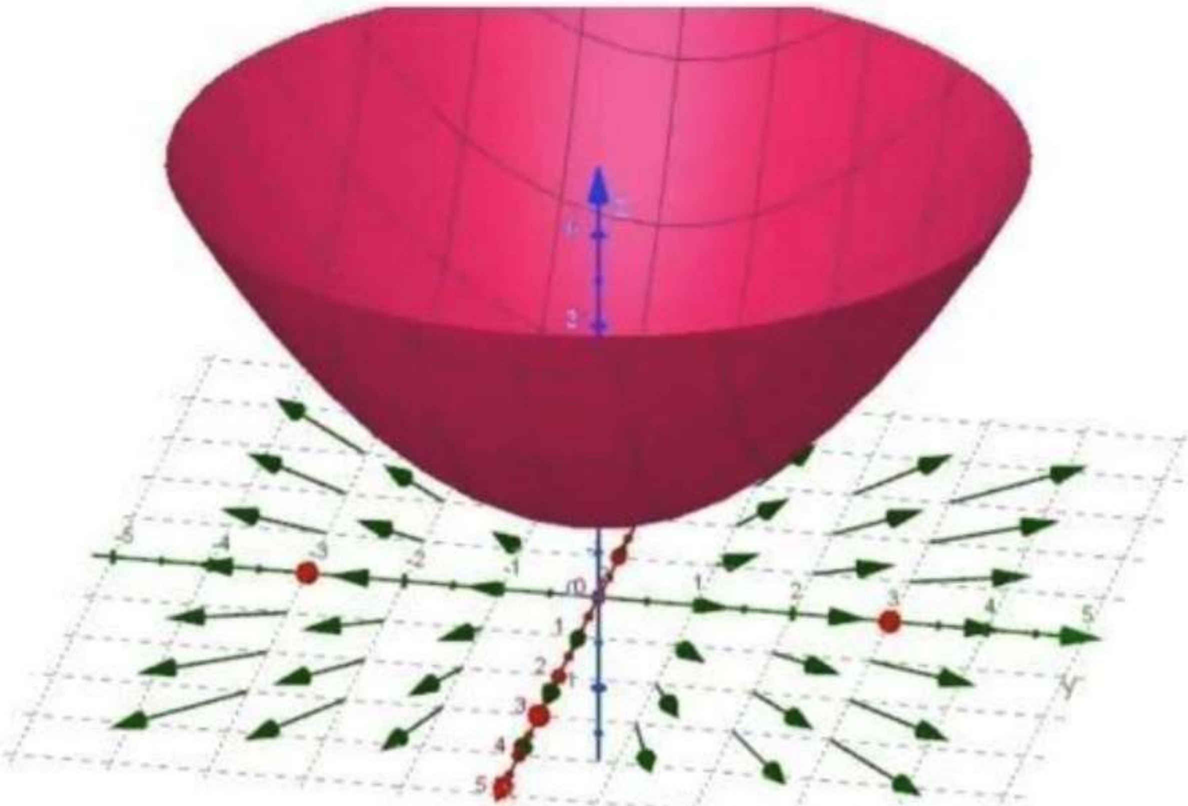
$$\Delta\phi = -\eta \frac{\partial J_r}{\partial \phi}$$

$\Delta\phi > 0$, i.e. in the next step you will be closer to the origin than you are now.

$$\Delta\phi = -\eta \frac{\partial J_r}{\partial \phi}$$

$$-\frac{\partial J_r}{\partial \phi}$$

Perceptron Learning Algorithms
Gradient Descent (MIT Rule)



Perceptron Learning Algorithms
Gradient Descent (MIT Rule)

Define a cost function

$$J_r = \frac{1}{2}e^2 = \frac{1}{2}(d-\tau)^2$$

Gradient Descent is

$$\Delta\phi_i = -\eta\frac{\partial J_r}{\partial\phi_i}$$

$$\begin{aligned}\Delta\phi_i &= -\eta(d-\tau)\frac{\partial(d-\tau)}{\partial\phi_i}\\ &= -\eta(d-\tau)\left(\frac{\partial d}{\partial\phi_i}-\frac{\partial\tau}{\partial\phi_i}\right)\\ &= \eta(d-\tau)\frac{\partial\tau}{\partial\phi_i}\\ &= \eta\, e\frac{\partial\tau}{\partial\phi_i}\end{aligned}$$

- Check the source code we have already seen. It uses gradient descent for parameter tuning

Perceptron Learning Algorithms

Summary for the Second Algorithm

- Initialize the weights and the bias to randomly selected small numbers
- Present a pattern **[** $\boldsymbol{u_1}$**,** $\boldsymbol{u_2}$**, …,** $\boldsymbol{u_m}$**]** obtain τ
- Calculate error $\boldsymbol{e} = \boldsymbol{d} - \tau$
- Adapt the weights (Choose η and tuning law)

$$\Delta\phi_i = \eta\, e \frac{\partial \tau}{\partial \phi_i} = \eta\, e\, f'\left(\underline{\phi}^{\mathrm{T}} \underline{u} - \theta\right) u_i$$

$$\Delta\theta = \eta\, e \frac{\partial \tau}{\partial \phi_i} = \eta\, e\, f'\left(\underline{\phi}^{\mathrm{T}} \underline{u} - \theta\right)(-1)$$

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

or

$$f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} = \tanh(x)$$

where η is the learning rate (adaptation gain) satisfying $0 < \eta < 1$.

- Above tuning law is known as Gradient Descent

Perceptron Learning Algorithms
Summary for the Third Algorithm

- Initialize the weights and the bias to randomly selected small numbers
- Present a pattern **[u_1, u_2, ..., u_m]** obtain τ
- Adapt the weights (Choose η and tuning law)

$$\Delta\phi_i = \eta(1 - d\tau)d\, u_i$$
$$\Delta\theta = \eta(1 - d\tau)d\,(-1)$$

$$f(x) = \begin{cases} 1 & x \geq 0 \\ -1 & x < 0 \end{cases}$$

where η is the learning rate (adaptation gain) satisfying $0 < \eta < 1$.

Perceptron Learning Algorithms

Summary for the Fourth Algorithm

- Initialize the weights and the bias to randomly selected small numbers
- Present a pattern **[** $u_1, u_2, \ldots, u_m$ **]** obtain τ
- Adapt the weights (Choose η and tuning law)

$$\Delta\varphi_i = \begin{cases} -2\eta\ \tau\ u_i & \text{if } \tau \neq d \\ 0 & \text{otherwise} \end{cases} \quad \text{and } \Delta\theta = \begin{cases} -2\eta\ \tau\ (-1) & \text{if } \tau \neq d \\ 0 & \text{otherwise} \end{cases}$$

$$f(x) = \begin{cases} 1 & x \geq 0 \\ -1 & x < 0 \end{cases}$$

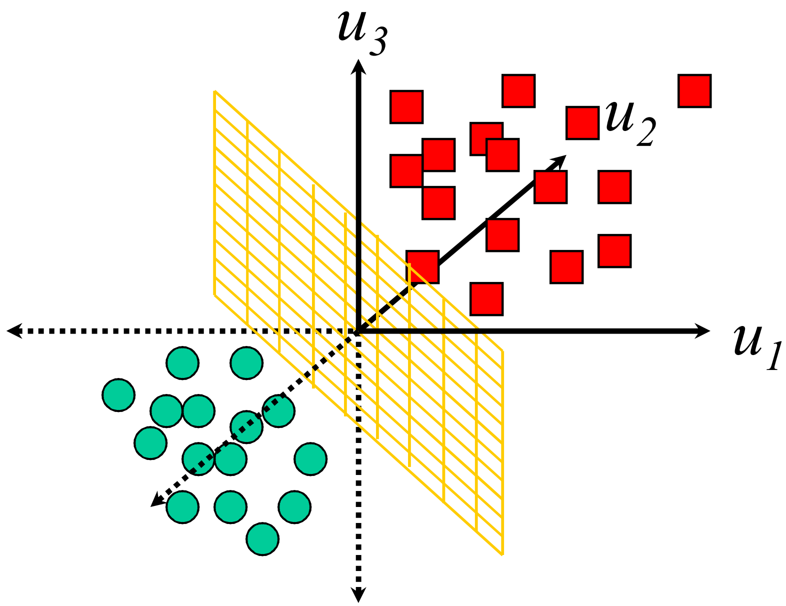
where η is the learning rate (adaptation gain) satisfying $0 < \eta < 1$.

Perceptron Learning Algorithms

HOMEWORK #2

In 3D space generate ten patterns in two different quadrants. This means, you will have 2 classes.

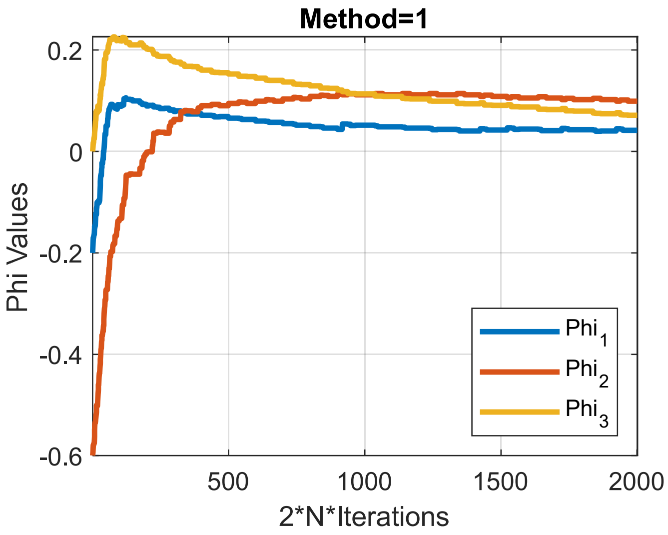
Plot them and show the separating hyperplane by using each one of the methods.

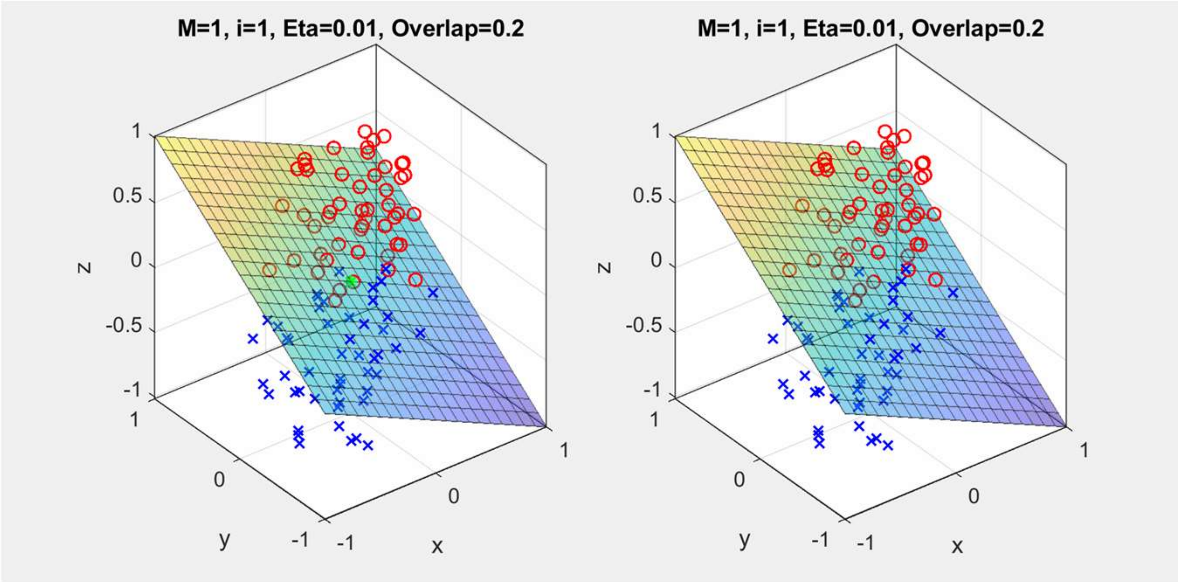


```
% Number of points in each class
N=50;

% Amount of intersection in between the classes
% If Intersect>0 then there will be some overlap in between the classes
Overlap = 0;

% Positive class point coordinates
U1 =  rand(N,3)-Overlap;

% Negative class point coordinates
U2 = -rand(N,3)+Overlap;

% Positive class output
Y1 =  ones(N,1);

% Negative class output
Y2 = -ones(N,1);

% Concatenate the input coordinates
U = [U1;U2];

% COncatenate the output coordinates
Y = [Y1;Y2];

% Initial values of the adjustable parameter vector
Phi = [-0.2 -0.6 0]';

% Learning rapte
Eta = 0.01;

% Data collection variable for Phi
PHI(1,:)=Phi';

% Mesh coordinates
[x y]=meshgrid(-1:0.1:1,-1:0.1:1);

% Chosen adaptation method
method = 1;
```

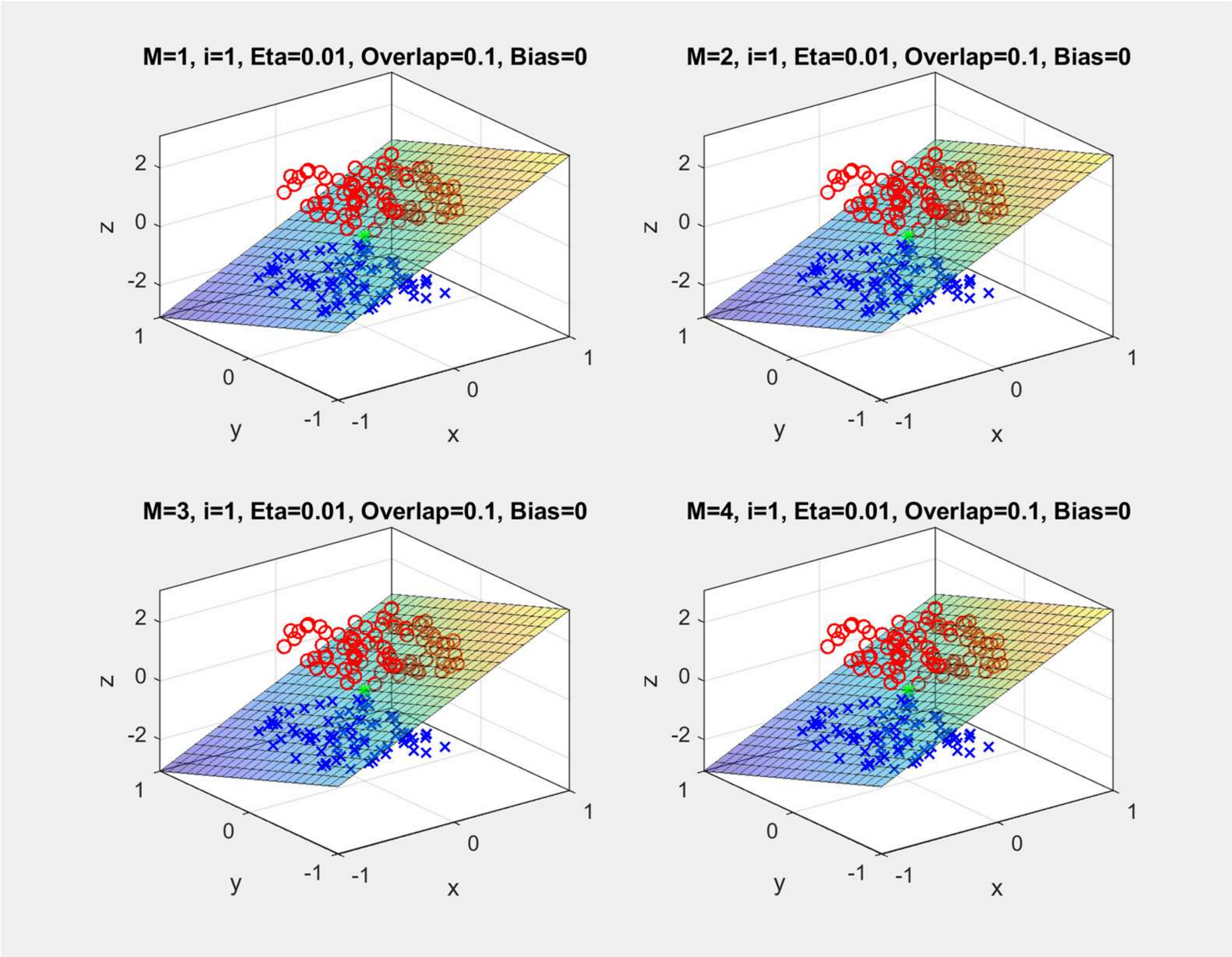
```
% Loop below
for count=1:20

    % Loop for 2N samples available in [U Y] set
    for sample=1:2*N

        % Choose the input pattern coordinates
        inputvector=U(sample,:)';

        % Depending on the 'method' calculate the output
        if method==1 || method==3 || method==4
            Yn(sample) = sign(Phi'*inputvector);
        elseif method==2
            Yn(sample) = tanh(Phi'*inputvector/2);
        else
            disp(' The variable <method> must be 1,2,3 or 4.')
            break
        end

        % Calculate the output error
        error = Y(sample)-Yn(sample);

        % Update laws
        if method ==1
            Phi=Phi+Eta*error*inputvector;
        elseif method ==2
            Phi=Phi+Eta*error*(1/2)*(1-Yn(sample)^2)*inputvector;
        elseif method==3
            Phi = Phi+Eta*(1-Y(sample)*Yn(sample))*Y(sample)*inputvector;
        elseif method==4
            if Y(sample) ~= Yn(sample)
                Phi=Phi-Eta*2*Yn(sample)*inputvector;
            end
        else
            disp(' The variable <method> must be 1,2,3 or 4.')
            break
        end

        % Write the parameters to PHI variable
        PHI=[PHI;Phi'];
    end
end
```

Without bias term (θ=0), little overlap, 50 samples/class, Method=1



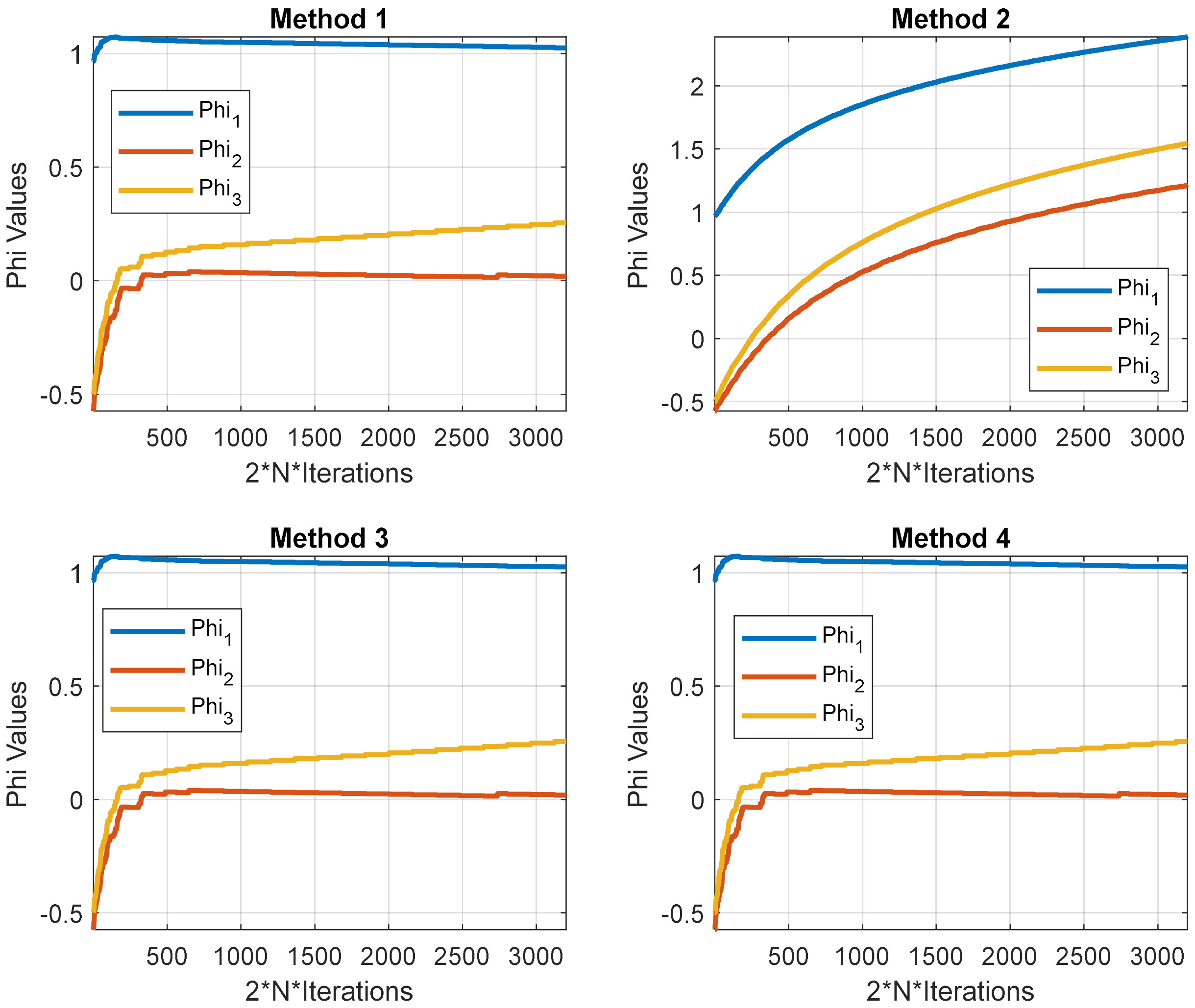


Without bias term (θ=0), little overlap, 80 samples/class



Mehmet Önder Efe, *Neural Networks*, Lecture Notes, 2022.

Without bias term (θ=0), little overlap, 80 samples/class



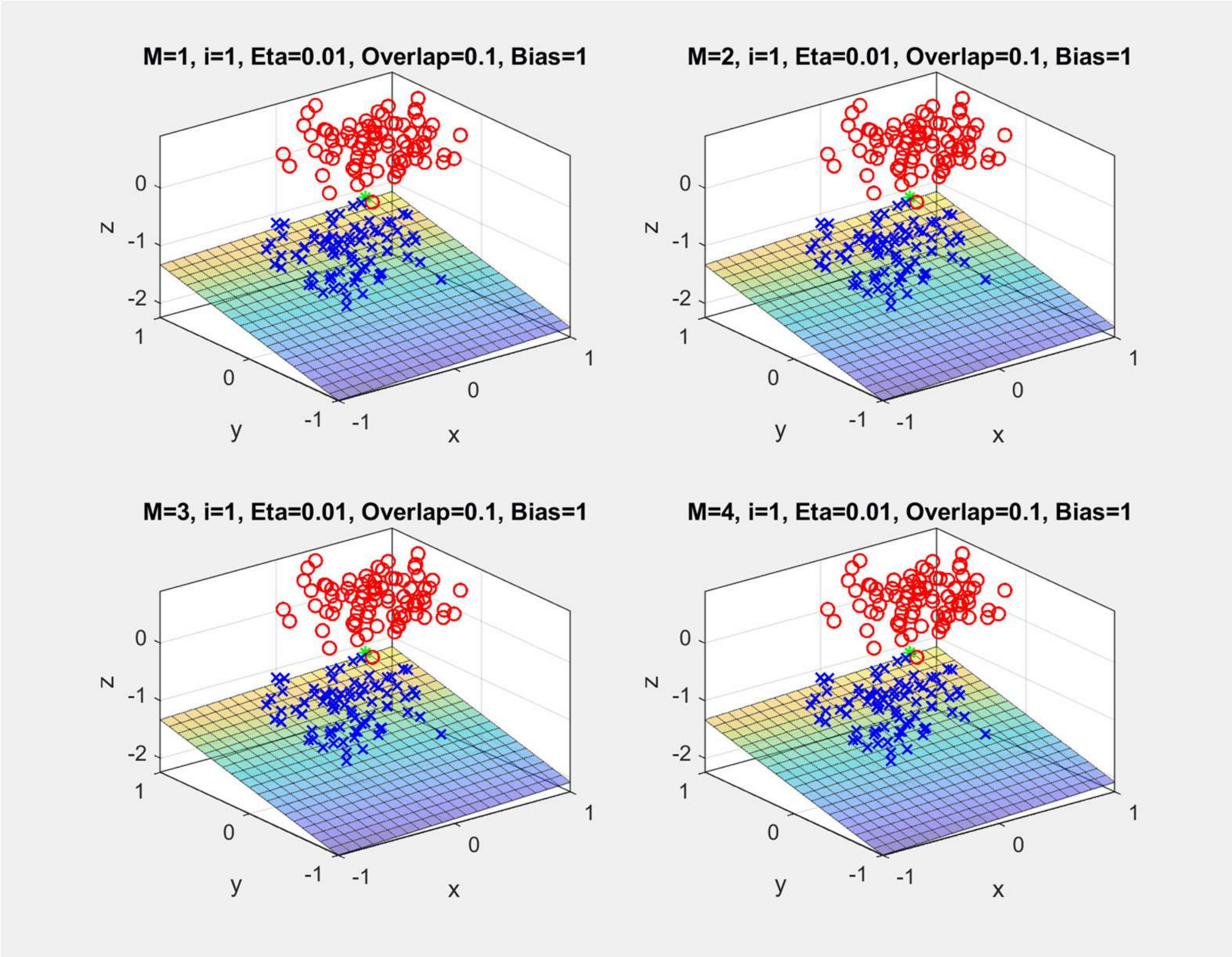
Mehmet Önder Efe, *Neural Networks*, Lecture Notes, 2022.

```
% Method 2

U = [0 0;0 1;1 0;1 1];
Y = [0 0 0 1]';

Phi = 2*rand(3,1)-1;
Eta = 0.8;
PHI(1,:)=Phi';

for count=1:1000
  epoche_error(count) = 0;
  for sample=1:4
    inputvector=[U(sample,:)';-1];
    Yn(sample) = 1/(1+exp(-(Phi'*inputvector)));
    errorvector = Y(sample)-Yn(sample);
    Phi=Phi+Eta*errorvector*Yn(sample)*(1-Yn(sample))*inputvector;
    PHI((count-1)*4+sample+1,:)=Phi';
    epoche_error(count) = epoche_error(count) + errorvector'*errorvector;
  end
  epoche_error(count)
end
[U,Y,Yn']
```

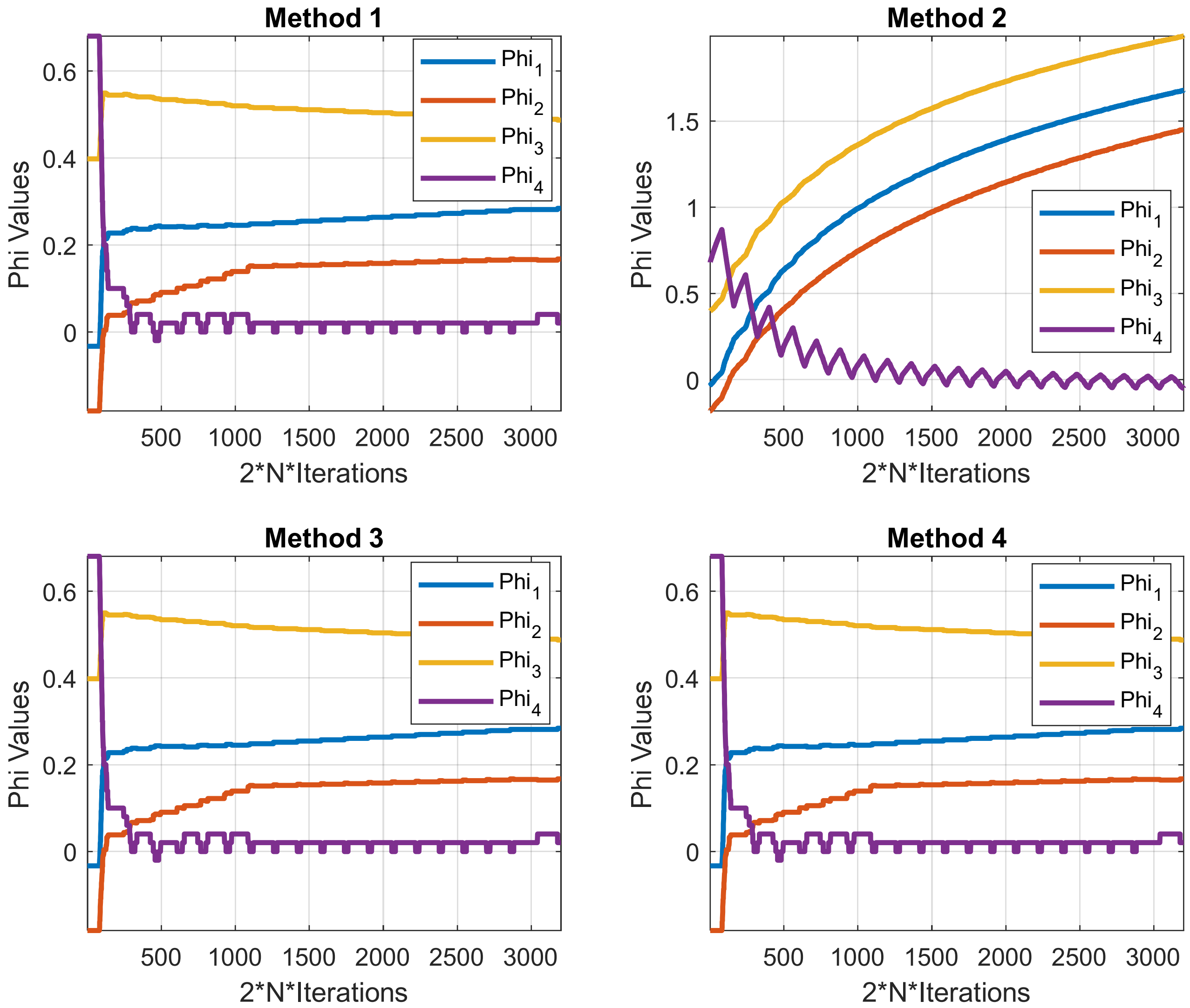
```
» [U,Y,Yn']

ans =

   0    0    0    0.0004
   0    1    0    0.0625
   1    0    0    0.0626
   1    1    1    0.9255
```

We saw how this works!

With bias term, little overlap, 80 samples/class

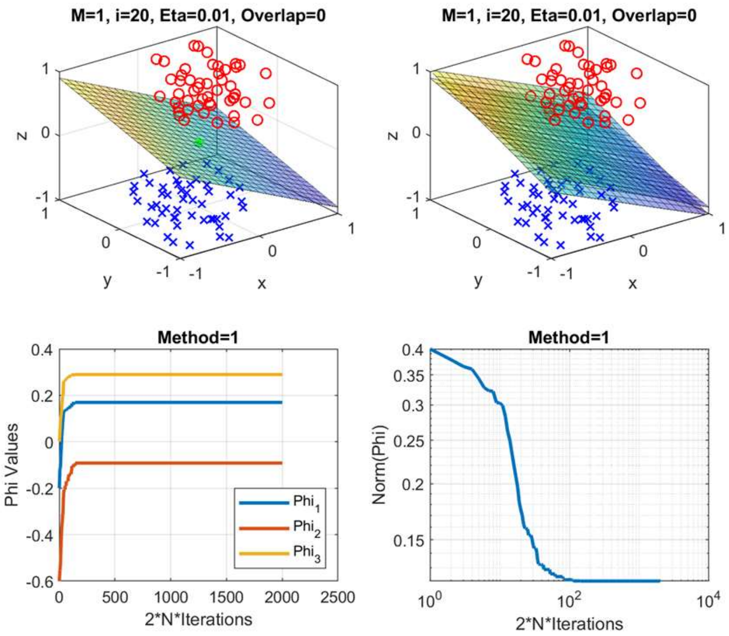


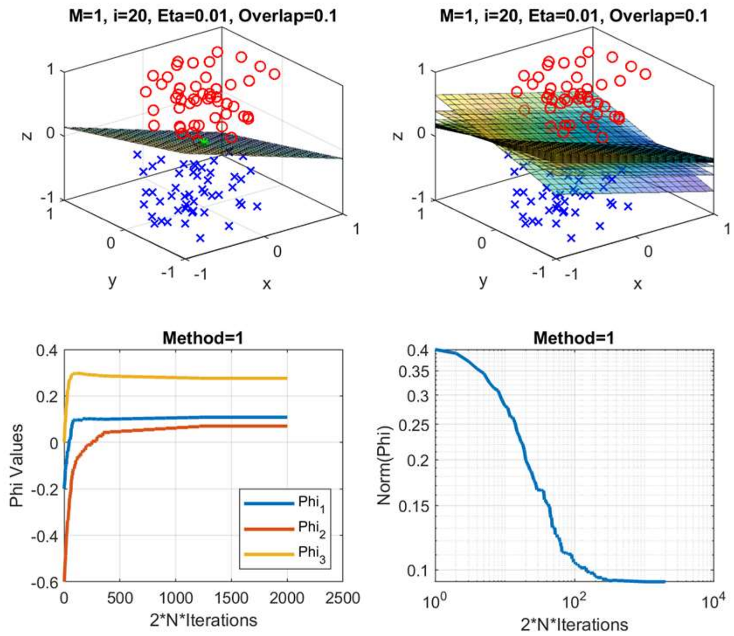
With bias term, little overlap, 80 samples/class



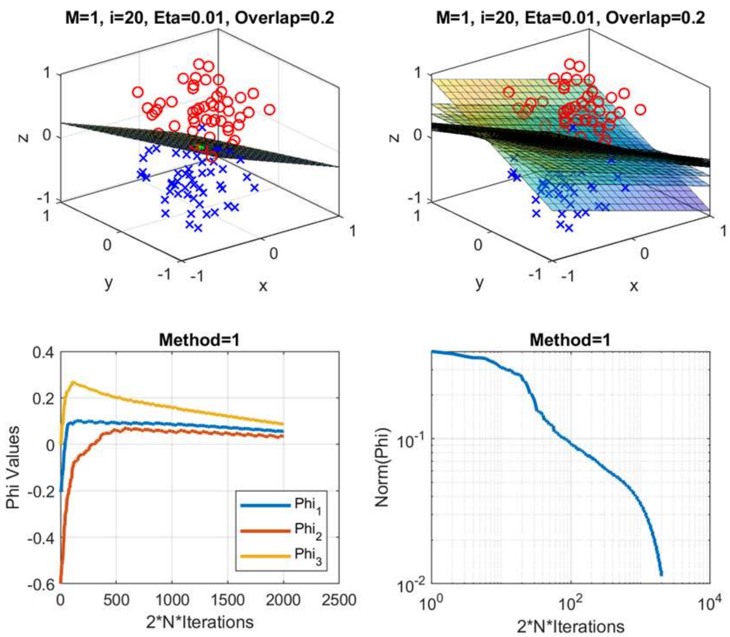
Mehmet Önder Efe, *Neural Networks*, Lecture Notes, 2022.

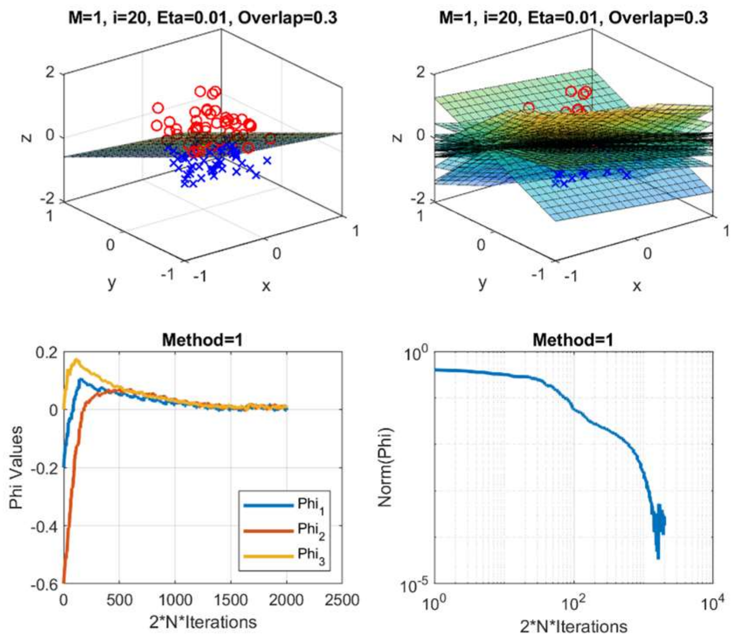
Let us check the 4 algorithms for four different overlap levels.

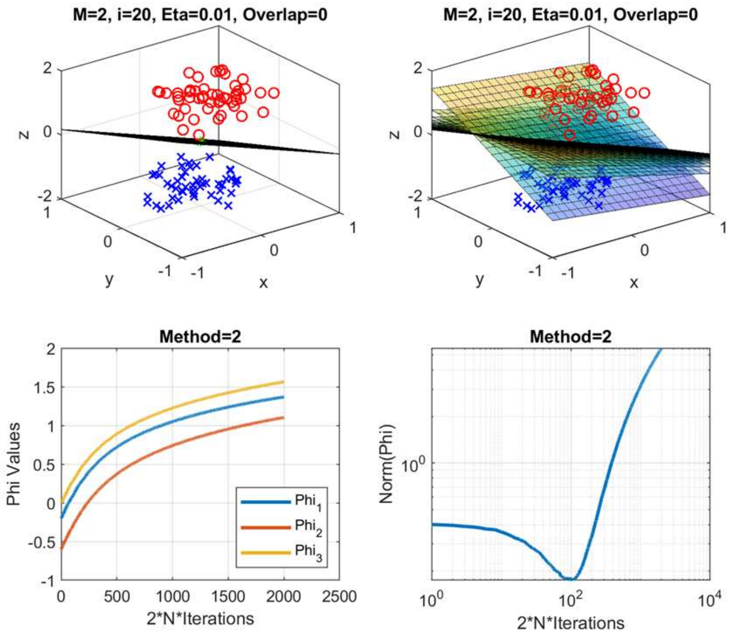


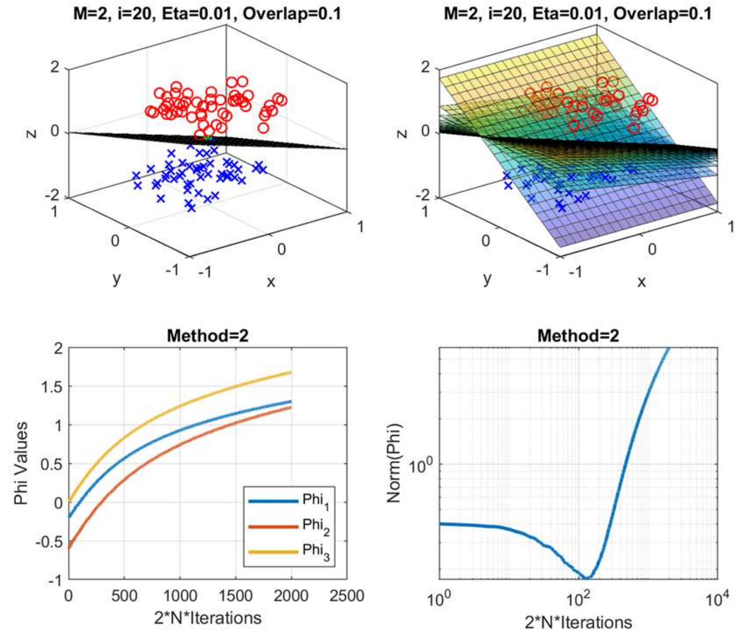


First Method

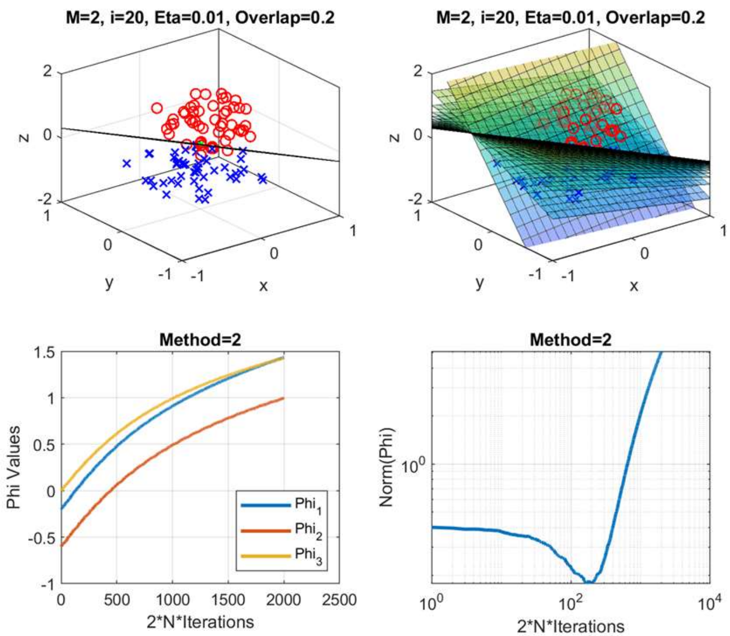


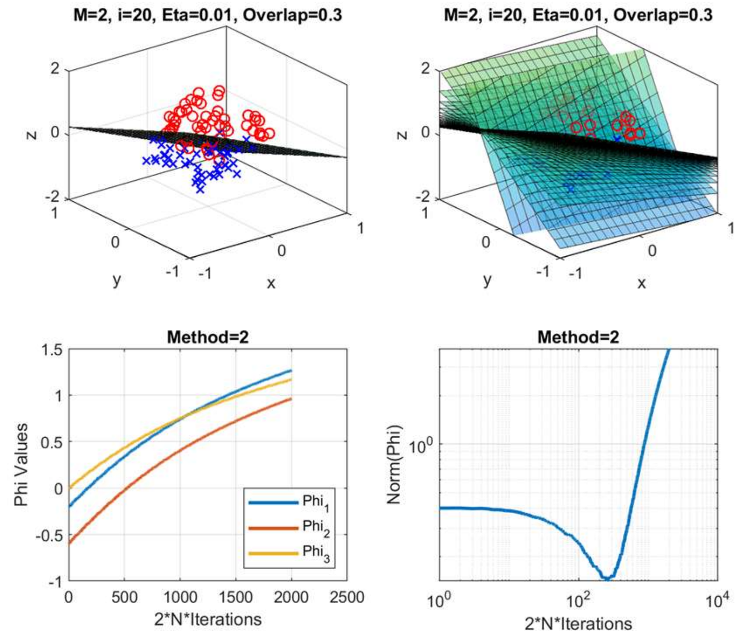


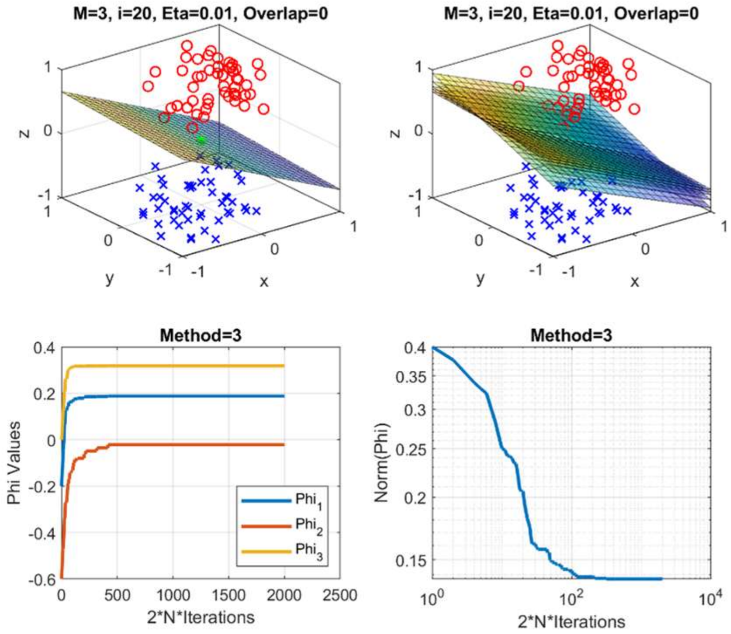


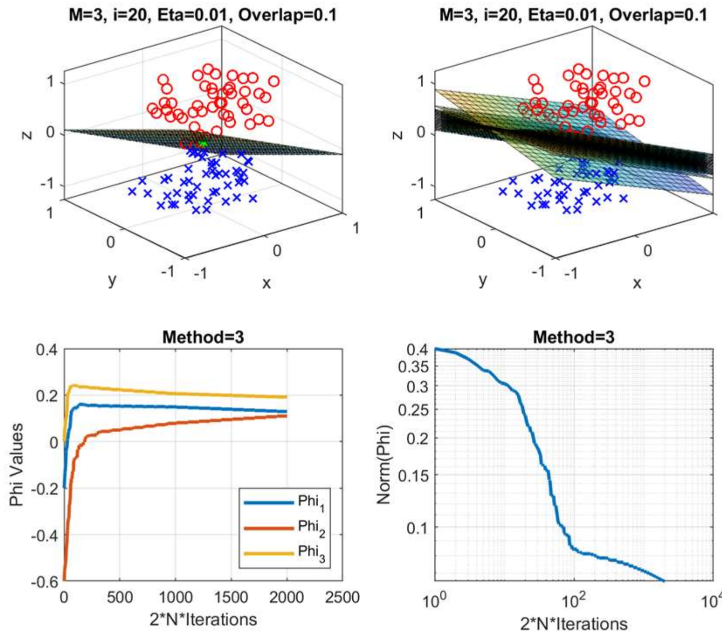


Second Method

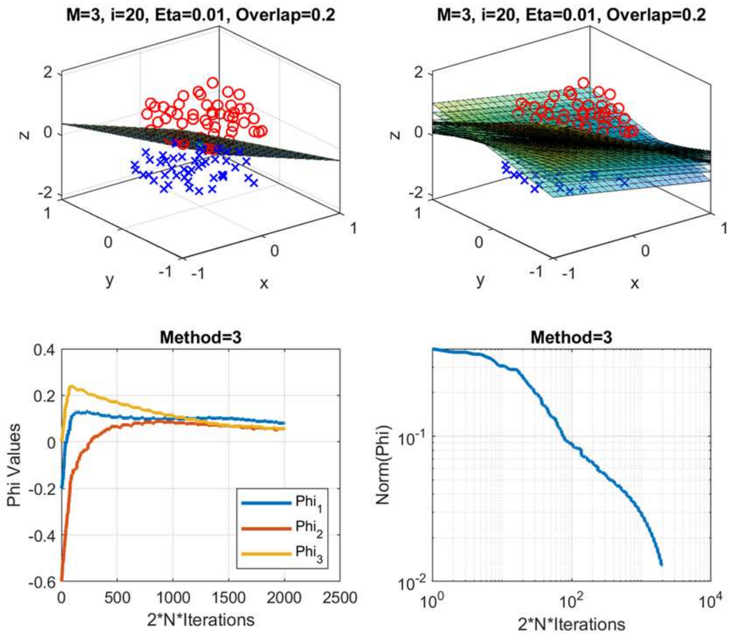


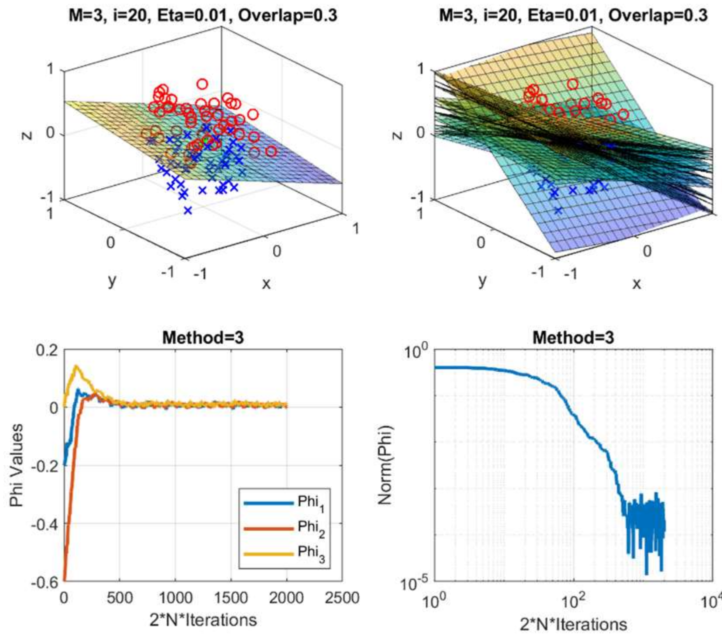


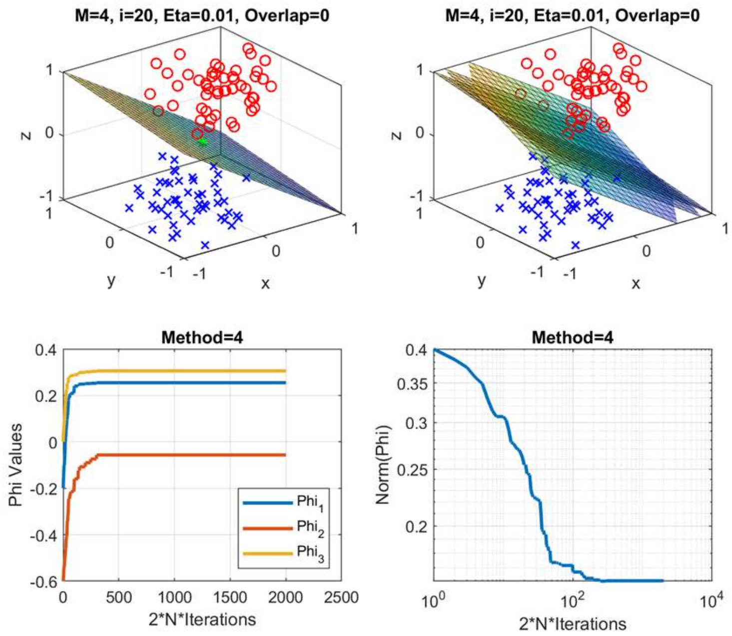


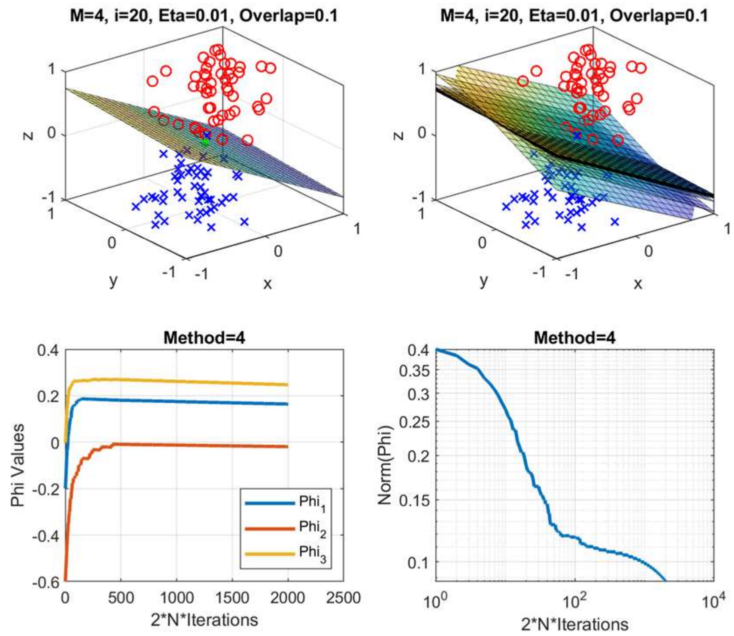


Third Method

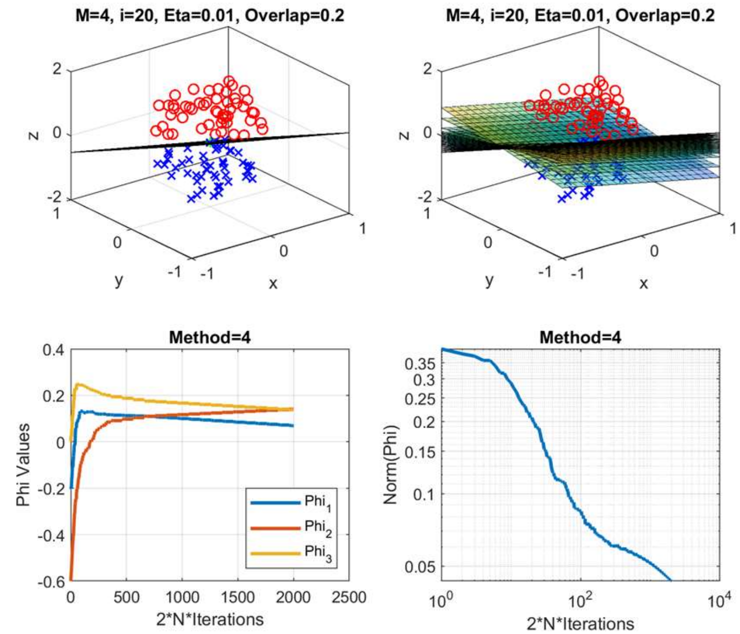


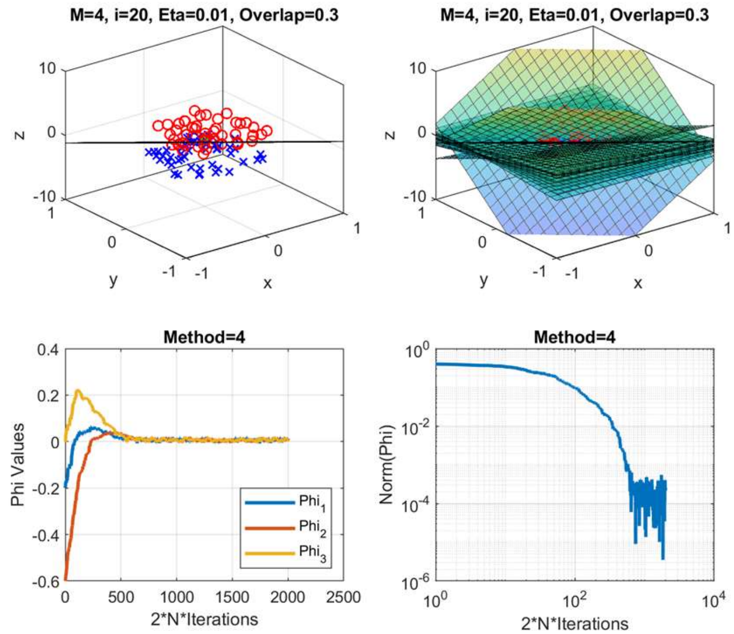






Fourth Method

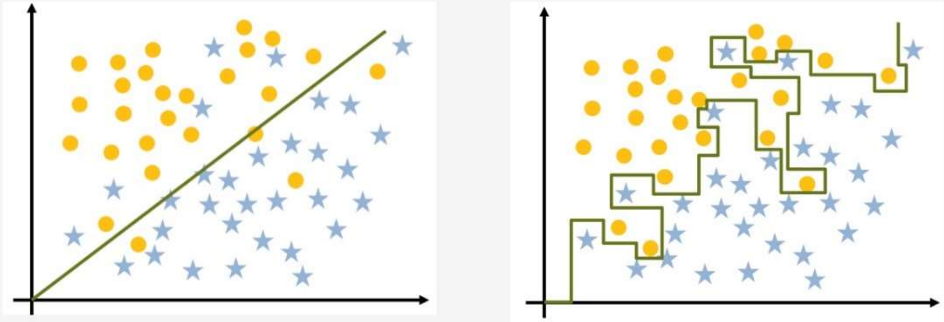




Mehmet Önder Efe, *Neural Networks*, Lecture Notes, 2022.

Simple Model
84% Accuracy

Complex Model
100% Accuracy



You can see that the complex model adapts better to the training data with a performance of a 100% vs. 84% for the simple model. It would be tempting to declare the complex model the winner. However, let's see the results if I apply the testing dataset (new data that was not used during training) to these models:

Simple Model
70% Accuracy

Complex Model
60% Accuracy

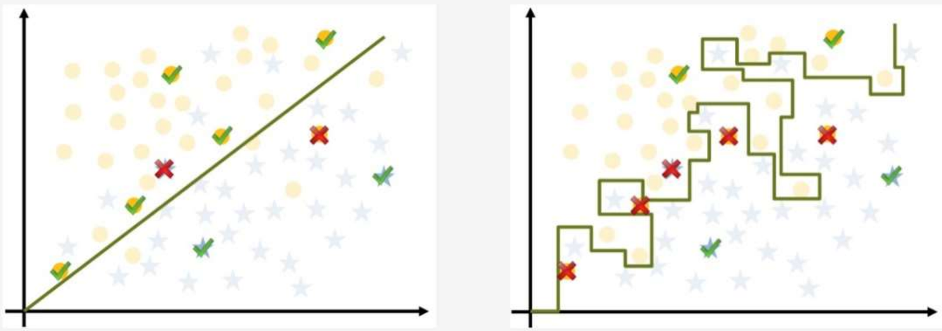
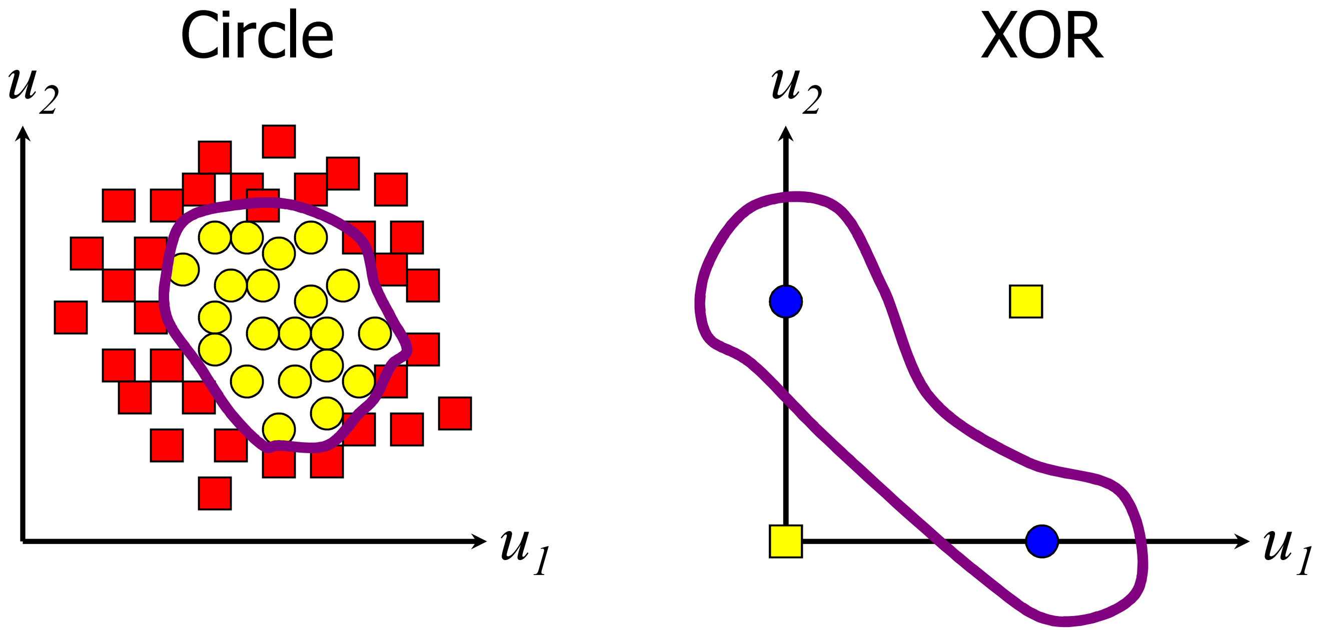


Image taken from internet

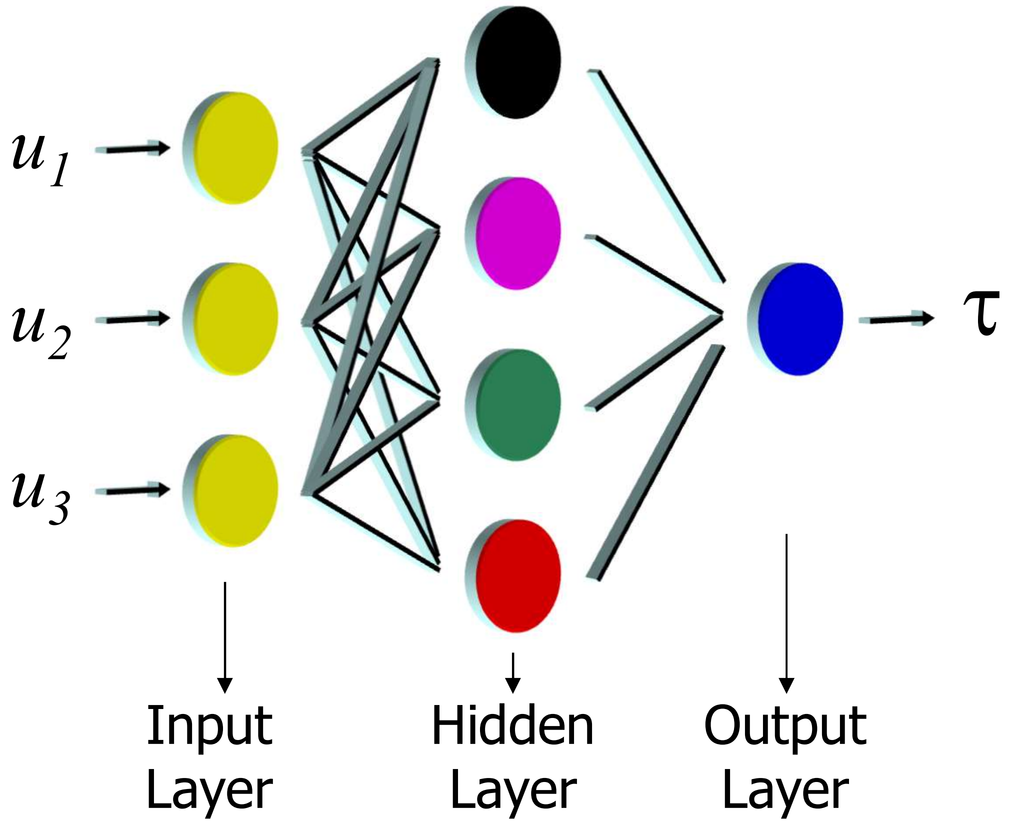
Perceptron Learning Algorithms

REMARKS



Given the data shown, can a single perceptron draw the decision boundary between two clusters?

Multilayer Perceptron (MLP) and Error Backpropagation (EBP)



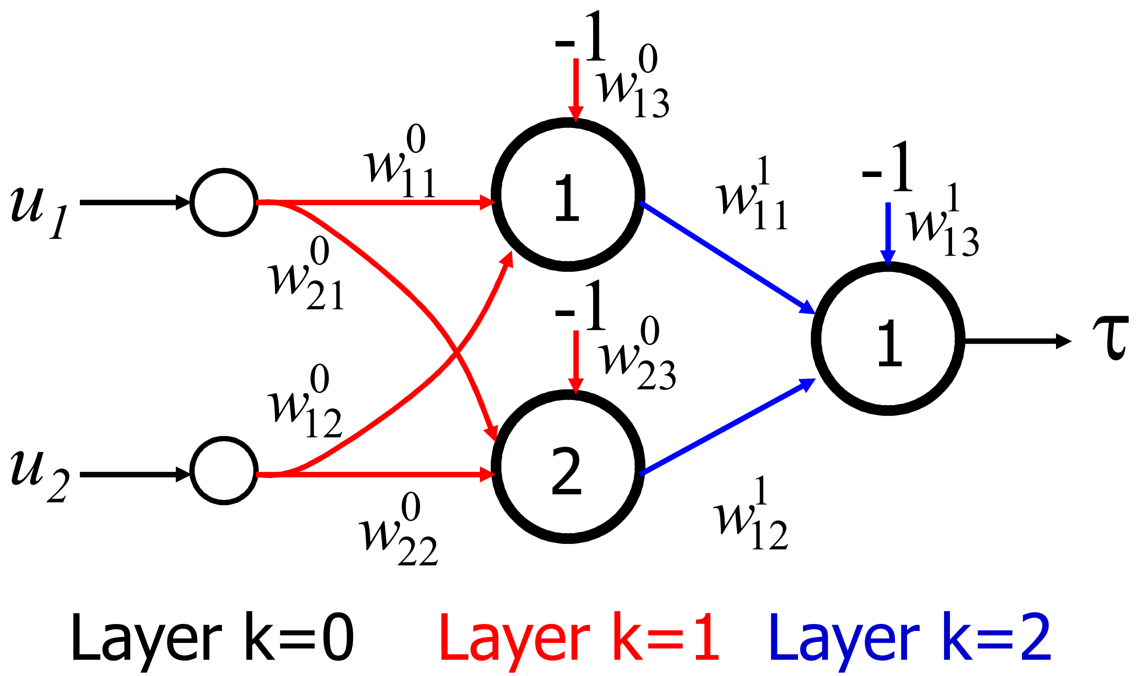
3-4-1 configuration

MLP and EBP

- A Pair is composed of a particular input vector and the corresponding desired output vector.
- Training data set is composed of some number of pairs
- Choosing one pair, applying the input part of it to a neural network and obtaining the network output vector is called a forward pass
- Calculating the error and adjusting the parameters is called a backward pass
- Sample error is defined as the square of the norm of the output error d-τ
- An epoche is completed when all pairs are passed through the network and the relevant parameter update is made.
- Epoche error is the sum of the sample errors for every pair in the training data set
- Mean Squared Error is epoche error over #of pairs.

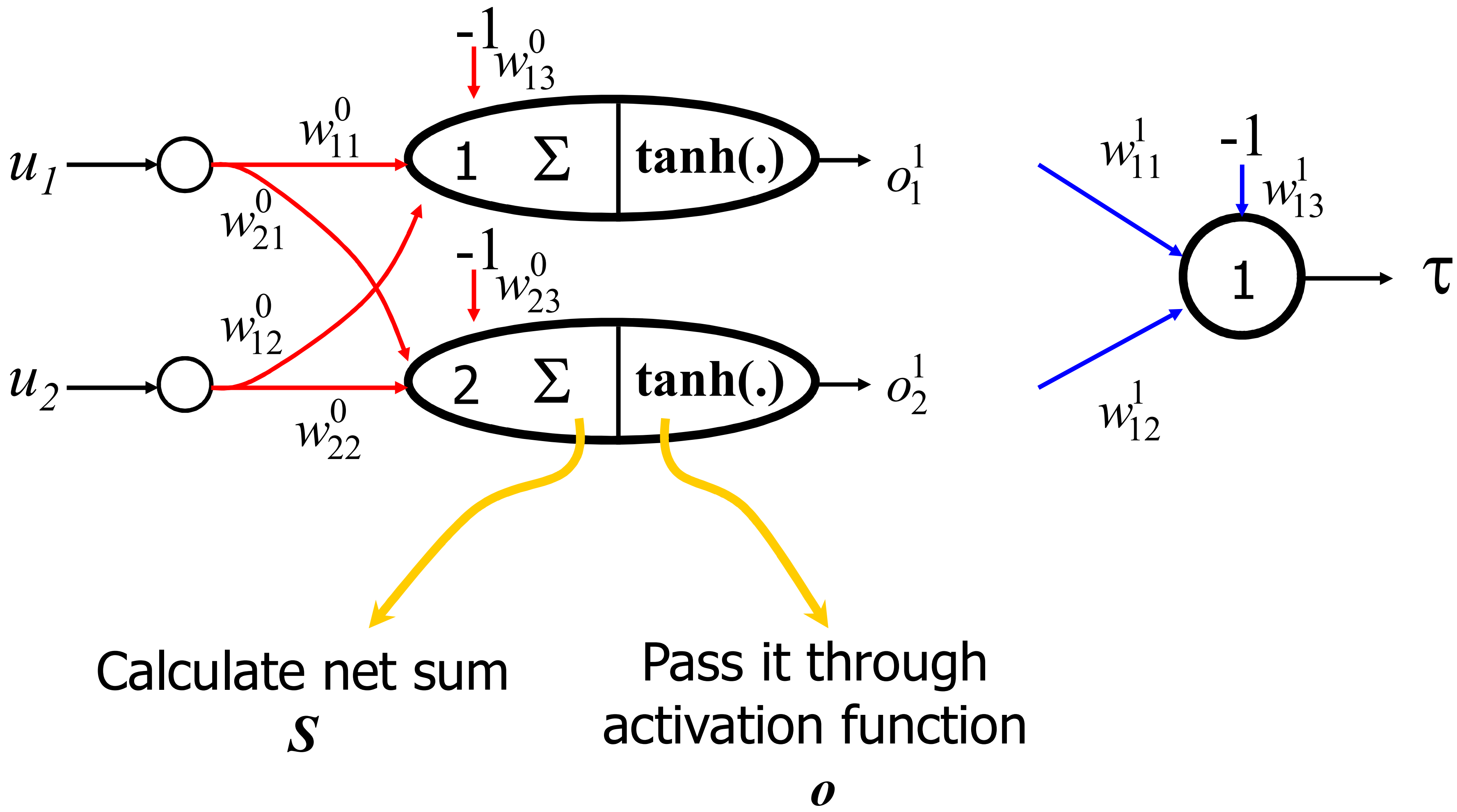
MLP and EBP

- Note that EBP is based on Gradient Descent
- We will start with a simple example then we will generalize the approach
- The problem is XOR, Configuration is 2-2-1 and activation functions for the hidden layer are tanh(.) and for the output layer it is linear.



MLP and EBP

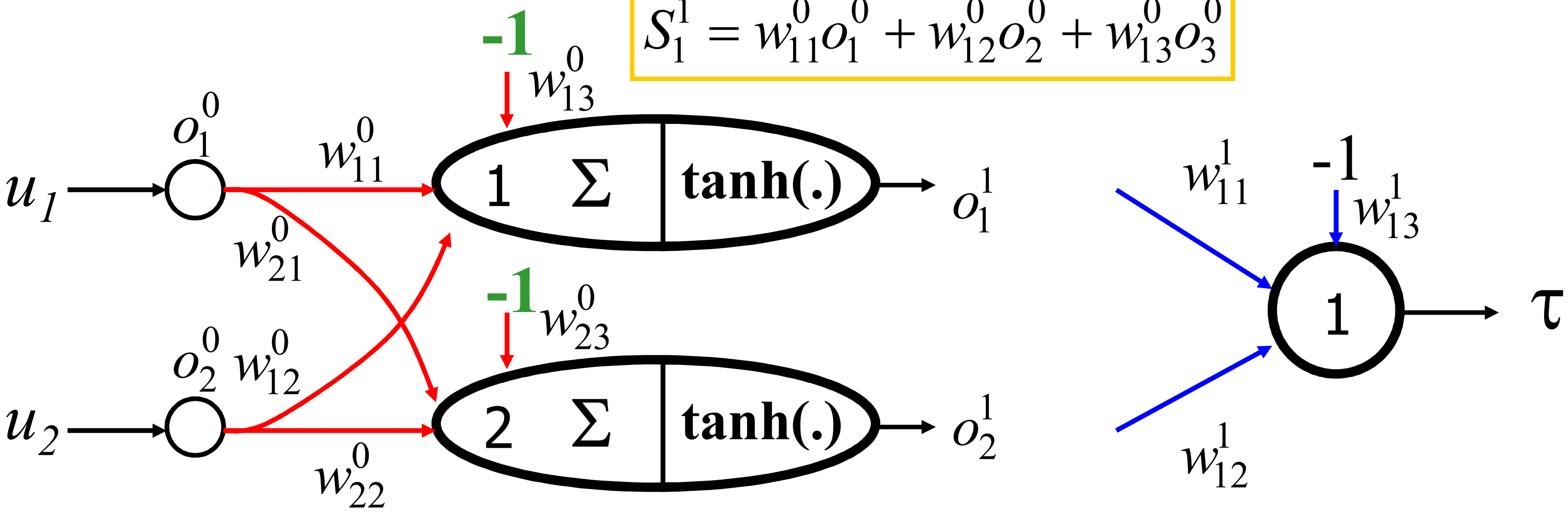
Forward Pass



Mehmet Önder Efe, *Neural Networks*, Lecture Notes, 2022.

MLP and EBP

Forward Pass



$$S_1^1 = w_{11}^0 o_1^0 + w_{12}^0 o_2^0 + w_{13}^0 o_3^0$$

$$o_3^0 = -1$$

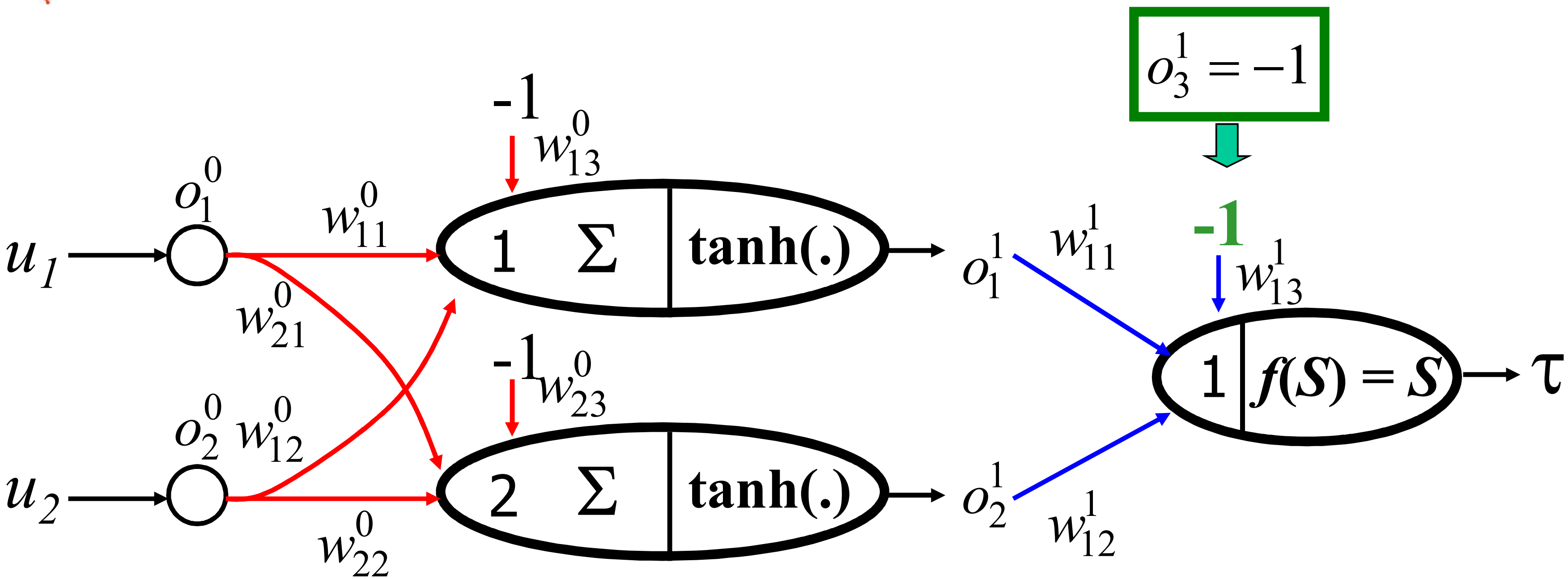
$$S_2^1 = w_{21}^0 o_1^0 + w_{22}^0 o_2^0 + w_{23}^0 o_3^0$$

$$S_i^1 = \sum_{j=1}^{3} w_{ij}^0 o_j^0$$

$$o_i^1 = \tanh(S_i^1)$$

MLP and EBP

Forward Pass



$$S_1^2 = \sum_{j=1}^{3} w_{1j}^1 o_j^1$$

$$\tau = S_1^2$$

MLP and EBP
Forward Pass

$$S_1^1 = w_{11}^0 o_1^0 + w_{12}^0 o_2^0 + w_{13}^0 o_3^0 \quad \Rightarrow \quad o_1^1 = \tanh(S_1^1)$$

$$S_2^1 = w_{21}^0 o_1^0 + w_{22}^0 o_2^0 + w_{23}^0 o_3^0 \quad \Rightarrow \quad o_2^1 = \tanh(S_2^1)$$

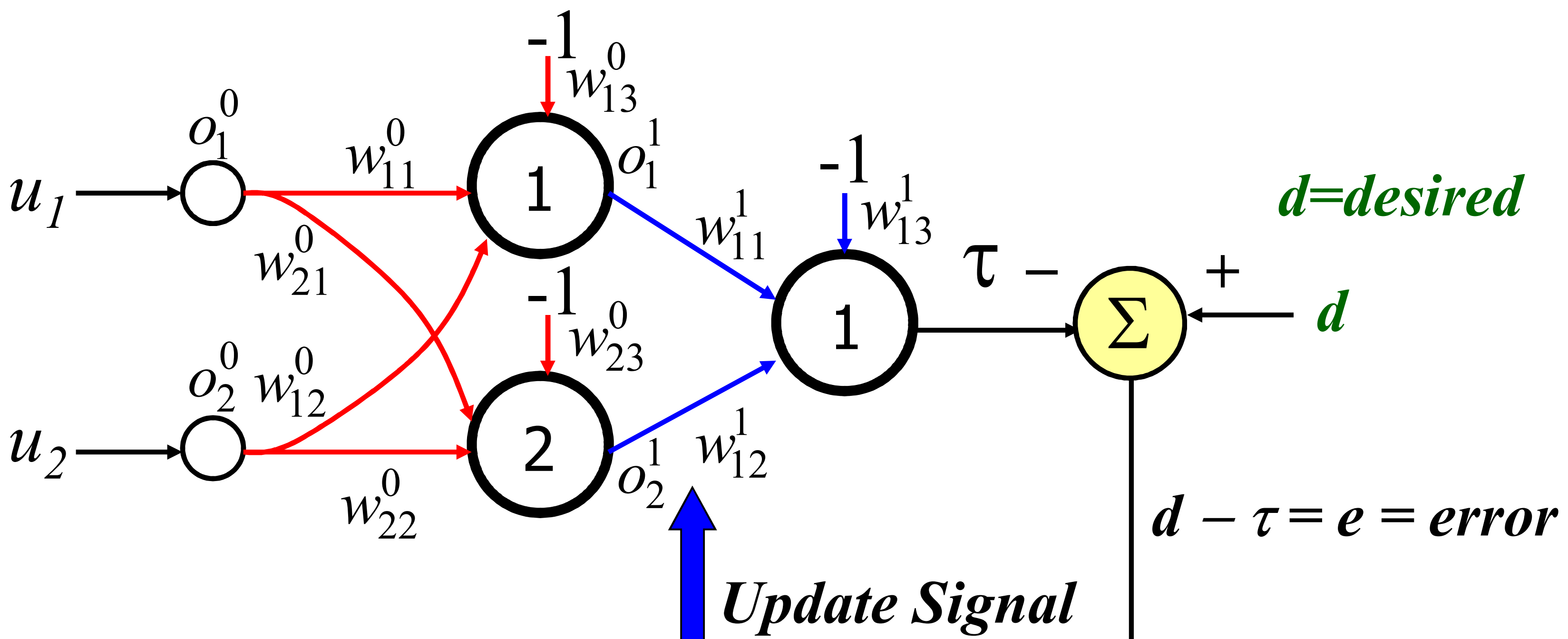
$$o_3^0 = -1$$

$$o_3^1 = -1$$

$$S_1^2 = \sum_{j=1}^{3} w_{1j}^1 o_j^1 \quad \Rightarrow \quad \tau = S_1^2$$

MLP and EBP

Backward Pass for the Output Layer



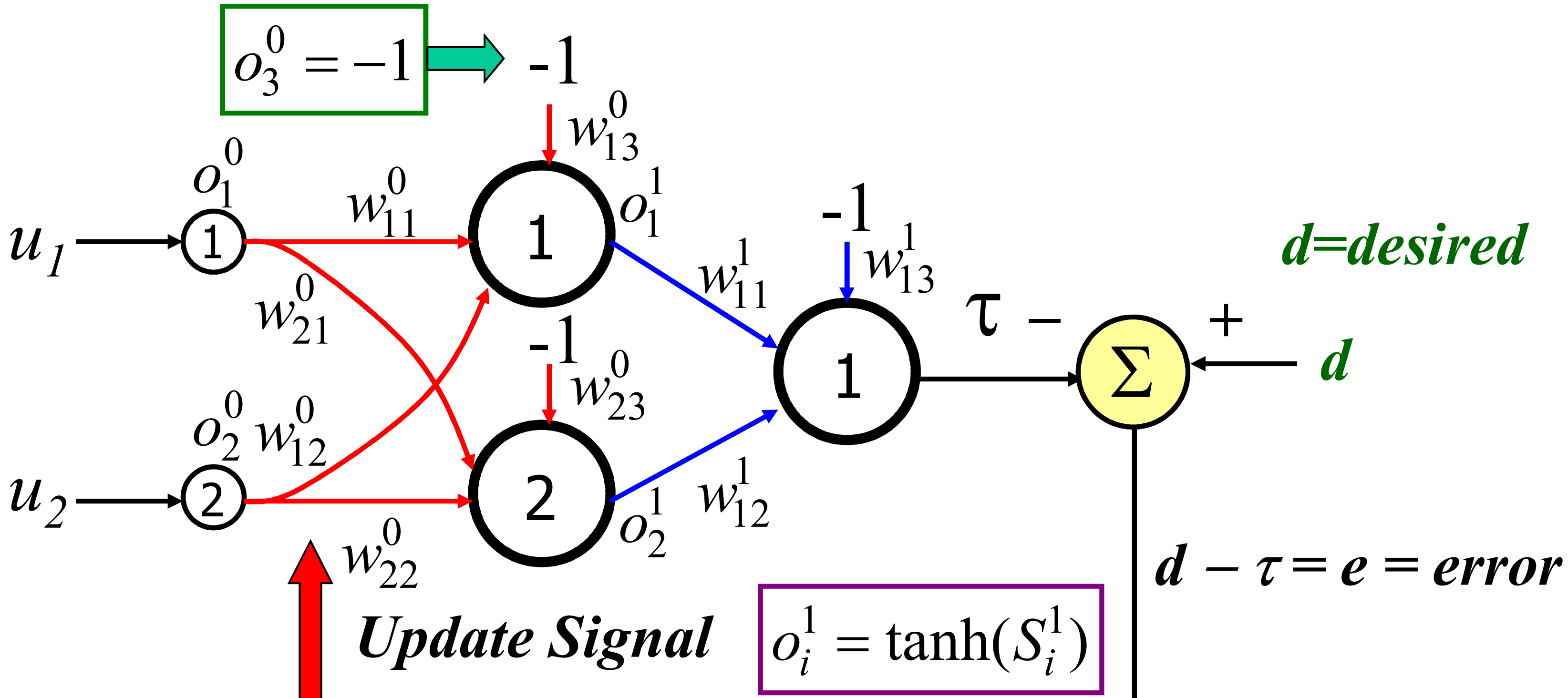
$$S_1^2=\sum_{j=1}^{3}w_{1j}^1 o_j^1$$

$$\tau=S_1^2$$

$$\Delta w_{1j}^1=\eta\, e\frac{\partial\tau}{\partial w_{1j}^1}=\eta\, e\frac{\partial\tau}{\partial S_1^2}\frac{\partial S_1^2}{\partial w_{1j}^1}=\eta\, e\, o_j^1$$

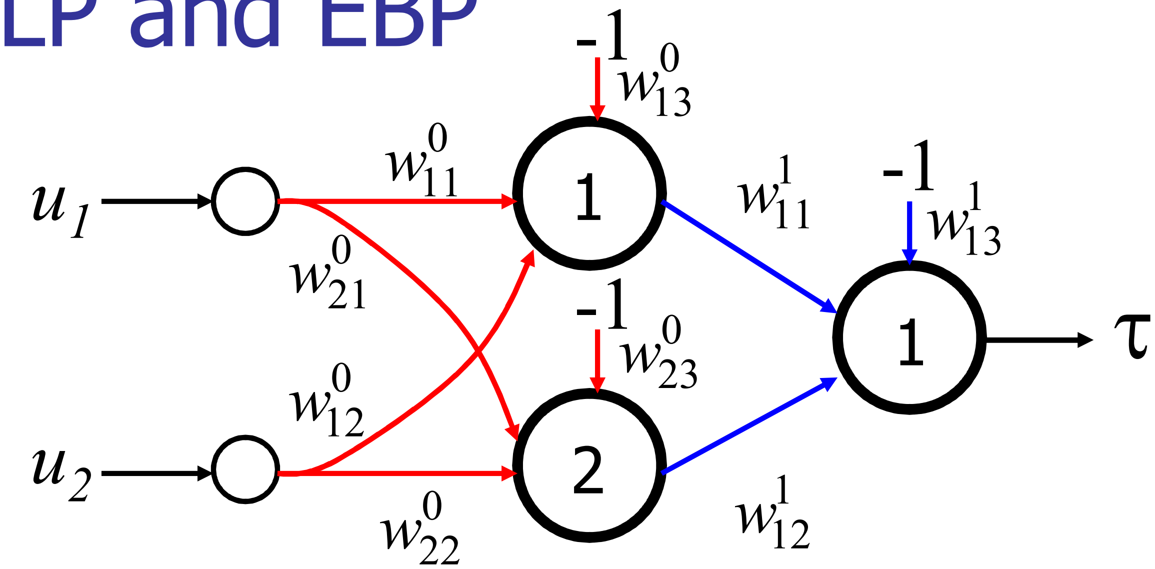
MLP and EBP

Backward Pass for the Hidden Layer



$$\Delta w_{ij}^0 = \eta\, e \frac{\partial \tau}{\partial w_{ij}^0} = \eta\, e \frac{\partial \tau}{\partial o_i^1} \frac{\partial o_i^1}{\partial S_i^1} \frac{\partial S_i^1}{\partial w_{ij}^0} = \eta\, e\, w_{1i}^1 \left(1 - \left(o_i^1\right)^2\right) o_j^0$$

MLP and EBP

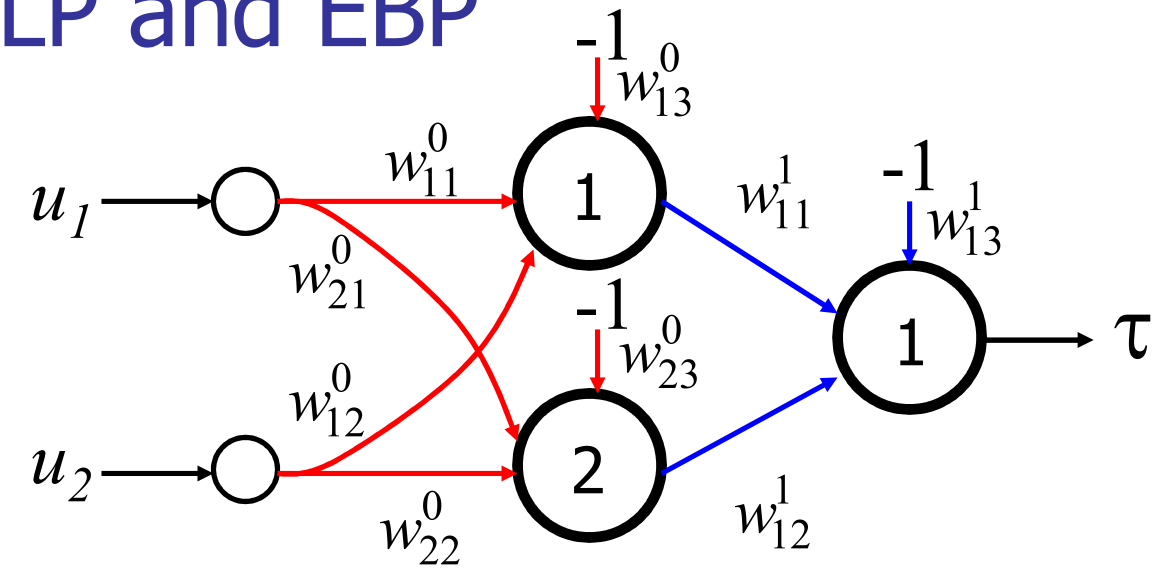


$$J(W_{Rij},W_{Lij},B_{Rij},B_{Lij}) = \frac{1}{2}e^2 = \frac{1}{2}(d-\tau)^2 = \frac{1}{2}\left(d - (\underbrace{W_R \overbrace{\tanh(\underbrace{W_L u - B_L}_{S})}^{o} - B_R}_{\tau}) \right)^2$$

$$\frac{\partial J}{\partial W_{Rij}} = -e\frac{\partial \tau}{\partial W_{Rij}}, \frac{\partial J}{\partial W_{Lij}} = -e\frac{\partial \tau}{\partial W_{Lij}}$$

$$\frac{\partial J}{\partial B_{Rij}} = -e\frac{\partial \tau}{\partial B_{Rij}}, \frac{\partial J}{\partial B_{Lij}} = -e\frac{\partial \tau}{\partial B_{Lij}}$$

MLP and EBP



$$J(W_{ij}^1, W_{ij}^0, B_{ij}^1, B_{ij}^0) = \frac{1}{2}e^2 = \frac{1}{2}(d-\tau)^2 = \frac{1}{2}\left(d - (\underbrace{W^1 \overbrace{\tanh(\underbrace{W^0 u - B^0}_{S})}^{o} - B^1}_{\tau}) \right)^2$$

$$\frac{\partial J}{\partial W_{ij}^1} = -e\frac{\partial \tau}{\partial W_{ij}^1}, \frac{\partial J}{\partial W_{ij}^0} = -e\frac{\partial \tau}{\partial W_{ij}^0}$$

$$\frac{\partial J}{\partial B_{ij}^1} = -e\frac{\partial \tau}{\partial B_{ij}^1}, \frac{\partial J}{\partial B_{ij}^0} = -e\frac{\partial \tau}{\partial B_{ij}^0}$$

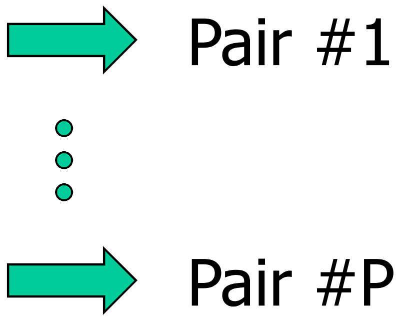
MLP and EBP

A Pseudo Code

```
Choose your network configuration
Initialize the weights to randomly chosen small numbers
Choose Learning Rate η
FOR counter=1 to 100
        Epoche_Error=0
        FOR p=1 to P
                Choose pair #p
                Forward Pass
                Calculate Sample_Error
                Epoche_Error += Sample_Error
                Backward Pass
        END
        Sum Squared Error [count] = Epoche_Error
        Print Epoche_Error
END
Save your network data
```

MLP and EBP

u_1	u_2	y
0	0	0
0	1	0
1	0	0
1	1	1

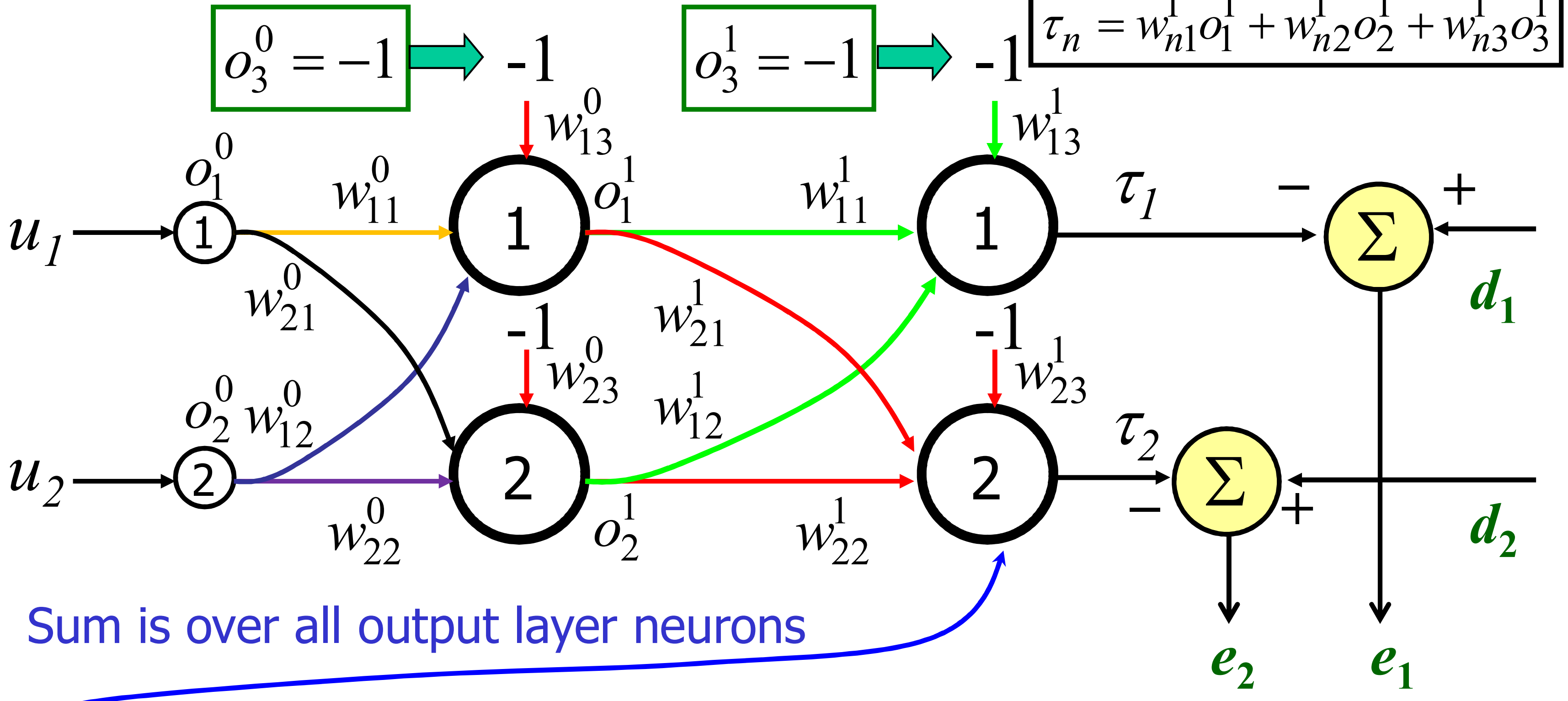


- For XOR problem you will have 4 inputs, i.e. there are finite number of input combinations
- There will be one output
- You may choose the number of hidden layers and neurons in them

MLP and EBP

Let's try this one for backward pass computations

$$\tau_n = w_{n1}^1 o_1^1 + w_{n2}^1 o_2^1 + w_{n3}^1 o_3^1$$



Sum is over all output layer neurons

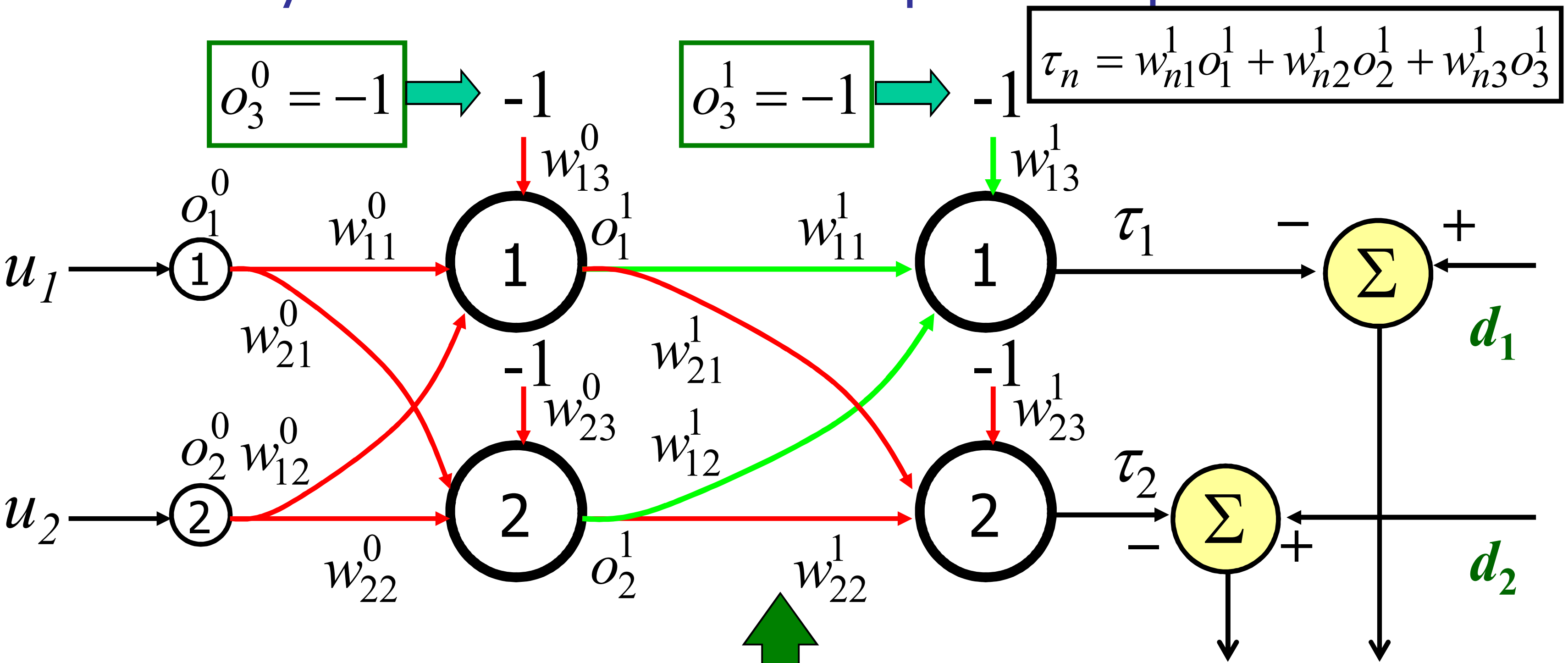
$$J = \frac{1}{2}\sum_{n=1}^{2}\left(d_n - \tau_n\right)^2 = \frac{1}{2}\sum_{n=1}^{2} e_n^2$$

$$\Delta w = -\eta\frac{\partial J}{\partial w} = -\eta\sum_{n=1}^{2} e_n \frac{\partial e_n}{\partial w} = \eta\sum_{n=1}^{2} e_n \frac{\partial \tau_n}{\partial w}$$

A generic weight/bias of the NN

MLP and EBP

Let's try this one for backward pass computations

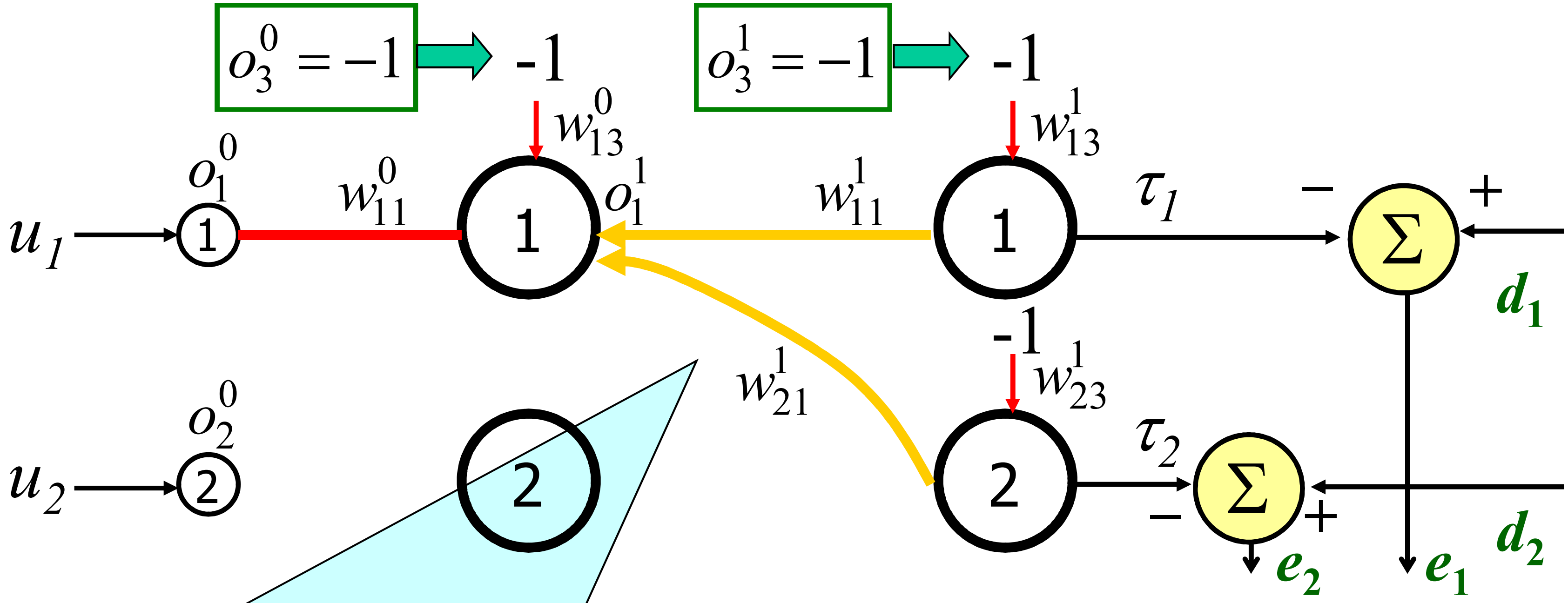


Try every i and j, see which term drops out in each case

$$\Delta w_{ij}^{1} = -\eta \frac{\partial J}{\partial w_{ij}^{1}} = -\eta \sum_{n=1}^{2} e_n \frac{\partial e_n}{\partial w_{ij}^{1}} = \eta \left(e_1 \frac{\partial \tau_1}{\partial w_{ij}^{1}} + e_2 \frac{\partial \tau_2}{\partial w_{ij}^{1}} \right) = \eta\, e_i\, o_j^{1}$$

MLP and EBP

For the parameters with superscript 0

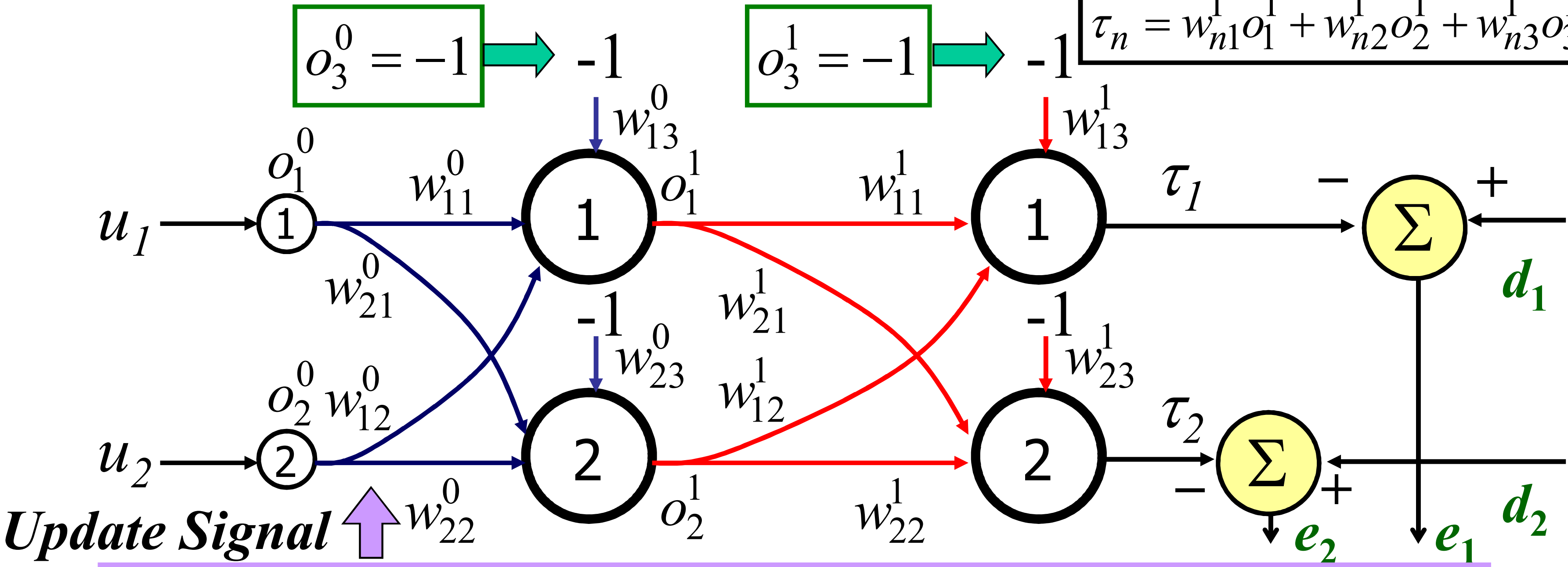


There are multiple paths to every weight in the first layer
You have to compute the contribution of every one of them.
Make use of the layered structure to generalize this…
The Error Backpropagation!

MLP and EBP

Let's try this one for backward pass computations

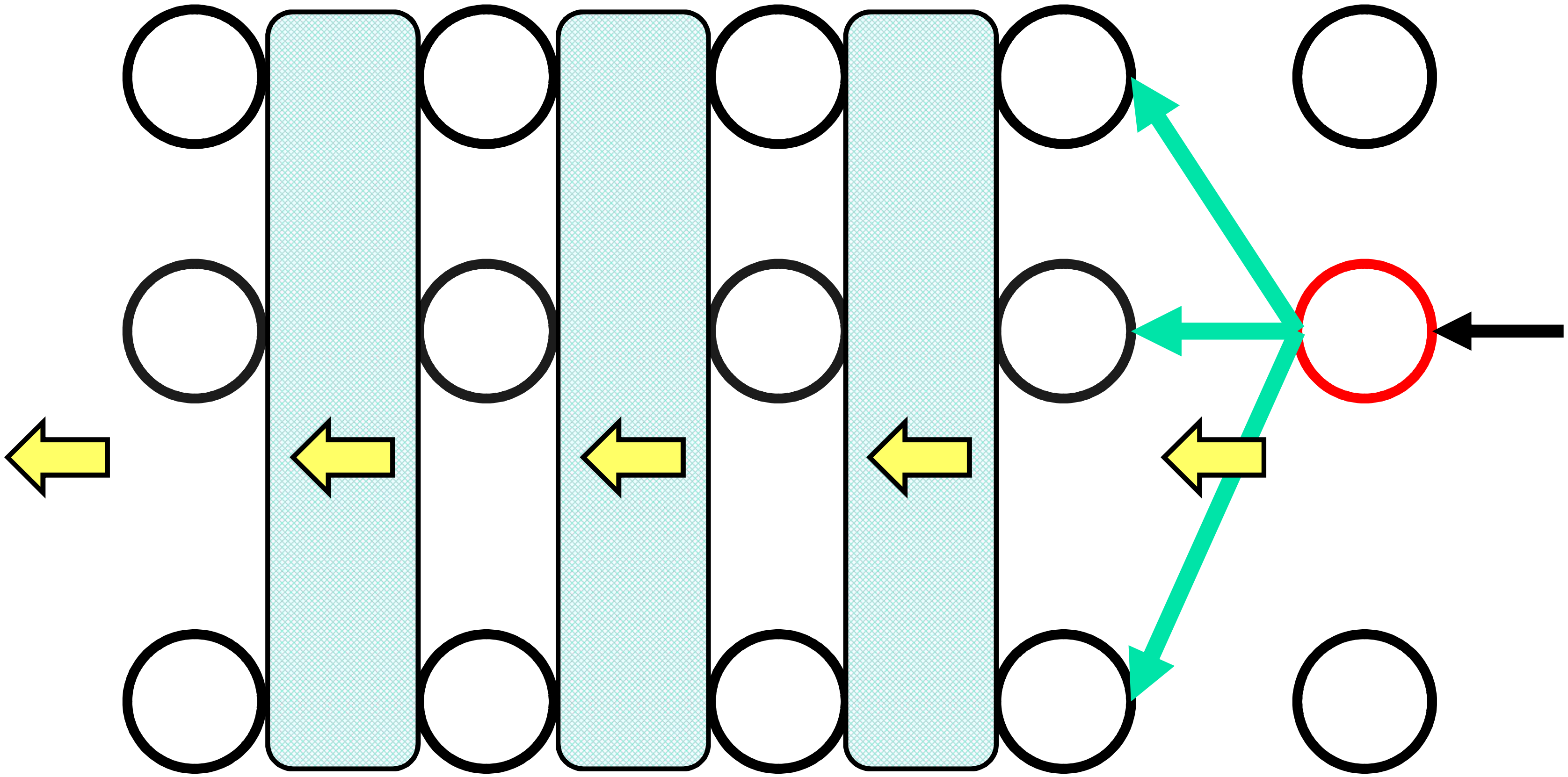
$$\tau_n = w_{n1}^1 o_1^1 + w_{n2}^1 o_2^1 + w_{n3}^1 o_3^1$$



$$\Delta w_{ij}^0 = -\eta \frac{\partial J}{\partial w_{ij}^0} = -\eta \sum_{n=1}^{2} e_n \frac{\partial e_n}{\partial w_{ij}^0} = \eta \sum_{n=1}^{2} e_n \frac{\partial \tau_n}{\partial w_{ij}^0} = \eta \sum_{n=1}^{2} e_n \frac{\partial \tau_n}{\partial o_i^1} \frac{\partial o_i^1}{\partial w_{ij}^0}$$

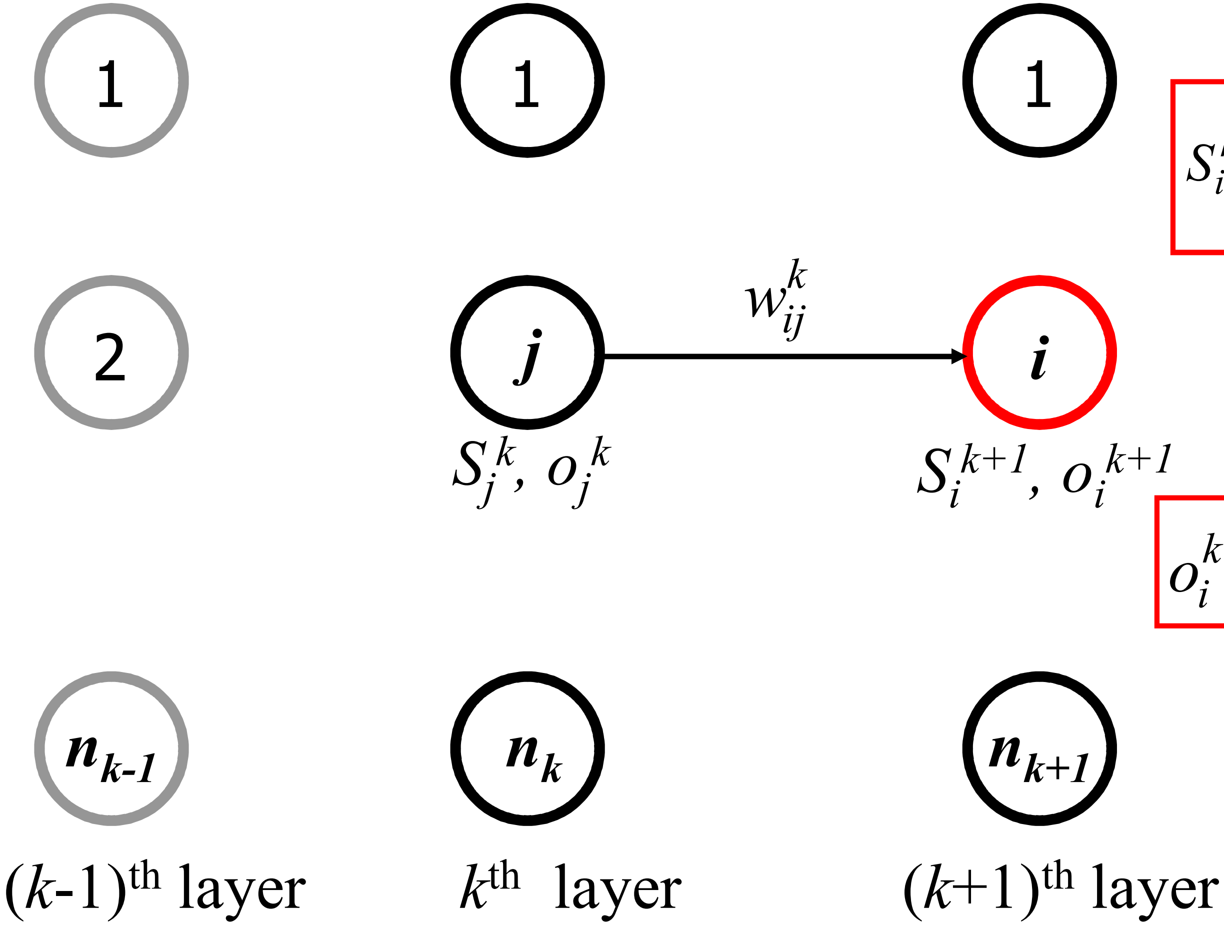
$$= \eta \sum_{n=1}^{2} e_n w_{ni}^1 \frac{\partial o_i^1}{\partial w_{ij}^0} = \eta \sum_{n=1}^{2} e_n w_{ni}^1 \frac{\partial o_i^1}{\partial S_i^1} \frac{\partial S_i^1}{\partial w_{ij}^0} = \eta \sum_{n=1}^{2} e_n w_{ni}^1 \left(1 - \left(o_i^1\right)^2\right) o_j^0$$

MLP and EBP



MLP and EBP

Generalization of the Tuning Law



$$S_i^{k+1} = \sum_{j=1}^{n_k} w_{ij}^k o_j^k$$

$$o_i^{k+1} = f\left(S_i^{k+1}\right)$$

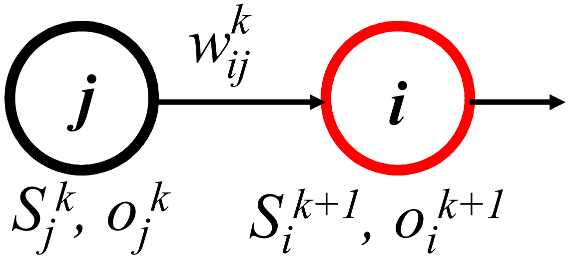
MLP and EBP

Generalization of the Tuning Law - Output Layer

- Assume (k+1)th layer is the output layer
- The output layer neurons have activation functions denoted by f(x)=x
- The cost function is

$$S_i^{k+1}=\sum_{j=1}^{n_k} w_{ij}^k o_j^k$$

$$J=\frac{1}{2}\sum_{i=1}^{n_{k+1}}\left(d_i-o_i^{k+1}\right)^2=\frac{1}{2}\sum_{i=1}^{n_{k+1}} e_i^2$$



- The update law is

$$\Delta w_{ij}^k := -\eta\frac{\partial J}{\partial w_{ij}^k}=-\eta\frac{\partial J}{\partial o_i^{k+1}}\frac{\partial o_i^{k+1}}{\partial S_i^{k+1}}\frac{\partial S_i^{k+1}}{\partial w_{ij}^k}$$

$$\Delta w_{ij}^k=(-\eta)\left(-\left(d_i-\tau_i\right)\right)f'\left(S_i^{k+1}\right)\left(o_j^k\right)$$

$$o_i^{k+1}=f\left(S_i^{k+1}\right)$$

$$o_i^{k+1}=\tau_i$$

MLP and EBP

Generalization of the Tuning Law - Output Layer

$$J = \frac{1}{2}\sum_{i=1}^{n_{k+1}}\left(d_i - \tau_i\right)^2 = \frac{1}{2}\sum_{i=1}^{n_{k+1}} e_i^2$$

$$\Delta w = -\eta \nabla_w J$$

$$\frac{\partial J}{\partial w_{ij}^k} = \frac{\partial J}{\partial o_i^{k+1}} \frac{\partial o_i^{k+1}}{\partial S_i^{k+1}} \frac{\partial S_i^{k+1}}{\partial w_{ij}^k}$$

$$\frac{\partial J}{\partial o_i^{k+1}} = -\left(d_i - o_i^{k+1}\right)$$

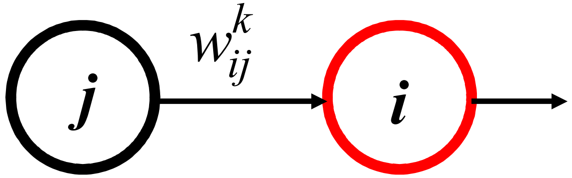
$$\frac{\partial o_i^{k+1}}{\partial S_i^{k+1}} = \frac{df\left(S_i^{k+1}\right)}{dS_i^{k+1}} = f'\left(S_i^{k+1}\right)$$

$$\frac{\partial S_i^{k+1}}{\partial w_{ij}^k} = \frac{\partial}{\partial w_{ij}^k}\left[\sum_{j=1}^{n_k} w_{ij}^k o_j^k\right] = o_j^k$$

$$\delta_i^{k+1} := -\frac{\partial J}{\partial S_i^{k+1}}$$

$$\delta_i^{k+1} = \left(d_i - o_i^{k+1}\right) f'\left(S_i^{k+1}\right)$$

$$\Delta w_{ij}^k = \eta \delta_i^{k+1} o_j^k$$

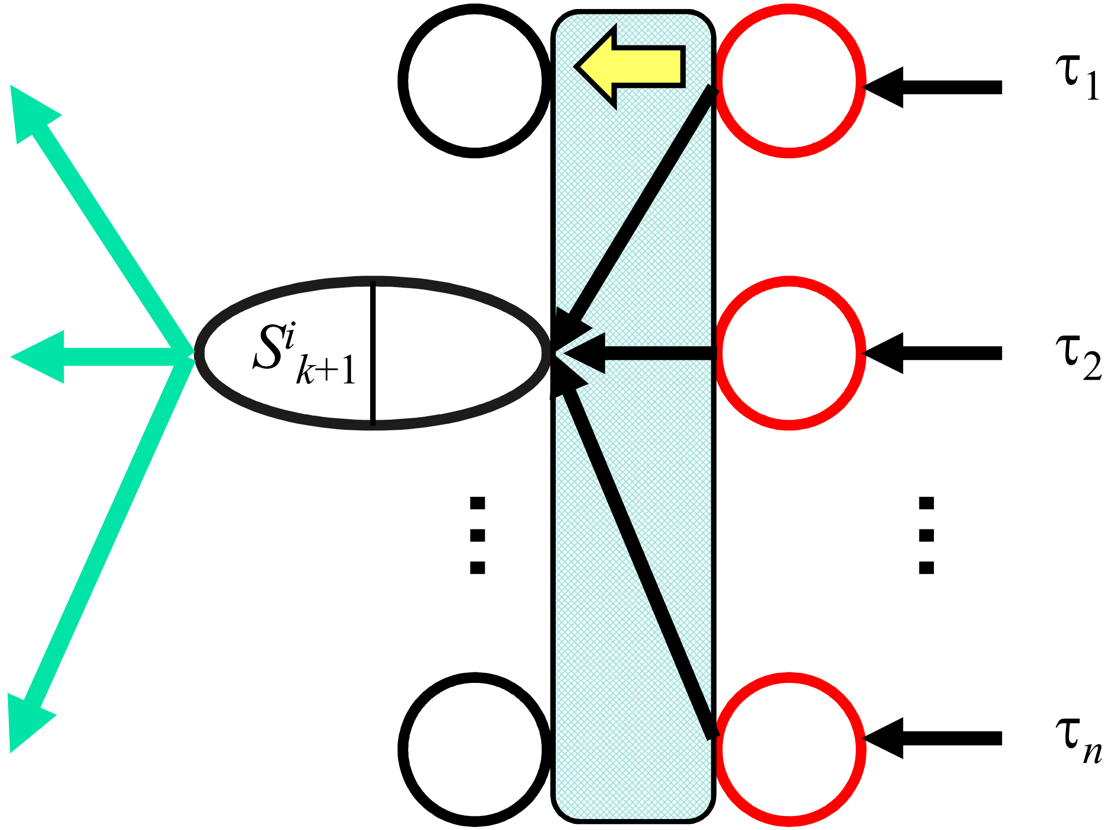


w_{ij}^k

$S_j^k,\ o_j^k$ $S_i^{k+1},\ o_i^{k+1}$

MLP and EBP

Generalization of the Tuning Law - Output Layer

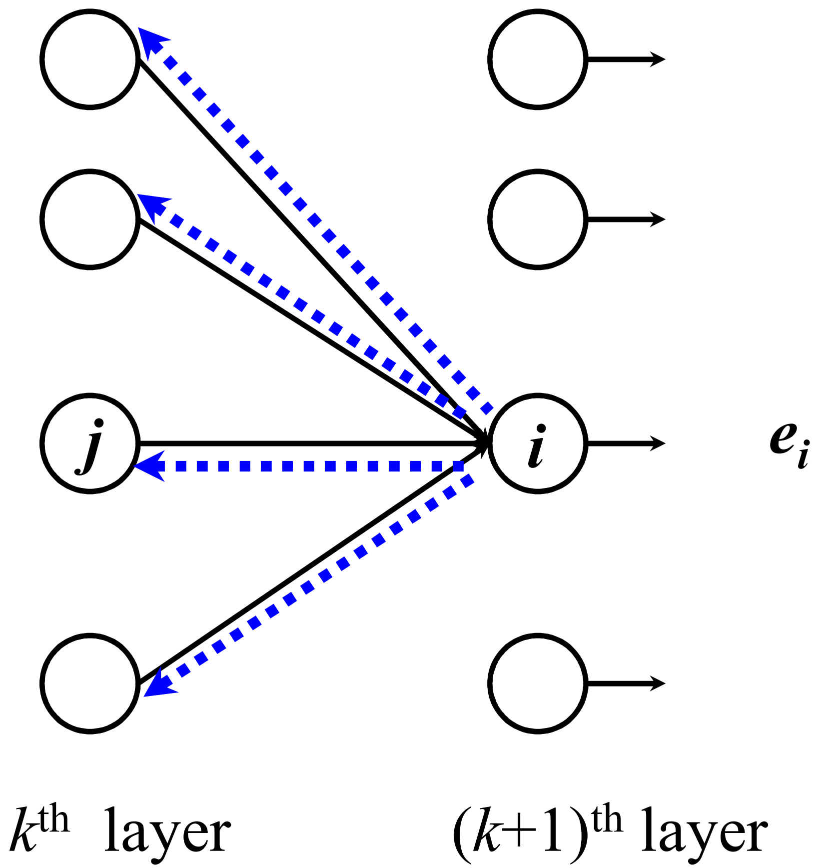


MLP and EBP

Generalization of the Tuning Law - Output Layer

$$\Delta w_{ij}^{k} = \eta \delta_i^{k+1} o_j^{k}$$

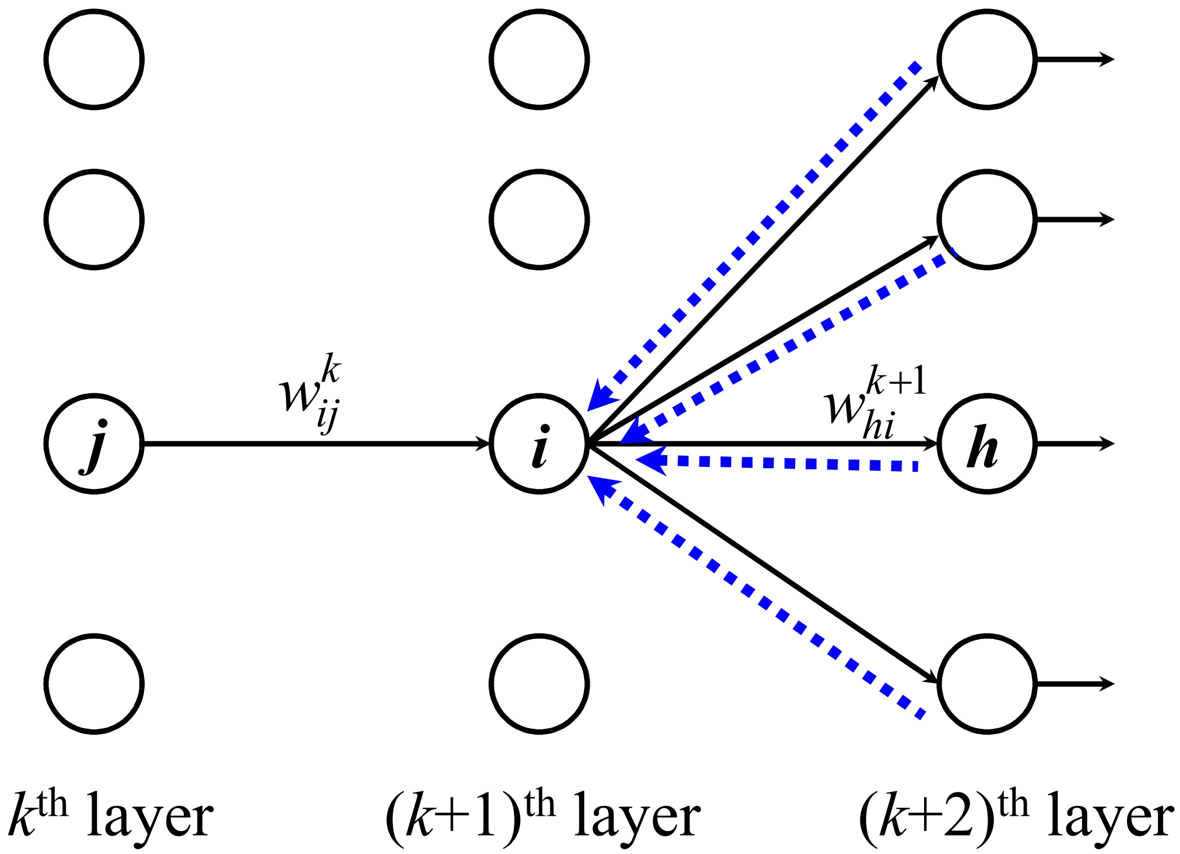
$$\delta_i^{k+1} = \left(d_i - o_i^{k+1}\right) f'\left(S_i^{k+1}\right)$$
$$= e_i \; f'\left(S_i^{k+1}\right)$$



Notice which weights are affected by e_i

MLP and EBP

Generalization of the Tuning Law - Hidden Layers



$$\frac{\partial J}{\partial w_{ij}^{k}} = \frac{\partial J}{\partial o_i^{k+1}} \frac{\partial o_i^{k+1}}{\partial S_i^{k+1}} \frac{\partial S_i^{k+1}}{\partial w_{ij}^{k}}$$

$$\frac{\partial J}{\partial o_i^{k+1}} = \sum_{h=1}^{n_{k+2}} \frac{\partial J}{\partial S_h^{k+2}} \frac{\partial S_h^{k+2}}{\partial o_i^{k+1}}$$

$$\frac{\partial J}{\partial o_i^{k+1}} = \sum_{h=1}^{n_{k+2}} \frac{\partial J}{\partial S_h^{k+2}} w_{hi}^{k+1}$$

MLP and EBP

Generalization of the Tuning Law - Hidden Layers

$$\frac{\partial J}{\partial w_{ij}^{k}} = \frac{\partial J}{\partial o_i^{k+1}}\frac{\partial o_i^{k+1}}{\partial S_i^{k+1}}\frac{\partial S_i^{k+1}}{\partial w_{ij}^{k}}$$

$$\frac{\partial J}{\partial o_i^{k+1}} = \sum_{h=1}^{n_{k+2}}\frac{\partial J}{\partial S_h^{k+2}}\frac{\partial S_h^{k+2}}{\partial o_i^{k+1}}$$

$$\frac{\partial J}{\partial o_i^{k+1}} = \sum_{h=1}^{n_{k+2}}\left[\frac{\partial J}{\partial S_h^{k+2}}\frac{\partial}{\partial o_i^{k+1}}\left(\sum_{i=1}^{n_{k+1}} w_{hi}^{k+1} o_i^{k+1}\right)\right]$$

$$\frac{\partial J}{\partial o_i^{k+1}} = \sum_{h=1}^{n_{k+2}}\frac{\partial J}{\partial S_h^{k+2}} w_{hi}^{k+1}$$

$$\delta_i^{k+1} = -\frac{\partial J}{\partial S_i^{k+1}}$$

$$\frac{\partial J}{\partial o_i^{k+1}} = -\sum_{h=1}^{n_{k+2}}\delta_h^{k+2} w_{hi}^{k+1}$$

$$\frac{\partial o_i^{k+1}}{\partial S_i^{k+1}} = \frac{df\left(S_i^{k+1}\right)}{dS_i^{k+1}} = f'\left(S_i^{k+1}\right)$$

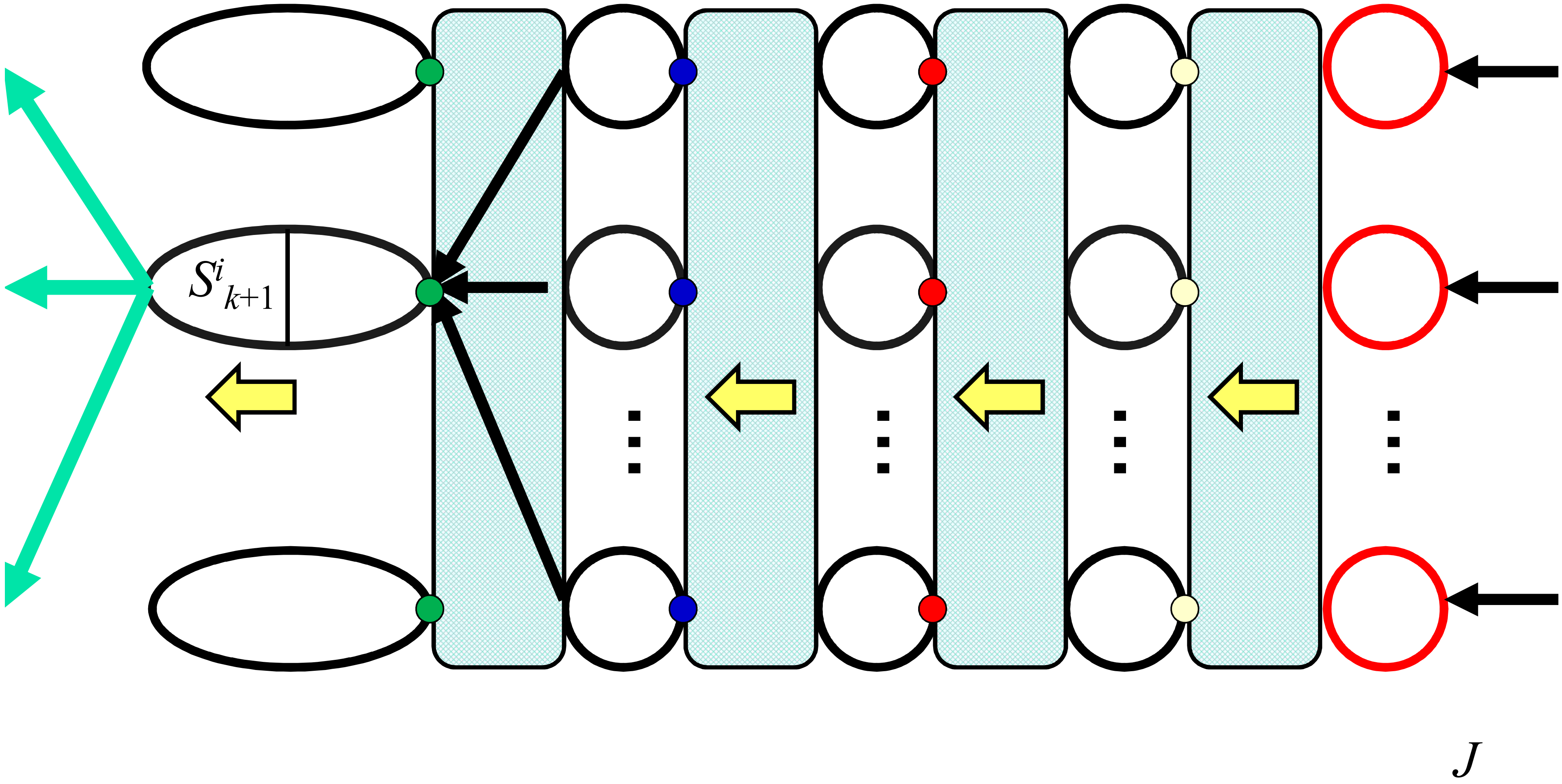
$$\frac{\partial S_i^{k+1}}{\partial w_{ij}^{k}} = o_j^k$$

$$\delta_i^{k+1} = \left(-\sum_{h=1}^{n_{k+2}}\delta_h^{k+2} w_{hi}^{k+1}\right) f'\left(S_i^{k+1}\right)$$

$$\Delta w_{ij}^{k} = \eta \delta_i^{k+1} o_j^k$$

MLP and EBP

Generalization of the Tuning Law - Hidden Layers



MLP and EBP

Pattern and Batch Learning

- Pattern Learning (Update after every pattern presentation):

Loop {
Present 1st pattern >> Update the parameters
Present 2nd pattern >> Update the parameters
...

- Batch Learning (Update after every epoche):

Loop {
Present 1st pattern >> Calculate D=Δw
Present 2nd pattern >> Calculate D=D+Δw
…
Present P-th pattern >> Calculate D=D+Δw
Update the parameters with the cumulative value D

MLP and EBP

Online and Offline Learning

- Offline Learning:

Train {
Data is available to train the network
Train the network
}

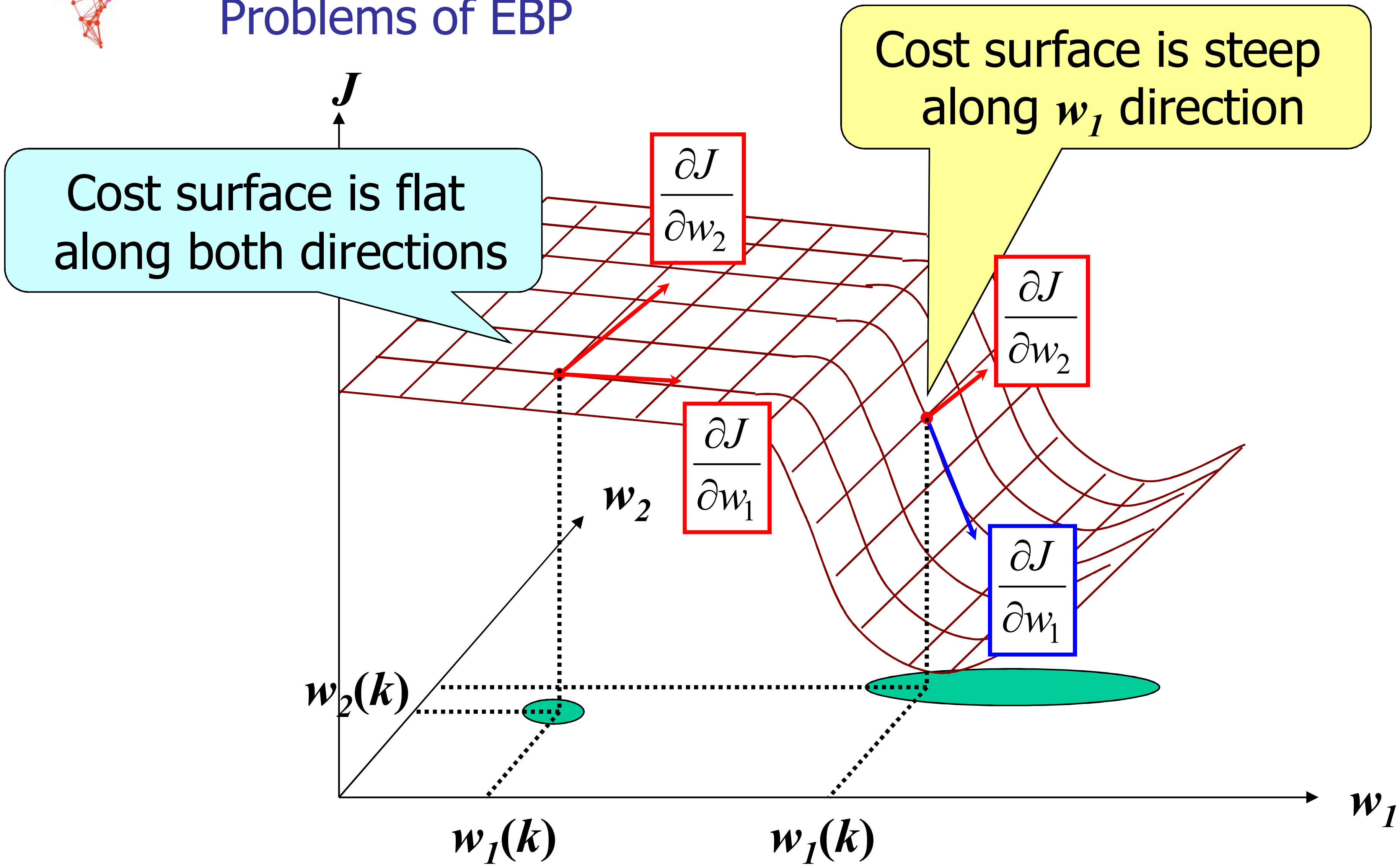
Test {
Unplug it from training loop, install into test system
Test it
}

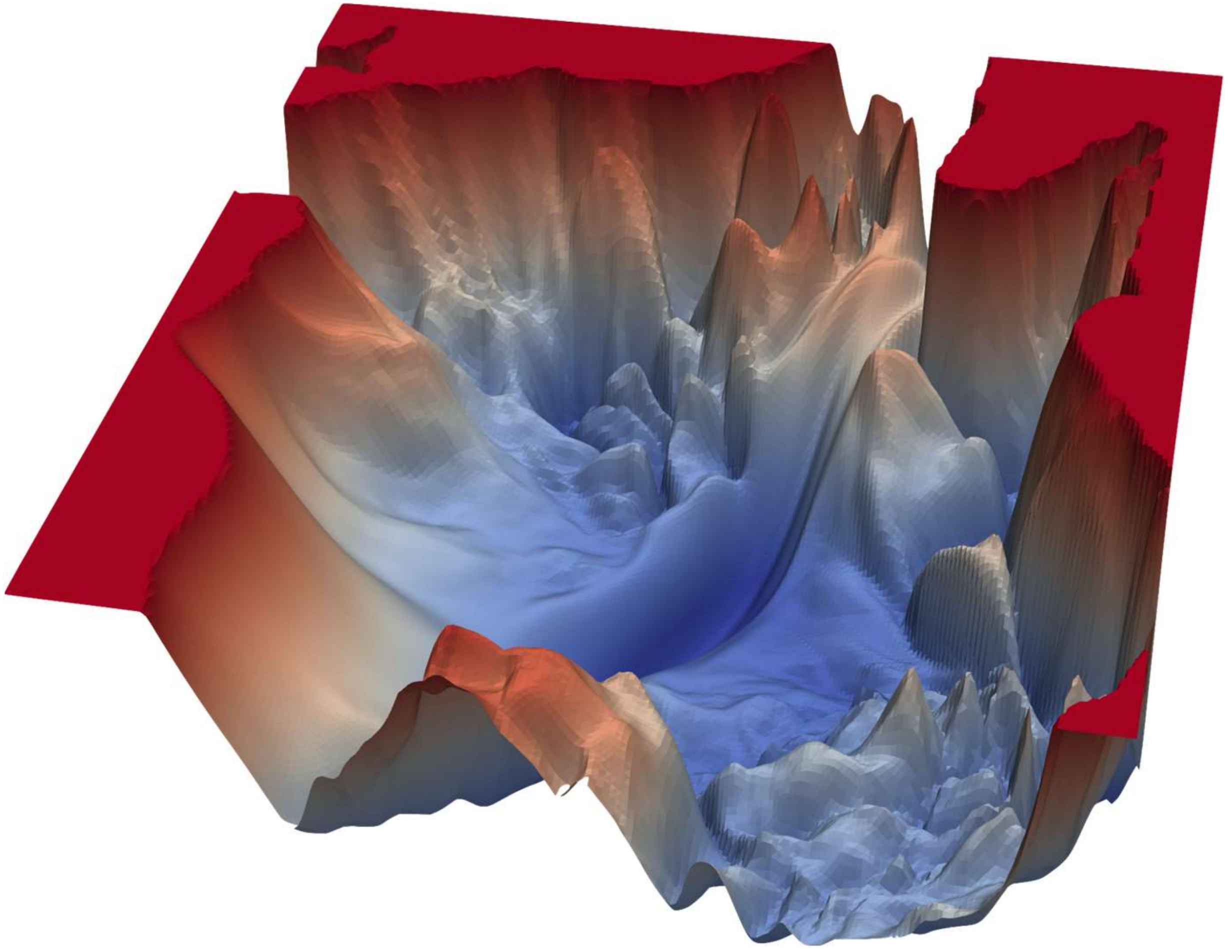
- Online Learning (Real-Time):

Train & Test {
Now $t=t_0$
An input/output pair emerges
Apply it, obtain output, tune the parameters
Now $t=t_0+\Delta t$
Another input/output pair emerges
Apply it, obtain output, tune the parameters
...
}

MLP and EBP

Problems of EBP





MLP and EBP
Problems of EBP - Momentum Term Addition

- Learning with EBP is a slow process!
  Look for methods to speed it up…
  - Momentum Term Addition

$$\Delta w_{ij}^{k}(t) = \mu \Delta w_{ij}^{k}(t-1) - \eta \frac{\partial J}{\partial w_{ij}^{k}}$$

where $0 < \mu < 1$

This term preserves some portion of the previous weight change so that the weight update dynamics is less influenced by the instant fluctuations. This operation acts like a filter!

MLP and EBP

Problems of EBP - Learning Rate Adaptation

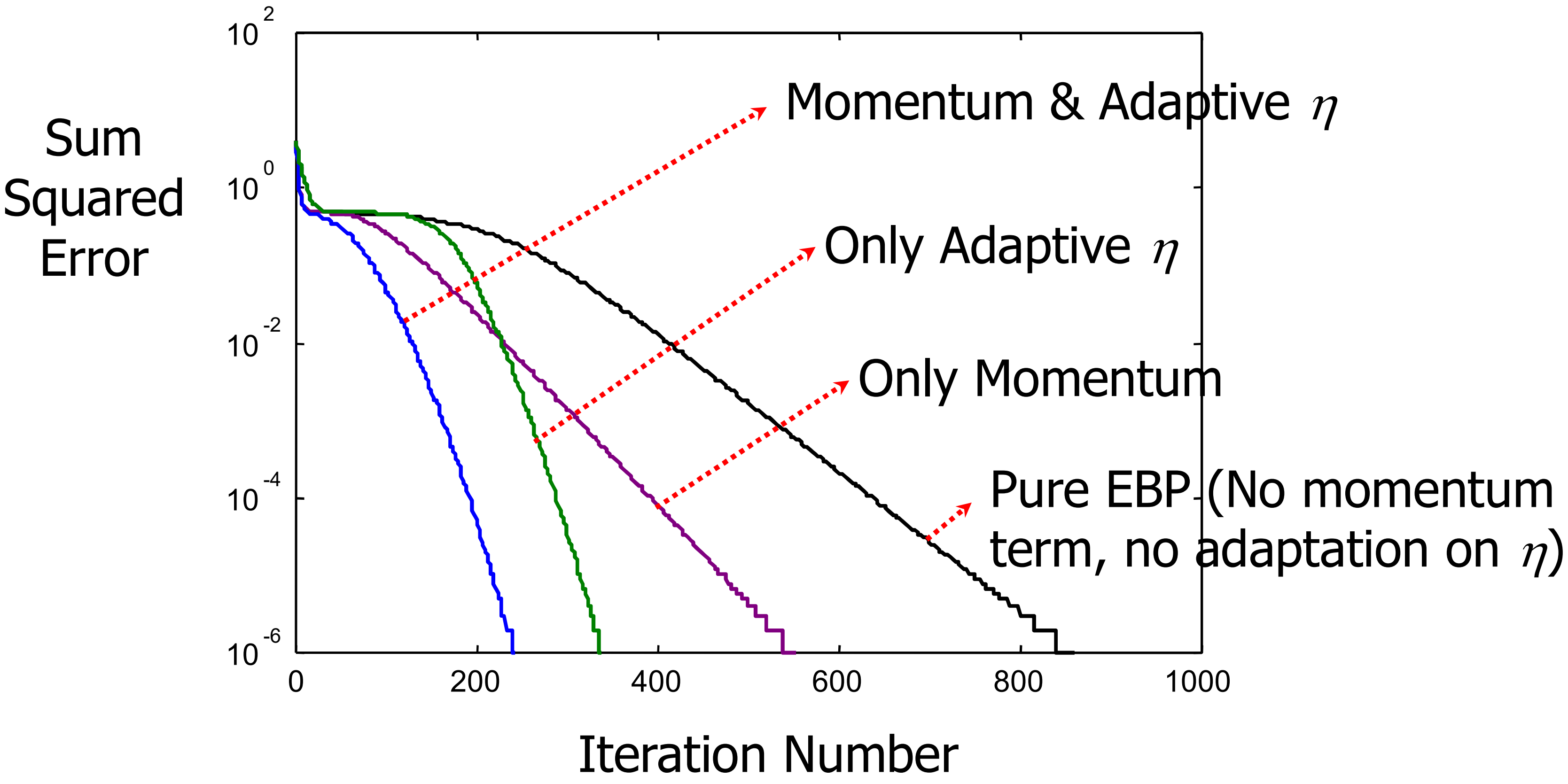
- Learning with EBP is a slow process!
  Look for methods to speed it up…
  - Learning Rate Adaptation

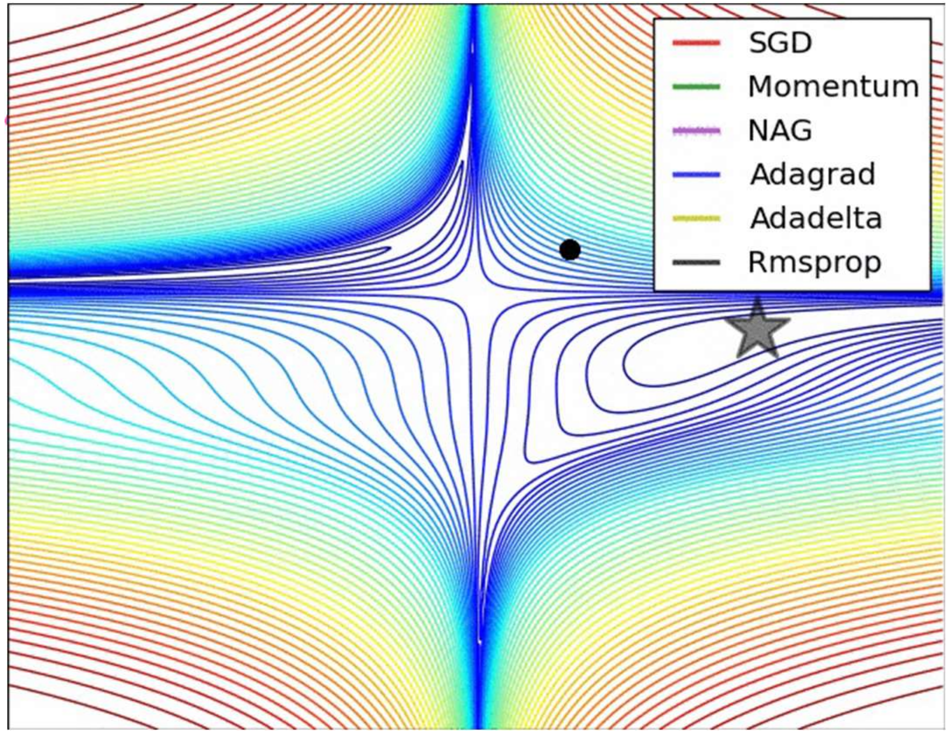
$$\eta(t) = \begin{cases} \eta(t-1) + \gamma & J(t) < J(t-1) \\ \beta\eta(t-1) & J(t) > J(t-1) \\ 0 & \text{otherwise} \end{cases}$$

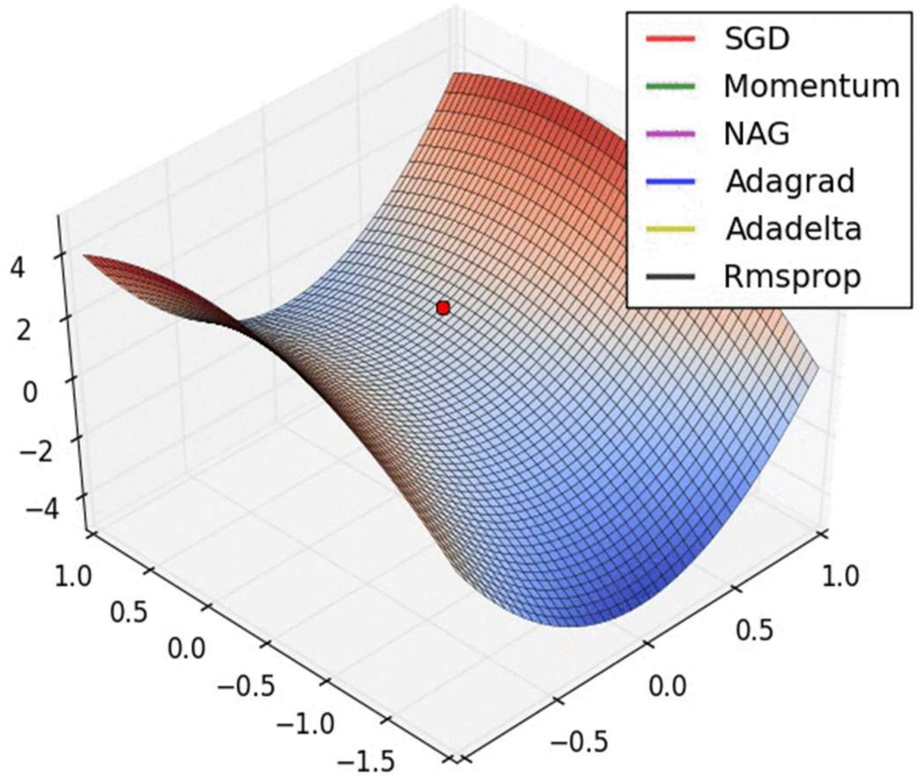
where $0 < \beta, \gamma < 1$

IF the cost is decreasing for several steps
THEN increase the learning rate by giving an increment γ
IF the cost is increasing for several steps
THEN decrease the learning rate geometrically
IF there is no change, go on searching...

MLP and EBP
A Comparison for XOR Problem

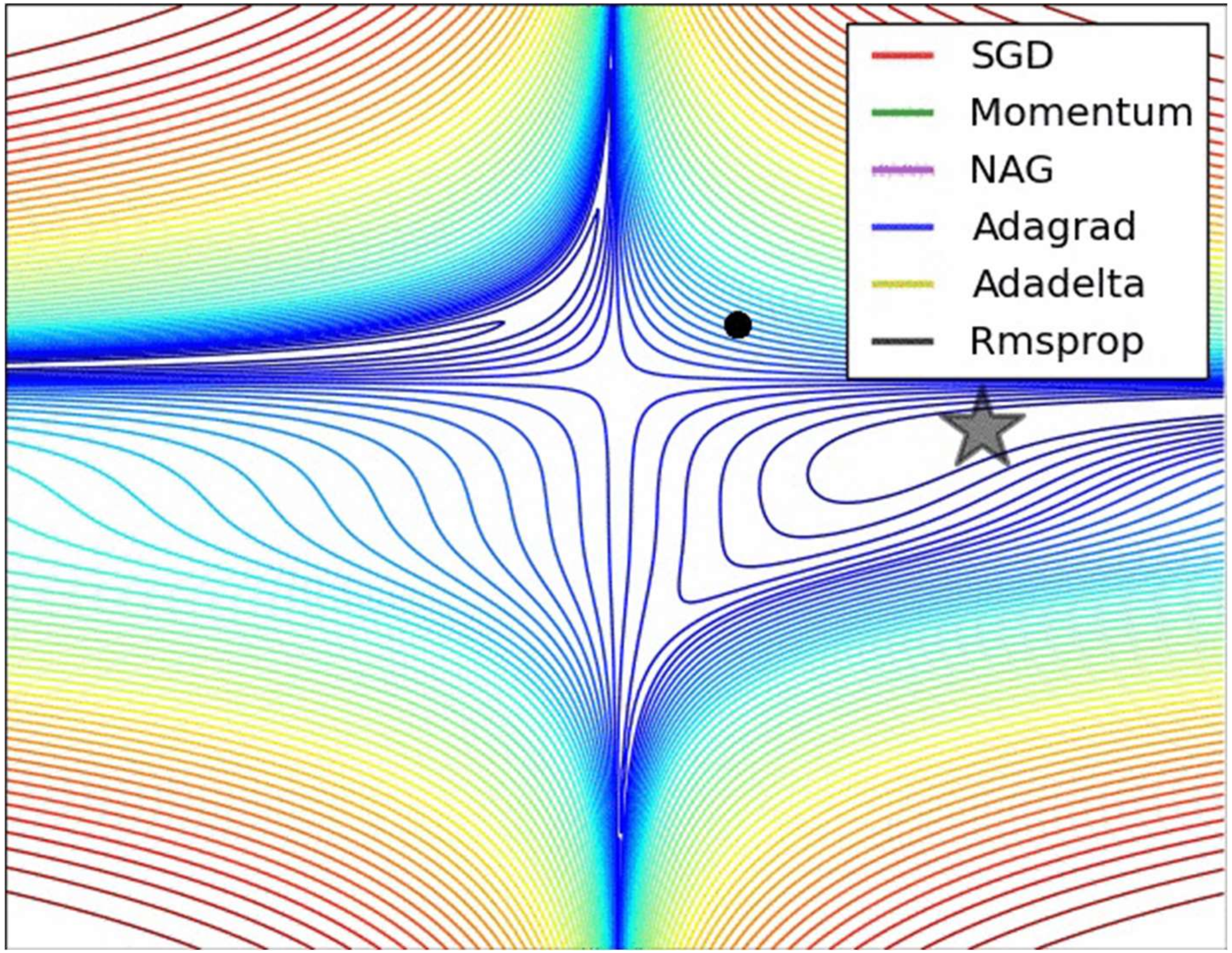






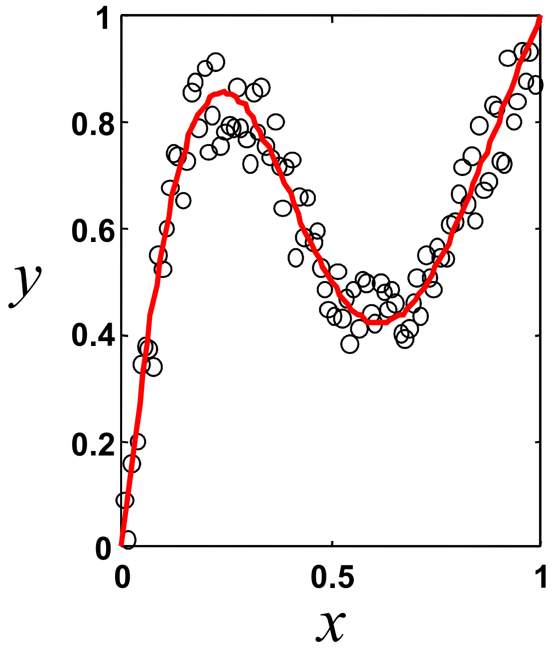
There are a number of other alternatives that perform poor or good, depending on your data and problem. Photo: CS231 Stanford, Credit: Alec Radford.

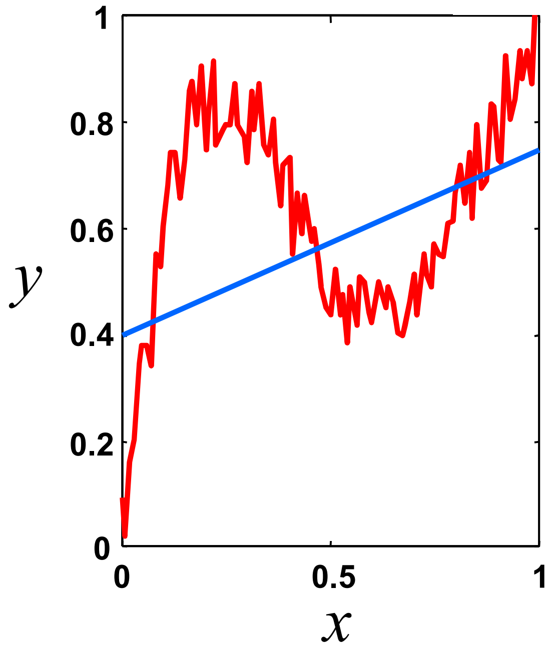
Read Dradient Descent discussion at: https://ruder.io/optimizing-gradient-descent/ Figures taken from this website.


SGD
Momentum
NAG
Adagrad
Adadelta
Rmsprop

MLP and EBP

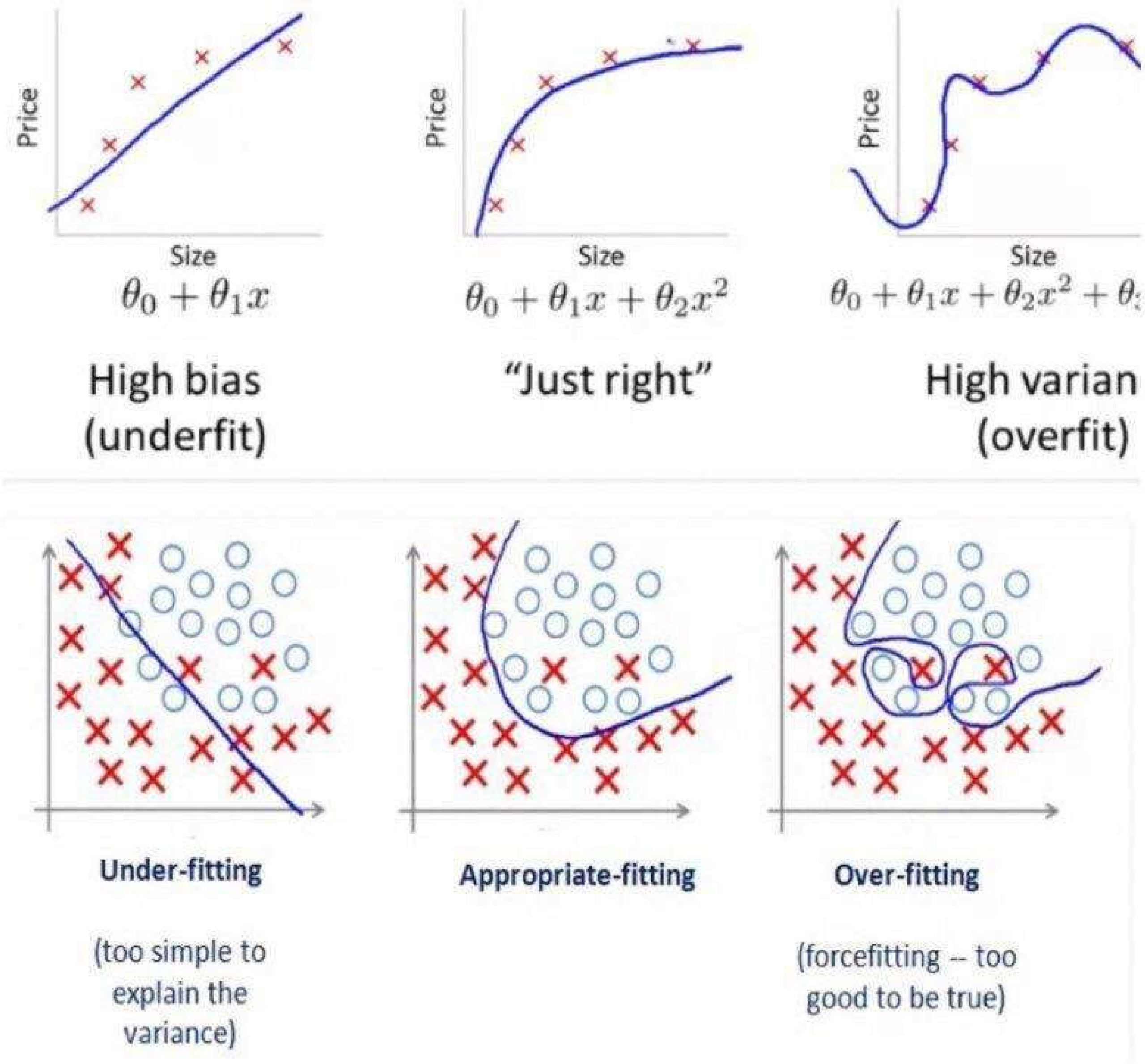
Memorization (Overfitting, Overtraining) and Generalization





- Which one is a better generalization of the depicted data?

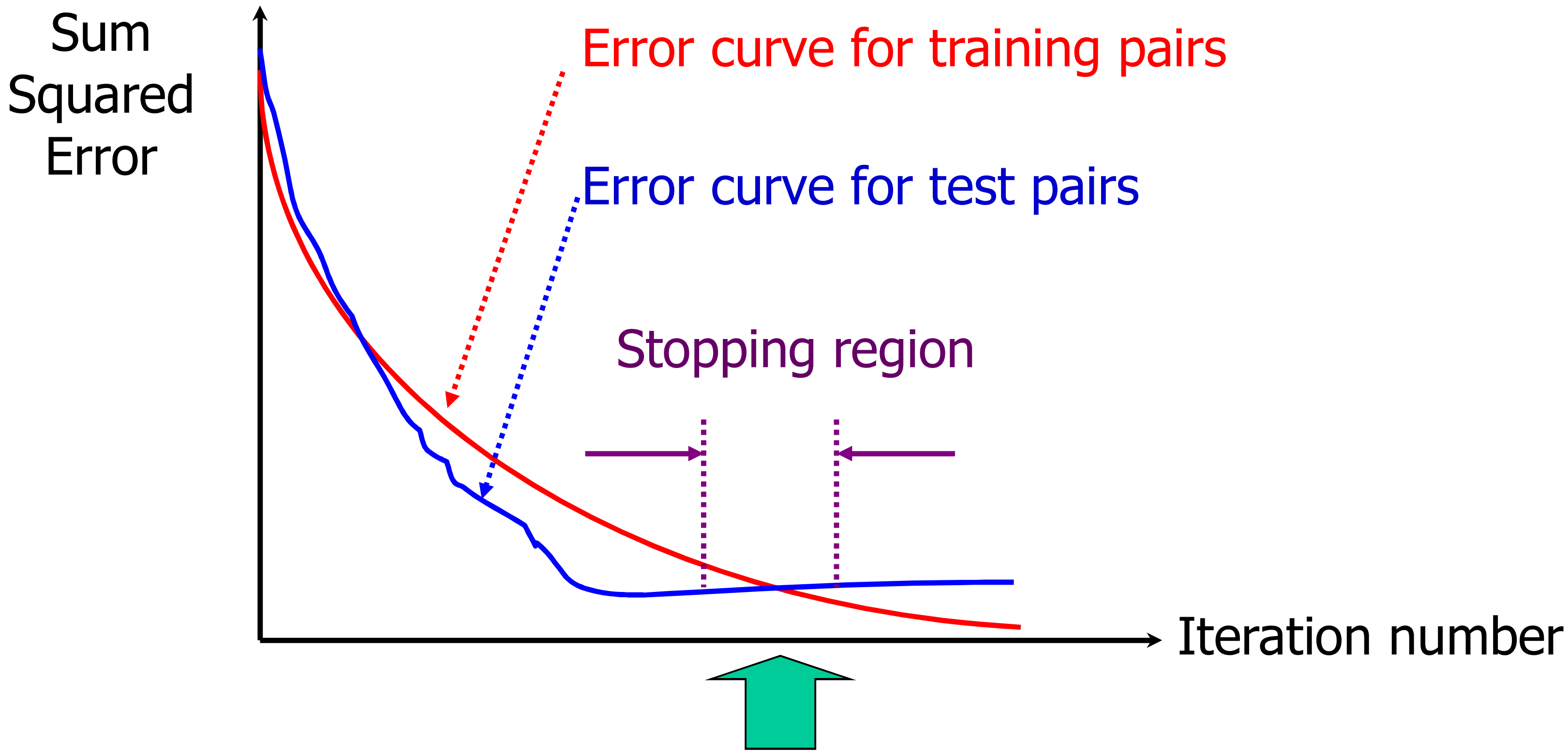
Over-, Under- and Well-fitted Models



Mehmet Önder Efe, *Neural Networks*, Lecture Notes, 2022.

MLP and EBP

Memorization (Overfitting, Overtraining) and Generalization



Critical region to stop. After some time memorization starts and the final hypersurface is forced to pass exactly through the given data points in the training data set.

MLP and EBP
Bias variance tradeoff

What is bias?: Bias is the difference between the average prediction of our model and the correct value which we are trying to predict. Model with high bias pays very little attention to the training data and oversimplifies the model. It always leads to high error on training and test data.

$$\mathrm{Bias}_D\left[\hat{f}(x;D)\right] = \mathrm{E}_D\left[\hat{f}(x;D)\right] - f(x)$$

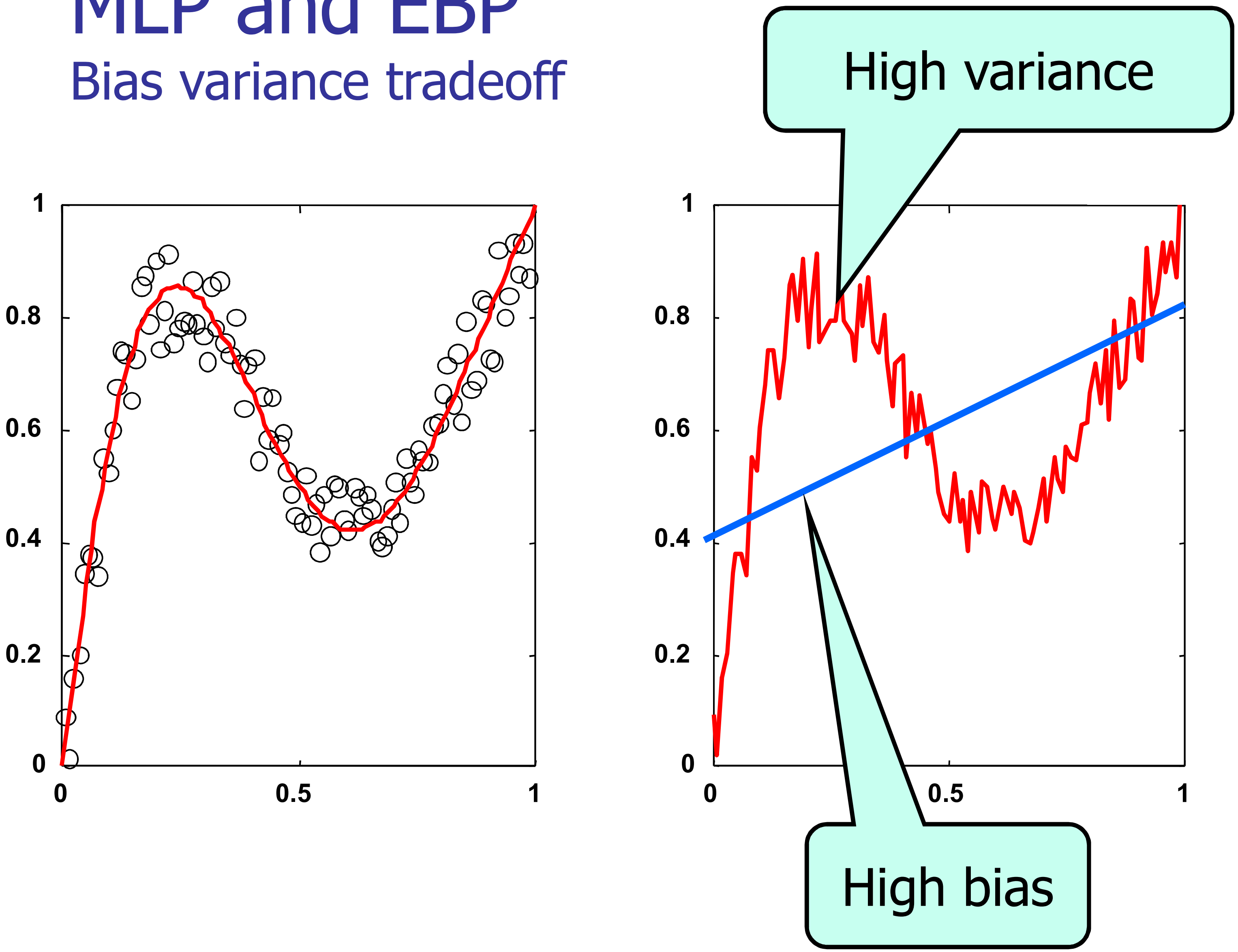
MLP and EBP

Bias variance tradeoff

What is variance?: Variance is the variability of model prediction for a given data point or a value which tells us spread of our data. Model with high variance pays a lot of attention to training data and does not generalize on the data which it hasn't seen before. As a result, such models perform very well on training data but has high error rates on test data.

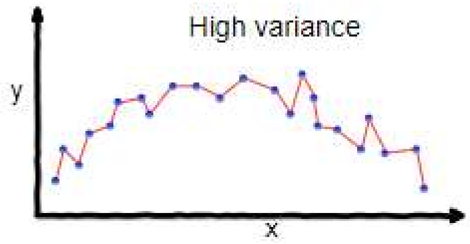
$$\mathrm{Var}_D\left[\hat{f}(x;D)\right] = \mathrm{E}_D\left[\left(\mathrm{E}_D[\hat{f}(x;D)] - \hat{f}(x;D)\right)^2\right]$$

MLP and EBP
Bias variance tradeoff

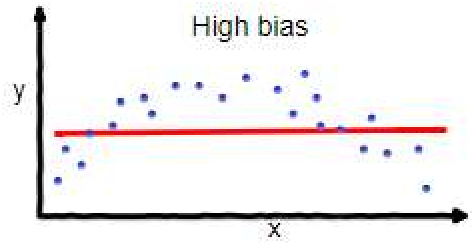


Mehmet Önder Efe, *Neural Networks*, Lecture Notes, 2022.

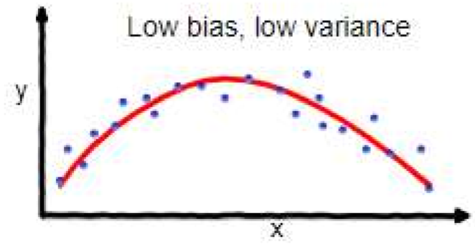
MLP and EBP
Bias variance tradeoff



overfitting



underfitting



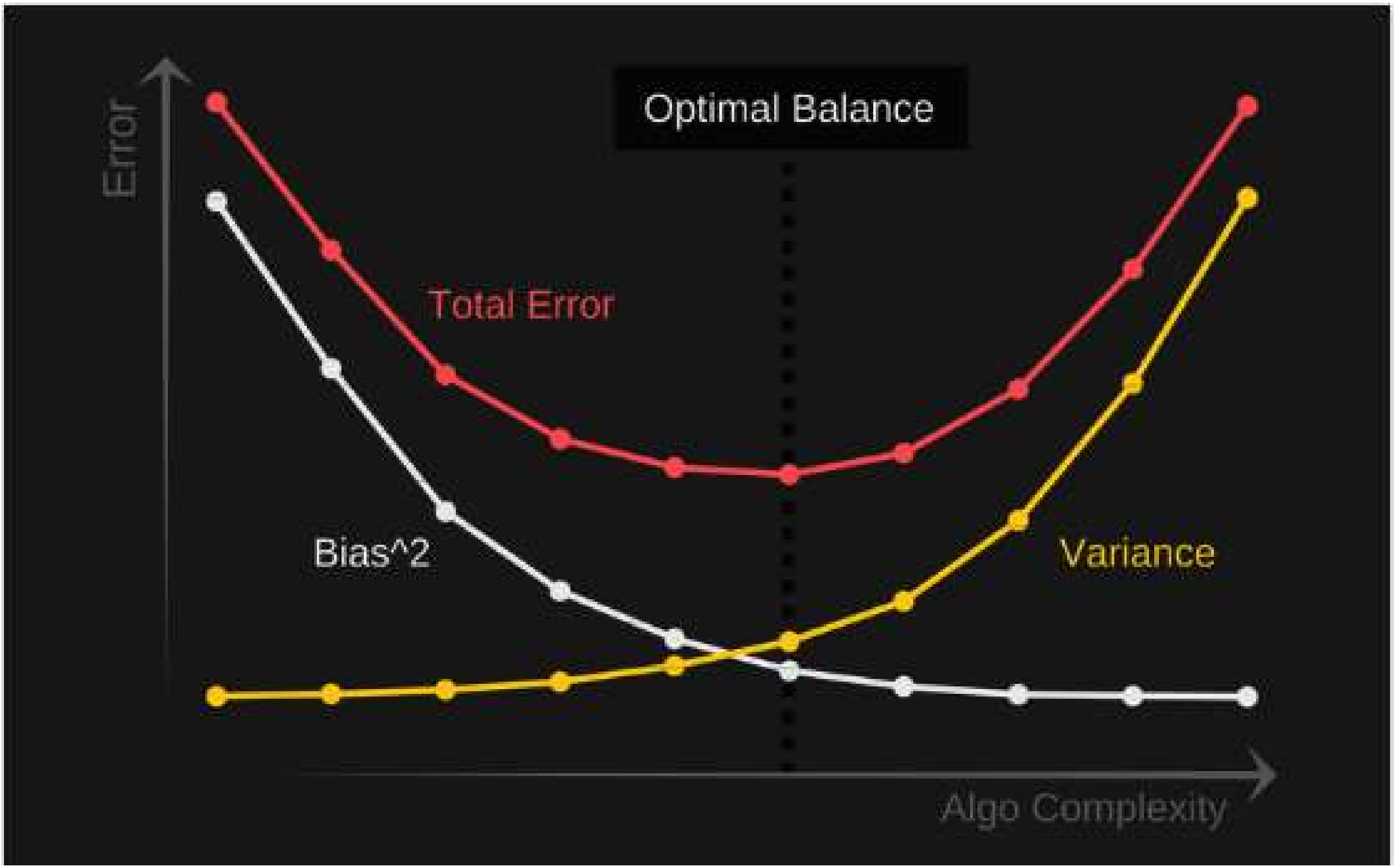
Good balance

$$\mathrm{E}_{D,\varepsilon}\left[\left(y-\hat{f}\left(x;D\right)\right)^2\right]=\left(\mathrm{Bias}_D\left[\hat{f}\left(x;D\right)\right]\right)^2+\mathrm{Var}_D\left[\hat{f}\left(x;D\right)\right]+\sigma^2$$

$$Err(x)=\left(E[\hat{f}\left(x\right)]-f(x)\right)^2+E\left[\left(\hat{f}\left(x\right)-E[\hat{f}\left(x\right)]\right)^2\right]+\sigma^2$$

$$Err(x)=\mathrm{Bias}^2+\mathrm{Variance}+\mathrm{Irreducible\ Error}$$

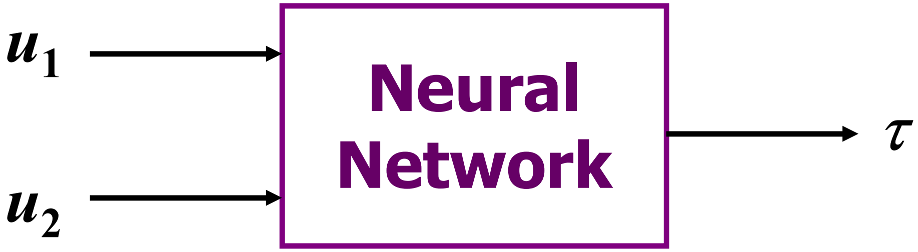
MLP and EBP
Bias variance tradeoff



Picture taken from https://towardsdatascience.com/understanding-the-bias-variance-tradeoff-165e6942b229

MLP and EBP

Normalization of Training Data



- Assume that you are given a set of training data, the entries of which are from the following intervals

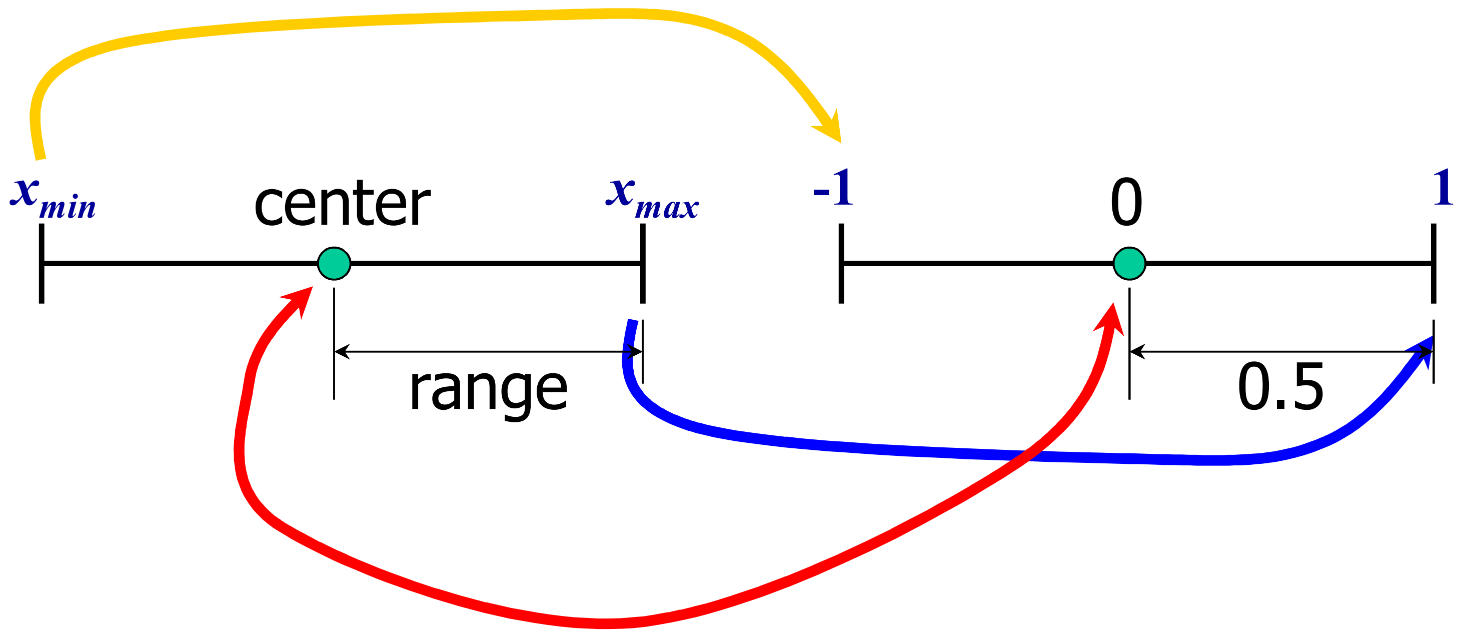
$$-100 \le \boldsymbol{u_1} \le 300 \ \text{ and } -0.07 \le \boldsymbol{u_2} \le 0.01 \text{ and } \ -3 \le \tau \le -1$$

- Can your neural network distinguish the given ranges? The answer is no! The network has a regular structure. Map every variable to the interval $-1 \le \boldsymbol{x} \le 1$. This lets the network operate on the same level of numerical accuracy.

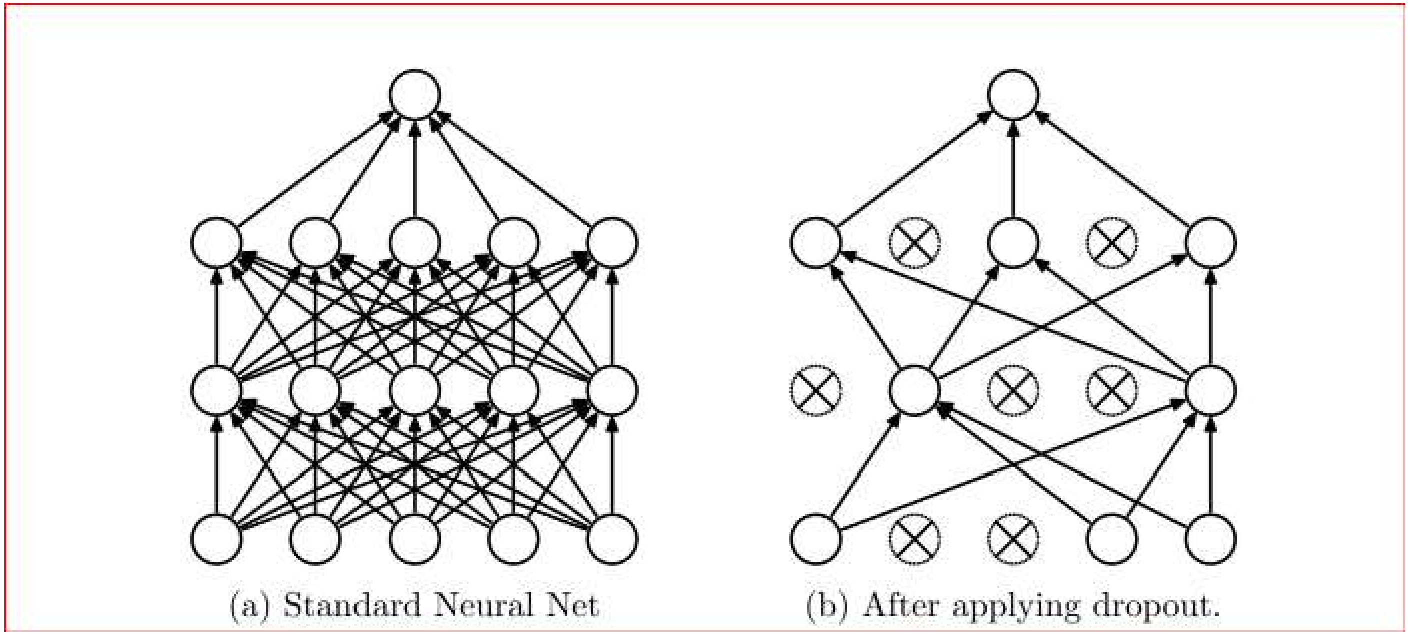
MLP and EBP

Normalization of Training Data

- $x_{min} \leq x \leq x_{max}$ is given
- center = $(x_{min} + x_{max})/2$
- range = $(x_{max} - x_{min})/2$
- Mapped data is given by $X_i = (x_i\text{-center})/\text{range}$



MLP and EBP
Dropout



- Pick a random number, if it is above a predefined threshold update the chosen neuron's weights, if not, those weights are kept the same.
- This distributes the total task over the entire neural structure

Figure taken from: https://medium.com/analytics-vidhya/neural-network-and-dropouts-b6690c869a18

MLP and EBP
Dropout

- Do not choose too large thresholds to break the connection from inputs to the outputs
- Works well when there are many training data
- You may consider dropping out individual weights as well
- See *N. Srivastava, Hinton, Krizhevsky, Sutskever and Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. University of Toronto. June 2014*.

MLP and EBP

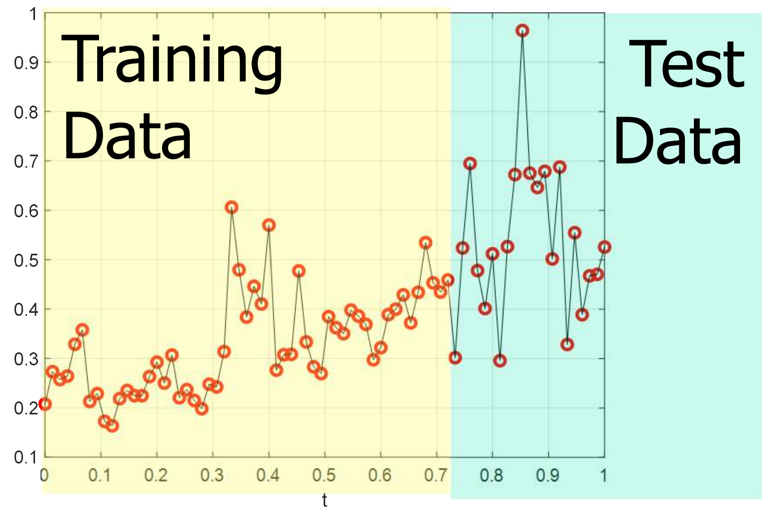
K-Fold Cross Validation: Why do we need it?

Training Data

...
...
...
...
...

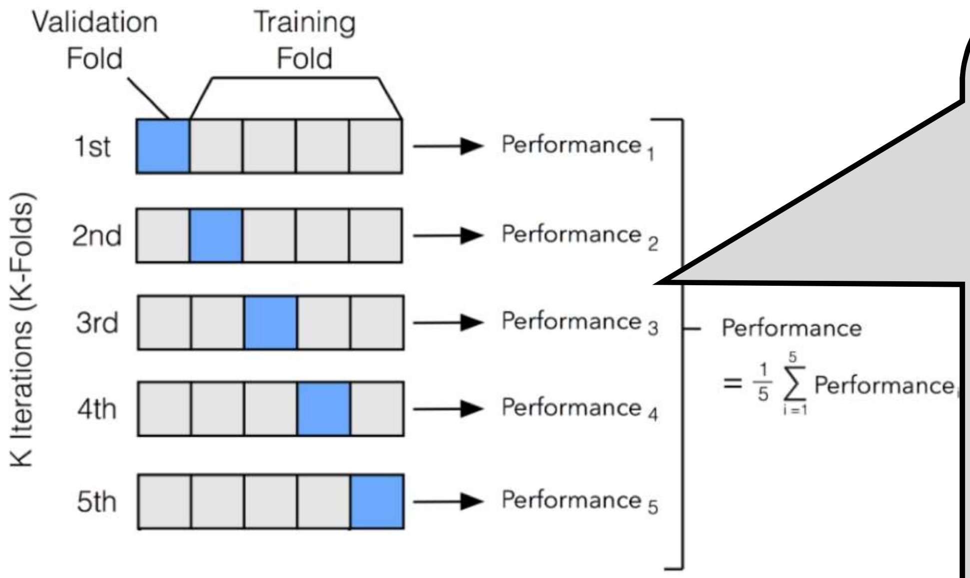
Test Data

...
...



What happens if this set contains totally dissimilar patterns you used in training dataset. You can never reduce the error caused by those samples.

MLP and EBP
Simple K-Fold Cross Validation

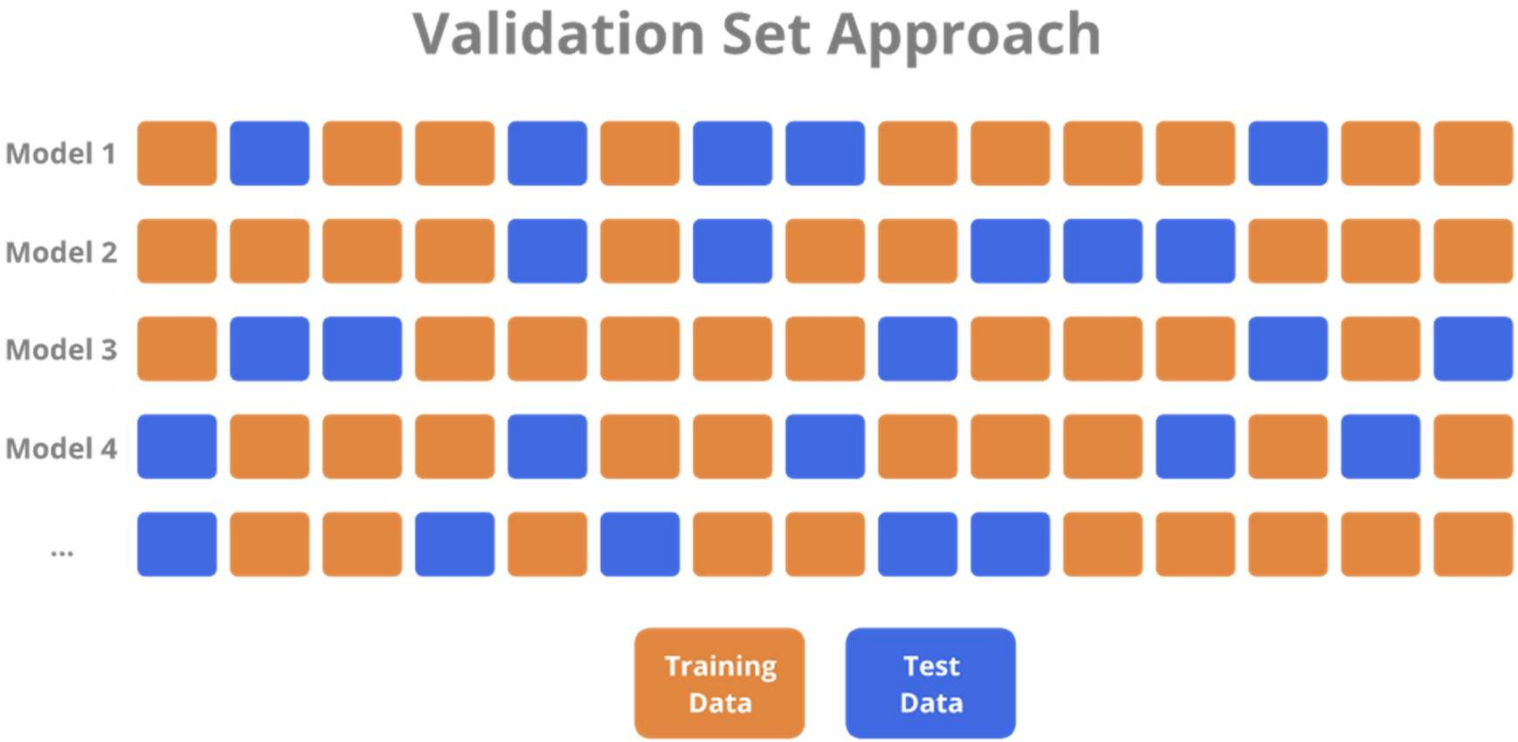


All patterns enter the training phase four times and test phase once. This scheme generates better models

- Do we have the problem of overfitting?
- See the performance of the model

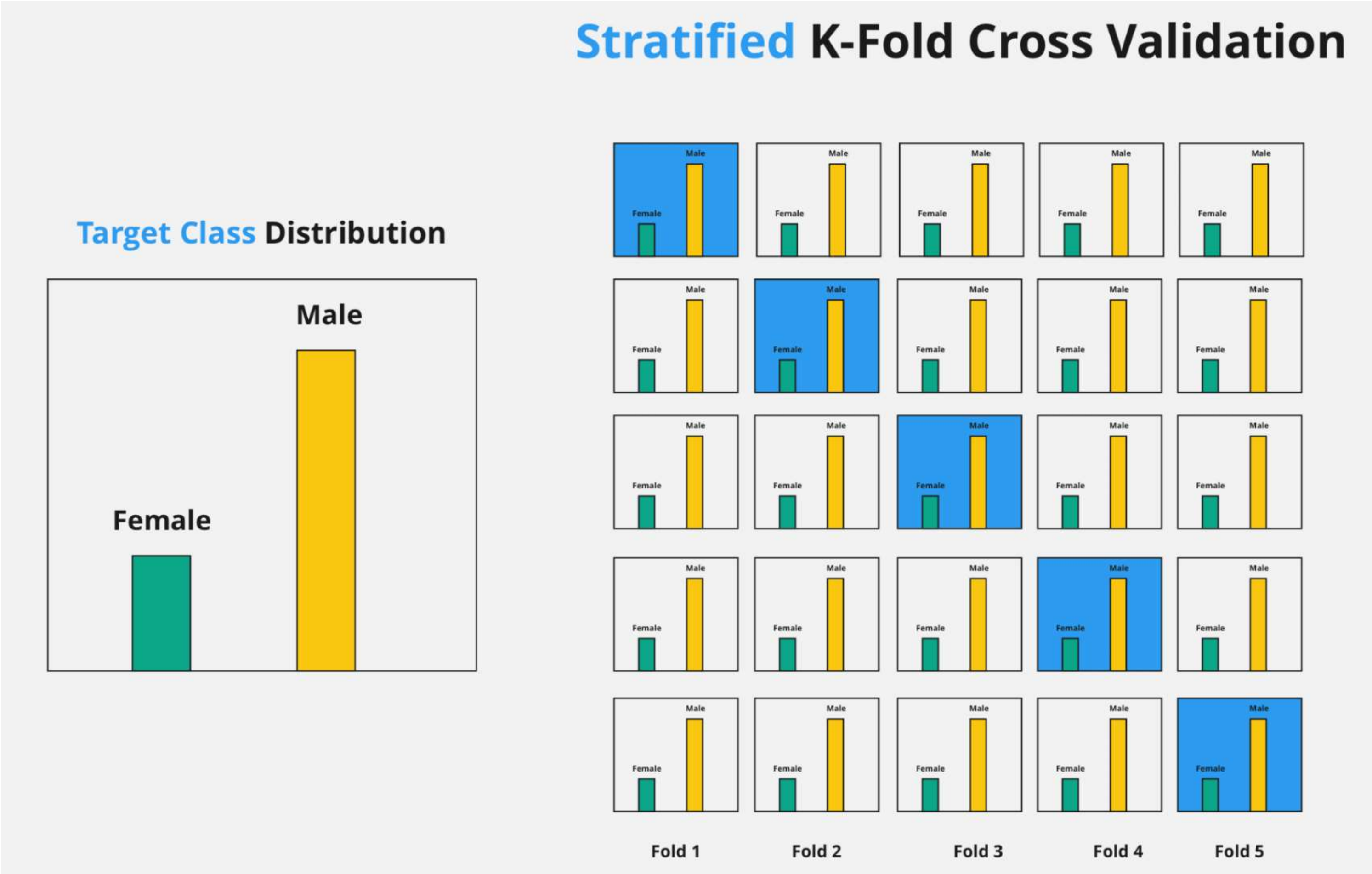
Figure taken from: https://medium.com/@gulcanogundur/model-se%C3%A7imi-k-fold-cross-validation-4635b61f143c

MLP and EBP
Leave-One-Out Cross Validation



MLP and EBP

Stratified K-Fold Cross Validation (Preserve Distribution)



MLP and EBP

Regularization

L_1 (Lasso) Regularization

$$J = \underbrace{\frac{1}{2}\sum_{i=1}^{n_{k+1}}\left(d_i - o_i^{k+1}(u,w)\right)^2}_{\text{Loss function}} + \underbrace{\lambda\sum_{\forall w_{ij}}\left|w_{ij}\right|}_{\text{Regularization term}}$$

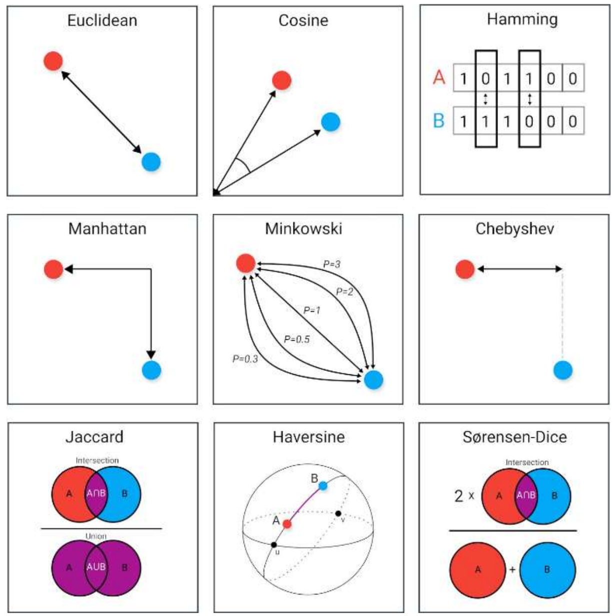
L_2 (Ridge) Regularization

$$J = \underbrace{\frac{1}{2}\sum_{i=1}^{n_{k+1}}\left(d_i - o_i^{k+1}(u,w)\right)^2}_{\text{Loss function}} + \underbrace{\lambda\sum_{\forall w_{ij}} w_{ij}^2}_{\text{Regularization term}}$$

- This prevents unnecessarily large values for few weights

MLP and EBP
Alternative cost (loss) functions

symbol	name	equation
$\mathcal{L}_1$	L_1 loss	$\|\mathbf{y}-\mathbf{o}\|_1$
$\mathcal{L}_2$	L_2 loss	$\|\mathbf{y}-\mathbf{o}\|_2^2$
$\mathcal{L}_1 \circ \sigma$	expectation loss	$\|\mathbf{y}-\sigma(\mathbf{o})\|_1$
$\mathcal{L}_2 \circ \sigma$	regularised expectation loss[1]	$\|\mathbf{y}-\sigma(\mathbf{o})\|_2^2$
$\mathcal{L}_\infty \circ \sigma$	Chebyshev loss	$\max_j \lvert\sigma(\mathbf{o})^{(j)}-\mathbf{y}^{(j)}\rvert$
hinge	hinge [13] (margin) loss	$\sum_j \max(0, \frac{1}{2}-\hat{\mathbf{y}}^{(j)}\mathbf{o}^{(j)})$
hinge^2	squared hinge (margin) loss	$\sum_j \max(0, \frac{1}{2}-\hat{\mathbf{y}}^{(j)}\mathbf{o}^{(j)})^2$
hinge^3	cubed hinge (margin) loss	$\sum_j \max(0, \frac{1}{2}-\hat{\mathbf{y}}^{(j)}\mathbf{o}^{(j)})^3$
log	log (cross entropy) loss	$-\sum_j \mathbf{y}^{(j)} \log\sigma(\mathbf{o})^{(j)}$
$\log^2$	squared log loss	$-\sum_j [\mathbf{y}^{(j)} \log\sigma(\mathbf{o})^{(j)}]^2$
tan	Tanimoto loss	$\frac{-\sum_j \sigma(\mathbf{o})^{(j)}\mathbf{y}^{(j)}}{\|\sigma(\mathbf{o})\|_2^2+\|\mathbf{y}\|_2^2-\sum_j \sigma(\mathbf{o})^{(j)}\mathbf{y}^{(j)}}$
D_{CS}	Cauchy-Schwarz Divergence [3]	$-\log\frac{\sum_j \sigma(\mathbf{o})^{(j)}\mathbf{y}^{(j)}}{\|\sigma(\mathbf{o})\|_2\|\mathbf{y}\|_2}$

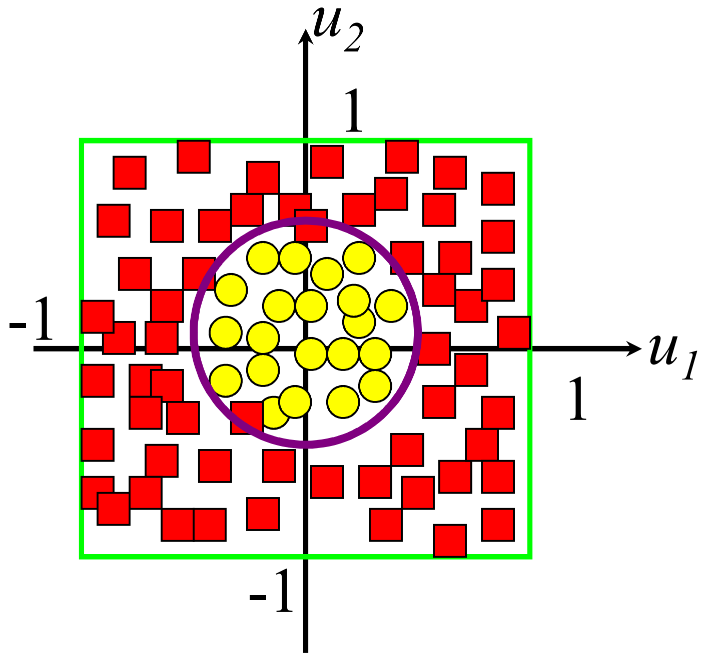


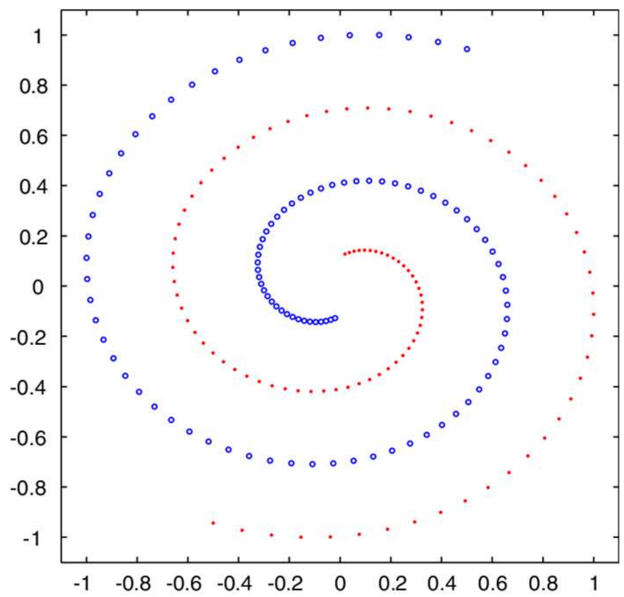
J. Janocha, W.M. Czarnecki, "On Loss Functions for Deep Neural Networks in Classification"

MLP and EBP

HOMEWORK #3

- Code EBP in Matlab (1 Hidden Layer is enough)
- Generate the training data for the below shown classes
- Train your network, show the result
- Circle radius on the left is 0.5

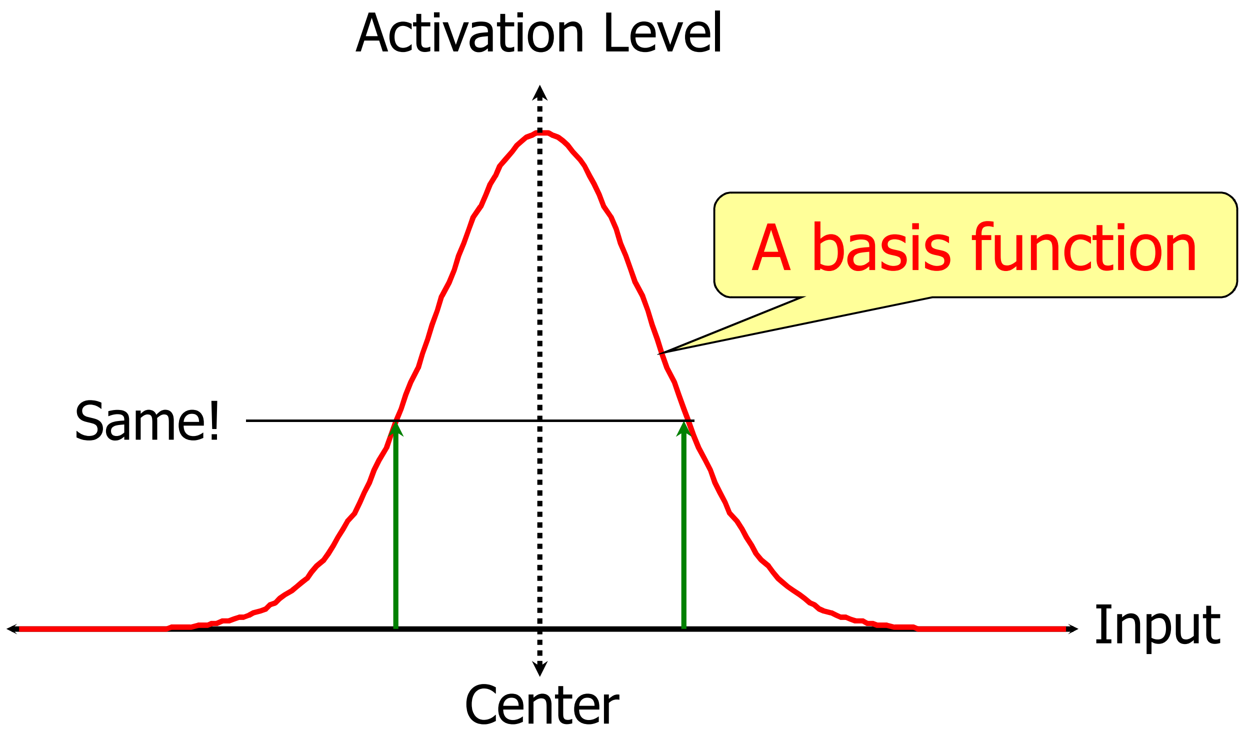




- Show your results together with error curve

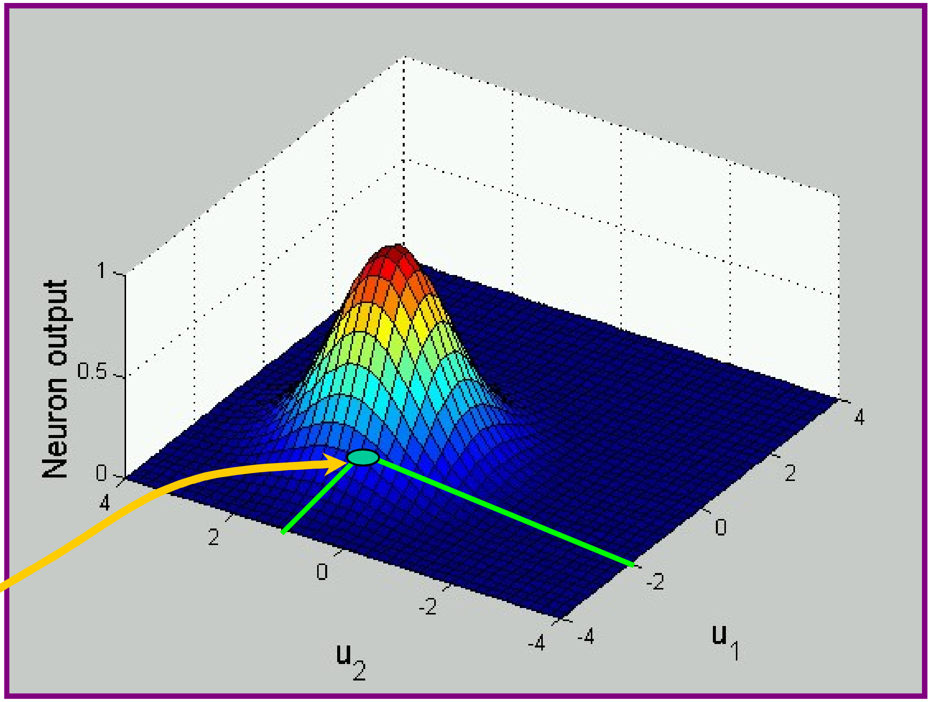
- Radial Basis Function Neural Networks
- Dynamic Neural Networks
- Second Order Training Schemes
    - Levenberg-Marquardt Algorithm
    - Gauss-Newton Algorithm

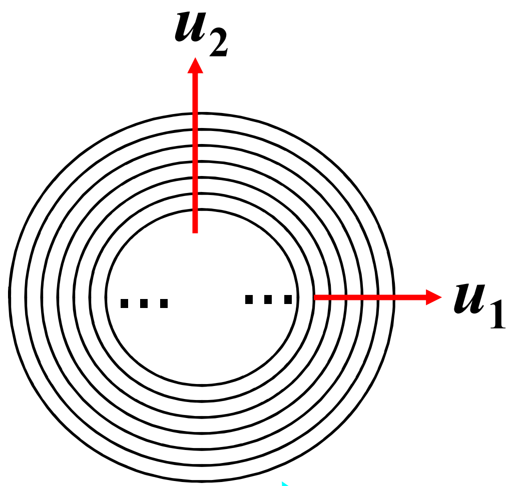
Radial Basis Function Neural Nets



- Inputs that are equal distance to the center return the same level of activation
- Notice the radial direction in 1D example above

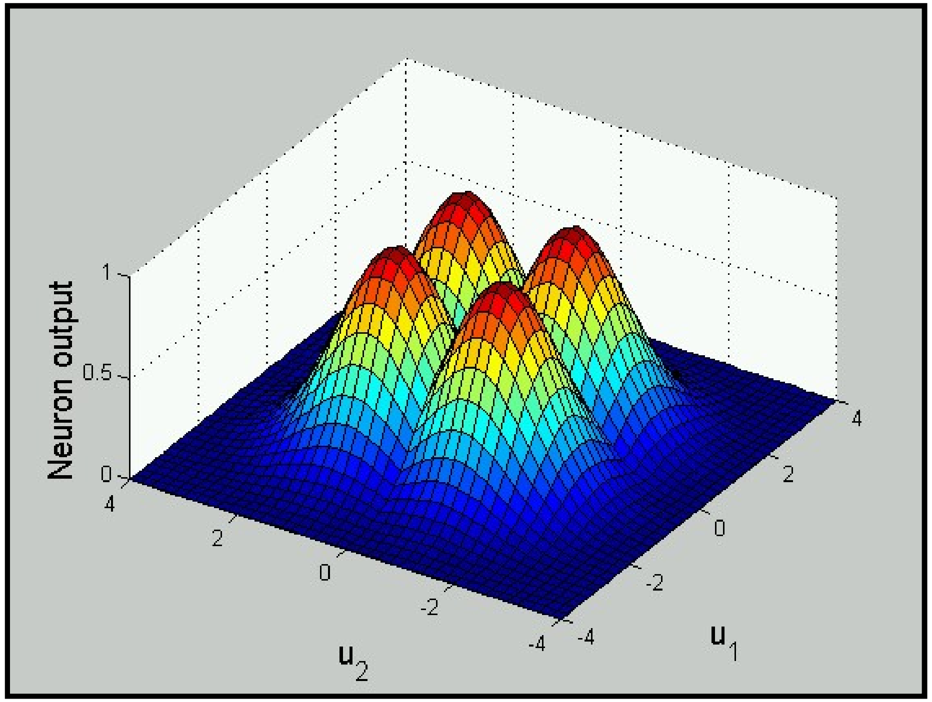
Radial Basis Function Neural Nets





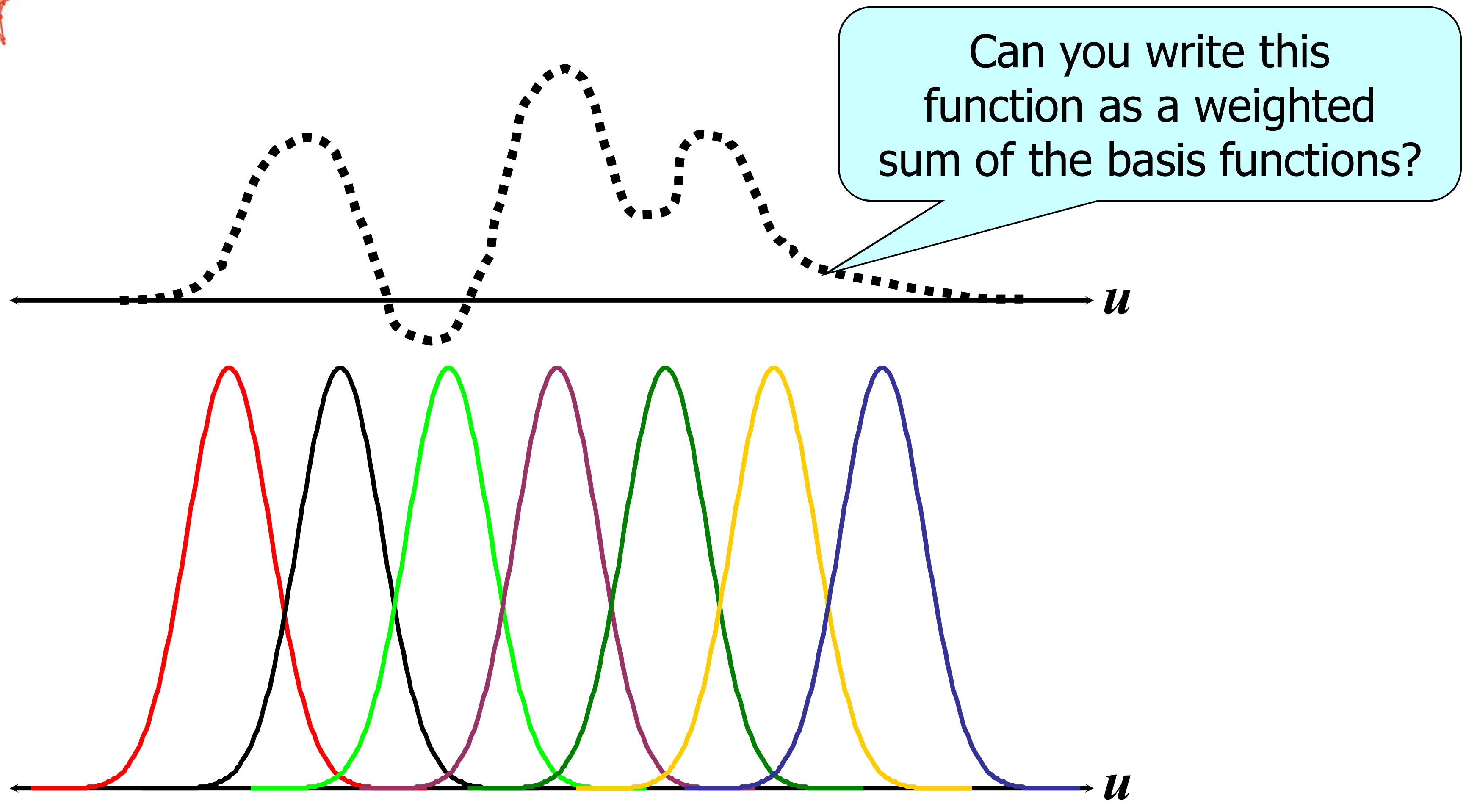
- As the input gets away from the center, the return value i.e. the level of activation decreases
- Notice the radial direction in 2D example above
- Center vector ($[-2\ 1]^T$) is a **feature**

Radial Basis Function Neural Nets

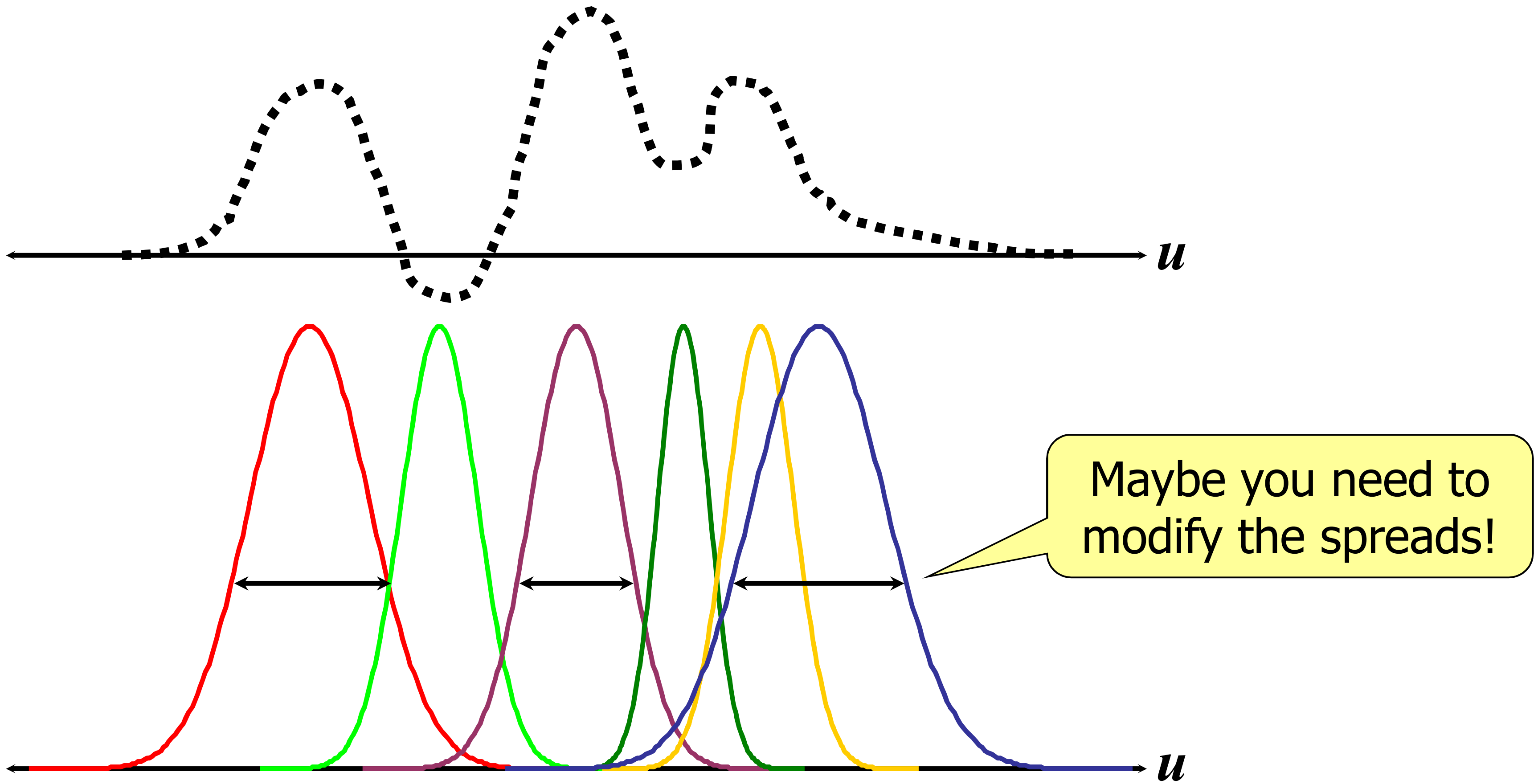


- If we cover the input space, with enough number of features (i.e. basis functions), we can express the events taking place over this domain in terms of the known features.
- This is a kind of decomposition of an event over the features

Radial Basis Function Neural Nets

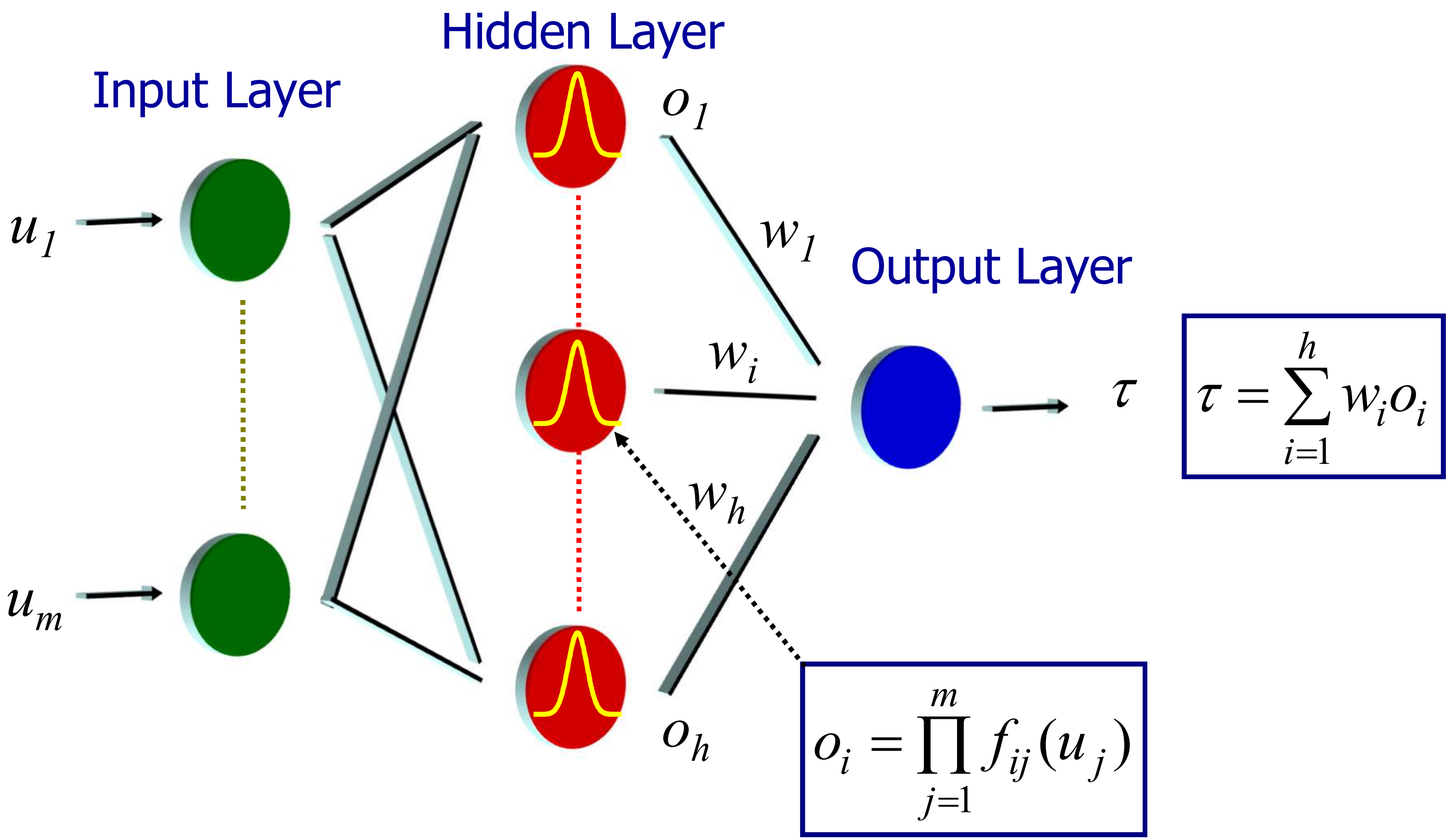


Radial Basis Function Neural Nets



- Let's make this a network and analyze its properties...

Radial Basis Function Neural Nets



- What functions are used as basis functions in the common practice?

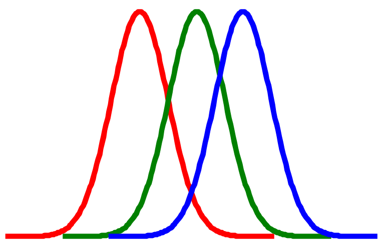
Radial Basis Function Neural Nets

Gaussian Basis Function

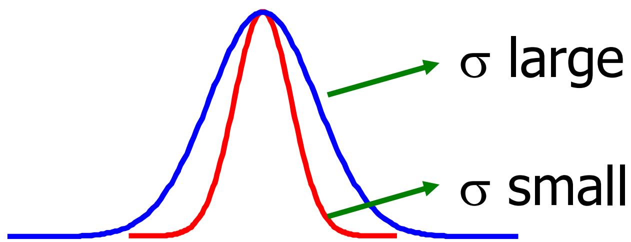
Input — Center for i-th neuron’s j-th input

$$f_{ij}(u_j) = \exp\left\{ -\left(\frac{u_j - c_{ij}}{\sigma_{ij}} \right)^2 \right\}$$

Variance (or spread)



Changing center



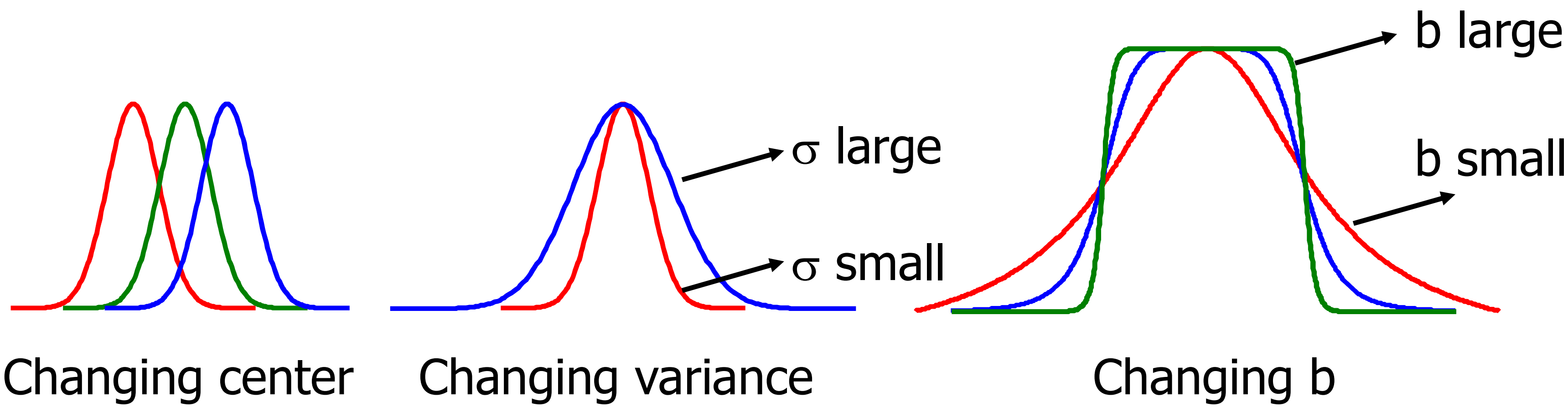
Changing variance

Radial Basis Function Neural Nets

Bell Shaped Function

Another shape parameter

$$f_{ij}(u_j) = \frac{1}{1+\left(\dfrac{u_j - c_{ij}}{\sigma_{ij}}\right)^{2b}}$$



Mehmet Önder Efe, *Neural Networks*, Lecture Notes, 2022.

Radial Basis Function Neural Nets

Computational Issues - A Tradeoff

$$f_{ij}(u_j) = \exp\left\{-\left(\frac{u_j - c_{ij}}{\sigma_{ij}}\right)^2\right\}$$

$$f_{ij}(u_j) = \frac{1}{1+\left(\frac{u_j - c_{ij}}{\sigma_{ij}}\right)^{2b}}$$

- One of them has 2 adjustable parameter, while the other has 3
- Gaussian is computationally inexpensive
- Bell-shaped one has more degrees of freedom in terms of representational flexibility

Radial Basis Function Neural Nets

Parameter Adjustment with Gradient Descent

$$\tau = \sum_{i=1}^{h} w_i o_i \qquad o_i = \prod_{j=1}^{m} f_{ij}(u_j) \qquad \Delta\phi = -\eta \frac{\partial J}{\partial \phi} = \eta (d - \tau) \frac{\partial \tau}{\partial \phi}$$

For Gaussian RBFNN

i*=1,2,…,*h

$$\frac{\partial \tau}{\partial w_i} = o_i$$

$$\frac{\partial \tau}{\partial c_{ij}} = \frac{\partial \tau}{\partial o_i} \frac{\partial o_i}{\partial f_{ij}} \frac{\partial f_{ij}}{\partial c_{ij}} = w_i \frac{o_i}{f_{ij}} \frac{\partial f_{ij}}{\partial c_{ij}}$$

$$\frac{\partial \tau}{\partial \sigma_{ij}} = \frac{\partial \tau}{\partial o_i} \frac{\partial o_i}{\partial f_{ij}} \frac{\partial f_{ij}}{\partial \sigma_{ij}} = w_i \frac{o_i}{f_{ij}} \frac{\partial f_{ij}}{\partial \sigma_{ij}}$$

Radial Basis Function Neural Nets

Parameter Adjustment with Gradient Descent

$$f_{ij}(u_j) = \exp\left\{-\left(\frac{u_j - c_{ij}}{\sigma_{ij}}\right)^2\right\}$$

$$\frac{\partial f_{ij}}{\partial c_{ij}} = \left(\frac{\partial}{\partial c_{ij}}\left\{-\left(\frac{u_j - c_{ij}}{\sigma_{ij}}\right)^2\right\}\right) f_{ij} = 2\frac{u_j - c_{ij}}{\sigma_{ij}^2} f_{ij}$$

$$\frac{\partial f_{ij}}{\partial \sigma_{ij}} = \left(\frac{\partial}{\partial \sigma_{ij}}\left\{-\left(\frac{u_j - c_{ij}}{\sigma_{ij}}\right)^2\right\}\right) f_{ij} = 2\frac{\left(u_j - c_{ij}\right)^2}{\sigma_{ij}^3} f_{ij}$$

Radial Basis Function Neural Nets

Parameter Adjustment with Gradient Descent

$$\tau = \sum_{i=1}^{h} w_i o_i$$

$$o_i = \prod_{j=1}^{m} f_{ij}(u_j)$$

$$\Delta\phi = -\eta \frac{\partial J}{\partial \phi} = \eta (d - \tau) \frac{\partial \tau}{\partial \phi}$$

For Gaussian RBFNN

i*=1,2,…,*h

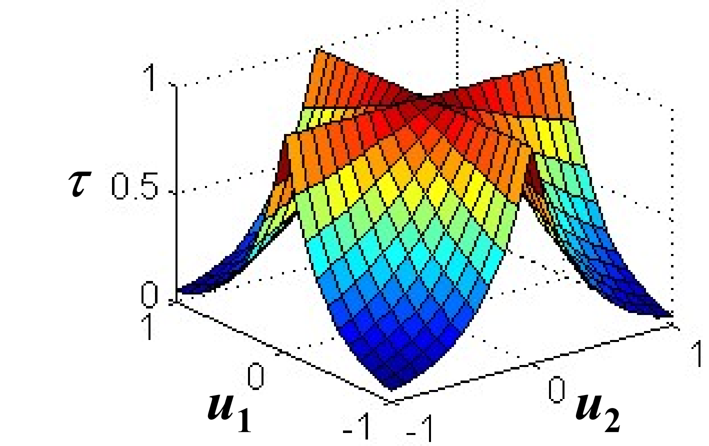
$$\Delta w_i = \eta (d - \tau) o_i$$

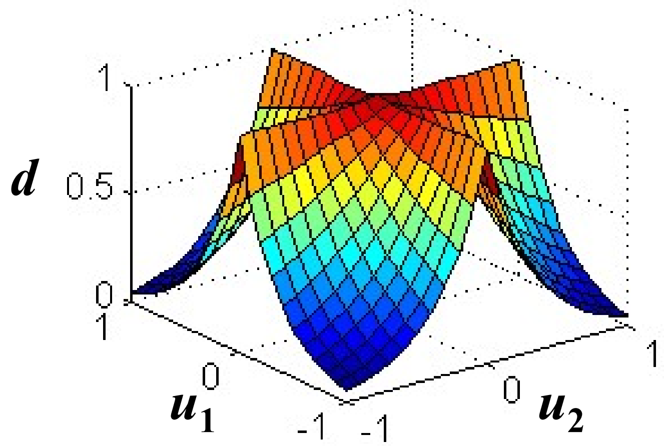
$$\Delta c_{ij} = 2\eta (d - \tau) w_i o_i \frac{u_j - c_{ij}}{\sigma_{ij}^2}$$

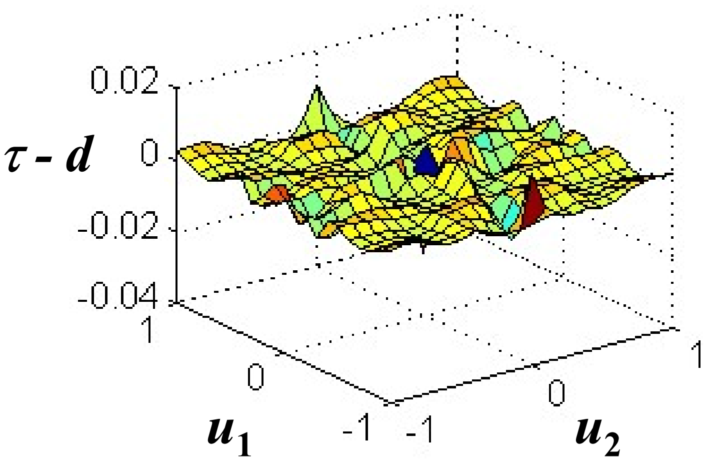
$$\Delta \sigma_{ij} = 2\eta (d - \tau) w_i o_i \frac{\left(u_j - c_{ij}\right)^2}{\sigma_{ij}^3}$$

Radial Basis Function Neural Nets

An Example

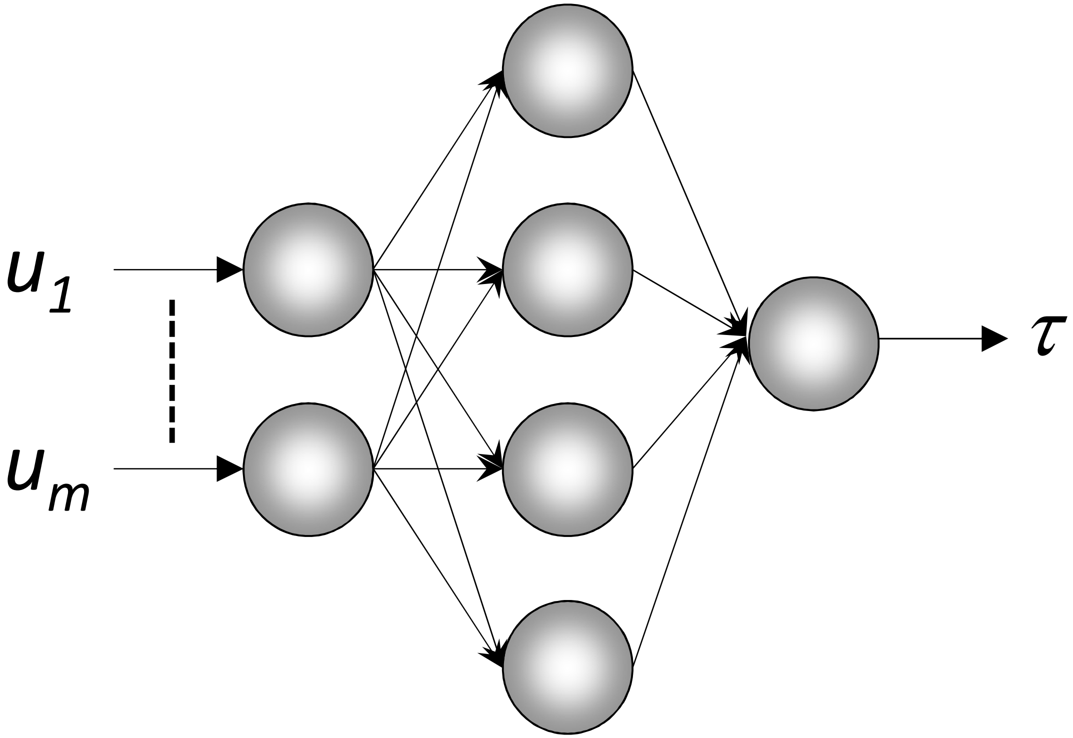






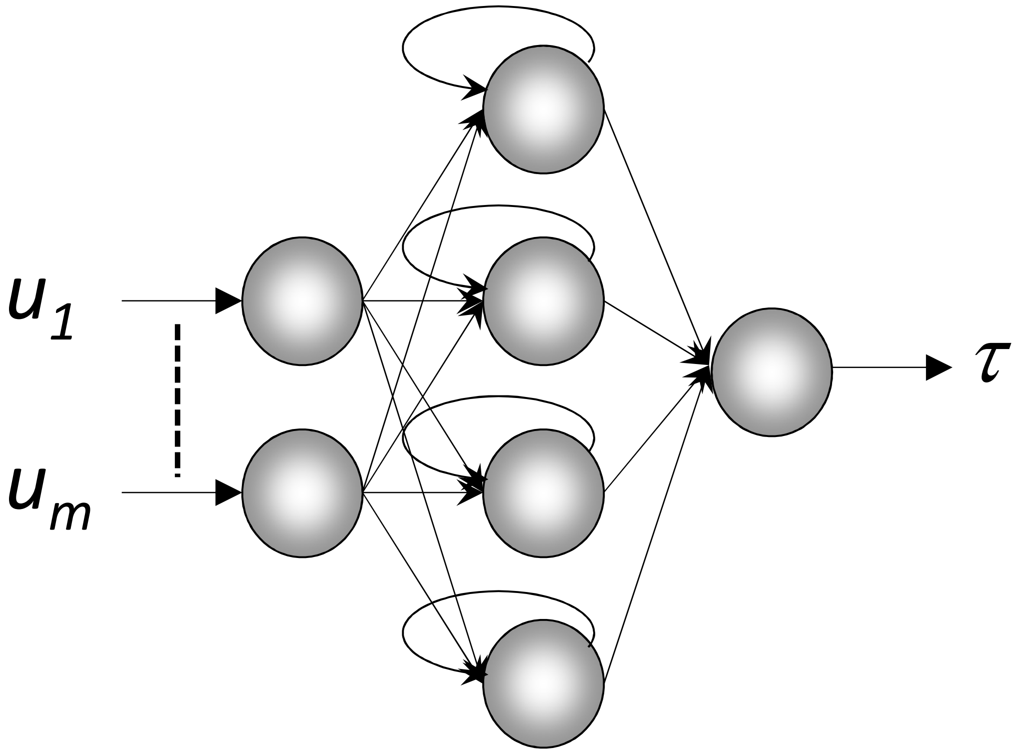
- 2-25-1 GRBFNN configuration
- Linearly sampled 441 pairs
- SSE decreases to 5e-4

Architectural Varieties



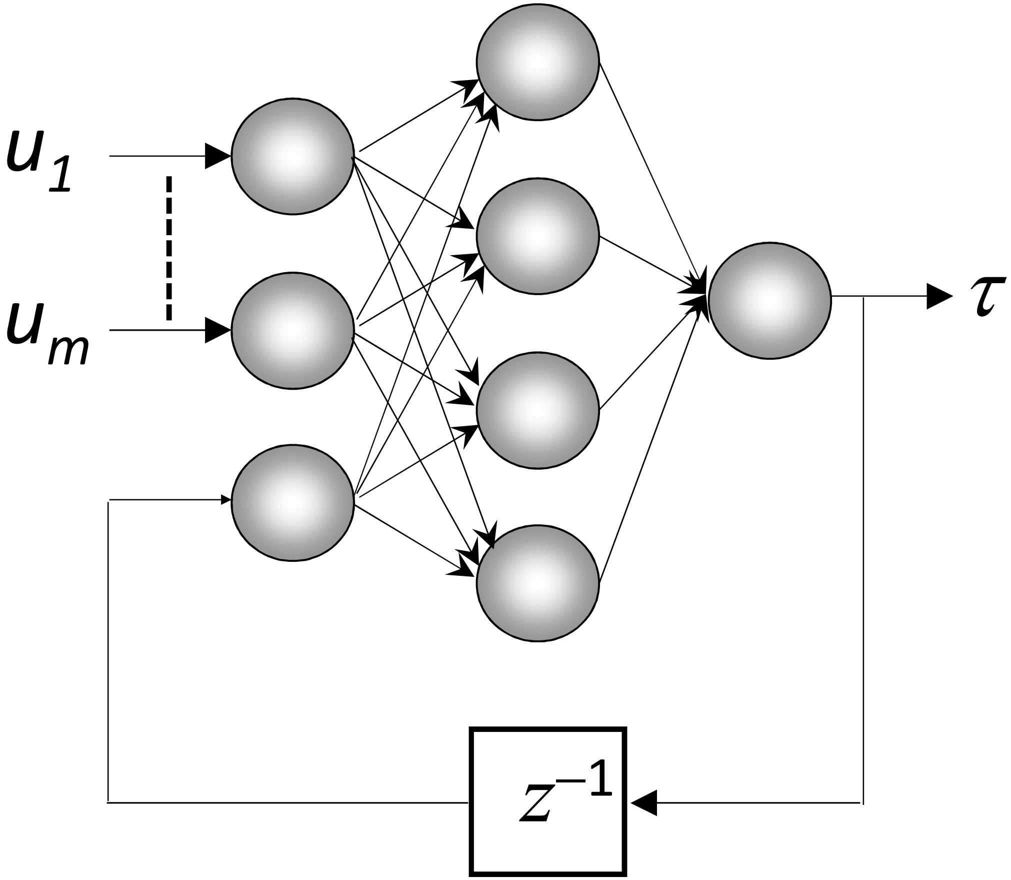
M.Ö. Efe and C. Kasnakoğlu, "A Comparison of Architectural Varieties in Radial Basis Function Neural Networks," World Congress on Computational Intelligence (WCCI'08) June 1-6, Hong Kong, pp.66-71, 2008.

Architectural Varieties



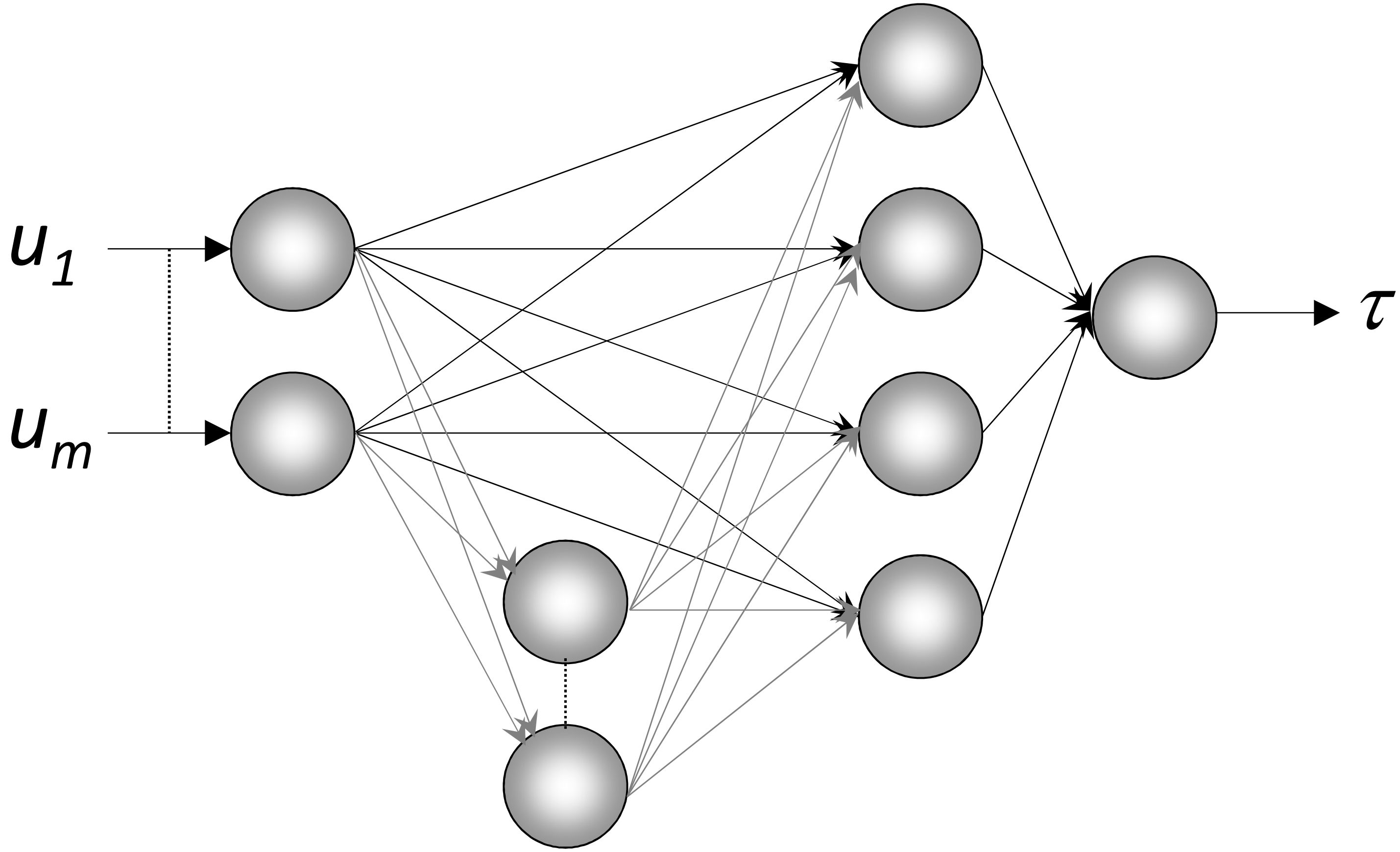
M.Ö. Efe and C. Kasnakoğlu, "A Comparison of Architectural Varieties in Radial Basis Function Neural Networks," World Congress on Computational Intelligence (WCCI'08) June 1-6, Hong Kong, pp.66-71, 2008.

Architectural Varieties



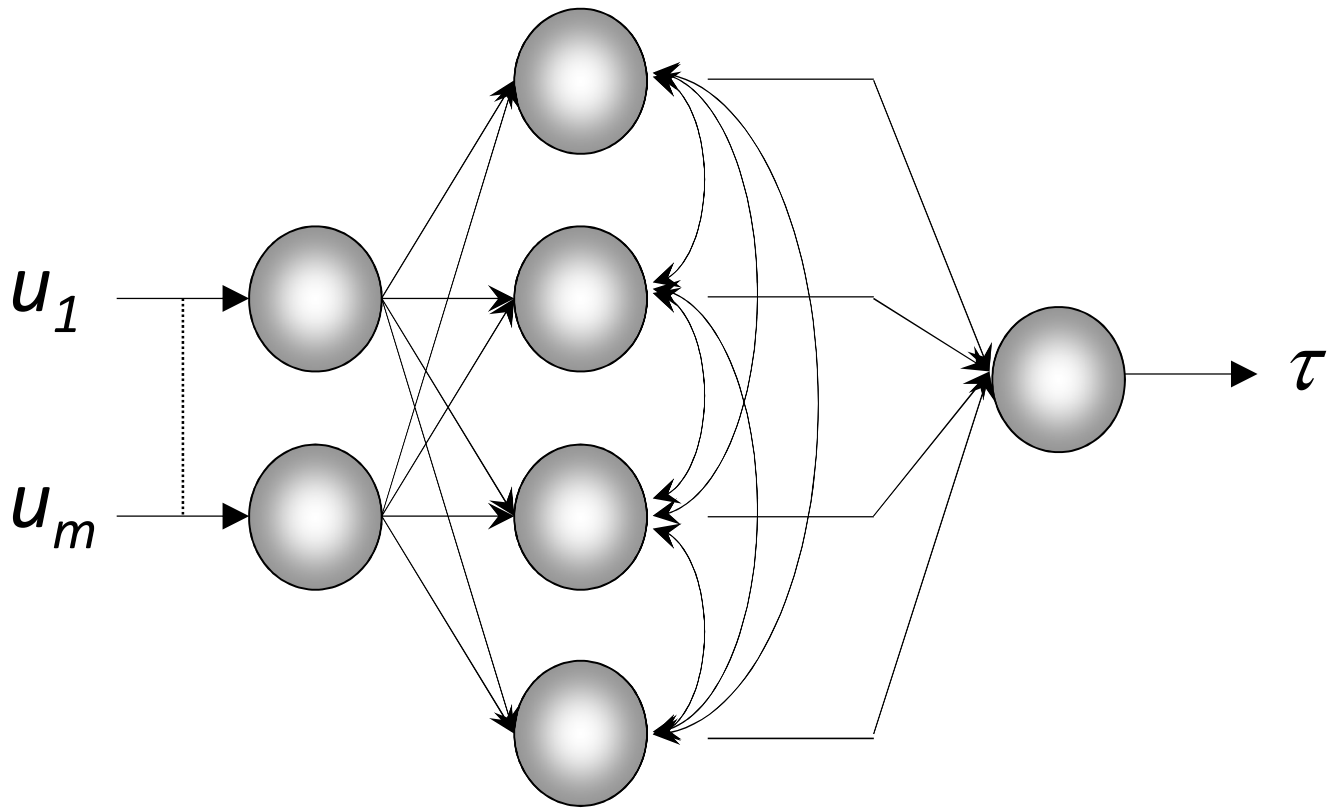
M.Ö. Efe and C. Kasnakoğlu, "A Comparison of Architectural Varieties in Radial Basis Function Neural Networks," World Congress on Computational Intelligence (WCCI'08) June 1-6, Hong Kong, pp.66-71, 2008.

Architectural Varieties



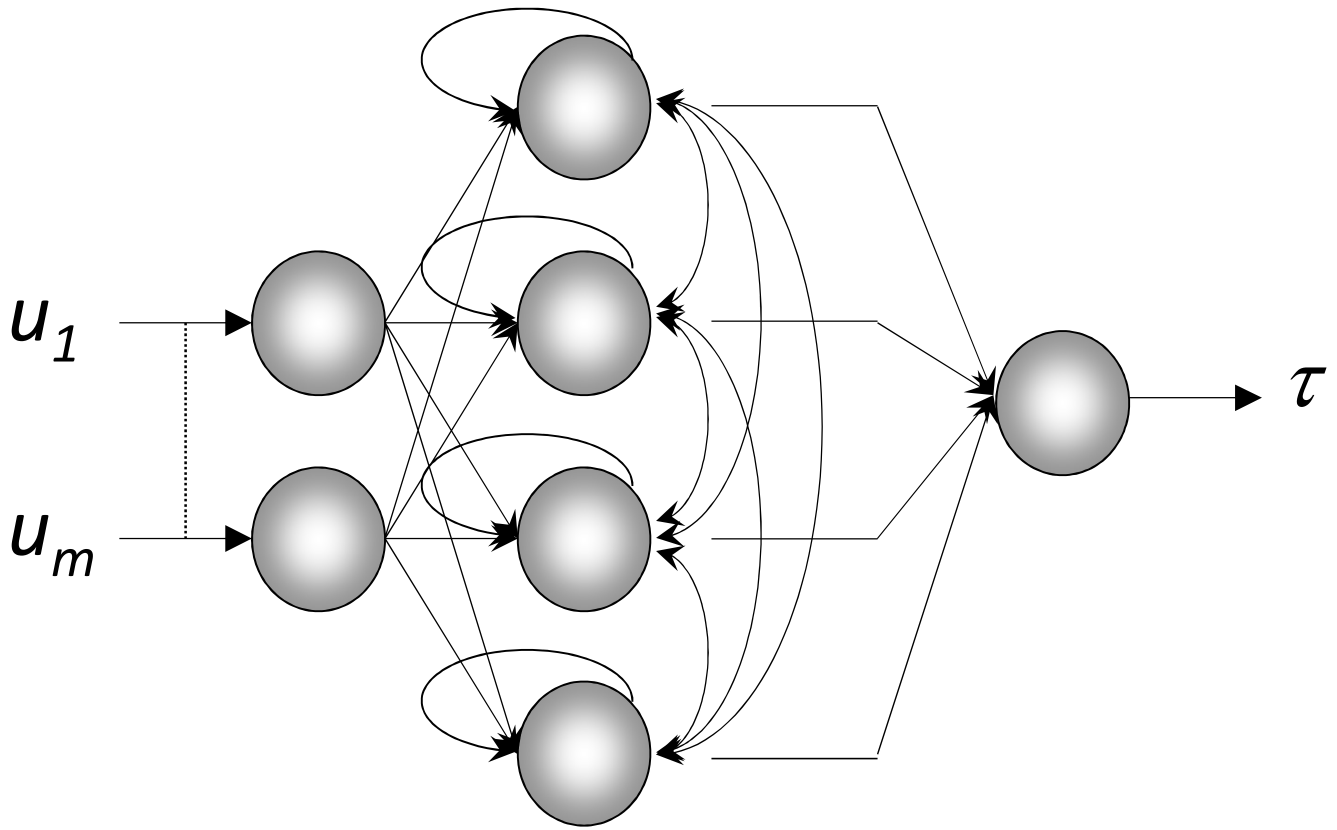
M.Ö. Efe and C. Kasnakoğlu, "A Comparison of Architectural Varieties in Radial Basis Function Neural Networks," World Congress on Computational Intelligence (WCCI'08) June 1-6, Hong Kong, pp.66-71, 2008.

Architectural Varieties



M.Ö. Efe and C. Kasnakoğlu, "A Comparison of Architectural Varieties in Radial Basis Function Neural Networks," World Congress on Computational Intelligence (WCCI'08) June 1-6, Hong Kong, pp.66-71, 2008.

Architectural Varieties



M.Ö. Efe and C. Kasnakoğlu, "A Comparison of Architectural Varieties in Radial Basis Function Neural Networks," World Congress on Computational Intelligence (WCCI'08) June 1-6, Hong Kong, pp.66-71, 2008.

Radial Basis Function Neural Nets

Questions & Answers

- Which cases are suitable for MLP and which are for RBFNN?
  Quite speculative! Try and see. This heavily depends on what you are trying to do, or in other words, it depends on what sort of a data you are trying to teach.

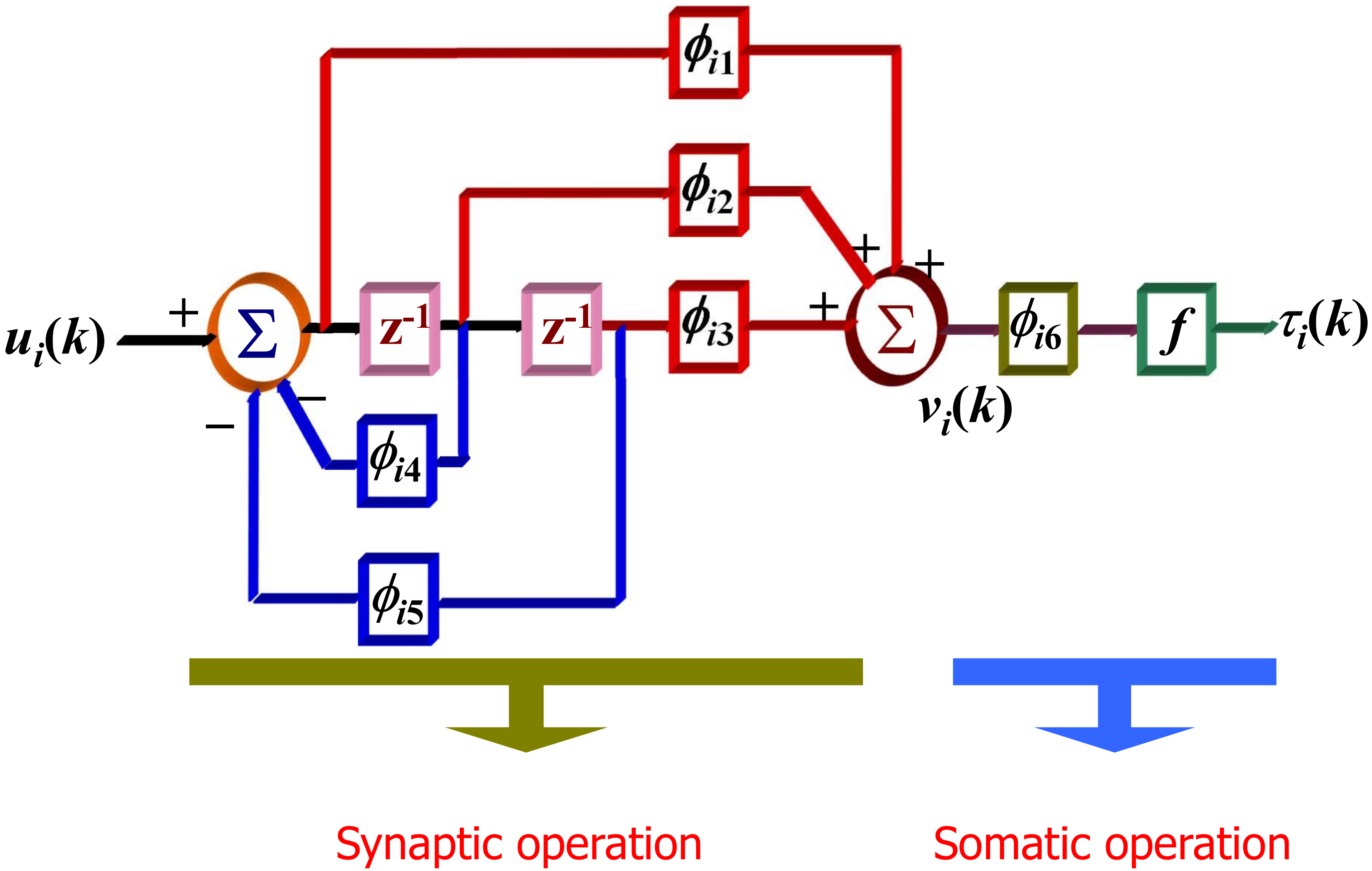
- Can I use momentum term and learning rate adaptation with RBFNN?
  Yes

- Can I have more than one hidden layer?
  Typical RBFNN does not have more than one hidden layer.

- Can I use other types of radial basis functions for activation?
  Yes, as long as they are **radial** basis functions...

Dynamic Neural Networks

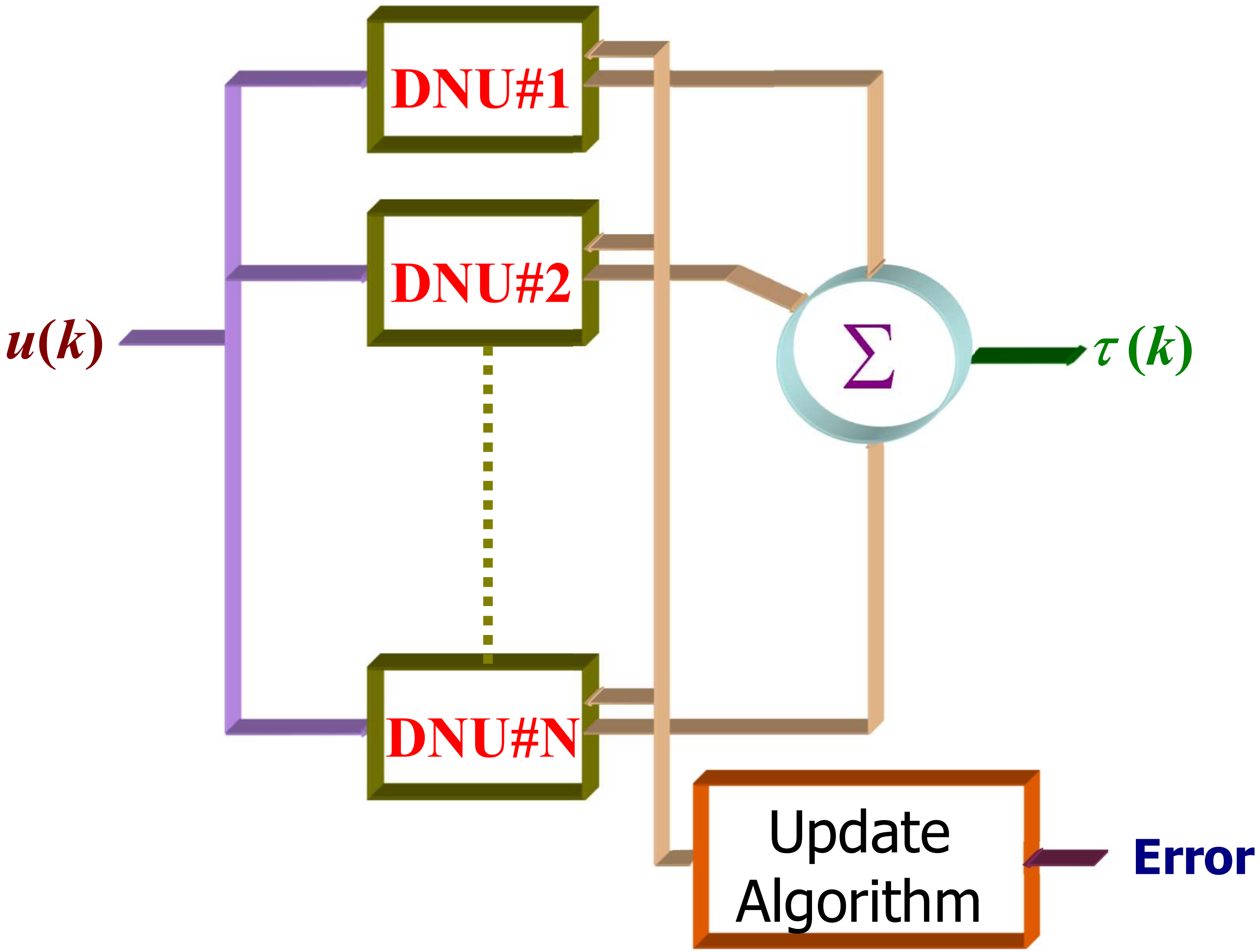
A Single Neuron



Mehmet Önder Efe, *Neural Networks*, Lecture Notes, 2022.

Dynamic Neural Networks

A Networked Structure



Mehmet Önder Efe, *Neural Networks*, Lecture Notes, 2022.

Dynamic Neural Networks

Functional Relationship

$$v_i(k) = \phi_{i1}u_i(k) + \phi_{i2}u_i(k-1) + \phi_{i3}u_i(k-2) - \phi_{i4}v_i(k-1) - \phi_{i5}v_i(k-2)$$

→ Synaptic sum

$$o_i(k) = f\left(\phi_{i6}v_i(k)\right)$$

→ Single neuron output

$$\tau(k) = \sum_{i=1}^{N} o_i(k)$$

→ Network output

- Adjustable parameters are $\phi_{i,1\ldots6}$ for each dynamic neuron
- Parameter update strategy for DNN structure is EBP technique
- This is a recurrent network structure!

Dynamic Neural Networks
Parameter Adjustment with Gradient Descent

$$J=\frac{1}{2}\left(d(k)-\tau(k)\right)^2=\frac{1}{2}e(k)^2$$

$$\tau(k)=\sum_{i=1}^{N}o_i(k)$$

$$o_i(k)=f\left(\phi_{i6}v_i(k)\right)$$

$$\Delta\phi_{ij}=-\eta\frac{\partial J}{\partial\phi_{ij}}=\eta e(k)\frac{\partial\tau(k)}{\partial\phi_{ij}(k)}=\eta e(k)\frac{\partial\tau(k)}{\partial o_i(k)}\frac{\partial o_i(k)}{\partial v_i(k)}\frac{\partial v_i(k)}{\partial\phi_{ij}(k)}$$

$$=\eta e(k)\phi_{i6}(k)f'\left(\phi_{i6}(k)v_i(k)\right)\frac{\partial v_i(k)}{\partial\phi_{ij}(k)}$$

for $j=1,2,\ldots,5$

$$v_i(k)=\phi_{i1}u_i(k)+\phi_{i2}u_i(k-1)+\phi_{i3}u_i(k-2)-\phi_{i4}v_i(k-1)-\phi_{i5}v_i(k-2)$$

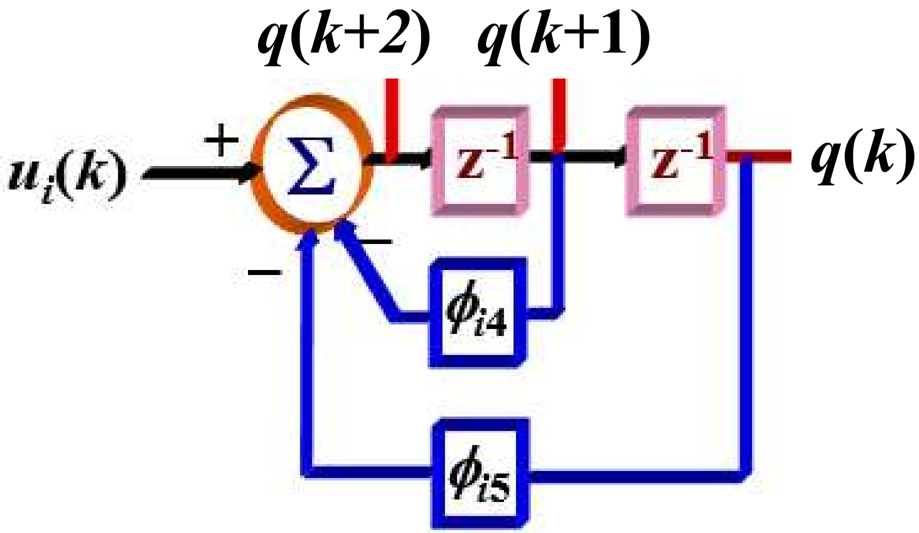
$$\Delta\phi_{i6}=-\eta\frac{\partial J}{\partial\phi_{i6}}=\eta e(k)\frac{\partial\tau(k)}{\partial\phi_{i6}(k)}=\eta e(k)\frac{\partial\tau(k)}{\partial o_i(k)}\frac{\partial o_i(k)}{\partial\phi_{i6}(k)}=\eta e(k)v_i(k)f'\left(\phi_{i6}(k)v_i(k)\right)$$

Dynamic Neural Networks

Parameter Adjustment and Stability

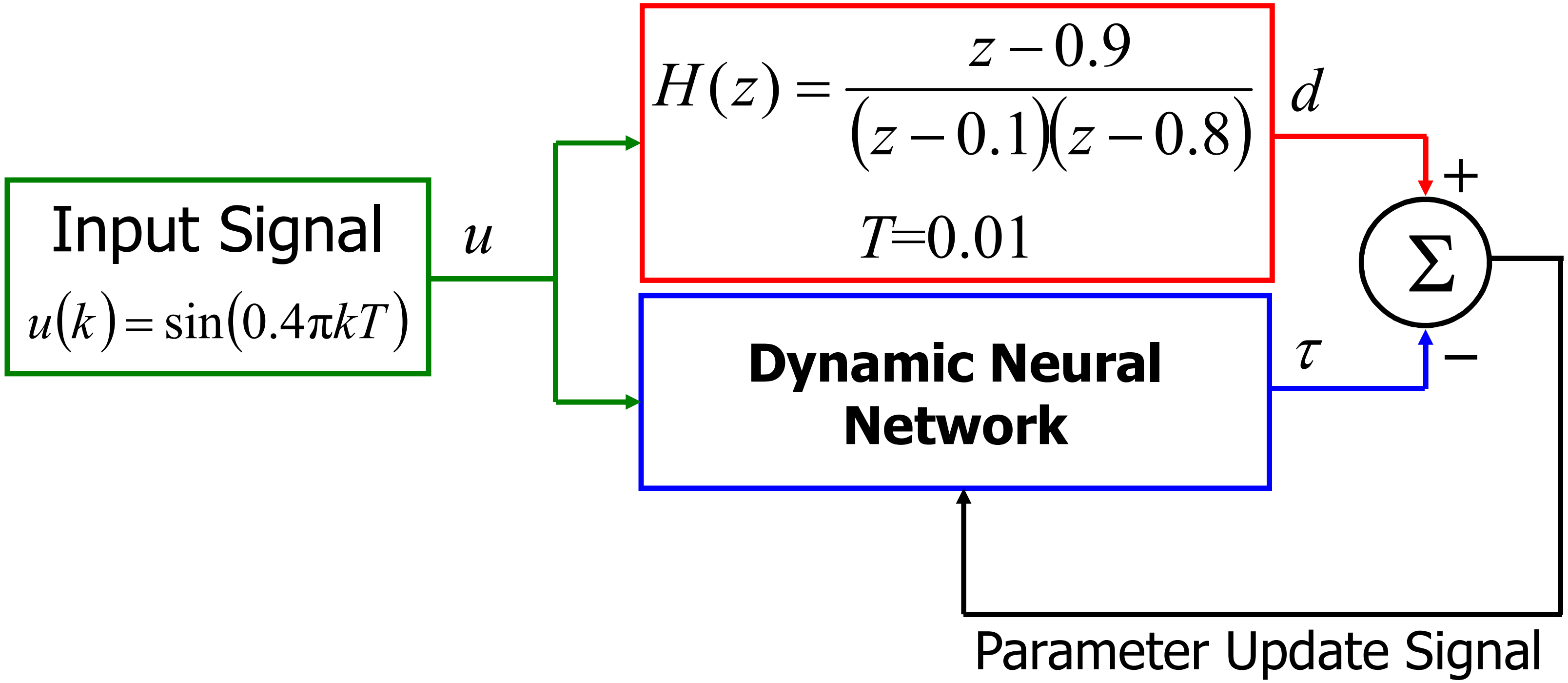
$$z^2 + \phi_{i4} z + \phi_{i5} = 0 \Rightarrow \left| z_{1,2} \right| < 1$$

$$\frac{q(z)}{u_i(z)} = \frac{1}{z^2 + \phi_{i4} z + \phi_{i5}}$$



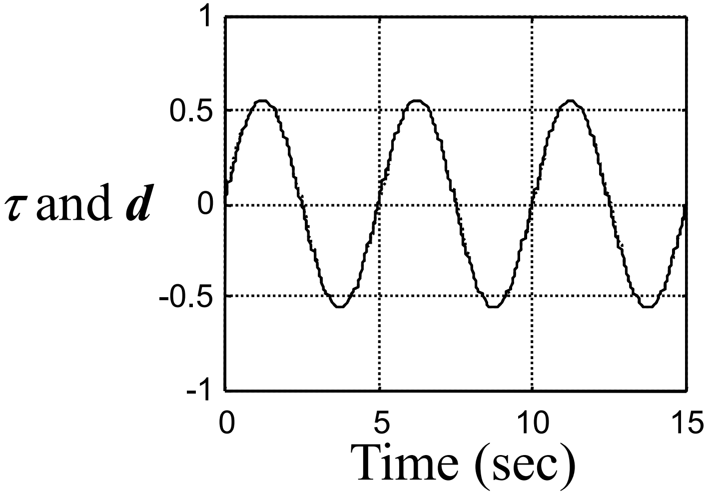
Dynamic Neural Networks

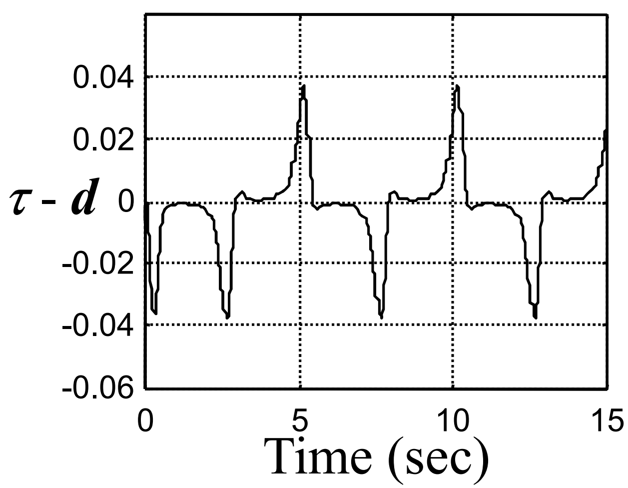
An Identification Example

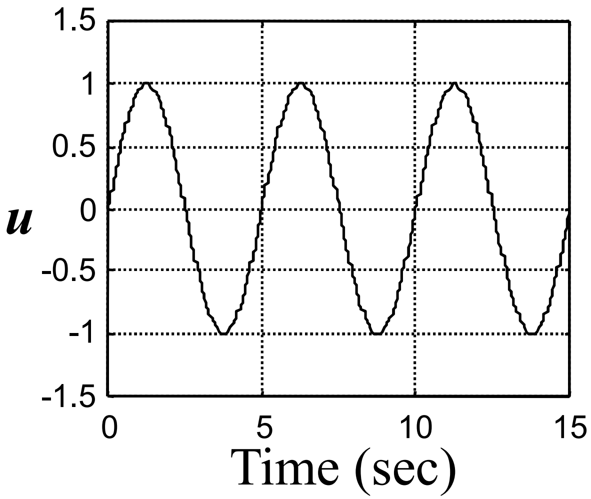


- Notice that the tuning here is online
- The above system is an identification system
- In an identification system, the input must be persistently exciting

Dynamic Neural Networks
An Identification Example





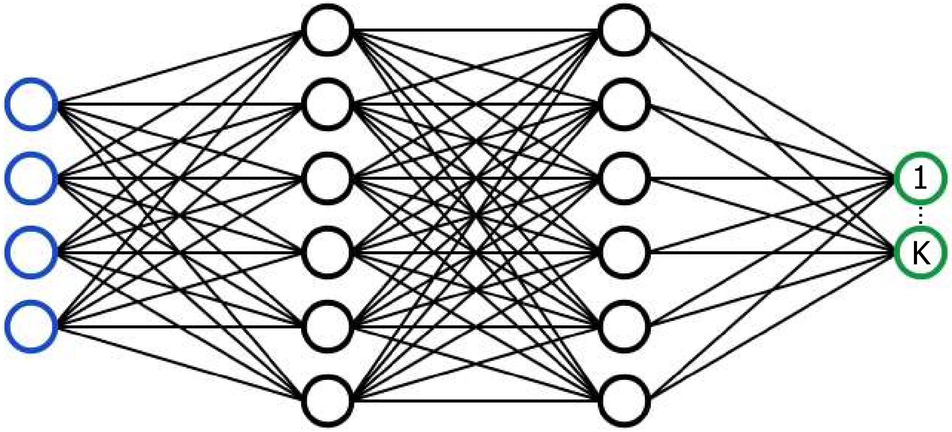


- Initial parameters are from [0.1 , 0.4]
- N=5

- What about stability in synaptic parts?

Second Order Training Schemes

Levenberg-Marquardt Algorithm



$$\overline{J}(\mathbf{w}) = \sum_{p=1}^{P}\left[\sum_{k=1}^{K}\left(d_k^p - \tau_k^p\right)^2\right]$$

	d_1	d_2	...	d_K
1	0.1	0.5	...	0.6
2	0.3	-0.1	...	0.2
3	-0.2	**-0.7**	...	-0.3
P	-1	0	...	0.4

τ_1	τ_2	...	τ_K
0.12	0.51	...	0.61
0.31	-0.1	...	0.23
-0.22	**-0.7**	...	-0.39
-0.1	0.2	...	0.48

e_2^3

Mehmet Önder Efe, *Neural Networks*, Lecture Notes, 2022.

Second Order Training Schemes

Levenberg-Marquardt Algorithm

$$\bar{J}(\mathbf{w}) = \sum_{\mathrm{p}=1}^{\mathrm{P}} \left[\sum_{\mathrm{k}=1}^{\mathrm{K}} \left(d_k^p - \tau_k^p \right)^2 \right] \quad \Longleftrightarrow \quad \bar{J}(\mathbf{w}) = E^{\mathrm{T}} E$$

$$E = \left[e_1^1 \quad \ldots \quad e_{\mathrm{K}}^1 \quad e_1^2 \quad \ldots \quad e_{\mathrm{K}}^2 \quad \ldots \quad e_1^{\mathrm{P}} \quad \ldots \quad e_{\mathrm{K}}^{\mathrm{P}} \right]^{\mathrm{T}}$$

For the 1st pattern | **For the 2nd pattern** | **For the P-th pattern**

$$\mathrm{e}_k^p = d_k^p - \tau_k^p, \quad k = 1, \cdots, \mathrm{K}, \quad p = 1, \cdots, \mathrm{P}$$

$$J = \begin{bmatrix} \dfrac{\partial e_1^1}{\partial w_1} & \dfrac{\partial e_1^1}{\partial w_2} & \cdots & \dfrac{\partial e_1^1}{\partial w_N} \\ \dfrac{\partial e_2^1}{\partial w_1} & \dfrac{\partial e_2^1}{\partial w_2} & \cdots & \dfrac{\partial e_2^1}{\partial w_N} \\ \vdots & \vdots & & \vdots \\ \dfrac{\partial e_K^1}{\partial w_1} & \dfrac{\partial e_K^1}{\partial w_2} & \cdots & \dfrac{\partial e_K^1}{\partial w_N} \\ \vdots & \vdots & & \vdots \\ \dfrac{\partial e_1^P}{\partial w_1} & \dfrac{\partial e_1^P}{\partial w_2} & \cdots & \dfrac{\partial e_1^P}{\partial w_N} \\ \dfrac{\partial e_2^P}{\partial w_1} & \dfrac{\partial e_2^P}{\partial w_2} & \cdots & \dfrac{\partial e_2^P}{\partial w_N} \\ \vdots & \vdots & & \vdots \\ \dfrac{\partial e_K^P}{\partial w_1} & \dfrac{\partial e_K^P}{\partial w_2} & \cdots & \dfrac{\partial e_K^P}{\partial w_N} \end{bmatrix}$$

$$\mathbf{w} = [w_1 \quad w_2 \quad \cdots \quad w_N]^T$$

Sub-block for the first pattern

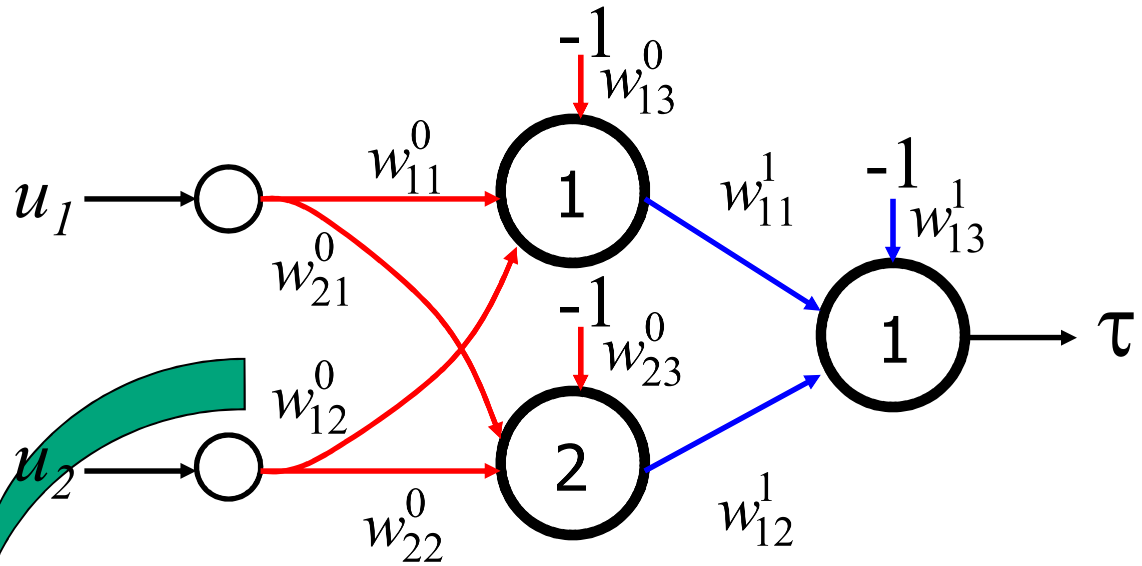
$$e_k^p = d_k^p - \tau_k^p$$
$$k = 1, \cdots, K,$$
$$p = 1, \cdots, P$$

One epoch!

Sub-block for the last pattern

Second Order Training Schemes

Levenberg-Marquardt Algorithm

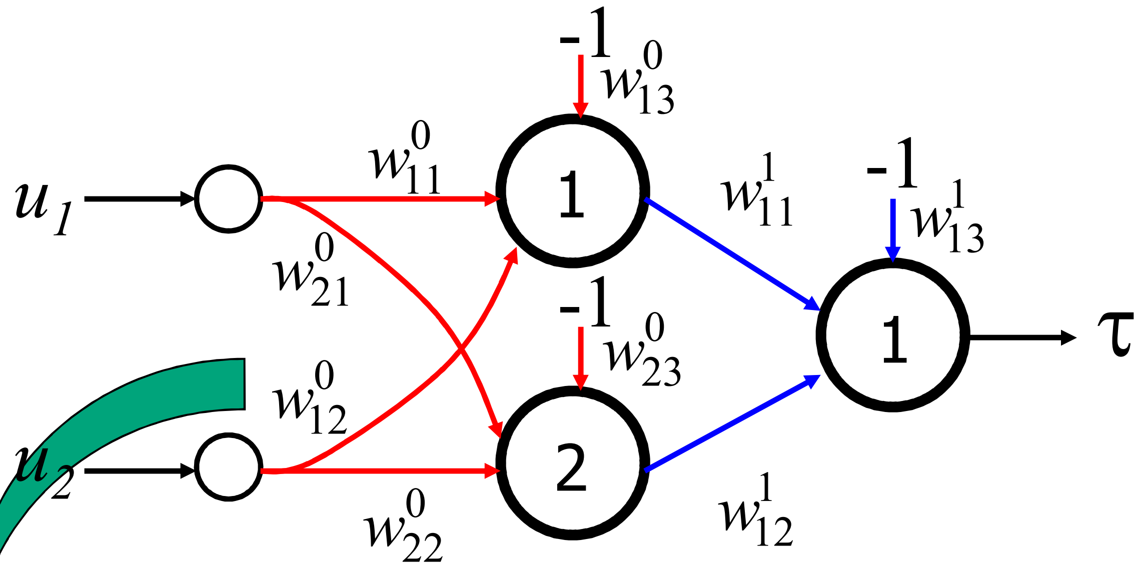


Rearrange the weight **matrices** in the network in a new **vector** $\mathbf{w}$

$$\mathbf{w} = [w_1 \quad w_2 \quad \cdots \quad w_N]^{\mathrm{T}}$$

Second Order Training Schemes

Levenberg-Marquardt Algorithm



Rearrange the weight **matrices** in the network in a new **vector** $\mathbf{w}$

$$\mathbf{w} = [w_1 \quad w_2 \quad w_3 \quad w_4 \quad w_5 \quad w_6 \quad w_7 \quad w_8 \quad w_9]^T$$

$$\mathbf{w} = [w_{11}^0 \quad w_{12}^0 \quad w_{13}^0 \quad w_{21}^0 \quad w_{22}^0 \quad w_{23}^0 \quad w_{11}^1 \quad w_{12}^1 \quad w_{13}^1]^T$$

Mehmet Önder Efe, *Neural Networks*, Lecture Notes, 2022.

Second Order Training Schemes

Levenberg-Marquardt Algorithm

```
FOR t=1:10
 FOR p=1:P
  FOR k=1:K
   FOR i=1:N
        Compute ∂e_k^p/∂w_i
   END
   % One row of J is ready
  END
  % One sub-block of J is ready
 END
 % J is ready, update now!

 w_{t+1} = w_t − (J_t^T J_t + μ_t I)^{-1} J_t^T E_t

END
```

Compute $\dfrac{\partial e_k^p}{\partial w_i}$

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \left(J_t^T J_t + \mu_t I\right)^{-1} J_t^T E_t$$

$$J = \begin{bmatrix} \dfrac{\partial e_1^1}{\partial w_1} & \dfrac{\partial e_1^1}{\partial w_2} & \cdots & \dfrac{\partial e_1^1}{\partial w_N} \\ \dfrac{\partial e_2^1}{\partial w_1} & \dfrac{\partial e_2^1}{\partial w_2} & \cdots & \dfrac{\partial e_2^1}{\partial w_N} \\ \vdots & \vdots & & \vdots \\ \dfrac{\partial e_K^1}{\partial w_1} & \dfrac{\partial e_K^1}{\partial w_2} & \cdots & \dfrac{\partial e_K^1}{\partial w_N} \\ \vdots & \vdots & & \vdots \\ \dfrac{\partial e_1^P}{\partial w_1} & \dfrac{\partial e_1^P}{\partial w_2} & \cdots & \dfrac{\partial e_1^P}{\partial w_N} \\ \dfrac{\partial e_2^P}{\partial w_1} & \dfrac{\partial e_2^P}{\partial w_2} & \cdots & \dfrac{\partial e_2^P}{\partial w_N} \\ \vdots & \vdots & & \vdots \\ \dfrac{\partial e_K^P}{\partial w_1} & \dfrac{\partial e_K^P}{\partial w_2} & \cdots & \dfrac{\partial e_K^P}{\partial w_N} \end{bmatrix}$$

Second Order Training Schemes

Levenberg-Marquardt Algorithm

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \left(J_t^{\mathrm{T}} J_t + \mu_t I\right)^{-1} J_t^{\mathrm{T}} E_t$$

- $\mu > 0$ is the stepsize
- If $\mu = 0$, we get Gauss-Newton algorithm
- If μ is too large, we get standard EBP with learning rate $\approx 1/\mu$

Therefore, LM algorithm is a smooth transition between Gauss-Newton algorithm and EBP with the advantage of

- Removing the slow convergence of EBP
- Removing the invertibility problem in Gauss-Newton

Second Order Training Schemes

A Source Code-LM

```
clear all;close all;clc
NETINDIM    = 2;
HIDNEURONS = 4;
NETOUTDIM  = 1;

P = [0 0
     0 1
     1 0
     1 1];

D = [0
     1
     1
     0];

% Determine th range of the data
PR = [min(P)'  max(P)'];
% Form the network
net = newff(PR,[HIDNEURONS NETOUTDIM],{'tansig' 'purelin'});
```

```
% Loop for 10 epoches
net.trainParam.epochs    = 10;
net.trainParam.mem_reduc = 1;

% Show after every iteration
net.trainParam.show      = 1;

% Train the network
net = train(net,P',D');

% Print the results on the screen
Tau = sim(net,P')

% Pront the error E
E = D'-Tau

% Save your network weights etc.
save network.mat net
```

Second Order Training Schemes

A Numerical Example – The Problem

- Process — $y=ax+b$
- Input — x
- Output — y
- Available Data

Pair no	x	y
1	0	1
2	1	2
3	2	3

- NN Model — $y_n=w_1x+w_2$
- Initial Conditions — $w_1(0)=0.1,\ w_2(0)=0.2$

Second Order Training Schemes

A Numerical Example –Jacobian

x y

$$yn^1 = 0.1 * 0 + 0.2 = 0.2 \qquad e^1 = 1 - yn^1 = 0.8$$

$$yn^2 = 0.1 * 1 + 0.2 = 0.3 \qquad e^2 = 2 - yn^2 = 1.7$$

$$yn^3 = 0.1 * 2 + 0.2 = 0.4 \qquad e^3 = 3 - yn^3 = 2.6$$

$$\frac{\partial e^1}{\partial w_1} = -x^1 = 0 \qquad \frac{\partial e^1}{\partial w_2} = -1$$

$$\frac{\partial e^2}{\partial w_1} = -x^2 = -1 \qquad \frac{\partial e^2}{\partial w_2} = -1$$

$$\frac{\partial e^3}{\partial w_1} = -x^3 = -2 \qquad \frac{\partial e^3}{\partial w_2} = -1$$

$$J = \begin{bmatrix} \frac{\partial e_1^1}{\partial w_1} & \frac{\partial e_1^1}{\partial w_2} & \cdots & \frac{\partial e_1^1}{\partial w_N} \\ \frac{\partial e_2^1}{\partial w_1} & \frac{\partial e_2^1}{\partial w_2} & \cdots & \frac{\partial e_2^1}{\partial w_N} \\ \vdots & \vdots & & \vdots \\ \frac{\partial e_K^1}{\partial w_1} & \frac{\partial e_K^1}{\partial w_2} & \cdots & \frac{\partial e_K^1}{\partial w_N} \\ \vdots & \vdots & & \vdots \\ \frac{\partial e_1^P}{\partial w_1} & \frac{\partial e_1^P}{\partial w_2} & \cdots & \frac{\partial e_1^P}{\partial w_N} \\ \frac{\partial e_2^P}{\partial w_1} & \frac{\partial e_2^P}{\partial w_2} & \cdots & \frac{\partial e_2^P}{\partial w_N} \\ \vdots & \vdots & & \vdots \\ \frac{\partial e_K^P}{\partial w_1} & \frac{\partial e_K^P}{\partial w_2} & \cdots & \frac{\partial e_K^P}{\partial w_N} \end{bmatrix}$$

Mehmet Önder Efe, *Neural Networks*, Lecture Notes, 2022.

Second Order Training Schemes

A Numerical Example –Jacobian

$$\frac{\partial e^1}{\partial w_1} = -x^1 = 0 \qquad \frac{\partial e^1}{\partial w_2} = -1$$

$$\frac{\partial e^2}{\partial w_1} = -x^2 = -1 \qquad \frac{\partial e^2}{\partial w_2} = -1$$

$$\frac{\partial e^3}{\partial w_1} = -x^3 = -2 \qquad \frac{\partial e^3}{\partial w_2} = -1$$

$$J = \begin{bmatrix} 0 & -1 \\ -1 & -1 \\ -2 & -1 \end{bmatrix} = \begin{bmatrix} -x & -1_{3\times 1} \end{bmatrix}$$

Second Order Training Schemes

A Numerical Example – The Code

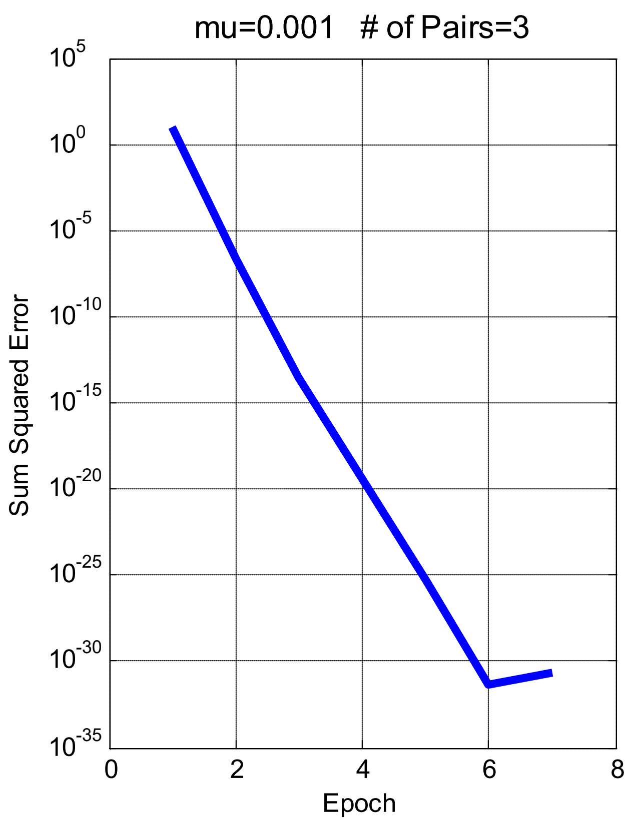
```
x=[0 1 2]';
y=[1 2 3]';
w=[0.1 0.2]';
m=1;

W=w;
for k=1:5
    yn=w(1)*x+w(2);
    E=y-yn;
    J=[-x -ones(size(x))];
    w=w-inv(m*eye(size(J'*J))+J'*J)*J'*E;
    cost(k)=(E'*E);
    W=[W w];
end
```

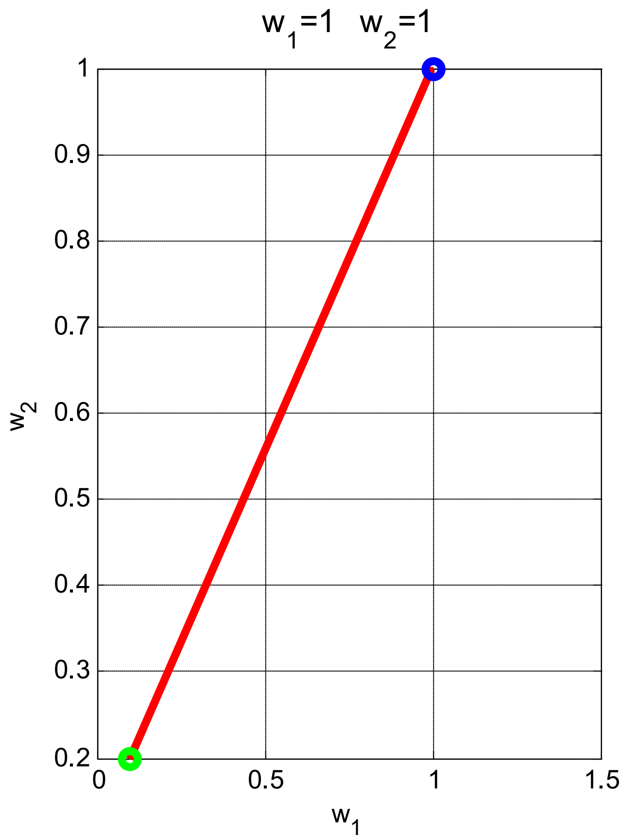
$$\mathbf{w}_{t+1} = \mathbf{w}_t - \left(J_t^T J_t + \mu_t I\right)^{-1} J_t^T E_t$$

$$J = \begin{bmatrix} -x & -1_{3\times 1} \end{bmatrix}$$

Second Order Training Schemes

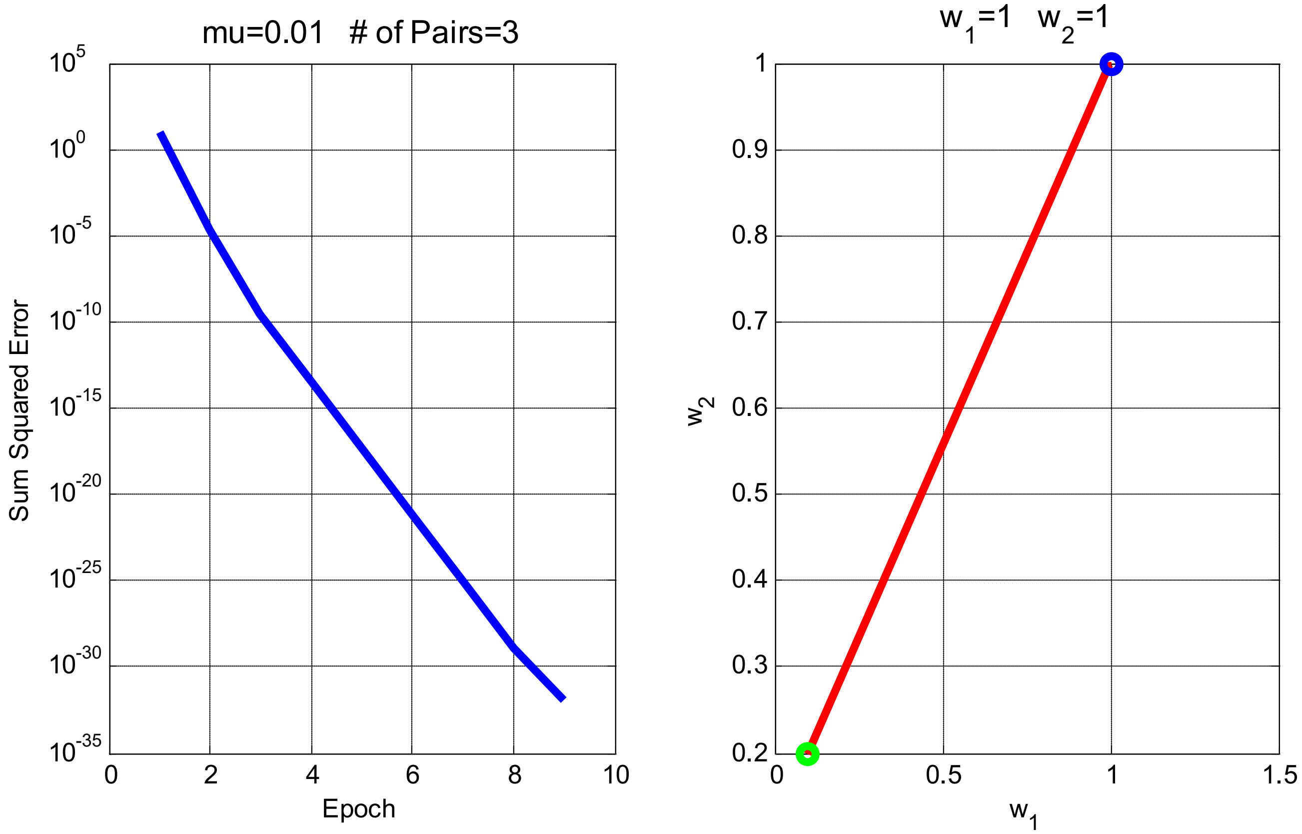
A Numerical Example – Results, mu=0.001





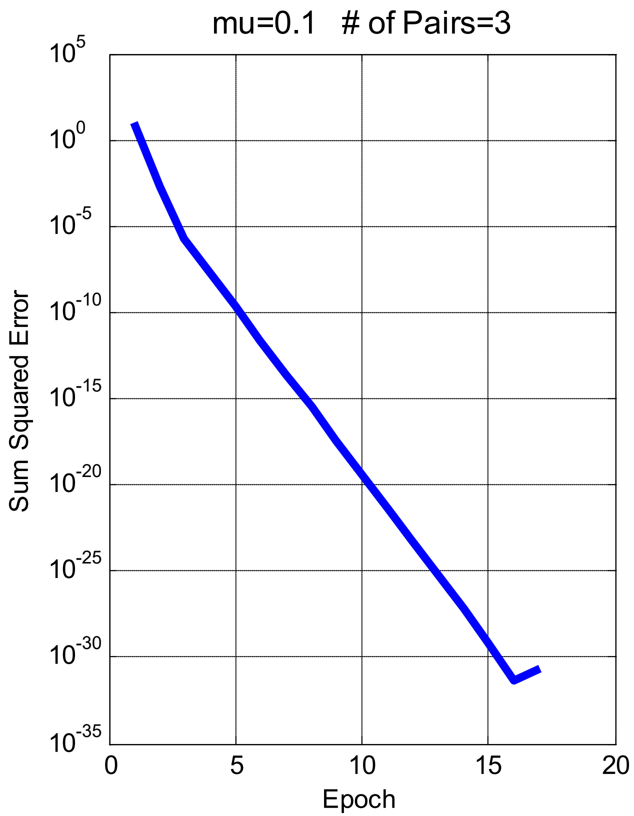
Second Order Training Schemes

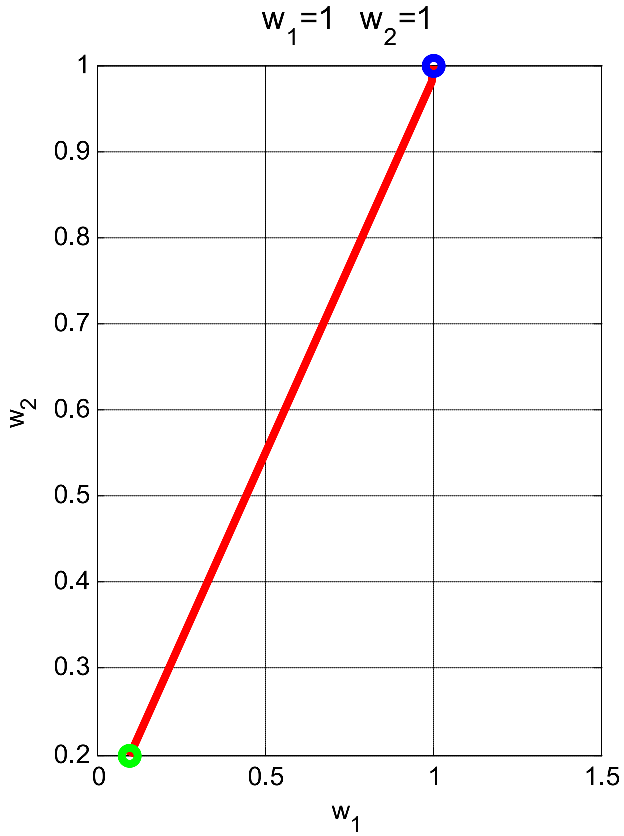
A Numerical Example – Results, mu=0.01



Second Order Training Schemes

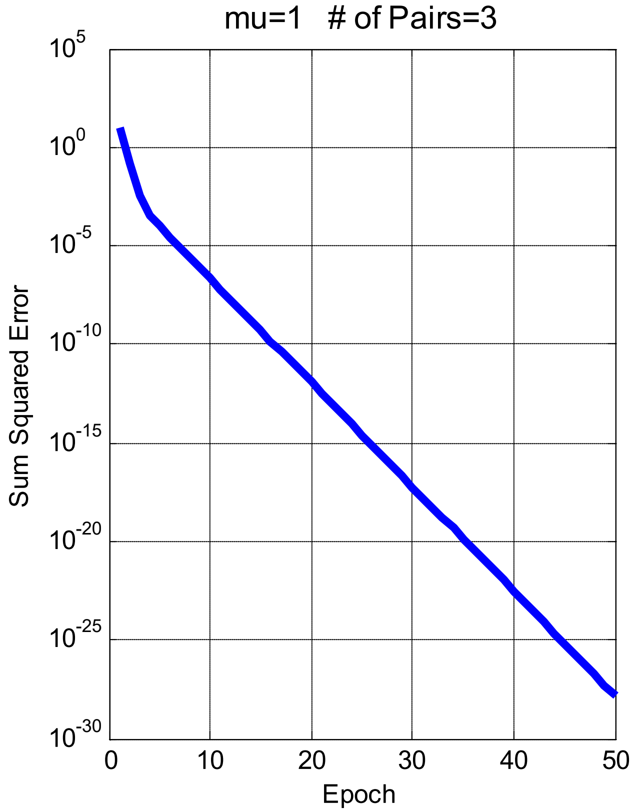
A Numerical Example – Results, mu=0.1

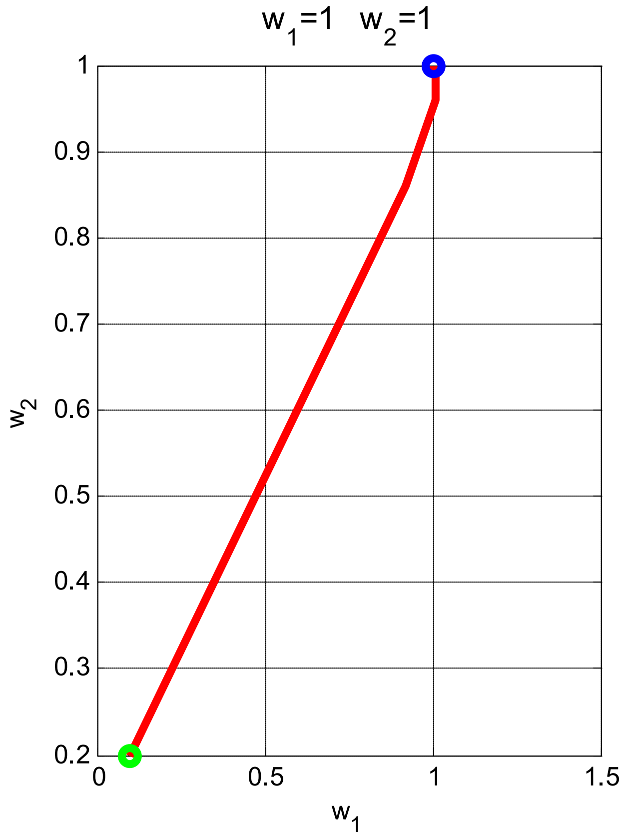




Second Order Training Schemes

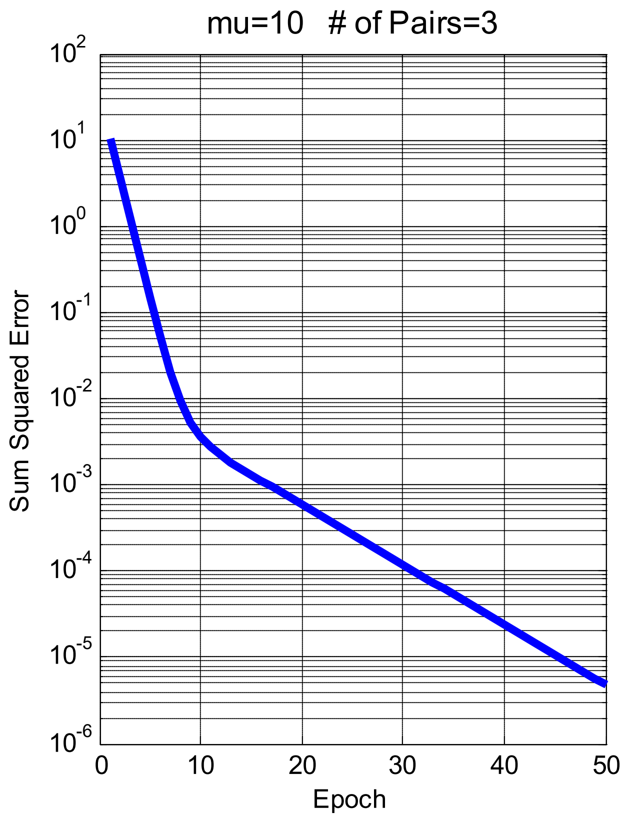
A Numerical Example – Results, mu=1

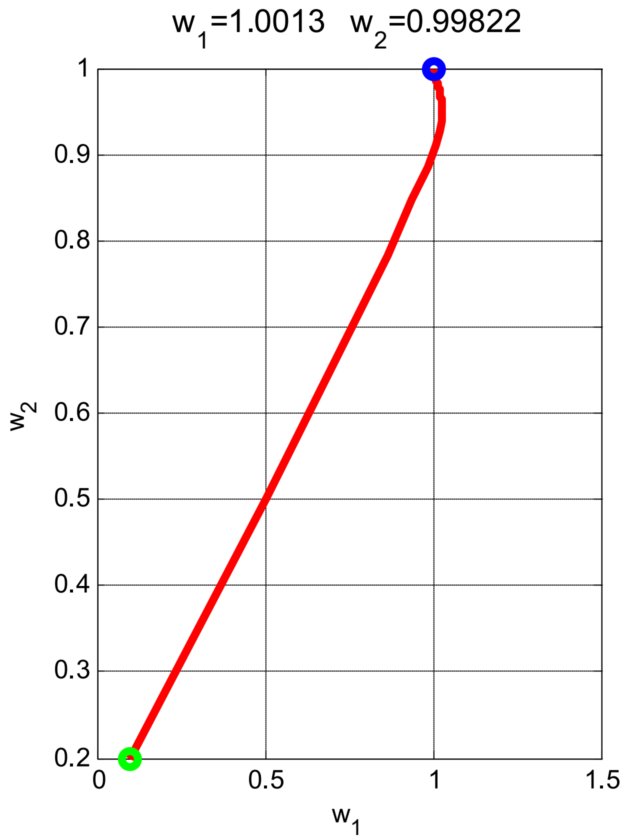




Second Order Training Schemes

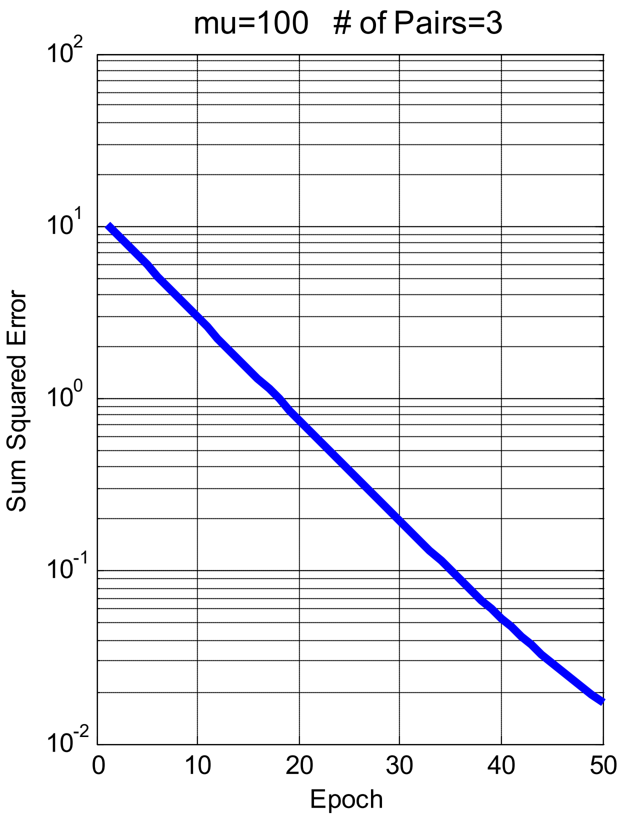
A Numerical Example – Results, mu=10

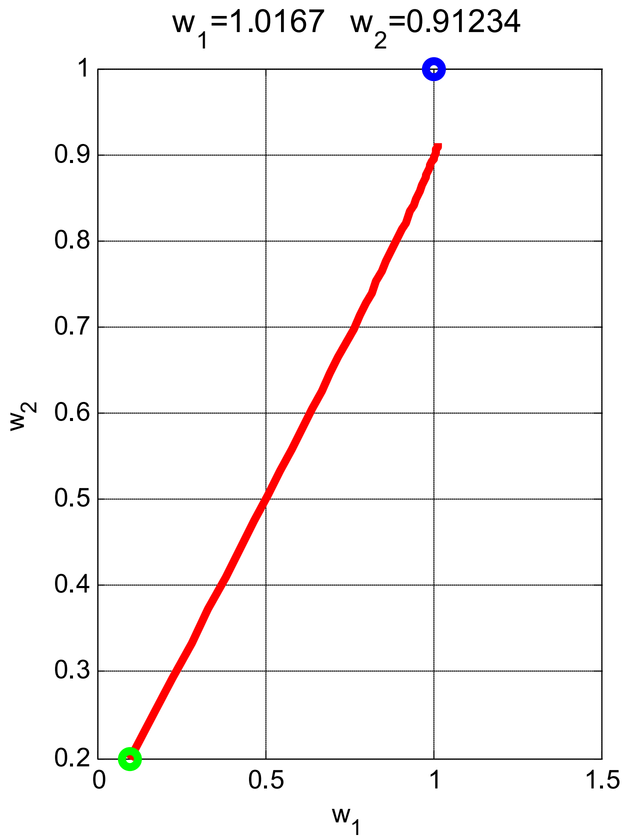




Second Order Training Schemes

A Numerical Example – Results, mu=100





Second Order Training Schemes

A Numerical Example – What happens with P=2?

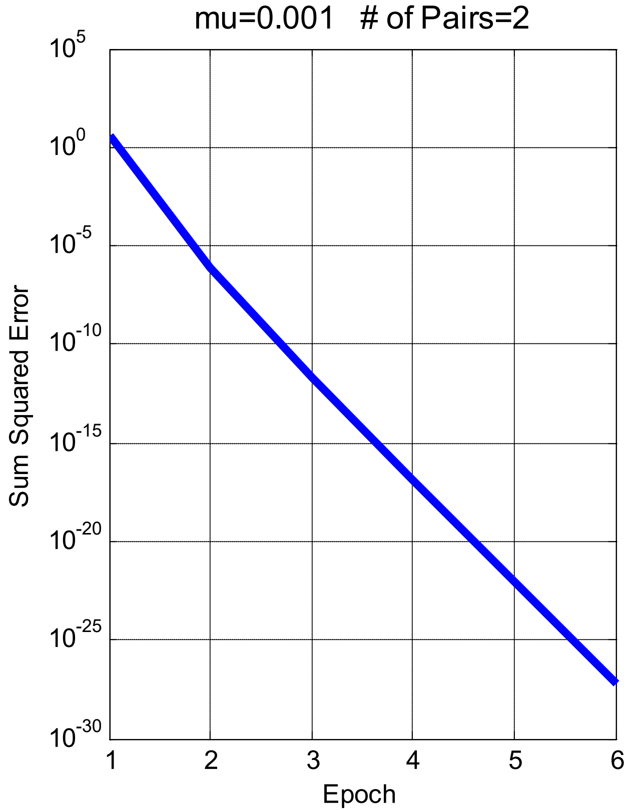
- Process $y=ax+b$
- Input x
- Output y
- Available Data

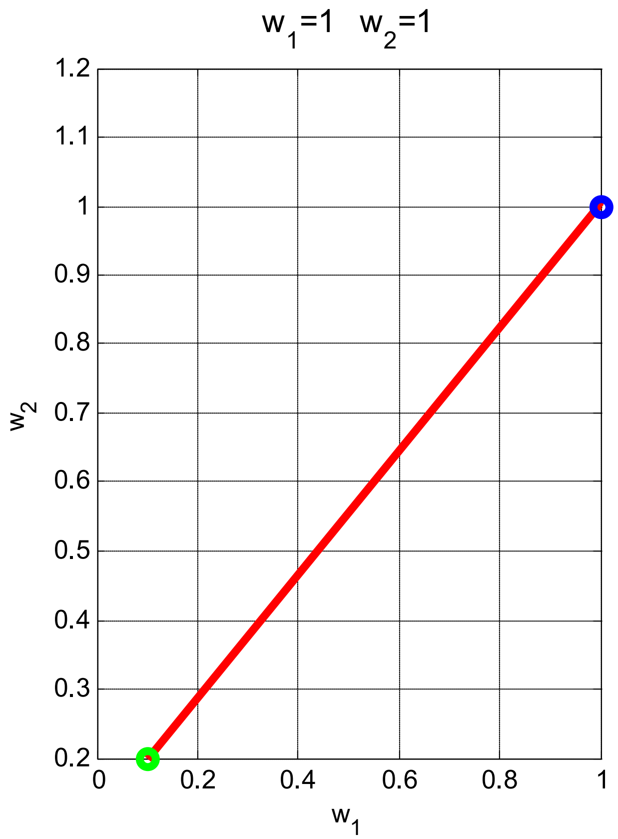
Pair no	x	y
1	0	1
2	1	2
~~3~~	~~2~~	~~3~~

- NN Model $y_n=w_1x+w_2$
- Initial Conditions $w_1(0)=0.1,\ w_2(0)=0.2$

Second Order Training Schemes

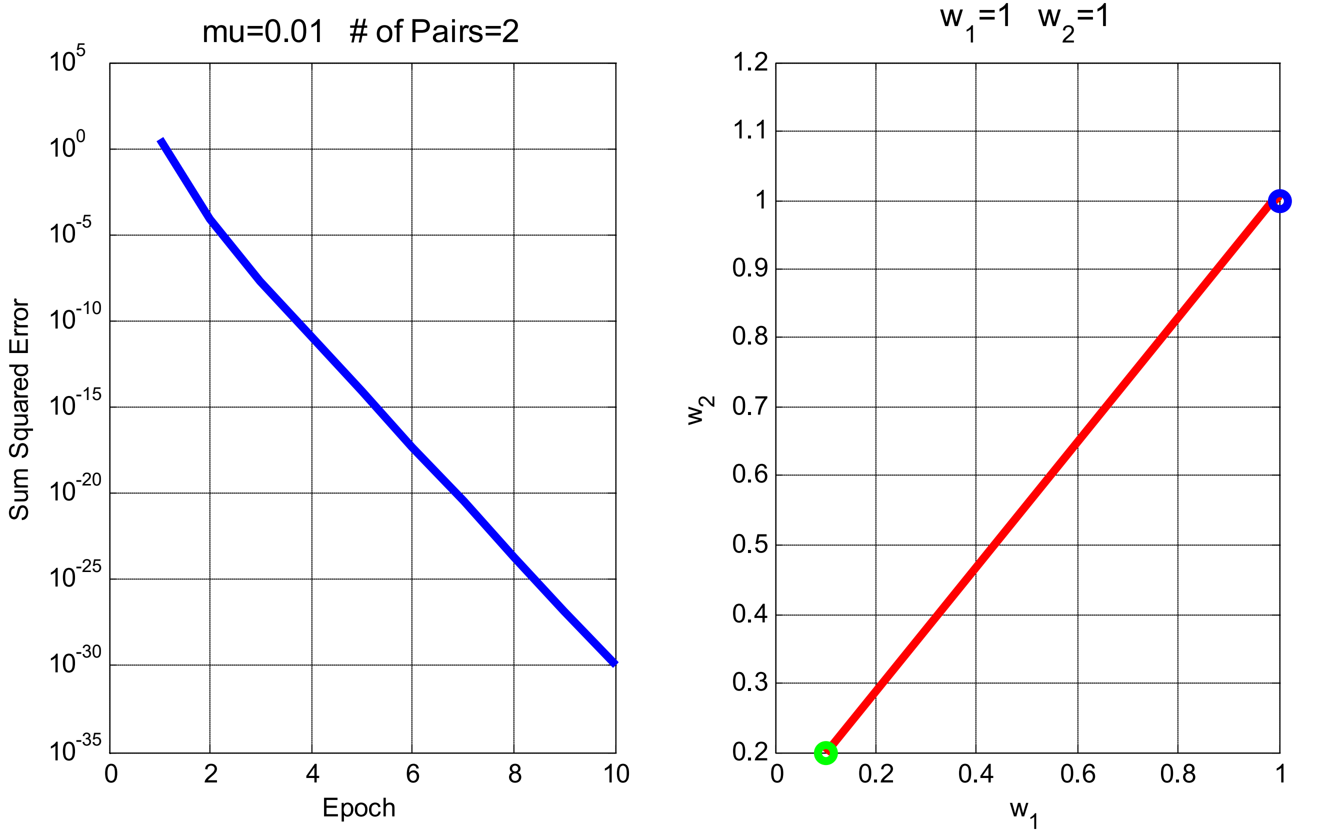
A Numerical Example – What happens with P=2?





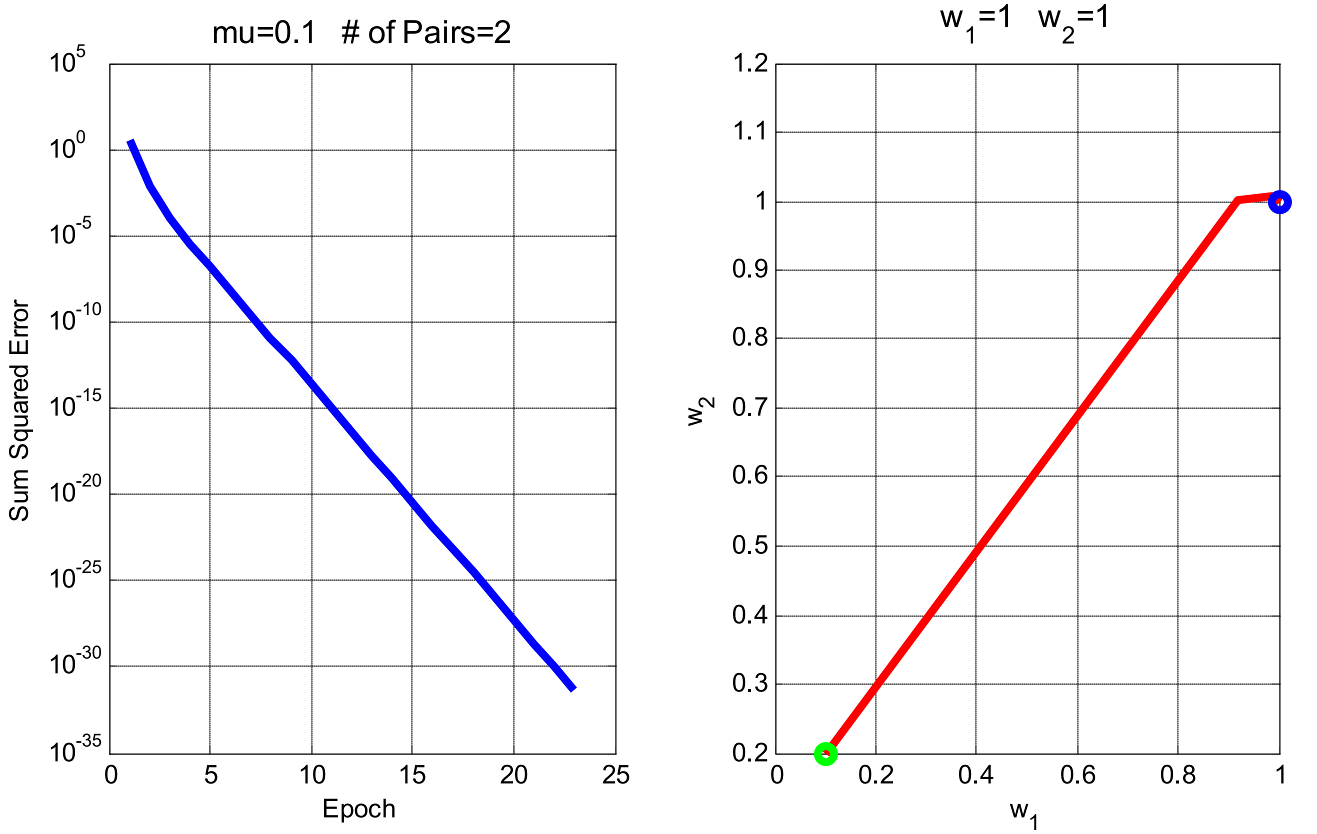
Second Order Training Schemes

A Numerical Example – What happens with P=2?



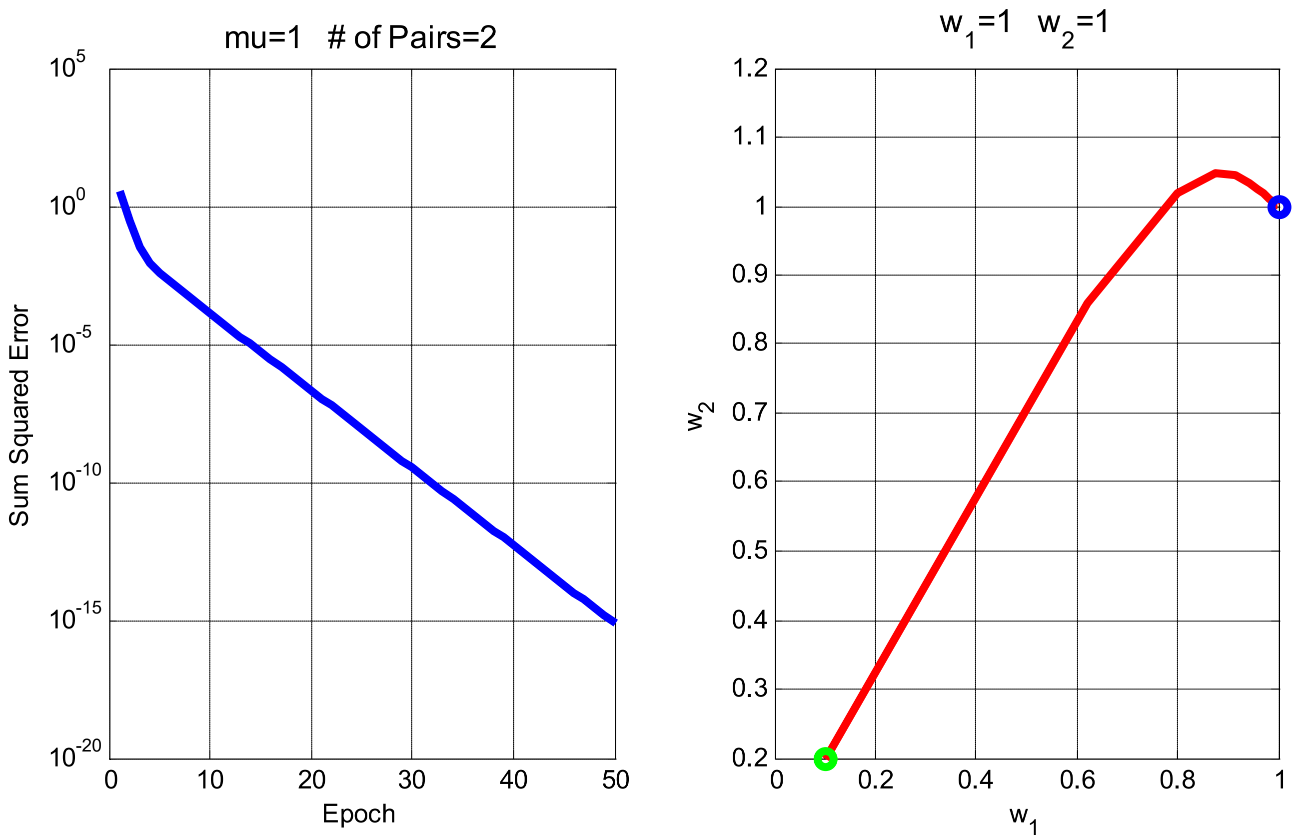
Second Order Training Schemes

A Numerical Example – What happens with P=2?



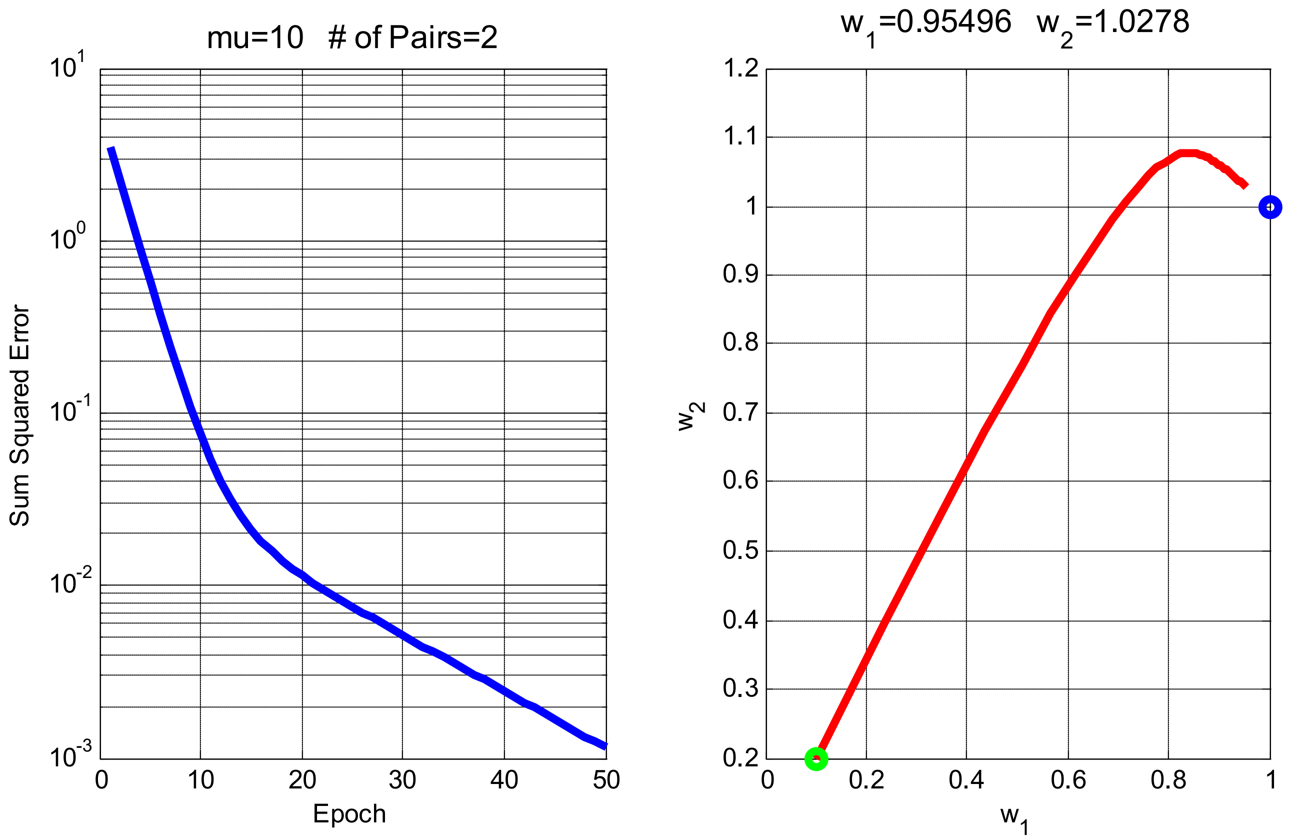
Second Order Training Schemes

A Numerical Example – What happens with P=2?



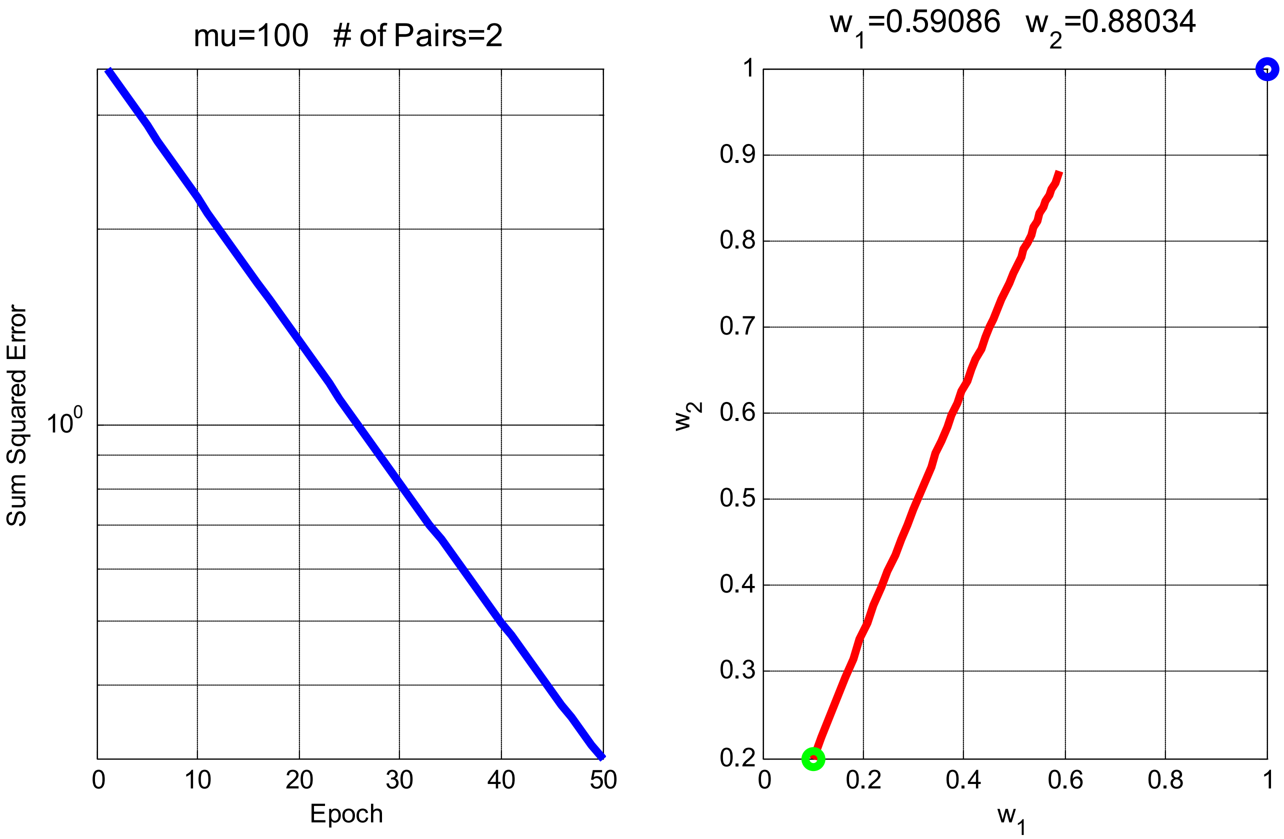
Second Order Training Schemes

A Numerical Example – What happens with P=2?



Second Order Training Schemes

A Numerical Example – What happens with P=2?



Second Order Training Schemes

A Numerical Example – What happens with P=1? The Extreme Case

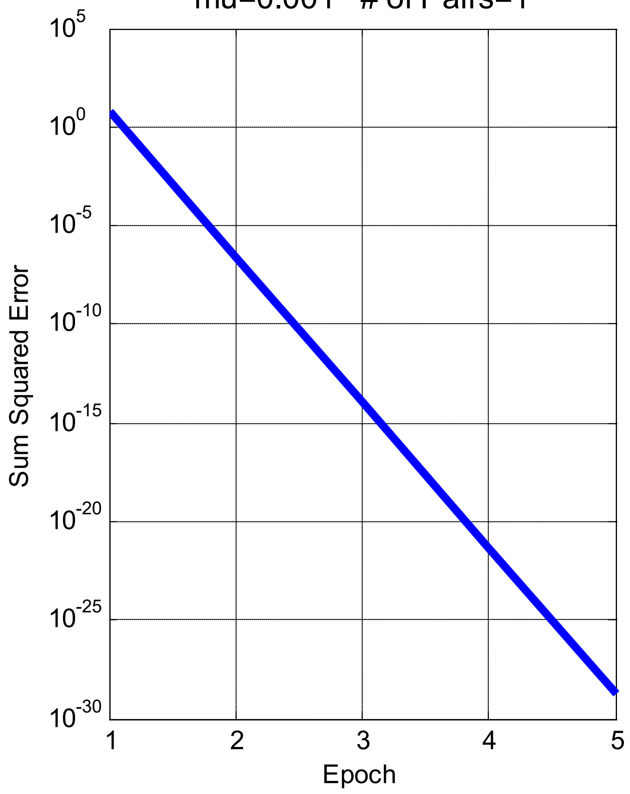
- Process $y=ax+b$
- Input x
- Output y
- Available Data

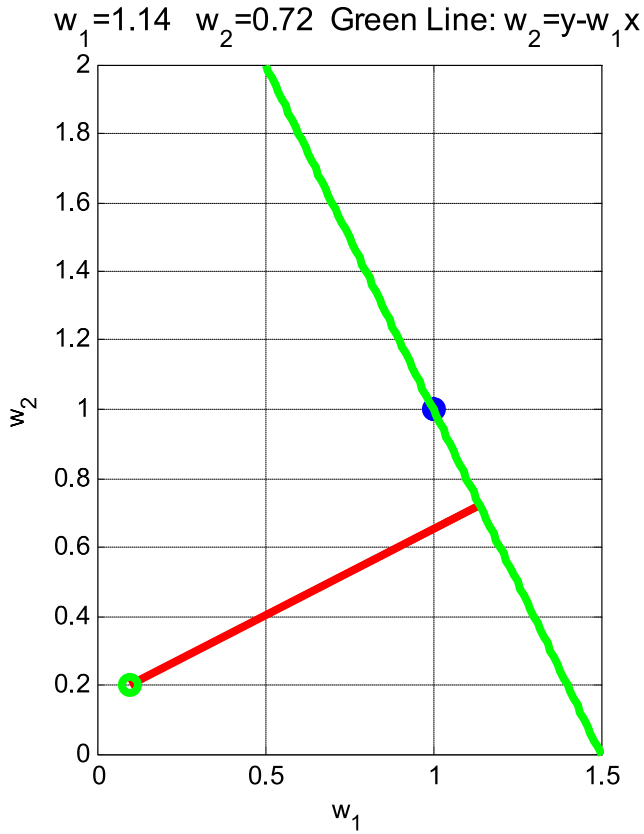
Pair no	x	y
1	2	3

- NN Model $y_n=w_1x+w_2$
- Initial Conditions $w_1(0)=0.1,\ w_2(0)=0.2$
- The converged model will be $3=2w_1+w_2$

Second Order Training Schemes

A Numerical Example – What happens with P=1? The Extreme Case, x=2 and y=3





Second Order Training Schemes

A Numerical Example – What happens with P=1? The Extreme Case, x=1 and y=2

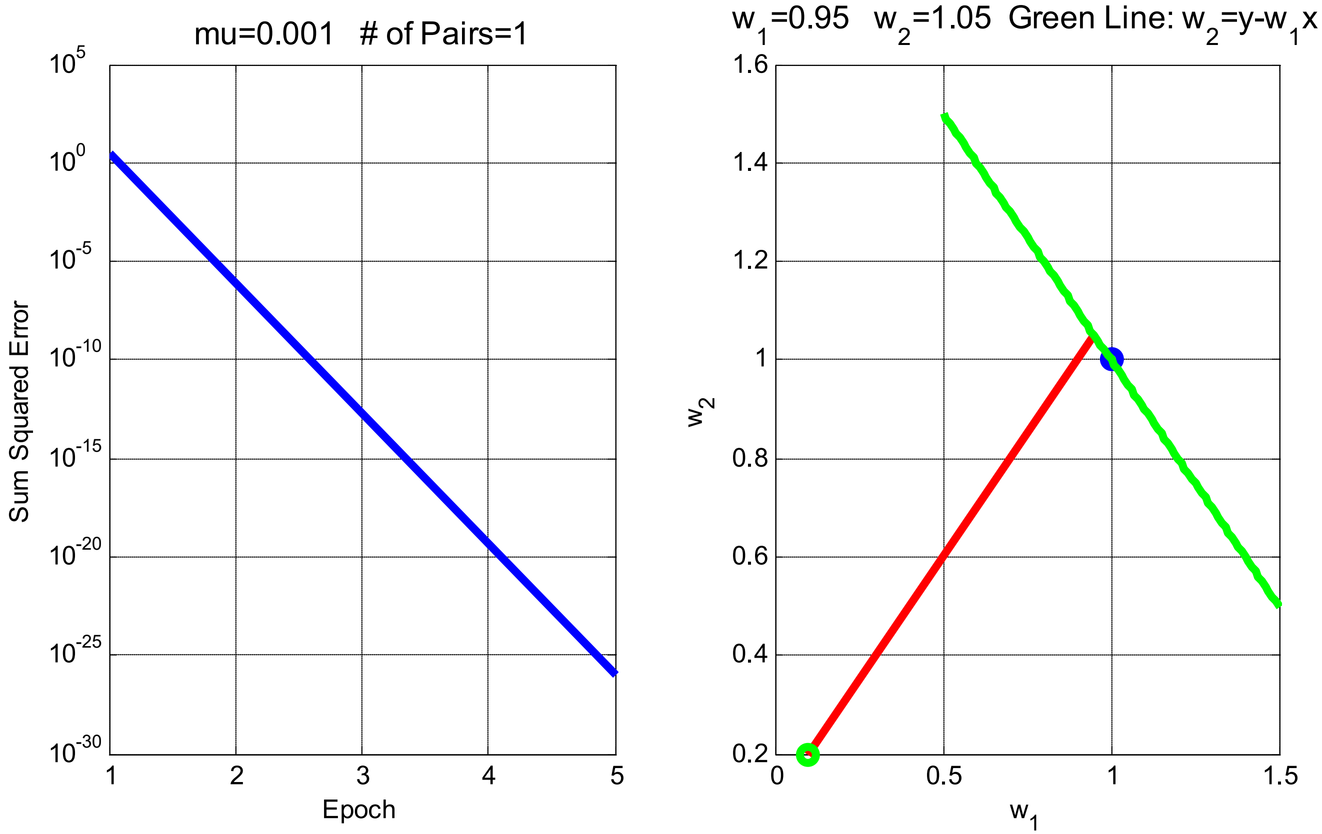
- Process $y=ax+b$
- Input x
- Output y
- Available Data

Pair no	x	y
1	1	2

- NN Model $y_n=w_1x+w_2$
- Initial Conditions $w_1(0)=0.1,\ w_2(0)=0.2$
- The converged model will be $2=w_1+w_2$

Second Order Training Schemes

A Numerical Example – What happens with P=1? The Extreme Case



Second Order Training Schemes

A Numerical Example – What happens with P=1? The Extreme Case, x=0 and y=1

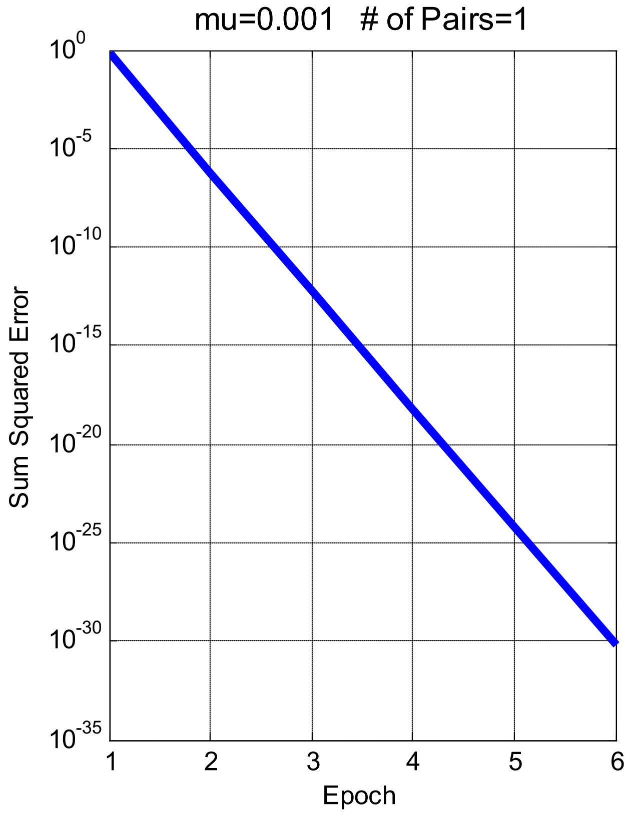
- Process $y=ax+b$
- Input x
- Output y
- Available Data

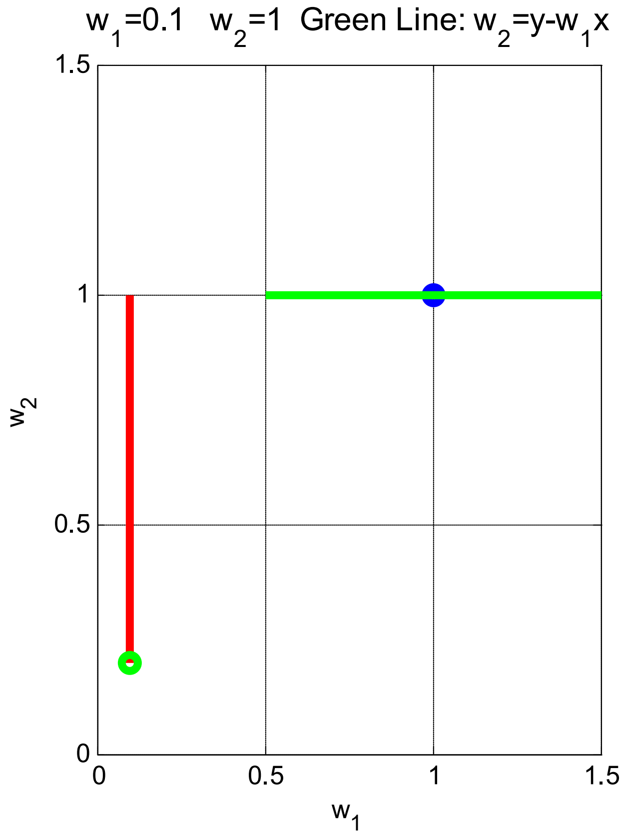
Pair no	x	y
1	0	1

- NN Model $y_n=w_1x+w_2$
- Initial Conditions $w_1(0)=0.1,\ w_2(0)=0.2$
- The converged model will be $1=0w_1+w_2$

Second Order Training Schemes

A Numerical Example – What happens with P=1? The Extreme Case, x=0 and y=1





Second Order Training Schemes

Set mu=0, model is linear, process is linear and no noise. Convergence happens in one iteration!

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \left(J_t^T J_t + \mu I\right)^{-1} J_t^T E_t$$
$$= \mathbf{w}_t - \left(J_t^T J_t + \mu I\right)^{-1} J_t^T (y - y_n)$$

$$y = ax + b$$

$$J_t = [-x \quad -1_{3\times 1}]$$

$$y = -J_t \begin{bmatrix} a \\ b \end{bmatrix} = -J_t \mathbf{w}^*$$

$$y_n = -J_t \begin{bmatrix} w_{1t} \\ w_{2t} \end{bmatrix} = -J_t \begin{bmatrix} w_1(t) \\ w_2(t) \end{bmatrix} = -J_t \mathbf{w}_t$$

$$y - y_n = -J_t\left(\mathbf{w}^* - \mathbf{w}_t\right)$$

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \left(0I + J_t^T J_t\right)^{-1} J_t^T (y - y_n)$$
$$= \mathbf{w}_t - \left(J_t^T J_t\right)^{-1} J_t^T \left(-J_t\left(\mathbf{w}^* - \mathbf{w}_t\right)\right)$$
$$= \mathbf{w}_t + \left(J_t^T J_t\right)^{-1} J_t^T J_t \left(\mathbf{w}^* - \mathbf{w}_t\right)$$
$$= \mathbf{w}_t + \left(\mathbf{w}^* - \mathbf{w}_t\right)$$
$$= \mathbf{w}^*$$

Second Order Training Schemes

Remarks

- For Levenberg-Marquardt algorithm, it is possible to adjust the parameter μ

- There are other methods which are 2nd order and similar in principle to Levenberg-Marquardt algorithm. Conjugate Gradient method is an example to this.

- In order to tune the parameters of a neural network, one may also use derivative-free optimization techniques. EBP, LM, GN, CG approaches are all based on the gradients.

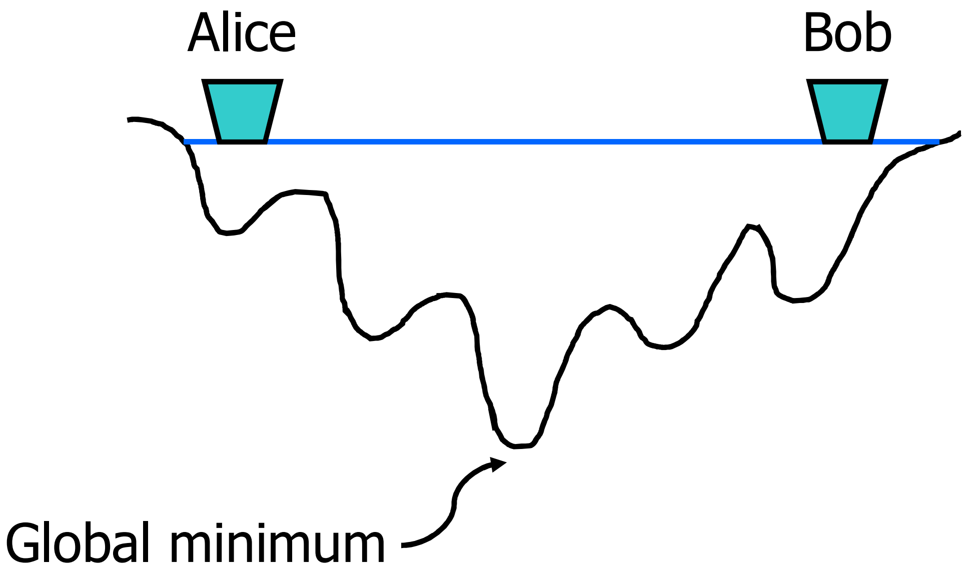
- Derivative Free Optimization
  Particle Swarm Optimization (PSO) for NN Training

Derivative Free Optimization
Particle Swarm Optimization (PSO)

- PSO does not use gradients! No derivatives are required
- Successfully applied in various fields e.g. machine learning, operations research etc.
- R.C. Eberhart and J. Kennedy, “A New Optimizer Using Particle Swarm Theory,” 1995.
- Algoritm is simple yet powerful.

Derivative Free Optimization

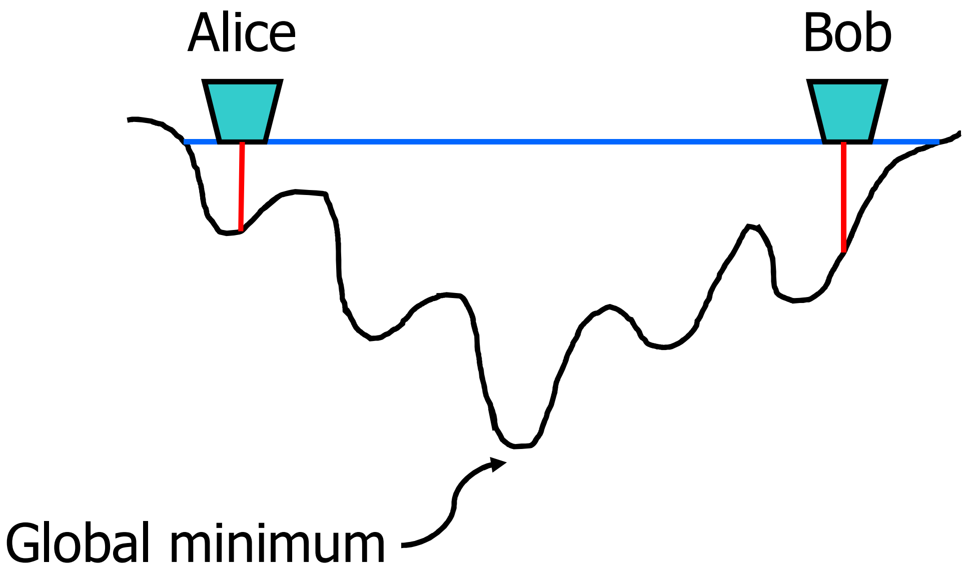
Particle Swarm Optimization



- Alice (A) and Bob (B) cooperate to find the deepest location of the lake
- This is a search problem and it can be stated as an optimization problem
- A and B have two boats and measurement tools to measure the depths

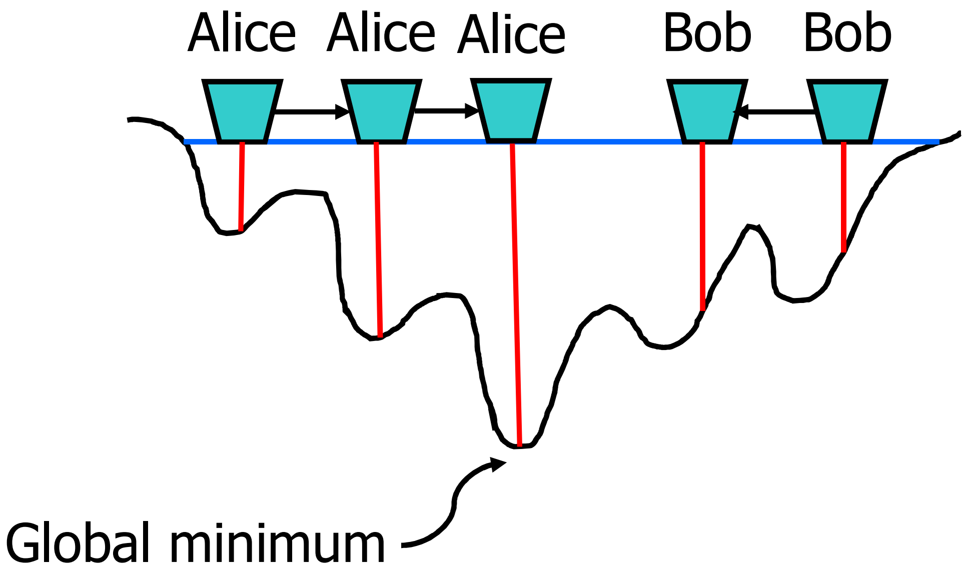
Derivative Free Optimization

Particle Swarm Optimization (PSO)



- They make measurements and inform each other

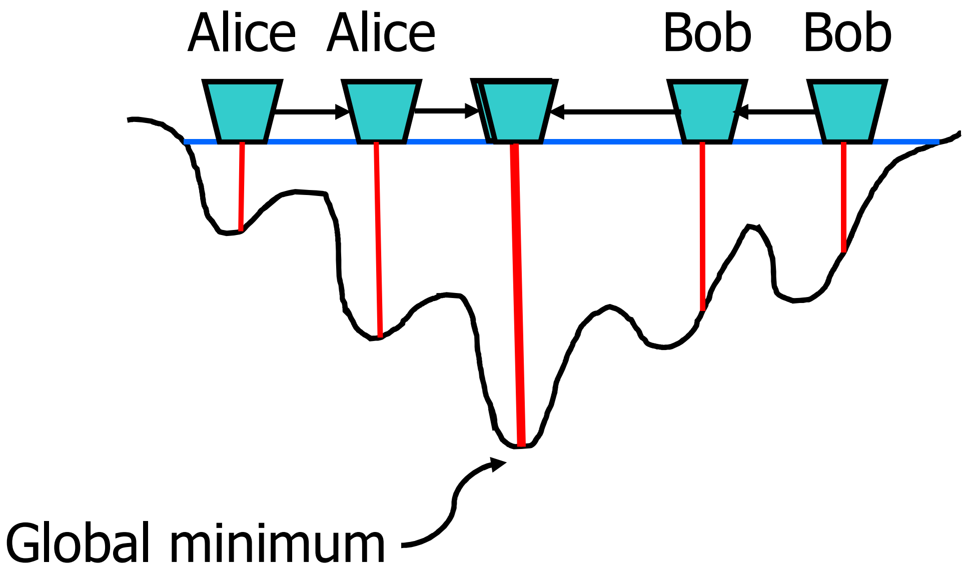
Derivative Free Optimization
Particle Swarm Optimization (PSO)



- They make measurements and inform each other
- In the next step, each one moves a little bit and make new measurements and inform each other
- Alice and Bob do not know the global minimum, they must cooperate to locate it

Derivative Free Optimization

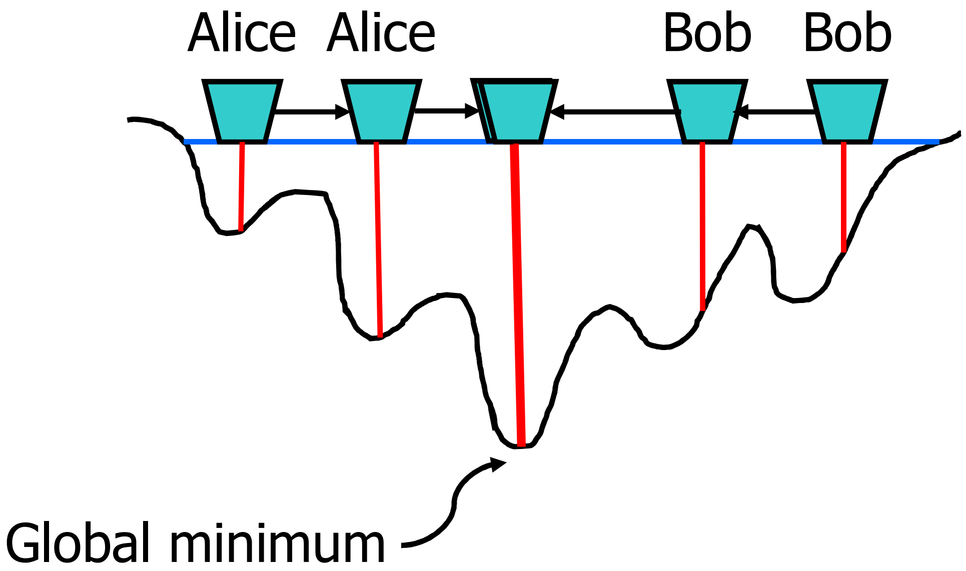
Particle Swarm Optimization (PSO)



- In this Picture, Alice found the global minimum and she cannot find a better location around, she informs Bob continuously, and Bob moves toward Alice
- They meet at the global minimum!

Derivative Free Optimization

Particle Swarm Optimization (PSO)

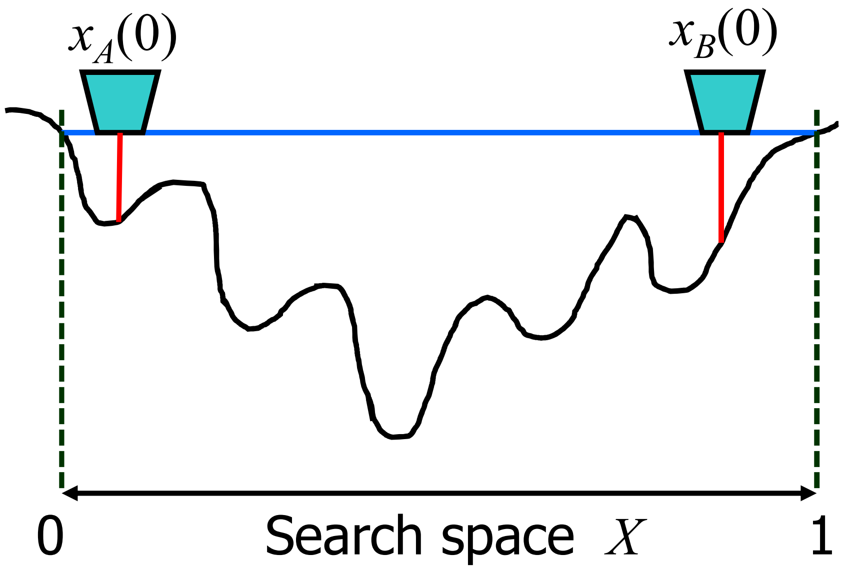


- Communication + learning
- Communication: A and B inform each other
- Learning: A moves towards B or B moves towards A so that they learn a better location

Derivative Free Optimization

Particle Swarm Optimization (PSO)

- Birds, ants and fish use a similar strategy to find food
- Agents (Bob or Alice in our example) are unintelligent but the swarm is
- PSO contains a population of candidate solutions
- Each particle has a position vector in the (possibly multidimensional) search space
- Each particle has a velocity vector



$x_i(t)$

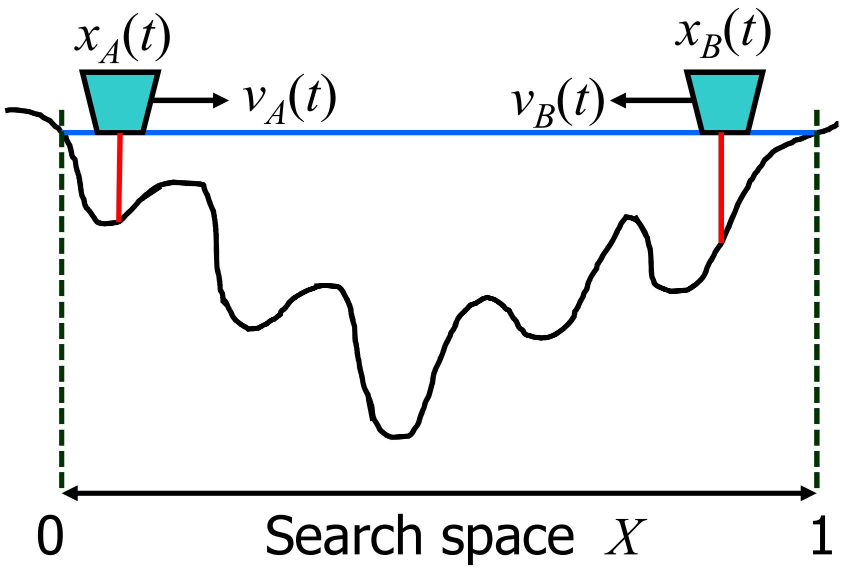
$x_i(t)$: Position vector

t: Discrete time index

Derivative Free Optimization

Particle Swarm Optimization (PSO)

- Birds, ants and fish use a similar strategy to find food
- Agents (Bob or Alice in our example) are unintelligent but the swarm is
- PSO contains a population of candidate solutions
- Each particle has a position vector in the (possible multidimensional) search space
- Each particle has a velocity vector



$x_i(t)$
$v_i(t)$

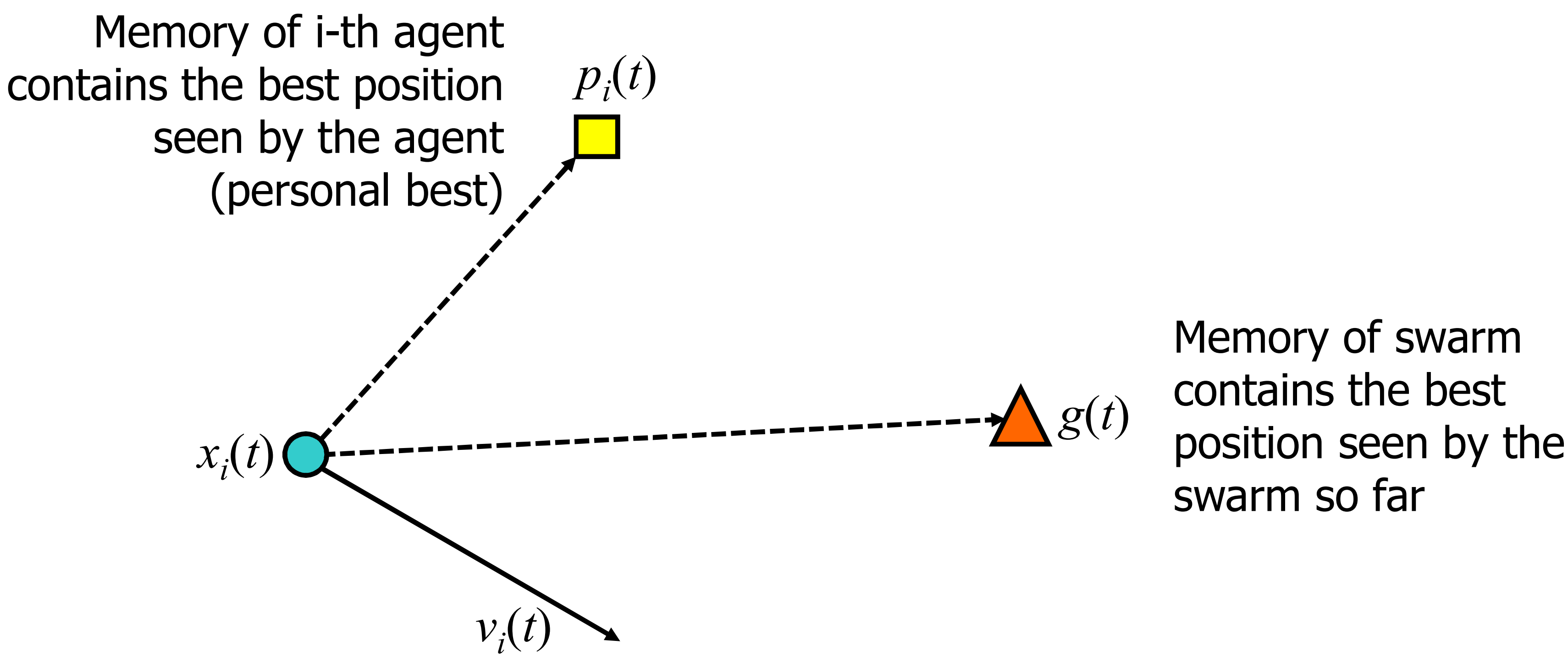
$x_i(t)$: Position vector

$v_i(t)$: Velocity vector

t: Discrete time index

Derivative Free Optimization

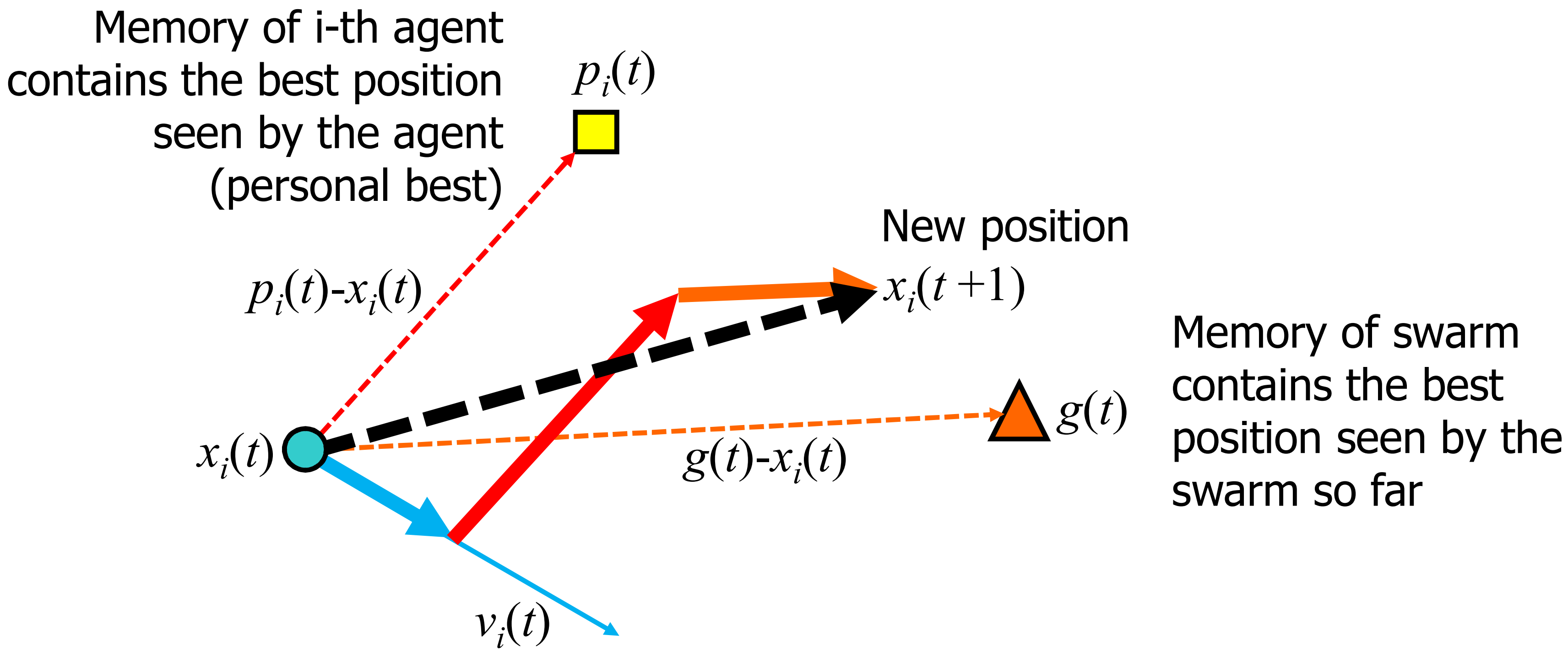
Particle Swarm Optimization (PSO)



Mehmet Önder Efe, *Neural Networks*, Lecture Notes, 2022.

Derivative Free Optimization

Particle Swarm Optimization (PSO)



$$x_i(t+1) = x_i(t) + v_i(t+1)$$

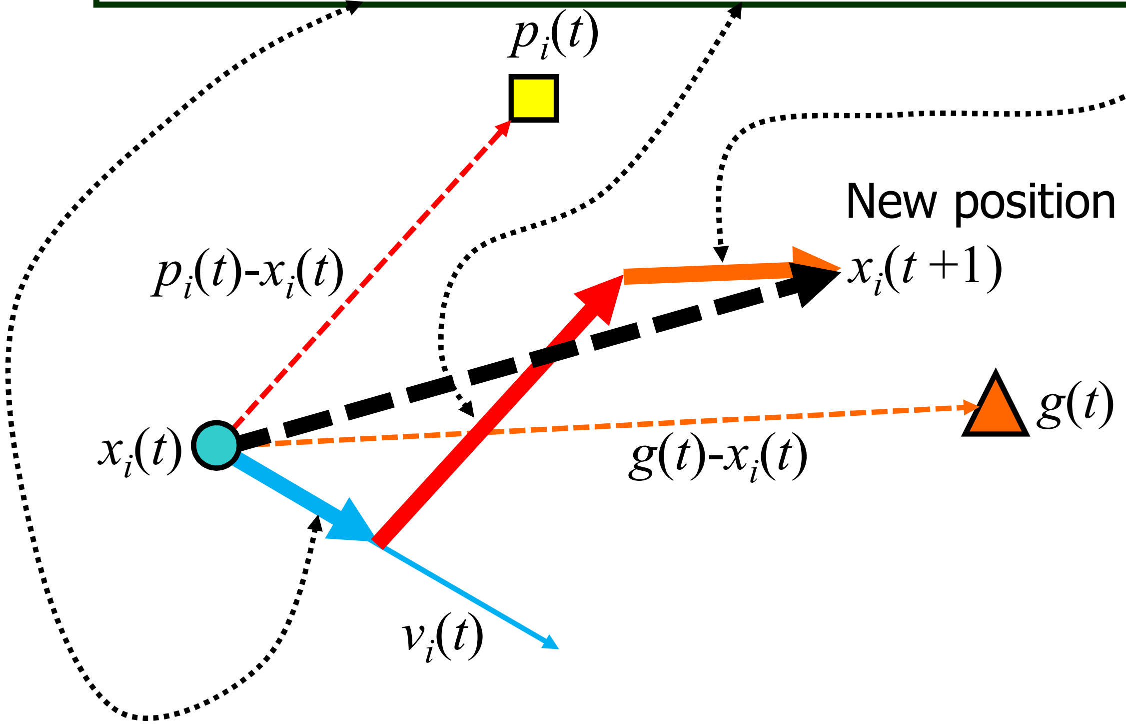
$$v_i(t+1) = wv_i(t) + c_1(p_i(t) - x_i(t)) + c_2(g(t) - x_i(t))$$

Derivative Free Optimization

Particle Swarm Optimization (PSO)

$$x_i(t+1) = x_i(t) + v_i(t+1)$$

$$v_i(t+1) = wv_i(t) + c_1(p_i(t) - x_i(t)) + c_2(g(t) - x_i(t))$$



Derivative Free Optimization

Particle Swarm Optimization (PSO)

$$x_i(t+1) = x_i(t) + v_i(t+1)$$

$$v_i(t+1) = wv_i(t) + c_1(p_i(t) - x_i(t)) + c_2(g(t) - x_i(t))$$

Inertia coefficient (w)

Acceleration coefficients (c_1, c_2)

Derivative Free Optimization

Particle Swarm Optimization (PSO)

$$x_i(t+1) = x_i(t) + v_i(t+1)$$

$$v_i(t+1) = wv_i(t) + c_1(p_i(t) - x_i(t)) + c_2(g(t) - x_i(t))$$

Inertia term

Keep the momentum partially

Social component

Due to the swarm's best experience

Cognitive component

Due to its best experience

Practical implementation makes use of randomness in coefficients.

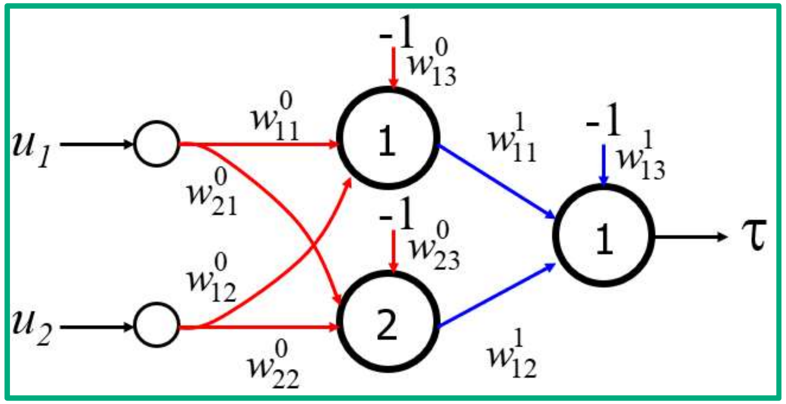
$r_1, r_2 \in \mathrm{U}(0,1)$ Random number distributed uniformly

$$x_i(t+1) = x_i(t) + v_i(t+1)$$

$$v_i(t+1) = wv_i(t) + r_1c_1(p_i(t) - x_i(t)) + r_2c_2(g(t) - x_i(t))$$

Derivative Free Optimization

Particle Swarm Optimization (PSO)



$$\mathbf{w} = \begin{bmatrix} w_{11}^0 \\ w_{12}^0 \\ w_{13}^0 \\ w_{21}^0 \\ w_{22}^0 \\ w_{23}^0 \\ w_{11}^1 \\ w_{12}^1 \\ w_{13}^1 \end{bmatrix}_{9\times 1}, \quad x_i(t) = \begin{bmatrix} w_{11}^{0i}(t) \\ w_{12}^{0i}(t) \\ w_{13}^{0i}(t) \\ w_{21}^{0i}(t) \\ w_{22}^{0i}(t) \\ w_{23}^{0i}(t) \\ w_{11}^{1i}(t) \\ w_{12}^{1i}(t) \\ w_{13}^{1i}(t) \end{bmatrix}_{9\times 1}, v_i(t) = \begin{bmatrix} v_{11}^{0i}(t) \\ v_{12}^{0i}(t) \\ v_{13}^{0i}(t) \\ v_{21}^{0i}(t) \\ v_{22}^{0i}(t) \\ v_{23}^{0i}(t) \\ v_{11}^{1i}(t) \\ v_{12}^{1i}(t) \\ v_{13}^{1i}(t) \end{bmatrix}_{9\times 1}$$

$$x_i(t+1) = x_i(t) + v_i(t+1)$$
$$v_i(t+1) = wv_i(t) + c_1(p_i(t) - x_i(t)) + c_2(g(t) - x_i(t))$$

$$\overline{J}(\mathbf{w}) = \sum_{p=1}^{P}\left[\sum_{k=1}^{K}\left(d_k^p - \tau_k^p\right)^2\right]$$

Mehmet Önder Efe, *Neural Networks*, Lecture Notes, 2022.

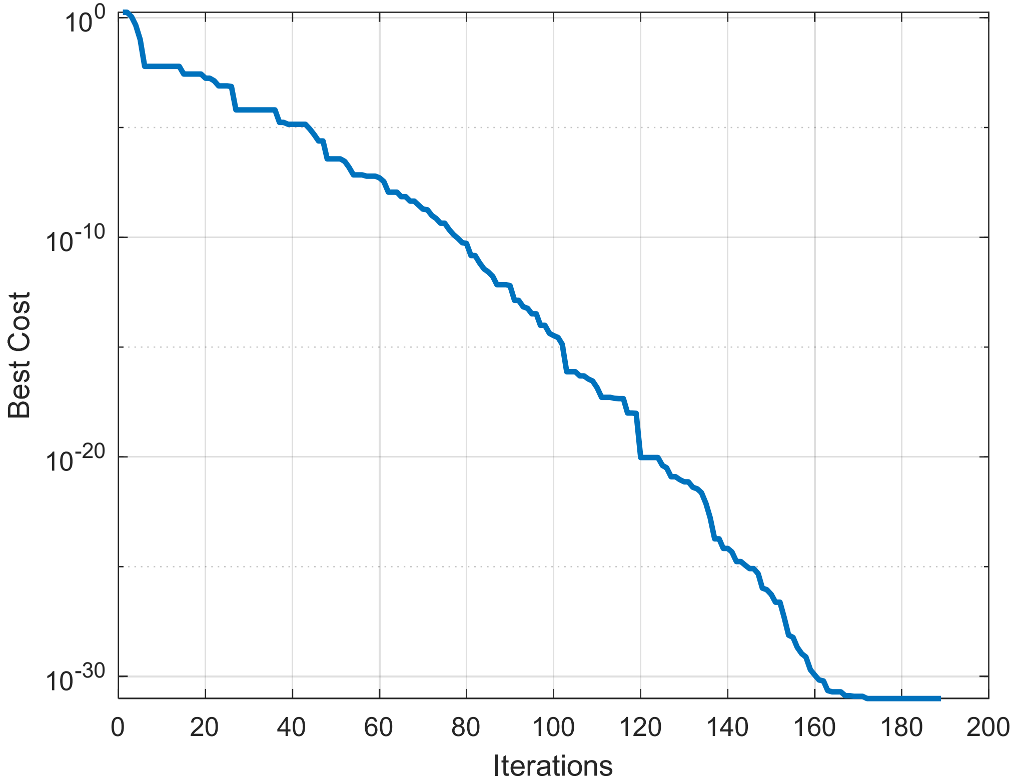
Derivative Free Optimization
Particle Swarm Optimization (PSO)

- For the implementation details, watch
- https://www.youtube.com/watch?v=sB1n9a9yxJk
- https://www.youtube.com/watch?v=xPkRL_Gt6PI
- https://www.youtube.com/watch?v=ICBYrKsFPqA

Derivative Free Optimization

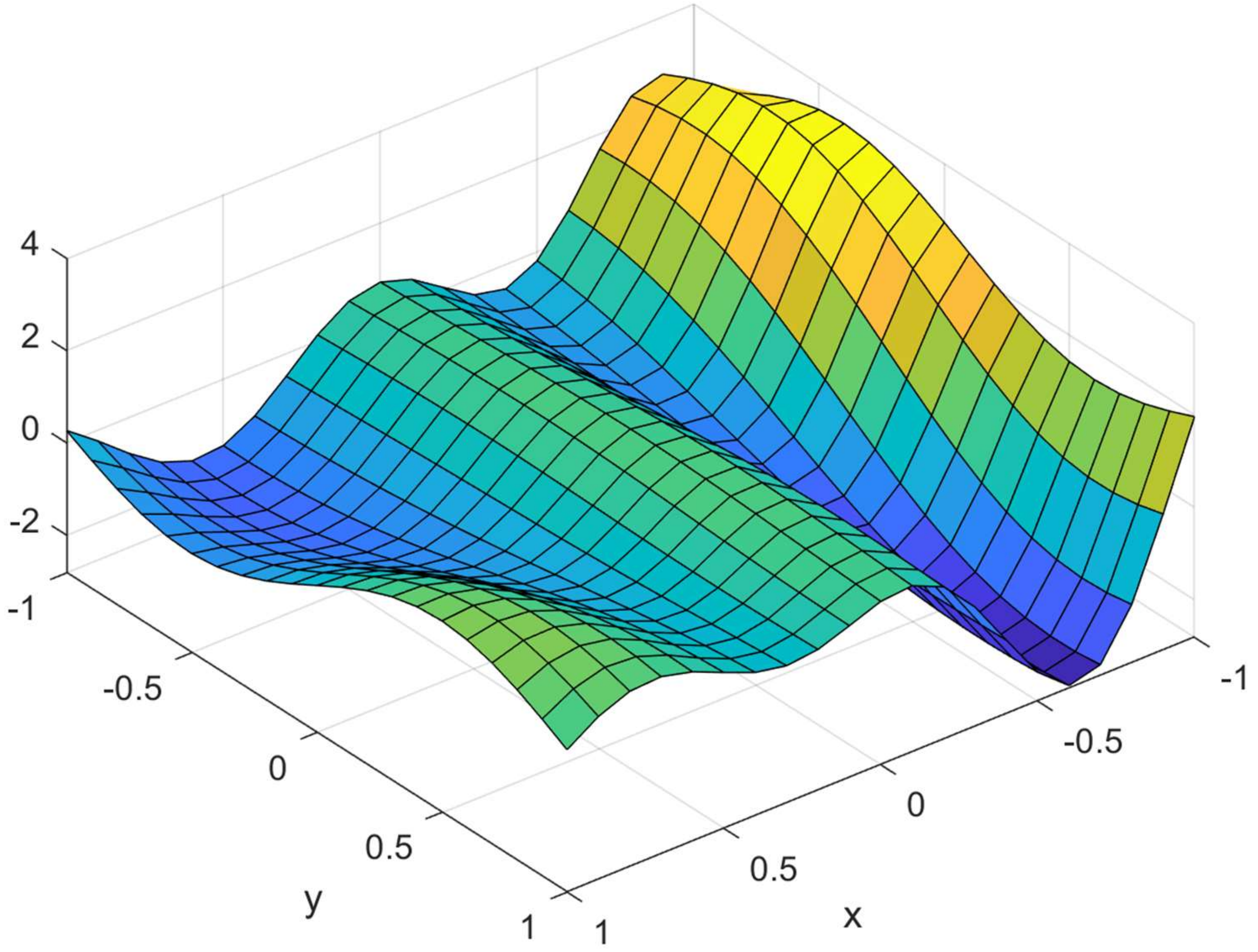
Particle Swarm Optimization (PSO)

- Problem: XOR
- NN Structure: 2-5-5-1
- Max Iterations: 1000
- Population size: 50



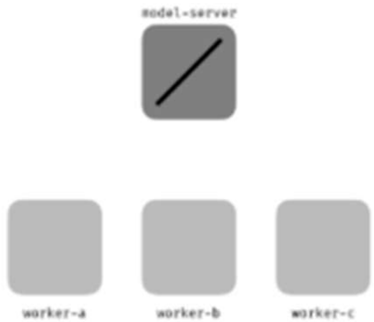
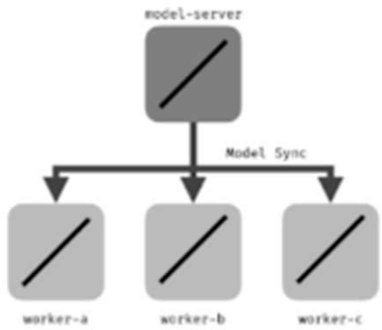
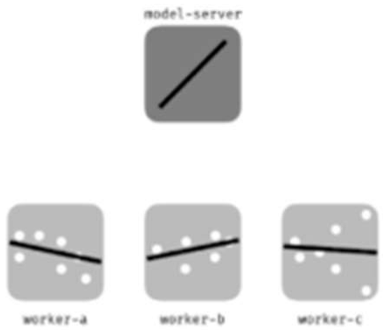
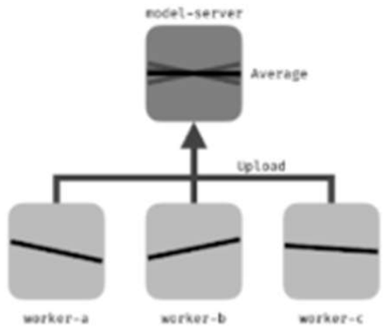
Mehmet Önder Efe, *Neural Networks*, Lecture Notes, 2022.

Federated Learning

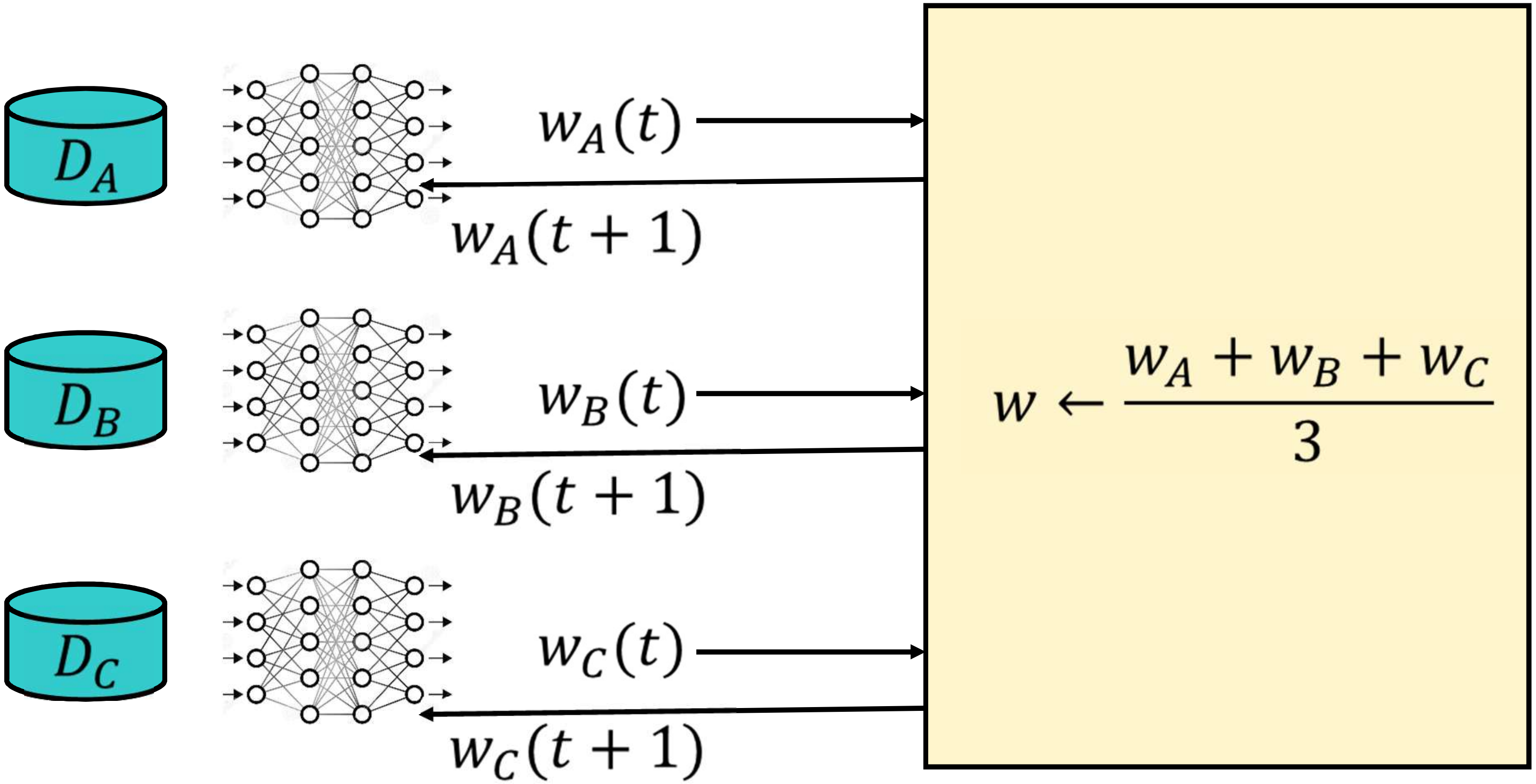


Mehmet Önder Efe, *Neural Networks*, Lecture Notes, 2022.

Federated Learning

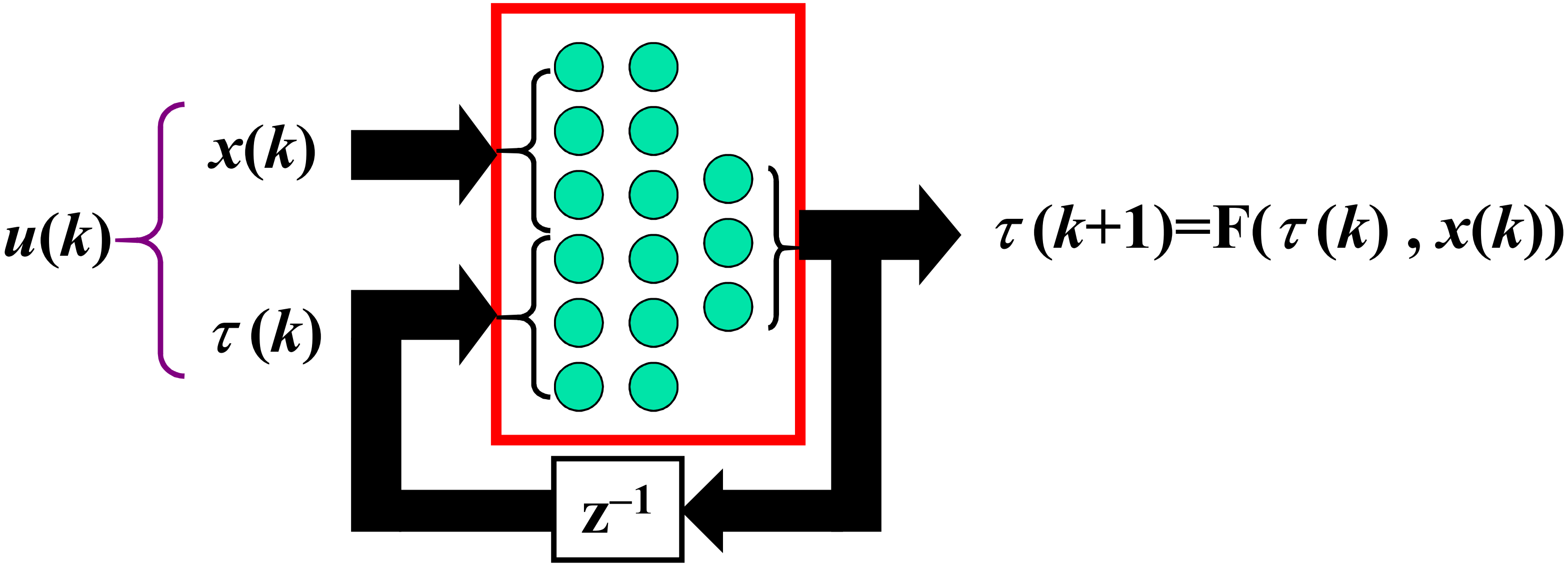
Step 1	Step 2	Step 3	Step 4
			
Central server chooses a statistical model to be trained	Central server transmits the initial model to several nodes	Nodes train the model locally with their own data	Central server pools model results and generate one global mode without accessing any data

Federated Learning



- Recurrent Neural Network Structures
- Several Applications of Neural Networks
    - Identification of Nonlinear Systems
    - Neurocontrol Structures
    - Noise Elimination
    - Adaptive Noise Cancellation
    - VLSI Implementation of NNs
    - NNs in Medical Diagnosis
    - NNs for Financial Applications

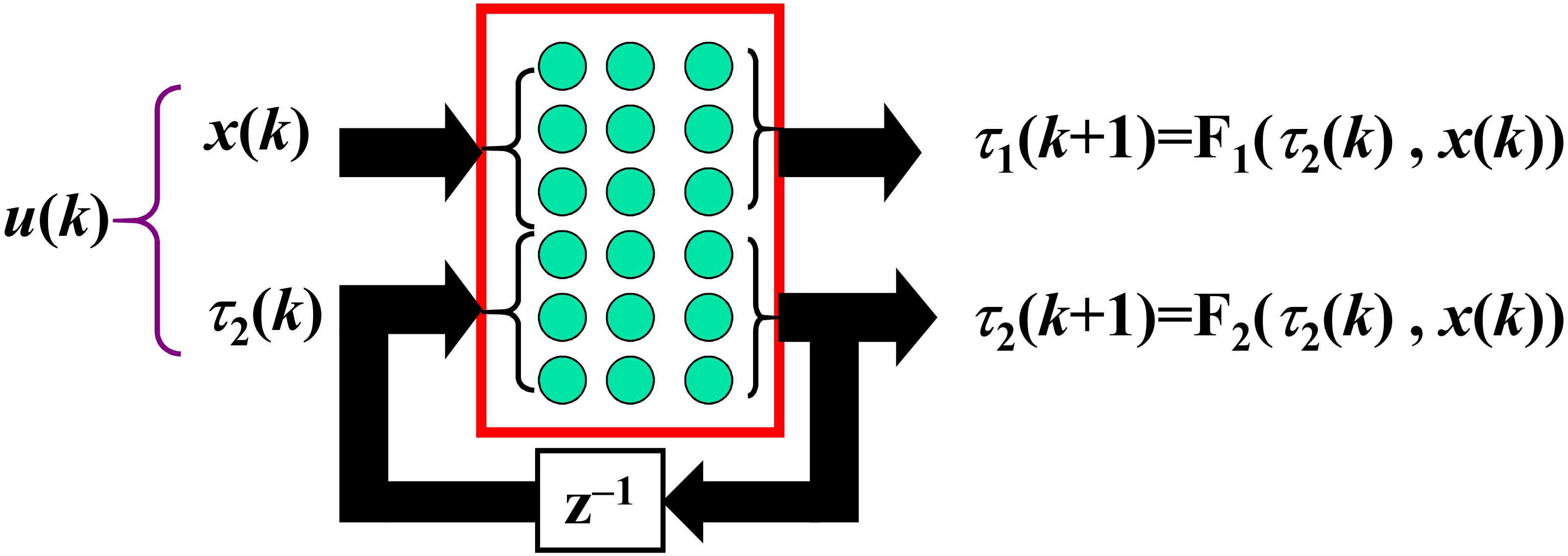
Recurrent Neural Net Structures



Real time recurrent network

- Note that, only the structure of the input vector changes
- Without any modification, EBP applies
- You may have as many hidden layers as you want
- Useful for short term prediction

Recurrent Neural Net Structures

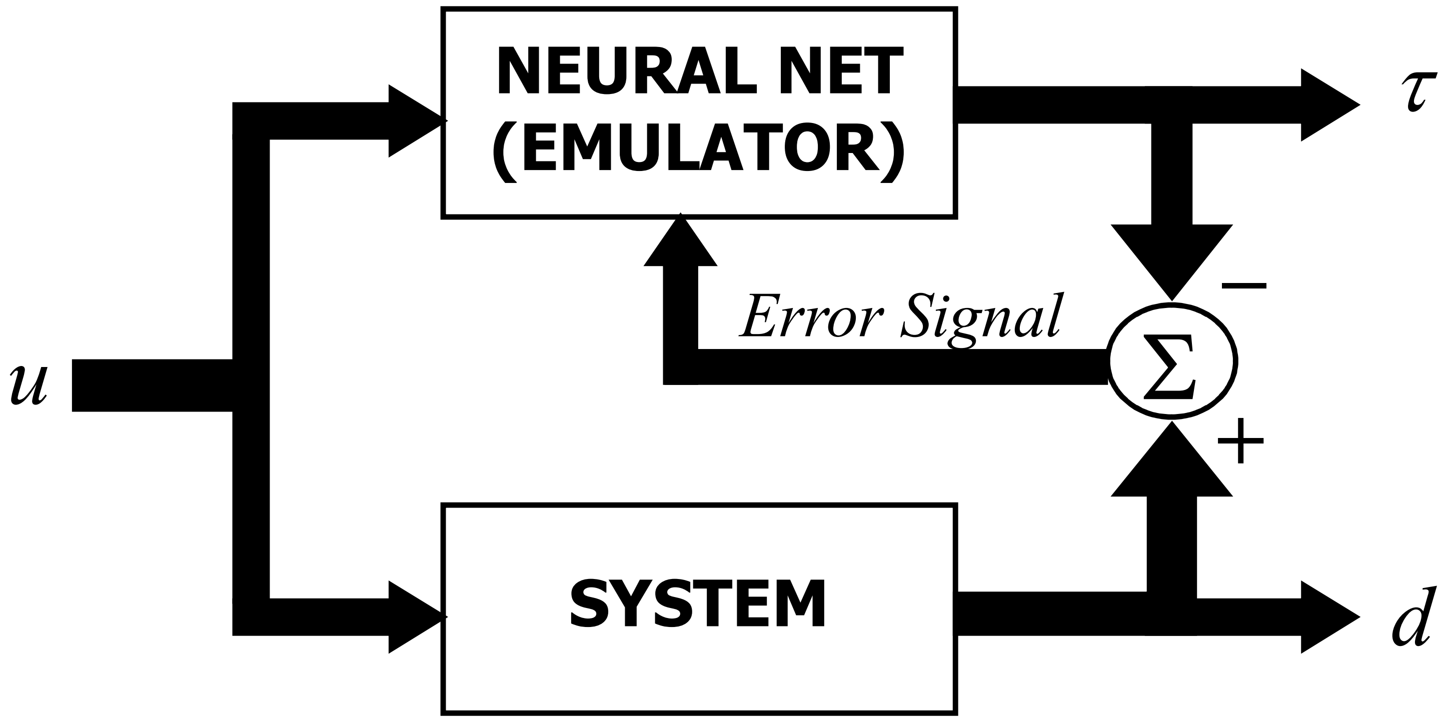


Partially recurrent network

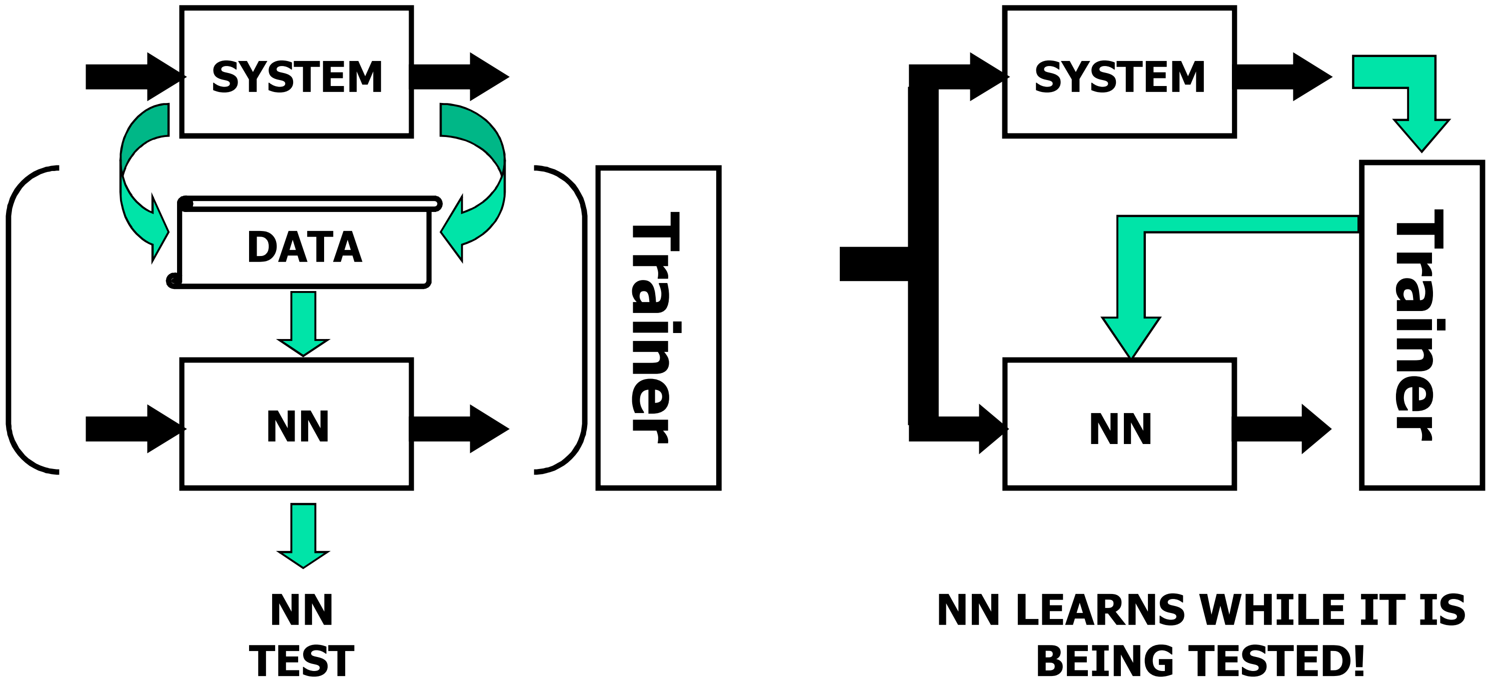
- Note that, the structure of the input and output vectors change
- Without any modification, EBP applies
- You may have as many hidden layers as you want
- Useful for short term prediction

Applications of Neural Networks

Identification of Nonlinear Systems

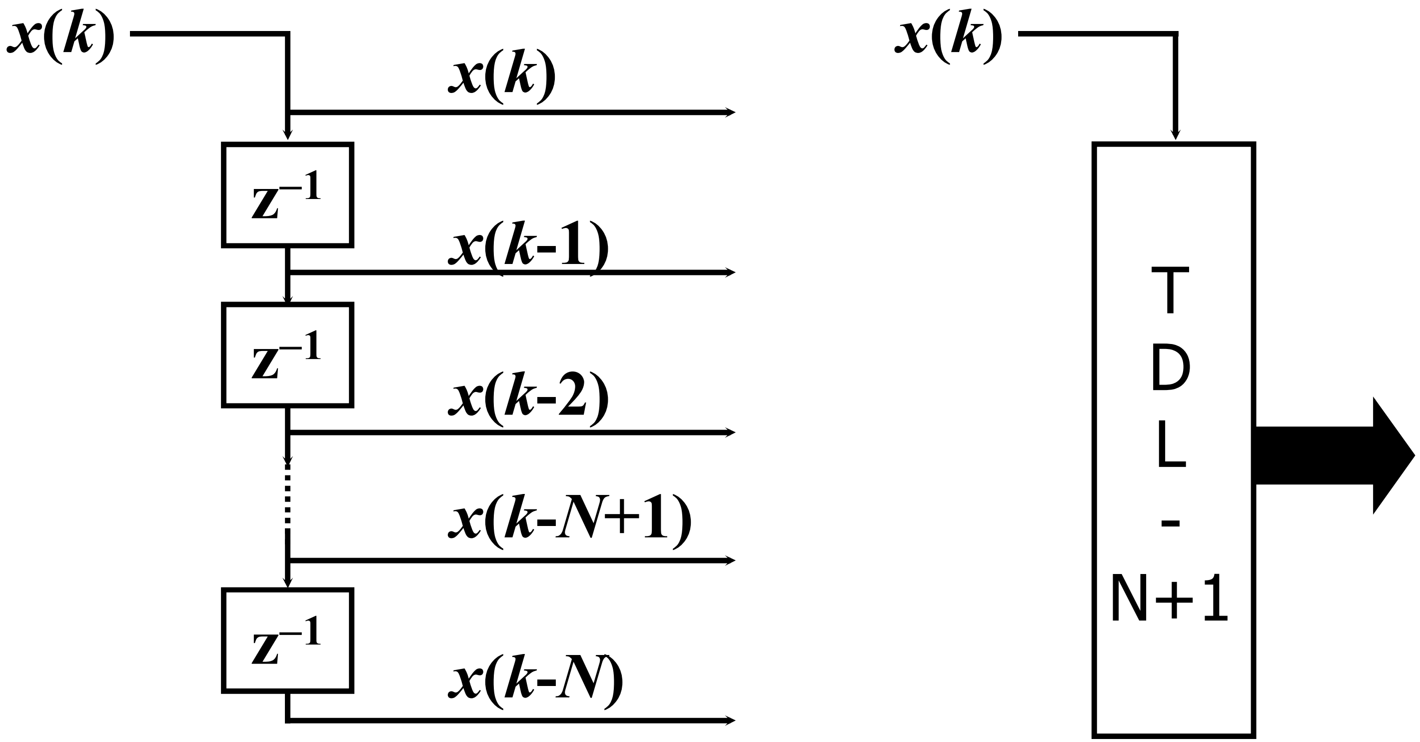


Applications of Neural Networks
Identification of Nonlinear Systems



- Online Identification: At time t, you have one pair, process it
- Offline Identification: You have **a set of data**, process it

Applications of Neural Networks
Identification of Nonlinear Systems



- TDL-N stands for Tapped Delay Line with delay depth = N

Applications of Neural Networks

Identification of Nonlinear Systems

4 IEEE TRANSACTIONS ON NEURAL NETWORKS, VOL. 1, NO. 1, MARCH 1990

Identification and Control of Dynamical Systems Using Neural Networks

KUMPATI S. NARENDRA FELLOW, IEEE, AND KANNAN PARTHASARATHY

***Abstract*—The paper demonstrates that neural networks can be used effectively for the identification and control of nonlinear dynamical systems. The emphasis of the paper is on models for both identification and control. Static and dynamic back-propagation methods for the adjustment of parameters are discussed. In the models that are introduced, multilayer and recurrent networks are interconnected in novel configurations and hence there is a real need to study them in a unified fashion. Simulation results reveal that the identification and adaptive control schemes suggested are practically feasible. Basic concepts and definitions are introduced throughout the paper, and theoretical questions which have to be addressed are also described.**

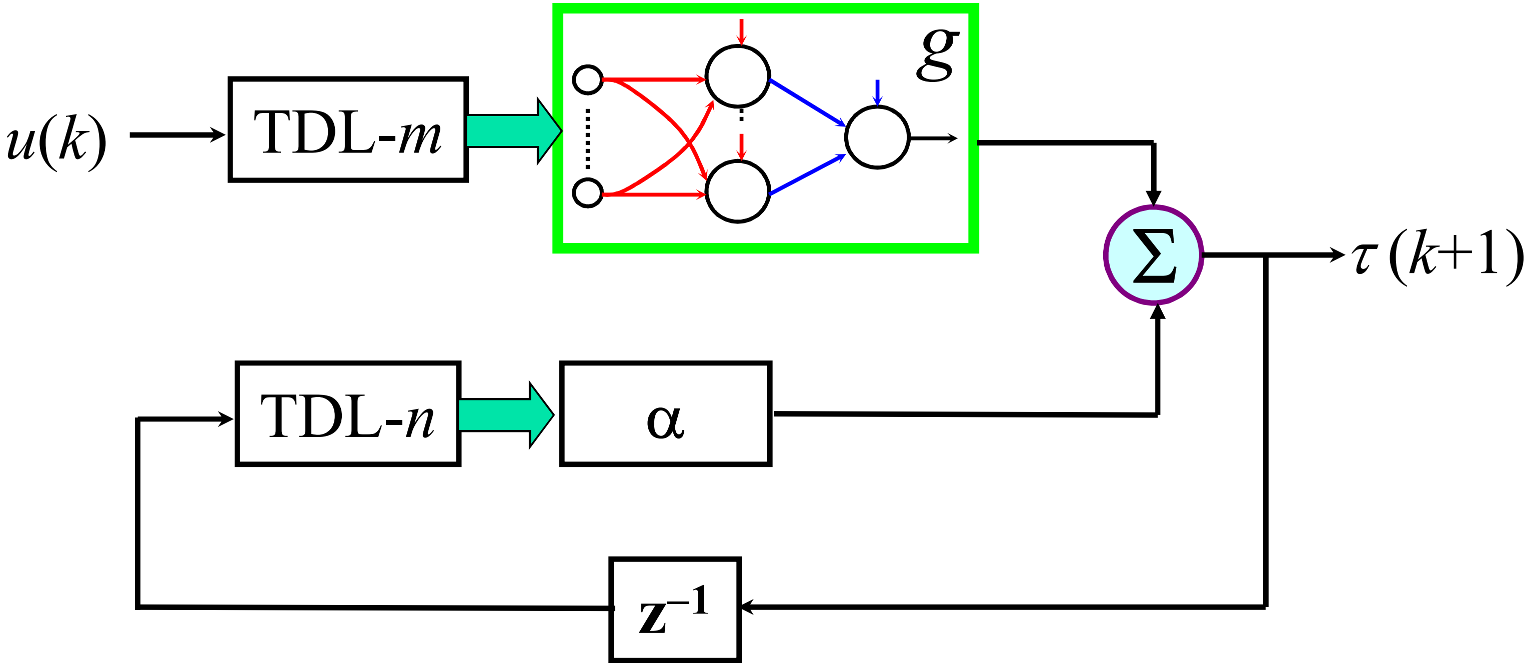
are well known for such systems [1]. In this paper our interest is in the identification and control of nonlinear dynamic plants using neural networks. Since very few results exist in nonlinear systems theory which can be directly applied, considerable care has to be exercised in the statement of the problems, the choice of the identifier and controller structures, as well as the generation of adaptive laws for the adjustment of the parameters.

Two classes of neural networks which have received

Applications of Neural Networks

Identification of Nonlinear Systems

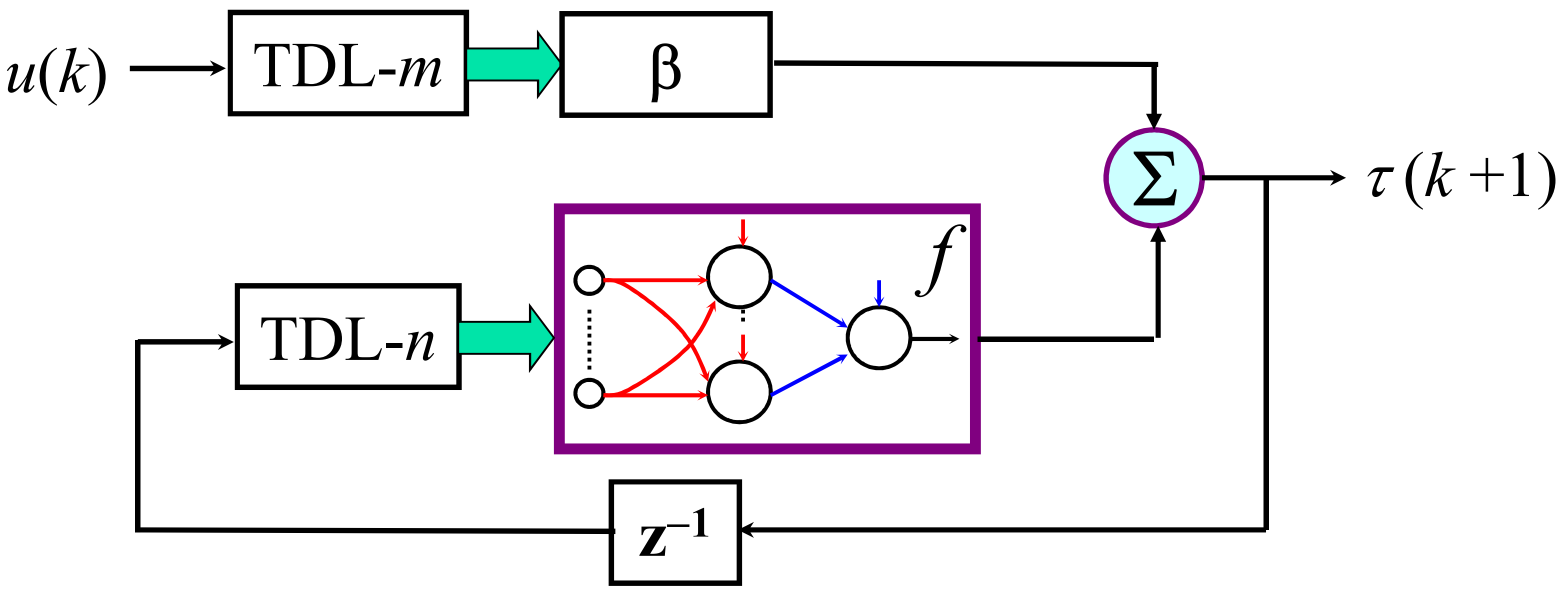
$$\tau(k+1) = \sum_{i=0}^{n-1} \alpha_i \tau(k-i) + g\big(u(k), u(k-1), \ldots, u(k-m+1)\big)$$



Applications of Neural Networks

Identification of Nonlinear Systems

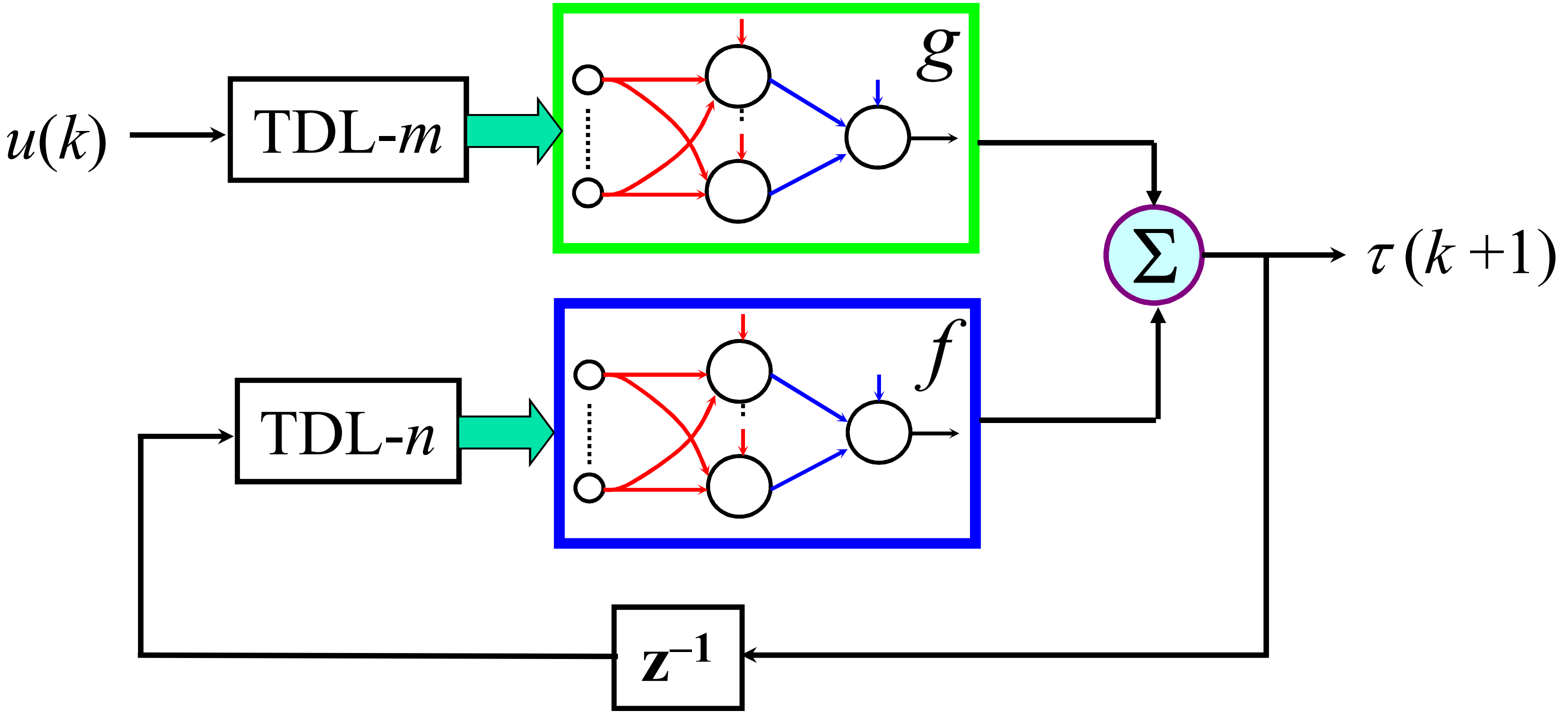
$$\tau(k+1) = f\big(\tau(k), \tau(k-1), \ldots, \tau(k-n+1)\big) + \sum_{i=0}^{m-1} \beta_i u(k-i)$$



Applications of Neural Networks

Identification of Nonlinear Systems

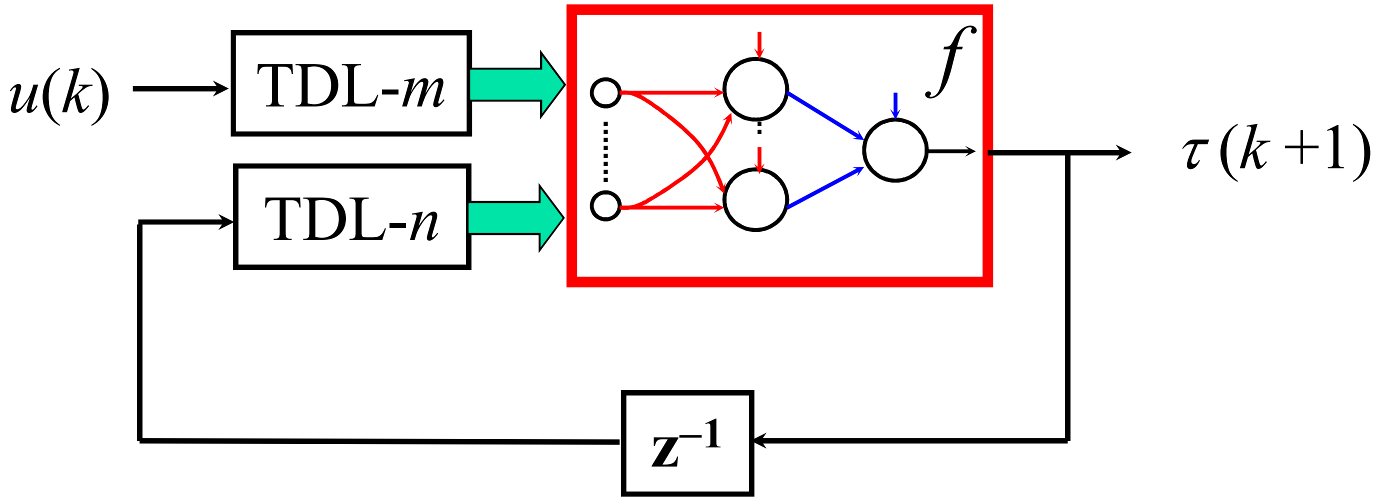
$$\tau(k+1) = f\big(\tau(k), \tau(k-1), \ldots, \tau(k-n+1)\big) + g\big(u(k), u(k-1), \ldots, u(k-m+1)\big)$$



Mehmet Önder Efe, *Neural Networks*, Lecture Notes, 2022.

Applications of Neural Networks
Identification of Nonlinear Systems

$$\tau(k+1) = f\left(\tau(k), \tau(k-1), \dots, \tau(k-n+1), u(k), u(k-1), \dots, u(k-m+1)\right)$$



Applications of Neural Networks
Identification of Nonlinear Systems - An Example

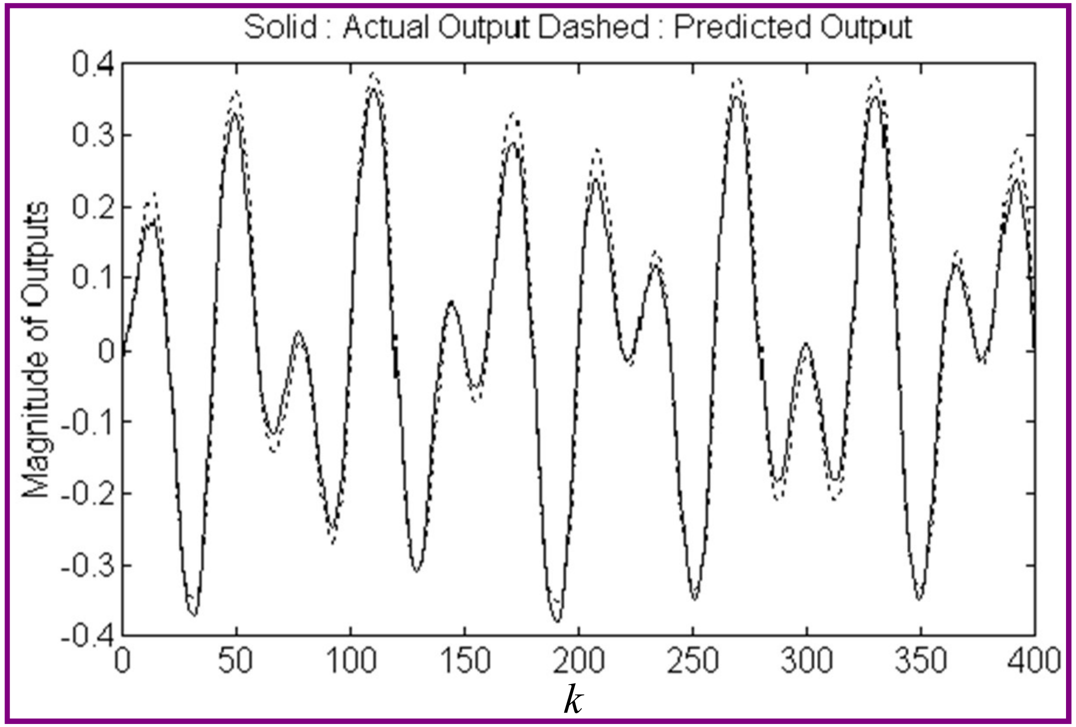
$$\tau(k+1) = f\big(-\tau(k) - 0.6\tau(k-1) + 0.3\tau(k-2) + u(k) + u(k-1)\big)$$

- The function f(.) is unknown
- The output of the system is available, so we can form a data set
- The network has 5-20-10-1 configuration with linear output neuron
- We trained the network with the input output pairs
  and tested with

$$f(x) = \frac{\tanh(x)}{1+x^2}$$

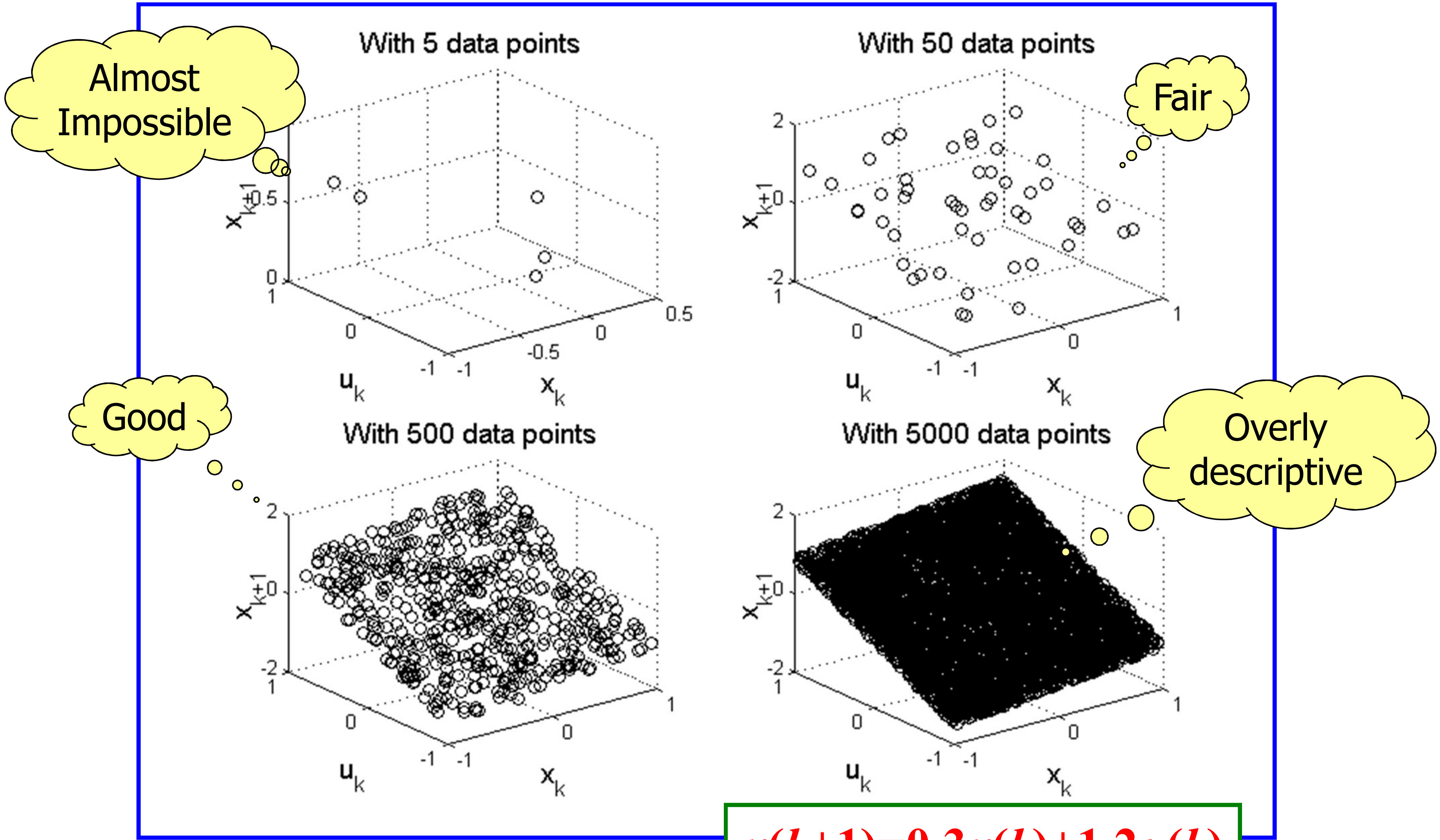
$$u(k) = \frac{1}{2}\sin\left(\frac{2\pi k}{150}\right)\sin\left(\frac{2\pi k}{40}\right)$$

Applications of Neural Networks
Identification of Nonlinear Systems - An Example



Applications of Neural Networks

Information sufficiency (How descriptive is it?)

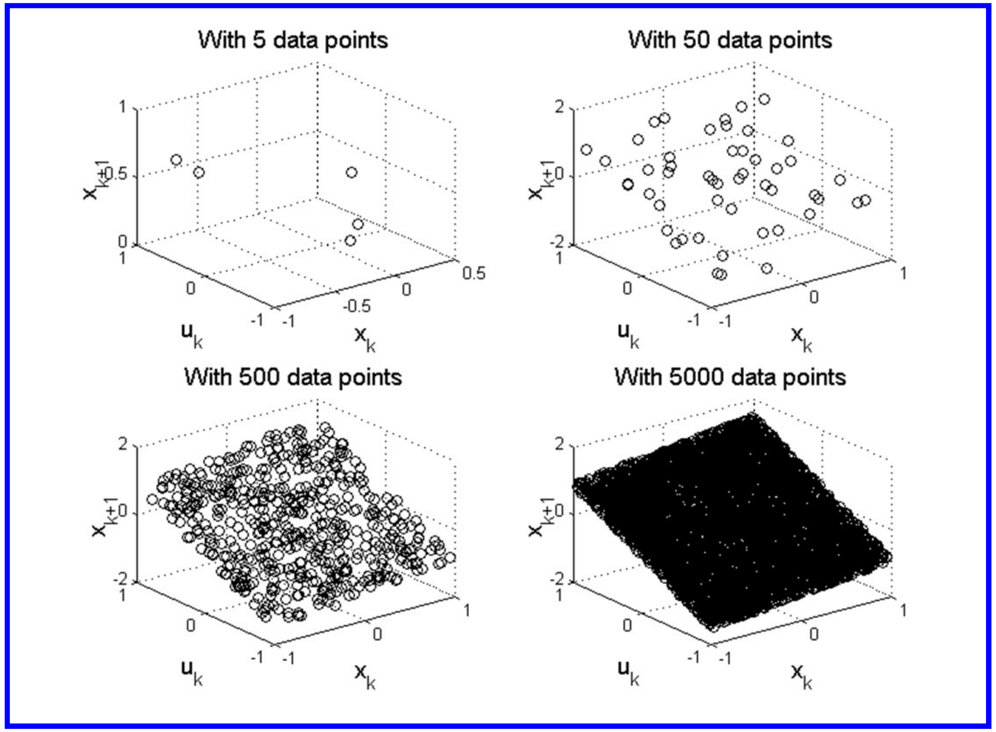


$x(k+1)=0.3x(k)+1.2u(k)$

Mehmet Önder Efe, *Neural Networks*, Lecture Notes, 2022.

Applications of Neural Networks

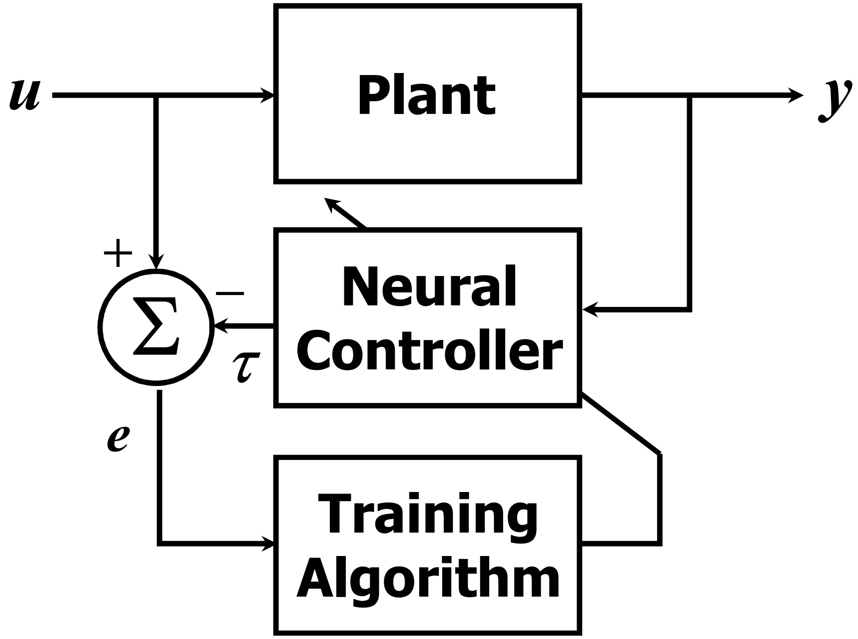
Information sufficiency (How descriptive is it?)



- Input domain for a NN is $x(k) \in [-1,1]$, $u(k) \in [-1,1]$
- If you fail to find a reasonably accurate neural representation of a dynamical system, pay attention to the data on which your model is based

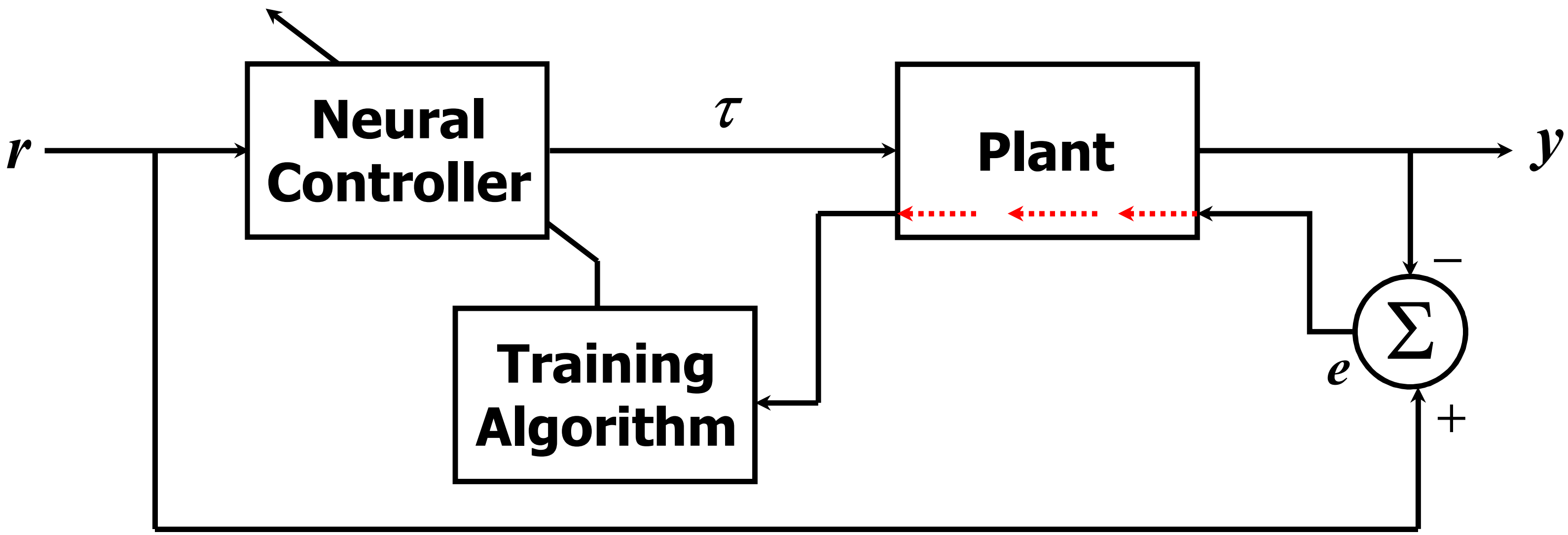
Applications of Neural Networks

Neurocontrol Structures



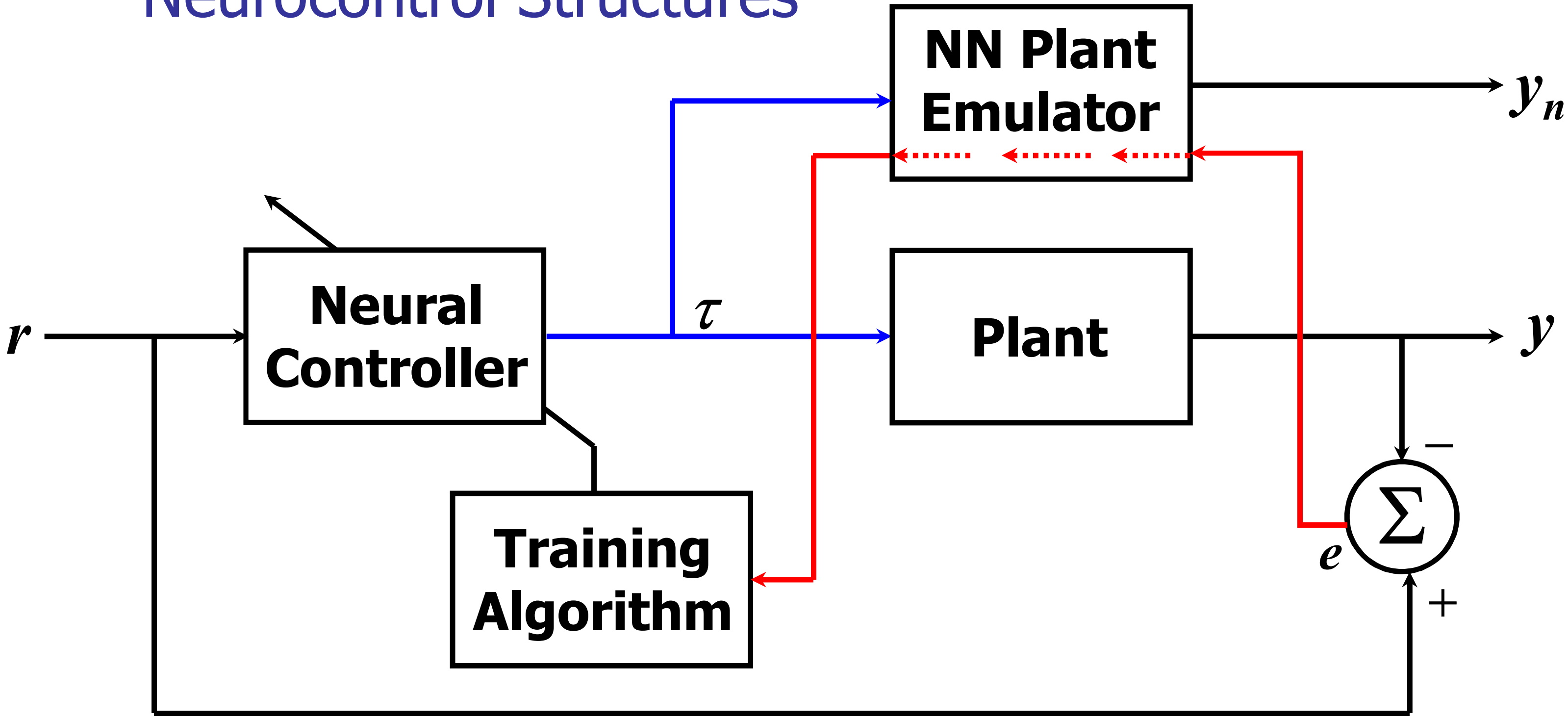
- This structure is known as **Generalized Learning Structure**
- To perform training, you have to choose $\boldsymbol{u}$, but which signals should be used as $\boldsymbol{u}$? You simply choose some set of signals as the training signals, and the controller learns the generalized inverse of the plant
- Controller learns how to reproduce the input signal $\boldsymbol{u}$

Applications of Neural Networks
Neurocontrol Structures



- This structure is known as **Specialized Learning Structure**
- Starting with Generalized Learning Structure provides good initial conditions, then continuing with Specialized Learning Structure lets us design the controller easily
- How is the error passed through the plant?

Applications of Neural Networks
Neurocontrol Structures



- An emulator neural network is prepared offline
- It is installed as shown in the above figure
- The output error is passed through the emulator **without** modifying the weights
- The error at the output of the controller is obtained
- This error is backpropagated through the controller **with** parameter tuning

Applications of Neural Networks

Neurocontrol Structures - Offline synthesis of NNC

- Given the system $\mathbf{x(k+1) = f(x(k)) + g(x(k))u(k)}$
- You know that the transition

$x(k) \Rightarrow \text{SYSTEM} \Rightarrow x(k+1)$

is due to the input $u(k)$. Therefore

- The forward map $[x(k),u(k)] \Rightarrow \text{NN} \Rightarrow x(k+1)$ is an emulator
- The backward map $[x(k+1)\ x(k)] \Rightarrow \text{NN} \Rightarrow u(k)$ is a controller

Read the controller as follows: You are given state $x(k)$, and you want to move to $d(k)$ (which is $x(k+1)$), which $u(k)$ leads to this transition?

Generate the data from the plant, teach it the transitions...

Applications of Neural Networks
Neurocontrol Structures - Offline synthesis of NNC

$$x(k+1) = Ax(k) + Bu(k)$$

$$P^T x(k+1) - P^T Ax(k) = P^T Bu(k),\ P^T B \neq 0$$

$$u(k) = \left[\left(P^T B\right)^{-1} P^T\right] x(k+1) - \left[\left(P^T B\right)^{-1} P^T A\right] x(k)$$

Given the state transition data, a map is available

$x(k+1)$: $r(k)$

Applications of Neural Networks

Neurocontrol Structures - Offline synthesis of NNC

$$x(k+1) = Ax(k) + Bu(k)$$

$$x(k+2) = A^2x(k) + ABu(k) + Bu(k+1)$$

$$x(k+3) = A^3x(k) + A^2Bu(k) + ABu(k+1) + bu(k+2)$$

$$\vdots$$

$$x(k+n) = A^n x(k) + \underbrace{\begin{bmatrix} B & AB & \cdots & A^{n-1}B \end{bmatrix}}_{W_c} \underbrace{\begin{bmatrix} u(k+n-1) \\ u(k+n-2) \\ \vdots \\ u(k) \end{bmatrix}}_{U}$$

Controllability Matrix

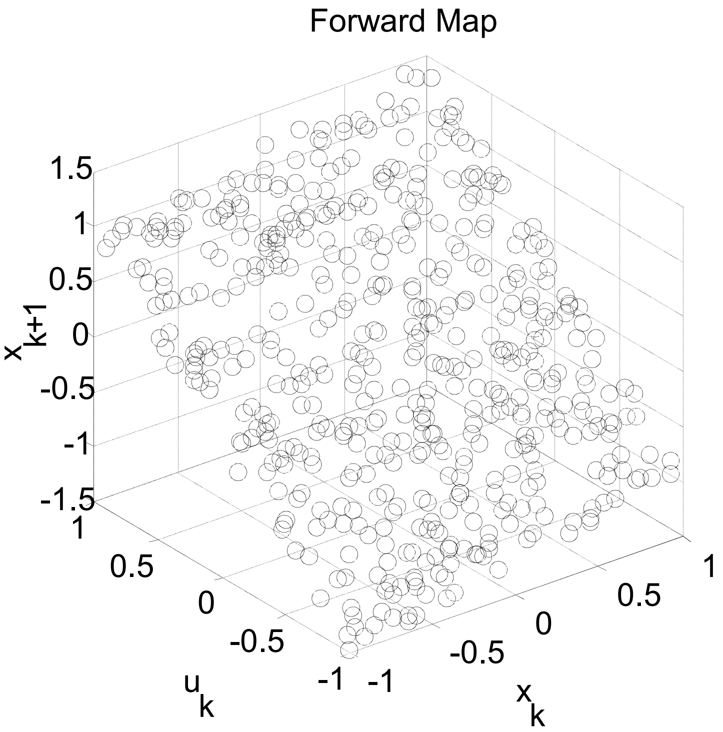
Choose u(k)

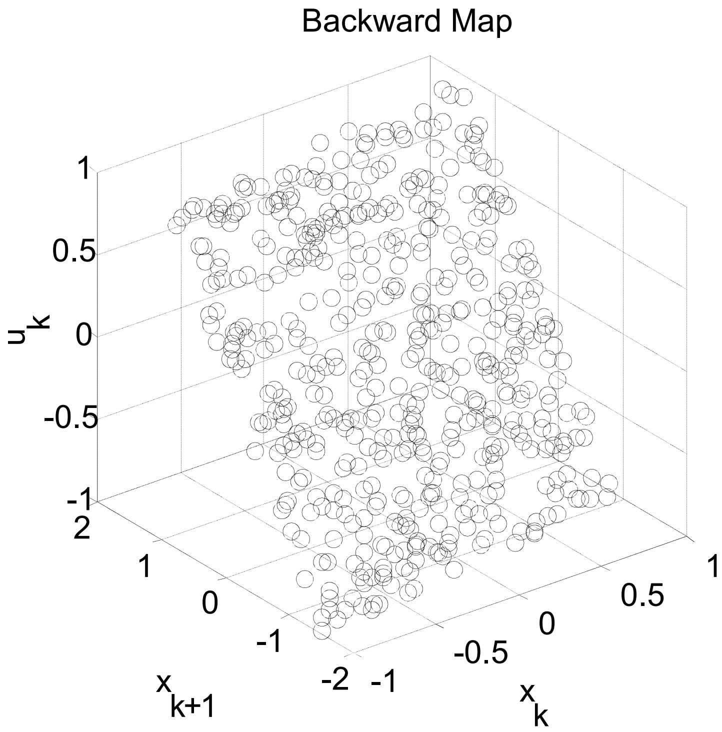
$$x(k+n) = A^n x(k) + W_c U$$

$$U = W_c^{-1} x(k+n) - W_c^{-1} A^n x(k)$$

Applications of Neural Networks

Neurocontrol Structures - Offline synthesis of NNC





SYSTEM: x(k+1) =f(x(k))+g(x(k))u(k)

SYSTEM: x(k+1) =0.3x(k)+1.2u(k)

Applications of Neural Networks

Neurocontrol Structures - Offline synthesis of NNC

$$\dot{c}_1 = -c_1 w + c_1(1-c_2)e^{\frac{c_2}{0.48}}$$

$$\dot{c}_2 = -c_2 w + c_1(1-c_2)e^{\frac{c_2}{0.48}} \frac{1.02}{1.02-c_2}$$

This is the dynamic model of a bioreactor.

$$c_1(k+1) = c_1(k) + \Delta\left(-c_1(k)w(k) + c_1(k)(1-c_2(k))e^{\frac{c_2(k)}{0.48}} \right)$$

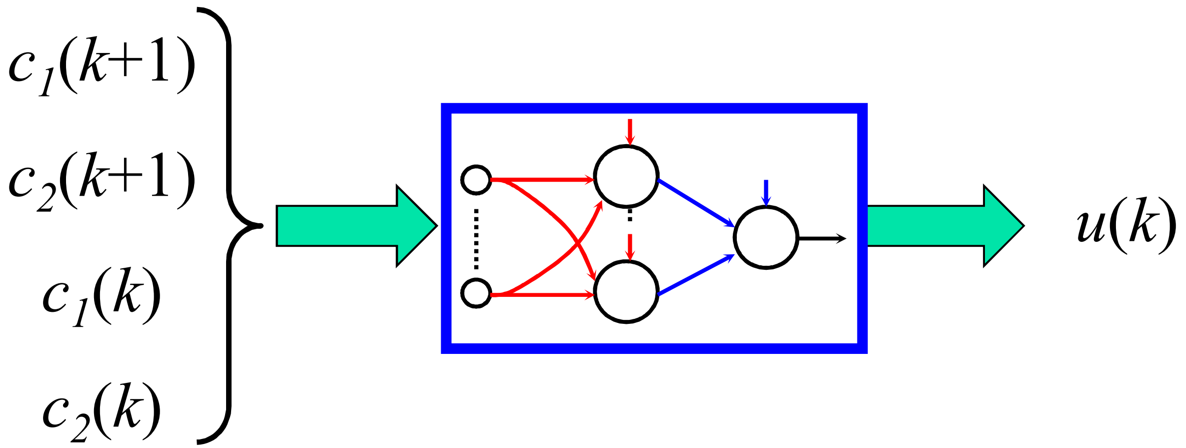
$$c_2(k+1) = c_2(k) + \Delta\left(-c_2(k)w(k) + c_1(k)(1-c_2(k))e^{\frac{c_2(k)}{0.48}} \frac{1.02}{1.02-c_2(k)} \right)$$

Applications of Neural Networks

Neurocontrol Structures - Offline synthesis of NNC

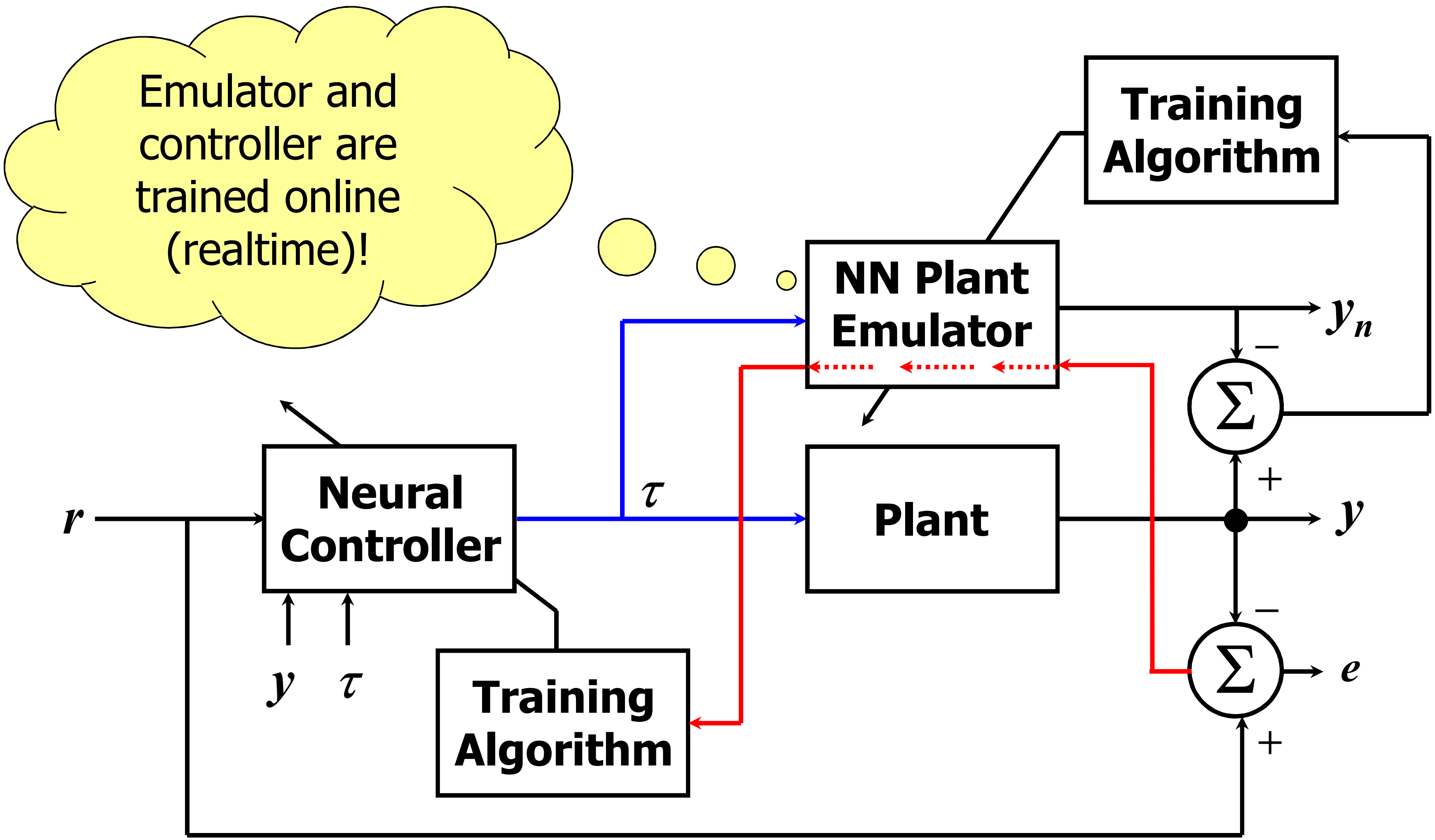
$$c_1(k+1) = c_1(k) + \Delta\left(-c_1(k)w(k) + c_1(k)(1-c_2(k))e^{\frac{c_2(k)}{0.48}} \right)$$

$$c_2(k+1) = c_2(k) + \Delta\left(-c_2(k)w(k) + c_1(k)(1-c_2(k))e^{\frac{c_2(k)}{0.48}} \frac{1.02}{1.02 - c_2(k)} \right)$$



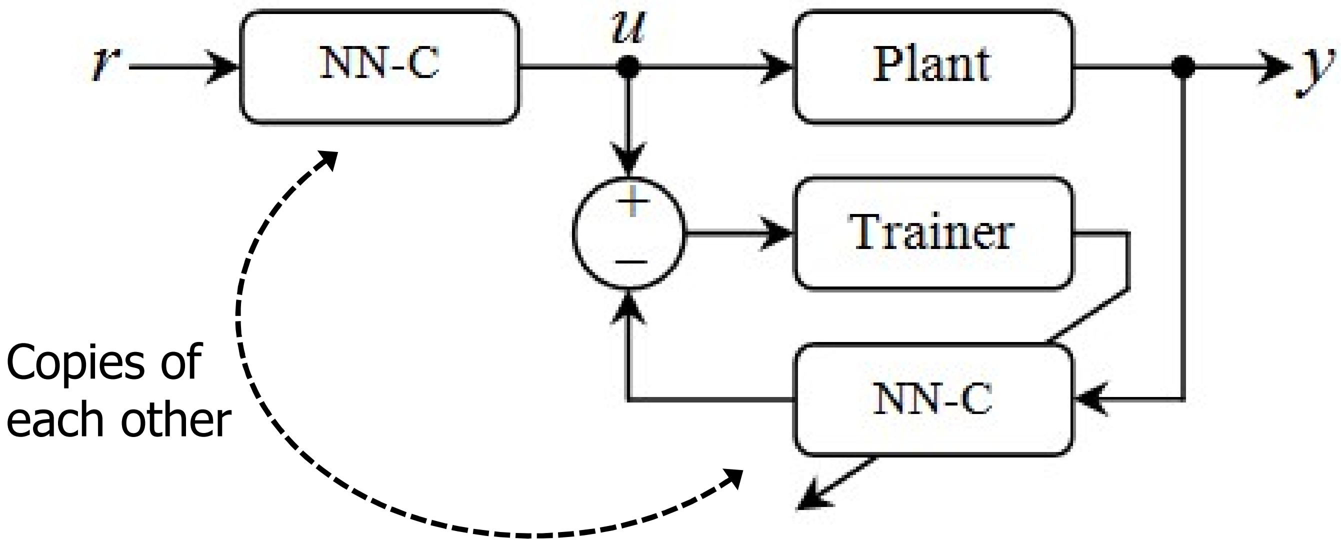
Applications of Neural Networks

Neurocontrol Structures - Indirect Adaptive Control



Applications of Neural Networks

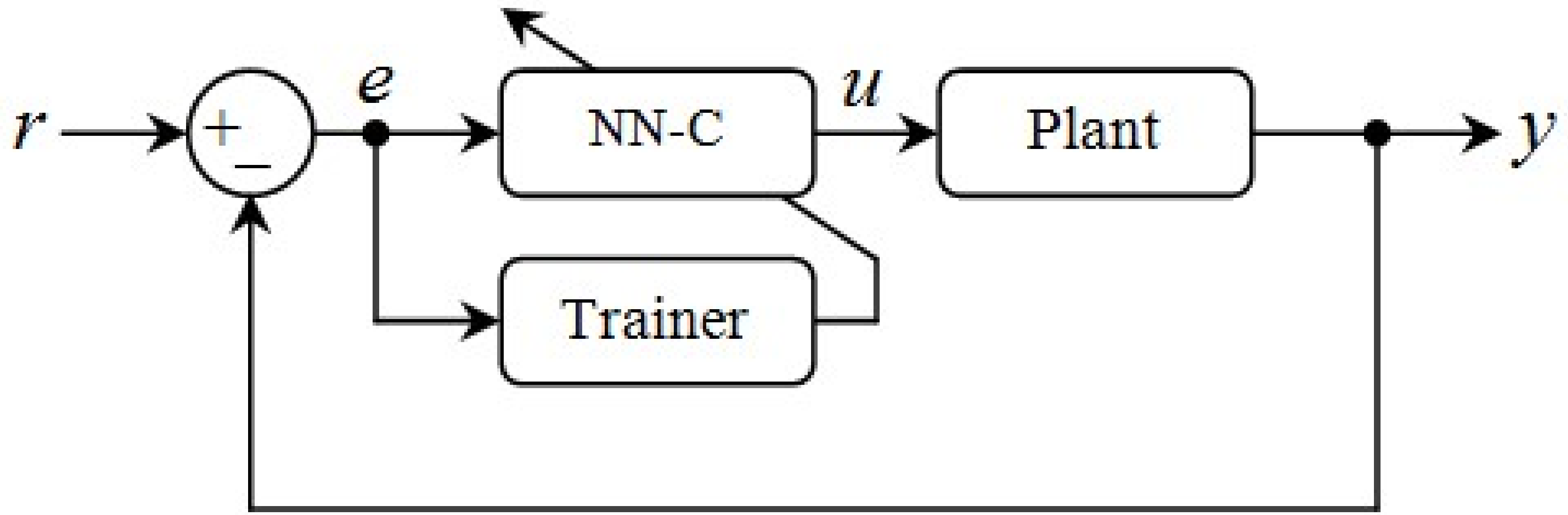
Neurocontrol Structures - Indirect learning architecture



G.W. Ng, Application of Neural Networks to Adaptive Control of Nonlinear Systems, Research Studies Press, Somerset, England, 1997.

Applications of Neural Networks

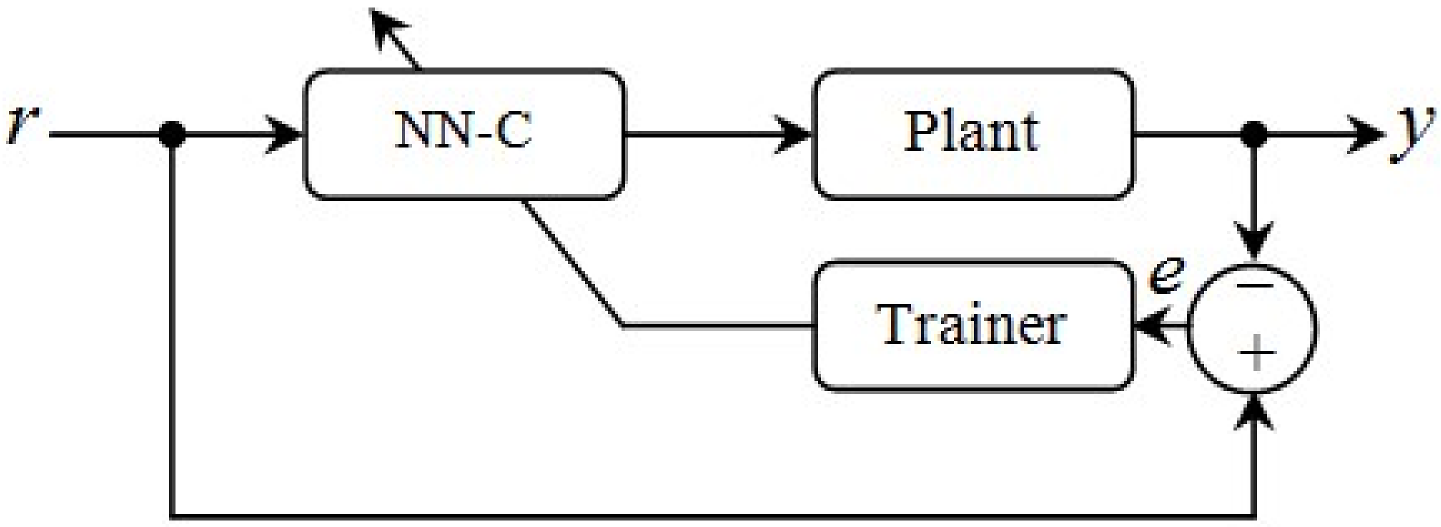
Neurocontrol Structures - Closed loop direct inverse control



G.W. Ng, Application of Neural Networks to Adaptive Control of Nonlinear Systems, Research Studies Press, Somerset, England, 1997.

Applications of Neural Networks

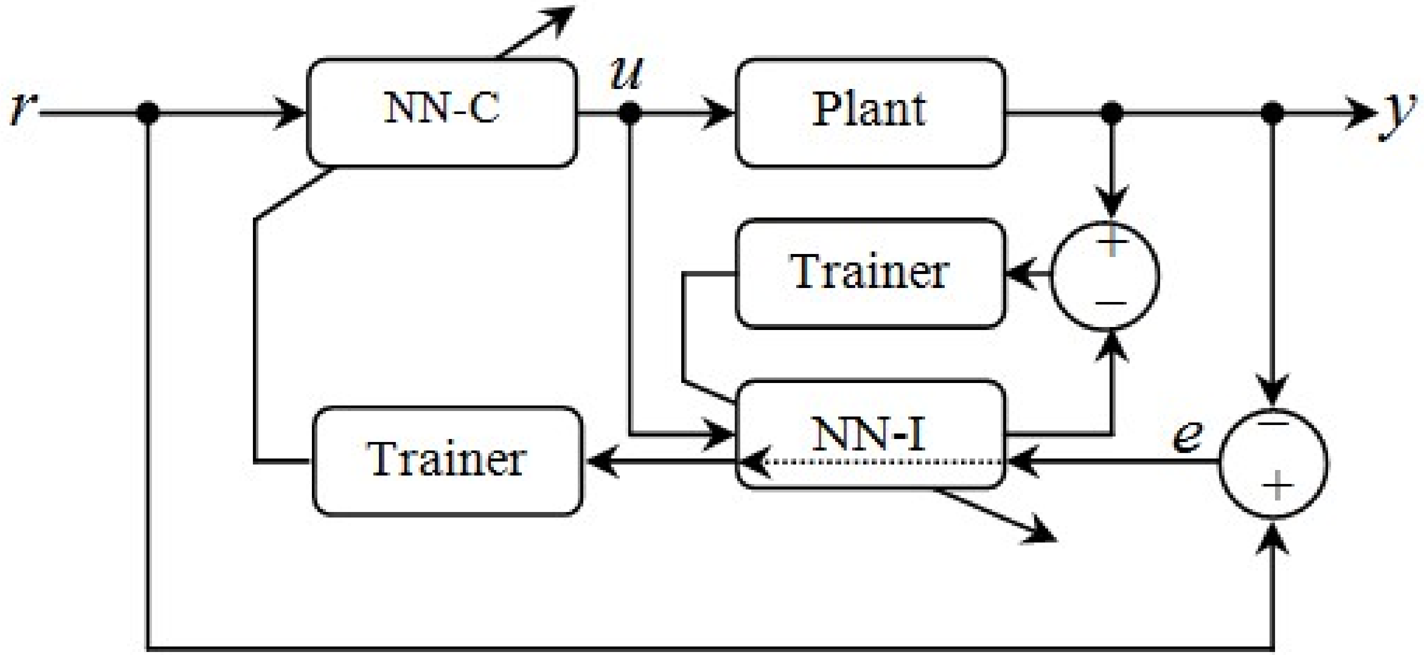
Neurocontrol Structures - Specialized learning architecture



G.W. Ng, Application of Neural Networks to Adaptive Control of Nonlinear Systems, Research Studies Press, Somerset, England, 1997.

Applications of Neural Networks

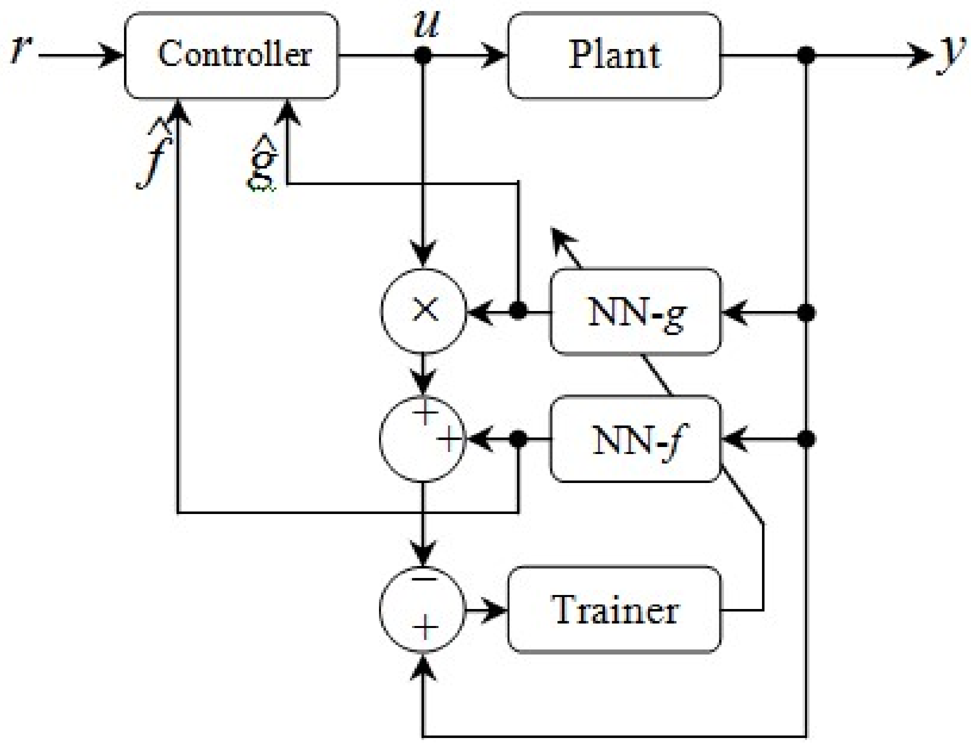
Neurocontrol Structures - Indirect Adaptive Control Scheme



G.W. Ng, Application of Neural Networks to Adaptive Control of Nonlinear Systems, Research Studies Press, Somerset, England, 1997.

Applications of Neural Networks

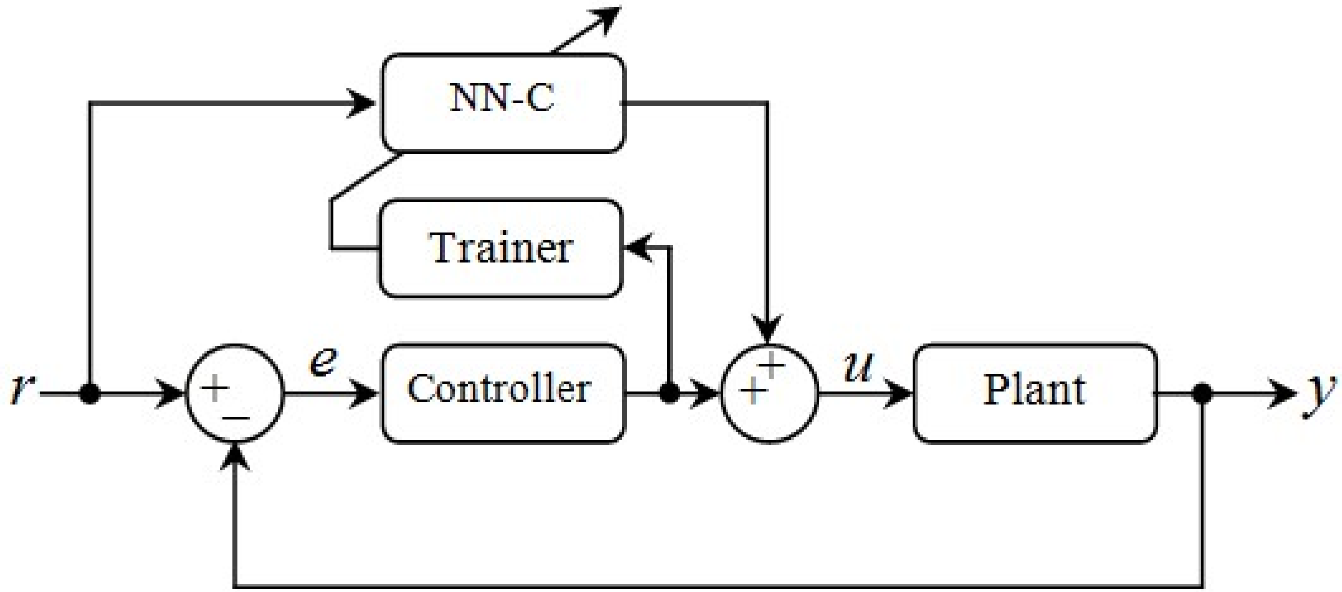
Neurocontrol Structures - Feedback linearization via neural networks



G.W. Ng, Application of Neural Networks to Adaptive Control of Nonlinear Systems, Research Studies Press, Somerset, England, 1997.

Applications of Neural Networks

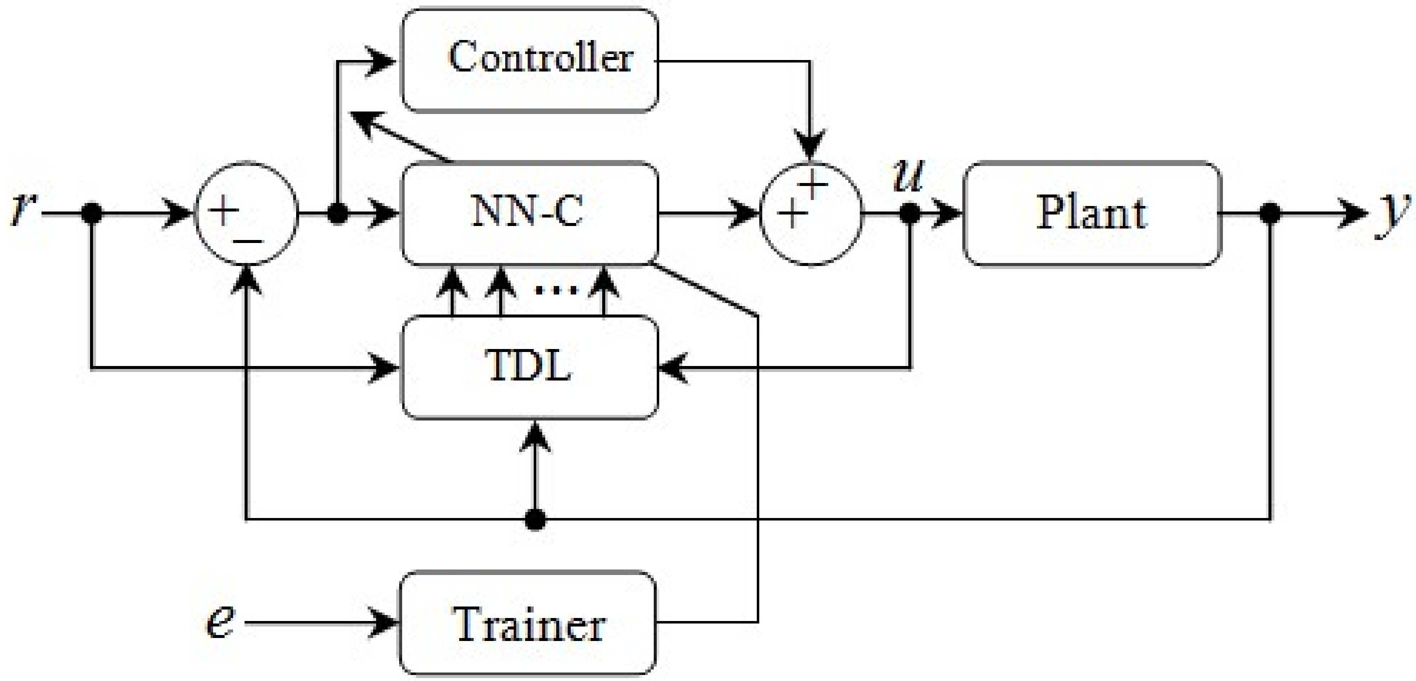
Neurocontrol Structures - Feedback error learning architecture



G.W. Ng, Application of Neural Networks to Adaptive Control of Nonlinear Systems, Research Studies Press, Somerset, England, 1997.

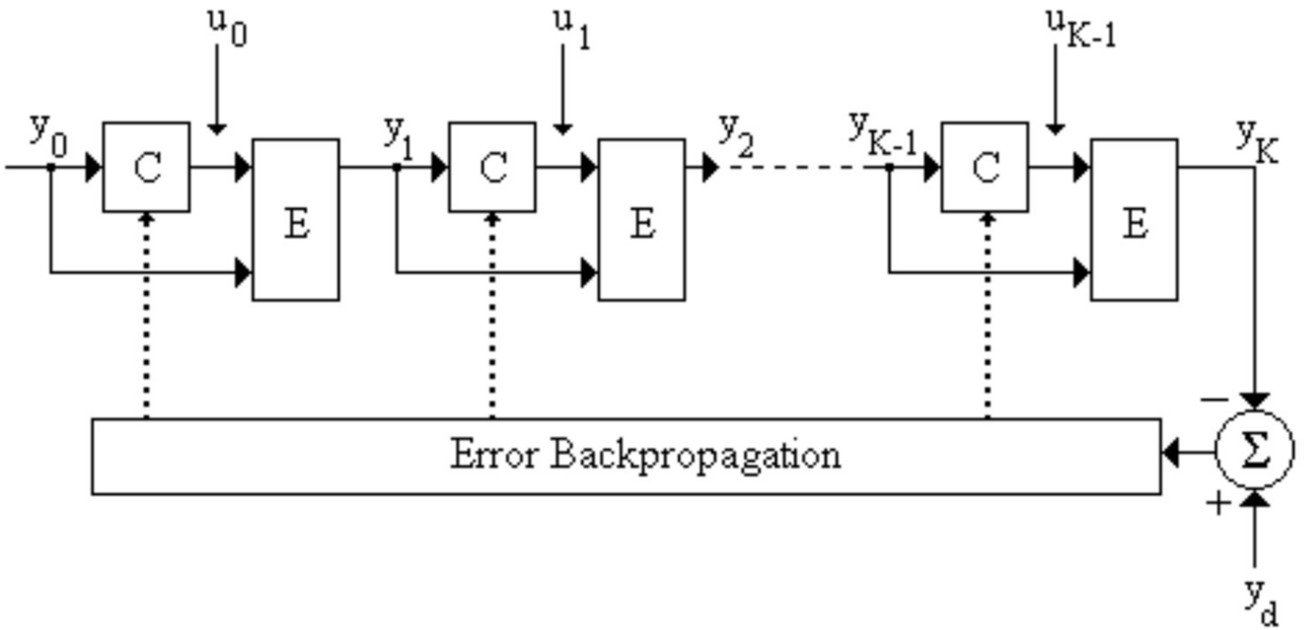
Applications of Neural Networks

Neurocontrol Structures - typical neural network based control architecture



G.W. Ng, Application of Neural Networks to Adaptive Control of Nonlinear Systems, Research Studies Press, Somerset, England, 1997.

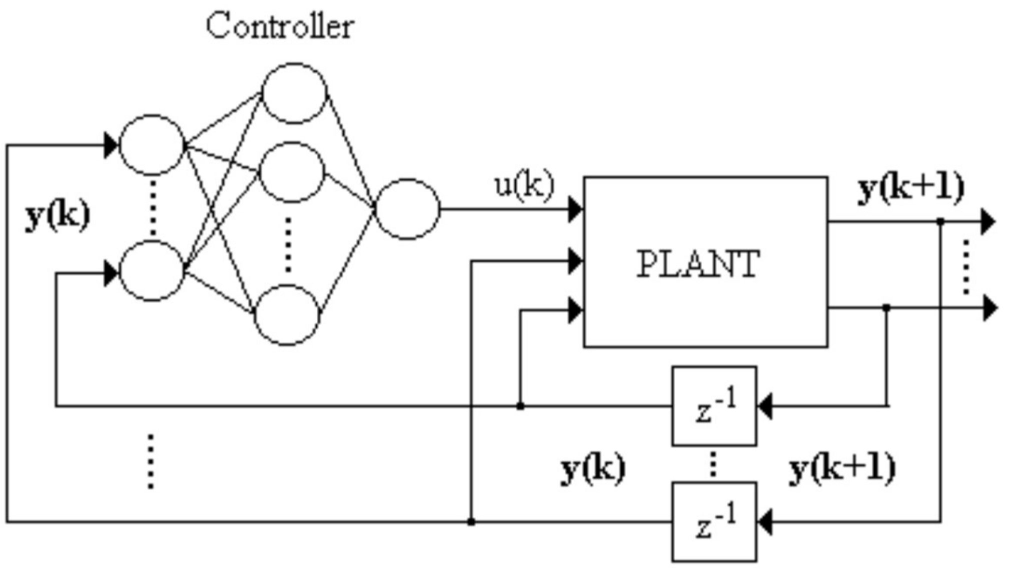
Applications of Neural Networks
Neurocontrol Structures – Self Learning Control



- Controller training architecture

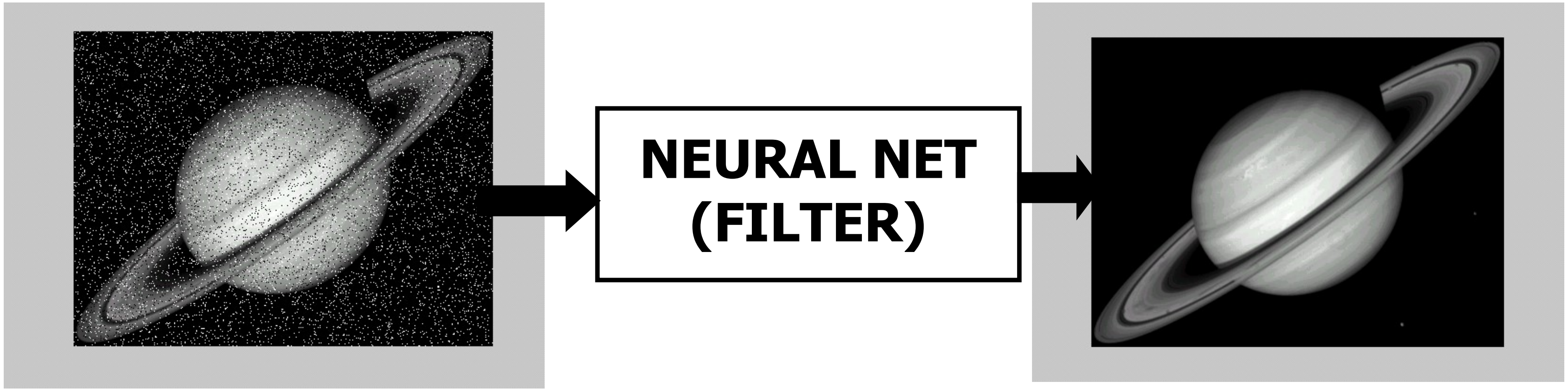
Mehmet Önder Efe, *Neural Networks*, Lecture Notes, 2022.

Applications of Neural Networks
Neurocontrol Structures – Self Learning Control



❑ Feedback loop structure

Applications of Neural Networks
Noise Elimination

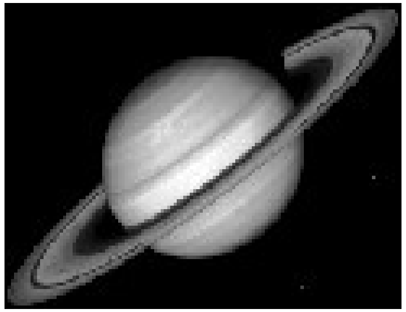


- How do we filter out the noise from the source?
- How do we teach *what to filter out* and *how to filter out* ?

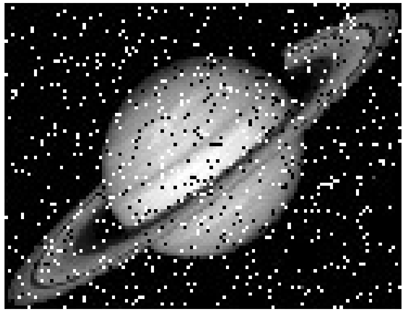
Applications of Neural Networks

Noise Elimination

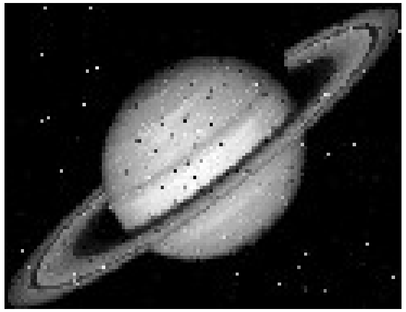
Original Image



Noisy Image

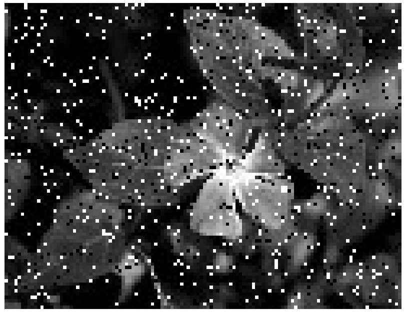


Filtered Image



Original Image

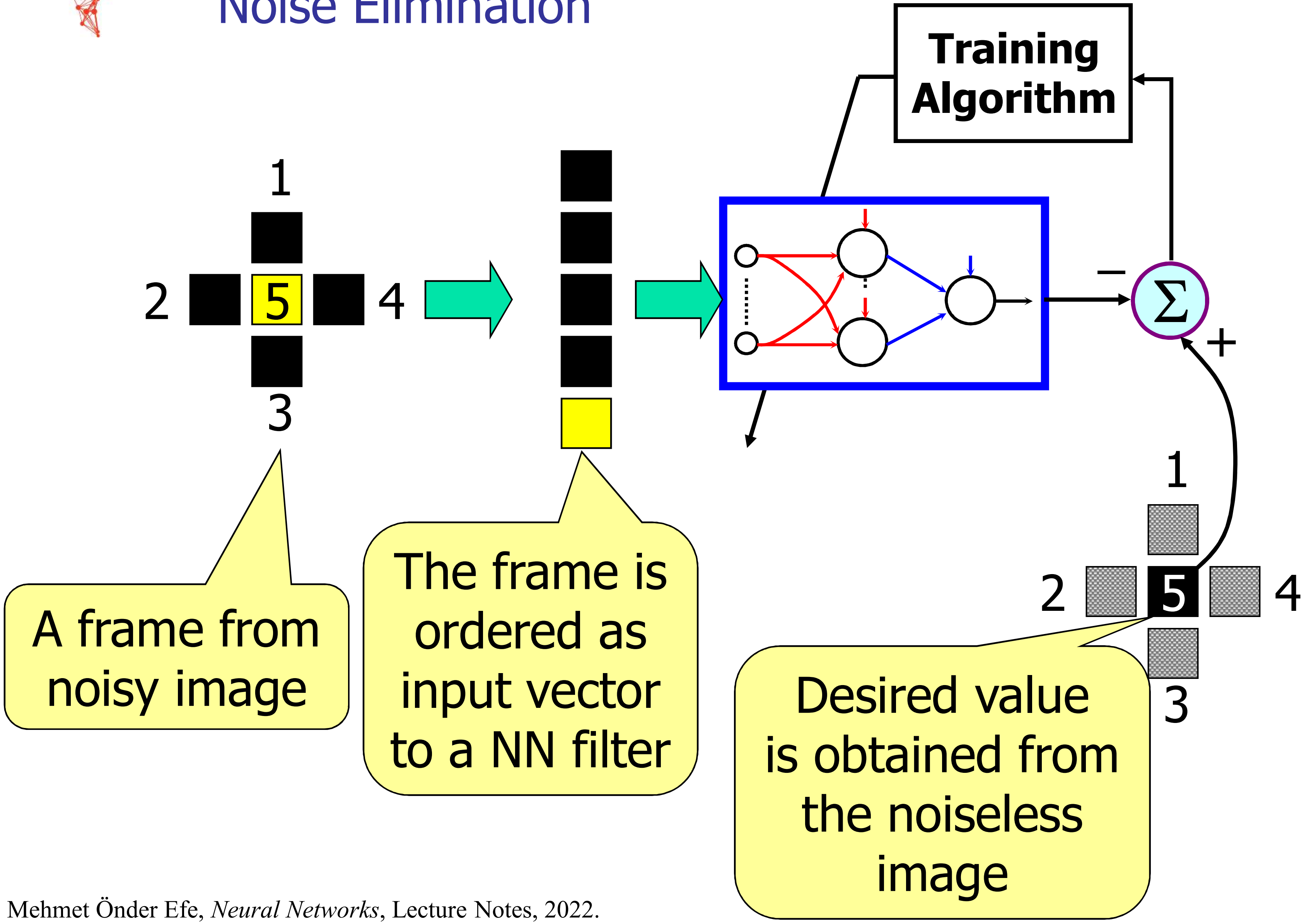
Noisy Image



Filtered Image

Applications of Neural Networks

Noise Elimination



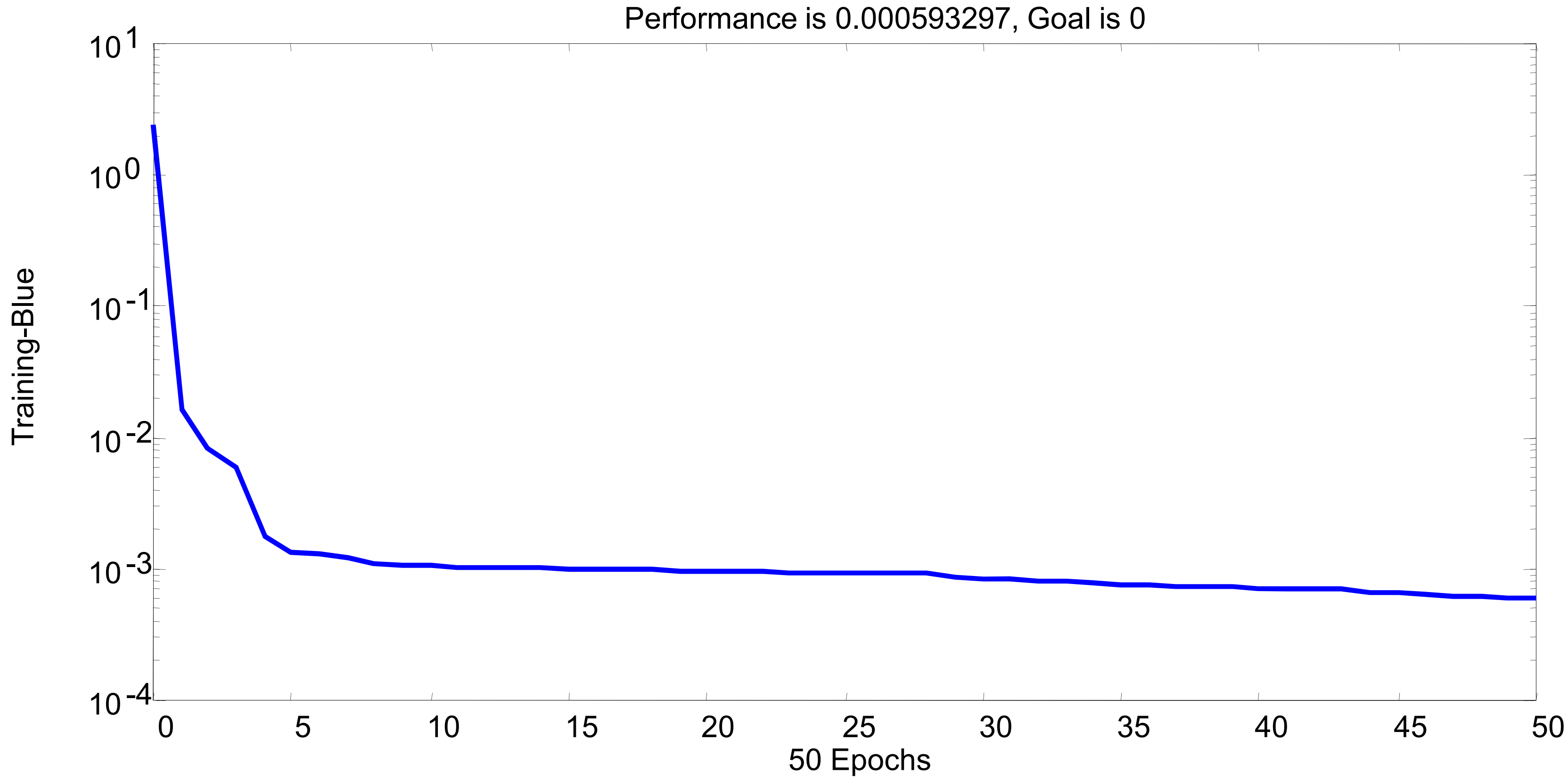
Applications of Neural Networks
Noise Elimination

- Scan the image (all rows, all columns)
- At every frame, reorder to form input vector
- Choose the corresponding output from the original image
- Train the neural network

- I trained the network for saturn image (offline training)
- Tested also for the Vinca image to show this filter is not specific to Saturn image only! i.e. no memorization
- The NN has 5-10-1 structure with sigmoidal nonlinearity for the hidden neurons, output neuron is linear
- 2000 Training patterns have been selected randomly
- Training continued for 50 epoches
- MSE decreased to 0.000593297

Applications of Neural Networks

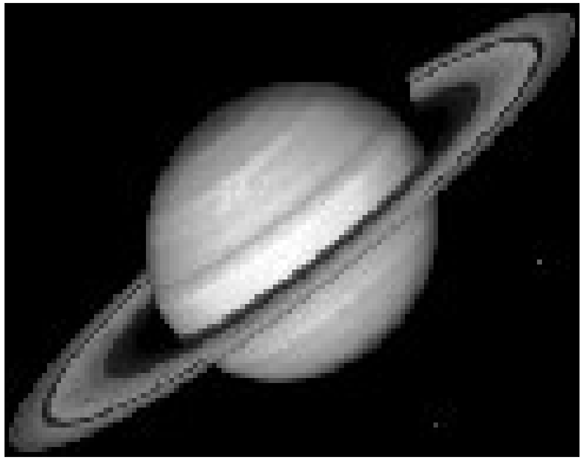
Noise Elimination - Training stage



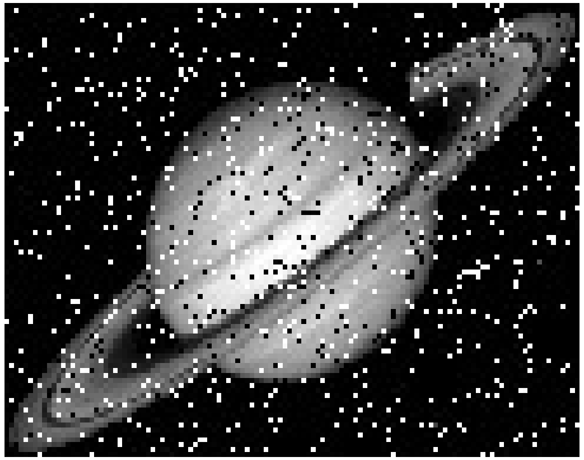
Applications of Neural Networks

Noise Elimination - Compare again...

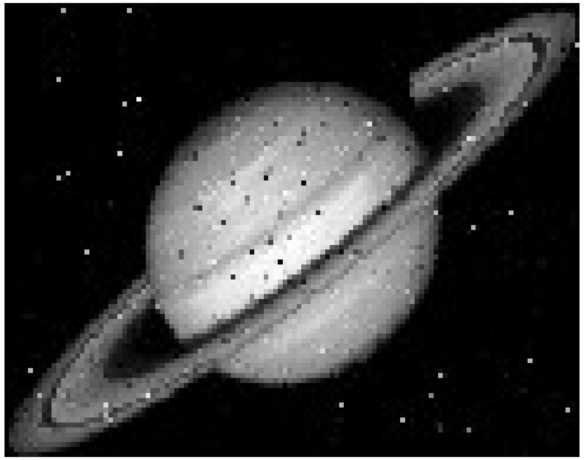
Original Image



Noisy Image

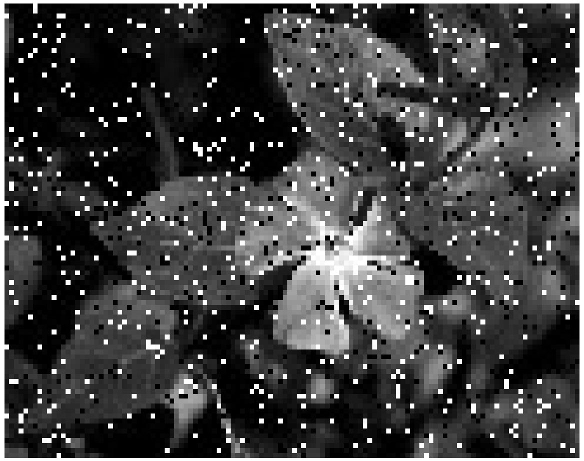


Filtered Image



Original Image

Noisy Image



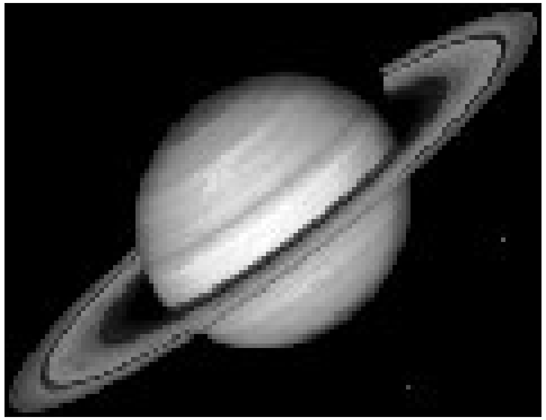
Filtered Image

Mehmet Önder Efe, *Neural Networks*, Lecture Notes, 2022.

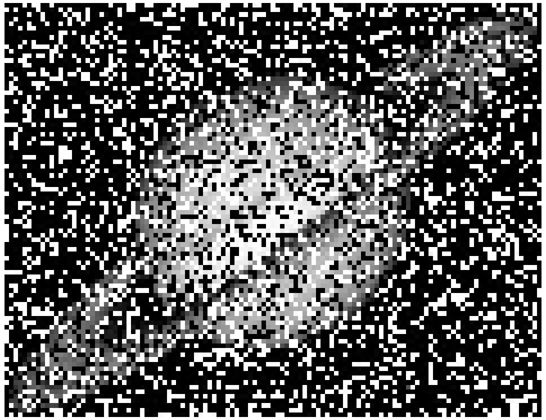
Applications of Neural Networks

Noise Elimination - Same NN, Higher Noise Level

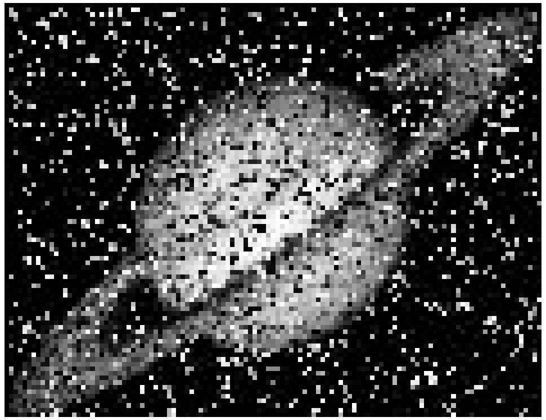
Original Image



Noisy Image

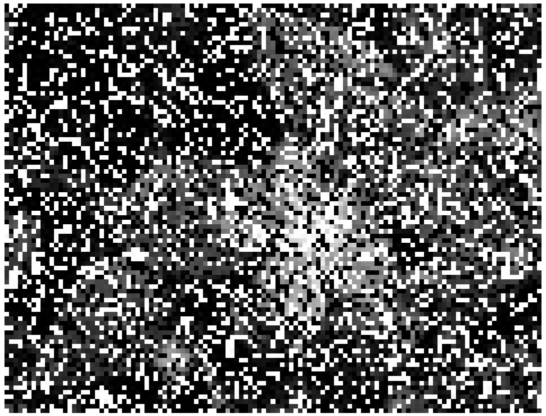


Filtered Image

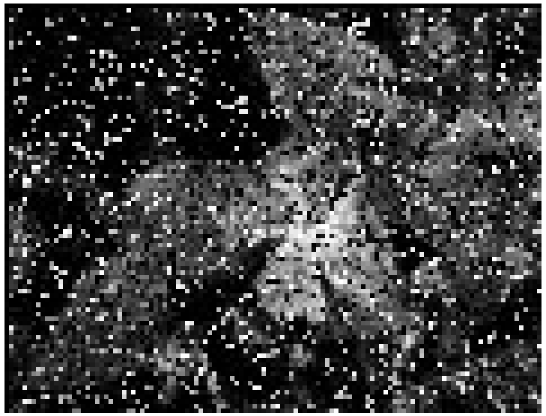


Original Image

Noisy Image



Filtered Image

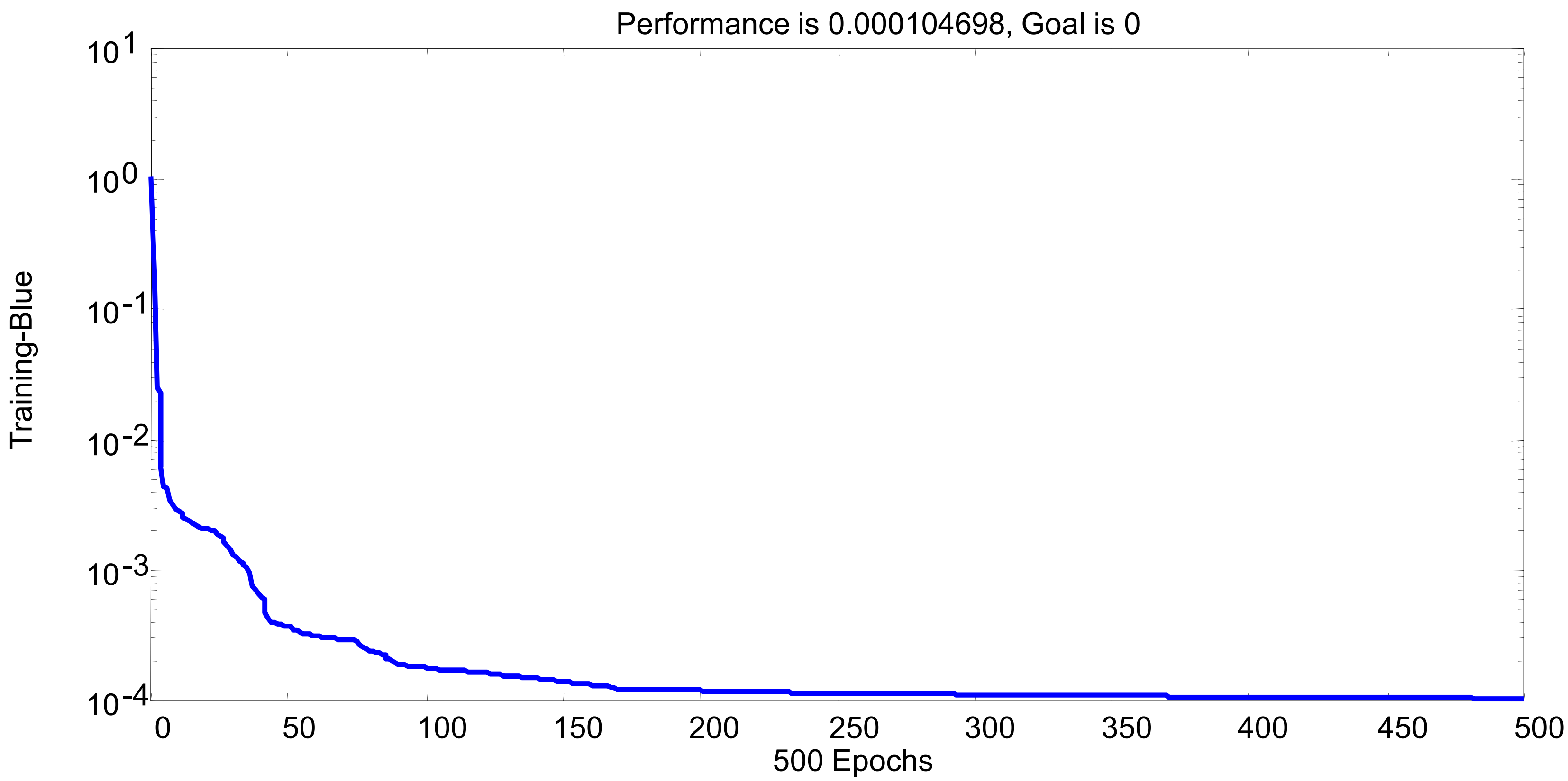


Applications of Neural Networks

Noise Elimination - Another train/test

- Neural Network Structure 5-5-5-1
- First hidden layer has hyperbolic tangent activation fcns.
- Second hidden layer has sigmoidal activation fcns.
- Output layer has a linear neuron
- I trained the network for saturn image
- Tested also for the Vinca image to show this filter is not specific to Saturn image only! i.e. no memorization
- 2000 Training patterns have been selected randomly
- Training continued for 500 epoches
- MSE decreased to 0.000104698
- Training noise density was 0.1
- Test noise density was 0.5

Applications of Neural Networks
Noise Elimination

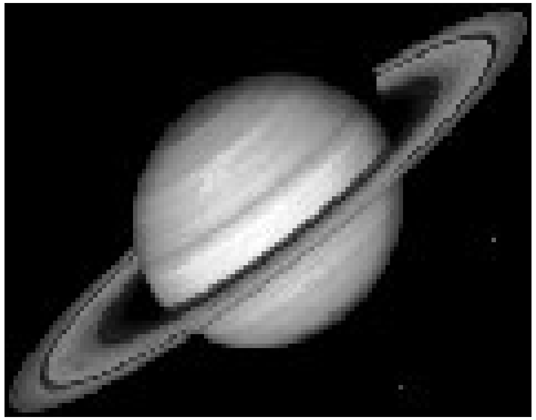


Mehmet Önder Efe, *Neural Networks*, Lecture Notes, 2022.

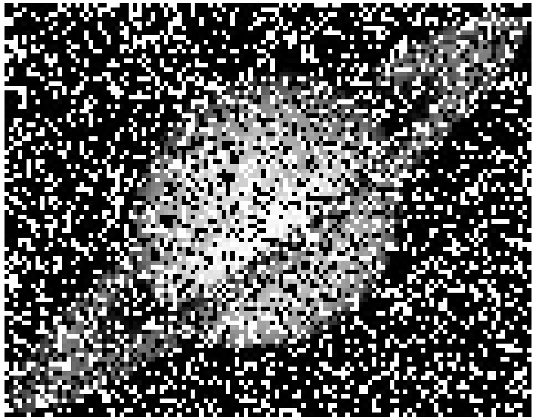
Applications of Neural Networks

Noise Elimination

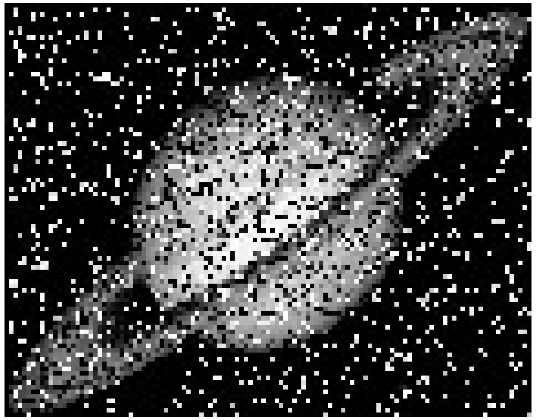
Original Image



Noisy Image

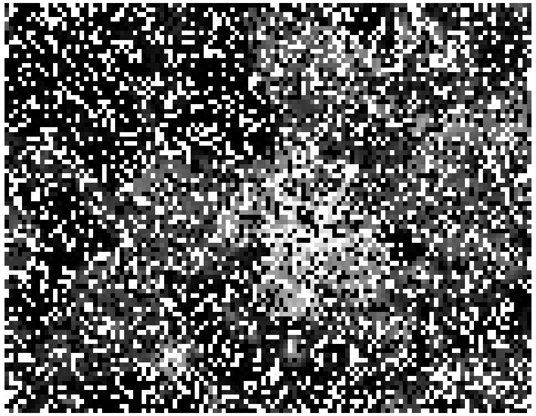


Filtered Image

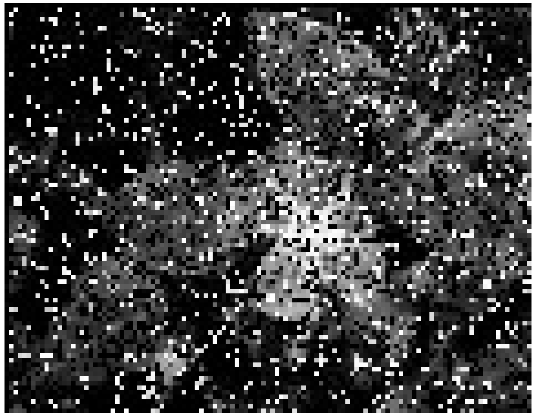


Original Image

Noisy Image

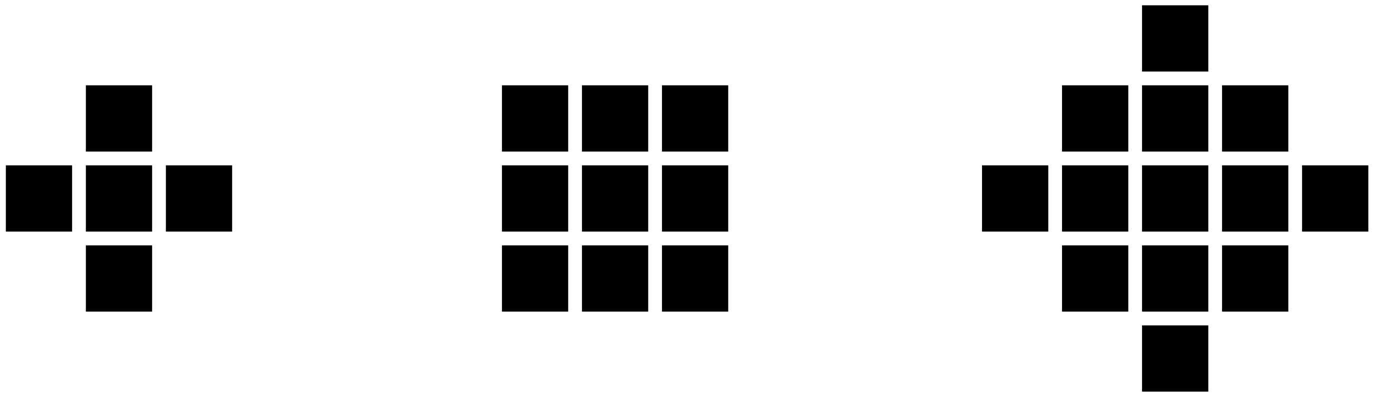


Filtered Image



Mehmet Önder Efe, *Neural Networks*, Lecture Notes, 2022.

Applications of Neural Networks
Alternatives

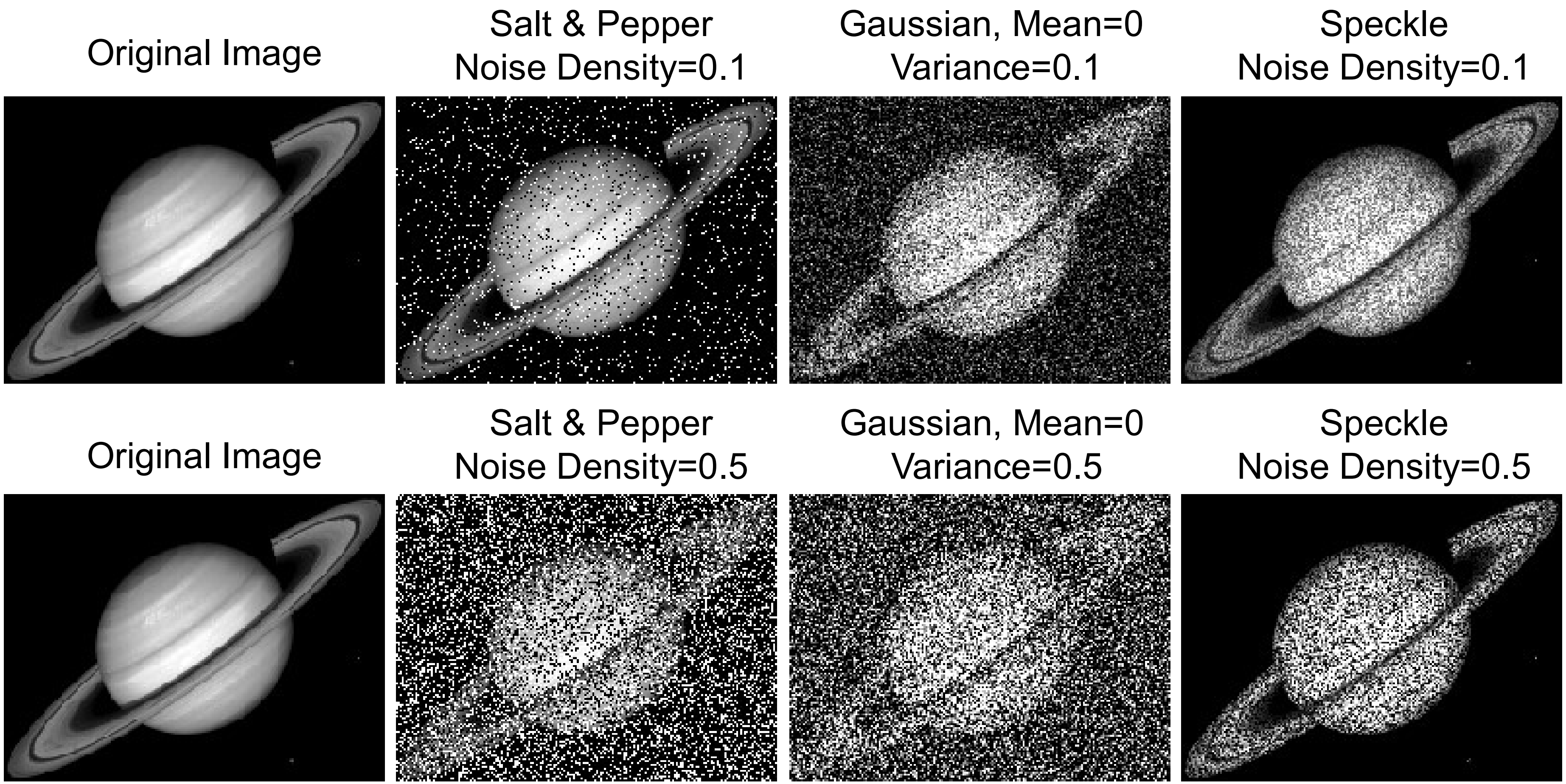


5-input NN 9-input NN 13-input NN

- Different frames can be considered
- Computational complexity (i.e. processing time) changes!
- You may use other available techniques of image processing

Applications of Neural Networks

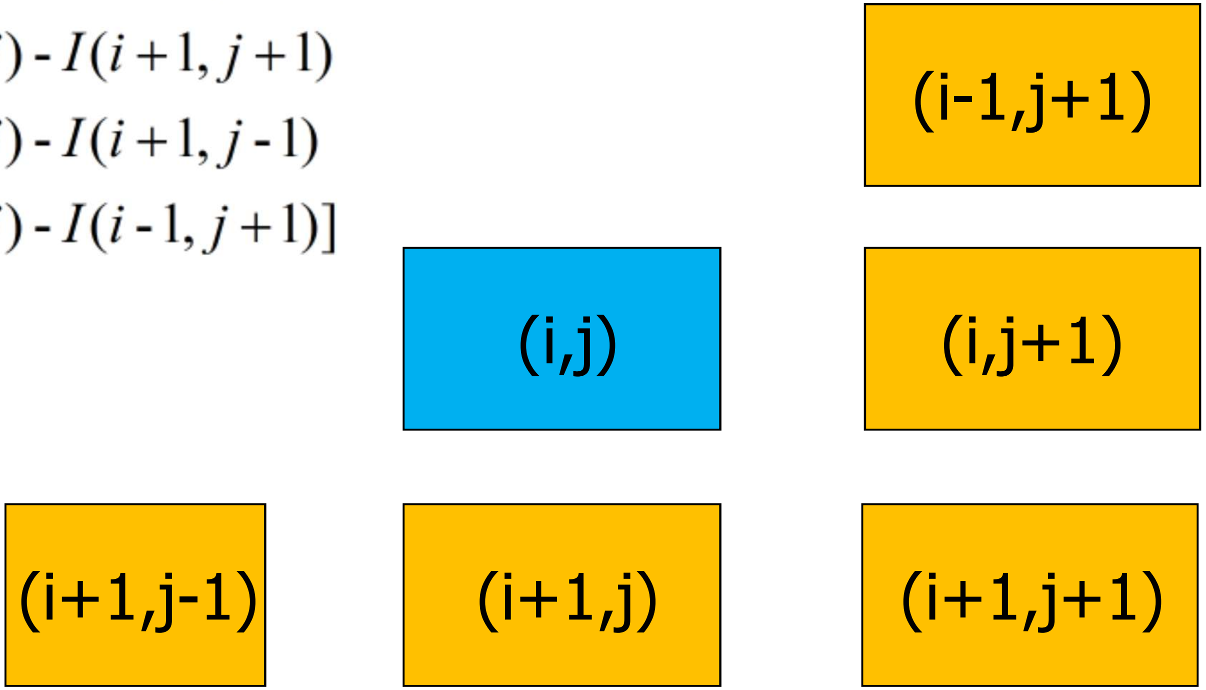
Different Noise Types...



Applications of Neural Networks

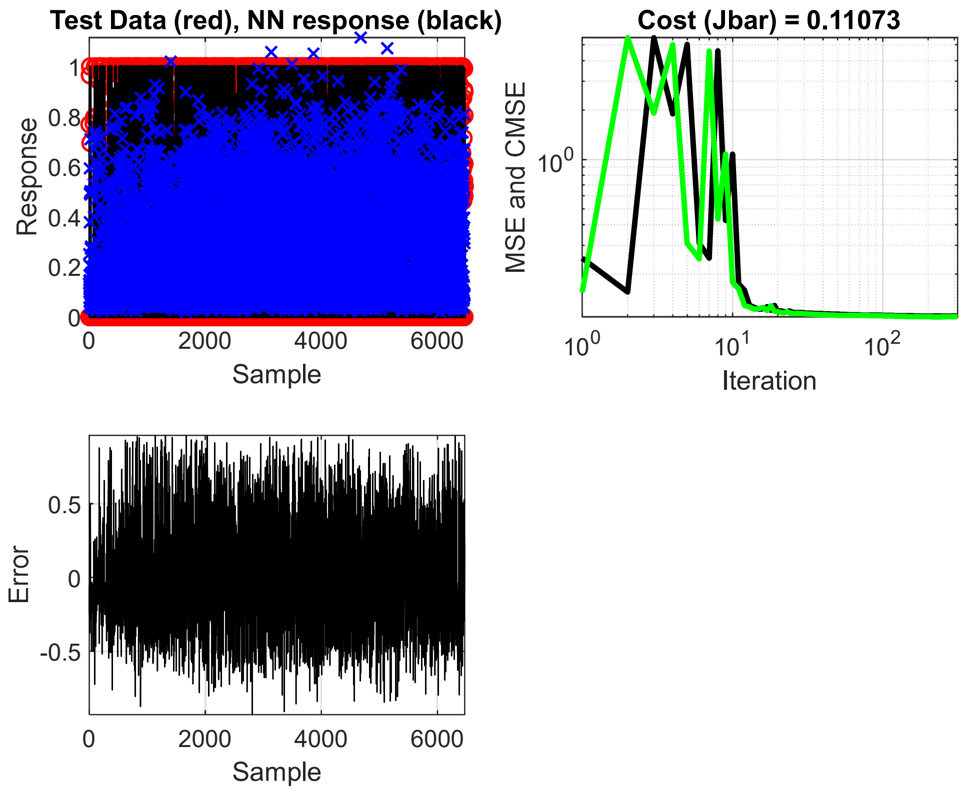
Edge Detection (Canny Edge Detector)

$$
\begin{aligned}
input = [\,&I(i,j) - I(i,j+1) \\
&I(i,j) - I(i+1,j) \\
&I(i,j) - I(i+1,j+1) \\
&I(i,j) - I(i+1,j-1) \\
&I(i,j) - I(i-1,j+1)\,]
\end{aligned}
$$



Applications of Neural Networks

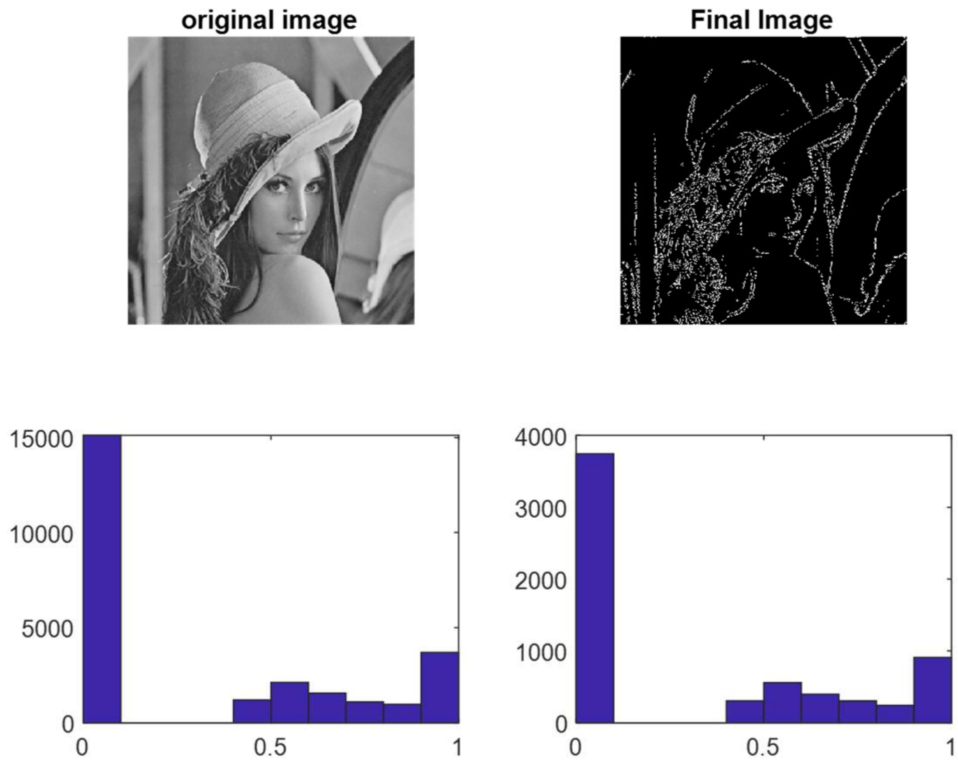
Edge Detection (Canny Edge Detector), FNN



5-12-1 NN, tanh/linear structure

Applications of Neural Networks

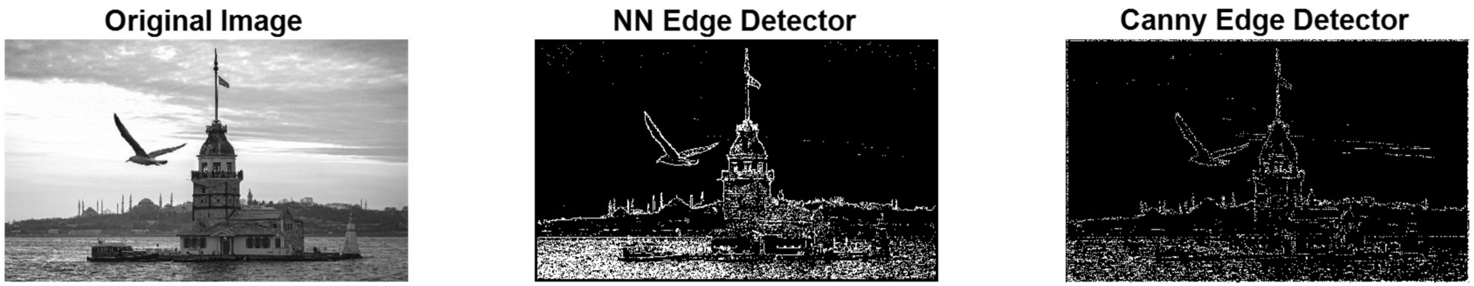
Edge Detection (Canny Edge Detector), FNN



Training data

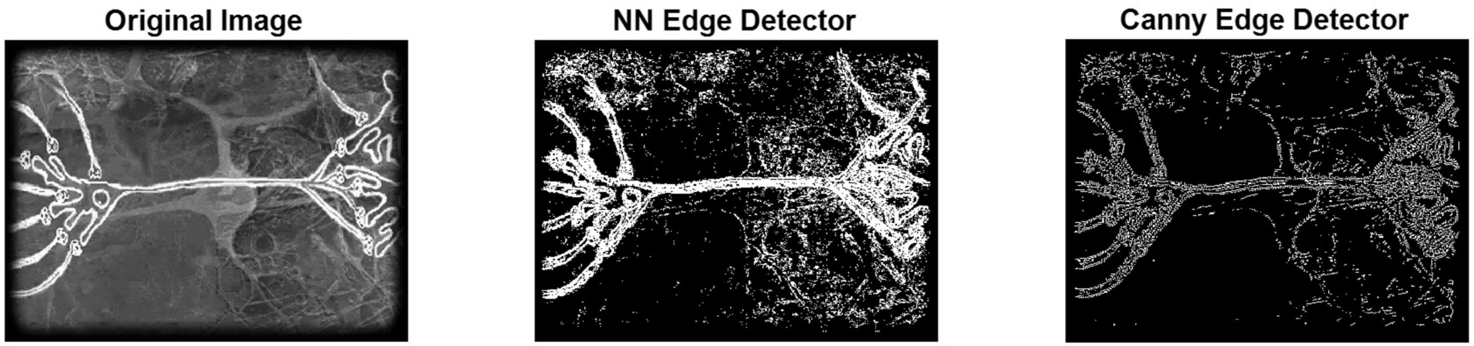
Applications of Neural Networks

Edge Detection (Canny Edge Detector),FNN



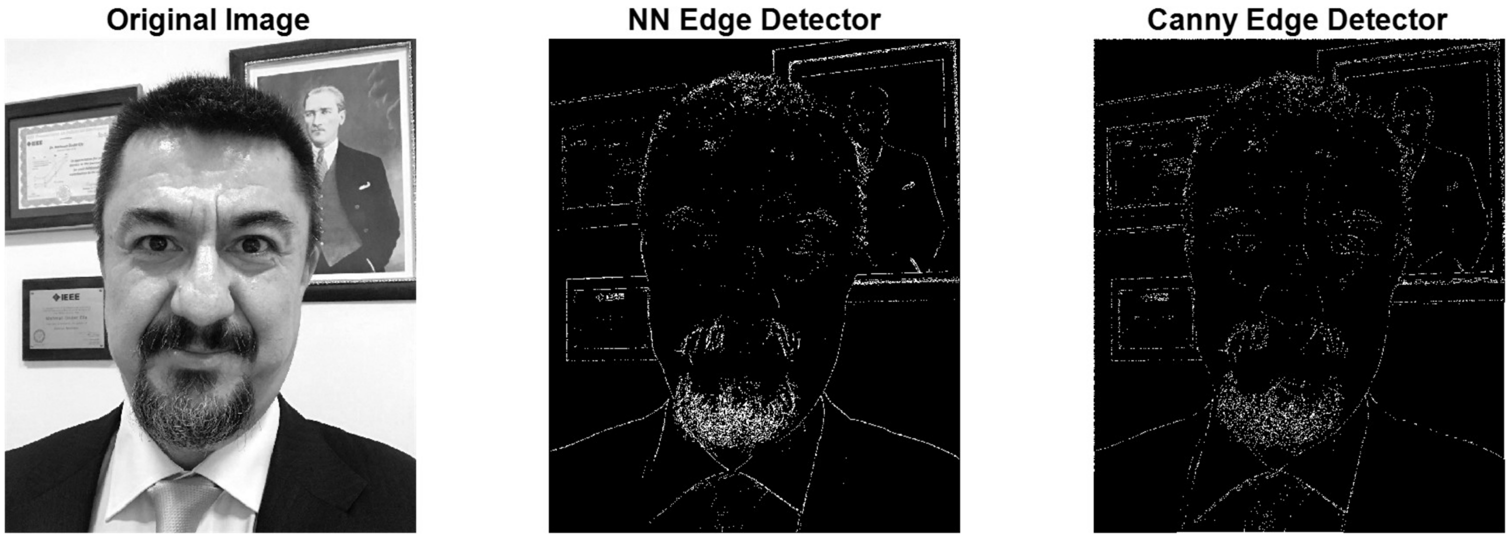
Applications of Neural Networks

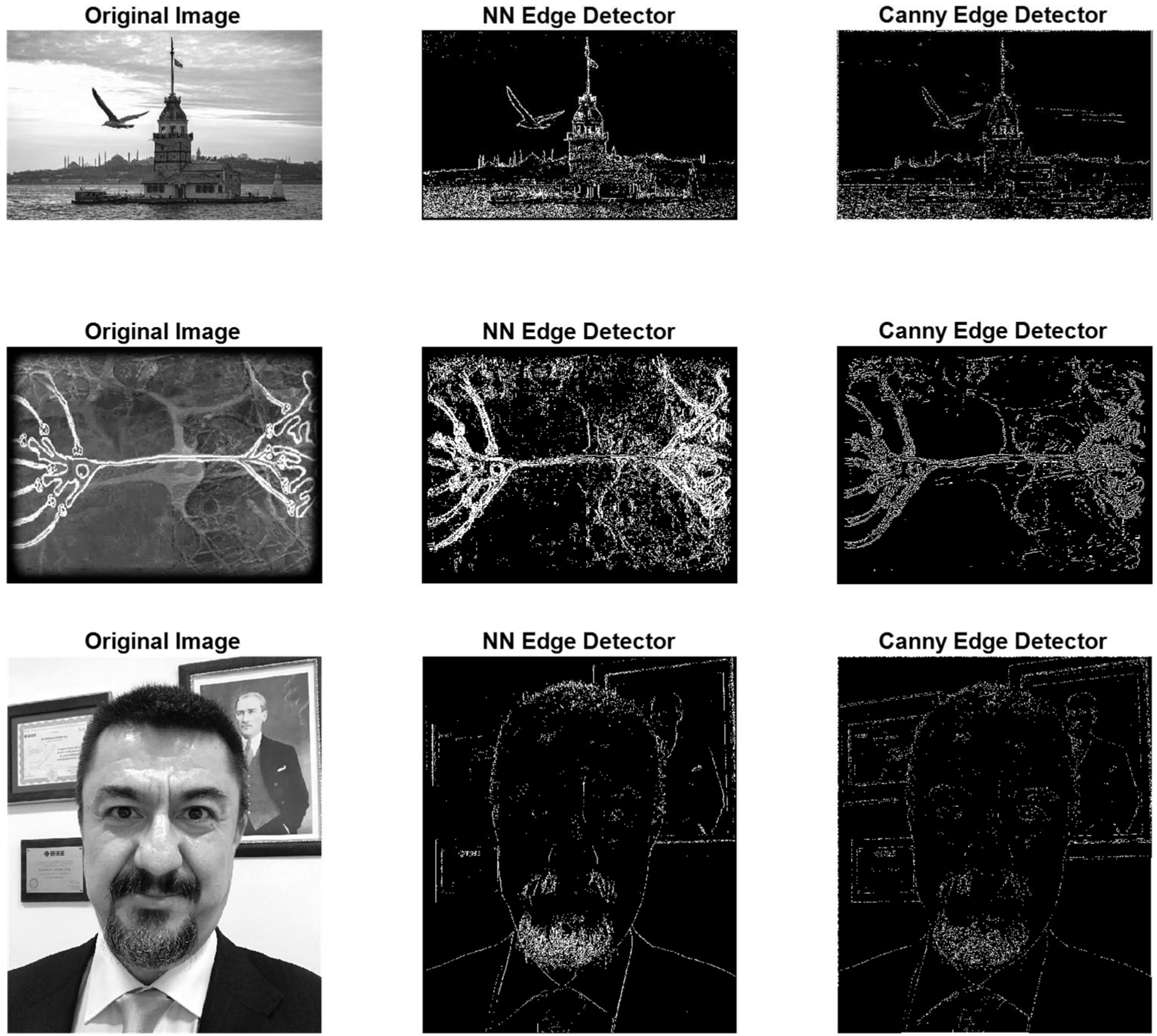
Edge Detection (Canny Edge Detector),FNN



Applications of Neural Networks

Edge Detection (Canny Edge Detector),FNN

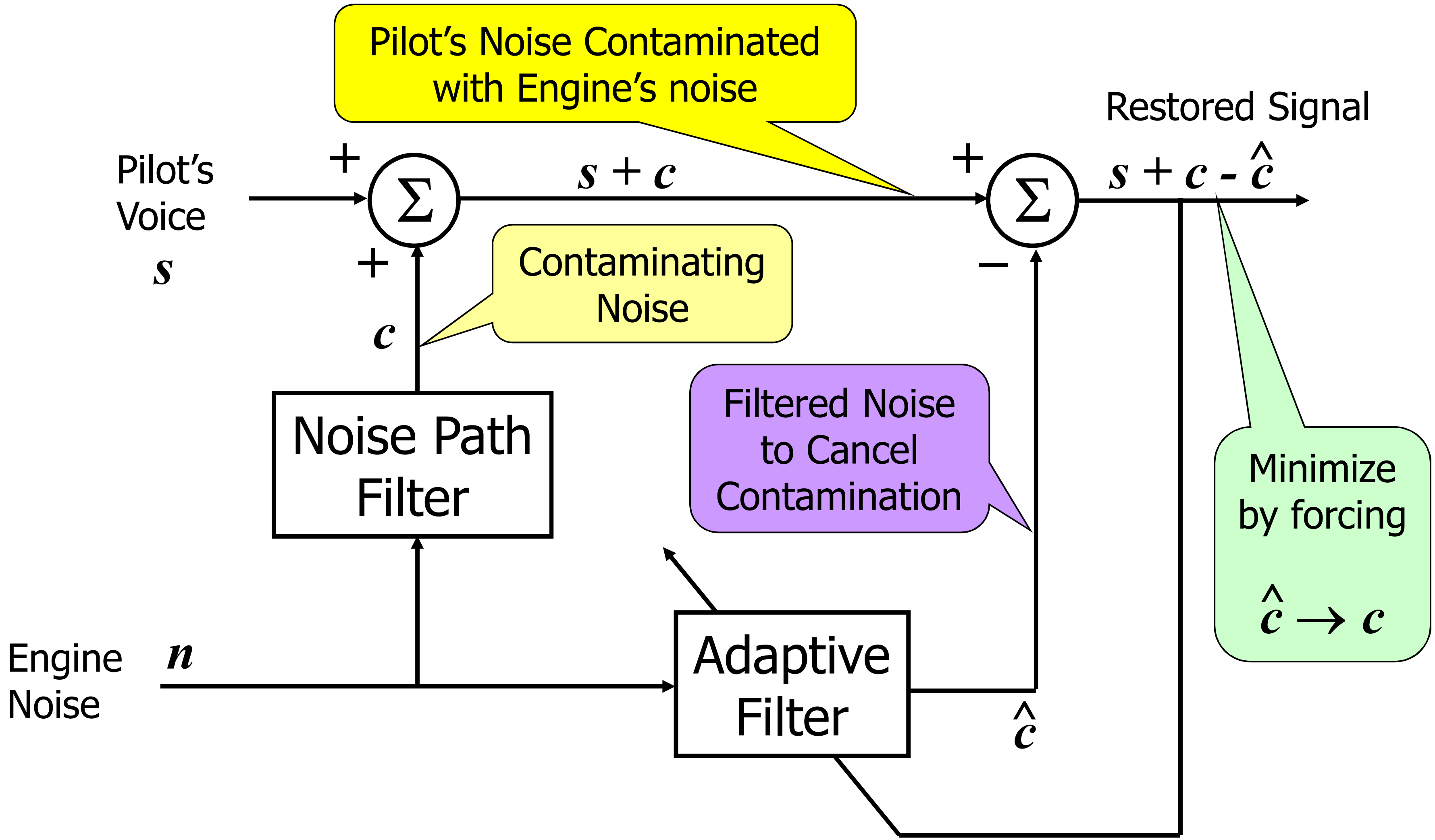




5-2-1 NN Structure

Applications of Neural Networks

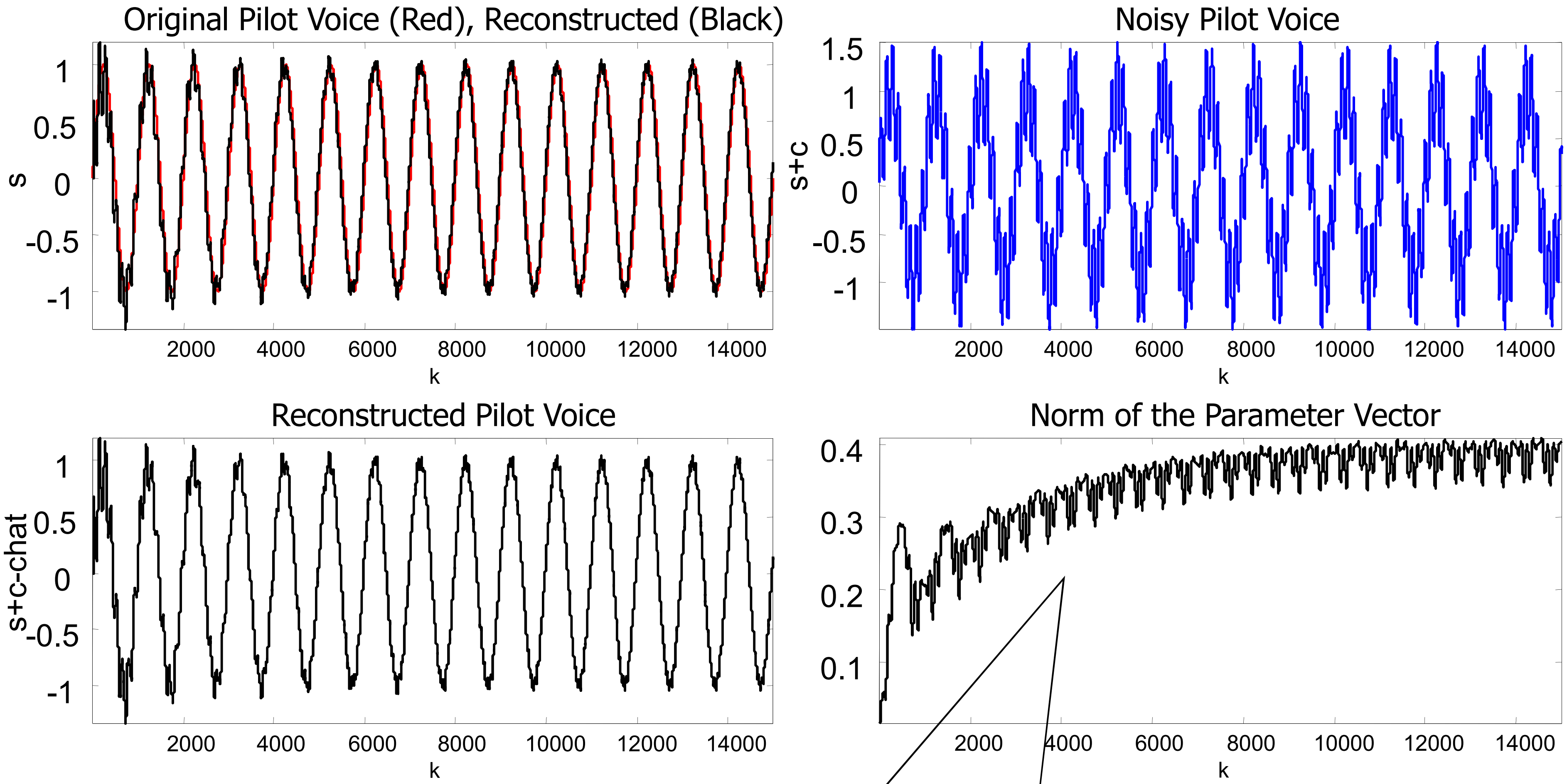
Adaptive Noise Cancellation



Mehmet Önder Efe, *Neural Networks*, Lecture Notes, 2022.

Applications of Neural Networks

Adaptive Noise Cancellation - Adaptive FIR Filter



Behavior is convergent!

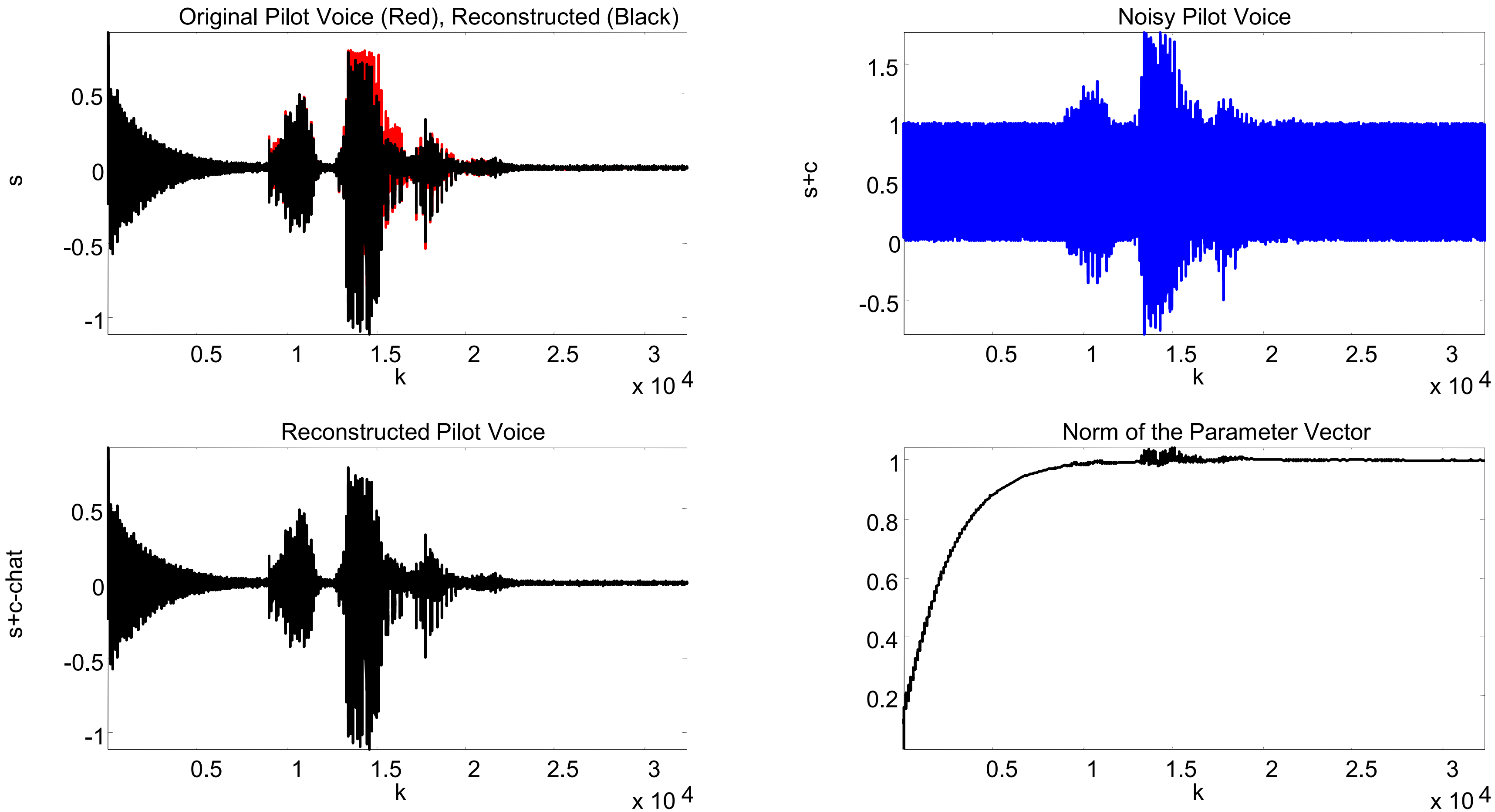
Mehmet Önder Efe, *Neural Networks*, Lecture Notes, 2022.

Applications of Neural Networks
Adaptive Noise Cancellation

- FIR Filter is composed of an ADALINE
- It has 25 inputs with a bias term
- EBP is used to tune (no momentum, no LR adaptation)
- A simple signal is chosen as the Pilot Voice
- Filter successfully reconstructs the noise and lets us have the Pilot Voice at the output
- Notice that the training is on-line here

Applications of Neural Networks

Adaptive Noise Cancellation



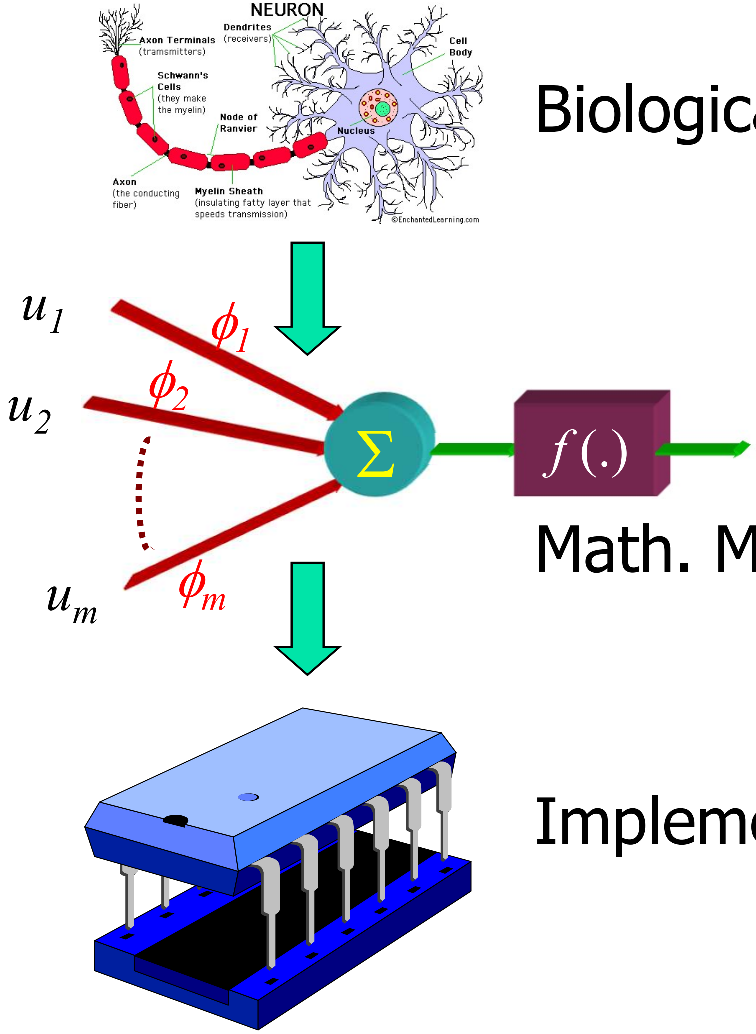
The Pilot says: Istanbul

Applications of Neural Networks
Adaptive Noise Cancellation

- FIR Filter is composed of an ADALINE
- It has 25 inputs with a bias term
- EBP is used to tune (no momentum, no LR adaptation)
- Pilot says: Istanbul
- Filter successfully reconstructs the noise and lets us have the Pilot Voice at the output
- Notice that the training is on-line here
- We also give the result with the final filter coefficients
- Listen Now...

"Istanbul"	Noisy "Istanbul"	On-line Filtered	With Final Coeffs.

Applications of Neural Networks
VLSI Implementation of Neural Networks



Biological Reality

Math. Modeling

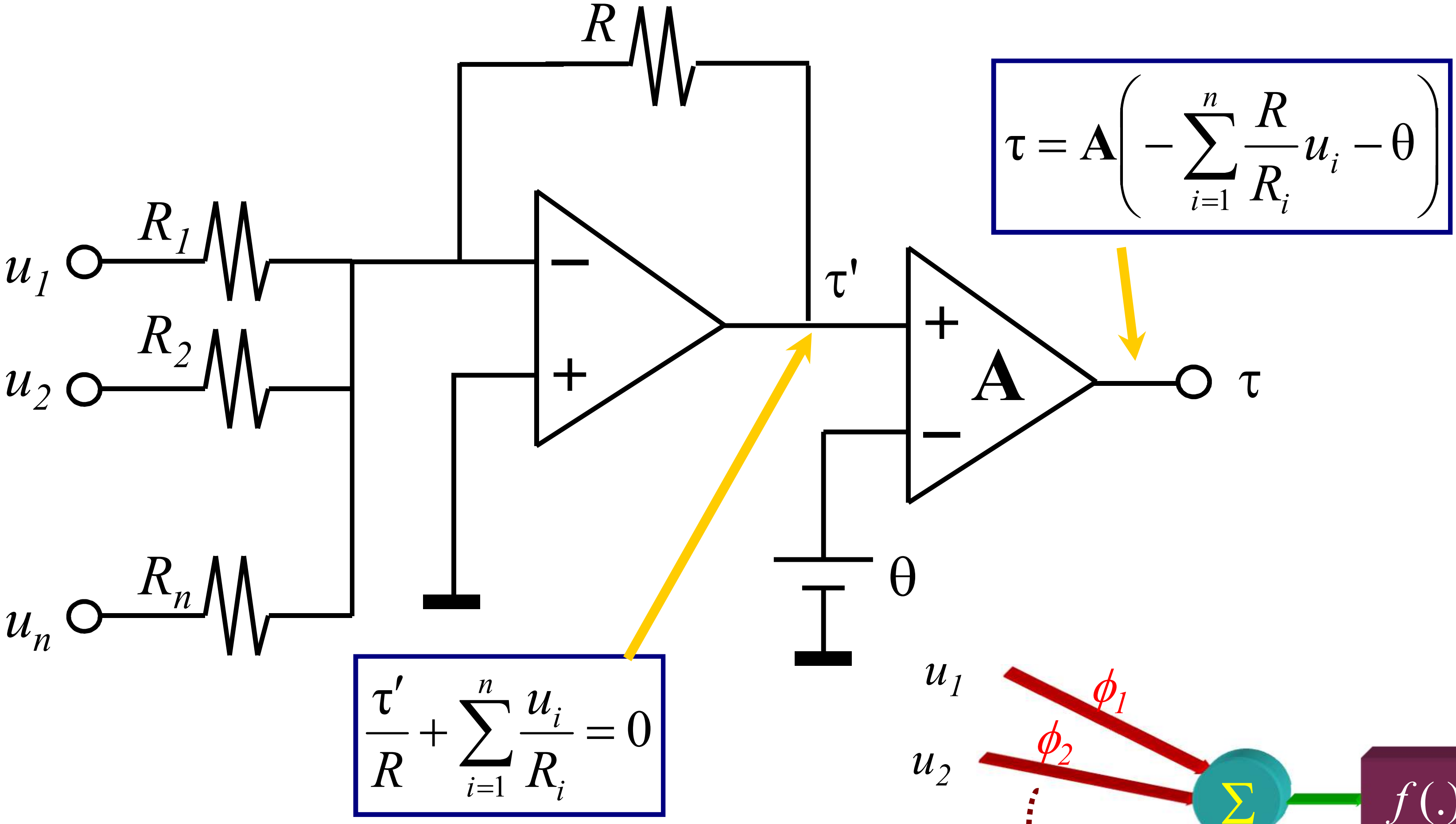
Implementation

<u>Key Actions</u>

- Multiplication
- Summation
- Thresholding

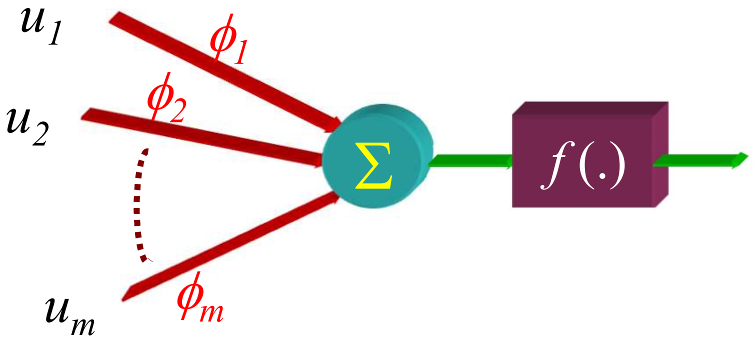
Applications of Neural Networks

VLSI Implementation of Neural Networks

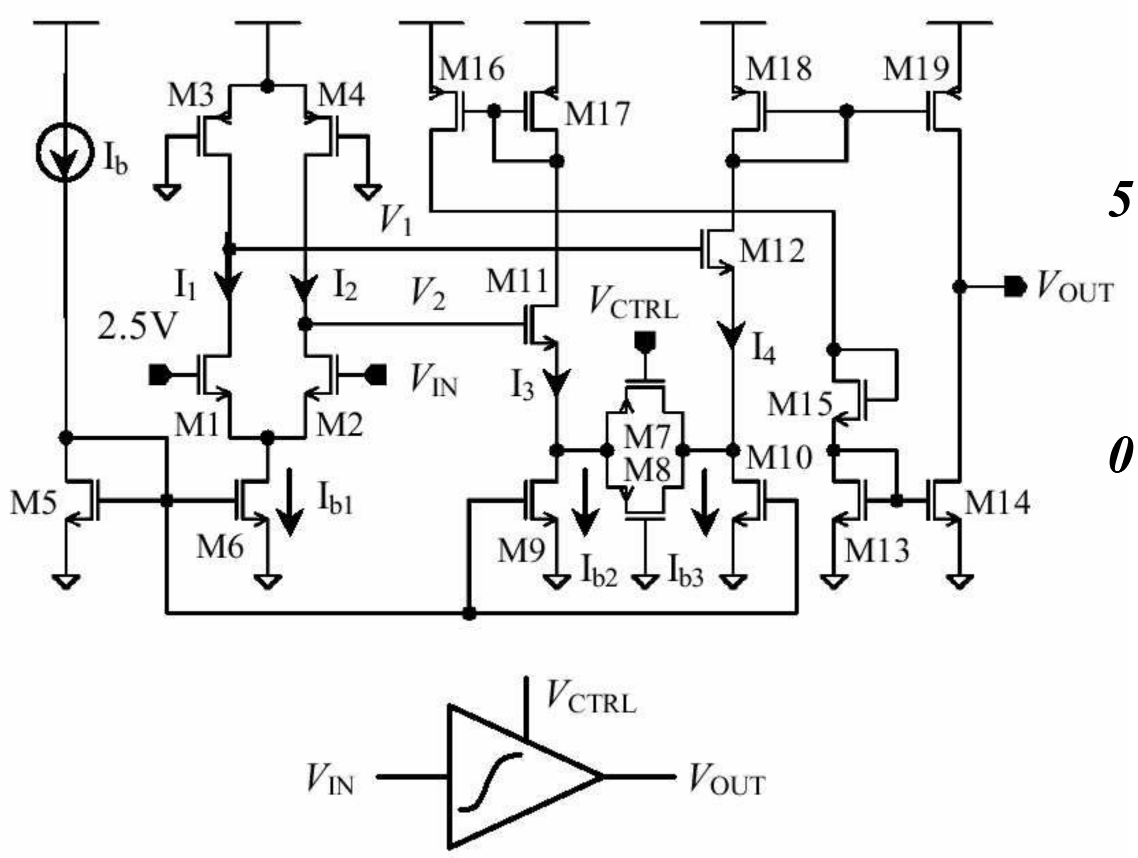


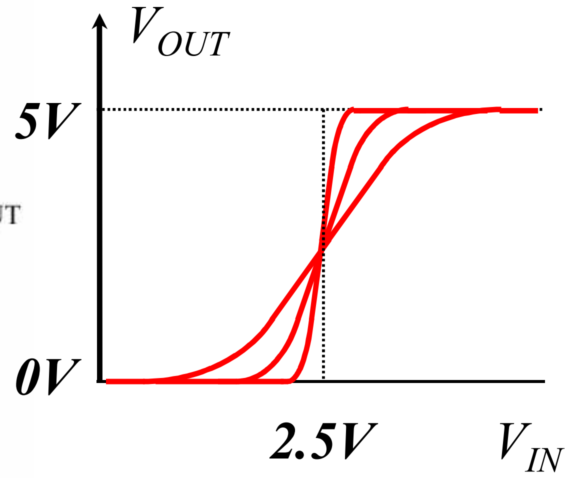
$$\frac{\tau'}{R}+\sum_{i=1}^{n}\frac{u_i}{R_i}=0$$

$$\tau=\mathbf{A}\left(-\sum_{i=1}^{n}\frac{R}{R_i}u_i-\theta\right)$$



Applications of Neural Networks
VLSI Implementation of Neural Networks





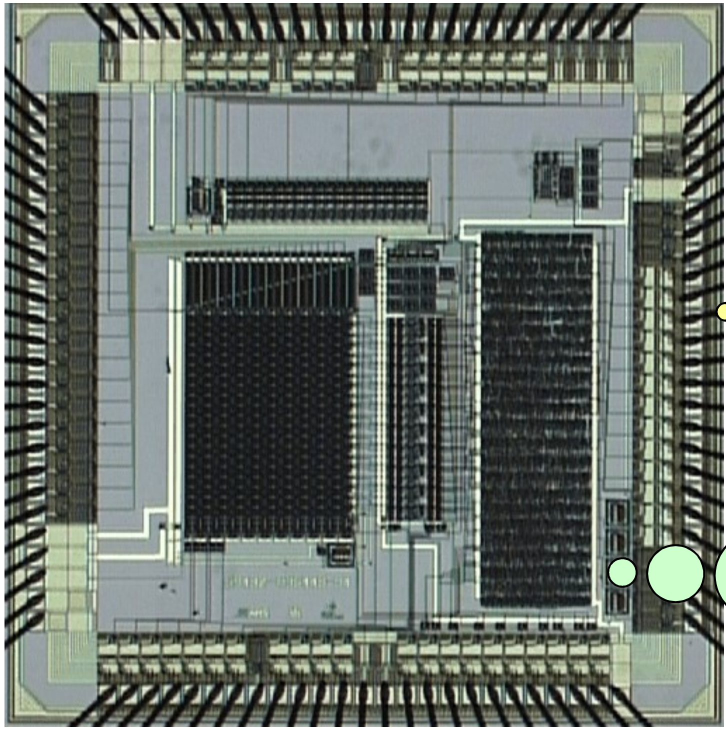
V_{CTRL}=0.8 is sharp

V_{CTRL}=1.6 is smooth

- Figure is taken from: L.Chen and B.Shi, "CMOS PWM Implementation of Neural Network," Proc. of IJCNN-2000.

Applications of Neural Networks

VLSI Implementation of Neural Networks



- Size (Chip area)
- Power consumption
- Operating speed

- Training (on-chip or chip-in-the-loop)
- Part nonidealities
- Operating speed
- Quantization Errors

- Image is taken from: http://vlsi.wpi.edu/P0498/11.html

Applications of Neural Networks

Neural Networks in Medical Diagnosis

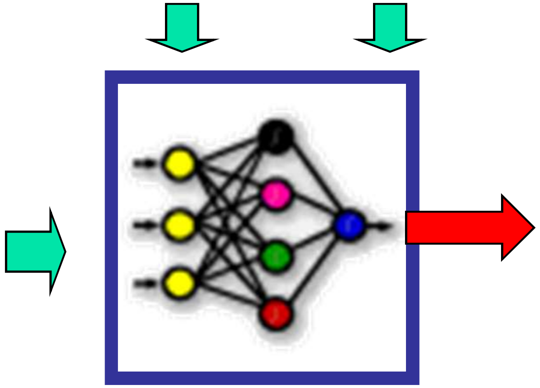
Questionnaire

- Frequent coughing?
- Chest pain?
- Shortness of breath?
- Wheezing?
- Repeated bouts of pneumonia or bronchitis?
- Hoarseness?
- Coughing up of excess mucous?
- Bloody or rust-colored phlegm?

Imaging

http://www.cnn.com/2000/HEALTH/cancer/11/16/lung.cancer/

Laboratory Inspections

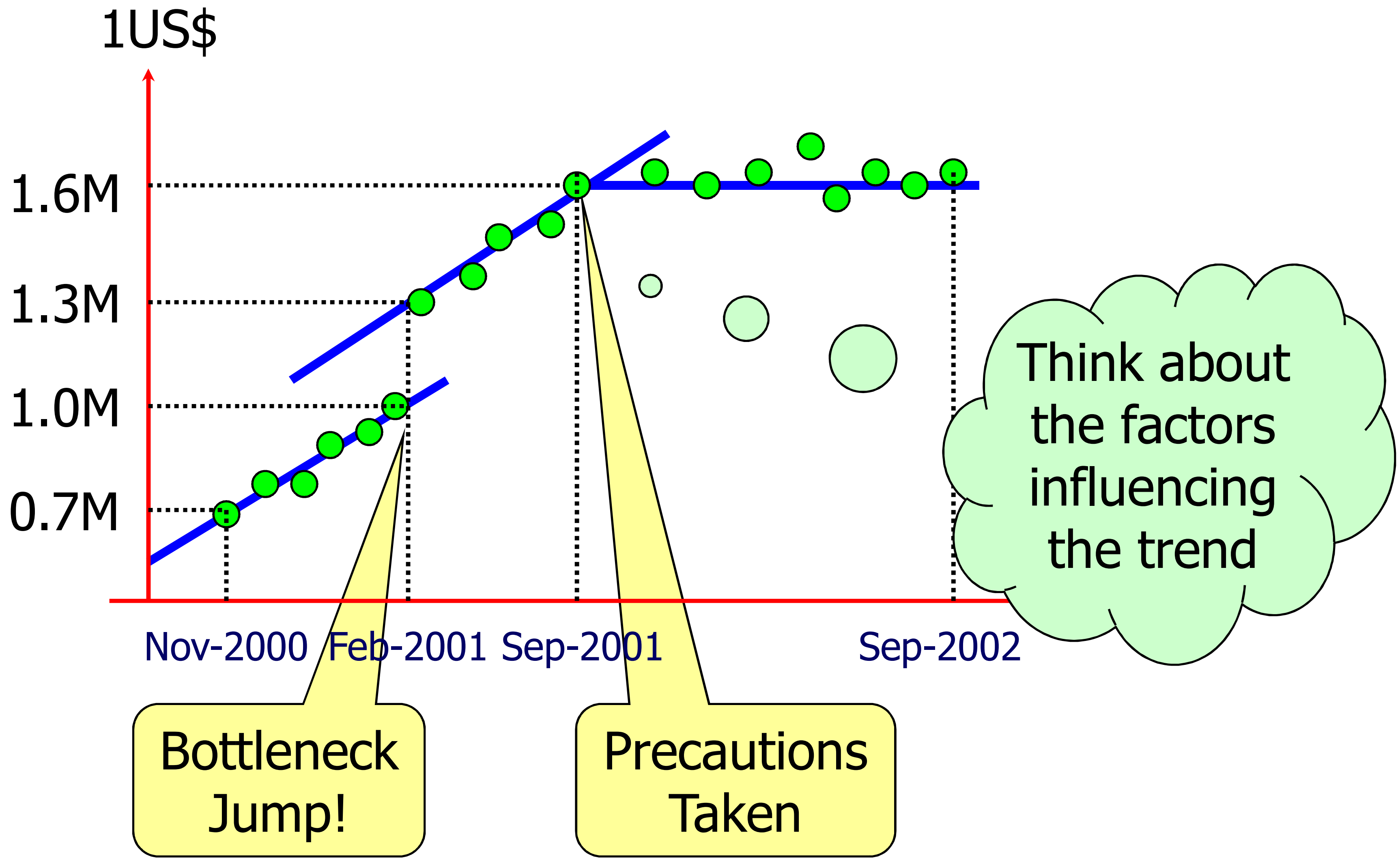


Decision

Train the NN for the data of prior instances and update as new instances occur

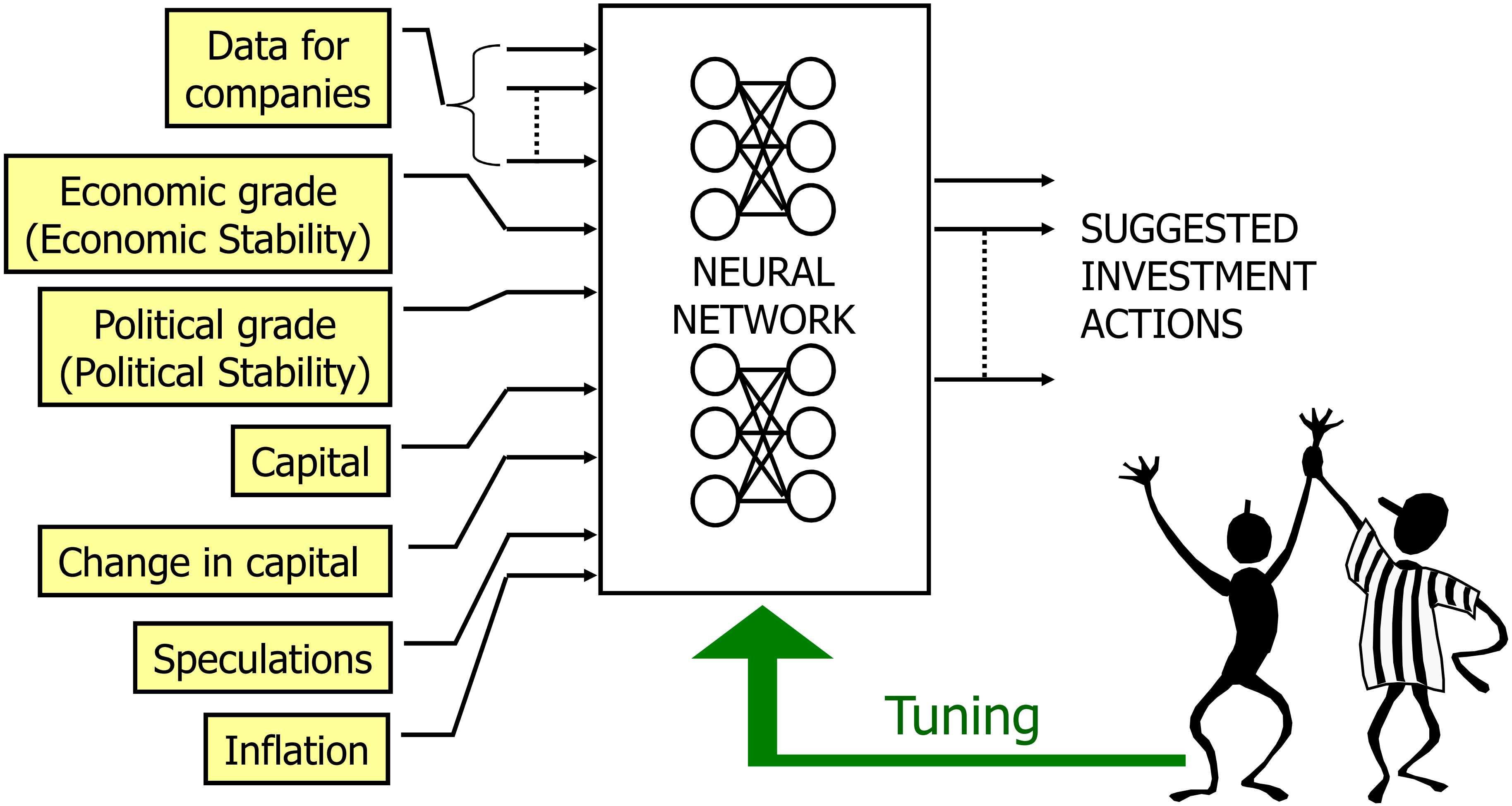
Applications of Neural Networks

Neural Networks for Financial Applications



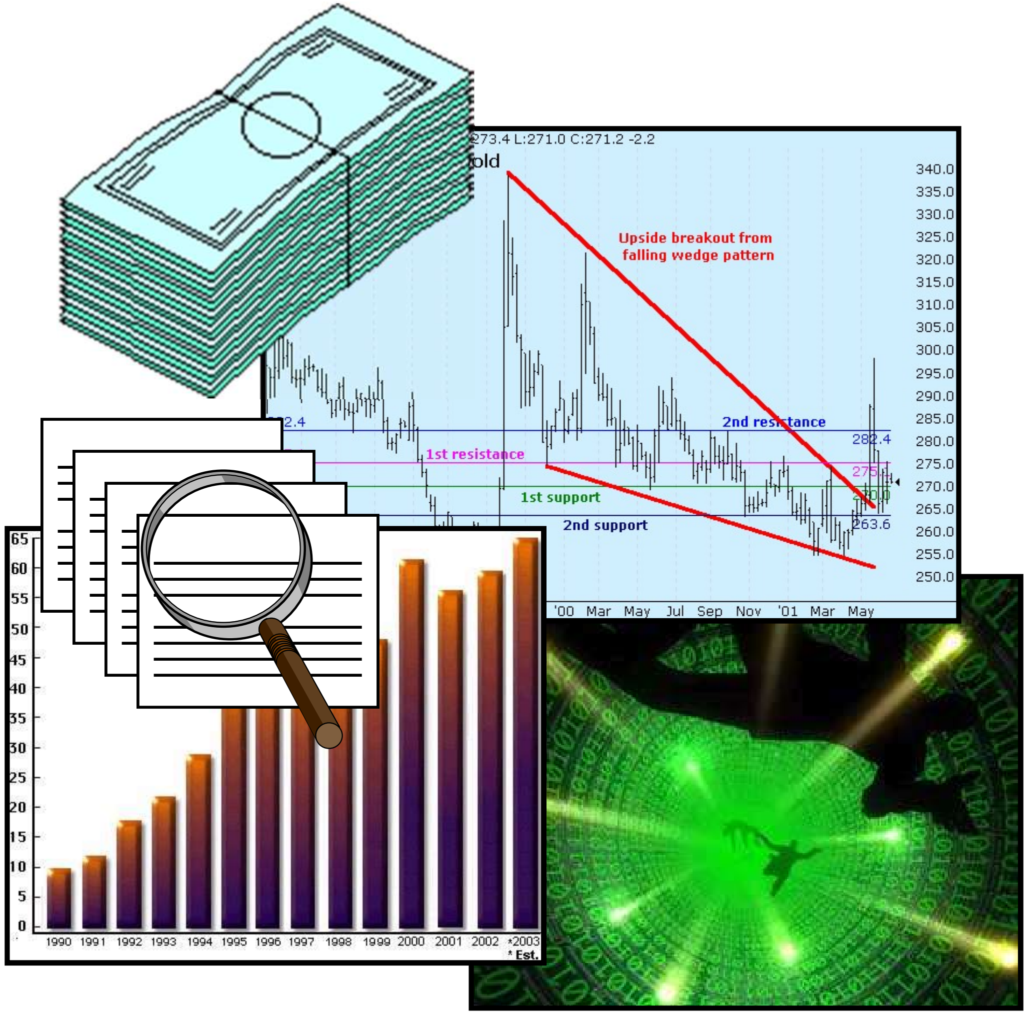
Applications of Neural Networks

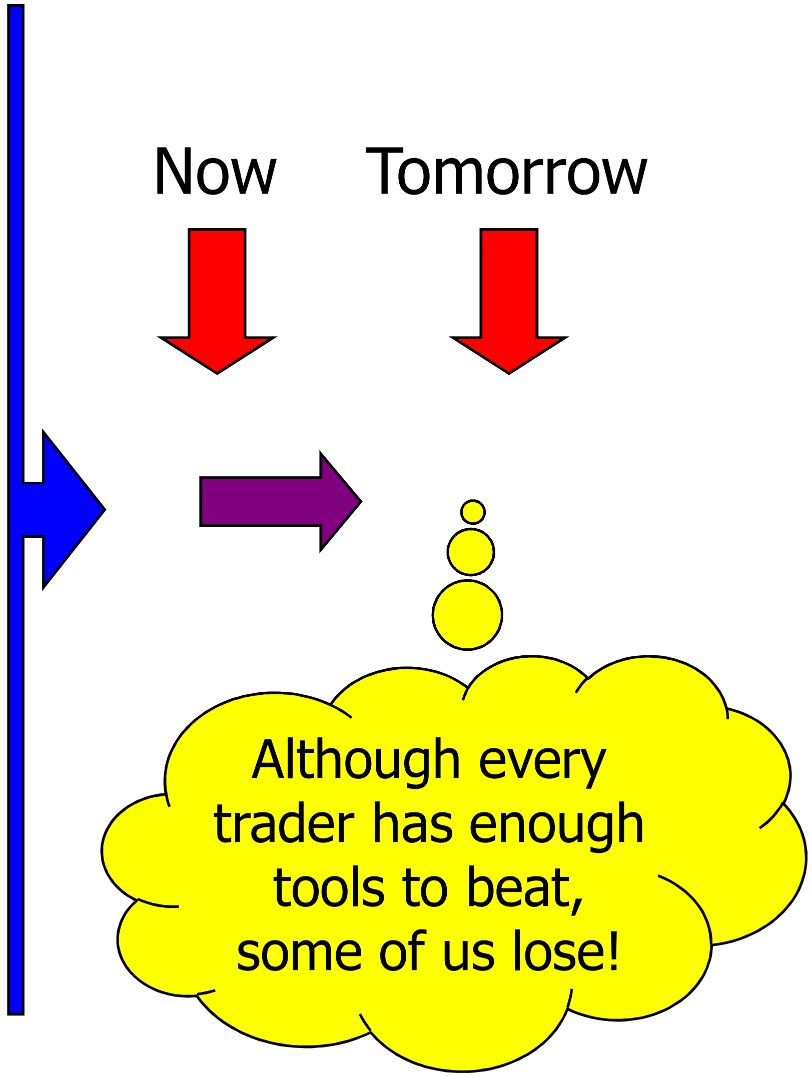
Neural Networks for Financial Applications



Applications of Neural Networks

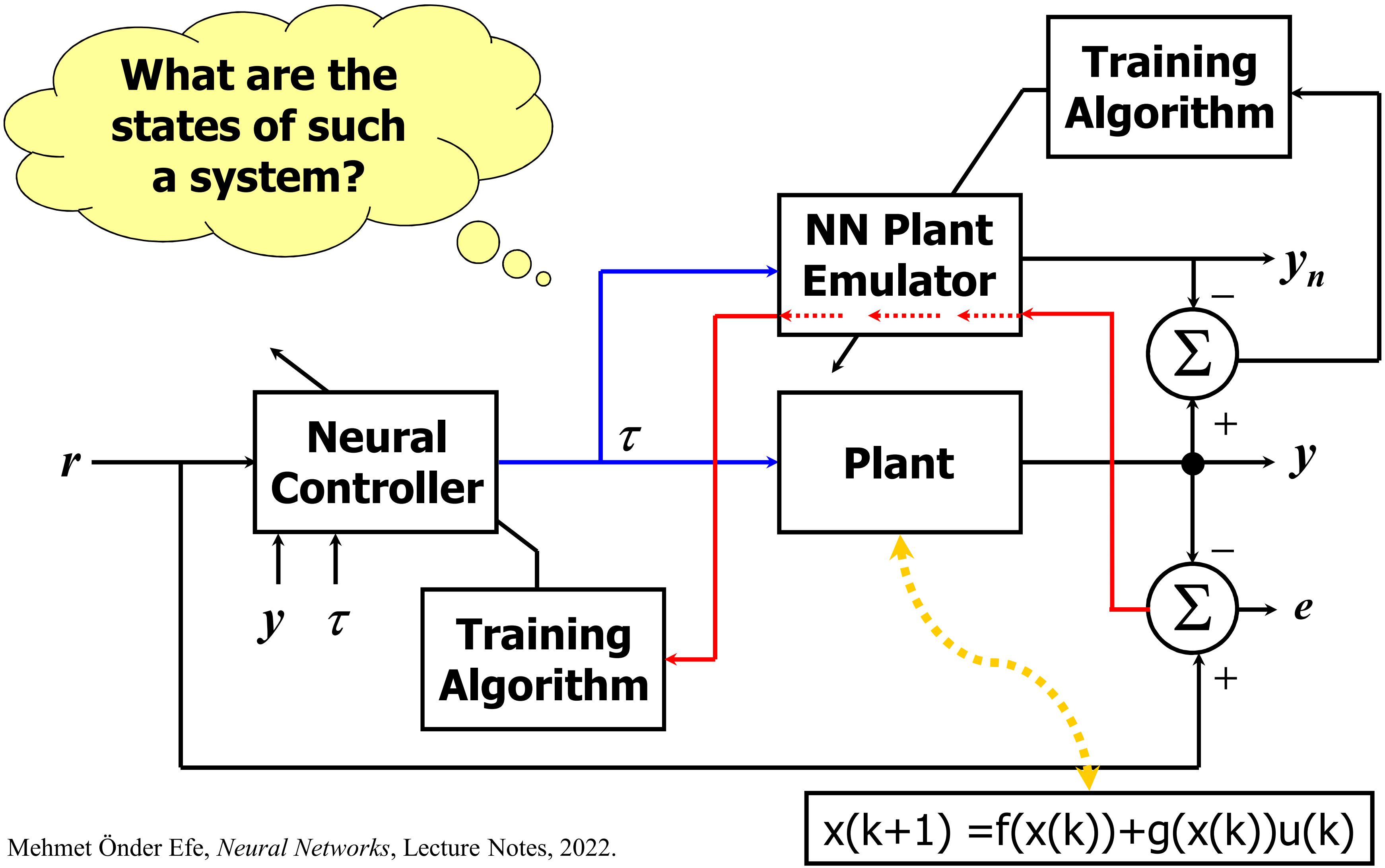
Neural Networks for Financial Applications





- An Open Question - Stability in Learning Systems
- Reinforcement Learning
- Unsupervised Learning

An Open Question
Stability in Learning Systems



An Open Question
Stability in Learning Systems

$$x(k+1) = f\big(x(k)\big) + g\big(x(k)\big)u(k)$$ → System

$$w_{ij}^{\ell}(k+1) = w_{ij}^{\ell}(k) + \eta(y - y_n)\frac{\partial y_n}{\partial w_{ij}^{\ell}}$$ → Emulator weights

$$\omega_{ij}^{\zeta}(k+1) = \omega_{ij}^{\zeta}(k) + \eta(y - y_n)\frac{\partial y_n}{\partial \tau}\frac{\partial \tau}{\partial \omega_{ij}^{\zeta}}$$ → Controller weights

- For a successful application
  Emulator: $y_n \to y$ Closed Loop: $y \to r$ and

$$\sum_{\forall \ell}\sum_{\forall i}\sum_{\forall j}\left(w_{ij}^{\ell}(k)\right)^2 + \sum_{\forall \zeta}\sum_{\forall i}\sum_{\forall j}\left(\omega_{ij}^{\zeta}(k)\right)^2 \to \text{A constant}$$

MLP and EBP

A remedy is regularization technique

L_1 (Lasso) Regularization

$$J = \underbrace{\frac{1}{2}\sum_{i=1}^{n_{k+1}}\left(d_i - o_i^{k+1}(u,w)\right)^2}_{\text{Loss function}} + \underbrace{\lambda\sum_{\forall w_{ij}}\left|w_{ij}\right|}_{\text{Regularization term}}$$

L_2 (Ridge) Regularization

$$J = \underbrace{\frac{1}{2}\sum_{i=1}^{n_{k+1}}\left(d_i - o_i^{k+1}(u,w)\right)^2}_{\text{Loss function}} + \underbrace{\lambda\sum_{\forall w_{ij}} w_{ij}^2}_{\text{Regularization term}}$$

- This prevents unnecessarily large values for few weights

MLP and EBP

Another remedy is Lyapunov approach

A General Backpropagation Algorithm for Feedforward Neural Networks Learning

Xinghuo Yu, M. Onder Efe, and Okyay Kaynak

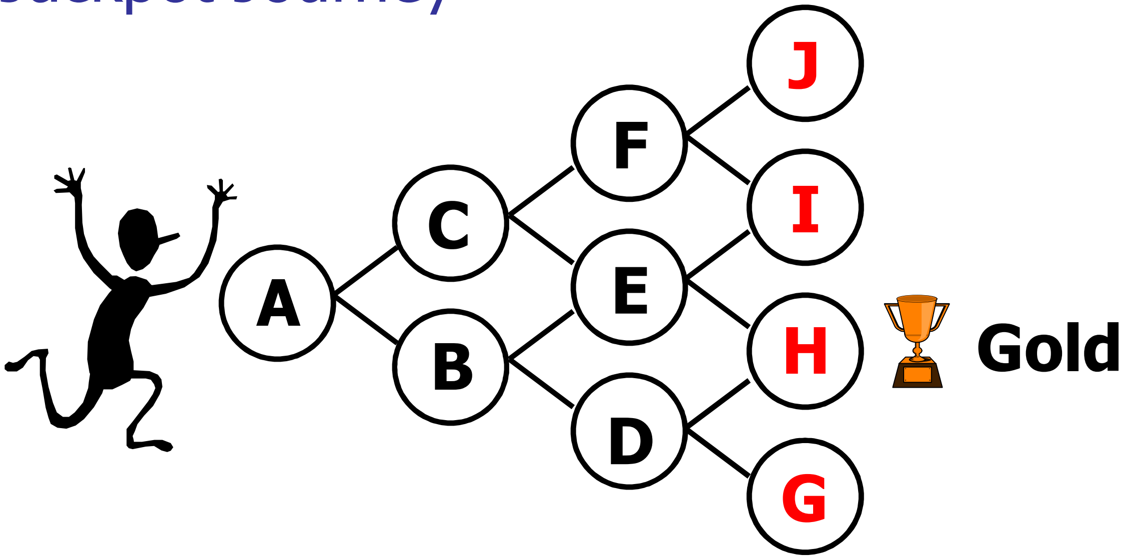
***Abstract*—In this letter, a general backpropagation algorithm is proposed for feedforward neural networks learning with time varying inputs. The Lyapunov function approach is used to rigorously analyze the convergence of weights, with the use of the algorithm, toward minima of the error function. Sufficient conditions to guarantee the convergence of weights for time varying inputs are derived. It is shown that most commonly used backpropagation learning algorithms are special cases of the developed general algorithm.**

***Index Terms*—Backpropagation, feedforward neural networks, stability, training.**

X. Yu, **M.Ö. Efe** and O. Kaynak, "A General Backpropagation Algorithm for Feedforward Neural Networks Learning," *IEEE Transactions on Neural Networks*, v.13, no.1, pp. 251-254, January 2002.

Reinforcement Learning

Jackpot Journey



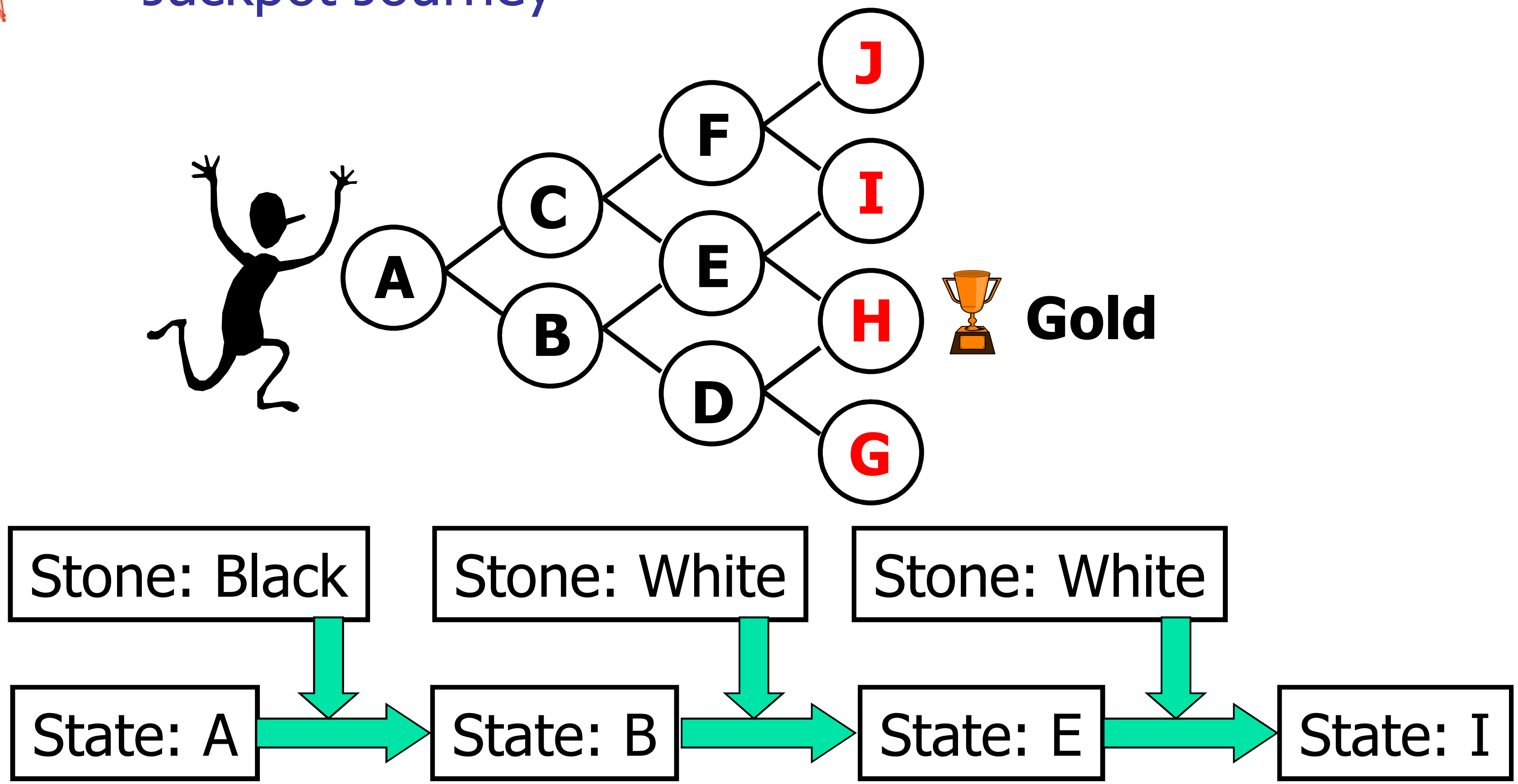
- Find the gold by developing a search policy
- Apply a reward-penalty scheme
- Each signpost has black and white stones
- Pick a stone, if it is BLACK then GO DOWN
  if it is WHITE then GO UP
- Failure is the surest path to success...

$$p_{down} = \frac{\#Black}{\#Black + \#White}$$

$$p_{up} = \frac{\#White}{\#Black + \#White}$$

Reinforcement Learning

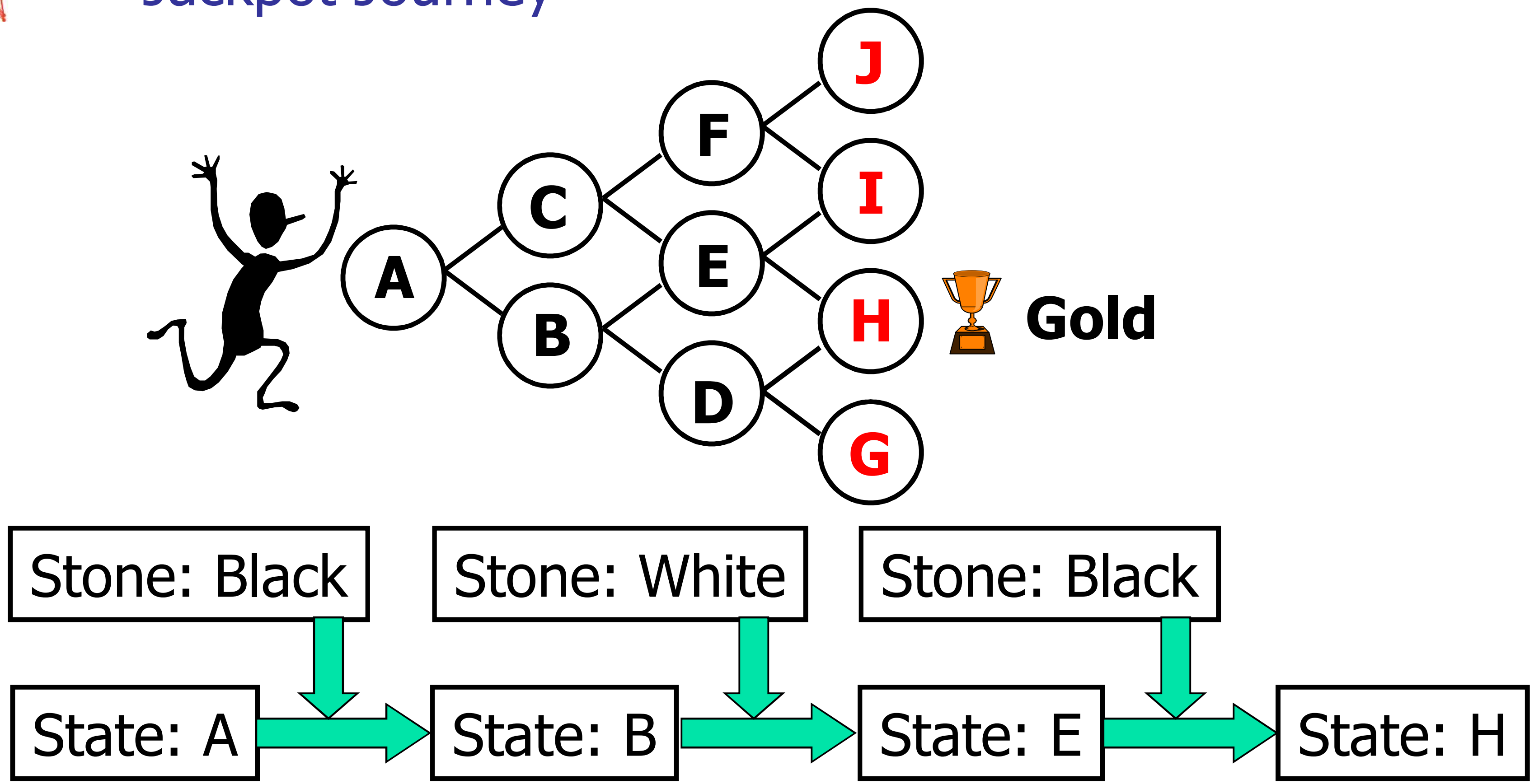
Jackpot Journey



- Failure! Apply the penalty scheme. Take away the stones that make you fail. Now think about the probabilities...

Reinforcement Learning

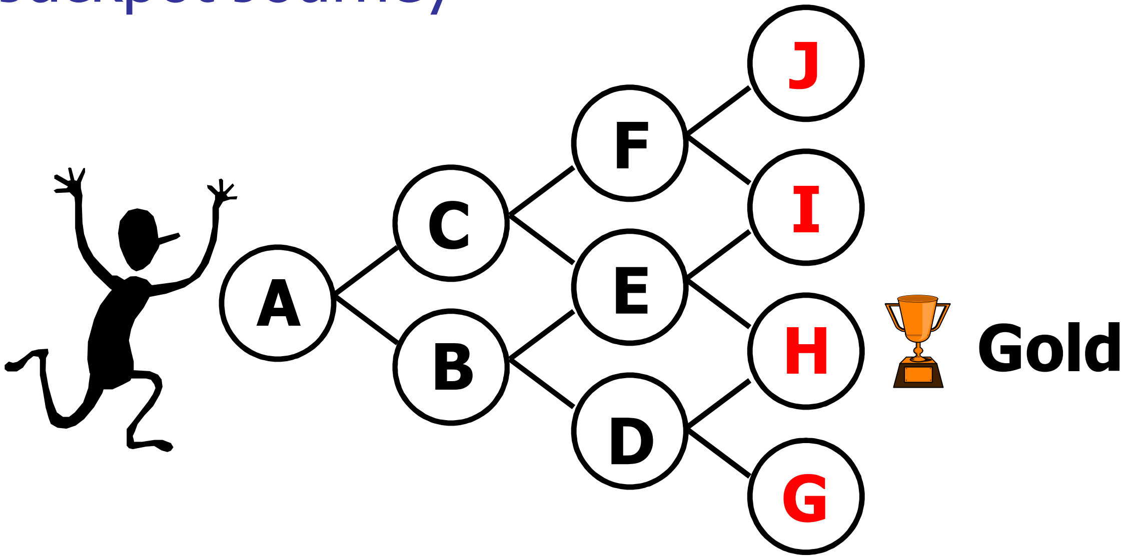
Jackpot Journey



- Success! Apply the reward scheme. Put the stones back into the signposts and put additional one with the same color. Now think about the probabilities...

Reinforcement Learning

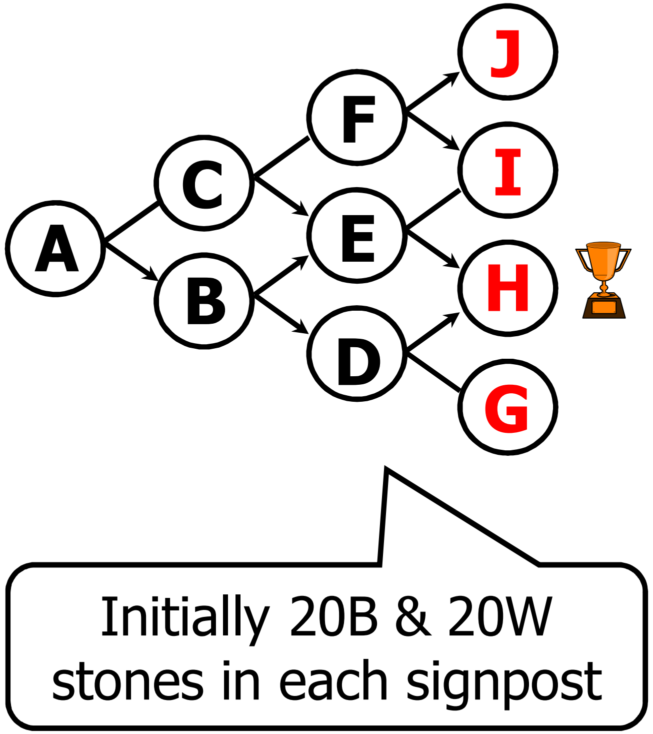
Jackpot Journey

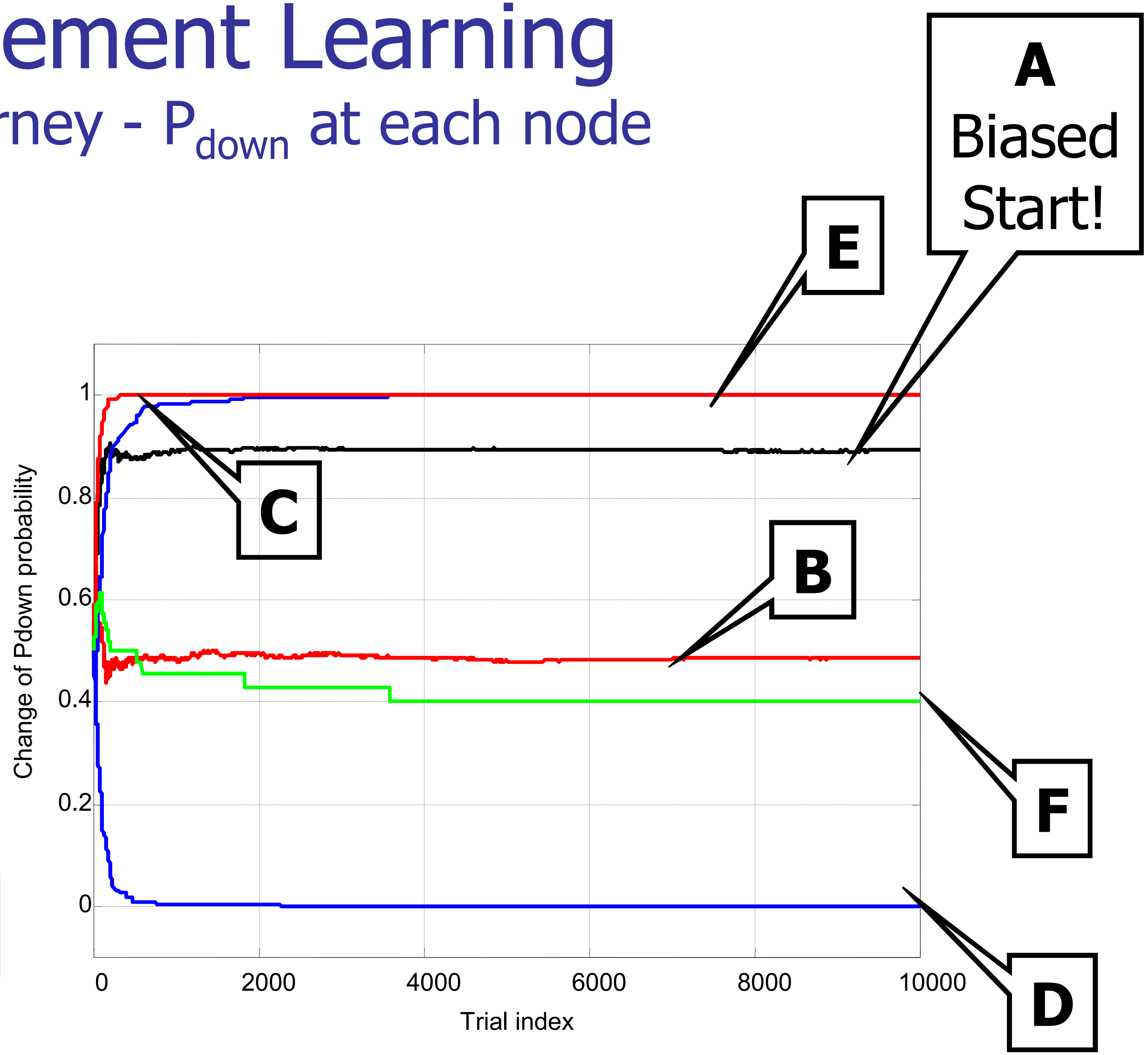


- Perform many voyages to reinforce…
- As you fail, the probability of the action that makes you fail is reduced by the penalty scheme
- As you succeed, the probability of the action that makes you succeed is strenghtened

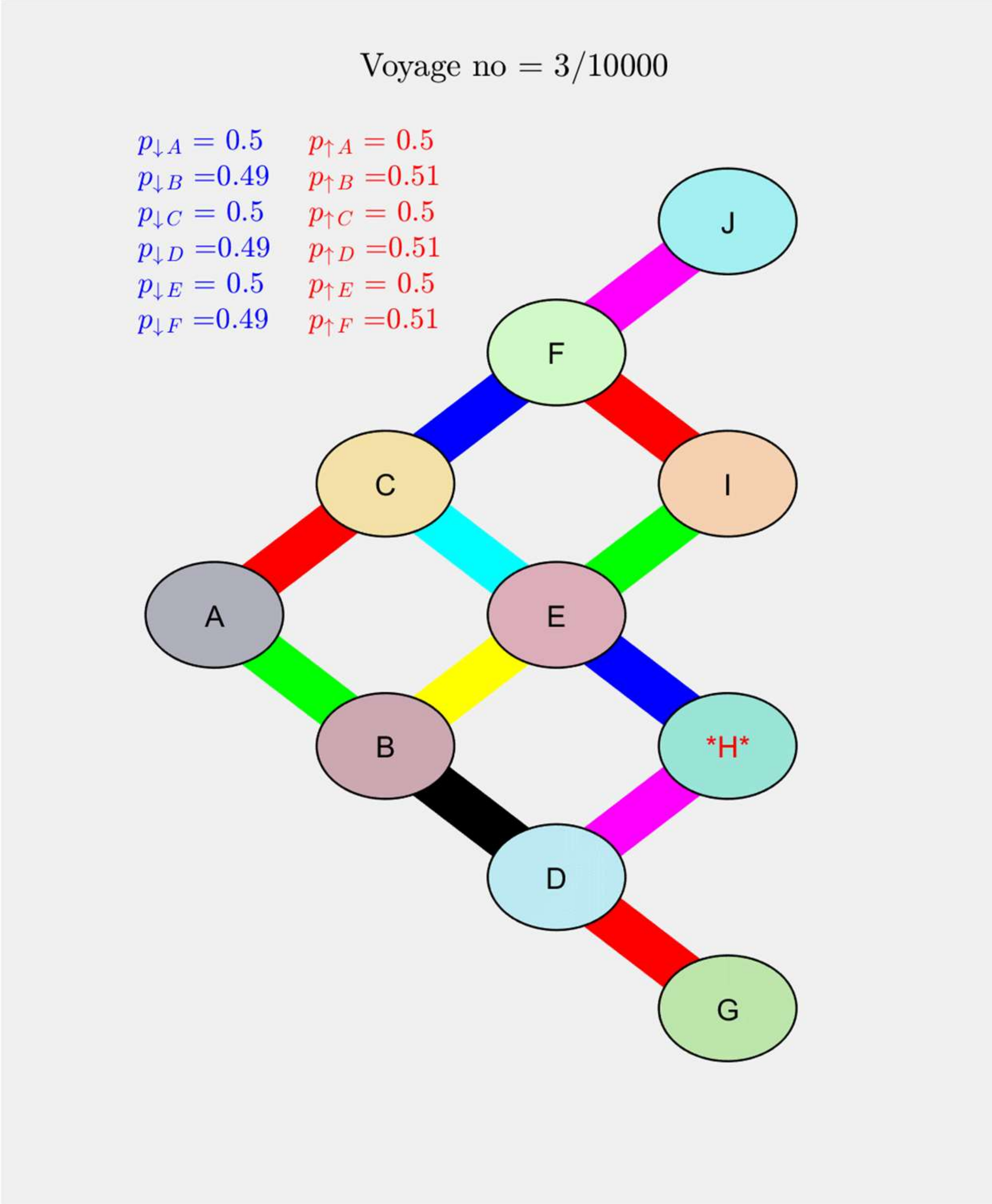
Reinforcement Learning

Jackpot Journey - P_{down} at each node





- Pay attention to B and F. If you are at B, you find the gold no matter which way you choose. For F, you cannot...

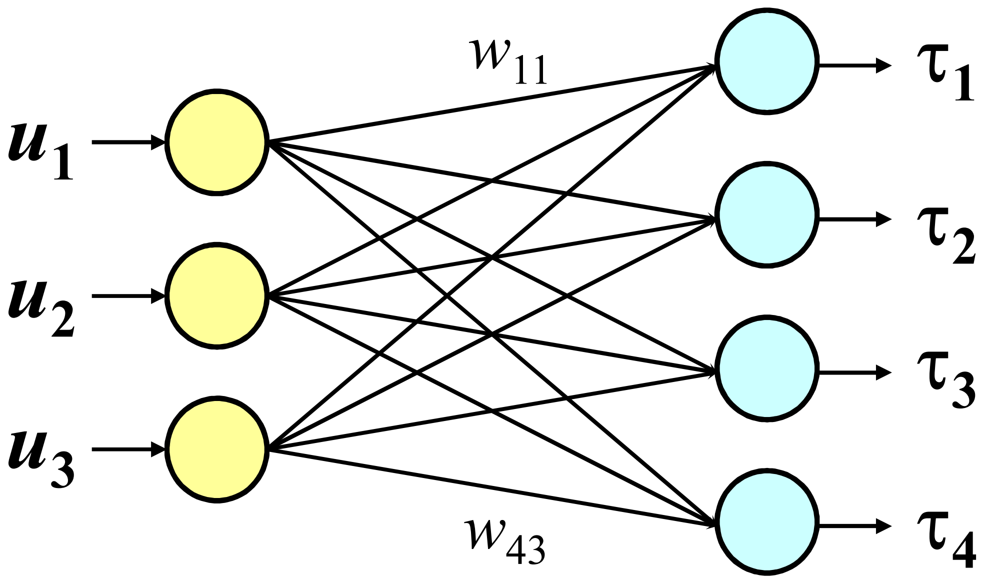


Voyage no = 3/10000
$p_{\downarrow A} = 0.5$
$p_{\downarrow B} = 0.49$
$p_{\downarrow C} = 0.5$
$p_{\downarrow D} = 0.49$
$p_{\downarrow E} = 0.5$
$p_{\downarrow F} = 0.49$
$p_{\uparrow A} = 0.5$
$p_{\uparrow B} = 0.51$
$p_{\uparrow C} = 0.5$
$p_{\uparrow D} = 0.51$
$p_{\uparrow E} = 0.5$
$p_{\uparrow F} = 0.51$
J
F
C
I
A
E
B
H
D
G

Unsupervised Learning
General Remarks

- No external teacher (supervisory information) available
- Only the input vectors will be used for learning
- Unsupervised learning system $\Leftrightarrow$ Agent
- The Agent extracts the regularities, associations in the data
- The data contains several persistent features available redundantly
- Unsupervised learning is used for Data Clustering, Feature Extraction and Similarity Detection
- Dissimilar input patterns excite different internal parts of a network. This leads to the development of specialized internal structures in the neural network

Unsupervised Learning
Competitive Learning (Winner-take-all Learning)

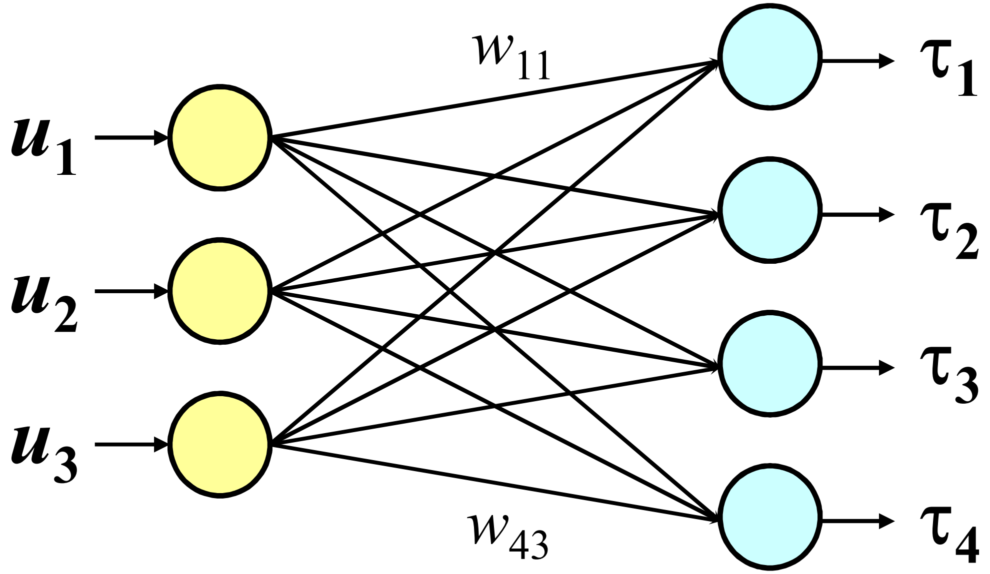


$$S_i = \sum_{j=1}^{3} w_{ij} u_j$$

Calculate this inner product (activation level) for all output neurons and choose the neuron having maximum activation value. Say that one is k-th neuron

Unsupervised Learning

Competitive Learning (Winner-take-all Learning)



$$S_i = \sum_{j=1}^{3} w_{ij} u_j$$

$$\tau_k = \operatorname{sgn}(S_k)$$

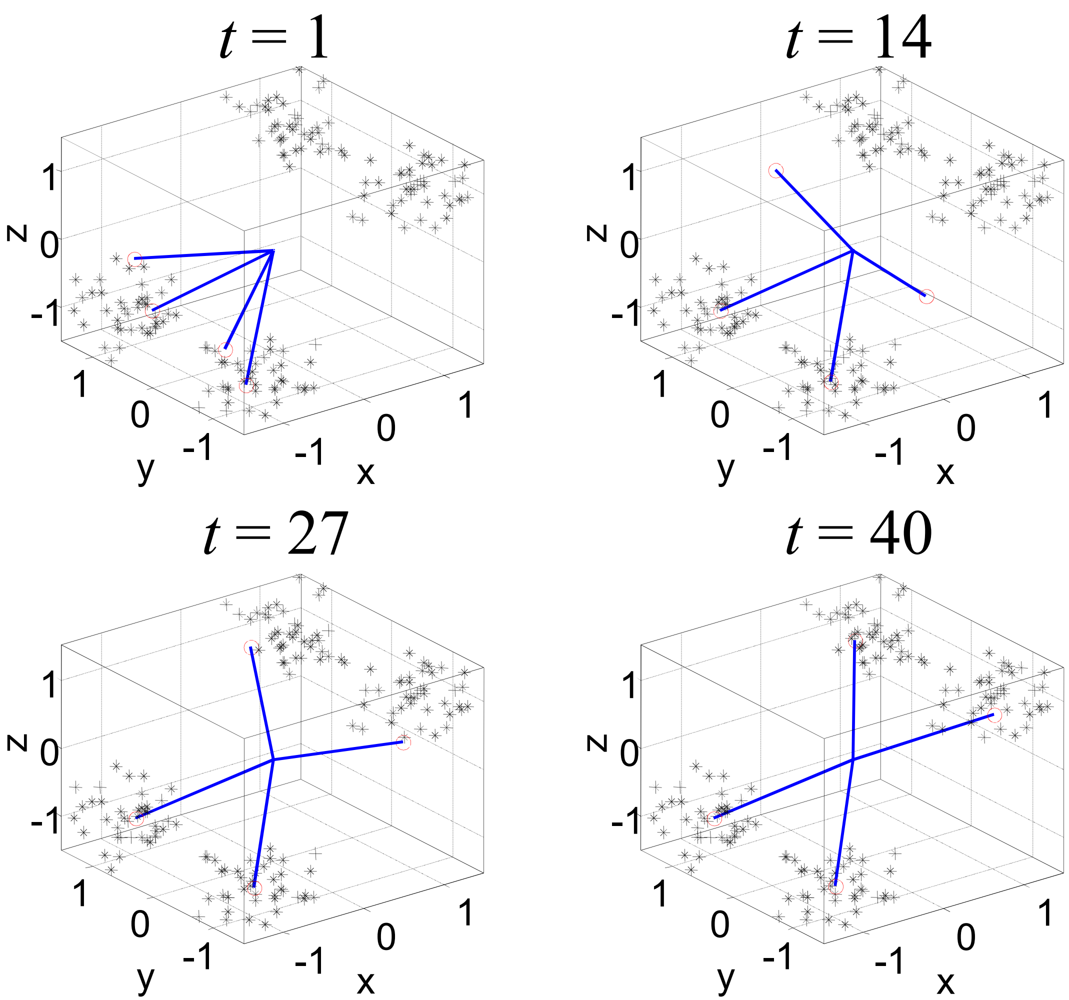
$$S_k = \underline{w}_k^T \underline{u}$$

$$\underline{w}_k(t+1) = \frac{\underline{w}_k(t) + \eta\left(\underline{u}(t) - \underline{w}_k(t)\right)}{\left\| \underline{w}_k(t) + \eta\left(\underline{u}(t) - \underline{w}_k(t)\right) \right\|}$$

- Note that only the weights of the winner are updated

Unsupervised Learning

Competitive Learning (Winner-take-all Learning)



Unsupervised Learning

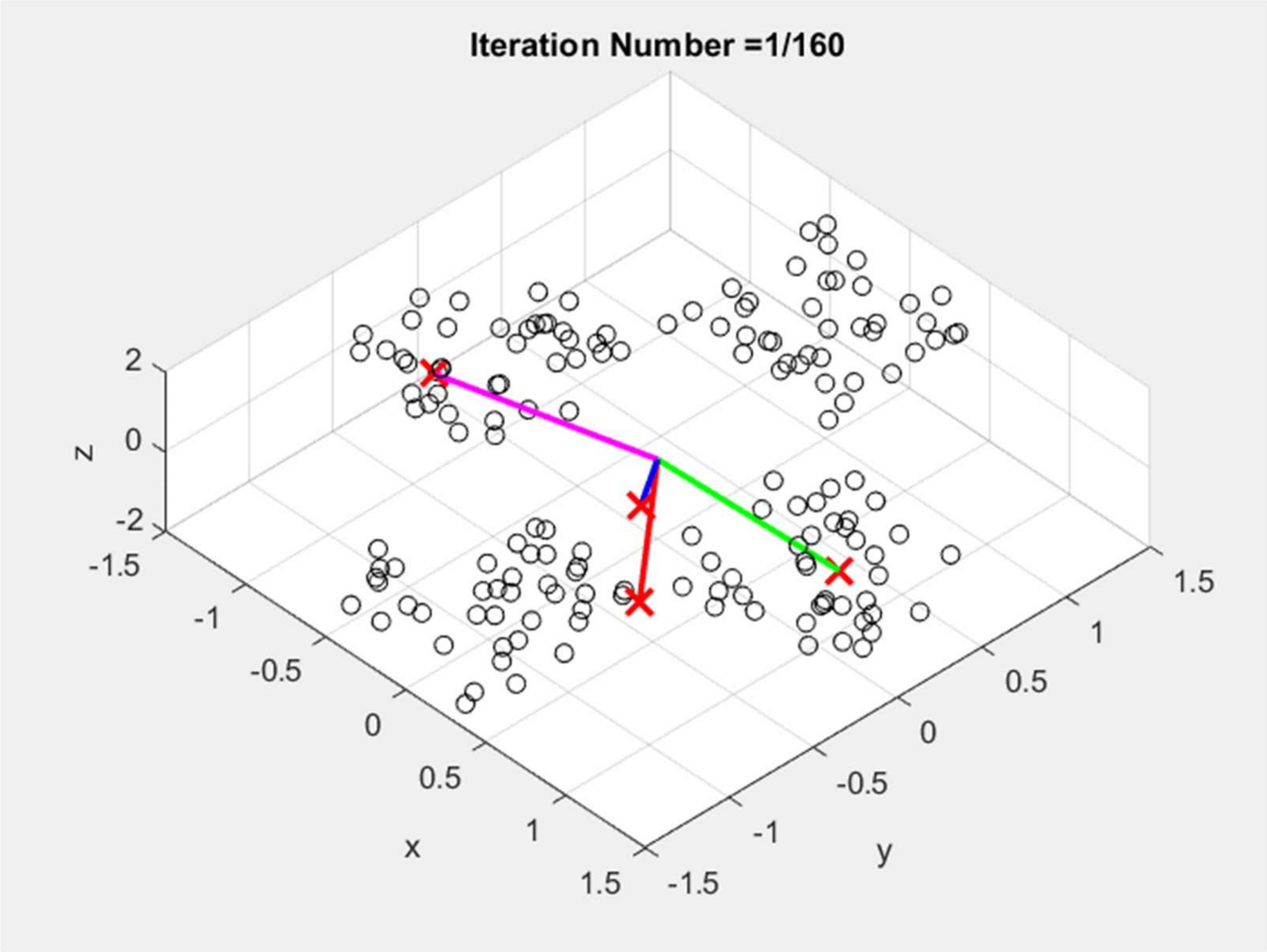
Competitive Learning (Winner-take-all Learning)

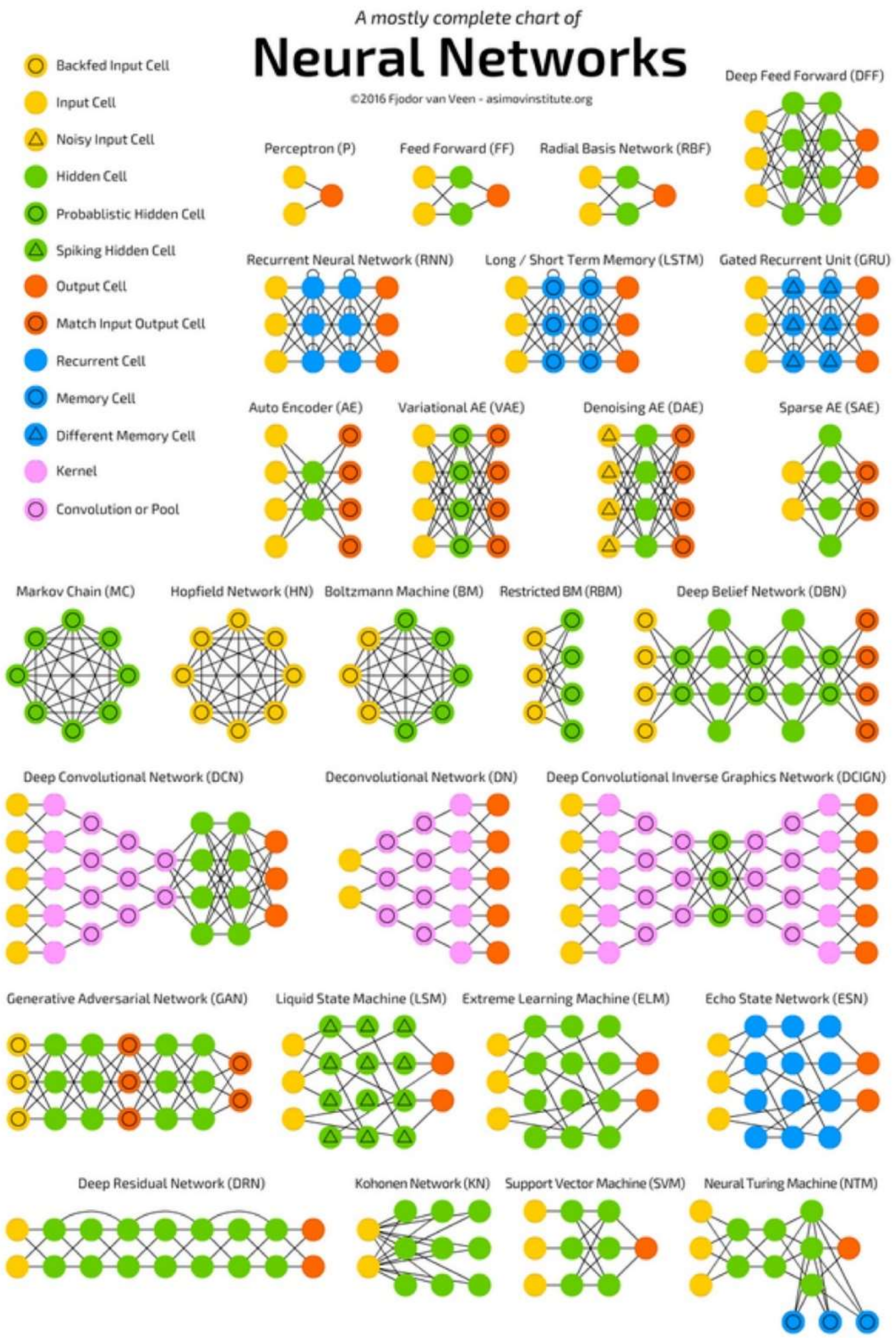
- Four clusters available in the data, and we have chosen 4 output neurons to find those clusters.
- Data might have more than 4 clusters, then the final vectors would converge **at most** to 4 of them. For example, data has 6 clusters, you have 4 neurons and you find out 3 clusters!
- We have initialized the weights to randomly chosen input patterns. This is because of the following: After random initialization, some weights can be far away from the data and those weights never get updated! The procedure overcomes this drawback.

- Watch the movie...

Unsupervised Learning

Competitive Learning (Winner-take-all Learning)




A mostly complete chart of
Neural Networks
©2016 Fjodor van Veen - asimovinstitute.org
Backfed Input Cell
Input Cell
Noisy Input Cell
Hidden Cell
Probablistic Hidden Cell
Spiking Hidden Cell
Output Cell
Match Input Output Cell
Recurrent Cell
Memory Cell
Different Memory Cell
Kernel
Convolution or Pool
Perceptron (P)
Feed Forward (FF)
Radial Basis Network (RBF)
Deep Feed Forward (DFF)
Recurrent Neural Network (RNN)
Long / Short Term Memory (LSTM)
Gated Recurrent Unit (GRU)
Auto Encoder (AE)
Variational AE (VAE)
Denoising AE (DAE)
Sparse AE (SAE)
Markov Chain (MC)
Hopfield Network (HN)
Boltzmann Machine (BM)
Restricted BM (RBM)
Deep Belief Network (DBN)
Deep Convolutional Network (DCN)
Deconvolutional Network (DN)
Deep Convolutional Inverse Graphics Network (DCIGN)
Generative Adversarial Network (GAN)
Liquid State Machine (LSM)
Extreme Learning Machine (ELM)
Echo State Network (ESN)
Deep Residual Network (DRN)
Kohonen Network (KN)
Support Vector Machine (SVM)
Neural Turing Machine (NTM)

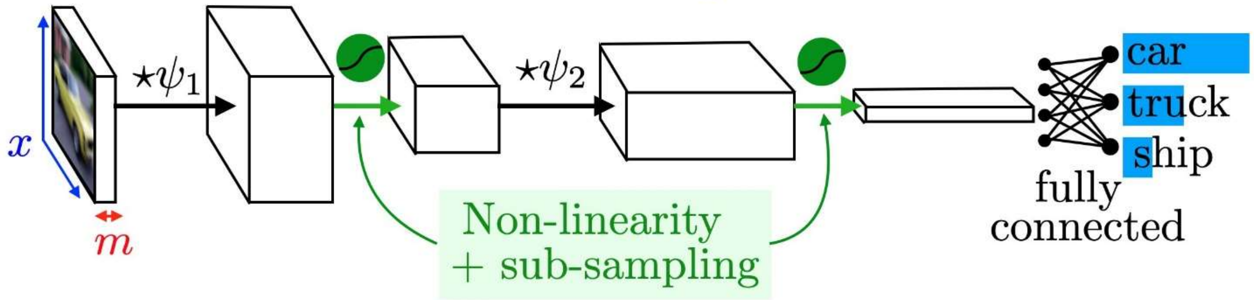
Where to go from here?

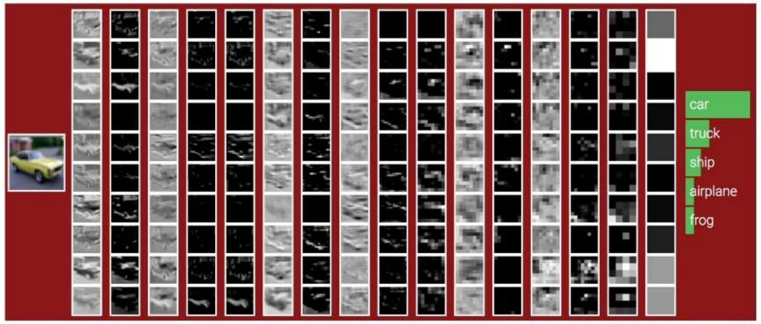
- ❑ Convolutional Neural Networks
- ❑ Transfer Learning
- ❑ Graph Neural Networks
- ❑ Generative Adversarial Networks
- ❑ Transformers

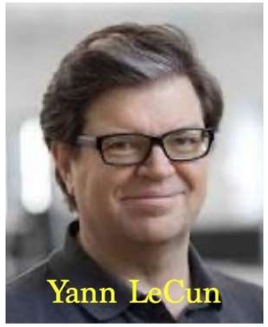
Convolutional Neural Networks

Multi-canal convolution: $(f \star \psi)_m(x) \overset{\text{def.}}{=} \sum_{\ell} \sum_{y+z=x} f_\ell(y)\psi_{m,\ell}(z)$

canal position







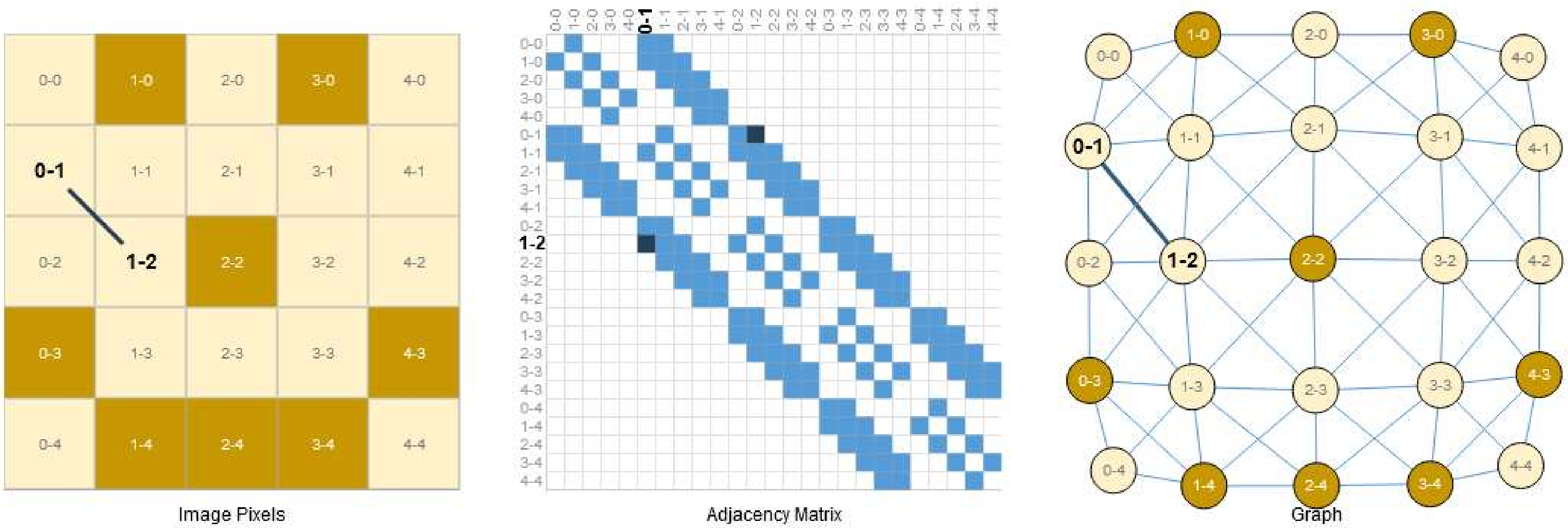
http://vision.stanford.edu/teaching/cs231n/

@gabrielpeyre: Convolutional neural networks are shift invariant representations obtained by iterating convolutions and pointwise non-linearities. Championed by LeCun in the 80s and used everywhere for computer vision nowadays.

Transfer Learning

Graph Neural Networks

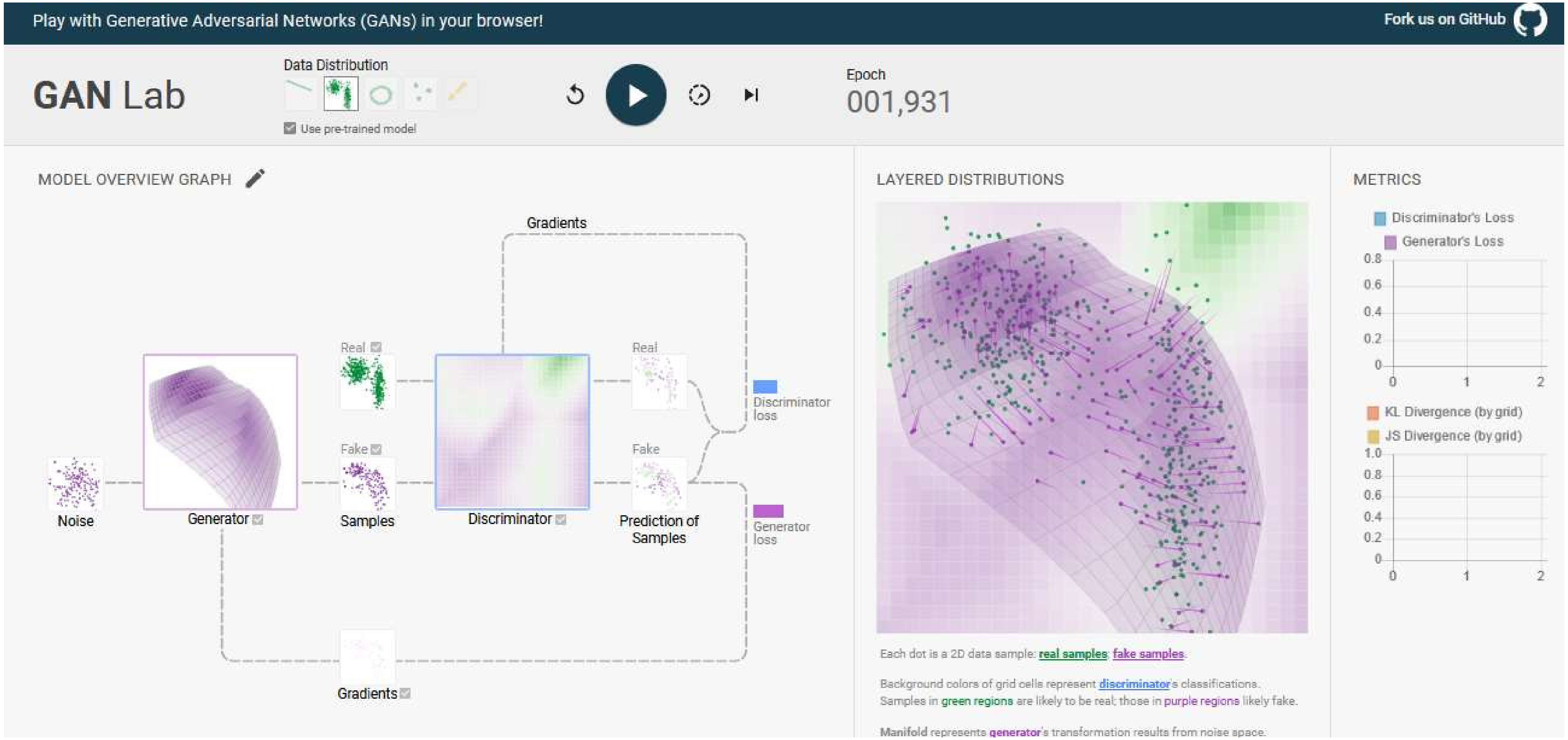
See https://distill.pub/2021/gnn-intro/



Click on an image pixel to toggle its value, and see how the graph representation changes.

Generative Adversarial Networks

See https://poloclub.github.io/ganlab/



Transformers

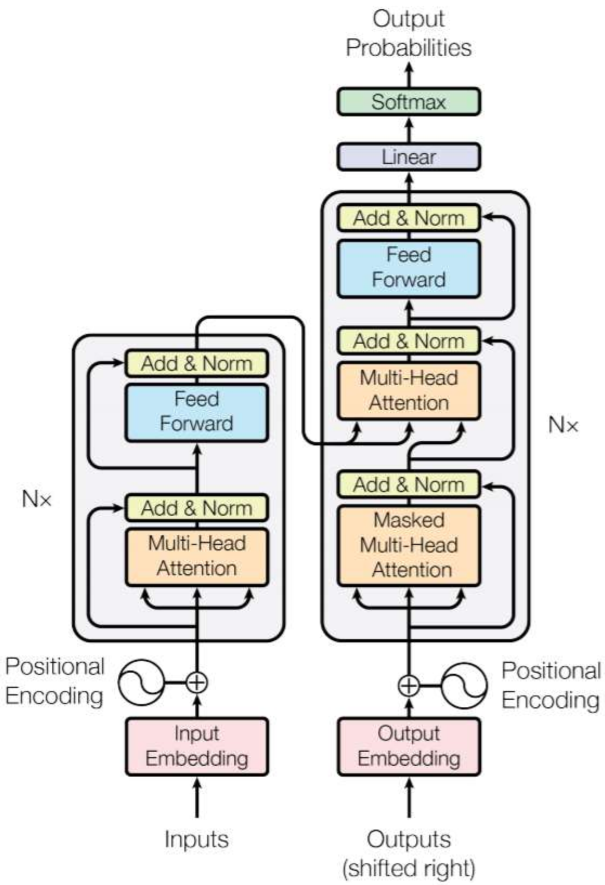


Figure 1: The Transformer - model architecture.