

PMR5406 Redes Neurais e Lógica Fuzzy

Aula 3

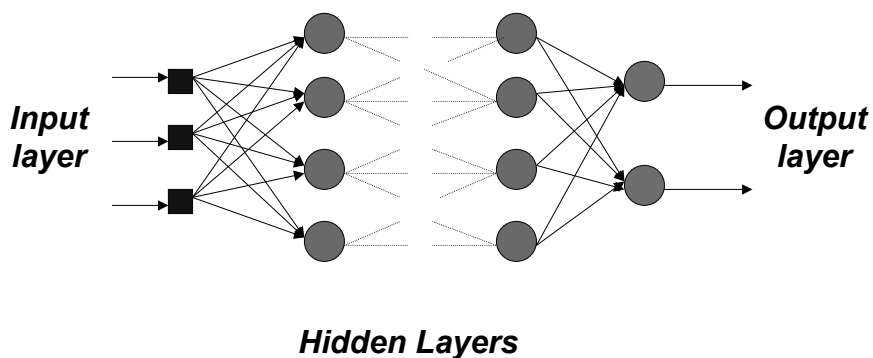
Multilayer Percetrons

Baseado em:

Neural Networks, Simon Haykin, Prentice-Hall, 2nd
edition

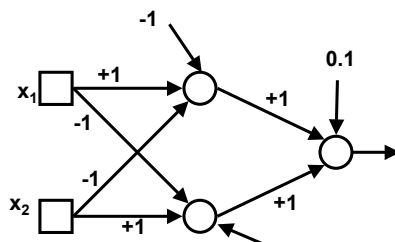
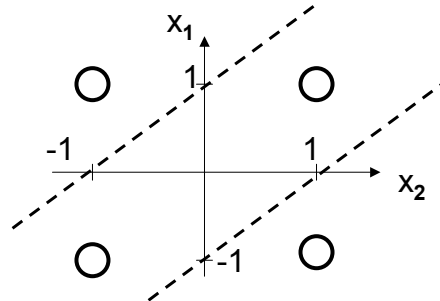
Slides do curso por Elena Marchiori, Vrije
Unviersity

Multilayer Perceptrons Architecture



A solution for the XOR problem

x_1	x_2	$x_1 \text{ XOR } x_2$
-1	-1	-1
-1	1	1
1	-1	1
1	1	-1



$$\varphi(v) = \begin{cases} 1 & \text{if } v > 0 \\ -1 & \text{if } v \leq 0 \end{cases}$$

φ is the sign function.

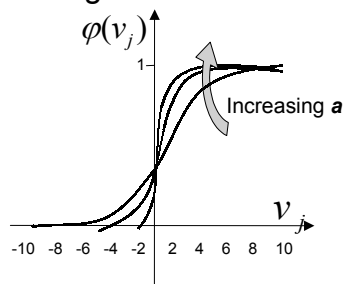
PMR5406 Redes Neurais e
Lógica Fuzzy

Multilayer Perceptrons

3

NEURON MODEL

- Sigmoidal Function



$$\varphi(v_j) = \frac{1}{1 + e^{-av_j}}$$

$$v_j = \sum_{i=0, \dots, m} w_{ji} y_i$$

- v_j induced field of neuron j
- Most common form of activation function
- $a \rightarrow \infty \Rightarrow \varphi \rightarrow$ threshold function
- Differentiable

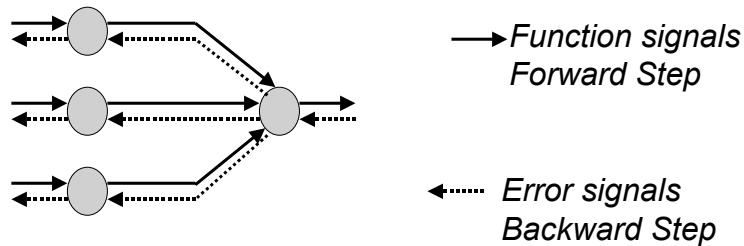
PMR5406 Redes Neurais e
Lógica Fuzzy

Multilayer Perceptrons

4

LEARNING ALGORITHM

- Back-propagation algorithm



- It adjusts the weights of the NN in order to minimize the average squared error.

PMR5406 Redes Neurais e
Lógica Fuzzy

Multilayer Perceptrons

5

Average Squared Error

- Error signal of output neuron j at presentation of n -th training example:

$$e_j(n) = d_j(n) - y_j(n)$$

- Total energy at time n :

$$E(n) = \frac{1}{2} \sum_{j \in C} e_j^2(n)$$

- Measure of learning performance:

$$E_{AV} = \frac{1}{N} \sum_{n=1}^N E(n)$$

C : Set of
neurons
in output
layer

N : size of
training set

- **Goal: Adjust weights of NN to minimize E_{AV}**

PMR5406 Redes Neurais e
Lógica Fuzzy

Multilayer Perceptrons

6

Notation

e_j Error at output of neuron j

y_j Output of neuron j

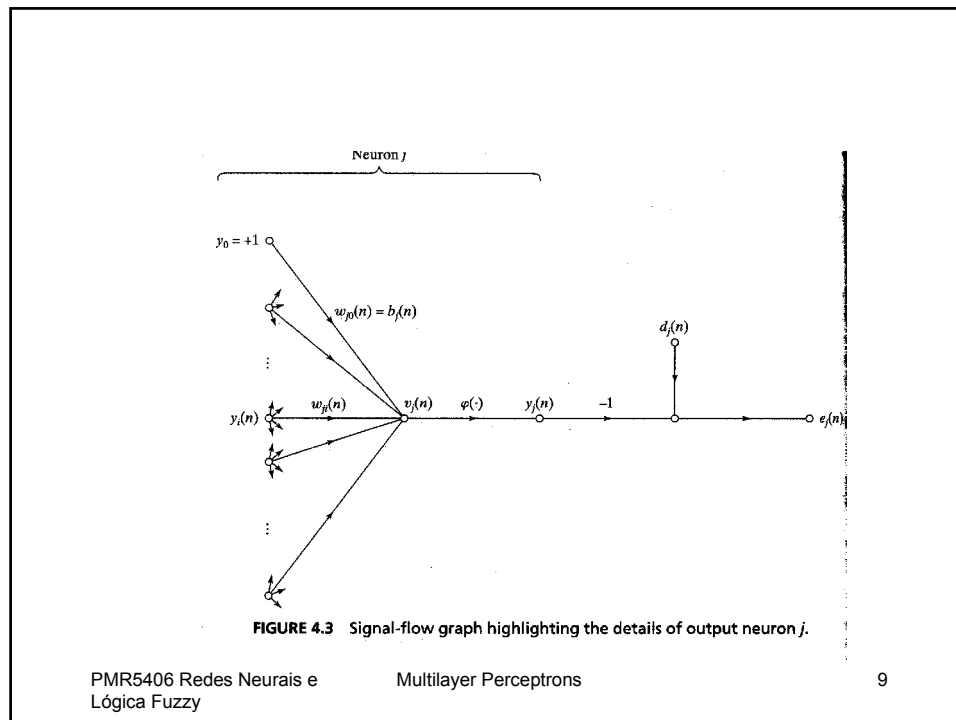
$v_j = \sum_{i=0, \dots, m} w_{ji} y_i$ Induced local field of neuron j

Weight Update Rule

Update rule is based on the gradient descent method
take a step in the direction yielding the maximum
decrease of E

$$\Delta w_{ji} = -\eta \frac{\partial E}{\partial w_{ji}} \quad \text{Step in direction opposite to the gradient}$$

With w_{ji} weight associated to the link from neuron i
to neuron j



Definition of the Local Gradient of neuron j

$$\delta_j = - \frac{\partial E}{\partial v_j} \quad \text{Local Gradient}$$

We obtain

$$\delta_j = e_j \phi'(v_j)$$

because

$$- \frac{\partial E}{\partial v_j} = - \frac{\partial E}{\partial e_j} \frac{\partial e_j}{\partial y_j} \frac{\partial y_j}{\partial v_j} = - e_j (-1) \phi'(v_j)$$

Update Rule

- We obtain
- because

$$\Delta w_{ji} = \eta \delta_j y_i$$

$$\frac{\partial E}{\partial w_{ji}} = \frac{\partial E}{\partial v_j} \frac{\partial v_j}{\partial w_{ji}}$$

$$-\frac{\partial E}{\partial v_j} = \delta_j \quad \frac{\partial v_j}{\partial w_{ji}} = y_i$$

PMR5406 Redes Neurais e
Lógica Fuzzy

Multilayer Perceptrons

11

Compute local gradient of neuron j

- The key factor is the calculation of e_j
- There are two cases:
 - Case 1): j is a output neuron
 - Case 2): j is a hidden neuron

PMR5406 Redes Neurais e
Lógica Fuzzy

Multilayer Perceptrons

12

Error e_j of output neuron

- Case 1: ***j output neuron***

$$e_j = d_j - y_j$$

Then

$$\delta_j = (d_j - y_j)\phi'(v_j)$$

Local gradient of hidden neuron

- Case 2: ***j hidden neuron***
- the local gradient for neuron j is recursively determined in terms of the local gradients of all neurons to which neuron j is directly connected

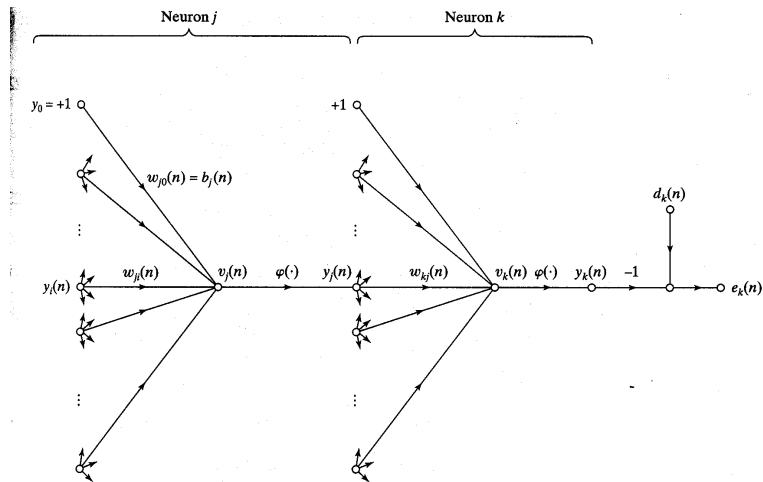


FIGURE 4.4 Signal-flow graph highlighting the details of output neuron k connected to hidden neuron j .

PMR5406 Redes Neurais e
Lógica Fuzzy

Multilayer Perceptrons

15

Use the Chain Rule

$$\delta_j = - \frac{\partial E}{\partial y_j} \frac{\partial y_j}{\partial v_j} \quad \frac{\partial y_j}{\partial v_j} = \phi'(v_j)$$

$$E(n) = \frac{1}{2} \sum_{k \in C} e_k^2(n)$$

$$- \frac{\partial E}{\partial y_j} = - \sum_{k \in C} e_k \frac{\partial e_k}{\partial y_j} = \sum_{k \in C} e_k \left[- \frac{\partial e_k}{\partial v_k} \right] \frac{\partial v_k}{\partial y_j}$$

$$\text{from} \quad - \frac{\partial e_k}{\partial v_k} = \phi'(v_k) \quad \frac{\partial v_k}{\partial y_j} = w_{kj}$$

$$\text{We obtain} \quad - \frac{\partial E}{\partial y_j} = \sum_{k \in C} \delta_k w_{kj}$$

PMR5406 Redes Neurais e
Lógica Fuzzy

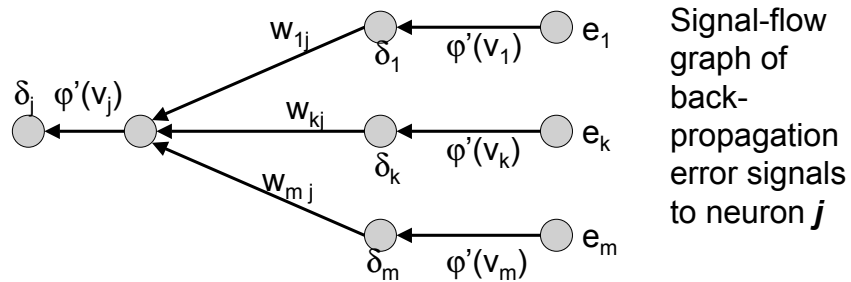
Multilayer Perceptrons

16

Local Gradient of hidden neuron j

Hence

$$\delta_j = \phi'(v_j) \sum_{k \in C} \delta_k w_{kj}$$



PMR5406 Redes Neurais e
Lógica Fuzzy

Multilayer Perceptrons

17

Delta Rule

- **Delta rule** $\Delta w_{ji} = \eta \delta_j y_i$

$$\delta_j = \begin{cases} \phi'(v_j)(d_j - y_j) & \text{IF } j \text{ output node} \\ \phi'(v_j) \sum_{k \in C} \delta_k w_{kj} & \text{IF } j \text{ hidden node} \end{cases}$$

C: Set of neurons in the layer following the one containing j

PMR5406 Redes Neurais e
Lógica Fuzzy

Multilayer Perceptrons

18

Local Gradient of neurons

$$\phi'(v_j) = ay_j[1 - y_j]$$

$$a > 0$$

$$\delta_j = \begin{cases} ay_j[1 - y_j] \sum \delta_k w_{kj} & \text{if } j \text{ hidden node} \\ ay_j[1 - y_j][d_j^k - y_j] & \text{if } j \text{ output node} \end{cases}$$

Backpropagation algorithm

- Two phases of computation:
 - **Forward pass**: run the NN and compute the error for each neuron of the output layer.
 - **Backward pass**: start at the output layer, and pass the errors backwards through the network, layer by layer, by recursively computing the local gradient of each neuron.

Summary

Chapter 4 Multilayer Perceptrons

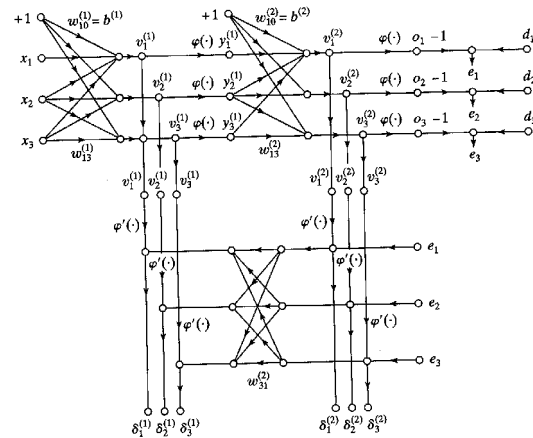


FIGURE 4.7 Signal-flow graphical summary of back-propagation learning. Top part of the graph: forward pass. Bottom part of the graph: backward pass.

PMR5406 Redes Neurais e
Lógica Fuzzy

Multilayer Perceptrons

21

Training

- **Sequential mode** (on-line, pattern or stochastic mode):
 - $(x(1), d(1))$ is presented, a sequence of forward and backward computations is performed, and the weights are updated using the delta rule.
 - Same for $(x(2), d(2)), \dots, (x(N), d(N))$.

PMR5406 Redes Neurais e
Lógica Fuzzy

Multilayer Perceptrons

22

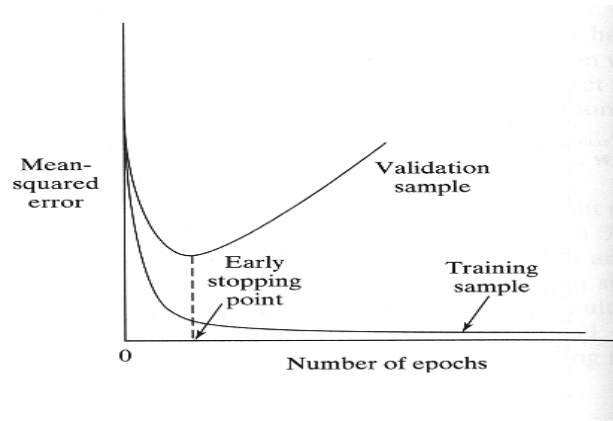
Training

- The learning process continues on an epoch-by-epoch basis until the stopping condition is satisfied.
- From one epoch to the next choose a randomized ordering for selecting examples in the training set.

Stopping criteria

- Sensible stopping criteria:
 - Average squared error change:
Back-prop is considered to have converged when the absolute rate of change in the average squared error per epoch is sufficiently small (in the range $[0.1, 0.01]$).
 - Generalization based criterion:
After each epoch the NN is tested for generalization. If the generalization performance is adequate then stop.

Early stopping



PMR5406 Redes Neurais e
Lógica Fuzzy

Multilayer Perceptrons

25

Generalization

- Generalization: NN generalizes well if the I/O mapping computed by the network is nearly correct for new data (test set).
- Factors that influence generalization:
 - the size of the training set.
 - the architecture of the NN.
 - the complexity of the problem at hand.
- Overfitting (overtraining): when the NN learns too many I/O examples it may end up memorizing the training data.

PMR5406 Redes Neurais e
Lógica Fuzzy

Multilayer Perceptrons

26

Generalization

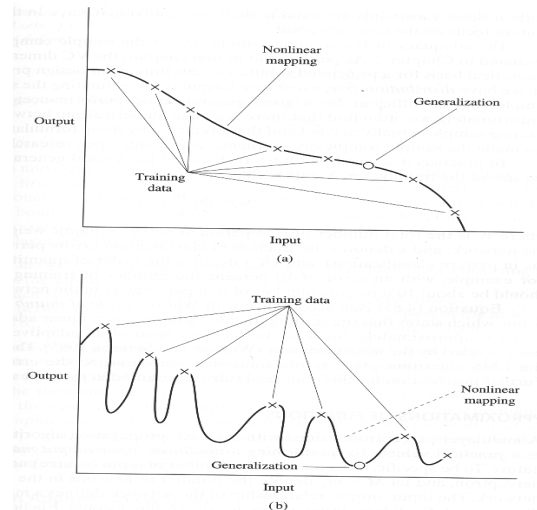


FIGURE 4.19 (a) Properly fitted data (good generalization) (b) Overfitted data (poor generalization).

PMR5406 Redes Neurais e
Lógica Fuzzy

Multilayer Perceptrons

27

Expressive capabilities of NN

Boolean functions:

- Every boolean function can be represented by network with single hidden layer
- but might require exponential hidden units

Continuous functions:

- Every bounded continuous function can be approximated with arbitrarily small error, by network with one hidden layer
- Any function can be approximated with arbitrary accuracy by a network with two hidden layers

PMR5406 Redes Neurais e
Lógica Fuzzy

Multilayer Perceptrons

28

Generalized Delta Rule

- If η small \Rightarrow Slow rate of learning
 If η large \Rightarrow Large changes of weights
 \Rightarrow NN can become unstable (oscillatory)
- Method to overcome above drawback:
include a momentum term in the delta

$$\Delta w_{ji}(n) = \alpha \Delta w_{ji}(n-1) + \eta \delta_j(n) y_i(n)$$

Generalized delta function

momentum constant

Generalized delta rule

- the momentum accelerates the descent in steady downhill directions.
- the momentum has a stabilizing effect in directions that oscillate in time.

η adaptation

Heuristics for accelerating the convergence of the back-prop algorithm through η adaptation:

- Heuristic 1: Every weight should have its own η .
- Heuristic 2: Every η should be allowed to vary from one iteration to the next.

NN DESIGN

- Data representation
- Network Topology
- Network Parameters
- Training
- Validation

Setting the parameters

- How are the weights initialised?
- How is the learning rate chosen?
- How many hidden layers and how many neurons?
- Which activation function ?
- How to preprocess the data ?
- How many examples in the training data set?

Some heuristics (1)

- Sequential x Batch algorithms: the sequential mode (pattern by pattern) is computationally faster than the batch mode (epoch by epoch)

Some heuristics (2)

- Maximization of information content: every training example presented to the backpropagation algorithm must maximize the information content.
 - The use of an example that results in the largest training error.
 - The use of an example that is radically different from all those previously used.

Some heuristics (3)

- Activation function: network learns faster with antisymmetric functions when compared to nonsymmetric functions.

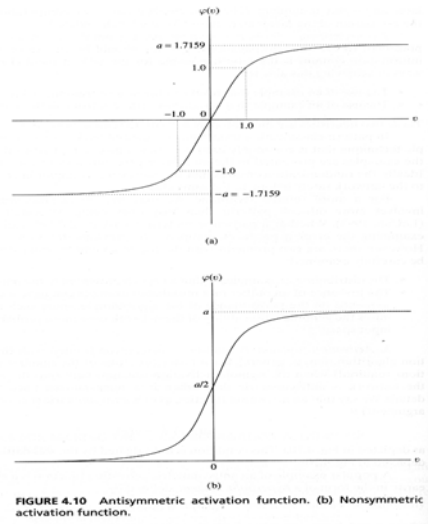
$$\varphi(v) = \frac{1}{1 + e^{-av}}$$

Sigmoidal function is nonsymmetric

$$\varphi(v) = a \tanh(bv)$$

Hyperbolic tangent function is nonsymmetric

Some heuristics (3)



PMR5406 Redes Neurais e
Lógica Fuzzy

Multilayer Perceptrons

37

Some heuristics (4)

- Target values: target values must be chosen within the range of the sigmoidal activation function.
- Otherwise, hidden neurons can be driven into saturation which slows down learning

PMR5406 Redes Neurais e
Lógica Fuzzy

Multilayer Perceptrons

38

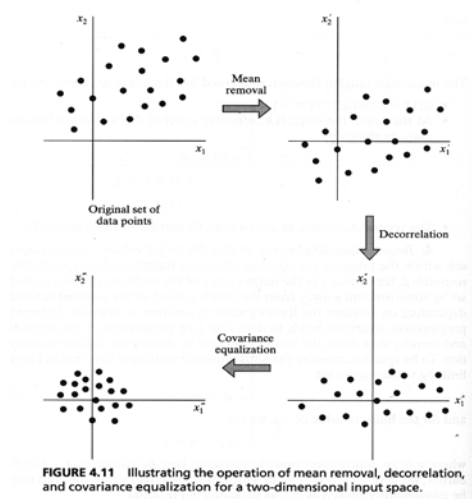
Some heuristics (4)

- For the antisymmetric activation function it is necessary to design ϵ
- For $a+$: $d_j = a - \epsilon$
- For $-a$: $d_j = -a + \epsilon$
- If $a=1.7159$ we can set $\epsilon=0.7159$ then $d=\pm 1$

Some heuristics (5)

- Inputs normalisation:
 - Each input variable should be processed so that the mean value is small or close to zero or at least very small when compared to the standard deviation.
 - Input variables should be uncorrelated.
 - Decorrelated input variables should be scaled so their covariances are approximately equal.

Some heuristics (5)



PMR5406 Redes Neurais e
Lógica Fuzzy

Multilayer Perceptrons

41

Some heuristics (6)

- Initialisation of weights:
 - If synaptic weights are assigned large initial values neurons are driven into saturation. Local gradients become small so learning rate becomes small.
 - If synaptic weights are assigned small initial values algorithms operate around the origin. For the hyperbolic activation function the origin is a saddle point.

PMR5406 Redes Neurais e
Lógica Fuzzy

Multilayer Perceptrons

42

Some heuristics (6)

- Weights must be initialised for the standard deviation of the local induced field v lies in the transition between the linear and saturated parts.

$$\sigma_v = 1$$

$$\sigma_w = m^{-1/2} \quad m = \text{number of weights}$$

Some heuristics (7)

- Learning rate:
 - The right value of η depends on the application. Values between 0.1 and 0.9 have been used in many applications.
 - Other heuristics adapt η during the training as described in previous slides.

Some heuristics (8)

- How many layers and neurons
 - The number of layers and of neurons depend on the specific task. In practice this issue is solved by trial and error.
 - Two types of adaptive algorithms can be used:
 - **start from a large network and successively remove some neurons and links until network performance degrades.**
 - **begin with a small network and introduce new neurons until performance is satisfactory.**

Some heuristics (9)

- How many training data ?
 - Rule of thumb: the number of training examples should be at least five to ten times the number of weights of the network.

Output representation and decision rule

- M-class classification problem

$$Y_{k,j}(x_j) = F_k(x_j), k=1, \dots, M$$



Data representation

$$d_{k,j} = \begin{cases} 1, & x_j \in C_k \\ 0, & x_j \notin C_k \end{cases} \quad \begin{bmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix} \leftarrow \text{Kth element}$$

MLP and the a posteriori class probability

- **A multilayer perceptron classifier (using the logistic function) approximate the a posteriori class probabilities, provided that the size of the training set is large enough.**

The Bayes rule

- An appropriate output decision rule is the (approximate) Bayes rule generated by the a ***posteriori*** probability estimates:
- $x \in C_k$ if $F_k(x) > F_j(x)$ for all $j \neq k$

$$F(x) = \begin{bmatrix} F_1(x) \\ F_2(x) \\ \dots \\ F_M(x) \end{bmatrix}$$