Multi-Valued and Universal Binary Neurons: New Solutions in Neural Networks

Igor Aizenberg
Texas A&M University-Texarkana,
Department of Computer and Information Sciences

igor.aizenberg@tamut.edu
http://www.eagle.tamutedu/faculty/igor
http://www.freewebs.com/igora
NEURAL NETWORKS

Traditional Neurons

Neuro-Fuzzy Networks

Complex-Valued Neurons

Multi-Valued and Universal Binary Neurons

Generalizations of sigmoidal Neurons
Multi-Valued and Universal Binary Neurons (MVN and UBN)

- MVN’s and UBN’s complex-valued activation functions are phase-dependent: their outputs are completely determined by the weighted sum’s argument and they are always lying on the unit circle.
Initial ideas

- Generalization of the principles of Boolean threshold logic for the multiple-valued case
- Extension of a set of the Boolean functions that allow learning using a single neuron (overcoming of the Minsky-Papert limitation)
- Consideration of new corresponding types of neurons
Multi-Valued and Universal Binary Neurons

**Multi-Valued Neuron (MVN):**
learns mappings described by multiple-valued functions (functions of multiple-valued logic including continuous-valued logic)

**Universal Binary Neuron (UBN):**
learns mappings described by Boolean functions that are not necessarily threshold (linearly separable)
Contents

- Multiple-Valued Logic ($k$-valued Logic) over the Field of the Complex Numbers
- Multi-Valued Neuron (MVN)
- Discrete and continuous-valued MVN
- Learning Algorithm for MVN
- Threshold Boolean Functions and $P$-realizable Boolean Functions
- Universal Binary Neuron (UBN)
- Relation between MVN and UBN
- Learning algorithm for UBN
- Solving the XOR and parity problems on a single UBN
- MVN-based multilayer feedforward neural network and its derivative free backpropagation learning algorithm
- Applications
- Further work
Multi-Valued Neuron (MVN)

- A Multi-Valued Neuron is a neural element with \( n \) inputs and one output lying on the unit circle, and with the complex-valued weights.

- The theoretical background behind the MVN is the Multiple-Valued \((k\text{-valued})\) Threshold Logic over the field of complex numbers.
Multiple-Valued Threshold Logic over the field of Complex numbers and a Multi-Valued Neuron
Multiple-Valued Logic: classical view

- The values of multiple-valued (\(k\)-valued) logic are traditionally encoded by the integers \(\{0, 1, \ldots, k-1\}\).
Multiple-Valued Logic over the field of complex numbers: basic idea

- Let $\mathcal{M}$ be an arbitrary additive group and its cardinality is not lower than $k$.

- Let $A_k = \{a_0, a_1, \ldots, a_{k-1}\}, A_k \in \mathcal{M}$ be a structural alphabet.
Multiple-Valued Logic over the field of complex numbers: basic idea

- Let $A_k^n$ be the nth Cartesian power of $A_k$

- **Basic Definition:** Any function of $n$ variables $f(x_1, ..., x_n) | f : A_k^n \to A_k$ is called a function of $k$-valued logic over group $M$
Multiple-Valued Logic over the field of complex numbers: basic idea

- If \( A_k = \{0, 1, \ldots, k - 1\} \) then we obtain traditional \( k \)-valued logic.

- In terms of previous considerations \( A_k = \{0, 1, \ldots, k - 1\} \) is a group with respect to mod \( k \) addition.
Multiple-Valued Logic over the field of complex numbers: basic idea

- Let us take the field of complex numbers $\mathbb{C}$ as a group $M$.
- Let $\varepsilon = \text{exp}(i \frac{2\pi}{k})$, where $i$ is an imaginary unity, be a primitive $k^{th}$ root of unity.
- Let $E_k = \{\varepsilon^0, \varepsilon, \varepsilon^2, ..., \varepsilon^{k-1}\}$ be set $A_k$. 

Important conclusion: Any function of \( n \) variables \( f(x_1, \ldots, x_n) \) is a function of \( k \)-valued logic over the field of complex numbers according to the basic definition
Multiple-Valued \((k\text{-valued})\) logic over the field of complex numbers

(Proposed and formulated by Naum Aizenberg in 1971-1973)

\[ j \in \{0, 1, \ldots, k - 1\} \]

regular values of \(k\)-valued logic

\( j \rightarrow \varepsilon^j = \exp(i2\pi j / k) \)

one-to-one correspondence

\( \varepsilon^j \in \{\varepsilon^0, \varepsilon, \varepsilon^2, \ldots, \varepsilon^{k-1}\} \)

The \(k\)th roots of unity are values of \(k\)-valued logic over the field of complex numbers

\( \varepsilon = \exp(i2\pi / k) \)

primitive \(k\)th root of unity
Important advantage

- In multiple-valued logic over the field of complex numbers all values of this logic are algebraically (arithmetically) equitable: their absolute values are equal to 1
Discrete-Valued ($k$-valued) Activation Function

\[ P(z) = \exp \left( i \frac{2\pi j}{k} \right) = \varepsilon^j, \]

if \( 2\pi \frac{j}{k} \leq \text{arg}(z) < 2\pi \frac{(j + 1)}{k} \)

Function $P$ maps the complex plane into the set of the $k^{th}$ roots of unity
Multiple-Valued ($k$-valued) Threshold Functions

The $k$-valued function $f(x_1,\ldots, x_n)$ is called a $k$-valued threshold function, if such a complex-valued weighting vector $(w_0, w_1,\ldots, w_n)$ exists that for all $X = (x_1,\ldots, x_n)$ from the domain of the function $f$:

$$f(x_1,\ldots, x_n) = P(w_0 + w_1 x_1 + \ldots + w_n x_n)$$
Multi-Valued Neuron (MVN)
(introduced by I.Aizenberg & N.Aizenberg in 1992)

\[ f(x_1,\ldots,x_n) = P(w_0 + w_1 x_1 + \ldots + w_n x_n) \]

\( f \) is a function of \( k \)-valued logic
\( (k\text{-valued threshold function}) \)

\[ z = w_0 + w_1 x_1 + \ldots + w_n x_n \]
The key properties of **MVN**: 
- Complex-valued weights 
- The activation function is a function of the argument of the weighted sum 
- Complex-valued inputs and output that are lying on the unit circle ($k^{th}$ roots of unity) 
- Higher functionality than the one for the traditional neurons (e.g., sigmoidal) 
- Simplicity of learning
MVN Learning

- Learning is reduced to movement along the unit circle
- No derivative is needed, learning is based on the error-correction rule

\[ \delta = E^q - E^s \]
- error, which completely determines the weights adjustment

\[ E^q - \text{Desired output} \]

\[ E^s - \text{Actual output} \]
Learning Algorithm for the Discrete MVN with the Error-Correction Learning Rule

\[ W_{r+1} = W_r + \frac{\alpha_r}{(n+1)} (\varepsilon^q - \varepsilon^s) \bar{X} \]

- \( W \) – weighting vector; \( X \) - input vector
- \( \bar{X} \) is a complex conjugated to \( X \)
- \( \alpha_r \) – learning rate (should be always equal to 1)
- \( r \) - current iteration;
- \( r+1 \) – the next iteration

\( \varepsilon^q \) is a desired output (sector)
\( \varepsilon^s \) is an actual output (sector)
Continuous-Valued Activation Function
(introduced by Igor Aizenberg in 2004)

Continuous-valued case ($k \to \infty$):

$$P(z) = \exp(i \arg z) = e^{i\text{Arg} \, z} = \frac{z}{|z|}$$

Function $P$ maps the complex plane into the unit circle
Learning Algorithm for the Continuous MVN with the Error Correction Learning Rule

\[ W_{r+1} = W_r + \frac{\alpha_r}{(n + 1)} \left( \mathcal{E}^q - \frac{Z_r}{|Z_r|} \right) \overline{X} \]

- \( W \): weighting vector; \( X \): input vector
- \( \overline{X} \): is a complex conjugated to \( X \)
- \( \alpha_r \): a learning rate (should be always equal to 1)
- \( r \): current iteration;
- \( r+1 \): the next iteration
- \( Z \): the weighted sum
- \( \mathcal{E}^q \): is a desired output
- \( \frac{Z}{|Z|} \): is an actual output
- \( \delta = \mathcal{E}^q - \frac{Z}{|Z|} \): neuron’s error
A role of the factor $1/(n+1)$ in the Learning Rule

The weights after the correction:

$$
\tilde{w}_0 = w_0 + \frac{\delta}{n+1}; \quad \tilde{w}_1 = w_1 + \frac{\delta}{n+1} \bar{x}_1; \quad \ldots; \quad \tilde{w}_n = w_n + \frac{\delta}{n+1} \bar{x}_n
$$

The weighted sum after the correction:

$$
\tilde{z} = \tilde{w}_0 x_0 + \tilde{w}_1 x_1 + \ldots + \tilde{w}_n x_n =
$$

$$
= (w_0 + \frac{\delta}{n+1} + (w_1 + \frac{\delta}{n+1} \bar{x}_1) x_1 + \ldots + (w_n + \frac{\delta}{n+1} \bar{x}_n) x_n =
$$

$$
= w_0 + \frac{\delta}{n+1} + w_1 x_1 + \frac{\delta}{n+1} + \ldots + w_n x_n + \frac{\delta}{n+1} =
$$

$$
= w_0 + w_1 x_1 + \ldots + w_n x_n + \delta = z + \delta.
$$
Self-Adaptation of the Learning Rate

\[ W_{r+1} = W_r + \frac{\alpha_r}{(n+1)|z_r|} \left( \varepsilon^q - \frac{z_r}{|z_r|} \right) \bar{X} \]

- \(1/|z_r|\) is a self-adaptive part of the learning rate
- \(|z_r|\) is the absolute value of the weighted sum on the previous (\(r^{th}\)) iteration.
- \(1/|z_r|\) is a self-adaptive part of the learning rate
Modified Learning Rules with the Self-Adaptive Learning Rate

\[ W_{r+1} = W_r + \frac{\alpha_r}{(n+1)|z_r|} (\epsilon^q - \epsilon^s) \bar{X} \]

1/|z_r| is a self-adaptive part of the learning rate

\[ W_{r+1} = W_r + \frac{\alpha_r}{(n+1)|z_r|} \left( \epsilon^q - \frac{z}{|z|} \right) \bar{X} \]

Discrete MVN
Continuous MVN
Convergence of the learning algorithm

- It is proven that the MVN learning algorithm converges after not more than $k!$ iterations for the $k$-valued activation function.

- For the continuous MVN the learning algorithm converges with the precision $\lambda$ after not more than $(\pi/\lambda)!$ iterations because in this case it is reduced to learning in $\pi/\lambda$-valued logic.
MVN as a model of a biological neuron

- The State of a biological neuron is determined by the frequency of the generated impulses.
- The amplitude of impulses is always a constant.

- Excitation → High frequency
- Intermediate State → Medium frequency
- No impulses → Inhibition → Zero frequency
MVN as a model of a biological neuron

- The state of a biological neuron is determined by the frequency of the generated impulses.
- The amplitude of impulses is always a constant.
- The state of the multi-valued neuron is determined by the argument of the weighted sum.
- The amplitude of the state of the multi-valued neuron is always a constant.
MVN as a model of a biological neuron

Intermediate State

Maximal inhibition

Intermediate State

Maximal excitation
MVN as a model of a biological neuron

Maximal inhibition

Maximal excitation
MVN:

- Learns faster
- Adapts better
- Learns even highly nonlinear functions
- Opens new very promising opportunities for the network design
- Is much closer to the biological neuron
- Allows to use the Fourier Phase Spectrum as a source of the features for solving different recognition/classification problems
- Allows to use hybrid (discrete/continuous) inputs/output
$P$-realizable Boolean Functions over the field of Complex numbers and a Universal Binary Neuron
Boolean Alphabets

\{0, 1\} – a classical alphabet

\{1, -1\} – an alternative alphabet used here

\[ y \in \{0, 1\}; \quad x \in \{1, -1\} \]

\[ x = 1 - 2y \implies 0 \rightarrow 1; \quad 1 \rightarrow -1 \]
Threshold Boolean Functions

(a classical definition)

The Boolean function $f(x_1, \ldots, x_n)$ is called a threshold function, if such a real-valued weighting vector $(w_0, w_1, \ldots, w_n)$ exists that for all $x \in X = (x_1, \ldots, x_n)$:

$$
\begin{cases}
    w_0 + w_1 x_1 + \ldots + w_n x_n \geq 0, & \text{if } f(x_1, \ldots, x_n) = 1 \\
    w_0 + w_1 x_1 + \ldots + w_n x_n < 0, & \text{if } f(x_1, \ldots, x_n) = -1
\end{cases}
$$
A threshold function is a linearly separable function. 

\[
\begin{align*}
  f(-1, 1) &= -1 \\
  f(1, 1) &= 1 \\
  f(-1, -1) &= -1 \\
  f(1, -1) &= -1
\end{align*}
\]

\[f(x_1, x_2)\] is the OR function.
Threshold Boolean Functions

- The threshold function could be implemented on the single neuron.
- The number of threshold functions is very small in comparison to the number of all functions (104 of 256 for $n=3$, about 2000 of 65536 for $n=4$, etc.).
- Non-threshold functions cannot be implemented on a single neuron (Minsky-Papert, 1969).
XOR – a classical non-threshold function

\[ f(-1,1) = -1 \]
\[ f(1,1) = 1 \]
\[ f(-1,-1) = 1 \]
\[ f(1,-1) = -1 \]
Is it possible to overcome the Minsky’s-Papert’s limitation for the classical perceptron?

Yes !!!
$P$-realizable Boolean functions

(introduced by I. Aizenberg in 1984-1985)

Binary activation function:

$$P_B(z) = (-1)^j,$$

if $$(2\pi j/m) \leq \arg(z) < (2\pi (j+1)/m)$$

$$P_B(z) = 1$$
The Boolean function $f(x_1,\ldots,x_n)$ is called a $P$-realizable function, if such a complex-valued weighting vector $(w_0, w_1,\ldots, w_n)$ exists that for all $X = (x_1,\ldots,x_n)$ from the domain of the function $f$: \[
abla x_1,\ldots, x_n \implies f(x_1,\ldots,x_n) = P_B (w_0 + w_1 x_1 + \ldots + w_n x_n)\]
Universal Binary Neuron (UBN) (introduced in 1985)

\[ f(x_1, \ldots, x_n) = P_B(w_0 + w_1 x_1 + \ldots + w_n x_n) \]

\[ z = w_0 + w_1 x_1 + \ldots + w_n x_n \]
The key properties of UBN:
- Complex-valued weights
- The activation function is a function of the argument of the weighted sum
- Ability to learn arbitrary (not necessarily threshold) Boolean functions on a single neuron
- Simplicity of learning
XOR problem

\[ P_B = -1 \quad \quad \quad \quad P_B = 1 \]

\[
\begin{array}{c|c}
-1 & i \\
-1 & 1 \\
-1 & -1 \\
1 & 1 \\
1 & -1 \\
-1 & 1 \\
-1 & -1 \\
\end{array}
\]

\[ n=2, \ m=4 \quad \text{– four sectors} \]
\[ W=(0, \ 1, \ i) \quad \text{– the weighting vector} \]

\[
\begin{array}{c|c|c|c|c}
x_1 & x_2 & z = w_0 + w_1 x_1 + w_2 x_2 & P_B(z) & f(x_1, x_2) \\
1 & 1 & 1+i & 1 & 1 \\
1 & -1 & 1-i & -1 & -1 \\
-1 & 1 & -1+i & -1 & -1 \\
-1 & -1 & -1-i & 1 & 1 \\
\end{array}
\]
### Parity problem

$n=3, m=6 : 6$ sectors

$W=(0, \varepsilon, 1, 1) -$ the weighting vector

<table>
<thead>
<tr>
<th>$X_1$</th>
<th>$X_2$</th>
<th>$X_3$</th>
<th>$Z = w_0 + w_1 x_1 + w_2 x_2 + w_3 x_3$</th>
<th>$P_B(Z)$</th>
<th>$f(x_1, x_2, x_3)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>$\varepsilon + 2$</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>$\varepsilon$</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>$\varepsilon$</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>$\varepsilon - 2$</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>$-\varepsilon + 2$</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>-1</td>
<td>1</td>
<td>-1</td>
<td>$-\varepsilon$</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>$-\varepsilon$</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>$-\varepsilon - 2$</td>
<td>-1</td>
<td>-1</td>
</tr>
</tbody>
</table>
Dependence of the number of sectors on the number of inputs

- A proven estimation for $m$, to ensure $P$-realizibility of all functions

- Estimation for self-dual and even functions (experimental fact, mathematically open problem)

- $P$-realizability of all functions may be reduced to $P$-realizability of self-dual and even functions

\[ 2n \leq m \leq 2^{2n} \]

\[ 2n \leq m \leq 3n \]
Relation between $P$-realizable Boolean functions and $k$-valued threshold functions

- If the Boolean function $f$ is $P$-realizable with a separation of the complex plane into $m$ sectors then at least one $m$-valued threshold function $f_m$ (partially defined on the Boolean domain) exists, whose weighting vector is the same that for the Boolean function $f$.

- The learning of $P$-realizable function may be reduced to the learning of partially defined $m$-valued threshold function
Learning Algorithm for UBN

\[ W_{r+1} = W_r + \frac{\alpha_r (\epsilon^q - \epsilon^s)}{(n+1)} X, \]

Semi-supervised/semi-self-adaptive learning

\[ q = s-1 \text{ (mod } m) \text{, if } Z \text{ is closer to } (s-1)^{st} \text{ sector} \]

\[ q = s+1 \text{ (mod } m) \text{, if } Z \text{ is closer to } (s+1)^{st} \text{ sector} \]
Main Applications
MVN- based Multilayer Feedforward Neural Network (MLMVN) and a Backpropagation Learning Algorithm

Suggested and developed by Igor Aizenberg et al in 2004-2006
MVN-based Multilayer Feedforward Neural Network

Hidden layers          Output layer
MLMVN: Key Properties

- Derivative-free learning algorithm based on the error-correction rule
- Self-adaptation of the learning rate for all the neurons
- Much faster learning than the one for other neural networks
- A single step is always enough to adjust the weights for the given set of inputs independently on the number of hidden and output neurons
- Better recognition/prediction/classification rate in comparison with other neural networks, neuro-fuzzy networks and kernel based techniques including SVM
MLMVN: Key Properties

- MLMVN can operate with both continuous and discrete inputs/outputs, as well as with the hybrid inputs/outputs:
  - continuous inputs $\rightarrow$ discrete outputs
  - discrete inputs $\rightarrow$ continuous outputs
  - hybrid inputs $\rightarrow$ hybrid outputs
A Backpropagation Derivative-Free Learning Algorithm

\[ T_{km} \] - a desired output of the \( k \)th neuron from the \( m \)th (output) layer

\[ Y_{km} \] - an actual output of the \( k \)th neuron from the \( m \)th (output) layer

\[ \delta^*_{km} = T_{km} - Y_{km} \] - the network error for the \( k \)th neuron from output layer

\[ \delta_{km} = \frac{1}{S_m} \delta^*_{km} \] - the error for the \( k \)th neuron from output layer

\[ S_m = N_{m-1} + 1 \] - the number of all neurons on the previous layer \((m-1, \text{to which the error is backpropagated})\) incremented by 1
A Backpropagation Derivative-Free Learning Algorithm

The error backpropagation:

The error for the $k^{th}$ neuron from the hidden ($j^{th}$) layer, $j=1, \ldots, m-1$

$$
\delta_{kj} = \frac{1}{s_j} \sum_{i=1}^{N_{j+1}} \frac{1}{w_{k}^{ij+1}} \left( \delta_{ij+1} (\bar{w}_{k}^{ij+1}) \right) = \frac{1}{s_j} \sum_{i=1}^{N_{j+1}} \delta_{ij+1} (w_{k}^{ij+1})^{-1}
$$

$s_j = N_{j-1} + 1, \quad j = 2, \ldots, m; \quad s_1 = 1$

-the number of all neurons on the previous layer
(previous to $j$, to which the error is backpropagated) incremented by 1
Correction rule for the neurons from the $m^{th}$ (output) layer ($k^{th}$ neuron of $m^{th}$ layer):

\[
\tilde{w}_i^{km} = w_i^{km} + \frac{C_{km}}{(n+1)} \delta_{km} \tilde{Y}_{im-1}, \quad i = 1, \ldots, n
\]

\[
\tilde{w}_0^{km} = w_0^{km} + \frac{C_{km}}{(n+1)} \delta_{km}
\]
A Backpropagation Derivative-Free Learning Algorithm

Correction rule for the neurons from the 2nd through m-1st layer (kth neuron of the jth layer (j=2, ..., m-1):

\[
\tilde{w}_{i}^{kj} = w_{i}^{kj} + \frac{C_{kj}}{(n+1) |z_{kj}|} \delta_{kj} \tilde{Y}_{i}, \quad i = 1, ..., n
\]

\[
\tilde{w}_{0}^{kj} = w_{0}^{kj} + \frac{C_{kj}}{(n+1) |z_{kj}|} \delta_{kj}
\]
A Backpropagation Derivative-Free Learning Algorithm

Correction rule for the neurons from the 1st hidden layer:

\[
\tilde{w}_i^{k1} = w_i^{k1} + \frac{C_{k1}}{(n+1)|z_{k1}|} \delta_{k1} x_i, \quad i = 1, \ldots, n
\]

\[
\tilde{w}_0^{k1} = w_0^{k1} + \frac{C_{k1}}{(n+1)|z_{k1}|} \delta_{k1}
\]
Criteria for the convergence of the learning process

Learning should continue until either minimum MSE/RMSE criterion will be satisfied or zero-error will be reached.
MSE criterion

\[
\frac{1}{N} \sum_{s=1}^{N} \sum_{k} (\delta_{km_s}^*)^2 (W) = \frac{1}{N} \sum_{s=1}^{N} E_s \leq \lambda
\]

\(\lambda\) is a maximum possible MSE for the training data

\(N\) is the number of patterns in the training set

\(E_s = \sum_{k} (\delta_{km_s}^*)^2\) is the network square error for the \(s^{th}\) pattern
RMSE criterion

\[
\sqrt{\frac{1}{N} \sum_{s=1}^{N} \sum_{k} (\delta_{km}^*)^2 (W)} = \sqrt{\frac{1}{N} \sum_{s=1}^{N} E_s} \leq \lambda
\]

- $\lambda$ is a maximum possible RMSE for the training data
- $N$ is the number of patterns in the training set
- $E_s = \sum_k (\delta_{km}^*)^2$ is the network square error for the $s^{th}$ pattern
MLMVN Learning: Example

Suppose, we need to classify three vectors belonging to three different classes:

\[ X_1 = (\exp(4.23i), \exp(2.10i)) \rightarrow \tilde{T}_1, \]
\[ X_2 = (\exp(5.34i), \exp(1.24i)) \rightarrow \tilde{T}_2, \]
\[ X_3 = (\exp(2.10i), \exp(0i)) \rightarrow \tilde{T}_3. \]

\[ T_1 = \exp(0.76i), \quad T_2 = \exp(2.56i), \quad T_3 = \exp(5.35i). \]

Classes \[ \tilde{T}_1, \tilde{T}_2, \tilde{T}_3 \]

are determined in such a way that the argument of the desired output of the network must belong to the interval

\[ [\arg(T_j) - 0.05, \arg(T_j) + 0.05], \quad j = 1, 2, 3, \]
MLMVN Learning: Example

Thus, we have to satisfy the following conditions:

\[ |\arg(T_j) - \arg(e^{i\text{Arg } z})| \leq 0.05 \]

, where

\[ e^{i\text{Arg } z} \]

is the actual output.

and for the mean square error

\[ E \leq 0.05^2 = 0.0025 \]
MLMVN Learning: Example

Let us train the 2→1 MLMVN
(two hidden neurons and the single neuron in the output layer)

\[ f(x_1, x_2) \]
MLMVN Learning: Example

The training process converges after 7 training epochs.

Update of the outputs:
### MLMVN Learning: Example

<table>
<thead>
<tr>
<th>Epoch</th>
<th>MSE</th>
<th>MSE</th>
<th>MSE</th>
<th>MSE</th>
<th>MSE</th>
<th>MSE</th>
<th>MSE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2.4213</td>
<td>0.1208</td>
<td>0.2872</td>
<td>0.1486</td>
<td>0.0049</td>
<td>0.0026</td>
<td>0.0009</td>
</tr>
</tbody>
</table>

\[ E \leq 0.05^2 = 0.0025 \]
MLMVN: Simulation Results
Simulation Results: Benchmarks

All simulation results for the benchmark problems are obtained using the network with $n$ inputs:

- $n \rightarrow S \rightarrow 1$ containing
- a single hidden layer with $S$ neurons
- and a single neuron in the output layer:
The Two Spirals Problem
Two Spirals Problem: complete training of 194 points

<table>
<thead>
<tr>
<th>Structure of the network</th>
<th>A network type</th>
<th>The results</th>
</tr>
</thead>
<tbody>
<tr>
<td>2→40→1</td>
<td>MLMVN</td>
<td>Trained completely, no errors</td>
</tr>
<tr>
<td></td>
<td>A traditional MLP, sigmoid activation function</td>
<td>still remain 4% errors</td>
</tr>
<tr>
<td>2→30→1</td>
<td>MLMVN</td>
<td>Trained completely, no errors</td>
</tr>
<tr>
<td></td>
<td>A traditional MLP, sigmoid activation function</td>
<td>still remain 14% errors</td>
</tr>
</tbody>
</table>
Two Spirals Problem:
cross-validation (training of 98 points and prediction of the rest 96 points)

The prediction rate is stable for all the networks from $2 \rightarrow 26 \rightarrow 1$ till $2 \rightarrow 40 \rightarrow 1$:

68-72%

The prediction rate of 74-75% appears 1-2 times per 100 experiments with the network $2 \rightarrow 40 \rightarrow 1$

The best known result obtained using the Fuzzy Kernel Perceptron (Nov. 2002) is 74.5%, But it is obtained using more complicated and larger network
The “Sonar” problem

208 samples:
60 continuous-valued inputs
1 binary output

The “sonar” data is trained completely
using the simplest possible MLMVN $60 \rightarrow 2 \rightarrow 1$
(from 817 till 3700 epochs in 50 experiments)
The “Sonar” problem: prediction

104 samples in the training set
104 samples in the testing set

The prediction rate is stable for 100 experiments with the simplest possible MLMVN $60 \rightarrow 2 \rightarrow 1$: 88-93%

The best result for the Fuzzy Kernel Perceptron (Nov., 2002) is 94%. The best result for SVM is 89.5%. However, both FKP and SVM used are larger and their training is more complicated task.
Mackey-Glass time series prediction

Mackey-Glass differential delay equation:

\[
\frac{dx(t)}{dt} = \frac{0.2x(t - \tau)}{1 + x^{10}(t - \tau)} - 0.1x(t) + n(t),
\]

The task of prediction is to predict \( x(t + 6) \)

from \( x(t), x(t - 6), x(t - 12), x(t - 18) \)
Mackey-Glass time series prediction

Training Data:

Testing Data:
Mackey-Glass time series prediction

RMSE Training:

RMSE Testing:
Mackey-Glass time series prediction

Testing Results:

Blue curve – the actual series;
Red curve – the predicted series
Mackey-Glass time series prediction

The results of 30 independent runs:

<table>
<thead>
<tr>
<th># of neurons on the hidden layer</th>
<th>50</th>
<th>50</th>
<th>40</th>
</tr>
</thead>
<tbody>
<tr>
<td>ε- a maximum possible RMSE</td>
<td>0.0035</td>
<td>0.0056</td>
<td>0.0056</td>
</tr>
<tr>
<td>Actual RMSE for the training set (min - max)</td>
<td>0.0032 - 0.0035</td>
<td>0.0053 – 0.0056</td>
<td>0.0053 – 0.0056</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RMSE for the testing set</th>
<th>Min</th>
<th>Max</th>
<th>Median</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min</td>
<td>0.0056</td>
<td>0.0083</td>
<td>0.0089</td>
<td>0.0097</td>
</tr>
<tr>
<td>Max</td>
<td>0.0083</td>
<td>0.0101</td>
<td>0.0089</td>
<td>0.0098</td>
</tr>
<tr>
<td>Median</td>
<td>0.0063</td>
<td>0.0089</td>
<td>0.0089</td>
<td>0.0098</td>
</tr>
<tr>
<td>Average</td>
<td>0.0066</td>
<td>0.0089</td>
<td>0.0098</td>
<td>0.0098</td>
</tr>
<tr>
<td>SD</td>
<td>0.0009</td>
<td>0.0005</td>
<td>0.0011</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Number of training epochs</th>
<th>Min</th>
<th>Max</th>
<th>Median</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min</td>
<td>95381</td>
<td>272660</td>
<td>145137</td>
<td>162180</td>
</tr>
<tr>
<td>Max</td>
<td>24754</td>
<td>116690</td>
<td>56295</td>
<td>58903</td>
</tr>
<tr>
<td>Median</td>
<td>34406</td>
<td>137860</td>
<td>62056</td>
<td>70051</td>
</tr>
</tbody>
</table>
Mackey-Glass time series prediction

Comparison of MVN to other models:

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0056</td>
<td>0.0066</td>
<td>0.0061</td>
<td>0.02</td>
<td>0.0074</td>
<td>0.009</td>
<td>0.014</td>
<td>0.02</td>
</tr>
</tbody>
</table>

MLMVN outperforms all other networks in:

• The number of either hidden neurons or supporting vectors
• Speed of learning
• Prediction quality
Real World Problems:
Solving Using MLMVN
Classification of gene expression microarray data
“Lung” and “Novartis” data sets

- 4 classes in both data sets
- “Lung”: 197 samples with 419 genes
- “Novartis”: 103 samples with 697 genes
$K$-folding cross validation with $K=5$

- “Lung”: 117-118 samples of 197 in each training set and 39-40 samples of 197 in each testing set
- “Novartis”: 80-84 samples of 103 in each training set and 19-23 samples of 103 in each testing set
MLMVN

- Structure of the network: \( n \rightarrow 6 \rightarrow 4 \) (\( n \) inputs, 6 hidden neurons and 4 output neurons)
- The learning process converges very quickly (500-1000 iterations that take up to 1 minute on a PC with Pentium IV 3.8 GHz CPU are required for both data sets)
Results: Classification Rates

- MLMVN: 98.55% ("Novartis")
- MLMVN: 95.945% ("Lung")
- kNN (k=1) 97.69% ("Novartis")
- kNN (k=2) 92.55% ("Lung")
Generation of the Genetic Code using MLMVN
Genetic code

- There are exactly four different nucleic acids: Adenosine (A), Thymidine (T), Cytidine (C) and Guanosine (G)
- Thus, there exist $4^3=64$ different combinations of them “by three”.
- Hence, the genetic code is the mapping between the four-letter alphabet of the nucleic acids (DNA) and the 20-letter alphabet of the amino acids (proteins)
Genetic code as a multiple-valued function

- Let $G_j, j = 1, \ldots, 20$ be the amino acid
- Let $x_i \in \{A, G, C, T\}, i = 1, 2, 3$ be the nucleic acid
- Then a discrete function of three variables
  \[ G_j = f(x_1, x_2, x_3) \]
- is a function of a 20-valued logic, which is partially defined on the set of four-valued variables
Genetic code as a multiple-valued function

- A multiple-valued logic over the field of complex numbers is a very appropriate model to represent the genetic code function.

- This model allows to represent mathematically those biological properties that are the most essential (e.g., complementary nucleic acids $A \leftrightarrow T; \ G \leftrightarrow C$ that are stuck to each other in a DNA double helix only in this order).
DNA double helix

The complementary nucleic acids

\[ A \leftrightarrow T; \ G \leftrightarrow C \]

are always stuck to each other in pairs.
The complementary nucleic acids

\[ A \leftrightarrow T; \quad G \leftrightarrow C \]

are located such that

\[ A=1; \quad G=i \]

\[ T = -A = -1 \quad \text{and} \]

\[ C = -G = -i \]

All amino acids are distributed along the unit circle in the logical way, to insure their closeness to those nucleic acids that form each certain amino acid.
Generation of the genetic code

- The genetic code can be generated using MLMVN $3 \rightarrow 4 \rightarrow 1$ (3 inputs, 4 hidden neurons and a single output neuron – 5 neurons)

- The best known result for the classical backpropagation neural network is generation of the code using the network $3 \rightarrow 12 \rightarrow 2 \rightarrow 20$ (32 neurons)
Blurred Image Restoration (Deblurring) and Blur Identification by MLMVN
Problem statement: capturing

- Mathematically, a variety of capturing principles can be described by the Fredholm integral of the first kind:

\[ z(x) = \int_{\mathbb{R}^2} v(x, t) y(t) dt, \quad x, t \in \mathbb{R}^2 \]

- Where \( x, t \in \mathbb{R}^2 \), \( v(t) \) is a point-spread function (PSF) of a system, \( y(t) \) is a function of a real object, and \( z(x) \) is an observed signal.
Image deblurring: problem statement

- Mathematically, blur is caused by the convolution of an image with the distorting kernel.
- Thus, removal of the blur is reduced to the deconvolution.
- Deconvolution is an ill-posed problem, which results in the instability of a solution. The best way to solve it is to use some regularization technique.
- To use any kind of regularization technique, it is absolutely necessary to know the distorting kernel corresponding to a particular blur: so it is necessary to identify the blur.
The observed image given in the following form:

\[ z(x) = (v * y)(x) + \varepsilon(x), \]

where "\(*\)" denotes a convolution operation, \( y \) is an image, \( v \) is point-spread function of a system (PSF), which is exactly a distorting kernel, and \( \varepsilon \) is a noise.

In the continuous frequency domain the model takes the form:

\[ Z(\omega) = V(\omega) \cdot Y(\omega) + \varepsilon(\omega), \]

where \( Z(\omega) = F\{z(x)\}, \omega \in R^2 \) is a representation of the signal \( z \) in the Fourier domain and \( F\{\} \) denotes a Fourier transform.
Blur Identification

- We use MLMVN to recognize **Gaussian, motion** and **rectangular (boxcar)** blurs.

- We aim to identify simultaneously both blur (PSF), and its parameters using a single neural network.
Considered PSF

**PSF in time domain**

**PSF in frequency domain**

Gaussian  
Motion  
Rectangular
Considered PSF

The **Gaussian** PSF:

\[ v(t) = \frac{1}{2\pi \tau^2} \exp \left( -\frac{t_1^2 + t_2^2}{\tau^2} \right) \]

\( \tau^2 \) is a parameter (variance)

The **uniform linear motion**:

\[ v(t) = \begin{cases} 
\frac{1}{h}, & \sqrt{t_1^2 + t_2^2} < h/2, \ t_1 \cos \phi = t_2 \sin \phi, \\
0, & \text{otherwise},
\end{cases} \]

\( h \) is a parameter (the length of motion)

The **uniform rectangular**:

\[ v(t) = \begin{cases} 
\frac{1}{h^2}, & |t_1| < \frac{h}{2}, \ |t_2| < \frac{h}{2}, \\
0, & \text{otherwise},
\end{cases} \]

\( h \) is a parameter (the size of smoothing area)
Degradation in the frequency domain:

True Image  Gaussian  Rectangular  Horizontal Motion  Vertical Motion

Images and log of their Power Spectra $\log|Z|$
Training Vectors

- We state the problem as a recognition of the shape of $V$, which is a Fourier spectrum of PSF $v$ and its parameters from the Power-Spectral Density, whose distortions are typical for each type of blur.
Training Vectors

The **training vectors** $X = (x_1, \ldots, x_n)$ are formed as follows:

\[
x_j = \exp \left( 2\pi i \cdot (K - 1) \frac{\log \left( |Z(\omega_{k_1,k_2})| \right) - \log (|Z_{\min}|)}{\log (|Z_{\max}|) - \log (|Z_{\min}|)} \right),
\]

for

\[
\begin{align*}
j &= 1, \ldots, L/2 - 1, & & \text{for } k_1 = k_2, \ k_2 = 1, \ldots, L/2 - 1, \\
j &= L/2, \ldots, L - 2, & & \text{for } k_1 = 1, \ k_2 = 1, \ldots, L/2 - 1, \\
j &= L - 1, \ldots, 3L/2 - 3, & & \text{for } k_2 = 1, \ k_1 = 1, \ldots, L/2 - 1,
\end{align*}
\]

- $L \times L$ is a size of an image
- the length of the **pattern vector** is $n = 3L/2-3$
Examples of training vectors

- True Image
- Gaussian
- Rectangular
- Horizontal Motion
- Vertical Motion
Neural Network

Training (pattern) vectors

Hidden layers

Output layer

Blur 1
Blur 2
Blur N
Output Layer Neuron

Reservation of domains on the unit circle for the output neuron
Experiment 1 (2700 training pattern vectors corresponding to 72 images): six types of blur with the following parameters:

MLMVN structure: 5$\rightarrow$35$\rightarrow$6

1) The Gaussian blur is considered with $\tau \in \{1, 1.33, 1.66, 2, 2.33, 2.66, 3\}$;
2) The linear uniform horizontal motion blur of the lengths 3, 5, 7, 9;
3) The linear uniform vertical motion blur of the length 3, 5, 7, 9;
4) The linear uniform diagonal motion from South-West to North-East blur of the lengths 3, 5, 7, 9;
5) The linear uniform diagonal motion from South-East to North-West blur of the lengths 3, 5, 7, 9;
6) rectangular has sizes 3x3, 5x5, 7x7, 9x9.
## Results

### Classification Results

<table>
<thead>
<tr>
<th>Blur</th>
<th>MLMVN, 381 inputs, 5→35→6, 2336 weights in total</th>
<th>SVM Ensemble from 27 binary decision SVMs, 25,717,500 support vectors in total</th>
</tr>
</thead>
<tbody>
<tr>
<td>No blur</td>
<td>96.0%</td>
<td>100.0%</td>
</tr>
<tr>
<td>Gaussian</td>
<td>99.0%</td>
<td>99.4%</td>
</tr>
<tr>
<td>Rectangular</td>
<td>99.0%</td>
<td>96.4</td>
</tr>
<tr>
<td>Motion horizontal</td>
<td>98.5%</td>
<td>96.4</td>
</tr>
<tr>
<td>Motion vertical</td>
<td>98.3%</td>
<td>96.4</td>
</tr>
<tr>
<td>Motion North-East Diagonal</td>
<td>97.9%</td>
<td>96.5</td>
</tr>
<tr>
<td>Motion North-West Diagonal</td>
<td>97.2%</td>
<td>96.5</td>
</tr>
</tbody>
</table>
Restored images

Blurred noisy image: rectangular 9x9

Blurred noisy image: Gaussian, \( \sigma = 2 \)

Restored

Restored
Main Publications


Main Publications


A book

Igor Aizenberg, Naum Aizenberg and Joos Vandewalle

The latest publications


Some Prospective Applications

- Solving different recognition and classification problems, especially those, where the formalization of the decision rule is a complicated problem
  - Classification and prediction in biology
  - Time series prediction
  - Modeling of complex systems including hybrid systems that depend on many parameters
  - Nonlinear filtering (MLMVN can be used as a nonlinear filter)
  - Etc. …
CONCLUSION

If it be now, 'tis not to come;
if it be not to come, it will be now;
if it be not now, yet it will come:
the readiness is all…

Shakespeare

The best results are expected in the near future…