Synthesis of parameterized families of correctly functioning sigma-pi neurons

Z M Shibzukhov$^{1,2}$

$^1$ Institute of Mathematics and Informatics of Moscow Pedagogical State University, Moscow, Russia
$^2$ Moscow Institute of Physics and Technology, Moscow, Russia

E-mail: intellimath@mail.ru

Abstract. The ΣΠ-neuron is a biologically inspired formal model for logical information processing. The ΣΠ-neuron model adequately reflects information processing processes in the cerebral cortex and in the dendritic trees of neurons. The advantage of the ΣΠ-neuron model is the ability to accurately represent any Boolean function and the possibility of constructive learning (direct construction) in a single pass of the training sample. Another possibility is the direct construction of an ensemble of ΣΠ-neurons that function correctly on the training sample. This article discusses a new algorithm for constructing such ensemble of ΣΠ-neurons in parameterized form. This form can also be easily represented as a single ΣΠ-neuron with a hidden layer of linear and threshold linear units. In some cases, this makes it easier to retrain on new inputs by setting the appropriate control parameter values.

1. Introduction

The ΣΠ-neuron is biologically inspired formal model for logical information processing. The ΣΠ-neuron model adequately reflects the processes of information processing in the cerebral cortex [1,2] and in dendritic trees of neurons [3]. ΣΠ-neuron is able to represent arbitrary partial boolean function. The advantage of the ΣΠ-neuron model is the possibility of its constructive training (direct construction) in one pass of the training sample [4]. Using the constructive training procedure, it is also possible to build ensembles of ΣΠ-neurons. At the same time, each of them will function correctly on the training sample [5,6]. The ΣΠ-neuron model is very effective in correcting Boolean algorithms [7]. Effective electrical circuit design for ΣΠ-neurons was discussed in [8]. Using ΣΠ-neurons can reduce the number of layers in deep networks architecture [9].

The ΣΠ-neuron model defines a boolean function in the following form:

\[ y = \text{spn}(x) = \text{sgn}(\text{sp}(x)), \]

where \( y \in \{0,1\}, \ x \in \{0,1\}^n, \ \text{sgn}(s) = [s \geq 0], \ [S] = 1 \text{ if } S \text{ is true and } [S] = 0 \text{ if } S \text{ is false.} \]

\[ \text{sp}(x) = \theta + \sum_{k=1}^{M} w_k \prod_{i \in k} x_i, \]

(1)
\( \theta \in \mathbb{R} \) is a free constant, \( w_k \in \mathbb{R} \) is a weight of the \( k \)-th synaptic cluster formed by inputs with indices from the multi-index \( I_k \subseteq \{1, \ldots, n\} \).

There is a constructive procedure for constructing (possibly large) ensembles of \( \Sigma\Pi \)-neurons that function correctly on their training set. This is very useful when building majority-voting procedures that help to increase stability and generalization ability of the \( \Sigma\Pi \)-neuron model. The possibility of parametric representation of such ensembles could allow us to solve the problems of searching for \( \Sigma\Pi \)-neurons that meet certain criteria that can be formulated in terms of these parameters (for example, find \( \Sigma\Pi \)-neurons with a potentially minimal number of multiplicative terms). It can also simplify the retraining process for new inputs.

In this paper, we propose a method for constructive training of parametric families of \( \Sigma\Pi \)-neurons that function correctly on a training set. By setting the values of free parameters, you can get all \( \Sigma\Pi \)-neurons from this family. In addition, it will be shown that the values of the weights of \( \Sigma\Pi \)-neurons can be calculated using linear and threshold-linear neurons with integer weights, the input of which receives the values of free parameters. The example shows the possibility of easy additional training by setting the values of free parameters.

2. The problem
Let \( X = \{x_1, \ldots, x_N\} \) be a finite set of Boolean \( n \)-dimensional vectors, \( Y = \{y_1, \ldots, y_N\} \) be a set of Boolean values \( (y_k \in \{0,1\}) \) corresponding to the set \( X \). We consider Boolean functions \( f(x) \) such that \( y_k = f(x_k) \) for all \( k = 1, \ldots, N \).

It is required to construct the representation of \( f(x) \) in the form of a \( \Sigma\Pi \)-neuron. Since for \( N \ll 2^n \) there is a very large number of such representations, it is necessary to obtain some parametric description of this family, which would make it possible to extract \( \Sigma\Pi \)-neurons from it that satisfy some optimization criteria.

To describe the algorithm for constructing a \( \Sigma\Pi \)-neuron, we define the relation \( \succeq \) on the set of Boolean vectors.

**Definition.** For \( x', x'' \in \{0,1\}^n \) and \( i \subseteq \{1, \ldots, n\} \) we have relation \( x' \succeq x'' \) if there is \( i \in \) such that \( x'_i < x''_i \) (i.e., \( x'_i = 0 \) and \( x''_i = 1 \)).

We define the relation of triangular ordering on the set of Boolean vectors.

**Definition.** The sequence \( X = \{x_1, \ldots, x_N\} \) is triangularly ordered with respect to the sequence of multi-indices \( I = \{i_1, \ldots, i_N\} \) if

- for all \( k = 1, \ldots, N \) and all \( i \in I_k \) there is \( x_{ki} = 1 \);
- for all pairs \( j < k : x_j \succeq x_k \).

**Fact.** If \( X \) is ordered by increasing number of ones in the vector \( x \), then it is triangularly ordered with respect to \( I(X) = \{i(x_1), \ldots, i(x_N)\} \), where \( i(x) = \{i : x_i = 1\} \).

The concept of triangular ordering has the following consequences when using the operator of the product of Boolean values.

**Fact.** If \( X \) is triangularly ordered with respect to \( I \), then

- for all \( k = 1, \ldots, N : \prod_{i \in I_k} x_{ki} = 1 \);
- for all pairs \( j < k : \prod_{i \in I_k} x_{ji} = 0 \).

This fact serves as the basis for the direct construction of \( \Sigma\Pi \)-neurons.
3. Construction of ΣΠ-neurons
Here is the constructive learning procedure for the ΣΠ-neuron, which is a slightly modified version of the standard learning procedure. To describe it, we introduce the notation of the multi linear function at \( p \)-th step:

\[
sp^p(x) = \theta + \sum_{k=1}^{p} w_k \prod_{i \in I_k} x_i.
\]

This multi-linear function will be used for representation of ΣΠ-neuron at \( p \)-th step, which is function correctly for all previous training Boolean inputs.

The procedure for constructing a ΣΠ-neuron RCL-SPN (Recurrent Constructive Learning of SPN) from the sequence \( X \), triangularly ordered relative to \( I \), and the corresponding sequence of values \( Y \), has the form:

\[
def RCL-SPN(I, X, Y):
sp^0(x) \leftarrow \theta
\]
for \( k \) in \( 1 \ldots N \):

\[
s \leftarrow sp^{k-1}(x_k)
\]
if \( \text{sgn}(s) = y_k \):  

\[
sp^k(x) \leftarrow sp^{k-1}(x)
\]
else:

\[
w_k \leftarrow \tilde{y}_k - 1 - s
\]

\[
sp^k(x) \leftarrow sp^{k-1}(x) + w_k \prod_{i \in I_k} x_i
\]

\[
\text{return } \text{sgn}(sp^N(x))
\]

It is easy to establish that applying the RCL-SPN procedure to \( X, Y \) we get correctly functioning ΣΠ-neuron \( \text{spn}(x) \), i.e. for all \( k = 1, \ldots, N \): \( y_k = \text{spn}(x_k) \).

4. Construction of a parameterized family of ΣΠ-neurons
The set of all sequences of multi-indices with respect to which the sequence \( X \) is triangularly ordered, we denote \( \mathcal{I}(X) \) and introduce the relation \( \prec \).

**Definition.** \( I' < I'' \), if

- for all \( k = 1, \ldots, N \): \( I_k' \subseteq I_k'' \);
- there is \( k \) such that \( I_k' \subset I_k'' \).

Note that for any sequence \( I \in \mathcal{J}(X) \): \( I < I(X) \) or \( I = I(X) \).

Let \( \mathcal{J}'(X) \) denote the set of minimal elements in \( \mathcal{J}(X) \) with respect to the relation \( \prec \).

**Fact.** If \( \{i_1, \ldots, i_N\} \in \mathcal{J}'(X) \), then for all \( k = 1, \ldots, N \) multi-index \( I_k \in \mathcal{J}(X) \), where \( \mathcal{J}(X) \) is the set of minimal multi-indices \( I \subseteq I(x_k) \) such that

- for all \( i \in I \): \( x_{ki} = 1 \);
- for all \( j < k \): \( x_j \geq_1 x_k \).

For each \( 1 \leq k \leq N \) we define a multi-linear function
\[ P_k(x) = \sum_{i \in J_k(X)} y_i \prod_{i \in 1} x_i, \] 

where \( y_i \geq 0 \) and 

\[ \sum_{i \in J_k(X)} y_i = 1. \] 

**Fact.** \( P_k(x) \) has the following properties:

- for all \( k = 1, \ldots, N: P_k(\bar{x}_k) = 1; \)
- for all \( j < k: P_k(\bar{x}_j) = 0. \)

The set \( J_k(X) \) can be constructed using a procedure from [5].

Define a parameterized multilinear functions:

\[ sp^0(x) = \theta + \sum_{k=1}^p w_k P_k(x), \quad 1 \leq p \leq N. \]

The following algorithm constructs a parameterized form of \( \Sigma \Pi \)-neuron.

```python
def RCL-SPN2(I, X, Y):
    sp^0(x) = \theta
    for k in 1 \ldots N:
        s = sp^{k-1}(x_k)
        if sgn(s) = y_k:
            sp^k(x) = sp^{k-1}(x)
        else:
            w_k = y_k - 1 - s
            sp^k(x) = sp^{k-1}(x) + w_k P_k(x)
    spn(x) = sgn(sp^0(x))
    return spn(x)
```

It is easy to establish that applying the RL-SPN2 procedure to \( X, Y \) we obtain a parameterized \( \Sigma \Pi \)-neuron \( spn(x) \) such that for any set of parameter values and for all \( k = 1, \ldots, N: y_k = spn(x_k) \) under the condition (3) for all \( y_1 \), where \( i \in J_k(X) \).

**5. Illustrative example**

Consider the training set given by the table 1. In addition to the columns \( x_1, \ldots, x_4 \) and \( y \), which specify the values at the inputs and outputs, the column \( J_k \), which defines the minimum allowed multi-indices, and the column \( P_k \), defining the corresponding multi-linear functions (2-3).

| \( x_1 \) | \( x_2 \) | \( x_3 \) | \( x_4 \) | \( y \) | \( J_k \) | \( P_k \) |
|------|------|------|------|------|------|------|
| 1    | 0    | 0    | 0    | 0    | 0    | 1    |
| 2    | 1    | 1    | 0    | 0    | 1    | \{1\}, \{2\} \( ax_4 + (1 - a)x_2 \) |

Table 1. The training set.
The sequence of multi-linear functions $s_p(x)$ that are obtained during the construction process is as follows:

$$s_p(x) = s_p(x) + a x_1 + (1-a)x_2;$$

$$s_p(x) = s_p(x) + b x_3 + (1-b)x_4;$$

$$s_p(x) = s_p(x) + (1-a-b)x_1x_3;$$

$$s_p(x) = s_p(x) + (a+b-1)x_2x_4;$$

$$s_p(x) = s_p(x) + (a+b-1)x_2x_4;$$

$$s_p(x) = s_p(x) + (b-a)x_1x_4,$$

where $(s)_+ = \max\{s,0\}$.

Here $0 \leq a, b \leq 1$. As a result, we obtain a family of multi-linear functions:

$$sp(x; a,b) = -1 + ax_1 + (1-a)x_2 + bx_3 + (1-b)x_4 + (1-a-b)x_1x_3 + (a+b-1)x_2x_4 + (a-b)x_2x_3 + (b-a)x_1x_4.$$ 

For clarity, we give several variants (table 2).

**Table 2.** Variants.

| $a$ | $b$ | $sp(x)$ |
|-----|-----|---------|
| 0   | 0   | $-1 + x_2 + x_4 + x_1x_3$ |
| 0   | 1   | $-1 + x_2 + x_3 + x_1x_4$ |
| 1   | 0   | $-1 + x_1 + x_4 + x_2x_3$ |
| 1   | 1   | $-1 + x_1 + x_3 + x_2x_4$ |
| 1   | 1   | $-1 + x_1 + x_3 + x_2x_4$ |
Since the minimum number of terms corresponds to combinations of Boolean values $a$ and $b$, in the above table we can see such minimal $\Sigma\Pi$-neurons.

It is easy to see that if the parameters $a, b$ are considered as additional control parameters, then the $\Sigma\Pi$-neuron takes the following form:

$$\text{sp}(x, u) = -1 + u_1 x_1 + u_2 x_2 + u_3 x_3 + u_4 x_4 + u_5 x_1 x_3 + u_6 x_2 x_4 + u_7 x_2 x_3 + u_8 x_1 x_4,$$

where

$$u_1 = a, \quad u_2 = 1 - a, \quad u_3 = b, \quad u_4 = 1 - b,$$

$$u_5 = (1 - a - b)^+, \quad u_6 = (a + b - 1)^+, \quad u_7 = (a - b)^+, \quad u_8 = (b - a)^+.$$

As a result, we get a $\Sigma\Pi$-neuron, in which the synaptic parameters are calculated using linear and ReLU control elements. In essence, it can be considered as single $\Sigma\Pi$-neuron, which, in addition to the inputs $x_1, ..., x_4$, also has inputs from an intermediate layer consisting of threshold-linear neurons (1) – (2).

In some cases, it will be easy to adapt the parameterized $\Sigma\Pi$-neuron without retraining on 2 new inputs. For example, let $(1,0,0,0)$ be new input with expected value $y = 1$: $\text{sp}(1,0,0,0; a, b) = -1 + ax_1$. So, we may set $a = 1$ and get new parameterized $\Sigma\Pi$-neuron with

$$\text{sp}(x, b) = -1 + x_1 + bx_3 + (1 - b)x_4 + bx_2 x_4 + (1 - b)x_2 x_3,$$

where $0 \leq b \leq 1$. On next new input $(0,0,1,0)$ with $y = 1$: $\text{sp}(0,0,1,0; b) = -1 + b$. So, we may set $b = 1$ and get

$$\text{sp}(x) = -1 + x_1 + x_3 + x_2 x_4.$$

Thus, parameterized $\Sigma\Pi$-neurons can be adapted to new inputs by setting values to the corresponding free control parameters.

6. Conclusions

Thus, we have built a constructive procedure that allows us to present a parametric representation for a family of $\Sigma\Pi$-neurons that function correctly on the training set. This family is represented by a single $\Sigma\Pi$-neuron with an intermediate layer of linear and threshold linear elements that calculate the weights of any $\Sigma\Pi$-neuron from this family for given values of input control parameters. This is especially useful if the $\Sigma\Pi$-neurons must participate in some sequential information processing process. The parametric representation of the $\Sigma\Pi$-neuron family that functions correctly on the training set allows us to search for optimal combinations of control parameters within the framework of various optimization criteria. A simple illustrative example explains that it can also simplify the retraining process for new inputs.

References

[1] Mel B W 1990 The sigma-pi column: a model of associative learning in cerebral cortex Technical Report CNS Memo 6 Computation and Neural Systems Program California Institute of Technology

[2] Jadi M P, Behabadi B F, Poleg-Polsky A, Schiller J and Mel B W 2014 An Augmented Two-Layer Model Captures Nonlinear Analog Spatial Integration Effects in Pyramidal Neuron Dendrites
Proceedings of the IEEE 102(5) 782-98

[3] Mel B W 2016 Toward a simplified model of an active dendritic tree Dendrites 3(16) (Oxford University Press) p 405-86

[4] Timofeev A V and Pshibikhov V Kh 1974 Learning Algorithms and Minimizing the Complexity of Polynomial Recognizing Systems Izv. AN SSSR Technical cybernetics 7 214-7

[5] Shibzukhov Z M 2006 Constructive learning methods of ΣΠ-neural networks MAIK Nauka 150

[6] Shibzukhov Z M 2014 Correct Aggregation Operations with Algorithms Pattern Recognition and Image Analysis 24(3) 377-82

[7] Lyutikova L A 2018 Sigma-Pi Neural Networks: Error Correction Methods Procedia Computer Science 145 312-8

[8] Potapov A A, Zhang W, Feng T and Rekhviashvili S Sh 2019 Sigma-pi neural unit: switching circuit simulation AIIPCC '19: Proceedings of the International Conference on Artificial Intelligence Information Processing and Cloud Computing Article 21 1-5 https://doi.org/10.1145/3371425.3371477

[9] Delalleau O and Bengio Y 2011 Shallow vs. Deep Sum-Product Networks Advances in Neural Information Processing Systems 24 666-74