Temporal Coding of Neural Stimuli

Adrian Horzyk [1[0000-0001-9001-4198]], Krzysztof Gołdon [1[0000-0003-4540-1812]] and Janusz A. Starzyk [2,3[0000-0003-2678-5515]]

[1] AGH University of Science and Technology, Krakow, Poland
[2] University of Information Technology and Management in Rzeszow, Rzeszow, Poland
[3] School of EECS, Ohio University, Athens, USA
horzyk@agh.edu.pl, krzysztofgoldon@gmail.com, starzykj@gmail.com

Abstract. Contemporary artificial neural networks use various metrics to code input data and usually do not use temporal coding, unlike biological neural systems. Real neural systems operate in time and use the time to code external stimuli of various kinds to produce a uniform internal data representation that can be used for further neural computations. This paper shows how it can be done using special receptors and neurons which use the time to code external data as well as internal results of computations. If neural processes take different time, the activation time of neurons can be used to code the results of computations. Such neurons can automatically find data associated with the given inputs. In this way, we can find the most similar objects represented by the network and use them for recognition or classification tasks. Conducted research and results prove that time space, temporal coding, and temporal neurons can be used instead of data feature space, direct use of input data, and classic artificial neurons. Time and temporal coding might be an important branch for the development of future artificial neural networks inspired by biological neurons.

Keywords: temporal coding, temporal neurons, feature representation in the time space, stimuli receptor transformation into the time space, associative temporal neural networks, associative graph data structure.

1 Introduction

Different types of artificial neural networks either directly use external data as inputs, or use normalization, standardization, various transformations of input data space (e.g. PCA, ICA) [1, 7], or code them through preprocessing operations. Most of the artificial neural networks use discrete time iterations where all or a part of neurons are evaluated and calculate outputs in the same iteration (discrete time step) making computation totally synchronous (e.g. Hebb's rule, Oja's rule, McCulloch-Pitts model, Perceptron) [7, 22]. It can be perceived as positive because some processes in the brain seem to be synchronized (due to brain waves of various frequencies). Nonetheless, the majority of neuron activations are asynchronous [1, 3] and difficult to implement because they require ordering and simulation in time on contemporary computers. Time is used in spiking models of neurons [6, 14, 15, 20] that handle various internal processes in time, and time dependencies influence the stimulation processes in their networks [7, 8, 10, 14].

On the other hand, synchronous processes are more likely used because discrete time iterations can be easily parallelized using GPUs. Modern deep neural network architectures [7, 8] are also updated in discrete time steps. The synchronicity is an important factor to achieve efficient computational models on contemporary computers thanks to the use of vectorization and parallelization of computations of the same operations on multiple data. Nevertheless, we should not ignore the nature where asynchronous processes take place and have a great part in brain processes [1, 19]. Such processes have significance in the creation of new biology inspired strategies and algorithms.

The major difference between asynchronous and synchronous computations is that the synchronously fired neurons lose the possibility to code and differentiate external stimuli and internal results using time because the time is the same for all synchronously triggered neurons (e.g. in one iterative epoch or training cycle). Biological neural systems do not work in this way, and each neuron can be activated independently of other neurons if only it is charged to its activation threshold level [16, 19]. The synchronicity between the biological neurons is secondary and depends on the synchronic stimulations from outside. Asynchronous activations of neurons in continuous time allow neurons to compete and produce time-coded results based on such competitions. The first activated neurons in time identify objects or classes the most associated with the input stimuli, can influence other processes, inhibit competing neurons, and produce memories. In nature, the memories are usually created for the strongest and the most frequent stimuli associated with the earliest activated neurons representing memorized objects or their classes, temporal sequences, or spatial neighborhood of objects.

This paper defines a new model of neural networks which are based on temporal coding, temporal neurons, and special receptors that transform external stimuli (input data) into an internal temporal form (time space). Temporal neurons work asynchronously and must be simulated using a global queue mechanism responsible for ordering processes in time and allowing for competitions between such neurons. In our paper, this mechanism will be described and used in the conducted experiments. Temporal neurons are connected using associative rules that link features and objects in the same way as in the associative graph data structures (AGDS) [12, 13] where each unique value of each attribute is represented by a single node that aggregates the representation of all duplicates of this value in a dataset. Unlike many other approaches which use many-to-many connections between neurons organized in layers or matrices, the presented approach does not use layers but links similar features and connects them to the objects defined by these features. In nature and various biologically inspired models of neural networks, there is a wide variety of connections that only seldom form a multilayer structure where neurons are connected to other consecutive layers [1, 16, 19, 24]. Hence, the major part of the contemporary used artificial neural network models is based on unrealistic and highly restrictive fundamentals. We use these models in computational intelligence because they produce valuable results, but they do not work like real neuronal systems that can exhibit the ability to self-develop appropriate cognitive and intelligent architectures [4, 5, 17, 23]. This paper partially removes the established limitations of contemporary neuronal models and shows new abilities of neural systems that can be used and exploited in future research. These abilities will be achieved using the time space, temporal coding, and time-based processing in the neural network.

2 Temporal Coding and Receptors

Temporal coding means the transformation of values from the input feature space into the time space, i.e. to the appropriate charging periods of the neurons connected to receptors. Each receptor is most sensitive to the given feature value v_k^n and less sensitive to other close feature values. The sensitiveness of the *n*-th receptor of the *k*-th attribute and simultaneously its strength with which it stimulates the connected neuron can be expressed by the following condition:

$$s_k^n = 1 - \frac{|v_k^n - v_k|}{R_k} \tag{1}$$

where $R_k = v_k^{max} - v_k^{min}$ is a range of values of the attribute *k*. Each receptor is connected to a neuron that is charged by this receptor for the period p_n according to the similarity of the input stimulus v_k to the value v_k^n represented by this receptor:

$$p_n = 1 - s_k^n = \frac{|v_k^n - v_k|}{R_k} \tag{2}$$

This provides the firing of neurons at a different time according to the presented input value. Using the continuous time of activations, input stimuli can be precisely coded in time space, i.e. without rounding. Thus, input data (external stimuli) are transformed into different charging periods and activation times of the connected neurons.

Fig. 1 illustrates possible stimulation flow as a reaction to the input value v_k sensed by the receptors that charge the V_k^n and V_k^{n+1} neurons, their neighbors V_k^{n-1} and V_k^{n+2}, the connected temporal object neurons O^{j_2} and O^{j_4} of the strongest stimulated temporal value neuron V_k^n, the subsequent class neuron C^{l_1}, and finally, the stop neuron *S* which activation stops the stimulation process of the neural network for the presented input.

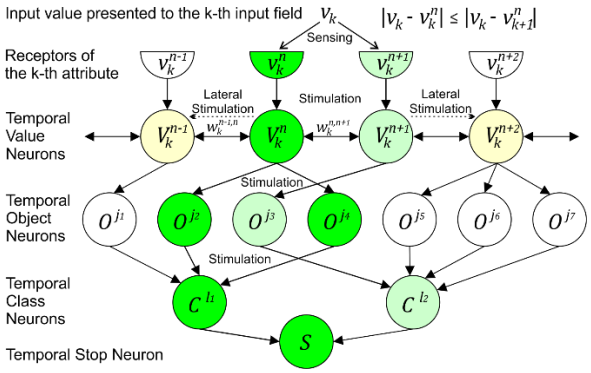

Fig. 1. Stimulation schema of temporal processes in the described neural network. The temporal process is started by the input stimulation of value v_k that is sensed by the receptors representing the closest values with the different strengths (1) that influence charging periods (2) of connected temporal value neurons which start stimulation of connected neighboring temporal value neurons (through lateral connections) and temporal object neurons when activated. Activated temporal object neurons subsequently activate temporal class neurons, and finally, the temporal stop neuron.

Always one or two receptors react to the input value v_k, computing the charging periods p_n of the connected neurons. If the input value v_k exactly equals to the value v_k^n represented by one of the receptors, then only a single receptor reacts to such an input stimulus and the activation period p_n of the connected neuron V_k^n is equal to 0 according to (2), i.e. it activates immediately after receiving the input stimulus. If the input v_k differs from all values v_k^n represented by the receptors of the attribute *k*, then two receptors representing the closest smaller and the closest bigger values react to such an input stimulus (except when the value v_k is minimal or maximal). In this case, the activation periods p_n of two connected neurons are computed after (2), and those periods are greater than 0.

3 Associative Temporal Neurons

Associative Temporal Neuron (ATN) is a new proposed model of neurons that differentiate charging periods according to the stimulation strengths and create associative connections representing various relations between input features and objects. ATN neurons work in time, but their way of working is different from classic spiking neuron models [6, 14, 15, 20]. The different charging periods determine the activation moments of ATN neurons and the charging processes of other connected neurons in the network. The order of the activations of neurons is used to produce and interpret results. The earliest activated neurons represent the most strongly associated objects to the input stimuli, where the associations can represent similarity, the order in sequence or spatial neighborhood. Thus, the earliest activated neurons can point out the most similar objects represented by the neural network. In conscious thinking, we usually take into account the first thoughts in response to the asked question or external stimuli. The ATN model also defines a minimum number of charge stimuli for each neuron, which allows it to be activated. This number is defined as a number of features which define the object represented by the temporal object neuron. It is equal to one for all value neurons representing simple one-value input features. This number increases the confidence of the classification when the neuron represents a class. Summarizing, ATN neuron *n* can be activated when two following conditions are simultaneously true:

1. The number of input stimuli s_n is equal to or greater than a threshold number θ_n of the required stimuli.
2. Current simulation time *T* of the network for the current run operation can be established from the *f* function value (4) of the sum S_n (3) of the g function values of the earliest s_n charging periods $p_1, \ldots, p_{s_n}$, where functions *f* and g must be positive:

$$S_n = \sum_{i=1}^{s_n} g(p_i) \tag{3}$$

$$p_n = f(S_n) \tag{4}$$

$$p_i = T_i - T_o \tag{5}$$

where p_i is a period (5) computed as a difference between the activation time T_i of the neuron sending the stimulus and the starting time T_o of the currently running operation in the network, where $p_1 \leq \cdots \leq p_{s_n}$. Each ATN neuron counts incoming input stimuli

(s_n) and sums the charging periods transformed by the function g according to (3). When this count achieves the threshold number θ_n, then the function f is evaluated, and the final activation period p_n of the charged neuron n is calculated (4). This means that after the period p_n of charging elapses, the neuron will be activated, its activation time T_n during simulation will be established according to this period and the simulation time T_o when the charging operation was started:

$$T_n = T_o + p_n \tag{6}$$

The activation time T_n together with the information about the neuron is added to the global event queue (GEQ) that sorts events in the incremental order to control the sequence of updating neurons. The GEQ is responsible for activating the neurons in the appropriate time (6) allowing for their competitions.

The activated ATN neuron switches to the refraction period that prevents cyclic retrograde stimulations of mutually stimulating neurons. During the refraction period, ATN neurons are insensitive to any input stimuli. After the refraction period, the neuron returns to its resting (initial) state waiting for the next stimuli. To switch the neuron from its refraction state to its resting state, its parameters must be initialized. To perform this efficiently, each ATN neuron stores a special counter of the network operations (OPCntr). It points to the number of operation during which it has been updated the last time. The OPCntr counter is also stored in the network and incremented by one each time when the network is stimulated by new inputs (e.g. starts a new classification process). If the OPCntr counter of the neuron is less than the OPCntr counter of the network, the neuron variables are outdated and must be updated (e.g. initialized) before the next operation on this neuron. After each operation, the neuron OPCntr is updated to be equal to the network OPCntr of to avoid the repetitive updating (e.g. initialization) before the subsequent updates of this neuron during the currently running operation. Owing to this mechanism, neurons are initialized exactly once before they are used in the next operation. This mechanism avoids the necessity to initialize variables of all neurons of the neural network before starting a new operation, which saves a lot of time.

When the neuron is activated, it sends stimuli to the connected neurons. If the connected neuron represents another numerical value v_k^m of the same attribute k, then the charging period p_m of this neuron is calculated from:

$$p_m = 1 - w_k^{m,n} \tag{7}$$

where $w_k^{m,n}$ is a connection weight between neurons V_k^n and V_k^m representing neighbor values v_k^n and v_k^m. It is defined as the absolute difference between these values v_k^n and v_k^m normalized by the range R_k of values of the k-th attribute:

$$w_k^{m,n} = 1 - \frac{|v_k^m - v_k^n|}{R_k} \tag{8}$$

The weights between value neurons V_k^m and V_k^n are symmetrical, i.e. $w_k^{n,m} = w_k^{m,n}$. Value neurons representing symbolic data are not connected and weighted.

In summary, each temporal value neurons V_k^n representing an input feature v_k^n is connected to at least one object neuron that represents a training sample defined by a set of input features. The activated neuron representing a feature v_k^n stimulates all connected object neurons. The stimulus contains the time when this neuron has been activated. Object neurons count up the incoming stimuli and compute the sums as defined by (3).

4 Associative Temporal Neural Networks

Associative Temporal Neural Networks (ATNN) are built from associative temporal neurons (ATN) and special receptors which transform input data into the time space and implement temporal processes between associatively connected neurons. ATN neurons connect to reproduce relations between elements represented by these neurons, where elements can be single-value features or various objects defined by features or other objects. Some objects can define classes or clusters or have a special meaning in the network. Associative connections between neurons are created for the known relationships which can be deduced from the input (training) data. All unique single-value data (like numbers, symbols, or strings) are represented by separate neurons connected to the receptors which are sensitive for these values. Neurons representing orderable values are additionally connected to the neurons representing neighbor values, and the connections are weighted (8). The associative representation means that all duplicates of the features of each attribute separately are aggregated and represented by the same neurons [9, 13]. Therefore, objects defined by a subset of the same features are indirectly connected by a subset value neurons representing these features. Objects defined by similar (not the same) features are also indirectly connected through connected value neurons representing similar features. Such associative organization of the network automatically groups objects according to their similarities and facilitates the inference processes [11]. For classic training data used for classification tasks composed from objects defined by a given number of features, the created network of elements can be described by the associative graph data structure (AGDS) [12, 13] that is used as a backbone structure of a proposed ATNN structure. Both these structures can be further accelerated using AVB+trees [13], which allow typically for less than the logarithmic time of the search of appropriate receptors for the input data. Any training dataset can be easily transformed into this graph structure as described in [12, 13].

Assume that we have a dataset $\mathbb{S} = \{P^1, \dots, P^N\}$ consisting of N samples $P^n = [v_1^n, \dots, v_k^n, \dots, v_K^n]$ where v_k^n is a k-th attribute value defining the n-th sample (object) P^n, and K is the number of attributes. Each unique sample P^n is represented in the ATNN network by the **temporal object neuron** O^j where the number of object neurons J is less or equal to the number of training samples ($J \leq N$) because of possible duplicates of samples in the dataset $\mathbb{S}$ and their aggregated representation in the ATNN network.

The training samples represent L classes $\{c^1, \dots, c^L\}$ represented by L **temporal class neurons** $\{C^1, \dots, C^L\}$, and each training sample can belong to one or more classes dependently on the considered classification type (single-label classification or multi-label classification) [11, 21, 25]. In this paper, we consider only single-label classification problems to compare ATNN networks to KNN classifiers. Each unique k-th attribute value (feature) v_k^n is represented by a so-called **temporal value neuron** V_k^m in the ATNN network. Because the same feature can define many objects in the training data set, the number M_k of the temporal value neurons $\{V_k^1, \dots, V_k^{M_k}\}$ representing the k-th attribute unique values is typically smaller than the number N of all training samples, i.e. $M_k \leq N$ (usually $M_k \ll N$ for real datasets due to the big number of duplicated attribute values defining various objects).

Temporal value neurons are also connected to object neurons, which represent training samples defined by the values represented by these connected value neurons. Connection weights between value and object neurons are equal to one. A group of object neurons can also define classes represented by temporal class neurons. The connections between object and class neurons are also equal to one. The activation threshold number of stimuli of temporal class neurons is set to one, so they immediately react to activation of any object neuron that defines a sample of the given class.

The ATNN network can be used as a k nearest neighbor (KNN) classifier, where we will also use a single **temporal stop neuron** that is connected to all class neurons. This neuron is responsible for stopping the network calculations when the demanded goal of computations (e.g. recognition, classification, or sorting) is finished. We defined the threshold of this neuron accordingly to the task (e.g. 1 for recognition, N for sorting) where this network is used. The weights between class and stop neurons are also equal to one, reacting to all classified samples immediately, but the threshold of this neuron can be set to the required number of searched samples. The activated ATN neurons of all kinds will stimulate other connected neurons, and each stimulus includes information about the time T_n (6) of the activation to compute appropriate periods (4) for computing sums (3) and charging periods (5) for the connected neurons.

5 Applying ATNN for Ordering Objects

The main idea of ATNN networks is to allow neurons to be activated over time in a succession that comes from the associative (relationship) strengths, e.g. similarity, between input data and the object. Therefore, the object neurons representing objects that are similar to input data (external stimuli) should be activated faster than neurons representing less similar objects. In fact, the ATNN can quickly order all represented objects (training samples) according to their similarities to the input data (Fig. 2), starting from the most similar one(s) (activated earlier, darker red in Fig. 2) and finishing with the least similar one(s) (activated later, lighter red in Fig. 2). Here, the similarities between represented objects by object neurons and given inputs will be defined by the Euclidean measure in the time space of charging periods of temporal value neurons representing transformed features into the charging periods.

Associative Distance Sorting Algorithm using an ATNN network:

```
Input: T: training data, x: classified sample,
Output: sortedObjects: objects sorted in descending order

OrderingObjects(T, x)
   atnn = CreateATNN(T)
   GEQ.InitializeSimulationTime()
   foreach attribute of atnn do
      attribute.FindAndStartStimulate(x[numberOfAttribute]) // using AVB+trees
   while (stopNeuron.State < threshold)
      currentEvent = GEQ.RemoveFirst
      simulationTime=currentEvent.UpdateTime
      currentEvent.Simulate() // add activated object neurons to sortedObjects
   return sortedObjects
```

In the first step of this algorithm, the structure of the ATNN network based on AGDS [13] is created. Values of each attribute are represented via an AVB+tree [13] that supports efficient access, and all operations are processed in at most logarithmic time. The stop condition of this algorithm is provided by the temporal stop neuron which is connected to all object neurons and which threshold is equal to the number of all pattern neurons to stop stimulation when all object neurons are activated. To aggregate representation of all duplicates during the construction of the ATNN structure, a special insertion operation using an AVB+tree [13] is used to increase counter if the inserted value already exists in this structure or to insert it and rebalance the tree (if necessary) when the inserted value is new and must be added:

```
SearchOrInsertNeuron(value)
   foreach neuron in node
      if (neuron.Value == value)
         neuron.IncrementCounterOfDuplicates() // aggregate their representation
         return true
   if (childList.Count==0) // this is a leaf of AVB+tree
      CreateNewNeuronAndInsertItIntoAVBTreeNode() // rebalance a tree if necessary
   else
      if (Neurons.First.Value < value)
         return Neurons.First.SearchOrInserNeuron(value)
      elseif (Neurons.Last.Value > value)
         return Neurons.Last.SearchOrInserNeuron(value)
      else return Neurons.Middle.SearchOrInsertNeuron(value)
```

In the second step, the ATNN is stimulated by the sample as long as the stop neuron is activated. During this process, a list of activated object neurons is created in the order coming from the activation moments of these neurons. Thus, the samples (represented by object neurons) are ordered according to the similarity (e.g. defined by the Euclidean distance) to the sample presented on the input of the network. To start the stimulation process of the ATNN network quickly, the previously created AVB+trees for all attributes are used to search for temporal value neurons which will be stimulated appropriately to the presented input values. Subsequently, temporal value neurons compute their activation moments and put them to the GEQ that handles their ordering and triggering in the right simulation time. In each simulation step, always the first event from the GEQ is run until the temporal stop neuron is activated. During the simulation process, all activated neurons stimulate connected neurons which insert new events to the GEQ according to the computed activation periods as described in the previous sections.

The similarity to input data in Fig. 2 is represented by the activation times (marked with *t* in the lowest raw of circles of neurons that represents training samples) – the smaller times are, the more similar the training samples are. Such ordering is used by various methods, e.g. KNN classifiers, which base their classification decisions on the given number of nearest neighbors, various sorting routines, recognition, clustering, and classification algorithms. This ordering of ATNN networks is also quite fast because the algorithm needs to stimulate the neurons only once for a given input (training) dataset to get results.

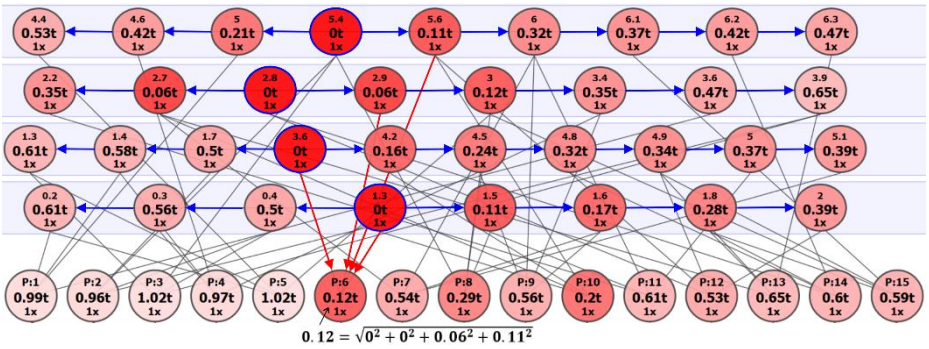

Fig. 2. Example of sorting objects by the ATNN according to the similarity to the given input sample [5.4, 2.8, 3.6, 1.3] to 15 Iris samples [26]. The temporal value neurons representing features of four attributes are stimulated laterally by the previously activated value neurons as marked by blue arrows. The sorted training samples are represented by temporal object neurons in the last row which compute their activation times using an Euclidean distance of activation times of the stimulating value neurons as presented on a sample object neuron activated in time 0.12. Each neuron presents the value or ID in the upper part, the activation time in the middle, and the number of its activations in the bottom part of its circle. The presented activation times of object neurons reflect the similarity of the input sample to all training samples from this dataset. The similarities expressed by the activation times produced the following sequence of training samples: P:6, P:10, P:8, P:12, P:7, P:9, P:15, P:14, P:11, P:13, P:2, P:4, P:1, P:3, and P:5.

The computational complexity of ordering all objects according to their similarities to the given input sample is linear $O(N + \sum_{k=1}^{K} h_k(N)) \leq O\big((K+1) \cdot N\big)$, where N is the number of objects, and K is the number of attributes, and the function h_k defines the number of unique values for the k-th attribute. If we do the same operation on a tabular data structure, we need to compute similarity factors for each object and sort final results according to these factors. This operation would cost log-linear time $O(K \cdot N \log N)$ for quicksort, heapsort, or merge sort algorithms [2] or $O(K \cdot N)$ for counting sort or radix sort [2]. Thus ATNN networks based on the AGDS structure can theoretically have the same or better complexity than the best classic sorting algorithms based on a tabular structure especially when the sorted data consist of many duplicated values defining objects in the dataset. Unfortunately, the simulation of temporal neurons on a sequential machine (which are most common today) takes extra time because events (representing activation times of neurons) must be ordered, so the presented sorting on ATNN is not always faster than classic approaches, however, on asynchronously parallel machines, it would run much faster. Therefore, the presented sorting approach using ATNN networks can only explain how biological neurons can deal with ordering tasks when working on a different computational platform than contemporary computers.

The ATNN sorts objects starting from the most similar, so the efficiency of ATNN will be better when searching for a limited subset of the most similar objects because the classic approaches usually require to compute all similarities and sort all objects before selecting the most similar ones. The results presented in Table 1 show the sorting times of the most similar objects to the given input sample computed using the classic

quicksort approach on data stored in tables compared to the sorting made by the simulated ATNN networks on a sequential machine.

Table 1. Comparisons of the recognition speed of the classic and ATNN sorting routines.

Training Datasets*	No of Instances	No of Attributes	Unique Features	Sorting Time [timer ticks]	
				Classic	ATNN
Immunotherapy	90	7	27,46%	5456	487
Iris	150	4	20,50%	21825	596
Wine	178	12	59,74%	35466	2565
Banknote	1372	4	91,40%	43651	16136
Wine Quality Red	1599	11	8,26%	35466	10924
Eye	14980	14	2,58%	504719	428724
HTRU2	17898	8	52,81%	373765	1150166
Telescope	19020	10	77,34%	428329	1591250
Credit Card	30000	23	25,30%	2515411	2272482
Shuttle	43500	9	0,26%	774812	2600721
Drive	58509	48	59,75%	18036210	27835709
Skin	245057	3	0,10%	1355921	556790

* Datasets were taken from UCI ML Repository [26]

In most of the cases, ATNN network is faster than quicksort, but the major goal of this paper is to prove that data can be transformed into the uniform time space and temporal spikes, not a better processing speed on the contemporary computer. Thus a higher efficiency of our approach is an extra bonus point.

6 Applying ATNN for Recognition Tasks

The first studies in comparisons of classic implementation of KNN classifiers based on the associative graph data structures (AGDS) were described in [12]. The results demonstrated that the associative organization of data with additional relations between values of the same attribute could greatly accelerate KNN classifiers. In this work, we show that KNN classifiers can also be modeled by ATNN networks and that the classification process can be fully automatic.

If $g(p_i) = p_i$ and $f(S_j) = S_j$ are defined as the identity functions, then the computed activation periods reflects the normalized Manhattan distance between input data and the features defining an object represented by an object neuron:

$$P_n^k = \sum_{i=1}^{n_j} p_i = \sum_{i=1}^{n_j} \frac{|v_k^m - v_k^n|}{R_k} \tag{9}$$

If $g(p_i) = {p_i}^2$ and $f(S_j) = \sqrt{S_j}$, then the computed activation time reflects the Euclidean distance between input data and the features defining an object represented by an object neuron:

$$P_n^k = \sqrt{\sum_{i=1}^{n_j} {p_i}^2} = \sqrt{\sum_{i=1}^{n_j} \left(\frac{|v_k^m - v_k^n|}{R_k}\right)^2} \tag{10}$$

Hence, ATN neurons can be used to compute the nearest neighbors for KNN classifiers because the Manhattan or Euclidean distances of training samples to the classified sample can be transformed into the activation periods of object neurons representing these samples. The charging periods linearly change as these distances, so they model original training data space in the internal time space of ATNN networks. Therefore, the first k activated ATN temporal object neurons represent the same k nearest neighbors (objects) as computed by the KNN classifier.

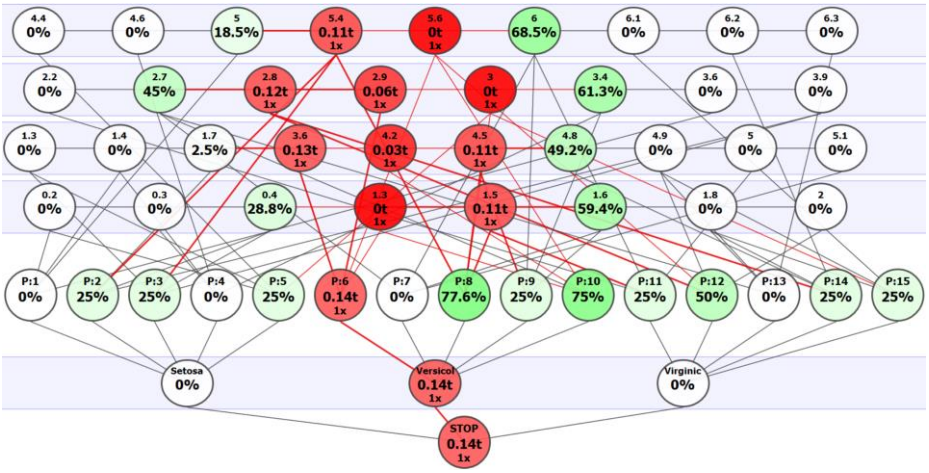

Fig. 3. Example of recognition (for k=1) of the most similar object (here P:6) to the presented inputs [5.6, 3.0, 4.1, 1.3] made by the ATNN network created for the training data consisting of 15 Iris samples [26]. The temporal value neurons for each attribute separately are presented in the four top rows. In the fifth row, there are temporal object neurons defined by the appropriate combinations of temporal value neurons (representing training samples). The sixth row consists of temporal class neurons. The temporal STOP neuron in the last row stops the simulation when the most similar object is found. Neurons painted in red symbolized the activated neurons at different times. Only activated neurons stimulate other connected neurons which do not always achieve activations (neurons painted in green). White colored neurons do not take part in the presented recognition process, which saves time in comparison to the KNN classification process, where all training samples take part in the classification process.

When k=1 the classification process points out the most similar object to the presented sample on the input, so this process can also be viewed as a recognition process as shown in Fig. 3. The comparisons of the speed of the classic KNN classifier with k=1 and ATNN used in recognition tasks are presented in Table 2 where ATNN networks win in recognition speed.

Table 2. Comparisons of the recognition speed of the classic and ATNN approaches.

Training Datasets*	No of Instances	No of Attributes	Unique Features	Recognition [timer ticks]	
				Classic	ATNN (k=1)
Immunotherapy	90	7	27,46%	301	38
Iris	150	4	20,50%	282	25
Wine	178	12	59,74%	309	84
Banknote	1372	4	91,40%	476	33
Wine Quality Red	1599	11	8,26%	550	81
Eye	14980	14	2,58%	2814	363
HTRU2	17898	8	52,81%	2863	84
Telescope	19020	10	77,34%	3233	85
Credit Card	30000	23	25,30%	7319	7472
Shuttle	43500	9	0,26%	5629	4803
Drive	58509	48	59,75%	25948	1429
Skin	245057	3	0,10%	19741	79

* Datasets were taken from UCI ML Repository [26]

In most of the cases, the proposed approach was more efficient than the classical one. The results were obtained on a sequential machine and were from 2% slower in case of Credit Card data set to 250 times faster in case of Skin data set than the classic KNN approach. Notice, that our temporal coding network would be much faster if implemented on a network of neurons, since neurons pass information to its neighbors concurrently, while classic algorithm would have no such additional efficiency increase since finding the nearest neighbor would require sequential comparisons. However, once again, the high efficiency of our approach was its extra bonus. We aimed to demonstrate that both ordering and recognition of the input data was possible using temporal coding of input stimuli and associative temporal neurons that may represent sensory values, objects, and classes. A network made of such associative temporal neurons may be a useful addition to neural network structures that work with numeric as well as symbolic data. Such networks may be good to obtain grounded semantic memories needed for efficient learning and motor control functions in robots.

7 Conclusions and Remarks

This paper presented temporal coding of the input data using associative temporal neurons and the resulting associative structures named associative temporal neural networks. The contribution of this paper was to show that it is possible to code features of various attributes using time, namely, various periods of charging and activations of neurons. The temporal coding also requires an appropriate associative neural structure which reproduces relations between features and objects. That is why we used the associative graph data structure AGDS to reproduce the structure of features and samples by value neurons and object neurons which were appropriately connected, and the connections weighted to emphasize the relationships between them. On this basis, neurons and their connections were set up to be charged in different periods according to the similarity of the features and objects to the presented input samples. As a result of such

coding, temporal neurons were activated over time after different periods, which reproduced the Euclidean or Manhattan distance between training samples and the input samples. Using such a network, we could define a sorting routine which allowed to designate the most similar samples and sort them due to the computed similarities. The computed similarities were used to designate k nearest neighbors, order samples, or recognize the most similar sample. The created networks can be used for classification, sorting, and recognition tasks. Thanks to the use of the associative structure of the created associative temporal neural networks, not all objects need to be compared in ATNN networks, so we can usually compute results faster, especially when running them in parallel. In spite of the experimentally proven efficiency, the major goal of this paper was to show how original data space can be transformed into the time space. Experiments proved that the time-based computations using associative temporal neurons are possible, and classic coding of input data can be replaced by the presented temporal one using the time space instead of the original one.

This work was supported by the grant from the National Science Centre, Poland DEC-2016/21/B/ST7/02220 and AGH 11.11.120.612.

References

1. Carpenter, G.A., Grossberg, S.: Adaptive resonance theory. In The Handbook of Brain Theory and Neural Networks, M. Arbib (Ed.), MIT Press, Cambridge, MA, pp. 87–90, 2003.
2. Cormen, T., Leiserson, Ch., Rivest, R., Stein, C.: Introduction to Algorithms. 3nd ed., MIT Press and McGraw-Hill, pp. 484–504 (2009).
3. Deuker, L. et al.: Memory Consolidation by Replay of Stimulus-Specific Neural Activity. Jour. of Neuroscience, Vol. 33, No. 49, pp. 19373–19383 (2013).
4. Duch, W.: Brain-inspired conscious computing architecture, Journal of Mind and Behaviour, Vol. 26, pp. 1–22 (2005).
5. Franklin, S., Madl, T., D'Mello, S., Snaider, J.: LIDA: A Systems-level Architecture for Cognition, Emotion, and Learning. IEEE Trans. on Autonomous Mental Development, Vol. 6, No. 1, pp. 19–41 (2014).
6. Gerstner, W., Kistler, W.: Spiking Neuron Models. Cambridge University Press (2002).
7. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press (2016).
8. Graupe, D.: Deep Learning Neural Networks. World Scientific (2016).
9. Horzyk, A.: Neurons Can Sort Data Efficiently. In: Rutkowski L., Korytkowski M., Scherer R., Tadeusiewicz R., Zadeh L., Zurada J. (eds), Artificial Intelligence and Soft Computing, Springer-Verlag, LNCS, Vol. 10245, 64–74, DOI: 10.1007/978-3-319-59063-9_6 (2017).
10. Horzyk, A.: Deep Associative Semantic Neural Graphs for Knowledge Representation and Fast Data Exploration. In: Proc. of KEOD 2017, Scitepress Digital Lib., pp. 67–79 (2017).
11. Horzyk, A., Starzyk, J.A.: Fast Neural Network Adaptation with Associative Pulsing Neurons. In: 2017 IEEE Symposium Series on Computational Intelligence, IEEE Xplore, pp. 339–346 (2017).
12. Horzyk, A., Gołdon, K.: Associative Graph Data Structures Used for Acceleration of K Nearest Neighbor Classifiers, In: 27th International Conference on Artificial Neural Networks (ICANN 2018), Springer-Verlag, LNCS 11139, pp. 648–658 (2018).
13. Horzyk, A.: Associative Graph Data Structures with an Efficient Access via AVB+trees. In: 2018 11th International Conference on Human System Interaction (HSI), IEEE Xplore, pp. 169–175 (2018).

14. Izhikevich, E.M.: Neural excitability, spiking, and bursting, Int. J. Bifurcat. Chaos, 10:1171–1266 (2000).
15. Izhikevich, E.M.: Simple Model of Spiking Neurons. IEEE Transactions on Neural Networks, Vol. 14, No. 6, pp. 1569–1572 (2003).
16. Kalat, J.W.: Biological grounds of psychology. 10th ed., Wadsworth Publishing (2008).
17. Laird, J.E.: Extending the Soar Cognitive Architecture. In Proc. of the First Conference on AGI, Memphis, Tenn, pp. 224–235 (2008).
18. Larose, D.T.: Discovering knowledge from data. Introduction to Data Mining. PWN, Warsaw (2006).
19. Longstaff, A.: Neurobiology. PWN, Warsaw (2006).
20. Maass, W.: Networks of spiking neurons: The third generation of neural network models. Neural Networks, Vol. 10, Issue 9, pp. 1659–1671, Elsevier (1997).
21. Read, J., Pfahringer, B., Holmes, G., Frank, E.: Classifier Chains for Multi-label Classification. Machine Learning Journal, Vol. 85(3), Springer (2011).
22. Rutkowski, L.: Techniques and Methods of Artificial Intelligence. PWN, Warsaw (2012).
23. Tadeusiewicz, R.: Introduction to intelligent systems, Fault Diagnosis. Models, Artificial Intelligence, Applications, CRC Press, Boca Raton, FL (2011).
24. Tyukin, I., Gorban, A.N., Calvo, C., Makarova, J., Makarov, V.A.: High-Dimensional Brain: A Tool for Encoding and Rapid Learning of Memories by Single Neurons. Bulletin of Mathematical Biology, Special Issue: Modelling Biological Evolution: Developing Novel Approaches, pp. 1-33, https://doi.org/10.1007/s11538-018-0415-5, Springer US (2018).
25. Zhang, M.L., Zhou, Z.H.: Multi-label neural networks with applications to functional genomics and text categorization. IEEE Transactions on Knowledge and Data Engineering, Vol. 18, pp. 1338–1351 (2006).
26. UCI ML Repository, http://archive.ics.uci.edu/ml/datasets/Iris, last accessed 2018/04/14.