Kanban Cell Neuron Network

Stock Trading System (KCNSTS)

Colin James III
Ersatz Systems Machine Cognition, LLC
3925 Elisa Ct, Colorado Springs CO 80904 USA
info@cec-services.com +011 (719) 210.9534

***Abstract*—To predict stock trading signals, a novel system for machine cognition includes Kanban cells (KC), Kanban cell neurons (KCN), and Kanban cell neuron networks (KCNN), patent pending. The KC is an asynchronous AND-OR gate without feedback that is self-timing by the input data to process until input is equal to output. For the KCN, the input example is a four-valued logic (4VL) based on the 2-tuple as the four logical values in the set of {"", 01, 10, 11} of four-valued bit code (4vbc) where "" is equivalent to 00. The unique algorithm of multiple KCNs in the KCNN emulates the human neuron with nine logical inputs and one output. The KCNN model in parallel is scalable for large data sets and is adaptable as a forward-looking rules-engine as based on bivalent trial and error. The real-time algorithm is implemented by a look up table (LUT). In software the LUT occupies 64 KB, and on a desktop processes 1MM KCNs per second. Access to a sparsely filled look up table (LUT) is minimized in hardware with a 2-bit value per logical signal. In hardware the LUT occupies 194 KB, and on a $40 device processes at 1.8 BB KCNs per second, or about 1600 times faster. The immediate application of KCNN is analytics for time series of econometrics as the Kanban Cell Neuron Stock Trading System (KCNSTS). Two virtual examples are given for the prediction of trading signals. For 126-trading days, 24 Asian electronic traded funds (ETF) produced an annualized 6.5% return on 70 no load trades. For 49-trading days, one OTC stock produced an annualized 67% return on 10 no load trades.**

Keywords*—*analytics; AND-OR gate; ETF; Kanban cell neuron; multi-valued logic; OTC; stock trading signals

I. Historical background

Originally the Kanban cell (KC) was the production part of a linear pull system used to minimize the change of parts inventory for the Just in Time (JIT) assembly of automobiles. The KC was used by Toyota in about 1964.

Fig. 1 shows the KC in the Petri net, a bipartite directed graph, and with or without a failure or idle subnet as an abstraction of the generalized stochastic Petri net (GSPN) of flexible manufacturing systems (FMS), which are push production systems (for example, using pallets to load incomplete parts and to unload completed parts by continuous transportation as by conveyer or automatic guided vehicle (AGV)).

The system in Fig. 1 is a Petri net of a KC [4] as abstracted for an accounting arithmetic system [3]. Step 104 is a transition. Step 101 is the input and output place. Steps 105 and 106 are feedback paths of the feedback loops of the paths 103 to 104 and 102 to 104. In this context, feedback paths serve as decision branches in the logic of the KC, and are commonly referred to in their totality as feedback loops. (Steps marked as m1, m2, m6, and t2 are true to the original labels and equivalent to the respective Steps 101, 102, 103, and 104.)

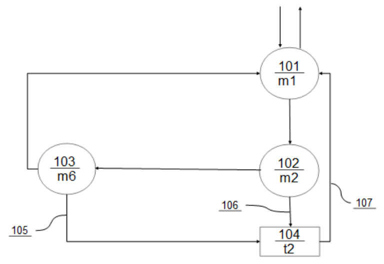

Fig. 1. A Kanban cell in a Petri net

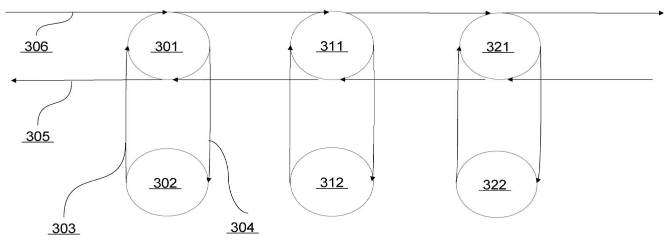

Fig. 2. Synchronous, self-timing neural network as feedback loops

Fig. 2 shows a network that is a synchronous, self-timing neural network as a series of feedback loops. The network consists of data places, through which data flows as data places 301, 311, and 321, and of timing places as timing places 302, 312, and 322, which stimulate the data places as 301, 311, and 321. The direction the data flows is bidirectional as in paths 305 and 306. The direction of timing paths is bidirectional as in paths 303 and 304. The timing places 302, 312, and 322 effectively open and close the data places 301, 311, and 321 to control when waiting data is allowed to flow. The timing places 302, 312, and 322 may be either physical clock cycles or logical looping structures, the duration for which constitutes a delay in that network.

Neural components can be represented as vector spaces such as an adaptive linear neuron or adaptive linear element (Adaline) composed of weight and bias vectors with a summation function (OR circuit), and also a multi-layered Adaline (Madaline) where two such units emulate a logical XOR function (exclusive OR circuit). Such components are examples of probabilistic methods to mimic the neuron.

A perceptron can be represented as a binary classifier using a dot product to convert a real-valued vector to a single binary value and serve as a probabilistic method to mimic a neuron.

A spike neuron or spiking neuron can be represented as a conductive network based on temporal or time bias and differential equations or a calculus which serves also as a probabilistic method to mimic a neuron.

The first deficiency with these neural networks is that as they are based on a vector space, so a solution is ultimately not bivalent, is probabilistic, and hence is undecidable. (Bivalency is not a vector space [5].)

The second deficiency with these networks is that an exclusive OR (XOR) function is sometimes mistakenly developed in them to mimic a neuron. The logical XOR connective is orthogonal or effectively perpendicular as a mathematical operator. However, biological bodies are not rectilinear, but rather based on a phi or Phi ratio of (1 ± (5 ^ 0.5)) / 2. This means that there are no right angles (90-degree arcs) in biology per se. While the logical XOR connective may be constructed from the NOR or NAND logical connectives, there is no evidence that the XOR function is built into the neuron, or necessarily should be.

The third deficiency with these networks is that the perceptron and spike neuron can accept any input without discrimination.

The Kanban cell (KC), Kanban cell neuron (KCN), and Kanban cell neuron network (KCNN) are improvements on the above background components. For example, the KC processes three input values and one output value. The values are only in a multi-valued logic (MVL) such as the 2-tuple set of { "00", "01", "10", "11"}. This is a four-valued logic (4VL) of four-valued bit code (4vbc). It also includes a four-valued logic with null (4VLN) and four-valued bit code with null (4VBCN).

The KC is a logic gate using solely the OR and AND logical connectives, that is, with only the arithmetical operations of addition and multiplication. The KC performs self-timing and terminates processing when the first of the three input values equals the output value. The KCN maps three such KCs into one output, that is, nine inputs as dendrites into one output as axon. The KCNN consists of KCNs in sequential and/or parallel operation.

The purpose of the KCNN is: to process input and output more quickly than the ion transfers; not to require synchronous, repetitive feedback or timing paths; and to use the attributes of the input and output data to self-time itself internally. The KCNN goal is faster asynchronous machine cognition of itself. A benefit of KCNN is also to map neurons into a non-vector space that is not probabilistic and hence to ensure bivalency and decidability.

II. Introduction

The system described below is a massive forest or collection of trees with branches or bundles of leaves or nodes. A node is the Kanban cell (KC). A bundle is a Kanban cell neuron (KCN). A tree is cascaded KCNs, each with a Kanban cell level (KCL). A forest is a Kanban cell neuron network (KCNN, KCN2, or KCN2, and pronounced KCN-Two or KCN-Squared). The KC is effectively a sieve that processes signals in a highly selective way. The KC is the atomic and logical mechanism of the system. Three KCs make up a KCN. Hence the KCN is a radix-3 tree. Multiple KCs as nodes at the same level form a KCL to better describe the KCN to which the KCs belong. The KCNN contains nine KCNs to produce one KCN result. Hence the KCNN is also a radix-9 tree.

The KCN maps the biological mechanism of the human neuron as a series of 9-dendrites that process input signals concurrently into 1-axon, the path of the output signal. The input signals are equivalent to the logical values of null, contradiction, true, false, and tautology. These are respectively assigned as a 2-tuple of the set of {"", "00", "01", "11"} or as the set of {"", 0, 1, 2, 3}, depending on which set of valid results is required.

As a sieve, the KCN filters three input signals into one output signal. The number of all input signals to produce one valid output signals as a result is in the ratio of about 15 to 1.

The computational or machine mechanism to accomplish this is basically the same for software or hardware implementation. The core concept is that LUTs produce output results at a faster rate than by the brute force of computational arithmetic.

III. Theory of input signals

The Kanban cell (KC) is defined as

$$(ii_1 \wedge pp1_1) \vee qq1_1 = kk1_1 \tag{1}$$

where $\wedge$ is the logical connective of AND, and $\vee$ is the logical connective of OR.

The Kanban cell neuron (KCN) is four of these connected KCs and is defined as

$$(((ii_1 \wedge pp_1) \vee qq_1 = kk_1) \wedge ((ii_2 \wedge pp_2) \vee qq_2 = kk_2)) \vee ((ii_3 \wedge pp_3) \vee qq_3) = kk_3) = kk_4. \tag{2}$$

The utility of the KCN is that it matches the human neuron with 9-inputs as dendrites and 1- output as axon. The 9-inputs are: ii_1, pp_1, qq_1; ii_2, pp_2, qq_2; ii_3, pp_3, qq_3. The 1-output is: kk_4. This algorithm is the program for the KC. Pseudo code to process input values of ii, pp, qq into kk is presented here:

```
LET kk_lut = lut( ii, pp, qq)      ! 3-D LUT indexed by ii, pp, qq for kk
LET kk_output = ii                 ! Preset output kk to ii if test fails below
IF kk_lut ^ ii THEN LET kk_output = kk_lut      ! iff LUT result <> ii
```

The input to the KC is in the form of 3 x 2-tuples or three dibit values as effectively a 2-tuple set of {ii, pp, qq}. To produce a single dibit value as kk output, the three input values are required. Hence the expression "3-inputs to 1-output" accurately describes the KC. When KCs are chained, three inputs are required to produce each of the three outputs accepted into the next consecutive KC, for a total of 9-input signals. Hence the KCN is expressed as "9- inputs to 1-output."

The number of inputs required in KC1 to produce KCn is given by the formula below where n > 0

$$KCn = 3^{n} \quad (3)$$

It follows that KCs in parallel and chained in succession represent a permutation of exponential complexities. Each successively complex level of KCs hence has its number of KCs as a power of 9 (= 3^2), such as (3^2) ^0 = 1, (3^2)^1 = 9, (3^2)^2 = 81, … , (3^2)^n.

The number of groups of signals of {ii, pp, qq} required for levels of KCs as KCLs may be tabulated. The number of groups where three groups are processed concurrently for KCL is the result of reducing the cumulative signals by a factor of three of the cumulative number of the groups of signals. For KCL-12 or 3 ^ 13, the number of discrete signals is 1,594,323, and the cumulative number of KCNs is 531,444. For sizing in hardware implementation, each such result occupies 2-bits for a respective KCL-12 storage requirement of 1,062,882 bits.

A commonly published statistic is that on average there are seven to nine dendrites per neuron. This means two to three complete groups of signals (six to nine discrete signals) can be processed concurrently at receptor points for the dendrites along a neuron.

In the formula (ii AND pp) OR qq = kk of a KC in (1), there are 64-combinations of 2-tuples (or dibits) of the set of {00, 01, 10, 11}. When ii and kk are the same value (ii = kk), this represents the condition where input is processed to the same output result. This also means there was no change to the input after processing. Consequently, this event marks the termination of processing for that particular signal, and hence the result is final and produces no new logical result.

When ii, pp, or qq is "00", a contradiction is present to mean something is both true and false. This has the same effect as a null value because no logical information is disclosed other than that the there is a void result. Consequently the values "00" and "" are folded together into "". Hence the four-valued logical set of values as the 2-tuple of the set of {"00", "01", "10", "11"} becomes {"", "01", "10", "11"}. This is named four-valued logic with null (4VLN) composed of four-valued bit code with null (4vbcn).

All possible combinations of the values within the 2-tuple produce 64-values. A LUT for these values has an index in the inclusive interval range of [0, 63]. It is named LUT-64. This represents a sparsely filled LUT of three inputs {ii, pp, qq} to produce one output {kk} for a KC, in Table 1 below. When three KCs are combined to make a KCN, there are nine inputs to produce one output. This is named LUT-9 and is built by combining three LUTs of 64 entries each into 64 ^ 3 entries or 262,144 entries. The sparsely filled LUT-9S is indexed as (0 ... 63, 0 ... 63, 0 ... 63).

Statistics for the percentage of signals processed from KCN-18 are presented here. The signals accepted and rejected for the KCL-18 cascade are for input of 129,140,163 discrete random signals to a single result as a dibit (2-tuple) where 8,494.377 signals are accepted at about 7%, for a ratio of accepted signals to rejected signals of 1 to 13. This also indicates how the KCN overcomes the deficiencies of accepting all signals as in the Background section above.

Of interest is the relative distribution of the four-valued bit code (4vbc) for contradiction (00), true (01), false (10), and tautology (11). The logical results of the KCN favor the tautology (11) by about 34560 / 50432 = 69% over the other frequency of the other combined logical values. In the same way, true (01) and false (10) represent about (7936 + 7936) / 50432 = 31%. These statistics imply that the KCN filters about: 2-valid assertions to 9-invalid assertions; equal numbers of true- and false-assertions; and 1-true or 1-false to 2-tautologies. This implies that the KCN places an onus on rejecting invalid assertions and on finding tautologies. This also indicates how the KCN overcomes the problem of accepting all signals as in the historical neurons described in the Background section above.

IV. Models based on the Kanban cell

Fig. 3 illustrates a logical circuit for the KC that consists of multiple inputs, one output, and synchronous feedback loops. Steps 505 and 506 represent back propagating paths. Exactly how these feedback paths are stimulated is a matter of sequencing, either by decision based on data, by external clock, or by both. This KC model is ultimately synchronous, and not asynchronous.

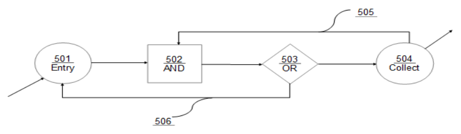

Fig. 3. Synchronous Kanban cell in feedback loops as external propagation delays.

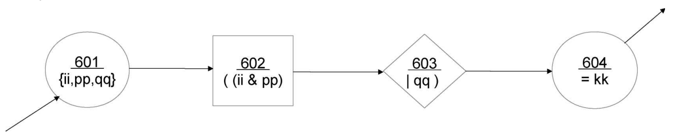

Fig. 4. Functional diagram of the Kanban cell in symbols for nodes and processes

Fig. 4 illustrates a functional process diagram of the KC using symbols for nodes and processes according to an asynchronous model. Steps 601 and 604 are the respective input and output places. Step 602 is a circuit for the logical connective AND. Step 603 is a circuit for the connective OR. This KC model includes two logical connective gates, an AND gate 602 and an OR gate 603. The signals specified for processing are the set of {ii, pp, qq} in 601 where ii AND pp is processed in 602, and that result is OR qq in 603 to produce the result renamed kk in 604. Note that the instant KC model in Fig. 4 is a simplification of the previous KC model in Fig. 3 in that the instant shows no feedback loops connecting from 603 to 601 or from 604 to 602, as is the case in the previous from 503 to 501 and from 504 to 502. There are no feedback paths present in the instant model. This means it is asynchronous or untimed by its data and with straight through data flow.

This model is made further unique by the method to terminate the input of signals to 601. If kk in 604 is determined to be equal to ii from 601, then this instance of system 60 stops processing. This feature inhibits the model from the otherwise potentially endless processing of results kk to 604. This method effectively makes the model into a self-timing KC where the input ii of 601 and the output kk of 604 determine the next state of system 60 as active or dormant. This feature means that the model is immune to external timing constraints and is wholly self-reliant for control of its asynchronous operation on input and output data.

Fig. 5 illustrates a flow diagram of the KCN (Kanban cell neuron) using symbols for nodes according to an asynchronous model of the KC. The three inputs into one output is known as 3- to-1 processing. From previous operations, the respective kk results are in 801, 802, and 803. These serve as the subsequent inputs of ii, pp, and qq into 804 as a kk result. That in turn serves as an input to the next subsequent operations if any. This model maps the paths of signals named as nodes in a network tree. The input nodes are for ii in 801, pp in 802, and qq in 803, and serve as three inputs in this model to produce one output for kk in 804. The signal values at any labeled location are automatically stored there as data which is persistent for the electrical life of the model or until changed.

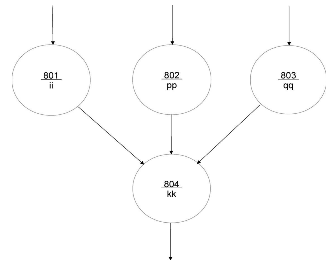

Fig. 5. Flow diagram of the Kanban cell neuron

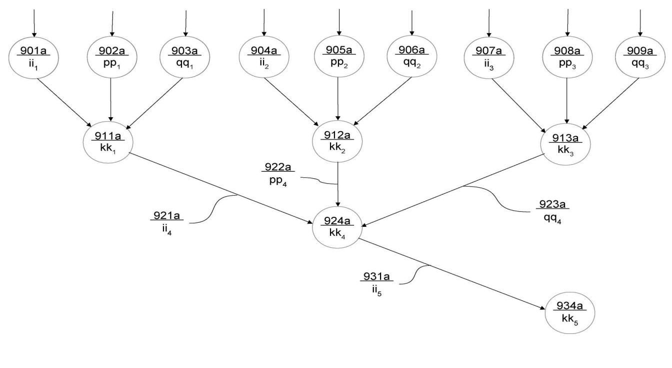

Fig. 6. Flow diagram of Kanban cell neuron network

Fig. 6 illustrates a flow diagram of the KCNN (Kanban cell neuron network) using symbols for nodes according to an asynchronous model. This network implements the KCN as nine inputs into one output, and named as 9-to-1 processing. Inputs 901a, 902a, and 903a result in 911a. Inputs 904a, 905a, and 906a result in 912a. Inputs 907a, 908a, and 909a result in 913a. Results 911a, 912a, and 913a are renamed respectively as 921a, 922a, and 923 a. The input set of {901a, ... , 909a} is a set of 9-values that produce 924a as a single output result. The result of 924a is renamed to 931a as one of three prospective inputs to the result in 934a.

This network maps a multitude of the paths of signals named as nodes in a network tree for the model of the KCN in Fig. 5. A unique feature is how consecutive outputs of kk in the set of {kk_n, kk_{n+1}, kk_{n+2}} serve as inputs of the set of {ii_{n+3}, pp_{n+3}, qq_{n+3}} to produce kk_{n+3}. This method effectively passes results from one level of the nodes in a network tree into the next level of the nodes in the cascade of the nodes in a network tree. This mechanism is inherently combined with the method from the Fig.6 above where individual KCNs become dormant when input ii is equal to output kk, to make this KCNN terminate when input signals are exhausted.

For example, if ii 1 in 901a is equal to kk_1 in 911a, then ii_4 in 921a is null, not set to kk_1 of 911a, and 911a terminates that KCN. This means that for a cascade of network paths to proceed requires no interruptions in the consecutive sequence of valid input values. Therefore if kk_1 in 911a is null, then any subsequent output value, such as kk_4 in 924a and kk_5 in 934a are null paths and no longer active. What follows from this is that kk_2 in 912a and kk_3 in 913a are also ignored by this network, and the next set of input signals, beginning with the unattached potential node 9nn, are potentially processed.

V. Implementation by look up table (LUT)

A software implementation is presented here. The values in a LUT of 64-entries may be represented as the same respective values but in three different formats such as a natural number, the character string of that natural number, or character digits representing exponential powers of arbitrary radix bases. As numeric symbols, the four valid results are in the set of {0, 1, 2, 3}, and the invalid results are specified as a null set of {-1}. As character string symbols, the four valid results are in the set of {"0", "1", "2", "3"}, and the invalid results are specified as a null set of {""}. As character strings, four valid results are the set of {"00", "01", "10", "11"} or of {00, 01, 10, 11}.

The representation of the data elements within a LUT is important because the format affects the size of the LUT and the speed at which data is manipulated. For example, to test a number for invalid result requires determining if it is a negative number, that is less than zero. To test a character string for an invalid result requires determining if it is a null character {""}, that is, not within the set of {"1", "2", "3"}. A faster method is to test the length of the character string because a null string has a length of zero. The size of the LUT is also smaller for a literal character string: 64-elements as numbers occupies 64 * 8 = 512-characters, whereas 64-elements as characters occupy 64 * 1 = 64-characters or 1/8 less.

A LUT for 9-inputs [LUT-9] consists of a 2-tuple each (2-bits) to make 9 * 2 = 18-bits. The binary number 2A18 is decimal 262,144 or those numbers in the range of the inclusive interval of [0, 262143]. This means LUT-9 is an array indexed from 0 to 262,143 that is sparsely populated with kk results as the set of {"", "1", "2", "3"} for binary "", 01, 10, 11. (The null symbol means that if it is used in a multiplication or exponential calculation, the resulting number is likely to raise exceptions.) The fill rate for the sparsely populated LUT-9 also shows that a single KCN rejects about 93% of all signals, and accepts about 7%. This reiterates how the KCN overcomes the deficiencies of accepting all signals described in the Background section above.

The design flow of the software implementation consists of three parts: build the LUT (as above), populate the top-tier of the KCL with input values, and process the subsequent lower-tier KCLs. For testing purposes, the input values are generated randomly in the range interval of [0, 2462143], that is, at the rate of 9-input signals at once. These are checked for results (valid or invalid) and used to populate the top-tier of KCL. The size of the top-tier level is determined by the maximum memory available in the software programming environment. In the case of True BASIC®, the maximum array size is determined by the maximum natural number in the IEEE-format which is (2 ^ 32) - 1. The largest radix-3 number to fit within that maximum is (3 ^ 20) - 1. However the compiler system allows two exponents less at 3 ^ 18 (3 ^ 18.5, to be exact). Hence the top-tier KCL is set as KCL-18. Subsequent lower-tier KCLs are processed by string manipulation. Consecutive blocks of 9-signal inputs are evaluated for all valid results. The valid results as single ordinal characters are multiplied to the respective exponent power of four and summed into an index value for the LUT. If the indexed LUT result is a valid result, namely not null, then the result is stored at the point in the KCL tier. This phase is KCN performance.

To access the three dimensional array faster, it may be rewritten in a one dimensional array. This is because while the three indexes of ii, pp, and qq are conceptually easier to digest, a single index of 262,144 elements in the range interval [0, 262143] requires only one index value. Incidental arrays used to perfect the LUT may be re-indexed to zero or null to reclaim memory space. This table is named LUT-241K and is implemented in software as 2-tuples or 2-bits for 262,144 * 2-bits or 524,288 bits at 8-bits per byte for 64K bytes.

A LUT with a 6-character key is presented here. The values of searchable element values for "iippqq" should also have the same key string-length, to enhance a radix search or binary search. Hence the numerical value of the index in the interval of [0, 63] is converted into a 2-character string value of the index in the interval ["00", "63"]. The subsequent indexes for the remaining two array dimensions are concatenated onto the first string index to form a 6-character key. The interval of digital search keys as [000000, 636363] contains potentially 636,364 keys. However, this is not exactly the case as some keys are impossible because the range is sparsely occupied. Excluding consecutive null values at the extrema of the range, the interval range of valid keys is [010002, 586353], but again not all key combinations therein are possible as the frequency or cycle of valid results is in runs of four separated by blocks of seven nulls. This is named LUT- 636K.

These are calculated as a radix-3 function where 3^12 or 531,441 entries at 2-bits each is 1,062,882 bits. The LUT and data structure occupy a total requirement of 1,587,470 bits. This example is directed to the use of many field programmable gate arrays (FPGAs) to build the KCNN system at a lower cost of less than $40 per target device. In addition, performance becomes a factor for faster or slower devices. On average in hardware, one access to LUT-241K takes 13.25 nanoseconds for processing at the rate of 1.8 BB KCNs per second, which is about 1,600 times faster than in software.

Fig. 7 illustrates an abstraction of Fig. 6. A unique feature of this method is that if all input and output signals are acceptable and not null, then the output result of kk_4 in 931b may be obtained directly by one access to a LUT as indexed by the nine input signals in the set of $\{ii_1, pp_1, qq_1, ii_2, pp_2, qq_2, ii_3, pp_3, qq_3\}$. In other words, this KCNN is based on nine-input signals to one-output signal as in the ratio of (3 ^ 2) to 1 or 9:1. It is the KCNN model that performs most quickly, and hence is suited for implementation in hardware over software.

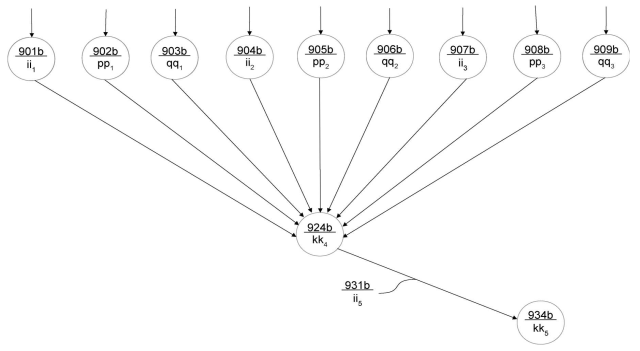

Fig. 7. Flow diagram of Kanban cell neuron network as a LUT

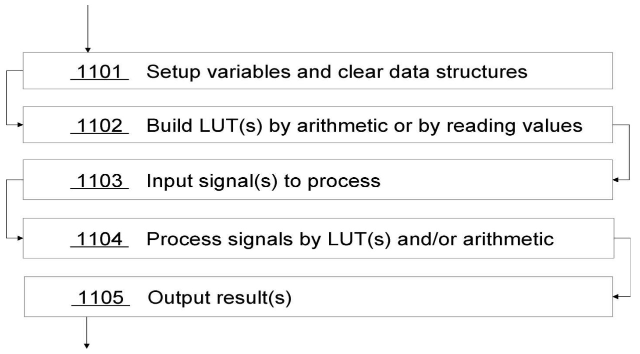
1101 Setup variables and clear data structures
1102 Build LUT(s) by arithmetic or by reading values
1103 Input signal(s) to process
1104 Process signals by LUT(s) and/or arithmetic
1105 Output result(s)

Fig. 8. Behavioral diagram of the Kanban cell neuron network in blocks of tasks

Fig. 8 illustrates a behavioral diagram of the KCNN using symbols for blocks of computer programming tasks according to an asynchronous model. Fig. 8 contains the flow chart steps to program the KCNN. In 1101, data structures and variables are initialized. In 1102, LUT(s) are built by arithmetic from primitives or reading from a constant list of supplied values. In 1103, the signals to process are input. In 1104, results from the input values are processed by LUT, by arithmetical calculation, or by both. The results from 1104 are output in 1105.

Of interest is the method to build LUTs in 1102. Results may be obtained by logical arithmetic, or LUTs may be constructed by either logical arithmetic or by reading data directly from a specification list, or a combination of both.

However the size or extent of the LUT may be limited by the number and type of datum. The MVL chosen for exposition here by example is 4VL or a 4vbc where the values are in the set of {00, 01, 10, 11} and taken to express respectively the logical states of {contradiction, true, false, tautology} and the decimal digits of {0, 1, 2, 3}.

Of further interest to this approach is the rationale behind folding the 2-tuple 00 into null "". Contradiction or 00 means "not false and not true" or in other words "true and false" as absurdum. Null on the other hand has the meaning of nothing or no value. Absurdum imparts no information about the state of true (01), false (10), or tautology (11) as "false or true". Hence the informational value of absurdum is as void to the state of falsifiability as is null. Hence the 4VL adopted here is the set of {"", 01, 10, 11}. This means that the whenever an input signal of 00 or "" is encountered, it short circuits and voids that KCN processing it.

The method for building the LUT for three inputs to one output is presented here. For the three input variables as in the set of {ii, pp, qq}, each variable of which is a 2-tuple as in the set of {"", "01", "10", "11"}, there are 2 ^ 6 or 64-combinations possible, and typically indexed as in the inclusive interval range of [0, 63]. Of these 64-combinations, there are 14-combinations that do not include the value "" as in Table 1.

Table 1. Connective assignments to 4vbc

Connective No.	((ii	AND qq)	OR pp)	= kk
090	01	01	10	11
095	01	01	11	11
106	01	10	10	10
111	01	10	11	11
122	01	11	10	11
127	01	11	11	11
149	10	01	01	01
159	10	01	11	11
165	10	10	01	11
175	10	10	11	11
181	10	11	01	11
191	10	11	11	11
213	11	01	01	01
234	11	10	10	10

The connective number is the decimal equivalent of the binary digits. For example, binary " 11 10 10 10", with most significant on the left, is decimal 234. The connective number is meaningful as an identifier in the mathematical theory of 4vbc which has 256 8-bit logical connectives as < 0, 1, ... , 254, 255>. When the 14-combinations of Table 1 are placed in the LUT of 64-entries, the frequency of distribution is sparse.

The implementation method for building the LUT for nine inputs to one output is presented here. Three instances of the table of 64-elements are manipulated to produce all possible combinations. Each combination of three inputs and one output is further checked for the exclusive condition of ii = kk for which that combination is excluded as null "".

The resulting LUT consists of 64 ^ 3 or 262,144 entries, each of which is a 2-tuple. In the source code in True BASIC®, this LUT is populated by manipulation of input arrays from minimal DATA statements. In the source code in VHDL, this LUT is enumerated bit-by-bit and occupies over 300-pages of text.

VI. Forward looking rules engine

The KCNN makes use of clusters to assign logical values to statistics in time series.

The data set of a variable is sorted in rank order then divided evenly into the number of logical values of the multi valued logic. For example with four logical values as in 4VL, the sorted list of <10, 20, 30, 40, 50, 60, 70, 80> is assigned respectively to logic values of the list of <00, 01, 10, 11> as clusters: <00> for <10, 20> ; <01> for <30, 40>; <10> for <50, 60>; and <11> for <70, 80>. Alternatively the numeric values are assigned as <1, 2, 3, 4>. Similarly, the assignment respectively of logical values to clusters could be in a reverse or different order as clusters: <11> for <10, 20> ; <10> for <30, 40>; <01> for <50, 60>; and <00> for <70, 80>. In this example, the assignment of values does not include weighting, such that all clusters do not have the same count of statistical values as cluster assignments of: <00> for <10> ; <01> for <20, 30, 40>;<10> for <50, 60, 70>; <11> for <80>.

To determine cluster assignments the sorted order of values for variables may be sorted in ascending or descending order. For example with econometric time series by trading date and/or time, cluster values may be selected for volume and the price at open, close, high, and low [6]. The justification for which type of sorting order is determined by trial and error tests of the data.

To determine parameters for the forward looking rules engine, a comprehensive tabulation of all possible combinations of variable values indicates the logical signals of interest. For example in either data set above, there are five variables each in ascending and descending order but only three such variables taken at a time, and excluding repetition of the same variable in opposite sort order, to serve as valid inputs. For n = 5 * 2 = 10 and k = 3, from (k - 1)!(n - (k - 1))! there are 80,640 possible combinations.

Some of these combinations are irrelevant because they do not make sense. For example with econometric data, if the variables for price at open and close are deemed irrelevant (except to show return on a sell or buy position), then there are three remaining variables of price at high and low and volume. Because there is no known inverse relation among these variables, they are all either in ascending or descending sort order. Hence from n = 3 and k = 3 there are 2 possible combinations depending on the sort order.

These procedures establish the forward looking rules engines for the financial examples below. It is the assignment of the time series variables to the input variables in the KCN formula and especially the subsequent evaluation of output signals that is left for the analyst.

This method is distinct from other approaches such as Gaussian distribution-based clustering and from density-based clustering. For example, the Gaussian distribution-based clustering method uses the expectation-maximization [EM] algorithm to group probabilities of sets of fixed points. The density-based clustering method uses the frequency of occurrence as the density criterion for demarking estimated borders of clusters. As such, those other approaches evaluate probabilities rather than the counted data points.

VII. Application to econometrics

The Kanban cell neuron stock trading system (KCNSTS) applies the KCN to two types of stock trading signals, the buy-side and sell-side. Fig. 9 illustrates this as the graph of closing prices over time. A financial market generally follows that graph as divided in two sides of increasing slope 1212 or buy-side and decreasing slope 1213 or sell-side. Stock traders know the buy-side as buy-low 1203 and sell-high 1204. Options traders know the sell-side as sell-high 1208 and buy-low 1209.

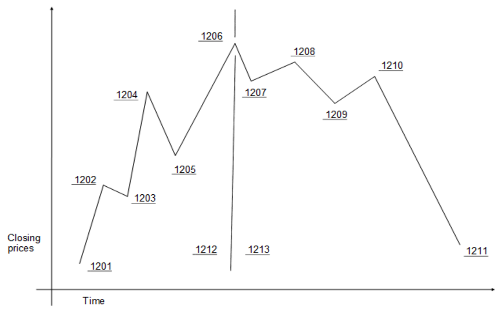

Fig. 9. Graph of financial market closing prices in time.

KCNSTS is implemented for buy-side trading where buy-side signals are in the sequential order of buy-low 1203 and sell-high 1204. KCNSTS is also adapted for sell-side trading where sell- side signals are in the sequential order of sell-high 1208 and buy-low 1209. Hence the implementation for sell-side analysis reverses the rules of the buy-side analysis. This is known as inverse logic. It is a helpful approach to validate what the rules are not, and hence by way of negation what the rules can be.

An example of how the KCN is applied to prediction of stock trading signals is in Table 2 [2]. This is a virtual book of performance or no load trades of one share. It is for 126 consecutive trading days from 2014.12.31 back to 2014.06.30. From 2014.12.31 back to 2014.12.05 it was based on daily statistics at the close. Previous to that back to 2014.06.30 it was based on weekly statistics at Friday close. The equities traded are mostly Chinese exchange traded funds (ETF) which are tied to indexes such as Standard and Poor's SPY which always rise slowly in time. These ETFs hold issues in business sectors manipulated by the Chinese government. Nonetheless after 126 trading days there is an annualized profit which beats the market.

An example in Table 3 is for an OTC stock EXLLF. In 49-trading days there are five buys, the first with asterisk returning profit at the end of the period. The annualized return is 67%.

In contrast to [9], KCNSTS does not fit a traditional model of the artificial neural network (ANN) which is non-linear, a vector space, stochastic, and hence undecidable. A potential by product of linear ANN such as KCNSTS is concept drift [7]. This tests for the three indicators of performance measures, classification models, and data properties. The three metrics are respectively accuracy of classifiers, complexity of rules, and membership clusters. Adaptive statistical windows implement the tests in an If-Then-Else decision tree. If a time series is found acceptable, then then window parameters are validated. If a times series is not acceptable, then other parameters are selected for retesting by trial and error approximation.

However, KCNSTS is not subject to concept drifts because the only criterion for success is accuracy of the prediction of the stock trading signal to produce profit, a bivalent result. For example, success is gauged by published statistics such as the daily closing price rather than instant real-time activity.

KCNSTS also applied to prediction of trading signals for commodity and note instruments in [6]. This required assigning the KCN variables of ii, pp, qq, and kk to meaningful statistics in those markets.

VIII. Contrast to other financial methods

Two methods bear remote resemblance to KCNN, but are in fact disparate.

KCNN is distinct from the approach of Bollinger bands [1] which uses means (M) in simple and exponential moving averages (MA) with a trading day or time series period (N) of typically 20-days and a scaling factor (K) of 2 for standard deviation (σ) as MA $\pm$ Kσ for upper and lower bands. Statistical studies found no evidence of consistent performance for a buy and hold strategy, but it implied that an opposite, contrarian approach could produce return in some markets (or hence significance in some time series). Bollinger bands are not linear, as is the KCN, but require stop-loss orders according to Forbes Magazine and other trading sources.

KCNN is also further distinct from the Elliot wave principle [8] which uses cycles in time series to identify periods based on the phi or Phi ratio ((1 $\pm$ $\sqrt{5}$) / 2) or golden ratio which produces Fibonacci numbers. Hence statistics in a time series may fit into cycles in an unlimited number of ways, depending on the practitioner. The Elliot wave principle is thus derived from a defective hypothesis.

References

[1] J. Bollinger, Bollinger on Bollinger Bands. McGraw Hill, 2002.

[2] http:\\ersatz.systems.com/current_book.pdf, 2015.

[3] C. James III, A reusable database engine for accounting arithmetic, Proceedings of the Third Biennial World Conference on Integrated Design & Process Technology, 2: 25-30, 1998.

[4] C. James III, Recent advances in logic tables for reusable database engines. Proceedings of the American Society of Mechanical Engineers

International, Petroleum Division, Energy Sources Technology Conference & Exhibition, ETCE99-6628, 1999.

[5] C. James III, Proof of four valued bit code (4vbc) as a group, ring, and module. World Congress and School on Universal Logic III, 2010.

[6] C. James III, Recent advances in algorithmic learning theory of the Kanban cell neuron network. IEEE Proceedings of International Joint Conference on Neural Networks. August, 2158-63, 2013.

[7] R. Klinkenberg, I. Renz, "Adaptive information filtering: learning in the presence of concept drifts", AAAI Technical Report WS-98-05, 1998.

[8] R.R. Prechter, ed., R.N. Elliott's Masterworks: The Definitive Collection. New Classics Library, 2004.

[9] N. Tudoroiu, C. Caludiu, M. Grigore, "Neural networks architectures for modeling and simulation of the economy system dynamics", Spiru Haret University, Romania, 2009.

Table 2. 126 consecutive trading days from 2014.12.31 back to 2014.06.30

126	**Trade days**				**Annual**
Symbol	**Buy date**	**Buy $**	**Sell date**	**Sell $**	**Net Rate%**
ASHR	2014/07/11	22.19	2014/07/21	22.62	0.43
CHIE vv	2014/12/17	13.11	2014/12/22	13.79	0.68
CHIM vv	2014/12/18	14.94	2014/12/29	15.17	0.23
CHIQ	2014/12/17	12.57	2014/12/26	12.99	0.42
CHIQ	2014/12/30	12.48	2014/12/31	12.62	0.14
CHLC vl	2014/08/11	25.50	2014/08/18	25.60	0.10
CHXF vl	2014/10/05	52.00	2014/10/09	52.57	0.57
CHXF vl	2014/12/10	51.01	2014/12/12	51.42	0.41
CHXX non	2014/06/30	16.88	2014/07/21	18.25	1.37
CN vv	2014/07/21	26.72	2014/08/11	28.68	1.96
CN vv	2014/10/27	28.62	2014/11/03	29.40	0.78
DSUM	2014/07/11	24.84	2014/08/11	25.04	0.20
DSUM	2014/12/19	24.33	2014/12/26	24.34	0.01
ECNS	2014/06/30	45.54	2014/07/21	45.96	0.42
EWH	2014/12/17	20.11	2014/12/26	20.74	0.63
EWS S	2014/12/18	12.76	2014/12/19	12.85	0.09
EWT T	2014/12/17	14.56	2014/12/29	15.10	0.54
EWY J	2014/12/19	55.12	2014/12/22	55.93	0.81
FCHI vv	2014/06/30	46.78	2014/07/21	48.48	1.70
FXI	2014/06/30	37.04	2014/07/11	37.90	0.86
FXI	2014/08/29	40.47	2014/09/05	42.52	2.05
FXP	2014/12/23	41.86	2014/12/26	42.28	0.42
FXP	2014/12/29	42.45	2014/12/30	43.76	1.31
JPP vv	2014/12/22	43.03	2014/12/26	43.55	0.52
PEK	2014/07/11	28.10	2014/08/11	31.14	3.04
PEK	2014/10/27	31.96	2014/11/03	33.66	1.70
PGJ	2014/12/30	27.86	2014/12/31	27.94	0.08
QQQC	2014/12/17	21.47	2014/12/19	21.64	0.17
QQQC	2014/12/22	21.51	2014/12/26	21.91	0.40
SCJ	2014/12/22	51.98	2014/12/26	52.35	0.37
YANG	2014/11/17	13.78	2014/11/19	15.95	2.17
YANG	2014/12/05	11.90	2014/12/11	13.54	1.64
YANG	2014/12/22	11.93	2014/12/23	12.34	0.41
YANG	2014/12/29	11.49	2014/12/30	12.00	0.51
YINN	2014/08/29	33.89	2014/09/05	39.12	5.23
Totals		990.78		1023.15	6.53
Symbol	**Buy date**	**Buy $**	**Sell date**	**Sell $**	**Net AnRate%**
	vl volume low	non not Nasdaq			J, K Np, Kr
	vv volume var	____ not followed			S, T Sg, Tw

Table 3. 49-trading days for EXLLF from 2015.01.13 back to 2014.11.17

EXLLF Signal	Trade date	Close	Volume	High	Low
Sell	2015/01/13	0.72	73737	0.74	0.65
* Sell	2015/01/12	0.64	65144	0.66	0.56
Sell	2015/01/09	0.57	82146	0.57	0.56
Sell	2015/01/08	0.56	66853	0.57	0.55
Sell	2015/01/07	0.54	2300	0.57	0.54
Sell	2015/01/06	0.57	36428	0.57	0.52
Sell	2015/01/05	0.54	3621	0.56	0.54
Sell	2015/01/02	0.56	28377	0.56	0.52
Sell	2014/12/31	0.57	13404	0.57	0.54
Sell	2014/12/30	0.57	3396	0.58	0.56
sell	2014/12/29	0.57	4206	0.57	0.53
Buy	2014/12/26	0.55	14200	0.56	0.54
hold sell	2014/12/24	0.55	1520	0.56	0.54
Sell	2014/12/23	0.58	13500	0.58	0.55
sell	2014/12/22	0.56	10270	0.57	0.53
sell	2014/12/19	0.55	5992	0.55	0.48
Buy	2014/12/18	0.48	7392	0.49	0.47
hold sell	2014/12/17	0.45	15000	0.47	0.45
Sell	2014/12/16	0.46	12453	0.50	0.45
hold sell	2014/12/15	0.50	2500	0.52	0.48
Hold	2014/12/12	0.49	25100	0.51	0.49
sell	2014/12/11	0.52	10020	0.54	0.52
Sell	2014/12/10	0.52	25600	0.58	0.51
Sell	2014/12/09	0.59	23700	0.59	0.51
sell	2014/12/08	0.50	44300	0.54	0.49
hold sell	2014/12/05	0.53	2500	0.53	0.53
hold sell	2014/12/04	0.55	4900	0.55	0.51
sell	2014/12/03	0.55	2961	0.55	0.50
Sell	2014/12/02	0.55	1000	0.56	0.55
sell	2014/12/01	0.56	39194	0.58	0.50
hold sell	2014/11/28	0.55	4000	0.58	0.53
sell	2014/11/26	0.60	1500	0.60	0.60
sell	2014/11/25	0.60	122060	0.62	0.58
hold sell	2014/11/24	0.58	82309	0.59	0.54
Buy	2014/11/21	0.56	4400	0.56	0.54
hold sell	2014/11/20	0.55	54448	0.55	0.53
hold sell	2014/11/19	0.57	40250	0.57	0.55
sell	2014/11/18	0.57	98700	0.60	0.57
hold sell	2014/11/17	0.56	40620	0.58	0.56
hold sell	2014/11/14	0.57	4725	0.59	0.57
Sell	2014/11/13	0.57	3300	0.57	0.57
Sell	2014/11/12	0.59	2024	0.59	0.57
Buy	2014/11/11	0.53	110716	0.55	0.52
sell	2014/11/10	0.56	2500	0.56	0.53
hold sell	2014/11/07	0.56	43900	0.56	0.55
Buy	2014/11/06	0.51	47224	0.54	0.51
hold sell	2014/11/05	0.51	114745	0.60	0.51
Buy *	2014/11/04	0.60	119067	0.69	0.57
hold sell	2014/11/03	0.72	36550	0.73	0.69